由于程序的转移概率不会很低，数据分布的离散性较大，因此单纯依靠并行主存系统提高主存系统的效率是有限的。高速缓存 Cache 拥有比主存更快的速度，因此在 CPU 和主存之间设置 Cache 可以显著提高存储系统的效率。Cache 由 SRAM 组成，通常直接集成在 CPU 中。

程序访问的局部性原理

时间局部性	时间局部性是指最近的未来要用到的信息，很可能是现在正在使用的信息，因为程序中存在循环和需要多次重复执行的子程序段，以及对数组的存储和访问操作。
空间局部性	空间局部性是指最近的未来要用到的信息，很可能与现在正在使用的信息在存储空间上是邻近的，因为指令通常是顺序存放、顺序执行的，数据一般也是以向量、数组等形式簇聚地存储的。

高速缓冲技术就是利用局部性原理，把程序中正在使用的部分数据存放在一个高速的、容量较小的 Cache 中，使 CPU 的访存操作大多数针对 Cache 进行，从而提高程序的执行速度。

Cache 的基本工作原理

根据 Cache 的读、写流程，可知实现 Cache 时需解决:

- 数据查找：如何快速判断数据是否在 Cache 中。
- 地址映射：主存块如何存放在 Cache 中，如何将主存地址转换为 Cache 地址。
- 替换策略：Cache 满后，使用何种策略对 Cache 块进行替换或淘汰。
- 写入策略：如何既保证主存块和 Cache 块的数据一致性，又尽量提升效率。

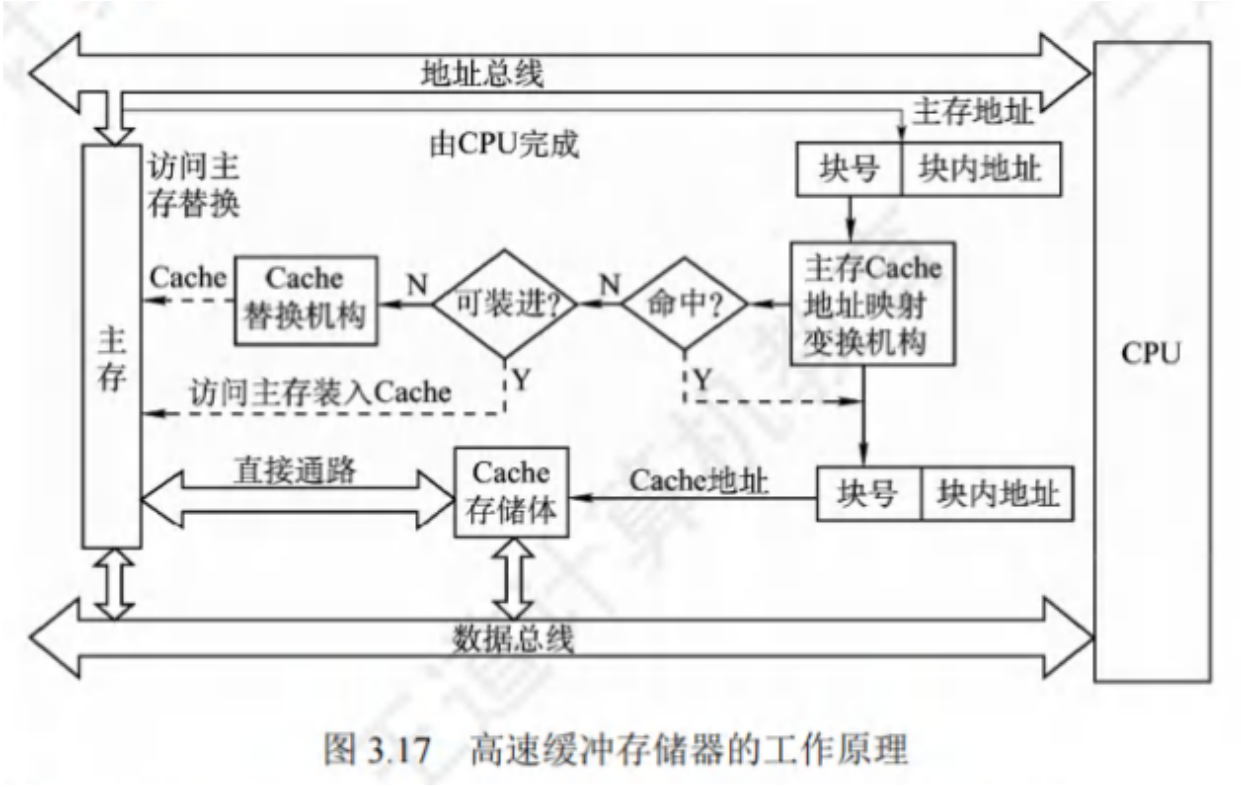


图 3.17　高速缓冲存储器的工作原理

为了便于 Cache 与主存交换信息，Cache 和主存都被划分为大小相等的块，Cache 块也称 Cache 行，每块由若干字节组成，块的长度称为块长 (也称行长)。因为 Cache 的容量远小于主存的容量，所以 Cache 中的块数远少于主存中的块数，Cache 中仅保存主存中最活跃的若干块的副本。因此，可按照某种策略预测 CPU 在未来一段时间内欲访存的数据，将其装入 Cache。

Cache 命中对 CPU 执行时间影响分析	当 CPU 发出读请求时，若访存地址在 Cache 中命中，就将此地址转换成 Cache 地址，直接对 Cache 进行读操作，与主存无关；若 Cache 不命中，则仍需访问主存，并把此字所在的块一次性地从主存调入 Cache。若此时 Cache 已满，则需根据某种替换算法，用这个块替换 Cache 中原来的某块信息。整个过程全部由硬件实现。值得注意的是，CPU 与 Cache 之间的数据交换以字位单位，而 Cache 与主存之间的数据交换则以 Cache 块为单位。 当 CPU 发出写请求时，若 Cache 命中，有可能会遇到 Cache 与主存中的内容不一致问题。例如，由于 CPU 写 Cache，把 Cache 某单元中的内容从 X 修改成 X’，而主存对应单元中的内容仍然是 X，没有改变，因此若 Cache 命中，需要按照一定的写策略处理。	某些计算机中也采用同时访问 Cache 和主存的方式，若 Cache 命中，则终止访存。
Cache 命中率的计算	CPU 欲访问的信息已在 Cache 中的比率称为 Cache 的命中率。设一个程序执行期间，Cache 的总命中次数为 Nc，访问主存的总次数为 Nm，则命中率 H 为 H=Nc/(Nc+Nm) 可见为提高访问效率，命中率 H 越接近 1 越好。设 tc 为命中时的 Cache 访问时间，tm 为未命中时的访问时间，1-H 表示未命中率，则 Cache-主存系统的平均访问时间 Ta 为 Ta=Htc+(1-H)tm	\

Cache缺失率对主存带宽的影响	\	\

Cache 和主存的映射方式

由于 Cache 行数比主存块数少得多，因此主存中只有一部分块的信息可放在 Cache 中，因此在 Cache 中要为每块加一个标记位，指明它是主存中哪一块的副本。该标记的内容相当于主存中块的编号。为了说明 Cache 行中的信息是否有效，每个 Cache 行需要一个有效位。

Cache 行中的信息是主存中某个块的副本，地址映射是指把主存地址空间映射到 Cache 地址空间，即把存放在主存中的信息按照某种规则装入 Cache。

直接映射

主存块中的每一块只能装入 Cache 中的唯一位置。若这个位置已有内容，则产生块冲突，原来的块将无条件地被替换出去 (无须使用替换算法)。直接映射实现简单，但不够灵活，即使 Cache 的其他许多地址空着也不能占用，这使得直接映射的块冲突概率最高、空间利用率最低。

$$\text{Cache行号} = \text{主存块号} \bmod \text{Cache总行数}$$

假设 Cache 共有 2^c 行，主存有 2^m 块，在直接映射方式中，主存的第 0 块、第 2^c 块、第 2^{c+1} 块等只能映射到 Cache 的第 0 行；而主存的第 1 块、第 2^c+1 块、第 2^{c+1}+1 块等只能映射到 Cache 的第 1 行，以此类推。由映射函数可看出，主存块号的低 c 位正好是它要装入的 Cache 行号。给每个 Cache 行设置一个常为 t=m-c 的标记 (tag)，当主存某块调入 Cache 后，就将其块号的高 t 位设置在对应 Cache 行的标记中。

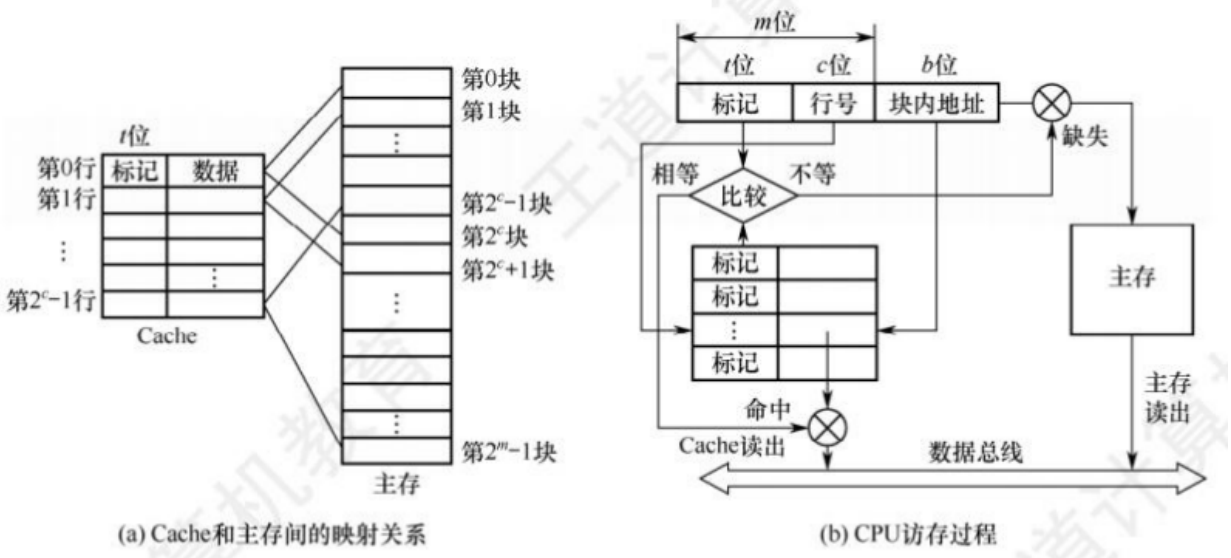
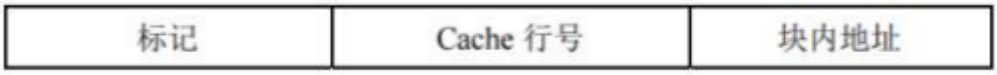


图 3.18　Cache 和主存之间的直接映射方式

直接映射的地址结构为



CPU 访存过程如图所示。首先根据访存地址中间的 c 位，找到对应的 Cache 行，将对应 Cache 行中的标记和主存地址的高 t 位标记进行比较，若相等且有效位为 1，则访问 Cache “命中”，此时根据主存地址中低位的块内地址，在对应的 Cache 行中存取信息；若不相等或有效位为 “0”，则 “不命中”，此时 CPU 从主存中读出该地址所在的一块信息送到对应的 Cache 行中，将有效位置 1，并将标记设置为地址中的高 t 位，同时将该地址中的内容送 CPU。

全相联映射

主存中的每一块可以装入 Cache 中的任何位置，每行的标记用于指出该行来自主存的哪一块，因此 CPU 访存时需要与所有 Cache 行的标记进行比较。

优点	缺点
Cache 块的冲突概率低，只要有空闲 Cache 行，就不会发生冲突 空间利用率高 命中率高	标记的比较速度较慢 实现成本较高，通常需采用按内容寻址的相联存储器

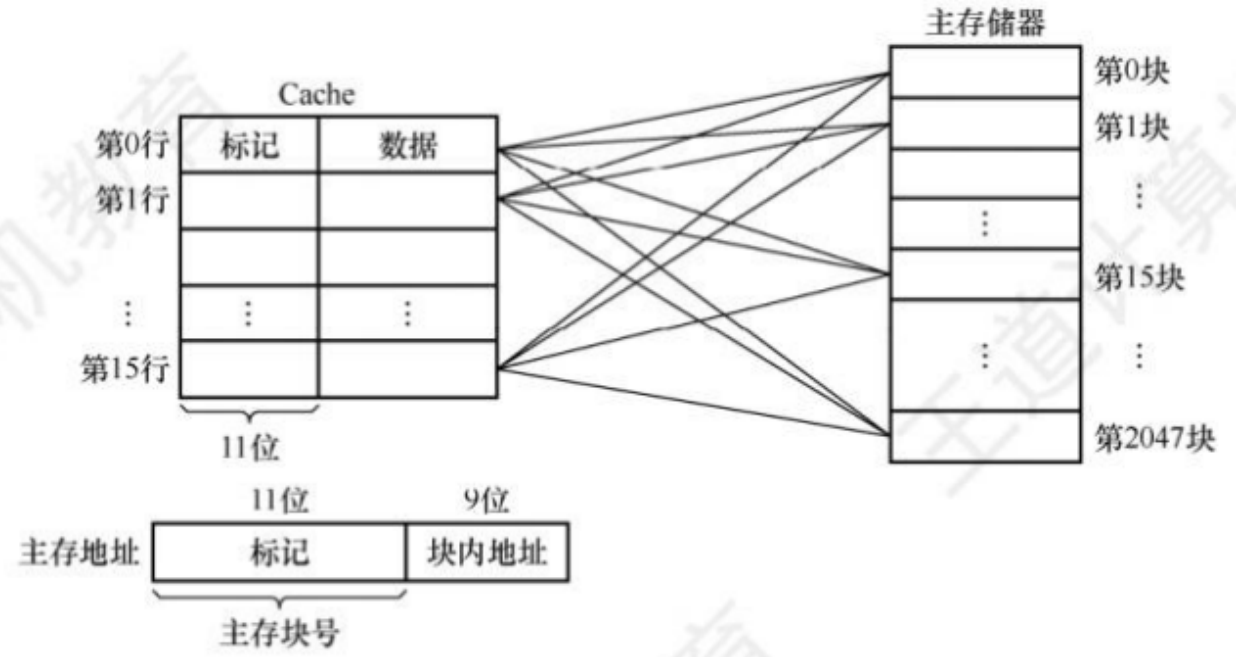


图 3.19　Cache 和主存之间的全相联映射方式

全相联映射的地址结构为

CPU 访存过程如下：首先将主存地址的高位标记 (位数 = log_2主存块数) 与 Cache 各行的标记进行比较，若有一个相等且对应有效位为 1，则命中，此时根据块内地址从该 Cache 行中取出信息；若都不相等，则不命中，此时 CPU 从主存中读出该地址所在的一块信息送到 Cache 的任意一个空闲行中，将有效位置 1，并设置标记，同时将该地址中的内容送 CPU。

通常为每一个 Cache 行都设置一个比较器，比较器位数等于标记字段的位数。访存时根据标记字段的内容来访问 Cache 行中的主存块，因而其查找过程是一种“按内容访问”的存取方式，所以是一种“相联存储器”。这种方式的时间开销和硬件开销都较大，不适合大容量 Cache。

组相联映射

将 Cache 分成 Q 个大小相等的组，每个主存块可以装入固定组中的任意一行，即组间采用直接映射、而组内采用全相联映射的方式。它是对直接映射和全相联映射的一种折中，当 Q=1 时变为全相联映射，当 Q=Cache 行数时变为直接映射。路数越大，即每组 Cache 行的数量越大，发生块冲突的概率越低，但相联比较电路也越复杂。选定适当的数量，可使组相联映射的成本接近直接映射，而性能上仍接近全相联映射。假设每组有 r 个 Cache 行，则称为 r 路组相联。

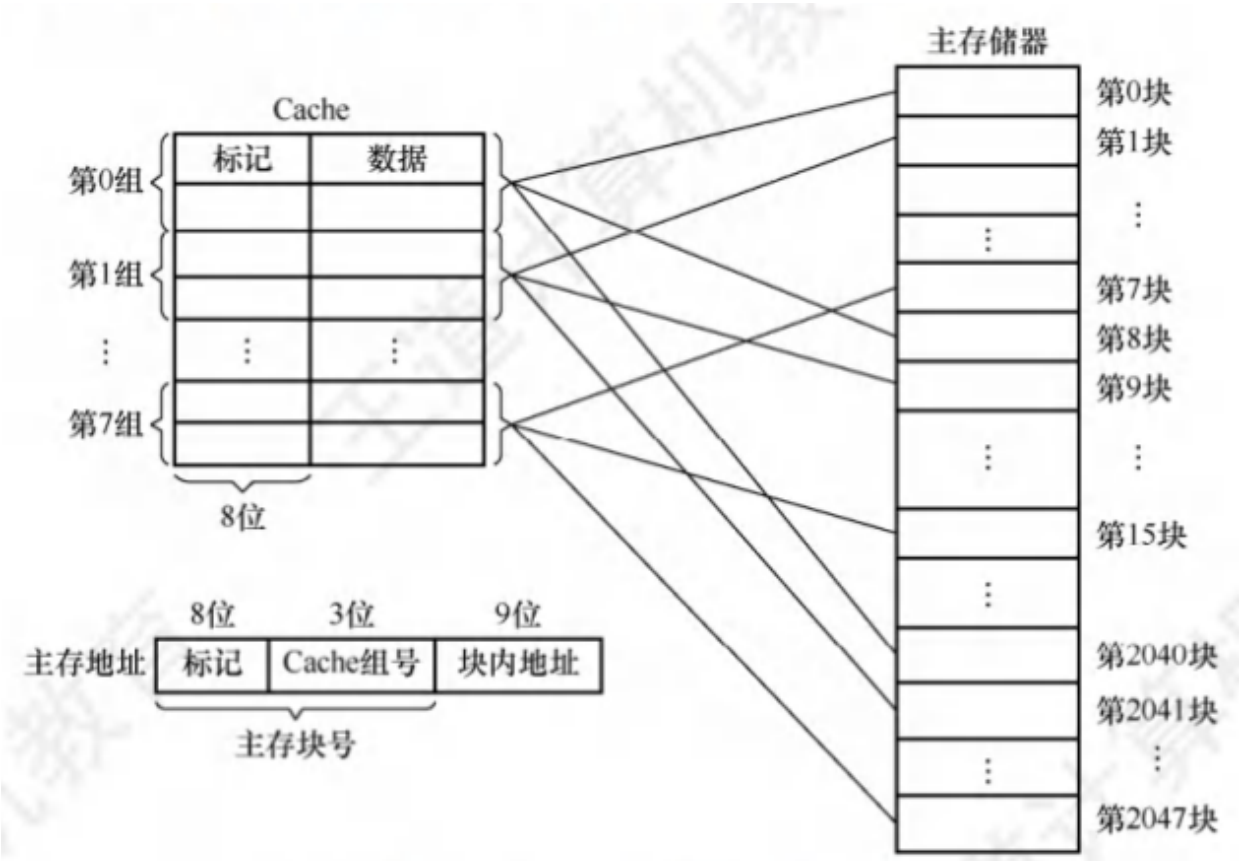
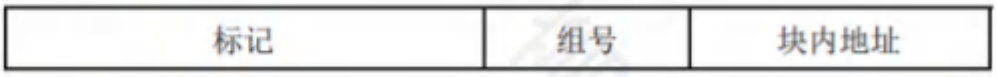


图 3.20　Cache 和主存之间的二路组相联映射方式

组相联映射的关系可以定义为

$$\text{Cache 组号} = \text{主存块号} \bmod \text{Cache 组数}(Q)$$

组相联映射的地址结构为



组相联映射的访存过程及Cache缺失处理过程	CPU 访存过程如下：首先根据访存地址中间的组号找到对应的 Cache 组；将对应 Cache 组中每个行的标记与主存地址的高位标记进行比较；若有一个相等且有效位为 1，则访问 Cache 命中，此时根据主存地址中的块内地址，在对应 Cache 行中存取信息；若都不相等或虽相等但有效位为 0，则不命中，此时 CPU 从主存中读出该地址所在的一块信息到对应 Cache 组的任意一个空闲行中，将有效位置 1，并设置标记，同时将该地址中的内容送 CPU。	
组相联映射中比较器的个数和位数	直接映射因为每块只能映射到唯一的 Cache 行，因此只需设置 1 个比较器。而 r 路组相联映射需要在对应分组中与 r 个 Cache 行进行比较，因此需要设置 r 个比较器。	

直接映射、组相联映射相关标记位及总容量的分析

假设某个计算机的主存地址空间大小为 256 MB，按字节编址，其数据 Cache 有 8 个 Cache 行，行长为 64 B。

1. 若不考虑用于 Cache 的一致维护性位 (脏位) 和替换算法控制位，并且采用直接映射方式，则该数据 Cache 的总容量为多少?
2. 若该 Cache 采用直接映射方式，则主存地址为 3200 (十进制) 的主存块对应的 Cache 行号是多少？采用二路组相联映射时又是多少?
3. 以直接映射方式为例，简述访存过程 (设访存的地址为 0123456 H)。

---

1. 因为 Cache 包括了可以对 Cache 中所包含的存储器地址进行跟踪的硬件，即

$$Cache\text{的总容量} = \text{存储容量} + \text{标记阵列容量}(\text{有效位}, \text{标记位})$$

   标记字段长度的计算：

$$\begin{aligned}&\text{主存地址有}28\text{位}(256\text{MB} = 2^{28}\text{B})\\&6\text{位为块内地址}(2^6\text{B} = 64\text{B})\\&3\text{位为行号}(2^3 = 8)\\&28 - 6 - 3 = 19\text{位为标记字段}\\&\text{Cache的总容量为}8 \times (1 + 19 + 512) = 4256\text{位}\end{aligned}$$

2. 直接映射方式中，主存按照块的大小划分，主存地址 3200 对应的字块号位 3200B/64B=50。而 Cache 只有 8 行，则 50 mod 8=2，因此对应的 Cache 行号为 2.

二路组相联映射方式，实质上就是将两个 Cache 行合并，内部采用全相联方式，外部采用直接映射方式，50 mod 4=2，对应的组号为2，即对应的 Cache 行号为 4 或 5。

3. 直接映射方式中，28 位主存地址可分为 19 位的主存标记位，3 位的块号，6 位的块内地址，即 0000 0001 0010 0011 010 位主存标记位，001 为块号，010110 为块内地址。

   首先根据块号，查 Cache (即 001 号 Cache 行) 中对应的主存标记位，看是否相同。若相同，再看 Cache 行中的有效位是否为 1，若是，称此访问命中，按块内地址 010110 读出 Cache 行所对应的单元并送入 CPU 中，完成访存。

   若出现标记位不相等或有效位为 0 的情况，则不命中，访问主存将数据取出并送往 CPU 和 Cache 对应块中，把主存地址的高 19 位写入 001 行的标记位，并将有效位置 1。

每个 Cache 行对应一个标记项 (包括有效位、脏位、替换算法位、标记位)，在组相联中，将每组各行的标记项排成一行，将各组从上到下排列，构成一个二维的标记阵列。查找 Cache 时就是查找标记阵列的标记项是否符合要求。

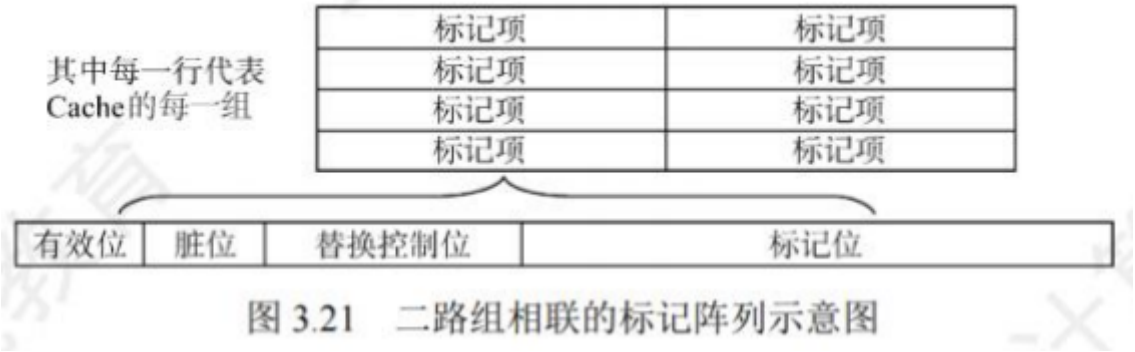


图 3.21　二路组相联的标记阵列示意图

因此本题中每行相关的存储器容量如图。

图 3.22　Cache 行的存储容量示意图

三种映射方式中，直接映射的每个主存块只能映射到 Cache 中的某一固定行；全相联映射可以映射到所有 Cache 行；N 路组相联映射可以映射到 N 行。当 Cache 大小、主存块大小一定时，

1. 直接映射的命中率最低，全相联映射的命中率最高。
2. 直接映射的判断开销最小、所需时间最短，全相联映射的判断开销最大、所需时间最长。
3. 直接映射标记所占的额外空间开销最少，全相联映射标记所占的额外空间开销最大。

Cache 中主存块的替换算法

在采用全相联映射或组相联映射方式时，从主存向 Cache 传送一个新块，当 Cache 或 Cache 组中的空间已被占满时，就需要使用替换算法置换 Cache 行。而采用直接相联映射时，一个给定的主存块只能放到唯一的固定 Cache 行中，所以在对应 Cache 行已有一个主存块的情况下，新的主存块毫无选择地把原先已有的那个主存块替换掉、因而无须考虑替换算法。

随机算法 RAND	随机地确定替换的 Cache 行。它的实现比较简单，但未依据程序访问的局部性原理，因此可能命中率较低。
先进先出算法 FIFO	选择最早调入的 Cache 行进行替换。它比较容易实现，但也未依据程序访问的局部性原理，因为最早进入的主存块也可能是目前经常要用的。
近期最少使用算法 LRU	依据程序访问的局部性原理，选择近期内长久未访问过的 Cache 行进行替换，其平均命中率要比 FIFO 的高，是堆栈类算法。
最不经常使用算法 LFU	将一段时间内被访问次数最少的 Cache 行换出。每行也设置一个计数器，新行装入后从 0 开始计数，每访问一次，被访问的行计数器加 1，需要替换时比较各特定行的计数值，将计数值最小的行换出。这种算法与 LRU 类似，但不完全相同。

LRU 替换位及其位数的计算

LRU 算法对每个 Cache 行设置一个计数器 (也称 LRU 替换位)，用计数值来记录主存块的使用情况，并根据计数值选择淘汰某个块，计数值的位数与 Cache 组大小有关，二路时有 1 位 LRU 位，四路时有 2 位 LRU 位。假定采用四路组相联，有 5 个主存块 {1, 2, 3, 4, 5} 映射到 Cache 的同一组，对于主存访问序列 {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}，采用 LRU 算法的替换过程如图所示。图中左边阴影的数字是对应 Cache 行的计数值，右边的数字是存放在该行中的主存块号。

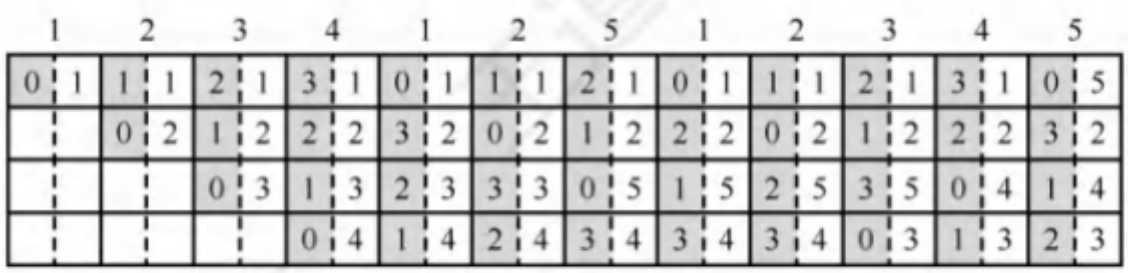


图 3.23　LRU 算法的替换过程示意图

计数器的变化规则：

1. 命中时，所命中的行的计数器清零，比其低的计数器加 1，其余不变；
2. 未命中且还有空闲行时，新装入的行的计数器置 0，其余全加 1；

3. 未命中且无空闲行时，计数值为 3 的行的信息块被替换，新装入的行的计数器置 0，其余全加 1.

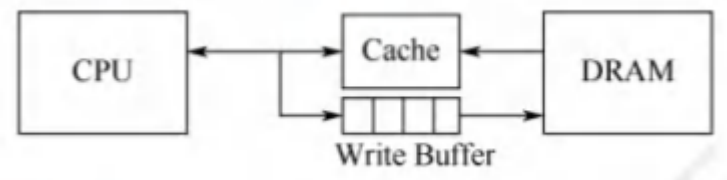
当集中访问的存储区超过 Cache 组的大小时，命中率可能变得很低，如上例的访问序列变为 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, ...，而 Cache 每组只有 4 行，则命中率为 0，这种现象称为抖动。

Cache 的一致性问题

因为 Cache 中的内容是主存块副本，当对 Cache 中的内容进行更新时，就需选用写操作策略使 Cache 内容和主存内容保持一致。

全写法和回写法都对应 Cache 写命中 (要被修改的块在 Cache 中) 时的情况。

Cache 写操作命中 (write hit)	
全写法 直写法 write-through	当 CPU 对 Cache 写命中时，必须把数据同时写入 Cache 和主存。当某一块需要替换时，就不必把这一块写回主存了，用新调入的块直接覆盖即可。这种方法实现简单，能随时保持主存数据的正确性。缺点是增加了访存次数，降低了 Cache 的效率。 写缓冲：为减少全写法直接写入主存的时间损耗，在 Cache 和主存之间加一个写缓冲 (Write Buffer)。CPU 同时写数据到 Cache 和写缓冲中，写缓冲再将内容写入主存。写缓冲是一个 FIFO 队列，写缓冲可以解决速度不匹配的问题。但若出现频繁写时，会使写缓冲饱和溢出。
回写法 write-back	当 CPU 对 Cache 写命中时，只把数据写入 Cache 而不立即写入主存，只有当此块被替换出时才写回主存。这种方法减少了访存次数，但存在数据不一致的隐患。为了减少写回主存的次数，给每个 Cache 行设置一个修改位 (脏位)。若修改位为 1，则说明对应 Cache 行中的块被修改过，替换时须写回主存；若修改位为 0，则说明对应 Cache 行中的块未被修改过，替换时无须写回主存。



Cache 写操作不命中	
写分配法 write-allocate	更新主存单元，然后把这个主存块调入 Cache。它试图利用程序的空间局部性，缺点是每次写不命中都要从主存读一个块到 Cache 中。
非写分配法 not-write-allocate	只更新主存单元，而不把主存块调入 Cache。

非写分配法通常与全写法合用，写分配法通常和回写法合用。