第3章 存储系统

本章要求读者正确理解存储系统的基本概念、二级存储层次结构，掌握主存的组成结构和扩展技术；理解 Cache 的工作原理和性能指标，掌握主存和 Cache 之间的地址映像，以及地址转换方法，并能使用多种替换算法实现块的替换；正确理解虚拟存储器和辅助存储器的基本概念、存储器管理方式及工作原理。

3.1 存储系统概述

存储系统是计算机系统的重要组成部分之一，用来存储程序和数据。有了存储系统，计算机就有了记忆能力，从而能自动地从存储系统中取出存储的指令并按一定的顺序进行操作。计算机中所用的存储部件有多种类型，如寄存器、静态 RAM、动态 RAM、闪存、磁盘存储器、磁带存储器、光盘存储器等，它们各自有不同的速度、容量和成本，各类存储器按照层次化的方式构成了存储系统的层次化结构。

3.1.1 存储器的性能指标

存储器的性能指标主要有存储容量、存取时间、存取周期、价格和带宽等。在存储器中，一般将 8 个二进制位定义为 1 字节，也称为字节存储单元，其地址称为字节地址。如果计算机是按字节存储单元进行寻址工作的，则称其为按字节编址的计算机；如果计算机是按字存储单元（多字节）进行寻址工作的，则称其为按字编址的计算机。目前，多数计算机均支持这两种编址方式。存储器的主要性能指标如下：

（1）存储容量是以字节（或字）为单位来表示存储器存储单元的总数，即存储容量 S= 存储字节（或字）数×存储字长度（位数）。存储器的地址线位数决定了存储器的可直接寻址的最大空间。计量存储空间常用的单位词头有 K（表示 2^{10}）、M（表示 2^{20}）、G（表示 2^{30}）、T（表示 2^{40}）等。

（2）存取时间又称为存储器访问时间，是指从启动一次存储器操作到完成该操作所经历的时间。

（3）存取周期指连续启动两次独立的存储器操作（如连续两次读或写操作）所需间隔的最小时间。通常，存取周期≥存取时间。存取时间和存取周期反映了存储器的速度。

（4）存储器的价格通常以每位价格来衡量，即：

$$P（每位价格）= C（总成本）/S（总容量）$$

（5）存储器带宽是单位时间存储器所能存取的信息量，通常以位/秒（b/s）或字节/秒（B/s）为单位，可以表示为 B_M（数据传输速率）$=W$（数据宽度）$/T_M$（存取周期）。

其他的性能指标还有可靠性、存储密度、存储的长期性、功耗、物理尺寸（集成度）等。

3.1.2 存储器的分类

按照存储器的存储介质、性能及使用方法的不同，存储器有如下分类方法：

（1）按存储介质分类。目前使用的存储介质主要有半导体器件、磁性材料和光学方式存储器件。采用半导体器件构成的存储器称为半导体存储器，如主存；采用磁性材料构成的存储器称为磁表面存储器，如磁盘存储器；采用光学方式存储器件构成的存储器有光盘存储器。

（2）按读/写信息可改性分类。有些半导体存储器存储的内容是固定不变的，即只能读出不能写入，这种半导体存储器称为只读存储器（ROM）；既能读出又能写入的半导体存储器称为随机存取存储器（RAM）。

（3）按信息存取可保存性分类。断电后信息即消失的存储器称为非永久记忆的存储器，如随机存取存储器（RAM）；断电后仍能存储信息的存储器称为永久性记忆的存储器，如只读存储器（ROM）、磁盘存储器和光盘存储器。

（4）按信息存取方式分类。按信息存取方式分类可分为随机存取存储器、顺序存取存储器、直接存取存储器和相联存储器四种形式。

① 随机存取存储器。随机存取存储器（RAM）是由按地址访问的半导体存储器构成的，只要给出存储器某个地址就可以读/写存储器单元中的内容。在计算机中，随机存取存储器通常用于高速缓冲存储器（Cache）或主存。

② 顺序存取存储器。顺序存取存储器的特点是信息按顺序存储和读出，其存取时间取决于信息的存储位置，以记录块为单位编址。例如，磁带存储器就是采取顺序存取方式工作的。

③ 直接存取存储器。直接存取存储器的存取方式是首先采用随机访问的方式来寻找信息所在的区域，然后按顺序方式进行存取信息。例如，磁盘存储器就是采用直接存取方式进行工作的。

④ 相联存储器。上述三类存储器都是按所需信息的地址来访问，但有些情况下可能不知道所访问信息的存储地址，只知道要访问信息的内容特征。此时，只能按内容检索到存储地址后再进行读/写。这种存储器称为按内容访问存储器或相联存储器。例如，存储系统中的快表 TLB（转换旁路缓冲存储器，详见 3.5.3 节）通常是采用相联存储器构成的。

（5）按功能分类。

① 高速缓冲存储器。目前高速缓冲存储器（Cache）由静态 RAM 芯片组成，位于主存和 CPU 之间，存取速度接近 CPU 的运算速度，用来存储 CPU 经常使用到的指令和数据。

② 主存储器。指令直接面向的存储器是主存储器，简称主存。CPU 执行指令时给出的存储器地址是主存地址（在虚拟存储系统中，需要将指令给出的逻辑地址转换成主存地址）。因此，主存是存储器分层体系结构中的核心存储器，用来存储系统的启动程序及其数据，主存目前一般由 MOS 型半导体存储器构成。

③ 辅助存储器。在系统运行时，直接与主存交换信息的存储器称为辅助存储器（简称辅

存)。目前，通常将磁盘存储器或光盘存储器作为辅存，辅存中存储的内容需要调入主存后才能被 CPU 访问。由于辅存通常是在计算机主板的外部，因此也被称为外部存储器(简称外存)。

3.1.3　存储系统的层次化结构

计算机对存储系统的要求是容量大、速度快、成本低。但是，目前只使用一种存储器是很难同时兼顾这三个方面的要求的。

在实际应用中，为了满足存储器容量大、速度快和成本低这三个方面的要求，通常采用三种不同类型的存储器，即高速缓冲存储器（Cache)、主存和辅存，从而形成一个层次化的二级存储结构。其中一个层次是由主存和 Cache 构成的高速缓冲存储系统，另一个层次是由主存和辅存构成的虚拟存储系统。前一层次主要解决存储系统的速度问题，后一个层次主要解决存储系统的容量问题。其中主存与 Cache 之间的数据交换是由硬件自动完成的，主存与辅存之间的数据交换是由硬件和操作系统共同完成的。中央处理单元（CPU）可以直接访问主存和 Cache，但不能够直接访问辅存。辅存中的数据必须调入主存后，才能由 CPU 进行处理。存储器层次结构示意图如图 3-1 所示。

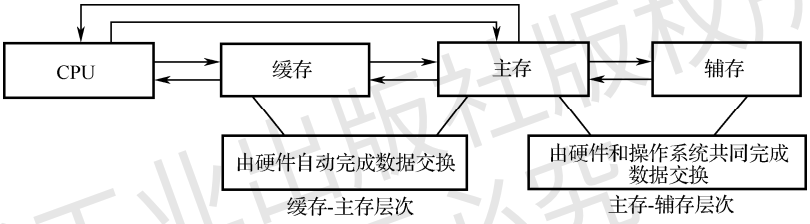


图 3-1　存储器层次结构示意图

为了提高计算机的处理速度，常利用 Cache 来临时存放部分指令和数据。与主存相比，Cache 的存取速度快，但存储容量小。主存用来存放计算机运行期间的大量程序和数据，在主存与 Cache 之间交换数据和指令是按存储块进行的，而在 Cache 与 CPU 之间交换数据是按字进行的。

目前，最常使用的辅存是由磁表面存储器（如硬盘存储器）和光盘存储器构成的，其特点是存储容量大，通常用来保存系统程序和大型数据文件及数据库。在主存和辅存之间交换数据是按页或段进行的。存储系统的各个层次如图 3-2 所示。

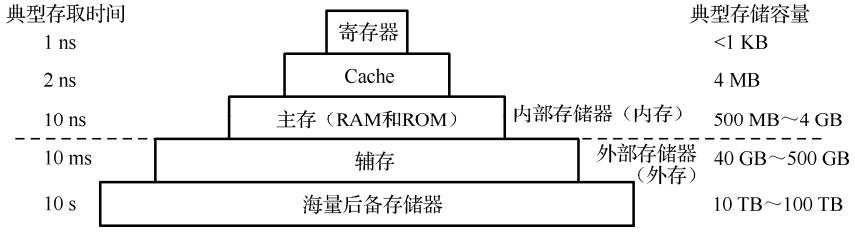


图 3-2　存储系统的各个层次

3.2 主存储器

目前，计算机的主存储器一般是由半导体存储器构成的，由于通常被放置在计算机的主板上，因此又被称为内部存储器（简称为内存）。

3.2.1 主存储器概述

主存储器的功能结构如图 3-3 所示。其中，虚线框内的存储器地址寄存器（MAR）和存储器数据寄存器（MDR）在 CPU 芯片内。实线框内的地址译码器、读/写控制电路和存储块（Memory Bank，MB）均制作在存储芯片中，存储芯片与 CPU 通过地址总线、数据总线和控制信号连接。

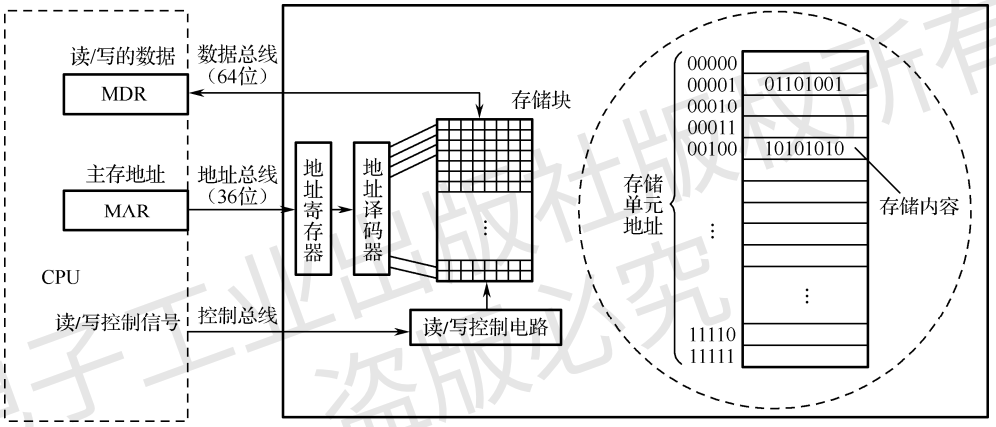


图 3-3　主存储器的功能结构

在主存储器（主存）中，由存储单元构成的存储阵列是存储器的核心部分。为了读取存储块中的数据，必须对存储单元进行编号，所编的号码就是地址。存储单位是指具有相同地址的那些位元构成的一个单位，可以是 1 个字节或 1 个字。对各存储单元进行编号的方式称为编址方式，可以按字节编址，也可以按字编址。现在大多数通用计算机都采用按字节编址的方式，此时存储块内一个地址中有 1 个字节。许多专用于科学计算的大型计算机采用 64 位编址，这是因为科学计算中数据大多是 64 位浮点数。

在图 3-3 中，实线框为主存。主存通过 M 位地址总线（如 32 位）、N 位数据总线（如 64 位）和控制总线同 CPU 交换数据。M 位地址总线用来指出所需访问的存储单元的地址（访问地址最大为 2^M），N 位数据总线用来在 CPU 与主存之间交换数据，控制总线用来协调和控制 CPU 与主存之间的读/写操作。当 CPU 启动一次读/写主存的操作时，先将地址码由 CPU 通过地址总线送入存储地址寄存器（MAR），然后使控制总线中的读/写控制信号有效，MAR 中地址码经过地址译码器后选中该地址对应的存储单元，通过读/写控制电路即可完成主存的读/写操作。

地址译码器的功能是接收地址总线上的地址信号，进行地址译码后选中存储块中的某个

存储单元。从存储芯片上具有地址译码器的数量来区分，有单译码和双译码两种方式。例如单译码方式的地址线有 12 根，译码后输出 4096 根选择线可寻找 4096 个单元。因此，单译码方式一般应用于小容量的存储器，双译码方式中一般分行、列两个方向的译码器。例如，双译码方式的地址线有 12 条，其中行、列方向各 6 条，经行、列两个译码器的输出选择线只需要 64+64=128 条，同样可对应寻址 64×64=4096 个存储单元。因此，双译码方式适合大容量的存储器，这种方式也被称为矩阵译码器。

从工作特点、作用和制作工艺的角度来看，主存主要由半导体随机存取存储器（RAM）和半导体只读存储器（ROM）组成。其中 RAM 在程序执行过程中可读可写，故一般用于存储用户程序。ROM 在程序执行过程中只能读出，因此一般用于存储部分系统程序（如计算机中的输入/输出导引程序 BIOS 等）。

在计算机中，主存一般是由多个半导体存储芯片经扩展后组成的，这些存储芯片与 CPU 连接既可按字节编址也可按字编址。例如，一个存储容量为 16 MB 的存储器，按字节编址的寻址范围是 0～16×1048576（16M）；如果按照 16 位字长（2 字节）编址则寻址范围为 0～8M；按照 32 位字长（4 字节）编址的寻址范围为 0～4M。在微机中，16 位存储字或 32 位存储字的地址是 2 个或 4 个存储单元中最低端的字节存储单元的地址。

3.2.2　半导体随机存取存储器

半导体随机存取存储器（Random Access Memory，RAM）中数据既能被读也能够被写，但是写入的数据在断电后会立即消失（俗称易失性存储器）。通常，RAM 按电路结构和存储原理又分为静态随机存取存储器（SRAM）和动态随机存取存储器（DRAM）两种结构形式。

SRAM 是采用触发器的工作原理来存储数据，由于其速度快、无须刷新等特点被用于 Cache。DRAM 是利用电容存储电荷的原理来存储数据的，所以要在指定的时间内需要内部刷新，否则所存数据就会丢失。由于 DRAM 的集成度高、功耗小、价格低，因此被广泛用于计算机的主存（内存条）。

1．静态随机存取存储器

静态随机存取存储器（Static RAM，SRAM）是利用半导体触发器的两个稳定状态表示 1 和 0 的。由最简单的 TTL 电路组成的 SRAM 是由两个双发射极晶体管和两个电阻构成的触发器电路，而由 MOS 管组成的单极型 SRAM 是由 6 个 MOS 管组成的双稳态触发电路。SRAM 的特点是只要保持供电，写入 SRAM 的数据就不会消失，不需要刷新电路。在读出时不会破坏原来存储的数据，一经写入后可多次读出。SRAM 的功耗较大、容量较小，但存取速度较快。

（1）SRAM 的工作原理。例如，存储容量为 1K×4 位的 Intel 2114 SRAM 芯片的基本单元电路是由 6 个 MOS 管组成的，Intel 2114 SRAM 芯片的结构如图 3-4 所示，其外部引脚如图 3-5 所示。

图 3-5 中 A_9～A_0 为芯片的 10 个地址输入端，其寻址长度为 2^{10}=1K；I/O_1～I/O_4 为芯片的 4 位数据输入/输出端；$\overline{CS}$ 为芯片选择线（简称片选线，低电平有效）；$\overline{WE}$ 为芯片写允许信号线（低电平有效）；V_{CC} 为芯片电源端；GND 为芯片接地端。

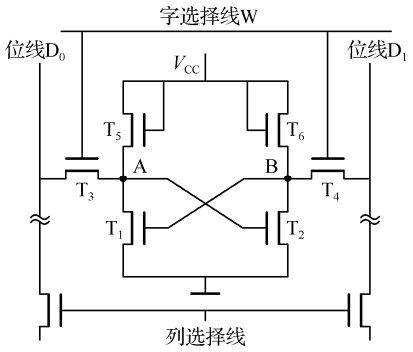


图 3-4　Intel 2114 SRAM 芯片的结构

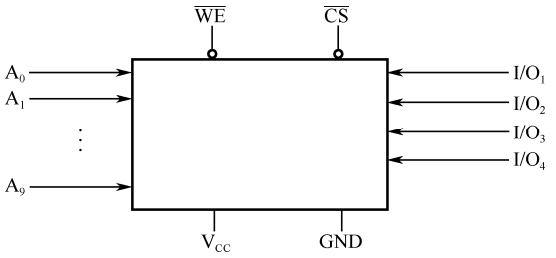


图 3-5　Intel 2114 SRAM 芯片的外部引脚

当需要对 Intel 2114 SRAM 芯片中的某个存储单元进行读/写操作时，首先要输入对应存储单元的地址，以便选中该存储单元。此刻若进行读操作，则$\overline{CS}$为低电平，$\overline{WE}$为高电平，这样可在读/写控制电路的输入/输出端 I/O_1～I/O_4输出该存储单元中存储的 4 位数据。若进行写操作时，则可将写入数据送至 I/O_1～I/O_4，并使$\overline{CS}$为低电平、$\overline{WE}$为低电平，这 4 位数据可写入该地址对应的存储单元。

（2）SRAM 芯片的读/写时序。在与 CPU 连接时，CPU 的时序与存储器的读/写时序之间的配合是非常重要的。对于所应用的 SRAM 芯片，其读/写时序是已知的。图 3-6 为 Intel 2114 SRAM 芯片的读时序图，图 3-7 为 Intel 2114 SRAM 芯片的写时序图。

① 读时序。图 3-6 中，t_{RC}为读周期时间，t_A为读出时间，t_{CO}为片选到数据输出延迟时间，t_{CX}为片选到输出有效的时间，t_{OTD}为从断开片选到输出变为三态时所需的时间，t_{OHA}为地址改变后的维持时间。从读时序可以看出：在给出有效地址后，经过译码电路、驱动电路的延迟，到读出所选存储单元中的数据，再经过 I/O 电路延迟后在外部数据总线上稳定地输出读数据信息，这一过程总共需要 t_A时间，所以称 t_A为读出时间。

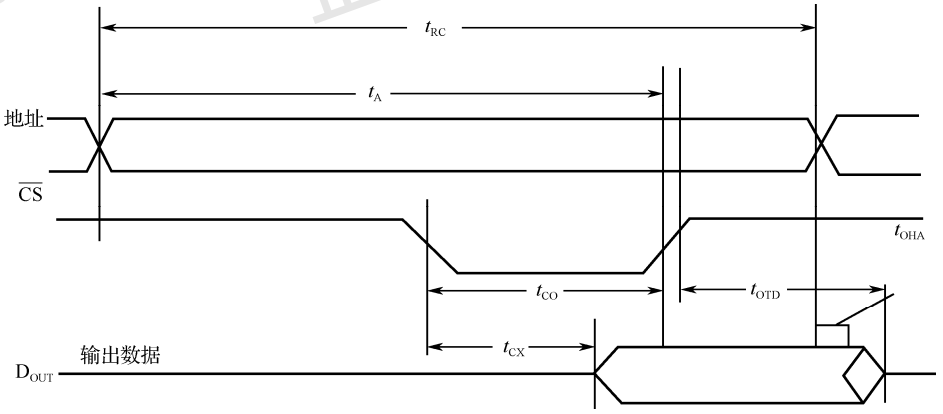


图 3-6　Intel 2114 SRAM 芯片的读时序图

读周期时间与读出时间是两个不同的概念，读周期时间 t_{RC} 表示芯片进行两次连续读操作时所必须间隔的时间，一般情况下 $t_{RC} \geqslant t_A$。显然，CPU 读取存储器中的数据时，从给出有效地址起，只有经过 t_A时间才能在数据总线上可靠地输出数据。而连续的读操作必须保证间隔时间要达到 t_{RC}，否则存储器就无法正常工作，CPU 的读操作就会失败。

② 写时序。在图 3-7 中，t_{WC} 为写周期时间，t_W 为写入时间，t_{WR} 为写恢复时间，t_{DTW} 是从写信号有效到输出三态时的时间，t_{DW} 是数据有效时间，t_{DH} 是写信号无效后数据保持时间。如果要芯片进行写操作，则必须要求 $\overline{CS}$ 和 $\overline{WE}$ 都为低电平。为了要使数据总线上的数据能够可靠地写入存储器，要求 $\overline{CS}$ 与 $\overline{WE}$ 相“与”的宽度至少要保持 t_W。

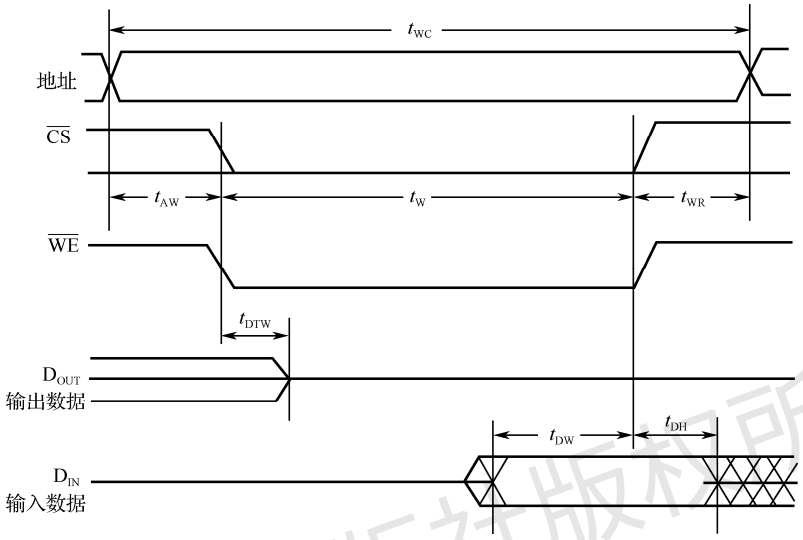


图 3-7　Intel 2114 SRAM 芯片的写时序图

为了保证在地址变化时不会发生写入错误而破坏存储器中的数据，$\overline{WE}$ 在地址变化时必须为高。只有在地址有效后再经过 t_{AW} 时间后，$\overline{WE}$ 才能有效。只有 $\overline{WE}$ 变为高电平后再经过 t_{WR} 时间，地址信号才允许改变。为了保证有效数据的可靠写入，要求 $t_{WC}=t_{AW}+t_W+t_{WR}$。为了保证在 $\overline{WE}$ 和 $\overline{CS}$ 变为无效前可以把数据可靠地写入存储芯片，要求写入的数据必须在 t_{DW} 以前保证在数据总线上已经稳定。

③ 存取周期。存取周期是指存储器进行一次读/写操作所需要的时间，也就是存储器进行连续的读/写操作的最短间隔时间。存取周期应等于访问时间加上下一次存取开始前所要求的附加时间，一般用 T_M 表示，即 $T_M=t_{RC}+t_{WC}$。

存储器的带宽 B 表示存储器被连续访问时可以提供的数据传输速率，通常用每秒传输数据的位数（或字节数）来衡量。

2．动态随机存取存储器

动态随机存取存储器（Dynamic RAM，DRAM）是利用 MOS 管的栅极对其衬底间的分布电容来存储数据的，以存储电荷的多少（即电容电压的高低）来表示 1 和 0。DRAM 的每个存储单元一般是由单个或者 3 个 MOS 管组成的，因此 DRAM 的集成度较高、功耗较低。但其缺点是存储在 DRAM 中的数据会随着电容漏电而逐渐消失，其数据一般存取时间为几毫秒（ms），这样就需要每隔 1～2 ms 对 DRAM 中所有的存储单元进行一次恢复充电（也称为刷新），因此采用 DRAM 的计算机需要配置存储器刷新电路。另外，DRAM 的存取速度比 SRAM 慢，但其容量大、集成度高。计算机中的内存条通常采用 DRAM。

（1）单管存储单元。单管存储单元的内部结构如图 3-8 所示。

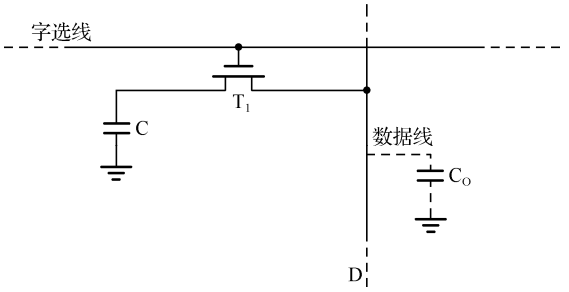


图 3-8　单管存储单元的内部结构

单管存储单元的工作过程如下：对某个单管存储单元进行写操作时，字选线为高电平，T_1 导通，写入的数据由数据线存入电容 C 中；进行读操作时，字选线为高电平，存储在电容 C 上的电荷通过 T_1 输出到数据线上，通过读出放大器即可得到存储的数据。单管存储单元的优点是线路简单、速度快，但是在读出时会损失电荷，故需要立即对单管存储单元进行“重写”以恢复原数据；单管存储单元的读出信号很小，要求使用高灵敏度的读出放大器。

（2）DRAM 的存储原理。Intel 2164 是一种存储容量为 64K×1 位的 DRAM 芯片，现以该芯片为例来介绍 DRAM 的内部结构及工作原理。图 3-9 是 64K×1 位 DRAM 的结构框图，其内部存储单元是由单个 MOS 管构成的。在大容量的 DRAM 中，为了减少地址总线的封装引脚数，一般采用行、列两个译码器（矩阵）方式，这样地址码需要分成两次送入存储器。64K×1 位的 DRAM 有 16 位地址码，CPU 将地址码分两批（每批 8 位）送至存储器，即先送行地址后送列地址。行地址由行地址选通信号 $\overline{RAS}$ 送入，列地址由列地址选通信号 $\overline{CAS}$ 送入。Intel 2164 DRAM 芯片内部具体是由 4 个 128 行×128 列存储器阵列组成的，其中行地址 A_7 与列地址 A_7 选择 $I/O_{1\sim4}$ 控制电路之一和 4 个存储器阵列之一。在进行读操作时，读出放大器输出存储单元中的数据，同时又使其内部的数据自动恢复，所以读出放大器也称为再生放大器。

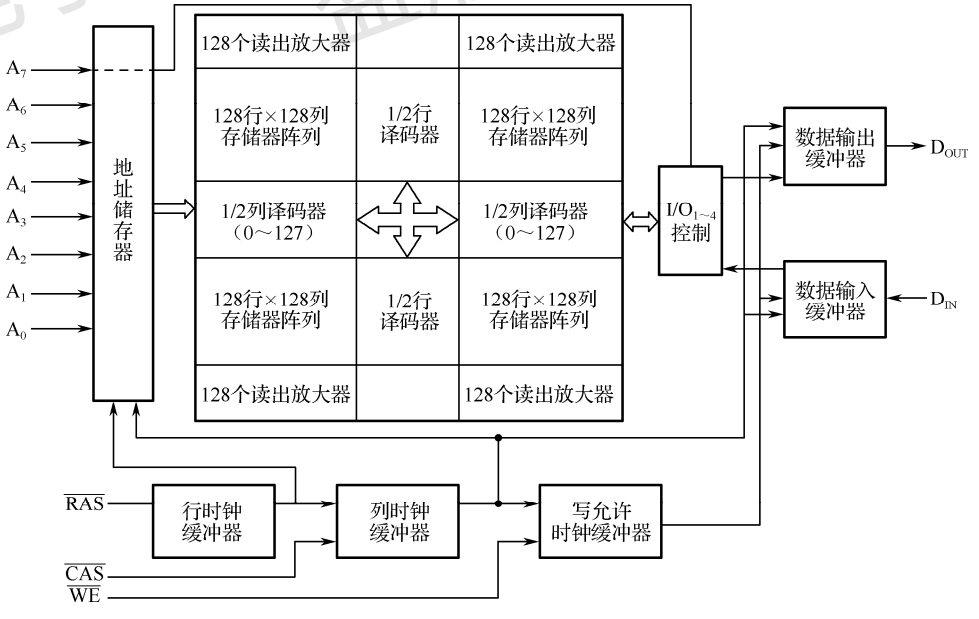


图 3-9　64K×1 位 DRAM 结构框图

由于 DRAM 的每行都有独立的读出放大器，因此只要在一段时间内对存储器阵列的每一行上的存储单元同时进行读操作，就可完成了对存储器的刷新。刷新逻辑电路可以保证 MOS 管 DRAM 的刷新工作，通过对每行的定时刷新，会使 MOS 管 DRAM 中的数据不丢失。所以 DRAM 的读出过程又称为刷新周期。DRAM 可采用集中式刷新和分布式刷新两种刷新方式。

① 集中式刷新是指在一个刷新周期内，利用一段固定的时间依次对存储器的所有行逐一进行刷新。这种刷新方式的缺点是在刷新期间内 CPU 不能访问存储器，因此会影响计算机系统的正常工作。

② 分布式刷新是指在规定的时间内分散地将 DRAM 的所有行都刷新一遍，具体做法是用刷新周期除以行数，得到两次刷新操作之间的时间间隔 t，再利用刷新逻辑电路每隔 t 时间产生一次刷新请求，这些刷新逻辑电路一般集成在 DRAM 芯片中。例如，在计算机中每隔 15.6 μs 刷新逻辑电路就发出一次刷新请求，DRAM 的存储器阵列内部由 128 行组成，则全部刷新一遍的时间为 2 ms（128 个刷新周期）。

（3）DRAM 芯片技术的发展。目前经常使用的 DRAM 的类型有同步 DRAM（SDRAM）和双数据率 SDRAM（DDR SDRAM）。其中，SDRAM 与 CPU 之间的数据传输是同步的，它的读/写周期为几纳秒，采用 64 位数据读/写方式。DDR SDRAM 允许在时钟脉冲的上升沿和下降沿传输数据，数据传输速率比 SDRAM 更高，可达数 10^6 位/秒（Gb/s）。

例如，DDR3 SDRAM 芯片内部 I/O 缓冲可以进行 8 位预取。如果存储芯片内部 CLK 时钟的频率为 200 MHz，意味着存储器总线上的时钟频率应为 800 MHz，存储器总线在每个时钟内可传输两次数据，若每次传输 64 位数据，则对应存储器总线的最大数据传输速率（即带宽）为 200 MHz×8×64/8=800 MHz×2×64/8=12.8 GB/s。

3.2.3　半导体只读存储器和 Flash 存储器

半导体只读存储器（Read-Only Memory，ROM）的特点是用户在使用时只能读出其中的数据，不能修改和写入新的数据。如果 ROM 中的数据是由制造厂商在生产 ROM 时写入的，则这种 ROM 称为掩膜 ROM（Masked ROM）。此外，ROM 还有以下几种类型：

① 可编程 ROM（Programmable ROM，PROM）。PROM 中的程序和数据是由用户通过编程器自行写入的，但一经写入就无法更改，它是一次写入多次读出的 ROM。

② 可擦除可编程 ROM（Erasable Programmable ROM，EPROM）。EPROM 中的程序和数据也是由用户通过编程器自行写入的，写入后的程序和数据可由紫外线灯照射擦除，EPROM 可多次擦除多次写入。

③ 电擦除可编程 ROM（Electrically Erasable Programmable ROM，EEPROM）。EEPROM 是在线采用专用程序进行擦除和改写的存储器，其特点是使用方便，芯片不离开插件板便可擦除或改写其中的数据但 EEPROM 的存取速度较慢、价格较贵。

④ 闪速存储器（Flash Memory）。闪速存储器简称闪存，也称为 Flash 存储器，借鉴了 EPROM 结构简单的特点，又吸收了 EEPROM 电擦除的特点，本身具有可以整块芯片电擦除和部分单元电擦除的功能。另外，闪存还具有耗电低、集成度高（容量大）、体积小、可靠性高、无须后备电池（在不加电情况下，数据可存储 10 年之久）、可重新改写、重复使用性好（至少可反复使用百万次以上）等优点。闪存的访问时间可低至几十纳秒

（ns），比硬盘驱动器快近百倍，由于没有机械运动部件，所以抗振能力强。闪存使用先进的 CMOS 制造工艺，目前广泛应用于计算机的 PC 卡存储器（固态硬盘）以及用来存储主板和显卡上的 BIOS。利用闪存制成的“优盘”（又称 U 盘）已广泛用来替代软盘，成为移动式存储器。

3.2.4 主存与 CPU 的连接及主存容量的扩展方式

1. 主存与 CPU 的连接原理

在主存与 CPU 相连时，特别要注意两者之间的地址总线、数据总线、读/写控制线和片选线的连接。

① 地址总线的连接：由于 CPU 的地址总线数往往要比主存的地址总线数多，通常是将 CPU 地址总线的低位部分与主存的地址总线相连，而 CPU 地址总线的高位部分在扩展主存时作为片选信号或悬空。

② 数据总线的连接：在实际中，CPU 的数据总线位数一般多于或者等于主存的数据总线位数。如果主存的数据总线位数与 CPU 的数据总线位数不相等，则必须扩展主存数据总线位数，使其数据总线位数与 CPU 的数据总线位数相等。

③ 读/写控制线的连接：CPU 的读/写控制线一般可直接与主存的读/写控制端相连，通常高电平为读，低电平为写。在实际中，有些 CPU 的读/写控制线是分开的，此时 CPU 的读控制线应与主存的允许读控制端连接，CPU 的写控制线应与存储芯片的允许写控制端相连接。

④ 片选线的连接：在扩展主存时，片选线的连接是 CPU 与主存连接的关键。如果主存是由多个存储芯片扩展而成的，那么哪一个存储芯片会被选中则完全取决于该存储芯片的片选控制 $\overline{\text{CS}}$ 是否能接收到来自 CPU 的片选有效信号。

另外，有些存储芯片的片选有效信号还与 CPU 的访问控制信号 $\overline{\text{MREQ}}$（低电平有效）有关，只有当 CPU 要求访问时，才要求选择存储芯片。若 CPU 访问 I/O 接口，则 $\overline{\text{MREQ}}$ 为高电平，表示不要求存储器工作。此外，在采用全译码扩展方式时，由于 CPU 的地址总线多于主存的地址总线，CPU 的高位地址还会与一些逻辑部件来共同产生储器芯片的片选信号。

合理选择存储芯片主要是指存储芯片类型的选择。通常，ROM 用于存储系统程序、标准子程序和各类常数等，RAM 用于存储 CPU 所需的数据。此外，在考虑存储芯片数量时，要尽量使其连线简单、方便。

2. 主存容量的扩展方式

主存的最基本访问单元是由 8 位半导体存储器构成的，在实际应用中，单个存储芯片往往无法满足计算机主存系统容量的要求，通常可采用位扩展、字（地址长度）扩展或字、位同时扩展方式将多个存储芯片组织和扩展起来构成主存。

（1）位扩展方式。当单个存储芯片的地址长度与主存的地址长度相同，而各存储单元中所存数据的位数少于主存所要求的位数时，可采用位扩展的方式。

例 3.1 假设主存总容量为 16K×8 位，而所选用的存储芯片容量为 16K×4 位时，主存应由 2 个存储芯片采用位扩展方式来构成。位扩展方式连接图如图 3-10 所示。如果主存容量仍

为 16K×8 位，而所选用的存储器芯片容量为 16K×1 位，那么主存就需要由 8 个存储芯片采用位扩展方式来构成。

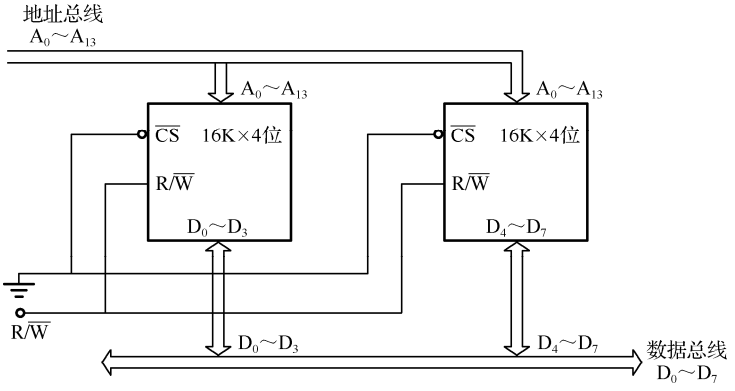


图 3-10　位扩展方式连接图

当 CPU 访问主存时，由 CPU 的地址总线（A_{13}～A_0）、读/写允许信号 R/$\overline{W}$ 并行连接到每个存储芯片，各存储芯片内部的片选信号 $\overline{CS}$ 也要并行连接，这样就能选中每个存储芯片中的同一地址，可从该地址中同时读出 8 位数据并送至数据总线，或将数据总线上的 8 位数据同时写入各存储芯片的同一地址中。

（2）字扩展方式。当存储芯片内部存储单元中的位数与主存中存储单元中的位数相同而地址范围（字数）小于主存的要求时，可以采用字扩展方式。

例 3.2　假设主存的容量为 64K×8 位，而所选用的存储芯片容量为 16K×8 位时，则主存应由 4 个存储芯片采用字扩展方式来构成。字扩展方式连接如图 3-11 所示。

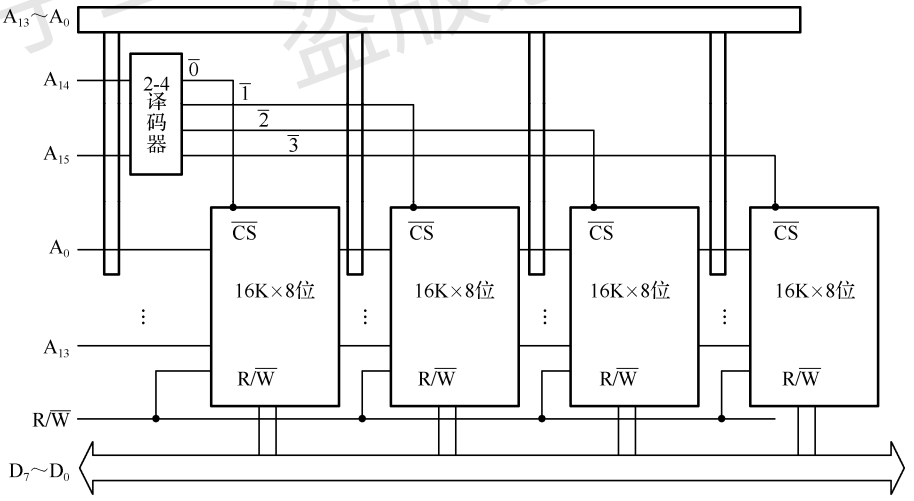


图 3-11　字扩展方式连接图

从图 3-11 中可以看到，由于每个存储芯片本身只有 14 条地址总线（A_{13}～A_0），所以将 CPU 输出的 16 条地址总线分成两部分：其中低端部分 A_{13}～A_0 直接与 4 个存储芯片内部的 14 条地址总线相连，剩余高端部分的 2 条地址总线（A_{15} 和 A_{14}）经 2-4 译码器译码后分别形

成 4 个存储芯片的片选信号 $\overline{CS}$。这样使用 4 个存储芯片构成的主存系统就包含了 16 KB 的地址区域，第 1 个到第 4 个存储芯片的地址范围分别是 0000～3FFFH、4000～7FFFH、8000～BFFFH、C000～FFFFH。

当 CPU 访问主存时，给定的任何地址码（通过地址总线 A_{15}～A_0 给出）都会位于指定的某一个存储芯片中，至于选中的是哪个存储芯片则由片选信号（高端地址）经 2-4 译码器译码后来确定，所读出和写入的 8 位数据都来自同一个存储芯片。

如果主存容量仍为 64K×8 位，而所选用的存储芯片容量为 8K×8 位，那么主存应由 8 个存储芯片采用字扩展方式构成。由于 8K×8 位存储芯片内部地址共 13 位（A_{12}～A_0），所以高端地址（片外地址）变成 3 位（A_{15}～A_{13}），这样就需要使用 3-8 译码器来完成片选功能。在字扩展方式中，增加存储芯片会扩展主存的地址范围，主存扩展范围的大小取决于存储芯片本身的容量和数量。

（3）字、位同时扩展方式。当存储器芯片的字数（地址范围）和字长（位数）均不能满足主存容量的要求时，就要采用字、位同时扩展方式来构成主存。例如，主存的容量是 $M\times N$ 位，而存储芯片的容量是 $L\times K$ 位，那么主存共需要 $M/L\times N/K$ 个存储芯片。

例 3.3 某主存容量为 64K×8 位，所选存储器芯片的容量为 8K×4 位，主存应由 64/8×8/4=16 个存储芯片扩展构成。字、位同时扩展方式连接图如图 3-12 所示。

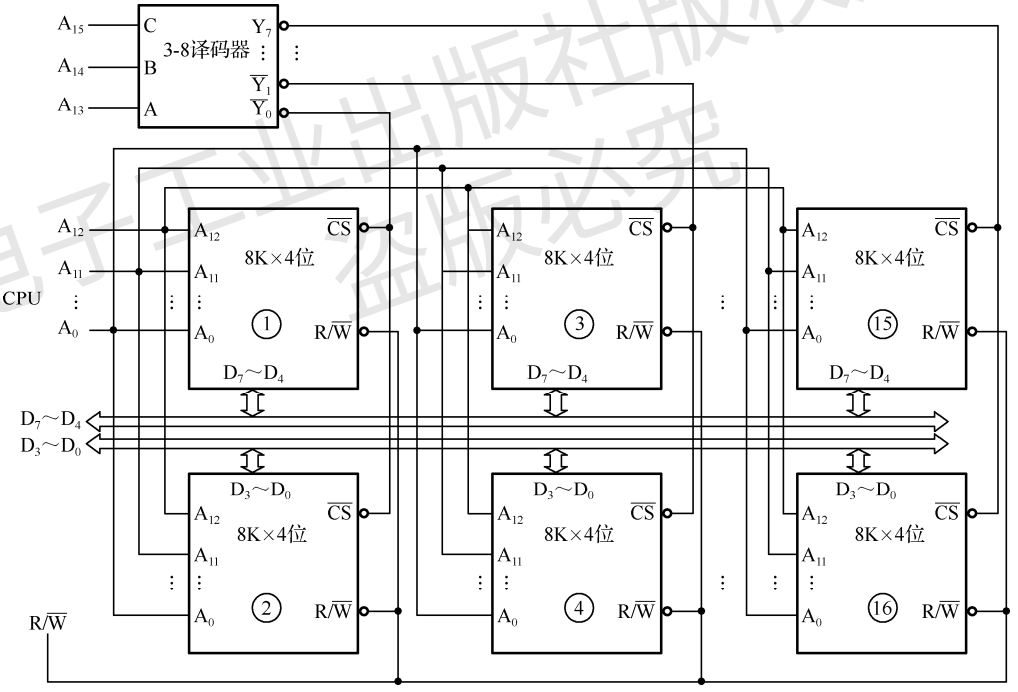


图 3-12　字、位同时扩展方式连接图

从图 3-12 中可看出，由于每个存储芯片只有 4 位，所以位方向需要由 2 个存储芯片扩展为 8 位。又由于每个存储芯片内只包含 8K 个字，所以其字数方面需要由 8 个芯片才能扩展为 64K 个字。这样，采用字、位同时扩展方式共需要 2×8=16 个存储芯片。CPU 地址总线中低端的 13 位地址线（A_{12}～A_0）与 16 个存储芯片内部的 13 位地址总线直接相连，总地址线中高端的 3 位地址码（A_{15}～A_{13}）经 3-8 译码器译码后的 8 条片选信号线分别连接到 8 组存

储芯片的$\overline{CS}$端。其中，每组中的 2 个存储芯片的$\overline{CS}$端并接。这样主存就被分为 8 个组（区域），每个组内包含 8K 地址范围。任何时候读出或写入的 8 位数据分别对应于同一组内的 2 个存储芯片的同一地址（高 4 位、低 4 位）。

假如主存的容量仍然为 64K×8 位，而选用的存储芯片的容量为 32K×4 位，则主存就需要由 4 个存储芯片构成（即 2 个组，每个组内包含 2 个存储芯片）。

3．主存扩展中的译码方式

CPU 访问主存时需要给出相应存储单元的地址码，其长度取决于 CPU 可直接访问的最大存储空间。在扩展主存时，一般要将其地址码分为片内地址（低端地址）和片选地址（高端地址）两部分。例如，存储芯片容量为 8K×4 位或 8K×1 位，它们的片内地址（低端地址）相同，均为 13 位；高端的地址码为片选地址，经译码后产生连接存储芯片内的片选信号$\overline{CS}$，因此，译码只涉及片选地址部分（高端地址）。片选地址的译码方式有部分译码方式和全译码方式两种。

（1）部分译码方式。在实际使用的存储空间比 CPU 可访问的最大存储空间小，而且对其地址范围没有严格要求的情况下可采用部分译码方式。

例 3.4　CPU 提供的地址总线为 16 位，而实际使用的主存容量只为 16 KB，拟采用 4K×4 位的存储芯片和部分译码方式进行扩展。问：

① 应该采用何种扩展方式？使用多少个存储芯片？

② 画出扩展连接图，写出各组存储芯片的寻址范围。

解：① 需要采用字、位同时扩展方式，共需要 16/4×8/4=8 个存储芯片。

② 采用部分译码方式时，首先将 8 片存储芯片按照扩展需要分成 4 组（区），每组 2 片。采用部分译码方式的扩展连接图如图 3-13 所示（图中未画出数据线）。

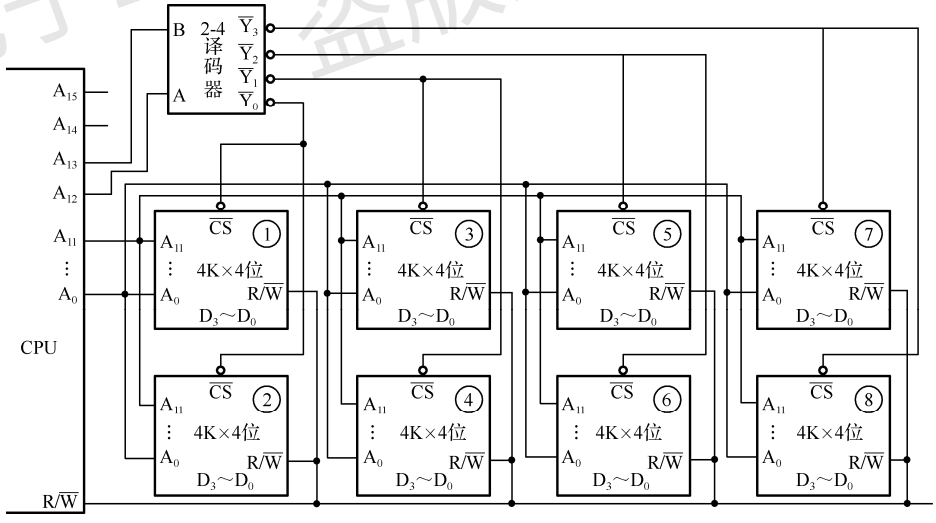


图 3-13　采用部分译码方式的扩展连接图

从图 3-13 中可以看出，在 CPU 的 16 条地址总线中，低端地址中的 12 位（A_{11}～A_0）用于片内地址，直接与 8 个存储芯片的地址总线并连；高端地址中的低 2 位地址（A_{13}和 A_{12}）用于片选地址，经译码后产生 4 个片选信号分别与 4 组芯片的$\overline{CS}$端相连；最高的 2 个地址

（A_{15} 和 A_{14}）悬空，没有参加译码。

采用部分译码方式，会使得各组芯片的地址范围不再是唯一的。以由存储芯片①、②构成的第一组为例，由于 CPU 地址总线中的最高 2 位地址悬空，因此每组存储芯片（4 KB）内的存储单元都对应有 4 个不同地址。部分译码方式地址分配范围如表 3-1 所示。

表 3-1　部分译码方式地址分配范围

片选地址		片内地址												译码输出	地址范围
$A_{15}A_{14}$	$A_{13}A_{12}$	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0		
00 00	00 00	00	00	00	00	00	00	11	11	11	11	11	11	$\overline{Y_0}=0$	0000H～0FFFH
01 01	00 00	00	00	00	00	00	00	11	11	11	11	11	11	$\overline{Y_1}=0$	4000H～4FFFH
10 10	00 00	00	00	00	00	00	00	11	11	11	11	11	11	$\overline{Y_2}=0$	8000H～8FFFH
11 11	00 00	00	00	00	00	00	00	11	11	11	11	11	11	$\overline{Y_3}=0$	C000H～CFFFH

同样，其他三组存储芯片的地址范围分别如下：

第 2 组存储芯片的地址范围为：1000H～1FFFH、5000H～5FFFH、9000H～9FFFH、D000H～DFFFH。

第 3 组存储芯片的地址范围为：2000H～2FFFH、6000H～6FFFH、A000H～AFFFN、E000H～EFFFH。

第 4 组存储芯片的地址范围为：3000H～3FFFH、7000H～7FFFH、B000H～BFFFH、F000H～FFFFH。

可以看出，采用部分译码方式时使得各组存储芯片的地址范围出现了重叠，其重叠区的个数取决于没有参加译码的地址总线的位数。在例 3.4 中，2 位地址总线（A_{15}、A_{14}）没有参加译码，所以每组存储芯片都出现 4 个重叠区。

（2）全译码方式。全译码方式是指片选地址部分（高端地址）全部参加译码，以下两种情况通常需要采用全译码方式。

① 主存实际地址使用的存储空间与 CPU 可访问的最大存储空间相同。

例如，CPU 的地址总线 16 位（A_{15}～A_0），即可访问的最大存储空间为 64 KB，而实际要求设计的存储空间同样为 64 KB。如果存储芯片的容量是 16K×4 位，那么采用字、位同时扩展方式时需要 8 个存储芯片，具体分为 4 个组（区），各存储芯片的片内地址是 14 位。地址总线的片选地址（高端地址）为 2 位，这 2 位片选地址全部参加译码时可产生 4 个片选信号，分别用于 4 组存储芯片的片选信号。这 4 组存储芯片的地址范围分别为 0000H～3FFFH、4000H～7FFFH、8000H～BFFFH、C000H～FFFFH。

② 主存实际使用的存储空间小于 CPU 可访问的最大存储空间，并且对实际存储空间的地址范围有严格的要求（如每个存储单元要求具有唯一的地址）。

例 3.5　CPU 的地址总线为 16 位（A_{15}～A_0），即可访问的最大存储空间为 64 KB。而系统中实际使用的存储空间只有 8 KB，且存储芯片的容量为 4K×2 位，并要求这 8 KB 的地址范围必须在 4000H～5FFFH 内，问：

① 应该采用何种扩展方式？使用多少个存储芯片？

② 画出扩展连接图，写出各组存储芯片的寻址范围。

解：① 需要采用字、位同时扩展方式，共需要 8/4×8/2=8 个存储芯片。

② 采用全译码方式时，首先需要将 8 个存储芯片按照扩展分成 2 组（区），每组 4 个存储芯片。采用全译码方式的扩展连接图如图 3-14 所示（图中未画出数据总线和读/写控制线）。

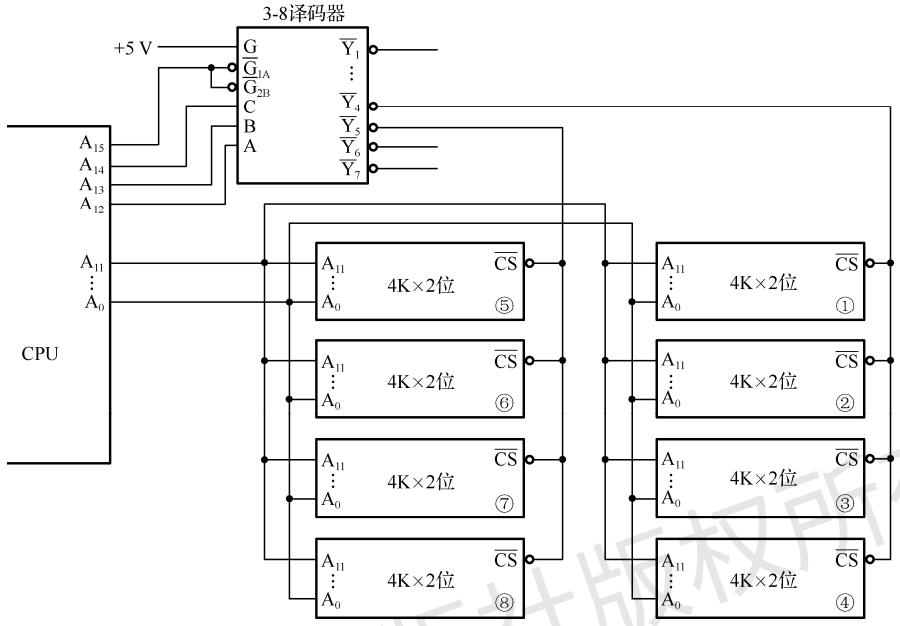


图 3-14　采用全译码方式的扩展连接图

从图 3-14 中可看出，全译码方式地址分配范围如表 3-2 所示。从表 3-2 中可看出，当前使用的存储空间的地址范围被严格地定义在 4000H～5FFFH 范围内。

表 3-2　全译码方式地址分配范围

存储芯片	片选地址				片内地址												译码输出	地址范围
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0		
①②③④	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_0$ =0	0000H～0FFFH
	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1		
	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_1$ =0	1000H～1FFFH
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1		
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_2$ =0	2000H～2FFFH
	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1		
	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_3$ =0	3000H～3FFFH
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
⑤⑥⑦⑧	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_4$ =0	4000H～4FFFH
	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1		
	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_5$ =0	5000H～5FFFH
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1		
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_6$ =0	6000H～6FFFH
	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1		
	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	$\overline{Y}_7$ =0	7000H～7FFFH
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		

以上两种情况均属于全译码方式，它们的共同特点是所使用的存储芯片内部所有存储单元的地址都是唯一的。另外，除了全译码方式和部分译码方式，还有线译码方式。在线译码方式中，可以将 CPU 中的高端地址直接连接在被扩展存储芯片的片选地址上，CPU 的低端地址直接与被扩展存储芯片的片内地址相连。这种方式的特点是连接方便、简单，但是会造成存储地址空间不连续，浪费地址资源，不利于以后的扩展。

4．内存条和内存条插槽

受集成度和功耗等因素的限制，单个存储芯片的容量不可能很大，往往需要通过存储芯片的扩展技术，将多个芯片集成在一个主存模块（如内存条）上，然后由多个主存模块，以及主板或扩充板上的 RAM 和 ROM 组成计算机所需的主存空间，再通过系统总线和 CPU 相连。

目前，计算机中的内存条通过内存条插槽内的引线连接到主板上，再通过主板连接到北桥芯片或 CPU。计算机中有多条总线可同时进行数据传输，支持两条总线同时进行数据传输的内存条插槽称为双通道内存条插槽，还有三通道、四通道内存条插槽，其总线的传输带宽可以分别提高到单通道的 2 倍、3 倍和 4 倍。

3.2.5　Pentium 计算机的主存系统组成

Pentium 计算机的数据总线为 64 位，地址总线为 36 位，对外的地址引脚为 A_{35}～A_3，使能信号为 BE_7～BE_0（8 字节），其地址总线比 Intel 486 计算机多了 4 位，但 A_{35}～A_{32} 并不作为物理地址使用，所以 Pentium 计算机对应的物理存储空间仍是 2^{32}= 4096 MB=4 GB。

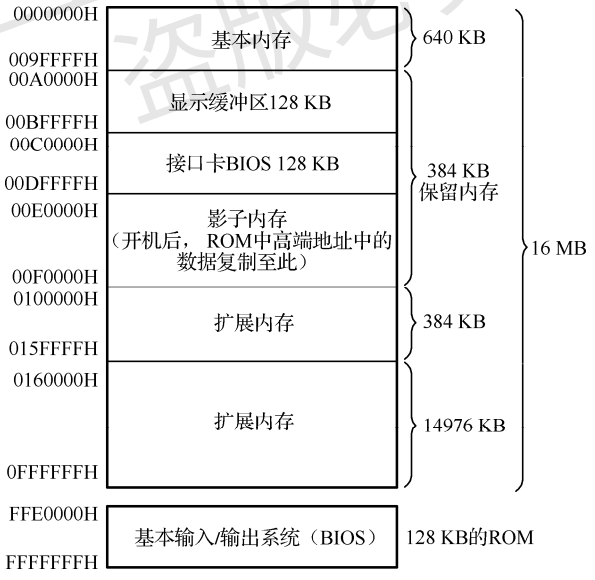


图 3-15　主存为 16 MB 的 Pentium 计算机物理存储空间的分配

考虑到系统软件的兼容性，除 128 KB 的系统程序区外，存储空间被分成基本内存、保留内存和扩展内存三部分。图 3-15 所示为主存为 16 MB 的 Pentium 计算机物理存储空间的分配。

3.3　高速缓冲存储器

在存储系统中，加入高速缓冲存储器（Cache）是为了解决 CPU 和主存之间速度不匹配而采用的一项重要技术。Cache 是介于 CPU 和主存之间的高速小容量存储器，其工作速度是主存的数倍，其内部全部功能由硬件实现，能高速地向 CPU 提供指令和数据，加快程序的执行速度。

3.3.1　Cache 简介

Cache 位于主存与 CPU 的通用寄存器组之间，新型的 CPU 芯片通常集成了 1～2 级 Cache，第 1 级 Cache 的容量一般只有几 KB 到几百 KB，第 2 级 Cache 的容量一般有几 MB，一些高端的 CPU 芯片甚至集成了第 3 级 Cache。Cache 用来存储当前使用最频繁的程序和数据，作为主存某些局部区域数据的副本，如存储现行指令地址附近的程序，以及当前要访问的数据区内容。

由于编程时指令地址基本上是连续的，对循环程序段的执行往往要重复若干遍，在一个较短的时间内，对存储器的访问大部分集中在一个局部区域中，这种现象称为程序的局部性。我们将这一局部区域的内容从主存复制到 Cache 中，可以使 CPU 高速地从 Cache 中读取程序与数据，其速度比从主存中读取要高 5～10 倍。这一过程由硬件实现，编程地址仍是主存地址，程序员看到的仍是访问主存。Cache 对用户是透明的，随着程序的执行，Cache 中的内容也会按一定的规则进行更新。

为了实现 Cache 的上述功能，需要解决以下几个问题：首先是主存和 Cache 之间的地址映像关系；其次是如何实现地址转换，将访问主存的地址转换成对应的 Cache 地址；然后是 Cache 的读/写方式；最后是更新 Cache 内容的替换算法。

1．Cache 的组成

和主存一样，Cache 也被分成若干个大小相同的存储块，每个存储块由若干字（或字节）组成。CPU 对主存和 Cache 的读/写以存储字（字）为单位，而主存与 Cache 之间的数据传输以存储块（简称块）为单位，一个块由若干定长的存储字（或字节）组成。CPU 在执行程序的同时，还要将用到的存储块从主存复制到 Cache 中，然后由 Cache 向 CPU 提供程序和数据。Cache 内部存储的总是部分主存内容的副本。

由于 CPU 在访问主存时，执行的指令中给出的是主存地址。在访问 Cache 时，必须知道被访问存储字的 Cache 地址。因此必须在 Cache 地址和主存地址之间建立一个确定的逻辑关系，从而可以将主存地址转换成 Cache 的地址，以便在 Cache 被命中时，能正确地在 Cache 中访问到对应的存储字。Cache 同主存之间的这种地址间的逻辑关系称为地址映像。通过地址映像后，才可以将主存地址转换为 Cache 的地址，这一过程称为地址转换。

反映主存单元和 Cache 单元的地址映像关系的表格称为地址映像表，该表采用高速器件实现，以便提高查表速度。为了使查表与访问 Cache 结合起来，地址映像表可以和 Cache 数据项结合起来，即在 Cache 中为每个存储块都增加一个地址映像标记。该标记可以在主存以

存储块的方式调入 Cache 时，将该存储块在主存中的地址作为标记写入 Cache 中的对应位置。当访问主存的地址与 Cache 中的这一地址映像标记相符时，表示所要访问的数据就在 Cache 中（被命中），同时也找到了数据在 Cache 中的存储位置。通常，为了识别一个 Cache 存储块中的数据是否有效，还要在标记中增加一个有效位。

由上述要求可知，Cache 包含两部分内容：其一是存储地址映像表的内容（标记区，包括存储块在主存中的块地址和有效位）；其二是信息块内地址。主存的地址也是由两部分组成的，其中高端地址称为主存块号地址，用于标识出每一个存储块；低端地址称为块内地址或偏移量，用于在块内寻址。由于主存和 Cache 的存储块的存储空间相同，所以主存的块内地址可直接作为 Cache 的块内地址。主存和 Cache 的编址结构如图 3-16 所示。

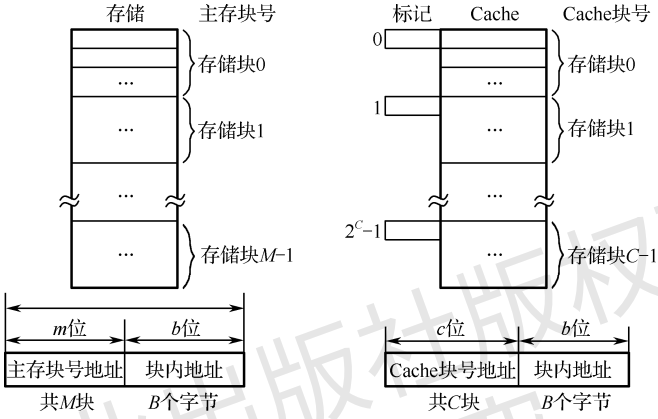


图 3-16　主存和 Cache 的编址结构

在图 3-16 中，主存被分为 $M=2^m$ 个存储块，存储块大小为 2^b 字节（或者字），这样主存地址的总位数为 $n=m+b$ 位。Cache 被分为 $C=2^c$ 个存储块（大写 C 表示存储块数，小写 c 表示位地址数），存储块大小为 2^b 个字节（或者字），Cache 地址的总位数为 k 位，$k=c+b$。

2．Cache 的工作原理

Cache 的存储容量较小，通常由快速的 SRAM 构成，直接集成在 CPU 芯片内，速度几乎与 CPU 一样快。在 CPU 和主存之间设置 Cache，其目的是把主存中被频繁访问的程序和数据复制到 Cache 中。由于程序访问的局部性，在大多数情况下，CPU 能直接从 Cache 中取得程序和数据，而不必访问主存。

（1）Cache 的有效位。在系统启动或复位时，Cache 的每行都为空，其中的数据无效，只有装入了主存的存储块后数据才有效。为了说明 Cache 每行中的信息是否有效，Cache 的每行都需要一个有效位。

通过将有效位清 0 可以“淘汰”Cache 某行中的主存存储块，称为刷新。装入一个新的主存存储块时，再将有效位置 1。

（2）CPU 访问 Cache 的过程。Cache 中存储的数据是主存中使用最频繁的若干存储块的数据副本，当 CPU 访问某主存单元时，先用该主存地址的块号地址去查询地址映像表，判定该主存地址存储单元的副本是否在 Cache 中。若在 Cache 中（称 Cache 被命中），经过地址转换可将主存的块号地址转换为 Cache 的块号地址，并且与块内地址一起生成访问 Cache 的地址。若被访问的主存地址存储单元的副本不在 Cache 中（称 Cache 未命中），这时 CPU 就需要

访问主存读取所需存储字，并同时还要将该存储字所在存储单元的存储块调入 Cache。此时如果 Cache 中还有空闲的存储空间，可直接将该存储块装入 Cache 中，同时还要将块号填入地址映像表中。如果 Cache 中的存储空间已经全部被占用，没有存储该存储块的地方，即块冲突，在这种情况下，需要按照某种替换算法替换 Cache 中的某一存储块的数据副本，以便将主存中的存储块装入 Cache 中，并修改地址映像表。CPU 访问 Cache 的过程如图 3-17 所示。

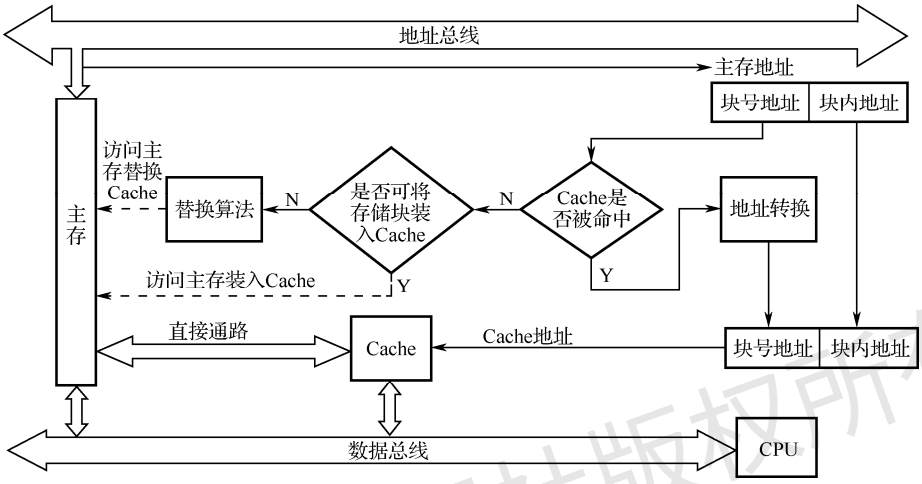


图 3-17　CPU 访问 Cache 的过程

注意：在 Cache 被命中时，首先通过地址转换将其主存地址转换成 Cache 地址；在 Cache 未命中时，不仅要将主存的存储块装入 Cache 中，同时还要将访问的存储字装入 CPU，这一切都是由硬件来实现的。

3．Cache 的性能指标

衡量 Cache 的性能指标不仅有存储芯片本身的各项指标，而且还有 Cache 的命中率、平均存取时间、访问效率和加速比等指标。

CPU 所要访问的数据在 Cache 中的比率称为命中率。设 N_c 表示在 Cache 中完成存取数据的总次数，N_m 表示在主存中完成存取数据的总次数，则命中率 h 为：

$$h = N_c/(N_c+N_m)$$

1−h 表示未命中率。若 t_c 表示 Cache 的存取周期，t_m 表示主存的存取周期，则 Cache、主存系统的平均访问时间 t_a 为：

$$t_a = ht_c+(1-h)t_m$$

Cache 的访问效率 e 为：

$$e = t_c/t_a$$

采用 Cache 后，设主存和 Cache 同时工作，对存储系统而言，其加速比为：

$$S_P = t_m / t_a$$

例 3.6　在 CPU 执行一段程序时，Cache 完成存取数据的次数为 1900 次，主存完成存取数据的次数为 100 次，已知 Cache 的存取周期 t_c 为 50 ns，主存的存取周期 t_m 为 250 ns，求带有 Cache 的主存系统的命中率 h、平均访问时间 t_a 和访问效率 e。

解：命中率为：

$$h=N_c/(N_c+N_m)=1900/(1900+100)=0.95$$

平均访问时间：

$$t_a=ht_c+(1-h)t_m=0.95\times50\ \text{ns}+0.05\times250\ \text{ns}=47.5\ \text{ns}+12.5\ \text{ns}=60\ \text{ns}$$

访问效率为：

$$e=t_c/t_a=50\ \text{ns}/60\ \text{ns}\approx83.3\%$$

3.3.2 Cache 的地址映像方式

主存和 Cache 之间的数据交换是以固定大小的存储块为基本单位整体进行的，因此，主存与 Cache 的存储空间都应划分成若干个大小相同的存储块。由于 Cache 的存储空间小而主存的存储空间大，因此在地址映像过程中，主存和 Cache 之间的存储块如何进行对应即成为一个关键的环节。在实际中，通常采用直接映像、全相联映像（Fully Associative Mapping）或者组相联映像三种方式来完成地址映像。

1．直接映像方式

例如，某计算机的主存和 Cache 都按字节编址，其主存的存储容量为 1 MB（地址总线为 20 位），按每个存储块按 512 B 划分，共被划分成 2048 个存储块，块号为 0～2047；Cache 的存储容量为 8 KB，每个存储块的大小也是 512 B，故 Cache 划分成 16 个存储块，块号为 0～15。

在直接映像方式中，首先根据 Cache 的存储容量对主存进行分组，要求主存各组的存储容量都等同于 Cache 的存储容量。在进行地址映像时，规定主存各组中的某一存储块只能映像到 Cache 中的一个固定的存储块中（即主存各组内的块号要与 Cache 的块号相同），这种对应关系称为直接映像方式，如图 3-18 所示。

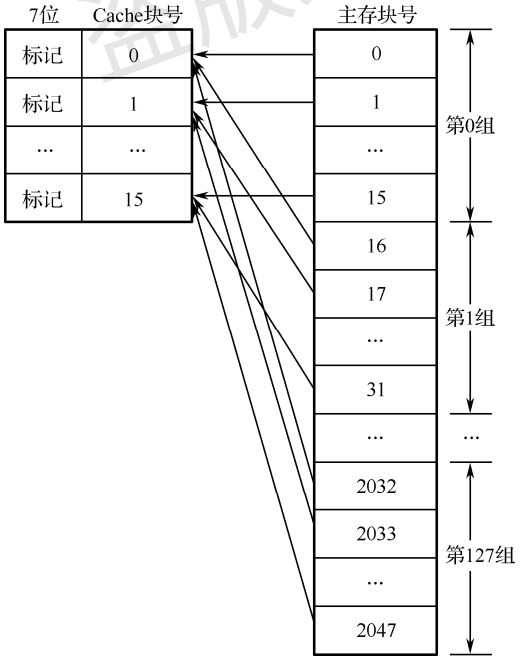


图 3-18　直接映像方式

直接映像方式具有下列对应关系：

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

其中，K 为 Cache 块号，J 为主存块号，c 为 Cache 块号的二进制代码位数，2^c 实际上就是主存每组内包含的存储块数量。

在图 3-18 中，主存共有 2048 个存储块，c=4，Cache 共 16 个存储块，主存再被划分 128 组，每组有 16 个存储块（与 Cache 的存储块数相同）。根据直接映像方式的规则可知，主存的第 J（全局序号）个存储块只能映像到与其组内序号（J 除以 16 的余数，即 $J \bmod 2^c$）相同的第 K 个 Cache 存储块。

这样，在直接映像方式下，主存的每个存储块只能被复制到某个固定的 Cache 存储块。基本映像规律是：将主存的 2048 个存储块按顺序分为 128 组，每组的 16 个存储块分别与 Cache 的 16 个存储块是一一对应的。具体而言，主存第 0 块、第 16 块、第 32 块、…、第 2032 块（共 128 个存储块），这些存储块的全局序号 J 与 16 相除以后得余数 K=0，故它们只能映像到 Cache 的第 0 个存储块。

同理，主存的第 1 块、第 17 块、第 33 块、…、第 2033 块（共 128 块），这些块的全局序号 J 与 16 相除以后得余数 K 为 1，故它们就只能映像到 Cache 第 1 个存储块。以此类推，主存其他的存储块在 Cache 中的映像位置，如主存的第 15 块、第 31 块、…、第 2047 块（共 128 块），也只能映像到 Cache 的第 15 个存储块。

当访问主存的数据时，CPU 会先给出一个 20 位的主存地址，其中地址的最高 7 位是主存分组后的组号（范围为 0～127）；随后的 4 位是组内的块号（范围为 0～15）；最后的 9 位是主存存储块中的字节序号（也称为块内地址，范围为 0～511）。因此，该 20 位的主存地址码在逻辑上就被分解成组号（7 位）+组内的块号（4 位）+块内的字节序号（9 位），其结构如图 3-19 所示。

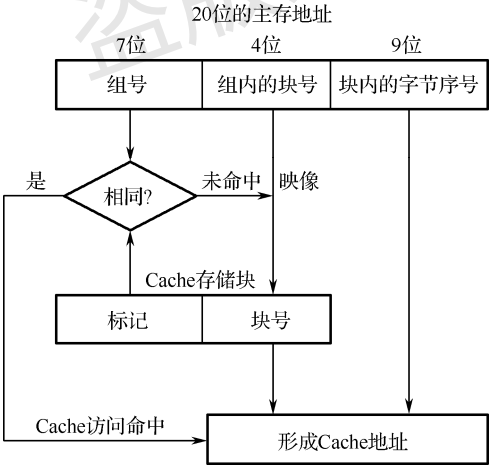


图 3-19　主存地址的结构

在具体映像时，主存的每组中都有一个存储块可以映像到 Cache 的同一存储块上，因而单靠主存地址分解得到的组内块号，只能确定该主存存储块在 Cache 中可能的位置，并不能确定该存储块确实已被映像到对应的 Cache 存储块。例如，主存第 0 组的第 0 块、第 1 组的第 0 块、第 2 组的第 0 块都可以映像到 Cache 的第 0 块上，假设当根据主存的 20 位主存地址

访问主存第 1 组的第 0 块时，该块的直接映像位置为 Cache 的第 0 块，但 Cache 的第 0 块就一定是主存第 1 组的第 0 块映像过来的吗？答案是不确定的，也可能是由主存第 0 组的第 0 块或者第 3 组的第 0 块映像过来的。

为了准确地判断 Cache 的某个存储块具体是由哪个主存存储块映像过来的，在 Cache 中为每个存储块设立一个 7 位的 Cache 组号标记，该组号标记与主存分组的组号对应。如果 Cache 的第 0 块是由主存第 16 块（全局序号）映像过来的，则该 Cache 存储块对应的组号标记位设置 1，用以表示当前的 Cache 存储块是由主存第 1 组中的某个存储块映像过来的。根据直接映像方式的规则，该主存存储块必须是第 1 组的第 0 块（组内序号）。因此在访问时，只需两步就可以确定 Cache 访问是否命中。

第一步，根据直接映像方式的规则确定主存存储块对应的 Cache 存储块。

第二步，比较主存地址中的高 7 位（组号）与 Cache 存储块的 7 位组号标记，如果两者相同，则表明主存存储块已被映像到对应的 Cache 存储块中了，Cache 被命中；否则，表示 Cache 未命中。

直接映像方式的优点是在硬件实现方面比较容易，只需容量较小的可按地址访问的组号标记存储器和少量的比较电路，硬件成本很低，映像速率快。其缺点是不够灵活、Cache 存储块的冲突概率很高，可能使 Cache 存储的存储空间得不到充分利用。

例如，需将主存第 0 块和第 16 块同时映像到 Cache 中时，由于它们都只能映像到 Cache 中的第 0 块，即使 Cache 的其他存储块空闲，也始终会有一个主存存储块不能被映像到 Cache 中，这会使 Cache 的命中率急剧下降。

例 3.7 假设某计算机的存储系统采用直接映像方式的 Cache，其存储容量为 8 KB，要求在每个存储块内存储 16 B；主存的存储容量是 512 KB，求：

① 该 Cache 地址结构是如何组成和具体分配的？

② 主存的地址结构是如何组成和具体分配的？

③ 主存第 513 个存储块存储在主存内的组号为多少？将其装入 Cache 后被存储的对应存储块号为多少？

④ 在③的基础上，当 CPU 输出数据的主存地址为 04011H 时，Cache 是否能被命中？如果未命中，则应存储在 Cache 内的哪个存储块中？

解：根据已知条件可知：

① Cache 的地址结构是由块号地址和块内地址组成的。已知存储容量 8 KB=8192 B=2^{13} B，字节地址位数是 13 位。划分的存储块有 8192B/16B=512 块=2^9 块，块号地址共 9 位。块内存储容量 16 B=2^4 B，块内地址（字节地址）共 4 位，即 Cache 的地址位数=块地址+块内地址=9+4=13 位。

② 主存地址结构包括组号地址、块号地址和块内地址。已知主存的存储容量 512 KB=2^{19} B，故按字节编址的总地址位数为 19 位，可被划分为 512 K/8 K=64 组=2^6 组，组号地址共 6 位。组内块号地址和块内地址的分配与 Cache 相同，分别为 9 位和 4 位，即主存地址=组号地址+块号地址+块内地址=6+9+4=19 位。

③ 主存第 513 块所在主存组号为[513/512]=1（从 0 编号），即组号标记为 000001。其中，[ ]表示取整运算。根据主存存储块与 Cache 存储块之间的对应存储关系可知，主存第 513 块映像 Cache 中的 513 mod 512 =1，即第 1 号存储块。主存第 513 块存储在第 1 组第 1 号存储块。或者将 513 转换成二进制数，即 513D=000 0010 0000 0001B，高 6 位为组号地址 000001，

随后 9 位是组内块号地址 0 0000 0001，表示存储在第 1 组第 1 号存储块。结果同上。

④ CPU 输出数据的 19 位主存地址为 04011H，即 000 0100 0000 0001 0001B，此地址所在的主存组号为 000010B（地址高 6 位），即存储在第 2 组。组内地址为 0 0000 0001B（随后 9 位，即第 1 块号）。而主存第 513 块在主存第 1 组第 1 块，由③可知要装入 Cache 中第 1 块，自然就不会命中。采用十进制数分析可知，输出的主存地址为 0401H，对应的十进制数为 1025，具体存储位置为 1025/512=2 余 1，即存储在主存的第 2 组第 1 块，结果同上，因此 Cache 未命中，Cache 中的第 1 块内容将被替换。本例计算过程如图 3-20 所示。

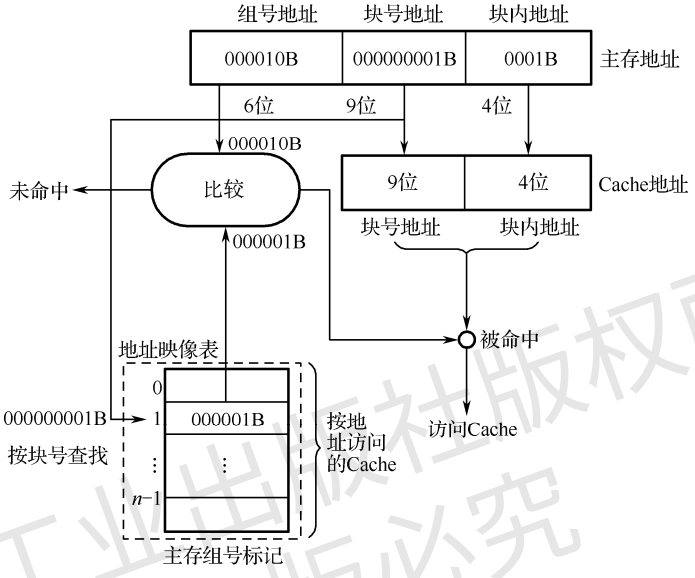


图 3-20　例 3.7 的计算过程

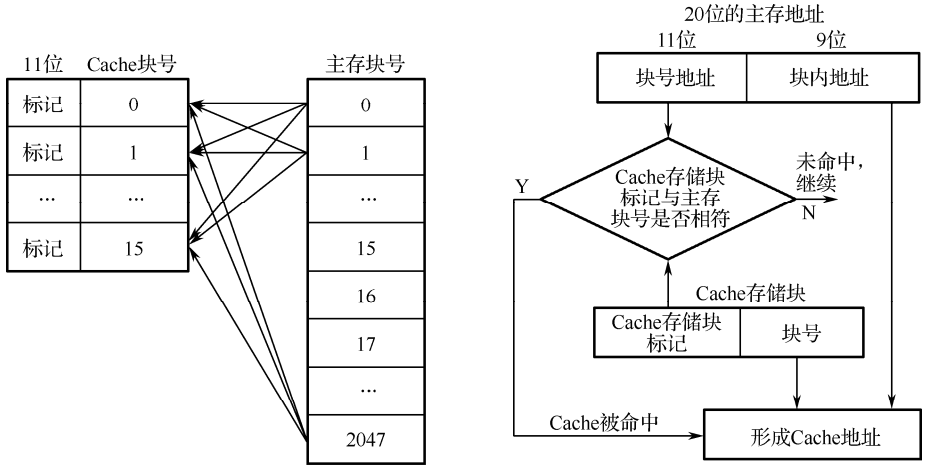
2．全相联映像方式

全相联映像方式的基本特征是主存和 Cache 均不进行分组，当主存与 Cache 进行数据交换时，主存中的每个存储块可以随机映像到 Cache 中的任意一个存储块。同样，假设某计算机的主存和 Cache 都按字节编址，主存的存储容量为 1 MB（主存地址为 20 位），按每个存储块 512 B 划分，共分成 2048 个存储块，块号为 0～2047；Cache 个存储块的容量为 8 KB，每个存储块的大小也是 512 B，故 Cache 分成 16 个存储块，块号为 0～15。全相联映像方式如图 3-21（a）所示。

采用全相联映像方式时，如果“淘汰”了 Cache 中某一存储块，则可装入主存中任何一个存储块，因而比直接映像方式更加灵活，但也存在一些严重的缺陷。

在全相联映像方式中，由于不存在存储块的分组，因此可以把 CPU 给出的 20 位主存地址看成两个部分，把高 11 位可看成主存的块号地址（0～2047），而把低 9 位看成块内地址（0～511），主存地址结构如图 3-21（b）所示。

由于 Cache 中的每个存储块都可由主存中的 2048 个存储块的任何一个映像过来，因此 Cache 中每个存储块的标记也需要 11 位（与主存块号对应），这样才能通过标记来确定 Cache 中的存储块是由主存中的哪个存储块映像过来的。因此，与直接映像方式相比，在全相联映像方式中 Cache 存储块的标记的位数会增加，从而导致其硬件比较逻辑的成本也有所增加。



（a）全相联映像方式　　（b）主存地址结构

图 3-21　全相联映像方式及主存地址结构

采用全相联映像方式时，Cache 存储块的冲突概率最低。只有当 Cache 存储块全部装入后才可能出现块冲突，因此这种方式的 Cache 利用率最高。

根据主存地址访问主存时，由于该地址所在的主存存储块可以被映像到 Cache 的任何一个存储块，因此要从 Cache 的第 0 块开始比较其标记与该主存存储块的序号，两者相同则表示 Cache 被命中，否则表示 Cache 未命中。最好情况是第一次比较就判定 Cache 被命中，最差的情况是从 Cache 的第 0 块开始，逐一比较全部 16 个存储块的标记，直到找到符合的标记（Cache 被命中），或者全部比较完后仍无符合的标记（Cache 未命中）为止，最终才能判断本次映像 Cache 是否被命中。因此，全相联映像方式的速率比直接映像方式慢，不能凸显缓存应有的高速性能。为了提高速率而把主存块号与各 Cache 存储块的标记进行比较，则比较电路会较复杂，硬件成本较高。

3．组相联映像方式

组相联映像的基本特征是主存和 Cache 均要分组，先将 Cache 分成若干组，每组若干个存储块（或称为 *n* 路）；再将主存分组，主存中每组包含的存储块数量与 Cache 划分的组数是一样的。

组相联映像就是指当主存与 Cache 之间以存储块为单位进行映像时，主存中的每一个存储块，只能映像到特定 Cache 组中的任意一个存储块，此时要求主存存储块的组内块号与 Cache 存储块所属的组号必须相等，如图 3-22（a）所示。

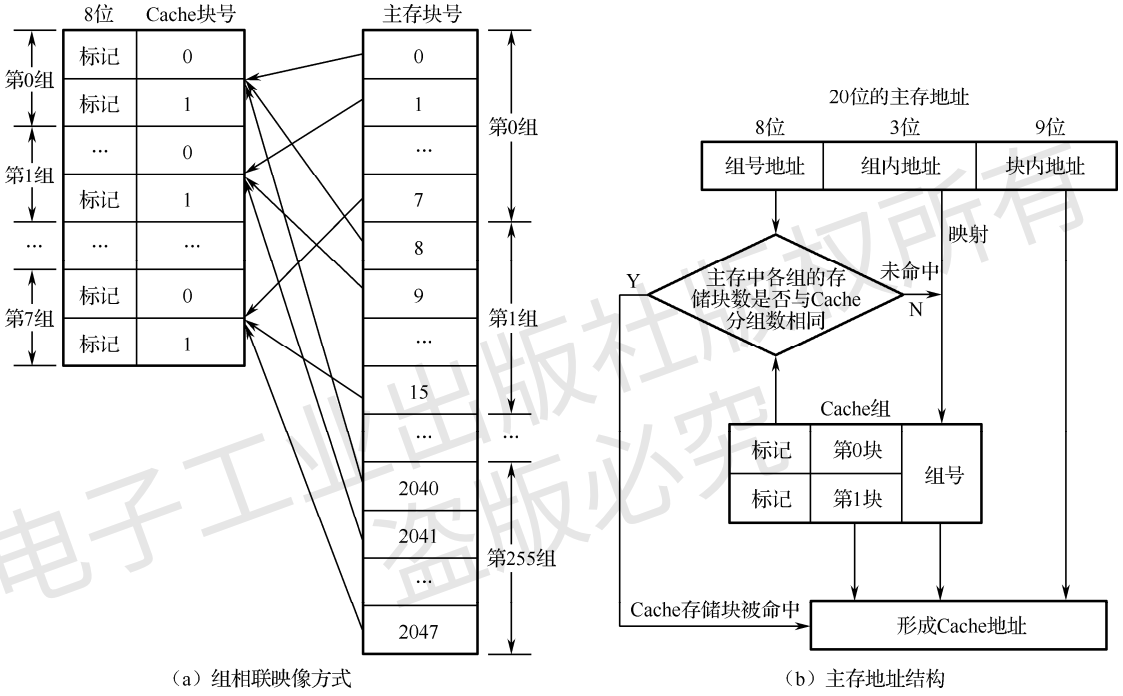
在 Cache 分组时，若每组包括 2 个存储块，则 Cache 为 2 路，相应的组相联映像方式称为 2 路组相联映像方式。同理，若 Cache 每组包括 4 个存储块，则相应的组相联映像方式称为 4 路组相联映像。

在实际情况中，通常根据速度和命中率来设计组相联映像方式的路数，且 Cache 对程序员是透明的，主存与 Cache 之间的地址转换和存储块替换都是采用硬件来实现的。

在直接映像方式中，要对主存进行分组，主存组内的各存储块只能映像到唯一的 Cache 存储块，两者之间存在固定的映像关系，且主存各组中均有一个存储块映像到某一个特定的 Cache 存储块。在全相联映像方式中，主存和 Cache 均不分组，两者之间以存储块为单位随

机映像，没有固定的映像关系。在组相联映像方式中，主存和 Cache 都会进行分组，主存每组内的各存储块只能映像到一个唯一的 Cache 组，但可以与 Cache 组内的存储块是随机映像的，没有固定的映像关系。在组相联映像方式中，在确定主存存储块应该映像到哪一个 Cache 组时，采用的是直接映像方式，当主存存储块映像到某个 Cache 组后，具体再映像到此 Cache 组内的哪一个 Cache 存储块，则采用的是全相联映像方式。由此可见，组相联映像方式实际上是直接映像方式和全相联映像方式的一种折中。

如前所述，组相联映像方式要求主存和 Cache 都进行分组，主存中各组内的存储块数与 Cache 分组的组数相同。图 3-22（a）所示为 2 路组相联映像方式，Cache 分成 8 组（0～7），每组 2 个存储块（0～1）；主存分成 256 组（0～255），每组 8 个存储块（0～7）。



（a）组相联映像方式　　（b）主存地址结构

图 3-22　组相联映像方式及主存地址结构

两者之间的具体映像情况如下：

主存的第 0 块，即第 0 组的第 0 块，应映像到 Cache 第 0 组中任意一个存储块。

主存的第 1 块，即第 0 组的第 1 块，应映像到 Cache 第 1 组中任意一个存储块。

……

主存的第 7 块，即第 0 组的第 7 块，应映像到 Cache 第 7 组中任意一个存储块。

主存的其他存储块，以此类推。

若将 Cache 分成 16 组，每组只包括 1 个存储块（1 路组相联），此时主存应分为 128 组，每组 16 个存储块。按组相联映像方式的规则执行时，主存各组内的存储块先映像到对应的 Cache 组，但由 Cache 组只包括 1 个存储块，此时主存存储块就只能映像到该块位置，故此时的 1 路组相联映像方式便退化成了直接映像方式。反之，若 Cache 只分成 1 组（组号为 0），该组包含 16 个存储块（16 路组相联映像方式，相当于 Cache 未分组），此时主存应分为 2048 组，每组也仅 1 个存储块（组内的块号为 0）。按组相联映像方式的规则执行时，主存存储块

应先映像到组号也为 0 的 Cache 组中，由于这个唯一的 Cache 组包含有 16 个存储块，因此主存储块可以进一步映像到这 16 个存储块中的任意块，可见此时的 16 路组相联映像方式等效于典型的全相联映像方式。

当需要访问主存时，CPU 会给出一个 20 位的主存地址，如图 3-22（b）所示。此主存地址可以被分成三段信息：高 8 位实际上是主存的组号地址（0～255），随后的 3 位是代表组内地址（0～7），最低的 9 位是表示主存块内地址（0～511）。因此，CPU 给出的任意 20 位的主存地址，都被分成三段分别表示不同含义的地址：8 位组号块序号+3 位组内+9 位块内地址。

采用组相联映像方式时，主存存储块是按其组内地址映像到特定的 Cache 组中的，进入 Cache 组后，由于每个 Cache 组中可能又有若干个存储块，因此主存存储块再按全相联映像方式随机映像到 Cache 组内的某个存储块。换个角度看，在一个 Cache 组中，某个存储块也可能来自主存的任意一个分组。对于某个 Cache 存储块，通过其所属的 Cache 组号可以断定该存储块是由主存分组中的第几存储块映像而来的，但却不能断定存储块对应的主存组号。因此，为了确保与主存存储块所属组号的映像关系，需要为 Cache 的各存储块再增设一个 8 位的标记。

如图 3-22（b）所示，在判断 Cache 访问是否被命中时，按 2 路组相联映像方式的规则，先将主存地址中的第 2 部分（即组内地址，3 位）映像成 Cache 的组号（0～7），从而确定该主存存储块对应的 Cache 组；再将主存地址码中第 1 部分（即组号，8 位）分别与 Cache 组内的各存储块（序号 0 和 1）设置的标记（8 位）进行逐一比较。如果在该 Cache 组内找到了与主存存储块所属组号一致的 Cache 存储块标记，则表明当前 Cache 存储块被命中，随即生成 Cache 地址，据此把对主存的访问转化为对 Cache 的访问。如果比较结束，在 Cache 组内的 2 个存储块中均未找到与主存组号相同的标记，则表明 Cache 存储块未命中。对于 2 路组相联映像方式，比较标记的次数最多只需 2 次即可判断当前的 Cache 存储块是否被命中。

例 3.8 某计算机的存储系统按字节编址，Cache 共包括 16 个存储块，采用 2 路组相联映像方式，每个存储块的大小为 64 字节，主存第 268 号单元应映像到 Cache 的第几组？

解：Cache 分成 8 组，每组 2 个存储块，则主存对应分成若干组，每组也应是 8 个存储块。对主存第 268 号单元，268÷64=4 余 12，则该单元是主存第 4 块中的第 12 字节，且 4÷8=0 余 4，则该单元属于主存第 0 组内的第 4 块。

在组相联映像方式中，主存块的组内序号与 Cache 的组号对应，则第 268 号单元应装入第 4 组中。

在 Cache 分组时，每组有若干个存储块可以供主存存储块选择，因此它在主存与 Cache 之间进行地址映像时比直接映像方式更加灵活，命中率也更高。Cache 组内的存储块数量有限，因而对标记进行比较时付出的代价也不是很大，2 路组相联映像方式最多只需比较 2 次就可判断 Cache 存储块是否被命中，而全相联映像方式中最多时需要比较 16 次才能判断 Cache 存储块是否被命中，显然组相联映像方式比全相联映像方式速度更快。

存储器三种地址映像方式的特点如下：

（1）在直接映像方式中，主存各组中的任一存储块只能与 Cache 中的某一存储块映像（一对一的关系），在访问时判断比较的地址标记次数少，所用逻辑电路简单、访问速度快，但 Cache 的利用率低，会影响命中率。

（2）在全相联映像方式中，主存中的任意存储块与 Cache 中的存储块随机映像，这样就有了最大的使用灵活性，Cache 的利用率最高，但是其地址标记位数增多，在判断比较地址

时影响了速度。由于其标记位数多，所需的硬件开销大，只适合用于 Cache 容量很小的情况。

（3）在组相联映像方式中，主存中的存储块可以与 Cache 中的存储块有限度地随机映像，它是全相联映像方式和直接映像方式的一种折中方案，有利于提高命中率，访问速度较快，地址比较判断逻辑电路适中，是一种比较好的选择方式。

3.3.3　Cache 的读/写策略

CPU 对主存和 Cache 的访问包括读、写两种操作。

1．读操作

当 CPU 访问主存时，一方面将主存地址送往主存，启动读主存，同时将主存地址送往 Cache，按存储系统定义的地址映像方式将主存地址转换为 Cache 地址，如主存组号地址、块号地址和块内地址，定位到 Cache 存储块并读取内容，并将相应的 Cache 存储块标记与主存地址中相应的地址标记进行比较，如果二者相同，则表示 Cache 存储块被命中，直接将 Cache 存储块中的数据读出并送往访问源（如 CPU 中的某寄存器），放弃对主存内容的访问。

如果标记不符合，或者按地址映像方式搜索完全部 Cache 存储块后，仍未找到标记相符的 Cache 存储块，这就表明本次 Cache 存储块未命中。此时只能从主存中读取数据并送给访问源，并且把该数据所在的主存存储块整体调入 Cache 存储块中，同时要修改 Cache 存储块相应的标记。

2．写操作

Cache 中的内容是主存中部分内容的副本，应该和主存中的内容保持一致。但在操作过程中，如果有写操作，则 Cache 中的内容将发生变化。如何保持主存和 Cache 内容的一致性，这就是写操作要解决的问题。

主存处于计算机系统信息传输的中心地位，除处理器外还有其他设备（如 I/O 设备）也要直接在主存中读/写数据。如果 CPU 修改了 Cache 中的内容，却没有同时修改主存中的内容，那么主存中没有及时修改的内容就不能被别的设备使用，否则会出错。反之，如果 I/O 设备修改了主存的内容，而 Cache 中的内容没有同时修改，那 Cache 中没有及时修改的内容就应该作废，不能再用。

在多处理器系统中这个问题会变得更加复杂。在多处理器系统中，系统总线上接有多个处理器，每个处理器都可以带有自己的 Cache，只要有一个处理器的 Cache 中的内容修改了，其他处理器的 Cache 中的内容和主存中的有关内容也应该全部作废，不能再使用。

这里介绍两种保持主存和 Cache 内容一致性的写操作。

（1）全写法。全写法（Write Through）又称为写直达法。当进行写 Cache 命中时，CPU 要将修改的信息同时写入主存和 Cache 之中，这样可保证每次修改后主存和 Cache 内容的一致性。这种方法简单可靠，但是在对 Cache 更新的同时还要修改主存的内容，其整体存取时间就会变长。

（2）写回法。写回法（Write Back）是指当 CPU 写 Cache 命中时，只是修改 Cache 中的内容并做好标记，不立即写入主存。只有当该存储块要被替换时，才将修改过的 Cache 内容写入主存中的相应存储单元。这样在 CPU 的写操作中省去了不必要的每次立即写回主存的操

作，减少了 CPU 访问主存的次数，可节约整体存取时间。

采用写回法时，要求在每个 Cache 存储块中都要增添一个修改标记，以反映该存储块在替换前是否被修改过。在当存储块首次被调入 Cache 时，其修改标记置为 0；在 CPU 对其进行写入时，将修改标记置为 1，表示被修改过。当某个存储块被替换时，就按照其修改标记是 0 还是 1 来决定该存储块是被简单地被覆盖，还是需要将已经被修改的内容写回到主存。因此，采用写回法的系统机构相对要复杂一些。

3.3.4 Cache 的替换算法

在 CPU 访问带有 Cache 的主存时，如果 Cache 存储块未命中，CPU 在对主存直接读取相关存储字的同时，还要将该存储字所在的存储块调入 Cache 中。由于地址映像方式的不同，相应地有不同的处理方式。

（1）若采用直接映像方式，则调入的存储块只能存入 Cache 中固定的存储块。如果该存储块位置是空的，则存入该空间；如果该位置已被占用，就需要用新的存储块直接替换掉原有的存储块。

（2）若采用全相联映像方式或组相联映像方式，新调入的位置块可存入 Cache 中任意位置（全相联映像方式）或组内任意位置（组相联映像方式）。如果 Cache 的存储空间或组内空间已被占满，就存在一个新存储块会替换掉原有的哪一个存储块的问题。这就是替换策略（算法）要解决的问题。

常用的替换算法有三种：随机算法、先进先出算法和近期最少使用算法，在选择时要考虑存储器的总体性能，以提高 Cache 的命中率。

1．随机（RAND）算法

RAND 算法是指从 Cache 中随机取出一个存储块作为替换块，把新的存储块调入即可。RAND 算法容易实现、替换速度快，但该算法没有考虑程序的局部性原理，随机替换的存储块有可能马上又要被访问，会降低命中率和工作效率。

2．先进先出（FIFO）算法

FIFO 算法的基本思想是按调入 Cache 的先后顺序决定淘汰的顺序，在需要更新 Cache 的存储块时，总是淘汰最先装入 Cache 的存储块。FIFO 算法容易实现，系统开销（为实现替换算法而要求系统花费的时间和代价等）较小，但有些存储块虽然装入较早，但可能仍在继续使用，因此这种替换算法也存在缺陷。

3．近期最少使用（LRU）算法

LRU 算法的基本思想是先为 Cache 的各存储块建立一个 LRU 目录，该目录按某种方法记录存储块的使用情况。当需要替换存储块时，选择在最近一段时间内最久未被使用或者最少使用的存储块予以替换。显然，近期最少使用和近期最久未被使用是按使用频繁程度和使用情况决定被替换的存储块的，比较合理，能够使 Cache 的命中率较高，因而 LRU 算法使用得较多。但 LRU 算法比 FIFO 算法复杂，系统开销也稍大。

3.3.5　多层次 Cache

随着半导体器件集成度的进一步提高，Cache 已集成到 CPU 芯片中，其工作速度接近于 CPU 的速度，从而能组成两级以上的 Cache 系统。

1．指令 Cache 和数据 Cache

在计算机开始使用 Cache 时，Cache 是将指令和数据混合存储的，这种结构的 Cache 称为合一型（Unified）Cache。随着计算机技术的发展和处理速度的提高，存取数据的操作会经常与取指令的操作发生冲突，从而会延迟指令的读取，于是将指令 Cache（I-Cache）和数据 Cache（D-Cache）分为两个相互独立的 Cache，这种结构的 Cache 称为分裂型或称哈佛结构 Cache。

在 Cache 的总存储容量不变的情况下，合一型 Cache 有较高的利用率，因为在执行不同程序时，Cache 中指令和数据所占的比例是不同的，在合一型 Cache 中，指令和数据的空间可以自动调节。在新型计算机体系结构中，为了加快执行速度，一般采用将指令 Cache 和数据 Cache 分开的结构，如图 3-23 所示。

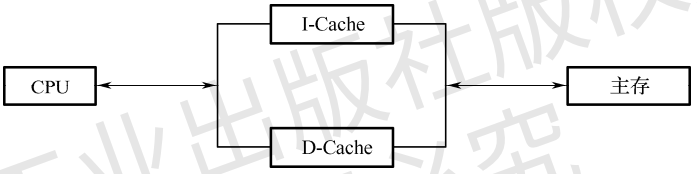


图 3-23　指令 Cache 和数据 Cache 分开的结构

假设指令 Cache 和数据 Cache 的访问时间均为 t_c，主存的访问时间为 t_m，指令 Cache 的命中率为 h_i，数据 Cache 的命中率为 h_d，CPU 取指令的比例为 f_i，则存储系统的等效访问时间为：

$$t_a=f_i[h_it_c+(1-h_i)t_m]+(1-f_i)[h_dt_c+(1-h_d)t_m]$$

例 3.9　某计算机采用主存和 Cache 组成的两级存储系统，高速缓存存取访问时间 t_c=50 ns，主存的访问时间 t_m=400 ns。访问 Cache 的命中率 h_i 为 0.96。

（1）存储系统等效的访问时间 t_a 为多少？

（2）如果将 Cache 分为指令 Cache 与数据 Cache，使等效访问时间减小了 10%。在所有的访问操作中有 20%是访问指令 Cache，而访问指令 Cache 的命中率仍为 0.96（假设不考虑写操作一致性的问题），访问数据 Cache 的命中率应是多少？

解：（1）存储系统的等效访问时间为：

$$t_a= h_it_c+(1-h_i)t_m =0.96\times50+(1-0.96)\times400=64\text{ ns}$$

（2）设改进后的访问数据 Cache 的命中率为 h_d，按公式：

$$t_a=f_i[h_it_c+(1-h_i)t_m]+(1-f_i)[h_dt_c+(1-h_d)t_m]$$

可得：

$$64\times(1-10\%)=0.2[0.96\times50+(1-0.96)\times400]+(1-0.2)[h_d\times50+(1-h_d)\times400]$$

即：

$$280h_d=275.2$$

则：

$$h_d=0.983$$

2．多层次 Cache

当芯片集成度提高后，可以将更多的电路集成在一个 CPU 芯片中，近年来设计的 CPU 芯片都将 Cache 集成在片内，片内 Cache 的读取速度要比片外 Cache 快得多。

受芯片集成度的限制，片内 Cache 的存储容量一般在几十 KB 以内，因此命中率比大容量 Cache 低。为此推出了两级 Cache 方案，其中第一级（L1）Cache 在 CPU 芯片内部；第二级（L2）Cache 在 CPU 芯片外，采用 SRAM，两级 Cache 之间一般都由专用总线相连。

当然，由于 CPU 芯片面积的限制，L1 Cache 的存储容量要小于 L2 Cache 的存储容量，但是 L1 Cache 的速度要快于 L2 Cache 的速度。L1 Cache 和 L2 Cache 又构成了一个新的存储层次，同样 L1 Cache 和 L2 Cache 之间可采用与主存和 Cache 之间类似的映像算法、替换算法和写入策略。

目前高性能的 CPU 已经支持 L3 Cache，哈佛结构的 L1 Cache 和 L2 Cache 被设置在 CPU 芯片内部。例如，Intel 公司 2004 年推出的 Montecito 双核多线程处理器，每个处理器核含有 16 KB 的 L1 指令 Cache、16 KB 的 L1 数据 Cache、1 MB 的 L2 指令 Cache、256 KB 的 L2 数据 Cache 和 12 MB 的 L3 Cache。

3.4 存储器性能的改进技术

由于 CPU 在工作中要频繁地与主存交换数据，因此主存的访问速度成为计算机速度的“瓶颈”。如何加快主存的速度一直都是计算机追求的主要目标之一。通常，为加快主存速度所采取的措施有以下几种方式：

（1）采取高速器件来尽可能缩短主存的访问时间，这要取决于器件的发展水平。

（2）在 CPU 与主存之间增设一级 Cache，以提高主存的等效速度。目前这种方式已经被广泛应用。

（3）加长主存的字位长度。显然，存储器的字位越长，访问一次存储器交换的数据就越多，主存的带宽就越宽。

（4）采取并行主存的方式来提高主存的等效速度。

在计算机中，通常采用双端口存储器、多模块交叉存储器等新技术来实现加快主存的速度。前者为时间并行，后者为空间并行。

3.4.1 分体存储体系结构（哈佛体系结构）

1945 年，冯·诺依曼首先提出了存储程序的概念和二进制原理，后来人们把利用这种概念和原理设计的电子计算机系称为冯·诺依曼体系结构计算机。冯·诺依曼体系结构计算机将指令和数据存储在一个存储器的不同物理位置，因此指令和数据的宽度相同。但是，由于指令和数据共享同一总线的结构，使得数据的传输成为限制计算机性能的瓶颈，影响了数据处理速度的提高。

目前，CPU 大都采用流水线来加快指令的解释，而流水线中的取指部件和执行部件很可能需要同时访问存储器，这就出现了访问冲突。解决访问冲突的一个办法就是采用分体存储体系结构（也称为哈佛体系结构），将指令存储空间和数据存储空间分开为各自独立的存储模

块，在执行时可以预先读取下一条指令，使得 CPU 具有较高的执行效率。分体存储体系结构的特点为：使用两个独立的存储模块来分别存储指令和数据，每个存储模块都具有一条独立的地址总线和一条独立的数据总线，利用公用的地址总线可以访问这两个存储模块，公用的数据总线则用来完成程序存储模块或数据存储模块与 CPU 之间的数据传输。

3.4.2　双端口存储器

常规的存储器都是单端口存储器，即每次只接收一个地址并且只能访问一个编址单元，从中读取或存入 1 个字节或 1 个字。尤其是在执行双操作数指令时，就需要分两次读取操作数，运算速度较低。在高速计算机系统中，主存是信息交换的中心，一方面 CPU 需要频繁地访问主存，从中读取指令、存取数据；另一方面，外设也需要频繁地与主存交换信息。由于单端口存储器每次只能接收一个访问者（读或写），这也影响了工作速度。因此，在很多系统或部件中，通常使用双端口存储器来加快计算机的工作速度。

双端口存储器具有两个彼此独立的读/写端口，每个读/写端口各自拥有一套独立的地址寄存器和译码电路，可以按各自接收的地址同时进行读/写操作，或一个写入而另一个读出操作。与两个独立的存储器不同，两个读/写端口的访问空间是一个存储块，可以访问存储块内的同一区间、同一单元。

双端口存储器可以在运算器中作为通用寄存器组来快速提供双操作数，或快速实现寄存器间的数据传输。双端口存储器的另一种应用是让其中的一个读/写端口面向 CPU，通过专门的存储总线（也称局部总线）连接 CPU 与主存，使 CPU 能快速访问主存；让另一个读/写端口则面向外设或输入/输出处理机（IOP），通过共享的系统总线进行连接，具有较大的信息吞吐量。此外，在多处理器系统中常采用双端口存储器甚至多端口存储器作为各 CPU 的共享存储器，实现多 CPU 之间的通信。双端口存储器如图 3-24 所示。

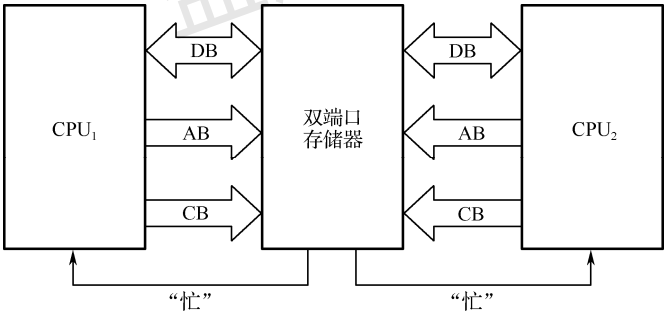


图 3-24　双端口存储器

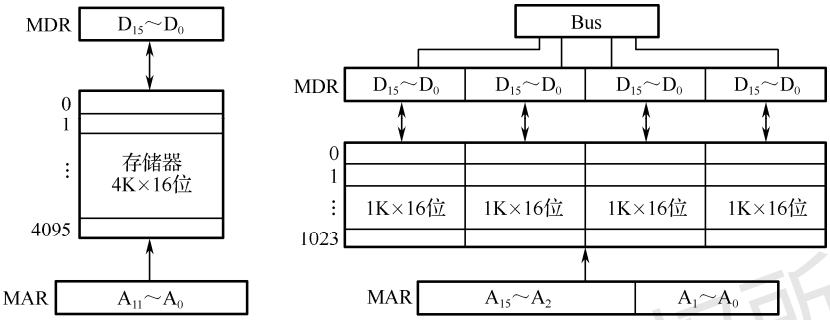
由于双端口存储器内具有两个独立的读/写端口，每个读/写端口都有自己的片选控制信号和输出允许控制信号。若两个读/写端口出现同时访问相同的存储单元时，就会发生读/写冲突，其解决方法是由判断逻辑（优先权）决定暂时将某一个读/写端口延迟，即将其置“忙”。

3.4.3　多模块交叉存储器

在多模块交叉存储器中，常用的有单体多字并行存储器和多体交叉并行存储器两种形式。

1．单体多字并行存储器

常规的主存是指单体单字存储器，其内部只包含一个存储块，访问一次存储器只能读/写一个存储字的信息。例如，一个 4K×16 位的单体单字存储器如图 3-25（a）所示。如果将一个 4K×16 位单体单字存储器分成四个部分，这样每部分都包含 1K×16 位的存储空间，其地址寄存器仍保持是 12 位的，另外为每部分存储空间独立设置一个 16 位的存储器数据寄存器（MDR），即可构成单体 4 字并行存储器，如图 3-25（b）所示。



（a）单体单字存储器　　（b）单体4字并行存储系统

图 3-25　单体单字存储器及单体 4 字并行存储器

当 CPU 访问存储器时，仍需提供 12 位地址。而在存储器内部控制部件的控制下，只需用其高端的 10 位地址（A_{11}～A_2）作为选择单体 4 字并行存储器的地址，这样就可以在一次访问存储器的时间内同时读/写 4 个字的信息到 4 个寄存器中，访问速度大约是单体单字存储系统的 4 倍。余下的低端 2 位地址（A_1～A_0）可用来控制 4 个 MDR 分时使用总线（Bus）进行数据传输。虽然采用单体 4 字并行存储器的访问速度提高了，但是要达到这个要求是有条件的，即同时读出的 4 个字地址必须是特定的连续地址。例如，访问存储器的 4 个地址为 0、1、2、3，4、5、6、7 或 1020、1021、1022、1023…。如果不是这样，那么它就像单体单字存储器一样，每次只能读/写 1 个字的数据。这就是单体多字并行存储器的局限性。

2．多体交叉并行存储器

如果在图 3-25（b）所示的单体 4 字并行存储器中，给每一部分设置独立的地址寄存器，使得它们成为 4 个独立的 1K×16 位存储器，总的存储容量仍然是 4K×16 位，4 体交叉并行存储器的结构如图 3-26 所示。

从图中可以看出，由于存储器的 4 个分体（M_3～M_0）各有自己的 MAR，4K 个地址（0～FFFH）在 4 个分体中是交叉排列分配的，因此称其为 4 体交叉并行存储器。在这种存储器中，只要不产生分体冲突就可一次读/写 4 个字的数据，存储器的等效速度大约提高 4 倍。分体冲突是指有两个访问地址位于同一个分体中。

在多体交叉并行存储器中，CPU 可以在同一个存取周期内访问多个存储块，其结构是 *m* 个相同的存储块有各自独立的工作电路（*m* 套），采用的访问方式是 CPU 同时发送的 *m* 个地址，由存储器分时使用数据总线进行数据传输。其地址的高位作为存储器的地址，低位负责选择数据。

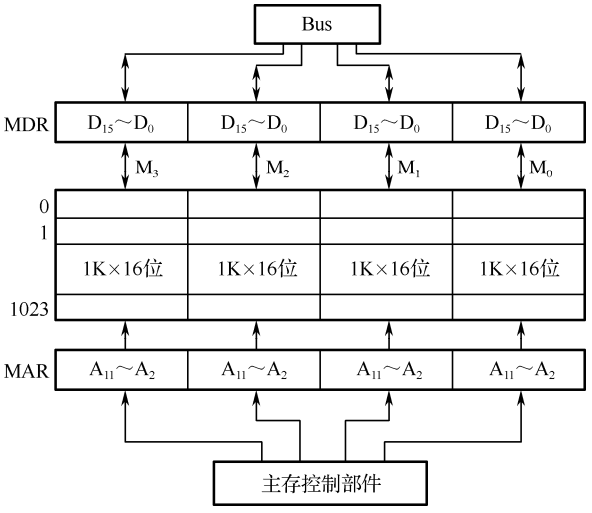


图 3-26　4 体交叉并行存储器的结构

相对来说，多体交叉并行存储器更适合采用流水线方式。当 CPU 连续访问一个字块时，可以较大地提高存储系统的带宽。

3.4.4　相联存储器

在计算机中，查找存储数据的准则称为关键字。在 RAM 中地址是关键字，人们可以按照地址获得相应的数据。相联存储器不是按照地址来进行访问的，而是按照所存数据的全部内容或部分内容进行查找或存储的。这个关键字是数据域的一部分而不是地址。例如，在虚拟存储器中，将虚地址的虚页号与相联存储器中所有行的虚页号进行比较，若有内容相等的行，则将其相应的实页号取出。在 Cache 中的地址映像表也可以用相联存储器构成，它们都是按数据的部分内容来进行检索的。

相联存储器主要由存储块、检索寄存器、屏蔽寄存器、符合寄存器、比较线路、代码寄存器、译码选择电路组成，其组成框图如图 3-27 所示。

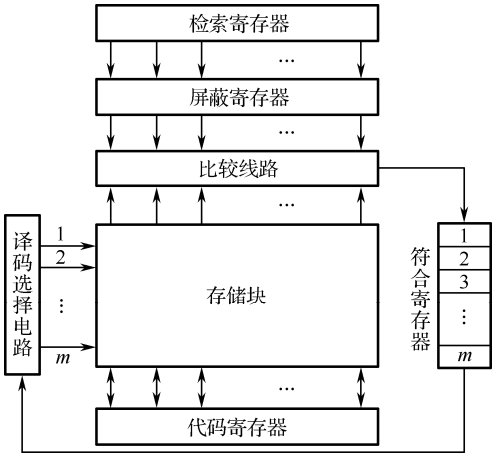


图 3-27　相联存储器的组成框图

（1）检索寄存器。检索寄存器用来存储检索字，其位数和相联存储器的字长相等，在每次检索时，取检索寄存器中若干位为检索项。

（2）屏蔽寄存器。屏蔽寄存器用来存储屏蔽码，其位数和检索寄存器的位数相等。在每次检索时，检索寄存器中作为检索项的部分相对于屏蔽寄存器中的对应位被置为 1，其余位被置为 0。在进行比较时，屏蔽码使相应位不参与比较，不论相应位是 1 还是 0 都不会影响比较结果。例如，假设检索寄存器的位数为 n，则屏蔽寄存器的位数也是 n，在进行某次检索时，取检索寄存器中前 16 位为检索项，那么屏蔽寄存器的前 16 位被置为 1，而屏蔽寄存器的第 17～n 位均被置为 0，这样经过比较线路时就会将检索寄存器中第 17～n 位屏蔽掉，只有前 16 位参加对存储块中所有存储单元的检索比较。

（3）符合寄存器。。符合寄存器用来存储与检索项内容与相符的存储块存储单元地址，其位数等于相联存储器的存储单元数，每一位对应一个存储单元，位的序数是相联存储器的存储单元地址。例如，设相联存储器有 W 个字，那么符合寄存器的字长就是 W 位。如果比较结果是第 i 个字满足要求，则将符合寄存器中第 i 位置为 1，将其余位均置为 0；同时，如果有 m 个字满足要求，则相应地就有 m 位被置为 1。

（4）比较线路。比较线路用来比较检索项和从存储块中读出的所有存储单元内容，如果有某个存储单元和检索项符合，就把符合寄存器的相应位置为 1，表示该字已被检索。

（5）代码寄存器。代码寄存器用来保存从存储块中读出的代码或向存储块中写入的代码。

（6）存储块。存储块通常是高速半导体存储器，以便进行快速存取。

相联存储器的最大优点是可对存储器中所有存储单元的 1 位或部分位同时进行比较。利用这一优点，可以通过相联存储器进行诸如大于、小于、等于、是否处于给定的上/下界范围、求最大值、求最小值等多种类型的逻辑检索。由于相联存储器存储单元、存储器结构都比较复杂，并且造价比较高、功耗也比较大，因此存储容量无法做得太大。相联存储器主要用于快速检索的场合，如主存-Cache 存储层次的地址映像表和虚拟存储器中的快表等。

相联存储器除了可以应用于虚拟存储器与 Cache，还经常用于数据库与知识库中按关键字进行检索。通常，要从按地址访问的存储器中检索出某一存储单元，平均约进行 $m/2$ 次操作（m 为存储单元数）；而在相联存储器中仅需要进行一次检索操作即可，可大大提高处理速度。近年来，相联存储器也应用到了一些新型的并行处理和人工智能系统结构中，如在语音识别、图像处理、数据流计算机等。

3.5 虚拟存储系统

目前计算机的主存主要采用 DRAM，由于技术和成本等原因，主存的存储容量受到限制，并且不同计算机所配置的存储容量也不相同。但在使用中，用户显然不希望受到特定计算机的主存存储容量大小的制约，因此，如何解决这两者之间的矛盾是一个亟须解决的问题。

为了解决上述问题，在计算机中采用了虚拟存储技术。程序员可以在一个不受物理内存空间限制并且比物理内存空间大得多的虚拟逻辑地址空间中编写程序，就好像每个程序都独立拥有一个巨大的存储空间一样。在程序执行过程中，把当前执行到的一部分程序和相应的数据调入主存，其他暂时不用的部分程序和数据仍保留在辅存（如硬盘存储器）中。这种借用辅存为程序提供的很大的虚拟存储空间称为虚拟存储器。

在执行指令时，计算机通过硬件将指令中的逻辑地址（也称虚拟地址或虚地址）转化为主存的物理地址（也称主存地址或实地址）。在地址转换过程中，需要检查是否发生访问信息缺失、地址越界或访问越权。若发生信息缺失，则由操作系统进行主存和辅存之间的信息交换；若发生地址越界或访问越权，则由操作系统进行存储访问的异常处理。由此可以看出，虚拟存储技术既解决了编程空间受限的问题，又解决了多道程序共享主存带来的安全问题。

虚拟存储器是由硬件与操作系统共同协作实现的，涉及计算机系统许多层面，包括操作系统中的许多概念，如存储器管理、虚地址空间、缺页处理等。

3.5.1　虚拟存储器简介

1．虚拟存储器的功能和特点

虚拟存储器指的是主存-辅存层次，它能使计算机具有辅存的存储容量、接近于主存的速度以及辅存的每位成本价格，可以使程序员按比主存大得多的存储空间来编写程序，即按虚存空间编址。当然，主存实际容量的大小也会影响系统的工作效率。如果程序过大而主存容量过小，频繁的替换会使程序运行速度明显下降。辅存只与主存进行信息交换，不能直接运行，需要依靠辅助的软、硬件来实现两者之间的自动调入工作。虚拟存储器的特点如下：

（1）允许用户访问比实际存储空间大得多的地址空间，虚存空间取决于机器所能提供的虚地址长度。

（2）可以将当前和频繁使用的内容自动从辅存调入主存，其他暂时不用的程序和数据则放在辅存中。每次访问都可以自动进行虚、实地址的转换，为用户提供了极大的方便。

2．虚拟存储器与 Cache 的比较

从工作原理的角度来看，主存-辅存层次和主存-Cache 层次之间有很多相似之处。在虚拟存储系统中所采取的地址映像方式同样有全相联映像、组相联映像和直接映像等方式。替换算法也多采用近期最少使用（LRU）算法。实际上，这些替换算法和地址映像方式最早是应用于虚拟存储系统中的，后来才应用到 Cache 中。主存-辅存层次与主存-Cache 层次的对比如表 3-3 所示。

表 3-3　主存-辅存层次与主存-Cache 层次的对比

层　次	不 同 点	相 同 点
主存-辅存层次	时间较长，基本存储单元大（段或页）	采用地址映像方法、替换算法
主存-Cache 层次	时间较短，基本存储单元小（字块）	采用地址映像方法、替换算法

3.5.2　虚拟存储器的管理方式与存储保护

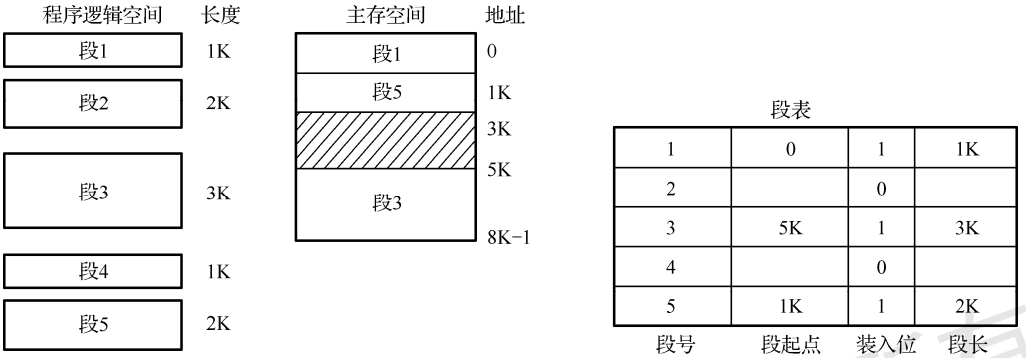
主存-辅存层次的信息传输可采用段、页或段页式三种不同的管理方式。

1．段式管理方式

在虚拟存储器中，实行按程序段或数据段进行分配管理的方式称为段式管理方式。段式

管理方式的优点是段的分界与程序的自然分界相对应，各段的逻辑独立性使它易于编译、管理、修改和保护，也便于多道程序共享。但是正因为各段的长度各不相同，容易在段间留下许多空余的零碎存储空间而无法利用，会造成一定的浪费。

在采用段式管理方式的虚拟存储器中，为了进行地址映像需要为每个用户建立一个段表，通过段表来指明各程序段或数据段在主存中的位置。段式管理方式如图 3-28 所示。



（a）段式管理方式中虚拟存储器的存放关系　　（b）段表

图 3-28　段式管理方式

段表包含段号、段起点、段长、装入位和访问权等。任何一个段都可从 0 地址开始编址，其访问权是指该段所允许的访问形式，便于以段为单位进行存储保护。例如，程序段只允许执行、不允许写，数据段只允许读/写、不允许执行，常数段只允许读、不允许写等。在程序装入主存时，由操作系统自动构造段表。在程序运行时，访问段表自动实现主存和辅存之间的地址映像。段式管理方式的虚拟存储器地址映像过程如图 3-29 所示。

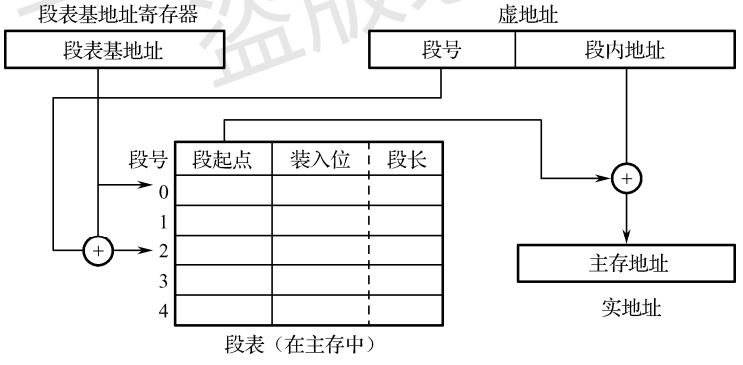


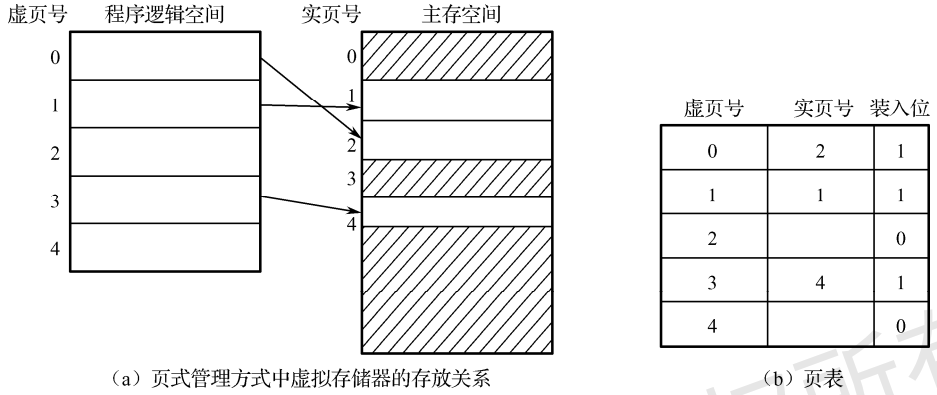
图 3-29　段式管理方式的虚拟存储器地址映像过程

2．页式管理方式

页式管理方式的信息传输是以指定长度的存储页为单位来进行的，所以主存和辅存的存储空间都要被划分为等长的存储页。一般将辅存的存储页称为虚页或逻辑页，而将主存的存储页称为实页或物理页。由于各页的起点和终点地址都是固定的，这样给构建页表带来了方便。主存内新页的调入也很容易，只要有空白页就可以进行。如果在主存中无空白页，就需要进行页的替换。页式管理方式比段式管理方式浪费的存储空间要小得多。由于各存储页不

是逻辑上独立的实体，处理、维护、共享和保护等操作都不如段式管理方式方便。

虚地址一般分为两个部分，其中高位字段称为虚页号，低位字段称为页内行地址。实地址也要分为两个部分，其中高位字段为实页号，低位字段为页内行地址。由于两者的页大小一样，所以页内行地址是相同的。虚地址到实地址的映像由存储在主存中的页表来实现，在页表中，每一个虚页有一个表目。页式管理方式如图 3-30 所示。



（a）页式管理方式中虚拟存储器的存放关系

虚页号	实页号	装入位
0	2	1
1	1	1
2		0
3	4	1
4		0

（b）页表

图 3-30　页式管理方式

页表中至少要包含该虚页对应存储在主存中的页地址（实页号），用它来作为访问实（主存）地址的高位地址字段，然后与虚地址的页内行地址字段相拼接（二者的页内行地址相同），就产生了完整的实地址，据此可访问实存。页式管理方式的地址映像如图 3-31 所示。

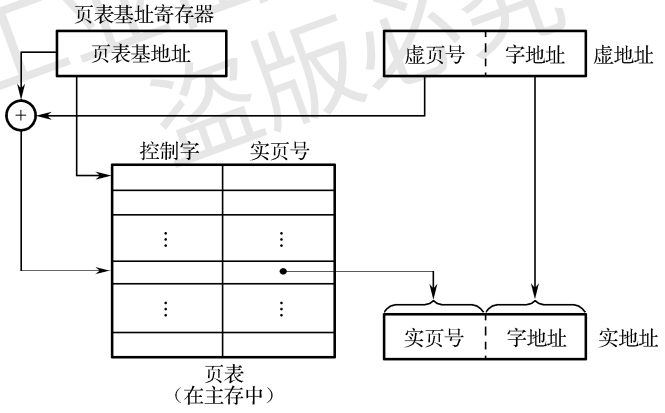


图 3-31　页式管理方式的地址映像

通常，在页表的表项中还包括由装入位（有效位）、修改位和保护位等多种标志所组成的控制字段。例如，装入位为 1 时表示该虚页已从外存调入主存；装入位为 0 时表示对应的虚页尚未调入主存，这就需要产生页失效中断来启动输入/输出子系统，通过辅存的页表来获得将要被调入的虚页在辅存中的具体地址，由辅存控制器将读出新的虚页存储到主存中。修改位用于表示实页中的内容是否被修改过，在实页被替换时要将其写回辅存。

值得注意的问题是，在页式管理方式中，如果页表存储在主存（称为慢表）中，那么在访问存储器时要先查页表再访问主存才能取得数据。如果出现页失效情况，则需要进行页替换、页修改，访问主存次数就更多了。如果采用把页表的使用最频繁的部分存储在快速存储

器（如相联存储器）中（称为快表），就会减少访问主存的次数，缩短访问时间。目前，计算机通常采用快表与慢表相结合的方式来实现内部的地址映像，如图 3-32 所示。

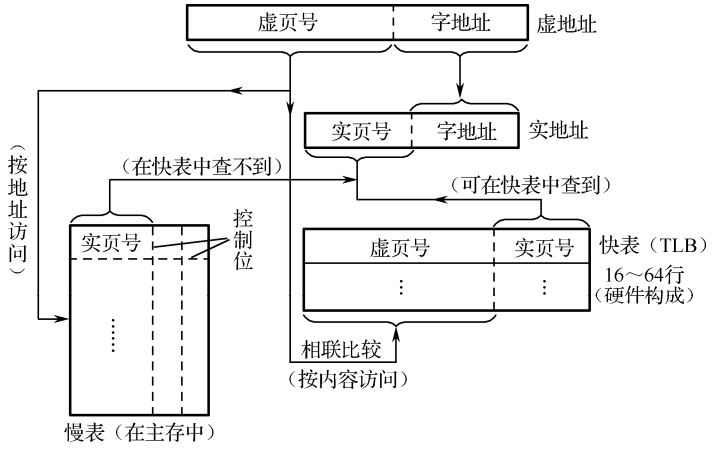


图 3-32　采用快表与慢表相结合的方式实现内部的地址映像

快表也称为转换旁路缓冲器（Translation Lookaside Buffer，TLB），其实际存储空间较小，一般为 16～64 行。快表只是慢表（主存中的页表）的部分副本。在查表时，根据虚页号同时查快表和慢表，当快表中有该虚页号时，就能很快地找到对应的实页号并送入主存地址寄存器，并同时终止对慢表的查找。如果在快表中查不到该虚页号，那么就要花费一个访问主存时间来查询慢表，然后将从慢表中查到的实页号送入主存地址寄存器，并将和该实页号对应的虚页号送入快表中存储。当然，在替换快表中某一行内容的过程中，也同样需要使用替换算法。

3．段页式管理方式

段页式管理方式是段式管理方式和页式管理方式的结合，这种方式将物理空间分页，逻辑（虚存）空间分段，段内分成页。程序对主存的调入/调出是按页进行的，但它又可以按段实现共享和保护。因此，段页式管理方式具有页式管理方式和段式管理方式的优点，这种方式的缺点是在地址映像过程中需要多次查表，这是由于存储系统中的每道程序都是通过一个段表和一个页表来进行定位的。在段表中的各表目都对应一个程序段，在每个表目中都有一个指向该段的页表起始地址（页号）以及该段的控制保护信息。由页表指明该段各页在主存中的位置以及是否装入、修改等状态信息。

如果有多个用户进程在计算机中运行，则称为多道程序。多道程序中的每一道（每个用户）都需要一个基号（用户标志号），可由它指明该道程序的段/表起点（存储在基址寄存器中），这样虚地址就应包括基号、段号、页号、页内行地址。在段页式管理方式中，在虚地址向实地址的映像过程中至少需要查找两次表（段表与页表）才能获得实地址。

4．存储保护

为使计算机存储系统能正常工作，不仅要防止由于一个程序出错而破坏其他的程序和系统软件，还要防止一个程序不合法地访问不是分配给它的主存区域。为此，系统应提供存储保护。目前在进行存储保护时，主要采用存储区域保护和访问方式保护两种方式。

（1）存储区域保护。在主存中一般采用界限寄存器方式，由系统软件经特权指令设置上下界寄存器，为每个程序划定存储区域，禁止越界访问。在虚拟存储器中，由于一个程序的各页可能被离散地存储在主存中，通常需要采用段表和页表保护、键保护以及环保护等方式。

① 段表和页表保护。每个程序都有自己的段表和页表，段表和页表本身都设置有自己的保护功能。无论地址如何出错，也只能影响到相应的几个实页。这种段表和页表保护方式是在没有形成实地址前的保护。若在地址映像过程中出现了错误，那么这种保护就是无效的，因此还需要其他保护方式。

② 键保护。键保护的基本思想是为主存的每页分配一个位，称为存储键，它相当于一把“锁”。存储键是由操作系统赋予的，每个用户的实页的存储键都相同。为了打开这个“锁”必须有“钥匙”，称为访问键。访问键赋予每道程序，存储在该道程序的状态寄存器中，当数据要写入主存的某一页时访问键要与存储键相比较，若两键相符则允许访问该页，否则拒绝访问。

③ 环保护。以上两种保护方式都是保护别的程序区域不被破坏，而正在运行的程序本身则得不到保护。环保护则可以对正在执行的程序本身进行保护。环号的大小表示保护的等级，环号越大，等级越低。所以系统程序应在内层（环号小），用户程序（环号大）应在外层，内层允许访问外层的存储区域，而外层不能访问内层的存储区域。

（2）访问方式保护。主存内的信息有读（R）、写（W）和执行（E）三种方式，其中，执行是针对指令的。相应的访问方式保护就有 R、W、E 三种以及由这三种方式形成的逻辑组合。访问方式保护可以与存储区域保护方式结合起来使用，以上所讲的存储区域保护都是由硬件实现的。在一些计算机系统中，还可通过特权指令来实现保护。

3.5.3　虚拟存储器的工作过程

虚拟存储器往往包含主存空间、辅存（磁盘存储器）空间和虚存空间三种地址，分别对应主存地址、辅存地址和虚地址，程序员在虚存空间中编写程序。在直接寻址方式下，由指令的地址码给出的地址就是虚地址，虚地址是指辅存的逻辑地址，由虚页号及页内行地址组成。当访问辅存时，还需要进行虚地址与辅存实地址之间的映像。以磁盘存储器为例，其辅存实地址包括磁盘机号、柱面号、磁头号、块号和块内地址。

虚拟存储器具有虚地址到辅存实地址的映像功能。辅存一般是按存储块进行编址的，而不是按字进行编址的。若一个存储块的大小等于一个虚面的大小，这样就只需把虚页号 N_v（不含页内行地址）映像成实页号（不含辅存块内地址）即可完成虚地址到辅存实地址的映像。在采用页表式管理方式时，将由虚页号 N_v 转换成辅存实页号 N_{vd} 的表称为外页表，而把由 N_v 转换成主存实页号的表称为内页表。例如，在一个具有 Cache 和页式管理方式的虚拟存储器中，CPU 的一次访问操作可能涉及快表、内页表、外页表、Cache、主存和辅存。在具体的访问过程中，CPU 首先给出访存所需的虚地址 VA，然后判断对应页是否在快表中。如果在快表中，则直接将虚地址映像为实地址，然后进行访问 Cache 的一系列操作。如果不在快表中，则访问工作过程如图 3-33 所示。图 3-33 所示为虚拟存储器的工作过程。