```
OUT PORT, REG
```

CPU可以将REG的内容写入到控制寄存器中。大多数早期计算机，包括几乎所有大型主机，如IBM 360及其所有后续机型，都是以这种方式工作的。

在这一方案中，内存地址空间和I/O地址空间是不同的，如图5-2a所示。指令

```
IN R0, 4
```

和

```
MOV R0, 4
```

在这一设计中完全不同。前者读取I/O端口4的内容并将其存入R0，而后者则读取内存字4的内容并将其存入R0。因此，这些例子中的4引用的是不同且不相关的地址空间。

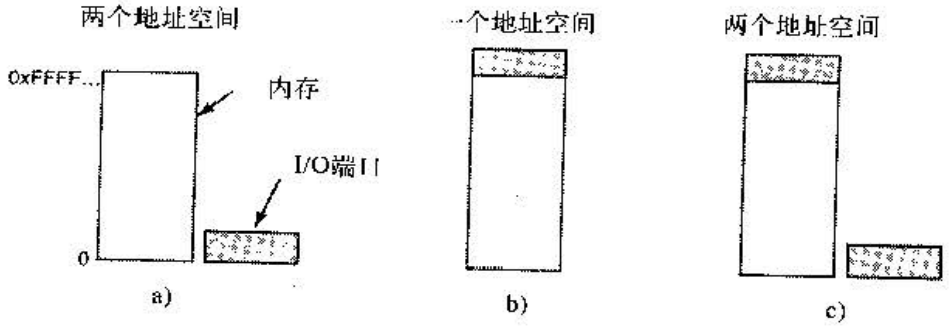

图5-2 a) 单独的I/O和内存空间；b) 内存映射I/O；c) 混合方案

第二个方法是PDP-11引入的，它将所有控制寄存器映射到内存空间中，如图5-2b所示。每个控制寄存器被分配惟一的一个内存地址，并且不会有内存被分配这一地址。这样的系统称为内存映射I/O（memory-mapped I/O）。通常分配给控制寄存器的地址位于地址空间的顶端。图5-2c所示是一种混合的方案，这一方案具有内存映射I/O的数据缓冲区，而控制寄存器则具有单独的I/O端口。Pentium处理器使用的就是这一体系结构。在IBM PC兼容机中，除了0到64K−1的I/O端口之外，640K到1M−1的地址保留给设备的数据缓冲区。

这些方案是怎样工作的？在各种情形下，当CPU想要读入一个字的时候，不论是从内存中读入还是从I/O端口中读入，它都要将需要的地址放到总线的地址线上，然后在总线的一条控制线上置起一个READ信号。还要用到第二条信号线来表明需要的是I/O空间还是内存空间。如果是内存空间，内存将响应请求。如果是I/O空间，I/O设备将响应请求。如果只有内存空间（如图5-2b所示的情形），那么每个内存模块和每个I/O设备都会将地址线和它所服务的地址范围进行比较，如果地址落在这一范围之内，它就会响应请求。因为绝对不会有地址既分配给内存又分配给I/O设备，所以不会存在歧义和冲突。

这两种寻址控制器的方案具有不同的优缺点。我们首先来看一看内存映射I/O的优点。第一，如果需要特殊的I/O指令读写设备控制寄存器，那么访问这些寄存器需要使用汇编代码，因为在C或C++中不存在执行IN或OUT指令的方法。调用这样的过程增加了控制I/O的开销。相反，对于内存映射I/O，设备控制寄存器只是内存中的变量，在C语言中可以和任何其他变量一样寻址。因此，对于内存映射I/O，I/O设备驱动程序可以完全用C语言编写。如果不使用内存映射I/O，就要用到某些汇编代码。

第二，对于内存映射I/O，不需要特殊的保护机制来阻止用户进程执行I/O操作。操作系统必须要做的全部事情只是避免把包含控制寄存器的那部分地址空间放入任何用户的虚拟地址空间之中。更为有利的是，如果每个设备在地址空间的不同页面上拥有自己的控制寄存器，操作系统只要简单地通过在其页表中包含期望的页面就可以让用户控制特定的设备而不是其他设备。这样的方案可以使不同的设备驱动程序放置在不同的地址空间中，不但可以减小内核的大小，而且可以防止驱动程序之间相互干扰。

第三，对于内存映射I/O，可以引用内存的每一条指令也可以引用控制寄存器。例如，如果存在一条指令TEST可以测试一个内存字是否为0，那么它也可以用来测试一个控制寄存器是否为0，控制寄存器为0可以作为信号，表明设备空闲并且可以接收一条新的命令。汇编语言代码可能是这样的：