第6章 存储器层次结构

存储器概述

半导体随机存取存储器

外部存储器

高速缓冲存储器(cache)

层次结构存储系统

- 主要教学目标
  - 理解CPU执行指令过程中为何要访存
  - 理解访存操作的大致过程及涉及到的部件
  - 了解层次化存储器系统的由来及构成
  - 了解CPU与主存储器之间的连接及读写操作
  - 掌握Cache机制并理解其对程序性能的影响
  - 理解程序局部性的重要性并能开发局部性好的程序

层次结构存储系统

- 分以下四个部分介绍
  - 第一讲：存储器概述
  - 第二讲：半导体随机存取存储器
    - 基本存储元件、DRAM芯片、 SDRAM芯片技术
    - 内存条及其与CPU的连接
    - 存储器芯片的扩展、主存控制器
  - 第三讲：外部存储器
    - 磁盘存储器、闪速存储器、U盘、固态硬盘
  - 第四讲：高速缓冲存储器(cache)
    - cache的基本工作原理
    - 映射方式、替换算法、写策略
    - Cache的设计
    - Cache和程序性能

回顾：程序及指令的执行过程

° 在内存存放的指令实际上是机器代码（0/1序列）

```
08048394 <add>:
1  8048394:   55           push   %ebp
2  8048395:   89  e5        mov    %esp, %ebp
3  8048397:   8b  45 0c     mov    0xc(%ebp), %eax
4  804839a:   03  45 08     add    0x8(%ebp), %eax
5  804839d:   5d           pop    %ebp
6  804839e:   c3           ret
```

栈是主存中的一个区域！

° 对于add函数的执行，何时需要访存？

- ü 每条指令都需从主存单元取到CPU执行 取指
- ü PUSH指令需把寄存器内容压入栈中 存数 POP指令则相反 取数
- ü 第3条mov指令需要从主存中取数后送到寄存器 取数
- ü 第4条add指令需要从主存取操作数到ALU中进行运算 取数
- ü ret指令需要从栈中取出返回地址，以能正确回到调用程序执行 取数

访存是指令执行过程中一个非常重要的环节！ 取指、取数、存数

基本术语

- 记忆单元 （存储基元 / 存储元 / 位元） （Cell）
  - 具有两种稳态的能够表示二进制数码0和1的物理器件
- 存储单元 / 编址单位（Addressing Unit）
  - 具有相同地址的位构成一个存储单元，也称为一个编址单位
- 存储体/ 存储矩阵 / 存储阵列（Bank）
  - 所有存储单元构成一个存储阵列
- 编址方式（Addressing Mode）
  - 字节编址、按字编址
- 存储器地址寄存器（Memory Address Register – MAR）
  - 用于存放主存单元地址的寄存器
- 存储器数据寄存器（ Memory Data Register–MDR（或MBR） ）
  - 用于存放主存单元中的数据的寄存器

存储器分类

依据不同的特性有多种分类方法

（1）按工作性质/存取方式分类

- 随机存取存储器 Random Access Memory（RAM）
  - 每个单元读写时间一样，且与各单元所在位置无关。如：内存。
    （注：原意主要强调地址译码时间相同。）
- 顺序存取存储器 Sequential Access Memory（SAM）
  - 数据按顺序从存储载体的始端读出或写入，因而存取时间的长短与信息所在位置有关。例如：磁带。
- 直接存取存储器 Direct Access Memory(DAM)
  - 直接定位到读写数据块，在读写数据块时按顺序进行。如磁盘。
- 相联存储器 Associate Memory（AM）
  Content Addressed Memory（CAM）
  - 按内容检索到存储位置进行读写。例如：快表。

存储器分类

（2）按存储介质分类

半导体存储器：双极型，静态MOS型，动态MOS型

磁表面存储器：磁盘（Disk）、磁带 （Tape）

光存储器：CD，CD-ROM，DVD

（3）按信息的可更改性分类

读写存储器（Read / Write Memory)：可读可写

只读存储器（Read Only Memory)：只能读不能写

（4）按断电后信息的可保存性分类

非易失（不挥发）性存储器(Nonvolatile Memory)

信息可一直保留，　不需电源维持。

（如 ： ROM、U盘、磁表面存储器、光存储器等）

易失（挥发）性存储器(Volatile Memory)

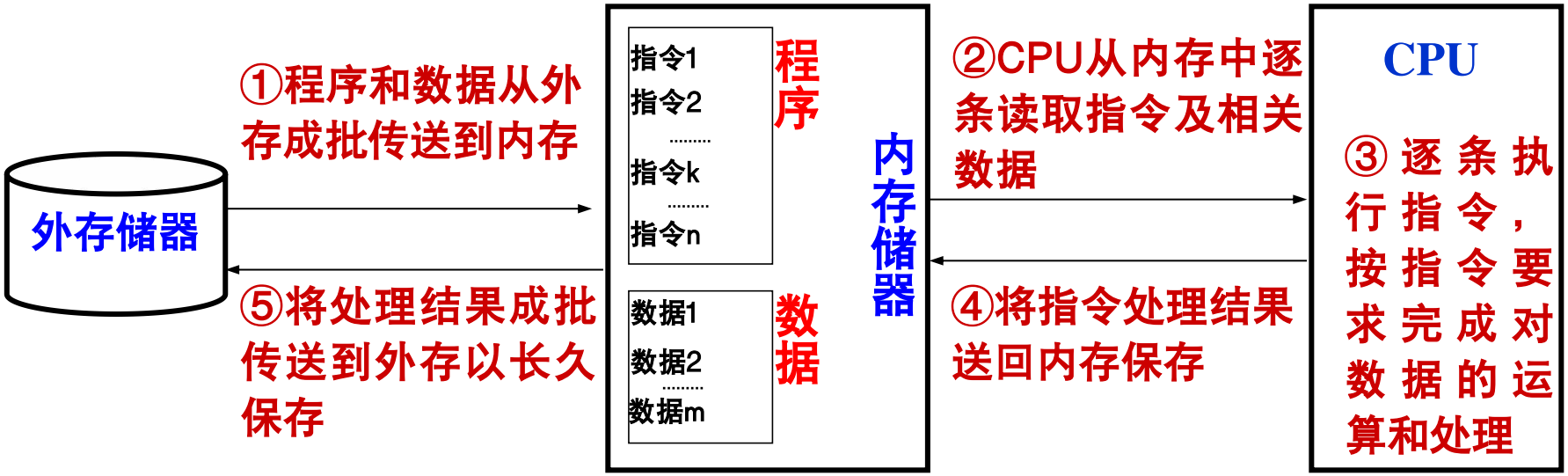
电源关闭时信息自动丢失。（如：RAM、Cache等）

存储器分类

（5）按功能/容量/速度/所在位置分类

- 寄存器(Register)
  - 封装在CPU内，用于存放当前正在执行的指令和使用的数据
  - 用触发器实现，速度快，容量小（几~几百个）
- 高速缓存(Cache)
  - 位于CPU内部或附近，用来存放当前要执行的局部程序段和数据
  - 用SRAM实现，速度可与CPU匹配，容量小（几MB）
- 内存储器MM（主存储器Main（Primary）Memory）
  - 位于CPU之外，用来存放已被启动的程序及所用的数据
  - 用DRAM实现，速度较快，容量较大（几GB）
- 外存储器AM（辅助存储器Auxiliary / Secondary Storage）
  - 位于主机之外，用来存放暂不运行的程序、数据或存档文件
  - 用磁盘、SSD等实现，容量大而速度慢

内存与外存的关系及比较



ü 外存储器（简称外存或辅存）

- 存取速度慢
- 成本低、容量很大
- 不与CPU直接连接，先传送到内存，然后才能被CPU使用。
- 属于非易失性存储器，用于长久存放系统中几乎所有的信息

ü 内存储器（简称内存或主存）

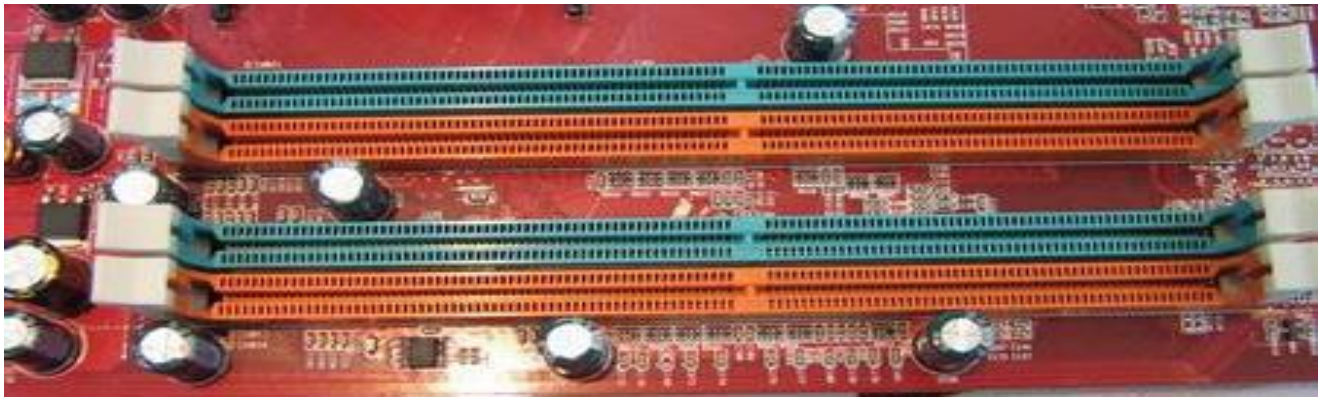
- 存取速度快
- 成本高、容量相对较小
- 直接与CPU连接，CPU对内存中可直接进行读、写操作
- 属于易失性存储器(volatile)，用于临时存放正在运行的程序和数据

PC机主存储器的物理结构

- 内存条的组成:

  把若干片DRAM芯片焊装在一小条印制电路板上制成

- 内存条必须插在主板上的内存条插槽中才能使用



主存的基本结构

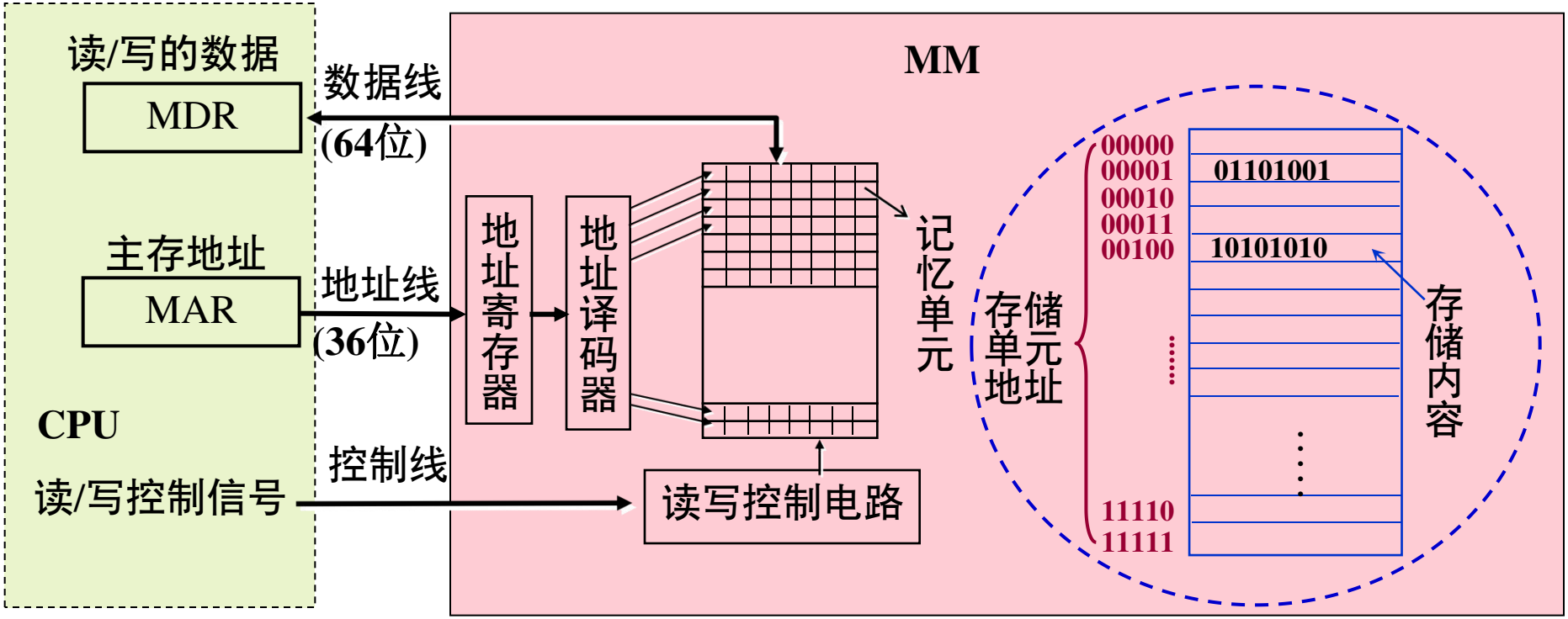
问题：主存中存放的是什么信息？CPU何时会访问主存？

指令及其数据！CPU执行指令时需要取指令、取数据、存数据！

问题：地址译码器的输入是什么？输出是什么？可寻址范围多少？

输入是地址，输出是地址驱动信号（只有一根地址驱动线被选中）。

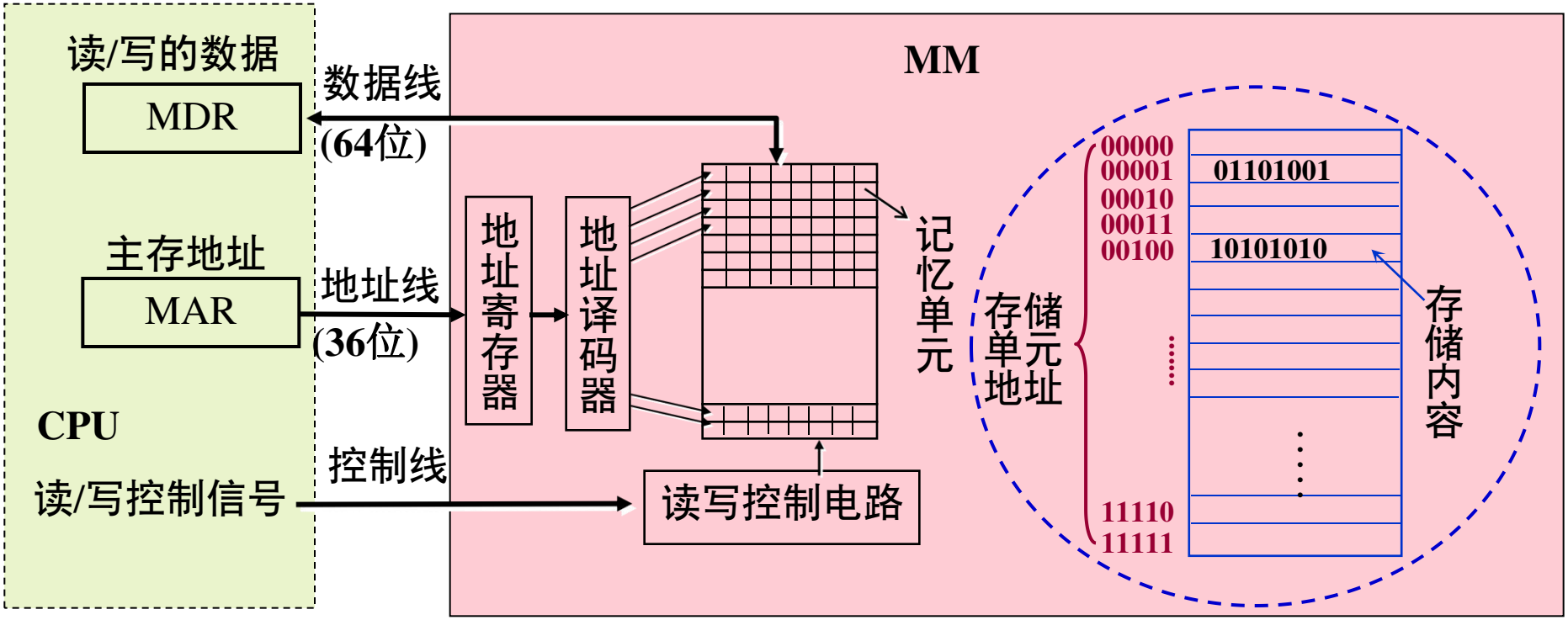
可寻址范围为0~2^{36}–1，即主存地址空间为64GB（按字节编址时）。



主存地址空间大小不等于主存容量（实际安装的主存大小）！

若是字节编址，则每次最多可读/写8个单元，给出的是首(最小)地址.

主存的基本结构



- 图中的MDR和MAR属于CPU中的总线接口部件。
- 上述仅是示意图。实际上CPU并非与主存芯片直接交互，而是先与主存控制器（memory controller）交互，再由主存控制器控制主存芯片进行读写。
- 现代处理器采用DRAM作为主存，因此主存控制器也称为DRAM控制器。

访存指令访问主存的过程

① 若CPU支持虚存，则需将指令给出的虚拟（逻辑）地址转换成主存（物理）地址

② 通过主存地址查询高速缓存，若命中，则直接访问高速缓存中的内容

③ 若不命中，则通过系统总线向DRAM控制器发送访存请求事务，具体将通过地址线发送主存地址，通过控制线发送读/写信号及其他控制信息，若为写操作，则还需通过数据线发送写入的数据

④ DRAM控制器接收到访存请求事务后，根据控制线上的信号将该访存请求事务转换为与DRAM芯片通信的存储器总线请求，具体包括DRAM芯片内部地址和DRAM芯片的命令，若为写操作，则还包括写入的数据

⑤ DRAM芯片通过地址译码器对DRAM芯片内部地址进行译码，并根据命令访问选中的存储单元：若为写操作，则将数据写入选中的存储单元；若为读操作，则读出选中存储单元的内容，并通过存储器总线返回给DRAM控制器

⑥ DRAM控制器向高速缓存返回系统总线请求事务的回复，若为读操作，则同时返回从DRAM芯片读出的数据。

⑦ 高速缓存根据系统总线请求事务的回复更新缓存内容，若为读操作，则向CPU返回读出的数据。

主存的主要性能指标

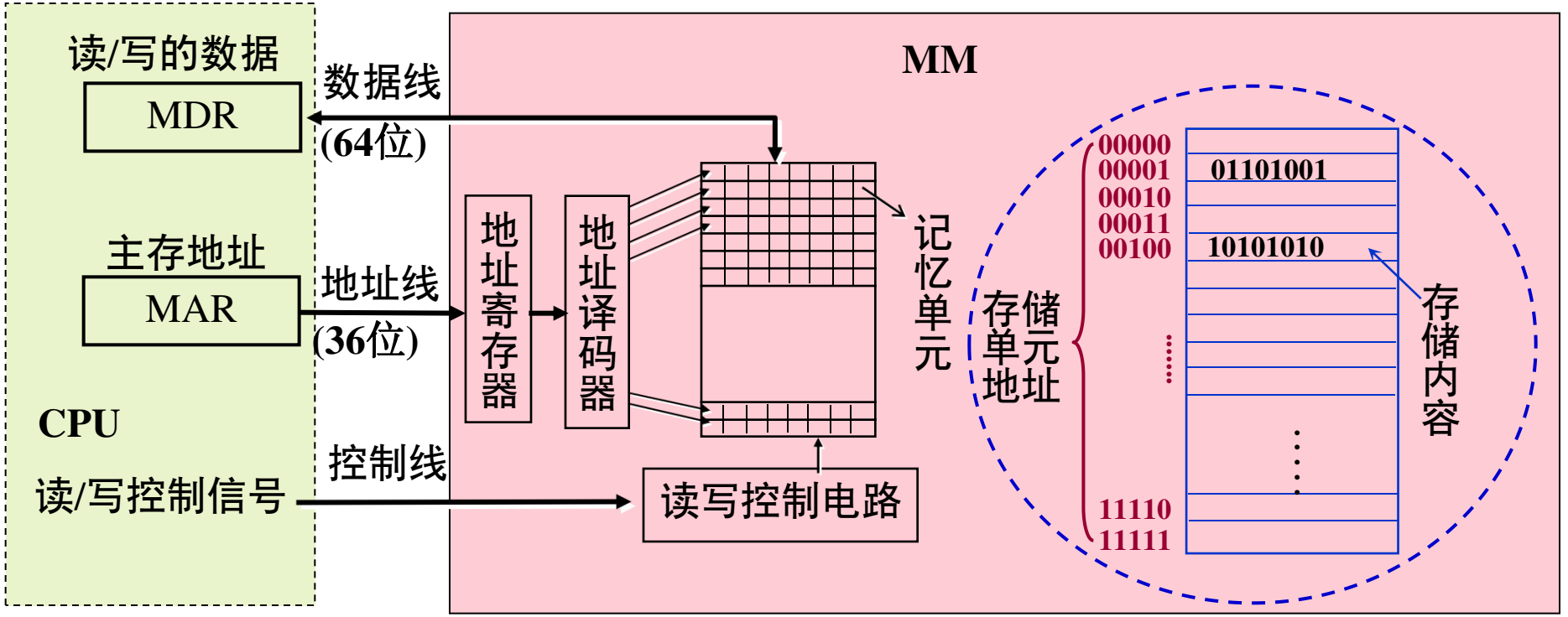
- 性能指标:

  按字节连续编址，每个存储单元为1个字节（8个二进位）

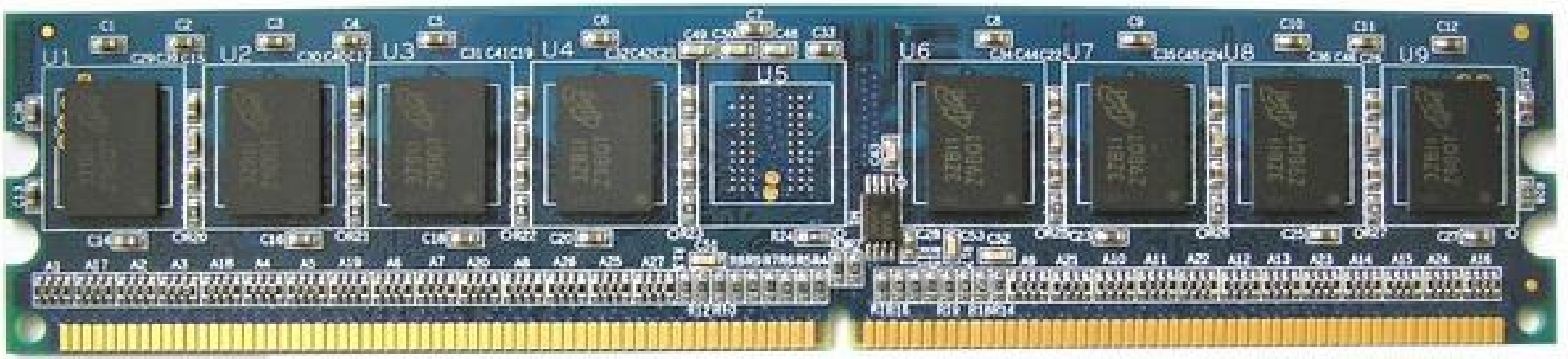
  - 存储容量：所包含的存储单元的总数（单位：MB或GB）
  - 存取时间T_A：从CPU送出内存单元的地址码开始，到主存读出数据并送到CPU（或者是把CPU数据写入主存）所需要的时间（单位：ns，1 ns = 10^{-9} s），分读取时间和写入时间
  - 存储周期T_{MC}：连读两次访问存储器所需的最小时间间隔，它应等于存取时间加上下一次存取开始前所要求的附加时间，因此，T_{MC}比T_A大（ 因为存储器由于读出放大器、驱动电路等都有一段稳定恢复时间，所以读出后不能立即进行下一次访问。 ）

  （就像一趟货车运货时间和发车周期是两个不同概念一样。）

主存的基本结构



数据线、地址线、控制线对应芯片引脚，内存条插槽就是存储器总线



时间、存储容量（或带宽）的单位

Notations and Conventions for Numbers

Prefix	Abbreviation	Meaning	Numeric Value
mill	m	One thousandth	10^{-3}
micro	μ	One millionth	10^{-6}
nano	n	One billionth	10^{-9}
pico	p	One trillionth	10^{-12}
femto	f	One quadrillionth	10^{-15}
atta	a	One quintillionth	10^{-18}
kilo	K (or k)	Thousand	10^{3} or 2^{10}
mega	M	Million	10^{6} or 2^{20}
giga	G	Billion	10^{9} or 2^{30}
tera	T	Trillion	10^{12} or 2^{40}
peta	P	Quadrillion	10^{15} or 2^{50}
exa	E	Quintillion	10^{18} or 2^{60}

为避免混淆，Patterson和Hennessy将2的幂次和10的幂次进行了区分。

回顾： 2的幂次和10的幂次单位

- 硬盘和文件使用的单位
  - 不同的硬盘制造商和操作系统用不同的度量方式，因而比较混乱
  - 为避免歧义，国际电工委员会（IEC）给出了二进制前缀字母定义，可用不同的前缀表示所采用的度量方式

十进制前缀			IEC 定义的二进制前缀			值差
单词	前缀	值	单词	前缀	值	（%）
kilobyte	KB/kB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

希望的理想存储器

到目前为止，已经了解到有以下几种存储器:

寄存器，SRAM，DRAM， SSD、硬盘

	Capacity	Latency	Cost
Register	<1KB	1ns	$$$$
SRAM	1MB	2ns	$$$
DRAM	1GB	10ns	$
Hard disk*	1000GB	10ms	¢
Want	100GB	1ns	cheap

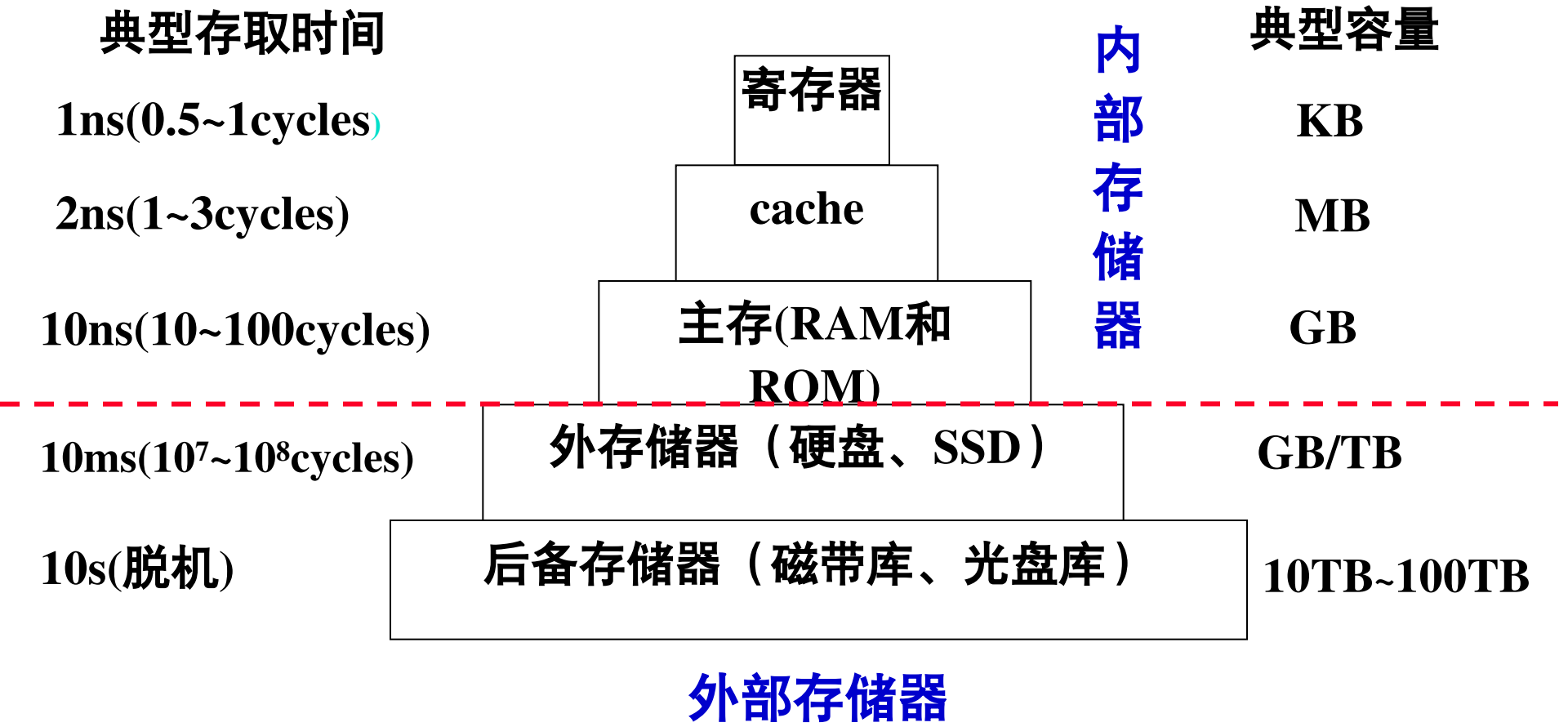
* non-volatile

问题：你认为哪一种最适合做计算机的存储器呢?

单独用某一种存储器，都不能满足我们的需要!

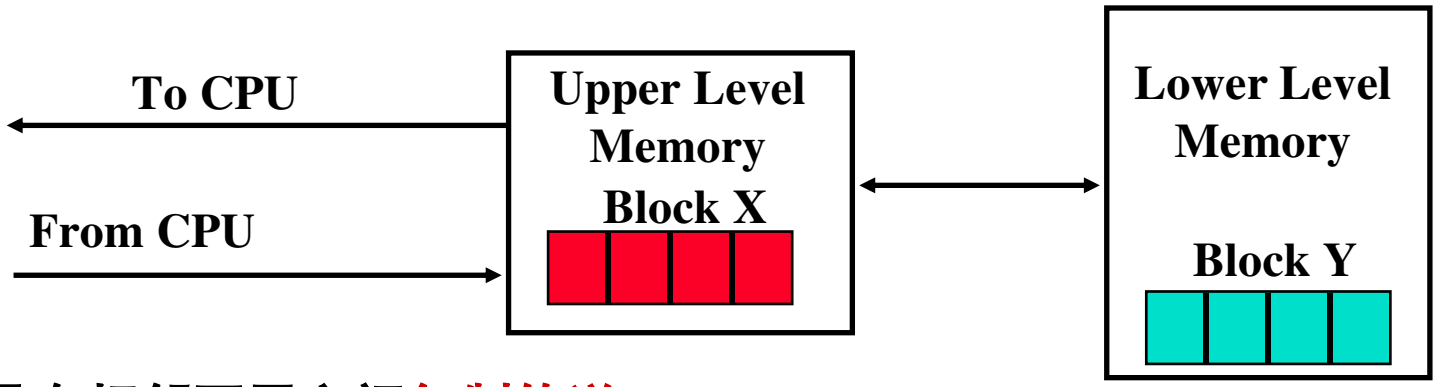
采用分层存储结构来构建计算机的存储体系!

存储器的层次结构



列出的时间和容量会随时间变化，但数量级相对关系不变。

层次化存储器结构（Memory Hierarchy）



数据总是在相邻两层之间复制传送

Upper Level: 上层更靠CPU

Lower Level: 下层更远离CPU

相当于工厂中设置了多级仓库!

Block: 传送单位，比所需数据块大得多，互为副本

问题：为什么这种层次化结构是有效的?

程序访问局部化特点!
例如，写论文时图书馆借参考书：欲借书附近的书也是欲借书!

- 时间局部性（Temporal Locality）

  含义：刚被访问过的单元很可能不久又被访问

  做法：让最近被访问过的信息保留在靠近CPU的存储器中

- 空间局部性 （Spatial Locality）

  含义：刚被访问过的单元的邻近单元很可能不久被访问

  做法：将刚被访问过的单元的邻近单元调到靠近CPU的存储器中

加快访存速度措施之三：引入Cache

- 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！

程序的局部性原理举例1

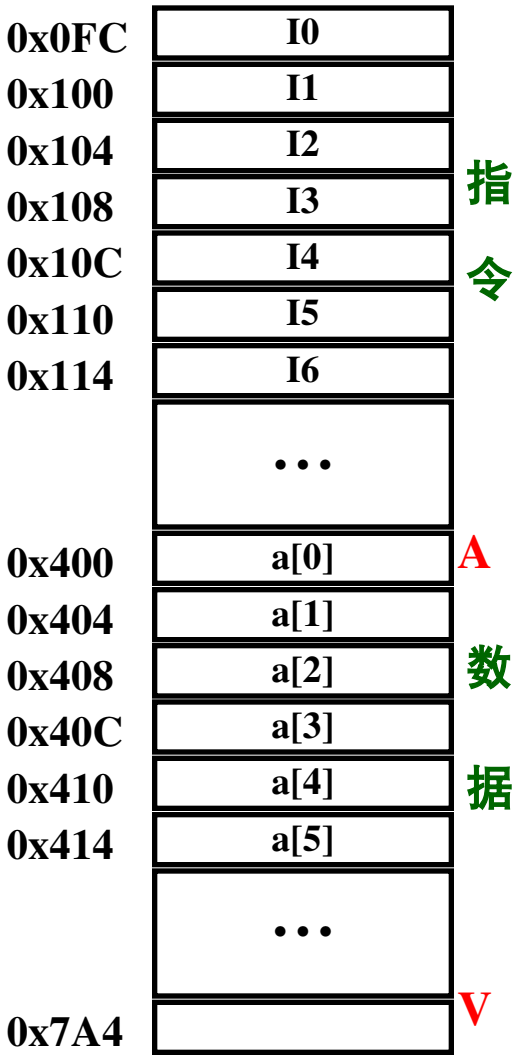
高级语言源程序

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

对应的汇编语言程序

```
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

程序的局部性原理举例1

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

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）

循环n次

→0x120（I9）

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！

数据：只有数组在主存中：

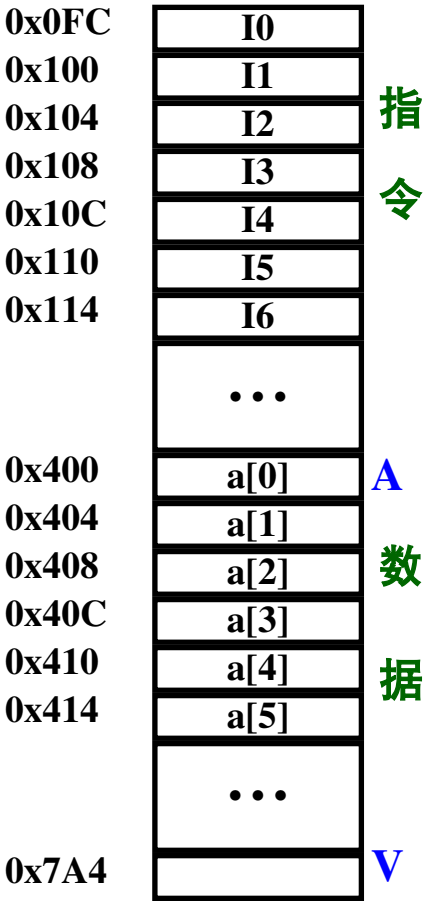
0x400→0x404→0x408

→0x40C→……→0x7A4

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。

主存的布局：



程序的局部性原理举例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？

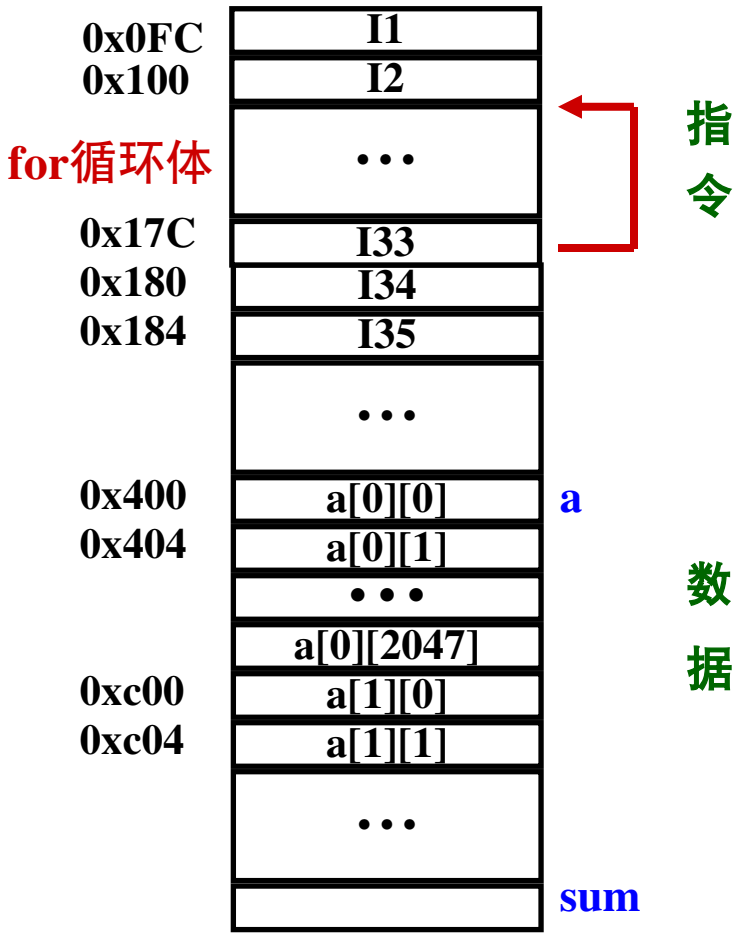
程序段A:

```
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]；

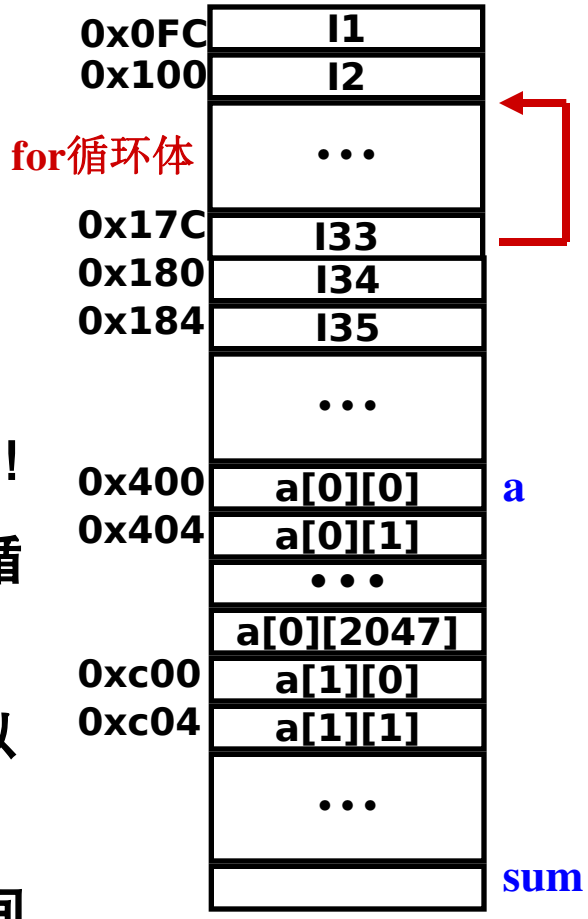
……，与存放顺序一致，故空间局部性好！

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

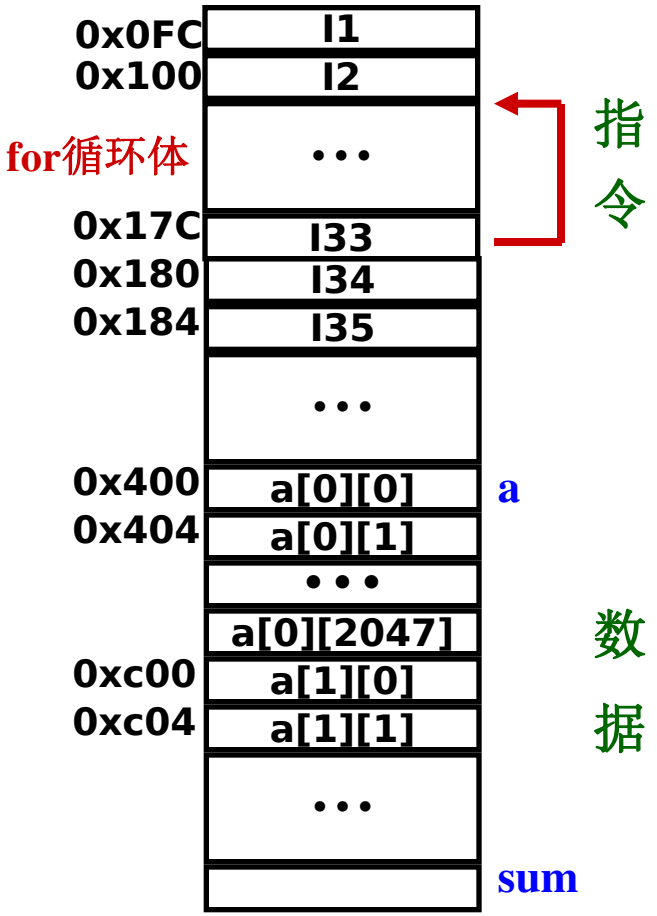
程序段B的时间局部性和空间局部性分析

（1）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）

（2）变量sum: （同程序A ）

（3） 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

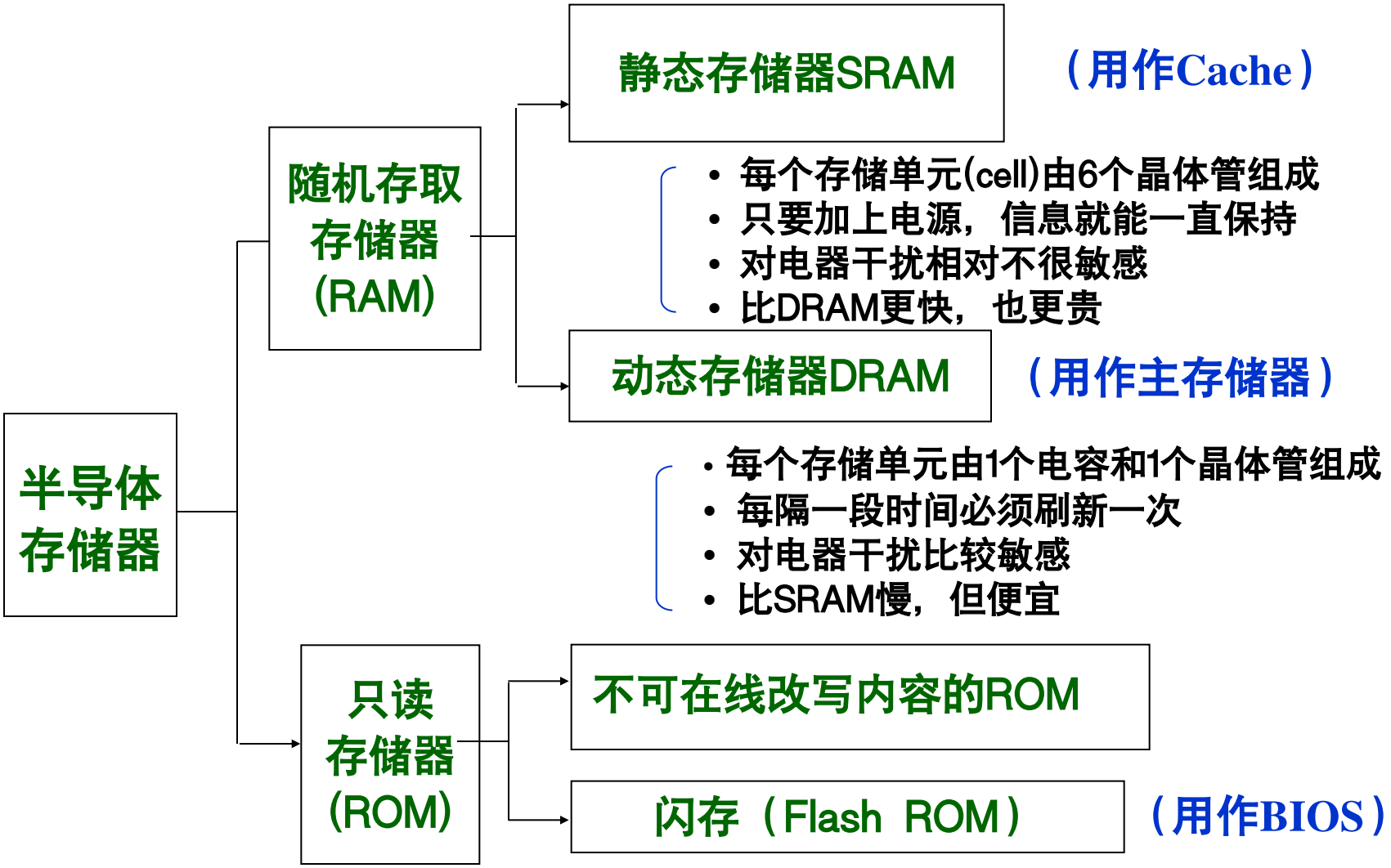
程序A比程序B快21.5 倍!!

层次结构存储系统

- 分以下四个部分介绍
  - 第一讲：存储器概述
  - 第二讲：半导体随机存取存储器
    - 基本存储元件、DRAM芯片、 SDRAM芯片技术
    - 内存条及其与CPU的连接
    - 存储器芯片的扩展、主存控制器
  - 第三讲：外部存储器
    - 磁盘存储器、闪速存储器、U盘、固态硬盘
  - 第四讲：高速缓冲存储器(cache)
    - cache的基本工作原理
    - 映射方式、替换算法、写策略
    - Cache的设计
    - Cache和程序性能

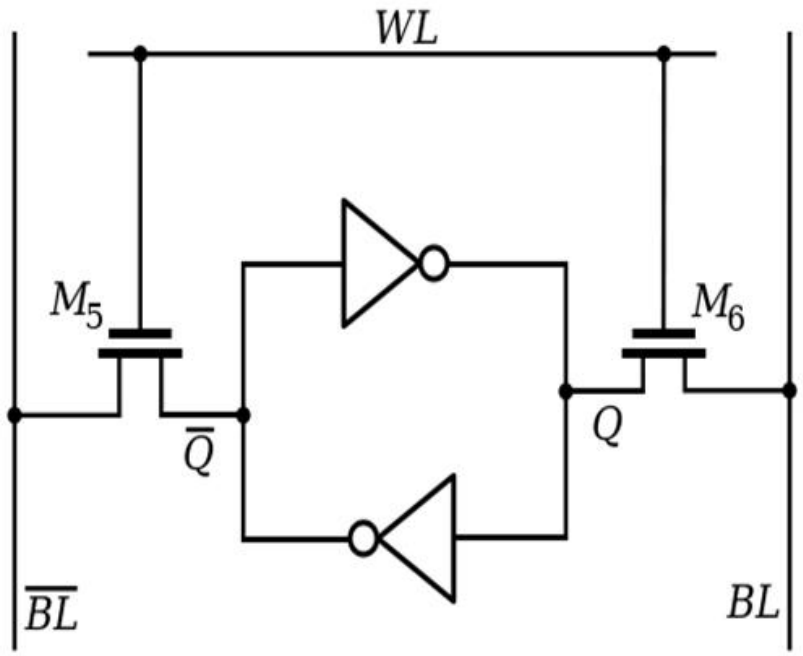
回顾：内部存储器的分类及应用

- 内部存储器由半导体存储器芯片组成，芯片有多种类型：



回顾：六管静态MOS管电路

6管静态NMOS记忆单元



使用6个MOS管组成一个存储元件，其中一个反相器由两个MOS管构成

两个反相器反向连接构成1位锁存器，用于存储信息Q，若Q点为高电平，则存储状态为1，否则为0

保持时：

- 字线WL为0（低电平）

写入时：

- 位线BL和$\overline{BL}$上是被写入的二进位信息0或1
- 置字线WL为1
- 存储单元按位线上的状态设置成0或1

读出时：

- 置2个位线为高电平
- 置字线WL为1
- 存储元件状态不同，位线BL和$\overline{BL}$的输出不同

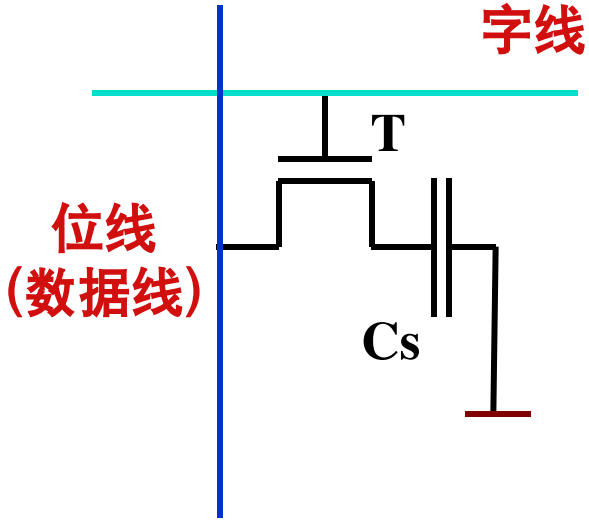
单管动态MOS管电路

读写原理： 字线上加高电平，使T管导通。

写“0”时，数据线加低电平，使C_S上电荷对数据线放电；

写“1”时，数据线加高电平，使数据线对C_S充电；

读出时，数据线上有一读出电压。它与C_S上电荷量成正比。



优点： 电路元件少，功耗小，集成度高，用于构建主存储器

缺点： 速度慢，是破坏性读出（需读后再生），需定时刷新

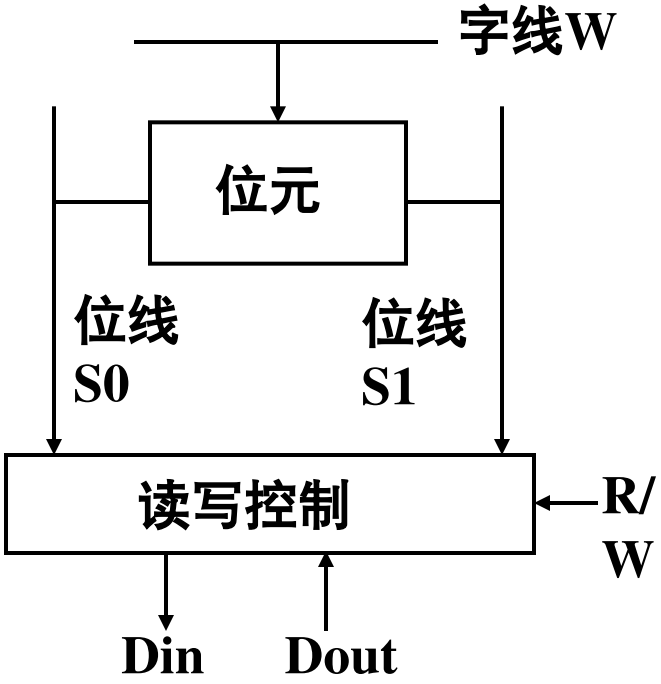
刷新： DRAM的一个重要特点是，数据以电荷的形式保存在电容中，电容的放电使得电荷通常只能维持几十个毫秒左右，相当于1M个时钟周期左右，因此要定期进行刷新（读出后重新写回），按行进行（所有芯片中的同一行一起进行），刷新操作所需时间通常只占1%~2%左右。

半导体RAM的组织

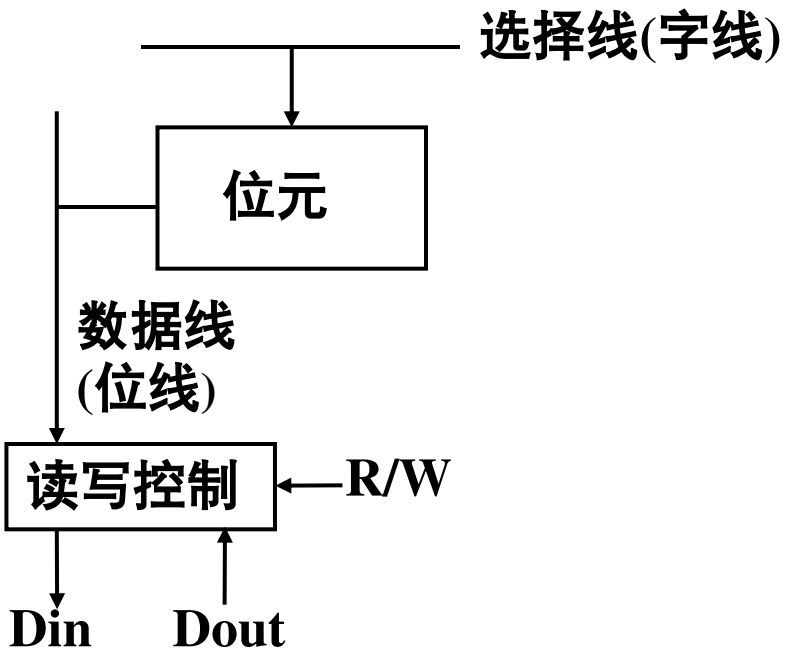
记忆单元(Cell)→存储器芯片(Chip)→ 内存条（存储器模块）

存储体(Memory Bank)： 由记忆单元(位元)构成的存储阵列

记忆单元的组织:

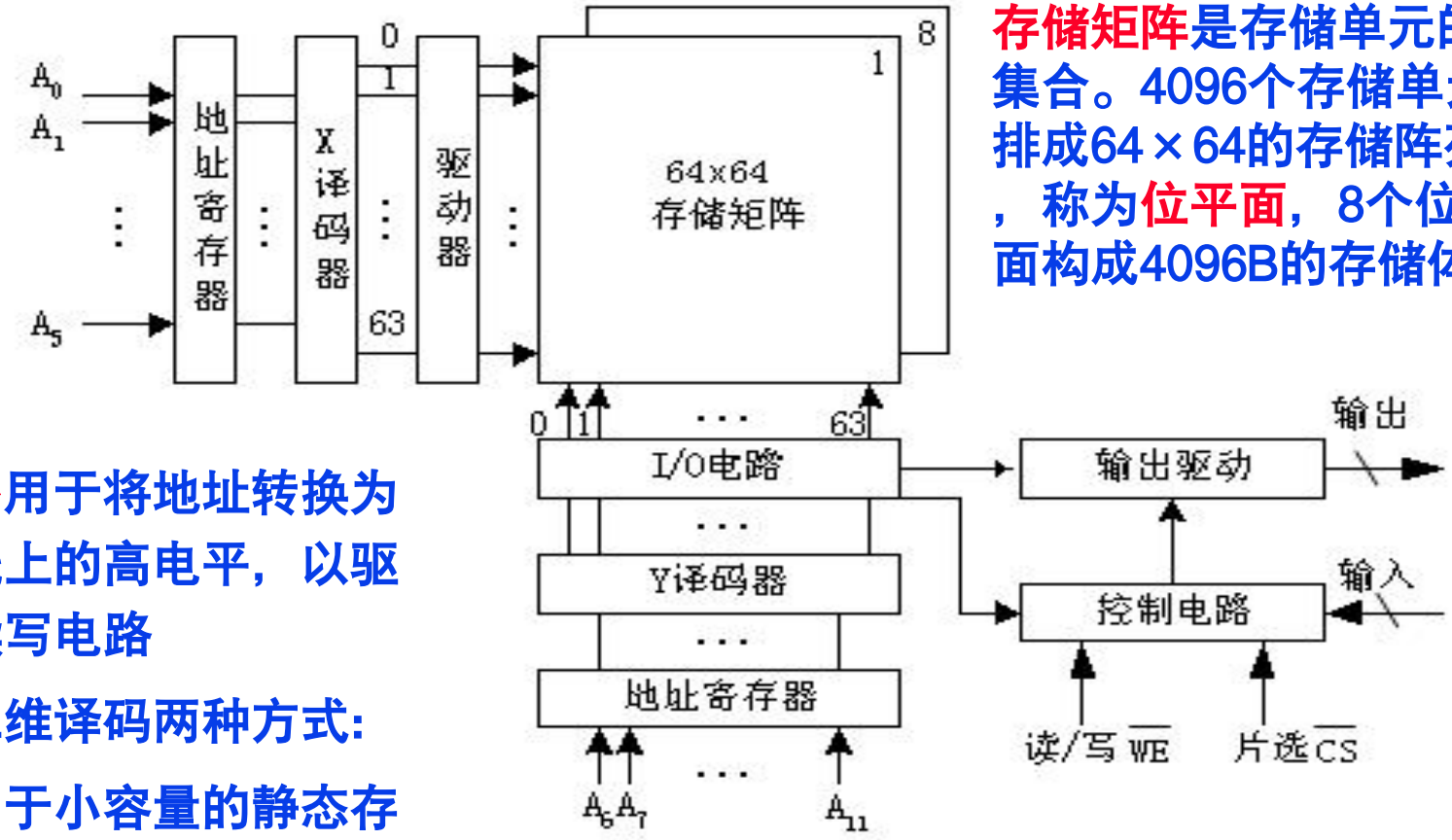


SRAM



DRAM

存储器芯片内部结构



存储矩阵是存储单元的集合。4096个存储单元排成64×64的存储阵列，称为位平面，8个位平面构成4096B的存储体

地址译码器用于将地址转换为译码输出线上的高电平，以驱动相应的读写电路

有一维和二维译码两种方式:

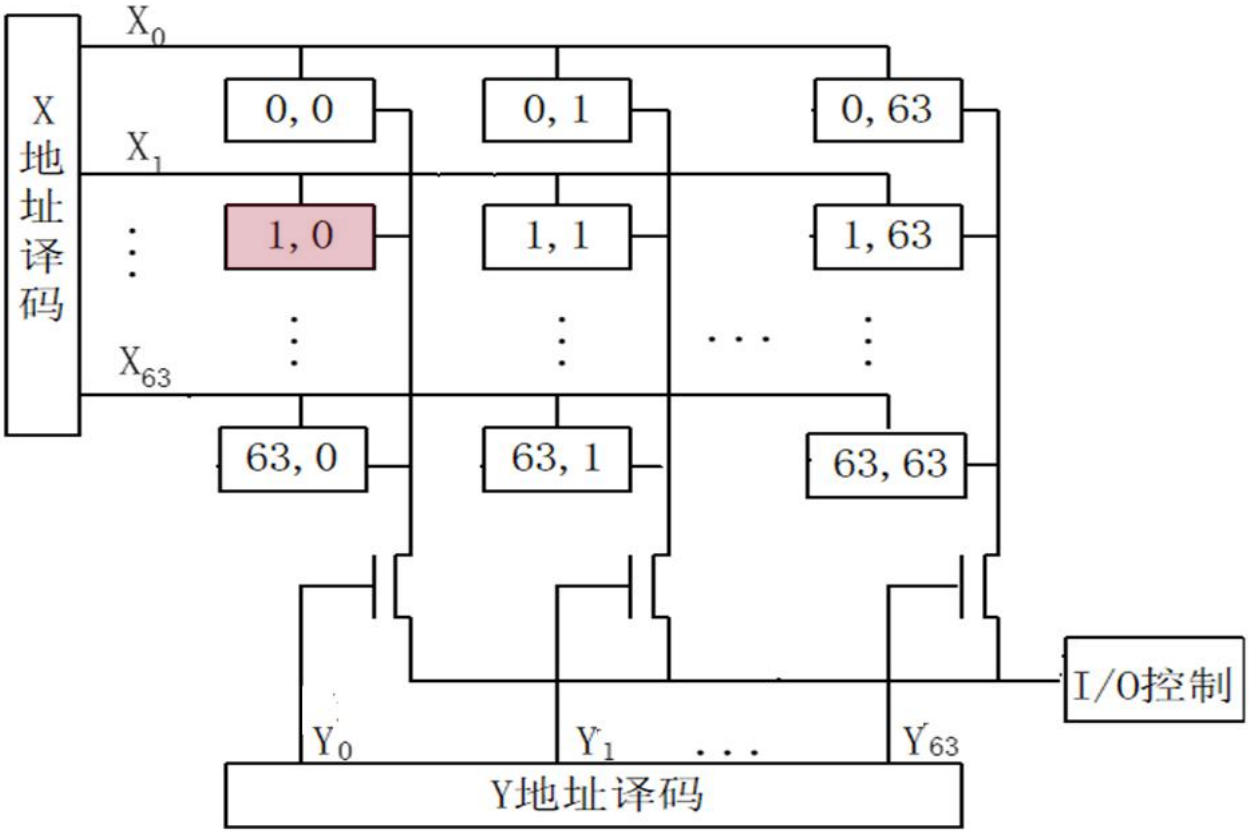
一维方式用于小容量的静态存储器，二维方式用于容量较大的动态存储器

图中为二维方式，在X方向和Y方向进行译码，也称为双译码

一维方式只有一个行地址译码器，同一行中所有存储单元的字线连在一起接到地址译码器输出端，被选中行中的各单元构成一个字，可被同时读出或写入，称为单译码，此结构存储器芯片称为字片式芯片

存储器芯片内部结构

若$A_0A_1\cdots A_{11}$=000001000000，则X地址译码器的译码输出线X_1为高电平，与它相连的64个记忆单元的字选择线为高电平。Y地址译码器的译码输出线Y_0为高电平。在X、Y译码的联合作用下，存储矩阵中坐标（1，0）单元被选中



存储阵列有4096单元，需12根地址线A_0~A_{11}，其中，A_0~A_5送X地址译码器，有64条译码输出线X_0~X_{63}，各连接一行所有记忆单元的字选择线；A_6~A_{11}送Y地址译码器，有64条译码输出线Y_0~Y_{63}，分别控制一列单元的位线控制门

举例：16M位DRAM芯片（4Mx4）

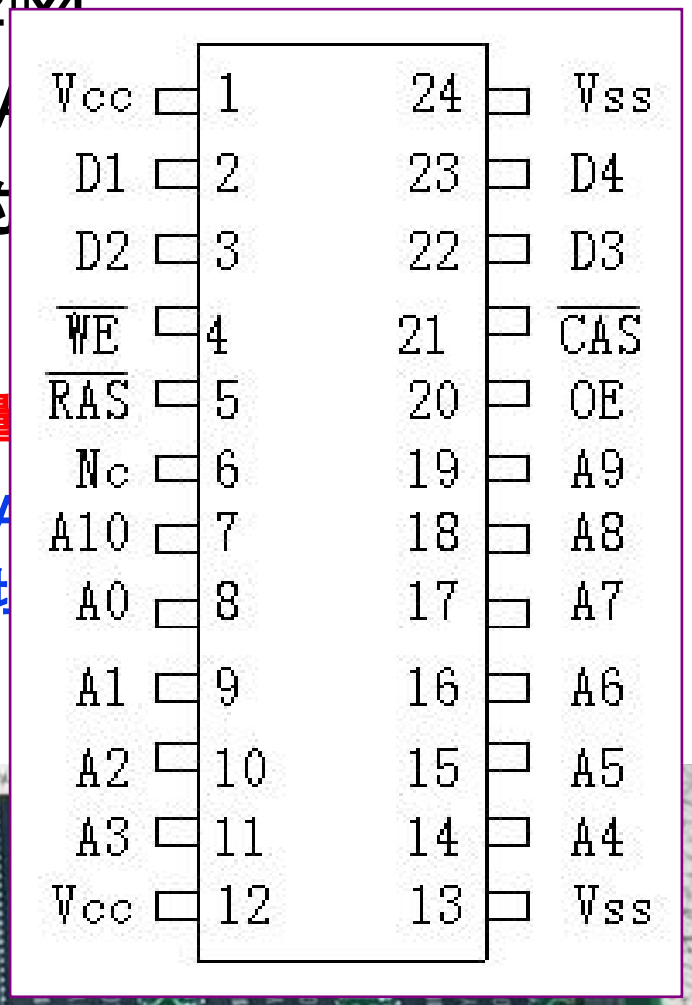
SKIP、

16M位 = 4Mbx4 = 2048x2048x4 = 2^{11}x2^{11}x4

(1) 地址线：11根线分时复用，由RAS和CA

(2) 需4个位平面，对相同行、列交叉点的

(3) 内部结构框图



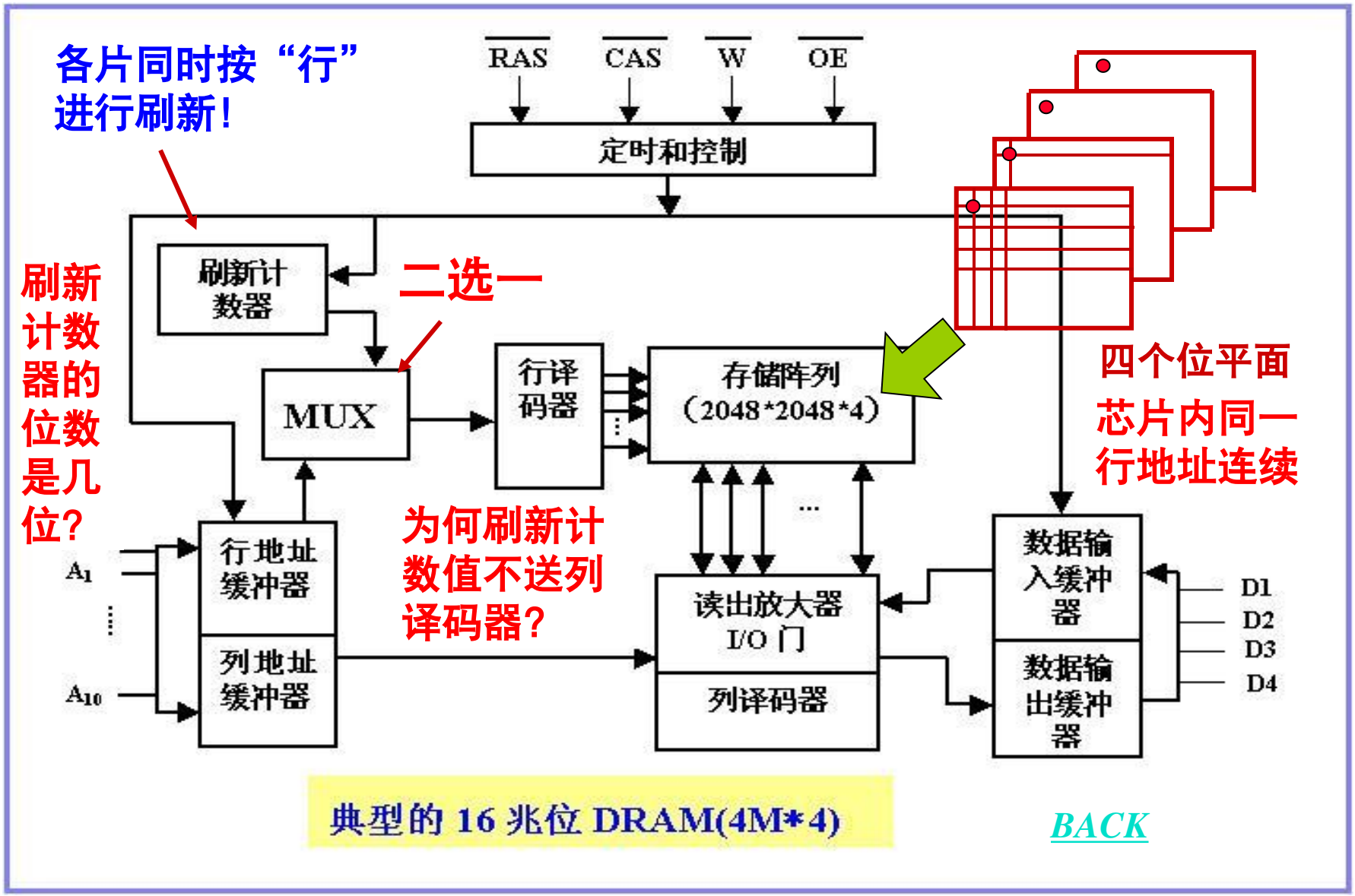
问题：为什么每出现新一代DRAM芯片，容量

行地址和列地址分时复用，每出现新一代DRA

地址线。每加一根地址线，则行地址和列地址

列数各增加一倍。因而容量至少提高到4倍。

举例：典型的16M位DRAM（4Mx4）



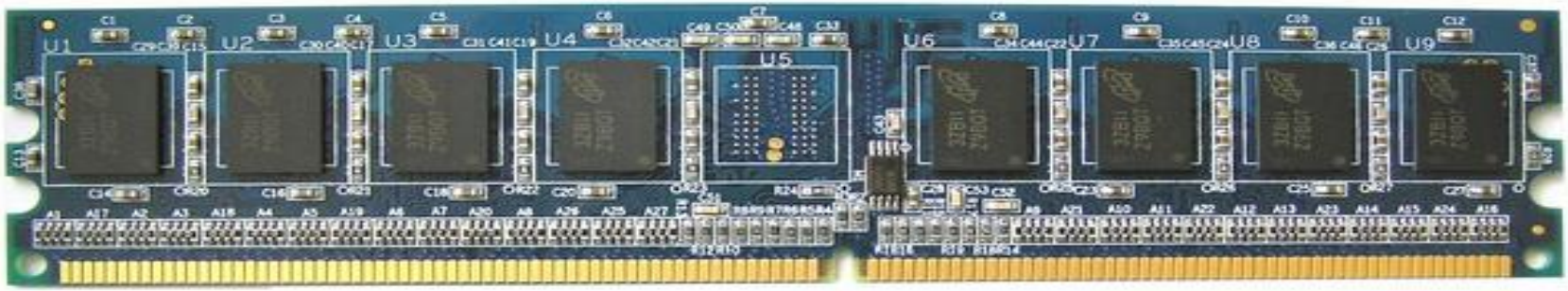
DRAM芯片的规格

- 若一个$2^n \times b$位DRAM芯片的存储阵列是r行×c列，则该芯片容量为$2^n \times b$位且$2^n = r \times c$。如：16K×8位DRAM，则r=c=128。
- 芯片内的地址位数为n，其中行地址位数为$\log_2 r$，列地址位数为$\log_2 c$。如：16K×8位DRAM，则n=14，行、列地址各占7位。
- n位地址中高位部分为行地址，低位部分为列地址
- 为提高DRAM芯片的性价比，通常设置的r和c满足r≤c且|r−c|最小。
  - 例如，对于8K×8位DRAM芯片，其存储阵列设置为2^6行×2^7列，因此行地址和列地址的位数分别为6位和7位，13位芯片内地址$A_{12}\ A_{11}\cdots A_1A_0$中，行地址为$A_{12}\ A_{11}\cdots A_7$，列地址为$A_6\cdots A_1A_0$。因按行刷新，为尽量减少刷新次数，故行数越少越好，但是，为了减少地址引脚，应尽量使行、列地址位数一致

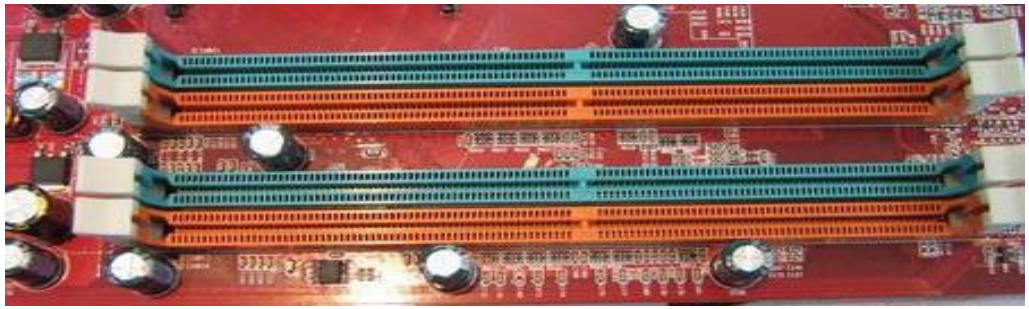
SDRAM芯片技术

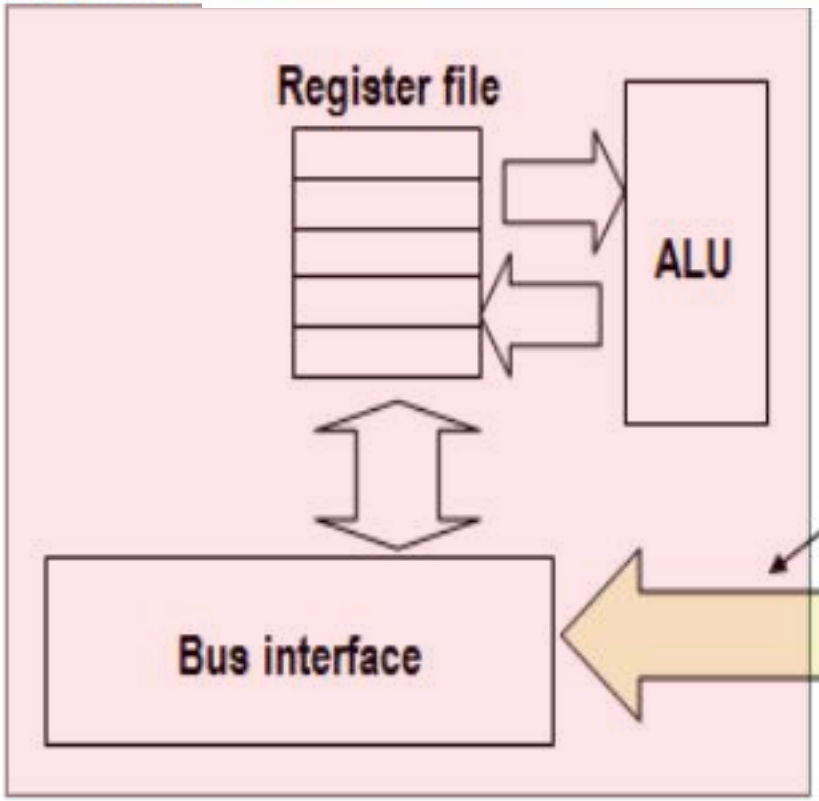
- SDRAM（Synchronous DRAM）是同步存储器芯片
  - 每步操作都在系统时钟控制下进行，有确定的等待时间
  - 支持突发传输（Burst）方式
  - 从收到读命令（与CAS信息同时发送）开始到数据线有效的时间, 称为CAS潜伏期（CAS Latency，CL），例如 CL=2 clks
  - 连续传送数据个数称为突发长度（Burst Length，BL），如 BL=1 / 2 / 4 / 8
  - 多体(缓冲器)交叉存取，实现数据预取
  - 利用总线时钟上升沿与下降沿（每周期2次）同步传送

主存模块的连接和读写操作



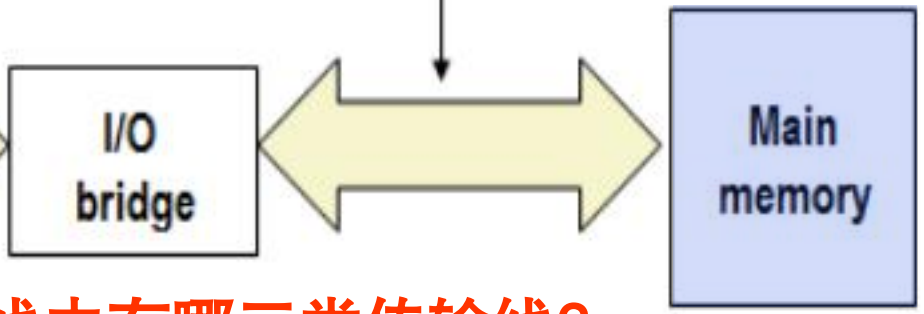
CPU chip





前端总线

存储器总线

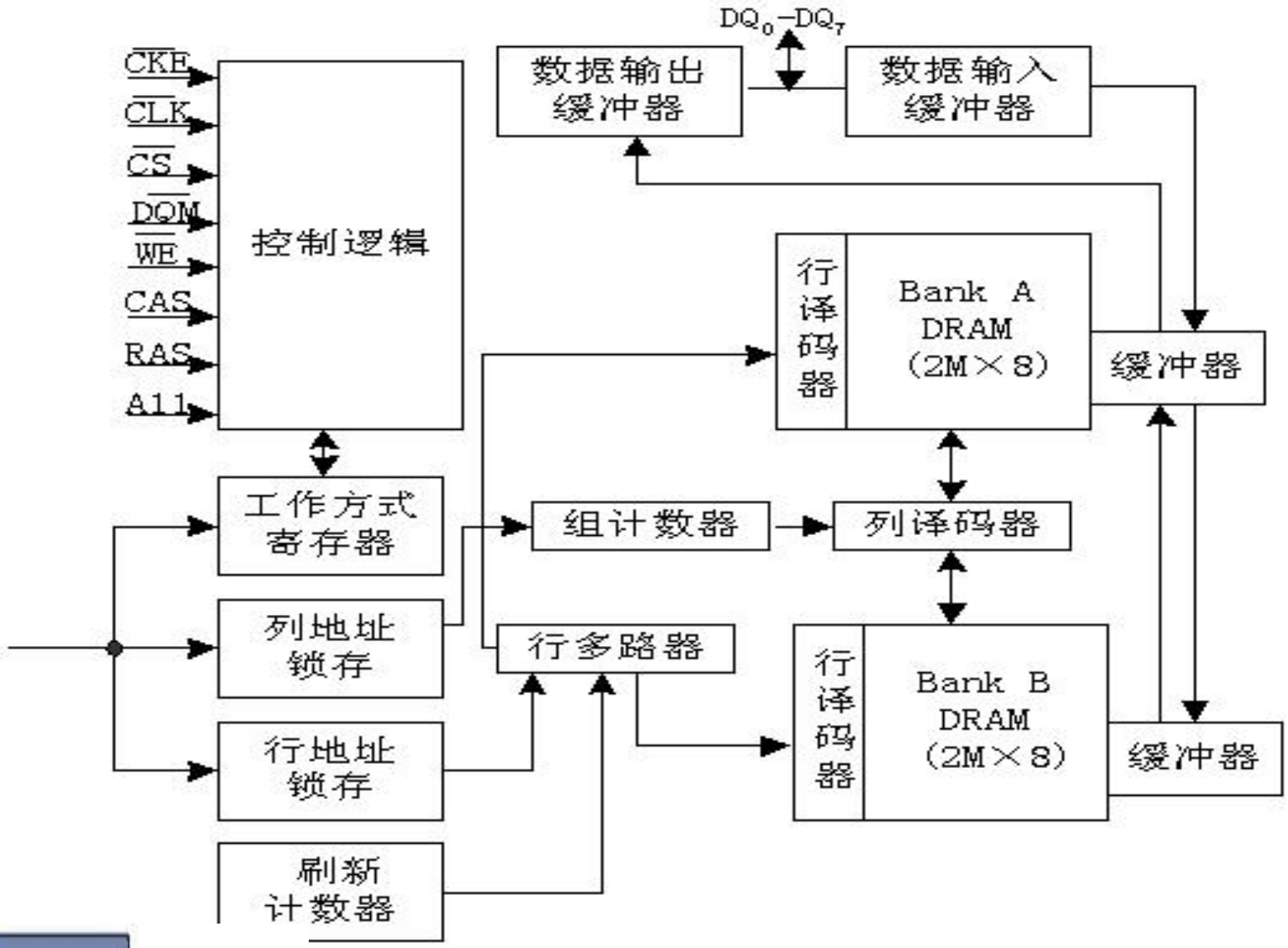


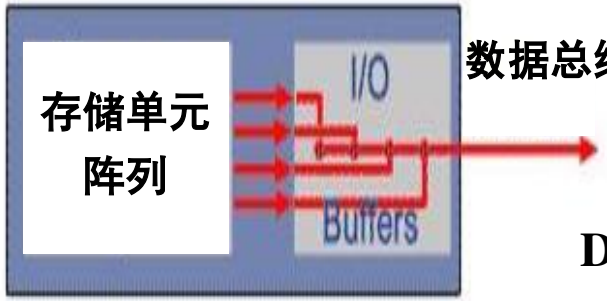
总线中有哪三类传输线？

数据线、地址线、控制线

SDRAM芯片的内部结构

同步方式





数据总线

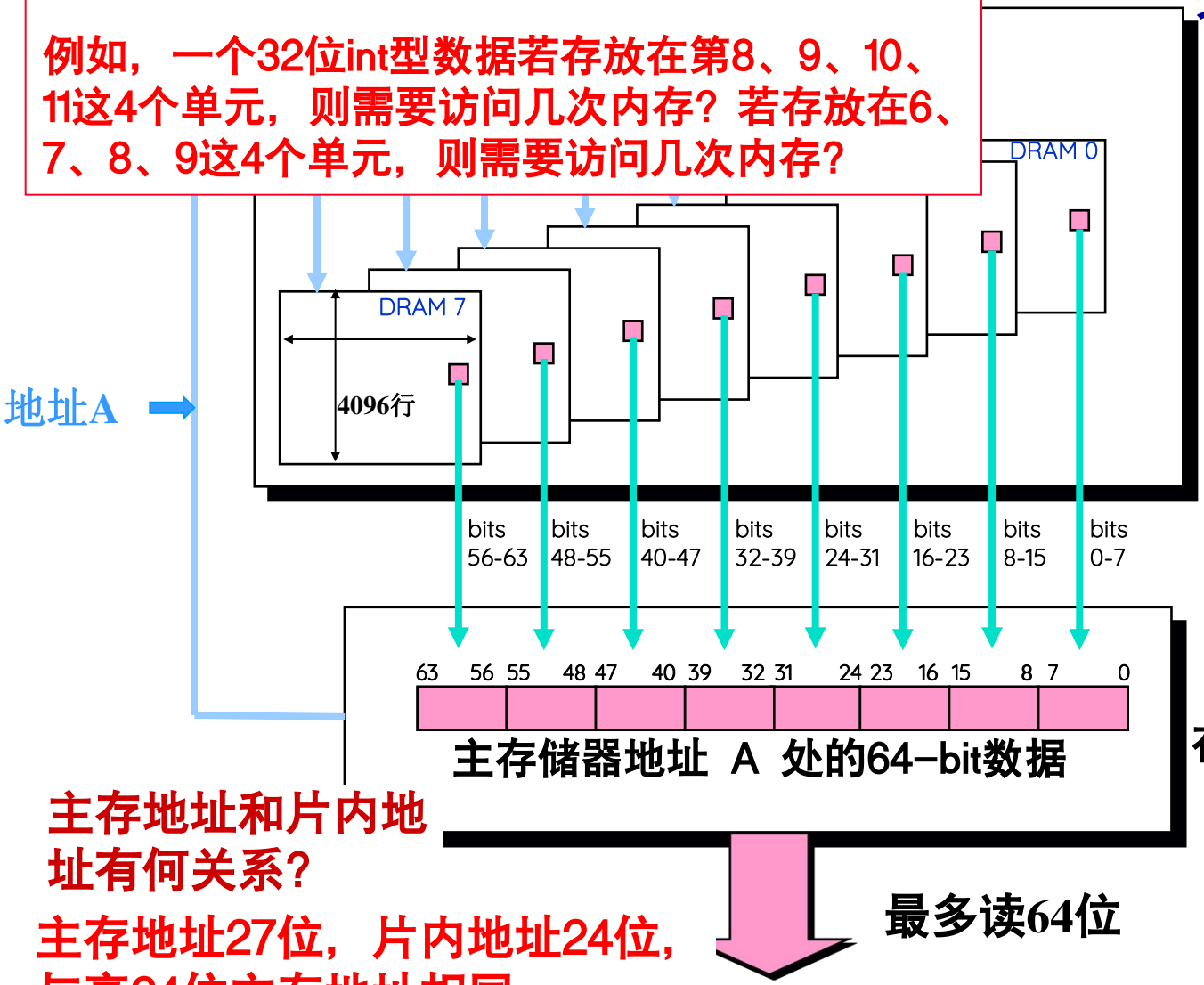
DDR2 SDRAM

DDR3：8位预取，一个时钟内8路交叉读取8个数据到缓冲器中

举例：128MB的DRAM存储器

从该存储器结构可理解为什么规定数据对齐存放。

例如，一个32位int型数据若存放在第8、9、10、11这4个单元，则需要访问几次内存？若存放在6、7、8、9这4个单元，则需要访问几次内存？



分别访问1次和2次

- 由8片DRAM芯片构成
- 每片 16Mx8 bits
- 行地址、列地址各12位
- 每行共4096列(8位/列)
- 选中某一行并读出之后再由列地址选择其中的一列(8个二进位）送出

芯片内地址是否连续？

不连续，交叉编址，可同时读写所有芯片。

□ 行、列地址为(i,j)的8个单元

主存地址和片内地址有何关系？

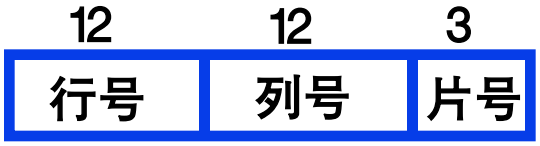
主存地址27位，片内地址24位，与高24位主存地址相同。

主存低3位地址的作用是什么？

确定8个字节中的哪个，即用来选片。

128MB的DRAM存储器

地址A如何划分？　低3位用来选芯片



在DRAM行缓冲中数据的地址有何特点？

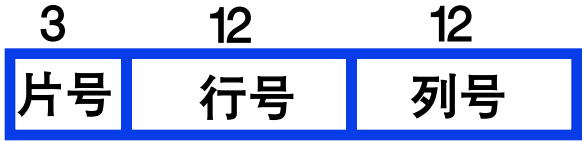
假定第k行首地址为 i（i=32768*k），则地址分布如下：

	Chip0	Chip1	…	Chip7
第0列	i		i+1	
i+7				
第1列	i+8	i+9		i+15
…				
第4095列	i+8*4095	i+1+8*4095		i+7+8*4095

每行地址连续，共8*4096B=2^{15}B=32768个单元

通常，一个主存块包含在行缓冲中
可降低Cache缺失损失

如果芯片内地址连续，
则地址A如何划分？



回顾：Alignment(对齐)

如：int i, short k, double x, char c, short j,……

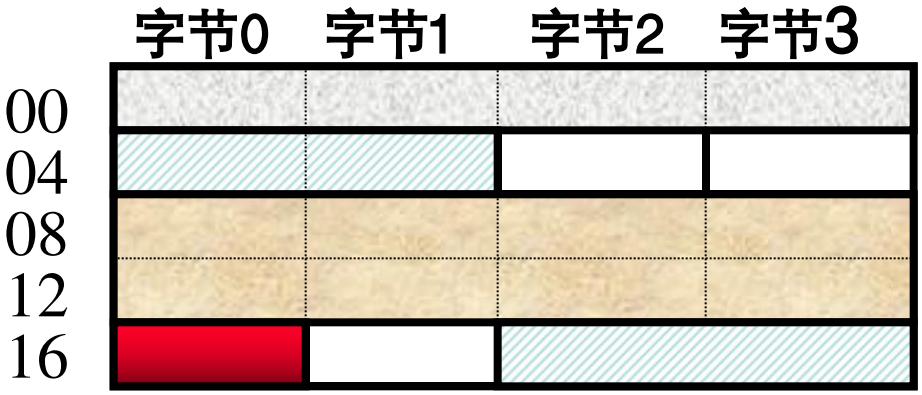
按字节编址

每次只能读写某个字地址开始的4个单元中连续的1个、2个、3个或4个字节

按边界对齐

x：2个周期

j：1个周期

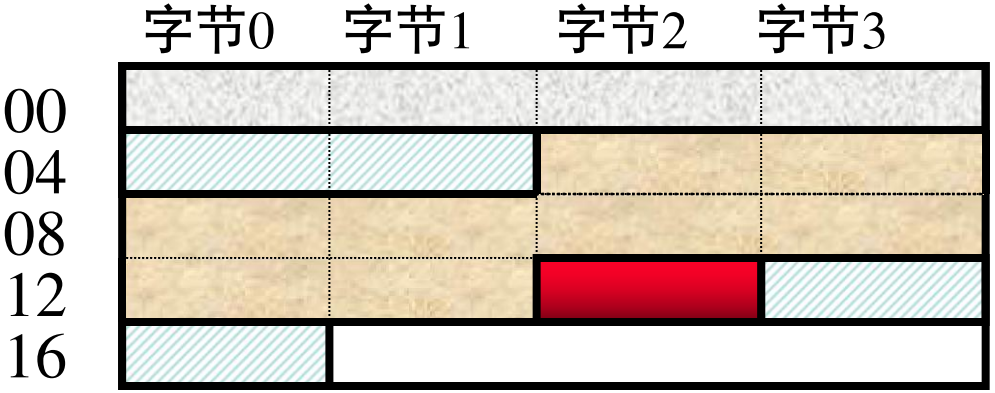


则：&i=0; &k=4; &x=8; &c=16; &j=18;……

边界不对齐

x：3个周期

j：2个周期



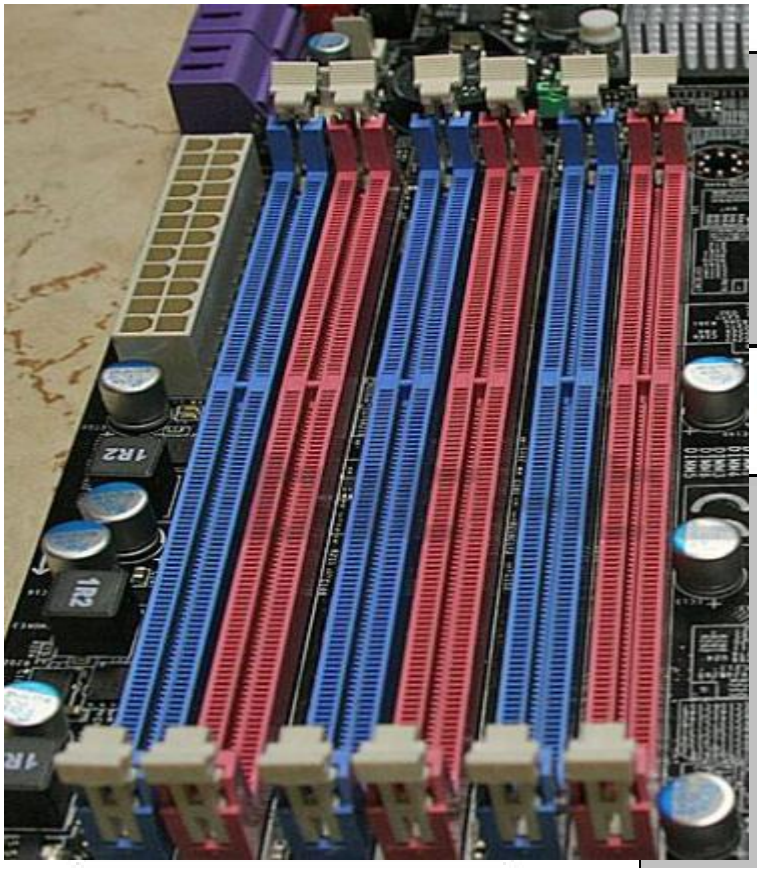
则：&i=0; &k=4; &x=6; &c=14; &j=15;……

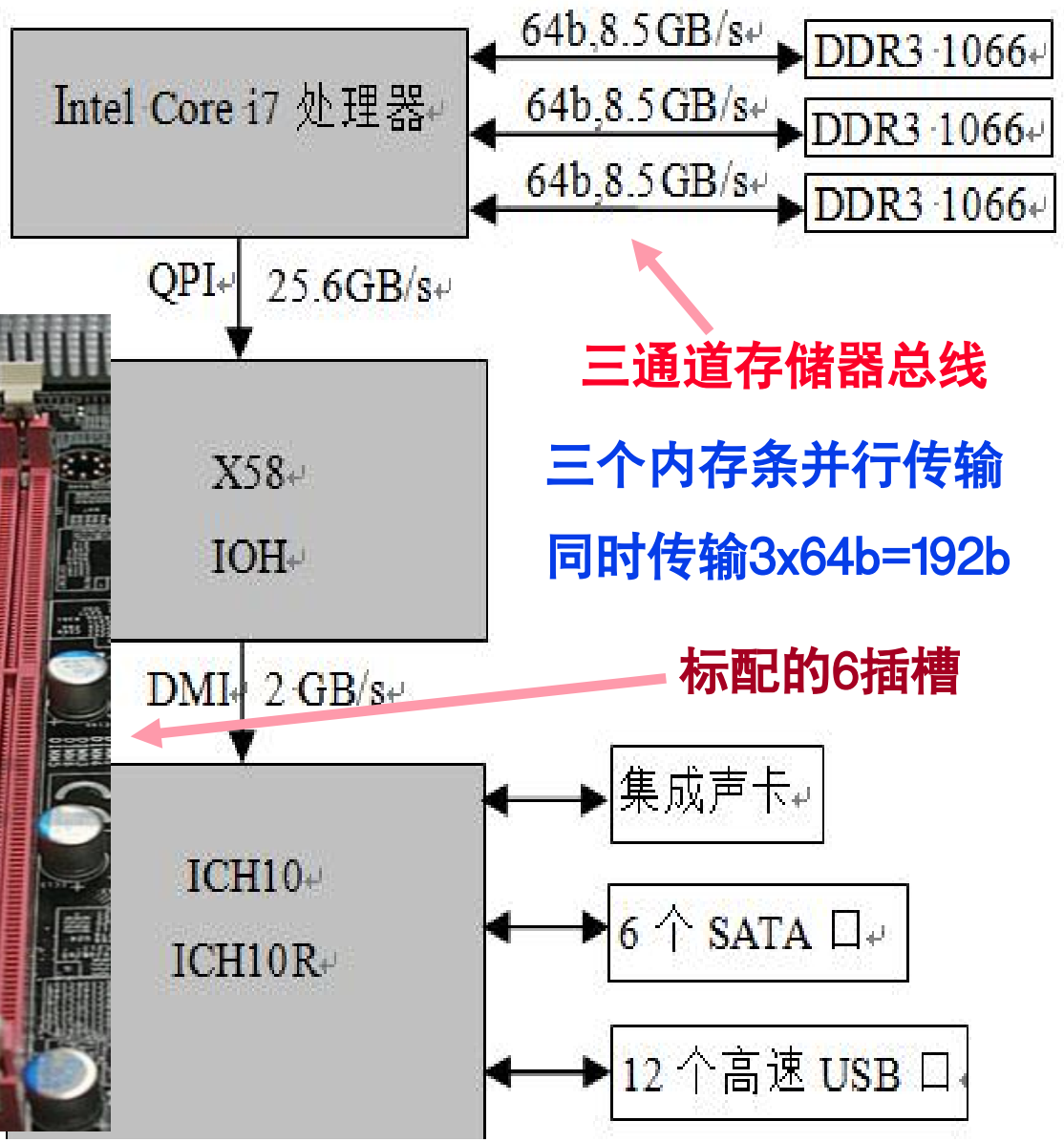
虽节省了空间，但增加了访存次数！

需要权衡，目前来看，浪费一点存储空间没有关系！

计算机系统互连

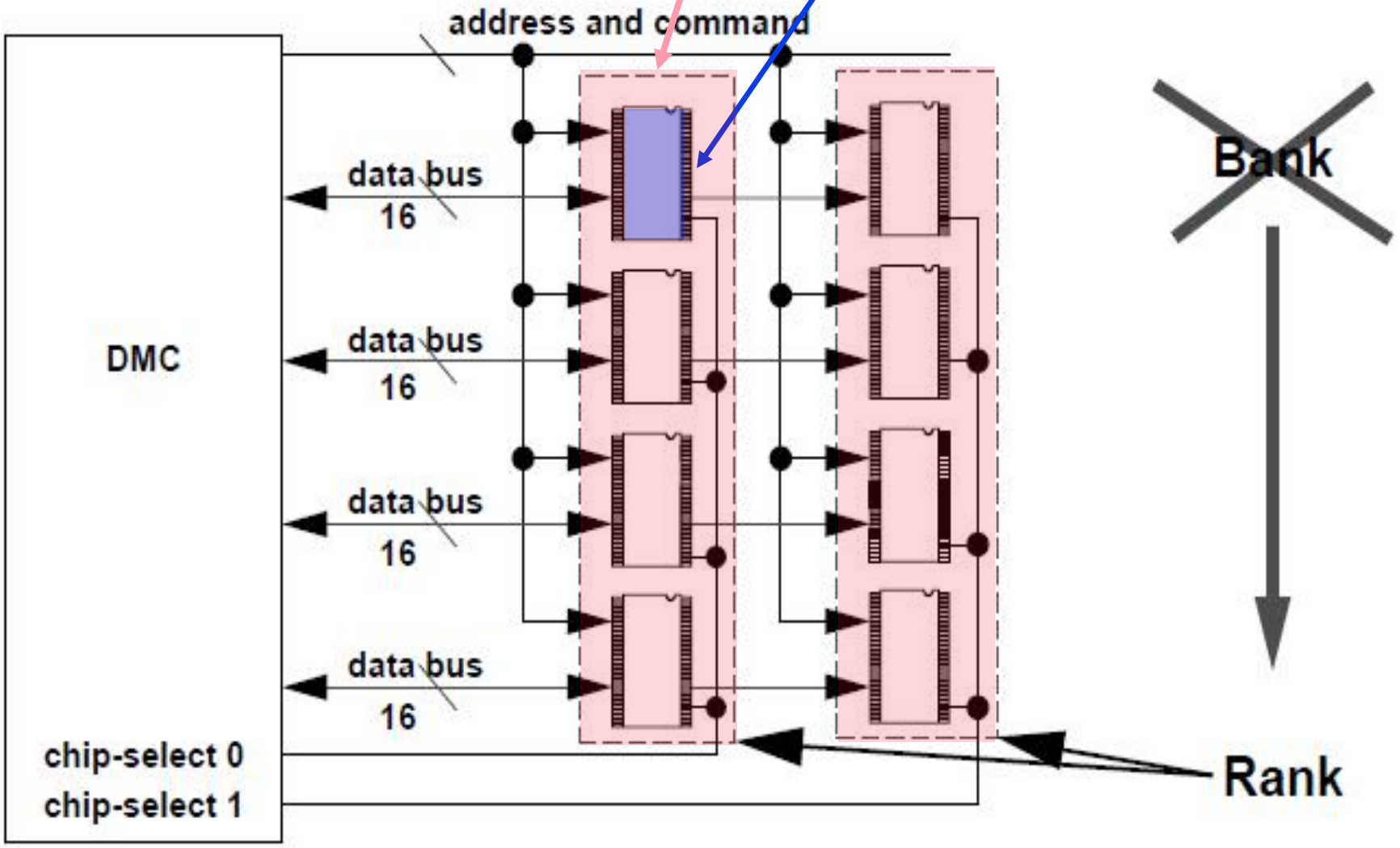
只要将同色的三个内存插槽插上内存条，系统便会自动识别进入三通道模式





DRAM内存条结构

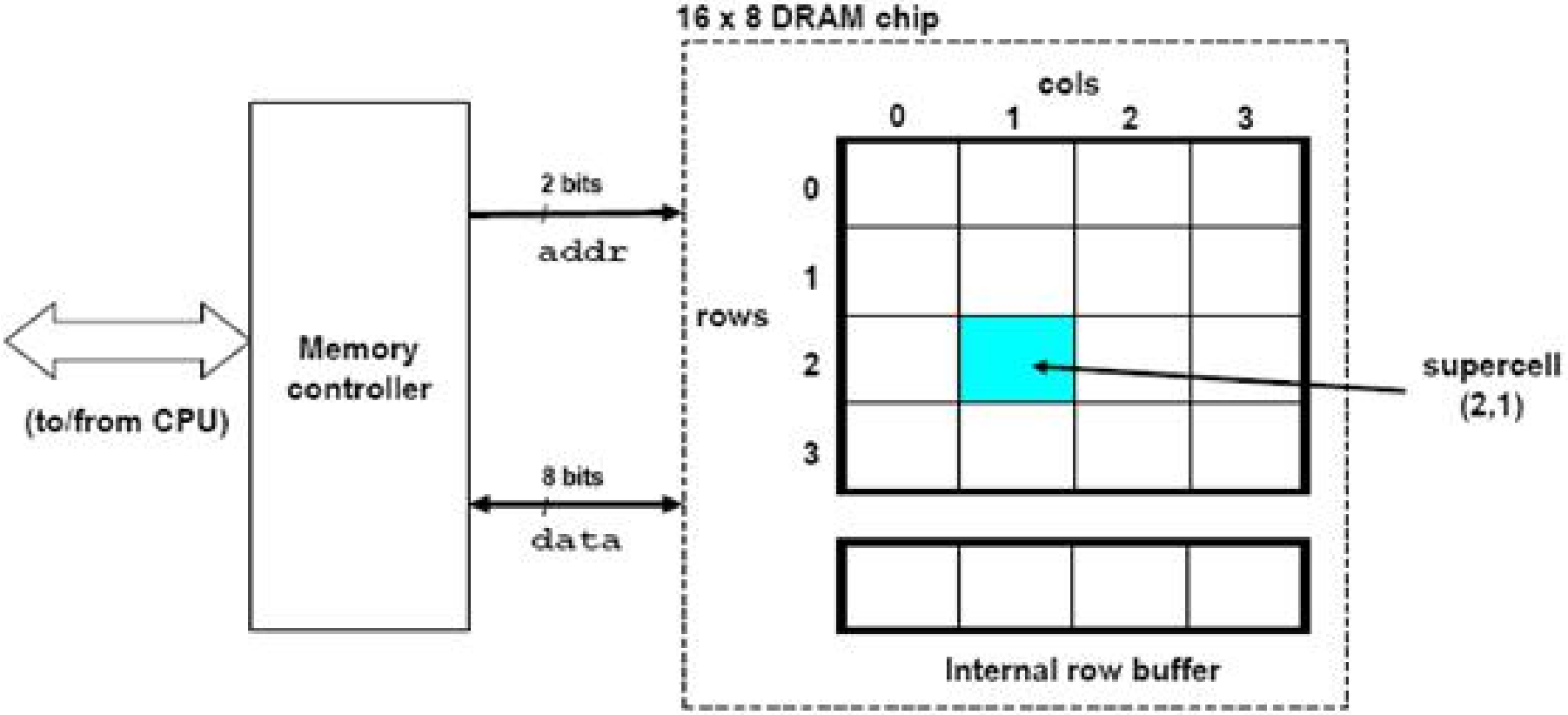
存控给出的地址包括：Channel、Rank、Bank、Row、Coloum



主存模块的连接和读写操作

° DRAM芯片内部结构示意图

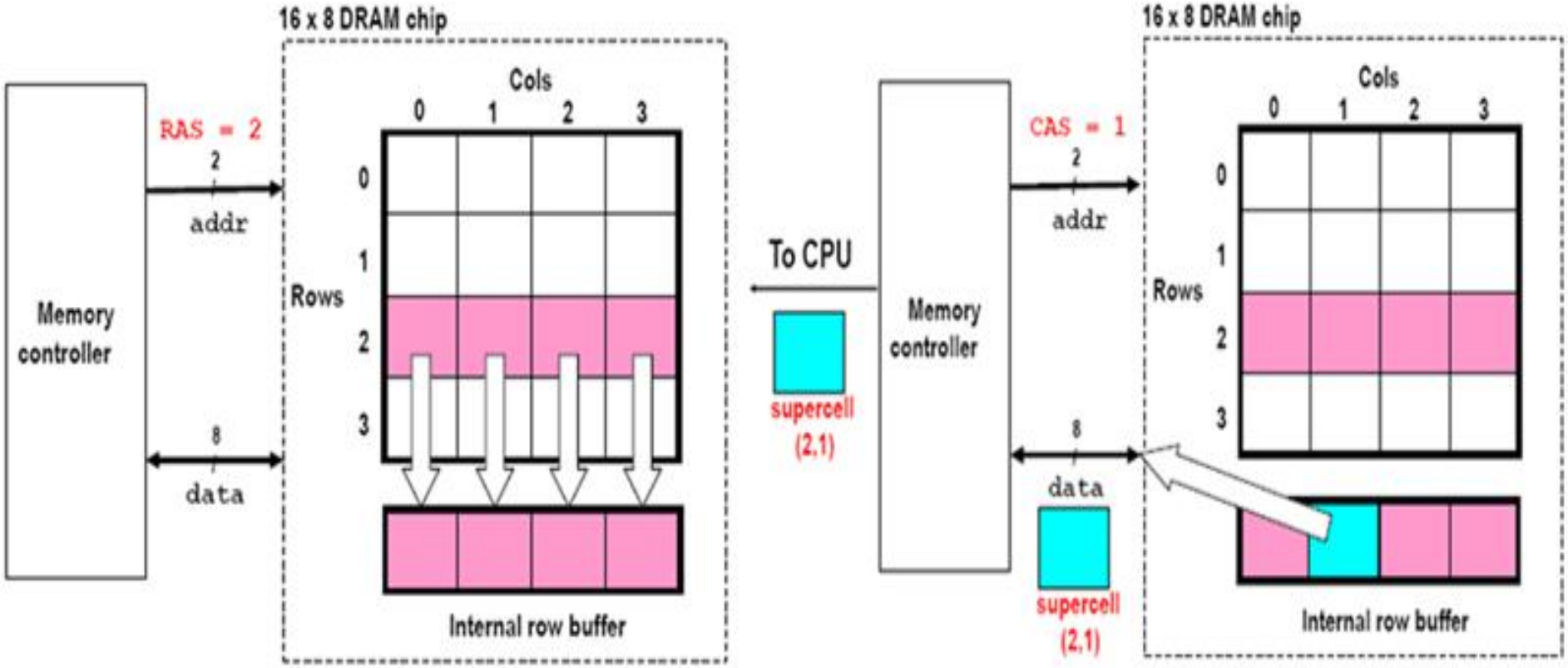
同时有多个芯片进行读写



图中芯片容量为16×8位，存储阵列为4行×4列，地址引脚采用复用方式，因而仅需2根地址引脚，每个超元（supercell）有8位，需8根数据引脚，有一个内部的行缓冲（row buffer），通常用SRAM元件实现。

主存模块的连接和读写操作

° DRAM芯片读写原理示意图



首先，主存控制器将行地址“2”送行译码器，选中第“2”行，此时，整个一行数据被送行缓冲。然后，主存控制器将列地址“1”送列译码器，选中第“1”列，此时，将行缓冲第“1”列的8位数据supercell(2,1)读到数据线，并继续送往CPU。

主存控制器

- 也称DRAM控制器或存储器控制器，一侧连接系统总线，接收来自CPU的访存请求，另一侧连接存储器总线，向主存芯片发送命令进行读写和刷新
- 主要工作包括以下几个方面：

  ① 事务调度。主存控制器可能会通过系统总线收到来自多个CPU甚至是外设的访存请求，通常根据优先级决定先处理哪些请求。

  ② 地址转换。需要根据存储器芯片的组织结构将物理地址划分成对应的行地址、列地址等字段，并根据地址高位生成片选信号，用于控制采用字扩展方案组织的多个存储器芯片。

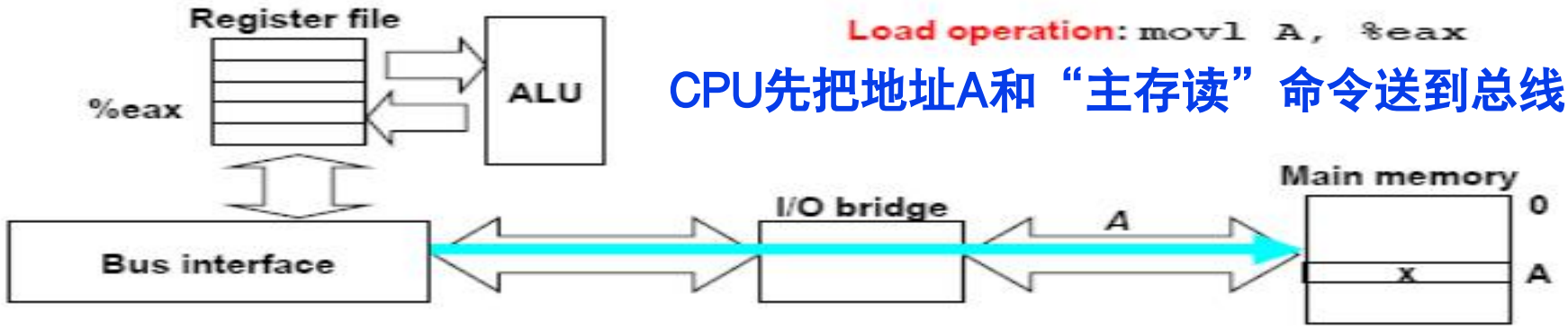
  ③ 命令调度。将系统总线中控制线的信号转换为发往存储器芯片的命令序列，并将其放入命令队列中。

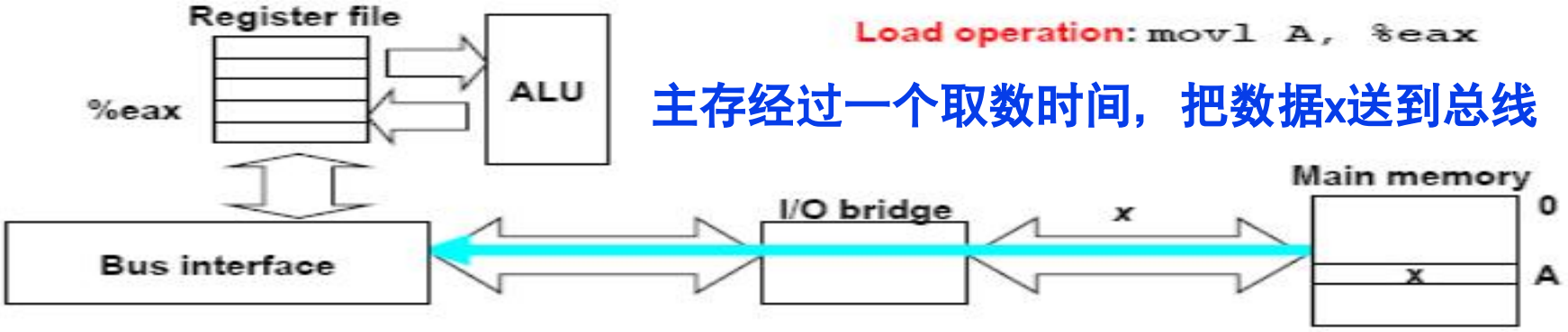
  ④ 访问存储器芯片。根据存储器总线协议，将命令队列中的命令转换为相应的信号，并将这些信号通过存储器总线输出到存储器芯片引脚

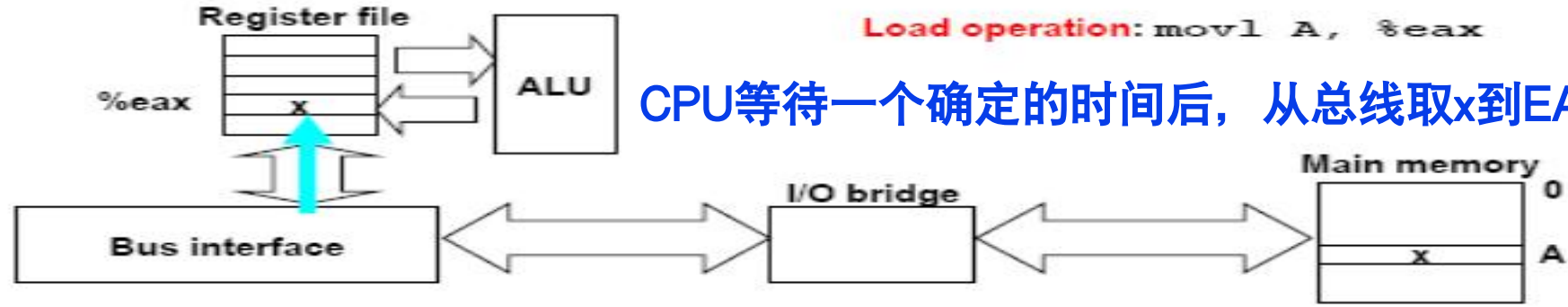
  ⑤ 定时刷新。计算刷新间隔并据此维护一个刷新计数器，定时向存储器芯片发送刷新命令。

指令“movl 8(%ebp), %eax”操作过程

得到地址A的过程较复杂，涉及MMU、TLB、页表等重要概念！

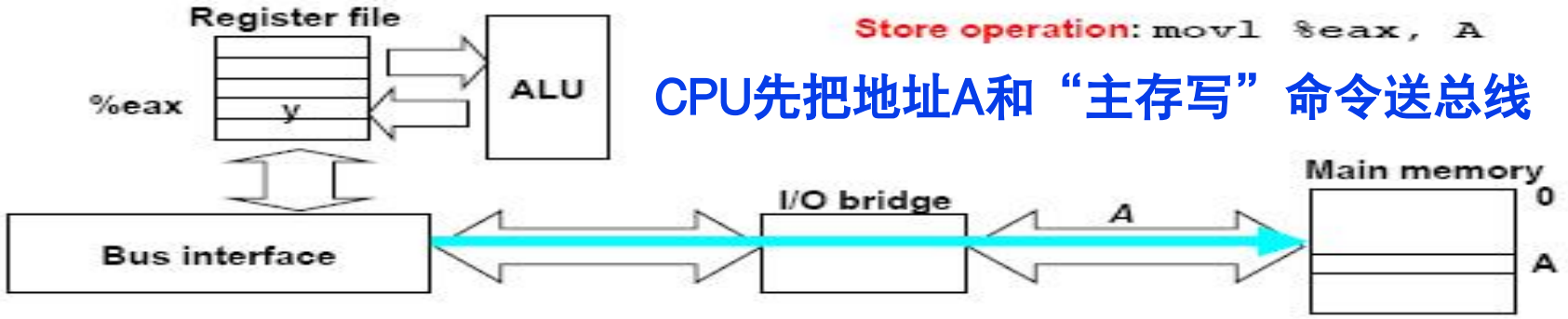






指令“movl %eax,8(%ebp)”操作过程

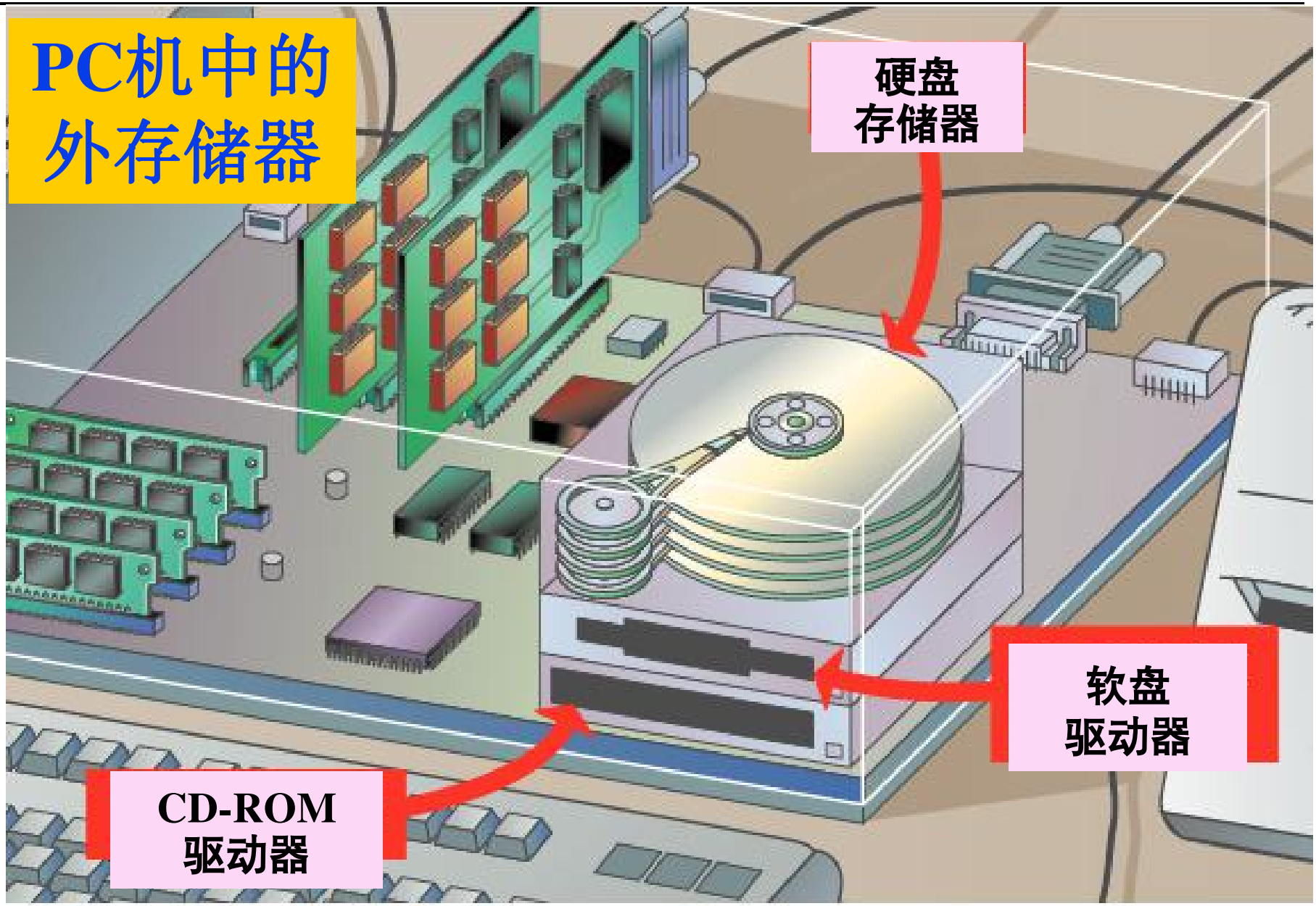
得到地址A的过程较复杂，涉及MMU、TLB、页表等重要概念！



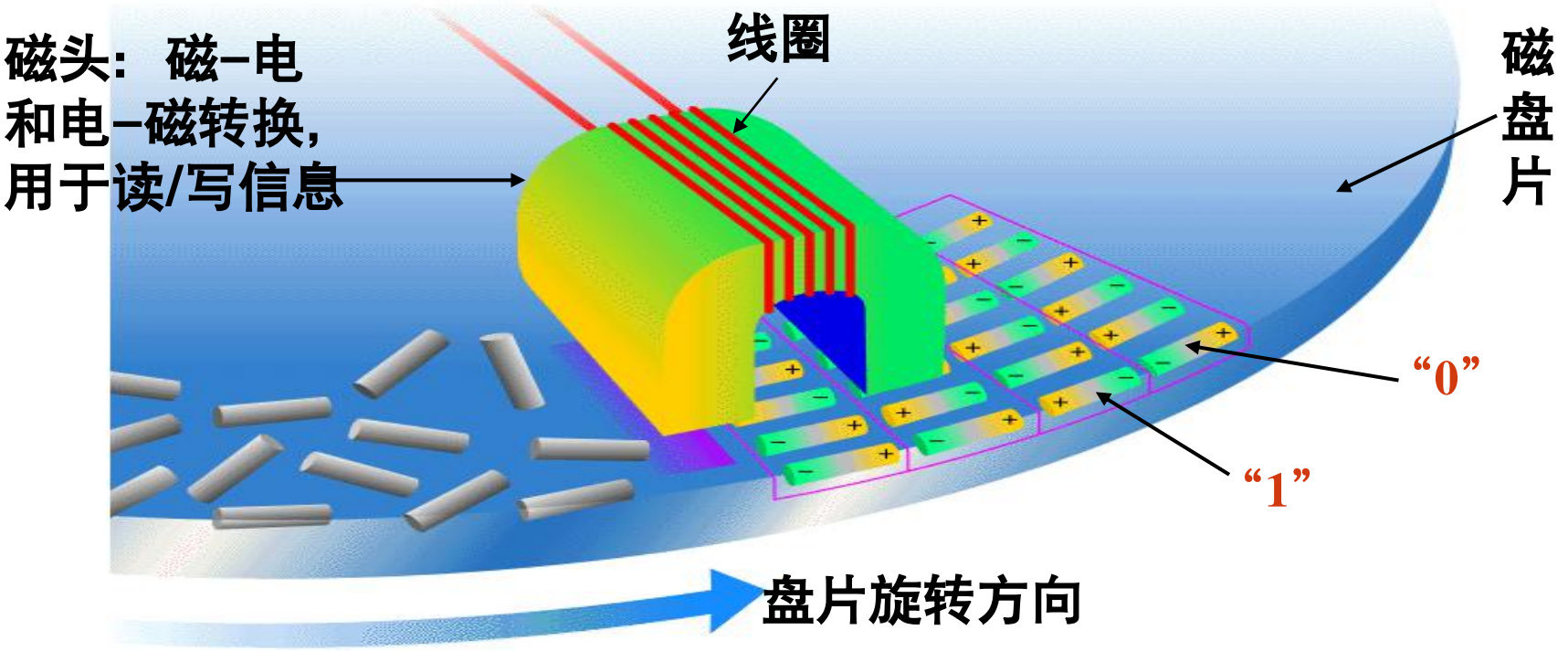
层次结构存储系统

- 分以下四个部分介绍
  - 第一讲：存储器概述
  - 第二讲：半导体随机存取存储器
    - 基本存储元件、DRAM芯片、 SDRAM芯片技术
    - 内存条及其与CPU的连接
    - 存储器芯片的扩展、主存控制器
  - 第三讲：外部存储器
    - 磁盘存储器、闪速存储器、U盘、固态硬盘
  - 第四讲：高速缓冲存储器(cache)
    - cache的基本工作原理
    - 映射方式、替换算法、写策略
    - Cache的设计
    - Cache和程序性能

PC中的外存储器



磁盘存储器的信息存储原理



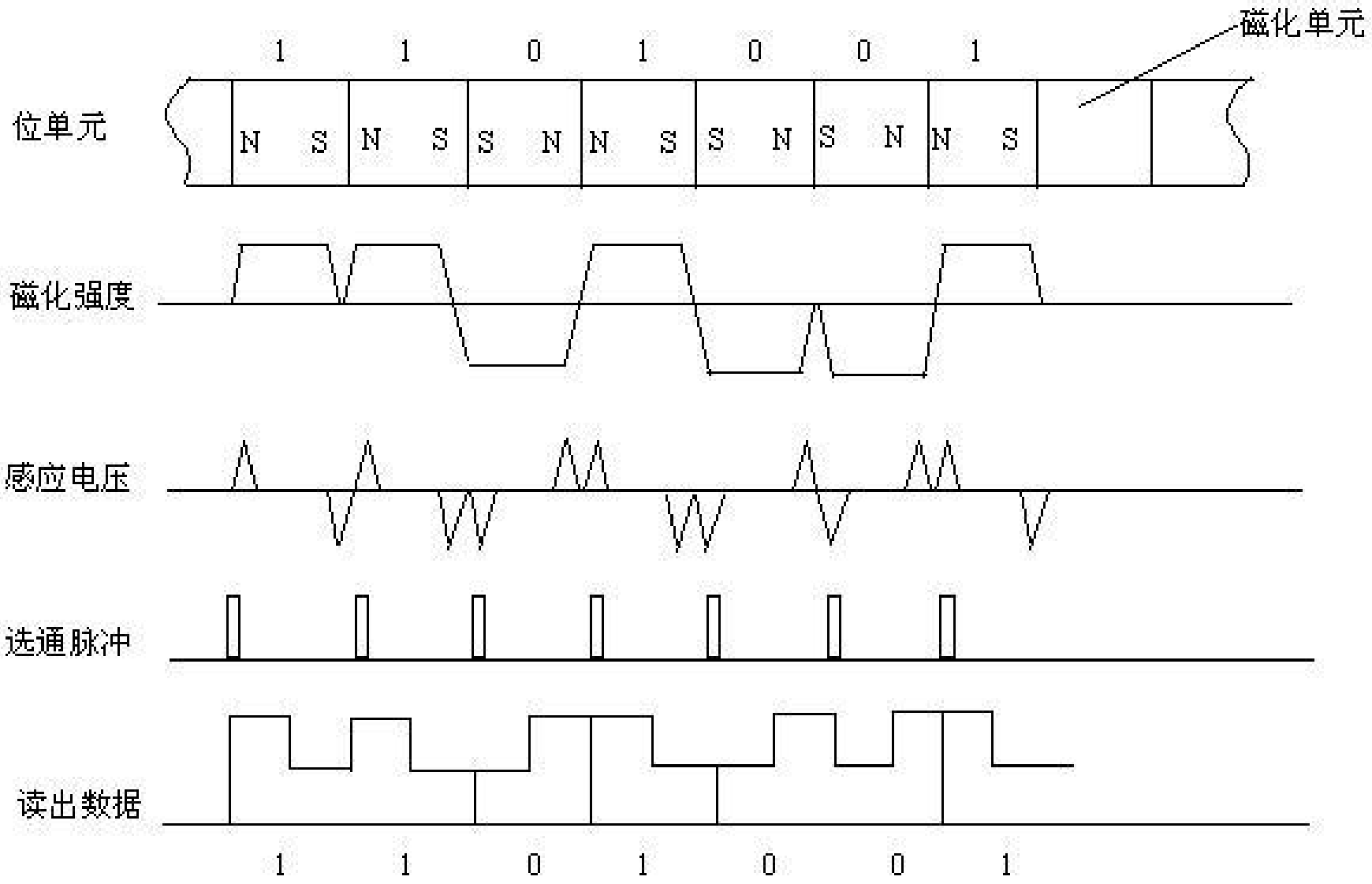
写1：线圈通以正向电流，使呈N–S状态

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

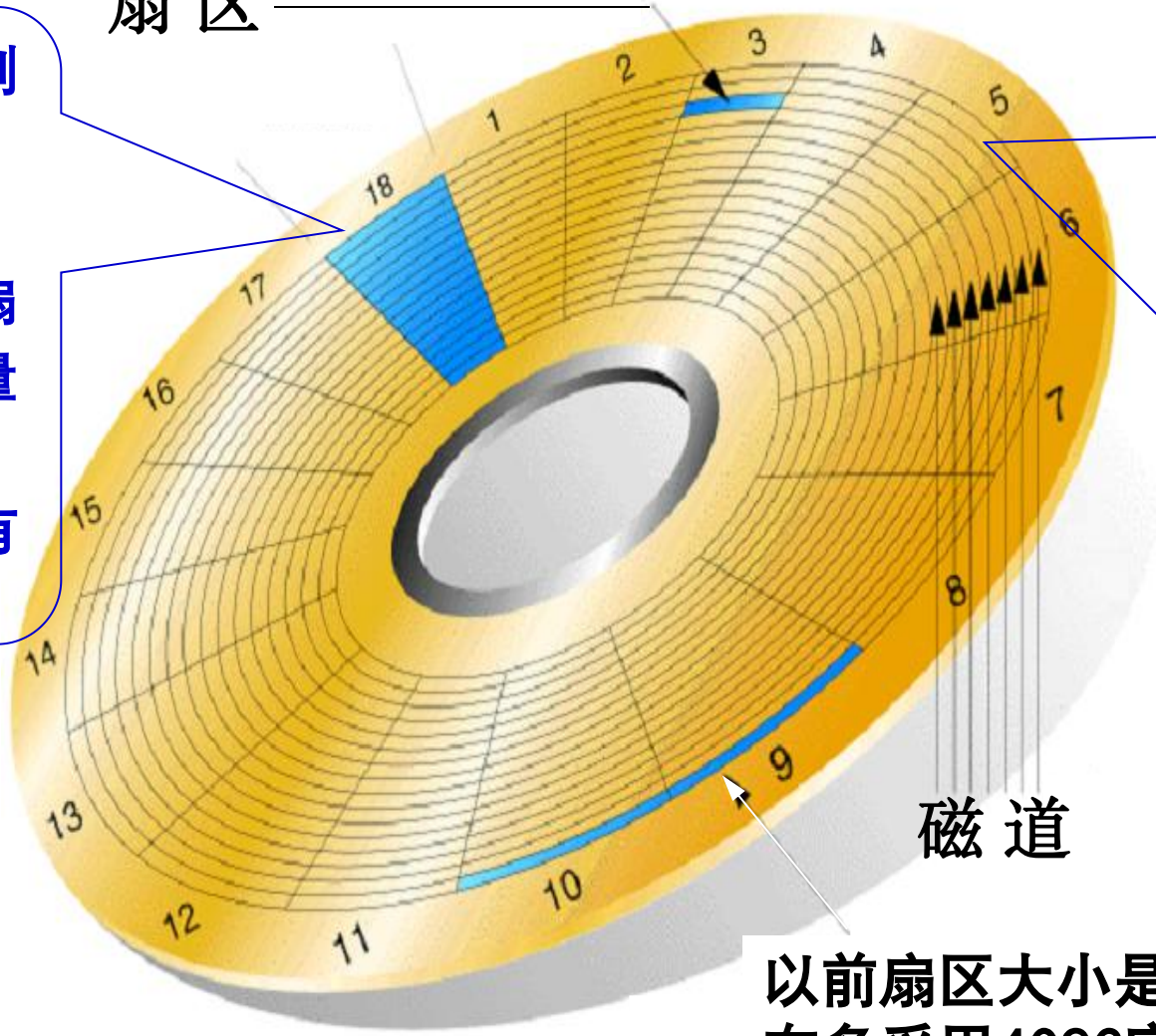
读时：磁头固定不动，载体运动。因为载体上小的磁化单元外部的磁力线通过磁头铁芯形成闭合回路，在铁芯线圈两端得到感应电压。根据感应电压的不同的极性，可确定读出为0或1。

磁表面信息读出过程


磁化单元
1 1 0 1 0 0 1
位单元
N S N S S N N S S N S N N S
磁化强度
感应电压
选通脉冲
读出数据
1 1 0 1 0 0 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

以前扇区大小是512字节。现在多采用4096字节扇区，通常称为4K扇区。

磁盘磁道的格式

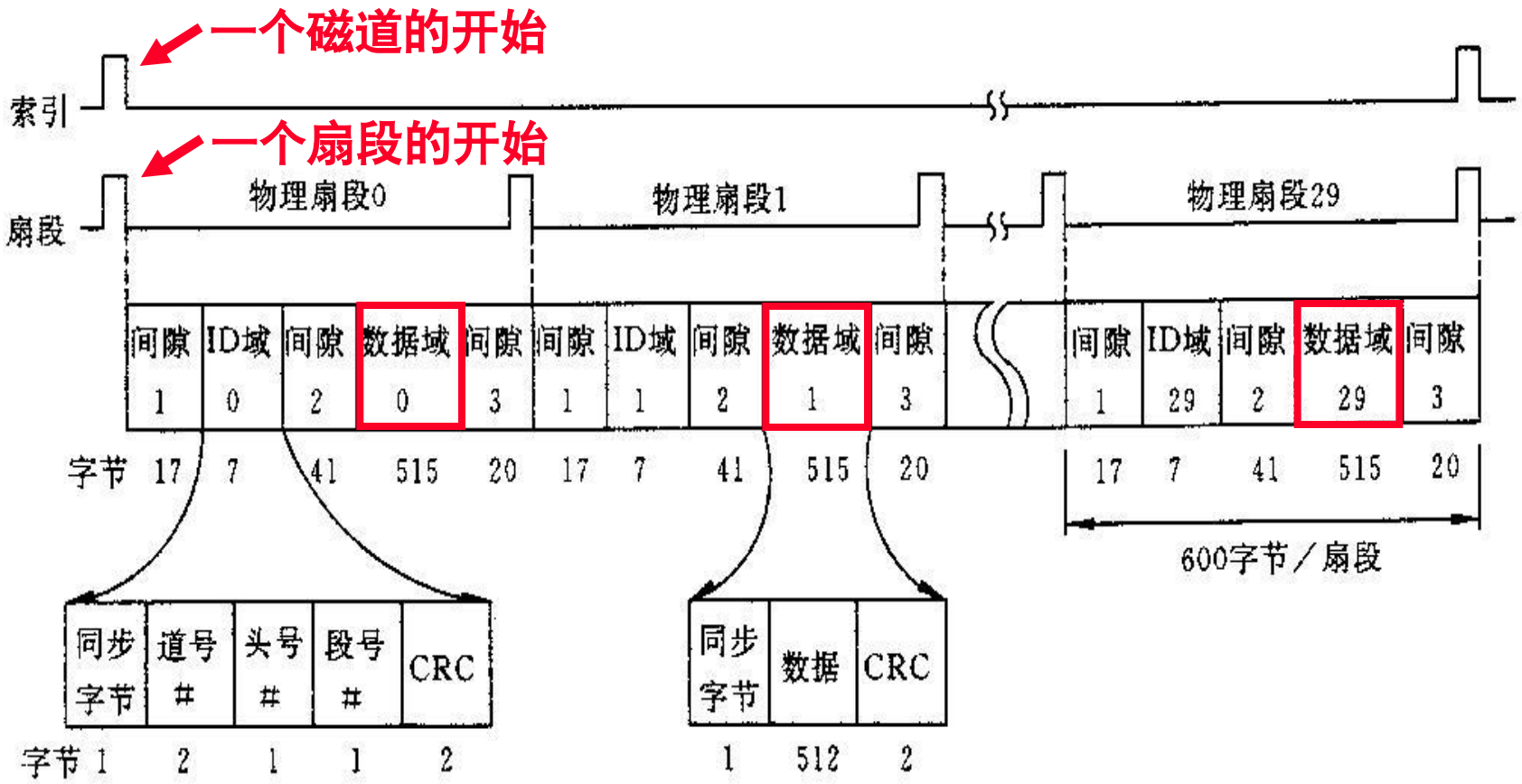
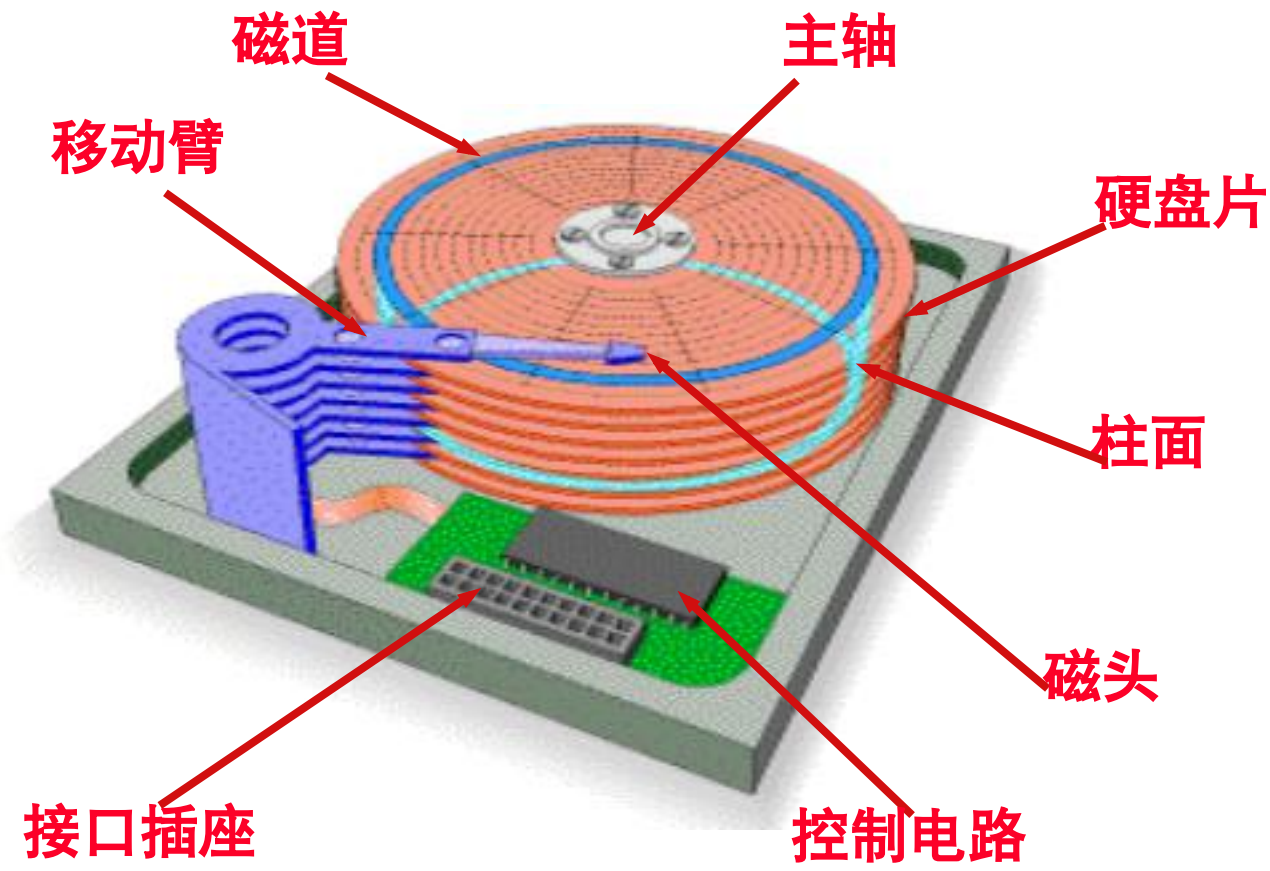


图 5.2 温彻斯特磁盘磁道格式(Seagate ST506)

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

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)。

磁盘驱动器

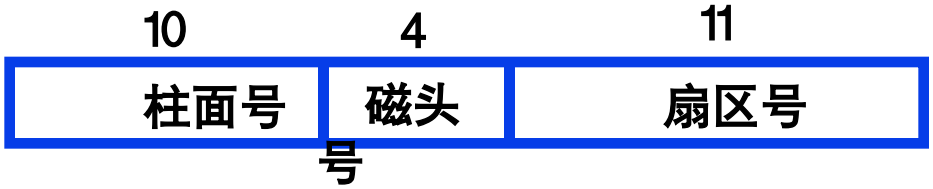


假定有5片、1000个同心圆，每个圆分2000个扇区

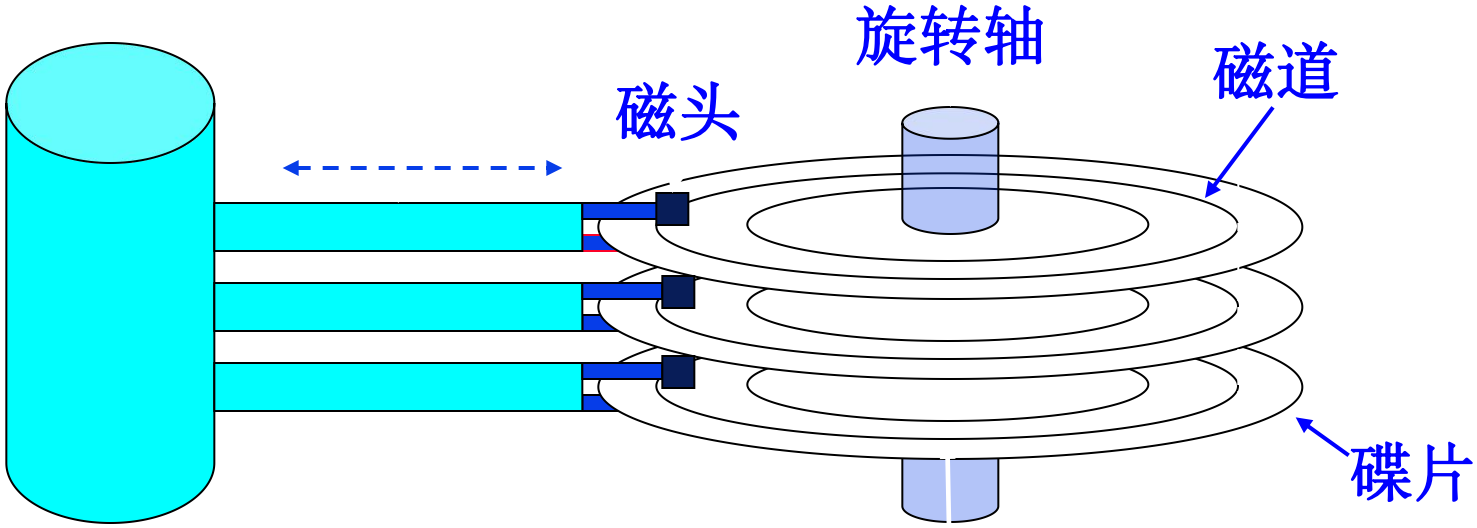
磁盘空间划分为1000个柱面、每柱面10个磁头，每个磁头形成一个磁道，每个磁道2000个扇区，每扇区512B

磁道号就是柱面号、磁头号就是盘面号，每片有两个面，每面一个磁头

磁盘数据地址：



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（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
    （约4～6ms）（ 转速： 4200 / 5400 / 7200 / 10000rpm ）
  - 数据传输时间——（ 大约0.01ms / 扇区 ）

磁盘响应时间计算举例

- 假定每个扇区512字节， 磁盘转速为5400 RPM，声称寻道时间（最大寻道时间的一半）为12 ms，数据传输率为4 MB/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？

Disk Response Time= Queuing Delay + Controller Time + Seek time + Rotational Latency + Transfer time

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

= 0 + 1 ms + 12 ms + 0.5 / 90 RPS + 0.128 / 1000 s

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

= 18.6 ms

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%！

1+12/3+5.5+0.1=10.6ms

为什么实际的寻道时间可能只有1/3？ 磁盘转速非常重要！

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！

每道有多少扇区？ （4MBx60/5400）/512B ≈ 87个扇区

硬盘存储器的组成

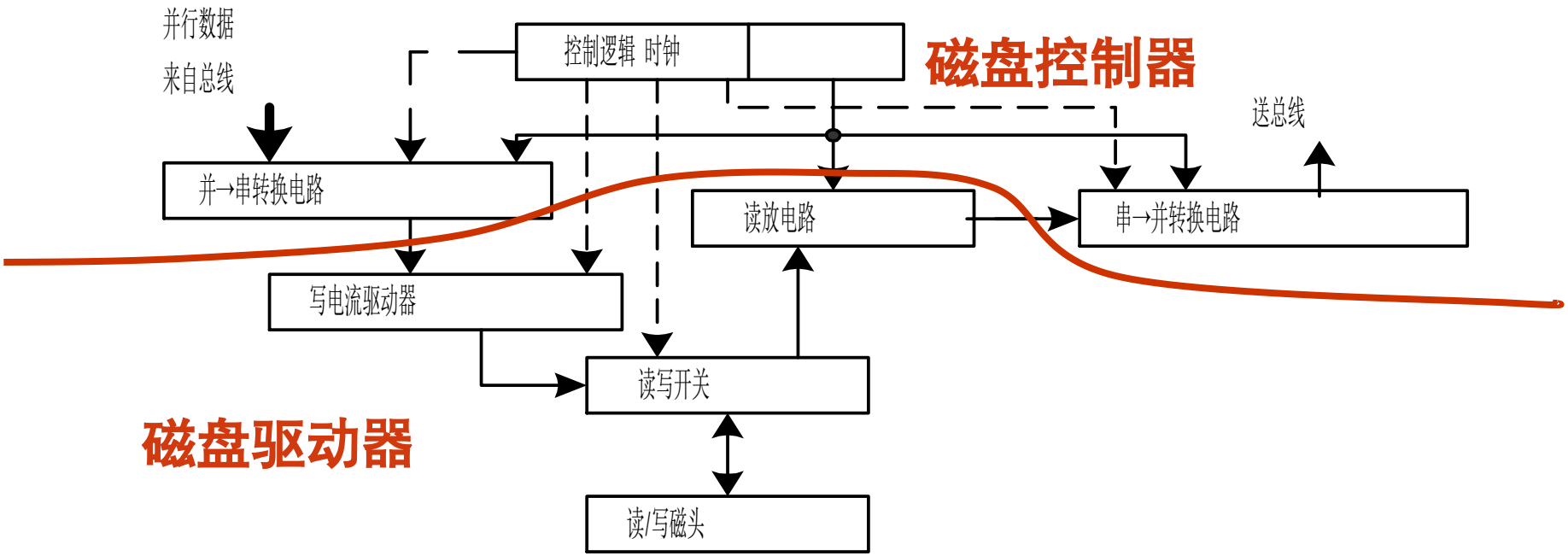
- 硬盘存储器的基本组成

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、磁头与磁头定位伺服系统等

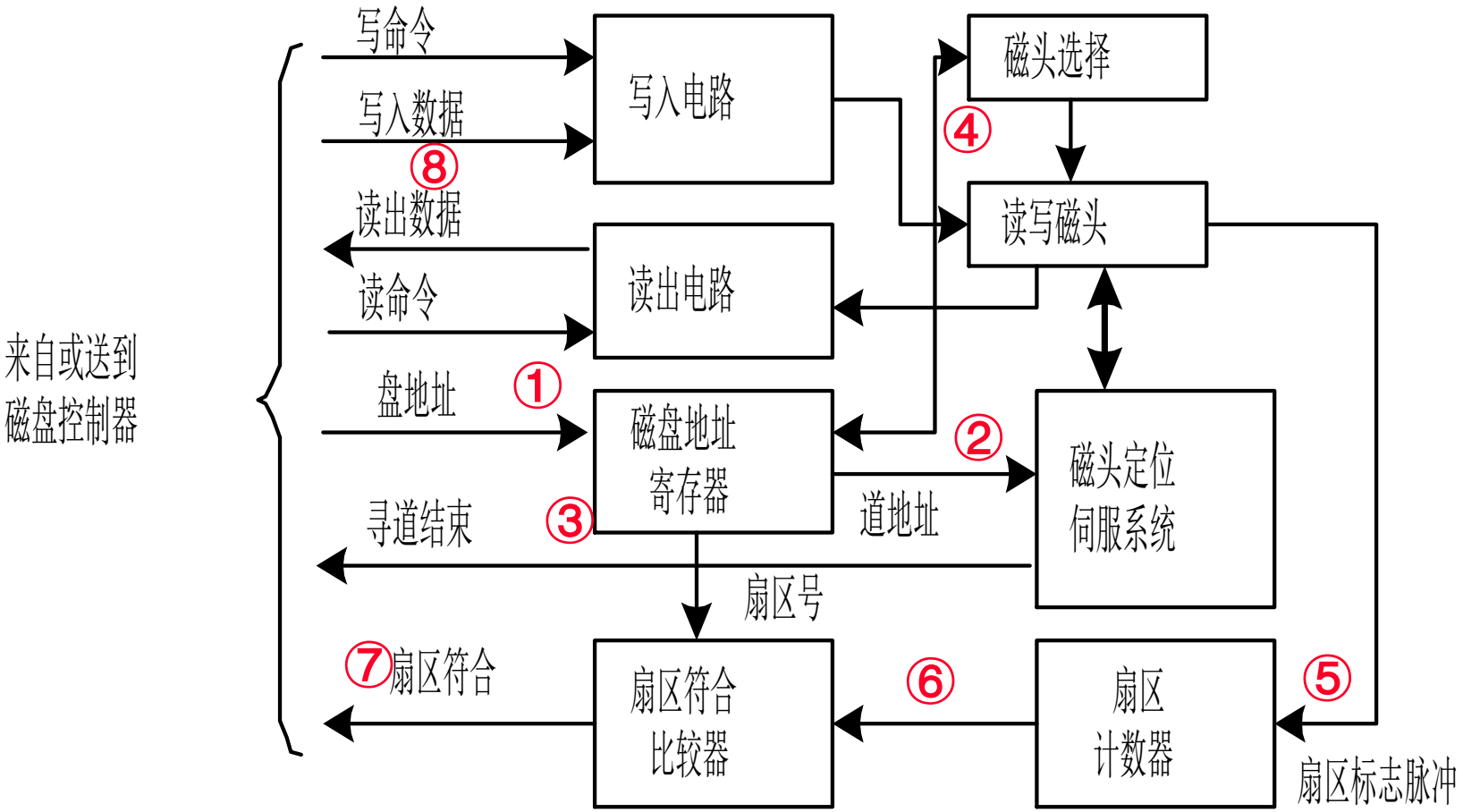
磁盘控制器： 包括控制逻辑、时序电路、“并→串”转换和“串→并”转换电路等。（用于连接主机与盘驱动器）

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）？

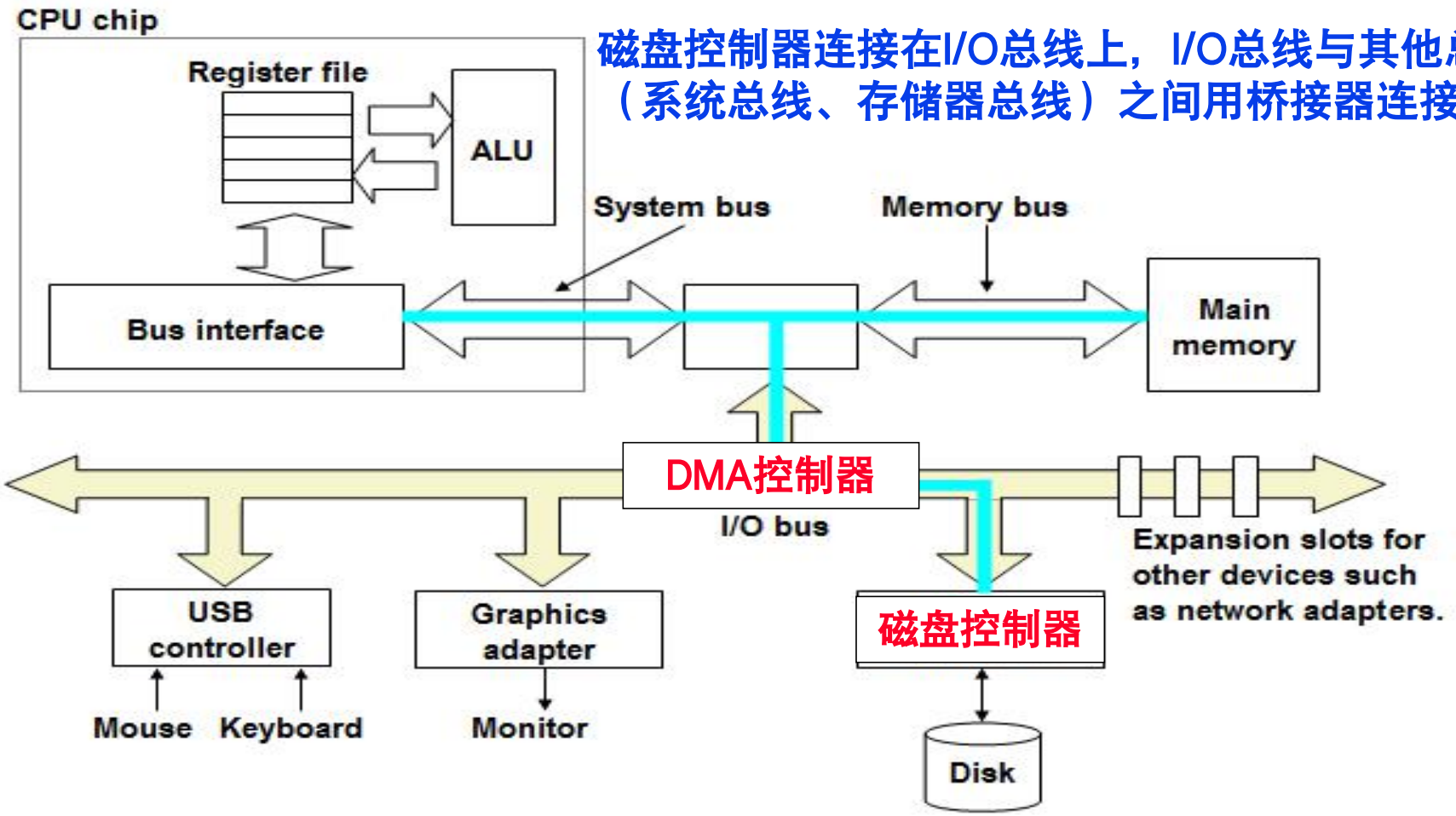
柱面(磁道)号、磁头（盘面）号、扇区号

操作过程？

寻道、旋转、读/写

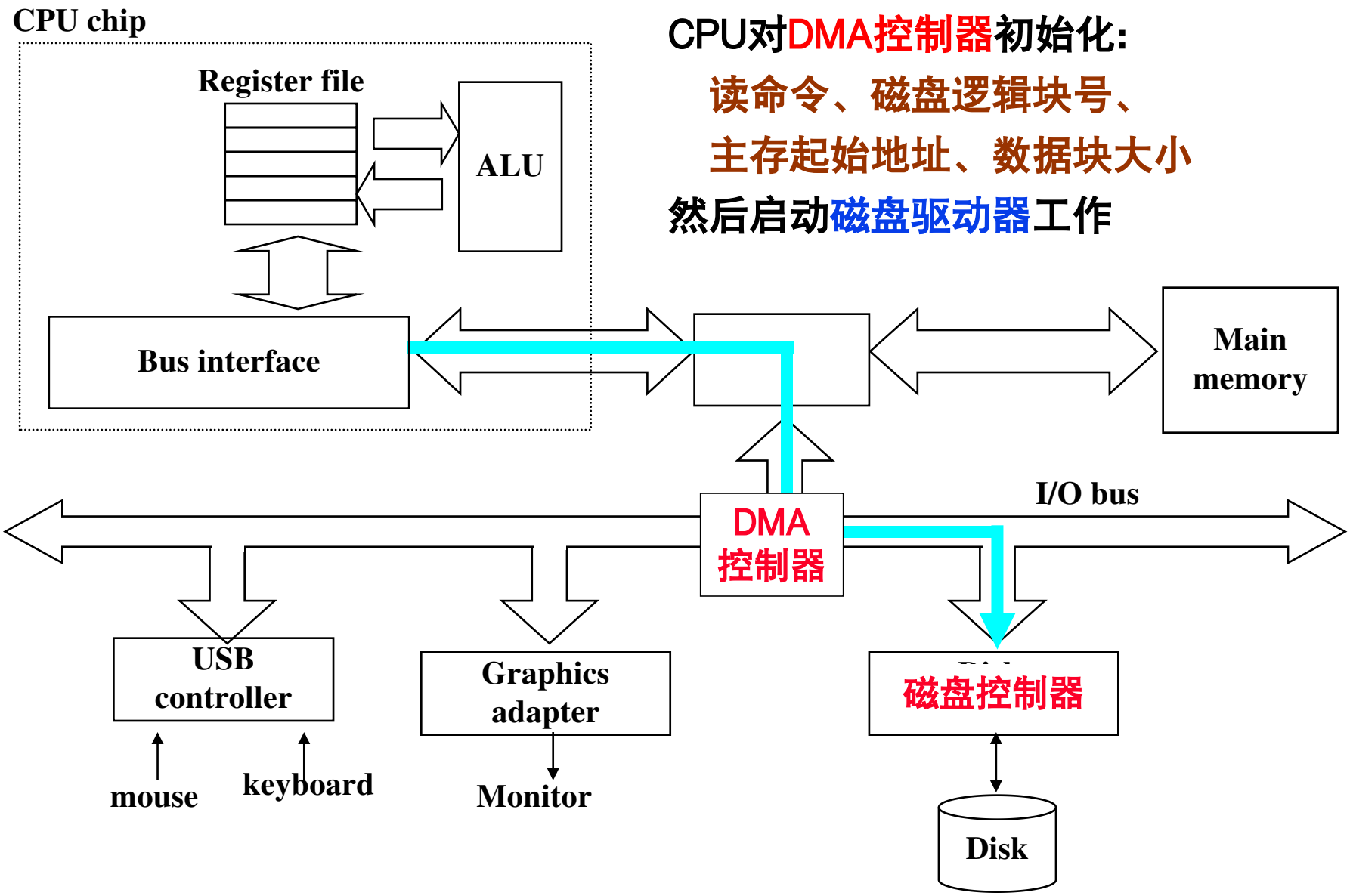
磁盘存储器的连接

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



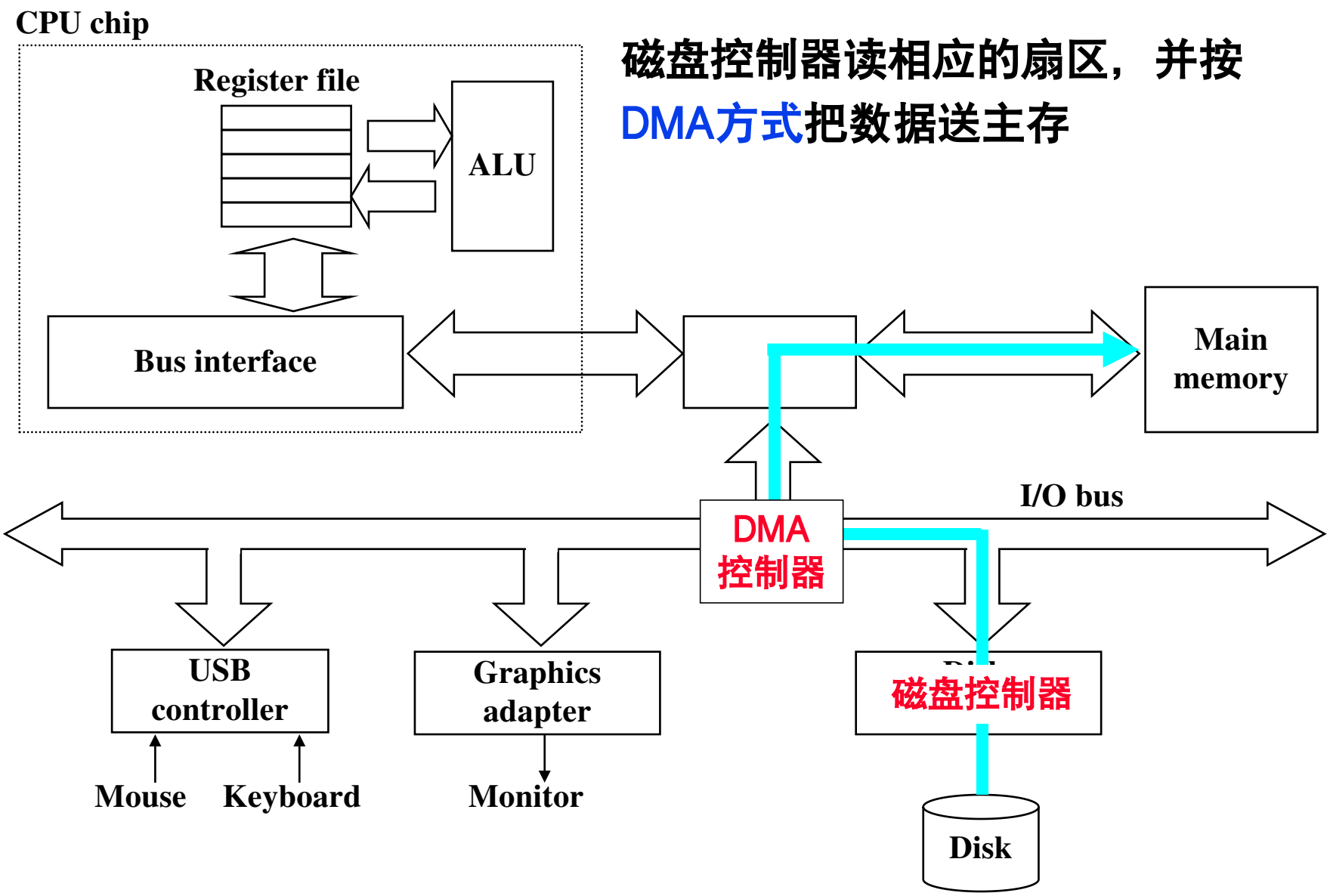
磁盘的最小读写单位是扇区，因此，磁盘按成批数据交换方式进行读写，采用直接存储器存取（DMA，Direct Memory Access）方式进行数据输入输出，需用专门的DMA接口来控制外设与主存间直接数据交换，数据不通过CPU。通常把专门用来控制总线进行DMA传送的接口硬件称为DMA控制器

读一个磁盘扇区 – 第一步


CPU chip
Register file
ALU
Bus interface
CPU对DMA控制器初始化:
读命令、磁盘逻辑块号、
主存起始地址、数据块大小
然后启动磁盘驱动器工作
Main memory
I/O bus
DMA 控制器
USB controller
Graphics adapter
磁盘控制器
mouse
keyboard
Monitor
Disk

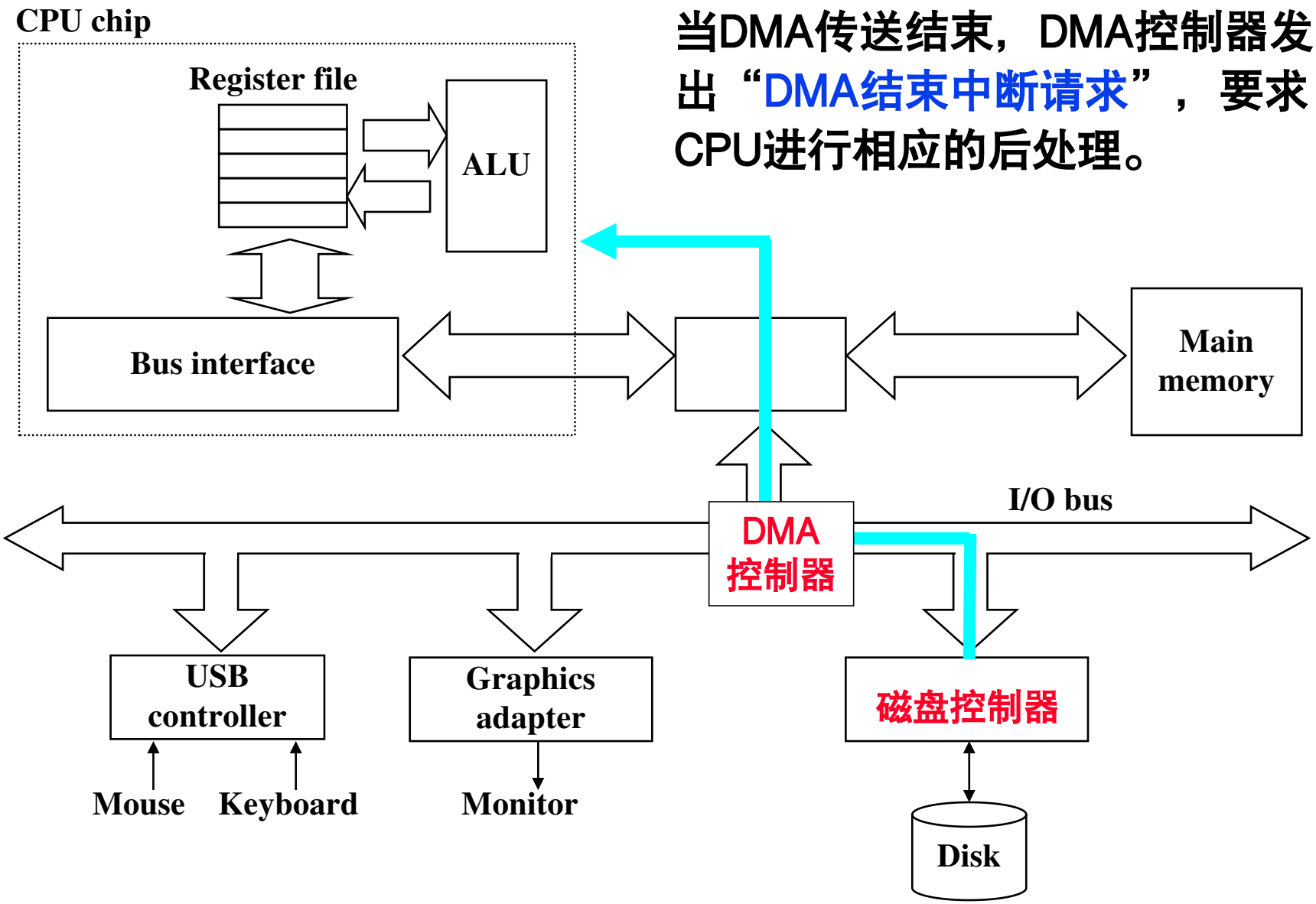
读一个磁盘扇区 – 第二步

磁盘控制器读相应的扇区，并按DMA方式把数据送主存



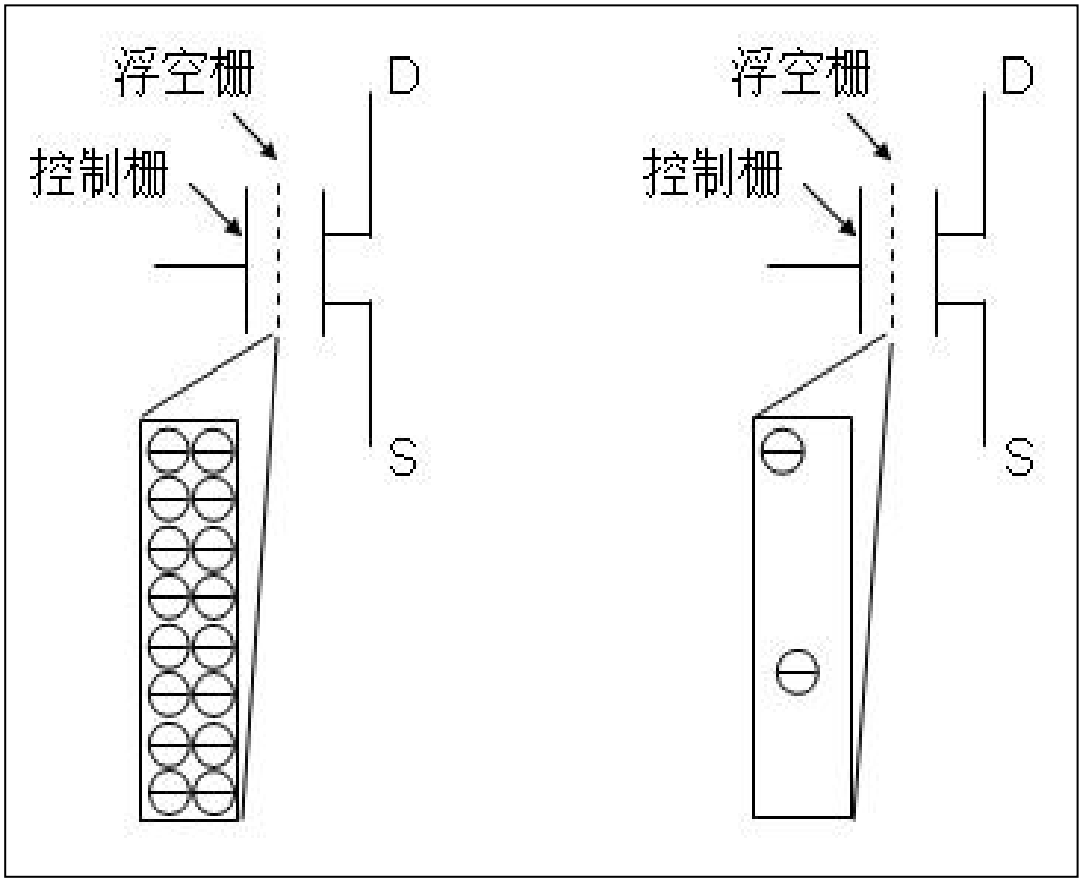
读一个磁盘扇区 – 第三步

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



闪存（Flash Memory）

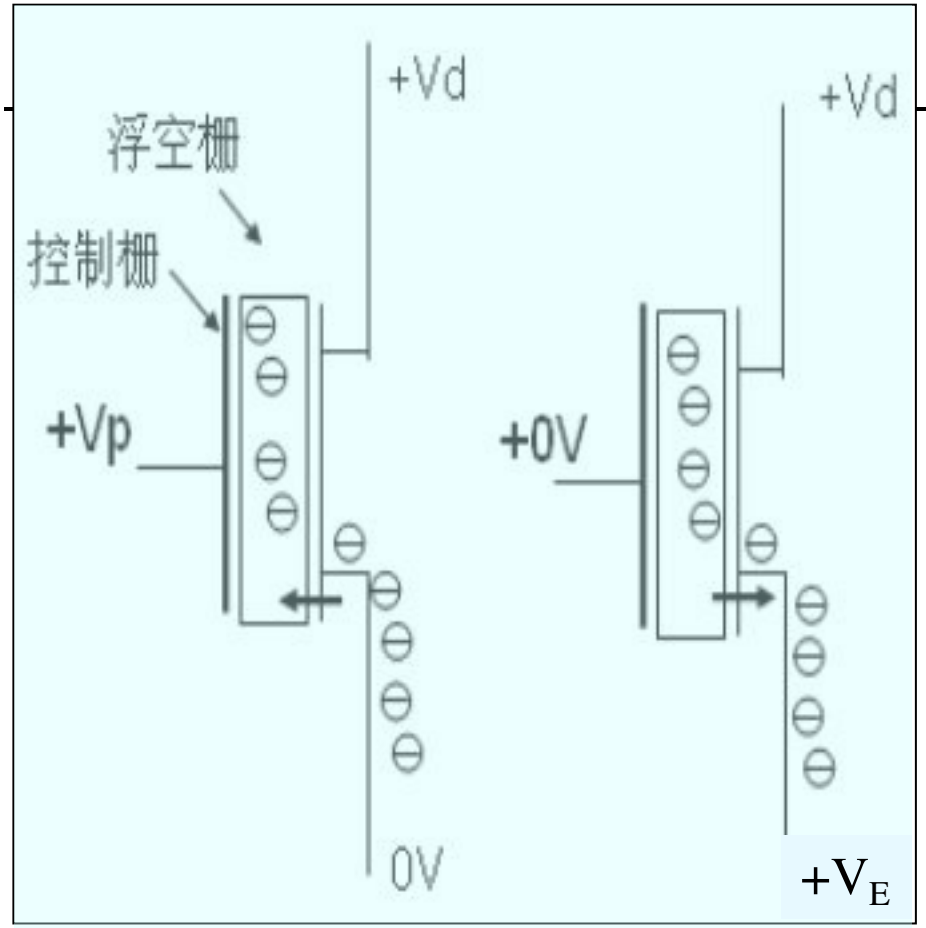
Flash 存储元：

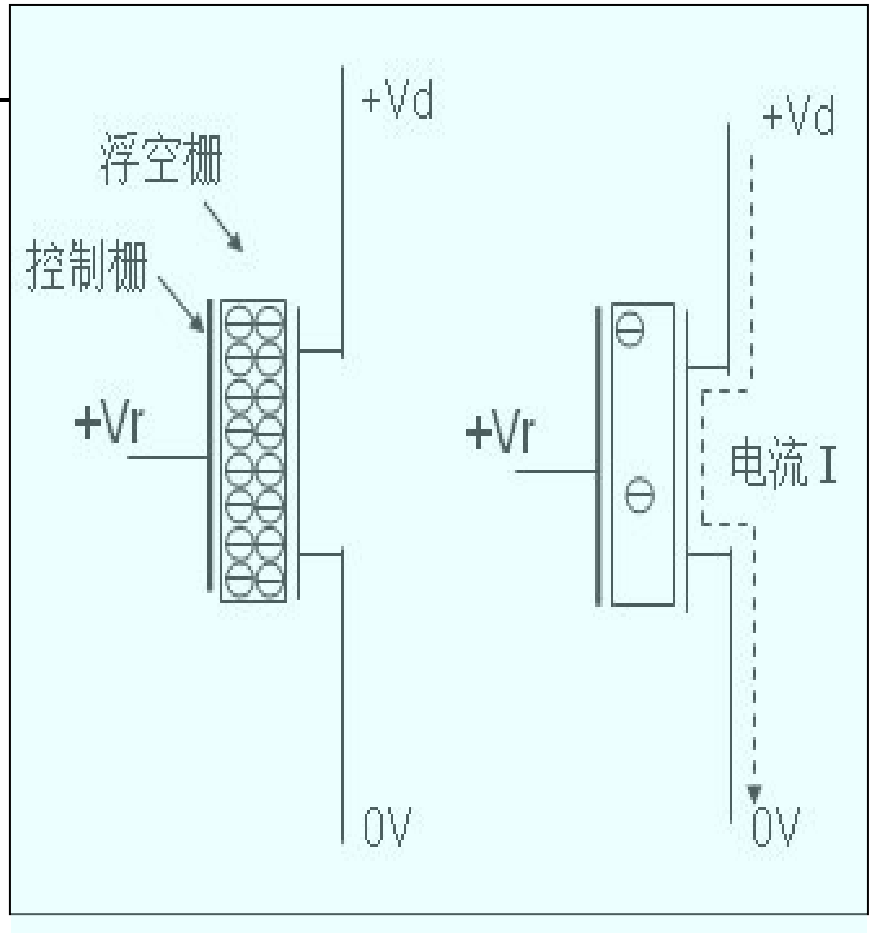


(a)“0”状态　　(b) “1”状态

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。





(a) 编程:写“0”　(b) 擦除:写“1”　(a) 读“0”　(b) 读“1”

有三种操作：擦除（写1）、编程（写0）、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

- 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。
- 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

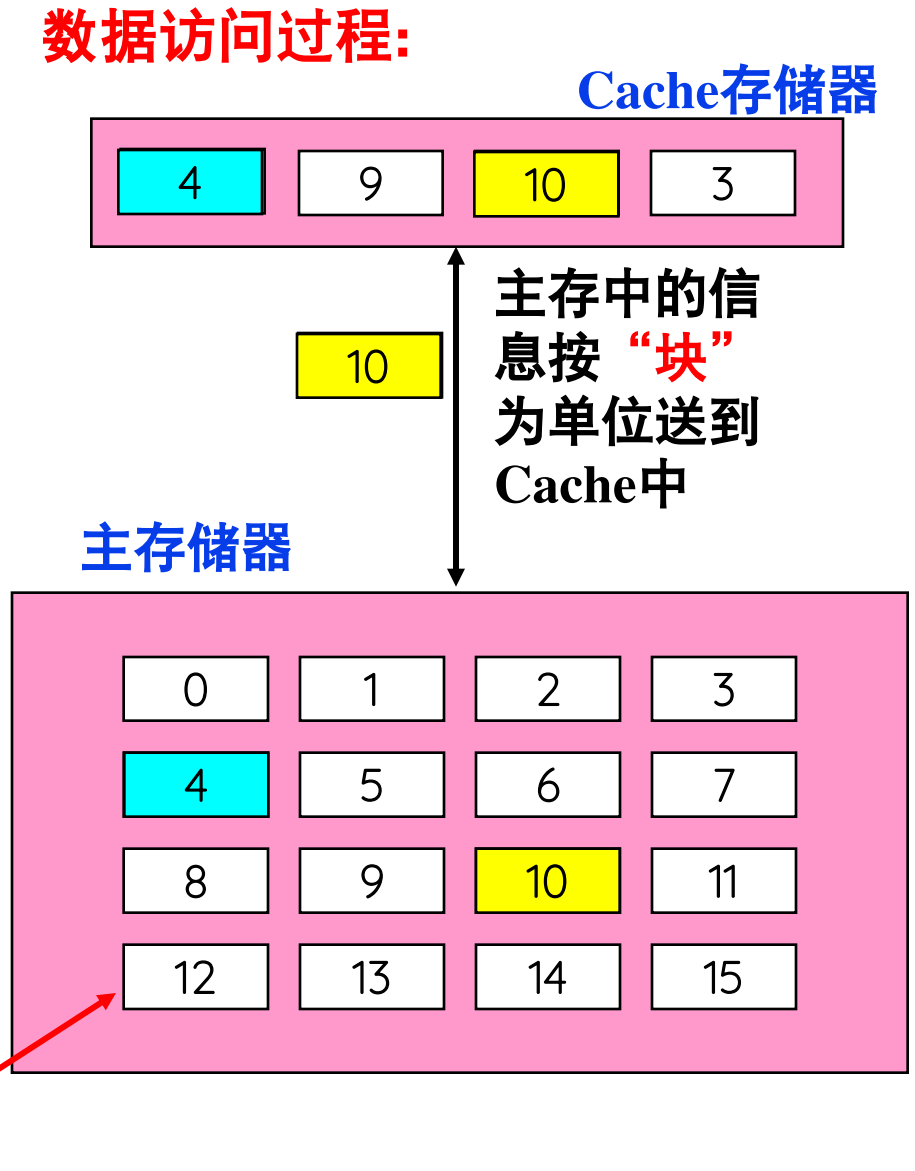
- 它用闪存颗粒代替了磁盘作为存储介质，利用闪存的特点，以区块写入和抹除的方式进行数据的写入。
- 写操作比读操作慢。顺序读比顺序写大致快一倍，随机读比随机写大致快10倍。
- 随机读写时延比硬盘低两个数量级（随机读约几十微秒，随机写约几百微秒）。
- 一个闪存芯片由若干个区块组成，每个区块由若干页组成。通常，页大小为512B~4KB，每个区块由32~128个页组成，因而区块大小为16KB~512KB，数据可以按页为单位进行读写。
- 当需要写某页信息时，必须先对该页所在的区块进行擦除操作。一旦一个区块被擦除过，区块中的每一页就可以直接再写一次。若某一区块进行了大约100 000次重复写之后，就会被磨损而变成坏的区块，不能再被使用。因此，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。由此可见，对于物理区块的写优化是由SSD中的硬件实现的，无需软件进行写优化。

层次结构存储系统

- 分以下四个部分介绍
  - 第一讲：存储器概述
  - 第二讲：半导体随机存取存储器
    - 基本存储元件、DRAM芯片、 SDRAM芯片技术
    - 内存条及其与CPU的连接
    - 存储器芯片的扩展、主存控制器
  - 第三讲：外部存储器
    - 磁盘存储器、闪速存储器、U盘、固态硬盘
  - 第四讲：高速缓冲存储器(cache)
    - cache的基本工作原理
    - 映射方式、替换算法、写策略
    - Cache的设计
    - Cache和程序性能

Cache(高速缓存)是什么样的?

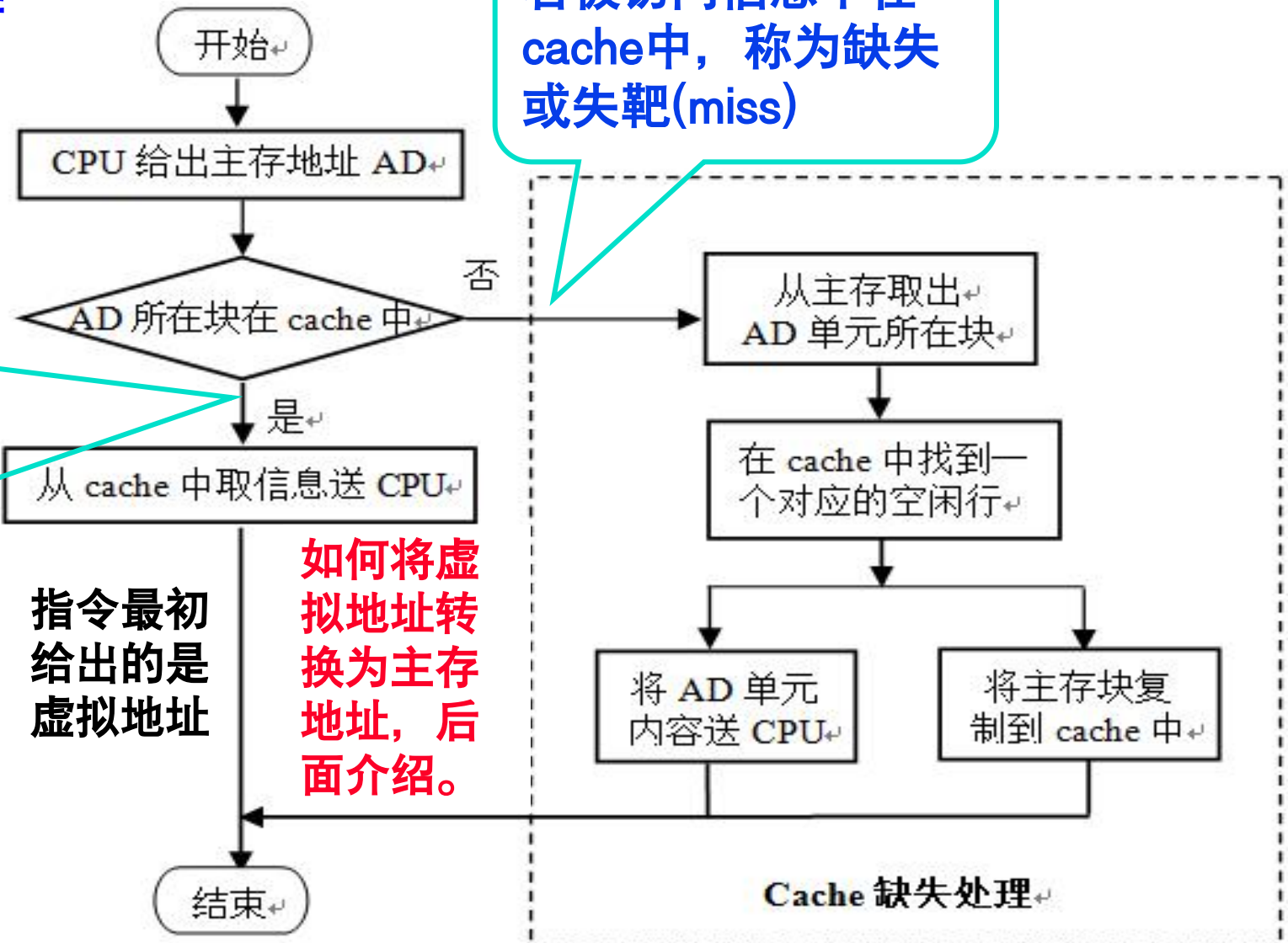
- Cache是一种小容量高速缓冲存储器，它由SRAM组成。
- Cache直接制作在CPU芯片内，速度几乎与CPU一样快。
- 程序运行时，CPU使用的一部分数据/指令会预先成批拷贝在Cache中，Cache的内容是主存储器中部分内容的副本。
- 当CPU需要从内存读(写)数据或指令时，先检查Cache，若有，就直接从Cache中读取，而不用访问主存储器。



Cache 的操作过程

问题：什么情况下，CPU产生访存要求？

执行指令过程中！



若被访问信息不在cache中，称为缺失或失靶(miss)

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。

Cache（高速缓存）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映射(Cache Mapping)

- 什么是Cache的映射功能?
  - 把访问的主存块取到Cache中时，该放到Cache的何处?
  - Cache行比主存块少，因而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

The Simplest Cache: Direct Mapped Cache

° Direct Mapped Cache（直接映射Cache举例）

- 主存每一块只能映射到固定的Cache行（槽）
- 也称模映射(Module Mapping)
- 映射关系为:

举例：书和书架的关系
块（行）都从0开始编号

Cache行号=主存块号 mod Cache行数

举例：4=100 mod 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。）

u 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

SKIP

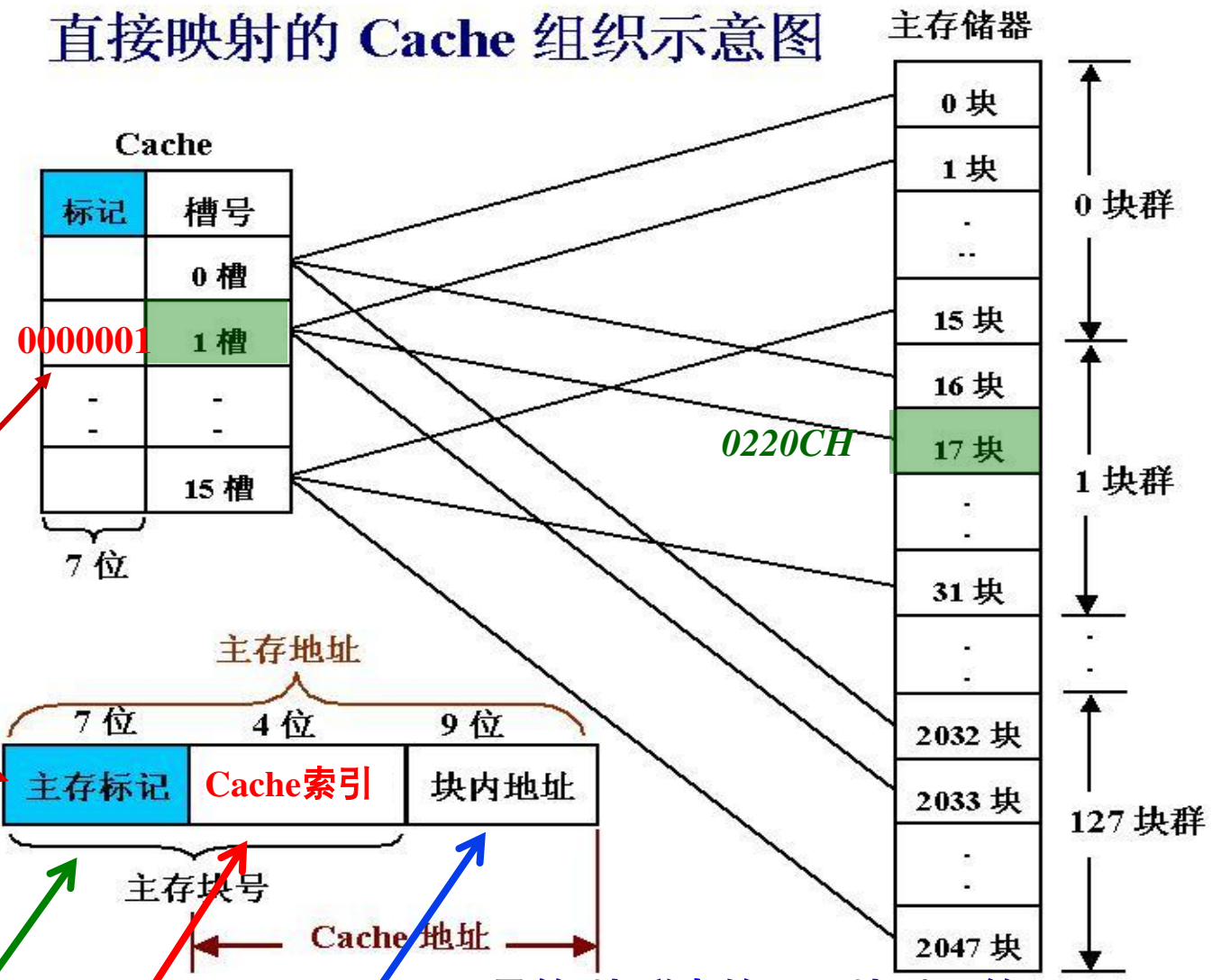
例如，需将主存第0块与第16块同时复制到Cache中时，由于它们都只能复制到Cache第0行，即使Cache其它行空闲，也有一个主存块不能写入Cache。这样就会产生频繁的 Cache装入。

直接映射Cache组织示意图

假定主存块为512B。

Cache大小：
2^{13}B=8KB=16行 x 512B/ 行

主存大小：
2^{20}B=1024KB=2048块 x 512B/块



Cache标记(tag)指出对应行取自哪个主存块群

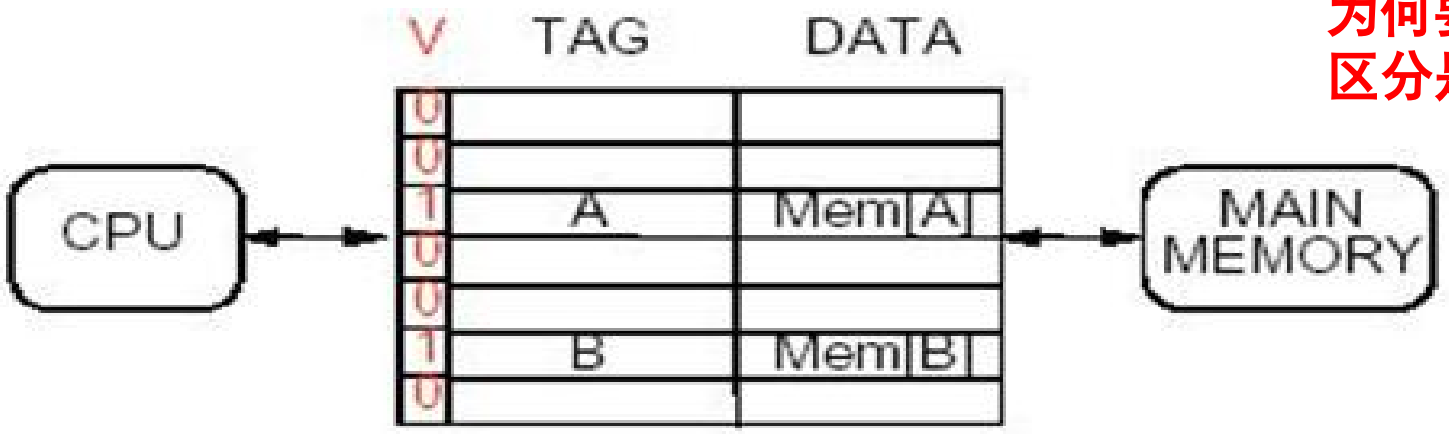
指出对应地址位于哪个块群

例：如何对0220CH单元进行访问?

0000 0010 0010 0000 1100B 是第1块群中的0001块（即第17块）中第12个单元！

有效位（Valid Bit）

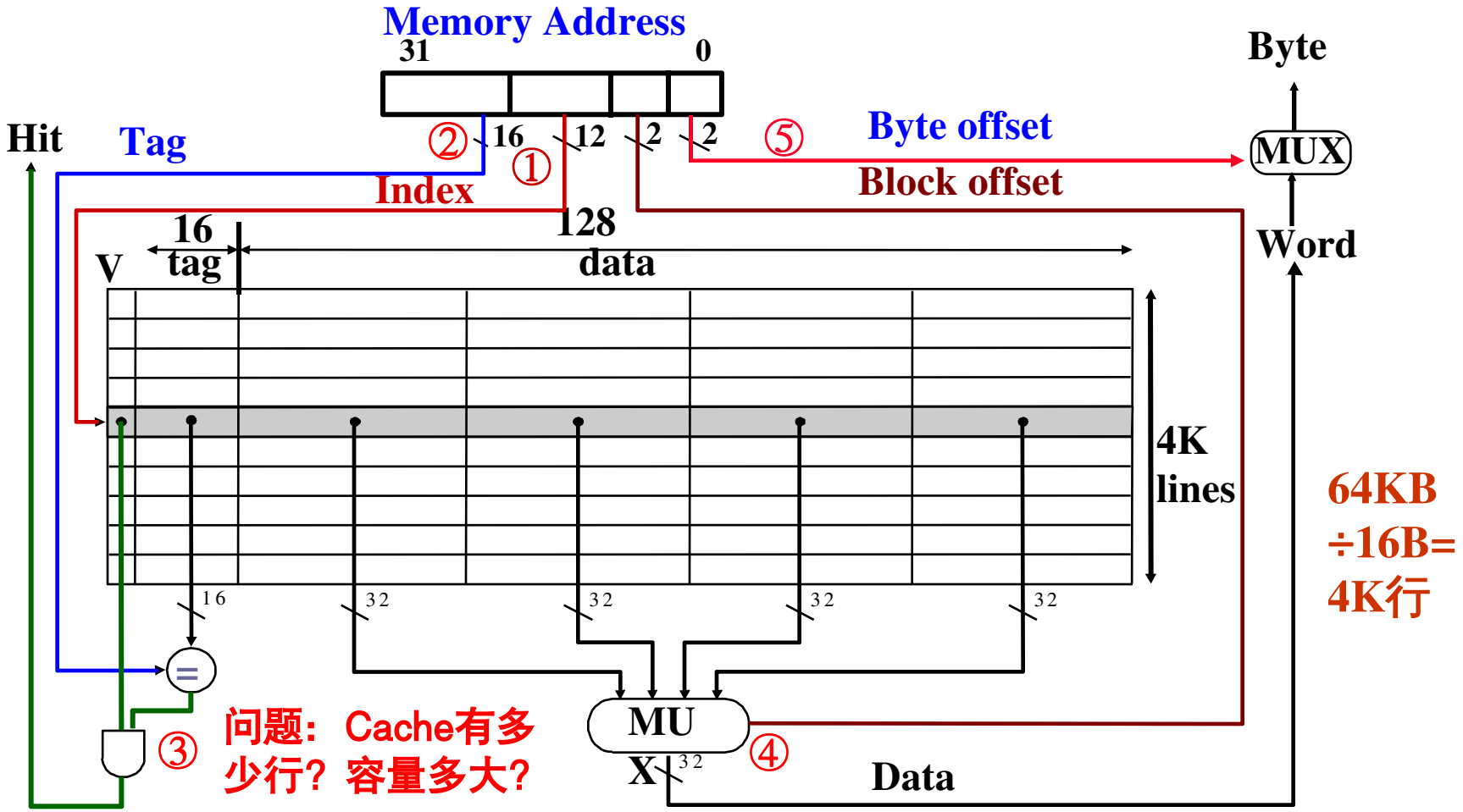
为何要用有效位来区分是否有效?



- 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!

64 KB Direct Mapped Cache with 16B Blocks

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。



64KB ÷16B= 4K行

容量 4Kx(1+16)+64Kx8=580Kbits=72.5KB,
数据占64KB / 72.5KB = 88.3%

如何计算Cache的容量?

Consider a cache with 64 Lines and a block size of 16 bytes.

What line number does byte address 1200 map to?

地址1200对应存放在第11行。1200/16取整=75，75 module 64 = 11

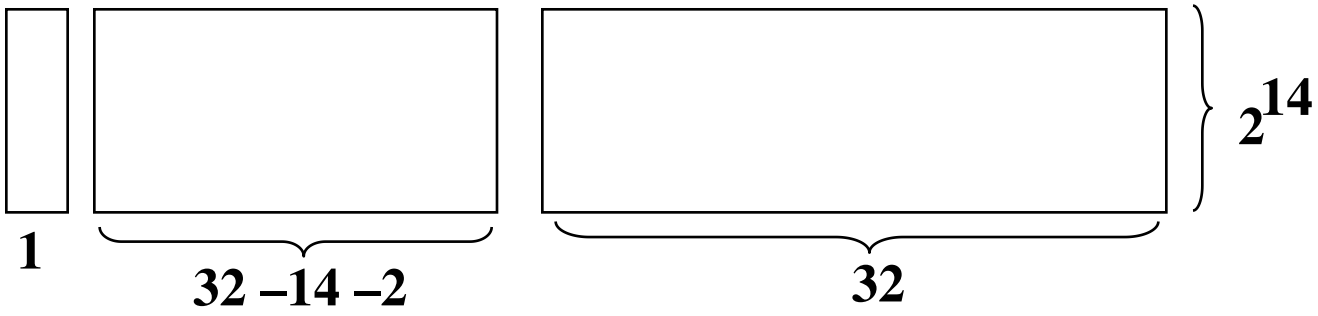
1200 = 1024+128+32+16 = 0…01 001011 0000 B

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: $2^{14} \times (32 + (32-14-2)+1) = 2^{14} \times 49 = 784$ Kbits

若块大小为4个字呢? $2^{14} \times (4 \times 32 + (32-14-2-2)+1) = 2^{14} \times 143 = 2288$ Kbits

若块大小为2^m个字呢? $2^{14} \times (2^m \times 32 + (32-14-2-m)+1)$

BACK

全相联映射Cache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 x 512B/行

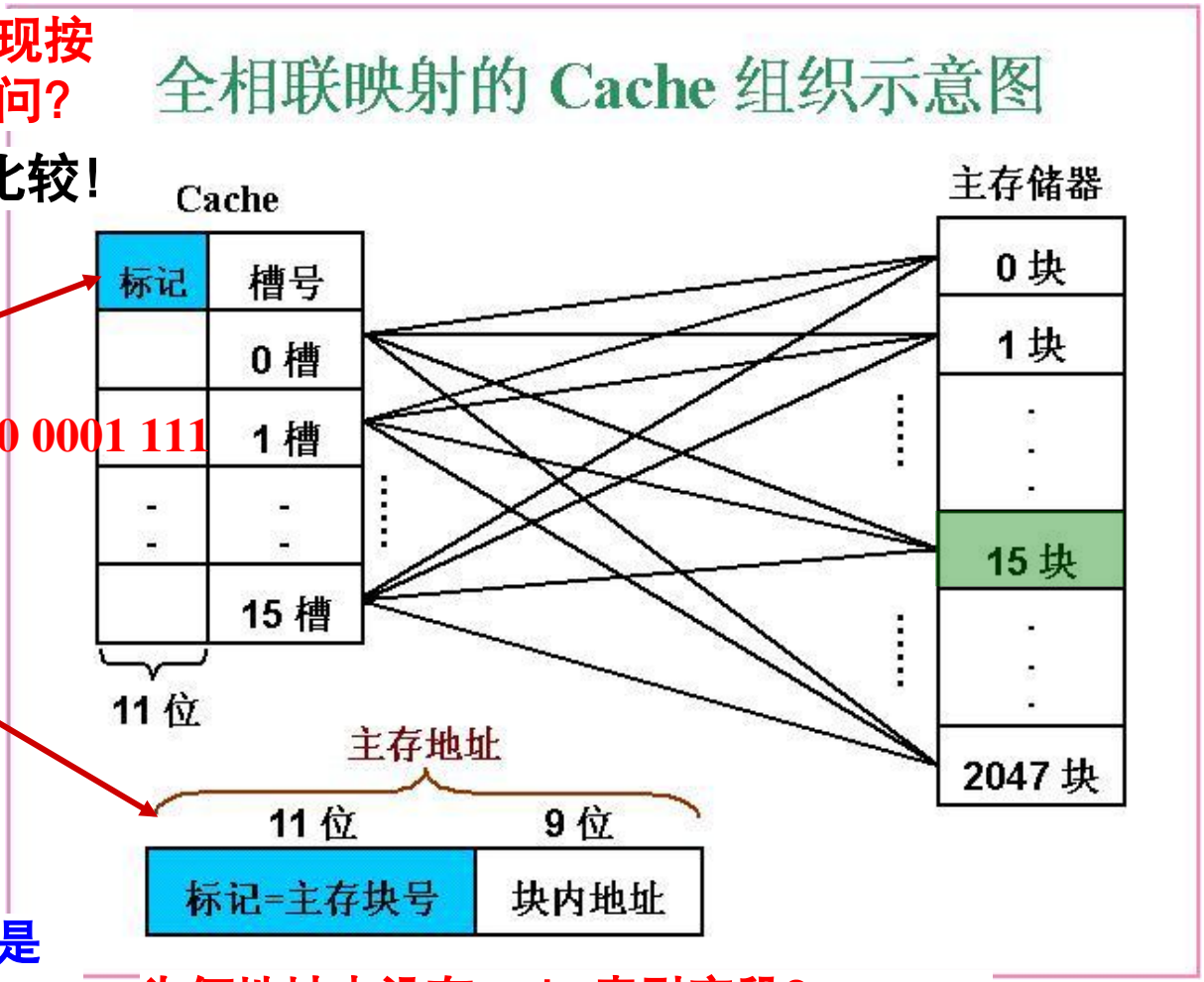
主存大小：
2^{20}B=1024KB=2048块 x 512B/块

按内容访问，是相联存取方式！

每个主存块可装到Cache任一行中。

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 0000 0001 111

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？

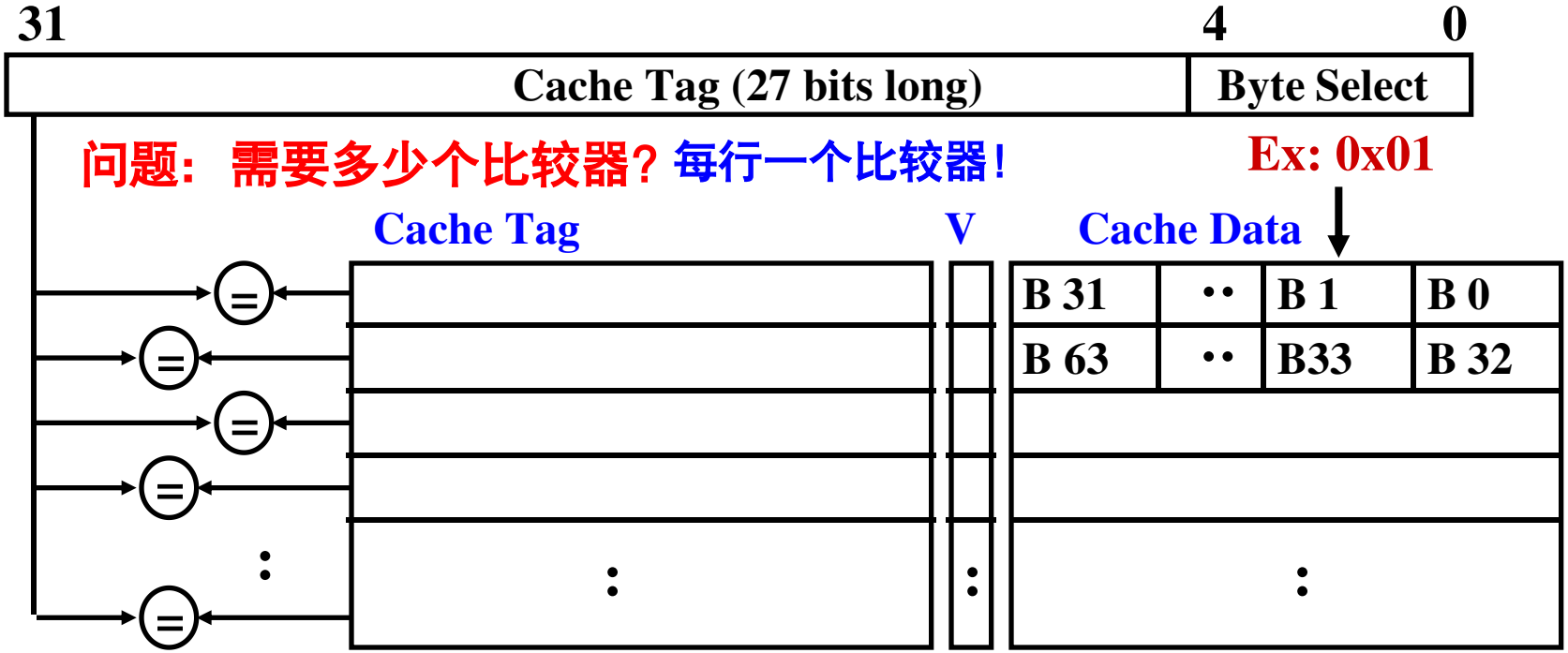
0000 0001 1110 0000 1100B 是第15块中的第12个单元！

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！

举例： Fully Associative

- Fully Associative Cache
  - 无需Cache索引，为什么？ 因为同时比较所有Cache行的标志
- By definition: Conflict Miss = 0
  - (没有冲突缺失，因为只要有空闲Cache块，都不会发生冲突)
- Example: 32bits memory address, 32 B blocks. 比较器位数多长?
  - we need N 27−bit comparators

缺点：比较器个数多，比较器位数多！



最多只有一个比较器的输出为1

组相联映射（Set Associative）

- 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：

  Cache组号 = 主存块号 mod Cache组数

  举例：假定Cache划分为：8K字=8组x2行/组x512字/行

  4=100 mod 8

  （主存第100块应映射到Cache的第4组的任意行中。）

u 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 Cahce和L3 Cahce中使用4-路以上。

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 x 512B/行

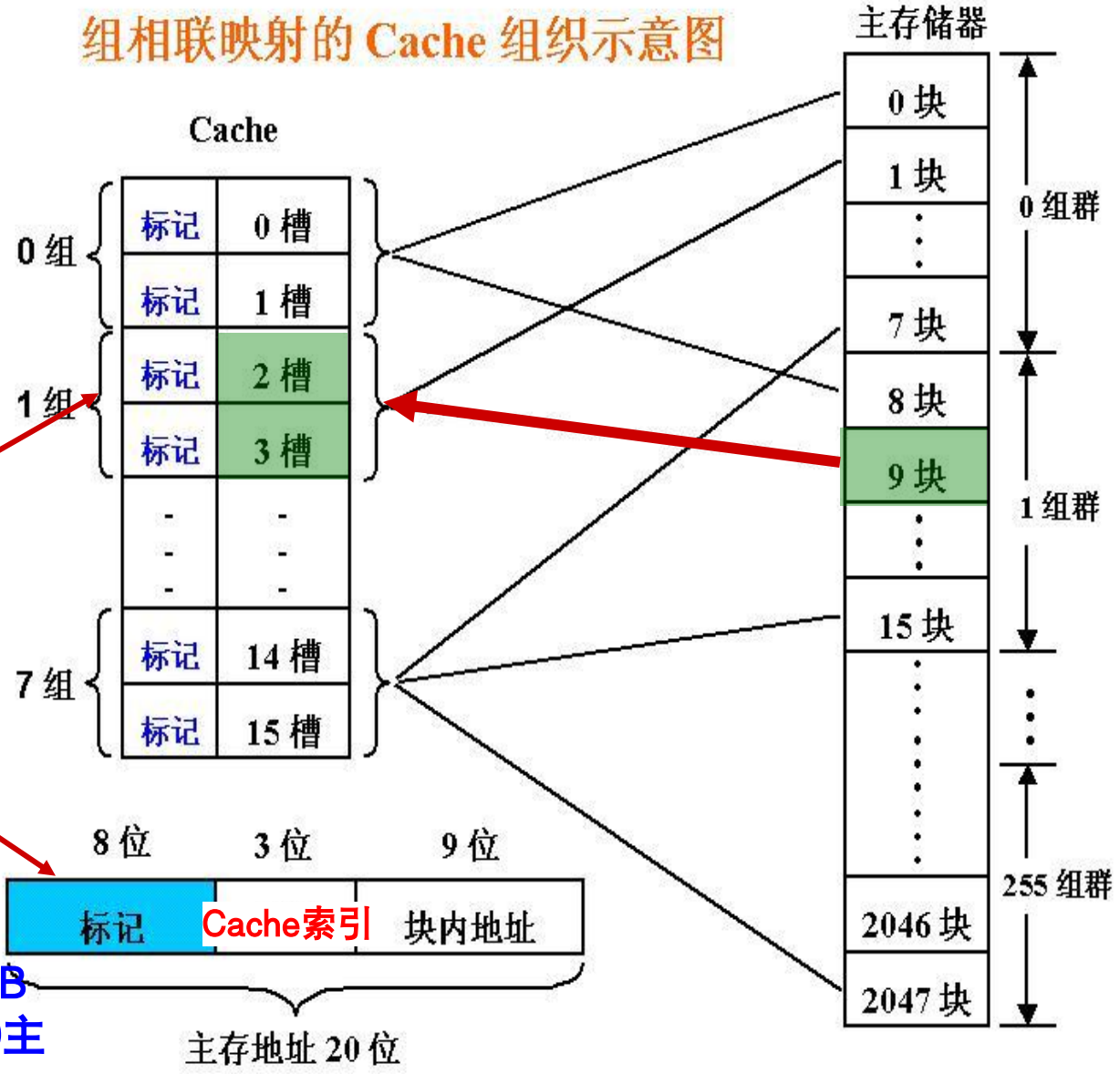
主存大小：
2^{20}B=1024KB=2048块 x 512B/块

指出对应行取自哪个主存组群

指出对应地址位于哪个主存组群中

例：如何对0120CH单元进行访问？

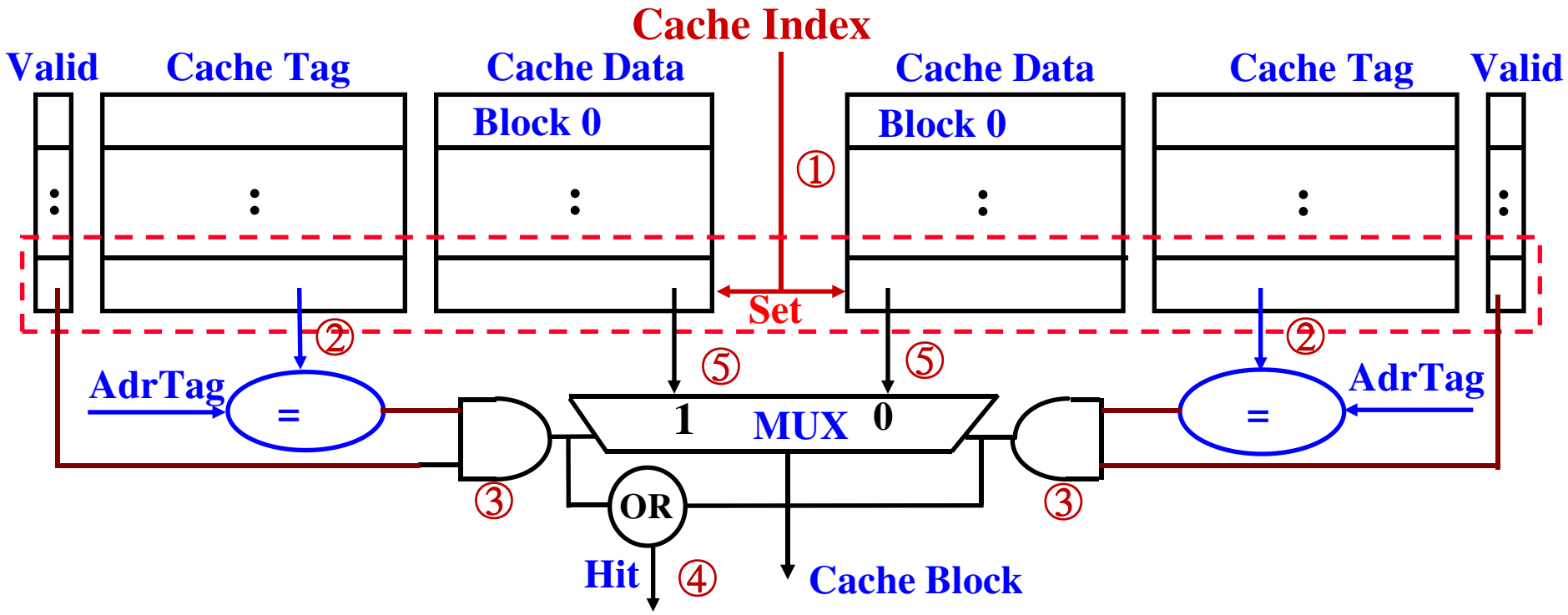
0000 0001 0010 0000 1100B是第1组群中的001块（即第9主存块）中第12个单元。 所以，映射到第一组中。



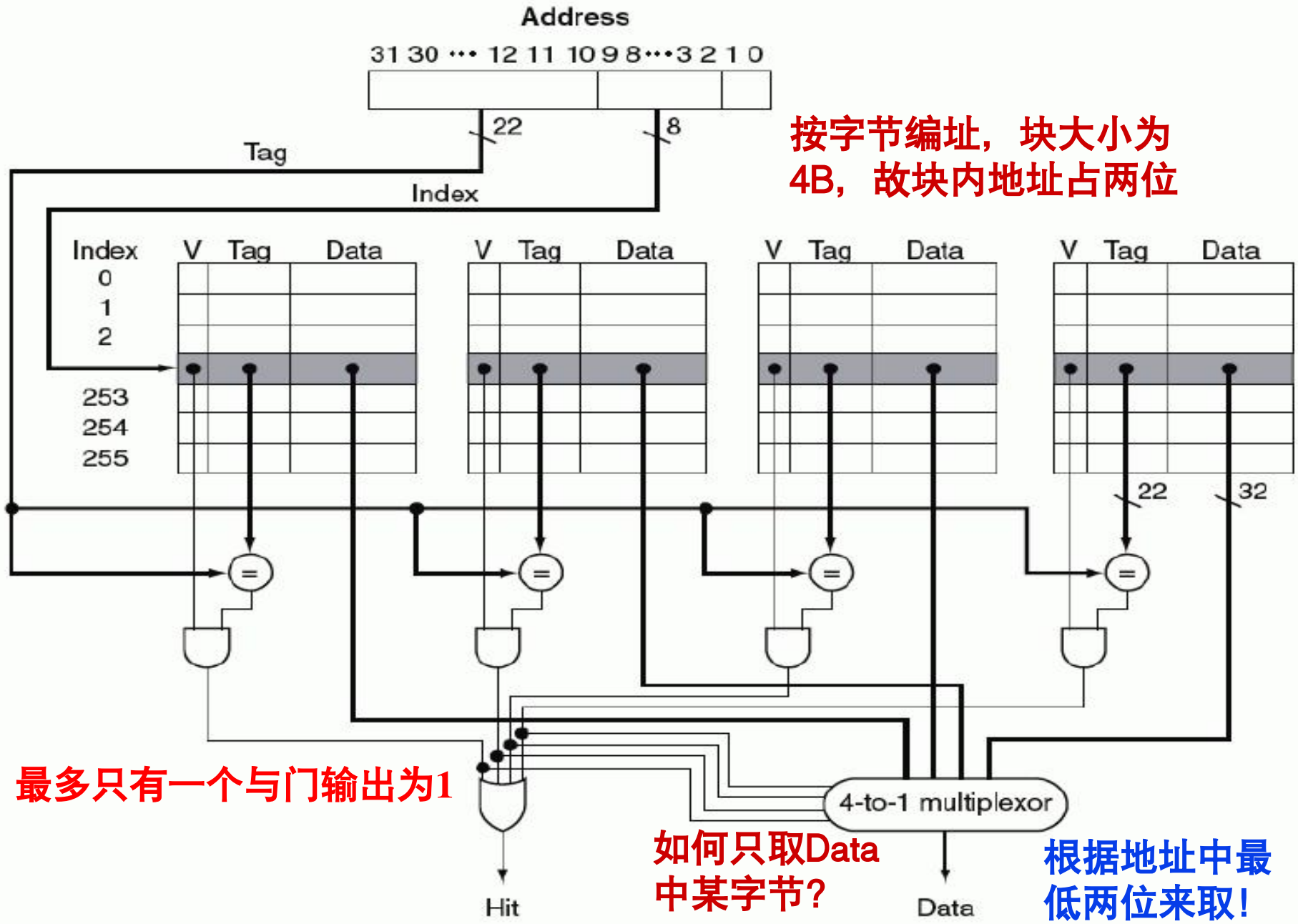
将主存地址标记和对应Cache组中每个Cache标记进行比较！

例1： A Two-way Set Associative Cache

- N-way set associative
  - N 个直接映射的行并行操作
- Example: Two-way set associative cache
  - Cache Index 选择一个Cache行集合Set（共2行）
  - 对集合中的两个Cache行的Tag并行进行比较
  - 根据比较结果确定信息在哪个行，或不在Cache中



例2：4-路组相联方式



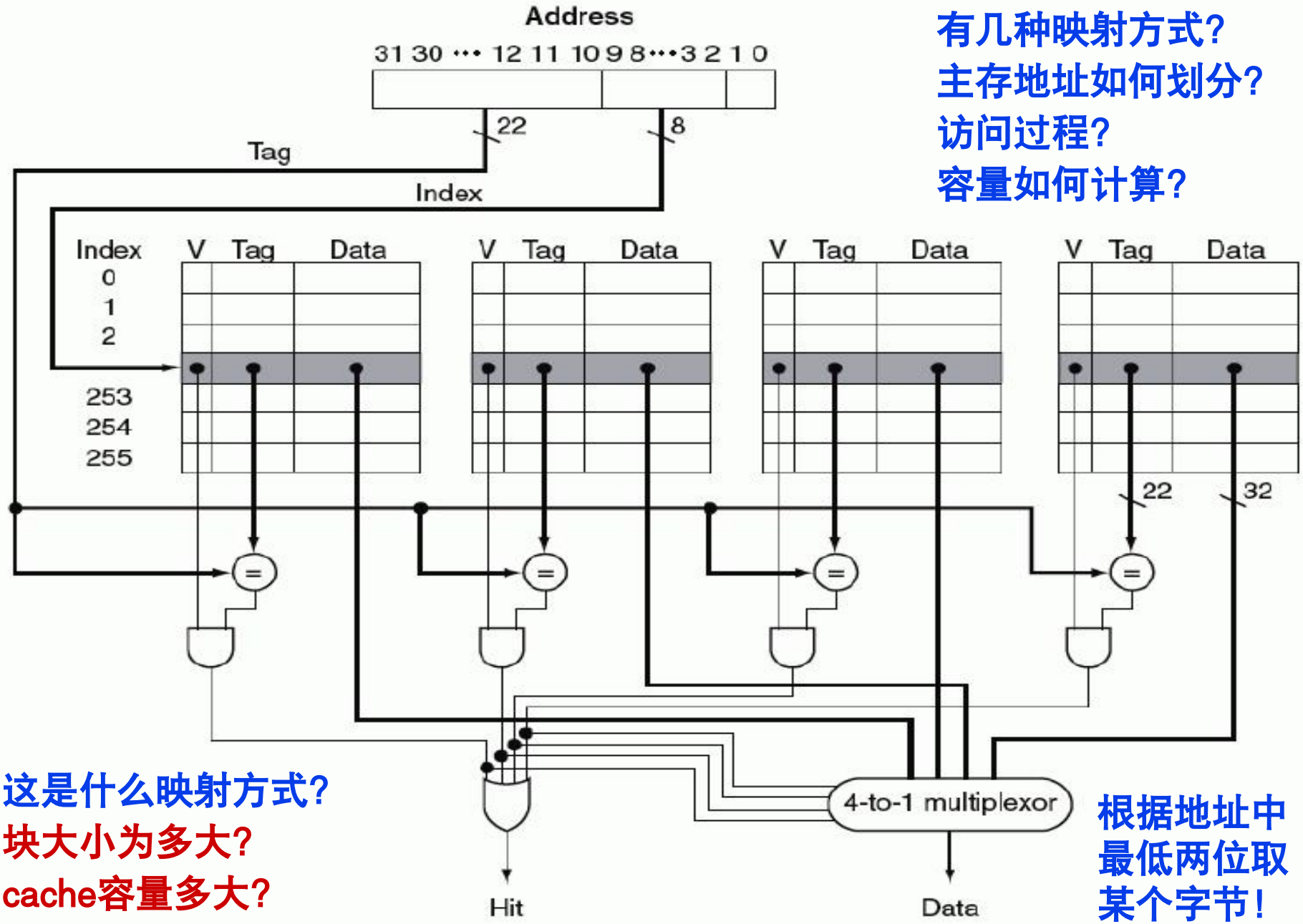
按字节编址，块大小为4B，故块内地址占两位

最多只有一个与门输出为1

如何只取Data中某字节？

根据地址中最低两位来取！

复习：cache和主存映射方式



有几种映射方式?
主存地址如何划分?
访问过程?
容量如何计算?

这是什么映射方式?
块大小为多大?
cache容量多大?

根据地址中最低两位取某个字节!

命中率、缺失率、缺失损失

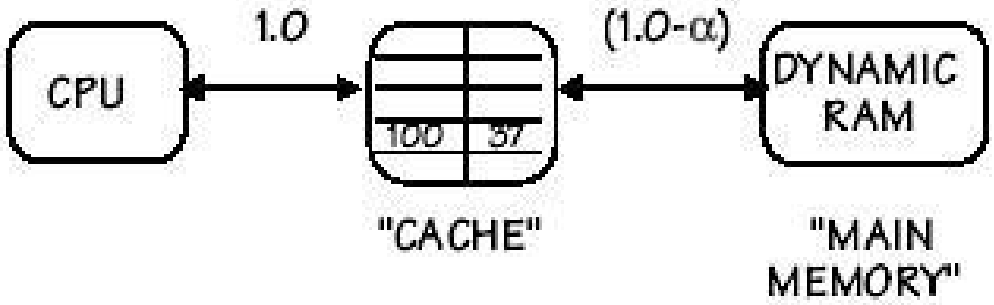
- Hit: 要访问的信息在Cache中
  - Hit Rate(命中率): 在Cache中的概率
  - Hit Time（命中时间）: 在Cache中的访问时间，包括:

    Time to determine hit/miss + Cache access time

    (即: 判断时间 + Cache访问)
- Miss: 要找的信息不在Cache中
  - Miss Rate（缺失率）= 1 – (Hit Rate)
  - Miss Penalty (缺失损失): 访问一个主存块所花时间
- Hit Time << Miss Penalty （Why?）

Average access time(平均访问时间)

Program-Transparent Memory Hierarchy



Cache contains TEMPORARY COPIES of selected main memory locations... eg. Mem[100] = 37

GOALS:

1) Improve the *average access* time

要提高平均访问速度，必须提高命中率！

α HIT RATIO: Fraction of refs found in CACHE.

(1-α) MISS RATIO: Remaining references.

$$t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$$

2) Transparency (compatibility, programming ease)

Challenge:
To make the hit ratio as high as possible.

Cache对程序员来说是透明的，以方便编程！

命中率到底应该有多大?

How high of a hit ratio?

Suppose we can easily build an on -chip static memory with a 4 nS access time, but the fastest dynamic memories that we can buy for main memory have an average access time of 40 nS. How high of a hit rate do we need to sustain an average access time of 5 nS?

$$\alpha = 1 - \frac{t_{ave} - t_c}{t_m} = 1 - \frac{5-4}{40} = 97.5\%$$

WOW, a cache really needs to be good?

三种映射方式的比较

- 对于一个主存块来说，三种映射方式下所能映射到cache行的数量不同，这种特性可用关联度来度量。
  - 直接映射关联度为1；全相联可映射到任意行，关联度为cache总行数；N路组相联可以映射到N行，关联度为N。
- 当cache大小、主存块大小一定时，关联度和命中率、命中时间、标记所占额外开销等有如下关系
  - 关联度越低，命中率越低
  - 关联度越低，判断是否命中的电路开销越小，电路延迟越短
  - 关联度越低，每个cache行中的标记所占额外空间越少

若主存地址32位，按字节编址，主存块大小为16B，则：

关联度为1（即直接映射）时，每组1行，共4K组，标记占32－4－12=16位，总位数为4K×16=64K位；

关联度为2（即2路组相联）时，每组2行，共2K组，标记占32－4－11=17位，总位数为4K×17=68K位；

关联度为4（即4路组相联）时，每组4行，共1K组，标记占32－4－10=18位，总位数为4K×18=72K位；

全相联时，每组4K行，标记占32－4=28位，总位数为4K×28 =112K位。

替换(Replacement)算法

- 问题举例:

  组相联映射时，假定第0组的两行分别被主存第0和8块占满，此时若需调入主存第16块，根据映射关系，它只能放到Cache第0组，因此，第0组中必须调出一块，那么调出哪一块呢?

  这就是淘汰策略问题，也称替换算法。

- 常用替换算法有:
  - 先进先出FIFO （first-in-first-out）
  - 最近最少用LRU （ least-recently used）
  - 最不经常用LFU （ least-frequently used）
  - 随机替换算法（Random）

  等等

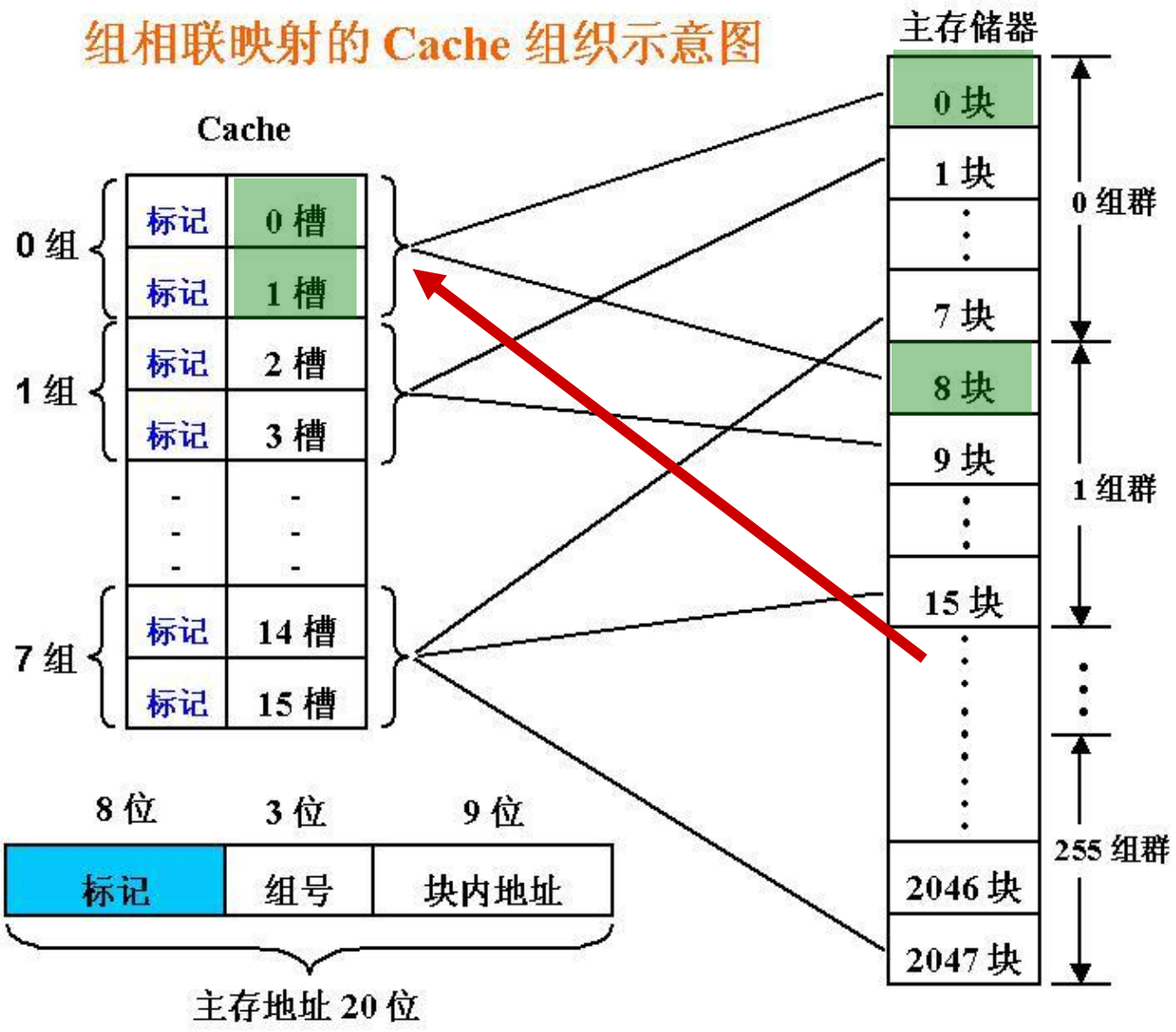
SKIP

这里的替换策略和后面的虚拟存储器所用的替换策略类似，将是以后操作系统课程的重要内容，本课程只做简单介绍。有兴趣的同学可以自学。

假定第0组的两行分别被主存第0块和第8块占满，此时若需调入主存第16块

该怎么办?

第0组中必须调出一块，那么，调出哪一块呢?



BACK

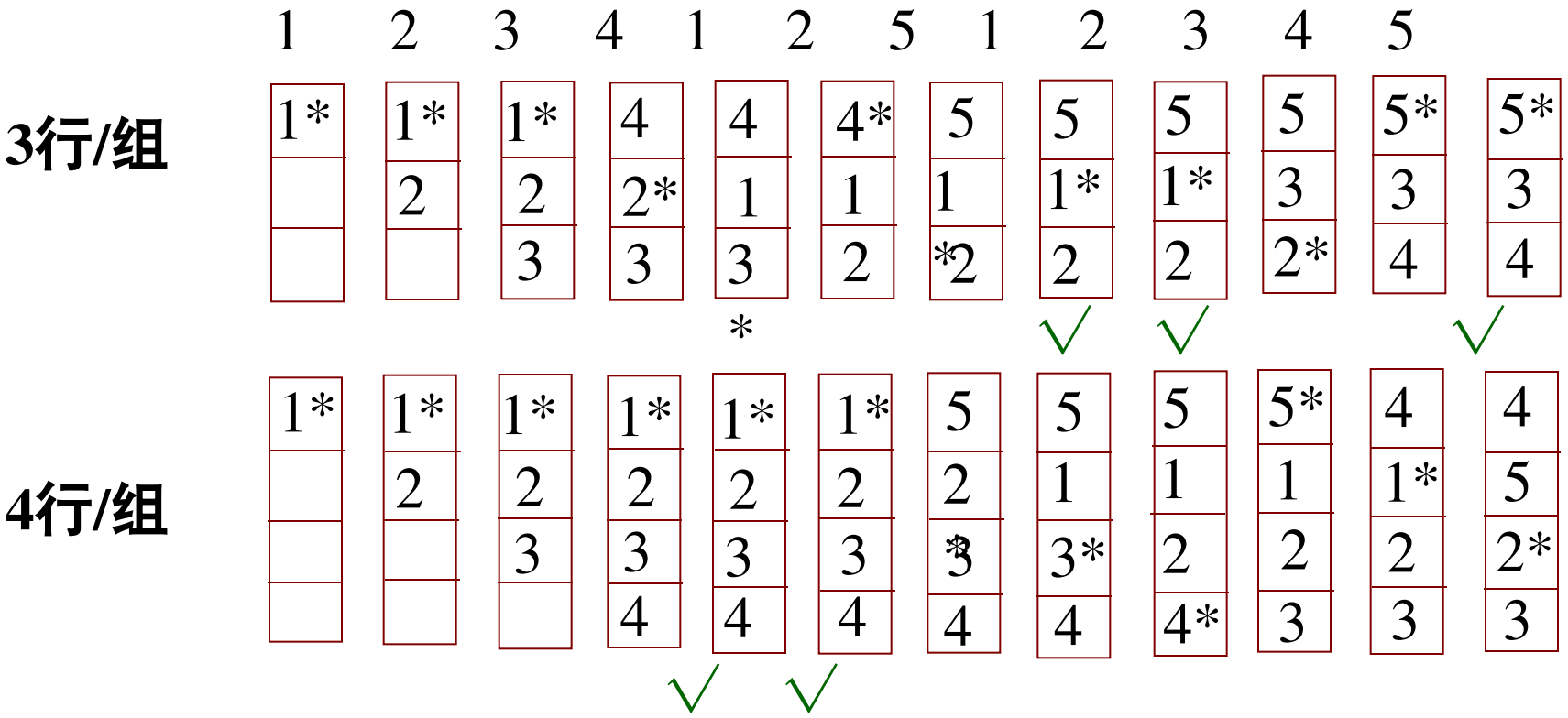
替换算法-先进先出（FIFO）

- 总是把最先进入的那一块淘汰掉。

总是把最先从图书馆搬来的书还回去！

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组的情况。

注：通常一组中含有2^k行，这里3行/组主要为了简化问题而假设



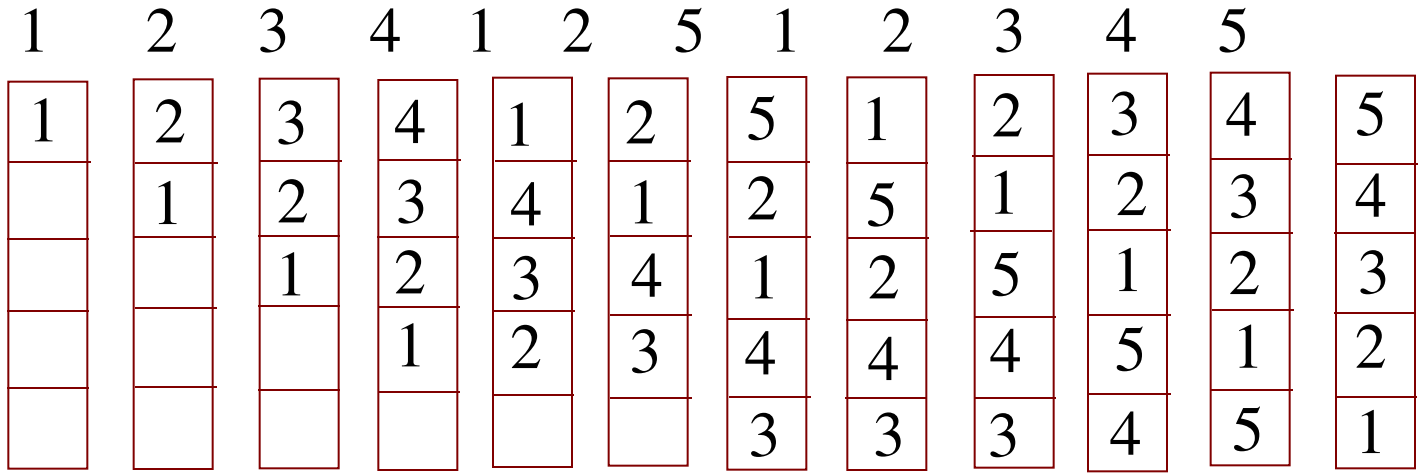
由此可见，FIFO不是一种栈算法，即命中率并不随组的增大而提高。

替换算法-最近最少用(LRU)

总是把最长时间不看的书还回去！

- 总是把最近最少用的那一块淘汰掉。

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组、 5行/组的情况。



3行/组
4行/组
5行/组

LRU是一种栈算法其命中率随组的增大而提高

LRU实现时不能移动块，而是给每个cache行设定一个计数器，以记录块的使用情况，称LRU位

替换算法-最近最少用

通过计数值（LRU位）来确定cache行中主存块的使用情况

即：计数值为0的行中的主存块最常被访问，计数值为3的行中的主存块最不经常被访问，先被淘汰！

- 计数器变化规则：
  - Ø 每组4行时，计数器有2位。计数值越小则说明越被常用。
  - Ø 命中时，被访问行的计数器置0，比其低的计数器加1，其余不变。
  - Ø 未命中且该组未满时，新行计数器置为0，其余全加1。
  - Ø 未命中且该组已满时，计数值为3的那一行中的主存块被淘汰，新行计数器置为0，其余加1。

1		2		3		4		1		2		5		1		2		3		4		5	
0	1	1	1	2	1	3	1	0	1	1	1	2	1	0	1	1	1	2	1	3	1	0	5
		0	2	1	2	2	2	3	2	0	2	1	2	2	2	0	2	1	2	2	2	3	4
				0	3	1	3	2	3	3	3	0	5	1	5	2	5	3	5	0	4	2	3
						0	4	1	4	2	4	3	4	3	4	3	4	0	3	1	3	1	2

The Need to Replace! (何时需要替换？)

- Direct Mapped Cache:
  - 映射唯一，毫无选择，无需考虑如何替换
- N-way Set Associative Cache:
  - 每个主存块有N个Cache行可选择，需考虑如何替换
- Fully Associative Cache:
  - 每个主存块可存放到Cache任意行中，需考虑如何替换

结论：若Cache miss in a N-way Set Associative or Fully Associative Cache，则需考虑如何替换。其过程为：

- 从主存取出一个新块
- 选择一个有映射关系的空Cache行
- 若对应Cache行被占满时又需调入新主存块，则必须考虑从Cache行中替换出一个主存块

举例

- 假定计算机系统主存空间大小为32Kx16位，且有一个数据区为4K字的4路组相联Cache，主存块大小为64字。假定Cache开始为空，处理器顺序地从存储单元0、1、…、4351中取数，一共重复10次。设Cache比主存快10倍。采用LRU算法。试分析Cache的结构和主存地址的划分。说明采用Cache后速度提高了多少？采用MRU算法后呢？
- 答：假定主存按字编址。每字16位。

  主存：32K字=512块 x 64字 / 块

  Cache：4K字=16组 x 4行 / 组 x 64 字 / 行

  主存地址划分为：

标志位	组号	字号
5	4	6

4352/64=68，所以访问过程实际上是对前68块连续访问10次。

举例

	第0 行	第1 行	第2 行	第3 行
第0组	0/64/48	16/0/64	32/16	48/32
第1组	1/65/49	17/1/65	33/17	49/33
第2组	2/66/50	18/2/66	34/18	50/34
第3组	3/67/51	19/3/67	35/19	51/35
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

LRU算法： 第一次循环,每一块只有第一字未命中,其余都命中;
以后9次循环,有20块的第一字未命中,其余都命中.
所以,命中率p为 (43520−68−9x20)/43520=99.43%
速度提高： tm/ta=tm/(tc+(1−p)tm)=10/(1+10x(1−p))=9.5倍

举例

	第0 行	第1 行	第2 行	第3 行
第0组	0/16/32/48	16/32/48/64	32/48/64/0	48/64/0/16
第1组	1/17/33/49	17/33/49/65	33/49/65/1	49/65/1/17
第2组	2/18/34/50	18/34/50/66	34/50/66/2	50/66/2/18
第3组	3/19/35/52	19/35/51/67	35/51/67/3	51/67/3/19
第4组	4	20	36	52
……	…	…	…	…
……	…	…	…	…
第15组	15组	31	47	63

MRU算法：第一次68字未命中；第2,3,4,6,7,8,10次各有4字未命中；第5,9次各有8字未命中；其余都命中.

所以,命中率p为（43520−68−7x4−2 x8)/43520=99.74%

速度提高：tm/ta=tm/(tc+(1−p)tm)=10/(1+10x(1−p))=9.77倍

写策略

- 为何要保持在Cache和主存中数据的一致?
  - 因为Cache中的内容是主存块副本，当对Cache中的内容进行更新时，就存在Cache和主存如何保持一致的问题。
  - 以下情况也会出现“Cache一致性问题”
    - 当多个设备都允许访问主存时

      例如：DMA控制器读写主存时，如果对应Cache行中被修改，则DMA读出的内容无效；若DMA修改了主存单元的内容，则对应Cache行中内容无效。
    - 当多个CPU（核）都有各自的Cache而共享主存（L3 cache）时

      某个CPU（核）修改了自身Cache中的内容，则对应的主存单元和其他CPU（核）中对应Cache内容都变为无效。
- 写操作有两种情况
  - 写命中（Write Hit）：要写的单元已经在Cache中
  - 写不命中（Write Miss）：要写的单元不在Cache中

写策略

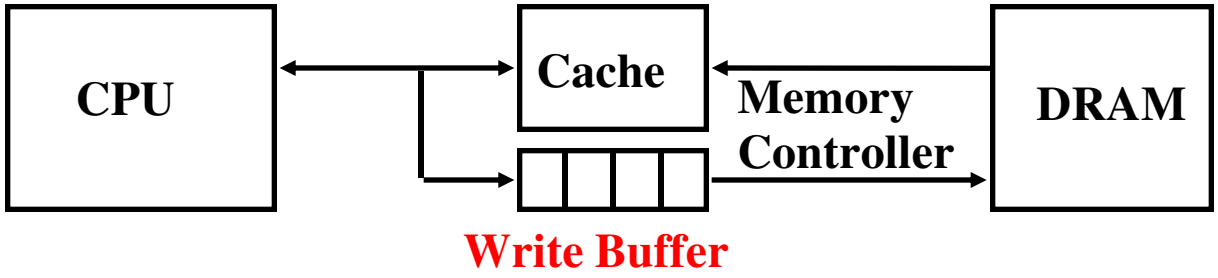
- 处理Cache读比Cache写更容易，故指令Cache比数据Cache容易设计
- 对于写命中，有两种处理方式
  - Write Through（通过式写、写直达、直写）
    - 同时写Cache和主存单元
    - What!!! How can this be? Memory is too slow(>100Cycles)?
      10%的存储指令使CPI增加到：1.0+100x10%=11
    - 使用写缓冲（Write Buffer）
  - Write Back（一次性写、写回、回写）
    - 只写cache不写主存，淘汰时一次写回，每行有个修改位（“dirty bit-脏位”），大大降低主存带宽需求，但控制较复杂
- 对于写不命中，有两种处理方式
  - Write Allocate（写分配）
    - 将主存块装入Cache，然后更新相应单元
    - 试图利用空间局部性，但每次都要从主存读一个块
  - Not Write Allocate（非写分配）
    - 直接写主存单元，不把主存块装入到Cache

直写Cache可用非写分配或写分配
回写Cache通常用写分配

为什么?

SKIP

Write Through中的Write Buffer



- 在 Cache 和 Memory之间加一个Write Buffer
  - CPU同时写数据到Cache和Write Buffer
  - Memory controller（存控）将缓冲内容写主存
- Write buffer（写缓冲）是一个FIFO队列
  - 一般有4项
  - 在存数频率不高时效果好
- 最棘手的问题
  - 当频繁写时，易使写缓存饱和，发生阻塞
- 如何解决写缓冲饱和?
  - 加一个二级Cache
  - 使用Write Back方式的Cache

BACK

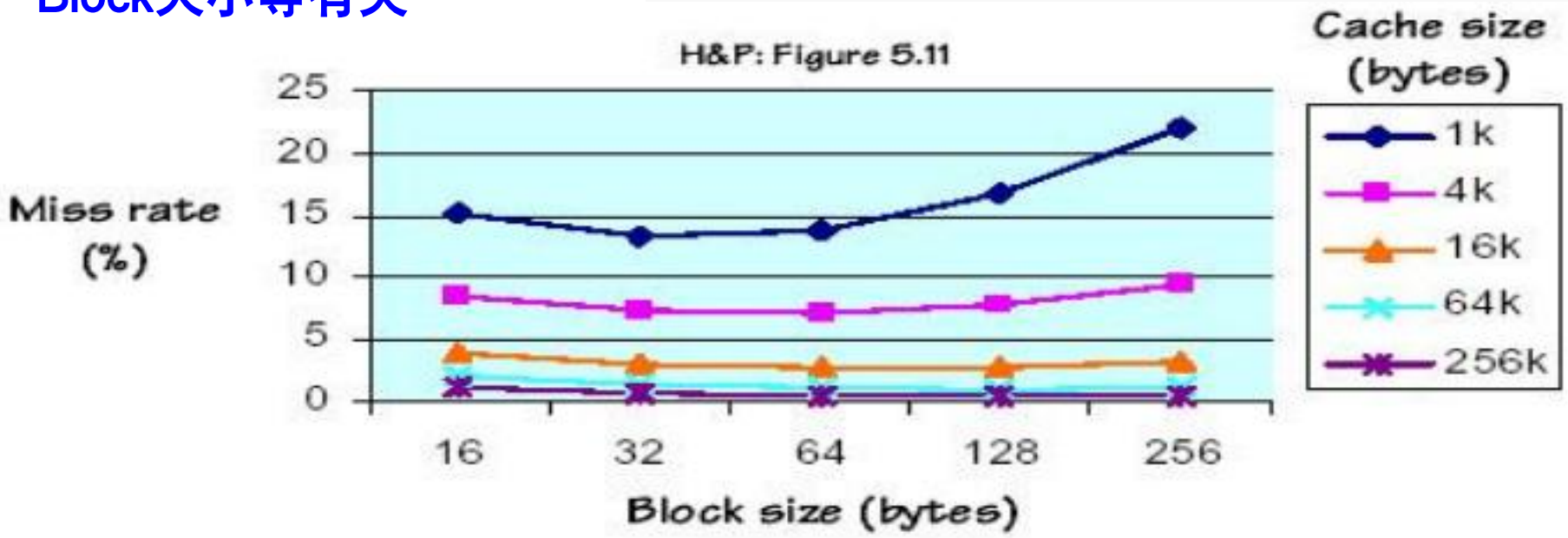
Cache大小、Block大小和缺失率的关系

Cache性能由缺失率确定

而缺失率与Cache大小、Block大小等有关

书架越大，在书架上找到书的概率就越大！

书架容量一定时，每排放的书越多，排数就越少。每排的大小会影响找到书的概率



- spatial locality: larger blocks → reduce miss rate
- fixed cache size: larger blocks
  - → fewer lines in cache
  - → higher miss rate, especially in small caches

Cache大小：Cache越大，Miss率越低，但成本越高！

Block大小：Block大小与Cache大小有关，不能太大，也不能太小！

系统中的Cache数目

SKIP

- 刚引入Cache时只有一个Cache。近年来多Cache系统成为主流
- 多Cache系统中，需考虑两个方面：

[1] 单级/多级?

外部(Off-chip)Cache:不做在CPU内而是独立设置一个Cache

片内(On-chip)Cache: 将Cache和CPU作在一个芯片上

单级Cache: 只用一个片内Cache

多级Cache: 同时使用L1 Cache和L2 Cache，甚至有L3 Cache，L1 Cache更靠近CPU，其速度比L2快，其容量比L2小

[2] 联合/分立?

分立: 指数据和指令分开存放在各自的数据和指令Cache中

一般L1Cache都是分立Cache，为什么?

两级Cache时L1Cache的命中时间比命中率更重要！为什么?

联合: 指数据和指令都放在一个Cache中

一般L2Cache都是联合Cache，为什么?

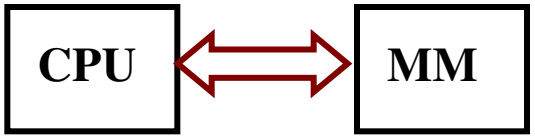
若只有两级cache，则L2Cache的命中率比命中时间更重要!

因为缺失时需从主存取数，并要送L1和L2cache，缺失损失大!

设计支持Cache的存储器系统

- 指令执行若发生Cache缺失，必须到DRAM中取数据或指令
- 在DRAM和Cache之间传输的单位是Block
- 问题：怎样的存储器组织使得Block传输最快（缺失损失最小）？

假定存储器访问过程：



CPU发送地址到内存：1个总线时钟

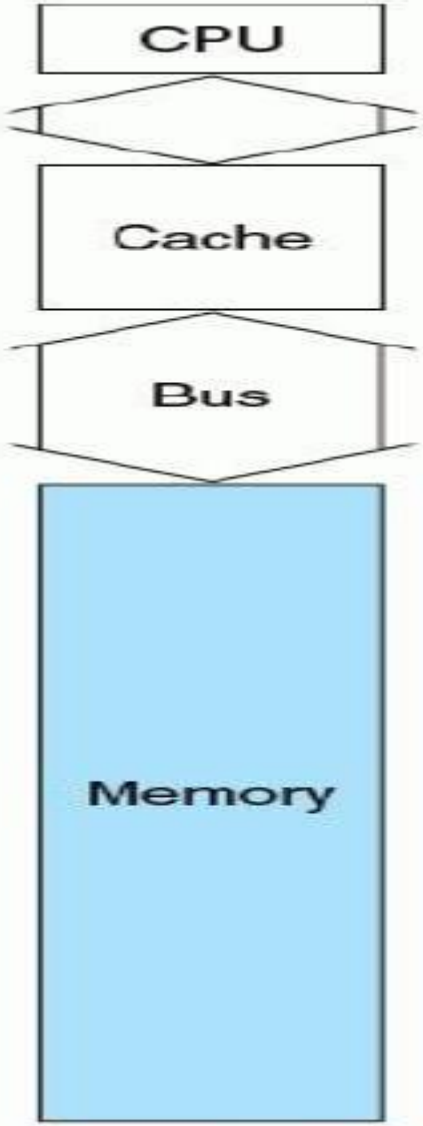
访问内存的初始化时间：10个总线时钟

从总线上传送一个字：1个总线时钟

可以有三种不同的组织形式！

假定一个Block有4个字，则缺失损失各为多少时钟?

设计支持Cache的存储器系统



a. One-word-wide memory organization

假定存储器访问过程:

CPU发送地址到内存：1个总线时钟

内存访问时间：10个总线时钟

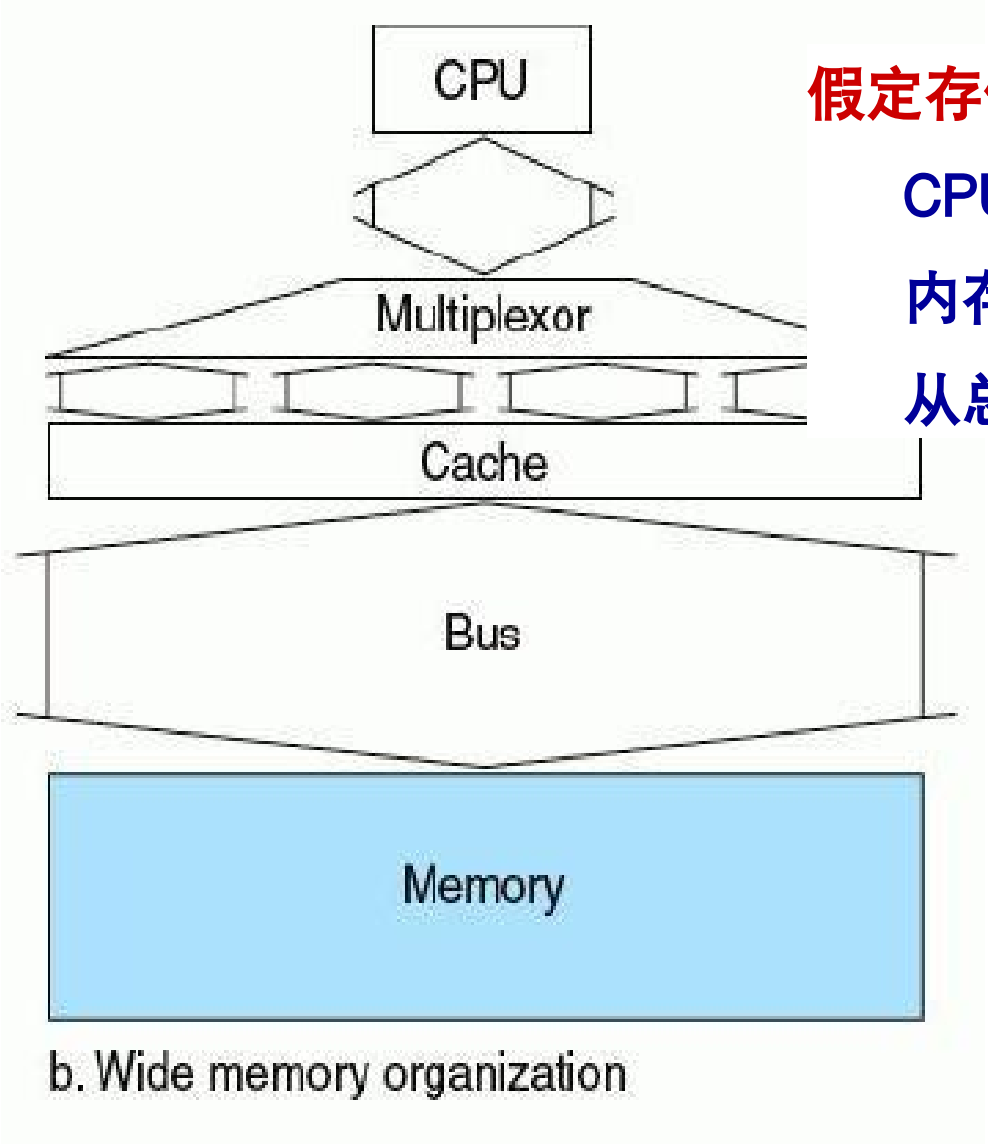
从总线上传送一个字：1个总线时钟

$4x(1+10+1)=48$

缺失损失为48个时钟周期

代价小，但速度慢!

设计支持Cache的存储器系统



b. Wide memory organization

假定存储器访问过程:

CPU发送地址到内存：1个总线时钟

内存访问时间：10个总线时钟

从总线上传送一个字：1个总线时钟

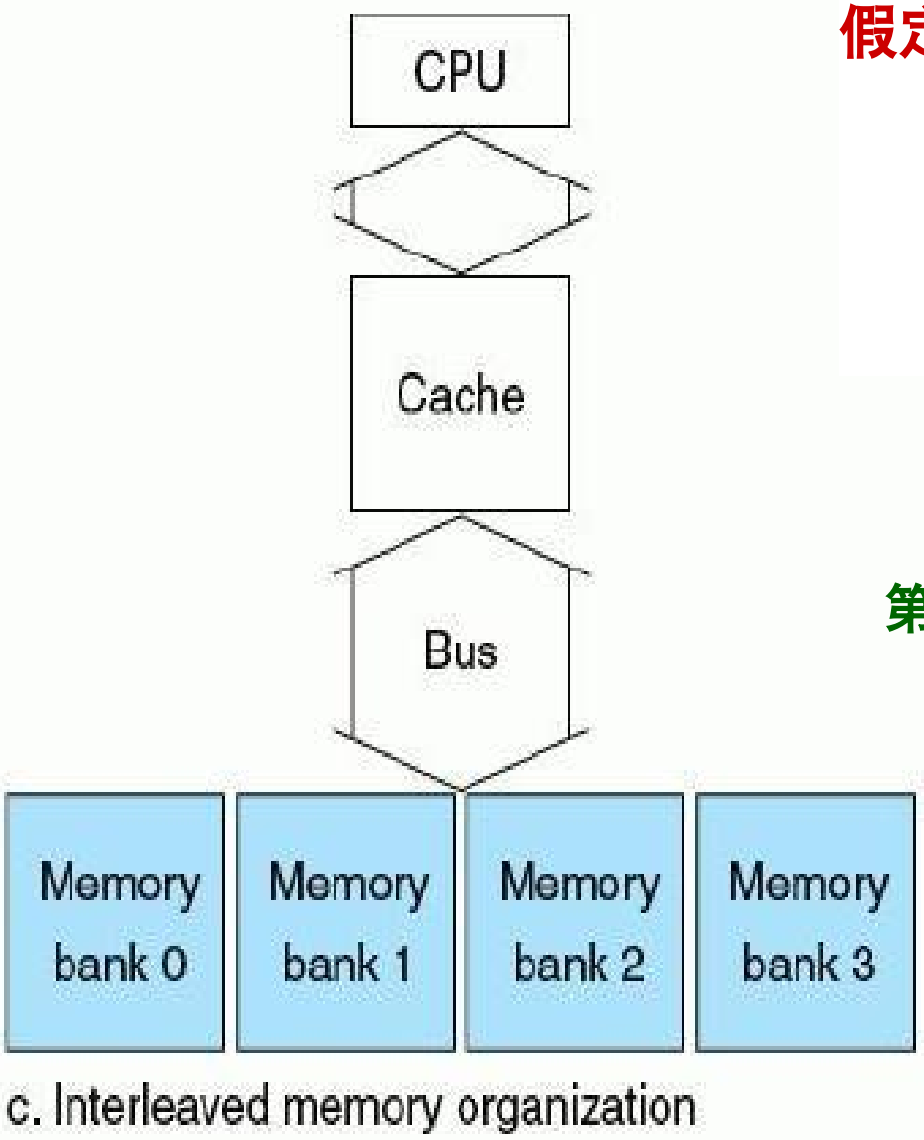
Two-word: 2x(1+10+1)=24

Four-word: 1+10+1=12

缺失损失各为24或12个时钟周期

速度快，但代价大!

设计支持Cache的存储器系统



c. Interleaved memory organization

假定存储器访问过程:

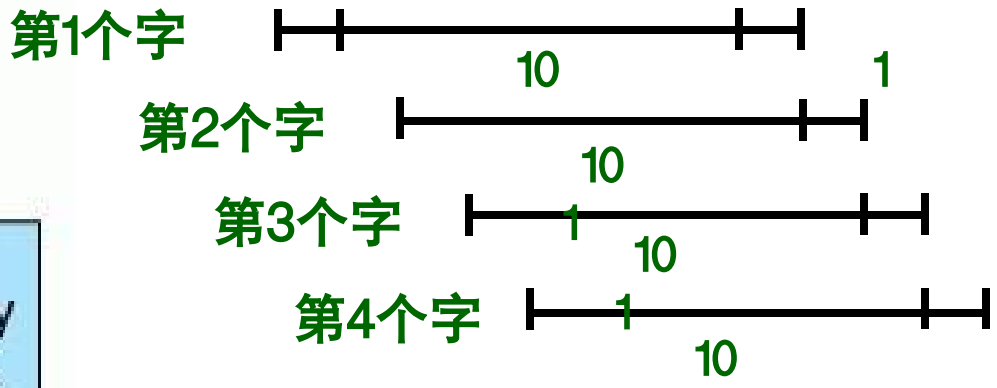
CPU发送地址到内存：1个总线时钟

内存访问时间：10个总线时钟

从总线上传送一个字：1个总线时钟

Interleaved four banks

one-word: 1+1x10+4x1=15

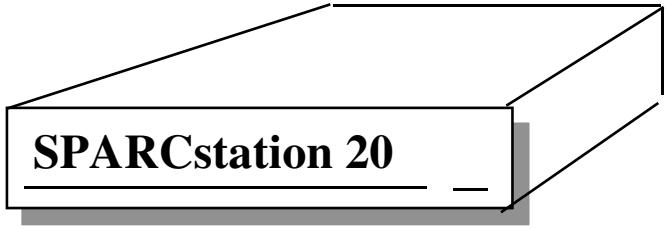


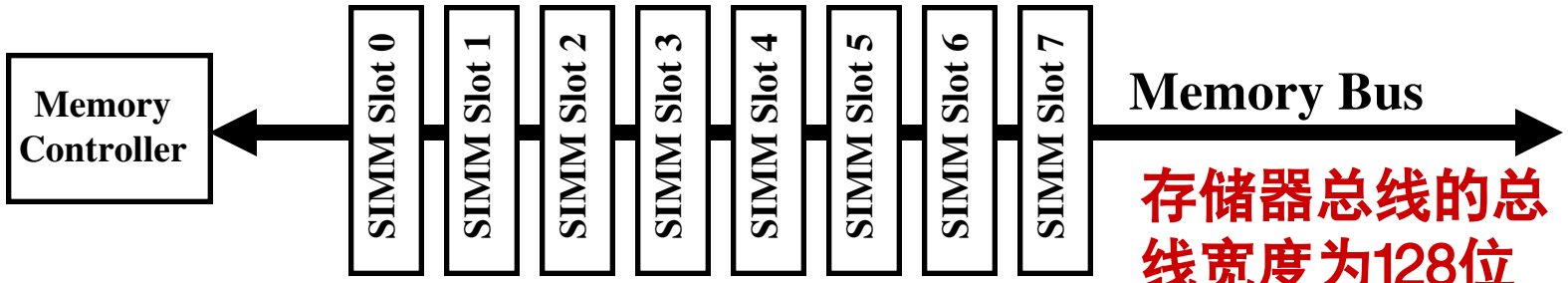
缺失损失为15个时钟周期

代价小，而且速度快！

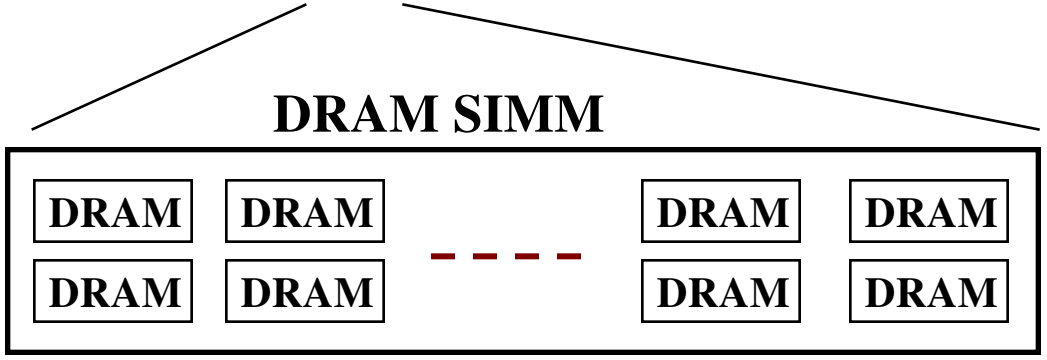
举例：SPARCstation 20's Memory Module

总线宽度是指总线中数据线的条数





存储器总线的总线宽度为128位



每个内存条最多能同时读出128位数据

每次访存操作总是在某一个内存条内进行！

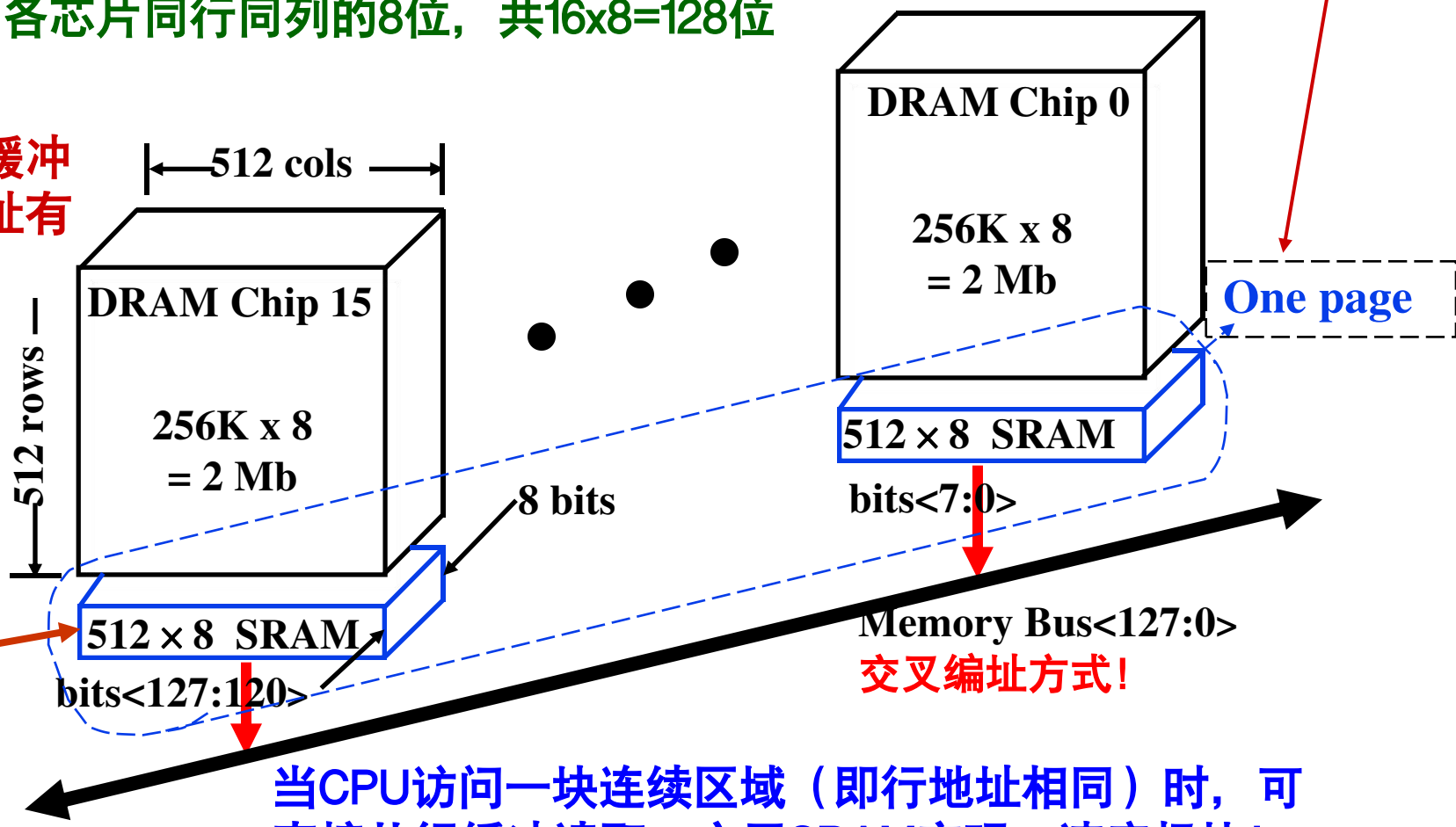
举例： SPARCstation 20’s内存条(模块)

最小配置: 4 MB = 16x 2Mb DRAM chips, 8 KB of Page SRAM
最大配置: 64 MB = 32x 16Mb chips, 16 KB of Page SRAM

最小：2Mb=2^9x2^9x8b=512行x512列并有8个位平面
每次读/写各芯片同行同列的8位，共16x8=128位

问题：行缓冲数据的地址有何特点?

一定在同一行中！即地址是连续的！



当CPU访问一块连续区域（即行地址相同）时，可直接从行缓冲读取，它用SRAM实现，速度极快！

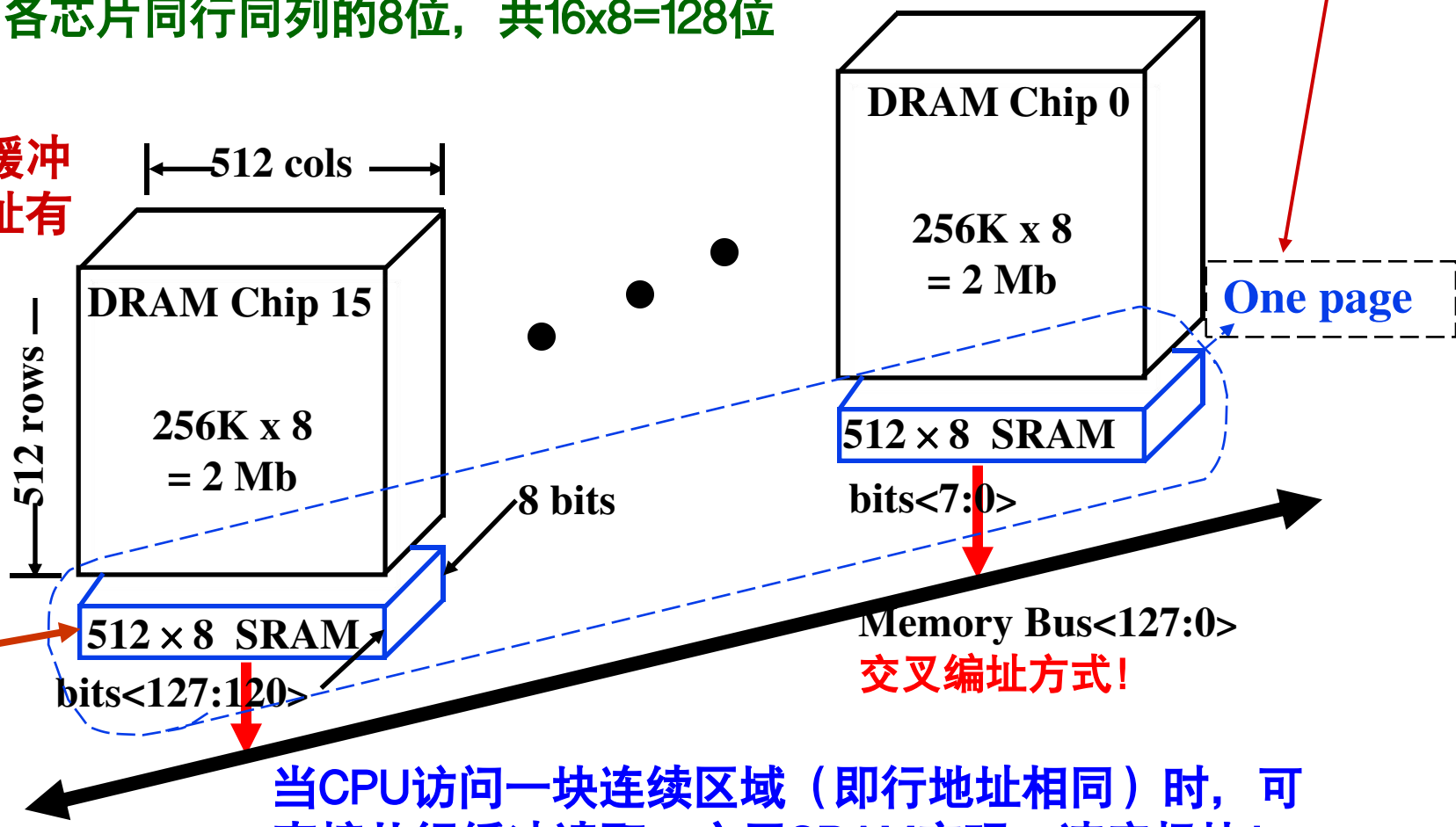
举例： SPARCstation 20’s内存条(模块)

最小配置: 4 MB = 16x 2Mb DRAM chips, 8 KB of Page SRAM
最大配置: 64 MB = 32x 16Mb chips, 16 KB of Page SRAM

最小：2Mb=2^9x2^9x8b=512行x512列并有8个位平面
每次读/写各芯片同行同列的8位，共16x8=128位

问题：行缓冲数据的地址有何特点?

一定在同一行中！即地址是连续的！



当CPU访问一块连续区域（即行地址相同）时，可直接从行缓冲读取，它用SRAM实现，速度极快！

实例：奔腾机的Cache组织

主存：4GB=2^{20}x 2^{7}块x 2^{5}B/块

Cache：8KB=128组x2行/组

替换算法：

LRU，每组一位LRU位

0：下次淘汰第0路

1：下次淘汰第1路

写策略：

默认为Write Back，可动态设置为Write Through。

Cache一致性：

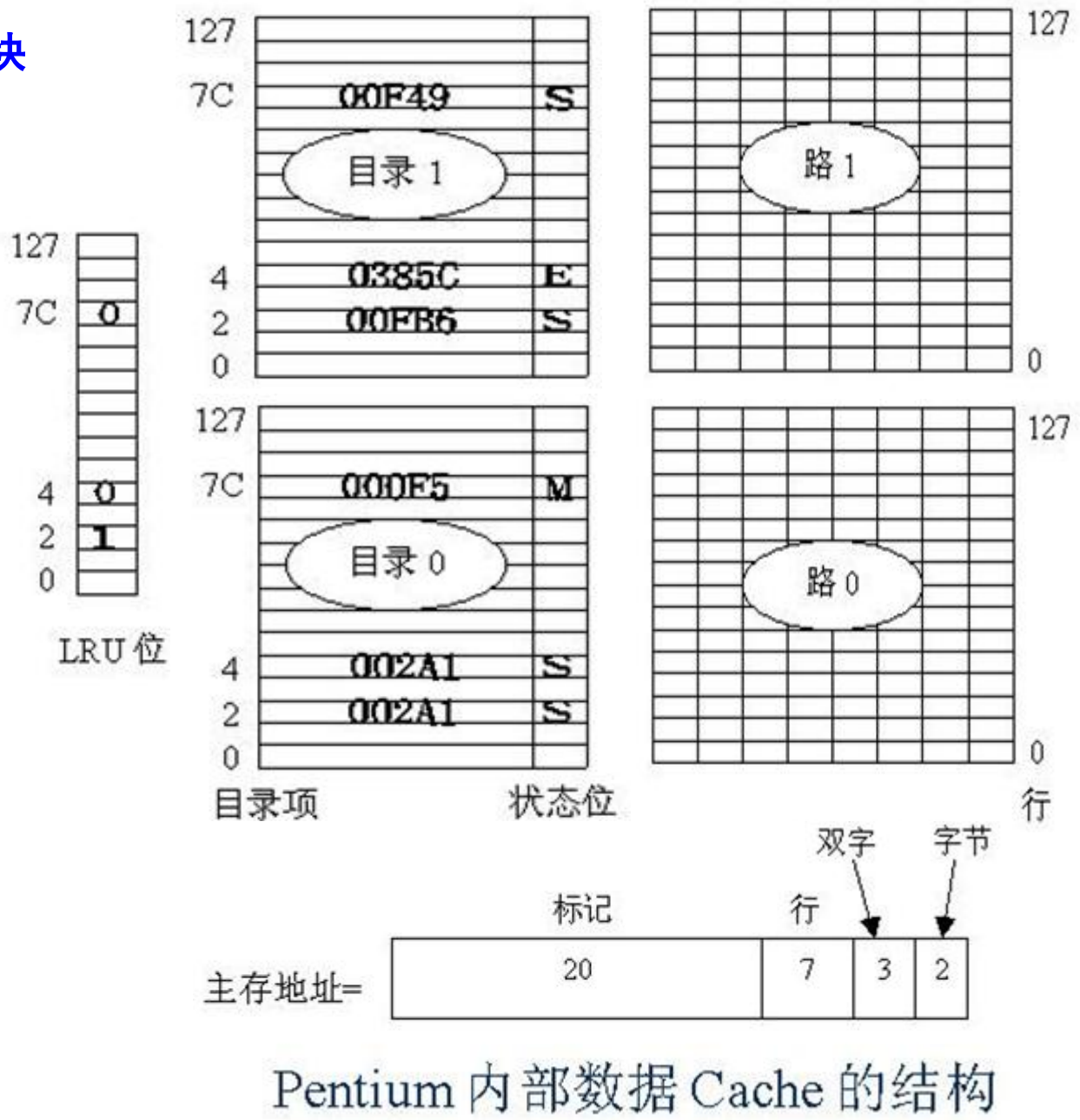
支持MESI协议

M：修改

E：互斥

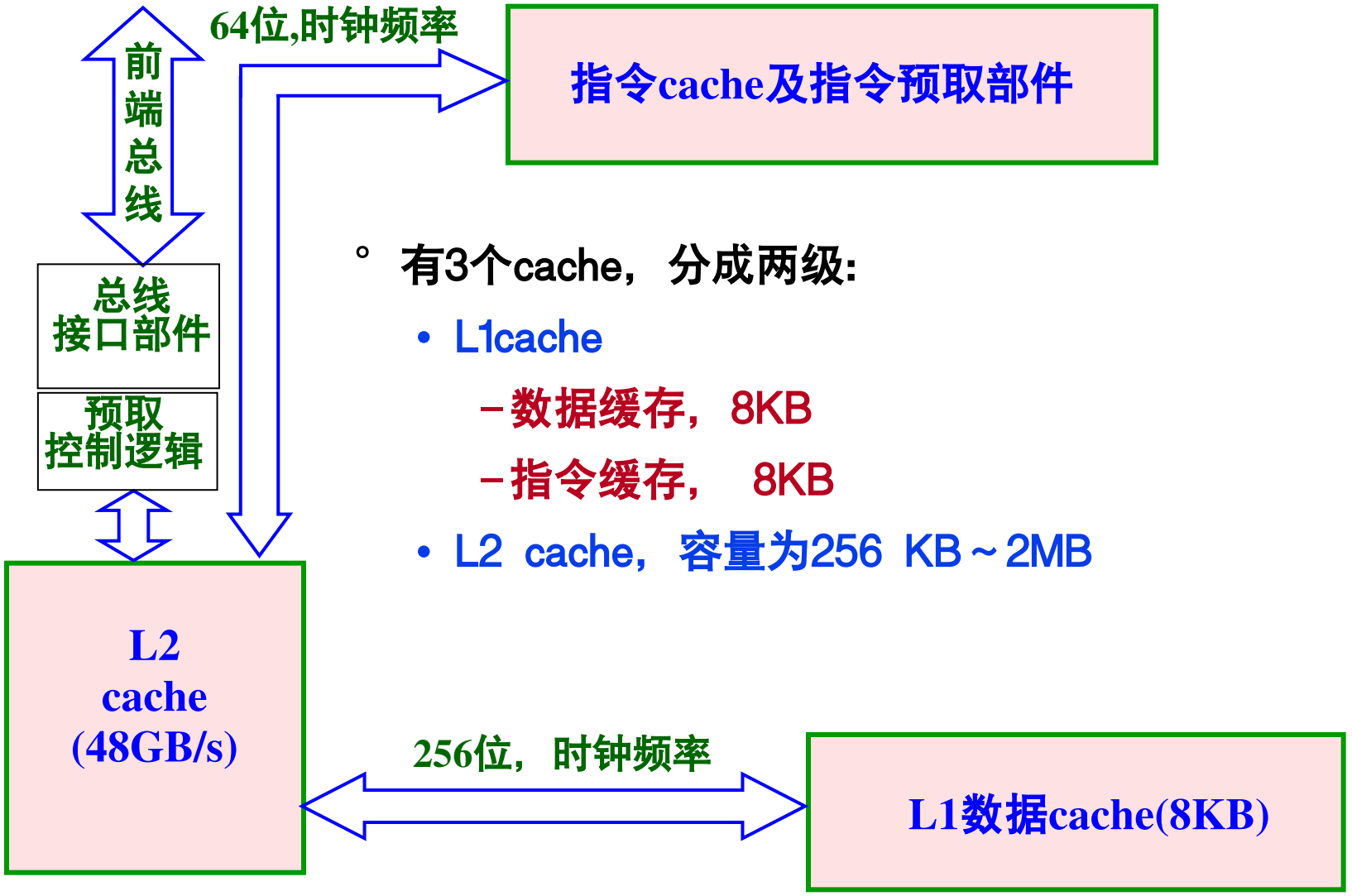
S：共享

I：无效



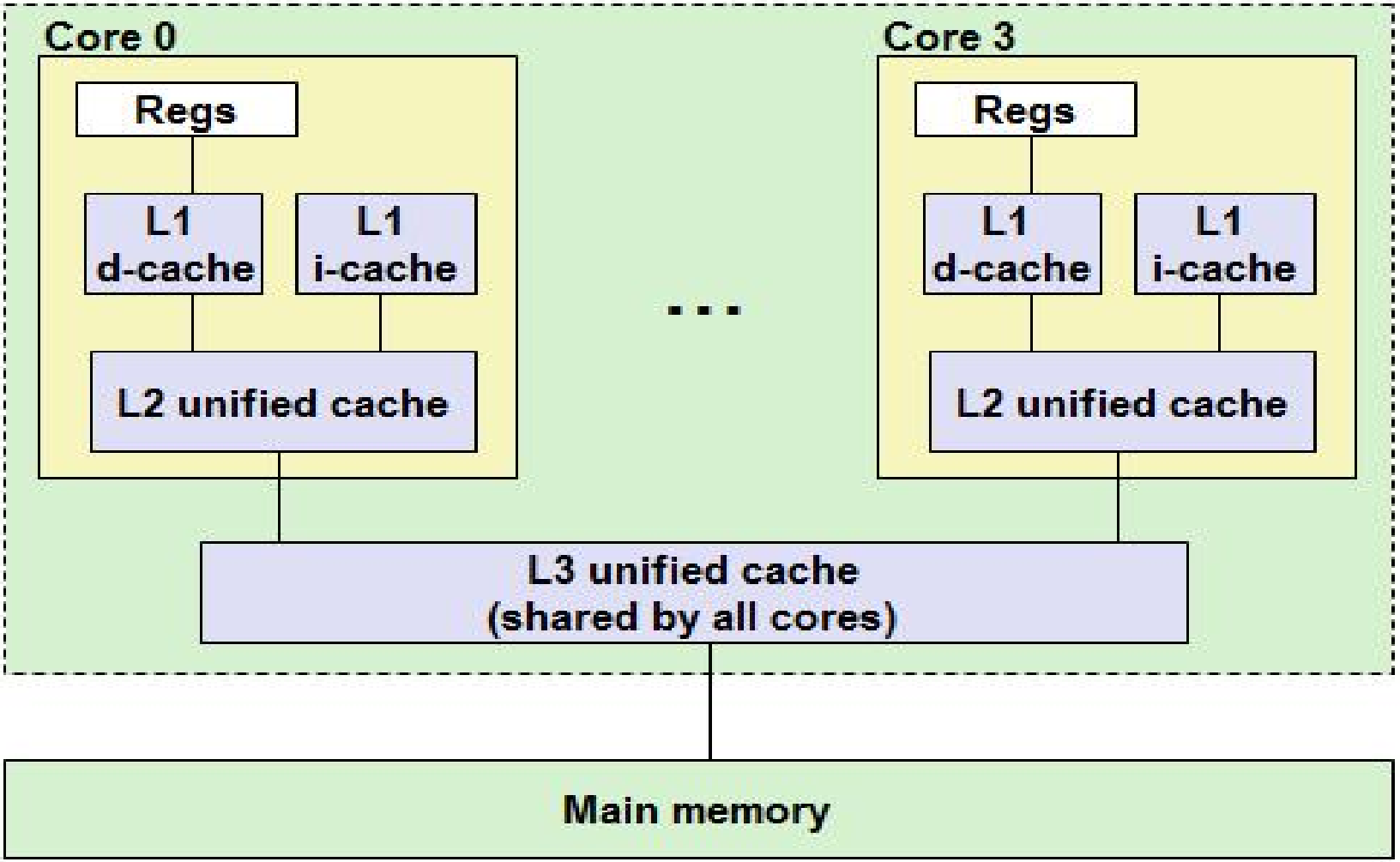
Pentium 内部数据 Cache 的结构

实例： Pentium 4的cache存储器



- 有3个cache，分成两级:
  - L1cache
    - –数据缓存，8KB
    - –指令缓存， 8KB
  - L2 cache，容量为256 KB～2MB

实例： Intel Core i7处理器的cache结构



i-cache和d-cache都是32KB、8路、4个时钟周期；L2 cache：256KB、8路、11个时钟周期。所有核共享的L3 cache：8MB、16路、30~40个时钟周期。Core i7中所有cache的块大小都是64B

缓存在现代计算机中无处不在

Type	What cached	Where cached	Latency (cycles)	Managed by
CPU registers	4-byte word	On-chip CPU registers	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware
L1 cache	32-byte block	On-chip L1 cache	1	Hardware
L2 cache	32-byte block	Off-chip L2 cache	10	Hardware
Virtual memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Cache和程序性能举例

- 举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i，j，sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。

程序 A：

```
int a[256][256];
……
int sum_array1 ( )
{
        int i, j, sum = 0;
        for ( i = 0; i < 256; i++)
                for (j = 0; j < 256; j++)
                        sum += a[i] [j];
        return sum;
}
```

程序 B：

```
int a[256][256];
……
int sum_array2 ( )
{
        int i, j, sum = 0;
        for ( j = 0; j < 256; j++)
                for ( i = 0; i < 256; i++)
                        sum += a[i] [j];
        return sum;
}
```

（1）不考虑用于一致性和替换的控制位，数据cache的总容量为多少？

（2）a[0][31]和a[1][1]各自所在主存块对应的cache行号分别是多少？

（3）程序A和B的数据访问命中率各是多少？哪个程序的执行时间更短？

Cache和程序性能举例

- 举例：某32位机器主存地址空间大小为256 MB，按字节编址。指令cache和数据cache均有8行，主存块为64B，数据cache采用直接映射。假定编译时i，j，sum均分配在寄存器中，数组a按行优先方式存放，其首址为320。

（1）主存地址空间大小为256MB，故主存地址为28位，其中6位为块内地址，3位为cache行号（行索引），标志信息有28−6−3=19位。在不考虑用于cache一致性维护和替换算法的控制位的情况下，数据cache的总容量为：

$8\times(19+1+64\times8)=4256$位$=532$字节 。

（2）a[0][31]的地址为$320+4\times31=444$，[444/64]=6（取整），因此a[0][31]对应的主存块号为6。6 mod 8=6，对应cache行号为6。

或：444=0000 0000 0000 0000 000 110 111100B，中间3位110为行号（行索引），因此，对应的cache行号为6。

a[1][1]对应的cache行号为：

[(320+4×(1×256+1))/64] mod 8=5。

Cache和程序性能举例

a[0][0]所在主存块号为：320/64=5

一个主存块占64B/4B=16个元素

总访问次数为:256x256=64K

总块数(缺失次数)为64Kx4B/64B=4K

缺失率为:4K/64K=1/16

命中率为:1−4K/64K=15/16

° 程序A对数组元素的访问过程:

直接映射

5#: a[0][0], a[0][1], ……, a[0][15] → → → 第5行

6#: a[0][16], a[0][17], ……, a[0][31] → → → 第6行

7#: a[0][32], a[0][33], ……, a[0][47] → → → 第7行

8#: a[0][48], a[0][49], ……, a[0][63] → → → 第0行

……

……, a[255][255]

每块都是第一个不命中，可以仅考虑一个主存块的情况:
第1次不命中，以后15次都命中，故命中率为15/16

（3）A中数组访问顺序与存放顺序相同，共访问64K次，占4K个主存块；首地址位于一个主存块开始，故每个主存块总是第一个元素缺失，其他都命中，共缺失4K次，命中率为：1−4K/64K=93.75%。

方法二：每个主存块的命中情况一样。对于一个主存块，包含16个元素，需访存16次，其中第一次不命中，因而命中率为15/16=93.75%。

Cache和程序性能举例

a[0][0]所在主存块号为：320/64=5

一个主存块占64B/4B=16个元素

每行数组元素占256x4B=1024B
即1024B/64B=16块

a[i][0]和a[i+1][0]之间相差1024B，即16块，因为16 mod 8=0
因此，被映射到cache同一行中！

° 程序B对数组元素的访问过程：

直接映射

5#: a[0][0], a[0][1], ……, a[0][15] → → → 第5行

……

21#: a[1][0], a[1][1], ……, a[1][15] → → → 第5行

……

37#: a[2][0], a[2][1], ……, a[2][15] → → → 第5行

……

……, a[255][255]

访问后面数组元素时，总是把上一次装入到cache中的主存块覆盖掉！

B中访问顺序与存放顺序不同，依次访问的元素分布在相隔256×4=1024的单元处，它们都不在同一个主存块中,cache共8行，一次内循环访问16块，故再次访问同一块时，已被调出cache，因而每次都缺失，命中率为0。