第7章 存储系统

在理想情况下，我们希望存储容量无限大，这样任何一个特定的...字都可以立刻获取...我们...不得不认识到构建存储器层次结构的可能性，它们中的每一层都比其上一层具有更大的容量和更慢的访问速度。

A.W.Burks， H.H.Goldstine， 和J.von Neumann

电子计算设备逻辑设计预备讨论会（1946）

7.1 存储系统的基本知识

7.1.1 存储系统的层次结构

1. 计算机系统结构设计中关键的问题之一：

   如何以合理的价格，设计容量和速度都满足计算机系统要求的存储器系统？

2. 人们对这三个指标的要求

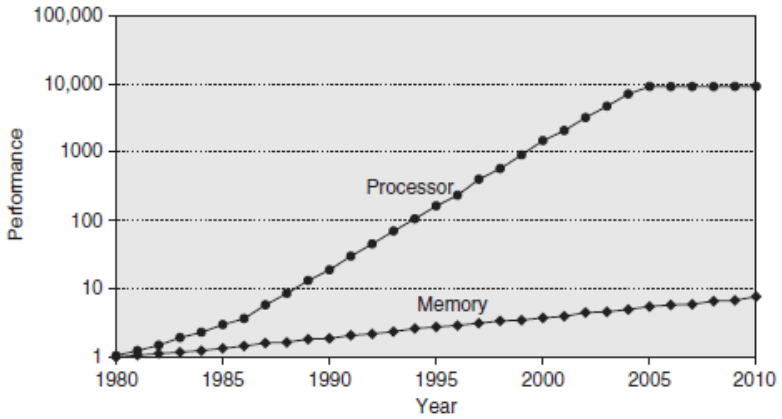
   容量大、速度快、价格低

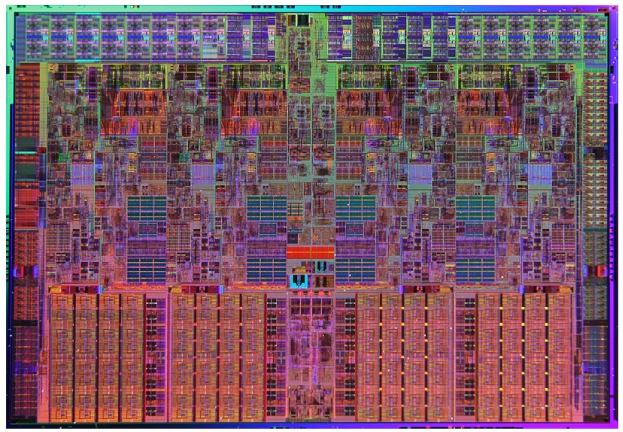
3. 三个要求是相互矛盾的

   - 速度越快，每位价格就越高
   - 容量越大，速度越慢
   - 容量越大，每位价格就越低

目标：每字节的成本和最便宜的相同
速度和最快的相同

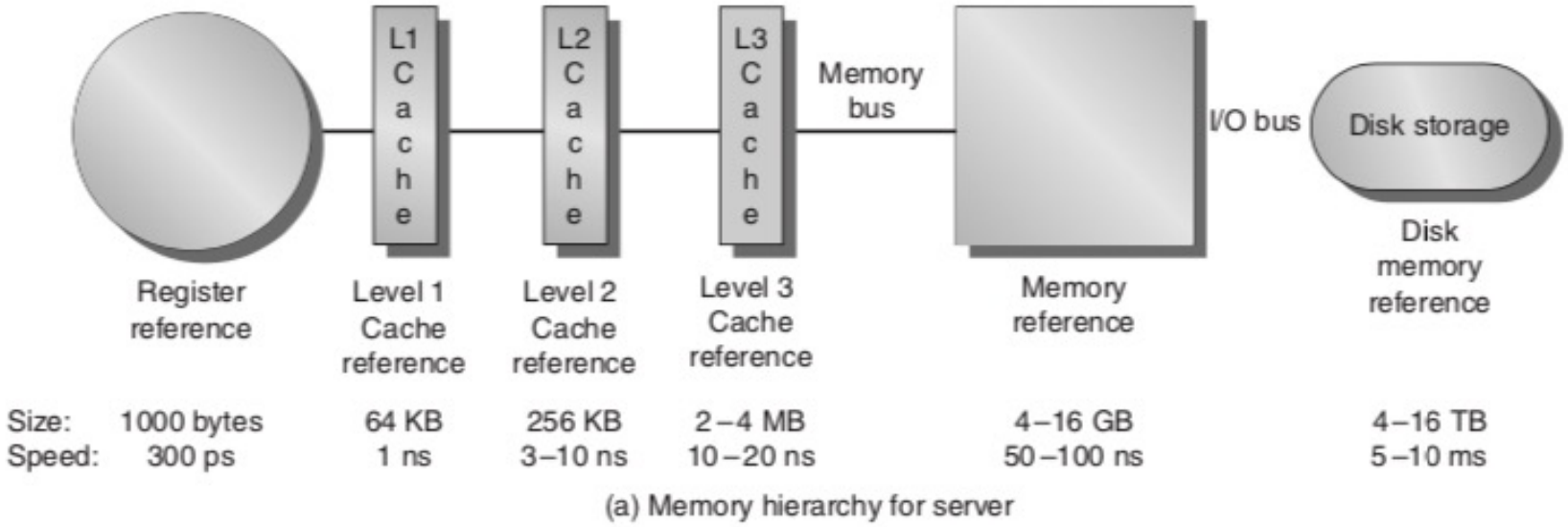
Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<1 KB	32 KB–8 MB	<512 GB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0.15–0.30	0.5–15	30–200	5,000,000
Bandwidth (MB/sec)	100,000–1,000,000	10,000–40,000	5000–20,000	50–500





4. 解决方法：采用多种存储器技术，构成多级存储层次结构。

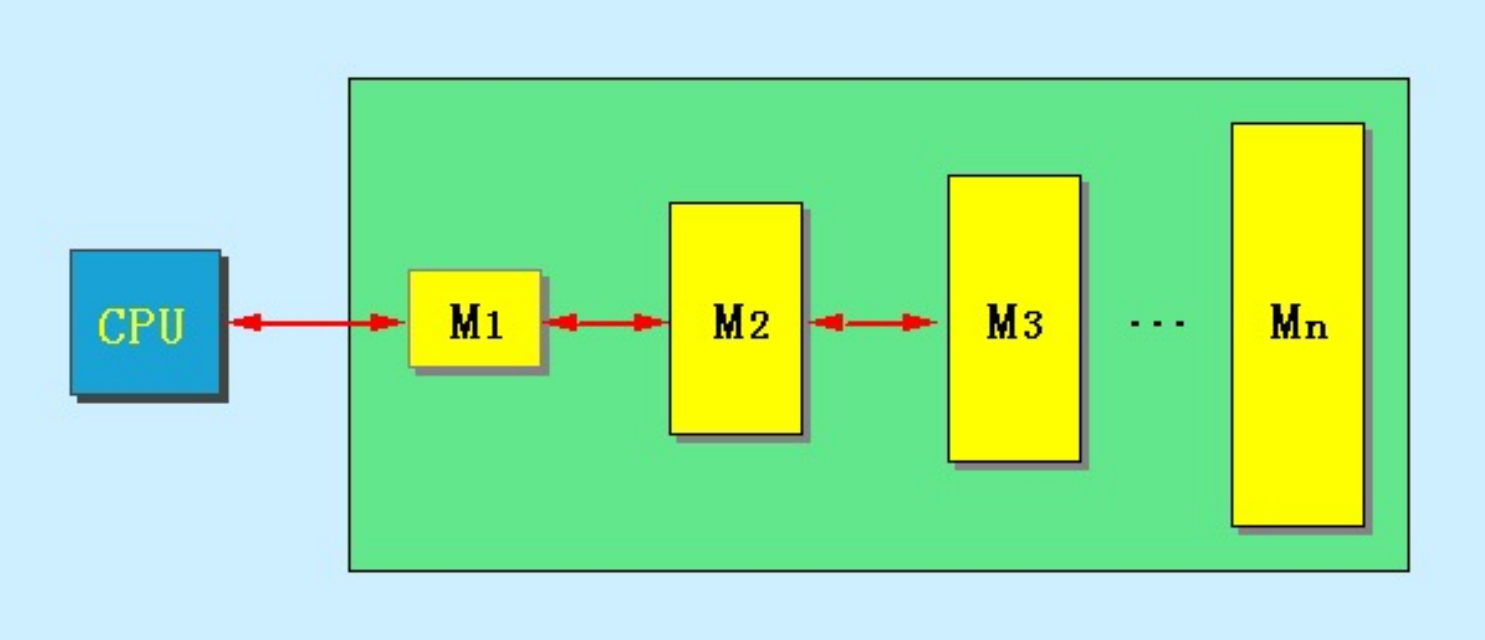
- 程序访问的局部性原理：对于绝大多数程序来说，程序所访问的指令和数据在地址上不是均匀分布的，而是相对簇聚的。
- 程序访问的局部性包含两个方面
  - 时间局部性：程序马上将要用到的信息很可能就是现在正在使用的信息。
  - 空间局部性：程序马上将要用到的信息很可能与现在正在使用的信息在存储空间上是相邻的。



(a) Memory hierarchy for server

不同层次的存取时间的变化10^9：从皮秒到毫秒之间变化；而容量大小的变化10^{12}：从几字节到几千兆字节之间变化。

5. 存储系统的多级层次结构



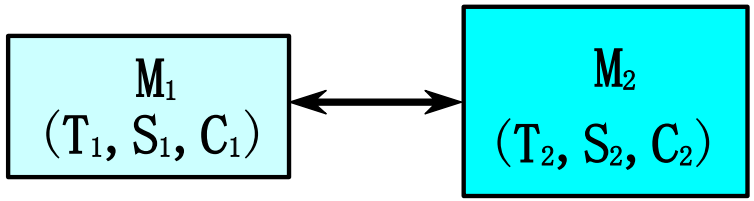
- 层次结构中的各层之间通常是**子集关系**
- 每一层都要从一个较大的存储器空间中把地址映射到一个较小、但是更快的、位于上一层的存储器中

- 假设第i个存储器M_i的访问时间为T_i，容量为S_i，平均每位价格为C_i，则
  - 访问时间：$T_1 < T_2 < \ldots < T_n$
  - 容量：$S_1 < S_2 < \ldots < S_n$
  - 平均每位价格：$C_1 > C_2 > \ldots > C_n$
- 整个存储系统要达到的目标：从CPU来看，该存储系统的速度接近于M_1的，而容量和每位价格都接近于M_n的。
  - 存储器越靠近CPU，则CPU对它的访问频度越高，而且最好大多数的访问都能在M_1完成。

7.1.2 存储层次的性能参数

下面仅考虑由M_1和M_2构成的两级存储层次：

- M_1的参数：S_1，T_1，C_1
- M_2的参数：S_2，T_2，C_2



1. 存储容量S

   - 一般来说，整个存储系统的容量即是第二级存储器M_2的容量，即$S=S_2$。

2. 每位价格C

$$C=\frac{C_1S_1+C_2S_2}{S_1+S_2}$$

当$S_1 \ll S_2$时，$C \approx C_2$。

3. 命中率*H* 和不命中率*F*

- 命中率：CPU访问存储系统时，在M_1中找到所需信息的概率。

$$H=\frac{N_1}{N_1+N_2}$$

  - N_1 —— 访问M_1的次数
  - N_2 —— 访问M_2的次数

- 不命中率（缺失率）：F＝1－H

4. 平均访问时间T_A

$$T_A = HT_1 + (1-H)(T_1 + T_M)$$
$$= T_1 + (1-H)T_M$$
$$\text{或 } T_A = T_1 + FT_M$$

分两种情况来考虑CPU的一次访存：

- 当命中时，访问时间即为T_1（命中时间）
- 当不命中时，情况比较复杂。

  不命中时的访问时间为：$T_M = T_2 + T_B$

  - 不命中开销T_M：从向M_2发出访问请求到把整个数据块调入M_1中所需的时间。
  - 传送一个信息块所需的时间为T_B。

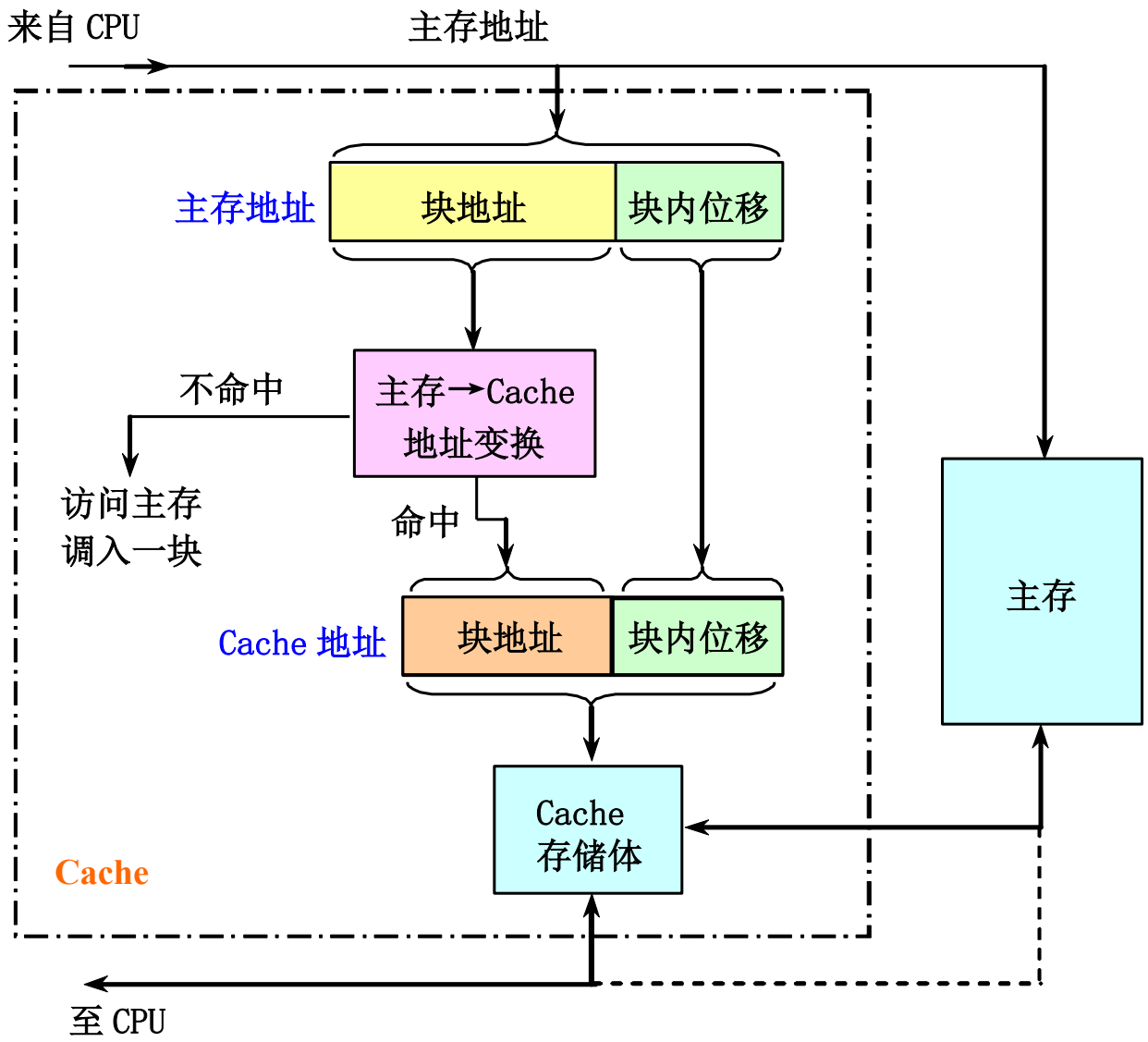
7.2 Cache基本知识

7.2.1 基本结构和原理

1. Cache和主存分块

- Cache是按块进行管理的。Cache和主存均被分割成大小相同的块。信息以块为单位调入Cache。
  - 主存块地址（块号）用于查找该块在**Cache**中的位置。
  - 块内位移用于确定所访问的数据在该块中的位置。

3. Cache的基本工作原理示意图



存储层次的四个问题

1. 当把一个块调入高一层(靠近CPU)存储器时，可以放在哪些位置上?

   （映象规则）

2. 当所要访问的块在高一层存储器中时，如何找到该块?（用低层的地址在高层存储器中查找）

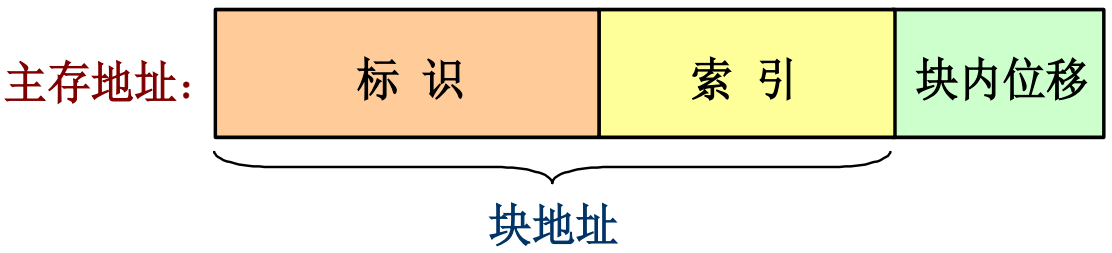
   （查找算法）

3. 当发生不命中时，应替换哪一块？

   （替换算法）

4. 当进行写访问时，应进行哪些操作?

   （写策略）



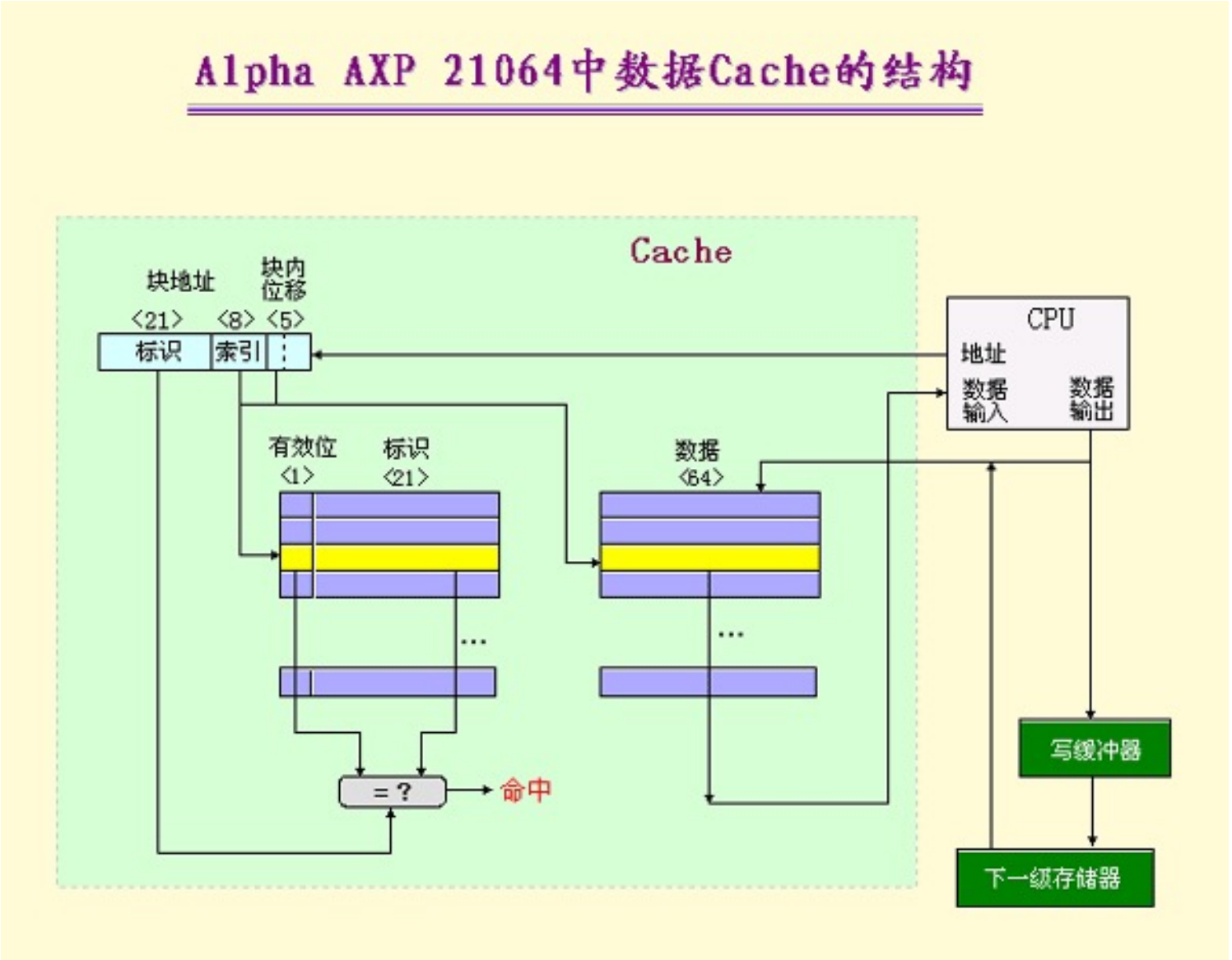
7.2.6 Cache的工作过程

例子：DEC的Alpha AXP21064中的内部数据Cache

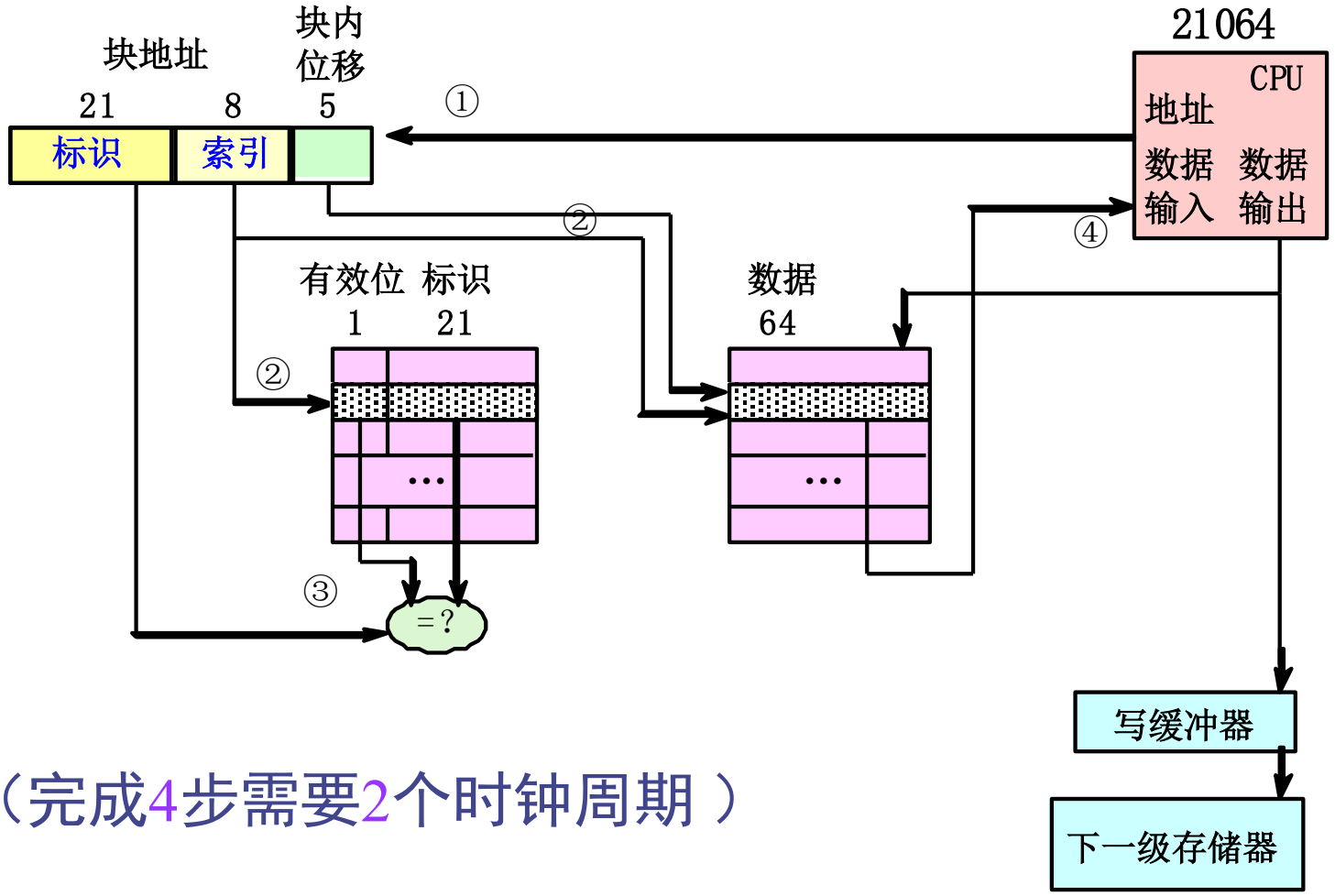
1. 简介

- 容量：8KB
- 块大小：32B
- 块数：256
- 映象方法：直接映象
- 写策略：写直达+不按写分配

2. 结构图


Alpha AXP 21064中数据Cache的结构
Cache
块地址
块内位移
〈21〉
〈8〉
〈5〉
标识
索引
有效位
〈1〉
标识
〈21〉
数据
〈64〉
CPU
地址
数据输入
数据输出
写缓冲器
下一级存储器
= ?
命中

3. 工作过程



（完成4步需要2个时钟周期）

3. 工作过程

① 处理器传送给Cache物理地址

- **Cache的容量与索引index、相联度、块大小之间的关系**

  Cache的容量=2^{index}×相联度×块大小

- **把容量为8192、相联度为1、块大小为32（字节）代入：**

  索引index：8位　　标识：29－8＝21位

3. 工作过程

② 由索引选择标识的过程

- 根据索引从目录项中读出相应的标识和有效位

③ 从Cache中读出标识之后，用来同从CPU发来的块地址中标志域部分进行比较

- 为了保证包含有效的信息，必须要设置有效位

④ 如果有一个标识匹配，且标志位有效，则此次命中

- 通知**CPU**取走数据

➢ 共需2个周期

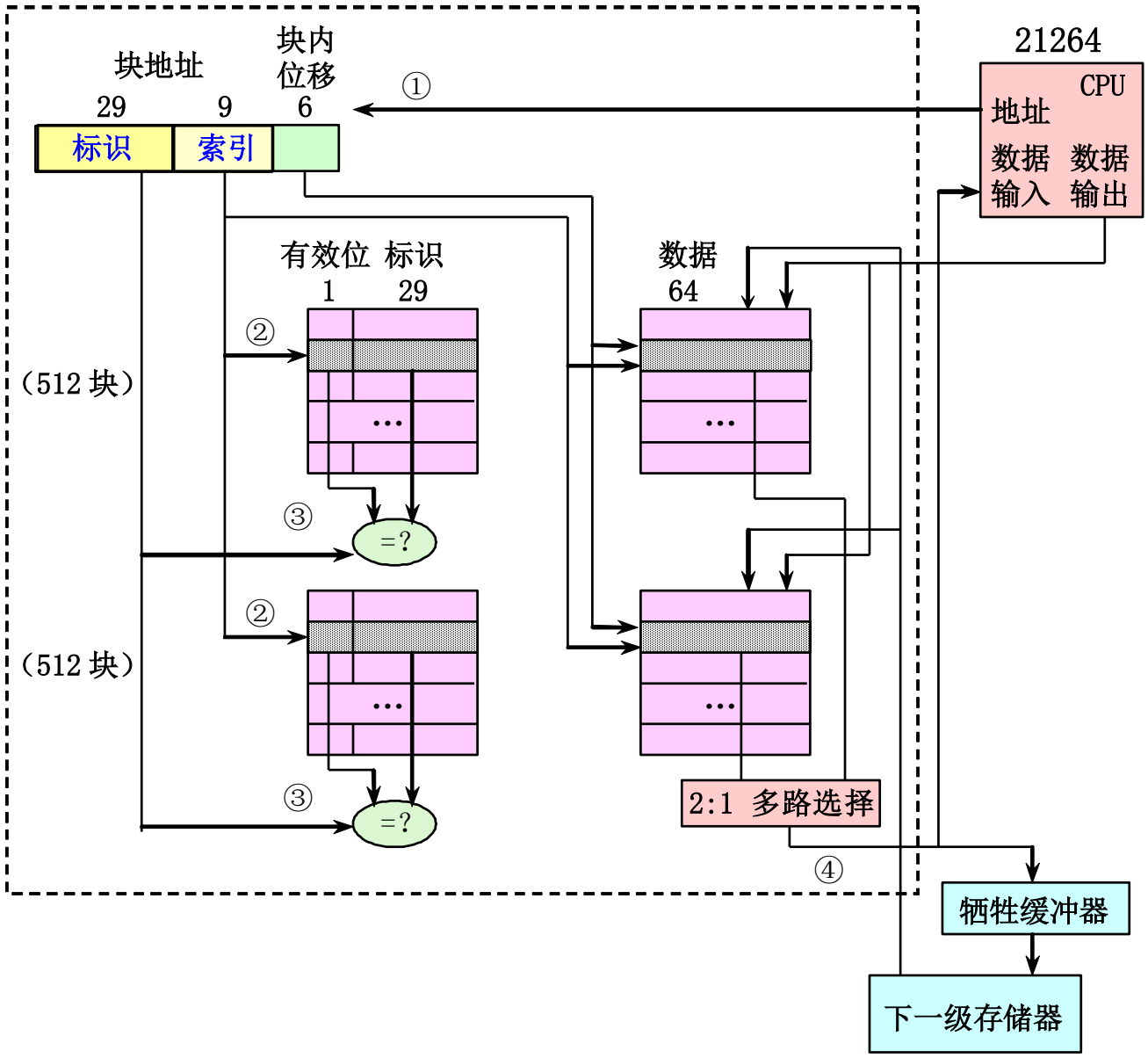
- “写”访问命中
  - 前三步一样，只有在确认标识匹配后才把数据写入
- 设置了一个写缓冲器

  （提高“写”访问的速度）
  - 按字寻址的，它含有4个块，每块大小为4个字。
  - 当要进行写入操作时，如果写缓冲器不满，那么就把数据和完整的地址写入缓冲器。对CPU而言，本次“写”访问已完成，CPU可以继续往下执行。由写缓冲器负责把该数据写入主存。
  - 在写入缓冲器时，要进行写合并检查。即检查本次写入数据的地址是否与缓冲器内某个有效块的地址匹配。如果匹配，就把新数据与该块合并 。

- 发生读不命中与写不命中时的操作
  - 读不命中：向CPU发出一个暂停信号，通知它等待，并从下一级存储器中新调入一个数据块（32字节）
    - Cache与下一级存储的数据通路宽度为16B，传送一次需5个周期，因此，一次传送需要10个周期
  - 写不命中：将使数据“绕过”Cache，直接写入主存。
    - 写直达 –不按写分配
  - 因为是写直达，所以替换时不需要写回

对比：Alpha AXP 21264的数据Cache结构

- 容量：64KB　块大小：64字节　FIFO替换策略
- 主要区别
  - 采用2路组相联
  - 采用写回法
- 没有写缓冲器


块地址
块内
位移
29
9
6
标识
索引
①
21264
CPU
地址
数据
输入
数据
输出
有效位
标识
1
29
数据
64
②
③
=?
(512 块)
(512 块)
2:1 多路选择
④
牺牲缓冲器
下一级存储器

- 最后一步是通知CPU根据2选1多路选择器的有效输出从Cache中加载正确数据
- 21264允许在3个时钟周期内完成这四个步骤
  - 如果在随后两个时钟周期内的指令要使用这些数据的话，那么它们需要等待加载数据完成
- 由于21264是乱序执行的，只有等到指令提交并且Cache标志检验结果是命中时，数据才被写到Cache中

7.2.7 Cache的性能分析

1. 不命中率
   - 与硬件速度无关
   - 容易产生一些误导
2. 平均访存时间

平均访存时间 = 命中时间＋不命中率×不命中开销

- 可以采用绝对时间——如一次命中需0.25-1.0纳秒
- 或是用CPU等待存储器的时钟周期数——如缺失代价用75-100个时钟周期表示
- 仍旧不能代替执行时间

例　分体Cache与一体Cache

- 一个16K指令Cache和16K数据Cache与一个32K的一体Cache相比较，哪一个具有更低的缺失率？
- 假设36%的指令是数据传输指令，缺失率如图所示。假定Cache命中需要1个时钟周期，缺失代价是100个时钟周期，在一体Cache 中，Load和Store命中额外需要一个时钟周期，因为只有一个Cache端口来满足这两个同时发生的请求。使用上一章提到流水线的术语，一体Cache导致结构冲突。每种情况下的平均存储器存取时间是多少呢？假定是带有写缓存的写直达Cache，而且写缓存的停顿时间可以忽略不记

容量	指令CACHE	数据CACHE	一体CACHE
8KB	8.16	44.0	63.0
16KB	3.82	40.9	51.0
32KB	1.36	38.4	43.3
64KB	0.61	36.9	39.4
128KB	0.30	35.3	36.2
256KB	0.02	32.6	32.9

不同容量指令Cache、数据Cache以及一体Cache的每千条指令缺失率。数据Cache块大小是64字节，采用2-路组相联映射方式。

一条指令只进行一次取指令的访存操作，故指令的缺失率为：

$$\text{缺失率}_{16k\text{指令}cache} = \frac{3.82/1000}{1.00} = 0.004$$

由题目的假设可知36%的指令是数据操作指令，故数据缺失率为：

$$\text{缺失率}_{16k\text{数据}cache} = \frac{40.9/1000}{0.36} = 0.114$$

$$\text{缺失率} = \frac{\text{缺失次数}}{\text{1000条指令}} \Big/ 1000 = \frac{\text{内存存取次数}}{\text{指令数}}$$

大约74%的存储器访问是访问指令的。因此，分立Cache的总的缺失率为：

$$(74\% \times 0.004) + (26\% \times 0.114) = 0.0324$$

一体Cache的缺失率要包括指令和数据的缺失率：

$$\text{缺失率}_{32k\text{一体}cache} = \frac{43.3/1000}{1.00 + 0.36} = 0.0318$$

$$\text{平均存储器存取时间} = \text{指令所占几率} \times (\text{命中时间} + \text{指令缺失率} \times \text{缺失代价}) + \text{数据所占几率} \times (\text{命中时间} + \text{数据缺失率} \times \text{缺失代价})$$

$$\text{平均存储器存取时间}_{\text{分立}cache} = 74\% \times (1 + 0.004 \times 100) + 26\% \times (1 + 0.114 \times 100)$$

$$= (74\% \times 1.38) + (26\% \times 12.36) = 1.023 + 3.214 = 4.24$$

$$\text{平均存储器存取时间}_{\text{一体}cache} = 74\% \times (1 + 0.0318 \times 100) + 26\% \times (1 + 1 + 0.0318 \times 100)$$

$$= (74\% \times 4.18) + (26\% \times 5.18) = 3.096 + 1.348 = 4.44$$

因此，本例中分离Cache——在每个时钟周期提供两个存储器端口，因此可以避免结构冲突——比只有一个端口的一体Cache有更好的平均存储器存取时间，尽管它的有效缺失率更高一些。

3. 程序执行时间

CPU时间＝（CPU执行周期数+存储器停顿周期数）× 时钟周期时间

其中：

- 存储器停顿时钟周期数＝“读”的次数×读不命中率×读不命中开销＋“写”的次数×写不命中率×写不命中开销
- 存储器停顿时钟周期数＝访存次数×不命中率×不命中开销

CPU时间＝（CPU执行周期数+访存次数×不命中率×不命中开销）

× 时钟周期时间

$$CPU\text{时间} = IC \times \left(CPI_{execution} + \frac{\text{访存次数}}{\text{指令数}} \times \text{不命中率} \times \text{不命中开销} \right) \times \text{时钟周期时间}$$

$$= IC \times （CPI_{\text{execution}} + \text{每条指令的平均访存次数} \times \text{不命中率}$$

×不命中开销）× 时钟周期时间

例7.1 用一个和Alpha AXP类似的机器作为第一个例子。假设Cache不命中开销为50个时钟周期，当不考虑存储器停顿时，所有指令的执行时间都是2.0个时钟周期，访问Cache不命中率为2%，平均每条指令访存1.33次。试分析Cache对性能的影响。

解

$$\text{CPU时间}_{\text{有cache}} = IC \times (CPI\text{execution} + \text{每条指令的平均访存次数} \times \text{不命中率} \times \text{不命中开销}) \times \text{时钟周期时间}$$

$$= IC \times (2.0 + 1.33 \times 2\% \times 50) \times \text{时钟周期时间}$$

$$= IC \times 3.33 \times \text{时钟周期时间}$$

考虑Cache的不命中后，性能为：

$CPU时间_{有cache}$＝IC×3.33×时钟周期时间

实际*CPI* ： 3.33

3.33/2.0 = 1.67(倍)

CPU时间也增加为原来的1.67倍。

但若不采用Cache,则：

CPI＝2.0＋50×1.33＝68.5

4. Cache不命中对于一个CPI较小而时钟频率较高的CPU来说，影响是双重的：

   - $CPI_{execution}$越低，固定周期数的Cache不命中开销的相对影响就越大。
   - 在计算CPI时，不命中开销的单位是时钟周期数。因此，即使两台计算机的存储层次完全相同，时钟频率较高的CPU的不命中开销较大，其CPI中存储器停顿这部分也就较大。

     因此Cache对于低CPI、高时钟频率的CPU来说更加重要。

- 虽然追求最小平均存储器存取时间是一个合理的目标
- 但是要注意最终目标是降低CPU执行时间。

例7.2 考虑两种不同组织结构的Cache：直接映象Cache和两路组相联Cache，试问它们对CPU的性能有何影响？先求平均访存时间，然后再计算CPU性能。分析时请用以下假设：

(1)理想Cache（命中率为100%）情况下的CPI为2.0，时钟周期为2ns，平均每条指令访存1.3次。

(2)两种Cache容量均为64KB，块大小都是32字节。

(3)在组相联Cache中，由于多路选择器的存在而使CPU的时钟周期增加到原来的1.10倍。这是因为对Cache的访问总是处于关键路径上，对CPU的时钟周期有直接的影响。

(4) 这两种结构Cache的不命中开销都是70ns。（在实际应用中，应取整为整数个时钟周期）

(5) 命中时间为1个时钟周期，64KB直接映象Cache的不命中率为1.4%，相同容量的两路组相联Cache的不命中率为1.0%。

解 平均访存时间为：

平均访存时间＝命中时间＋不命中率×不命中开销

因此，两种结构的平均访存时间分别是：

平均访存时间$_{1路}$＝2.0＋（0.014×70）＝2.98ns

平均访存时间$_{2路}$＝2.0×1.10＋（0.010×70）＝2.90ns

两路组相联Cache的平均访存时间比较低。

CPU时间＝*IC*×（*CPI*execution＋每条指令的平均访存次数×不命中率×不命中开销）× 时钟周期时间

＝*IC*×（*CPI*execution× 时钟周期时间＋每条指令的平均访存次数×不命中率×不命中开销×时钟周期时间）

因此：

$$\text{CPU时间}_{1路} = IC\times(2.0\times2+(1.3\times0.014\times70)) = 5.27\times IC$$

$$\text{CPU时间}_{2路} = IC\times(2.0\times2\times1.10+(1.3\times0.010\times70)) = 5.31\times IC$$

$$\frac{\text{CPU时间}_{2路}}{\text{CPU时间}_{1路}} = \frac{5.31\times IC}{5.27\times IC} = 1.01$$

与平均存储器存取时间的比较结果相反，直接映象Cache的平均性能好一些。

- 2-路组相联情况下，尽管它的缺失率低一些，但是所有指令时钟周期都延长了。

7.2.8 改进Cache的性能

1. 平均访存时间＝命中时间＋不命中率×不命中开销
2. 可以从三个方面改进Cache的性能：
   - 降低不命中率
   - 减少不命中开销
   - 减少Cache命中时间

7.3 降低Cache不命中率

7.3.1 三种类型的不命中

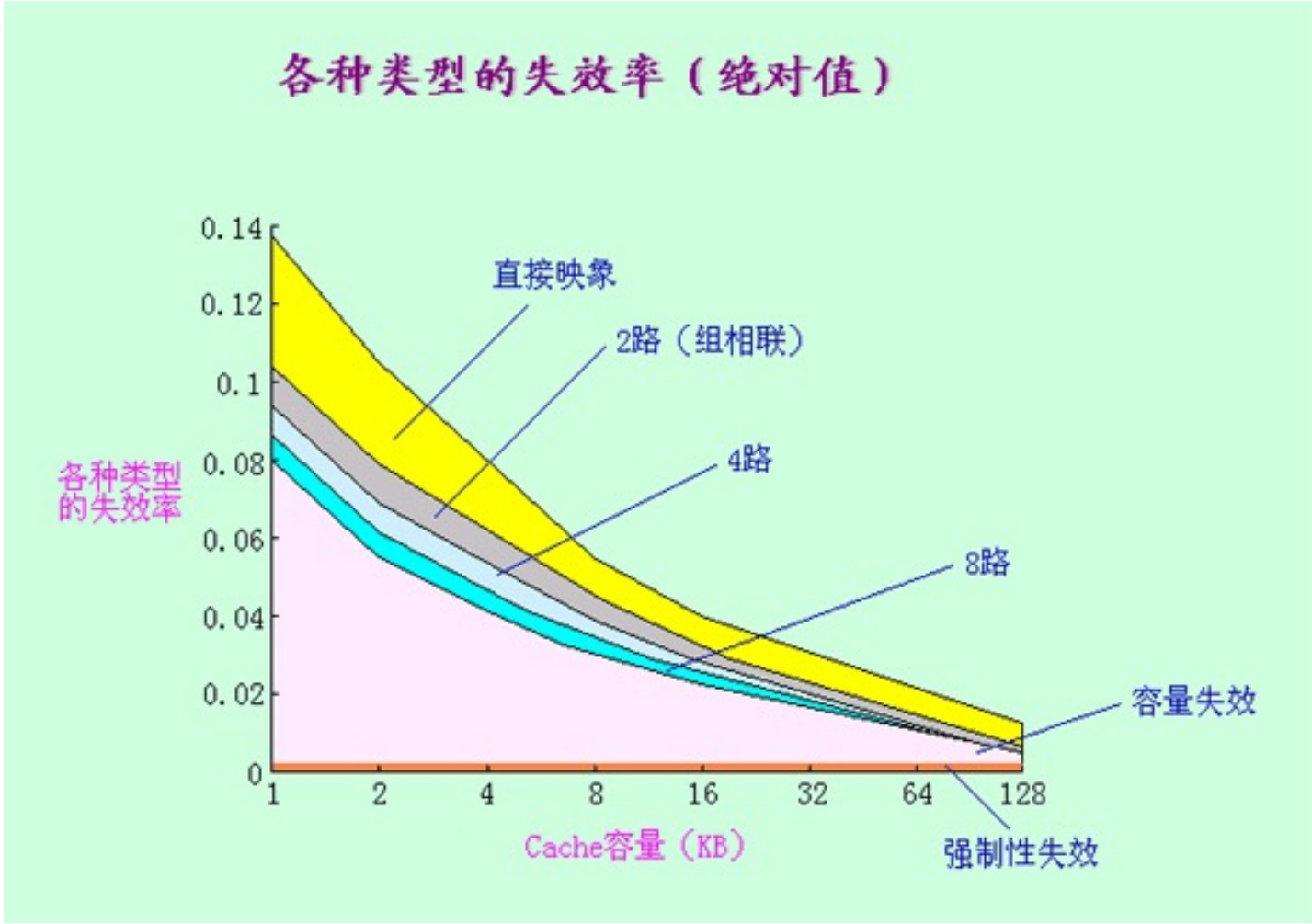
1. 三种类型的不命中(3C)

- 强制性不命中(Compulsory miss)
  - 当第一次访问一个块时，该块不在Cache中，需从下一级存储器中调入Cache，这就是强制性不命中。(冷启动不命中，首次访问不命中)
- 容量不命中(Capacity miss)
  - 如果程序执行时所需的块不能全部调入Cache中，则当某些块被替换后，若又重新被访问，就会发生不命中。这种不命中称为容量不命中。

- 冲突不命中(Conflict miss)
  - 在组相联或直接映象Cache中，若太多的块映象到同一组(块)中，则会出现该组中某个块被别的块替换(即使别的组或块有空闲位置)，然后又被重新访问的情况。这就是发生了冲突不命中。

    (碰撞不命中，干扰不命中)

7.3 降低Cache不命中率



- 可以看出：
  - 相联度越高，冲突不命中就越少；
  - 强制性不命中和容量不命中不受相联度的影响；
  - 强制性不命中不受Cache容量的影响，但容量不命中却随着容量的增加而减少。

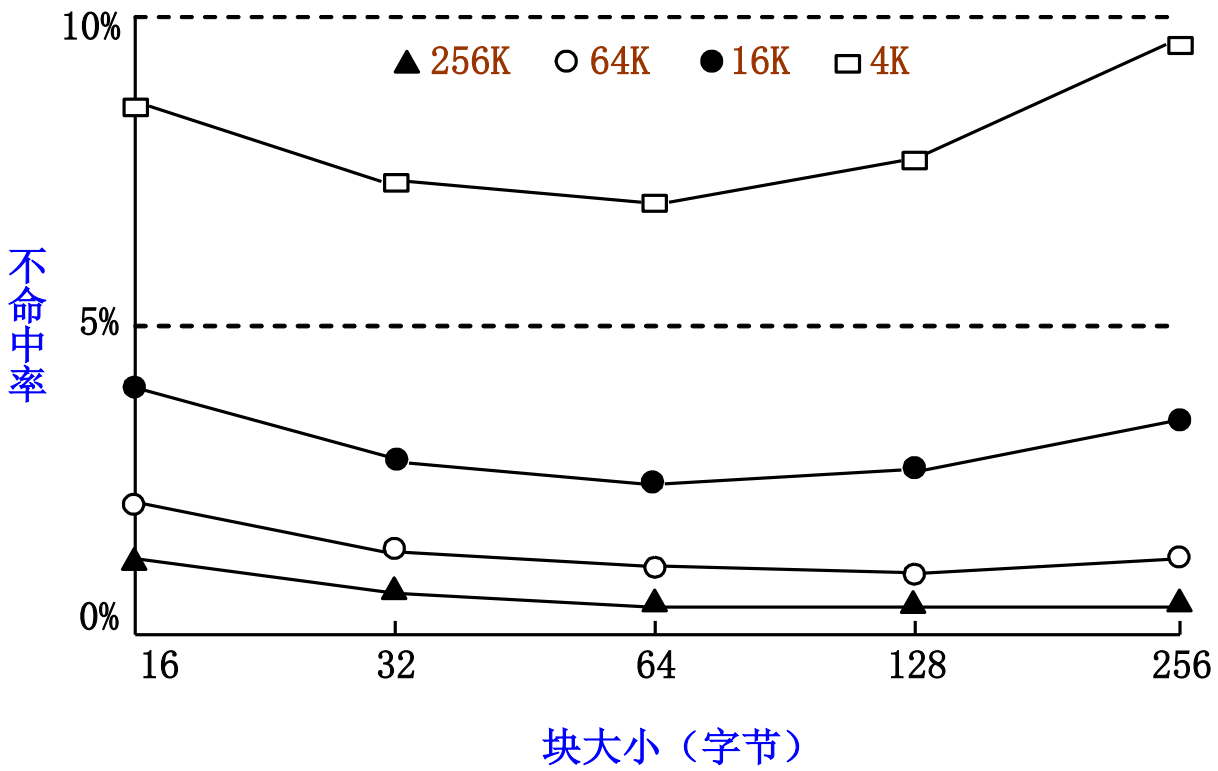
- 减少三种不命中的方法
  - 强制性不命中：增加块大小
  - 容量不命中：增加容量
  - 冲突不命中：提高相联度

（理想情况：全相联）

- 许多降低不命中率的方法会增加命中时间或不命中开销
- 采用技术降低缺失率的同时，必须与提高系统整体性能的目标进行平衡

7.3.2 增加Cache块大小

1. 不命中率与块大小的关系



7.3.2 增加Cache块大小

1. 不命中率与块大小的关系

- 对于给定的Cache容量，当块大小增加时，不命中率开始是下降，后来反而上升了。

  原因：

  - 一方面它减少了强制性不命中（利用了空间的局部性原理）；
  - 另一方面，由于增加块大小会减少Cache中块的数目，所以有可能会增加冲突不命中。
  - 如果**Cache**容量较小的话，甚至会有容量缺失

- Cache容量越大，使不命中率达到最低的块大小就越大。

7.3.3 增加Cache的容量

1. 最直接的方法是增加Cache的容量

- 缺点:
  - 增加成本
  - 可能增加命中时间

7.3.4 提高相联度

1. 采用相联度超过8的方案的实际意义不大。
   - 从实际上讲，8路组相联和相同容量的全相联Cache在降低缺失次数方面同样有效

7.3.4 提高相联度

1. 采用相联度超过8的方案的实际意义不大。
   - 从实际上讲，8路组相联和相同容量的全相联Cache在降低缺失次数方面同样有效
2. 2:1 Cache经验规则（128K以内）

   容量为N的直接映象Cache的不命中率和容量为N/2的两路组相联Cache的不命中率差不多相同。
3. 提高相联度是以增加命中时间为代价。

7.3.5 伪相联 Cache (列相联 Cache)与路预测

1. 多路组相联的低不命中率和直接映象的命中速度

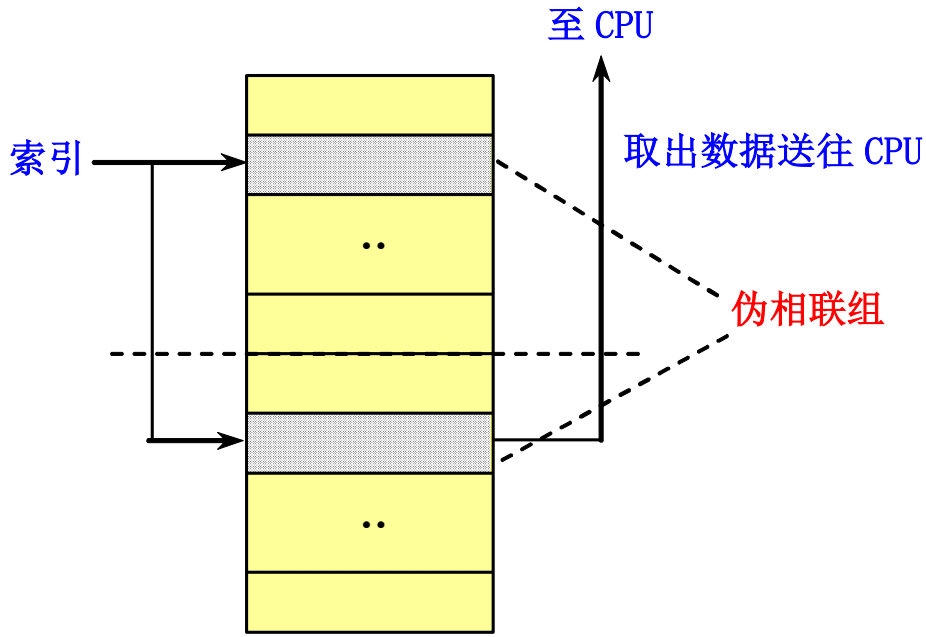
	优　点	缺　点
直接映象	**命中时间小**	**不命中率高**
组相联	**不命中率低**	**命中时间大**

2. 伪相联Cache的优点

- 命中时间小
- 不命中率低

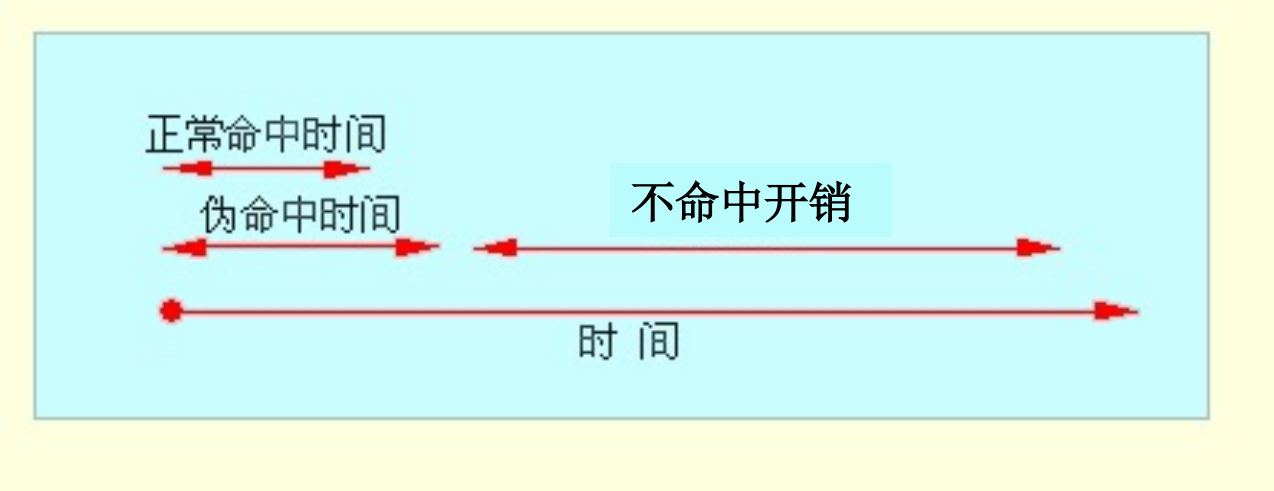
3. 基本思想及工作原理

在逻辑上把直接映象Cache的空间上下平分为两个区。对于任何一次访问，伪相联Cache先按直接映象Cache的方式去处理。若命中，则其访问过程与直接映象Cache的情况一样。若不命中，则再到另一区相应的位置去查找。若找到，则发生了伪命中，否则就只好访问下一级存储器。



4. 快速命中与慢速命中

要保证绝大多数命中都是快速命中。



5. 缺点：

多种命中时间，流水设计复杂

6. 路预测

- 能够在降低冲突缺失的同时，保持了直接映射Cache命中速度
- 在Cache中另设特殊位，用来预测下一次Cache访问中可能会在组中用到的路或块
  - 提前使用多路选择器来选择即将被访问的块，而且在那个时钟周期内只需比较一个简单的标志位。如果缺失，则在接下来时钟周期中检查其他的块是否匹配。
  - Alpha 21264在它的2－路组相联指令Cache中使用了路预测。如果预测正确，则指令Cache的延迟只有一个时钟周期；否则，时间延迟为3个时钟周期。SPEC95仿真结果显示，路预测正确率超过了85。