计算机组成原理

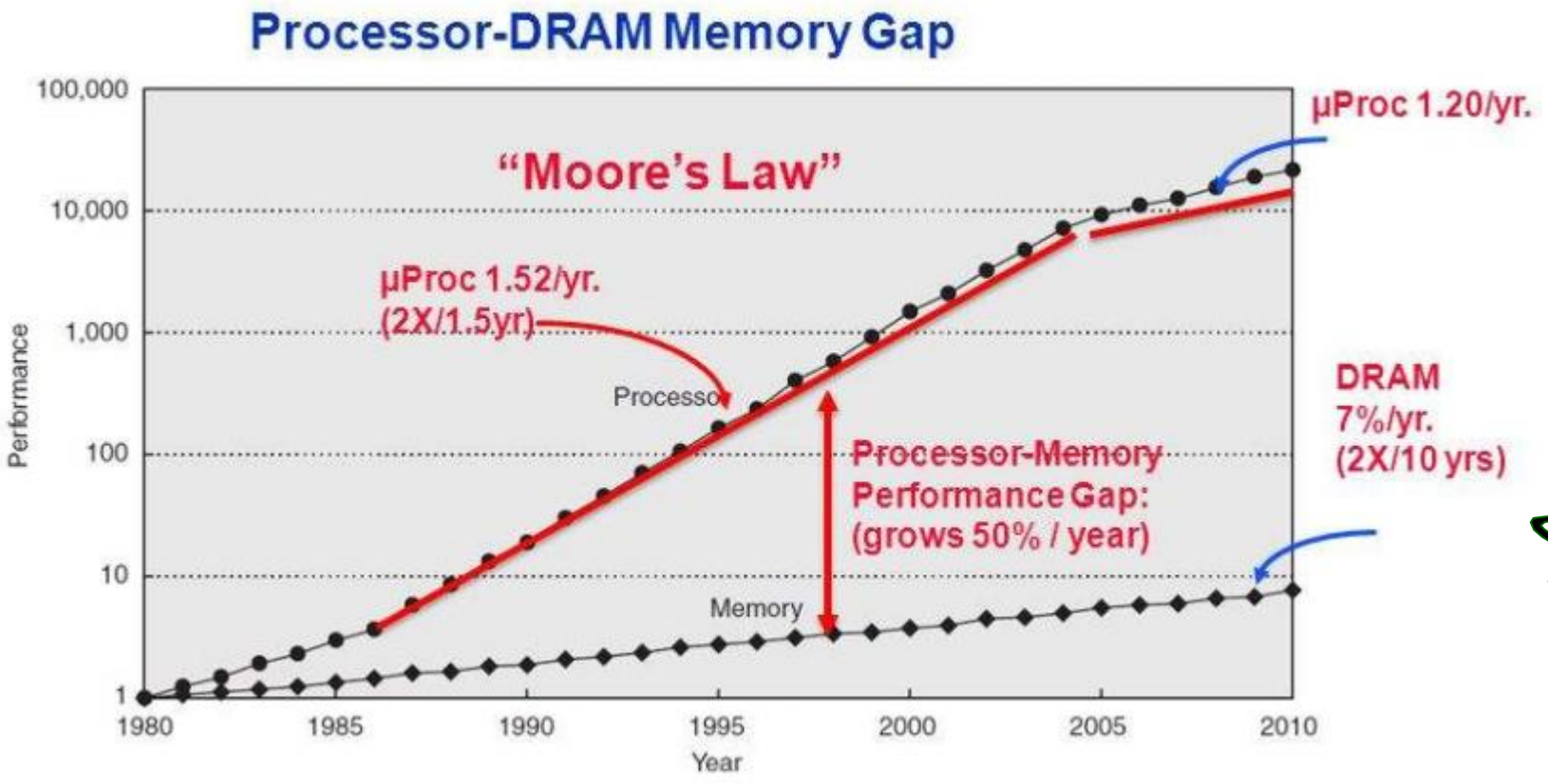
第5章 层次化存储

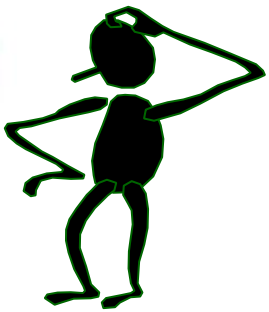
概述，Cache

llxx@ustc.edu.cn

Memory Wall：1995，Wulf@Univ of Virginia，$1.7

- 主存速度跟不上CPU性能(25MHz的80386之后)
  - 100MHz的Pentium处理器平均10ns执行一条指令，而DRAM典型访问时间60～120ns。
  - 指令流水线：单周期访存
- “处理器性能提升对系统的贡献被DRAM性能所掩盖”





PC机中的存储子系统：层次化

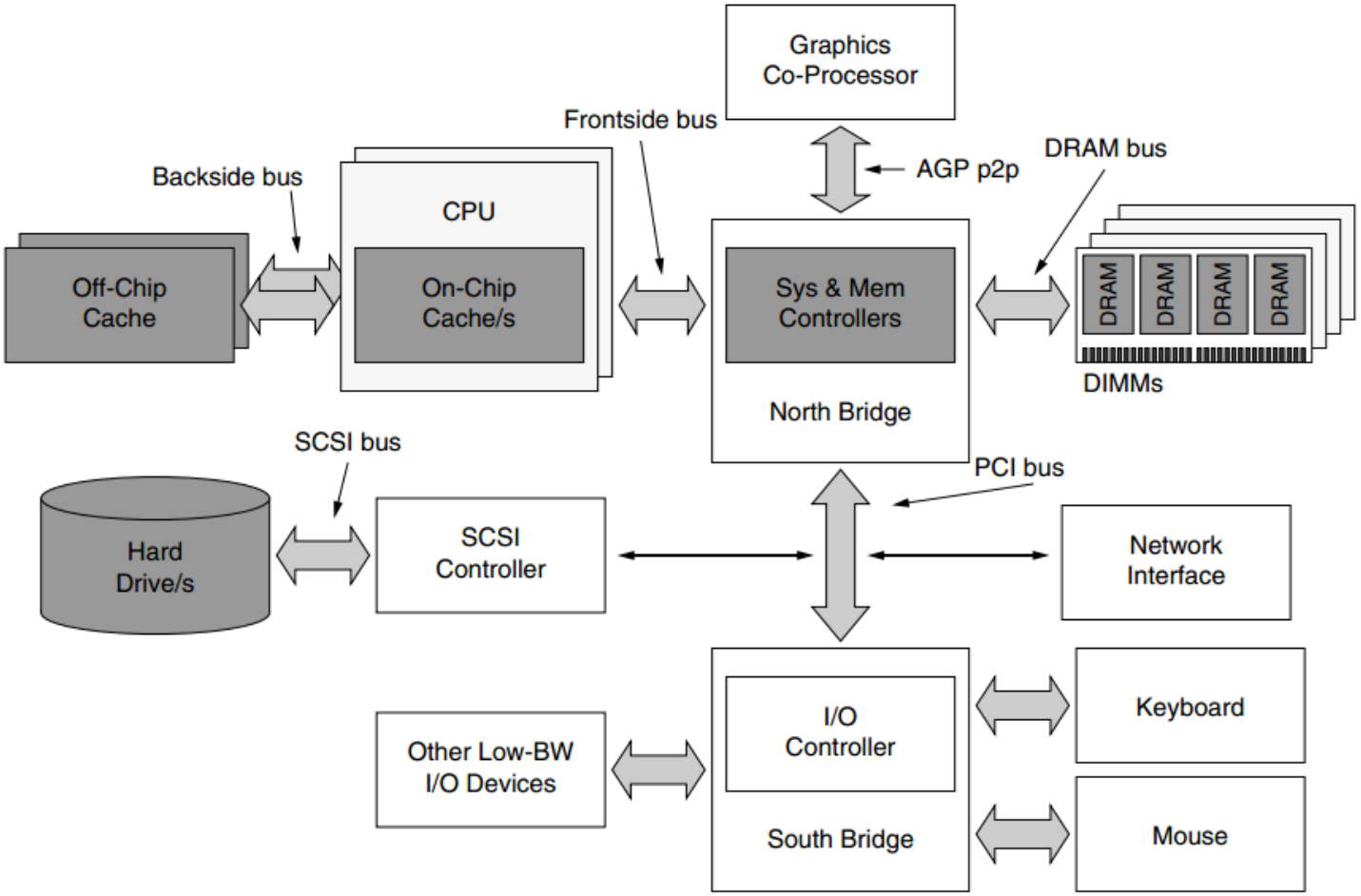
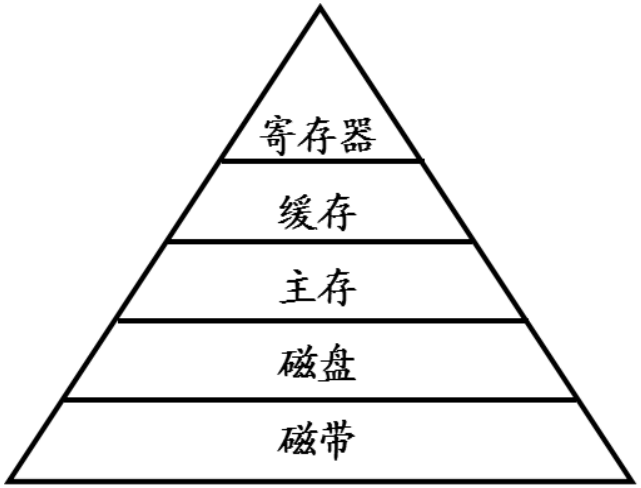
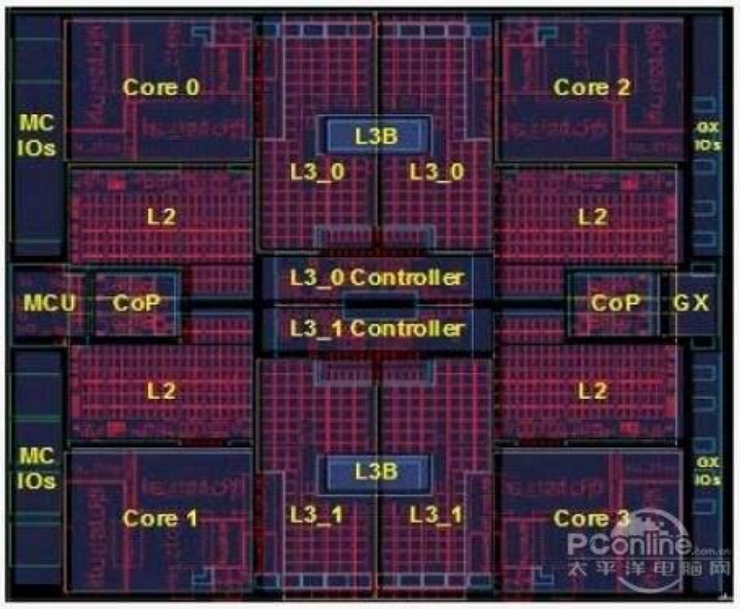


FIGURE Ov.3: Typical PC organization. The memory subsystem is one part of a relatively complex whole. This figure illustrates a two-way multiprocessor, with each processor having its own dedicated off-chip cache. The parts most relevant to this text are shaded in grey: the CPU and its cache system, the system and memory controllers, the DIMMs and their component DRAMs, and the hard drive/s.

Bruce Jacob，Memory Systems: Cache, DRAM, Disk，2008

层次化存储：性能、容量、价格





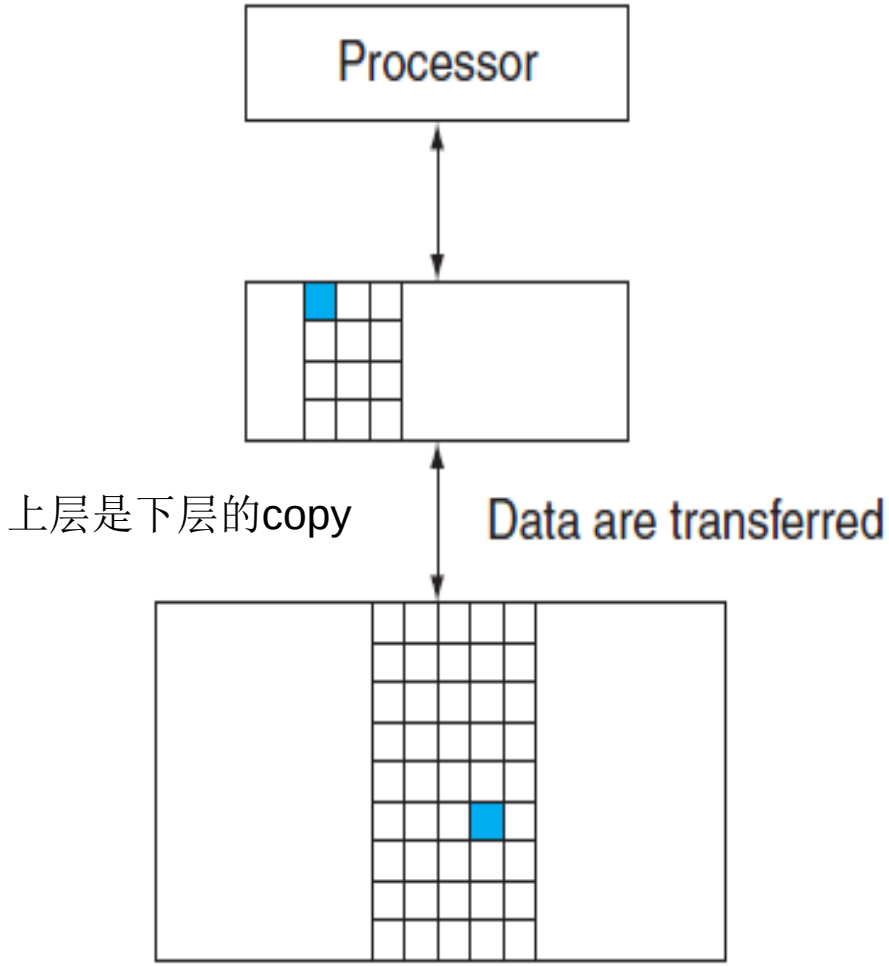


图5-2

Cache对处理器性能的影响：CPI=1

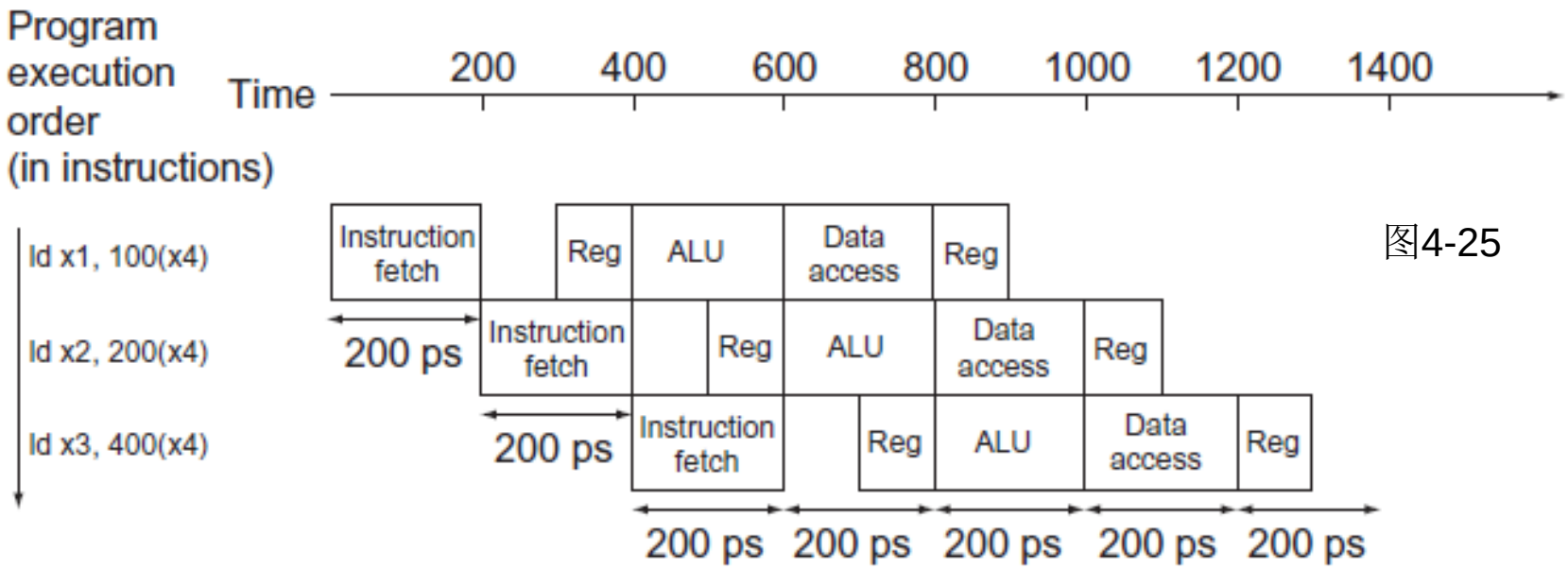
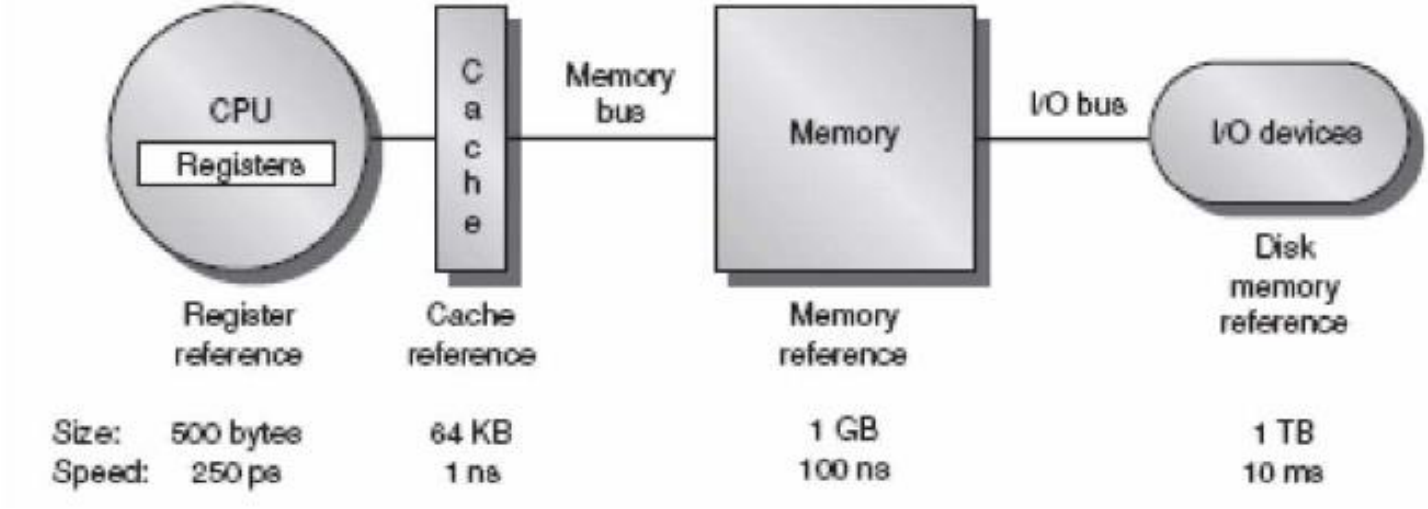
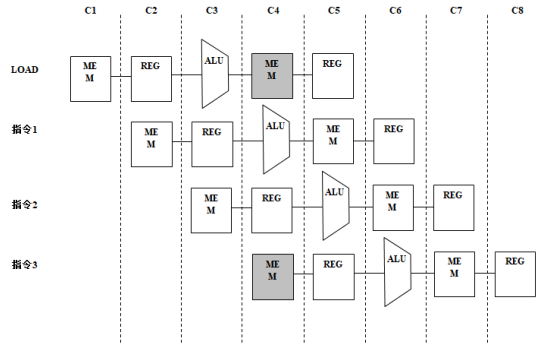


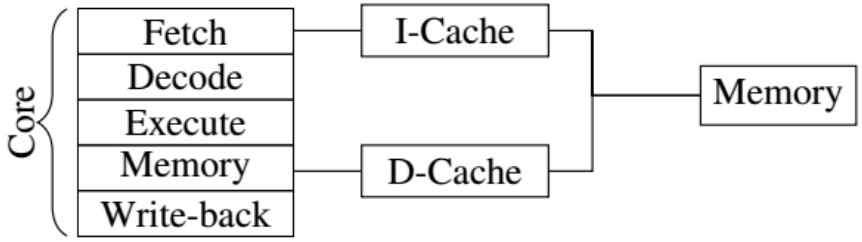
图4-25

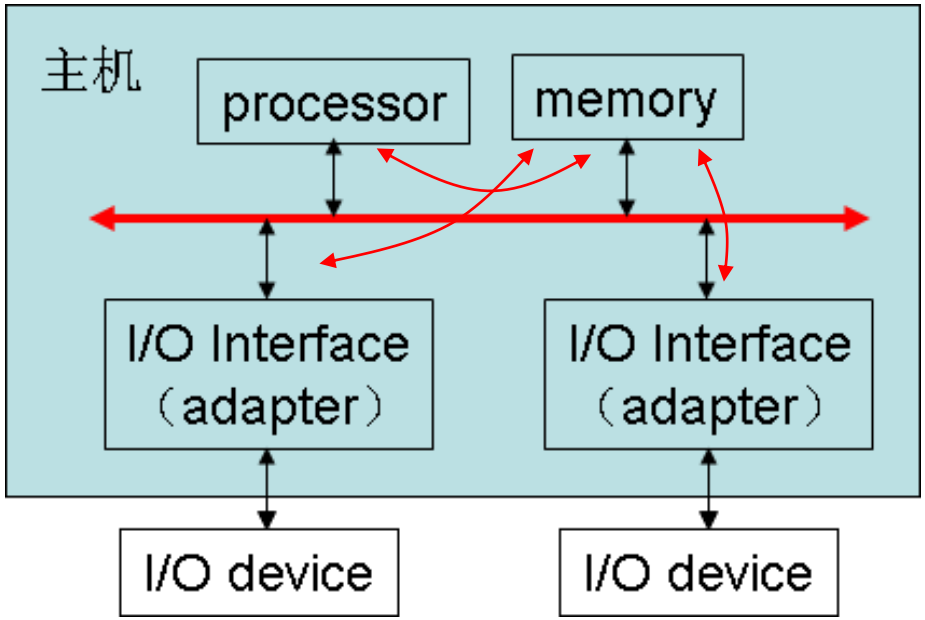


Cache对系统结构的影响

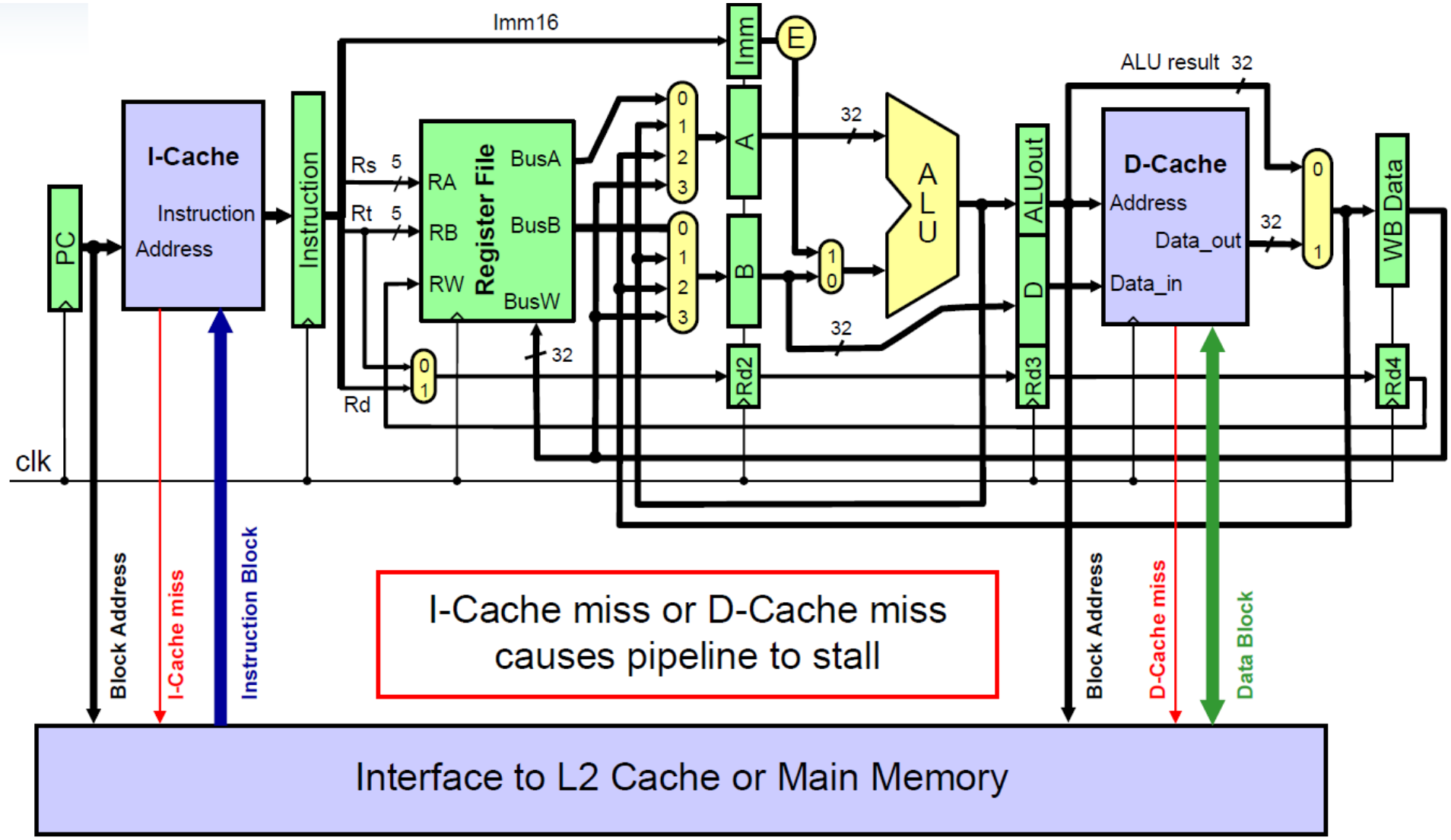
- 存储器冲突：取指与数据读写
  - 分体Cache
- 总线占用：CPU和I/O竞争访问主存
  - 减少CPU访问主存
- 副作用：一致性，时序可预测性





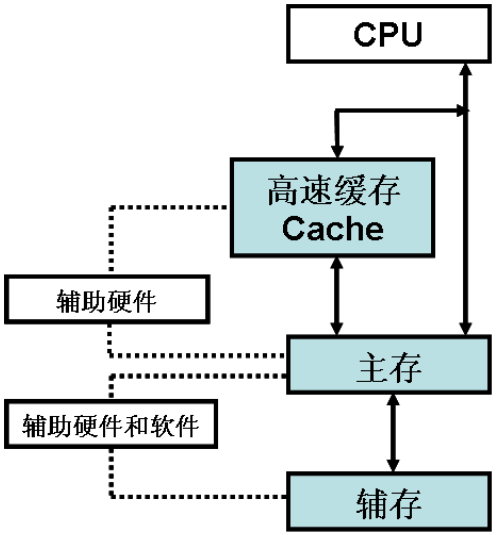


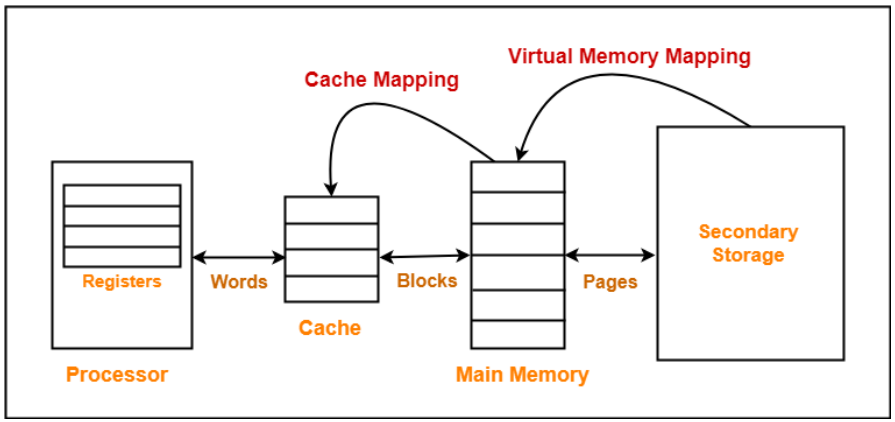
Cache miss：阻塞式（blocking，stall多个周期？）



“Cache - 主存”与“主存 - 辅存”层次的区别

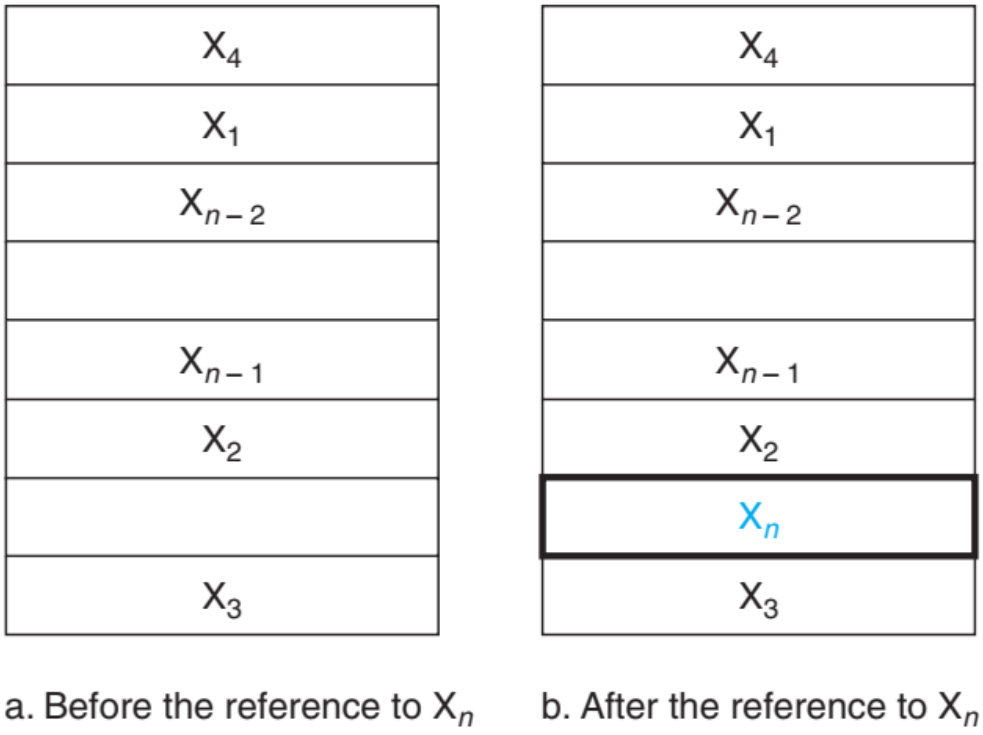
比较项目 \ 存储层次	“Cache – 主存”层次	“主存 – 辅存”层次
目　　的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理实现	主要由专用硬件实现	由硬件和软件实现
访问速度的比值(第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程



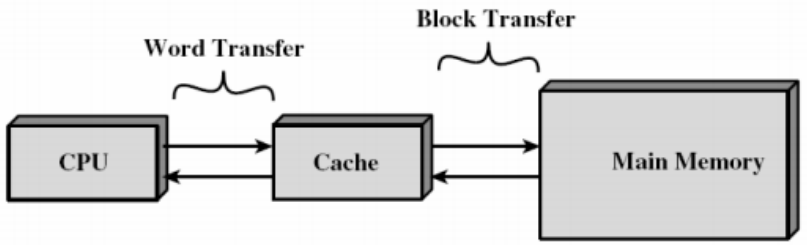


例：访问X_n前后Cache变化：miss -> hit

图5.7



- 设初始时，Xn不在Cache中
- 两个问题
  - 数据放在哪儿?
  - 如何判断是否命中?



本讲内容：Cache系统（单处理器）

- 为什么需要Cache?
  - 性能、结构
- Cache有效性的理论基础
  - 局部性原理：时间，空间
- 影响Cache命中率的因素
- Cache的基本结构，5.3
- Cache的读写操作过程，5.3，5.8
  - Cache一致性
  - 阻塞式Cache
- Cache-MEM映射机制，5.4，5.8
  - 块放哪儿?
- Cache的替换策略，5.4，5.8
- Cache控制器：5.9，5.12
- *Cache 性能分析*
  - *主要见体系结构课*
- *Cache Coherence，5.10*
  - *见体系结构课*

- 三个关键问题：PWR
- the mapping function（映射）
  - the link between a block's address in memory and its location in the cache;
  - Block Placement Schemes
- the write policy
  - how the processor writes data to the cache so that main memory eventually gets updated;
- the replacement algorithm
  - the method used to figure out which block to remove from the cache in order to free up a line.

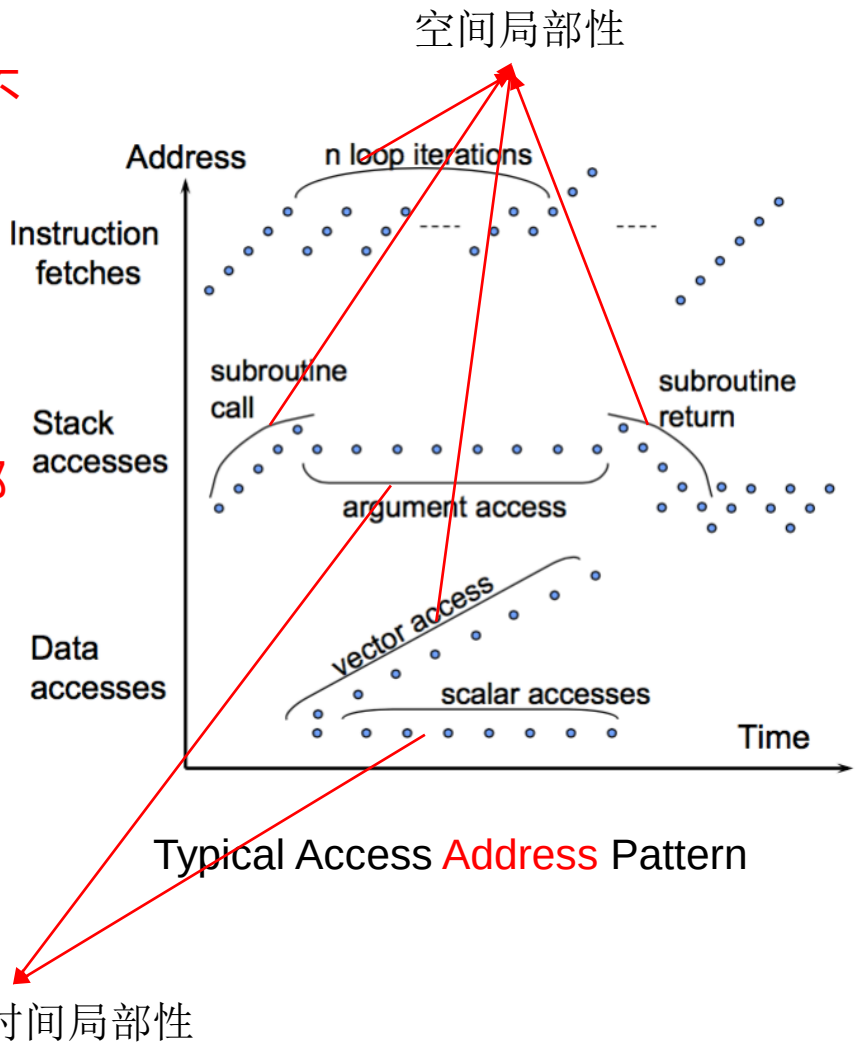
- **COD5**
  - **5.3, 5.4, 5.8, 5.9, 5.12**
- **唐：4.3，附录4A**

程序的访存特性

- 时间局部性temporal locality
  - 最近的访问项（指令/数据）很可能在不久的将来再次被访问
    - 频繁访问某个地址
  - 策略：保留data，复用
    - 内存地址不一定集中!
- 空间局部性spatial locality
  - 一个进程访问的访问项其地址彼此相邻
    - 往往会访问在存储器空间的同一区域
  - 策略：保留data及其相邻者，预取
    - 内存地址连续!
- 例：时间局部性？空间局部性?
  - for i := 0 to 10000 do

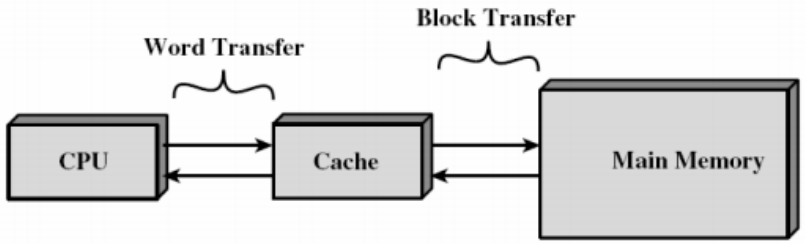
    A[i] := 0;

- 时空局部性实现：内存分块

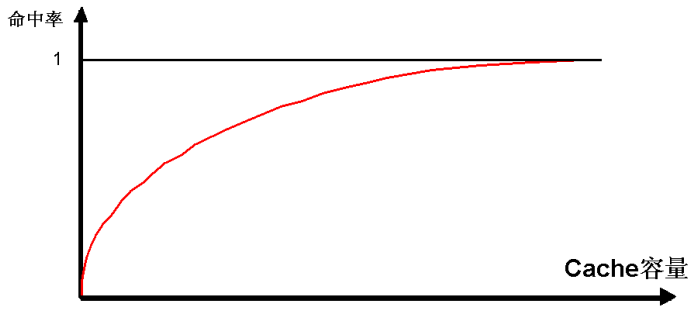


命中、不命中、命中率

- Cache命中（hit）
  - 欲访问的数据在缓存中
  - 命中时间



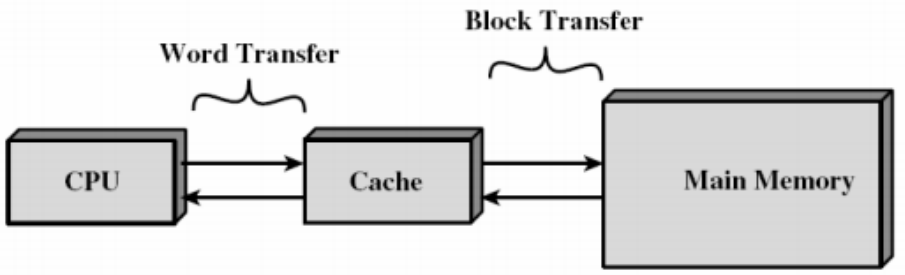
- Cache不命中（miss, 缺失/“失效”）
  - CPU欲访问的数据不在Cache内，或数据无效
    - 需将数据所在主存块（block）一次性调入
    - 缺失损失（penalty）：时间 = mem->cache->cpu
      - CPU可阻塞（blocking，stall）或非阻塞（non-blocking）
- 命中率（Hit rate）
  - CPU要访问的信息已在Cache内的比率。
    - 通常用命中率来衡量Cache的效率。
- 不命中率（Miss rate）



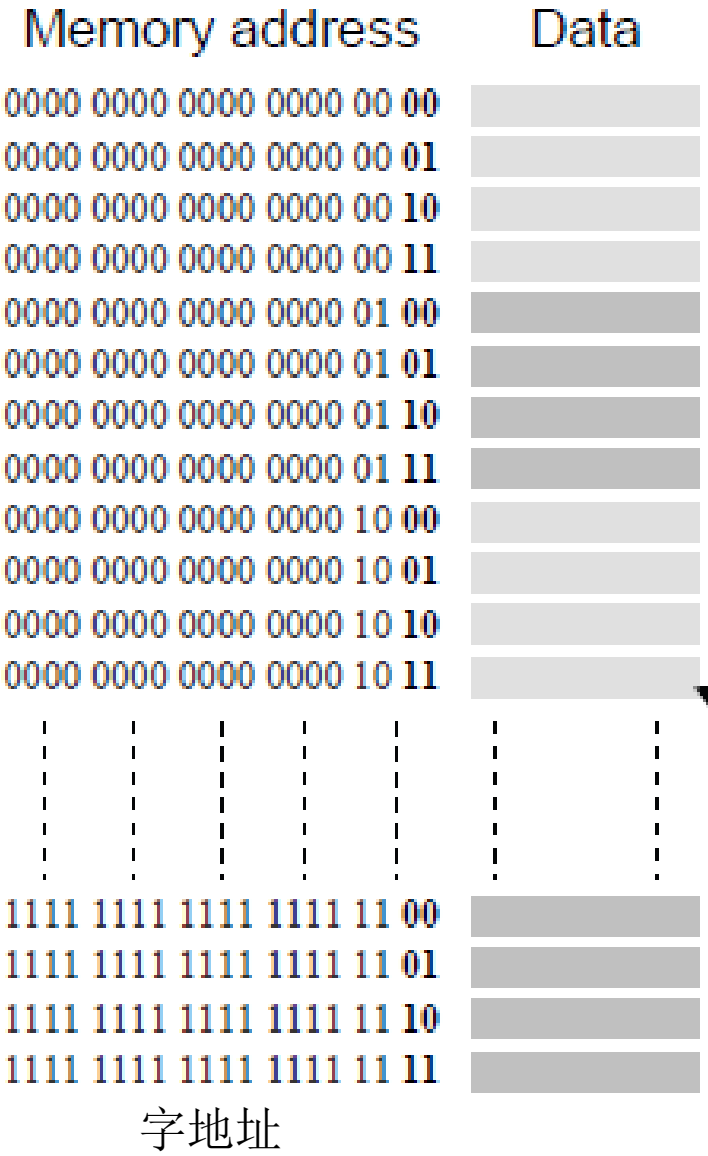
例：Cache基本结构参数

块（行）大小	1——32字
命中时间	1——2时钟周期（常规为1）
缺失（失配）时间	8——100时钟周期
（访问时间）	（6——60时钟周期）
（传送时间）	（2——40时钟周期）
失配率	0.5%——10%
Cache容量	1KB——1MB

•Cache-line size match the width of the DRAM simplified the design.



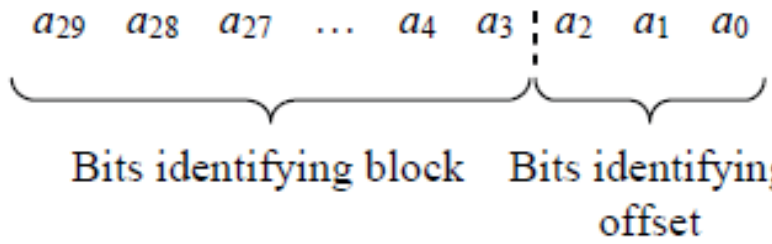
内存分块（block，字块），一块多字



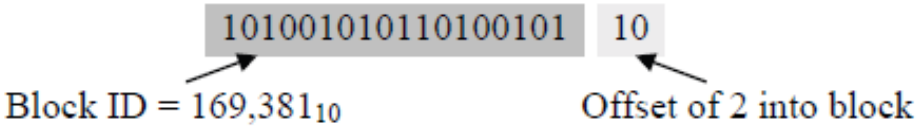
Block identification

Block 0

4个words



Block 1

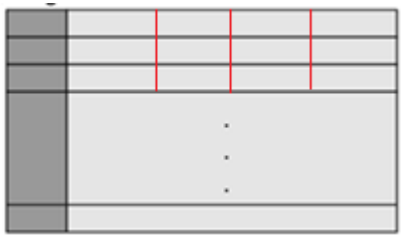


Block 2

Each gray block（one data）represents an addressable memory location containing a word

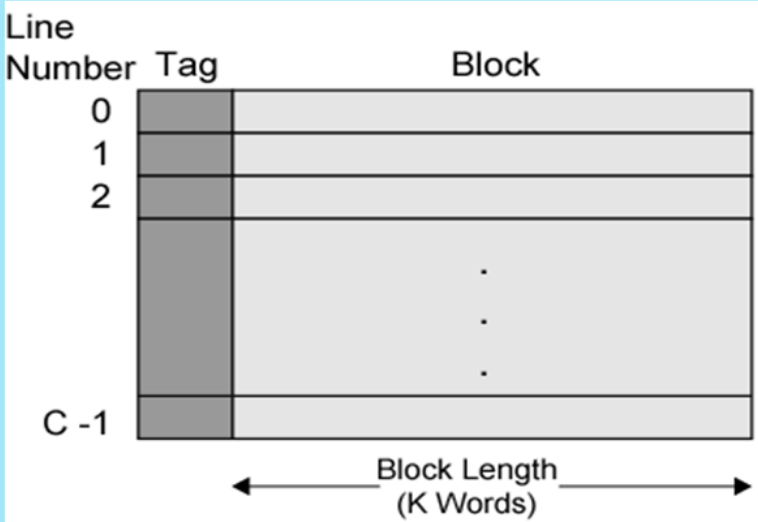
Block $2^{(20-2)} - 1 = 262,143$

一个Cache block（4个data）

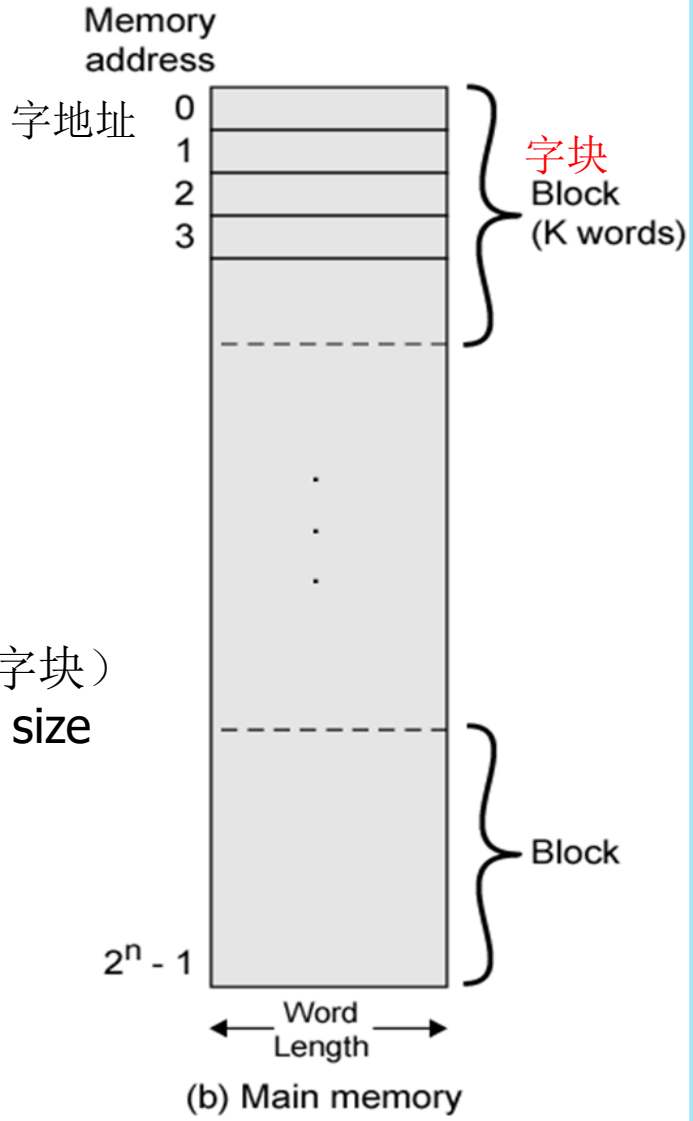


块大小多少合适？

Cache/Main Memory Structure：行/字块



(a) Cache



(b) Main memory

内存分块（字块），Cache分行，一行一块（字块）

Cache line（Cache Block） size = Mem Block size

Main memory = 2^n words = $2^n/K$ blocks

Cache has C lines of K words each

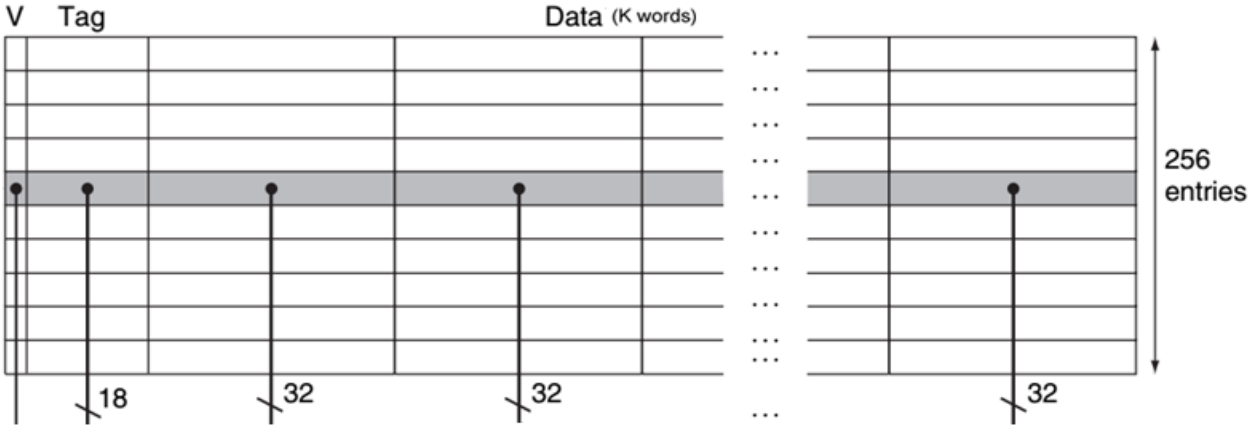
Tag – to identifies line——“按地址访问”

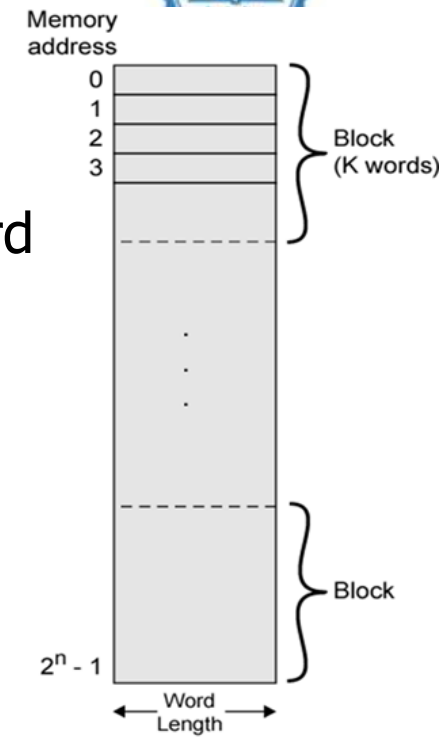
Line = **way**

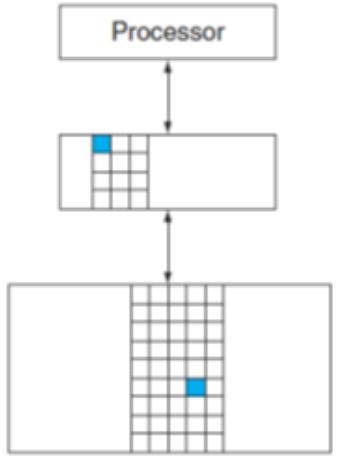
Cache Line Structure

Cache Line = tag + block(K words) + VCD

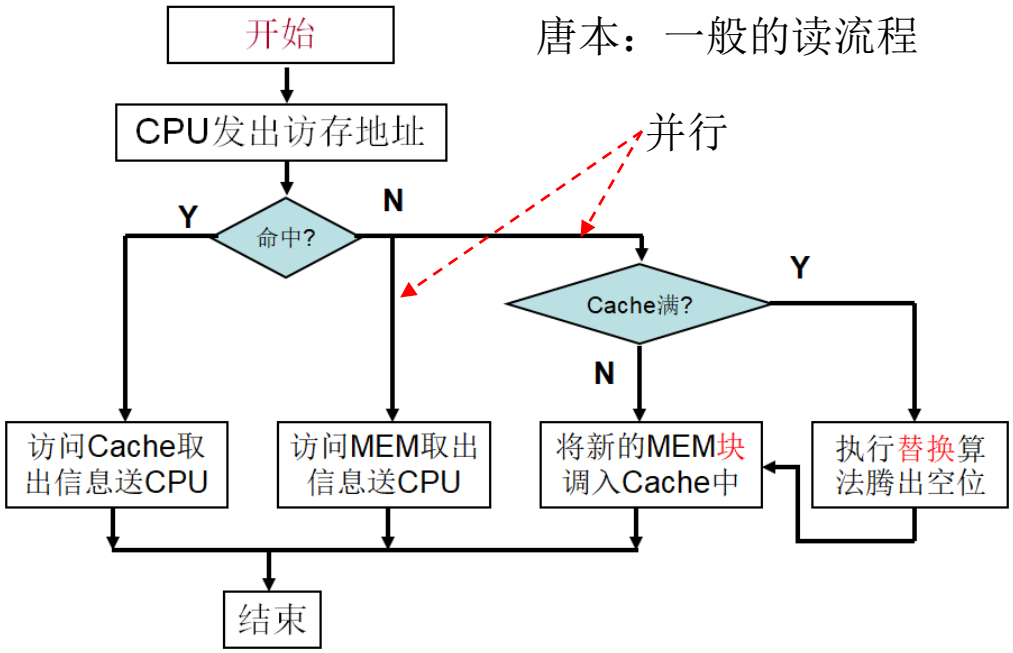
- 一行一块（字块），一块多个data，一个data一个word
- Tag：to identifies block，= 内存块号?
- control bits： VCD
  - 有效位（Valid）：数据是否有效？
    - 无效数据：cold start/process migration/first reference
    - 写操作的使无效法（Invalidated）
  - 重写位（overwrite，Dirty）：数据是否修改？
    - 行替换时需要写回
  - 计数位（Count）：访问频度?
    - 替换算法选择标识



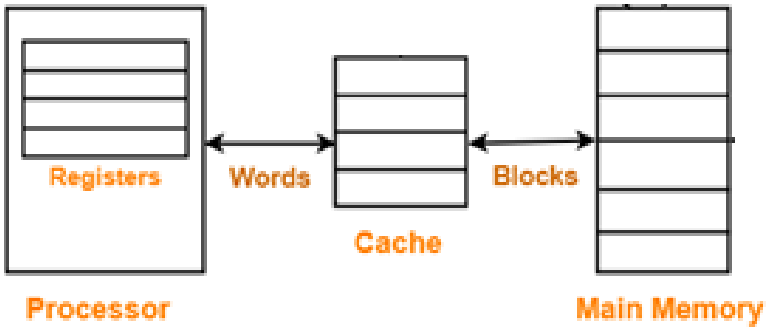


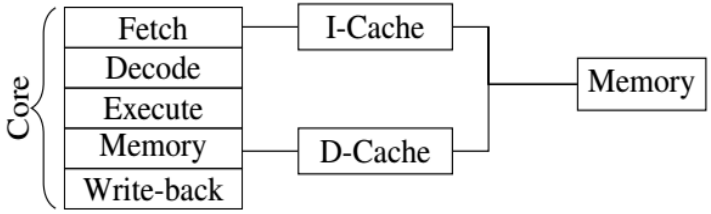


Cache读访问过程，$5.3.2



唐本：一般的读流程



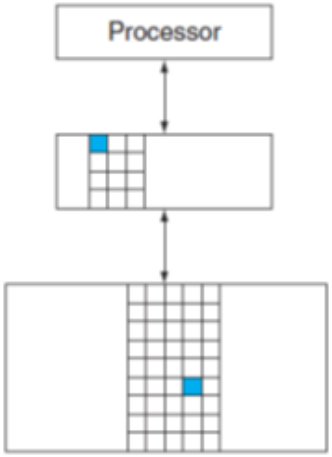


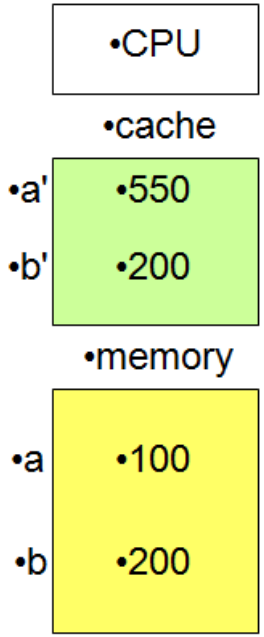
两个Cache同时miss?

- 并行：CPU主控制器+Cache控制器
- I$ miss
  - 阻塞式：CPU stall，等待读操作完成，重新取指
    - 换入换出?
- D$ miss：与I$类似，dirty时换出需要写MEM
  - 阻塞/非阻塞式（OOO）

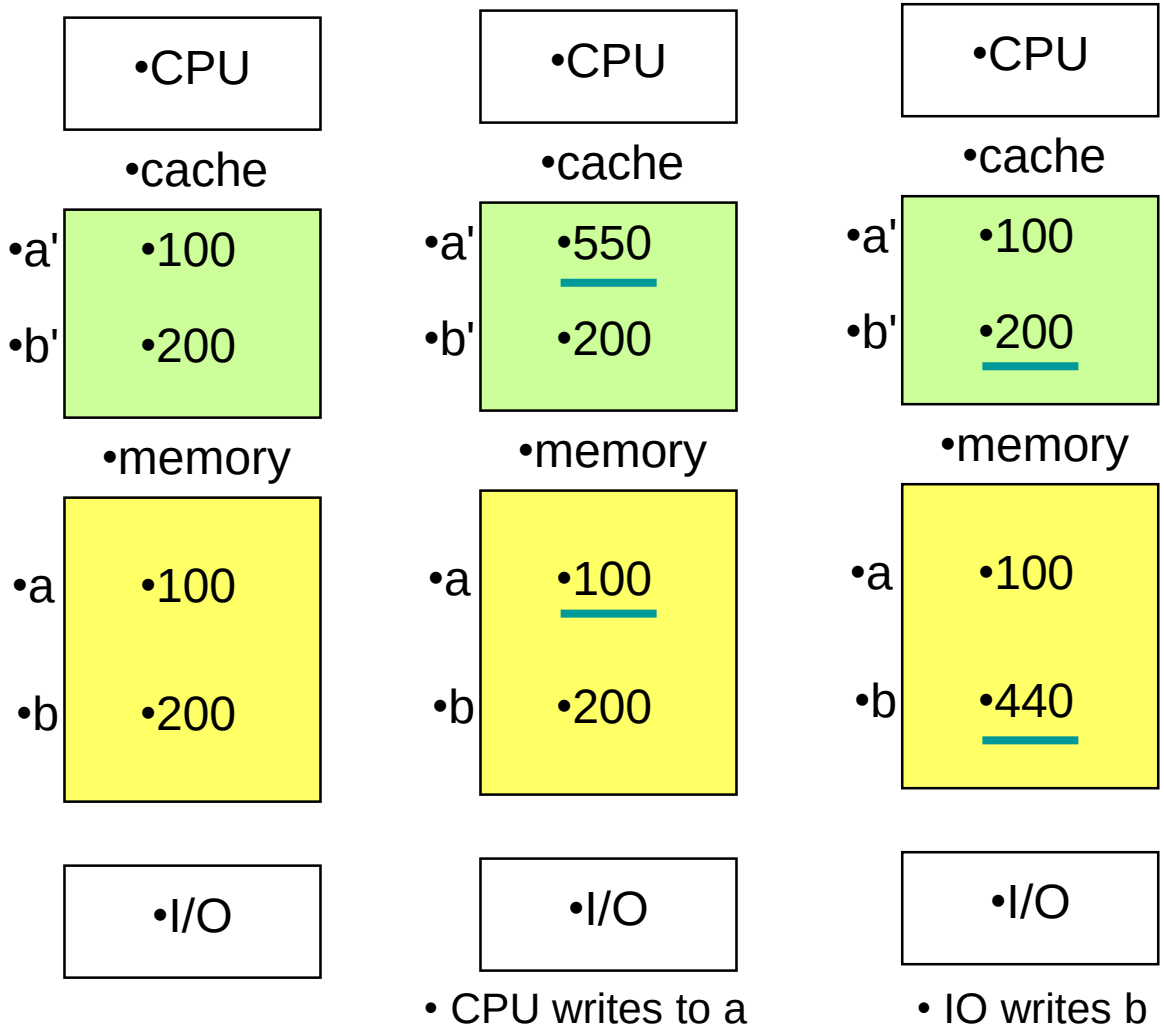
write policy，$5.3.3

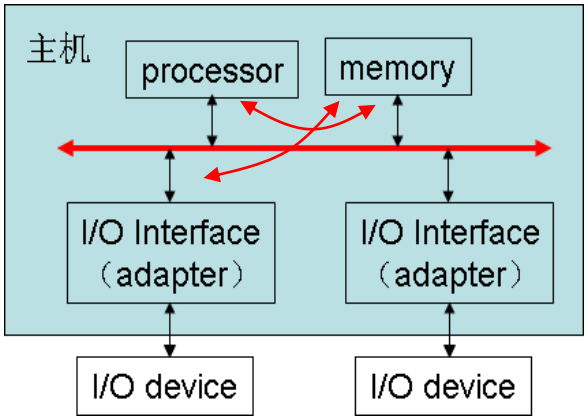
- 命中
  - 写透（Write-through/Store-through，写直达/“写穿透”）
    - 同时写入Cache和MEM
    - 写MEM≈200cc：写缓冲（write buf，store buf）。满，阻塞
  - 写回（Write-back，“写返回”）
    - 只写Cache，置Dirty（不一致），替换时再整行写回MEM（写回缓冲）
- 不命中
  - 写分配（write allocate）：从mem读入Cache后再写
    - 也称fetch on write，MIPS采用。
  - 写不分配(no write allocate，write around)：只写mem
    - 通常用于写操作较少或随机写
  - 写透两者都可以，写回只用写分配
- *写无效法（Invalidated）：多处理器cache写*
  - *只写主存，同时将各处理器的Cache相应块Valid位置0*
- 数据值与主存一致?
- 写操作性能?





Cache与主存的不一致问题：$5.3.3



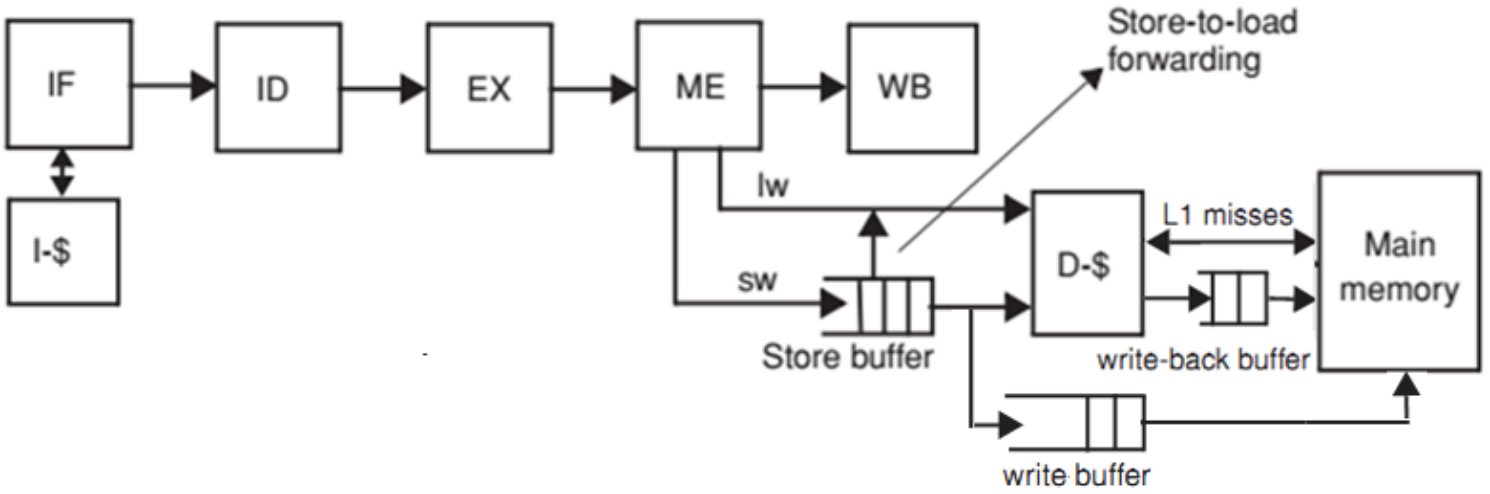


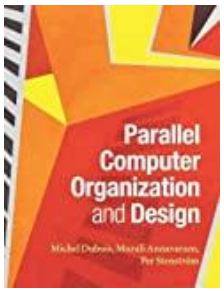
两种inconsistent场景：

1. CPU写操作
2. DMA写操作：HP *“I/O一致性”*
   - 方法1：置valid位；
   - 方法2：simple flush the cache before DMA write

加速“写”操作：$5.3.3【精解】，$5.8.4

- 结构：Store buf（= Write buf？支持store-load旁路），Write-back buf
- 写回：不能直接写Cache，可流水化
  - 命中：
    - CC1（= ppl的ME周期）：sw写store buf，指令完成。Cache控制器判断是否命中。
    - CC2（unused周期）：写入D$，由Cache控制器完成。
  - miss：如需替换，则写回脏块->读入所需内存块（写分配）->cpu写
    - write-back buf：存放待写入mem的脏块，尽快开始读入内存块
- 写透：可直接写Cache，一个CC完成
  - write buf（$5.8.4）：按字访问

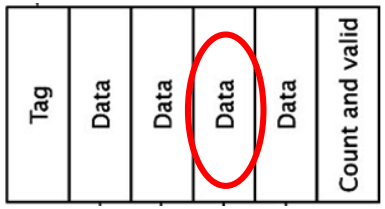


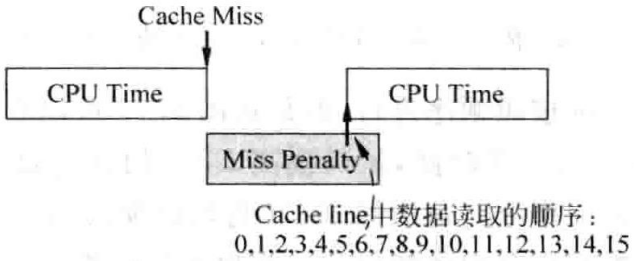


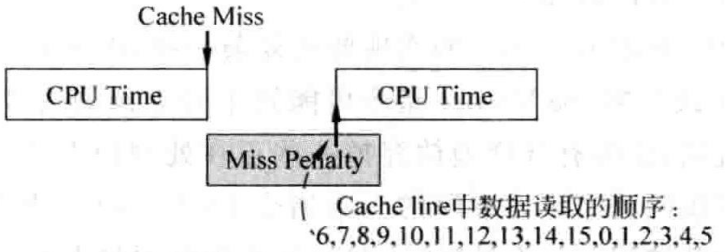
参考《Parallel Computer Organization and Design》Fig4.8、Fig7.3，llxx重绘

读写放大问题，$5.3.1【详解】

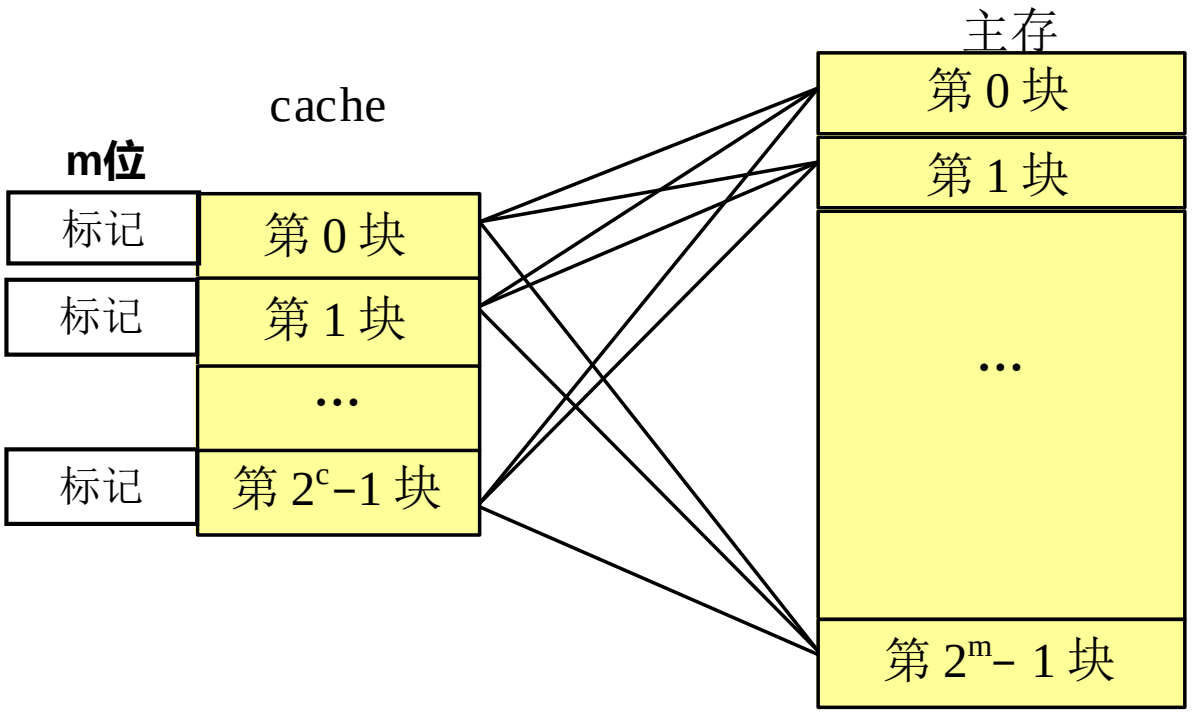
- 读写一个字，被放大为读写一个数据块
  - 缺失时要读整行，几百CC（假设MM命中）
    - 长时延问题
    - 如果MM不命中，则切换执行缺页ESR
- 如何减少CPU停顿?
  - 尽早重启（early restart）
    - 从块起始处开始读入。一旦请求字返回，CPU立即开始取数。适于I$。
  - 请求字优先（requested word first）
    - 从请求字处开始读，CPU立即恢复执行，同时并行取回数据块中的剩余数据，直至绕回。适于D$（？）。

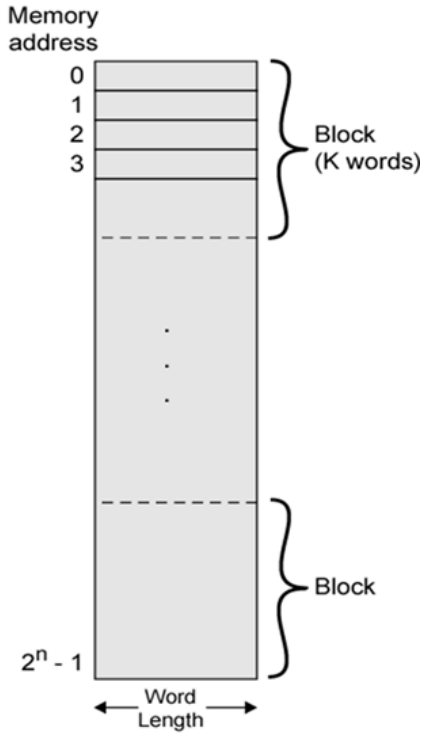






1. 全相联映射：字块位置任意





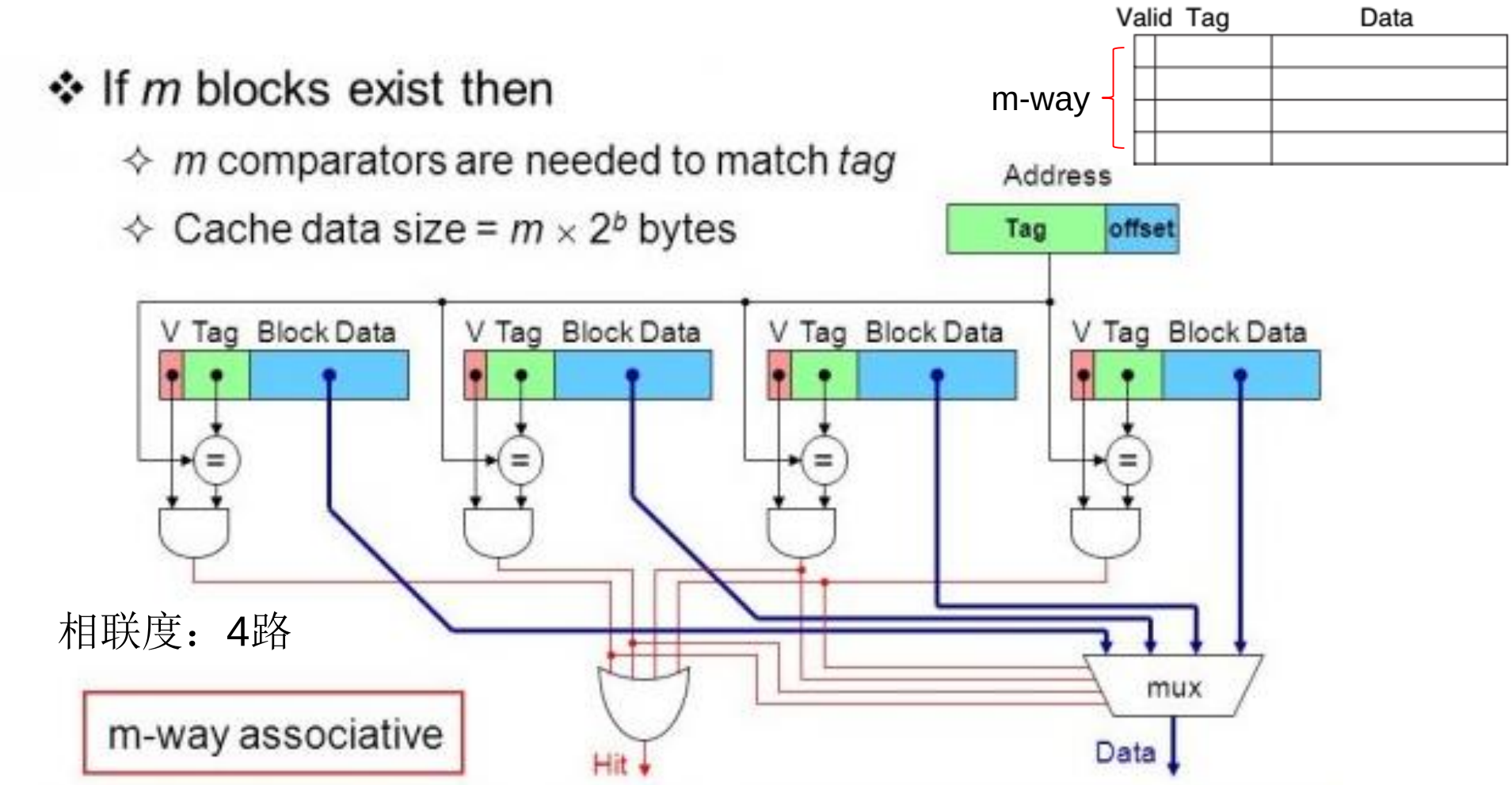
主存地址	主存块号	块内偏移地址
	Tag = m 位	b位

- Cache“标记位”多，比较位数长(m位)
  - 比较次数多（最坏m次），m个比较器（CAM）
- 块内偏移=字偏移 | 字节偏移

1. 全相联映像（续）


访问顺序 1 2 3 4 5 6 7 8
地址 22 26 22 26 16 4 16 18
字块号
块分配情况
0 1 2 3 4 5 6 7
操作状态 调进 调进 命中 命中 调进 调进 命中 调进

全相联映射数据访问过程



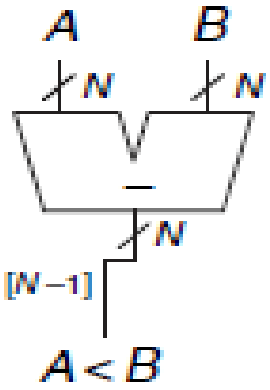
相联度：4路

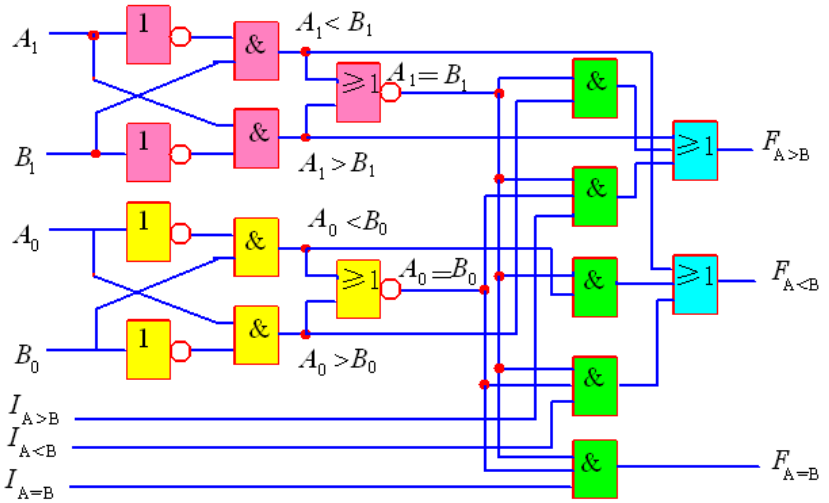
命中（比较tag和V），选行，选字？

命中比较与数据访问是并行or串行？——适于“小Cache”

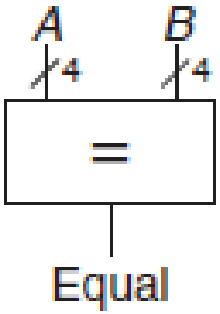
Comparator：数值大小，相等

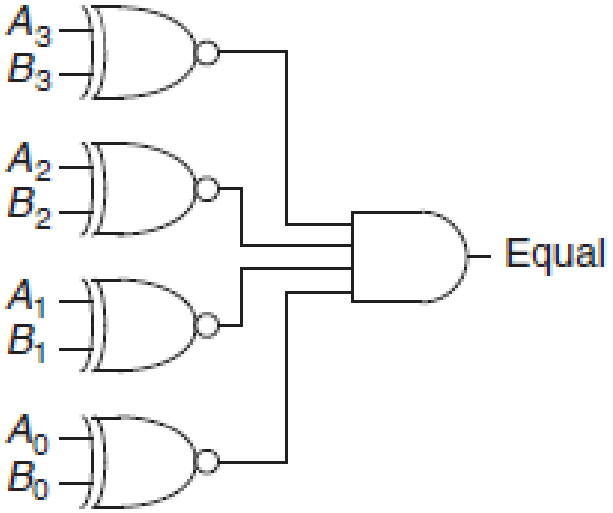
- magnitude comparator



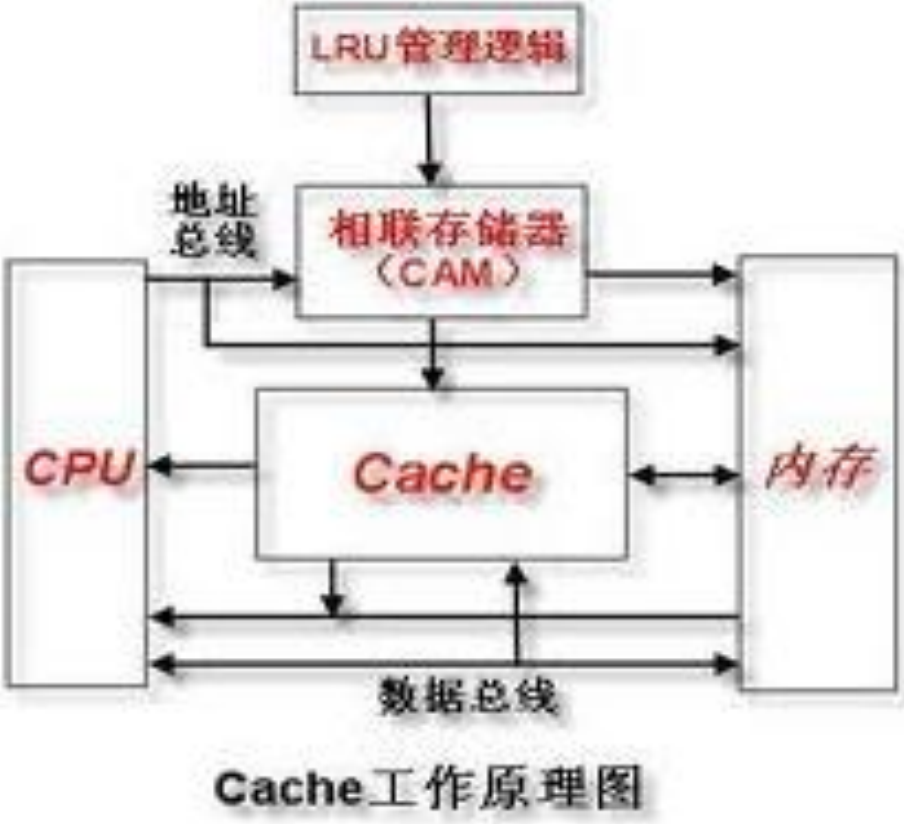


- equality comparator



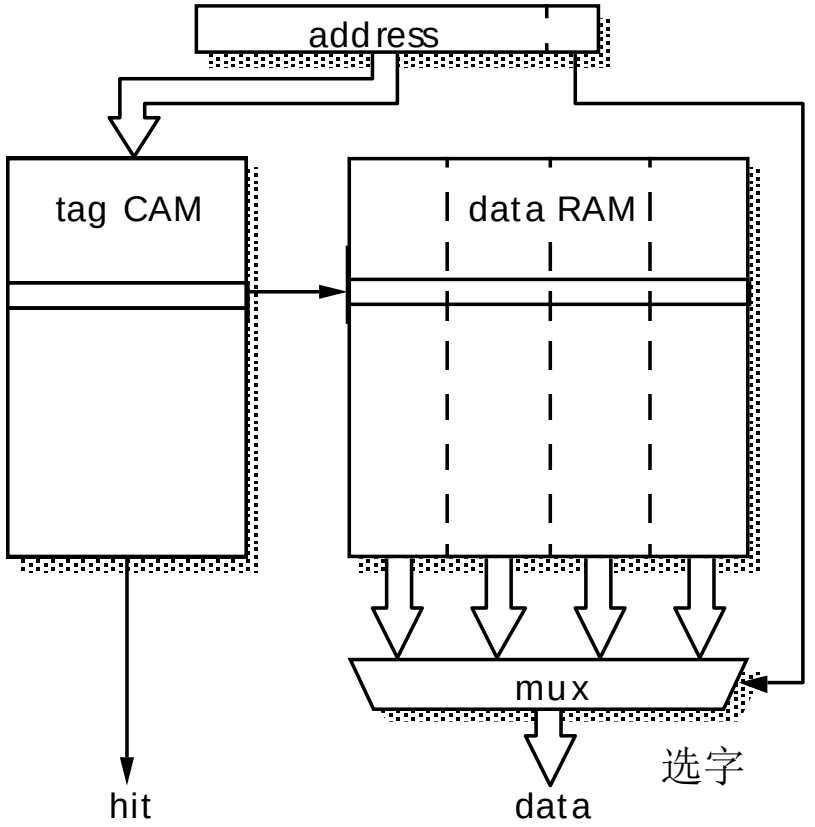


全相联Cache的Tag比较，$5.4.2【精解】



Cache工作原理图

CAM(Content Addressable Memory)



- CAM提高tag搜索性能：比较tag，选行
- 相联度≥8路时使用CAM

直接映射Cache：映射关系

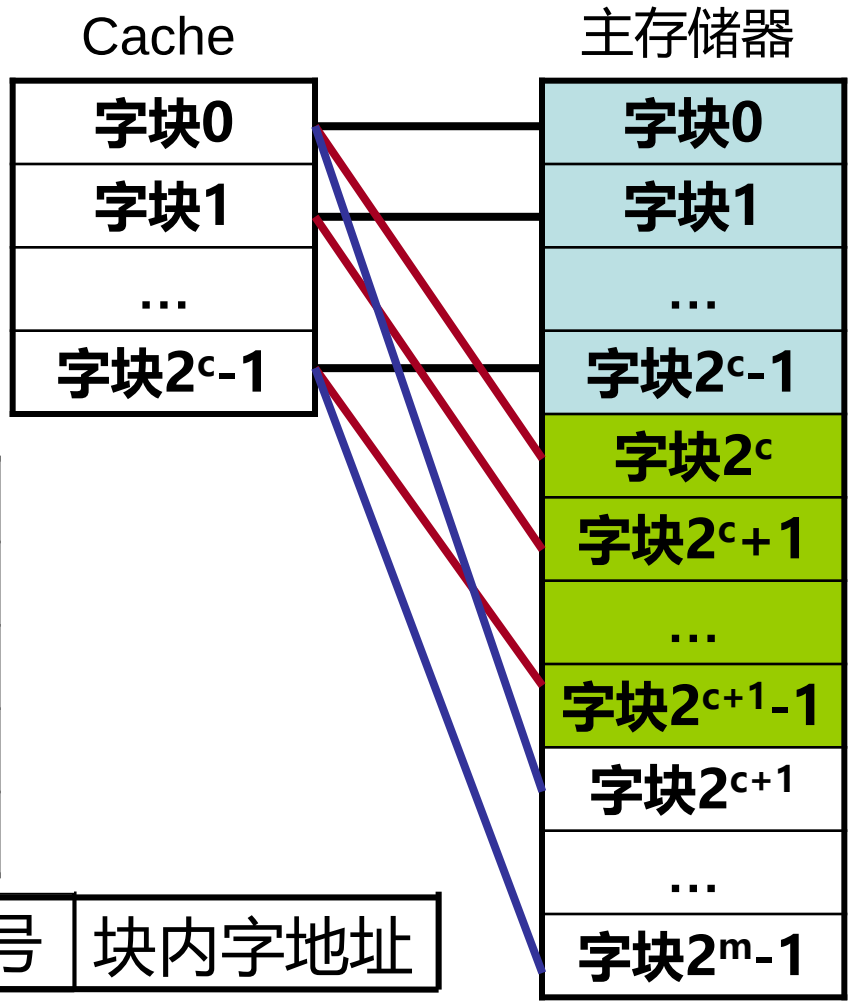
实现方式：字块 = 2^b字；主存分块，再按Cache行数分段

Cache一行一块，字块数：$C=2^c$

主存字块数：$M=2^m$

主存与缓存映射关系：i=j mod C

或 i=j mod 2^c



缓存块号 *i*	主存块号 *j*
0	0，C，……，2^m-C
1	1，C + 1，……，2^m-C + 1
……	……
C-1	C-1，2C-1，……，2^m-1

主存地址

主存段号	Cache行号	块内字地址

m=t+c位　　b位

例：direct mapped Cache

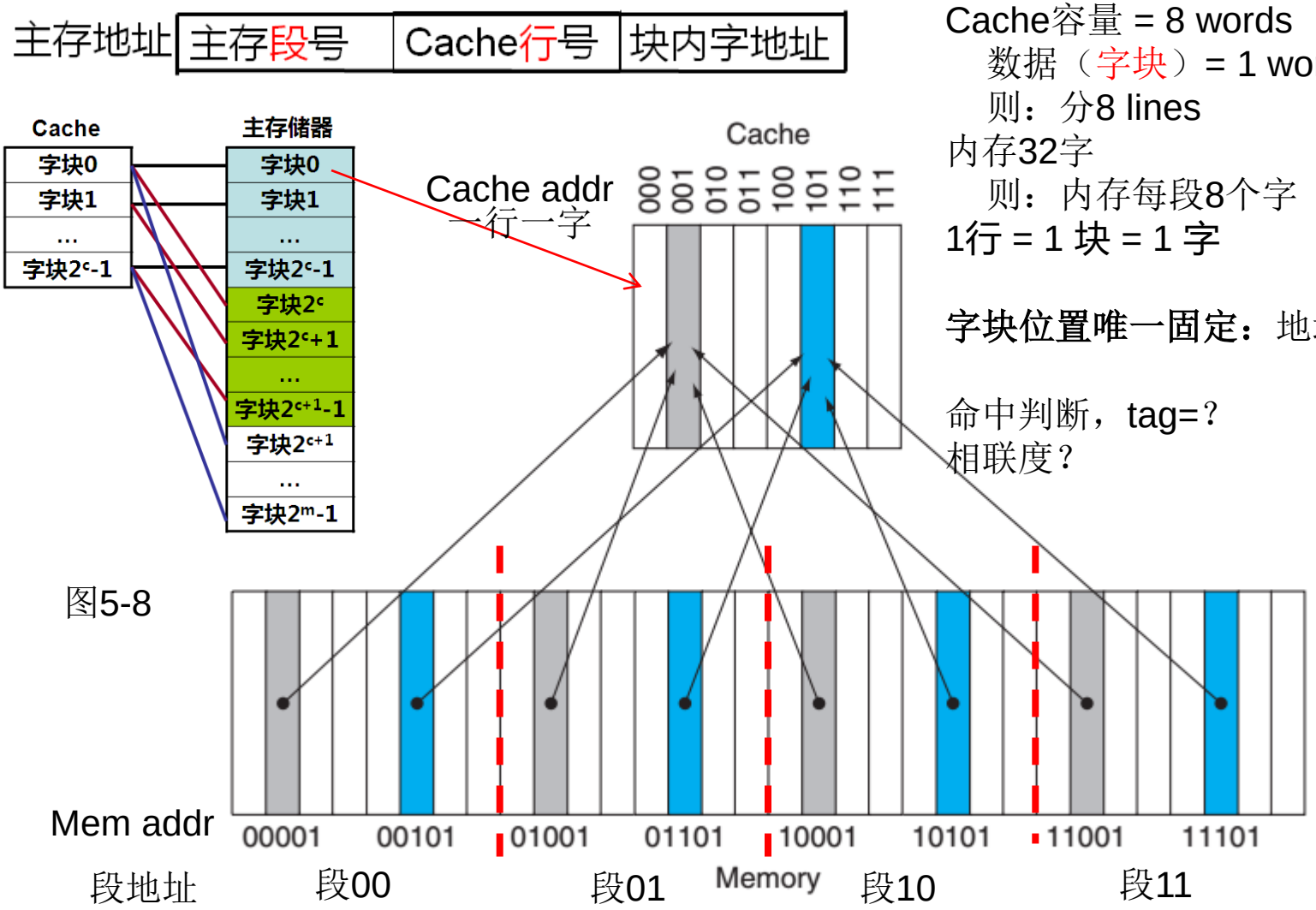


图5-8

Cache容量 = 8 words
　数据（字块）= 1 word
　则：分8 lines
内存32字
　则：内存每段8个字（块），分4段
1行 = 1 块 = 1 字

字块位置唯一固定：地址冲突？

命中判断，tag=？
相联度？

地址域：Tag（段#）| Index（Cacheline#）| BlockOffset | ByteOffset

例：Cache access

图5-9

- 9次读访存
- Cache=8行（块）
  - 初始为空，V=N

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

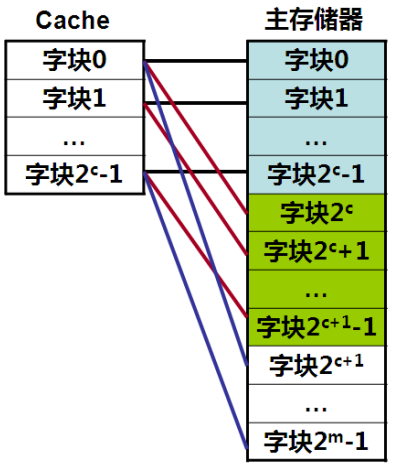
d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

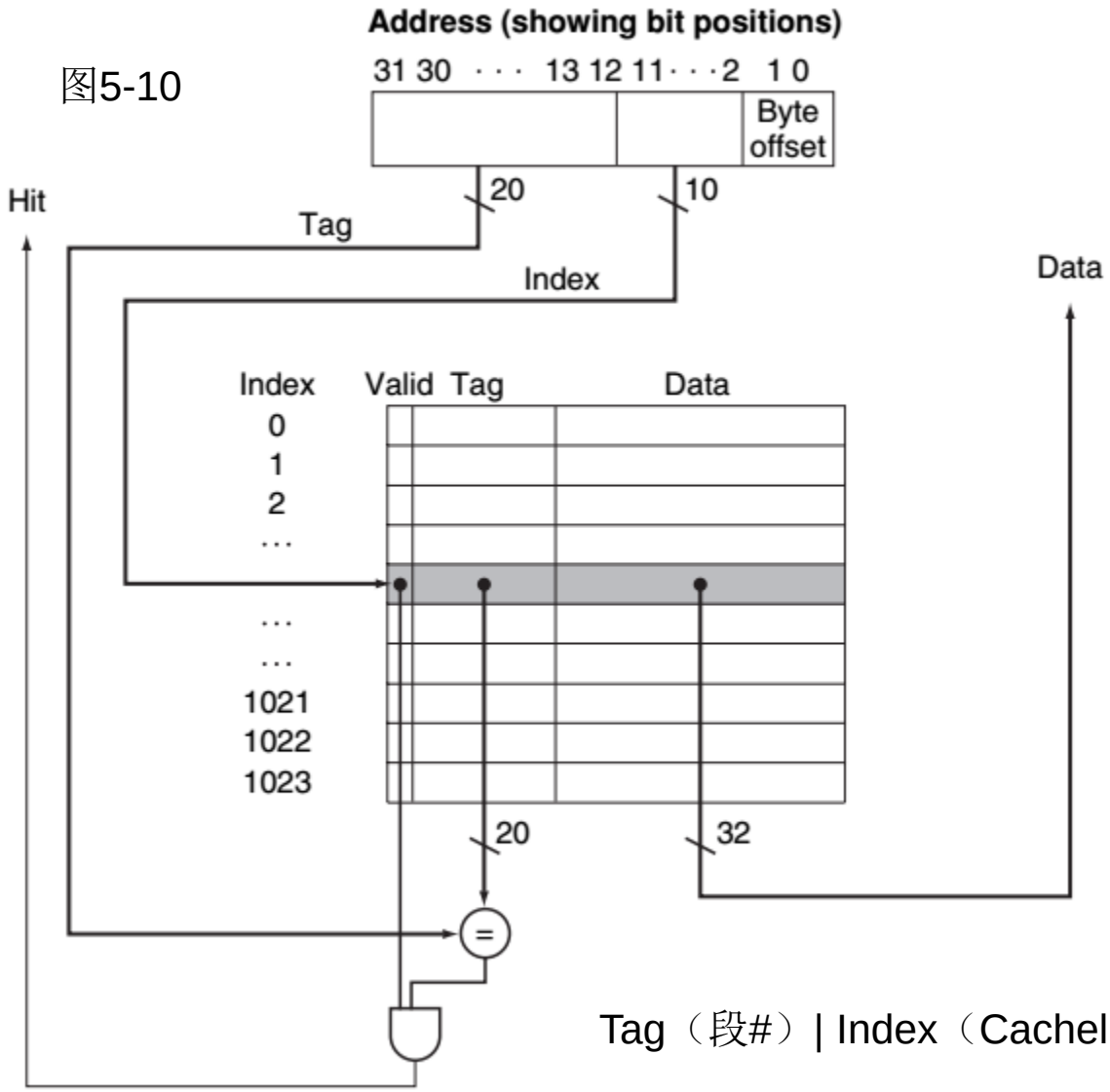


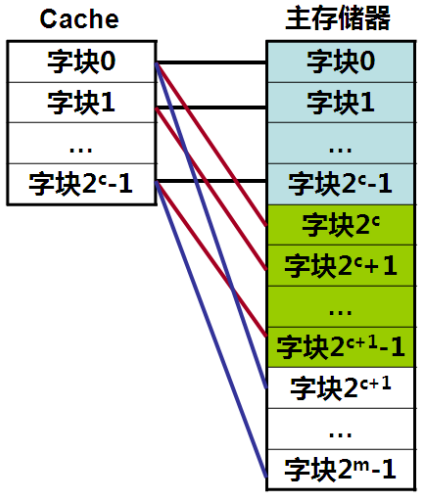
主存地址 | 主存段号 | Cache行号 | 块内字地址

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	miss (5.9c)	(11010_{two} mod 8) = 010_{two}
22	10110_{two}	hit	(10110_{two} mod 8) = 110_{two}
26	11010_{two}	hit	(11010_{two} mod 8) = 010_{two}
16	10000_{two}	miss (5.9d)	(10000_{two} mod 8) = 000_{two}
3	00011_{two}	miss (5.9e)	(00011_{two} mod 8) = 011_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}
18	10010_{two}	miss (5.9f)	(10010_{two} mod 8) = 010_{two}
16	10000_{two}	hit	(10000_{two} mod 8) = 000_{two}

例：读操作，命中?

图5-10





本例：data = 1 word；
块内字地址= BlockOffset= 0位

Tag与Data并行

Tag（段#）| Index（Cacheline#）| BlockOffset | ByteOffset

FastMATH处理器：直接映射

ARM芯片设计公司Intrinsity
$5.3.4：WriteBuf容量为1行

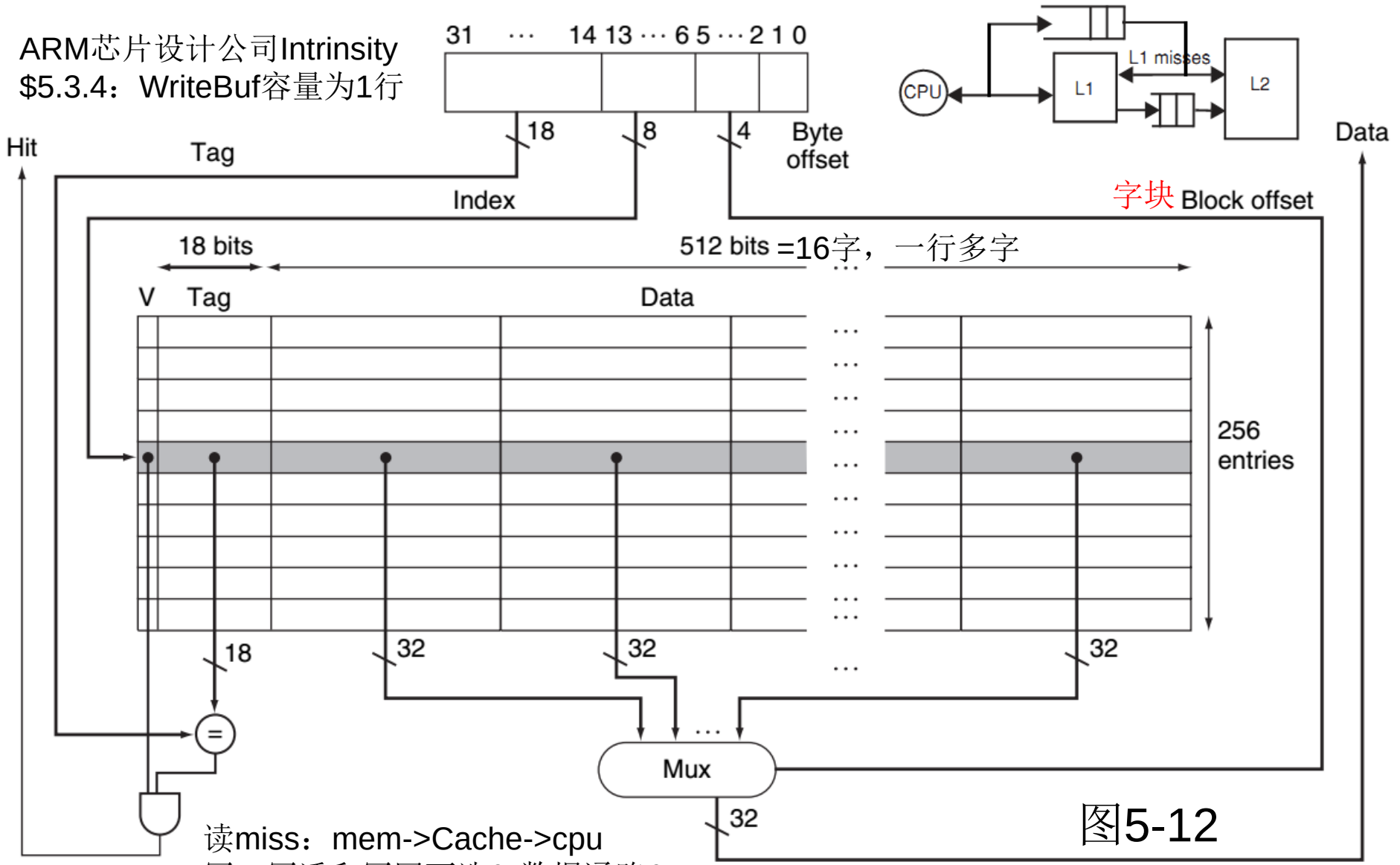


图5-12

读miss：mem->Cache->cpu
写：写透和写回可选？数据通路？

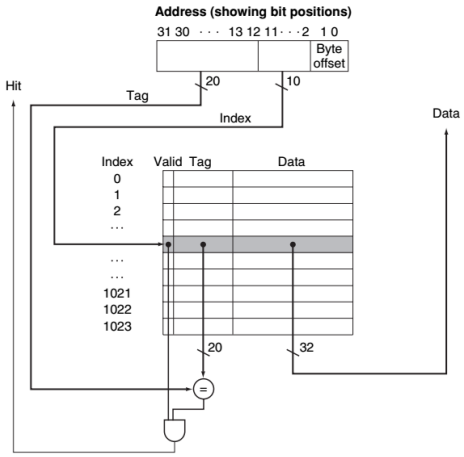
直接映象Cache访问：颠簸（乒乓）

访问顺序	1	2	3	4	5	6	7	8	
块地址	22	26	22	26	16	4	16	18	字块号
块分配情况	–	–	–	–	16	16	16	16	0
	–	–	–	–	–	–	–	–	1
	–	26	26	26	26	26	26	18	2
	–	–	–	–	–	–	–	–	3
	–	–	–	–	–	4	4	4	4
	–	–	–	–	–	–	–	–	5
	22	22	22	22	22	22	22	22	6
	–	–	–	–	–	–	–	–	7
操作状态	调进	调进	命中	命中	调进	调进	命中	替换	

如果连续访问26和18？

直接映象特点

- 优点：实现简单
  - 数据块映射位置固定
  - 一次比较：只需利用主存地址的某些位直接判断，就可确定所需字块是否在缓存中。
  - Fast：data先于tag比较
  - 无需count位



- 缺点：冲突，效率低。
  - 因为每个主存块固定地对应某个缓存块（有2^t个主存字块对应同一个Cache字块），如果这2^t个字块中有两个或两个以上的主存字块要调入Cache，必然会发生冲突。

2. 组相联映象（2 way-set-associated）

Cache分组，

主存分段，

段大小＝组数

r = 组内块数－1

相联度

Tag = 段号

r = 0？c组1路，直

r = c？1组c路，全

Cache（r = 1）

标记	字块0
标记	字块1
标记	字块2
标记	字块3
……	……
标记	字块2^c-2
标记	字块2^c-1

第0组

第1组

第2^{c-r}-1组

主存储器

字块0

字块1

……

字块2^{c-r}-1

字块2^{c-r}

字块2^{c-r}+1

……

字块2^{c-r+1}

……

字块2^m-1

段

主存地址

主存块标记	组地址	块内地址
s=t+r位	q=c-r位	b位

m位

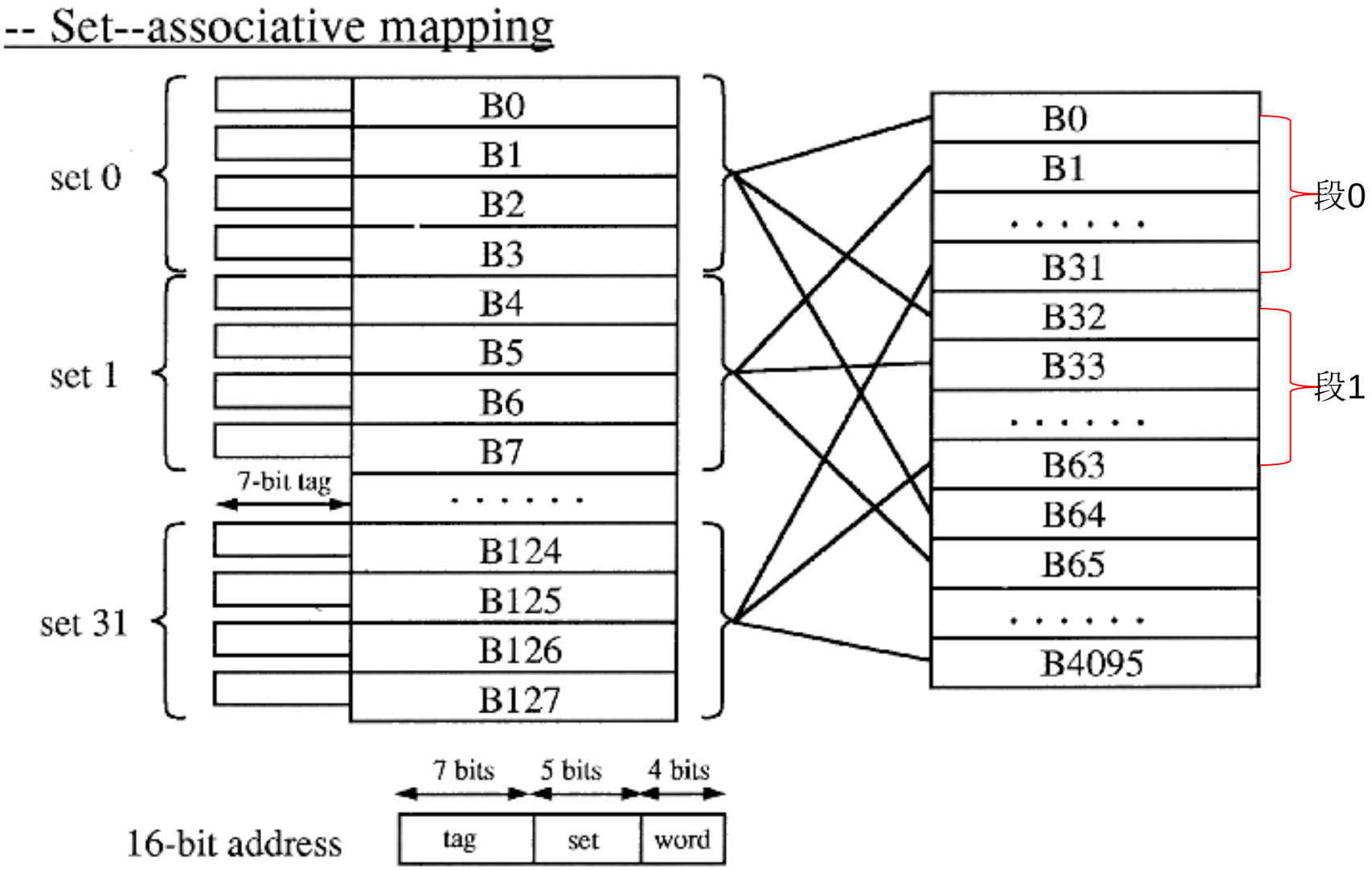
2. N路（X）组相联映象（续）

- 原理：把Cache分为Q(=2^q)组，每组有R(=2^r)块，且

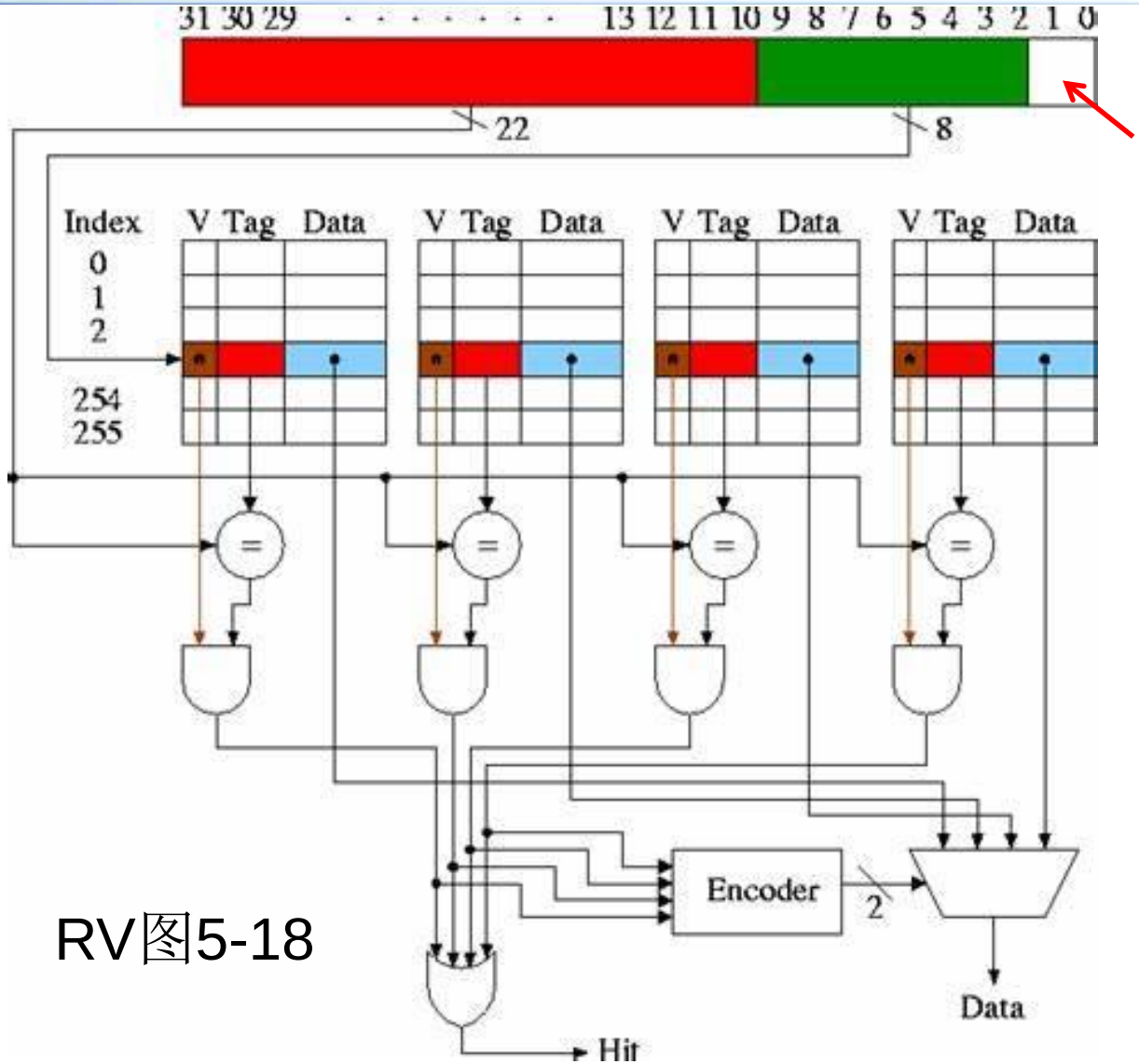
  $i=j \bmod Q$

- 其中，i为缓存的组号，j为主存的块号
  - **在主存块和Cache的组之间，为直接映象关系；**
  - **主存块可以映射到对应组内的任何一块，为全相联映象的关系。**
- 相联度，degree of associativity
  - r：一组的块数（路数，行数）
  - r＝0，直接相联；r＝c，全相联。

Ex：4-way set associative Cache



Ex：4-way set associative Cache



RV图5-18

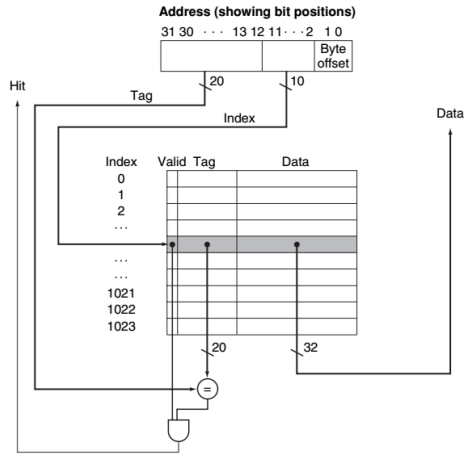
主存地址

主存子块标记	组地址	子块内地址
s=t+r位	q=c-r位	b位

m位

- 一组4路（line）
- 一路一字（4字节）

- 定位set
- 比较tag
- 选路选字合二为一



- 块（data）大小，组大小，Cache大小，内存段大小，内存大小?

主存数据块在Cache中的位置，$5.4.1

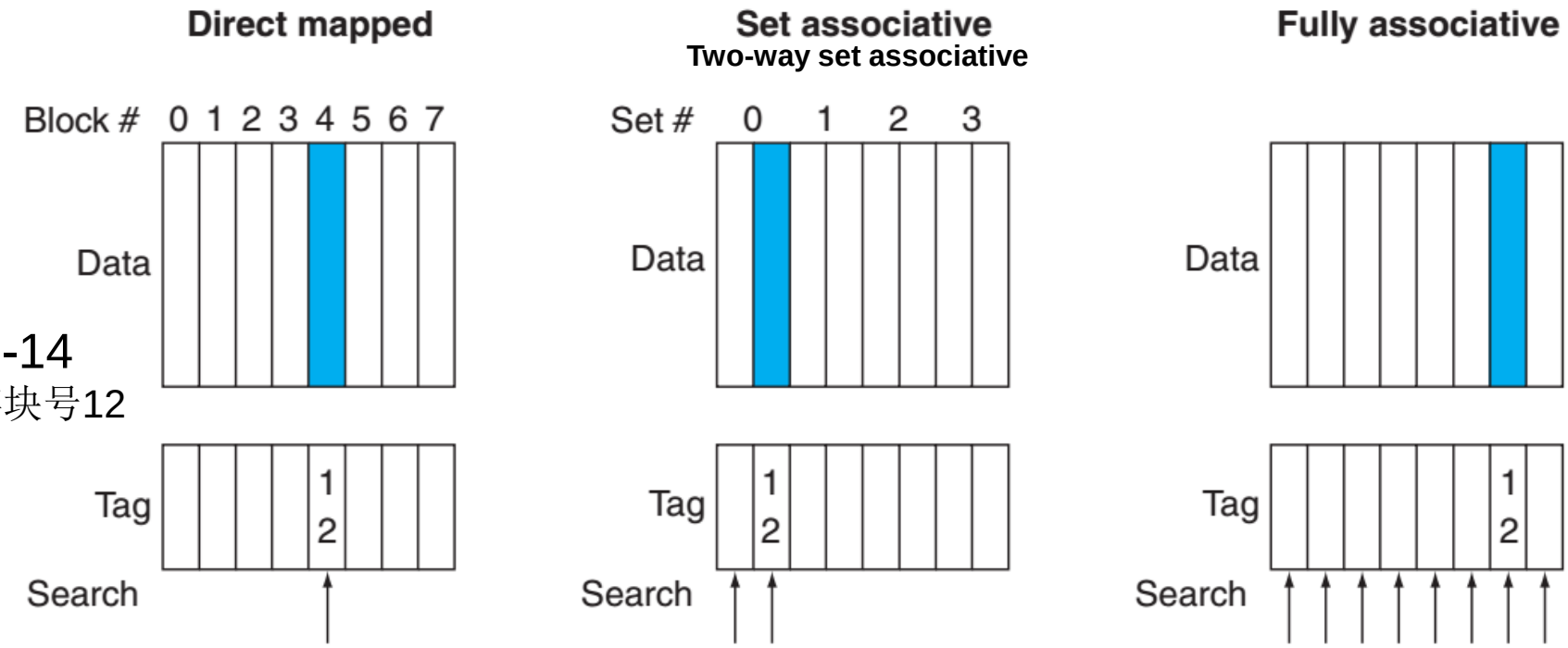


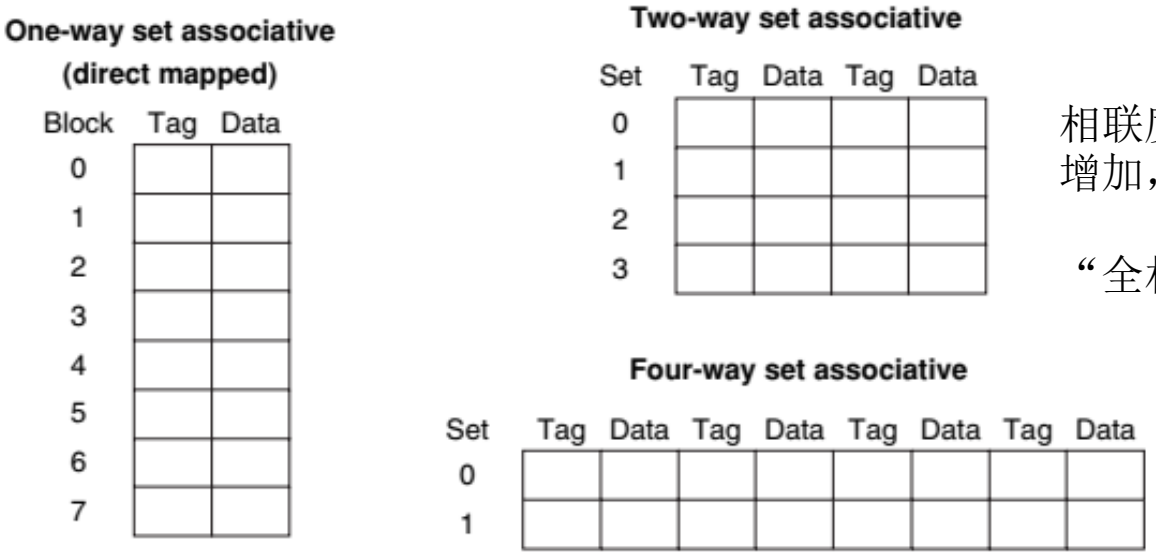
图5-14

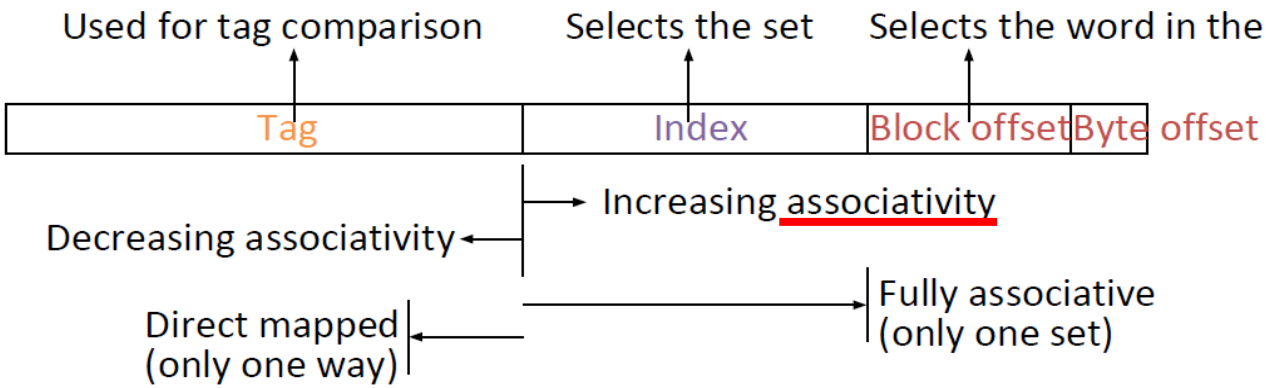
主存块号12

- 直接映射：内存块号 mod Cache块数
  - (Block number) modulo (Number of *blocks* in the cache)
- 组相联：内存块号 mod Cache组数，组内任一
  - (Block number) modulo (Number of *sets* in the cache)
- 全相联：任一

相联度associativity = Number of ways

图5-15





相联度（路数）增加，命中率增加，命中比较时间增加

“全相联用于小容量Cache”！

Cache容量（总行数）=组数×相联度

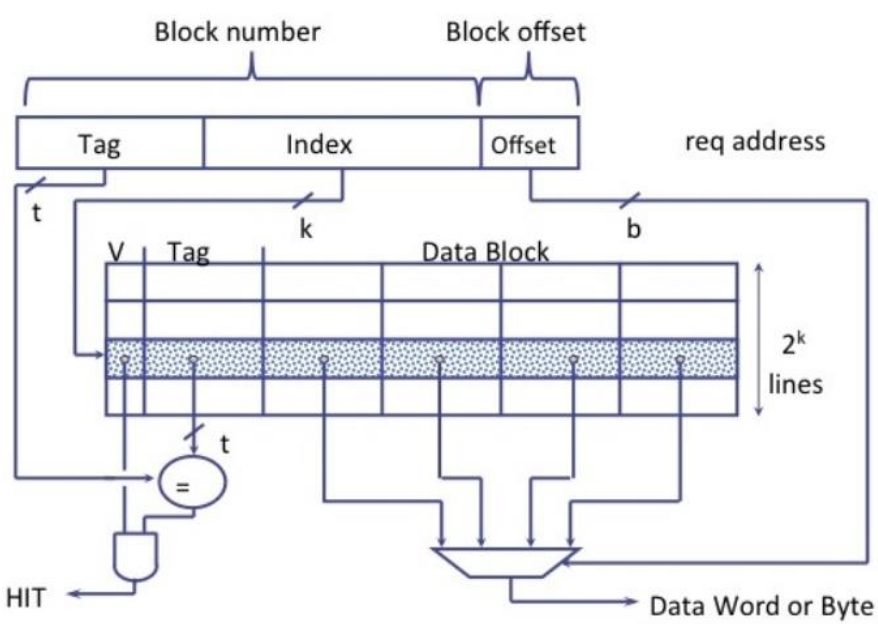
块/行/路，组/段

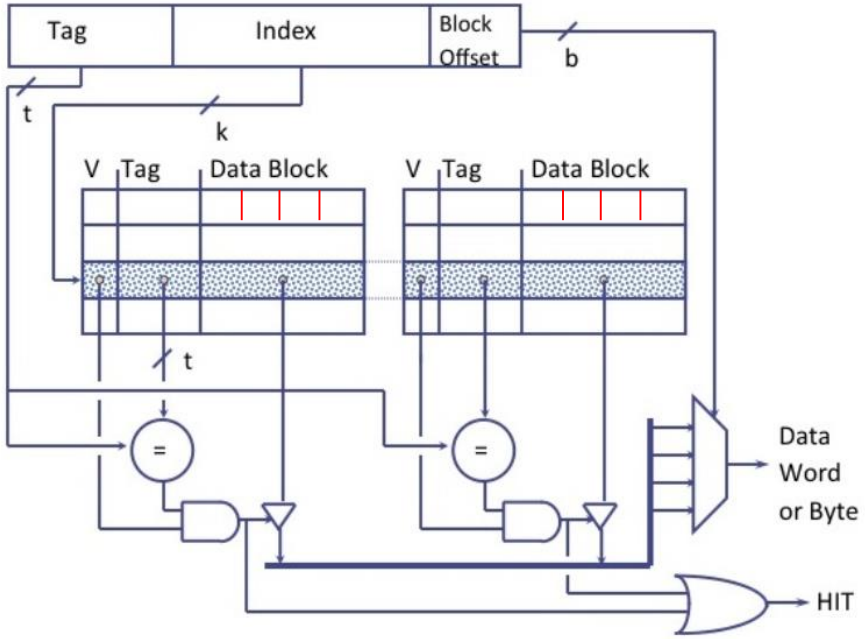
1）Cache和Mem分块；

2）Cache分组，Mem分段

直接映射，2路组相联

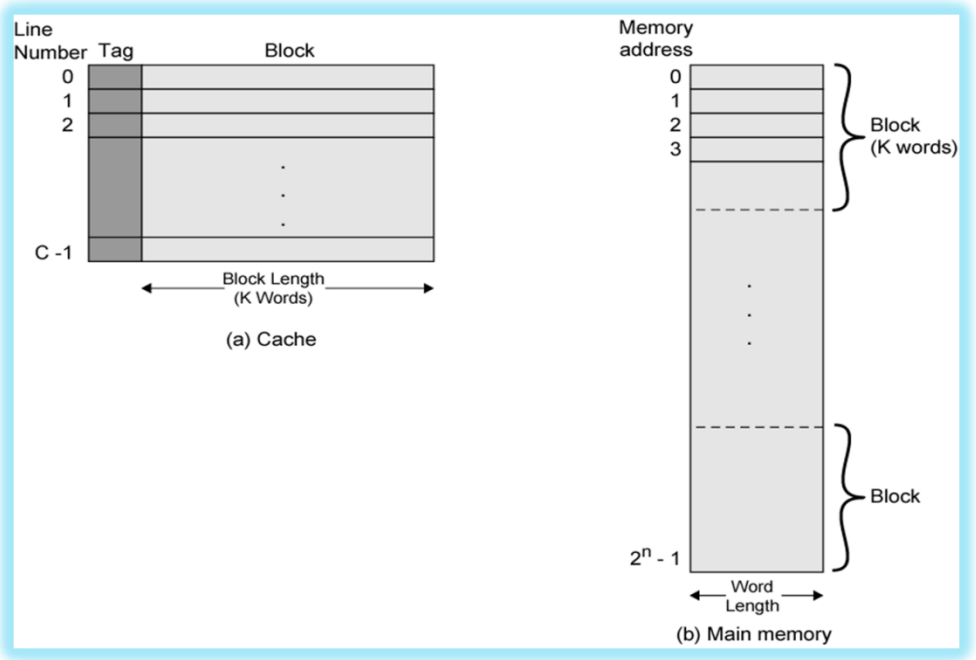
- N comparators vs. 1
- 组相联：Extra MUX delay for the data
  - 先选way，再选word
  - Data comes AFTER Hit/Miss decision and set selection
- 直接映射
  - 仅1 way
  - Cache Block is available BEFORE Hit/Miss

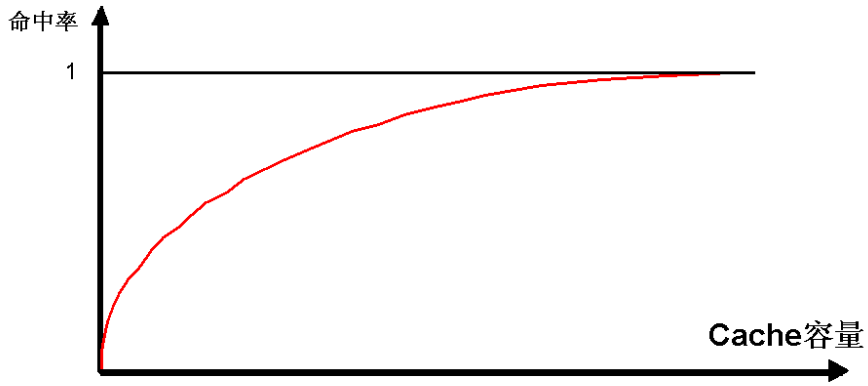




Cache命中率

- 容量、块长和相联度（映射方式）影响Cache效率。
- Cache容量越大，命中率越高。
  - 当Cahce容量达到一定值时，命中率不会因容量的增大而明显提高。
  - Cache容量大，成本增加，功耗增加。





Cache容量、块（行）大小、命中率

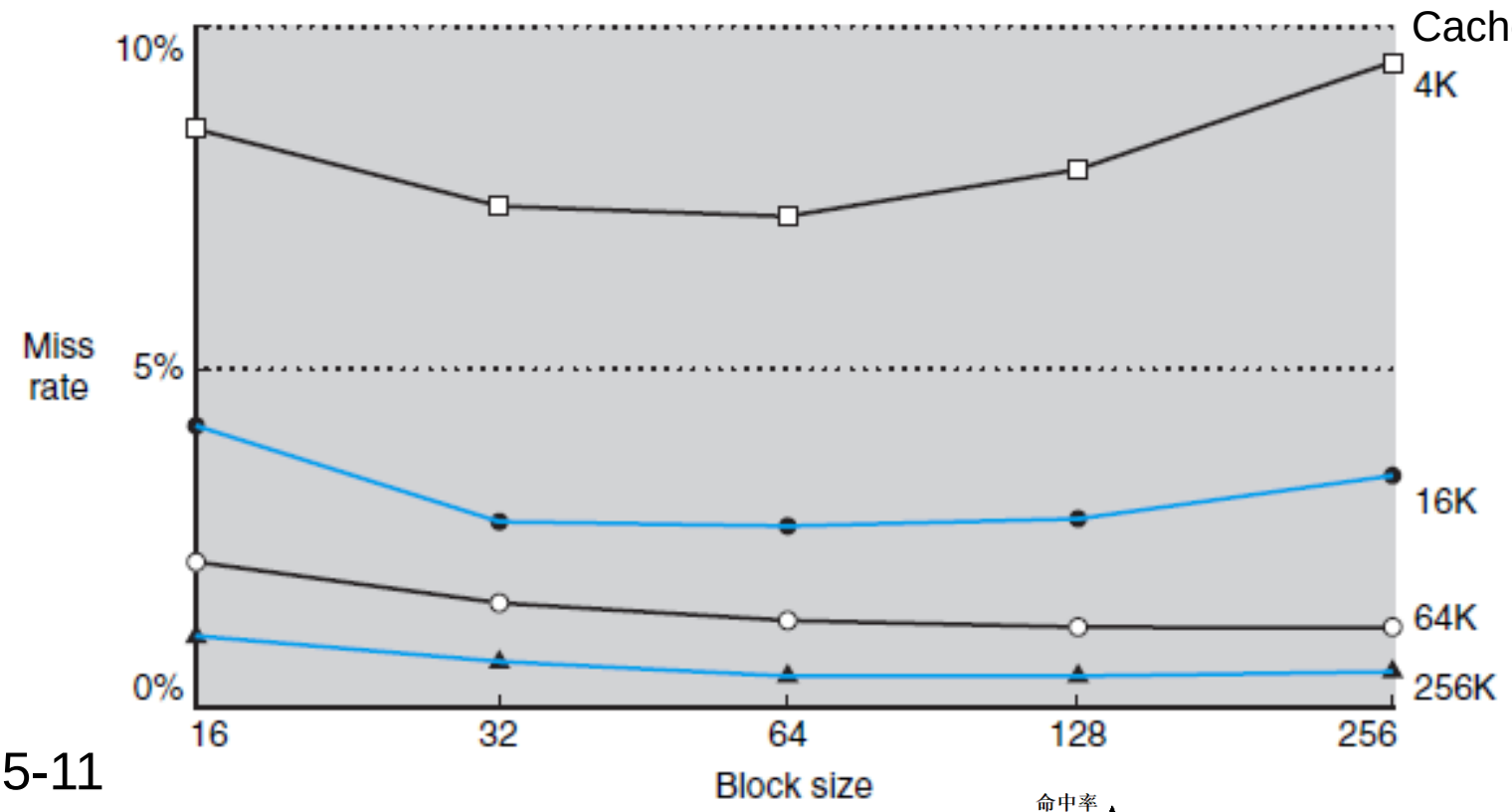
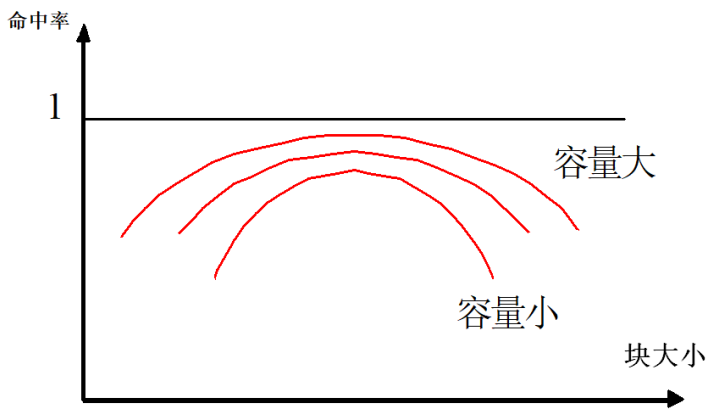


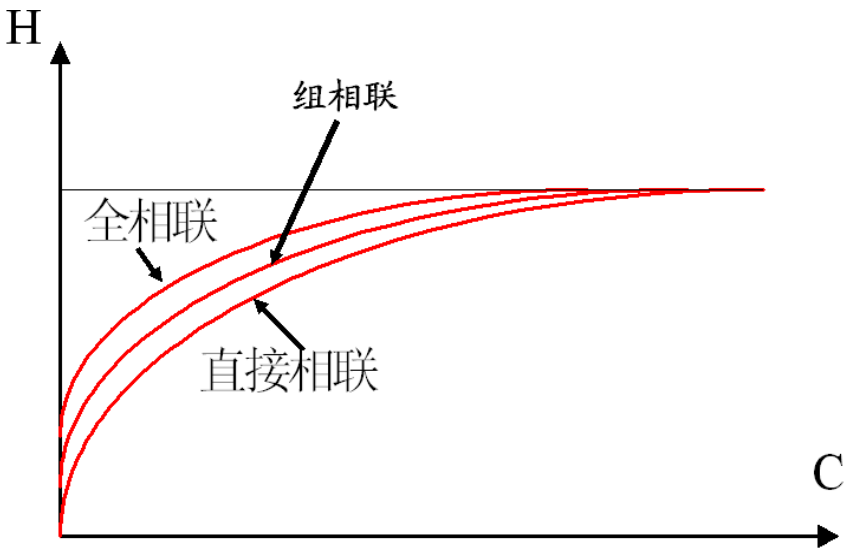
图5-11



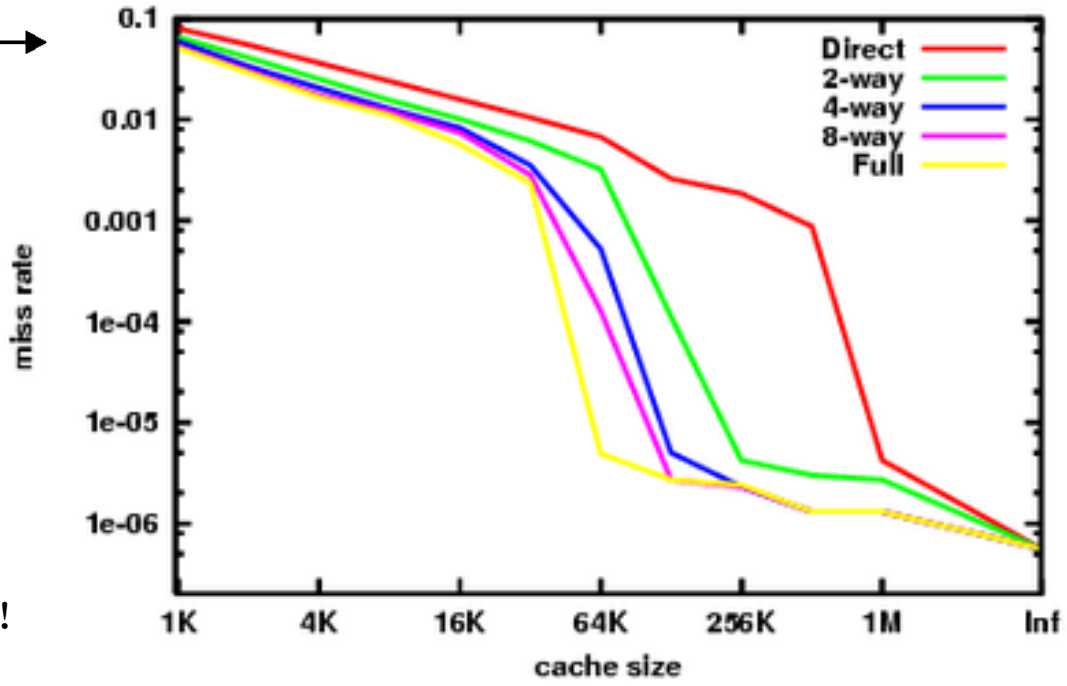
块大小增加，命中率增加；
Cache容量固定，块大小增加，局部性降低，命中率降低；
块大小增加，失效损失增加。

$5.3.5：“更大的数据块会降低失效率”？

映射方式比较：命中率与相联度



4路以上效果不显著



H&P $5.2，命中时间：

- 直接映射比2-ways快1.2~1.5倍
- 2-ways比4-ways快1.02~1.11倍
- 4-ways比全相联快1.0~1.08倍

第一级Cache：
小且简单（直接映射），减少命中时间！

$5.4.1例，命中率与相联度

- 直接映射

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

- 两路组相联

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

- 全相联

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

例

- 设Cache的速度是主存的5倍，命中率为95%，则采用Cache后性能提升多少?

  系统平均访问时间=*0.95*t+0.05*5t=1.5t*

  性能提升=*5t/1.5t=3.33倍*

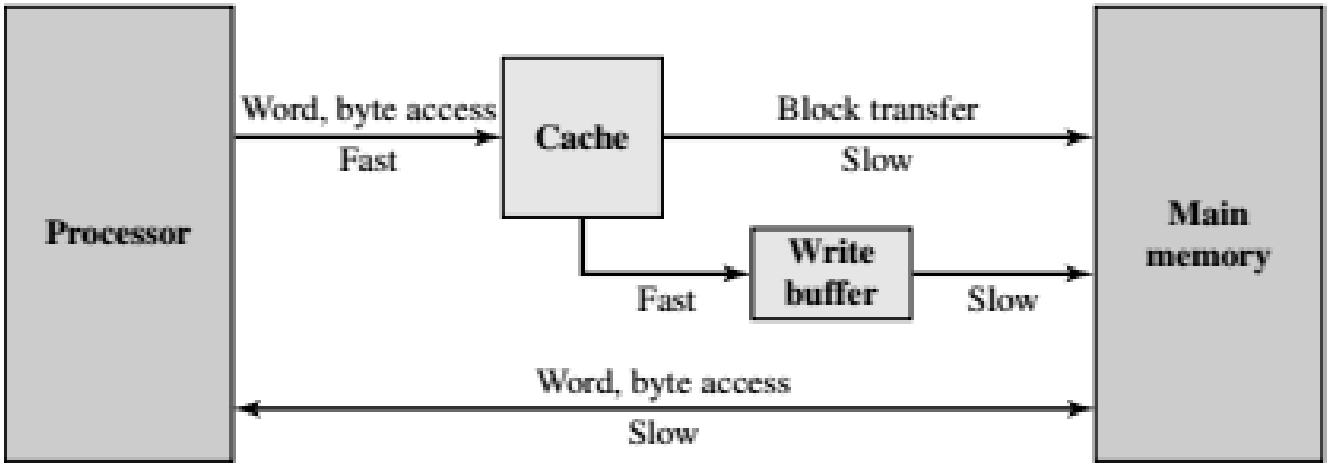


Figure 4.19 ARM Cache and Write Buffer
William Stallings，Computer Organization and Architecture，8th Edition，2010

例1：

- 某PC主存容量分2048块,每块512B,Cache容量8KB,分为16块,每块512B。
  - 用直接映象时，主存应被分几段？Cache标记几位?
  - 用全相联映象，Cache标记几位?
  - 用组相联映象，Cache每组2行（每组2块，两路组相联），主存应划分为几段？每段几块？Cache标记几位?

段号	组	路/块号	字/字节

例2：

设有一个cache的容量为2K字，每个块为16字，求

(1) 该cache可容纳多少个块?

(2) 如果主存的容量是256K字，则有多少个块?

(3) 主存的地址有多少位？cache地址有多少位?

(4) 在直接映像方式下，主存中的第*i*块映像到cache中哪一个块中?

(5) 进行地址映像时，存储器的地址分成哪几段？各段分别有多少位

解：(1) cache中有2048/16=128个块。

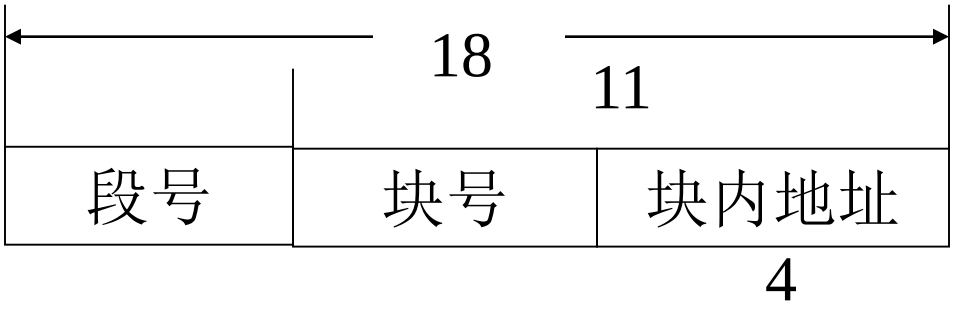
(2) 主存有256K/16=16384个块。

(3) 主存容量为256K=2^{18}字，字地址有18位。

cache容量为2K=2^{11}字，字地址为11位。

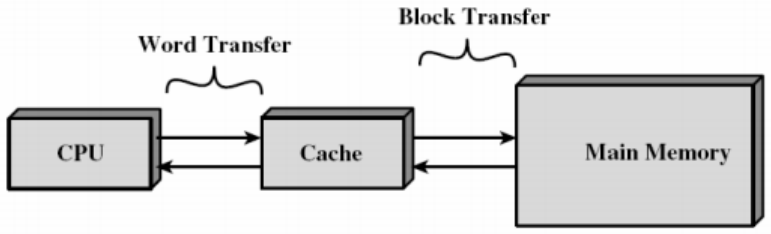
(4) 在直接映像方式下，主存中的第*i*块映像到cache中第 $i \bmod 128$个块中。

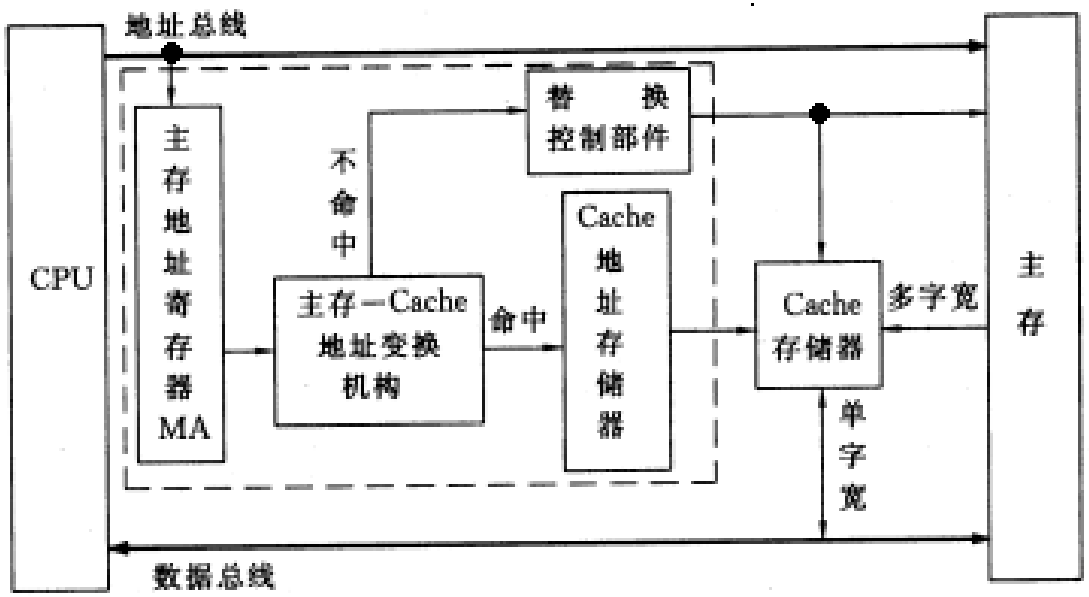
(5) 段号7位，块号为7位，块内字地址为4位。



替换：$5.4.3

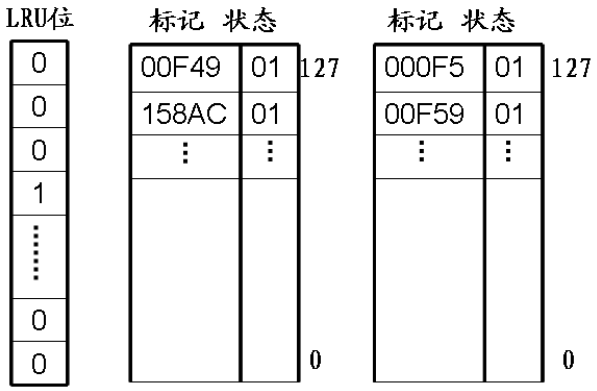
- 以块（line）为单位
  - 全相连：任意行
  - 组相联：固定组，任意路（行）
  - 直接映射：固定行





替换算法

- 最优替换（OPT）：未来最不可能使用者
  - 置换最长时间不会被使用的行：预知work sets
- 随机法，FIFO，最久优先，最少访问
  - FIFO：实现方便，但不能正确反映程序的局部性
    - 最先进入的字块也可能是目前经常要用的字块。
- LRU：最近最少使用
  - 计时法（绝对）：替换计时最长的cache line
  - 计数法（近似）：替换计数最大者——按访问次数
    - 一位计数：**NRU**（最近未使用），多用！
      - 组计数，*路计数（or 随机）*
    - 两位计数
  - 堆栈法
  - 相联度增大，实现成本高



时间 t	1	2	3	4	5	6	7	8	9	10	实际命中次数
地址流	P1	P2	P1	P5	P4	P1	P3	P4	P2	P4	
先进先出算法（FIFO 算法）	1	1	1	1*	4	4	4*	4*	2	2	
		2	2	2	2*	1	1	1	1*	4	
				5	5	5*	3	3	3	3*	
	调入	调入	命中	调入	替换	替换	替换	命中	替换	替换	2 次
最近最少使用算法（LRU 算法）	1	1	1	1	1	1	1	1*	2	2	
		2	2	2*	4	4	4*	4	4	4	
				5	5*	5*	3	3	3*	3*	
	调入	调入	命中	调入	替换	命中	替换	命中	替换	命中	4 次
最优替换算法（OPT 算法）	1	1	1	1	1	1*	3*	3*	3	3	
		2	2	2	2*	2	2	2	2	2	
				5*	4	4	4	4	4	4	
	调入	调入	命中	调入	替换	命中	替换	命中	命中	命中	5 次

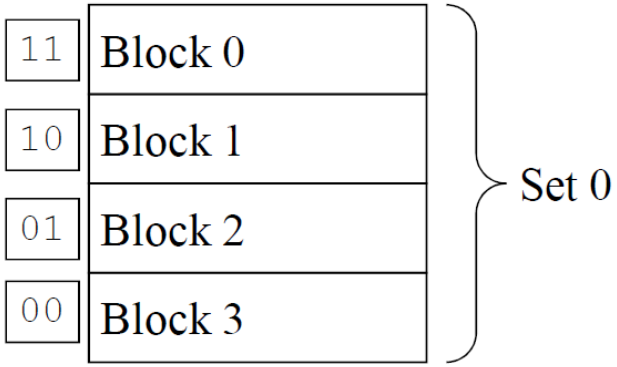
三种页面替换算法对同一个页地址流的调度过程

Implementing LRU replacement

- LRU：路，组

Cache Line = VCD + tag + block(K words)

- 例1：4-ways组相联，按路计数，换出计数最大者
  - Hit
    - Increment lower counters，命中者保持
    - Reset counter to 00
  - Miss
    - Replace the 11
    - Set to 00
    - Increment all other counters



- 例2：2-ways组相联
  - 按组计数

Offset

Index

		V	Tag	00	01	10	11	
0	0	X	XXX	0x??	0x??	0x??	0x??	LRU
	1	X	XXX	0x??	0x??	0x??	0x??	X
1	0	X	XXX	0x??	0x??	0x??	0x??	LRU
	1	X	XXX	0x??	0x??	0x??	0x??	X

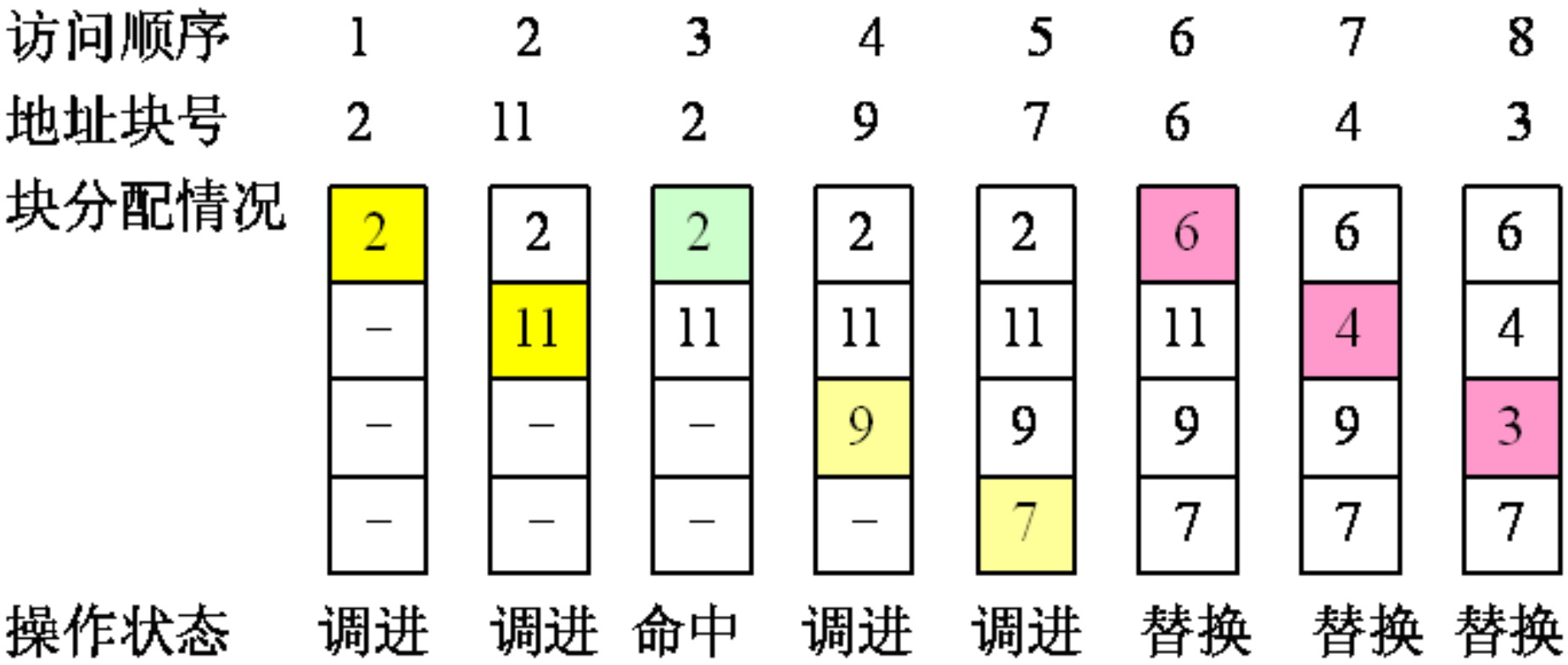
例：

设程序有5个信息块，Cache空间为3块，地址流为：

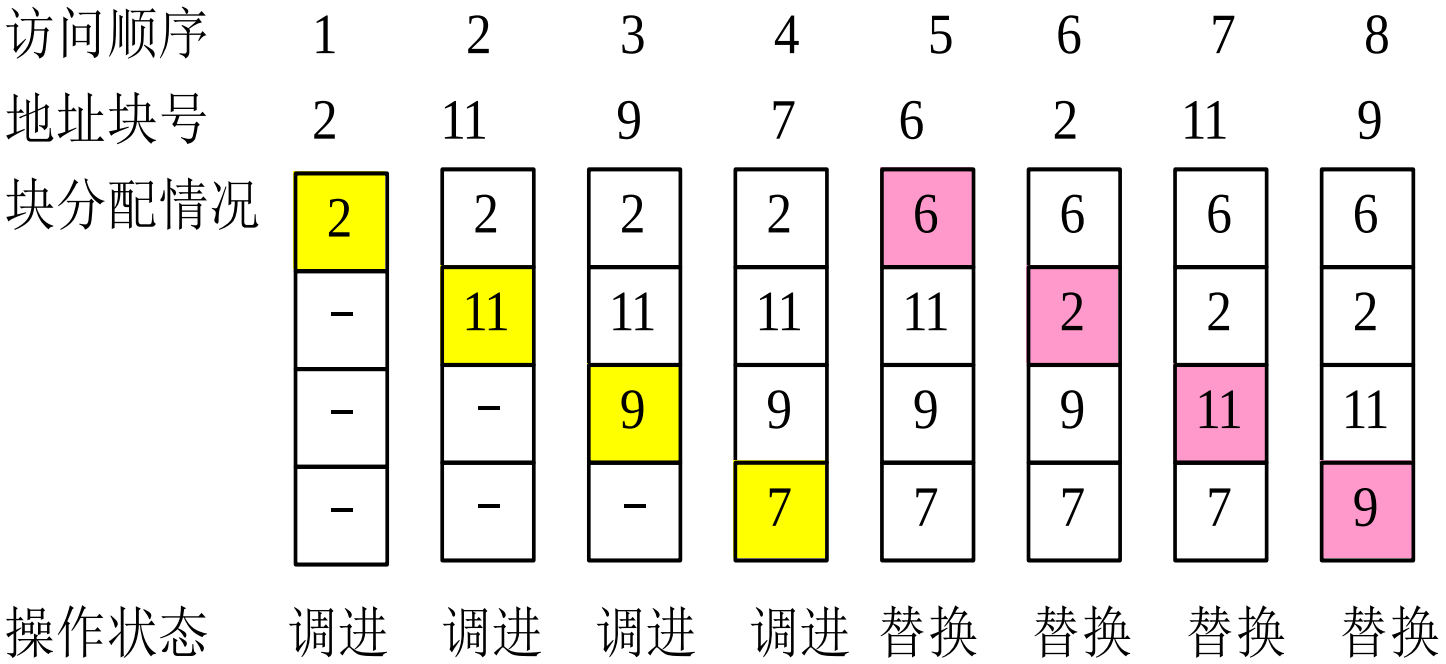
P1, P2, P1, P5, P5, P1, P3, P4, P3, P4

给出FIFO、LRU两种页面替换算法对这3块Cache的使用情况，包括调入、替换和命中等。

FIFO替换算法


访问顺序 1 2 3 4 5 6 7 8
地址块号 2 11 2 9 7 6 4 3
块分配情况
2 – – –
2 11 – –
2 11 – –
2 11 9 –
2 11 9 7
6 11 9 7
6 4 9 7
6 4 3 7
操作状态 调进 调进 命中 调进 调进 替换 替换 替换

颠簸(Thrashing)现象：Cache 污染



先进先出替换方式下的 cache 内容颠簸情况

近期最少使用算法LRU

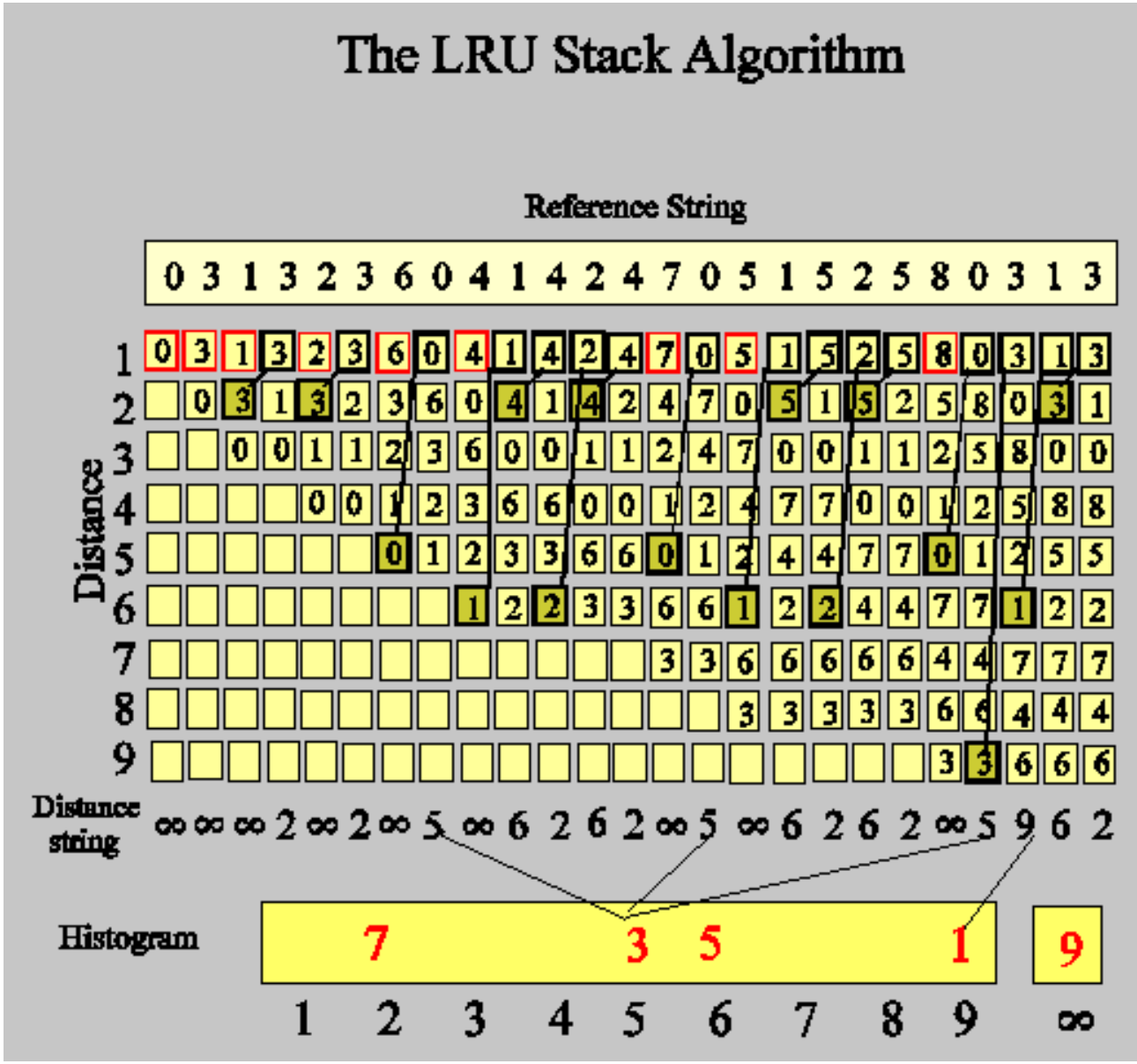
例：选最近4次访问期间最少使用Cache块作为被替换的块。

访问顺序	1	2	3	4	5	6	7	8
地址块号	2	11	2	9	7	6	4	3
块分配情况	2	2	2	2	2	2*	4	4
	–	11	11	11	11*	6	6	6
	–	–	–	9	9	9	9*	3
	–	–	–	–	7	7	7	7*
操作状态	调进	调进	命中	调进	调进	替换	替换	替换

*表示将要被替换者，计时？计数？

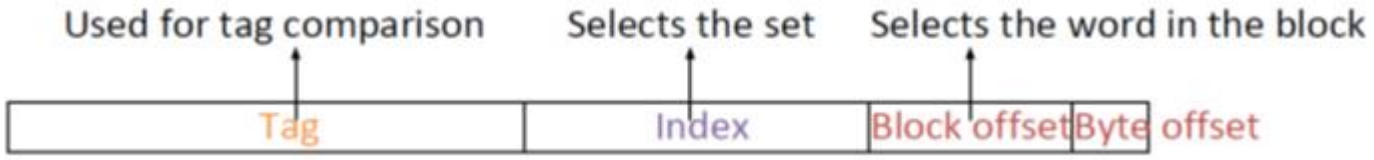
LRU stack算法

- stack distance
  - 访问某line时其在栈中的位置
  - 第一次访问时 = ∞
- 替换哪一行?
  - 双向链表
- histogram
  - 局部性



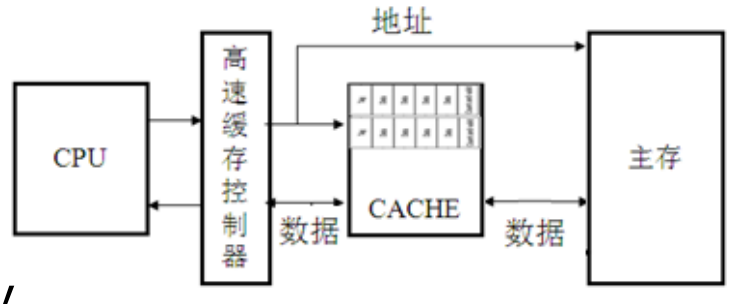
A Simple Cache：需求规格，$5.9，$5.12

- Direct mapped cache
- Block size is 4 words (16 bytes or 128 bits)
- Cache size is 16 KB, so it holds 1024 blocks
- 32bit byte addresses breakdown
  - Cache index is 10 bits
  - Block offset is 4 bits
  - Tag size is 32 - (10 + 4) or 18 bits（256K块）
- Write-back using write allocate
  - includes a valid bit and dirty bit per block

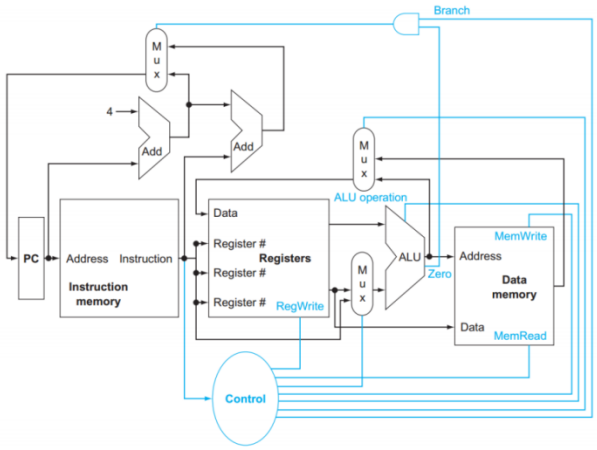


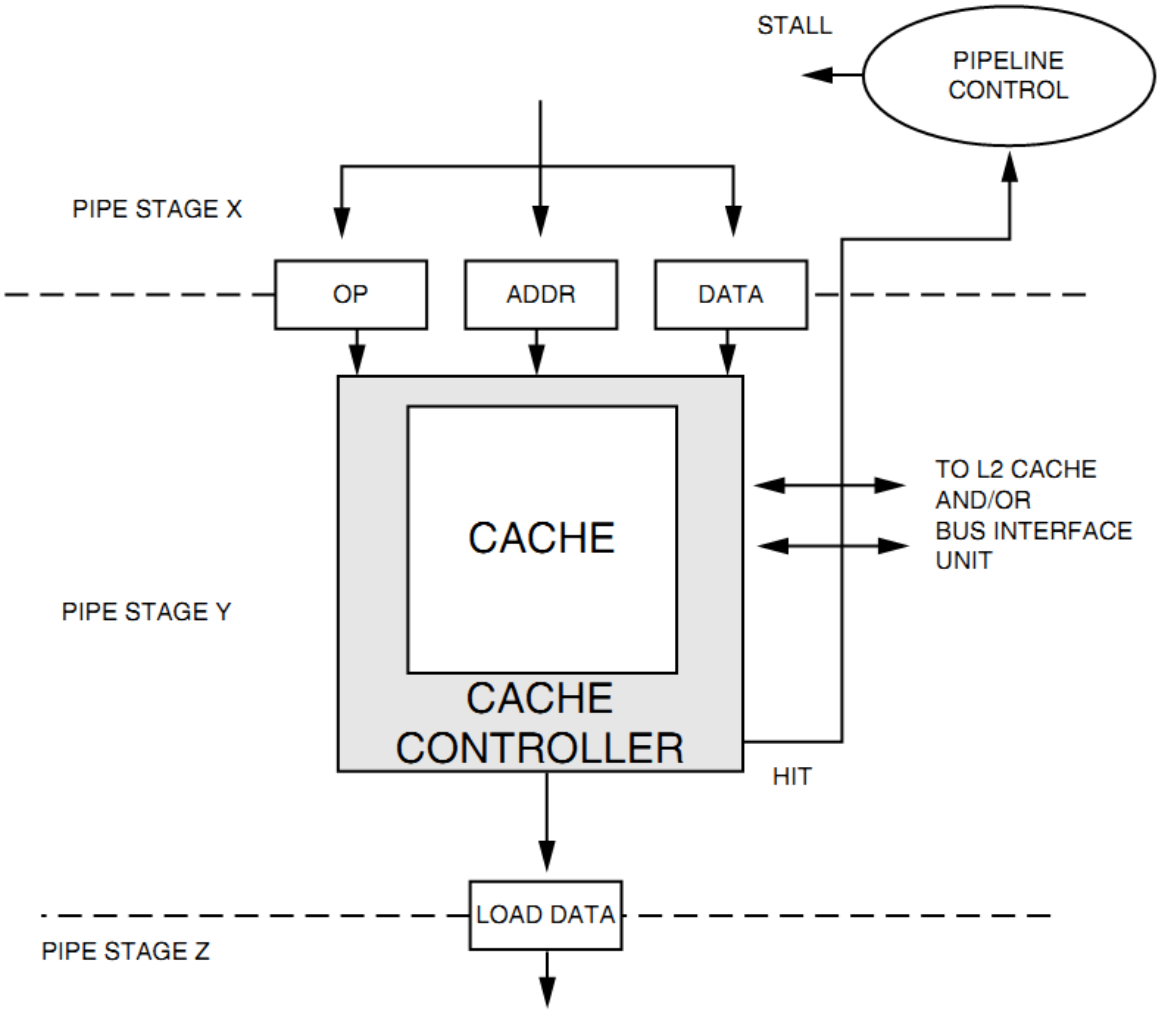
A Simple Cache：概要设计之接口

- CPU <-> Cache（Blocking cache）
  - 1bit *Read* or *Write* signal
  - 1bit *Valid* signal, a cache operation? ——是否访问Cache?
  - 32bit address
  - 32bit data from processor to cache
  - 32bit data from cache to processor
  - 1bit *Ready* signal, cache operation is complete
- Cache <->MEM
  - 1bit *Read* or *Write* signal
  - 1bit *Valid* signal, a memory operation?
  - 32bit address
  - 128bit data（块） from cache to memory
  - 128bit data from memory to cache
  - 1bit *Ready* signal, memory operation is complete
- CPU<->MEM：CPU不直接访问MEM!
  - 命中写回，不命中写分配



Pipeline and L1 Cache Interface



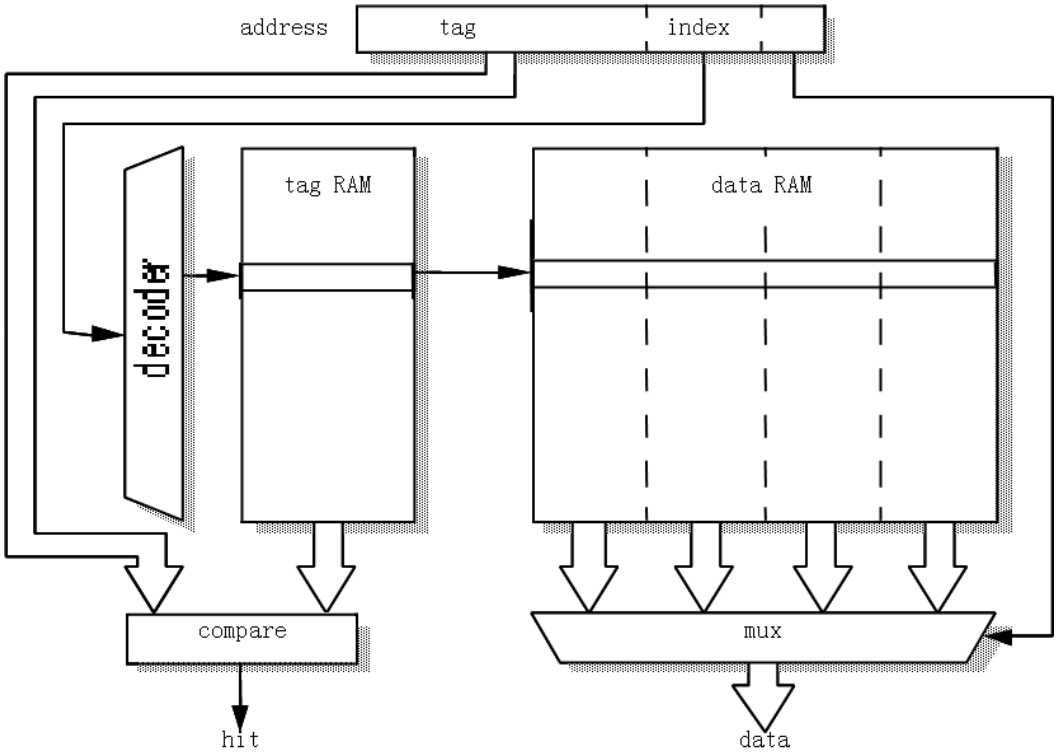


阻塞式blocking Cache，ld/st指令的执行周期是多少？

Nonblocking（$5.13）

Bruce L. Jacob, et al. Memory Systems: Cache, DRAM, Disk[M]. Morgan Kaufmann Publishers Inc. 2008.

Cache总体设计：数据通路 + 控制器



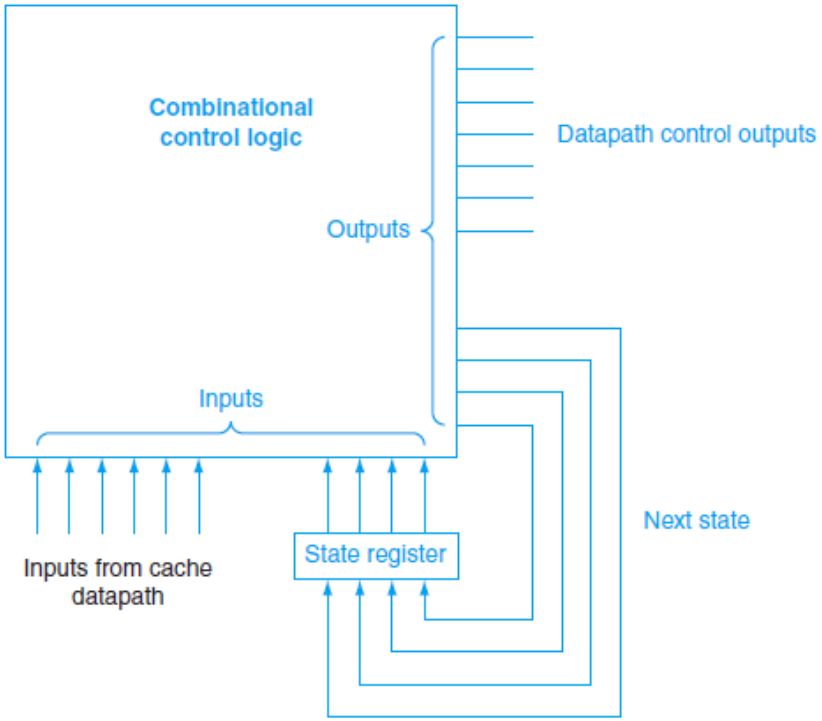
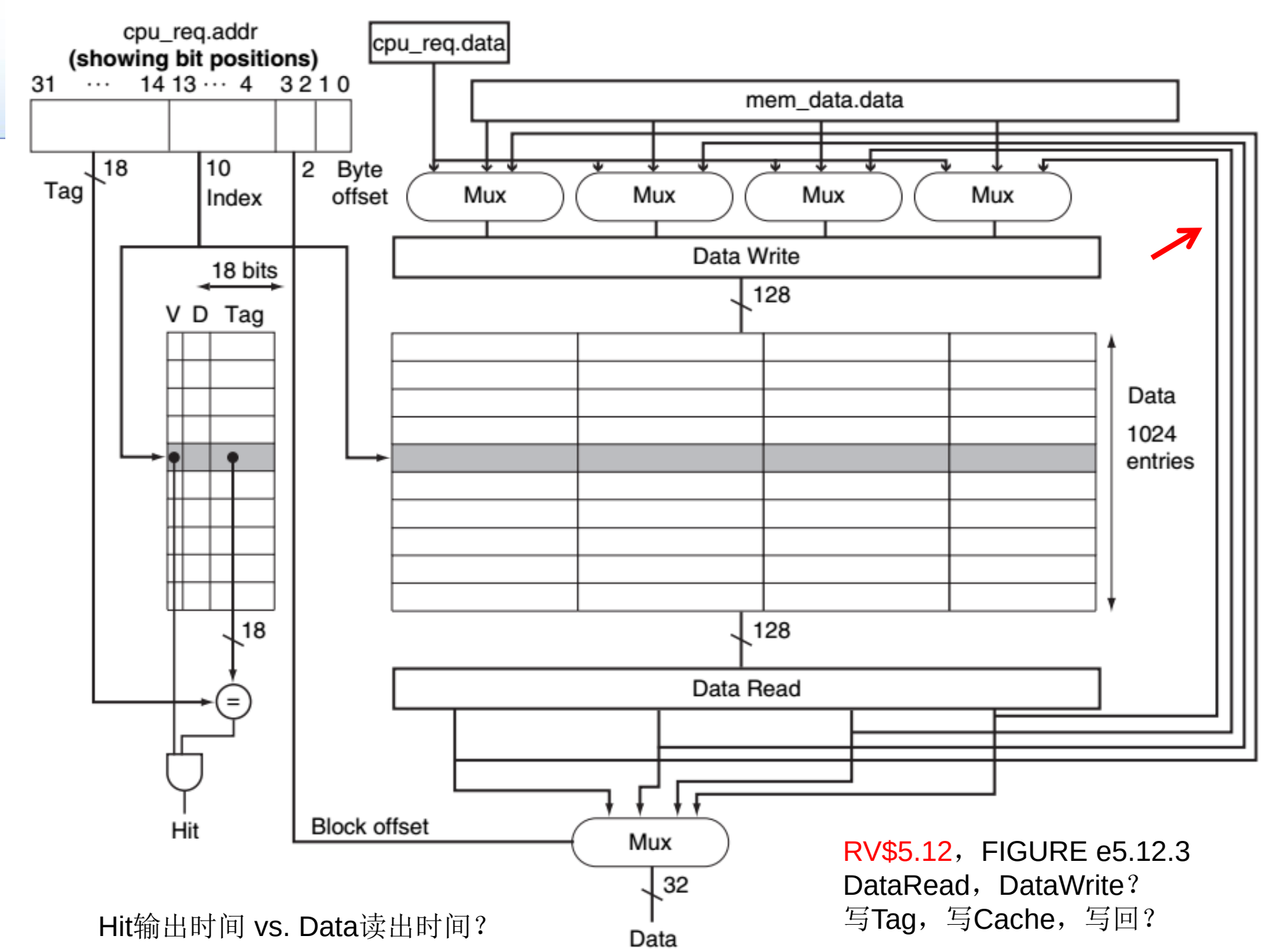


FIGURE 5.38

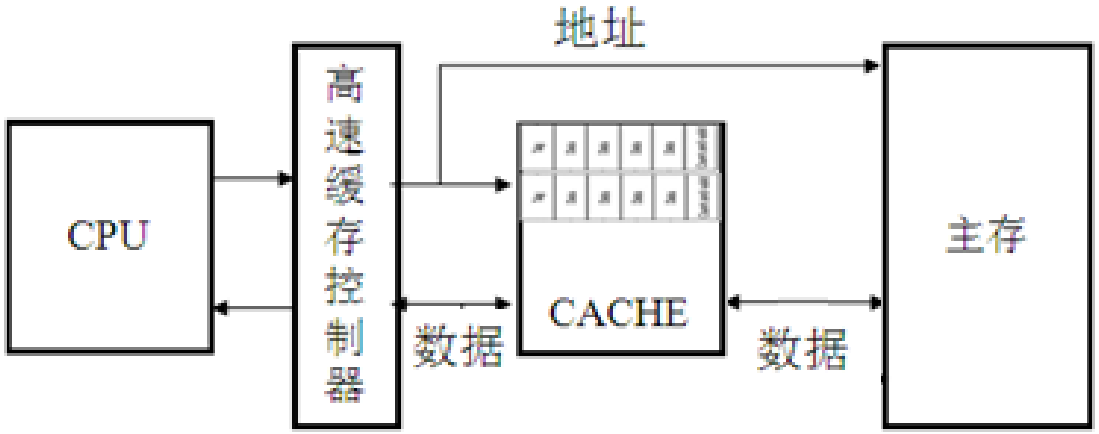


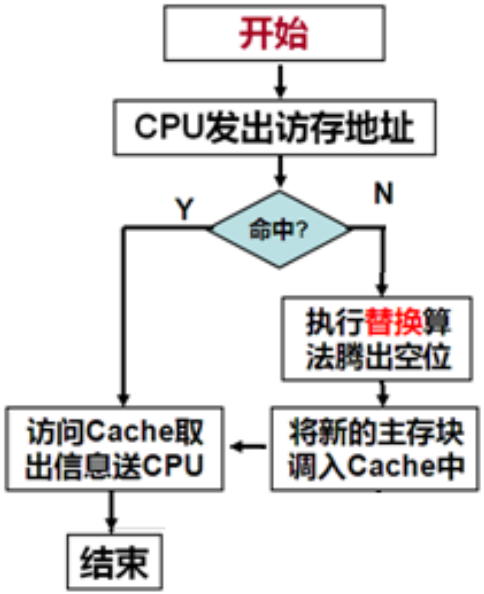
Hit输出时间 vs. Data读出时间？

RV$5.12，FIGURE e5.12.3
DataRead，DataWrite？
写Tag，写Cache，写回？

直接映射控制器设计

- Cache line位置唯一：读写miss时都要替换（脏要写回）
- 时序：半同步（CPU-Cache，Cache-MEM）
  - 阻塞式Cache：miss时指令stall
  - CPU不直接访问MEM
  - 读
    - 命中：直接读入，一个cc（判断hit，数据传输）
    - 不命中：换入（Clean直接换入，dirty写回），重读，多个cc
  - 写：Writeback using write allocate
    - 命中：写回（写cache，置dirty位），一个cc
    - 不命中：写分配，换入（Clean直接换入，dirty写回），重写Cache，多个cc





a Direct mapped Cache Controller

- 多周期
- CompTag
  - 命中：完成读写cache
  - Miss： “先换入，再读写”
    - 非脏：Allocate（写分配），多cc
    - 脏：Write-Back（先写回，再写分配），多cc
- Cache Controller<-> MEM controller
  - MEM Ready：换入换出完成握手，半同步

- 与CPU同步：IF或MEM阻塞
  - 周期数=?
- 可优化?
  - 拆分CompTag状态
  - 增加WriteBuf，存脏块
  - Idle必须?
  - 流水化?
- 写缓冲？——习题5.24
- 写透，写不分配，组相联，替换策略?

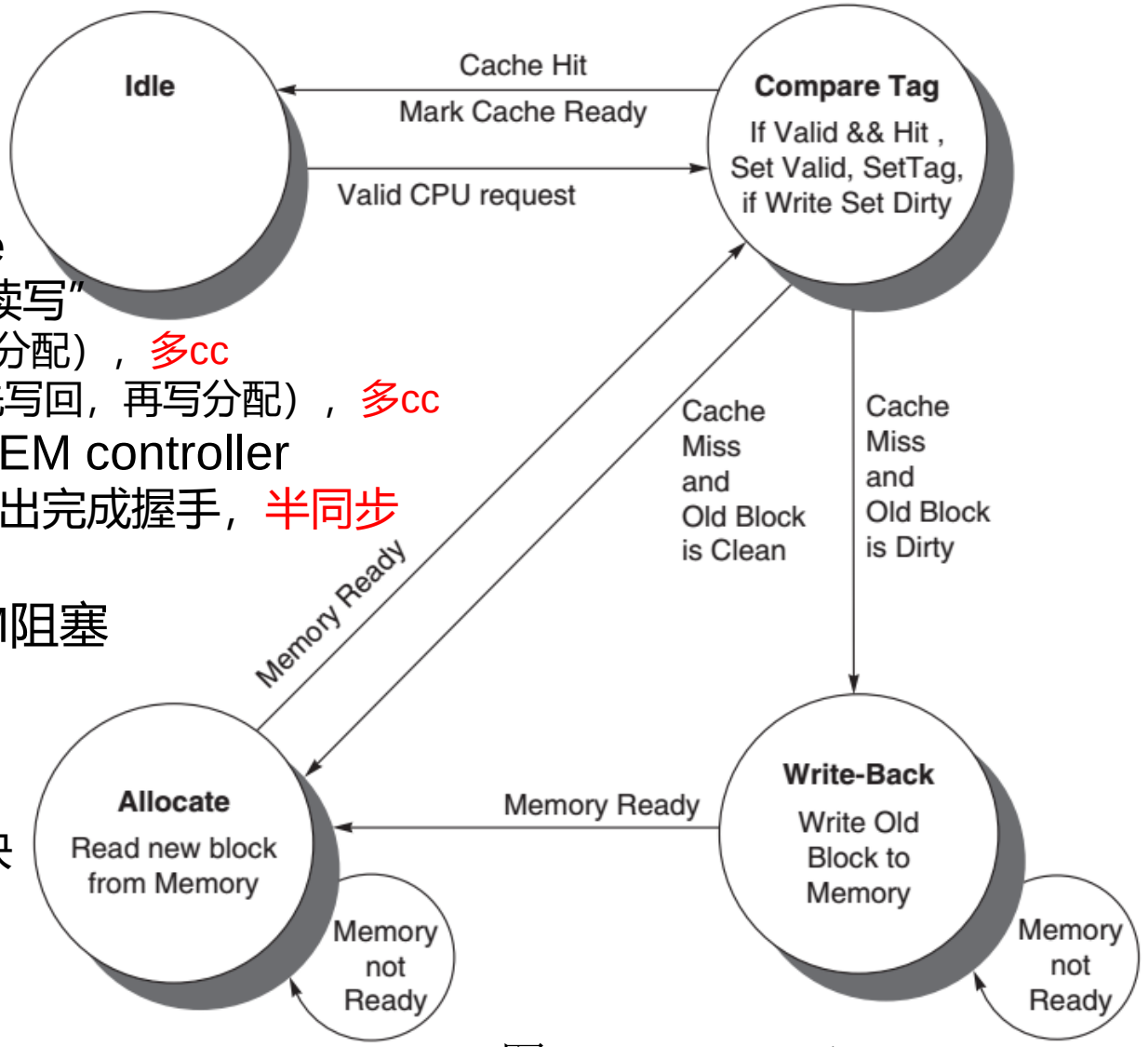
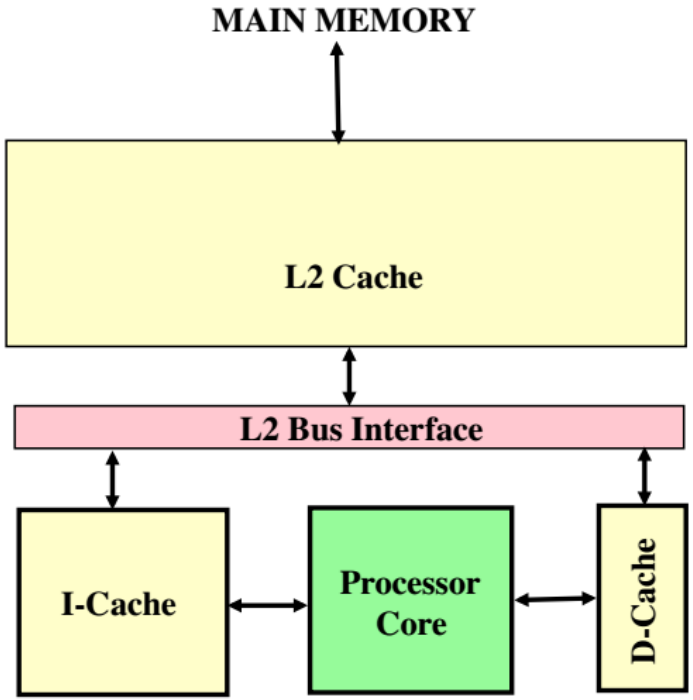
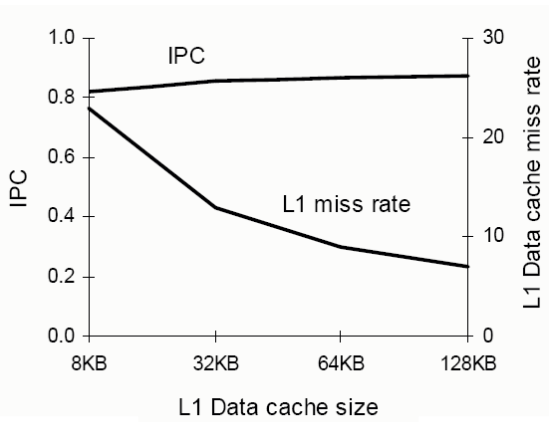


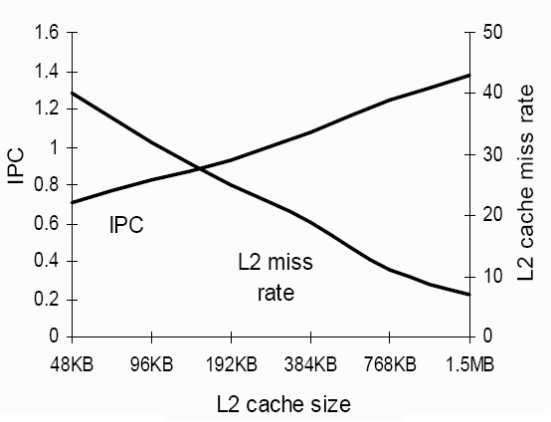
图5-39，Moore机？

多级Cache：$5.4.4

- 两层存储结构的存储访问时间
  - H为Cache命中率，T1和T2分别为两层存储器的访问时间，则系统访问时间Ts
    - Ts = T1 × H + （1 - H）×（T1 + T2）



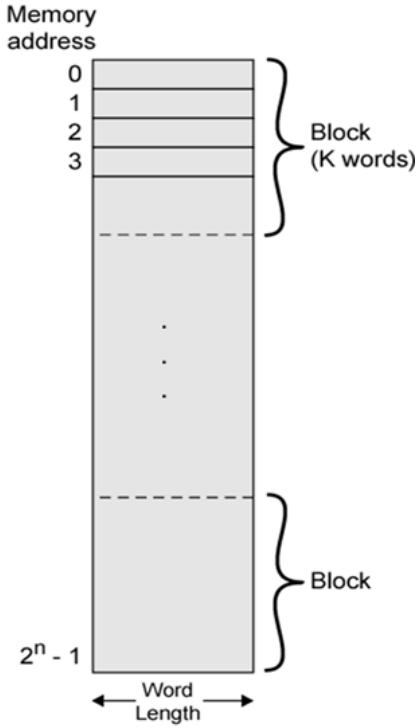


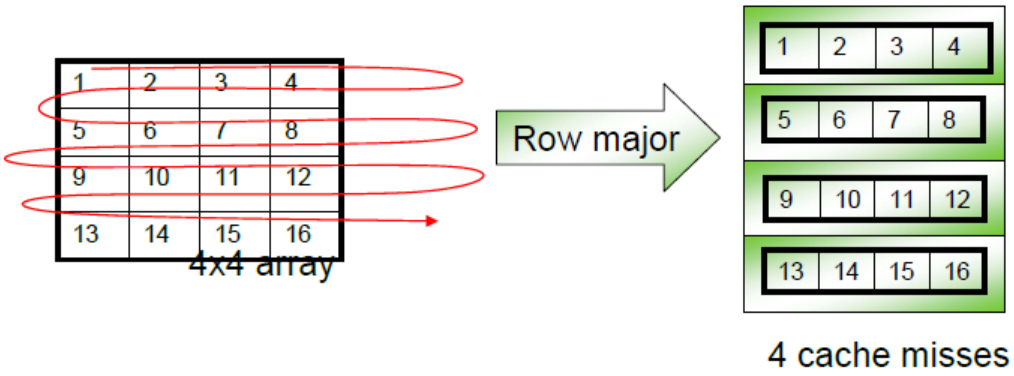


L1：速度，容量几乎对IPC没有影响？
L2：命中率，影响很大

Cache Effects

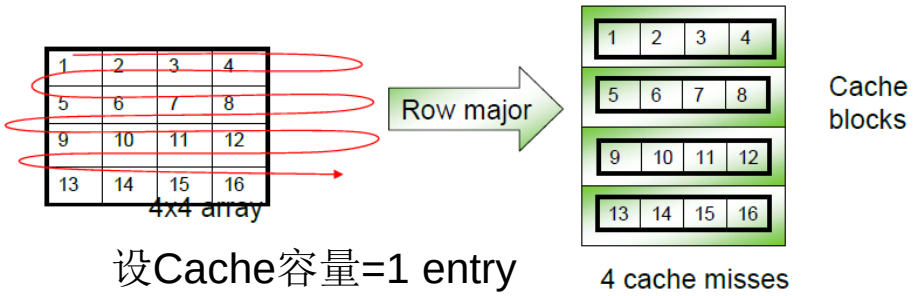
- Cache affinity：冷热、预热
  - 尽可能多的对已读取的数据进行操作，最大限度的发挥时间局部性；
- 注意循环：大部分计算和访存都发生在这里，$5.4.5
  - 按照数据对象（数组）在存储器中的存放顺序读取数据，从而最大限度的发挥空间局部性；
- Suppose：Cache size = one line
  - storing multidimensional arrays in linear memory
  - a program accesses the array one row at a time.
  - row-major order？
  - column-major order？
    - That would result in 16 cache misses





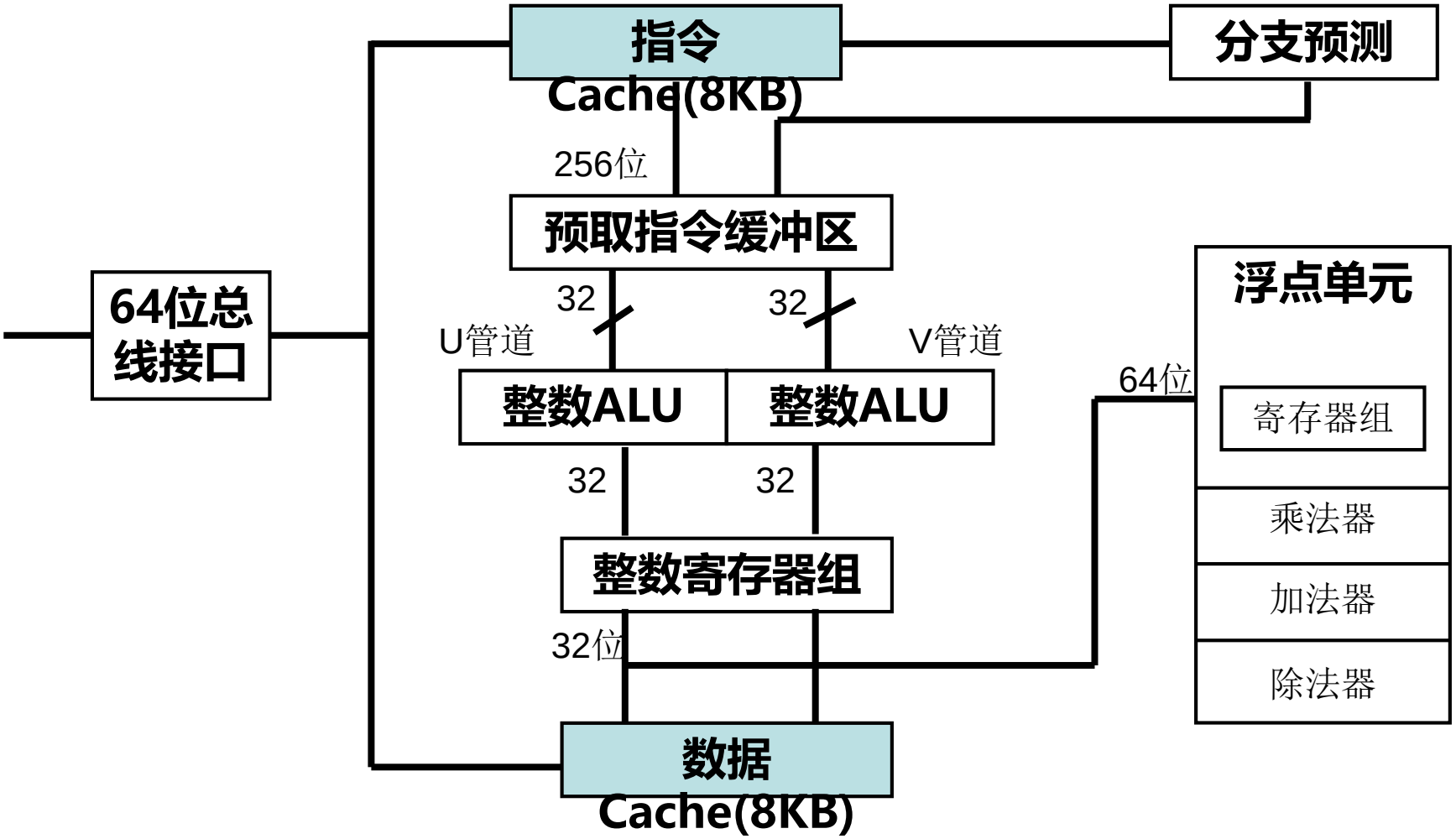
基于Cache的循环代码优化

- 循环交换（Loop Interchange）
  - 原程序(column major)
    - a[100][5000]=...//初始化
      for(j=0; j<5000; j=j+1) {
        for(i=0; i<100; i=i+1) {
          a[i][j] = 2 * a[i][j]; 每次都不命中
        }
      }
  - 改进 (row major)
    - a[100][5000]=...//初始化
      for(i=0; i<100; i=i+1) {
        for(j=0; j<5000; j=j+1) {
          a[i][j] = 2 * a[i][j]; 可连续命中若干次[cache行大小]
        }
      }



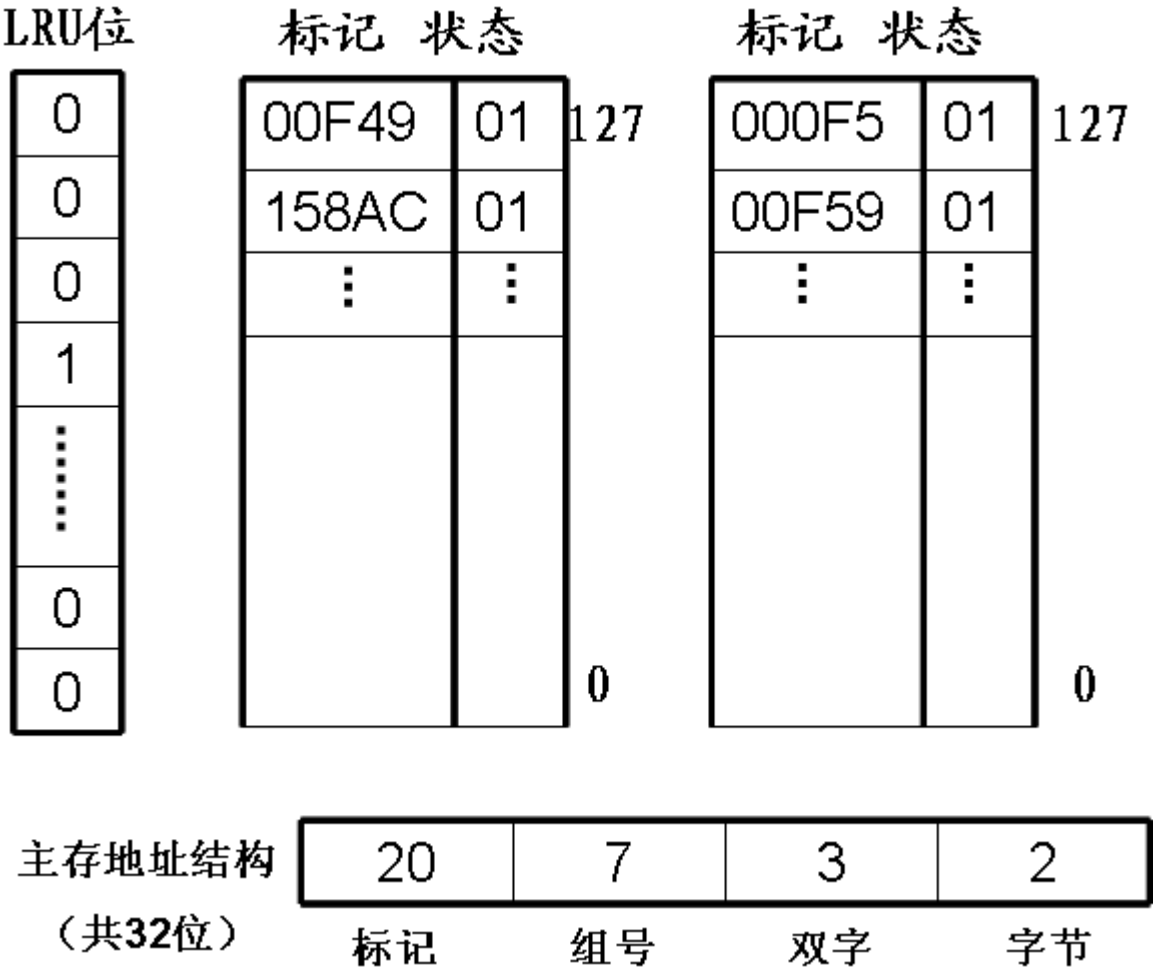
- 循环合并（Loop fusion）：多个循环体并入一个基本块
- 循环分块（Blocking）：DGEMM算法（ $5.4.5，$5.15）
- Cache-oblivious algorithm：与cache结构无关的算法

Pentium处理器框图



Pentium处理器的片内DCache

- 两路组相连: 共128组，每路8个双字（4X8＝32B）。
- 采用“写回”策略，可动态重构支持“写透”
- LRU：组计数，路随机
- 两条专用指令：清除或回写Cache



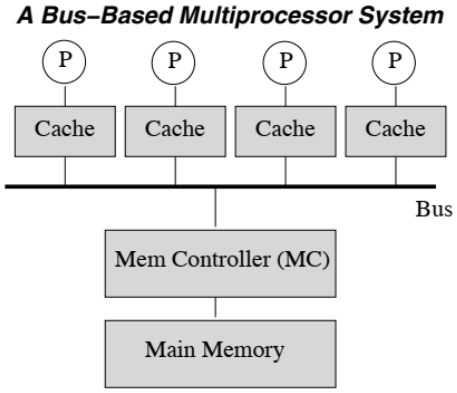
- **状态位**：4种状态，在Cache一致性协议（MESI）中使用。

Cache coherence，$5.10

- 程序员的“一致性”
  - 按程序序，对任一变量的读操作都能得到其最新写入值
- 多处理器系统
  - Cache coherence：读操作返回什么值（是否最终结果？）
    - write serialization：两个CPU对同一位置的两次写操作在所有CPU看来顺序相同
    - *Write invalidate protocol：基于监听snooping*
  - Memory consistency：写操作结果何时可见
    - 对多个位置的写操作顺序保持一致。
    - llxx：一旦看到新写，意味着所有之前的写操作都已完成。

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

FIGURE 5.40



Cache一致性实现

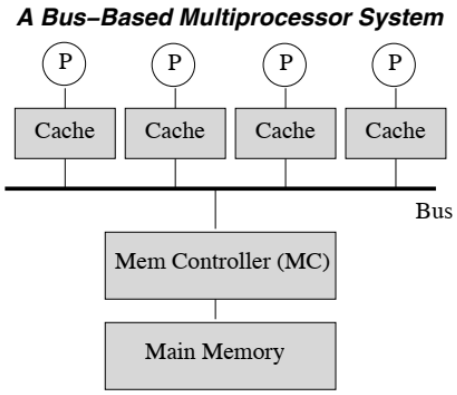
- Write invalidate protocol：总线监听协议
  - 写入时使其他Cache的副本无效
    - 各Cache监听总线上的写操作。如果本地Cache中有相同的目的地址，则使此数据块无效

FIGURE 5.41

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

FIGURE 5.40

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

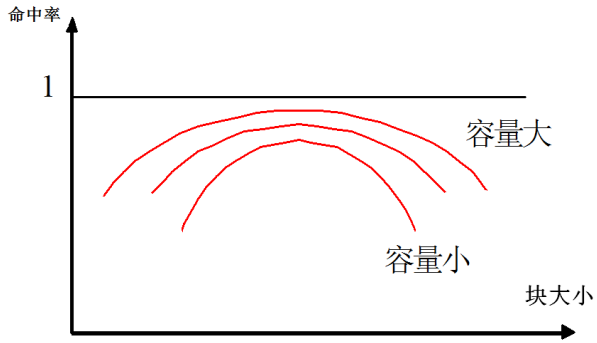


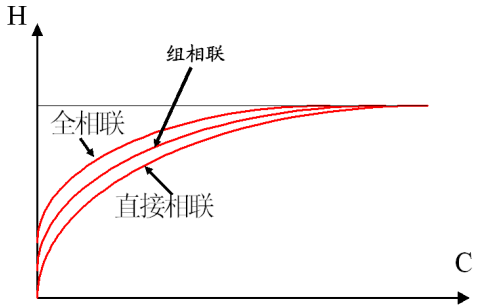
伪共享false sharing，$5.11

- 处理器各自处理不同的变量，但它们却在同一Cache line中，导致不必要的一致性维护开销
- 软硬件接口：程序员或编译器避免此问题

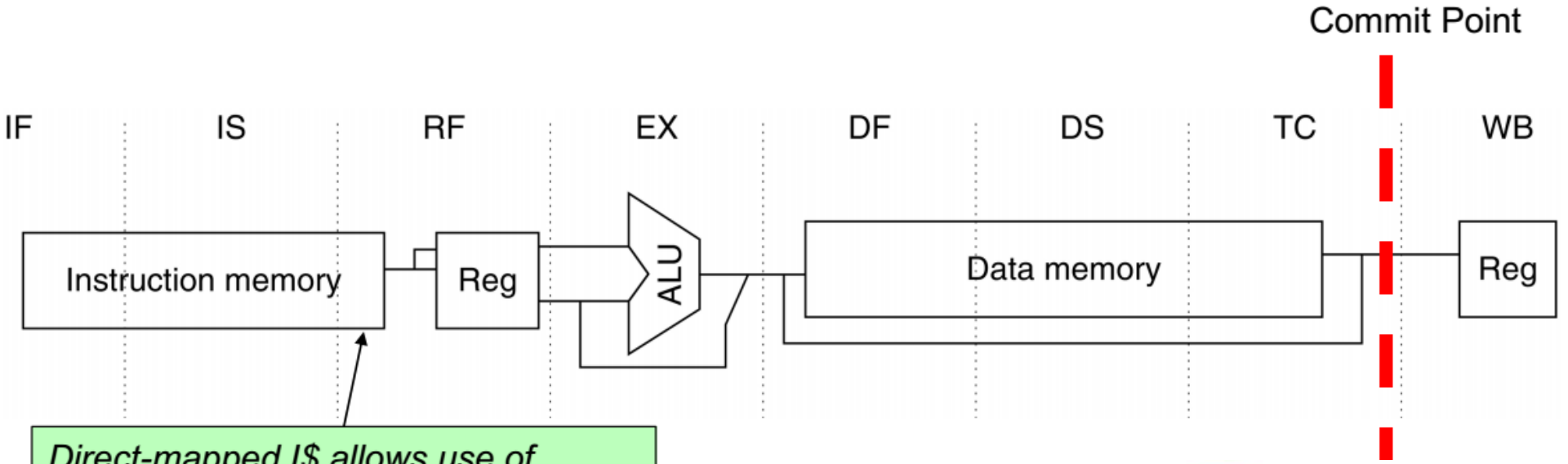
小结：4种Cache Misses，（$5.8.5 “3C”）

- Compulsory（必然）
  - cold start、first reference、process migration
  - “Cold” fact of life: not a whole lot you can do about it
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
- Capacity：cache大小，*block大小*
  - Cache cannot contain all blocks
  - Solution: increase cache size
- Conflict (collision)：相联度
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- *Coherence (Invalidation)：MP/MC*
  - *other process (e.g., I/O) updates memory*

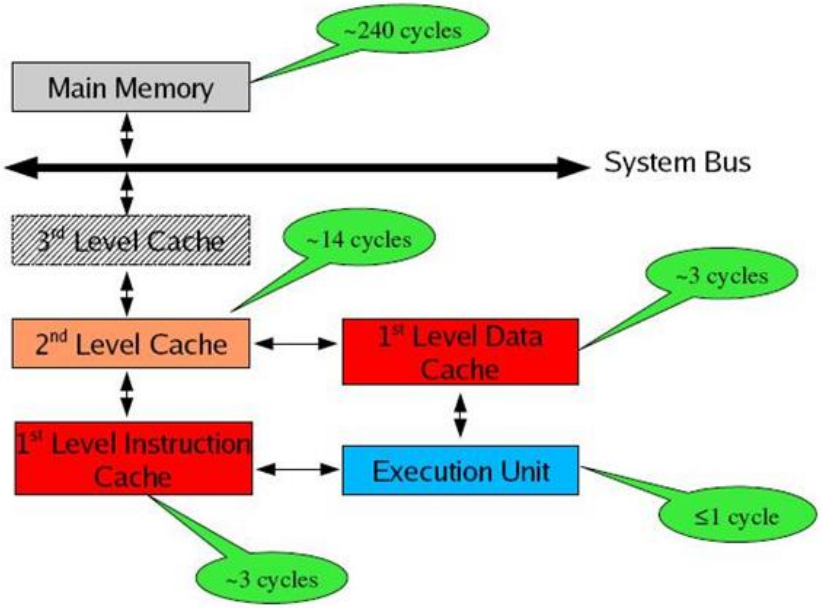




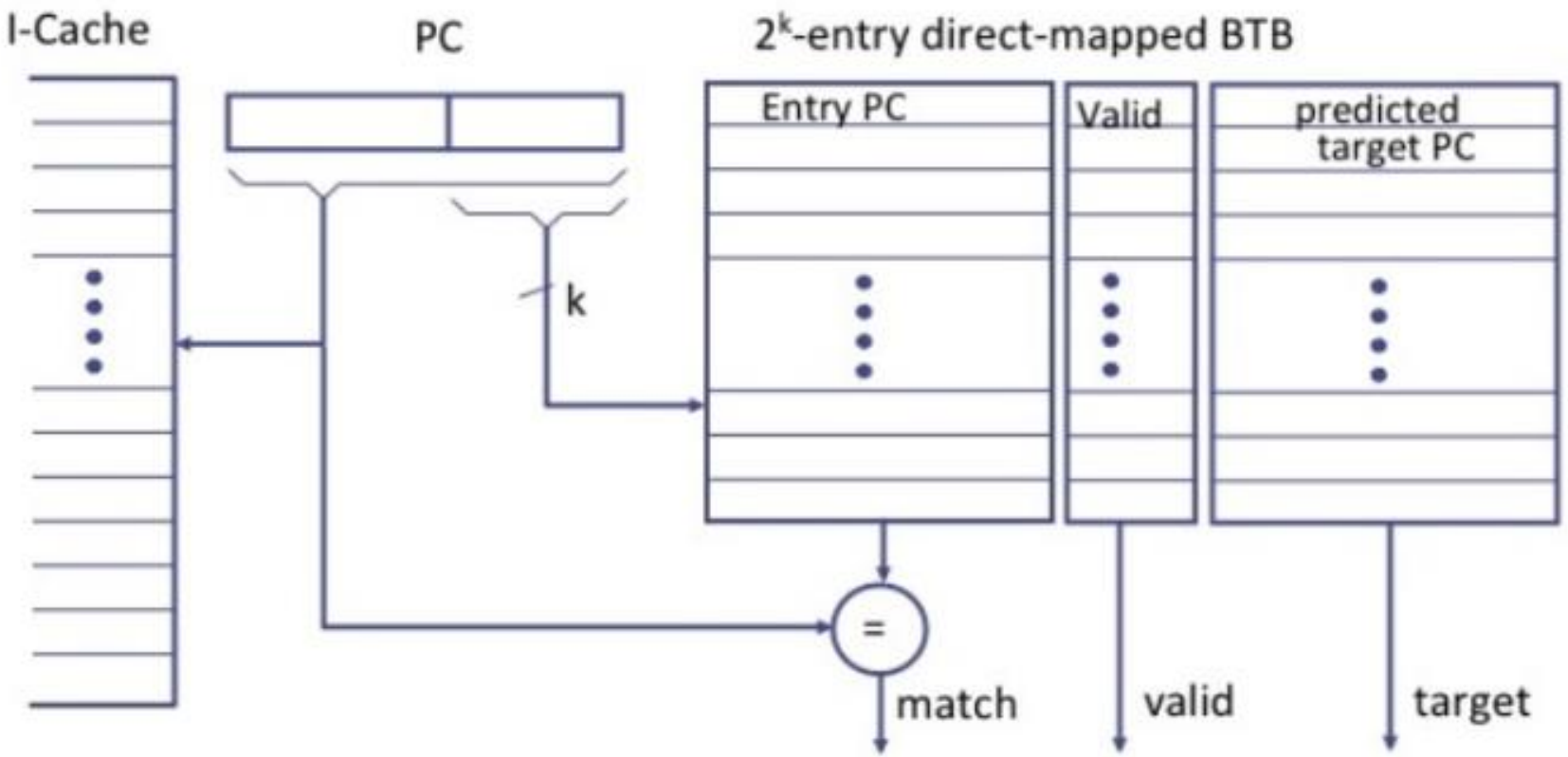
Deeper Pipelines: MIPS R4000



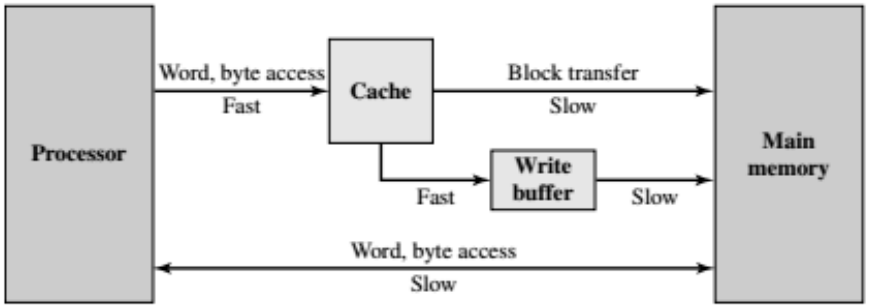
- Longer pipeline results in
  - Decreased cycle time
  - Increased load-use delay latency
  - Increased branch resolution latency
  - More bypass paths
- 两个Cache的写策略?
- EU 1 cycle，L1$ ~ 3 cycles

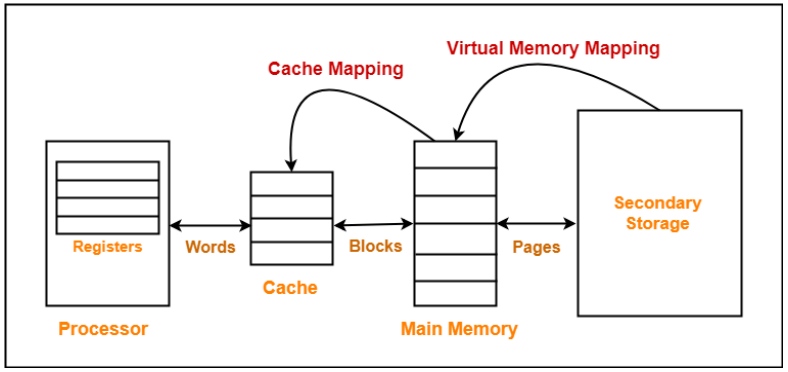


直接映射Cache与分支预测缓冲



小结





- Cache有效性
  - Cache效率和局部性的度量指标?
  - Cache miss的原因?
  - Cache的Side effect
    - 一致性：Cache与主存内容。
      - 单CPU，多CPU，DMA
    - 实时性：访存时间的确定性
    - 读写放大问题，伪共享
  - 发挥Cache的作用：利用局部性
- Cache组织结构，读写过程，映射机制，替换策略
  - 为何需要不同的映射模式?
    - 全相连：Tag=块号，选路，全部比较（C个比较器），可使用CAM
    - 直接映射：Tag=段号，按line数分段，定位到line，一个比较器
    - N-way set: Tag=段号，按组数分段，定位到组，选路，N个比较器
  - 三种映射方式各自需要哪种置换算法?
  - 编程实现LRU?
- 作业：5.5，5.11，5.24


Thank You