



计算机组成原理

第5章 层次化存储

概述, Cache

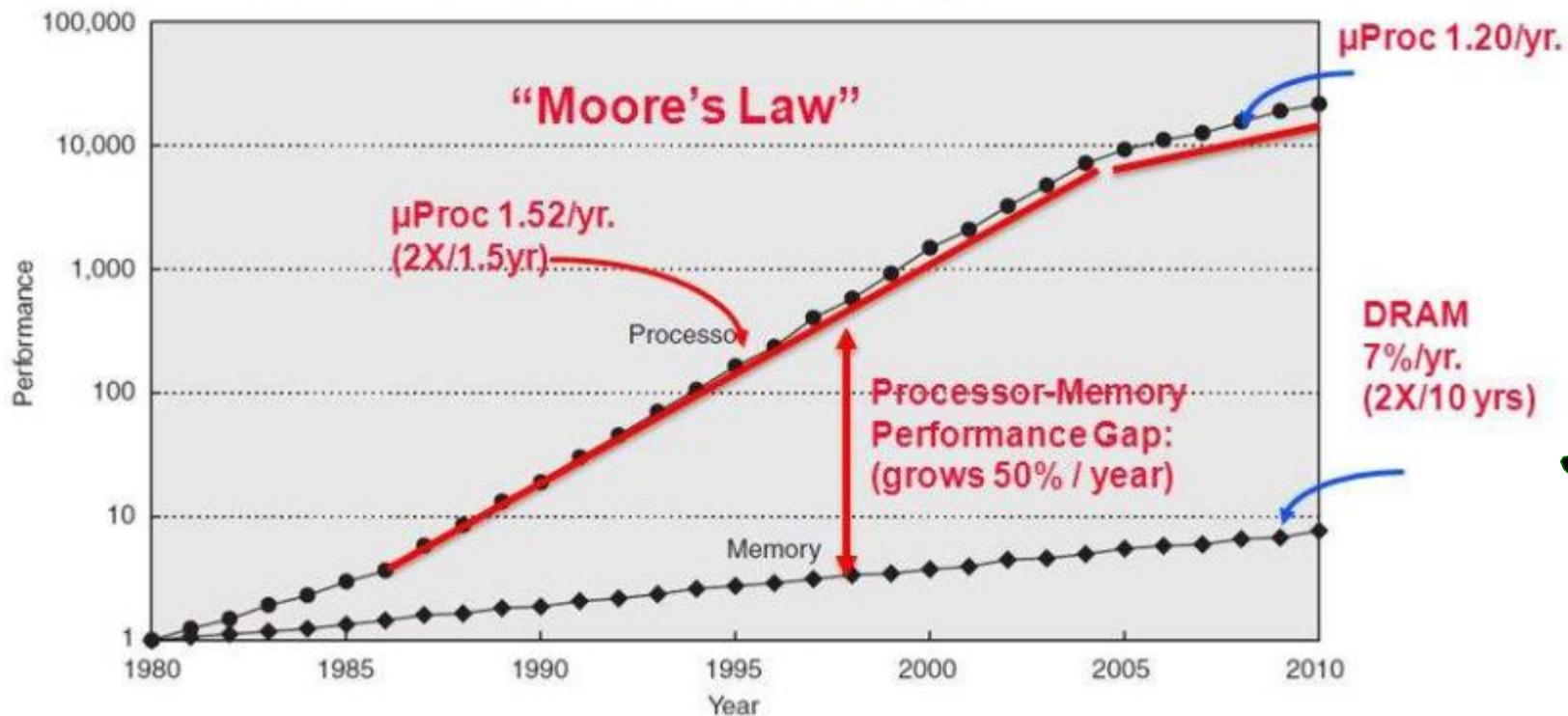
llxx@ustc.edu.cn



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”

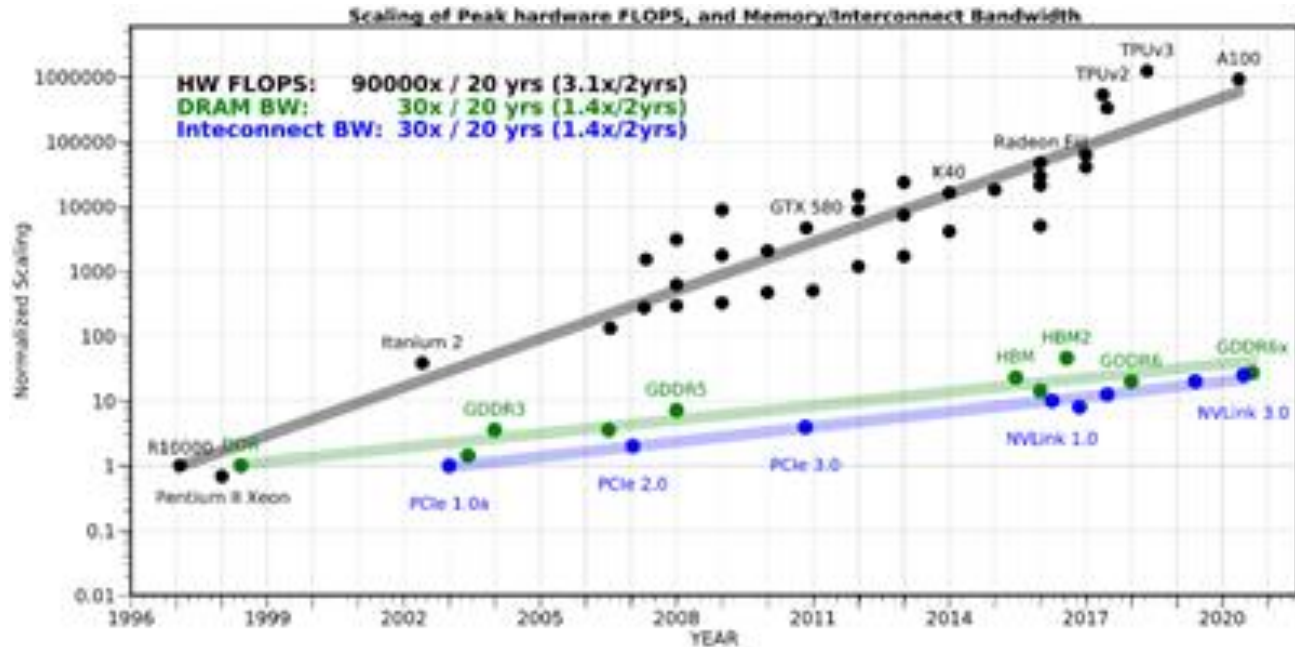
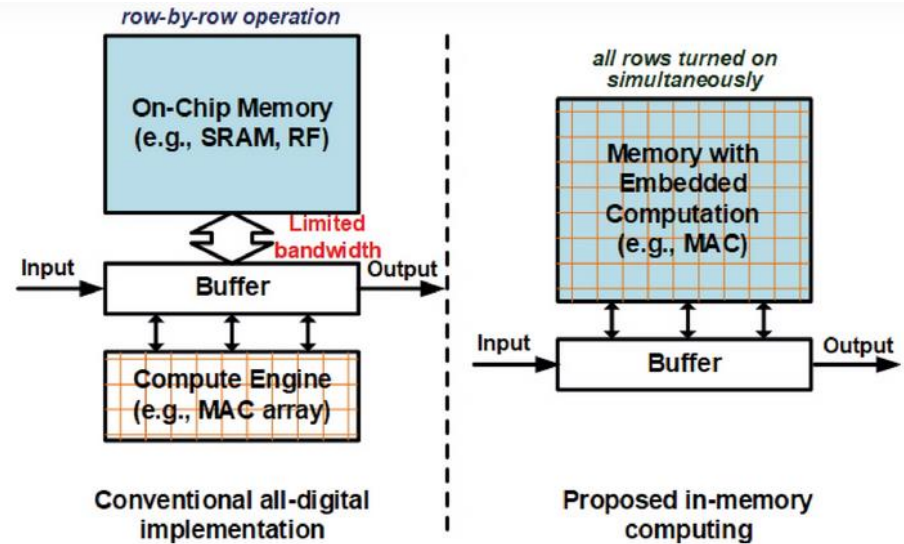
Processor-DRAM Memory Gap





Memory Wall: 1995, Wulf@Univ of Virginia

- **存算一体架构** (in-memory Computing), 2010年
 - 在存储器中嵌入计算能力, 以新的运算架构进行二维和三维矩阵乘法/加法运算。



层次化存储：性能、容量、功耗、价格

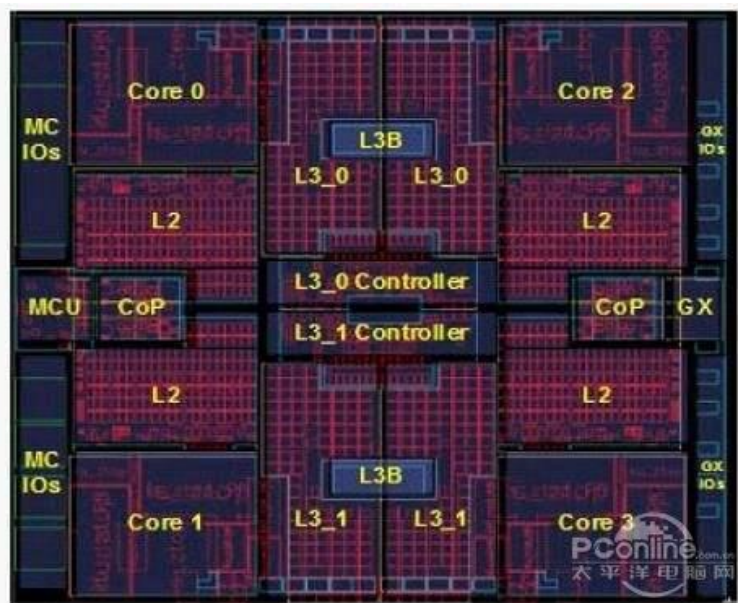
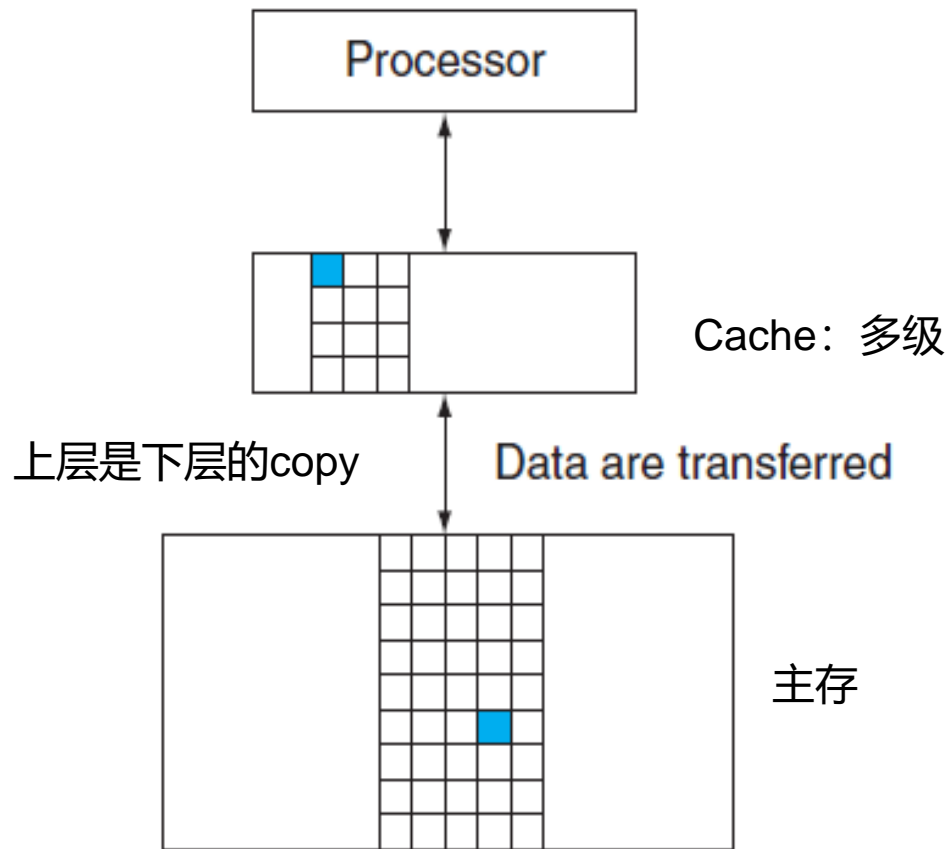
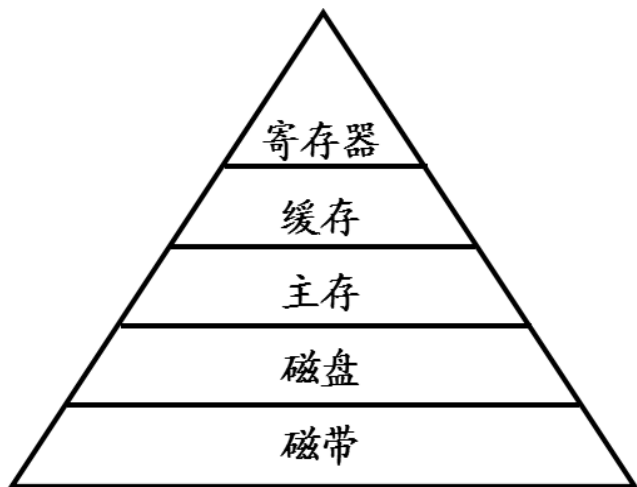
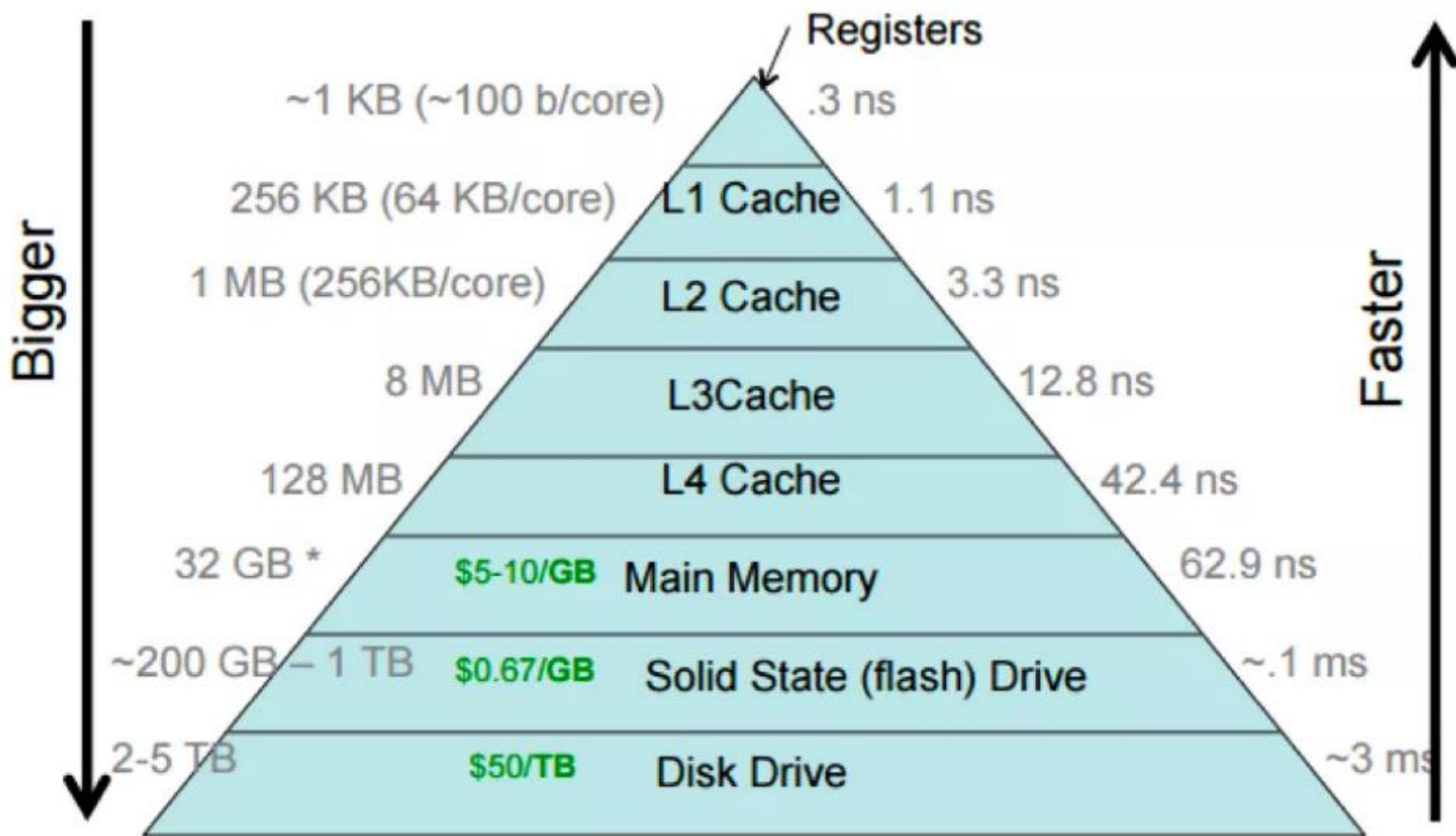


图5-2，层次化存储系统

“Cache能耗大约占整个处理器能耗的一半”

层次化存储系统



Cache对处理器性能的影响: CPI=1

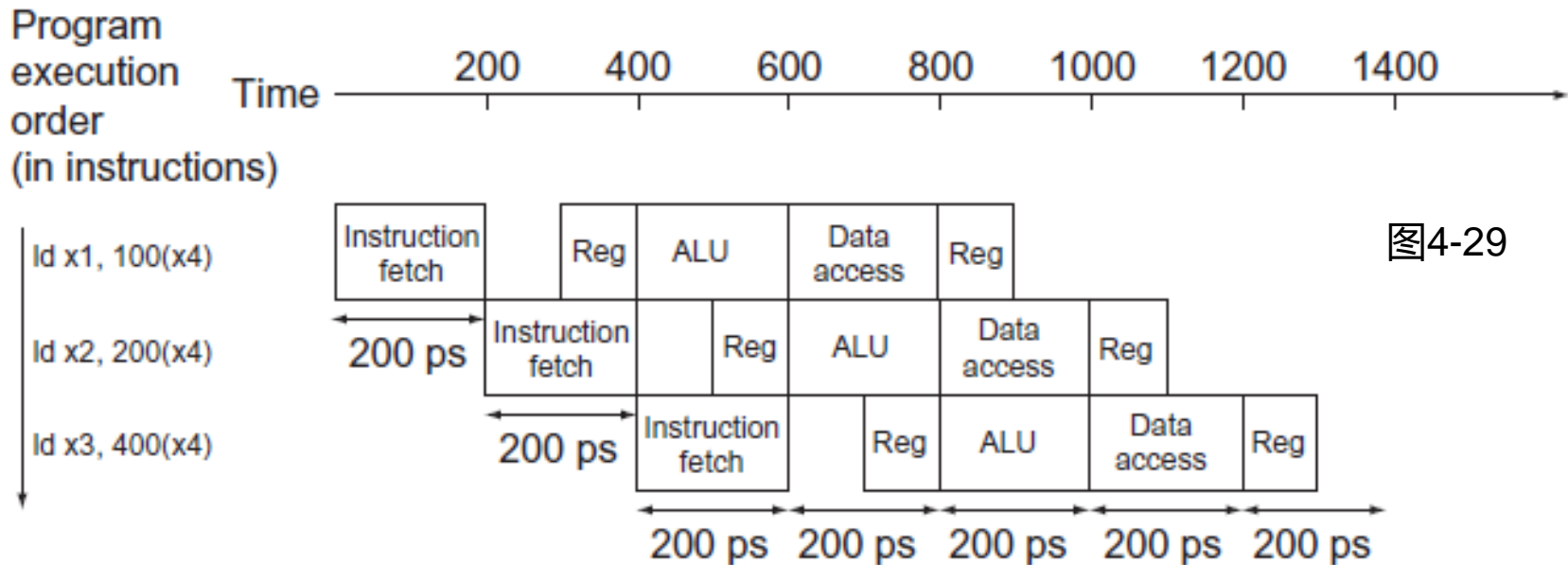
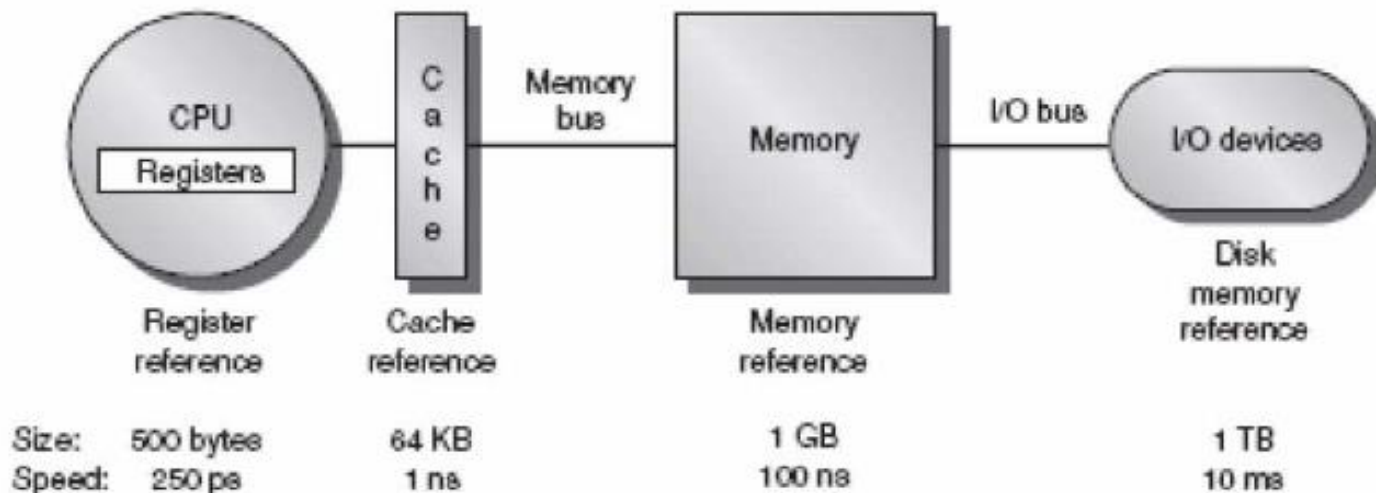
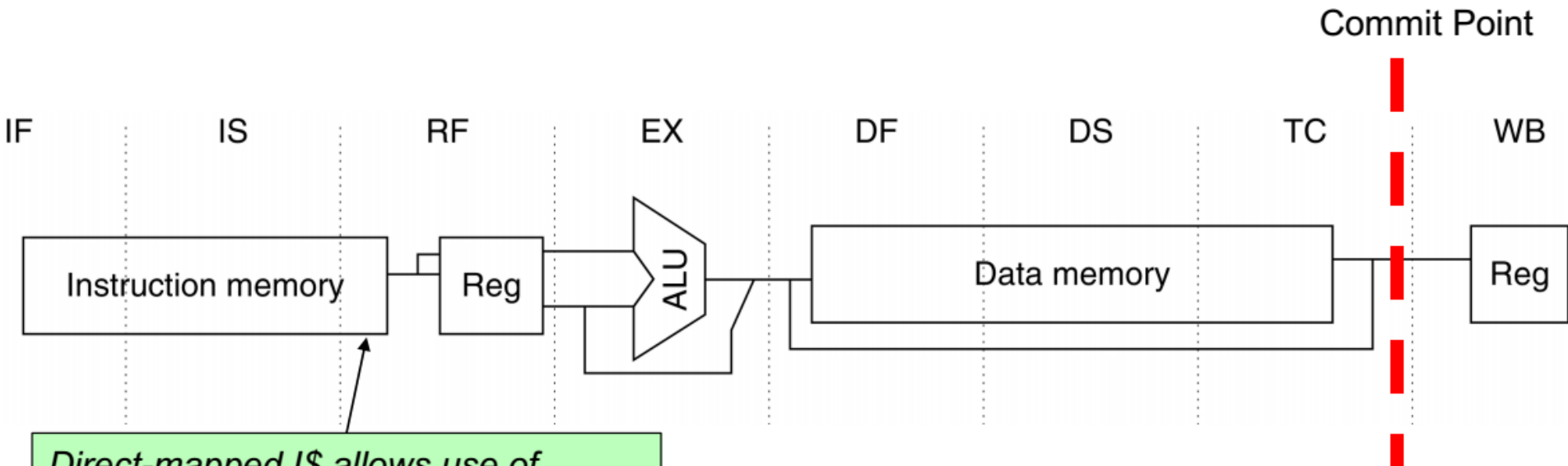


图4-29

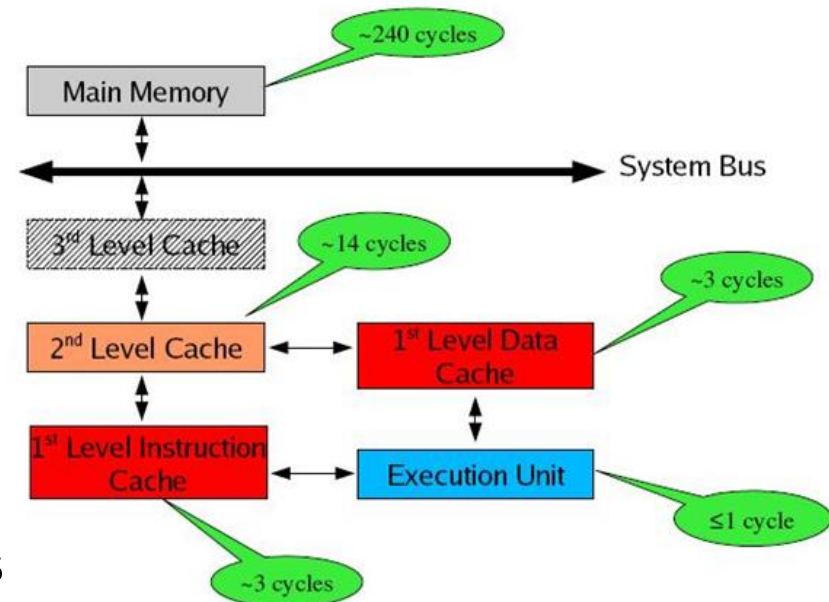


Deeper Pipelines: MIPS R4000



Direct-mapped I\$ allows use of instruction before tag check complete

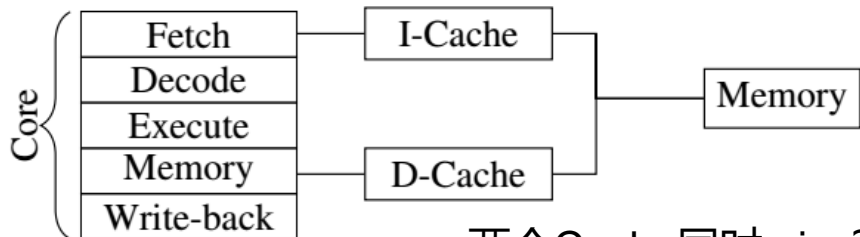
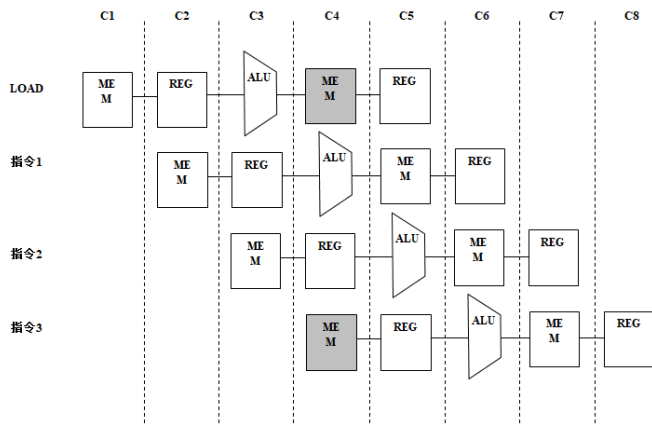
- Longer pipeline results in
 - Decreased cycle time
 - Increased load-use delay latency
 - Increased branch resolution latency
 - More bypass paths
- EU 1 cycle, L1\$ 命中时间 ~3 cycles



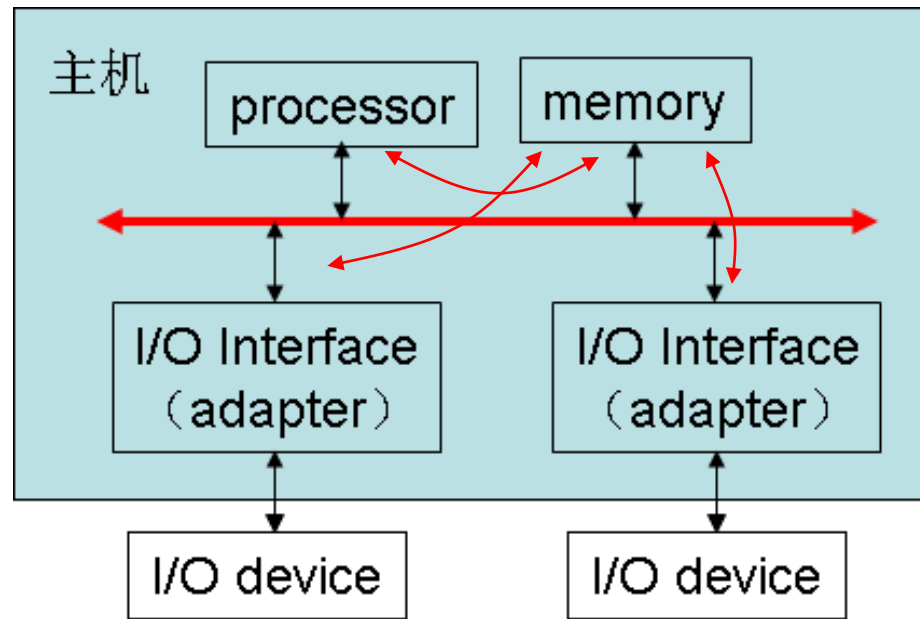


Cache对系统结构的影响

- 结构冒险
 - 流水线存储器冲突：IF取指与MM数据读写，分立Cache (split Cache)
 - 总线冲突：共享内存shared memory+共享总线
 - 取指与数据读写，CPU和I/O同时访问主存，竞争总线
 - Cache：位于处理器片上，减少CPU访问主存
- 副作用：一致性 (cache-mm/cache-cache)，时序可预测性



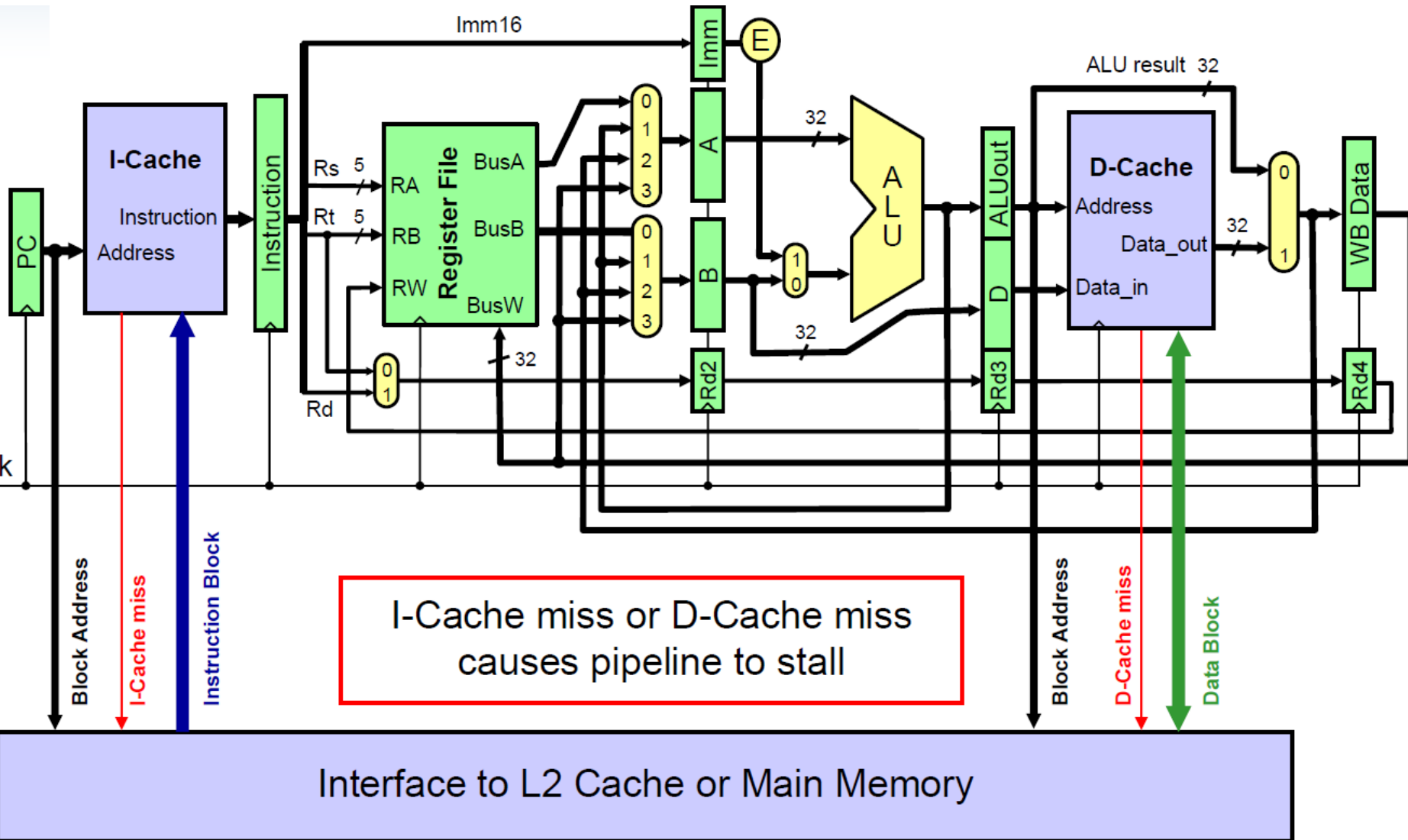
两个Cache同时miss?





时序可预测：stall多少个周期？

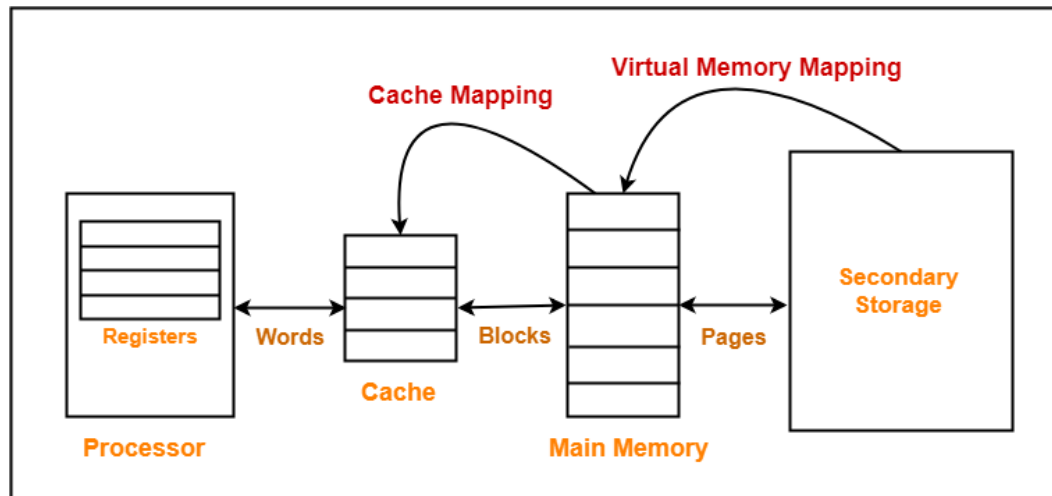
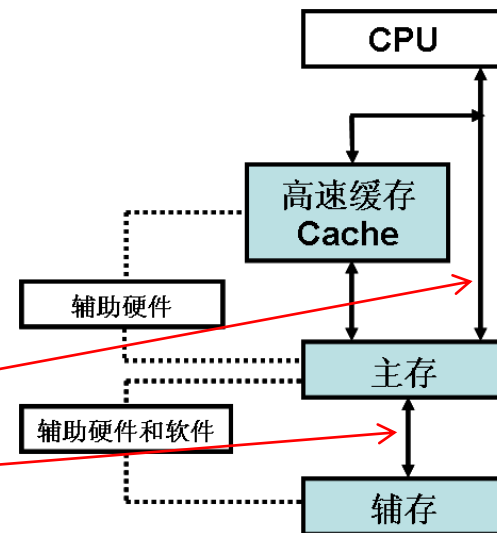
Cache miss: 阻塞式 (blocking) /非阻塞式, stall



“Cache - 主存”与“主存 - 辅存”层次的区别



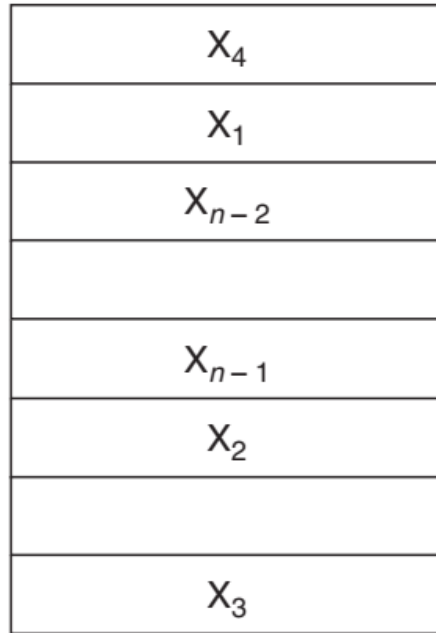
存储层次 比较项目	“Cache - 主存”层次	“主存 - 辅存”层次
目的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理实现	主要由专用硬件实现	由硬件和软件实现
访问速度的比值 (第一级和第二级)	几比一	几百比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的 访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程



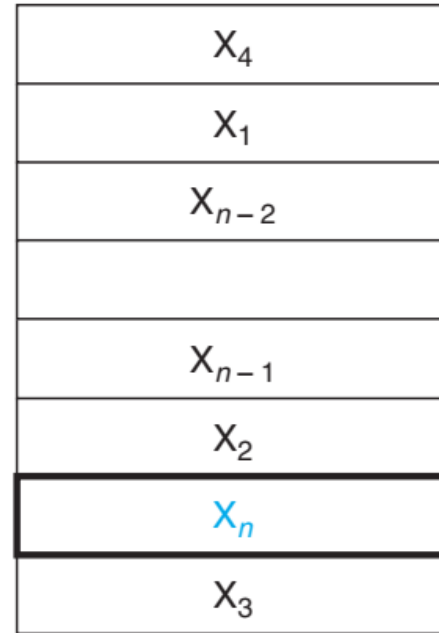


例：访问 X_n 前后Cache变化：miss -> hit

图5.7



a. Before the reference to X_n

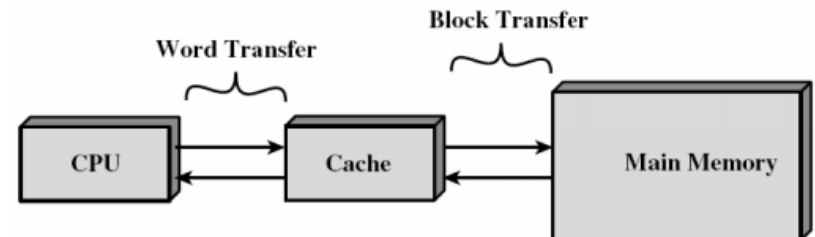


b. After the reference to X_n

- 设初始时, X_n 不在Cache中

• 关键问题

- 映射方式：数据块大小，放在哪儿？
- 如何判断是否命中？
- 替换策略：Cache满？
- 写策略：Cache与MM一致？



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 Coherence, 5.10
 - 详见体系结构课
- *Cache 性能分析*
 - *主要见体系结构课*
- 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, 5.10, 5.12**
- **唐: 4.3, 附录4A**

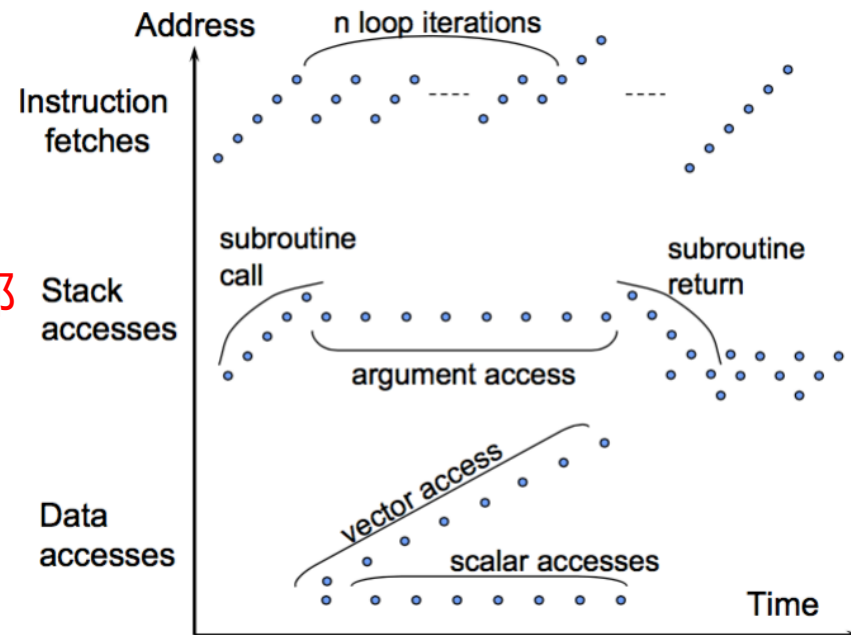
程序的访存特性：时空局部性原理， p259



- 冯诺依曼机：连续存放
- 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
 - 块大小、读写放大

Typical Access Address Pattern

空间局部性：地址范围有限



时间局部性：持续一段时间

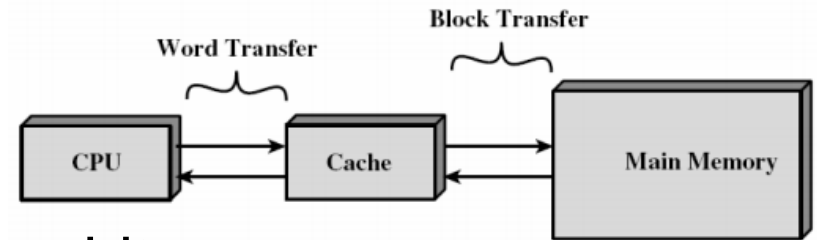
命中、不命中、命中率：\$5.1



- Cache命中 (hit)

- 欲访问的数据在缓存中

- 命中时间 hit time, Time for a hit



- 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

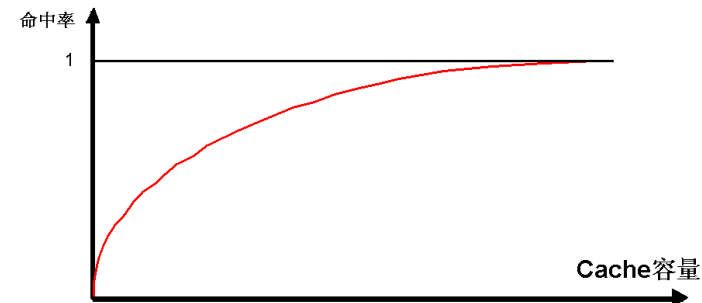
- CPU可阻塞 (blocking, stall) 或非阻塞 (non-blocking)

- 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)





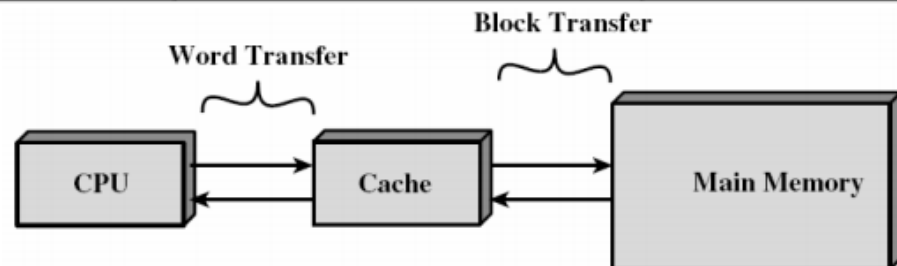
例：Cache基本结构参数

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

图5-33

Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250-2000	2500-25,000	16,000-250,000	40-1024
Total size in kilobytes	16-64	125-2000	1,000,000-1,000,000,000	0.25-16
Block size in bytes	16-64	64-128	4000-64,000	4-32
Miss penalty in clocks	10-25	100-1000	10,000,000-100,000,000	10-1000
Miss rates (global for L2)	2%-5%	0.1%-2%	0.00001%-0.0001%	0.01%-2%

- Cache容量：名义容量，实际容量
- Cache-line size match the **width** of the DRAM simplified the design.





内存分块 (block, 字块), 一块多字

Memory address	Data
0000 0000 0000 0000 00 00	█
0000 0000 0000 0000 00 01	█
0000 0000 0000 0000 00 10	█
0000 0000 0000 0000 00 11	█
0000 0000 0000 0000 01 00	█
0000 0000 0000 0000 01 01	█
0000 0000 0000 0000 01 10	█
0000 0000 0000 0000 01 11	█
0000 0000 0000 0000 10 00	█
0000 0000 0000 0000 10 01	█
0000 0000 0000 0000 10 10	█
0000 0000 0000 0000 10 11	█
⋮	⋮
1111 1111 1111 1111 11 00	█
1111 1111 1111 1111 11 01	█
1111 1111 1111 1111 11 10	█
1111 1111 1111 1111 11 11	█

字地址

Block identification

Block 0
4个words

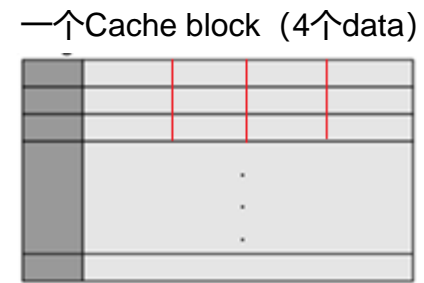
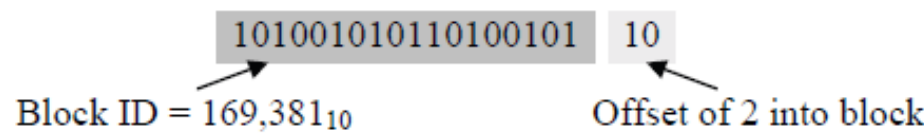
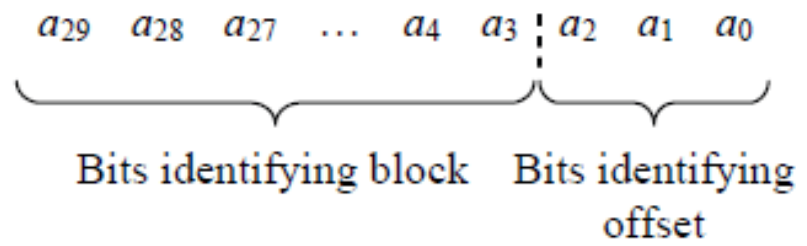
Block 1

Block 2

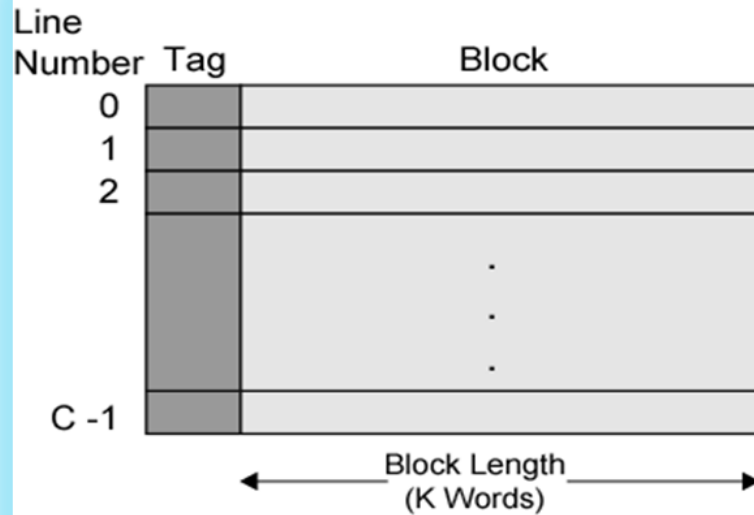
Each gray block (one data) represents an addressable memory location containing a word

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

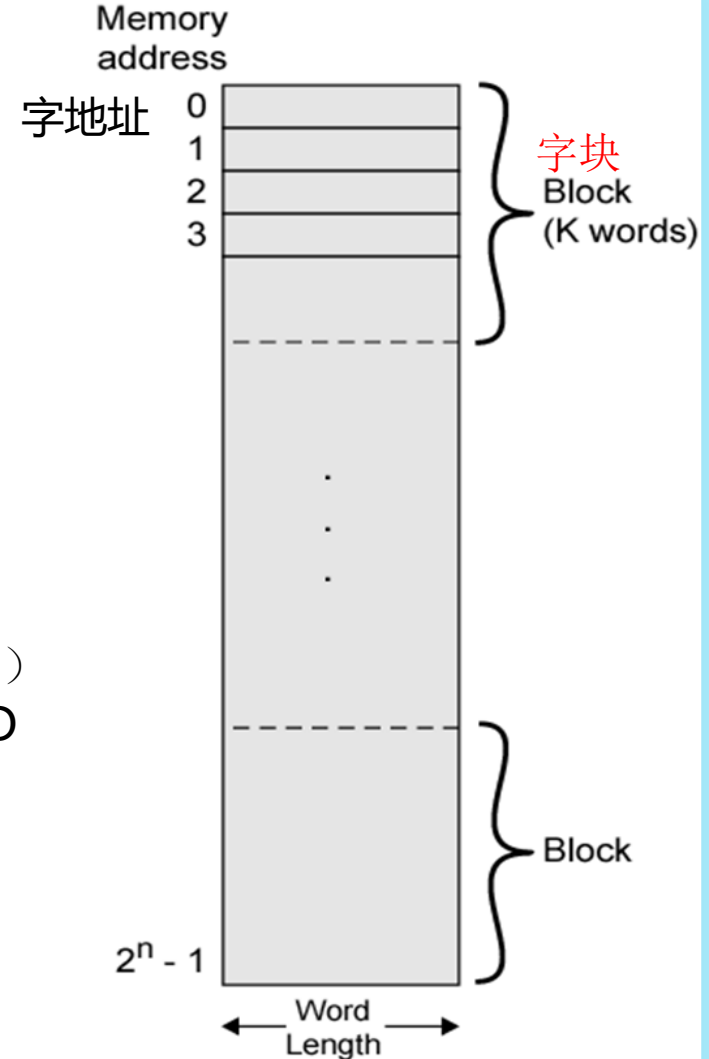
块大小多少合适?



Cache/Main Memory Structure: 行/字块



(a) Cache



(b) Main memory

内存和Cache分块（字块），一行一块（字块）
 Cache **line** size = Tag + Mem **Block** size + VCD

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行 = **way**路



Cache Line Structure

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

- 一行一块 (字块, data) , 一个data多个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

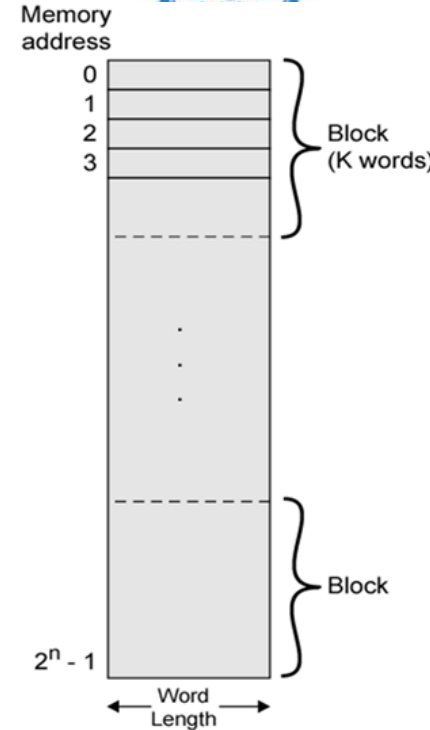
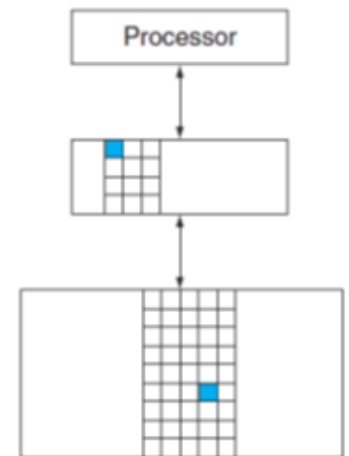
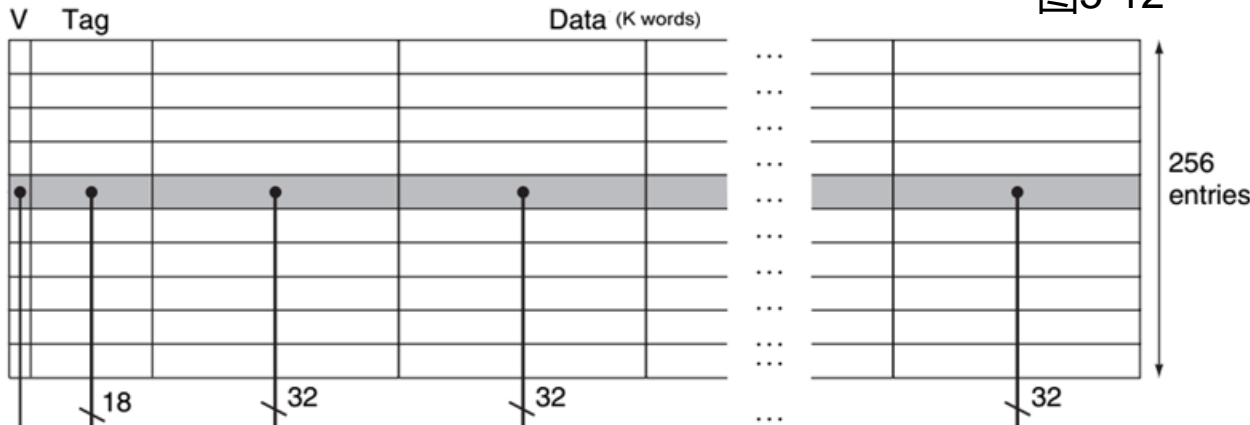


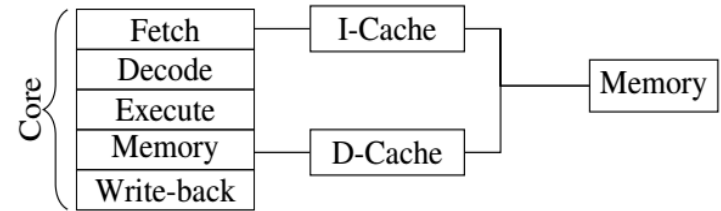
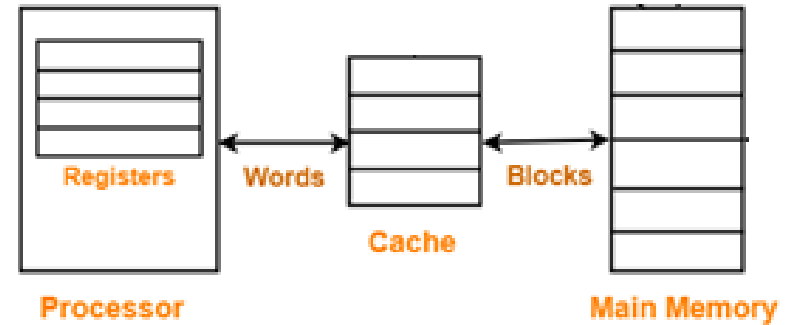
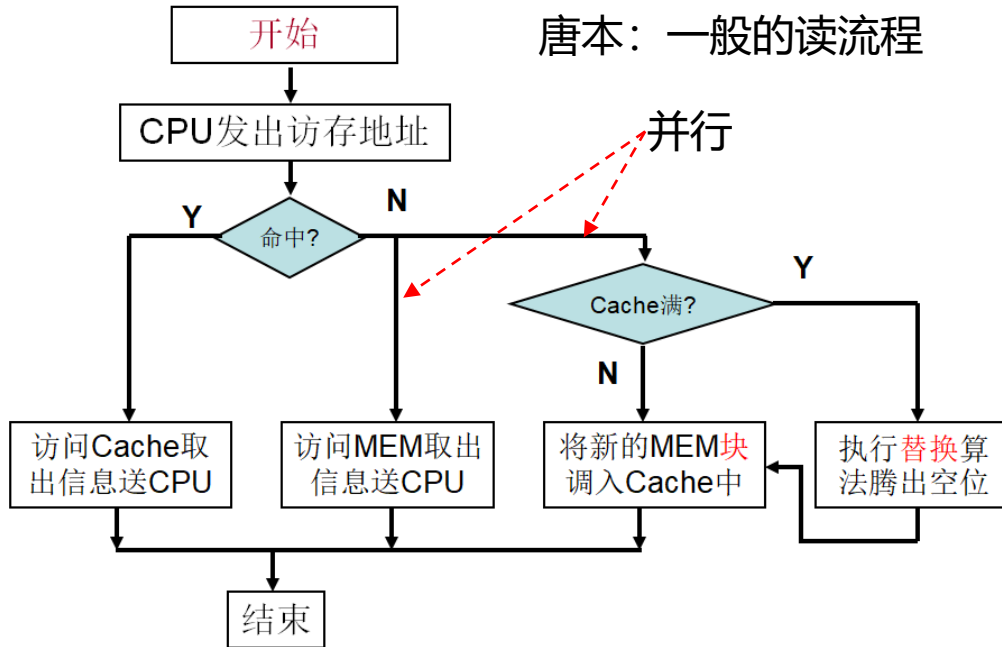
图5-12



Cache读访问过程, §5.3.1~2



唐本：一般的读流程



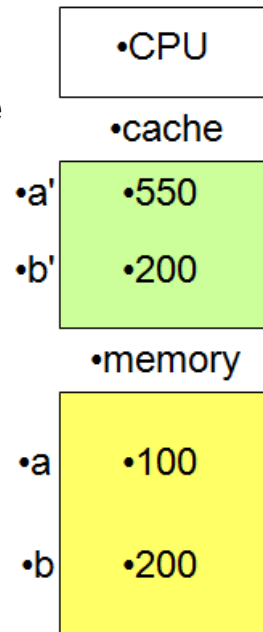
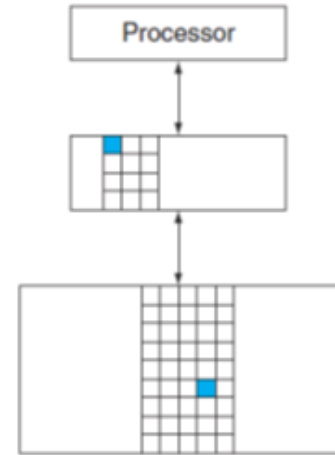
两个Cache同时miss?

- I\$ miss
 - 阻塞式: CPU stall, 访存将数据所在主存块 (block) 一次性调入, 置位控制位, 重新取指
 - 换入换出?
- D\$ miss: 处理过程与I\$类似。阻塞/非阻塞式 (OOO)
 - Dirty时换出需要写MEM
- 执行机构: CPU控制器+Cache控制器+MM控制器

write policy, §5.3.3

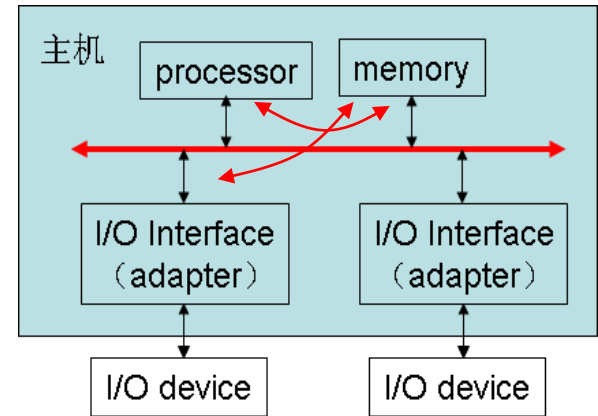
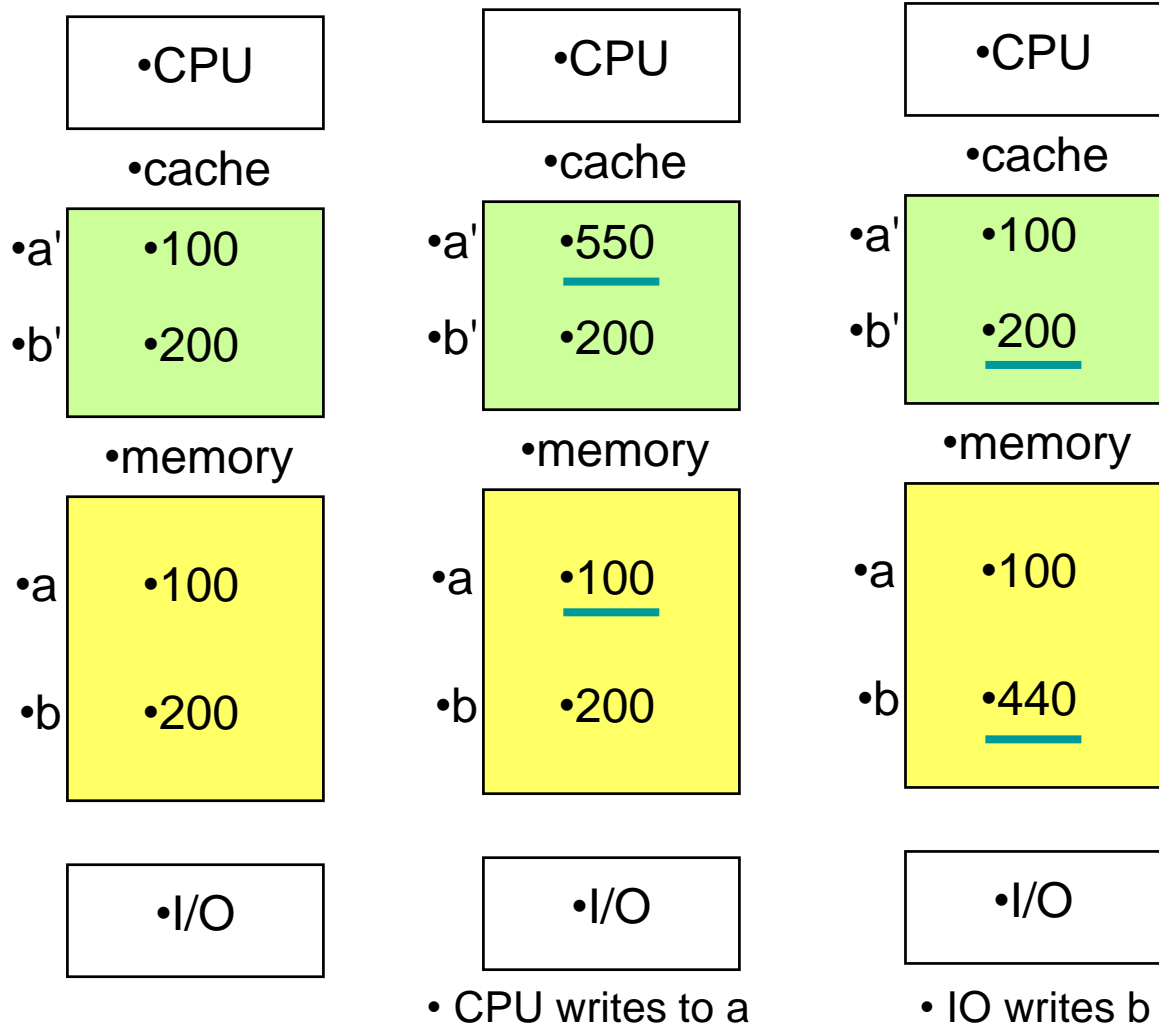


- 命中
 - 写透 (Write-through/Store-through, 写直达/“写穿透”)
 - 同时写入Cache和MEM
 - 写MEM~200cc: 写缓冲 (write buf, store buf)。满, 阻塞
 - 写回 (Write-back, “写返回”)
 - 只写Cache, 置Dirty (不一致), 替换时再整行写回MEM (写回缓冲)
- 不命中
 - 写分配 (write allocate) : 从mem读入Cache后再写
 - 也称fetch on write, MIPS采用。
 - 写不分配(no write allocate, write around): 只写mem, 不写cache
 - 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
 - 只写本地Cache和主存, 将其他处理器Cache相应块置 inValided
- 一致性问题: Cache值与主存, 多核的L1 Cache
- 写操作性能?





Cache与主存的不一致问题: \$5.3.3



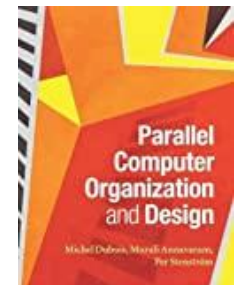
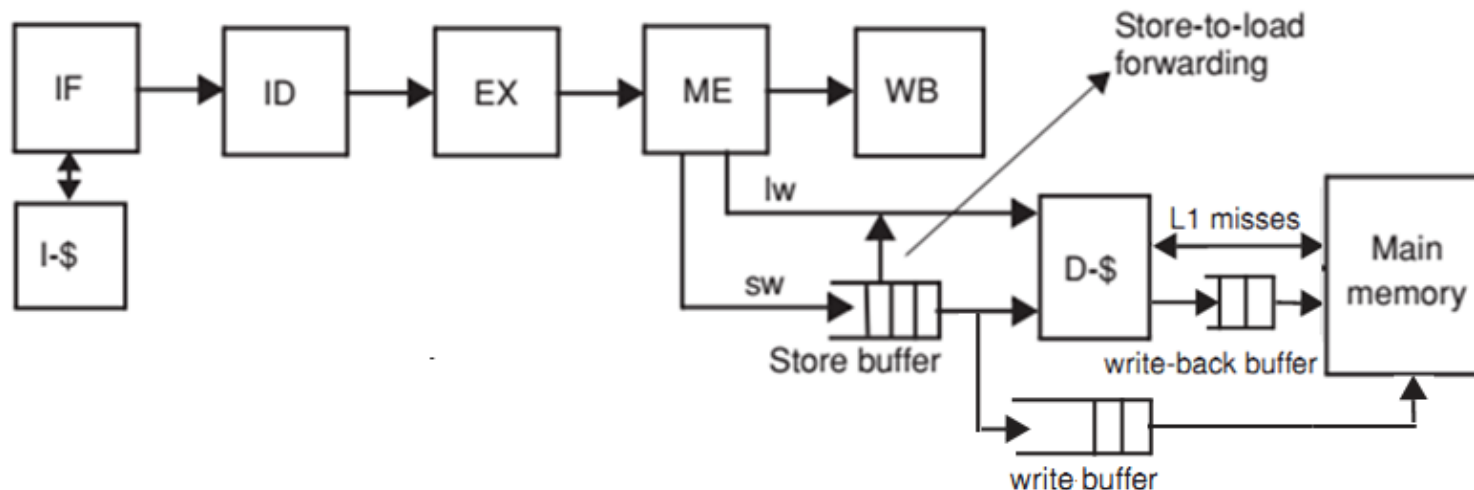
两种inconsistent场景:

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



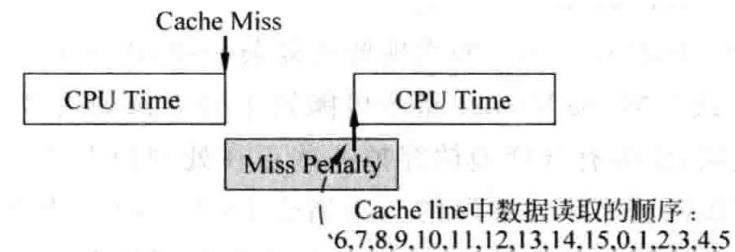
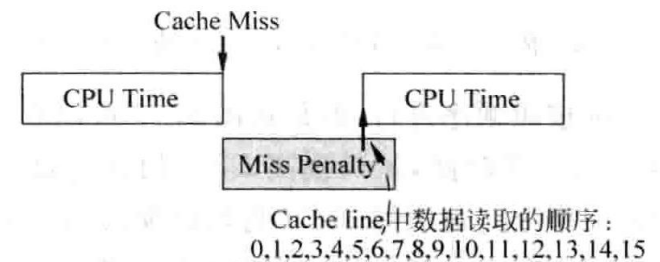
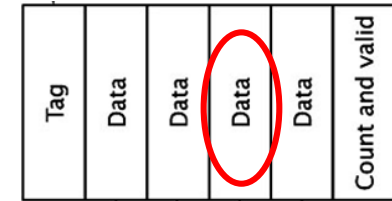
- 结构：Store buf (= Write buf? 支持store-load旁路)，Write-back buf
- 写回：不能直接写Cache，可流水化
 - 命中：两个周期
 - CC1 (= ppl的MM周期)：sw写store buf，指令完成。Cache控制器判断是否命中。
 - CC2 (unused周期)：将storebuf写入D\$，由Cache控制器完成。
 - miss：如需替换，则写回脏块->读入所需内存块（写分配）->storebuf写
 - write-back buf：存放待写入mem的脏块，尽快开始读入内存块
- 写透：可直接写Cache，一个CC完成
 - write buf (\$5.8.4)：按字访问



读写放大问题， §5.3.1 【详解】



- 读写一个**字**，被放大为读写一个**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\$。
 - “重启”：重新执行miss的指令
 - 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. 全相联映像 (续)

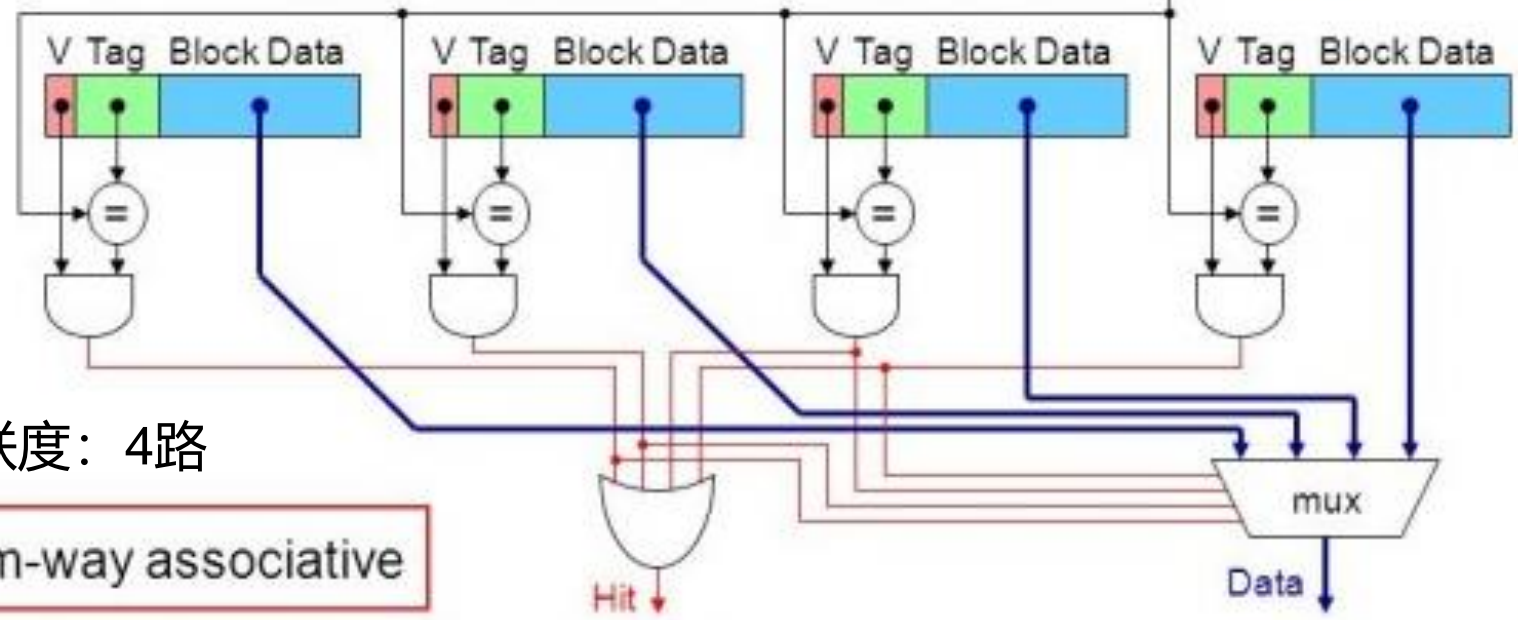
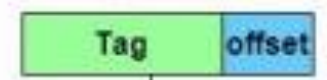
访问顺序	1	2	3	4	5	6	7	8	字块号
地址	22	26	22	26	16	4	16	18	
块分配情况	22	22	22	22	22	22	22	22	0
	-	26	26	26	26	26	26	26	1
	-	-	-	-	16	16	16	16	2
	-	-	-	-	-	4	4	4	3
	-	-	-	-	-	-	-	18	4
	-	-	-	-	-	-	-	-	5
	-	-	-	-	-	-	-	-	6
	-	-	-	-	-	-	-	-	7
操作状态	调进	调进	命中	命中	调进	调进	命中	调进	

全相联映射数据访问过程



❖ If m blocks exist then

- ❖ m comparators are needed to match tag
- ❖ Cache data size = $m \times 2^b$ bytes

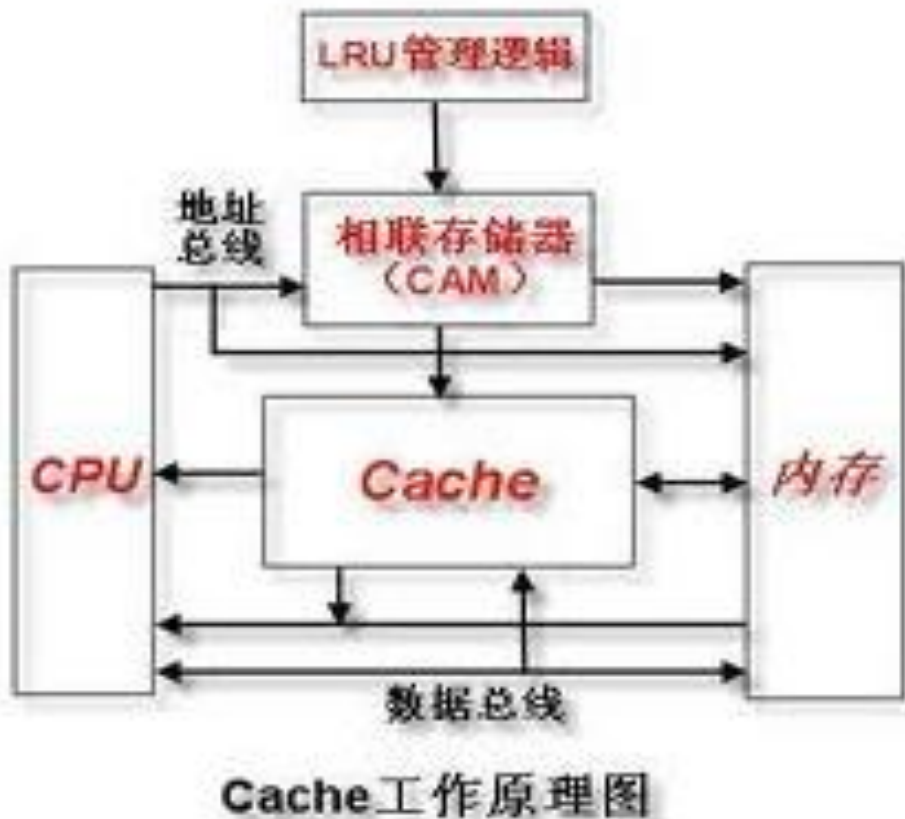


相联度：4路

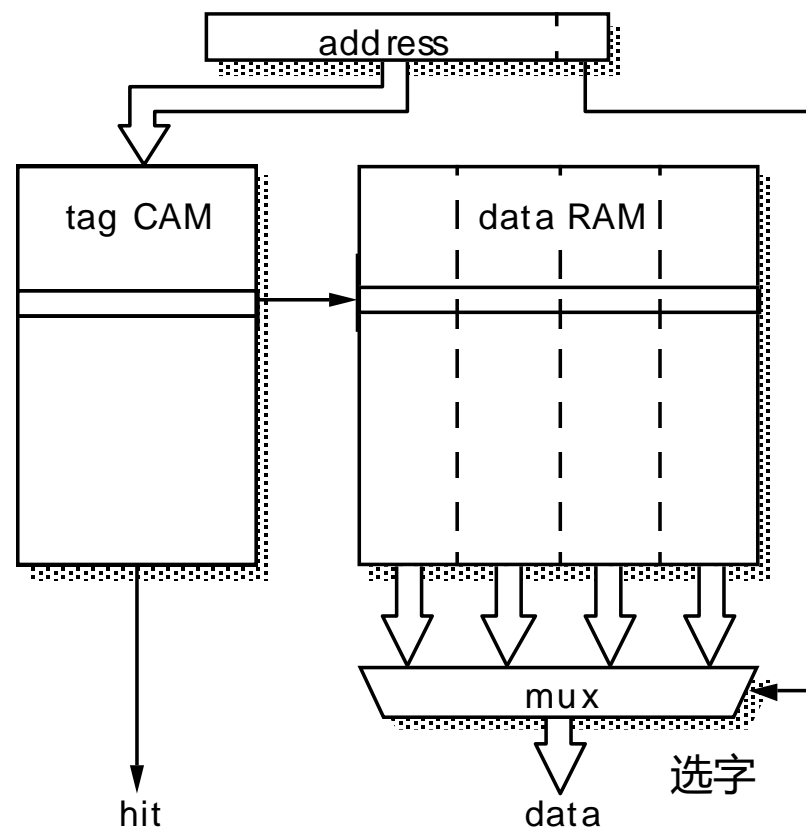
m-way associative

命中 (比较tag和V) , 选行, 选字? 命中比较与数据输出是并行or串行?
——适于 “小Cache”

全相联Cache的Tag比较, \$5.4.2 【精解】



CAM(Content Addressable Memory)



- Tag与Data存储相分离
- CAM提高tag搜索性能: 比较tag, 选行。与选字并行?
- 相联度 ≥ 8 路时使用CAM



2. 直接映射Cache：映射关系

主存分块（字块= 2^b 字），再按Cache行数分段。主存字块数： $M=2^m$

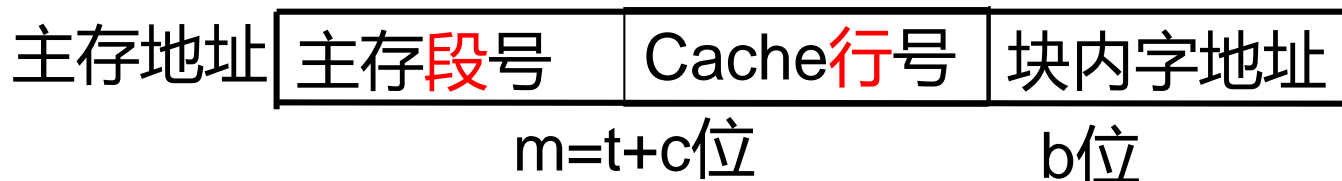
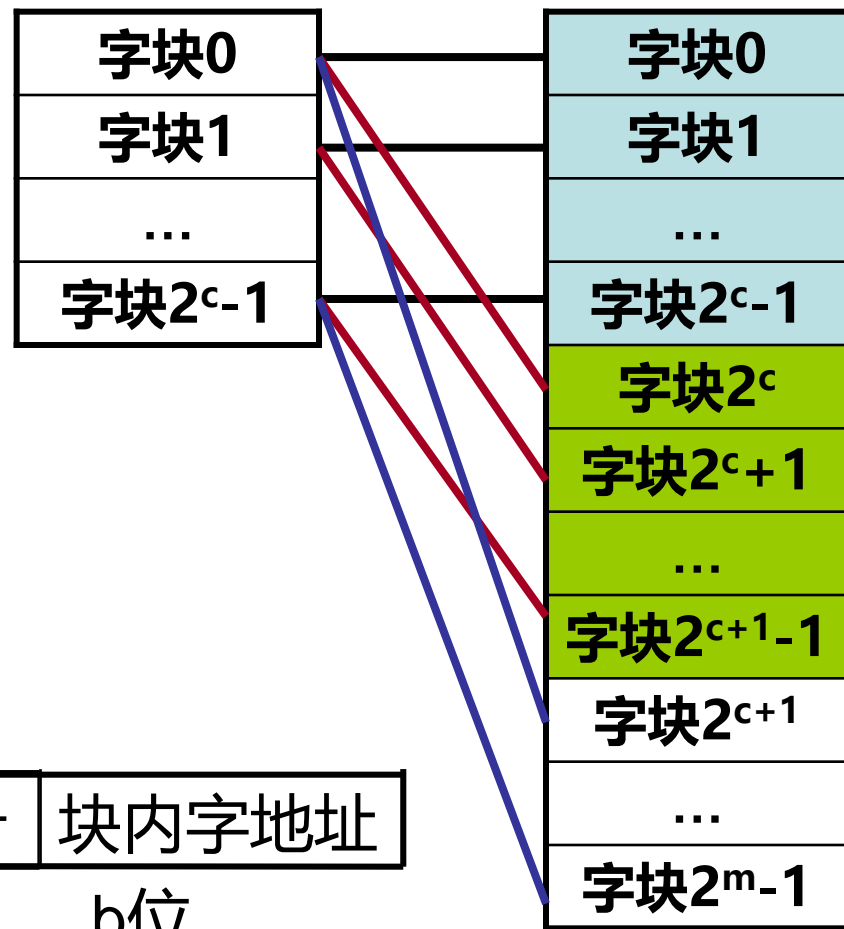
Cache一行一块，共 c 行，总字块数： $C=2^c$ Cache

主存储器

主存与缓存映射关系： $i=j \bmod C$

或 $i=j \bmod 2^c$

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





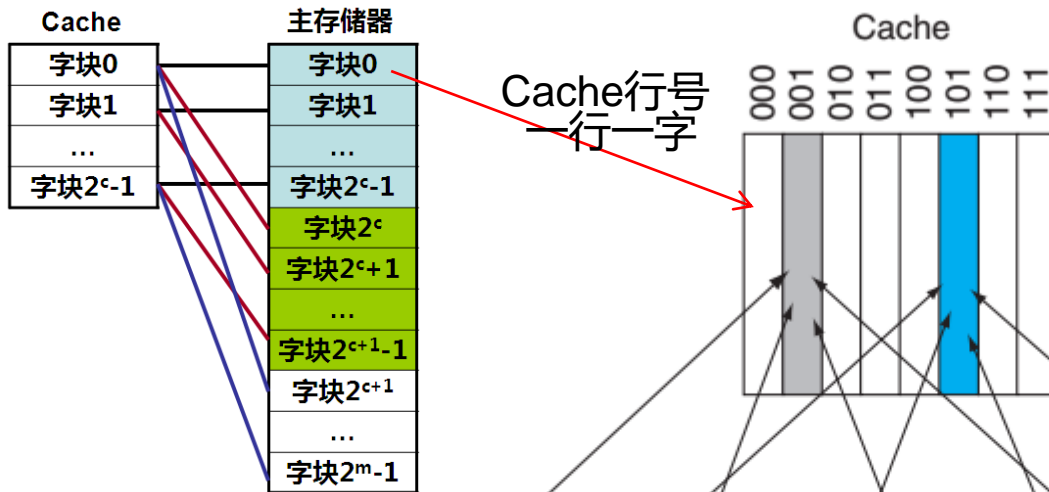
例：direct mapped Cache

主存地址

主存段号	Cache行号	块内字地址
------	---------	-------

Cache容量 = 8 words
数据 (字块) = 1 word
则：分8 lines

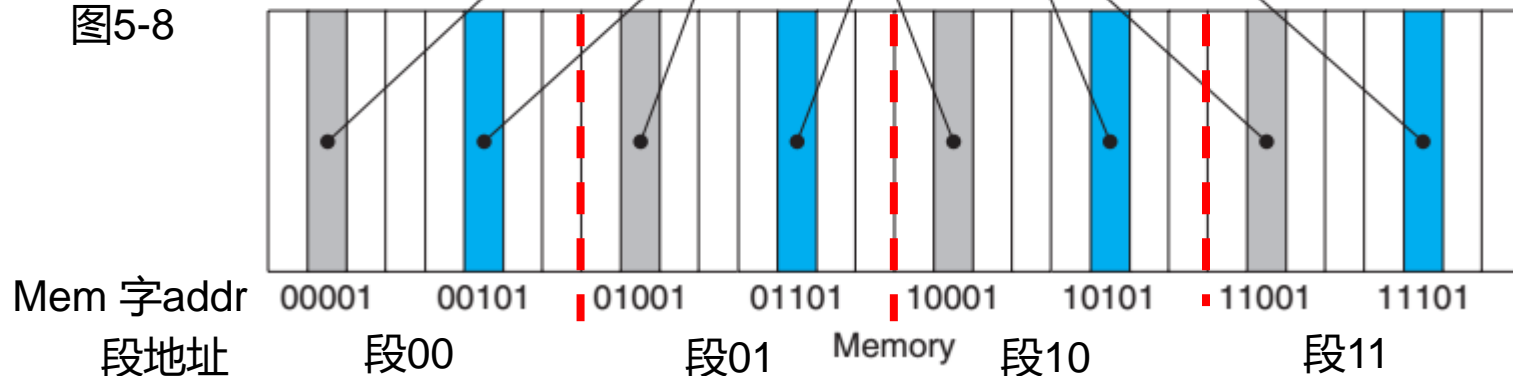
内存32字
则：内存每段8个字 (块)，分4段
1行 = 1 块 = 1 字



字块位置唯一固定：地址冲突？

命中判断，tag=?
相联度？

图5-8



地址域 (图5-12) : Tag (段#) | Index (Cacheline#) | BlockOffset | ByteOffset

直接映象Cache访问：颠簸（乒乓）



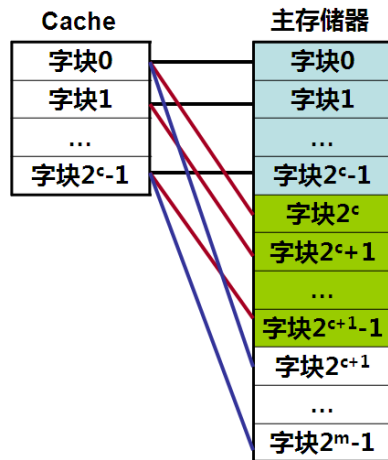
访问顺序	1	2	3	4	5	6	7	8	字块号
块地址	22	26	22	26	16	4	16	18	
块分配情况	-	-	-	-	16	16	16	16	0
	-	-	-	-	-	-	-	-	1
	-	26	26	26	26	26	26	18	2
	-	-	-	-	-	-	-	-	3
	-	-	-	-	-	4	4	4	4
	-	-	-	-	-	-	-	-	5
	-	-	-	-	-	-	-	18	6
	-	-	-	-	-	-	-	-	7
操作状态	调进	调进	命中	命中	调进	调进	命中	调进	

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

如果之后连续访问26和18?

例：Cache access

- Cache=8行 (块)
 - 初始为空, $V=N$
- 9次读访存
 - 第8次: 替换?



主存地址

主存段号	Cache行号	块内字地址
------	---------	-------

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})

图5-9

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})

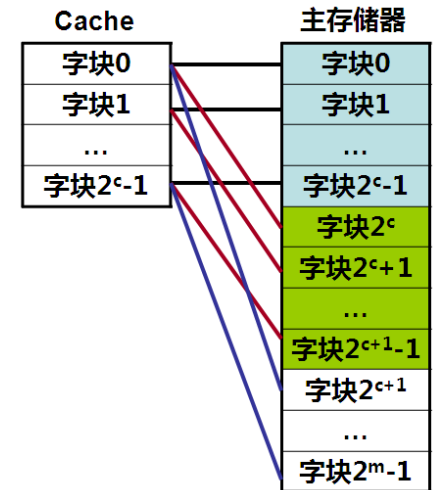
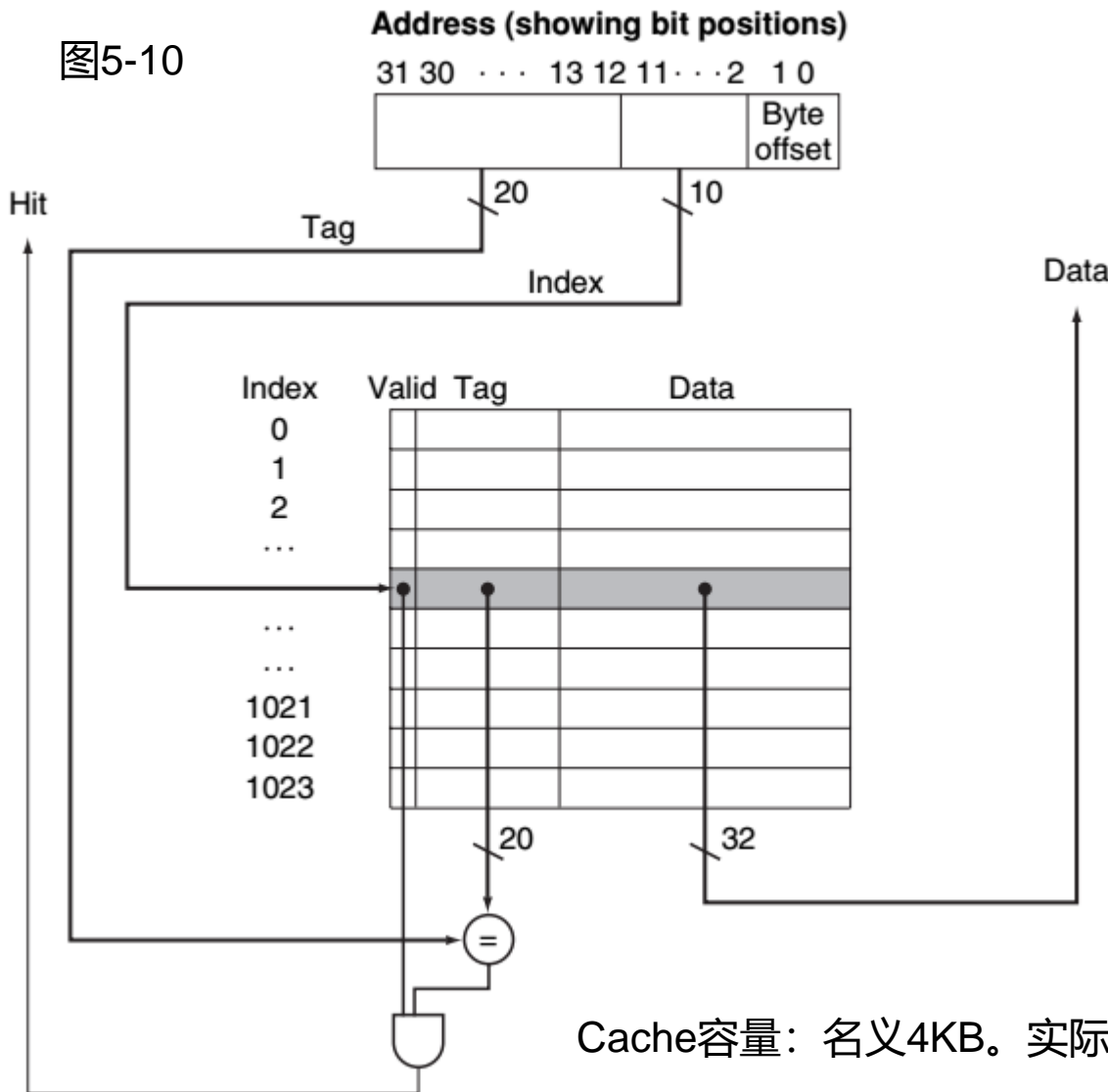
Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.9c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$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

例：读操作数据通路，命中？ Miss？



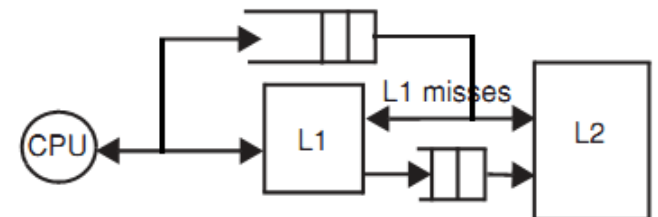
Tag (段#) | Index (Cacheline#) | BlockOffset | ByteOffset

图5-10



本例：data = 1 word;
 块内字地址 = BlockOffset = 0位
 字节地址2位
 Tag比较与取Data并行

与CPU和mem的接口？
 Hit信号去向？



Cache容量：名义4KB。实际？

FastMATH处理器：直接映射



Intrinsity公司
MIPS架构, 12级ppl
Cache容量? 内存大小?

读miss: mem->Cache->cpu
写: 写透和写回可选!
WriteBuf容量为1行, 数据通路?

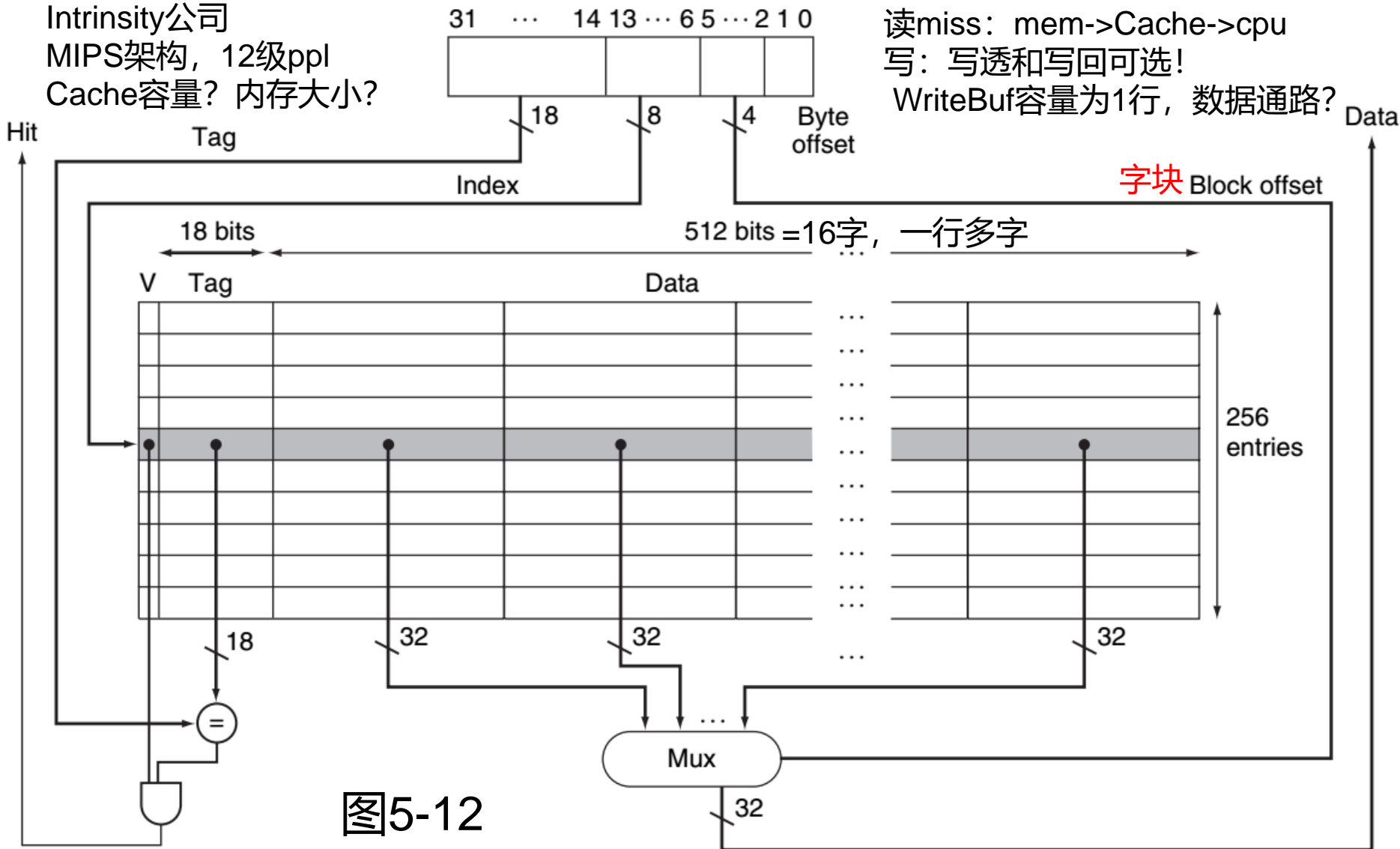


图5-12

直接映像特点

- 优点：实现简单

- 数据块映射位置固定

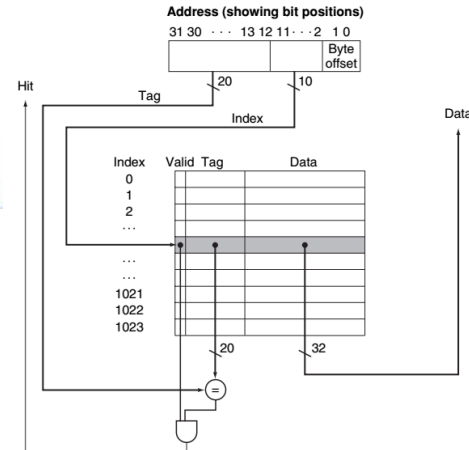
- 一次比较：只需利用主存地址的某些位（index）直接判断，就可确定所需字块是否在缓存中。

- 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。



3. 组相联映象 (2 way-set-associated)



主存储器

Cache分组,

主存分段,

段大小 = 组数

$r = \text{组内块数} - 1$

相联度

Tag = 段号

$r = 0 ? c \text{组} 1 \text{路}, \text{直}$

$r = c ? 1 \text{组} c \text{路}, \text{全}$

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

字块0

字块1

.....

字块 $2^{c-r} - 1$

字块 2^{c-r}

字块 $2^{c-r} + 1$

.....

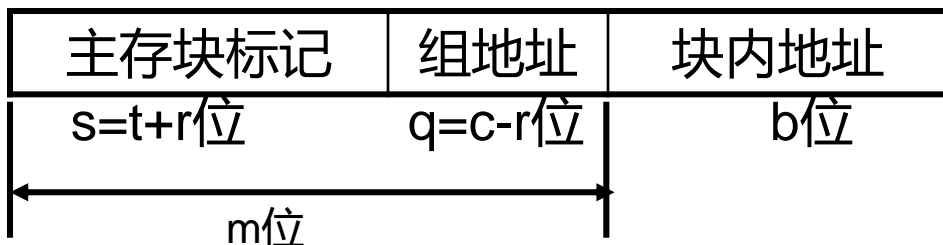
字块 2^{c-r+1}

.....

字块 $2^m - 1$

段

主存地址





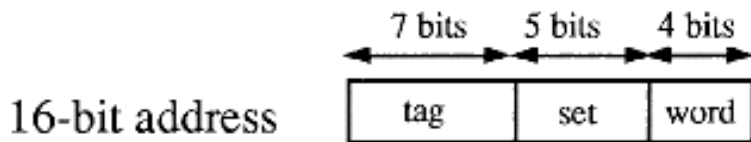
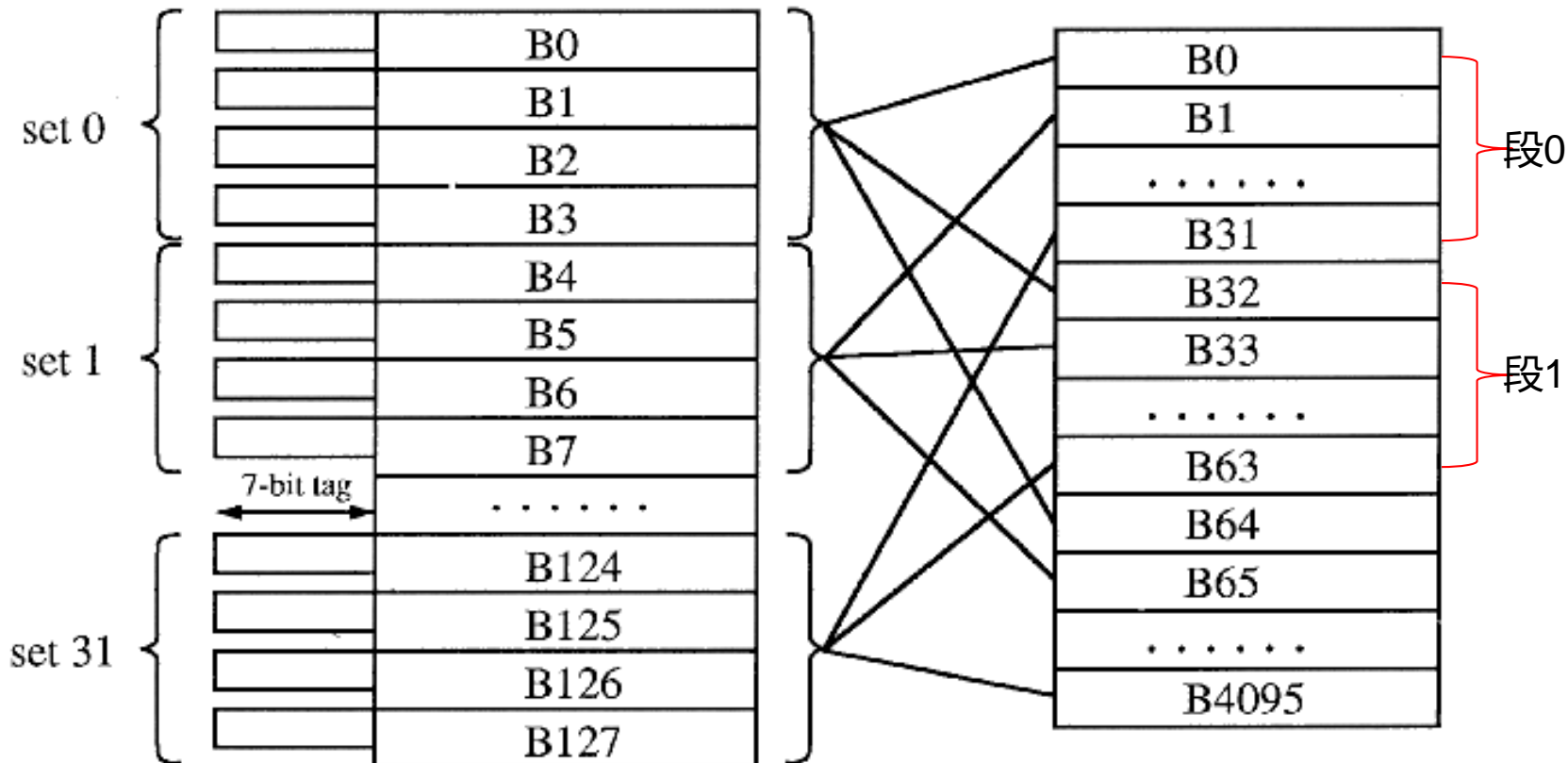
N路 (X) 组相联映象 (续)

- 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

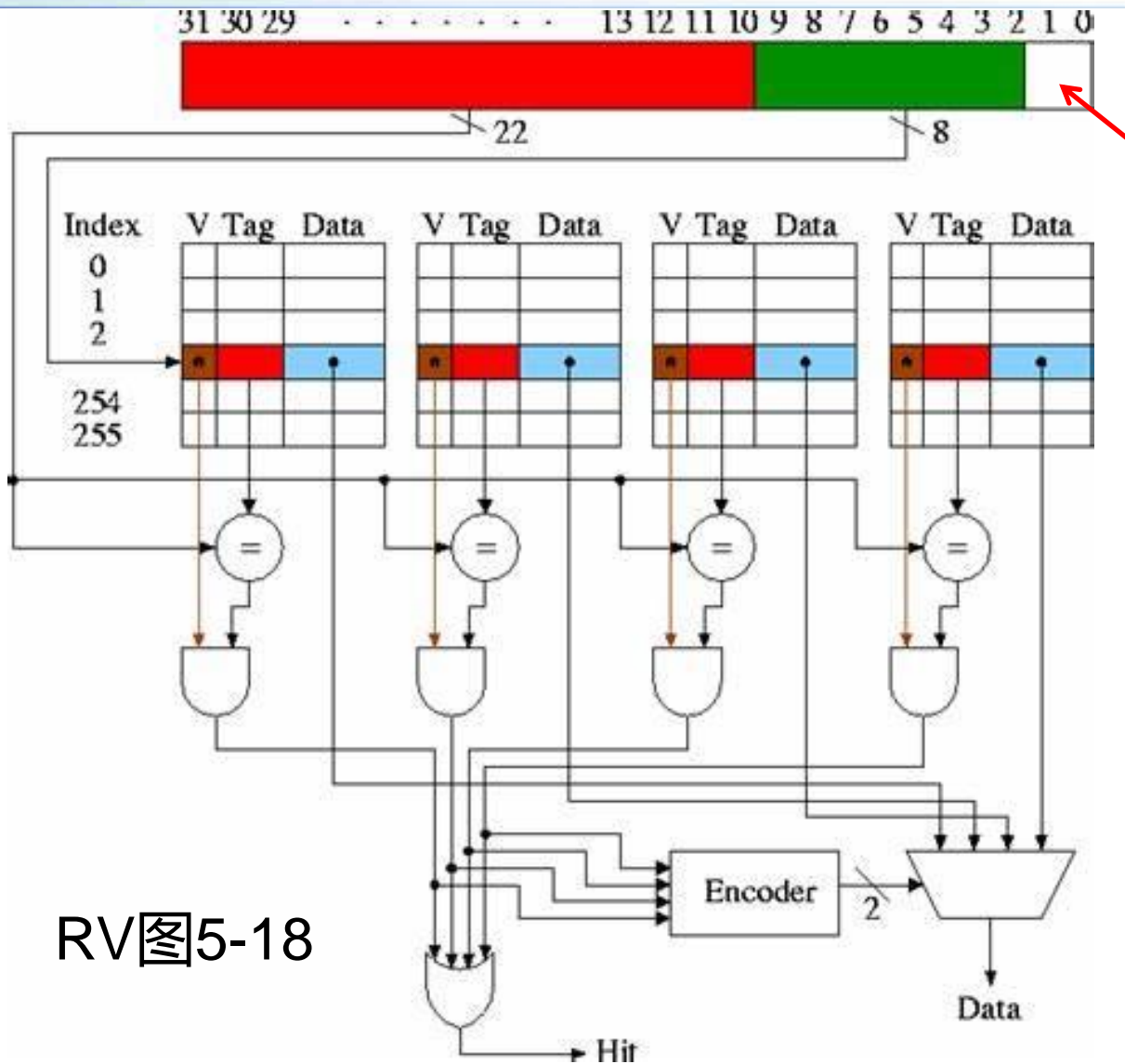


-- Set-associative mapping

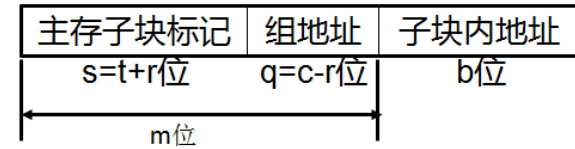




Ex: 4-way set associative Cache

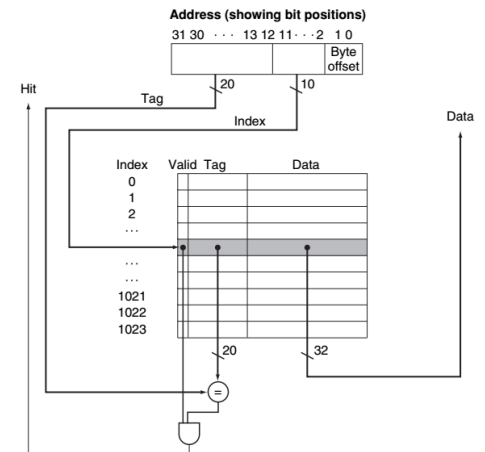


主存地址



- 一组4路 (line)
- 一路 (块) 一字 (4字节)
- 定位set (index)
- 比较tag&V
- 选路选字合二为一

RV图5-18



- 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

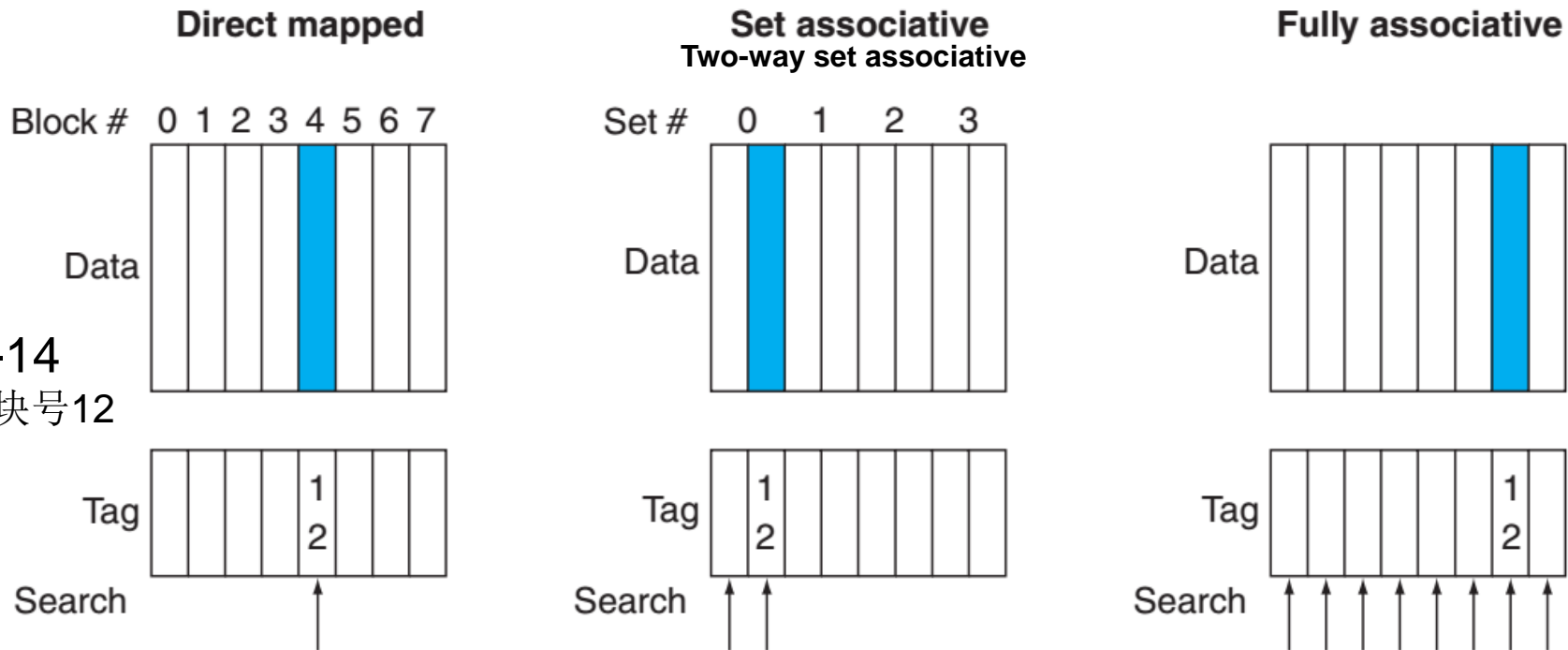


图5-14
主存块号12

- 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 = Num# of ways



图5-15

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

块/行/路, 组/段

- 1) Cache和Mem分块;
- 2) Cache分组, MM分段

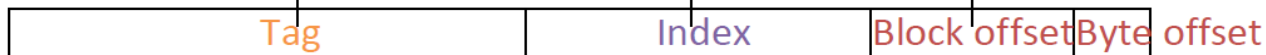
Data = n words

Cache容量 (总行数)
= 组数 × 相联度

Used for tag comparison

Selects the set

Selects the word in the block



Decreasing associativity ←

→ Increasing associativity

Direct mapped (only one way)

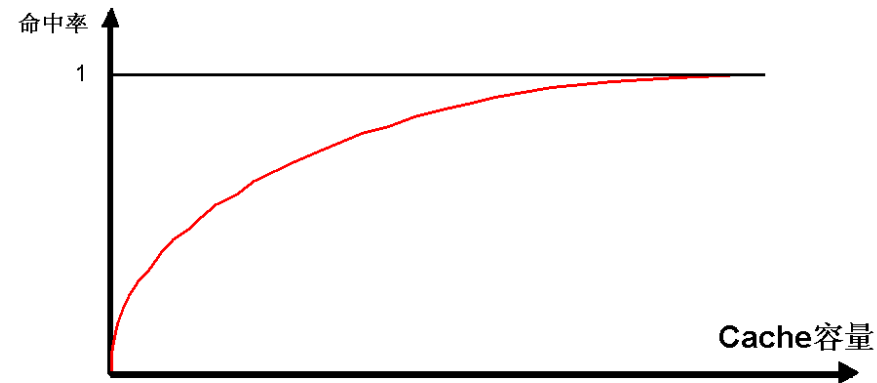
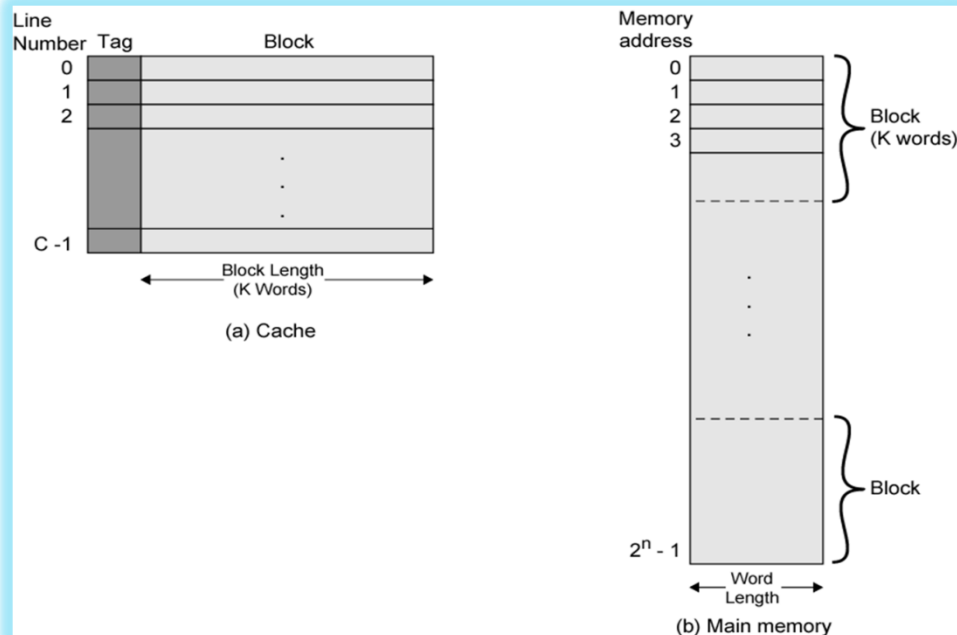
Fully associative (only one set)

相联度 “一般为2~16”, §5.8.1
“全相联用于小Cache” !



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Cache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

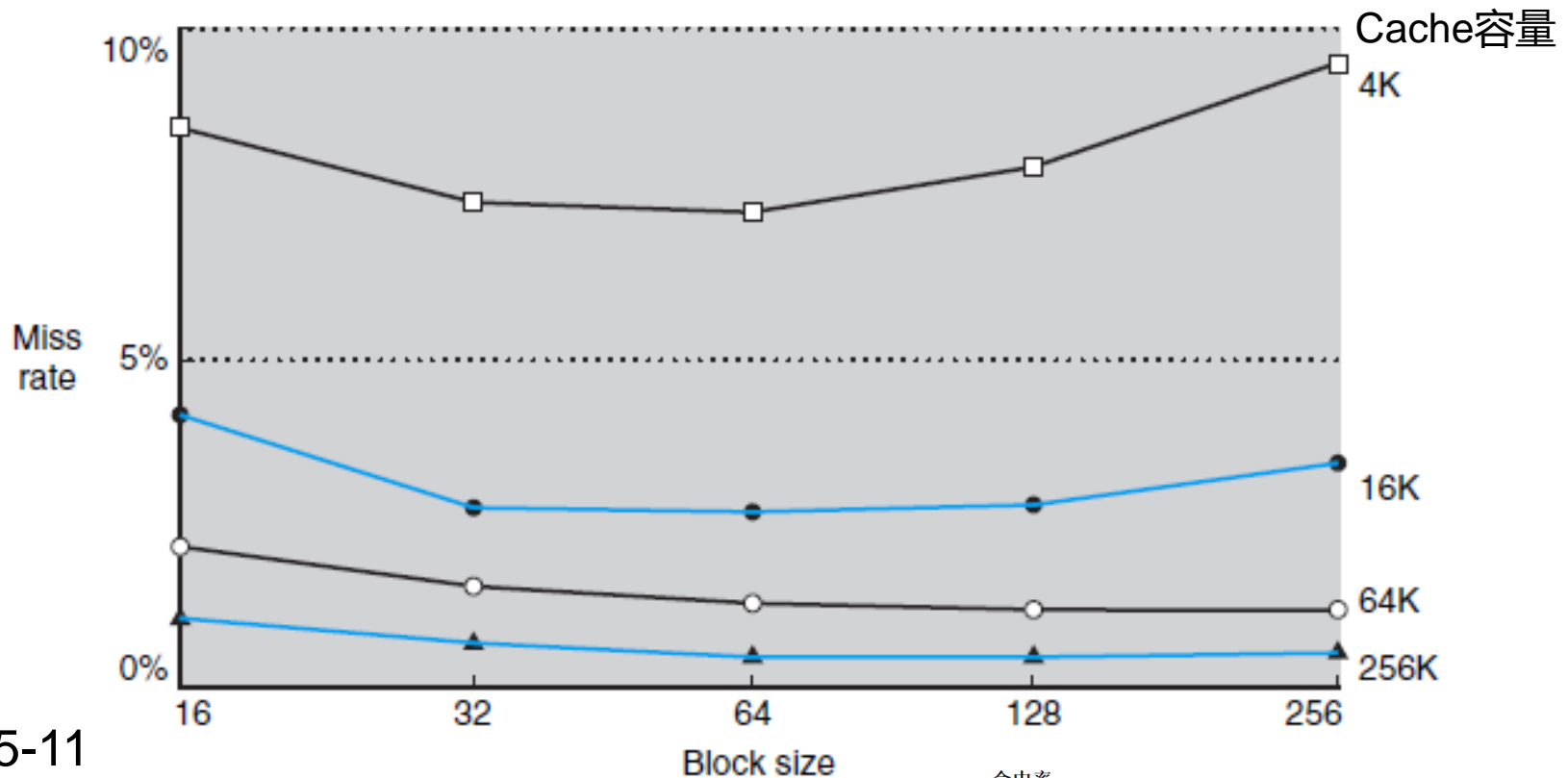
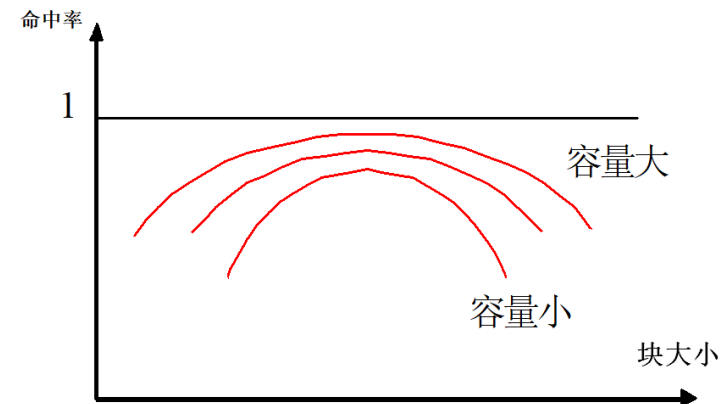


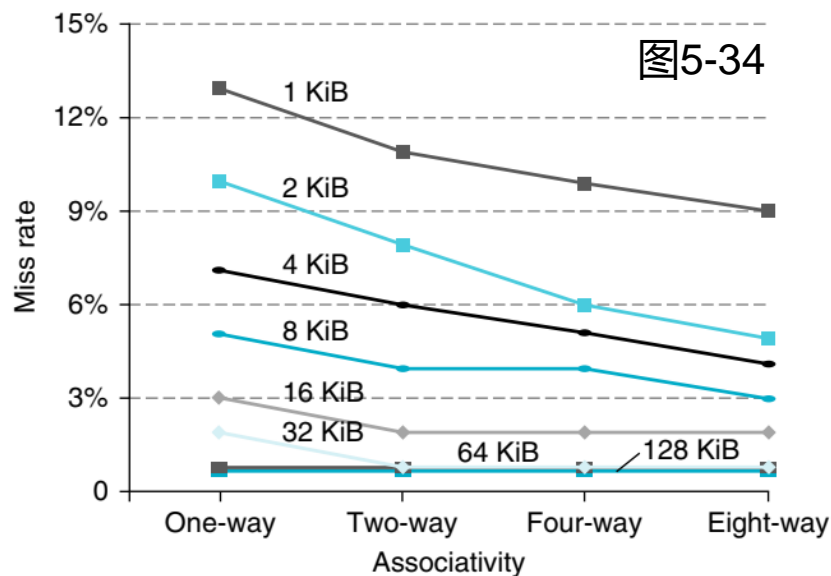
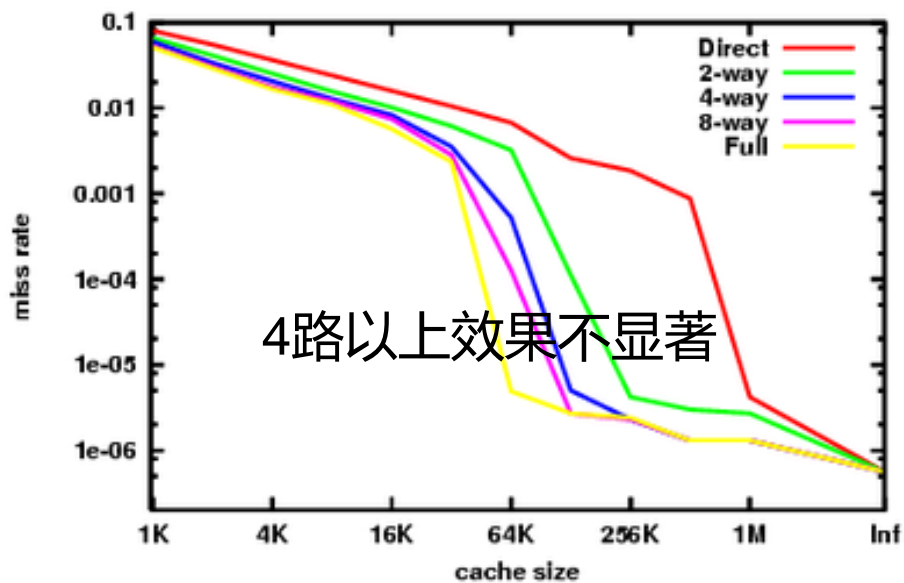
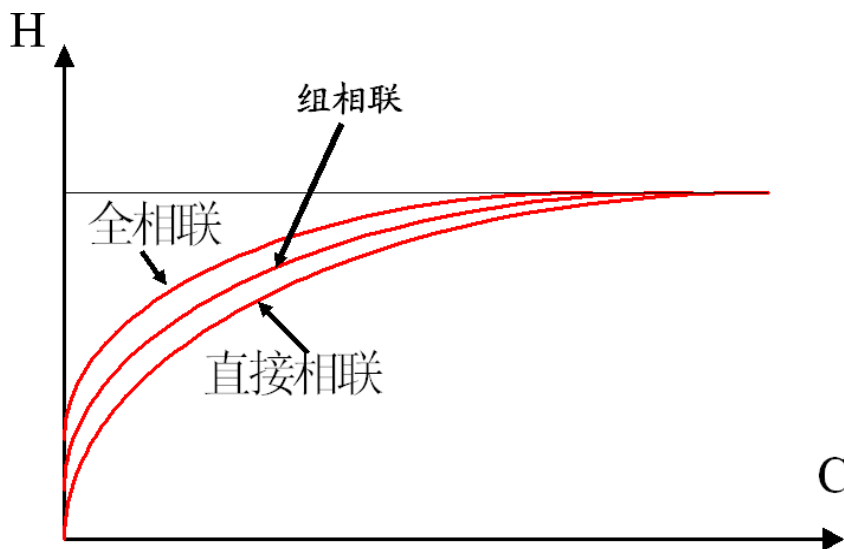
图5-11

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



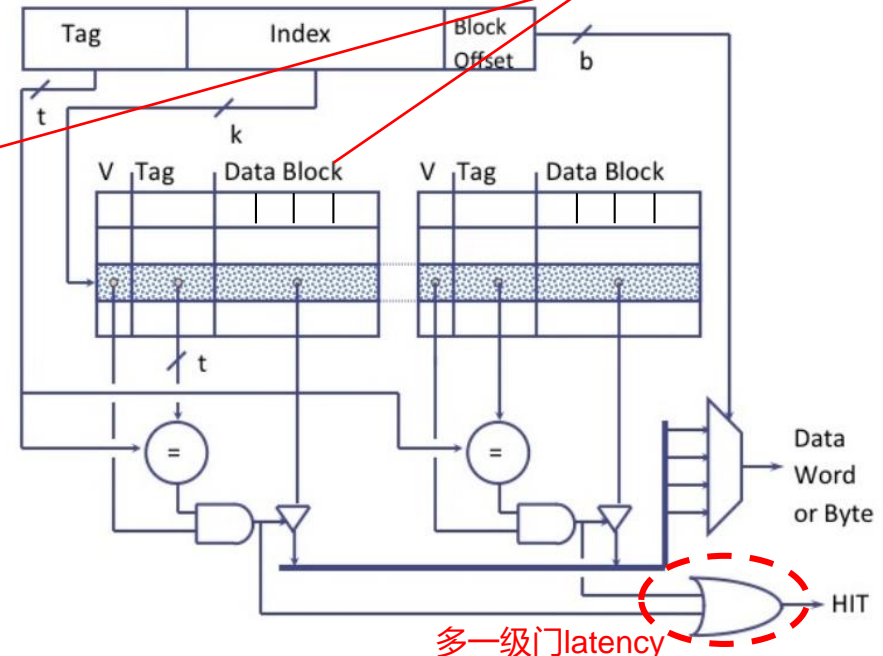
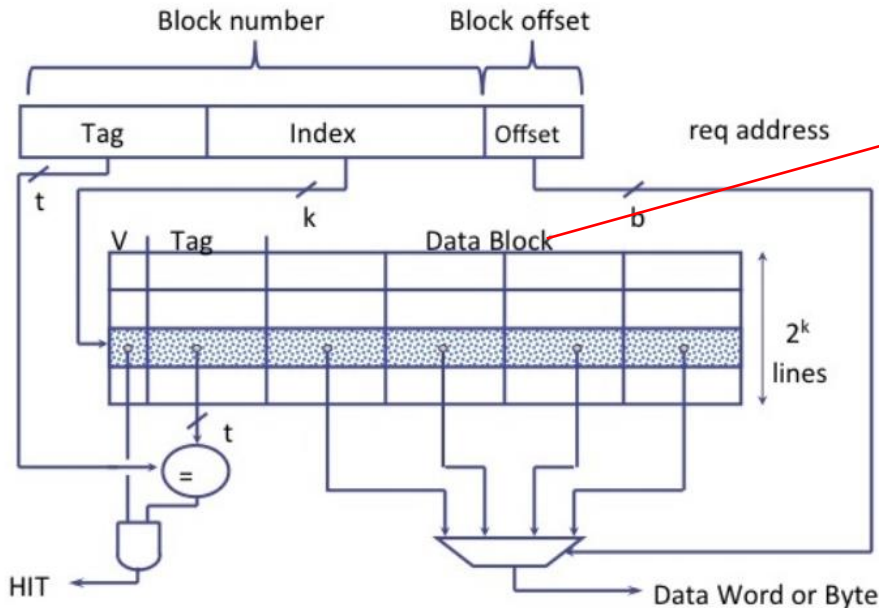


直接映射 vs 2路组相联：实现开销，命中时间

- tag比较开销：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：命中判断与data传输并行
 - 仅1 way
 - Cache Block is available **BEFORE** Hit/Miss
- 相联度（路数）增加，**命中率**增加，**命中时间**增加

H&P §5.2, 命中时间:

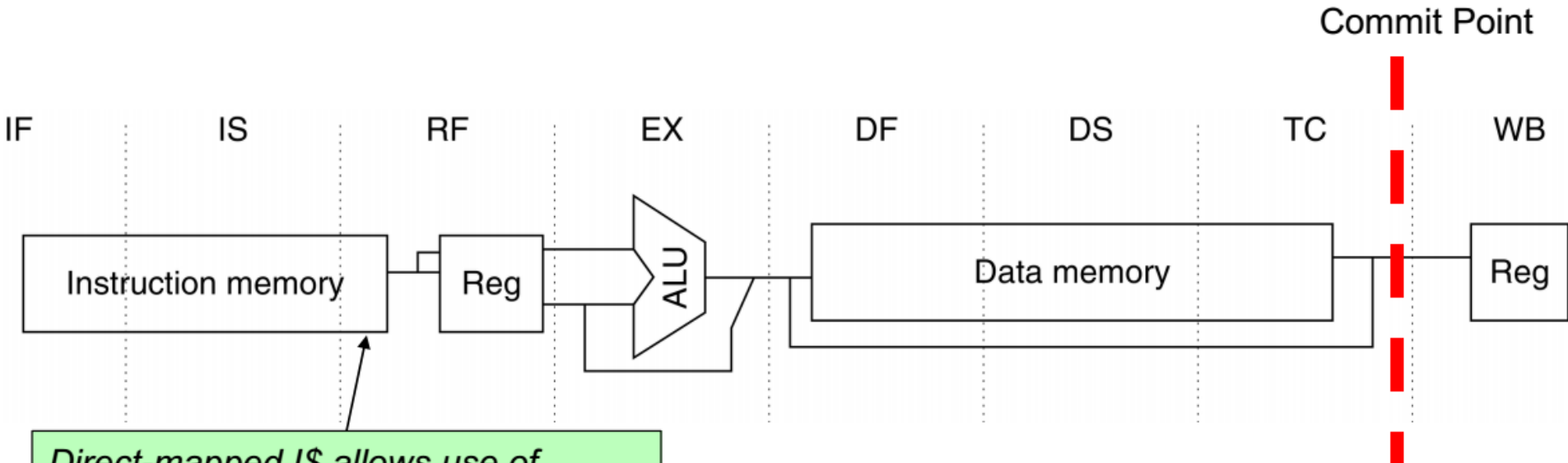
- 直接映射比2-ways快1.2~1.5倍
- 2-ways比4-ways快1.02~1.11倍
- 4-ways比全相联快1.0~1.08倍



一行=4 words

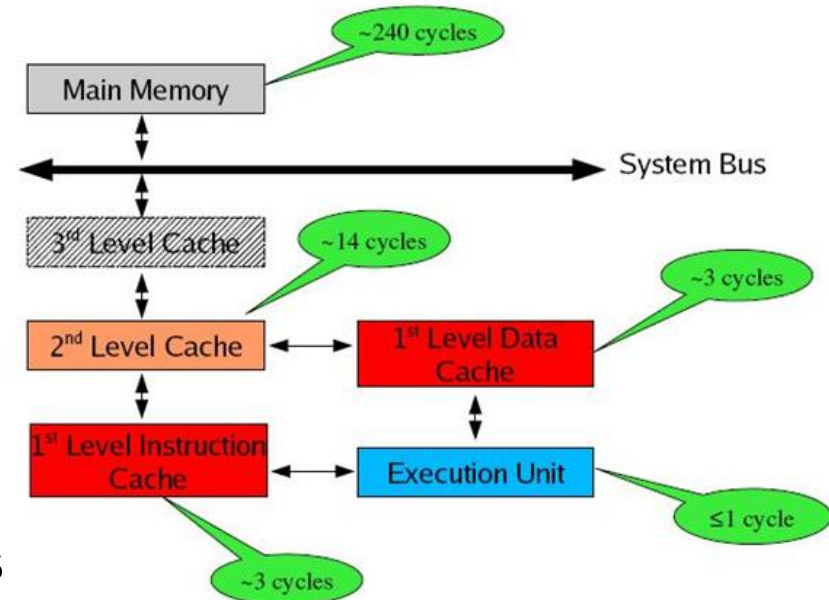
多一级门latency

Deeper Pipelines: MIPS R4000



Direct-mapped I\$ allows use of instruction before tag check complete

- Longer pipeline results in
 - Decreased cycle time
 - Increased load-use delay latency
 - Increased branch resolution latency
 - More bypass paths
- EU 1 cycle, L1\$ 命中时间 ~3 cycles





\$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

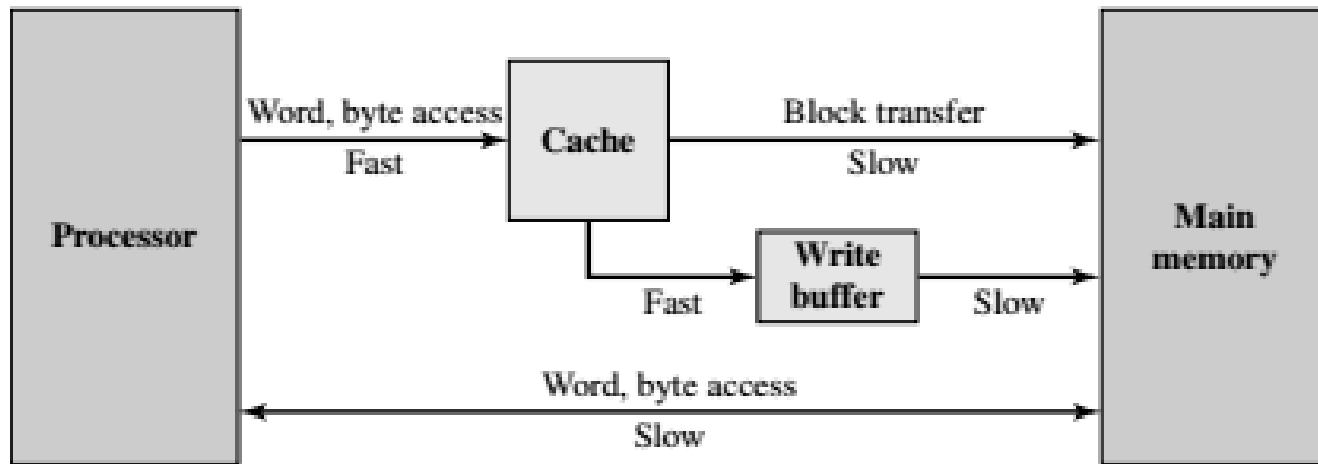


Figure 4.19 ARM Cache and Write Buffer

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。

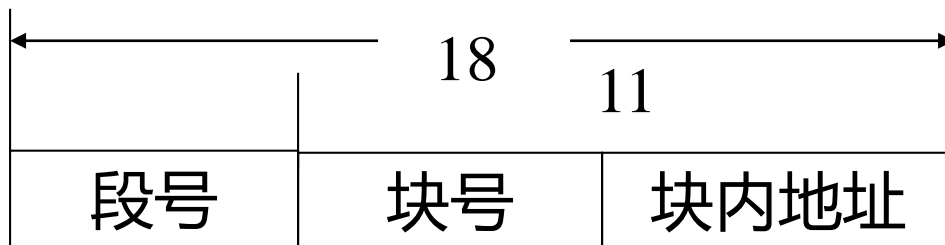
(2) 主存有 $256K/16=16384$ 个块。

(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。



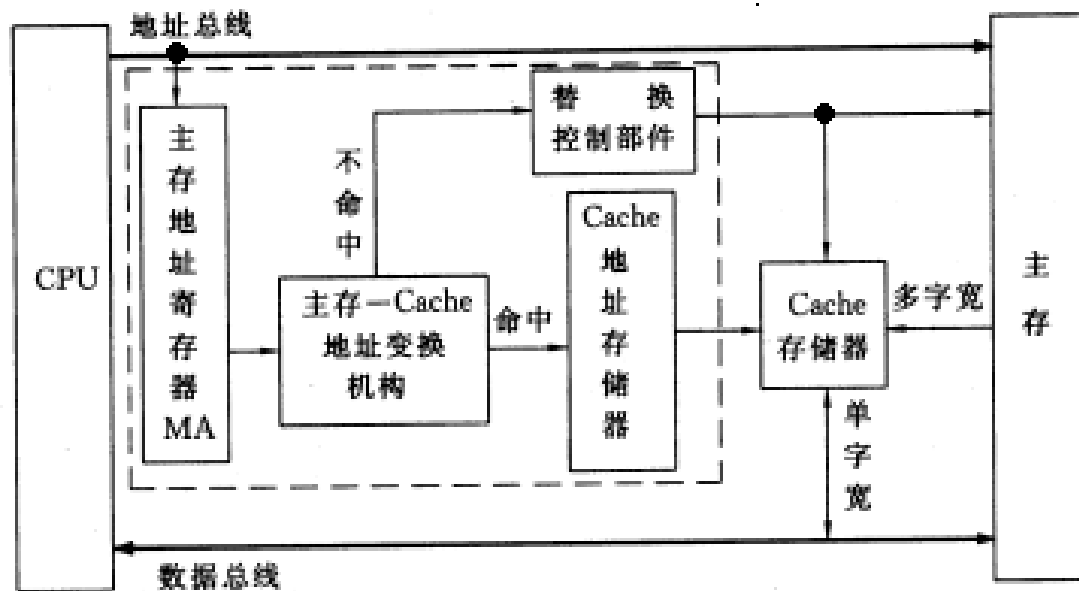
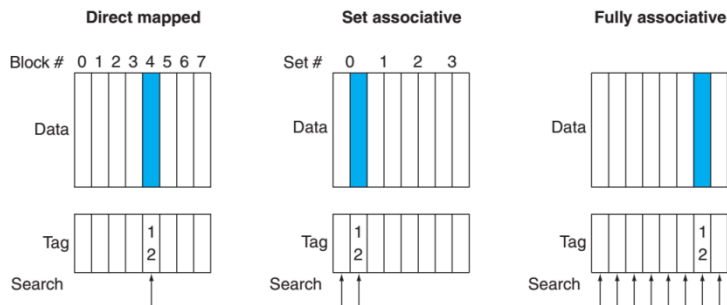
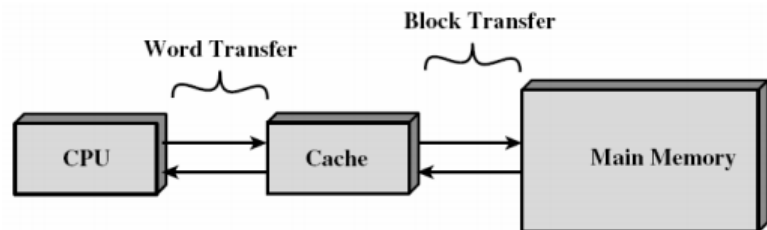


替换：\$5.4.3, \$5.8.3

- 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

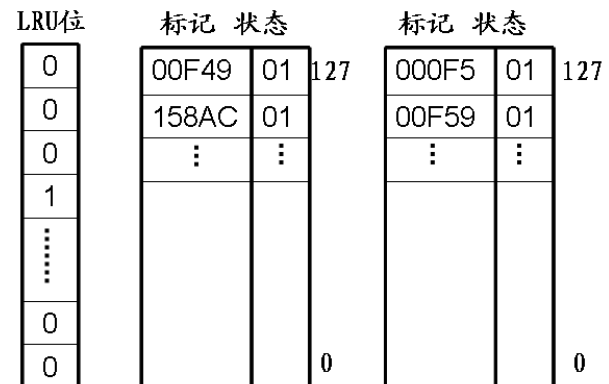
- 访存接口：地址，数据，命令，状态





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
 - 随机: 失效率比LRU高约1.1倍, 但可能比近似LRU好, §5.8.3
 - **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
 - PLRU
 - 相联度越大, 实现成本越高





替换算法与命中率

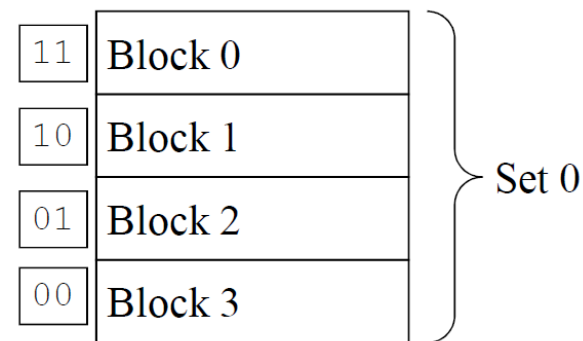
时间 t	1	2	3	4	5	6	7	8	9	10	实际命中次数
地址流	P1	P2	P1	P5	P4	P1	P3	P4	P2	P4	
先进先出算法 (FIFO 算法)	1	1	1	1*	4	4	4*	4*	2	2	2 次 颠簸
		2	2	2	2*	1	1	1	1*	4	
				5	5	5*	3	3	3	3*	
	调入	调入	命中	调入	替换	替换	替换	命中	替换	替换	
最近最少使用算法 (LRU 算法)	1	1	1	1	1	1	1	1*	2	2	4 次
		2	2	2*	4	4	4*	4	4	4	
				5	5*	5*	3	3	3*	3*	
	调入	调入	命中	调入	替换	命中	替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

三种页面替换算法对同一个页地址流的调度过程

Implementing LRU replacement



- 计数法：按路，按组，混合
 - CacheLine = VCD + tag + block(K words)
- 例1：4-ways组相联，按路计数，换出计数最大者
 - Hit：命中者保持
 - Increment lower counters
 - Reset counter to 00
 - Miss：Replace the 11, Set to 00
 - Increment all other counters



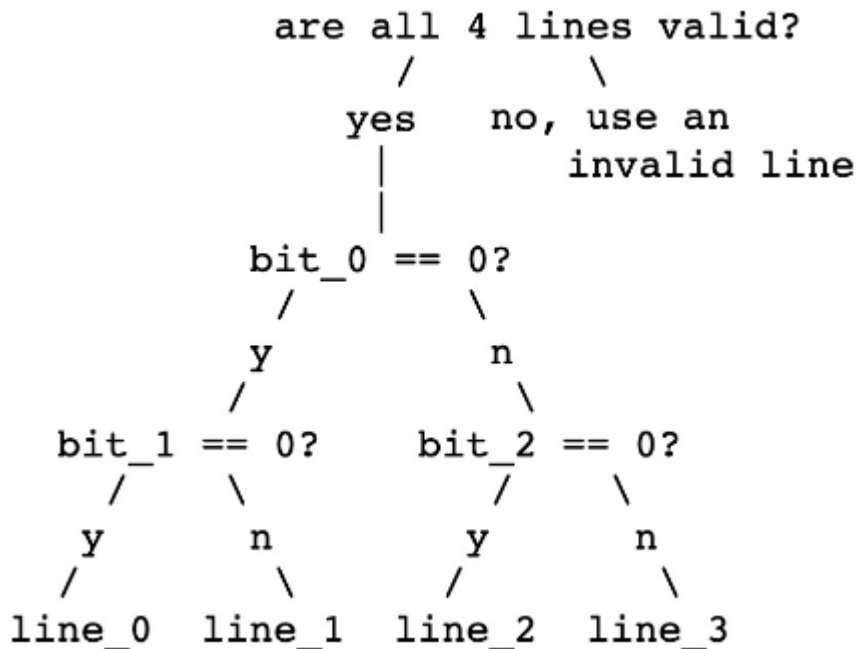
- 例2：2-ways组相联
 - 按组计数

		Offset						LRU
		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



psudo-LRU: Tree LRU

- 一个4路组相连的cache: 一组4行 (L0, L1, L2, L3)
- LRU位: 用路数w-1位表示, 如r0r1r2
- 访问历史二叉树: LRU位的每一位表示一个分叉点
 - 状态更新: 仅更新根到被访问叶节点路径上的节点状态。
 - 1表示最近访问left, 0表示访问right。例: ref to表
 - 复位或排空时, 全部LRU位都清零
- 置换选择: 0则left子树, 1则right子树。例: 置换树、replace表



ref to	next state	state	replace
line_0	11_	00x	line_0
line_1	10_	01x	line_1
line_2	0_1	1x0	line_2
line_3	0_0	1x1	line_3

('_' means unchanged)

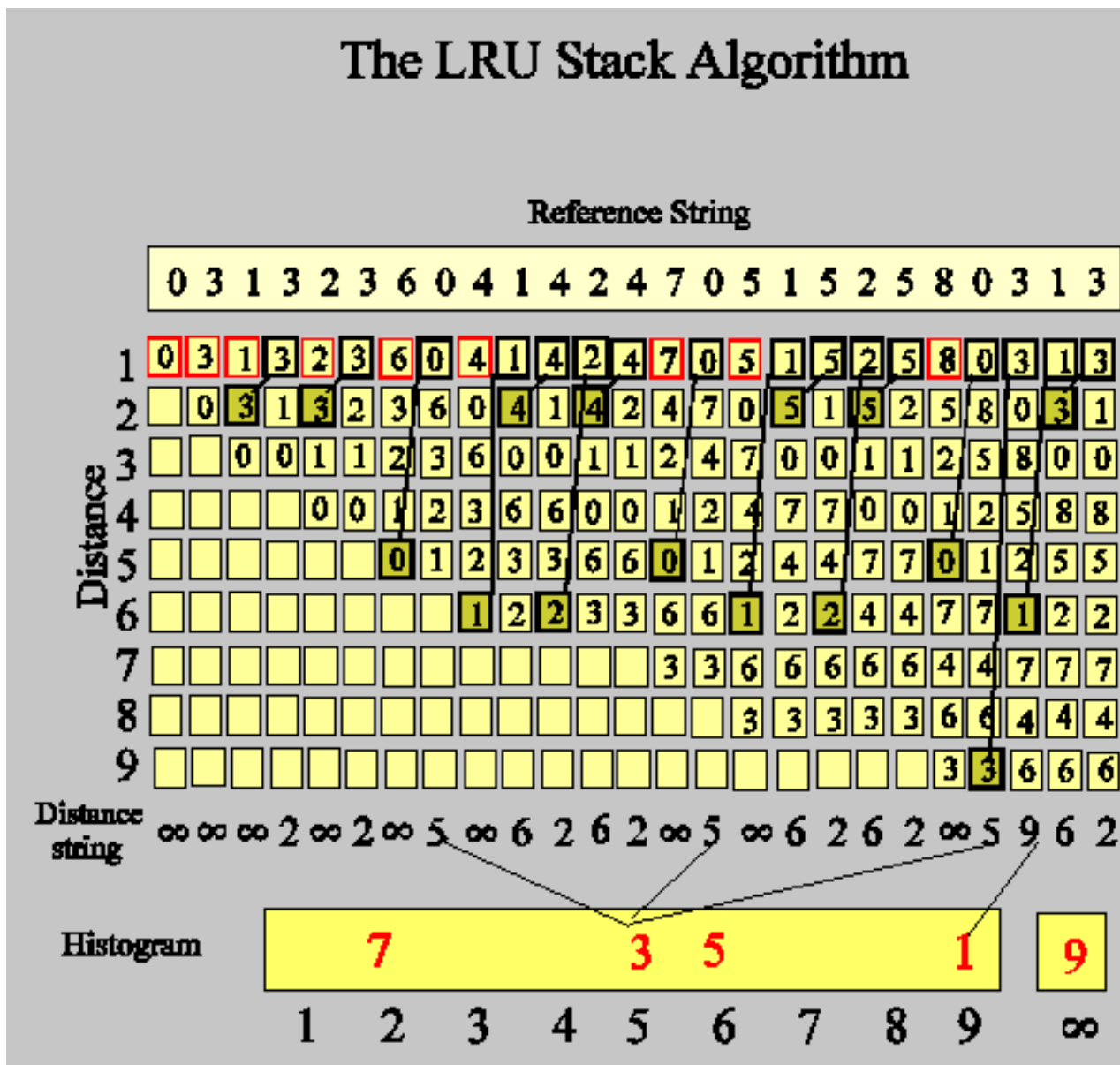
('x' means don't care)

知乎 @kkcqc



LRU stack算法：堆栈距离

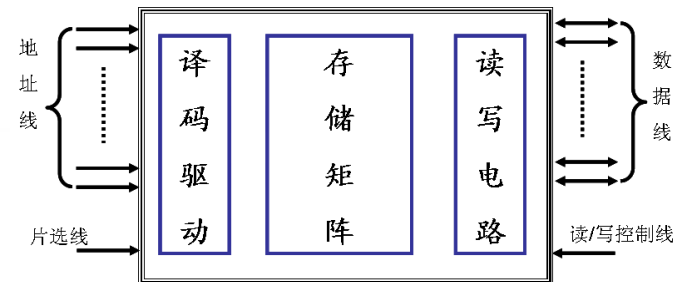
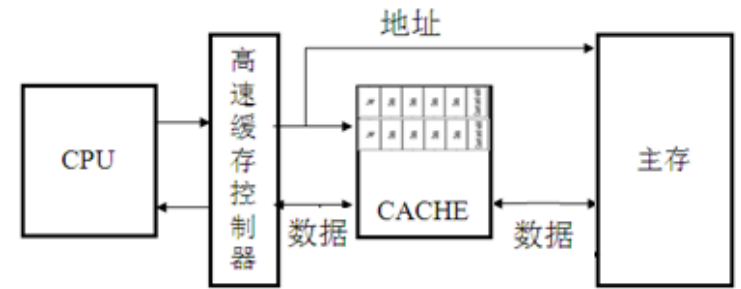
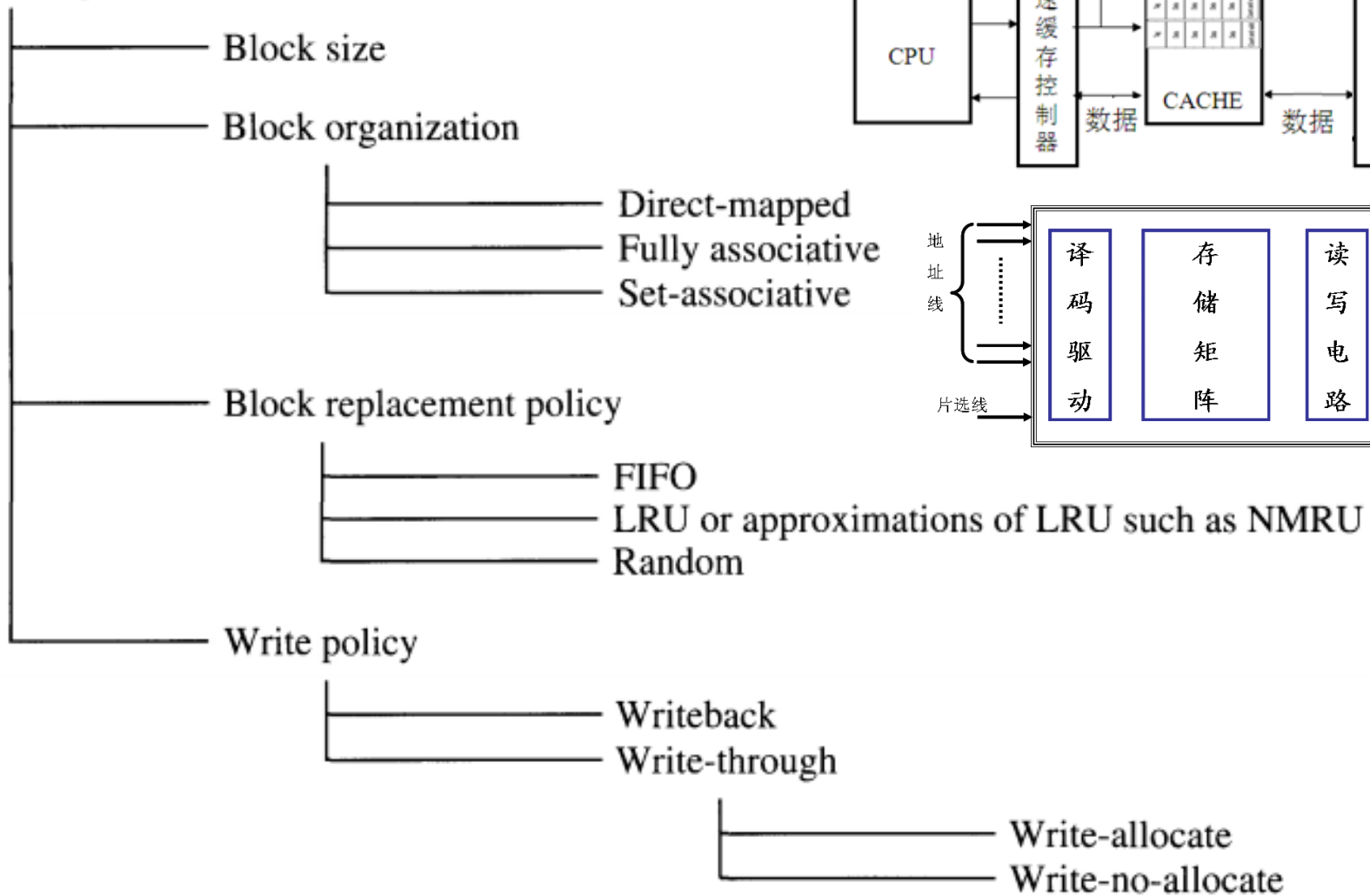
- stack distance
 - 访问某line时其在栈中的位置（深度）
 - 第一次访问时 = ∞
- 替换哪一行？
 - 双向链表
- histogram
 - 局部性



Cache Design: Parameters, PWR



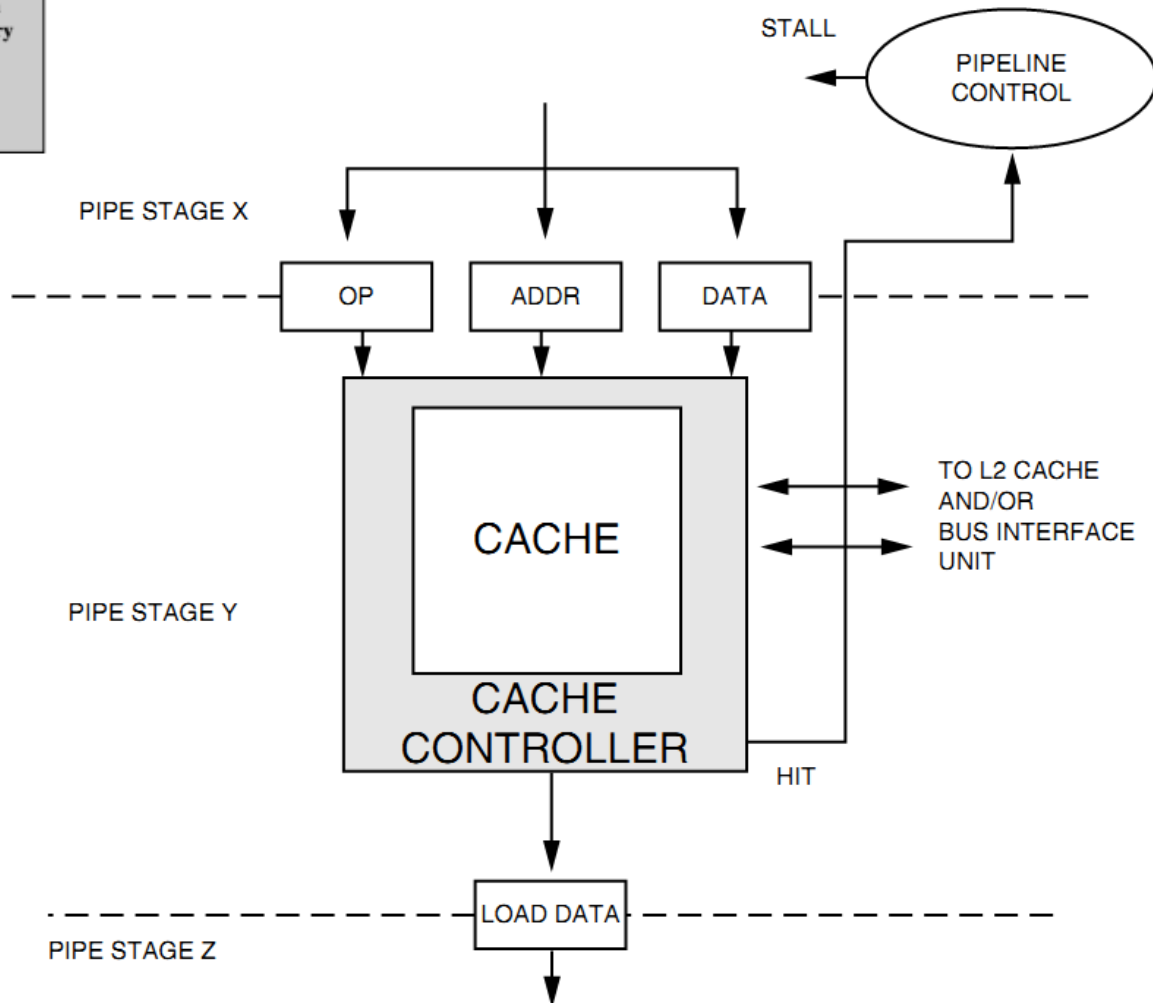
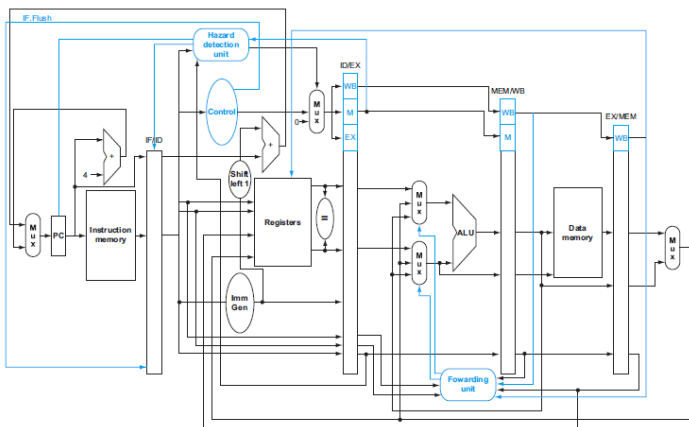
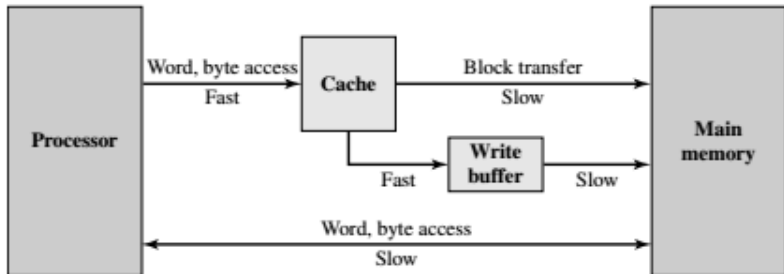
Cache design



Pipeline and L1 Cache Interface



ppi访存接口:
地址, 数据, 命令 (R/W), 状态 (ready/busy, err)



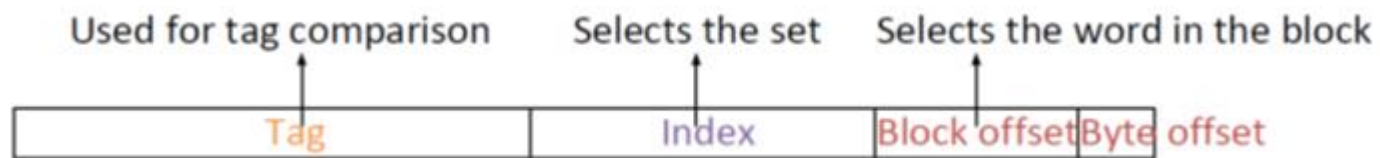
阻塞式blocking Cache
lw/sw的执行时间 (周期数) ?

Nonblocking (\$5.13)

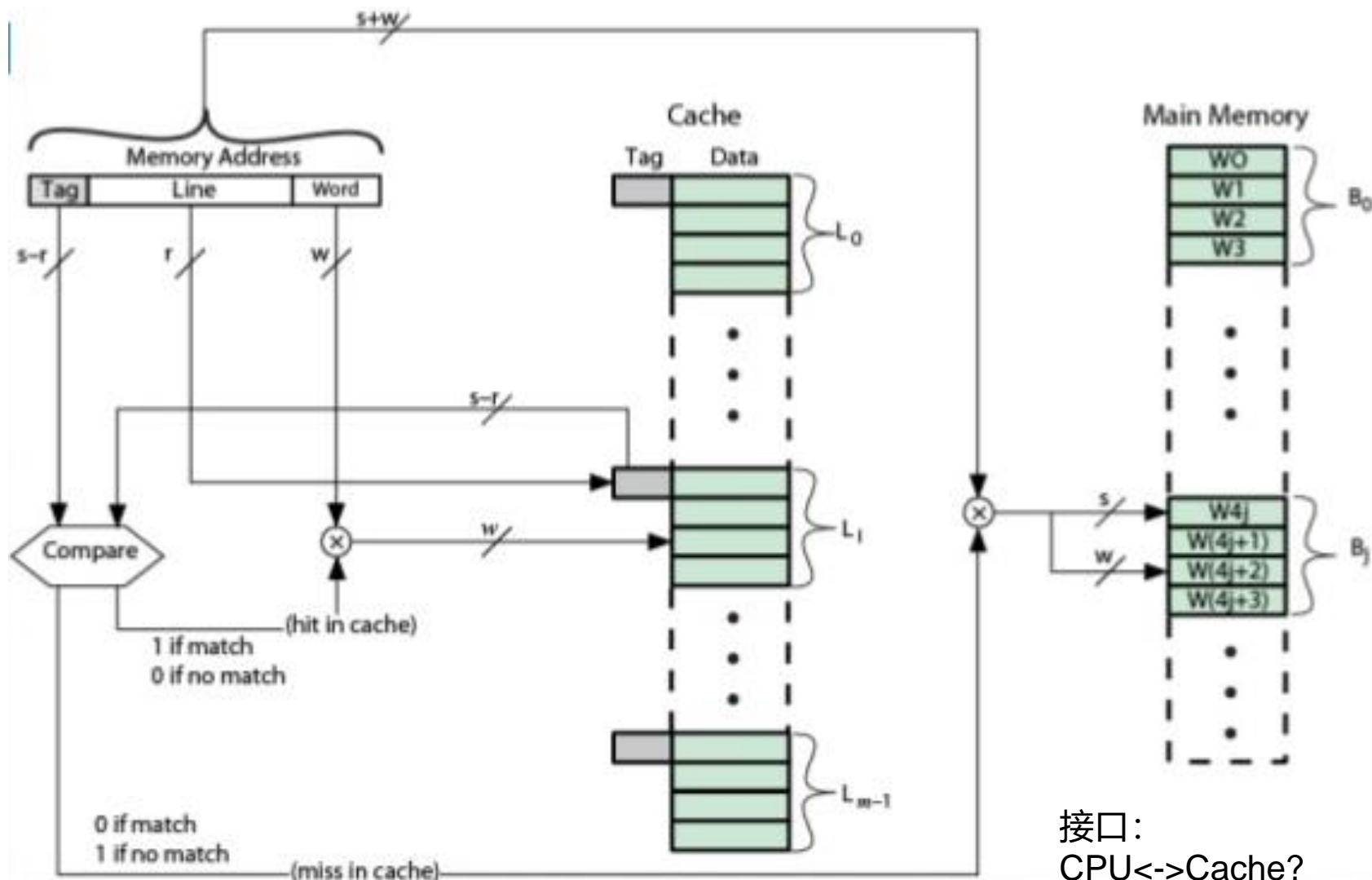
A Simple Cache: 需求规格, \$5.9, \$5.12



- 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



Direct-mapped cache行为



读写过程：命中，不命中？

接口：
CPU \leftrightarrow Cache?
Cache \leftrightarrow MM?
CPU \leftrightarrow MM?

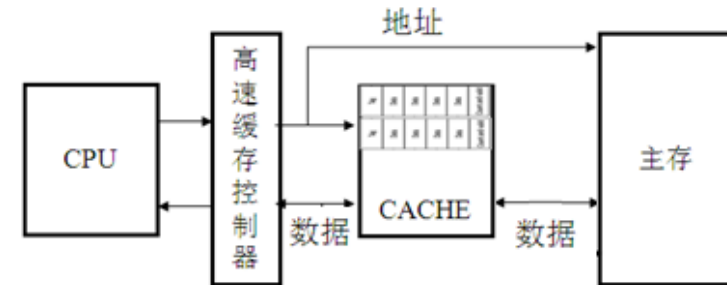
A Simple Cache: 概要设计之接口



- CPU <-> Cache (**Blocking** cache)
 - 1bit *Read* or *Write* signal
 - 1bit **Valid** signal, a cache operation? ——是否允许访问Cache?
 - 32bit address
 - **32**bit 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
- **128**bit data (块) from cache to memory
- 128bit data from memory to cache
- 1bit **Ready** signal, memory operation is complete



访存接口:

地址, 数据, 命令, 状态
hit信号?

- CPU<->MEM: 不直接访存?

- 命中**写回**, 不命中**写分配**。

Cache设计：数据通路 + 控制器

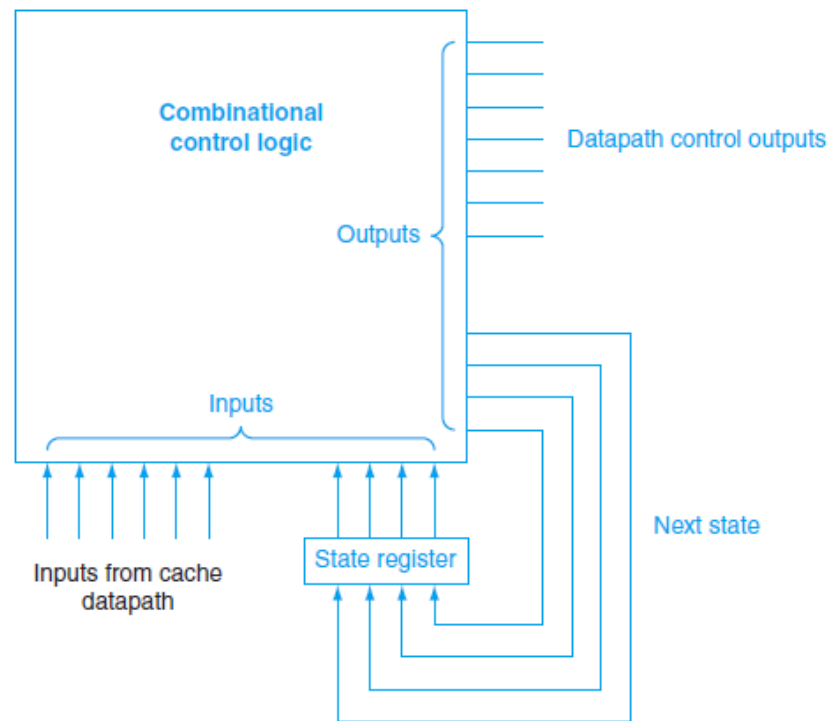
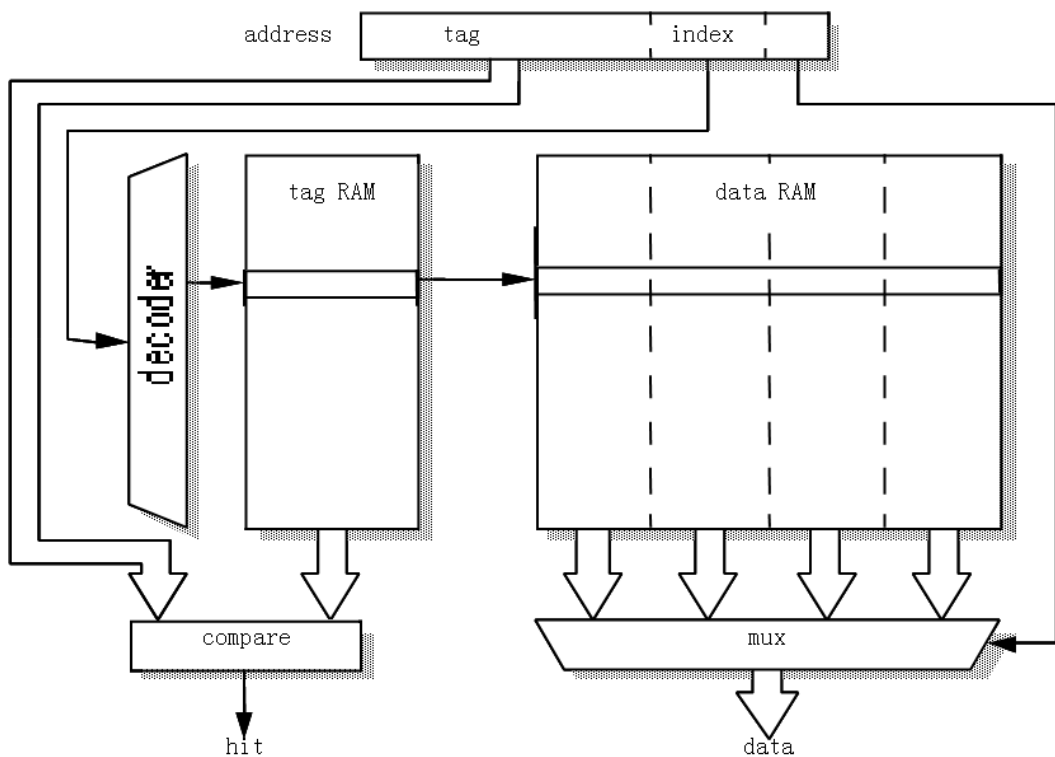
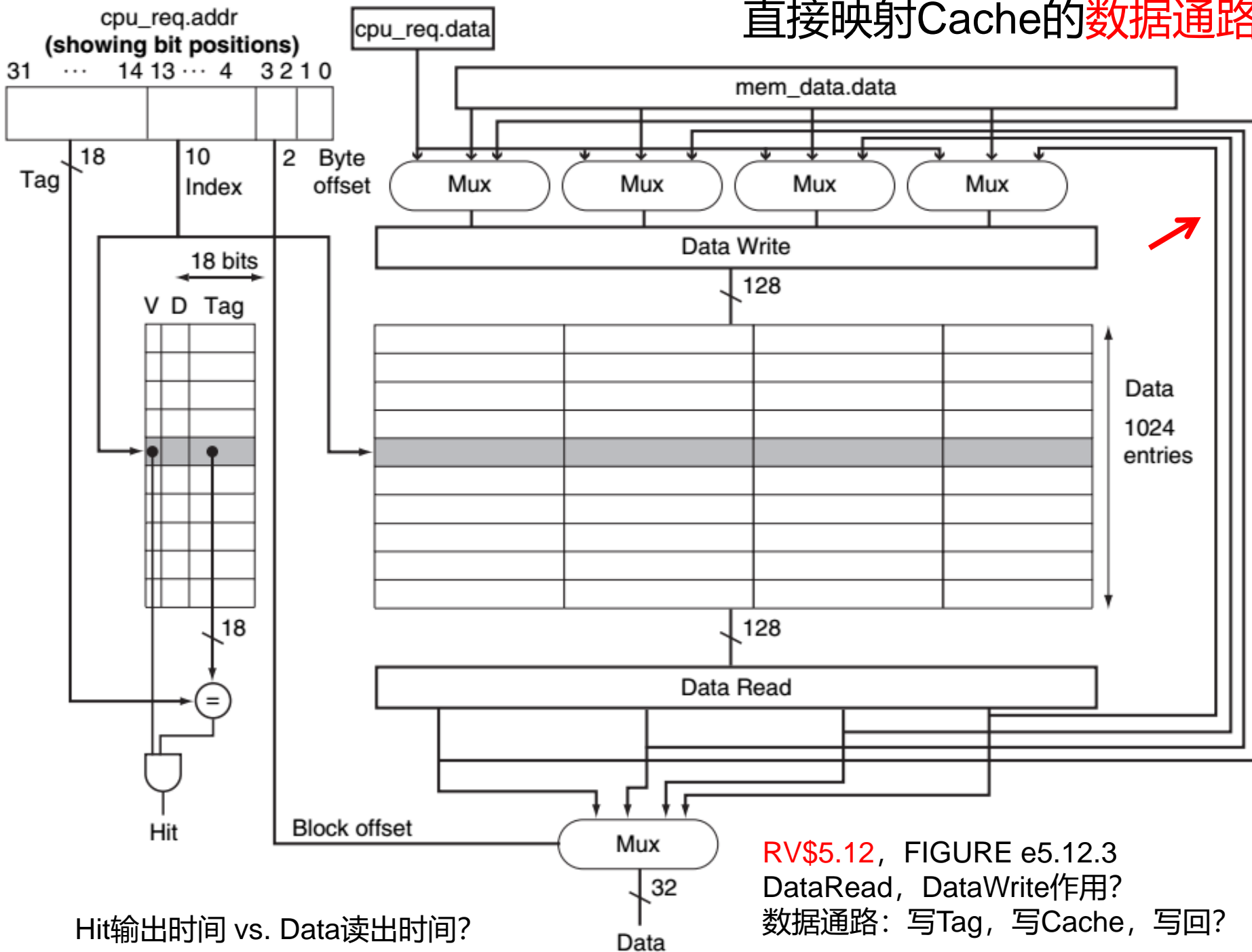


FIGURE 5.38

- Ixx四步法：分解，数据通路，控制器，综合
- 分立RAM：tag RAM，data RAM

直接映射Cache的数据通路



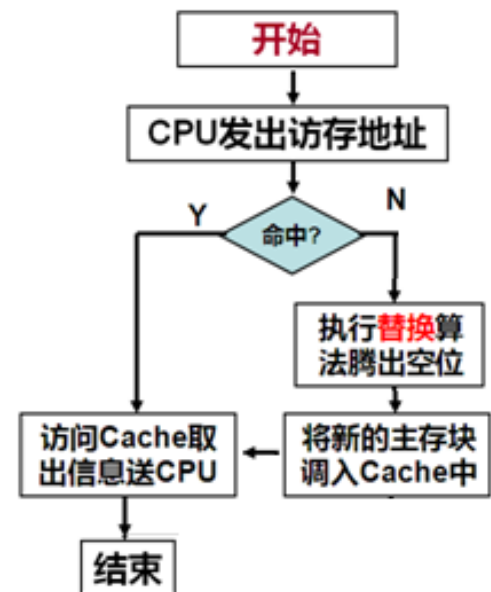
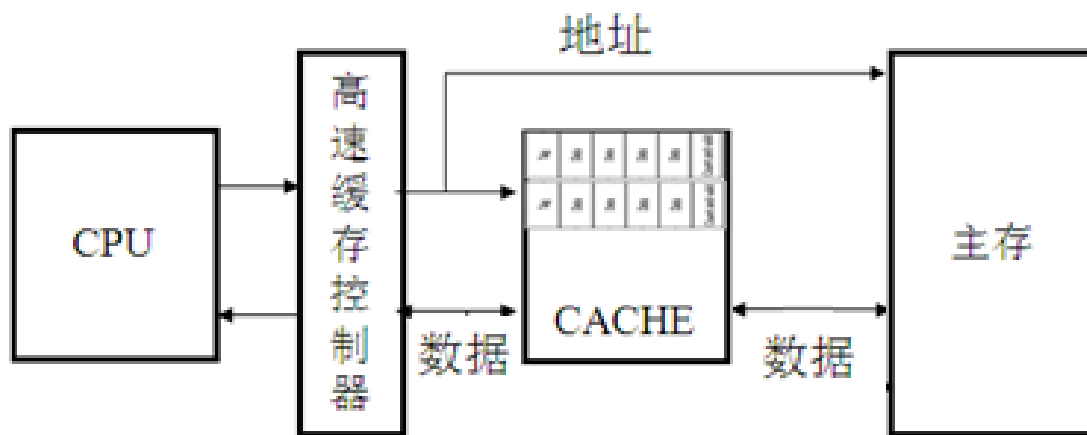
Hit输出时间 vs. Data读出时间?

RV\$5.12, FIGURE e5.12.3
 DataRead, DataWrite作用?
 数据通路: 写Tag, 写Cache, 写回?



直接映射Cache控制器

- Cache line位置唯一：读写miss时都要替换
- 时序：多周期，半同步
 - 阻塞式Cache：miss时CPU stall (1bit *Ready* signal)
 - CPU不直接访问MEM：写回写分配
 - 读
 - 命中：直接读入，一个cc (判断hit, 数据传输, 并行)
 - 不命中：换入 (clean, dirty写回MM, 读MM), 多个cc (写放大), CPU重读
 - 写：Writeback using write allocate
 - 命中：写cache, 置dirty位, 一个cc
 - 不命中：换入 (clean直接换入, dirty写回), 多个cc, CPU重写Cache



a Direct mapped Cache Controller, \$5.9.3



图5-39

• CompTag

- 命中：完成读写cache
 - 命中判断与数据读写并行
- Miss：写回写分配，多CC
 - 非脏：Allocate（写分配）
 - 脏：写回，再分配

• Cache Controller <-> MEM controller

- MEM Ready：换入换出完成的握手，半同步

• 与CPU同步：IF或MEM段阻塞，Hit

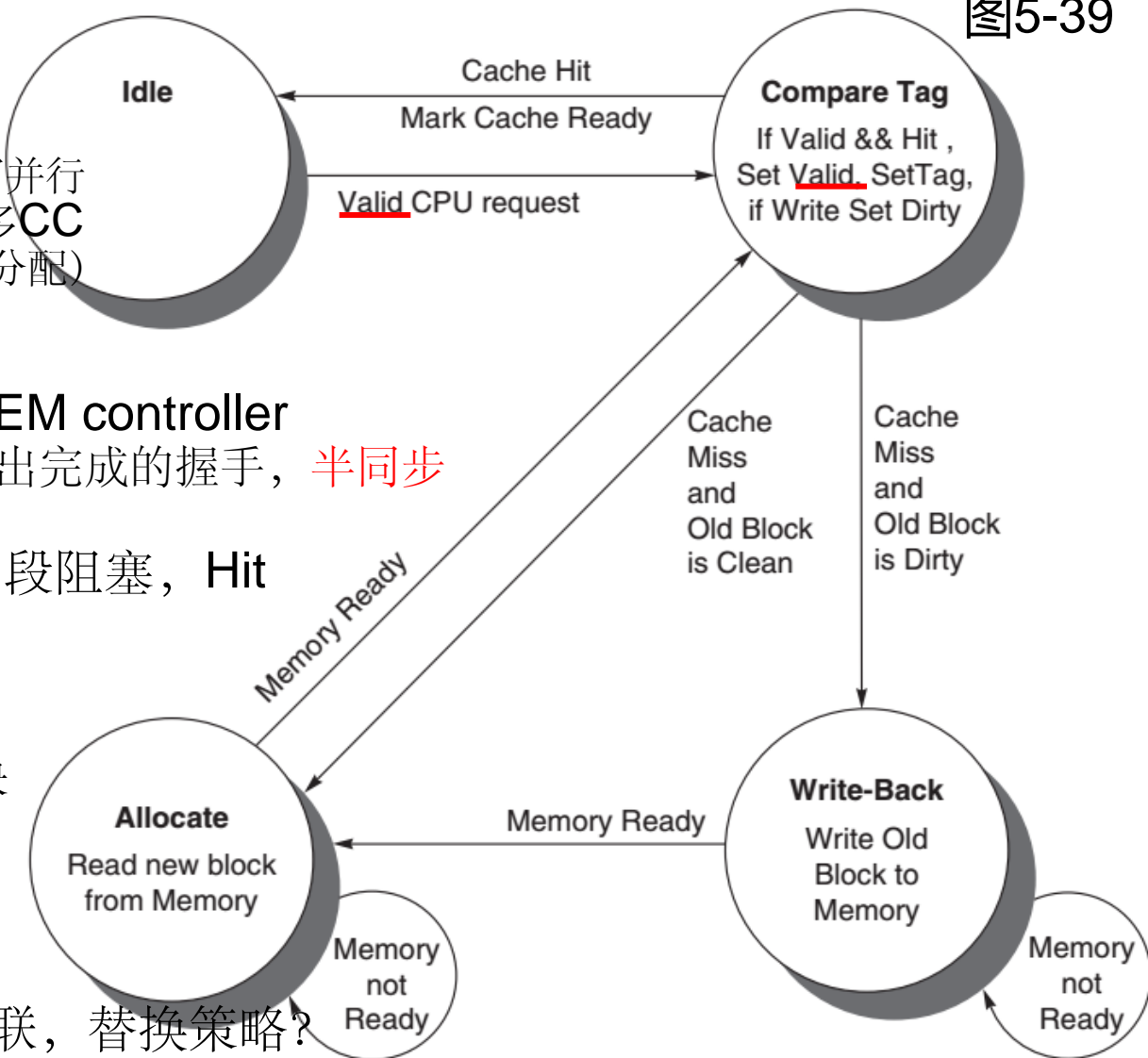
- 周期数=? 半同步

• 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？



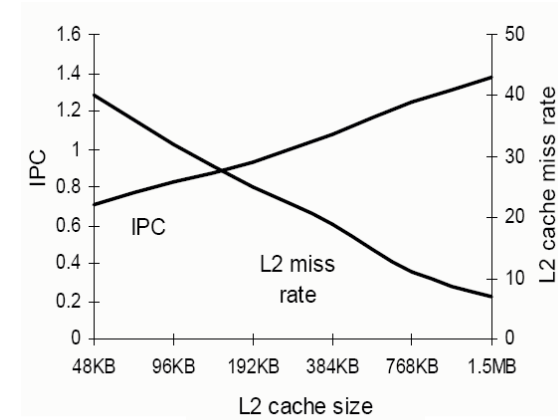
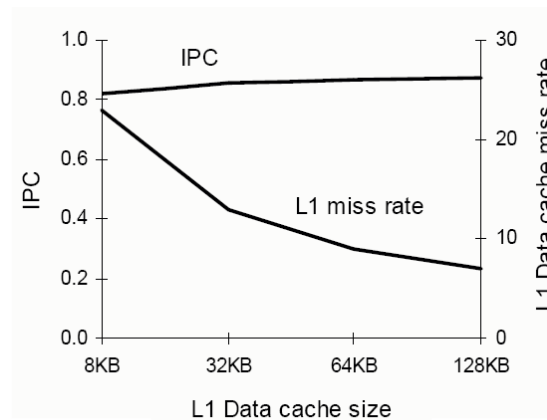
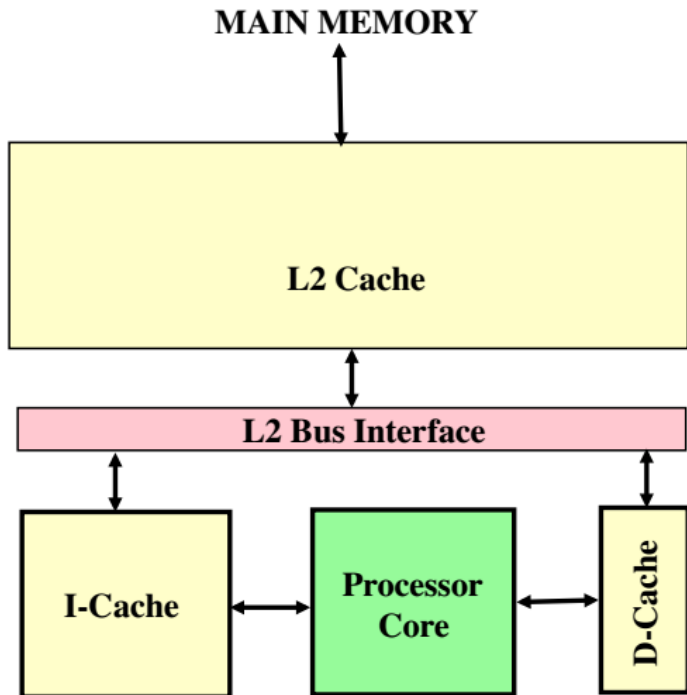
多级Cache: \$5.4.4例题翻译不清楚!



- 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

- $T_s = T_1 \times H + (1 - H) \times (T_1 + T_2)$

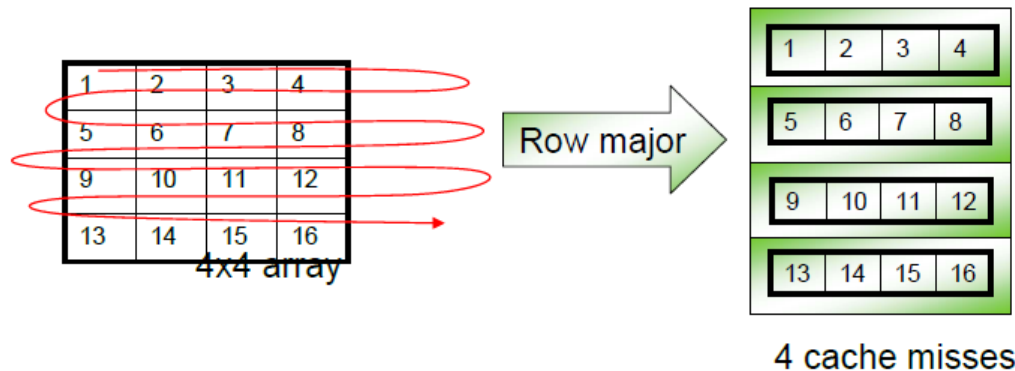
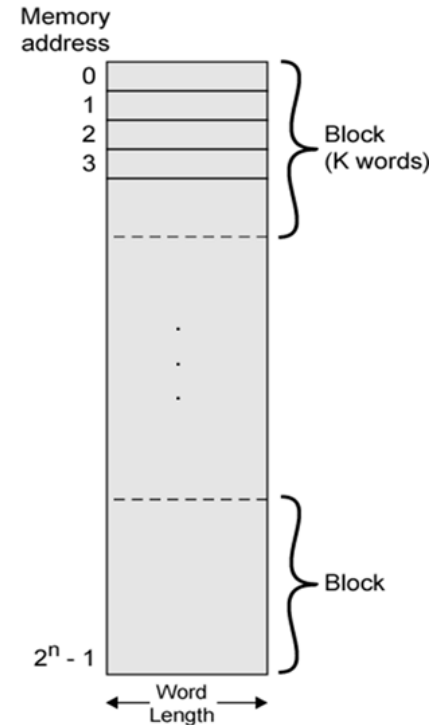


L1: 容量几乎对IPC没有影响?
L2: 命中率, 影响很大



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)

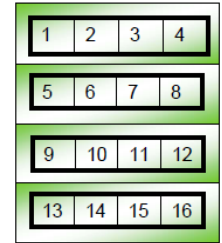
- 原程序 (column major)

```
• 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
  }
}
```



Row major

设Cache容量=1 entry



Cache blocks

4 cache misses

- 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]
  }
}
```

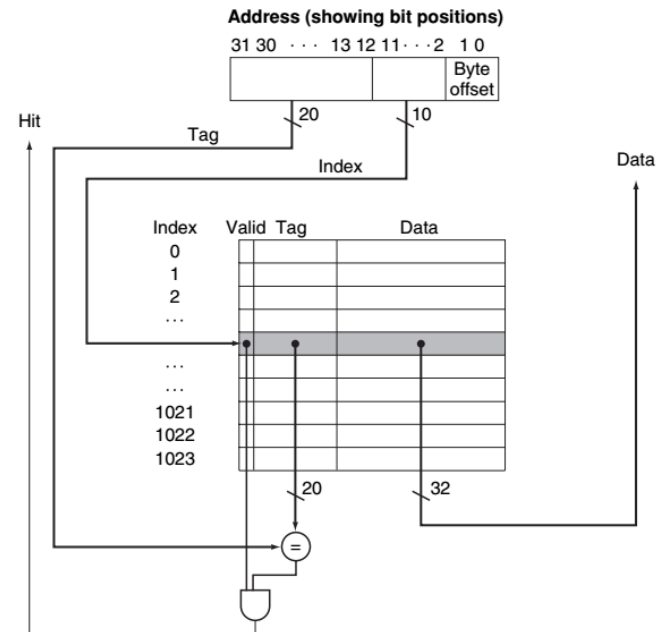
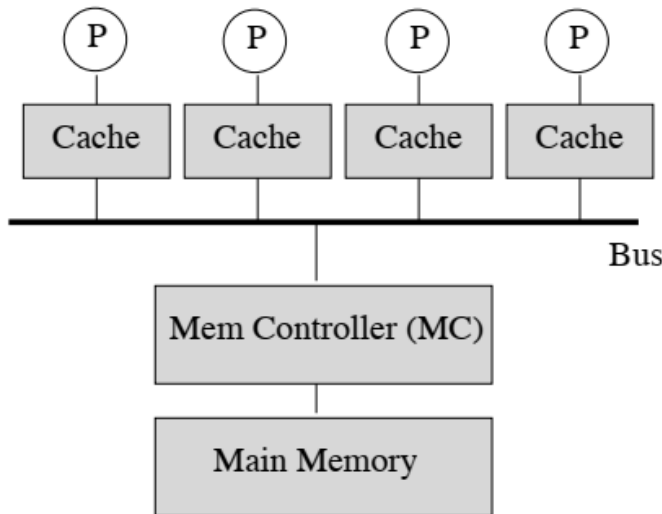
- 循环合并 (Loop fusion) : 多个循环体并入一个基本块
- 循环分块 (Blocking) : DGEMM算法 (\$5.4.5, \$5.15)
- Cache-oblivious algorithm: 与cache结构无关的算法

多处理器（多核）系统, \$5.10



- 程序员的程序行为“正确性”：顺序执行，读到最新值
 - 按**程序序**，对任一变量的**读**都能返回其**最新的**写值。
- 对称式共享存储多处理器（SMP, **集中式**）的统一存储器访问（UMA）
 - 统一存储器访问UMA：多处理器共享Main Memory，对主存的访存延迟相同
 - Memory consistency**模型**（一致性/连贯性/连续性）：高级语言C++，DBMS
 - 顺序一致性模型SC：**多个**操作对**多个**存储位置的**写读**操作顺序保持一致
 - 放松一致性模型RC：放松（Relaxed）顺序要求，允许OOO，如RVWMO
 - Cache coherence**协议**（数据一致性/同一性）：支持MCM，对程序员透明

A Bus-Based Multiprocessor System



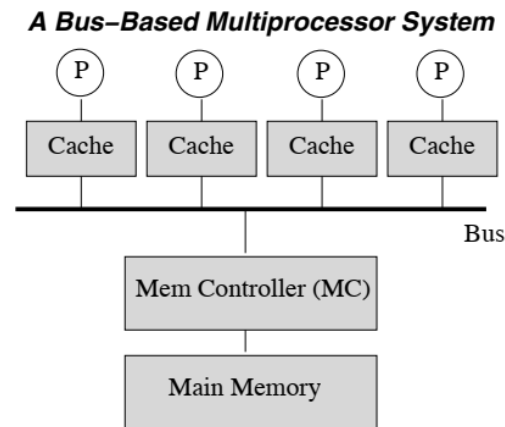
多处理器Cache coherence, \$5.10



- Cache一致要求
 - 读到最新值：自写自读（程序序），他写我读（一致性）
 - 写串行化：两个CPU对同一位置的两次写操作在所有CPU看来必须顺序相同
 - 例：P1先向MM的x写1，P2后向MM的x写2，所有处理器只能先读到1，再读到2
- Cache coherence协议CCP：读写操作执行过程
 - 保证读操作返回最新值：是否需要更新其他CPU的私有Cache（L1\$）？
 - 写无效策略：invalidation others, 通知=>ACK=>写
 - 写更新策略：同步更新所有Cache copy。需要Cache2Cache链路？
 - 写串行化：对总线结构，通过Cache竞争总线访问权，实现串行化

FIGURE 5.39, 设CPU A写透。
本地Cache和MEM中同一位置（地址X）的值不一致

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



Cache一致性实现, \$5.10, \$5.12



- CCP: VI、EI、MSI、MESI、MOESI、...
 - 跟踪每个copy的状态: Snooping bus式, Directory式
- 例: 监听一致性协议, Write **invalidation** protocol
 - 各Cache监听 (snooping) 总线上的写操作
 - 某Cache写入时, 使其他Cache中的副本无效

A Bus-Based Multiprocessor System

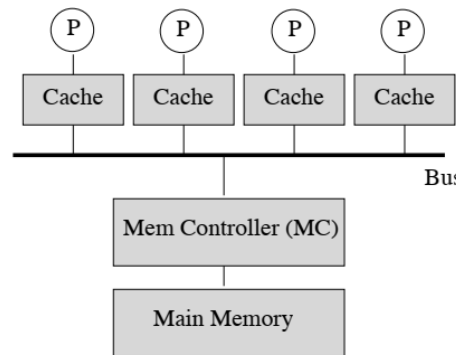


FIGURE 5.40, 写回, 使无效 (第4行cpuA写时, cpuB为“空白”), 一致

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	无效	0 未写回
CPU B reads X	Cache miss for X	1	1	1 已写回

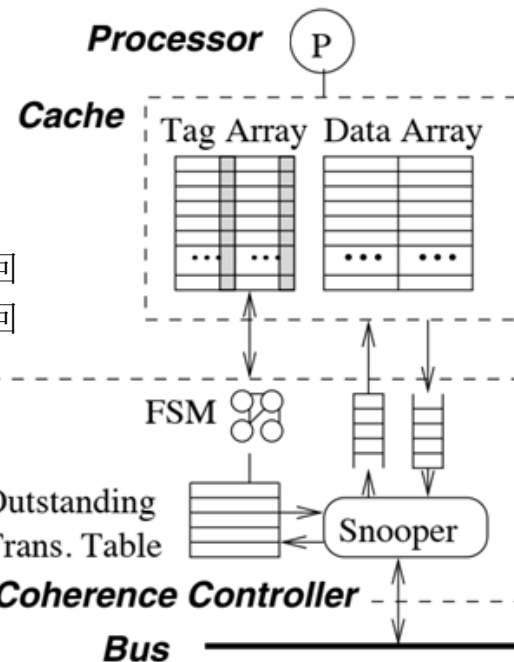


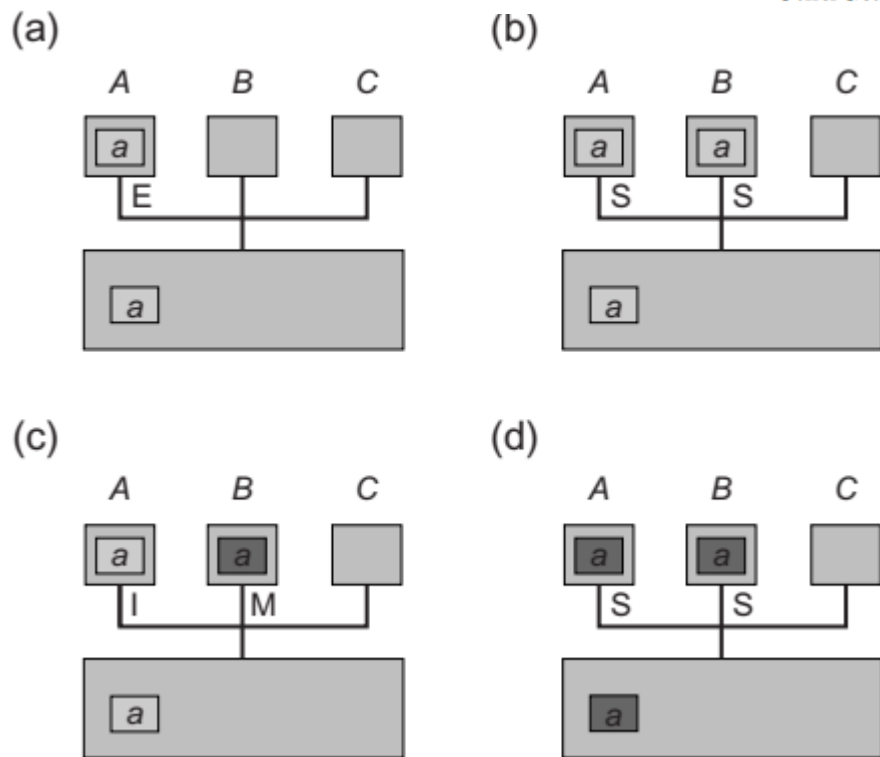
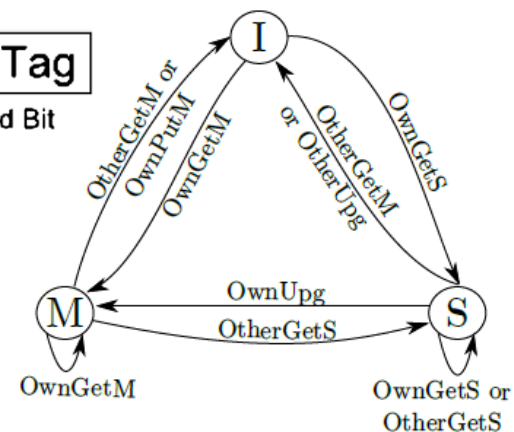
FIGURE 5.39, 写直达, 不一致

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

MESI cache coherence protocol



- MSI: \$5.12
- MESI: 减少总线通信
 - 图a: A读a
 - E: 独占, 块未修改。
 - 可本CPU读写。
 - 状态迁移: 发现其他CPU访问时须将E改为S
 - 图b: B读a
 - S: 共享。块未修改, 其他CPU可以将a装入其Cache
 - 图c: B写a
 - 状态迁移: I和M
 - 图d: A再次读a
 - B将a直接发给A! ?
 - 状态迁移: S

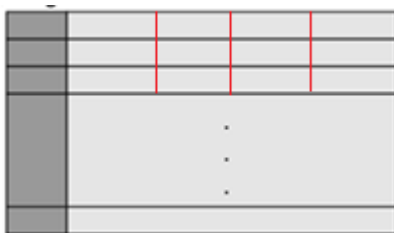


Cache伪共享false sharing, \$5.10详解

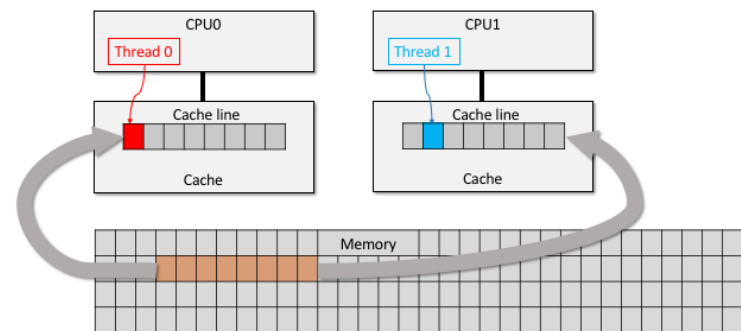


- Cache一致性协议通常以Cacheline为粒度维护数据一致性
 - 某些协议的粒度更细或更粗
- 伪共享：处理器各自处理不同的变量，但它们在**同一**Cache line中
 - 导致不必要的一致性维护开销
- 软硬件接口：程序员或编译器避免此问题

一个Cache block (4个data)



CPU Cache Line

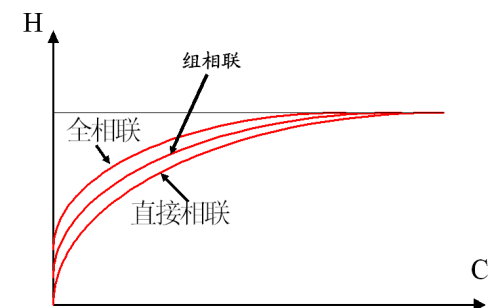
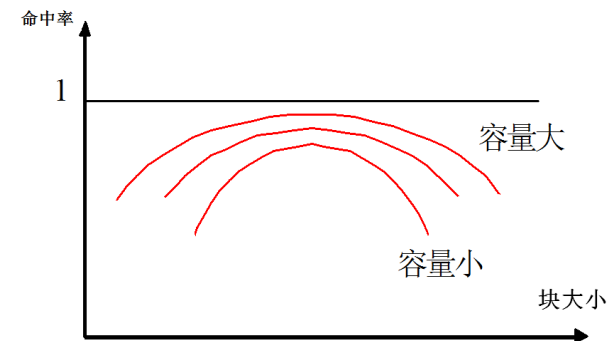


Note: 8 bytes per tile

小结：4种Cache Misses, (\$5.8.5 “3C”)



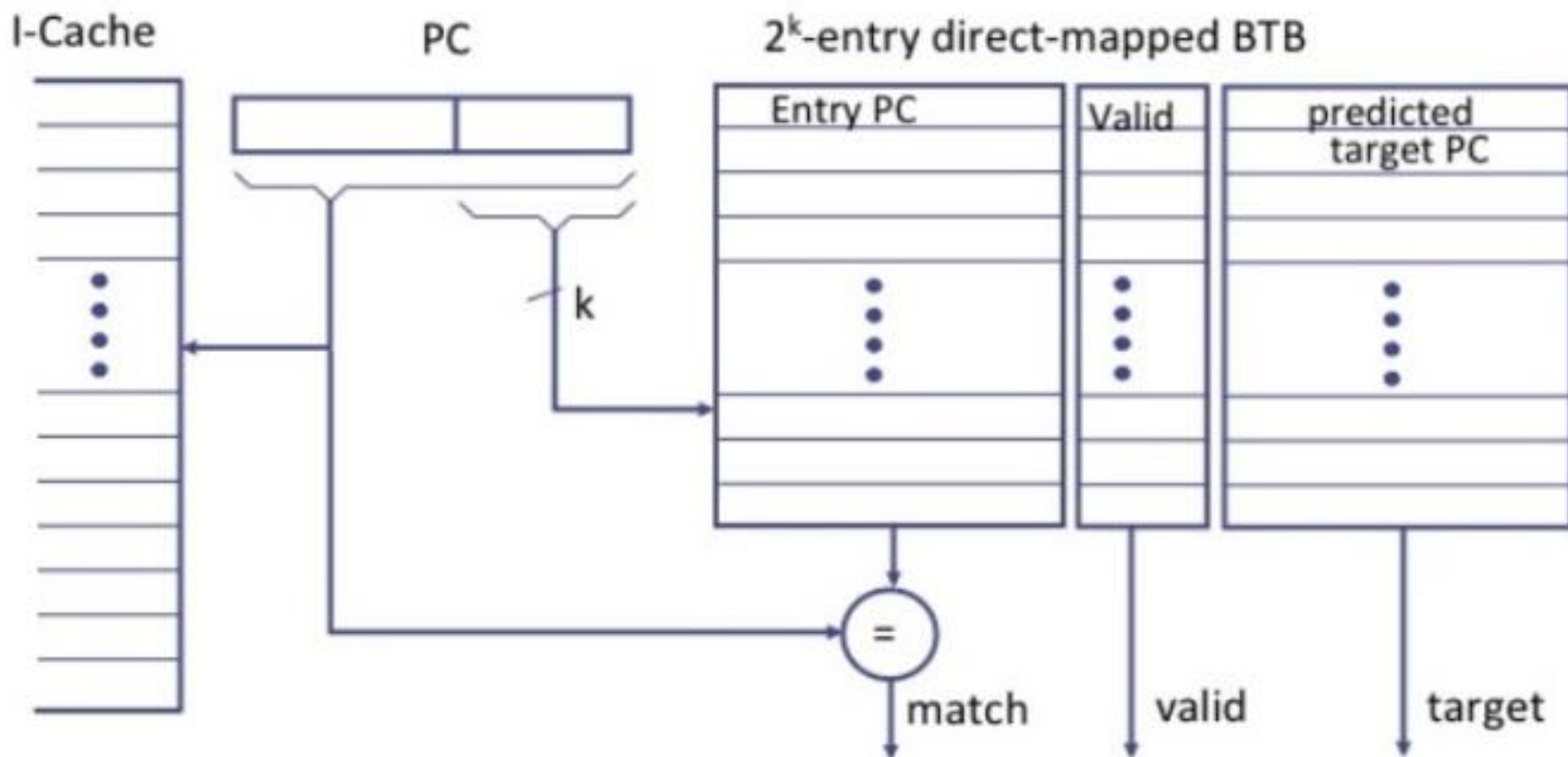
- **Compulsory** (必然)
 - cold start、first reference、process migration
 - “Cold” fact of life: not a whole lot you can do about it
 - Note: If you are going to run “billions” of instruction, Compulsory Misses are **insignificant**
- **Capacity**: cache大小, *block*大小
 - Cache cannot contain all blocks
 - 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*



直接映射Cache与分支预测缓冲



- BTB中记录了每个PC对应的nPC
- 取指时根据PC查找BTB取下一条指令
 - 一旦预测错误，将用正确的nPC更新BTB中的相应记录



图见《Computer Architecture-A Constructive Approach》，2012

小结

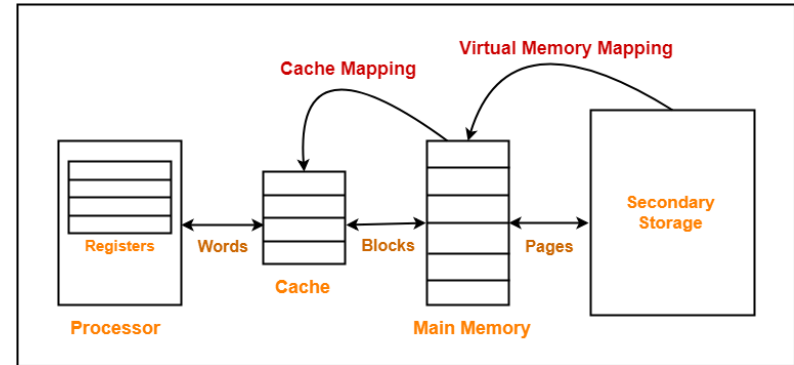
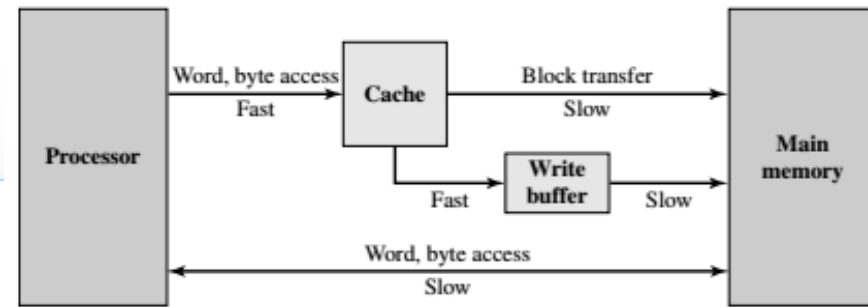
- Cache有效性

- 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