# 第 3 章　存储系统


存储器是计算机的记忆部件，用来存放程序和数据。设计大容量、高速度、低成本的存储器一直是计算机硬件发展的重要课题。本章讲述存储器的分类、分级与存储器的技术指标；各种半导体存储器的工作原理及与 CPU 的连接；高速存储器；高速缓冲存储器和虚拟存储器。


- 掌握存储器的基本知识，包括存储器的基本概念、存储器的分类和存储器的性能指标;
- 掌握各种半导体存储器的工作原理；存储器与 CPU 的连接；高速存储器;
- 掌握高速缓冲存储器 ( 包括 Cache 的基本结构及工作原理、Cache-主存地址映像、替换算法 );
- 理解虚拟存储器的结构与调度算法。

## 3.1　存储器系统概述

### 3.1.1　存储器分类

存储器是计算机系统的记忆部件，用来存放程序和数据。目前，计算机所用存储器的种类越来越多，现介绍以下三种分类方法。

1. CPU 之外的浮点运算器

凡是明显具有两种稳定状态的物质和元器件，都可以用来存储二进制代码“0”和“1”。这些物质可以作为存储器的存储介质。而存储器存取速度的快慢又取决于存储介质物理状态的改变速度。

（1）半导体存储器。利用半导体器件组成的半导体存储器体积小、速度快，但当电源断电时，信息也随之消失，属于易丢失存储器。常用作主存、高速缓存器。

（2）磁表面存储器。利用磁层上不同方向的磁化区域表示信息，特点是容量大，非破坏性读出，长期保存信息，但速度慢，常用作外存。

（3）光盘存储器。利用光斑的有无表示信息。特点是容量很大，非破坏性读出，长期保存信息，速度慢。常用作外存。

2. 按存取方式分类

（1）随机存取存储器 RAM（Random Access Memory）。在随机存取存储器中，以任意次序读写任意存储单元所用的时间都相同，与存储单元的地址无关，如半导体存储器。通常意义上的随机存储器多指可读写存储器，即它的每个存储单元的内容，可根据程序的要求随机地读出或写入，所以实际上称它为可读写存储器更准确，然而习惯上都把它叫做随机存储器。

（2）只读存储器 ROM（Read Only Memory）。这种存储器在程序执行过程中，存储单元的内容只能读出不能写入。一般用来存放不变的程序或数据，如系统引导程序、监控程序等。RAM 和 ROM 合起来构成内存。

（3）顺序存取存储器 SAM（Sequential Access Memory）。在顺序存取存储器中，只能以某种预先确定的顺序来读写存储单元，存取时间与存储单元的物理位置（或地址）有关。例如磁带存储器就是顺序存取存储器。磁盘存储器则介于随机存取和顺序存取之间，它的读写机构磁头能直接指向一个很小的存储区域，然后在这个磁道内进行顺序存取操作。

3. 按存储器在计算机中的作用分类

（1）主存储器。主存储器简称主存，是计算机系统的主要存储器，用来存放正在执行的程序和数据。它可以直接与 CPU 交换信息。其特点是存取速度较快，但存储容量较小且价格较高。目前主要采用半导体存储器，采用随机存取方式。因其设在主机内部，又称为内存储器（简称内存）。

（2）辅助存储器。辅助存储器简称辅存，它用来存放当前不使用的程序和数据，一般不能与 CPU 直接交换信息，CPU 要使用其中的某些程序和数据时，要事先将其调入主存储器，然后 CPU 直接访问。因其设在主机外部，属于输入输出的外围设备，又称为外存储器（简称外存）。其特点是存储容量大，价格低，可永久地保存信息，但存取速度慢。

辅助存储器分为磁表面存储器和光存储器。现在使用的磁表面存储器主要是磁带和磁盘，光存储器主要是光盘。

（3）高速缓冲存储器。高速缓冲存储器介于 CPU 和主存这两个工作速度不同的部件之间，当 CPU 和主存进行信息交换时起缓冲作用。高速缓存用来存放当前最可能频繁使用的程序和数据。其特点是速度快，但容量小，每位价格高。

存储器的分类如图 3-1 所示。

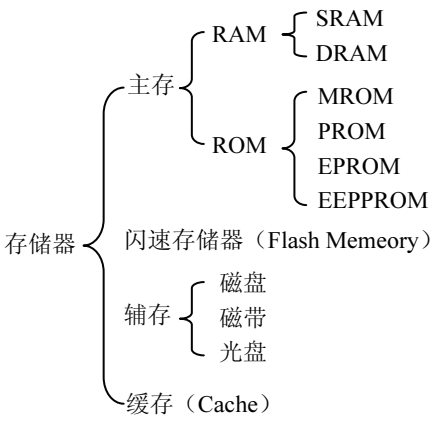


图 3-1　存储器分类图

### 3.1.2 三级存储体系结构

存储器的 3 个主要性能指标：容量、速度和每位价格（简称位价）。半导体存储器速度快，但容量不可能很大，且成本较高。磁表面存储器和光盘存储器成本低，容量可以很大，但速度低，与 CPU 高速处理能力不匹配。从整个计算机技术的发展来看，存在着这样一个明显的事实，即主存的工作速度总是落后于 CPU 的需要，主存的容量总是落后于软件的需求。因此，单从改进主存存储技术的途径来提高存储器性能，很难满足计算机系统对存储器提出的速度快、容量大和成本低的要求。为了解决这一问题，目前在计算机系统中，通常采用三级存储体系结构，即使用高速缓冲存储器（Cache）、主存储器和外存储器，如图 3-2 所示。

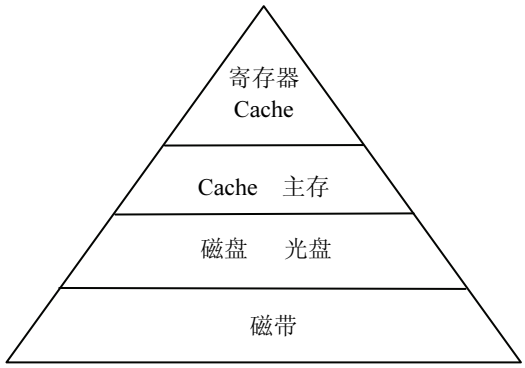


图 3-2 存储器分级结构图

图 3-2 中由上至下，速度越来越慢、容量越来越大、位价越来越低，CPU 访问的频度也越来越少。中央处理器能直接访问的存储器称为内存储器，它包括高速缓冲存储器和主存储器，中央处理器不能直接访问外存储器，外存储器的信息必须调入内存储器后才能为中央处理器进行处理。

实际上，存储系统层次结构主要体现在缓存－主存和主存－辅存这两个存储层次上，构成 Cache、主存和辅存三级存储结构，如图 3-3 所示。

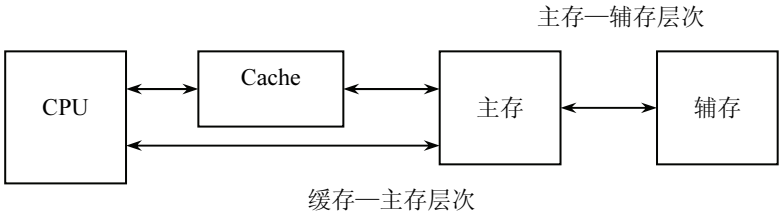


图 3-3 三级存储结构

缓存－主存层次主要解决 CPU 和主存速度不匹配的问题。由于缓存的速度比主存的速度高，只要将 CPU 近期要用的信息调入缓存，CPU 便可以直接从缓存中获取信息，从而提高访存速度。但由于缓存的容量小，因此需不断地将主存的内容调入缓存，使缓存中原来的信息被替换掉。主存和缓存之间的数据调动是由硬件自动完成的，对程序员是透明的。

主存－辅存层次主要解决存储系统的容量问题。辅存的速度比主存的速度低，而且不能

和 CPU 直接交换信息，但它的容量比主存大得多，可以存放大量暂时未用到的信息。当 CPU 需要用到这些信息时，再将辅存的内容调入主存，供 CPU 直接访问。主存和辅存之间的数据调动是由硬件和操作系统共同完成的。

从 CPU 角度来看，缓存－主存这一层次的速度接近于缓存，高于主存；其容量和位价却接近于主存，这就从速度和成本的矛盾中获得了理想的解决办法。主存－辅存这一层次，从整体分析，其速度接近于主存，容量接近于辅存，平均位价也接近于低速、廉价的辅存价位，这又解决了速度、容量、成本这三者的矛盾。现代的计算机系统几乎都具有这两个存储层次，构成了缓存、主存、辅存三级存储系统。

### 3.1.3　主存储器的基本结构

主存储器（简称主存）的基本结构已在第 1 章介绍过，如图 3-4 所示。来自地址总线的存储器地址由地址译码器译码（转换）后，找到相应的存储单元，由读/写控制电路根据相应的读、写命令来确定对存储器的访问方式，完成读写操作。数据总线则用于传送写入内存或从内存取出的信息。

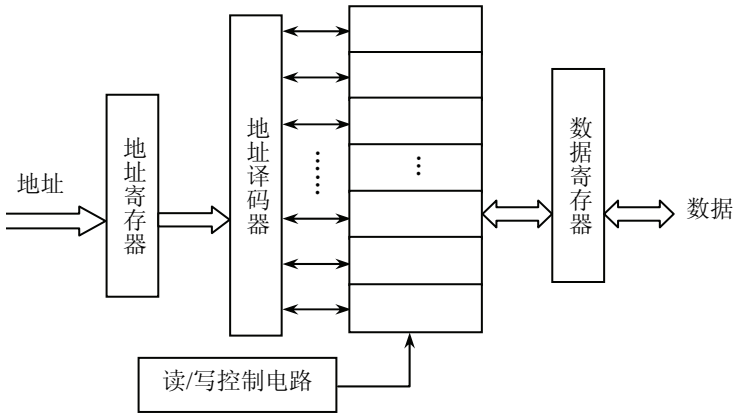


图 3-4　存储器基本结构图

1. *存储单元的编址方式*

一个存储单元可能存放一个字，也可能存放一个字节，这是由计算机的结构确定的。对于字节编址的计算机，最小寻址单位是一个字节，相邻的存储单元地址指向相邻的存储字节；对于字编址的计算机，最小寻址单位是一个字，相邻的存储单元地址指向相邻的存储字。

存储单元是 CPU 对主存可访问操作的最小存储单位。

例如，IBM 370 机是字长为 32 位的计算机，主存按字节编址，每一个存储字包含 4 个单独编址的存储字节，字地址即是该字高位字节的地址，其字地址总是等于 4 的整数倍，正好用地址码的最末两位来区分同一个字的四个字节。PDP-11 机是字长为 16 位的计算机，主存也按字节编址，每一个存储字包含 2 个单独编址的存储字节，它的字地址总是 2 的整数倍，但却是用低位字节地址作为字地址，并用地址码的最末 1 位来区分同一个字的两个字节。

2. *存储器的译码方式*

地址译码器的输入信息来自 CPU 的地址寄存器。地址寄存器用来存放所要访问（写入或

读出）的存储单元的地址，中央处理器要选择某一存储单元，就在地址总线上输出此单元的地址信号给地址译码器，地址译码器把用二进制代码表示的地址转换成输出端的高电位，用来驱动相应的读写电路，以便选择所要访问的存储单元。

地址译码有两种方式：单译码方式和双译码方式。

单译码结构也称字结构。在这种方式中，地址译码器只有一个，译码器的输出叫字选线，如图 3-5 所示。

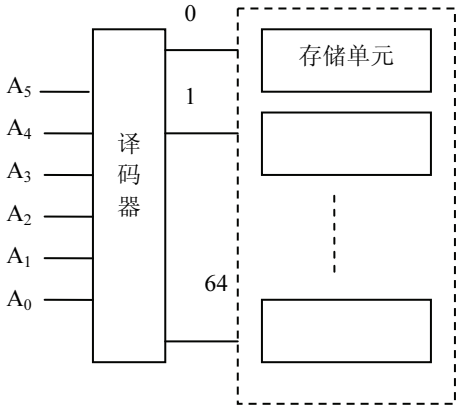


图 3-5　单译码方式

双译码方式将地址分成 x 向、y 向两部分，第一级进行 x 向（行译码）和 y 向（列译码）的独立译码,然后在存储阵列中完成第二级的交叉译码，如图 3-6 所示。双译码方式的优点是节省了译码器输出线的条数，适合于大容量存储器。

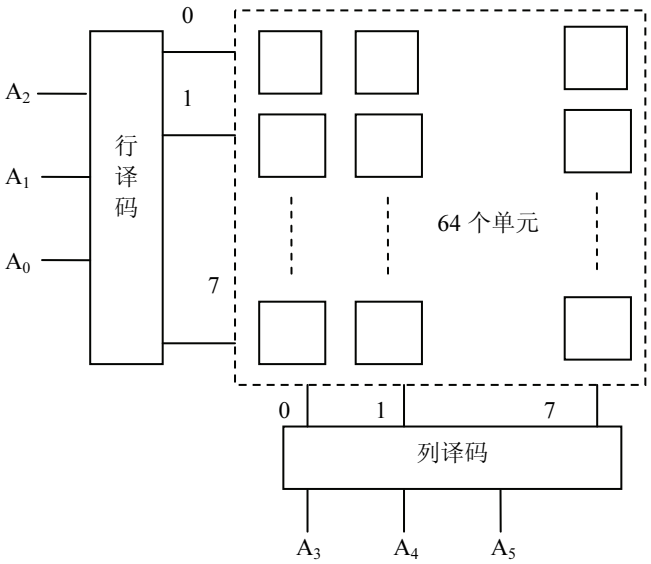


图 3-6　双译码方式

如 1K 存储单元，用单译码方式需要译码器输出 1024 条译码输出线；而采用双译码方式只需要 32+32=64 条输出线。

### 3.1.4　主存储器的主要技术指标

1. 存储容量

存储器可以容纳的二进制信息总量称为存储容量。容量越大，能存储的信息就越多，计算机系统的功能也就越强、使用越灵活。

存储容量常用字（Word）或字节（Byte）来表示，如 64K 字、512KB 等。也就是说，存储容量可用存储单元数×存储单元长度（即字长）来表示。例如，一个存储器的存储单元数为 4K，字长为 16 位，则存储容量可用 4096×16 来表示。字长越长，能存放的数的精度就越高。

存储容量的单位还有 MB、GB 和 TB，它们之间的关系是：

$1TB=1024GB=2^{40}B$

$1GB=1024MB=2^{30}B$

$1MB=1024KB=2^{20}B$

$1KB=1024B=2^{10}B$

2. 速度

衡量存储器速度的指标主要有三个：

（1）存储器存取时间。存储器存取时间指启动一次存储器操作（即收到读或写操作的命令）到完成该操作所需的时间，也称为存储器访问时间。

目前，大多数存储器的存取时间在 ns 级（$1ns=10^{-9}s$）。

（2）存储周期。存储周期指连续启动两次独立的存储器操作所需的最小时间间隔，也就是存储器进行一次完整的读写操作所需的全部时间。存储周期时间通常大于存取时间，这是因为存储器读写之后，还需要一定的时间来完成一些内部操作。

（3）存储带宽。单位时间内存储器所存取的信息量，通常以位/秒或字节/秒做度量单位。

3. 价格

价格是存储器的一个经济指标，一般用每位价格来表示。存储器的价格与存储容量、速度成正比。

衡量存储器性能的其他指标有功耗、可靠性、体积等。这些指标之间往往互相制约。在设计制造存储器时，应尽量提高存储器的性能价格比。

## 3.2　随机存取存储器

目前广泛适用的内存储器是半导体存储器。根据存储信息机理不同，分为静态读写存储器（SRAM）和动态读写存储器（DRAM）。按信息存储方式分，半导体存储器分为随机读写存储器和只读存储器。

### 3.2.1　SRAM

1. SRAM 存储元

我们把存放一个二进制位的物理器件称为存储元，它是存储器的最基本构件。

静态 RAM 是利用双稳态触发器记忆信息。六管静态 MOS 记忆单元电路中的 $T_1$～$T_6$ 管构

成一个记忆单元的主体，能存放一位二进制信息，其中：$T_1$、$T_2$ 管构成存储二进制信息的双稳态触发器，如图 3-7 所示。

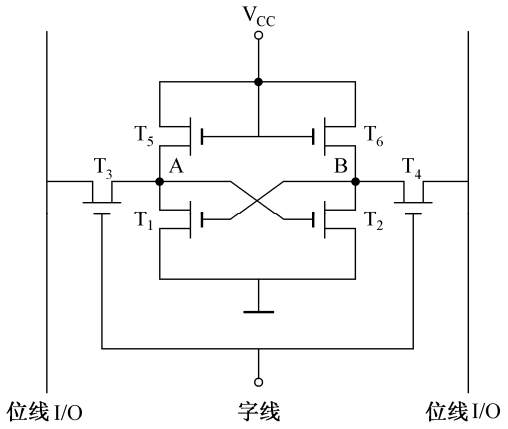


图 3-7　六管静态 MOS 记忆单元电路

由于静态是用触发器工作原理存储信息，因此即使信息读出后，它仍保持其原状态， 不需要再生。但电源掉电时，原存信息丢失，故它属易失性半导体存储器。

2. 四管动态 MOS 记忆单元电路

动态 RAM 是利用栅极电容上的电荷记忆信息。四管动态记忆单元电路中的 $T_0$、$T_1$ 管不再构成双稳态触发器，而靠 MOS 电路中的栅极电容 $C_0$、$C_1$ 来存储信息，如图 3-8 所示。

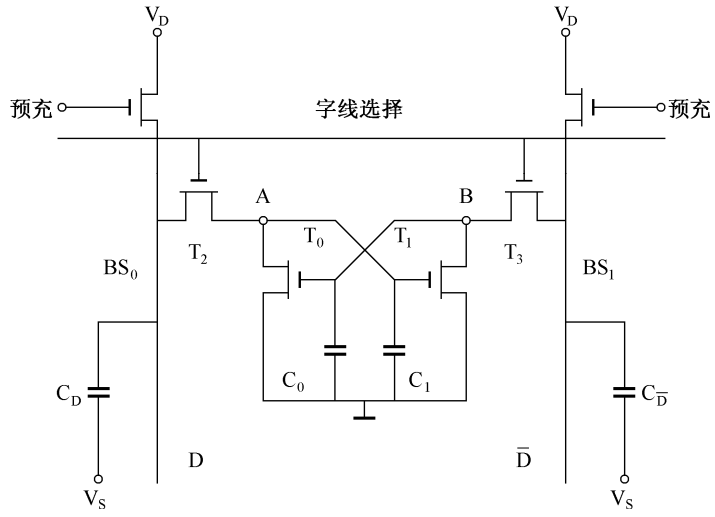


图 3-8　四管动态 MOS 记忆单元电路

3. 单管动态记忆单元电路

单管动态记忆单元由一个 MOS 管 $T_1$ 和一个存储电容 C 构成。单管动态记忆单元是破坏性读出，必须采取重写（再生）的措施。进一步减少记忆单元中 MOS 管的数目可形成更简单的三管动态记忆单元或单管动态记忆单元，如图 3-9 所示。

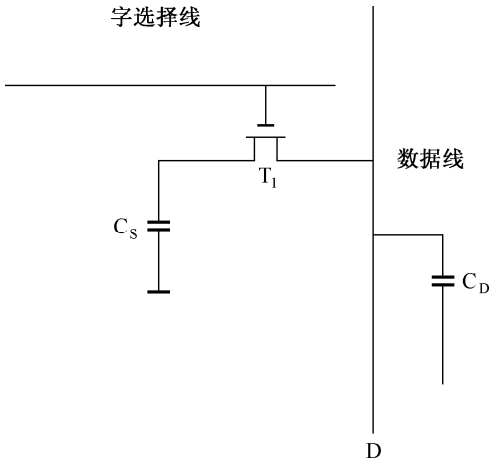


图 3-9　单管动态记忆单元电路

### 3.2.2　动态 RAM 的刷新

1. 刷新间隔

为了维持 MOS 型动态记忆单元的存储信息，每隔一定时间必须对存储体中的所有记忆单元的栅极电容补充电荷，这个过程就是刷新。

一般选定 MOS 型动态存储器允许的最大刷新间隔为 2ms，也就是说，应在 2ms 内将全部存储体刷新一遍。

2. 刷新方式

如前所述动态存储器是靠电容来储存信息的，电荷量会随着时间和温度而减少，所以必须定期刷新，以保证它们信息的正确性。刷新间隔主要根据栅极电容电荷的泄放速度来决定。典型标准时每隔 8ms 到 16ms 必须刷新一次，而某些器件的刷新周期可以更大。

刷新的过程实质上是先将原存信息读出，再由刷新放大器形成原信息并重新写入。然而通常情况下，人们不能准确地预知读操作出现的频率，因此无法阻止数据丢失。在这种情况下，必须对 DRAM 进行定期刷新。

常见的刷新方式有集中式、分散式和异步式三种：

（1）集中刷新方式。集中刷新方式在允许的最大刷新间隔内，按照存储芯片容量的大小集中安排若干个刷新周期，刷新时停止读写操作，如图 3-10 所示。

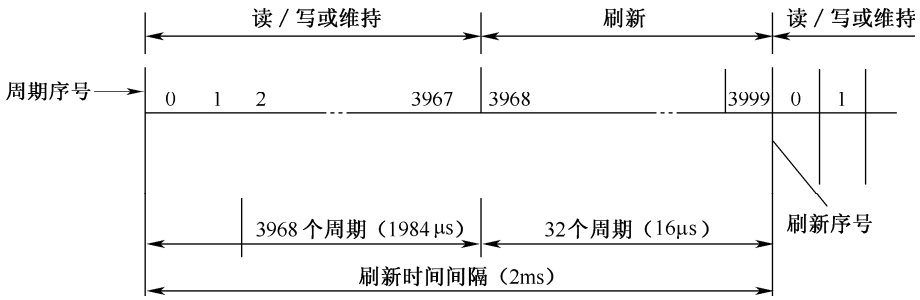


图 3-10　集中式刷新方式

刷新时间=存储体矩阵行数×刷新周期

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

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

如果对 32×32 的存储芯片进行刷新。该存储器的存取周期为 0.5μs，刷新周期为 2ms（占 4000 个存取周期）。采用集中式刷新方式，每行（32 个单元）占用一个周期，共需 16μs（32 个周期）完成全部单元的刷新，其余 1984μs 用来读/写或维持信息，如图 3-10 所示。由于在 16μs 时间内不能进行读写操作，故称为“死时间”。

（2）分散刷新方式。分散刷新是指把刷新操作分散到每个存取周期内进行，此时系统的存取周期被分为两部分，前一部分时间进行读/写操作或保持，后一部分时间进行刷新操作。一个系统存取周期内刷新存储矩阵中的一行，如图 3-11 所示。

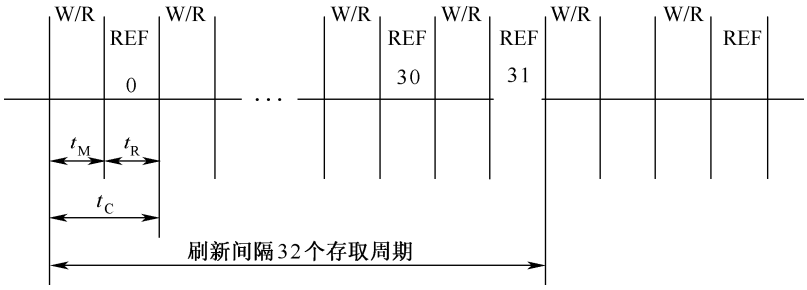


图 3-11　分散式刷新方式

分散刷新方式没有死区，这是它的优点，但是，它也有很明显的缺点：第一是加长了系统的存取周期，如存储芯片的存取周期为 0.5μs，则系统的存取周期应为 1μs，降低了整机的速度；第二是刷新过于频繁，尤其是当存储容量比较小的情况下，如 32×32 的矩阵，他们的刷新间隔只有 32μs，没有充分利用所允许的最大刷新间隔（2ms）。

（3）异步刷新方式。异步刷新方式（如图 3-12 所示）可以看成前述两种方式的结合，它充分利用了最大刷新间隔时间，把刷新操作平均分配到整个最大刷新间隔时间内进行，故有：

相邻两行的刷新间隔=最大刷新间隔时间/行数

对于 32×32 矩阵，在 2ms 内需要将 32 行刷新一遍，所以相邻两行的刷新时间间隔=2ms/32=62.5μs，即每隔 62.5μs 安排一个刷新周期，在刷新时封锁读/写。

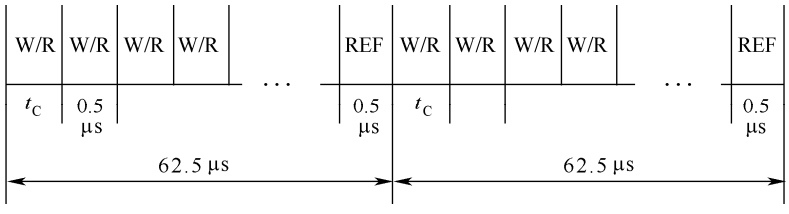


图 3-12　异步式刷新方式

异步刷新方式虽然也有死区，但比集中刷新方式的死区小得多，仅为 0.5μs。这样可以避

免使 CPU 连续等待过长的时间，而且减少了刷新次数，是比较实用的一种刷新方式。

3. 刷新控制

当刷新请求和访问存储器的请求同时发生时，应优先进行刷新操作。

MOS 型动态 RAM 的刷新要注意以下几个问题：

（1）刷新对 CPU 是透明的。

（2）刷新通常是逐行进行，每一行中各记忆单元同时被刷新，故刷新操作时仅需要行地址，不需要列地址。

（3）刷新操作类似于读出操作。

（4）因为所有芯片同时被刷新，所以在考虑刷新问题时，应当从单个芯片的存储容量着手，而不是从整个存储器的容量着手。

## 3.3　半导体只读存储器

只读存储器电路比 RAM 简单，因而集成度更高，成本更低。各类 ROM 均以非破坏性读出方式工作，而且是非易失性存储器。因此，半导体只读存储器 ROM 主要用来存放一些不需要修改的程序和数据，如一些系统软件和常数等，并可作为主存的一部分。

一种双极型的 ROM 存储单元电路如图 3-13 所示。字线通常处于低电平。若一个字被选中，则对应字线上的电压暂时升高，使得发射极连接到相应位线上的所有晶体管导通。从电源传到位线上的电流可由读出电路检测，有电流读出的单元读出 1，其余读出 0。所以发射极到位线连接的组合方式决定了给定字的内容。MOS 型电路也可组成类似的 PROM。

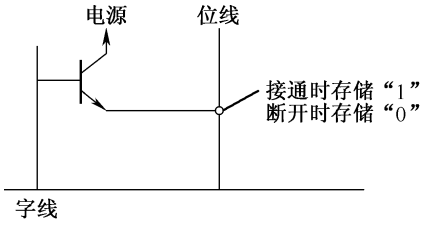


图 3-13　双极型的 ROM 存储单元电路

根据制造工艺的不同，半导体只读存储器可分为 ROM、PROM 和 EPROM 等几类。

掩膜只读存储器（ROM）是由制造厂家按照事先设计好的线路生产出来的，即所需的程序和数据是在制造时写入的，其存储的内容已经固化在 ROM 中，不能再修改。它适用于已经定型的、成批生产的产品，一般机器的自检程序、初始化程序，基本输入输出设备的驱动程序等都可放在 ROM 中。

可编程只读存储器（PROM）允许由用户把已调试好的程序和数据写入其中，但只能写入一次，写入之后就再也不能更改了。在发射极与位线之间接上熔丝即可做成 PROM。这时该存储器内容全部是 1。编程时需要哪些位为 0，只须用大电流脉冲将镕丝熔断即可。当然一旦熔断就不能再恢复了。这种存储器主要用于厂家针对用户的专门需要来烧制其中的内容。

可擦除可编程的只读存储器（EPROM）可以多次修改其中的内容，即允许擦除所存信息，然后再存放新的程序和数据，使用起来非常方便，在系统开发中得到了广泛的应用。根据擦除

方式的不同，又可分为紫外线擦除的 EPROM 和用电擦除的 E2PROM（一般称为 EEPROM）。在 EPROM 芯片上有一个石英玻璃窗口，紫外线通过这个窗口照在内部的硅片上，即可擦除原来存储在整个芯片的内容，EPROM 的写入是通过专用的 EPROM 写入器来完成的。E2PROM 是用电擦除，而且可以在线擦除，不必像 EPROM 那样把芯片在计算机、擦除器、写入器之间来回移动，解决了 EPROM 擦除信息不方便的缺点，另外，E2PROM 是按字节单元擦除信息，因此，写入时只需写入所需的单元即可，大大地节省了时间。

进人到 20 世纪 80 年代，又出现了一种闪速存储器（Flash Memory），称快擦型存储器，它是在 EPROM 和 EEPROM 工艺基础上产生的一种新型的存储器。与传统的固态存储器相比，Flash 存储器的主要特点如下：

（1）非易失性。这一特点与磁存储器相似，Flash 存储器不需要后备电源来保持数据。所以，它具有磁存储器无需电能保持数据的优点。

（2）易更新性。Flash 存储器具有电可擦除特点。相对于 EPROM（电可编程只读存储器）的紫外线擦除工艺，Flash 存储器的电擦除功能为开发者节省了时间，也为最终用户更新存储器内容提供了可能。

（3）成本低、密度高、可靠性好。与 EEPROM（电可擦除可编程的只读存储器）相比较，Flash 存储器的成本更低、密度更高、可靠性更好。

在需要周期性地修改存储信息的应用场合，闪速存储器是一个极为理想的器件，因为它至少可以擦写及编程 10,000 次，这足以满足用户的需要。它比较适合于作为一种高密度、非易失的数据采集和存储器件。在便携式计算机、工控系统及单片机系统中得到大量应用，近年来已将它用于微型计算机中存放输入输出驱动程序和参数等。

非易失性、长期反复使用的大容量闪速存储器还可替代磁盘。例如，在笔记本手掌型袖珍计算机中都大量采用闪速存储器做成固态盘替代磁盘，使计算机平均无故障时间大大延长，功耗更低，体积更小，消除了机电式磁盘驱动器所造成的数据瓶颈。

## 3.4 存储器的扩充

### 3.4.1 主存储器与 CPU 的连接

主存储器与 CPU 的连接时需要注意的几个问题：

1. 存储芯片的类型选择

RAM 最大的特点是其存储的信息可以在程序中用读/写指令随机读写，但掉电时信息丢失。所以 RAM 一般用于存储用户的调试程序（或程序存储器中的用户区）、程序的中间运算结果及掉电时无需保护（存）的 I/O 数据及参数等。

ROM 中的内容掉电不易失，但不能随机写入，故一般用于存储系统程序（监控程序）和无需在线修改的参数等。

2. CPU 总线的负载能力

通常 CPU 总线的直流负载能力（也称驱动能力）等同于一个 TTL 器件或 20 个 MOS 器件。因存储器基本上是 MOS 电路，直流负载很小，所以在小型系统中 CPU 可直接与存储器芯片连接。

而当 CPU 总线上需挂接的器件超过上述负载时，就应考虑在其总线与挂接的器件间加接缓冲器或驱动器，以增加 CPU 的负载能力。

3. CPU 的时序和存储器的存取速度之间的配合问题

CPU 在取指令和读写操作、存储器芯片读/写都有相应的固定时序。

选用存储芯片时，必须考虑它的存取时间与 CPU 的固定时序之间的匹配问题，即时序配合问题。

4. 存储芯片与 CPU 芯片连接时要特别注意的问题

（1）地址线的连接。

存储芯片的容量不同，其地址线数也不同，CPU 的地址线数往往比存储芯片的地址线数多。

通常总是将 CPU 地址线的低位与存储芯片的地址线相连。CPU 地址线的高位或在存储芯片扩充时用，或做其他用途，如片选信号等。

（2）数据线的连接。

同样，CPU 的数据线数与存储芯片的数据线数也不一定相等。此时，必须对存储芯片扩位，使其数据位数与 CPU 的数据线数相等。

（3）读/写命令线的连接。

CPU 读/写命令线一般可直接与存储芯片的读/写控制端相连，通常高电平为读，低电平为写。有些 CPU 的读/写命令线是分开的，此时 CPU 的读命令线应与存储芯片的允许读控制端相连，而 CPU 的写命令线则应与存储芯片的允许写控制端相连。

（4）片选线的连接。

片选线的连接是 CPU 与存储芯片正确工作的关键。存储器由许多存储芯片组成，哪一片被选中完全取决于该存储芯片的片选控制端 $\overline{\text{CS}}$ 是否能接收到来自 CPU 的片选有效信号。片选有效信号与 CPU 的访存控制信号 $\overline{\text{MRER}}$（低电平有效）有关，因为只有当 CPU 要求访存时，才需选择存储芯片。若 CPU 访问 I/O，则 $\overline{\text{MRER}}$ 为高电平，表示不要求存储器工作。此外，片选有效信号还和地址有关，因为 CPU 的地址线往往多于存储芯片的地址线，故那些未与存储芯片连上的高位地址必须和访存控制信号共同产生存储芯片的片选信号。通常需用到一些逻辑电路，如译码器及其他各种门电路，来产生片选有效信号。

常用的片选控制译码方法有线选法、全译码法。

1）线选法。当存储器容量不大，所使用的存储芯片数量不多，而 CPU 寻址空间远远大于存储器容量时，可用高位地址线直接作为存储芯片的片选信号，每一根地址线选通一块芯片，这种方法称为线选法。

2）全译码法。全译码法可以提供对全存储空间的寻址能力。当存储器容量小于可寻址的存储空间时，可从译码器输出。

### 3.4.2　位扩展

通常实际存储器容量比存储芯片的容量大。假定存储芯片容量为 mk×n 位/片，设计的存储器容量为 Mk×N 位。

根据实际应用情况，容量扩展有以下三种形式：位扩展、字扩展和字位同时扩展。

位扩展就是位数扩充，加大字长，以满足存储器字长的要求，而存储器的字数与存储芯片的字数一致。例如要组成 mk×N 位的存储器，需要 N/n 个 mk×n 位的存储芯片。

**例 1** 设有 32 片 256K×1 位的 SRAM 芯片。

（1）采用位扩展方法可构成多大容量的存储器？

（2）如果采用 32 位的字编址方式，该存储器需要多少地址线？

（3）画出该存储器与 CPU 连接的结构图，设 CPU 的接口信号有地址信号、数据信号和控制信号 MREQ#、R/W#。

**解**：（1）32 片 256K×1 位的 SRAM 芯片可构成 256K×32 位的存储器。

（2）如果采用 32 位的字编址方式，则需要 18 条地址线，因为 $2^{18}$=256K。

（3）用 MREQ#作为芯片选择信号，R/W#作为读写控制信号，该存储器与 CPU 连接的结构图如图 3-14 所示。

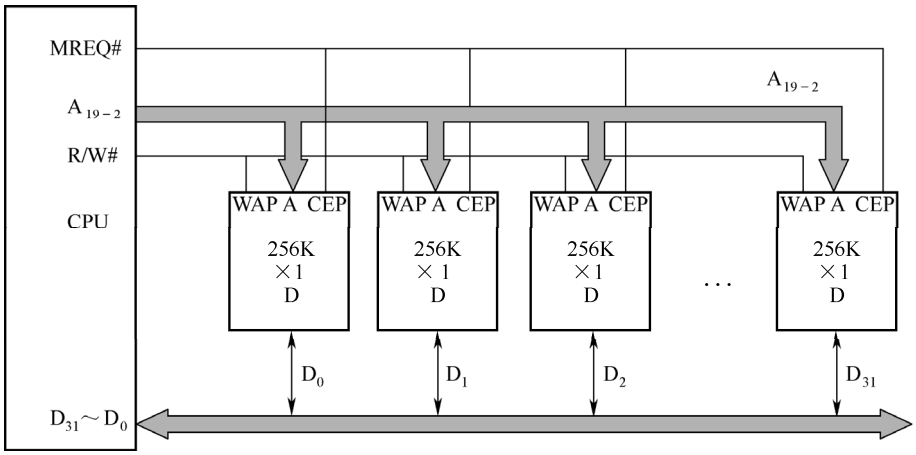


图 3-14 位扩展

### 3.4.3 字扩展

字扩展是增加存储器的字数，即存储单元个数，而存储器的位数即每个存储单元的位数与存储芯片的位数一致。字扩展通常是通过控制片选端来实现。例如存储器的容量为 Mk×n 位，则需要 M/m 个存储芯片。

**例 2** 设有若干片 256K×8 位的 SRAM 芯片。

（1）采用字扩展方法构成 2048KB 的存储器需要多少片 SRAM 芯片？

（2）该存储器需要多少地址线？

（3）画出该存储器与 CPU 连接的结构图，设 CPU 的接口信号有地址信号、数据信号和控制信号 MREQ#、R/W#。

**解**：（1）该存储器需要 2048K/256K=8 片 SRAM 芯片；

（2）21 条地址线，因为 $2^{21}$=2048K，其中高 3 位用于芯片选择，低 18 位作为每个存储器芯片的地址输入。

（3）用 MREQ#作为译码器芯片的输出许可信号，译码器的输出作为存储器芯片的选择信号，R/W#作为读写控制信号。CPU 访存的地址为 $A_{20}$～$A_0$。该存储器与 CPU 连接的结构图如图 3-15 所示。

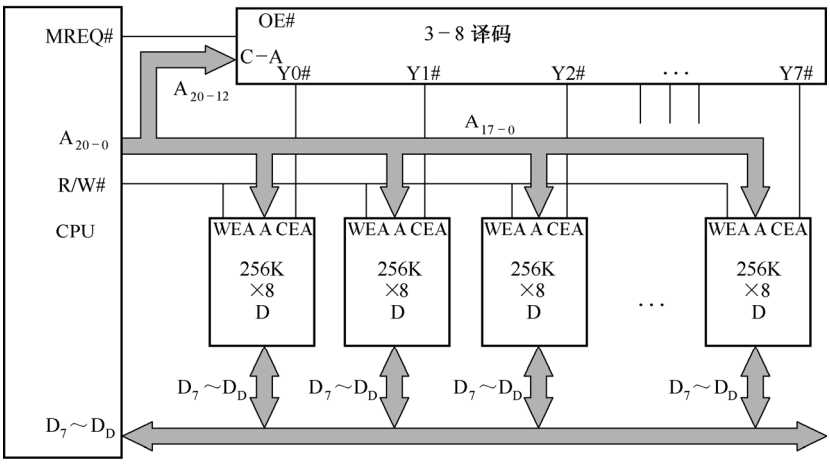


图 3-15　字扩展

### 3.4.4　字和位同时扩展

就是在选定存储芯片的基础上通过字和位同时扩展而成。

**例 3**　设有若干片 256K×8 位的 SRAM 芯片，请构成 2048K×32 位的存储器。

（1）需要多少片 RAM 芯片？

（2）该存储器需要多少地址线？

（3）画出该存储器与 CPU 连接的结构图，设 CPU 的接口信号有地址信号、数据信号和控制信号 MREQ#、R/W#。

**解**：（1）采用字位扩展的方法。该存储器需要(2048K/256K)×(32/8) = 32 片 SRAM 芯片，其中每 4 片构成一个字的存储器芯片组（位扩展），8 组芯片进行字扩展。

（2）用字寻址方式，需要 21 条地址线，其中高 3 位用于芯片选择，低 18 位作为每个存储器芯片的地址输入。

（3）因为存储器容量为 2048K×32=$2^{23}$KB，所以 CPU 访存的字地址为 $A_{22}$～$A_2$。用 MREQ# 作为译码器芯片的输出许可信号，译码器的输出作为存储器芯片的选择信号，R/W#作为读写控制信号，该存储器与 CPU 连接的结构图如图 3-16 所示。

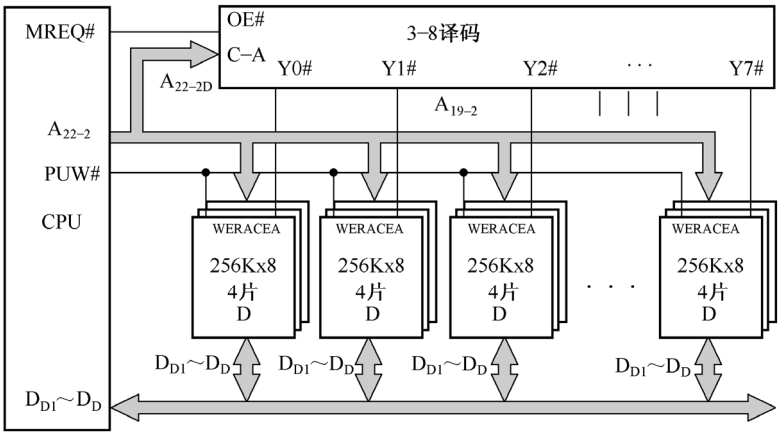


图 3-16　字和位同时扩展

## 3.5 高速存储器

随着计算机应用领域的不断扩大，处理的信息量越来越多，对存储器的工作速度和容量要求也越来越高。此外，主存和 CPU 之间在速度上是不匹配的，致使主存的存取速度已成为计算机系统的瓶颈。可见，提高访存速度也成为刻不容缓的任务。

为了使 CPU 不致因为等待存储器读写操作的完成而无事可做，可以采取一些加速 CPU 和存储器之间有效传输的特殊措施：

（1）在 CPU 内部设置多个通用寄存器；

（2）采用并行操作的存储器，双端口存储器和多模块交叉存储器；

（3）在 CPU 和主存之间插入 Cache；

（4）采用更高速的存储芯片。

### 3.5.1 双端口存储器

普通的存储器器件为单端口，也就是数据的输入输出只利用一个端口，设计了两个输入输出端口的就是双端口 SRAM。双端口存储器是指同一个存储器具有两组相互独立的读写控制线路,由于进行并行的独立操作，是一种高速工作的存储器。

如图 3-17 所示，双端口 RAM 提供了两个相互独立的端口，即左端口和右端口。它们分别具有各自的地址线、数据线和控制线，因而可以对存储器中任何位置的数据进行独立的存取操作。当两个端口的地址不相同时，在两个端口上进行读写操作，一定不会发生冲突。当任一端口被选中驱动时，就可对整个存储器进行存取，每一个端口都有自己的片选控制和输出驱动控制。

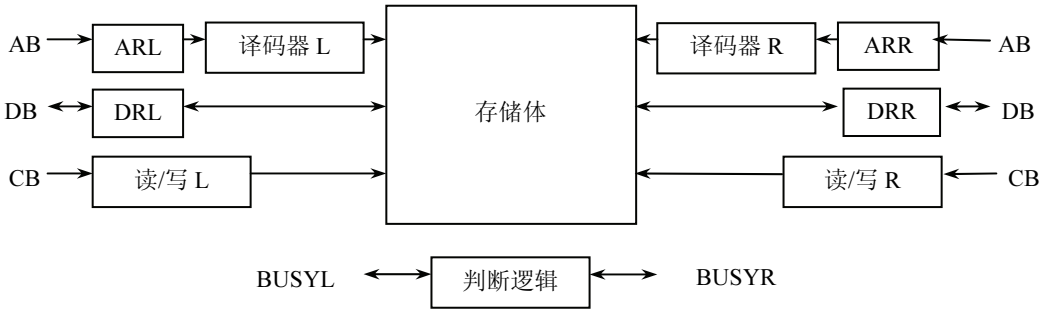


图 3-17 双端口存储器

（1）无冲突读写控制。

当两个端口的地址不相同时，在两个端口上进行读写操作，一定不会发生冲突。当任一端口被选中驱动时，就可对整个存储器进行存取，每一个端口都有自己的片选控制和输出驱动控制。

（2）有冲突的读写控制。

当两个端口同时存取存储器同一存储单元时，便发生读写冲突。为解决此问题，特设置了 BUSY 标志。由片上的判断逻辑决定对哪个端口优先进行读写操作，而暂时关闭另一个被

延迟的端口。

总之，当两个端口均为开放状态（BUSY 为高电平）且存取地址相同时，发生读写冲突。此时判断逻辑可以使地址匹配或片选使能匹配下降至 5ns，并决定对哪个端口进行存取。

双端口存储器的应用场合：

（1）实现 CPU 与 DMA（或 IOP）同时访问内存。

（2）在多机系统中，实现彼此间的信息交换。

（3）为运算器的两个输入端并行提供数据。

（4）双端口结构的 Cache，可同时与 CPU 和主存交换信息。

### 3.5.2　多模块存储器

1. 存储器的模块化组织

多模块存储器每个模块都有自己的地址寄存器，数据寄存器，读写控制电路及存储体，它们共用一个总线控制器以实现信息的输入输出。

在这种多模块存储器结构中，有两种编址方式：一种是顺序方式，一种是交叉方式。

顺序编址方式如图 3-18 所示，常规存储器多采用该设计。图中存储器容量为 32 字，分为 $M_0$，$M_1$，$M_2$，$M_3$ 四个模块，每个模块 8 个字。从 $M_0$ 模块开始顺序编址，接着为下一个模块分配地址。这样，存储器 32 个字可由 5 位地址寄存器指示，其中高位地址可以表示模块号，低位地址表示模块内地址。

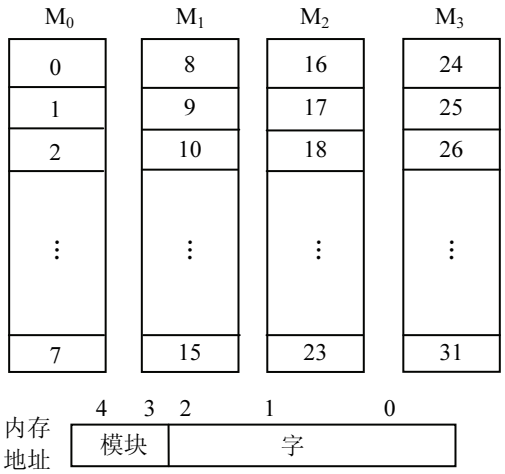


图 3-18　顺序编址

程序地址一般在低位连续推进，然后再传送到高位。程序选中一个模块后，等到该模块访问完毕后，才能跳到下一个模块工作。因此，只要调度合理，当一个模块用于执行程序时，另一个模块实现存储器与外部设备直接存储器访问（DMA）。在这种意义上讲，这种结构具有并行工作的能力，存储器总的吞吐量就提高了。这种编址方式由于一个体内的地址是连续的，有利于存储器的扩充，而且某块存储体出现故障不会影响其他存储器模块工作。

交叉编址方式如图 3-19 所示，即将单元地址依次排列在各个存储体中。如 $M_0$ 体的地址为 0,4,8,⋯,4i+0；$M_1$ 体的地址为 1,5,9,⋯,4i+1；$M_2$ 体地址为 2,6,10,⋯,4i+2；$M_3$ 体的地址为

3,7,11,…,4i+3。采用这样的编址方法，会使 4 个存储体对应的二进制地址最后两位的数码分布为它们的模块号 00，01，10 和 11。因而使用地址码的低位字段经过译码选择不同的存储体，而高位字段指向相应的存储体内部的存储字。这样连续地址分布在相邻的不同存储体内，而同一个存储体内的地址都是不连续的。在理想情况下如果程序段和数据块都是连续地在主存中存放或读取，那么将大大地提高共存的有效访问速度。

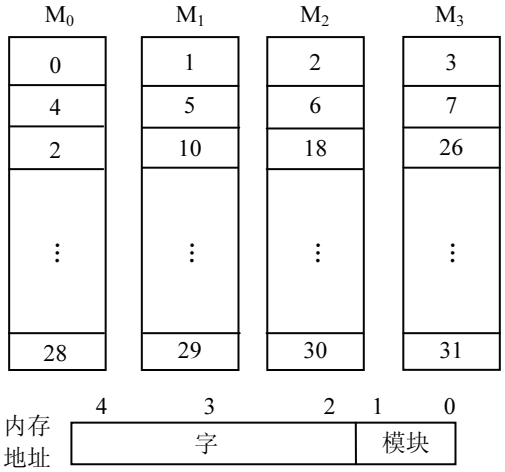


图 3-19　交叉编址

2. 多模块交叉存储器

多模块交叉存储器由于采用交叉编制，可以实现在不改变每个模块存取周期的前提下，提高存储器的带宽。如图 3-20 所示的四模块交叉存储器结构框图。每个模块都有自己的读写控制电路、地址寄存器和数据寄存器。CPU 同时访问四个模块，分时使用数据总线进行数据传递。这样，对每一个存储模块来说，从 CPU 给出访存命令直到读出信息仍然使用一个存取周期时间，而对 CPU 来说，它可以在一个存取周期内连续访问四个模块。

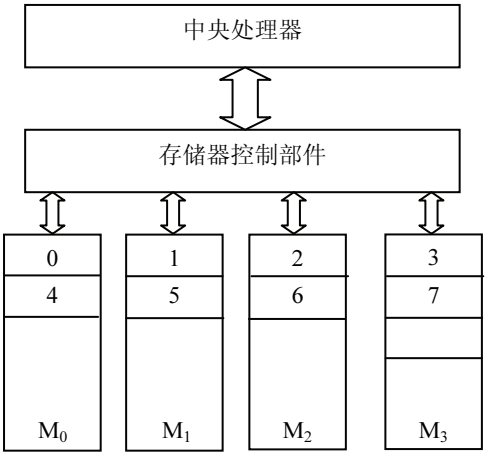


图 3-20　四模块交叉存储器结构图

假设每个体的存储字长和数据总线的宽度一致，并假设多体交叉存储器模块数为 m，存取周期为 T，总线传输周期为τ，即成块传送每经τ时间延迟后启动下一个模块。那么当采用流水线方式存取时，应满足 T=m*τ。为了保证启动某体后，经 m*τ时间再次启动该体时，它的上次存取操作已完成，要求低位交叉存储器的模块数大于或等于 m。以四体低位交叉编址的存储器为例，采用流水方式存取的示意图如图 3-21 所示。

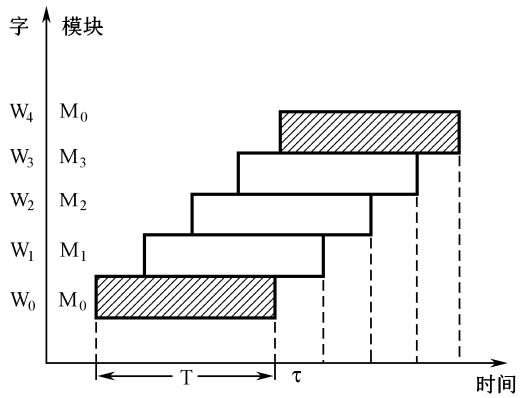


图 3-21　流水线方式存取示意图

**例 4**　设存储器容量为 64 字，字长 32 位，模块数 m=4，分别用顺序方式和交叉方式进行组织。存储周期=200ns，数据总线宽度为 64 位，总线传送周期τ=50ns。问顺序存储器和交叉存储器的带宽各是多少？

**解**：顺序存储器和交叉存储器连续读出 m=4 个字的信息总量都是：

$$q = 32M \times 4 = 128\text{位}$$

（1）顺序方式和交叉方式读出 4 个字所需时间分别是

$$t_1 = mT = 4 \times 200 = 800 \quad (\text{ns})$$

$$t_2 = T + (m-1)\tau = 200 + 3 \times 50 = 350 \quad (\text{ns})$$

（2）顺序方式和交叉方式的带宽分别是

$$B_1 = q / t_1 = 128 \div (800 \times 10 - 9) = 16 \times 107 \quad (\text{位/秒})$$

$$B_2 = q / t_2 = 128 \div (350 \times 10 - 9) = 36.5 \times 107 \quad (\text{位/秒})$$

## 3.6　高速缓冲存储器（Cache）

### 3.6.1　Cache 的基本原理

1. Cache 的功能

为弥补主存速度的不足以便使主存更好地与高速 CPU 匹配，在 CPU 与主存之间需设置一个速度较高、容量较小的缓冲存储器（Cache），构成 Cache－主存存储层次，目标是使得从 CPU 执行指令的角度来看，存储器速度接近于 Cache，而容量接近于主存。从功能上看，它是主存的缓冲存储器，由高速的 SRAM 组成。为了追求高速，包括管理在内的全部功能由硬件实现，能够实现对程序员的透明性。

随着半导体器件集成度的进一步提高，CPU 内部集成了 1~2 个 Cache，称为片内 Cache，而 CPU 外的称为片外 Cache。某些机器甚至有二级三级缓存，每级缓存比前一级缓存速度慢且容量大。而这时，一开始的高速小容量存储器就被人称为一级缓存。

Cache 的出现使 CPU 可以不直接访问主存，而与高速 Cache 交换信息。通过大量典型程序的分析,发现从主存取指令或取数据，在一定时间内，只是对主存局部地址区域的访问。由于指令和数据在主存内都是连续存放的，并且由于子程序、循环程序和一些常数的存在有些指令和数据往往会被多次调用，即指令和数据在主存的地址分布不是随机的，而是相对的集中，使得 CPU 在执行程序时，访问内存具有相对的局部性，这就称为程序访问的局部性原理。根据这一原理，只要将 CPU 近期要用到的程序和数据提前从主存送到 Cache，那么就可以做到 CPU 在一定时间内只需访问 Cache。一般 Cache 采用高速的 SRAM 制作，其价格比主存贵，但因其容量远小于主存，因此能很好地解决速度和成本的矛盾。

2. Cache 的命中率

CPU 与 Cache 以及 CPU 与主存之间的数据交换是以字为单位，而 Cache 与主存之间的数据交换是以块为单位。为了进行数据交换，主存与缓存都分成若干块，每块内又包含若干个字，并使它们的块大小相同（即块内的字数相同）。由于 Cache 容量小于主存，所以 Cache 包含的块数小于主存块数。

任何时刻都有一些主存块处在缓存块中。欲读取主存某字时，有两种可能：一种是所需要的字已在缓存中，即可直接访问 Cache，这种情况为 CPU 访问 Cache 命中；另一种是所需的字不在 Cache，这种情况为 CPU 访问 Cache 不命中，此时需将该字所在的主存整个字块一次调入 Cache 中。

Cache 命中率是指 CPU 要访问的信息已经在 Cache 中的比率。Cache 的容量与块长是影响 Cache 命中率的重要因素。从 CPU 来看，增加 Cache 的目的就是在性能上使主存的平均读出时间尽可能接近 Cache 的读出时间，这就要求 Cache 的命中率应尽量接近于 1。由于程序访问的局部性，实现这个目标是可能的。

在一个程序执行期间，设 $N_c$ 表示 Cache 完成存取的总次数，$N_m$ 表示主存完成存取的总次数，h 定义为命中率，则有

$$h = \frac{N_c}{N_c + N_m}$$

3. Cache 的基本结构

Cache 的基本结构如图 3-22 所示，它由 Cache 控制器和 Cache 数据存储器两大部分组成。Cache 控制器包括 Cache－主存地址映像变换机构、替换控制及总线控制。Cache 数据存储器由高速 SRAM 组成，用于存放 CPU 所需要的、从主存复制过来的信息。Cache－主存地址的变换机构与映像方式有关。地址映像变换机构内含地址映像表，其任务是接收主存地址中的标记和块号信息并与内部的映像表作比较，当命中时输出 Cache 地址的块号，不命中时对 Cache 块替换逻辑输出“不命中”信号。

Cache 的工作原理如下所示：

（1）CPU 通过地址总线给出访问主存的字地址；

（2）主存地址中的块号及标记信息输入到 Cache 地址映像变换结构，与其内部的地址映像表的有关项目进行相联比较，以判定该访问字是否在 Cache 中；

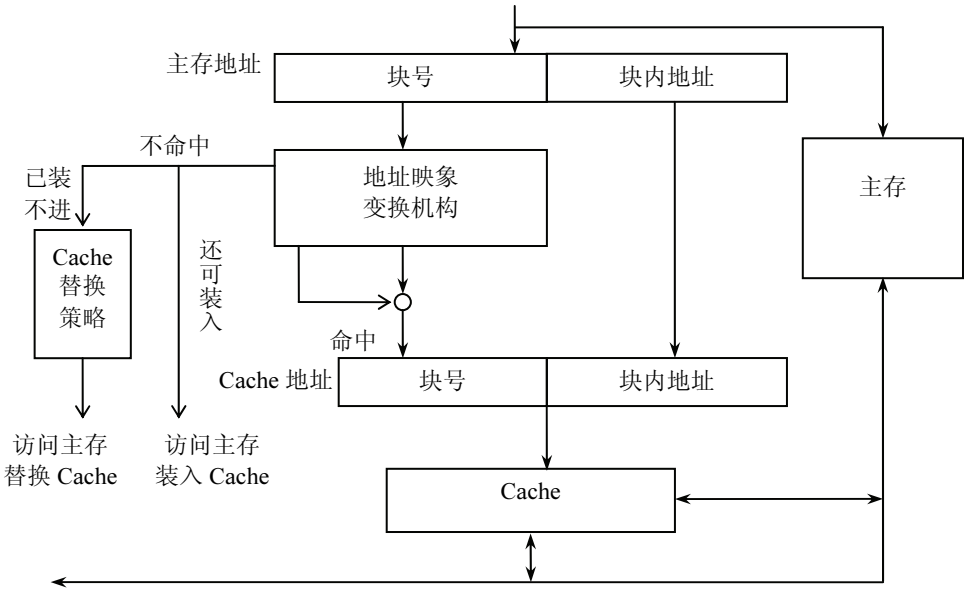


图 3-22　Cache 的基本结构

（3）如果访问字在 Cache 中（命中），则将块号及主存地址中的块内地址送入 Cache 地址寄存器，以该地址访问 Cache 数据存储器，并与 CPU 进行单字宽的信息传送；

（4）如果访问字不在 Cache 中（不命中），则块替换及控制逻辑检测是否还有块空闲，若有，则访主存并通过多字宽通路将包含该字的一块信息调入 Cache；

（5）如果访问字不在 Cache 中，Cache 数据存储器又已装满，即发生块冲突、这就需要按既定的替换算法选择被替换的块，并访主存调入新的块将其替换，然后修改地址映像表；

（6）对 Cache 块写入的信息，必须与被映射的主存块内的信息完全一致。目前主要采用写直达法和写回法。

### 3.6.2　地址映射

为了把信息放到 Cache 存储器中，必须应用某种函数把主存地址映像到 Cache，称作地址映像。在信息按照这种映像关系装入 Cache 后，执行程序时，应将主存地址变换成 Cache 地址，这个变换过程叫做地址变换。地址的映像和变换是密切相关的。

1. 直接映像

在直接映像方式中，主存和 Cache 中字块的对应关系如图 3-23 所示。

Cache 块号 i（$0 \leqslant i \leqslant 2^c - 1$）与主存块号 j（$0 \leqslant i \leqslant 2^m - 1$）之间有如下关系：

$$i = j \bmod 2^c$$

显然，主存块与缓存块是多对一的关系。主存的第 $0, C(C = 2^c), \ldots, 2^m - C$ 块只能映射到 Cache 的第 0 行；主存的第 $1, C+1, \ldots, 2^m - C + 1$ 块只能映射到 Cache 的第 1 块。

在直接映射方式中，主存地址被分为如图 3-14 所示三部分：最低字段为块内字地址共 b 位（主存和缓存块同样包含 $2^b$ 个字），中间字段为映射的 Cache 字块地址共 c 位，最高字段是主存字块标记共 $m - c$ 位，它被记录在建立了对应关系的缓存块的“标记”位中。

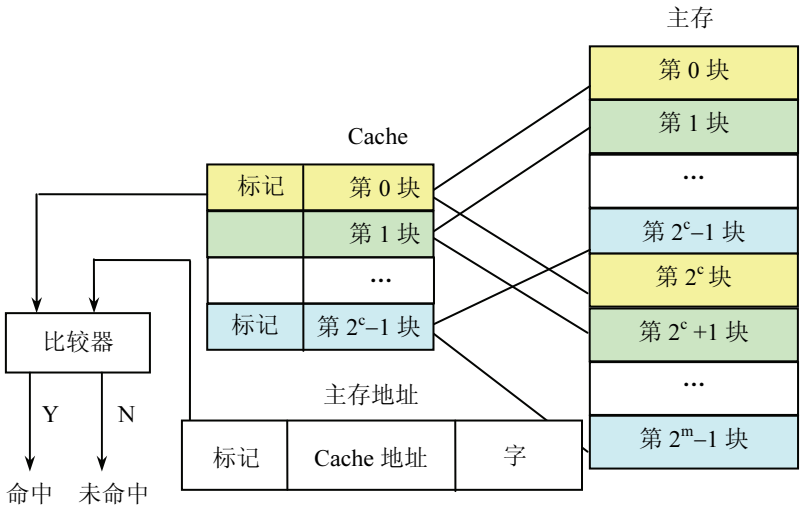


图 3-23　直接映像示意图

当 CPU 以一个给定的主存地址访问时，用中间的 c 位字段（若为 00…01）找到 Cache 的第 1 块，然后比较缓存字块 1 中的标记是否与主存地址中的高 m-c 位是否一致，如果一致则命中，否则未命中。

直接映像的优点是实现简单，只需利用主存地址按某些字段直接判断，即可确定所需字块是否已在 Cache 存储器中。

直接映像方式的缺点是不够灵活，即主存的每个字块只能对应唯一的 Cache 存储器字块，因此，即使 Cache 存储器别的许多地址空着也不能占用。这使得 Cache 存储空间得不到充分利用，并降低了命中率。

2. 全相联映像

全相联映像方式是最灵活但成本最高的一种方式，如图 3-24 所示。它允许主存中的每一个字块映像到 Cache 存储器的任何一个字块位置上，也允许从确实已被占满的 Cache 存储器中替换出任何一个旧的字块，这是一个理想的方案。

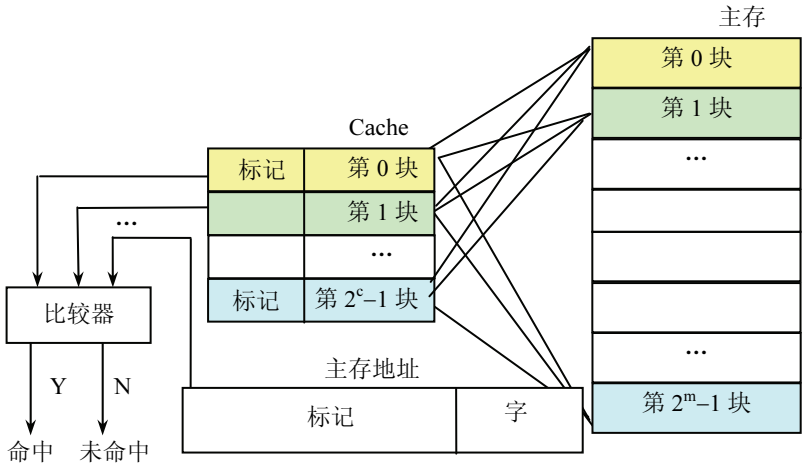


图 3-24　全相联映像示意图

与直接映射相比，它的主存字块标记即主存块号，这就使 Cache“标记”的位数增多，而且访问 Cache 时主存字块标记需要和 Cache 的全部“标记”位进行比较，才能判断出所访问主存地址的内容是否已在 Cache 内。这种比较通常采用“按内容寻址”的相联存储器来完成。

3．组相联映像

组相联映像方式是直接映像和全相联映像方式的一种折衷方案。按这种映像方式，组间的字块为直接映像，而组内的字块为全相联映像方式。组相联映像方式的性能与复杂性介于直接映像与全相联映像两种方式之间。

Cache 字块地址字段变为组地址字段 q 位，且 $q = c - r$。其中 $2^c$ 表示的总块数，$2^q$ 表示 Cache 的分组个数，$2^r$ 表示组内包含的块数。为了便于理解，假设 c=5，q=4，则 r=1。其实际含义为 Cache 共有 32 个字块，共分为 16 组，每组内包含 2 块。组内 2 块的组相联映射又称为二路组相联。

缓存组号 i 与主存块号 j 之间的关系为：

$$i=j \bmod 2^q$$

某一主存模块按照模 $2^q$ 映射到缓存的第 i 组内，如图 3-25 所示。

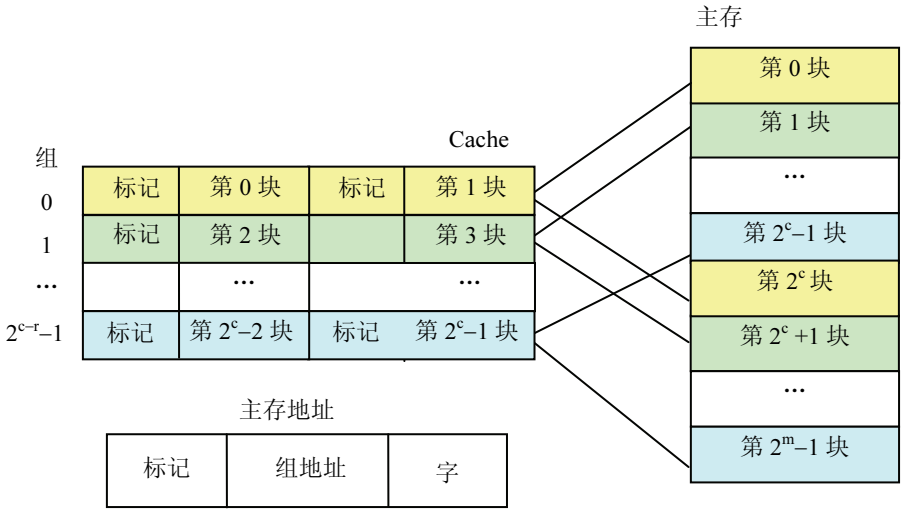


图 3-25　组相联映射示意图

根据上述假设条件，组相联映射的含义是：主存的某一字块可以按模 16 映射到 Cache 某组的任一字块中。即主存的第 0,16,32,…字块可以映射到 Cache 第 0 组 2 个字块中的任一字块；主存的第 15,31,47,…字块可以映射到 Cache 第 15 组中的任一字块。显然，主存的第 j 块会映射到 Cache 的第 i 组内，两者之间一一对应，属直接映射关系；另一方面，主存的第 j 块可以映射到 Cache 的第 i 组内中的任一块，这又体现出全相联映射关系。可见，组相联映射的性能及其复杂性介于直接映射和全相联映射两者之间，当 r=0 时是直接映射方式，当 r=c 时是全相联映射。

### 3.6.3　替换算法

在采用全相联映像和组相联映像方式，从主存向 Cache 传送一个新的块，而 Cache 中的

可用位置已被占满时，就产生了替换算法的问题，常用的方法有下述两种：

（1）先进先出（FIFO）算法。FIFO 算法的思想是：按照调入 Cache 的先后决定淘汰的顺序，即在需要更新时，将最先进入 Cache 的块作为被替换的块。这种方法不需要随时记录各个块的使用情况，容易实现，而且系统开销小。其缺点是可能会把一些需要经常使用的程序块（如循环程序）也作为最早进入 Cache 的块替换掉。

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

### 3.6.4 Cache 的写策略

（1）Cache 的读操作。当 CPU 发出读请求时，如果 Cache 命中，就直接对 Cache 进行读操作，与主存无关；如果 Cache 未命中. 则仍需访问主存，并把该块信息一次从主存调入 Cache 内，若此时 Cache 已满，则需根据某种替换算法，用这个块替换掉 Cache 中原来的某块信息。

（2）Cache 的写操作。当 CPU 发出写请求时，如果 Cache 命中，会遇到如何保持 Cache 与主存中的内容一致的问题，处理的方法主要有：

1）写直达法。即同时写入 Cache 和主存。这种方法实现简单，而且能随时保持主存数据的正确性。但可能增加多次向主存不必要的写入，降低存取速度。

2）写回法。就是信息暂时只写入 Cache，并用标志将该块加以注明，直到该块从 Cache 中替换出来时，才一次写入主存。这种方法操作速度快，但由于主存中的字块未经随时修改而有可能出错。

## 3.7 虚拟存储器

### 3.7.1 虚拟存储器的基本概念

在当前的计算机（包括微型机）中，其可寻址空间远远大于实际配置的主存的容量。例如 386PC 机，其地址位为 32，则可寻址的空间为 $2^{32}$=4GB，但实际配备的主存仅有 8MB 或 16MB 等。另一方面程序设计人员希望有一个大干（或等于）整个主存的空间供编程使用。这样就提出了虚拟存储器的概念。

由操作系统将辅存的一部分当作主存使用，因而扩大了程序可控制的空间。通常这种由主存和部分辅存组成的存储系统称为虚拟存储器。或者说，在主存和辅存之间，增加部分软件和必要的硬件支持，使主存和辅存形成一个有机整体，称为虚拟存储器，简称虚存。

从原理角度看，虚拟存储器和 Cache－主存体系有很多相似之处。虚存所采取的映像方式和 Cache－主存一样，也具有全相联映像，直接映像和组相联映像三种方式。两者也都采用 LRU 替换算法，即最近最少使用算法，即把存储器最近最少使用的存储块替换出去，以便将新的存储块调入。

但两者也有明显区别，主存－Cache 体系访问时间少，传送的信息块也小；虚存访问时间长，传送的信息块（段、页）比较大。

通常把能访问虚拟空间的指令地址码称为虚拟地址或逻辑地址，而把实际的地址称为物理地址或实存地址。物理地址对应的存储容量称为主存容量或实存容量。同时，为协调程序的局部性和存储区间管理，可以按程序的模块大小将存储器分割成不定长的块－段，也可以将存储器分割成定长的块－页，或者将段、页结合。因而可以形成常用的段式、页式、段页式多种虚拟存储器。下面分别予以介绍。

### 3.7.2　页式虚拟存储器

以页为基本信息传送单位的虚拟存储器称之为页式虚拟存储器。页式虚拟存储器把虚存空间、主存空间和辅存空间都分成固定大小的块，称为页面，简称页。各类计算机页面的大小设置不同，一页最少 512B，最大几 KB，通常为 2 的整数次幂。CPU 对虚拟存储器的每次访问，都要进行虚地址、实地址的变换。当所需信息不在主存时，还需将信息所在页从辅存调入主存，此时需将虚地址变换为辅存实地址。为实现这两种变换和页调度，需内页表、页式快表、外页表和主存页面表等多种数据结构。

1. 页表

页式虚拟存储器，实存地址由实页号和页内地址组成，页内地址为低位，位数由页面大小决定；实页号为高位，位数取决于主存容量。如主存容量为 128KB，字节编址，每页 2KB，则页内地址占低 11 位，实页号占高 6 位，该主存分为 64 个实页。虚存空间也按主存页面大小划分，虚地址由虚页号和页内地址组成，且虚存、实存页内地址位数相等，因虚存空间比主存空间大得多，所以虚页号位数比实页号位多。CPU 访问页式虚拟存储器时，送出的是程序虚地址，此时必须判断地址中的存储内容是否已调入主存，若未调入，则将该内容所在页按某种替换算法装入主存指定页，然后 CPU 才能执行；若已装入主存，就要找出在主存哪一页，这两种情况都要求建立一张虚页号与实页号的对照表，这张表称内页表，简称页表（Page table）。

页表是操作系统根据程序运行情况建立的，对应用程序员完全透明。页表存放在系统固定区域，每个程序都有一张页表，如表 3-1 所示。页表由页表项构成，程序的每个虚页对应一个页表项，记录与该页有关的信息。通常页表项应包括装入位、修改位、替换控制位、实页号和其他控制位。

表 3-1　页表结构

| 装入位 | 修改位 | 替换控制位 | 实页号 | 其他控制位 |
|---|---|---|---|---|
| 1 | | | 3 | |
| 1 | | | 5 | |
| 1 | | | 2 | |
| 1 | | | 8 | |

页表放在主存中，当 CPU 访存时，首先得访问页表，以实现虚、实地址变换，如图 3-26 所示。

2. 虚地址和实地址的映射

**例 1**　某计算机的页式虚存管理中采用长度为 32 字的页面。页表内容如表 3-2 所示，求当 CPU 程序按下列二进制虚拟字地址访存时产生的实际字地址。

（1）00001101。

（2）10000000。

（3）00101000。

**解**：页面长度为 32 字，则页内地址 5 位，8 位地址码中的高 3 位为虚页号，从表中查出 2 位实页号，与页内地址合并构成 7 位实际物理内存的地址。

虚页号为 000，查得实页号 01，与页内地址 01101 合并，得 0101101。

虚页号为 100，查得实页号 10，与页内地址 00000 合并，得 1000000。

虚页号为 001，查得该页未装入内存，没有相应的内存地址。

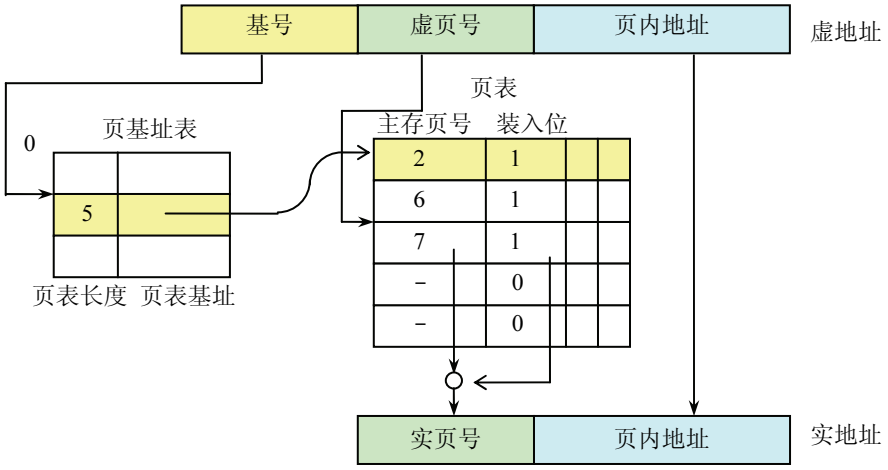


图 3-26　页式虚拟存储器虚实地址的变换

**表 3-2　页表内容**

| 虚页号 | 实页号 | 装入位 |
|---|---|---|
| 000 | 01 | 1 |
| 001 | - | 0 |
| 010 | 11 | 1 |
| 011 | 00 | 1 |
| 100 | 10 | 1 |
| 101 | - | 0 |
| 110 | - | 0 |
| 111 | - | 0 |

3. 页式虚拟存储器的优缺点

页式虚拟存储器页长度固定且可顺序编号，页表设置方便。页表可按页提供虚实映像关系，因此一个程序所占的各实页之间不必连续。当一个程序运行结束，所释放的页又可以以页为单位分配给其他程序。显见，因而页式虚拟存储器有利于存储空间的利用与调度，操作简单，且开销少。

但由于页长度固定，程序不可能正好是页面的整数倍，最后一页的零头无法利用而可能造成浪费。同时，机械地划分页面很难反映程序的逻辑结构。在逻辑上独立的程序模块本应作为一个整体处理，却可能被机械地划分开，甚至出现一条指令或一组相关数据跨页的情况。这就增加了查页表次数和页面失效的可能性，同时也会给保护、共享及其他存取控制带来麻烦。

### 3.7.3　段式虚拟存储器

编制程序大都采用模块化，复杂程序按逻辑可分成一系列相互关联且功能独立的简单模块。程序的执行过程也是从一个功能模块转到另一个功能模块的过程。段式虚拟存储器是适应模块化程序设计的一种结构，虚存和主存空间不再是机械地按固定的页划分，而是依程序的逻辑功能而定。每一段可以是主程序段、通用数据段、专用数据段、子程序段、堆栈段等，显然各段长度可能不相等。一般情况下，编程使用的虚地址由高位段号和低位段内地址两部分构成。

1. 段表

段式虚拟存储器需设置段表。每一程序都有一张段表。段表由段表项构成，程序的每一段对应一个段表项，记录该段的有关信息。系统在主存固定区域存放段表，在程序装入时填写段表。段表项通常包括装入位、段长、其他控制位和主存始地等内容。装入位为“1”表示该段已装入主存；为“0”，表示该段尚未装入主存。主存始地指出该段装入主存后的起始地址。段长给出该段程序的长度，以便在主存选择适当的空间定位。其他控制位主要为操作系统提供必要的信息，可用来指出段的类型，如是数据段、程序段或是零段区，也可作为保护码，共享控制等。

2. 虚地址和实地址的映射

虚实地址和变换如图 3-27 所示。

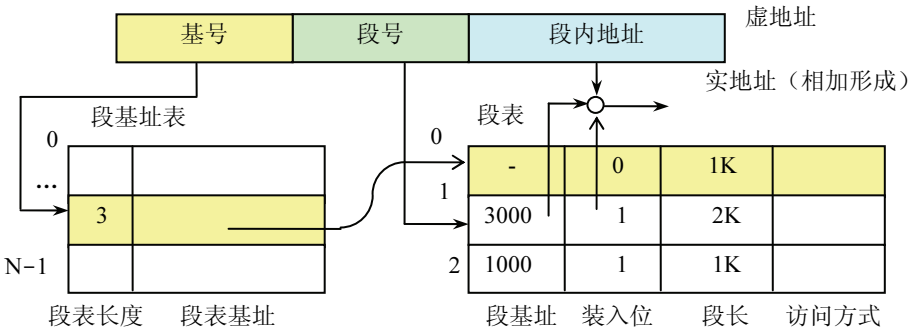


图 3-27　段式虚拟存储器虚实地址的变换

3. 段式虚拟存储器的优缺点

段式虚拟存储器面向程序的逻辑结构分段，段独立编址可大可小，因此程序可由多人分段并行编写。程序可以分段调试，思路清晰，容易检查错误，段的修改、增删对其他段也不会产生影响。存储空间的分段外以段为单位进行调度、传送、定位使得程序执行时命中率高且利于程序的共享与保护。

段式虚拟存储器段的大小可变，导致地址变换、存储空间的管理与调度都比较复杂。如段内信息必须连续存放，而各段首、尾地址又没有一定规律，访主存的地址需相加才能求得。

又如，当一个段的程序执行完，若新调入的程序段远小于现有的段空间时，段间就出现较大零头而造成较大浪费。若新段稍大于现有的段空间，则不能装入新段，程序不得不挂起，显著降低了计算机效率。

### 3.7.4 段页式虚拟存储器

段页式虚拟存储器对主存空间的管理与安排同页式虚拟存储器，而对逻辑空间则先依程序的逻辑结构分段，然后每一段再依主存空间页的大小划分成页。每个程序设一个段表，每段都有一张页表。段表由段表项构成，程序的每个段对应一个段表项，记录该段有关信息。程序的每一页都对应一个页表项，记录该页的有关信息。

系统在主存固定区域存放段表和页表，装入程序时填写段表和页表。

段页式虚拟存储器虚地址、主存实地址变换，首先通过段表查相应段的页表始地，再通过页表找到主存实页号，最后和页内地址排接成访主存的实地址。其变换过程如图 3-28 所示。

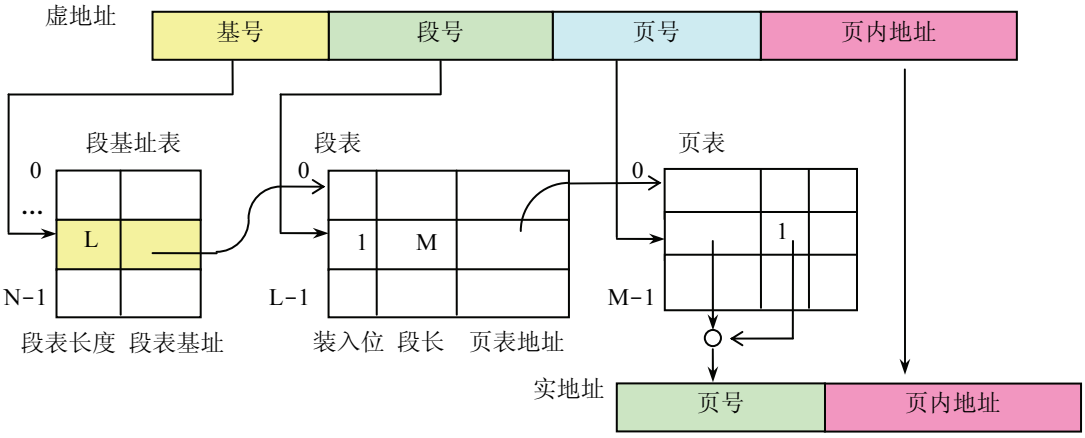


图 3-28 段页式虚拟存储器虚实地址的变换

段页式虚拟存储器兼有段式和页式优点，很多计算机系统采用它。对应用程序员而言，编程方法与段式完全相同，其所面向的就是段式虚拟存储器。而程序段如何分页、页的大小，页的调入、调出、传送等问题完全是系统程序员的事。因此段页式虚拟存储器的调度、管理将比页式、段式都会更复杂。


对存储器的要求是容量大、速度快、成本低。为了解决这三方面的矛盾，计算机采用多级存储体系结构，即 Cache、主存和外存。CPU 能直接访问内存（Cache、主存），但不能直接访问外存。存储器的技术指标有存储容量、存取时间、存储周期、存储器带宽。

广泛使用的 SRAM 和 DRAM 都是半导体随机读写存储器，前者速度比后者快，但集成度不如后者高。二者的优点是体积小，可靠性高，价格低廉，缺点是断电后不能保存信息。只读存储器正好弥补了 SRAM 和 DRAM 的缺点，即使断电也仍然保存原先写入的数据。特别是闪

速存储器能提供高性能、低功耗、高可靠性以及瞬时启动能力，因而有可能使现有的存储器体系结构发生重大变化。

单片存储芯片的容量总是有限的，利用若干存储芯片连在一起构成足够容量的存储器可以采用位扩展、字扩展和字位扩展法。另外，存储器在与 CPU 连接时，要注意地址线、数据线和控制线的连接方法。

双端口存储器和多模块交叉存储器属于并行存储结构。前者采用空间并行技术，后者采用时间并行技术。

Cache 是一种高速缓冲存储器，是为了解决 CPU 和主存之间速度不匹配而采用的一项重要的硬件技术，并又发展为多级 Cache 体系。要求 Cache 的命中率接近于 1。主存与 Cache 的地址映射有全相联、直接、组相联三种方式。其中组相联方式是前二者的折衷方案，适度地兼顾了二者的优点又尽量避免其缺点，从灵活性、命中率、硬件投资来说较为理想，因而得到了普遍采用。

虚拟存储器指的是主存—外存层次，它给用户提供了一个比实际主存空间大得多的虚拟地址空间。因此虚拟存储器只是一个容量非常大的存储器的逻辑模型，不是任何实际的物理存储器。按照主存外存层次的信息传送单位不同，虚拟存储器有页式、段式、段页式三种。


1．存储器的主要功能是什么？为什么要把存储系统分成若干个不同层次？主要有哪些层次？

2．什么是半导体存储器，它有什么特点？

3．存储器的地址译码方式有几种？试分析它们各自的特点和应用场合。

4．一般存储芯片都设有片选信号/CS，它有什么用途？

5．什么是高速缓冲存储器？它与主存是什么关系?其基本工作过程如何？

6．什么叫虚拟存储器？采用虚拟存储技术能解决什么问题？

7．假定有两种静态 RAM 芯片：1K×1 位 32 片，4K×1 位 8 片，RAM 芯片有/CS 和/WE 信号控制端；CPU 控制信号有 R/WE（读/写）和/MREQ（当存储器进行读或写操作时，该信号指示地址总线上的地址是有效的）。试用这些芯片构成 4K×16 位的存储器（要求所画出的 RAM 与 CPU 连接图能表明所用扩展法的三总线具体连接方法）。