第五章 存储系统

.

※本章主要内容

(1) 存储系统的层次结构　　—总体结构

概述(组成/性能)，组织(层数/层次管理/MEM结构)

(2) Cache的基本知识　　—层次MEM结构

工作原理，实现技术，性能分析

(3) Cache的性能优化

降低缺失率，减少缺失开销，减少命中时间

(4) 主存的性能优化

单体多字MEM、多体交叉MEM、并行存储器

(5) 虚拟存储器　　—面向软件的存储管理

组织(原理/管理/实现/性能优化)，保护(区域/访问)，层次结构综合

※总体要求

掌握层次结构MEM组成原理，理解性能优化技术

第1节　存储系统的层次结构

※主要内容：层次结构的组成、性能、组织(层数/管理)

一、层次结构概述

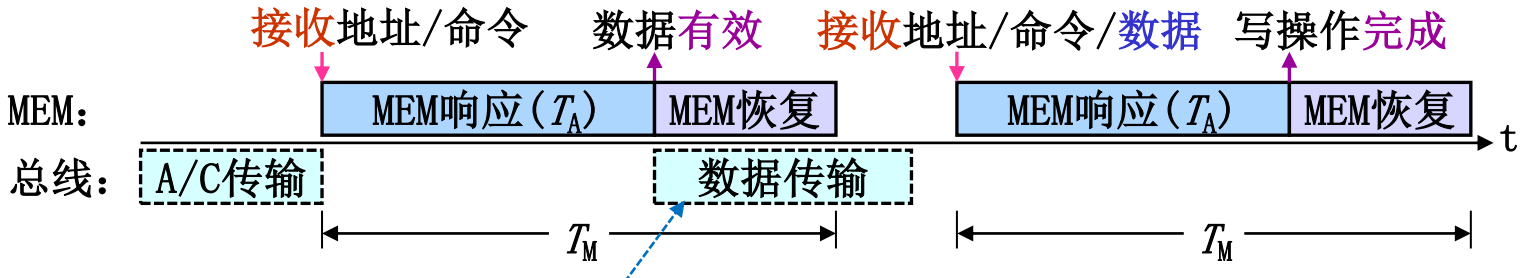
1、存储器的性能指标

*存储容量：$S_M = w \cdot l \cdot m$，w、l、m为存储体的字长、字数、个数

*存取速度：访问时间(T_A)—完成操作的时间　←从接收命令开始

存取周期(T_M)—两次操作的最短间隔时间

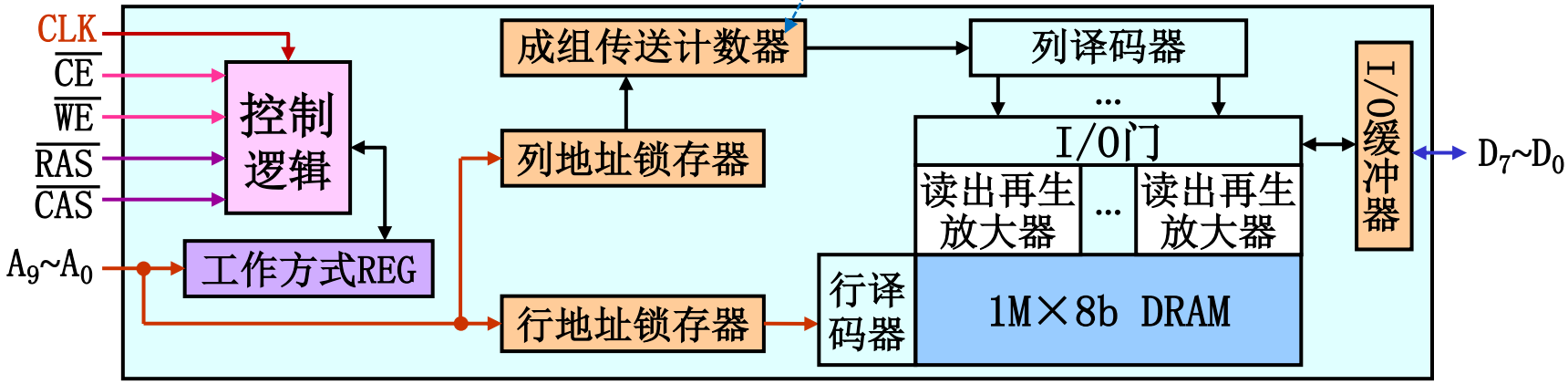
$T_M = T_A + T_{恢复}$，如DRAM再生/刷新∈$T_{恢复}$



*传输速度：MEM带宽(B_M)—最大数据传输率(2个数据的最小传输间隔)

如：m个体顺序编址时$B_M = w/T_M$，交叉编址时$B_M = w/(T_M/m)$

例：某SDRAM的CLK＝200MHz、支持突发传输，DRAM阵列访问时延（含译码器+读出再生放大器+I/O门）$T_{阵列访问}$＝4CLK、再生时延$T_{再生}$＝2CLK、I/O的锁存/缓冲时延 T_{IO}＝1CLK，SDRAM刷新采用异步刷新方式、通过外部操作实现，分析该SDRAM的性能。



S＝8b×1M×1＝1MB；　←有8个位面，每个位面1K行×1K列

T_A＝ $T_{行地址锁存}$+ $T_{列地址锁存}$+ $T_{阵列访问}$+T_{IO}＝1+1+4+1＝7CLK；

T_M＝ T_A+($T_{再生}$−T_{IO})＝7+(2−1)＝8CLK；　←再生与I/O同时进行

B_M＝ 1B/T_{IO}＝1B/(1/200MHz)＝200MBps　←每个CLK可传输1B

2、层次结构的引入

*MEM的用户需求： 大容量、高速度、低价格

*程序访问的局部性原理：

程序运行时，指令和数据访问所呈现出的相对簇聚现象

示例：for (i=0; i<n; i++) S=S+A[i];

时间局部性— 最近访问过的信息，将再次被访问

空间局部性— 最近访问信息的相邻信息，将很快被访问

*MEM的用户需求解决方案：

MEM组成— 层次结构的 快速MEM ＋ 慢速MEM

利用局部性

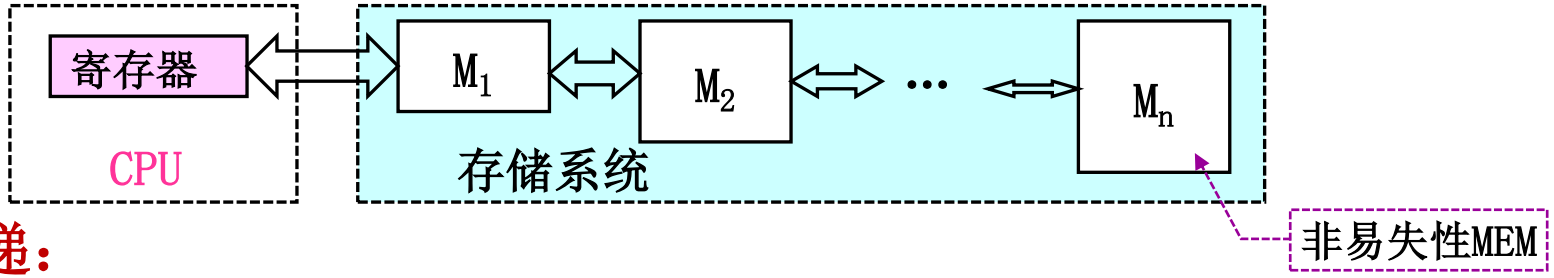
速度保证— 近期常用数据 近期未用数据

容量保证— 容量小 容量大 ＝ 容量大

价格保证— 占比小 ＋ 占比大 ＝ 价格低

3、存储系统的层次结构

*结构与组成：多种MEM级联，按速度分层、协调工作



*信息传递：

信息存储— 上级MEM中信息为下级MEM中信息的副本

上级改过后需写回下级←┘

信息传递— 外部只访问M_1，内部各级MEM间透明地传递信息

└← T_A不固定←————外部不可见←┘

*预期目标：①平均价格$c \approx c_n$

②平均访问时间$T_A \approx T_{A1}$

注—目标①和②实现后，容量S与T_A及c间已无矛盾

4、存储系统的性能参数

*存储容量：有效容量$S=S_{Mn}$ ←上层中内容为下层的子集

*每位平均价格(c)：

$$c=(c_1S_{M1}+c_2S_{M2}+\cdots+c_nS_{Mn})/(S_{M1}+S_{M2}+\cdots+S_{Mn})$$

或$$c=(c_1S_{M1}+c_2S_{M2}+\cdots+c_nS_{Mn})/S_{Mn}$$

*命中率(H)： $H=N_1/(N_1+N_2+\cdots+N_n)$，$N_i$为$M_i$中访问到的次数

缺失率(F)— $F=1-H$

*平均访问时间(T_A)：

层次结构所需(只访问M1)

$$T_A=H\cdot T_1+(1-H)(T_{A2}+T_1)=T_1+(1-H)T_{A2}$$

$$=T_{命中}+F\cdot T_{缺失}$$

←$T_{缺失}$即缺失引起的停顿时间(含替换)

*层次结构的实现要求：

远小于

目标①要求— $S_{M1}<<S_{M2}<<\cdots<<S_{Mn}$，$T_{M1}<<T_{M2}<<\cdots<<T_{Mn}$

目标②要求— H 较大 →$T_{缺失}$影响变小→┘

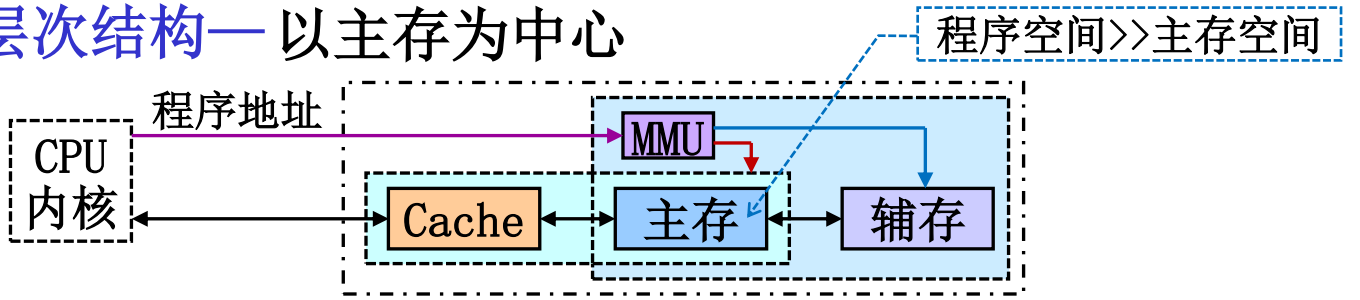
二、层次结构组织

1、层次数量

需求—程序存放在辅存中，执行前装入主存，执行时按程序地址访问

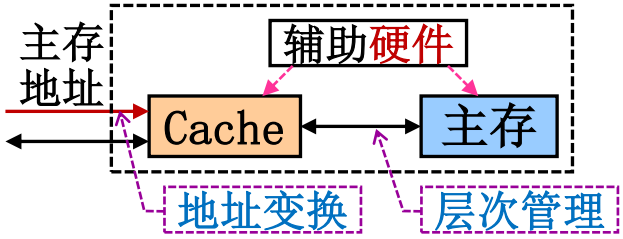
CPU只直接访问主存←┴←——┘ （先变换成主存地址）

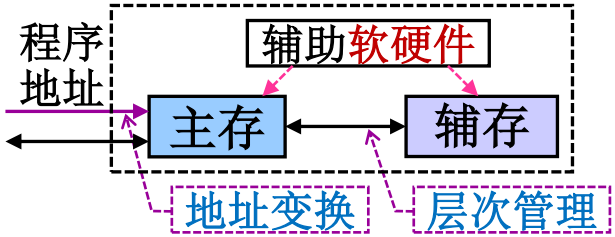
常见层次结构—以主存为中心



*Cache-主存层次： （解决主存速度问题）

实现要求—层次管理、地址变换全部由硬件实现





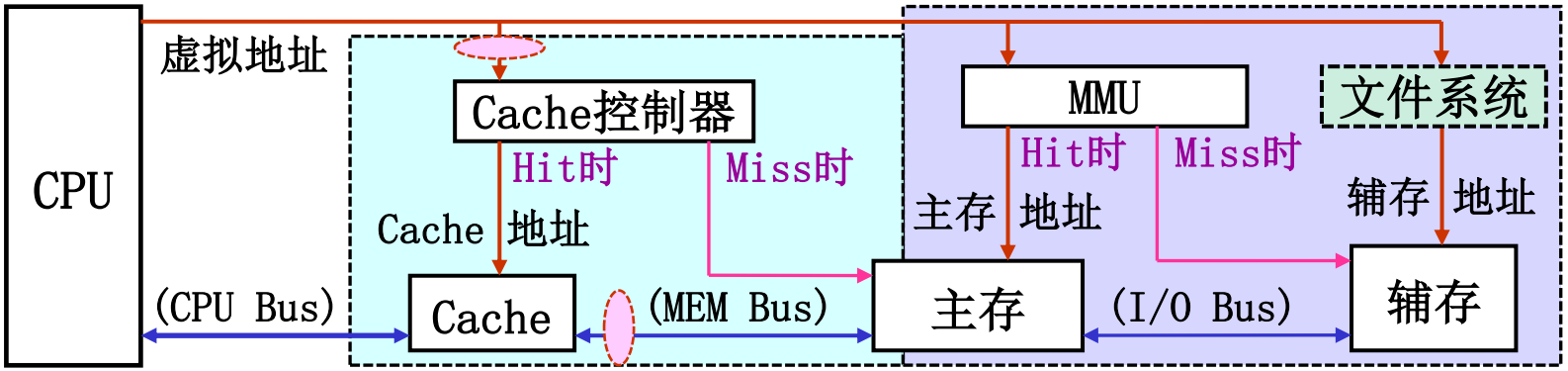
*主存-辅存层次： （解决主存容量问题）

实现要求—层次管理由软件实现，地址变换由硬件实现

OS→┴→常为虚拟存储器 └←MMU

2、层次管理的组织　　—即辅助软硬件

*目标：确定层次管理的参数/实现方式



*涉及内容：

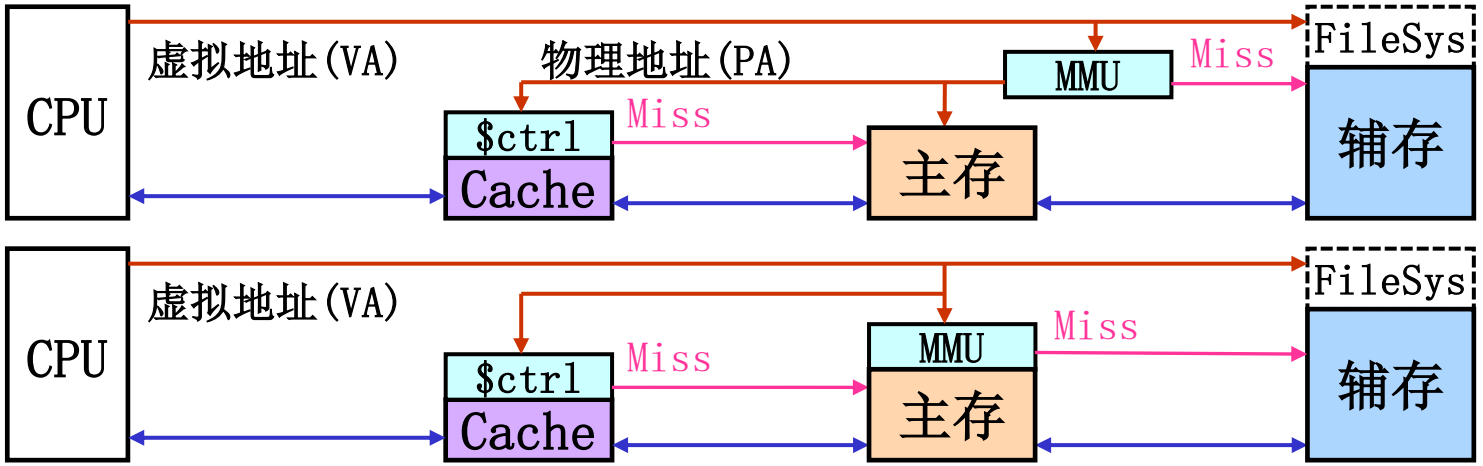
访问地址类型(虚/实地址)、层间信息交换单位(数据大小)，

层次管理实现方式(软件/硬件)

└←地址变换的实现方式均为硬件

*访问地址类型：

类型—主存-辅存为虚地址，Cache-主存可为实地址或虚地址



虚拟Cache：减少了1次地址变换(相对于物理Cache)；

实现困难(进程切换需清空$、数据共享无法实现)

[VA中含PID可解决] [无法2个VA→1个PA]

组织结果—Cache常为物理地址，L1$可为虚地址(稍后讨论)

访问与地址变换部分并行→┴→优化性能

△说明：后续讨论中，Cache均基于物理地址方式组织！

*层间信息交换单位：

依据— ①每次交换多个字，可减少 T_A ←H↑、$T_{调入}$↓

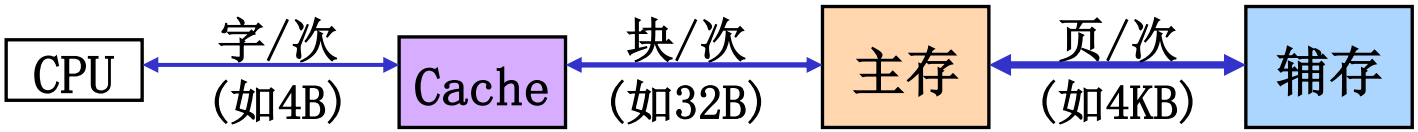
突发传送时 $T_{n个字} = T_{地址} + T_{存取} + n * T_{传输}$ →┘

思考①：常规传送时，n=8与n=2的$F * T_{调入}$哪个大？ $1/a*8T_{1字} \sim \lceil a/2 \rceil / a*2T_{1字}$，$a \leqslant 8$

②不同层间交换的平均速度相近，可消除瓶颈

└←$(F_{Mi} \cdot T_{n1个字})/n1 \approx (F_{Mj} \cdot T_{n2个字})/n2$

组织结果— MEM离CPU越远，n越大；n通过量化分析得到



*层次管理实现方式：

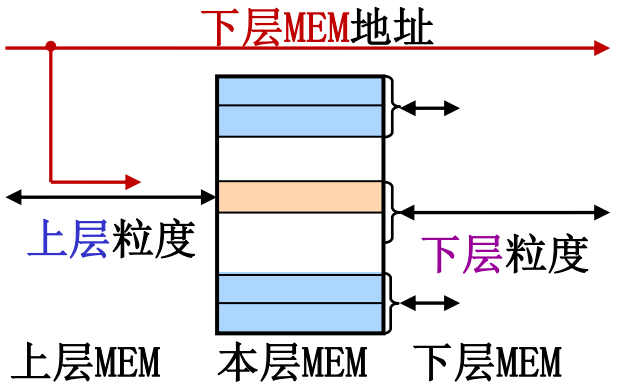
依据—满足性能、性/价的要求 思考②：=(2^32/2^12)*4B=4MB

思考②：主存按字节编址、32位地址，4KB/页，4B/页表项，进程页表大小？

组织结果—Cache-主存用硬件实现，主存-辅存用软件实现

目标是速度→┘ 管理表太大→┴←兼顾成本

3、层次MEM的结构

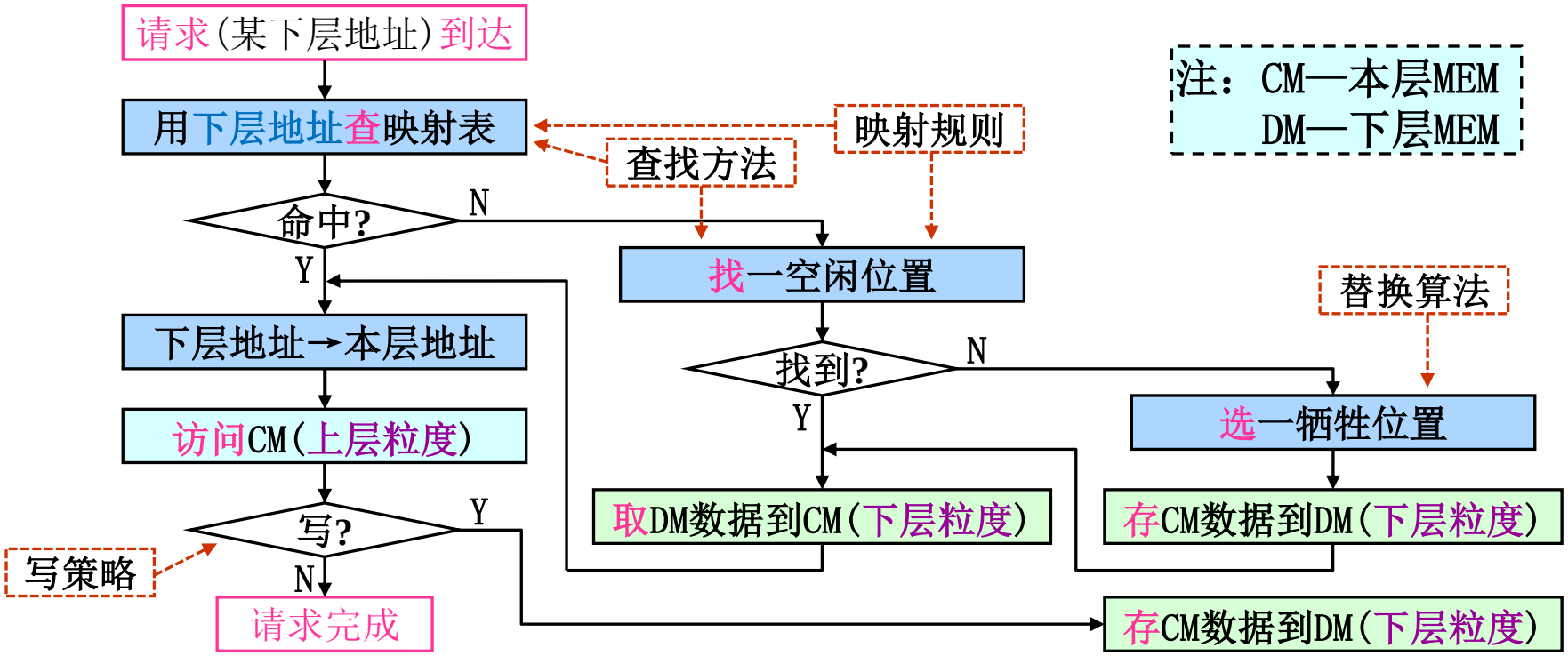


*MEM空间管理：

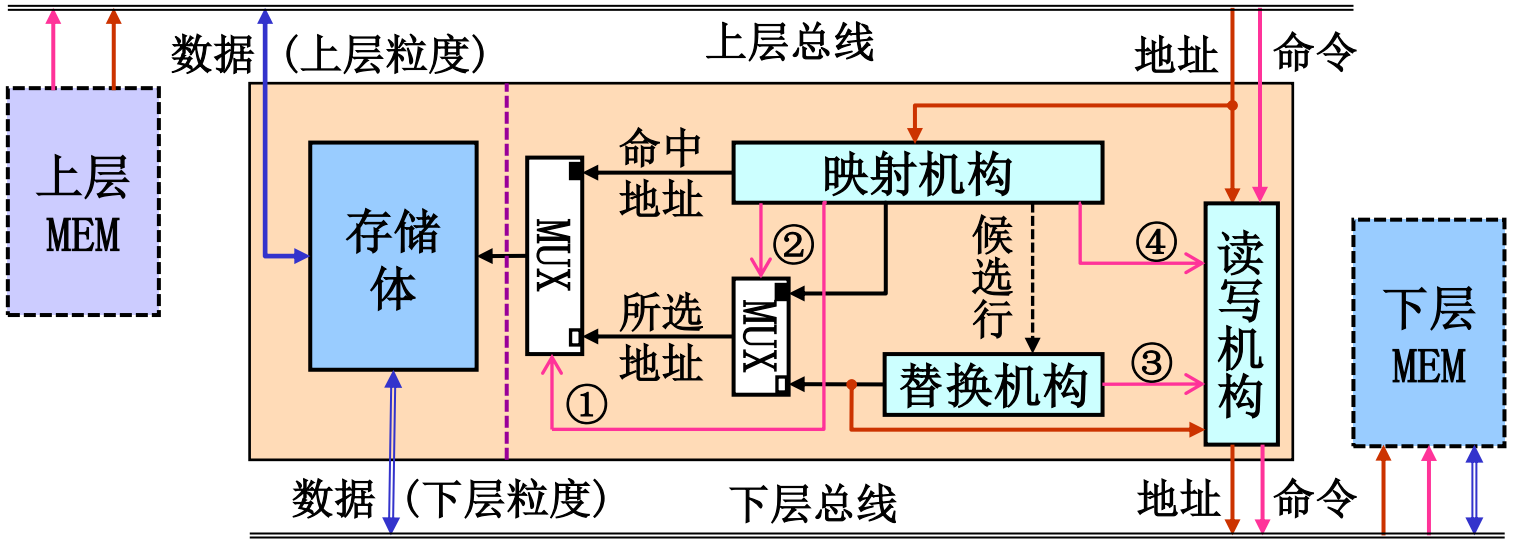
编址单位— 上层MEM的最小访问粒度

交换粒度— 两种(面向上层、下层)

*工作原理： 地址变换(缺失时先层间管理)、数据访问、数据写回



*硬件结构： 存储体、控制器(地址变换＋层次管理)



注：①是/否命中 ②是/否有空位置 ③是/否需写回 ④是否需调入

*性能优化：

$T_A = T_{命中} + F \cdot T_{缺失}$，$T_{命中} = T_{查找} + T_{访问}$，$T_{缺失} \geqslant T_{调入}$，

优化可从3个方面进行(Cache中讨论)

第2节　Cache的基本知识

※主要内容：工作原理，实现技术，性能分析

一、基本工作原理

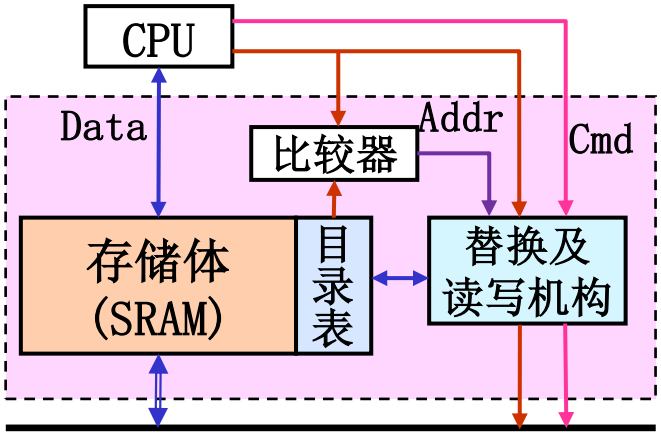
1、Cache的基本组成

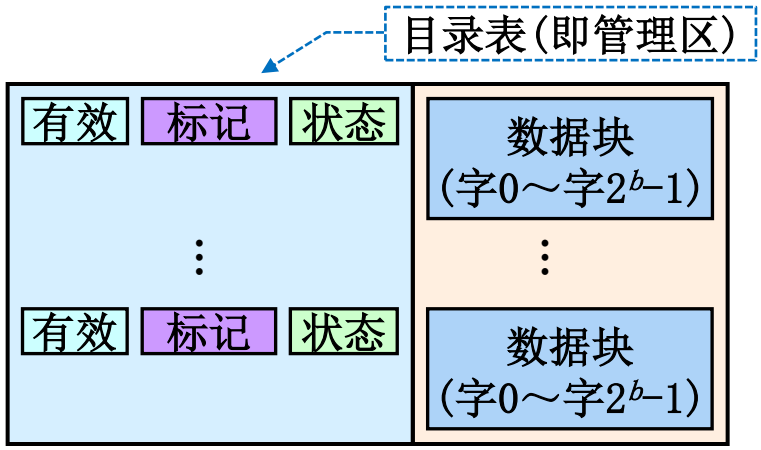
*存储空间管理：

信息交换单位—Cache-CPU间为字、Cache-主存间为块

信息交换管理— 目录表(Cache-主存的映射表)　　←即Cache管理区

*组成： 存储体、控制器(目录表+比较器+替换机构+读写机构)



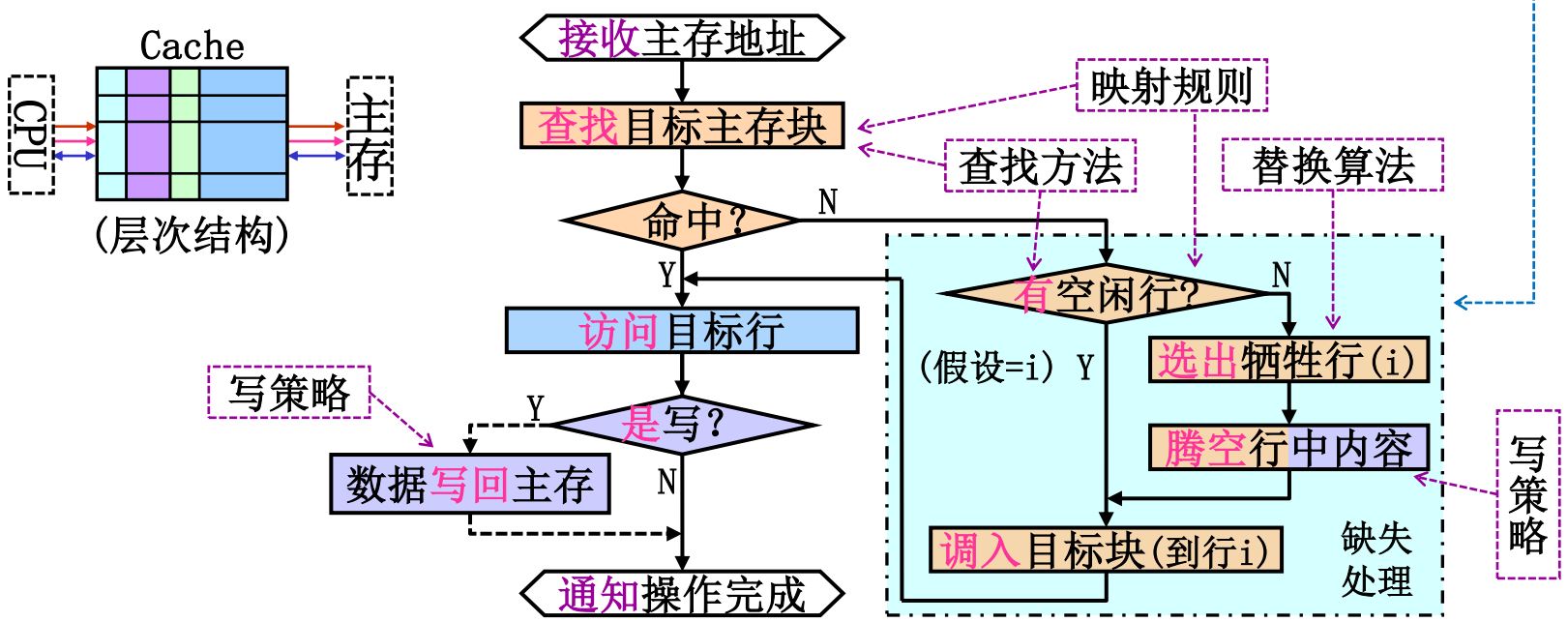


2、Cache的工作原理

*完成访问过程：

场景：好学生迟到时，应怎么给其安排座位？

①地址变换(找目标行)，②数据访问，③数据写回主存



*实现技术：映射规则(块可调入哪些行)，查找方法(如何找到目标行)

替换算法(如何选牺牲行)，写策略(何时/如何写回主存)

*实现要求：全部由硬件完成！ ←目标为高速访问

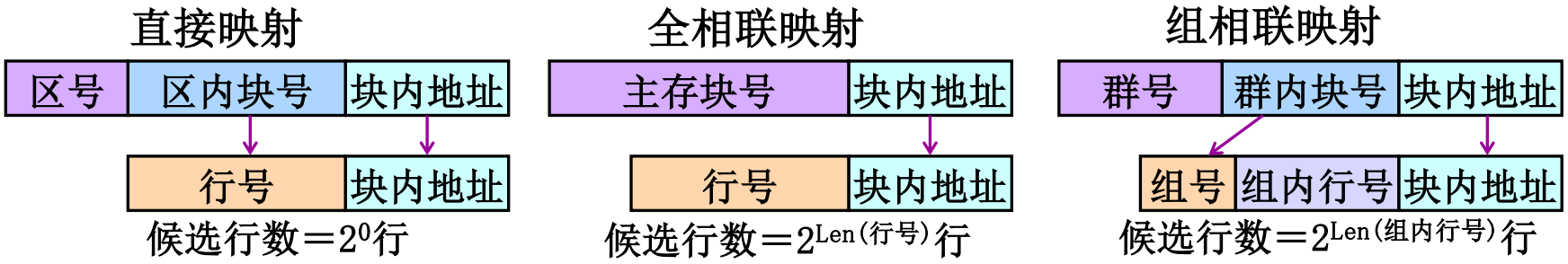
二、实现技术 (有4个方面)

1、映射规则

*任务：确定一个主存块可放到哪些Cache行(候选行)中

*性能指标：块调入时的冲突率

*常见规则：直接、全相联、组相联



思考：4路组相联Cache的相联度是多少？

性能分析—全相联＜组相联<<直接

*常见选择：组相联映射方式(全相联的查找/替换成本太高)

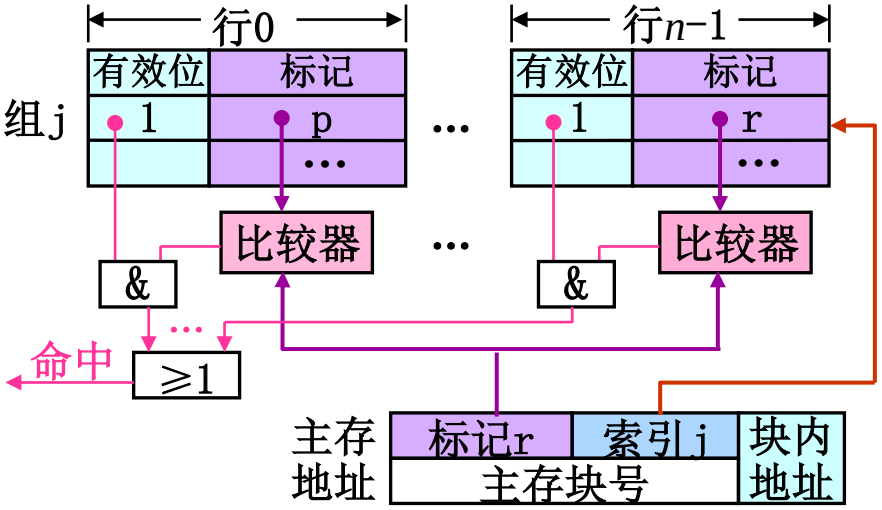
2、查找方法

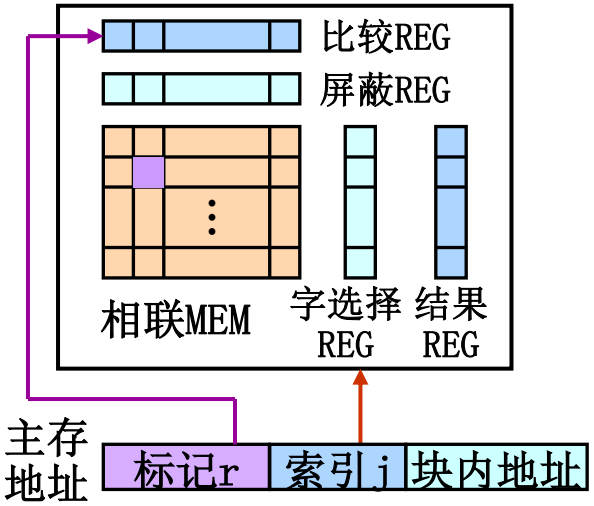
*任务：确定如何查找目标行　　←目标行∈候选行

*性能指标：查找的速度、成本

*方法：确定候选行—块地址＝<标记, 索引>　　←索引项确定候选行

并 行 查 找 —按地址查(单体多字MEM)、按内容查(相联MEM)





性能分析—速度相近，成本为直接＜组相联<<全相联

*常见选择：按地址相联查找 (性/价较好)

*Cache命中时的工作过程：（假设缓存块为单体RAM、写策略为写回法）

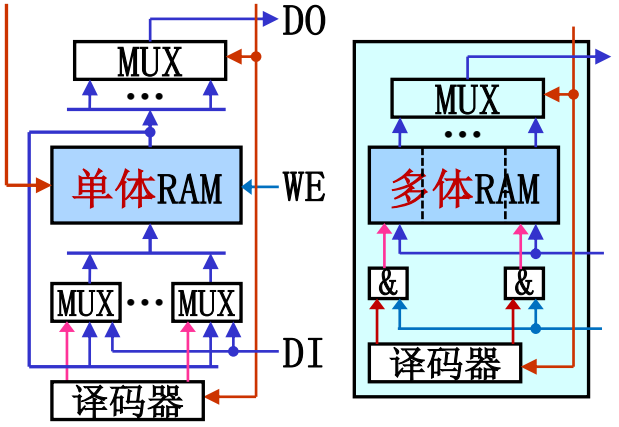
回顾：缓存块中如何实现读/写1个字？

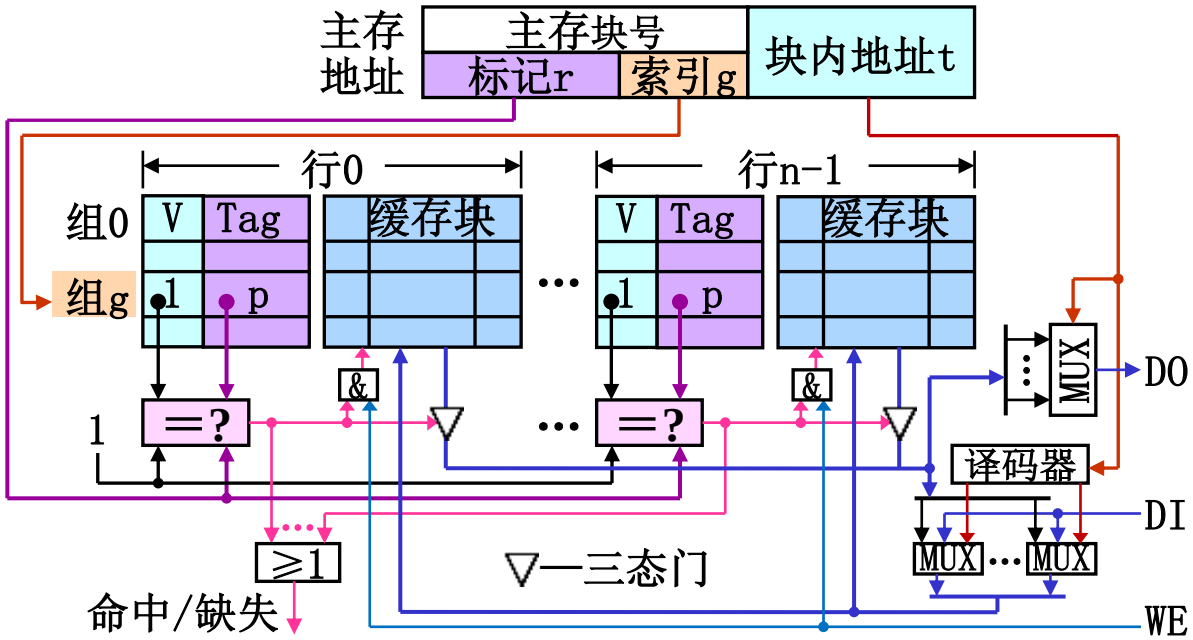
有单体RAM(Flash常用)、多体并行RAM两种

查找
- ①取目录项(各候选行的V/Tag/LRU等)
- ②比较标记(V及Tag)、判断是否命中(≤1行)

访问
- ③取目标块(同时取各候选行[共用MUX/译码器])
- ④读块内数据，或写块内数据、存目标块

访问
- ⑤通知操作完成，更新候选行状态(~替换算法)





思考①：哪些步骤可优化？

读：①及③、②、④、⑤

写：①及③、②、④、⑤

思考②：若缓存块为多体RAM，优化结果？

读：①、②、④、⑤

写：①、②、④直写、⑤

例1：主存按字节编址、地址为24位，Cache的容量为32KB、相联度为4，主存块大小为16B。⑴Cache行中标记(Tag)的长度？⑵CPU访存地址为123456H时，可能命中的Cache组号？命中条件？⑶若Cache初态为空，从0#单元起连续读出100B数据(2B/次)，此时Cache的命中率？⑷Cache有几个比较器？⑸若访问目录表、存储阵列的时延均为10ns，比较器时延为2ns，读块内数据(字)需1ns，则$T_{读命中}$是多少？

解：⑴Tag＝24-$\log_2$(16B/1B)-$\log_2$[(32KB/16B)/4]＝11b

⑵组号＝101000101B ，命中条件＝(V=1)·(Tag=00010010001B)

⑶读次数＝100B/2B＝50，缺失特征＝ 各块首次读时，H＝1-7/50

⑷4个(并行查找)

⑸理论上，$T_{读命中}=T_{取目录项}+T_{比较}+(T_{读缓存块}+T_{读块内数据})=23ns$；

实际上，$T_{读命中}=\max\{T_{取目录项}, T_{读缓存块}\}+T_{比较}+T_{读块内数据}=13ns$

思考：包含替换算法的$T_{读命中}$、$T_{写命中}$是多少？ 两者无关联

$T_{读命中}=\max\{T_{取目录项}, T_{读块}\}+T_{比较}+\max\{T_{读块内数据}, T_{改块状态}\}$

$T_{写命中}=\max\{T_{取目录项}, T_{读块}\}+T_{比较}+\max\{T_{改块内数据}+T_{写块}, T_{改块状态}\}$

3、替换算法

*任务：确定从候选行中如何选出一个牺牲块(行)

*性能指标：对命中率的影响程度、算法的实现成本

└←替换是/否遵循访问局部性

*常见算法：

	状态的个数	状态更新的时机	牺牲行的选择	对H的影响
RAND	1个随机数/Cache	块调入时，产生随机数	随机数对应的行	H随机
FIFO	1个计数值/行	块调入时，更新n个值	(n个)值最大的行	H随机
LRU	1个计数值/行	块访问时，更新n个值	(n个)值最大的行	H随n增大
注：n—组相联的路数(即候选行数)；计数值更新—刚调入/访问的行清零				

*常见选择：LRU算法(∈堆栈型算法)

*LRU算法的硬件实现：(以组为单位，常与目录项分开[每次更新])

堆栈法—寄存器堆栈(相联查找＋中部移出)，栈底对应牺牲行

比较对法—用触发器记录块间次序，用门进行判断

└←$C(n,2)$个　　└←n个n-1入端与门

4、写策略

*任务：确定写操作的数据，何时写到主存

*性能指标：对T_A、总线占用度的影响程度

*常见策略：（按写命中命名）

配对

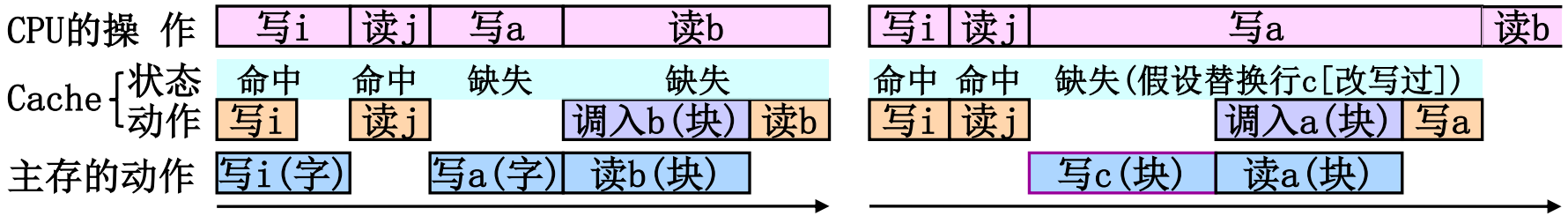
→ 全写法—$T_{写命中}=T_{Mem(字)}$，即同时写$和主存(字)；占用总线/次

→ 写回法—$T_{写命中}=T_{Cache(字)}$，替换时写主存(块)；有不一致性

*写缺失处理方案：

→ 不按写分配法—$T_{写缺失}=0$ ←不调入目标块，直接写主存(=$T_{命中}$)

→ 按写分配法—$T_{写缺失}\geqslant T_{Mem(块)}$ ←调入目标块(可能有替换)后，再写$

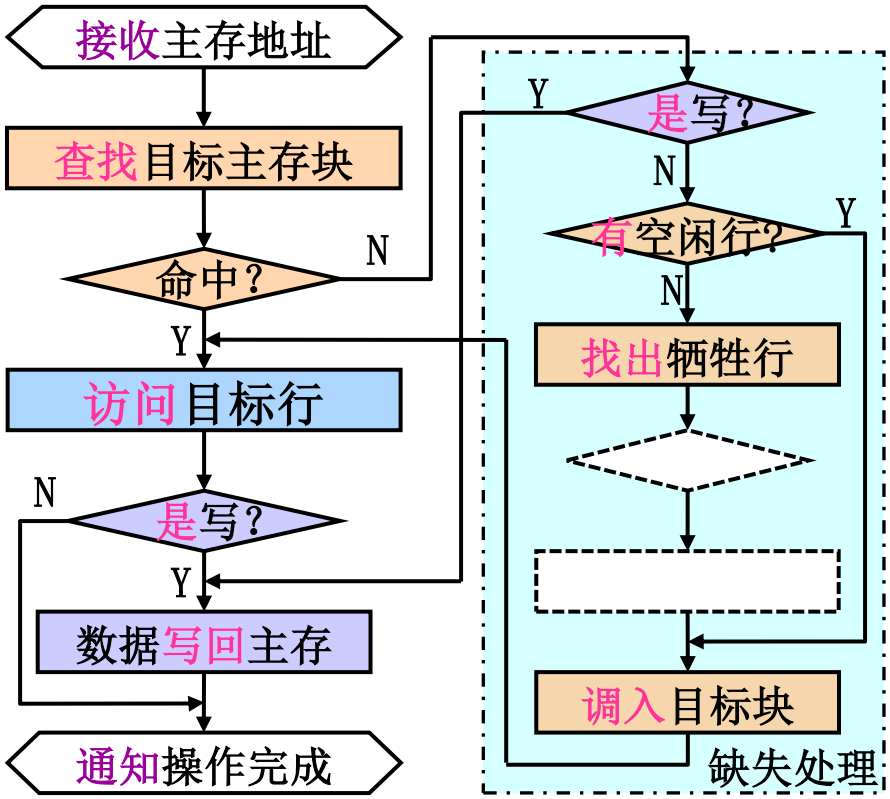


*常见选择：全写法配对不按写分配法，写回法配对按写分配法；

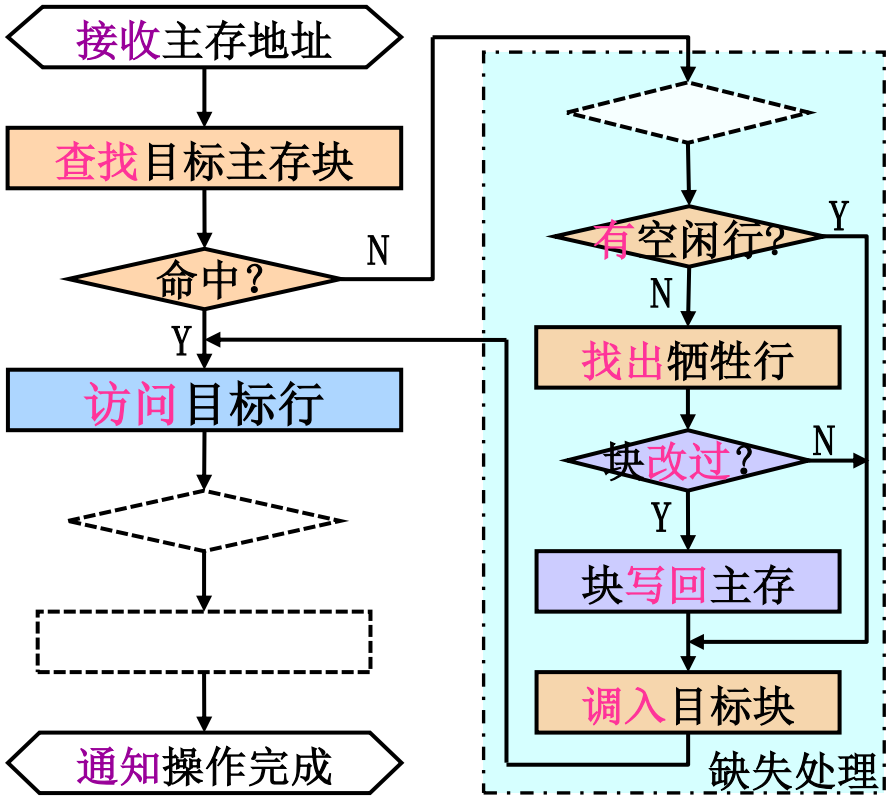
连接总线的Cache用写回法，其余Cache用全写法

└←T_A及总线占用度均好 └←一致性好、$T_{写命中}$较小

*Cache的工作过程：



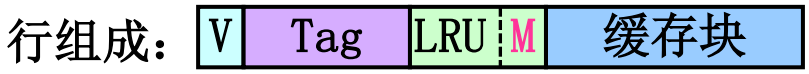
全写法Cache[不按写分配]



写回法Cache[按写分配]

*Cache的硬件配置：

全写法Cache[不按写分配]



写回法Cache[按写分配]

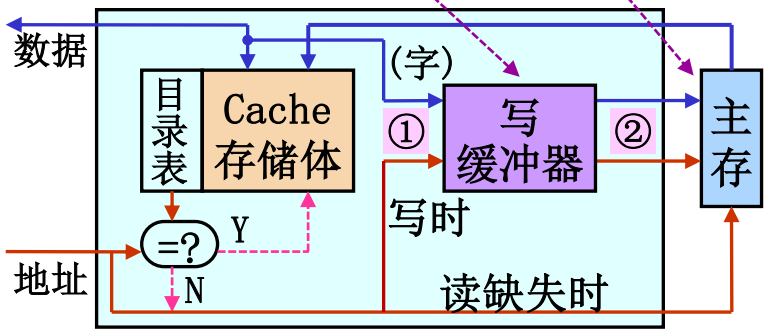
*Cache写主存的性能优化：

目标— 写提交代替写完成，即零等待写

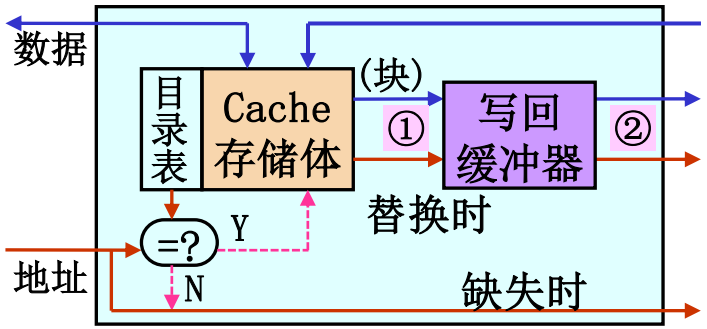
Cache空闲/命中时

└←送出写命令　　└←写完成不占用当前操作时间

实现— 设置写(回)缓冲器，先写入缓冲器、稍后写入主存

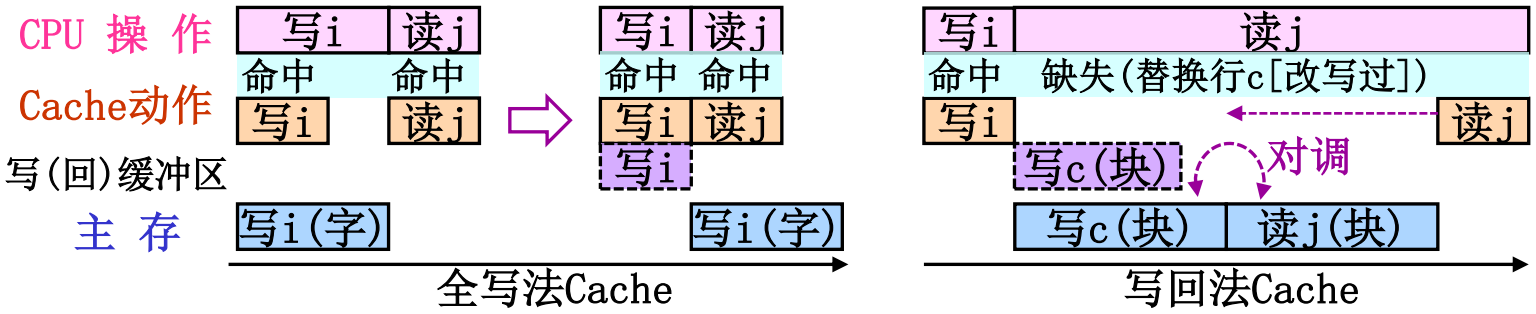


全写法Cache[不按写分配]



写回法Cache[按写分配]

性能— 全写法$T_{命中(写)} = T_{Cache(字)}$，　写回法$T_{缺失(写)} = T_{Mem(块)}$



缓冲器组成— 大小为2～4行

有效位	主存(块)地址	主存(块)数据

三、Cache性能分析

1、平均访问时间T_A

$T_A = T_{命中} + F \cdot T_{缺失}$，$T_{缺失}$包括块调入、替换等时延

$T_A = H \cdot T_{命中} + F \cdot (T_{命中} + T_{缺失})$

例2：设Cache读命中时间为1Tc，块缺失开销为40Tc，主存存取时间为10Tc。某程序执行时共有700次读、300次写操作，读、写操作分别缺失50次、15次。⑴求全写法Cache的T_A；⑵若全写法Cache设有写缓冲器，可隐藏80%的写主存时延，求其T_A；⑶求写回法Cache的T_A。

解：$T_{命中} = (T_{读命中} \times N_{读} + T_{写命中} \times N_{写}) / (N_{读} + N_{写})$，

$F \cdot T_{缺失} = (T_{读缺失} \times N_{读缺失} + T_{写缺失} \times N_{写缺失}) / (N_{读} + N_{写})$

⑴ $T_{写命中} = 10Tc$，$T_{写缺失} = 0Tc$，

故$T_A = (1*700+10*300)/1000+(40*50+0*15)/1000 = 5.7Tc$；

⑵ $T_{写命中} = 0.8*1Tc+0.2*(1Tc+10Tc) = 3Tc$，故$T_A = 3.6Tc$；

⑶ $T_{写命中} = 1Tc$，$T_{写缺失} = 40Tc$，

故$T_A = (1*700+1*300)/1000+(40*50+40*15)/1000 = 3.6Tc$。

2、处理器性能与T_A

$$T_{CPU}=I_N*CPI_{实际}*T_C=(CPU执行的T_C数+MEM停顿的T_C数)*T_C$$

$$=I_N*(CPI_{理想}*T_C+每条指令平均访存次数*F*T_{缺失})$$

└←访存时Cache命中　　乘积是平均值→┴→无需为T_C倍数

例3：某CPU理想CPI＝2.0，程序中每条指令平均访存1.3次；Cache采用直接映射、2路组相联映射时，T_C分别为1ns、1.2ns，$T_{命中}$均为1T_C，$T_{缺失}$均为66ns，相同容量时的F分别为1.4%、1.0%。⑴计算两种Cache的T_A；⑵求2路组相联Cache的实际CPI；⑶哪种CPU的性能更优？

解：(1) $T_{A(1路)}=1*1ns+0.014*66ns=1.924ns$，

$T_{A(2路)}=1*1.2ns+0.01*66ns=1.860ns$；

(2) $CPI_{实际}=CPI_{理想}+1.3*0.01*66ns/1.2ns=2.715$；

(3) $T_{CPU(1路)}=I_N*(2*1ns+1.3*0.014*66ns)=I_N*3.2012ns$，

$T_{CPU(2路)}=I_N*(2*1.2ns+1.3*0.01*66ns)=I_N*3.258ns$，

直接映射时的CPU性能更优。

第3节　Cache的性能优化

※主要内容：降低缺失率，减小缺失开销，减小命中时间

（$T_A = T_{命中} + F \cdot T_{缺失}$）

一、降低Cache的缺失率

*缺失的类型：

强制缺失—第一次访问某个块时，块不在Cache中

容量缺失—所需块不能全部调入，替换后又重新访问

冲突缺失—多个块映射到同一组，替换后又重新访问

*缺失的影响因素：

强制缺失—仅与块大小有关　　（与容量、相联度无关）

容量缺失—与Cache容量、访问地址流(软件工作特性)有关

冲突缺失—与相联度有关

*降低缺失率的方法：有多种(如增加容量、块大小、相联度，预取等)，

应尽量避免增加$T_{命中}$或$T_{缺失}$

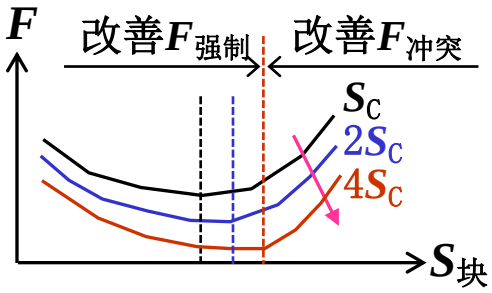
.

1、增加Cache块大小

*优化原理：可改善强制缺失率

*测试结果：① $S_{\$}$不变，$S_{块}$↑时 F先↓后↑

② $S_{\$}$增加，$F$最小时的$S_{块}$↑



*性能分析：$S_{块}$↑导致F↓、$T_{缺失}$↑，不影响$T_{命中}$

└←$T_{调入}=T_{存取}+n*T_{传输}$

思考①：为什么？

*块大小的选择：

依据—使$F*T_{缺失}$最小

例1：表中为各容量的F，若$T_{命中}=1Tc$，$T_{传输}=2Tc/16B$，请选择$S_{块}$

$S_{块}$及$T_{缺失}$ \ $S_{\$}$		4KB	16KB	64KB	256KB
16B	80+2	8.57%	3.94%	2.04%	1.09%
32B	80+4	7.24%	2.87%	1.35%	0.70%
64B	80+8	7.00%	2.64%	1.06%	0.51%
128B	80+16	7.78%	2.77%	1.02%	0.49%
256B	80+32	9.51%	3.29%	1.15%	0.49%

解：$T_A=T_{命中}+F\cdot T_{缺失}$，

$T_{A(4K/16)}=1+8.57\%*82=8.027Tc$,

$T_{A(4K/32)}=1+7.24\%*84=7.082Tc$,

$T_{A(4K/64)}=1+7.00\%*88=7.160Tc$,

得 4KB时$S_{块}=32B$，

16KB/64KB/256KB时$S_{块}=64B$

思考②：例1中80→40，结果会变化吗？

结果—$S_{块}$尽量大，受限于下层MEM延迟与带宽（$T_{调入}$越小$S_{块}$越大）

思考①：不同块大小，仅影响T命中的块内数据选择时间（可忽略）；②：会，根据计算结果

2、提高相联度（组内块数为n）

***优化原理：**可改善冲突缺失率

***测试结果：**（基于P207表7.3[基准测试程序的测试结果]）

① $S_{\$}$不变，$F_{8路} \approx F_{全相联}$

② $S_{\$} \leqslant 128KB$时，$S_{\$}$的$F_{1路} \approx S_{\$}/2$的$F_{2路}$

***性能分析：**n↑导致F↓、$T_{命中}$↑　　（$T_{命中} = T_{取目录项} + T_{比较} + T_{块访问}$）

例2：若$T_{命中}=1T_C$，$T_{C2路}=1.36T_{C1路}$、$T_{C4路}=1.44T_{C1路}$、$T_{C8路}=1.52T_{C1路}$，$T_{缺失}=25T_{C1路}$(与n无关)，表7.3中满足$T_{A8路} < T_{A4路} < T_{A2路} < T_{A1路}$的条件？

解：$T_{A2路}=1.36+F_{2路}*25$，$T_{A4路}=1.44+F_{4路}*25$，$T_{A8路}=1.52+F_{8路}*25$，

计算后可得，满足的条件为：$S_{\$} \leqslant 8KB$、$n \leqslant 4$时。

分析—$S_{\$}$较大时，$n$↑导致的$F_{冲突}$↓有限，$F \cdot T_{缺失}$↓＜$T_{命中}$↑

***相联度的选择：**n尽量大、应不增加$T_{命中}$　　←$T_{命中}$影响$CPI_{理想}$及T_C

L2\$不考虑$T_{命中}$→⊥→n可更大

思考：为何相联度为8时产生冲突概率很低？

①指令及数据的局部性较高，相邻信息放在不同组中，无冲突；

②同一时间片内执行的进程不多，相联度接近于进程数即可。

3、伪相联Cache

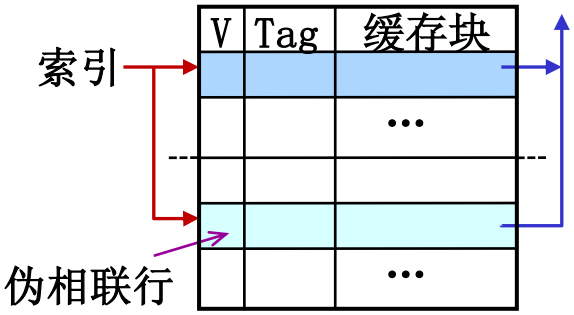
*优化原理：直接映射方式的候选行为2个 ←提高相联度的变种

*工作过程：首先查找直接映射行，

缺失时查找伪相联行，

└←如索引高位取反

再缺失时才访问下级MEM



*性能分析：$T_A = T_{命中直接} + (F_{直接} - F_{2路}) \cdot T_{命中伪相联} + F_{2路} \cdot T_{缺失}$

结果—$F\downarrow = F_{2路}$，$T_{命中} = T_{命中直接}$ 或 $T_{命中伪相联}$

$T_{命中}$优化—①伪命中时交换2个行的内容 ←利用局部性

└←包括调入时

②同时查找，$T_{命中伪相联}' = T_{命中伪相联} - T_{查表}$

$T_{命中}$的主要时延

*特点：流水线设计复杂化($T_{命中}$波动)，可用于L2$

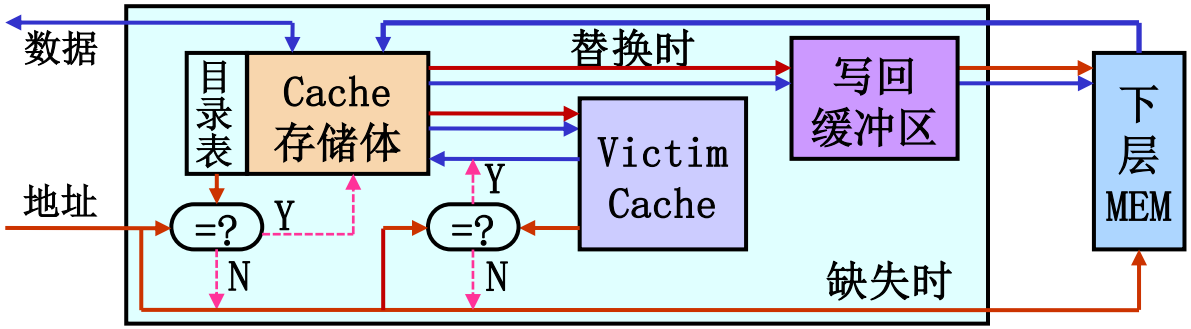
*应用：较少使用(L2$的相联度较高)

4、牺牲Cache (Victim Cache)

*优化原理：用全相联小Cache存放被替换的块(减少下层MEM访问)

*工作过程：Cache缺失时，查找Victim；

Victim命中时，目标块移入Cache，否则调入(下层MEM)



*性能分析：$T_A = T_{命中\$} + F_{\$} \cdot H_{Victim} \cdot T_{命中Victim} + (1 - H_{Victim}) \cdot F_{\$} \cdot T_{缺失}$

结果—$F\downarrow = (1 - H_{Victim}) \cdot F_{\$}$，不影响$T_{命中}$($T_{命中Victim}$可优化)

$T_{命中Victim}$优化—同时查表，$T_{命中Victim}' = T_{命中Victim} - T_{查表}$

$T_{命中}$主要时延

*Victim的容量：几个块，全相联映射

测试结果：$S_{直接映射\$}$=128块、$S_{Victim}$=4块的 H_{Victim}为20%~90%

思考：同时查表，$T_{命中Victim} = T_{命中Victim} - T_{查表}$

5、硬件预取

*优化原理：信息访问前，用预取器(PreF, Prefetch)预先调入

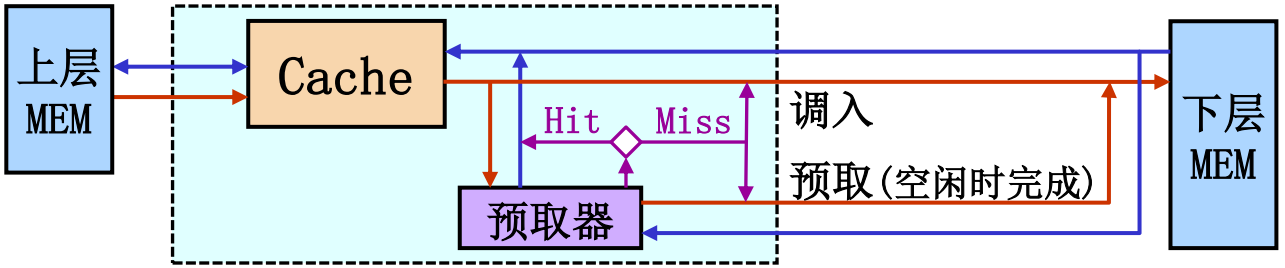
└←可放在Cache内部或外部

*工作过程：Cache缺失时，查找PreF；

PreF命中时，目标块移入Cache，

PreF缺失时，Cache发出调入请求、PreF发出预取请求

优先级较低



预取算法—有多种，如下个块，或根据最近2次缺失地址选择

*性能分析：$T_A = T_{命中} + F_{\$} \cdot H_{预取} \cdot T_{预取器命中} + (1 - H_{预取}) \cdot F_{\$} \cdot T_{缺失}'$

$T_{缺失}' = T_{缺失} + k$，　(MEM空闲时$k=0$、预取未完成时$k>0$)

结果—$F\downarrow = (1 - H_{预取}) \cdot F_{\$}$，$T_{缺失}$可能↑，不影响$T_{命中}$($T_{预取器命中}$可优化)

例3：某Cache的$T_{命中}=1T_C$，$T_{缺失}=50T_C$，$F=1.1\%$；采用硬件预取技术后，预取器命中需$1T_C$，$H_{预取}=25\%$；求优化后的缺失率。

解：$T_A'=T_{命中}+F\cdot H_{预取}\cdot T_{预取器命中}+F\cdot(1-H_{预取})\cdot T_{缺失}$，

$=1+1.1\%*25\%*1+1.1\%*(1-25\%)*50=1.415T_C$；

$T_A'=T_{命中}+F'\cdot T_{缺失}$，$F'=(1.415-1)/50=0.83\%$，

$F\downarrow=(1.1-0.83)/1.1=24.6\%$

*预取器的大小：几行，全相联映射

测试结果：$S_C=256$行、$n=1$的I$中，FBuff=1时$F\downarrow=15\sim25\%$，

FBuff=4时$F\downarrow=50\%$；

$S_C=2048$行、$n=4$的D$中，FBuff=8时$F\downarrow>50\%$

*应用举例：

2个源OPD/指令　　2个核无局部性

Core 2有8个预取器{(L1-I$*1+L1-D$*2)*2+L2$*2}；

L1$、L2$的预取器采用不同的预取算法

6、编译器控制的预取

*优化原理：编译器在代码的适当位置插入预取指令　　←提高$H_{预取}$

*预取的类型：

按存放位置分—寄存器预取(1个字/次)、Cache预取(1个块/次)

按处理方式分—故障性预取(产生异常)、非故障性预取(放弃预取)

└←指地址故障，如缺页或保护错

前瞻指令(编译时)　　预取指令(编译时)

*性能分析：$F\downarrow$(预取效率较高)，可能阻塞下次访存

└←预取指令需执行，应不影响正常指令

阻塞的处理—采用非阻塞Cache(当前请求缺失时，可处理后续请求)

*特点：预取效率较高，硬件成本无/较低，软件可移植性差

7、编译优化

*优化原理：通过软件优化减少指令/数据缺失率

*代码重组：

过程重排序—将有调用关系的过程靠近存放 ←提高局部性

块对齐—代码入口与块起始位置对齐 ←强制缺失率↓

转移校正—大概率转移目标放在分支指令之后 ←提高局部性

if（条件）go L1
基本块1(大概率)
goto L2
L1:基本块2(小概率)
L2: …

*数据重组：（主要针对数组）

数组合并—
```
int Val[100];   →  struct me  {
int Key[100];          int val;
                       int key } m_Arr[100];
```

循环交换— 根据数组的存储顺序进行交换 ←提高空间局部性

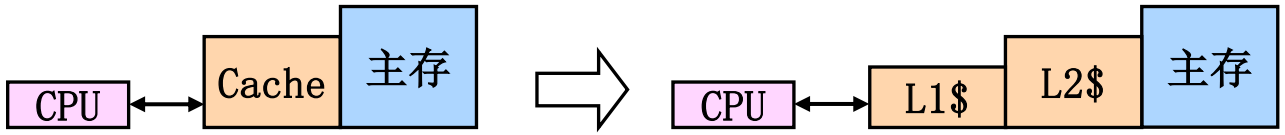
```
for (j=0;j<100;j++)        →   for (i=0;i<500;i++)
  for (i=0;i<500;i++)              for (j=0;j<100;j++)
    x[i][j]=2*x[i][j];               x[i][j]=2*x[i][j];
```

数据分块— 矩阵操作→子矩阵操作 ←提高时间局部性

二、减少Cache的缺失开销

1、两级Cache

*优化原理：采用层次结构减小$T_{缺失}$ ←有$S_{L1\$} << S_{L2\$}$



*性能分析：$T_A = T_{命中} + F \cdot T_{缺失} = T_{L1命中} + F_{L1} \cdot T_{L2命中} + F_{L1} \cdot F_{L2} \cdot T_{L2缺失}$

其中，F_{L1}、F_{L2}称为$F_{局部}$，$F_{L1} \cdot F_{L2}$称为$F_{全局}$

例4：1000次访存中，L1$缺失40次，L2$缺失20次，$T_{L1命中}$＝1Tc，$T_{L2命中}$＝10Tc，$T_{L2缺失}$＝100Tc，求仅L2$(即$F = F_{全局}$)、两级$时的T_A

解：F_{L1}＝4%，F_{L2}＝50%，$F_{全局}$＝4%*50%＝2%;

$T_{A(1级)}$＝10＋2%*100＝12Tc，$T_{A(2级)}$＝1＋4%*10＋2%*100＝3.4Tc

结果— $S_{L2\$} = S_{单级\$}$、$S_{L1\$} << S_{L2\$}$时，$F_{L1} \cdot F_{L2} \approx F_{单级}$;

L1$可减小$T_{命中}$及$T_C$，L2$仅影响$T_{L1缺失}$ ←L2$组织较灵活

优化— 减小F_{L2}可降低$T_{缺失}$ ←$T_{缺失} = (F_{L1} \cdot T_{L2命中} + F_{L1}F_{L2} \cdot T_{L2缺失}) / F_{单级}$

*L2$的组织：　$T_{L1缺失}=T_{A(L2)}/F_{L1}=(T_{L2命中}+F_{L2}\cdot T_{L2缺失})/F_{L1}$

$T_{A(L2)}$减小即可

容　量—可较大，如$S_{L2\$}=512KB$　←降低$F_{L2容量}$，$T_{L2命中}$基本不变

相联度—可较高，如$n_{L2\$}>n_{L1\$}$　←降低$F_{L2冲突}$，$T_{L2命中}$略有增加

例5：L2$中，若$T_{L2命中(1路)}=10Tc$、$F_{L2(1路)}=25\%$，$T_{L2命中(2路)}=10.1Tc$、$F_{L2(2路)}=20\%$，$T_{L2缺失}=100Tc$，分析L2$的相联度对$T_{L1缺失}$的影响

解：$T_{L1缺失(1路L2)}=10+25\%*100=35Tc$，

$T_{L1缺失(2路L2)}=10.1+20\%*100=30.1Tc$，

∴提高L2$相联度，可降低$T_{L1\$缺失}$！

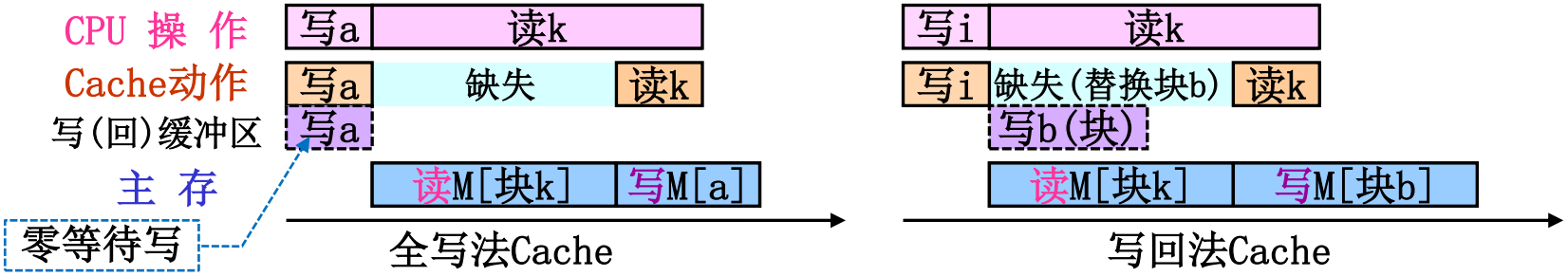
块大小—可较大，如$S_{块(L2\$)}>S_{块(L1\$)}$　←降低$F_{L2强制}$，不影响$T_{L2命中}$

思考①：Cache级数是否越多越好？　更多级不减小$T_{命中}$，F的优化两级即可实现

思考②：L1$可否不为L2$的子集？　可以，可降低$T_{缺失}$(同时查找)，实现复杂

2、读缺失优先于写

*优化原理：先处理读缺失，后处理写(回)操作　←需写缓冲区支持



*工作过程：Cache缺失时，查找写(回)缓冲区；　←可同时查找

缓冲区命中时，目标块移入Cache；　←全写法读为调块

缓冲区缺失时，立即调入目标块，稍后处理缓冲区

*性能分析：$T_A = T_{命中} + H_{缓冲区} \cdot F \cdot T_{缓冲区命中} + (1 - H_{缓冲区}) \cdot F \cdot T_{缺失}$

结果—$T_{读缺失}\downarrow = T_{读块}$，$T_{写缺失}\downarrow = 0$(全写法)或$T_{读块}$(写回法)，

$T_{读缺失}\downarrow$ ←原来$\geq T_{读块}$　$T_{写缺失}\downarrow = 0$ ←不变　$T_{读块}$ ←原来$\geq T_{读块}$

不影响$T_{命中}$、可降低F(命中写缓冲区)

3、写缓冲合并　　--适于全写法Cache

*写缓冲区组成：有几个缓冲行

缓冲行组成— 包含多个子项，各子项有独立的有效位

└←支持不同长度数据→┘

写请求处理— 有空闲行时添加请求，否则阻塞($T_{写命中} > T_{Cache}$)

	地址	有效位	子项3	子项2	子项1	子项0	请求内容
行0	00100H	0011			D1	D0	①[00100H]←data[2字节]
行1	00102H	0001				D2	②[00102H]←data[1字节]

*优化原理： 合并写请求，隐藏/减少写操作开销(次数)

合并实现— 请求地址有匹配行时合并请求，否则添加请求

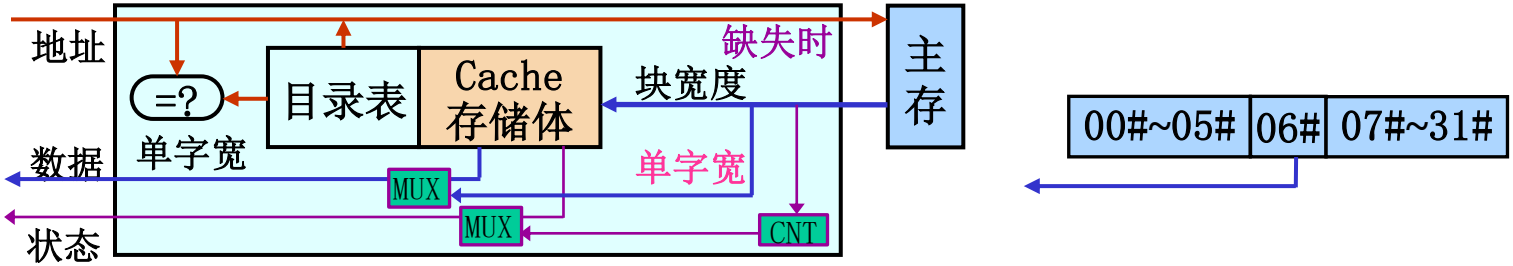
	地址	有效位	子项3	子项2	子项1	子项0	请求内容
行0	00100H	0111		D2	D1	D0	①[00100H]←data[2字节]
行1	00000H	0000					②[00102H]←data[1字节]

*性能分析： 减少写主存次数，提高写缓冲区利用率

4、请求字处理技术

*优化原理：块调入完成前，将请求数据送给CPU(操作提前完成)

*尽早重启方案：块按序调入，请求字到达时立即送给CPU



实现方法一 选择器(存储体[Hit]/主存[Miss])＋计数器(初值=块内地址)

*请求字优先方案： 先调入请求字送给CPU，后调入块中其他字

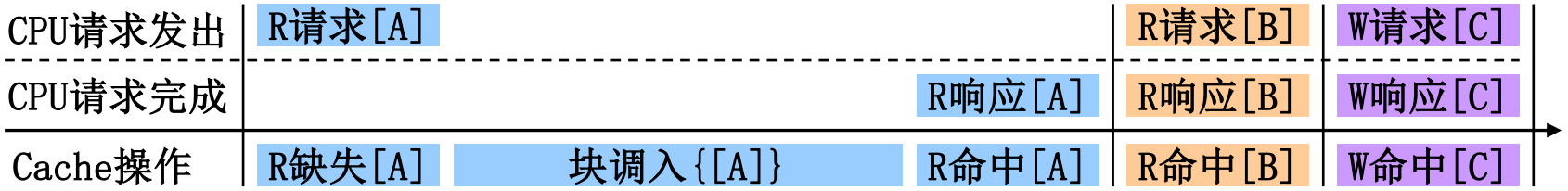
R06#	06#	00#~05#	07#~31#

实现方法一 特殊的行读总线事务

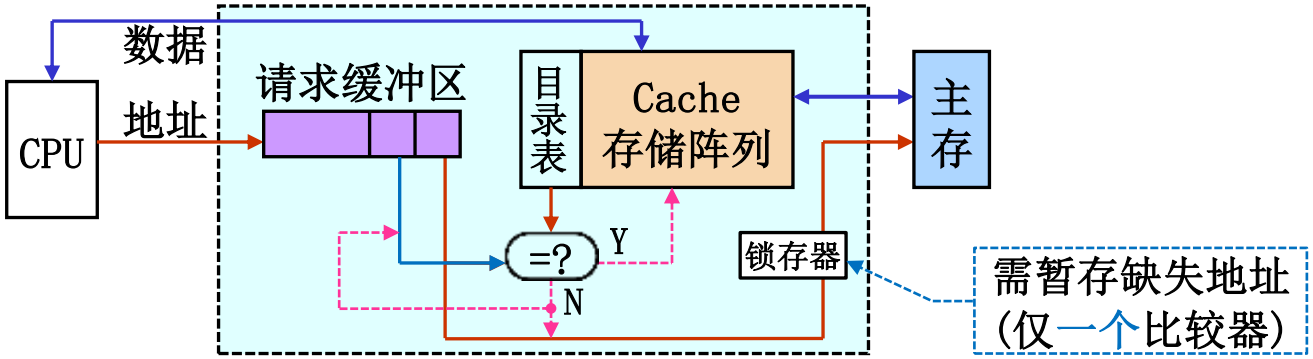
*性能分析： Cache块较大时效果明显

5、非阻塞Cache

*优化原理：当前请求缺失时，可处理后续请求(隐藏$T_{缺失}$)



*实现要求：增设请求缓冲区，当前请求缺失时处理后续请求



*性能分析：缺失后命中的$T_{缺失}$↓较好(多重缺失后命中的$T_{缺失}$↓有限)

└←$S_{请求缓冲区}=2$

思考：为什么？

可提高Cache带宽(接收访问的速度)

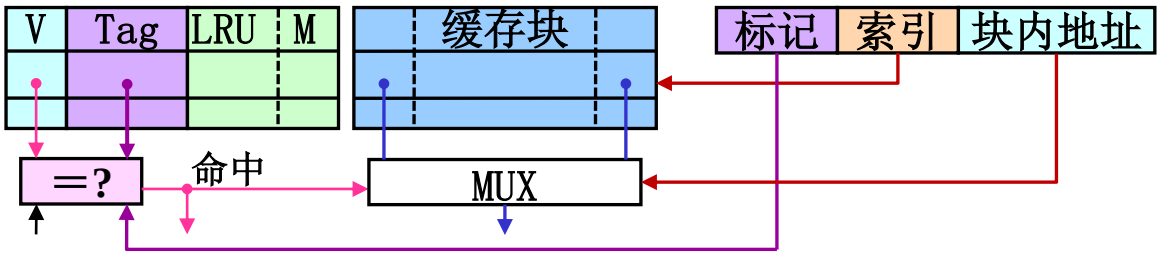
*应用：指令乱序执行CPU中，常采用单重缺失后命中方式

思考：①2个调入请求需串行完成，无法优化；②所请求数据应用时，指令间出现RAW的概率很大，CPU性能未提高

三、减少Cache的命中时间

1、小而简单的一级Cache

$T_{命中}=T_{取目录项}+T_{比较标记}+T_{访问数据}$ ← $T_{命中}$影响T_C



*优化原理：减小/隐藏$T_{命中}$子项，如$T_{比较标记}$与$T_{访问数据}$并行

*性能分析：容量小利于$T_{取目录项}$↓，结构简单利于$T_{比较标记}$↓

例6：L1$中，$T_{L1命中(2路)}=1Tc$、$F_{L1(2路)}=4.9\%$，$T_{L1命中(4路)}=1.1Tc$、$F_{L1(4路)}=4.4\%$，$T_{L1缺失}=10Tc$，哪种方式的$T_A$更好？

解：$T_{A(2路)}=1+0.049*10=1.49Tc$；

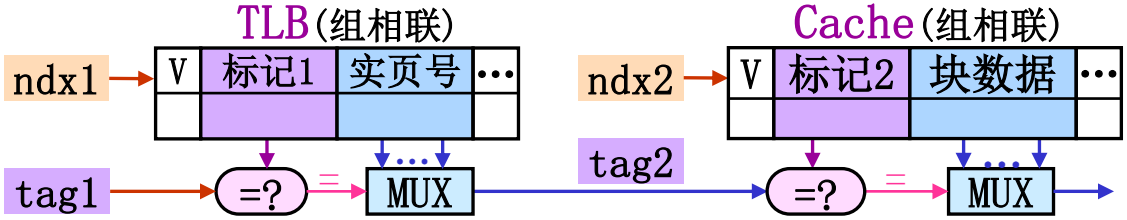
$Tc'=1.1Tc$，$T_{A(4路)}=(1+0.044*10/1.1)Tc'=1.54Tc$

└→CPU不访存时也变慢！（$T_{L1命中}=1Tc$时）　└←平均值不考虑Tc的整数倍

思考：比较器、译码器同时工作

2、虚拟索引Cache

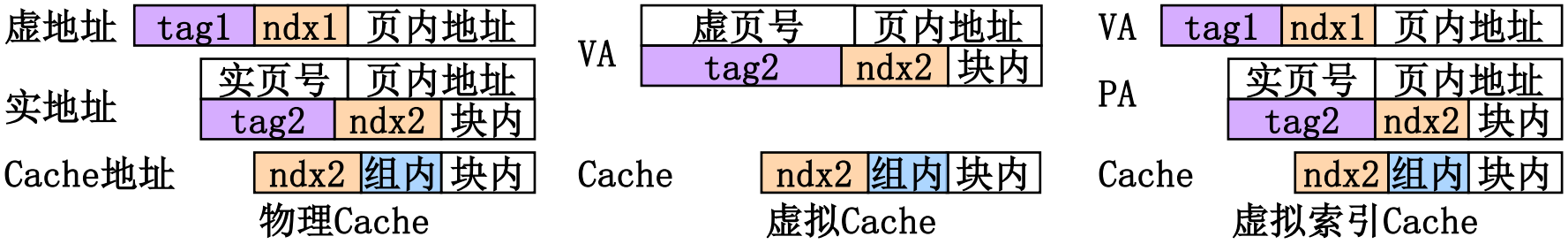
*物理Cache：物理索引-物理标识，先地址变换、后Cache操作



$T_{命中(物理Cache)} = T_{地址变换} + T_{取目录项} + T_{比较标记} + T_{访问数据}$

*虚拟Cache：虚拟索引-虚拟标识，无地址变换，实现困难

$T_{命中(虚拟Cache)} = T_{取目录项} + T_{比较标记} + T_{访问数据}$



*虚拟索引Cache： 虚拟索引-物理标识(ndx2∈页内地址、tag2∈实页号)

优化原理— 地址变换与Cache操作重叠，隐藏$T_{取目录项}$

$T_{命中(虚索引Cache)} = \max(T_{地址变换}, T_{取目录项}) + T_{比较标记} + T_{访问数据}$

应用—仅用于L1-Cache ←层次结构Cache的顶层

容量限制—因$L_{ndx2+块内地址} \leqslant L_{页}$， ←虚拟索引所要求

故$S = 2^{L_{ndx2}} * n * 2^{L_{块}} \leqslant$ 页大小*相联度n

例：PIII的L1-I\$，$S_{页}$=4KB、$S_{块}$=32B、n=4，容量$\leqslant 2^7*4*2^5$B＝16KB

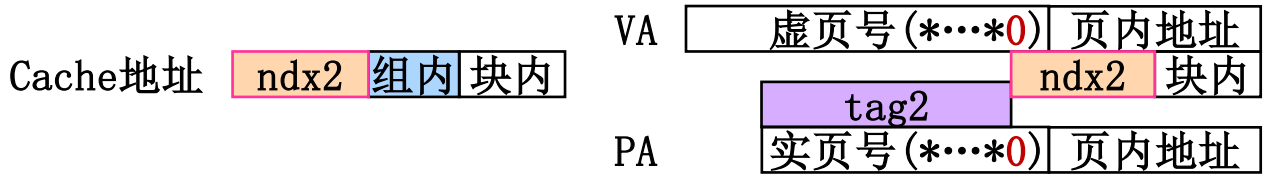
容量扩展—（保证$T_{命中}$时优化F）

提高相联度：$T_{命中} = k \cdot T_C$，k不变时增加n ←虚拟索引提供了可能

例：i7的L1-D\$，$S_{页}$=4KB、$S_{块}$=64B、n=8，容量$\leqslant 2^6*8*2^6$B＝32KB

页着色：OS使实页号与虚页号低m位相同(ndx2扩展m位)

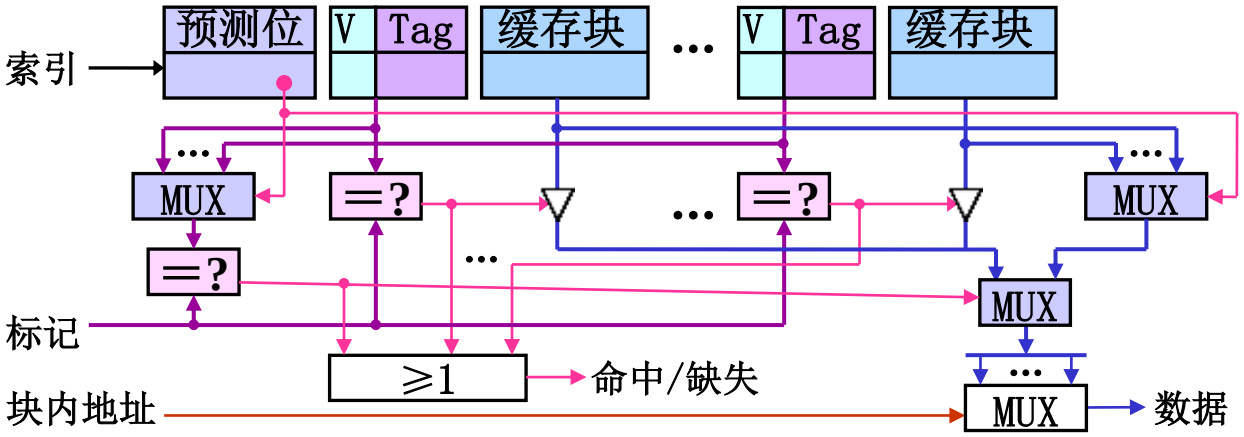
例：i7的L1-I\$，n=4，m=1(奇页←奇页)，容量$\leqslant 2^{(m+6)}*4*2^6$B＝32KB



3、路预测　　--适于相联度较大的Cache

*优化原理：每次预测下次访问的行(组内)，下次直接访问(减少时延)

*实现要求：每组设置1个预测位(宽度＝$\log_2 n$)，用于预测实现



*工作过程：①比较预测行标记，读该行缓存块；

②猜对时，访问数据、修改预测位；　←速度为直接映射

猜错时，进行组相联操作、修改预测位

*性能分析：猜对时$T_{命中}=k'T_C$，否则$T_{命中}=(k+1)T_C$

直接映射时延→┘　　组相联时延→┘　└←回头代价

测试—$k'=0.9k$；准确率在$n=2$时＞90%、$n=4$时＞80%

4、访问流水化

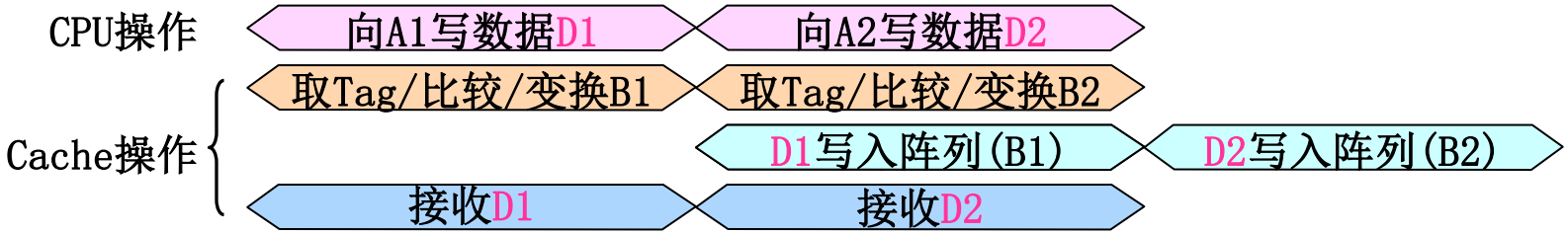
数据/管理操作可并行

$$T_{读命中}=\max\{T_{取目录项},T_{读块}\}+T_{比较标记}+\max(T_{选字},T_{改状态})$$

$$T_{写命中}=\max\{T_{取目录项},T_{读块}\}+T_{比较标记}+\max(T_{改字\&写块},T_{改状态})$$

$$T_{读命中}及T_{写命中}=kT_C\ (k\geqslant 1)$$

*优化原理：将Cache访问流水化，提高Cache带宽(时延未变)



*实现要求：设置段间寄存器，存储体采用双端口SRAM

如：SRAM→取目录项等→段间REG→读写操作等→SRAM

*性能分析：$T_{命中}$不变，Cache带宽提高，$T_{命中}$不影响T_C

等价于$T_{命中}$↓

利于Cache及CPU优化

*应用示例：PII流水线(参见PPT4的P48图)

※Cache优化小结：

优化技术	F	$T_{缺失}$	$T_{命中}$	带宽	O(硬件)	说明
增加块大小	+	−			0	
增加Cache容量	+				1	
提高相联度	+		−		1	L1$受限
伪相联Cache	+				2	适于直接映射，较少用
牺牲Cache	+				2	
硬件预取	+				I-2, D-3	
编译器控制的预取	+				3	
编译优化	+				0	
两级Cache		+			2	L2$容量、相联度可较大
读失效优先于写		+			1	需配置写缓冲器
写缓冲合并		+			1	适于全写法Cache
请求字处理技术		+			2	适于块较大时
非阻塞Cache		+		+	3	
小而简单的Cache	−		+		0	适于L1$
虚拟索引Cache			+		2	适于L1$、容量受限
路预测			+		1	
Cache访问流水化			+	+	3	

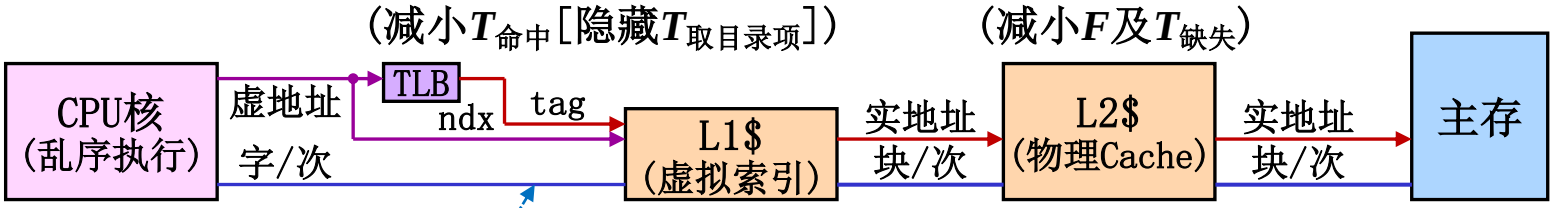
.

※常见Cache结构：

虚-实地址变换

$T_A = T_{地址变换} + T_{命中} + F \cdot T_{缺失}$，　$T_{命中} = T_{取目录项} + T_{比较标记} + T_{访问数据}$

(1)两级Cache：L1$为虚拟索引Cache、L2$为物理Cache



技术一　较大的块(减小F)，设置写缓冲器(减少$T_{命中}$及$T_{缺失}$[隐藏$T_{写MEM}$])

(2)L1$：容量小、相联度$n$适中

←目标是$T_{命中}$较小、不影响T_C

降低F→┴←增大容量(≤$S_{页} \times n$)　←虚拟索引隐藏了时延

技术一　牺牲Cache＋硬件预取(减小F)，

请求字处理＋读失效优先于写＋非阻塞Cache(减小$T_{缺失}$)，

访问流水化(减小$T_{命中}$[增加带宽/不影响T_C])

(3)L2$：容量大、相联度$n$较高

←目标是减小F、减小$T_{缺失}$

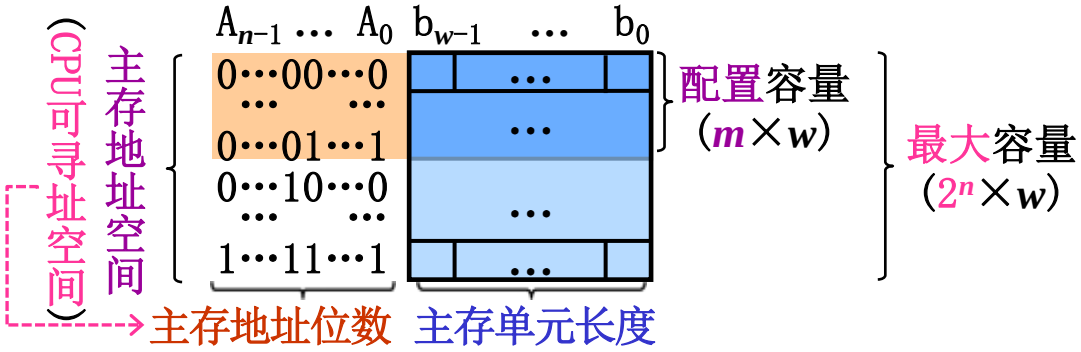
└←不影响$T_{命中}$(指L1$)→┘

技术一　牺牲Cache＋硬件预取，读失效优先于写＋非阻塞Cache

第4节　主存的性能优化

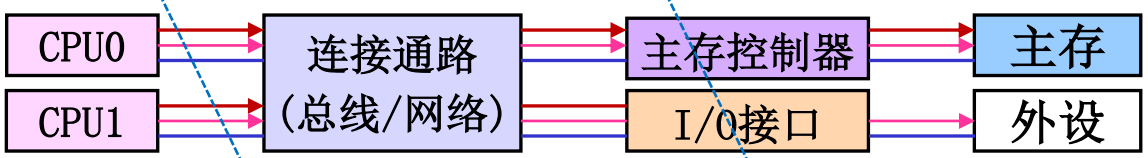
※主要内容：单体多字MEM，多体交叉MEM，并行MEM

*主存组成：ROM+RAM，参数均已给定(CA/用户)



*访存需求：每次多个连续单元，多次地址连续，并行访问

(1个数据＝多个单元)　(访问局部性)　(多处理器)



*主存性能：时延(如T_M)、带宽(如w/T_M)　←Cache关注时延,I/O关注带宽

*主存优化：改进工艺、并行处理(多个MEM)、层次结构(多种MEM)

└→如单体多字MEM、多体交叉MEM、多端口MEM

└→MEM内部，如FPM DRAM、SDRAM等

1、单体多字存储器

存储体—具有地址译码器、读/写电路的MEM模块/芯片

存储字长(记为W)：即存储单元长度　←内部，矩阵存取单位

编址单位(记为w)：即I/O宽度，可≠W　←外部，最小访问单位

*单字单字MEM：即$W=w$，可访问w/次　(设存储周期为T_M)

性能— 时延差(=$n*T_M$)、带宽低($B_M=w/T_M$)　←访问n个单元时

例1：某主存W=8B，T_M=1Tc(地址锁存)+12Tc(矩阵存取)+1Tc(I/O)；
Cache的$S_{块}$=32B，则$T_{调块}=4*(1+12+1+T_{传输})\geqslant 56Tc$

*单体多字MEM：即$W=m*w$，可访问$n*w$/次($1\leqslant n\leqslant m$)

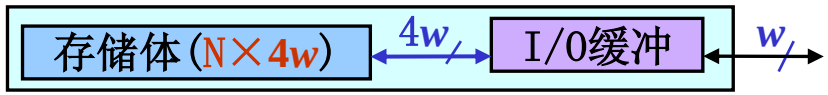
实现— 读时同时取(W)、分时输出($n*w$)；　←常设置缓冲器

写时同时取(W)、分时修改($n*w$)、同时存(W)

性能— $T_{M读}$较好、$T_{M写}$略大，$B_M=n*w/T_M$，功耗大(m个字/次)

例2：例1中W=32B、w=8B时，$T_{调块}=1+12+1*4+T_{传输}\geqslant 17Tc$

应用— DDR2 SDRAM、Flash等
(W=512B)

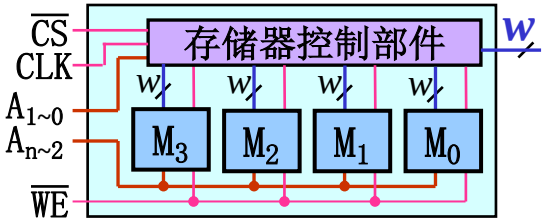


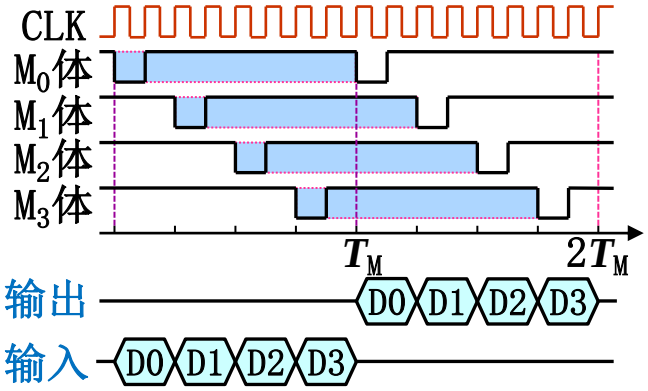
2、多体交叉存储器　（多体多字MEM）

*编址方式：交叉编址(便于多个字/次)

*交叉访问方式：

结构— I/O宽度＝w，可访问$n*w$/次





原理— 每隔T_M/m启动一个体，突发传送方式I/O　←$n \leqslant m$

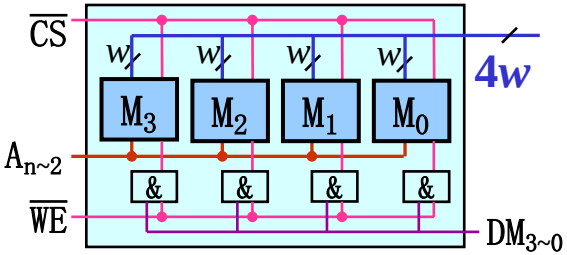
性能— T_M较好，$B_M=w/(T_M/m)$，功耗小(n个字/次)

*并行访问方式：

结构— I/O宽度＝$m*w$，可访问$n*w$/次

原理— 同时启动各个体，并行I/O

性能— T_M好，$B_M=m*w/T_M$，功耗小(n个字/次)

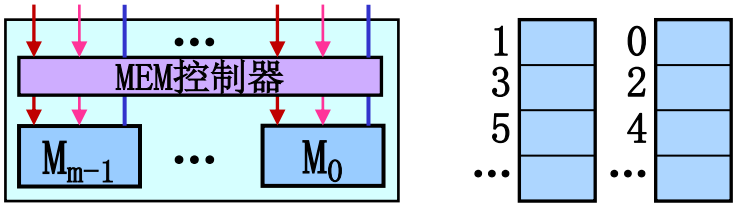


思考：与单体多字MEM的区别？

思考：I/O宽度增加，$T_{M写}$好，功耗小

*应用：　2级(并行访问+交叉访问)多体交叉MEM　←并行访问T_M↓，交叉访问B_M↑

3、并行存储器　（多个独立MEM）



*编址方式：交叉编址或顺序编址

*结构：每个体有独立的I/O信号线

*原理：同时管理/调度多个独立的请求，

没有体冲突的请求可并行完成

CPU0：①读M_0　③读M_1

CPU1：②读M_1　④读M_0

例3：右图中，请求①与②、③与④可以并行完成

思考：并行访问方式与并行MEM的主要区别？

仅一组I/O信号线
(多次访问串行)

*性能：时延好(=T_M)，$B_M=(w/T_M)*m$，功耗小(1个体/次)

*应用：交叉编址—本地的多通道存储器(Core i7为3通道内存)

顺序编址—远程的共享存储器

*体冲突的避免：（性能优化）

软件方法—循环交换(内/外循环交换→访问不同的体)

硬件方法—存储体个数为质数(按行/列访问的是不同体)

第5节　虚拟存储器

※主要内容：VM的组织，VM的保护，MEM层次结构综合

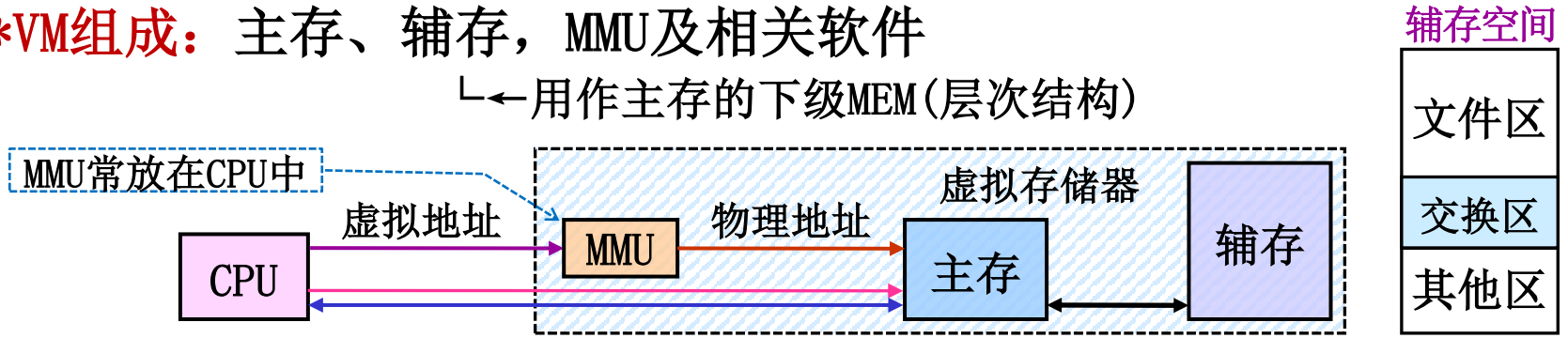
存储管理—含主存空间分配、存储空间扩充、进程空间保护
（分区/分页）　（覆盖/交换）　（区域/访问）

一、虚拟存储器的组织

1、虚拟存储器工作原理

*VM需求：程序自动装入 主存，CUP按程序地址访存
| └←OS负责(对程序员透明)　└←程序空间>>主存空间
└←存放在辅存(文件形式)中

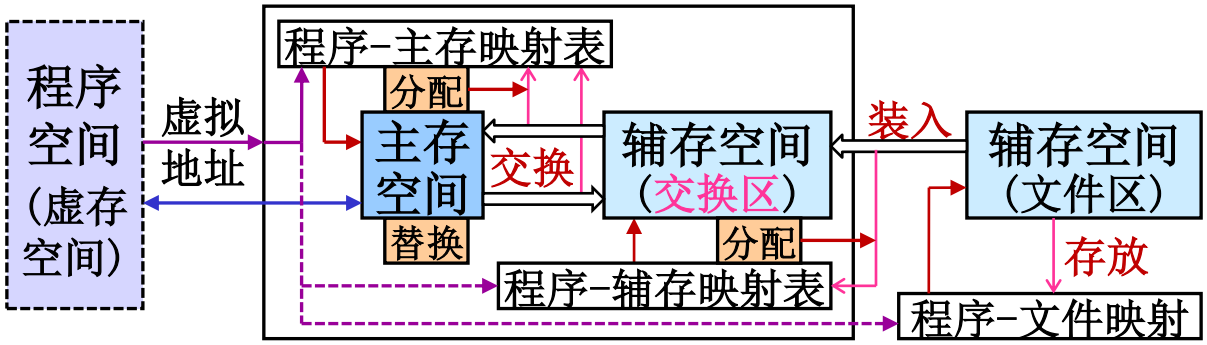
*VM组成：主存、辅存，MMU及相关软件
└←用作主存的下级MEM(层次结构)



策略—主存、辅存(交换区)用作VM的缓存，辅存兼作VM的宿主MEM

VM交换单元的状态— 缓存、未缓存、未分配

（主存中）（交换区中） （文件中）

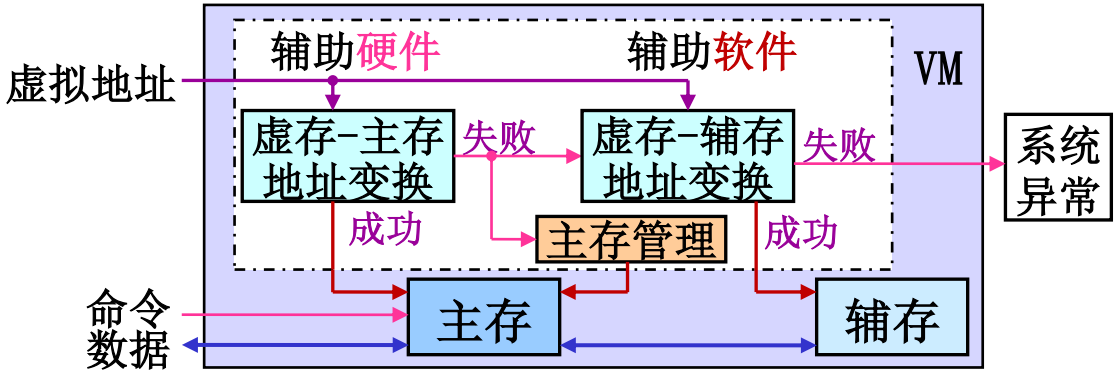


交换区：暂存替换出的程序信息

文件区：存放程序文件(与VM无关)

*VM工作原理：地址变换＋[层次管理＋]访问主存

←[]表示可缺省



VM的实质—是面向程序的存储器模型！

←程序MEM～物理MEM

2、虚拟存储器的组织

*虚存的存储管理方式：段式、页式、段页式　←基于交换单位

主存-辅存交换单位—程序段、程序页　←大小可变/固定

地址变换方法—段首址+段内偏移，〈页表[虚页号],页内偏移〉

*虚存-主存的层次管理：

映射方式—全相联　←提高主存利用率

映射表的存放：放在主存中　←降低成本(表空间巨大)

查找方法—按地址访问1次　←主存较慢

映射表的索引：用虚地址索引，行数＝$S_{程序}/S_{交换}$

思考：行数≠$S_{虚存}/S_{交换}$的实现要求？　保存表基址+表长，访问时判断

替换算法—伪LRU　←改状态成本高(表在主存/全相联)

写 策 略—写回法　←访存开销大

映射表项的组成：

段表项：

装入位	段首址	段长	访问位	修改位	…

页表项：

装入位	实页号	访问位	修改位	…

思考：映射表除表基址外，还需附加表长信息，
地址变换时判断长度。

*虚存-辅存的层次管理：（主存缺失时）

程序-辅存映射— 程序头表＋文件管理表

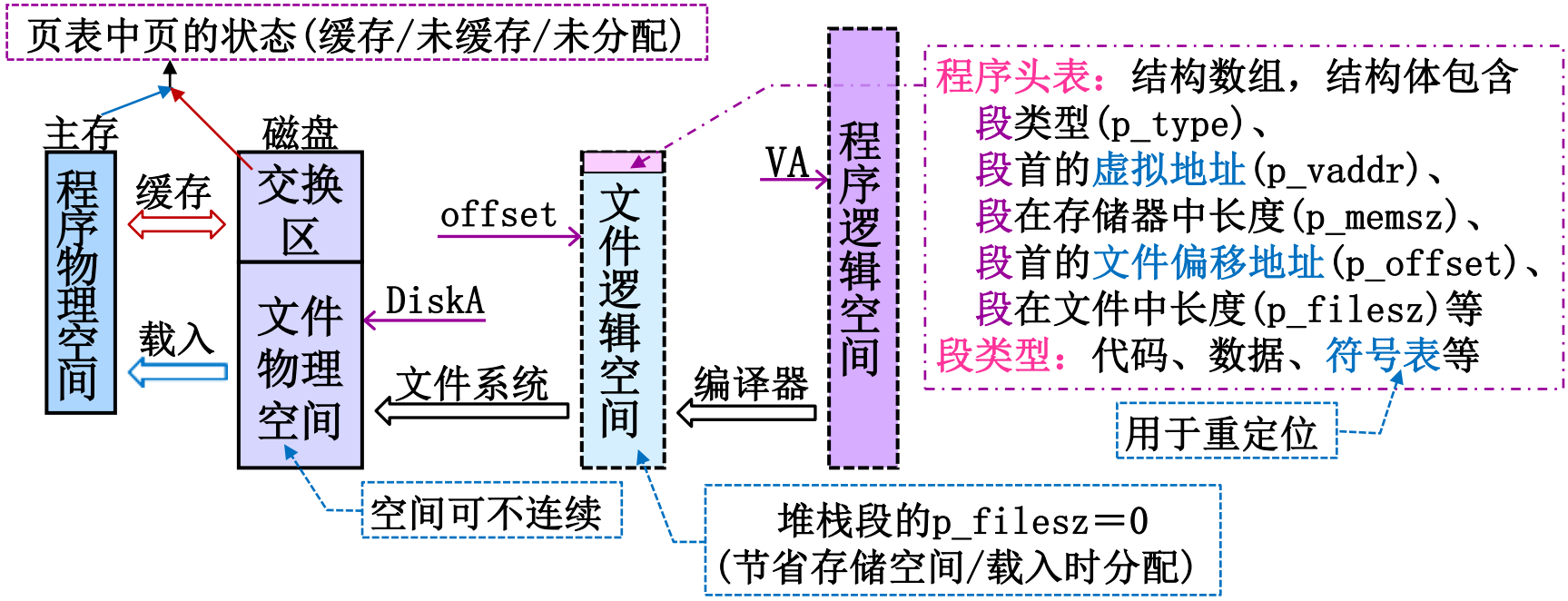
（程序→文件）（文件→辅存）

信息载入管理— 分配主存空间、拷贝数据、修改映射表项

└→可获得PA　　└←DiskA＝Func(fname,VA)

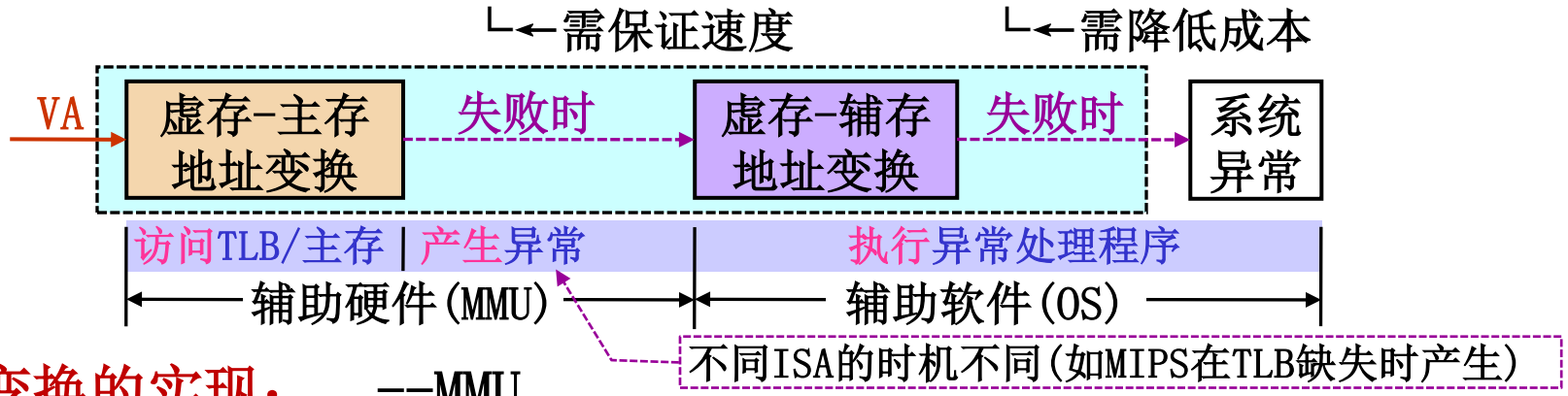
=<段号,段内偏移> →┘

信息交换管理— 同Cache缺失处理(调入或替换)



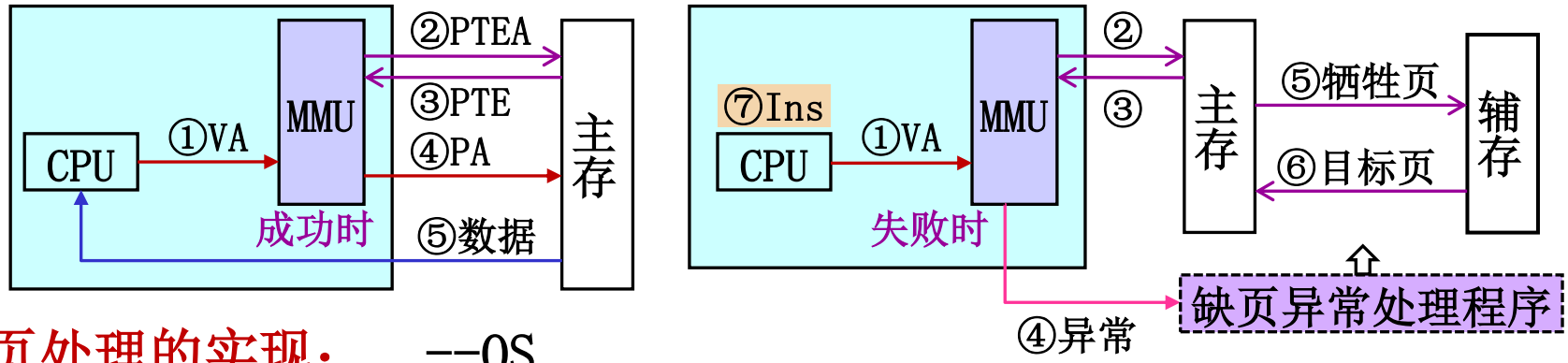
3、虚拟存储器的实现

*软/硬件功能分配：地址变换由MMU实现，其他由OS实现



*地址变换的实现：　--MMU

计算表项地址、读取表项、形成物理地址(或产生异常)



*缺页处理的实现：　--OS

执行异常处理程序＋返回到缺页的指令(重新执行)

PTEA--Page Table Entry Address

4、虚拟存储器的性能优化

$T_{VA}=T_{V命中}+F_{数据}\cdot T_{V缺失}$，$T_{V缺失}=T_{请求排队}+T_{事件响应}+T_{事件处理}$，

$T_{V命中}=(T_{表项地址计算}+T_{访存(表项)}+T_{地址变换})+T_{访存(数据)}$

*降低F的方法：增加$S_{主存}$，增加$S_{页}$，预取页式调度

*减少$T_{V缺失}$的方法：缺失作为异常事件($T_{请求排队}$↓)，←异常优先级＞中断

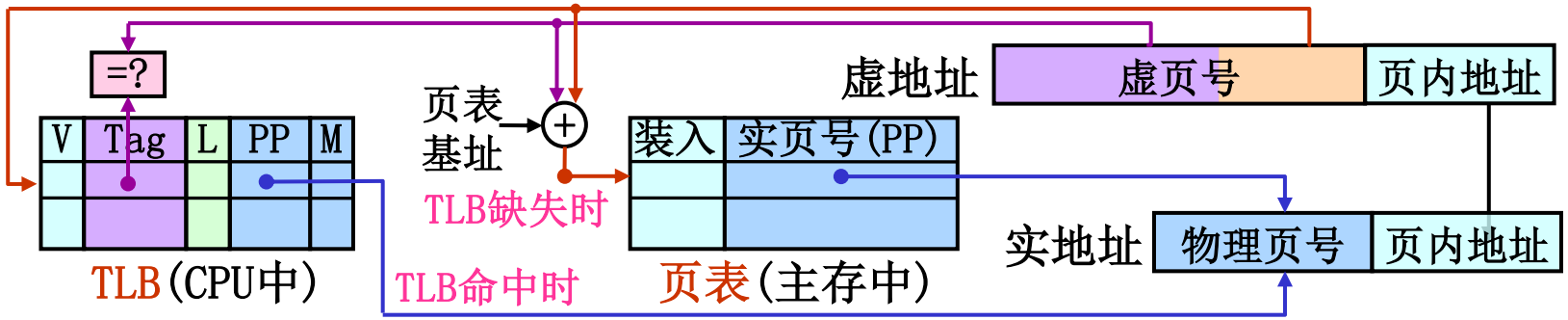
增设缺页向量寄存器($T_{事件响应}$↓)←获取入口无需访存

*减少$T_{V命中}$的方法：

增设快表TLB—$T_{访存(表项)}'=T_{TLB命中}+F_{TLB}\cdot T_{TLB缺失}$，$T_{TLB缺失}\approx T_{访存(表项)}$

$T_{TLB命中}$很小(TLB放在CPU中)，F_{TLB}很小(1行/页)

并行查TLB和页表—隐藏$T_{表项地址计算}$ ←故快表称为Translation Lookaside Buffer

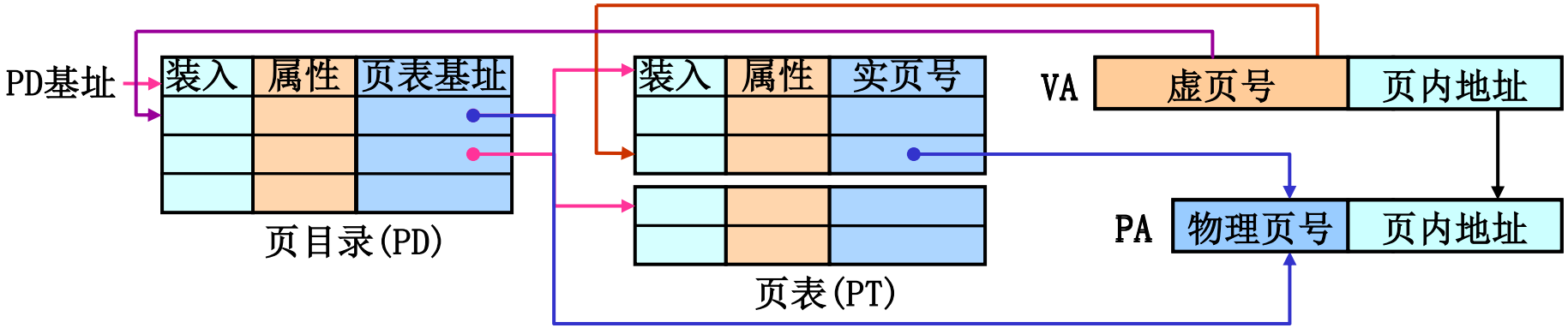


用硬件处理TLB缺失—减少$T_{TLB缺失}$ ←软件处理的理由是F_{TLB}极小、性/价较好

支持多种页大小—减少$T_{访存(表项)}$及$T_{地址变换}$次数

起因：页表常为多级($S_{页表} > S_{页}$时无法按虚页号索引[页框可不连续])

正常页地址变换：m级页表，$T_{访存(表项)}$及$T_{地址变换}$为m次

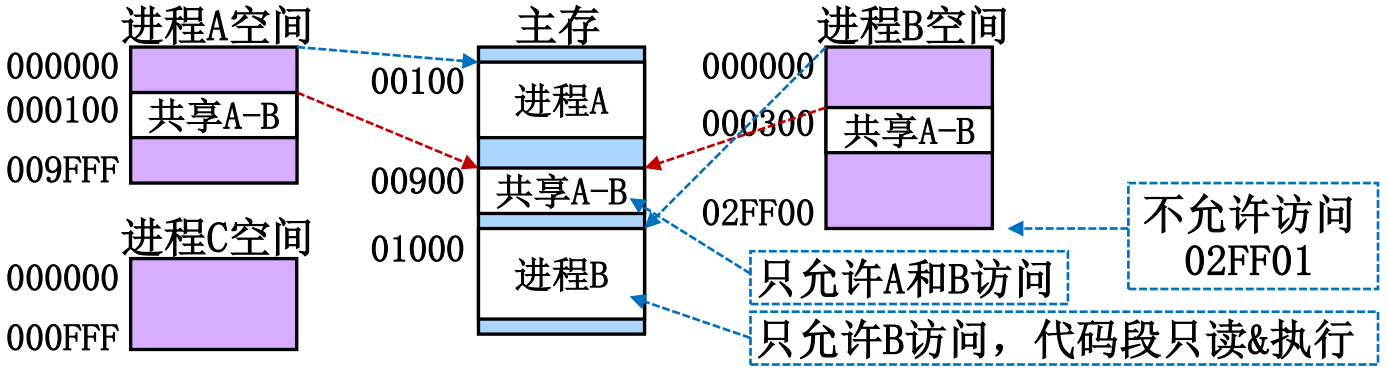


特大页地址变换：m-1级页表，$T_{访存(表项)}$及$T_{地址变换}$为m-1次

页大小的标识：页目录PD的属性中增加标志域

二、虚拟存储器的保护

*应用需求：进程的共享(同步&通信)及保护(安全)



共享方式— 访问(先打开)同一个 内存映像文件 ←OS还没讲 😭

- ├←创建/查找系统打开文件表表项、分配文件的主存缓存等
- └←创建进程打开文件表表项(含系统打开文件表索引)

└←基于进程打开文件表句柄，得到对应的主存缓存地址

保护类型— 区域保护(界限/共享)、访问保护(读/写/执行)

*实现策略：(1)进程的保护信息(区域/访问)均放在映像表中

(2)进程只能访问被授权的映像表 ← 如段表/页表

(3)地址变换时(∈处理请求)比较请求信息与保护信息

└——←MMU实现→——┘→结果为通过/异常

1、区域保护及其实现

*保护类型：界限(子区域&映像表)、共享授权(进程级别&指定进程)

*映像表保护：　--界限保护&私有信息保护

共享授权管理—

①进程的地址空间分为私有区域、共享区域

②进程的私有区域用私有映像表(如段表/文件打开表)管理　←1个/进程

└←包含共享区域的表项　←多种方案

所有的共享区域用公共映像表(如段表/文件打开表)管理　←1个/系统

└←创建映像文件时，公共表建立表项，私有表P=1、基址=公共表索引

③进程只能访问自身私有映像表、公共映像表　←共2个

保护方法—

①映像表项中含有该区域的界限(如段长)等保护信息

└←映像表含表长信息

②访存请求中带有地址的映像表类型(私有/共享)信息　←程序自知

③地址变换时选择映像表(2选1)、检查界限(对应子区域&映像表)

例1：若段内偏移为16位，进程A、B及公共的段表如下，进程B的2#段为共享段(用S表示)。假设共享段抽象为：通过公共段表的同一个段(源于内存映像文件)进行关联，进程段表项的段基址为公共段表相应段号。请求地址用<x, y>表示，x为段号，y为段内偏移地址，进程A的访问地址为<1, 0050H>、<0, 0D00H>、<2, 0050H>、<3, 0050H>、<4, 0050H>时，保护结果分别是什么？

进程A的段表

段号	装入	段基址	段长	…
0	1	0000H	3K	
1	1	1000H	2K	
2	0	1800H	2K	
3	1	0		S

进程B的段表

段号	装入	段基址	段长	…
0	1	2000H	3K	
1	1	2400H	2K	
2	1	0		S

系统公共的段表

段号	装入	段基址	段长	…
0	1	1400H	2K	
1	0	2800H	1K	

解：访问<1, 0050H>时， 通过，PA＝1000H＋0050H＝1050H

访问<0, 0D00H>时， 越界(0D00H＞段长)， MMU产生保护异常

访问<2, 0050H>时， 缺页(装入位＝0)， MMU产生缺页异常

访问<3, 0050H>时， 通过，PA＝1400H＋0050H＝1450H

访问<4, 0050H>时， 非法表项(4＞表长)， MMU产生保护异常

思考：表长放在哪里？

缺点—无法实现共享信息的保护

(如共享段只可被系统进程访问、不可被用户进程访问)

思考：表基址REG中包含表首址、表长信息

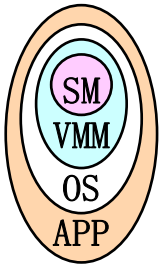
*环式保护：　　--共享的分级保护

级别管理—每个进程拥有环号(如放在其PSR中)，内层＞外层

保护方法—①映像表项中含有该区域的保护环号

②访存请求中带有该进程的请求环号

③地址变换时比较请求环号与保护环号



缺点—无法实现共享区域的指定保护(如同级别的2个进程间共享)

*键式保护：　　--共享的指定保护

配对管理—共享区域的拥有进程设有存储键(符号形式)，

访问进程使用访问键(符号形式)进行访问

保护方法—①公共映像表项中含有该区域的存储键(进程号)

②访存请求中带有该进程的访问键(进程号)

③地址变换时比较访问键与存储键

2、访问保护及其实现

*访问类型：读(R)、写(W)、执行(E)

*保护方法：①映像表项中含有该区域的允许类型

②地址变换时比较访问类型与允许类型

例2：续例1，若进程A的0#段及2#段为代码段、其余为数据段，进程A对地址<1,0050H>、<0,0050H>进行写操作时，保护结果是什么？

解：<1,0050H>∈数据段，通过(有效表项/不缺页/未越界/可访问)；

<0,0050H>∈代码段，非法访问，MMU产生保护异常

※虚存保护的常见应用

①同时采用区域保护(映像表/环状等)及访问保护

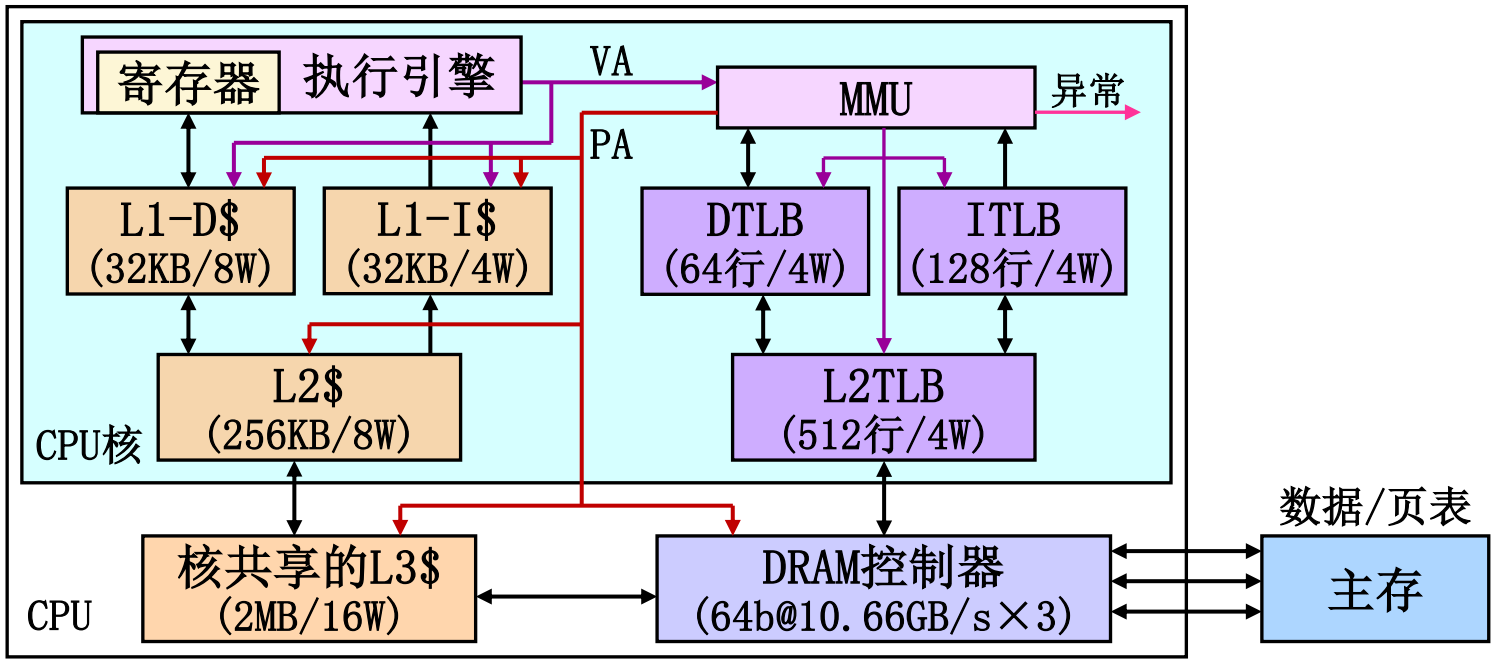
②映像表项中含所有保护信息(界限/环号/访问类型等)

③地址变换时进行(MMU实现)各种保护

保护模式—指采用虚拟存储器、提供信息保护机制的MEM管理模式！

三、存储器层次结构综合 （以Core i7为例）

*MEM层次结构： （TLB及Cache）



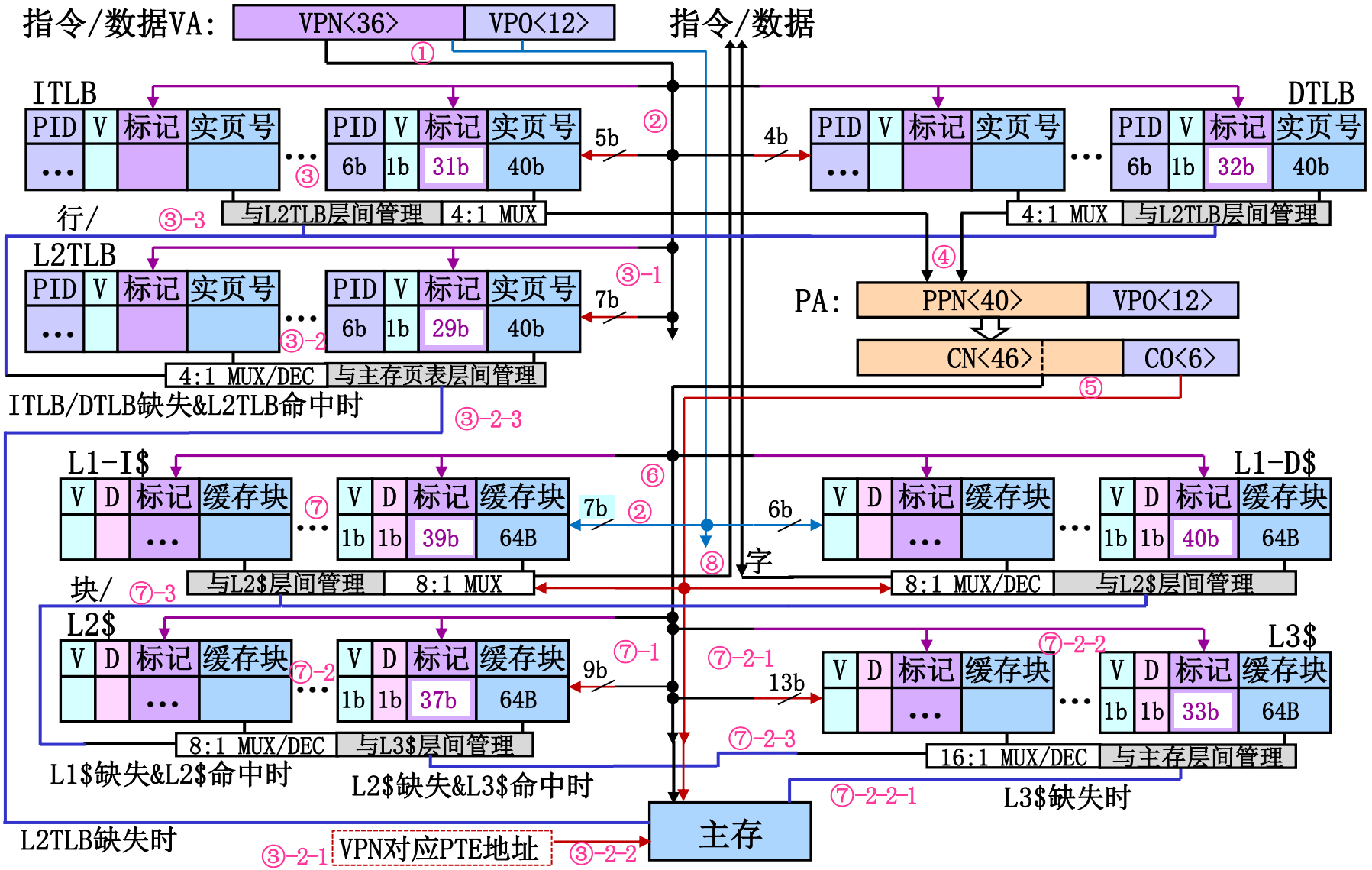
虚存参数—VA＝48b，PA＝52b，页式管理($S_{页}$＝4KB/4MB)

TLB参数—LRU、写回法，TLB缺失不产生异常(自动访问页表)

Cache参数—物理索引(L1$为虚拟索引)、$S_{块}$=64B、LRU、写回法

主存参数—DDR3，支持三通道方式

*MEM工作过程：地址变换(ITLB/DTLB+层间)＋数据访问(L1-I$/L1-D$+层间)



例3：若Core i7的TLB及Cache命中时间如下表所示，$T_{命中(L1\$)}$不包含与ITLB/DTLB重叠的时间，访问页表项的时延为80Tc，调入主存块的时延为95Tc。下列访存操作的最小时延分别是多少？

ITLB	DTLB	L2TLB
1Tc	1Tc	3Tc

L1-I$	L1-D$	L2$	L3$
1Tc	1Tc	6Tc	20Tc

(1)DTLB命中、L1-D$命中　　(2)L2TLB命中、L1-D$命中

(3)DTLB命中、L2$命中　　(4)L2TLB缺失、L3$命中

解：(1)T＝1Tc＋1Tc＝2Tc

(2)T≥(1Tc＋3Tc)＋1Tc＝5Tc　　←ITLB/DTLB可能有替换操作

(3)T≥1Tc＋(1Tc＋6Tc)＝8Tc　　←L1$可能有替换操作

(4)T≥(1Tc＋3Tc＋80Tc)＋(1Tc＋6Tc＋20Tc)＝111Tc

思考：是否会发生TLB缺失、Cache命中情况？

是否会发生TLB命中、Cache缺失、缺页的情况？

思考：会，PTE在主存中（TLB项被替换），数据已经调入Cache；不会，页装入主存→PTE有效(装入位=1)→TLB命中，不可能发生缺页现象

第五章课后复习思考题

(1)层次结构存储系统的产生原因是什么？为什么只有两个存储层次？不同层间信息交换单位大小为何不相同？

(2)如何实现组相联按地址并行查找？写一次法写策略的特征是什么？

(3)试画出采用读失效优先于写、尽早重启方案的非阻塞Cache结构图。

(4)虚存与Cache的实现技术有何异同？虚存访问的全过程有哪些环节？

(5)若页式虚存的页大小为4KB，TLB有4行，采用全相联映射、LRU替换算法。页表及TLB的初态如下，访问地址流为4669,2227,13916,34587,48870, 12608,49225。

虚页号	0	1	2	3	4	5	6	7	8	9	10	11
装入位	1	0	0	1	1	1	0	1	0	0	1	1
实页号	5	—	—	6	9	11	—	4	—	—	3	12

行号	0	1	2	3
有效位	1	1	1	0
标记	11	7	3	4
实页号	12	4	6	9
LRU位	2	1	0	3

a)给出访问后TLB内容，及TLB、页表命中次数。

b)若TLB采用2路组相联映射，初态如右表所示，给出访问后TLB内容，及TLB、页表命中次数。

行号	0	1	2	3
有效位	1	0	1	1
标记	5	7	1	5
实页号	3	4	6	12
LRU位	0	1	0	1

(6)相对于用硬件处理TLB缺失，TLB缺失用软件处理时的性能差别有哪些？

(7)优化VM性能的方法有哪些？虚存保护的种类、原理是什么？

(8)Pentium如何实现虚存保护的？

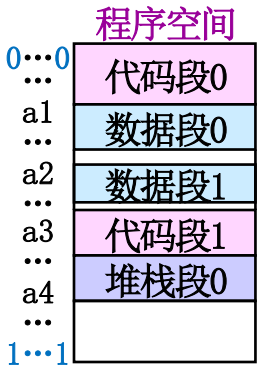
※附录1：程序组成基础

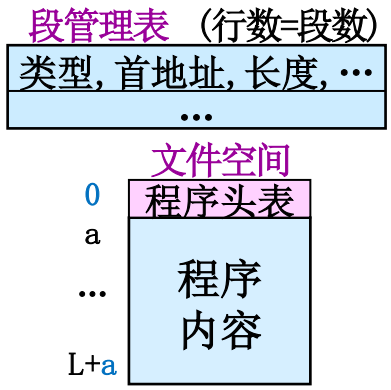
程序示例—

C语言程序：

```
void main(){
  int x,b,y;
  if (x==0)
     y=b;
  else
     y=b+2;
}
```

地址	程序内容	
0000	R[1]←M[1000]	代码
0001	R[2]←M[1001]	
0002	JNE R[1], 0004	
0003	R[2]←R[2]+2	
0004	M[1002]←R[2]	
0005	HALT(程序结束)	
1000	x	数据
1001	b	
1002	y	





思考①：程序中为何要设置程序地址？单元长度是多少？ 跳转指令&数据访问，同主存

引子：代码不能修改、数据不能执行，进程内/间需共享，对程序结构的要求？

程序结构—由多种段组成，每种段可有多个，各个段访问权限不同

程序地址组成—程序地址＝<段号，段内地址> ←段内地址从0开始

└←或段首地址(如=段号0…0)

程序地址空间—大小(即段号&段内地址位数)固定 ←便于系统级管理

程序的组成—程序内容＋段管理表 ←段长可变(面向存储)

程序的存放—程序头表＋程序内容 ←头表项含程序地址-文件地址映射

思考②：程序需分段，不同段有不同权限