第3章 存储器

随着超大规模集成电路工艺的发展，CPU 的速度变得越来越快，而存储器速度的提高则十分缓慢，这使计算机整体的运行速度在很大程度上受到了制约，因此提高存储器的速度变得尤为重要。

为了更好理解本章内容，首先对本章将使用到的名词加以解释。

（1）存储器：存放程序和各种数据的器件。

（2）存储位：存放一位二进制代码，是存储器最小的存储单位，也称其为存储元件、存储基元或存储元。

（3）存储单元：每个存储单元包含若干个存储位，即每个存储单元能够存储一串二进制代码。

（4）存储字：存储单元中存储的一串二进制代码称存储字。

（5）存储字长：存储单元中存储的一串二进制代码的位数称为存储字长，它可以是 8 位、16 位或 32 位等。

（6）存储体：大量存储单元的集合组成存储体。

（7）存储单元地址：存储单元的编号。

（8）字编址：对存储单元按字编址。

（9）字节编址：对存储单元按字节编址。

（10）寻址：由地址寻找数据，从对应地址的存储单元中访问数据。

3.1 存储器概述

存储器是计算机的记忆部件，它采用具有两种稳定状态（分别对应二进制的 0 和 1）的物理元器件来存储信息（通常为程序和数据）。

3.1.1 存储器分类

存储器的分类方式很多，本小节将根据存储器在计算机中的作用、存取方式、存储介质和信息的可保存性来介绍其分类。

1. 根据在计算机中的作用分类

根据在计算机中的作用不同，存储器可分为主存储器（简称主存，也称内存）、辅助存储器（简称辅存，也称外部存储器或简称外存）和缓冲存储器（简称缓存或 Cache）等。

1）主存

主存最为重要的特点是可以和 CPU 直接交换信息，它用于存放计算机运行期间需要的程序和数据，主存的速度快、容量小、每位价格高。

2）辅存

辅存用来存放暂时不用的程序和数据，它不能与 CPU 直接交换信息，辅存的速度慢、容量大、每位价格低。

3）缓存

缓存用在两个速度不匹配的部件之间，如 CPU 与主存之间可设置缓存，起到缓冲作用。

2. 根据存取方式分类

根据存取方式的不同，存储器可分为随机存储器、只读存储器和串行访问存储器（又称顺序存取存储器）。

1）随机存储器

随机存储器（Random Access Memory，RAM）是一种可读写存储器，其特点是存储器中任何一个存储单元的内容都可以随机存取，而且存取时间与存储单元的物理位置无关。计算机系统中的主存都采用这种随机存储器。基于存储信息原理的不同，RAM 又分为静态 RAM（Static RAM，SRAM，利用触发器原理来保存信息）和动态 RAM（Dynamic RAM，DRAM，利用电容充放电原理保存信息）。

2）只读存储器

只读存储器（Read Only Memory，ROM）中的内容固定，一般仅对其进行读取操作。这种存储器一旦存入了原始信息后，在程序执行过程中，只能将内部信息读出，而不能随意重新写入新的信息去改变或替换原始信息。因此，通常用它存放固定不变的程序、常数以及汉字的字库等，也可用于操作系统的固化。它与随机存储器可共同作为主存的一部分，统一构成主存的地址域。

早期只读存储器的内容根据用户的需求，采用掩模工艺把原始信息记录在芯片中，一旦制成后无法更改，称为掩模型只读存储器（Masked ROM，MROM）。随着半导体技术的发展和用户需求的变化，只读存储器先后派生出可编程只读存储器（Programmable ROM，PROM）、可擦除可编程只读存储器（Erasable Programmable ROM，EPROM）以及用电可擦除可编程的只读存储器（Electrically Erasable Programmable ROM，EEPROM 或 E^2PROM）。近年来还出现了闪速存储器（Flash Memory，又称快擦型存储器或快闪存储器，简称闪存），它具有 EEPROM 的特点，但速度比 EEPROM 快得多。

3）串行访问存储器

如果对存储单元进行读写操作时，需按其物理位置的先后顺序寻找地址，则称其为串行访问存储器。当信息所在存储器中的位置不同时，对应的读写时间通常也不同。例如磁带，不论信息处在哪个位置，读写时必须从其始端开始顺序往后寻找，因此这类串行访问的存储器也称为顺序存取存储器。还有一种属于部分串行访问的存储器，如磁盘。在对磁盘进行读写时，首先直接指出该存储器中的某个小区域（磁道），即随机访问，然后在该磁道上顺序访问，直至找到位置，即串行访问，这种存储器也称为半顺序存取存储器。

3. 根据存储介质分类

存储介质是存储数据的载体，它必须能够显示两种（通常为二进制的 0 和 1）有明显区别的物理状态，且这两种物理状态的变换速度决定着存储器的存取速度。目前，存储介质主要有半导体器件、磁性材料和光盘等，相应的存储器有半导体存储器、磁表面存储器、光盘存储器。

1）半导体存储器

半导体存储器是指由半导体器件组成的存储器。现代半导体存储器都用超大规模集成电路工艺制成芯片，其优点是体积小、功耗低、存取时间短；其缺点是当电源消失时，所存储的信息也随即丢失，它是一种易失性存储器。近年来已研制出用非挥发性材料制成的半导体存储器，克服了信息易失的缺点。

半导体存储器又可按其材料的不同，分为 TTL（Transistor-Transistor Logic，晶体管-晶体管逻辑电路）型半导体存储器和 MOS（Metal Oxide Semiconductor，金属氧化物半导体）型半导体存储器两种。前者具有高速的特点，而后者具有高集成度的特点，并且制造简单、成本低廉、功耗小，因此 MOS 型半导体存储器被广泛应用。

2）磁表面存储器

磁表面存储器是在金属或塑料基体的表面上涂一层磁性材料作为记录介质，工作时磁层随载磁体高速运转，用磁头在磁层上进行读/写操作，故称为磁表面存储器。

磁表面存储器又可根据其磁体形状的不同，分为磁盘、磁带和磁鼓。现代计算机已很少采用磁鼓。由于用具有矩形磁滞回线特性的材料作磁表面物质，它们按其剩磁状态的不同而区分 0 或 1，而且剩磁状态不会轻易丢失，故这类存储器具有非易失性的特点。

3）光盘存储器

光盘存储器是应用激光在记录介质（磁光材料）上进行读写的存储器，具有非易失性的特点。此外，光盘记录具有密度高、耐用性好、可靠性高和可互换性强等优点。

4. 根据信息的可保存性分类

根据信息的可保存性（即断电后存储的信息是否消失），存储器可分为易失性存储器和非易失性存储器。

断电后存储的信息便消失的存储器称为易失性存储器，如 RAM。断电后存储的信息仍保持的存储器称为非易失性存储器，如 ROM。

结合上述分类方式，可用图 3-1 表示存储器的分类。

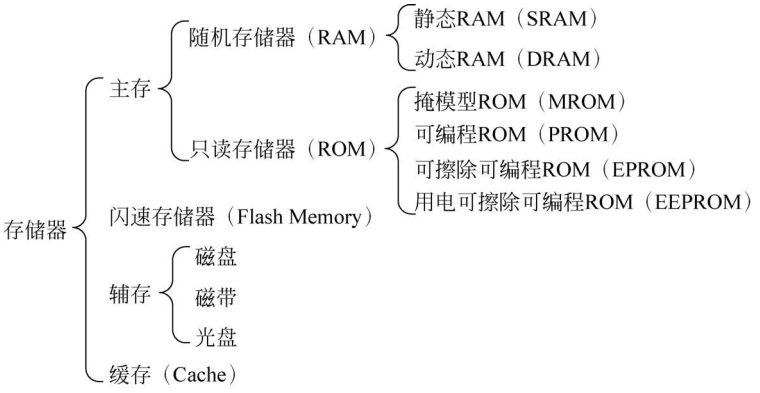


图 3-1　存储器的分类

3.1.2 存储器的主要技术指标

存储器的主要技术指标包括存储容量、存取时间、存储周期(又称存取周期)、存储器带宽以及存储器的可靠性等。

1. 存储容量

存储容量有两种表示方式,一种方式是用存储器能够存放的二进制代码的总位数来表示,即存储容量=存储单元个数×存储字长;另一种方式是用字节总数来表示,即存储容量=存储单元个数×存储字长/8(为了满足字符处理的需要,常用8位二进制数表示一个字节,即存储一个字节需要用8位的存储位)。目前,计算机的存储容量多以字节总数来表示。

2. 存取时间

存取时间是反映存储速度的指标之一。通常把启动一次存储器操作(读或写操作)到完成该操作所需的全部时间称为存储器访问时间(即存取时间),它决定了CPU进行一次读写操作必须等待的时间。

存取时间分为读出时间和写入时间,读出时间是指从存储器接收到有效地址开始,直至产生有效输出为止所需的全部时间;写入时间是指从存储器接收到有效地址开始,直至数据被写入选中的存储单元为止所需的全部时间。

注意:

(1) 通常把信息存入存储器指定位置的操作称为写操作,而把从存储器指定位置取出信息的操作称为读操作。

(2) 访问是指读操作、写操作或两者兼而有之。

3. 存储周期

存储周期(也称存取周期)是反映存储速度的指标之一,它指存储器连续进行两次独立的访问操作(连续两次读或写操作)所需的最小间隔时间。由于一次访问操作结束后,还需要花费一定的时间进行恢复处理(恢复时间)后才能进行下一次访问,因此通常存取周期大于存储时间,如图3-2所示。

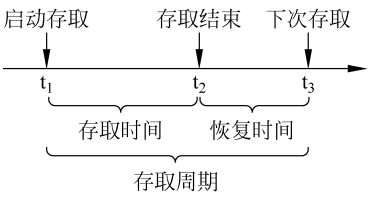


图3-2 存取时间和存取周期的关系

4. 存储器带宽

存储器带宽表示单位时间内存储器存取信息的数量,通常以字每秒、字节每秒(B/s)或位每秒(bit/s)为单位,它与存储周期密切相关。例如,存储周期为320ns(纳秒),每个存取周期可取32bit(位),则存储带宽为32bit/(320×10^{-9}s)=10M位每秒=10Mbit/s。

带宽是衡量数据传输率的重要技术指标,为了提高存储器的带宽,可以采取以下措施:

(1) 缩短存取周期。

(2) 增加存储字长,使得每个存储周期可以读写更多的二进制位数。

(3) 增加存储体(将在后续章节中介绍)。

5. 存储器的可靠性

存储器的可靠性用其平均故障间隔时间(Mean Time Between Failures,MTBF)来衡量。MTBF可以理解为两次故障之间平均时间的间隔,MTBF越长,表示可靠性越高,即保持正常工作的能力越强。

3.1.3 存储器层次结构

通常存储器的速度、容量和每位价格(简称位价)这 3 项指标大致反映了其性能。一般来说,速度越高,位价就越高;容量越大,位价就越低,且容量越大,速度就越低。实际应用时,总是希望存储器的容量尽可能地大,并且速度也尽可能地快,同时位价最好也尽可能地低,但在一个存储器中要求同时兼顾这 3 项指标是很困难的。为了解决这一矛盾,在现代计算机系统中设计存储器时,通常采用如图 3-3 所示的分级结构。

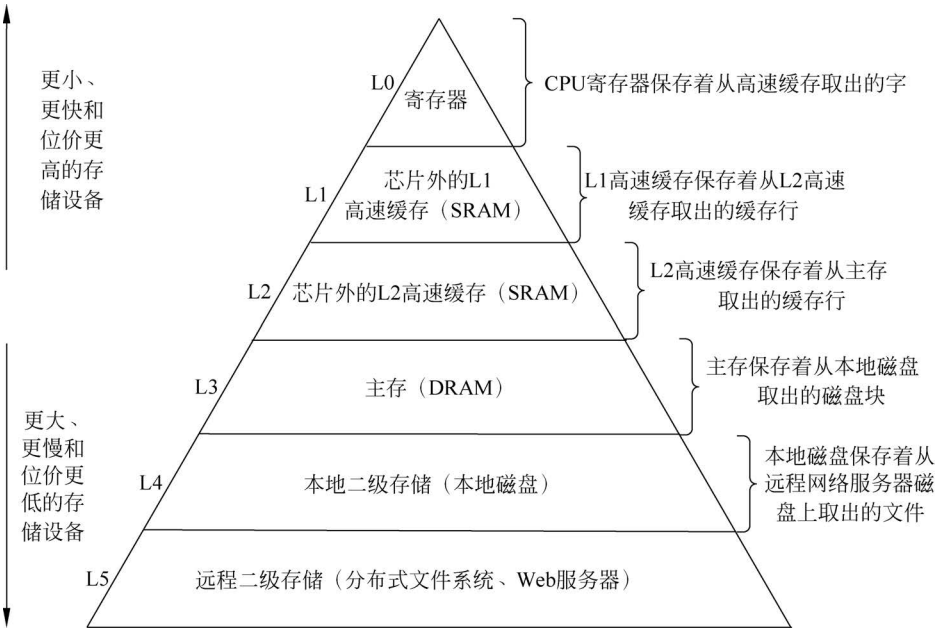


图 3-3 存储器分级结构

图 3-3 中由上至下,存储器的容量越来越大,而速度则越来越慢,位价越来越低,CPU 访问的频度也越来越少。

1. 寄存器

寄存器通常被内置在 CPU 芯片里,其速度最快、位价最高、容量最小,用于存放即将参与运算的数据。

2. 主存

和寄存器相比,主存的速度较慢、位价较低、容量较大,用于存放将要参与运行的程序和数据。

3. 高速缓存(也称为高速缓冲存储器)

由于主存的速度与 CPU 的速度相差甚远,因此,在主存和 CPU 之间设置了一种比主存速度更快,但容量相对较小的高速缓存(可设置一级缓存,也可设置多级)。

寄存器、主存和高速缓存三种存储器构成了缓存—主存的层次结构,通常它们都被设置在主机内,现代计算机也将高速缓存内置在 CPU 里。

4. 本地二级存储和远程二级存储

本地二级存储和远程二级存储均属于辅存,它们的容量比主存大得多,主要用来存放暂

时未用到的程序和数据。CPU 不能直接访问辅存，只能借助于主存间接访问辅存，它们一起构成了主存—辅存的层次结构。

综上所述，CPU 能和缓存、主存直接交换信息；缓存能和 CPU、主存直接交换信息；主存能和 CPU、缓存以及辅存直接交换信息。缓存—主存和主存—辅存这两个存储层次体现了存储器的分层结构，具体如图 3-4 所示。

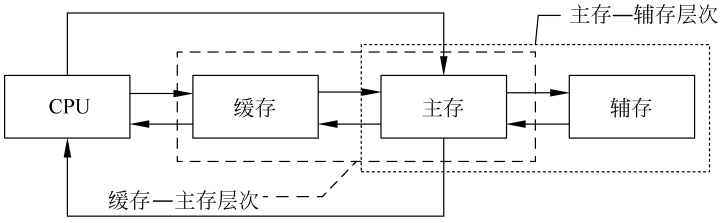


图 3-4　各类存储器间的关系

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，解决了 CPU 与主存速度不匹配的问题，且使容量和位价接近于主存。通过将 CPU 近期要使用的信息从主存调入缓存中，这样 CPU 便可以从速度较高的缓存中获取信息，从而提高访存速度（即 CPU 访问主存的速度）。但由于缓存容量还是较小，当 CPU 近期要使用的信息较多或产生变化时，存在一个替换的问题，这将在后续章节中介绍。

从整体来看，主存—辅存这一层次的速度接近于主存，容量和位价则接近于辅存，这解决了速度、容量、成本三者间的矛盾。辅存不能和 CPU 直接交换信息，但它的容量比主存大得多，可以用来存放大量暂时不使用的信息，当 CPU 要使用这些信息时，再将其从辅存调入主存中。主存—辅存这一层次经过不断发展，逐渐形成了虚拟存储系统，在后续章节中将对其进行介绍。

现代计算机系统几乎都具有缓存—主存和主存—辅存这两个存储层次，从而构成了缓存、主存、辅存三级存储系统。

3.2　主存储器

主存储器（Main Memory）是计算机的一个重要部件，CPU 能直接随机存取主存储器中的信息。

3.2.1　主存储器简介

从 20 世纪 70 年代起，主存储器已逐步采用大规模集成电路（目前用的最普遍、最经济的是 DRAM）构成，而在追求速度和可靠性的场合，则通常采用存取速度达到了 1～15ns 的 SRAM，但其价格较贵。

无论主存采用 DRAM 还是 SRAM 芯片构成，在断电时存储的信息都会“丢失”，因此计算机设计者应考虑发生这种情况时，设法维持若干毫秒的供电以保存主存中的重要信息，以便供电恢复时计算机能恢复正常运行。鉴于上述情况，在某些主存中，存储重要而相对固定的程序和数据时，采用“非易失性”的存储器芯片（如 EPROM，闪速存储芯片等）；对于完全固定的程序、数据区域则采用只读存储器（ROM）芯片，这样做不仅可以应对供电中断，而且

还可以防止病毒侵入。

主存是按存储单元的地址存放信息的，存取速度一般与地址无关。当需要访问某一信息时，需要给出其存储单元的地址，为了实现按地址访问，主存中还必须配置两个寄存器：一个是存储器地址寄存器(Memory Address Register，MAR)，用来存放待访问存储单元的地址，其位数对应存储单元的个数(例如，有 $2^{10}=1024$ 个存储单元，则 MAR 为 10 位)；另一个是存储器数据寄存器(Memory Data Register，MDR)，用来存放从某存储单元中取出来的信息或准备存入某存储单元的信息，其位数与存储字长相等。

实际上，根据 MAR 中的地址并不能够直接访问对应的存储单元，还需要经过地址译码电路对地址进行译码，再由相应的驱动电路进行驱动，才能找到所要访问的存储单元。为了将被选中单元中的存储字读出，需要经过读出电路，才能将其送到 MDR 中。向存储单元写入信息时，也必须经过写入电路，才能将内容写入到被选中的存储单元中。因此，主存的基本组成如图 3-5 所示。

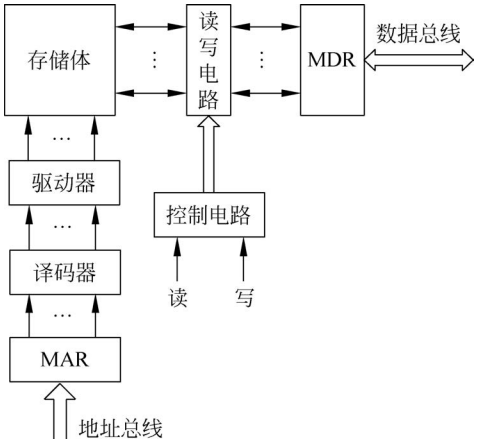


图 3-5　主存的基本组成

计算机系统既可以按字节寻址，也可以按字寻址，即按存储字长寻址，存储字长都取 8 的倍数。通常不同的机器，其存储字长一般是不同的，但均用 8 位二进制数表示一个字节。

【例 3-1】 已知 IBM370 机的存储字长为 32 位，从高位字节开始对存储单元进行编号，MAR 为 24 位，试分析其按字节寻址和按字寻址的范围。

解：由于 8 位二进制数表示一个字节，所以 IBM370 机的每个存储字包含 4(=32/8)个可独立寻址的字节，字地址用该字高位字节的地址来表示，故其字地址是 4 的整数倍。具体如图 3-6 所示。

(1) 按字节寻址。MAR 为 24 位，$2^{24}=2^4\text{M}=16\text{M}$，因此，按字节寻址的范围是 16M。

(2) 按字寻址。由于每个存储字包含 4 个可独立寻址的字节，16M/4=4M，因此，按字寻址的范围是 4M。

【例 3-2】 已知 PDP-11 机的存储字长为 16 位，从低位字节开始对存储单元进行编号，MAR 为 24 位，试分析其按字节寻址和按字寻址的范围。

解：由于 8 位二进制数表示一个字节，所以 PDP-11 机的每个存储字包含 2(=16/8)个可独立寻址的字节，字地址用该字低位字节的地址来表示，故其字地址是 2 的整数倍。具体如图 3-7 所示。

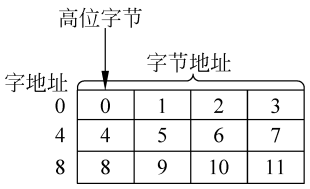


图 3-6　IBM370 的主存地址分配

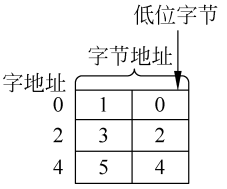


图 3-7　PDP-11 的主存地址分配

(1) 按字节寻址。MAR 为 24 位，$2^{24}=2^4$M=16M，因此，按字节寻址的范围是 16M。

(2) 按字寻址。由于每个存储字包含 2 个可独立寻址的字节，16M/2=8M，因此，按字寻址的范围是 8M。

3.2.2　随机存储器

随机存储器分为静态随机存储器(SRAM)和动态随机存储器(DRAM)，它们都是以半导体集成电路(半导体存储芯片)作为存储介质的存储器。接下来首先介绍半导体存储芯片。

1. 半导体存储芯片

1) 半导体存储芯片的基本结构

现代半导体存储器都是用超大规模集成电路工艺制成的芯片，在芯片内集成了存储矩阵、译码驱动电路和读写电路等，其基本结构如图 3-8 所示。

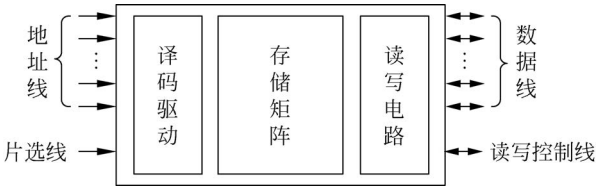


图 3-8　半导体存储芯片的基本结构

译码驱动将接收到的地址信号译码为对应的存储单元的选择信号，然后由读写电路确定对被选中的单元执行何种操作。

存储芯片通过地址线、数据线和控制总线与外部相连。地址线是单向输入的，其位数与存储单元的个数有关；数据线是双向的(既可以输入，也可以输出，有些芯片也用成对的数据线分别作为输入和输出)，其位数与一次可读出或写入的数据位数有关。

地址线和数据线各自的位数一起反映了存储芯片的容量。例如，地址线有 10 根，数据线有 8 根，则存储芯片的容量为 $2^{10}\times8$ 位=8K 位。

控制总线主要有片选线和读写控制线，不同存储芯片的片选线和读写控制线的数目可以不同。有的存储芯片的片选线用 1 根(如 2114)，有的用 2 根(如 6264)；有的存储芯片的读写控制线共用 1 根(如 2114)，有的分用 2 根(如 6264)。

半导体存储器一般是由多个芯片组成的，因此在执行读写操作时，需要使用片选线来选择存储芯片。例如，组成 32K×8 位的存储器需要 16K×1 位的存储芯片 16 片(8 位/1 位=8，32K/16K=2，8×2=16 片)，如图 3-9 所示，片选线选中 8 片存储芯片。读写控制线用来决定对存储芯片进行读操作还是写操作。

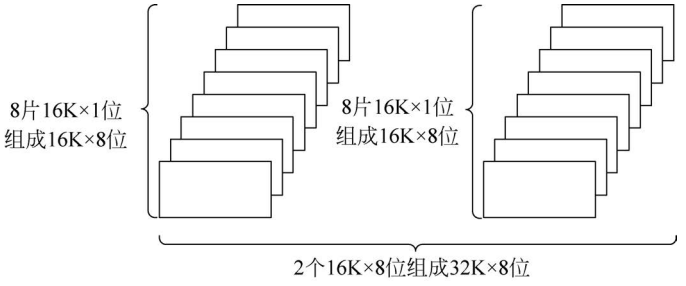


图 3-9　32K×8 位的存储器

2）半导体存储芯片的译码驱动方式

半导体存储芯片的译码驱动方式分为线选法和重合法。

（1）线选法。线选法是使用一根字选择线（简称字线）选中一个存储单元的各位。如图 3-10 所示为 16×8 位（$16=2^4$ 故地址线为 4 根，分别为 A_0、A_1、A_2 和 A_3；8 位故数据线为 8 根，分别为 D_0、D_1……D_7）的存储芯片采用线选法作为译码驱动方式的结构示意图。

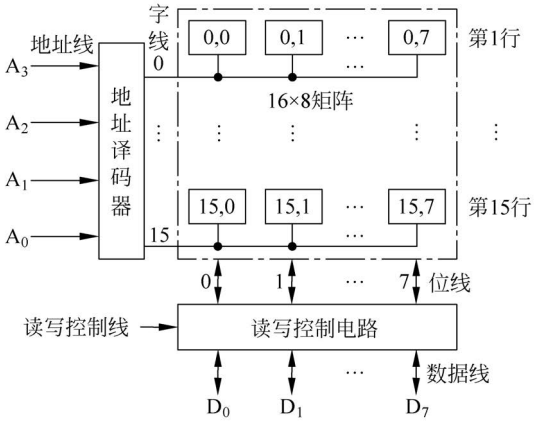


图 3-10　16×8 位线选法结构示意图

16×8 位的存储芯片可看作一个 16×8 的矩阵，根据地址线送来的地址信号确定字线，再由该字线选中矩阵的某一行，然后就可以对该行的 8 位存储位执行读写操作。例如，当 $A_3A_2A_1A_0=0000$ 时，第 0 根字线被选中，就可以对第 1 行的 8 位存储位执行读写操作。

（2）重合法。重合法是由横向（X 方向）的选择线和纵向（Y 方向）的选择线一起确定一个存储位。如图 3-11 所示为 1K×1 位（$1K=2^{10}$，故地址线为 10 根，分别为 A_0、A_1……A_9，其中 A_0、A_1、A_2、A_3 和 A_4 为横向的地址线，A_5、A_6、A_7、A_8 和 A_9 为纵向的地址线）的存储芯片采用重合法作为译码驱动方式的结构示意图。

1K×1 位的存储芯片可看作是一个 32×32 的矩阵，根据横向地址线送来的地址信号确定 X 方向的选择线，根据纵向地址线送来的地址信号确定 Y 方向的选择线，再由两个方向上确定的选择线选中某一存储位，然后就可以对该存储位执行读写操作。例如，$A_4A_3A_2A_1A_0=00000$，X 方向上第 0 根选择线被选中；$A_9A_8A_7A_6A_5=00000$，Y 方向上第 0 根选择线被选中，就可以对第 1 行第 1 列的存储位执行读写操作。

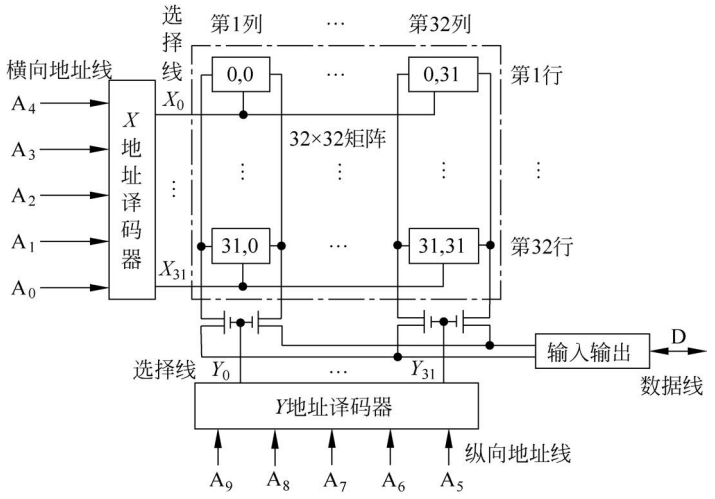


图 3-11　1K×1 位重合法结构示意图

2. 静态 RAM

1）静态 RAM 的工作原理

静态 RAM 的基本单元电路(即指构成存储位的电路)是双稳态触发器(六管 MOS)，它是在静态触发器的基础上附加门控管而组成的，依靠触发器的自保持功能存储数据，其示意图如图 3-12 所示。

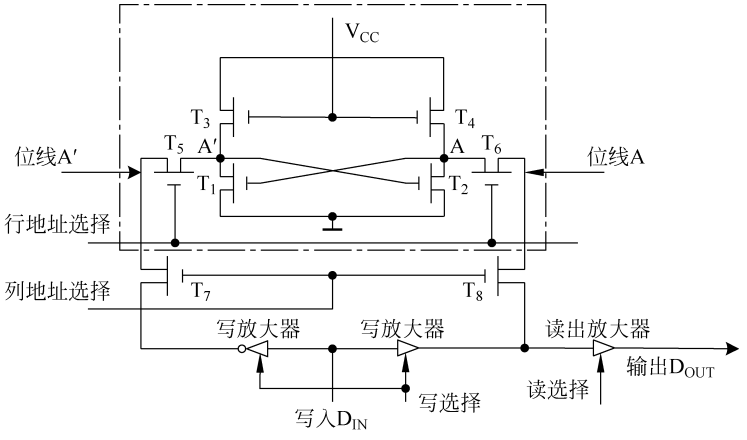


图 3-12　静态 RAM 的基本单元电路

T_1～T_8 均为 MOS 管，其中，T_1～T_4 组成触发器基本电路(当 T_1 连通，T_2 断开时，A′点高电平，触发器存信号 0；当 T_1 断开，T_2 连通时，A 点高电平，触发器存信号 1)，T_5、T_6 的作用相当于一个开关，受行地址选择信号控制，T_7、T_8 受列地址线的控制，分别与位线 A′和 A 相连。静态 RAM 的基本单元电路由 T_1～T_6 组成，并不包含 T_7 和 T_8(它们是芯片内同一列的各个基本存储单元电路所共有的)。

若需要执行读操作，即读选择有效(假设当前触发器已存有信号 1，即 T_1 断开，T_2 连通，A 点高电平)，则首先给出地址，此时行和列地址选择均有效，T_5、T_6、T_7 和 T_8 均导通，A 点高电平通过 T_6 后，再由位线 A 通过 T_8 作为读出放大器的输入信号，且此时读选择有

效，于是将信号 1 读出，此时电路仍保持其原状态，不需要再生。但当电源掉电时，存有的信息就会丢失，所以静态 RAM 为易失性存储器。

若需要执行写操作，即写选择有效，则首先将欲写入的信息送至 D_{IN} 端，经过两个写放大器，向两端输出相反电平。同时给出欲写入的地址，当行和列地址选择均有效时，使 T_5、T_6、T_7 和 T_8 均导通，并将 A 与 A′点置成完全相反电平，从而把信息写入到该基本单元电路中。

接下来以写入信号 1 为例描述其工作过程。由于欲写入信号 1，即 $D_{IN}=1$，经过左边的写放大器使位线 A′为低电平，从而使 A′点为低电平；经过右边的写放大器使位线 A 为高电平，从而使 A 点为高电平，这样就写入了信号 1。

2）静态 RAM 的读写时序

（1）静态 RAM 进行读操作时，需要提供以下外部信号：地址信号、片选信号、写允许（$\overline{WE}=1$）。如图 3-13 所示为静态 RAM 读时序图。

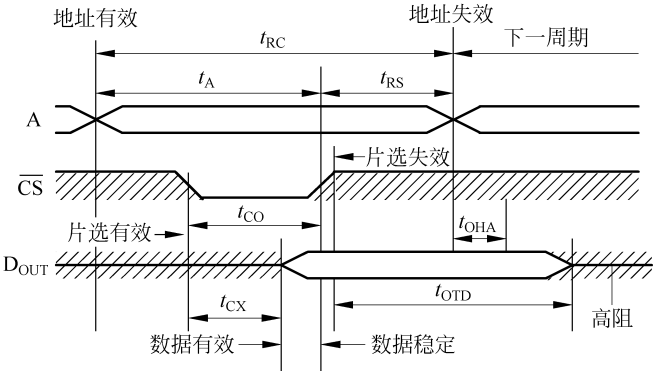


图 3-13　静态 RAM 读时序

① t_A 为读时间，表示从地址有效到数据稳定所需要的时间。

② t_{RS} 为读恢复时间，表示读操作结束后，进行相关恢复处理所需要的时间。

③ $t_{RC}=t_A+t_{RS}$ 为读周期，表示进行两次连续读操作的最小时间间隔。

④ t_{CO}：从片选有效到输出稳定的时间。

⑤ t_{OHA}：地址失效后，数据线上的有效数据维持时间，以保证所读数据可靠。

⑥ t_{CX}：从片选有效到数据有效的时间。

⑦ t_{OTD}：片选失效后数据还需在数据总线上保持的时间。

（2）静态 RAM 芯片进行写操作时，需要提供以下外部信号：地址信号、片选信号、写允许（$\overline{WE}=0$）。如图 3-14 所示为静态 RAM 写时序图。

① t_{AW} 为滞后时间，因为地址有效后，必须经过 t_{AW} 时间，$\overline{WE}$ 信号才能有效（低电平），否则可能产生写出错。

② t_W 为写入时间，保证数据可靠写入。

③ t_{WR} 为写恢复时间，$\overline{WE}$ 无效后，经 t_{WR} 时间后地址才能改变，否则也可能错误地写入。

④ $t_{WC}=t_{AW}+t_W+t_{WR}$ 为写周期，表示进行两次连续写操作的最小时间间隔。

⑤ t_{DW}：写入数据必须在写无效之前 t_{DW} 时间就送到数据总线上。

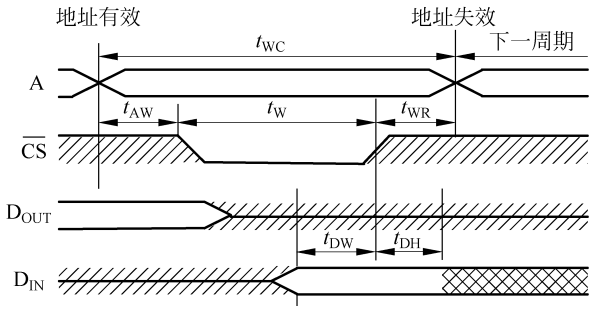


图 3-14　静态 RAM 写时序

⑥ t_{DH}：$\overline{WE}$ 无效后，数据还要保持的时间。

3）静态 RAM 与系统总线的连接

静态 RAM 与系统总线的连接方式有全地址译码方式和部分地址译码方式两种。

（1）全地址译码。全地址译码是将系统总线中的所有地址线与存储芯片的地址线相连接，参与存储器地址的译码，使存储芯片的每一个存储单元都唯一地占据内存空间的一个地址。

例如，MOS 静态 RAM 芯片 6264 与 8088 CPU 系统总线采用全地址译码方式连接的示意图如图 3-15 所示。

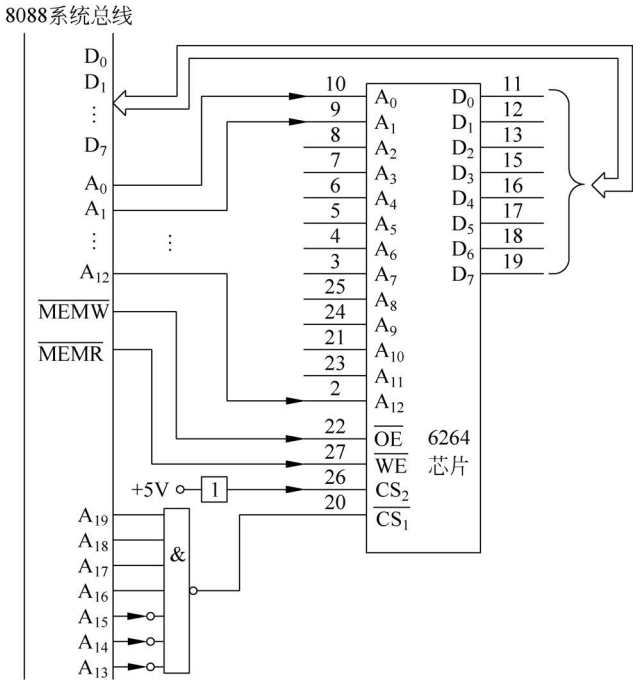


图 3-15　6264 芯片全地址译码电路

6264 芯片有 $A_0 \sim A_{12}$ 共 13 根地址总线，故共有 8K（$=2^{13}$）个存储单元，且有 $D_0 \sim D_7$ 共 8 根数据总线，因此该芯片的容量为 8K×8b。8088 CPU 的 20 根地址线（$A_0 \sim A_{19}$）全部与存储芯片的地址线相连接，参与芯片的译码。

6264 芯片的真值表见表 3-1。

表 3-1　6264 芯片的真值表

$\overline{WE}$	$\overline{CS_1}$	CS_2	$\overline{OE}$	$D_0 \sim D_7$
0	0	1	×	写入
1	0	1	0	读出
×	0	0	×	三态(高阻)
×	1	1	×	
×	1	0	×	

注意：表格中的×表示不考虑。

片选 $\overline{CS_1}$ 有效时选中该 6264 芯片，此时 $\overline{CS_1}=0$，则 A_{19} A_{18} $A_{17}A_{16}=1111$，A_{15} A_{14} $A_{13}=000$，故存储单元的高位地址为 1111000。由于 6264 芯片有 13 根地址总线，所以存储单元的低位地址为 0000000000000～1111111111111。因此，6264 芯片占据了 8088 系统内存中 00001111000000000000000～00001111000011111111111（即 0F0000H～0F1FFFH）共 8KB 的空间，该空间内的地址与 6264 芯片的存储单元一一对应。

(2) 部分地址译码。部分译码是将系统总线中的部分地址线与存储芯片的地址线相连接，参与存储器地址的译码。

例如，MOS 静态 RAM 芯片 6264 与 8088 CPU 系统总线采用部分地址译码方式连接的示意图如图 3-16 所示。

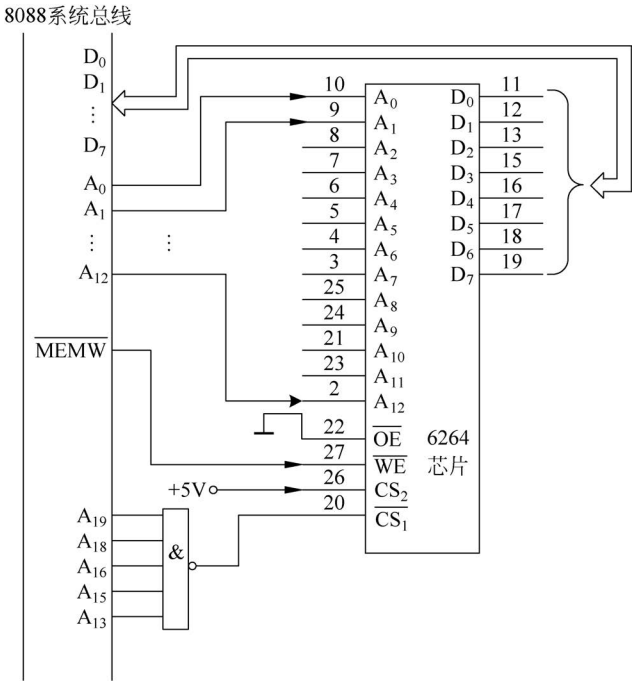


图 3-16　6264 芯片部分地址译码电路

采用部分地址译码方式时，8088 CPU 只有 18 根地址线（$A_0 \sim A_{13}$、A_{15} 和 A_{16} 以及 A_{18} 和 A_{19}）与存储芯片的地址线相连接，参与存储器地址的译码。

片选 $\overline{CS_1}$ 有效时选中该 6264 芯片，此时 $\overline{CS_1}=0$，则 $A_{19}A_{18}$ $A_{16}A_{15}A_{13}=11111$，根据 A_{14} 和 A_{17} 取值的组合情况，对应有如下 4 段内存地址空间：

① $A_{17}A_{14}=00$。故 $A_{19}\ A_{18}A_{17}A_{16}\ A_{15}\ A_{14}A_{13}=1101101$，即存储单元的高位地址为1101101，由于6264芯片有13根地址总线，所以存储单元的低位地址为0000000000000～1111111111111。此时，6264芯片占据了8088系统内存中00001101101000000000000～00001101101111111111111111(即0DA000H～0DBFFFH)共8KB的地址空间。

② $A_{17}A_{14}=01$。故 $A_{19}\ A_{18}A_{17}A_{16}\ A_{15}\ A_{14}A_{13}=1101111$，即存储单元的高位地址为1101111，由于6264芯片有13根地址总线，所以存储单元的低位地址为0000000000000～1111111111111，此时，6264芯片占据了8088系统内存中00001101111000000000000～00001101111111111111111(即0DE000H～0DFFFFH)共8KB的地址空间。

③ $A_{17}A_{14}=10$。故 $A_{19}\ A_{18}A_{17}A_{16}\ A_{15}\ A_{14}A_{13}=1111101$，即存储单元的高位地址为1111101，由于6264芯片有13根地址总线，所以存储单元的低位地址为0000000000000～1111111111111，此时，6264芯片占据了8088系统内存中00001111101000000000000～00001111101111111111111(即0FA000H～0FBFFFH)共8KB的地址空间。

④ $A_{17}A_{14}=11$。故 $A_{19}\ A_{18}A_{17}A_{16}\ A_{15}\ A_{14}A_{13}=1111111$，即存储单元的高位地址为1111111，由于6264芯片有13根地址总线，所以存储单元的低位地址为0000000000000～1111111111111，此时，6264芯片占据了8088系统内存中00001111111000000000000～00001111111111111111111(即0FE000H～0FFFFFH)共8KB的地址空间。

此时，8KB的6264芯片占据了4段8KB的内存地址空间，因此，译码时存在地址重叠的问题，而重叠的地址区域不可以分配给其他芯片，只能空着，否则会因总线竞争而使计算机系统无法正常工作。

这意味着参与译码的高位地址(A_{13}～A_{19})越少，一块芯片占据的重叠地址区域就越多，则译码时存在的地址重叠问题就越严重。

3. 动态RAM

1) 动态RAM的工作原理

动态RAM的基本单元电路利用了电容存储电荷的原理来存储信息。当电容上充满电荷时，表示存的是1；当电容上没有电荷时，表示存的是0。

由于电容上的电荷在电源不掉电的情况下一般也只能维持1～2ms，因此，必须在2ms内对其所有的存储单元执行一次恢复原状态的操作(即再生或刷新)。

常见的动态RAM的基本单元电路主要有三管式和单管式两种，而单管式是由三管式改进的，接下来先介绍三管动态RAM的基本单元电路，其示意图如图3-17所示。

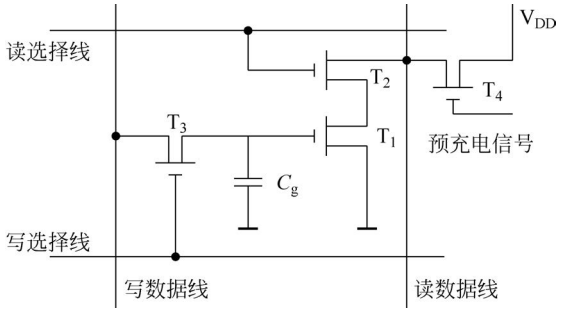


图3-17　三管动态RAM的基本单元电路

T_1～T_4 均为 MOS 管，三管动态 RAM 的基本存储单元由 T_1～T_3 组成，并不包含 T_4（它是芯片内同一列的各个基本存储单元电路所共有的）。

若需要执行读操作，即读选择有效，则首先对 T_4 置一预充电信号，使读数据线为高电平 V_{DD}。因此时读选择有效，故由读选择线打开 T_2，若 T_1 的极间电容 C_g 存有足够多的电荷（能够被认为存的是 1），使 T_1 导通，则因 T_1、T_2 均导通接地，使读数据线降为零电平，读出 0 信息。若 C_g 没有电荷（能够被认为存的是 0），则不能使 T_1 导通，读数据线为高电平不变，读出 1 信息。

若需要执行写操作，则将写入信号加到写数据线上，再由写选择线打开 T_3，这样 C_g 便能根据输入信息充电（写 1）或放电（写 0）。

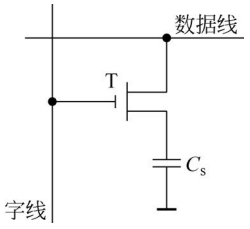


图 3-18　单管动态 RAM 的基本单元电路

为了提高集成度，单管动态 RAM 的基本单元电路将三管动态 RAM 的基本存储单元中的 T_1 去掉，再将 T_2 和 T_3 用一根 MOS 管 T 代替，其示意图如图 3-18 所示。

若需要执行读操作，则因字线为高电平，使 T 导通。此时，若 C_S 上有电荷，则经 T 在数据线上产生电流，可视为读出 1；若 C_S 上无电荷，则经 T 在数据线上无电流，可视为读出 0。

若需要执行写操作，则因字线为高电平，使 T 导通。若欲写入 1，则数据线为高电平，经 T 对 C_S 充电，使其充满电荷而实现存 1；若欲写入 0，则数据线为低电平，C_S 经 T 放电，使其无电荷而实现存 0。

2）动态 RAM 的读写时序

（1）动态 RAM 进行读操作时，需要提供以下外部信号：地址信号、片选信号、写允许（$\overline{WE}=1$）。如图 3-19 所示为动态 RAM 读时序图。

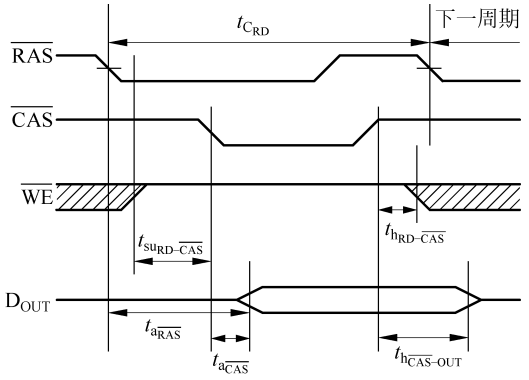


图 3-19　动态 RAM 读时序

① $t_{C_{RD}}$：动态 RAM 完成一次读操作所需的最短时间（也称读工作周期）。

② $t_{su_{RD-\overline{CAS}}}$：通常为写允许 $\overline{WE}=1$ 建立的时间。

③ $t_{h_{RD-\overline{CAS}}}$：通常为写允许 $\overline{WE}=1$ 撤除的时间。

④ $t_{a_{\overline{RAS}}}$：$\overline{RAS}$ 有效后，数据稳定前的一段时间。

⑤ $t_{a_{\overline{CAS}}}$：$\overline{CAS}$ 有效后，数据稳定前的一段时间。

⑥ $t_{h_{\overline{CAS}-OUT}}$：$\overline{CAS}$ 失效后的一段时间。

(2) 动态 RAM 芯片进行写操作时，需要提供以下外部信号：地址信号、片选信号、写允许($\overline{WE}=0$)。如图 3-20 所示为动态 RAM 写时序图。

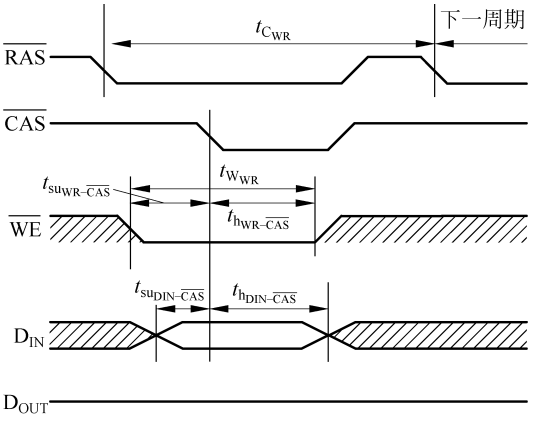


图 3-20　动态 RAM 写时序

① $t_{C_{WR}}$：动态 RAM 完成一次写操作所需的最短时间。

② $t_{su_{WR-\overline{CAS}}}$：$\overline{CAS}$ 有效前的一段时间。

③ $t_{h_{WR-\overline{CAS}}}$：$\overline{CAS}$ 有效后的一段时间。

④ $t_{su_{DIN-\overline{CAS}}}$：写入数据在 $\overline{CAS}$ 有效前的一段时间。

⑤ $t_{h_{DIN-\overline{CAS}}}$：写入数据在 $\overline{CAS}$ 有效后的一段时间。

3) 动态 RAM 的刷新

由于电容上的电荷只能保持 1～2ms，因此在 2ms 内必须对动态 RAM 的存储单元进行刷新。通常把从上一次对整个存储器的刷新结束到本次对整个存储器的刷新结束所经历的时间称为刷新周期。

动态 RAM 的刷新通常有 3 种方式，分别为集中刷新方式、分散刷新方式和异步刷新方式(结合集中和分散刷新)。

(1) 集中刷新方式。集中刷新是指在一个刷新周期内，利用一段固定的时间依次对存储器的所有行进行刷新，在此期间不能对存储器执行读或写操作。

【例 3-3】 已知存取周期为 0.5μs，刷新周期为 2ms，试采用集中刷新方式对 128 行的存储芯片进行刷新。

解： 刷新周期 2ms(=2000μs，占 2000μs/0.5μs=4000 个存取周期)，对 128 行集中刷新共需 64μs(=128×0.5μs，占 128 个存取周期)，剩下的 1936μs(=2000μs−64μs，占 3872 个存取周期)用来读、写或维持信息，如图 3-21 所示。

(2) 分散刷新方式。分散刷新是将对存储器所有行的刷新分散到每个存取周期中去完成。具体做法是将存取周期 t_C 分为两半，前半段时间 t_M 用于读、写或维持操作，后半段时间 t_R 用于刷新，即 $t_C=t_M+t_R$，且 $t_M==t_R$。可见，分散刷新将存取周期延长了，因此严重降低了系统的速度。

【例 3-4】 已知读写周期为 0.5μs，试采用分散刷新方式对 128 行的存储芯片进行刷新。

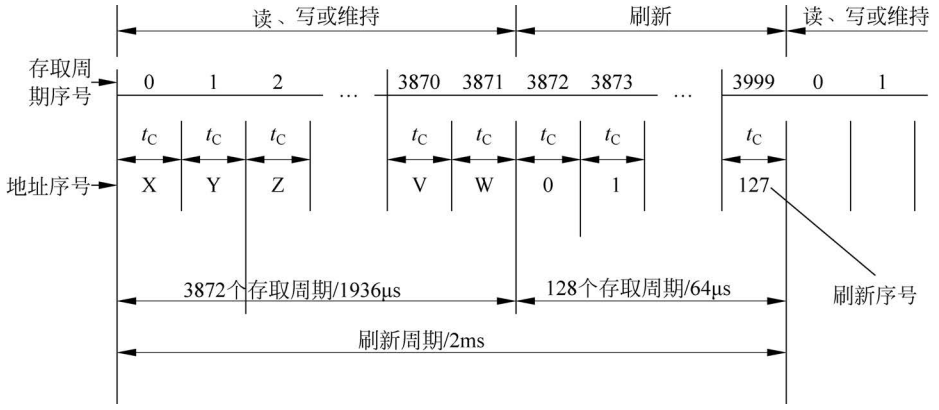


图 3-21 集中刷新的时间分配

解：读写周期为 0.5μs，即 $t_M=0.5\mu s$，则刷新时间 $t_R=0.5\mu s$，存取周期 $t_C=0.5\mu s+0.5\mu s=1\mu s$，故每隔 128μs(=128×1μs)就可以将存储芯片的所有行全部刷新一遍，如图 3-22 所示。

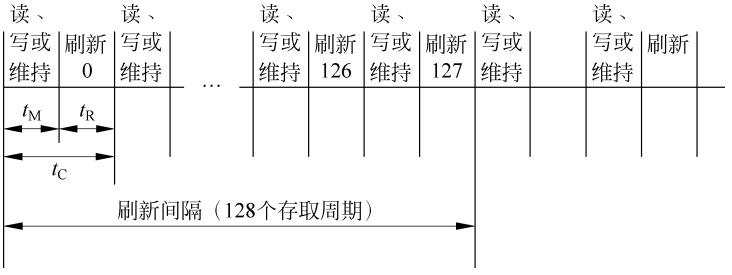


图 3-22 分散刷新的时间分配

(3) 异步刷新方式。将以上两种刷新方式结合起来便得到异步刷新方式，它一方面缩短了每次因刷新导致的不能进行读写操作的时间，另一方面又充分利用了 2ms 的最大刷新间隔。具体做法是用刷新周期除以行数，得到两次刷新操作之间的时间间隔 t，再将 t 分为两段时间，前段时间用于读、写或维持操作，后段时间用于刷新，然后利用逻辑电路每隔时间 t 产生一次刷新请求。

【例 3-5】 假设刷新周期为 2ms，存取周期为 0.5μs，试采用异步刷新方式对 128 行的存储芯片进行刷新。

解：将刷新周期分割为 128 个时间段，则每个时间段为 15.6μs(≈2000μs/128)，每个 15.6μs 的前一部分时间用于读、写或维持，后一部分(0.5μs)用于将某一行存储单元刷新，如图 3-23 所示。

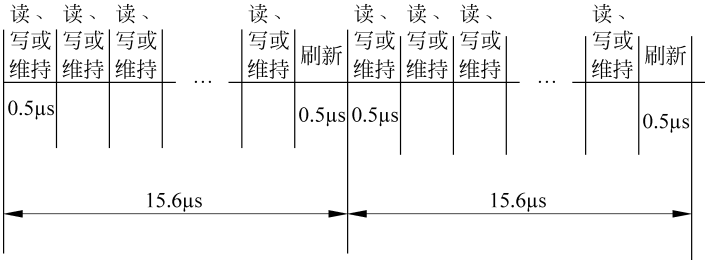


图 3-23 异步刷新的时间分配

4. 静态 RAM 和动态 RAM 的比较

为了能够更好地掌握静态 RAM 和动态 RAM，现将它们的特点总结如表 3-2 所示。

表 3-2 静态 RAM 和动态 RAM 的特点

特点 \ 类型	静态 RAM(SRAM)	动态 RAM(DRAM)
存储信息原理	触发器	电荷
需要刷新	不需要	需要
送行列地址	同时送	分两次送
运行速度	快	慢
集成度	低	高
功耗	大	小
存储成本	高	低
主要用途	高速缓存	内存

注意：

(1) 在同样大小的芯片中，动态 RAM 的集成度远高于静态 RAM。例如，动态 RAM 的基本单元电路可集成为一个 MOS 管，而静态 RAM 的基本单元电路通常为 4～6 个 MOS 管。

(2) 动态 RAM 的行列地址是按先后顺序传送的，这是为了减少芯片引脚，同时封装的尺寸也在减小。

3.2.3 只读存储器

早期的掩模型只读存储器 MROM 一旦写入信息后便无法对其内容进行更改，灵活性十分差。为了解决这一问题，先后设计出了可编程只读存储器(PROM)、可擦除可编程只读存储器(EPROM)以及用电可擦除可编程的只读存储器(EEPROM)。

1. 掩模只读存储器

在制造掩模 ROM 时，生产厂家采用掩模技术将信息写入存储器中，通常根据行选择线和列选择线交叉处是否导通来区分存 0 还是存 1，一旦制成后，掩模 ROM 所存储的信息将无法更改。

如图 3-24 所示为 MOS 型掩模 ROM，其容量为 1K×1 位(1K＝2^{10}，共有 10 根地址线，行和列地址线各 5 根)，采用重合法进行地址译码，行和列地址线经其译码器译码后，由相应的行和列选择线确定待操作的存储位。

若行选择线和列选择线的交叉处有耦合元件 MOS 管，则因导通使列选择线为低电平，经读出放大器反向为高电平，从而输出 1；若行选择线和列选择线的交叉处无耦合元件 MOS 管，则无法导通，列选择线仍为高电平，经读出放大器反向为低电平，从而输出 0。

由于在掩模 ROM 制成后，行选择线和列选择线的交叉处是否有耦合元件 MOS 管已确定，因此无法更改其所存储的信息。

2. 可编程只读存储器

为了能够批量生产只读存储器并满足用户的需求，于是设计出了可以实现一次性编程的只读存储器 PROM。如图 3-25 所示为一种 PROM 的基本单元电路，它是由双极型电路(由空穴和电子两种载流子同时导电形成的电路)和熔丝构成的，若熔丝断开，则表示存储的是 0；若熔丝未断，则表示存储的是 1。

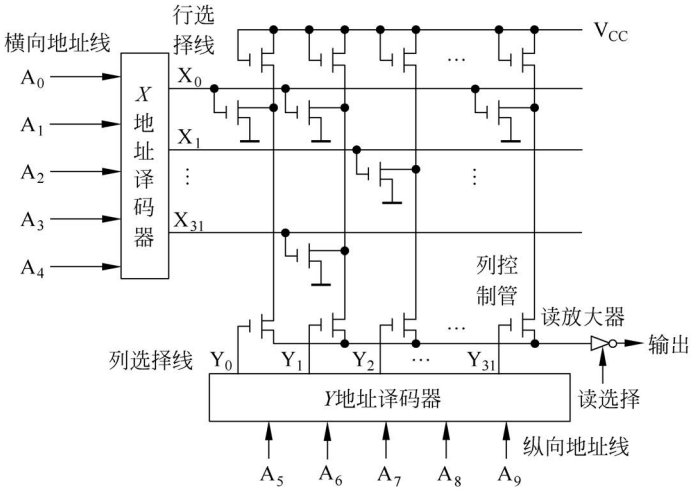


图 3-24　1K×1 位的 MOS 管掩模 ROM

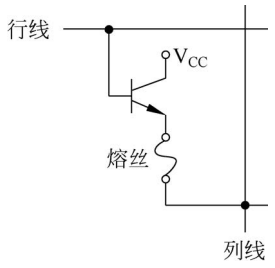


图 3-25　双极型镍铬熔丝式单元电路

生产商可利用该双极型镍铬熔丝式单元电路批量生产不同容量的 PROM 芯片。而用户在使用时，则可以选择合适容量的 PROM 芯片，然后按照自己的需求将信息依次存入芯片行列选择线交叉处的耦合元件内。若欲存入 0，则给耦合元件通以大电流将其熔丝烧断；若欲存入 1，则让熔丝保持原状。由于断开的熔丝是无法恢复的，因此这种 ROM 通常只能实现一次编程。如图 3-26 所示为已存入信息的 16×1 位的双极型镍铬熔丝式 PROM 芯片。

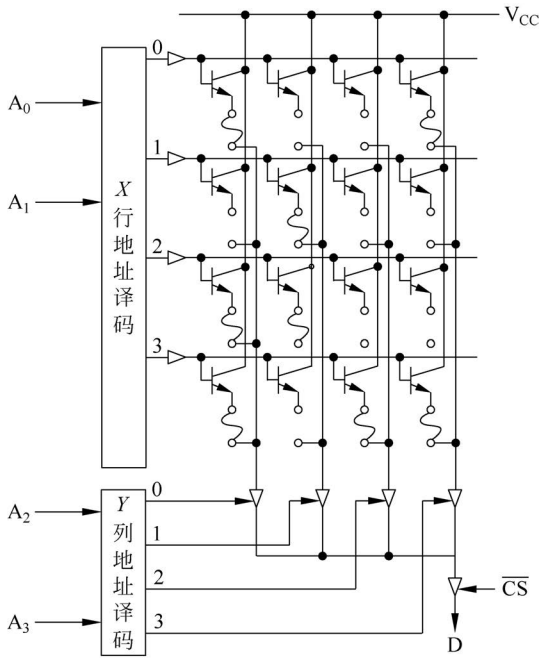


图 3-26　16×1 位双极型镍铬熔丝式 PROM

3. 可擦除可编程只读存储器

EPROM是一种可擦除可编程只读存储器。对该存储器存入信息后，若需要修改，则先将原信息全部除去，再重新存入新的信息，这样便可以实现多次存入信息。

EPROM分为两类：一类是由浮动栅雪崩注入型MOS管构成的，又称FAMOS(Floating gate Avalanche injection MOS)型EPROM；另一类是由叠栅雪崩注入型MOS管构成的，又称为SAMOS(Stacked gate Avalanche injection MOS)型EPROM。目前较为常用的是前者，如图3-27所示为N型沟道浮动栅MOS电路的逻辑符号图。

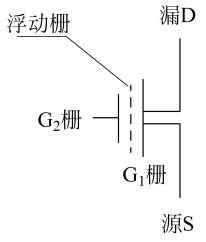


图3-27 N型沟道浮动栅MOS电路的逻辑符号

若对漏D端加上约几十伏的脉冲电压，使得沟道中的电场足够强，则会造成雪崩，产生很多高能量电子。此时，若在G_2栅上加上正电压，则使MOS管呈0状态；否则使MOS管呈1状态。当需要进行改写时，可先用紫外线照射，将原信息全部除去，然后再重新存入新的信息。

相对于PROM来说，EPROM的灵活性有了较大的提高，但改写过程仍比较复杂。因为有时候可能只需要改写很少的一部分信息，却需要对全部的信息进行擦写(即全局擦写)。

4. 电可擦可编程只读存储器

由于使用紫外线照射的方法对EPROM改写需要较长的擦除时间，并且不能对指定单元进行改写，因此需要对其进一步改进。通过使用电气方法既可实现针对需要改写的部分进行擦写(即局部擦写)，也可实现对全部的信息进行擦写(即全局擦写)，这种改进后的EPROM被称为EEPROM。

EEPROM的擦除不需要借助于其他设备，它仅用电子信号来修改其中的内容，而且最小修改单位为字节。EEPROM属于双电压芯片，在写入数据时，只需用厂商提供的专用刷新程序就可以轻而易举地进行改写。

基本输入输出系统(Basic Input Output System，BIOS)就很好地利用了EEPROM芯片的双电压特性。在对BIOS中的程序进行升级时，只需把相应地跳线开关拨至OFF位置(即给芯片加上相应的编程电压)，就可以方便地升级；平时使用时，则把跳线开关拨至ON位置，这样可以防止CIH类的病毒对BIOS芯片进行非法修改。

3.2.4 闪速存储器

闪速存储器是在EPROM和EEPROM制造工艺的基础上生产的一种高密度非易失性存储器，具有如下优点：

(1) 不仅价格便宜，而且集成度高、功耗低。

(2) 具有电可擦除可编程的特性，并且其擦写的速度比一般标准的EEPROM快得多，已经和RAM相当。

(3) 具有高速编程的特点(如采用快速脉冲编程算法对28F256闪速存储芯片每字节编程仅需100μs)。

(4) 存储器访问周期短。

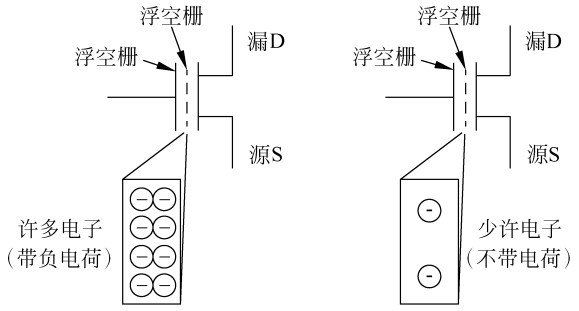
(5) 与计算机之间接口的设计比较简单。

综上所述，闪速存储器集成了众多类型存储器的优点，是存储技术发展史上的一大飞跃。闪速存储器通常有三个基本操作：编程操作、读取操作和擦除操作。

接下来详细介绍闪速存储器的基本单元电路及其阵列结构。

1. 闪速存储器的基本单元电路

如图 3-28 为闪速存储器的基本单元电路，由单个 MOS 晶体管组成。当浮空栅上有很多电子(带负电)时，意味着浮空栅上有很多负电荷，如图 3-28(a)所示，这种情况被认为存储的是 0；当浮空栅上只有少许电子时，意味着浮空栅上不带电荷，如图 3-28(b)所示，这种情况被认为存储的是 1。



(a) 存储0　　(b) 存储1

图 3-28　闪速存储器的基本单元电路

2. 闪速存储器的结构

闪速存储器的结构如图 3-29 所示。

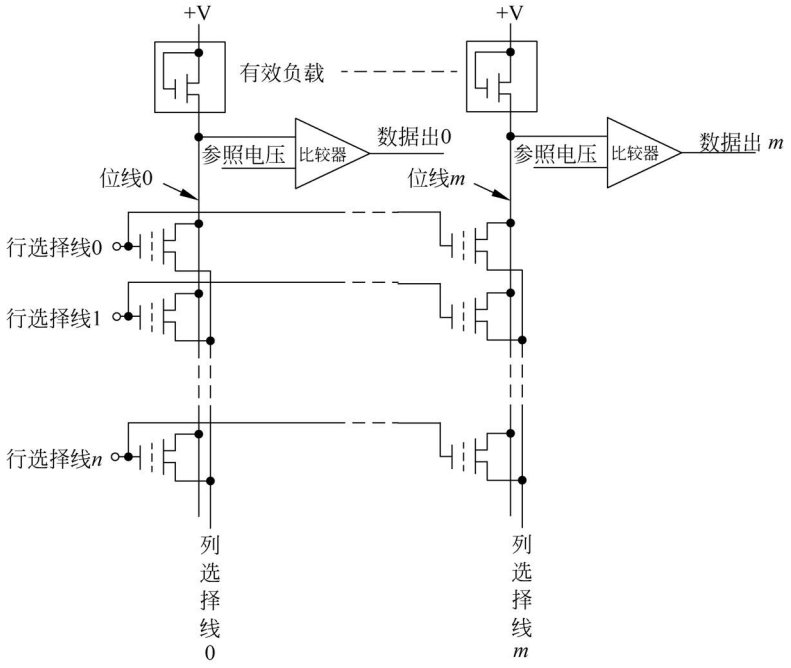


图 3-29　闪速存储器的结构

当执行读操作时，若某个存储位存 1，则晶体管导通，与它所在的位线接通，有电流通过位线，所经过的负载上产生一个电压降送至比较器的一个输入端，与另一端的参照电压做比较，比较器输出一个标志为逻辑 1 的电平；若某个存储位存 0，则晶体管不导通，没有电流通过位线，比较器输出一个标志为逻辑 0 的电平。

3.3　存储器与 CPU 的连接

CPU 是计算机运算和控制的核心，存储器是计算机的记忆部件，用于存放程序和数据。如果希望对存储器中的这些程序和数据执行相应地操作，就必须把它们送至 CPU，由此就产生了存储器与 CPU 的连接的问题。

3.3.1　存储容量的扩展

由于单个存储芯片的容量很难满足实际需求，因此需要扩大芯片的容量。给定某一存储芯片，通常有两种方案可以扩大其容量：一种方案是考虑增大芯片的面积以容纳更多的晶体管，从而扩大芯片的容量；另一种方案是考虑将若干片存储芯片以一定的方式连在一起，从而组成更大容量的存储器，即存储容量的扩展，在这一方案中，通常包括位扩展、字扩展和字位扩展。

1. 位扩展

位扩展是指增加存储字长，即将若干片 $S\times D_1$ 位的存储芯片连在一起组成 $S\times D_2$ 位的存储器，需要的存储芯片的数目为 $\dfrac{D_2}{D_1}$。

【例 3-6】 现有 8K×1 位的存储芯片若干，试采用位扩展将其扩展为一个 8K×8 位的存储器。

解：共需要 8÷1＝8 片。即需要 8 片 8K×1 位的存储芯片才能组成 8K×8 位的存储器，如图 3-30 所示。图中 8 片存储芯片分别与 CPU 的 8 根数据总线（D_0～D_7）相连，每片存储芯片的 13（2^{13}＝8K）根地址总线均与 CPU 的 13 根地址总线（A_0～A_{12}）相连。

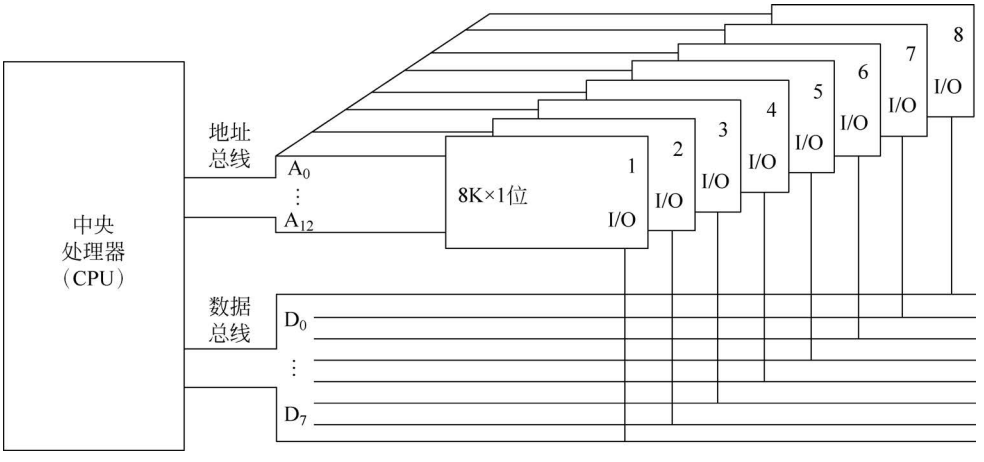


图 3-30　8 片 8K×1 位的芯片组成 8K×8 位的存储器

2. 字扩展

字扩展是指增加存储单元的数量，即将若干片 $S_1 \times D$ 位的存储芯片连在一起组成 $S_2 \times D$ 位的存储器，需要的存储芯片的数目为 $\frac{S_2}{S_1}$。

【例 3-7】 现有 16K×8 位的存储芯片若干，试采用字扩展将其扩展为一个 64K×8 位的存储器。

解： 共需要 64÷16＝4 片。即需要 4 片 16K×8 位的存储芯片才能组成 64K×8 位的存储器，如图 3-31 所示。

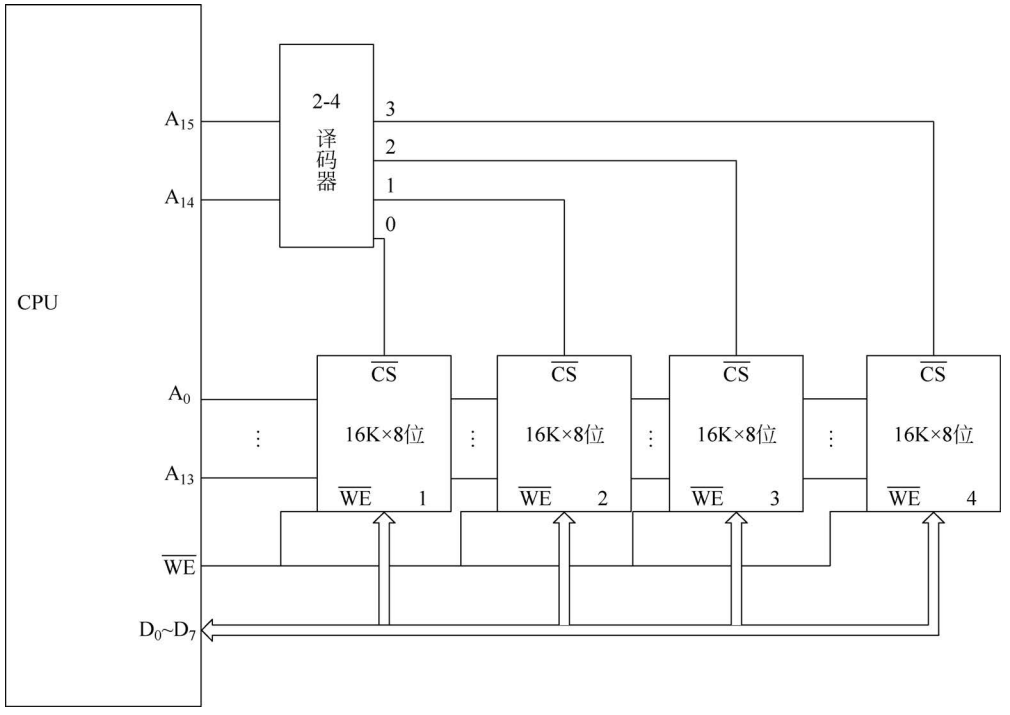


图 3-31　4 片 16K×8 位的存储芯片组成 64K×8 位的存储器

每片存储芯片的 8 根数据总线均与 CPU 的 8 根数据总线（$D_0 \sim D_7$）相连。CPU 高 2 位的地址总线（A_{14} 和 A_{15}）作为片选信号，该信号通过 2-4 译码器译码后确定选中的存储芯片，而低 14 位的地址总线（$A_0 \sim A_{13}$）则均分别与 4 片存储芯片低 14 位的地址总线相连。

当 $A_{15}A_{14}$＝00 时，2-4 译码器的 0 输出端有效，选中 1 号芯片，该芯片的地址范围为 00000000000000000～00111111111111111。

当 $A_{15}A_{14}$＝01 时，2-4 译码器的 1 输出端有效，选中 2 号芯片，该芯片的地址范围为 01000000000000000～01111111111111111。

当 $A_{15}A_{14}$＝10 时，2-4 译码器的 2 输出端有效，选中 3 号芯片，该芯片的地址范围为 10000000000000000～10111111111111111。

当 $A_{15}A_{14}$＝11 时，2-4 译码器的 3 输出端有效，选中 4 号芯片，该芯片的地址范围为

11000000000000000～1111111111111111。

3. 字位扩展

字位扩展是指既增加存储单元的数量又增加存储字长，即将若干片 $S_1 \times D_1$ 位的存储芯片连在一起组成 $S_2 \times D_2$ 位的存储器，需要的存储芯片的数目为 $\frac{S_2}{S_1} \times \frac{D_2}{D_1}$。

【例 3-8】 现有 16K×4 位的存储芯片若干，试采用字位扩展将其扩展为一个 64K×8 位的存储器。

解：共需要(64÷16)×(8÷4)片＝4×2 片＝8 片。即需要 8 片 16K×4 位的存储芯片才能组成 64K×8 位的存储器，如图 3-32 所示。

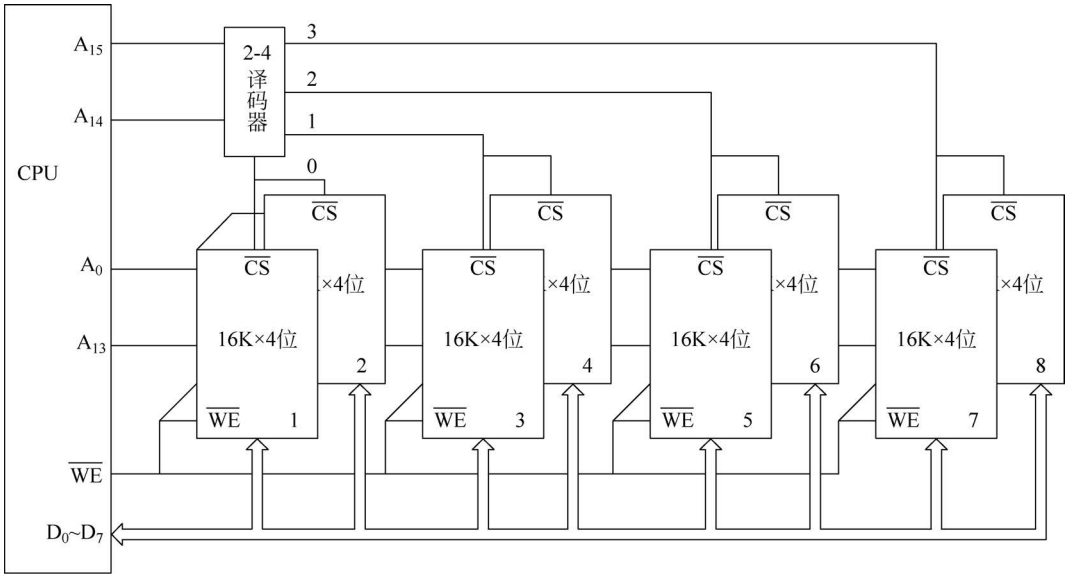


图 3-32　8 片 16K×4 位的存储芯片组成 64K×8 位的存储器

8 片存储芯片分为 4 组，每组两片，即由 2 片 16K×4 位的芯片组成 16K×8 位的芯片。在每组的两片存储芯片中，一片的 4 根数据总线与 CPU 高 4 位的数据总线相连，另一片的 4 根数据总线与 CPU 低 4 位的数据总线相连。

由于 2^{16}＝64K，即地址总线的宽度为 16 位。CPU 高 2 位的地址总线(A_{14} 和 A_{15})作为片选信号，该信号通过 2-4 译码器译码后确定选中的存储芯片组，而低 14 位的地址总线(A_0～A_{13})则均分别与 4 组的每片存储芯片低 14 位的地址总线相连。

3.3.2　存储芯片的地址分配和片选

由之前的介绍可知，存储器通常是由多片存储芯片组成的，因此，当 CPU 要访问存储器时，首先需要选择存储芯片(即片选)；然后在该存储芯片内选择存储单元(即字选)。

设 CPU 共有 M 根地址总线，S 片存储芯片的规格为 $2^L \times D$ 位。字选通常是由 CPU 的 L 根低位地址总线确定的，将用于字选的地址总线称为字选地址总线。$N=M-L$，当 $N \geqslant S$ 时，片选由 CPU 的位于字选地址总线之后的 S 根地址总线确定，称为线选片选法；

当 $N<S$ 时，片选由 CPU 的位于字选地址总线之后的 T（$T\leqslant N$ 且 $2^T\geqslant S$）根地址总线确定，称为译码片选法，将用于片选的地址总线称为片选地址总线。

1. 线选片选法

线选片选法是将片选地址总线分别与各存储芯片的片选端相连，当某片选地址总线的信息为 0 时（每次只能有一根片选地址总线的信息为 0，其他均为 1），就选中与之对应的存储芯片或芯片组。

【例 3-9】 假设 CPU 共有 16 根地址总线（$A_0\sim A_{15}$），组成存储器的 4 片存储芯片（分别编号为 0#、1#、2# 和 3#）的规格均为 $2^{12}\times 8$ 位，试讨论应采用何种方式进行片选，并分析根据片选地址总线的信息如何选中与之对应的存储芯片。

解：CPU 的低 12 位地址总线（$A_0\sim A_{11}$）作为字选总线，剩下的高 4 位地址总线（A_{12}、A_{13}、A_{14} 和 A_{15}）刚好分别与 4 片存储芯片相连，因此应采用线选片选法，分以下四种情况讨论每次选中的存储芯片。

（1）当 $A_{15}A_{14}A_{13}A_{12}=1110$，即倒数第 1 位为 0 时，选中编号为 0# 的存储芯片。

（2）当 $A_{15}A_{14}A_{13}A_{12}=1101$，即倒数第 2 位为 0 时，选中编号为 1# 的存储芯片。

（3）当 $A_{15}A_{14}A_{13}A_{12}=1011$，即倒数第 3 位为 0 时，选中编号为 2# 的存储芯片。

（4）当 $A_{15}A_{14}A_{13}A_{12}=0111$，即倒数第 4 位为 0 时，选中编号为 3# 的存储芯片。

线选片选法的优点是不需要对片选地址总线的信息进行译码就可以直接选中芯片或芯片组，故线路的设计较为简单；缺点是地址空间不连续，不能充分利用系统的存储器空间，从而造成地址的浪费。

2. 译码片选法

译码片选法是将片选地址总线与地址译码器相连，由地址译码器的输出结果选中对应的存储芯片或芯片组。

【例 3-10】 假设 CPU 共有 14 根地址总线（$A_0\sim A_{13}$），组成存储器的 4 片存储芯片（分别编号为 0#、1#、2# 和 3#）的规格均为 $2^{12}\times 8$ 位，试讨论应采用何种方式进行片选，并分析片选地址总线的信息如何选中与之对应的存储芯片。

解：CPU 的低 12 位地址总线（$A_0\sim A_{11}$）作为字选总线，剩下的高 2 位地址总线（A_{12} 和 A_{13}）无法分别与 4 片存储芯片直接相连，因此应采用译码片选法，分以下四种情况讨论每次选中的存储芯片。

（1）当 $A_{13}A_{12}=00$ 时，对应十进制数 0，此时选中编号为 0# 的存储芯片。

（2）当 $A_{13}A_{12}=01$ 时，对应十进制数 1，此时选中编号为 1# 的存储芯片。

（3）当 $A_{13}A_{12}=10$ 时，对应十进制数 2，此时选中编号为 2# 的存储芯片。

（4）当 $A_{13}A_{12}=11$ 时，对应十进制数 3，此时选中编号为 3# 的存储芯片。

3.3.3 存储器与 CPU 连接的实现

在连接前，需要选择合理的存储芯片组成存储器。连接时，一般要实现地址总线、控制总线（读写控制线和片选线等）与数据总线的连接。CPU 对存储器执行读或写操作时，首先由地址总线给出地址信号，然后由控制总线给出读或写操作的控制信号，最后由数据总线传送相应的数据。

1. 合理选择存储芯片

存储芯片的选择要考虑存储芯片的类型(RAM 或 ROM)和数量。在考虑芯片的类型时,通常选用 ROM 存放系统程序、标准子程序和各类常数,而选用 RAM 存放用户程序。在考虑芯片的数量时,要尽量使连线简单方便。

2. 地址总线的连接

CPU 的地址总线往往多于存储芯片的地址总线。在连接时,通常将 CPU 的低位地址总线与存储芯片的地址总线相连,即用于字选;而 CPU 的高位地址线则一般用于存储芯片扩充时芯片的选择,即用于片选。

3. 数据总线的连接

当 CPU 与存储芯片的数据总线的数目相等时,可直接连接;而当 CPU 与存储芯片的数据总线的数目不等时,必须先对存储芯片进行位扩展,使其数据总线的数目与 CPU 的相等才能进行连接。

4. 读写控制线的连接

CPU 读写控制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。但有些 CPU 的读写控制线是分开的(读为 $\overline{RD}$,写为 $\overline{WE}$,均为低电平有效),此时 CPU 的读控制线应与存储芯片的允许读控制端相连,而 CPU 的写控制线则应与存储芯片的允许写控制端相连。

5. 片选线的连接

片选线的连接是存储器和 CPU 连接的关键。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能否接收到来自 CPU 的片选有效信号。

片选有效信号与 CPU 的访存控制信号 $\overline{MREQ}$(低电平有效)有关,因为只有当 CPU 要求访存时,才需要选择存储芯片。若 CPU 要求访问 I/O,则 $\overline{MREQ}$ 为高电平,表示无需存储器工作。

【例 3-11】 假设 CPU 有 16 根地址总线,8 根数据总线,$\overline{MREQ}$ 为访存控制信号(低电平有效),$\overline{WE}$ 为读写控制信号(高电平为读,$\overline{WE}=1$,低电平为写,$\overline{WE}=0$)。现有 1K×4 位的 RAM、4K×8 位的 RAM、8K×8 位的 RAM、2K×8 位的 ROM、4K×8 位的 ROM 和 8K×8 位的 ROM 若干,以及 74138 译码器和各种门电路,如图 3-33 所示。试画出存储器和 CPU 连接的示意图,要求如下:

(1) 存储器地址空间分配:6000H～67FFH 为系统程序区,6800H～6BFFH 为用户程序区。

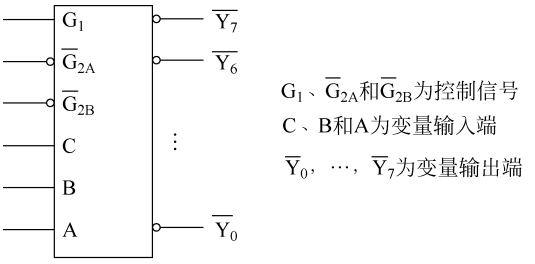
(2) 合理选用上述存储芯片,并说明各选几片。

(3) 详细画出存储芯片的片选逻辑图。

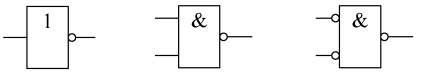
解:

(1) 合理选择存储芯片。将用十六进制表示的地址范围改用二进制表示,如图 3-34 所示,以便确定容量,并结合该地址范围在计算机中的作用来选择存储芯片。

系统程序区的容量为 2K×8 位,故应选择 1 片 2K×8 位的 ROM;用户程序区的容量为 1K×8 位,故应选择 2 片 1K×4 位的 RAM。



(a) 74138译码器



(b) 各种门电路

图 3-33　译码器和门电路

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	系统程序区
↓								↓								
0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	2K×8位
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	用户程序区
↓								↓								
0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1K×8位

图 3-34　十六进制表示的地址范围改写为用二进制表示

(2) 地址总线的连接。

① 将 CPU 的低 11 位地址总线(A_0～A_{10})与 2K×8 位 ROM 的地址总线相连。

② 将 CPU 的低 10 位地址总线(A_0～A_9)与 2 片 1K×4 位 RAM 的地址总线相连。

(3) 数据总线的连接。

① 将 CPU 的数据总线与 2K×8 位 ROM 的数据总线相连。

② 2 片 1K×4 位 RAM 的数据总线分别与 CPU 的高 4 位和低 4 位数据总线相连。

(4) 读写控制线的连接。RAM 芯片的读写控制端与 CPU 的读写控制端 $\overline{WE}$ 直接相连，而 ROM 只用于读，故不需要连接读写控制端。

(5) 片选线的连接。由图 3-33(a)给出的 74138 译码器输入逻辑关系可知，要使译码器正常工作，需保证控制端 G_1 为高电平，$\overline{G}_{2A}$ 和 $\overline{G}_{2B}$ 为低电平。在(1)中地址范围用二进制表示时，A_{14} 始终为高电平，A_{15} 始终为低电平，这刚好与 G_1 高电平，$\overline{G}_{2A}$ 低电平对应，即可将 G_1 与 A_{14} 相连，$\overline{G}_{2A}$ 与 A_{15} 相连。而访存控制信号 $\overline{MREQ}$(低电平有效)又刚好与 $\overline{G}_{2B}$ 低电平对应，即可将 $\overline{G}_{2B}$ 与 $\overline{MREQ}$ 相连。

CPU 剩下的地址总线 A_{13}、A_{12} 和 A_{11} 分别与译码器的输入端 C、B 和 A 相连，作为片选。当译码器的输出端 $\overline{Y}_4$ 有效时，选中 ROM；$\overline{Y}_5$ 和 A_{10} 同时有效均为低电平时，选中 2 片 RAM。

经过以上分析，可以画出存储器和 CPU 的连接示意图(图 3-35)。

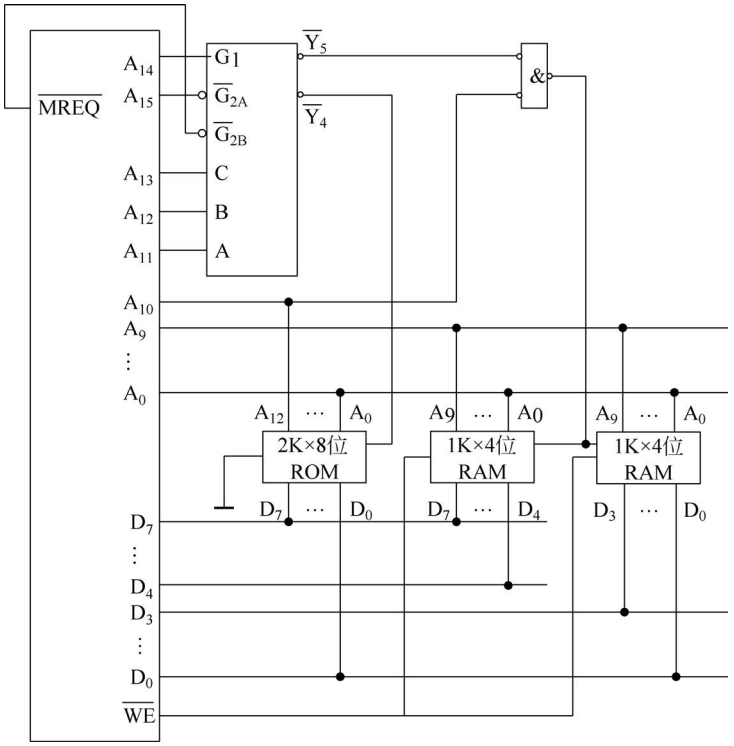


图 3-35 存储器和 CPU 的连接示意图

3.4 并行存储器

为了提高访存速度，除了寻找更高速的元件和采用存储器层次结构外，还可以采用具有并行技术的存储器，本节将要介绍的双端口 RAM 和多模块存储器就是基于并行技术实现的。

3.4.1 双端口 RAM

双端口 RAM 是指同一个存储器具有两组相互独立的端口（即地址总线、数据总线以及控制总线），如图 3-36 所示，这两组端口都可以对存储器执行读写操作。

通过两组端口执行读写操作时，若两组端口不同时对存储器执行读或写操作，则不会发生错误；若两组端口同时对存储器执行读或写操作，则情况如下：

① 对存储器的不同存储单元执行读或写操作，不会发生错误。

② 对存储器的同一存储单元执行读操作，不会发生错误。

③ 对存储器的同一存储单元执行写操作，会发生写入错误。

④ 对存储器的同一存储单元，一个执行写操作，另一个执行读操作，会出现读出错误。

为了防止通过两组端口执行读写操作时出现错误，双端口 RAM 设置了“忙”标志。若两组端口执行读写操作时会出现错误，则只允许一组端口访问，而对另一组端口置“忙”标志以延迟其操作。

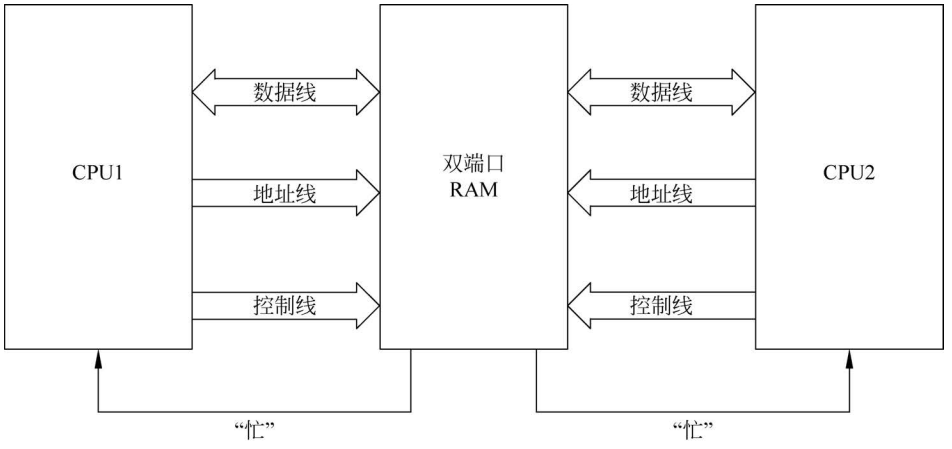


图 3-36　双端口 RAM 示意图

3.4.2　多模块存储器

1. 单体多字存储器

由于程序和数据在存储器中是连续存放的，因此 CPU 访存取出的信息也是连续的。如果可以在同一存取周期内取出 n 条指令，将每一条指令送至 CPU 执行，即每隔 $1/n$ 存取周期就向 CPU 送一条指令，这样将增加存储器的带宽，提高存储器的速度。如图 3-37 所示，单体多字存储器将存储器的存储字长增大 n 倍（同时数据总线的数目也要增大 n 倍），以存放 n 个指令字或数据字（每字 W 位）。

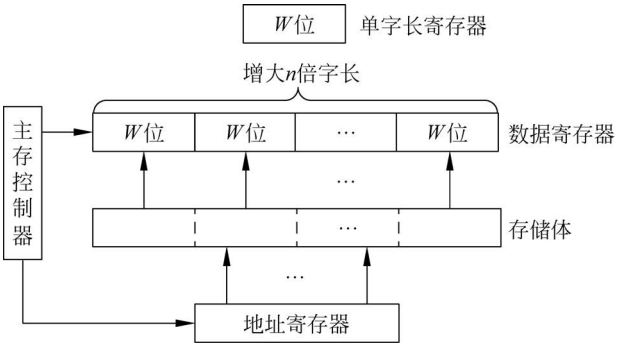


图 3-37　单体多字结构存储器

在一个存储周期内，单体多字存储器可以写入或读出的指令字或数据字的数目也增大了 n 倍。这也意味着，每次必须凑齐 n 个指令字或数据字才能作为一个存储字写入单体多字存储器中，而每次读出的一个存储字（含 n 个指令字或数据字）也不一定都是最近需要的，因此，若想最大化的实现单体多字存储器的作用，必须保证程序和数据在存储器中是连续存放的；否则一旦遇到转移指令或操作数不能连续存放，这种方法就无法充分利用 CPU 的资源。

2. 多体并行存储器

为了加快信息交换的速度，多体并行存储器采用多个模块组成主存以实现并行工作。每一模块的容量和存取速度相同，各模块都有独立的读写电路、地址译码、驱动电路、地址寄

存器和数据寄存器，它们既能并行工作又能交叉工作。

对多体并行存储器各模块的存储单元进行编址时有两种方式，分别为高位交叉编址方式和低位交叉编址方式，因此，多体并行存储器分为高位交叉编址的多体存储器和低位交叉编址的多体存储器两种。

1）高位交叉编址

高位交叉编址是对多体并行存储器的各模块按顺序编址，即对一个模块的存储单元编完址后再对下一个模块的存储单元进行编址。采用这种方式编址时，高位地址表示体号（即模块的编号），低位地址表示体内地址（即模块内的地址）。如图 3-38 所示为具有 4 个模块并采用高位交叉编址方式的多体并行存储器的示意图。

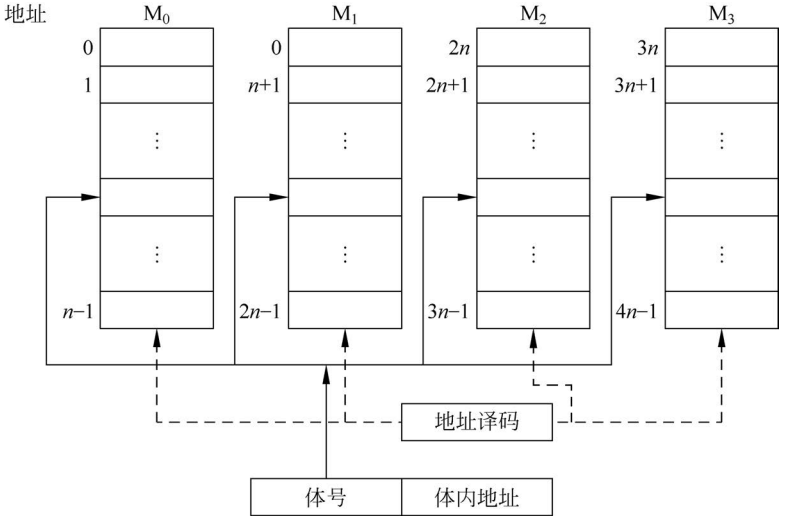


图 3-38　高位交叉编址的多体存储器

采用这种编址方式十分有利于存储器的扩充，并且只要进行合理的调动，就可以使不同的请求源同时访问不同的模块，从而实现并行工作。例如，一个模块正在和 CPU 交换信息的同时，另一个模块正在和输入输出设备交换数据。

2）低位交叉编址

低位交叉编址（也称为模 m 编址，m 为模块数）是对多体并行存储器的各模块同时编址，将连续的地址分布在相邻的模块内，故同一个模块的地址都是不连续的。采用这种方式编址时，低位地址表示体号，高位地址表示体内地址。如图 3-39 所示为具有 4 个模块并采用低位交叉编址方式的多体并行存储器的示意图。

虽然图中的 4 个模块的都有独立的读写电路、地址译码、驱动电路、地址寄存器和数据寄存器，可以实现同时获取所有模块的一个存储字，但由于通常数据总线的数目等于模块的字长，因此一次获取的多个存储字也只能在数据总线上依次传送，否则会产生冲突。此时，可以采用流水线的方式来实现对这些存储字的并行存取。

假设低位交叉编址存储器的模块数为 m，模块的存取周期为 T，总线的传送周期为 t，通常 $T>t$。为了实现流水线方式存取，可以在启动一个模块 t 时间后再启动另一个模块。这样，当先启动模块取出的一个存储字在数据总线上传送完毕后，刚好数据总线空闲，后启动模块取出的一个存储字就可以在其上传送。此时，应满足 $T=mt$。而为了保证启动某一

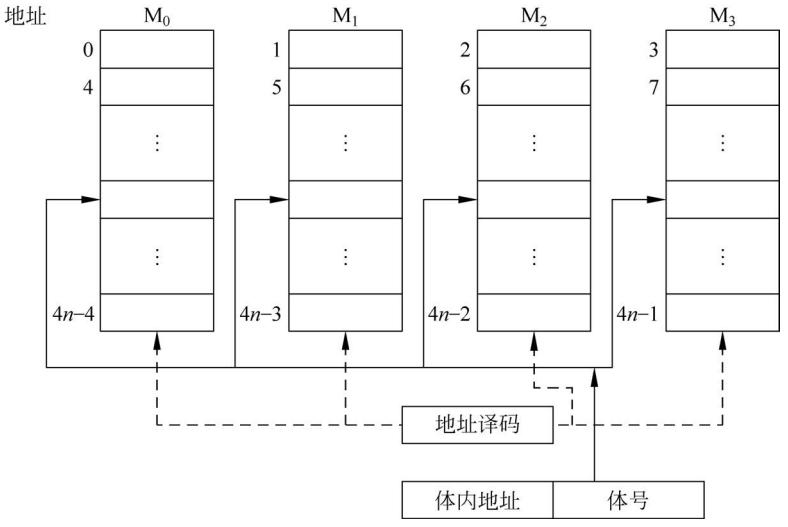


图 3-39　低位交叉编址的多体存储器

模块后，经 mt 时间再次启动该模块时，它的上一次存取操作已完成，要求存储器的模块数应大于等于 m。以四体（即 $m=4$）低位交叉编址存储器为例，采用流水线方式存取的示意图如图 3-40 所示。

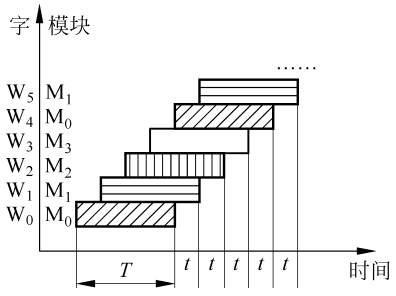


图 3-40　四体低位交叉编址存储器流水线方式存取示意图

分析图 3-40 可知，对于低位交叉编址的存储器，当需要连续读取 N 个字时，所需花费的时间 t_L 为

$$t_L=T+(N-1)t$$

而若采用高位交叉编址方式，则连续读取 N 个字所需花费的时间 t_H 为

$$t_H=NT$$

【例 3-12】 假设模块数 $m=4$，每个模块的存储字长为 64 位，存储周期 $T=200$ns，数据总线有 64 根，总线传送周期 $t=50$ns，试求出分别采用高位交叉编址和低位交叉编址时存储器的带宽。

解：连续读出 4 个字（因为 $m=4$）的信息总量为

$$M=64\times4=256\text{bit}$$

采用高位交叉编址和低位交叉编址时连续读出 4 个字的时间分别为

$$t_H=mT=4\times200\text{ns}=800\text{ns}$$

$$t_L=T+(4-1)t=200\text{ns}+3\times50\text{ns}=350\text{ns}$$

采用高位交叉编址和低位交叉编址时存储器的带宽分别为

$$W_H=M/t_H=256\text{bit}/800\text{ns}=320\text{Mbit/s}$$

$$W_L=M/t_L=256\text{bit}/350\text{ns}\approx731\text{Mbit/s}$$

多模块存储器是主存储器，因此，它可以与 CPU、辅存等请求源交换信息，故有可能出现多个请求源同时请求访问同一个存储体的情况，为了防止访问出错，需要按照一定的原则对不同的请求源设置优先级，具体原则如下：

（1）对易发生代码丢失的请求源设置最高优先级。

（2）对严重影响 CPU 工作的请求源设置次高优先级。

3.5 高速缓冲存储器

3.5.1 高速缓冲存储器简介

1. 高速缓冲存储器的产生

一方面由于辅存向主存请求的优先级高于CPU访存，因此当辅存和CPU同时请求主存时，CPU必须等待，这降低了CPU的工作效率；另一方面由于主存速度的提高始终跟不上CPU的发展，因此CPU的实际运行速度受到了主存速度的限制，这也降低了CPU的工作效率。为了提高CPU的工作效率，在CPU和主存之间设置了高速缓冲存储器Cache，它的速度比主存快，但容量比主存小，用于预存CPU近期可能需要访问的主存信息（即Cache的内容是主存一部分信息的副本），这样当CPU需要访问这些信息时，就可以直接从Cache中获取，不会因为此时主存正在响应更高优先级的请求源或主存的速度慢而降低CPU的工作效率。

CPU可以直接访问Cache而不用访问主存的前提是在Cache中预存的信息确实是CPU近期要访问的主存信息。通过对大量典型程序的执行过程进行分析，发现如果某数据或指令被使用，那么在不久之后它可能会被再次使用（称为时间局部性原理），其存储地址附近的数据或指令在不久之后也可能会被使用（称为空间局部性原理）。这是因为有些数据和指令会被多次调用（如子程序、循环程序和一些常数等），且数据和指令在主存中都是连续存放的。因此通过局部性原理（时间局部性原理和空间局部性原理的统称）就可以推测CPU近期要访问的主存信息，从而将其预存至Cache中。

2. 高速缓冲存储器的工作原理

如图3-41所示，为了与Cache（缓存）映射，将主存和缓存的存储空间都分成若干个大小相等的块，每个块包含若干个字。故主存的地址分为两部分：高 m 位表示主存的块号，低 b 位表示块内地址，则 $2^m=M$ 表示主存的块数。同样，缓存的地址也为两部分：高 c 位表示Cache的块号，低 b 位表示块内地址，则 $2^c=C$ 表示Cache的块数，且 C 远小于 M。主存和缓存都用地址的低 b 位表示块内地址，则 $2^b=B$ 表示块内的字数，它反映了块的大小，称为块长。

由于 C 远小于 M，故一个缓存块不能唯一映射到一个主存块，因此在每个缓存块中设置了标记，其内容为空或等于某一主存块号。当CPU欲访问某字时，将该字地址的高 m 位与缓存块的标记一一进行比较，若能找到与之相等的标记，则说明CPU欲访问的字已在Cache中，可直接访问Cache（CPU和Cache之间以“字”为单位交换信息），此时，称CPU访问Cache命中；否则说明CPU欲访问的字不在Cache中，需要到主存中访问之，并将该字所在的主存块调入Cache中（Cache与主存之间以“字块”为单位交换信息），此时，称CPU访问Cache未命中。

当CPU需多次执行访问操作时，Cache访问命中次数的多少反映了Cache的效率，通常用“命中率”来衡量该效率，它是指CPU要访问的主存信息已在Cache中的比率。

假设CPU访问Cache命中的次数为 N_c，未命中时访问主存的次数为 N_m，则命中率 h 为

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

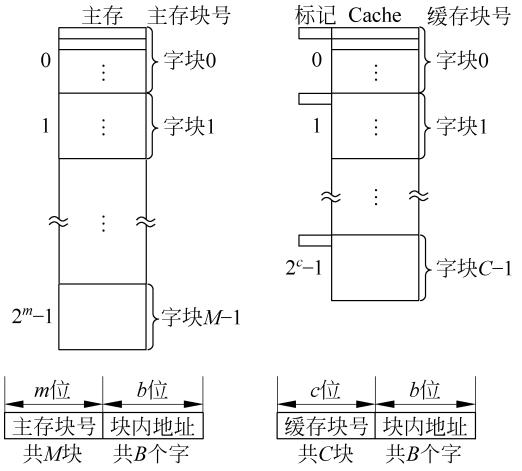


图 3-41 Cache-主存存储空间的基本结构

设每次命中时 CPU 访问 Cache 所需的时间为 t_c，而每次未命中时 CPU 访问主存所需的时间为 t_m，$1-h$ 表示未命中率，则 Cache-主存系统的平均访问时间 t_a 为

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

用 e 表示 Cache 的效率，则有

$$e = \frac{t_c}{t_a} \times 100\% = \frac{t_c}{ht_c + (1-h)t_m} \times 100\%$$

通常访问主存所需的时间 t_m 要比访问 Cache 所需的时间 t_c 大得多，因此，若希望提高 Cache 的效率，则应尽可能地增大命中率 h。

【例 3-13】 假设 CPU 执行某段程序时，访问 Cache 命中 1000 次，未命中时访问主存 25 次。已知 Cache 的存取周期为 50ns，主存的存取周期为 200ns，试求 Cache-主存系统的命中率、平均访问时间和效率。

解：

(1) 命中率 $h=1000/(1000+25)\approx 0.97$。

(2) 平均访问时间 $t_a=0.97\times 50\text{ns}+0.03\times 200\text{ns}=54.5\text{ns}$。

(3) 效率 $e=50\text{ns}/54.5\text{ns}\times 100\%\approx 91.7\%$。

理论上，Cache 的容量越大意味着命中率越高(Cache 的容量越大，能存放的主存块就越多)，效率也就越高。但是当 Cache 的容量达到一定值后，再增加 Cache 的容量时，命中率的增长就会变得十分缓慢，如图 3-42 所示。

从另一方面来看，随着 Cache 容量的增加，其成本也在不断地增加，因此需要同时考虑成本和命中率才能确定 Cache 的容量。例如，80386 主存的最大容量为 4GB，与其配套的 Cache 容量为 16KB 或 32KB 时，命中率就可以达到 95%以上。

此外，Cache 的命中率还与块长有关。程序的局部性原理指出，在已被访问的数据或指令附近的数据或指令，不久后它们可能也会被访问，因此增大块长后，一个块就可以存放更多有用的信息，从而提高了命中率。

但是当块长超过一定值后，可能会导致命中率下降，这是因为在 Cache 总容量不变的前提下，由于数据或指令是连续存放的，块长过大意味着某些近期不被访问的数据或指令占用

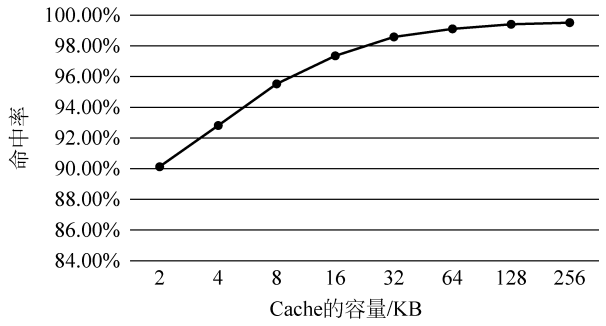


图 3-42 Cache 容量与命中率的关系

了 Cache 的空间，从而导致 CPU 当前需要访问的数据和指令不在 Cache 内（即 CPU 访问 Cache 未命中，故命中率降低）。

因此，存在一个最优的块长值，但该值很难确定，通常取 4 或 8 个字或字节（如 IBM 370/168 的字长为 64 位，即 8 个字节，其 Cache 块长为 32 个字节，即 4 个字），也可以取一个存储周期所能访问的信息长度（如 CRAY-1 的主存是 16 体低位交叉编址存储器，其存放指令的 Cache 块长为 16 个字）。

Cache 主要由存储体、地址映射变换部件以及替换部件三个模块组成，其基本结构如图 3-43 所示。

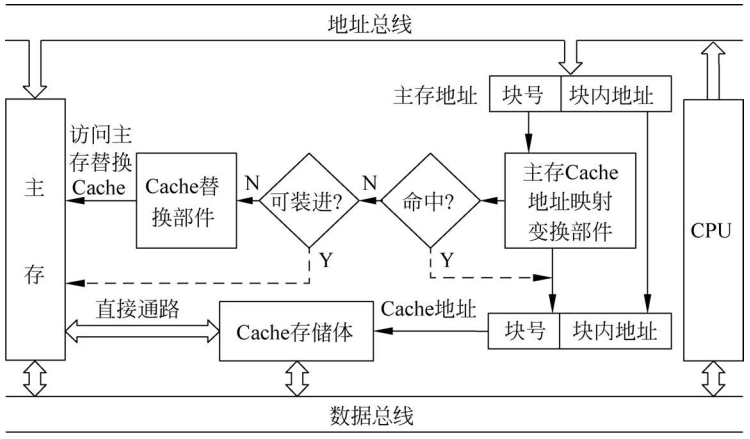


图 3-43 Cache 的基本结构

（1）Cache 存储体。Cache 存储体以块为单位与主存交换信息。由于缓存块远少于主存块，因此缓存块与主存块不能一一映射。

（2）地址映射变换部件。由于缓存块与主存块不能一一映射，因此主存地址与缓存地址就不同。地址映射变换部件负责将 CPU 送来的主存地址转换为 Cache 地址。考虑到主存地址和缓存地址都分为块号和块内地址两部分且主存块和缓存块的大小相等，因此它们的块内地址（即低位地址）相等，故地址变换主要是主存块号（即高位地址）和缓存块号之间的转换。而地址变换又与如何将主存地址映射到 Cache 中有关（这将在后续章节中详细介绍）。

（3）替换部件。若 Cache 已满，而 CPU 欲访问的主存信息又不在 Cache 中，则替换部

件按照相关的替换算法(将在后续章节中详细介绍)确定从 Cache 中移出哪个块返回主存,从而把新的主存块调入 Cache。

注意:Cache 和主存之间的信息交换对用户来说都是透明的,用户不需要知道待执行的指令或待访问的数据是否在 Cache 中,也不需要知道 Cache 的访问过程,这些都是 CPU 在执行指令过程中由某些软硬件协同工作完成的,用户只需给出这些指令或数据在主存中的地址即可。

3.5.2 地址映射与转换

地址映射是指将内存地址映射到缓存地址(即 Cache 地址)。地址映射的方式有多种,常用的有直接映射、全相联映射和组相联映射。

1. 直接映射

直接映射是先将缓存块和主存块一一映射,当缓存块不够时,再从第一个缓存块开始与余下的主存块继续一一映射,如此重复,直至主存块映射完(通常主存块数是缓存块数的整数倍),如图 3-44 所示。

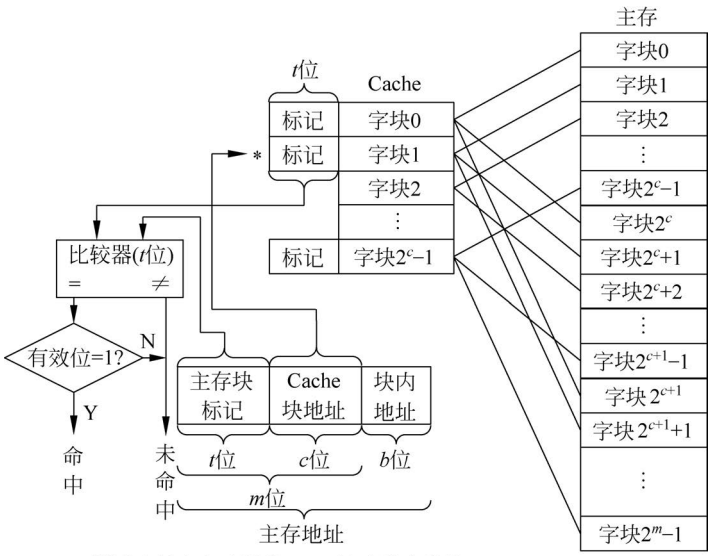


图 3-44 直接映射

主存块号与缓存块号之间的映射关系式为

```
i = j mod C
```

或

```
i = j mod 2^c
```

其中,i 为缓存块号,j 为主存块号,C 为缓存块数。主存块号对缓存块数取余的结果(即主存地址的 c 位 Cache 块地址)相等的主存块映射到同一个缓存块中,如表 3-3 所示。

表 3-3　直接映射方式主存块与缓存块的对应关系

缓存块号	主存块号
0	$0,C,\cdots,M-C$
1	$1,C+1,\cdots,M-C+1$
…	…
$C-1$	$C-1,2C+1,\cdots,M-C+1$

每个缓存块对应一组主存块(这组主存块的 Cache 地址字段的值与该缓存块的块号相等),而某一时刻该缓存块只能存放这组主存块中的某一个,存了哪一个主存块,就将其 $t(=m-c)$ 位的主存字块标记写入该缓存块的标记字段中。

当 CPU 需要执行访问操作时,首先通过比较器将给出的主存地址的主存字块标记字段与缓存块的标记字段进行比较,若两者相等且有效位为 1(该有效位用来表示 Cache 块中存储的数据是否有效,其值为 1 表示有效,为 0 表示无效)则命中;若两者不相等或有效位为 0 表明未命中,此时需要从主存中读入新的字块来替代旧的字块,同时将信息送往 CPU,并修改 Cache 的标记字段,若原来有效位为 0,还需要将其置为 1。

直接映射方式的优点是只需要利用主存地址的 t 位主存字块标记字段与缓存块的标记字段进行比较,就可以确定所需字块是否在缓存中,这加快了访问的速度。该方式的缺点是不够灵活,因为每个主存块只能固定地映射到某个缓存块中,即使其他缓存块都空着也不能与主存块建立映射关系。此外,如果程序恰好要重复访问对应同一缓存位置的不同主存块,那么就要不停地进行替换,这导致命中率较低。

2. 全相联映射

全相联映射是指主存的每一块可以映射到任一空闲的缓存块上,如图 3-45 所示。

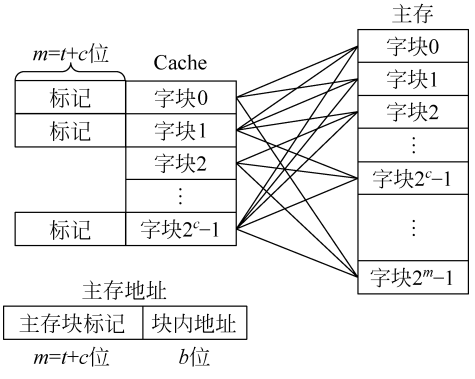


图 3-45　全相联映射

在全相联映射方式中,只要有空闲的缓存块就可以将主存块的信息存入其中,而且需要替换时,也可以选择任一缓存块进行替换。因此,这种映射方式是十分灵活的,命中率也更高。

由于这种映射方式没有任何规律性,因此主存的整个块号位($m=t+c$ 位)都要作为主存字块标记字段,这意味着 Cache 的标记字段也要由直接映射方式的 t 位增长至 m 位,故访问 Cache 时需要将 m 位的主存字块标记和 m 位的缓存标记字段逐位进行比较,才能判断要访问的内容是否在 Cache 中。这种比较通常由“按内容寻址”的相联存储器(将在后面介绍)完成。

3. 组相联映射

组相联映射结合了直接映射和全相联映射这两种方式,它先将所有的缓存块均分为 Q 组(通常建议 Q 为 2 的整数次幂),每组有 R 块(当 $R=2$ 时,称为二路组相联映射),然后采用直接映射的方式将主存块映射到缓存块组中去,即利用映射关系式

```
i = j mod Q
```

将主存块 j 按模 Q 映射到缓存的第 i 组内。再将分到同一缓存块组中的主存块组采用全相联映射的方式处理(该主存块组的任一主存块可以映射到该缓存块组的任一缓存块中),如图 3-46 所示。

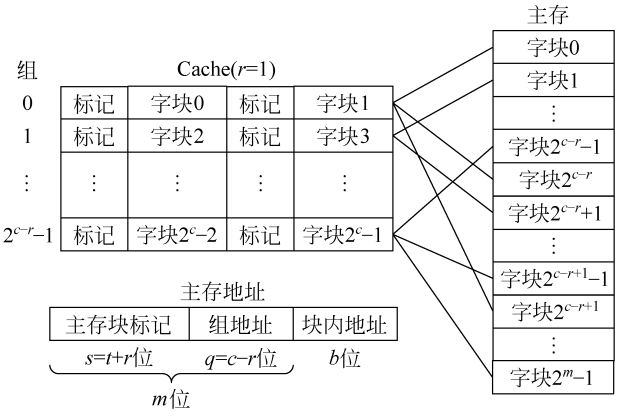


图 3-46 组相联映射方式

直接映射中 c 位的 Cache 块地址字段在组相联映射中变成了 q 位的组地址字段,且 $q=c-r$,其中,$2^c=C$ 表示缓存块数,$2^q=Q$ 表示缓存的分组个数,$2^r=R$ 表示每组包含的缓存块数。此外,主存字块标记字段由 t 位增长为 $s=t+r$ 位,故 Cache 的标记字段也要用 s 位表示。

【例 3-14】 假设主存容量为 512K×16 位,Cache 容量为 4096×16 位,块长为 4 个 16 位的字,访存地址为字地址。

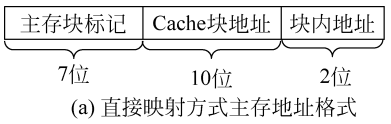
(1) 若采用直接映射,试设计主存的地址格式。

(2) 若采用全相联映射,试设计主存的地址格式。

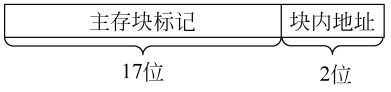
(3) 若采用二路组相联映射,试设计主存的地址格式。

解:

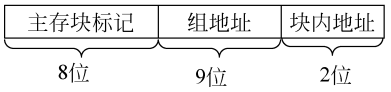
(1) 因为块长为 4($=2^2$)且访存地址为字地址,所以 Cache 和主存的块内地址均为 2 位,即 $b=2$。



(a) 直接映射方式主存地址格式



(b) 全相联映射方式主存地址格式



(c) 组相联映射方式主存地址格式

图 3-47 三种映射方式的主存地址格式

Cache 共有 $4096/4=2^{12}/2^2=2^{10}$,故缓存块号占 10 位,即 $c=10$。因为主存容量为 512K×16 位,512K$=2^{19}$ 所以主存字地址占 19 位,故主存字块标记占 19−10−2=7 位。主存的地址格式如图 3-47(a)所示。

(2) 采用全相联映射方式时,主存字块标记占 19−b=19−2=17 位。主存的地址格式如图 3-47(b)所示。

(3) 采用二路组相联映射方式时每组有 2 个缓存块,Cache 共有 1024 块,故分为 $1024/2=512=2^9$ 组,即 $q=9$,所以主存字块标记占 19−q−b=19−9−2=8 位。主存的地址格式如图 3-47(c)所示。

3.5.3 替换策略

当一个新的主存块需要调入 Cache，而该主存块映射到 Cache 中的所有缓存块却被其他主存块占满，此时需要选择一个缓存块将其替换，否则这一新的主存块将无法调入 Cache。

事实上，根据主存块到 Cache 的映射方式的不同，选择一个缓存块并将其替换的这一操作也会不同。在直接映射方式中，由于每个主存块固定映射到一个缓存块，故需要时可直接替换掉该缓存块；而在全相联和组相联映射方式中，由于每个主存块会映射到若干个缓存块，因此需要按照一定的策略（即替换算法）去选择某一缓存块进行替换。

常用的替换算法有先进先出（First In First Out，FIFO）算法、最近最少使用（Least Recently Used，LRU）算法、最不经常使用（Least Frequently Used，LFU）算法和随机算法等。

1. 先进先出（FIFO）算法

FIFO 算法通过记录字块调入缓存的起始时间或队列来实现替换最早调入 Cache 的字块。该算法不需要记录字块的使用情况，因此实现比较简单，开销也较小，但由于最早调入的字块可能会被经常使用（如循环程序），或者可能以后会被用到（程序的局部性原理），故 FIFO 算法不能提高命中率。

2. 最近最少使用（LRU）算法

LRU 算法利用了程序的局部性原理来选择近期最少使用的字块进行替换。该算法需要随时记录各字块的使用情况，实现该算法时为每一个缓存块设置一个计数器，Cache 每命中一次，就将对应缓存块的计数器清零，而将其他缓存块的计数器加一。当需要替换时，通过比较各计数器的值，将值最大的缓存块换出。LRU 算法的平均命中率比 FIFO 算法的高。

3. 最不经常使用（LFU）算法

LFU 算法选择近期使用次数最少的字块进行替换。通常实现该算法时也为每一个缓存块设置一个计数器，并从 0 开始计数，Cache 每命中一次，就将对应缓存块的计数器加一。当需要替换时，通过比较各计数器的值，将值最小的缓存块换出，并将其他缓存块的计数器清零。LFU 算法将计数周期限定在对特定行两次替换之间的间隔时间内，因此不能很好地反映近期访问情况，这使得其命中率要低于 LRU 算法。

4. 随机算法

随机算法是随机地确定替换的字块。通常实现该方法时需要设置一个随机数产生器，依据其产生的随机数确定应该被替换的字块。这种方法实现简单、速度较快，但因为随机换出的字块可能紧接着又要被使用（此时需要将该字块换入），这种情况将极大的降低命中率。

3.5.4 Cache 的一致性问题

Cache 中的内容是主存中一部分信息的副本，因此这些内容应当与主存中的内容保持一致。通常 CPU 访问 Cache 会执行读操作、写操作和读写操作，若当 CPU 访问 Cache 执行的仅为读操作时，由于该操作不会对 Cache 中的内容进行修改，因此执行这一操作时 Cache 中的内容始终与主存中的内容保持一致，而若包含写操作，就有可能会对其中的内容进行修改。因此，为了保持 Cache 中的内容与主存中的内容一致，可以使用以下三种方法（即写回法、写直达法和写一次法）。

1. 写回法

写回法(write-back),又称拷回法(copy-back),在正常情况下这一方法执行写操作时把数据写入 Cache,而不写入主存;而若出现将数据写入 Cache 失败的异常情况时,则需从主存中找出包含当前待修改数据的块,然后将其直接复制到 Cache 中,再执行写操作。对于上述情况,只有当 Cache 中的该数据要被替换出去时才将其写回主存。由于写回法有可能导致 Cache 中的数据与主存中的数据不一致,因此需要为 Cache 中的每一块设置一个标志位(dirty bit,即页面重写标志位,也称脏位,其初始时值为 0),该位用于标识 Cache 中的数据与主存中的数据是否一致。

实现这一方法时,需注意:

(1) 当对缓存中的某一块执行写操作时,先将该块写入 Cache 但不立即写入主存,同时将其标志位设置为 1。

(2) 当对其执行替换操作时,若其标志位为 1,则需先将其写回主存,再进行替换;否则直接进行替换。

2. 写直达法

写直达法(write-through),又称全写法,该方法在执行写操作时同时写入 Cache 和主存,因此在正常情况下可以保证 Cache 与主存中的内容一致,但这样增加了访问主存的次数,从而降低了 Cache 的工作效率。

若发生写入 Cache 失败而写入主存成功的异常情况,则需要考虑是否将修改过的主存块读到 Cache 中。此时既可以使用写分配法(Write Through with Write Allocate,WTWA)将修改过的主存块读到 Cache 中,并为它分配一个位置,也可以使用非写分配法(Write Through with NO Write Allocate,WTNWA)不将修改过的主存块读到 Cache 中,具体视情况而定。

通过在 Cache 和主存之间增加一个写缓冲,可以使得 CPU 在执行写操作时将数据同时写入到 Cache 和这一写缓冲中,然后再由写缓冲控制将该数据写入主存。这样可以减少全写法将数据直接写入主存的时间损耗,从而提高 Cache 的效率。事实上,写缓冲是一个 FIFO 队列,它可以解决 Cache 和主存之间速度不匹配的问题,但如果出现频繁的写操作,则可能会使其饱和溢出。为了解决这一问题,现代计算机通常设立多级 Cache(假定为二级,按离 CPU 的远近可命名为 L1 Cache 和 L2 Cache,L1 Cache 对 L2 Cache 使用全写法,L2 Cache 对主存使用写回法,由于 CPU 对 L2 Cache 的访问速度大于对主存的访问速度,这样就避免了因频繁写时造成的写缓冲饱和溢出)。

表 3-4 对写回法和写直达法进行了简单的比较。

表 3-4 两种方法的比较

性 能	方 法	
	写 回 法	写 直 达 法
一致性	可能无法随时保证 Cache 与主存中的内容一致	随时保证 Cache 与主存中的内容一致
复杂程度	实现复杂	实现简单
访存次数	减少了访存次数	增加了访存次数
工作效率	提高了 Cache 工作效率	降低了 Cache 工作效率

3. 写一次法

由于写回法和全写法各有优缺点,因此写一次法结合了这两种方法的优点,它在第一次执行写操作时,采用全写法(既写入 Cache 也写入主存),而在后续执行写操作时,则采用写

回法(只写入 Cache)。奔腾 CPU 的片内数据 Cache 就采用了这种方法。之所以写一次法在执行第一次写操作时采用全写法,是因为 CPU 要在总线上启动一个存储写周期,其他 Cache 监听到此主存块地址及写信号后,即可将该块从主存中拷贝至 Cache 中或将 Cache 中的该块作废,从而实现了全部 Cache 的一致性。

3.5.5　Cache 性能分析

Cache 系统的加速比和 Cache 的命中率是十分重要的两个性能参数,现介绍如下。

1. Cache 系统的加速比

在存储系统中,采用 Cache 技术的主要目的是提高存储器的访问速度,而加速比是其重要的性能参数,其定义如下,Cache 存储系统的加速比 S_P(speedup)为

$$S_P=\frac{t_m}{t_a}=\frac{t_m}{h\cdot t_c+(1-h)\cdot t_m}=\frac{1}{(1-h)+h\cdot\dfrac{t_c}{t_m}}=f\left(h,\frac{t_m}{t_c}\right)$$

其中:t_m 为主存储器的访问时间;t_c 为 Cache 的访问时间;t_a 则为 Cache 存储系统的平均访问时间;h 为命中率。

从加速比的定义可以看出,加速比的大小与两个因素有关:即命中率 h 及 Cache 与主存访问周期的比值 t_c/t_m。图 3-48 所示为加速比 S_P 与命中率 h 的关系,从图中可以看出,命中率 h 越高,加速比 S_P 越大。当命中率 $h=1$ 时,加速比 S_P 的值为

$$S_P=\frac{1}{(1-1)+1\cdot\dfrac{t_c}{t_m}}=\frac{t_m}{t_c}$$

此时 S_P 取得最大值,即为 S_{pmax}。

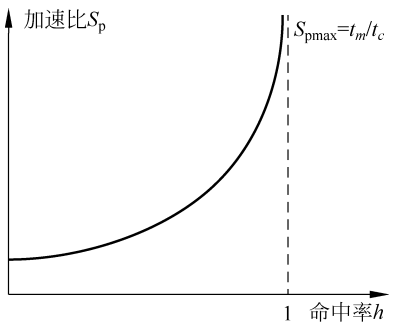


图 3-48　加速比 S_p 与命中率 h 的关系

2. Cache 的命中率

从 CPU 的角度来看,通过增加 Cache,可能使得 CPU 从主存中读取数据的平均时间接近于从 Cache 中读取数据的时间。由于 Cache 的速度比主存的速度快很多,这样就可以提高 CPU 的利用率。为了实现这个目标,则需要使得在所有的存储器访问中,由 Cache 中的数据满足 CPU 计算需要的部分应尽可能占更高的比例,而由主存中的数据满足 CPU 计算需要的部分应尽可能占更低的比例,即 Cache 的命中率应无限接近于 1(根据程序局部性原理,这是有可能实现的)。

影响 Cache 命中率的因素很多,如 Cache 的容量、块的大小、映射方式、替换策略以及程序执行中地址流的分布情况等。接下来从 Cache 的容量、块的大小、映射方式这三个方面来分别介绍它们对命中率的影响情况。

1) Cache 的容量对命中率的影响

Cache 的容量越大则命中率越高,但当其容量达到一定程度后,命中率则不会继续显著提高。

2) Cache 块的大小对命中率的影响

若增大 Cache 块,命中率也明显增加,但增加到一定程度之后反而会下降。

3) Cache 的映射方式对命中率的影响

对于常用的三种映射方式而言,直接映射方式命中率比较低,全相联映射方式命中率比

较高，组相联映射方式则会因为组分得越多，从而导致命中率越低。

【例 3-15】 某计算机系统的内存储器由 Cache 和主存构成，Cache 的存取周期为 20ns，主存的存取周期为 100ns，已知在一段给定的时间内，CPU 共访问内存 2000 次，其中 140 次访问主存。

(1) 求 Cache-主存系统的命中率、平均访问时间和加速比。

(2) 若 Cache 和主存的存取周期不变，而命中率变为 0.96，此时 Cache-主存系统的加速比有何变化？

解：

(1) Cache 的命中率

$$h=\frac{N_c}{N_c+N_m}=\frac{2000-140}{2000}=0.93$$

Cache 的平均访问时间为

$$\begin{aligned}t_a&=h\times t_c+(1-h)\times t_m\\&=0.93\times 20\text{ns}+(1-0.93)\times 100\text{ns}=25.6\text{ns}\end{aligned}$$

Cache 的加速比为

$$S_p=\frac{t_m}{t_a}\times 100\%=\frac{20\text{ns}}{25.6\text{ns}}\times 100\%=78.1\%$$

(2) 当命中率为 0.96，Cache 的等效访问周期为

$$\begin{aligned}t_a&=h\times t_c+(1-h)\times t_m\\&=0.96\times 20\text{ns}+(1-0.96)\times 100\text{ns}=23.2\text{ns}\end{aligned}$$

Cache 的加速比为

$$S_p=\frac{t_m}{t_a}\times 100\%=\frac{20\text{ns}}{23.2\text{ns}}\times 100\%=86.2\%>78.1\%$$

从这个例子可以看出，当命中率由 0.93 变为 0.96，Cache 的加速比则由 78.1%上升至 86.2%。这充分印证了命中率与加速比之间的关系。

3.5.6 相联存储器

相联存储器是由检索寄存器、屏蔽寄存器、存储体、代码寄存器、符合寄存器等组成，如图 3-49 所示。

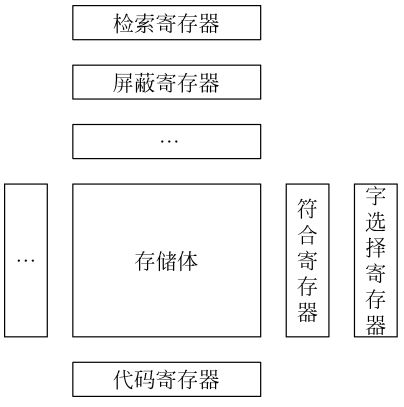


图 3-49 相联存储器的组成框图

现将相联存储器的各组成部件介绍如下。

(1) 检索寄存器(Common Register，CR)是用来存放检索字的，其位数与相联存储器的存储单元位数相等。每次检索时，取检索寄存器中若干位为检索项。

(2) 屏蔽寄存器(Mask Register，MR)是用来存放屏蔽码的，其位数与检索寄存器的位数相同，且内容与需要检索的字段有关。

(3) 符合寄存器(Results Register，RR)又称查找结果寄存器，它用来存放按检索项内容检索存储体中与之符合的单元地址，其位数等于相联存储器的存储

单元数，每一位对应一个存储单元。

(4) 字选择寄存器(Word Select Register，WSR)是用来确定哪些存储单元参与检索的。其位数与存储器的存储单元数相等。

相联存储器的每个字由若干个字段组成，每个字段描述了一个对象的属性(也称为一个内容)。它有三种基本操作：分别为读、写和检索，每次检索时将所有存储字的相关字段与待检索项同时进行比较，这与在 Cache 中将主存字块的标记同时与每个缓存字块的标记进行比较类似。相联存储器还可以进行各种比较运算(如大于、小于、等于、求最值等)和逻辑运算，这表明它不仅能储存，还能进行逻辑运算，所以它也被称为分布逻辑存储器。由于其功能强大，故电路比一般存储器复杂得多，进而导致价格不菲。

如图 3-50 所示的相联存储器，它以 4×4 的矩阵作为存储器，接下来基于该相联存储器来阐述其检索过程。

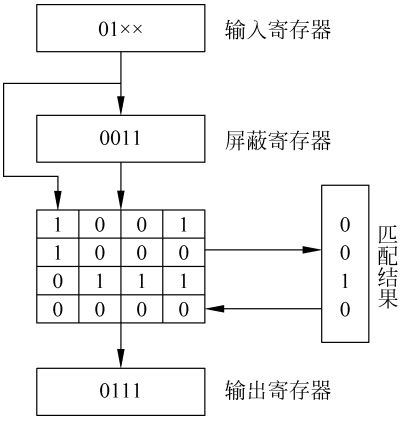


图 3-50　4×4 矩阵的相联存储器

假定待查找数据的高两位是 01，开始检索时先将该数据放在输入寄存器中，然后将屏蔽寄存器的高两位清 0，由于不需要查找该数据的低两位，故将屏蔽寄存器的低两位置 1，此时屏蔽寄存器的内容为 0011。通过将屏蔽寄存器的高两位在矩阵的高两列同时进行查找，发现只有第三行数据满足要求，所以就选择该行数据并将其放到输出寄存器中。

但如果采用一般存储器来查找该数据，则需要进行逐行匹配，即无法像相联存储器那样同时实现对两列进行查找，这就意味着当一般存储器的容量较大时，其查找过程需要耗费的时间可能要比相联存储器长很多。

由于相联存储器的逻辑电路十分复杂，从而导致其成本极高，因此早期并未得到广泛使用。随着大规模集成电路的快速发展，使得相联存储器的成本大大降低，因此已经被广泛应用于语音识别、图像处理、数据流计算机等领域。

注意：相联存储器既可以按地址寻址，又可以按内容寻址。若按内容寻址时，该内容通常是某些字段(亦称关键字)。由于传统存储器都是按地址寻址的，因此为了与之相区别，相联存储器又被称为按内容寻址的存储器。

3.6　虚拟存储器

在程序运行时，根据局部性原理，可以先将程序的一部分装入内存并运行，若该程序运行时所需访问的信息不在内存中，则可由操作系统将该信息调入内存，从而保证程序继续正常运行，这样相当于操作系统为用户提供了一个比实际物理内存(简称实存)大得多的存储器。由于这种存储器是操作系统通过综合使用部分装入、请求调页和页面置换等技术形成的，实际上它并不存在，故称其为虚拟存储器(简称虚存)。

虚拟存储器通过将计算机的内存和硬盘上的部分甚至全部临时空间相组合，从而使得应用程序认为自身拥有连续可用的内存(即为连续完整的地址空间)，但实际上该内存通常

是由被分隔成多个物理内存碎片和部分外部磁盘存储器组成(存放在外存上的数据仅在程序运行时需要才调入内存)。

在计算机系统中,虚拟存储器是一种极为常用的内存管理技术,目前大多数操作系统都使用了这一存储器,如 Windows 家族的“虚拟内存”和 Linux 的“交换空间”。

3.6.1 虚拟存储器简介

虚拟存储器的发展大致经历了两个阶段。

在第一阶段,由于使用单用户单任务的操作系统,即每一台计算机只有一个用户,且每次只能运行一个程序,此时单个程序所占用的内存空间通常不是很大,完全可以存放在实存中,因此虚拟存储器没有什么用处,故而未受到工程技术人员的重视。

但随后虚拟存储器的发展进入到了第二个阶段,这是因为:一方面,随着程序不断增大,运行时无法一次性载入到计算机系统实际的物理内存中;另一方面,在多用户多任务操作系统中,由于多个用户或任务共享全部内存,并要求同时执行多道程序,因此在编制程序时无法事先确定这些多道程序到底占用实际内存的哪一部分,必须等到运行时才由操作系统将实际的物理内存动态地分配给它们。

在这一阶段,程序员编制程序时既不考虑该程序是否能一次性载入到实存中,也不考虑该程序应该存放在什么物理位置,而是在程序运行时由操作系统分配给每个程序一定的空间,再由地址转换部件(硬件或软件)将编程时的地址转换成物理内存的地址。若某一程序运行时发现事先分配给其的内存不够,则只调入正在运行的或将要运行的程序块(即部分载入)。

通常将上述用户编制程序时使用的地址称为虚地址(逻辑地址),其对应的存储空间称为虚拟存储空间(逻辑地址空间,也称虚地址空间);而计算机物理内存的访问地址则称为实地址(物理地址),其对应的存储空间称为实际存储空间(物理地址空间,也称实地址空间)。

由于虚拟存储器是建立在主存空间(主存)和辅存空间(辅存)的基础之上,并由相关的硬件及操作系统等软件组成的一种存储体系。它通过将主存和辅存的地址空间统一编址,形成一个庞大的存储空间,在这个空间里,用户可以自由编程,完全不必考虑程序在主存中是否放得下或者放在什么位置等问题。如图 3-51 所示为典型的虚拟存储器系统硬件部分的结构。

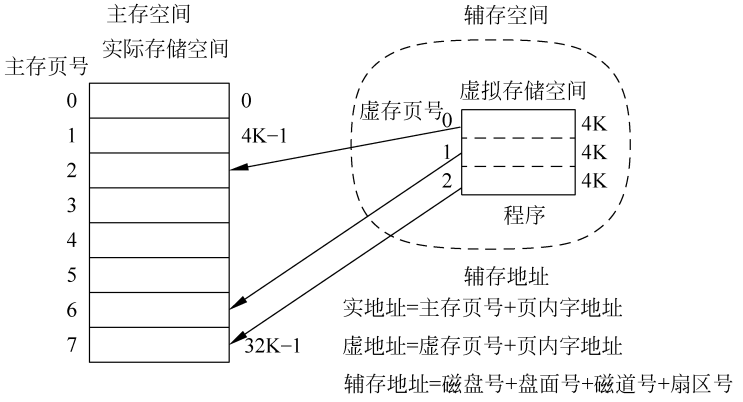


图 3-51 虚拟存储器系统硬件部分的结构

在使用虚拟存储器之后，程序将按如下方式运行。首先将程序存放在辅存（例如磁盘或磁带）中。然后由地址变换机构依据当时分配给该程序的实际存储空间把程序的一部分调入主存。若程序运行时涉及的代码或数据均在主存中，则进行地址转换并用实地址访问主存；否则需按照某种算法将存储在辅存中的代码或数据调入主存后再访问。

这意味着，每个程序的虚拟存储空间与计算机的实际存储空间是相对独立的。若某一程序运行时所需的实际存储空间较大，以至于运行该程序的计算机无法提供，此时则需要使用虚拟存储空间为该程序的运行提供帮助。而对于多用户或多任务系统，其主存空间通常较大（否则无法满足多用户使用或多任务运行），而单个用户运行的程序或单个任务在运行时可能不需要占用很大的地址空间，这样通过使用较小的虚存空间则可以缩短指令中地址字段的长度。

通过使用虚拟存储器，应用程序就可以透明地使用整个虚存空间。而对于应用程序而言，如果主存的命中率很高，则虚拟存储器的访问时间就可能接近于主存的访问时间，而虚拟存储器的空间最大则可能接近于辅存的大小。这样，虚拟存储器就可以具有辅存的容量和接近主存的访问速度。

当然，在使用虚拟存储器时也需要面对以下问题：

（1）调度问题：即在程序运行的过程中，需要有相应的调度算法来实时决策将哪些代码和数据调入主存。

（2）地址映射问题：包括内地址变换（即将虚地址变为实地址）和外地址变换（即将实地址变为虚地址）。

（3）替换问题：即在程序运行时需要将哪些代码或数据从主存中调出，从而使得当前程序需要的代码或数据能被调入。

（4）数据一致性问题：即将辅存上的数据调入主存后，及将主存中的数据调出至辅存上时，如何保证两者数据的一致性。

接下来将详细介绍几种典型的虚拟存储器：页式虚拟存储器、段式虚拟存储器、段页式虚拟存储器和快表。

3.6.2 页式虚拟存储器

页式虚拟存储器以页为基本单位，它将虚拟储存空间划分成同样大小的页，即称为虚页（逻辑页），同时也将实际存储空间划分成同样大小的页，即称为实页（物理页）。

在页式虚拟存储器中，通常把虚地址（逻辑地址）分为两个部分：虚页号和页内地址，同时把实地址（物理地址）也分为两个部分：实页号和页内地址。由于两者页面的大小一样，所以页内地址也是相同的。因此虚地址到实地址的转换就变成了将虚页号转换为实页号。

由于通常每一程序对应一张页表，而每一页表项的内容包含与某个虚页对应的虚页号、实页号和用来判断该页面是否在主存中的装入位等，因此在页表里记录了程序中的虚页调入主存时的位置，该位置即为其在主存中的地址，故将虚页号转换为实页号过程由页表来实现的。

图 3-52 所示为页式虚拟存储器的地址转换过程。

其中的页表基址寄存器中存放的是当前运行程序所对应页表的起始地址，将它和虚页号拼接可得到根据虚地址访存时所需的页表中的对应项（即页表项）的地址，再根据页表项

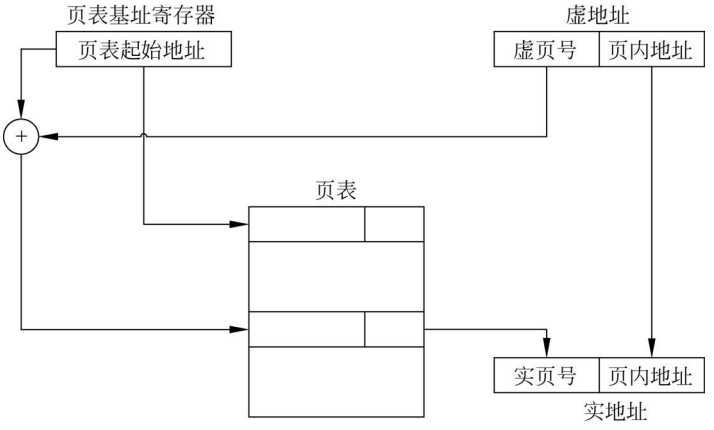


图 3-52　页式虚拟存储器的地址转换过程

的装入位来判断该页是否已调入主存。若已调入，则根据页表项的地址在页表中找到对应的实页号，并将其与虚地址中的页内地址部分拼接，从而得到完整的实地址，此时仅在查页表时需要访问一次主存；否则需要将该页从辅存中调入主存，并且还要通过多次访问主存进行页面替换（即发现所要访问的页面不在主存中，则产生缺页中断，此时若主存中没有空闲页面，则必须在主存中选择一个页面并将其移出主存，以便为即将调入的页面腾出空间）和页面修改。

【例 3-16】 在只有一个用户的计算机系统中，假如一个进程的虚地址空间为 128MB，每页的大小为 4KB。

（1）试问该进程的虚页数（即为虚地址空间除以页面大小所得的值）为多少？若这一进程的虚地址空间增大至 16GB，其虚页数将会如何？

（2）若该计算机系统同时运行了 100 个用户进程，每个进程的虚地址空间均为 16GB，且每个页表项在页表里占用 4B，此时页表需占用多少空间？假定该计算机系统的主存为 4GB，而操作系统需占用 3GB，此时主存中还剩余多少可用空间？

解：

（1）当虚地址空间为 128MB 时，其虚页数为

$$128\text{MB}/4\text{KB}=2^{17}/2^{12}=2^{5}$$

而当虚地址空间增大至 16GB 时，其虚页数为

$$16\text{GB}/4\text{KB}=2^{33}/2^{12}=2^{21}$$

由此可见，当虚存空间很大时，页表会特别长。

（2）对于一个进程来说，页表占用的空间为

$$2^{21}\times 4\text{B}=2^{23}\text{B}=8\text{MB}$$

而对于 100 个用户进程，页表占用的总空间为

$$8\text{MB}\times 100=800\text{MB}=0.78\text{GB}$$

此时主存中剩余的可用空间为

$$3\text{GB}-1\text{GB}-0.78\text{GB}=0.22\text{GB}$$

由上题可知，在题设给定条件下，主存中剩余的可用空间远远小于页表占用的空间，故导致用户的绝大部分空间都被页表占用。若该计算机系统有多个用户使用，主存空间则无

法存放所有页表。

因此在实际使用页式虚拟存储器时，不同的系统处理方式并不完全相同。一些系统为了节省主存空间，通常不将页表全部存放在主存中(即仅将当前运行的进程所对应的页表存放在其中)，而是把当前未运行进程所对应的页表存放在虚存中，这使得页表本身也要进行分页，即将页表的一部分存放在主存中，另一部分存放在辅存中；而另一些系统则会采用多级(二级及以上)页表结构来解决这一问题，接下来以二级页表结构为例进行介绍，通常也需先将页表进行分页，并为每个进程增加一个页目录表(即一级页表)，它的每个表项指向一个页表(即二级页表)。因此对于使用二级页表结构的系统来说，若一个进程的页目录表的长度(表项数)为 s，而其页表的长度(表项数)为 t，则最多可以有 $s \times t$ 页。

若在计算机系统中的实地址空间远远小于虚地址空间，则会导致实页数(即为实地址空间除以页面大小所得的值)远远小于虚页数。对于这种情况，可以采用反向页表(inverted page table)来完成实页号到虚页号的反向映射，在反向页表中，每一页表项对应某一实页，表项的内容包含该实页所对应的虚页号。

若使用反向页表访问主存，可先根据虚页号在反向页表中逐一查找，如果找到匹配的表项，则将该项对应的实页号与虚地址中的页内地址拼接，即可得到实地址；如果没有匹配的表项，则说明该页不在主存中。

使用反向页表的好处是大大缩小了页表所占的空间，但其不足之处为检索耗费的时间很长。

页式虚拟存储器的优点是页面的长度固定，页表简单，调入方便；缺点是因为程序不可能都是页面的整数倍，从而导致最后一页的零头通常将无法被使用，进而造成存储空间的浪费，并且由于页不是逻辑上独立的实体，这容易使得一个程序被迫拆分成两部分，一部分可能在主存中，而另一部分则可能在辅存中。因此，页式虚拟存储器在处理、保护和共享方面都不及段式虚拟存储器方便。

3.6.3 段式虚拟存储器

由于页式虚拟存储器可能会造成存储空间的浪费、还可能破坏编程的独立性，并给换入换出处理、存储保护和共享等操作来造成不便，因此可以使用分段的办法(即段式虚拟存储器)来解决上述问题。通过事先划分好不同长度的段，程序员可把子程序、操作数和常数等不同类型的数据划分到相应的段中，使得每段对应一个具有完整逻辑意义的程序模块。

在段式虚拟存储器中，虚地址分为两部分：段号和段内地址。段是按程序的逻辑结构划分的，每个段的长度因程序而异，并且每个程序可以有多个相同类型的段。通常为每个程序设置一个段表，其中每一段表项至少包含以下字段：

(1) 段号：用来标识每一段。

(2) 装入位：用来判断该项所对应的段是否已经调入主存。

(3) 段的起始地址：是指该项所对应的段调入主存后在主存中的首地址。

(4) 段长：是指该项所对应的段的实际长度。

与页式虚拟存储器中的页表相比，段式虚拟存储器中的段表增加了两个字段(即段的起始地址和段长)，这是因为段的长度是可变的。由于段表是程序的逻辑段与在主存中存放位置的对照表，故虚地址到实地址的转换由段表来实现的。

在段式虚拟存储器中，根据虚地址访存时，其地址转换过程如图 3-53 所示。

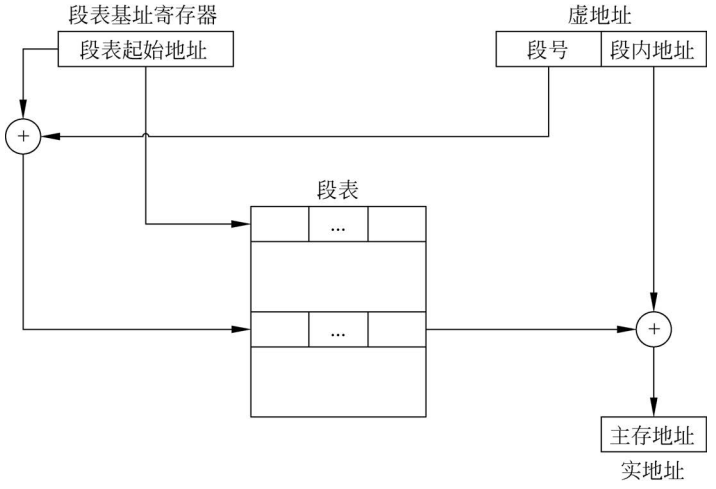


图 3-53　段式虚拟存储器的地址转换过程

首先将段号与段表起始地址拼接成段表项地址，通过该地址可以找到相应的段表项，再根据此表项中的装入位字段判断该段是否已经被调入主存。

(1) 如果已调入主存，则还需将虚地址中的段内地址部分与表项中的段长字段进行比较。若段内地址小于段长，则从相应的表项中读出该段的起始地址，再将其与虚地址中的段内地址部分相加，从而得到实地址；否则将会产生地址越界中断。

(2) 如果未调入主存，则需要将其从辅存中调入。

【例 3-17】 假设某一计算机系统采用段式虚拟存储器，为了将一个作业装入主存，建立的段表如表 3-5 所示。

表 3-5　某计算机系统采用段式虚拟存储器的段表

段　　号	装　入　位	段的起始地址	段　　长
0	0	2525	250
1	1	1800	360
2	1	3000	280
3	1	1200	500
4	1	85	860

请问通过[0,10]、[2,300]、[4,800](方括号中的第一个元素为段号，第二个为段内地址)这三个虚地址是否能正常访问主存？若能，请写出相应的主存地址；否则请解释原因。

解：根据虚地址访存时，需先通过其中的段号找到对应的表项。

对于第一个虚地址，因为对应表项中的装入位为 0(说明该段未调入主存)，所以暂时无法访问。

对于第二个虚地址，虽然对应表项中的装入位为 1(说明该段已调入主存)，但段长(即 280)小于虚地址中的段内地址(即 300)，故因地址越界产生中断，从而无法进行访问。

对于第三个虚地址，因为对应表项中的装入位为 1，且段长(即 860)大于虚地址中的段内地址(即 800)，故能够正常访问主存，且主存地址为 85＋800＝885。

与页式虚拟存储器相比，段式虚拟存储器有许多优点：①对程序员而言，分页是不可见的，但分段是可见的，这为程序员组织程序和数据提供了方便；②由于段具有逻辑独立性，

这使其易于编译、管理、修改和保护，也便于多道程序共享；③段长可以根据需要动态改变，且允许自由调度，这样消除了内存零头，从而更好地利用主存空间。

正是因为段长不固定，因此与页式虚拟存储器相比，段式虚拟存储器存在以下缺点：①主存空间分配比较麻烦；②容易在段与段之间留下许多碎片空间，从而造成存储空间的利用率较低；③在进行地址转换时需要更多的硬件支持（因为要对段的起始地址与段内地址进行求和才能得到实地址）。

表 3-6 为页式虚拟存储器和段式虚拟存储器的比较。

表 3-6　两种虚拟存储器的比较

	页式虚拟存储器	段式虚拟存储器
划分标准	页面长度固定且仅根据页面长度进行划分	段长不固定且是按程序的逻辑结构划分的
空间分配	空间分配比较简单且不会在页间留下碎片	空间分配比较麻烦且容易在段间留下碎片
空间使用	最后一页的零头通常无法被使用，会造成存储空间的浪费	每段对应一个具有完整逻辑意义的程序模块，故消除了内存零头
共享和保护	共享和保护不方便	共享和保护方便

3.6.4　段页式虚拟存储器

因为段式虚拟存储器和页式虚拟存储器各有优缺点，所以段页式虚拟存储器将两者结合起来。对于实存，将其分为同样大小的页；对于虚存，则先将程序按逻辑结构分段，再将每段按实存中的页面大小进行划分。程序仍以页为基本单位进行调入调出，但可按段进行编程、保护和共享。

在段页式虚拟存储器中，每个程序均通过一个段表和一组页表进行二级再定位。段表中的每个表项对应一个段，而每段又对应一个页表，且页表里记录了该段中的每一页在主存中的位置，是否已经装入或已经修改等信息。根据段和页的关系可知，段的长度必须是页长的整数倍，且起点必须是某一页的起点。

段页式虚拟存储器中的虚地址包括段号、段内页号和页内地址这三部分，实地址包括实页号和页内地址这两部分。段页式虚拟存储器的地址转换过程如图 3-54 所示。

在根据虚地址访存时，首先要将段号与段表起始地址拼接成对应的段表项地址，然后从段表中取出该段的页表起始地址，再将其与虚地址中的段内页号拼接得到页表项地址，最后从相应的页表中取出实页号，并将其与虚地址中的页内地址拼接，便可得到主存实地址。

注意：在多任务系统中，操作系统还会在每个虚地址前面增加一个称为基号的字段，用它来表明该程序在系统中的序号，那么在此情况之下，则需根据基号在基址寄存器中找到相应的段表起始地址。

【例 3-18】 在某一使用段页式虚拟存储器的计算机系统中，假定页的大小为 2KB，每个段的页表有 8 个表项，设某任务恰好被分成 4 个大小相等的段。试求每个段的最大长度和该任务的最大逻辑地空间。

解：因为每个段的页表有 8 个表项，所以每个段最多有 8 页，故每个段的最大长度为 8×2KB＝16KB。

又因为任务恰好被分成 4 个大小相等的段，所以该任务的最大逻辑地址空间为 4×16KB＝64KB。

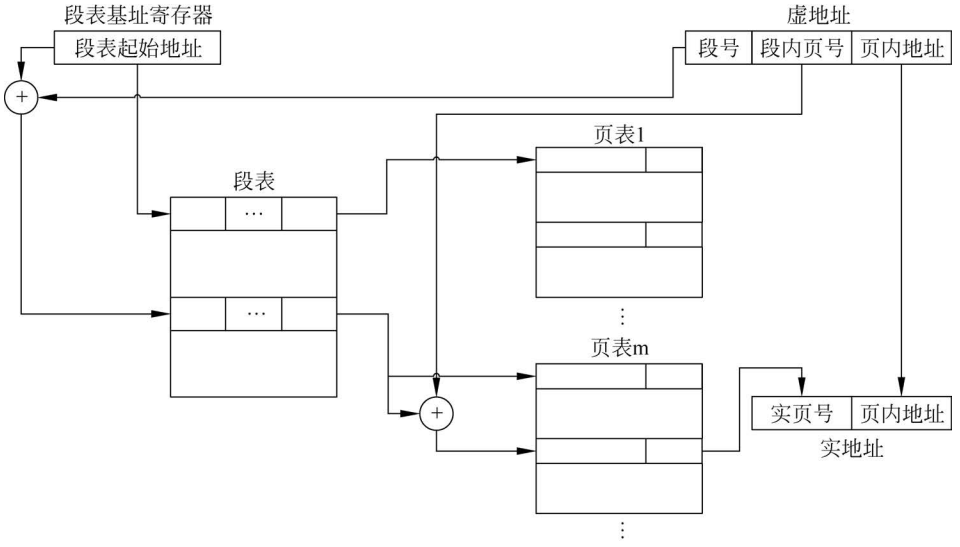


图 3-54　段页式虚拟存储器的地址转换过程

段页式虚拟存储器的优势在于它同时具有页式和段式虚拟存储器的优点，而不足之处为地址变换过程需要多次查表，从而导致系统复杂性较高，开销较大。

3.6.5　快表

在页式虚拟存储器中，即使虚页已被调入，也必须要先对主存中的页表进行一次访问以获取页面的物理地址，然后再根据该地址访问页面数据，这会使得存取时间翻倍。

因此，依据程序局部性原理，为了减少访存的时间延迟，对于那些在一段时间内经常访问的页，可以把页表中与它们对应的页表项置于由高速缓存组成的快表（Translation Lookaside Buffer，TLB，也称转换后援缓冲器）中，这样可以加快地址变换。此时，通常也把存放在主存中完整的页表称为慢表（Page）。

快表的容量比慢表小得多，且只存放了慢表中很少一部分的页表项，它的作用与 Cache 的作用十分类似，通常由相联存储器来实现，使用快表后，若根据虚地址访问主存，其地址转换过程如图 3-55 所示。

在使用 TLB 进行地址转换时，通常将根据虚地址的虚页号同时查找快表和慢表。若快表命中，则能很快地找到相应的实页号，再将其送入实地址寄存器，并取消慢表查找，从而导致虽采用虚拟存储器但访存速度几乎没有下降；若快表不命中，则在慢表中进行查找，若慢表也不命中，则还需要执行页面调度。

经过上述地址转换过程得到的物理地址（实地址）不一定是最终地址（Cache 中的地址），具体原因如下。

（1）如果 Cache 命中，还需要将其再转换成 Cache 地址，才能得到最终地址。这一完整的转换过程是从逻辑地址（虚拟地址）开始，中间经历了物理地址，再到最终地址。

（2）如果 Cache 不命中，则需要通过访问 TLB 或 Page 得到的地址直接去访问主存，可见在这种情况下完整的转换过程即是从逻辑地址转换为物理地址。

【例 3-19】 已知某计算机系统采用页式虚拟存储管理，页表存放在主存中，假设一次

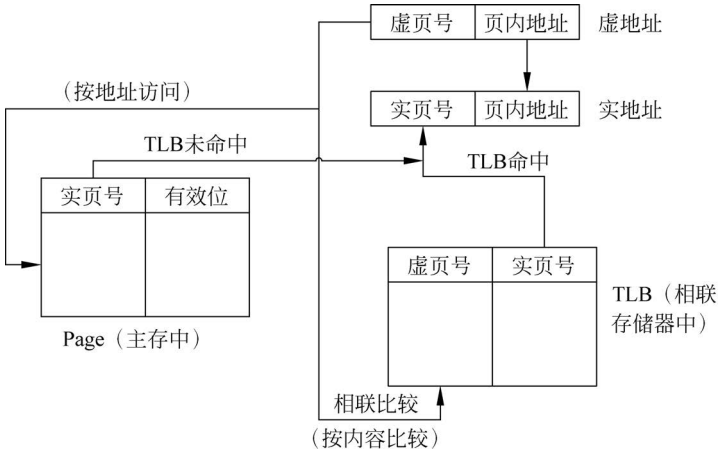


图 3-55 TLB 的地址转换过程

内存访问需要 10ns。现欲通过增加 TLB 来降低访存的时间延迟，假定忽略查找 TLB 占用的时间，并且 80％的页表访问命中 TLB，试计算内存的有效访问时间是多少？

解：由于增加了 TLB，故先在其中查找，若未找到，才会访问页表。由于页表存放在主存中，所以要实现访问页面数据需要访问两次主存，第一次是访问页表获取页面的物理地址，第二次才是根据该地址访问页面数据。故有效访问时间计算为

$$80\%\times10+(1-80\%)\times2\times10=12(\text{ns})$$

注意：在同时具有虚拟页式存储器（包含快表）和 Cache 的系统中，当 CPU 发出访存命令时，查找数据最好和最坏的情况如下。

（1）最好的情况是：虚地址通过快表得到相应的实地址（快表命中），且在 Cache 中可找到正确的数据（即 Cache 命中）。

（2）最坏的情况是：虚地址通过快表和慢表都无法得到相应的实地址（快表和慢表均不命中），且在 Cache 中也找不到所需的数据（即 Cache 不命中）。

此外，将 Cache 是否命中与 Page 和 TLB 命中的关系简要说明如下。若 Cache 命中，则说明所需页面已调入主存，Page 必然命中，但 TLB 不一定命中；若 Cache 不命中，则并不能说明所需页面未调入主存，这和 TLB 与 Page 是否命中也没有联系。

3.6.6 虚拟存储器的替换算法

在虚存中，当 CPU 要用到的数据或指令不在主存时，则需进行页面替换，此时要用相应的算法来实现。虚存中的页面替换算法与 Cache 中的页面替换算法极为类似，接下来仅介绍两者的不同之处。

（1）Cache 的替换全部依靠硬件实现，而虚存的替换则需要操作系统的支持。

（2）从对系统性能的影响程度来看，虚存缺页比 Cache 未命中时要大得多。这是因为虚存中调页需要访问辅存，并且要进行任务切换。

（3）在进行页面替换时，对虚存而言只要属于一个进程的页面都可以被替换。

对于虚存中的页面替换算法，通常需要在每一页表项设置一个修改位，用它来标识该页表项所对应的主存中的页是否被修改过。若是，则不必进行处理，否则就应该把该页重新写

入辅存，以保证辅存中数据的正确性。

【例 3-20】 假设主存只允许存放 a[0]、a[1]、a[2]三个页面，某次操作中进程访存的序列依次分别是 0、3、1、3、2、3、0、4、3、1(虚页号)，请用列表法分别求出采用 FIFO 算法和 LRU 算法作为替换策略时主存的命中率。

解：两种替换策略下主存替换过程和命中率如表 3-7 所示。

表 3-7 两种替换策略下主存替换过程和命中率

页面访问序列		0	3	1	3	2	3	0	4	3	1	命中率
FIFO 算法	a[0]	0	3	1	1	2	2	0	4	3	1	2/10=20%
	a[1]		0	3	3	1	1	2	0	4	3	
	a[2]			0	0	3	3	1	2	0	4	
					命中		命中					
LRU 算法	a[0]	0	3	1	3	2	3	0	4	3	1	3/10=30%
	a[1]		0	3	1	3	2	3	0	4	3	
	a[2]			0	0	1	1	2	3	0	4	
					命中		命中			命中		

在采用 FIFO 算法作为替换策略时，新调入的页面始终按先进先出的顺序依次占用 a[0]、a[1]和 a[2]的空间；而在 LRU 算法中，仅在页面初始化时(即 a[0]、a[1]或 a[2]为空时)依次占用 a[0]、a[1]、a[2]，后续调入新页面则是将最近最少使用的页面换出。

3.7 外部存储器

外部存储器，也称辅助存储器(简称外存或辅存)，它与主存共同组成了存储器系统的主存-辅存层次，目前被广泛使用的辅存有机械硬盘、固态硬盘、U 盘和光盘等。

3.7.1 外部存储器简介

外部存储器是指通过外部设备接口与 CPU 连接的存储设备，一般它们在断电后仍然可以保存数据。对于外部存储器，通常可以将它们大致分为三大类：磁表面存储器(包括磁盘存储器、磁带存储器和磁鼓存储器等)、闪速存储器(包括 U 盘和固态硬盘等)和光存储器(包括 CD-ROM、DVD-ROM、CD-R、CD-RW 等)。

1. 磁表面存储器

磁表面存储器是将某些磁性材料涂在不同形状(如盘状、带状等)的载体(如金属铝、塑料等)表面。它们记录信息的原理如下：即在这些涂有磁性材料的载体的高速运动时，由磁头在磁层上进行读/写操作，信息此时被记录在磁层上，而这些信息的轨迹就是磁道。

磁表面存储器有五个主要的技术指标，分别为记录密度、存储容量、平均寻址时间、数据传输率和误码率，简要介绍如下。

1) 记录密度

记录密度通常是指单位长度内所存储的二进制信息量。通常，磁盘存储器用道密度(指磁盘沿半径方向单位长度内的磁道数)、位密度(指单位长度磁道内所存储的二进制信息的位数)和面密度(指位密度和道密度的乘积)表示，而磁带存储器用位密度表示。

道密度的单位是道每英寸(Track Per Inch，TPI)或道每毫米(Track Per Millimeter，TPM)；

面密度的单位是位/平方英寸(Bits Per Square inch,BPSI);位密度的单位是位每英寸(Bits Per Inch,BPI)或位每毫米(Bits Per Millimeter,BPM)。

磁盘上的各磁道所记录的信息量是相同的,但长度不同,所以位密度也不同。通常最内圈磁道的位密度最大,它用作泛指磁盘位密度。

2) 存储容量

存储容量是指外存所能存储的二进制信息总量,一般以位或字节为单位。存储容量分为格式化容量和非格式化容量。非格式化容量是磁表面可以利用的磁化单元总数;格式化容量是指按某种标定的记录格式所能存储信息的总量,即用户可以使用的容量。

3) 平均寻址时间

磁盘采取的是直接存取方式,寻址时间分为寻道时间和等待时间两个部分。寻道时间是指磁头寻找目标磁道的时间,等待时间是指找到磁道后,磁头等待欲读/写的磁道扇区(即对磁盘的磁道进行等分后得到的弧段)旋转到磁头下方所需要时间。磁带采取的是顺序存取方式,磁头不动,磁带移动,故只要考虑磁头寻找记录区段的等待时间。

4) 数据传输率

数据传输率是指单位时间内磁表面存储器向主机传送数据的位数或字节数。

5) 误码率

误码率是用来衡量从磁表面存储器中读取信息时的出错概率,它被定义为出错的信息位数与总共读出的位数之比。

2. 闪速存储器

闪速存储器是一种非易失性(Non-Volatile)存储器,它属于存储器件的一种。

闪速存储器的优势在于它具有较快的读取速度,其读取时间小于100ns。与硬盘相比,闪存的动态抗震能力更强,因此它非常适合用在移动设备上,例如笔记本电脑、数码相机和智能手机等,当前十分流行的U盘就是闪存的一个典型应用。

但是闪速存储器在进行擦除操作时有两个限制:一是闪速存储器擦除已保存的数据不是以单个的字节为单位,而必须是以一个区块为单位进行,也就是说,闪存支持随机读取和写入,但是不允许随机改写;二是闪速存储器的擦除次数有有限,从10000次到1百万次不等,尽管可以通过一些技术延长其使用寿命,但是这个限制也导致了闪存不适用于大量数据读写循环的高可靠性数据存储应用。

3. 光存储器

光存储器是指用光学方法从光存储媒体上读取和存储数据的一种设备。由于该设备通常使用半导体激光器为光源,所以也之称为激光存储器。用于计算机系统的光存储器主要是光盘(Optical Disk),现在通常称为CD(Compact Disk),光盘存储技术将磁带的大存储容量和磁盘的快速随机检索结合在一起,并具有离线存储等一系列独特的优良性能,因此一度被认为是一种重要的数据存储技术。

光盘的优点是记录密度高,存储容量大,长期保存寿命长,价格低;而缺点是存取时间较长(光盘平均存取时间在100ms以上,而磁盘存取时间在10ms以下),数据传输率低。

3.7.2 磁表面存储器

接下来介绍磁表面存储器中的磁盘存储器、磁带存储器和磁鼓存储器。

1. 磁盘存储器

磁盘存储器(Magnetic Disk Storage)是以磁盘为存储介质的存储器,它利用磁记录技术在两面涂有磁记录介质的同心圆轨道(即为磁道)上进行数据存储,具有存储容量大、数据传输速率高、可被随机访问,存储的数据可长期保存等特点,在计算机系统中,磁盘存储器是主存储器的扩充,常用于存放程序和数据。磁盘存储器是一种应用广泛的直接访问存储设备(Direct Access Storage Device,DASD),通常由磁盘、磁盘驱动器(或称磁盘机)和磁盘控制器构成。

(1) 磁盘。磁盘是两面涂有可磁化介质的平面圆片(即盘片)。根据磁盘盘基(磁盘上用来承载磁层的支持体)的不同,磁盘可分为硬盘和软盘两类。硬盘盘基用非磁性轻金属材料制成;软盘盘基用挠性塑料制成。按照盘片的安装方式不同,磁盘有固定和可互换(可装卸)两类,其中可互换的磁盘结构部件有下列几种。

① 单片盘片安装在塑料或金属扁盒内的部件称盘盒。

② 几片盘片同轴连装在一起的部件称盘组。

③ 磁盘与磁头臂同装在一密闭容器内的部件称头盘组件。

(2) 磁盘驱动器。磁盘驱动器是驱动磁盘转动并在盘面进行写入读出动作的装置,它包括盘片主轴旋转机构、驱动电机、头臂与头臂支架、头臂驱动电机、净化盘腔与空气净化机构、写入读出电路、伺服定位电路和控制逻辑电路等。通过将盘组或盘盒装在驱动器上,磁盘以恒定转速旋转,悬挂在头臂上具有浮动面的头块(浮动磁头),靠加载弹簧的力量压向盘面,盘片表面带动的气流将头块浮起。头块与盘片间保持稳定的微小间隙,经滤尘器过滤的空气不断送入盘腔,保持盘片和头块处于高度净化的环境内,以防头块与盘面划伤。

根据控制器送来的磁道地址和寻道命令,定位电路驱动直线电机将头臂移至目标磁道上,伺服磁头读出伺服磁道信号并反馈到定位电路,使头臂跟随伺服磁道稳定在目标磁道上。读写与选头电路根据控制器送来的磁头地址接通应选的磁头,将控制器送来的数据以串行方式逐位记录在目标磁道上,或从选定的磁道读出数据并送往控制器。

头臂装在梳形架小车上,在寻道时所有头臂一同移动,所有数据面上相同直径的同心圆磁道总称圆柱面(即头臂定位一次所能存取的全部磁道)。每个磁道都按固定的格式记录,在标志磁道起始位置的索引之后,记录该道的地址(圆柱面号和头号)、磁道的状况和其他参考信息。在每一记录段的尾部附记有该段的纠错码,对连续少数几位的永久缺陷所造成的错误靠纠错码纠正,对有多位永久缺陷的磁道须用备份磁道代替。读写操作是以记录段为单位进行的。记录段的长度有固定段长和可变段长两种。

磁盘驱动器分为头臂固定型和头臂移动型两类,其中头臂移动型磁盘驱动器又可分为磁盘可换型和磁盘固定型两种。固定头臂磁盘存储器不需要头臂定位部件,而是让每个盘面安装尽可能多的头臂。固定头臂磁盘与磁鼓的性能特点相同,只是由于磁头的造价昂贵,应用范围很小。新型固定式磁盘由于采用了温彻斯特技术,因此又称温彻斯特磁盘驱动器,简称温式磁盘机(温盘),具体如下。

① 密封的头盘组件。即将磁头、盘组和定位机构等密封在一个盘腔内,后来发展到连主轴电机等全部都装入盘腔,可进行整体更换。

② 采用小尺寸和小浮力的接触起停式浮动磁头。借以得到超小的头盘间隙(亚微米级),以提高记录密度。

③ 采用具有润滑性能的薄膜磁记录介质。

④ 采用磁性流体密封技术。可防止尘埃、油、气侵入盘腔，从而保持盘腔的高度净化。

⑤ 采用集成度高的前置放大器等。

采用了温彻斯特技术的硬盘驱动器与可换式磁盘比，大幅度提高了记录密度和可靠性，并进一步小型化。

(3) 磁盘控制器。磁盘控制器即磁盘驱动器适配器，它是计算机与磁盘驱动器的接口设备。磁盘控制器接收并解释计算机的命令，然后向磁盘驱动器发出各种控制信号，如检测磁盘驱动器状态；按照规定的磁盘数据格式，把数据写入磁盘和从磁盘读出数据。

磁盘控制器尽管类型很多，但工作原理大体上是相同的。它主要由与计算机系统总线相连的控制逻辑电路，微处理器，完成读出数据分离和写入数据补偿的读写数据解码和编码电路，数据检错和纠错电路，根据计算机发来的命令对数据传递、串并转换以及格式化等进行控制的逻辑电路，存放磁盘基本输入输出程序的只读存储器和用于数据交换的缓冲区等部分组成。

磁盘存储器的主要技术指标包括存储密度、存储容量、存取时间及数据传输率，这与磁表面存储器的技术指标大致相同。

严格来说，磁盘包括硬盘和软盘两种，但事实上，软盘已经很难看到了，接下来简要介绍下硬盘和软盘。

1) 硬盘

硬盘(Hard Disk，HD)是计算机上使用的以旋转盘片为基础的非易失性存储器，它在平整的磁性表面存储和检索数据，数据通过离磁性表面很近的磁头由电磁流来改变极性的方式被写入到磁盘上，并可以通过盘片被读取。硬盘的读写是采用随机存取的方式，可以按任意顺序读取硬盘中的数据，但读取不同位置的资料，速度不相同。

硬盘是计算机最为重要的存储设备，存放着用户所有的数据信息，这些数据的价值远远高于硬盘本身，由于计算机的数据掉电时都存储在硬盘中，所以硬盘成为了计算机必不可少的一个部件。同时，硬盘作为一种存储设备，又是计算机的主要组成部分，它从计算机诞生时起就一直扮演着不可或缺的角色。硬盘性能的好坏直接影响计算机的运行速度和用户的操作体验，尽管它从诞生到现在经历过很多阶段，但始终都朝着体积小、速度快、容量大的方向发展。

2) 软盘

软盘(Floppy Disk，FD)是个人计算机(Personal Computer，PC)中最早用来储存数据文件的可移动介质，主要部分是一张薄软的磁存储介质盘片，盘片封装在矩形塑料壳中，内衬有用于清理灰尘的纤维织物。

软盘是早期计算机上必备的一个硬件，也是计算机最早使用的可移动介质，它存取速度慢，容量也很小，但可装可卸、携带方便。软盘仅适用于那些需要被移动的小文件，作为一种可移动的存储设备，它的读写是通过软驱(软磁盘驱动器，Floppy Disk Driver，FDD)完成的，软驱每次对磁盘的读写均以被称为簇的若干个扇区为单位进行(软盘的盘片的每一面都被划分为多个同心圆式的磁道，每个磁道又被划分为多个扇区)。

软盘可分为硬磁区(Hard-Sectored)及软磁区(Soft-Sectored)，有 8 英寸(容量为 100KB～1MB)、5.25 英寸(容量为 100KB～1.2MB)和 3.25 英寸(容量为 400KB～1.44MB)等规格。

最早的软盘为32英寸，后来被8英寸软盘替代，这就是我们常说的标准软盘的鼻祖。8英寸软盘是一种表面涂有金属氧化物的塑料质矩形磁盘，大小和一个一般的披萨饼差不多，容量接近100KB，但后来5.25英寸的软盘逐渐成为当时家庭电脑的标准移动存储设备，其单面容量为180KB(后来出现了双面容量为360KB的)，再后来又出现了3.5英寸双面720KB的软盘，这些都属于低密度软盘；而5.25英寸的双面高密度的1.2MB的软盘和3.5英寸双面高密度的1.44MB的软盘，以及在软盘淘汰前出现过的2.88MB的软盘，都属于高密度软盘。

硬盘和软盘的差别简要总结如下。

(1) 硬盘盘基用非磁性轻金属材料制成；而软盘盘基用挠性塑料制成。

(2) 硬盘的转速(即盘片在一分钟内所能完成的旋转次数)高，存取速度快，最高可达6000RPM；而软盘转速低，存取速度慢，仅为300RPM。

注意：转速的单位为转/每分钟(Revolutions Per Minute，RPM)。

(3) 硬盘有固定头、固定盘和盘组等结构；而软盘都是活动头，是可换盘片结构。

(4) 硬盘是浮动磁头读写，磁头不接触盘片；而软盘磁头是接触式读写。

(5) 硬盘系统及硬盘片价格都比较贵，且大部分盘片不能互换；而软盘造价低，盘片保管方便，使用灵活，且具有互换性。

(6) 硬盘对环境要求苛刻，要有超净措施；而软盘则对环境要求不太严格。

【例3-21】 假设某磁盘共有5个盘片，最外两层盘面不能记录，每面有200条磁道，每条磁道有10个扇区，每个扇区有512B，磁盘机以5000RPM的速度旋转，平均寻道时间为8ms。计算该磁盘的存储容量(磁盘存储器的存储容量＝盘面数×每个盘面的磁道数×每条磁道的扇区数×每个扇区的字节数)和平均寻址时间(平均寻址时间＝平均寻道时间＋平均等待时间)。

注意：存储容量的计算公式也可以为

存储容量＝磁头数×磁道(柱面)数×每道扇区数×每扇区字节数

解：该磁盘的5个盘片则有10个盘面，其中2个不能记录，所以只有8(10－2＝8)个盘面可用于存储，故

$$\begin{aligned}\text{磁盘存储器的存储容量}&=\text{盘面数}\times\text{每个盘面的磁道数}\times\text{每条磁道的扇区数}\times\\&\quad\text{每个扇区的字节数}\\&=10\times200\times10\times512\\&=10240000\text{B}\end{aligned}$$

由于平均寻址时间＝平均寻道时间＋平均等待时间，目前已知平均寻道时间为8ms，而平均等待时间未知。

已知磁盘转速为5000RPM，则磁盘旋转一周的时间为

$$1\text{min}/5000\text{RPM}=0.0002\text{min}=12\text{ms}$$

故平均等待时间为

$$\frac{0\text{ms}+12\text{ms}}{2}=6\text{ms}$$

所以

平均寻址时间＝平均寻道时间＋平均等待时间
＝8ms＋6ms
＝14ms

2．磁带存储器

磁带存储器是以磁带为存储介质，由磁带机及其控制器组成的存储设备，它是计算机的一种辅助存储器。磁带存储器和磁盘存储器的原理基本相同，但前者速度较慢（因为只能顺序访问）。接下来简单介绍一下什么是磁带。

磁带是一种用于记录声音、图像、数字或其他信号的载有磁层的带状材料（如数码音频磁带和数码线性磁带等），是产量最大和用途最广的一种磁记录材料。这种材料通常是在塑料薄膜带基（磁带上用来承载磁层的支持体）上涂覆一层颗粒状磁性材料或蒸发沉积上一层磁性氧化物或合金薄膜而成。以前曾使用纸和赛璐珞（一种塑料）等作带基，现主要用强度高、稳定性好和不易变形的聚酯薄膜。

磁带可以用于制作录音带、录像带、计算机带和仪表磁带等，具体如下。

1）录音带

在20世纪30年代，录音带开始出现。1963年，荷兰飞利浦公司成功研制出盒式录音带，它具有轻便、耐用、互换性强等优点从而得到迅速发展；1973年，日本研制成功了Avilyn包钴磁粉带；1978年，美国生产出金属磁粉带；而由日本日立玛克赛尔公司制成的微型及数码盒式录音带，又使录音带达到一个新的水平，并使音频记录进入了数字化时代。中国在20世纪60年代初就开始生产录音带，1975年试制成功了盒式录音带，并已达到较高水平。

在流行音乐领域，磁带录音方便可靠，价钱便宜，质量又好，使得投资不多的小型录音公司得以生存下去，为20世纪50年代独立唱片公司的发展壮大立下了汗马功劳，而之后出现的杜比技术，让可以录音的卡式磁带走进了消费者的家中，这极大的推动了录音带的普及和应用。

2）录像带

自从1956年美国安佩克斯公司制成录像机以来，录像带已从电视广播领域逐步进入到科学技术、文化教育、电影和家庭娱乐等各个领域。除了用二氧化铬包钴磁粉以及金属磁粉制成录像带外，日本还制成了微型镀膜录像带，并开发了钡铁氧体型垂直磁化录像带。

3）计算机带

计算机带在信息存储方面具有容量大、价格低的优点。它曾被大量用于计算机的外部存储器，如今仅在专业设备上使用（如车床控制机）。最为典型的产品是磁带机（tape drive），现简要介绍如下。

磁带机一般指单驱动器产品，通常由磁带驱动器和磁带构成，是一种经济、可靠、容量大、速度快的备份设备。这种产品采用高纠错能力编码技术和写后即读通道技术，可以大大提高数据备份的可靠性。根据装带方式的不同，一般分为手动装带磁带机和自动装带磁带机（即自动加载磁带机）。自动加载磁带机实际上是由磁带和磁带机结合组成的，它是一个位于单机中的磁带驱动器和自动磁带更换装置，可以从装有多盘磁带的磁带匣中拾取磁带并放入驱动器中，或执行相反的过程。

自动加载磁带机能够支持例行备份过程，自动为每日的备份工作装载新的磁带。一个

拥有工作组服务器的小公司可以使用自动加载磁带机来自动完成备份工作。提供磁带机的厂商很多，IT 厂商中 HP(惠普)、IBM、Exabyte(安百特)等均有磁带机产品，另外专业的存储厂商如 StorageTek、ADIC、Spectra Logic 等公司均以磁带机、磁带库等为主推产品。

4) 仪表磁带

仪表磁带也称仪器磁带或精密磁带。在近现代各种科学实验中，常常需要把人们无法接近的测量数据自动而连续地记录下来(即遥控遥测技术)，如原子弹爆炸和卫星空间探测都要求准确无误地同时记录成百上千项数据。仪表磁带就是在上述背景下发展起来的，因为它所记录的数据十分重要，所以对于性能和制造都有着严格的要求。

3. 磁鼓存储器

磁鼓是利用高速旋转的圆柱体磁性表面作记录媒体的存储设备。磁鼓存储器在 1950—1960 年用作计算机的主要外存储器，它利用了电磁感应原理进行数字信息的记录(写入)与再生(读出)，由作为信息载体的磁鼓筒、磁头、读写及译码电路和控制电路等主要部分组成。由于鼓筒旋转速度很高，因此存取速度快，但它最大的问题是利用率不高，一个大圆柱体只有表面一层用于存储数据，而磁盘的两面都能用来存储数据，显然后者利用率要高得多。因此，随着磁盘存储器的出现与发展，1960 年以后磁鼓存储器就逐渐被淘汰，仅用于特殊应用场合(如摄像机)。

3.7.3 闪速存储器

U 盘和固态硬盘是目前闪速存储器中最为典型的两种应用，现分别介绍如下。

1. U 盘

U 盘是“通用串行总线(Universal Serial Bus，USB)闪存盘”的简称，它基于 USB 接口，以闪存芯片(该芯片可电擦写，在通电以后可改变状态，否则就固定状态，所以断电以后资料能够保存)为存储介质且无须驱动器的新一代存储设备。

U 盘的结构大致可分为五部分：包括 USB 端口、主控芯片、闪存芯片、印制电路板(Printed Circuit Board，PCB，又称印刷线路板)底板和外壳封装。U 盘的基本工作原理也比较简单：即 USB 端口是数据输入或输出的通道，负责连接计算机；主控芯片是 U 盘的“大脑”，负责各部件的协调管理和下达各项动作指令，并使计算机将 U 盘识别为“可移动磁盘”；闪存芯片与计算机中内存条的原理基本相同，是保存数据的实体，其特点是断电后数据不会丢失，能长期保存；PCB 底板是负责提供相应的数据处理平台，且将各部件连接在一起。

U 盘的出现是移动存储技术领域的一大突破，其体积小巧，特别适合随身携带，可以随时随地、轻松交换资料数据，是理想的移动办公及数据存储交换产品。U 盘采用 USB 接口标准，读写速度较快。从稳定性上讲，U 盘没有机械读写装置，避免了移动硬盘容易因碰伤、跌落等原因造成的损坏，且可在任何带有 USB 接口的计算机中使用。

U 盘具有以下优点：①不需要驱动器，无需外接电源；②容量大；③体积非常小，仅大拇指般大小，重量仅约 20 克；④使用简便，即插即用，可带电插拔；⑤存取速度快，约为软盘速度的 15 倍；⑥可靠性好，可擦写达 100 万次，数据至少可保存 10 年；⑦抗震，防潮，耐高低温，携带十分方便；⑧使用 USB 接口；⑨具备系统启动、杀毒、加密保护和装载工具等功能。U 盘的这些优点将软盘赶出了市场。

2. 固态硬盘

固态硬盘(Solid State Disk,SSD)是一种主要以闪存作为永久性存储器的计算机存储设备。它由控制单元和存储单元(Flash 芯片或 DRAM 芯片)组成,被广泛应用于工控、视频监控、网络监控、网络终端、导航设备等诸多领域。固态硬盘最大的优点就是可以移动,而且数据保护不受电源控制,能适应于各种环境,适合于个人用户使用,此外,它还具有读写速度快、防震抗摔、低功耗、无噪音和工作温度范围大等优点。因此,虽然其成本较高,但也正在逐渐普及到 DIY(Do It Yourself 的英文缩写,意思是自己动手制作)市场。

由于固态硬盘技术与传统的机械硬盘技术不同,所以产生了不少新兴的存储器厂商(只需购买 NAND 存储器,再配合适当的控制芯片,就可以制造固态硬盘)。与机械硬盘相比,固态硬盘有着较高的读写速度,但成本较高,容量较低,而且一旦硬件损坏,数据较难恢复。新一代的固态硬盘普遍采用 SATA(Serial Advanced Technology Attachment,串行高级技术附件)-2 接口、SATA-3 接口、SAS(Serial Attached Small Computer System Interface,串行小型计算机系统接口)接口、mSATA 接口和 PCI-E(Peripheral Component Interconnect-Express,外设部件互连-快速)接口。

通常所说的 SSD 是基于闪存的固态硬盘(IDE Flash Disk、SATA Flash Disk),它采用闪存芯片作为存储介质,SSD 可以被制作成如笔记本硬盘、微硬盘、存储卡、U 盘等设备。另一种采用 DRAM 作为存储介质的固态硬盘可分为 SSD 硬盘和 SSD 硬盘阵列两种,应用范围较窄,属于非主流的设备,它仿效传统硬盘的设计,可被绝大部分操作系统的文件系统工具进行卷设置和管理,并提供工业标准的 PCI 和 FC(Fibre Channel,网状通道技术)接口用于连接主机或者服务器。DRAM 固态硬盘是一种高性能的存储器,而且使用寿命很长,美中不足的是需要独立电源来保护数据安全。表 3-8 为固态硬盘的发展情况。

表 3-8　固态硬盘的发展情况

序号	时间	事　　件
1	1989 年	世界上第一款固态硬盘出现
2	2006 年	三星率先发布一款 32GB 容量的固态硬盘笔记本电脑
3	2007 年	东芝推出了其第一款 120GB 固态硬盘笔记本电脑
4	2008 年	忆正 MemoRight SSD 的正式发布,标志着中国企业加速进军固态硬盘行业
5	2010 年	镁光发布了全球首款 SATA 6Gb/s 接口固态硬盘
6	2012 年	苹果公司在笔记本电脑上应用容量为 512G 的固态硬盘
7	2015 年	特科芯推出了首款 Type-C 接口的移动固态硬盘
8	2016 年	中国存储厂商特科芯发布了全球首款 Type-C 指纹加密 SSD

【例 3-22】 假设固态硬盘的擦写次数为 3000 次,普通用户每天写入的数据为 1GB,试计算 64GB 和 128GB 的固态硬盘的不间断地使用时间,若某一用户每天写入的数据分别为 10GB 和 100GB,试计算上述固态硬盘的使用时间。

解:

根据固态硬盘的平衡写入机制,64GB 的固态硬盘可擦写的总数据量为

$$64\text{GB}\times 3000=192000\text{GB}$$

由于普通用户每天写入的数据为 1GB,因此 64GB 的固态硬盘可以被不间断地使用的天数如下:

$$192000/1=192000\ \text{天}$$

同理，128GB的固态硬盘在同等条件下则可以被不间断地使用384000天，即约为1052年。

如果某一用户每天写入的数据分别为10GB和100GB，则对于64G的固态硬盘，使用时间分别为19200天和1920天；而对于128GB的固态硬盘，使用时间分别为38400天和3840天。

从此题可以看出，如果你用的是128G的固态硬盘的话，每天写入的数据是10GB，则可以不间断用38400天（以每年365天计算，近似为105年），这意味着固态硬盘就像普通硬盘一样，理论上可以无限制读写。

3.7.4 光存储器

光存储技术是一种通过光学的方法读写数据的存储技术。其基本物理原理是改变一个存储单元的某种性质，使其性质的变化反映被存储的数据。识别这种存储单元性质的变化，就可以读出存储的数据。光存储单元的性质，由于高能量的激光束可以聚焦成约1微米的光斑，因此它比其他存储技术有更高的存储容量。

光盘系统较早应用于小型音频系统中，它使得音响系统具有优异的音响效果。其中，光盘驱动器的读写头是用半导体激光器和光路系统组成的光头，光盘为表面具有磁光性质的玻璃或塑料等圆形盘片。20世纪80年代初光盘系统开始逐步进入计算应用领域，特别是在多媒体个人电脑（Multimedia Personal Computer，MPC，多媒体电脑）中扮演着极为重要的角色。由于多媒体应用存储的信息（主要是视频、图像和音频信息）数字化后要占用巨大的存储空间，以至于传统的存储设备如磁盘、磁带等已无法满足这一要求。这使光存储技术的发展和商品化就成为一个必然的趋势，因此光盘和光盘驱动器一度成为多媒体电脑的核心硬件设备（现在笔记本电脑已经不再配有光盘系统，如超级本）。

光盘有很多种类型，如只读光盘（Compact Disc Read Only Memory，CD-ROM）、高密度数字视频光盘（Digital Video Disc Read Only Memory，DVD-ROM）、可记录光盘（Compact Disk-Recordable，CD-R）、可擦写光盘机（Compact Disk-ReWritable，CD-RW）、写一次读多次（WORM Write Once，Read Many）和磁光盘（Megneto-Optical Disk，MO）。

注意：从严格意义上来讲，磁光盘不能算光盘，因为它是磁场技术和激光技术相结合的产物，由磁道和扇区组成，可以随机写入、擦除或重写。

光盘系统与磁盘系统主要存在以下不同。

（1）表达原理不同。磁盘系统单靠磁场来更改已储存的数据，光盘系统则是利用磁场和激光光束来更改已储存的数据。

（2）数据读写不同。磁盘系统是通过磁头以感应的方式从磁盘读写数据，磁头与高速旋转的磁盘必须保持一定的间隙。这种方式容易造成磁头碰撞盘片而损坏数据，光盘系统是以激光光束来进行读写，一般不会发生光头碰撞，安全性能好。

（3）传输速率不同。磁盘系统的传输率一般是恒定的，而光盘系统的传输速率则与激光输出功率息息相关。激光输出功率为20～30mW时，光盘系统的传输速率为每秒2～6MB，激光输出功率升至40mW时，光盘系统的传输速率可达每秒10MB。就是说，激光输出功率越高，数据传输速率就越快。

（4）存储容量不同。磁盘系统的容量为磁盘的格式所限定，光盘系统的容量则视激光波长而定，激光波长愈短，便能缩短间距而提高存储容量。光盘系统的主要优点是数据盘不

易损坏，使用寿命长；存储容量大且拆卸方便，性能价格比高，每兆字节的价格仅为软盘的百分之一。不足之处是速度没有硬盘快。

光盘系统的主要技术指标包括存储容量、平均存取时间及数据传输率，具体如下。

(1) 存储容量。光盘系统的存储容量指它所能读写光盘盘片的容量。光盘盘片的容量又分为格式化容量和用户容量。格式化容量指按某种标准格式化后的容量，采用不同的格式就会有不同的容量；用户容量一般比格式化容量小。

(2) 平均存取时间。光盘系统的平均存取时间包括三部分：即平均寻道时间、平均等待时间和光头稳定时间。一般取光头沿半径移动 1/3 所需时间为平均寻道时间，盘片旋转一周的一半时间为平均等待时间，二者加上光头稳定时间为平均存取时间。

(3) 数据传输率。光盘系统的数据传输率是指从光盘驱动器送出的数据率，它可以定义为单位时间内从光盘的光道上传送的数据比特数。

3.8 小结

存储器是计算机系统中的记忆设备，用来存放程序和数据。随着计算机的发展，存储器在系统中的地位越来越重要。由于超大规模集成电路的制作技术日新月异，使 CPU 的处理速度屡创新高，而相比之下，存储器的存取速度没有太大变化，因此很难与之适配，这使计算机系统的运行速度在很大程度上受到存储器速度的制约。

此外，一方面由于 I/O 设备不断增多，如果它们与存储器打交道都需要通过 CPU 来实现，这会大大降低 CPU 的工作效率；另一方面在多处理机的系统中，各处理机本身都需与其主存交换信息，而且各处理机在互相通信中，也都需共享存放在存储器中的数据，因此，存储器的地位更为重要。从某种意义上讲，存储器的性能已成为计算机系统的核心。

本章首先介绍了存储器的基本概念，主要的技术指标和层次结构。然后分别介绍了主存(包括随机存储器、只读存储器和闪速存储器)和辅存(包括磁表面存储器、光存储器和闪速存储器)，并行存储器，存储器和 CPU 的连接等相关知识，重点介绍了高速缓存和虚拟存储器。

其中，虚拟存储器和高速缓存两者的异同如下：

(1) 相同之处：两者都是通过构造分层存储结构提高存储系统的性价比；两者都基于程序运行的局部性原理将当前需要使用的程序或数据从外存调入高速缓存。

(2) 不同之处：高速缓存主要解决 CPU 与主存之间的速度不匹配，虚拟存储器则着眼于解决实存容量不够的问题；高速缓存对程序员是透明的，虚拟存储器对程序员不透明；两者若发生不命中时，系统的效率不一样。