```
LOOP:   TEST PORT_4      //检测端口4是否为0
        BEQ READY        //如果为0，转向READY
        BRANCH LOOP      //否则，继续测试
READY:
```

如果不是内存映射I/O，那么必须首先将控制寄存器读入CPU，然后再测试，这样就需要两条指令而不是一条。在上面给出的循环的情形中，就必须加上第四条指令，这样会稍稍降低检测空闲设备的响应度。

在计算机设计中，实际上任何事情都要涉及权衡，此处也不例外。内存映射I/O也有缺点。首先，现今大多数计算机都拥有某种形式的内存字高速缓存。对一个设备控制寄存器进行高速缓存可能是灾难性的。在存在高速缓存的情况下考虑上面给出的汇编代码循环。第一次引用PORT_4将导致它被高速缓存，随后的引用将只从高速缓存中取值并且不会再查询设备。之后当设备最终变为就绪时，软件将没有办法发现这一点。结果，循环将永远进行下去。

对内存映射I/O，为了避免这一情形，硬件必须针对每个页面具备选择性禁用高速缓存的能力。操作系统必须管理选择性高速缓存，所以这一特性为硬件和操作系统两者增添了额外的复杂性。

其次，如果只存在一个地址空间，那么所有的内存模块和所有的I/O设备都必须检查所有的内存引用，以便了解由谁做出响应。如果计算机具有单一总线，如图5-3a所示，那么让每个内存模块和I/O设备查看每个地址是简单易行的。

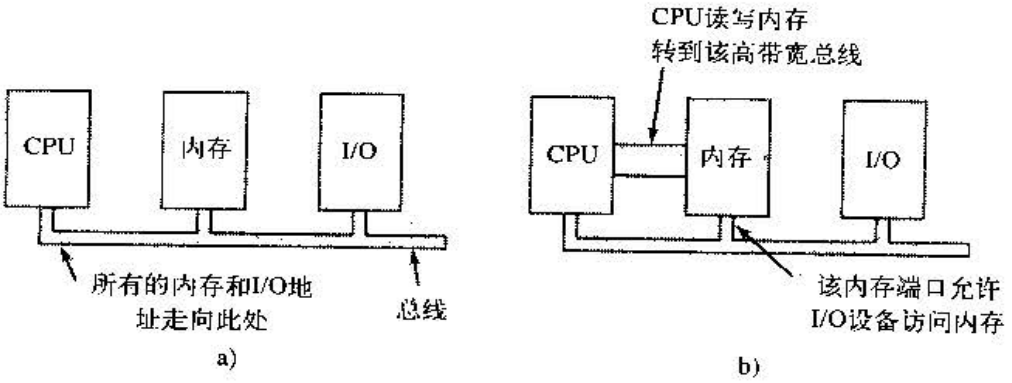

图5-3 a) 单总线体系结构；b) 双总线内存体系结构

然而，现代个人计算机的趋势是包含专用的高速内存总线，如图5-3b所示。顺便提一句，在大型机中也可以发现这一特性。装备这一总线是为了优化内存性能，而不是为了慢速的I/O设备而做的折中。Pentium系统甚至可以有多种总线（内存、PCI、SCSI、USB、ISA），如图1-12所示。

在内存映射的机器上具有单独的内存总线的麻烦是I/O设备没有办法查看内存地址，因为内存地址旁路到内存总线上，所以没有办法响应。此外，必须采取特殊的措施使内存映射I/O工作在具有多总线的系统上。一种可能的方法是首先将全部内存引用发送到内存，如果内存响应失败，CPU将尝试其他总线。这一设计是可以工作的，但是需要额外的硬件复杂性。

第二种可能的设计是在内存总线上放置一个探查设备，放过所有潜在地指向所关注的I/O设备的地址。此处的问题是，I/O设备可能无法以内存所能达到的速度处理请求。

第三种可能的设计是在PCI桥芯片中对地址进行过滤，这正是图1-12中Pentium结构上所使用的。该芯片中包含若干个在引导时预装载的范围寄存器。例如，640K到1M−1可能被标记为非内存范围。落在标记为非内存的那些范围之内的地址将被转发给PCI总线而不是内存。这一设计的缺点是需要在引导时判定哪些内存地址不是真正的内存地址。因而，每一设计都有支持它和反对它的论据，所以折中和权衡是不可避免的。

5.1.4 直接存储器存取

无论一个CPU是否具有内存映射I/O，它都需要寻址设备控制器以便与它们交换数据。CPU可以从I/O控制器每次请求一个字节的数据，但是这样做浪费CPU的时间，所以经常用到一种称为直接存储器存取（Direct Memory Access，DMA）的不同方案。只有硬件具有DMA控制器时操作系统才能使用