第十讲 存储系统

CACHE、虚拟存储器技术

华中科技大学
武昌 信息与通信学院
School of Electronic Information and Communications

2015年11月6日星期五

学习目标

- 了解缓存在计算机系统的作用和影响
- 高速缓存的映射策略
  - 直接映射
  - 全相联
  - 组相联
- Cache写策略、替换策略
- 虚拟存储技术

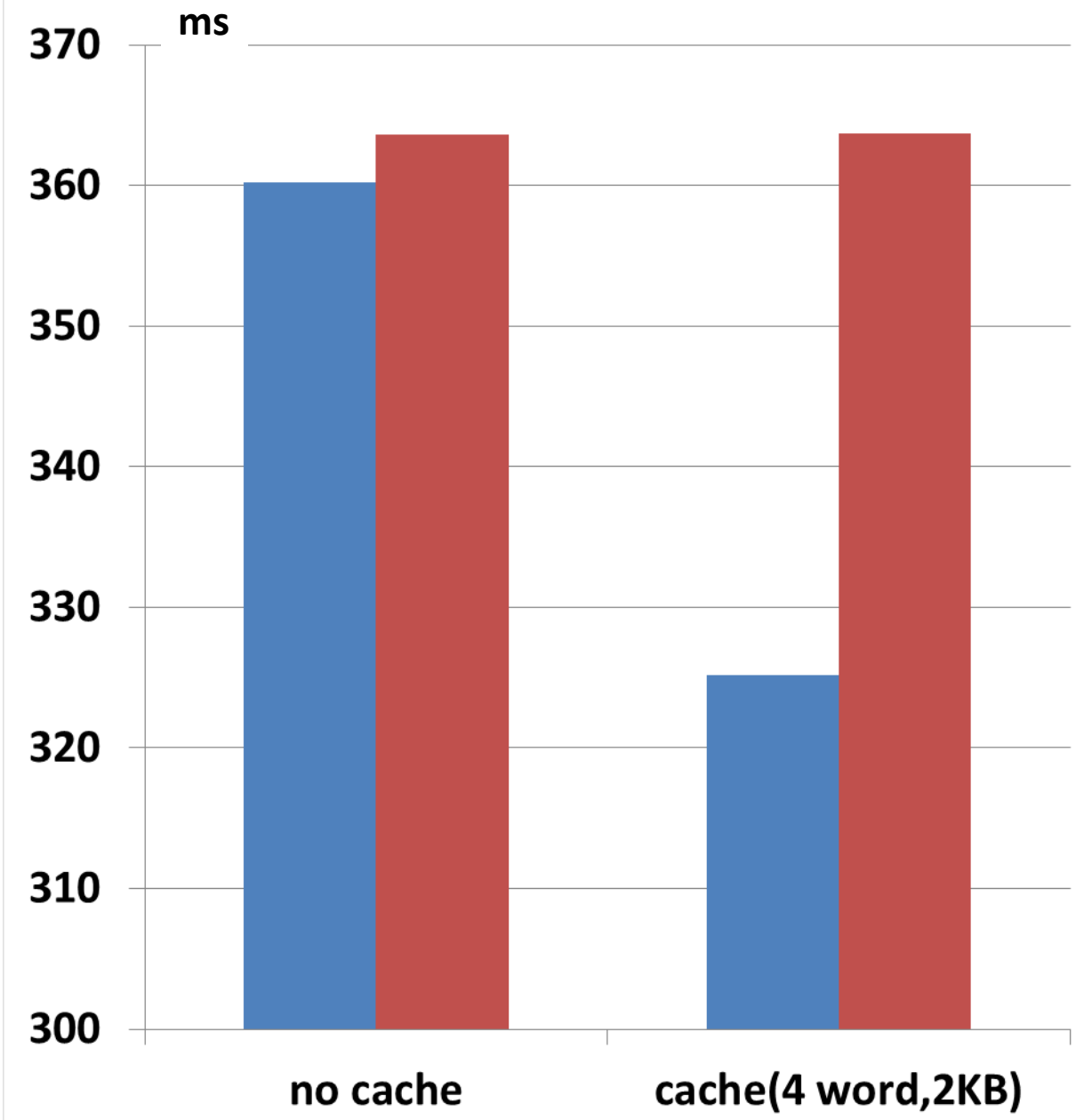
C. Can Memory Hierarchy improves the Performance

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

J first

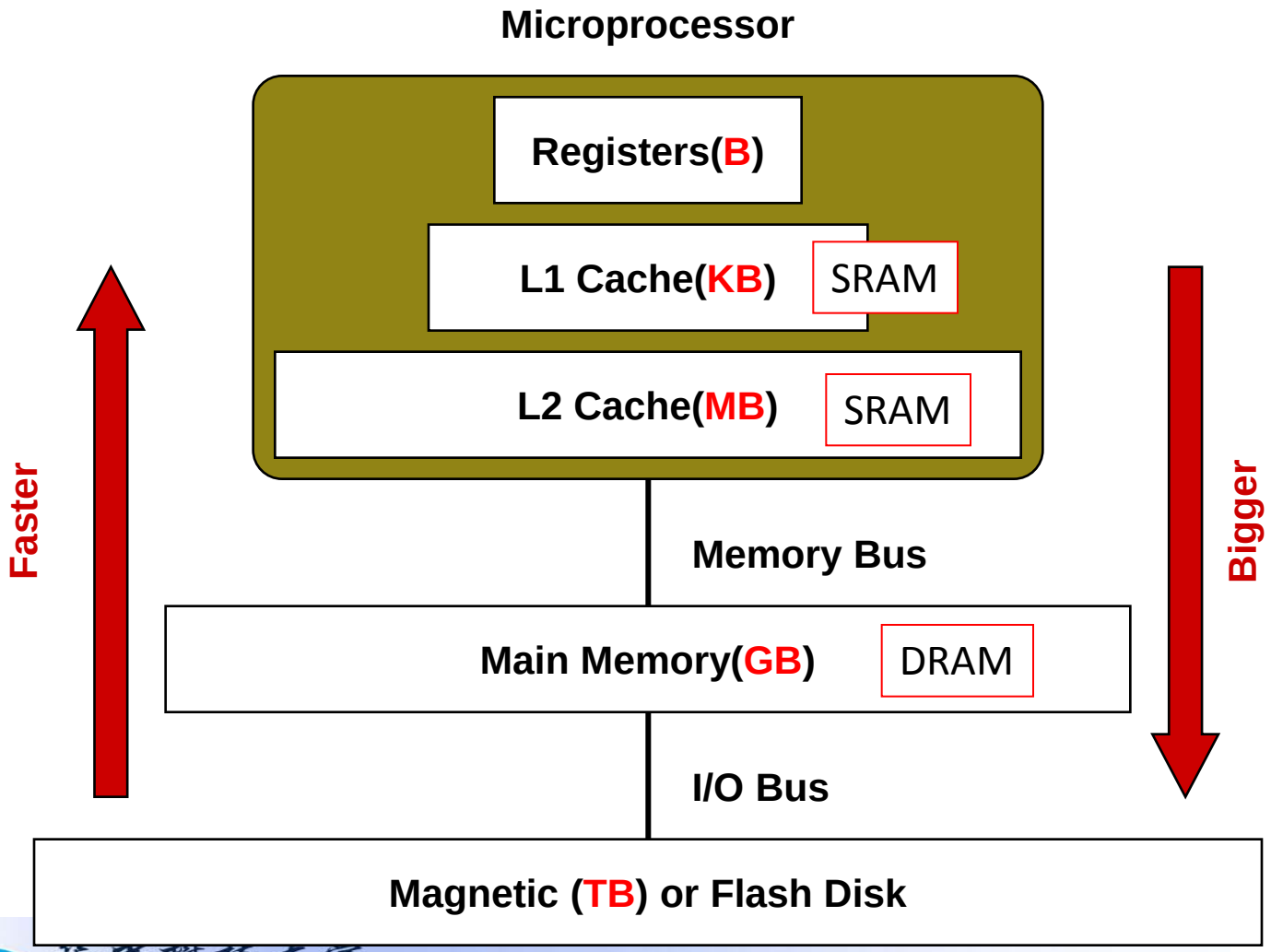
```
assign-array-rows()
{
……
  for (j=0; j<N;j++)
         for  (i=0; i<M; i++)
         sum= sum +a[i][j];
……
}
```

I first



M=N=1000
CPU f=100MHz

Typical Memory Hierarchy



4.4高速缓存原理（cache）

- Spatial Locality ——nearby items will tend to be referenced again soon.

```
for (i=0; i<N; i++)
        a[i]= …
```

- **Temporal Locality- items tend to be referenced again soon.**

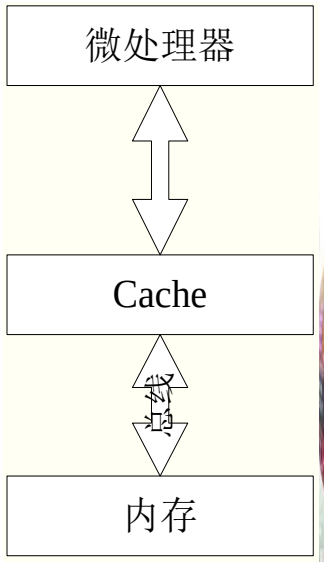
```
for (i=1; i<N; i++)
    a[i]= f(a[i-1]);
```

Take Advantage of Locality

- Temporal Locality
  - Keep most recently accessed data items closer to the processor
- Spatial Locality
  - Move blocks consists of contiguous words to the faster levels

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
- Cache非命中（Miss）
  - 不一致
- 命中率=命中cache次数/访问cache总次数
- 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。

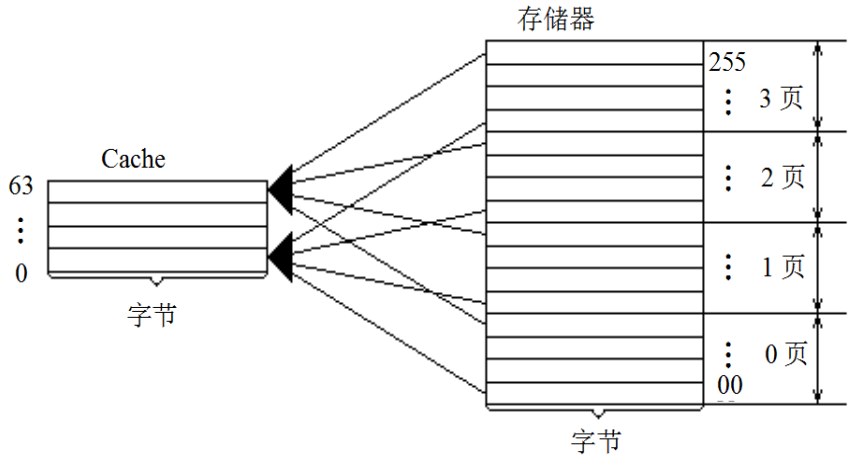


Cache构成原理

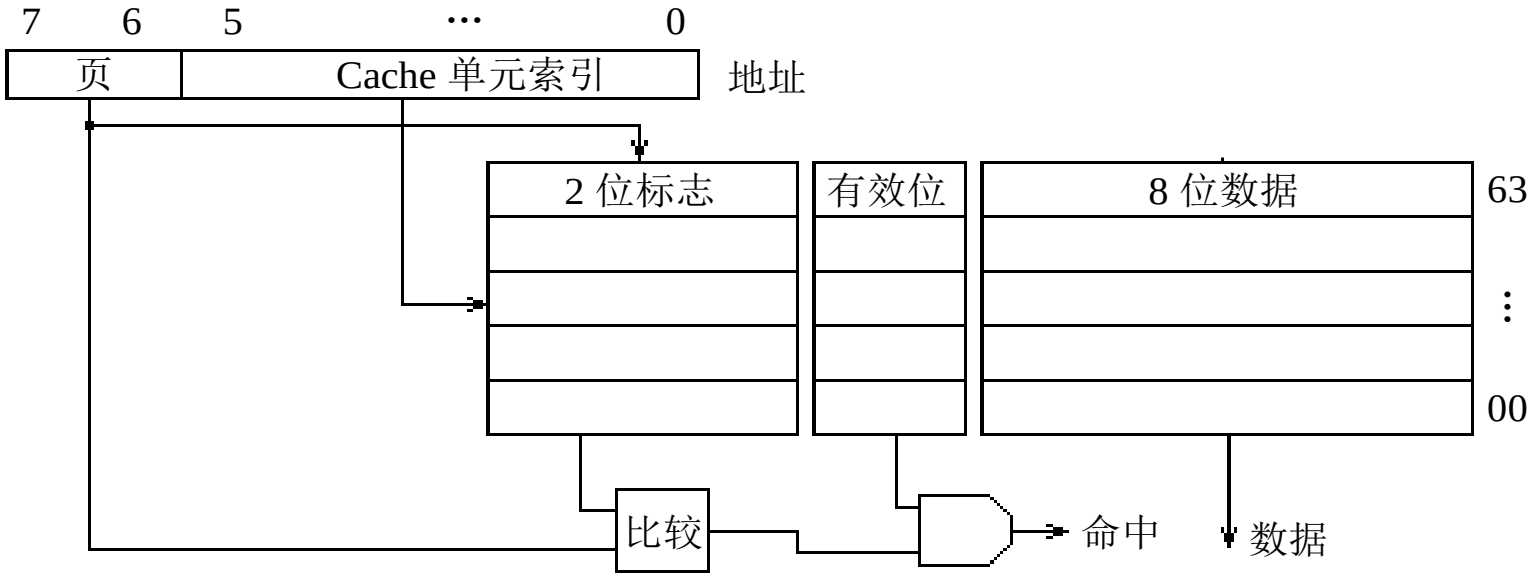
- cache存储单元与内存存储单元之间建立某种地址映像关系
  - 直接映像（Direct Mapped）、
  - 全相联（Full Associative）
  - 组相联(Set Associative)

直接映像

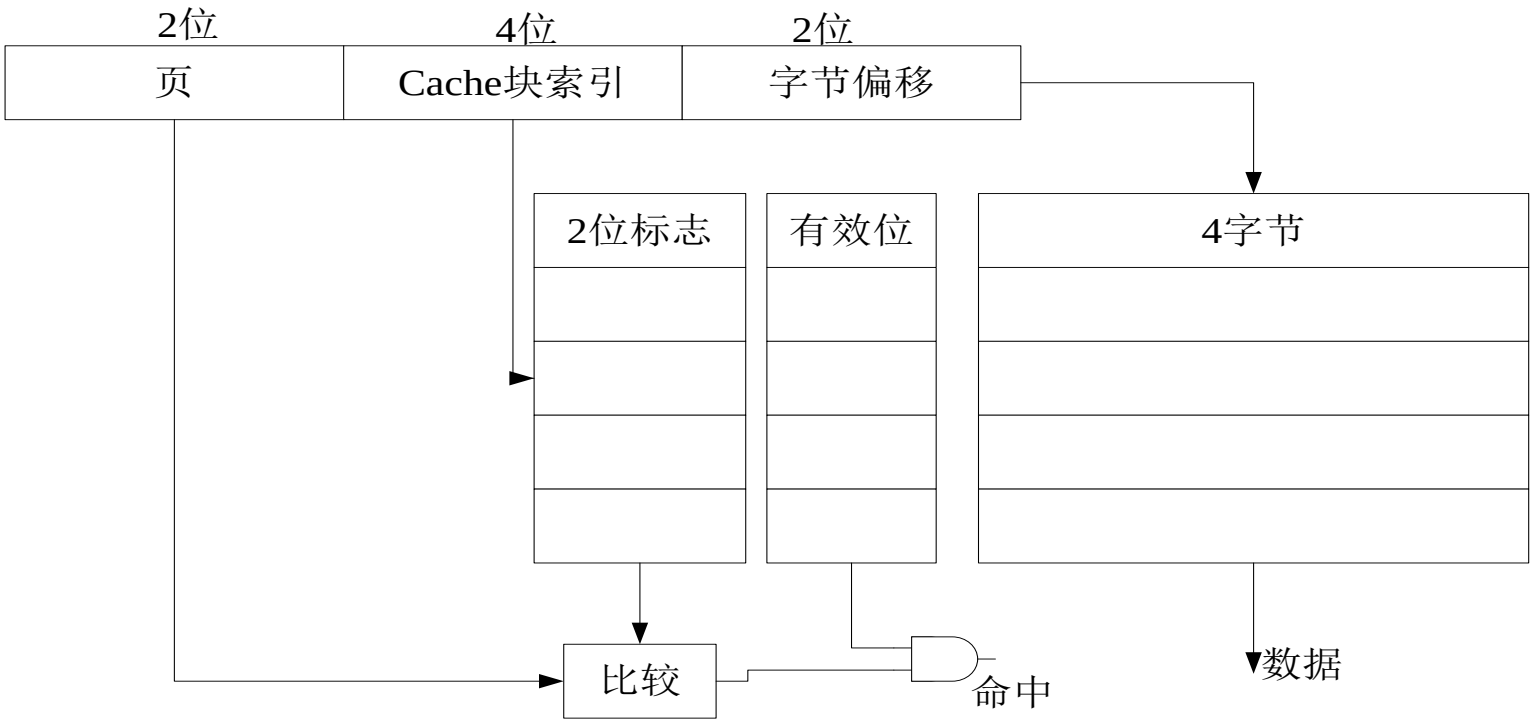
- 主存与缓存分成相同大小的数据块。
- 主存容量是缓存容量的整数倍，将主存空间按缓存的容量分成页，主存中每一页的块数与缓存的总块数相等。
- 主存中某页的一块存入缓存时只能存入缓存中块号相同的位置。



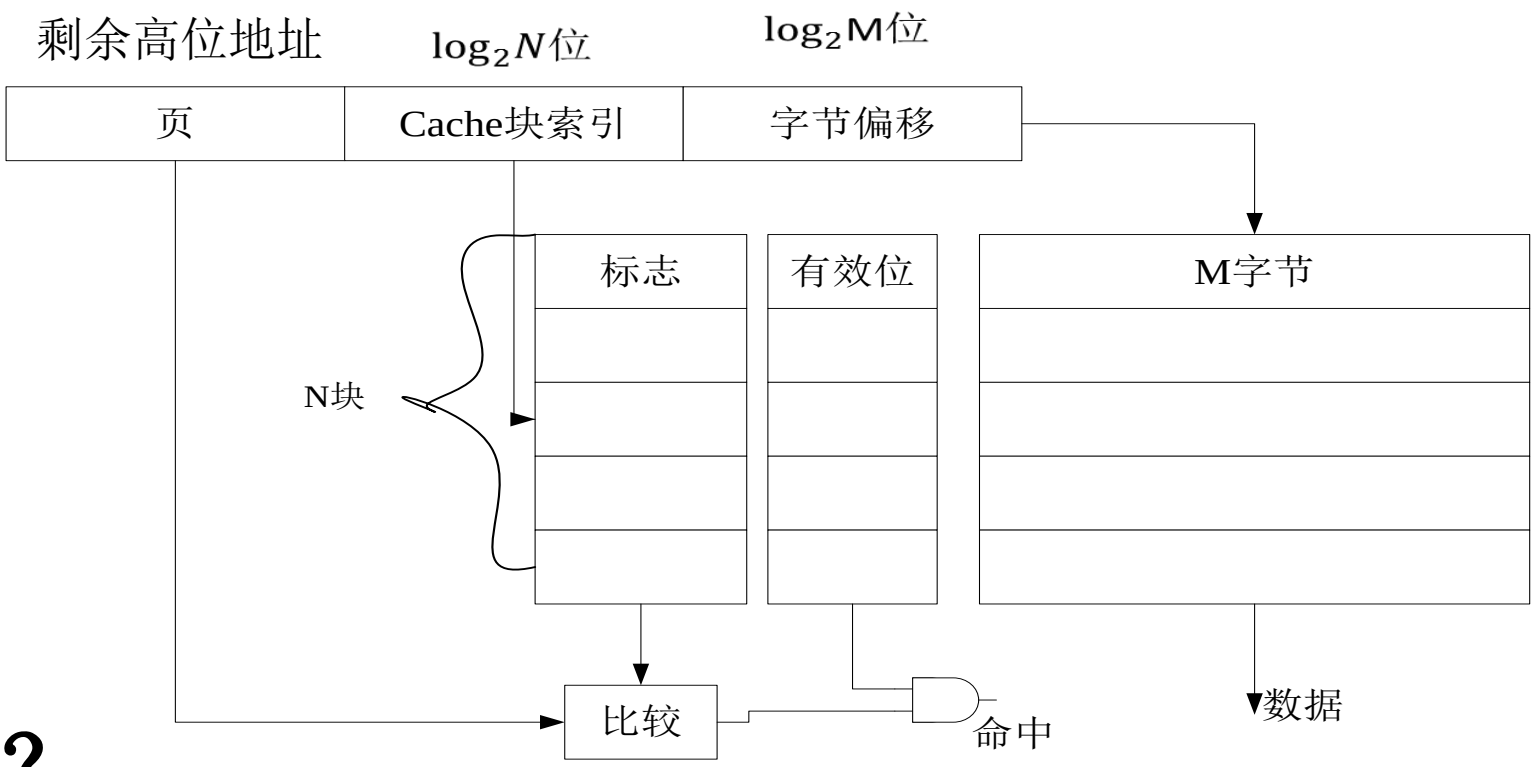
直接映射Cache结构原理框图



4字节块组织的直接映射Cache结构原理框



地址对应关系

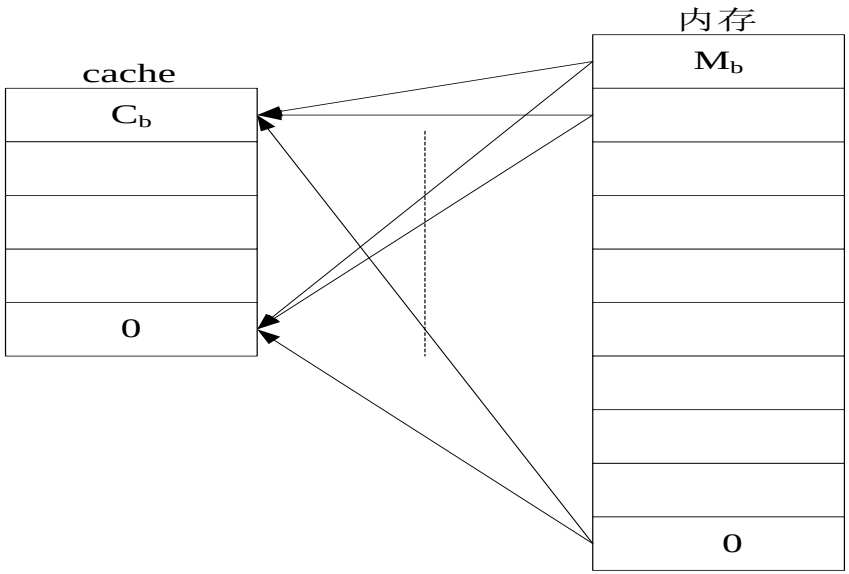


?

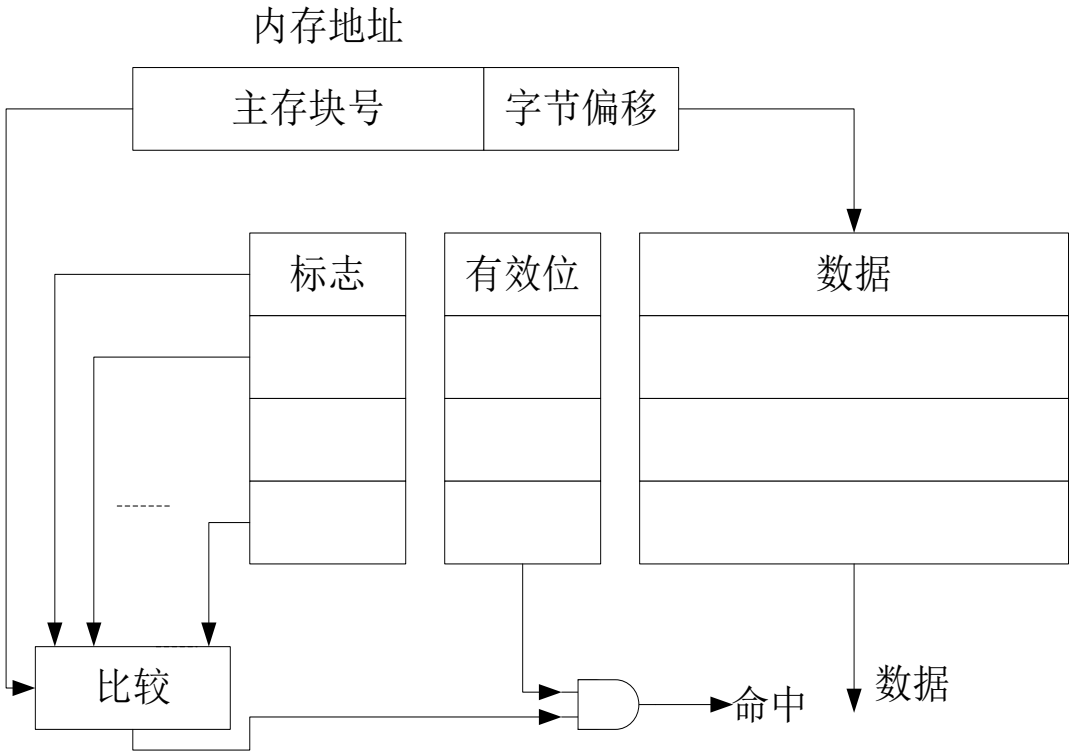
多个不同的页中处于同样行位置的数据访问比较频繁时，需要不停的更换同一个cache行的内容，cache替换操作频繁，命中率比较低。

全相联映像

- 主存与缓存分成相同大小的数据块。
- 主存的某一数据块可以装入缓存的任意一块空间中。



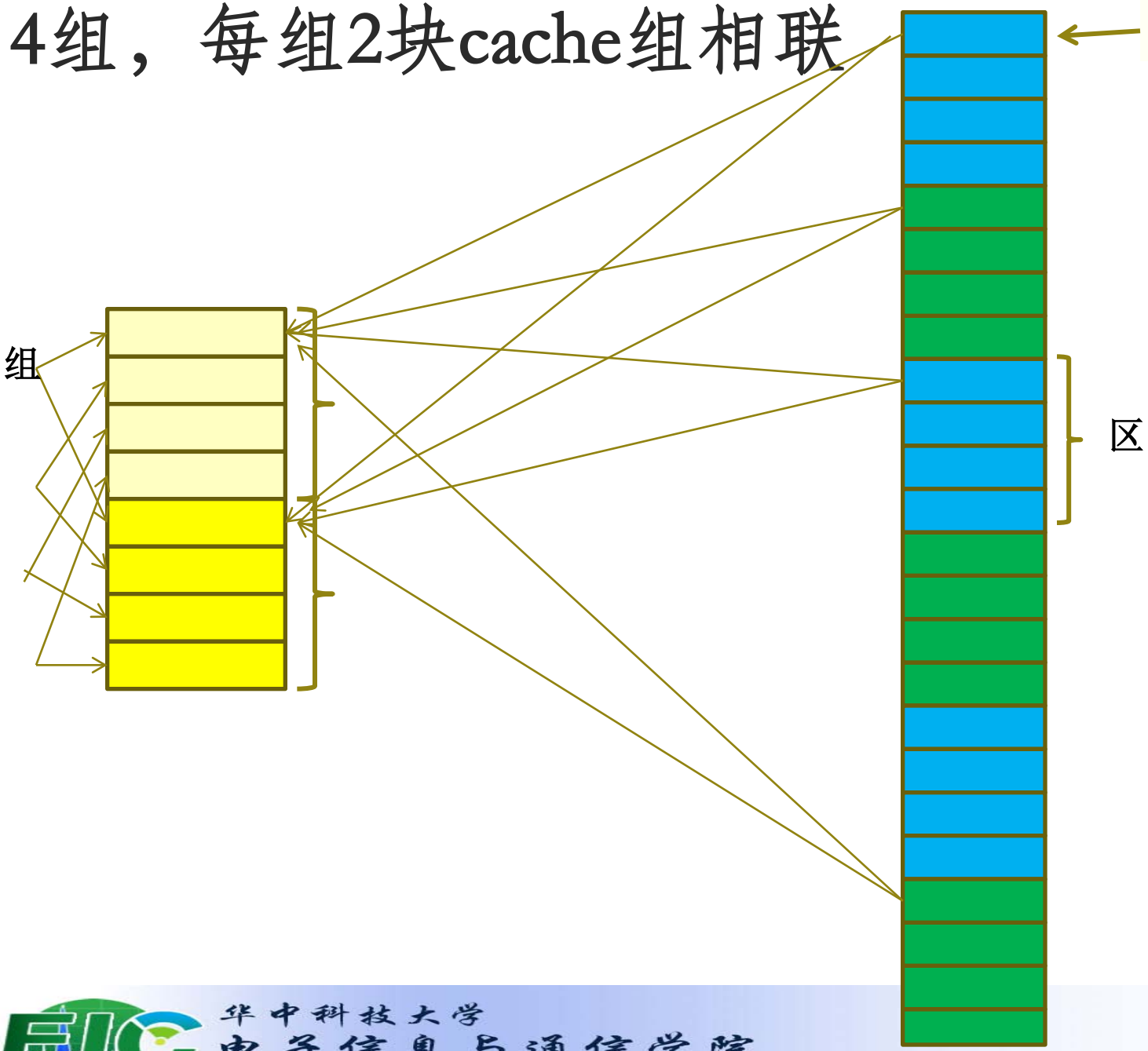
全相联映像cache结构原理



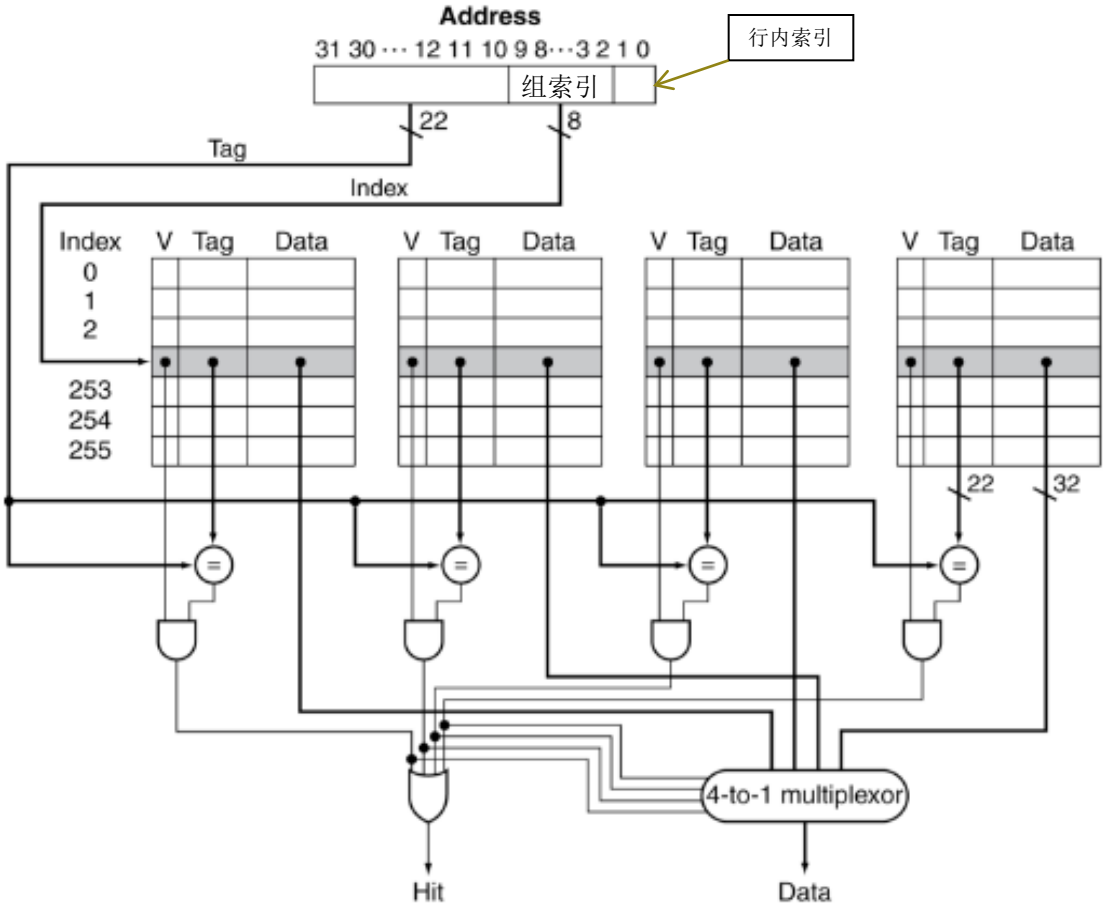
组相联映像

- 主存和Cache按同样大小划分成块。
- 主存和Cache按同样大小划分成区（区数即为路数）。各区中的块都从0开始编号
- 各区中相同块号的块属于同一组，组号即为区内的块号。
- 当主存的数据调入缓存时，主存与缓存的组号应相等，也就是主存各区中的某一块只能存入缓存中同组号的块内，但组内各块则可以任意存放，即从主存的组到Cache的组之间采用直接映象方式；在组内部采用全相联映象方式。

4组，每组2块cache组相联



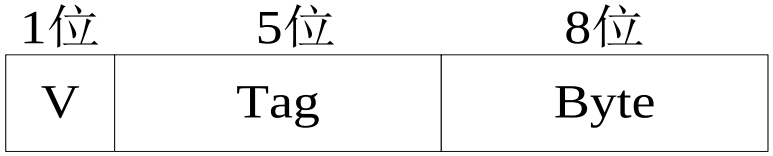
4路组相联映像cache结构原理



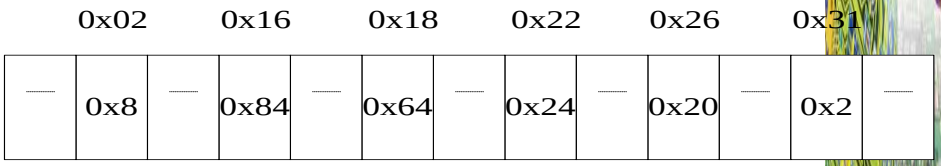
Cache读策略

- Cache行填充（Line Fill）
  - 当CPU所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
  - Cache控制器总是要将主存储器中包含该字节的一个完整的Cache行复制到Cache中

- 例4. 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。

索引	标志	有效位	数据
000		0	
001		0	
010		0	
011		0	
100		0	
101		0	
110		0	
111		0	

索引	标志	有效位	数据
000		0	
001		0	
010		0	
011		0	
100		0	
101		0	
110	00100	1	0010 0000
111		0	

CPU读取0x26单元没命中行填充

索引	标志	有效位	数据
000		0	
001		0	
010	00100	1	0010 0100
011		0	
100		0	
101		0	
110	00100	1	0010 0000
111		0	

CPU读取0x22单元没命中行填充

索引	标志	有效位	数据
000	00011	1	0110 0100
001		0	
010	00100	1	0010 0100
011		0	
100		0	
101		0	
110	00100	1	0010 0000
111		0	

CPU读取0x18单元没命中行填充

索引	标志	有效位	数据
000	00011	1	01100100
001		0	
010	00100	1	0010 0100
011		0	
100		0	
101		0	
110	00010	1	1000 0100
111		0	

CPU读取0x16单元没命中行替换

索引	标志	有效位	数据
000	00011	1	0110 0100
001		0	
010	00000	1	0000 1000
011		0	
100		0	
101		0	
110	00010	1	10000100
111		0	

CPU读取0x02单元没命中行替换

Cache写策略

- 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
  - 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
- 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

- 随机替换
- 先入先出（FIFO）替换
- 最近最少使用（LRU）替换（最常采用）

- 例4.5 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块，每块存储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

访问的内存地址相对于直接映射cache的行索引

内存地址	对应的cache行索引
0	（0%4）=0
8	（8%4）=0
6	（6%4）=2

块索引	
00	块0
01	块1
10	块2
11	块3

直接映射下cache的填充过程

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]	

命中0次

全相联cache

访问内存地址	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]	

命中2次

2路组相联cache

内存地址相对于2路组相联cache的块索引

内存地址	对应cache的组索引
0	（0%2）=0
8	（8%2）=0
6	（6%2）=0

块索引		
第1组 00	块0	块0
第2组 01	块1	块1

2路组相联cache的填充过程（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	
		块0	块0	块1	块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]		

命中1次

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

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[1][0] a[1][1]						
a[1][1]	中								
a[2][0]	否			a[2][0] a[2][1]					
a[2][1]	中								
a[3][0]	否				a[3][0] a[3][1]				
a[3][1]	中								
a[4][0]	否					a[4][0] a[4][1]			
a[4][1]	中								
a[5][0]	否						a[5][0] a[5][1]		
a[5][1]	中								
a[6][0]	否							a[6][0] a[6][1]	
a[6][1]	中								
a[7][0]	否	a[0][0] a[0][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[8][1]	中								
a[9][0]	否		a[9][0] a[9][1]	a[2][0] a[2][1]					
a[9][1]	中								
a[10][0]	否			a[10][0] a[10][1]	a[3][0] a[3][1]				
a[10][1]	中								
a[11][0]	否				a[11][0] a[11][1]	a[4][0] a[4][1]			
a[11][1]	中								
a[12][0]	否					a[12][0] a[12][1]	a[5][0] a[5][1]		
a[12][1]	中								
a[13][0]	否						a[13][0] a[13][1]	a[6][0] a[6][1]	
a[13][1]	中								
a[14][0]	否							a[14][0] a[14][1]	a[7][0] a[7][1]
a[14][1]	中								
a[15][0]	否								a[15][0] a[15][1]
a[15][1]	中								

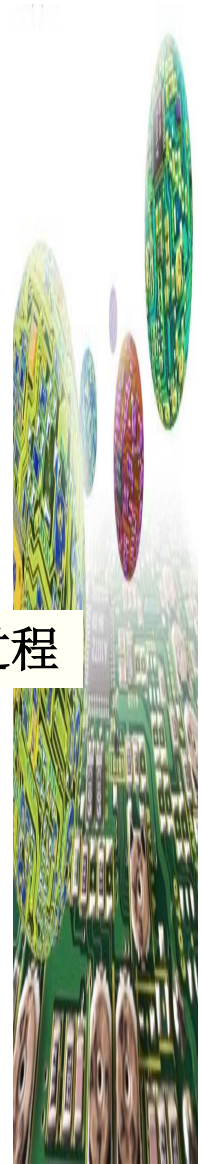
列优先访问填充过程

命中率50%

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]

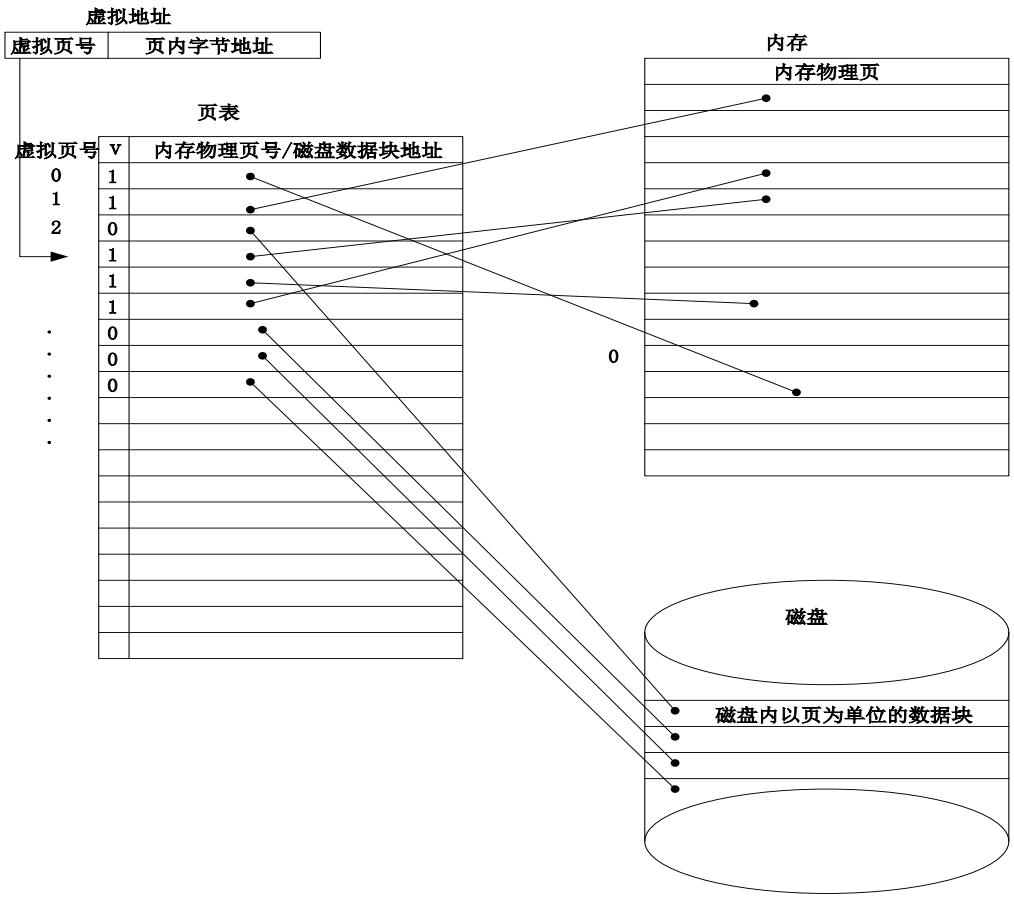
行优先访问填充过程

命中率 0

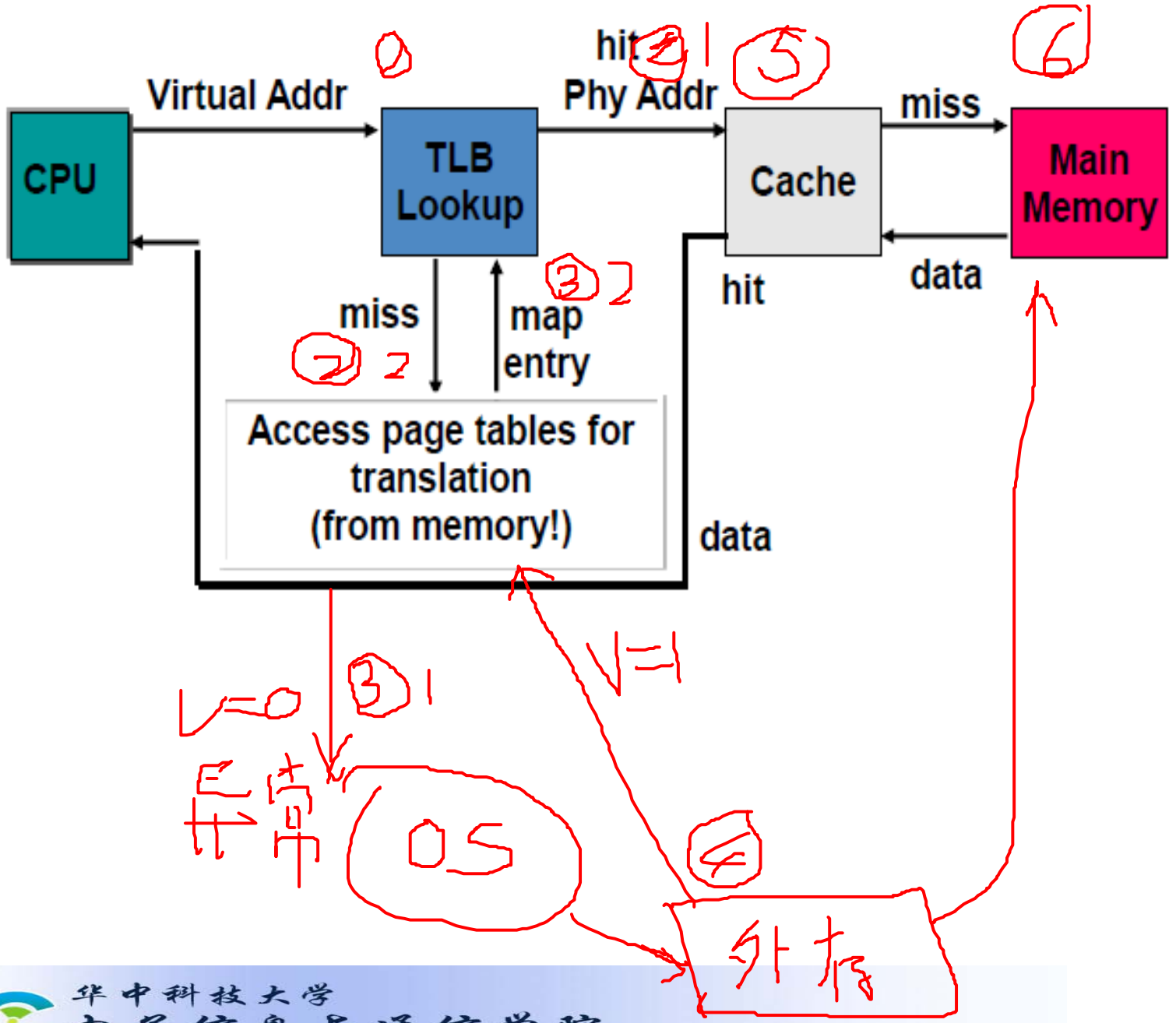


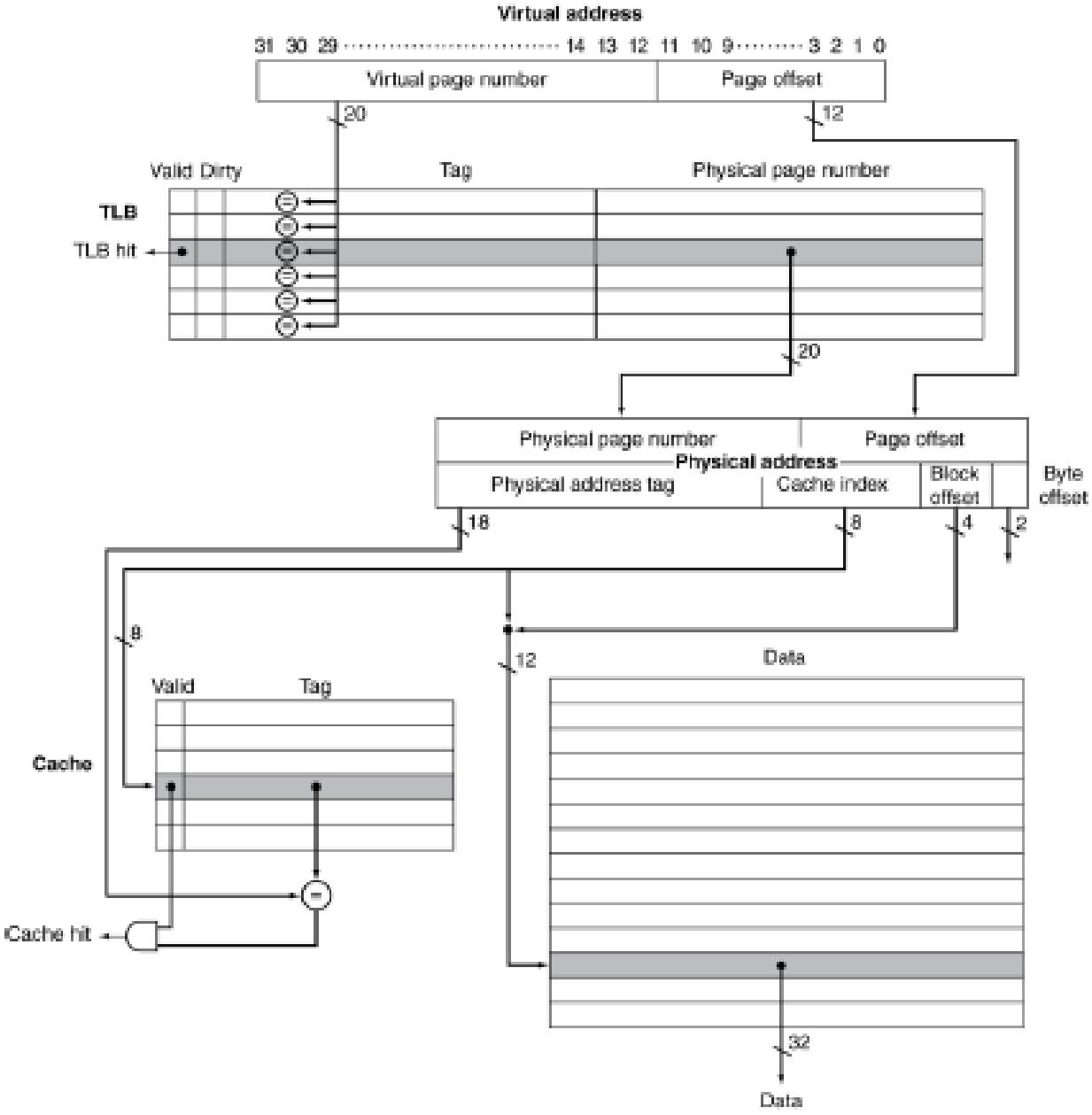
- cache块大小变为8字节，一次将拷贝两行数组元素进入cache块，因此，程序段A顺序访问两行数组元素时，只有第一个元素未命中，后面三个元素命中，命中率提高为75%，而程序段B的cache命中率提高为50%.

Chap4 虚拟存储器技术



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。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。


Virtual Addr
Phy Addr
hit
miss
CPU
TLB Lookup
Cache
Main Memory
data
hit
miss
map entry
Access page tables for translation (from memory!)
data
V=0
V=1
OS
外存


Virtual address
31 30 29 ..... 14 13 12 11 10 9 ..... 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Block offset
Byte offset
18
8
4
2
8
12
Data
Valid
Tag
Cache
Cache hit
32
Data

作业

- 10，11