



南京大學

NANJING UNIVERSITY

存储器层次结构

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



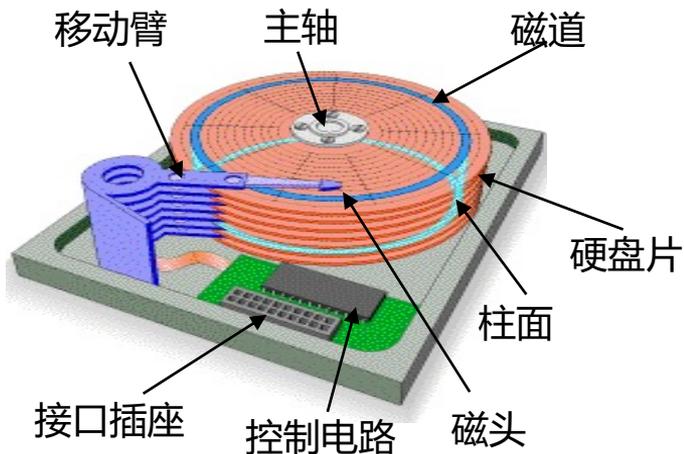
存储器层次结构

- 存储器概述
- 半导体随机存取存储器
- **外部辅助存储器**
- 存储器的数据校验
- 高速缓冲存储器
- 虚拟存储器

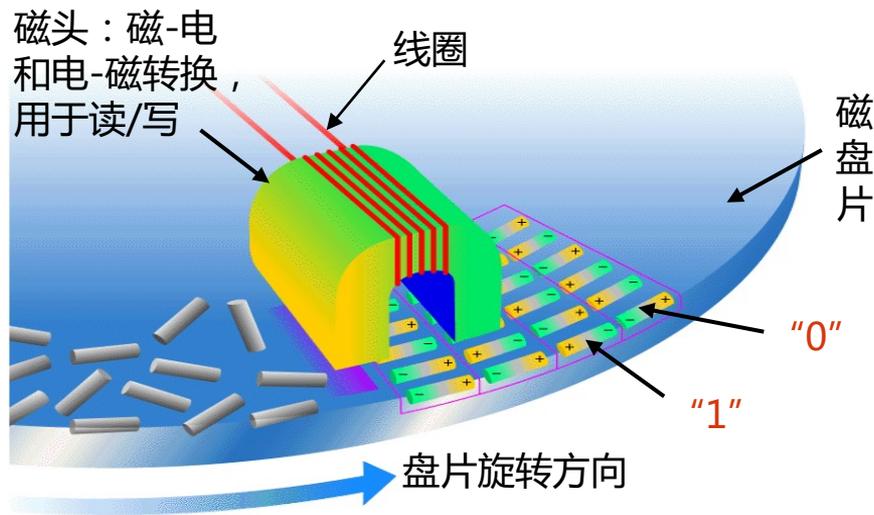




磁盘存储器的信息存储原理



磁盘驱动器的结构



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**

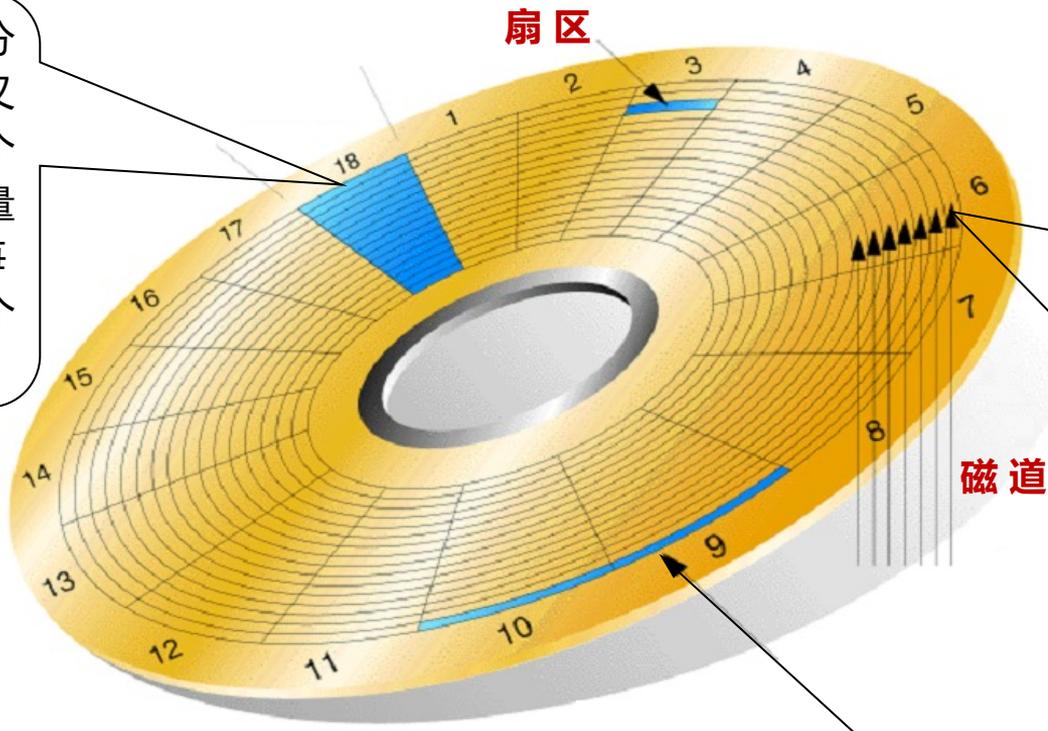
} 不同的磁化状态被记录在磁盘表面

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。



磁盘的磁道和扇区

每个磁道被划分为若干段（段又叫**扇区**），每个扇区的存储容量为512字节。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**。每个磁道都有一个编号，最外面的是**0磁道**

早期扇区大小是512字节。但现在已经迁移到更大、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。



磁盘的内部逻辑结构

• 寻道操作

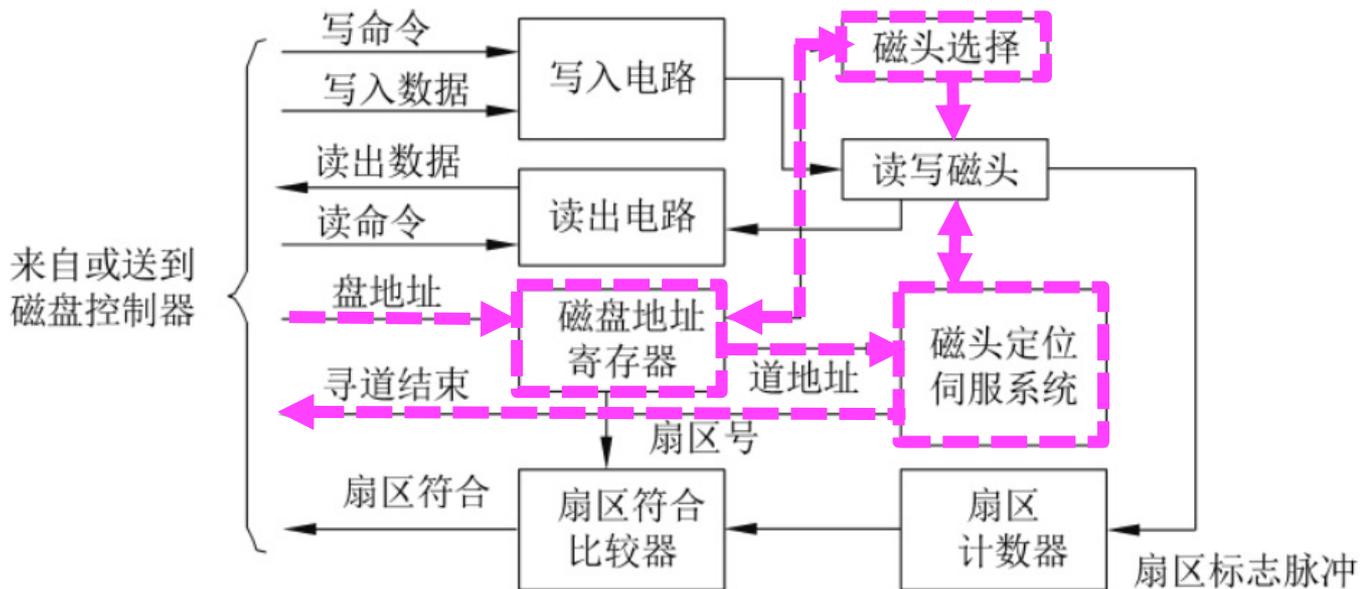


图 7.16 磁盘驱动器的内部逻辑结构



磁盘的内部逻辑结构

- 旋转等待操作

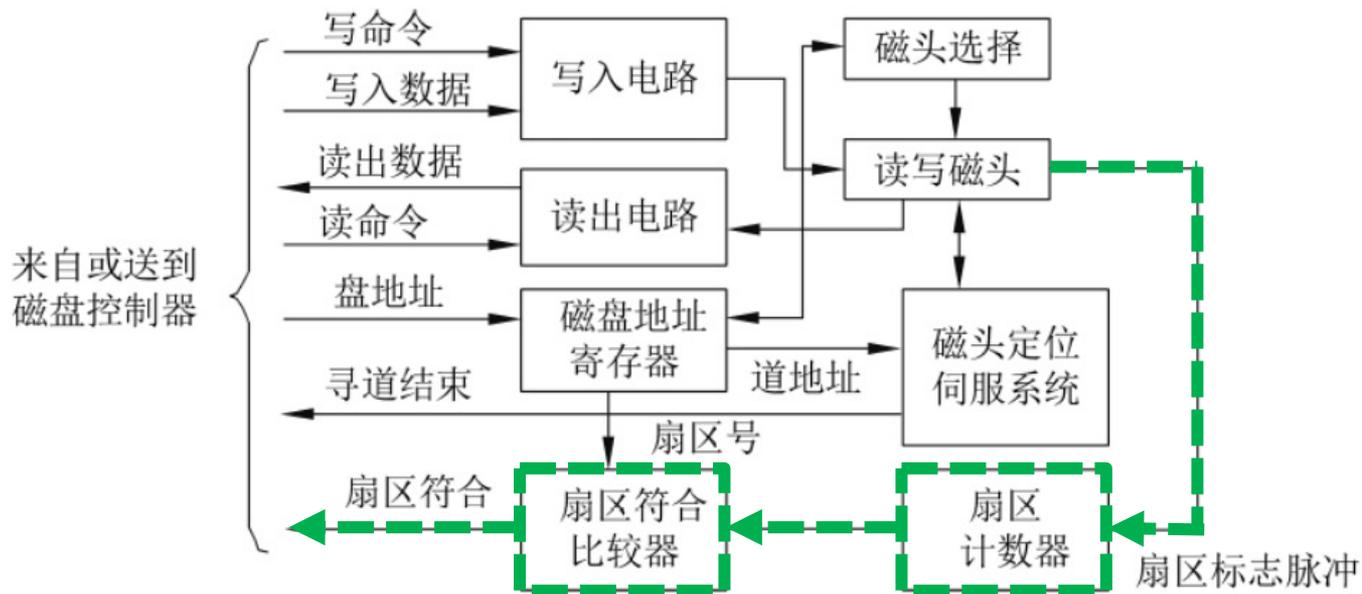


图 7.16 磁盘驱动器的内部逻辑结构



磁盘的内部逻辑结构

- 读写操作

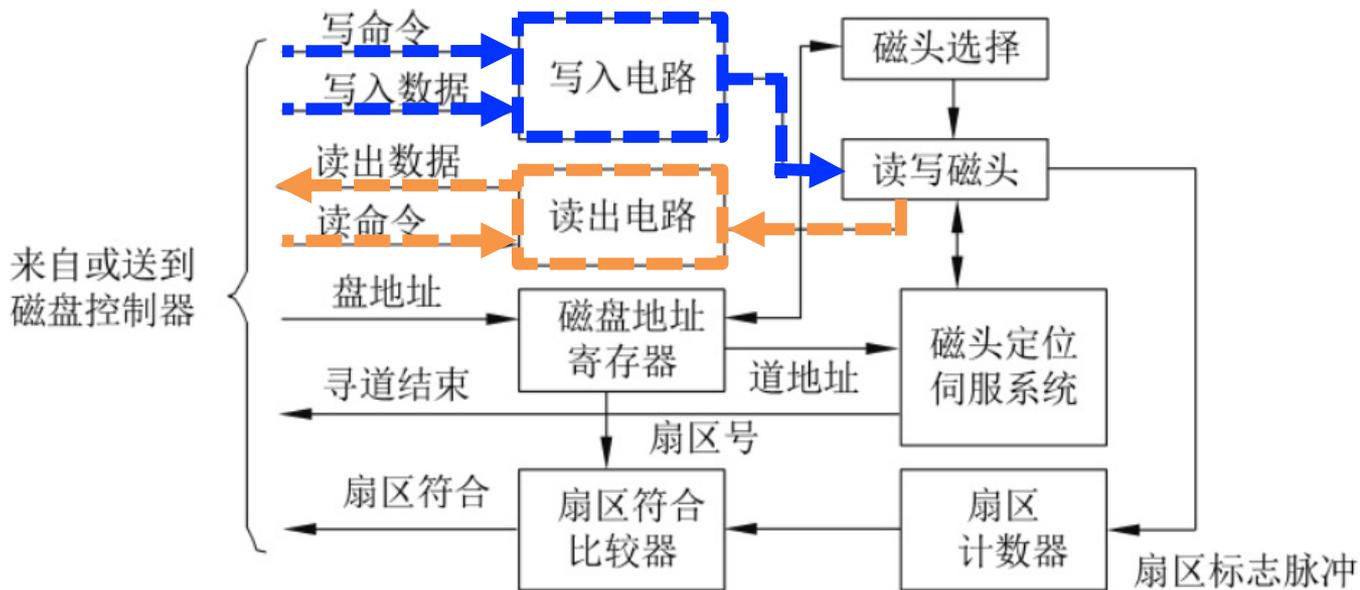


图 7.16 磁盘驱动器的内部逻辑结构



温切斯特磁盘的磁道记录格式

• 定长记录格式

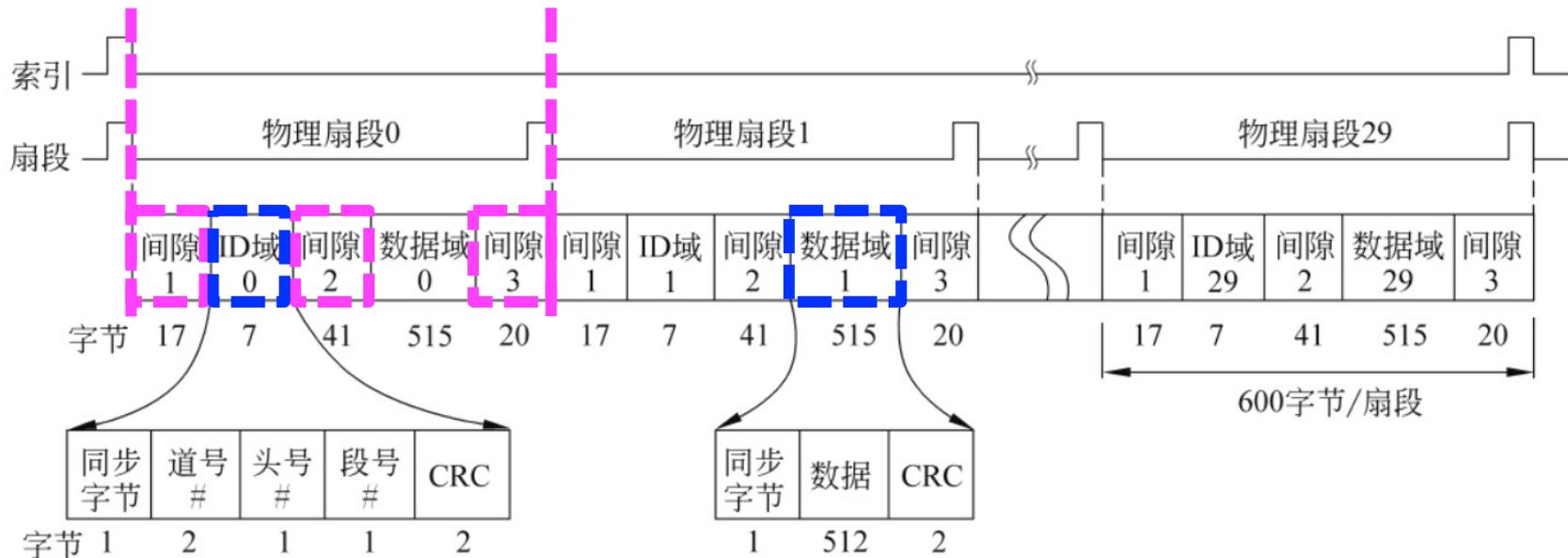


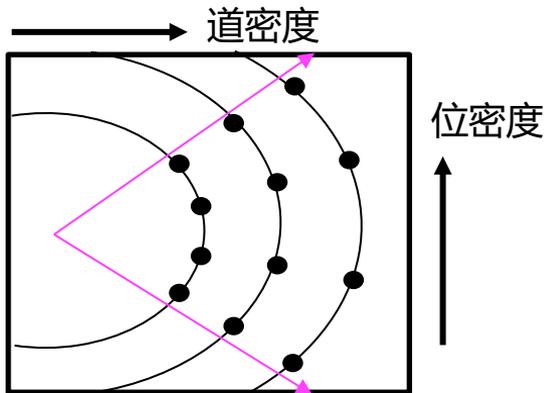
图 7.17 温切斯特磁盘的磁道记录格式



磁盘存储器的性能指标

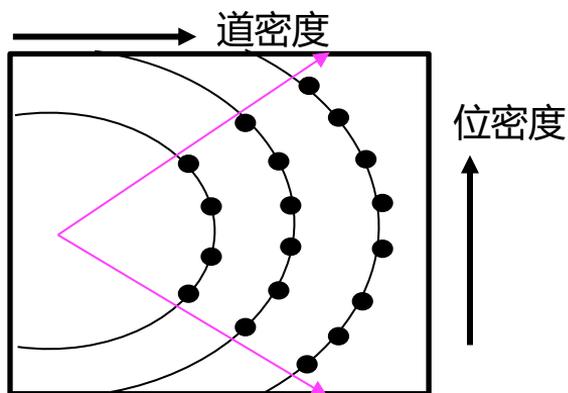
记录密度

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

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



磁盘存储器的性能指标

- **存储容量**

存储容量指整个存储器所能存放的**二进制信息量**，它与磁表面大小和记录密度密切相关。

低密度存储方式，**未格式化容量**的计算方法：

磁盘总容量 = 记录面数 × 理论柱面数 × 内圆周长 × 最内道位密度

低密度存储方式，**格式化容量**的计算方法：

磁盘实际数据容量 = 2 × 盘片数 × 磁道数/面 × 扇区数/磁道 × 512B/扇区

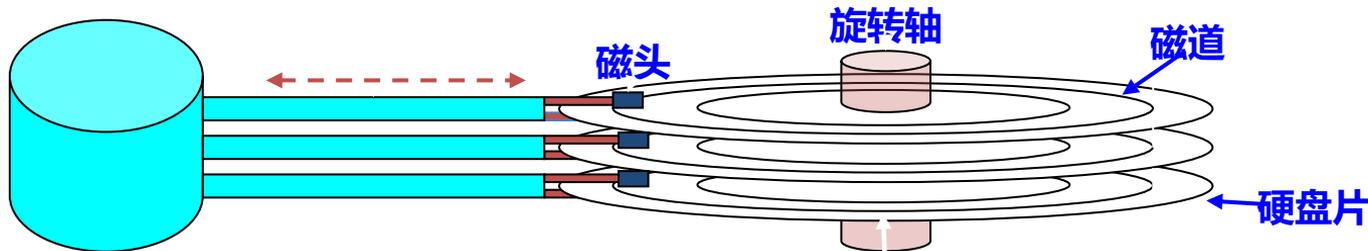
早起扇区大小一直是**512字节**，但现在已逐步更换到更大、更高效的**4096字节扇区**，通常称为4K扇区（这里1K=2¹⁰）。

- **数据传输速率**：单位时间内从存储介质上读出或写入的二进制信息量。



磁盘存储器的性能指标

- 平均存取时间



操作流程：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

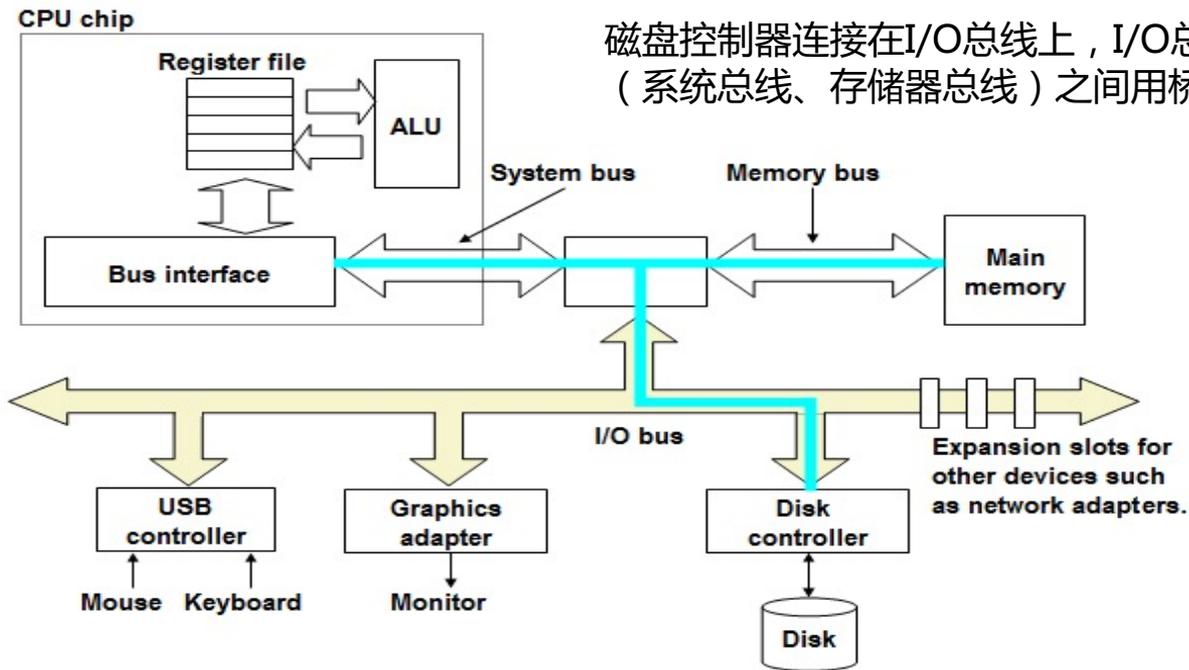
- 磁盘上的信息以扇区为单位进行读写，**平均存取时间为**：

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

- **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，取磁盘旋转一周所需时间的一半 (大约4~6ms) (转速：4200 / 5400 / 7200 / 10000rpm)
- **数据传输时间**——(大约0.01ms / 扇区)



磁盘存储器的连接



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

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。



冗余磁盘阵列

- 系统总体性能的提高不匹配
 - 处理器和主存性能改进**快**
 - 辅存性能改进**慢** *可靠性(Reliability)
- 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。



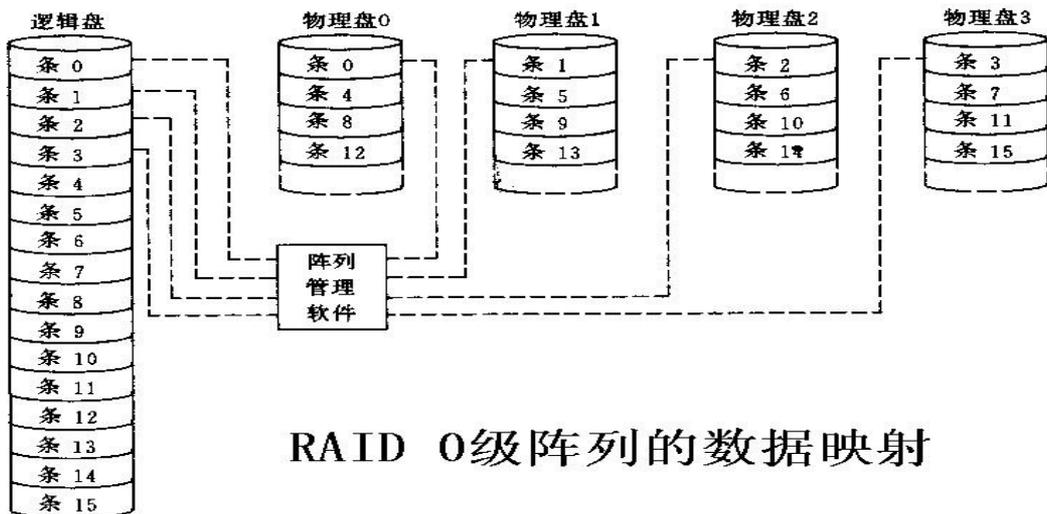
冗余磁盘阵列(RAID 0)

- 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中，以快速传输视频流。



RAID 0级阵列的数据映射



冗余磁盘阵列(RAID 1)

- 镜像盘实现1对1冗余(100% redundancy)

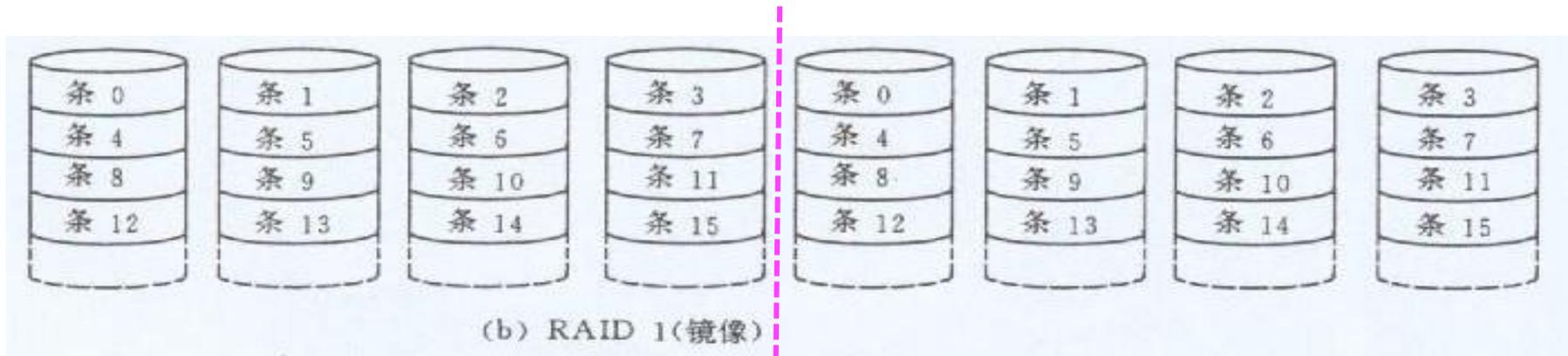
(1) **读**：一个读请求可由其中一个定位时间更少的磁盘提供数据。

(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。

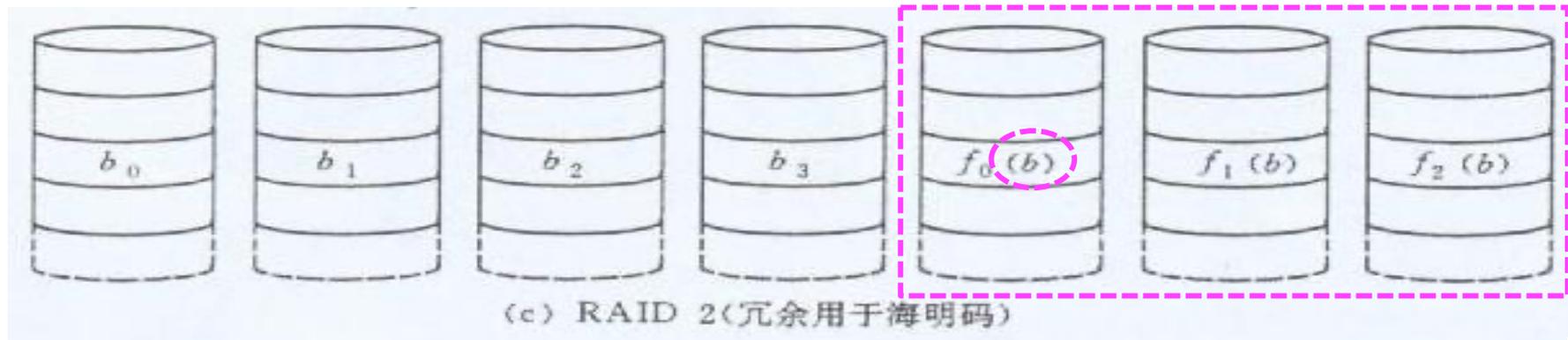




冗余磁盘阵列(RAID 2)

- 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。

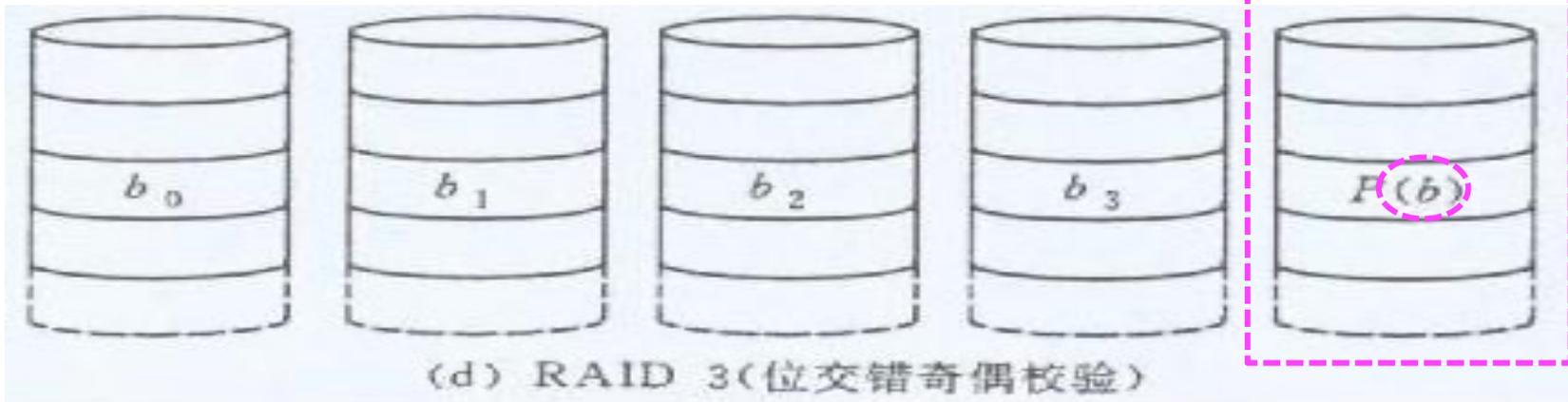
RAID2已不再使用！（冗余信息开销太大，价格较贵）





冗余磁盘阵列(RAID 3)

- 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。
- 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。





冗余磁盘阵列(RAID 4)

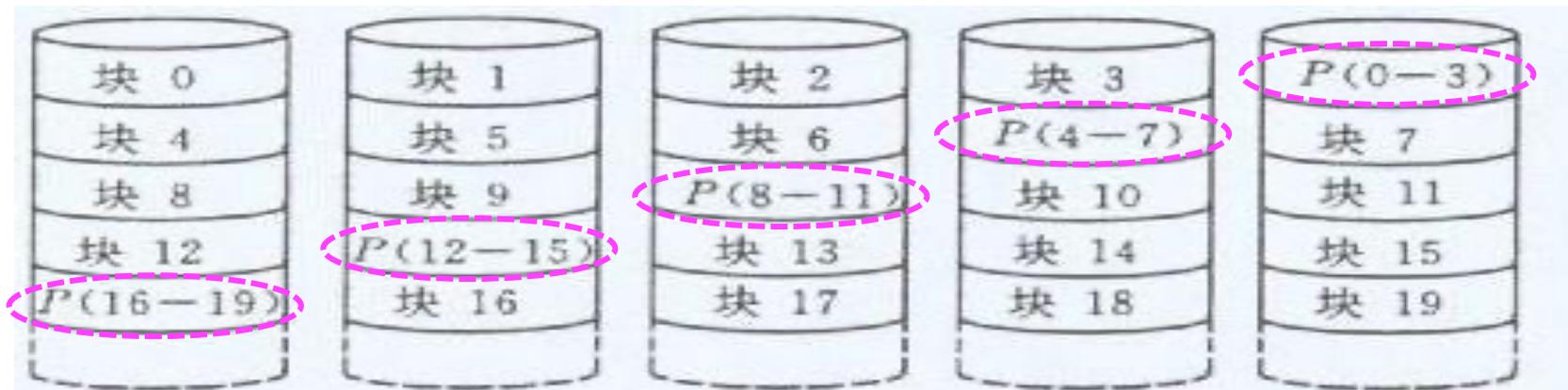
- 用一个冗余盘存放**相应块（块：较大的数据条区）的奇偶校验位。**
- 采用独立存取技术，每个磁盘的操作独立进行，所以，可同时响应多个I/O请求。因而它适合于要求I/O响应速度快的场合。
- 对于写操作，校验盘成为I/O瓶颈，因为每次写都要对校验盘进行。
 - **少量写**（只涉及个别磁盘）时，有“写损失”，因为一次写操作包含两次读和两次写
 - **大量写**（涉及所有磁盘的数据条区）时，则只需直接写入奇偶校验盘和数据盘。因为奇偶校验位可全部用新数据计算得到，而无须读原数据。





冗余磁盘阵列(RAID 5)

- 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。

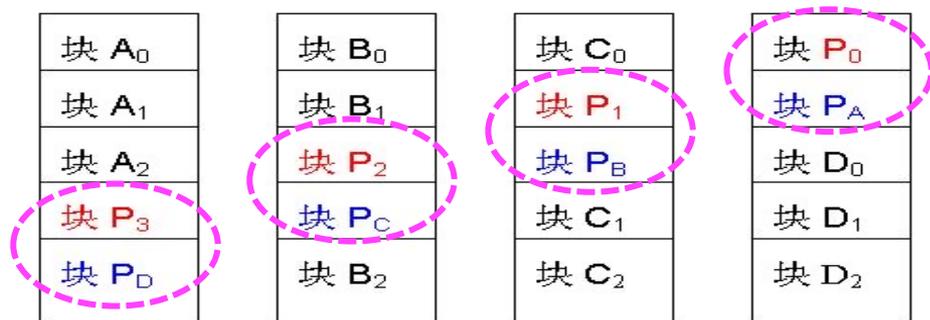


(f) RAID 5(块级分布式奇偶校验)



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。





Flash存储器和U盘

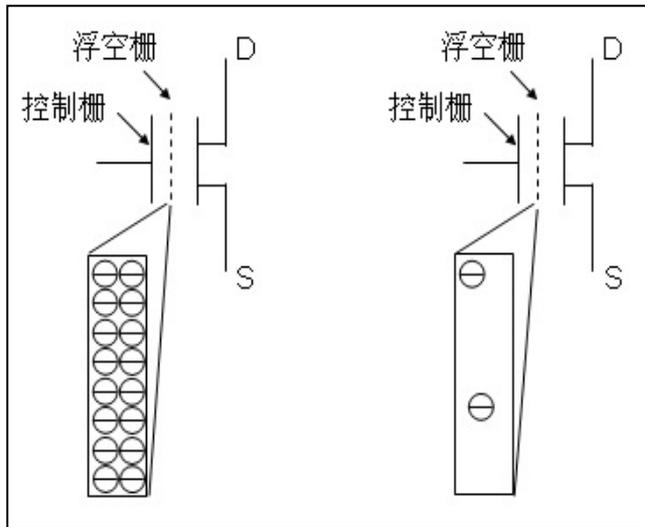
只读存储器ROM：

- **MROM (Mask ROM)**：掩膜只读存储器
- **PROM (Programmable ROM)**：可编程只读存储器
- **EPROM (Erasable PROM)**：可擦除可编程只读存储器
- **EEPROM (E²PROM , Electrically EPROM)**：电可擦除可编程只读存储器
- **flash memory**：闪存（快擦存储器）：快擦型电可擦除重编程ROM



Flash存储器和U盘

Flash 存储元：



(a) “0” 状态

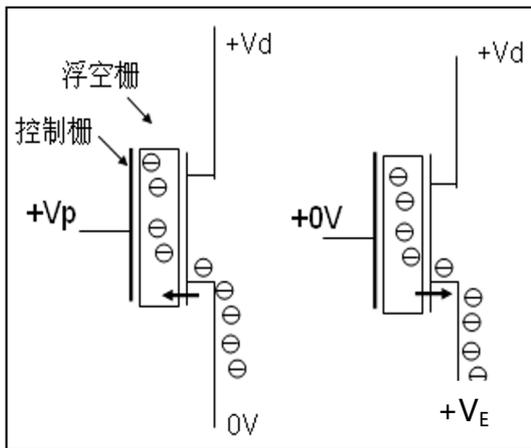
(b) “1” 状态

控制栅加足够正电压时，浮空栅储存大量负电荷，
为“0”态；

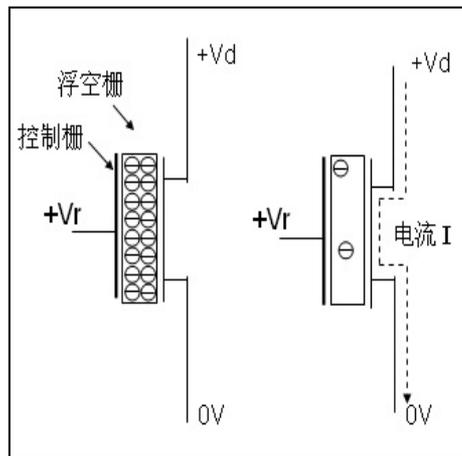
控制栅不加正电压时，浮空栅少带或不带负电荷，
为“1”态。



Flash存储器和U盘



(a) 编程:写 "0" (b) 擦除:写 "1"



(a) 读 "0" (b) 读 "1"

有三种操作：擦除、编程、读取

读快、写慢！

写入：快擦（所有单元为1）-- 编程（需要之处写0）

读出：控制栅加正电压，若状态为0，则读出电路检测不到电流；若状态为1，则能检测到电流。



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。



固态硬盘(SSD)

- SSD中一个闪存芯片由若干个**区块 (block)** 组成，每个区块由**若干页 (page)** 组成，通常，**页大小为512B~4KiB**，每个区块由32~128个页组成，因而区块大小为16KiB~512KiB，**数据按页为单位进行读写**。
- **SSD有三个限制：**
 - **写一个页的信息之前，必须先擦除该页所在的整个区块**
 - **擦除后区块内的页必须按顺序写入信息**
 - **只有有限的擦除/编程次数**
- 某一区块进行了**几千到几万次重复写**之后，就会被磨损而变成坏区块，不能再被使用。
- 闪存翻译层中有一个专门的**均化磨损 (wear leveling) 逻辑电路**，试图将擦除操作**平均分布**在所有区块上，以最大限度地延长SSD的使用寿命。



存储器层次结构

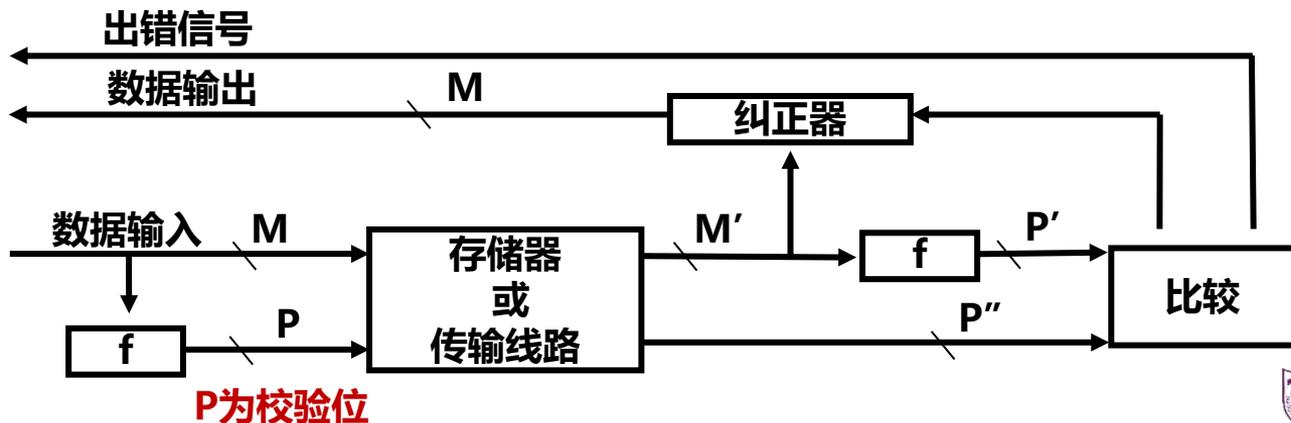
- 存储器概述
- 半导体随机存取存储器
- 外部辅助存储器
- **存储器的数据校验**
- 高速缓冲存储器
- 虚拟存储器





数据校验基本原理

- 为什么要进行数据的错误检测与校正？存取和传送时，由于元器件故障或噪音干扰等原因会出现差错。
- 措施：
 - (1) 从计算机**硬件本身的可靠性入手**，在电路、电源、布线等各方面采取必要的措施，提高计算机的抗干扰能力；
 - (2) 采取相应的**数据检错和校正措施**，自动地发现并纠正错误。
- 如何进行错误检测与校正？
 - 大多采用“**冗余校验**”思想，即除原数据信息外，还增加若干位编码，这些新增的代码被称为校验位。

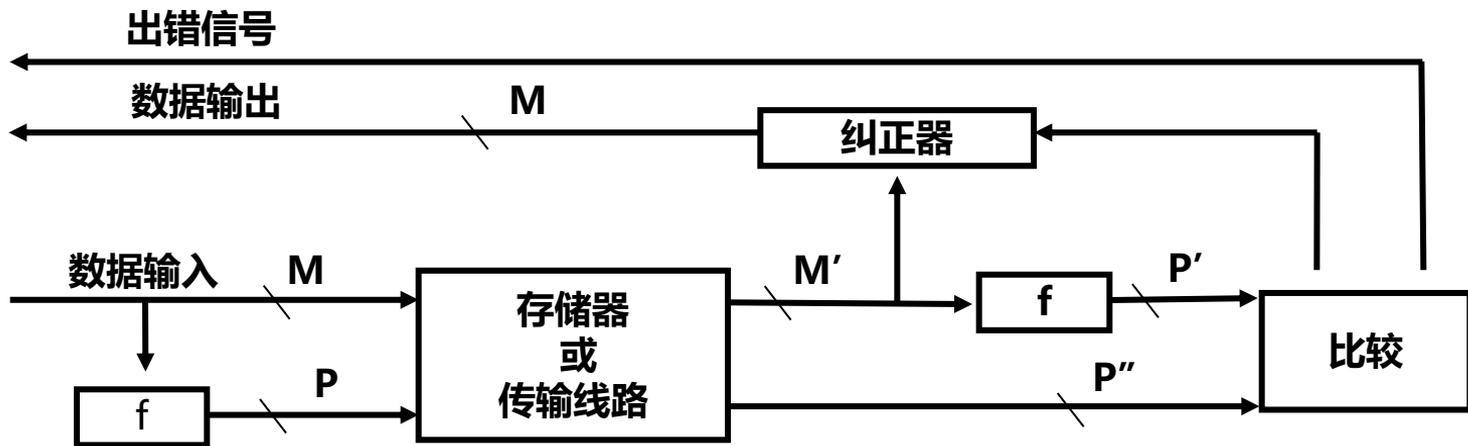




数据校验基本原理

比较的结果为以下三种情况之一：

- ① 没有检测到错误，得到的数据位直接传送出去。
- ② 检测到差错，并可以纠错。数据位和比较结果一起送入纠错器，将正确数据位传送出去。
- ③ 检测到错误，但无法确认哪位出错，因而不能进行纠错处理，此时，报告出错情况。





码字和码距

- 什么叫码距？

- 由若干位代码组成的一个字叫“码字”
- 两个码字中具有不同代码的位的个数叫这两个码字间的“距离”
- 码制中各码字间最小距离为“码距”，它就是这个码制的距离。

➤ 问题：“8421”码的码距是几？2 (0010) 和3 (0011) 间距离为1，“8421”码制的码距为1。

- 数据校验中“码字”指数据位和校验位按某种规律排列得到的代码

- 码距与检错、纠错能力的关系

- 若能检测 e 位错误，则码距 d 至少为 $e+1$ ；
- 若能纠正 t 位错误，则码距 d 至少为 $2t+1$ ；
- 若能同时检测 e 位错误，并纠正 t 位错误，则码距 d 至少为 $e+t+1$ 。

- 常用的数据校验码有：

奇偶校验码、海明校验码、循环冗余校验码。



奇偶校验码

- **基本思想**：增加一位奇（偶）校验位并一起存储或传送，根据终部件得到的相应数据和校验位，再求出新校验位，最后根据新校验位确定是否发生了错误。
- **实现原理**：假设数据 $B = b_{n-1}b_{n-2}\dots b_1b_0$ 从源部件传送至终部件。在终部件接收到的数据为 $B' = b_{n-1}' b_{n-2}' \dots b_1' b_0'$ 。

第一步：在源部件求出奇（偶）校验位 P 。

若采用奇校验，则 $P = b_{n-1} \oplus b_{n-2} \oplus \dots \oplus b_1 \oplus b_0 \oplus 1$ 。

若采用偶校验，则 $P = b_{n-1} \oplus b_{n-2} \oplus \dots \oplus b_1 \oplus b_0$ 。

第二步：在终部件求出奇（偶）校验位 P' 。

若采用奇校验，则 $P' = b_{n-1}' \oplus b_{n-2}' \oplus \dots \oplus b_1' \oplus b_0' \oplus 1$ 。

若采用偶校验，则 $P' = b_{n-1}' \oplus b_{n-2}' \oplus \dots \oplus b_1' \oplus b_0'$ 。

第三步：计算最终的校验位 P^* ，并根据其值判断有无奇偶错。

假定 P 在终部件接受到的值为 P'' ，则 $P^* = P' \oplus P''$

① 若 $P^* = 1$ ，则表示终部件接受的数据有奇数位错。

② 若 $P^* = 0$ ，则表示终部件接受的数据正确或有偶数个错。





奇偶校验码的特点

- **问题：奇偶校验码的码距是几？为什么？**

- 码距 $d=2$ 。

在奇偶校验码中，若两个数中有奇数位不同，则它们相应的校验位就不同；若有偶数位不同，则虽校验位相同，但至少有一位数据位不同。因而任意两个码字之间至少有一位不同。

- **特点**

- 根据码距和纠/检错能力的关系，**它只能发现奇数位出错**，不能发现偶数位出错，而且也不能确定发生错误的位置，不具有纠错能力。

- **开销小**，适用于校验一字节长的代码，故常被用于存储器读写检查或按字节传输过程中的数据校验。（因为一字节长的代码发生错误时，1位出错的概率较大，两位以上出错则很少，所以可用奇偶校验。）



海明校验码

- 由Richard Hamming于1950年提出，目前还被广泛使用。
- 主要用于存储器中数据存取校验。
- **基本思想：**
 - 奇偶校验码对整个数据编码生成一位校验位。因此这种校验码检错能力差，并且没有纠错能力。如果将整个数据按某种规律分成若干组，对每组进行相应的奇偶检测，就能提供多位检错信息，从而对错误位置进行定位，并将其纠正。
 - 海明校验码实质上就是一种多重奇偶校验码。
- **处理过程：**
 - 最终比较时按位进行异或，以确定是否有差错。
 - 这种异或操作所得到的结果称为故障字（ syndrome word ）。显然，校验码和故障字的位数是相同。

每一组一个校验位，校验码位数等于组数！

每一组内采用一位奇偶校验！





校验位的位数的确定

- 假定数据位数为 n ，校验码为 k 位，则故障字位数也为 k 位。 k 位故障字所能表示的状态最多是 2^k ，每种状态可用来说明一种出错情况。
- 若只有一位错，则结果可能是：
 - 数据中某一位错 (n 种可能)
 - 校验码中有一位错 (k 种可能)
 - 无错 (1种可能)

1+n+k种情况

假定最多有一位错，则 n 和 k 必须满足下列关系：

$$2^k \geq 1+n+k, \quad \text{即: } 2^k - 1 \geq n+k$$

- 有效数据位数和校验码位数间的关系（见下页）
- 当数据有8位时，校验码和故障字都应有4位。

说明：4位故障字最多可表示16种状态，而单个位出错情况最多只有12种可能（8个数据位和4个校验位），再加上无错的情况，一共有13种。所以，用16种状态表示13种情况应是足够了。



有效数据位数和校验码位数间的关系

n和k的关系： $2^k - 1 \geq n + k$ （n和k分别为数据位数和校验位数）

数据位数	单纠错		单纠错/双检错	
	校验位数	增加率	校验位数	增加率
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91





海明码的分组

- **基本思想**： n 位数据位和 k 位校验位按某种方式排列为一个 $(n+k)$ 位的码字，将该码字中每个出错位的位置与故障字的数值建立关系，通过故障字的值确定该码字中哪一位发生了错误，并将其取反来纠正。
- 根据上述基本思想，**按以下规则来解释各故障字的值。**
 - **规则1**：若故障字每位全部是0，则表示没有发生错误。
 - **规则2**：若故障字中有且仅有一位为1，则表示校验位中有一位出错，因而不需纠正。
 - **规则3**：若故障字中多位为1，则表示有一个数据位出错，其在码字中的出错位置由故障字的数值来确定。纠正时只要将出错位取反即可。





海明码的分组

- 以8位数据进行单个位检错和纠错为例说明。假定8位数据 $M = M_8M_7M_6M_5M_4M_3M_2M_1$ ，4位校验位 $P = P_4P_3P_2P_1$ 。根据规则将M和P按一定的规律排到一个12位码字中。
 - **据规则1**，故障字为0000时，表示无错。
 - **据规则2**，故障字中有且仅有一位为1时，表示校验位中有一位出错。此时，故障字只可能是0001、0010、0100、1000，将这四种状态分别代表校验位中 P_1 、 P_2 、 P_3 、 P_4 位发生错误，因此，它们分别位于码字的第1、2、4、8位。
 - **据规则3**，将其他多位为1的故障字依次表示数据位 $M_1 \sim M_8$ 发生错误的情况。因此，数据位 $M_1 \sim M_8$ 分别位于码字的第0011(3)、0101(5)、0110(6)、0111(7)、1001(9)、1010(10)、1011(11)、1100(12)位。即码字的排列为： $M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$
 - 这样，得到故障字 $S = S_4S_3S_2S_1$ 的各个状态和出错情况的对应关系表，可根据这种对应关系对整个数据进行分组。

是逻辑顺序，物理上M和P是分开的！



海明码的分组

码字： $M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$

故障字 $S_4S_3S_2S_1$ 每一位的值反映所在组的奇偶性

序号 含义 分组	1	2	3	4	5	6	7	8	9	10	11	12	故障字	正确	出错位												
	P_1P_2	M_1P_3	M_2M_3	M_4P_4	M_5M_6	M_7M_8												1	2	3	4	5	6	7	8	9	10
第4组								√	√	√	√	√	S_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1
第3组				√	√	√	√					√	S_3	0	0	0	0	1	1	1	1	0	0	0	0	1	
第2组			√	√			√	√			√	√	S_2	0	0	1	1	0	0	1	1	0	0	1	1	0	
第1组	√		√		√		√		√		√		S_1	0	1	0	1	0	1	0	1	0	1	0	1	0	

数据位或校验位出错一定会影响所在组的奇偶性。

例：若 M_2 出错，则故障字为0101，因而会改变 S_3 和 S_1 所在分组的奇偶性。故 M_2 同时被分到第3(S_3)组和第1(S_1)组。

问题：若 P_1 出错，则如何？若 M_8 出错，则如何？ $P_1 \sim 0001$ ，分在第1组； $M_8 \sim 1100$ ，分在第4组和第3组



校验位的生成和检错、纠错

- 分组完成后，就可对每组采用相应的奇（偶）校验，以得到相应的一个校验位。
- 假定采用偶校验（取校验位 P_i ，使对应组中有偶数个1），则得到校验位与数据位之间存在如下关系：

$$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8$$

$$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8$$

$$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$$

$$P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$$

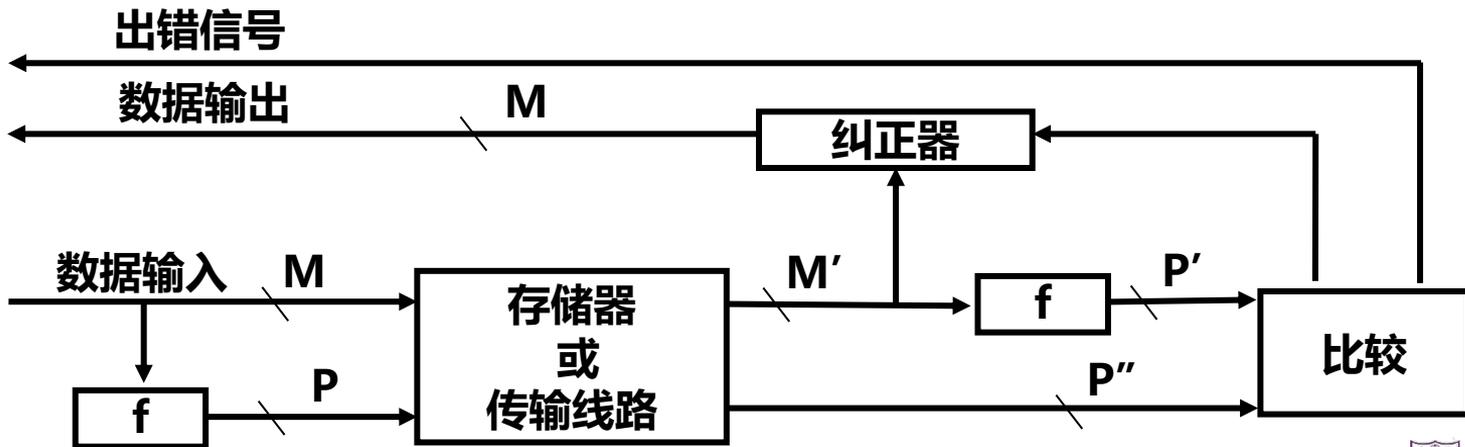
码字： $M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$





海明校验过程

- 根据公式求出每一组对应的校验位 P_i ($i=1,2,3,4$)
- 数据 M 和校验位 P 一起被存储，根据读出数据 M' 得新校验位 P'
- 读出校验位 P'' 与新校验位 P' 按位进行异或操作，得故障字 $S = S_4S_3S_2S_1$
- 根据 S 的值确定：无错、仅校验位错、某个数据位错





海明码举例

- 假定一个8位数据M为： $M_8M_7M_6M_5M_4M_3M_2M_1 = 01101010$ ，根据上述公式求出相应的**校验位为**：

$$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$$

- 假定**12位码字** ($M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$) 读出后为：

(1) 数据位 $M' = M = 01101010$ ，校验位 $P' = P = 0011$

(2) 数据位 $M' = 011\mathbf{1}010$ ，校验位 $P' = P = 0011$

(3) 数据位 $M' = M = 01101010$ ，校验位 $P' = \mathbf{1}011$

- 要求分别考察每种情况的**故障字**。

(1) 数据位 $M' = M = 01101010$ ，校验位 $P' = P = 0011$ ，即无错。

因为 $M' = M$ ，所以 $P' = P$ ，因此 $S = P' \oplus P = P \oplus P = 0000$ 。





海明码举例

(2) 数据位 $M' = 01111010$, 校验位 $P'' = P = 0011$, 即 M_5 错。

对 M' 生成新的校验位 P' 为 :

$$P_4' = M_5' \oplus M_6' \oplus M_7' \oplus M_8' = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$P_3' = M_2' \oplus M_3' \oplus M_4' \oplus M_8' = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_2' = M_1' \oplus M_3' \oplus M_4' \oplus M_6' \oplus M_7' = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$P_1' = M_1' \oplus M_2' \oplus M_4' \oplus M_5' \oplus M_7' = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

故障字 S 为 :

$$S_4 = P_4' \oplus P_4'' = 1 \oplus 0 = 1$$

$$S_3 = P_3' \oplus P_3'' = 0 \oplus 0 = 0$$

$$S_2 = P_2' \oplus P_2'' = 1 \oplus 1 = 0$$

$$S_1 = P_1' \oplus P_1'' = 0 \oplus 1 = 1$$

因此, 错误位是第9位, 排列的是数据位 M_5 , 所以检错正确, 纠错时, 只要将码字的第9位 (M_5) 取反即可。



海明码举例

(3) 数据位 $M' = M = 01101010$, 校验位 $P'' = 1011$,

即：校验码第4位(P_4)错。

因为 $M' = M$, 所以 $P' = P$, 因此故障位 S 为：

$$S_4 = P_4' \oplus P_4'' = 0 \oplus 1 = 1$$

$$S_3 = P_3' \oplus P_3'' = 0 \oplus 0 = 0$$

$$S_2 = P_2' \oplus P_2'' = 1 \oplus 1 = 0$$

$$S_1 = P_1' \oplus P_1'' = 1 \oplus 1 = 0$$

错误位是第1000位(即第8位) , 这位上排列的是校验位 P_4 , 所以检错时发现数据正确 , 不需纠错。



单纠错和双检错码

• 单纠错码 (SEC)

- 问题：上述($n=8/k=4$)汉明码的码距是几？
- 码距 $d=3$ 。因为，若有一位出错，则因该位至少要参与两组校验位的生成，因而至少引起两个校验位的不同。两个校验位加一个数据位等于3。例如，若 M_1 出错，则故障字为0011，即 P_2 和 P_1 两个校验位发生改变，12位码字中有三位 (M_1 、 P_2 和 P_1) 不同。
- 根据码距与检错、纠错能力的关系，知：这种码制能发现两位错，或对单个位出错进行定位和纠错。这种码称为单纠错码 (SEC)。

• 单纠错和双检错码 (SEC-DED)

- 具有发现两位错和纠正一位错的能力，称为单纠错和双检错码 (SEC-DED)。
- 若要成为SEC-DED，则码距需扩大到 $d=4$ 。为此，还需增加一位校验位 P_5 ，将 P_5 排列在码字的最前面，即： $P_5M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$ ，并使得数据中的每一位都参与三个校验位的生成。从表中可看出除了 M_4 和 M_7 外，其余位都只参与了两个校验位的生成。因此 P_5 按下式求值： $P_5 = M_1 \oplus M_2 \oplus M_3 \oplus M_5 \oplus M_6 \oplus M_8$

当任意一个数据位发生错误时，必将引起三个校验位发生变化，所以码距为4。





固态硬盘的数据校验

- 早期采用简单的海明码进行检错和纠错。
- 每页数据分成两组，各自进行检错和纠错。例如，对于512B大小的页，每组数据都是256行x8列的矩阵，分别对矩阵中的行 $R_0 \sim R_{255}$ 和列 $C_0 \sim C_7$ 计算出16位行校验位 $RP_0 \sim RP_{15}$ 和6位列校验位 $CP_0 \sim CP_5$ 。
- 16个行校验位占两个字节，6个列校验位再加两位1构成一个字节，三个字节组成ECC（错误检测和纠正）信息。
- 每页除数据区外，还有OOB（out of-band）区，ECC记录其中。
- 写某页时，会将该页数据对应ECC信息计算出来，并写入OOB区。在读该页时，重新计算数据对应的ECC信息，并和记录在OOB区中的原ECC信息进行比较运算，根据运算结果进行数据检/纠错。
- 海明码只能纠正1位错或检测2位错，已不能满足SDD要求，需要采用纠错能力更高的BCH码，甚至开始由BCH码向低密度奇偶校验码LDPC过渡来增加使用寿命。



循环冗余校验码

- 循环冗余校验码 (Cyclic Redundancy Check) , 简称CRC码
 - 具很强的检错、纠错能力。
 - 用于大批量数据存储和传送(如：外存和通信)中的数据校验。
 - 为什么大批量数据不用奇偶校验？
 - 在每个字符后增加一位校验位会增加大量的额外开销；尤其在网络通信中，对传输的二进制比特流没有必要再分解成一个个字符，因而无法采用奇偶校验码。
 - 通过某种数学运算来建立数据和校验位之间的约定关系。

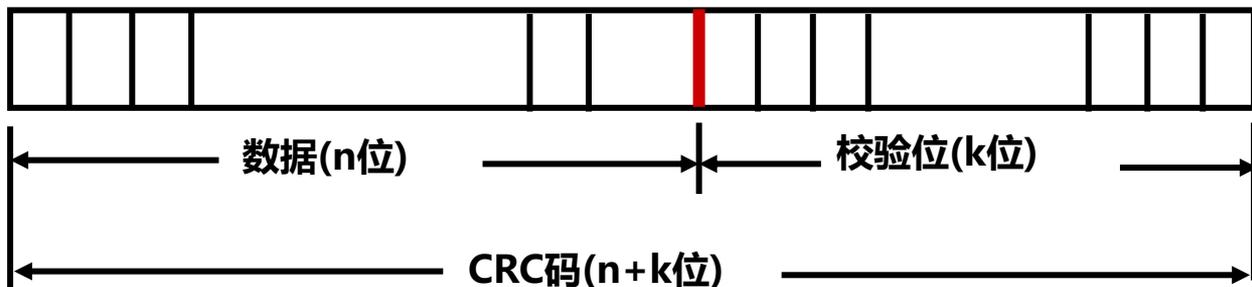
奇偶校验码和海明校验码都是以奇偶检测为手段的。



CRC码的检错方式

基本思想：

- 数据信息 $M(x)$ 为一个 n 位的二进制数据，将 $M(x)$ 左移 k 位后，用一个约定的“生成多项式” $G(x)$ 相除， $G(x)$ 是一个 $k+1$ 位的二进制数，相除后得到的 k 位余数就是校验位。校验位拼接到 $M(x)$ 后，形成一个 $n+k$ 位的代码，称该代码为循环冗余校验（CRC）码，也称 $(n+k, n)$ 码。
- 一个CRC码一定能被生成多项式整除，当数据和校验位一起送到接受端后，只要将接受到的数据和校验位用同样的生成多项式相除，如果正好除尽，表明没有发生错误；若除不尽，则表明某些数据位发生了错误。通常要求重传一次。





循环冗余校验码举例

- **校验位的生成**：用一个例子来说明校验位的生成过程。
 - 假设要传送的数据信息为：100011，即报文多项式为：
$$M(x) = x^5 + x + 1$$
。数据信息位数 $n=6$ 。
 - 若约定的生成多项式为： $G(x) = x^3 + 1$ ，则生成多项式位数为4位，所以校验位位数 $k=3$ ，除数为1001。
 - 生成校验位时，用 $x^3 \cdot M(x)$ 去除以 $G(x)$ ，即：100011000 \div 1001。
 - 相除时采用“模2运算”的多项式除法。



存储器层次结构

- 存储器概述
- 半导体随机存取存储器
- 外部辅助存储器
- 存储器的数据校验
- **高速缓冲存储器**
- 虚拟存储器





高速缓冲存储器

- **大量典型程序的运行情况分析结果表明**

- 在较短时间间隔内，程序产生的地址往往集中在一个很小范围内
这种现象称为程序访问的局部性：**空间局部性**、**时间局部性**

- **程序具有访问局部性特征的原因**

- 指令：指令按序存放，地址连续，循环程序段或子程序段重复执行
- 数据：连续存放，数组元素重复、按序访问

- **为什么引入Cache会加快访存速度？**

- 在CPU和主存之间设置一个快速小容量的存储器，其中总是存放最活跃（被频繁访问）的程序和数据，由于程序访问的局部性特征，大多数情况下，CPU能直接从这个高速缓存中取得指令和数据，而不必访问主存。

这个高速缓存就是位于主存和CPU之间的Cache！



程序访问的局部性原理举例1

高级语言源程序

对应的汇编语言程序

```

sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
*v = sum;

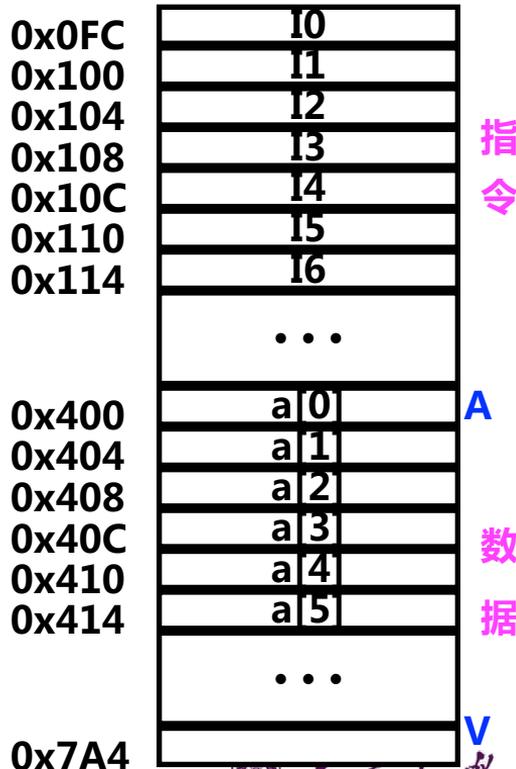
```

```

I0:          sum <-- 0
I1:          ap <-- A  A是数组a的起始地址
I2:          i <-- 0
I3:          if (i >= n) goto done
I4:  loop:   t <-- (ap) 数组元素a[i]的值
I5:          sum <-- sum + t  累计在sum中
I6:          ap <-- ap + 4  计算下个数组元素地址
I7:          i <-- i + 1
I8:          if (i < n) goto loop
I9:  done:   V <-- sum  累计结果保存至地址v

```

主存的布局:



每条指令4个字节；每个数组元素4字节；

指令和数组元素在内存中均连续存放

sum, ap, i, t 均为通用寄存器；A, V为内存地址



程序访问的局部性原理举例1

问题：指令和数据的时间局部性和空间局部性各自体现在哪里？

指令： 0x0FC (I0)

...

→0x108 (I3)

→0x10C (I4)

...

→0x11C (I8)

→0x120 (I9)



循环
n次

数据：只有数组在主存中：

0x400→0x404→0x408

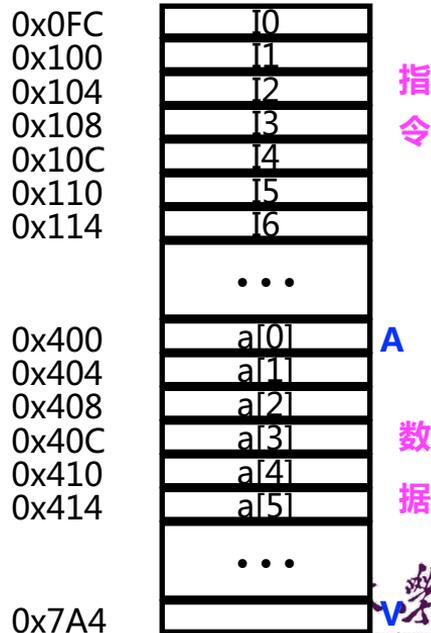
→0x40C→.....→0x7A4

若n足够大，则在一段
时间内一直在局部区域
内执行指令，故循环内
指令的时间局部性好；

按顺序执行，故程序空
间局部性好！

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
*v = sum;
```

主存的布局:



**数组元素按顺序存放，按顺序访问，故空间局部性好；
每个数组元素都只被访问1次，故没有时间局部性。**





程序访问的局部性原理举例2

- 以下哪个对数组a引用的空间局部性更好？时间局部性呢？变量sum的空间局部性和时间局部性如何？对于指令来说，for循环体的空间局部性和时间局部性如何？

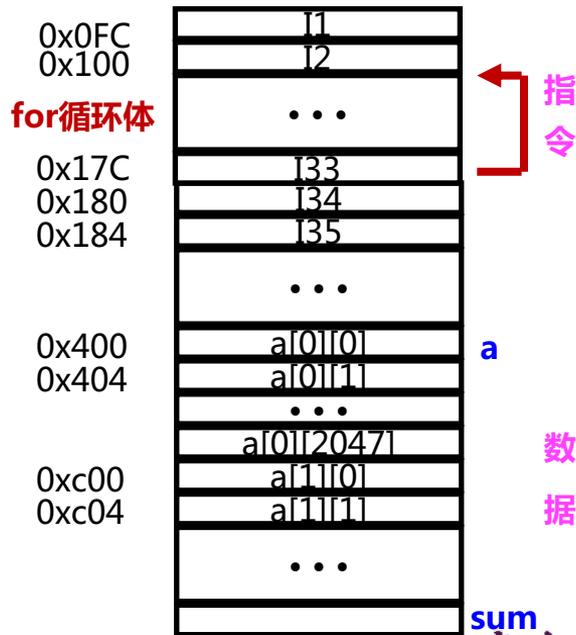
程序段A:

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum=0;
    for (i=0; i<M, i++)
        for (j=0; j<N, j++) sum+=a[i][j];
    return sum;
}
```

程序段B:

```
int sumarraycols(int a[M][N])
{
    int i, j, sum=0;
    for (j=0; j<N, j++)
        for (i=0; i<M, i++) sum+=a[i][j];
    return sum;
}
```

M=N=2048时主存的布局:



数组在存储器中按行优先顺序存放



程序访问的局部性原理举例2

程序段A的时间局部性和空间局部性分析：

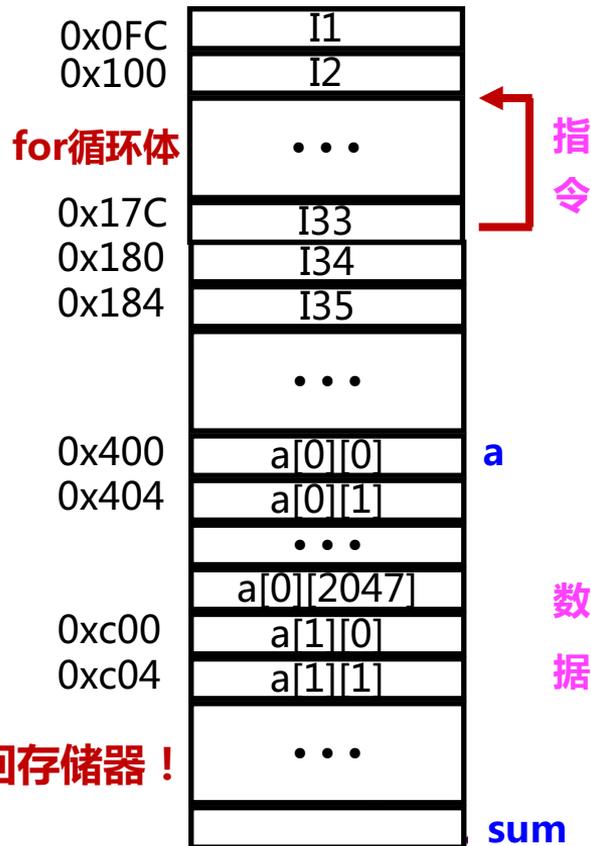
(1) 数组a：访问顺序为a[0][0], a[0][1],, a[0][2047]; a[1][0], a[1][1],, a[1][2047];

与存放顺序一致，故空间局部性好！

因为每个a[i][j]只被访问一次，故时间局部性差！

(2) 变量sum：单个变量不考虑空间局部性；每次循环都要访问sum，所以其时间局部性较好！

(3) for循环体：循环体内指令按序连续存放，所以空间局部性好！
循环体被连续重复执行2048x2048次，所以时间局部性好！



实际上，优化的编译器使循环中的sum分配在寄存器中，最后才写回存储器！



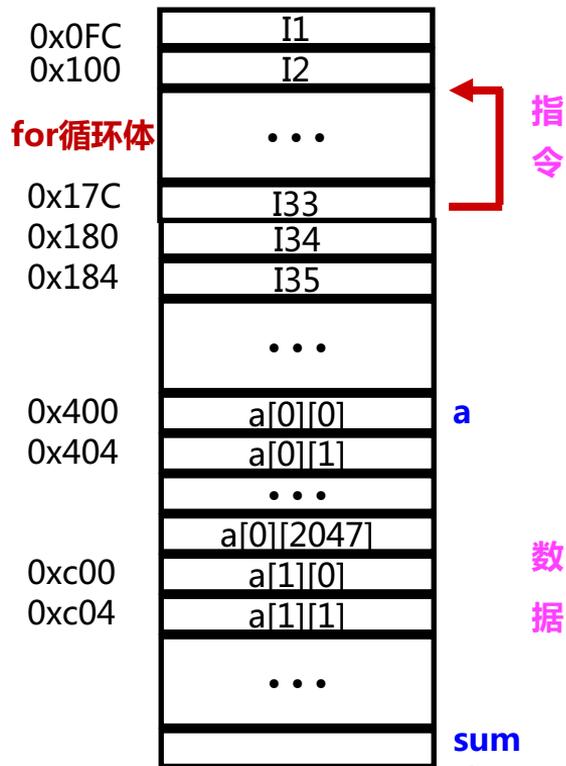
程序访问的局部性原理举例2

- 程序段B的时间局部性和空间局部性分析：
 - 数组a**：访问顺序为 $a[0][0]$, $a[1][0]$,, $a[2047][0]$;
 $a[0][1]$, $a[1][1]$,, $a[2047][1]$;,
 与存放顺序不一致，每次跳过2048个单元，若交换单位小于2KB，则没有空间局部性！
 (时间局部性差，同程序A)
 - 变量sum**：(同程序A)
 - for循环体**：(同程序A)

实际运行结果(2GHz Intel Pentium 4):

程序A：59,393,288 时钟周期
 程序B：1,277,877,876 时钟周期

程序A比程序B快20.5倍 !!

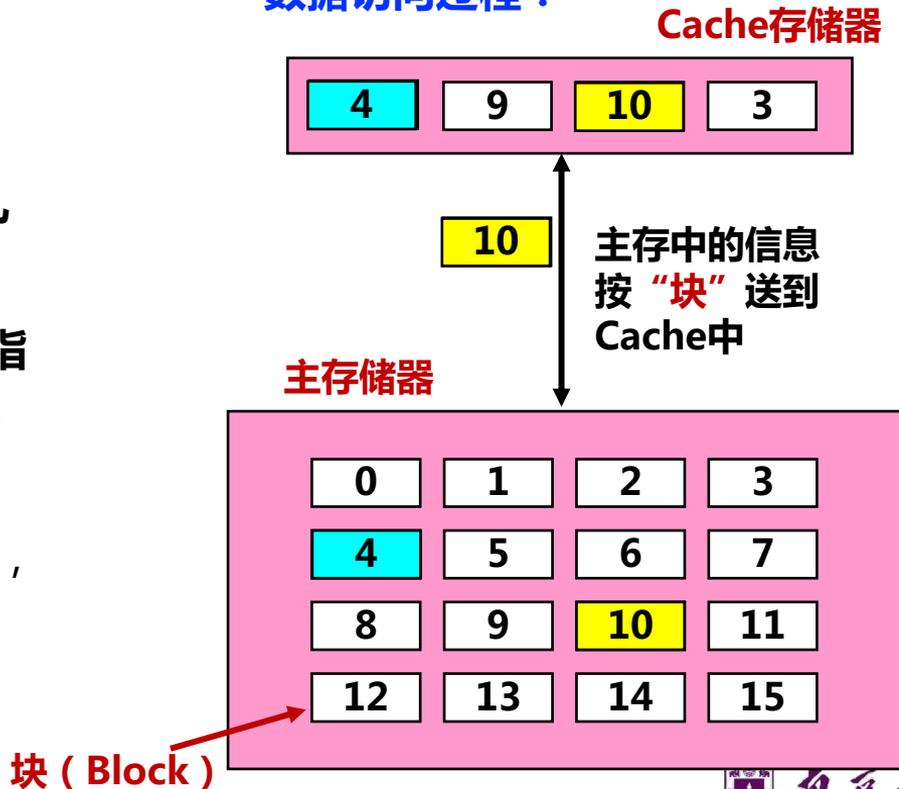




高速缓存(Cache)简介

- Cache是一种小容量高速缓冲存储器，它由SRAM组成。
- Cache直接制作在CPU芯片内，速度几乎与CPU一样快。
- 程序运行时，CPU使用的一部分数据/指令会预先成批拷贝在Cache中，Cache的内容是主存储器中部分内容的映象。
- 当CPU需要从内存读(写)数据或指令时，先检查Cache，若有，就直接从Cache中读取，而不用访问主存储器。

• 数据访问过程：





高速缓存(Cache)的操作过程

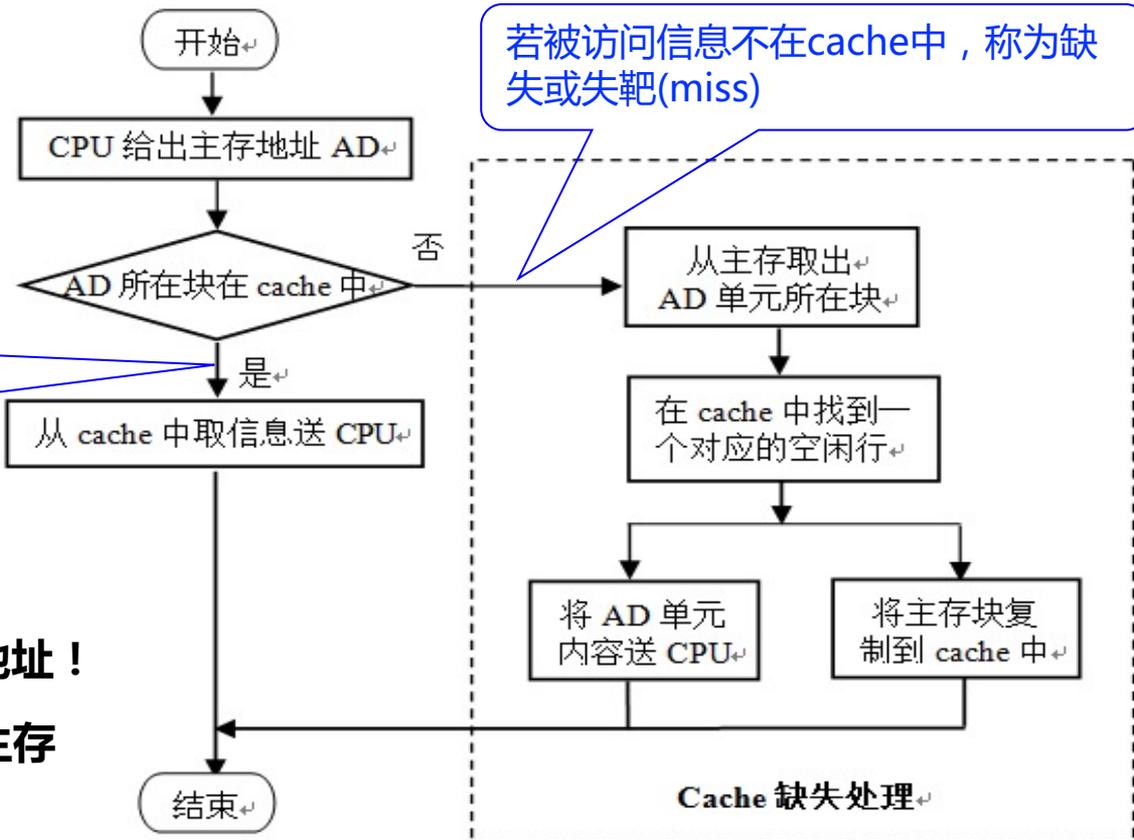
- **问题：什么情况下，CPU产生访存要求？**

➤ **执行指令时！**

若被访问信息在 cache 中，称为命中(hit)

指令最初给出的是虚拟地址！

如何将虚拟地址转换为主存地址，后面介绍。





高速缓存(Cache)的实现

- **问题：要实现Cache机制需要解决哪些问题？**

- 如何分块？
- 主存块和Cache之间如何映射？
- Cache已满时，怎么办？
- 写数据时怎样保证Cache和MM的一致性？
- 如何根据主存地址访问到cache中的数据？.....

主存被分成若干大小相同的块，称为主存块(Block)，Cache也被分成相同大小的块，称为Cache行(line)或槽(Slot)。

- **问题：Cache对程序员(编译器)是否透明？为什么？**

- 是透明的，程序员(编译器)在编写/生成高级或低级语言程序时无需了解Cache是否存在或如何设置，感觉不到cache的存在。
- 但是，对Cache深入了解有助于编写出高效的程序！



Cache行和主存块之间的映射方式

- **什么是Cache的映射功能？**
 - 把访问的局部主存区域取到Cache中时，该放到Cache的何处？
 - Cache行比主存块少，多个主存块映射到一个Cache行中
- **如何进行映射？**
 - 把主存空间划分成大小相等的主存块 (Block)
 - Cache中存放一个主存块的对应单位称为槽 (Slot) 或行 (line)
有书中也称之为块 (Block)，有书称之为页 (page) (不妥！)
 - 将主存块和Cache行按照以下三种方式进行映射
 - **直接(Direct)**：每个主存块映射到Cache的固定行
 - **全相联(Full Associate)**：每个主存块映射到Cache的任一行
 - **组相联(Set Associate)**：每个主存块映射到Cache固定组中任一行





直接映射

- **直接映射把主存的每一块映射到一个固定的Cache行（槽）**

- 也称模映射(Module Mapping)

- **映射关系为：Cache行号=主存块号 mod Cache行数**

举例： $4=100 \bmod 16$ （假定Cache共有16行）

（说明：主存第100块应映射到Cache的第4行中。）

块（行）都从0开始编号

- **特点：**

- **容易实现，命中时间短**

- 无需考虑淘汰（替换）问题

- **但不够灵活**，Cache存储空间得不到充分利用，命中率低

例如，需将主存第0块与第16块同时复制到Cache中时，由于它们都只能复制到Cache第0行，即使Cache其它行空闲，也有一个主存块不能写入Cache。

这样就会产生频繁的Cache装入。



直接映射的组织示意图

假定数据在主存和Cache间的传送单位为512B。

Cache大小：

$$2^{13}B = 8KB = 16行 \times 512B / 行$$

主存大小：

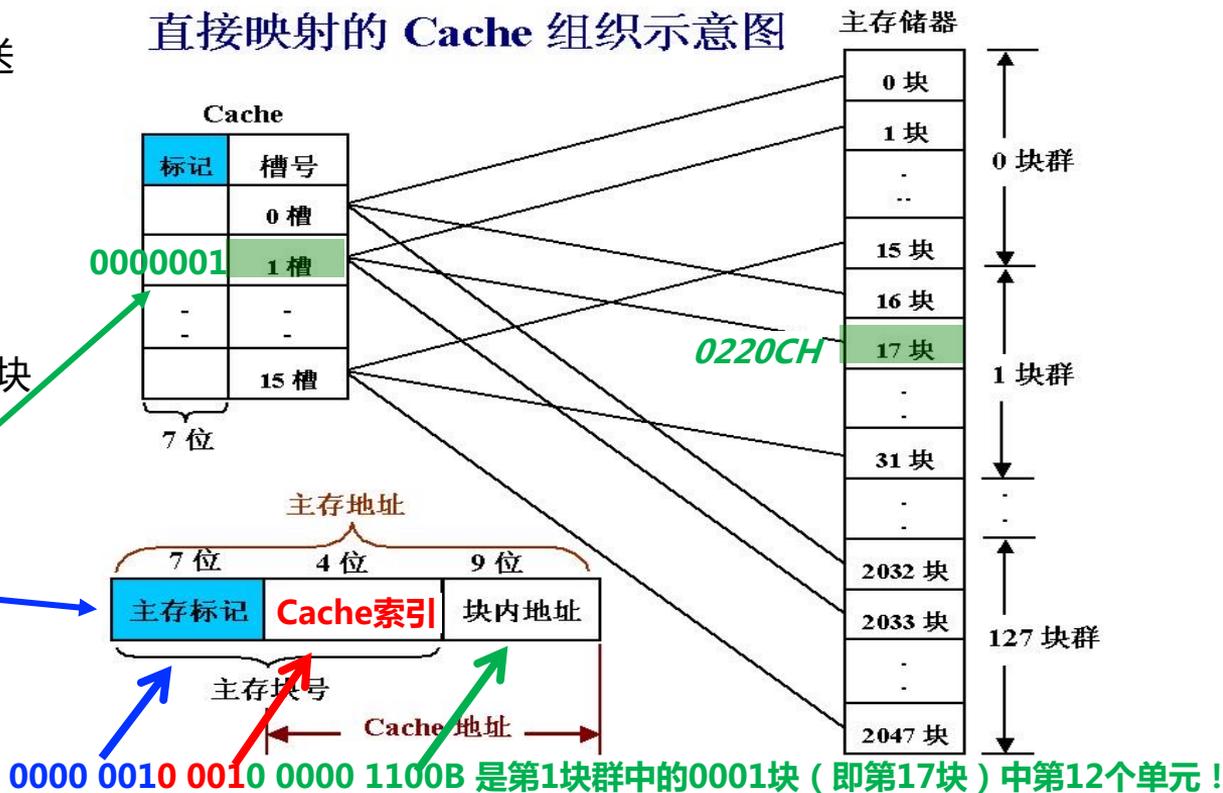
$$2^{20}B = 1024KB = 2048块 \times 512B / 块$$

Cache标记(tag)指出对应行取自哪个主存块群

指出对应地址位于哪个块群

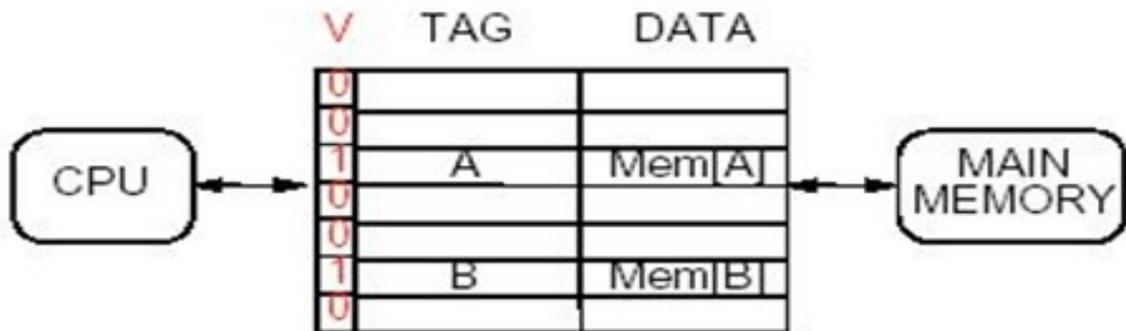
例：如何对0220CH单元进行访问？

直接映射的 Cache 组织示意图





有效位



为何要用有效位来区分是否有效？

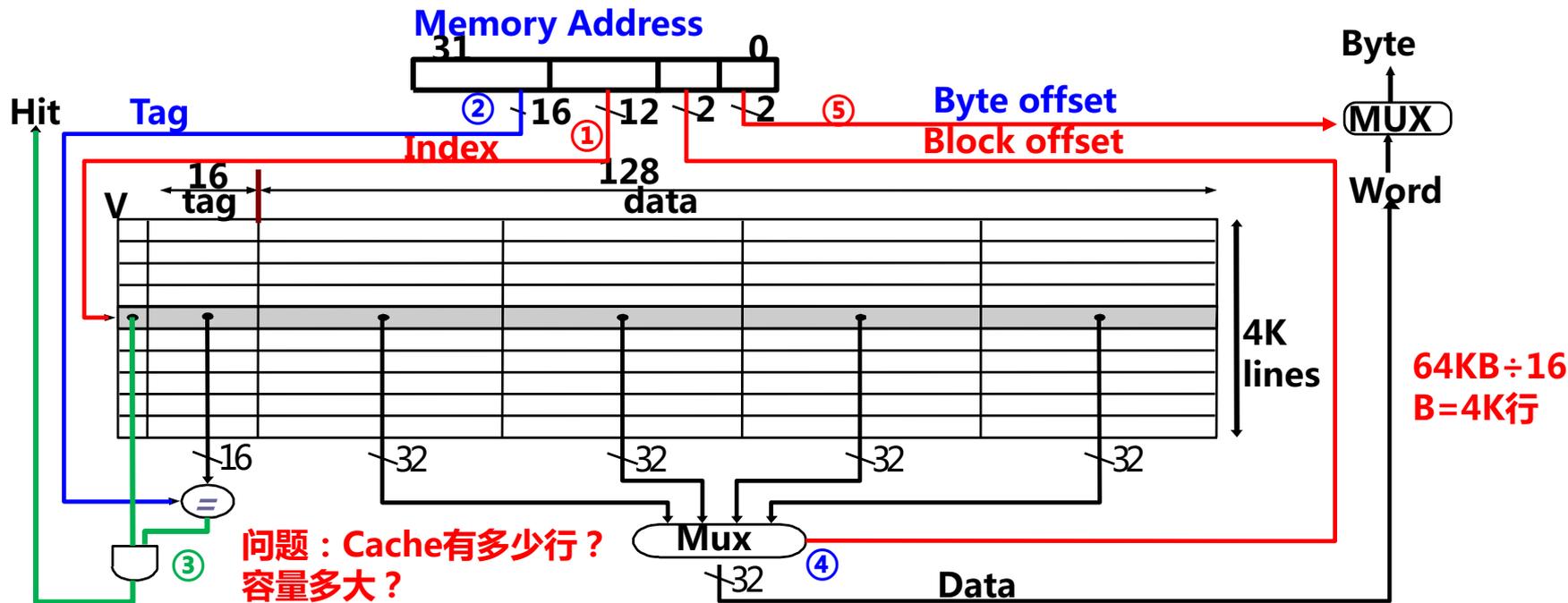
- V为有效位，为1表示信息有效，为0表示信息无效
- 开机或复位时，使所有行的有效位V=0
- 某行被替换后使其V=1
- 某行装入新块时 使其V=1
- 通过使V=0来冲刷Cache（例如：进程切换时，DMA传送时）
- 通常为操作系统设置“cache冲刷”指令，因此，cache对操作系统程序员不是透明的！





直接映射方式举例

- 主存和Cache之间直接映射，块大小为16B。Cache的数据区容量为64KB，主存地址为32位，按字节编址。
要求：说明主存地址如何划分和访存过程。

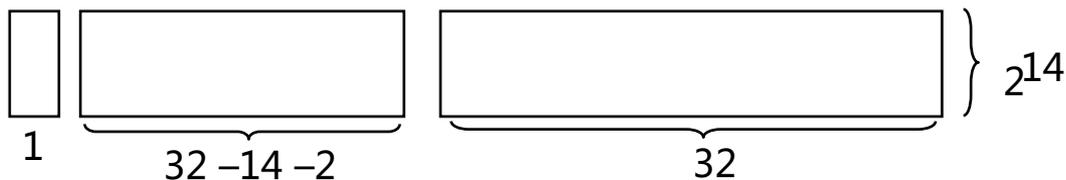


容量 $4K \times (1+16) + 64K \times 8 = 580Kbits = 72.5KB$ ，数据占 $64KB / 72.5KB = 88.3\%$



如何计算Cache的容量

- Cache : 64行 , 块大小为16字节 , 那么地址1200存放在哪一行 ?
 - 地址1200对应存放在第11行。因为 : $[1200/16=75] \text{ module } 64 = 11$
 $1200 = 1024+128+32+16 = 0\dots01\boxed{001011}0000 \text{ B}$
- 实现以下cache需要多少位容量 ? Cache : 直接映射、16K行数据、块大小为1个字 (4B)、32位主存地址。
 - 答 : Cache的存储布局如下 : Cache共有16K x 4B= 64KB数据



- 所以 , Cache的大小为 : $2^{14} \times (32 + (32-14-2)+1) = 2^{14} \times 49 = 784 \text{ Kbits}$
 若块大小为4个字呢 ? $2^{14} \times (4 \times 32 + (32-14-2-2)+1) = 2^{14} \times 143 = 2288 \text{ Kbits}$
 若块大小为 2^m 个字呢 ? $2^{14} \times (2^m \times 32 + (32-14-2-m)+1)$

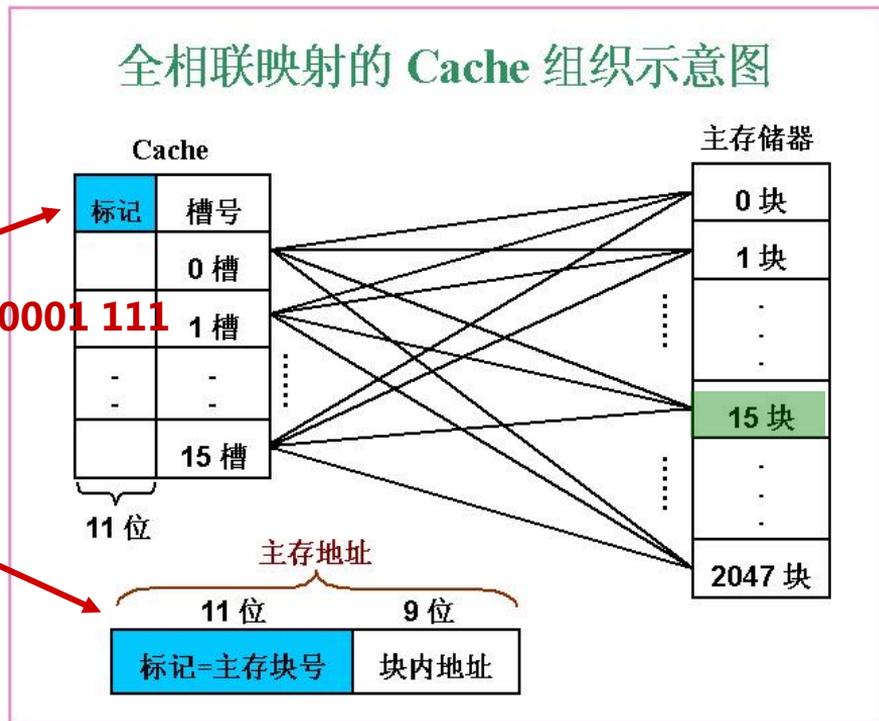


全相连映射的组织示意图

- 假定数据在主存和Cache间的传送单位为512字。
- **Cache大小**： 2^{13} 字=8K字=16行 x 512字/行
- **主存大小**： 2^{20} 字=1024K字=2048块 x 512字/块
- **按内容访问，是相联存取方式！**
- **如何实现按内容访问？直接比较！**
- Cache标记 (tag) 指出对应行取自哪个主存块
- 主存tag指出对应地址位于哪个主存块
- **如何对01E0CH单元进行访问？**
0000 0001 1110 0000 1100B 是第15块中的第12个单元！

每个主存块可装到Cache任一行中。

全相联映射的 Cache 组织示意图



为何地址中没有cache索引字段？因为可映射到任意一个cache行中！



组相连映射

- **组相联映射结合直接映射和全相联映射的特点**
- 将Cache所有行分组，把主存块映射到Cache固定组的任一行中。
 - 也即：组间模映射、组内全映射。
 - 映射关系为： $\text{Cache组号} = \text{主存块号} \bmod \text{Cache组数}$
 - 举例：假定Cache划分为：8K字=8组x2行/组x512字/行
 $4 = 100 \bmod 8$ (主存第100块应映射到Cache的第4组的任意行中。)
- **特点：**
 - 结合直接映射和全相联映射的优点。当Cache组数为1时，变为相联映射；当每组只有一个槽时，变为直接映射。
 - 每组2或4行（称为2-路或4-路组相联）较常用。通常每组4行以上很少用。在较大容量的L2 Cache和L3 Cache中使用4-路以上。





提问

Q & A

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



南京大學
NANJING UNIVERSITY