第七章　输入输出系统(I/O)

现代计算机的外部设备种类越来越多，各类设备的组成和工作原理差异很大，与计算机之间的连接和传输数据的方式也很不相同，因此，计算机的输入输出控制和输入输出子系统(简称I/O系统）就成为整个计算机系统中最具有多样性和复杂性的部分。

7.1　I/O 系 统 概 述

一、主机和外设的连接方式

主机和外设的连接方式大致可以分为三类，如图7-1所示。

图7-1a中，主机和各外设之间有单独的数据通路，称为辐射型连接。它的优点是控制简单，但结构复杂、连线多，尤其是扩充外设很麻烦。

图7-1b中，主机通过一组总线与外设连接，各外设通过集电极开路门或三态门挂接在总线上。这种连接方式结构简单且易于扩展，各外设之间也有可能通过总线直接通信。其缺点是所有的外设都通过同一组总线分时工作，由于信息吞吐量有限，将影响信息交换速度。这种结构广泛用于微、小型计算机中。

图7-1c的连接方式可看成上述两种方式的结合型。主机通过“通道”管理输入／输出操作，主机与通道间采用辐射型连接，通道和外设之间则通过总线相连。

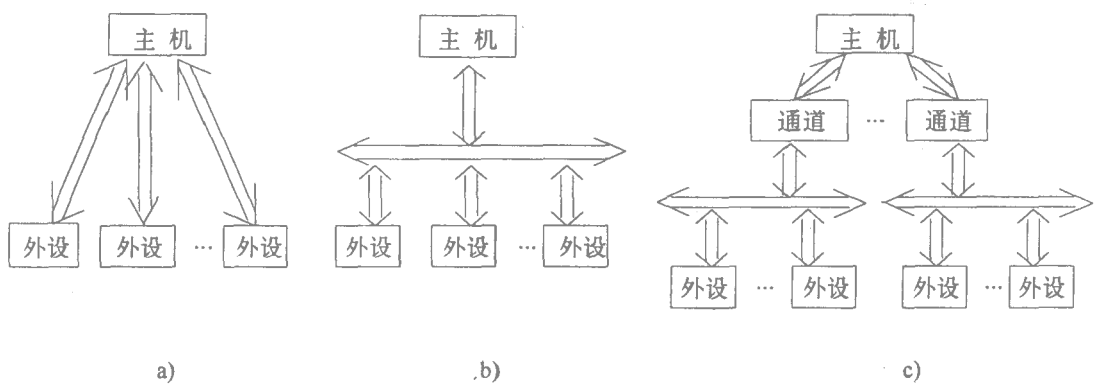


图7-1　主机与外设的连接方式

二、接口的概念

不论何种设备以何种方式与主机相连，其功能都是要实现计算机与外部世界的联系，其关键是两者之间的信息交换。然而，外部设备是多种多样的，常用的就有键盘、鼠标、触摸屏、CRT显示器、磁盘机、打印机、绘图仪、模／数(A/D)和数／模(D/A)转换器、扫描仪以及调制解调器(MODEM)等等，这些设备的结构与工作原理各不相同，涉及光、电、磁及机械传动等等。它们在工作速度、使用的数据格式等方面与主机不同，一般不能直接与主机相连。主机与外设的连接通常都要通过接口(Interface)实现，这便是I/O接口，如图7-2所示。I/O接口是主机和外设之间的交接界面，是CPU与外界各种检测、控制对象联络的纽带和桥梁。在下面的讨论中，不严格区分主机与CPU的概念，输入或输出是以CPU为中心确定的。

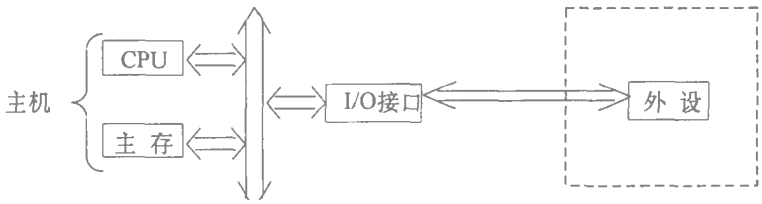


图7-2 I/O 系统组成

通常，I/O接口应解决如下几个问题。

(1) 协调CPU与外部设备之间在数据传送速度上的差异。为此，接口部件的输出端口一般包含数据锁存器，而输入端口一般为三态缓冲器，如图7-3所示。CPU 向外设输出数据时，由于数据在总线上停留的时间很短，必须采用数据锁存技术，利用I/O译码信号及I/O写控制信号 IOW 作为触发脉冲，把数据线上的状态及时地锁存到锁存器中， CPU写完此数据后便可以继续进行其他操作，慢速外设在需要时再把锁存器中的数据取走。外设向CPU输入数据时，外设把已准备好的数据送到三态缓冲器的输入端，再通知CPU读取。事实上，采用三态门也是多个外设分时使用数据总线的需要。

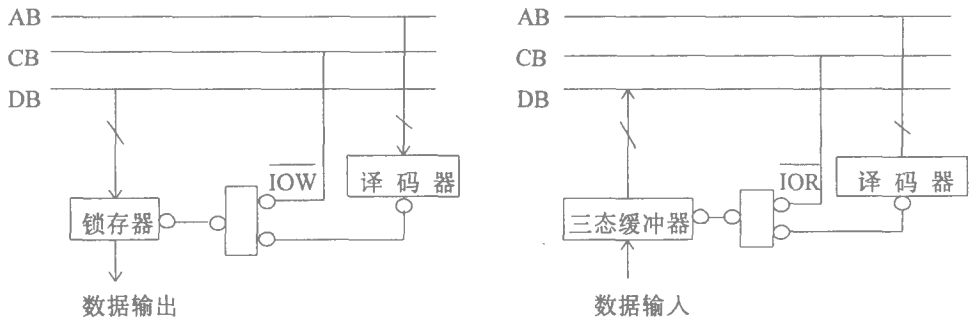


图7-3 数据缓冲原理

(2) CPU与外设是异步工作的，它们有各自独立的定时系统，在时间上是不同步的。对于一个要向计算机输入数据的输入设备（例如A/D转换器），CPU不能准确知道它什么时候已把要输入的数据准备好了，这个输入设备也不知道什么时候CPU已取走了该数据而应该开始准备下一个输入数据。对于一个接收CPU输出数据的输出设备(如打印机)，CPU不能准确知道什么时候这个输出设备已经做好了接收数据的准备，反之，输出设备也不能准确知道什么时候CPU输出了新的数据。因此，为了可靠而有效地传输数据，CPU与外设之间需要互相提供联络信息。联络信息可以理解为状态，如向接收方提供“数据已准备好”的状态信息，请对方取走数据。联络信息也可以理解为命令，如“启动一次A/D转换”。实际上，理解为状态或命令并没有严格的区别。如何向对方提供联络信息、以何种方式响应和处理对方提供的联络信息，是I/O接口设计要解决的最基本问题之一。此外，计算机系统中往往有多个外设，这就有可能出现两个以上外设同时要求与CPU交换信息的情况，此时要根据某种策略决定首先为哪个外设服务。这是I/O接口设计要解决的另一个基本问题。

在微型计算机系统中，通常有三种基本的输入输出方式，即程序查询方式、程序中断方式和DMA传送方式。不同的输入输出方式在解决上述两个基本问题时各具特点，这也是本章要重点讨论的内容。

(3) 提供I/O设备寻址信息，使CPU能从众多的外设中选择其中之一与之进行数据交换。

(4) 进行数据格式的变换。模拟接口把外设送来的模拟信号变换成CPU能接受的数字信号(A/D接口)，CPU向模拟设备输出信息时也必须进行变换，只是变换的方向相反而已(D/A接口)串行和并行数据格式的变换也是常见的(如串行通信接口)在LED或CRT显示接口中要进行字符码到字形码的代码变换。进行信号电平变换有时也是需要的。

接口的输入输出操作必须要有软件的支持，因此，接口还包括软件。用微计算机组成一个应用系统的关键技术是接口技术，而接口的研制与设计既有硬件电路设计的内容，也要为接口编写相应的驱动程序。随着各种大规模可编程接口芯片的出现与普及，I/O接口软件的地位愈加重要。

三、接口的分类

接口通常是以外部设备输入输出的信息特征进行分类。按照信息的形式分为数字接口和模拟接口，本章7.6节将专门讨论模拟接口。对数字接口，可按数据传送宽度的不同分为串行接口和并行接口。这里所说的数据传送宽度指外设和接口之间的传送宽度，而在主机(CPU)和接口一侧，数据总是并行传送的。并行接口中外设和接口之间一次同时传送一个字节（字）的所有位，传送速度快，但传输线的数目随着一个字的位数增多而增加。串行接口中外设和接口之间的数据是逐位串行传送的，第八章将专门讨论这种接口。此外，根据接口的使用范围，可分为通用接口和专用接口，通用接口是指功能上通用的一类接口，如并行I/O接口、串行I/O接口等；专用接口通常是为某一特定外部设备专用的，如软盘控制器、CRT控制器、键盘接口控制器等。

由于接口电路的设计比较复杂，同时考虑到同一类接口有许多要求是相似的，所以在微处理机一开始出现，就有大规模集成接口电路芯片与之配套。绝大部分接口芯片都可以由CPU写入适当的控制字来改变其工作方式或工作参数，这称为可编程。这一特性使接口

芯片的适用范围得到扩充。使用这类芯片时，首先要用程序设定接口芯片的工作方式和参数，这就是所谓的初始化编程。不同功能的接口芯片，其结构虽各有不同，但都是由寄存器和控制逻辑两大部分组成的，如图7-4所示。

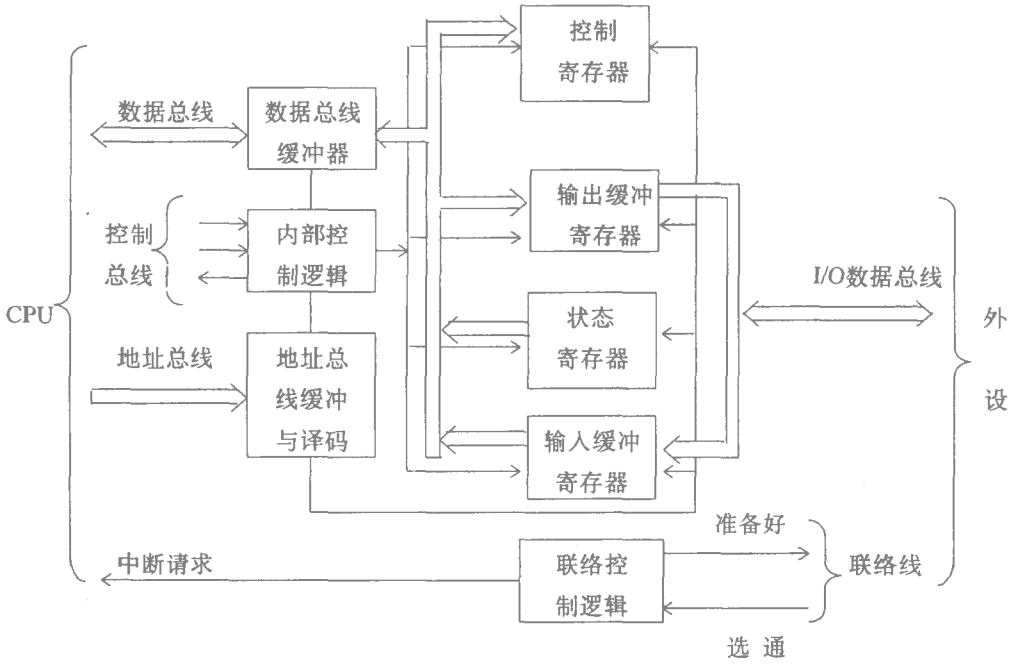


图 7-4　接口芯片基本结构框图

(1) 数据缓冲寄存器　包括输入缓冲寄存器和输出缓冲寄存器，前者应具有三态功能，当它没有被选中时处于高阻态，后者则要有数据锁存能力，这是I/O接口的基本结构特点之一。

(2) 控制寄存器　用来存放CPU发来的控制命令和其他信息，以确定接口的工作方式和功能，这是可编程接口芯片所必需的。

(3) 状态寄存器　用于保存外设当前的各种状态信息。当CPU以程序查询方式同外设交换信息时，该寄存器是必不可少的。

以上三种寄存器构成接口芯片的核心部分。

(4) 端口地址译码器　用来正确选择接口芯片内部各端口寄存器。

(5) 对外联络控制逻辑　主要有面向CPU一侧的中断请求和响应信号，面向外设一侧的准备就绪和选通等控制与应答信号。此外，还有数据总线和地址总线缓冲器以及内部控制逻辑等，它们分别用于接口芯片与CPU总线(DB、AB、CB)的连接。

并非所有接口芯片都具备上述全部内容。

7.2 I/O 端口寻址与地址译码方法

一、I/O端口寻址

所谓端口(Port)是指I/O接口(接口芯片或接口卡)中供CPU直接存取访问的寄存器或某些特定的硬件电路。一个I/O接口总要包括若干个端口，除常见的数据端口、控制命令端口和状态端口外，还有特殊用途的端口如方式控制端口、操作结果端口和地址索引端口等等。端口的多少及其相应的功能完全取决于一个I/O接口所关联的I/O设备，这将在以后各章节中详尽叙述。但应该指出，一个端口可设定为只读(一般为状态或结果信息)、读写(一般为数据或命令信息)、只写(读出无意义，如方式控制)等属性，这也是在设计I/O接口功能时决定的。

既然端口可被CPU访问，这就有一个寻址的问题。在当今流行的各类微型计算机中，对端口有两种编址方法——独立的I/O编址和存储器映象编址。

1.独立的I/O编址方式

这种方式有一个与存储器空间完全独立的I/O地址空间，I/O端口与存储器各自编址。对同一地址编号，CPU是寻址I/O端口还是寻址存储器呢？为此，要作以下约定：

(1) CPU设置两组读／写控制信号——存储器读／写信号MEMR / MEMW和I/O读写信号IOR / IOW，如图7-5所示。

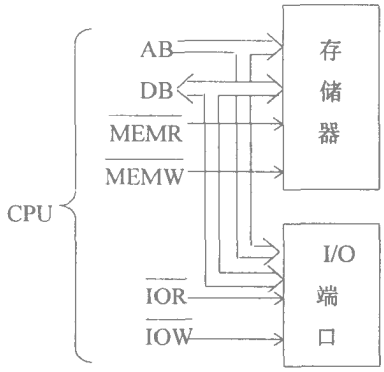


图7-5 CPU设置的两组读／写控制信号

(2) 在指令系统中设置专用I/O指令——输入指令IN和输出指令OUT。

I/O独立编址方式具有以下优点：

1) I/O端口不占用存储器空间。

2) 程序设计时，容易把I/O指令与存储器访问指令区分开。

3) I/O指令通常只需2个字节，寻址速度较快。

PC系列计算机由1024个I/O端口地址组成一个独立的I/O空间(000H ~ 3FFH)。其中低端512个端口地址(000H ~ 1FFH)供系统板电路使用，高端512个端口地址(200H ~ 3FFH)供扩

展插槽使用，一般用户可以使用300H～31FH地址。所有连接在系统地址总线上的I/O设备使用低10位地址线(A_9～A_0)，用户在设计I/O接口卡时，一定要使端口地址译码电路的A_9=1，以免发生I/O端口地址重叠和冲突。I/O设备的读／写控制使用IOR和IOW信号(由总线控制部件或DMA控制器提供)。

8086/8088 CPU的I/O指令仅支持I/O设备与累加器(AL/AX)之间的数据交换,其中AX对应地址相邻的两个8位I/O地址端口。

输入指令：
```
IN    AL/AX, PORT      ;AL← (PORT) /AX← (PORT+1: PORT)
```
或
```
MOV   DX, PORT
IN    AL/AX, DX
```
输出指令：
```
OUT   PORT , AL/AX
```
或
```
MOV   DX, PORT
OUT   DX, AL/AX        ; (PORT) ←AL/ (PORT+1: PORT)  ←AX
```

I/O端口寻址可以是直接寻址或寄存器间接寻址。前者在指令中直接写上8位的端口地址PORT，这种方法只能寻址0～255(0FFH)；后者使用16位寄存器DX放端口地址，寻址范围可达64KB。

80386/486 CPU允许I/O设备与内存之间直接进行数据传输。

输入指令：
```
MOV     DX, PORT
LEA     DI, BUFFER_IN
INSB/INSW
```
输出指令：
```
MOV     DX, PORT
LEA     SI, BUFFER_OUT
OUTSB/OUTSW
```

如果在INS和OUTS指令前使用重复前缀REP，可实现I/O设备与内存RAM之间的成批数据交换。PC/AT计算机系统的硬盘I/O控制程序(INT 13H）就是用这种方式实现硬盘扇区的读写操作。

2.存储器映象方式

存储器映象方式将I/O端口和存储单元同等看待，一起编址，即I/O端口地址空间是存储器地址空间的一部分，CPU用对存储单元的读／写操作来完成I/O端口的输入／输出。CPU访问存储单元和I/O端口(实质都是存储单元)统一使用存储器读／写信号MEMR / MEMW，两者的区分是通过地址译码实现的。Motorola公司生产的MC6800/68000系列、6502系列CPU以及INTEL公司的MCS-51系列单片计算机就是采用这种I/O编址方式。图7-6是存储器映象方式I/O编址的示意图。A_{15}=0时，A_{14}～A_0用于寻址存储器单元，A_{15}=1时由A_{14}～A_0寻址I/O端口，显然，整个存储空间被分为两半。

存储器映象方式编址的主要优点是：①对I/O端口与对存储器的操作完全相同，由于存储器操作指令较丰富，大大增强了系统的I/O功能，使得访问外设I/O端口的操作更方便灵活；②扩大了I/O端口的寻址范围(I/O端口数目只受总存储器容量的限制）。其主要缺点是：占用了存储器的部分地址空间，使可用的内存空间减少了。

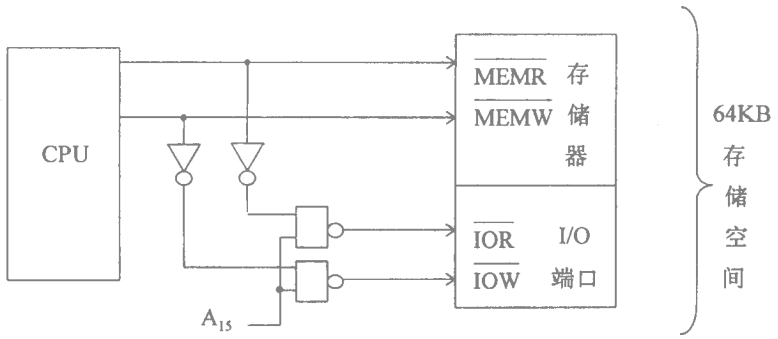


图7-6 存储器统一的I/O编址连接

二、I/O端口的地址译码方法

CPU要对I/O端口进行读／写操作，首先要确定与自己交换信息的端口地址。如何实现由CPU发来的地址码识别和确认I/O端口，这就是所谓I/O地址译码问题。译码方法灵活多样，一般由地址和控制信号(IOR/IOW，AEN等)的不同组合实现。译码电路采用的元器件通常有门电路和译码器，也有采用GAL或PAL器件进行译码的。

当仅需一个口地址时，采用门电路构成译码电路很简便。图7-7所示的电路，能译出进行读／写操作的端口地址2C7H(低电平有效)。图中用AEN参加译码，只有当AEN=0时，即不是DMA操作时译码才有效，从而避免在DMA周期由DMA控制器对该I/O口地址选中的外部设备进行读／写操作。

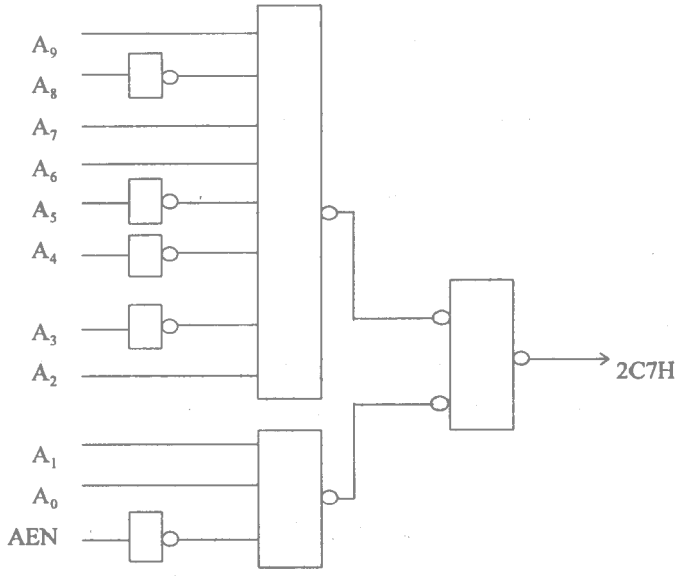


图7-7 由门电路构成的译码电路

微计算机系统通常使用多个接口芯片，每个芯片都有片选信号端 CS，且内部含有多个寄存器。这时I/O端口译码应包括两个方面：

(1) 外部译码，产生片选信号。一般采用译码器芯片实现。图 7-8是 PC机系统板上接口芯片的端口地址译码电路，图中由地址线 $A_9 \sim A_5$ 译码，产生 DMACS (8237)、INTRCS (8259)、T/C CS (8253)、PPICS (8255A) 等片选信号，地址总线的低 5位 $A_4 \sim A_0$ 在芯片内部进行局部译码以选择片内寄存器。很明显， 8237A芯片的端口地址范围是 000 ~ 01FH， 8259A芯片的端口地址范围是020H ~ 03FH 等等。这些端口地址我们以后会经常接触到。

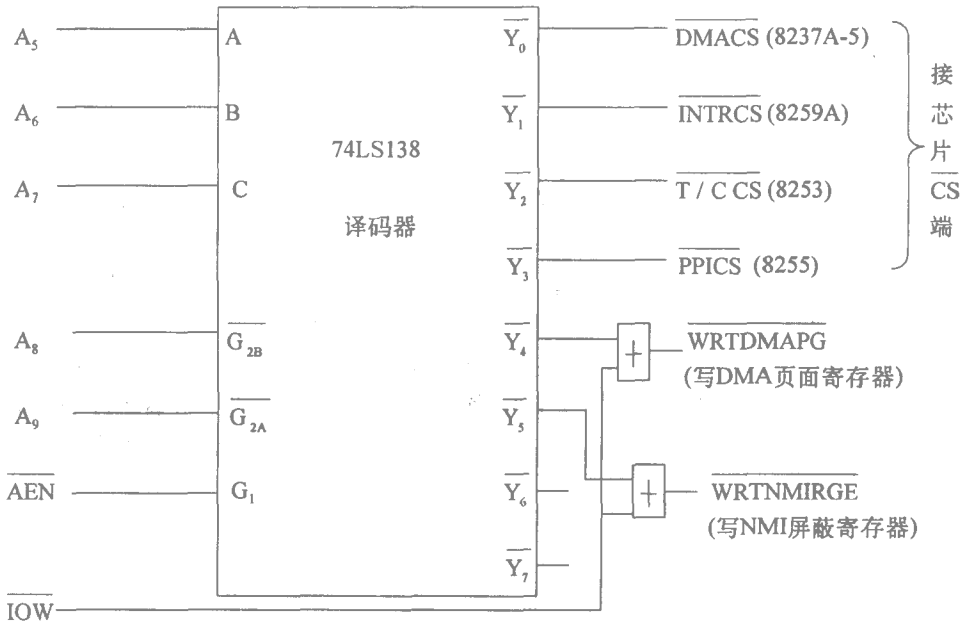


图 7-8 系统板上接口芯片的片选译码电路

(2) 片选信号 CS 确定了一个接口芯片的 I/O端口地址范围，芯片内各个寄存器的寻址是由芯片内部的逻辑电路完成的。

常用的内部译码有以下几种形式：

1.一般全译码电路

通常，一个芯片内各寄存器的端口地址采用连续编址的方式，因此用若干条最低地址线就可以区分它们。比如，对一个包含四个端口地址的接口芯片，使用 A_1A_0 两条最低位地址线译码即可。

2.计数译码电路

对接口芯片内各寄存器的寻址规定严格的先后次序，每执行一次I/O写入操作都同时对一个计数器计数，然后对计数器的计数值译码来保证正确的写入顺序。例如：假定芯片内共有四个寄存器 $R_0 \sim R_3$，设置一个两位的二进制计数器，系统复位后计数器的状态为00，经译码后使 R_0 的控制门打开，如图7-9所示。当 CPU对选中的接口芯片(片选 $\overline{CS}$ 信号有效)

写入数据时，首先写入到R_0中，同时使计数器加1。下一次CPU对同一地址进行的写入操作，则使数据进入R_1，其他寄存器依次类推。

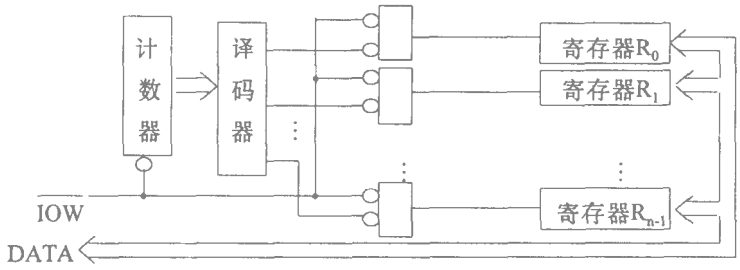


图7-9　计数译码器

这种方式工作的接口芯片只占用一个I/O地址，而不管它内部有几个寄存器。在对这种接口芯片进行初始化编程时，一定要注意写入的先后顺序问题。

3.命令字中携带地址信息

在一些较复杂的接口电路中，可能有很多种工作方式供选择，因此CPU要把各种工作方式的代码放到不同的寄存器中。为了节省地址资源，有的接口芯片利用命令字中的一些特定位携带地址信息，图7-10给出一个可能的例子。CPU写入的命令字中，规定八位数据中的第4、5位为地址信息，用以指定该命令字写入的寄存器；片内电路从命令字中引出第4、5位，作为译码器的输入信号，而译码输出作为片内寄存器的选择控制信号。

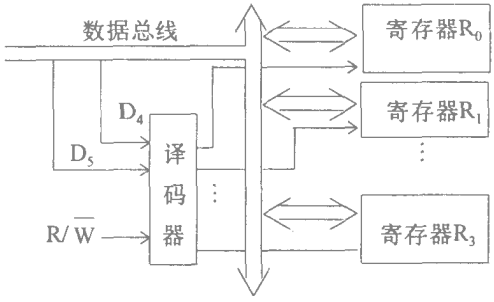


图7-10　命令字包含地址信息的片内寻址

在对这类接口芯片进行初始化编程时，写入的控制字的某些位要被指定为0或1。

4.用专用数据字节指定寄存器地址

前面的三种方法在寄存器数量比较少的情况下是适用的，如果寄存器的数量比较多，就不太合适了。于是提出用一个数据字节来指定内部寄存器地址的方法，图7-11是这种方法的原理图。图中设置一个触发器FF，它的初始态使地址寄存器打开，可以接收数据线上的数据；当CPU第一次往接口芯片写数据时，一方面将这个数据送入地址寄存器，另一方面使触发器FF的状态翻转，让寄存器$R_0 \sim R_{n-1}$处于允许接收数据的状态；CPU第二次对接口芯片进行写操作时，数据写入到由地址寄存器内容经译码后指定的寄存器中，写入结束时，触发器状态又一次翻转。

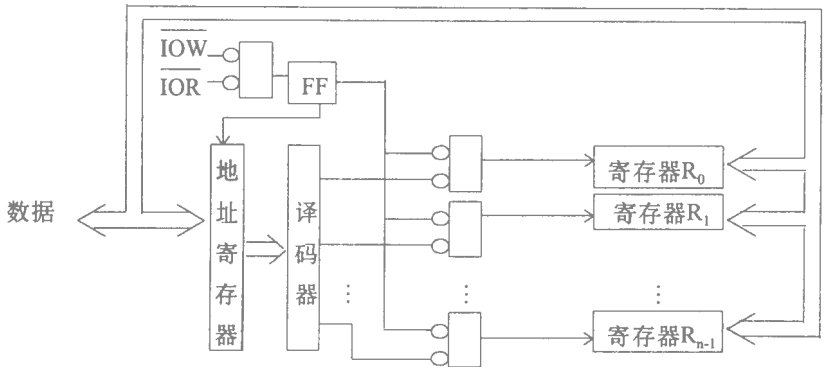


图7-11　用数据字节指定地址的片内寻址

以这种方式工作的接口芯片，虽然片内寄存器数量很多，但只需占用一个I/O口地址。它以先写地址后写数据的方式有效地节省了地址资源。当然也可以让地址寄存器单独占用一个地址，全部寄存器$R_0 \sim R_{n-1}$共用一个地址，如VGA/TVGA视频系统的寄存器组就采用这种方法寻址。

本书后面各章在介绍微型计算机系统设计方法时，要用到各种接口芯片，读者可以从对这些接口芯片的操作，特别是对它的初始化编程中体会这4种内部译码方式的特点及使用方法。有的接口芯片的内部译码逻辑还可能同时采用几种方法，以达到既节省地址资源又便于使用的目的。

7.3　程序查询方式及其接口

一、程序查询方式

程序查询方式是通过利用程序检查外设的状态来协调CPU与外设之间的速度差异，实现CPU与外设之间可靠传送数据的一种方法。其工作过程可用图7-12表示，程序查询的核心是图中虚线框出部分。一个数据传送过程由三个环节组成：

①CPU从接口读取状态字；

②CPU检测状态字的对应位是否满足“就绪”条件，如果不满足，则返回第一步继续读状态字，以等待条件的满足；

③如果状态字表明外设已处于“就绪”状态，则传送数据。

图7-13给出了程序查询方式接口的示意图。

查询式接口至少要有两个寄存器：一个是数据缓冲寄存器，用来存放与CPU进行传送的数据信息；另一个是供CPU查询的I/O设备状态寄存器，这个寄存器可以由多个标志位组成，但最重要的是外设准备就绪标志（输人和输出设备的准备就绪位可以不是同一位），当CPU得到该标志状态后就进行判断，以决定是继续等待还是进行一次I/O传送操作。在一些简单的接口电路中，可能只设置状态标志触发器来反映外设是否已准备就绪。

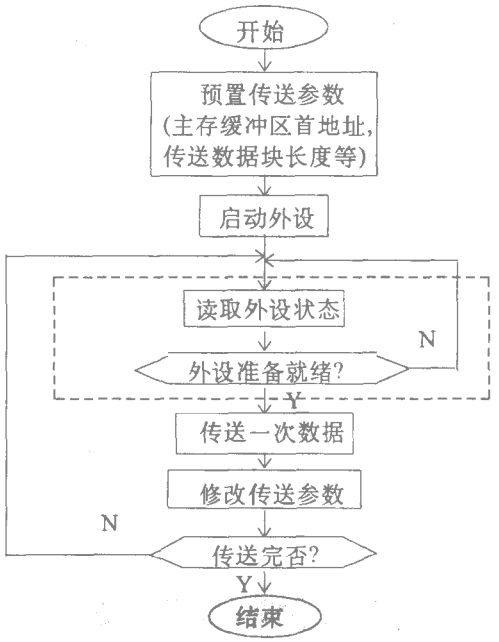


图7-12　程序查询方式的工作过程

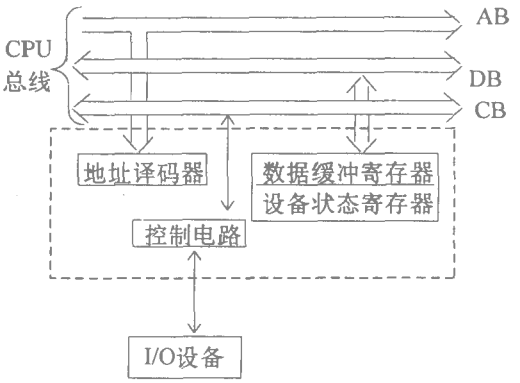


图7-13　程序查询方式接口示意图

二、查询式输入接口

图7-14为查询式输入接口电路，图中状态触发器提供准备就绪信号READY，它对应于设备状态寄存器的D_7位。在输入设备准备好数据时，发出一个选通信号STB，把数据送入锁存器，同时将状态触发器置“1”，以标识该接口电路已有数据（即准备就绪）。CPU要从外设输入数据时，先读入状态信息，如READY=1，则打开三态缓冲器(1)读取锁存器的内容，同时把状态触发器清“0”，表示上一次的数据已经取走，可以从外设接收下一个数据；如READY=0，则继续查询状态信息直至READY=1。

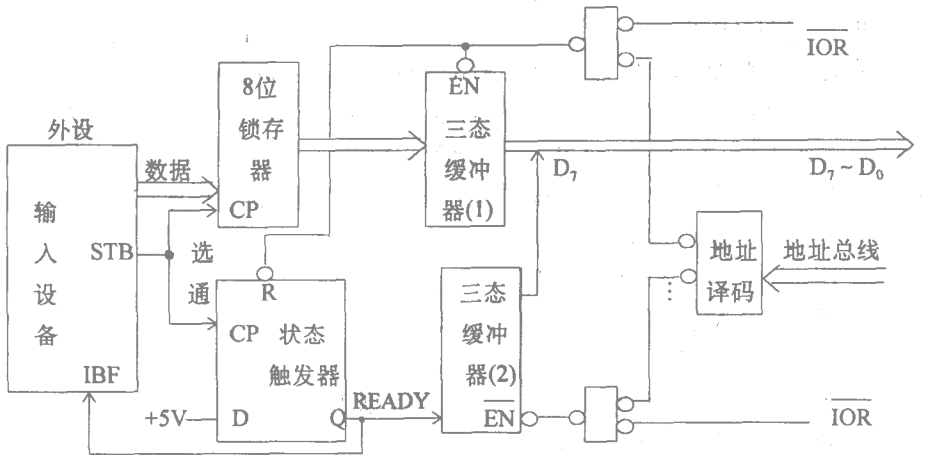
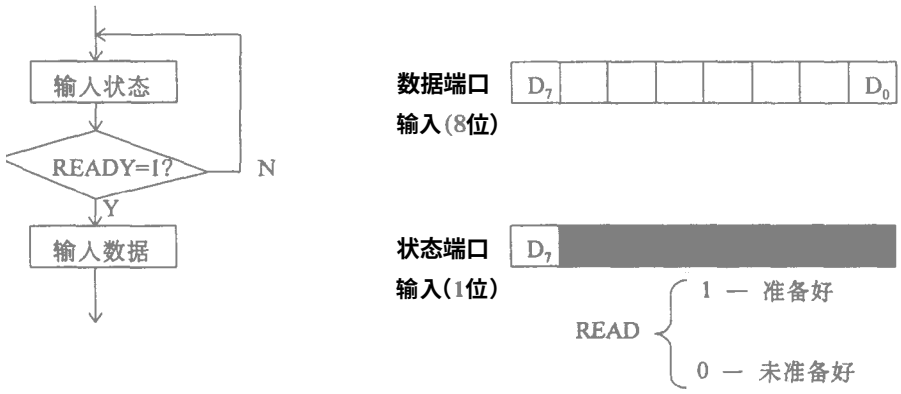


图7-14　查询式输入接口电路

查询方式输入的程序流程和数据、状态信息如图7-15所示。查询部分的源程序如下。

```
        ⋮
        ⋮
POLL:   IN      AL, STATUS_PORT     ;读入状态信息
        TEST    AL, 80H             ;检查状态位
        JZ      POLL                ;未准备就绪则继续查询
        IN      AL, DATA_POTR       ;已就绪则读入数据
        ⋮
```



a)询查方式输入流程图　　b)输入的数据及状态信息

图7-15　查询方式输入流程图和输入的数据、状态信息

三、查询式输出接口

图7-16为查询式输出接口电路，图中状态触发器提供BUSY信号，它对应于设备状态寄存器的D_0位。执行输出操作时，CPU首先读状态触发器，如BUSY=1，表示接口的输出锁存器是满的，CPU只能继续查询状态信息直至BUSY=0；如BUSY=0，表示接口的输出锁存器是空的，CPU可以向外设发送数据，此时，CPU执行输出指令将数据总线上的数据送入输出锁存器，并将状态触发器置“1”(BUSY=1)，它告诉外设现在接口中已有数据可供提取。当外设从接口中取走数据后，要回送一个应答信号ACK，它使状态触发器清0(BUSY=0)表示外设“不忙”。CPU了解到BUSY=0时，又开始输出下一个数据。

假定STORE是存放输出数据的工作单元，输出查询部分的源程序如下：

```
        ⋮
POLL:   IN      AL, STATUS_PORT     ;读入状态
        TEST    AL, 01H             ;检查忙标志位
        JNE     POLL                ;忙则等待
        MOV     AL, STORE           ;不忙则输出数据
        OUT     DATA_PORT, AL
        ⋮
```

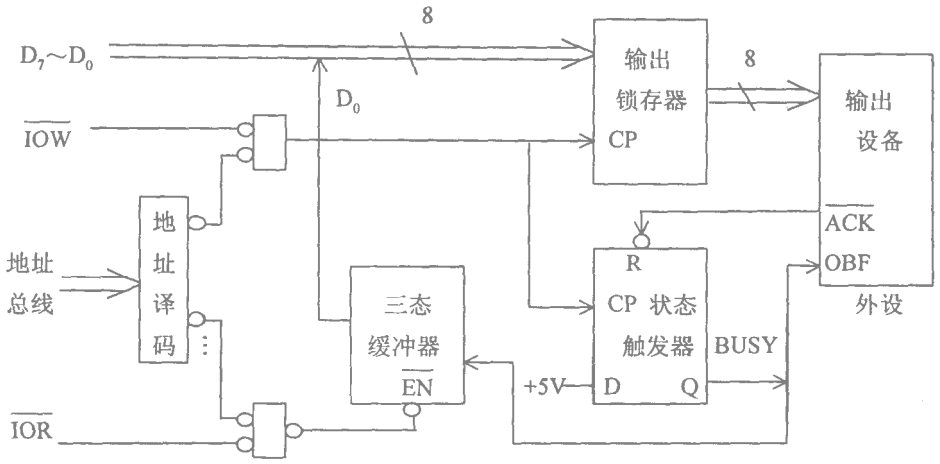


图 7-16　查询式输出接口

若有多个外设均是以查询方式与 CPU 交换数据，CPU 要逐个查询各外设状态，发现有准备就绪的外设就与之交换数据。

在某些应用场合，输入时可认为输入设备的数据始终是准备好的，如开关输入；输出时认为输出设备一定是“空闲”的，如 LED 显示器、数／模转换器 DAC 等。这时，不必查询外设的状态就可立即进行数据传送，这就是所谓无条件传送方式。无条件传送的输入／输出接口如图 7-17 所示。

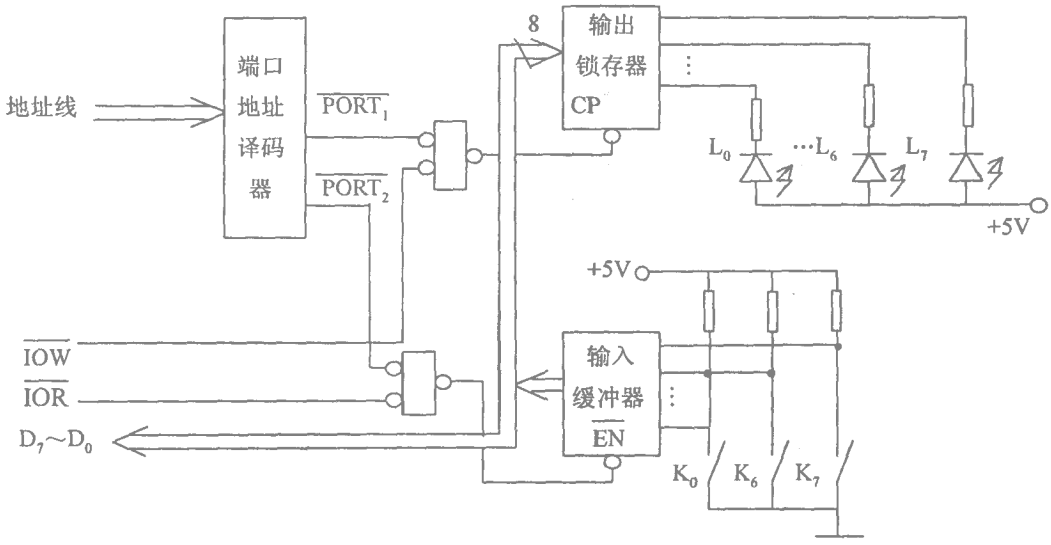


图 7-17　无条件传送方式接口电路

例如下面的程序段可使图 7-17 中的 8 个 LED 依次发亮，调用 DELAY 延时子程序是为了保证有足够的亮度。

```
START:  XOR AL, AL          清进位标志
        MOV AL, 01          先点亮与数据线D0位对应的LED
```

```
LOOP:   OUT     PORT, AL
        CALL    DELAY           调延时子程序
        RCL     AL, 1           ;AL内容左移一位，指向下一个LED
        JNC     LOOP            未循环完,继续
        JMP     START           开始下一轮循环
DELAY   PROC    NEAR            延时10ms子程序
        MOV     CX, 2801
WAIT:   LOOP    WAIT
        RET
DELAY   ENDP
```

7.4 中 断 系 统

一、中断的基本概念

1.中断的提出

在程序查询方式中，CPU要不断地询问外设状态直到外设准备好。由于大部分外设速度是很低的，这样就浪费了大量的CPU时间。此外，在程序查询方式中，CPU处于主动地位，外设处在被动地位，由于外设要求CPU服务都是随机的，有些外设的要求可能是很急迫的。因此，用查询方式很难使系统中每一个外设都工作在效率比较高的状态。

为了克服查询方式的缺点，50年代中期，中断技术被引入计算机系统。中断控制的基本思想是CPU不必查询等待外设状态,而是和外设并行工作。当外设准备好时（指输入时，外设输入数据已准备就绪；在输出时，外设的数据寄存器已空），向CPU发出一个中断申请，请求CPU为自己服务。CPU在符合响应条件时响应该中断请求，暂停原执行的程序而转到为该设备服务的中断服务子程序，待处理完之后，又返回断点继续执行原来的程序。图7-18给出了中断控制I/O方式示意图。

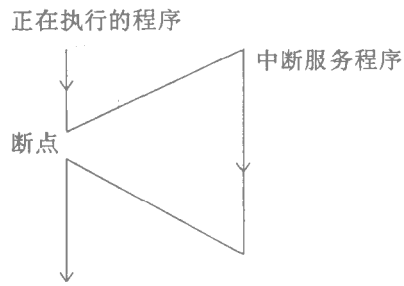


图7-18 中断控制I/O方式

采用中断技术后，一方面实现了CPU和外设并行工作，大大提高了CPU的效率；同时又能适应随机发生的情况，增强了计算机系统的实时性，具备对应急事件的处理能力，如电源断电、溢出处理等，提高了计算机系统的可靠性。可以这样说：中断技术为CPU带来了自由，也让外设赢得了主动。图7-19给出了CPU和外设并行工作的示意图。

现在，中断技术在分时操作、实时处理、人机联系、多道程序、程序的监视和跟踪以及多机系统中得到广泛的应用。

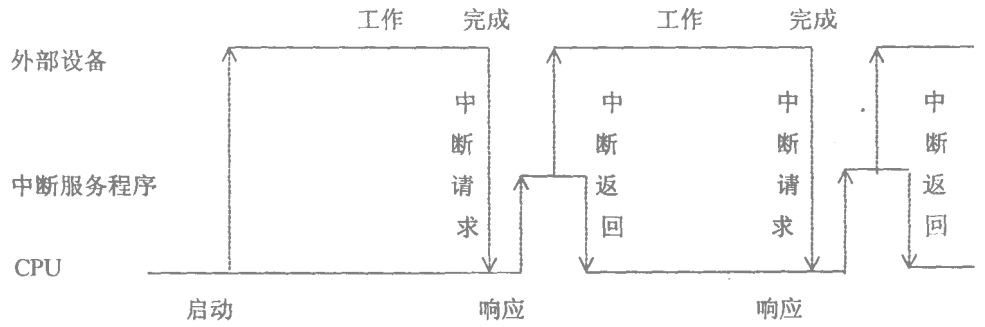


图7-19 CPU和外设并行工作的示意图

2.中断的定义

中断是指正在执行的程序由于计算机系统内、外的原因而中止，转而执行另外的处理程序，待处理完后又返回原程序的过程，通常称之为程序中断。

表面上看来，程序中断类似于调用子程序的过程，这里正在执行的程序相当于主程序，中断服务程序相当于子程序。它们之间究竟有哪些区别呢？

中断大体上分成两类—— 强迫中断和自愿中断。

(1) 强迫中断指由计算机硬件设备故障（如内存奇偶校验错、协处理器异常）、程序错误而引起的中断以及来自外部设备请求引起的中断，这类中断是随机发生的，它与具有确定性的子程序调用有本质的区别。

(2) 自愿中断指事先在程序中安排的陷阱指令或软中断指令引起的中断，例如INTEL 8086/80386 CPU的INT n 指令，其中n称软中断号。软中断指令放在程序中哪个位置、何时执行，程序员是可以预先知道的，所以自愿中断失去了随机性。事实上， INT n 指令与CALL指令作用相近，主要差别是INT n 指令在执行时要先将标志寄存器FLAG内容推入堆栈保存，因为FLAG记录了中断发生时程序指令运行的结果特征，当CPU处理完中断请求返回原程序时要保证原程序工作的连续性和正确性，因此，响应中断时保存FLAG 的内容是必要的，CALL 指令则没有这一步操作。INT n 指令可用 CALL等指令来模拟：

```
PUSHF                模拟INT n
CALL    ADDRESS
```

ADDRESS 是n号中断服务程序的入口地址。注意程序结束返回时要用中断返回指令IRET而不是RET，因为IRET将弹出断点的IP、CS和FLAG值。

当然，有部分软中断指令（如运算溢出中断指令INTO 等)是在程序执行过程中发生异常条件时才激活中断(如溢出标志位OF=1)，异常条件是否一定发生，程序员并不能事先知道，从这个意义上说，自愿中断也是具有随机性的。

一个完整的程序中断处理过程包括：

- 中断请求
- 中断判优
- 中断响应
- 中断处理
- 中断返回

二、中断请求

1.中断源和中断请求

中断源是指任何引起计算机中断的事件，中断源的数量可以很多。由于每个中断源向CPU发出中断请求是随机的，对这些中断请求，要用触发器加以锁存，通常称为中断请求触发器。当一个中断源提出中断请求时，其相应的中断请求触发器置“1”，多个中断请求触发器构成一个中断请求寄存器，其中每一位对应一种中断源。

中断请求触发器可以在CPU芯片内部或外部，这与CPU中断请求信号输入线的特性有关。

(1) 中断请求信号为边沿触发的CPU，例如MCS-51系列单片计算机的中断请求输入线$\overline{INT_1}$和INT_0可设置为下降沿有效，由于CPU在接到中断请求信号后不一定立即响应，因此，CPU内部设有中断请求触发器来记录这些中断请求。CPU响应该中断请求后会自动地清除中断请求触发器的相应位以撤消此次中断请求。显然，多个中断源不能共用一条边沿触发的中断请求输入线。

(2) 中断请求信号是电平有效的CPU，用户要为中断源设置中断请求触发器，并在中断服务程序结束前用指令清除该中断请求触发器以免出现对同一中断请求多次响应的情况。图7-20给出了一个中断方式输入的接口电路。

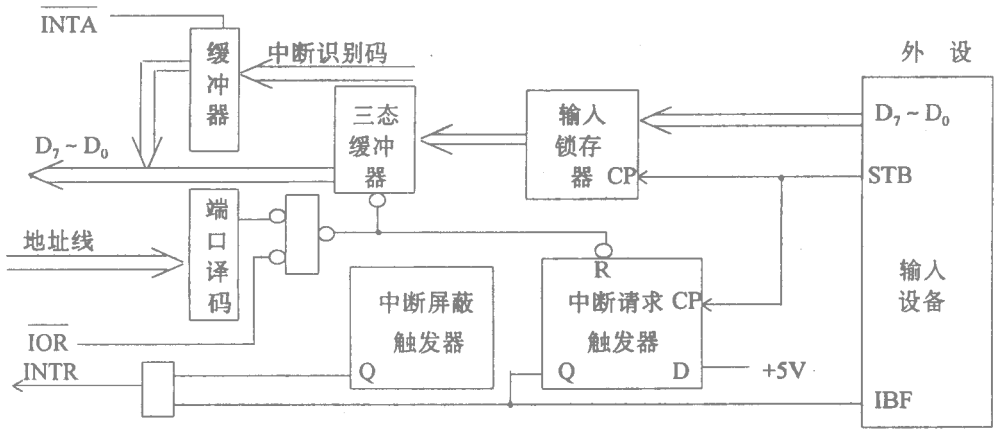


图7-20 中断方式输入的接口电路

从图7-20可以看到，当外设准备好一个数据供输入时，发出一个选通信号STB，它一方面把数据送入接口的输入锁存器中，同时使中断请求触发器置1。此时，如果中断屏蔽触发器的值为1，则INTR信号有效，请求CPU服务。中断屏蔽触发器的状态为1还是0决定了系统是否允许该接口发出中断请求。

进入中断服务例程后，CPU用IN指令打开三态缓冲器读取输入数据的同时，把中断请求触发器清“0”，以撤消此次中断请求信号，并为记忆下一次中断请求信号作准备。

还有一类CPU的中断请求信号输入线是前沿加电平有效的。

2. 中断请求信号的传送

(1) 独立中断请求线　每个中断源单独设置中断请求线，将中断请求直接送往CPU，见图7-21a 这种方式的特点是CPU在接到中断请求的同时也就知道了中断源是谁，中断的响应速度可以很快，但硬件代价较大，且CPU所能连接的中断请求数目有限，难以扩充。

(2) 公共请求线（单线）各中断源共用一根公共请求线，见图 7-21b。这种方式的特点是在负载允许的情况下，中断源的数目可随意扩充。但CPU在接到中断请求后，必须通过软件或硬件的方法来识别中断源。

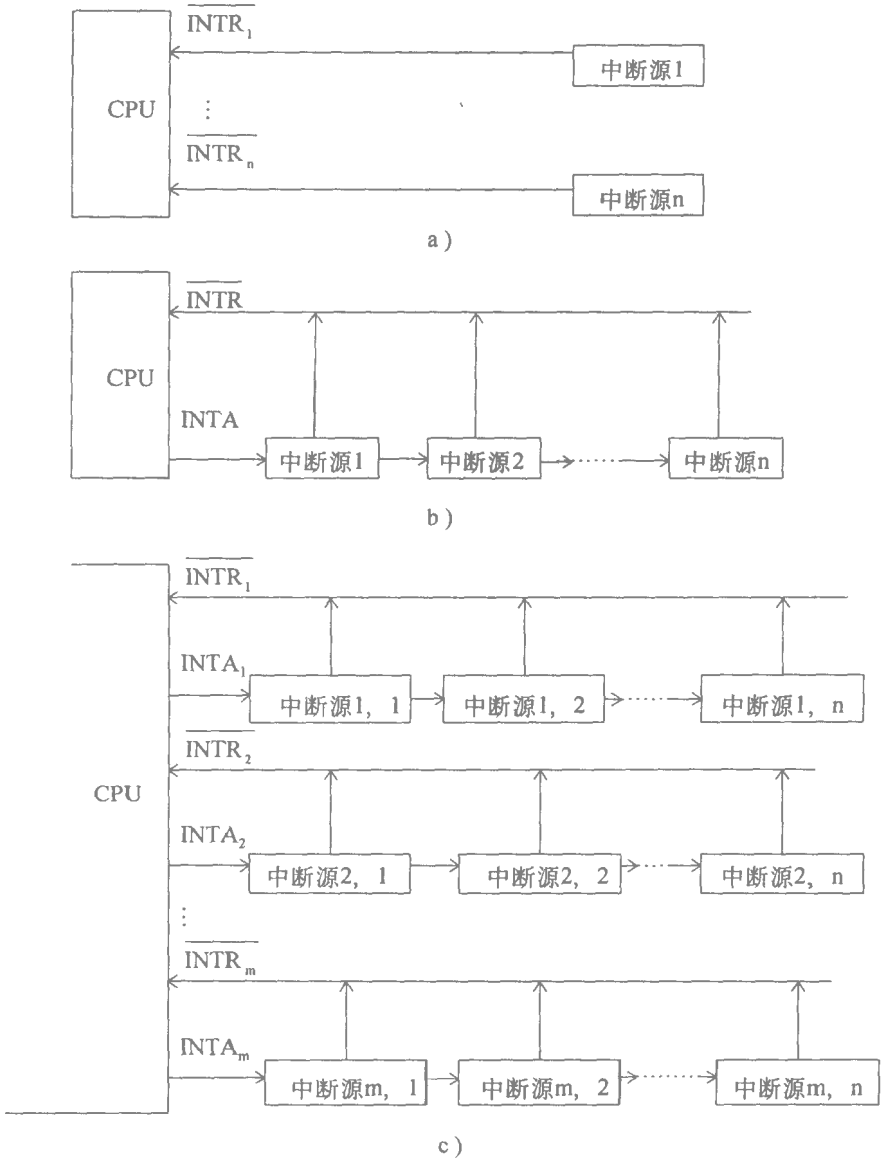


图7-21　中断请求信号的传送