第3章　存储系统

3.1　存储系统基本原理

3.1.1　存储系统的定义

两个或两个以上速度、容量和价格各不相同的存储器用硬件、软件或软件与硬件相结合的方法连接起来称为存储系统。存储系统与存储器的概念是不同的，即使一台机器中只有一种存储器或者多种存储器，若不构成存储系统也无法充分发挥这些存储器的性能。

存储系统是随计算机技术的不断发展而逐渐形成的。由于冯·诺依曼结构的缺陷，计算机系统结构由以 CPU 为中心转变为以主存储器为中心，如图 3-1 所示。

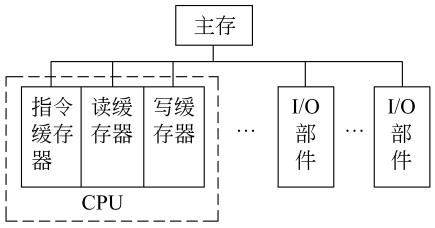


图 3-1　以主存储器为中心的计算机

计算机系统对存储器的希望是速度高到与 CPU 速度匹配，容量大到能存放尽可能多的程序数据、系统软件，价格低廉用户可以接受。而高速度、大容量、低价格是难以在一个存储器中同时实现的。为了解决上述矛盾，一方面致力于对存储器的介质、器件组成技术的研究，以提供性能良好的新型存储器；另一方面则对各种不同性能的存储器之间寻求一种相互联系、缺陷互补的方法，使其组成存储器系统以满足高性能的要求，因此出现了存储系统。图 3-2 中由三个存储器 M_1、M_2、M_3 组成存储系统，使该系统具有下述性能，从应用程序员的角度看来它就是一个存储器，它的速度应接近于其中最快的 M_1，而存储容量接近于容量最大的 M_3，价格接近最便宜的 C_3。也就是说，对用户而言，它用的是一个速度为 T_1、容量为 S_3、价格为 C_3 的存储器。

假设每级存储器的性能指标为速度、容量、价格，分别用下列参数表示。

速度 T：存储器访问周期（或称存取周期、读写周期等）。

容量 S：以字节数表示，单位为 B、KB、MB、GB、TB 等。

价格 C：表示单位容量的平均价值，单位为 $C/b（每一位二进制数合多少美分）或 $C/KB。

每级存储器的性能参数可以表示为 T_i，S_i，C_i。存储系统的性能可表示为：$T_i < T_{i+1}$，

$S_i < S_{i+1}$，$C_i > C_{i+1}$。存储系统示意图如图 3-2 所示。

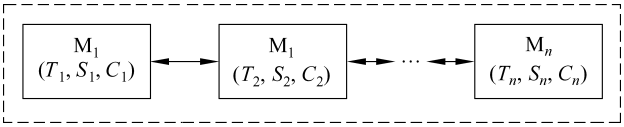


图 3-2　存储系统示意图

图 3-2 中理想的存储系统可以用如下形式表示。

$T \approx \min(T_1, T_2, \cdots, T_n)$，用存储周期表示。

$S \approx \max(S_1, S_2, \cdots, S_n)$，用 MB 或 GB 表示。

$C \approx \min(C_1, C_2, \cdots, C_n)$，用每位的价格表示。

衡量存储系统性能的这三个参数是组成存储系统的每个存储器都具备的必要因素，为了方便分析，假设采用 M_1 和 M_2 两个存储器构成一个两级存储层次结构。两个存储器的容量、价格和速度分别为 S_1、C_1、T_1 和 S_2、C_3、T_2，存储系统的容量、价格和速度分别为 S、C 和 T。

1. 容量

计算机中对存储系统进行编址的要求是对计算机的使用者提供尽可能大的地址空间，而且能对这个地址空间进行随机访问。一般可以选择 M_2 存储器进行编址，对于 M_1 存储器可以不编址，或者只在系统内部进行编址。

2. 价格

整个存储系统的单位容量平均价格 C 可以用式(3-1)计算。

$$C = \frac{C_1 \cdot S_1 + C_2 \cdot S_2}{S_1 + S_2} \tag{3-1}$$

由式(3-1)可以看出，当 $S_1 \ll S_2$ 时，则有 $C \approx C_2$，也就是说，整个存储系统的单位容量平均价格 C 接近于比较便宜的 M_2 存储器。

3. 速度

存储系统的速度一般用访问周期来衡量，而访问周期与命中率有着密切的联系。命中率是 CPU 访问存储系统时在 M_1 中找到所需信息的概率。假设通过统计得到访问 M_1 和 M_2 的次数分别为 N_1 和 N_2，则有式(3-2)：

$$H = \frac{N_1}{N_1 + N_2} \tag{3-2}$$

对应的不命中率为 $1\text{-}H$。

根据命中率可以得出整个存储系统的访问周期 T 的表达式如式(3-3)所示。

$$T = H \cdot T_1 + (1 - H) \cdot T_2 \tag{3-3}$$

其中，T_1 是访问 M_1 存储器的时间，T_2 是访问 M_2 存储器的时间。

从式(3-3)中可以看出，当命中率 $H \to 1$ 时，$T \to T_1$，即存储系统的访问周期 T 接近

于速度比较快的 M_1 存储器的访问周期 T_1。

存储系统的访问效率定义如式(3-4)所示。

$$\begin{aligned} e &= \frac{T_1}{T} \\ &= \frac{T_1}{H \cdot T_1 + (1-H) \cdot T_2} \\ &= \frac{1}{H + (1-H) \cdot \frac{T_2}{T_1}} \end{aligned} \tag{3-4}$$

访问效率越高，说明存储系统的访问效率与相对比较快的那个存储器的速度越接近。从式(3-4)中还可以看出，存储系统的访问效率主要与命中率和构成存储系统的两级存储器的速度之比有关。

如果要提高存储系统的速度与相对比较快的那个存储器的速度接近，有两条途径：一条途径是提高命中率 H；另一条途径是构成存储系统的两个存储器的速度不要相差太大。

3.1.2 存储系统的层次结构

计算机中的存储器根据工作速度、存储容量、访问方式、用途等构成一个层次结构，具体如图 3-3 所示。

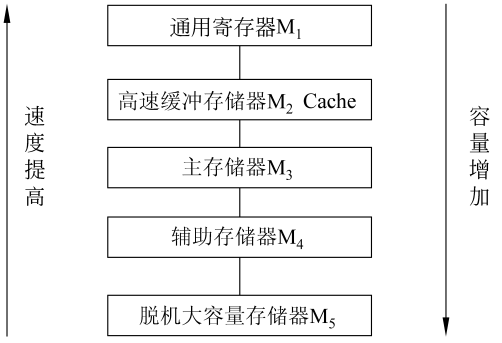


图 3-3 存储系统的层次结构

图中第一层 M_1 是通用寄存器组，位于 CPU 内部，是汇编程序员可编程的空间，也是高级语言中编译器可支配的空间。寄存器之间数据的传递只需要一个时钟周期，速度最快，可以与 CPU 直接匹配。

第二层 M_2 为高速缓冲存储器 Cache，置于主存储器与 CPU 之间，存放那些运行程序中近期即要用到的指令与数据，其速度比主存快很多，由存储器管理部件控制，Cache 对汇编级程序员是透明的。Cache 本身也可以根据需要分为 2～3 个层次，速度最高的部分也可以集成在处理机的芯片中。

第三层 M_3 为主存储器，是计算机的主要部件，是存储系统的核心。CPU 的指令可

以直接访问用户能够用到的编程空间。存储系统中的诸项技术措施，都是为了改善主存储器的性能。

第四层 M_4 为辅助存储器，由硬盘组成，具有容量大、价格低的特点。

第五层 M_5 为脱机存储器，指磁带机、光盘等。

如图 3-3 所示的层次结构的存储系统中，主要采取了两种技术。一是在主存储器与 CPU 之间增加了一级高速缓冲存储器，使得 CPU 在对主存储器访问时，所面对的存储器具有高速缓存的速度和主存储器的容量，以改善速度性能。二是在主存储器与辅助存储器之间采用了虚拟存储技术，使得 CPU 在访问主存储器时，所见到的存储器是具有近于主存储器的速度和辅助存储器的巨大存储空间，如果不考虑 M_1 及 M_5 两级，那么与 CPU 匹配的存储器系统将是一个近于 Cache 速度和辅助存储器容量的存储器。

如表 3-1 所示为目前各级存储器的主要性能指标及其所采用的材料工艺。

表 3-1 各级存储器的性能指标

存储器层次	通用寄存器	缓冲栈	Cache	主存储器	磁盘存储器	脱机存储器
存储周期	＜10ns	＜10ns	10～60ns	60～300ns	10～30ms	2～20min
存储容量	＜512B	＜512B	8KB～2MB	MB～1GB	GB～1TB	GB～10TB
价格/$C·KB^{-1}	1200	80	3.2	0.36	0.01	0.0001
访问方式	直接译码	先进先出	相联访问	随机访问	块访问	文件组
材料工艺	ECL	ECL	SRAM	DRAM	磁表面	磁、光等
分配管理	编译器分配	硬件调度	硬件系统	操作系统	系统/用户	系统/用户
带宽/MS·s^{-1}	400～8000	400～1200	200～800	80～160	10～100	0.2～0.6

3.1.3 多体交叉访问存储器

根据主存储器中存储体的个数以及 CPU 访问主存一次所能读出的信息的位数，可以将主存储器系统分为以下 4 种类型。

(1) 单体单字存储器。即存储器只有一个存储体，而且存储体的宽度为一个字，一次可以访问一个存储器字，此存储器字长 W 与 CPU 所要访问的字(数据字或指令字，简称 CPU 字)的字长 W 相同。

(2) 单体多字存储器。即存储器只有一个存储体，但存储体的总线宽度较大，可以是多个字。若要想提高主存储器频宽，使之与 CPU 速度匹配，在同样的器件条件下，需要提高存储器的字长。例如，改用图 3-4 所示的方式组成，主存储器在一个存储周期内就可以读出 4 个 CPU 字，相当于 CPU 从主存储器中获得信息的最大速率提高到原来的 4 倍，这种主存储器称为单体多字存储器。单体多字($m=4$)存储器示意图如图 3-4 所示。

(3) 多体单字交叉存取的存储器。例如，多体交叉存储器，因为每个存储体都是一个 CPU 字的宽度。

(4) 多体多字交叉存储器。它将多体并行存取与单体多字相结合。

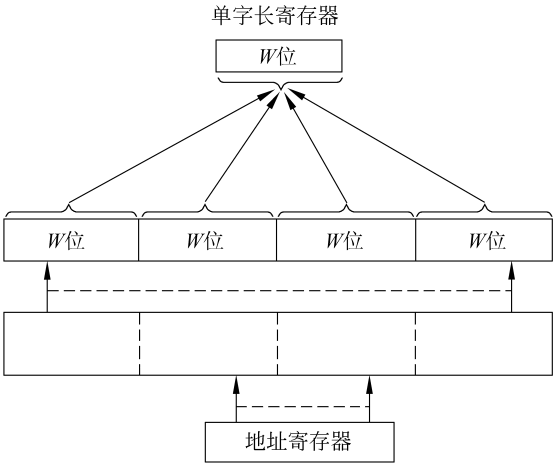


图 3-4　单体多字($m=4$)存储器

单体多字方式要求可并行读出的 m 个字必须是地址顺序排列且处于同一主存单元。多体单字方式采用 m 个存储体交叉编址，多个存储体并行进行存取操作，每个存储体的宽度一般是一个字的宽度。这种方式所花费的器件和总价格并不比采用单体多字方式的多太多，但其实际带宽却可以比较高。因为多体单字方式只要 m 个地址不发生分体冲突（即没有发生两个以上地址同属一个分体），即使地址之间不是顺序的，仍可并行读出，使实际带宽提高成单体单字的 m 倍。

能并行读出多个 CPU 字的单体多字、多体单字交叉、多体多字交叉存取的主存系统称为并行主存系统。

并行访问存储器的冲突主要来自如下几方面。

(1) 取指令冲突。在遇到转移指令而且转移成功时，同一个存储周期中读出的 n 条指令中，在转移指令后面的几条指令将无用。

(2) 读操作数冲突。一次同时读出的 n 个操作数，不一定都有用。换一种说法，需要的多个操作数不一定正好都存放在同一个存储字中。

(3) 写数据冲突。这种并行访问的存储器，必须凑齐了 n 个数之后才能一起写入存储器。如果只写一个字，必须先把属于同一个存储字的 n 个数据都读到数据寄存器中，然后在地址码的控制下修改其中的一个字，最后再把整个存储字写回存储器中。

(4) 读写冲突。当要读出的一个字和要写入存储器的一个字处在同一个存储字内时，无法在一个存储周期内完成。

这 4 种冲突中，第 1 种冲突的概率比较小，因为程序在大多数情况下是顺序执行的。第 2 种冲突的概率比较大，因为操作数的随机性比程序要大。第 3 种和第 4 种冲突，解决起来有一定困难，需要专门进行控制。

分析发生这些冲突的原因，从存储器本身看，主要是因为地址寄存器和控制逻辑只有一套。如果设置 n 个独立的地址寄存器和 n 套读写控制逻辑，则第 3 种和第 4 种冲突就可以解决了，第 1 种和第 2 种冲突也能有所缓解。

目前的存储器面对着多个访问源，所要访问的内容可能存放在任意一个存储体内的任意单元，这时就可以采用交叉访问。如图 3-5 所示是交叉访问的示意图，它具有多个访问源 S，可以同时向多个存储体 m 中的任何一个发出访问请求（$s=3$，$m=4$）。

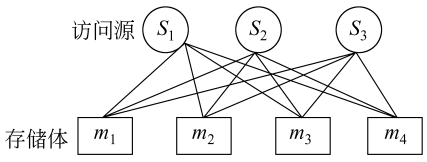


图 3-5　多体交叉存取示意图

并行交叉访问存储器的结构，依据编址方式的不同，可分为高位交叉访问与低位交叉访问两种，其中，低位交叉存储器能够有效地解决访问冲突问题。

1. 高位交叉访问存储器

计算机系统中的主存储器如果采用模块化结构一般都采用高位交叉方式。例如，用两个 512MB 的模块构成 1GB 的主存储器，或用 4 个 256MB 的模块构成 1GB 的主存储器等，这种方法可以很方便地改变主存储器的容量。高位交叉访问存储器的结构如图 3-6 所示。

图 3-6 中的存储器有 m 个存储体（$0\sim m-1$），每个存储体内的容量为 n 个字，存储单元的地址由两部分组成：地址码的低位部分为各存储体内地址 $0\sim n-1$；高位字段表示了各存储体的体号 $0\sim m-1$。寻址时，地址的高位字段指出寻址的存储体，低位字段送到该存储体的地址缓存器。

地址码长度 l 应为 $\log_2(m\times n)$，其中，地址的高位字段 $a=\log_2 m$，低位字段 $b=\log_2 n$。

在高位交叉访问方式中，每个存储模块都有各自独立的控制部件，包括地址寄存器、地址译码器、驱动电路、放大电路、读写控制电路、数据寄存器等，有些存储器模块中还有校验及校正电路、高速缓冲存储部件等。

高位交叉方式可以很方便地扩大主存容量。对单用户而言，虽然可以使用到一个很大容量的主存，但由于程序的连续性和局部性，在程序执行中被访问的指令序列和数据绝大多数都分布在同一个存储模块中，因此通常只有一个存储模块在不停地忙碌，其他存储模块是空闲的。如果在多任务或多用户的应用状态下，可以将不同的任务分别存放在不同的体内，减少访问冲突，发挥并行访问的优点。

2. 低位交叉访问存储器

为了达到提高主存储器速度的目的，可以采用低位交叉方式的存储器。在一个存储器周期内 n 个存储体必须分时启动。低位交叉访问存储器的结构中地址码的低位字段为存储体号，高位字段为体内的地址，低位字段译码后决定选择哪个体的数据。这种方式在一般情况下，并行性比较好，可以很有效地拓宽存储器的频带。低位交叉访问存储器的结构如图 3-7 所示。

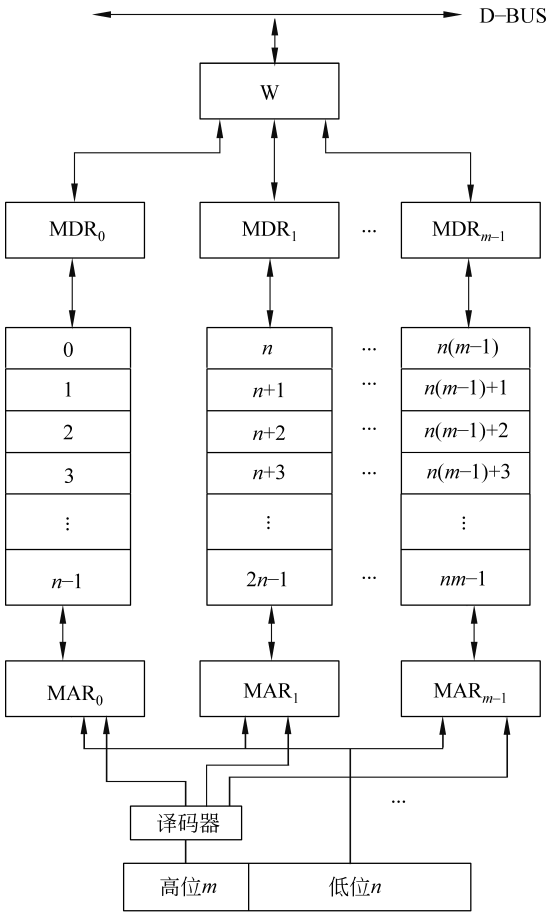


图 3-6　高位交叉访问存储器结构

由于从总线上传送来的地址码不可能是 m 个地址同时到，所以对于多个存储体采取分时启动的方式，如果存储器的访问周期为 T，那么可以分为 m 个时间段，每隔 $t=T/m$ 的时间，启动一个存储体。下面以 $n=8$、$m=4$ 为例，说明低位交叉编址结构分时启动的工作情况。由 8 个存储体构成，总存储容量为 64 的主存储器的低位交叉编址方式如图 3-8 所示。

如果请求访问的地址分别在 4 个存储体中，如 0，17，18，15 或 8，5，14，23，则可以 4 体并行工作，在一个存储周期中取得 4 个数据，速度提高了 4 倍。如果所访问的数据地址有两个或两个以上集中在一个存储体中，就会出现访问冲突。如访问地址为 1，9，10，26，则 1，9 在 1 体中冲突，10 与 26 在 2 体冲突，只好先取 1，10，第二次访问再取 9，26。这时就只有两个存储体并行工作，频带宽度降低了一半。如果 4 个地址都在一个存储体中冲突，则与单体工作一样，速度没有改善，因此防止访问冲突在并行存储器中十分重要，而且必须采取措施进行解决。

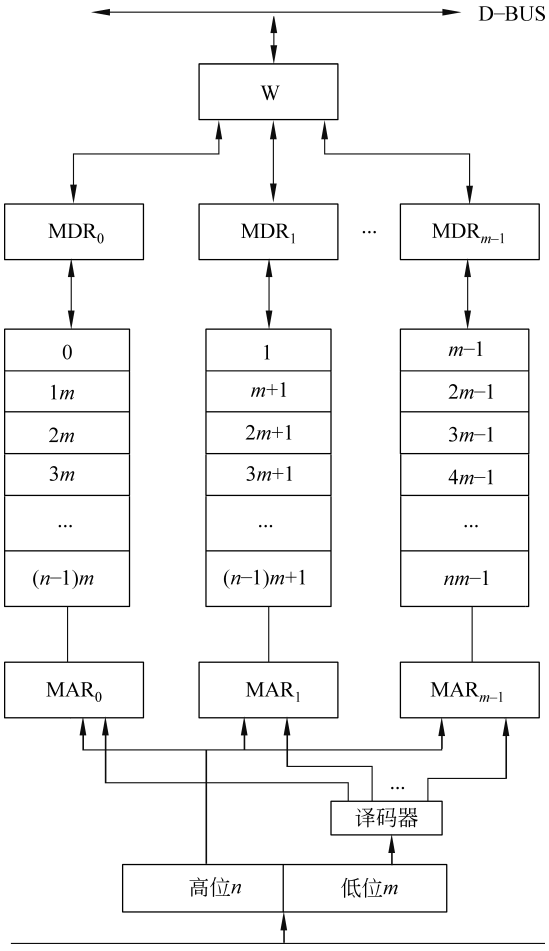


图 3-7　低位交叉访问存储器结构

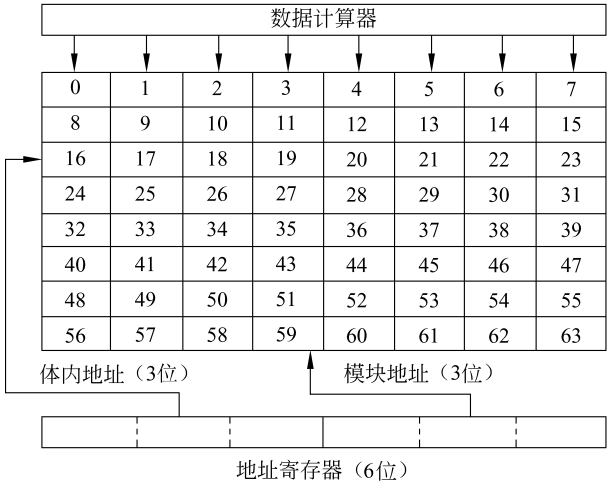


图 3-8　由 8 个存储体构成的主存储器的低位交叉编址方式

将一个主存周期分为4个时段，对于各个访问请求源而言，同一时刻只能访问一个存储体，每隔1/4主存周期，启动一个存储体。在结构上，4个存储体可以共用一套寻址译码驱动电路及控制部件。如图3-9所示给出了低位交叉编址主存储器的分时启动图。

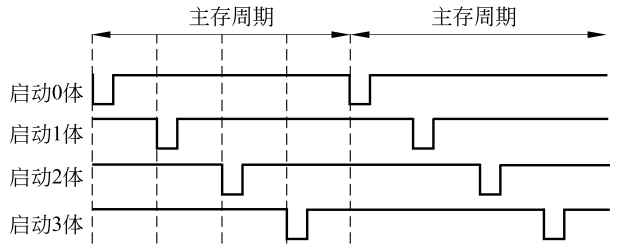


图3-9　低位交叉编址主存储器分时启动图

上述以低位交叉访问方式工作的存储器实际上是采用流水线方式进行存取的，由于能够明显地提高主存的频带宽度，改善速度性能，因而被广泛采用。特别是在多处理机、流水线处理机中，一般都采用这种低位交叉访问的多体存储器。在理想的情况下，整个存储器速度提高的倍数与分体数相同，即 m 倍。实际上这是不可能的，一是取指令时，遇到转移指令，使后取指令作废；二是取操作数时，存在数据的离散性，可能产生分体冲突。

3.2　高速缓冲存储器

根据存储系统的层次结构可知，为了弥补主存速度的不足，在CPU和主存之间设置了一个高速、小容量的缓冲存储器Cache。这样，从CPU的角度来看，速度接近于Cache，容量接近于主存。目前所有主流处理器都具有一级缓存和二级缓存，高端处理器还集成了三级缓存。其中，一级缓存可分为一级指令缓存和一级数据缓存。一级指令缓存用于暂时存储并向CPU递送各类运算指令；一级数据缓存用于暂时存储并向CPU递送运算所需数据。二级缓存相当于一级缓存的缓冲器：一级缓存制造成本很高，因此它的容量有限，二级缓存的作用就是存储那些CPU处理时需要用到的而一级缓存又无法存储的数据。同理，三级缓存和内存可以看作二级缓存的缓冲器，它们的容量递增，单位制造成本却递减。

3.2.1　高速缓冲存储器的基本结构与工作原理

Cache与主存储器之间通常以块为单位进行数据交换，块的大小以主存储器的一个存储周期中能访问到的数据长度为限。Cache的基本结构如图3-10所示。

因此，在Cache存储系统中，把Cache和主存储器都划分成相同大小的块，这样，主存地址由块号 B 和块内地址 W 两部分组成，Cache的地址也由块号 b 和块内地址 W 组成。

Cache主要分成以下几部分。

(1) Cache存储体：主要存放由主存调入的指令和数据。主存与缓存之间的信息传送是以块为基本单位，其划分的块的大小也与Cache相同，因此主存与缓存的块内地址字段相同。主存与缓存的地址格式如图3-11所示。

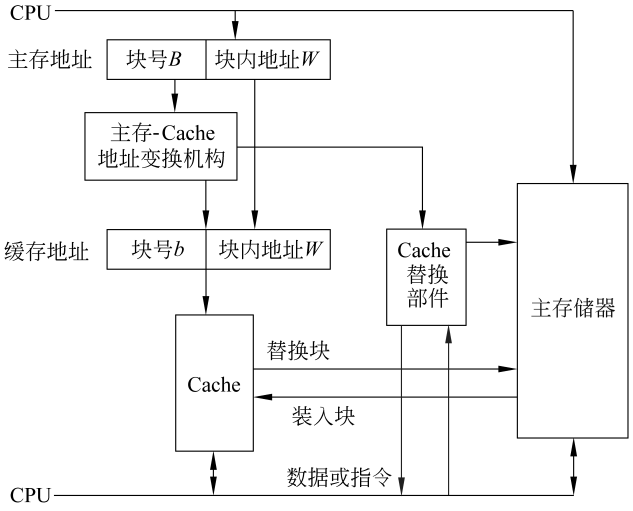


图 3-10　Cache 的基本结构

图 3-11　主存与缓存的地址格式

(2) 主存-Cache 地址变换机构：完成主存地址到缓存地址的转换。每当给出一个主存字地址进行访存时，都必须经过主存-Cache 地址变换机构判断该访问字所在的块是否已在 Cache 中。如果 Cache 命中，主存地址经地址变换机构变换成 Cache 地址去访问 Cache，Cache 与处理机之间进行信息传送；如果不在 Cache 中，即不命中时，产生 Cache 块失效，这时就需要从主存中将包含该字的块信息调入 Cache 中，同时将该字送给 CPU。如果 Cache 已满，则需使用替换部件。

地址转换的原理就是建立一个登记表(或称目录表)，主存信息调入时进行登记，在此登记表中记录下主存某一块存入缓存的块地址；当进行信息访问时，再到该记录表中查找，得到缓存地址。该部件的核心是一个相关存储器，也就是按内容存取的存储器。相关存储器的每一个单元(或称表项)必定有两个字段，即主存地址与缓存地址，以主存地址为索引，查找缓存地址。

(3) 替换部件：缓存已满时进行数据块的替换。首先按照一定的替换策略决定被替换掉的块，然后将主存中新调入的块调入主存，并修改地址转换部件。

下面介绍一下 Cache 的工作过程。

计算机刚开始工作时，Cache 为空。CPU 需要访问主存时，首先到 Cache 中查找，即在相关存储器中按内容访问，显然查找不到，即为不命中，则转向替换部件。由替换部件可知，此时 Cache 内尚有空间没有利用，则通知主存将所需的数据调入 Cache，并在目录表中进行登记。当计算机已运行过一段时间，Cache 内已经存入很多数据块，这时就会出

现以下两种情况。

（1）命中：在目录表中查到与主存块地址相同的表项，由此表项可知该块存入缓存的块号。该块号与主存地址中的块内地址组装即形成缓存地址，按照缓存地址到 Cache 存储体中进行访问即可获得 CPU 所需数据。

（2）不命中：目录表中没有相同的表项，则说明此块尚没有调入缓存，此时转向替换部件，准备调入新内容。这时又有两种情况：一是缓存有空间，则可直接通知主存调入；一是缓存已满，替换部件则按一定的策略选择出应该替换掉的块。如果该块尚有保存价值，则可以写回主存；如果无保存价值，主存可以调入新块将其覆盖。为了节省时间，在访问缓存的同时访问主存，如果不命中，那么主存将数据送入缓存的同时，可以送往 CPU。

根据上述过程，为了提高机器的运行速度，尽可能把主存中当前最活跃的那些指令与数据存放到 Cache 中。当 CPU 需要时，可以直接到 Cache 中去访问，即提高机器的命中率。后面章节将会详细介绍如何提高命中率。另外，为了更好地发挥 Cache 的高速性，减少 CPU 与 Cache 之间的传输延迟，应让 Cache 在物理位置上尽量靠近处理机或放在处理机中。对共用主存的多处理机系统，如果每个处理机都有它自己的 Cache，让处理机主要与 Cache 交换，就能大大减少使用主存的冲突，提高整个系统的吞吐量。

3.2.2 地址映像与转换

地址映像是指某一数据在主存中的地址与缓存中的对应关系，即主存中的数据是按一种确定的规则存放到缓存中，根据这种规则可以很容易地通过地址变换将主存地址转换成缓存地址。地址的映像和变换是紧密相关的，采用什么样的地址映像方法就必然有与这种映像方法相对应的地址变换方法。在进行地址映像和变换过程中，都以块为单位进行调度。下面介绍 3 种映像规则。

1. 全相联方式

地址映像与转换是在两个不同的时间进行的，在数据由主存调入缓存时，按照地址映像规则建立地址映像表，即目录表在 CPU 访问该数据时，根据地址目录表的记录，进行地址转换。

全相联的地址映像规则如下。

（1）主存与缓存分成相同大小的数据块。

（2）主存的任意一块可以装入 Cache 的任意一块中。

设 Cache 的容量为 C，每块的大小为 B，主存的容量为 M，每块的大小也为 B，具体映像规则如图 3-12 所示。

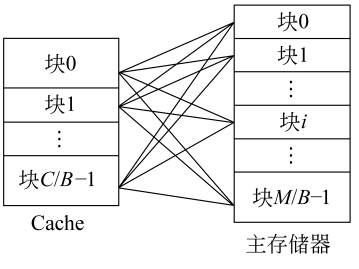


图 3-12　全相联映像方式

图 3-12 中主存分成 M/B 块，缓存分成 C/B 块，当调入某块时，可在目录表中进行登记。目录表由三部分组成：主存块号、Cache 块号和一个有效位，其行

数应与 Cache 的块数相同，字长为主存地址中的块号长度加上 Cache 地址中的块号长度再加上一个有效位。当该位为 1 时，此单元内容有效；为 0 时，则无效。当刚开机运行时，目录表中的这一位都为 0，随着主存内容的调入，将该位置“1”；当此块数据调出或作废时，该位清除为 0。此目录表是按内容进行访问的相联存储器。其地址变换过程如图 3-13 所示。

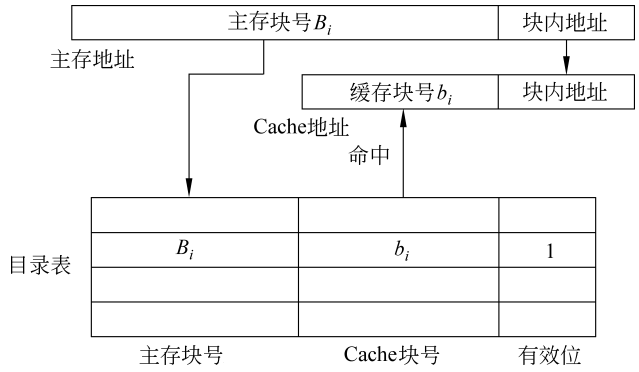
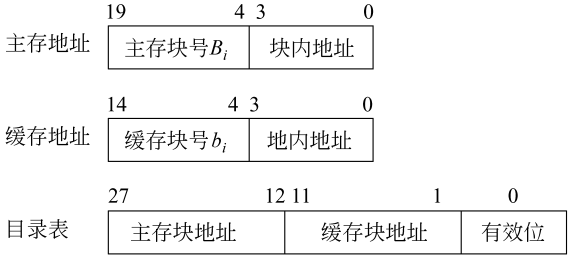


图 3-13　全相联地址转换

举例：某机主存容量为 1MB，Cache 的容量为 32KB，每块的大小为 16B，按全相联方式写出主、缓存的地址格式和目录表格式。



共 28 位＝16 位主存块地址＋11 位缓存块地址＋1 位有效位

容量：应与缓存块数量相同，即 $2^{11}=2048$。

全相联方式的优点是命中率比较高，块冲突率比较小，Cache 存储空间利用率高。缺点是相关存储器访问时，每次都要与全部内容比较。如上例中目录表的容量为 2048，如此大容量的相关存储器难以实现，而且速度低，代价也很高，因此在缓存结构中应用得比较少。

2. 直接相联方式

直接相联方式是一种最简单的映像方式。主存中的一块只能映像到 Cache 的一个确定块中。

直接相联的地址映像规则如下。

(1) 主存与缓存分成同样大小的块。

(2) 主存容量应是缓存容量的整数倍，将主存空间按缓存的容量分成区，主存中每一

区的块数与缓存的总块数相等。

（3）主存中某区的一块存入缓存时只能存入缓存中块号相同的位置。

根据上述规则，如果主存的块号为 N，Cache 的块号为 b，则二者之间的映像关系可以表示为：$b=N \bmod (C/B)$，其中，C/B 是 Cache 的块数。

直接相联的映像关系如图 3-14 表示。

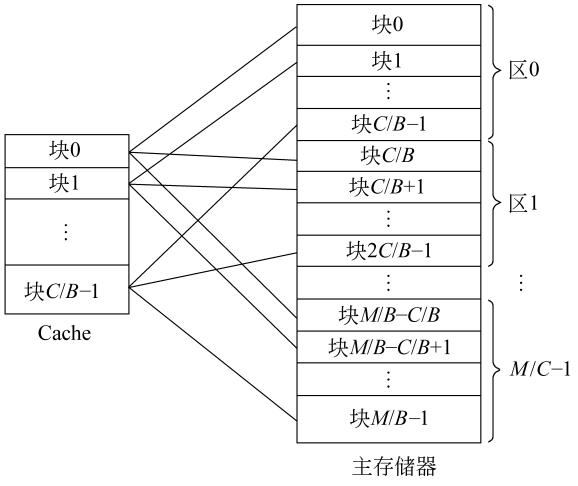


图 3-14　直接相联地址映像

首先将主存按照 Cache 的大小等分成区，因此这里主存的容量应是 Cache 容量的整数倍。每个区和 Cache 仍然分成同样大小的块，这样每个区的块数和 Cache 的块数相等。在这种情况下，直接相联方式就是把主存中各个区中相对块号相同的那些块映像到 Cache 中同一块号的位置上。

由于主存中任意一个区的数据块都可能映像到 Cache 中，因此需要有一个存放主存区号的小容量存储器，即目录表。此目录表按地址进行访问，具体格式为：区号，有效位。其行数与 Cache 的块数相同，字长为区号长度加上一个有效位。其地址变换如图 3-15 所示。

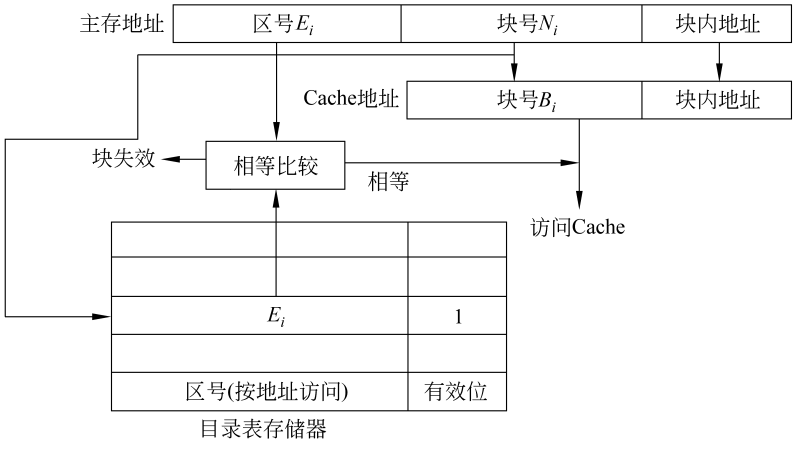


图 3-15　直接相联地址变换

图 3-15 中，首先由处理机给出主存地址，根据主存地址中的块号 B 按地址访问到对应的行，取出该行中的区号 E_i 与主存地址中的区号进行比较。如果比较结果相等，且有效位为“1”，此时说明要访问的那一块已经放入 Cache 中了，即为命中，按照块号和块内地址组合而成的 Cache 地址即可访问到所需数据。若比较结果不等，则为块失效状态，Cache 没有命中，说明被访问的块没有装入到 Cache 中。在这种情况下，对 Cache 的访问中止，继续访问主存储器，同时由硬件按照一定的规则将该块调入 Cache。

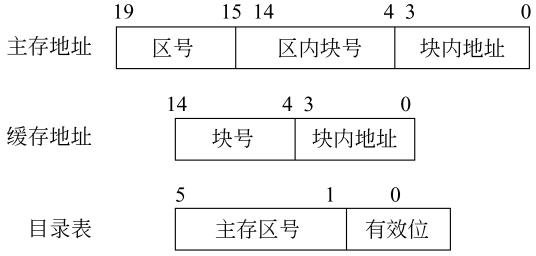
另外，还有以下两种情况需要考虑。

一是区号比较结果相等，但有效位为“0”，此时说明 Cache 中的该块已经作废。将按照该地址从主存中读出的新块装入到 Cache 的对应位置中即可，并将有效位置“1”，以备下一次访问使用。

二是区号比较结果不相等，有效位为“1”，表示原来在 Cache 中存放的那一块是有用的，此时必须把 Cache 中的这一块写回到主存储器中原来存放它的存储单元中，才能保证将主存中读出的新块装到 Cache 中。

上述两种情况在调入新块的同时都必须把对应的主存区号写到区号存储器的相应单元中。

举例：主存为 1MB，缓存为 32KB，块的大小为 16B，按直接相联方式写出主、缓存地址格式和目录表格式。



共 6 位＝5 位区号＋1 位有效位。

目录表的容量为 2048 个存储单元。

直接相联是一种最简单的地址映像方式，硬件实现很简单。在访问数据时，不需要进行复杂的地址转换，只检查区号是否相等即可。采用的是按地址访问的方式，而不是按内容访问，因此目录表不一定采用相关存储器，利用一个容量小的高速存储部件，可得到比较快的访问速度，硬件设备简单。

直接相联的缺点是 Cache 的块冲突率很高，导致命中率比较低。若两个或两个以上经常使用的块恰好被映像到 Cache 的同一块位置时，就会使 Cache 命中率急剧下降。即使 Cache 中有大量的空闲块也无法使用。早期的 IBM370/158 等计算机中采用的是直接相联映像方式，目前已经很少使用这种映像方式。

3. 组相联映像方式

组相联映像方式是介于直接相联和全相联方式之间的一种折中方案。组相联方式发扬了两者的优点，既能减少块冲突概率，提高 Cache 空间利用率，又能使地址映像机构及

地址变换速度比全相联的简单和快速。

组相联的映像规则如下。

（1）主存与缓存分成相同大小的块。

（2）主存与缓存分成相同大小的组。

（3）主存容量是缓存容量的整数倍，将主存空间按缓存的大小分成区，主存中每一区的组数与缓存的组数相同。

（4）当主存的数据调入缓存时，主存与缓存的组号应相等，也就是各区中的某一块只能存入缓存的同组号的空间内。组内各块地址之间可以任意存放，即组地址是采用直接相联的映像方式，组内的块采用全相联的映像方式。

根据规则，缓存分为 $C_g=C/B$ 组，每组包含 B 块，共有 C_b 块；主存因为是缓存的整数倍，$M_e=M/C$，所以共有 M_e 个区，同样每区有 C_g 组，每组有 B 块，共有 M_b 块。主存地址格式中应包含 4 个字段：区号、区内组号、组内块号、块内地址。缓存中包含 3 个字段：组号、组内块号、块内地址。主存地址与缓存地址的转换有两部分，组地址是按直接相联的方式，按地址进行访问；块地址是采用全相联方式，按内容访问。

组相联的地址转换部件也是采用相关存储器组成，具体映像关系如图 3-16 所示。

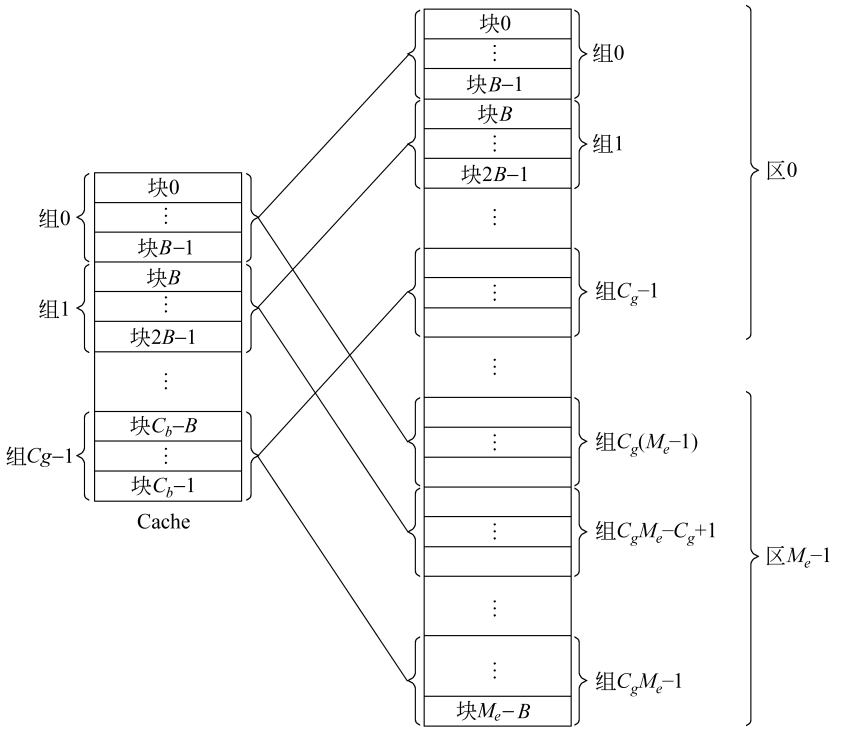


图 3-16　组相联地址映像

具体地址转换过程如图 3-17 所示。

为了实现主存块号到 Cache 块号的变换，需要一个由高速小容量存储器构成的块表存储器。相关存储器的格式为：主存地址中的区号 E_i 与组内块号 B_i，缓存块地址 b_i，其

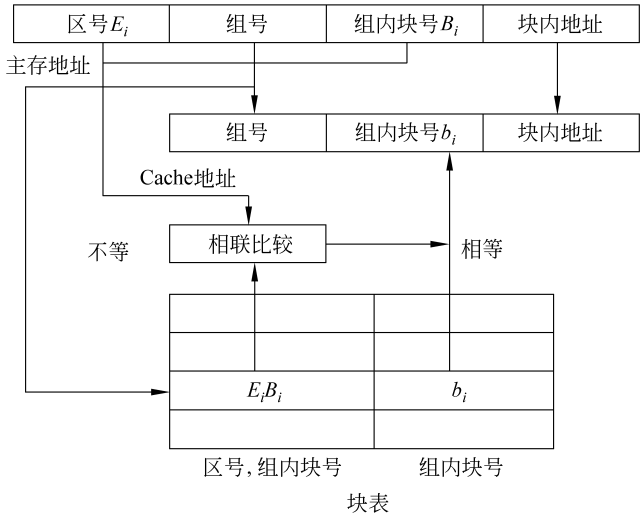


图 3-17　组相联映像的地址转换

中，E_i 和 B_i 组合在一起与 b_i 对应。相关存储器的容量与缓存的块数相同。

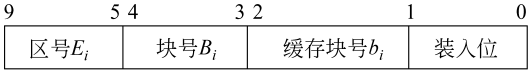
当进行数据访问时，先根据组号，在目录表中找到该组所包含的各块的目录，然后将被访数据的主存区号和组内块地址与各块的目录同时进行比较。如果比较相等，而且有效位为“1”则为命中，可将其对应的缓存块地址 b_i 送到缓存地址寄存器的块地址字段，与组号及块内地址组装即形成缓存地址。如果比较不相等，说明没命中，所访问的数据块尚没有进入缓存，进行组内替换。如果有效位为“0”，则说明缓存的该块尚未利用，或是原来数据作废，可重新调入新块。

举例说明：主存容量为 1MB，缓存容量为 32KB，每块为 64B，缓存共分为 128 组。

(1) 主存与缓存的地址格式。由于缓存分成 128 个组，组号占 7 位地址，所以每组内只包含 4 块。

$$\text{主存的分区数目}=\frac{\text{主存容量}}{\text{缓存容量}}=\frac{2^{20}}{2^{15}}=2^5=32\text{ 个区}$$

(2) 相关存储器的格式及容量。相关存储器的存储单元的格式应为：



相关存储器的容量应与缓存的块数相同，即组数×组内块数＝128×4＝512。需要注意的是，每次访问时，根据组地址找到组号，参加按内容比较的目录只有该组内的 4 项，因此可以得到比较快的转换速度。

组相联映像与全相联映像方式比较，实现起来相对容易，但Cache的命中率却和全相联映像方式很接近，因此，组相联方式应用得比较广泛。

组相联映像与直接相联映像方式比较，块冲突率比较低，因为在直接相联映像方式中，主存中的一块只能映像到Cache中的一个确定块位置。在组相联映像中，如果每组有四块，则主存中的一块可以映像到Cache的四个块的位置。但是，组相联映像方式中组内是相联比较，实现的难度和成本要比直接相联映像高。

另外，当每组的块数只有一块时，组相联就相当于直接相联映像方式。每组的块数与Cache的块数相等时，就相当于全相联映像方式。因此，直接相联映像方式和全相联映像方式是组相联映像方式的两个极端情况。

采用组相联映像方式的典型机器的Cache分组情况如表3-2所示。

表3-2　采用组相联映像方式的典型机器的Cache分组情况

机器型号	Cache的块数 C_b	每组的块数 G_b	Cache组数 C_g
DEC VAX-11/780	1024	2	512
Amdahl 470/V6	512	2	256
Intel i860 D-Cache	256	2	128
Honeywell 66/60	512	4	128
Amdahl 470/V7	2048	4	512
IBM370/168	1024	8	128
IBM3033	1024	16	64
Motorola 88110 I-Cache	256	2	128

从表3-2中可以看出，每组的块数一般都很小。有近一半的机器选择每组为2块，最多的为16块。因为当每个组的块数增加时，需要进行相联访问的存储器的容量将增加，从而查目录表的速度降低，实现的成本随之增加。当每个组的块数太少时，块的冲突概率和Cache的失效率会增大。其他一些新机型的Cache结构如表3-3所示。

表3-3　一些新机型的Cache结构

机器型号	Cache的结构
Pentium Pro	8KB两路组相联指令Cache和8KB的4路组相联数据Cache
龙芯2号	数据Cache和指令Cache各为32KB，2路组相联
Intel Core Microarchitecture（Core 2的2级Cache）	1级Cache分为32KB一级指令Cache和32KB一级数据Cache，都是8路组相联写回缓存，64字节/线，每个核拥有独立的一级Cache，共享二级Cache和总线接口，二级Cache为16路组相联，64字节/线，与一级Cache之间的数据带宽为256位
Pentium MMX Intel Core 2 Duo E7200	片内一级数据和指令的Cache，每个增到16KB，4路相联3MB二级缓存的Core 2 Duo E7200采用16路，相对于采用8路设计的E4000系列增加了一级

续表

机 器 型 号	Cache 的结构
IBM PowerPC 601	只包含一级片上高速缓存，容量为 32KB，组织形式为 8 路组相联，指令和数据合存
IBM PowerPC 604	只在芯片内设有 4 路组相联的 16KB 指令高速缓存和 16KB 的数据高速缓存
IBM PowerPC 620	包含两级片上高速缓存，在芯片内设有 4 路组相联的 32KB 指令高速缓存和 32KB 的数据高速缓存，在芯片外设有 1～128MB 的片外高速缓存

3.2.3 替换算法及实现

从上述地址映像和变换中可知，当 Cache 出现块失效时，需按某种替换算法将 Cache 中的块替换出去。

直接相联映像方式中，由于主存的一块只能存入缓存的对应的一块中，在不命中时只能将所需数据调入该块，因此在替换时没有选择替换算法的问题。全相联映像方式中，不命中时需要替换就必须在全部的 Cache 的块中进行选择，实现起来难度很大。在组相联映像方式中，因为只在缓存组内实现全相联，所以替换时只对组内各块进行选择。衡量 Cache 替换策略的主要指标是命中率。综合命中率、实现的难易及速度的快慢等各种因素，替换策略可分为随机法（RAND 法）、先进先出法（FIFO 法）、最近最少使用法（LRU 法）等。

1. 随机法

这是最简单的一种方法，即随机确定替换的存储块。具体实现方法是使用一个随机数产生器，依据所产生的随机数确定替换的块。PDP-11/70 的 Cache 采用组相联方式，每组只有两块，发生替换时，由一个随机数产生器的状态决定替换出去的块号。这种方法简单、易于实现，但没有依据程序局部性原理，所以命中率比较低。

2. 先进先出法

先进先出法是选择最先调入的块进行替换。这种方法考虑了程序运行的历史状况，但没有正确地反映程序的局部性。其命中率比随机法好些，但还不满足要求。先进先出法的优点是易于实现。

假设 Cache 采用组相联的方式，每组包含 4 块。如果访问主存的块地址顺序为 2，5，4，2，1，3，6，2，1，4，1，如图 3-18 所示是先进先出法替换的操作过程，共命中两次。

具体实现方法可以采用计数器。例如，Solar-16/65 机 Cache 采用组相联方式，每组为 4 块，每块都设定一个两位的计数器，当某块被装入或被替换时该块的计数器清“0”，而同组的其他各块的计算器均加“1”，当需要替换时就选择计数值最大的块被替换掉。具体实现中，还需在目录表中增加一个替换计数器字段，其长度与 Cache 地址中的组内块号字段的长度相同。

主存块地址 执行顺序	2	5	4	2	1	3	6	2	1	4	1
Cache 块0	2*	2	2	2	2	3*	3	3	3	3	1*
Cache 块1		5*	5	5	5	5	6*	6	6	6	6
Cache 块2			4*	4	4	4	4	2*	2	2	2
Cache 块3					1*	1	1	1	1	4*	4
操作	装入	装入	装入	命中	装入	替换	替换	替换	命中	替换	替换

图 3-18　FIFO 替换策略

在有些机型里可以为每个组设置一个计数器，即本组有替换时，计数器加“1”，计数器的值就是要被替换出去的块号。例如，NOVA3 计算机的 Cache 采用组相联映像方式，每组的块数为 8，每组设置一个 3 位计数器。需要替换时，计数器的值加“1”，用计数器的值直接作为被替换块的块号。

总体来说，先进先出法的实现比较简单，但是很有可能出现的情况是最先装入 Cache 的块，也是经常要使用的块，却被替换掉了，从而导致命中率低。

3. 最近最少使用法

LRU 法是指根据各块的使用情况，选择最近最少使用的块作为被替换的块。这种方法比较好地反映了程序局部性规律，因为最近最少使用的块，很可能在将来的近期也很少使用，所以 LRU 法的命中率比较高。但这种方法比较复杂，硬件实现起来比较困难，它不但要记录每块使用次数的多少，而且要反映出近期使用的次数。最久没有使用法（LFU 法）的技术思想与 LRU 法是一样的，只不过实现方法上有所不同。LRU 法是记录近期使用次数的多少，而 LFU 法是记录最久没有使用过，所以后者实现起来比较容易。

假设某 Cache 采用组相联的方式，而每组包含 4 块。如果访问主存的块地址顺序为 2，5，4，2，1，3，6，2，1，4，1，如图 3-19 所示为最久没有使用法的操作过程，共命中 4 次。

主存块地址 执行顺序	2	5	4	2	1	3	6	2	1	4	1
Cache 块0	2*	2	2	2*	2	2	2	2*	2	2	2
Cache 块1		5*	5	5	5	3*	3	3	3	4*	4
Cache 块2			4*	4	4	4	6*	6	6	6	6
Cache 块3					1*	1	1	1	1*	1	1*
操作	装入	装入	装入	命中	装入	替换	替换	命中	命中	替换	命中

图 3-19　LFU 法替换过程

由比较可以看出，LFU 的替换算法命中率比较高。

实现 LFU 法及 LRU 法策略的方法也有多种。下面简单介绍计数器法、寄存器栈法及比较对法的设计思路。

1）计数器法

缓存的每一块都设置一个计数器，具体操作规则如下。

(1) 被调入或者被替换的块,其计数器清 0,而其他的计数器则加 1。

(2) 当访问命中时,所有块的计数值与命中块的计数值进行比较,如果计数值小于命中块的计数值,则该块计数值加"1";如果块的计数值大于命中块的计数值,则数值不变,而命中块的计数器清 0。

(3) 需要替换时,则选择计数值最大的块被替换。例如,IBM370/65 的 Cache 使用组相联方式,每组有 4 块,采用 LFU 替换策略,每一块设置一个 2 位的计数器,其工作状态如图 3-20 所示。

主存访问块地址	块 4		块 2		块 3		块 5	
	块号	计数器	块号	计数器	块号	计数器	块号	计数器
Cache 块 0	1	10	1	11	1	11	5	00
Cache 块 1	3	01	3	10	3	00	3	01
Cache 块 2	4	00	4	01	4	10	4	11
Cache 块 3	空	××	2	00	2	01	2	10
操作	起始状态		装入		命中		替换	

图 3-20　计数器法实现 LFU 策略

2) 寄存器栈法

设置一个寄存器栈,其容量为 Cache 中替换时参与选择的块数。如在组相联方式中,则是同组内的块数。堆栈由栈顶到栈底依次记录主存数据存入缓存的块号,现以一组内 4 块为例说明其工作情况,如图 3-21 所示为寄存器栈法的工作原理,图中 1～4 为缓存中的一组的 4 个块号。

缓存操作	初始状态	装入 2	命中块 4	替换块 1
寄存器 0	3	2	4	1
寄存器 1	4	3	2	4
寄存器 2	1	4	3	2
寄存器 3	空	1	1	3

图 3-21　寄存器栈法工作原理

(1) 当缓存中尚有空闲时,如果不命中,则可直接调入数据块,并将新访问的缓存块号压入堆栈,位于栈顶。其他栈内各单元依次由顶到下顺压一个单元,直到空闲单元为止。

(2) 当缓存已满时,如果数据访问命中,则将访问的缓存块号压入堆栈,其他各单元内容由顶向底逐次下压,直到被命中块号的原来位置为止。

如果访问不命中,说明需要替换。此时栈底单元中的块号即是最久没有被使用的。所以将新访问块号压入堆栈,栈内各单元内容依次下压直到栈底。自然,栈底所指出的块被替换。