第 3 章　存储系统

存储器的功能是存放指令和数据，存储器使计算机具有了记忆能力，进而具备自动进行操作的能力，因此，存储器在计算机中的地位很重要。存储器设计中，采用什么存储介质、怎样控制操作过程是一个基本问题；怎样进行结构组织，使存取速度较快、总成本较小是另一个重要问题。

本章主要讨论主存储器、高速缓冲存储器的基本组成与工作原理，以及虚拟存储器的基本原理及实现技术。

3.1　存储系统概述

3.1.1　存储器的分类

存储器有很多属性，从不同角度可以有不同的分类方法。例如，按电源掉电后信息的可保存性来分类，存储器有易失性、非易失性两种类型。下面仅讨论几种主要的分类方法。

1．按存储介质分类

存储介质指可用两个稳定的物态来表示二进制 0 和 1 的物质或元器件。目前主流的存储介质有半导体器件、磁性材料和光介质材料 3 种。

半导体存储器利用电平高低表示信息，有 TTL 型、MOS 型两种类型，后者被广泛使用。磁性材料存储器利用剩磁状态表示信息，有磁芯、磁表面两种类型，磁芯存储器已被淘汰，磁表面存储器又可分为磁盘、磁带两种。光介质存储器利用不同物态的反光性表示信息，常见的是光盘。除半导体存储器为易失性存储器外，其余两类都是非易失性存储器。

2．按存取方式分类

存取方式指存储器中信息的定位方法及操作方式。按照存取方式，存储器可分为随机存取、顺序存取、直接存取和只读 4 种类型。

1）随机存取存储器（Random Access Memory，RAM）

这类存储器的访问时间固定，通过译码器定位信息存放位置。由于访问时间与访问地址值无关，为了提高存储空间的利用率，RAM 的编址单位可以很小，如存储字长通常是 1B。半导体存储器就是一种 RAM。

2）顺序存取存储器（Sequential Access Memory，SAM）

这类存储器的信息总是按顺序读出和存储，访问时间完全取决于信息存放位置与当

前位置的距离。为了提高存取效率，每次访问的数据量都很大。磁带就是一种 SAM，它的特点是存储容量大、存取速度慢，常用作海量数据的脱机备份存储器。

3）直接存取存储器（Direct Access Memory，DAM）

这类存储器的存取方式和存取性能都介于 SAM 和 RAM 之间。访问时，先用 RAM 方式进行信息所在区域的寻址，再用 SAM 方式进行信息的存取。为了提高存取效率及存储空间利用率，DAM 的编址单位通常是大小固定的记录块。磁盘是一种典型的 DAM，记录块称为扇区，扇区大小通常为 512B，单个盘片的磁盘地址由磁道号及扇区号两级地址组成。

4）只读存储器（Read Only Memory，ROM）

这类存储器的存取方式是只读不写，信息寻址方法取决于内部结构是 RAM 还是 DAM，而 RAM、SAM、DAM 属于读写存储器（RWM）。随着用户需求的变化，目前 ROM 也允许修改信息，只是 ROM 具有非易失性，这是 ROM 和 RWM 的最大区别。

3．按应用功能分类

按照存储器在计算机中的应用功能，存储器可分为主存储器、辅助存储器、高速缓冲存储器和控制存储器 4 种类型。

1）主存储器（Main Memory）

主存用来存放程序运行时的代码和数据，是 CPU 唯一按地址访问的存储器。因而，主存具有较快的存取速度，通常由 MOS 型半导体动态 RAM 构成。由于半导体存储器的价格较高，因此，主存的容量不会很大。

2）辅助存储器（Secondary Memory）

辅存指计算机运行期间与主存直接交换信息的存储器，是主存的后援存储器，存放主存暂时不用的程序和数据。与主存相比，辅存的容量更大，速度不要求很快，通常由磁性材料或光介质的 DAM 构成。后备存储器主要用于脱机时存放信息，由于容量巨大，通常称为海量存储器。

3）高速缓冲存储器（Cache）

高速缓冲存储器位于 CPU 与主存之间，速度比主存更快，存放主存中最近使用的信息，是主存的快速缓冲器。Cache 通常由 MOS 型半导体静态 RAM 构成。

4）控制存储器（Control Storage）

控制存储器用来存放解释机器语言指令的微程序，位于 CPU 内部。由于其信息不需要修改，通常由 MOS 型半导体 ROM 构成。

3.1.2 存储器的主要技术指标

存储器的主要技术指标是存储容量、存取速度及传输速度。

1）存储容量

存储容量指存储器能够存放的二进制信息的总量，基本单位为位（bit）或字节（Byte），用 b 或 B 表示。常用单位有 KB、MB、GB、TB 等。

2）存取速度

存取速度可用访问时间、存储周期来表示。

存取时间（Access Time）又称访问时间，指存储器从启动一次存储器操作（读或写）到完成该操作所需的时间。存取时间有读出时间、写入时间两种。读出时间是存储器从收到有效地址开始，到数据送到引脚的全部时间；写入时间是存储器从收到有效地址开始，到数据写到所选地址的全部时间。

存取周期（Memory Cycle, 存储周期）指存储器连续进行两次存储器操作的最短间隔时间。通常，存取周期大于存取时间，因为存储器完成一次操作后，需要一定的时间恢复到就绪状态。

存取时间、存取周期的单位通常为 ns（纳秒）或 μs（微秒）。

3）传输速度

传输速度通常用存储器带宽表示。存储器带宽指存储器被连续访问时，可以提供数据的速率，即单位时间内存储器最多可以传送的信息量，单位通常是 Mbps（兆位/秒）、MB/s（兆字节/秒）等。对基本 RAM 而言，存储器带宽等于 W/T_M，其中，W 为 RAM 的数据引脚位数，T_M 为 RAM 的存取周期。

3.1.3　层次结构存储系统

存储器的技术指标有容量和速度，对计算机用户而言，还很关注价格这个经济指标。假设存储器的容量为 S，存储器的总价、每位均价分别为 C 和 c，则 $c=C/S$。一般来说，速度快的存储器，每位价格也会高。

1．层次结构的引入

计算机中，随着应用领域不断扩大，应用程序不断增大，同时执行的应用程序不断增多，所需的存储器容量越来越大。存储器的容量增大，势必会导致速度下降；而提高存储器的速度来应对，又会增加存储器的价格。

我们每个计算机用户对存储器的需求永远是：大容量、高速度和低价格。从实现角度看，存储器的大容量和高速度是一对矛盾；从成本角度看，高速度与低价格又是一对矛盾。因此，用一种存储器根本无法满足用户的这种贪婪需求。

那么，怎么才能满足用户的这种需求呢？计算机界的前辈们发现了程序访问的局部性原理（principle of locality），从中找到了解决方案。

1）程序访问的局部性原理

程序访问的局部性原理是指，程序运行时，指令和数据访问所呈现出的相对簇聚的现象，局部性有时间局部性、空间局部性两个方面。时间局部性指最近访问过的信息，在不久将会被再次访问。空间局部性指与最近访问信息相邻的信息，在不久将会被访问。

对于下列 C 语言代码：“int i, s, A[100]; for（i = 0; i < 100; i ++） s = s + A[i];”，可以发现，对 s 的访问具有时间局部性，对 A[i] 的访问具有空间局部性。

根据编程经验可知，程序经常在 10%的代码上花费 90%的执行时间，这就是时间局部性；而代码中大多数是顺序型语句（指令），程序一般按顺序执行，除非遇到跳转语句，这就是空间局部性。

2）层次结构的引入

程序访问的局部性原理给我们的启迪是：根据程序近期访问信息的情况，可以预测出将来要访问哪些信息。因此，存储器需求矛盾的一个解决方案就出炉了：将近期访问的信息存放在前方的小容量、快速存储器中，而将近期未访问的信息存放在后方的大容量、慢速存储器中，前后方存储器之间可以传递信息。这是做的好处是，快速存储器可以满足速度需求，小容量可以降低总价格，内部的传递信息使其成为一个整体。

上述方案中，存储器之间的这种前后方关系，就是层次结构。由此，为了满足用户对存储器的需求，冯·诺依曼计算机中单一的存储器，就变成了由多种存储器组成的、按层次结构组织的存储系统。

2．存储系统的层次结构

存储系统的层次结构如图 3.1 所示，其中 M_1、M_2、…、M_n 为采用不同技术实现的存储器，M_1 速度最快，M_n 速度最慢。从 CPU 角度看，所有存储器在逻辑上是一个整体。

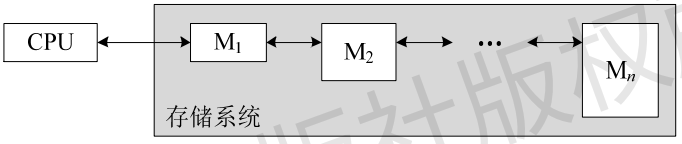


图 3.1　存储系统的层次结构

若存储系统的平均访问时间为 T_A，第 i 种存储器的容量、存储周期、每位价格分别为 S_i、T_i、c_i，则存储系统的每位平均价格 c 为：

$$c=(c_1S_1+c_2S_2+\cdots+c_nS_n)/(S_1+S_2+\cdots+S_n)$$

假如存储系统达到 $T_A\approx T_1$、$c\approx c_n$ 目标，S_n 就可任性配置，完全可以满足用户的需求。为了达到 $c\approx c_n$ 目标，各个存储器之间应满足如下条件：$S_1<<S_2<<\cdots<<S_n$，$T_1<<T_2<<\cdots<<T_n$。为了达到 $T_A\approx T_1$ 目标，各个存储器中的信息组织应符合程序访问局部性原理，并且 M_i 中存放的信息应是 M_{i+1} 中信息的副本。

由于各级存储器中的信息是包容关系，因此，各级存储器之间需要进行信息交换。又由于存储系统从外部看是一个整体，因此，存储器之间的信息交换对外部而言是透明的。

现代计算机中，主存是 CPU 可以直接按其地址访问的存储器，因此，存储系统以主存为中心。存储系统通常由高速缓冲存储器（Cache）、主存、辅存三种存储器组成。可分成 Cache−主存、主存−辅存两个存储层次，如图 3.2 所示。

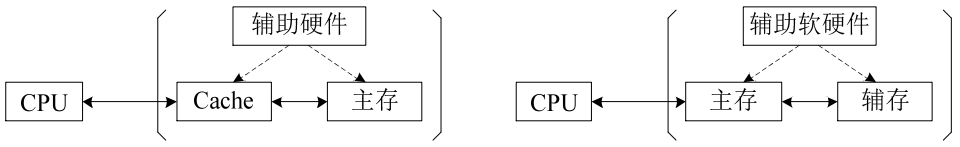


图 3.2　现代计算机的两个存储层次

Cache-主存层次主要解决主存的访问速度问题。借助辅助硬件，Cache 与主存构成一个整体。根据程序访问的局部性，将 CPU 近期访问信息存放在 Cache 中，其余信息存放在主存中，有需要时自动实现信息交换。由于信息交换管理由硬件自动完成，因此，Cache 对所有程序员是透明的。

主存-辅存层次主要解决主存的存储容量问题，借助辅助软件和少量硬件，构成一个整体。根据程序访问的局部性，将近期所用信息存放在主存中，其余信息存放在辅存中，有需要时再将信息从辅存调入到主存。由于有软件参与，层次管理对程序员并不透明。早期的层次管理由应用程序员负责，CPU 用主存地址访问程序和数据；现在的层次管理由操作系统负责，CPU 用程序地址进行存储器访问。

3．存储系统的工作过程

根据存储程序工作方式的要求，程序被调入主存后方可执行，程序执行时，CPU 是按主存地址访问存储系统的。

CPU 按主存地址访问 Cache 时，Cache-主存层次的辅助硬件控制整个访问过程，若访问信息在 Cache 中，Cache 立即完成访问；否则，从主存中调入访问信息后，再继续未完成的访问。由于存在程序访问的局部性，信息在 Cache 中找到的概率很高，平均访问速度接近于 Cache 速度。相关技术将在 3.4 节高速缓冲存储器中介绍。

程序总是存放在辅存中的，主存怎么分配给程序使用，目前由操作系统及相关硬件共同完成。程序执行时，不需要全部装入主存，有需要时自动在主存-辅存之间交换信息，并能够自动将程序地址转换成主存地址。这样，CPU 就可以按程序地址访问存储器，这个存储器的空间不再受主存容量限制了，其访问速度接近于主存速度。相关技术将在 3.5 节虚拟存储器中介绍。

可见，Cache-主存-辅存三级存储系统，可以达到容量为辅存容量、速度接近于 Cache 速度、价格略超过辅存价格的目标。

3.2　半导体存储技术

按照信息存储方法，半导体存储器可分为静态 RAM 和动态 RAM 两种，静态 RAM（Static RAM, SRAM）利用触发器电路状态存储信息，动态 RAM（Dynamic RAM，DRAM）利用 MOS 电容是否带电荷存储信息。半导体存储器中，RAM 属于易失性存储器，而具有非易失性的存储器只有 ROM。

3.2.1　静态存储器

1．SRAM 的存储元

存储器中，只能存放一位二进制信息的电路被称为存储元。

SRAM 存储元用触发器存储信息，由 6 个 MOS 管组成，如图 3.3 所示。其中，T_1、T_2 构成触发器，用来存储信息；T_3、T_4 是触发器的负载管；T_5、T_6 是门控管，用来控制是否对存储元进行操作。

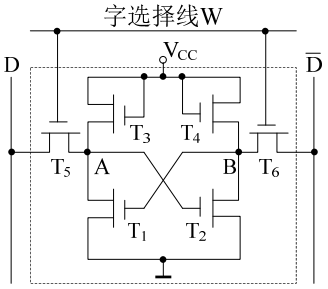


图3.3 6管SRAM存储元电路

触发器存储信息的基本原理是，若 T_1 截止，则 A 为高电平→T_2 导通→B 为低电平，B 为低电平更加使 T_1 截止，从而到达一个稳定状态。反之，若 T_2 截止，则 T_1 导通，到达另一个稳定状态。假定 T_2 导通表示所存的是信息 1，那么，T_1 导通存放的就是信息 0。

现在，我们来分析一下 SRAM 存储元的写入、读出和保持信息的原理。需要操作时，在字选择线 W 上加高电平，使 T_5、T_6 导通，从而可以对存储元进行读出或写入操作；否则，在 W 上加低电平时，使 T_5、T_6 截止，存储元进入保持状态。

对写操作而言，若要写 0，在 D 上加低电平、$\overline{D}$ 上加高电平，使 T_1 导通、T_2 截止，到达信息 0 状态。若要写 1，在 D 上加高电平、$\overline{D}$ 上加低电平，使 T_1 截止、T_2 导通，达到信息 1 状态。

对读操作而言，若所存信息为 0，则 D 为低电平、$\overline{D}$ 为高电平，反之，则 D 为高电平、$\overline{D}$ 为低电平；通过 D 可以读出所存信息。由于 D 和 $\overline{D}$ 流出电流不会改变 T_1、T_2 的状态，因此，读操作不会破坏所存信息状态，故 SRAM 的读操作为非破坏性读操作。

对信息保持而言，T_5、T_6 截止时，触发器与外部的连接断开，T_1、T_2 稳定地处于原有状态。只要不掉电，存储元中的信息可以永远保持不变，这种特性就是 SRAM 得名的原因。

2．SRAM 芯片的组成

最基本的 SRAM 是以芯片的形式出现的，SRAM 模块可以由多个 SRAM 芯片扩展而成。本节我们只讨论 SRAM 芯片的基本组成。

1）SRAM 芯片的基本组成

SRAM 芯片主要由存储矩阵、地址译码器、I/O 门、读写电路及控制电路组成，其结构如图 3.4 所示。为了便于理解，图中使用的是 2K×2 位 SRAM 芯片的引脚参数，即芯片有 2K 个存储单元，每个存储单元长度为 2 位。

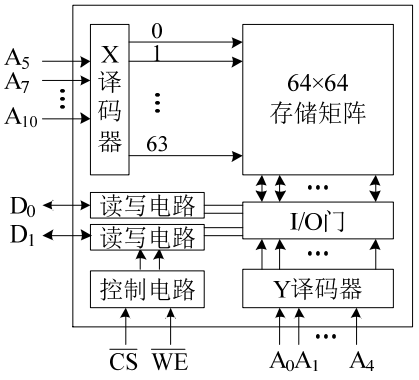


图 3.4 SRAM 芯片的内部结构

图中，A_{10}~A_0 为地址引脚，D_1~D_0 为数据引脚，$\overline{CS}$ 为片选引脚（低电平有效），$\overline{WE}$ 为读写引脚（低电平为写、高电平为读）。

（1）存储矩阵。存储矩阵由所有存储单元组成，存储单元的排列方式有一维和二维两种形式。由于信号延迟与线路长度成正比，存储矩阵通常都组织成二维的正方形，以便使存储元之间的连线最短、信号延迟最小，故称为存储矩阵，又称为存储阵列。

（2）地址译码器。功能是将每个地址信号转换为相应的信号线电平。存储单元的排列方式决定了其地址译码方式，相应地，地址译码有单译码、双译码两种方式。

单译码方式又称字选法，存储矩阵的每一行只有一个存储单元，存储单元的选择信号只需由一个译码器给出，译码器的输出信号线常称为字选择线。

双译码方式又称重合法，存储单元的选择信号由行、列两个方向的选择信号组合而成，对应的译码器常称为 X 译码器与 Y 译码器，其输出信号线常称为字选择线与位选择线，如图 3.5 所示。由于同一列的所有存储元，任何时候最多只有一个被选中，因此，同一列的存储元可共用 D 线、$\overline{D}$ 线及位选择线。

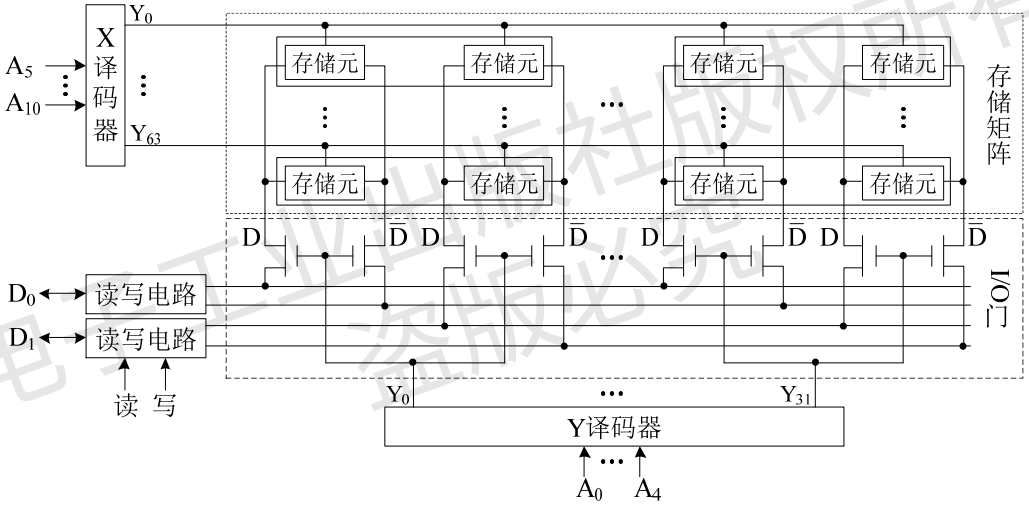


图 3.5　双地址译码结构

注意，Y 译码器的每个输出信号线（位选择线）选择的是一个存储单元，即存储单元中的所有存储元，图 3.5 中选择的是 2 个存储元。

（3）I/O 门。功能是从所有列的存储单元中，选择一个存储单元进行 I/O，如图 3.5 下部所示，又常称为列 I/O 电路。

（4）读写电路。读写电路由读出放大器、写放大器组成，用于控制被选中存储元的信息读出或写入，内部结构如图 3.6 所示。

注意，每个读写电路只能读写一位数据，芯片中读写电路的个数与存储单元长度相同。

（5）控制电路。功能是产生芯片内部的读/写操作控制信号。只有当芯片被选中（$\overline{CS}$ 有效）时，才可以对芯片进行读/写操作，电路组成如图 3.7 所示。为什么要设置 $\overline{CS}$ 引脚，稍后再解释。

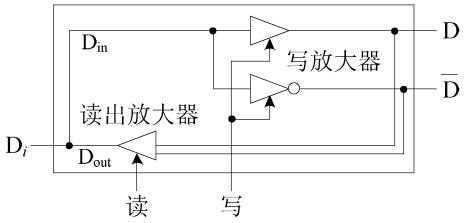


图 3.6　读写电路结构

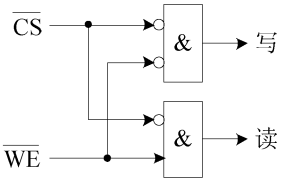


图 3.7　控制电路组成

2）SRAM 芯片的引脚组织

由图 3.4 可见，SRAM 芯片有地址引脚、数据引脚、控制引脚（片选及读写引脚）。

地址引脚的个数与芯片容量、存储单元长度都有关系。假设 SRAM 芯片的存储单元长度为 w 位，容量为 S 位，地址引脚为 n 个，则有 S＝存储单元个数×存储单元长度＝$2^n \times w$，$n = \log_2$(存储字数)$= \log_2(S/w)$。

数据引脚的个数只与存储单元长度有关。数据引脚有双向引脚、单向引脚两种组织方式。图 3.6 是双向引脚的组织方式，数据引脚的个数等于存储单元长度。只要将图 3.6 中的 D_{in}、D_{out} 直接变为引脚，就变成了单向引脚的组织方式，数据引脚的个数是存储单元长度的两倍。不特别指明时，默认采用双向引脚组织方式。

任何 RAM 芯片都会设置$\overline{CS}$引脚。这是因为，RAM 芯片的规格有限，只有允许用户利用已有芯片，进行存储模块设计，才能满足用户千变万化的需求。例如，一个 4K×2 位 SRAM 模块可以由 2 个 2K×2 位 SRAM 芯片组成，对该 SRAM 模块操作时，只会选中某一个芯片，因此，要求芯片具有$\overline{CS}$引脚，用来进行芯片选择。

例 3.1　若某 SRAM 芯片容量为 4Kb，数据引脚为 8 根，则地址引脚应为多少根？若将数据引脚改为 32 根，则地址引脚又应为多少根？

解：数据引脚为 8 根时，地址引脚数量＝$\log_2(4\times2^{10}\div8)$＝9 根；

数据引脚为 32 根时，地址引脚数量＝$\log_2(4\times2^{10}\div32)$＝7 根。

3．SRAM 芯片的读写时序

为了保证读/写操作的正确性，RAM 厂家对引脚信号的时序提出了严格要求。时序包括时长、次序两个方面，指信号的有效时刻和有效时长。

由 SRAM 的组成可知，SRAM 用$\overline{CS}$从无效→有效表示收到操作命令，此时，地址信号必须已经稳定，$\overline{CS}$从有效→无效表示结束操作。因此，对 SRAM 进行操作时，应先发送地址信号，后使$\overline{CS}$有效，并发送操作命令。

SRAM 读周期的时序如图 3.8 所示，$\overline{WE}$在操作期间为高电平（其余时间无所谓）。其中，t_{CO} 为开始操作到数据稳定输出所需的时间，t_{OTD} 为从$\overline{CS}$无效到数据线变为高阻所需的时间，t_A（访问时间）为从地址有效到数据有效的时间，t_{RC}（读周期）为连续两次读操作的间隔时间。由于需要将数据线恢复为高阻状态，有 $t_A < t_{RC}$。

SRAM 写周期的时序如图 3.9 所示，$\overline{WE}$在操作期间为低电平。其中，t_{AW} 为地址译码延迟，t_{DW} 为数据写入到存储单元所需的时间，t_{DH} 为$\overline{WE}$撤销后数据需要保持的时间，t_{WR} 为写恢复时间。t_{WC} 由 t_{AW}、t_W、t_{WR} 三部分组成，为了正确进行写操作，信号时序有如下要求：

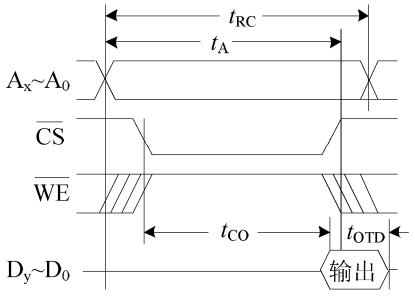


图 3.8　SRAM 读周期的时序

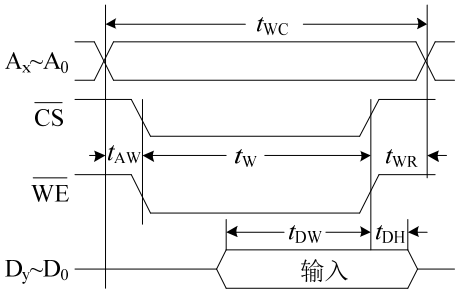


图 3.9　SRAM 写周期的时序

（1）地址信号有效后至少 t_{AW} 时间，$\overline{CS}$ 及 $\overline{WE}$ 才能有效，防止因地址来不及译码而出错。

（2）所写数据的发送时间应早于 $t_{AW}+t_W-t_{DW}$，且 $(t_{AW}+t_W-t_{DW})>t_{OTD}$。

对每个 RAM 芯片而言，读写周期时序中的参数都是固定的，与生产工艺有关。对 RAM 芯片操作时，所有信号的时长应比厂家规定更宽松些。

3.2.2　动态存储器

DRAM 用电容是否带电荷来存储信息，由于电容中的电荷总会缓慢泄漏的，必须在规定时间内，根据所存的信息重新补充电荷，才能长时间保持信息，因此，称为动态 RAM。相应地，用触发器保存信息的 RAM 称为静态 RAM。

1．DRAM 的存储元

DRAM 存储元的组成经历了四管 MOS、三管 MOS 及单管 MOS 等阶段，功耗、成本不断降低，集成度不断提高。下面，仅分析目前主流的单管 DRAM 存储元的基本组成。

单管 DRAM 存储元的基本电路如图 3.10 所示。电容 C_S 用于存储信息（有电荷表示 1、无电荷表示 0）；T_1 为门控管，用来控制是否对存储元进行操作。C_D 为预充电电容，每列只需设置一个，读操作需要用到它。

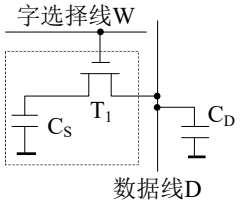


图3.10 单管DRAM存储元电路

对写操作而言，先在 W 线上加高电平，使 T_1 导通；再将所写数据加到 D 线上，C_S 随之充电或放电，信息被写入。

对信息保持而言，在 W 线上加低电平，使 T_1 截止，C_S 与外部无放电回路，其中的电荷可以暂存一定时间。需要定时进行刷新操作，以长久保持信息。

对读操作而言，先对 C_D 预充电，使 D 线为中等电位；再在 W 线上加高电位，使 T_1 导通，D 线上的电位将随 C_S 中所存的信息而产生变化，C_S 中信息为 0 时对 C_S 充电，D 线上电压下降，否则 C_S 放电，D 线上电压上升，放大该电位变化即可读出所存信息；最后还须进行再生操作。再生操作指用所读信息再次进行存储元写入，这是由于 C_S 中原有信息被破坏，必须立即恢复。

对刷新操作而言，操作步骤与读操作完全相同，因为读操作可以读出信息并重新写入。

2．DRAM 芯片的组成

由于 DRAM 存储元所用元件少，DRAM 芯片的集成度远高于 SRAM，等同面积的存储容量成倍增加，而地址引脚数量严重制约了芯片封装面积的减少。为此，DRAM 芯片都采用地址引脚复用技术，即地址由行地址、列地址组成，地址分两次送入 DRAM，地址引脚个数只有地址位数的一半。

为了实现地址分两次传输，DRAM 设置有行地址选通信号 $\overline{\text{RAS}}$、列地址选通信号 $\overline{\text{CAS}}$，来指明地址引脚上的当前地址类型。进一步地，使 $\overline{\text{RAS}}$ 在整个读写周期中全部有效，可以代替片选信号 $\overline{\text{CS}}$。

DRAM 芯片主要由存储矩阵、地址译码器、读写电路等组成，其结构如图 3.11 所示。与 SRAM 相比，DRAM 增加了地址锁存器、时序控制电路、再生电路。

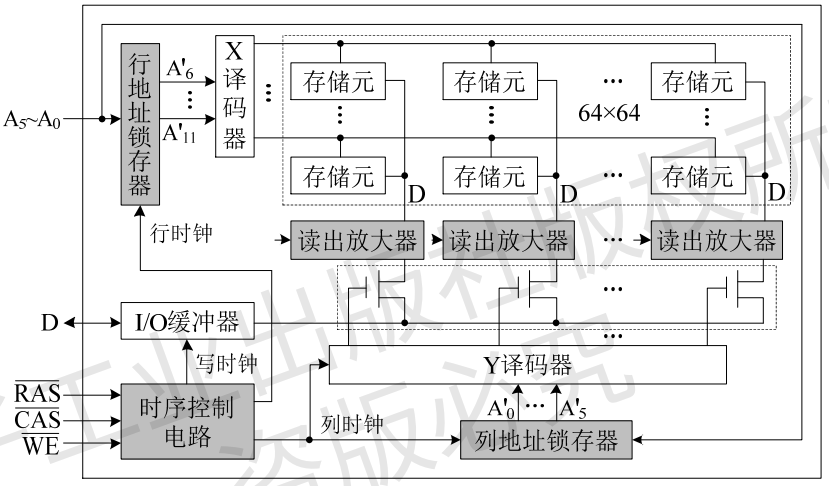


图 3.11　DRAM 芯片的内部结构

图中 DRAM 容量为 4K×1 位，地址引脚只有 6 根，是地址位数的一半。因此，地址需要分两次传送，芯片中设置地址锁存器来暂存行地址、列地址，通过 $\overline{\text{RAS}}$、$\overline{\text{CAS}}$ 产生的时钟信号，将地址写入到相应锁存器中。注意，本芯片没有 $\overline{\text{CS}}$ 引脚，是用 $\overline{\text{RAS}}$ 来代替的，这就要求使 $\overline{\text{RAS}}$ 在整个操作周期中全部有效。

时序控制电路中，行时钟、列时钟、写时钟信号是依次产生的，用来控制内部操作的时序。其中，行时钟由 $\overline{\text{RAS}}$ 控制产生，列时钟由 $\overline{\text{CAS}}$ 及行时钟控制产生，写时钟由 $\overline{\text{WE}}$ 及列时钟控制产生。可见，$\overline{\text{RAS}}$ 有效表示一个读写周期的开始。

图中的读出放大器，可以实现指定存储单元的读操作、写操作功能。读操作在输出所读信息的同时，自动进行再生操作。写操作与再生操作的原理相同，只是所写数据来自引脚。读出放大器的写操作功能应由写时钟控制（图中未标出信号名）。

3．DRAM 芯片的读写周期

由 DRAM 的组成可知，DRAM 的读写周期以 $\overline{\text{RAS}}$ 有效为开始标志，此时，行地址信号必须已经稳定，以 $\overline{\text{RAS}}$ 无效为结束标志。由于地址分两次传送，对 DRAM 进行操作时，应先分别发送行地址、列地址，然后发送操作命令。

DRAM 读周期的时序如图 3.12 所示。其中，t_{AH} 为地址锁存延迟，t_{CAC} 为开始操作到数据稳定输出时间，t_{DOH} 为 $\overline{RAS}$ 无效后数据线变为高阻的时间，$t_{CAS}-t_{CAC}$ 为再生操作时间。为了正确地进行读操作，信号时序有如下要求：

（1）$\overline{CAS}$ 后于 $\overline{RAS}$ 有效，且有 $t_{RCL}\geqslant t_{AH}$、$t_{CAS}\geqslant t_{AH}$，以避免地址来不及锁存而出错。

（2）$\overline{WE}$ 在 $\overline{CAS}$ 有效前无效（高电平），与 $\overline{RAS}$ 和 $\overline{CAS}$ 同时撤销，防止对存储单元进行误操作。

DRAM 写周期的时序如图 3.13 所示。其中，t_{DH} 为数据写入到存储单元所需时间。为了正确地进行写操作，信号时序有如下要求：

（1）$\overline{WE}$ 在 $\overline{CAS}$ 有效前有效（低电平），在 $\overline{CAS}$ 有效后至少保持 t_{DH} 时间，防止来不及写。

（2）写入数据应在 $\overline{CAS}$ 有效前发送到数据线上，且保持到 $\overline{WE}$ 无效之后。

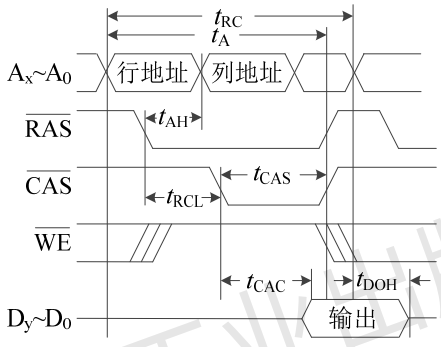


图 3.12　DRAM 读周期的时序

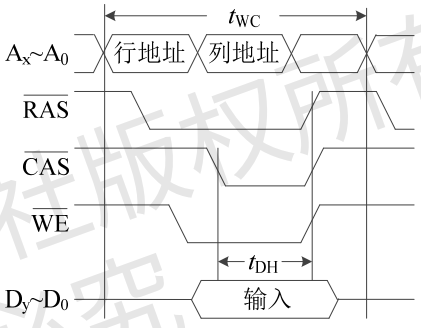


图 3.13　DRAM 写周期的时序

4．DRAM 芯片的刷新

由于使用电容保存信息，所存信息会因电荷泄漏而慢慢消失，必须周期性地对所有存储元，根据其所存信息重新进行电荷补充，这个过程称为刷新。

1）芯片的刷新操作

刷新可以采用读操作的方法实现，但所读信息不能输出到数据引脚上。由图 3.11 可见，刷新时，Y 译码器的输出必须全部无效，将存储元与数据引脚断开。如此一来，刷新时只要给出行地址即可，同一行的所有存储单元就可以同时刷新了，这种刷新方式称为行刷新。

行刷新方式是一种高效的刷新方法，例如图 3.11 中的刷新次数可从 4096 次减少为 64 次。采用行刷新方式，要求每一列都可独立地进行读写，因此，DRAM 中每一列都设置了读出放大器，如图 3.11 所示，就是为了支持行刷新的实现。

刷新操作中，只需要通过 $\overline{RAS}$ 提供行地址，使 $\overline{CAS}$ 一直保持无效，即可完成一行存储元的刷新，如图 3.14 所示。图中未画出 $\overline{WE}$ 信号，因为只要 $\overline{CAS}$ 一直无效，图 3.11 中的时序控制电路就一直不

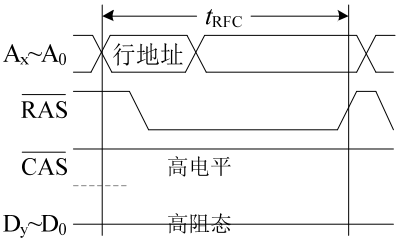


图3.14　DRAM刷新操作的时序

产生列时钟、写时钟信号，$\overline{WE}$ 的状态就与刷新操作无关。

那么，DRAM 如何区分操作是读还是行刷新呢？由图 3.11 可见，两者不同点在于 Y 译码器输出是否有效，只要使用列时钟信号来控制 Y 译码器是否输出，芯片内部就无须区分这两种操作。可见，刷新操作与读操作所需的时间是一样的。

2）芯片的刷新方式

刷新周期指每个存储元相邻两次刷新的最大间隔时间。刷新周期的时长由电容材料的特性决定，目前一般为 64ms。

采用行刷新方式后，DRAM 芯片的刷新实际上是在一个刷新周期内，对所有行逐行进行刷新操作。根据各行刷新时间的间隔不同，DRAM 芯片的刷新有集中、分散、异步三种方式。为了便于描述，假设刷新周期中，可以进行 n 次读写操作，需要进行 m 次刷新操作。

（1）集中式刷新。集中式刷新指各行的刷新操作集中在一段时间内连续进行。集中方式中，前一段时间用于读写操作，后一段时间只用于刷新操作，刷新时间分配如图 3.15 所示，其中 t_C 为 DRAM 的存取周期。刷新期间的读写操作只能等待，这段时间称为死时间（死区）。

（2）分散式刷新。分散式刷新指各行的刷新操作分散在各个存取周期中进行。分散方式中，存取周期的前一半用于读写操作，后一半用于刷新操作，刷新时间分配如图 3.16 所示。分散式刷新不存在死区，但存取周期的时长翻倍，存取效率比较低（很多 t_C 的后一半为空闲状态）。

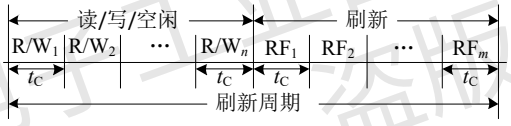


图 3.15　集中式刷新的时间分配

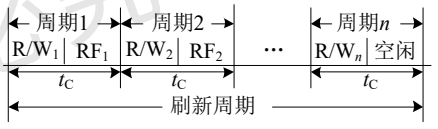


图 3.16　分散式刷新的时间分配

（3）异步式刷新。异步式刷新指各行的刷新操作均匀分散在整个刷新周期中进行。异步方式中，每过一定时间进行一次刷新操作，刷新周期结束时刚好刷新一遍，刷新时间分配如图 3.17 所示，其中，$d=n\div m$。异步式刷新的存取效率很好，死区仅为一个存取周期，可以忽略。

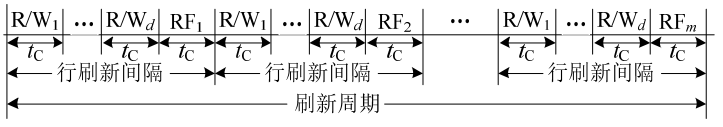


图 3.17　异步式刷新的时间分配

因此，DRAM 基本上都采用异步刷新方式。进一步地，为了减少刷新与读写的操作冲突，DRAM 可以包含两个以上的存储矩阵，使刷新和读写并行进行，只要两种操作不是针对同一个存储矩阵即可。

3）芯片刷新的实现

DRAM 芯片的刷新需要通过芯片以外的刷新电路实现，刷新电路通常做在 DRAM

控制器（DRAMC）中。DRAMC 管理所有 DRAM 芯片，一方面可以共享刷新电路，节约硬件成本；另一方面可以使不同芯片的读写与刷新并行，消除操作冲突。

现代计算机中，DRAM 只用作主存，因此，DRAMC 又称为主存控制器。DRAMC 是主存与外部的接口电路，主要有两大功能：一是管理对主存的操作，如 DRAM 芯片选择（产生芯片的 $\overline{\text{RAS}}$ 信号）、产生行/列地址等；二是实现所有 DRAM 芯片的刷新。

DRAMC 一般采用循环方式实现 DRAM 芯片的刷新，DRAMC 中刷新电路的逻辑框图如图 3.18 所示。其中，刷新定时器固化了所采用的刷新方式，会定时产生刷新命令；当前刷新操作的行地址由刷新地址计数器提供，计数器的初值为 DRAM 芯片的行数，在计算机启动时写入。

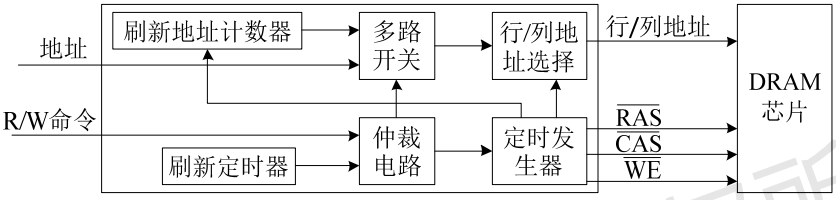


图 3.18　DRAMC 中刷新电路的逻辑框图

图中，仲裁电路负责处理操作冲突，确定 DRAM 芯片下面进行的是刷新操作，还是读写操作，或者无操作，仲裁结果决定了多路开关的选通方向。定时发生器根据当前的操作类型，产生相应的操作控制信号，控制 DRAM 完成操作；若当前是刷新操作，则控制刷新地址计数器计数一次。

5．DRAM 与 SRAM 的比较

DRAM 的存储元所需元件少（只有 1 个 MOS 管），地址引脚数量是地址位数的一半，相对于 SRAM，DRAM 的集成度高、封装尺寸小、功耗低（约为 SRAM 的 1/4）、成本小（约为 SRAM 的 1/100），DRAM 最大的缺点就是速度慢，约为 SRAM 的 1/10。

因此，DRAM 被用作大容量的主存，SRAM 则用作小容量的高速缓存（Cache）。

3.2.3　半导体只读存储器

相对于 RAM，ROM 的典型特征是具有非易失性。早期的 ROM 不可以修改信息，随着用户需求的改变，目前的 ROM 允许修改信息，通常，将具有非易失性的半导体存储器统称为 ROM。ROM 的类型有掩膜 ROM、PROM、EPROM、EEPROM、Flash 等。

由于存储元大多由半导体元件组成，ROM 都采用随机访问方式，因此，ROM 的结构与 SRAM 基本相同，由存储阵列、译码器、读写电路、控制电路等组成。不同类型的 ROM 主要区别是存储元的组成方式不同。下面我们只介绍存储元的逻辑实现。

1．掩膜 ROM（Mask ROM, MROM）

MROM 是一种不可编程的 ROM。用 MOS 管的有无来表示信息 1/0，如图 3.19 所示。当 W 为高电平时，存储元中有 MOS 管时输出低电平，无 MOS 管时输出高电平。

存储元中的 MOS 管是厂家在制造时，用掩膜工艺“写入”的，因此称为掩膜

ROM，用户无法修改。

2. 可编程 ROM（Programmable ROM，PROM）

PROM 是一种可以一次编程的 ROM，通常有 P-N 结击穿型及熔丝烧断型两种类型，它们的结构相同，信息表示方法不同。熔丝型 PROM 用熔丝的未断/已断来表示信息 1/0，如图 3.20 所示，出厂时全部为未断状态（信息 1）。

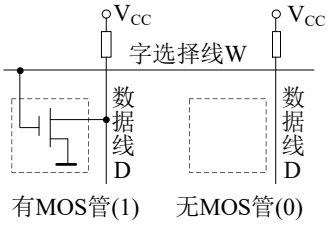


图 3.19　MROM 的存储元状态

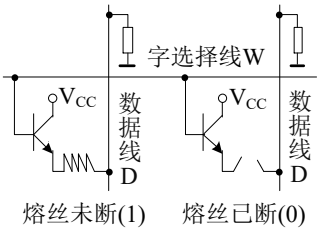


图 3.20　PROM 的存储元状态

读信息时，若熔丝未断，D 线为高电平；若熔丝已断，D 线为低电平。

写信息时，使字选择线 W 为高电平，若要写 1，则在 D 线上加地电压，熔丝保持未断状态；若要写 0，则在 D 线上加负电压，流过的大电流使熔丝呈断开状态。

3. 可擦除可编程 ROM（Erasable Programmable ROM，EPROM）

EPROM 是一种光擦除、可以多次编程的 ROM。FAMOS 型 EPROM 的存储元中采用了浮栅雪崩注入 MOS 管（Floating-gate Avalanche-Injection MOS），其结构如图 3.21 所示。

EPROM 存储元电路如图 3.22 所示，用 FAMOS 管的浮栅 G_f 上是/否带电荷来表示信息 1/0。

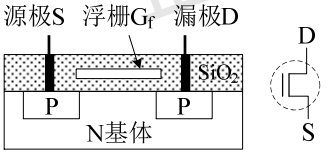


图 3.21　FAMOS 管的结构

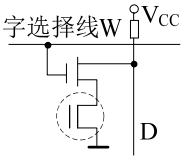


图 3.22　EPROM 存储元电路

读信息时，若浮栅上带电荷，FAMOS 因漏-源间形成导电沟道而导通，漏极 D 线因 FAMOS 导通为低电平，反向后输出 1；若浮栅上不带电荷，D 线因 FAMOS 截止为高电平，反向后输出 0。

若在漏极 D 加+25V、50ms 宽的脉冲电压，漏极-衬底间 PN 结产生雪崩击穿，高能量空穴使速度较快的电子穿过 SiO_2 层注入到浮栅中，当漏极 D 和源极 S 间的高电压去掉后，浮栅中的电子因无放电回路而得以保存下来，相当于写入了信息 1。

若要释放 FAMOS 浮栅上的电子，通过电路方式是不行的。如果用紫外线或 X 射线照射 FAMOS 管 20~30min，浮栅上的电子会获得光子能量穿过 SiO_2 层返回衬底，浮栅回到不带电荷的状态。由于光芒普照，芯片上全部存储元都被写入了信息 0。

习惯上，将一次对应多个存储单元的写操作被称为擦除，将一次对应一个存储单元

的写操作被称为写入。显然，FAMOS 型 EPROM 是一种光擦除的 EPROM，写入操作可以写 1 和写 0，而写 0 是假写，在 D 线上加地电压（0V），存储元的状态保持不变。

因此，EPROM 写信息时有两个步骤：先使用光擦除方法使全部存储元变为 0；再对所选的存储单元进行写入操作，根据所写信息为 1 或 0，在 D 线上相应地加正电压（＋25V）或地电压（0V）。

4. 电擦除可编程 ROM（Electrically Erasable Programmable ROM，EEPROM）

EEPROM（E^2PROM）是一种电擦除、可以多次编程的 ROM，其存储元中采用了浮栅隧道氧化层 MOS 管（Floating-gate Tunnel Oxide MOS，Flotox 管），Flotox 管的结构如图 3.23 所示。相对于 FAMOS 管，Flotox 管增加了一个控制栅 G_c，并在浮栅与漏区之间增加了一个氧化层（这个区域称为隧道区），使得写入和擦除的电压较小、速度较快。

EEPROM 存储元电路如图 3.24 所示，用 Flotox 管的浮栅 G_c 上是/否带电荷来表示信息 1/0。

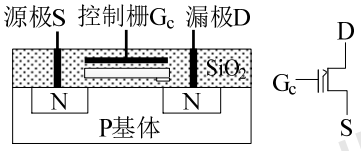


图 3.23　Flotox 管的结构

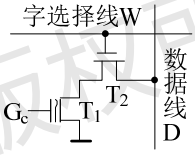


图 3.24　EEPROM 存储元电路

读信息时，在 G_c 上加＋3V 电压，若 Flotox 管浮栅上不带电荷，则 G_c 上的电压使漏-源间形成导电沟道，Flotox 管导通，D 线上读出低电平（信息 0）；若 Flotox 管浮栅上带电荷，则 G_c 上的电压不足以抵消所注入负电荷的影响（G_c 上加≥＋7V 电压才能使漏-源间形成导电沟道），Flotox 管截止，D 线上读出高电平（信息 1）。

若在 G_c 上加＋20V 左右、10ms 宽的脉冲电压，同时在 D 线上加地电压（＋0V），吸引漏区的电子注入浮栅，形成存储电荷，使 Flotox 管的开启电压上升到＋7V 以上，读信息时 G_c 上所加的＋3V 电压使 Flotox 截止，相当于写入了 1。若在 G_c 上加地电压（＋0V），同时 D 线上加＋20V 左右、10ms 宽的脉冲电压，浮栅上的存储电荷通过隧道区放电，使 Flotox 管的开启电压降为 0V 左右，读信息时 G_c 上所加的＋3V 电压使 Flotox 导通，相当于写入了 0。

为了节约成本，EEPROM 通常将同一行所有存储元的 G_c 连接在一起，因此，写 1 因对应多个存储单元属于擦除操作，写 0 则属于写入操作。

因此，EEPROM 写信息时也分为两步：先执行一个擦除操作，擦除同一行所有信息；再对同一行各存储单元执行写入操作，根据所写信息为 1 或 0，在 D 线上相应地加＋20V 或 0V 电压（存储元保持擦除后的信息 1）。

虽然 EEPROM 采用电压擦除信息，但由于写信息时需要较高的电压，且所需时间仍较长，因此，通常情况下，EEPROM 仍只工作在读出状态，做 ROM 使用。

5. 闪速存储器（Flash）

Flash 是新一代的电擦除、可编程 ROM，它既吸取了 EPROM 结构简单、编程可靠

的特点，又保留了 EEPROM 用隧道效应实现电擦除的特性。

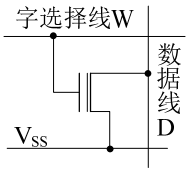


图3.25 Flash存储元电路

Flash 存储元采用了类似于 EEPROM 的叠栅 MOS 管，EEPROM 中浮栅与衬底间氧化层的厚度为 30～40nm，而 Flash 中氧化层的厚度只有 15nm，因此，这种叠栅 MOS 管的浮栅中，存储及释放电荷所需的电压较低（＋12V、10μs 的脉冲电压）。而正是这一特性，使 Flash 的存储元可以只由一个这种叠栅 MOS 管组成，如图 3.25 所示。

Flash 的读/写操作与 EEPROM 完全相同，只是 EEPROM 的擦除与 G_c 的连接有关，而 Flash 的擦除与源极 S 有关。早期，Flash 所有叠栅 MOS 管的源极 S 均连在一起，因而电擦除时所有存储元被同时擦除；现在，通常按字块（如 512B）将叠栅 MOS 管的源极 S 连在一起，因而电擦除时只同时擦除一个字块，大大提高了写信息的速度。

由于 Flash 的存储元由单管 MOS 组成，并且工作电压不高，因此，Flash 具有集成度高、容量大、速度快、成本低、功耗低等优点。正是由于这些优点，Flash 从其问世起便得到广泛的应用。

3.3 主存储器

3.3.1 主存储器的基本组成

我们知道，主存容量＝主存单元长度×主存单元个数，计算机结构设计时，确定了主存单元长度、主存地址空间这两个参数的值，软件和硬件都要遵守此规定。这个主存地址空间是一个线性空间，通常用位或个表示，如 32 位空间或 2^{32} 个地址，它限定了主存的最大可寻址空间。

由于主存是 CPU 唯一可以直接访问的存储器，因此，CPU 设计时，访问存储器（访存）时的地址位数、数据位数都需要受计算机结构规定的约束，与实际配置的主存容量无关。例如，IA32 约定，主存按字节编址，主存地址空间为 32 位，则 CPU 访问主存的地址为 32 位，每次可以访问一个字节的数据，而不管计算机配置了多大容量的主存。从 CPU 角度看，CPU 的可寻址空间的大小等于主存地址空间的大小。

为了保证访问速度，主存通常由 SRAM 或 DRAM 芯片组成。由于它们都是易失性存储器，无法解决计算机启动时，最先执行的程序和数据的存储问题，因此，主存空间由只读区域、读写区域组成，分别用 ROM 芯片、SRAM 或 DRAM 芯片实现。计算机启动完成后，系统通常会将 ROM 中的内容复制到 RAM 中，用来提高只读区域的访问速度。

因此，主存由 ROM 及 RAM 组成，如图 3.26 所示。对具体的计算机来说，主存单元长度、主存地址位数都是计算机结构所确定的，不可更改。主存容量可以改变，但必须小于等于主存最大容量。

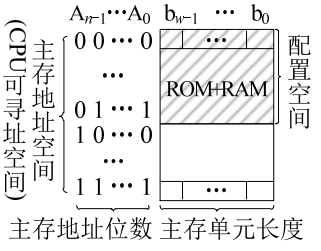


图3.26 主存的组成及参数

3.3.2　主存储器的逻辑设计

我们知道，主存容量＝存储字长×存储字数＝主存单元长度×主存单元个数，主存单元长度是计算机结构确定的固定值，主存单元个数是主存设计的具体目标值。由于 ROM、RAM 芯片的规格是有限的，主存往往需要由多个 ROM、SRAM 或 DRAM 芯片构成。

通常，将由多个存储器芯片构成的电路称为存储模块。为了区别于芯片，本教材将存储模块的对外接口称为信号线，而芯片称为引脚，如片选信号线 $\overline{CS}$ 指的是存储模块的信号。

可见，主存的逻辑设计实际上是一种存储器容量扩展，使用 ROM、SRAM 或 DRAM 芯片构成存储模块，存储模块的存储字长、存储字数分别等于主存单元长度、主存单元个数。

存储器容量扩展方法有位扩展、字扩展及字位扩展三种，位扩展可扩展存储字长，字扩展可扩展存储字数，字位扩展是位扩展和字扩展的简单组合。

1. 位扩展法

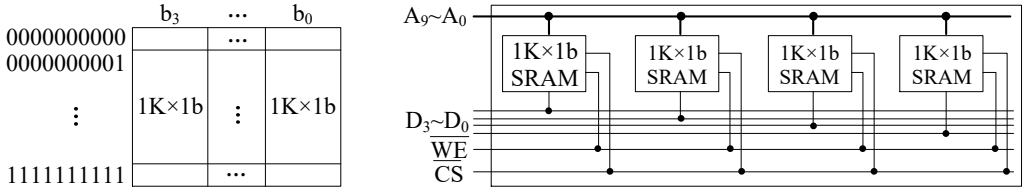
位扩展法又称位并联法，将多个存储器芯片并联起来增加存储字长，扩展后存储模块的字数与各芯片字数相同，存储字长为各芯片存储字长的和。

例 3.2　现有若干 1K×1 位 SRAM 芯片，要求组成一个 1K×4 位 SRAM 模块，请画出 SRAM 模块内部各 SRAM 芯片的引脚连接图。

解：1K×1 位（常记为 1K×1b）SRAM 芯片的地址、数据、片选、读写引脚分别记为 A_9~A_0、D、$\overline{CS}$、$\overline{WE}$。1K×4 位 SRAM 模块的数据信号线记为 D_3~D_0，其余信号线与 SRAM 芯片相同（且同名）。

由于两者的存储字数相同（1K÷1K＝1）、存储字长不同（4 位÷1 位＝4），扩展容量采用位扩展法，需 4 个 SRAM 芯片，SRAM 模块的存储空间组成如图 3.27(a) 所示。

由图 3.27(a) 可见，访问 SRAM 模块的某个存储单元时，需要同时访问所有 SRAM 芯片的这个存储单元，因此，各个 SRAM 芯片的 $\overline{CS}$、$\overline{WE}$、A_9~A_0 分别连接到 SRAM 模块的相应信号线上，而各个 SRAM 芯片的 D 分别连接到 SRAM 模块的 D_3~D_0 上，如图 3.27(b) 所示。



（a）存储空间组成　　（b）芯片引脚连接

图 3.27　用 1K×1 位 SRAM 芯片组成 1K×4 位 SRAM 模块

例 3.3　现有若干 1K×1 位 DRAM 芯片，要求组成一个 1K×4 位 DRAM 模块，请画出 DRAM 模块内部各 DRAM 芯片的引脚连接图。

解：1K×1 位 DRAM 芯片有[$\log_2(1K)$]÷2=5 根地址引脚，地址、数据、地址选通、读写引脚分别记为 A_4~A_0、D、$\overline{RAS}$及$\overline{CAS}$、$\overline{WE}$。1K×4 位 DRAM 模块的数据信号线数记为 D_3~D_0，其余信号线与 DRAM 芯片相同（且同名）。

类似于例 3.2，DRAM 模块采用位扩展法扩展容量，共需要 4 个 DRAM 芯片，存储空间组成与图 3.27(a) 相同。访问 DRAM 模块的某个存储单元时，需要访问所有 DRAM 芯片的同一个存储单元，故各个 DRAM 芯片的 A_4~A_0、$\overline{RAS}$及$\overline{CAS}$、$\overline{WE}$分别连接到 DRAM 模块的相应信号线上，而各个 DRAM 芯片的 D 线分别连接到 SRAM 模块的 D_3~D_0上。

从上面两个例题可以看出，位扩展只是扩展存储字长，存储字数不变，SRAM 与 DRAM 的位扩展方法相同，各个芯片连接时仅数据引脚连接不同，其余引脚连接相同。

2. 字扩展法

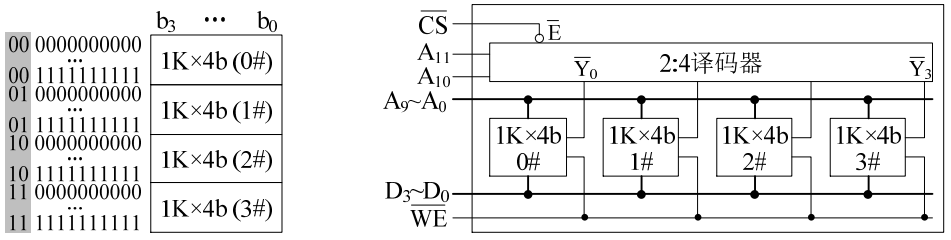
字扩展法又称地址串联法，用多个存储器芯片串联起来增加存储字数，扩展后存储器的字长与各个芯片字长相同，存储字数为各个芯片存储字数的和。

例 3.4 现有若干 1K×4 位 SRAM 芯片，要求组成一个 4K×4 位 SRAM 模块，请画出 SRAM 模块内部各 SRAM 芯片的引脚连接图。

解：1K×4 位 SRAM 芯片地址、数据、片选、读写引脚分别记为 A_9~A_0、D_3~D_0、$\overline{CS}$、$\overline{WE}$。1K×4 位 SRAM 模块的地址信号线记为 A_{11}~A_0，其余信号线与 SRAM 芯片相同（且同名）。

由于两者的存储字数不同（4K÷1K=4）、存储字长相同（4 位÷4 位=1），扩展容量采用字扩展法，需 4 个 SRAM 芯片，SRAM 模块的存储空间组成如图 3.28(a) 所示。

由图 3.28(a) 可见，访问 SRAM 模块中地址为 x 的存储单元时，只会访问某个 SRAM 芯片中地址与 x 的低位相同的存储单元，而 x 的高位用于选择所访问的 SRAM 芯片。因此，各个 SRAM 芯片的 A_9~A_0、D_3~D_0、$\overline{WE}$分别连接到 SRAM 模块的相应信号线上，如图 3.28(b) 所示，而 $A_{11}A_{10}$用于选择 SRAM 芯片（连接到芯片的$\overline{CS}$）。



（a）存储空间组成　　（b）芯片引脚连接

图 3.28　用 1K×4 位 SRAM 芯片组成 4K×4 位 SRAM 模块

由于 $A_{11}A_{10}$=00~11 访问的分别是 0#~3# SRAM 芯片，因此，0#~3#芯片的片选引脚的逻辑表达式应分别为$\overline{CS\cdot\overline{A_{11}}\cdot\overline{A_{10}}}$、$\overline{CS\cdot\overline{A_{11}}\cdot A_{10}}$、$\overline{CS\cdot A_{11}\cdot\overline{A_{10}}}$、$\overline{CS\cdot A_{11}\cdot A_{10}}$，其中的 CS 为对 SRAM 模块片选引脚$\overline{CS}$进行逻辑非操作的结果。

例 3.5 现有若干 1K×4 位 DRAM 芯片，要求组成一个 4K×4 位 DRAM 模块，请

画出 DRAM 模块内部各 DRAM 芯片的引脚连接图。

解：1K×4 位 DRAM 芯片有 5 根地址引脚，地址、数据、地址选通、读写引脚分别记为 A_4~A_0、D_3~D_0、$\overline{RAS}$ 及 $\overline{CAS}$、$\overline{WE}$。DRAM 模块的存储空间组成与图 3.28(a)相同。

一种扩展方法是锁存行列地址，DRAM 模块有 6 根地址信号线，其余信号线与 DRAM 芯片相同。由于 DRAM 模块的 $\overline{RAS}$ 有效时，用 6 位地址来选择 DRAM 芯片、提供 DRAM 芯片行地址不能两全（缺少 1 位），因此，DRAM 模块中，4 个芯片排列成 2×2 矩阵，增设行/列地址锁存器，用行地址和列地址的 A_5 信号来选择某一行的 2 个 DRAM 芯片，和某一列的 DRAM 芯片及 I/O。这种扩展方法只适宜于芯片内的存储体字扩展，由于每次有 2 个 DRAM 芯片参与操作，因此功耗较大。

另一种扩展方法是开放片选信号，DRAM 模块有 4 根 $\overline{RAS}$ 信号线，其余信号线与 DRAM 芯片相同。DRAM 模块内部，4 根 $\overline{RAS}$ 信号线用来选择芯片，其余信号线与芯片并联连接，如图 3.29 所示。这种扩展方法适宜于板级的 DRAM 芯片字扩展。

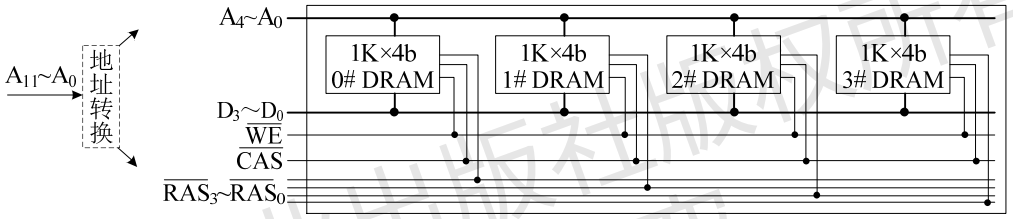


图 3.29　用 1K×4 位 DRAM 芯片组成 4K×4 位 DRAM 模块

参考图 3.28(a) 可以发现，对 DRAM 模块进行访问时，地址在哪个芯片范围内，就应使哪个芯片的 $\overline{RAS}$ 有效，以控制该 DRAM 芯片完成访问。而如何进行芯片选择，是由外部的地址转换电路实现的，如图 3.29 左部所示，该转换电路用地址高位（$A_{11}A_{10}$）选择某个 DRAM 芯片，将剩余地址（A_9~A_0）划分成行地址和列地址。

计算机中，DRAMC 是主存与外部的接口电路，它已经负责所有 DRAM 芯片的刷新，负责主存地址到 DRAM 地址的地址变换，那么，DRAM 字扩展的芯片选择也是它的职责。DRAMC 用地址高位选择 DRAM 芯片（使相应 $\overline{RAS}$ 信号有效），将剩余地址划分成行地址、列地址，用于对所选 DRAM 芯片的访问。

从上面两个例题可以看出，字扩展只是扩展存储字数，存储字长不变，SRAM 与 DRAM 的字扩展方法不同。

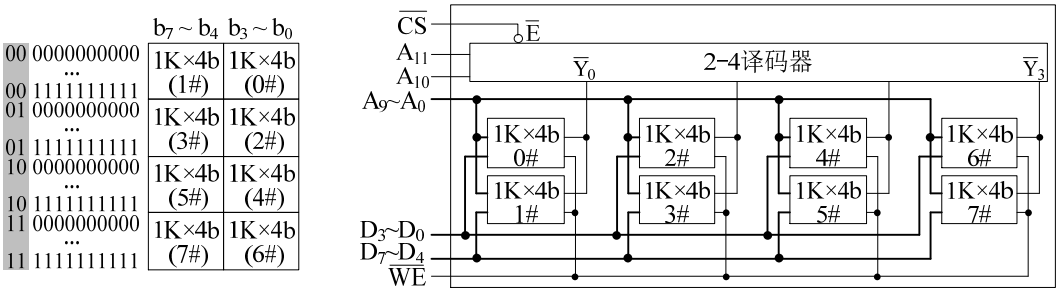
3. 字位扩展法

字位扩展法是位扩展和字扩展的组合，扩展后存储器的字数及字长都有所增加。

例 3.6　现有若干 1K×4 位 SRAM 芯片，要求组成一个 4K×8 位 SRAM 模块，请画出 SRAM 模块内部各 SRAM 芯片的引脚连接图。

解：SRAM 模块与 SRAM 芯片的存储字数和存储字长都不相同，容量扩展采用字位扩展法。需(8b÷4b)×(4K÷1K)＝8 个芯片，SRAM 模块的存储空间组成如图 3.30(a)所示。

由图 3.30(a) 可见，位扩展的两个 SRAM 芯片（记为 SRAM 芯片组）会同时被选中，$A_{11}A_{10}$ 用于选择 SRAM 芯片组。因此，SRAM 芯片组内部只有数据引脚连接不同，不同 SRAM 芯片组只有片选引脚连接不同，如图 3.30(b) 所示。



（a）存储空间组成　　（b）芯片引脚连接

图 3.30　用 1K×4 位 SRAM 芯片组成 4K×8 位 SRAM 模块

由于 $A_{11}A_{10}$ 用于选择 SRAM 芯片组，因此，各 SRAM 芯片组的片选引脚的逻辑表达式应分别为 $\overline{CS\cdot\overline{A_{11}}\cdot\overline{A_{10}}}$、$\overline{CS\cdot\overline{A_{11}}\cdot A_{10}}$、$\overline{CS\cdot A_{11}\cdot\overline{A_{10}}}$、$\overline{CS\cdot A_{11}\cdot A_{10}}$，用 2-4 译码器可以有效地实现各个 SRAM 芯片片选引脚的连接电路。

从上面这个例题可以看出，字位扩展只是字扩展、位扩展的组合。SRAM 与 DRAM 的字位扩展方法受限于它们的字扩展方法。

可以发现，任何一个存储器模块的设计，都可以利用已有的存储器芯片，使用某一种容量扩展方法进行。主存的逻辑设计只是其中的一个实例而已。

3.3.3　主存储器与 CPU 的连接

1. CPU 的存储器接口

现代计算机中，CPU 与主存及 I/O 设备都通过总线进行互连，CPU 对外部的访问需要遵循总线标准的相关协议。总线操作过程大体上分为地址/命令传送、数据传送两个步骤，1.3.3 节已有过简单介绍。

CPU 的存储器接口包含地址、数据、控制/状态引脚。下面以 Intel 8088 CPU 为例来说明。8088 CPU 中，主存按字节编址、地址空间为 20 位，数据引脚有 8 根，地址引脚有 20 根，CPU 并不限制主存由 SRAM 还是 DRAM 构成，也不限制主存的配置容量。

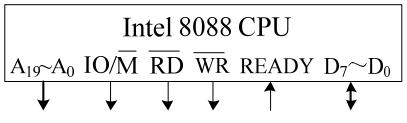


图3.31　Intel 8088 CPU的存储器接口

图 3.31 是 Intel 8088 CPU 的存储器接口，其中，IO/$\overline{M}$ 用于表示操作的目标部件类型（高电平为 I/O 设备、低电平为主存），$\overline{RD}$ 及 $\overline{WR}$ 用于表示操作类型（读/写/空闲），READY 用于表示操作的完成状态（高电平为已完成、低电平为未完成）。

可见，8088 CPU 的操作类型可用 IO/$\overline{M}$、$\overline{RD}$ 及 $\overline{WR}$ 来表示，$\overline{RD}\oplus\overline{WR}=0$ 表示无操作，$\overline{IO/\overline{M}}\cdot(\overline{RD}\oplus\overline{WR})=1$ 表示访存操作，IO/$\overline{M}\cdot(\overline{RD}\oplus\overline{WR})=1$ 表示 I/O 操作。从 Intel 80386 CPU 开始，操作类型改用 $\overline{ADS}$（地址数据选通）、M/$\overline{IO}$ 及 W/$\overline{R}$ 来

表示，$\overline{ADS}$在操作期间都有效，以便使主存或 I/O 设备可以立即进行响应，而不必等待操作命令。

2．主存与 CPU 的连接

主存可以由 SRAM 或 DRAM 组成，差别在于地址是一次传送还是两次传送。计算机组成不应限制主存的构成方式，为了不影响 I/O 操作的性能，通常，CPU 按地址一次传送的方式来设置地址引脚。

因此，SRAM 主存连接 CPU 较为简单，那么，DRAM 主存如何连接呢？前面已经讲过，DRAM 控制器（DRAMC）是 DRAM 芯片的外部接口，管理对 DRAM 芯片的操作，负责 DRAM 芯片的刷新。DRAMC 使 DRAM 主存与 SRAM 主存的外部接口完全相同，如图 3.32 所示。

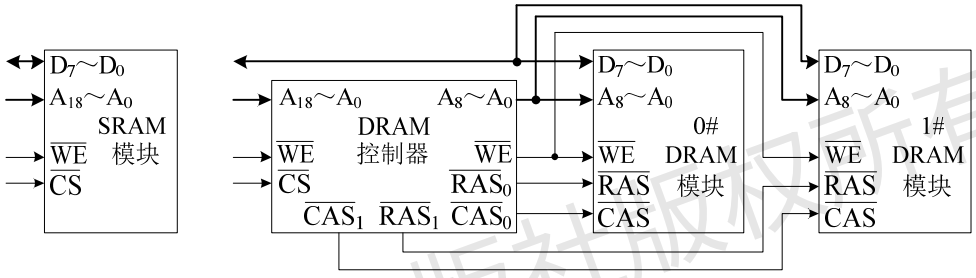


图 3.32　DRAM 主存的接口组织

由于 DRAM 主存与 SRAM 主存的接口相同，下面仅讨论 SRAM 主存与 CPU 的连接。主存与 CPU 的连接线有 3 组：数据线、地址线、控制线（读写线和片选线）。

1）数据线的连接

CPU 的数据线数决定了一次可访问数据的位数，由于 CPU 可以直接访问主存，主存数据线的位数与 CPU 数据线的位数必须相同。数据线连接时，主存的数据线与 CPU 的数据线一一连接，否则 CPU 无法正确地进行访问。

例如，主存与 8088 CPU（图 3.31）连接时，主存的数据线必须为 8 根，与 CPU 的 D_7～D_0 一一连接，如图 3.33(b) 所示。

注意，主存的数据线与其内部的 SRAM 或 DRAM 芯片数据引脚不是一个概念。

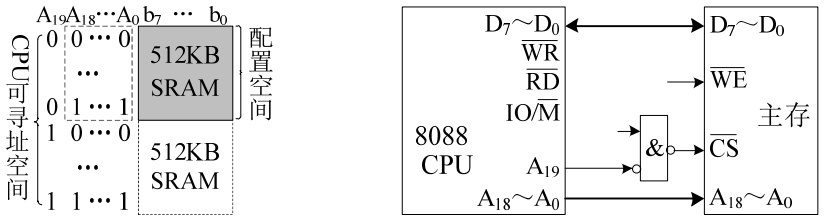
2）地址线的连接

CPU 的地址线数决定了主存的可寻址空间，而主存大小可以选配，因此，主存地址线的位数往往小于 CPU 地址线的位数。

为了便于硬件实现，通常将主存安排到 CPU 可寻址空间的低端。地址线连接时，主存的地址线与 CPU 地址线的低位信号线连接，CPU 地址线的高位信号线用来选择主存，即高位地址线的值为 0 时是对主存进行操作。

例如，若计算机的主存容量为 512KB，则主存有 19 根地址线 A_{18}～A_0，主存与 8088 CPU 连接时，主存空间通常映射到 1MB 空间的低端，如图 3.33(a) 所示。

地址线连接时，主存的 A_{18}~A_0 连接 CPU 的 A_{18}～A_0，CPU 的 A_{19} 用来选择主存，A_{19}＝0 时操作对象为主存，故 A_{19} 需要连接到主存的片选引脚 $\overline{CS}$，如图 3.33(b) 所示。



（a）地址空间映射　　（b）地址线及数据线的连接

图 3.33　主存与 CPU 的地址线及数据线连接

3）控制线的连接

SRAM 主存的控制线有 $\overline{WE}$ 及 $\overline{CS}$，CPU 可以对主存和 I/O 设备进行读/写操作，主存控制线与 CPU 连接的逻辑是：CPU 对主存操作时，使 $\overline{CS}$ 有效，否则使 $\overline{CS}$ 无效；CPU 的操作类型为写时，使 $\overline{WE}$ 有效，否则使 $\overline{WE}$ 无效。这个连接逻辑通常需要用电路来实现。

主存的 $\overline{WE}$ 线应与 CPU 的写操作控制线连接。由图 3.33(b) 可见，主存的 $\overline{WE}$ 线应与 CPU 的 $\overline{WR}$ 线连接。

主存的 $\overline{CS}$ 线应与 CPU 的控制线、地址线连接。上述连接逻辑中，CPU 对主存操作的逻辑可分解为有操作、操作目标为主存两个方面，操作目标为主存还包含“操作地址在主存的地址范围之内”这个条件。由图 3.31 可见，CPU 的 $\overline{RD} \oplus \overline{WR} = 1$ 表示有操作，$IO/\overline{M} = 0$ 表示操作目标为主存；由图 3.33(a) 可见，$A_{19}=0$ 时操作地址在主存的地址范围内，因此，主存 $\overline{CS}$ 线的连接逻辑为 $\overline{CS} = \overline{(\overline{RD} \oplus \overline{WR}) \cdot IO/\overline{M} \cdot \overline{A_{19}}}$。

注意，主存连接的部件不同，$\overline{CS}$ 的连接逻辑就不同。例如，主存与 80386 CPU 连接时，主存 $\overline{CS}$ 线的连接逻辑为 $\overline{CS} = \overline{ADS \cdot M/\overline{IO} \cdot \overline{A_{31}} \cdot \cdots \cdot \overline{A_{19}}}$。

例 3.7　ISA 总线有 24 根地址线（LA23~LA20、SA19~SA0），有 16 根数据线（SD15~SD0），对存储器、I/O 设备进行读、写操作的命令线为 $\overline{MEMR}$、$\overline{MEMW}$、$\overline{IOR}$、$\overline{IOW}$。某计算机中，主存的编址单位为 16 位，主存容量为 2MB，主存地址从零开始，画出主存连接到 ISA 总线时的连接图。

解：主存的数据线有 16 根（D_{15}~D_0），地址线有 $\log_2(2MB/16bit)=20$ 根（A_{19}~A_0），因此，主存的地址范围为 0～1M－1。

主存连接到 ISA 总线时，D_{15}~D_0 与 SD15~SD0 连接，A_{19}~A_0 与 SA19~SA0 连接。

ISA 总线中，$\overline{MEMR} \oplus \overline{MEMW} = 1$ 表示有主存操作，LA23~LA20＝0000 表示总线操作目标的地址范围为 0～1M－1，因此，主存与 ISA 总线的连接如图 3.34 所示。

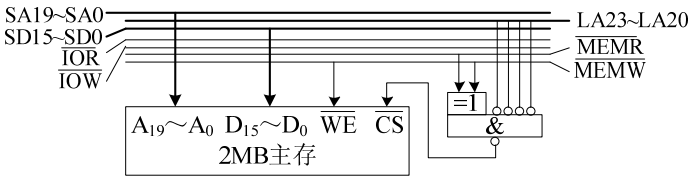


图 3.34　主存与 ISA 总线的连接

3.3.4　提高访存速度的技术

主存通常由 DRAM 存储器构成，与 CPU 所用的半导体工艺有所不同，两者在速度上的差异较大。主存的主要性能指标为带宽 B，$B=W/T_M$，其中 W 为数据宽度、T_M 为存取周期。

对于数据访问而言，由于不同数据类型的长度可能不同，一个数据可能占用多个连续的主存单元，一次访存应该可以访问多个连续的主存单元。由程序访问局部性可知，数据访问具有空间局部性，多次访存的地址很有可能是连续的。

因此，要提高访问主存的速度，可以从三个方面入手：第一，改进工艺及技术，提高存储器芯片本身的速度，来降低 T_M；第二，采用并行技术，在空间上和时间上并行访问存储器，来增加 W 或降低 $\overline{T_M}$；第三，采用层次结构，增设高速缓存 Cache，来降低 $\overline{T_M}$。

1．增强的 DRAM

目前的 DRAM 芯片，大都基于传统的 DRAM 芯片进行改进，提高其访问性能，下面介绍几种典型的 DRAM 芯片。

1）FPM DRAM（Fast Page Mode DRAM，快页模式 DRAM）

基本的 DRAM 每次都要用行地址、列地址进行访问，如图 3.11 所示。FPM DRAM 允许在选定某一行后，直接用列地址对该行的多个列进行操作。

对于同一行的连续访问，第一个访问通过 $\overline{\text{RAS}}$ / $\overline{\text{CAS}}$ 信号来控制，其余的访问通过 $\overline{\text{CAS}}$ 信号来控制（$\overline{\text{RAS}}$ 一直保持有效），$\overline{\text{CAS}}$ 在数据传送结束时才无效。FPM DRAM 中，页指同一行中所有存储单元的组合，快页模式指可以加速同一页中单元的访问。

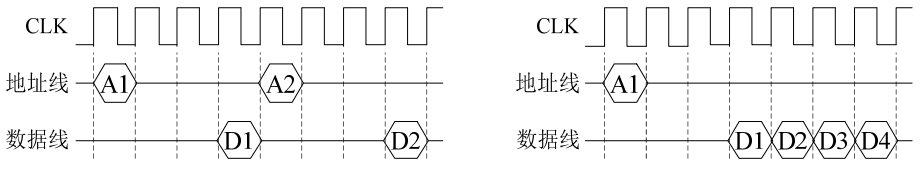
FPM DRAM 的改进版是 EDO DRAM（Extended Data Output DRAM，扩展数据输出 DRAM）。EDO DRAM 可以在传送数据的同时，启动下一个访问，即 $\overline{\text{CAS}}$ 可以在访问结束前就变为无效，因而，EDO DRAM 的访问速度更快。

2）SDRAM（Synchrobous DRAM，同步 DRAM）

传统 DRAM、FPM DRAM 都属于异步 DRAM，访问过程中，地址和控制信号要一直保持，CPU 及 DRAMC 在访问完成前都必须一直等待，降低了系统的性能。SDRAM 的所有操作都在时钟信号控制下进行，在确定的几个时钟周期后给出响应，CPU 及 DRAMC 在此期间无须等待。

SDRAM 的数据传送采用同步传输方式，所有动作都基于时钟信号进行，如图 3.35(a) 所示。图中，假设 SDRAM 的存取周期为 3 个时钟周期，第 4 个时钟周期用于数据传输，当两次访问的地址连续时，访存性能也不能提高。由于同步传输不需要异步传输的信号握手，因而速度更快。

SDRAM 还可以支持突发（burst, 猝发）传输模式，在第一个数据被访问后，可以连续传输多个数据，如图 3.35(b) 所示。通常，将连续传输的数据个数称为突发长度（Turst Length, BL），常规传输常看作 BL＝1 的突发传输。突发传输模式对一次需要访问多个连续存储单元的访存操作很有效，是 SDRAM 相对异步 DRAM 的最大优点。



（a）常规传输模式　　（b）突发传输模式

图 3.35　SDRAM 的两种传输模式

图 3.36 是一个 SDRAM 芯片的逻辑结构，SDRAM 都需设置时钟引脚 CLK，I/O 缓冲器通常也设置有锁存器，信号锁存及 I/O 都受 CLK 控制，内部操作与异步 DRAM 相同。

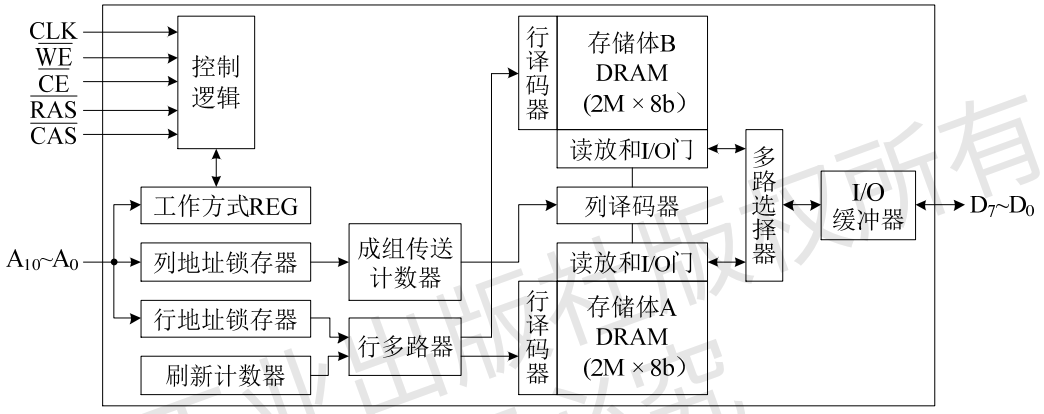


图 3.36　4M×8 位 SDRAM 的逻辑结构

SDRAM 由两个存储体组成，由于是字扩展，列译码器只需 1 个。采用多个存储体的原因有两个，一是工艺上要求存储体不宜太大，二是便于实现芯片自刷新，操作与刷新并行。

SDRAM 突发传输时，每传送一个数据，成组传送计数器就加 1。SDRAM 只支持几种 BL，当前 BL 存放在工作方式 REG 中，因此，每次操作都要先设置 BL，再进行数据访问，当上次操作的的 BL 与本次相同时，BL 设置就可以缺省。如何设置 BL 呢？由图可见，需要用特殊命令（如 $\overline{\text{RAS}}$、$\overline{\text{CAS}}$ 及 $\overline{\text{WE}}$ 同时有效）进行设置，因此，需要增设片选引脚 $\overline{\text{CE}}$。

3）DDR SDRAM（Double Data Rate SDRAM，双倍数据速率 SDRAM）

DDR SDRAM 是 SDRAM 的优化产品，采用了更先进的同步电路，能够在时钟脉冲的上升沿、下降沿分别进行数据传输，因此称为 DDR。

DDR SDRAM 的基本原理是，存储阵列中单元宽度为 I/O 宽度的两倍，在 I/O 缓冲器中增设预取缓冲区（1 个单元宽度），读操作时将缓冲区中数据拆分后分时送出，写操作时数据填满缓冲区后一起写入。可见，DDR SDRAM 的内部工作时钟频率与芯片 I/O 时钟的频率相同，而存取速度却是 SDRAM 的 2 倍，这得益于预取缓冲区大小是 I/O 引脚宽度的双倍，这就是所谓的 2 位预取（基于 I/O 宽度是 1 位的假设）。

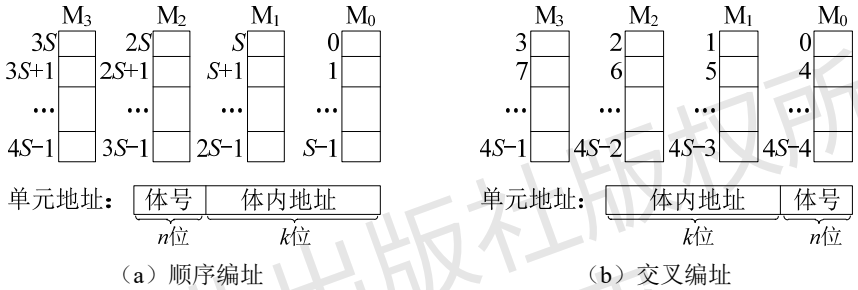
DDR2 SDRAM 与 DDR SDRAM 的原理基本相同，只是将存储阵列中单元宽度改为 I/O 引脚宽度的 4 倍（4 位预取），I/O 时钟频率是内部工作时钟频率的两倍，因此，存取速度是 SDRAM 的 4 倍。随着时钟频率的提升，时钟信号受温度、电阻等因素的影响越大，数据的精准传输不易控制，对内部同步电路的要求越来越高。

目前，市面上的内存条已经是 DDR3 SDRAM、DDR4 SDRAM 了，不难理解它们的工作原理，可见制造工艺更加精密。

2．多体交叉存储器

多体存储器指存储器由多个容量相同的存储体组成，每个存储体都有独立的存储阵列、读写电路。例如，图 3.36 的 SDRAM 就是一个双体存储器。

多体存储器的存储单元有顺序编址和交叉编址两种编址方式，如图 3.37 所示。其中，M_0~M_3 为 m 个存储体（$m=4$），每个体都有 S 个存储单元，则 $n=\log_2 m$，$k=\log_2 S$。



（a）顺序编址　　（b）交叉编址

图 3.37　多体存储器的编址方式

顺序编址又称连续编址，同一存储体的存储单元地址是连续的。交叉编址又称模 m 编址，同一存储体的存储单元地址都相差 m（m 为存储体数）。

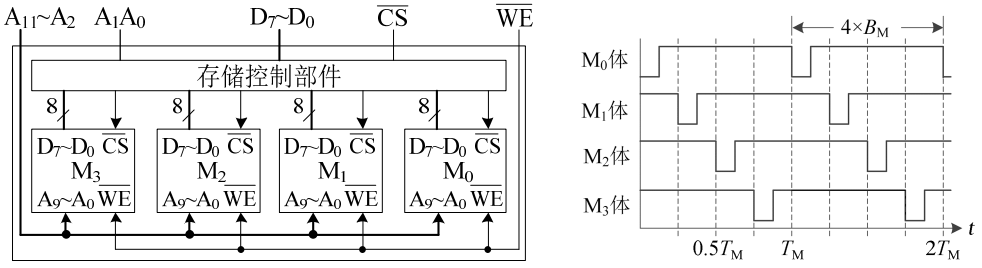
可以发现，两种编址方式的多体存储器，都是存储体按字扩展方式组成的存储模块，区别在于同一存储体的各个存储单元的地址是否连续。同时访问各个存储体的同一存储单元时，顺序编址方式访问的是多个不连续的存储单元，并行访问毫无意义，多体模式无法提高存取速度；交叉编址方式访问的是多个连续的存储单元，多体模式可以提高存取速度。

交叉编址的多体存储器是一种并行存储器，一次可以读写多个数据，访问方式有交叉访问、并行访问两种类型，通常，前者称为多体交叉存储器，后者称为多体并行存储器。为了便于理解，下面以 SRAM 为例来说明，方法同样适用于 DRAM。

1）交叉访问方式

交叉访问方式指轮流访问各个存储体，多个数据分时 I/O，多体交叉存储器结构如图 3.38(a) 所示。图中，各个存储体的数据端通过多路选择器连接到数据引脚 D_7~D_0，各个存储体被访问的都是同一行，地址引脚 A_1A_0 用于选择存储体。

交叉访问通常采用轮流启动方式（流水访问方式），即每隔 $1/m$ 个存取周期（T_M）启动一个存储体，经过 T_M 时间后，每隔 $1/m$ T_M 即可读出或写入一个数据，如图 3.38(b) 所示。由图 3.38(a) 可见，当 $\overline{CS}$ 引脚有效时，存储控制部件每隔 1/4 T_M 使一个存储体的 $\overline{CS}$ 有效，实现轮流启动；数据 I/O 时，每隔 1/4 T_M 改变多路选择器的控制信号。



（a）存储器结构　　（b）交叉访问方式

图 3.38　多体交叉存储器

可见，交叉访问方式的效果类似于 SDRAM 的突发传输或 DDR SDRAM 的存取，存储器带宽提高到 4 倍。

由图 3.38(a) 可见，交叉访问的多体存储器的外部引脚与单体存储器相同，与连接 CPU 的连接方法自然也相同。

2）并行访问方式

并行访问方式指可以同时访问各个存储体，多个数据同时 I/O，多体并行存储器结构如图 3.39 所示，图中，数据引脚宽度为存储字长的 4 倍，DM_3~DM_0 为数据掩码。

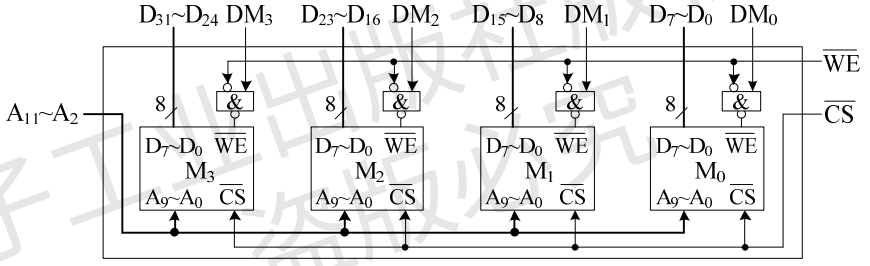


图 3.39　多体并行存储器

现代 CPU 为了提高访存速度，都采用并行访问方式，一次允许访问多个存储单元，因此，并行访问方式的多体存储器应该可以同时读/写一个、几个或 m 个存储字。为了便于实现，读操作每次都读出 m 个存储字；而写操作必须精确到存储字，每个存储体必须设置数据掩码（DM_3~DM_0），屏蔽不必要的写操作，如 DM_3~DM_0＝1100 时只写高 2 个字节。

注意，多体并行存储器与单体多字存储器是不同的，前者写的长度有多种，后者写的长度只有一种（长度为 m）。

CPU 采用并行访问方式时，其存储器接口需要进行相应的调整，数据线宽度等于机器字长，地址线的低位信号线用数据掩码信号线代替，如图 3.39 所示。

多体并行存储器与连接 CPU 时，连接方法与单体存储器类似，不同之处是数据线宽度有所增加，地址线没有低位信号线，增加了数据掩码线的连接。

实际应用中，主存通常集多种技术于一身，例如，Core 2 CPU 的主存按字节编址，DDR2 SDRAM 内存条宽度为 64 位（等于机器字长），既采用 SDRAM、预取等芯片技术减小存取周期 T_M，又采用多体并行存储器技术提高数据宽度 W。

3．双端口存储器

双端口存储器在一个存储器中提供两组独立的读写控制电路及 I/O 电路，因而可以支持对两个数据的同时访问，从而提高了存储器带宽。由于可以同时进行操作，因此，它和多体交叉存储器一样，属于一种并行存储器。

图 3.40 是一个 2K×8 位的双端口 SRAM 逻辑结构示意图，它提供了两个相互独立的操作端口，任意一个端口都可以独立地进行操作。与普通 SRAM 不同，双端口 SRAM 存储元有两个独立的字选择线、列数据线，连接到两个 I/O 电路，以实现同时对两个存储元操作。

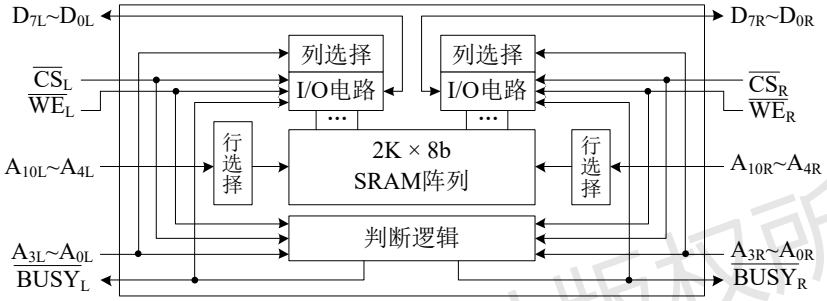


图 3.40　2K×8 位双端口 SRAM 的逻辑结构

有操作时，内部的判断逻辑电路进行冲突检测，若两个 $\overline{CS}$ 同时有效、两个行地址和列地址均相同时表明有冲突，由判断逻辑决定哪个端口优先操作（置该端口的 $\overline{BUSY}$ 为高），另一端口暂时等待（置该端口的 $\overline{BUSY}$ 为低）；否则，对双端口存储器的操作无冲突，可以同时进行。

双端口存储器常用于多个主设备访问共享数据的场合，例如，双端口寄存器组，同时支持两个操作，提高寄存器的访问速度；Cache 目录表，同时处理 CPU 侧及总线侧的事务，减小 Cache 的访问延迟。

3.4　高速缓冲存储器

3.3.4 节中提到，提高芯片本身速度及采用并行存储器都可以提高访存速度，由于主存的速度提高始终跟不上 CPU 的发展，因此，通过高速缓冲存储器（Cache）来提高访存速度，便成为一个很重要的手段。

3.4.1　Cache 的基本原理

从层次结构看，Cache 是主存的上一级存储器，主存通常由 DRAM 构成，Cache 应该由速度更快的 SRAM 构成，同时，Cache 中信息应该是主存中信息的副本（复制）。由于存储系统以主存为中心，CPU 按主存地址访问 Cache-主存层次。

1）Cache 的外部接口

Cache-主存层次中，辅助硬件的功能主要是负责 Cache-主存之间的信息交换、主存

地址→Cache 阵列地址的地址变换等工作。

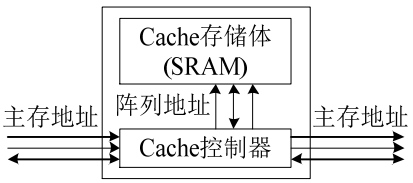


图3.41　Cache的外部接口

为了便于实现，通常将这个辅助硬件放在 Cache 中，因此，Cache 由 Cache 存储体、Cache 控制器组成，Cache 控制器实现的就是 Cache-主存层次中辅助硬件的功能，Cache 的外部接口如图 3.41 所示。由于 Cache 仅是主存的缓冲器，故按主存地址进行访问。

2）Cache 的性能指标

衡量 Cache 性能的指标主要有命中率及平均访问时间。

若访存的信息已经在 Cache 中，则称为命中（Hit），否则称为缺失（Miss）。访存操作在 Cache 中命中的概率称为命中率，等于命中次数与访存总次数的比值，常用 H 表示。

Cache 命中时，Cache 完成访存操作所用的全部时间 T_c 称为命中时间（Hit Time）；缺失时，Cache 需要先与主存交换信息，再完成访存操作，所用的时间由主存访问时间 T_m 和命中时间 T_c 组成，这个主存访问时间 T_m 称为缺失损失（Miss Penalty）或缺失开销。

平均访问时间 T_A 指 Cache-主存层次完成访存操作平均所用的时间，则

$$T_A=H\times T_c+(1-H)\times(T_c+T_m)=T_c+(1-H)\times T_m=T_{命中}+(1-H)\times T_{缺失}$$

例 3.8　假设 CPU 的时钟周期为 10ns，某程序执行时共访存 1000 次，其中 50 次未在 Cache 中命中，已知 Cache 存取一个信息的时间为 2 个时钟周期，每次缺失平均停顿 20 个时钟周期，求 Cache 执行该程序时的命中率，以及平均访问时间。

解：Cache 的命中率 $H=(1000-50)/1000=0.95$。

平均访问时间 $T_A=2\times10\text{ ns}+(1-0.95)\times20\times10\text{ ns}=30\text{ ns}$。

1．Cache 的存储空间管理

1）Cache-主存之间的信息交换单位

由平均访问时间 $T_A=T_c+(1-H)\times T_m$ 可以看出，要想提高 Cache 的性能，主要是提高命中率 H、减小缺失损失 T_m，而命中时间主要由 Cache 器件性能决定。

由于程序访问具有空间局部性，Cache 缺失从主存中调入（复制）信息时，若将多个相邻信息一起调入 Cache 中，则可以提高 Cache 的命中率 H；利用 SDRAM 的突发传输模式，多个信息一起调入的存取效率较高，为多个相邻信息一起调入提供了可行性。

因此，Cache-主存之间的信息交换单位为块（Block），块由多个相邻的存储字（主存字）组成，大小固定。块大小是由计算机结构确定的，通常为 8 个机器字长。

2）Cache 的存储空间管理

由于 CPU 用主存地址访问 Cache，因此，Cache 存储阵列的编址单位必须与主存一致，即按主存字（主存单元中信息）大小编址，但与主存的信息交换单位为块。

为了便于 Cache-主存之间按块交换信息，需要将主存空间、Cache 的数据空间划分成若干个大小为块的区域，主存中的区域称为主存块，Cache 中的区域称为缓存块。为

了进行层次管理，Cache 中的每一个缓存块都要对应有一些管理信息，通常将存放缓存块和相应管理信息的存储空间称为行（Line）或槽（Slot），如图 3.42 所示。可见，主存地址由主存块号、块内地址组成，Cache 地址由 Cache 行号、块内地址组成，块内地址又称为块内偏移地址，即从 0 开始的块内序号。

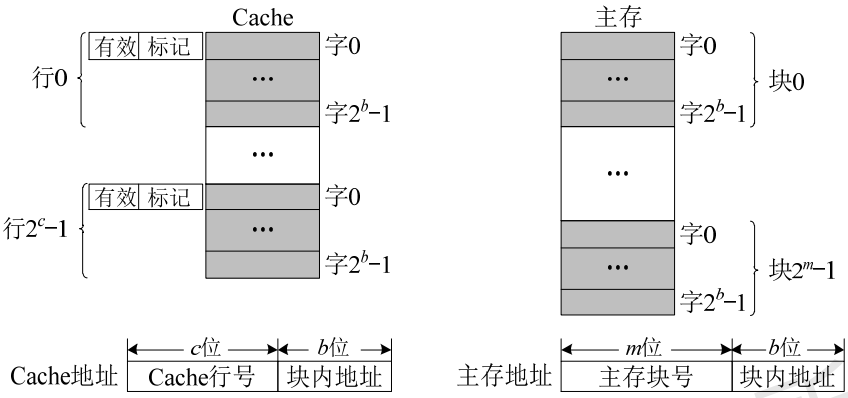


图 3.42　Cache 与主存空间划分

为了表示 Cache 行中的数据是否有效，每个行需要设置一个有效位（Valid bit）字段，常用 V 表示。有效位 V＝1 表示该行的数据有效，V＝0 表示该行空闲。若要清空某一行中的数据，可以通过 V←0 来实现。

为了表示 Cache 行中的数据来自哪个主存块，每个行需要设置一个标记（Tag）字段。这个标记通常是主存块号的一部分，这样 Cache 行与主存块就有了对应关系。通过查找各个 Cache 行的标记，就可以知道目标主存块在哪个 Cache 行中了。

因此，Cache 的存储空间可以组织成行的数组，每个 Cache 行中包含有效位、标记等管理信息，以及缓存块信息，如图 3.43 所示。所有行的管理信息常合称为目录表，所有行的块数据信息合称为数据区。

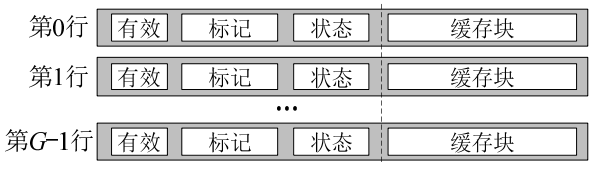


图 3.43　Cache 的存储空间组织

2．Cache 的工作流程

访问 Cache 时，首先需要将主存地址变换成 Cache 地址，而 Cache−主存之间按块交换信息，因此，地址变换进行的是行地址变换（获得当前行地址），块内地址是相同的。由于 Cache 中信息是主存中信息的副本，因此，两者需要保持一致性。

因此，Cache 完成访问的过程分三步：第一，将主存地址变换成 Cache 地址，即查找请求字所在的 Cache 行，缺失时需要先指定一个空闲行，再调入目标主存块；若没有空闲行，还需要先找出一个牺牲行并腾空其中内容。第二，读/写 Cache 存储阵列，完成访问要求。第三，在适当时候将所写数据写回主存。Cache 的工作流程如图 3.44 所示。

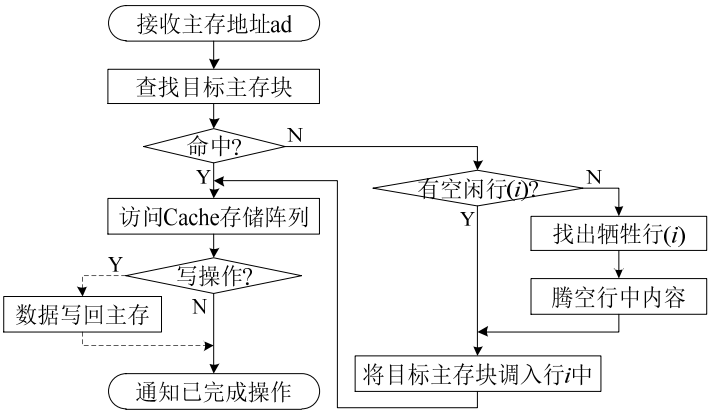


图 3.44　Cache 的工作流程

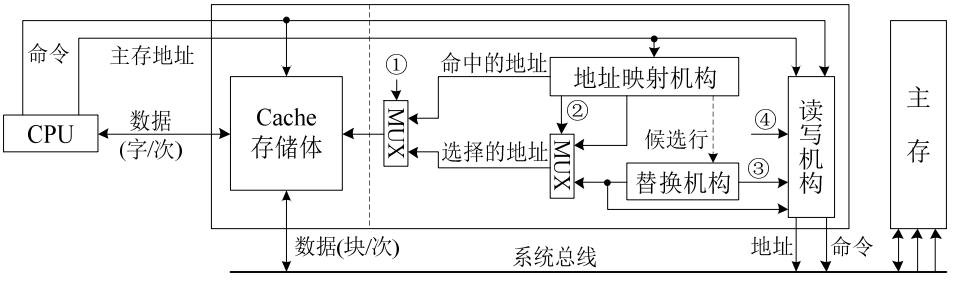
图 3.44 中，目标主存块指存储单元 ad 所在的主存块；查找主存块、查找空闲行的方法与映射规则有关；命中时已经完成了地址变换（获得了 Cache 当前行地址），访问 Cache 存储阵列只需用块内地址访问当前行中数据即可；找出牺牲行的方法与替换算法有关，数据写回主存的时机、腾空行中内容的方法都与写策略有关，图中的虚线表示数据不一定在此时写回主存。

映射规则指确定主存块可以放到哪些 Cache 行中的方法，映射规则决定了查找范围及查找方法。替换算法指在查找范围内找出牺牲行的方法，替换算法对 Cache 命中率有较大影响。写策略指数据写回主存的时机及方法，时机有立即写和稍后写两种，写策略对命中时间有较大影响，还决定了腾空牺牲行中内容的方法。

Cache 工作流程中，所有功能都在 Cache 控制器的控制下完成。由于 Cache 的目标是提高主存速度，因此，Cache 的全部工作都由硬件完成，对所有软件人员透明。

3．Cache 的基本结构

Cache 的工作流程表明，按主存地址完成访问包括地址变换、访问阵列、数据写回主存三个步骤，地址变换过程中，缺失时需要调入主存块，可能还需要替换主存块。由于 Cache 的所有功能都由硬件完成，因此，Cache 主要由存储体、地址映射机构、替换机构及读写机构组成，如图 3.45 所示。其中，虚线右边为 Cache 控制器，管理信息可以放在 Cache 存储体中。



注：①—是/否命中，②—有/无空闲行，③—是/否需要写回块，④—是/否需要调入块

图 3.45　Cache 的基本结构

地址映射机构的功能是判断 Cache 是否命中，命中时得到当前行地址；缺失时选择一个空闲行作为当前行，没有空闲行时由替换机构给出一个行。查找的范围由映射规则确定。

替换机构的功能是在查找范围内选择出一个行，并腾空其中的内容。查找的范围由映射规则确定，选择的方法由替换算法决定，腾空的方法由写策略决定。

读写机构的功能是从主存读出主存块，或将数据（块或字）写回主存。读出的时机为 Cache 缺失时，写入的时机由写策略决定。

基于图 3.45，再来看图 3.44 的 Cache 工作流程，很容易就能理解 Cache 的工作原理。下面，我们来讨论映射规则、替换算法、写策略的组织方法。

3.4.2　Cache 的地址映射

为了将主存数据放到 Cache 中，必须将主存地址空间映像到 Cache 地址空间，这个过程称为地址映射。地址空间映射的方法称为映射规则，映射规则确定了一个主存块可以放到哪些 Cache 行中，因此，映射规则又常称为映射函数。

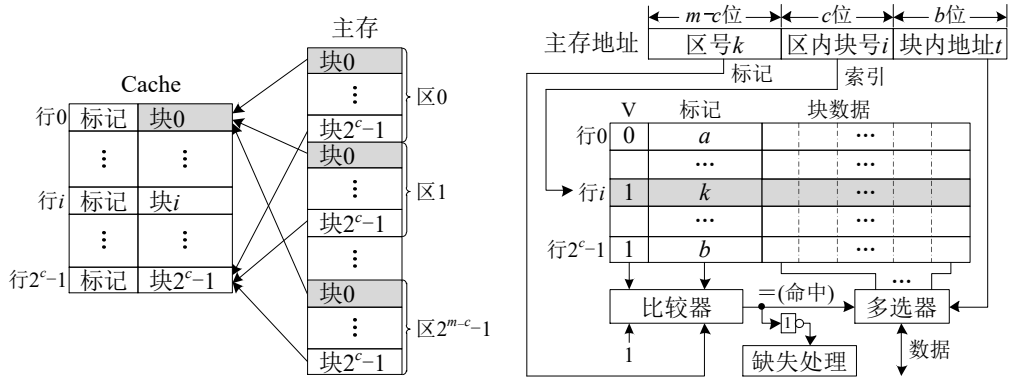
地址映射规则通常有直接映射、全相联映射、组相联映射三种。

通常，将可能存放某个主存块的那些 Cache 行称为候选行。不同映射规则的候选行的行数不同，主存块调入时的冲突概率不同，地址变换的速度及成本也不同。

1．直接映射（Direct mapping）

直接映射指一个主存块只能存放到一个 Cache 行中，候选行数只有 1 行。若主存块 i 可以存放到 Cache 行 j 中，则映射函数为：$j=i \bmod G$，其中 G 为 Cache 的行数。

假设 $G=2^c$，主存有 2^m 个块，每 2^c 个块称为一个区，则主存每个区的第 0 块仅映射到 Cache 第 0 行，每个区的第 1 块仅映射到 Cache 第 1 行，…，映射关系如图 3.46(a)所示。注意，区是本教材为便于描述而定义的概念，区的大小等于 Cache 容量。



（a）直接映射关系　　（b）直接映射的组织

图 3.46　直接映射方式

由图 3.46(a) 可见，主存地址可以划分成区号、区内块号、块内地址三个字段，其中，区号、区内块号合起来为主存块号。直接映射时，区内块号等于 Cache 行号，因

此，区内块号为 c 位，区号为 $m-c$ 位。由于区内块号可用作索引，因此，为了节省空间，Cache 行只要使用区号作为标记（Tag），即可表示行中缓存块来自哪个主存块。

直接映射的地址变换过程为：用主存地址的中间 c 位为索引，找到对应的 Cache 行；用主存地址的高 $m-c$ 位与该 Cache 行的标记进行比较，若相等且有效位为 1，则表示 Cache 命中，否则为 Cache 缺失，如图 3.46(b) 所示。

Cache 命中时，用主存地址的低 b 位（块内地址），对当前行缓存块中的存储字进行读/写，就可以完成图 3.44 中“访问 Cache 存储阵列”的功能；Cache 缺失时，进行腾空行中数据、调入目标块等缺失处理工作。注意，多选器应具有三态功能，命中时才读/写数据。

直接映射的候选行只有 1 行，只要这一行的有效位为 1，块调入时就产生冲突，因而冲突概率最高。地址变换只需要一个比较器，速度最快、成本最低。

例 3.9 假设计算机的主存按字节编址，CPU 的可寻址空间为 1M，Cache 数据区容量为 8KB，主存块大小为 16B，Cache 与主存之间采用直接映射方式。问：

（1）为了实现映射，主存地址应该如何划分？各字段长度分别为多少位？

（2）Cache 行的标记为多少位？

（3）若 CPU 访问主存的地址为 06454H 时，则可能命中的 Cache 行号是多少？命中行的标记的值是多少？

解：依题意，Cache 有 8KB/16B＝512 行，主存地址为 $\log_2(1M)=20$ 位。

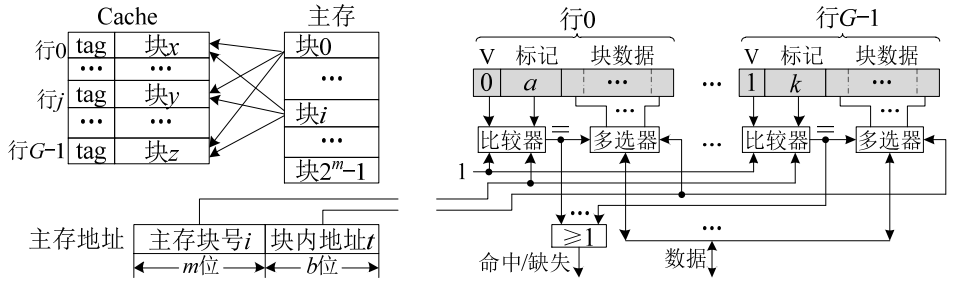
（1）主存地址可划分为区号、区内块号、块内地址，其中，块内地址为 $\log_2(16B/1B)=4$ 位，区内块号与 Cache 行号等长，为 $\log_2512=9$ 位，区号为 20－9－4＝7 位。

（2）Cache 行使用区号作为标记，标记字段为 7 位。

（3）主存地址 06454H＝0000 0110 0100 0101 0100B，高 7 位 0000 011B 为区号，中间 9 位 0 0100 0101B 为区内块号，命中时 Cache 行号等于区内块号，即 0 0100 0101B＝045H＝69；命中行的标记的值为 0000 011B＝03H＝3。

2．全相联映射（Fully associate mapping）

全相联映射指一个主存块可以存放到 Cache 任意一行中，候选行为 Cache 所有行。若主存块 i 可以存放到 Cache 行 j 中，则映射函数为随机函数，即 $j\in\{0,1,\cdots,G-1\}$，其中 G 为 Cache 的行数，映射关系如图 3.47(a) 所示。



（a）全相联映射关系　　（b）全相联映射的组织

图 3.47　全相联映射方式

由图 3.47(a) 可见，主存地址只需要划分成主存块号、块内地址两个字段。由于地址映射没有索引条件，Cache 行只能使用主存块号作为标记（Tag），来表示行中缓存块来自哪个主存块。

地址变换时需要比较所有行，全相联映射有两种组织方式，一是使用一个比较器，轮流比较所有 Cache 行；二是使用 G 个比较器，同时比较所有 Cache 行。显然，只能采用第二种方式，因为设置 Cache 的目的是提高访存速度，第一种方案实在太慢了。

全相联映射的地址变换过程为：用主存地址的高 m 位与各个 Cache 行的标记进行比较，再对所有比较器的输出端进行逻辑或运算，若结果为真，则表示 Cache 命中，否则为 Cache 缺失，如图 3.47(b) 所示。

Cache 命中时，用主存地址的低 b 位（块内地址）对所有 Cache 行进行存储字读/写操作。由于最多只有一个 Cache 行命中，只要多选器带有三态功能，各个 Cache 行同时读/写就不会产生错误。

全相联映射的候选行有 G 个行，只要还剩一个空闲行，块调入时就不会产生冲突，因而冲突概率最低。地址变换需要 G 个比较器，速度很快，成本最高。

例 3.10　假设计算机的主存编址单位为字节，CPU 可寻址空间为 20 位，Cache 容量为 8KB，主存块大小为 16B，Cache 与主存之间采用全相联映射方式。问：

（1）为了实现映射，主存地址应该如何划分？各字段长度分别为多少位？

（2）Cache 有多少行？Cache 行的标记为多少位？

（3）若 CPU 访问主存的地址为 06454H 时，则可能命中的 Cache 行号是多少？命中行的标记的值是多少？

解：（1）主存地址可以划分为主存块号、块内地址，其中，主存地址为 20 位，块内地址为 $\log_2$(16B/1B)＝4 位，主存块号为 20－4＝16 位。

（2）Cache 有 8KB/16B＝512 行，Cache 行使用主存块号作为标记，标记为 16 位。

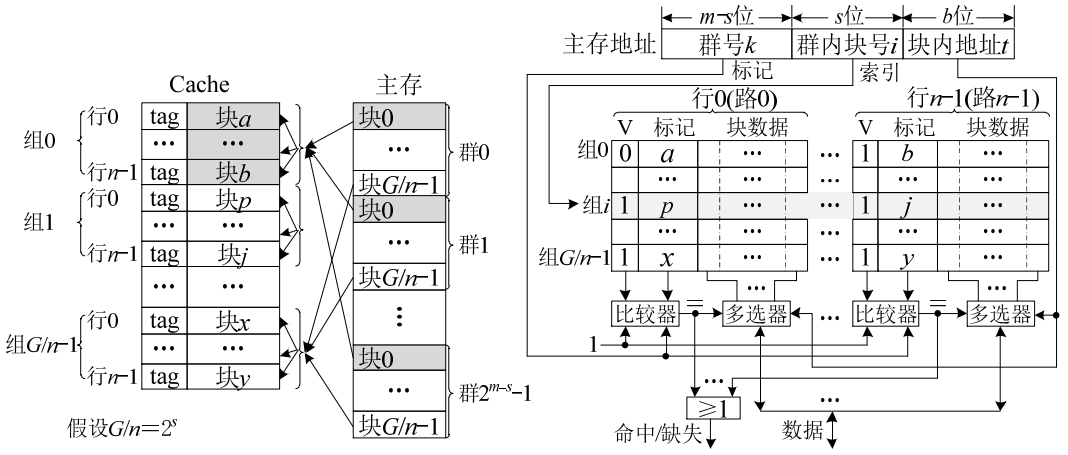
（3）主存地址 06454H＝0000 0110 0100 0101 0100B，高 16 位 0645H 为主存块号，Cache 的任意一行 j 都可能被命中，即 $0 \leqslant j \leqslant G-1$；命中行的标记的值为 0645H。

3．组相联映射（Set associate mapping）

组相联映射是直接映射和全相联映射的一种折中方案。组相联映射将 Cache 的行划分成若干个组，每个组所含行数为 n，一个主存块可以存放到一个 Cache 组的任意一行中，即组间直接映射、组内全相联映射，候选行数有 n 行。

若主存块 i 可以存放到 Cache 组 j 的任意一行中，则映射函数为：$j=i \bmod G/n$，其中 G 为 Cache 的行数，n 为组内行数，映射关系如图 3.48(a) 所示。通常，将组内行数为 n 的组相联映射称为 n 路组相联映射。

由图 3.48(a) 可见，主存地址可以划分成群号、群内块号、块内地址三个字段，其中，群号、群内块号合起来为主存块号；Cache 行号由组号、组内行号组成，组内行号为 $\log_2 n$ 位。组相联映射时，群内块号等于 Cache 组号，因此，群内块号为 s 位，群号为 $m-s$ 位。由于群内块号用作索引，因此，Cache 行只要使用群号作为标记（Tag），就可以表示行中缓存块来自哪个主存块。注意，类似于区，群也是本教材为便于描述而定义的概念，群的大小等于 Cache 容量的 $1/n$。



（a）组相联映射关系　　（b）组相联映射的组织

图 3.48　组相联映射方式

与全相联映射类似，为了保证速度，地址变换时应同时比较组内所有 Cache 行。因此，组相联映射的地址变换过程为：用主存地址的中间 s 位为索引，找到对应的 Cache 组；用主存地址的高 $m-s$ 位与组内各个 Cache 行的标记进行比较，再对所有比较器的输出端进行逻辑或运算，若结果为真，则表示 Cache 命中，命中行的多选器输出非高阻态，否则为 Cache 缺失，如图 3.48(b) 所示。

Cache 命中时，用主存地址的低 b 位（块内地址），对组内的所有 Cache 行进行存储字读/写。由于每个组最多只有一个 Cache 行命中，多选器带有三态功能时，各个 Cache 行的同时读/写就不会产生错误。

组相联映射的候选行有 n 个行，只要组内 n 个行中还有一个空闲行，块调入时就不会产生冲突，冲突概率随着 n 增大而快速下降。地址变换需要 n 个比较器，速度很快，成本一般。可见，组相联映射结合了直接映射、全相联映射的优点。

例 3.11　假设计算机的存储器按字节编址，CPU 可寻址空间为 20 位，Cache 可存放 8KB 数据，主存块大小为 16B，Cache 与主存之间采用 4 路组相联映射方式。问：

（1）Cache 有多少个组？Cache 行号应如何划分？各字段长度分别为多少位？

（2）为了实现映射，主存地址应该如何划分？各字段长度分别为多少位？

（3）Cache 行的标记为多少位？

（4）若 CPU 访问主存的地址为 06454H 时，则可能命中的 Cache 组号是多少？命中行的标记值是多少？

（5）若 Cache 初始状态全部为空闲，CPU 从主存第 0 号单元起，依次读出 100 个字节（每次读出一个字节），并重复再读 4 次，则 Cache 命中率是多少？

解：（1）Cache 有 8KB/16B＝512 行，有 512/4＝128 个组；Cache 行号由组号、组内行号组成，组内行号为 $\log_2 4=2$ 位，组号为 $\log_2 512-2=\log_2 128=7$ 位。

（2）主存地址可以划分为群号、群内块号、块内地址，其中，主存地址为 20 位，块内地址为＝4 位，群内块号与 Cache 组号等长（7 位），群号为 20－4－7＝9 位。

（3）Cache 行使用群号作为标记，标记为 9 位。

（4）主存地址 06454H＝0000 0110 0100 0101 0100B，高 9 位 0000 0110 0B 为群号，中间 7 位 100 0101B 为群内块号，命中时的 Cache 组号等于群内块号，因此，可能命中的组号为 100 0101B，命中行的标记值为 0000 0110 0B＝00CH。

（5）第 0 号主存单元起的 100 个字节，存放在连续的⌈(100＋0)/16⌉＝7 个主存块中，即第 0~6 号主存块。组相联映射方式中，第 0~6 号主存块可映射到 Cache 第 0~6 组的任意一行，由于 Cache 的初始状态为空闲，第 0~6 号主存块调入到 Cache 时，不会产生冲突。由于是连续读出，CPU 第 1 遍读时，每个块的第一次读操作都会缺失，其余字节都会命中，因此有 7 次缺失。而第 2~5 遍读时，由于 7 个主存块已经在 Cache 中，读所有字节都会命中，因此，Cache 命中率为(100×5－7)÷(100×5)＝0.986。

注意，数据所占主存块数与存放首地址有关。若上例（5）中的 100 个字节从第 814 号单元起存放，则数据存放在连续的⌈(100＋814%16)/16⌉＝8 个主存块中。

下面我们来比较一下三种映射方式的特性。

三种映射方式可以统一用组相联方式表示，直接映射为 1 路组相联映射，全相联映射为 G 路组相联映射（G 为 Cache 行数），组相联映射为 n 路组相联映射（$1<n<G$）。三种映射方式的候选行数分别为 1、G、n。

候选行数通常称为相联度，即一个主存块能够映射到的 Cache 行数。可以发现，相联度越大，调入时的冲突率越低，Cache 命中率就越大；Cache 行标记的位数越多，所需成本越大（比较器越多）。

3.4.3　Cache 的替换算法

当 Cache 需要调入主存块，而又没有空闲行时，就需要用某种方法选择一个牺牲块，腾出空间用于块调入。替换算法指从候选行中找出牺牲块的方法，又称为淘汰策略。替换算法找出了牺牲块，也意味着找出了牺牲行。候选行为哪些 Cache 行由地址映射规则确定，三种映射方式的候选行数分别为 1、G 和 n。

替换算法通常有随机（RAND）、先进先出（FIFO）、最近最少使用（LRU）等类型。

不同替换算法对 Cache 命中率的影响不同，实现开销也不同。对 Cache 命中的影响主要看所选择行中数据将来的使用情况，是否遵循程序访问局部性，即是否很少使用。

1．随机算法（Random，RAND）

随机算法的基本思想是在候选行中随机选择一个牺牲块。这种算法的选择，与所选行中数据的使用情况无关，候选行数多少与 Cache 命中率无关，因此，RAND 算法对 Cache 命中率的影响是随机的，也就是说可能会降低命中率。

这种算法实现时，整个 Cache 只需要一个随机数发生器，即可产生牺牲行的行号，实现开销最低。

2．先进先出算法（First In First Out，FIFO）

先进先出算法的基本思想是，选择最早调入的块作为牺牲块。这种算法的选择，同样没有反映程序访问局部性，因为最早调入的块可能是经常被使用的块，增加候选行数

也不会提高 Cache 命中率，因此，FIFO 算法对 Cache 命中率的影响也是随机的，但比随机算法要好得多。

这种算法实现时，需要在候选行的每一行设置一个计数器，用各个计数器的值来表示候选行中各个块的调入顺序，替换时选择计数值最大的行作为牺牲行，每当有块被调入时更新所有计数器的值。

由于计数器仅用于表示候选行中的顺序，每个计数器的位数为 $\log_2$(候选行数)，因此，FIFO 算法较适宜与组相联 Cache，不适宜用于全相联 Cache。

先后顺序也可以使用比较对法实现，其原理与计数器法相同，只是成本更低些。

3．最近最少使用算法（Least Recently Used，LRU）

最近最少使用算法的基本思想是，选择近期最少使用的块作为牺牲块。这种算法的选择较好地反映了程序访问局部性，因为基于程序访问局部性的推论是：近期最少使用的块，通常也是将来最少被访问的块。由于该算法的选择遵循了程序访问局部性，增加候选行数时，可以提高 Cache 的命中率，因此，LUR 算法不会对 Cache 命中率产生负面影响。

这种算法实现时，硬件配置与 FIFO 算法完全相同，但处理方法不同。LRU 算法用各个计数值来表示各个块的访问顺序，替换时选择计数值最大的行作为牺牲行，每当有块被访问时更新所有计数器的值。

所有计数器的初值为全 1（最大计数值），计数器的更新方法为：计数值比被访问行小的计数器加 1，被访问行的计数器清 0。

Cache 进行 LRU 算法组织时，通常在各个 Cache 行的管理信息中设置“LRU 位”，放在图 3.43 的状态字段中，“LRU 位”的位数就是计数器的位数。

由于 LRU 算法较适宜于组相联 Cache，而且 Cache 命中率随着相联度的增加而提高，因此，现代的组相联 Cache 都采用 LRU 替换算法。

可见，三种替换算法中，只有 LRU 算法不会降低 Cache 命中率。

例 3.12 假设某全相联 Cache 有 4 行，采用 LRU 替换算法，且 Cache 的初态为全部空闲，CPU 访存时每个块连续访问 4 次，访问的块地址流为 2、11、2、9、7、6、3、9、6、3，计算此时的 Cache 命中率。

解：全相联映射只有一个组，候选行为 4 行，LRU 算法的计数器为 $\log_2 4=2$ 位。

Cache 处理 CPU 访问的过程如表 3.1 所示，表中，用底纹标出的单元格表示该行此时处于空闲状态，单元格中虚线左边为标记（块号），右边为计数器的值。

表 3.1 Cache 处理 CPU 访问的过程示例

块地址流		2	11	2	9	7	6	3	9	6	3
行状态	行 0	2 : 0	2 : 1	2 : 0	2 : 1	2 : 2	2 : 3	3 : 0	3 : 1	3 : 2	3 : 0
	行 1	: 3	11 : 0	11 : 1	11 : 2	11 : 3	6 : 0	6 : 1	6 : 2	6 : 0	6 : 1
	行 2	: 3	: 3	: 3	9 : 0	9 : 1	9 : 2	9 : 3	9 : 0	9 : 1	9 : 2
	行 3	: 3	: 3	: 3	: 3	7 : 0	7 : 1	7 : 2	7 : 3	7 : 3	7 : 3
操作状态	第 1 次	调入	调入	命中	调入	调入	替换	替换	命中	命中	命中
	第2~4 次	命中	命中	命中	命中	命中	命中	命中	命中	命中	命中

可见，LRU 算法更新时，只更新自己、计数值比自己小的计数器；LRU 算法替换时，选择计数值最大的行作为牺牲行。

从表中可以看出，CPU 共进行了 10×4＝40 次访问，其中有 6 次不命中，故 Cache 命中率＝(40－6)/40×100%＝85%。

3.4.4　Cache 的写策略

由于 Cache 中信息是主存中信息的副本，应当保持与主存一致，若每次写操作都写主存，又会增加命中时间，Cache 很难兼顾提高访存速度以及与主存保持一致性这两个矛盾的要求。根据侧重点的不同，Cache 有不同的写操作处理方法。

写策略指 Cache 将 CPU 所写数据写回主存的时机及方法，涉及写命中、写缺失两种情况，通常有全写法、写回法两种。不同写策略对平均访问时间的影响不同。

1．全写法（Write Through）

全写法又称写直达法，基本思路是，写命中时，将数据写入 Cache，同时写入主存；写缺失时，通常直接将数据写入主存，而不将目标主存块调入 Cache。

可见，全写法采用了立即写思想，Cache 与主存一直保持一致性，缺点是命中时写延迟较大（写主存的延迟），对总线带宽的要求较高（每次写操作都占用总线）。

写缺失处理共有两种方法。一种称为按写分配法（write allocate），先将目标块调入 Cache，再将数据写入该 Cache 行；另一种称为不按写分配法（no write allocate），直接将数据写入主存，而不将目标块调入 Cache。

全写法的写缺失处理通常采用不按写分配法，以提高写缺失性能，写缺失的延迟与写命中相同。可见，全写法只有在读缺失时，才会将主存块调入 Cache。

由于 Cache 与主存保持了一致性，全写法在替换时，只需将牺牲行抛弃即可（有效位清零），而无须将其写入主存。基于全写法的 Cache 工作流程如图 3.49(a) 所示。

2．写回法（Write Back）

写回法又称回写法，基本思路是：写命中时，只将数据写入 Cache，不写入主存；写缺失时，通常先将目标主存块调入 Cache，再将数据写入 Cache；仅当 Cache 行中主存块被替换、且主存块被修改过时，才将该行暂存的主存块写回主存。

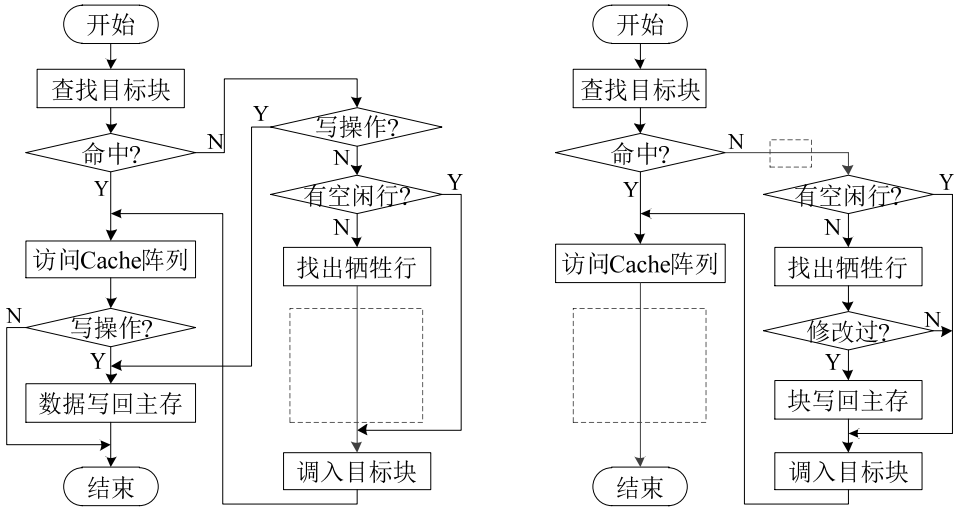
可见，写回法采用了稍后写思想，命中时写延迟很小（写 Cache 的延迟），占用总线次数少（替换时才占用总线），但 Cache 没有与主存保持一致性。

写回法的写缺失处理通常采用按写分配法，以利用访问局部性，来减小平均访问时间，但缺失开销比较大。

由于仅当牺牲行中内容被改写过时，才将该主存块写回主存，因此，每个 Cache 行的管理信息中，需要设置一个修改位（Dirty bit），又称脏位，表明该行是否被改写过。

因此，Cache 在处理写操作时，需要将脏位置为 1；在进行替换处理时，需要判断脏位，确定是否将该缓存块写入主存。基于写回法的 Cache 工作流程如图 3.49(b) 所示。

由图 3.49 可见，两种写策略的主要区别是立即写主存还是替换时写主存，写主存的数据粒度也有所不同。



（a）全写法　　（b）写回法

图 3.49　不同写策略时的 Cache 工作流程

例 3.13　假设计算机的主存按字节编址，CPU 可寻址空间为 32 位，Cache 可存放 8 KB 数据，主存块大小为 16B，Cache 与主存之间采用 4 路组相联映射方式，Cache 采用 LRU 替换算法、写回法写策略，则每个 Cache 行的管理信息至少有多少位？

解：Cache 有 8KB÷(16B×4)＝128 组，主存地址的群号为 32－$\log_2$16－$\log_2$128＝21 位，因此，Cache 行的标记为 21 位。4 路组相联映射的候选行为 4 行，LRU 替换算法的 LRU 位为 $\log_2$4＝2 位。写回法写策略需要每行设置脏位（1 位）。

每个 Cache 行的管理信息包括有效位（V）、标记（Tag）、LRU 位、脏位（M），共有 1＋21＋2＋1＝25 位。

3.4.5　Pentium 的 Cache 组织

Cache 的主要性能指标为平均访问时间，即 $T_A = T_{命中} + (1-H) \times T_{缺失}$，影响 T_A 的因素有很多，如主存块大小、相联度、替换算法、Cache 容量等，与 Cache 的结构也有很大关系。在此，我们先介绍 Pentium 使用的 Cache 结构，然后再了解 Pentium 的 Cache 组织方法。

1．新的 Cache 结构

1）多级 Cache 结构

Cache 技术开始出现时，存储系统中只有单级 Cache。随着 CPU 的速度越来越快，对 Cache 的要求越来越高，单一的 Cache 已不能满足要求。

目前，Cache 都采用多级结构，如 L1 Cache 和 L2 Cache，且 L1 Cache 中的内容是 L2 Cache 的子集。L1 Cache 容量最小，致力于减小命中时间；L2 Cache 容量较大，主要目标是提高命中率、减少缺失开销。

实践证明，两级 Cache 系统性能较好。目前的三级 Cache 是由多核 CPU 引起的，每个 CPU 核是两级 Cache，各个 CPU 核再同享 L3 Cache，以减少缺失损失。

2）哈佛结构

现代 CPU 都采用流水线来提高性能，而流水线中的各个操作都是重叠的，需要并行访问存储器。例如指令流水线中，取指令、取操作数可能会同时发生，从而产生存储器访问冲突。

将 Cache 组织成独立的指令 Cache（I-Cache）和数据 Cache（D-Cache），可有效解决指令和数据的并行访存冲突。这种结构由哈佛大学提出，故命名为哈佛结构。通常，将采用哈佛结构的 Cache 称为分离 Cache，将指令和数据混存的 Cache 称为联合 Cache。

哈佛结构将存储器分为指令存储器、数据存储器，颠覆了冯·诺依曼模型计算机一个存储器的模式，相应地，有人将单个存储器的结构称为冯·诺依曼结构。

由于现代计算机的存储系统仍是单一存储器模型，因此，只有紧邻 CPU 的 L1 Cache 会采用哈佛结构，其余 Cache 都为联合 Cache。

2．Pentium 的 Cache 组织

Pentium CPU 是一个 32 位 CPU，具有 2 路超标量流水线，即两条流水线可以同时执行。Pentium 的 Cache 子系统如图 3.50 所示，采用的是两级 Cache 结构，其中，L1 Cache 采用哈佛结构，L1 Cache 与 L2 Cache 为级联（贯穿式）结构。PII CPU 开始，所有 Cache 都放在 CPU 芯片内部。

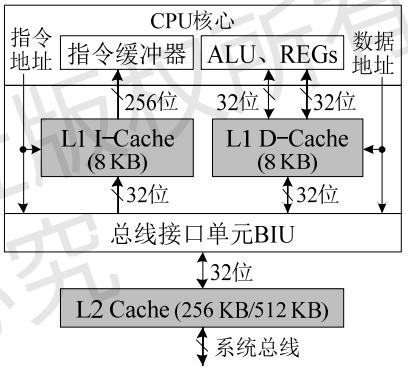


图3.50　Pentium的Cache子系统

由于 CPU 中有两条并行流水线，为了保证流水线性能，L1 D-Cache（常记为 L1-D$）与 CPU 间起码有 2 条独立的数据链路，故 L1-D$由双端口存储器组成；又由于 2 条流水线执行相邻指令，因此，L1-I$只需要增加链路宽度即可。

Pentium CPU 的主存按字节编址，CPU 可寻址空间为 32 位，主存块大小为 32 B，L1$及 L2$都采用 2 路组相联映射方式及 LRU 替换算法，L1-I$为只读 Cache，L1-D$采用写一次法写策略（写回法的变种），L2$采用写回法写策略。

下面，我们以 L1-D$为例，来了解 Pentium 的 Cache 是如何组织的。

L1-D$的容量为 8KB，采用 2 路组相联映射方式，因此，共有 8KB÷32B＝256 个行，有 256/2＝128 个组。为了保证地址变换速度，Cache 应该能够同时查找和比较一个组中所有行的信息，因此，L1-D$采用一个组一行的组织方式，如图 3.51 所示，目录表分为目录表 0 和目录表 1，共有 128 行。

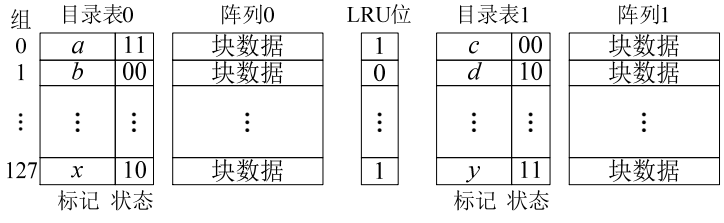


图 3.51　Pentium 的 L1 D-Cache 组织

Cache 的行号由组号（7 位）、组内块号（1 位）组成，主存地址由群号（20 位）、群内块号（7 位）及块内地址（5 位）组成。由于地址映射时，群内块号与 Cache 组号是直接映射，因此，群内块号为 7 位，群号为 32－7－5＝20 位。因此，Cache 行的标记为 20 位。

对 L1-D$的每个行而言，管理信息至少应该由有效位（1 位）、标记（20 位）、LRU 位（1 位）及修改位（1 位）组成。图中的状态就是有效位和修改位。由于是 2 路组相联，组内 2 个行的 LRU 位的值互斥，因此，可以一个组共用一个 LRU 位。

因此，每个组的管理信息有(1＋20＋1)×2＋1＝45 位，目录表大小为 45 位×128＝720B，Cache 的总容量为 8×1024B＋720B＝8912B。

3.5 虚拟存储器

现代计算机都是多任务系统，允许多个进程同时运行，这些进程需要分享主存的存储空间。主存空间如何分配给进程使用，如何进行进程空间的保护，如何消除主存容量对编程的限制等，都是存储管理（又称主存管理）必须面对的问题。

存储管理的方案由计算机系统结构确定，由操作系统及硬件协同实现，硬件主要为存储器管理单元 MMU（Memory Management Unit），计算机组成的主要任务是有效地组织 MMU。

3.5.1 存储管理的相关概念

存储管理指的是主存管理，辅存管理属于设备管理范畴。存储管理的内容主要包括存储空间分配、进程空间保护、存储空间扩充等方面。下面，仅介绍与存储空间相关的基本概念。

1．程序的地址空间

计算机中，程序需要装入主存才可以执行，因此，程序中的存储单元长度必须与主存单元相同；程序生成时并不知道其装入主存的具体位置，因此，每个程序使用的存储器地址都从 0 开始编址。通常，将程序中的存储器地址称为逻辑地址，对应的地址空间称为逻辑地址空间。相应地，将主存的地址称为物理地址，主存地址空间又称为物理地址空间。

为了便于管理，同一计算机中各个程序的逻辑地址空间大小是一致的；为了面向各种应用，逻辑地址空间通常都比较大。例如，Intel i7 CPU 支持 48 位（256TB）逻辑地址空间。

程序通常由多个逻辑上独立的段构成，如 Windows 系统中的程序，逻辑上独立指每个段内部都从 0 开始编址，因此，逻辑地址由段号及段内地址组成。为了便于管理，同一计算机中，所有程序的段号、段内地址的位数都是固定的，如 IA32 中分别为 16 位、32 位。程序分段时，代码、数据、堆栈等不同内容使用不同的段，好处是很容易实现进程保护，缺点是存储管理比较麻烦。程序也可以不分段，如 Linux 系统中的程序，代码、数据等不同内容使用同一个逻辑地址空间，不同内容的起始地址是固定的，优缺点

与程序分段方式相反。

程序都是以文件形式存放在辅存中的，为了实现逻辑空间与文件空间的映射，程序文件都由程序头、各段内容组成，程序头区域中存放有每个段的映射信息，如段号、段长、段类型、段在文件中的偏移地址等，其中，每个段的段号及段长指明了该段所在的逻辑地址范围。

2．主存空间的分配

存储空间分配指如何将主存分配给各个进程使用，早期有分区（Partitioning）、分页（Paging）两种管理方式。给进程分配主存空间后，每个逻辑地址就有了唯一的物理地址，访存时需要先将逻辑地址转换成物理地址，存储空间管理方式不同，对应的地址变换方法就不同。地址变换都由硬件实现，本书将该地址变换硬件也称为 MMU。

1）分区方式

分区方式是将主存划分成若干个分区，分配给进程时，每个进程占一个分区。根据分区的大小是否可以改变，分区又有固定分区、可变分区两种方式。

固定分区方式中，各个分区的大小不可改变，存储空间分配时，从空闲分区中选择超过进程所占空间的最小分区。由于分区大小固定，各个进程所占空间不同，每个分区都会有不少无法使用的空间（称为碎片），因此，固定分区方式会浪费大量主存空间。

可变分区方式中，各个分区的大小可以改变，存储空间分配时，从空闲分区中切割出一个进程所占空间的分区。虽然可以通过拼接技术合并空闲分区，但运行若干个进程后，还是会有不少碎片存在，使用移动技术可以减少碎片，但所需开销很大。

分区方式中，进程全部装入主存，占用连续的主存空间，因此，地址变换的方法是：物理地址＝基地址＋逻辑地址，其中基地址为分区的首地址。地址变换由 MMU 实现，MMU 只需要配置一个基址寄存器，一个地址加法器，程序执行时，操作系统预先将所分配的分区首地址写入到基址寄存器中即可。

2）分页方式

分页方式是将主存空间划分成若干大小相等的区域，每个区域称为一个页框（Page Frame），对应地，每个进程所占空间也划分成若干大小相等的片段，每个片段称为一个页，页和页框大小相同。存储空间分配时，一次性分配进程所需的页框数，页框可以是不连续的，用页表（Page Table）来管理页和页框的映射关系。与分区方式相比，由于页比较小（通常为 4KB），每个进程产生的碎片一般不超过一个页框，因此，主存空间利用率比较高，但管理开销稍大。

分页方式中，主存地址由物理页号及页内地址组成，逻辑地址由逻辑页号及页内地址组成，进程全部装入主存，占用不连续的主存空间，因此，地址变换的方法是：物理页号＝页表[逻辑页号]，即物理页号是用逻辑页号查页表得到的，物理地址由物理页号及页内地址拼接而成。MMU 需要配置的硬件较复杂，在虚拟存储器中一并讨论。

3．存储空间的扩充

分区和分页存储管理方式都要求程序全部装入主存，实际应用中，程序大小很可能超过主存容量，怎么办？

早期使用的是覆盖技术，程序员将程序分成多个可覆盖的片段，给程序分配的存储空间为不可覆盖片段之和，程序执行过程中，由用户程序控制程序片段的装入与换出，由程序员保证程序不会访问未装入的片段。覆盖技术虽然允许程序大小≥主存容量，但对程序员有一定的负担。

目前使用的是虚拟存储技术，程序执行过程中，只将当前所需的程序装入主存，其余部分暂存在辅存中，主存缺失时自动与辅存进行信息交换，这种借助于辅存，为程序提供的比主存空间大得多的存储空间称为虚拟存储器（Virtual Memory, VM），又称虚拟主存，或虚存。

有人不禁要问，为什么不一开始就使用虚拟存储技术？其实，虚拟存储技术早就提出来了，但它需要硬件的支持，受限于器件成本，时机成熟时才能广泛应用。

3.5.2 虚拟存储器的基本原理

1. 虚拟存储器的组成

虚拟存储器（VM）是一个以透明方式为程序提供的、比主存空间大得多的存储空间。通常，将虚拟存储器的存储单元地址称为虚拟地址（虚地址），将虚拟存储器的地址空间称为虚拟地址空间，主存地址相应地称为物理地址（实地址）。

对每个进程而言，虚拟存储器的特性是：存储器可以独占，存储器的地址空间比主存空间大得多。独占指每个进程都有一个虚拟地址空间，为了简化存储管理，各个进程的虚拟地址空间大小相同，由于虚拟地址空间是进程私有的，很便于进程空间保护。

虚拟存储器的基本思想是，借助于辅存来扩充主存的存储空间，并能够按虚拟地址来进行访问。可见，虚拟存储器是用主存与辅存实现的、按虚拟地址访问的存储器模型，如图 3.52 所示。

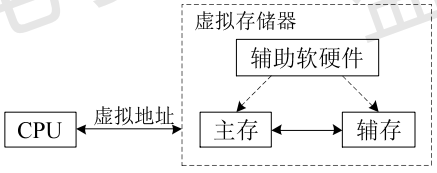


图3.52　虚拟存储器的组成

虚拟存储器的相关参数都由计算机系统结构确定，通常虚拟地址空间大小等于程序逻辑地址空间大小，逻辑地址又常称为虚拟地址。例如，Intel Core i7 的虚拟地址为 48 位，虚拟地址空间为 2^{48}＝256TB，而物理地址为 44 位，物理地址空间为 2^{44}＝16TB。

与 Cache-主存层次不同的是，由于虚存的存储空间巨大，考虑成本因素，层次管理只能采用硬件、软件相结合的方法来实现，其中的硬件主要是存储器管理单元 MMU。

2. 虚拟存储器的工作过程

虚拟存储器中的信息存放在主存和辅存中，主存用作辅存的高速缓存，虚拟存储器按虚拟地址进行访问，因此，虚拟存储器中存在三个地址空间：虚存地址空间、主存地址空间、辅存地址空间，存在两种映射：虚存-主存映射、虚存-辅存映射，主存-辅存交换信息时需要查找两个映射表。

与 Cache 类似，虚拟存储器在处理访问请求时，首先进行虚存-主存地址变换，若变换成功，则访问主存完成请求；否则，进行虚存-辅存地址变换，同时在主存中找出（或腾出）一个空闲位置，将信息从辅存装入到主存中，再访问主存完成请

求，如图 3.53 所示。

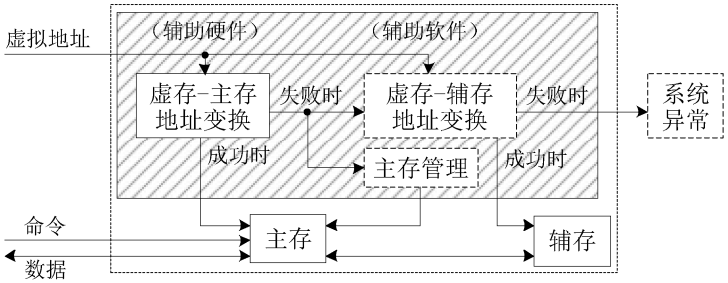


图 3.53　虚拟存储器的工作过程

为了保证访存速度，层次管理工作由硬件、软件协同完成。虚存-主存地址变换必须由 MMU 完成，变换失败时产生异常信号，触发执行缺页异常处理程序来完成剩余工作，并重新执行产生缺页异常的指令，以保证程序正确地执行。

可见，虚拟存储器实际上是一个按虚拟地址访问的主存-辅存层次，是面向程序的存储器模型。注意，并不是所有的主存-辅存层次都是虚拟存储器，前提是 CPU 可以按虚拟地址访问这个存储层次。

3.5.3　虚拟存储器的存储管理

虚拟存储器中，主存用作辅存的高速缓存，因此，主存-辅存之间的信息交换单位就是虚存-主存之间、虚存-辅存之间的存储管理单位。

程序通常由多个段组成，为了提高地址变换速度、便于实现进程保护，主存-辅存之间适宜用程序段作为信息交换单位。为了提高主存空间利用率、减少碎片数量，主存-辅存之间适宜用主存页作为信息交换单位。相应地，虚拟存储器的存储管理方式有段式、页式、段页式三种类型，其中，段页式管理方式是段式管理与页式管理的混合体，可以具备两者的优点。

虚拟存储器的存储管理方式不同，其实现方法就有所不同。虚拟存储器通常用其存储管理方式来命名，因此，共有三种虚拟存储器：段式虚拟存储器、页式虚拟存储器、段页式虚拟存储器。

1．段式虚拟存储器

段式虚拟存储器中，程序由多个段组成，段的类型有多种，如代码段、数据段、堆栈段等，同一类型的段可以有多个，每个段的大小可以不同。

段式虚拟存储器采用段式存储管理方式。段式管理指按程序逻辑结构将虚存空间划分为若干个段，段大小为程序中段的大小，主存空间以段为单位分配给虚存使用，虚拟地址由段号及段内地址组成。

段式管理中，用段表来指明虚存各个段在主存中的位置，由于段的长度可变，段表中应有所指示，为虚存保护提供源数据。通常，段表的表项由装入位、段首址、段长等字段组成，如图 3.54 所示，程序的每个段占一行，装入位表示该段是/否已装入主存，与 Cache 中有效位的含义相同。

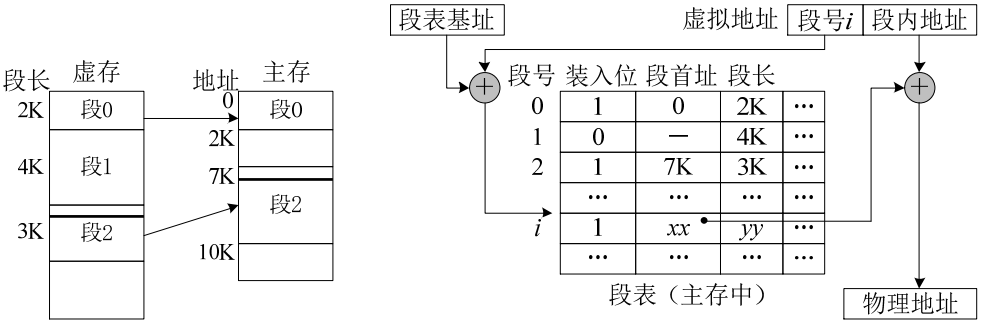


图 3.54　段式虚存的段表及地址变换过程

假设段号为 16 位，段表项长度为 8B，则段表最大为 512KB，32 个进程的段表就可能需要 16MB 空间。为了降低管理成本，段表只能放在主存中，进而，查表时只能访问一行。因此，段表必须按虚存段号进行索引，段表项中无须设置段号字段。

虚拟存储器的地址变换由 MMU 实现，MMU 使用寄存器保存段表的基地址。地址变换时，先要计算出目标段表项在主存中的首地址，然后读回段表项，再判断装入位、计算物理地址，具体过程如图 3.54 所示，图中的⊕表示地址加法器。

段式虚存的优点是，段的分界就是程序段的分界，便于实现程序的共享与保护，如代码段不能写可以防病毒。缺点是主存会产生许多碎片，空间利用率不高。

2．页式虚拟存储器

页是大小固定的存储块。称为页的原因主要是为了有别于 Cache−主存层次的块，由于辅存的缺失开销巨大，页比块要大得多，通常为 4KB。

页式虚拟存储器采用页式存储管理方式。页式虚存管理指将虚存空间及主存空间按页大小划分成若干个页，主存以页为单位分配给虚存使用，虚拟地址由虚拟页号（VP）及页内地址组成，主存地址由物理页号（PP）及页内地址组成。注意，虚拟地址、主存地址的位数都是固定的，由计算机系统结构确定，与虚存管理方式无关。

页式管理中，用页表来指明虚存所有页在主存中的位置，页表项由装入位、物理页号等字段组成，如图 3.55 所示，进程的每个页占一行。出于与段表一样的原因，页表也放在主存中，页表按虚拟页号（虚页号）进行索引，注意页表项中没有虚拟页号字段。

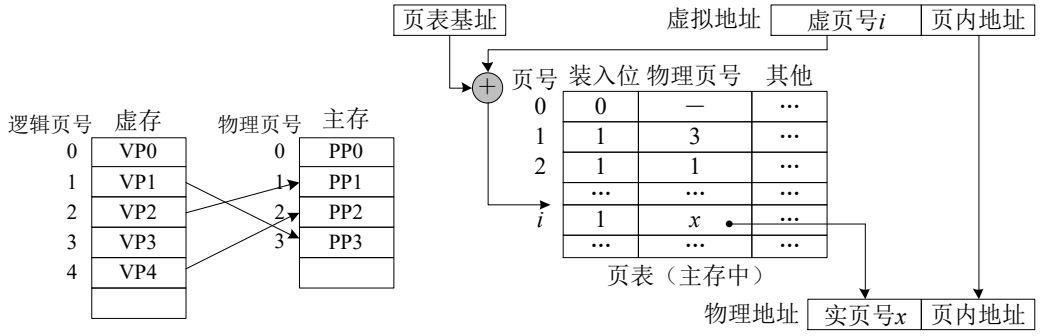


图 3.55　页式虚存的页表及地址变换过程

页式虚存的页表基址保存在 MMU 中，地址变换过程如图 3.55 所示，与段式虚存的地址变换基本相同，只是物理地址不需要进行计算，而是通过拼接来实现。

页式管理的优缺点与段式管理相反，主存空间利用率较高，但不易实现程序的共享与保护。

3．段页式虚拟存储器

段页式虚拟存储器采用段页式存储管理方式。段页式虚存管理指先按程序逻辑结构将虚存空间分段，再对每个段按页大小分页，主存仅按页大小分页，主存以页为单位分配给虚存使用。可见，段页式虚拟存储器中，主存-辅存的信息交换单位是页，虚拟地址由段号、虚页号及页内地址组成，虚页号与页内地址的位数等于段内地址的位数，主存地址由物理页号及页内地址组成。

段页式管理中，用一个段表和一组页表来指明虚存所有页在主存中的位置。段表与段式管理的段表基本相同，只是表项中的段首址换成了页表基址，页表与页式管理的页表完全相同，如图 3.56 所示。

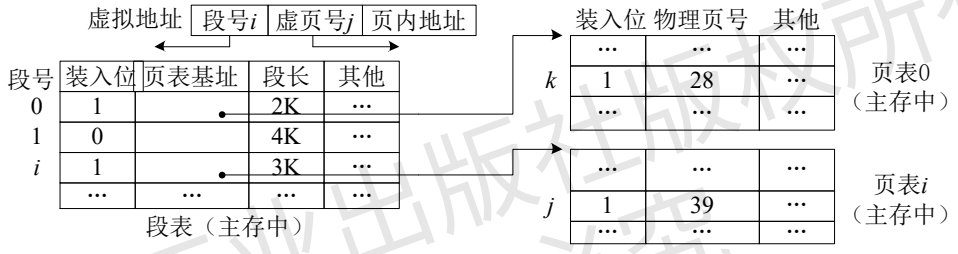


图 3.56　段页式虚存的段表和页表

段页式虚存只有段表基址保存在 MMU 中，地址变换时，先查段表找到该段的页表基址，再查页表进行页式地址变换。

段页式虚存兼备了段式虚存和页式虚存的优点，缺点是地址变换时需要两次查表。

实际应用中，比较重视主存空间利用率，通常使用页式虚拟存储器、段页式虚拟存储器。下面就以页式虚拟存储器为例，来讨论虚拟存储器的组织。

3.5.4　页式虚拟存储器的实现

页式虚拟存储器采用页式存储管理方式，主存空间以页为单位分配给虚存使用。虚存实现时，涉及缓存管理、页表组织、地址变换、缺页（Page Fault）处理、存储保护等方面。

1．缓存的管理

虚拟存储器中，主存是辅存的缓冲存储器，主存-辅存的信息交换单位为页，虚存空间中的页会被映射到主存及辅存中，因此，存在虚存-主存、虚存-辅存两种映射。

由于主存的缺失开销巨大，为了提高主存的利用率，虚存-主存的映射只能采用全相联映射方式。为了提高访存命中率，替换算法应采用近似 LRU 算法或更精密的算法，近似 LRU 算法的思想是 LRU，实现不能采用计数器式更新方法，因为页表在主存中，访存开销较大。为了减少命中时间，主存的写策略只能采用写回法。

虚存-辅存映射指的是虚存空间与程序文件的存储空间的映射，这个映射由逻辑空间-文件空间、文件空间-文件存储空间两个环节组成，前者由程序文件的程序头来实现（3.5.1 节已有说明），后者由操作系统的文件系统等来实现，暂不讨论。

2．页表的组织

前面已经讲过，由于页表所占空间很大，页表必须放在主存中，查表时应该只访问一次主存，因此，页表必须按虚拟页号进行索引。

页表项（PTE）的基本字段有装入位（又称有效位）、物理页号，还需要包含缓存管理及存储保护所使用的访问位、修改位、保护位等字段，基本字段用于实现虚存-主存映射。下面是一个页表项示例，其中，修改位用于主存的写回法写策略，访问位表示近期是否被访问过，用于主存的近似 LRU 替换算法，读/写位用于访问保护，禁止缓存位表示该页是否可以装入 Cache，用于 Cache-主存间的一致性管理。

装入位	物理页号	访问位	修改位	读/写位	禁止缓存位	…

为了减少页表所占存储空间，页表长度都是程序所占的行数，而不是虚存地址空间对应的行数，因此，MMU 及页表中需要进行相应处理，来实现进程空间保护，如防止越界访问。

地址变换时，页表项的首地址是使用加法得到的，如图 3.55 所示，若页表大小超过一个页面时，地址变换就会出错，因为页的分配是可以不连续的。常见的解决方案是采用多级页表，类似于图 3.55，第一级页表的表项中存放第二级页表的基地址，第二级页表的表项中才存放物理页号；虚页号也划分为一级虚页号、二级虚页号；地址变换时，分别进行索引，需要两次查表。

3．地址变换的实现

由于页表按虚拟页号为索引进行组织，页表存放在主存中，因此，页式虚存管理的地址变换过程，就是一个查表、判断过程，如图 3.55 所示。地址变换由 MMU 负责实现，页表基址、页表长度都存放在 MMU 的寄存器中。

MMU 接收到访问请求后，地址变换的操作步骤如图 3.57 所示，第②步生成页表项首地址，生成方法参见图 3.55；第③步读出页表项内容；第④步判断装入位，为 1 时用页表项构造物理地址，并用该地址访存（第⑤步），构造方法参见图 3.55，否则，产生一个异常，触发执行缺页异常处理程序，进行第⑤~⑦步的操作。

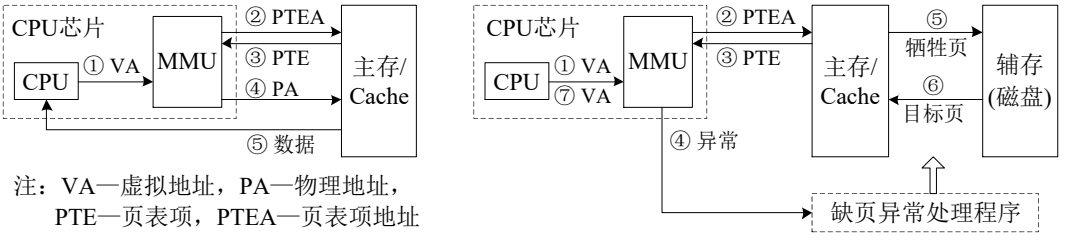


图 3.57　页式虚拟存储器的访存操作过程

缺页处理的方法是，从主存找出一个空闲页框或选出一个牺牲页框，若牺牲页已被修改，则先将它写到磁盘中；然后调入目标页到主存的指定页框中；最后返回到原来的进程继续执行，即重新执行导致缺页的指令。

缺页时的虚存-辅存地址变换，由操作系统通过软件来实现，此处不再展开讨论。

4．快表的组织

由图 3.57(a) 可见，CPU 每次按虚拟地址访存时，最少需要两次访问主存，即地址变换和数据访问，与 CPU 按物理地址访存相比，访存性能成倍下降，若不解决地址变换的访存问题，虚拟存储器就没有实用意义。

为了减少地址变换的访存次数，现代 CPU 都利用层次结构的思想，在 MMU 中设置了页表的缓冲存储器，称为后备转换缓冲区（Translation Lookaside Buffer，TLB）。由于 TLB 在 CPU 中，TLB 又称为快表，主存中的页表相应地称为慢表。

TLB 是一个小容量的、用虚拟地址访问的高速缓存，每个条目（entry, 项）存放由一个页表项组成的块，条目类似于 Cache 中的行。由于虚存-主存采用全相联映射方式、写回法写策略，因此，某 TLB 条目被替换时，对应的页是否被改写等信息必须写回到页表中。

为了提高命中率，减少访存次数，TLB 都具有较高的相联度（如 8 路组相联映射），采用 LRU 替换算法及写回法写策略。下面是一个 TLB 条目的结构示例，其中，用底纹标出的是由页表项组成的信息块，其余的是 TLB 条目的管理信息，TLB 标记的位数等于虚页号位数减去 TLB 组号的位数。

有效位	TLB 标记	LRU 位	脏位	物理页号	修改位	…

TLB 的容量应该设置多大？这取决于它的命中率有多高。我们来看一个例子，假设页大小为 4KB，CPU 每次按 4B 大小连续访问同一页内所有信息时，地址变换的命中率为 1023/1024。可见，TLB 的命中率远远高于 Cache，因此，TLB 的大小通常只有几十个条目。与 Cache 一样，TLB 也由 SRAM 组成。

设置了 TLB 后，MMU 进行地址变换时，首先访问 TLB，如图 3.58 所示，其中的 VPN 为虚页号。TLB 命中时，访存操作过程如图中第④步、第⑤步所示；TLB 缺失时，才访问主存中的页表，从图 3.57(a) 的第②步开始进行地址变换，若存在页缺失，还需要启动图 3.57(b) 的过程。可见，TLB 命中时，地址变换无须访存。

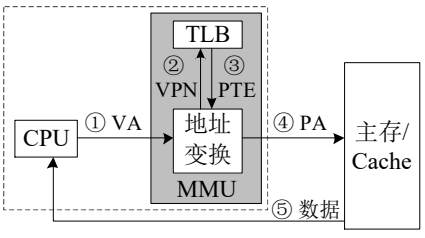


图3.58　TLB命中时的访存操作过程

可见，虚拟存储器由操作系统及 MMU 实现，操作系统负责虚拟存储器的主存管理、缺页处理等工作，MMU 主要负责地址变换、虚存保护等工作。

5．虚存及 Cache 访问举例

某计算机中，主存按字节编址，主存地址为 12 位；L1 Cache 采用直接映射方式，主存块大小为 4B，共 16 个行；页式虚存的虚拟地址为 14 位，页面大小为 64B，TLB 采用 4 路组相联映射方式，共 16 行。

因此，页内地址为 $\log_2(64B/1B)=6$ 位，虚拟地址由虚页号 VPN（8 位）、页内地址 VPO（6 位）组成，物理地址由实页号 PPN（6 位）、页内地址 PPO（6 位）组成。

页表按虚页号 VPN 进行索引，共有 $2^8=256$ 个页表项（PTE），页表项至少有装入位、实页号 PPN 两个字段。

TLB 按虚页号 VPN 进行访问，共有 16/4=4 个组，组号（TLBI）为 $\log_2 4=2$ 位，VPN 中的高 8−2=6 位用作 TLB 项的标记（TLBT）。

L1 Cache 的块内地址（CO）为 $\log_2(4B/1B)=2$ 位，行号（CI）为 $\log_2 16=4$ 位，主存地址中的高 12−2−4=6 位用作 Cache 行的标记（CT）。

这个由 TLB、L1 Cache 组成的小存储系统，某一时刻的状态如图 3.59 所示。其中，页表只列出了前 16 个页表项，无效栏用破折号标出。

组号	标记	PPN	有效	标记	PPN	有效	标记	PPN	有效	标记	PPN	有效
0	03	—	0	09	0D	1	00	—	0	07	02	1
1	03	2D	1	02	—	0	04	—	0	0A	—	0
2	02	—	0	08	—	0	06	—	0	03	—	0
3	07	—	0	03	0D	1	0A	34	1	02	—	0

（a）TLB：4 路组相联，共 16 行

VPN	PPN	V
00	28	1
01	—	0
02	33	1
03	02	1
04	—	0
05	16	1
06	—	0
07	—	0

VPN	PPN	V
08	13	1
09	17	1
0A	09	1
0B	—	0
0C	—	0
0D	2D	1
0E	11	1
0F	0D	1

（b）页表：前 16 个 PTE

行	标记	V	块数据(0~3)			
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03

行	标记	V	块数据(0~3)			
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
10	2D	1	93	15	DA	3B
11	0B	0	—	—	—	—
12	12	0	—	—	—	—
13	16	1	04	96	34	15
14	13	1	83	77	1B	D3
15	14	0	—	—	—	—

（c）L1 Cache：直接映射，共 16 行

图 3.59　小存储系统在某一时刻的状态

假设 CPU 发出地址为 00 0011 1101 0100B 的读请求，参考图 3.58，小存储系统的处理过程（第②~⑤步）如下：

②MMU 从虚拟地址中抽出 VPN（00001111B）、VPO（010100B），用它来查 TLB。

③TLB 处理时，从 VPN 中抽出 TLBT（000011B）、TLBI（11B），结果是第 3 组的第 1 行命中，PPN 为 0DH（001101B）。

④MMU 用 PPN、VPO 构造出物理地址（001101 010100B），用它来访问 L1 Cache。

⑤Cache 处理时，从物理地址抽出 CT（001101B）、CI（0101B）及 CO（00B），由于第 5 行的有效位为 1、标记与 CT 匹配，故 Cache 命中，读出第 0 个字节（36H），将它返回给 CPU，访存请求完成。

地址变换的过程与虚拟地址的值有关，可能有不同的路径，如图 3.57 中的 TLB 缺失或缺页。

习题 3

1. 解释以下概念或术语。

（1）RAM、SAM、DAM、ROM　　（2）存取时间、存取周期、存储器带宽

（3）时间局部性、空间局部性　　（4）存储元、存储单元、存储阵列

（5）SRAM、DRAM、SDRAM、DDR SDRAM，EEPROM、Flash

（6）行刷新、刷新周期，集中式刷新、分散式刷新、异步式刷新，DMAC

（7）主存地址空间、CPU 可寻址空间　（8）同步传输、突发传输

（9）顺序编址、交叉编址　　（10）交叉访问、并行访问

（11）命中时间、缺失率、缺失开销　（12）行、标记、有效位、目录表

（13）RAND、FIFO、LRU　　（14）直接映射、全相联映射、组相联映射

（15）全写法、写回法　　（16）按写分配法、不按写分配法

（17）哈佛结构　　（18）可变分区、页式管理，页框，MMU

（19）虚拟地址、物理地址　　（20）虚页号、实页号，页表、页表基址

（21）段式、页式、段页式　　（22）TLB

2. 存储器层次结构中，不同存储器的容量、速度应满足什么要求？计算机中有哪些基本的存储层次？原因是什么？每个层次的目标是什么？

3. RAM 芯片为什么常采用双译码方式？为什么要设置片选引脚？SRAM 芯片为什么要求地址信号先于 $\overline{CS}$ 建立、后于 $\overline{CS}$ 撤销？读操作、写操作的时序有什么不同？

4. 16K×4 位 SRAM 芯片的引脚有哪些？

5. 读 DRAM 存储元的速度比读 SRAM 存储元慢的原因有哪些？

6. DRAM 芯片为什么要设置 $\overline{RAS}$、$\overline{CAS}$ 引脚，为什么没有 $\overline{CS}$ 引脚？

7. DRAM 芯片为什么需要刷新？行刷新对 DRAM 芯片组成有什么要求？刷新周期的信号时序是什么？芯片刷新如何实现？

8. 某 DRAM 芯片的存储阵列有 1024 行，存取时间为 0.5μs，存储元的最大刷新周期为 2ms，芯片刷新一遍共需要多少时间？若 CPU 在 1μs 内至少要访问一次，请选择你认为较合理的芯片刷新方式，并说明理由，以及两次刷新的时间间隔。

9. SRAM 芯片和 DRAM 芯片各有哪些特点？各适用于什么场合？

10. 主存地址空间、主存地址引脚个数、CPU 可寻址空间之间有什么关系？

11. 若用 16K×4 位 SRAM 芯片构成 16K×16 位 SRAM 模块，请回答下列问题：

（1）需要 SRAM 芯片多少片？

（2）各个芯片在 SRAM 模块中的地址范围、存储单元中的位置各是多少？

（3）各个芯片片选线的有效逻辑是什么？

（4）画出 SRAM 模块的信号线与内部各个芯片引脚的连接图。

12. 若用 4K×16 位 SRAM 芯片构成 16K×16 位 SRAM 模块，回答与题 11 相同的问题。

13. 现有 64K×4 位 ROM 芯片、32K×8 位 SRAM 芯片若干，要求构成 256K×8 位

的存储模块，前 64KB 为 ROM 空间，画出存储模块的信号线与内部各个芯片引脚的连接图。

14. 若用 16K×4 位 DRAM 芯片构成 16K×16 位 DRAM 模块，请回答下列问题：

（1）需要 DRAM 芯片多少片？

（2）各个芯片在 DRAM 模块中的地址范围、存储单元中的位置各是多少？

（3）画出 DRAM 模块的信号线与内部各个芯片引脚的连接图。

15. 若用 4K×16 位 DRAM 芯片构成 16K×16 位 DRAM 模块，回答与题 14 相同的问题。

16. 根据所学内容，DRAM 控制器应有哪些功能？

17. 某计算机中，CPU 有 20 根地址引脚（A_{19}～A_0）、8 根数据引脚（D_7～D_0），控制引脚由 $\overline{ADS}$、IO/$\overline{M}$ 及 R/$\overline{W}$ 组成。若配置 256KB 的主存（SRAM），放在 CPU 可寻址空间的低端。请回答下列问题：

（1）主存的地址线、数据线各有多少位？

（2）主存连接到 CPU 时，其片选信号 $\overline{CS}$ 的有效逻辑是什么？画出主存各引脚与 CPU 的连接图。

18. 有哪些方法可以减小存储周期或平均存储周期，其原理是什么？又有哪些方法不改变存储周期或平均存储周期，但可以提高存储器的带宽，其原理是什么？

19. 现有 4K×16 位 SRAM 芯片若干，要求构成 16K×16 位的多体交叉 SRAM 模块，画出 SRAM 模块与内部各个芯片引脚的连接图。

20. 某 4K×8 位 SRAM 芯片的存储周期为 200ns，该芯片的带宽是多少？由 4 个上述芯片构成的 16K×8 位多体交叉存储器，若采用交叉访问方式，则访问 64 个字节最少需要多少个存储周期？启动各存储体轮流工作的时钟频率是多少？

21. 为什么 Cache 按主存地址进行访问？若 Cache 与主存之间不支持突发传输方式，以块为信息交换单位时，可以减小平均访问时间吗？请说明理由。

22. 某计算机中，主存按字节编址，CPU 有 20 根地址引脚、8 根数据引脚，配置有 64KB 的 Cache，Cache 与主存采用直接映射方式，主存块大小为 16B。回答下列问题：

（1）为了实现映射，主存地址应该如何划分？各个字段分别为多少位？

（2）每个 Cache 行的标记为多少位？说明理由。

（3）若访存地址分别为 2D058H 和 2D078H，Cache 命中时的标记分别是多少？

23. 将题 22 中 Cache 改用全相联映射方式，回答与题 22 相同的问题。

24. 将题 22 中 Cache 改用 4 路相联映射方式，回答与题 22 相同的问题。

25. 某 2 路组相联 Cache 有 4 个行，采用 LRU 替换算法，主存块大小为 8 个字。假设 Cache 初态为空，CPU 先从地址 0000H 起升序连续访问 48 个字，再从地址 002FH 起降序连续访问 48 个字，每次访问 1 个字，求此时的 Cache 命中率。

26. 若题 25 中 Cache 改用全相联映射方式，回答与题 25 相同的问题。

27. 提高相联度通常会提高命中率，但并不总是这样。对于采用 LRU 替换算法的 2 路组相联映射 Cache，以及相同容量的直接映射 Cache，请给出一个访问的块地址序列，使得前者的命中率比后者低。

28. 某计算机的存储器按字节编址，主存地址空间为 24 位，配置有 4MB 的主存，主存块大小为 32B，Cache 有 256 个行，采用 4 路组相联映射、LRU 替换算法、写回法写策略，Cache 行的管理信息至少有多少位？

29. 某 Cache-主存层次中，Cache、主存的存取周期分别为 30ns、150ns，Cache 读/写一个主存块的时间为 600ns。CPU 执行某程序时，共有 700 次读操作、300 次写操作，其中，读操作缺失 50 次，写操作缺失 15 次。若忽略块替换带来的损失时间，请分别计算 Cache 采用全写法、写回法写策略时的平均访问时间。

30. 早期的主存空间分配有哪两种方式？各有什么特点？

31. 主存-辅存层次能否构成虚拟存储器的标志是什么？

32. 虚存的管理表为什么都放在主存中？又为什么用段号或虚页号进行索引？

33. 假设主存按字节编址，主存地址为 32 位，逻辑地址由 8 位段号及 32 位段内地址组成，段式虚拟存储器中，段表最多有多少行？只考虑地址变换的实现，段表项至少有多少位？

34. 假设主存按字节编址，主存地址为 20 位，逻辑地址为 24 位，页式虚拟存储器中，页大小为 8KB，页表最多有多少行？只考虑地址变换的实现，页表项至少有多少位？若 TLB 采用 2 路组相联映射，共 8 个条目，TLB 条目至少有多少位？

35. 基于图 3.59 所示的存储系统状态，若读请求的地址为 0294H，写出存储系统处理访存请求的过程。