Ch 8: Interconnecting and I/O

互连及输入输出组织

第一讲 I/O设备和磁盘存储器及其使用

第二讲 总线类型、系统中的设备间的互连及I/O接口

第三讲 I/O设备与其它设备间的传输方式(I/O传输方式)

第一讲 I/O设备和磁盘存储器及其使用

主 要 内 容

- I/O系统概述
  - I/O系统的功能及性能
  - OS在I/O系统中的角色
- I/O设备概述
  - I/O设备的通用模型
- 磁盘存储器及其使用
  - 磁盘存储器的读写原理
  - 磁盘存储器的性能指标
  - 冗余磁盘阵列 (RAID)

I/O System的性能

- 两个常用的性能指标：
  - Throughput: I/O bandwidth (吞吐率，即：I/O带宽)：
    - 单位时间内从系统输入/输出多少数据？
    - 单位时间内实现了多少次输入/输出操作？
  - Response time: Latency (响应时间，即：等待延迟)：
    - 在多长时间内完成请求的任务？
- 不同的任务对性能的要求不同：
  - 要求吞吐量高的场合：
    - 如：多媒体应用（音/视频的播放要流畅！）
  - 要求响应时间短的场合：
    - 如：事务处理系统（存/取款的速度要快！）
  - 要求吞吐率高且响应时间短的场合：
    - 文件服务器、Web服务器等

I/O System的功能

- 输入/出系统的功能：
  - 解决各种形式信息的输入和输出

    即：用户如何将所需的信息（文字、图表、声音、视频等）通过不同的外设输入到计算机中，以及计算机内部处理的结果信息如何通过相应的外设输出给用户
- 要实现上述功能需解决以下一系列的问题：
  - 怎样在CPU、主存和外设间建立一个高效信息传输 “通路”；
  - 怎样将用户的I/O请求转换成设备的命令；
  - 如何对外设进行编址；
  - 怎样使CPU方便地寻找到要访问的外设；
  - I/O硬件和操作系统如何协调完成主机和外设之间的数据传送

    等等

以上是本章的主要内容

外设发展与分类

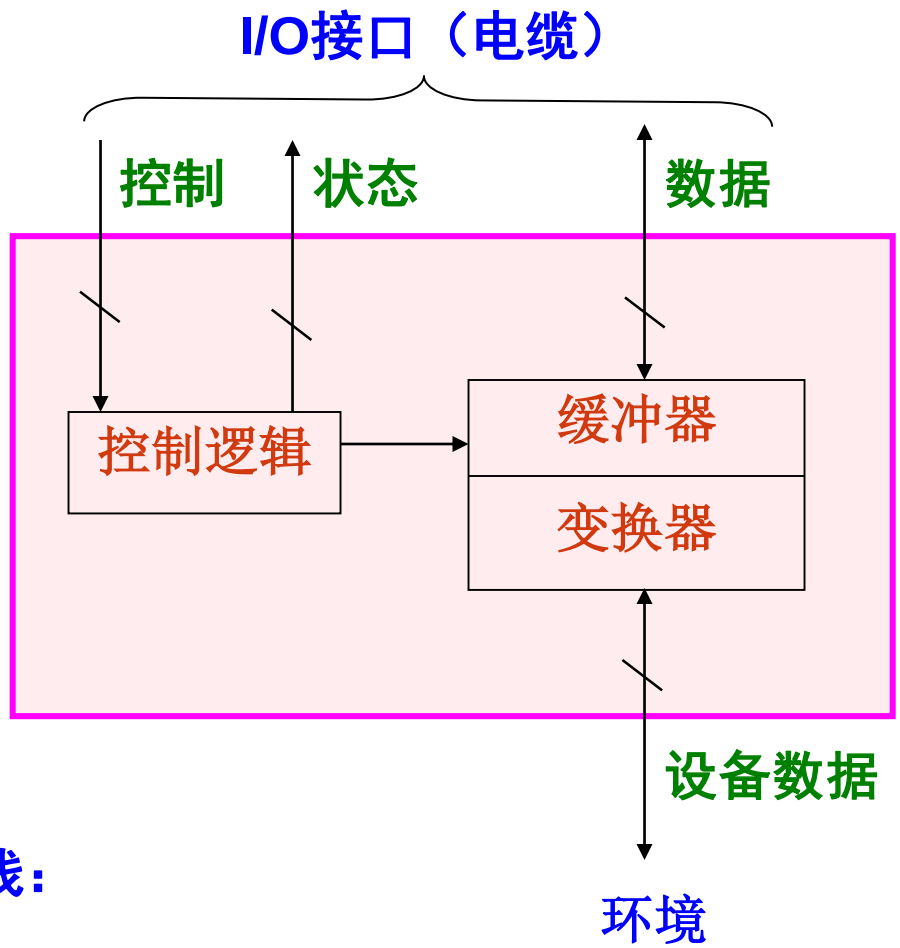
- 从交互方式上来分，外设分为：
  - 人-机交互设备
    - 输入/输出的信息是人可读的
    - 如：键盘、鼠标、扫描仪、打印机、显示器等
  - 机器可读设备
    - 输入/输出的信息是机器可读的，人无法读取
    - 如：网络、Modem、D/A、A/D、磁盘、声音输入设备等
- 从功能行为来分，外设分为：
  - 输入/输出设备（大部分为字符型设备）
    - 用于信息的输入/输出
    - 输入设备：键盘、鼠标、扫描仪等
    - 输出设备：打印机、显示器等
  - 外部存储设备（大部分为成块传送设备）
    - 用于信息的存储（其输入/出的信息是机器可读的）
    - 如：磁盘、磁带、光盘等

常用外部设备

- **输入设备：**
  - **键盘、触摸屏**
  - **图形输入设备(鼠标、图形板、跟踪球、操纵杆、光笔)**
  - **图像输入设备(摄像机、扫描仪、传真机)**
  - **条形码阅读机、光学字符识别设备(OCR)**
  - **音、视频输入设备**
- **输出设备：**
  - **显示器(字符、汉字、图形、图像)**
  - **打印设备(点阵、激光、喷墨)**
  - **绘图仪 (平板式、滚筒式)**
  - **声音输出设备**
- **其它：**
  - **终端设备(键盘+显示器)**
  - **外存储器(磁盘、磁带、光盘)**

外部设备的通用模型

- 通过**电缆**与计算机内部I/O接口进行数据、状态和控制信息的传送
- **控制逻辑**根据控制信息控制设备的操作，并检测设备状态
- **缓冲器**用于保存交换的数据信息
- **变换器**用于在电信号形式（内部数据）和其他形式的设备数据之间进行转换



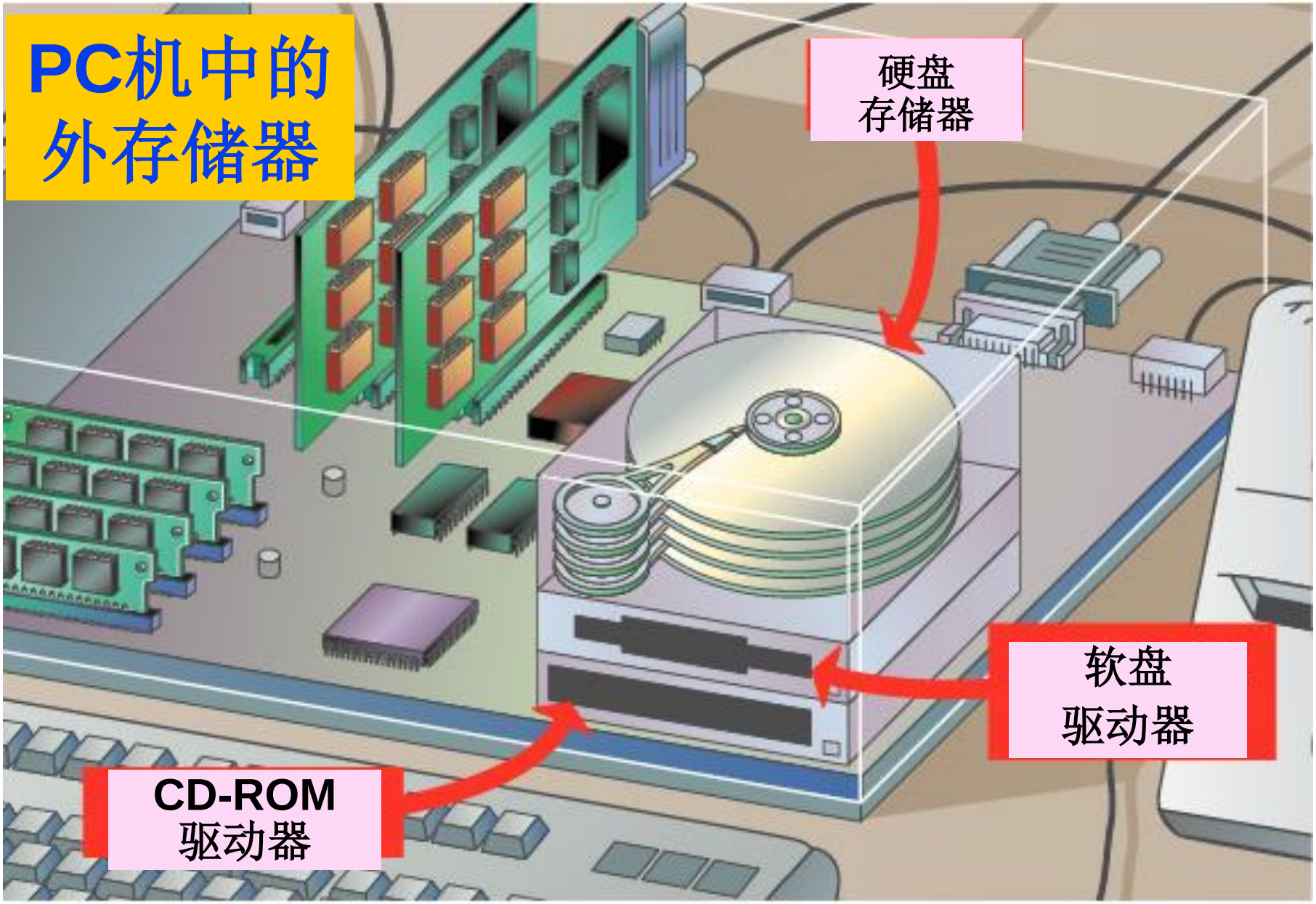
所有设备都可以抽象成这个通用模型！

设备所用的电缆线中有以下三种信号线：

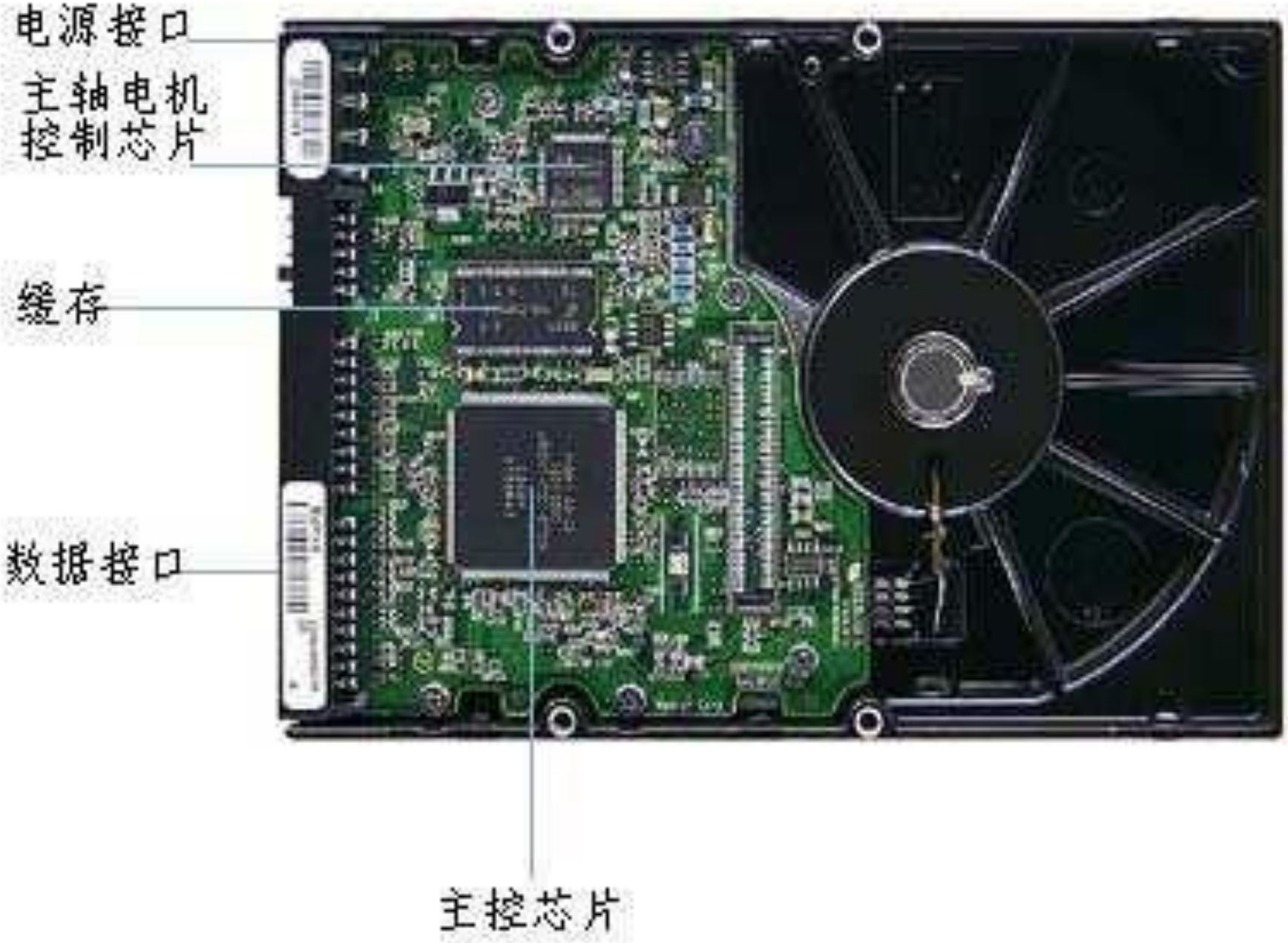
控制信号、状态信号、数据信号

下面以磁盘为例，说明外部设备的工作原理

PC中的外存储器



PC中的外存储器—磁盘外表面



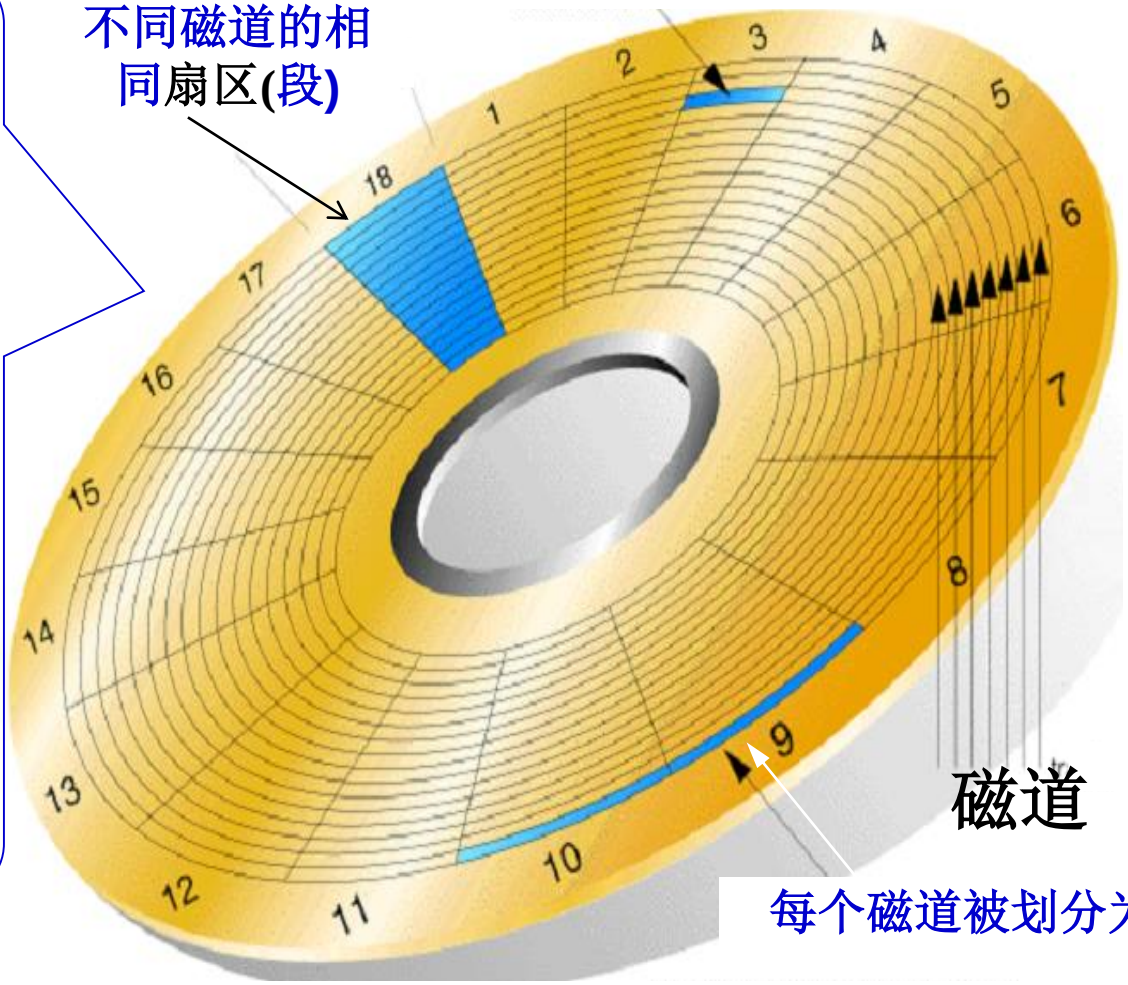
PC中的外存储器-磁盘内部

磁盘的磁道和扇区

1个扇区(也叫段),磁盘最小的物理存储单元,每个扇区都有一个编号

磁盘表面被分为许多同心圆，每个同心圆称为一个磁道。每个磁道存储的数据容量相同.每个磁道都有一个编号，最外面的是0磁道

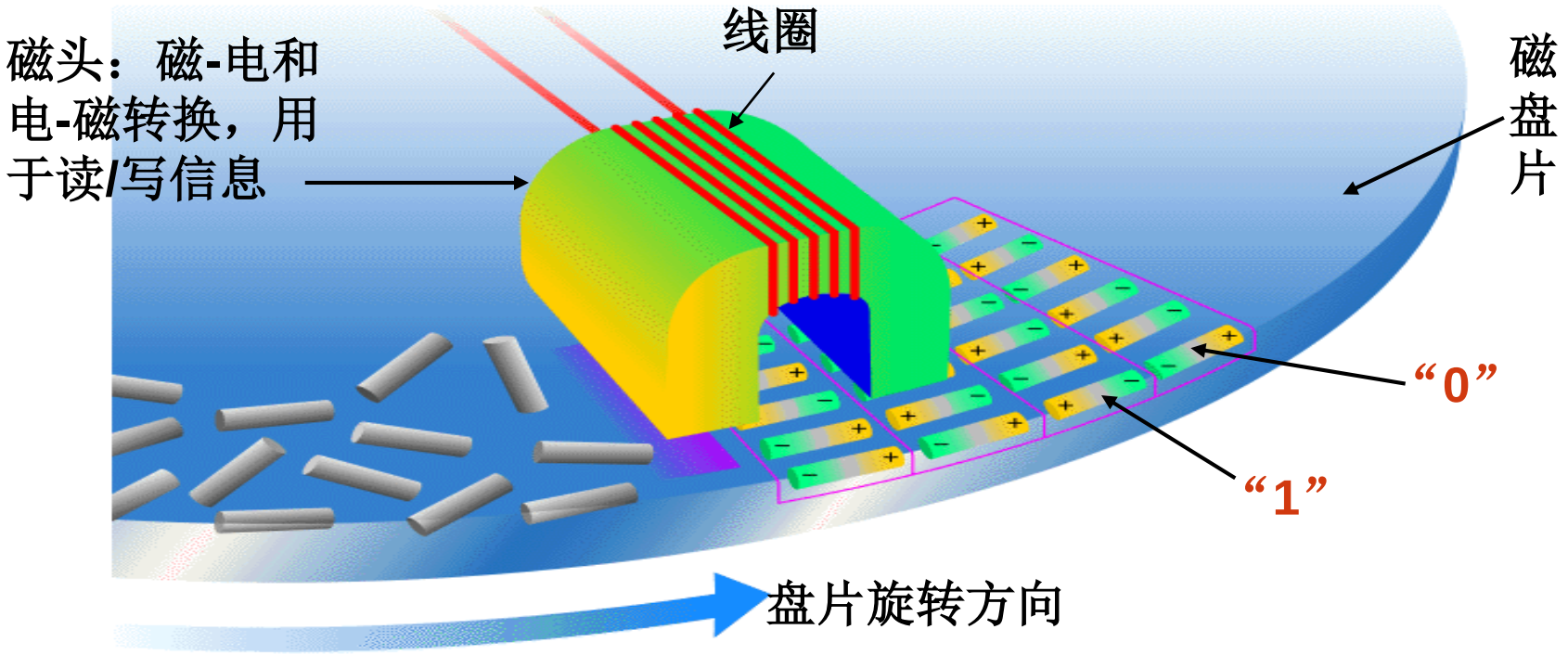
不同磁道的相同扇区(段)



近三十年来，扇区大小一直是512字节。但最近几年正迁移到更大、更高效的4096字节扇区，通常称为4K扇区。国际硬盘设备与材料协会（IDEMA）将之称为高级格式化。

由于操作系统无法对数目众多的扇区进行寻址，所以操作系统就将相邻的扇区组合在一起，形成一个簇，然后再对簇进行管理。每个簇可以包括2、4、8、16、32或64个扇区。簇是操作系统使用的逻辑概念，非磁盘的物理特性。

磁盘存储器的信息存储原理



写1：线圈通以正向电流，使呈N-S状态

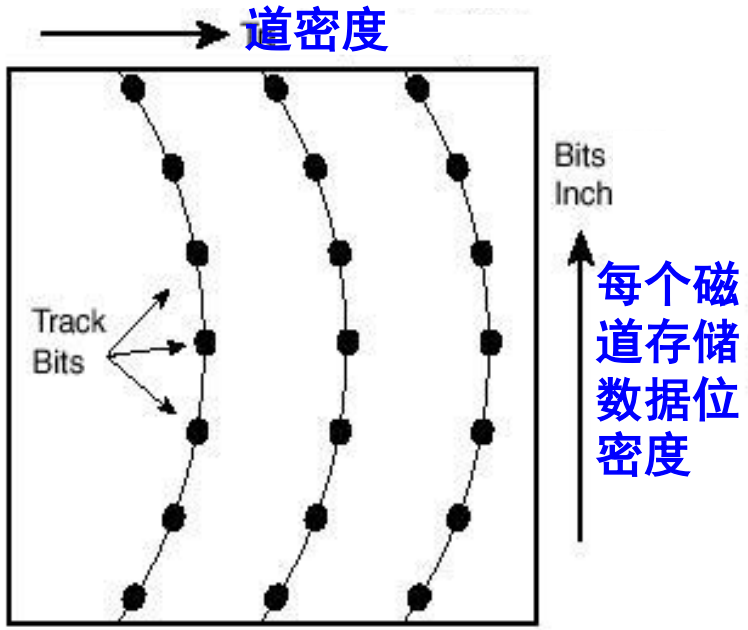
写0：线圈通以反向电流，使呈S-N状态

} 不同的磁化状态被记录在磁盘表面

读时：磁头固定不动，载体运动。因为载体上小的磁化单元外部的磁力线通过磁头铁芯形成闭合回路，在铁芯线圈两端得到感应电压。根据感应电压的不同的极性，可确定读出为0或1。

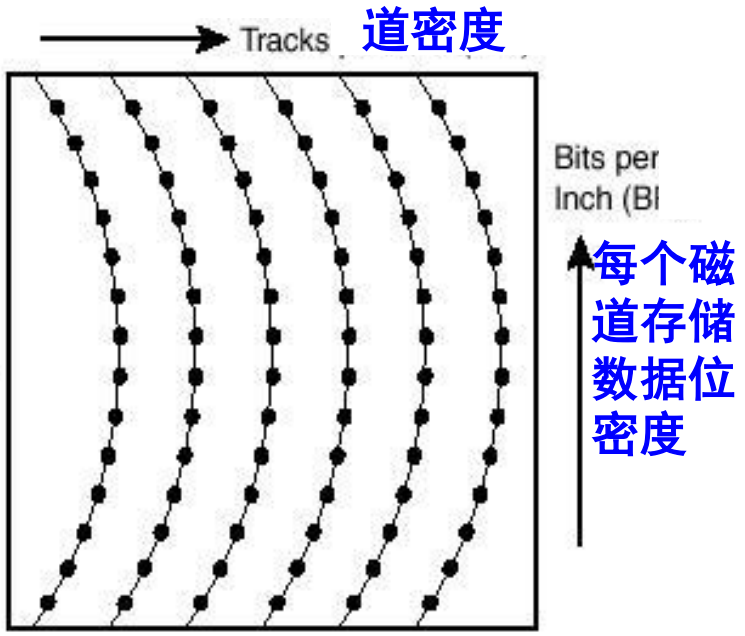
如何增大磁盘片的容量？

- 提高盘片上的信息记录密度！
  - 增加磁道数目——提高磁道密度
  - 增加扇区数目——提高位密度，并采用可变扇区数



低密度存储示意图

早期磁盘所有磁道上的扇区数相同，所以位数相同，内道上的位密度比外道位密度高

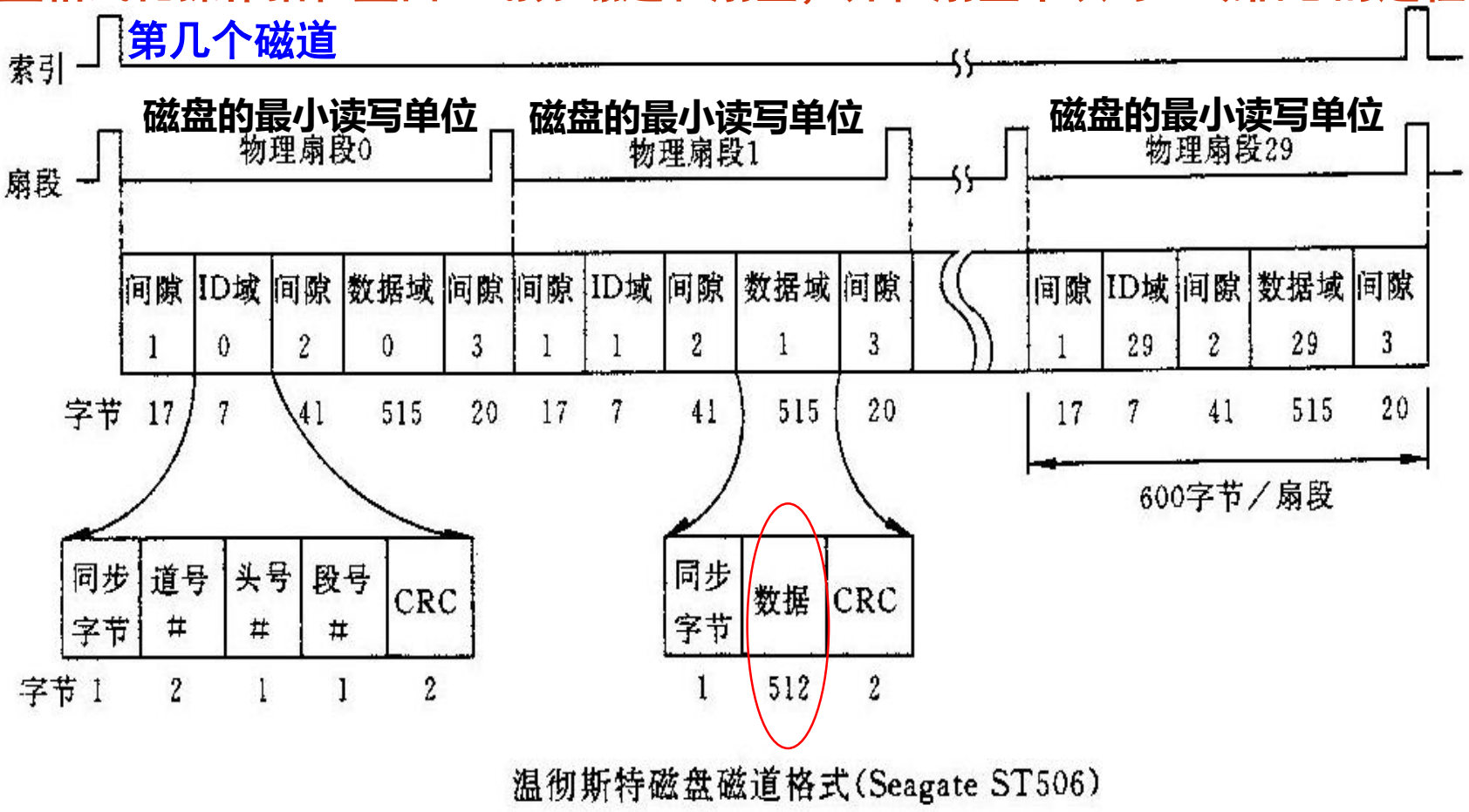


高密度存储示意图

现代磁盘磁道上的位密度相同，所以，外道上的扇区数比内道上扇区数多，使整个磁盘的容量提高

磁盘磁道的格式

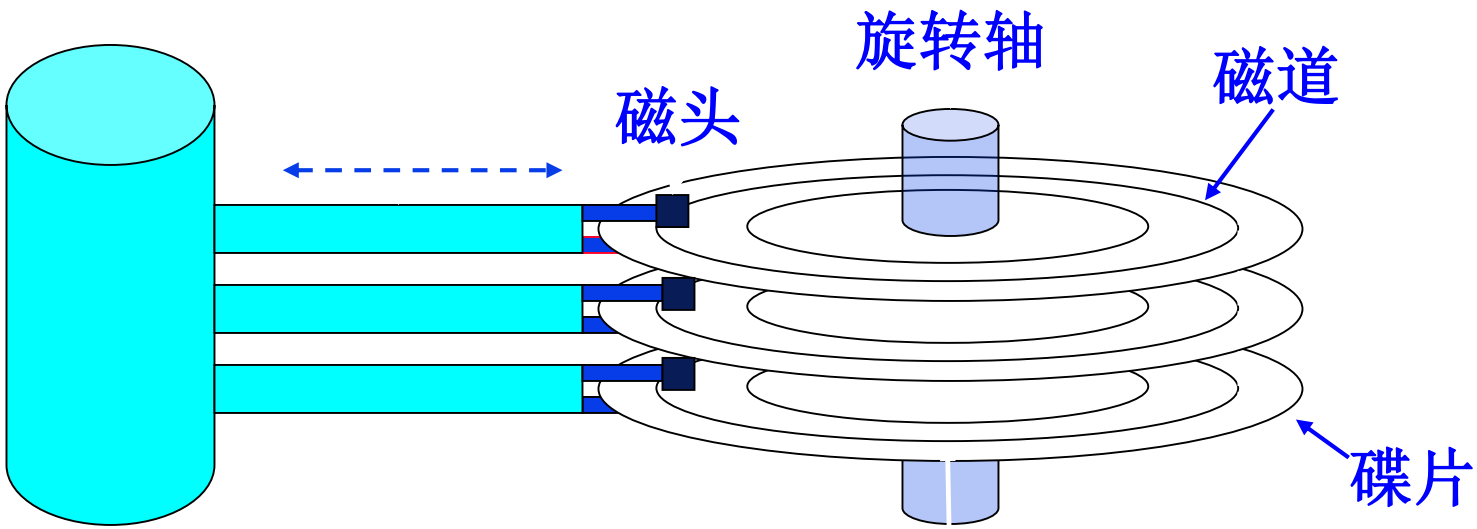
磁盘格式化操作指在盘面上划分磁道和扇区，并在扇区中填写ID域信息的过程



温彻斯特磁盘磁道格式(Seagate ST506)

在此例中，每个磁道包含30个固定长度的扇段，每个扇段有600个字节(17+7+41+515+20=600)。

平均存取时间



硬盘的操作流程如下：

所有磁头同步寻道（由柱面号控制）→ 选择磁头（由磁头号控制） →
被选中磁头等待扇区到达磁头下方（由扇区号控制） → 读写该扇区中数据

- **磁盘上的信息以扇区为单位进行读写，平均存取时间为：**

  T = 平均寻道时间 + 平均旋转等待时间 + 数据传输时间（忽略不计）

  - **平均寻道时间——磁头寻找到指定磁道所需平均时间 (大约5ms)**
  - **平均旋转等待时间——指定扇区旋转到磁头下方所需平均时间 (大约4～6ms)**
    (转速： 4200 / 5400 / 7200 / 10000rpm)
  - **数据传输时间——(大约0.01ms / 扇区)**

磁盘响应(用户请求读数据)时间计算举例

假定每个扇区512字节， 磁盘转速为5400RPM(Round Per Minute圈(或转)/分钟)(=5400/60=90RPS(圈/秒))，声称寻道时间为12 ms, 数据传输率为4 MB/s (=4*10^6 B/s,不是4*1024*1024B/s), 磁盘控制器开销为1 ms, 不考虑排队时间，则磁盘响应时间为多少？

磁盘响应时间

=寻道时间 +平均旋转(半圈)时间 + 数据传输时间+ 控制器时间 + 排队等待时间

= 12 ms + 0.5 / 5400 RPM + 0.5 KB / 4 MB/s + 1 ms + 0

= 12 ms + 0.5 / 90 RPS + 0.125 / 1024 s + 1 ms + 0

= 12 ms + 5.5 ms + 0.1 ms + 1 ms + 0 ms

= 18.6 ms(毫秒)

如果实际的寻道时间只有1/3的话，则总时间变为10.6ms，这样旋转等待时间就占了近50%！((寻道时间12ms)/3)+5.5+0.1+1=10.6ms 所以，磁盘转速非常重要！

为什么实际的寻道时间可能只有1/3？访问局部性使得每次磁盘访问大多在局部几个磁道，实际寻道时间变少！

能否算出每道有多少个扇区？

每分钟传送数据: ((4MB/S)x(60秒))＝(4*10^6 B/s) * 60秒= 240ＭＢ＝240 *10^6 B

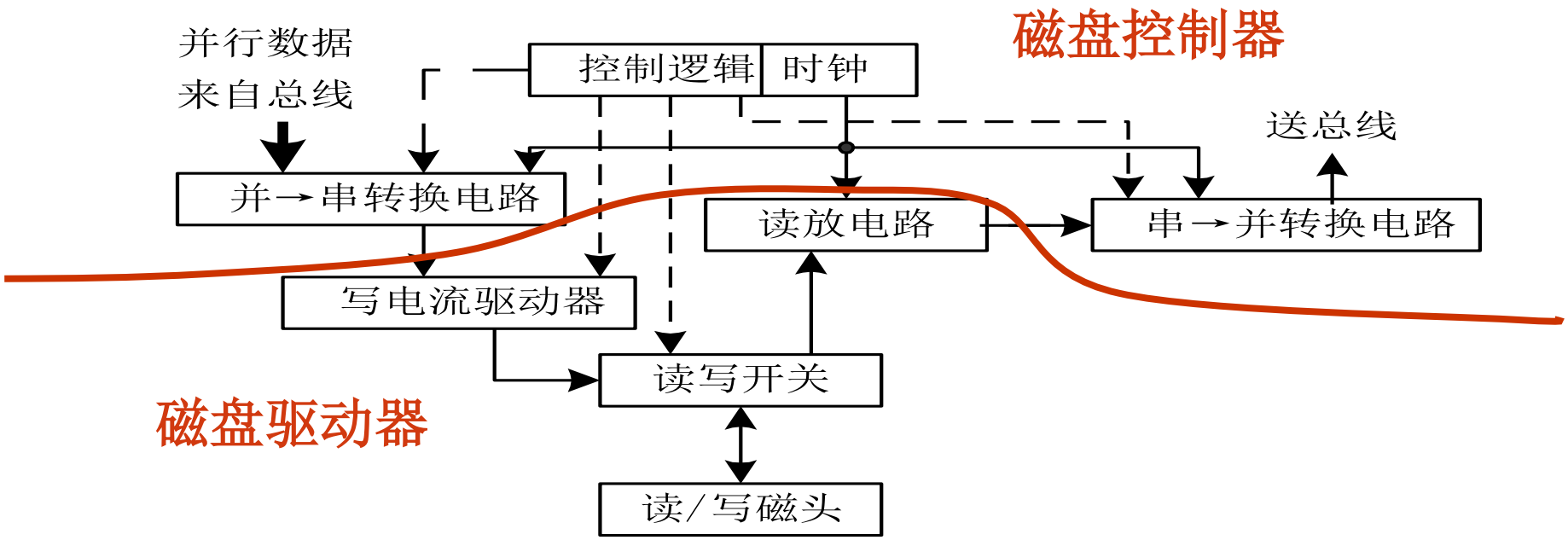
每分钟转5400圈,平均每圈传送数据量:(60秒*(4*10^6 B/秒)) /(5400圈/分钟) ≈44444.44B

每圈扇区数量(假定每个扇区512字节) ≈ 44444.44B/512B = 87个扇区

硬盘存储器的组成

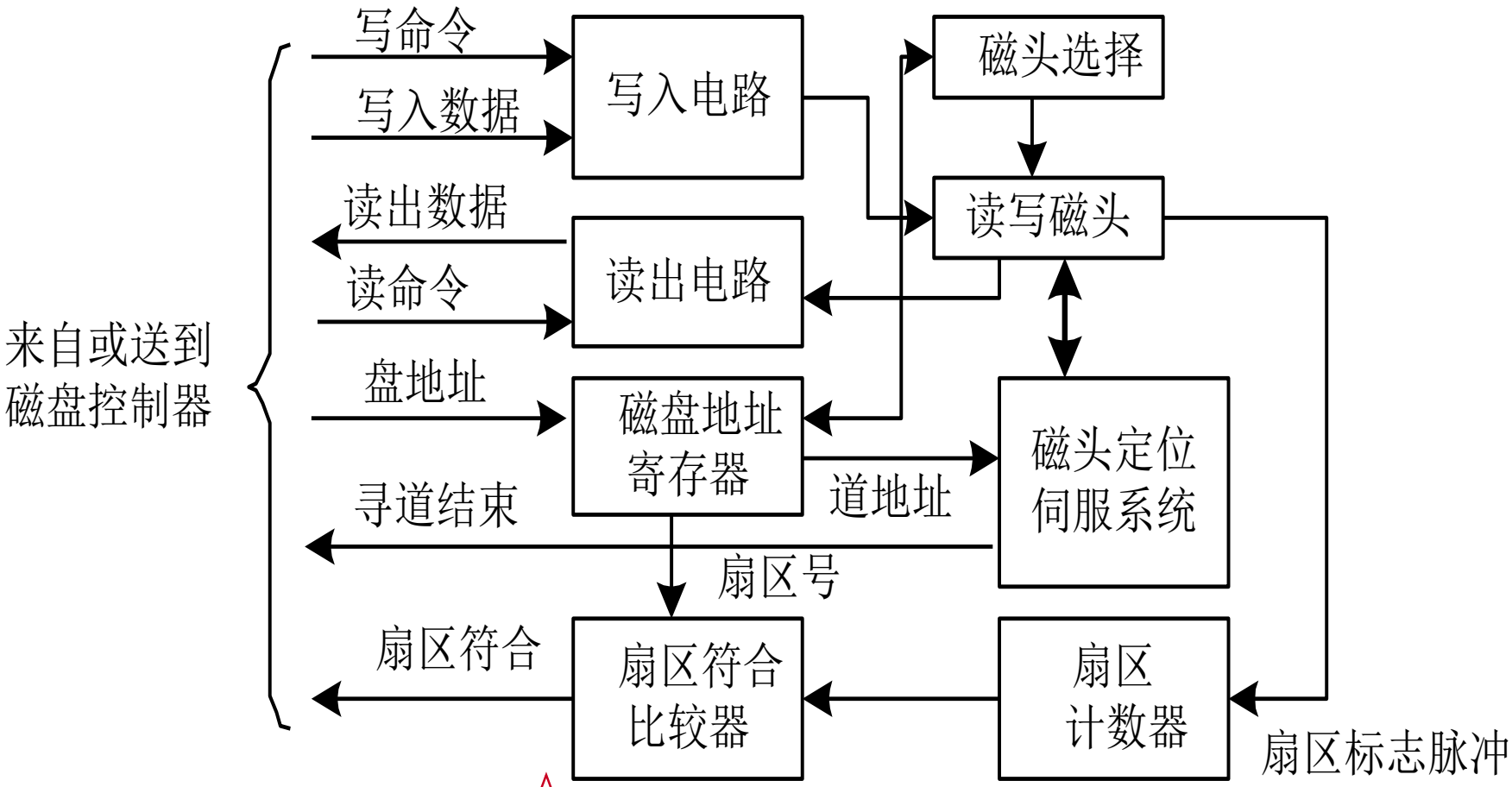
- 硬盘存储器的基本组成
  - 磁记录介质：用来保存信息
  - 磁盘驱动器：包括读写电路、读\写转换开关、读写磁头与磁头定位伺服系统等
  - 磁盘控制器：包括控制逻辑、时序电路、“并→串”转换和“串→并”转换电路等。（用于连接主机与盘驱动器）

还包括数据缓存器、控制状态寄存器等。



硬盘存储器的简化逻辑结构

硬盘驱动器的逻辑结构



与磁盘控制器之间的接口

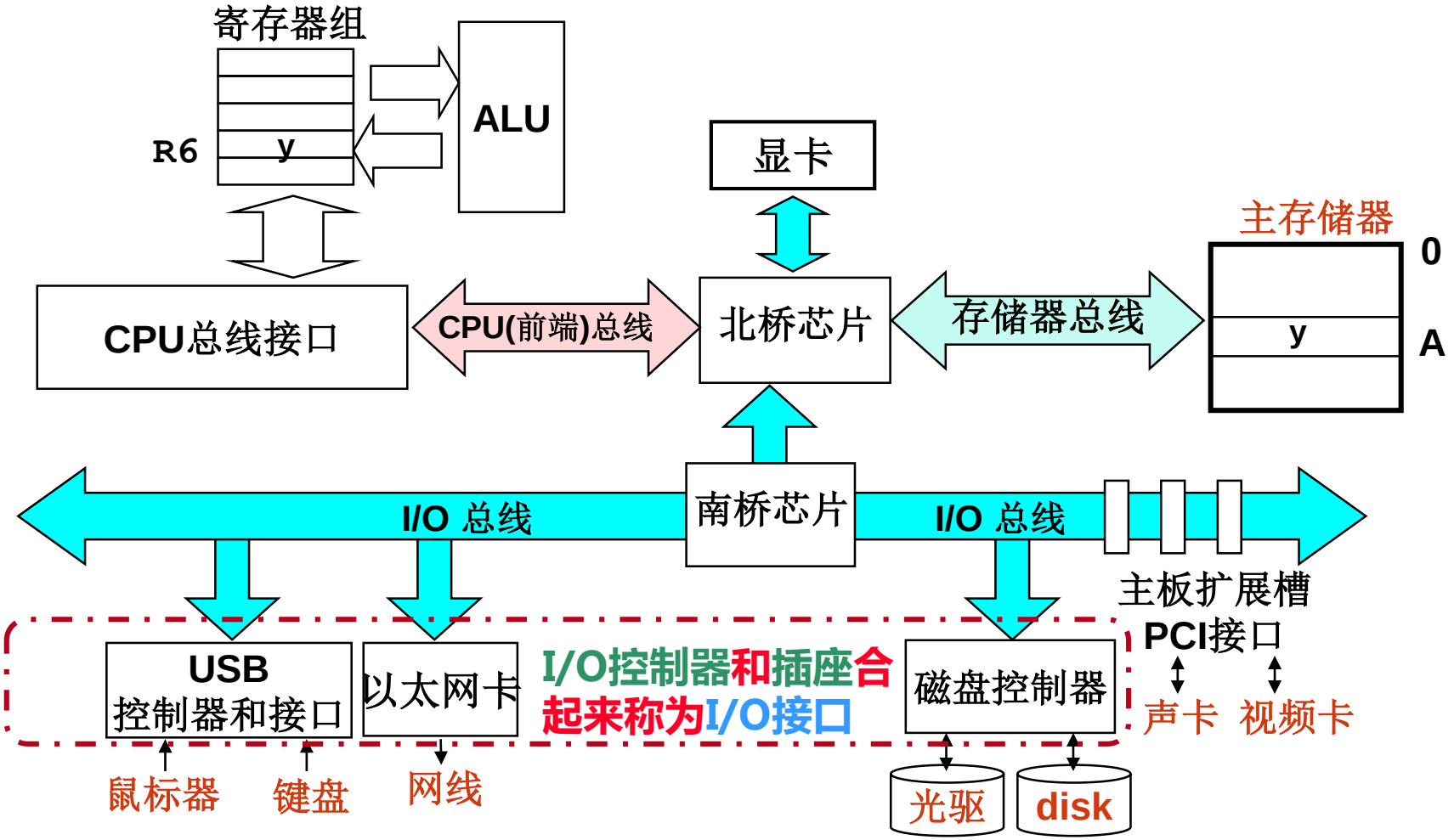
如何定位磁盘上的数据（磁盘地址格式）？

柱面(磁道)号、磁头（盘面）号、扇区号

操作过程？　寻道、旋转、读/写

I/O总线、I/O接口与I/O设备的关系

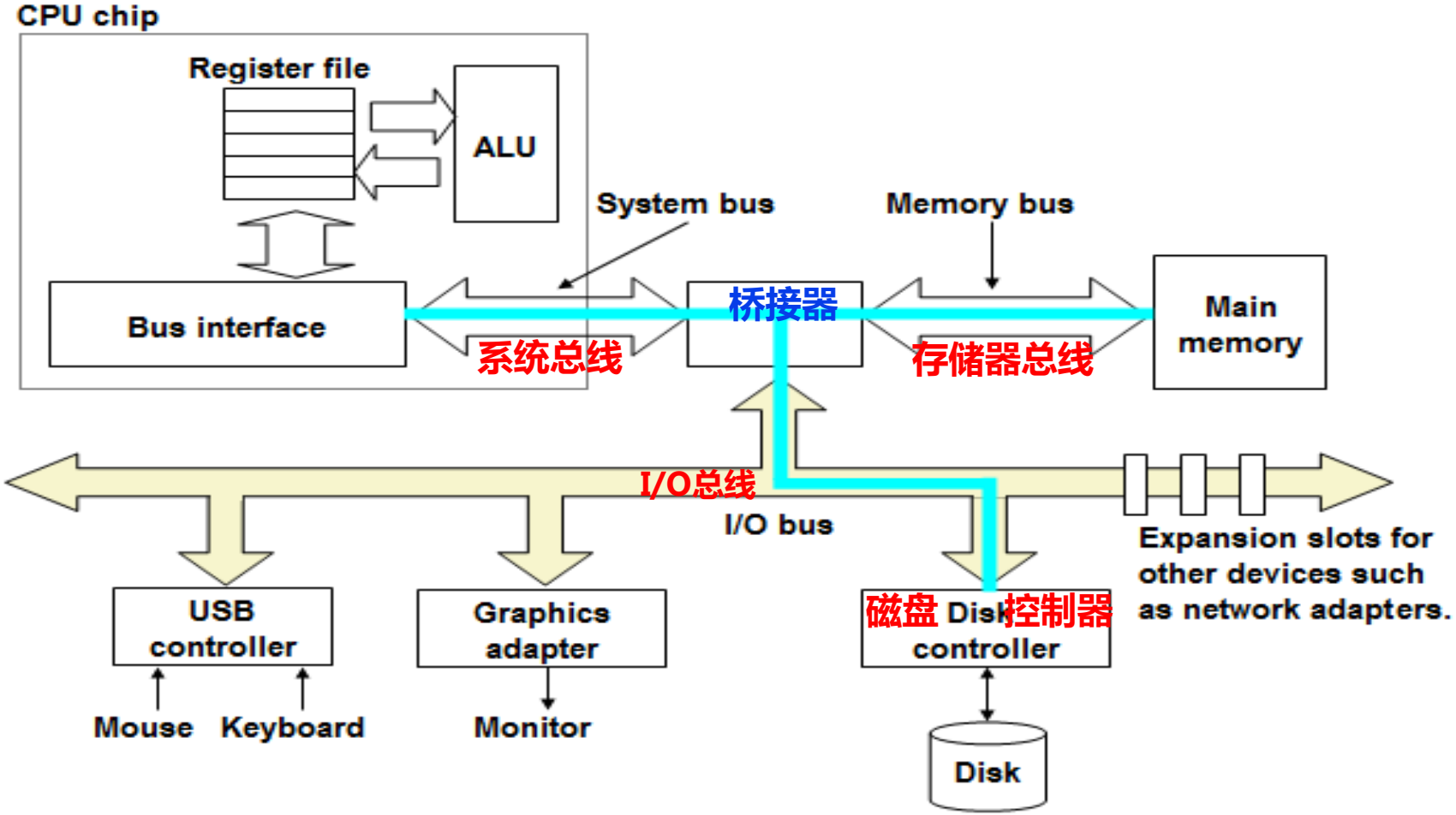
磁盘控制器连接在I/O总线上，再用桥接器与其他总线(系统总线、存储器总线)连接



磁盘的最小读写单位是扇区，因此，磁盘按成批数据交换方式进行读写，采用直接存储器存取（DMA，Direct Memory Access）方式进行数据输入输出，需用专门的DMA接口来控制外设与主存间直接数据交换，数据不通过CPU。通常把专门用来控制总线进行DMA传送的接口硬件称为DMA控制器

磁盘存储器的连接

磁盘控制器连接在I/O总线上，再用桥接器与其他总线(系统总线、存储器总线)连接



磁盘的最小读写单位是扇区，因此，磁盘按**成批数据交换**方式进行读写，采用**直接存储器存取（DMA，Direct Memory Access）方式**进行数据输入输出，需用专门的DMA接口来控制外设与主存间直接数据交换，数据不通过CPU。通常把专门用来控制总线进行DMA传送的接口硬件称为**DMA控制器**

读一个磁盘扇区 - 第一步CPU对磁盘控制器初始化

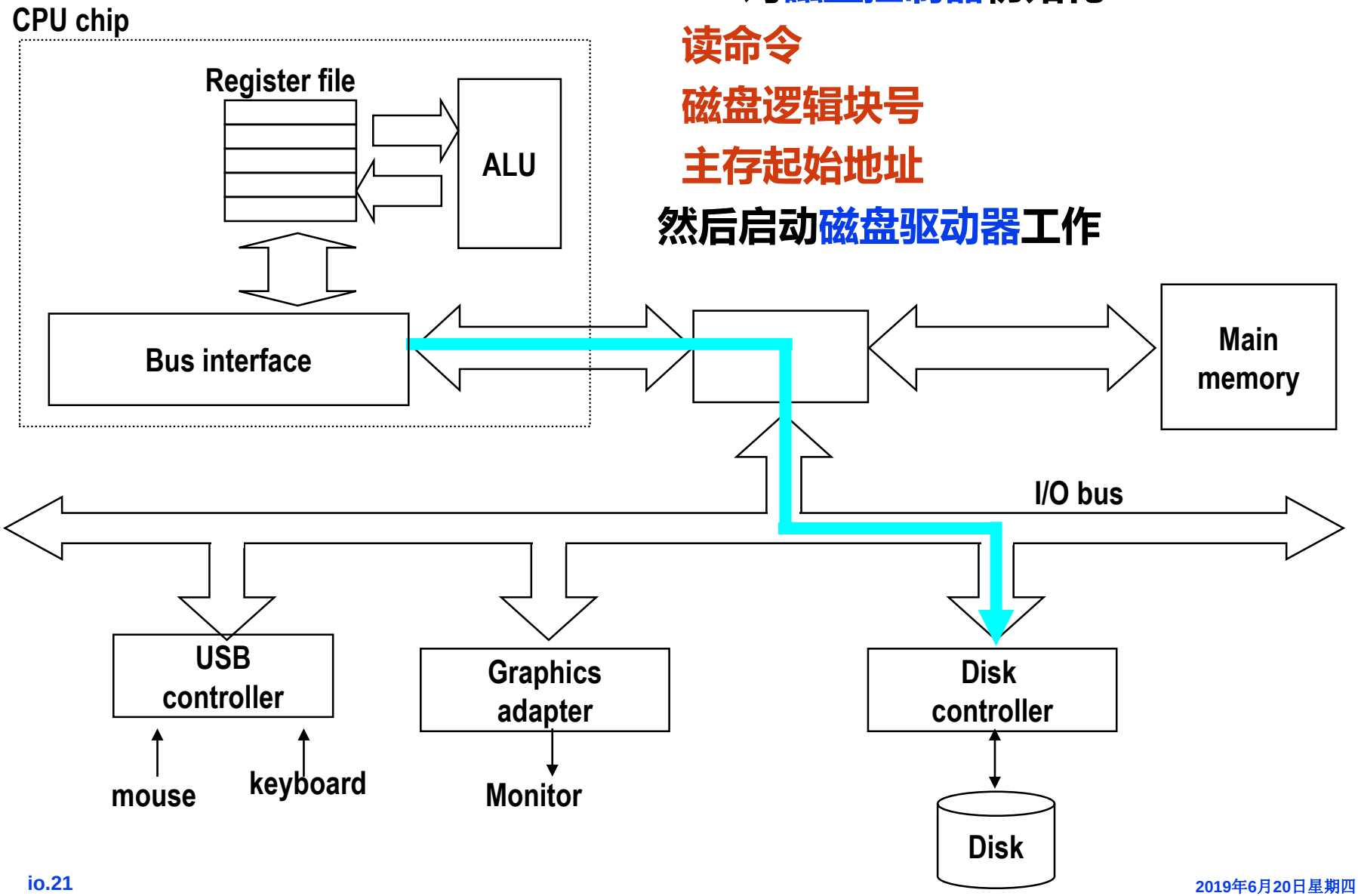
CPU对磁盘控制器初始化：

读命令

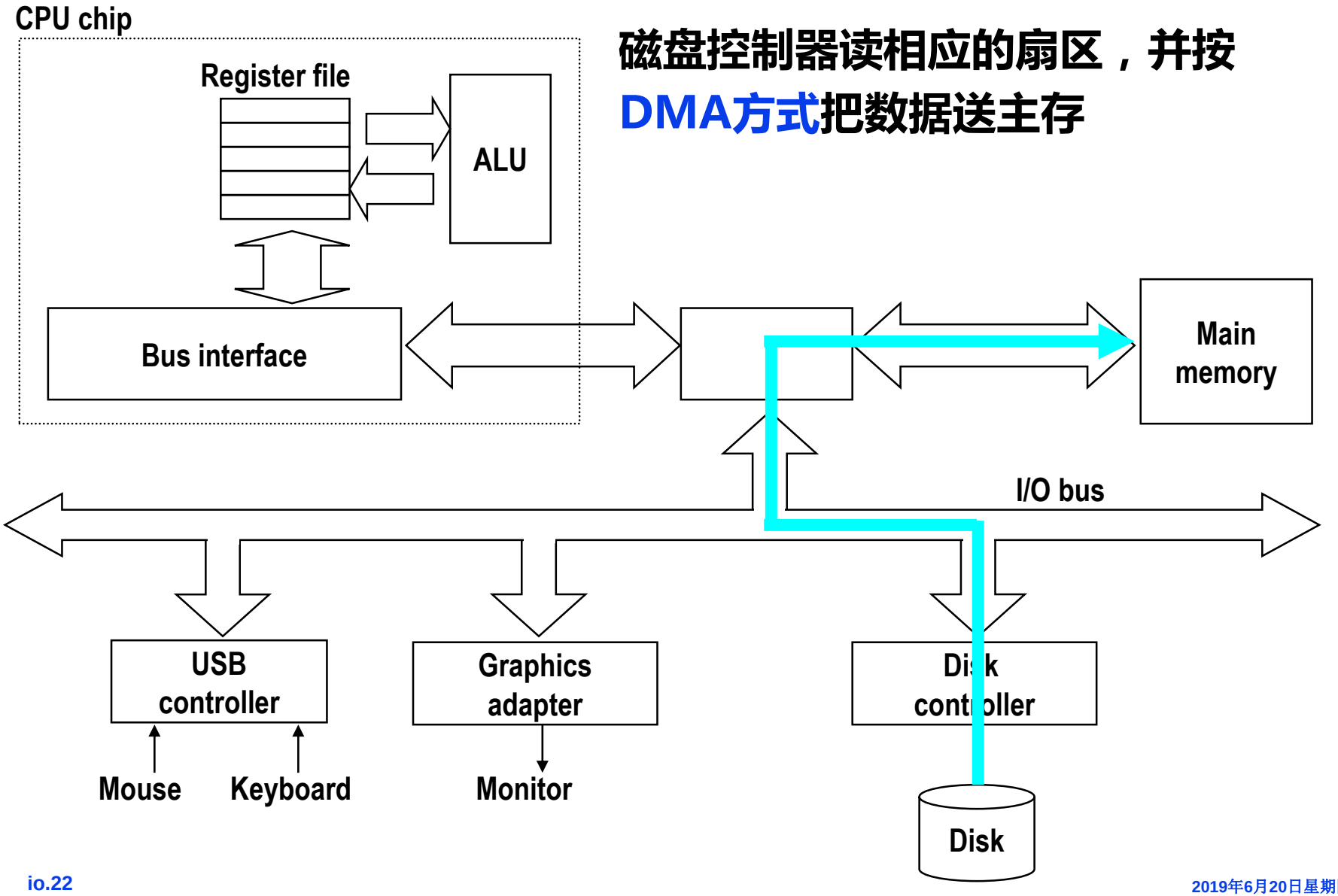
磁盘逻辑块号

主存起始地址

然后启动磁盘驱动器工作

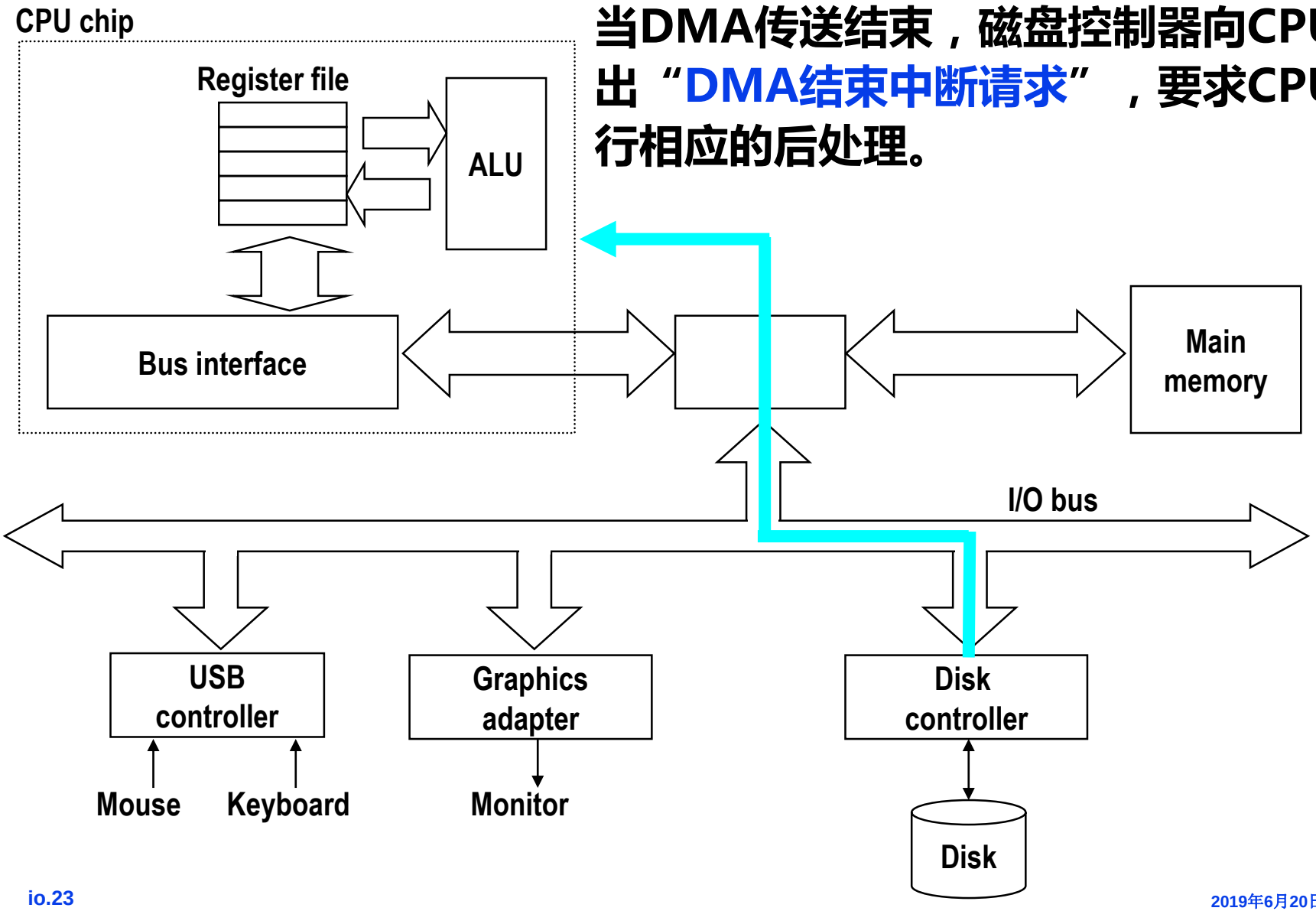


读一个磁盘扇区 - 第二步磁盘控制器读相应的扇区



磁盘控制器读相应的扇区，并按DMA方式把数据送主存

读一个磁盘扇区 - 第三步DMA传送结束要求CPU后处理



当DMA传送结束，磁盘控制器向CPU发出“DMA结束中断请求”，要求CPU进行相应的后处理。

冗余磁盘阵列(RAID)

- 系统总体性能的提高不匹配
  - 处理器和主存性能改进快
  - 辅存性能性能改进慢
- 所用措施：RAID-Redundant Arrays of Inexpensive Disk （磁盘冗余阵列）
- RAID的基本思想：

  将多个独立操作的磁盘按某种方式组织成磁盘阵列(Disk Array)，以增加容量，利用类似于主存中的多体交叉技术，将数据存储在多个盘体上，通过使这些盘并行工作来提高数据传输速度，并用冗余(redundancy)磁盘技术来进行错误恢复(error correction)以提高系统可靠性。
- RAID特性：

  （1）RAID是一组物理磁盘驱动器，在操作系统下被视为一个单逻辑驱动器。

  （2）数据分布在一组物理磁盘上。

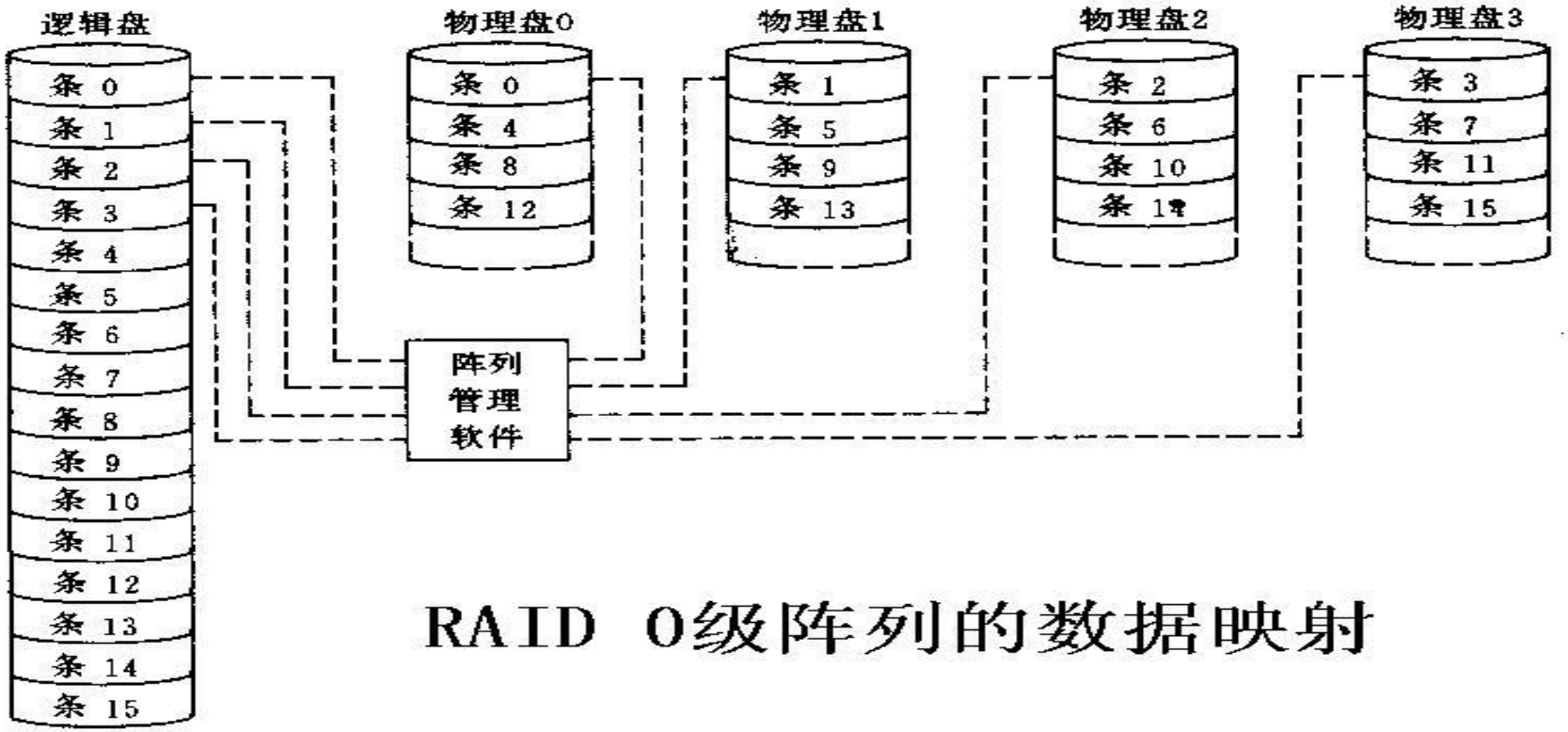
  （3）冗余磁盘用于存储奇偶校验信息，保证磁盘万一损坏时能恢复数据。
- RAID级别
  - 目前已知的RAID方案分为8级（0-7级），以及RAID10（结合0和1级）和RAID30 （结合0和3级）和 RAID50 （结合0和5级） 。但这些级别不是简单地表示层次关系，而是表示具有上述3个共同特性的不同设计结构。

冗余磁盘阵列 (RAID 0)- 无磁盘冗余

- 不遵循特性(3)，无冗余。适用于容量和速度要求高的非关键数据存储的场合
  - 与单个大容量磁盘相比有两个优点：

(1) 连续分布或大条区交叉分布时，如果两个I/O请求访问不同盘上的数据，则可并行发送。减少了I/O排队时间。具有较快的I/O响应能力。

(2) 小条区交叉分布时，同一个I/O请求有可能并行传送其不同的数据块(条区)，因而可达较高的数据传输率。例如，可以用在视频编辑和播放系统



RAID 0级阵列的数据映射

冗余磁盘阵列 (RAID 1)- 1对1磁盘冗余

- 镜像盘实现1对1冗余(100% redundancy)

  （1）读：一个读请求可由其中一个定位时间更少的磁盘提供数据。

  （2）写：一个写请求对对应的两个磁盘并行更新。故写性能由两次中较慢的一次写来决定，即定位时间更长的那一次。

  （3）检错：数据恢复简单。当一个磁盘损坏时，数据仍能从另一个磁盘读取。

  - 特点；可靠性高，但价格昂贵。

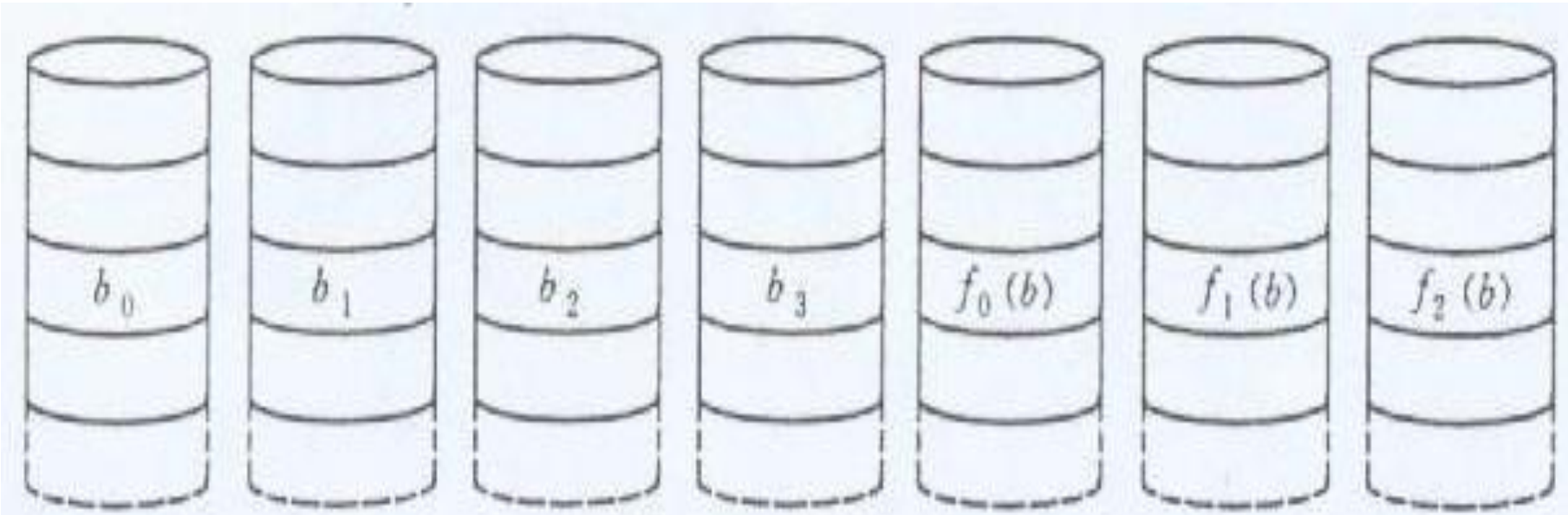
  常用于可靠性要求很高的场合，如系统软件的存储，金融、证券等系统。

(b) RAID 1(镜像)

冗余磁盘阵列（RAID2 ）-已不再使用！

- 用海明校验法生成多个冗余校验盘，实现纠正一位错误、检测两位错误的功能。
- 采用条区交叉分布方式，且条区非常小（有时为一个字或一个字节）。这样，可获得较高的数据传输率，但I/O响应时间差。
- 采用海明码，虽然冗余盘的个数比RAID1少，但校验盘与数据盘成正比。所以冗余信息开销太大，价格贵。
- 读操作性能高（多盘并行）。
- 写操作时要同时写数据盘和校验盘。

RAID2已不再使用！　　为什么？



(c) RAID 2(冗余用于海明码)

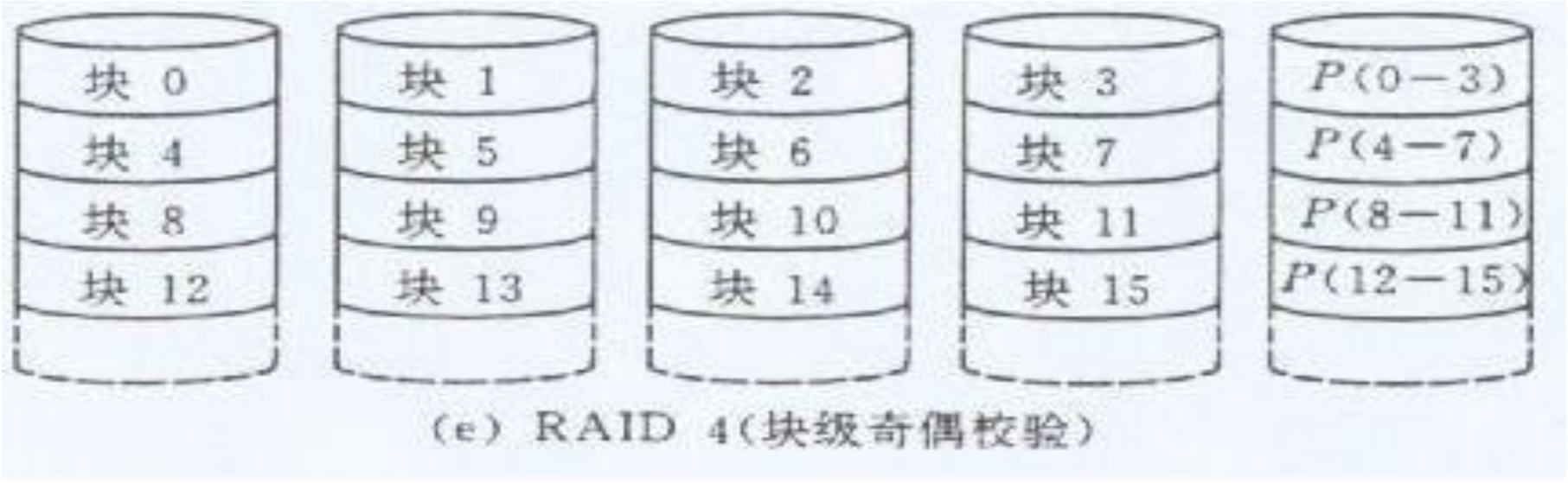
冗余磁盘阵列 (RAID 3)-用奇偶校验法生成单个冗余盘

- 采用奇偶校验法生成单个冗余盘。
- 与RAID 2相同，也采用条区交叉分布方式，并使用小条区。这样，可获得较高的数据传输率，但I/O响应时间差。为什么？
- 用于大容量的 I/O请求的场合，如：图像处理、CAD 系统中。
- 某个磁盘损坏但数据仍有效的情况，称为简化模式。此时损坏的磁盘数据可以通过其它磁盘重新生成。数据重新生成非常简单，这种数据恢复方式同时适用于RAID3、4、5级。

(d) RAID 3(位交错奇偶校验)

冗余磁盘阵列（RAID 4）-用一个冗余盘存放相应块的奇偶校验位

- 用一个冗余盘存放相应块（块：较大的数据条区）的奇偶校验位。
- 采用独立存取技术，每个磁盘的操作独立进行，所以，可同时响应多个I/O请求。因而它适合于要求I/O响应速度块的场合。
- 对于写操作，校验盘成为I/O瓶颈，因为每次写都要对校验盘进行。
  - 少量写（只涉及个别磁盘）时，有“写损失”，因为一次写操作包含两次读和两次写
  - 大量写（涉及所有磁盘的数据条区）时，则只需直接写入奇偶校验盘和数据盘。因为奇偶校验位可全部用新数据计算得到。而无须读原数据



(e) RAID 4(块级奇偶校验)

RAID4的少量写—1个奇偶校验盘

假定考虑一个有5个磁盘的阵列，X_0到X_3保存数据，P是奇偶校验盘，初始时对每位i有下列关系式：

$$p(i)= X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i)$$

假定仅$X_0(i)$改为$X'_0(i)$，其余不变，则新校验位为：

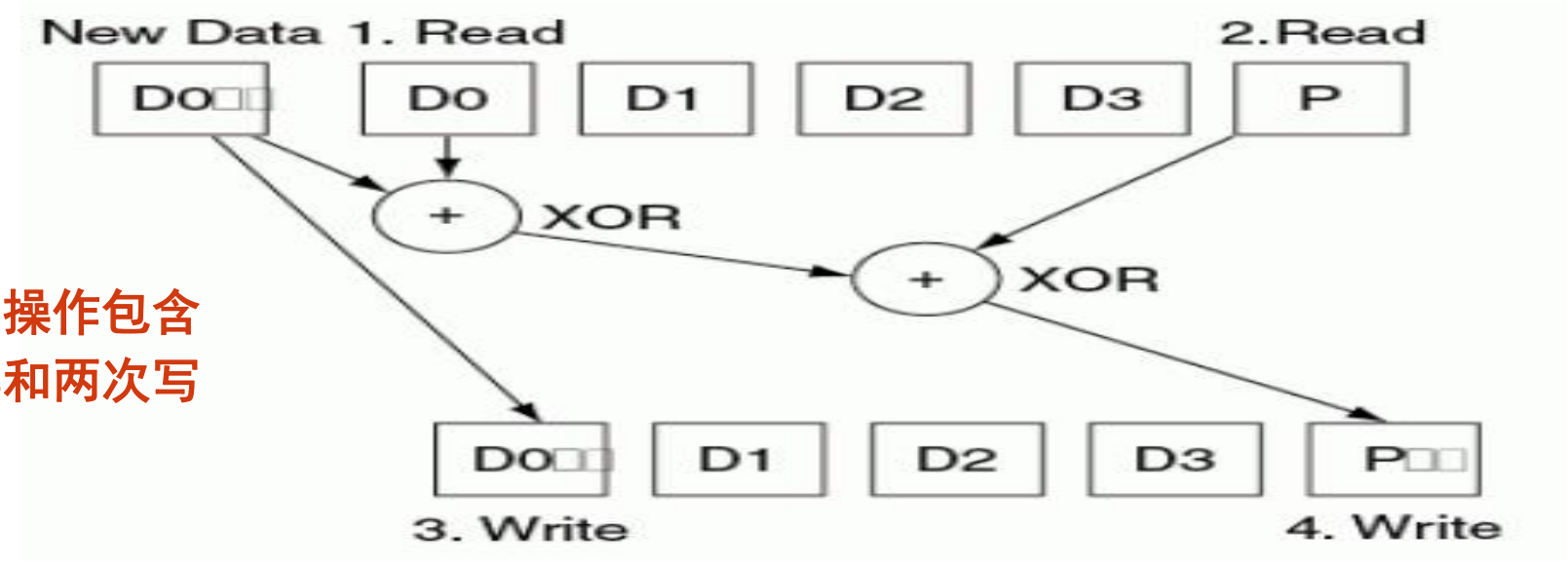
$$P'(i) = X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X'_0(i)$$

$$= X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X'_0(i) \oplus (X_0(i) \oplus X_0(i))$$

化简后得到： $p'(i)= p(i) \oplus (X_0(i) \oplus X'_0(i))$

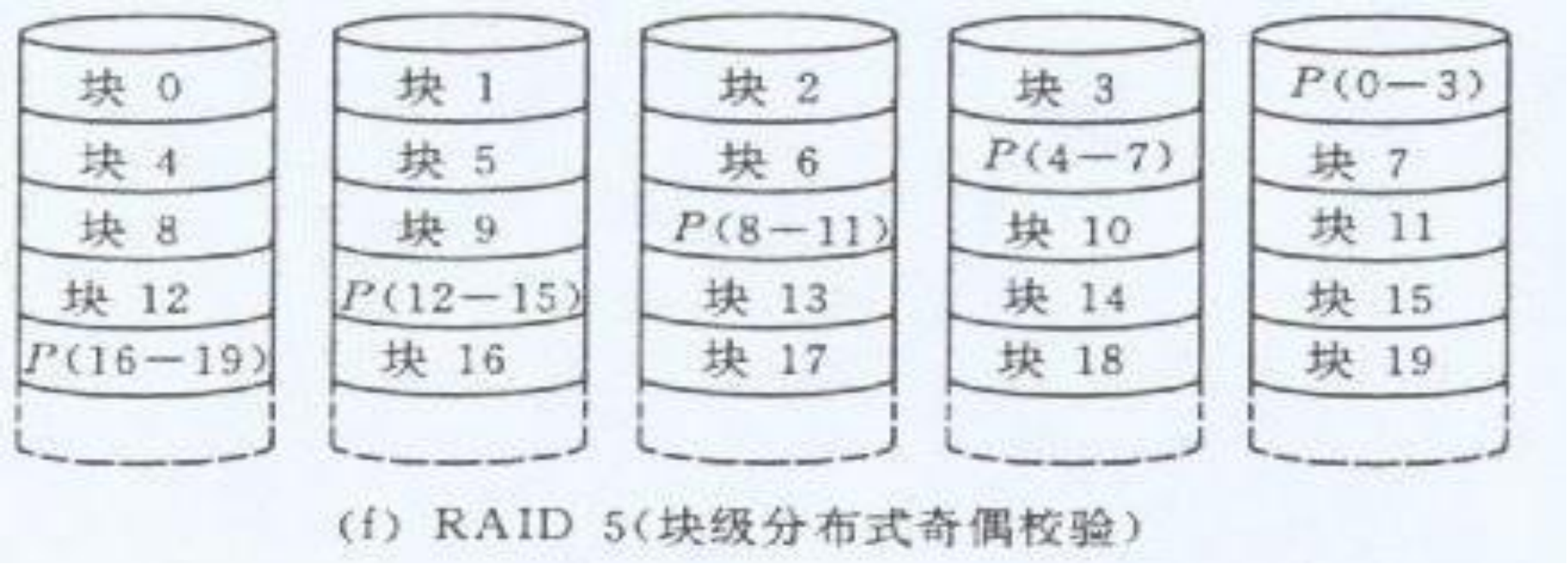
由此可见，要更新一个$X_0(i)$，必须先读$p(i)$和$X_0(i)$，然后写$X'_0(i)$和$p'(i)$

一次写操作包含
两次读和两次写



冗余磁盘阵列 (RAID 5)-奇偶校验块分布在各个磁盘中

- 与RAID 4的组织方式类似，只是奇偶校验块分布在各个磁盘中，所以，所有磁盘的地位等价，这样可提高容错性，并且避免了使用专门校验盘时潜在的I/O瓶颈。
- 与RAID 4一样，采用独立的存取技术，因而有较高的I/O响应速度。
- 小数据量的操作可以多个磁盘并行操作。
- 成本不高但效率高，所以，被广泛使用。



(f) RAID 5(块级分布式奇偶校验)

P块为校验块，分布在不同的磁盘中

冗余磁盘阵列 (RAID 6)

- 冗余信息均匀分布在所有磁盘上，而数据仍以块交叉方式存放
- 双维块交叉奇偶校验独立存取盘阵列，容许双盘出错
- 它是对RAID 5的扩展，主要是用于要求数据绝对不能出错的场合
- 由于引入了第二种奇偶校验值，对控制器的设计变得十分复杂，写入速度也比较慢，用于计算奇偶校验值和验证数据正确性所花费的时间比较多
- RAID 6级以增大开销的代价保证了高度可靠性

RAID 6 级双维块交叉奇偶校验独立存取盘阵列示意图

P_0代表第0条区的奇偶校验值，而P_A代表数据块A的奇偶校验值

冗余磁盘阵列 (RAID 7)

- 带Cache的盘阵列
- 在RAID6的基础上，采用Cache技术使传输率和响应速度都有较大提高
- Cache分块大小和磁盘阵列中数据分块大小相同，一 一对应
- 有两个独立的Cache，双工运行。在写入时将数据同时分别写入两个独立的Cache，这样即使其中有一个Cache出故障，数据也不会丢失
- 写入磁盘阵列以前，先写入Cache中。同一磁道的信息在一次操作中完成
- 读出时，先从Cache中读出，Cache中没有要读的信息时，才从RAID中读

Cache和RAID技术结合，弥补了RAID的不足（如：分块的写请求响应性能差等），从而以高效、快速、大容量、高可靠性，以及灵活方便的存储系统提供给用户

固态硬盘（SSD）

- 固态硬盘（Solid State Disk，简称SSD）也被称为电子硬盘。
- 它并不是一种磁表面存储器，而是一种使用NAND闪存组成的外部存储系统，与U盘并没有本质差别，只是容量更大，存取性能更好。
- 它用闪存颗粒代替了磁盘作为存储介质，利用闪存的特点，以区块写入和抹除的方式进行数据的读取和写入。
- 写操作比读操作慢得多。
- 电信号的控制使得固态硬盘的内部传输速率远远高于常规硬盘。
- 其接口规范和定义、功能及使用方法与传统硬盘完全相同，在产品外形和尺寸上也与普通硬盘一致。目前接口标准上使用USB、SATA和IDE，因此SSD是通过标准磁盘接口与I/O总线互连的。
- 在SSD中有一个闪存翻译层，它将来自CPU的逻辑磁盘块读写请求翻译成对底层SSD物理设备的读写控制信号。因此，这个闪存翻译层相当于磁盘控制器。
- 闪存的擦写次数有限，所以频繁擦写会降低其写入使用寿命。

第二讲 总线类型、系统中的设备间的互连及I/O接口

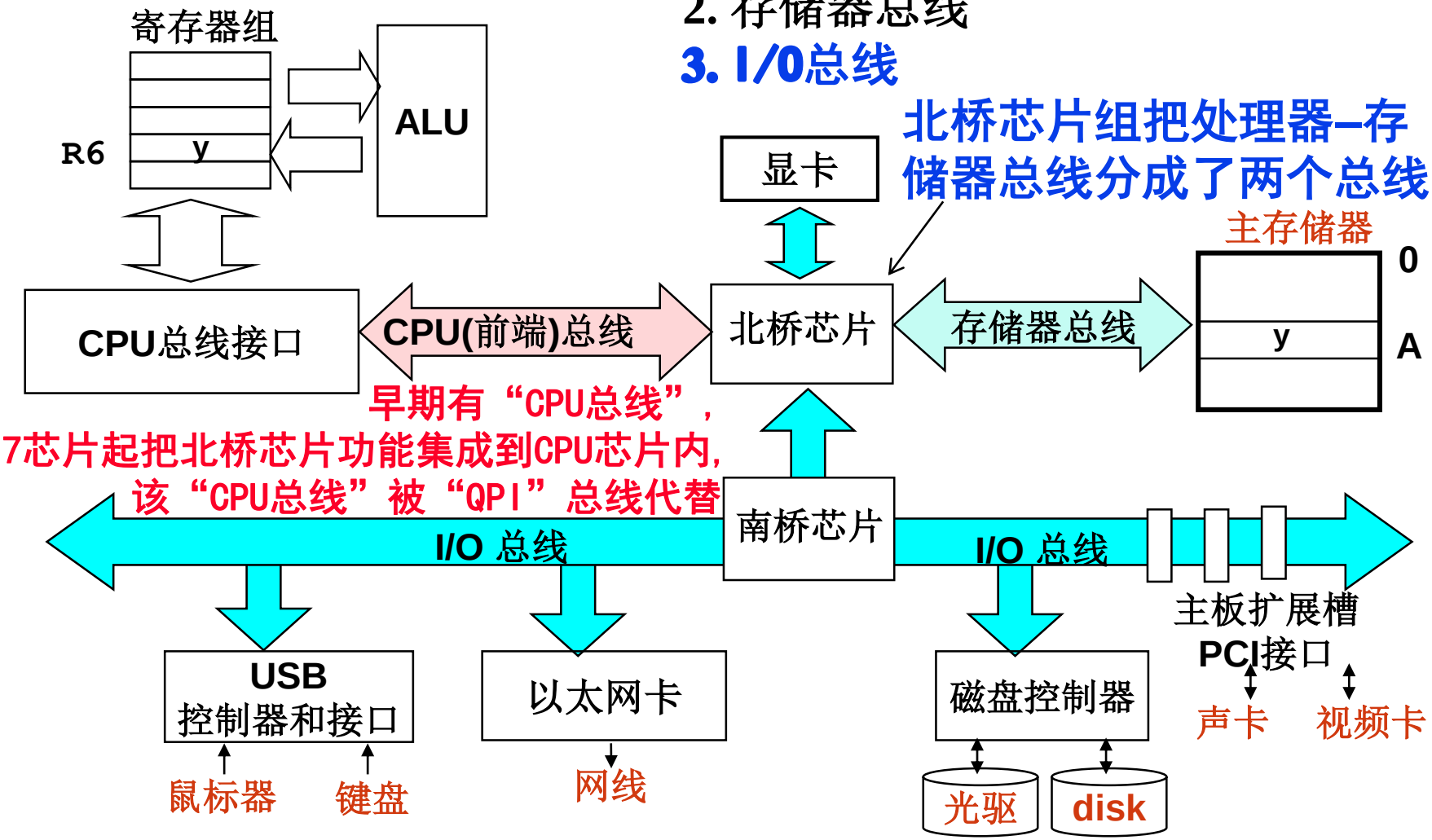
主 要 内 容

- 总线概述及其分类
- 计算机系统中基于(不同)总线的互连结构
- I/O接口
  - I/O接口的分类
  - I/O控制器的结构
  - I/O控制器的职能
  - I/O端口的概念
  - I/O设备的寻址

计算机系统的总线与I/O控制器及I/O设备的关系

本章主要介绍的系统总线指：

1. 处理器总线(系统总线，前端总线）
2. 存储器总线
3. I/O总线



系统总线上传输的信息有：数据（指令、操作数、中断号）、地址、其他控制/状态/定时等信号！

总线的分类

- 总线在各层次上提供部件之间的连接和交换信息通路
- 分为以下几类：
  - 芯片内总线：在芯片内部各元件之间提供连接
    - 例如，CPU芯片内部，各寄存器、ALU、指令部件等之间有总线相连
  - 系统总线：在系统主要功能部件（CPU 、MM和各种I/O控制器）间提供连接
    - 单总线结构
      - 将CPU、MM和各种I/O适配卡通过底板总线(Backplane Bus)互连，底板总线为标准总线(Industry standard)
    - 多总线结构
      - 将CPU、Cache、MM和各种I/O适配卡用局部总线、处理器-主存总线、高速I/O总线、扩充I/O总线等互连。主要有两大类：

        Processor- Memory Bus (Design specific or proprietary)

        » 短而快，仅需与内存匹配，使CPU-MM之间达最大带宽

        I/O Bus (Industry standard)

        » 长而慢，需适应多种设备，一侧连接到Processor- Memory Bus或 Backplane Bus，另一侧连到I/O控制器

    （注：Intel公司在推出845、850等芯片组时，对“System Bus”有专门的定义，将处理器总线称为前端总线(Front Bus)或系统总线）
  - 通信总线：在主机和I/O设备之间或计算机系统之间提供连接
    - 通常是电缆式总线，如SCSI、RS-232、USB、PS-2等

系统总线的组成

- 系统总线通常由一组控制线、一组数据线和一组地址线构成。也有些总线没有单独的地址线，地址信息通过数据线来传送，这种情况称为数据/地址复用。
  - 数据线（Data Bus）：承载在源和目部件之间传输的信息。数据线的宽度反映一次能传送的数据的位数。
  - 地址线（Address Bus）：给出源数据或目的数据所在的主存单元或I/O端口的地址。地址线的宽度反映最大的寻址空间。
  - 控制线（Control Bus）：控制对数据线和地址线的访问和使用。用来传输定时信号和命令信息。典型的控制信号包括：
    - 时钟（Clock）：用于总线同步。
    - 复位（Reset）：初始化所有设备。
    - 总线请求（Bus Request）：表明发出该请求信号的设备要使用总线。
    - 总线允许（Bus Grant）：表明接收到该允许信号的设备可以使用总线。
    - 中断请求（Interrupt Request）：表明某个中断正在请求。
    - 中断回答（Interrupt Acknowledge）：表明某个中断请求已被接受。
    - 存储器读（memory read）：从指定的主存单元中读数据到数据总线上。
    - 存储器写（memory read）：将数据总线上的数据写到指定的主存单元中。
    - I/O读（I/O read）：从指定的I/O端口中读数据到数据总线上。
    - I/O写（I/O Write）：将数据总线上的数据写到指定的I/O端口中。
    - 传输确认（transmission Acknowledge）：表示数据已被接收或已送总线

基本概念

- 总线裁决

  早期：总线多是共享传输，需确定哪个设备使用总线。

  现在：总线多是点对点传输，无需裁决。

- 总线定时

  定义总线事务中的每一步何时开始、何时结束。

  Synchronous (同步)：用时钟信号来确定每个步骤

  Asynchronous(异步)：用握手信号来定时，前一个信号结束就是下一个信号的开始

  半同步：结合使用时钟信号和握手信号来定时

- 并行/串行传输

  并行传输：一个方向同时传输多位数据信号，故位与位需同步，慢！

  串行传输：一个方向只传输一位数据信号，无需在位之间同步，快！

现在总线设计的趋势是：点对点、异步、串行

总线的性能指标

总线宽度

- 总线中数据线的条数，决定了每次能同时传输的信息位数。

总线工作频率

- 总线工作频率=次传送/秒=总线时钟频率F/ 完成一次数据传送所用时钟周期数N
- 早期的总线通常一个时钟周期传送一次数据，此时，工作频率等于总线时钟频率
- 现在有些总线一个总线时钟周期可以传送2次或4次数据，因此，工作频率是时钟频率的2倍或4倍。QPI总线工作频率是其总线时钟频率的2倍.

总线带宽: 总线的最大数据传输率

- 对于同步总线，总线带宽计算公式： B=(总线宽度)*总线工作频率= W×(F/N)
- W-总线宽度；F-总线时钟频率；N-完成一次数据传送所用时钟周期数。
- F/N实际上就是总线工作频率

总线传送方式

- 非突发传送：每个总线事务都传送地址，一个地址对应一次数据传送。
- 突发传送：即为成块数据传送。突发传送总线事务中，先传送一个地址，后传送多次数据，后续数据的地址默认为前面地址自动增量。

CPU(处理器)总线

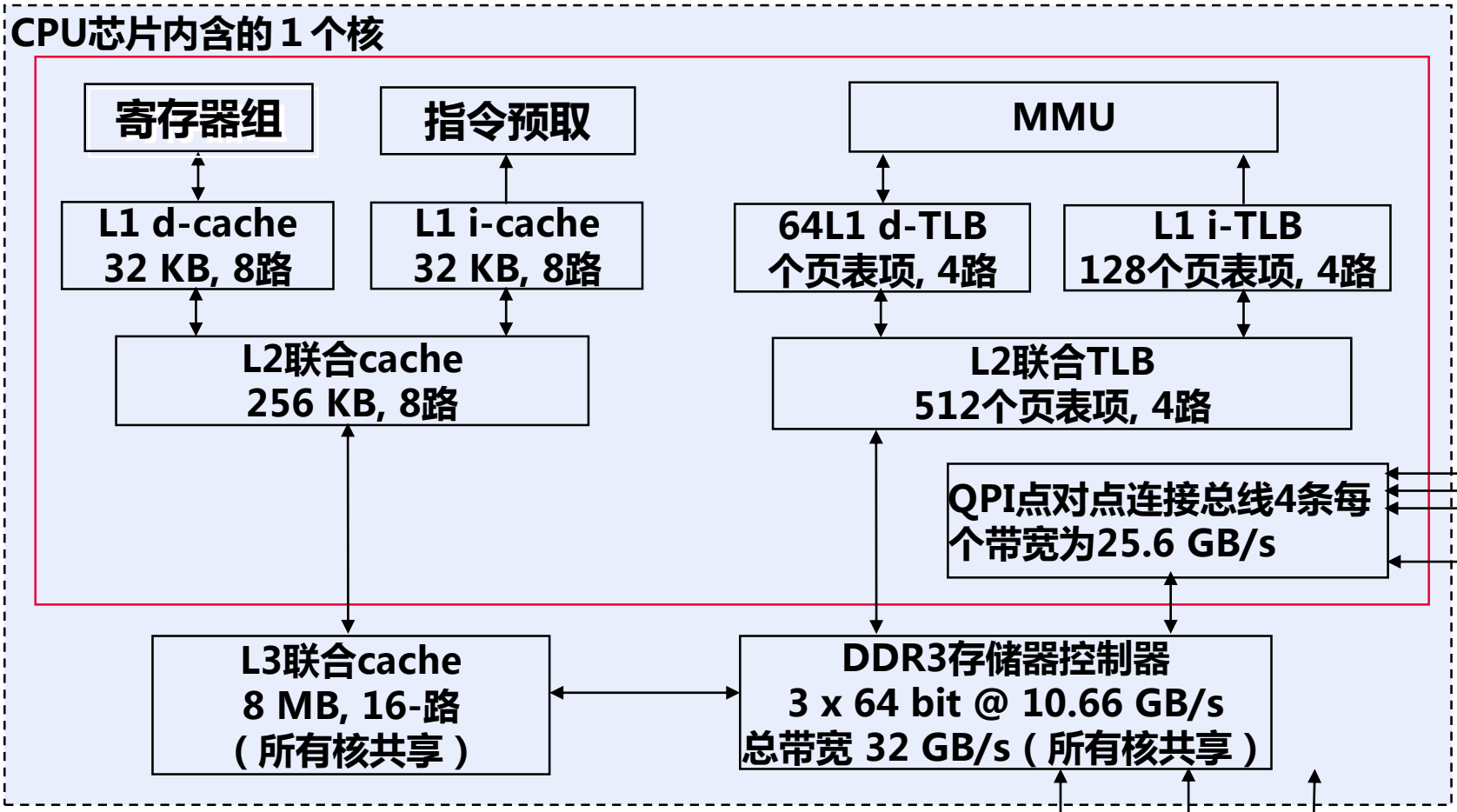
(早期Intel架构使用)前端总线（Front Side Bus，FSB）

- 位于CPU芯片与北桥芯片之间互连
- 并行传输、同步定时方式
- 从Pentium Pro开始，FSB采用quad pumped技术：每个总线时钟周期传送4次数据。
- 若工作频率为1333MHz（实际单位应是MT/s，表示每秒传送1333M次数据，实际时钟频率为333MHz），总线宽度为64位，则总线带宽为1333MT/s×8B=10.5GB/s。

(2008年i7CPU芯片起)QPI（Quick Path Interconnect,快速通道相联)总线

- 能够每个时钟周期传两次数据,在Intel CPU芯片内部核之间、CPU芯片之间、CPU芯片与IOH（I/O Hub）芯片之间，都通过QPI总线互连,
- 是基于包交换的串行、高速点对点连接：发送方和接收方各有时钟信号，双方可以同时传输数据(各有20条数据线),每个QPI数据包含80位，分两个时钟周期传送，单向每个时钟周期传两次，每次传20位(16位数据+4位校验位),QPI总线(传送数据)(最大)带宽为:(每秒传送次数T/s)×(2B数据/1次传送)×2(表示双向)
- QPI总线的速度单位及工作频率为GT/s，表示每秒传送多少G次。若QPI总线时钟频率为2.4GHz，每个总线时钟周期传2次, 则速度为4.8GT/s，(传送数据)(最大)带宽为 4.8G×2B×2 = 19.2GB/s.

存储器总线



单个存储器总线64位(=8B)宽,速度为1333MT/s，

每秒传送1333M次数据,3通道(3个存储器总线)总带宽为：3x(8B)x1333M = 32GB/s.

从Core i7 CPU芯片开始，北桥在该芯片内，CPU通过存储器总线(即内存条插槽，图中为三通道插槽)直接和内存条相连。3个存储器控制器(存控)包含在CPU芯片内

I/O总线

I/O总线用于为系统中的各种I/O设备提供输入输出通道

I/O总线在物理上可以是主板上的I/O扩展槽，如：

第一代：ISA/EISA总线、VESA总线，早被淘汰

第二代：PCI(Peripheral Component Interconnect)、AGP、PCI-X，被逐渐淘汰

第三代：PCI-Express（串行总线，主流总线）

PCI-Express总线

两个PCI-Express设备之间以一个链路（link）相连

每个链路包含多条通路（lane），可以是1,2,4,8,16或32条

PCI-Express×n表示一个具有n条通路的PCI-Express链路

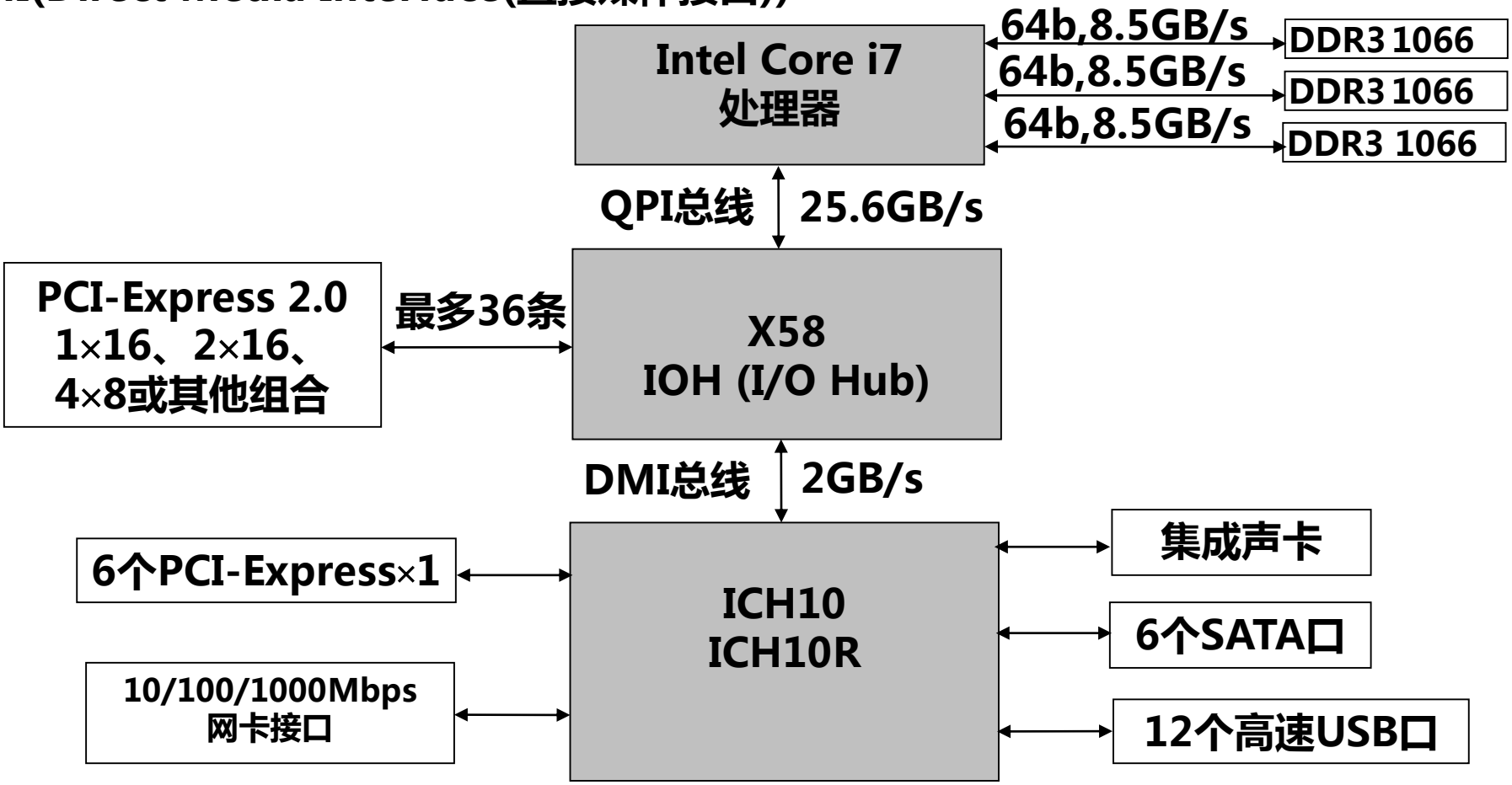
每条通路可同时发送和接受，每个数据字节被转换为10位信息被传输

PCI-Express1.0下，每条通路的发送和接受速率都是2.5Gb/s，故PCI-Express×n的带宽为：2.5Gb/s×2×n/10=0.5GB/s×n。

PCI-Express1.0下，PCI-Express×2的带宽为1GB/s，PCI-Express×4的带宽为2GB/s，PCI-Express×16的带宽为8GB/s.

基于Core i7系列处理器的互连结构举例

DMI(Direct Media Interface(直接媒体接口))



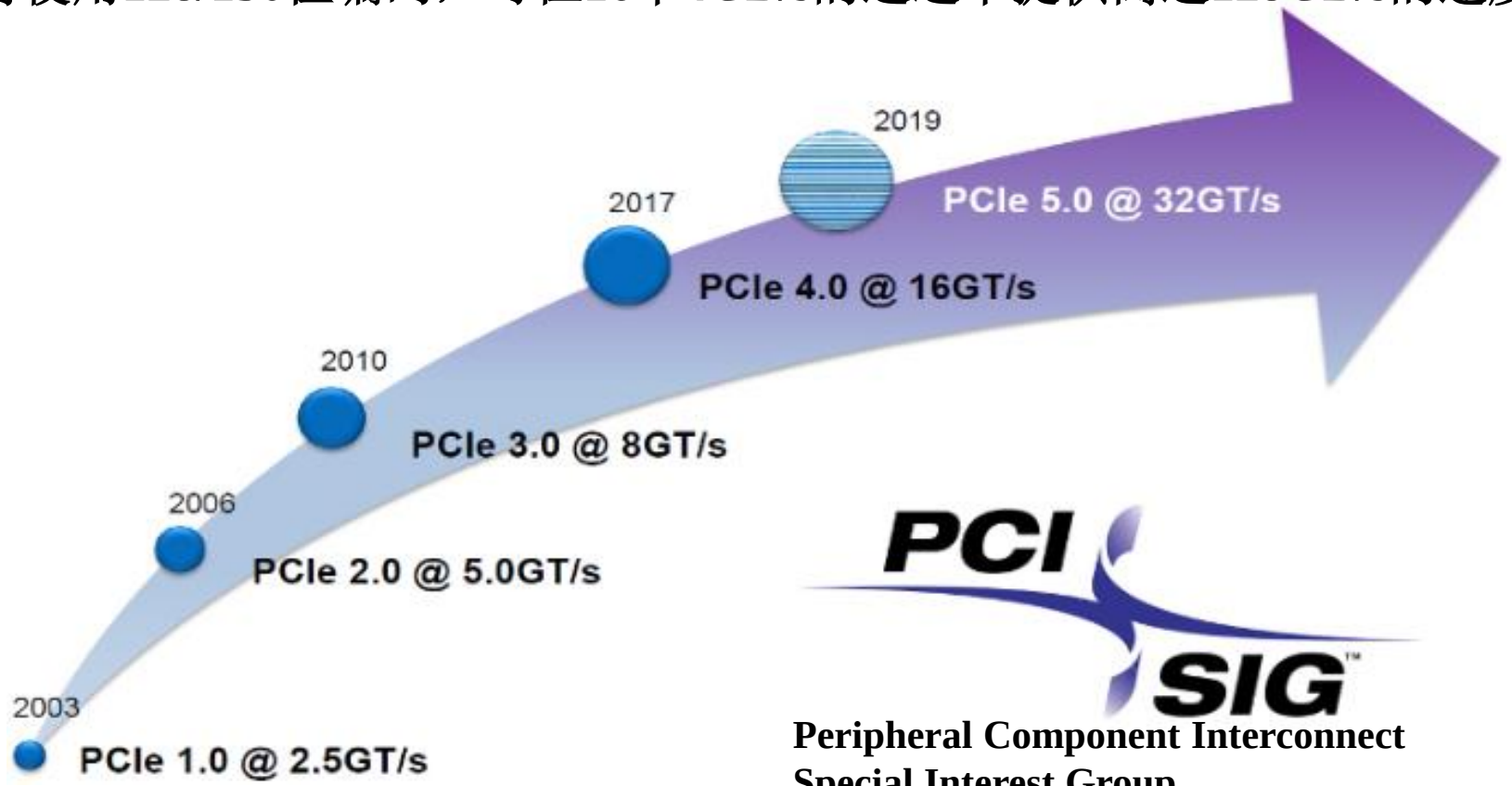
QPI总线的带宽为：6.4GT/s×2B×2=25.6GB/s

每个存储器总线的带宽为：(64b/8)×(1066 MT/s) = 8.5 GB/s .

各代PCI-E工作频率及PCI-E 5.0的特点

PCI-E 5.0的特点

- **利用及依据 PCIe 4.0 规范的基础，通过扩展性提供更高的速度**
- **实现用电量的变化，以改善连接器的信号完整性和产品的性能**
- **针对 AIC 附加卡提供全新向后兼容的 CEM 连接器**
- **保持与 PCIe 4.0、3.x、2.x 及1.x 规范的向后兼容性**
- **将使用128/130位编码，可在16个4GB/s的通道中提供高达128GB/s的速度**

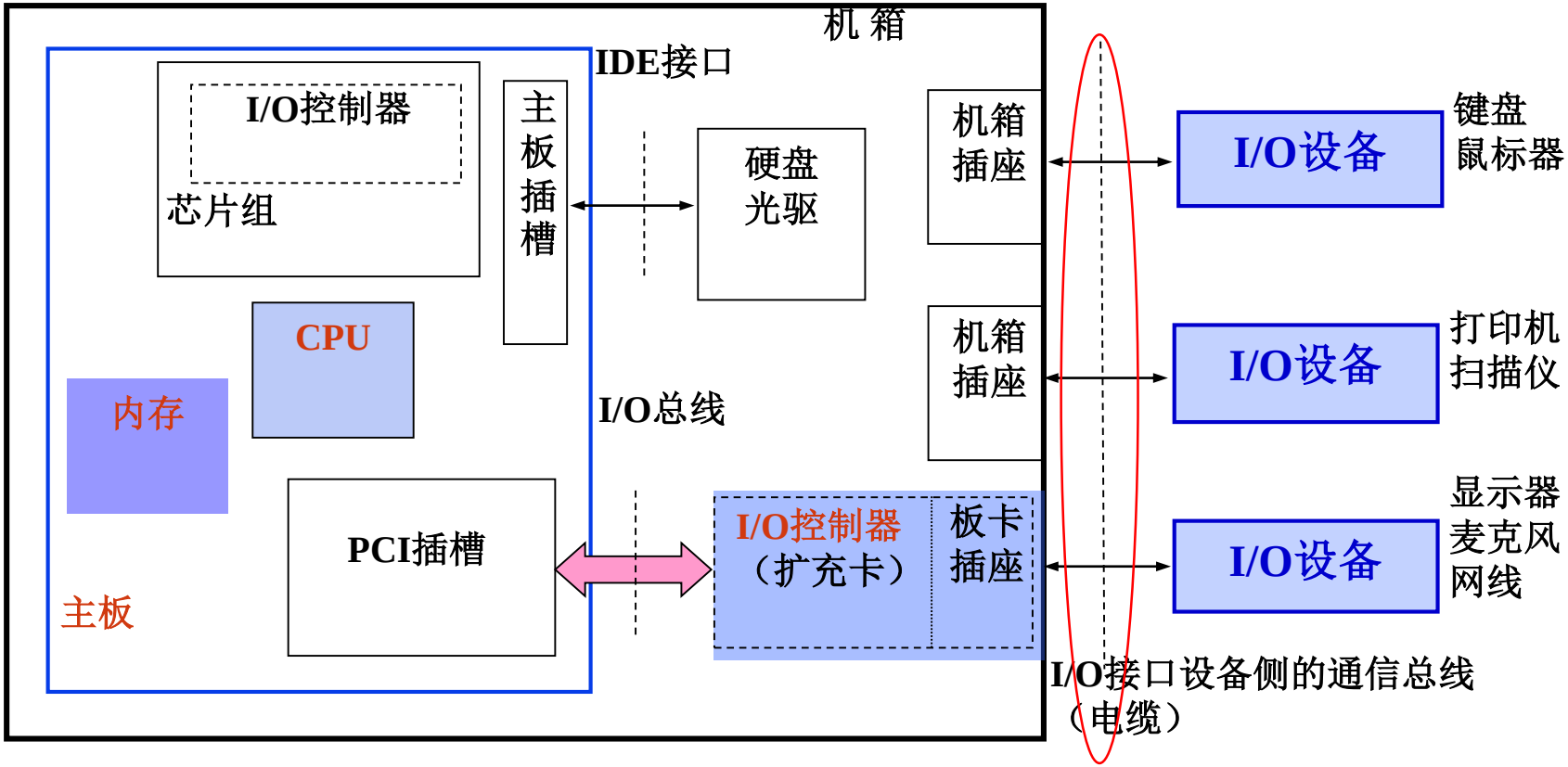


PCI SIG™

Peripheral Component Interconnect Special Interest Group

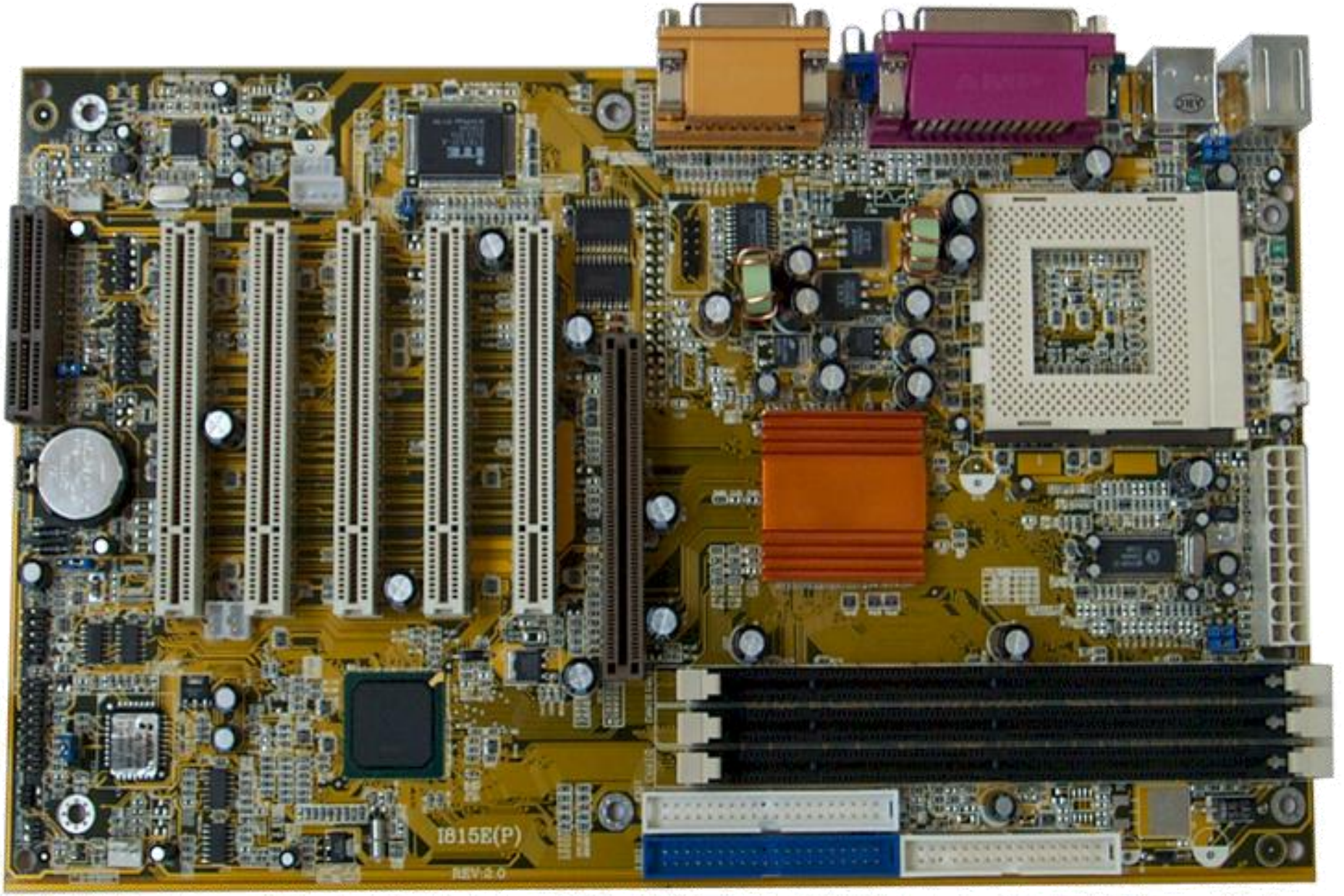
I/O总线,I/O控制器与I/O设备的关系

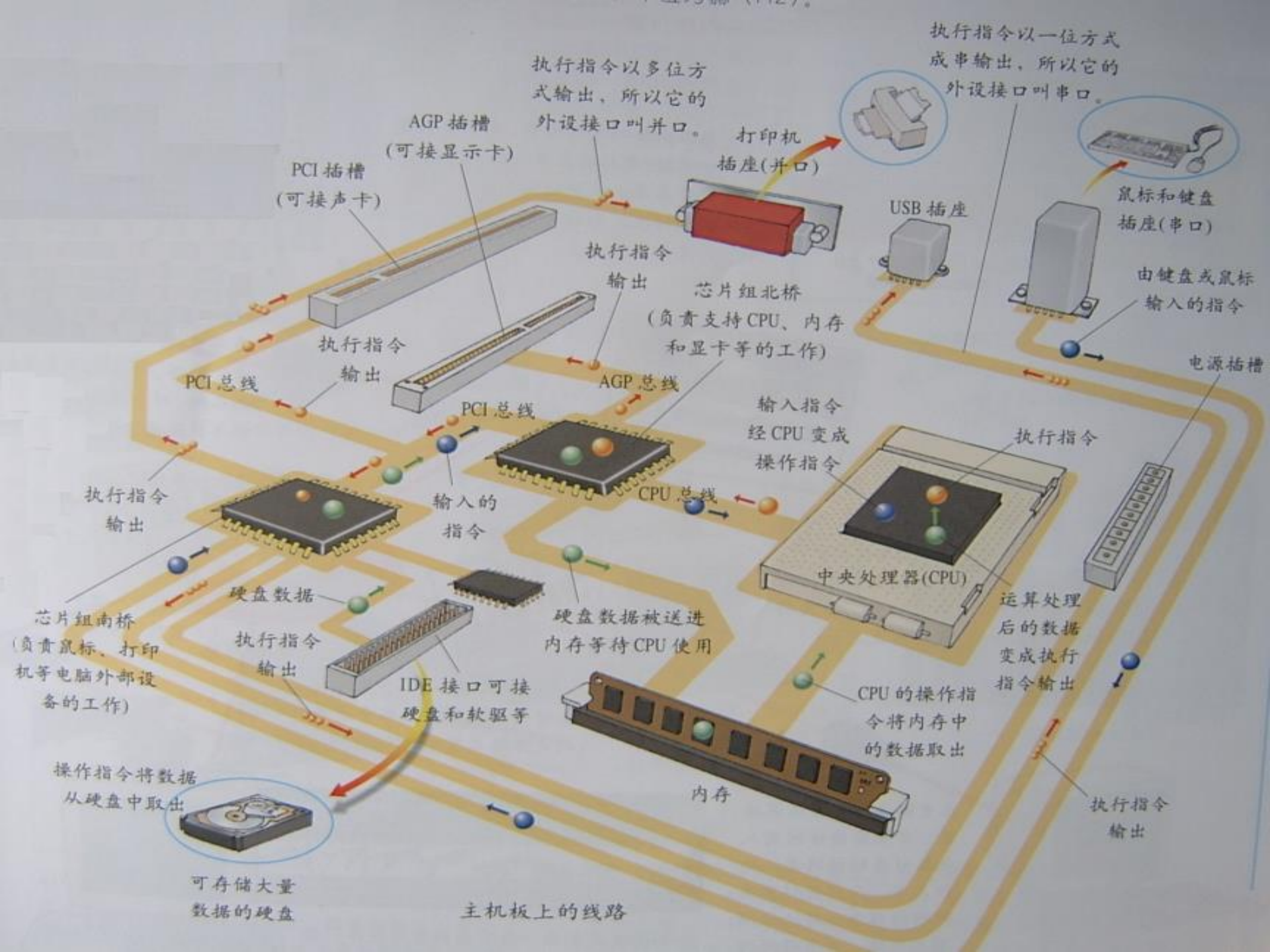
IDE(Integrated Drive Electronics)，它把控制器与盘体集成在一起的硬盘驱动器，IDE是表示硬盘的传输接口，是通信总线还是I/O总线？通信！



- I/O设备通常是物理上相互独立的设备，它们一般通过通信总线与I/O控制器连接
- I/O控制器（I/O接口）通过扩展卡或者南桥芯片与I/O总线连接
- I/O总线经过北桥芯片与内存、CPU连接

主板上的扩充插槽都是一种总线插座



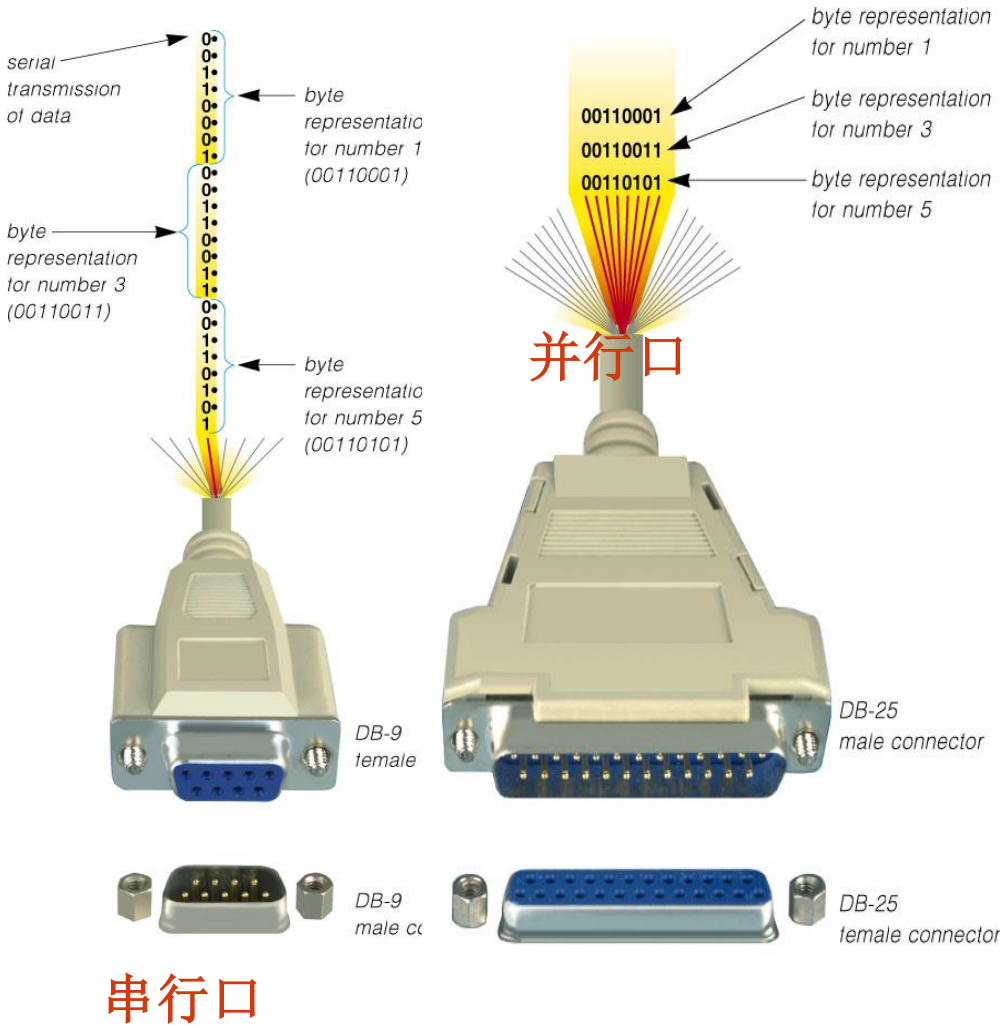

执行指令以一位方式成串输出，所以它的外设接口叫串口。
执行指令以多位方式输出，所以它的外设接口叫并口。
AGP 插槽
(可接显示卡)
打印机
插座(并口)
PCI 插槽
(可接声卡)
USB 插座
鼠标和键盘
插座(串口)
执行指令
输出
芯片组北桥
(负责支持CPU、内存和显卡等的工作)
由键盘或鼠标输入的指令
执行指令
输出
PCI 总线
AGP 总线
电源插槽
PCI 总线
输入指令
经CPU 变成
操作指令
执行指令
执行指令
输出
输入的
指令
CPU 总线
中央处理器(CPU)
硬盘数据
芯片组南桥
(负责鼠标、打印机等电脑外部设备的工作)
硬盘数据被送进内存等待CPU 使用
运算处理后的数据变成执行指令输出
执行指令
输出
IDE 接口可接硬盘和软驱等
CPU 的操作指令将内存中的数据取出
内存
操作指令将数据从硬盘中取出
执行指令
输出
可存储大量数据的硬盘
主机板上的线路

关于I/O接口

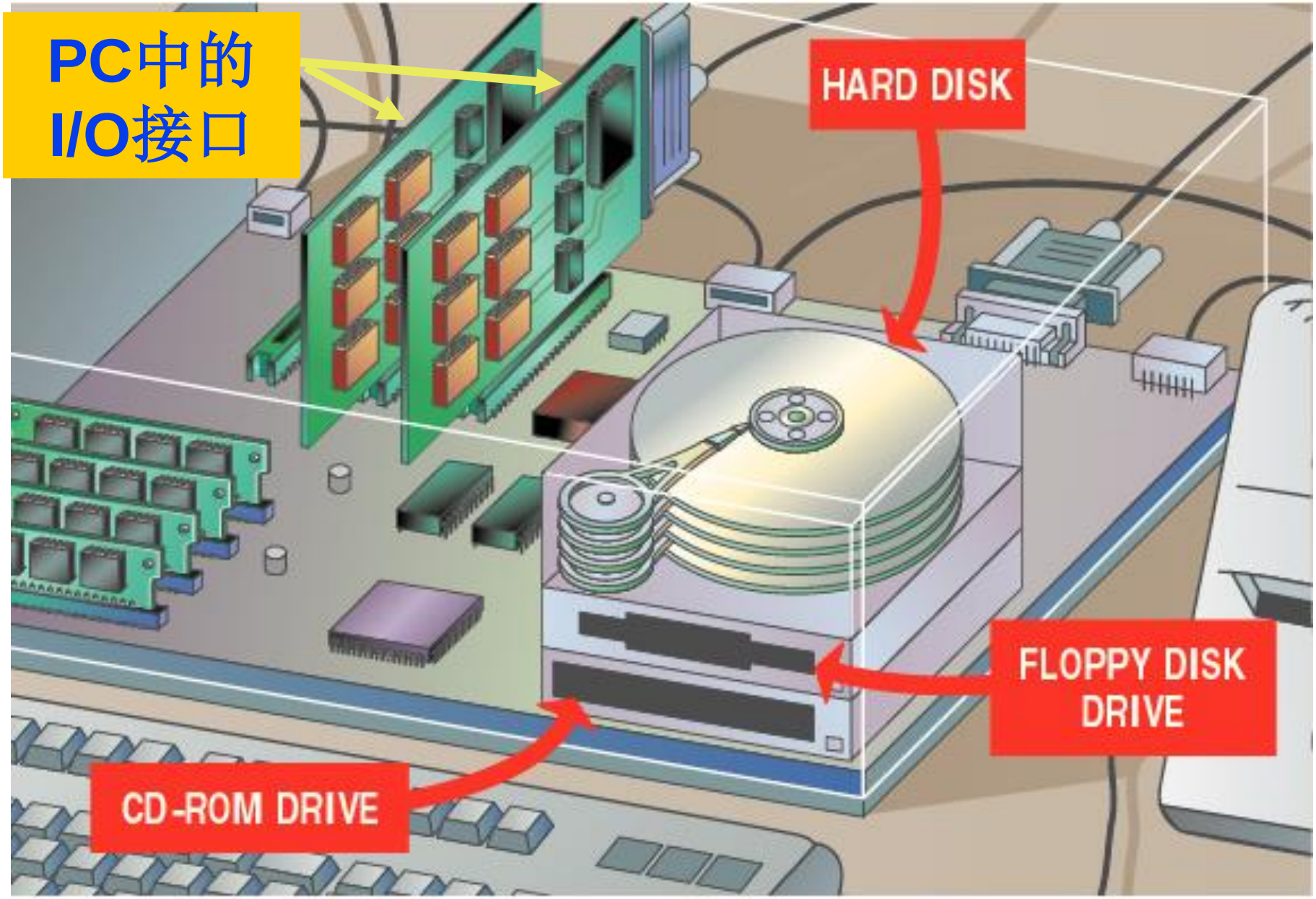
- I/O接口：I/O设备控制器及其插座（如网卡、显卡、键盘适配器、磁盘控制器）

  包括：插头 / 插座的形式、通讯规程和电器特性等

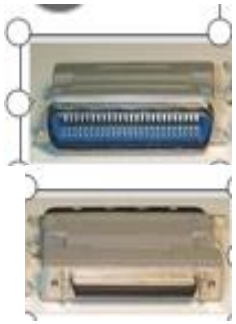
- 分类：
  - 从数据传输方式来分：
    - 串行（一次只传输1位）
    - 并行（多位一起进行传输）
  - 从是否能连接多个设备来分：
    - 总线式（可连接多个设备）
    - 独占式（只能连接1个设备）
  - 从是否符合标准来分：
    - 标准接口 （通用接口）
    - 专用接口 （专用接口）
  - 按功能选择的灵活性来分：
    - 可编程接口
    - 不可编程接口



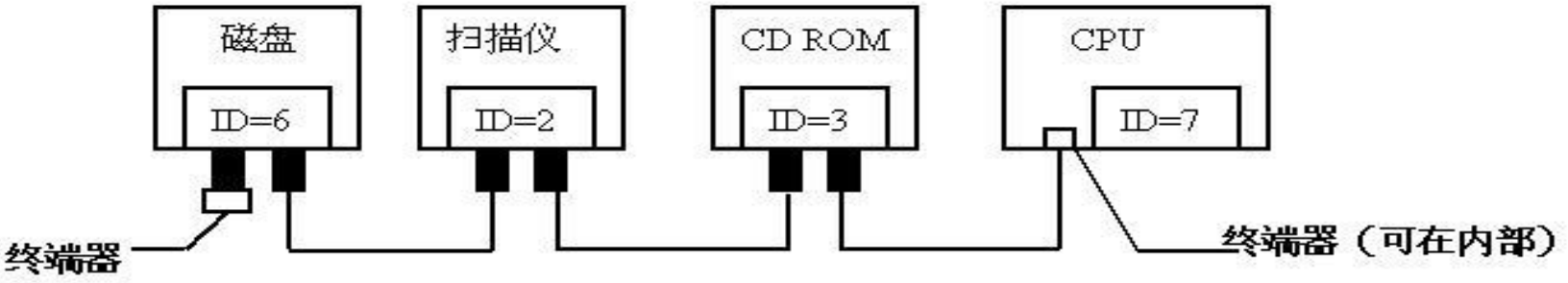
PC中I/O接口



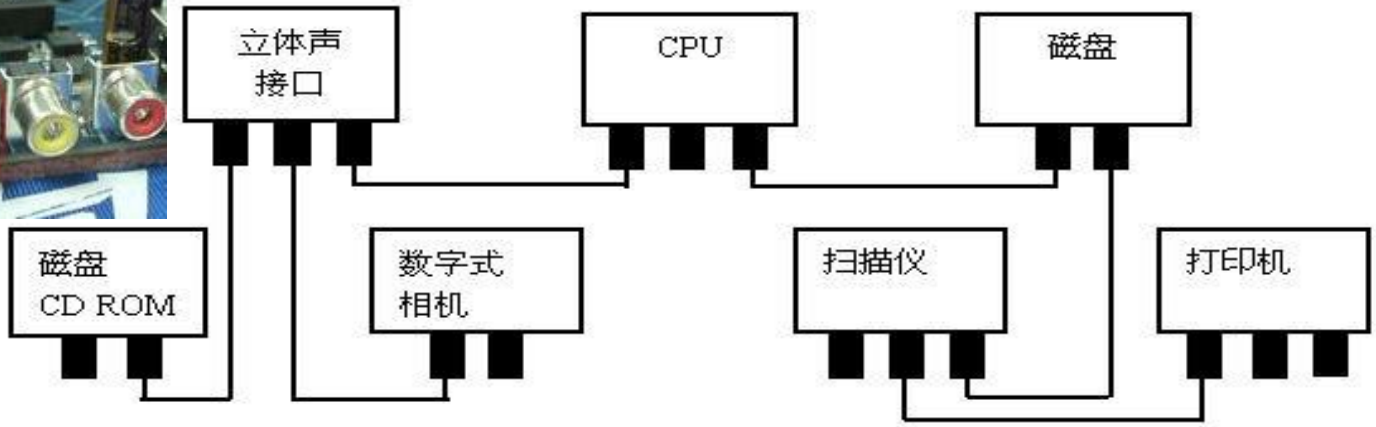
总线式I/O接口-SCSI小型计算机系统接口及P1394串行接口



SCSI(Small Computer System Interface).一种用于计算机和智能设备之间（硬盘、软驱、光驱、打印机、扫描仪等）系统级接口的独立处理器标准。

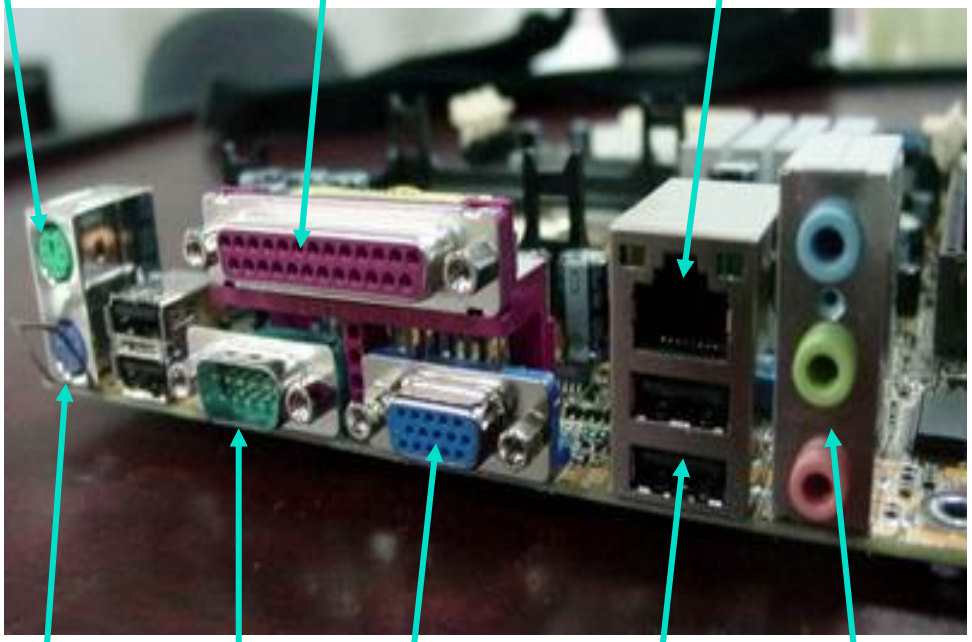


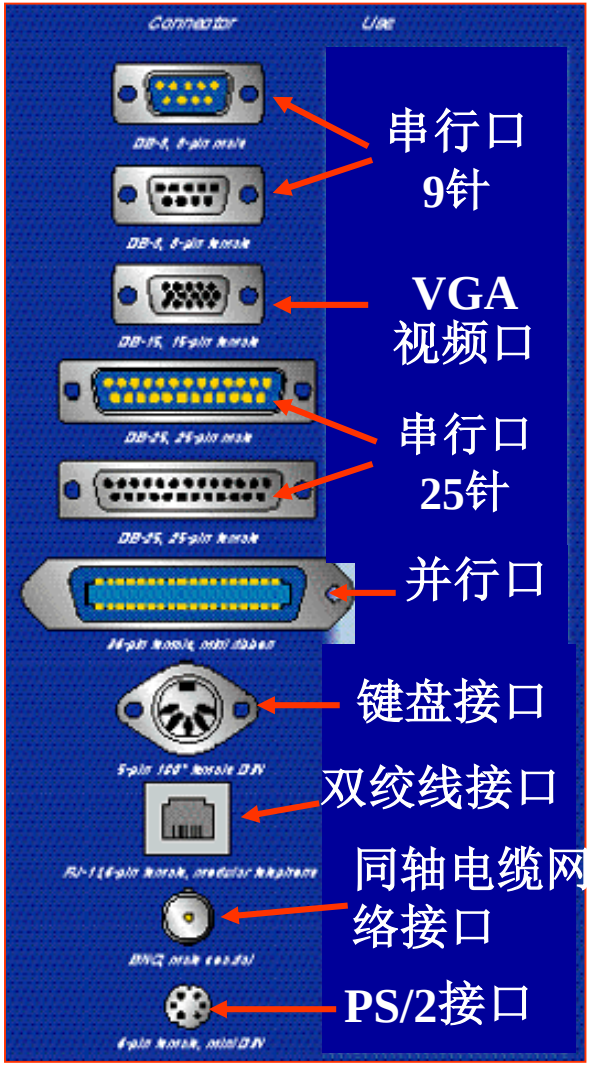
（ a ）SCSI 配置举例



（ b ）P1394 配置举例

I/O设备接口插座（连接器）

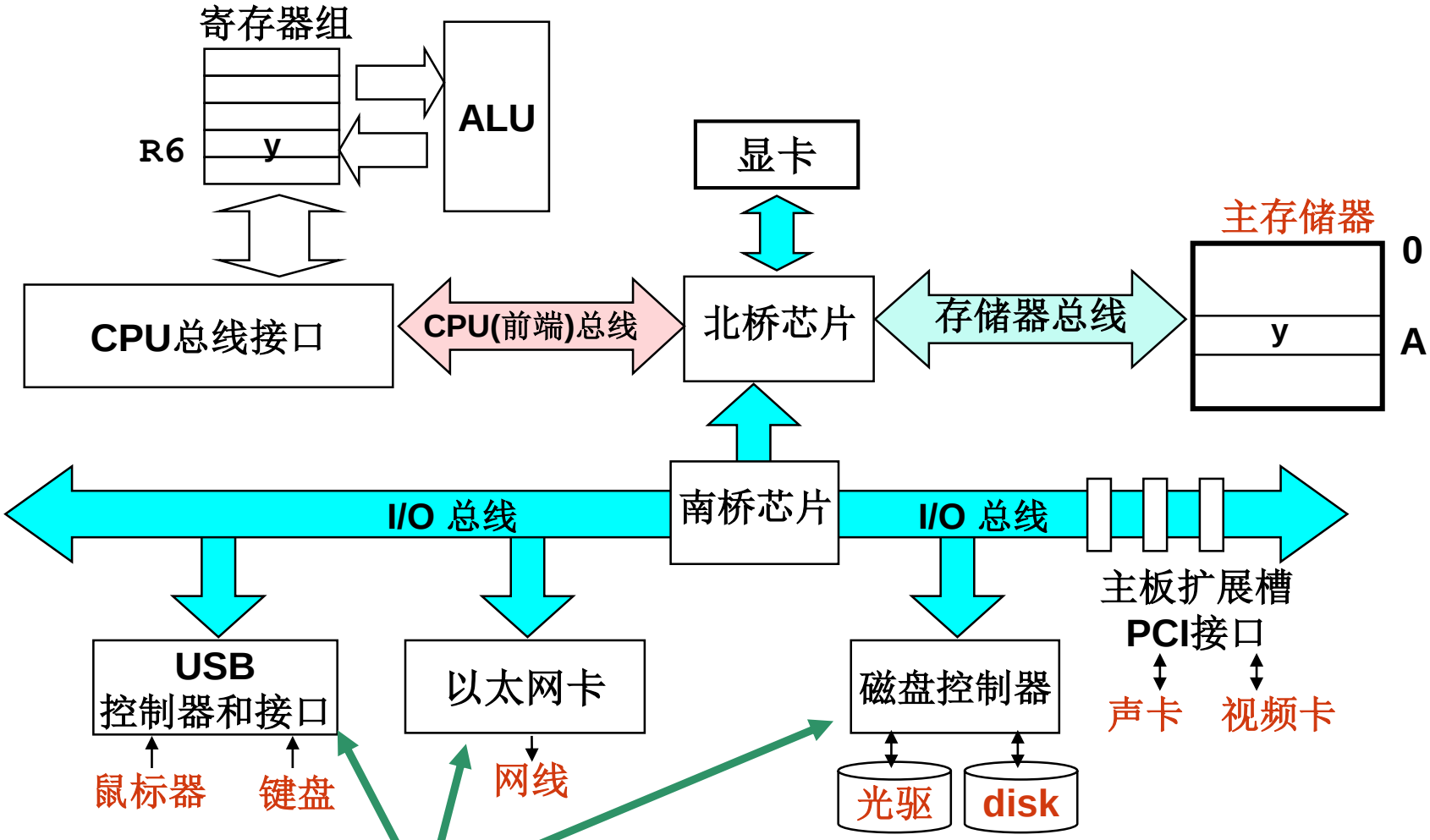




(安装在主板上的I/O设备接口插座)

回顾： I/O总线、I/O接口与I/O设备的关系

I/O接口: 本课程把I/O控制器和插座合起来称为I/O接口



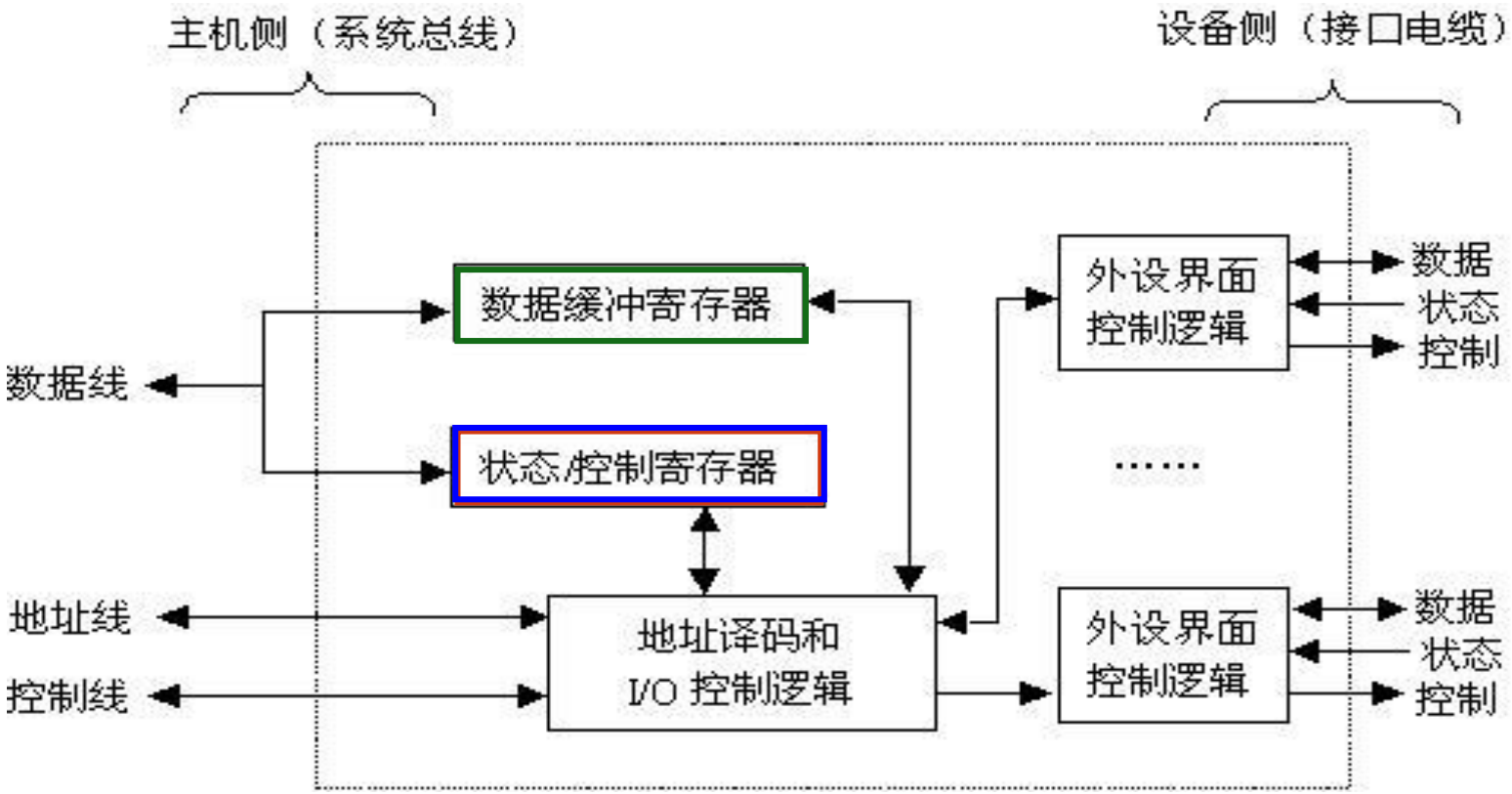
I/O控制器和插座合起来称为I/O接口。

I/O接口的职能

- **数据缓冲:** 提供数据缓冲寄存器，以达到主机和外设工作速度的匹配
- **错误或状态检测:** 提供状态寄存器，以保存各种错误或状态信息供CPU查用
- **控制和定时:** 提供控制和定时逻辑，以接受从系统总线来的控制定时信号
- **数据格式转换:** 提供数据格式转换部件使通过外部接口得到的数据转换为内部接口需要的格式，或在相反的方向进行数据格式转换。
- **与主机和设备通信:**上述功能通过I/O接口与主机之间、I/O接口与设备之间的通信来完成。

I/O接口的结构

- I/O控制器的一般结构：不同I/O模块在复杂性和控制外设的数量上相差很大



通过发送命令字到I/O控制寄存器来向设备发送命令

通过从状态寄存器读取状态字来获取外设或I/O控制器的状态信息

通过向I/O控制器发送或读取数据来和外设进行数据交换

将I/O控制器中CPU能够访问的各类寄存器称为I/O端口

对外设的访问通过向I/O端口发命令、读状态、读/写数据来进行

I/O设备的寻址方式

（1）(与主存一起)统一编址方式（内存映射方式）

与主存空间统一编址，将主存空间分出一部分地址给I/O端口进行编号。

（该方法是将I/O端口映射到某主存区域，故也称为“存储器映射方式”）

例如，RISC机器、Motorola公司的处理器等采用该方案

（2）独立编址方式(也称“特殊I/O指令方式”)--用特殊专门I/O指令实现

不和主存单元一起编号，而是单独编号，使成为一个独立的I/O地址空间

例如，Intel公司和Zilog公司的处理器就是独立编址方式

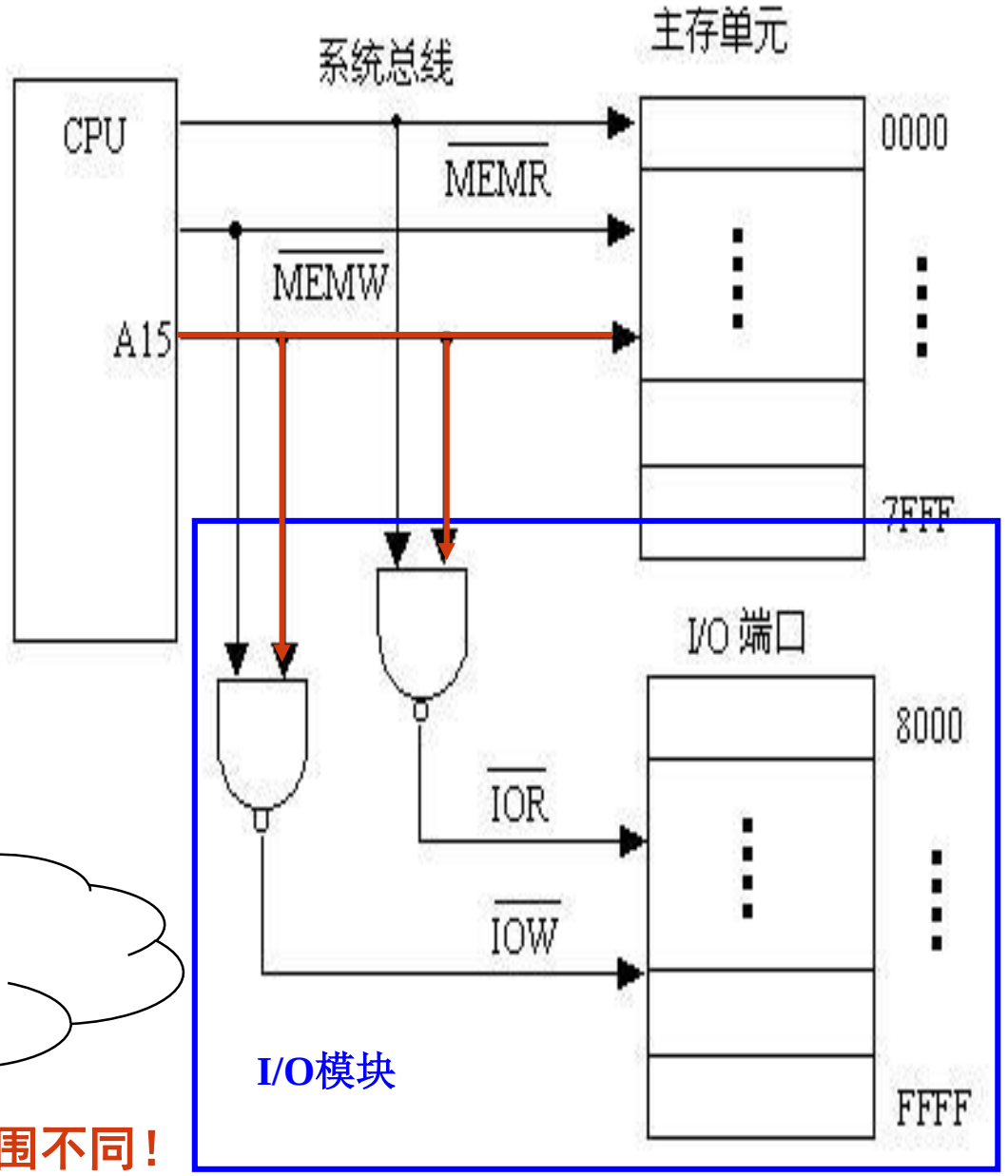
如专门指令 IN AL, 80H(外设端口地址) #从80H端口读外设的数据到AL累加器

OUT 81H(外设端口地址)，AL #将AL累加器数据送给81H端口的外设

- 对I/O端口读写，就是向I/O设备送出命令或从设备取得状态或读/写设备数据
- 一个I/O控制器可能会占有多个端口地址
- I/O端口必须编号后，CPU才能访问它
- I/O设备的寻址方式就是I/O端口的编号方式

统一编址方式

- 无需设置专门I/O指令，只要用一般访存指令就可存取I/O端口。
- 地址线的高位参与片选控制逻辑。
- CPU不直接通过读写控制信号IOR、 IOW对I/O端口读写，而是根据I/O端口在地址空间的位置，通过地址译码来实现。

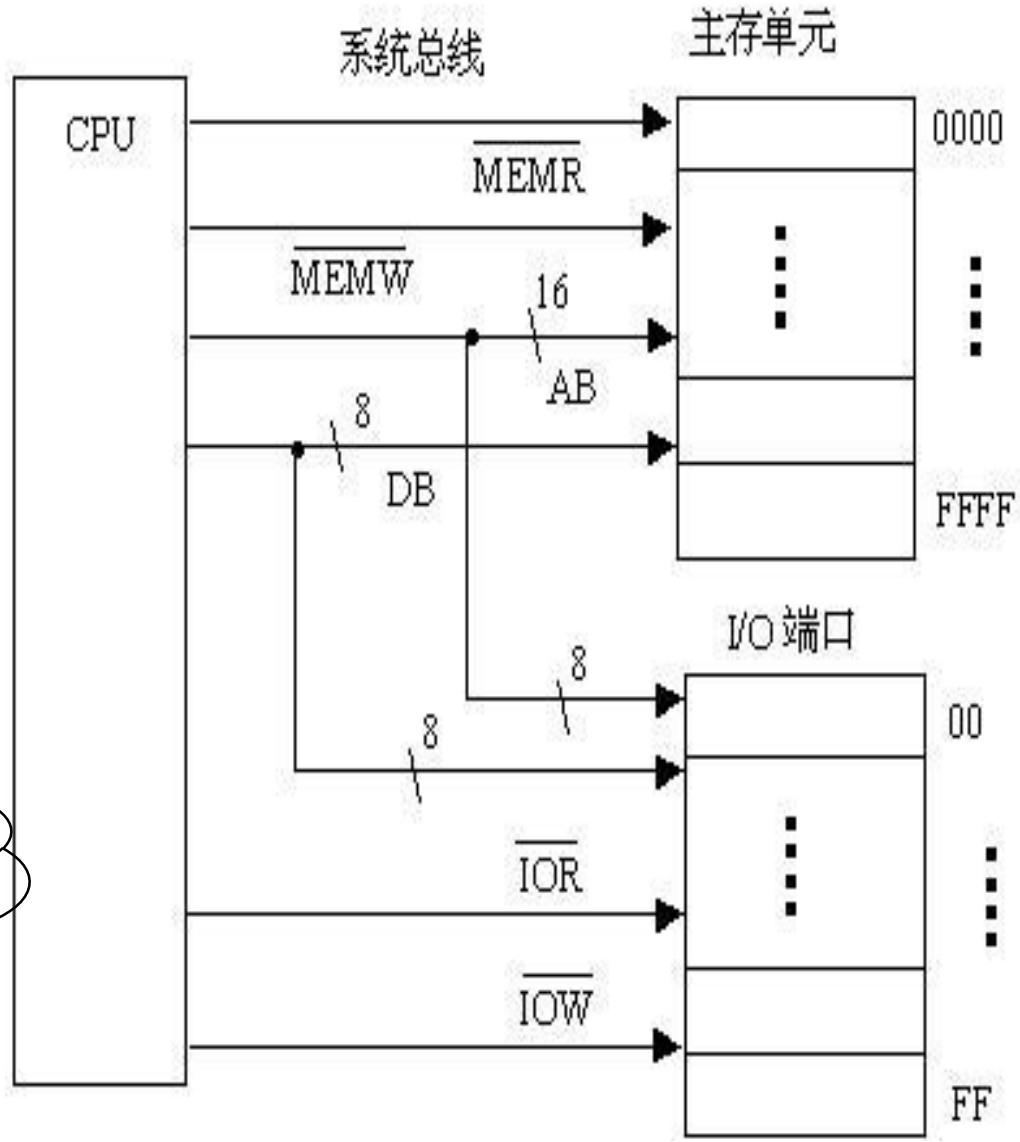


也由访存指令发出，访问的地址范围不同！

独立编址方式

- 通过不同的读写控制信号$\overline{IOR}$、$\overline{IOW}$和$\overline{MEMR}$、$\overline{MEMW}$来控制对I/O 端口和存储器的读写。
- 一般I/O端口比存储器单元少，所以选择I/O端口时，只需少量地址线。
- 指令系统必须设计专门的I/O指令。

由专门的I/O指令确定，指令中给的地址可能相同，但操作命令不同！



奔腾机的I/O端口编址方式-采用专门的I/O指令:IN和OUT

- 采用独立编址方式，I/O地址空间由2^{16}(64K)个8位端口组成
- 虽然具有64K字节的寻址空间，但一般只使用其中1K字节的I/O空间，故只用低10位地址线寻址
- 两个连续的8位端口可作为一个16位端口；四个连续的8位端口可作为一个32位端口。所以一次可传送32位、16位或8位数据
- 采用专门的I/O指令：IN和OUT（处理器执行到这些指令时产生相应的I/O读写命令信号）
- 部分外设的I/O地址分配表

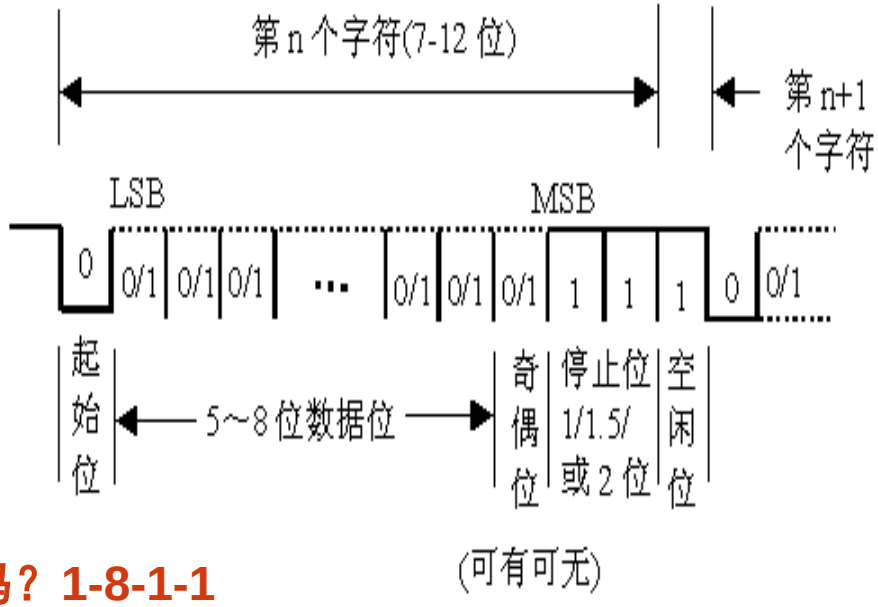
奔腾机I/O端口地址分配表

部分外设的 I/O 地址分配表

输入/出设备	I/O 地址	占用地址数
DMA 控制器 1	000-01FH	32
中断控制器 1	020-03FH	32
定时器/计数器	040-05FH	32
键盘控制器	060-06FH	32
实时时钟，NMI屏蔽寄存器	070-07FH	16
DMA 页面寄存器	080-09FH	32
中断控制器 2	0A0-0BFH	32
DMA 控制器 2	0C0-0DFH	32
硬盘控制器 2	170-177H	8
硬盘控制器 1	1F0-1F8H	9
游戏 I/O 口	200-207H	8
并行打印机口 2	278-27FH	8
串行口 4	2E8-2EFH	8
串行口 2	2F8-2FFH	8
软盘控制器 2	370-377H	8
并行打印机口 1	378-37FH	8
单色显示器/打印适配器	3B0-3BFH	16
彩色/图形监视器适配器	3D0-3DFH	16
串行口 3	3E8-3EFH	8
软盘控制器 1	3F0-3F7H	8
串行口 1	3F8-3FFH	8

并行传输和串行传输（数据通信课学过？）

- 并行传输方式
  - 多位数据在多条数据线上并行传送
  - 最大传输率为：时钟频率 x 数据线宽度
- 串行传输方式
  - 波特率：每秒钟通过信道传输的码元数
  - 比特率：每秒钟传输的比特位数
  - 两相调制时，波特率=比特率
  - 异步串行



还记得PS/2的格式吗？1-8-1-1

  - 每个字符的开始是随机的，需起始位，字符内的位之间同步
  - 有效数据位为5位时，停止位取1位或1.5位，其他情况取1位或2位
  - 一个字符可能由7～12位信息组成，称其为一个数据帧
  - 缺点：每个字符都有额外信息，实际字符传输率低
  - 同步串行
  - 字符之间、字符内的位之间都同步

还记得PS/2的格式吗？1-8-1-1: 1个起始位总是逻辑0, 8个数据位(LSB)低位在前, 1个奇偶校验位奇校验, 1个停止位总是逻辑1.(还有1个应答位仅用在主机对设备的通讯中）

第三讲 I/O设备与其它设备间的传输方式(I/O传输方式)

主　要　内　容

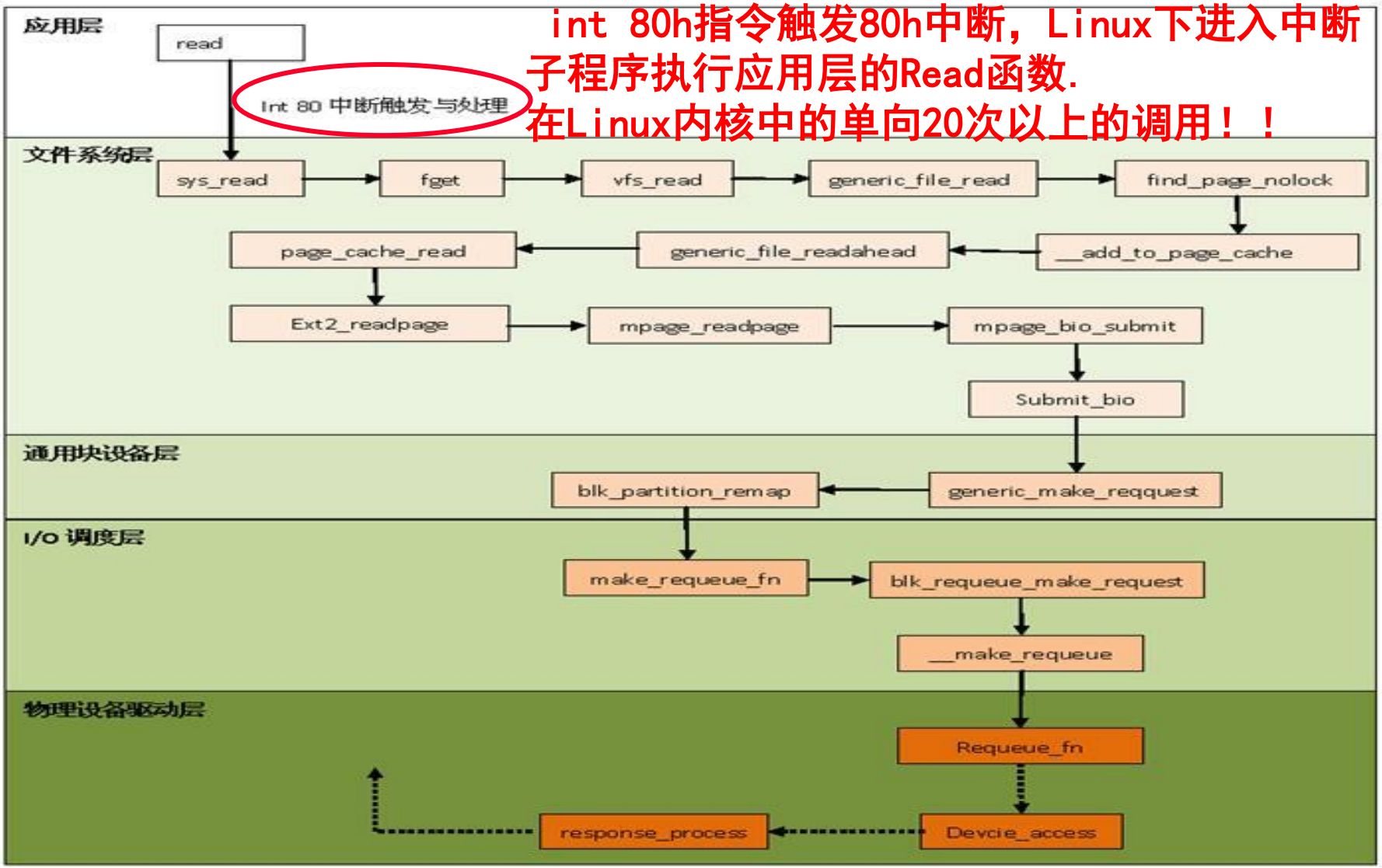
- OS在I/O系统中的职责
- I/O设备与其它设备间的传输方式(I/O传输方式）
  - 轮询方式（程序直接控制 / 程序查询方式）
  - 程序中断方式（中断驱动方式）
    - 中断响应的条件和中断响应过程
    - 中断处理过程
    - 中断控制器
    - 多重中断和中断屏蔽
  - 直接存储器访问方式（DMA方式）
    - DMA方式的要点
    - DMA控制器的结构
    - DMA的三种控制方式
    - DMA传输过程

操作系统OS在I/O中扮演的角色

- I/O系统的三个特性
  - (多个程序)共享性：I/O系统被处理器执行的多个程序共享，由OS统一调度管理
  - 复杂性：I/O设备控制的细节比较复杂，不能由上层用户程序来实现，需OS提供专门的驱动程序
  - (通常使用)采用中断I/O方式：I/O系统通常使用外部中断请求来要求处理器执行专门的输入/出程序。中断导致向内核态转移，故必须由OS来处理
- I/O系统的三个特性决定了OS在I/O中的职能
  - 保证用户程序只能访问自己有权访问的那部分I/O设备
  - 为用户程序提供设备驱动程序以屏蔽设备控制的细节
  - 处理外部I/O中断，提供中断服务程序
  - 对共享的I/O资源提供合理的调度管理，使系统的吞吐率达到最佳
- 主机必须和I/O设备进行以下三类信息的通信
  - OS必须能向I/O设备提供命令，如：磁盘寻道
  - 需要知道：何时I/O设备能完成操作？何时遇到什么异常问题？
  - 数据必须能在主机（主存或CPU）和I/O设备之间进行传输

操作系统在I/O中扮演的角色

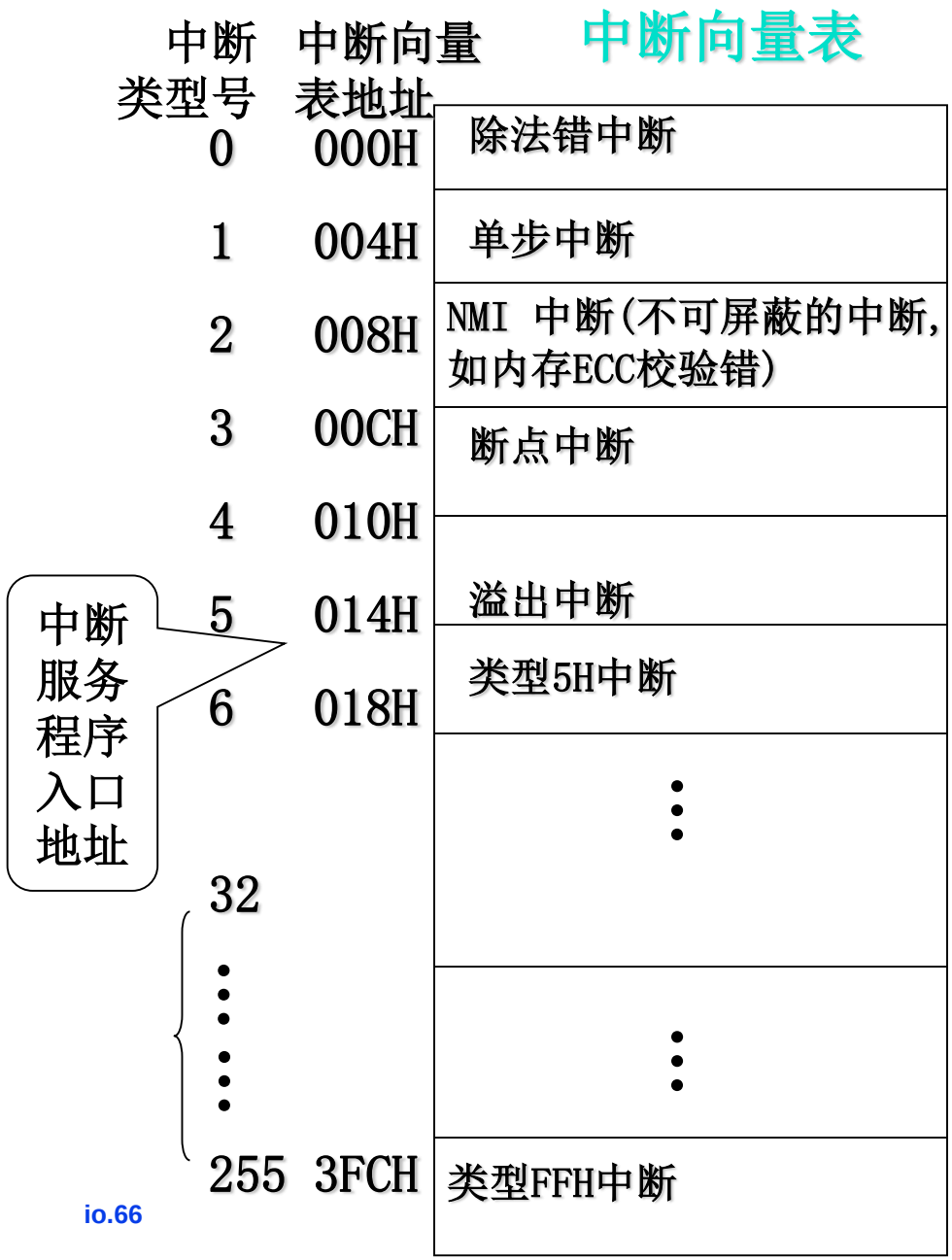
各类用户的I/O请求需要通过某种方式传给OS：



操作系统OS在I/O中扮演的角色

- 系统调用是一个(软件产生的)软中断，中断类型号是0x80，它是上层应用程序与Linux系统内核进行交互通信的唯一接口。这个中断的设置在kernel/sched.c中441行中,下面为该文件片段:
- void sched_init(void)
- { int i; struct desc_struct * p;
- .....
- set_system_gate(0x80,&system_call); } // 441行
- 最后一句就将”0x80 ”这个”中断类型号” 与system_call（系统调用）联系起来。通过汇编指令”int 0x80”，就可使用OS内核资源。通常应用程序都是使用具有标准接口定义的C函数库间接的使用内核的系统调用，即应用程序调用C函数库中的函数，C函数库中再通过int 0x80进行系统调用。所以，系统调用过程是这样的：普通用户的应用程序调用libc中的函数－> libc中的函数引用系统调用宏 －> 系统调用宏中再使用” int 0x80”指令完成系统调用并返回

中断向量表



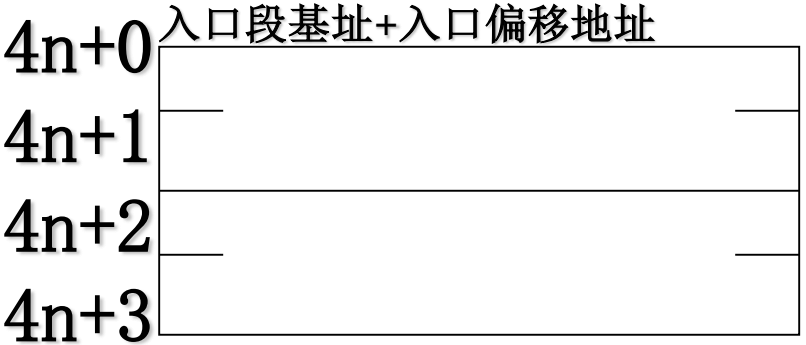
80x86CPU由中断向量表提供中断服务程序的入口地址。中断向量表建立在内存最低端的1KB RAM区，地址范围为000H~3FFH， 只可存放256个中断中断类型的中断向量。

0~17：内部中断

18~31：备用

32~255：用户,例如,中断类型号(中断号)128,就是80H， **int 80H**

中断向量表地址：V=4n



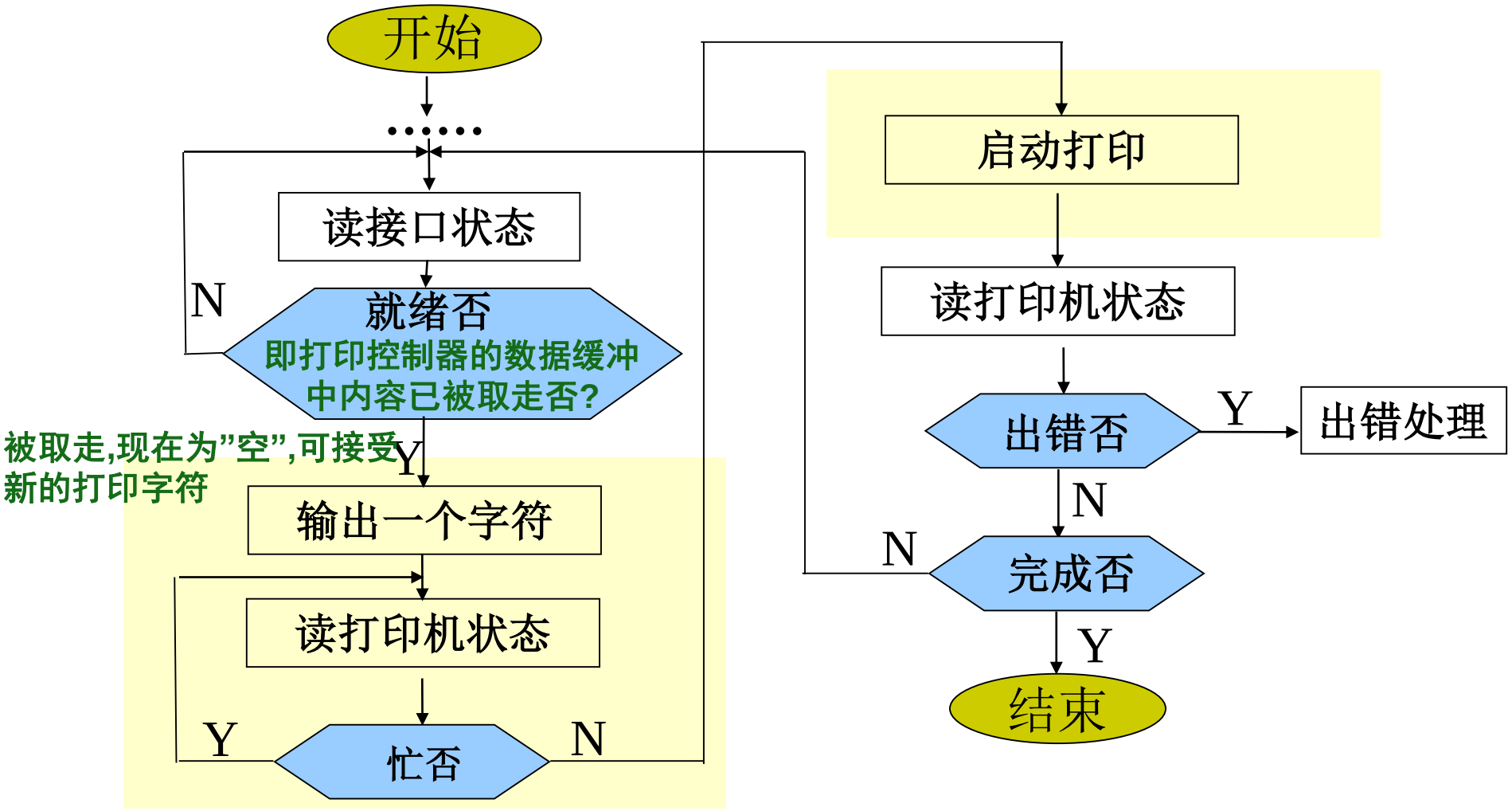
I/O设备与主机进行数据交换的三种基本方式

- 程序直接控制方式（最简单的I/O方式）
  - 无条件传送：对简单外设定时（同步）进行数据传送
  - 条件传送：Polling (轮询，查询)：OS主动查询，也称为程序查询方式
    - I/O设备（包括I/O接口）将自己的状态放到一个状态寄存器中
    - OS阶段性地查询状态寄存器中的特定状态，以决定下一步动作
- I/O Interrupt (中断I/O方式): 几乎所有系统都支持的中断I/O方式
  - 若一个I/O设备需要CPU干预，它就通过中断请求通知CPU
  - CPU中止当前程序的执行，调出OS（中断处理程序）来执行
  - 处理结束后，再返回到被中止的程序继续执行
  - OS是被动调出的，也称为中断驱动I/O方式
- Direct Memory Access (DMA方式): 磁盘等高速外设特有的I/O方式
  - 磁盘等高速外设成批地直接和主存进行数据交换
  - 需要专门的DMA控制器控制总线，完成数据传送
  - 当外设准备好数据后，向DMA控制器发DMA请求信号，DMA控制器再向CPU发总线请求，CPU让出总线后，由DMA控制器控制总线进行传输，无需CPU干涉

程序直接控制（程序查询）方式--打印输出为例

- 举例：用程序直接控制方式控制打印输出

下图“就绪”的含义表示打印控制器的数据缓冲中内容已被取走，现为“空”，可接受新的打印字符



X86机器用于打印输出的标准 (汇编)子程序

功能：打印AL寄存器中的字符。

访问I/O的指令、检查状态位的指令各是什么？

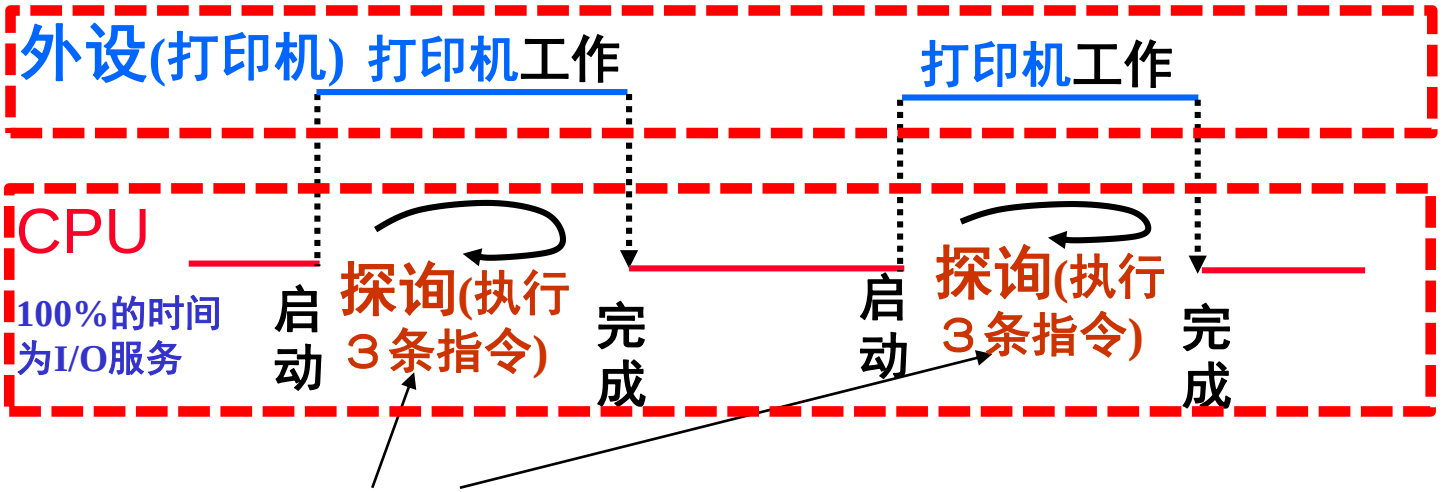
IN / OUT指令！ TEST指令!

```
PRINT   PROC  NEAR
        PUSH  AX          ；保留用到的寄存器
        PUSH  DX          ；保留用到的寄存器
        MOV   DX, 378H    ；输入数据锁存器口地址
        OUT   DX, AL      ；输出要打印的字符到数据锁存器
        MOV   DX, 379H    ；输入状态寄存器口地址,将379H赋给DX
WAIT:   IN    AL, DX      ；读打印机状态位
        TEST  AL, 80H     ；检查忙碌位
        JE    WAIT        ；等待直到打印机不忙
        MOV   DX, 37AH    ；输入命令(控制)寄存器口地址
        MOV   AL, 0DH     ；置选通位＝1
        OUT`  DX, AL      ；使控制卡的命令锁存器中选通位置1
        MOV   AL, 0CH     ；置选通位＝0
        OUT   DX, AL      ；使控制卡的命令锁存器中选通位置0
        POP   DX
        POP   AX          ；恢复寄存器
        RET
PRINT   ENDP
```

下页PPT起的“探询”操作要用到这三条指令

回顾：过程/函数/子程序中的开始总是先要保护现场，最后总是要恢复现场！

程序控制I/O （查询I/O方式）--打印输出为例



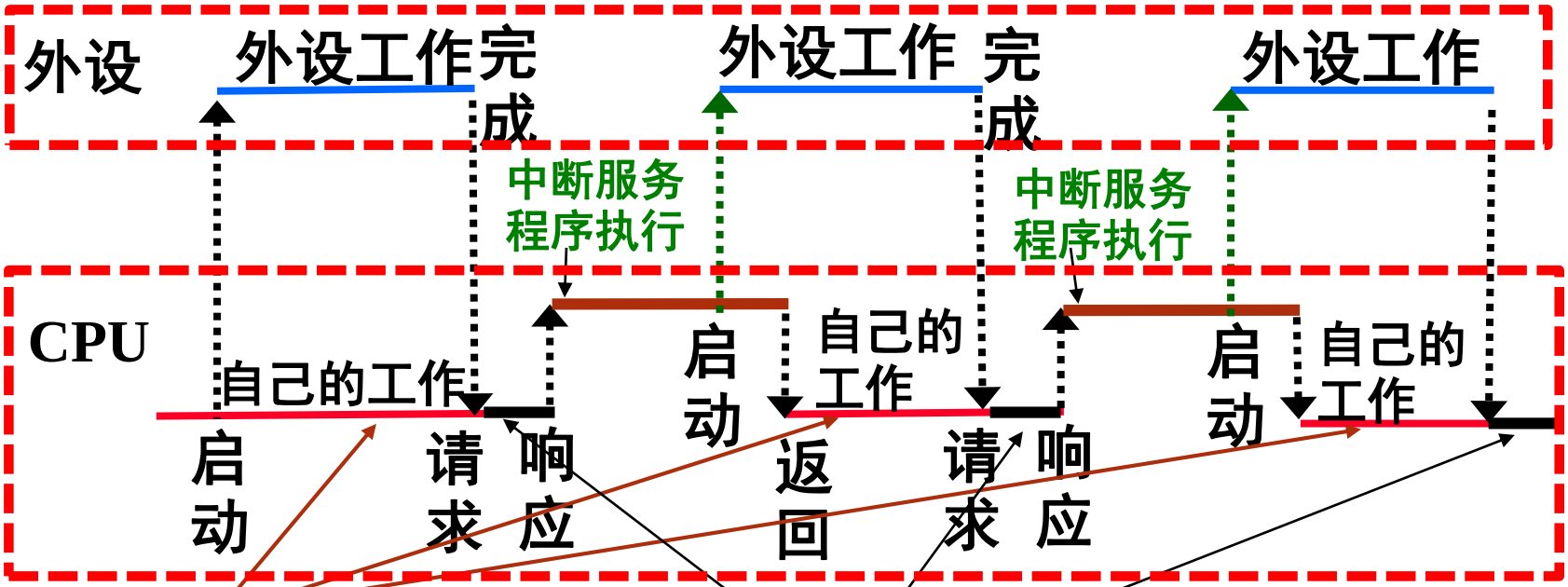
“踏步”现象,此时，CPU处于停止状态吗？不是！只是不断执行(前页PPT中)“ IN-TEST-JE” 3条指令,即读打印机状态位--检查忙碌位--等待直到打印机不忙。“探询”期间，可一直不断查询（独占查询），也可定时查询（需保证数据不丢失！）。

- 特点：
  - 简单、易控制、外围接口控制逻辑少；
  - CPU与外设串行工作，效率低、速度慢，适合于慢速设备
  - 查询开销极大 (CPU完全在等待“外设完成”）
- 工作方式：完全串行工作方式或部分串行，CPU用100%的时间为I/O服务！

中断I/O方式

- **基本思想：**

当外设准备好时，便向CPU发中断请求，CPU响应后，中止现行程序的执行，转入一个“中断服务程序”进行输入/出操作，实现主机和外设接口之间的数据传送，并启动外设工作。“中断服务程序”执行完后，返回原被中止的程序断点处继续执行。此时，外设和CPU并行工作。



上述哪段时间CPU和外设并行工作？

(响应中断过程的)程序切换由硬件执行“中断隐指令”完成，时间为1时钟周期

复习：处理器中的异常（中断）处理机制

异常(中断)发生时，处理器必须做以下基本处理（执行中断隐指令以响应中断）：

① 保护断点和程序状态：

将返回原程序执行的断点和程序状态保存到堆栈或特殊寄存器中

PC=>堆栈 或 EPC

PSWR=>堆栈 或 EPSWR

（注：PSW（Program Status Word）：程序状态字

PSWR（PSW寄存器）：用于存放程序状态。如，X86的FLAGS）

② 识别异常事件，并转到具体的异常处理程序执行

有两种不同的方式：软件识别和硬件识别（向量中断方式）

（1）软件识别（MIPS采用）：设置一个异常状态寄存器（MIPS中为Cause寄存器），用于记录异常原因。操作系统使用一个统一的异常处理程序（MIPS的入口为0x8000 0180），该程序按优先级顺序查询异常状态寄存器，识别出异常事件。

（2）硬件识别（向量中断）（80x86采用）：用专门的硬件查询电路按优先级顺序识别异常，得到一个“中断类型号”，根据此号，到中断向量表中读取对应的中断服务程序的入口地址。

除上述2个任务外，还有一个首要任务！是什么？关中断（禁止中断）！即：将中断允许（触发器）标志清0。

8086/8088的中断向量表

中断向量表，也称中断入口地址表（或异常表），位于0000H～03FFH。共256组，每组占四个字节 CS:IP 。

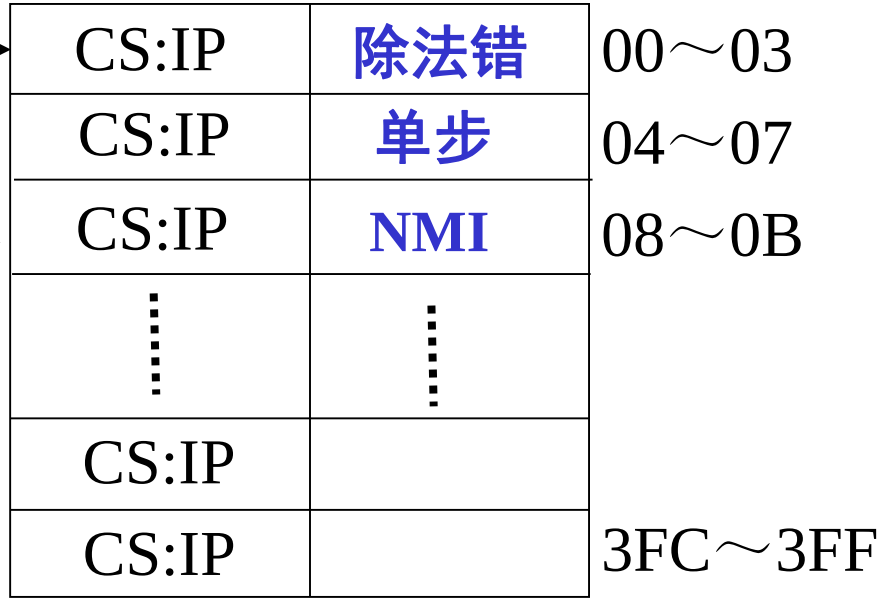
中断向量表（异常表）中每一项是对应异常处理程序的入口地址，被称为中断向量(Interrupt Vector)

共有256个不同中断类型号。有了中断类型号就可得到中断向量，向量地址=中断类型号x4,从而转到中断服务程序执行。中断类型号怎么得到呢？

中断向量表的起始地址存放在一个异常表基址寄存器中。

例1：除法错的中断类型号为0，
故其中断向量地址 为：0x4=0

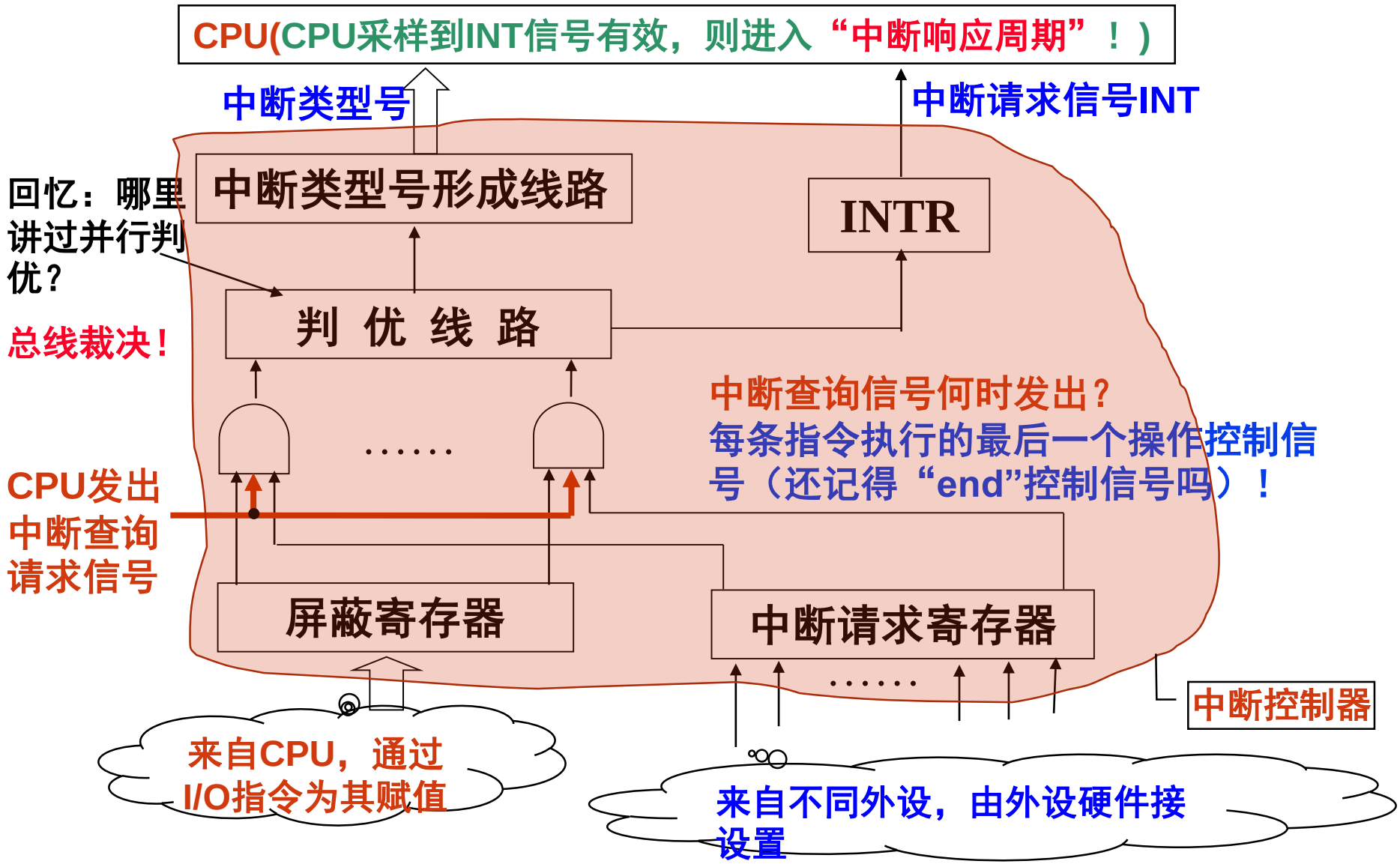
例2：NMI的中断类型号为2，
故其中断向量地址为：2x4=8



中断控制器的基本结构—以8259A中断控制芯片为例

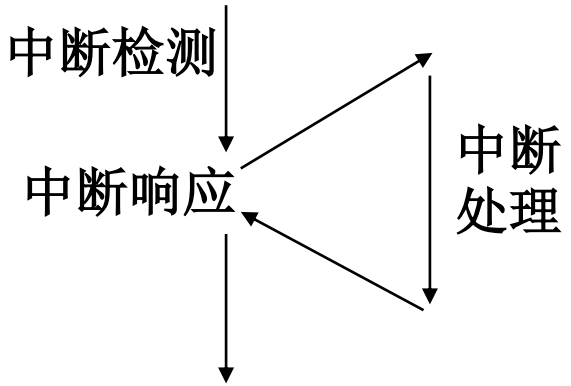
中断类型号送到什么线上？数据线上！为什么？

何时采样中断请求信号？中断查询信号发出后的固定时间内



中断驱动I/O方式

- 中断过程
  - 中断检测（硬件实现）
  - 中断响应（硬件实现）
  - 中断处理（软件实现）



- 中断响应
  - 中断响应是指主机发现外部中断请求，(执行完当前指令后)中止现行程序(其它指令)的执行，到调出中断服务程序这一过程

(1) 中断响应的条件

① CPU处于开中断状态

② 在一条指令执行完

③ 至少要有一个未被屏蔽的中断请求

问题：中断响应的时点与异常处理的时间点是否相同？为什么？

通常在一条指令执行结束后开始查询有无中断请求，有的话立即响应，所以，一般在指令执行完时响应中断；而“异常”发生在指令执行过程中，所以，不能等到指令执行完才进行异常处理。

中断驱动I/O方式(续)

(2) 中断响应过程:执行一条隐指令，完成一次总线操作，从总线上取中断类型号

具体来说，处理器做三件事：

① (处理器硬件)关中断

0 => 中断允许触发器C_{IEN}

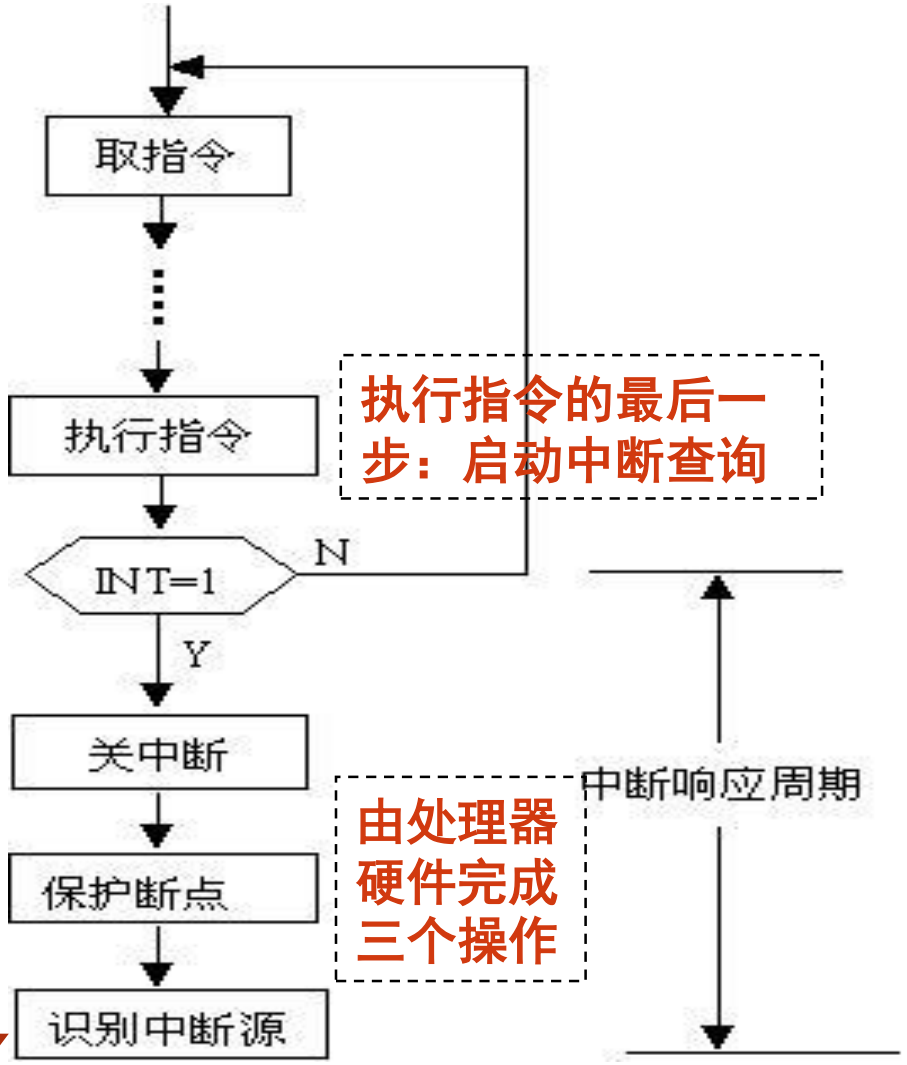
② (处理器硬件)保护断点和程序状态

PC => 堆栈（或特殊寄存器EPC）

PSW => 堆栈

③ (处理器硬件)识别中断源

取得中断服务程序首地址和初始PSW分别送PC和PSWR



从总线的数据线上取得中断类型号后得到中断服务程序首址（向量中断）或直接转中断查询程序（软件查询）

复习：中断源的识别方法

- 软件方法（轮询）

  中断查询程序根据中断请求状态，按优先级顺序来识别

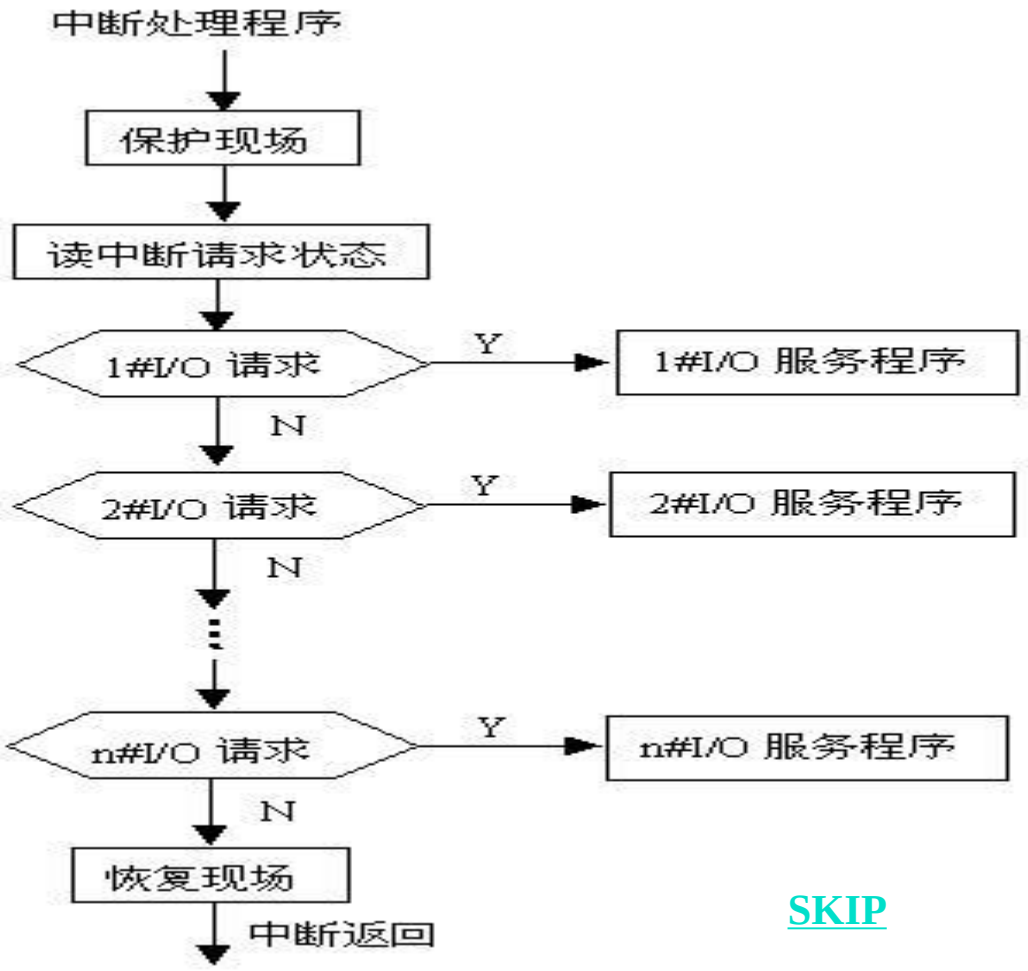
  如：MIPS的异常/中断查询程序入口为：

  0x8000 0180

- 硬件方法（向量中断）

  将所有中断请求状态送到一个排队电路中，根据中断优先级识别出最高优先级的中断请求

  - 链式查询（菊花链）
  - 独立请求（并行判优）

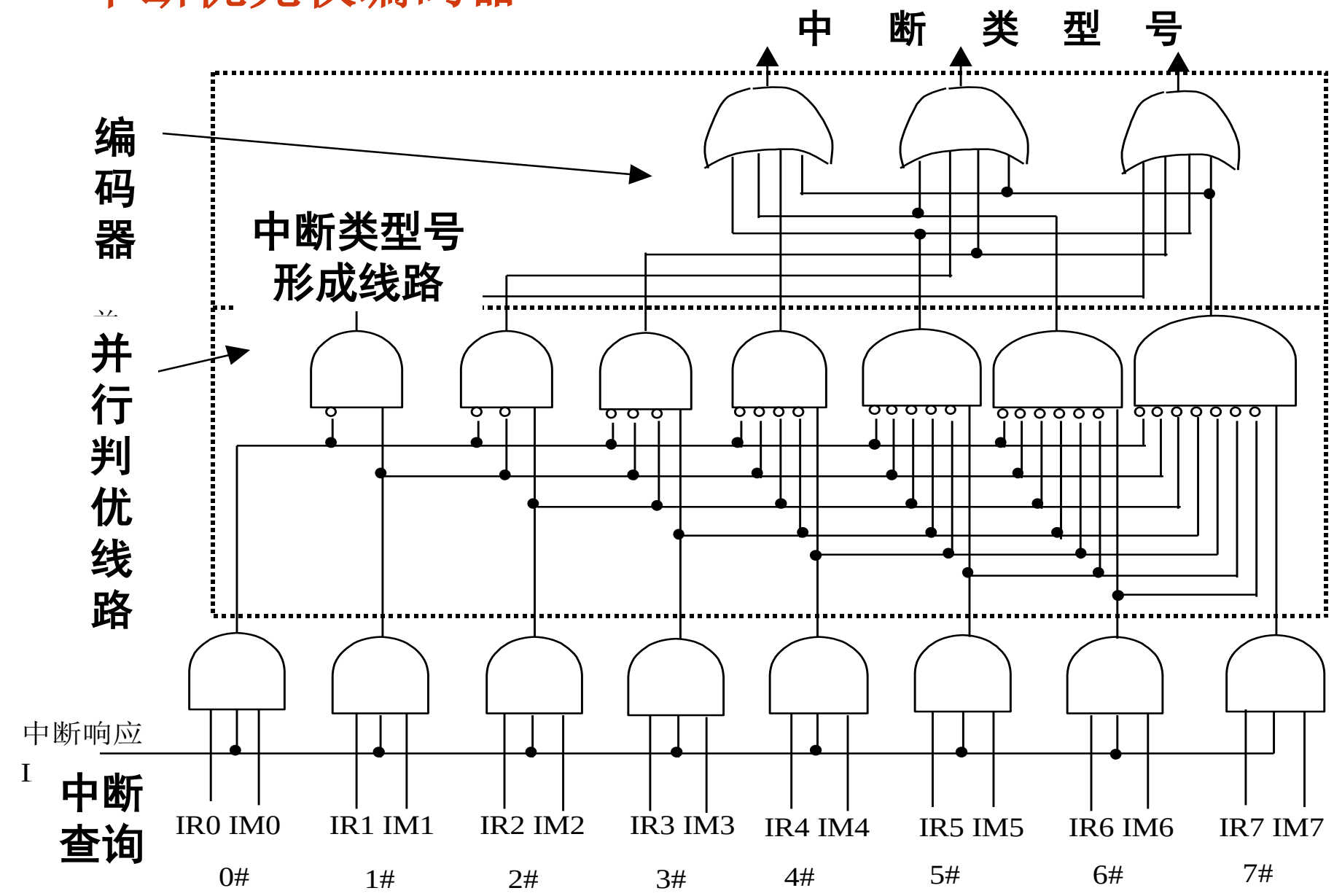


SKIP

中断控制器中的“中断优先权编码器” 专门用来进行中断源识别

注：内部异常和外部中断都有优先级，通常所有内部异常的优先级都比外部中断高。为什么？

中断优先权编码器

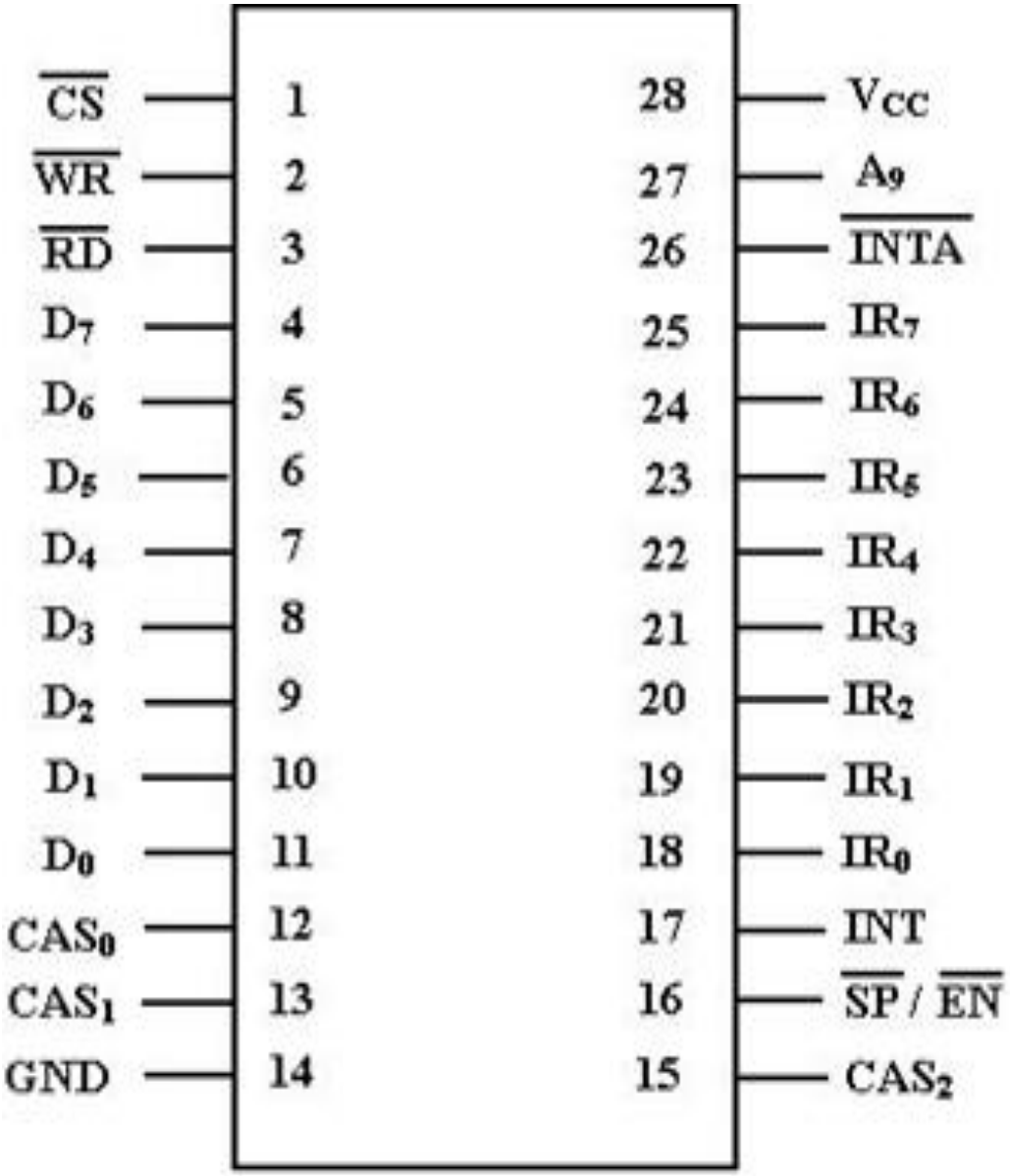


IR:Interrupt request(某个设备提出的)(中断请求)； M:Interrupt mask(对某个设备)(中断屏蔽)

中断控制器举例-8259A

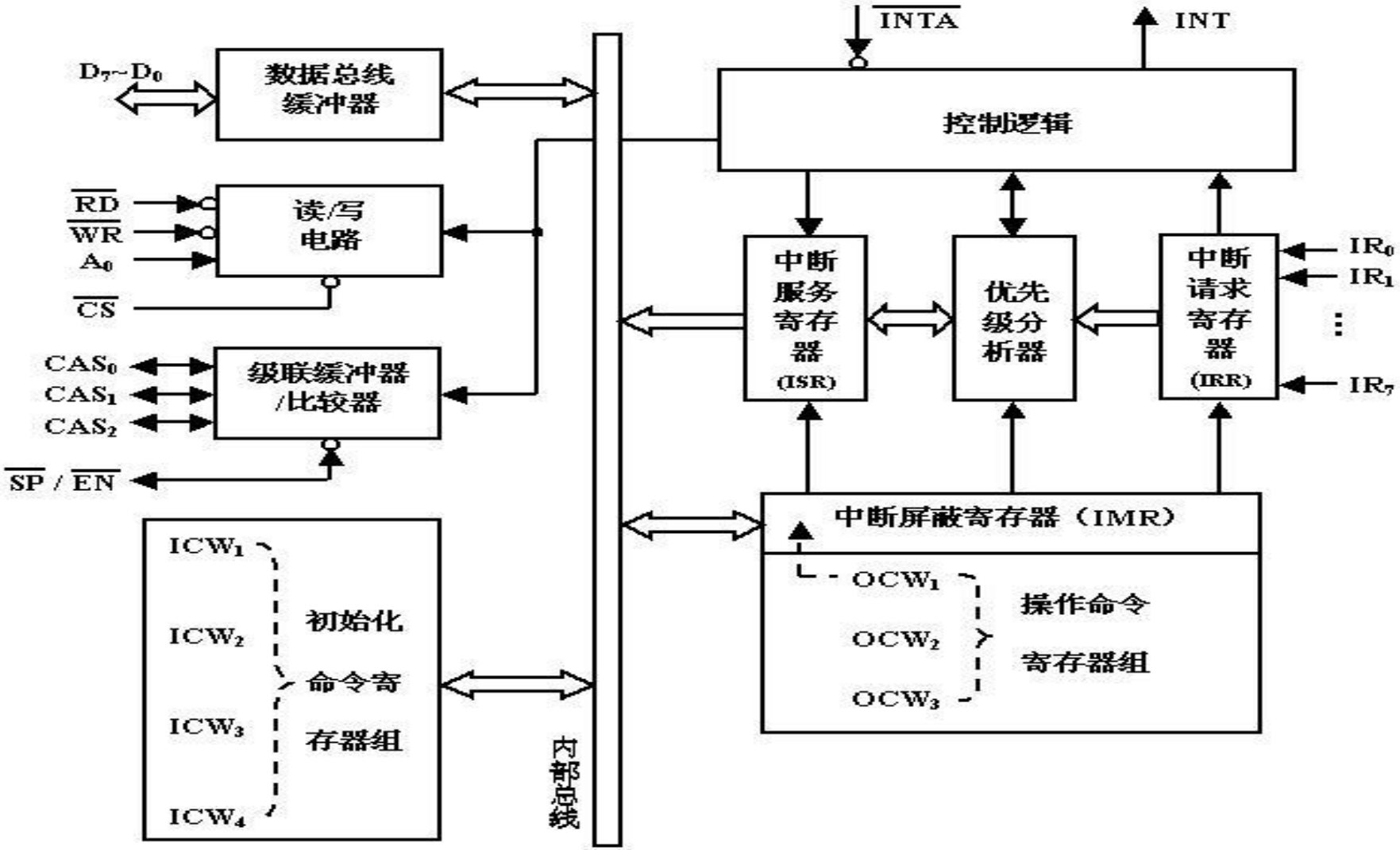
- 8259A是可编程中断控制器
- 8259A的功能
  - 包含中断请求锁存、中断屏蔽、优先级排队、中断优先权编码器等电路
  - 既可支持程序查询式中断，又可支持向量式中断
  - 支持8级中断，通过多片级联最多可构成64级中断
  - 各种中断功能可通过编程设定或更改

芯片有8个数据引脚，中断类型号最大范围为0-63，即最多可支持64级中断。



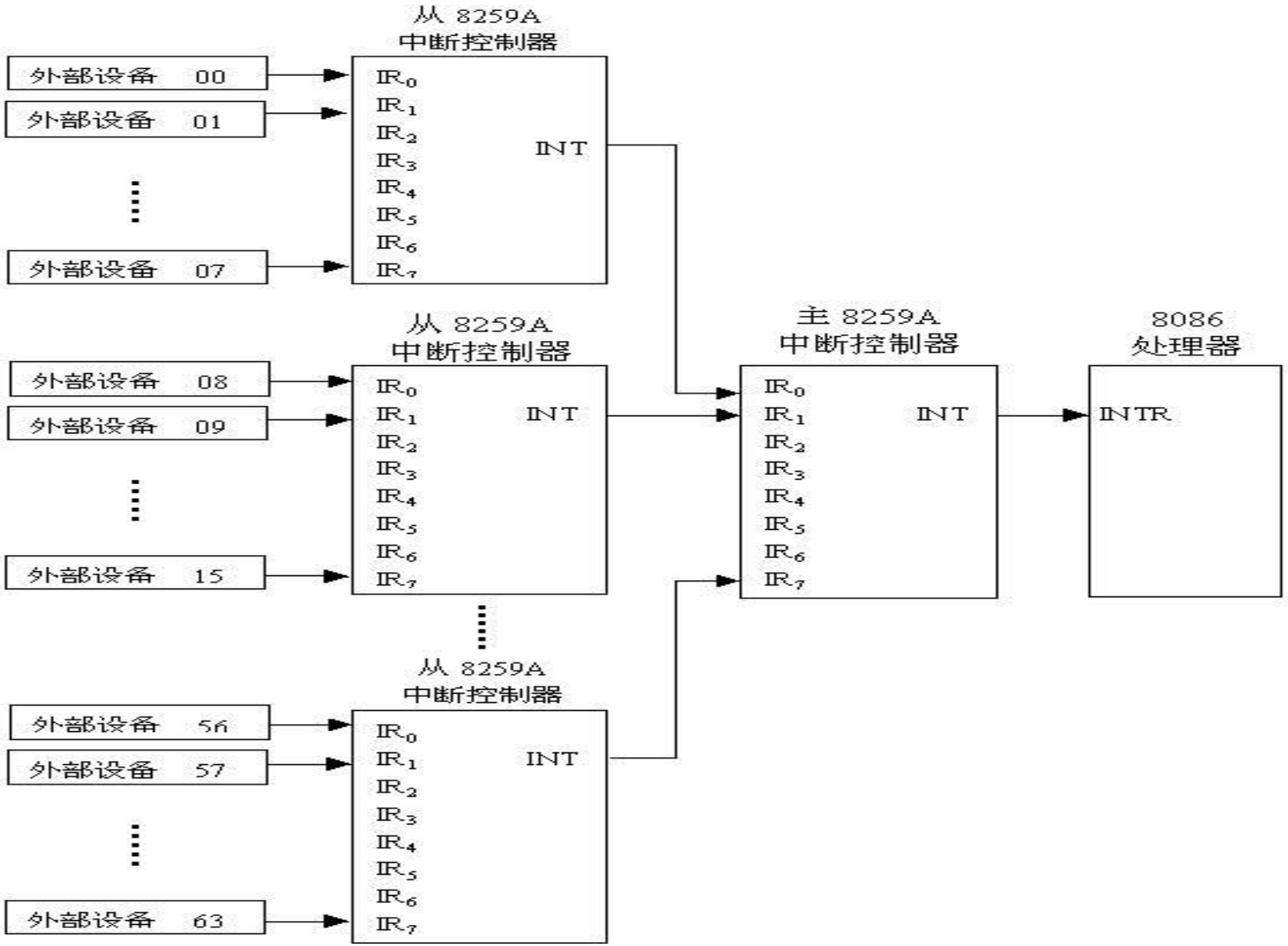
8259A的内部结构

一片8259A芯片可以接受并管理8级可屏蔽中断请求



8259A 内部结构框图

8259A芯片的级联


从 8259A
中断控制器
外部设备 00
外部设备 01
外部设备 07
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INT
从 8259A
中断控制器
外部设备 08
外部设备 09
外部设备 15
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INT
从 8259A
中断控制器
外部设备 56
外部设备 57
外部设备 63
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INT
主 8259A
中断控制器
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INT
8086
处理器
INTR

单重中断、多重中断和中断优先级的概念

- 单重中断不允许在中断处理时被新的中断打断，因而直到中断返回前才会开中断。
- 多重中断的概念：

  在一个中断处理（即执行中断服务程序）过程中，若又有新的中断请求发生，而新中断优先级高于正在执行的中断，则应立即中止正在执行的中断服务程序，转去处理新的中断。这种情况为多重中断，也称中断嵌套。
- 中断优先级的概念：

  中断响应优先级----由查询程序或硬联排队线路决定的优先权，反映多个中断同时请求时选择哪个响应。

  中断处理优先级----由各自的中断屏蔽字来动态设定，反映本中断与其它中断间的关系。
- 多重中断和中断处理优先权是动态分配的

  回想一下，中断屏蔽字在何处用到的？

中断处理过程

中断过程：中断响应 + 中断处理

中断响应的结果就是调出相应的中断服务程序 （处在“禁止中断”状态）

中断处理是指执行相应中断服务程序的过程

- 不同的中断源其对应的中断服务程序不同。
- 典型的多重中断处理（中断服务程序）分为三个阶段：
  - 先行段（准备阶段）

    保护现场及旧屏蔽字

    查明原因（软件识别中断时）

    设置新屏蔽字(单重中断系统无需设置)

    开中断

    处在“禁止中断”状态，不允许被打断
  - 本体段（具体的中断处理阶段）

    处在“允许中断”状态，可被新的处理优先级更高的中断打断
  - 结束段（恢复阶段）

    关中断

    恢复现场及旧屏蔽字

    清“中断请求”

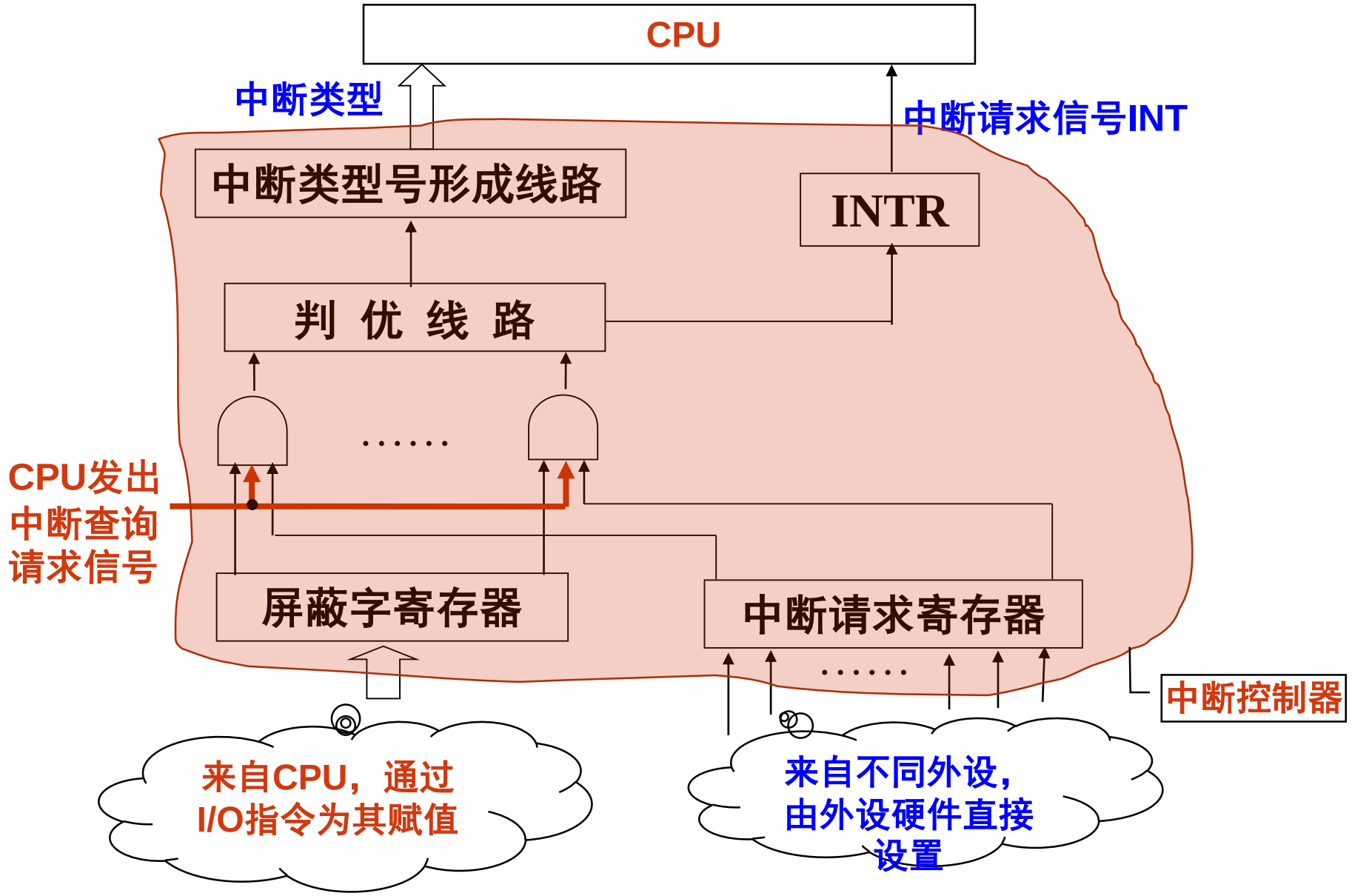
    开中断

    中断返回

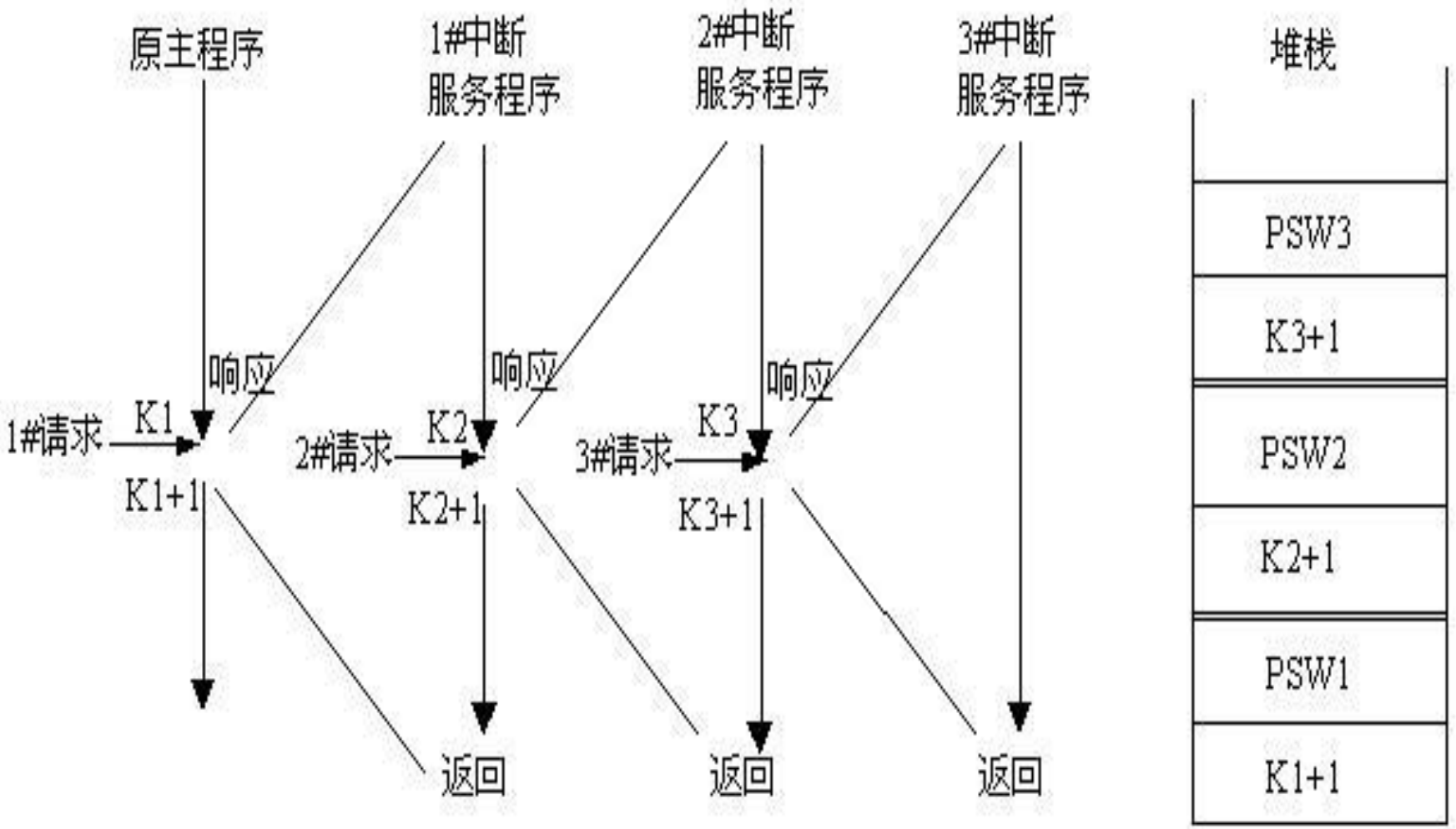
    处在“禁止中断”状态，不允许被打断

再强调：单重中断系统无需设置中断屏蔽字

中断控制器的基本结构（如：8259A）



多重中断嵌套



中断优先级的顺序是：

3# > 2# > 1#

1#对2#开放（不屏蔽）

2#对3#开放（不屏蔽）

中断优先权的动态分配

- 举例：假定某中断系统有四个中断源，其响应优先级为1>2>3>4。假定在用户程序时同时发生1、3、和4级中断请求，执行3级中断服务程序时发生2级中断请求。分别写出处理优先级为1>2>3>4和1>4>3>2时各中断的屏蔽字及CPU完成中断处理的过程。

(1) 中断处理优先级为1>2>3>4时：

中断程序级别 中断处理程序优先级	屏蔽字 最高响应优先级 1级 2级 3级 4级 最低响应优先级
优先级最高 第1级	1 1 1 1
第2级	执行2级中断处理程序时，仅对1级中断请求开放 0 1 1 1
第3级	执行3级中断处理程序时，仅对1和2级中断请求开放 0 0 1 1
优先级最低 第4级	执行4级中断处理程序时，仅对1、2和3级中断请求开放 0 0 0 1

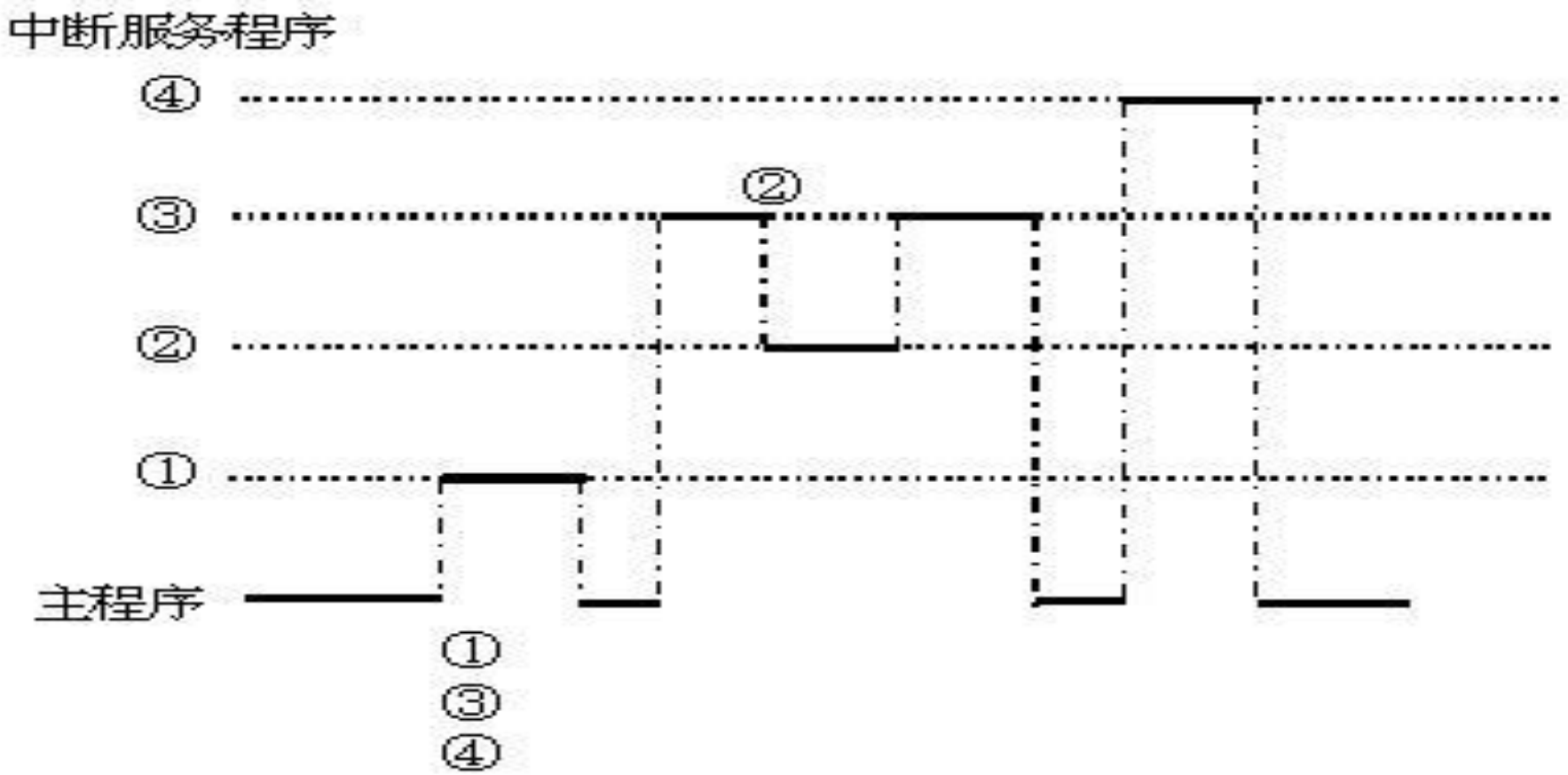
（假定1是屏蔽，0是开放）

不接受新中断　　接受新中断

中断优先权的动态分配

- 举例：假定某中断系统有四个中断源，其响应优先级为1>2>3>4。假定在用户程序时同时发生1、3、和4级中断请求，执行3级中断服务程序时发生2级中断请求。分别写出处理优先级为1>2>3>4和1>4>3>2时各中断的屏蔽字及CPU完成中断处理的过程。

(1) 中断处理优先级为1>2>3>4时：



中断优先权的动态分配

- 举例：假定某中断系统有四个中断源，其响应优先级为1>2>3>4。假定在用户程序时同时发生1、3、和4级中断请求，执行3级中断服务程序时发生2级中断请求。分别写出处理优先级为1>2>3>4和1>4>3>2时各中断的屏蔽字及CPU完成中断处理的过程。

(2) 中断处理优先级为1>4>3>2时：

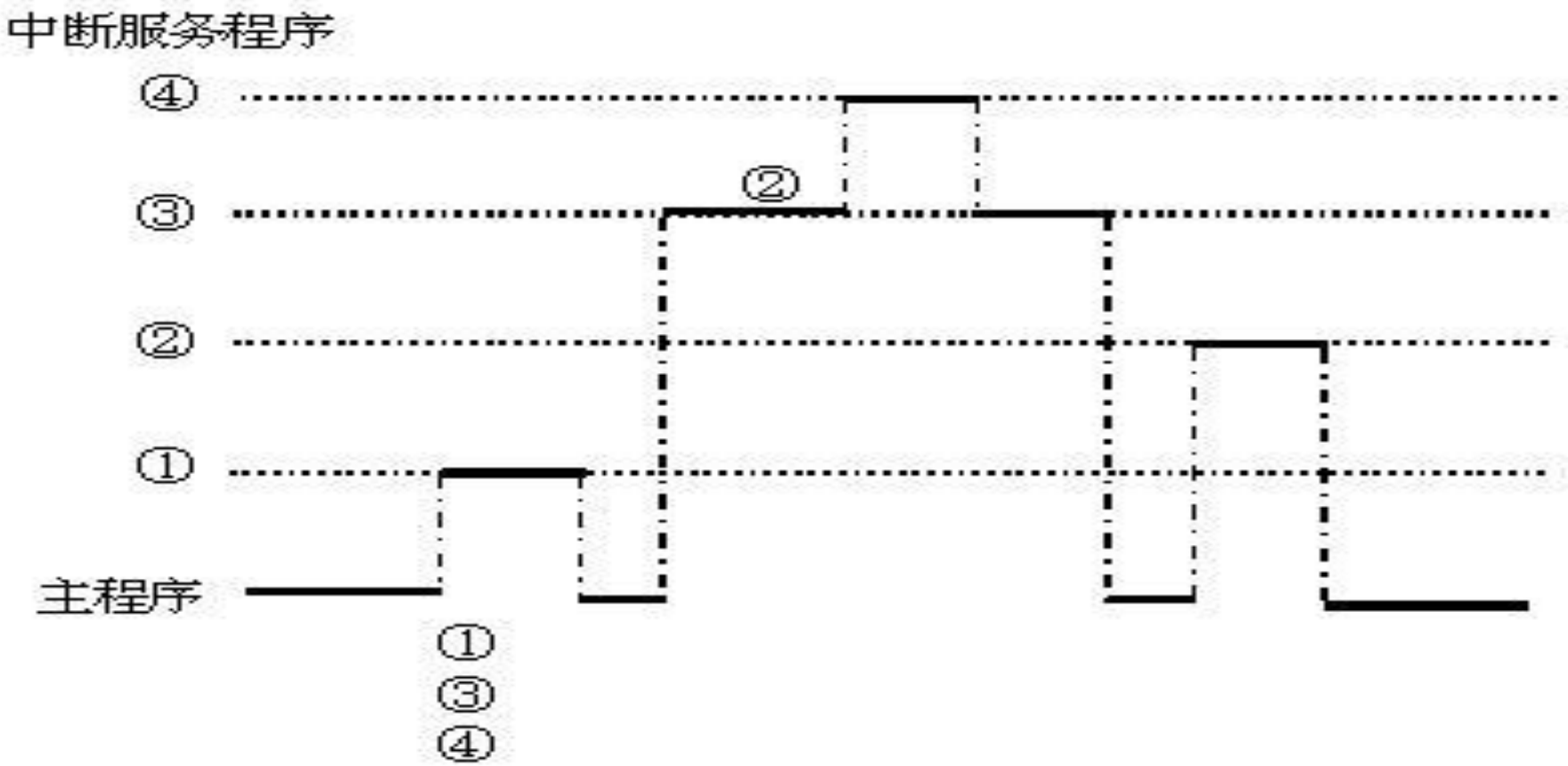
中断程序级别	屏蔽字			
	1级	2级	3级	4级
第1级	1	1	1	1
第2级	0	1	0	0
第3级	0	1	1	0
第4级	0	1	1	1

（假定1是屏蔽，0是开放）

中断优先权的动态分配

- 举例：假定某中断系统有四个中断源，其响应优先级为1>2>3>4。假定在用户程序时同时发生1、3、和4级中断请求，执行3级中断服务程序时发生2级中断请求。分别写出处理优先级为1>2>3>4和1>4>3>2时各中断的屏蔽字及CPU完成中断处理的过程。

(2) 中断处理优先级为1>4>3>2时：



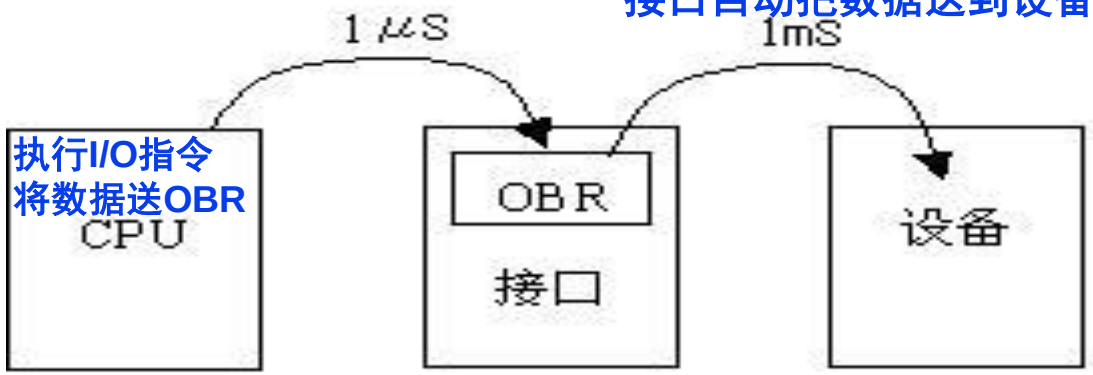
轮询方式和中断方式的比较举例

假定某机控制一台设备输出一批数据。数据由主机输出到接口的数据缓冲器(OBR)，需要1μs(1微秒,10^{-6}秒)。再由OBR输出到设备，需要1ms(毫秒,10^{-3}秒)。设一条指令的执行时间为1μs(包括隐指令)。试计算采用程序(轮询)传送方式和中断传送方式的数据传输速度和对主机的占用率。

CPU如何把数据送到缓冲器OBR?CPU执行I/O指令来将数据送OBR；

I/O接口如何把缓冲器OBR中的数据送到设备?I/O接口则是自动把数据送到设备

接口自动把数据送到设备



主机占用率：在进行I/O操作过程中，处理器花多少时间在输入/出操作上

数据传送速度（吞吐量、I/O带宽）：单位时间内传送的数据量。

假定每个数据的传送都要重新启动！即是字符型设备

轮询方式和中断方式的比较

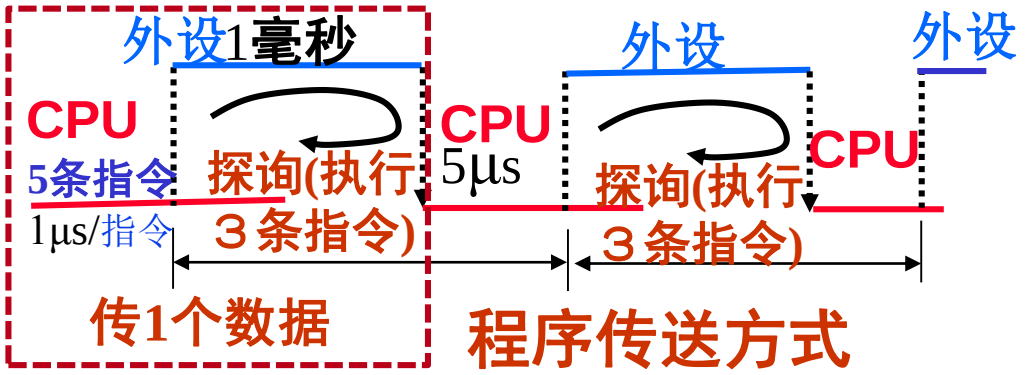
数据由主机输出到接口的数据缓冲器要1微秒，由缓冲器输出到设备要1毫秒

（1）程序直接控制传送方式

若查询程序(红线)有10条，第5条为启动设备的指令，则：

数据传输率为：(1个数据)/((1000+5)μs)，约为每秒995个数据。

主机占用率=100%

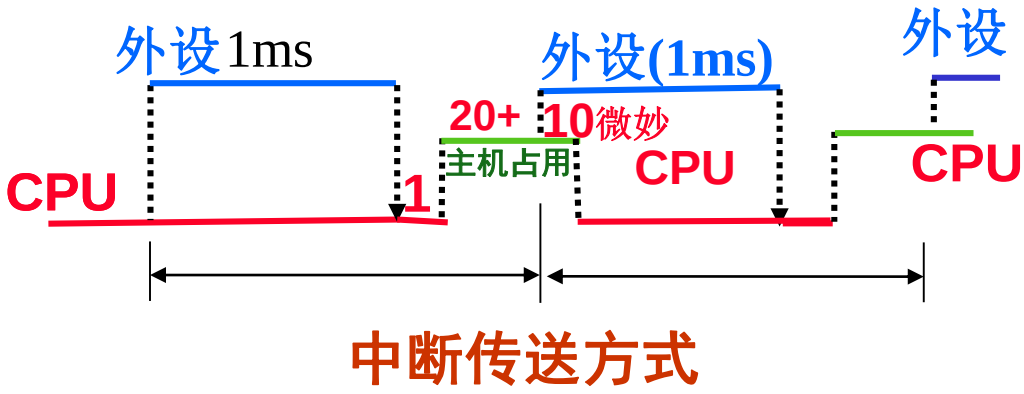


（2）中断传送方式

若中断服务程序(绿线)有30条，在第20条启动设备，则：

数据传输率为：
1/(1000+1+20)μs，约为每秒979个数据。

主机占用率为：
(1+30)/(1000+1+20)=3%



为什么中断服务程序比查询程序长？因为中断服务程序有额外开销，如：保存现场、保存旧屏蔽字、开中断、（查询中断源）等

DMA输入/出方式

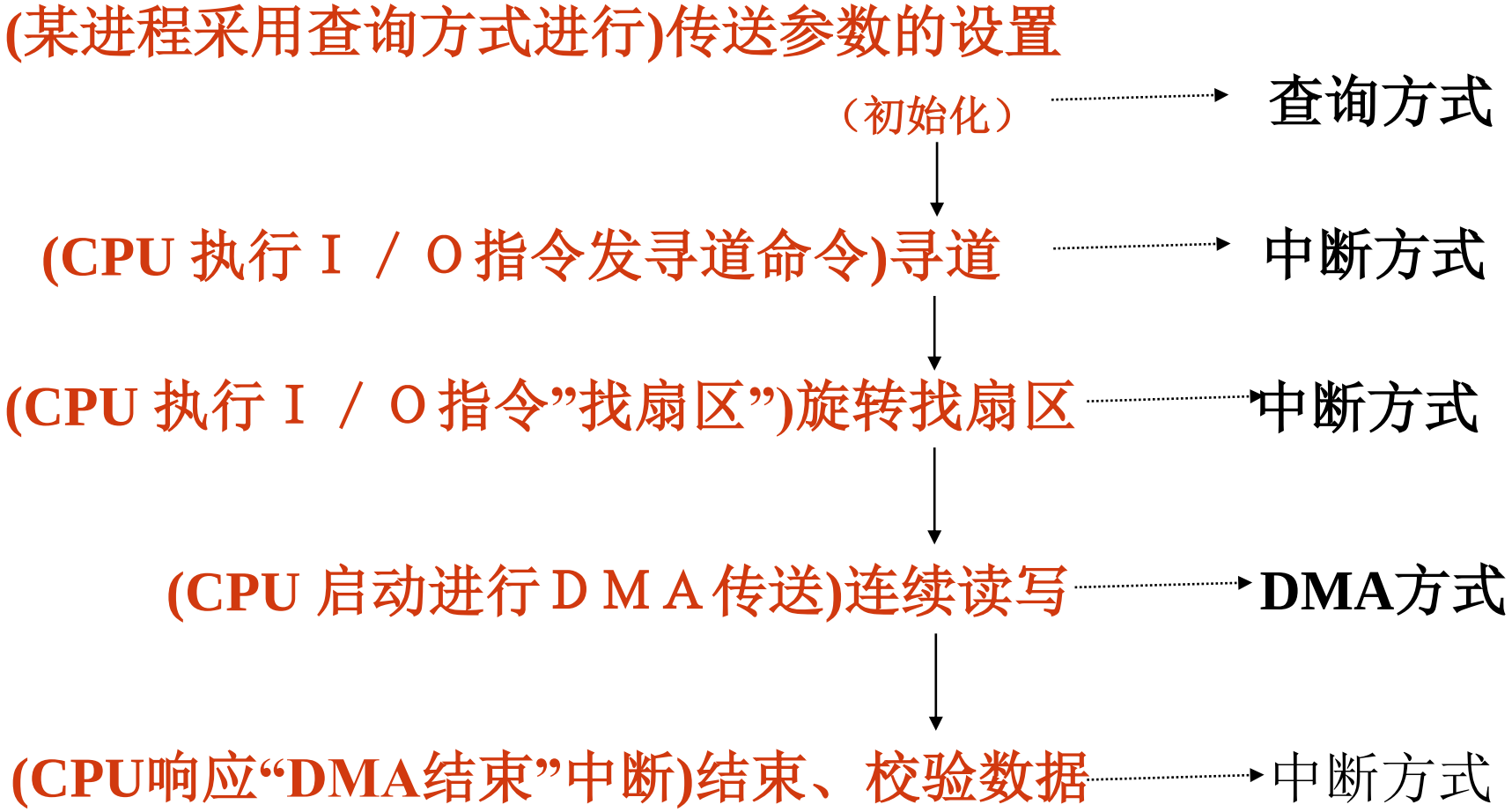
- DMA的全称Direct Memory Access, 直接存储器存取
- 为什么要引入DMA方式？
  - 程序直接控制方式(如轮询法)受“踏步”现象的限制，效率低下，不适合高速设备和主机间的数据传送。
  - 中断控制方式虽比程序直接控制方式有效，CPU和外设有一定的并行度，但由于下列原因也不适合高速设备和主机间的数据传送
    - 对I/O请求响应慢。每传送一个数据都要等待外设的中断请求，并增加许多中断响应和中断处理前、后的附加开销（保护断点、现场等），不能及时响应I/O请求。
    - 数据传送速度慢。数据传送由软件完成（由CPU执行相应的中断服务程序来完成)，速度慢 。

DMA方式的基本要点

- DMA方式的基本思想
  - 在高速外设和主存间直接传送数据
  - 由专门硬件（即：DMA接口）控制总线进行传输
- DMA方式适用场合
  - 高速设备（如：磁盘、光盘等）
  - 成批数据交换，且数据间间隔时间短，一旦启动，数据连续读写
- 采用“请求-响应”方式
  - 每当高速设备准备好数据，就进行一次“DMA请求”，DMA控制器接受到DMA请求后，向CPU申请(请求)总线使用权
  - DMA控制器的对总线的使用优先级比CPU的高，为什么？
- 与中断控制方式结合使用
  - DMA从硬盘传送数据到主存前，“寻道”“旋转”等操作结束时，通过“中断”告知CPU
  - 在DMA控制器控制总线进行数据传送时，CPU执行其他程序
  - DMA传送结束时，通过”DMA结束中断”告知CPU再对相应数据处理

与中断控制方式结合使用

- 举例：用于磁盘和主存间数据交换时



DMA数据传送方式

由于DMA接口和CPU共享主存，所以可能出现两者争用主存的现象，为使两者协调使用主存，DMA通常采用以下三种方式进行数据传送。

(1) CPU停止法(成组传送)

DMA传输时，CPU脱离总线，停止访问主存，直到DMA传送一块数据结束。

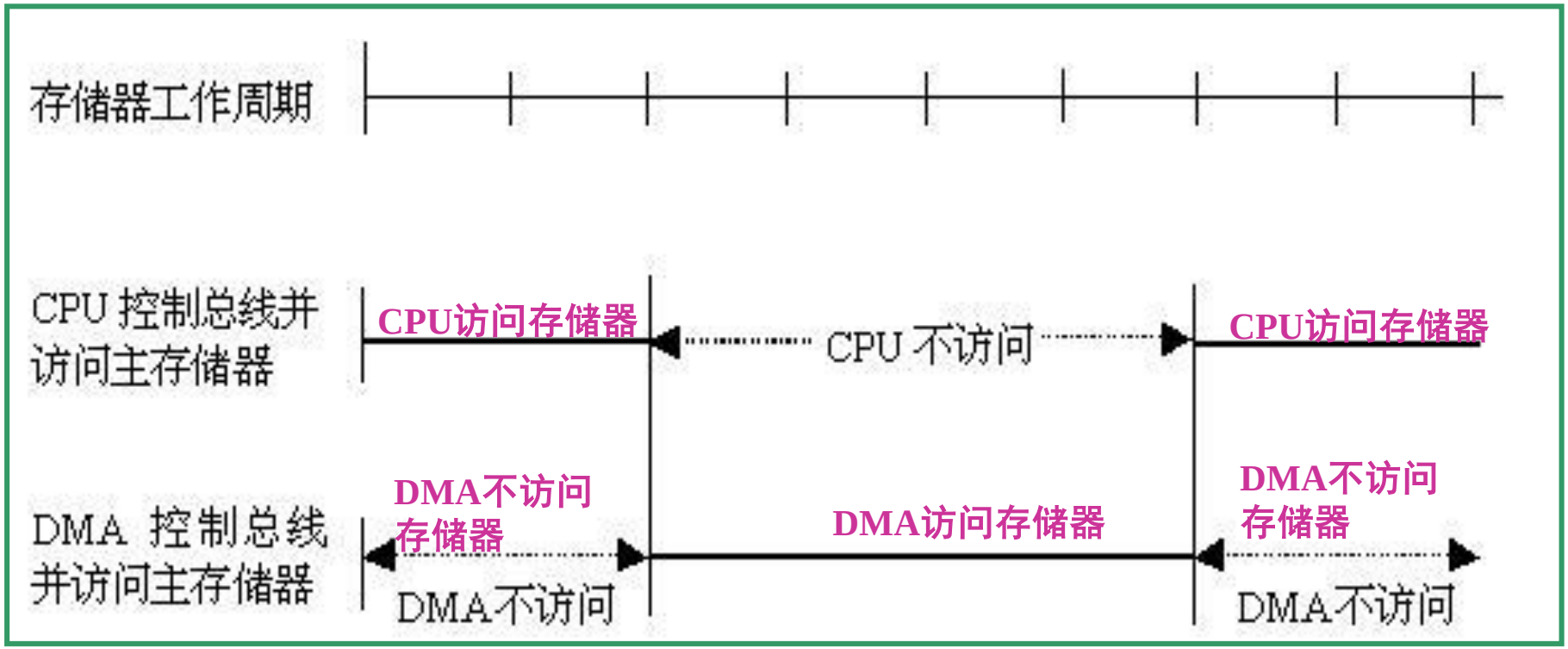
(2) 周期挪用(窃取)法(单字传送)

DMA传输时，CPU让出一个总线事务周期，由DMA控制总线来访问主存，传送完一个数据后立即释放总线。

(3)交替分时访问法

每个存储周期分成两个时间片，一个给CPU，一个给DMA，这样在每个存储周期内，CPU和DMA都可访问存储器。

CPU停止法



优点： 控制简单、适用于传输率很高的外设实现成组数据传送。

缺点： CPU工作受影响。DMA访存时,CPU基本上处于停止状态。主存周期没有被充分利用。即使I/O设备高速运行，但两个数据之间的准备间隔时间也总大于一个存储周期，所以主存周期没有被充分利用。

弥补CPU停止法缺点的做法

在DMA接口中引入缓冲器

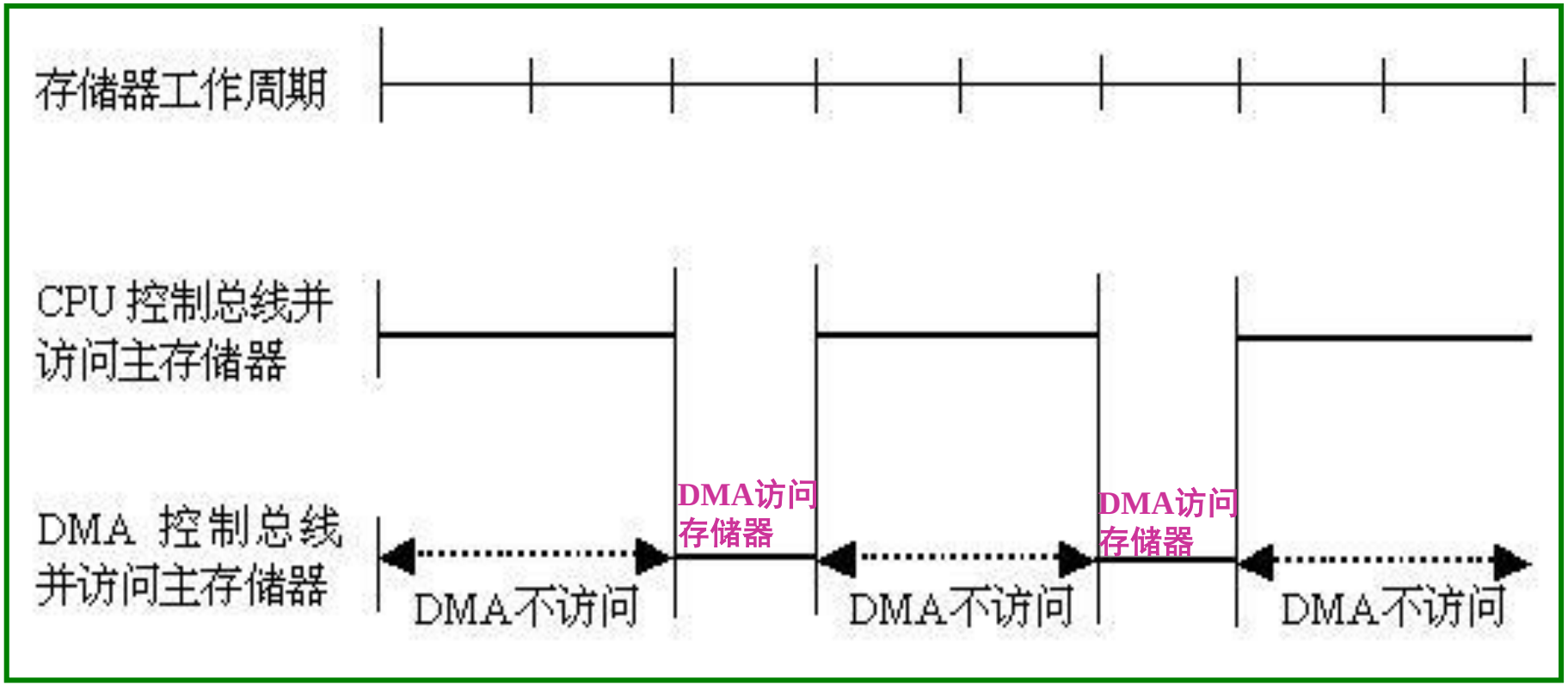
在DMA接口中采用一个小容量的半导体存储器，使I/O设备先和这个小容量存储器交换数据，然后再使用总线由小容量存储器与主存进行数据交换。这样可减少DMA传送数据时占用总线的时间，也就减少了CPU的等待时间。

采用周期挪用（窃取）法

挪用一个存储周期进行外设和主存的一个数据交换。

每次DMA传送完一个数据就释放总线，使在外设准备下一数据时，CPU能插空访问主存。

周期挪用法



优点：既能及时响应I/O请求，又能较好地发挥CPU和主存的效率。这种方式下，在下一数据的准备阶段，主存周期被CPU充分利用。因此适合于I/O设备的读写周期大于主存周期的情况。

缺点：每次DMA访存都要申请总线控制权、占用总线进行传送、释放总线，因此，会增加传输开销。

周期挪用法-I/O设备要求DMA传送时可能会遇到的情况

- CPU不需访问主存

  此时，不会发生冲突，两者并行。

  如: CPU正在执行乘法指令，要花很长时间而不需马上访存。

- CPU正在访问主存

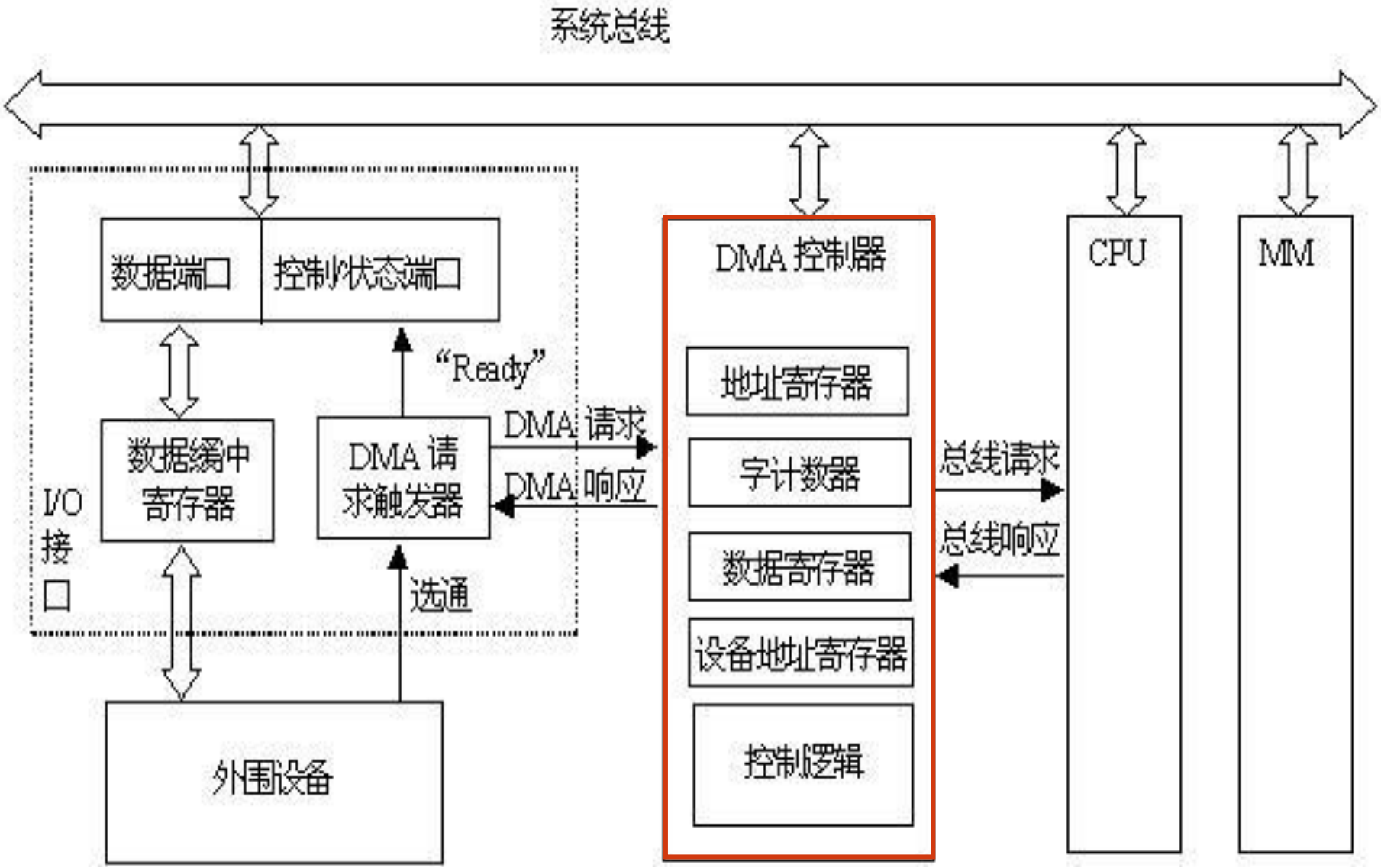
  此时须等到存储周期结束，CPU让出总线，DMA才能访存。

- CPU也同时要访问主存

  此时出现访存冲突。因为不马上响应DMA请求的话，高速设备可能会发生数据丢失，所以，DMA的总线优先权比CPU高。这时，先让DMA占用总线，窃取一个主存周期，完成一个数据的交换。这样，CPU便要延迟一段时间才能访存。

DMA方式结构

- DMA方式下的系统逻辑结构



DMA控制器的功能

一个DMA控制器，实际上是采用DMA方式的外围设备与系统总线之间的接口电路，这个接口电路是在中断接口的基础上再加DMA机构组成。习惯上将DMA方式的接口电路称为DMA控制器。DMA数据传送过程由DMA接口的控制逻辑完成，DMA 接口也称DMA控制器,其功能为：

(1) 请求。能接收外设发来的“DMA请求”信号，并能向CPU发“总线请求”信号。

(2) 响应。当CPU发出“总线响应”信号响应请求后，能接管对总线的控制。

(3) 修改主存地址。能在地址线上给出主存地址，并自动修改主存地址。

(4) 识别传送方向。能识别传送方向以在控制线上给出正确的读写控制信息。

(5) 确定传送数据个数。

(6) 能发出DMA结束信号。引起一次DMA中断，进行数据校验等一些后处理。

DMA控制器的操作步骤

- DMA操作步骤

  (1) DMA控制器的预置(初始化)----软件实现

  - 准备内存
  - 设置参数
  - 启动外设

  (2) DMA数据传送----硬件实现

  - DMA请求：选通-〉DMA请求-〉总线请求
  - DMA响应：总线响应(CPU让出总线)-〉DMA响应
  - DMA传送：DMA控制总线进行数据传送

  (3) DMA结束处理----软件实现

  根据计数值为“0”，发出DMA结束信号去接口控制产生DMA中断请求信号，转入中断服务程序，做一些数据校验等后处理工作。

SKIP

DMA控制器的初始化

- DMA控制器的预置(初始化)
  - 准备内存区
    - * 输入：在内存设置好缓冲区
    - * 输出：先在内存准备好数据
  - 设置传送参数

    执行I/O指令，测试外设状态，对DMA控制器设置各种参数：
    - * 内存首址=〉地址寄存器
    - * 字计数值=〉字计数器
    - * 传送方向=〉控制寄存器
    - * 设备地址=〉设备地址寄存器
  - 启动外设，然后执行其他程序

BACK

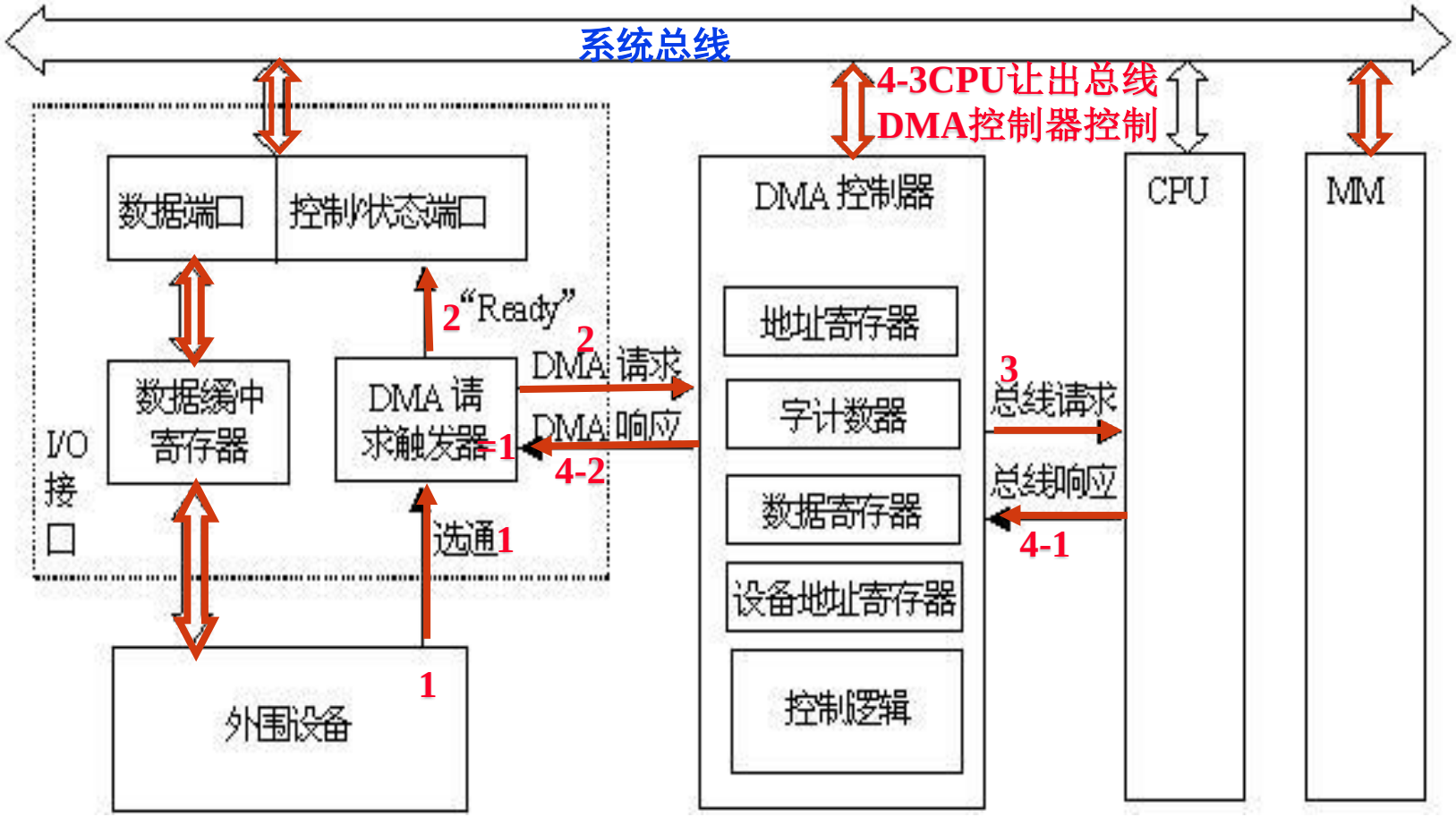
DMA的结构与传输过程

(1) 当外设准备好数据，或准备好接收数据时，发“选通”信号，使数据送数据缓冲寄存器，同时 DMA请求触发器置“1”。

(2) DMA请求触发器向控制/状态端口发“Ready”信号，同时向DMA控制器发“DMA请求”信号。

(3) DMA控制器向CPU发“总线请求”信号。

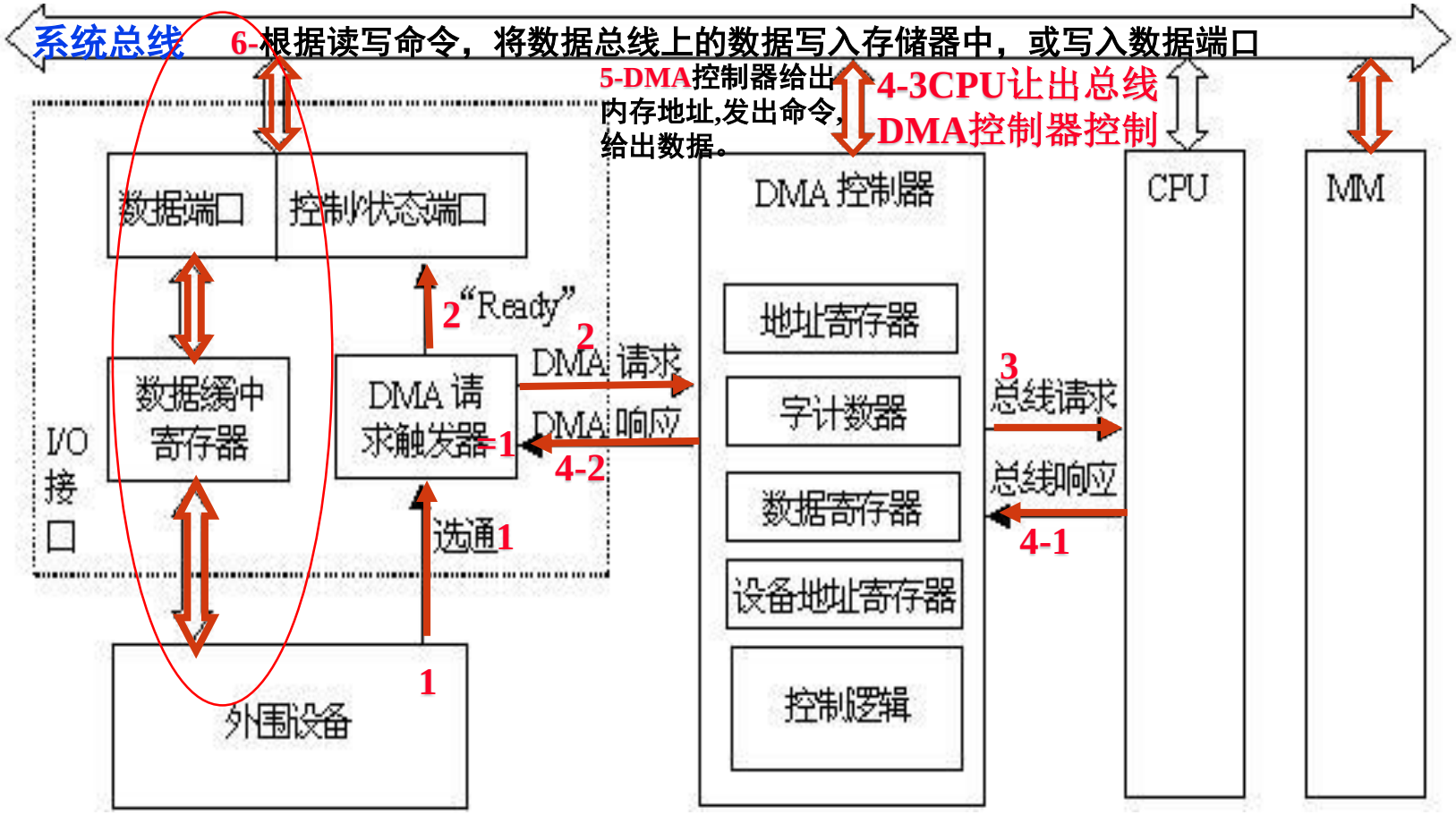
(4) CPU完成现行机器周期后，响应DMA请求，发出“总线响应”信号。DMA控制器接受到该信号后，发出“DMA响应”信号，使DMA请求触发器复位。此时，CPU浮动它的总线，让出总线控制权，由DMA控制器控制总线。



(4) CPU完成现行机器周期后，响应DMA请求，发出“总线响应”信号。DMA控制器接受到该信号后，发出“DMA响应”信号，使DMA请求触发器复位。此时，CPU浮动它的总线，让出总线控制权，由DMA控制器控制总线。

(5)DMA控制器给出内存地址，并在其读/写线上发出”读”或”写”命令，随后在数据总线上给出数据。

(6)根据读写命令，将数据总线上的数据写入存储器中，或写入数据端口，并进行主存地址增量，计数值减1，若采用“CPU停止法”，则循环第6步，直到计数值为“0”。若采用“周期挪用法”，则释放总线（下次数据传送时再按过程(1)到(6)进行）。



DMA传输过程

(1) 当外设准备好数据，或准备好接收数据时，发“选通”信号，使数据送数据缓冲寄存器，同时DMA请求触发器置“1”。

(2) DMA请求触发器向控制/状态端口发“Ready”信号，同时向DMA控制器发“DMA请求”信号。

(3) DMA控制器向CPU发“总线请求”信号。

(4) CPU完成现行机器周期后，响应DMA请求，发出“总线响应”信号。DMA控制器接受到该信号后，发出“DMA响应”信号，使DMA请求触发器复位。此时，CPU浮动它的总线，让出总线控制权，由DMA控制器控制总线。

(5) DMA控制器给出内存地址，并在其读/写线上发出“读”或“写”命令，随后在数据总线上给出数据。

(6) 根据读写命令，将数据总线上的数据写入存储器中，或写入数据端口，并进行主存地址增量，计数值减1，

若采用“CPU停止法”，则循环第6步，直到计数值为“0”。

若采用“周期挪用法”，则释放总线（下次数据传送时再按过程(1)到(6)进行）。

BACK

DMA方式和中断方式的区别

(1) DMA方式下数据传送由硬件(DMA控制器)完成；中断方式下，数据传送由软件（CPU执行中断服务程序）完成。

(2) DMA请求的是对存储器访问，也即对总线控制权的请求，没有中止现行程序的必要；而中断请求要处理器转去执行中断服务程序，因此要中止现行程序，保存断点、现场等。

(3) 中断除了能完成外设和主机的数据交换，还能处理异常事件；而DMA方式下不能处理异常事件。

(4) 中断响应在一个指令周期结束后；DMA响应是在一个总线周期后

(5) DMA方式用于高速设备；而中断方式用于低、慢速设备。

(6) DMA方式下，外设与CPU并行度高；而中断方式下，外设与CPU并行度低。（体现在数据传送时的并行性）

例：中断、DMA方式下硬盘数据传输时CPU的开销

设处理器按500MHz的速度执行，硬盘控制器中有一个16B的数据缓存器，磁盘传输速率为4MB/Sec，在磁盘传输数据过程中，要求没有任何数据被错过，并假定CPU访存和DMA访存没有冲突

（1）若用中断驱动I/O，每次传送的开销（包括用于中断响应和处理的时间）是500个时钟周期。如果硬盘仅用5%的时间进行传送，那么处理器用在硬盘I/O操作上所花的时间百分比（主机占用率）为多少？

（2）若用DMA方式，处理器花1000个时钟进行DMA传送的初始化设置，并且在DMA完成后的中断处理需要500个时钟。如果每次DMA传送8000B的数据块，那么当硬盘进行传送的时间占100%（即：硬盘一直进行读写，并传输数据）时，处理器用在硬盘I/O操作上的时间百分比（主机占用率）为多少？

想象一下：假定大仓库门口有一个箱子，可放16个零件。要将大仓库中的一批零件运到小仓库中，可以有几种方法？

中断方式：每装满一个箱子就喊车床上的技工来运到车间，再从车间运到小仓库

DMA方式：车床技工停下来告诉搬运工说，一次要8000个零件放到小仓库固定的地方，然后回到车床工作；搬运工开始分两组工作，一组从大仓库搬货到箱子中，另一组将箱子直接运到小仓库指定地方，8000个运完后，技工再停下来检查；然后继续下一次8000个零件的搬运，…

上述两种方式中，哪种方式的生产效率更高呢？

例：中断方式下硬盘数据传输时CPU的开销

设处理器按500MHz的速度执行(每秒产生500x106 个时钟周期)，硬盘控制器中有一个16B的数据缓存器，磁盘传输速率为4MB/Sec，在磁盘传输数据过程中，要求没有任何数据被错过，并假定CPU访存和DMA访存没有冲突

(1)若用中断驱动I/O，每次传送的开销（包括用于中断响应和处理的时间）是500个时钟周期。如果硬盘一直进行读写，并传输数据,处理器用在硬盘I/O操作上所花的时间百分比（主机占用率）为多少？如果硬盘仅用5%的时间进行传送，那么处理器用在硬盘I/O操作上所花的时间百分比（主机占用率）为多少？

中断传送：

- 硬盘每次中断，传送16个字节(为1次传送). 每秒应该中断的次数= (4MB /16B) = 250k次,才能保证没有任何数据被错过.
- 每秒钟用于(传送数据的)中断的时钟周期数为(250k)x500=125x10^6 时钟周期
- 假定硬盘一直进行读写，并传输数据,每秒钟用于 (传送数据的)中断的时钟周期数为 (125x10^6) 时钟周期,处理器花费在I/O上的时间(时钟周期)占总的CPU时间的百分比为：(125x10^6)/500MHz=(125x10^6)/(500x10^6) = 25%
- 假定硬盘仅用其中5%的时间来传送数据，则每秒钟用于 (传送数据的)中断的时钟周期数为 ((125x10^6) x 5%) 时钟周期,花费在I/O上的时间占总的CPU时间的百分比为:((125x10^6)x5%)/500MHz=((125x10^6)x5%)/(500x10^6)= 1.25%

例：DMA方式下硬盘数据传输时CPU的开销

设处理器按500MHz的速度执行，硬盘控制器中有一个16B的数据缓存器，磁盘传输速率为4MB/Sec，在磁盘传输数据过程中，要求没有任何数据被错过，并假定CPU访存和DMA访存没有冲突

（2）若用DMA方式，处理器花1000个时钟进行DMA传送的初始化设置，并且在DMA完成传输数据后的中断处理需要500个时钟。如果每次DMA传送8KB(=8*1024B)的数据块，那么当硬盘进行传送的时间占100%（即：硬盘一直进行读写，并传输数据）时，处理器用在硬盘I/O操作上的时间百分比（主机占用率）为多少？

- DMA传送：
  - 每次DMA传送将花费: (8KB数据块) / (4MB/Sec) ≈ $2x10^{-3}$秒；
  - 一秒钟内有$1/(2x10^{-3})$=500次DMA传送；
  - 如果硬盘一直在传送数据的话，处理器每秒钟必须花(500次)x (1000时钟周期准备+500时钟周期后处理)=500X(1000+500) =$750x10^3$个时钟周期来为硬盘I/O操作服务；
  - 在硬盘I/O操作上处理器花费的时间占：

(750x10^3时钟周期)/(500MHz)= (750x10^3)/(500x10^6)=1.5x10^{-3}=0.15%

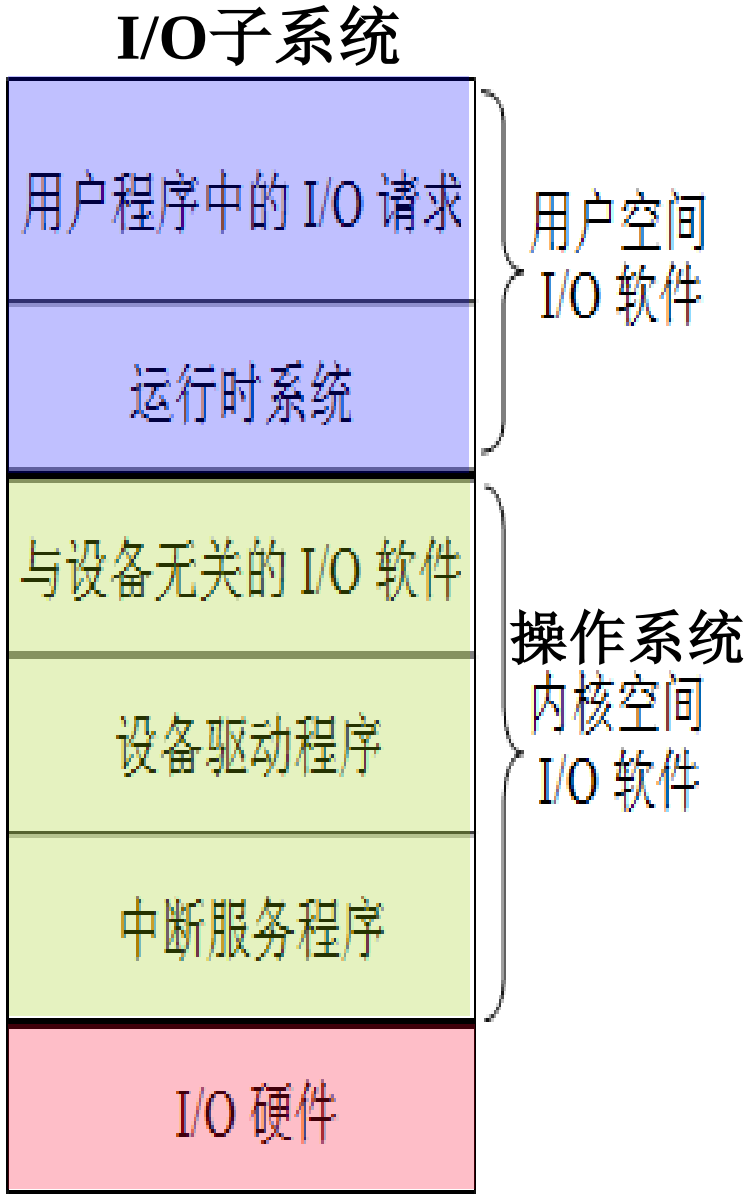
上述两种方式中，哪种方式的生产效率更高呢？DMA

I/O子系统概述

- I/O子系统由I/O硬件和I/O软件组成
- 所有高级语言的运行时（runtime）都提供了执行I/O功能的机制

  例如，C语言中提供了包含像printf()和scanf()等这样的标准I/O库函数，C++语言中提供了如 <<（输入）和 >>（输出）这样的重载操作符。
- 从高级语言程序中通过I/O函数或I/O操作符提出I/O请求，到设备响应并完成I/O请求，涉及到多层次I/O软件和I/O硬件的协作。
- I/O子系统也采用层次结构

从用户I/O软件切换到操作系统内核I/O软件的唯一办法是“异常”机制：系统调用（自陷）



I/O子系统概述

各类用户的I/O请求需要通过某种方式传给操作系统（OS）：

- 最终用户：键盘、鼠标通过操作界面传递给OS
- 用户程序：通过函数（高级语言）转换为系统调用传递给OS

I/O软件被组织成从高到低的四个层次，层次越低，则越接近设备而越远离用户程序。这四个层次依次为：

(1) 用户层I/O软件（I/O函数调用系统调用）

(2) 与设备无关的操作系统I/O软件

(3) 设备驱动程序

(4) I/O中断处理程序

OS在I/O系统中极其重要！
OS部分

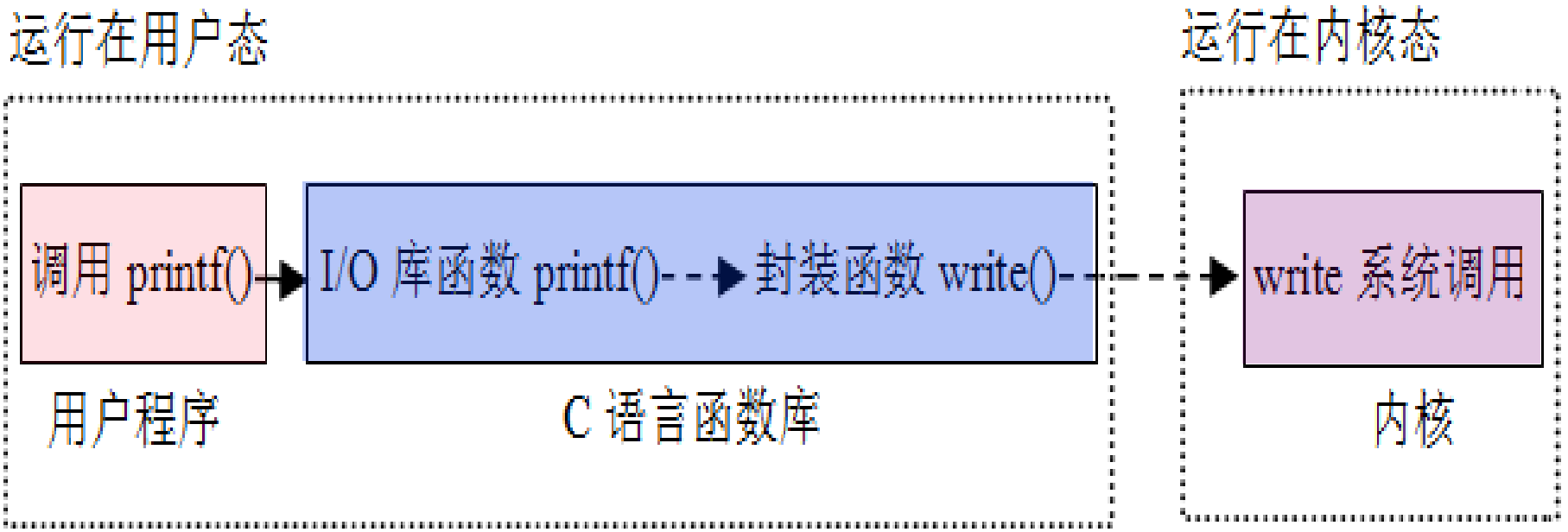
大部分I/O软件都属于操作系统内核态程序，最初的I/O请求在用户程序中提出。

用户程序、C函数和内核

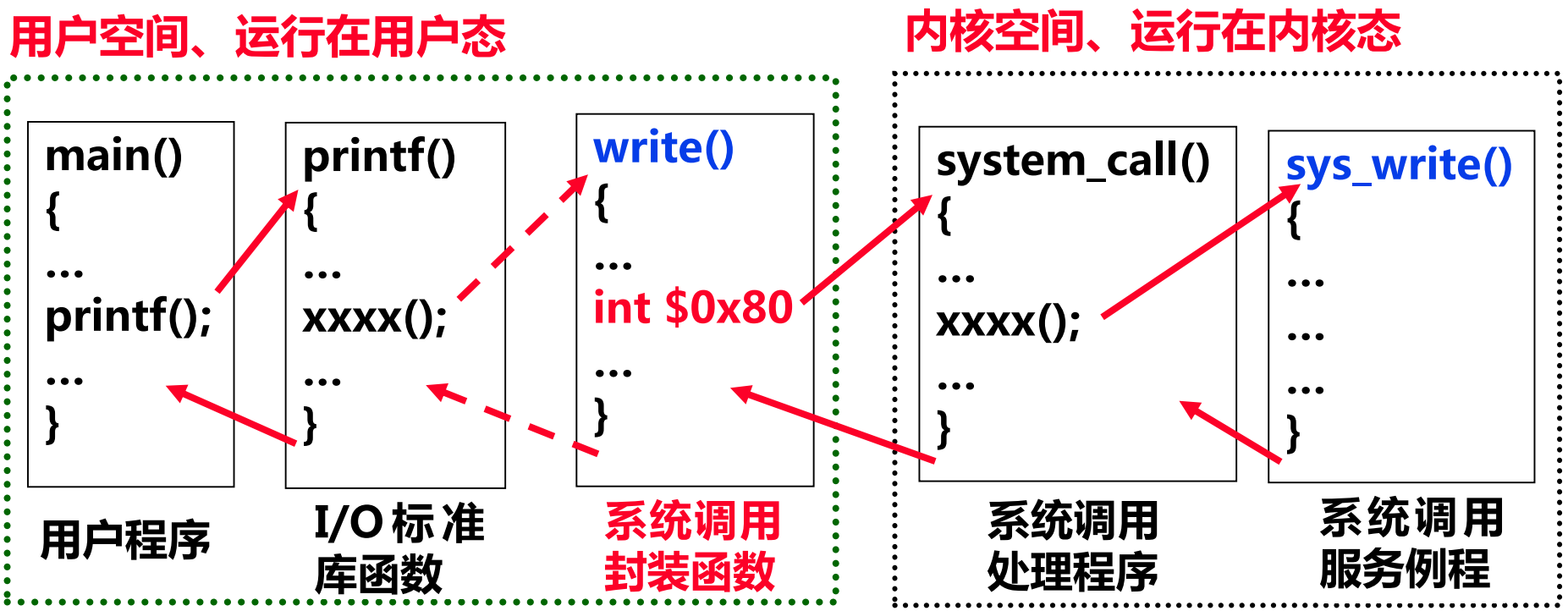
- 用户程序总是通过某种I/O函数或I/O操作符请求I/O操作。

例如，读一个磁盘文件记录时，可调用C标准I/O库函数fread()，也可直接调用系统调用封装函数read()来提出I/O请求。不管是C库函数、API函数还是系统调用封装函数，最终都通过操作系统内核提供的系统调用来实现I/O。

printf()函数的调用过程如下：



Linux系统中printf()函数的执行过程



- 某函数调用了printf()，执行到调用printf()语句时，便会转到C语言I/O标准库函数printf()去执行；
- printf()通过一系列函数调用，最终会调用函数write()；
- 调用write()时，便会通过一系列步骤在内核空间中找到write对应的系统调用服务例程sys_write来执行。

在system_call中如何知道要转到sys_write执行呢？根据系统调用号！

程序查询（Polling）方式

- I/O设备（包括设备控制器）将自己的状态放到状态寄存器中
  - 打印缺纸、打印机忙、未就绪等都是状态
- OS阶段性地查询状态寄存器中的特定状态，以决定下一步动作
  - 如：未“就绪”时，则一直“等待”
- 例如：sys_write进行字符串打印的程序段大致过程如下：

```
copy_string_to_kernel ( strbuf, kernelbuf, n);  // 将字符串复制到内核缓冲区
for (i=0; i < n; i++) {                          // 对于每个打印字符循环执行
  while ( printer_status != READY);              // 等待直到打印机状态为“就绪”
  *printer_data_port=kernelbuf[i];               // 向数据端口输出一个字符
  *printer_control_port=START;                   // 发送“启动打印”命令
}
return_to_user ( );                              // 返回用户态
```

如何判断“就绪”？如何“等待”？

读取状态寄存器，判断特定位（1-就绪；0-未就绪）是否为1

等待：读状态、判断是否为1；不是，则继续读状态、判断、……

中断I/O方式

例子：采用中断方式进行字符串打印

sys_write进行字符串打印的程序段:

```
copy_string_to_kernel ( strbuf, kernelbuf, n);// 将字符串复制到内核缓冲区
enable_interrupts ( );                        // 开中断，允许外设发出中断请求
while ( printer_status != READY);             // 等待直到打印机状态为“就绪”
*printer_data_port=kernbuf[i];                // 向数据端口输出第一个字符
*printer_control_port=START;                  // 发送“启动打印”命令
scheduler ( );                                // 阻塞用户进程P，调度其他进程执行
```

“字符打印”中断服务程序：

```
if (n==0) {                                   // 若字符串打印完，则
unblock_user ( );                             // 用户进程P解除阻塞，P进就绪队列
} else {
   *printer_data_port=kernelbuf[i];           // 向数据端口输出一个字符
   *printer_control_port=START;               // 发送“启动打印”命令
    n = n-1;                                  // 未打印字符数减1
    i = i+1;                                  // 下一个打印字符指针加1
}
acknowledge_interrupt();                      // 中断回答（清除中断请求）
return_from_interrupt();                      // 中断返回
```

sys_write是如何调出来的？

系统调用！

中断服务程序是如何调出来的？

外设完成任务！

DMA方式

例子：采用DMA方式进行字符串输出

sys_write进行字符串输出的程序段:

```
copy_string_to_kernel(strbuf, kernelbuf, n);  // 将字符串复制到内核缓冲区
initialize_DMA ( );                           // 初始化DMA控制器（准备传送参数）
*DMA_control_port=START;                      // 发送“启动DMA传送”命令
scheduler ( );                                // 阻塞用户进程P，调度其他进程执行
```

DMA控制器接受到“启动”命令后，控制总线进行DMA传送。通常用”周期挪用法“：设备每准备好一个数据，挪用一次”存储周期“，使用一次总线事务进行数据传送，计数器减1。计数器为0时，发送DMA结束中断请求

” DMA结束“中断服务程序：

```
acknowledge_interrupt();      // 中断回答（清除中断请求）
unblock_user ( );             // 用户进程P解除阻塞，进入就绪队列
return_from_interrupt();      // 中断返回
```

CPU仅在DMA控制器初始化和处理“DMA结束中断“时介入，在DMA传送过程中不参与，因而CPU用于I/O的开销非常小。

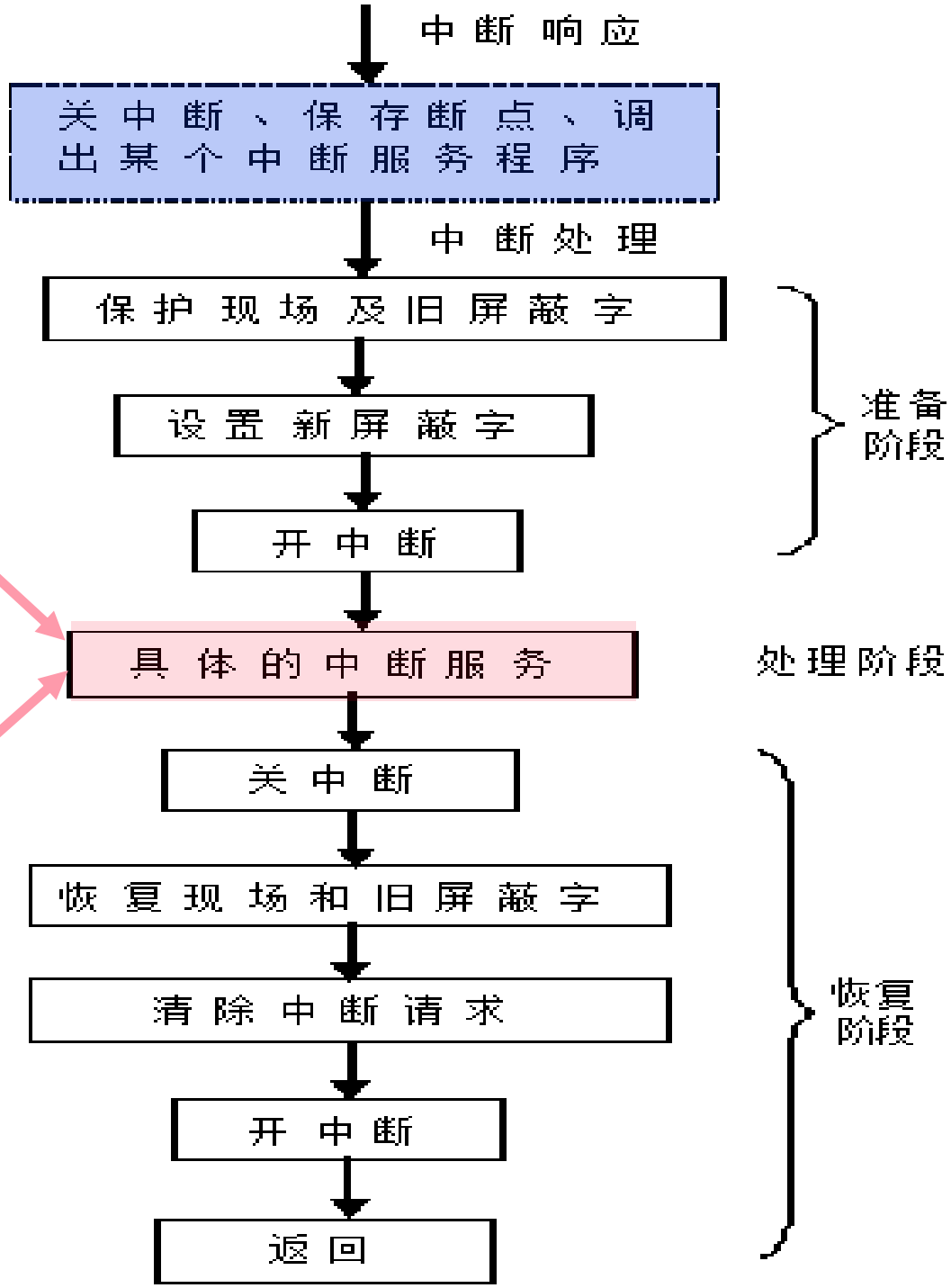
设备驱动程序

- 每个外设具体的I/O操作需通过执行设备驱动程序来完成
- 外设种类繁多、其控制接口不一，导致不同外设的**设备驱动程序千差万别**，因而设备驱动程序与设备相关
- 每个外设或每类外设都有一个**设备控制器**，其中包含各种**I/O端口**。 CPU通过执行设备驱动程序中的**I/O指令**访问个各种I/O端口
- 设备所采用的I/O控制方式不同，驱动程序的实现方式也不同
  - **程序直接控制**：驱动程序完成用户程序的I/O请求后才结束。这种情况下，用户进程在I/O过程中不会被阻塞，内核空间的I/O软件一直代表用户进程在内核态进行I/O处理 。（干等！）
  - **中断控制**：驱动程序启动第一次I/O操作后，将调出其他进程执行，而当前用户进程被阻塞。在CPU执行其他进程的同时，外设进行I/O操作，此时，CPU和外设并行工作。外设完成I/O时，向CPU发中断请求，然后CPU调出相应中断服务程序执行。在中断服务程序中再次启动I/O操作。
  - **DMA控制**：驱动程序对DMA控制器初始化后，便发送“启动DMA传送”命令，外设开始进行I/O操作并在外设和主存间传送数据。同时CPU执行处理器调度程序，转其他进程执行，当前用户进程被阻塞。DMA控制器完成所有I/O任务后，向CPU发送一个“DMA完成”中断请求信号。

中断服务程序

- 中断控制和DMA控制两种方式下都需进行中断处理
- **中断控制方式**：中断服务程序主要进行**从数缓器取数或写数据到数缓器**，然后启动外设工作
- **DMA控制方式**：中断服务程序进行**数据校验**等后处理工作

在内核I/O软件中用到的I/O指令、“开中断”和“关中断”等指令都是特权指令，只能在操作系统内核程序中使用



本章总结1

- I/O系统概述
  - I/O系统的性能主要有吞吐率和响应时间，两者是对立统一的关系
  - I/O系统的功能是在主机（寄存器和主存）和外设之间传输数据
  - I/O系统的具体任务是：构建传输通路、对设备寻址、向设备发命令、取状态、并提供相应的传输机制来读/写设备数据等。
  - OS在I/O系统的职责是：
    - 对共享设备进行管理、提供设备驱动程序、处理中断请求
- I/O设备概述
  - I/O设备通过I/O接口和主机相连
  - 外设分类：I/O设备和存储设备、机读设备和人读设备
- 磁盘存储器
  - 磁盘存储器的读写原理：两种不同磁化状态表示“0”和“1”
  - 磁盘存储器的性能指标：寻道时间、旋转等待时间、数据传输率
  - 冗余磁盘阵列 (RAID)：多个物理盘组成一个逻辑盘，以提高磁盘存取速度、容量和可靠性
- 网络：作为一种特殊的外部设备，实现计算机系统之间的数据交换

本章总结2

- I/O接口是I/O控制器以及与外设之间连接的电缆插座（很多教材把I/O控制器和I/O接口综合称为I/O接口，不严格区分控制逻辑部分（I/O控制器）和插座（I/O插口）部分）
- I/O控制器中一般有数据缓冲器、状态/控制寄存器、串-并转换、设备控制逻辑、地址译码逻辑等。用于在主机和设备之间进行命令、数据、状态信息的传递和转换。
- I/O端口是指I/O控制器中CPU可访问的寄存器，对I/O设备的寻址就是对I/O端口的访问
- I/O端口的编址方式有两种：内存映射方式（统一编址）和特殊I/O指令方式（独立编址）

本章总结3

- 有三种基本的I/O传输方式
  - 轮询方式（程序直接控制 / 程序查询方式）：通过查询程序定时到I/O端口取状态来查询外设和I/O控制器的状况，以控制设备进行相应的动作
  - 程序中断方式（中断驱动方式）：当外设完成任务或发生特殊情况时，由外设主动向CPU提出中断请求，CPU在每条指令结束后查询有无中断请求，有则转内核态，调出操作系统中的中断处理(服务)程序来处理外设请求。在中断处理程序中完成CPU和外设的数据传送
    - 中断响应过程（硬件-处理器）：关中断、保护断点、转中断服务程序
    - 中断服务程序的入口地址可以是一个中断查询程序入口，也可以由中断控制器给出中断类型号，再根据类型号到中断向量表中取入口地址
    - 中断处理过程（软件-OS）：准备阶段、处理阶段、恢复并返回阶段
    - 中断控制器：记录所有中断请求，在中断掩码（屏蔽码）的作用下，对所有未被屏蔽的中断请求进行优先权编码，送出优先级最高的中断类型号给CPU
    - 多重中断：在处理中断时又发生新中断请求的处理机制
  - 直接存储器访问（DMA）方式：用于磁盘等高速外设与主存之间直接数据交换
    - DMA控制器结构：字计数器、地址寄存器、设备地址寄存器、控制逻辑等
    - 三种控制方式：CPU停止法、周期挪用法、交替分时访问法
    - DMA传输过程：控制器初始化、数据传输、结束处理

本章作业

◦ 2（7）（8）（10），3、4、5、6、8

10、12、13 、14