第 3 章　存储器层次结构

存储系统是由几个容量、速度和价格各不相同的存储器构成的层次结构系统。本章讨论存储系统的分类和层次结构，半导体随机存取存储器的特点，主存储器与 CPU 连接的一般原则和方法，以及利用双口 RAM、多模块存储器、高速缓冲存储器和虚拟存储器等提高存储系统性能的基本方法。

3.1　408 考纲要求

（一）存储器的分类
（二）存储器的层次化结构
（三）半导体随机存取存储器
1. SRAM 存储器
2. DRAM 存储器
3. 只读存储器
4. Flash 存储器
（四）主存储器与 CPU 的连接
（五）双口 RAM 和多模块存储器
（六）高速缓冲存储器（Cache）
1. Cache 的基本工作原理
2. Cache 和主存之间的映射方式
3. Cache 中主存块的替换算法
4. Cache 写策略
（七）虚拟存储器
1. 虚拟存储器的基本概念
2. 页式虚拟存储器

3. 段式虚拟存储器
4. 段页式虚拟存储器
5. TLB(快表)

3.2 知识结构梳理

3.2.1 存储器的分类

1. 按存储器在计算机系统中的作用分类

如果将存储器按照其在计算机系统中起到的作用来分类,可以分为高速缓冲存储器(Cache)、主存储器、辅助存储器(外存储器)。

2. 按存取方式分类

如果将存储器按照其存取方式来分类,可以分为随机存取存储器(RAM)、只读存储器(ROM)、顺序存取存储器(SAM)、直接存取存储器(DAM)。

3. 按存储介质分类

如果将存储器按照其存储介质来分类,可以分为磁心存储器、半导体存储器、磁表面存储器、光存储器。

4. 按信息的可保存性分类

断电后存储信息即消失的存储器称为易失性存储器。断电后信息仍然保存的存储器称为非易失性存储器。

如果某个存储单元所存储的信息被读出时,原存信息将被破坏,则称破坏性读出;如果读出时,被读单元原存信息不被破坏,则称非破坏性读出。对于破坏性读出的存储器,每当一次读出操作之后,必须紧接一个重写(再生)操作,以便恢复被破坏的信息。

3.2.2 存储器的层次化结构

为了解决存储容量、存取速度和价格之间的矛盾,通常把各种不同存储容量、不同存取速度的存储器按一定的体系结构组织起来,形成一个统一整体的存储系统。

由高速缓冲存储器、主存储器、辅助存储器构成的三级存储系统可以分为两个层次，其中高速缓存和主存间称为 Cache-主存存储层次（Cache 存储系统），主存和辅存间称为主存-辅存存储层次（虚拟存储系统）。

3.2.3 半导体随机存取存储器

1. SRAM

通常把存放一个二进制位的物理器件称为记忆单元，地址码相同的多个记忆单元构成一个存储单元。SRAM（静态随机存取存储器）的记忆单元是用双稳态触发器来记忆信息的。

SRAM 存取速度快，但集成度低，功耗也较大，所以一般用来组成高速缓冲存储器和小容量主存系统。

2. DRAM

DRAM（动态随机存取存储器）是利用记忆单元电路中栅极电容上的电荷来存储信息的。由于栅极电容上的电荷会随着时间的推移不断泄漏，所以每隔一定的时间必须向栅极电容补充一次电荷，这个过程称为刷新。

DRAM 集成度高，功耗小，但存取速度慢，一般用来组成大容量主存系统。

3. 只读存储器

只读存储器（ROM）的内容只能随机读出而不能写入，这类存储器用来存放那些不需要改变的信息。ROM 中的内容是事先写入的，写入的过程称为对 ROM 编程，根据编程方法的不同，ROM 通常可以分为掩模式 ROM、一次可编程 ROM、可擦除可编程 ROM 3 类，而擦除的方式有紫外线擦除和电擦除两种。

4. Flash 存储器

Flash 存储器又称为闪速存储器，这是 20 世纪 80 年代中期出现的一种快擦写型存储器。它的主要特点是既能在不加电的情况下长期保存信息，又能在线进行快速擦除与重写，兼具电擦除可编程 ROM 和 RAM 的优点。

3.2.4 主存储器与 CPU 的连接

主存储器是整个存储系统的核心，包括 RAM 和 ROM 两大部分，用来存放计算机运行期间需要的程序和数据。其中，ROM 区只能读不能写，常用于存放系统程序或不需要修改的数据；RAM 区既可读又可写，常用于存放用户程序、数据或作为系统程序的工作区。

1. 主存储器的存储单元

位是二进制数的最基本单位，一个二进制数由若干位组成，当它作为一个整体存入或取出时，这个数称为存储字。存放存储字或存储字节的主存空间称为存储单元，存储单元是 CPU 对主存可访问的最小存储单位。

2. 主存容量的扩展

要组成一个主存，首先要考虑选片的问题，然后就是如何把芯片连接起来的问题。根据存储器所要求的容量和选定的存储芯片的容量，可以计算出总的芯片数。

主存扩展的方式有位扩展、字扩展、字和位同时扩展3种。

3. 存储芯片的地址分配和片选

CPU与存储器连接时，特别是在扩展存储容量的场合下，主存的地址分配是一个重要的问题。确定地址分配后，又有存储芯片的片选信号的产生问题。片选信号的产生可细分为线选法、全译码法和部分译码法。

4. 主存和CPU的连接

主存和CPU之间的连接包括硬连接和软连接。硬连接既包含存储芯片之间的连接，也包含整个主存与CPU之间的连接；而软连接就是CPU向主存发出的读或写命令，这是这两个部件之间有效工作的关键。

3.2.5 双口RAM和多模块存储器

采用双口RAM和多模块存储器，可以在一个存取周期内并行地读写多个字，从而提高存储器的访问速度。

1. 双口RAM

双口RAM是指同一个存储器具有两组相互独立的读写控制电路，是一种高速工作的存储器。它有两个独立的端口，分别具有各自的地址线、数据线和控制线，可以对存储器中任何位置上的数据进行独立的存取操作。

2. 多模块存储器

多模块存储器的每个模块都有独立的地址寄存器、数据寄存器、地址译码电路、驱动电路和读/写电路，它们既能并行工作，又能交叉工作。

多模块交叉存储器是线性编址的，地址在各模块中有两种安排方式，分别是高位交叉编址（顺序方式）和低位交叉编址（交叉方式）。

3.2.6 高速缓冲存储器

1. Cache的基本工作原理

程序的局部性有两个方面的含义：时间局部性和空间局部性。时间局部性是指如果一个存储单元被访问，则该单元可能很快被再次访问。空间局部性是指如果一个存储单元被访问，则该单元邻近的单元也可能很快被访问。高速缓冲技术正是利用了程序的局部性原理。

2. Cache 和主存之间的映射方式

在 Cache 中，地址映射是指把主存地址空间映射到 Cache 地址空间，也就是把存放在主存中的程序按照某种规则装入 Cache 中。地址映射的方式有全相联映射、直接映射和组相联映射 3 种。

3. Cache 中主存块的替换算法

在采用全相联映射和组相联映射方式从主存向 Cache 传送一个新块，而 Cache 中的空间已被占满时，就需要把原来存储的一块替换为新块。常用的替换算法有随机算法、先进先出算法、近期最少使用算法 3 种。

4. Cache 写策略

为了解决 Cache 与主存内容不一致的问题，需要选择合适的 Cache 更新策略。Cache 有两种更新策略：直写法和写回法。

3.2.7 虚拟存储器

1. 虚拟存储器的基本概念

虚拟存储器将主存或辅存的地址空间统一编址，形成一个庞大的存储空间。在这个大空间里，用户可以自由编程，用户编程的地址称为虚地址或逻辑地址，实际的主存单元地址称为实地址或物理地址。程序运行时，CPU 以虚地址访问主存，由辅助硬件找出虚地址和实地址之间的对应关系。

2. 页式虚拟存储器

以页为基本单位的虚拟存储器叫页式虚拟存储器。主存空间和虚存空间都划分成若干大小相等的页。页式虚拟存储器的每页长度是固定的，页表的建立很方便，新页的调入也容易实现。但是由于程序不可能正好是页的整倍数，最后一页的零头将无法利用而造成浪费。同时，页不是逻辑上独立的实体，使程序的处理、保护和共享都比较麻烦。

3. 段式虚拟存储器

以段为基本单位的虚拟存储器叫段式虚拟存储器。段式虚拟存储器中的段是按照程序的逻辑结构划分的，各个段的长度因程序而异。由于段的分界与程序的自然分界相对应，所以段具有逻辑独立性，便于程序的编译、管理、修改和保护，也便于多道程序共享。但是，因为段的长度参差不齐，起点和终点不定，给主存空间分配带来了麻烦；容易在段间留下不能利用的零头，造成浪费。

4. 段页式虚拟存储器

段页式虚拟存储器是将程序按其逻辑结构分段，每段再划分为若干页；主存空间也划分为若干同样大小的页，虚存和实存之间以页为基本传送单位。段页式存虚拟存储器综合了前两种结构的优点，但要经过两级查表才能完成地址转换，比较费时。

5. 快表(TLB)

为了尽可能提高速度,借鉴 Cache 的思路,将页表中最活跃的部分放在高速存储器中构成快表,快表扮演的角色是作为页表的 Cache。对快表的查找和管理全用硬件来实现。

3.3 重点难点分析

1. 存储系统和存储器

在同一台计算机中,有各种工作速度、存储容量、访问方式、用途等均不相同的存储器,这些存储器构成层次结构,如图 3-1 所示。从上到下,各种存储器的存储容量越来越大,每位的价格越来越便宜,但存取周期越来越长。

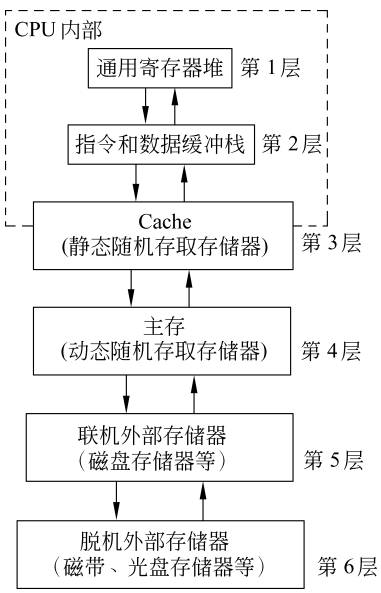


图 3-1 存储器的层次结构

并非将各种用途不同的存储器放在一起就构成了一个存储系统。存储系统是指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、软件或硬软件相结合的方法连接起来的一个系统。这个系统对应用程序员透明,可以把它看作一个存储器,其速度接近速度最快的那个存储器,其存储容量与容量最大的那个存储器相等或接近,其单位容量的价格接近最便宜的那个存储器。

所以,存储系统和存储器是两个完全不同的概念。如果在一台计算机中只有存储器,甚至有多种存储器,但没有构成存储系统,这台计算机的性能将会是很差的,这些存储器的性能也不可能得到充分的发挥。

2. 主存储器组织

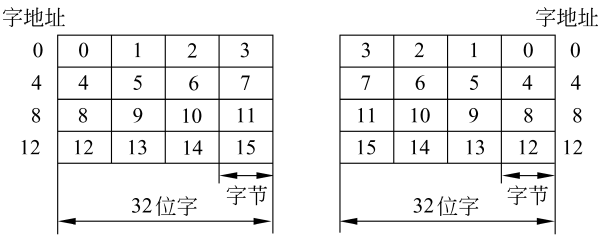
主存储器的核心是存储体,程序和数据都存放在存储体中。存储体是由若干存储单元组成的,存储单元的编号称为地址,地址和存储单元之间有一对一的对应关系。这就像一座大楼有许多房间,而每个房间都有其唯一的房间号一样。

位是二进制数的最小单位,是半导体存储器的基本记忆单元。存储字由若干二进制位组成,可以作为一个整体存入或取出。一个存储单元可能存放一个字,也可能存放一字节,这是由计算机的结构确定的。对于字编址的计算机,最小寻址单位是一个字,相邻的存储单元地址指向相邻的存储字;对于字节编址的计算机,最小寻址单位是一字节,相邻的存储单

元地址指向相邻的存储字节。所以,存储单元是 CPU 对主存可访问的最小存储单位,根据存储单元的地址可以找到相应存储单元的内容。

3. 字节编址计算机的大端方式和小端方式

许多数据格式使用多字节来表示一个数值。有两种常用的多字节数据排列顺序,即每个字中的字节顺序可以从左到右或从右到左排列,前者称为大端方式,后者称为小端方式。Intel 80x86 是采用小端方式的计算机,IBM 370、Motorola 680x0 和大多数 RISC 计算机则采用大端方式,Power PC 是既支持大端方式又支持小端方式的计算机。图 3-2(a)描述的是使用大端方式的 32 位计算机的一段主存,图 3-2(b)描述的是使用小端方式的 32 位计算机的一段主存。其中每个存储单元上的数字表示字节顺序。



(a) 大端方式　　(b) 小端方式

图 3-2　大端方式和小端方式

例如,十六进制数 01020304H 从存储单元 100H 开始存储,则存储结果如表 3-1 所示。依照大端方式,最高字节存储在 100H 中,次高字节存储在 101H 中,以此类推;依照小端方式,最低字节存储在单元 100H 中,次低字节存储在 101H 中,以此类推。

表 3-1　大端方式和小端方式存储结果

大端方式		小端方式	
存储器地址	数据(十六进制)	存储器地址	数据(十六进制)
100	01	100	04
101	02	101	03
102	03	102	02
103	04	103	01

大端方式和小端方式存放 ASCII 码字符串和 BCD 码数据的顺序是相反的,但是必须指出,不管是上述哪个系统,在表示一个 32 位整数时两个方式是一致的。例如整数 6,都是在最右边的(最低位)3 位上存放 110,前面 29 位都是 0。也就是说,采用大端方式,110 这 3 位应该放在字节 3(或 7、11 等)中;而采用小端方式,110 这 3 位应该放在字节 0(或 4、8 等)中。

4. 主存储器的存取时间和存取周期

存取时间(T_a)又称为访问时间或读写时间,是指从启动一次存储器操作到完成该操作所经历的时间。例如,读出时间是指从 CPU 向存储器发出有效地址和读命令开始,直到将

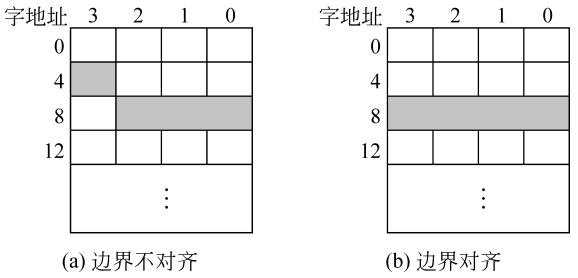
被选单元的内容读出为止所用的时间;写入时间是指从 CPU 向存储器发出有效地址和写命令开始,直到信息写入被选单元为止所用的时间。显然,T_a 越小,存取速度越快。

存取周期(T_m)又称为读写周期、访存周期,是指存储器进行一次完整的读写操作所需的全部时间,即连续两次访问存储器操作的开始时间之间的最短时间。显然,一般情况下,$T_m > T_a$。这是因为,对于任何一种存储器,在读写操作之后,总要有一段恢复内部状态的复原时间。对于破坏性读出的存储器,存取周期往往比存取时间要大得多,甚至可以达到 $T_m = 2T_a$,这是因为存储器中的信息读出后需要马上进行重写(再生)。

与存取周期密切相关的指标是主存带宽(B_m),它又称为数据传输率,表示主存每秒进出信息的最大数量,单位为字节每秒(B/s)或位每秒(b/s)。

5. 边界对齐的数据存放方法

现代计算机在某一时刻可以读出多个字节的数据。在数据对齐存储方式下,要求一个数据字占据完整的一个存储字的位置,而不是分成两部分。例如,一个 32 位的数据字放在 32 位宽度的主存储器中,若字地址为 n,则在对齐方式下数据实际占据的是字节地址为 n、$n+1$、$n+2$ 和 $n+3$ 的存储单元,这个数据字可以一次读取或者写入。如果这个数据字不按对齐方式存储,假设数据字实际占据的是字节地址为 $n-1$、n、$n+1$ 和 $n+2$ 的存储单元,这样的数据字在 32 位的主存中需要分两次读取或者写入。在有些计算机中,规定数据的存储必须按边界对齐的方式进行。图 3-3(a)是一个边界不对齐的例子,图 3-3(b)是一个边界对齐的例子。



(a) 边界不对齐　　(b) 边界对齐

图 3-3　边界不对齐和边界对齐的例子(小端方式)

边界对齐简单地说就是使存储多字节值的起始单元刚好是某个多字节读取模块的开始单元,所以边界对齐的数据存放方式对数据的存放位置有下列要求:

- 8 位数据占用一个存储单元,其地址为×…×××(任意)。
- 16 位数据占用两个存储单元,存放数据的起始地址为×…××0(2 的整倍数)。
- 32 位数据占用 4 个存储单元,存放数据的起始地址为×…××0 0(4 的整倍数)。
- 64 位数据占用 8 个存储单元,存放数据的起始地址为×…×0 0 0(8 的整倍数)。

6. 动态随机存储器的刷新

动态随机存储器利用栅极电容来存储信息,电容上的电荷会随着时间推移泄漏,必须定时刷新。

在讨论刷新问题时,首先要搞清楚刷新和重写(再生)这两个完全不同的概念。重写仅

针对破坏性读出的存储器(如磁芯、单管动态 RAM 等),重写是随机的,某个存储单元只有在破坏性读出之后才需要重写;刷新则针对动态 RAM,刷新是定时的,许多记忆单元长期未被访问,若不及时补充电荷,信息也会丢失。重写是按存储单元进行的,破坏性地读出了哪个单元,就只对哪个单元重写,而不需要涉及其他的存储单元;而刷新则不论某个单元是否被读出均需要进行,所以刷新是以存储体矩阵中的一行为单位进行的。

常见的刷新方式有以下 3 种:

(1) 集中刷新方式。

集中刷新是指在允许的最大刷新间隔内,按照存储芯片容量的大小集中安排若干刷新周期,刷新时停止读写操作。

$$\text{刷新时间}=\text{存储矩阵行数}\times\text{刷新周期}$$

这里刷新周期是指刷新一行所需要的时间,由于刷新过程就是假读的过程,所以刷新周期就等于存取周期。

集中刷新方式的优点是读写操作时不受刷新工作的影响,因此系统的存取速度比较快。其主要缺点是在集中刷新期间必须停止读写,这一段时间称为死区,而且存储容量越大,死区就越长。

(2) 分散刷新方式。

分散刷新是指把刷新操作分散到每个存取周期内进行,此时系统的存取周期被分为两部分,前一部分时间进行读写操作或保持,后一部分时间进行刷新操作。在一个系统存取周期内刷新存储矩阵中的一行。

分散刷新方式没有死区,但是它有两个明显的缺点:第一是加长了系统的存取周期,降低了整机的速度;第二是刷新过于频繁,尤其是当存储容量比较小的情况下,没有充分利用存储器允许的最大刷新间隔(2ms)。

(3) 异步刷新方式。

异步刷新方式是前述两种方式的结合,它充分利用最大刷新间隔时间,把刷新操作平均分配到整个最大刷新间隔时间内进行,故有:

$$\text{相邻两行的刷新间隔}=\text{最大刷新间隔}\div\text{行数}$$

异步刷新方式虽然也有死区,但比集中刷新方式的死区小得多。

刷新通常是一行一行地进行的,每一行中各记忆单元同时被刷新,故仅需要行地址,不需要列地址。

整个存储器中的所有芯片同时被刷新。在考虑刷新问题时,应当从单个芯片的存储容量着手,而不是从整个存储器的容量着手。

7. 用若干芯片构成主存

主存是整个存储系统的核心,通常分为 RAM 和 ROM 两大部分。RAM 和 ROM 在主存中是统一编址的。

存储芯片的容量是有限的,主存往往要由一定数量的芯片构成。根据主存所要求的总容量和选定的存储芯片的容量,就可以计算出芯片数,即

$$\text{芯片数}=\frac{\text{总容量}}{\text{存储芯片容量}}$$

1）位扩展

当主存的字数与单个存储芯片的字数相同而位数（字长）不同时，可采用位扩展方式组织多个芯片构成主存。

位扩展仅在位数方向扩展（加大字长），位扩展的连接方式是将各存储芯片的地址线、片选线和读写线相应地并联起来，而将各芯片的数据线单独列出。

2）字扩展

当主存的字长与单个存储芯片的字长相同而字数不同时，可采用字扩展方式组织多个芯片构成主存。

字扩展仅在字数方向扩展，而位数不变。字扩展将芯片的地址线、数据线、读写线相应地并联，由片选信号区分各个芯片。字扩展时有地址分配问题，地址线的高位部分通过译码器产生若干片选信号 CS_i，分别选中若干芯片中的一个。

3）字和位同时扩展

实际中更多的是存储芯片的字数和字长均不能满足主存总容量要求的情况，这时要采用字和位同时扩展的方式构成主存。

8. CPU 的访存地址分配

CPU 访问主存时需要给出地址码，其长度取决于 CPU 可直接访问的最大存储空间，一般要将其地址码分成片内地址和片选地址两部分。

片内地址由低位的地址码构成，其长度取决于所选存储芯片的字数。例如，芯片容量为 8K×4 位和 8K×1 位，它们的片内地址相同，均为 13 位（$2^{13}=8\times2^{10}$，即“8K”），片内地址用于从选中的芯片中选择相应的存储单元，以进行数据的存取，故又称为字选。

片选地址由高位的地址码构成，用于选择存储芯片。大多数情况下由片选地址通过译码后产生存储芯片的片选信号（$\overline{CS}$）。

9. 片选地址的全译码和部分译码

所谓全译码方式是指所有的片选地址全部参加译码。有两种情况必须采用全译码方式。

（1）实际使用的存储空间与 CPU 可访问的最大存储空间相同。

例如，CPU 给出的访存地址长 16 位（$A_{15}\sim A_0$），即可访问的最大存储空间为 64KB，选用的存储芯片容量为 16K×4 位，共 8 片，构成 4 个小组，这时片内地址为 14 位，片选地址为 2 位。2 位片选地址必须全部参加译码才能产生 4 个选片信号，分别用作 4 个小组的片选信号。

（2）实际使用的存储空间小于 CPU 可访问的最大存储空间，而对实际空间的地址分配有严格的要求。

例如，CPU 给出的访存地址长为 16 位（$A_{15}\sim A_0$），即可访问的最大存储空间为 64KB，而系统中实际使用的存储空间只有 8KB，且选用存储容量为 4K×2 位的芯片共 8 片，并要

求其地址范围必须是 4000H～5FFFH，其地址译码方式如图 3-4 所示。

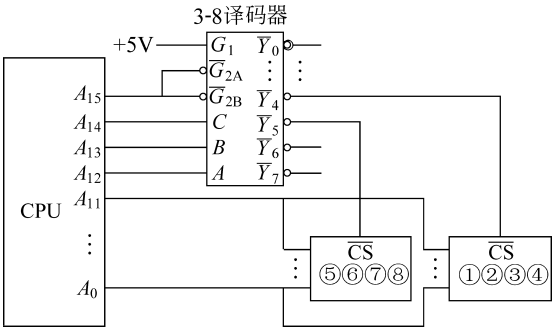


图 3-4　地址码采用全译码方式

从图 3-4 可看出，其地址分配如表 3-2 所示。

表 3-2　全译码方式的地址分配

所选芯片号	片选地址 $A_{15}A_{14}A_{13}A_{12}$	片内地址 $A_{11}A_{10}A_9\cdots A_0$	译码输出	地址分配
	0 0 0 0 0 0 0 0	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_0=0$	0000H～0FFFH
	0 0 0 1 0 0 0 1	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_1=0$	1000H～1FFFH
	0 0 1 0 0 0 1 0	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_2=0$	2000H～2FFFH
	0 0 1 1 0 0 1 1	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_3=0$	3000H～3FFFH
①②③④	0 1 0 0 0 1 0 0	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_4=0$	4000H～4FFFH
⑤⑥⑦⑧	0 1 0 1 0 1 0 1	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_5=0$	5000H～5FFFH
	0 1 1 0 0 1 1 0	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_6=0$	6000H～6FFFH
	0 1 1 1 0 1 1 1	0 0 0⋯ 0 1 1 1⋯ 1	$\overline{Y}_7=0$	7000H～7FFFH

从表 3-2 中可以看出，按照这种译码方式，当前使用的存储空间的地址范围被严格地定义为 4000H～5FFFH，根据图 3-4 可知，最大可扩充到 32KB 的存储空间。如果要将主存储器容量扩充到 64KB，只需将译码电路改为 4-16 译码器即可。

全译码方式的共同特点是使用的存储芯片的地址范围是唯一的。

实际使用的存储空间比 CPU 可访问的最大存储空间小，而且对其地址分配没有严格要求的情况下，可采用部分译码方式。

例如，CPU 可提供的地址为 16 位（A_{15}～A_0），而实际使用的存储空间为 16KB，拟采用 4K×4 位的存储芯片共 8 片，则可采用的部分译码方式如图 3-5 所示。

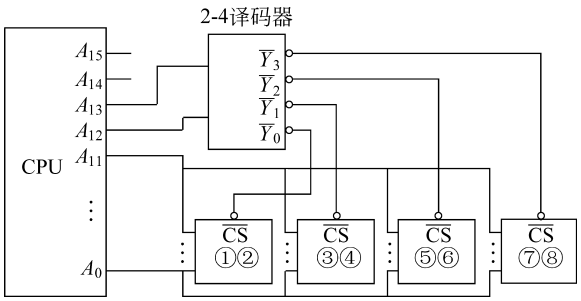


图 3-5　地址码采用部分译码方式

由于采用部分译码方式，使得各组芯片的地址范围不再是唯一的。4 组芯片的地址分配如下：

第 1 组：0000H～0FFFH，4000H～4FFFH，8000H～8FFFH，C000H～CFFFH。

第 2 组：1000H～1FFFH，5000H～5FFFH，9000H～9FFFH，D000H～DFFFH。

第 3 组：2000H～2FFFH，6000H～6FFFH，A000H～AFFFH，E000H～EFFFH。

第 4 组：3000H～3FFFH，7000H～7FFFH，B000H～BFFFH，F000H～FFFFH。

可以看出，采用部分译码方式时，各组芯片出现了重叠的地址范围，其地址重叠区的个数取决于没有参加译码的地址码的位数，由于有两位地址码（A_{15}、A_{14}）没有参加译码，所以每组芯片都出现 4 个地址重叠区。

10. CPU 对主存的基本操作

主存和 CPU 之间的连接包括硬连接和软连接。其中软连接就是 CPU 对主存的基本操作，这是这两个部件之间有效工作的关键。CPU 对主存进行读写操作时，首先在地址总线上给出地址信号，然后发出相应的读或写命令，并在数据总线上交换信息。读写的基本操作如下。

（1）读操作是指从 CPU 送来的地址指定的存储单元中取出信息，再送给 CPU，其操作过程如下：

```
地址→MAR→AB                (CPU 将地址信号送至地址总线)
Read                        (CPU 发读命令)
Wait for MFC                (等待存储器工作完成信号)
M (MAR)→DB→MDR              (读出信息经数据总线送至 CPU)
```

（2）写操作是指将要写入的信息存入 CPU 指定的存储单元中，其操作过程如下：

```
地址→MAR→AB                (CPU 将地址信号送至地址总线)
数据→MDR→DB                (CPU 将要写入的数据送至数据总线)
Write                       (CPU 发写命令)
Wait for MFC                (等待存储器工作完成信号)
```

11. 多模块存储器

多模块存储器的各模块具有相同的容量和存取速度，各模块都有独立的地址寄存器、数

据寄存器、地址译码电路、驱动电路和读/写电路，它们既能并行工作，又能交叉工作。

多模块交叉存储器是线性编址的，地址在各模块中有高位交叉编址和低位交叉编址两种方式。高位交叉编址的多模块存储器用地址码的高位区分存储模块，地址码的低位选择存储单元；低位交叉编址的多模块存储器用地址码的低位区分存储模块，地址码的高位选择存储单元。

低位交叉访问存储器如图 3-6 所示。存储器地址寄存器的低位部分经过译码选择不同的存储体，而高位部分则指向存储体内的存储字。现以由 4 个分体组成的低位交叉存储器为例，说明常用的编址方式。4 个分体 M_0、M_1、M_2、M_3 的编址序列如表 3-3 所示，称为模 4 交叉编址序列。

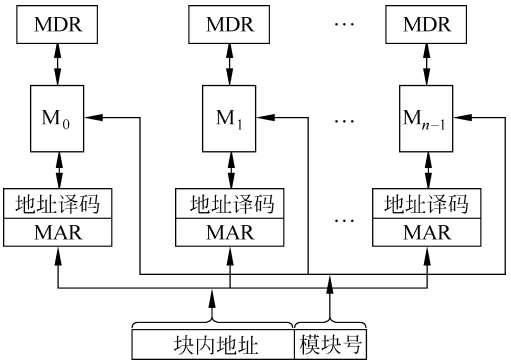


图 3-6 低位交叉访问存储器

表 3-3 模 4 交叉编址序列

模 块 号	地址编址序列	对应二进制地址的最低两位
M_0	0,4,8,12,…,$4i$+0,…	00
M_1	1,5,9,13,…,$4i$+1,…	01
M_2	2,6,10,14,…,$4i$+2,…	10
M_3	3,7,11,15,…,$4i$+3,…	11

在这种交叉存储器中，连续的地址分布在相邻的存储体中，而同一存储体内的地址都是不连续的。这种方式可以在不改变每个模块存取周期的前提下，提高整个主存的速度。例如，有 4 个模块，在第一个存取周期的开始时刻启动模块 M_0，在$\frac{T_m}{4}$、$\frac{T_m}{2}$、$\frac{3T_m}{4}$时刻分别启动模块 M_1、M_2、M_3，在 4 个分体完全并行的理想情况下，整个主存的有效周期缩小到原来的模块存取周期的 1/4，数据传送的平均速度提高到原来的 4 倍。

12. Cache 的基本工作原理

Cache 的工作原理基于程序访问的局部性，程序访问的局部性有时间局部性和空间局部性两方面的含义。时间局部性是指，如果一个存储单元被访问，则该存储单元可能会很快被再次访问。这是因为程序存在着循环。空间局部性是指，如果一个存储单元被访问，则该

存储单元邻近的存储单元也可能很快被访问。这是因为程序中大部分指令是顺序存储、顺序执行的，数据一般也是以向量、数组、树、表等形式簇聚地存储在一起的。

CPU 访存时的基本原则是由近到远：首先访问 M_1，若在 M_1 中找不到所要的数据，就要访问 M_2，将包含所需数据的块或页调入 M_1；若在 M_2 中还找不到，就要访问 M_3；依此类推。

利用程序访问的局部性原理，把程序中正在使用的部分存放在一个高速的、容量较小的 Cache 中，使 CPU 的访存操作大多数针对 Cache 进行，从而使程序的执行速度大大提高。

Cache 和主存都被分成若干大小相等的块，每块由若干字节组成。由于 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，它保存的信息只是主存中最急需执行的若干块的副本。当 CPU 发出主存地址后，首先判断该存储字是否在 Cache 中，若是（称为命中），则直接访问 Cache；若不是（称为不命中或失效），则访问主存并将该字所在的主存块装入 Cache。

命中率 H 定义为 CPU 产生的逻辑地址能在 Cache 中访问到的概率。在一个程序执行期间，设 N_1 为访问 Cache 的命中次数，N_2 为访问主存的次数，则

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

Cache-主存存储层次的等效访问时间 T_A 与主存访问的启动时间有关：

- 若 Cache 访问和主存访问是同时启动的，$T_A=H\times T_{A1}+(1-H)\times T_{A2}$。
- 若 Cache 不命中时才启动主存，$T_A=T_{A1}+(1-H)\times T_{A2}$。

Cache-主存存储层次的访问效率为 $e=\frac{T_{A1}}{T_A}$。

13. Cache 和主存之间的映射方式

为便于在 Cache 和主存间交换信息，Cache 和主存都被划分为相等的区域。主存的区域称为主存块，它是 Cache 与主存之间交换信息的单位，Cache 中存放主存块的区域称为行（块）。为更好地加以区分，在这里统一描述为主存块和 Cache 行。注意，它们的大小是相等的。

由于 Cache 的容量小，因此 Cache 中的内容会经常地被新的主存块替换，它们之间就存在地址映射问题。常见的地址映射方法有 3 种：全相联映射、直接映射和组相联映射。

1）全相联映射

全相联映射就是让主存中的任何一个块均可以映射到（装入）Cache 中任何一个行（块）的位置上，如图 3-7(a)所示。全相联映射方式比较灵活，Cache 的块冲突概率最低，空间利用率最高，但是地址变换速度慢，而且成本高，实现起来比较困难。

2）直接映射

直接映射是指主存中的每一个块只能被放置到 Cache 中唯一的指定位置上，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。直接映射方式是最简单的地址映射方式，成本低，易实现，地址变换速度快，但这种方式不够灵活，Cache 的块冲突概率最高，空间利用率最低。

直接映射规则如图 3-7(b)所示。例如，主存的第 0 块、第 8 块只能映射到 Cache 的第 0 行，而主存的第 1 块、第 9 块只能映射到 Cache 的第 1 行，等等。直接映射的关系可定义为

$$K = I \bmod 2^c$$

式中，K 为 Cache 的行号，I 为主存的块号，2^c 为 Cache 行数。

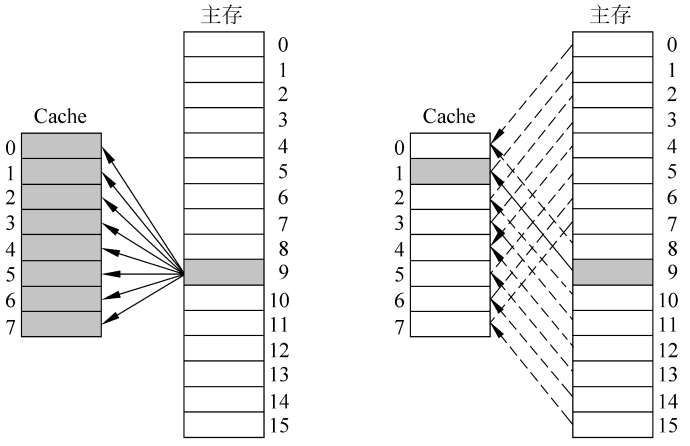
3）组相联映射

组相联映射将 Cache 空间分成大小相同的组，让主存中的一块直接映射到 Cache 中对应组的任何一行位置上，即，组间采取直接映射，而组内采取全相联映射。因此，组相联映射实际上是全相联映射和直接映射的折中方案，其优点和缺点介于全相联映射和直接映射方式之间。当组数等于 1（不再分组）时，组相联映射就变成全相联映射；当组数等于 Cache 中行的数目时，组相联映射就变成直接映射。

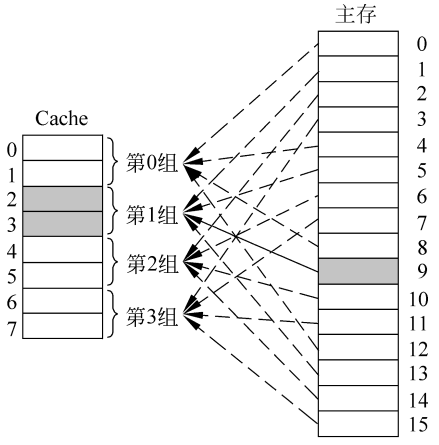
组相联映射规则如图 3-7(c)所示。Cache 分为 4 组，每组两行。主存的第 9 块将可以映射到 Cache 第 1 组的位置上。组相联映射的关系可以定义为：

$$J = I \bmod Q$$

式中，J 为 Cache 的组号，I 为主存的块号，Q 为 Cache 的组数。



(a) 全相联映射　　(b) 直接映射



(c) 组相联映射

图 3-7　3 种映射规则

14. 3种不同映射方式下主存地址的变换

Cache中的信息来自主存中的某个块，在将主存块复制到Cache中时，主存块和Cache行之间必须遵循以上3种映射规则之一。这样，CPU在访问某个主存单元时，就可以依据映射规则，到Cache中查找要访问的信息，而不用在整个Cache中查找。假定主存容量1×2^{20}字（为方便书写，以下将1×2^{20}写为1M），Cache容量（指数据区）为8×2^{10}字（以下将8×2^{10}写为8K），块大小为512字，按字编址。下面是在3种不同映射下的地址映射以及CPU对主存单元0240CH的访问情况。

1）全相联映射

Cache容量为8K字，分为16行（8K÷512）；主存容量为1M字，分为2048块（1M÷512）。20位的主存地址高11位为标记字段，低9位为块内地址字段。主存地址0240CH写成二进制为0000 0010 0100 0000 1100。

访问0240CH单元的过程是：首先根据高11位标记00000010010与Cache中每一行的标记进行比较，若有一个相等并且对应有效位为1，则命中，CPU根据块内地址000001100从该Cache行中取出信息；若都不相等，则不命中，此时需要将0240CH单元所在的主存第00000010010块（即第18块）复制到Cache的任意一个空闲行中，并置有效位为1，置标记为00000010010。

2）直接映射

因为主存的每16块与Cache的16行对应，所以20位的主存地址高7位为标记字段，中间4位为索引（Cache块号）字段，低9位为块内地址字段。

访问0240CH单元的过程是：首先根据中间4位索引0010找到Cache第2行，再将高7位标记0000001与Cache第2行中的标记进行比较，若相等并且对应有效位为1，则命中，CPU根据块内地址000001100从该Cache行中取出信息；若不相等，则不命中，此时需要将0240CH单元所在的主存第00000010010块（即第18块）复制到Cache的第0010行（即第2行），并置有效位为1，置标记为0000001（因为主存的每16块和Cache的16行一一对应，所以将主存的每16块看成一个块群，主存第18块位于主存第1块群）。

3）组相联映射

以2路组相联为例，将原来的一维结构调整为二维结构，Cache的所有行分为8组（16÷2），所以20位的主存地址高8位为标记字段，中间3位为索引（Cache组号）字段，低9位为块内地址字段。

访问0240CH单元的过程是：首先根据中间3位索引010找到Cache第2组，再将高8位标记00000010与Cache第2组中两行的标记同时比较，若有一个相等并且对应有效位为1，则命中，CPU根据块内地址000001100从该Cache行中取出信息；若都不相等，则不命中，此时需要将0240CH单元所在的主存第00000010010块（即第18块）复制到Cache第010组（即第2组）的任意一个空闲行中，并置有效位为1，置标记为00000010（因为主存的每8块和Cache的8组一一对应，所以将主存的每8块看成一个组群，主存第18块位于主存第2组群）。

15. 组相联映射方式的两个映射方案

相比于全相联映射和直接映射，对组相联映射的理解要困难一些。通常将组内两行的

组相联映射称为 2 路组相联映射，组内 4 行的组相联映射称为 4 路组相联映射。关于组相联映射方式的具体映射实现方案，目前在不同的教材中有两种不同的说法，以 2 路组相联为例，图 3-8(a)所示的方案称为方案一，图 3-8(b)所示的方案称为方案二。

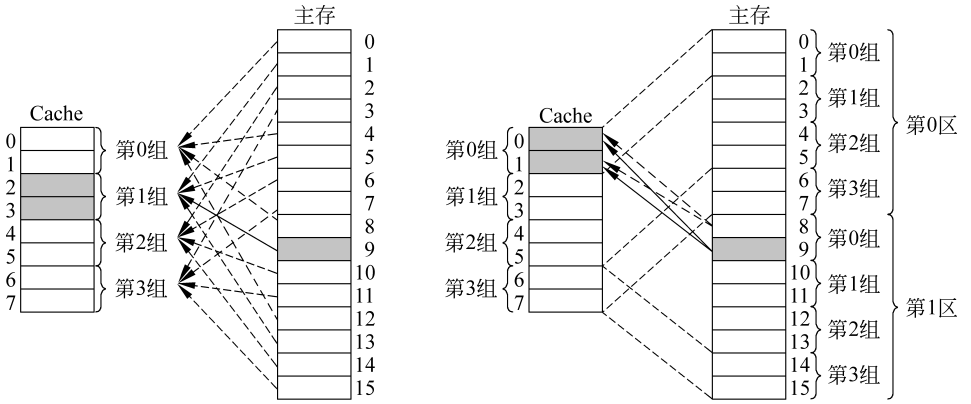


图 3-8　2 路组相联映射方式的两个具体映射实现方案

例如，主存的第 9 块，按方案一，将映射到 Cache 的第 1 组中，放在 Cache 的第 2 行或第 3 行的位置上；而按方案二，将映射到 Cache 的第 0 组中，放在 Cache 的第 0 行或第 1 行的位置上。比较这两个方案可以发现，两者的区别在于主存是否要按照 Cache 的大小再分区。这两种方案对应的主存地址是有区别的，如图 3-9 所示。

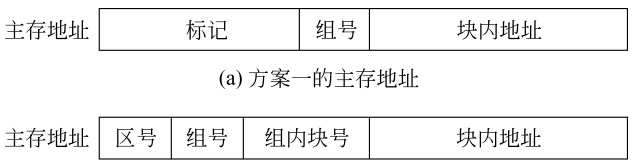


图 3-9　两种组相联映射方式实现方案的主存地址

图 3-9(a)是方案一的主存地址，分为 3 个字段。图 3-9(b)是方案二的主存地址，分为 4 个字段，其中组内块号字段指出组相联映射中的一个 Cache 组中块的数量，也就是组相联映射中的路数。相比之下，方案一的实现比较简单，用得更多。

16. 映射方式关联度分析

对于一个主存块来说，3 种映射方式下所对应的 Cache 行的个数不同。直接映射是唯一映射，每个主存块只有一个固定的 Cache 行与之对应；全相联映射是任意映射，每个 Cache 行都可对应；N 路组相联映射有 N 个 Cache 行对应。这种特征可以用关联度来度量。关联度是指一个主存块映射到 Cache 中可能存放的位置个数。直接映射的关联度最低，为 1；全相联映射的关联度最高，为 Cache 总行数；N 路组相联映射的关联度居中，为 N。

当 Cache 大小和主存块大小一定时，关联度和命中率、命中时间、标记所占额外开销有

如下关系：

(1) 关联度越低，命中率越低。因此，直接映射的命中率最低，全相联映射的命中率最高。

(2) 关联度越低，判断命中的开销越小，命中时间越短。因此，直接映射的命中时间最短，全相联映射的命中时间最长。

(3) 关联度越低，标记所占额外空间开销越少。因此，直接映射额外空间开销最少，全相联映射额外空间开销最大。例如，假定主存地址 32 位，按字节编址，主存块大小为 16 字节，Cache 最多能存放 4K 个主存块数据。则有：

- 当关联度为 1(直接映射)，每组 1 行，共 4K 组，标记占 32－4－12＝16 位，总位数为 4K×16＝64K 位。
- 当关联度为 2(2 路组相联)，每组 2 行，共 2K 组，标记占 32－4－11＝17 位，总位数为 4K×17＝68K 位。
- 当关联度为 4(4 路组相联)，每组 4 行，共 1K 组，标记占 32－4－10＝18 位，总位数为 4K×18＝72K 位。
- 全相联：整个为 1 组，每组 4K 行，标记占 32－4＝28 位，总位数为 4K×28＝112K 位。

17. 数据 Cache 的实现

在许多系统中，存储层次结构中有多级存储器都是采用 Cache 来实现的。最常见的情况是，将第 1 级 Cache(它最接近处理器)作为独立指令 Cache 和数据 Cache 来实现，而将其他各级 Cache 作为统一 Cache 来实现。

数据 Cache 通常包含一个标记阵列和一个数据阵列，还有一个命中/缺失逻辑。标记阵列包含了 Cache 中所含数据的地址，而数据阵列中包含的是数据本身。将 Cache 分成独立的标记阵列和数据阵列，可以减少 Cache 的访问时间，因为标记阵列通常要比数据阵列包含较少的位数，所以访问它要比访问数据阵列或者数据阵列和标记阵列的某种组合快得多。

一般来说，标记阵列被组织成二维结构，该结构中所包括的每一行标记项都代表 Cache 中的每一组，其列数等于 Cache 的关联度。图 3-10 显示了一个 4 路组相联 Cache 的标记阵列结构。

图 3-10　4 路组相联 Cache 的标记阵列

每个标记项描述一个数据 Cache 块，标记项由标记字段、有效位以及脏位（仅适用于写回式 Cache）3 部分组成，如图 3-11 所示。Cache 块的大小又称为 Cache 的行长。

图 3-11 标记项

标记阵列需要占用的存储空间是由以下因素决定的：Cache 中的块数、每个标记项需要的标记字段位数、有效位和脏位。

Cache 中数据阵列的结构类似于标记阵列的结构。数据阵列也被组织成二维阵列，其中的每一行代表该 Cache 中的每一组，其列数等于该 Cache 的关联度（路数）。

命中/缺失逻辑负责将主存地址中的标记字段与该组中各个标记项的标记字段内容进行比较。当这些字段的各位相匹配，并且对该标记项的有效位进行设置以后，就会产生一次命中，如图 3-12 所示。

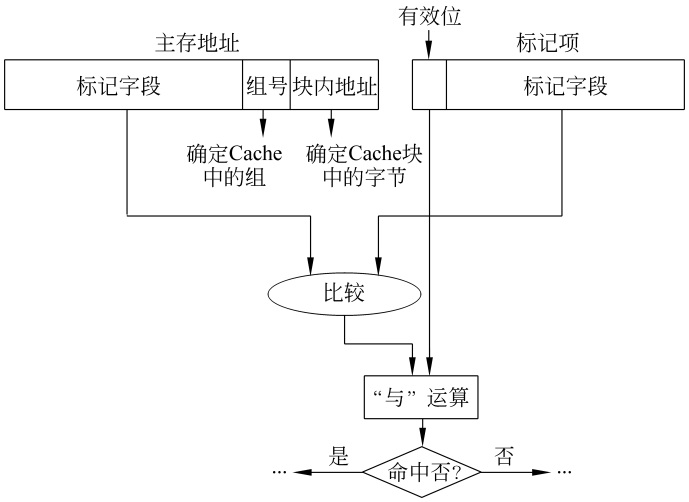


图 3-12 命中/缺失逻辑

18. Cache 的替换算法和更新策略

并不是 3 种映射方式都存在替换算法问题。直接映射时主存中的每一块只能被放置到 Cache 中唯一的指定位置，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。而采用全相联映射和组相联映射方式则不同，当从主存向 Cache 传送一个新块，而 Cache 中的空间已被占满时，就需要把原来存储的一块替换掉。常用的替换算法有下述 3 种。

1）随机算法

最简单的替换算法是随机算法。随机算法完全不管 Cache 过去、现在及将来的使用情况，简单地根据一个随机数，选择一块替换掉。

2）先进先出算法

先进先出（FIFO）算法的思想是：按调入 Cache 的先后决定淘汰的顺序，即在需要更新

时,将最先进入 Cache 的块作为被替换的块。这种方法要求为每块建立一个记录,记下它们进入 Cache 的先后次序。这种方法容易实现,而且系统开销小。其缺点是可能会把一些需要经常使用的程序块(如循环程序)也作为最早进入 Cache 的块替换掉。

3) 近期最少使用算法

近期最少使用(LRU)算法是把 CPU 近期最少使用的块作为被替换的块。这种替换方法需要随时记录 Cache 中各块的使用情况,以便确定哪个块是近期最少使用的块。LRU 算法更合理,但实现起来比较复杂,系统开销较大。通常需要对每一块设置一个称为"年龄计数器"的硬件或软件计数器,用于记录其被使用的情况。

Cache 更新策略又称为写策略,可分为直写法和写回法两种。

直写法是指 CPU 在执行写操作时,必须把数据同时写入 Cache 和主存。当某一块需要替换时,也不必把这一块再写入主存,新调入的块可以立即把这一块覆盖掉。这种方法实现简单,而且能随时保持主存数据的正确性,但可能增加多次不必要的主存写入,会降低存取速度。

写回法是指 CPU 在执行写操作时,被写数据只写入 Cache,不写入主存。仅当需要替换时,才把已经修改过的 Cache 块写入主存。在采用这种更新策略的 Cache 块表中,一般有一个标志位(称为脏位),当一块中的任何一个单元被修改时,该标志位被置 1。在需要替换这一块时,如果标志位为 1(表示该块数据已经"脏"了),则必须先把这一块写入主存之后,才能再调入新的块;如果标志位为 0(表示该块数据还是"干净"的),则这一块不必写入主存,只要用新调入的块覆盖这一块即可。这种方法操作速度快,但因主存中的字块未及时修改而有可能出错。

19. 虚实地址转换与页式虚拟存储器

虚拟存储器将主存或辅存的地址空间统一编址,用户可以自由编程,完全不必考虑程序在主存中是否装得下以及这些程序将来在主存中的实际存放位置。用户编程的地址称为虚地址或逻辑地址,实际的主存单元地址称为实地址或物理地址,虚地址空间要比实地址空间大得多。

在实际的物理存储层次上,程序和数据在操作系统管理下先被送入磁盘,然后操作系统将当前运行所需要的部分调入主存,供 CPU 使用,其余暂不运行部分留在磁盘中。

程序运行时,CPU 以虚地址来访问主存,由辅助硬件找出虚地址和实地址之间的对应关系。虚拟存储器有页式、段式和段页式之分,其中最常用的是页式虚拟存储器。对于页式虚拟存储器,主存空间和虚存空间都划分成若干个大小相等的页,主存的页称为实页,虚存的页称为虚页。虚地址到实地址的转换是由页表实现的。页表是一张存放在主存中的虚页号和实页号的对照表,用于记录程序的虚页调入主存时被安排在主存中的位置。如果某个虚地址指示的存储单元内容已在主存中,则通过地址转换,CPU 可直接访问主存的实际单元;如果不在主存中,则把包含这个字的一页调入主存后再由 CPU 访问。如果主存已满,则由替换算法从主存中将暂不运行的一页调回辅存,再从辅存中调入新的一页到主存。页式虚拟存储器的虚实地址的转换过程如图 3-13 所示。