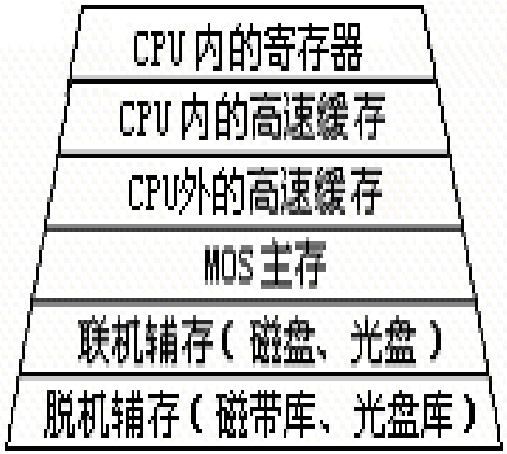
第四章 存储系统

内存管理及虚拟存储技术

学习目标

- 了解计算机系统存储系统的分级结构特点
- 内存的组织管理方式
- Cache的组织管理方式
- 虚拟存储器的基本原理

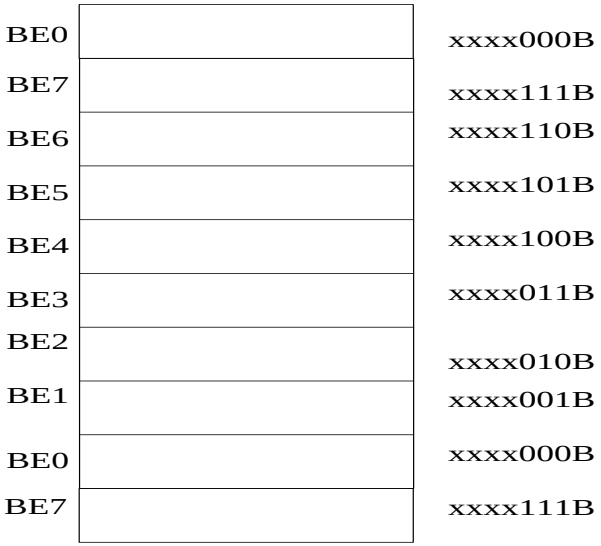
4.1计算机存储系统构成



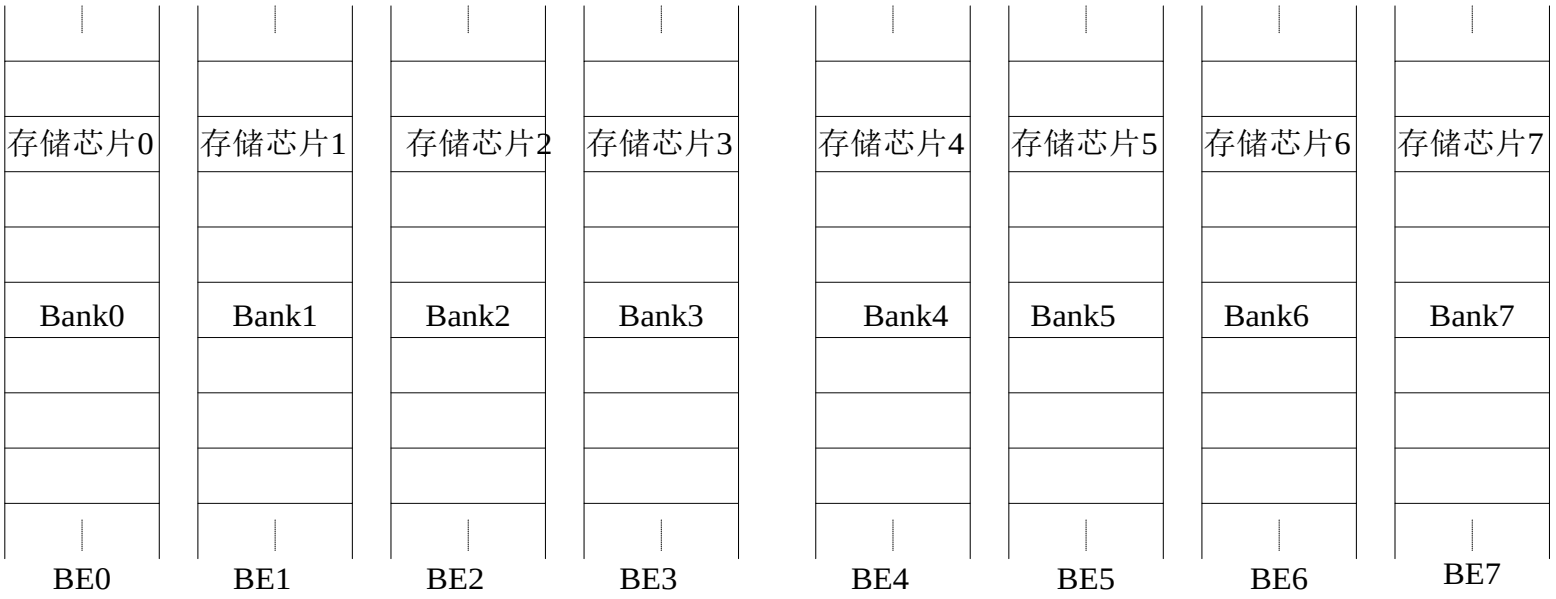
存储器类型	CPU寄存器	Cache	主存储器	硬盘等外部存储器
访问速度	ns级	ns级	≤70 ns	ms级
	→自左至右速率逐层下降→			
存储容量	→自左至右容量逐层增加→			
	数十数百个	≤2 MB	数100 MB	≥1 GB

4.2 内部存储器

- 存储器分块组织
  - 大部分微处理器支持访问8位，16位，32位到64位不等位宽的数据



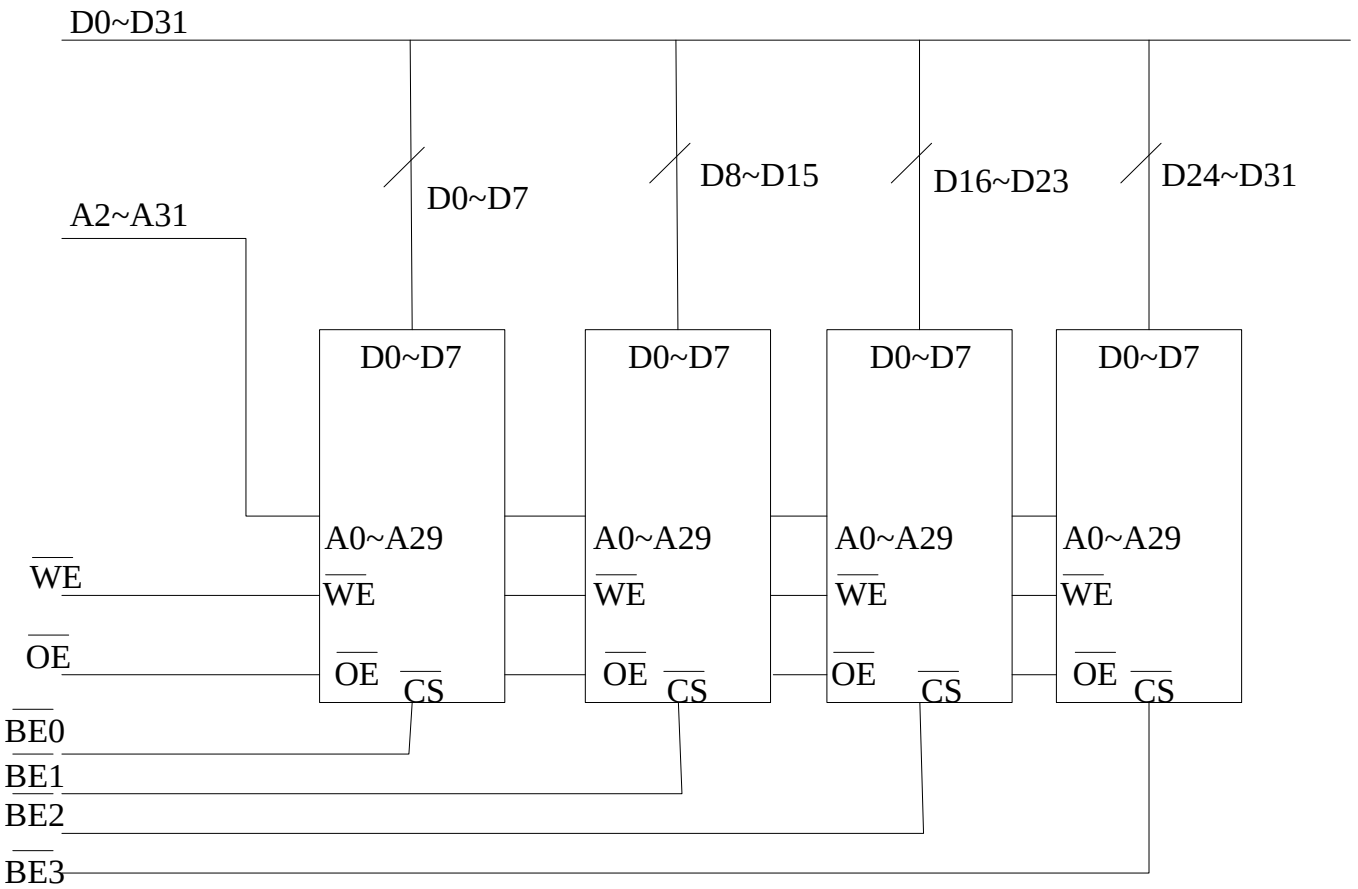
存储器分块示意图


存储芯片0
存储芯片1
存储芯片2
存储芯片3
存储芯片4
存储芯片5
存储芯片6
存储芯片7
Bank0
Bank1
Bank2
Bank3
Bank4
Bank5
Bank6
Bank7
BE0
BE1
BE2
BE3
BE4
BE5
BE6
BE7

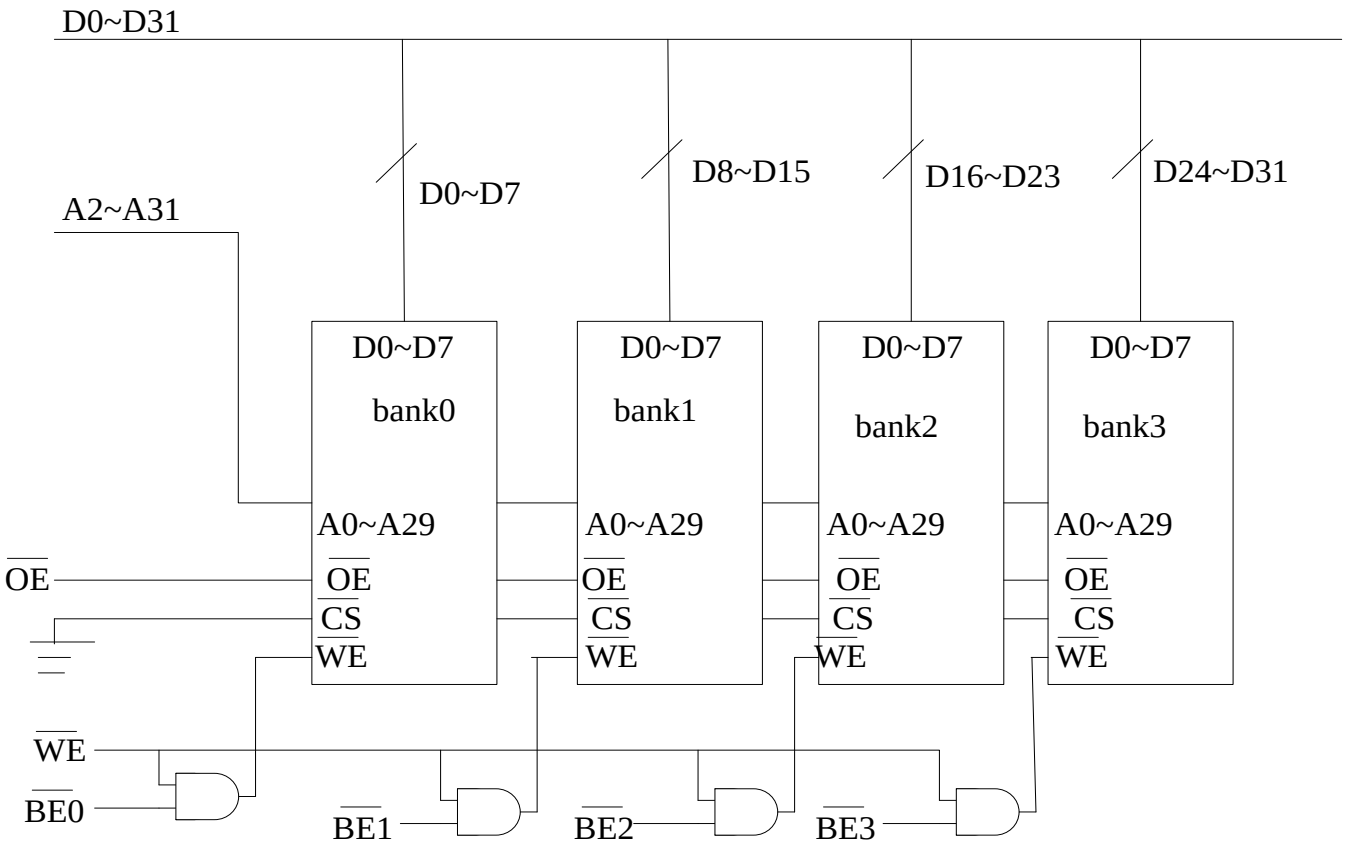
- 实现多类型数据访问的存储器接口控制有两种方式
  - 使用字节使能信号与高位地址译码产生存储芯片片选信号。
  - 使用字节使能信号与存储器写控制信号译码产生存储芯片写控制信号。

- 例4．1为一个32位的微处理器设计一个存储空间为4GB的SRAM存储器，要求支持字节、半字、字类型数据访问，采用1G*8bit的SRAM存储芯片，如何设计该存储器？
  - 存储器分为4块，表示存储器的A0~A1无效，存储芯片的A0~A29对应连接到存储器的A2~A31上

采用BE0~BE3分别控制4片芯片的片选线



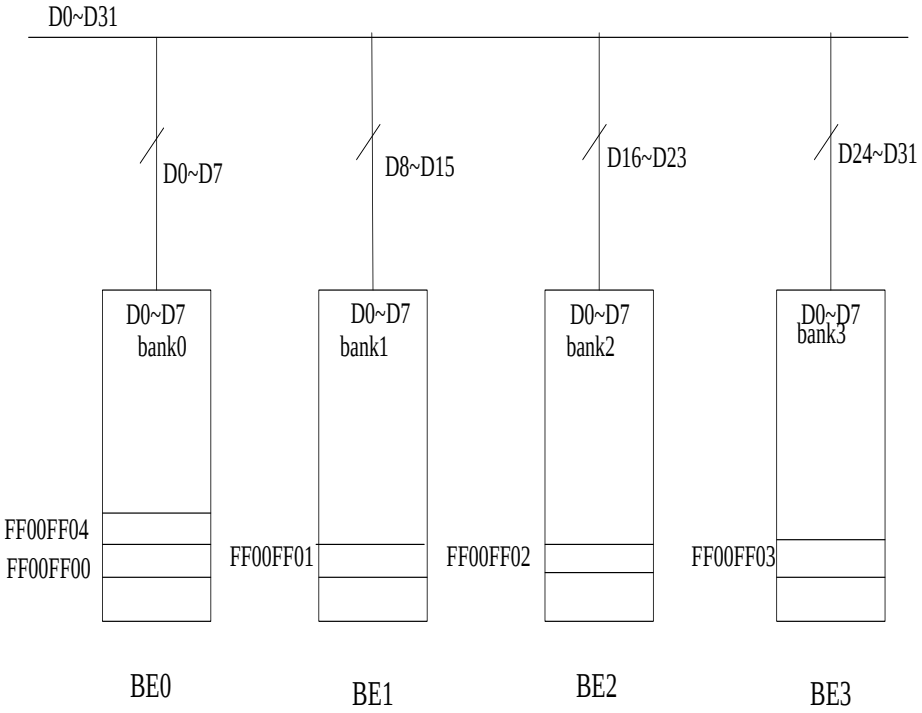
BE信号与存储器写信号控制存储芯片写信号实现不同类型的数据访问

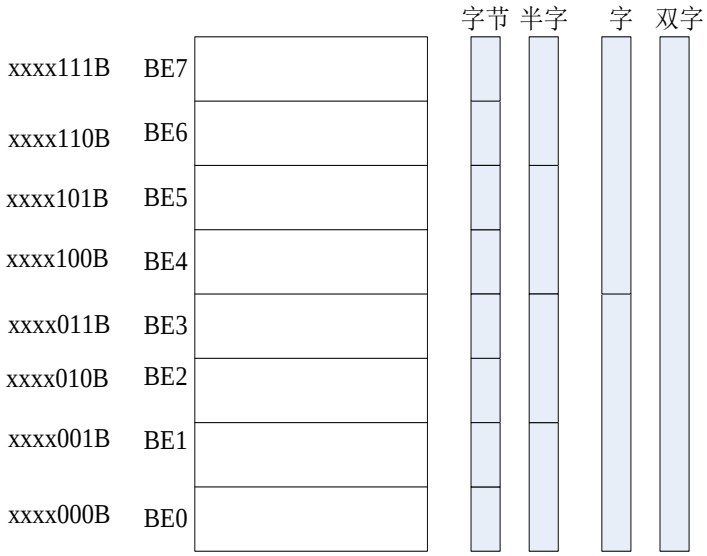


内存条

- DIMM内存条由8片8位数据宽度的同型号IC芯片组成，有的则由9片组成，增加的1片作校验位用。

内存访问边界对齐



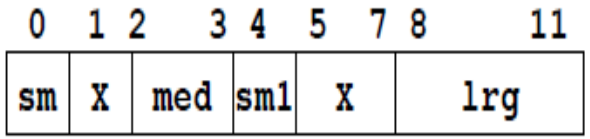


- 一次总线操作实现半字类型的数据访问要求字节使能信号从偶字节使能信号开始的连续两个字节有效，
- 一次总线操作实现字类型的数据访问要求字节使能信号从4字节使能信号开始的连续4个字节有效，
- 一次总线操作实现双字类型的数据访问要求字节使能信号从8字节使能信号开始的连续8个字节有效

- 例4．2 已知某32位计算机系统，编译器采用边界对其方式为数据分配内存空间，若定义了以下数据结构：

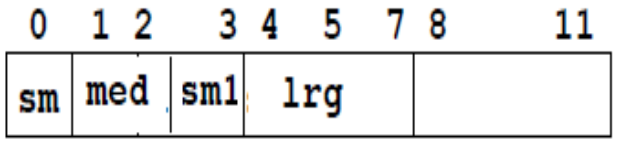
```
struct foo {
char sm; /*1字节*/
short med; /*2 字节*/
char sm1; /*1字节*/
int lrg; /*4字节*/
}
```

```
struct foo1 {
char sm; /*1字节*/
char sm1; /*1字节*/
short med; /*2 字节*/
int lrg; /*4字节*/
}
```



foo边界对齐的内存映像

浪费内存空间

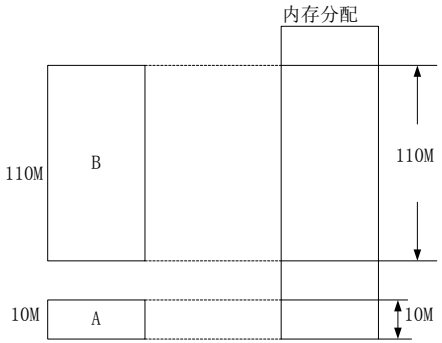


foo非边界对其的内存映像

增大访问时延

4.3内存管理

- 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）
- 程序运行的地址不确定（软件设计复杂）



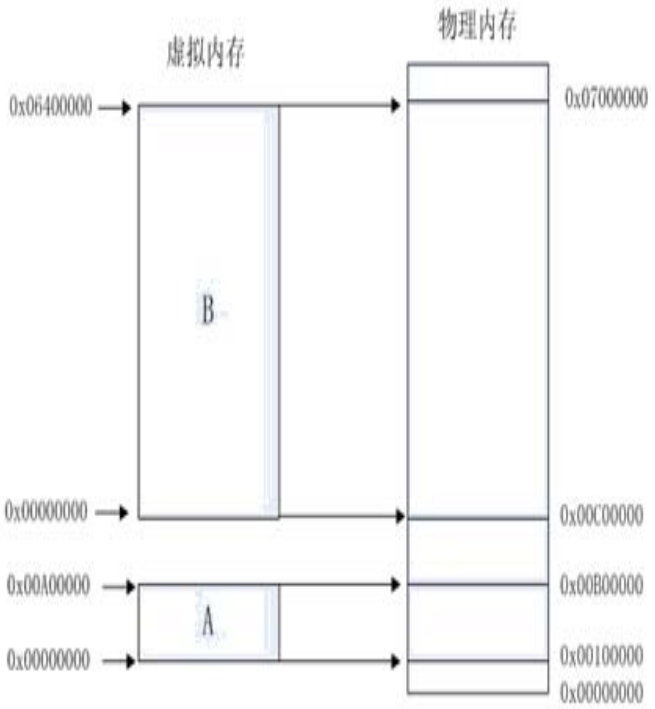
增加一个中间层，利用一种间接地址访问方法访问物理内存
程序中访问的内存地址不再是实际物理内存地址，而是一个虚拟地址，
然后由操作系统将这个虚拟地址映射到适当的物理内存地址上

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

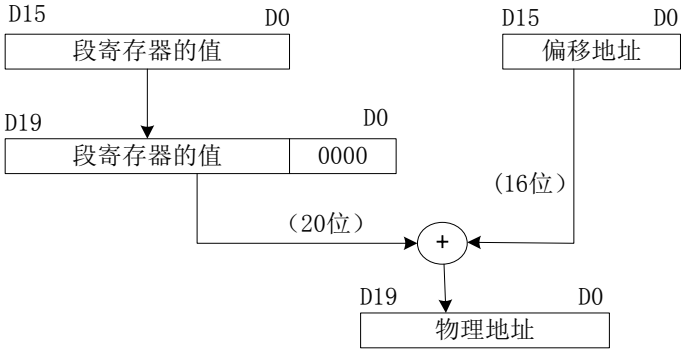
- 在虚拟地址空间和物理地址空间之间做一一映射。
- 比如说虚拟地址空间中某个10M大小的空间映射到物理地址空间中某个10M大小的空间
- 应用程序并不关心进程A究竟被映射到物理内存的哪块区域上，只需利用偏移地址访问内存单元即可



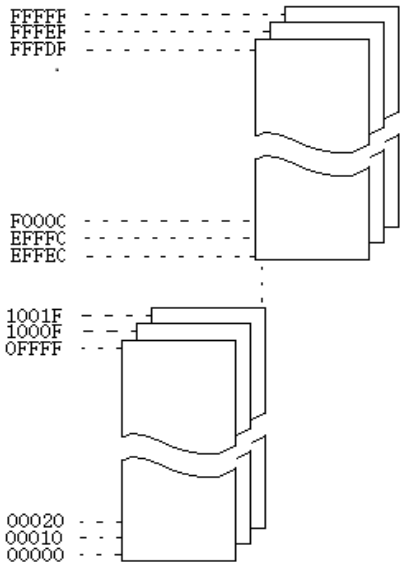
当微处理器要访问程序中的某个内存单元时，
将利用该程序映射到的物理地址空间段的起始地址（段地址）和偏移地址相结合的方式来实现寻址

分段实例

- Intel 实地址模式
- 保护模式



物理地址的形成



实地址模式分段

保护模式

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

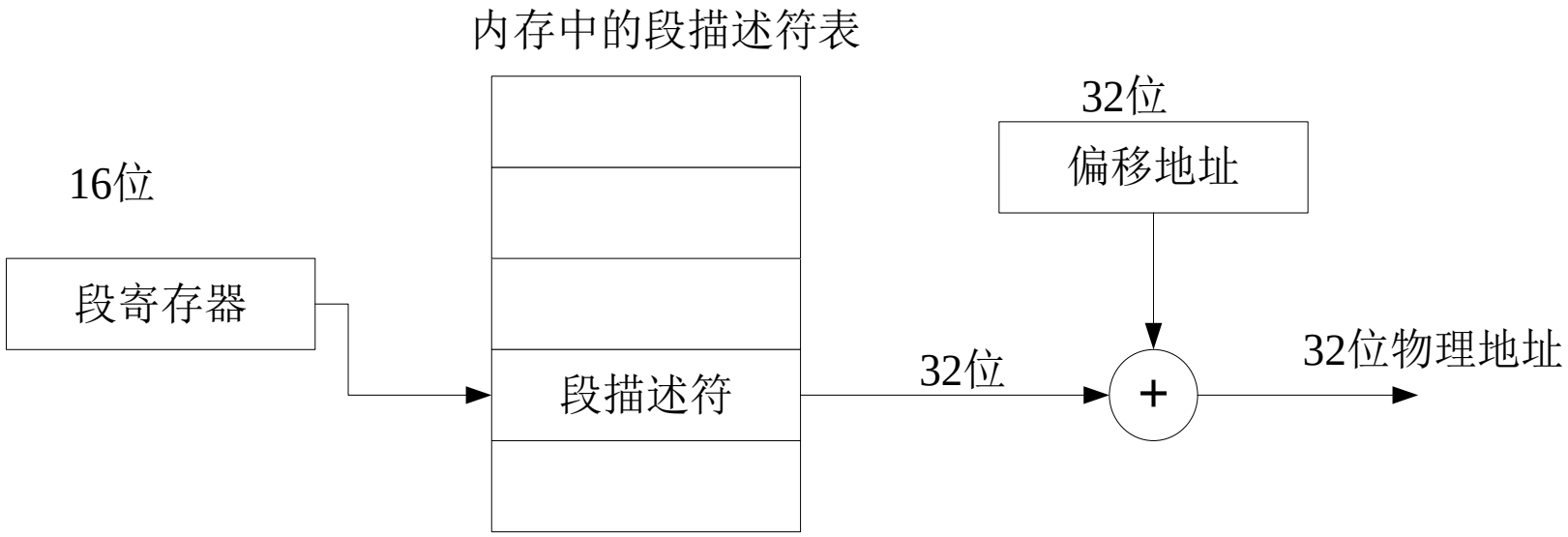
段描述符结构

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。

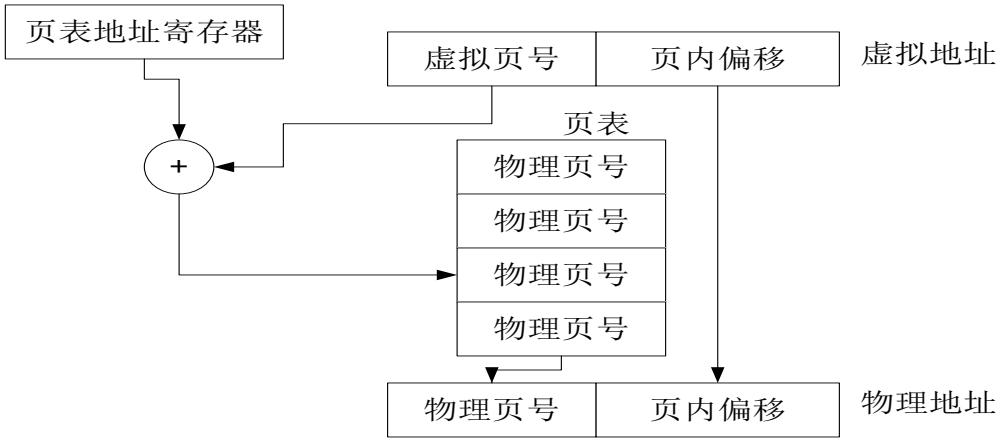
- 已知某32位intel微处理器的段描述符为0x34 d3 00 23 12 89 01 03，试指出该段描述符对应的段的起始地址与结束地址。
  - 段的起始地址：字节7，字节4，字节3，字节2，为：0x34231289.
  - 该段的界限为：字节6的低4位，字节1，字节0构成，即为：0x30103.
  - 该描述符的粒度G为1，因此实际的段界限为0x30103*4k,即为0x30103000.
  - 段的结束地址=起始地址+段界限-1.
  - 因此段的结束地址为：0x34231289+0x30103000-1=0x64334288.

保护模式下物理地址形成过程

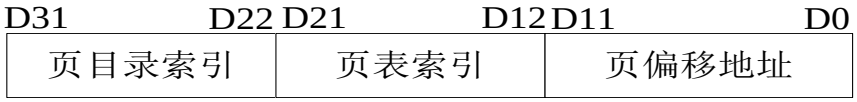


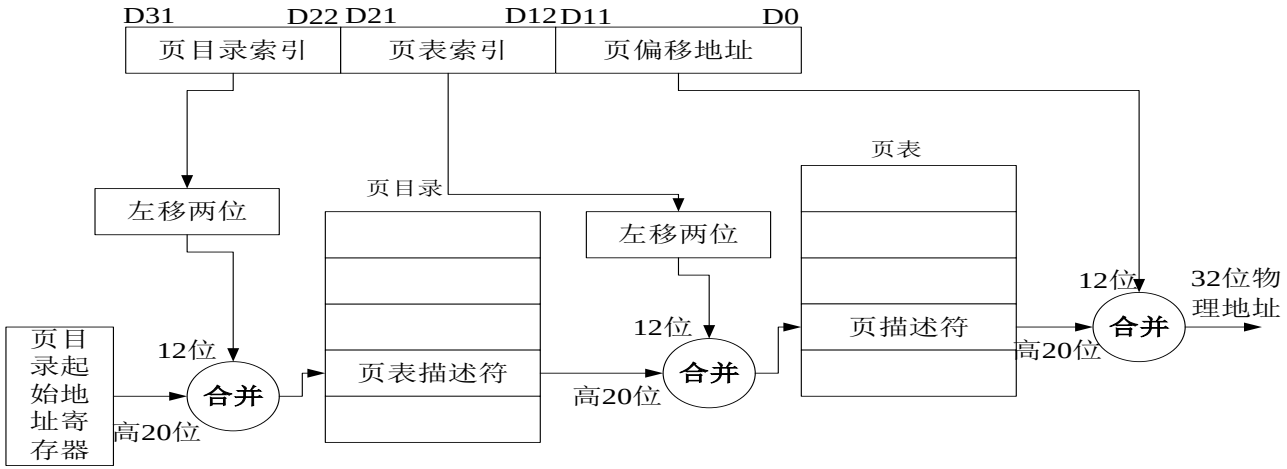
分页管理

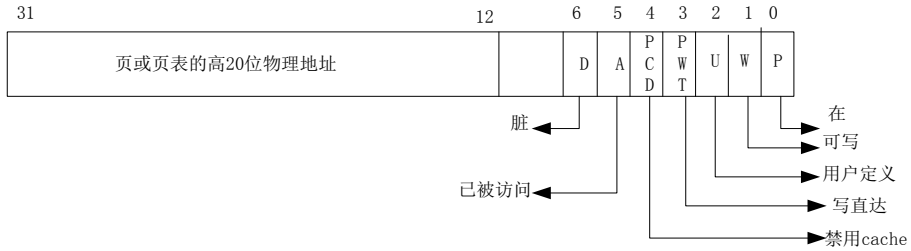
- 将地址空间分成许多相同大小的页
- 页的大小不同，决定了地址中用于寻址页内存储单元的位数



Intel 分页



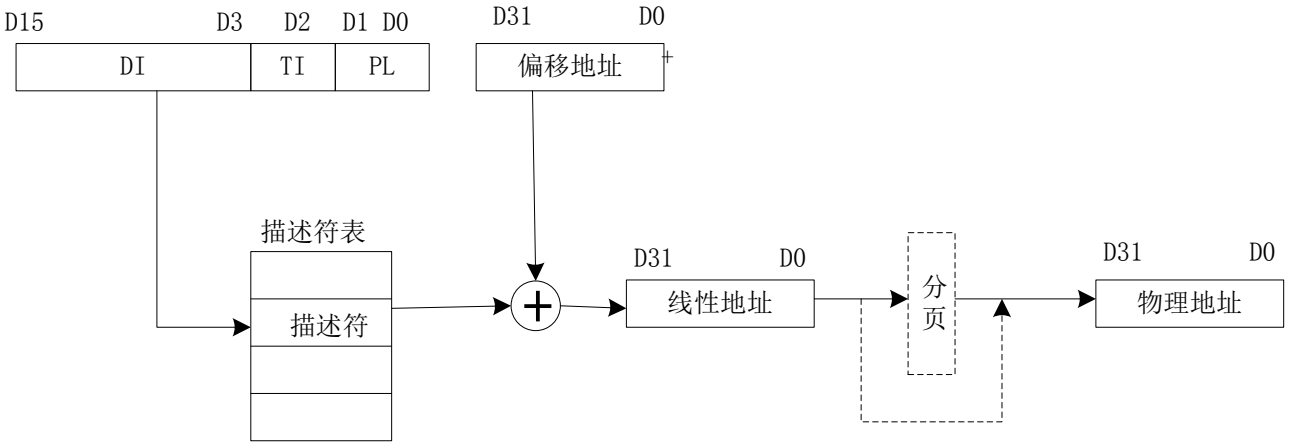




描述符结构

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。



4.4高速缓存原理（cache）

- 程序访问内存局部性原理
  - 程序访问的时间局部性
    - 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
    - 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。

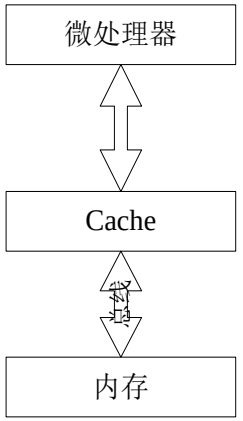
```
assign-array-rows ( )
{
  int i, j;
    short a[M][N];
  for  (i= 0; i<M; i++)
                  for (j= 0; j<N; j++)
                                a[i][j]=0;
                   }
```

```
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？

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。

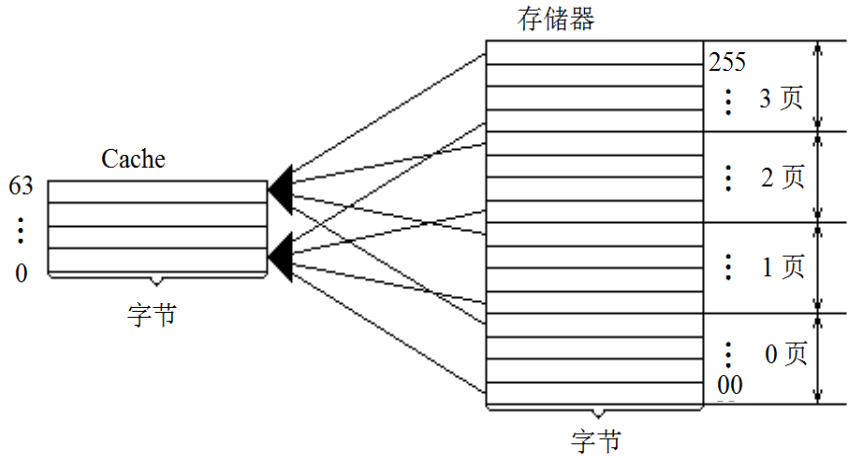


Cache构成原理

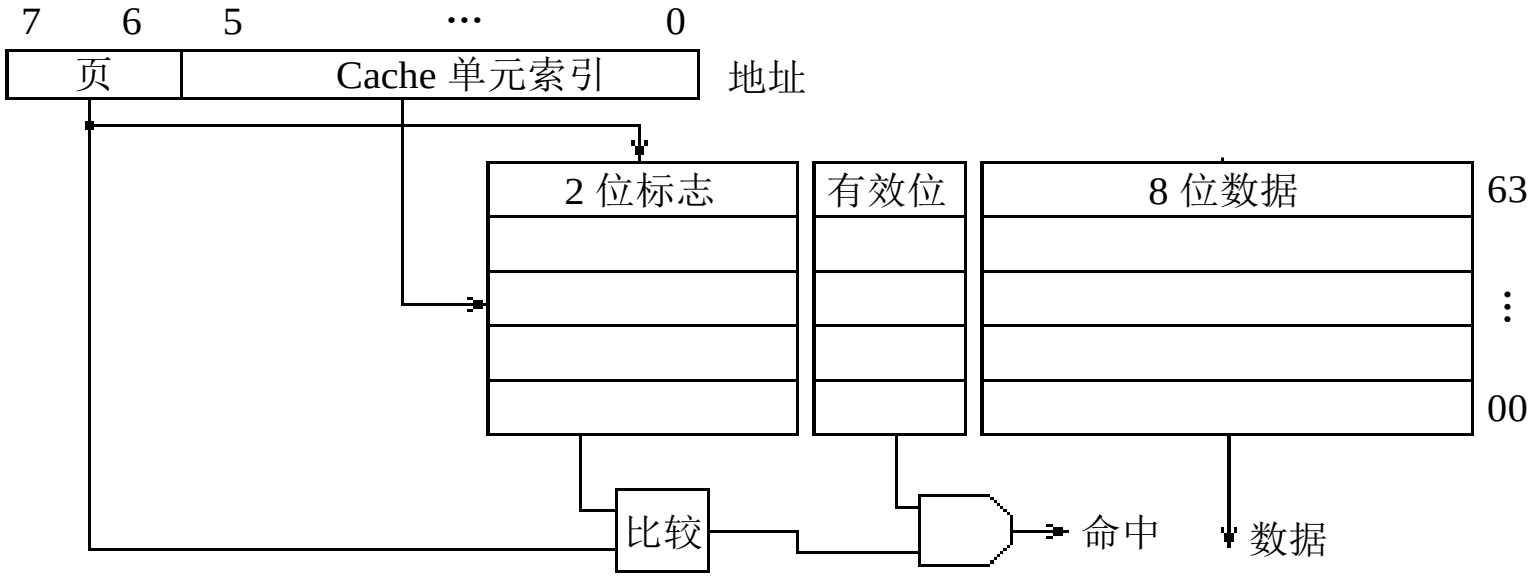
- cache存储单元与内存存储单元之间建立某种地址映像关系
  - 直接映像（Direct Mapped）、
  - 全相联（Full Associative）
  - 组相联(Set Associative)

直接映像

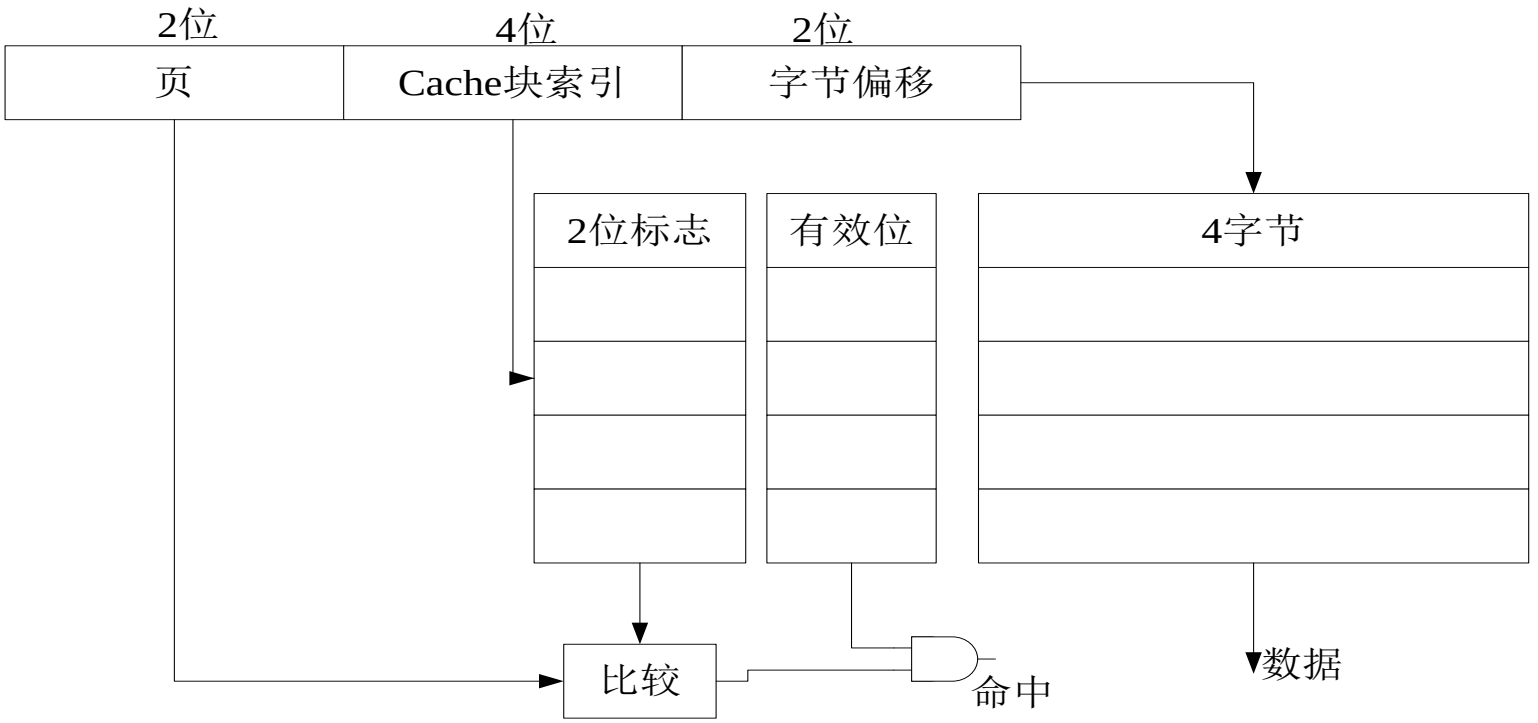
- 主存与缓存分成相同大小的数据块。
- 主存容量是缓存容量的整数倍，将主存空间按缓存的容量分成页，主存中每一页的块数与缓存的总块数相等。
- 主存中某页的一块存入缓存时只能存入缓存中块号相同的位置。



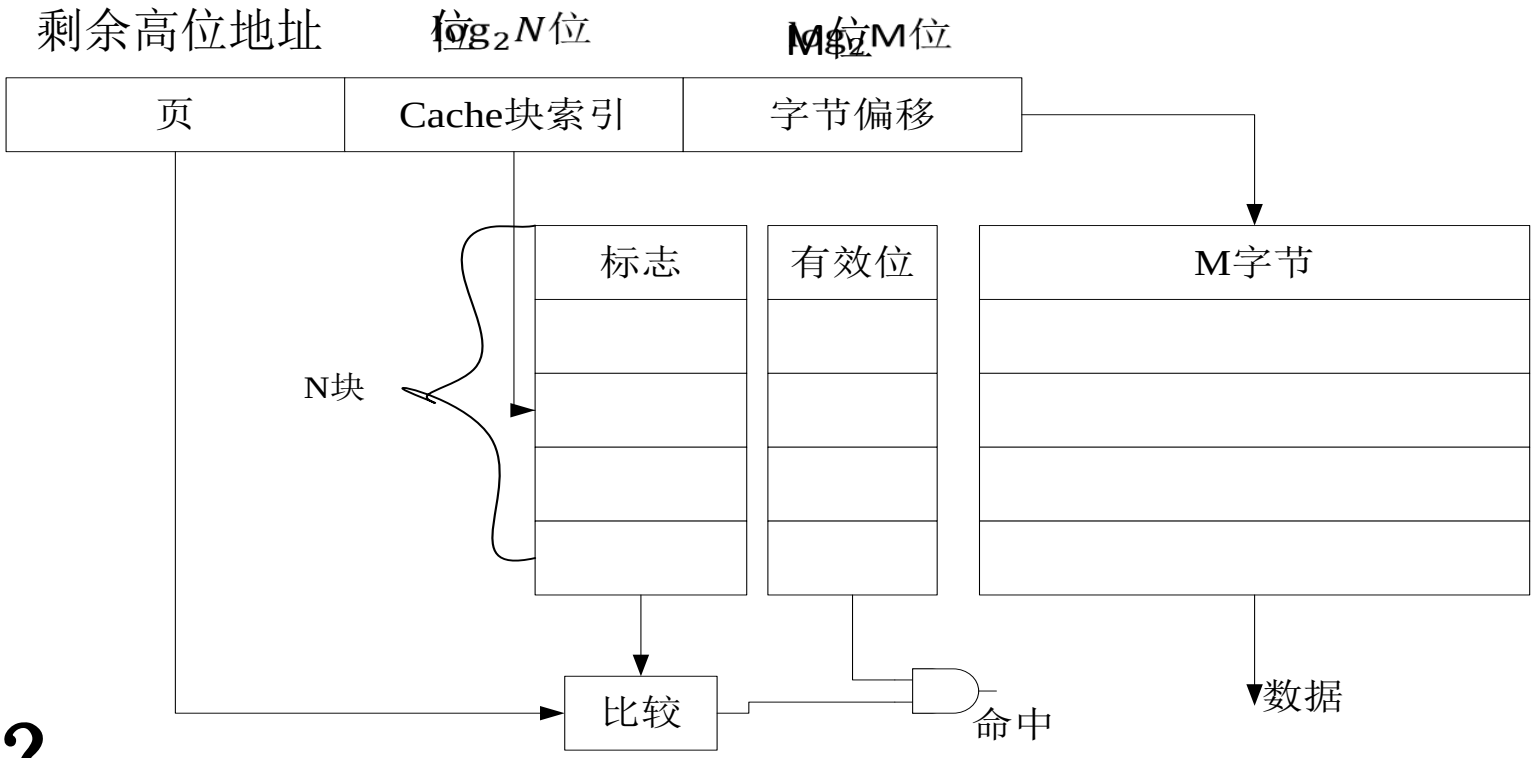
直接映射Cache结构原理框图



4字节块组织的直接映射Cache结构原理框



地址对应关系

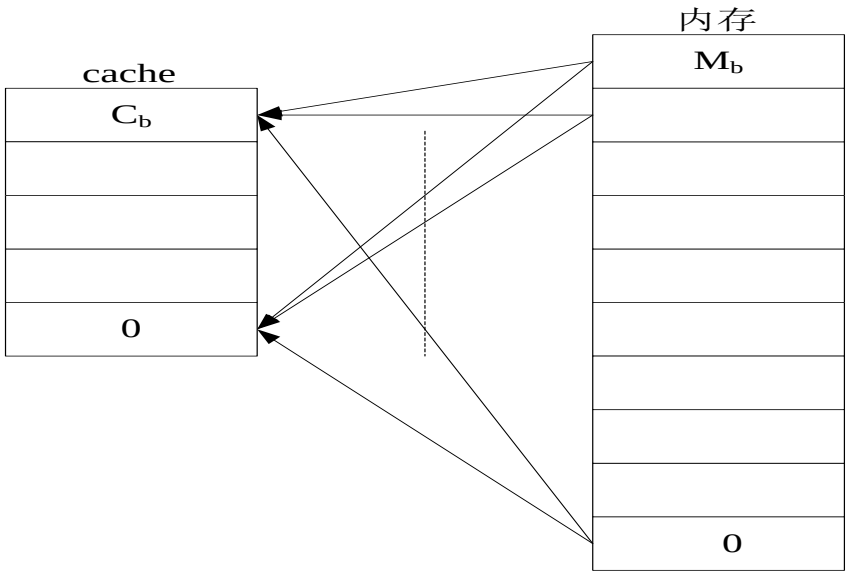


?

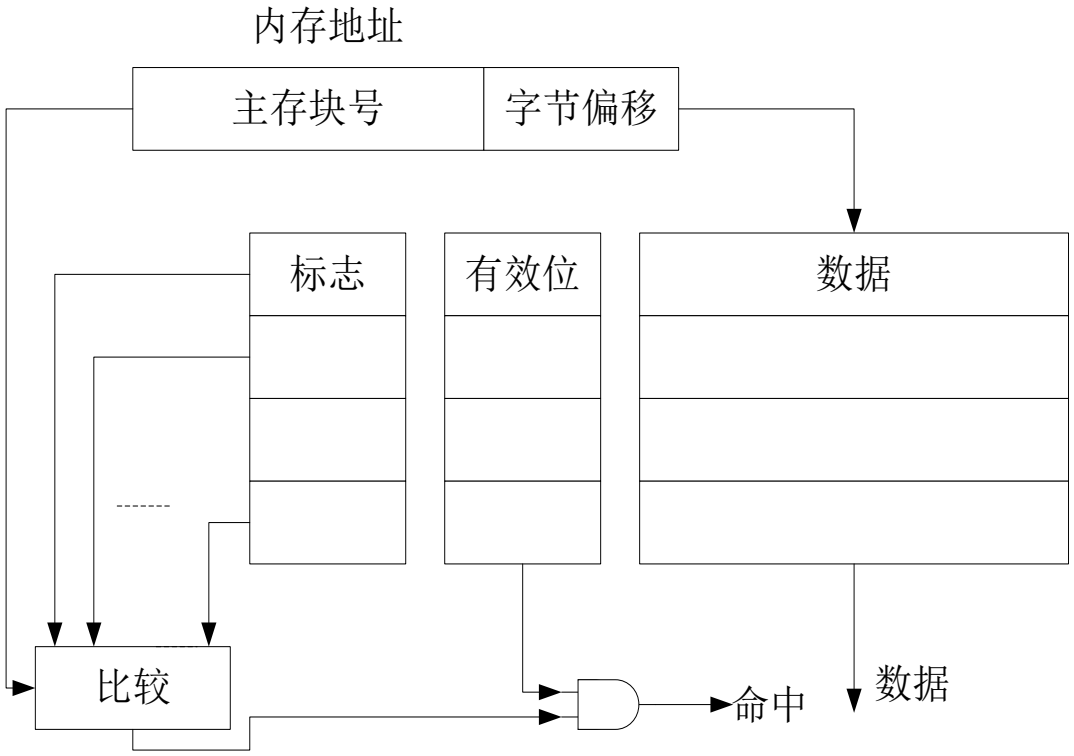
多个不同的页中处于同样行位置的数据访问比较频繁时，需要不停的更换同一个cache行的内容，cache替换操作频繁，命中率比较低。

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。



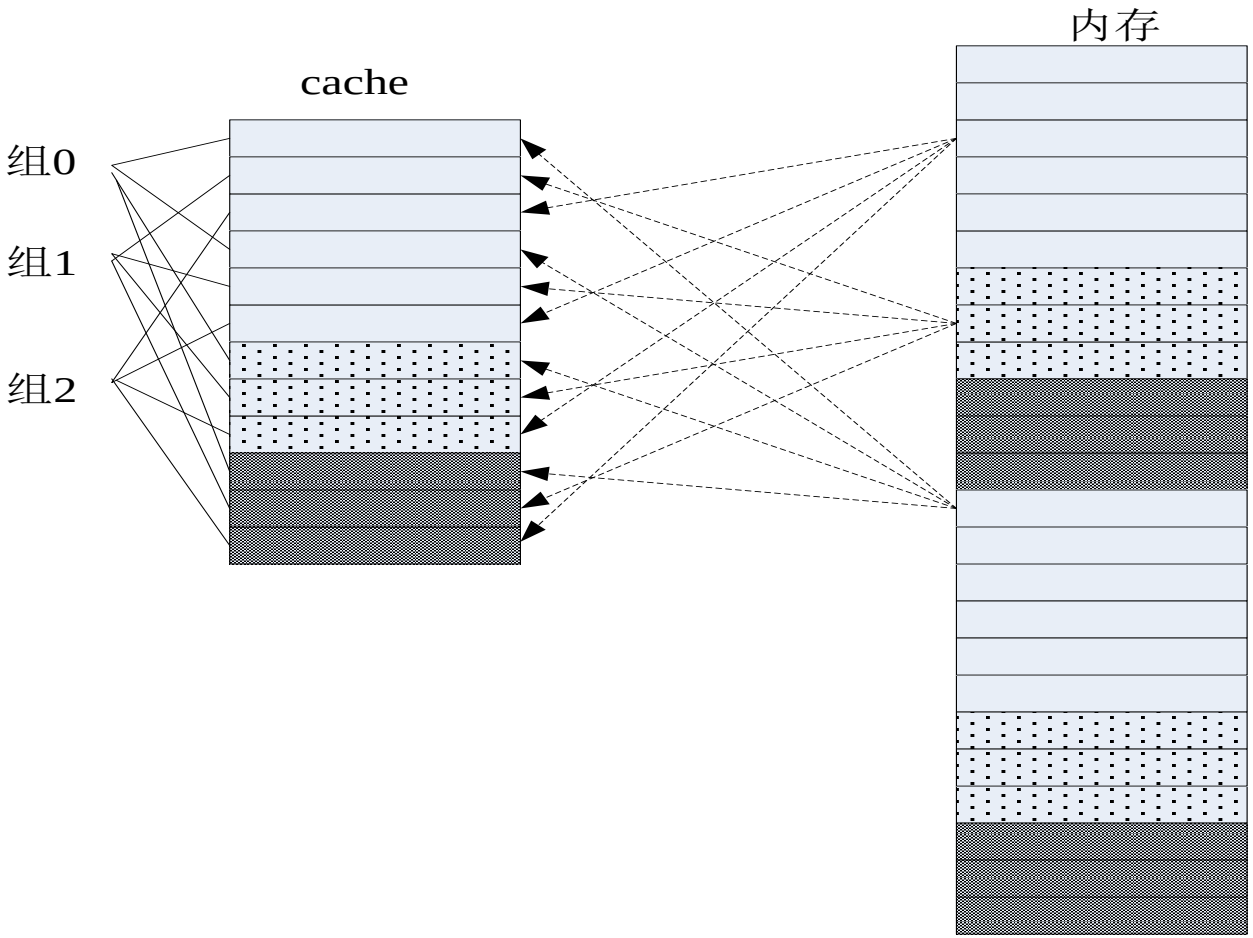
全相联映像cache结构原理



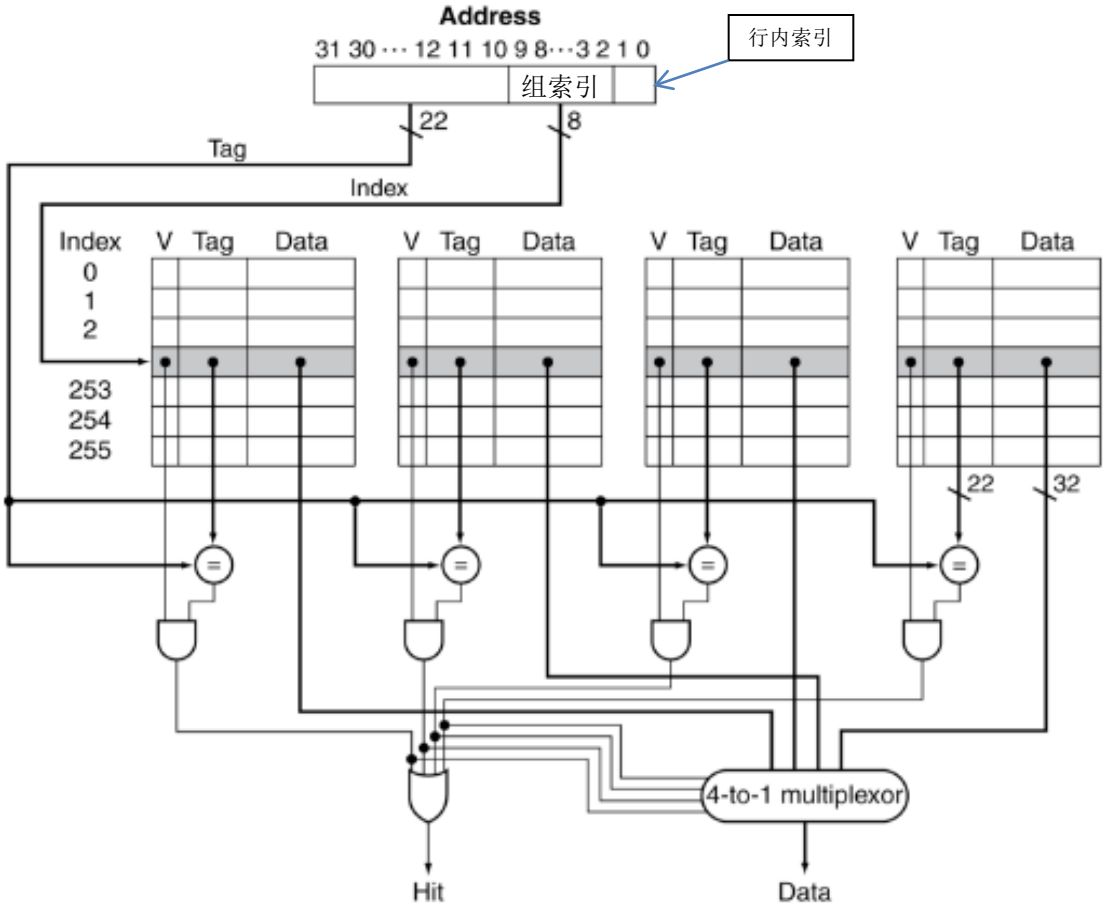
组相联映像

- 主存和Cache按同样大小划分成块。
- 主存和Cache按同样大小划分成组。
- 主存容量是缓存容量的整数倍，将主存空间按cache的大小分成区，主存中每一区的组数与缓存的组数相同。
- 当主存的数据调入缓存时，主存与缓存的组号应相等，也就是各区中的某一块只能存入缓存的同组号的空间内，但组内各块地址之间则可以任意存放，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。

3组，每组4行cache组相联



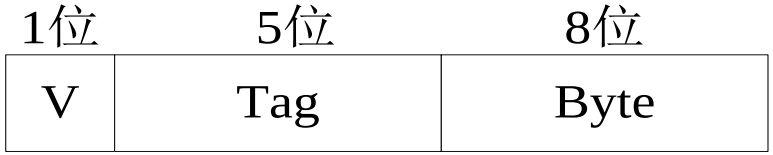
4路组相联映像cache结构原理



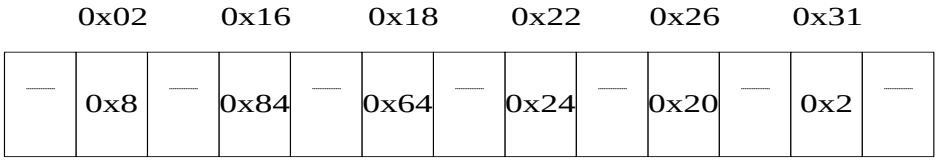
Cache读策略

- 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		0	
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		0	
001		0	
010	00100	1	0010 0100
011		0	
100		0	
101		0	
110	00000	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	00100	1	0010 0000
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

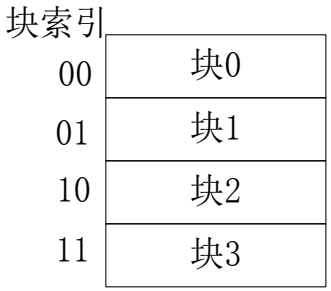
- 随机替换
- 先入先出（FIFO）替换
- 最近最少使用（LRU）替换（最常采用）

- 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



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]			

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								

行优先访问填充过程

命中率50%

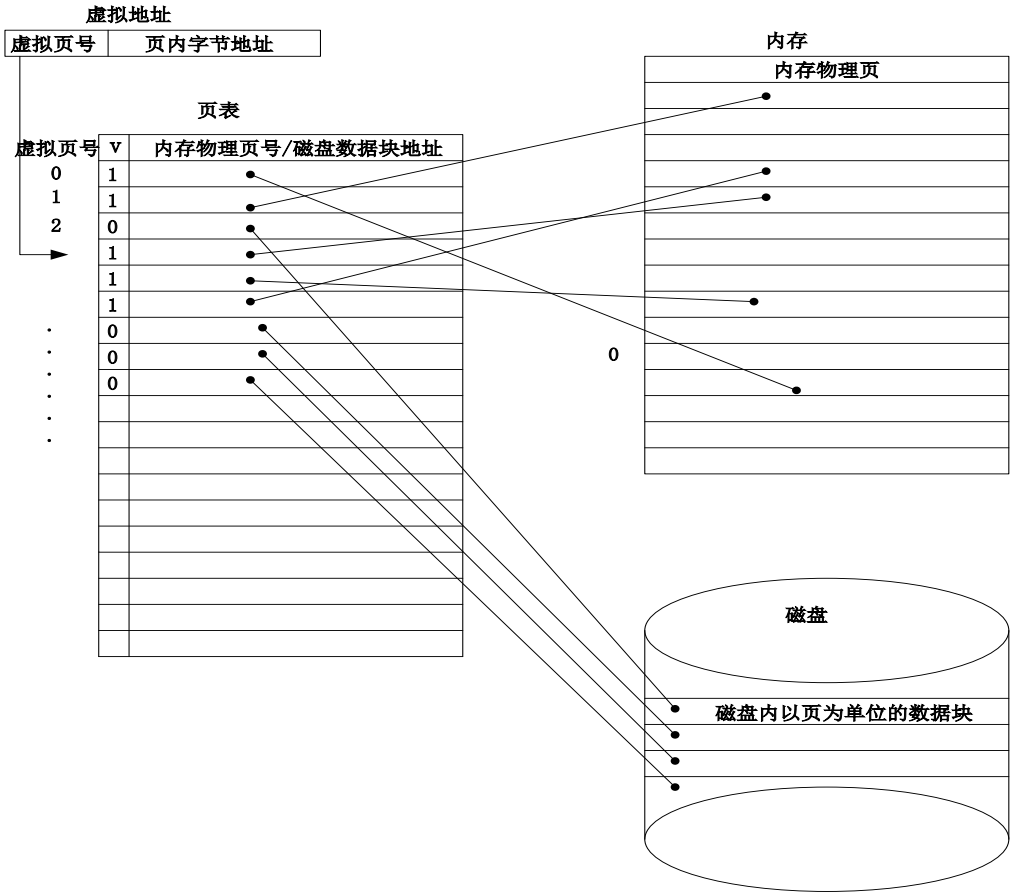
数组元素	命中否	cache内容							
		块0	块1	块2	块3	块4	块5	块6	块7
a[0][0]	否	a[0][0] a[0][1]							
a[1][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]						
a[2][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]					
a[3][0]	否	a[0][0] a[0][1]	a[1][0] a[1][1]	a[2][0] a[2][1]	a[3][0] a[3][1]				
a[4][0]	否	a[0][0] a[0][1]	a[1][0] 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]

列优先访问填充过程

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。

作业

- 2，5，6，　11