目录

第3章　存储器系统

存储器是计算机中用于保存信息的部件。计算机中存储器系统由处于不同位置的多个存储部件组成，如在 CPU 内部有寄存器和高速缓存、在主机内部有主存储器、在主机外部有外部辅助存储器。通常主存储器通过系统总线与 CPU 连接，外部辅助存储器通过特定的接口电路和计算机主机连接。这些不同的存储部件协同工作，向 CPU 提供数据或保存来自 CPU 的数据。

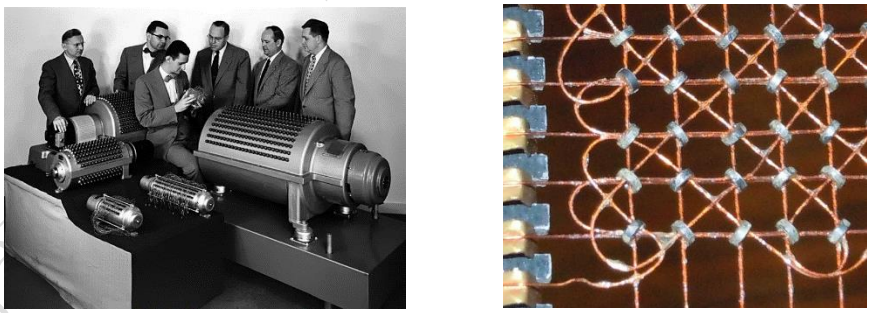
本章介绍不同类型存储器件的原理及其在计算机层次化存储系统中的应用；阐述存储器地址空间的概念，分析存储器芯片与 CPU 连接时需要考虑的问题；阐述高速缓存（即高速缓冲存储器）工作原理及关键技术问题；讨论基于虚拟存储器的存储器管理技术。

3.1 概述

3.1.1 存储器的类型

1. 磁介质存储器

磁介质存储器（Magnetic Memory）以磁性材料作为信息存储的基本单元，利用磁性材料两种不同的磁化状态可以表征二进制的“0”和“1”，属于非易失性存储器。早期出现过磁泡（Bubble Memory）、磁鼓（Magnetic Drum Memory）和磁芯[1]（Magnetic Core）等形式，后来主要采用磁盘、磁带等磁表面存储器。



(a) 早期计算机主存采用的磁鼓　　(b) 二十世纪 50 年代诞生的磁芯存储器

图 3.1 磁鼓和磁芯存储器

如今计算机中仍广泛使用的硬磁盘 HDD（Hard Disk Drive，也称机械硬盘，或简称硬盘），就是在一片或者多片金属盘片表面覆盖磁性材料，由磁头随机存取表面被磁化的不同信息。整块硬盘包括盘片、磁头、磁盘旋转机构和控制器，由控制器驱动盘片高速运动，通过磁头沿径向运动存取盘片特定位置的信息。软磁盘 FDD（Floppy Disk Drive，简称软盘）的存取原理与硬磁盘类似，不过磁性材料覆盖在柔软聚酯材料制成的塑料圆盘上。

[1] 1948 年美籍华人王安（即王安电脑公司的创始人）开发出“读后即写”的技术，使磁芯存储的应用成为可能。在二十世纪 50 年代至 70 年代之间，计算机的主存均采用磁芯存储器。

磁带存储器中磁性材料覆盖在柔软的带状介质上。读写磁带的磁带机由磁带传送机构、磁头、读写电路及控制电路组成。磁带传送机构驱动磁带相对磁头运动，从而磁头可读写磁带不同位置的磁介质信息。磁带存储器不仅在计算机中使用，在二十世纪的 80 年代前后，录音机、录像机等家用消费电子产品也广泛使用磁带存储技术。

2. 半导体存储器

半导体存储器（Semi-conductor Memory）指以半导体器件作为存储介质的存储器。主要分为随机存储器 RAM 和只读存储器 ROM 两种类型。RAM 型存储器的读写速度较快，但掉电之后所存储的内容就消失了；而 ROM 型存储器具有非易失性，掉电后信息不丢失。鉴于 RAM 和 ROM 的特点，RAM 常用于存储运行程序及与之相关的动态数据；而 ROM 一般用于存储一些固定的数据或者程序。

早期的 ROM 存储器只能读取数据（Read Only），后来 ROM 器件逐渐发展出可一次性写入、多次写入等新特性。从读写特性角度看，现在的常用的 ROM 已经不再是只读的存储器了，用“非易失性存储器 NVM（non-volatile memory）”来描述更为合适，但习惯上很多资料中仍使用 ROM 这一称呼。

半导体存储器具有体积小、存储速度快、存储密度高等特点。由于本身就是半导体器件，存储单元电路与输入输出接口电路很容易集成在同一芯片上，因而得到了广泛的应用。在计算机主存技术中，半导体存储器已全部替代过去的磁性存储器。

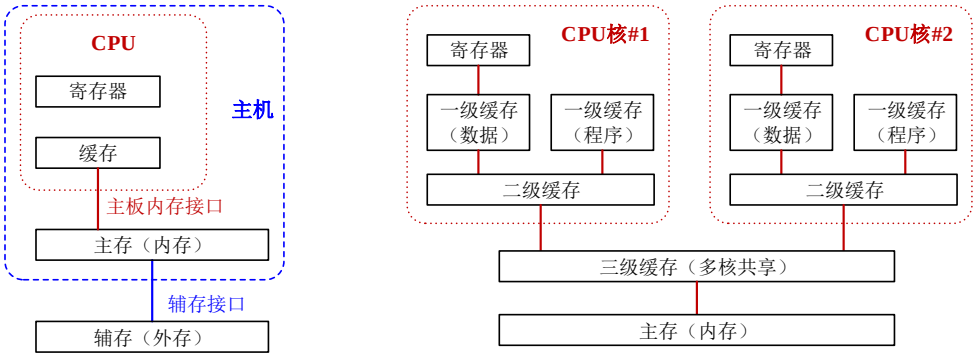
3. 光存储器

光存储器（optical storage）采用光学方法从光存储介质上读写数据。光存储的原理是投射激光到光盘上，并接收反射光，用反射光的强弱表征二进制的“0”和“1”，反射光强弱则通过在光盘上刻不同的凹坑来控制。读取光盘数据的设备称作光驱。光驱包括激光头组件（激光发生器、半反光棱镜、透镜及光电二极管）、驱动盘片旋转的电机、驱动激光头改变位置的伺服电机及控制电路、光电转换电路等。

光存储技术自二十世纪 80 年代初开始商用，1982 年出现了存储数字音乐的 CD-DA（Compact Disc Digital Audio），随后演变出存储数据的 CD-ROM（Compact Disc Read Only Memory），存储视频的 VCD（Video Compact Disc）和 DVD（Digital Video Disc）等。其存储密度比传统磁存储器大，成本低廉，在二十世纪 90 年代成为计算机辅助存储器的主流配置，也在音像领域获得了广泛的应用。但本世纪初以来半导体存储技术飞速发展，U 盘等小体积的外部存储设备开始逐步取代光盘。

3.1.2 层次化的存储器系统

如 2.2.5 小节所述，现代计算机往往采用分级存储体系结构。在这样的分层次存储系统中，离 CPU 越近的存储器速度越快（每字节的成本也越高、容量也越小）。按照与 CPU 由近到远的距离，不同的存储器常包括：寄存器、高速缓存、内存、本地磁盘、网络存储。图 3.2 (a) 所示为一种典型的四级存储器结构，图示结构中，CPU 芯片内部集成了少量寄存器和一定容量的高速缓存，在主机内（CPU 芯片外）安装内存，辅助存储器（外存）则通过输入/输出接口电路连接至主机。



(a) 典型四级存储结构　　(b) 典型多级缓存结构

图 3.2 多级存储器结构示例

1. 寄存器

寄存器是 CPU 内部的若干高速存储单元，采用触发器作为信息存储的基本电路单元。CPU 与寄存器之间的数据交换是通过内部总线直接进行的，所以 CPU 与寄存器之间的数据传送速度最快。其存储存储容量有限（kB 级别），用来暂存指令、数据和地址。CPU 访问内存数据时，往往先把数据取到寄存器中，以确保 ALU 执行运算的速度。

2. 高速缓存

高速缓存（Cache）介于 CPU 与内存之间，采用 SRAM 作为信息存储的基本电路单元。与寄存器相比，缓存的容量比 CPU 内部的寄存器大，但读写速度没有寄存器快；与内存相比，缓存的速度比内存快，但容量没有内存大。一把来说高速缓存置于 CPU 芯片内部，有些计算机中不仅 CPU 内部有高速缓存，在主板上也有缓存芯片，从而构成多级缓存。还有些 CPU 芯片内部也是多级的缓存结构，如图 3.2 (b) 所示。

高速缓存是加快 CPU 访问内存数据的桥梁。在预制定策略的控制下，缓存内保存着一部分内存中的数据。CPU 需要读取内存数据时，首先查询缓存中是否已有对应数据，若有直接读取，若没有则再从内存中读取。现代计算机系统中，缓存是分层存储系统非常重要的一个部分。

3. 主存

主存储器简称为主存，因位于计算机主机内部，常称作内存。内存普遍采用 DRAM 作为信息存储的基本电路单元。内存是计算机运行过程中的存储主力，用来存储指令、各种常量或变量等信息。需要运行的程序及相关数据先调入到内存中，然后再送入 CPU。内存的存储容量应足够大，以确保一个拟运行的程序可以完整载入（如果内存容量不足以载入完整程序，就需要虚拟内存技术，在 3.6 节讨论）。现代的微型计算机，其内存的大小往往在 4GB 至 16GB 之间。

4. 外存

外部辅助存储器被称为外存，分为联机外存和脱机外存。外存在掉电后数据不丢失，容量大，因而适合用于存储暂时不用的海量数据，如暂时不需要运行的程序、各类数据和电子文档等。

联机外存一般安装在计算机的主机箱内，主要采用机械硬盘（磁存储）或者基于半导体存储的固态硬盘 SSD（Solid State Disk）。很长一段时间以来，磁介质机械硬盘的存储空间

大，价格便宜，是使用最广泛的联机外存。近些年来半导体存储技术发展较快，固态硬盘，在一些领域已经取代了机械硬盘。脱机外存储器，常见如移动硬盘、光盘、U 盘、Flash 等，也有一些场合采用磁带作为脱机外存。

3.1.3 外存的接口标准

不同类型的辅助存储器与计算机连接时需要相应的接口电路。在计算机技术发展的过程中，逐步形成了一些全球通用的外存接口技术规范，如硬盘接口标准、Flash 存储卡接口标准等。通用的接口标准为不同厂家、不同类型辅助存储器产品接入计算机提供了有效保障，也便于产品升级。

1. IDE 接口

IDE（Integrated Drive Electronics）是 1984 年由 Western Digital 和 Compaq 联合开发的一种硬盘接口，也称作 ATA（Advanced Technology Attachment）接口。在二十世纪 90 年代初开始应用于微型计算机，使用一根 40 芯的扁平电缆与主板进行连接，一根电缆能连接两个硬盘。发展出多个不同的版本，后期的几个版本均称作 Ultra DMA，其中最快的是 Ultra DMA133，速度达到了 133MB/s。40 芯连接线中数据线共 16 芯，是典型的并行数据传输接口，曾经是微型计算机最常见的硬盘接口。由于后来微型计算机上硬盘接口逐步演变为 SATA（Serial ATA），为明确区别，IDE 常被称作 PATA（Parallel ATA）。

2. SCSI 接口

SCSI（Small Computer System Interface）接口是一种通用接口，可以连接磁盘、磁带、光驱、打印机、扫描仪等不同外设。在 SCSI 控制器的管理下，可以连接多达 15 个不同的外设。往往服务器中配置双通道的 SCSI 控制器，可连接 30 个外设。这种可连接多个外设的特性使 SCSI 特别适用于工作站、服务器的环境，反之在微型计算机上使用不多。

二十世纪 90 年代，SCSI 不断发展，演变出多个版本，早期版本使用 50 芯连接电缆（其中数据线 8 芯），后期版本使用 68 芯或 80 芯连接电缆（其中数据线 16 芯），Ultra 640 SCSI 最高速率可达 640MB/s。

3. SATA 和 SAS 接口

SATA（Serial ATA）由 Intel、IBM、Dell、APT、Maxtor 和 Seagate 公司共同提出。SATA 采用串行方式传送数据，减少了 SATA 接口的针脚数目，从而连接电缆芯线数目减少到 4 根。SATA 使用差动信号系统（Differential-signal-amplified-system）来提高高速率传输的可靠性，同时可降低工作电压。与 PATA 高达 5V 的传输电压相比，SATA 的峰峰值差模电压仅 500mV。SATA 1.0 规范支持 150MB/s 的理论传输速率，线缆长度小于 1 米；SATA 2.0 支持 300MB/s，线缆长度小于 1.5 米；SATA 3.0 的传输速度达 600MB/s，线缆长度小于 2 米。由于 SATA 接口结构简单、支持热插拔、执行效率高，目前已取代传统 ATA，成为微型计算机的标准接口之一。

SAS（Serial Attached SCSI）接口即串行 SCSI，由并行 SCSI 物理存储接口演化而来，可以兼容 SATA，但 SATA 并不兼容 SAS。SAS 采用与 SATA 类似的串行传输技术来获得更高的传输速度，SAS 1.0、2.0、3.0 版本支持的传输速率分别达到 300MB/s、600MB/s 和 1200MB/s。

4. SD 接口

SD（Secure Digital Memory Card）卡是一种广泛用于移动设备的标准存储卡。SD 卡由松下电器、东芝和 SanDisk 共同研制，1999 年 8 月发布。其前身是用于移动电话和数字影像设备的 MMC（MultiMedia Card）卡，MMC 规范 1997 年由西门子和 SanDisk 发布。SD 卡尺寸与 MMC 卡相似，为 32mm×24mm×2.1mm（比 MMC 卡厚 0.7mm）。

SD 卡内部集成了闪存芯片颗粒和闪存读写控制器，通过 9 针的接口与专门的驱动器连接。由于 SD 卡是一体化固态介质，故便于携带且不易损坏。SD 卡 3.0 规范支持理论最大容量为 2TB，理论最高读写速率 104MB/s，可以满足一般电子产品访问速率的需求，被广泛用于数码照相机、数字摄像机等消费电子产品。

SD 卡还衍生出了 Mini SD 和 Micro SD（T-Flash，简称 TF 卡）两种不同尺寸的子类型。Micro SD 卡具有更小的尺寸，已经在很多场合取代了传统 SD 卡，如智能手机中广泛集成的就是 Micro SD 卡接口。Micro SD 卡可以通过 SD 卡适配器装入传统 SD 卡槽使用，目前已成为脱机外存储器的主流。

5. eMMC 接口

eMMC（embedded Multi Media Card）是 JEDEC 协会制定的嵌入式存储器标准接口。eMMC 规范是一个嵌入式存储解决方案，包括闪存颗粒和闪存读写控制器，其作用类似微型计算机上使用的 SSD 固态硬盘。其设计初衷是简化闪存与 CPU 连接的接口及其控制。2019 年 1 月发布的 eMMC 5.1 版本可提供 400MB/s 的数据传输速率，接口电压可以是 1.8V 或者是 3.3V。

通常 eMMC 生产厂家把 NAND Flash 芯片和读写控制芯片设计成 MCP（Multiple Chip Package）芯片，用户购买 eMMC 芯片后可直接加入到电路板中，无需处理 NAND Flash 兼容性和管理问题，从而简化了存储器的设计。

JEDEC 制定的 UFS（Universal Flash Storage）接机口，同样是整合了主控芯片的闪存。与 eMMC 不同，UFS 接口采用串行传输技术，并且使用了 SCSI 模型并支持对应的 SCSI 指令集。2020 年 1 月发布的 UFS 3.1 接口数据传输速率可达 2.9GB/s。目前 UFS 已经开始逐渐取代 eMMC，很可能成为未来主流手机存储接口。

3.1.4 存储器性能指标

存储器的性能指标包括存储容量、存取时间、存储周期和存储器带宽，以及可靠性、功耗、价格、电源种类等。其中最基本的指标是存储容量和存取速度。

1. 存储容量

一颗存储芯片可容纳的存储单元总数通常称为该存储器的存储容量。存储容量用位数、字数或字节数来表示。以位数表示时，存储器芯片容量＝单元数×数据线位数。若该存储器芯片地址线位数为 n，数据线位数为 m，则可编址的单元总数为：2^n，存储器芯片容量为 $2^n \times m$ 位。

为方便描述存储器的容量，常采用不同数量级的计量单位。参考 ISO/IEC 80000-13 关于信息科学领域的计量单位标准，不同数量级的存储单位如表 3.1 所示。由表 3.1 可知，B 表示字节，一个字节定义为 8 个二进制位。例如，1KB 表示 1000 字节，1KiB 表示 1024 字节。

但由于历史形成的习惯，多数计算机类的资料中并未区分 KB 和 KiB，很多使用 KB 的场合事实上意为 KiB。鉴于此，本书中也未对 KiB 和 KB 进行严格区分，按多数资料的习惯，使用 KB 替代了 KiB。

计算机的字长通常为 8 的倍数，故习惯以字节（B）做存储器容量的计量单位。较小的存储容量可表示为 64KB、512KB、1MB 等；较大的存储容量则采用 GB、TB 等表示。

表 3.1 计算机存储单位换算表

中文单位	中文简称	英文单位	英文简称	十进制字节数	二进制英文单位	二进制英文简称	二进制字节数
位	比特	bit	b	0.125	bit	b	0.125
字节	字节	Byte	B	1	Byte	B	1
开字节	开	KiloByte	KB	10^3	KibiByte	KiB	2^{10}
兆字节	兆	MegaByte	MB	10^6	MebiByte	MiB	2^{20}
吉字节	吉	GigaByte	GB	10^9	GibiByte	GiB	2^{30}
太字节	太	TeraByte	TB	10^{12}	TebiByte	TiB	2^{40}
拍字节	拍	PetaByte	PB	10^{15}	PebiByte	PiB	2^{50}
艾字节	艾	ExaByte	EB	10^{18}	ExbiByte	EiB	2^{60}
泽字节	泽	ZettaByte	ZB	10^{21}	ZebiByte	ZiB	2^{70}
尧字节	尧	YottaByte	YB	10^{24}	YobiByte	YiB	2^{80}

虽然位是存储电路可以实现的最小单位，但在计算机中，习惯使用字节或字作为存储单元的大小。存放一个机器字的存储单元，通常称为字存储单元，相应的单元地址叫字地址。而存放一个字节的单元，称为字节存储单元，相应的地址称为字节地址。

如果计算机中可编址的最小单位是字存储单元，则该计算机称为按字编址的计算机。如果计算机中可编址的最小单位是字节，则该计算机称为按字节编址的计算机。不同字长的计算机中，一个字包含的字节数目不同。如 32 位计算机中一个字对应 4 个字节，64 位计算机中一个字对应 8 个字节。

2. 存取时间和存取周期

存取时间，也称存储器访问时间，常记为 T_A。指从启动一次存储器操作到完成该操作经历的时间。以 CPU 读存储器为例，从读操作命令发出，直至数据读入寄存器，整个过程所经历的时间即存取时间。通常超高速存储器的存取时间小于 20ns；中速存储器的存取时间在 100～200ns 之间；低速存储器的存取时间在 300ns 以上。

存取周期，常记为 T_M。指连续启动两次独立的存储器操作（“读”或者“写”）所需间隔的最小时间。通常，存取周期 T_M 略大于存取时间 T_A。

3. 数据传输速率

数据传输速率，也称为带宽，常记为 B_M。指单位时间内能够传送的信息量。若系统的总线宽度（即数据总线的位数）为 W，则 $B_M=W/T_M$（b/s）。例如，若 W=32b，T_M=100ns，则 B_M=32b /100× 10^{-9}s=320M b/s，换算为单位时间内能够传输的字节数，即为 40M B/s。

3.2 只读存储器

如前所述，ROM 与 RAM 相比最大的特点是非易失性，一般用于存储固定的数据或者程序。ROM 的工作方式与 RAM 不同，在正常工作状态下，只能从 ROM 中读出数据。并非所有类型的 ROM 都只能读，在特定条件下，一些 ROM 是可以执行写入操作的。依据是否可编程（Programmable，意为向 ROM 芯片写入数据）、编程的次数及编程方式可以分为掩模 ROM、可编程 ROM、可擦除可编程 ROM、电可擦除可编程 ROM 等不同类型。如今广泛用于 U 盘、SSD 和 eMMC 等移动存储的闪存 Flash，属于广义的电可擦除可编程 ROM。

3.2.1 掩模 ROM

掩模 ROM（Mask ROM），生产时采用掩模工艺制作 ROM。制作过程需要根据用户对存储数据的要求，设计专门的掩模板，依据该掩模板制作的 ROM 在出厂时内部数据就已固化，用户不能修改。

掩模 ROM 电路包括存储矩阵、地址译码器和输出缓冲器三个组成部分。其存储核心是存储矩阵，如图 3.3 中虚线框内所示，存储矩阵由多个位存储单元按照行、列排列而成。位存储单元可以由二极管构成，也可以采用双极型三极管或 MOS 管。每个位存储单元可以存放一位信息（“0”或“1”），通常会将一组位存储单元（如图 3.3 中存储矩阵的一列）编址为一个地址代码。

地址译码器将输入的地址代码译成 ROM 内部的控制信号，该控制信号用于选中存储矩阵中的指定单元，并将存放其中的数据送到输出缓冲器。输出缓冲器的作用有两个，一是提高负载能力，二是实现对输出状态的三态控制，控制输出信号线与系统总线的连接与脱离。

图 3.3 所示 ROM 为一个 4×4 位的 MOS 管 ROM，每个存储单元 4 位（图中的一列），每个地址码可以选中 1 个 4 位的存储单元。地址译码器的输入是两位地址线 A_1、A_0，译码后可译出 4 种状态 W_3~W_0，输出 4 条选择线，分别选中 4 个单元，每个单元有 4 位输出。W_3~W_0 中任意一根线上给出高电平时，都会在输出信号线 d_3~d_0 上输出一个 4 位二值代码。通常将每个输出二值代码称为一个“字”，并将 W_3~W_0 称为字线，将 d_3~d_0 称为位线（或数据线），而 A_1、A_0 称为地址线。输出端的缓冲器用来提高负载能力，并将输出的高、低电平变换为标准的逻辑电平。同时，通过给定 EN#信号实现对输出的三态控制。

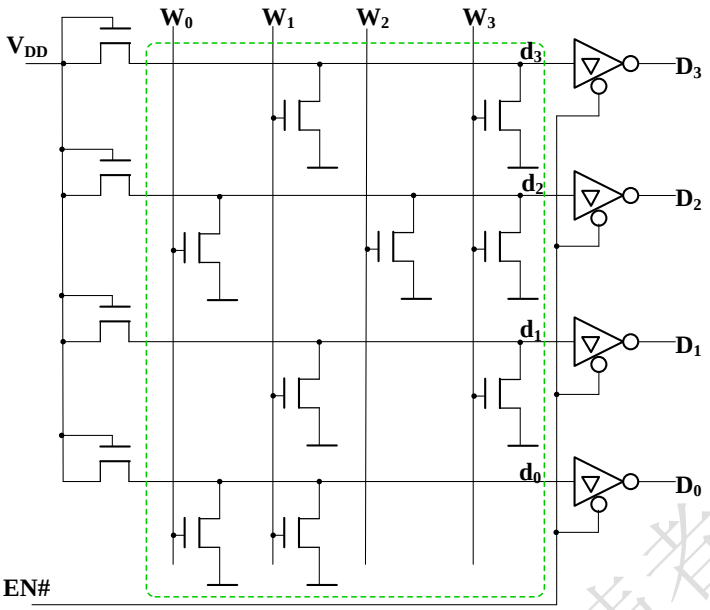


图 3.3 掩模 ROM 电路原理图

图 3.3 所示存储矩阵中，行和列的交叉点，有的连接了 MOS 管，有的没有连接。根据用户提供的掩模图形，进行二次光刻，进而确定行和列的交叉点是否连接 MOS 管，由于制作流程需要使用掩模（mask），故称为掩模 ROM。若地址线 A_1A_0=00，则选中 0 号单元，即字线 0 为高电平，若有 MOS 管与字线 0 相连（如位线 d_2 和 d_0），相应的行列交叉处的 MOS 管导通，对应位线输出低电平（对应“0”）；而位线 d_1 和 d_3 没有 MOS 管与字线相连，输出为高电平（对应“1”）。图 3.3 存储矩阵的内容如表 3.2 所示。

表 3.2 掩模 ROM 存储矩阵的内容

位 / 单元	d_3	d_2	d_1	d_0
0	1	0	1	0
1	1	0	1	1
2	1	1	1	0
3	0	0	0	1

3.2.2 PROM

可编程 ROM（Programmable ROM）简称 PROM。PROM 的电路结构与掩模 ROM 类似，由存储矩阵、地址译码器和输入/输出控制电路组成。但 PROM 生产时在存储矩阵的所有行列交叉点上全部放置了存储元件，相当于在所有存储单元中都存入了“1”。用户可以根据需要将其中的某些单元写入数据“0”以实现对其“编程（即修改信息）”的目的。有些 PROM 在出厂时数据全为“0”，用户编程就是将其中的部分单元写入“1”。

图 3.4 所示为一种典型的熔丝型 PROM 的存储单元原理图。存储单元由一只三极管和串接于发射极的熔丝组成。三极管的 be 结相当于连接在字线与位线间的二极管。熔丝采用很细的低熔点合金丝或多晶硅导线制作。如果在存储单元上通以足够大的电流，并持续一定

时间，熔丝就会烧断，从而达到改写该存储单元信息的目的。

熔丝在集成电路中会占据较大的芯片面积，故又出现了反熔丝结构的 PROM。原有熔丝的熔丝被替换为由两个反相串联肖特基二极管构成的绝缘体，出厂时处于绝缘状态，需要编程时，施以大电流会使绝缘体被永久性击穿，从而连接点两端导通。

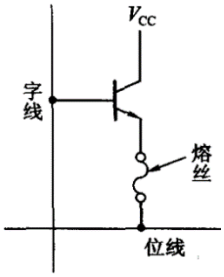


图 3.4 熔丝型 PROM 存储单元

图 3.5 是一个 16×8 位 PROM 的结构原理图。地址线经地址译码器后译为字选择线 W_0、W_1、……、W_{15}，用于选中要编程的存储单元（8 位，对应图中一行）。编程时，将 V_{CC} 和选中的字线提高到编程所需的高电平，同时在位线上加载编程脉冲（幅度约 20V，持续时间约十几微秒）。此时写入放大器 A_W 的输出为低电平、低内阻状态，有较大的脉冲电流流过熔丝，将其熔断。而正常工作状态（读取数据）下，读出放大器 A_R 输出的高电平不足以使 D_Z 导通，A_W 不工作。

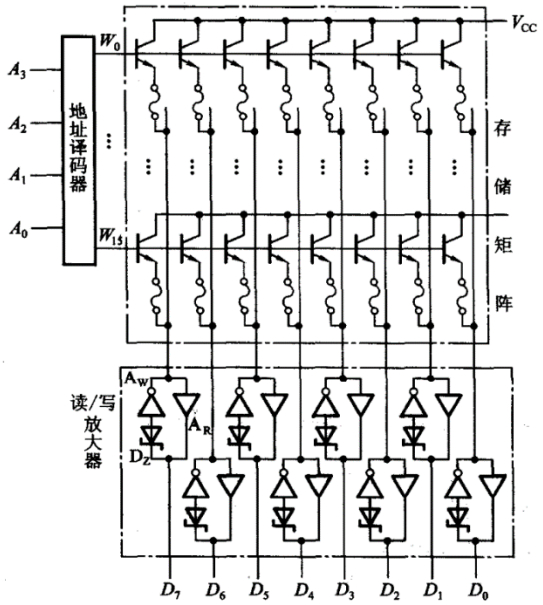


图 3.5 PROM 电路结构示意图

3.2.3 EPROM

可擦除可编程 ROM（Erasable Programmable ROM）简称 EPROM。利用编程器将信息写入 EPROM 后，数据可长久保持。需要重新写入新内容时，则利用擦除器（紫外线灯照射）将原有存储信息擦除，各单元内容复原为初始值；然后再利用 EPROM 编程器编程，因此 EPROM 可以反复使用。

1. EPROM 的存储单元电路

通常 EPROM 中采用叠栅注入 MOS 管（Stacked-gate Injection Metal-Oxide Semiconductor，简称 SIMOS 管）形成存储单元，其构造如图 3.6 所示。它是一个 N 沟道增强型的 MOS 管，有两个重叠的栅极—控制栅 G_c 和浮栅 G_f。控制栅 G_c 用于控制读出和写入，浮栅 G_f 用于长期保存注入电荷。

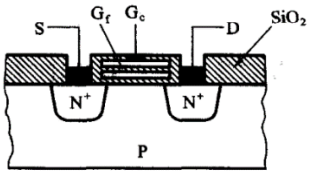


图 3.6 SIMOS 管结构

浮栅上未注入电荷以前，在控制栅上加入正常的高电平能够使漏-源之间产生导电沟道，MOS 管导通。反之，在浮栅上注入了负电荷以后，必须在控制栅上加更高的电压才能抵消注入电荷的影响而形成导电沟道，因此在栅极加上正常的高电平信号时 SIMOS 管将不会导通。

当漏-源间加以较高的电压（约+20～+25V）时，将发生雪崩击穿现象。如果同时在控制栅上加以高压脉冲（幅度约+25V，宽度约 50ms），则在栅极电场的作用下，一些高能量的电子便穿越 SiO_2 层到达浮栅，被浮栅捕获而形成注入电荷。浮栅上注入了电荷的 SIMOS 管相当于写入了“1”，未注入电荷的相当于存入了“0”。漏极和源极间的高电压去掉以后，由于浮栅被 SiO_2 维缘层包围，注入到浮栅上的电荷没有放电通路，所以在室温和无光照的条件下能长期地保存在浮栅中。

消除浮栅电荷的办法是利用紫外线光照射，由于紫外线光子能量较高，从而可使浮栅中的电子获得能量，形成光电流从浮栅流入基片，使浮栅恢复初态。EPROM 芯片上方有一个石英玻璃盖板，在紫外线照射足够时间后，读出各单元的内容均为 0x00，则说明该 EPROM 已擦除。在写好数据以后应使用不透明胶带将石英盖板遮蔽，以防止数据丢失。

将一个浮栅管和 MOS 管串起来组成如图 3.7 所示的存储单元电路。浮栅中注入了电子的 MOS 管源—漏极不导通，当地址译码器字线选中该存储单元时，相应的位线为高电平，即读取值为“0”。而未注入电子的浮栅管的源—漏极导通，故读取值为“1”。在原始状态(即厂家出厂)，没有经过编程，浮栅中无注入电子，位线上总是“1”。

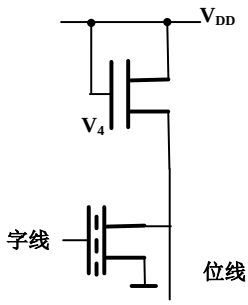


图 3.7 使用 SIMOS 管的存储单元电路

2. EPROM 芯片示例

EPROM 芯片有多种型号，如 2716(2 K×8 位)、2732(4 K×8 位)、2764(8 K×8 位)、27128(16 K×8 位)、27256(32 K×8 位）等。下面以 2764A 为例，介绍 EPROM 的性能和工作方式。

Intel 2764A 有 13 条地址线，8 条数据线，2 个电压输入端 V_{CC} 和 V_{PP}，一个片选端 CE#(功能同 CS#)，此外还有输出允许 OE#和编程控制端 PGM#，其功能框图见图 3.8。

Intel 2764A 有七种工作方式，如表 3.3 所示。

（1）读方式

读方式是 2764A 通常使用的方式，此时两个电源引脚 V_{CC} 和 V_{PP} 都接至+5 V，PGM#接至高电平，当从 2764A 的某个单元读数据时，先通过地址引脚接收来自 CPU 的地址信号，然后使控制信号和 CE#、OE#都有效，于是经过一个时间间隔，指定单元的内容即可读到数据总线上。

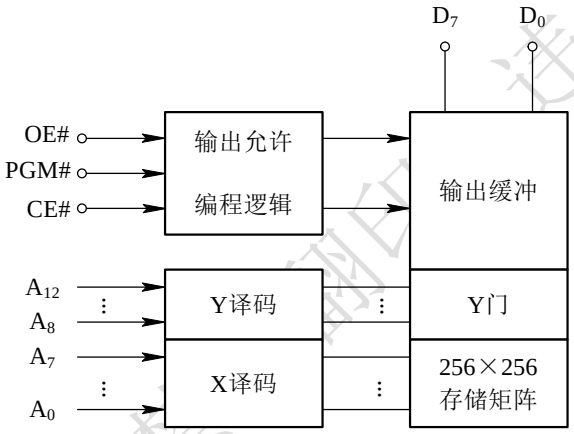


图 3.8 2764A 功能框图

但把 A_9 引脚接至 11.5～12.5 V 的高电平，则 2764A 处于读 Intel 标识符模式。要读出 2764A 的编码必须顺序读出两个字节，先让 A_1～A_8 全为低电平，而使 A_0 从低变高，分两次读取 2764A 的内容。当 A_0=0 时，读出的内容为制造商编码(陶瓷封装为 89H，塑封为 88H)，当 A_0=1 时，则可读出器件的编码(2764A 为 08H，27C64 为 07H)。

（2） 备用方式

CE#为高电平时，2764A 工作于备用方式。此时芯片处于低功耗状态，所需电流由 100 mA 下降到 40 mA。

（3） 编程方式

这时，V_{PP} 接+12.5V，V_{CC} 仍接+5V，从数据线输入这个单元要存储的数据，CE#端保持低电平，输出允许信号 OE#为高，每写一个地址单元，都必须在 PGM#引脚端给一个低电平有效，宽度为 45 ms 的脉冲，如图 3.9 所示。

（4） 编程禁止

在编程过程中，只要使该片 CE#为高电平，编程就立即禁止。

（5） 编程校验

在编程过程中，为了检查编程时写入的数据是否正确，通常在编程过程中包含校验操作。

在一个字节的编程完成后，电源的接法不变，但 PGM#为高电平，CE#、OE#均为低电平，则同一单元的数据就在数据线上输出，这样就可与输入数据相比较，校验编程的结果是否正确。

表 3.3 2764A 的工作方式选择表

引脚 / 方式	CE#	OE#	PGM#	A_9	A_0	V_{CC}	V_{PP}	数据端功能
读	低	低	高	×	×	5V	V_{CC}	数据输出
输出禁止	低	高	高	×	×	5V	V_{CC}	高阻
备用	高	×	×	×	×	5V	V_{CC}	高阻
编程	低	高	低	×	×	V_{CC}	12.5V	数据输入
校验	低	低	高	×	×	V_{CC}	12.5V	数据输出
编程禁止	高	×	×	×	×	V_{CC}	12.5V	高阻
标识符	低	低	高	高	低 高	5V 5V	V_{CC} V_{CC}	制造商器件编码

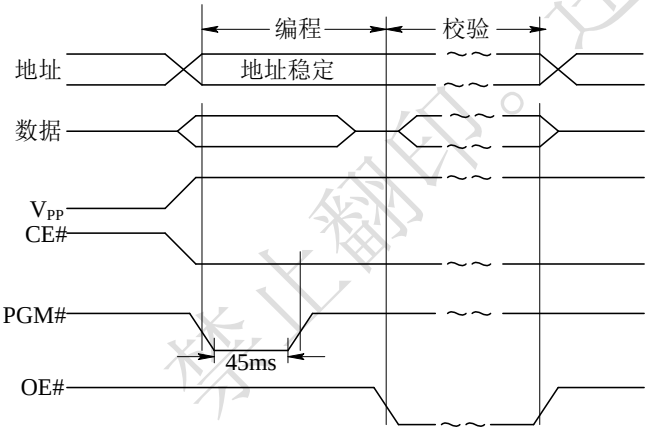


图 3.9 2764A 编程波形

（6）快速编程方式

当两个电源端 V_{CC} 和 V_{PP} 都接至+5 V，CE#=OE#=0 时，PGM#为高电平，这时与读方式相同。另外，在对 EPROM 编程时，每写一个字节都需 45 ms 的 PGM#脉冲，速度太慢，且容量越大，速度越慢。为此，Intel 公司开发了一种新的编程方法，比标准方法快 6 倍以上，其流程图如图 3.10 所示。

实际上，按这一思路开发的编程器有多种型号。编程器中有一个卡插在 I/O 扩展槽上，外部接有 EPROM 插座，所提供的编程软件可自动提供编程电压 V_{PP}，按菜单提示，可读、可编程、可校验，也可读出器件的编码，操作很方便。

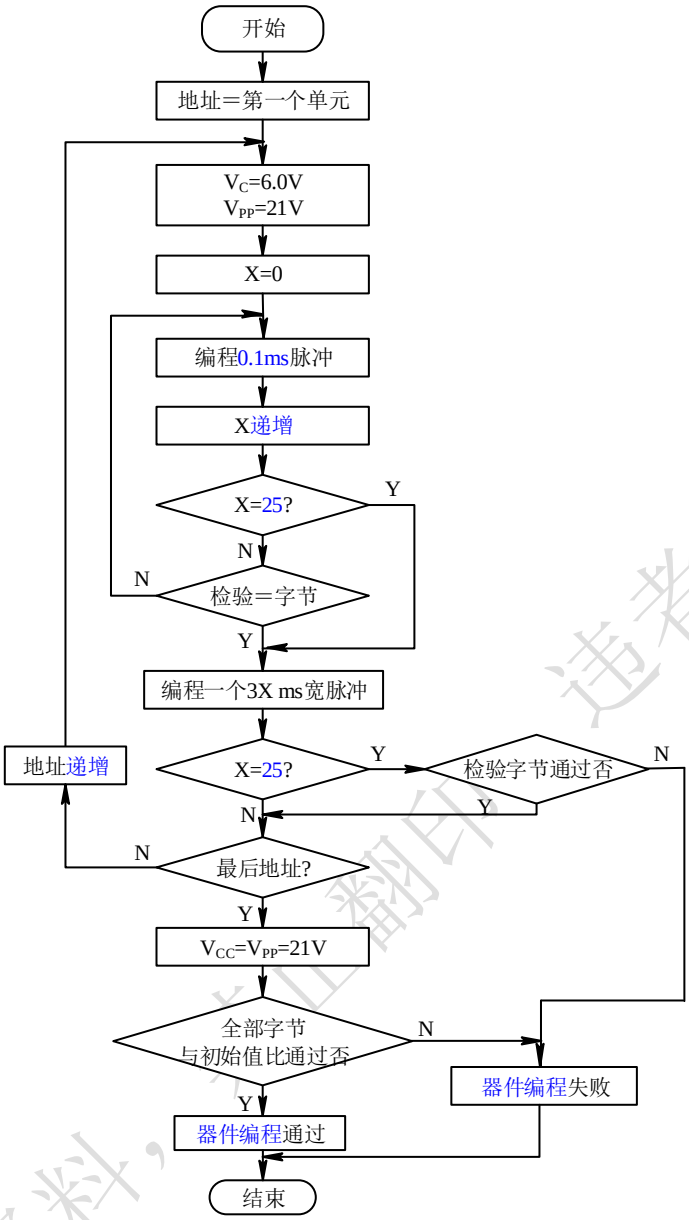


图 3.10 Intel 对 EPROM 编程算法流程图

3.2.4 EEPROM

EPROM 的优点是一块芯片可多次重新编程使用，缺点是整个芯片即使只写错一位，也必须从电路板上取下擦掉重写，擦除时间很长，因而使用不便。在实际应用中，往往希望能够以字节为单位进行擦写。电可擦除可编程 ROM（Electrically Erasable Programmable ROM，简称 EEPROM 或 E^{2}PROM）克服上述缺点，其擦出和写入均采用电的方式。

1. E^{2}PROM 工作原理

如图 3.11 所示，存储单元中采用了一种称为浮栅隧道氧化层 MOS 管（Floating gate Tunnel Oxide，简称 Flotox 管）。Flotox 管与 SIMOS 管相似，属于 N 沟道增强型的 MOS 管，有两个栅极—控制栅 G_C 和浮栅 G_f。不同的是 Flotox 管的浮栅与漏区之间有一个氧化层极薄（厚度在 2×10^{-8}m 以下）的区域。这个区域称为隧道区。当隧道区的电场强度大到一定程度

时（>10^7V/cm），便在漏区和浮栅之间出现导电隧道，电子可以双向通过，形成电流。这种现象称为隧道效应。

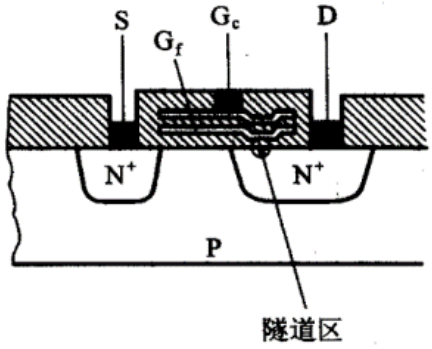


图 3.11 Flotox 管结构

加到控制栅 G_C 和漏极 D 上的电压是通过浮栅-漏极间的电容和浮栅-控制栅间的电容分压加到隧道区上的。为了使加到隧道区上的电压尽量大，需要尽可能减小浮栅和漏区间的电容，因而要求把隧道区的面积做得非常小。可见，在制作 Flotox 管时对隧道区氧化层的厚度、面积和耐压的要求都很严格。

为了提高擦、写的可靠性，并保护隧道区超薄氧化层，在 E^2PROM 的存储单元中除 Flotox 管以外还附加了一个选通管，如图 3.12 所示。图中的 T_1 为 Flotox 管（也称为存储管），T_2 为普通的 N 沟道增强型 MOS 管（也称为选通管）。根据浮栅上是否充有负电荷来区分单元的“1”或“0”状态。

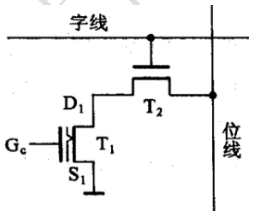


图 3.12 E^2PROM 存储单元

（1）读出：在字线施加高电平使选通管 T2 导通。在控制栅 G_C 施加+3V 电压，如果 T_1 管的浮栅上没充有负电荷，则 T_1 管导通，在位线上读出“0”（低电平）；如果 T_1 管的浮栅上充有负电荷，则 T_1 截止，在位线上读出“1”。

（2）写入：使写入为“0”的存储单元的 T_1 管浮栅放电。控制栅 G_C 接 0 电平，同时在字线和位线上施加+20V 左右、宽度约 10ms 的脉冲电压。此时浮栅上的存储电荷将通过隧道区放电，使 T_1 管的开启电压降为 0V 左右，成为低开启电压管。从而，读出时在控制栅 G_C 施加+3V 电压时，T_1 管为导通状态。虽然 E^2PROM 改用电压信号擦除了，但由于擦除和写入时需要加高电压脉冲，而且擦、写的时间仍较长，所以在系统的正常工作状态下，E^2PROM 仍然只能工作在它的读出状态，作 ROM 使用。

（3）擦除：控制栅和字线上都施加+20V 左右、宽度约为 10ms 的脉冲电压，漏区接 0 电平。这时经 G_C—G_f 间电容和 G_f—漏区电容分别在隧道区产生强电场，吸引漏区的电子通过隧道区到达浮栅，形成存储电荷，使 Flotox 管的开启电压提高到+7V 以上，成为高开启电压管。读出时施加在 G_C 上的电压只有+3V，T_1 管不会导通。一个字节擦除后，所有的存储

单元均为“1”状态。

2. 2816 E²PROM 的基本特点

2816 是容量为 2 K×8 位的 E²PROM，它的逻辑符号如图 3.13 所示。芯片为 24 脚 DIP 封装，其引脚排列与 EPROM 芯片 2716 一致，只是在引脚定义上，数据线引脚对 2816 来说是双向的，以适应读写工作模式。

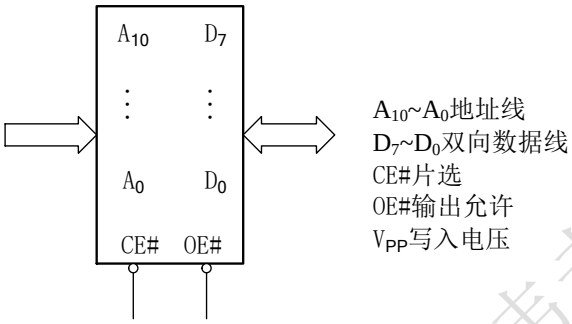


图 3.13 2816 的逻辑符号

2816 的读取时间为 250ns，可满足多数微处理器对读取速度的要求。2816 最突出的特点是可以字节为单位进行擦除和重写。擦或写用 CE#和 OE#信号加以控制，一个字节的擦写时间为 10ms。2816 也可整片进行擦除，整片擦除时间也是 10ms。无论字节擦除还是整片擦除均在机内进行。

3. 2816 的工作方式

2816 有六种工作方式，每种工作方式下各个控制信号所需电平如表 3.4 所示。从表中可见，除整片擦除外，CE#和 OE#均为 TTL 电平，而整片擦除时电压为+9～+15V，在擦或写方式时 V_{PP} 均为+21V 的脉冲，而其他工作方式时电压为+4～+6V。

表 3.4　2816 的工作方式

引脚 / 方式	CE#	OE#	V_{PP}/V	数据端功能
读	低	低	+4~+6V	输出
备用	高	×	+4~+6V	高阻
字节擦除	低	高	+21V	输入为高电平
字节写	低	高	+21V	输入
片擦除	低	+9~+15V	+21V	输入为高电平
擦写禁止	高	×	+21V	高阻

（1）读方式

在读方式时，允许 CPU 读取 2816 的数据。当 CPU 发出地址信号以及相关的控制信号后，与此相对应，2816 的地址信号和 CE#、OE#信号有效，经一定延时，2816 可提供有效数据。

（2）写方式

2816 具有以字节为单位的擦写功能，擦除和写入是同一种操作，即都为写，只不过擦除是固定写“1”而已。因此，在擦除时，数据输入是 TTL 高电平。在以字节为单位进行擦除和写入时，CE#为低电平，OE#为高电平，从 V_{PP} 端输入编程脉冲，宽度最小为 9ms，最大为 70ms，电压为+21V。为保证存储单元能长期可靠地工作，编程脉冲要求以指数形式上升到+21V。

（3）片擦除方式

当 2816 需整片擦除时，也可按字节擦除方式将整片 2KB 逐个进行，但最简便的方法是依照表 3.4，将 CE#和 V_{PP} 按片擦除方式连接，将数据输入引脚置为 TTL 高电平，而使 OE#引脚电压达到+9～+15V，则约经 10ms，整片内容全部被擦除，即 2KB 的内容全为 FFH。

（4）备用方式

当 2816 的 CE#端加上 TTL 高电平时，芯片处于备用状态，OE#控制无效，输出呈高阻态。在备用状态下，其功耗可降到 55%。

3.2.5 Flash

Flash（闪存）是一种类似于 EPROM 的单管叠栅结构的存储单元。Flash 既吸收了 EPROM 结构简单、编程可靠的优点，又保留了 E^2PROM 用隧道效应擦除的快捷特性，而且集成度可以做得很高。

1. 闪存 Flash 工作原理

图 3.14 所示为 Flash 采用的叠栅 MOS 管的结构示意图。其结构与 EPROM 中的 SIMOS 管极为相似，若浮空栅上保存有电荷，则在源、漏极之间形成导电沟道，达到一种稳定状态，可以定义该基本存储单元电路保存信息“0”；若浮空栅上没有电荷，则在源、漏之间无法形成导电沟道，为另一稳定状态，可定义保存信息“1”。

Flash 与 EPROM 最大的区别是浮栅与衬底间氧化层的厚度不同。在 EPROM 中这个氧化层的厚度一般为 30~40nm，而在 Flash 中仅为 10~15nm。而且浮栅与源区重叠的部分是由源区的横向扩散形成的，面积极小，因而浮栅——源区间的电容要比浮栅——控制栅间的电容小得多。当控制栅和源极间加上电压时，大部分电压都将降在浮栅与源极之间的电容上。

Flash 读出时，源极 V_{SS} 接地，字线为 5V 逻辑高电平。Flash 写入时，源极 V_{SS} 接地、漏极接 6V、控制栅 12V 脉冲、宽 10s，基于热电子效应或隧道效应向浮栅注入电荷。而 Flash 擦除时，控制栅接地、源极接+12V 脉冲、宽为 100ms，利用隧道效应将浮栅电荷释放。

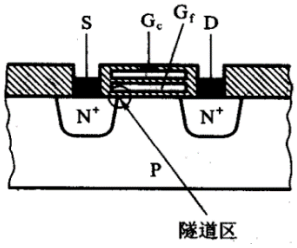


图 3.14 快闪存储器中的叠栅 MOS 管

E^2PROM 的存储单元用了两只 MOS 管，但 Flash 的存储单元由单管组成，如图 3.15 所示。对比图 3.12，Flash 没有选通管 T，因此集成度高，容量大，但稳定性和擦写次数都不如 E^2PROM。

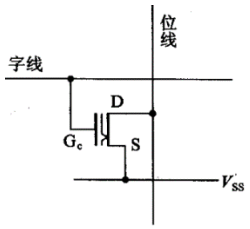


图 3.15 快闪存储器的存储单元

Flash 的编程和擦除操作不需要使用编程器，写入和擦除的控制电路集成于存储器芯片中，工作时只需要 5V 的低压电源，使用极其方便。由于叠栅 MOS 管浮栅下面的氧化层极薄，经过多次编程以后可能发生损坏，所以 Flash 的编程次数是有限的，一般在 10 000~100 000 次之间。随着制造工艺的改进，可编程的次数有望进一步增加。

自二十世纪 80 年代末期 Flash 问世以来，便以其高集成度、大容量、低成本和使用方便等优点而引起普遍关注。产品的集成度在逐年提高，大容量闪存的产品已经大规模应用。应用领域迅速扩展，已经在嵌入式和桌面应用中取代了机械硬盘。

2. NAND Flash 和 NOR Flash

Flash 分为 NAND flash 和 NOR flash 二种。NAND flash 的擦和写均是基于隧道效应，电流穿过浮栅极与硅基层之间的绝缘层，对浮栅极进行充电（写数据）或放电（擦除数据）。而 NOR flash 擦除数据仍是基于隧道效应（电流从浮栅极到硅基层），但在写入数据时则是采用热电子注入方式（电流从浮栅极到源极）。

1988 年 Intel 开发出 NOR Flash 芯片。NOR Flash 的特点是芯片内执行（XIP，eXecute In Place），具有随机存取和对字节执行写（编程）操作的能力，这样应用程序和数据可以直接在 Flash 闪存内运行，不必读取到系统 RAM 中。NOR Flash 允许单字节或单字编程，但不能单字节擦除，必须以块为单位或对整片执行擦除操作，在对存储器进行重新编程之前需要对块或整片进行预编程和擦除操作。由于 NOR 技术局限 Flash Memory 的擦除和编程速度较慢，而块尺寸又较大，因此擦除和编程操作所花费的时间很长。NOR Flash 的传输效率很高，在小容量时具有很高的成本效益，但是很低的写入和擦除速度影响到它的性能。

1989 年，东芝公司发表了 NAND Flash 结构，强调降低每比特的成本，有更高的性能，并且像磁盘一样可以通过接口升级。NAND Flash 的结构能提供极高的单元密度，可以达到高存储容量。以页为单位进行读和编程操作，以块为单位进行擦除操作。具有快速编程和擦除的优势；数据、地址采用同一总线，实现串行读取。随机读取速度慢且不能按字节随机编程。芯片尺寸小，引脚少，位成本低。芯片包含有失效块。

NAND Flash 支持速率超过 5Mbps 的持续写操作，其区块擦除时间短至 2ms，而 NOR Flash 是 750ms。然而，NAND Flash 不适合直接随机存取，适合于纯数据存储和文件存储，主要作为 U 盘、Smart Media 卡、Compact Flash 卡、固态盘等的存储介质。

3.3 随机存取存储器

RAM 类型的存储器的工作特点是可随时从中快速读取或写入数据，属于易失性存储器，当关机或断电时，其中的信息都会随之丢失。按照存储单元的特征，RAM 分成动态随机存取存储器 DRAM（Dynamic Random Access Memory）和静态随机存取存储器 SRAM（Static Random Access Memory）。

SRAM 的存储电路以双稳态触发器为基础，状态稳定，只要不掉电，信息不会丢失。优点是不需刷新，缺点是集成度低。适用于不需要大存储容量的微型计算机（例如，单板机和单片机）中。DRAM 的存储单元以电容为基础，电路简单，集成度高。但也存在问题，即电容中电荷由于漏电会逐渐丢失，因此 DRAM 需定时刷新。它适用于大存储容量的计算机。

3.3.1 静态 RAM

1. 静态 RAM 的基本存储电路

静态 RAM 存储单元是在 SR 锁存器的基础上附加门控管而构成的。因此，它是靠锁存器的自保功能存储数据的。

如图 3.16 所示是一种六 MOS 管 SRAM 存储单元电路结构。在此电路中，V_1～V_4 管组成双稳态锁存器，用于记忆 1 位二值代码。若 V_1 截止，则 A 点为高电平，它使 V2 导通，于是 B 点为低电平，这又保证了 V_1 的截止。同样，V_1 导通而 V_2 截止，这是另一个稳定状态。因此，可用 V_1 管的两种状态表示“1”或“0”。由此可知，静态 RAM 保存信息的特点是和这个双稳态触发器的稳定状态密切相关的。显然，仅仅能保持这两个状态的一种还是不够的，还要对状态进行控制，于是就加上了控制管 V_5、V_6。

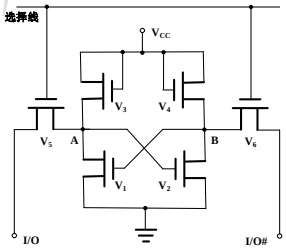


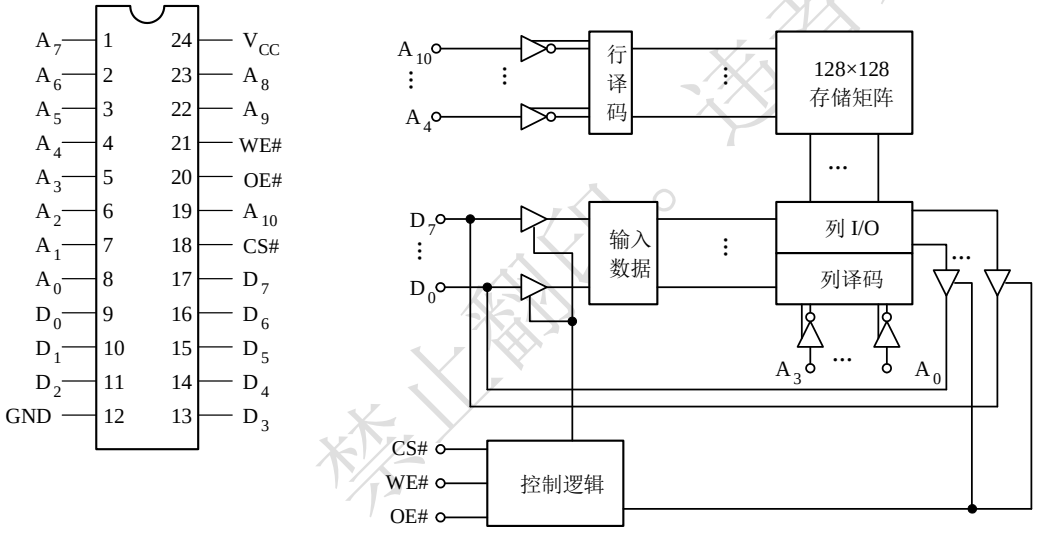
图 3.16 六管 SRAM 存储单元

当地址译码器的某一个输出线送出高电平到 V_5、V_6 控制管的栅极时，V_5、V_6 导通，于是，A 点与 I/O 线相连，B 点与 I/O#线相连。这时如要写“1”，则 I/O 为“1”，I/O#为“0”，它们通过 V_5、V_6 管与 A、B 点相连，即 A=“1”，B=“0”，使 V_1 截止，V_2 导通。而当写入信号和地址译码信号消失后，V_5、V_6 截止，该状态仍能保持。如要写“0”，则 I/O 线为“0”，I/O#线为“1”，这使 V_1 导通，V_2 截止。只要不掉电，这个状态会一直保持，除非重新写入一个新的数据。对所存的内容读出时，仍需地址译码器的某一输出线送出高电平到 V_5、V_6 管栅极，即此存储单元被选中，此时 V_5、V_6 导通。于是，V_1、V_2 管的状态被分别送至 I/O 线、I/O#线，这样就读取了所保存的信息。

2. 静态 RAM 的结构

静态 RAM 内部是由多个如图 3.16 所示的基本存储电路组成，容量为单元数与数据线位数之乘积。为了选中某一个单元，往往利用矩阵式排列的地址译码电路。例如，1K 单元的内存需 10 根地址线，其中 5 根用于行译码，另 5 根用于列译码。译码后在芯片内部排列成 32 条行选择线和 32 条列选择线，这样可选中 1024 个单元中的任何一个，而每一个单元的基本存储电路的个数与数据线位数相同。

常用的典型 SRAM 芯片有 6116、6264、62256、628128 等。Intel 6116 的引脚及功能框图如图 3.17 所示。6116 芯片的容量为 2K×8 位，有 2048 个存储单元，需 11 根地址线，7 根用于行地址译码输入，4 根用于列译码地址输入，每条列线控制 8 位，从而形成了 128×128 个存储阵列，即 16384 个存储位。6116 的控制线有三条，片选 CS#、输出允许 OE#和读写控制 WE#。



（a）6116 引脚　　（b）6116 功能框图

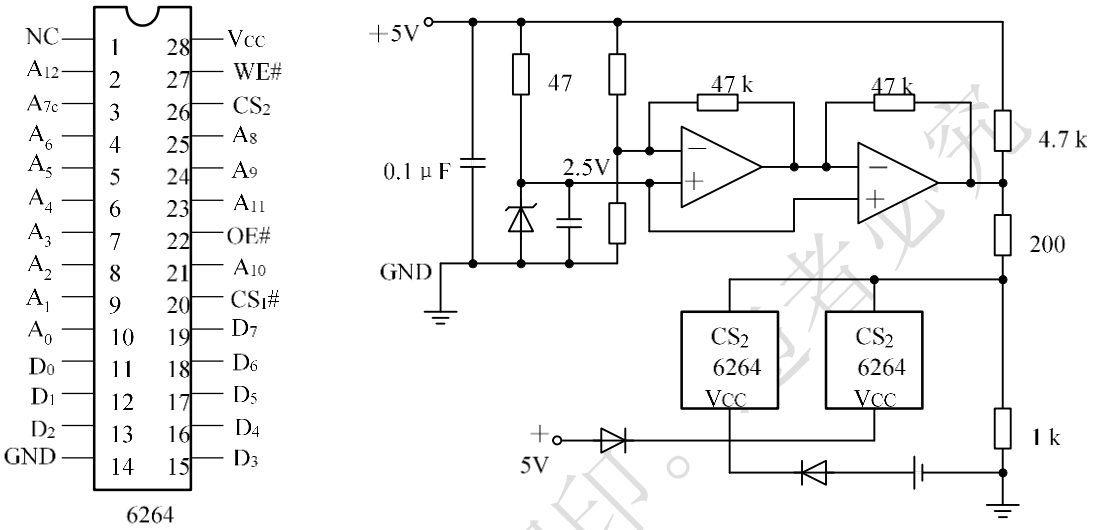
图 3.17　6116 引脚和功能框图

Intel 6116 存储器芯片的工作过程如下：

读出时，地址输入线 A_{10}～A_0 送来的地址信号经地址译码器送到行、列地址译码器，经译码后选中一个存储单元（8 个存储位），由 CS#、OE#、WE#构成读出逻辑(CS#=0，OE#=0，WE#=1），打开右面的 8 个三态门，被选中单元的 8 位数据经 I/O 电路和三态门送到 D_7～D_0 输出。写入时，地址选中某一存储单元的方法和读出时相同，不过这时 CS#=0，OE#=1，WE#=0，打开左边的三态门，从 D_7～D_0 端输入的数据经三态门和输入数据控制电路送到 I/O 电路，从而写到存储单元的 8 个存储位中。当没有读写操作时，CS#=1，即片选处于无效状态，输入输出的三态门为高阻状态，从而使存储器芯片与系统总线断开。6116 的存取时间在 85～150 ns 之间。

其他静态 RAM 的结构与 6116 相似，只是地址线不同而已。常用的型号有 6264、62256，它们都是 28 个引脚的双列直插式芯片，使用单一的+5V 电源，它们与同样容量的 EPROM 引脚相互兼容，从而使接口电路的连线更为方便。

值得注意的是，6264 芯片还设有一个 CS_2 引脚，通常接到＋5V 电源，当掉电时，电压下降到小于或等于+0.2V 时，只需向该引脚提供 2μA 的电流，则在 V_{CC}=2V 时，该 RAM 芯片就进入数据保护状态。根据这一特点，在电源掉电检测和切换电路的控制下，当检测到电源电压下降到小于芯片的最低工作电压（CMOS 电路为+4.5V，非 CMOS 为+4.75V）时，将 6264RAM 切换到由镍铬电池或锂电池提供的备用电源供电，即可实现断电后长时间的数据保护。数据保护电路如图 3.18 所示。



（a）6264 引脚　　（b）保护电路

图 3.18　6264 的数据保护电路

3.3.2 动态 DRAM

1. 动态 RAM 存储电路

由图 3.19 所示，DRAM 存放信息靠的是存储电容 C。电容 C 有电荷时，为逻辑“1”，没有电荷时，为逻辑“0”。但由于电容自身特性都存在漏电现象，因此，当电容 C 存有电荷时，过一段时间由于电容的放电形成电荷流失，造成信息丢失。解决的办法是进行刷新，即每隔一定时间(一般为 2ms）就要刷新一次，使原来处于逻辑电平“1”的电容的电荷又得到补充，而原来处于电平“0”的电容仍保持“0”。在进行读操作时，根据行地址译码，使某一条行选择线为高电平，于是使本行上所有的基本存储电路中的 MOS 管 V 导通，使连在每一列上的刷新放大器读取对应存储电容上的电压值，刷新放大器将此电压值转换为对应的逻辑电平“0”或“1”，又重写到存储电容上。而列地址译码产生列选择信号，所选中那一列的基本存储电路才受到驱动，从而可读取信息。

在写操作时，行选择信号为“1”，V 管处于导通状态，此时列选择信号也为“1”，则此基本存储电路被选中，于是由外接数据线送来的信息通过刷新放大器和 V 管送到电容 C 上。刷新是逐行进行的，当某一行选择信号为“1”时，选中了该行，电容上信息送到刷新放大器上，刷新放大器又对这些电容立即进行重写。由于刷新时，列选择信号总为“0”，因此电容上信息不可能被送到数据总线上。

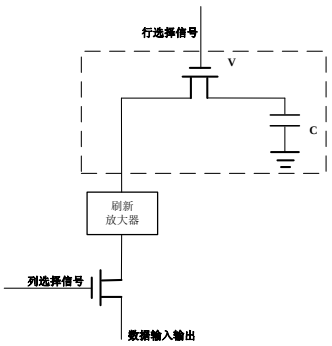


图 3.19　单管 DRAM 存储单元

2. 动态 RAM 芯片

DRAM 芯片 2164A（如图 3.20 所示）的容量为 64K×1 位，即片内有 65536 个存储单元，每个单元只有 1 位数据，用 8 片 2164A 才能构成 64K 字节的存储器。若想在 2164A 芯片内寻址 64K 个单元，则需要用 16 条地址线。但为减少地址线引脚数目，地址线又分为行地址线和列地址线，进行分时工作，这样 DRAM 对外部只需引出 8 条地址线。芯片内部有地址锁存器，利用多路开关，由行地址选通信号 RAS(Row Address Strobe)，把先送来的 8 位地址送至行地址锁存器加以锁存。由随后出现的列地址选通信号 CAS(Column Address Strobe)把后送来的 8 位地址送至列地址锁存器加以锁存。这 8 条地址线也用于刷新。刷新时一次选中一行，2ms 内全部刷新一次。Intel 2164A 的内部结构示意图如图 3.20 所示。

图中 64 K 存储体由 4 个 128×128 的存储矩阵组成，每个 128×128 的存储矩阵，由 7 条行地址线和 7 条列地址线进行选择，在芯片内部经地址译码后可分别选择 128 行和 128 列。在行地址锁存器中锁存的七位行地址 RA6～RA0 同时加到 4 个存储矩阵上，在每个存储矩阵中都选中一行，则共有 512 个存储电路可被选中，它们存放的信息被选通至 512 个读出放大器，经过鉴别后锁存或重写。锁存在列地址锁存器中的七位列地址 CA_6～CA_0（等同于地址总线的 A_{14}～A_8），在每个存储矩阵中选中一列，然后经过 4 选 1 的 I/O 门控电路（由 RA_7、CA_7 控制）选中一个单元，对该单元进行读写。2164A 数据的读出和写入是分开的，由 WE#信号控制读写。当 WE#为高时，实现读出，即所选中单元的内容经过三态输出缓冲器在 DOUT 脚读出。而 WE#当为低电平时，实现写入，DIN 引脚上的信号经输入三态缓冲器对选中单元进行写入。2164A 没有片选信号，实际上用行选 RAS#、列选 CAS#信号作为片选信号。

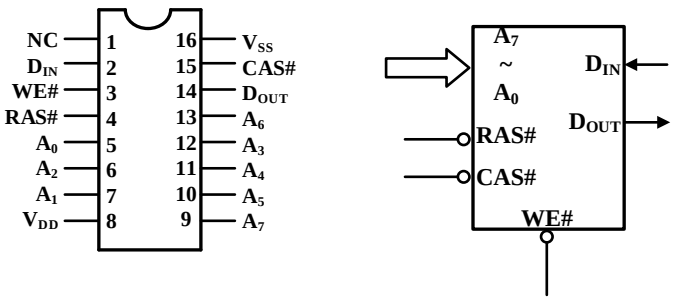


图 3.20 Intel 2164A 引脚与逻辑符号

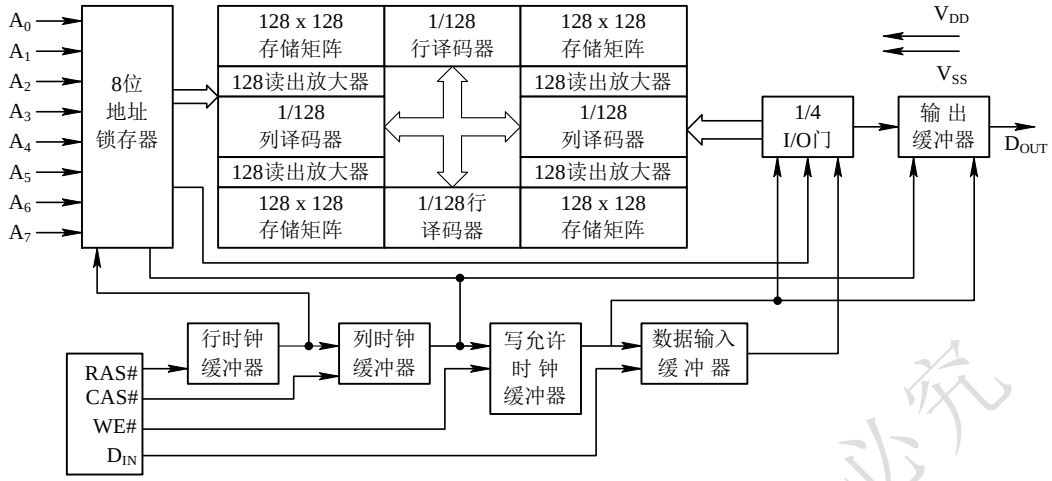


图 3.21 Intel 2164A 内部结构示意图

3. DRAM 存储条

由于微型计算机的实际配置内存已高达 16GB、32GB，服务器更高达 256GB，因此要求配套的 DRAM 集成度也越来越高，容量为 1Gb 以及更高集成度的存储器芯片已大量使用。通常，把这些芯片放在内存条上，用户只需把内存条插到系统板上提供的存储条插座上即可使用。

图 3.22 是采用 HYM59256A 存储芯片，构成 256K×9 位存储容量的存储条，其中，2 片 256K×4 位的存储芯片通过位扩展形成 256KB 的存储单元，1 片 256K×1b 的存储芯片作为奇偶校验。图中给出了引脚和方块图，其中 A_8～A_0 为地址输入线，DQ_7～DQ_0 为双向数据线，PD 为奇偶校验数据输入，PCAS#为奇偶校验的地址选通信号，PQ 为奇偶校验数据输出，WE#为读写控制信号，RAS#、CAS#为行、列地址选通信号，V_{DD} 为电源（+5V），Vss 为地线。30 个引脚定义是存储条的通用标准之一。

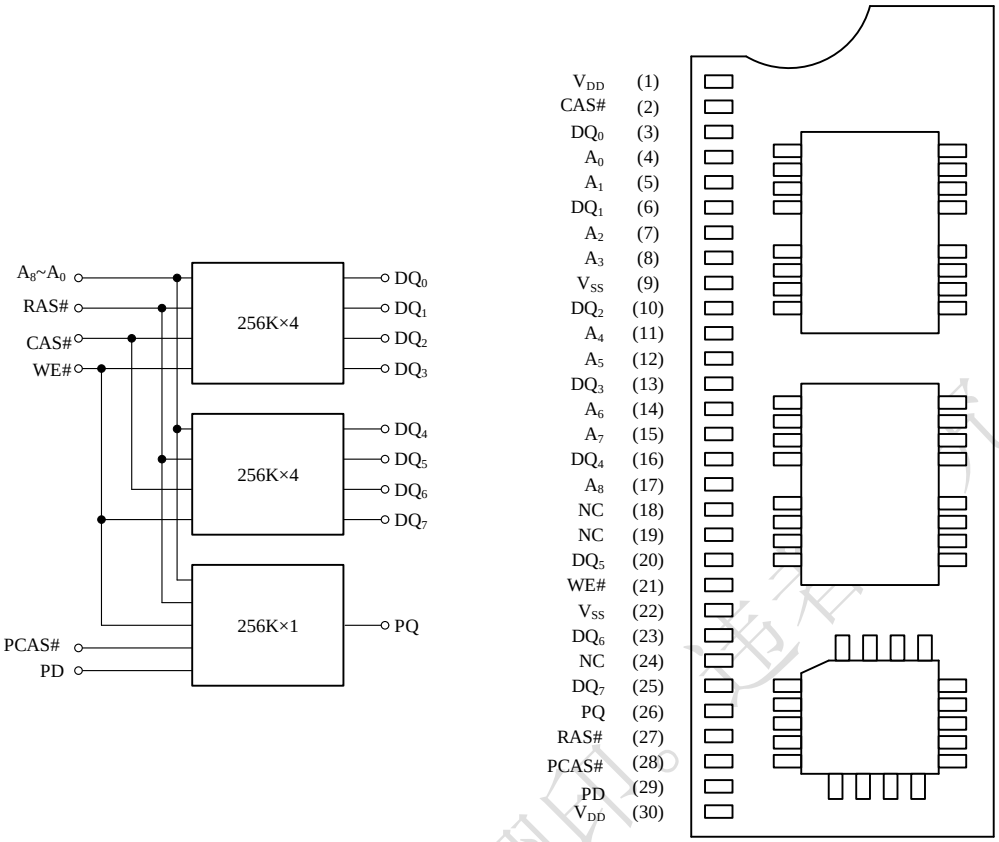


图 3.22 256K×9 位存储条

再如，1M×8 位的内存条，HYM58100 由 8 片 1M×1 位的 DRAM 组成，也可由 2 片 1M×4 位的 DRAM 组成。更高集成度的内存条请参阅存储器手册。

3.4 存储器与 CPU 的连接

在计算机系统中，计算机运行时所需的指令和数据都存放在存储器中。故而存储器芯片需要与 CPU 连接起来才能真正发挥作用。存储器与 CPU 之间相连接的信号线包括地址线、数据线和控制线。如 2.2.4 小节所述，CPU 访问存储器时，需要通过地址总线 AB 向存储器发送指定存储单元的地址，通过控制总线 CB 向存储器发出读或写的命令，被选中的存储单元还需要连接数据总线 DB 才能实现存储单元内容的输出或输入。

从原理角度看，存储器与 CPU 的连接表现为地址信号线、数据信号线和控制信号线的连接关系。从电气特性角度看，互联时需要考虑总线的负载能力、各信号线的时序配合等因素。而从工程设计角度看，在实际计算机系统中，CPU 和存储器往往处于不同的电路板上，通常 CPU 模块直接安装在计算机主板上，存储器则以内存条的形式（即独立的子电路板）通过插槽插入计算机主板。本小节首先分析存储器与 CPU 间的信号线连接关系，然后讨论内存条的有关技术。

3.4.1 地址空间与存储器连接

计算机中地址总线 AB 的宽度决定了存储器空间的最大寻址范围，常把这个寻址范围称为地址空间。例如，16 位宽度的地址总线可寻址空间为 2^{16}=64K，即可寻址 64K 个存储单元，存储单元的地址处于 0x0000 到 0xFFFF 的范围。

计算机地址总线的宽度即 CPU 地址线的数目。以 32 位 CPU 来说，如果 CPU 的地址线数目也是 32，其所能寻址的空间大小为 0~4G，若按照字节为单位进行编址（按字节编址的计算机），则可寻址 4GB；若按照字为单位进行编址（按字编址的计算机），由于字长是 4 字节，故可寻址 16GB。通常资料中分析的是按字节编址的计算机。

数据线数目（即数据总线宽度）决定了一次存储器操作可访问操作数（字）的位数，地址线数目（即地址总线宽度）决定了 CPU 的寻址空间大小。由于计算机中不仅要连接内存芯片，也会连接一些其他存储器芯片或接口电路单元，通常 CPU 可寻址的地址空间被划分成不同区域。在连接存储器芯片和 CPU 的时候，要根据地址空间的划分设计存储器芯片地址线和 CPU 地址线的连接方式。

例如，采用 256K×8 位的存储器芯片连接具有 20 根地址线的 CPU，图 3.23 显示了三种不同的连接方式。图 3.23 (a) 中，CPU 地址信号 A_{18} 和 A_{19} 均为低电平时存储器片选信号 CS 才会有效，故存储器芯片在地址空间的区域 0x00000 ~ 0x3FFFF。图 3.23 (b) 连接方式中，CPU 地址信号 A_{18} 为高电平、A_{19} 为低电平时存储器片选信号 CS 才会有效，故存储器芯片在地址空间的区域 0x40000～0x7FFFF；图 3.23 (c) 连接方式中，CPU 地址信号 A_{18} 为低电平、A_{19} 为高电平时存储器片选信号 CS 才会有效，存储器芯片在地址空间的区域 0x80000～0xBFFFF。

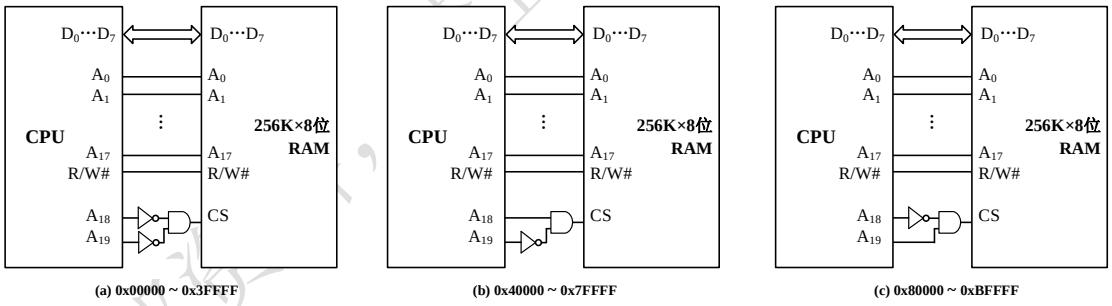


图 3.23 三种不同的存储器地址线连接

3.4.2 存储器扩展

计算机按照一定方式组织存储芯片可以实现预期的存储器规模。单颗存储器芯片的容量总是有限的，往往单颗个芯片不能满足计算机存储容量的需求；同时，存储器数据线的数量也可能与 CPU 的字长不匹配。所以，常需要使用多颗存储器芯片组合来设计计算机的存储子系统。针对不同容量需求，采用多颗存储器芯片组合来构造存储器系统称为存储器扩展。

在 2.2.1 小节中，图 2.3 给出了一种由四个独立的存储体（芯片）构成存储器系统的组织方式。具体来说，存储器的扩展方式包括了位扩展、字扩展和字位同时扩展三种情形。下面依次讨论这三种方式的细节。

1. 位扩展

位扩展，就是在存储器芯片的字数不变（即寻址范围不变）的前提下，进行数据位数扩展。图 3.24 为采用 8 片 1M×1 位的芯片扩展为 1M×8 位的 RAM 并与 CPU 总线连接的示意图。

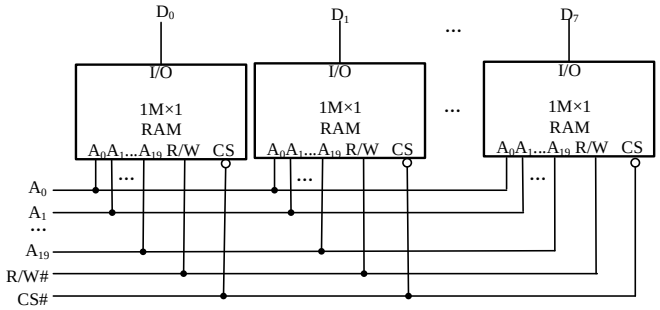


图 3.24　用 1M×1 位芯片位扩展为 1M×8 位的 RAM

图 3.24 所示连接示例中，进行存储器位扩展时，CPU 与 RAM 的地址总线、数据总线及控制总线的连接要求如下。

（1）每颗 RAM 芯片的一根数据线 I/O 分别连接 CPU 数据总线 D_7~D_0 的不同位；

（2）各芯片的地址线 A_{19}~A_0 均与 CPU 地址总线对应地址线相连（并联的结构）；

（3）每颗 RAM 芯片的读写控制线 R/W#均与 CPU 读写控制线连接，各芯片的片选信号线 CS#均与 CPU 的片选控制线 CS#连接。即控制信号线也采用并联的结构。

按照图 3.24 所示连接方式，所构成的 1M×8 位的 RAM 在 CPU 地址空间的区域 0x00000 ~ 0xFFFFF，按字节编址，每个存储单元 8 位（1 字节）。

2. 字扩展

字扩展，是在存储器芯片位数满足要求的前提下，进行字数扩展（即扩充寻址范围）。如图 3.25 所示，单颗存储器芯片为 256K×8 位，采用 4 片进行字扩展为 1M×8 位的 RAM，并与 CPU 总线连接的示意图。

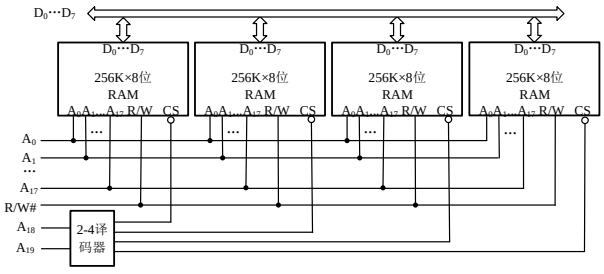


图 3.25　用 4 片 256K×8 位存储芯片扩展为 1M×8 位的存储器电路

图 3.25 所示连接示例中，进行存储器字扩展时，CPU 与 RAM 的地址总线、数据总线及控制总线的连接要求如下。

（1）每颗芯片的各数据线 D_7~D_0 均连接数据总线 D_7~D_0。虽然各芯片数据线是并联的，

但同一时间只会有一颗芯片的片选信号有效，故仅选择一颗芯片进行数据线与数据总线连接，其他芯片的数据线置于高阻状态，从而实现隔离；

（2）各芯片的低位地址线 A_{17}~A_0 与 CPU 地址总线对应地址线并联；

（3）高位地址线 A_{19}、A_{18} 通过 2 线-4 线译码器分别产生不同的译码输出信号，控制其中一个存储器芯片的片选端 CS 有效，使对应的芯片可操作；

（4）各芯片的读写控制线 R/W#与 CPU 读写控制线 R/W#分别并联方式连接。

3. 复合扩展

在位长和字数均不足时，需采用复合扩展方式。其具体过程为，先进行位扩展，然后再进行字扩展。如将 256K×1 位的芯片扩展为 1M×8 位的存储器系统，先用 8 片 256K×1 位的芯片进行位扩展，构成 256K×8 位的存储器。再将其作为整体进行字扩展，用 4 个 256K×8 位的存储器，从而构成 1M×8 位的存储器系统。关于数据线和地址线的连接方式，请读者自行分析，此处不再赘述。

3.4.3 嵌入式系统的存储器扩展

前述存储器扩展围绕计算机主存系统的设计。在微型计算机中，用于主存的 DRAM 芯片通过地址总线、数据总线和控制总线与 CPU 连接；用作外存的硬磁盘等则通过 SATA 等外存接口与计算机相连。通常需要运行的程序保存在外存中，在需要的时候通过外存接口传送至内存，再从内存送往 CPU。而在嵌入式系统中，存储器子系统不同，没有传统意义内存、外存的区分。嵌入式微处理器芯片常内置了一定容量 NOR Flash 闪存和小容量的 SRAM。此时 NOR Flash 闪存用于保存程序，而 SRAM 用于载入动态信息，二者均与 CPU 的地址总线、数据总线和控制总线连接。

如果嵌入式微处理器芯片内的存储器不够用，需要更大容量的存储器，就需要进行扩展。此时，根据需要选择扩展 RAM 还是扩展 ROM。如果需要扩充程序存储空间，常选择非易失性的 EEPROM、Flash 存储器。若需要扩展外部载入（如从 USB 接口或网络接口启动并下载）程序和数据的存储空间，可选择 SRAM 或者 DRAM。

由于不同类型的存储器需要不同类型的接口电路，嵌入式处理器内部预置外存接口电路时常常对 CPU 的地址空间做预分配。如，连接大容量的 NAND Flash 时，其地址范围被指定到地址空间的一个特定区域，而连接 SDRAM 时，被指定为另外一个区域。关于 ARM 处理器地址空间的预分配方案的详细信息，可参考本书第 5 章 5.4 节。

下面以三星公司 S3C2440 系列 ARM 处理器芯片为例，简要分析该芯片扩展不同类型外接存储器时的差异。关于 S3C2440 更多的细节，可参阅本书第 8 章 8.3.2 小节。

1) NOR Flash 闪存的存储器扩展设计

以 HY29LV160 NOR Flash 闪存芯片扩展与嵌入式微处理器连接，构成外置的扩展存储器电路，如图 3.26 所示。该芯片共有数据线 16 根，DQ[15:0]，其中 DQ[15]可用作地址线；地址线 20 根，A[19:0]，与 DQ[15]/A[-1]配合可构成 21 位地址。控制信号包括：片选 CE#（Chip Enable）、读使能 OE#（Output Enable）、写使能 WE#（Write Enable）、复位信号 RESET#、

就绪指示 RY/BY#（Ready/Busy Status）信号，以及字节选择信号 BYTE#，低电平表示仅使用低 8 位数据线。按照图示的电路连接，构成 16 位的数据宽度，其存储容量为 2M 字节。

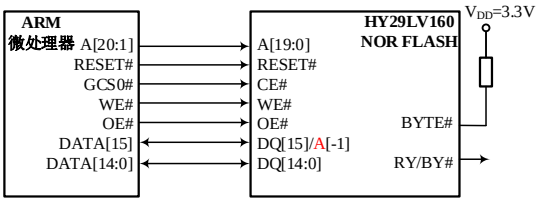


图 3.26 NOR Flash 与 ARM 微处理器的扩展存储器接口电路示意图

ARM 微处理器对 NOR Flash 的访问不需要额外的软件设置，系统在上电复位后，从 NOR Flash 的 0x0 地址开始执行第 1 条指令，即开始执行 NOR Flash 存储器的启动代码。

2） NAND Flash 闪存的存储器接口设计

如 3.2.5 小节所述，NAND Flash 以页为单位进行读和编程操作，以块为单位进行擦除操作。为有效支持这种读写方式，NAND Flash 接口电路较 NOR Flash 复杂，需要额外的控制信号，故需要 NAND Flash 控制器。S3C2440 系列芯片内部集成了 NAND Flash 控制器，该控制器可以完成 NAND Flash 接口到 ARM 处理器总线的转接。图 3.27 为 S3C2440 芯片集成的 NAND Flash 控制器的内部结构方框图。图中寄存器包括：NAND Flash 控制寄存器、NAND Flash 地址寄存器、NAND Flash 数据寄存器。

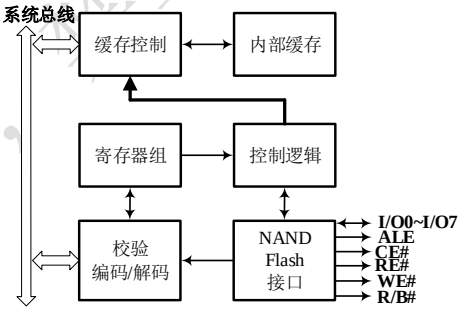


图 3.27 NAND Flash 控制器的内部结构方框图

相应地，NAND Flash 芯片也需要配套的接口电路。图 3.28 所示为 NAND Flash 芯片 K9F2808U0A（16M x 8 Bits）内部结构示意。该芯片的地址线和数据线复用 8 根 I/O 线；除了常规的片选 CE#、读使能 RE#（Read Enable）和写使能 WE#控制信号外，NAND Flash 需要额外的命令锁存使能 CLE（Command Latch Enable）、地址锁存使能 ALE（Address Latch Enable）、写保护 WP#（Write Protect）、就绪指示 R/B#（Ready/Busy）信号。

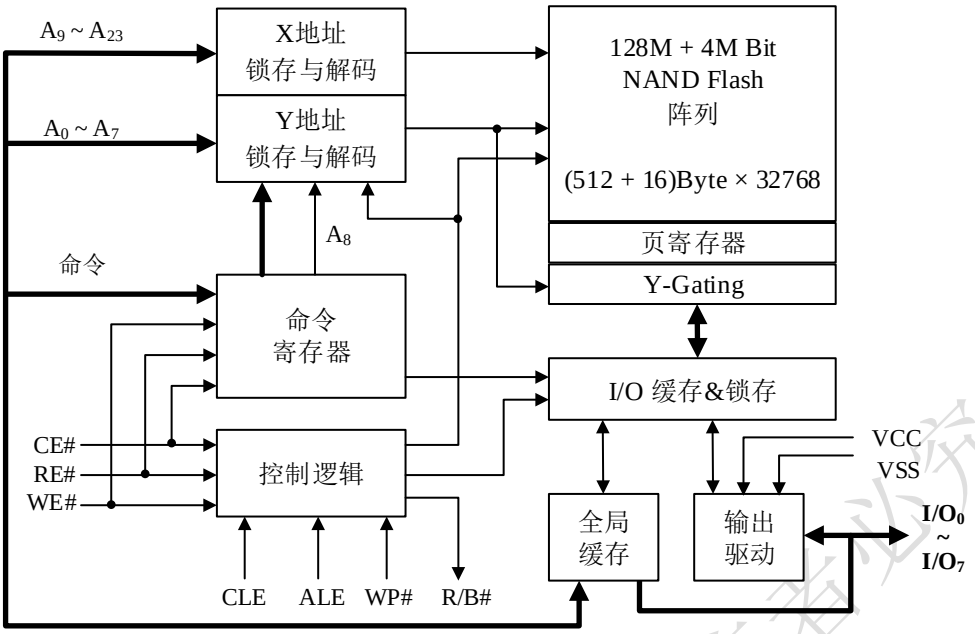


图 3.28 NAND Flash 芯片 K9F2808U0A 内部结构示意图

访问 NAND Flash 时，不同控制信号组合完成如下功能。①若 CLE 与 WE#有效，则在 I/O 上传输命令（该命令指示后续要传输的是命令、地址还是数据）。②若 ALE 与 WE#有效，则在 I/O 上传输地址（该地址指示拟访问的存储单元）。③写保护 WP#（Write Protect）信号用于防止电源变化期间的意外写入。④就绪指示 R/B#用来指示当前写操作或读操作是否完成。

图 3.29 所示为 NAND Flash 与 ARM 嵌入式微处理器的接口电路示意图。在图 3.29 连接方式的基础上，还需要软件配合才能够完成 NAND Flash 的读写操作。其大致过程为：①写 S3C2440 内部的 NAND Flash 控制寄存器；②写 S3C2440 内部的 NAND Flash 地址寄存器；③读或者写 S3C2440 内部的数据寄存器。

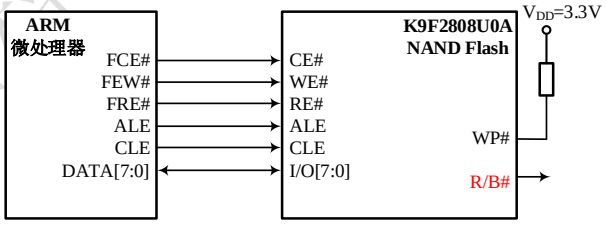
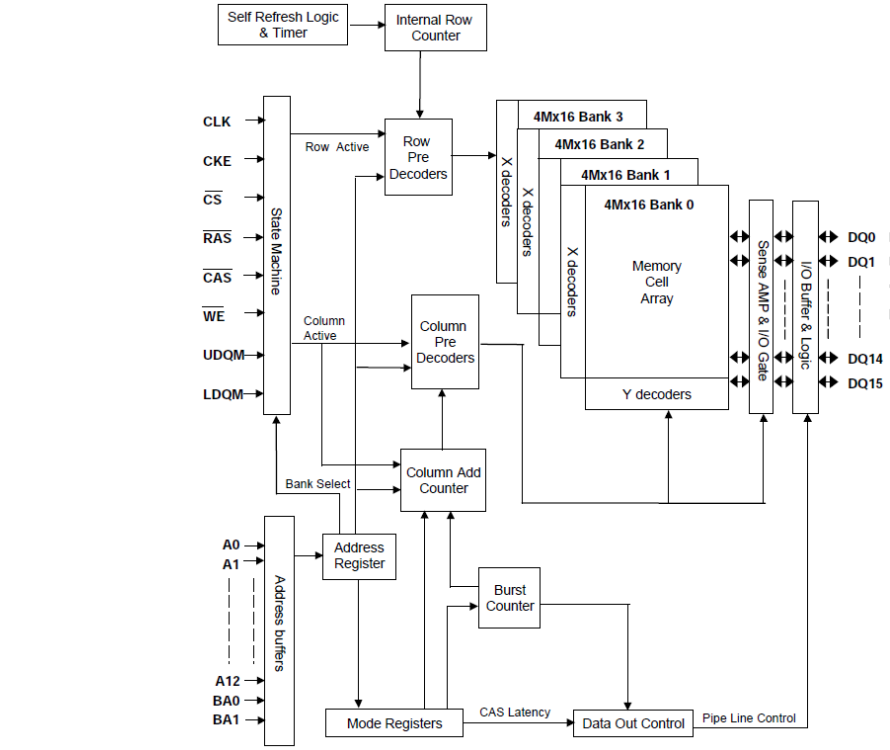


图 3.29 NAND Flash 与 ARM 微处理器的接口电路示意图

3) SDRAM 的存储器扩展设计

SDRAM 常设计为多 Bank 结构，每一个独立的 Bank 对应一个存储矩阵。这种设计的原因在于，DRAM 需要定时进行刷新操作以保证存储信息的电容内有足够的电荷。以两 Bank 的 SDRAM 芯片为例，第一个 Bank 进行数据读取时，第二个 Bank 可以进行刷新（预充电）操作；而第二个 Bank 刷新结束后被读取时，第一个 Bank 又可以进行刷新操作。从而提高了 SDRAM 存储器的访问效率。

图 3.30 所示为 HY57V561620（L）T 型号的 SDRAM 芯片内部结构示意图。该芯片内部共有 4 个 Bank。Bank 地址线 BA0、BA1 用于选择芯片内部的 Bank。由于 SDRAM 是同步动态存储器，故芯片提供外部输入时钟引脚 CLK，CKE（Clock Enable）为该时钟的使能信号。DQ0~DQ15 为数据线；A0~A12 为地址线；控制信号线包括片选 CS#、行地址选通（Row Address Strobe）RAS#、列地址选通（Column Address Strobe）CAS#、写使能 WE#信号。UDQM 和 LDQM 为字节屏蔽设置， LDQM 有效时屏蔽低 8 位、UDQM 有效时屏蔽高 8 位，从而实现字节访问的控制。



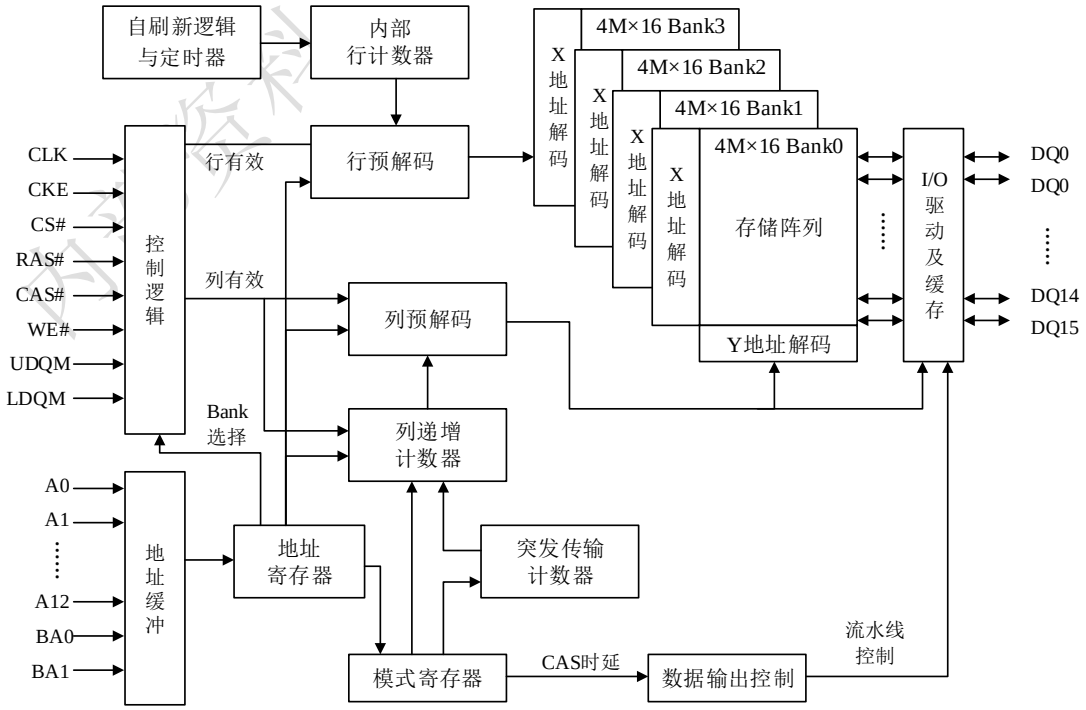


图 3.30 HY57V561620 内部结构示意图

此处以三星公司 S3C2440 系列 ARM 处理器芯片为例说明进行 SDRAM 存储器扩展的基本方法。S3C2440 片内集成了 SDRAM 访问控制器，可以直接连接外部的 SDRAM 芯片。在不同芯片容量、存储单元数目、存储单元位宽情形下，可根据表 3.5 所示选择 SDRAM 芯片的颗数及 Bank 地址线与处理器地址线的连接方式。

与 ARM 微处理器进行电路连接时，要根据实际扩展的主存容量、芯片颗数、单元数、芯片位宽与 Bank 之间的关系进行配置，如表 3.5 所示（表中仅显示了少部分配置，完整配置可参阅 S3C2440 系列 ARM 处理器芯片手册）。

表 3.5　s3c2440SDRAM 控制器 Bank 地址配置（部分配置）

每 Bank 地址空间	数据总线位宽	芯片容量	Bank 地址线	配置总容量（单元数×位宽×Bank 数×颗数）
8MB	×8	64Mb	A[22:21]	(2M × 8 × 4B) × 1
8MB	×16	64Mb	A[22]	(2M × 16 × 2B) × 1
16MB	×8	128Mb	A[23:22]	(4M × 8 × 4B) × 1
16MB	×16	128Mb	A[23:22]	(2M × 16 × 4B) × 1
32MB	×16	64Mb	A24	(8M x 4 x 2B) x 4
32MB	×8	256Mb	A[24:23]	(4M × 16 × 4B) × 1
32MB	×16	256Mb	A[24:23]	(8M × 8 × 4B) × 1

图 3.31 所示为 S3C2440 系列 ARM 微处理器与 HY57V561620（L）T SDRAM 的连接示意图。HY57V561620（L）T 的存储器按照 4M×16 位×4Banks 方式组织存储单元，依据表 3.5 所示，处理器的地址线 A23、A24 分别连接 SDRAM 的 BA0 和 BA1，构成 32MB（256Mb）的总存储容量。需要注意，按照图 3.31 完成硬件连接后，还需要配置 s3c2440 芯片中与一些相关的控制寄存器（SDRAM 刷新参数、突发模式等）后 SDRAM 才可以正常使用。

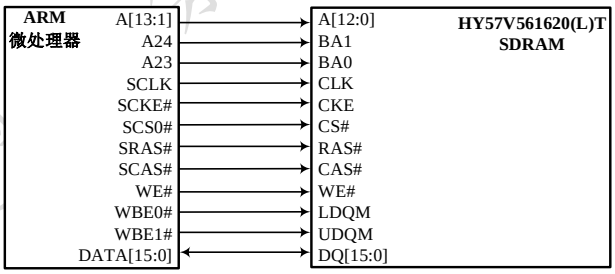
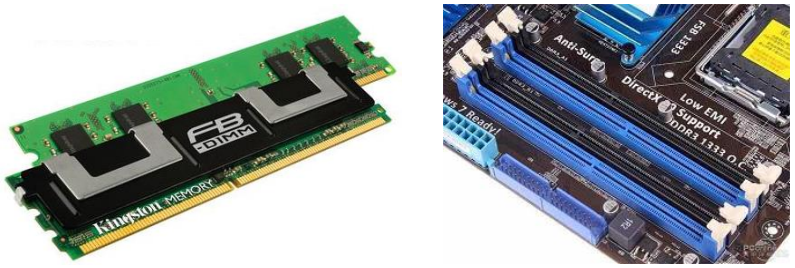


图 3.31 ARM 微处理器与 HY57V561620（L）T SDRAM 的连接示意图

3.4.4 存储器模块

早期的计算机中，存储器芯片直接固化在线路板上，无法拆卸更换，这导致计算机内存的扩展非常麻烦。为了便于存储器的升级扩充，逐渐形成了存储器模块。存储器模块是一块小的电路板，电路板上安装了一定数量的存储器芯片，模块可以以插槽的方式安装至计算机主板，这种模块被称作内存条，如图 3.32 所示。



(a)内存条　　(b)内存条插槽

图 3.32 内存条及计算机主板上的插槽

1. 内存条的组成

内存条由内存颗粒、PCB 电路板、SPD 芯片、引脚（俗称金手指）以及阻容元件组成。

1) 内存颗粒

内存颗粒指的是采用一定的封装技术进行封装的 DRAM 芯片。芯片封装将芯片与外界隔离，可防止空气中的杂质腐蚀芯片内的电路，同时也便于芯片的安装和运输。早期采用双列直插的 DIP（Dual In-line Package）封装，后来为了减小芯片在电路板上的占用面积，内存颗粒常采用 TSOP（Thin Small Outline Package）封装或 BGA（Ball-Gird-Array）封装。一般在封装的外表面会印刷厂家、产品编号、存储容量、数据宽度、存取速度、工作电压等基本参数。

2) SPD 芯片

SPD(Serial Presence Detect）是小型 E^2PROM 芯片，用于记录内存条出厂时预先存入的基本参数，如存取速度、工作频率、存储容量、工作电压等。计算机启动时自动读取 SPD 信息，进而依据读取的参数对主存进行设置，以便让内存运行在最佳工作状态。

3) PCB 电路板

内存条的 PCB（Print Circuit Board）用于安装内存颗粒、SPD 芯片等器件，同时将芯片引脚与金手指连接。为了减小电路板的体积、提高稳定性，通常采用 4 层或 6 层的多层电路板结构。

4) 金手指

金手指则是内存条电路板上的引脚，用于与计算机总线连接。内存条插入主板上的插槽，插槽既起到固定内存条的作用，也将金手指连接到计算机总线上的相关信号线。按照引脚布局，常见的金手指布局有：SIPP（Single In-line Pin Package）、SIMM（Single In-line Memory Modules）、DIMM（Dual In-line Memory Module）等形式。SIPP 内存条是单排插针的形式，SIMM 内存条的电路板两侧金手指都提供相同信号，DIMM 内存条不像 SIMM 那样两侧金手指是互通的，电路板两侧金手指各自独立传输信号。

5) 阻容元件

内存条的电路板上会放置很多阻容元件，用于提高信号完整性，如防止信号反弹、滤除高频干扰等。

2. 内存条的演变

随着存储器技术的发展，内存速度与容量不断提升，通常隔几年内存条就会更新换代。不同代的内存条最直观的变化是金手指，从 DIP、SIPP 发展到 SIMM，SIMM 从 30 引脚扩展到 70 引脚，后来又采用 DIMM，DIMM 从 168 引脚扩展到 184 引脚。

除了上述内存条接口外观的变化，不同代内存条也通过优化 DRAM 存储单元读写控制过程来提升内存条的访问速率。接下来简要分析常用 DRAM 读写控制优化技巧。

1） 异步 DRAM

早期的内存频率与 CPU 外频不同步，采用异步 DRAM。在传统 DRAM 基础上，发展出 FPM DRAM（Fast Page Mode DRAM）和 EDO DRAM（Extended Data Out DRAM）。

传统 DRAM 的访问，需要依次经过“发送行地址”、“发送列地址”、“读写数据”三个阶段，一次内存访问的时间是三个阶段所需时间之和。在 FPM（快速页面模式）中，访问地址连续（列址相同）的多个存储单元时，除了访问第一个数据需要发送列地址外，后续访问只需要经历“发送行地址”、“读写数据”两个阶段，从而缩短了访问时间。而 EDO（扩展数据输出）模式中，“发送地址”和“读写数据”的操作在时间上可重叠，进一步缩短了内存访问时间。具体表现为，在输入下一个行地址时，仍然允许数据输出进行，扩展了数据输出的时间，“EDO”也因此而得名。

2） SDRAM

SDRAM（Synchronous　DRAM）即同步 DRAM，内存频率与 CPU 外频同步。同时内存条的金手指采用 168 引脚的 DIMM，开始支持 64 位的数据位宽，大幅提升了数据传输效率。在同步方式下，送往 SDRAM 的地址、数据和控制信号都在一个时钟信号的上升沿被采样和锁存，SDRAM 输出的数据也在另一个时钟上升沿锁存到芯片内部的输出寄存器。这种对齐到时钟上升沿的操作方式有利于不同类型操作按照流水线方式并行。

SDRAM 还增加了时钟信号和内存命令的概念。SDRAM 收到地址和控制信号之后，在内部进行操作。在此期间，处理器和总线主控器可以处理其它任务（例如，启动其它存储体的读操作），无需等待，从而提高了存储系统性能。

3） DDR SDRAM

DDR（Double Data Rate，双倍速率）SDRAM 是在传统 SDRAM 基础上的革命性演进。在 DDR SDRAM 出现后，传统 SDRAM 被称为 SDR（Single Data Rate）　SDRAM。SDR 仅在时钟脉冲的上升沿进行一次写或读操作，而 DDR SDRAM 在时钟的上升沿与下降沿各传输一次。DDR SDRAM 内部有两个存储器，以乒乓方式交替工作，还有 2bits 的预取缓冲，可在时钟上升沿和下降沿都进行一次写操作或读操作，因而数据传输速度是同频率 SDR SDRAM 的两倍。

DDR SDRAM 采用 184 脚的 DIMM 插槽，防呆缺口从 SDR SDRAM 的两个变成一个，常见工作电压 2.5V。DDR 工作频率有 100/133/166/200/266MHz 等，由于是双倍速率，因此芯片型号常标识工作频率×2，即标称为 DDR200、266、333、400 和 533。DDR 内存条最初只有单通道，后来出现了支持双通芯片组，两根 DDR-400 内存条组成双通道，让内存的带宽直接翻倍。DDR 内存条支持的容量从 128MB 到最大 1GB。

4） DDR2 SDRAM

DDR2/DDR II（Double Data Rate 2）SDRAM 采用了 4bits 预读取技术，数据通过四条线路串行传输到 I/O 缓存区，在不改变内存单元的情况下，DDR2 能达到 DDR 数据传输速度的两倍。如 DDR2-400，其数据传输速率为（100MHz×4）×（64b/8b）=3200MB/s=3.2GB/s。

DDR2 的标准电压下降至 1.8V，比 DDR 产品更为节能。同时采用了 OCD(Off-Chip Driver)、ODT（On Die Terminator）等技术提高信号完整性。其工作频率从 400MHz 到 1200MHz，主流的是 DDR2-800。DDR2 内存条支持容量从 256MB 至最大 4GB。

5） DDR3 SDRAM

DDR3 SDRAM 采用了 8bits 预取，如 100MHz 的 DDR3-800，带宽可达到：（100MHz×8）×（64b/8b）＝ 6.4GB/s。这使得同样核心频率的 DDR3 能够提供两倍于 DDR2 的带宽。DDR3 内存条的工作电压 1.5V，容量从 512MB 至最大 8GB。此外 DDR3 还采用了 CWD、Reset、ZQ、STR、RASR 等新技术，让内存在休眠时也能够随着温度变化去控制对内存颗粒的充电频率，以确保系统数据的完整性。

6） DDR4 SDRAM

从 DDR1 到 DDR3，每一代 DDR 技术的内存预取位数都会翻倍，以此达到内存带宽翻倍的目标。若再次提升预取位数，I/O 控制器的频率需要再次翻倍，并且增加数据线数量，在频率已经很高时，技术难度和成本都会大幅增加。因而 DDR4 在预取位上保持了 DDR3 的 8 位设计，转而提升 Bank 数量，它使用的是 BG（Bank Group）设计，4 个 Bank 作为一个 BG 组，可自由使用 2~4 组 BG，每个 BG 可独立操作。如，使用 2 组 BG，则每次操作的数据就是 16 位；使用 4 组 BG 则能达到 32 位操作。相当于提高了预取位宽。

DDR4 内存条在 2014 年推出，首款支持 DDR4 内存条的是 Intel 旗舰级的 x99 平台。DDR4 内存的针脚从 DDR3 的 240 个提高到了 284 个，防呆缺口也与 DDR3 的位置不同。在散热允许情况下，DDR4 采用 3D 堆叠封装技术，使得单根内存条的容量成倍增加。DDR4 内存的标准电压是 1.2V，工作频率从 2133MHz 到最高 4200MHz。单条容量有 4GB、8GB 和 16GB，已基本取代了 DDR3。

7） DDR5 SDRAM

2020 年 7 月，JEDEC 协会正式公布 DDR5 标准。DDR5 主要特性是芯片容量，在采用 4GB 内存颗粒时，理论上可以支持 512GB 的最高容量。DDR5 工作电压则从 1.2V 降至 1.1V，功耗减少 30%。工作频率最低 4800MHz，最高 6400MHz。每个模块使用两个独立的 32/40 位通道，支持 ECC（Error Correcting Code）。此外，DDR5 具有改进的命令总线效率，有利于进一步提高传输效率。

3.5 高速缓存

由于 CPU 的运行速度比大容量主存（常用 DRAM）的存取速度高很多，主存的访问速度往往限制了 CPU 性能的发挥。高速缓存（Cache）是一项用来协调 CPU 与主存速度差异的技术。Cache 是介于 CPU 与主存之间的小容量存储器，存取速度比主存快。Cache 通常为半导体存储器，采用 SRAM 或其他访问速度高的易失性存储器。

Cache 保存大容量主存中部分信息的副本，CPU 访问存储器时，首先在 Cache 中查找所需信息是否在 Cache 中，如果在直接从 Cache 读取，如果不在，则从内存获取，并按照预先制定的策略更新 Cache，将所需信息调入 Cache。如果 CPU 能够从 Cache 中获取大部分所需信息，那么计算机的整体性能就会有较为明显的提升。本节首先介绍与 Cache 有关的基础知识，随后分析 Cache 系统设计中较为基本的两个问题：主存地址与缓存地址的映射及转换、Cache 内容的替换策略。

3.5.1 Cache 概述

1. 局部性原理

统计结果表明，在一段时间内，程序运行过程对存储器的访问常集中在一个较小的地址范围内。这是由于程序的代码在存储器中是连续存储的，循环体、子程序调用等会重复执行，且程序对数组和变量等数据的访问也有一定的重复性。这种对存储器局部范围频繁访问，而此范围外访问较少的现象，称为局部性原理。

局部性有两种不同的形式：时间局部性和空间局部性。时间局部性是指被访问过的存储器位置很可能在不远的将来会被再次访问。空间局部性则是指，如果一个存储器位置被访问了一次，那么程序很可能在较短的时间内会访问与之相邻的另一个存储器位置。上述局部性原理对硬件和软件的设计都有极大的影响。如，Cache 采用小容量 SRAM 存放频繁访问的程序和数据，能够在很大程度上提升程序运行速度。3.6 节将要讨论的虚拟存储技术，也利用了局部性原理。

2. Cache 的组织方式

Cache 的基本单元称为行（Line，Cache line）或字块或块。每个字块应包括的基本信息如下。有①数据字段：保存从主存单元复制过来的数据，单位是（区）块。每个区块的大小一般介于 4~128 字节之间，典型大小为 32 或 64 字节。②标记字段：保存数据字段在主存中的地址信息，又称为地址标记寄存器，记为 Tag。③有效位字段：指示区块和 Tag 是否有效。

依据 Cache 更新与替换策略的不同，Cache 字块中还可以包含如下信息。④一致性控制位（“脏”位）字段：指示区块数据是否被 CPU 更新但并未写回至主存。⑤替换控制位字段：向替换算法指示区块状态。

图 3.33 所示为一个典型的 Cache 行，每行的缓存数据是 512 位、标记信息是 14 位、有效位 1 位、一致性控制位 1 位、替换控制位 2 位，故而存储每行需要 530（=512+14+1+1+2）位。如果要实现 32 个 Cache 行，则需要 16960（=530×32）位。需要注意的是，图 3.33 所示为 Cache 行的逻辑结构，在实际电路实现时，依据 Cache 管理方式的不同，往往图 3.33 中标记信息项会采用与数据信息项不同的电路实现方式。

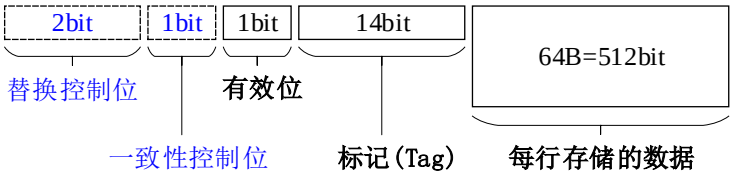


图 3.33 Cache 行的典型结构

Cache 系统中，主存是以区块为单位映像到 Cache 中。以 CPU 读取一个字节的数据为例，如果所需字节不在 Cache 中，在 CPU 从内存中读取数据的同时，Cache 控制器将把该

字节所在的整个区块从主存复制到 Cache。

3. 多级 Cache 结构

现代计算机中，一般采用多级 Cache 结构，典型的包括一级缓存（L1 Cache）、二级缓存（L2 Cache）和三级缓存（L3 Cache）。L1 Cache 集成于 CPU 芯片内，分为指令缓存 I-Cache 和数据缓存 D-Cache。分别用来存放指令和数据，两个 Cache 可同时被 CPU 访问，减少了指令和数据争用 Cache 造成的冲突。Cache 由 SRAM 组成，故在 CPU 芯片面积受限情况下，L1 Cache 容量不可能做得太大。通常 L1 Cache 容量介于 32~256KB 之间。

L2 Cache 分 CPU 内部和 CPU 外部两种。早期使用外部 L2 Cache，集成在主板或 CPU 电路板上；后来都集成到 CPU 芯片内部。L2 Cache 不区分数据 D-Cache 和指令 I-Cache。一般微型计算机的 CPU 中，L2 Cache 的容量一般为 512KB，而服务器和工作站级 CPU，其 L2 Cache 为 256~1MB，还有些超过 2MB。

L3 Cache，早期也置于 CPU 外的主板上，随着集成电路工艺水平提升，后来也全部集成到 CPU 芯片内。在多核的处理器芯片中，L3 Cache 常用于多个 CPU 核心间共享数据的缓存。

4. Cache 的命中率

Cache 的容量远远小于内存，同一时刻只能存放内存的一部分数据。局部性原理也不能保证所请求的数据全部在 Cache 中。任一时刻 CPU 能从 Cache 中获取所需数据的几率，称作 Cache 命中率（Hit Rate）。命中率是评价 Cache 性能的关键指标，命中率越高，总体性能越好。命中率的计算方法为：$h=N_C/(N_C+N_m)$。式中，N_C 和 N_m 是对 Cache 和主存的存取次数。只有当 N_C 足够大时，h 才能趋于 1。

影响命中率的因素很多，例如 Cache 容量、Cache 结构、Cache Line 大小、地址映射方法、替换算法、写回操作处理方法等。有统计表明，拥有二级缓存的 CPU， L1 Cache 命中率约为 80%。即， CPU 从 L1 Cache 获得的有用数据占总数据量的 80%，剩 20%将从 L2 Cache 中读取。在拥有 L3 Cache 的 CPU 中，仅约 5%的数据需要从主存中调入。

5. 影响 Cache 性能的因素

命中率越高，CPU 从 Cache 获取指令和数据的可能性越大，单位时间里访问主存的次数就越少，从而提高 CPU 的运行效率。

常见 Cache 脱靶的原因包括：（1）分块太小。程序开始执行时，主存块逐步复制进 Cache，因此容易脱靶，需经过一段时间后 Cache 才装满。首次执行产生脱靶的次数，与分块大小有关，块越大，不命中次数就越小。（2）容量太小。不能将所需指令和数据都调入 Cache，因此频繁的替换，导致 CPU 访问慢速主存次数增多。（3）替换进的主存块过大或过多。替换进 Cache 的主存块数目太多，会把下次要访问的指令或数据替换出去；数据块太大，替换所传数据量越大；Cache 所含块数减少，少数块刚装入就被覆盖掉。这些都导致命中率下降。

针对存在的问题，采取相应的方法，以提高 Cache 命中率。例如，（1）增大 Cache 容量。Cache 太小致命中率太低，过大改善不明显。一般选 Cache 与内存容量比 4:1000，命中率 90%以上。（2）通过结构设计减少不命中次数。如把指令、数据两类 Cache 分开，并采用二、三级 Cache 结构。（3）通过预取技术提高命中率。预测将要访问的指令和数据，提前将下条要执行指令取入 Cache，提高 CPU 取指令的速度。

3.5.2 地址映射与转换

主存与 Cache 间以块为单位进行信息交换。主存和 Cache 是两个物理上独立的存储器，在各自的地址空间编址。故而主存块调入 Cache 时，需要同时记录该块在主存中的地址及其在 Cache 中的地址（即图 3.33 所示的标记信息）。常把记录块主存地址和 Cache 地址映射关系的表称作块表，为保证查询块表的速度，块表使用一种称作相联存储器[2]（Associative Memory）的特殊存储器件实现。

由于 Cache 和主存在存储空间上的差异巨大，主存中不同位置的块可能会被调入 Cache 中同一个位置（一个块的区域），即若干个主存块将映射到同一个 Cache 块。按照不同的地址对应方法，有以下三种地址映射方案。

1. 全相联映射方式

全相联映射方式的地址映射规则是，主存的任意一块都可映射到 Cache 的任意一块。需要将主存与缓存都分成相同大小的数据块，主存的某一数据块可以调入 Cache 任意一块的空间中。假设 Cache 的块数为 2^C，主存的块数为 2^M，则主存块和 Cache 块可能的对应关系如图 3.34 所示，映射关系共有 $2^C \times 2^M$ 种。

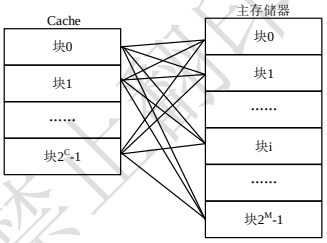


图 3.34 全相联映射方式

主存的块数为 2^M 时，主存地址可分为块号 M 和块内地址 W 两部分。而 Cache 的块数为 2^C 时，Cache 地址可分为块号 C 和块内地址 W 两部分。W 指向块内当前被访问数据所在的位置，对于主存和 Cache 来说都是同一个值。通过查找一个建立在相联存储器中的块号映射表（块表），即可实现主存地址到 Cache 地址的转换。

图 3.35 给出了块表的格式和地址变换规则。主存中块号为 M_i 的数据块调入 Cache 中的某个位置（块号 C_j 的位置）后，就会在块表中添加一条映射关系。当 CPU 需要访问主存数据的时候，地址线上给出的是拟访问数据的主存地址；①根据这个主存地址中的块号 M_i，去块表中查询其对应的 Cache 块号 C_j；②如果找到，则 Cache 命中，③将 Cache 块号 C_j 与块内地址 W 合成得到该主存地址对应的 Cache 地址；进而④利用该地址访问到 Cache 中的特定存储单元。如果没有在块表中找到块号 M_i 对应的信息，则 Cache 未命中。

[2]相联存储器（又称关联存储器），是一种根据存储内容来进行存取的存储器，可以实现快速地查找块表。这种存储器既可以按照地址寻址也可以按内容寻址，常用于 Cache 映射表、路由表等关系结构数据的存储。为了区别于传统存储器，也称为按内容寻址的存储器。

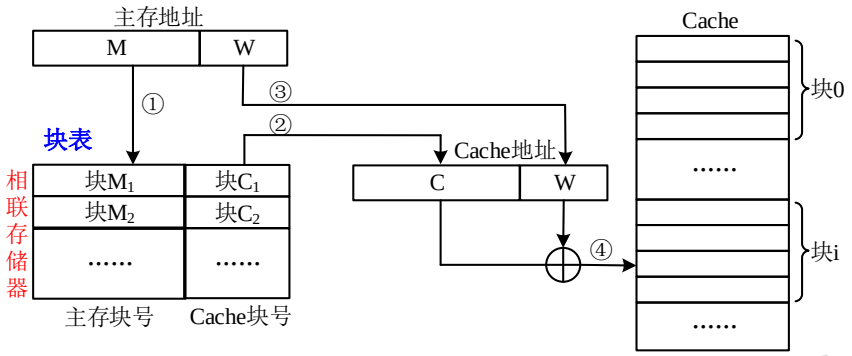


图 3.35　全相联映射的地址转换

如 3.5.1 小节所述，在组织 Cache 时，每个 Cache 块（Cache line）的基本信息包括：数据字段、标志字段（即 Tag）、有效位字段。图 3.35 中的块表的大小（所能存储映射关系的数目）和 Cache 的块数 2^C 一致，即每个 Cache 块对应块表的一行。故在实现块表[3]时，可以直接把 Cache 块 C_j 所对应主存块号 M_i 直接作为块 C_j 的 Tag（Tag 采用相联存储器实现），从而通过比较 M 位与 Cache 块的 Tag 实现块表查询[4]。

全相联映射方式的优点是：Cache 存储空间利用率高，不易产生冲突，命中率比较高。其缺点在于：比较和替换策略都需要硬件实现，电路复杂，只适用于小容量 Cache。访问相关存储器时，每次都要与全部内容比较，速度低，成本高，故应用较少。

2. 直接相联映射方式

直接相联映射，是将主存空间按 Cache 大小分成若干页，页内再分块，每页的大小和 Cache 容量相同。地址映射时，主存储器的某一块只能映射到 Cache 中一个特定块的位置。如图 3.36 所示。主存中各页内相同块号的数据块都映射至 Cache 中块号相同的地址，但同时只能有一个页的块存入缓存。在这样的规则下，不需要像前述全相联映射方式那样用块表来存储主存块与 Cache 块的映射关系，只记录调入块的页号即可。

[3] 图 3.35 中块表仅是一个逻辑结构，实际电路实现时可以有多种不同的方式。若基于相联存储器实现块表，实际上仅需要按 Cache 块号递增顺序存储其对应的主存块号。检索特定主存块号 M_i 时，如果命中，相联存储器可以直接输出对应 Cache 块的选择信号，或者输出 Cache 块号。

[4]在一些计算机组成原理的教材中，并未采用术语“块表”，而直接使用标记字段（Tag）来描述 CPU 访存的过程。全相联映射方式情形，若主存的块数为 2^M，则每个 Cache 块的 Tag 为 M 位。

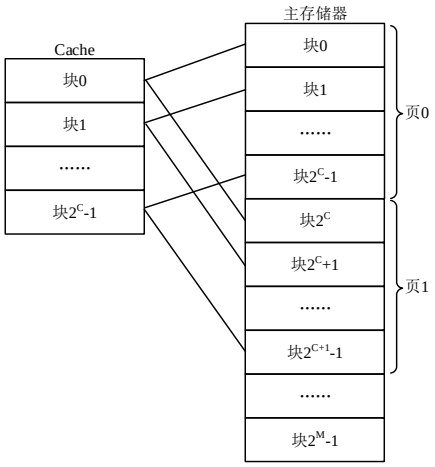


图 3.36 直接相联映射方式

假设主存大小是 Cache 的整数倍，Cache 容量为 2^C 块，主存容量是 2^M 块。每 2^C 块为一页，则主存有 $2^M/2^C$ 页（令 T=M-C，则主存有 2^T 页）。主存页中任意数据块号 j 映射到 Cache 中 i 块号时，i 和 j 需要满足关系式：$i = j \bmod 2^C$，式中 mod 表示做模运算。

CPU 拟访问的主存数据，一定是在 2^T 页中的某一页，在该页中 2^C 块中的某一块。即，主存地址由三部分组成：页号 T、块号 C 和块内地址 W，如图 3.37 所示。为了记录调入 Cache 的各块在主存中位置，每个 Cache 块分配了一个标记字段 Tag，标识该块原来在主存中的页号。当一主存块调入 Cache 时，会将主存地址的页号 T 存入 Cache 块的标记字段 Tag。

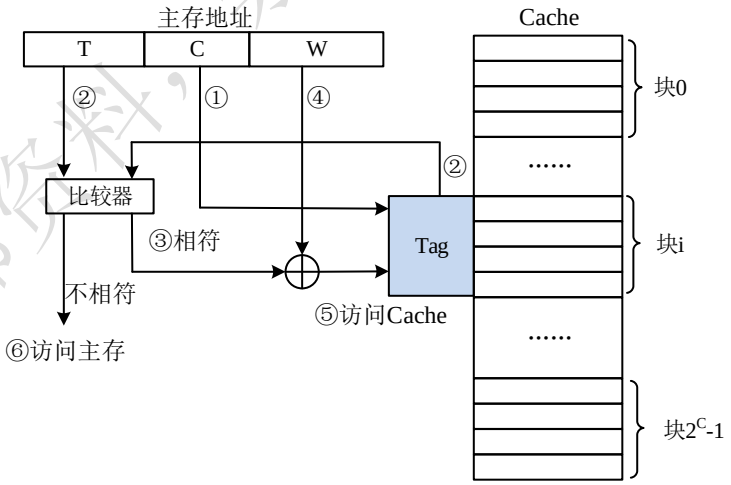


图 3.37 直接相联映射的地址变换规则

图 3.37 还给出主存地址到 Cache 地址变换的过程。当 CPU 需要访问主存数据的时候，地址线上给出的是拟访问数据的主存地址；①根据这个主存地址中的块号 C，查得 Cache 中块 C 的标记字段 Tag；②将 Tag 与主存地址的 T 进行比较[5]；③若相符表示该主存块已调入

[5] 图 3.37 所示地址变换过程中，仅需要一次比较操作。这是因为主存块号和 Cache 块号相同，可以从主存块号唯一确定 Cache 块号。故而直接相联映射方式下，Cache 块的标记 Tag 不需要采用相联存储器实现。

Cache（命中），④随后可使用块内地址 W 直接访问 Cache；若不相符，脱靶，则使用该地址直接访问主存。

直接相联映射方式的优点是：地址映射方式简单，检查页号是否相等仅需简单硬件电路，可以得获得较快的访问速度。其缺点在于：替换操作频繁。主存每个块在 Cache 中只有一个对应位置，若另一页中相同块号的块也要调入该位置，就会发生冲突，导致命中率低。

3. 组相联映射方式

组相联映射方式是直接映射和全相联映射的折中，能较好兼顾前两种方式优点。映射规则如下。

(1）将主存和 Cache 划分成大小相等的块，Cache 总块数为 2^C，主存总块数为 2^M。

(2）将 Cache 划分成大小相等的组，将总块数为 2^C 的 Cache 分成 2^u 组，每组 2^v 块。

(3）主存容量是 Cache 容量的整数倍，将总块数为 2^M 的主存划分为 2^s 页，每页 2^u 块，即主存每页的大小与 Cache 的组数相等。

(4）主存数据调入时，主存块号与 Cache 组号应相等，但组内各块地址之间则可以任意存放。即从主存的块到 Cache 的组之间采用直接映射方式，而与组内的各块则是全相联映射。

图 3.38 示出了组相联的映射关系，图中 Cache 共分 2^u 个组，每组包含有 2^v 块；主存是 Cache 的 2^s 倍，故分成 2^s 个页，每个页有 2^u 个组，每组有 2^v 个块。

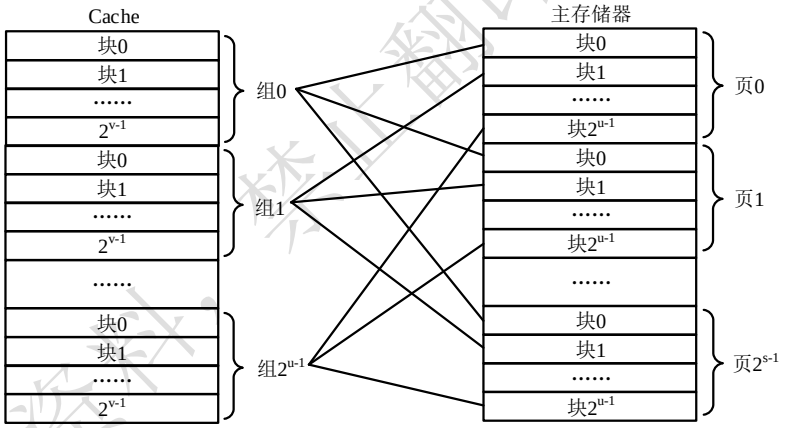


图 3.38 组相联的映射关系

主存地址格式中应包含三个字段：页号 s、页内块号 u、块内地址 W。缓存中包含三个字段：组号 u、组内块号 v 和块内地址 W。主存地址与缓存地址的转换有两部分，组地址是直接映射方式，按地址进行访问，而块地址是采用全相联方式，按内容访问。组相联的块表存储主存地址中页号 s 和 Cache 地址中组内块号 v 的映射信息，如图 3.39 所示。

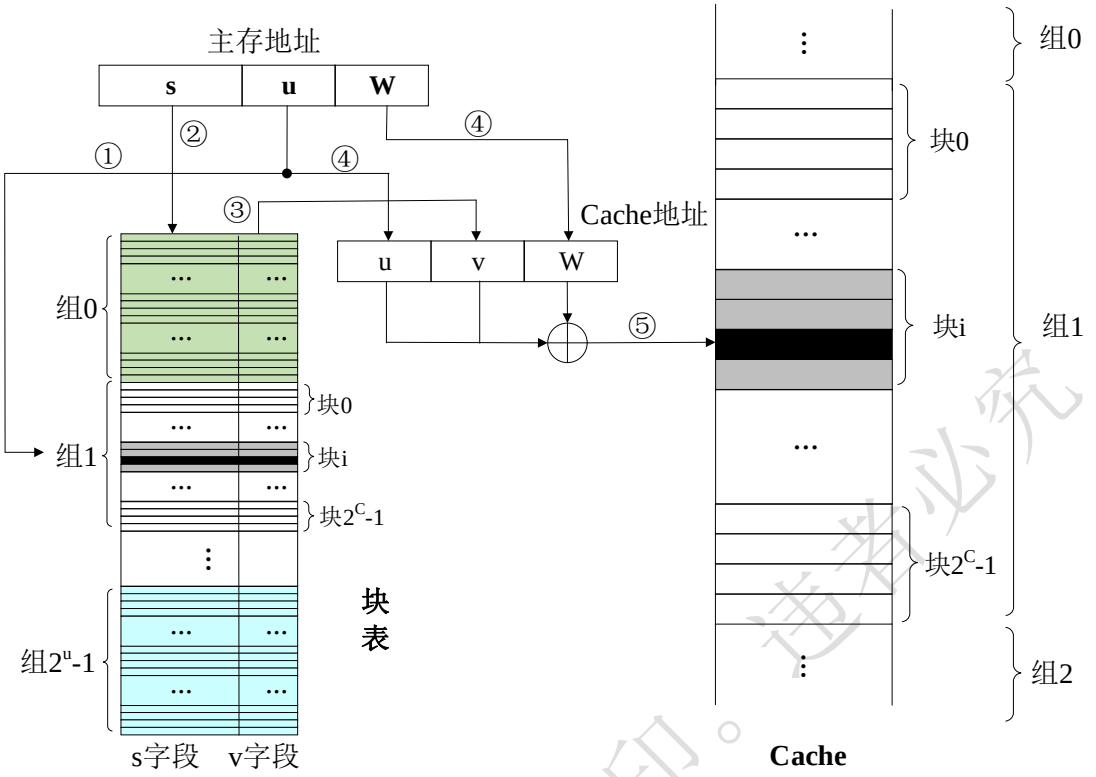


图 3.39 组相联的地址转换

当主存块被调入 Cache 时，将其地址的前 s 位写入块表的 s 字段（此即 Cache line 的 Tag，2^s 页对应 Tag 为 s 位）。如图 3.39 所示，CPU 访问存储器时，①首先根据主存地址中的 u 字段，找到块表对应的组，②然后将该组的所有项的前 s 位，与主存的 s 字段比较。③若相符，则表明主存块在 Cache 中，则将该项中的 v 字段取出，作为 Cache 地址的 v 字段。④Cache 地址的 u、W 字段由主存地址同样字段形成，这样⑤构成了完整的访问 Cache 地址。若不相符，为未命中，则直接用主存地址访问主存。

由于组内各 Cache 块采用全相联映射方式，上述过程中查询主存 s 字段会涉及多次比较操作，故块表需要使用相联存储器。在实现块表时，可以直接把 Cache 块所对应的 s 字段作为该 Cache 块的 Tag（Tag 采用相联存储器实现）。对比全相联映射、直接相联映射和组相联映射三种不同方式，主存地址与 Cache 块标记 Tag 之间的关系如图 3.40 所示。图 3.40 中，直接相联映射方式下，主存块号 C 即对应 Cache 块号；组相联映射方式下，页内块号 u 即对应到 Cache 的组号。

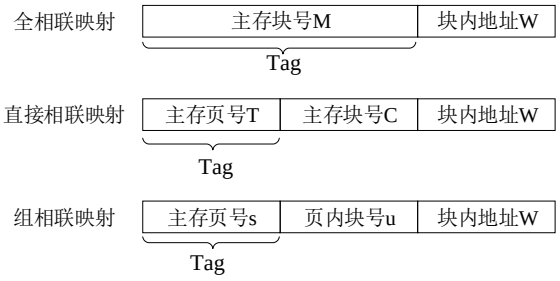


图 3.40 主存地址与 Cache 标记的关系

组相联映射方式中块的冲突概率比较低，块的利用率较高，块失效率明显降低。其硬件实现难度比直接映射方式高，但低于全相联方式。组相联映射方式是现代计算机中最常用的 Cache 地址映射方式。

实际上，全相联映射方式和直接相联映射方式分别对应组相联映射方式的两种极端情况。若 u＝0，v＝C，即全相联映射方式；若 u＝C，v＝0，即直接相联映射方式。应用时，组相联映射方式每组的块数一般取值较小，如 4 核心的英特尔第 4 代酷睿处理器 i7-4790K 中，一级缓存每组块数为 8，称作 8 路组相联（8-way）、二级缓存每组块数为 16，称作 16 路组相联（16-way）。再如，组合了 8 个性能核心（P-core）与 4 个能效核心（E-core）的英特尔第 12 代酷睿处理器 i7-12700K 中，性能核心的一级缓存为 12 路组相联（12-way）、独享二级缓存为 10 路组相联（10-way），效能核心的一级缓存为 12 路组相联（8-way）、4 核共享二级缓存为 16 路组相联（16-way）。

3.5.3 Cache 更新与替换策略

Cache 中的内容在不断更新，但总是主存内容的一部分，必须保持主存和 Cache 内容的一致性，这就涉及如何在 CPU、Cache 和主存间存取数据的问题。为此专门设计了两种读取结构，即贯穿读出式和旁路读出式读取结构；同时设计了两种写入策略，用来实现 Cache 的更新，即写通法和写回法。

一旦出现未命中的情况，或者发现 Cache 满了，替换控制部件就会按一定的规则，进行数据块的替换，扔掉一部分旧数据块，换进新数据块，实现 Cache 的更新。确定替换的规则也叫替换策略或替换算法，常用的替换算法有近期最少使用法、最不经常使用法和随机法等。

1. 读取结构

读取结构由硬件组成。当读取的数据在 Cache 中时，直接访问 Cache；否则，访问主存，同时将数据（所在的主存数据块）装入 Cache，并设置 Cache 块有效位字段。。

1) 贯穿读出（Look Through）

该方式将 Cache 隔在 CPU 与主存之间，CPU 对主存的所有数据请求都首先送到 Cache，由 Cache 在其中查找。如果命中，则切断 CPU 对主存的请求，并将数据送出；不命中，则将数据请求传给主存。该方法的优点是降低了 CPU 对主存的请求次数，缺点是延迟了 CPU 对主存的访问时间。

2) 旁路读出（Look Aside）

该方式下，CPU 是同时向 Cache 和主存发出数据请求。由于 Cache 速度更快，如果命中，则 Cache 在将数据回送给 CPU 的同时，还来得及中断 CPU 对主存的请求；不命中，则 Cache 不做任何动作，由 CPU 直接访问主存。该方法的优点是没有时间延迟，缺点是每次 CPU 都存在主存访问，从而占用一部分总线时间。

2. 写入更新策略

Cache 是主存中一部分内容的副本，需要保持与主存内容的一致。当 CPU 对 Cache 数

据做了修改后，应同时对主存相应位置内容进行修改。此时通过写入更新策略，来确保一致性。

1）写通方式（Write Through）

也称直写或者贯通方式。任一从 CPU 发出的写信号送到 Cache 的同时，也写入主存，以保证主存的数据能同步地更新。此方法的优点是操作简单，较好地保持了 Cache 与主存内容的一致性，可靠性高。但这种写的速度就是主存写的速度，由于主存的慢速，降低了系统的写速度并占用了总线的时间，没有发挥 Cache 高速访问的优势。

2）写回方式（Write Back）

该方式下，更新数据只写到 Cache。这样有可能出现 Cache 中的数据得到更新而主存中的数据不变的情况。可在 Cache 中设置一致性控制位（“脏”位），Cache 中有被修改的数据时，该控制位置“1”。每次 Cache 有数据更新时，判断该标志位。只有该标志位为 1，即 Cache 中的数据被再次更改而需要换出时，才将原更新的数据写入主存相应的单元中，然后再接受再次更新的数据。这样保证了 Cache 和主存中的数据不致产生冲突。这种方式克服写通方式每次数据写入时都要访问主存从而导致系统写速度降低并占用总线时间的弊病，减少了对主存的访问次数。但有 Cache 与主存数据不一致的隐患，控制也较复杂。

3. 替换策略

从主存向 Cache 传送一个新块，而 Cache 中可用位置已被占用时，就产生了 Cache 替换问题。替换问题与 Cache 的组织方式紧密相关：直接映射方式下，只要把可用位置上的主存块换出 Cache 即可；而全相联和组相联方式下，需要从若干个可用位置中选取一个位置，把其中的主存块换出 Cache。替换算法用硬件电路实现，常用的有四种策略。

1）随机（Random）替换策略

随机替换是最简单的替换算法。随机替换算法完全不管 Cache 的情况，随机选择一块替换出去。随机替换算法在硬件上容易实现，速度快。但缺点也很明显，被换出的数据可能马上就需要再次使用，增加了映射装入的次数，降低了命中率和 Cache 效率。

2）最不经常使用（Least Frequently Used，LFU）替换策略

将一段时间内被访问次数最少的块替换出去。给每块设置一个计数器（即为每个 Cache 块增加替换控制位字段，根据算法不同，可以是一位或多位），从 0 开始计数，每访问一次计数器就增 1。当需要替换时，将计数值最小的块换出，同时将所有块的计数器清零。算法的计数周期是两次替换之间的间隔时间，不能准确反映近期访问情况，新调入的块很容易被替换出去。

3）先进先出（FIFO）替换策略

根据进入 Cache 的先后次序来替换，先调入的 Cache 块被首先替换掉。这种策略不需要随时记录各个块的使用情况，容易实现，且系统开销小。其缺点是一些需要经常使用的程序块可能会被调入的新块替换掉。

4） 近期最少使用（Least Recently Used，LRU）替换策略

将 CPU 近期使用次数最少的块替换出去。它需要随时记录 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。具体方法为，给每个块设置一个“未访问次数计数器”（即为每个 Cache 块增加替换控制位字段）。每次 Cache 命中时，命中块的计数器清 0，其它块的计数器加 1。每当有新块调入时，将计数值最大的块替换出去。确保新加入的块保留，还可把频繁调用后不再需要的数据淘汰掉，提高 Cache 利用率和命中率。这种替换算法相对合理，命中率最高，硬件实现也并不困难，是最常采用的方法。

3.6 虚拟存储器

虚拟存储器（Virtual Memory，VM）是一种计算机系统内存管理技术。其效果是让程序员在编程时面向一个连续可用的地址空间，而程序运行时，可以被划分为多个分片，按需将一部分分片调入物理内存（Physical Memory），其余部分暂时存储在辅助外存（磁盘）。

虚拟存储器可视为操作系统提供的一种抽象主存，其实现由操作系统软件和底层硬件配合完成，典型技术如 Windows 的“虚拟内存”以及 Linux 的“交换空间”。通常由操作系统决定哪些程序与数据应被调入（或调出）物理内存；底层硬件配合完成外存与内存间数据交换过程的地址空间转换。本节首先介绍虚拟存储器的有关概念，然后介绍较为常见的段式、页式、段页式等三种虚拟存储器的工作原理。

3.6.1 虚拟存储器概述

1. 虚地址与虚存空间

在 3.4.1 小节中，已讨论了存储器的地址和地址空间。CPU 通过地址总线发出的地址可对应到物理内存的特定存储单元，此地址也称作物理地址（Physical Address，PA）。早期的计算机系统中，程序员编程时直接使用物理地址。但这种方式存在一些固有的缺陷。首先，若主存储器容量太小（早期计算机内存只有几十至数百 KB），不足以载入一个完整程序，程序运行时需要分块多次载入内存，控制繁琐。其次，若程序访问了一个错误的地址（如操作系统核心代码所在位置），易导致系统崩溃。第三，在多任务并发执行时，不同任务对应的多个程序可能会访问内存的同一区域，从而导致冲突。

虚拟存储器就是在这样的背景下逐步发展起来的一种内存管理技术。用户编程时使用的地址（由编译程序生成）称为虚地址（Virtual Address），对应的存储空间称为虚存空间。通常虚存空间的大小就是 CPU 可寻址的最大范围。有了虚拟存储器概念后，程序设计可使用整个虚存空间，但程序运行时到底载入物理内存中的哪一部分，是在程序运行时才动态分配。图 3.41 所示为两个程序载入内存时虚拟存储器空间到物理存储器空间的映射的示例，各程序使用的虚拟地址由专门的转换部件转换为实际内存的物理地址。图中程序 1 使用整个虚拟存储器空间 0x0000~0xFFFF，其代码占用 0x0000~0x0300 范围，在映射到物理内存时，被映射到 0x400~0x5FF 和 0xA00~0xAFF 两处不连续的区域；程序 2 在虚拟存储器空间中占用 0x0000~0x02FF 范围，被映射到物理内存 0x100~0x3FF 区域。

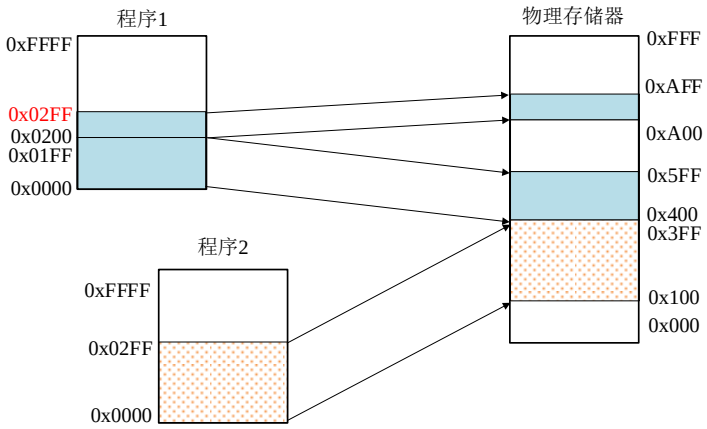


图 3.41 虚拟存储器到物理存储器的映射及地址转换

2. 虚拟存储器对主存的“扩充”

虚拟存储器是存储器抽象模型，其容量可以比实际主存空间大。此时，可以在操作系统的管理下，选择性地把虚拟存储器中的部分区域映射到实际的物理存储器；物理存储器无法容纳的其他部分可仍存放在磁盘等辅助存储器上。如图 3.42 所示，这种机制可以借助磁盘来“扩充”主存容量，以服务更大或更多的程序。当一个较大的程序运行时，可能只会用到其一小部分代码和数据，可以仅把这部分放到较快速的物理内存中，其他大部分仍存放在大容量、低速的外存（硬盘）。同理，多个程序并行执行的时候，可以把每个程序用到的部分调入物理内存，其余部分留在外存中。从而有助于提升物理内存的利用率和计算机整体运行效率。

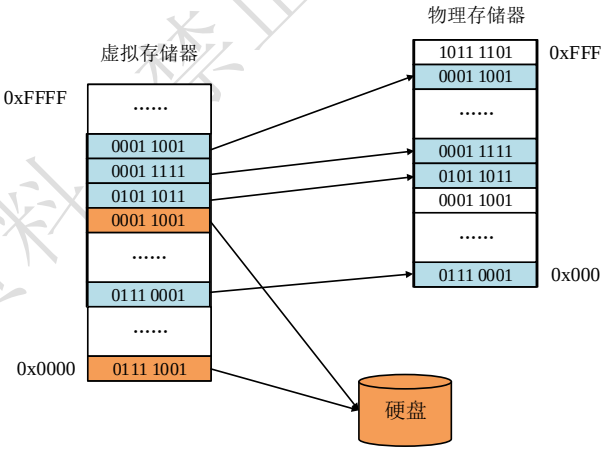


图 3.42 虚拟存储器借助外存扩充内存示意图

3. 虚拟存储器和 Cache 的异同

虚拟存储器和 Cache 有很多相似之处。（1）二者均把程序分成较小的信息块、运行时把信息块从慢速存储器调入快速存储器。快速存储器中信息块的更新都需要按预定义的策略进行，以提高命中率。（2）使用信息块时都需要按照映射关系进行地址变换。CPU 访问 Cache 时要将主存地址转换为 Cache 地址，外存信息调入内存时需要将虚拟地址转换为物理地址。（3）依据程序局部性原理，二者均有利于提升计算机整体性能。Cache 可以缓解 CPU 与内存访问速度的矛盾，虚拟存储器则可以形成内存容量的补充，并尽可能减少慢速辅存的影响。

虚拟存储器和 Cache 的区别也十分显著。（1）Cache 使用定长的信息块，通常是几十字

节；虚拟存储器所划分信息块既可以是定长的，也可以不定长，长度可达数百 KB。（2）通常 Cache 的速度是内存的 5~10 倍，而内存（访问时间纳秒级别）的速度是外存（访问时间毫秒级别）的 100~1000 倍，因而虚拟存储器未命中的性能损失要远大于 Cache 未命中情形。（3）CPU 与 Cache 和主存之间均有直接访问通路，而虚存所依赖的辅存与 CPU 之间不存在直接的数据通路。（4）Cache 管理由硬件实现，对各类程序员均透明，即程序员感知不到 Cache 的存在。而虚存管理由操作系统软件和硬件共同完成，虚存对实现存储管理的程序员是不透明的，而普通的应用程序而言，虚存提供了一个庞大的逻辑空间可自由使用，故对应用程序员来说又是透明的。

3.6.2 段式虚拟存储器

将外存中的信息块调入物理内存的时候，划分信息块有多种不同的方式。段式存储管理把物理内存按段分配，内存（主存）与辅存间传送不定长的段。段是按照程序的自然分界划分出的长度可变的区域，如代码段、数据段、堆栈段等。段作为独立的逻辑单位可以被其他程序段调用，这样可形成段间连接，产生规模较大的程序。

段式存储管理方式中，程序的地址空间被划分成若干个段，每一个段包含一组完整的信息，如主程序段、子程序段、数据段及堆栈段等。每一个段都有各自的名字，都是从地址 0x0 开始编址的一段连续的地址空间，各段长度不等，如图 3.43 所示。

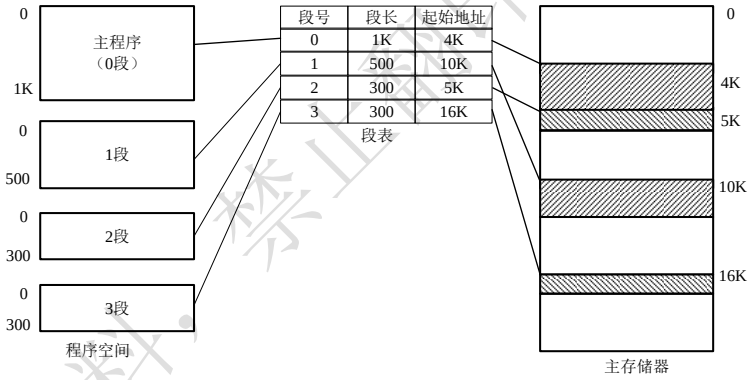


图 3.43 段式存储的地址映射

虚地址由段号 S 和段内偏移地址 D 组成。如图 3.44 所示，虚地址和物理地址之间的映射关系记录在段表中，由段表记录每个段的段号、段长、起始地址。段表存放于内存中，CPU 通过段表基址寄存器获取段表的起始地址，依据段号 S 在段表中查询该段的对应信息，段信息中的段起始地址与段内偏移地址 D 相加得到虚地址在物理内存中的实际地址。

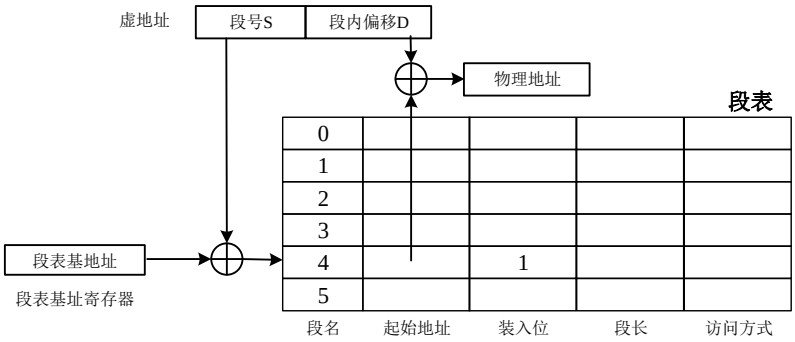


图 3.44 段式虚拟存储的物理地址变换

除了段名、段起始地址、段长外，图 3.44 所示段表还包含装入位、访问方式信息。装入位提示该段是否已经载入内存，访问方式信息指示该程序段的访问方式（只读、可读可写、只能执行等）。由于程序的不同段可以设置各自的安全属性，如只读的代码段、可读写的数据段，因而段式存储管理易于实施存储器保护和多程序的共享。但缺点是内存载入不同段时，容易在段间形成空余的存储空间碎片。若某个段非常大，从外存载入内存（或从内存写回外存）时较耗时，易导致机器卡顿。

3.6.3 页式虚拟存储器

页式存储管理是一种把主存按页分配的存储管理方式，主存与辅存间信息传送的是定长的页。程序使用的虚存地址空间划分成大小相等的区域，称为页（Page）。对应地，将主存空间划分成与页同样大小的若干个物理块，称为页框（Page Frame）。在为程序分配主存时，可以将程序的若干页分别装入多个物理块（相邻或不相邻的页框）中。如图 3.45 所示。程序运行时，当 CPU 需要读取特定页，但却发现该页内容未加载时，会触发一个缺页错误，操作系统捕获这个错误，然后找到对应的页并加载到内存中。

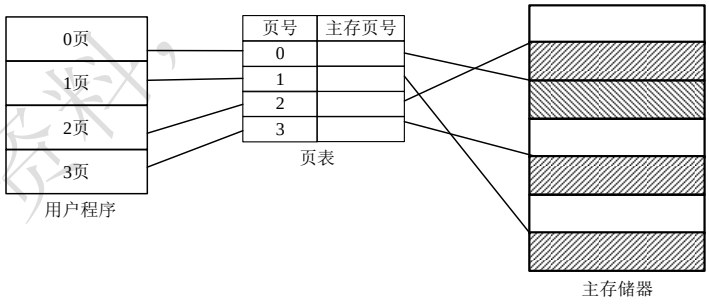


图 3.45 页式存储的地址映射

虚地址由虚页号 P 和页内偏移地址 D 组成。如图 3.46 所示，虚地址和物理地址之间的映射关系记录在页表中，页表记录每个页的页号，以及是否装入内存、是否在内存中被修改等各种标识信息。页表存放于内存中，CPU 通过页表基址寄存器获取页表的起始地址，依据虚页号 P 在页表中查询该页在内存中的实际页（框）号，由实际页（框）号、页内偏移地址 D 计算得到该虚地址在物理内存中的实际地址。由于页的大小是固定的，所以虚地址到实际物理地址的转换比段式管理情形要简单很多。

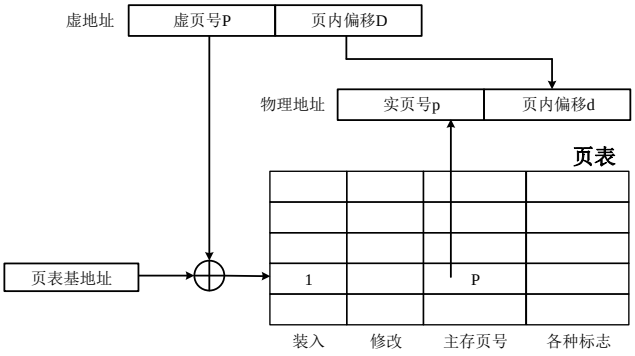


图 3.46 页式虚拟存储的物理地址变换

相比于段式管理，页式存储管理粒度更细致，故内存页碎片的浪费会小很多。同时因为页要比段小得多（Linux 下默认为 4KB），所以页在进行交换时，不会出现段交换那般卡顿。所以，多数操作系统采用页式存储管理。页式管理的缺点在于，页不是程序模块的自然对应，页的保护和共享都不如段方便。

注意，页表存放在主存中，因而即便是逻辑页已在主存中，也要访问两次物理存储器才能实现一次存储器读写操作，这将使虚拟存储器的存取时间加倍。为减少主存访问次数的增多，现代计算机中，对页表本身进行缓存，把页表中的最活跃的部分存放在专用 Cache 中。这个用于页表缓存的专用 Cache 被称为 TLB（Translation Lookaside Buffer），又称快表，与之对应，主存中的完整页表则称为慢表。TLB 通常由相联存储器实现，可进行硬件高速检索。

实际应用中，还需要考虑页表的大小。以 32 位逻辑地址空间、页面大小 4KB、页表项大小 4B 为例。如果要实现完整逻辑地址空间的映射，则需要 2^{20}（约 100 万）个页表项。即，每个程序仅页表这一项就需要 4MB 主存空间，代价高昂。故而常采用二级页表来降低页表存储量要求，关于二级页表，以及相关的页表置换算法、页面分配策略等详情可参阅操作系统相关资料。

3.6.4 段页式虚拟存储器

段页式虚拟存储器，先将程序按自然逻辑关系分为若干个段，再把每一个段分成若干页。主存与辅存间的信息传送以页为单位，如图 3.47 所示。段页式存储管理结合了段式和页式两种管理方法的优点，提高了主存利用率，也可以按段实现共享和保护。

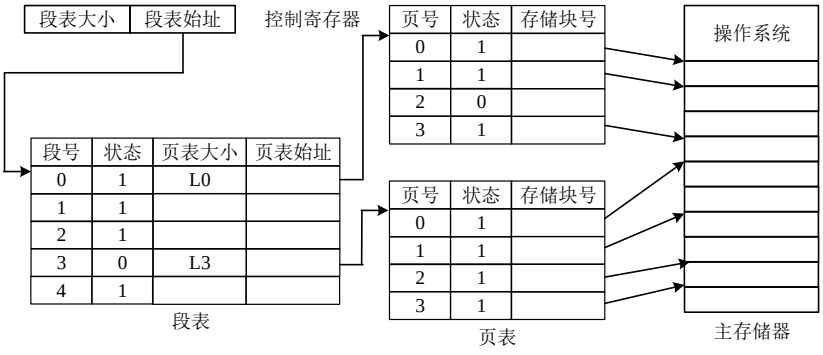


图 3.47 段页式存储地址映射

虚地址由段号 S、虚页号 P、页内偏移 D 组成。如图 3.48 所示，段表中记录页表长度和页表地址。通过段表基地址和段号 S 相加，可得到页表起始地址和页表长度，进而通过页表找到物理内存中的实际页号，实际页号与页内偏移 D 拼接得到虚地址对应的物理地址。

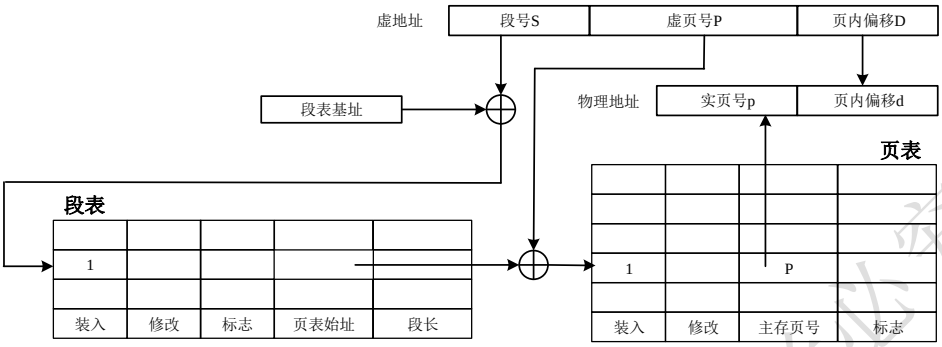


图 3.48 段页式虚拟存储的物理地址变换

3.7 习题

3.1 请说明存储器的类型及其特点。

3.2 半导体存储器有几种？分别具有的优点、缺点以及使用场合？

3.3 RAM 有几种工作原理？ROM 的类型和工作方式？

3.4 磁介质存储器的类型和应用？光存储器的主要类型和工作方式？

3.5 什么是主存储器？什么是外存储器？

3.6 辅助存储器常见的几种接口标准？

3.7 选择存储器主要考虑哪些因素？

3.8 EPROM、EEPROM、Flash 的共同特点是什么？这些存储器具有读写功能，为什么仍然称作 ROM？

3.9 计算机系统为什么要采用内存条？

3.10 从 DDR 到 DDR4 有哪些改进？

3.11 存储器系统的层次架构有何意义？

3.12 虚拟存储器解决了哪些问题？

3.13 虚拟地址与物理地址有何区别和联系？

3.14 页式虚拟存储器的页面大小对操作速度产生什么影响

3.15 虚拟存储器有几种管理方式？其地址转换是怎样实现的？

3.16 Cache 的工作原理、作用是什么？在哪些场合使用 Cache？

3.17 为什么要保证 Cache 内容与主存内容的一致性？有哪些方法？

3.18 Cache line 应该含有那些基本信息项？

3.19 Cache 常用的替换策略有哪些？

3.20 某计算机按字节编址，其主存容量为 1MB，Cache 容量为 16KB，Cache 和主存之间交换的块大小为 64B，采用直接相联映射方式。（1）Cache 共有多少个字块（Cache line）？（2）

主存地址为 02021H 的单元装入 Cache 后对应的 Cache 地址是？（3）主存地址为 02021H 的单元装入 Cache 后存放在 Cache 中的第几字块中（Cache 起始字块为第 0 字块）？

3.21 某计算机按字节编址，其主存容量为 1MB，Cache 容量为 16KB，Cache 和主存之间交换的块大小为 64B，采用 8 路组相联映射方式。（1）主存地址中页号 s、页内块号 u、块内地址 W 各占多少位？（2）主存地址为 02021H 的单元装入 Cache 后存放在 Cache 中的第几组（起始组为第 0 组）？（3）Cache line 对应的 Tag 字段占用多少位？

3.22 某按字节编址的计算机系统，使用了 40 位地址线，16 位数据线，请问其存储器容量为多少字节？

3.23 试用 1M×1 位的芯片构成 1M×8 位的存储器。

3.24 试用 4K×8 位的芯片构成 4K×16 位的存储器。

3.25 设计一个采用 64K×8 位构成的 256K×8 位的存储器系统。

3.26 试用 8 片 1K×8 位的芯片构成 8K×8 位的存储器。

3.27 设计一个用 4 片 4K×8 位的芯片构成 8K×16 位的存储器。

3.28 NOR 与 NAND Flash 在嵌入式系统中如何连接？功能有何异同？

3.29 ARM 微处理器如何扩展 RAM？