计算机组成原理与接口技术——基于MIPS架构

Mar, 2022

第4讲 存储系统

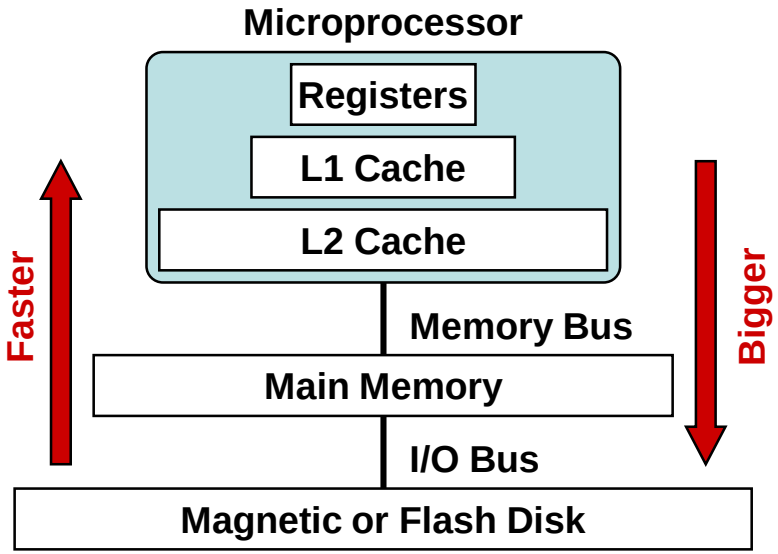
杨明
华中科技大学电信学院
myang@hust.edu.cn

第4讲 存储系统

- 内容
  - 存储器的作用及分类
  - Cache的三种地址映射策略：直接映射、全相联、组相联
  - Cache读策略、写策略、替换策略
  - 内存的三种管理方式：分页式、分段式、段页式
  - 虚拟存储技术
- 目的
  - 了解计算机系统存储系统的分级结构特点
  - 理解Cache的基本概念
  - 掌握Cache的三种映射策略，理解Cache对计算机系统的作用和影响
  - 了解存储器虚拟地址到物理地址映射的机制
  - 了解虚拟存储器的基本原理

4.1 计算机存储系统构成

▶ 存储器：存储程序、数据



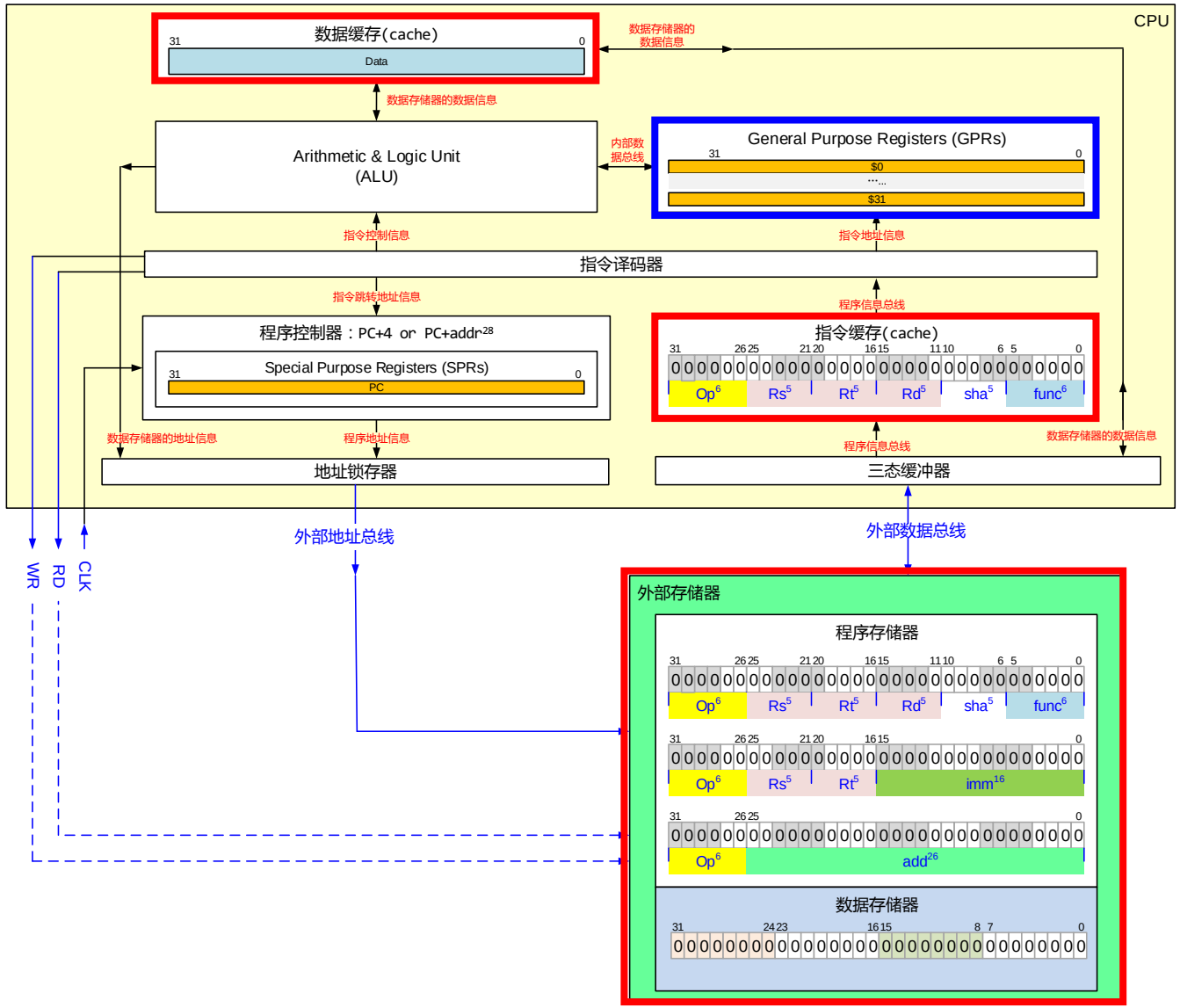
Registers are at the top of the hierarchy
- Typical size < 1 KB
- Access time < 0.5 ns

Level 1 Cache (8 – 64 KB) —— SRAM
- Access time: 1 ns

L2 Cache (512KB – 8MB) —— SRAM
- Access time: 3 – 10 ns

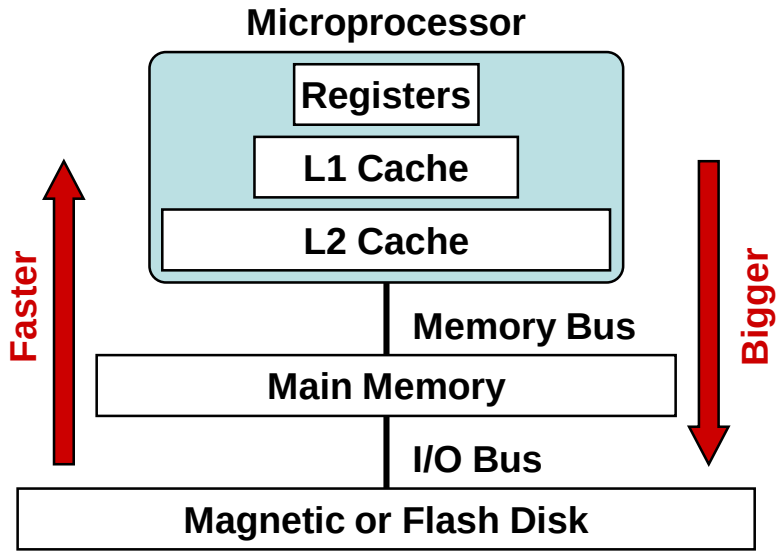
Main Memory (4 – 32 GB) —— DRAM
- Access time: 50 – 100 ns

Disk Storage (> 0.5 TB) —— Flash
- Access time: 5 – 10 ms



4.1 计算机存储系统构成

▶ 存储器：存储程序、数据



Registers are at the top of the hierarchy
- Typical size < 1 KB
- Access time < 0.5 ns

Level 1 Cache (8 – 64 KB) —— SRAM
- Access time: 1 ns

L2 Cache (512KB – 8MB) —— SRAM
- Access time: 3 – 10 ns

Main Memory (4 – 32 GB) —— DRAM
- Access time: 50 – 100 ns

Disk Storage (> 0.5 TB) —— Flash
- Access time: 5 – 10 ms

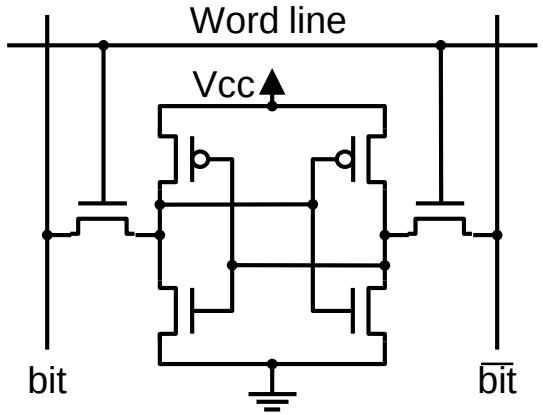
4.1 计算机存储系统构成

- 分类
  - Cache / 主存（内存）/ 外存（读写速度从高到低，容量从小到大）；
  - 存储介质：半导体/磁/光；
  - RAM(Random Access Memory)和ROM(Read Only Memory)
    - RAM：可读可写，断电后信息丢失；存放程序运行中的变量数据。PC系统中的内存、COMS属于RAM；
    - ROM：可读，不能按照普通方法写；断电后信息不丢失；存放程序代码、掉电后需要保存的数据等。BIOS芯片。
  - RAM可进一步分为静态（SRAM）和动态（DRAM）；
  - ROM有ROM/PROM/EPROM/EEPROM；
  - 目前还有能按常规方法(无需紫外线擦、高压写)可读、可写的Flash；

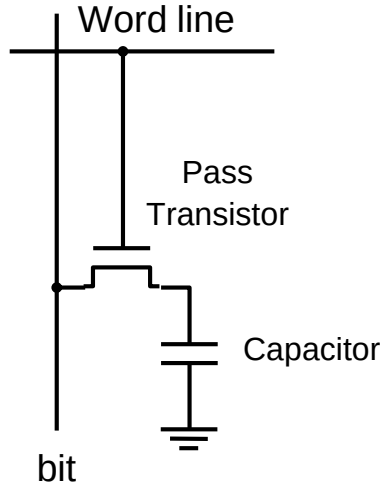
4.1 计算机存储系统构成

► 分类

- RAM可进一步分为静态RAM和动态RAM；
  - SRAM——Static RAM
    - 静态RAM依靠双稳态触发器存储信息，即一个双稳态电路单元存放一位二进制信息，一种稳态为0，另一种稳态为1。只要电源正常就能长期保存信息，不需动态刷新，所以称为静态存储器。静态RAM的速度快，功耗较大，集成度较低，常用于小容量的存储器中。
  - DRAM——Dynamic RAM
    - 动态RAM依靠电容暂存电荷来存储信息，电容充电至高电平为1，放电至低电平为0。由于暂存电荷会逐渐泄漏，需要定期补充电荷来维持为1的存储内容，这种方法称为动态刷新。动态RAM即使在不断电的时候，也必须定时刷新，但这种刷新是自动进行的并不需要使用人员干预。动态RAM功耗较小，集成度较高，但速度稍慢一些。常用来构成容量较大的存储器。
  - DRAM还可分为DRAM/EDO-RAM/SDRAM/DDR-SDRAM



Typical SRAM cell



Typical DRAM cell

4.1 计算机存储系统构成

► 分类

- DRAM还可分为DRAM/EDO-RAM /SDRAM/DDR-SDRAM ；
  - DRAM——Dynamic RAM
    - DRAM没有系统时钟，存取速度慢，其接口多为72线的SIMM类型。已淘汰；
  - EDO-RAM——Extended Data Out RAM
    - EDO RAM同DRAM相似，它取消了扩展数据输出内存与传输内存两个存储周期之间的时间间隔，故而速度比普通DRAM快15-30%。早期的Pentium/K6中使用，已淘汰。
  - SDRAM——Synchronized Dynamic RAM
    - 利用一个单一的系统时钟同步所有的地址数据和控制信号。使用SDRAM不但能提高系统表现，还能简化设计、提供高速的数据传输。访问速度最大可达到133MHz。810/815(E/P)/KT133(A)等芯片组搭配的内存都为SDRAM。
  - DDR-SDRAM——Double Data Rate SDRAM
    - 现代PC中的CPU处理速度越来越快，对内存的要求也更高。若内存的访问速度不能提高，CPU速度再快，整个系统的性能也会受到影响。
    - DDR，是“双倍速率SDRAM”的意思，它在时钟的上、下沿都能进行数据传输，其性能为SDRAM的两倍，目前主流PC的内存都是DDR型的。

4.1 计算机存储系统构成

- 分类
  - ROM有ROM/PROM/EPROM/EEPROM；
    - ROM中的信息只能被读出，而不能被操作者修改或删除，故一般用于存放固定的程序，如监控程序、汇编程序等，以及存放各种表格，通常为OTP(One Time Programmable)型，即只能一次性编程，不能重复使用；若要重复使用，需用PROM：EPROM / EEPROM

存储器	擦除/写入方式	主要用途
SRAM	在系统	数据与程序运行
DRAM	在系统	数据与程序运行
ROM	只读	程序
OTP	一次性编程	程序
EPROM	需要紫外灯与编程器	程序
EEPROM	需要编程器（加高压）	程序
FLASH	常规电压/在系统	程序与数据

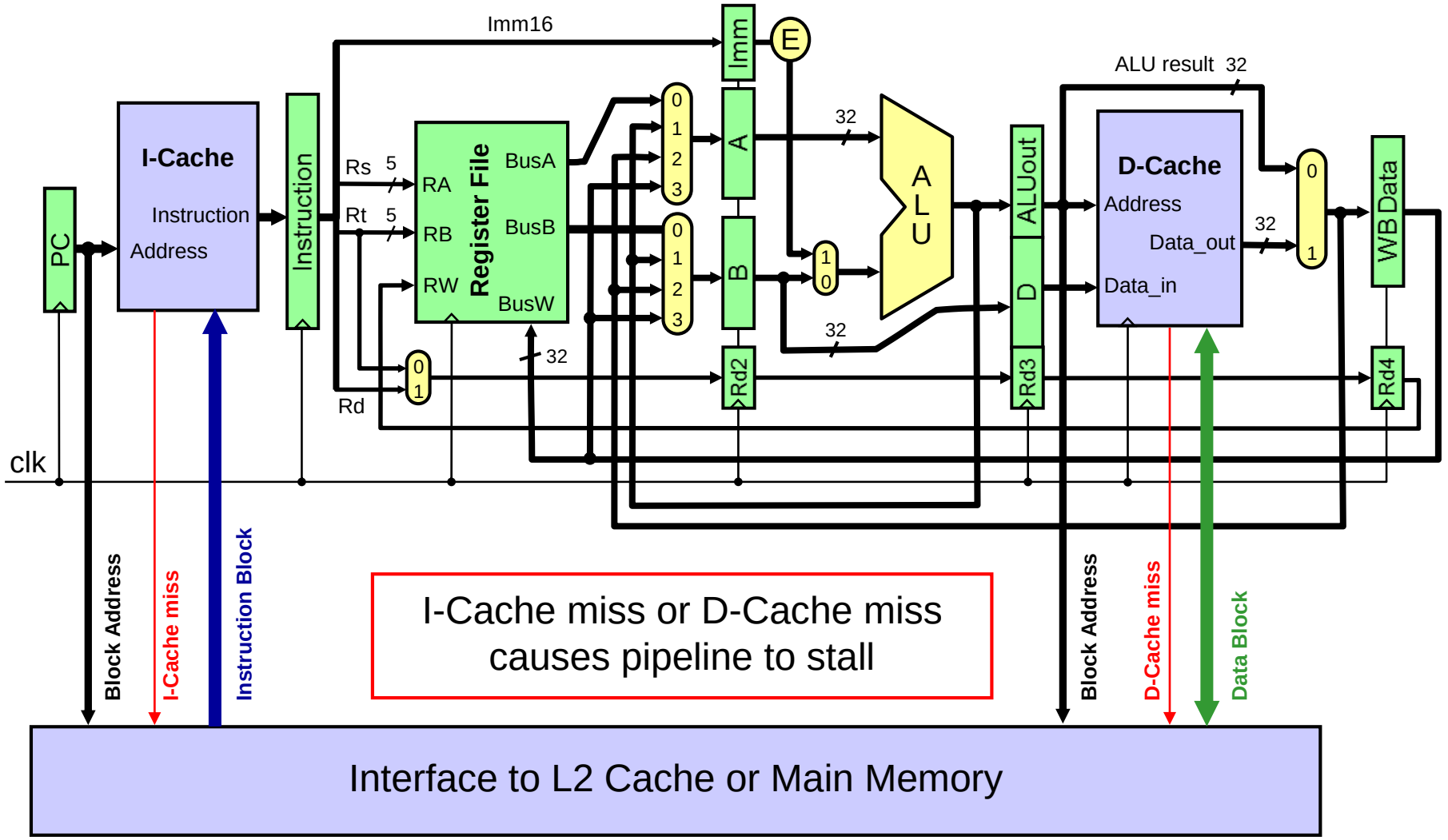
    - EPROM（Erasable Programmable ROM）：可擦除可编程ROM，擦除内部信息需要用紫外灯照射；写时需要专门编程器；
    - EEPROM（Electrically Erasable Programmable ROM）：电可擦除可编程ROM ，擦除、写入都需要专门编程器；
  - Flash
    - 又称闪存，由于擦除、写入无需额外的高电压；支持在线、在系统编程；可重复擦、写万次以上；数据至少可以保持10年；擦写速度较快(几个ms内)等特性，在在数码相机、USB接口闪盘等电子产品中大显身手，其透人的优异性能得到了广大用户的青睐。

第4讲 存储系统

- 内容
  - 存储器的作用及分类
  - Cache的三种地址映射策略：直接映射、全相联、组相联
  - Cache读策略、写策略、替换策略
  - 内存的三种管理方式：分页式、分段式、段页式
  - 虚拟存储技术
- 目的
  - 了解计算机系统存储系统的分级结构特点
  - 理解Cache的基本概念
  - 掌握Cache的三种映射策略，理解Cache对计算机系统的作用和影响
  - 了解存储器虚拟地址到物理地址映射的机制
  - 了解虚拟存储器的基本原理

4.2 高速缓存(cache)原理

- Cache：CPU内部的存储器，临时存放那些近期需要运行的指令和数据



4.4 高速缓存(cache)原理

► 程序访问内存局部性原理

- 程序访问的时间局部性：对同一存储空间重复访问
  - 对大量典型程序运行情况的分析结果表明，在一个较短的时间间隔内，程序产生的地址往往集中在存储器逻辑地址空间很小的范围内。指令地址的分布本来就是连续的，再加上循环程序段和子程序段要重复执行多次。因此，对这些地址的访问就自然地具有时间上集中分布的倾向
- 程序访问的空间局部性：对相邻存储空间连续访问
  - 数据分布的集中倾向不如指令明显，但对数组的存储和访问以及工作单元的选择都可以使存储器地址相对集中。对局部范围的存储器地址频繁访问，而对此范围以外的地址则访问甚少。

程序段A（列优先访问）：

```
assign-array-rows()
{
    int       i, j;
    short     a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
            a[i][j] = 0;
}
```

程序段B（行优先访问）：

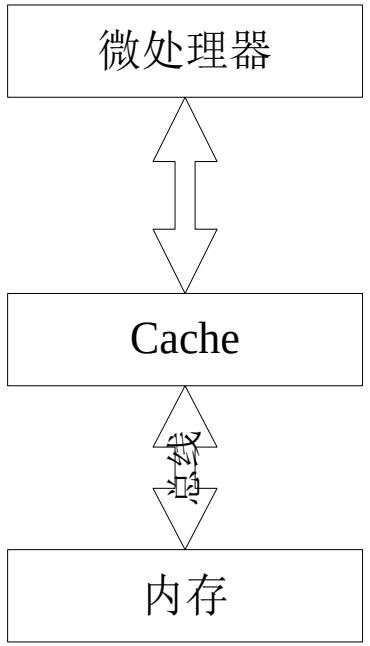
```
assign-array-cols()
{
    int       i, j;
    short     a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
            a[i][j] = 0;
}
```

这两个程序段性能有差别吗？

4.2 高速缓存(cache)原理

基本概念

- Cache命中（Hit）
  - CPU访问存储器时在地址线上输出存储器的地址信息，Cache控制器将判断该地址是否与Cache中存放数据的地址一致。若一致，则命中。
- Cache非命中（Miss）
  - 不一致
  - 没有命中的数据，CPU只好直接从内存获取。获取的同时，也把它拷进Cache，以便下次访问。
- Hit时，CPU从Cache中读取数据，不需要等待，效率高
- Miss时，CPU只能到低速的主存中去获取数据
- 命中率=命中cache次数/访问cache总次数
- 一般规定Cache与内存的空间比为4:1000，即128kB Cache可以映射32MB内存，命中率可在90%以上。



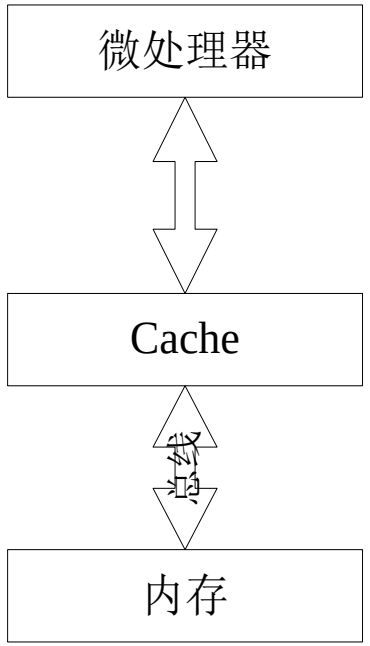
4.2 高速缓存(cache)原理

▶ Cache构成原理

- Cache是CPU内部的存储器，临时存放那些近期需要运行的指令和数据，以提高CPU对主存的访问速度

- Cache容量小于内存容量，需要在cache存储单元和内存存储单元之间建立某种地址映像关系

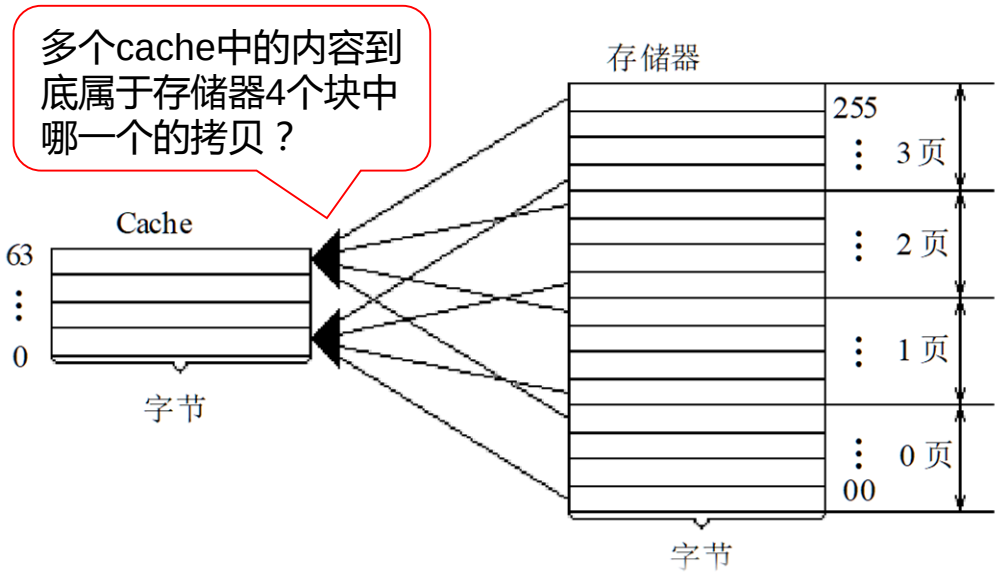
- 地址映射：某一数据在内存中的地址与在Cache中的地址两者之间的对应关系，常见的映射方法:
  - 直接映像（Direct Mapped）
  - 全相联（Full Associative）
  - 组相联（Set Associative）

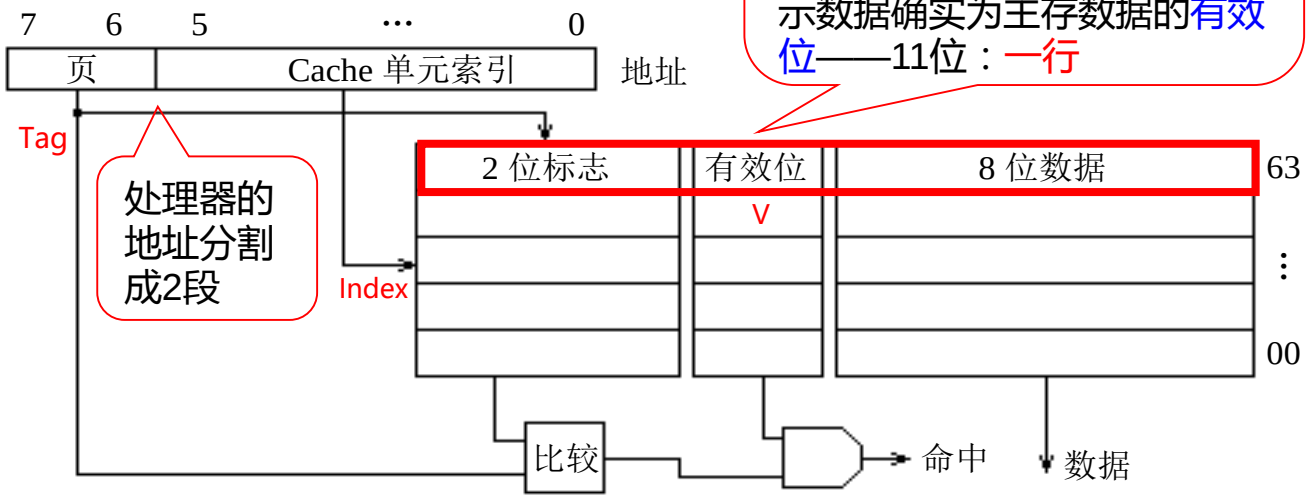


4.2 高速缓存(cache)原理

▶ Cache构成原理

- 直接映像（Direct Mapped）
  - 主存与缓存分成大小相同的数据行（如2^n个字节）。
  - 主存容量是缓存容量的整数倍，将主存空间按缓存的容量分成页(块)，主存中每一页的行数与缓存的总行数相等。
  - 主存中某页的一行存入缓存时只能存入缓存中行编号相同的位置。



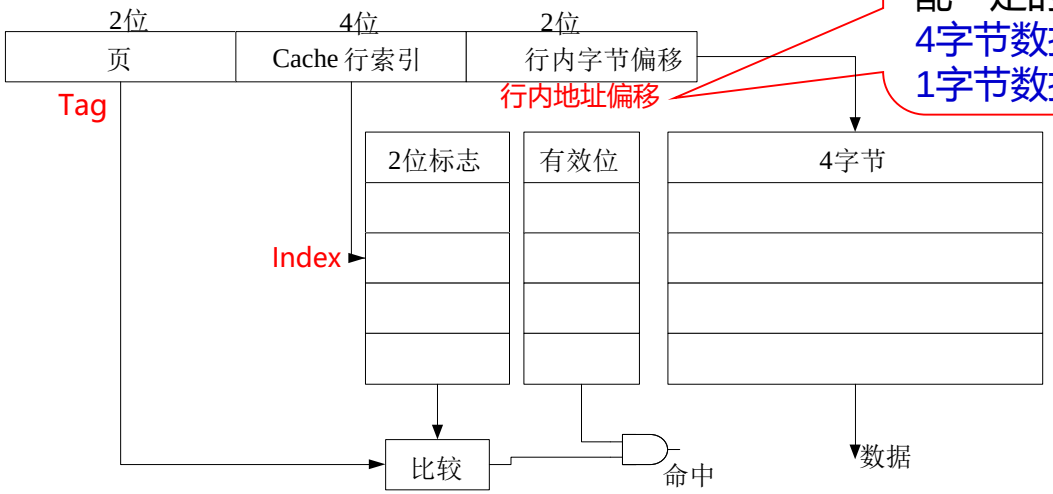


直接映射Cache结构原理框图

4.2 高速缓存(cache)原理

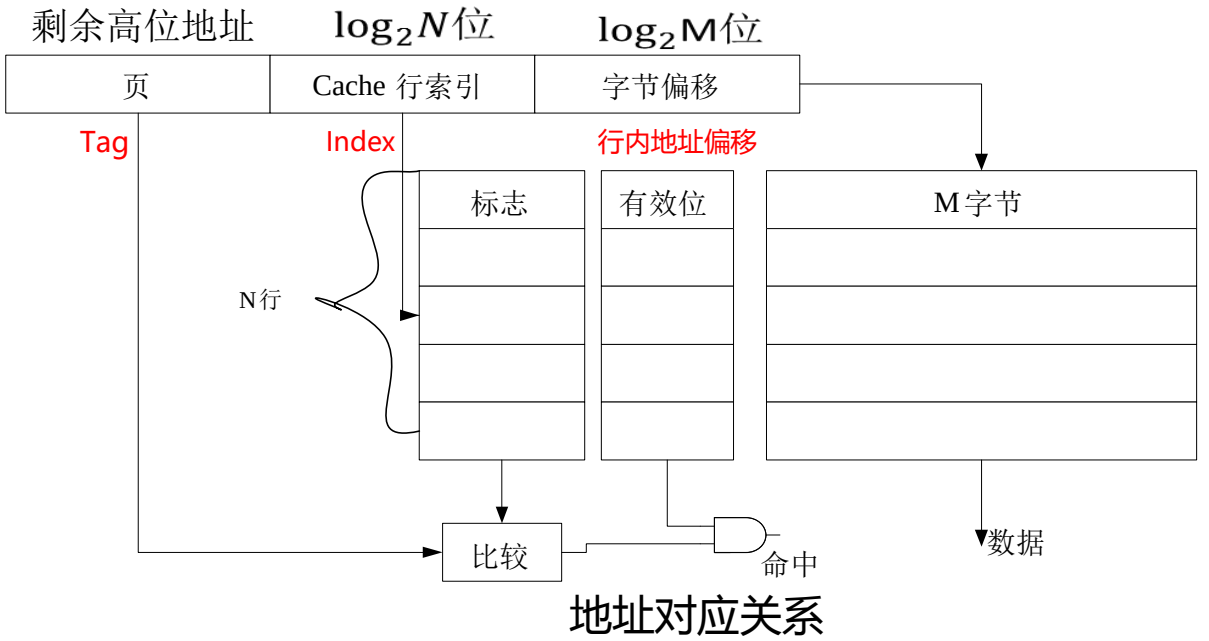
▶ Cache构成原理

- 直接映像（Direct Mapped）



实际中，Cache中一行的数据块大小远不止1Byte，需要在地址分配中分配一定的位来指示行内字节偏移：
4字节数据——地址偏移2位
1字节数据——地址偏移0位

处理器地址的分割策略：受一行数据块大小M、一页总行数N和地址总线宽度的限制。



地址对应关系

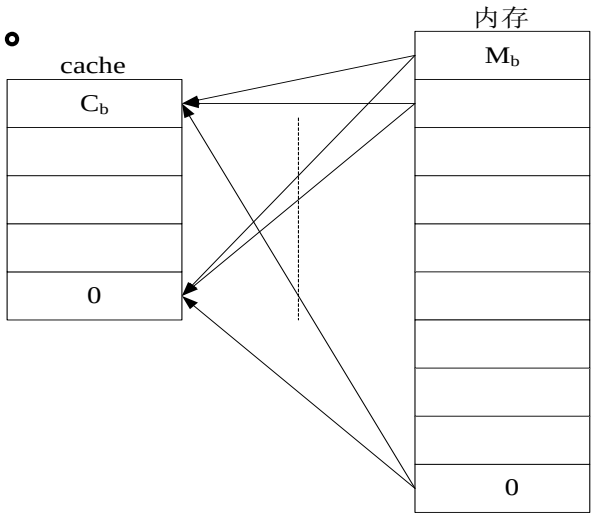
4字节块组织的直接映射Cache结构原理框

- 优点
  - 访问速度快、硬件实现简单
- 缺点
  - 多个不同的页中处于同样行位置的数据访问比较频繁时，需要不停的更换同一个cache行的内容，cache替换操作频繁，命中率比较低。

4.2 高速缓存(cache)原理

▶ Cache构成原理

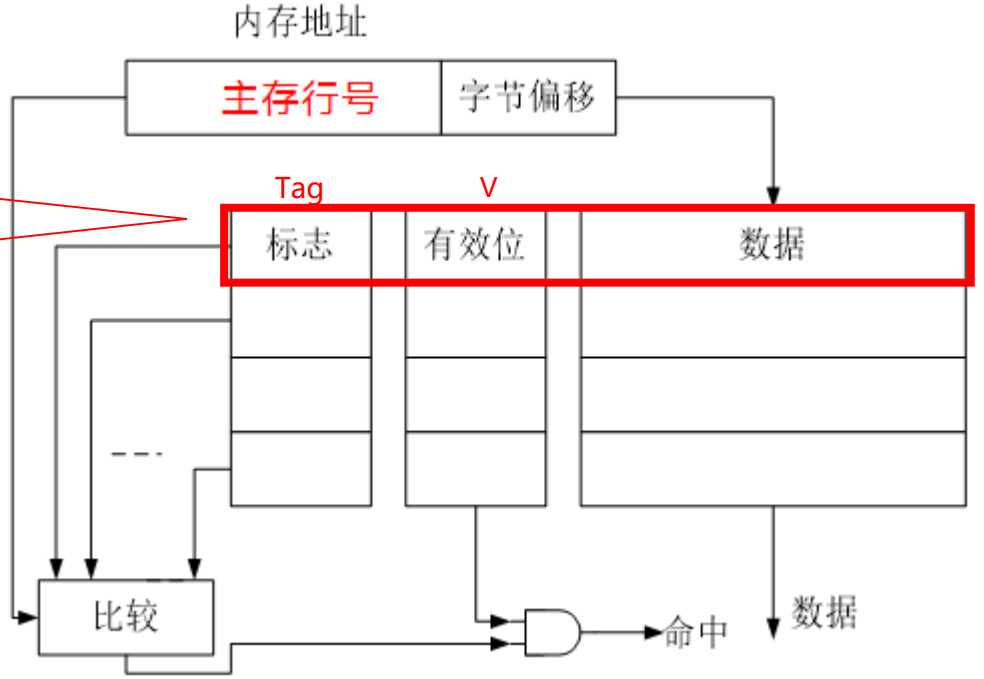
- 全相联（Full Associative）
  - 主存与缓存分成相同大小的行。
  - 主存的某一数据行可以装入缓存的任意一行空间中。



  - 优点
    - 实现任意映像，Cache行的利用率高，行冲突的概率低，命中率高
  - 缺点
    - 访问效率低，速度慢，硬件成本高

全相联方式下，一个Cache行不需要Index!

- 全相联映像cache结构原理



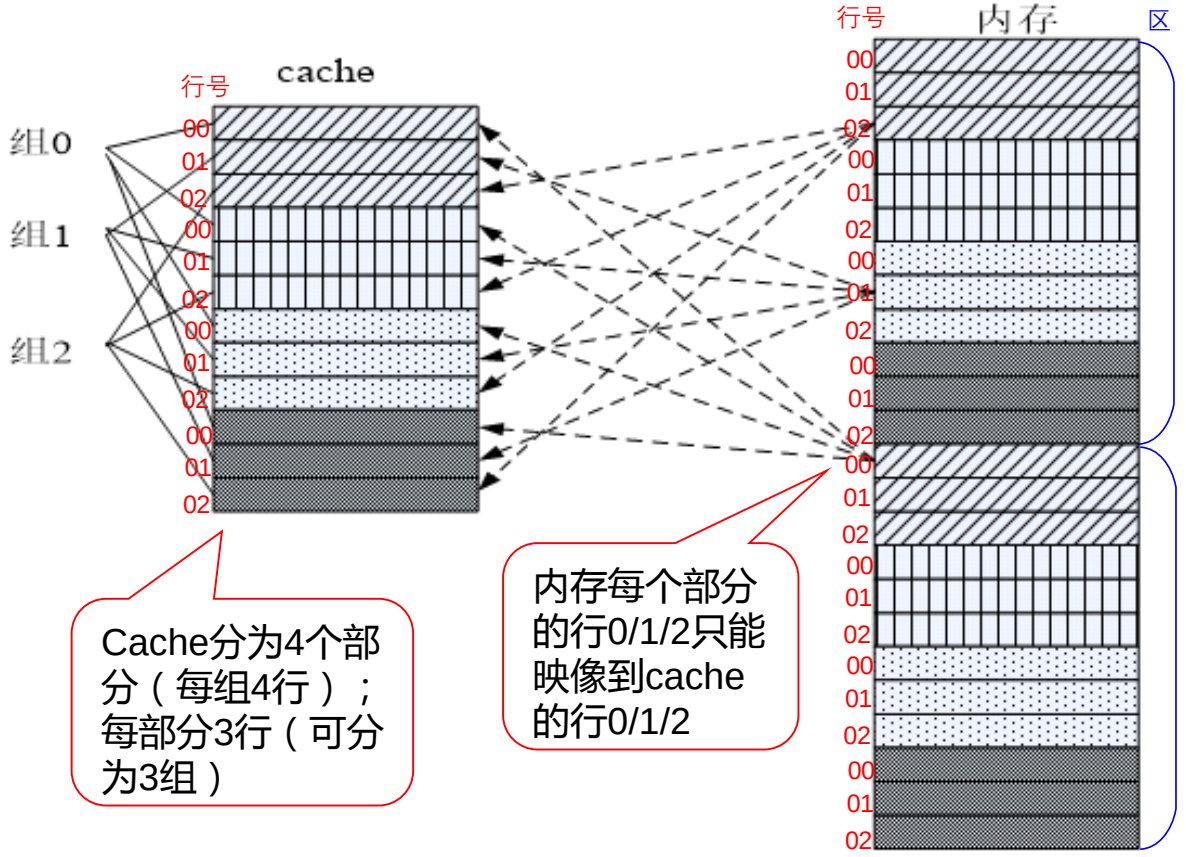
CPU如何找到内存的某行映射到了哪个Cache行？——必须搜索整个Cache，只有主存行号相一致且有效位为1才命中。

4.2 高速缓存(cache)原理

▶ Cache构成原理

- 组相联（Set Associative）
  - 内存和缓存按同样大小分行
  - 内存和缓存按同样大小分块，常为 2^m 行
  - 内存和缓存各块中的行都从0 开始编号，且缓存各块中编号相同的行构成组，行号即为组号
  - 当内存数据调入缓存时，内存各块中的某行只能存入缓存中组号相同的行内，但可以存放到组内任意行。即从内存的行到缓存的组之间采用直接映射方式，在缓存组内采用全相联映射方式

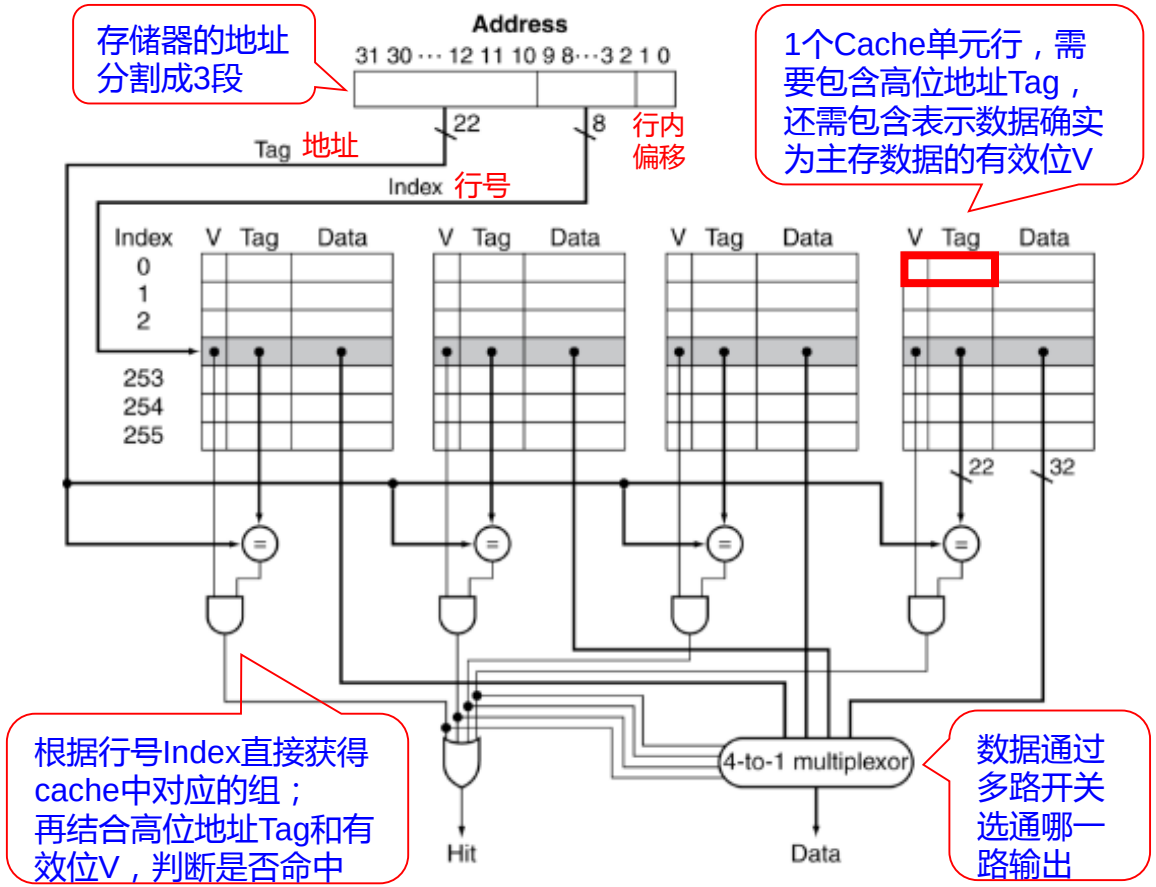
- 3组，每组4行cache组相联



4.2 高速缓存(cache)原理

▶ Cache构成原理

- 组相联（Set Associative）
  - 4路组相联映像cache结构原理

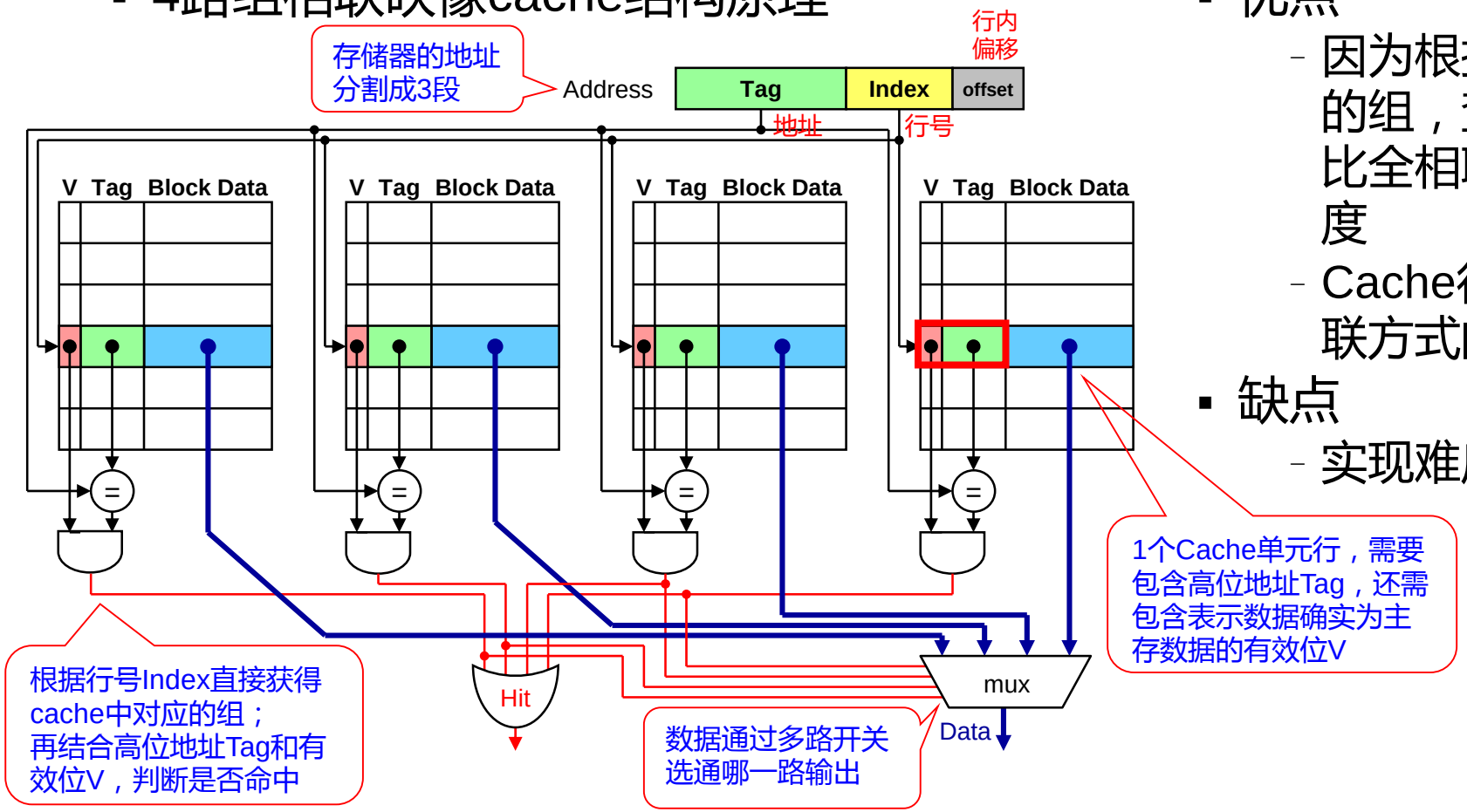


- 优点
  - 因为根据Index直接获得Cache中对应的组，查找就只限定在对应的组内，相比全相联方式，加快了Cache的访问速度
  - Cache行的利用率高、命中率比直接相联方式的高
- 缺点
  - 实现难度比直接相联方式的高

4.2 高速缓存(cache)原理

Cache构成原理

- 组相联（Set Associative）
  - 4路组相联映像cache结构原理



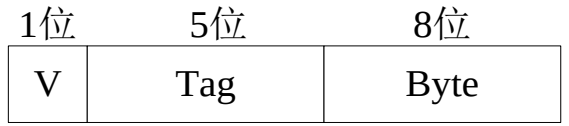
- 优点
  - 因为根据Index直接获得Cache中对应的组，查找就只限定在对应的组内，相比全相联方式，加快了Cache的访问速度
  - Cache行的利用率高、命中率比直接相联方式的高
- 缺点
  - 实现难度比直接相联方式的高

4.2 高速缓存(cache)原理

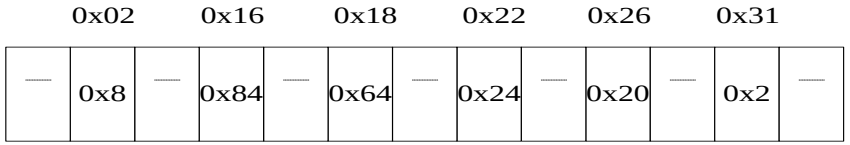
► Cache读策略

- Cache的读写操作实际上在读写主存储器时发生，涉及CPU、主存、Cache三者的协调。
  - CPU从主存储器读数据时，Cache控制器会判断其地址是否定位在Cache中，如果在（命中），CPU的数据就会从Cache读取数据；否则从主存储器读数据，同时进行行填充。
- Cache行填充（Line Fill）
  - 当CPU所完整的Cache行复制到Cache中
  - 需的数据或代码不在Cache中而出现非命中时，Cache控制器就必须在主存储器中读取数据
  - 当CPU由主存储器读入数据时，同时还要将该数据拷贝到Cache中
    - 即使当前CPU仅读取一个字节，Cache控制器总是要将主存储器中包含该字节的一个完整的行复制到Cache中——从主存储器向Cache传送一行数据的操作就称为Cache行填充（Line Fill）

- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据



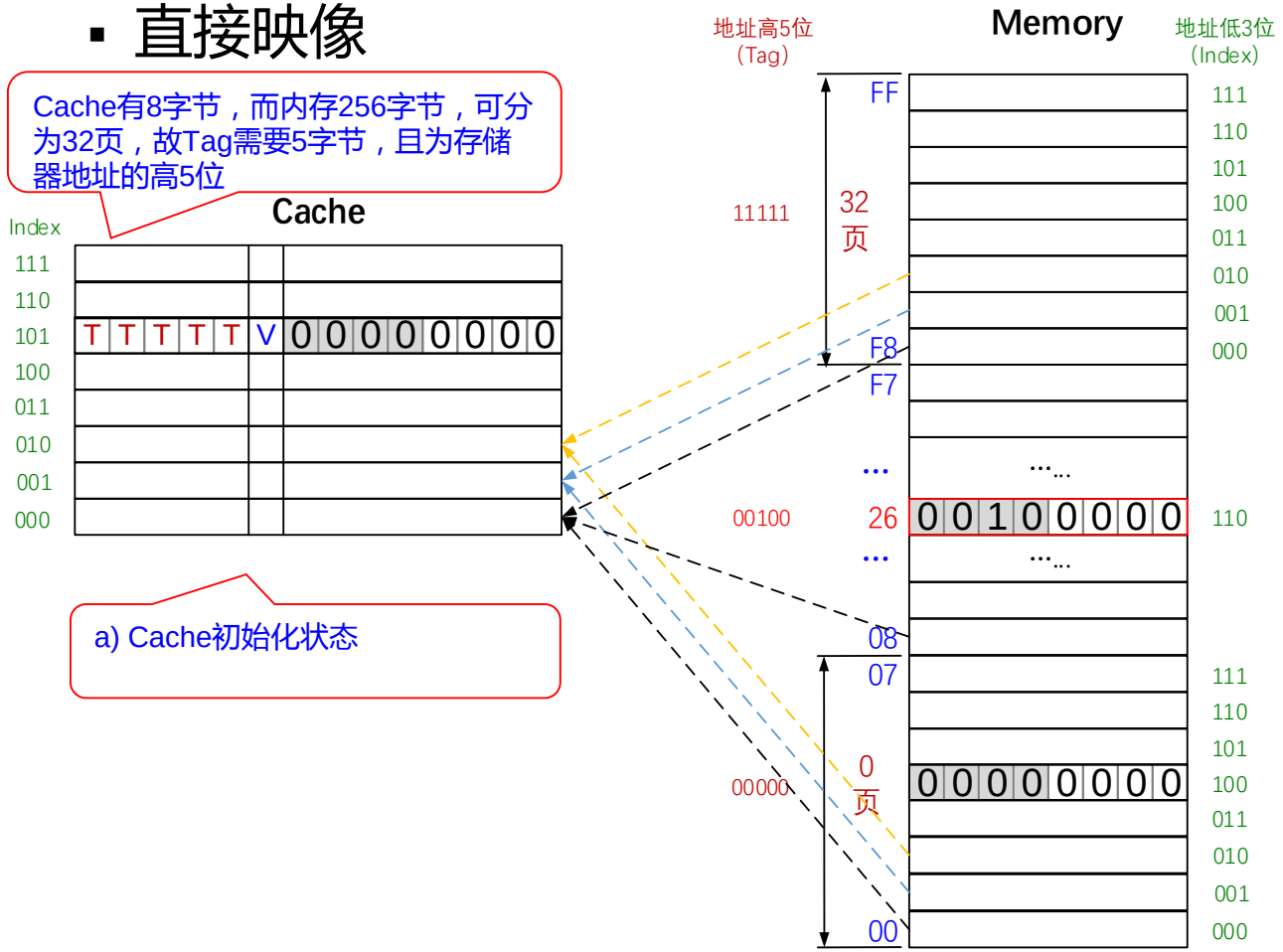
  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

▶ Cache读策略

• Cache行填充（Line Fill）

▪ 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构

1位	5位	8位
V	Tag	Byte

  - 内存中一段区域的初始数据

0x02	0x16	0x18	0x22	0x26	0x31
— 0x8 —	0x84 —	0x64 —	0x24 —	0x20 —	0x2 —

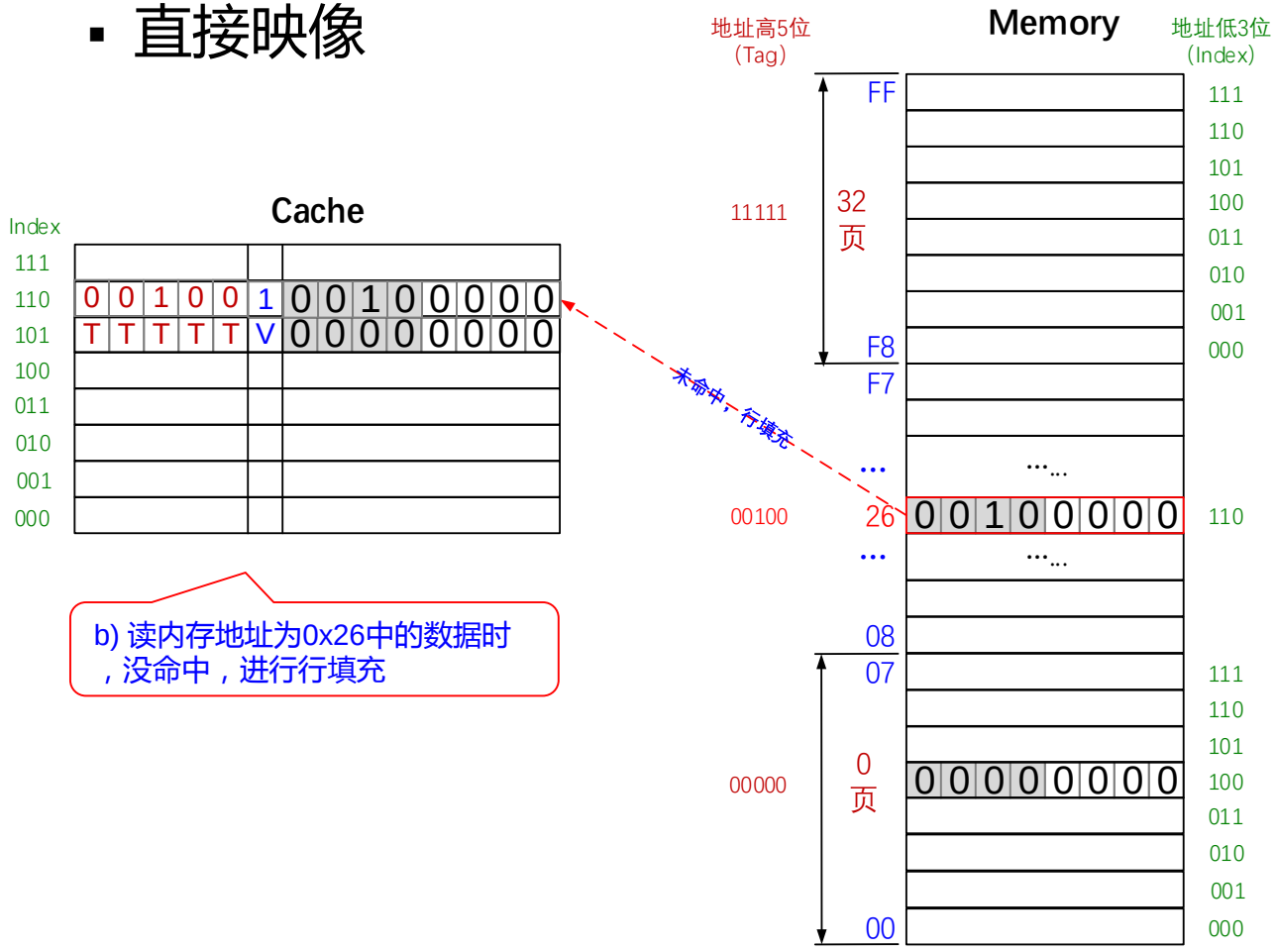
  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

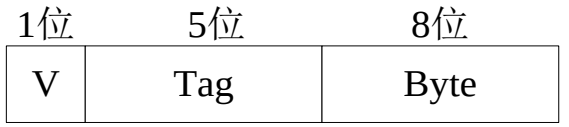
► Cache读策略

• Cache行填充（Line Fill）

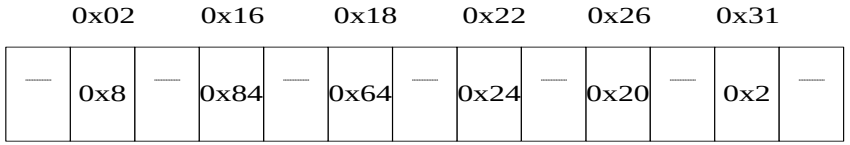
▪ 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - ➢ 内存中一段区域的初始数据

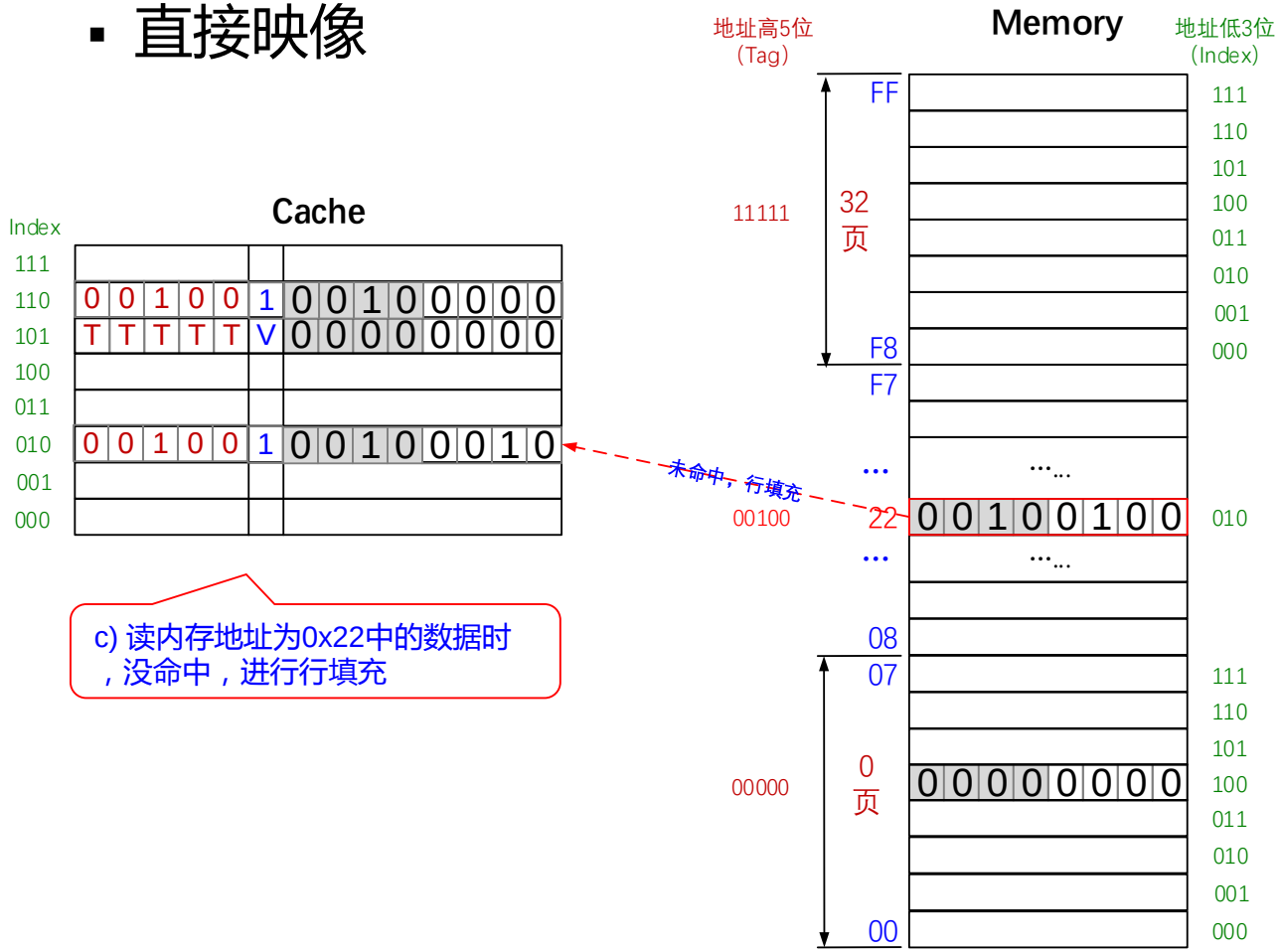


  - ➢ 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

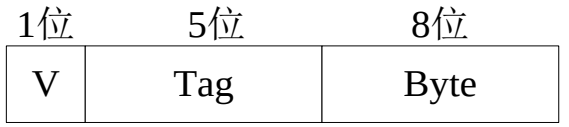
4.2 高速缓存(cache)原理

▶ Cache读策略

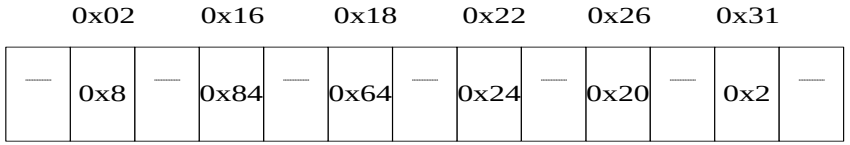
- Cache行填充（Line Fill）
  - 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据



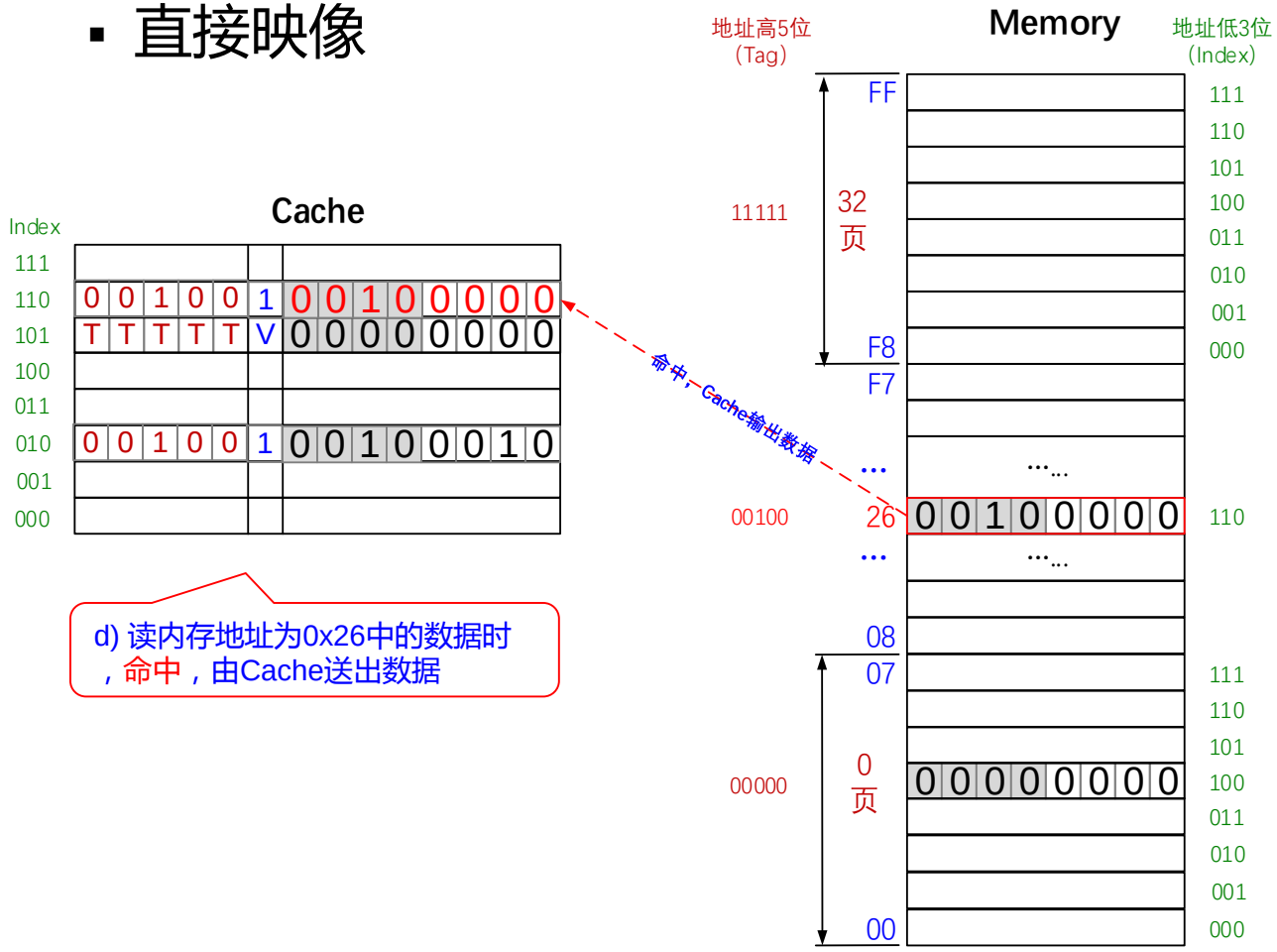
  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

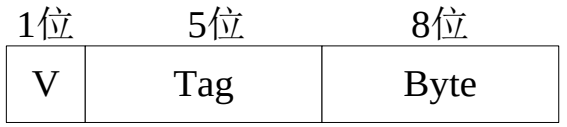
► Cache读策略

• Cache行填充（Line Fill）

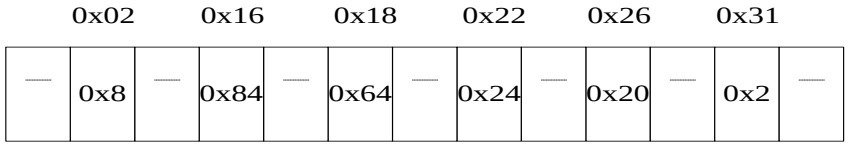
▪ 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据

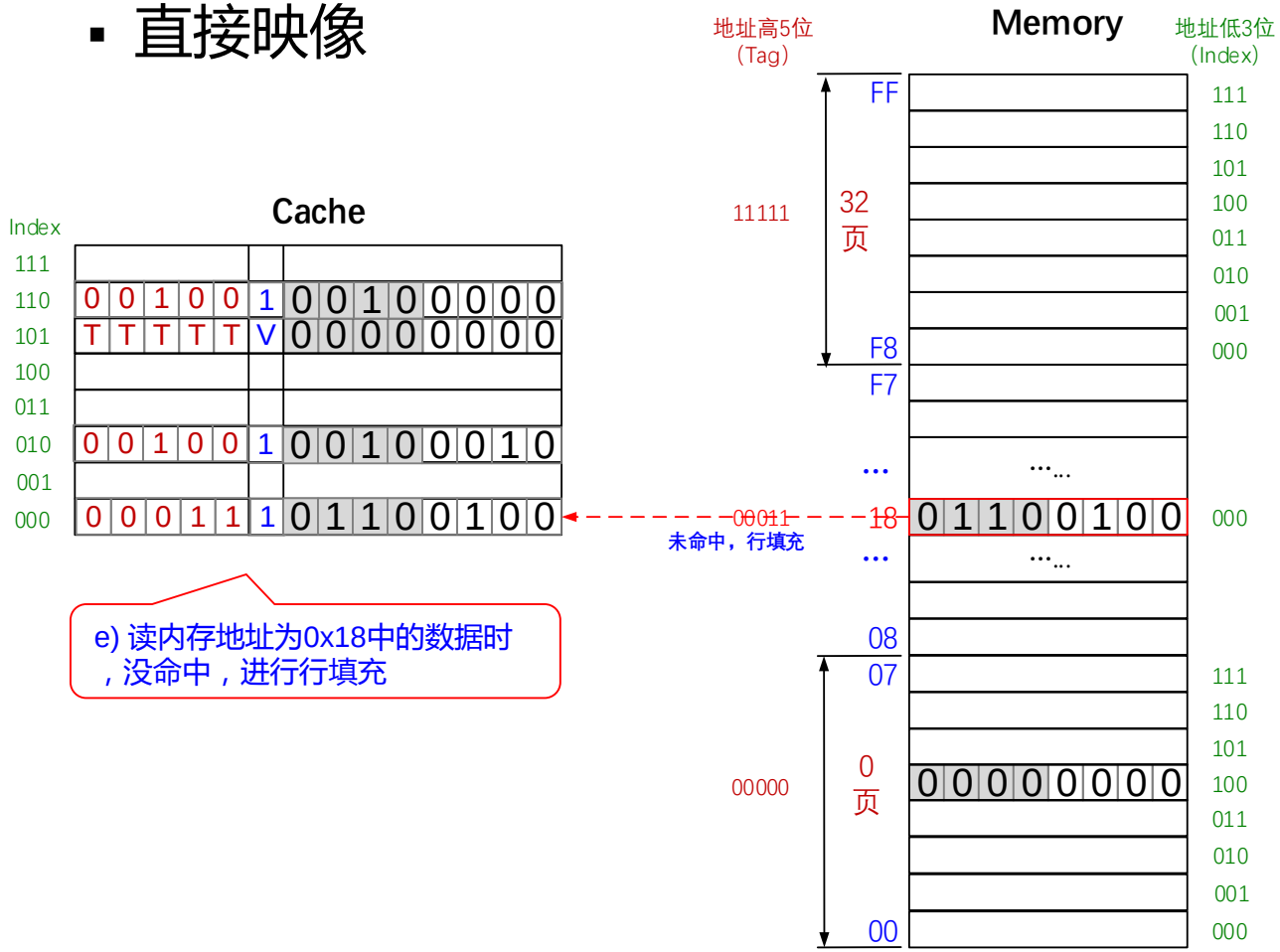


  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

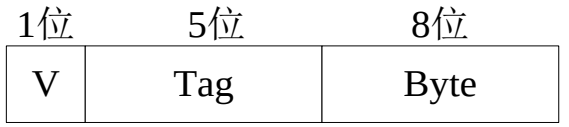
4.2 高速缓存(cache)原理

▶ Cache读策略

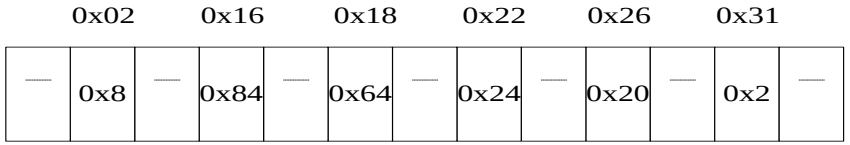
- Cache行填充（Line Fill）
  - 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - ➢ 内存中一段区域的初始数据



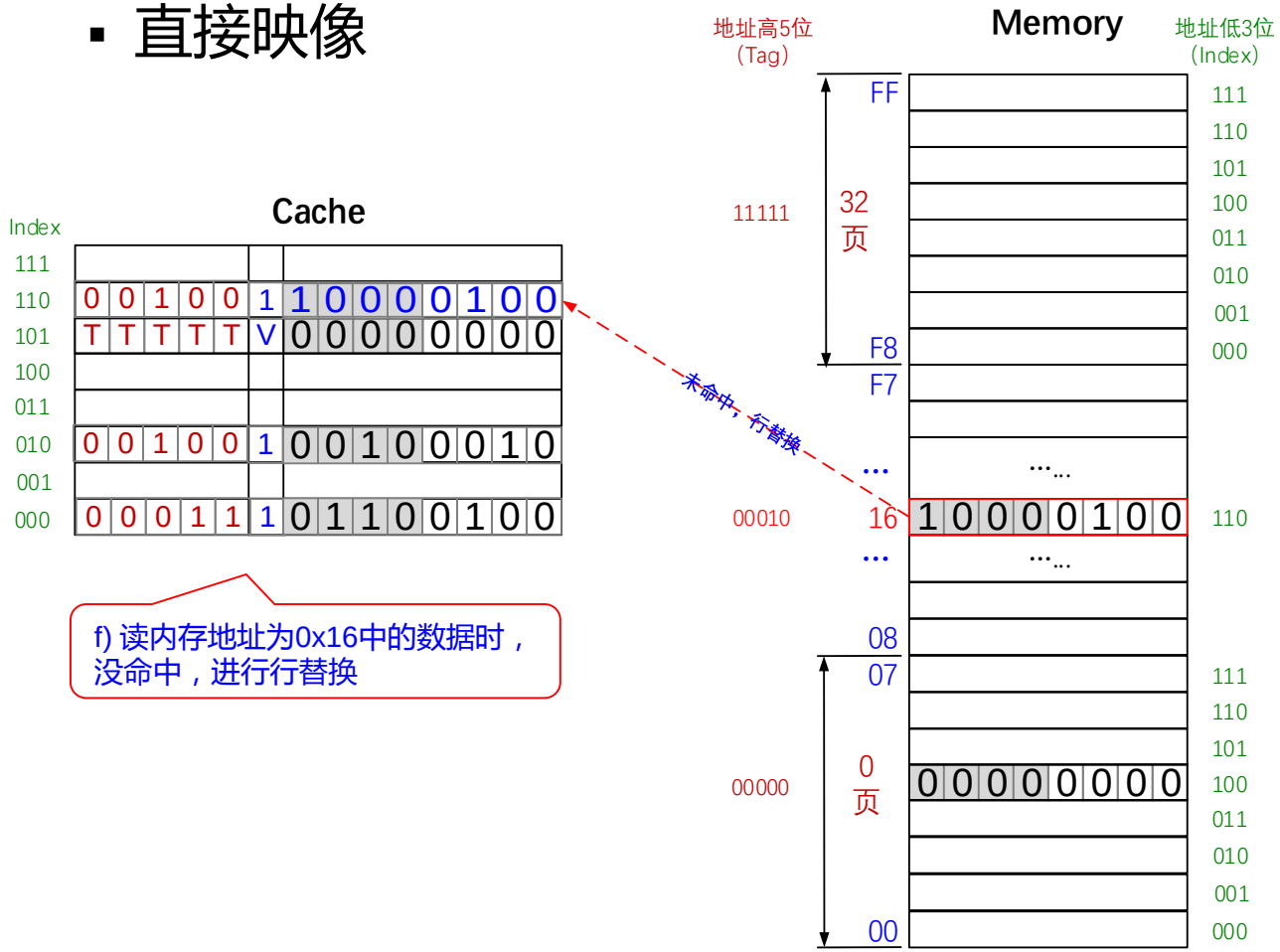
  - ➢ 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

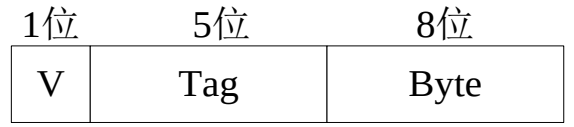
▶ Cache读策略

• Cache行填充（Line Fill）

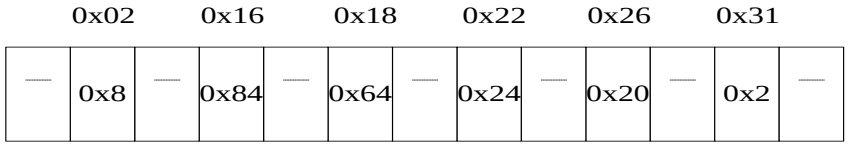
▪ 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据

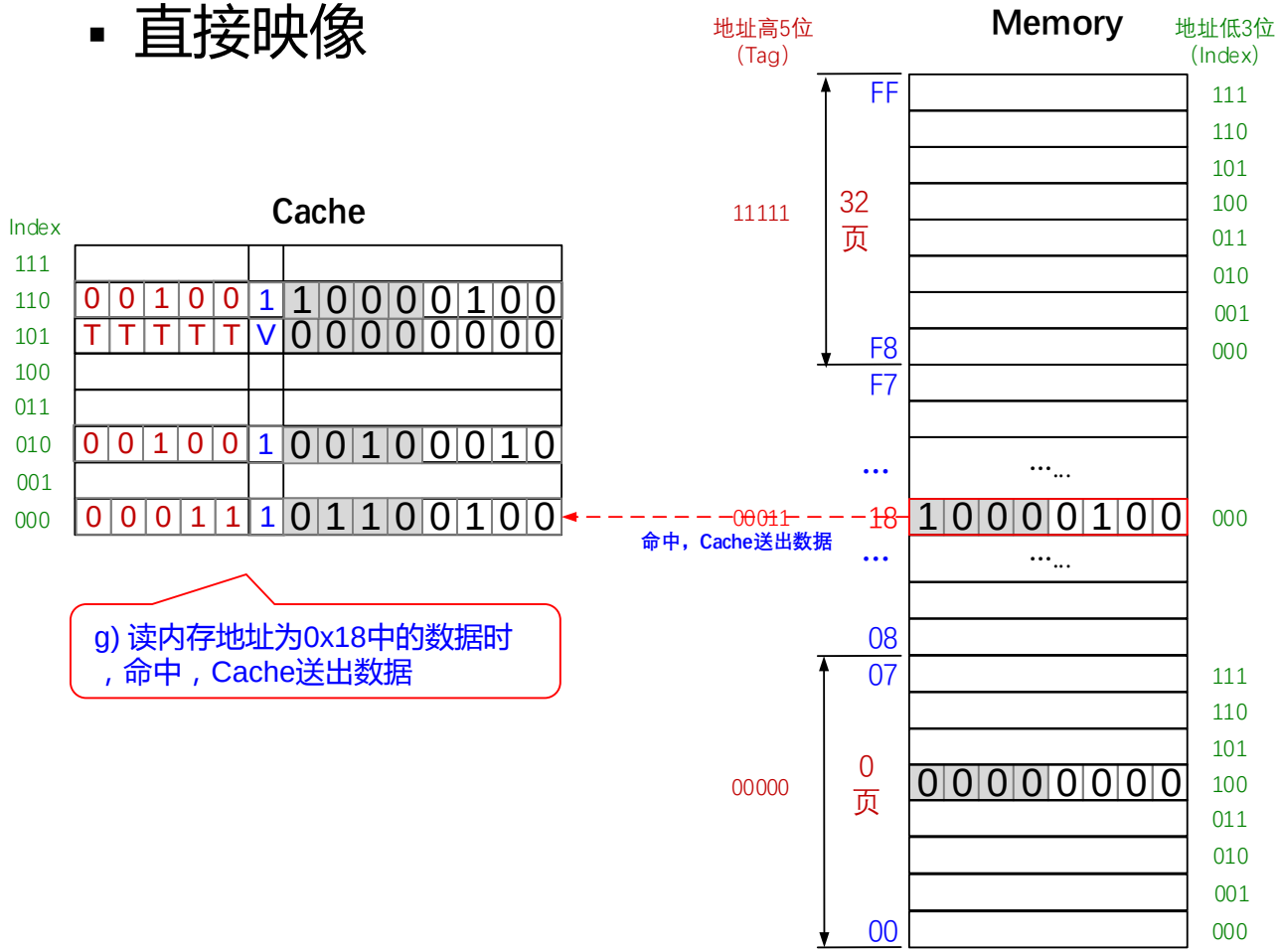


  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

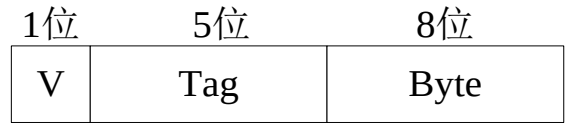
4.2 高速缓存(cache)原理

▶ Cache读策略

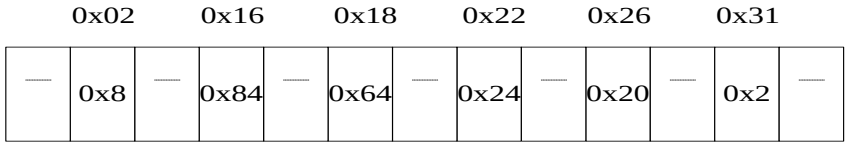
- Cache行填充（Line Fill）
  - 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据



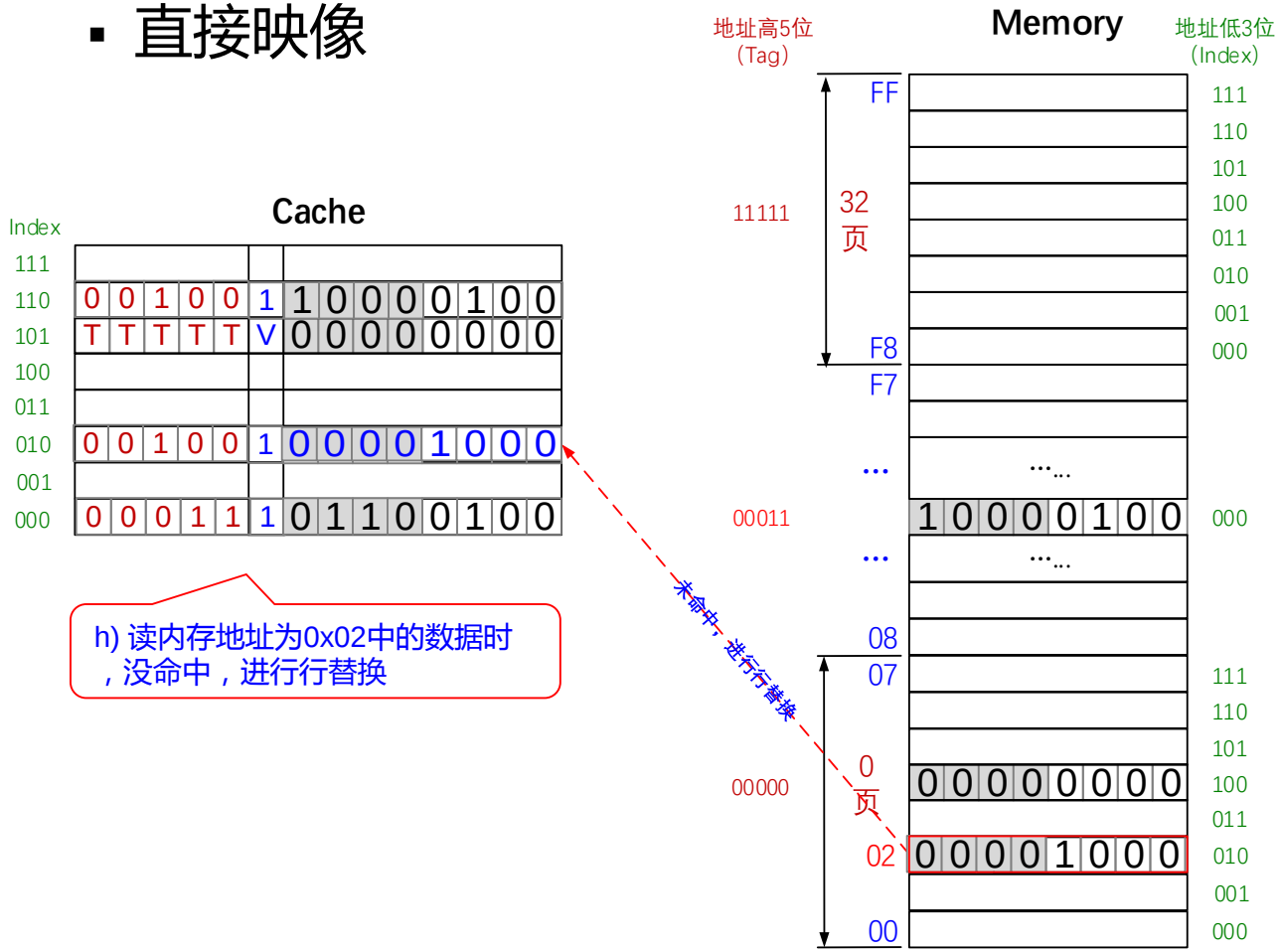
  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

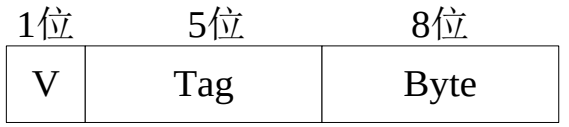
► Cache读策略

• Cache行填充（Line Fill）

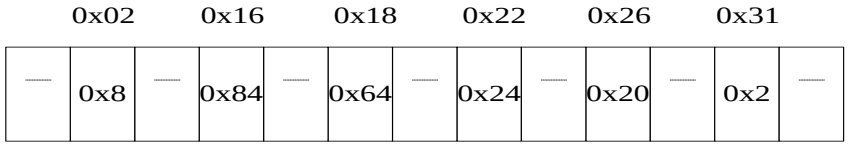
▪ 直接映像



- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据



  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

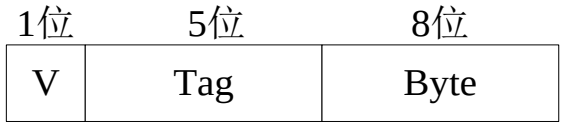
▶ Cache读策略

- Cache行填充（Line Fill）
  - 直接映像

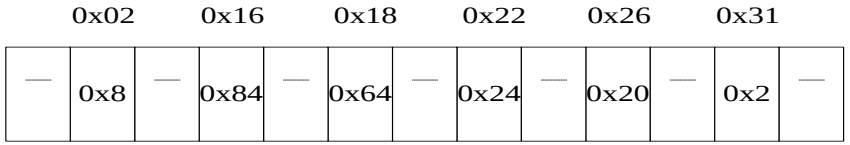
CPU操作	是否命中	Cache操作
b）读0x26单元	未命中	填充110行
c）读0x22单元	未命中	填充010行
d）读0x26单元	命中	送出110行数据
e）读0x18单元	未命中	填充000行
f）读0x16单元	未命中	替换110
g）读0x18单元	命中	送出000行数据
h）读0x02单元	未命中	替换010行

- 行容量过小，会产生频繁的行填充、替换操作，整机效率不会明显提高；
- 行容量过大，Cache填充所需的时间较长，且许多数据并一定是CPU最近所需的；
- Cache容量大小，需要折中考虑。

- 例1. 假定cache的每行仅存储一个字节的数据，共8行。CPU共8位地址总线，可以访问256个字节空间。上电初始化时，cache中没有存储任何内存单元数据的拷贝，所有行的有效位都为0。假定cache各行的结构



  - 内存中一段区域的初始数据



  - 依次读取内存地址为：0x26, 0x22, 0x26, 0x18, 0x16, 0x18, 0x02中的数据。

4.2 高速缓存(cache)原理

Cache写策略

- CPU向主存储器写数据时，Cache控制器会判断其地址是否定位在Cache中，如果在（命中），CPU的数据就会写到Cache中，否则写到主存储器。

- Cache命中
  - 透写
    - 既写Cache也写主存储器，能保持Cache与主存储器内容的一致，但使系统效率降低
  - 回写
    - 只写Cache，而不写主存储器，仅当Cache数据要被替换时才将其写到主存储器，效率高，但是Cache行中需要增加一位“修改”标志位。

- Cache非命中
  - 配写
    - CPU将数据写到主存储器后，再由Cache控制器向Cache中拷贝一个新的Cache行
  - 不配写
    - CPU只写主存储器而不写Cache。

► Cache替换策略

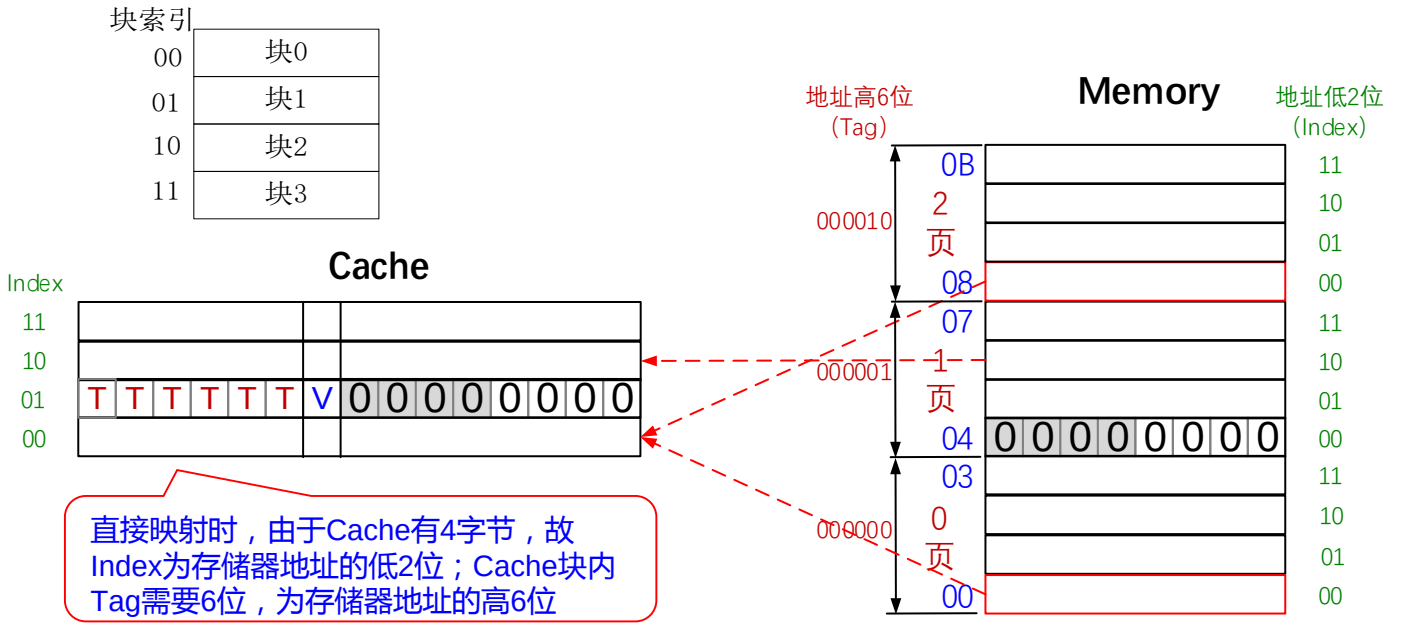
- 当Cache已经装满后，主存中的新数据可能还要不断地替换掉Cache中过时的数据，这就产生了Cache行的替换策略，替换哪些Cache行才能提高命中率呢？
  - 随机替换
    - 不管Cache块的过去、现在、将来使用情况，而随机的选择某块替换掉。
  - 先入先出（FIFO）替换
    - 依据进入Cache的先后次序来替换，先进的首先被替换掉
  - 最近最少使用（LRU）替换（最常采用）
    - 先替换掉最近用得不多的块

Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 直接映射

内存地址	对应的cache块索引(Index：地址低2位)
0	(0%4)=0
8	(8%4)=0
6	(6%4)=2

内存地址	命中否	cache各块存放的数据			
		块0	块1	块2	块3
0	未命中	mem[0]			
8	未命中	mem[8]			
0	未命中	mem[0]			
6	未命中	mem[0]		mem[6]	
8	未命中	mem[8]		mem[6]	



直接映射时，由于Cache有4字节，故Index为存储器地址的低2位；Cache块内Tag需要6位，为存储器地址的高6位

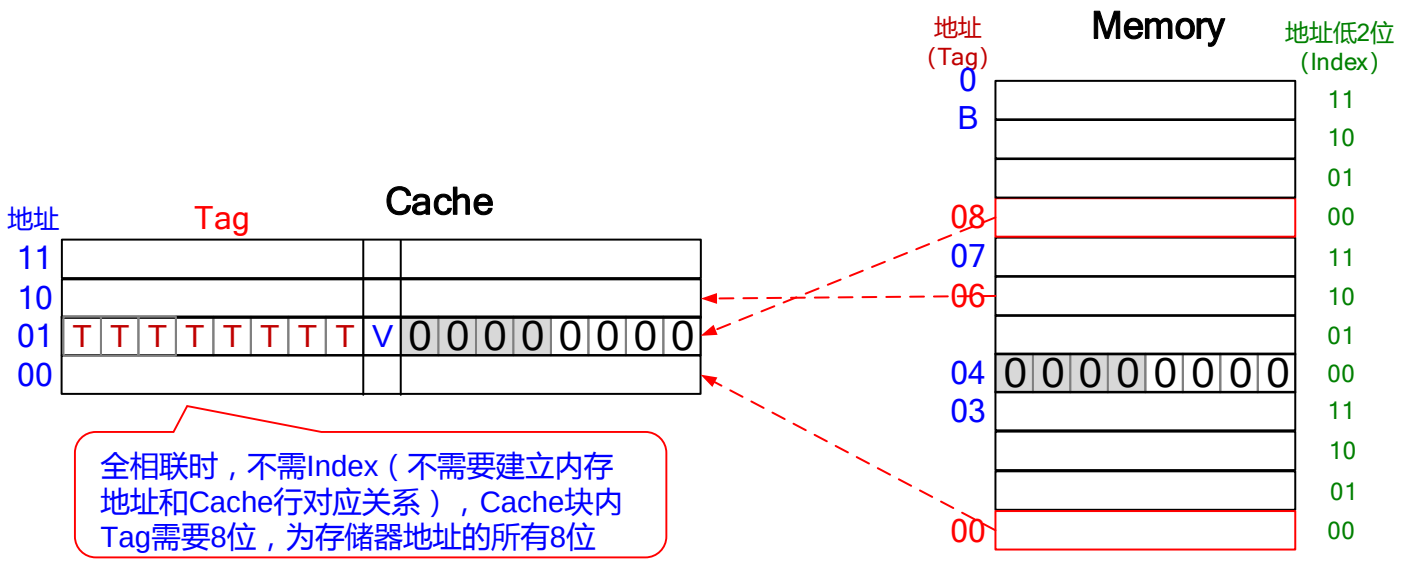
5次访问，命中0次

Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 全相联cache

内存地址	命中否	cache各块存放的数据			
		块0	块1	块2	块3
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[8]	mem[6]	
8	命中	mem[0]	mem[8]	mem[6]	

5次访问，命中2次



4.2 高速缓存(cache)原理

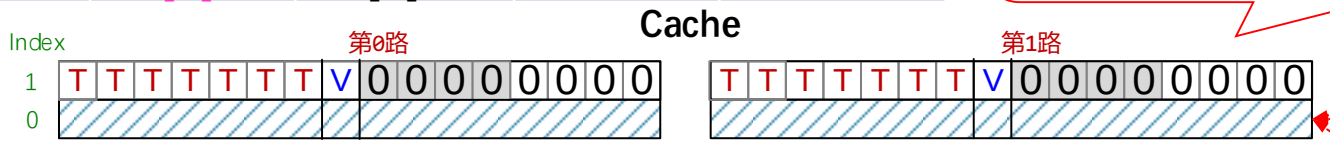
▶ Cache替换策略

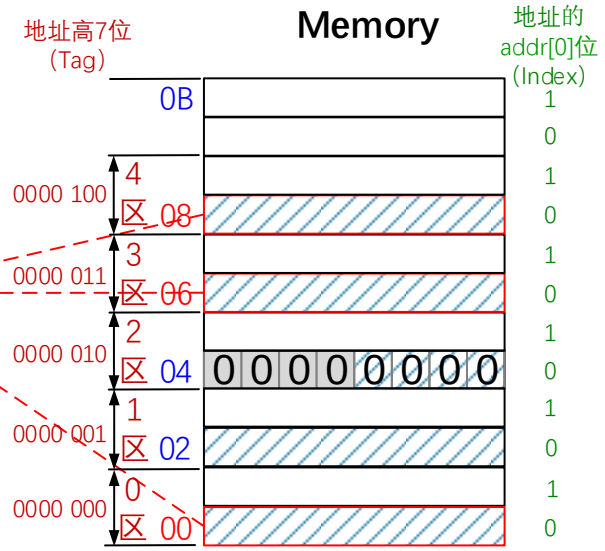
- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

5次访问，命中1次

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位





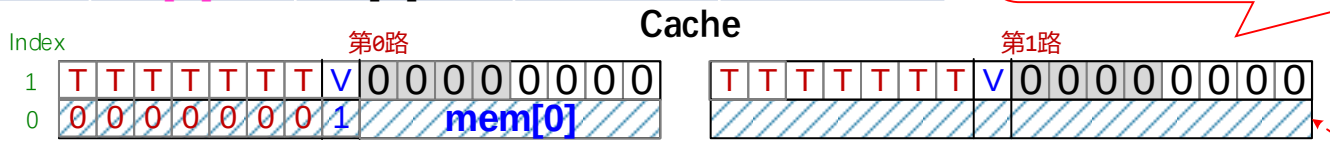
1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

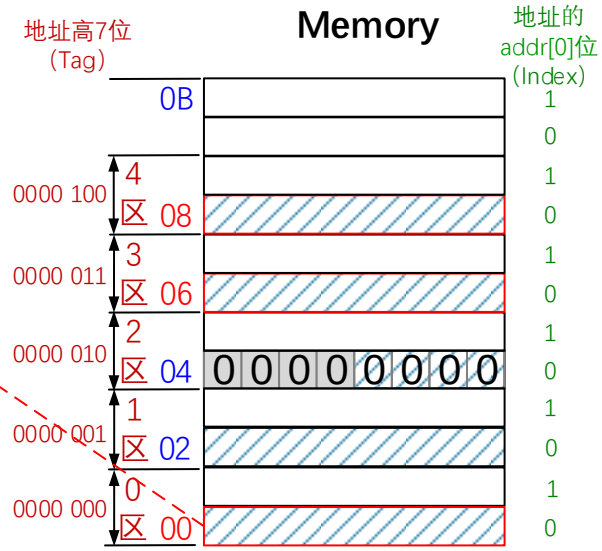
▶ Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位





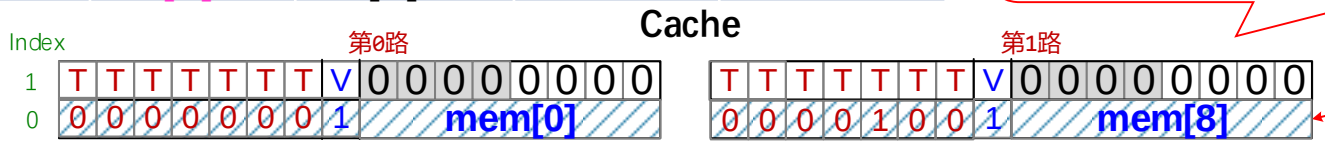
1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

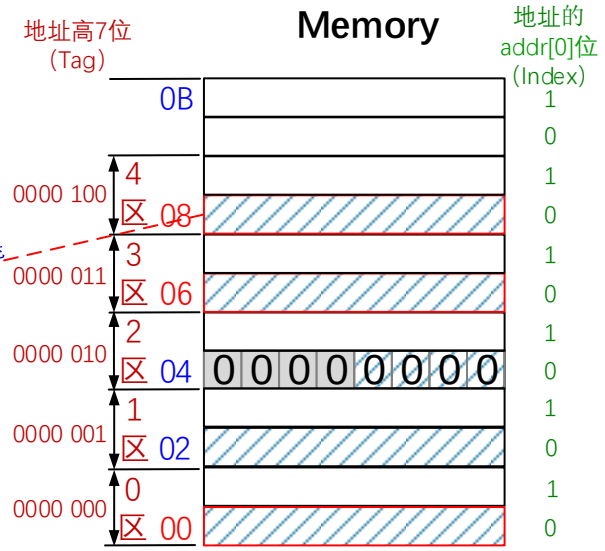
▶ Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位





1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

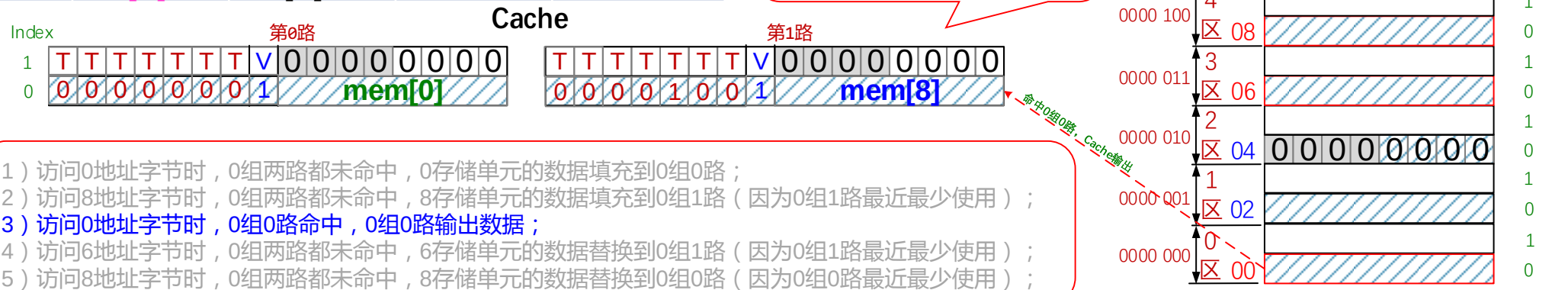
4.2 高速缓存(cache)原理

▶ Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位



1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

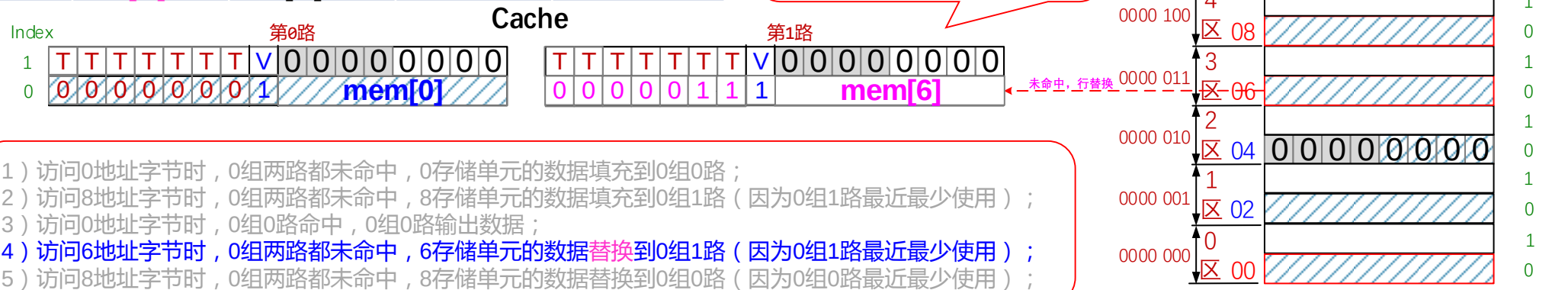
4.2 高速缓存(cache)原理

▶ Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位



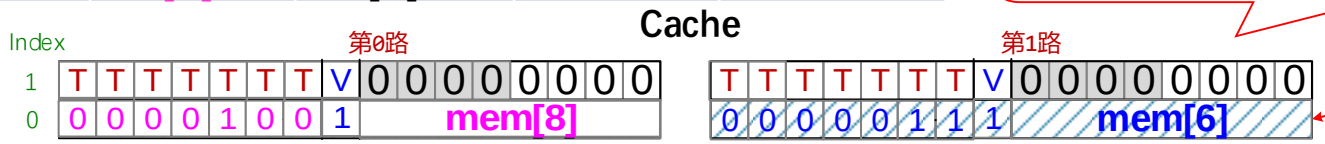
1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

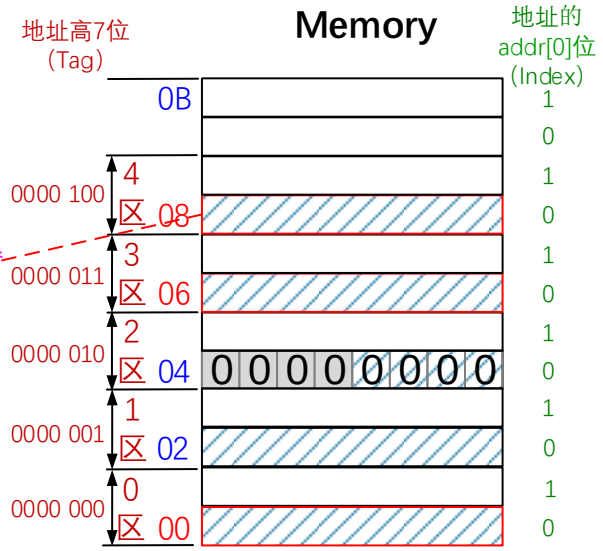
Cache替换策略

- 例2 假定具有采用三种映射方式的三个小容量cache，每个cache具有4行，每行存储1个字节数据，试说明CPU访问以下连续内存地址空间时：0，8，0，6，8，每种cache未命中的次数。
  - 2路组相联（采用最近最少使用优先替换）

访问内存地址	命中否	cache各块存放的数据			
		第0组		第1组	
		路0	路1	路0	路1
0	未命中	mem[0]			
8	未命中	mem[0]	mem[8]		
0	命中	mem[0]	mem[8]		
6	未命中	mem[0]	mem[6]		
8	未命中	mem[8]	mem[6]		

2路组相联时，Cache分为2路，每路2个行，故Index为存储器地址的addr[0]位；Cache块内Tag需要7位，为存储器地址的高7位





5次访问，命中1次

1）访问0地址字节时，0组两路都未命中，0存储单元的数据填充到0组0路；
2）访问8地址字节时，0组两路都未命中，8存储单元的数据填充到0组1路（因为0组1路最近最少使用）；
3）访问0地址字节时，0组0路命中，0组0路输出数据；
4）访问6地址字节时，0组两路都未命中，6存储单元的数据替换到0组1路（因为0组1路最近最少使用）；
5）访问8地址字节时，0组两路都未命中，8存储单元的数据替换到0组0路（因为0组0路最近最少使用）；

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

列优先访问策略，命中率50%

a[0/8][0] = 0 时，Cache行000未命中，a[0/8][0]、a[0/8][1]共4B进行行填充/替换；
a[0/8][1] = 0 时，Cache行000命中， 由Cache行送出a[0/8][0]、a[0/8][1]内容；
。。。
a[7/15][0] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行填充/替换；
a[7/15][1] = 0 时，Cache行111命中， 由Cache行送出a[7/15][0]、a[0/15][1]内容；

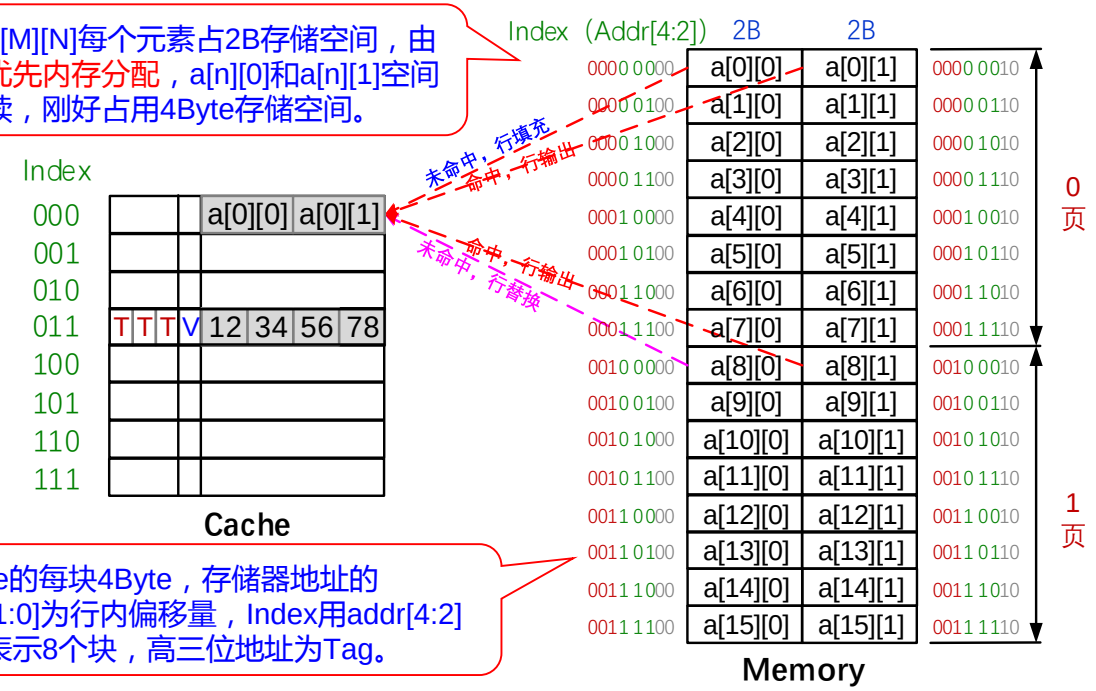
程序段A（列优先访问）：

```
assign-array-rows()
{
    int         i, j;
    short       a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
                a[i][j] = 0;
}
```

程序段B（行优先访问）：

```
assign-array-cols()
{
    int         i, j;
    short       a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
                a[i][j] = 0;
}
```

数组a[M][N]每个元素占2B存储空间，由于列优先内存分配，a[n][0]和a[n][1]空间上连续，刚好占用4Byte存储空间。



Cache的每块4Byte，存储器地址的addr[1:0]为行内偏移量，Index用addr[4:2]三位表示8个块，高三位地址为Tag。

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

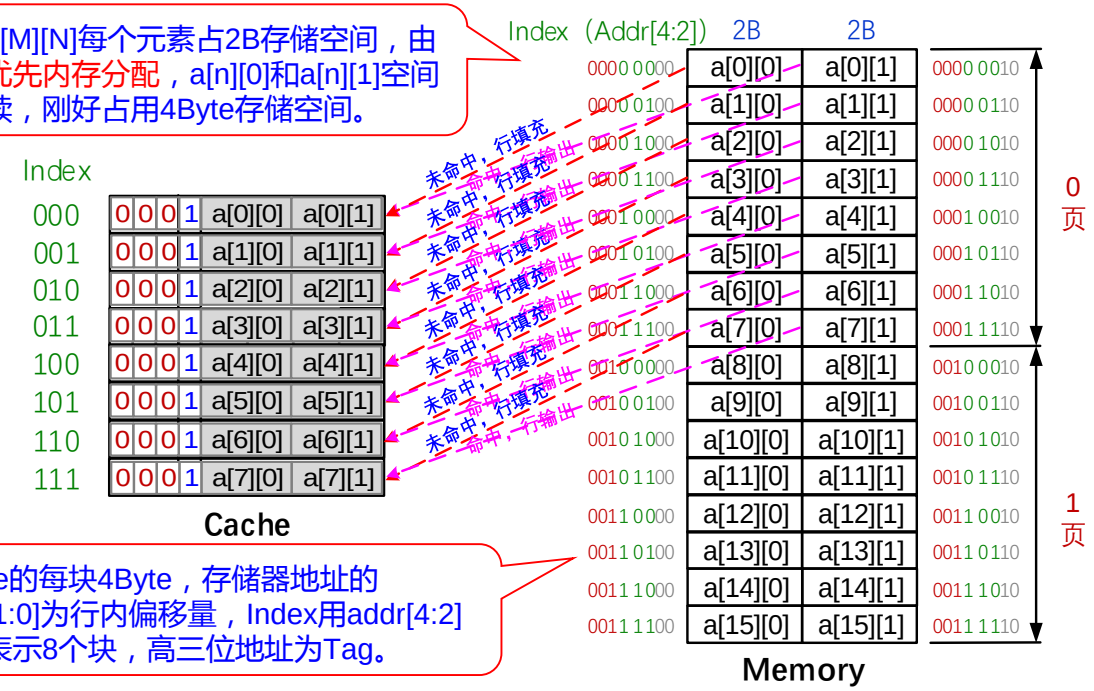
列优先访问策略，命中率50%

a[0/8][0] = 0 时，Cache行000未命中，a[0/8][0]、a[0/8][1]共4B进行行填充/替换；
a[0/8][1] = 0 时，Cache行000命中， 由Cache行送出a[0/8][0]、a[0/8][1]内容；
。。。
a[7/15][0] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行填充/替换；
a[7/15][1] = 0 时，Cache行111命中， 由Cache行送出a[7/15][0]、a[0/15][1]内容；

```
程序段A（列优先访问）：
assign-array-rows()
{
    int         i, j;
    short       a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
            a[i][j] = 0;
}
```

```
程序段B（行优先访问）：
assign-array-cols()
{
    int         i, j;
    short       a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
            a[i][j] = 0;
}
```

数组a[M][N]每个元素占2B存储空间，由于列优先内存分配，a[n][0]和a[n][1]空间上连续，刚好占用4Byte存储空间。



Cache的每块4Byte，存储器地址的addr[1:0]为行内偏移量，Index用addr[4:2]三位表示8个块，高三位地址为Tag。

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

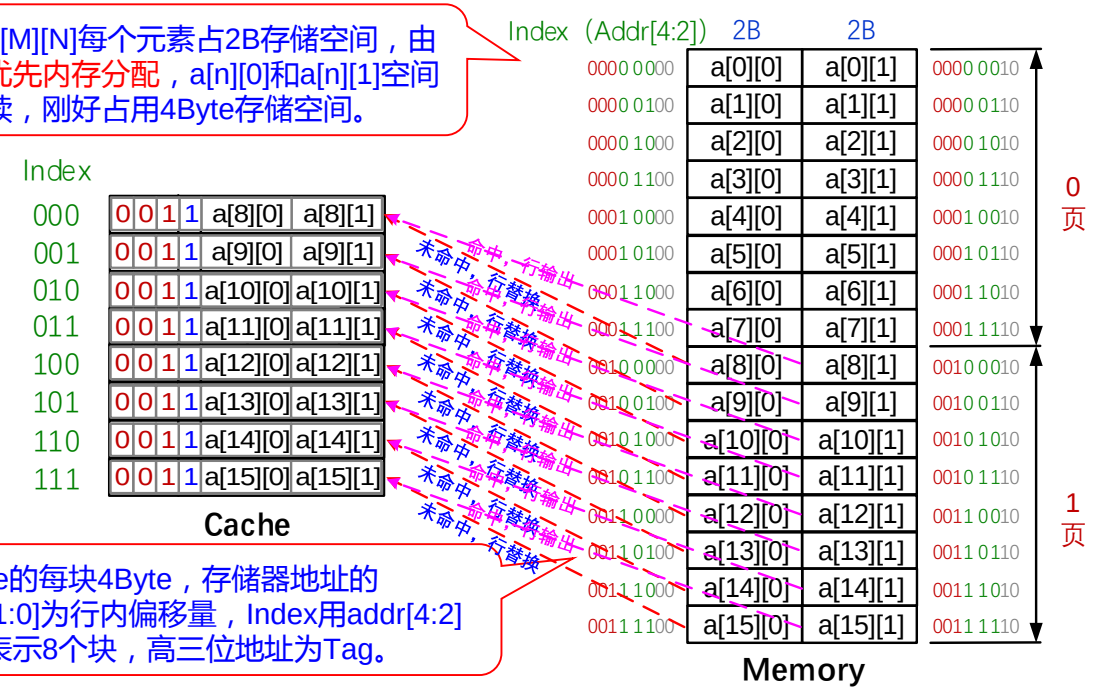
列优先访问策略，命中率50%

a[0/8][0] = 0 时，Cache行000未命中，a[0/8][0]、a[0/8][1]共4B进行行填充/替换；
a[0/8][1] = 0 时，Cache行000命中， 由Cache行送出a[0/8][0]、a[0/8][1]内容；
。。。
a[7/15][0] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行填充/替换；
a[7/15][1] = 0 时，Cache行111命中， 由Cache行送出a[7/15][0]、a[0/15][1]内容；

```
程序段A（列优先访问）：
assign-array-rows()
{
    int         i, j;
    short       a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
            a[i][j] = 0;
}
```

```
程序段B（行优先访问）：
assign-array-cols()
{
    int         i, j;
    short       a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
            a[i][j] = 0;
}
```

数组a[M][N]每个元素占2B存储空间，由于列优先内存分配，a[n][0]和a[n][1]空间上连续，刚好占用4Byte存储空间。



Cache的每块4Byte，存储器地址的addr[1:0]为行内偏移量，Index用addr[4:2]三位表示8个块，高三位地址为Tag。

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

列优先访问策略，命中率50%

数组元素	中否	cache内容							
		块0	块1	块2	块3	块4	块5	块6	块7
a[0][0]	否	a[0][0] a[0][1]							
a[0][1]	中								
a[1][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]						
a[1][1]	中								
a[2][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]					
a[2][1]	中								
a[3][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]				
a[3][1]	中								
a[4][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]			
a[4][1]	中								
a[5][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]		
a[5][1]	中								
a[6][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	
a[6][1]	中								
a[7][0]	否	**a[0][0] a[0][1]**	**a[1][0] a[1][1]**	**a[2][0] a[2][1]**	**a[3][0] a[3][1]**	**a[4][0] a[4][1]**	**a[5][0] a[5][1]**	**a[6][0] a[6][1]**	**a[7][0] a[7][1]**
a[7][1]	中								
a[8][0]	否	a[8][0] a[8][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[8][1]	中								
a[9][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[9][1]	中								
a[10][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[10][1]	中								
a[11][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[11][1]	中								
a[12][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[12][1]	中								
a[13][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[13][1]	中								
a[14][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[7][0] a[7][1]
a[14][1]	中								
a[15][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[15][1]	中								

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

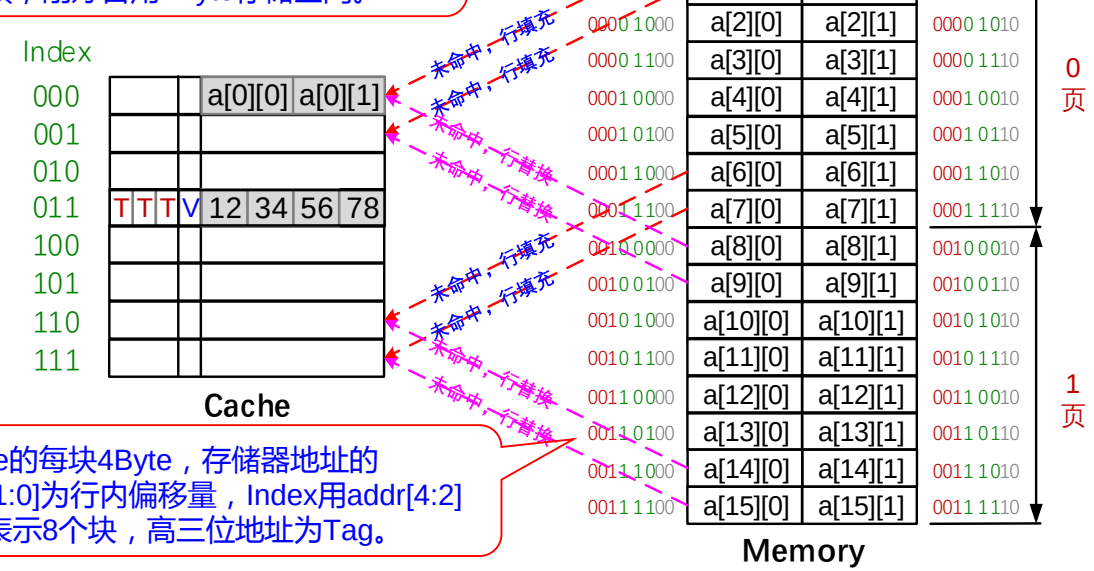
行优先访问策略，a[0][0]到a[15][0],命中率0%

a[0/8][0] = 0 时，Cache行000未命中，a[0/8][0]、a[0/8][1]共4B进行行填充/替换；
a[1/8][0] = 0 时，Cache行000未命中，a[1/8][0]、a[1/8][1]共4B进行行填充/替换；
。。。
a[7/15][0] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行填充/替换；
a[7/15][0] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行填充/替换；

```
程序段A（列优先访问）：
assign-array-rows()
{
    int         i, j;
    short       a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
            a[i][j] = 0;
}
```

```
程序段B（行优先访问）：
assign-array-cols()
{
    int         i, j;
    short       a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
            a[i][j] = 0;
}
```

数组a[M][N]每个元素占2B存储空间，由于列优先内存分配，a[n][0]和a[n][1]空间上连续，刚好占用4Byte存储空间。



Cache的每块4Byte，存储器地址的addr[1:0]为行内偏移量，Index用addr[4:2]三位表示8个块，高三位地址为Tag。

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

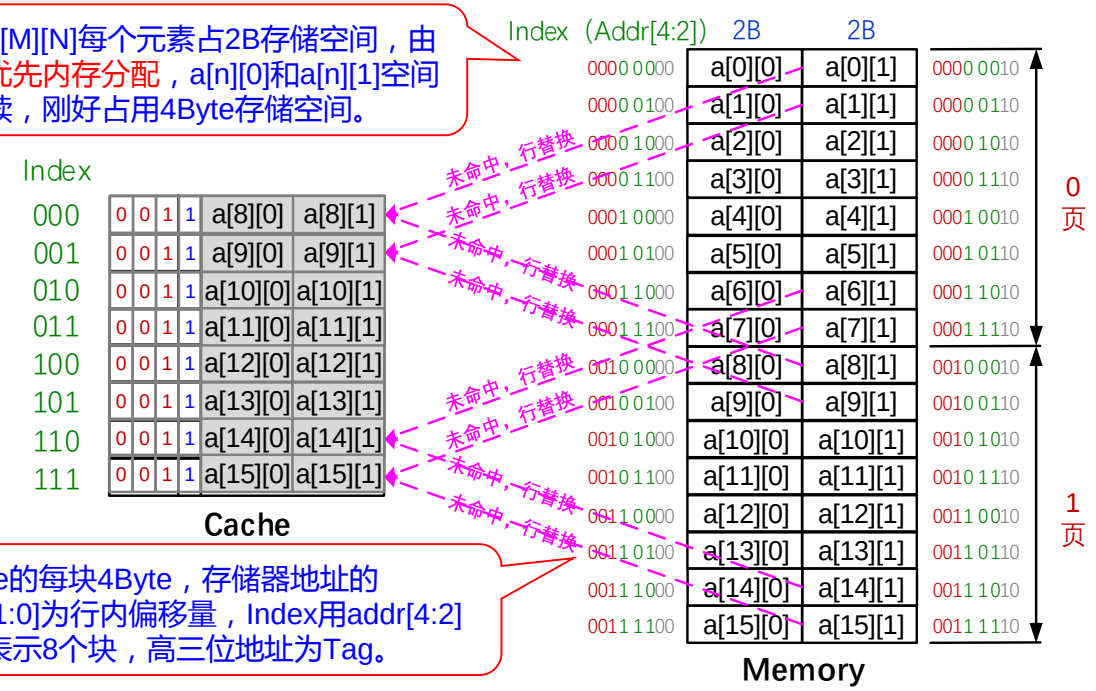
行优先访问策略，a[0][1]到a[15][1],命中率0%

a[0/8][1] = 0 时，Cache行000未命中，a[0/8][0]、a[0/8][1]共4B进行行替换；
a[1/8][1] = 0 时，Cache行000未命中，a[1/8][0]、a[1/8][1]共4B进行行替换；
。。。
a[7/15][1] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行替换；
a[7/15][1] = 0 时，Cache行111未命中，a[7/15][0]、a[7/15][1]共4B进行行替换；

```
程序段A（列优先访问）：
assign-array-rows()
{
    int         i, j;
    short       a[M][N];
    for(i=0; i<M; i++)
        for(j=0; j<N; j++)
            a[i][j] = 0;
}
```

```
程序段B（行优先访问）：
assign-array-cols()
{
    int         i, j;
    short       a[M][N];
    for(j=0; j<N; j++)
        for(i=0; i<M; i++)
            a[i][j] = 0;
}
```

数组a[M][N]每个元素占2B存储空间，由于列优先内存分配，a[n][0]和a[n][1]空间上连续，刚好占用4Byte存储空间。



Cache的每块4Byte，存储器地址的addr[1:0]为行内偏移量，Index用addr[4:2]三位表示8个块，高三位地址为Tag。

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - 已知一个32B的cache，每块4个字节，共8块；或每块8个字节，共4块。

- 每块4个字节，共8块的cache结构
  - cache块大小为4字节，数组a[M][N]为short类型，即每个元素占据2个字节，当N为2时，每一行仅2个元素，因此一行数组元素占据4个字节正好对应cache的一块

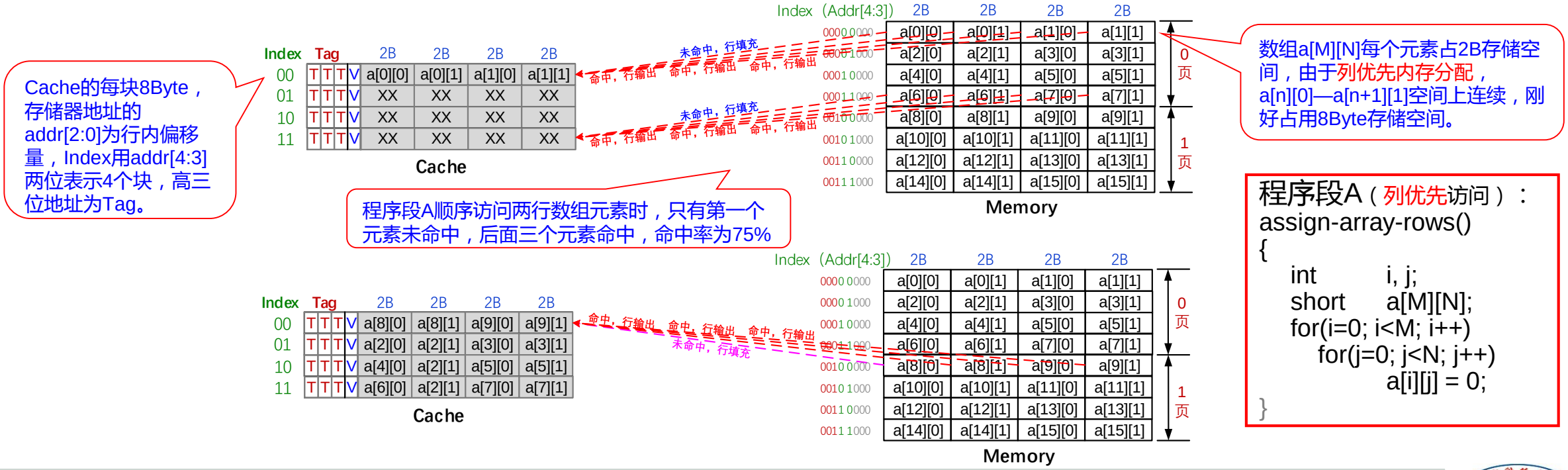
行优先访问策略，命中率 0%

数组元素	命中否	cache内容							
		块0	块1	块2	块3	块4	块5	块6	块7
a[0][0]	否	**a[0][0]** **a[0][1]**							
a[1][0]	否	a[0][0] a[0][1]	**a[1][0]** **a[1][1]**						
a[2][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	**a[2][0]** **a[2][1]**					
a[3][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	**a[3][0]** **a[3][1]**				
a[4][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	**a[4][0]** **a[4][1]**			
a[5][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	**a[5][0]** **a[5][1]**		
a[6][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	**a[6][0]** **a[6][1]**	
a[7][0]	否	**a[0][0]** **a[0][1]**	**a[1][0]** **a[1][1]**	**a[2][0]** **a[2][1]**	**a[3][0]** **a[3][1]**	**a[4][0]** **a[4][1]**	**a[5][0]** **a[5][1]**	**a[6][0]** **a[6][1]**	**a[7][0]** **a[7][1]**
a[8][0]	否	**a[8][0]** **a[8][1]**	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[9][0]	否	a[8][0] a[8][1]	**a[9][0]** **a[9][1]**	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[10][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	**a[10][0]** **a[10][1]**	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[11][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	**a[11][0]** **a[11][1]**	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[12][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	**a[12][0]** **a[12][1]**	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[13][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	**a[13][0]** **a[13][1]**	a[6][0] a[6][1]	a[7][0] a[7][1]
a[14][0]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	**a[14][0]** **a[14][1]**	a[7][0] a[7][1]
a[15][0]	否	**a[8][0]** **a[8][1]**	**a[9][0]** **a[9][1]**	**a[10][0]** **a[10][1]**	**a[11][0]** **a[11][1]**	**a[12][0]** **a[12][1]**	**a[13][0]** **a[13][1]**	**a[14][0]** **a[14][1]**	**a[15][0]** **a[15][1]**
a[0][1]	否	**a[0][0]** **a[0][1]**	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[1][1]	否	a[0][0] a[0][1]	**a[1][0]** **a[1][1]**	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[2][1]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	**a[2][0]** **a[2][1]**	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[3][1]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	**a[3][0]** **a[3][1]**	a[12][0] a[12][1]	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[4][1]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	**a[4][0]** **a[4][1]**	a[13][0] a[13][1]	a[14][0] a[14][1]	a[15][0] a[15][1]
a[5][1]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	**a[5][0]** **a[5][1]**	a[14][0] a[14][1]	a[15][0] a[15][1]
a[6][1]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	**a[6][0]** **a[6][1]**	a[15][0] a[15][1]
a[7][1]	否	**a[0][0]** **a[0][1]**	**a[1][0]** **a[1][1]**	**a[2][0]** **a[2][1]**	**a[3][0]** **a[3][1]**	**a[4][0]** **a[4][1]**	**a[5][0]** **a[5][1]**	**a[6][0]** **a[6][1]**	**a[7][0]** **a[7][1]**
a[8][1]	否	**a[8][0]** **a[8][1]**	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[9][1]	否	a[8][0] a[8][1]	**a[9][0]** **a[9][1]**	a[2][0] a[2][1]	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[10][1]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	**a[10][0]** **a[10][1]**	a[3][0] a[3][1]	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[11][1]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	**a[11][0]** **a[11][1]**	a[4][0] a[4][1]	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[12][1]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	**a[12][0]** **a[12][1]**	a[5][0] a[5][1]	a[6][0] a[6][1]	a[7][0] a[7][1]
a[13][1]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	**a[13][0]** **a[13][1]**	a[6][0] a[6][1]	a[7][0] a[7][1]
a[14][1]	否	a[8][0] a[8][1]	a[9][0] a[9][1]	a[10][0] a[10][1]	a[11][0] a[11][1]	a[12][0] a[12][1]	a[13][0] a[13][1]	**a[14][0]** **a[14][1]**	a[7][0] a[7][1]
a[15][1]	否	**a[8][0]** **a[8][1]**	**a[9][0]** **a[9][1]**	**a[10][0]** **a[10][1]**	**a[11][0]** **a[11][1]**	**a[12][0]** **a[12][1]**	**a[13][0]** **a[13][1]**	**a[14][0]** **a[14][1]**	**a[15][0]** **a[15][1]**

4.2 高速缓存(cache)原理

▶ Cache影响程序性能

- 分析以下两种cache大小分块组织方式下，short型数组a[M][N] 列优先内存分配，采用行、列优先访问策略，当M，N分别为16，2时，cache直接映像策略的命中率。
  - 假定数据元素从内存地址0x0000 0000开始分配
  - cache块大小变为8字节，共4块。
    - 一次将拷贝两行数组元素进入cache块，因此，程序段A顺序访问两行数组元素时，只有第一个元素未命中，后面三个元素命中，命中率提高为75%（见下图），而程序段B的cache命中率提高为50%（自行分析）。



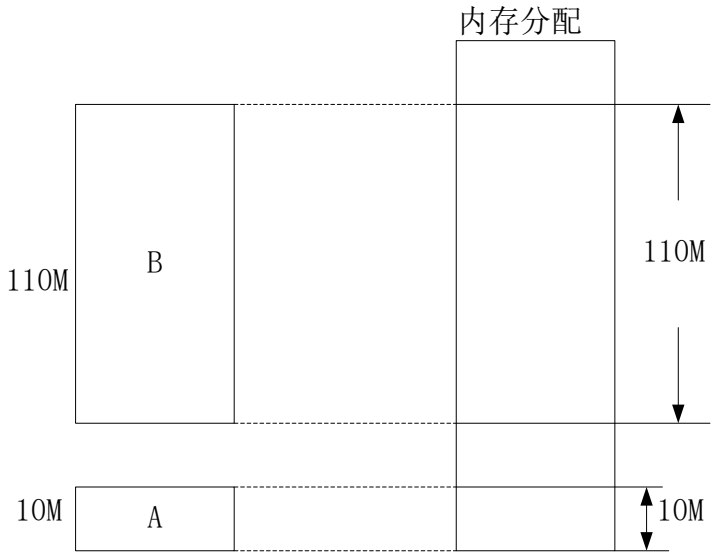
第4章 作业（一）

- 作业题（第2版，P164）
  - 1
  - 7
  - 8
  - 11

- 要求：
  - 微助教提交；
  - 下次课前提交。

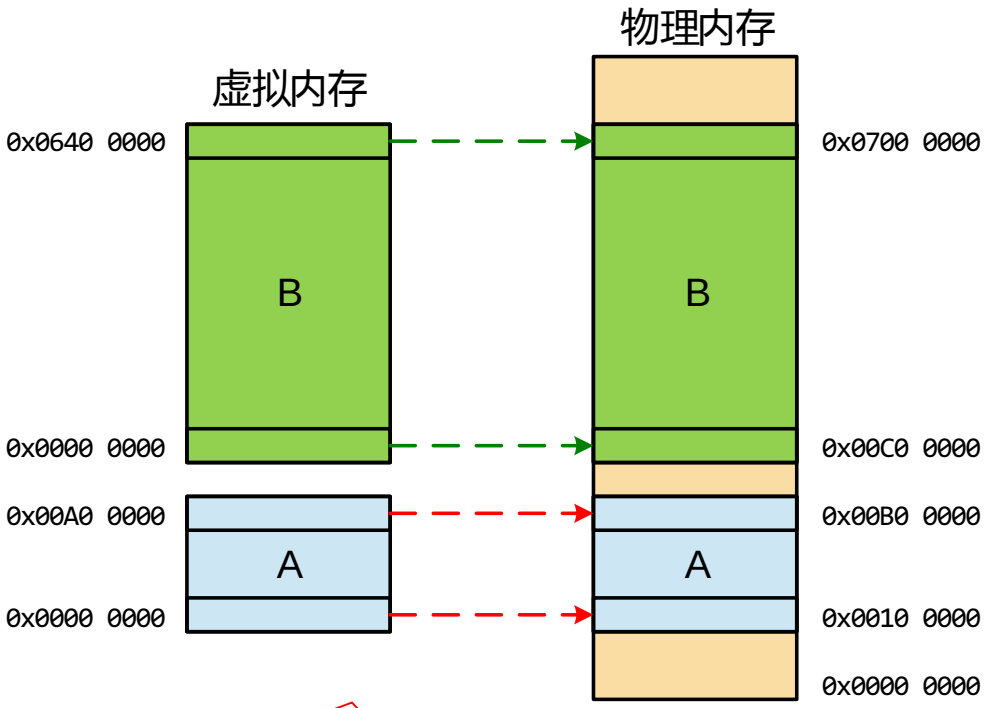
- 内容
  - 存储器的作用及分类
  - Cache的三种地址映射策略：直接映射、全相联、组相联
  - Cache读策略、写策略、替换策略
  - 内存的三种管理方式：分页式、分段式、段页式
  - 虚拟存储技术
- 目的
  - 了解计算机系统存储系统的分级结构特点
  - 理解Cache的基本概念
  - 掌握Cache的三种映射策略，理解Cache对计算机系统的作用和影响
  - 了解存储器虚拟地址到物理地址映射的机制
  - 了解虚拟存储器的基本原理

- 早期的简单内存分配方法
  - 程序一般都在内存中运行
  - 运行多个程序时，必须保证用到的内存总量小于计算机实际物理内存。
  - 实地址：早期，程序访问的内存地址就是实际的物理地址，问题多
    - 进程地址空间不隔离（误操作或恶意代码）
    - 内存使用效率低（切换粒度为程序）
    - 程序运行地址不确定（软件设计复杂）
  - 解决办法：虚拟地址
    - 增加一个中间层，利用一种间接地址访问方法访问物理内存
      - 程序中访问的内存地址不再是实际物理内存地址，而是一个虚拟地址，
      - 然后由操作系统将这个虚拟地址映射到适当的物理内存地址上



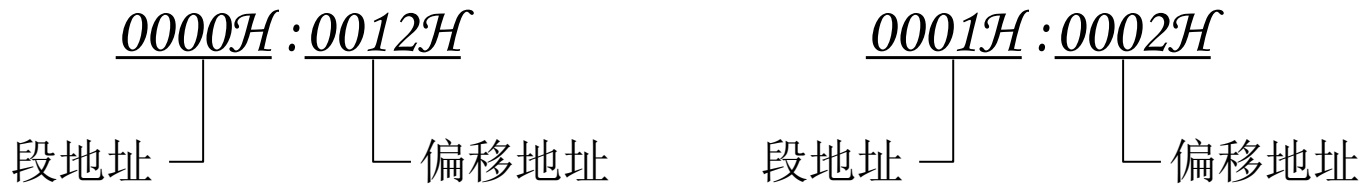
4.3 内存管理

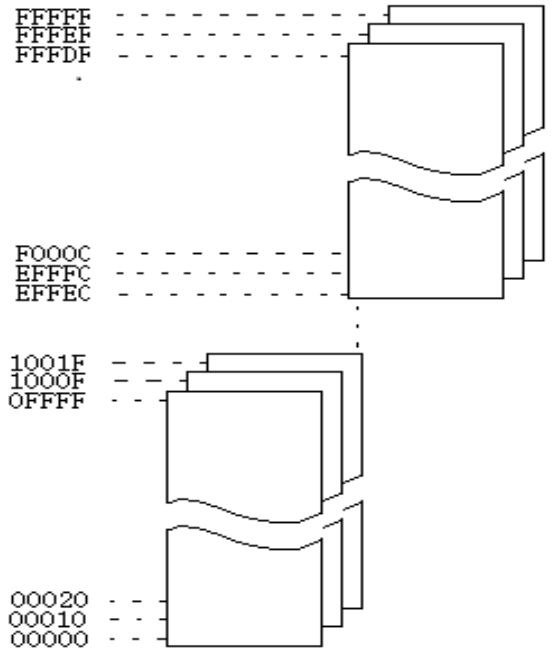
- 内存管理策略
  - 分段：解决地址不确定以及冲突
  - 分页：解决粗粒度调度
  - 段页式
- 分段
  - 在虚拟地址空间和物理地址空间之间做一一映射。
    - 比如说虚拟地址空间中某个10M大小的空间映射到物理地址空间中某个10M大小的空间
  - 由操作系统保证不同进程的物理地址空间彼此分开
  - 应用程序并不关心进程A究竟被映射到物理内存的哪块区域上，只需利用偏移地址访问内存单元即可



当微处理器要访问程序中的某个内存单元时，将利用该程序映射到的物理地址空间段的起始地址（段地址）和偏移地址相结合的方式来实现寻址

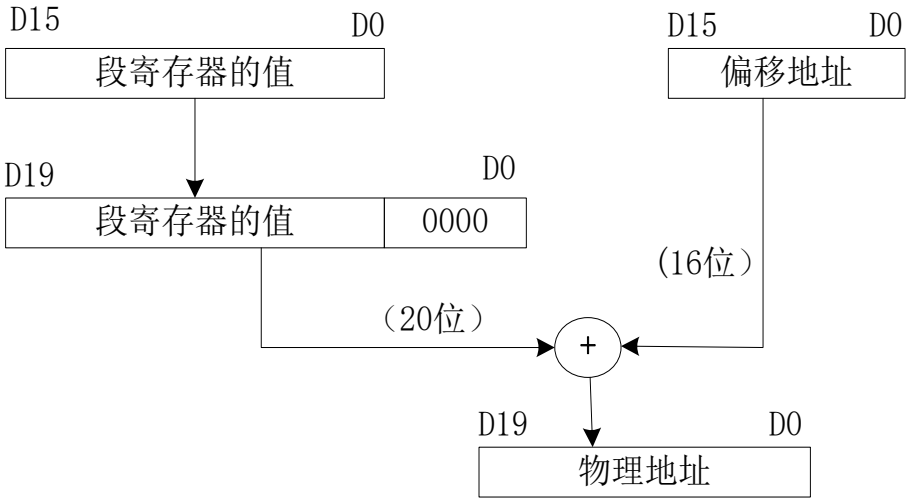
- 分段
  - 分段实例
    - Intel 实地址模式
      - 8086/8088 MPU外部具有20根地址总线，而内部寄存器都是16位，不能直接由寄存器输出20位地址，所以存储器采用分段管理方式：把2^{20}B=1MB的空间分成一个个2^{16}B=64KB的段，并且使每个段的起始地址(20位)的低四位都为0，每个段的起始地址(20位)的高十六位地址称为该段的段地址，段内再用一个16位的二进制数来表示存储单元到首地址的距离，称为偏移地址；
      - 如此，一个存储单元的地址除了用20位的物理地址表示外，还可以用段地址：偏移地址的形式表示，并称这种地址为逻辑地址；
      - 2^{20}B(1MB)空间分成4K个（段地址）64KB（偏移地址）的逻辑段，程序访问的是逻辑空间，采用16位段地址：16位偏移地址的形式。





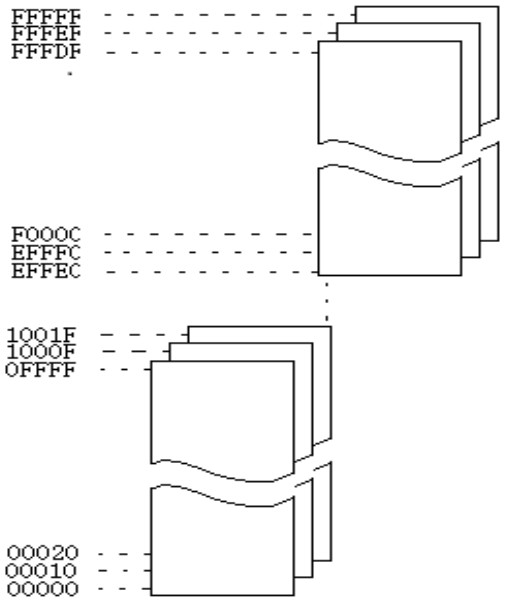
实地址模式分段

- 分段
  - 分段实例
    - Intel 实地址模式：逻辑地址与物理地址的关系
      - 程序中使用逻辑地址，其中段地址存放在段寄存器DS、SS、CS、ES中，偏移地址存放在BX、SI、DI、 BP 、SP、IP中，如：
        - DS:BX、DS:SI、DS:DI、DS:BP (数据区)
        - SS:SP (堆栈区)
        - CS:IP (代码区)
      - CPU 访问内存、外设使用物理地址；
      - 物理地址由CPU中的地址加法器将段地址左移4位，低位补0，再与16位偏移地址相加形成：
        - 物理地址=段地址×10H+偏移地址

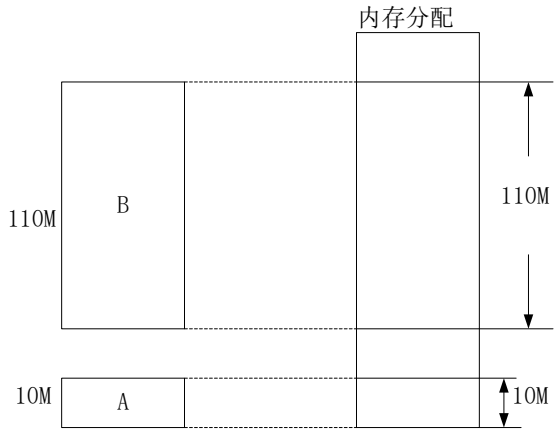


物理地址的形成

- 分段
  - 分段实例
    - Intel 实地址模式
      - 仍然可能产生访问越界的问题。
        - 多个逻辑地址可能指向同一个物理地址，如：
          0000H：0020H的物理地址为 00020H
          0001H：0010H的物理地址为 00020H
          0002H：0000H的物理地址为 00020H
        - 可能导致程序A非法修改程序B中的数据，带来系统稳定性、可靠性问题。



实地址模式分段



► 分段

• 分段实例

▪ 保护模式

- 段的起始地址不再由段寄存器直接指示
- 段的起始地址通过段描述符(8Byte)来表示

字节7	段地址字节3	G	D	0	AV	界限高4位	字节6
字节5	访问权限	段地址字节2					字节4
字节3	段地址字节1	段地址字节0					字节2
字节1	界限字节1	界限字节0					字节0

- 已知某32位intel微处理器的段描述符为0x34 d3 00 23 12 89 01 03，试指出该段描述符对应的段的起始地址与结束地址。
  - 段的起始地址：字节7，字节4，字节3，字节2，为：0x34231289
  - 该段的界限为：字节6的低4位，字节1，字节0构成，即为：0x30103.
  - 该描述符的粒度G为1，因此实际的段界限为0x30103*4k,即为0x30103000.
  - 段的结束地址=起始地址+段界限-1.
  - 因此段的结束地址为：0x34231289+0x30103000-1=0x64334289

►32位段地址：7、4、3、2字节

►20位段的界限：6字节低4位和1、0字节

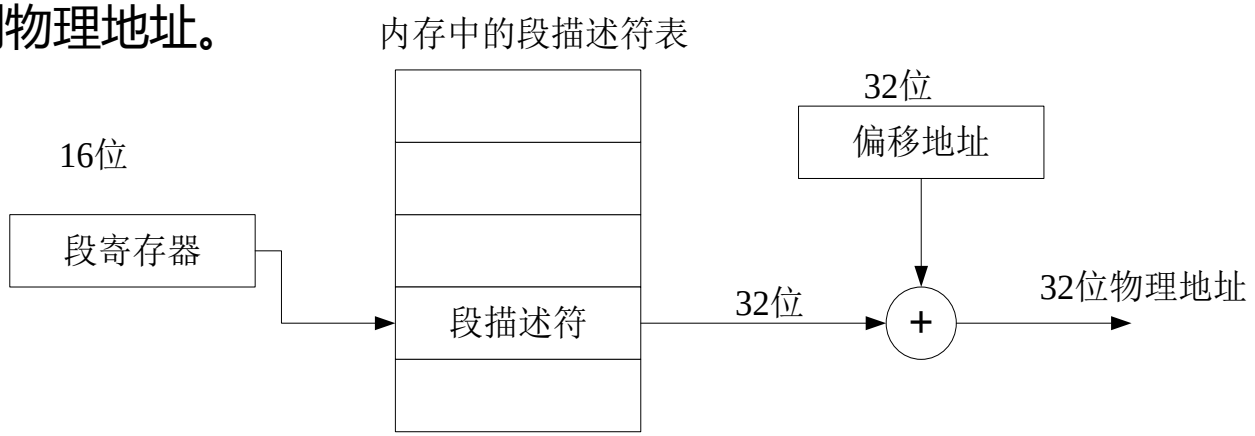
►G为界限的粒度，

- ► 当G=1时，段的最大偏移地址需要在20位界限表示的地址基础上乘以4K；
- ► 当G=0，段的最大偏移地址即为20位界限表示的地址

►AV表示此存储段是否有效：AV=1，表示有效，AV=0，表示无效。

►D表示指令模式：D=1，表示32位指令模式，D=0，表示16位指令模式。

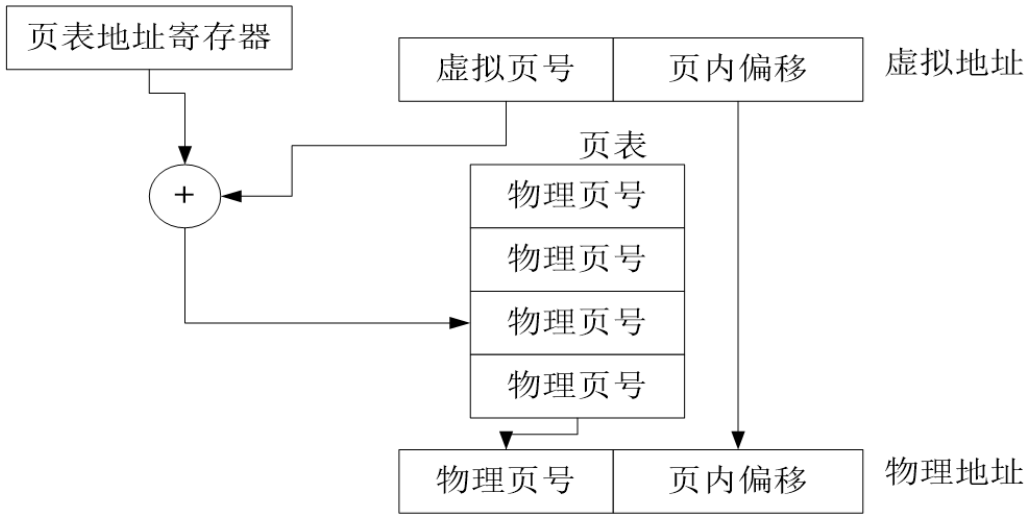
- 分段
  - 分段实例
    - 保护模式
      - 保护模式下物理地址形成过程
        - 程序仍然采用段寄存器和偏移地址形式来寻址内存单元
        - 但是段寄存器的值仅表示该段描述符在段描述表中的索引
        - 首先根据段寄存器的值查找到段描述符，然后从段描述符中获得该段的段地址，最后段地址与偏移地址相加得到物理地址。



      - 解决了内存地址重叠和不确定性，但是没能解决内存的使用效率问题，映射方法稍显粗糙，粒度比较大（换出内存的都是整个程序）。
      - 根据程序运行的“局部性”特点，可用粒度更小的内存分割和映射——分页

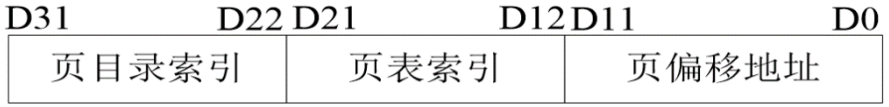
▶ 分页管理

- 基本方法
  - 存储器地址由两部分组成：页号、页内偏移
    - 将存储器地址空间分成许多相同大小的页，页的大小由CPU决定，页的大小决定了地址线中用于寻址页内存储单元的位数，地址线中的其余位则用来寻址页
  - 如：32位地址线的计算机系统，可寻址4GB存储空间
    - 若页的大小设为4KB
      - ▶则页内偏移地址用12位
      - ▶剩下的20位用来寻址不同的页，共1M个页
    - 但实际系统可能只有1GB物理内存，共256K页
      - ▶需要将4GB的虚拟内存空间映射到1GB的物理空间
  - 虚拟页到物理页的映射
    - 就是虚拟地址高20位到物理地址的高18位的映射
    - 通过建立一个页表来维护虚拟到物理的映射关系
      - ▶每项页表中包含物理页号
      - ▶虚拟页号就是物理页号在页表中的索引
      - ▶页表存储位置？——专门的页表地址寄存器

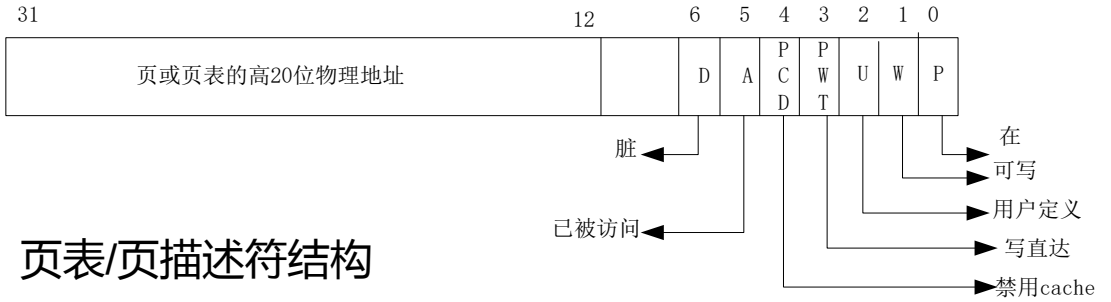


▶ 分页管理

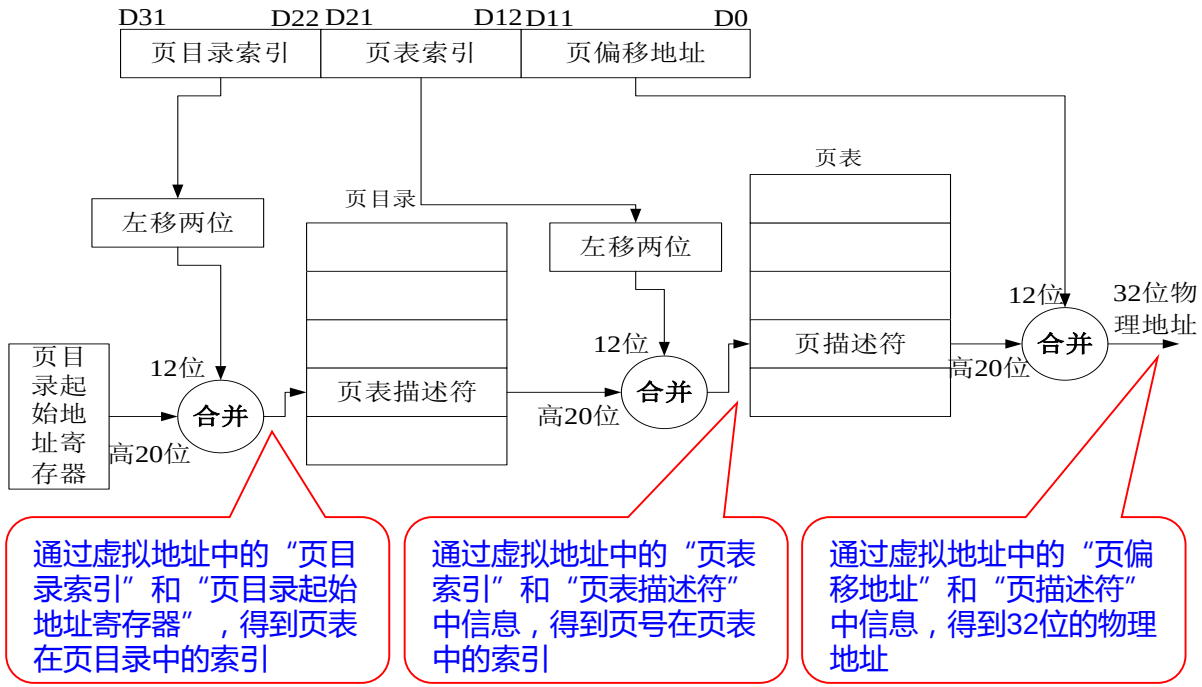
- Intel 的分页管理
  - PC上采用4KB大小的页
    - 则页内偏移地址用12位
  - 程序中仍然用32位虚拟地址来寻址内存空间
    - 虚拟地址空间为4GB
    - 32位地址分为3个部分



    - ▶12位页内偏移地址，可寻址页内任何一个存储单元
    - ▶10位页表索引，指示页号在页表中的第几项
    - ▶10位页目录索引，指示页表在页目录中的第几项
  - CPU提供保存页目录起始地址的寄存器
  - 操作系统在内存中建立页目录和页表
    - ▶每个页表描述符和页描述符4个字节，具有相同的结构

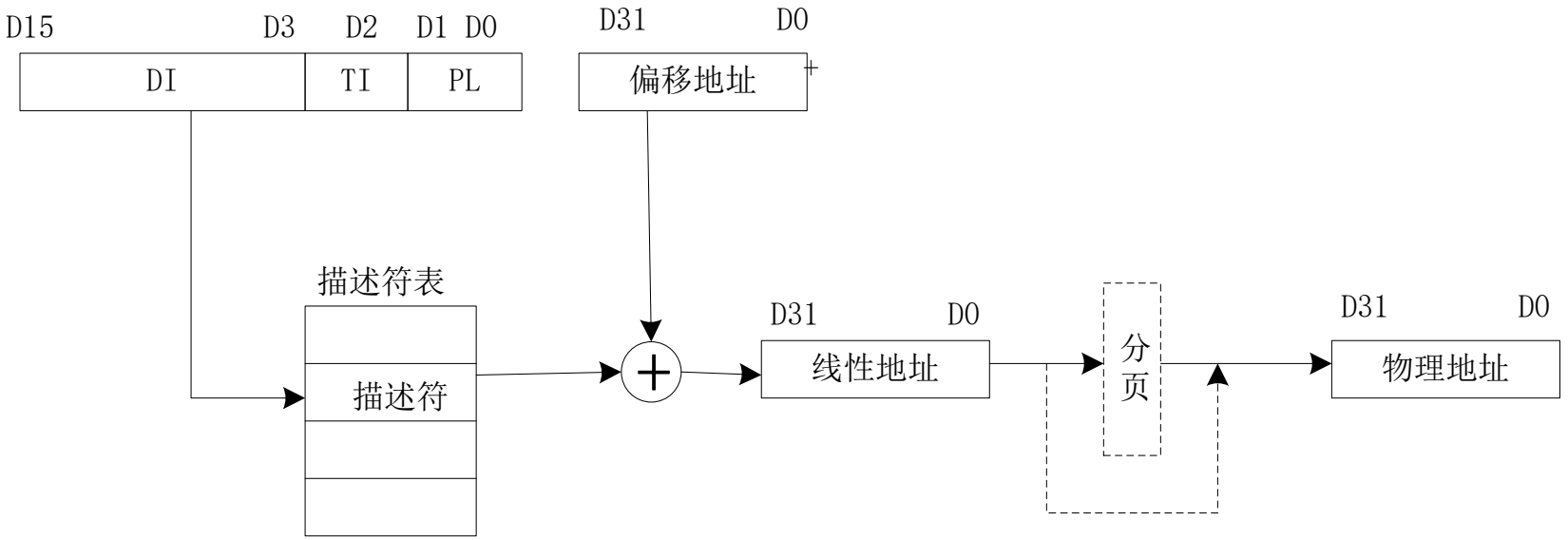


页表/页描述符结构



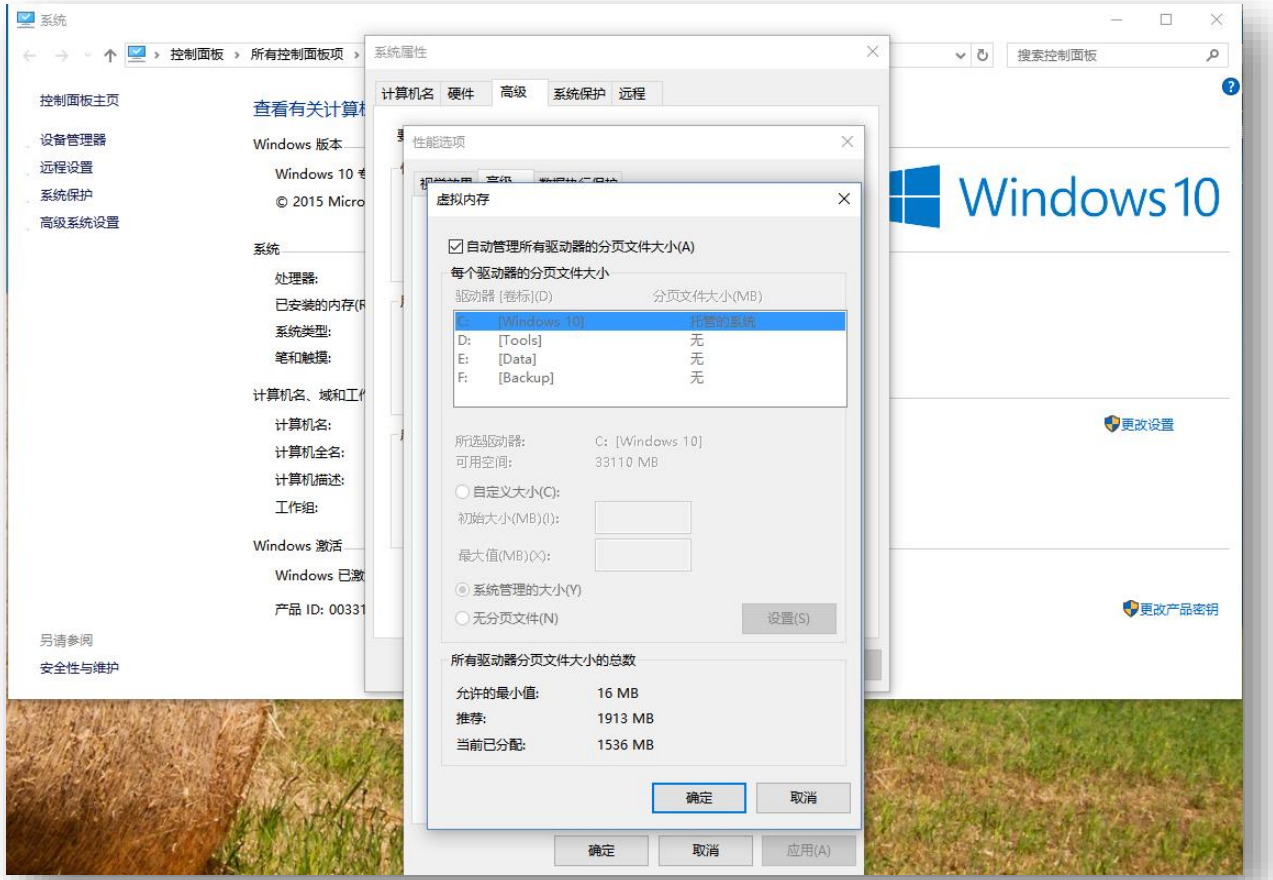
▶ 段页式管理

- 段页式管理是分段管理和分页管理的结合。
  - 物理内存被等分成相同大小的页。
  - 每个程序先按逻辑结构分段，每段再按照物理内存页大小分页。
  - 程序访问内存时给出的逻辑地址需要先经过分段管理部件转换为线性地址，然后再经过分页管理部件转换为物理地址。



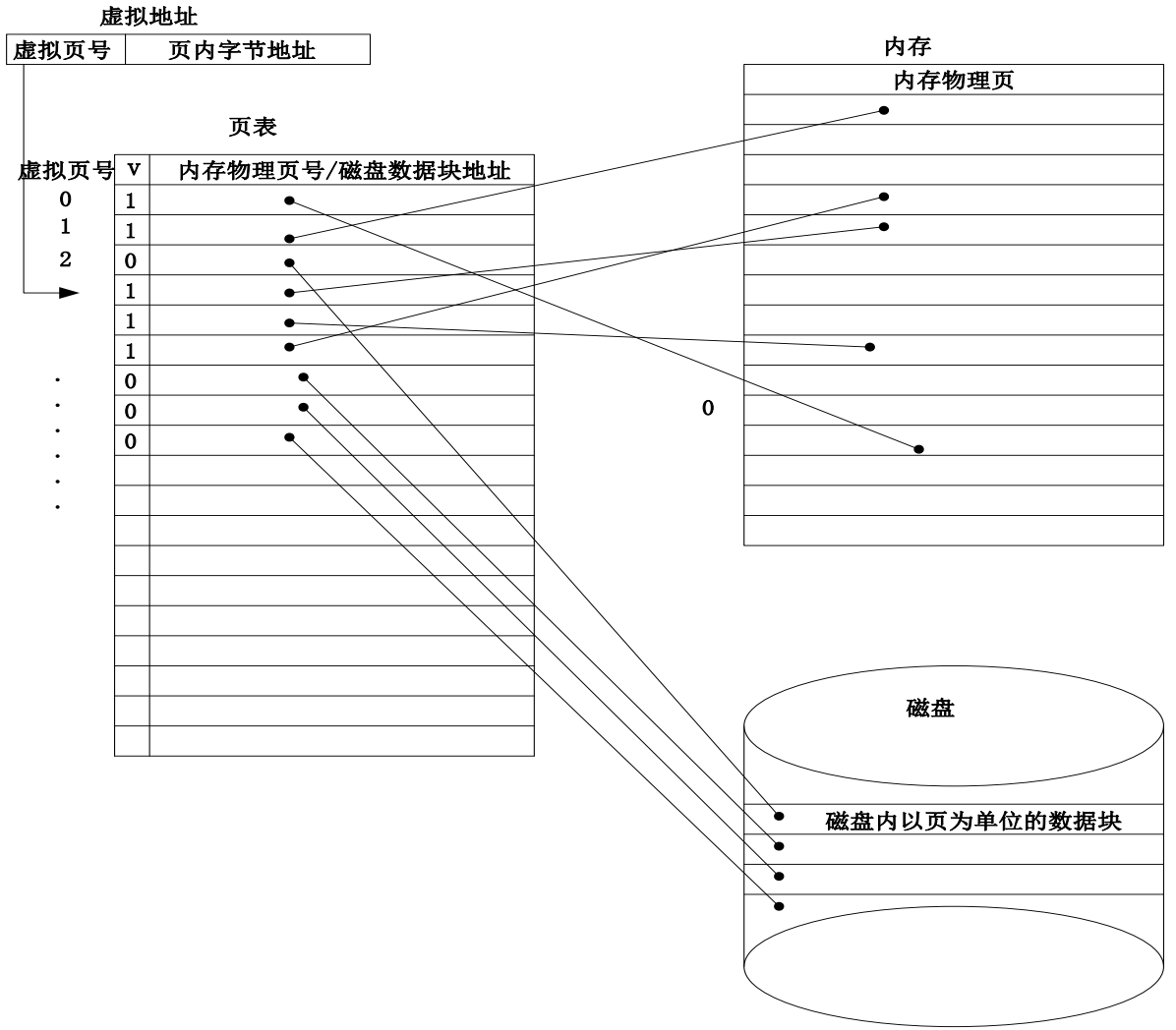
► 4.5 虚拟存储器技术

- 计算机运行的程序、数据都要加载到内存中；
- 内存容量有限，程序大小、多少无限，可能会导致内存消耗殆尽或内存访问冲突；
- 虚拟存储技术
  - 利用外部存储器充当内存，保存即将运行的程序；
  - 实现的基本原理和Cache一致：即仅把马上要运行的程序或数据存在内存中，其余部分保存在外存中；
  - 要访问的指令或数据在内存中就直接从内存中读取，如果不在则要将外存中的指令或数据调入内存。



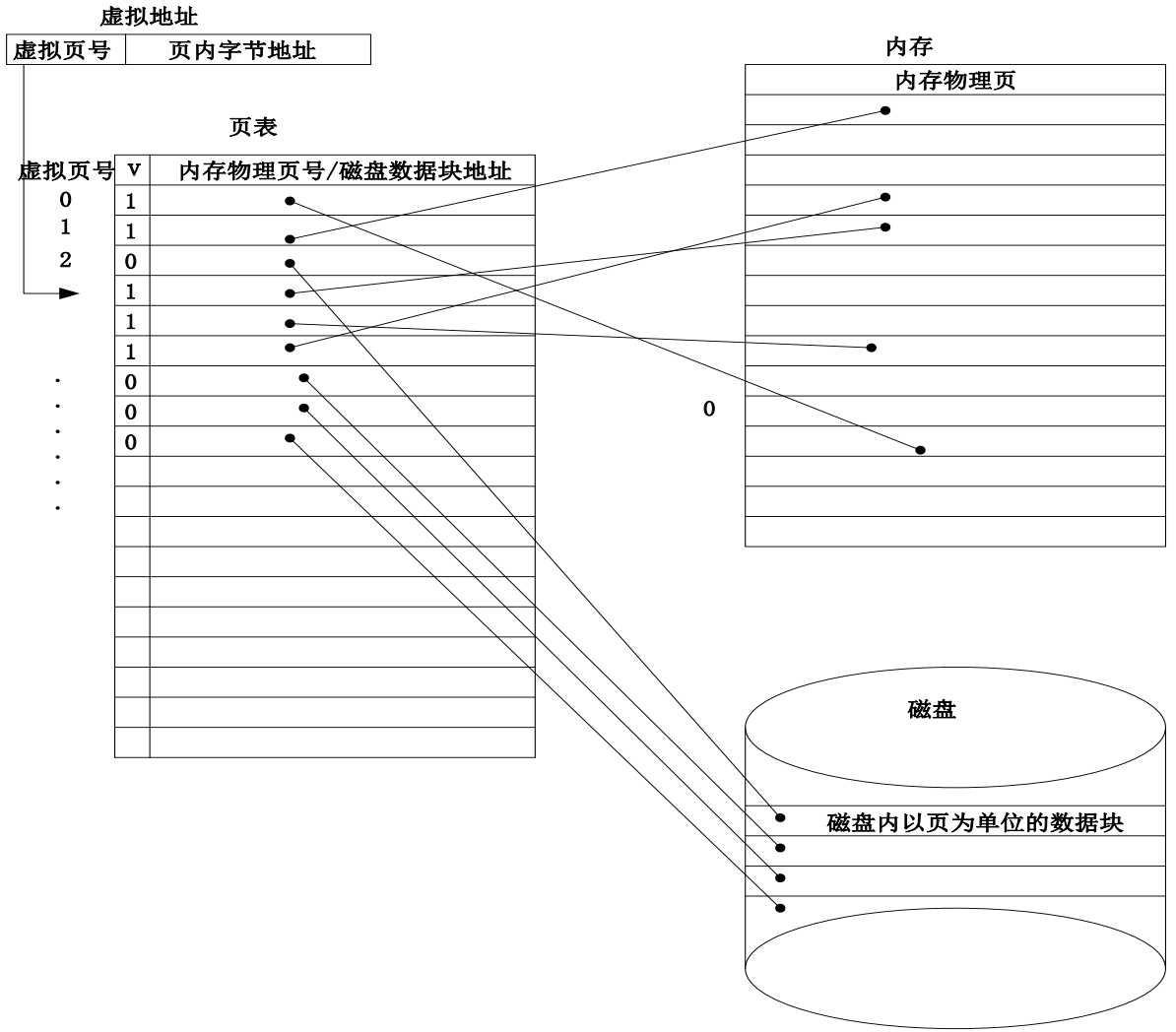
4.3 内存管理

- 虚拟存储器是由硬件和操作系统共同实现存储信息调度和管理的。
- 调度方式
  - 分页式
  - 分段式
  - 段页式
- 分页式调度方式
  - 逻辑地址空间和物理地址空间都分成固定大小的页
  - 程序给出虚拟地址，微处理器将虚拟地址分为虚拟页号和页内地址两部分
    - 通过虚拟页号作为索引查找页表，从而确定该页是否已经调入内存以及实际的内存物理页号



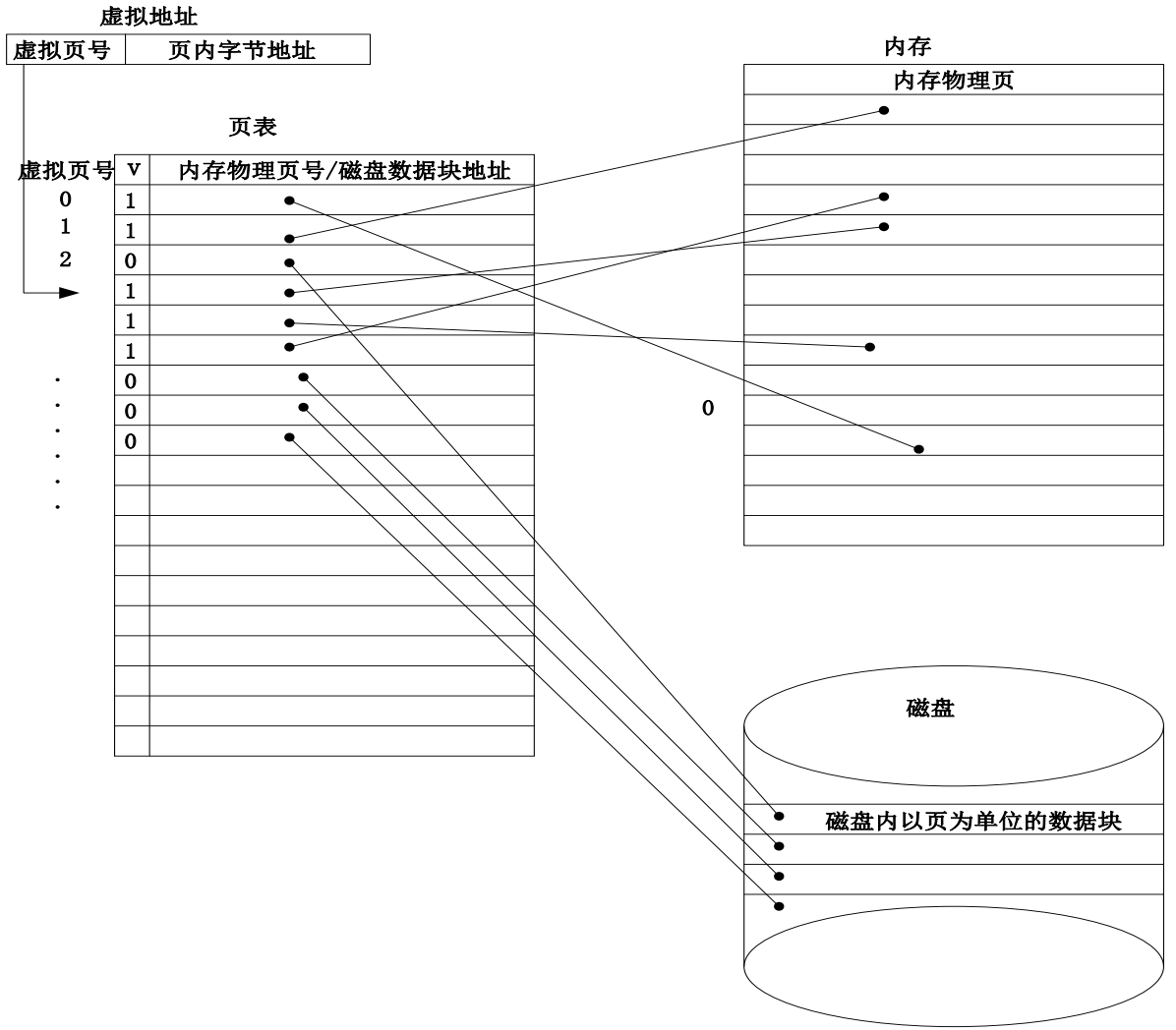
▶ 分页式调度方式

- 可执行文件会被分为很多页，在可执行文件执行过程中，它往内存中装载的单位就是页。
- 分页方法的核心思想
  - 当可执行文件执行到第 x 页时，就为第 x 页分配一个内存页 y，然后再将这个内存页添加到进程虚拟地址空间的映射表中，这个映射表就相当于一个 $y=f(x)$ 函数，应用程序通过这个映射表就可以访问 x 页关联的 y 页了。



▶ 分页式调度方式

- 不足
  - 当微处理器需要访问内存时，需要首先在内存中查找页目录和页表，然后才能形成内存单元的物理地址，最后才能访问到实际的内存数据，这种方式降低了微处理器访问内存的效率。
- 改进
  - 为弥补这个缺陷，微处理器在cache中建立映射表缓冲区（TLB）保存最近使用的内存页的映射关系来减少内存访问次数，从而提高内存数据的访问效率。



第4章 作业（二）

- 作业题（第2版，P164）
  - 12

- 要求：
  - 微助教答题；
  - 当堂提交。

Thanks