# 第三章 多层次的存储器

- 3.1 存储器概述
- 3.2 SRAM存储器
- 3.3 DRAM存储器
- 3.4 只读存储器和闪速存储器
- 3.5 并行存储器
- 3.6 Cache存储器
- 3.7 虚拟存储器
- 3.8 奔腾系列机的虚存组织

# 3.1存储器概述

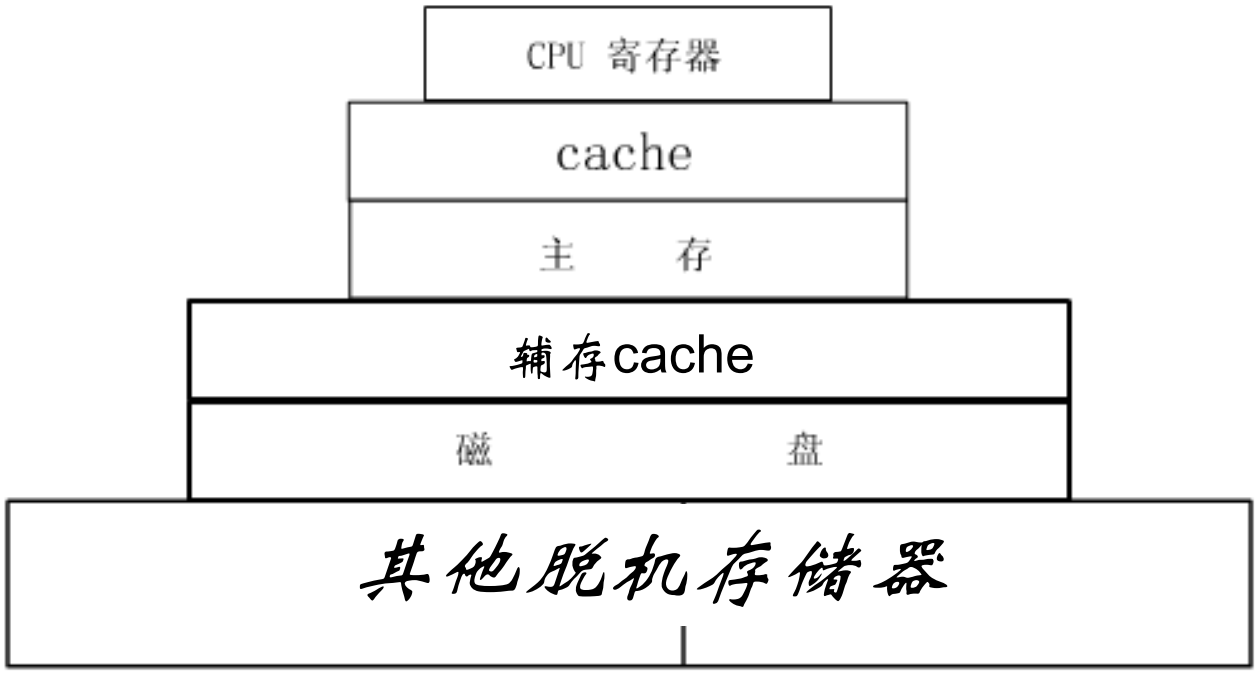
- 存储器是计算机系统中的记忆设备，用来存放程序和数据；
- 两大功能：
  - 1、 存储（写入Write）　　2、 取出（读出Read）
- 基本要求：
  - 1、大容量　　2、高速度　　3、低成本
- 存储元：基本存储单元，存储一位（bit）二进制代码的存储元件称为基本存储单元（或）。
- 存储单元：主存中最小可编址的单位，是CPU对主存可访问操作的最小单位， $2^n$个存储元组成。每个存储单元都有一个编号，别于访问，称为存储单元地址。
- 存储器：多个存储单元按一定规则组成一个整体。
  - cpu与存储器进行数据交换的单位是存储单元，不是存储元。

# 3.1.1 存储器的分类

- 按存储介质分类：
  - 磁表面存储器：用磁性材料做成的存储器
  - 半导体存储器：用半导体器件组成的存储器
- 按存取方式分类：
  - 随机存储器：任何存储单元的内容都能被随机存取，且存取时间和存储单元的物理位置无关。数组形式？
  - 顺序存取存储器（磁带）：只能按某种顺序来存取，存取时间和存储单元的物理位置有关。堆栈形式？
- 按读写功能分类：
  - **RAM /** 随机读写存储器：双极型/MOS
  - **ROM /** 只读存储器：MROM/PROM/EPROM/EEPROM
- 按信息的可保存性分类：永久性和非永久性的
- 按存储器系统中的作用分类：
  - 主存储器、辅助存储器、高速缓冲存储器、控制存储器

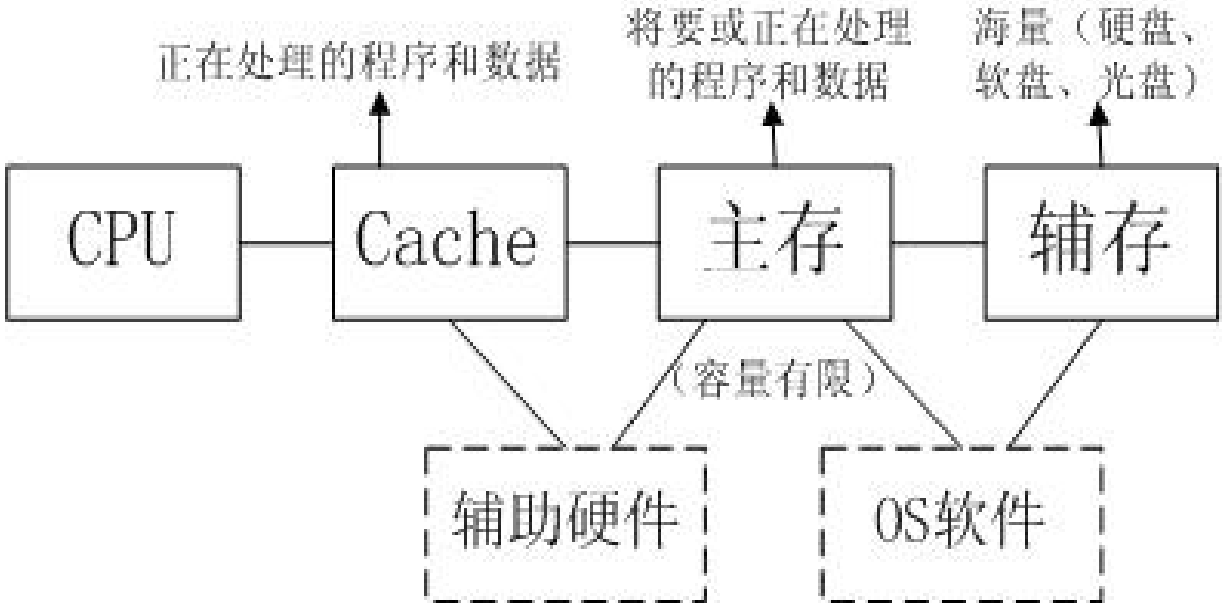
# 3.1.2 存储器的分级

- 综合考虑速度和价格方面的因素作折中考虑，计算机系统中建立了分层次的存储器体系结构。
  - 高速缓冲存储器简称**cache**，它是计算机系统中的一个高速小容量半导体存储器。
  - 主存储器简称主存，是计算机系统的主要存储器，用来存放计算机运行期间的大量程序和数据。
  - 外存储器简称外存，它是大容量辅助存储器。



# 3.1.2 存储器的分级

- 分层存储器系统之间的连接关系
  - 寄存器：微处理器内部的存储单元，保存当前运算数据；
  - 高速缓存（Cache）：正在使用的数据；复杂运算的多个数据
  - 主存储器：存放当前运行程序和数据，采用半导体存储器构成
  - 辅助存储器：磁记录/光记录方式，以外设方式连接和访问



# 3.1.3 主存储器的技术指标

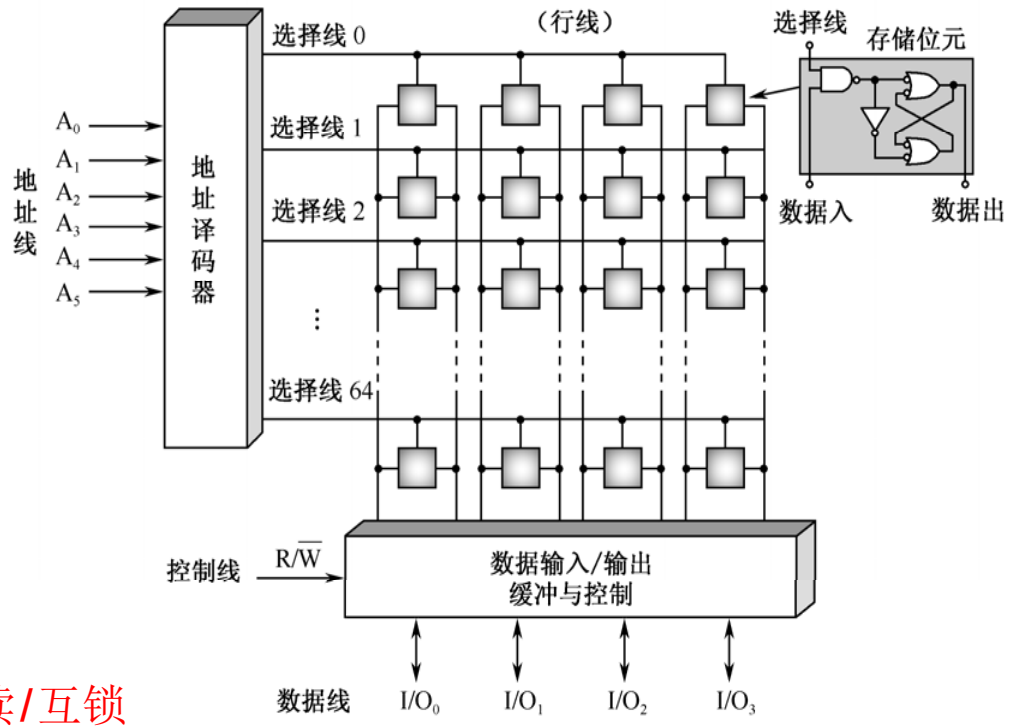
- 字节存储单元：
  - 存储器划分为以存储 字节数据/8位2进制数据 为单位的结构；每个存储单元是一个 字节存储单元，对应一个唯一的地址编号，是存储单元的地址，字节地址。
- 字存储单元：
  - 使用连续的多个 字节存储单元 来存放一个多字节的 机器字，这多个 字节存储单元块 就是一个**字存储单元**；块中最小的字节地址就是字存储单元的地址，**字地址**。
  - 机器字 在**字节块**中的保存方式：**高地址**高字节，**低地址**低字节。
- 存储容量：
  - 存储器中 储单元 总数。存储容量越大，能存储的信息就越多。
  - 存储容量=**存储单元个数×每个存储单元保存数据位数**
    - 存储容量2KB=2K×8；2K个存储单元，每个存储单元保存8位数据
    - 存储容量16K×32；16K个存储单元，每个存储单元保存32位数据
- 存取时间/存储器访问时间：
  - 指一次读操作命令发出到该操作完成所经历的时间。通常取写操作时间等于读操作时间，故称为存储器存取时间。
- 存储周期：
  - 指连续启动两次读操作所需间隔的最小时间。通常，存储周期略大于存取时间，其时间单位为ns。
- 存储器带宽：
  - 单位时间里存储器所存取的信息量，通常以位/秒或字节/秒做度量单位。

# 3.2 SRAM存储器

- 主存（内部存储器）是半导体存储器。根据信息存储的机理不同可以分为两类：
  - 静态读写存储器(**SRAM---- Static RAM**)：存取速度快
    - 以触发器为基本存储单元
    - 不需要额外的刷新电路
    - 速度快，但集成度低，功耗和价格较高
  - 动态读写存储器(**DRAM---- Dynamic RAM**)：存储容量大。
    - 以单个MOS管为基本存储单元
    - 要不断进行刷新（Refresh）操作
    - 集成度高、价格低、功耗小，但速度较SRAM慢

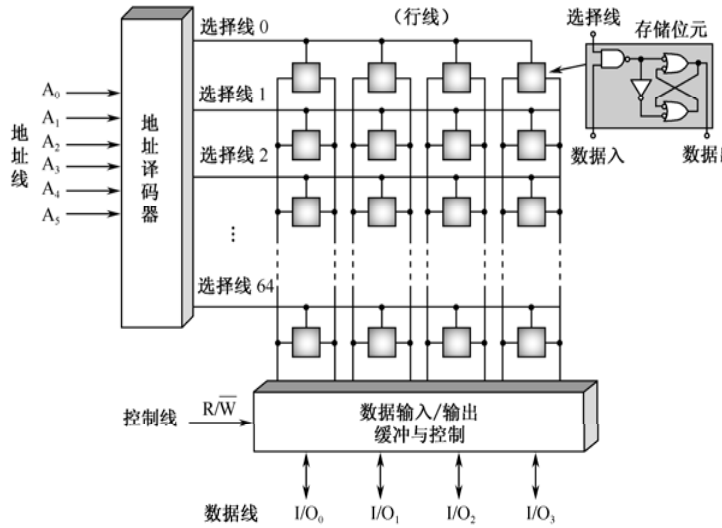
# 3.2.1 基本的静态存元阵列

- **SRAM存储元**：一个触发器，保存1位2进制。
  - n个 存储元 组成SRAM中的一个**存储单元，n一般为8；**
  - 全部 存储元 组成SRAM的元阵列；元阵列中 存储元 排列成矩阵形式；
- 与元阵列交换数据，需要三组信号线：
  - **地址线**：选择存储元
  - 选择方式：
    - 选择行，选择列
    - 同时选择行/列
  - **数据线**：提供数据
  - **控制线**
    - 读写控制R/~W
    - 片选~CS
    - 读出使能~OE
      - 控制读就不写，写就不读/互锁



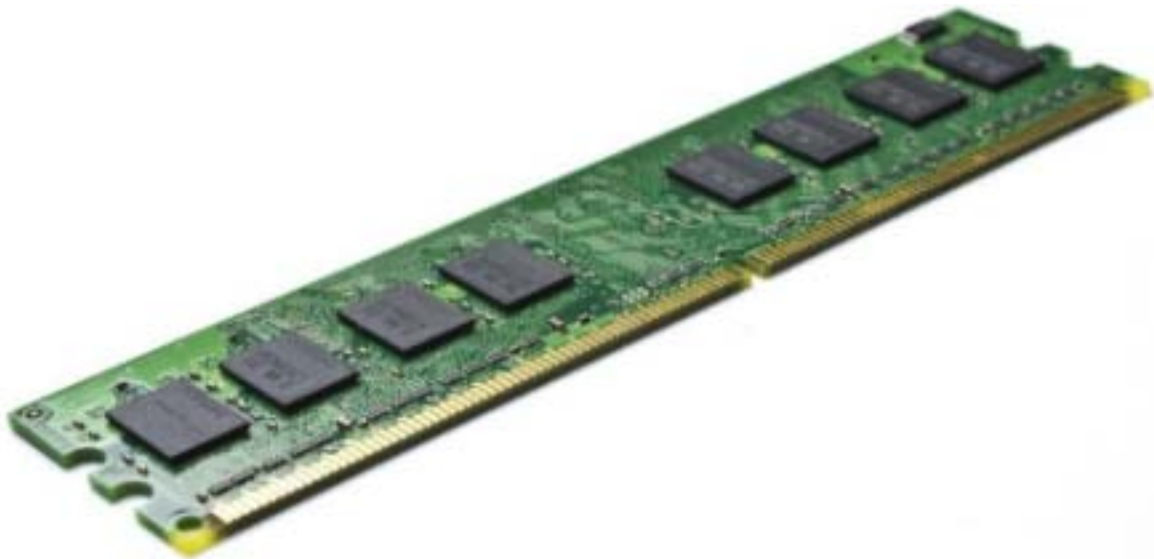
# 3.2.1 基本的静态存元阵列

- 元阵列存储元排列成**64×4**的矩阵：64个存储单元，字长4位
  - 地址线$A_0$~$A_5$：6条，存储元阵列的存储单元为$2^6=64$个；
    - n位地址线，存储单元$=2^n$
  - 数据线$I/O_0$~$I/O_3$：4条，存储器的字长是4位，及每个存储单元有4个存储元，可以保存1个 4位2进制数据。
    - 默认每个存储单元有8个存储元。
  - 存储元总数/存储容量=存储单元个数×字长=64×4=256个存储元



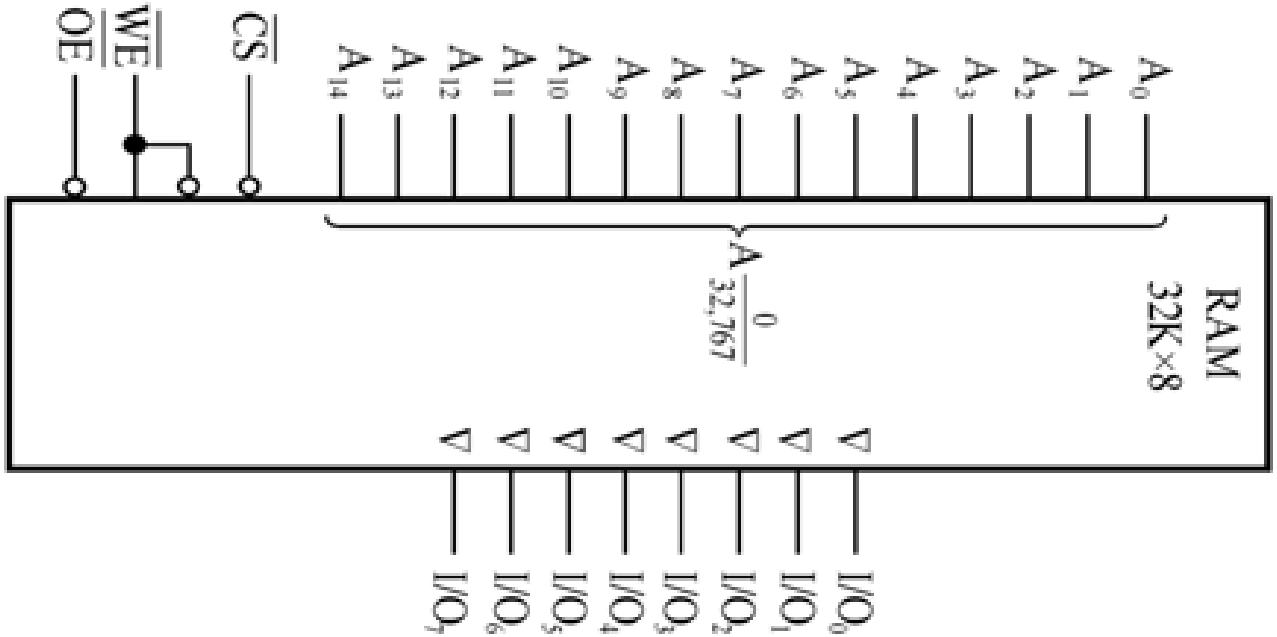
# 3.2.2 基本的SRAM逻辑结构

- N个 存储元 组成一个存储单元，大量存储单元$2^n$构成存储器芯片。
- 存储器芯片容量：
  - 存储单元数×每个存储单元的数据位数 ＝$2^n$×N＝芯片的存储容量
    - n＝芯片地址线的个数
    - N＝数据线的个数
- 为了方便管理/访问阵列中的存储元，SRAM中的所有存储元排列成矩阵形式，可以使用二级译码，确定阵列中要访问的存储元：
  - 将访问地址分成x向/行、y向/列两部分提供行地址和列地址，行地址和列地址共同指定的存储单元被选中。



# 3.2.2 基本的SRAM逻辑结构

- 设计存储容量=32K×8位的SRAM
  - 存储元总数是32K×8，存储单元32K个，字长8，每个存储单元8个存储元/8位数据
  - 为了访问存储单元，地址线：15 $A_0$~$A_{14}$
  - 为了与存储单元交换数据，数据线：8 $I/O_0$~$I/O_7$

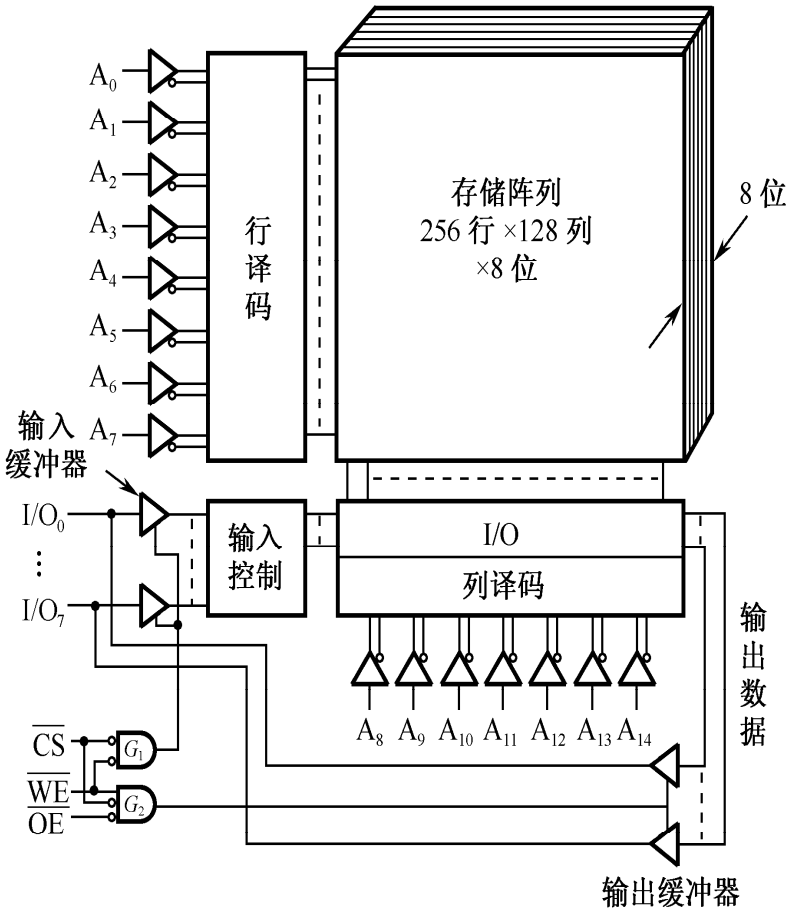


# 3.2.2 基本的SRAM逻辑结构

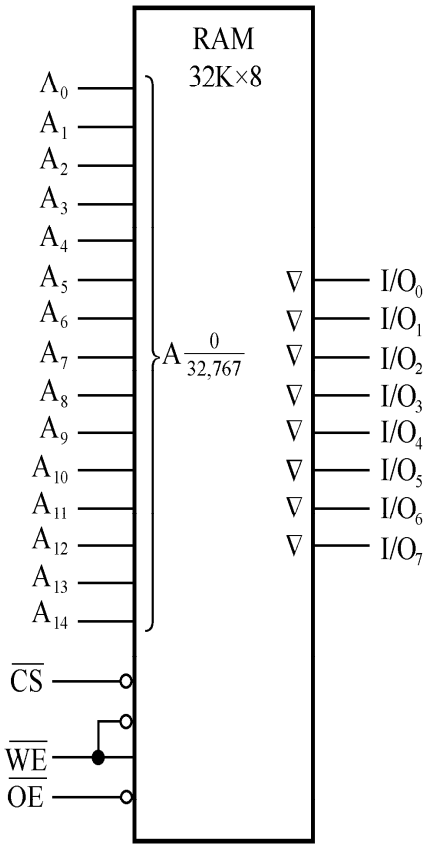
- 设计存储容量=32K×8位的SRAM
  - 使用**32K×1**的芯片设计：
    - 确定芯片数：32K×8/32K×1=8，需要8片芯片，每个存储单元的8位数据，分别保存在8片芯片的同一位置；
    - 组织每片芯片存储元：每片芯片容量/存储元=32K×1，将存储元排列组合成矩阵形式，分解$32K=2^{15}=2^8\times2^7=256\times128$，则将32K个存储元排列成256×128的存储元矩阵；
    - 地址线$A_0\sim A_{14}$分配：要在256行128列的存储元矩阵中选的确定的1个存储元，需要分配行地址8位$A_0\sim A_7$，列地址7位$A_8\sim A_{14}$，行/列地址以同样方式连接8片芯片，对应确定的一个地址信号，在每片芯片上选择1个存储元，8片芯片同时选择8个存储元，组成一个8位数据；
    - 数据线$I/O_0\sim I/O_7$：每位数据线连接1片芯片。

# 3.2.2 基本的SRAM逻辑结构

- 设计存储容量=32K×8位的SRAM
  - 逻辑图：



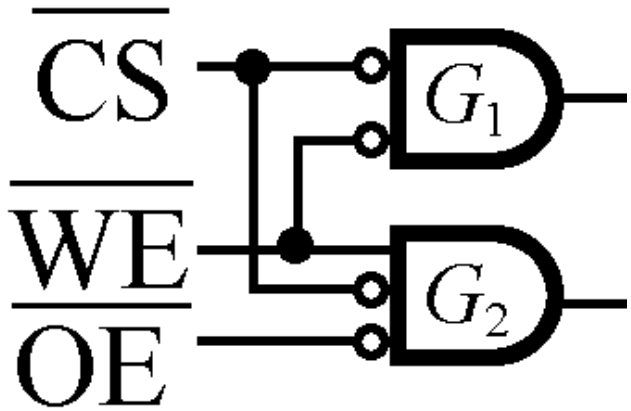

(a)




(b)

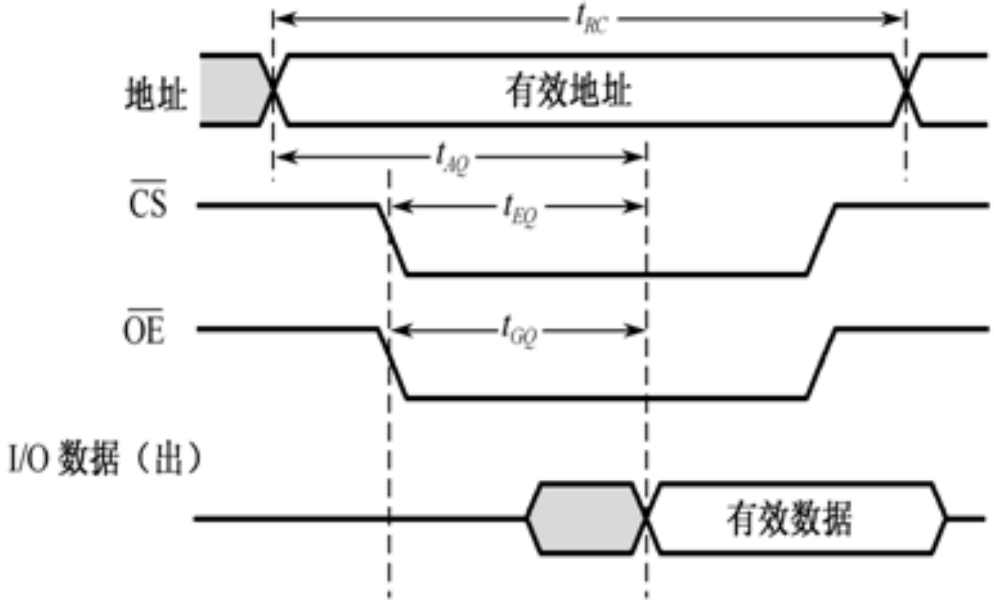
# 3.2.2 基本的SRAM逻辑结构

- 读与写的互锁逻辑
  - 实现读取SRAM中存储单元的数据与写入数据
    - 控制信号中CS是片选信号，CS有效时（低电平），门G1、G2均被打开。
    - OE为读出使能信号，OE有效时（低电平），门G2开启，当写命令WE=1时（高电平），门G1关闭，存储器进行读操作。
    - 写操作时，WE=0，门G1开启，门G2关闭。
    - 门G1和G2是互锁的，一个开启时另一个必定关闭，这样保证了读时不写，写时不读。



# 3.2.3 读写周期波形图

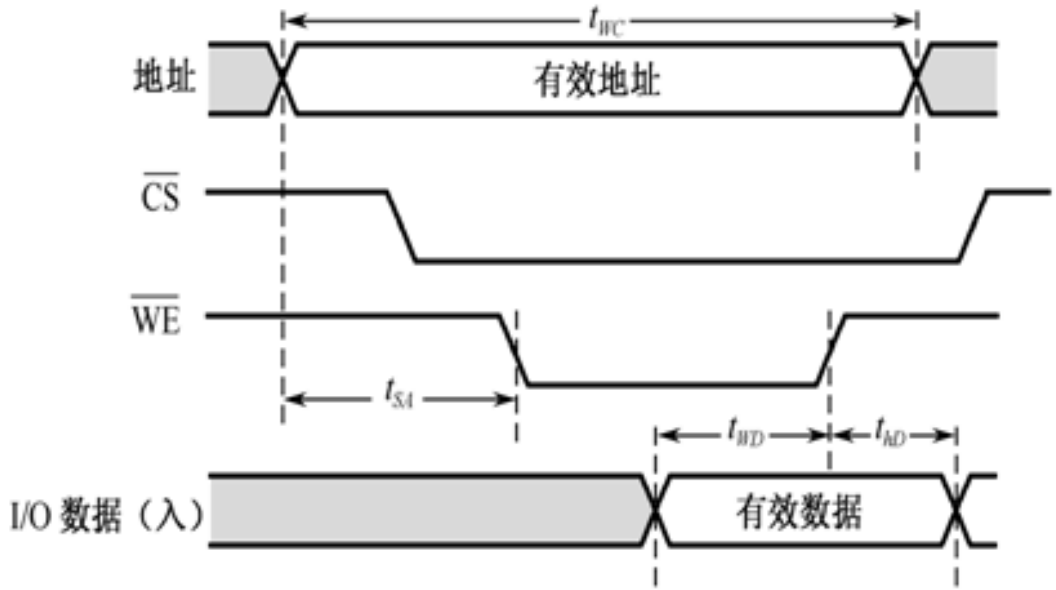
- 读周期~WE=1
  - 读出时间$t_{AQ}$：是从给出有效地址到外部数据总线上稳定地出现读出 数据所经历的时间。
  - 读周期时间$t_{RC}$：则是存储片进行两次连续读操作时所 必须间隔的时间，它总是大于或等于读出时间。
  - 过程：地址信号有效 → 片选/使能信号 有效→ 读出数据→片选/使能信号 无效→开始下一个读/写操作




(a) 读周期 ($\overline{WE}$ 高)

# 3.2.3 读写周期波形图

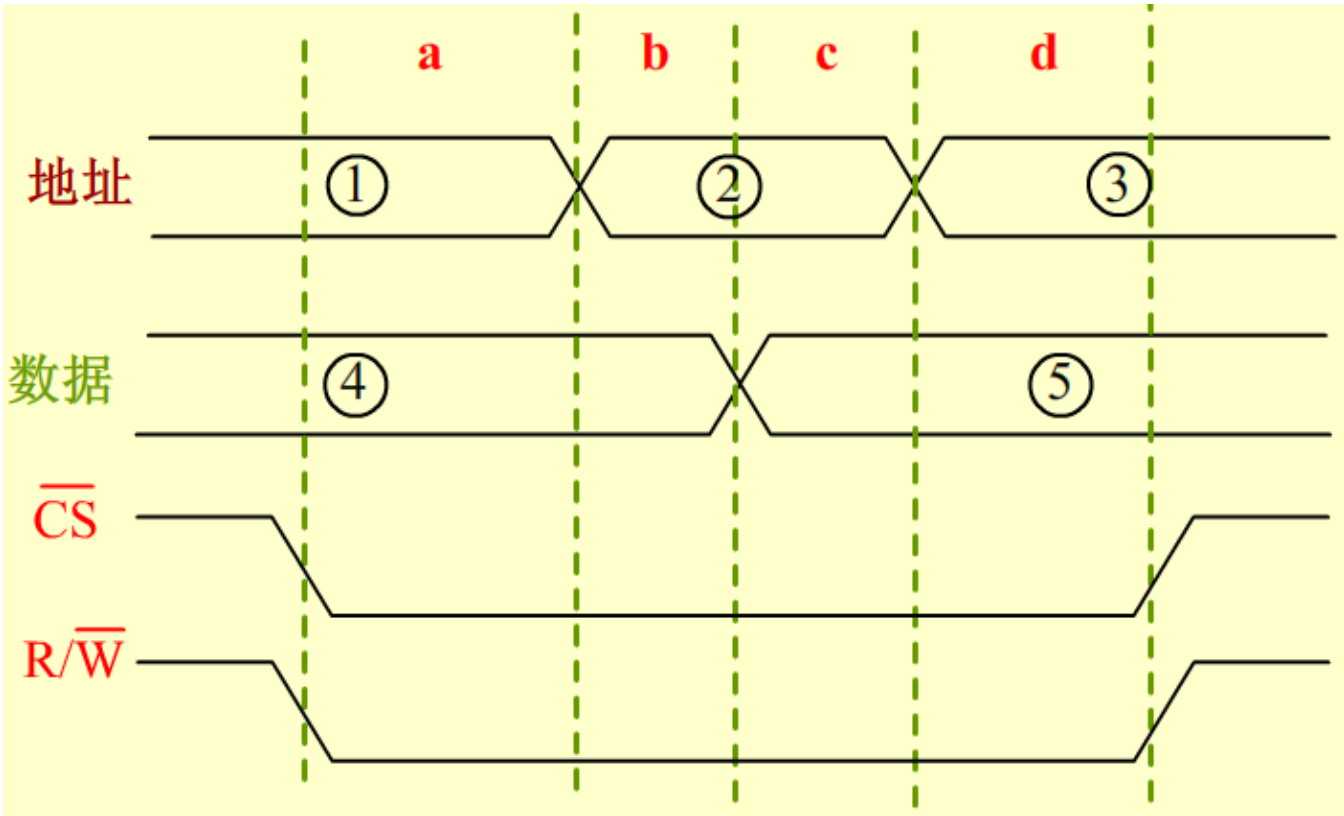
- 写周期~WE=0
  - 写入时间$t_{WD}$：将数据写入存储器的时间。
  - 写周期时间$t_{WC}$：完成一次写操作的时间。
  - 过程：地址信号有效 → 片选有效→ 写信号有效 →写信号无效→片选信号 无效→开始下一个读/写操作
  - 一般$t_{AQ} = t_{WD}$，存取时间




(b) 写周期($\overline{WE}$低)

# 3.2.3 读写周期波形图

- 例：P68 图是SRAM的写入时序图。其中R/W是读/写命令控制线，当R/W线为低电平时，存储器按给定地址把数据线上的数据写入存储器。请指出图写入时序中的错误，并画出正确的写入时序图。一个有效的读写期间，要求对应的地址和数据是稳定的。



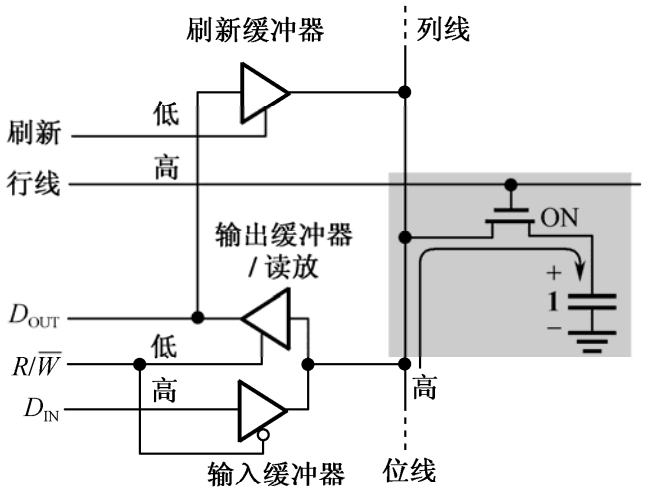

(b) 正确时序

# 3.3 DRAM存储器

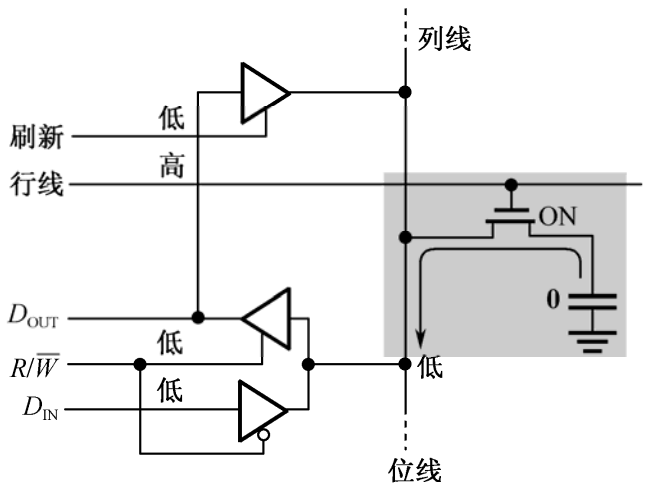
- **SRAM**存储元：触发器，它具有两个稳定的状态。
  - 触发器的稳定状态可以保持；
- **DRAM**存储元：**MOS**晶体管和电容器组成的记忆电路。
  - 当电容器充满电荷时，代表存储了1，
  - 当电容器放电没有电荷时，代表存储了0。
    - 电容会漏电？

# 3.3.1 DRAM存储位元的记忆原理

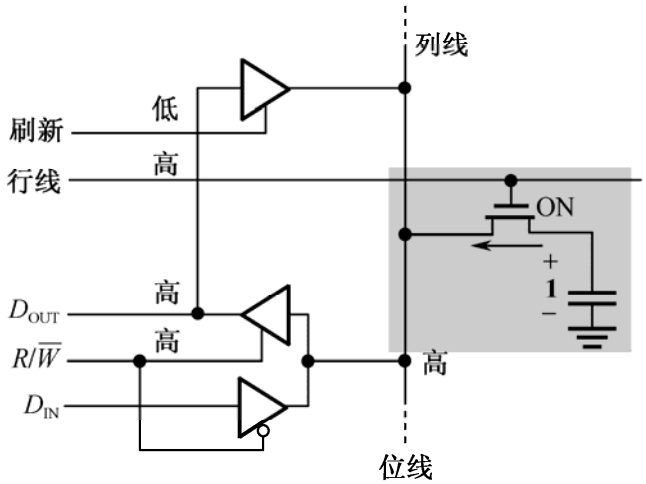
- **DRAM读写示意图**



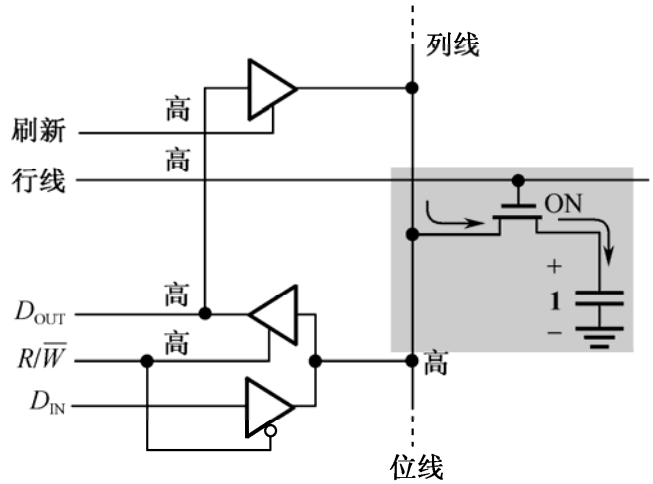

(a) 写 1 到存储位元




(b) 写 0 到存储位元



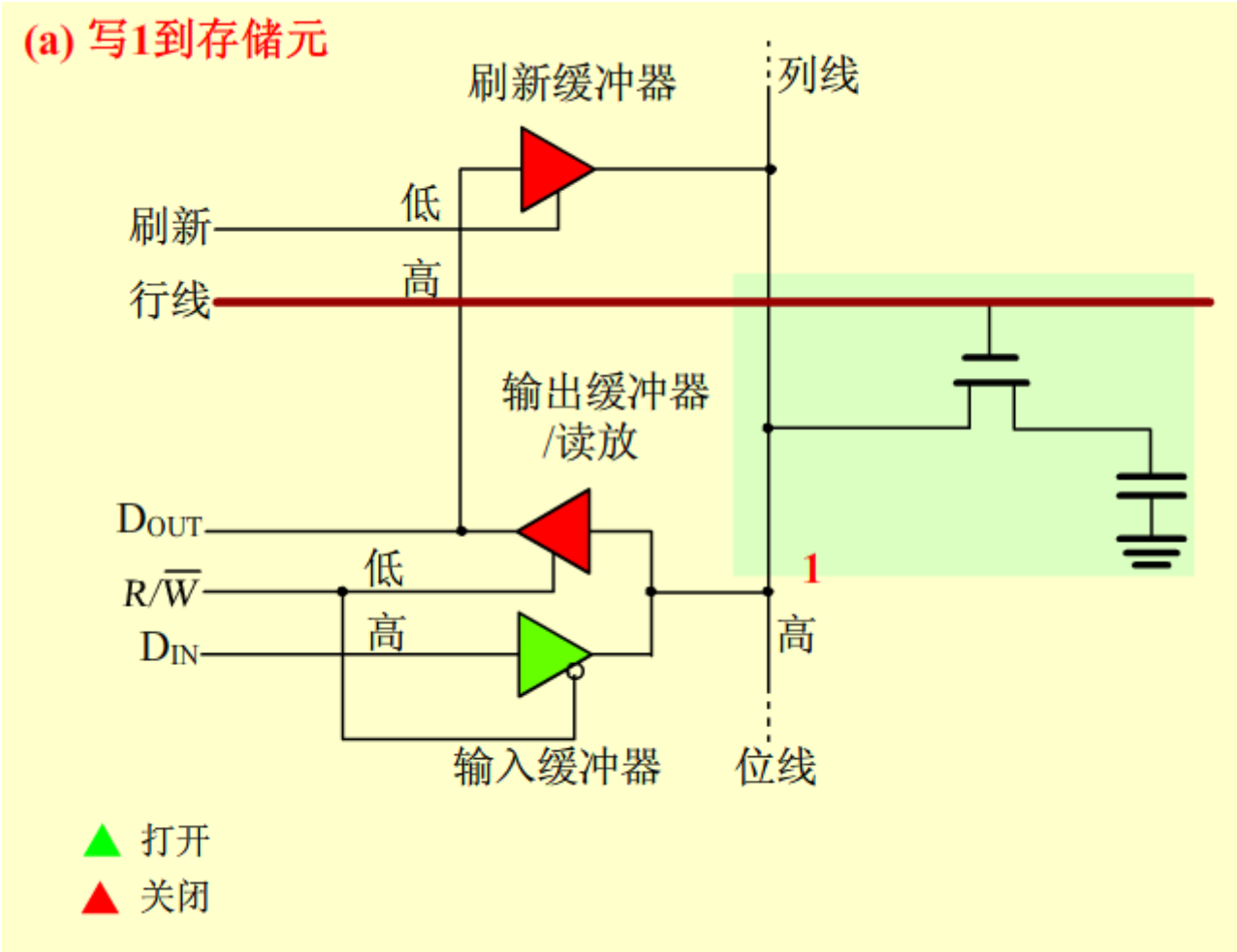

(c) 从存储位元读出 1




(d) 刷新存储位元的 1

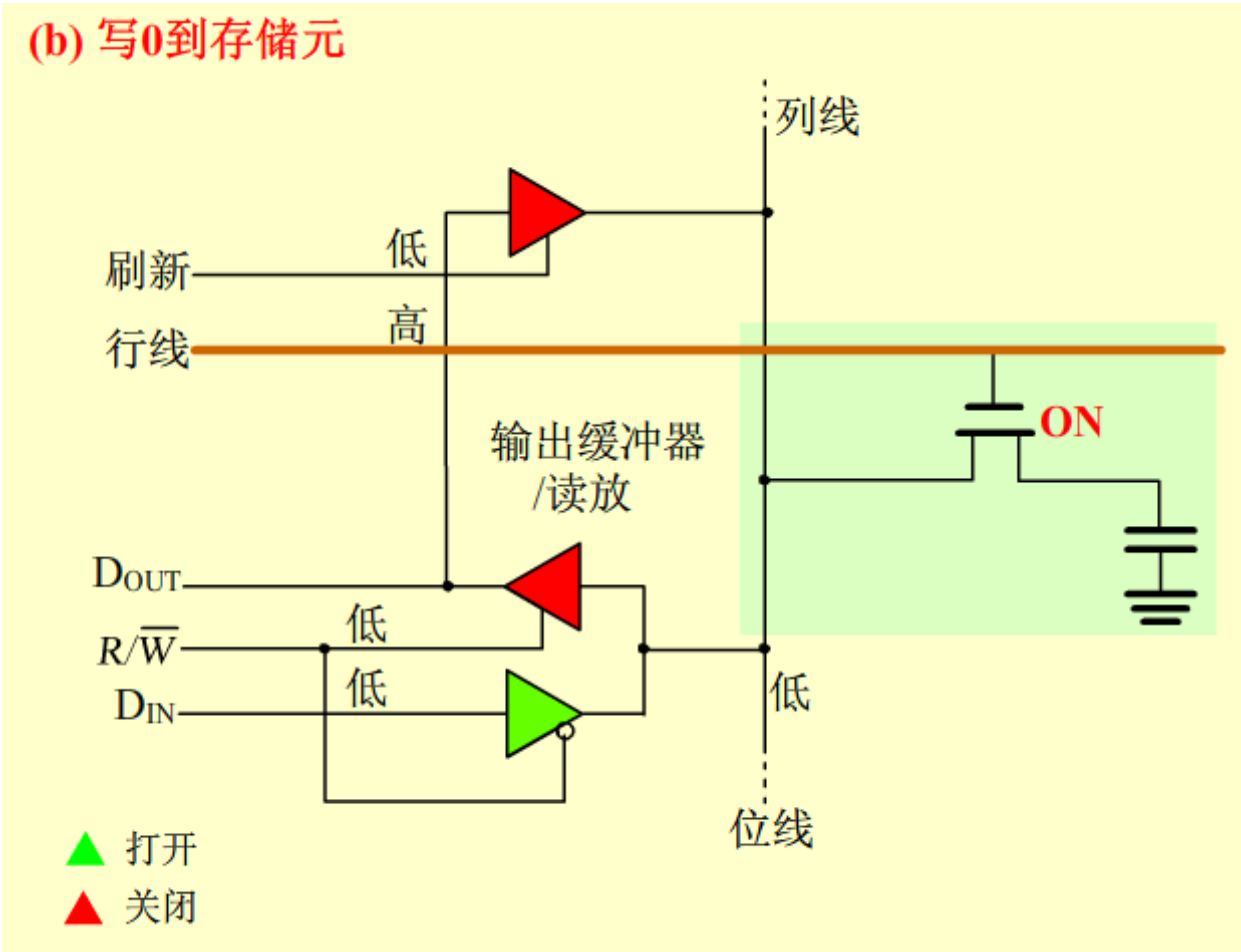
# 3.3.1 DRAM存储位元的记忆原理

- **DRAM**写1到存储位元。
  - 输出缓冲器关闭、刷新缓冲器关闭，输入缓冲器打开（R/W为低），输入数据DIN=1送到存储元位线上，而行选线为高，打开MOS管，于是位线上的高电平给电容器充电，表示存储了1。//了解



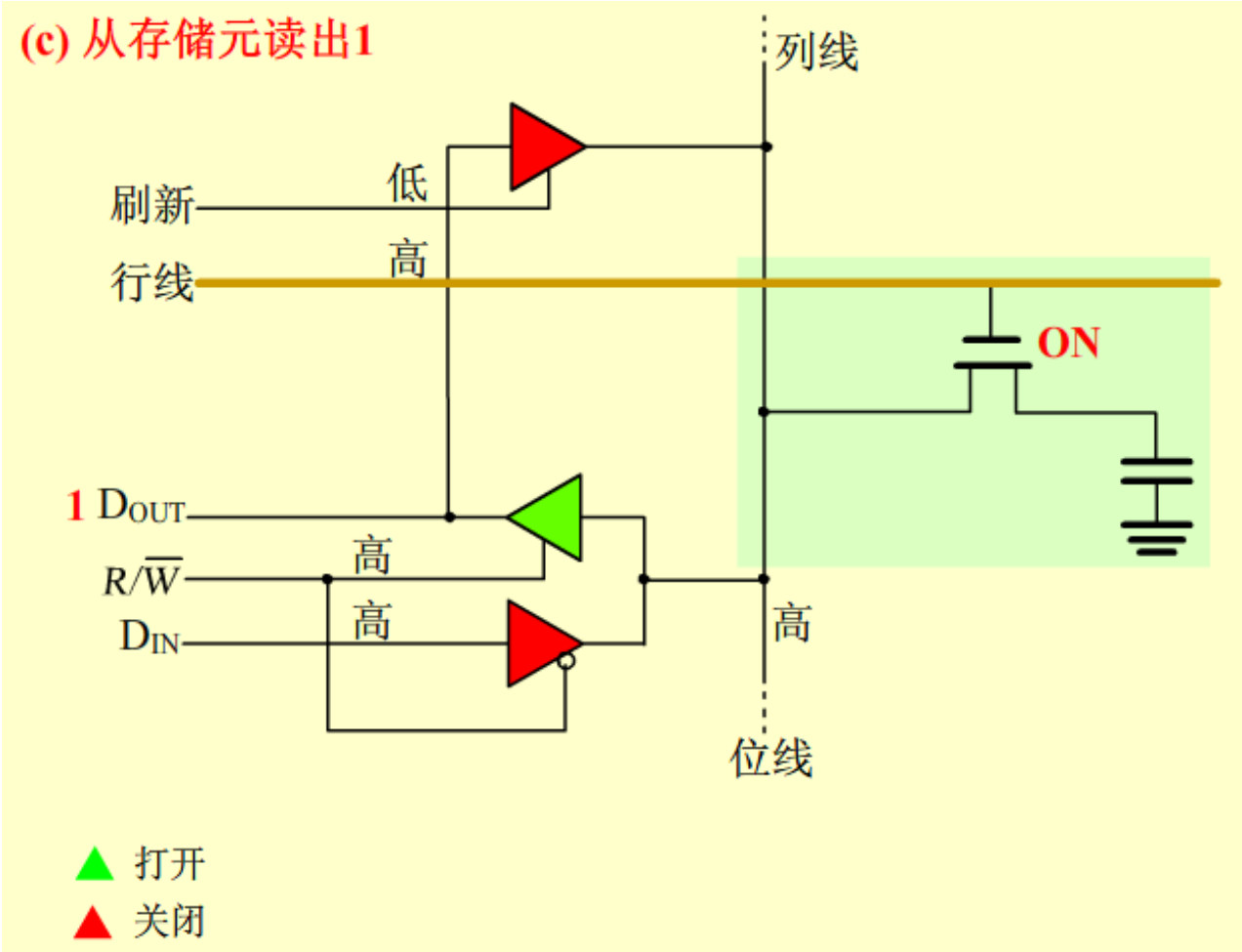
# 3.3.1 DRAM存储位元的记忆原理

- **DRAM**写0到存储位元。
  - 输出缓冲器和刷新缓冲器关闭，输入缓冲器打开，输入数据DIN=0送到存储元位线上；行选线为高，打开MOS管，于是电容上的电荷通过MOS管和位线放电，表示存储了0。 //了解



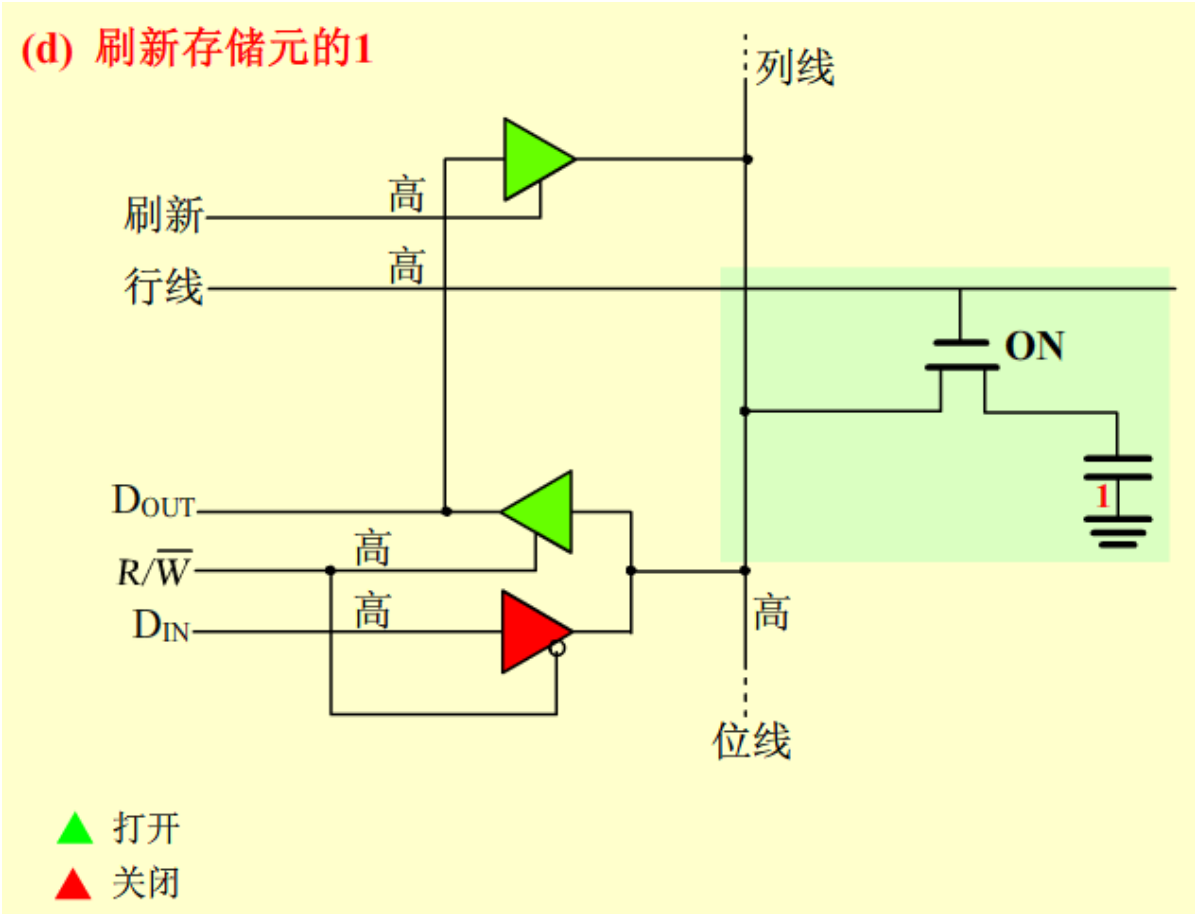
# 3.3.1 DRAM存储位元的记忆原理

- **DRAM**从存储位元读出1。
  - 输入缓冲器和刷新缓冲器关闭，输出缓冲器/读放打开（R/W为高）。行选线为高，打开MOS管，电容上所存储的1送到位线上，通过输出缓冲器/读出放大器发送到$D_{OUT}$,即$D_{OUT}=1$。 //了解



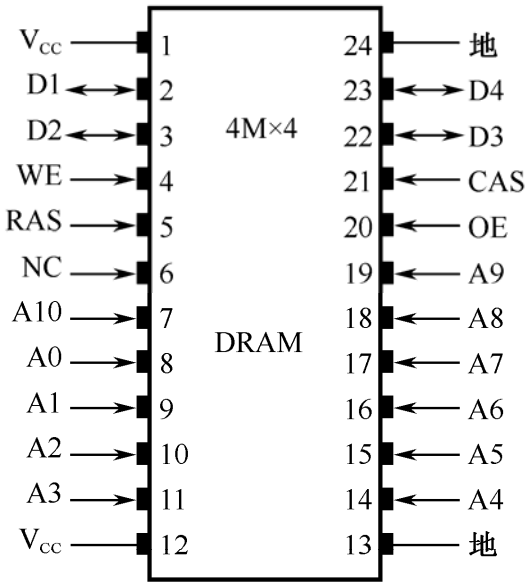
# 3.3.1 DRAM存储位元的记忆原理

- **DRAM**读出1后在存储位元重写1。
  - 由于读出1是破坏性读出，必须做恢复。输入缓冲器关闭，刷新缓冲器打开，输出缓冲器/读放打开，$D_{OUT}=1$经刷新缓冲器送到位线上，再经MOS管写到电容上。//了解

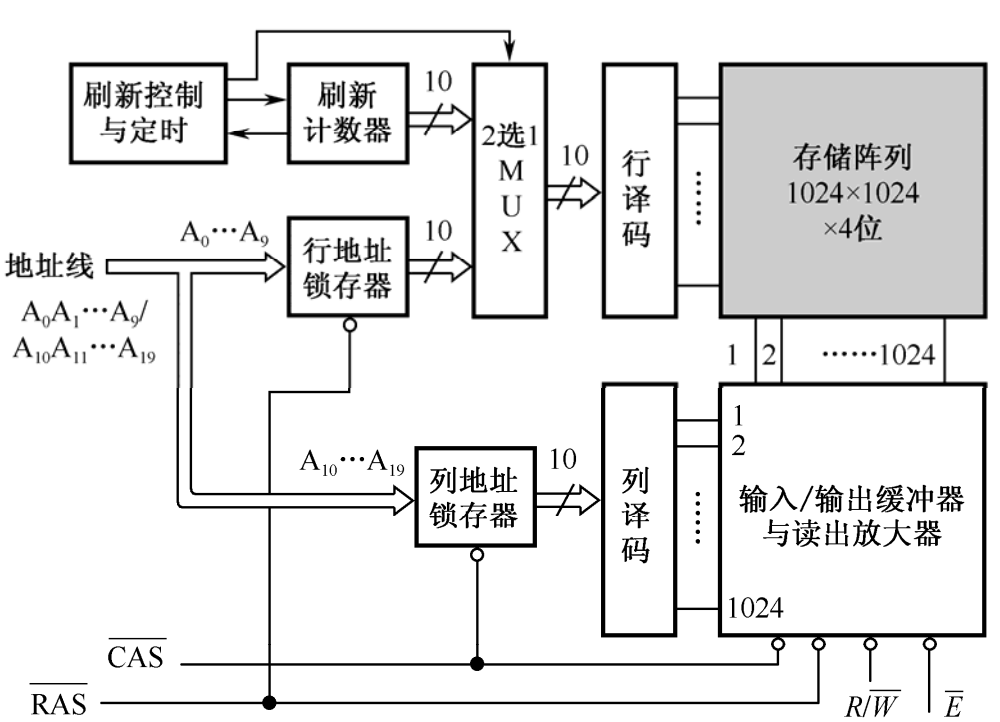


# 3.3.2 DRAM芯片逻辑结构

- 1M×4位DRAM芯片的管脚图及逻辑结构图
  - **1M个存储单元**，存储单元**字长是4位**；需要20位地址$A_0$~$A_{19}$，4位数据线；
  - 两个电源脚、两个地线脚，一个空脚（NC）。
  - **RAS/CAS**：行/列选通信号。控制地址线提供行/列地址。



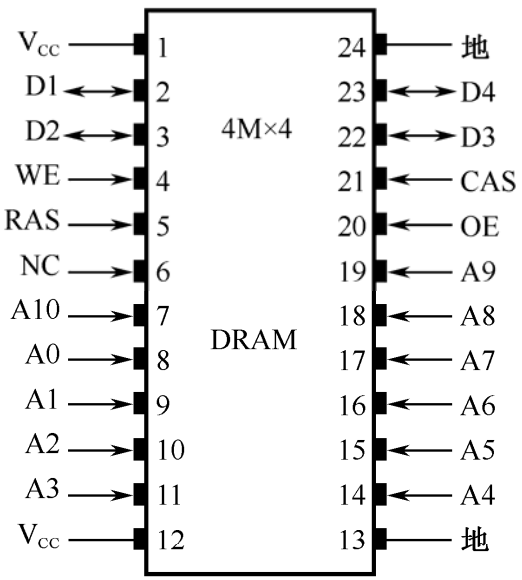

(a) 管脚图



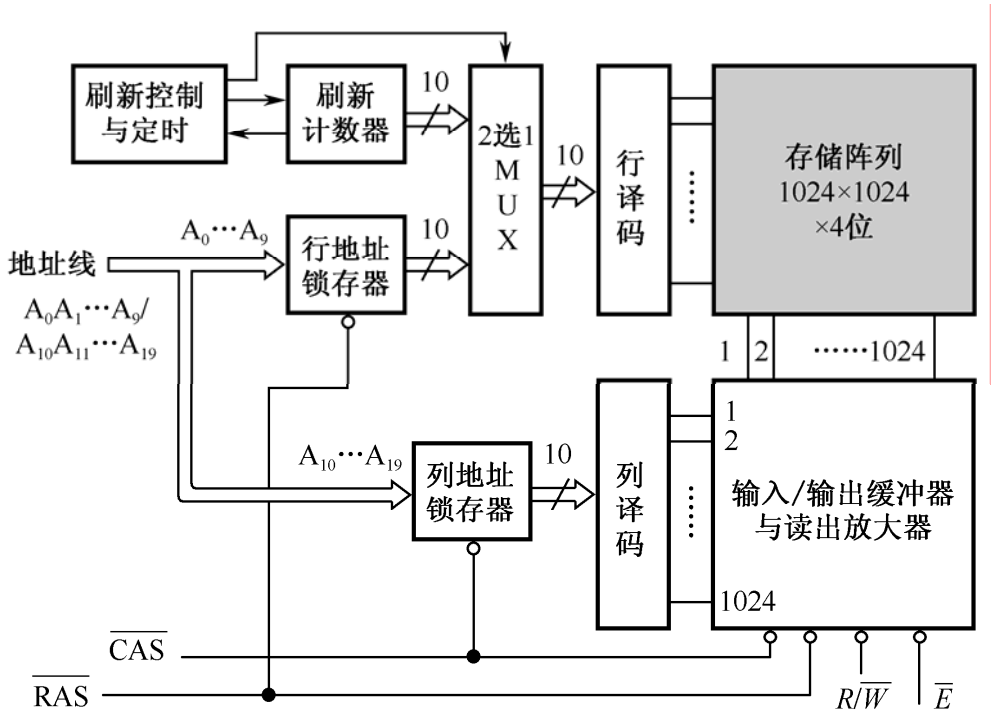

(b) 逻辑结构图

# 3.3.2 DRAM芯片逻辑结构

- 1M×4位DRAM芯片的管脚图及逻辑结构图
  - **$A_0$~$A_9$**：复用地址线；使用复用地址线两次，提供20位地址。
    - RAS有效/行选通，提供$A_0$~$A_9$。
    - CAS有效/列选通，提供$A_{10}$~$A_{19}$。
  - **$D_1$~$D_4$**：4位地址线。




(a) 管脚图




(b) 逻辑结构图

# 3.3.2 DRAM芯片逻辑结构

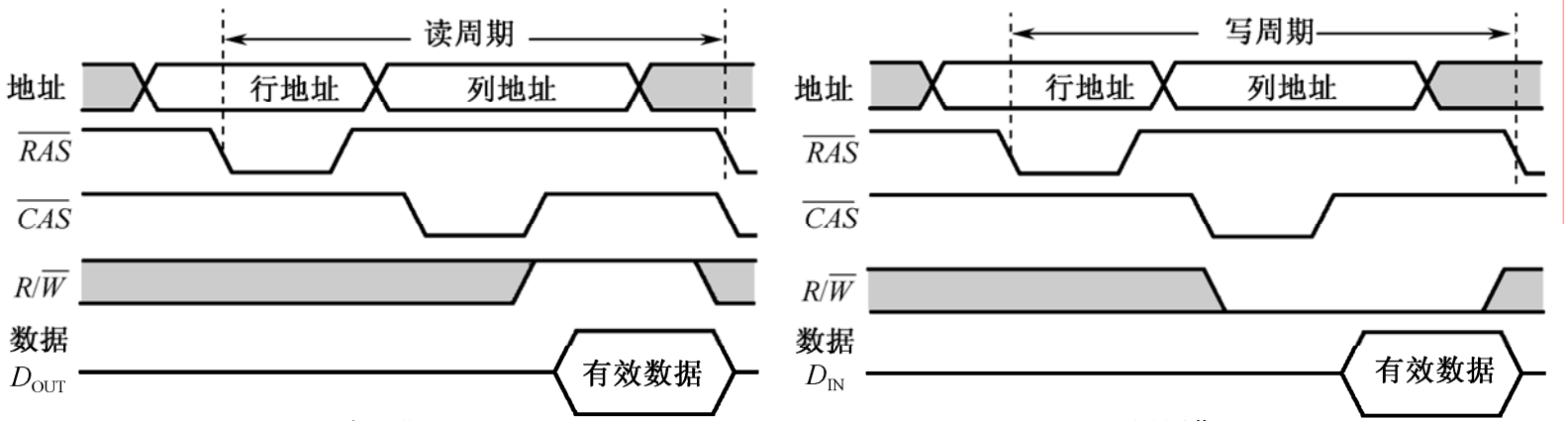
- DRAM与SRAM的不同：
  - 增加了行地址锁存器和列地址锁存器
    - 由于DRAM存储器容量很大，地址线宽度相应要增加，为减少DRAM芯片管脚数目，采取地址分时传送的方法：
      - 先使用10位地址线传送地址码$A_0$～$A_9$，使用行选通信号**RAS**将当前10位地址保存到行地址锁存器；
      - 再次使用10位地址线传送地址码$A_{10}$～$A_{19}$，使用列选通信号**CAS**将当前10位地址保存到列地址锁存器；
      - 将两部分地址组合成20位地址信号，可寻址存储单元数=1M。

# 3.3.2 DRAM芯片逻辑结构

- DRAM与SRAM的不同：
  - 增加了刷新计数器和相应的控制电路
    - DRAM存储元是CMOS电容，存在漏电问题。而且读操作是破坏性的，则DRAM存储器需要刷新操作。
  - 刷新操作有两种刷新方式：
    - 集中式刷新：DRAM的所有行在每一个刷新周期中都被刷新。
      - 例如刷新周期为8ms的内存来说，所有行的集中式刷新必须每隔8ms进行一次。
      - 一般将8ms时间分为两部分：前一段时间进行正常的读/写操作，后一段时间（8ms至正常读/写周期时间）做为集中刷新操作时间。
    - 分散式刷新：每一行的刷新插入到正常的读/写周期之中。
      - 如果DRAM存储器中有1024行，如果刷新周期为8ms，则每一行必须每隔8ms÷1024=7.8us进行一次。

# 3.3.3 读/写周期、刷新周期

- **读周期**：RAS下降沿→下一个RAS下降沿
  - 地址线行地址有效，RAS=0，保存行地址。
  - 地址线列地址有效，CAS=0，保存列地址。
  - 行/列译码器译码，R/W=1，$D_{OUT}$=读出有效数据。
- **写周期**： RAS下降沿→下一个RAS下降沿
  - 地址线行地址有效，RAS=0，保存行地址。
  - 地址线列地址有效，CAS=0，保存列地址。
  - 行/列译码器译码，R/W=0，$D_{IN}$=写入有效数据。



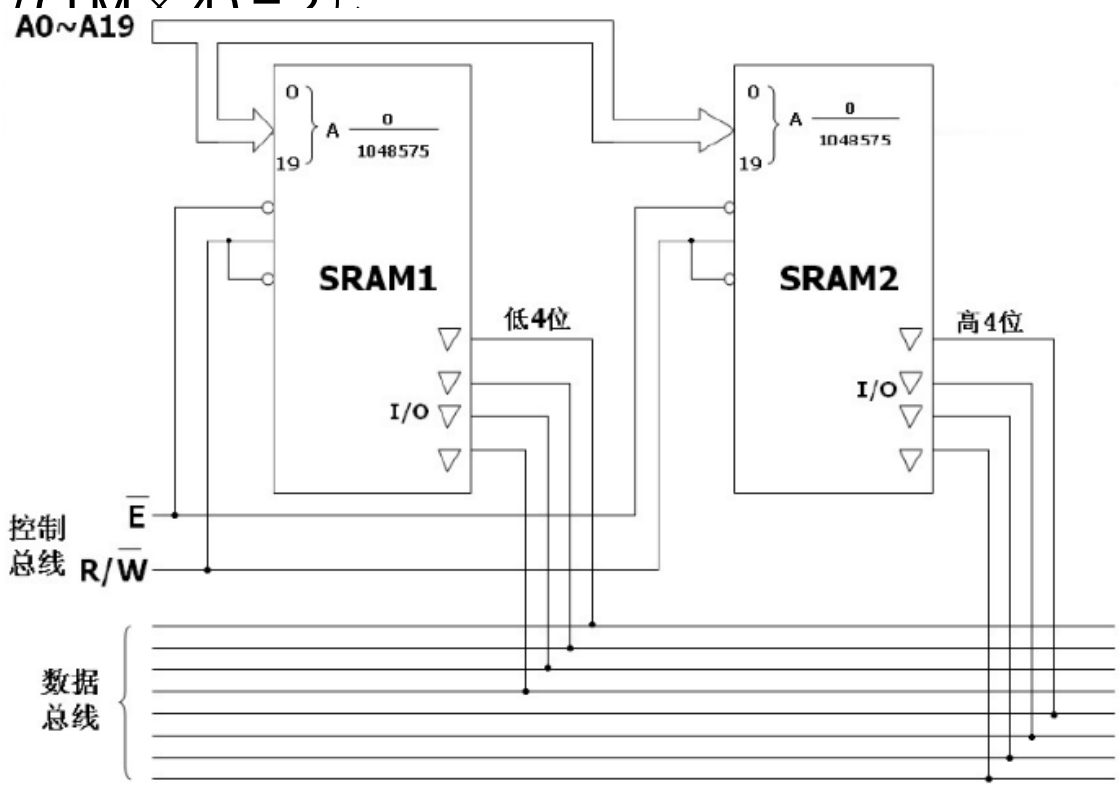

(a) 读周期　　(b) 写周期

# 3.3.4 存储器容量的扩充

- m×n的芯片构成M×N的存储器。
  - m/M：存储单元数；字数
  - n/N：字长位数；位数/字长
  - 需要的芯片数=（M×N）/（m×n）
- 位扩展：n→N
  - 给定的芯片字长位数较短，不满足设计要求，需要用多片芯片扩展字长位数。
  - 三组信号线/(地址，数据，控制)，数据线增加，地址线不变。
    - 地址线和控制线公用，同时连接所有芯片。
    - 数据线分组连接多位芯片。
- 字扩展：m→M
  - 给定的芯片存储单元较小（字数少），不满足设计要求，此时需要用多片给定芯片来扩展字数。
  - 三组信号线/(地址，数据，控制)，数据线不变，地址线增加。
    - 数据线公用。
    - 地址线分两部分使用：第1部分，低位地址线连接所有的芯片，用于芯片m的寻址。第2部分，高位地址线用于提供芯片的片选信号。

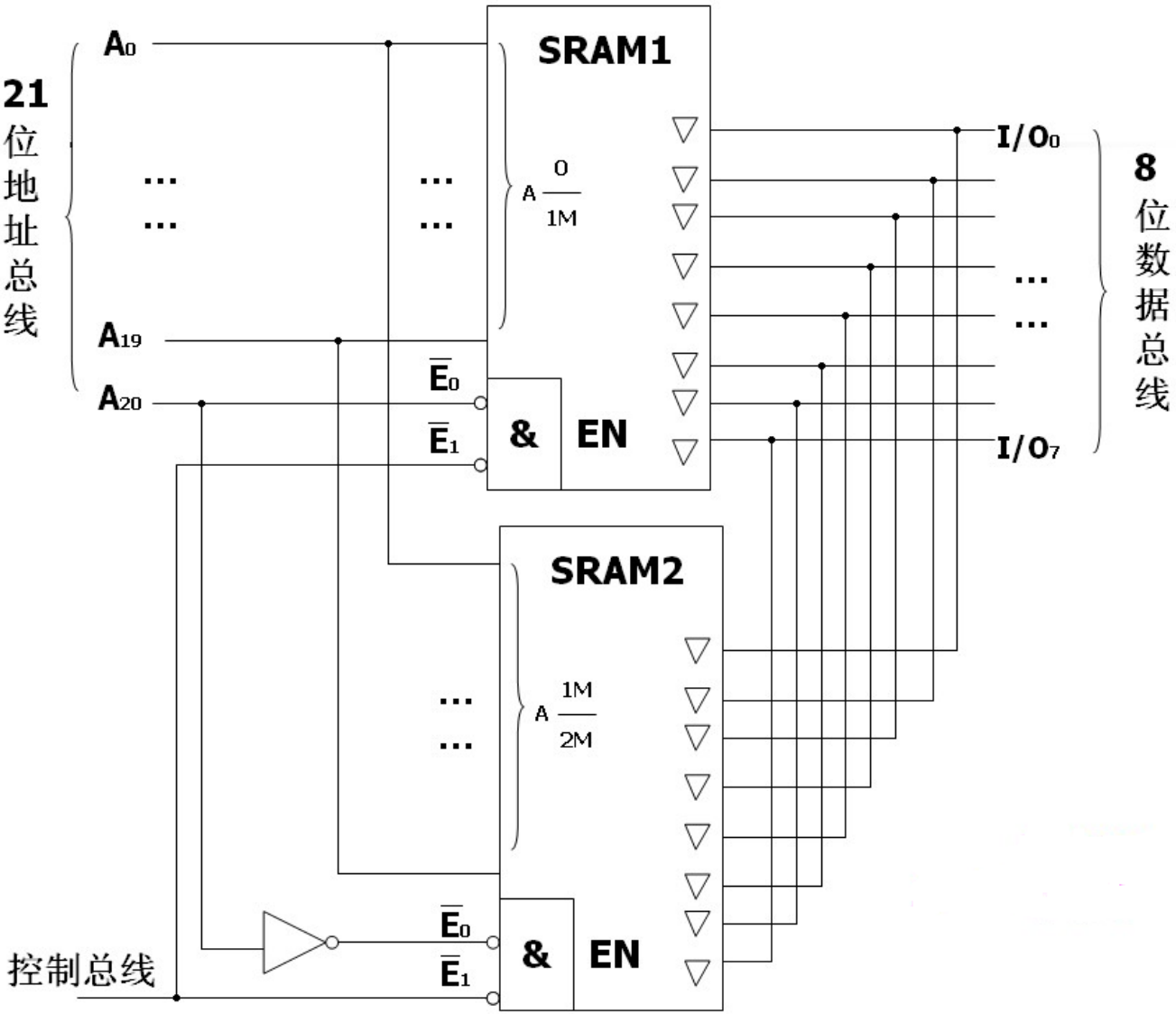
# 3.3.4 存储器容量的扩充

- 例：位扩展。利用**1M×4**位的SRAM芯片，设计一个存储容量为**1M×8**位的SRAM存储器。
- 解：字长增加为8位，存储器容量不变。
  - **1M×8**位的SRAM存储器需要数据线8位，地址线20位
  - **1M×4**位的SRAM芯片需要数据线4位，地址线20位
  - 所需芯片数量=(1M×8)/(1M×4)=2片
- 三组信号线连接：
  - 20位地址线、控制线公用，同时连接两片芯片。
  - 8位数据线分为高4位、低4位两组，分别连接两片 SRAM芯片的I/O端相连接。



# 3.3.4 存储器容量的扩充

- 例 存储容量为 2^21
- 解
  - 21 位地址总线
  -
  -
- 三
  -
  -
  -




地址总线会同时工作。

# 3.3.4 存储器容量的扩充

- 存储器模块
  - 主存通常以内存条形式生产销售，内存条一般是在一个条状形的小印制电路板上，用一定数量的存储器芯片，组成一个存储容量固定的存储模块。如图所示。
- 内存条有30脚、72脚、100脚、144脚、168脚等多种形式。
  - 30脚内存条设计成8位数据线，存储容量从256KB～32MB。
  - 72脚内存条设计成32位数据总线
  - 100脚以上内存条既用于32位数据总线又用于64位数据总线，存储容量从4MB～512MB。

# 3.3.4 存储器容量的扩充

- 例：字/位 扩展。利用256K×8位的DRAM芯片，设计一个存储容量为1M×16位的DRAM存储器。
- 解：
  - 位扩展8→16，字扩展 256K→1M
  - 1M×16位的DRAM存储器需要地址线20位，数据线16位。
  - 256K×8位的DRAM芯片需要地址线18位，数据线8位。
  - 需要的芯片数=(1M×16)/(256K×8)=8片。
  - 8片芯片分4组，每组2片。
    - Why?1片提供8位，组中的两片可以提供16位数据。
- 三组信号线连接：
  - 16位数据线同时连接4组，在每组中，分高8位和低8位，分别连接组内的两片芯片。
  - 地址总线$A_0$~$A_{19}$：
    - 低18位$A_0$~$A_{17}$：连接所有芯片，用于芯片内256K的寻址；
    - 高2位$A_{18}$~$A_{19}$：输入使用2线/4线的地址译码器，4个输出分别连接4个组的芯片片选信号；
  - 其他控制线公用，同时连接所有芯片。

# 3.3.4 存储器容量的扩充

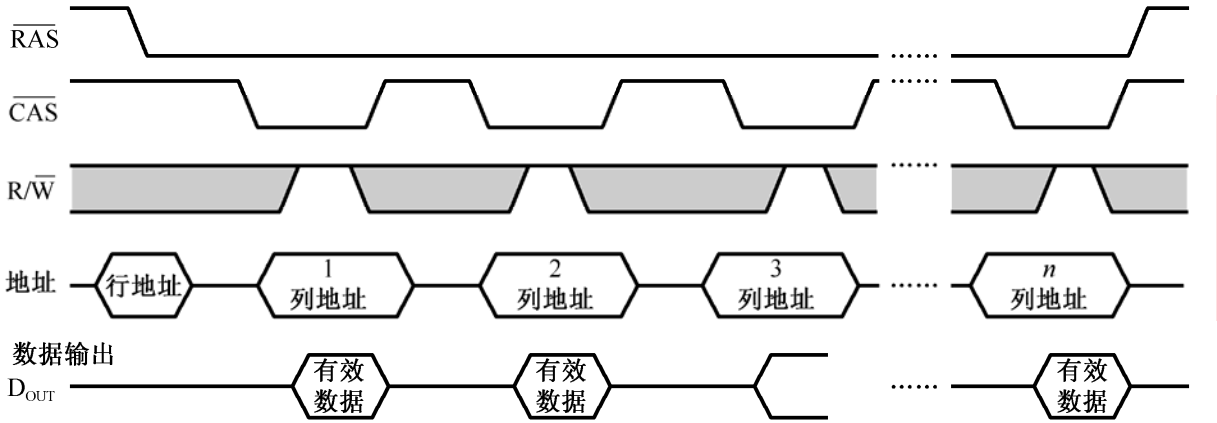
- 例：设有一个具有20位地址和32位字长的存储器，问：
  - 1、该存储器能存储多少个字节的信息？
  - 2、如果存储器由512K×8位SRAM芯片组成，需要多少片？
  - 3、需要多少为地址作为芯片选择?

- 例：已知**64位机** 主存采用半导体存储器，其地址码为26位，若使用4M×8位的DRAM芯片组成该机所允许的最大主存空间，并选用内存条结构形式，问：
  - 1、若每个内存条为16M×64位，共需要几根内存条？
  - 2、每个内存条共有多少DRAM芯片？
  - 3、主存共需要多少DRAM芯片？
  - 提示：主存容量=$2^{26}$×64=128M×64 位

# 3.3.5 高级DRAM结构

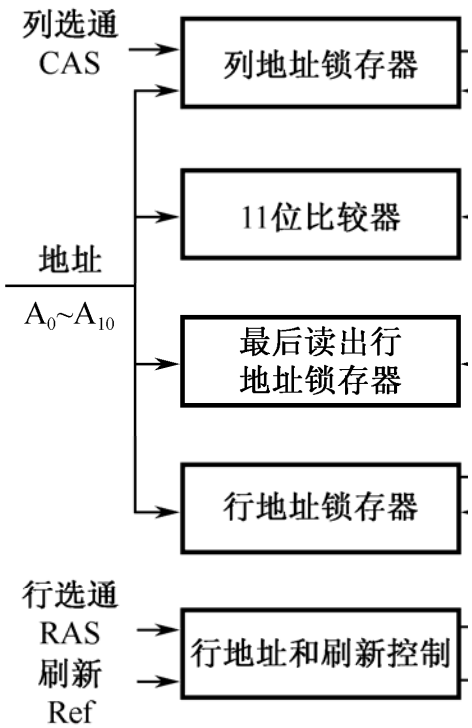
- 传统的DRAM芯片局限性：读写速度慢
- 解决思路：
  - 提高时钟频率和带宽，缩短存取时间，增加芯片的引脚等。

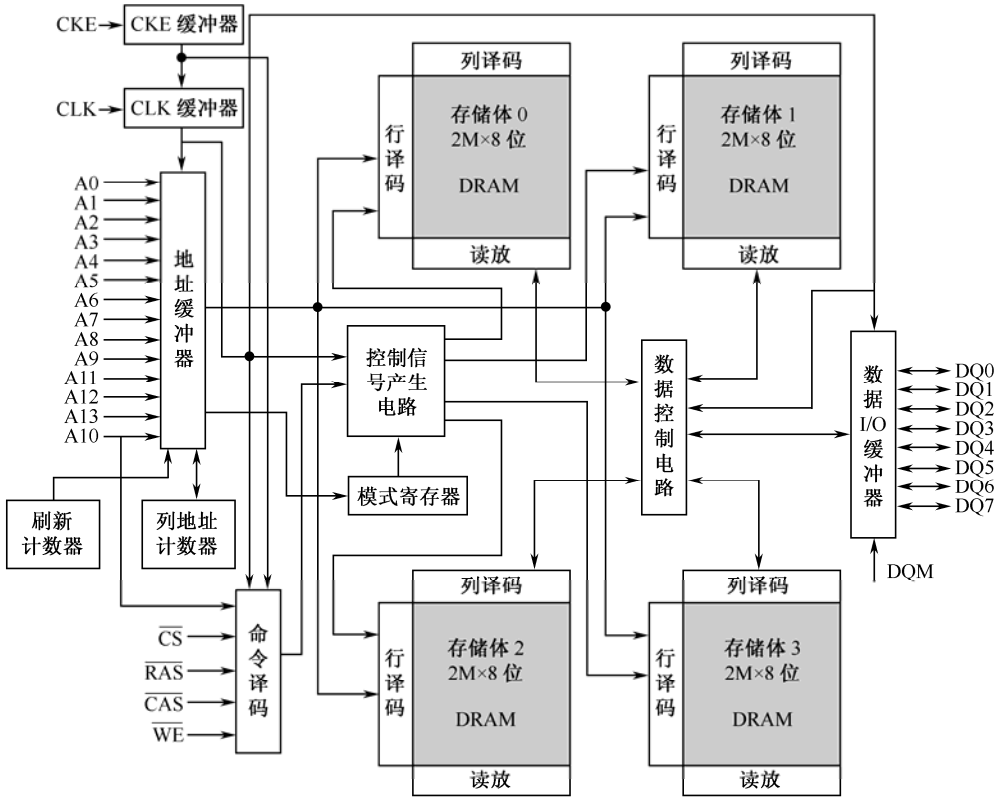
- 高级DRAM芯片：
  - **FPM-DRAM**：快速页模式动态存储器。
    - 根据程序的局部性原理来实现的。读周期和写周期中，为了寻找一个确定的存储单元地址，首先由低电平的行选通信号RAS确定行地址/页地址，然后由低电平的列选信号CAS确定列地址/页中的位置。下一次寻找操作，RAS保持，CAS变化选的不同的保存位置。如图



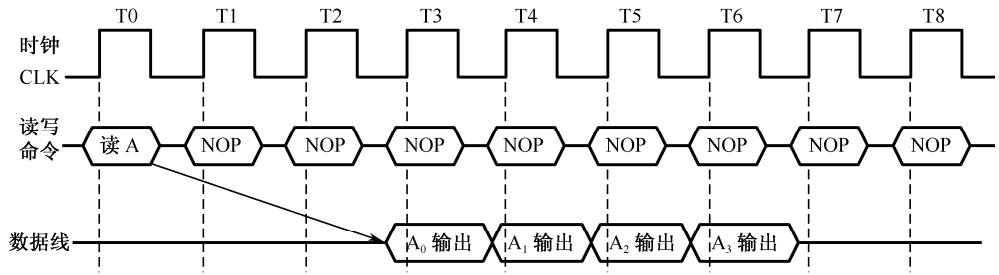
# 3.3.5 高级DRAM结构

- 高级DRAM芯片：
  - **CDRAM**：带高速
    - 通常的DRAM芯片
    芯片的性能得到
    片的结构框图，
  -



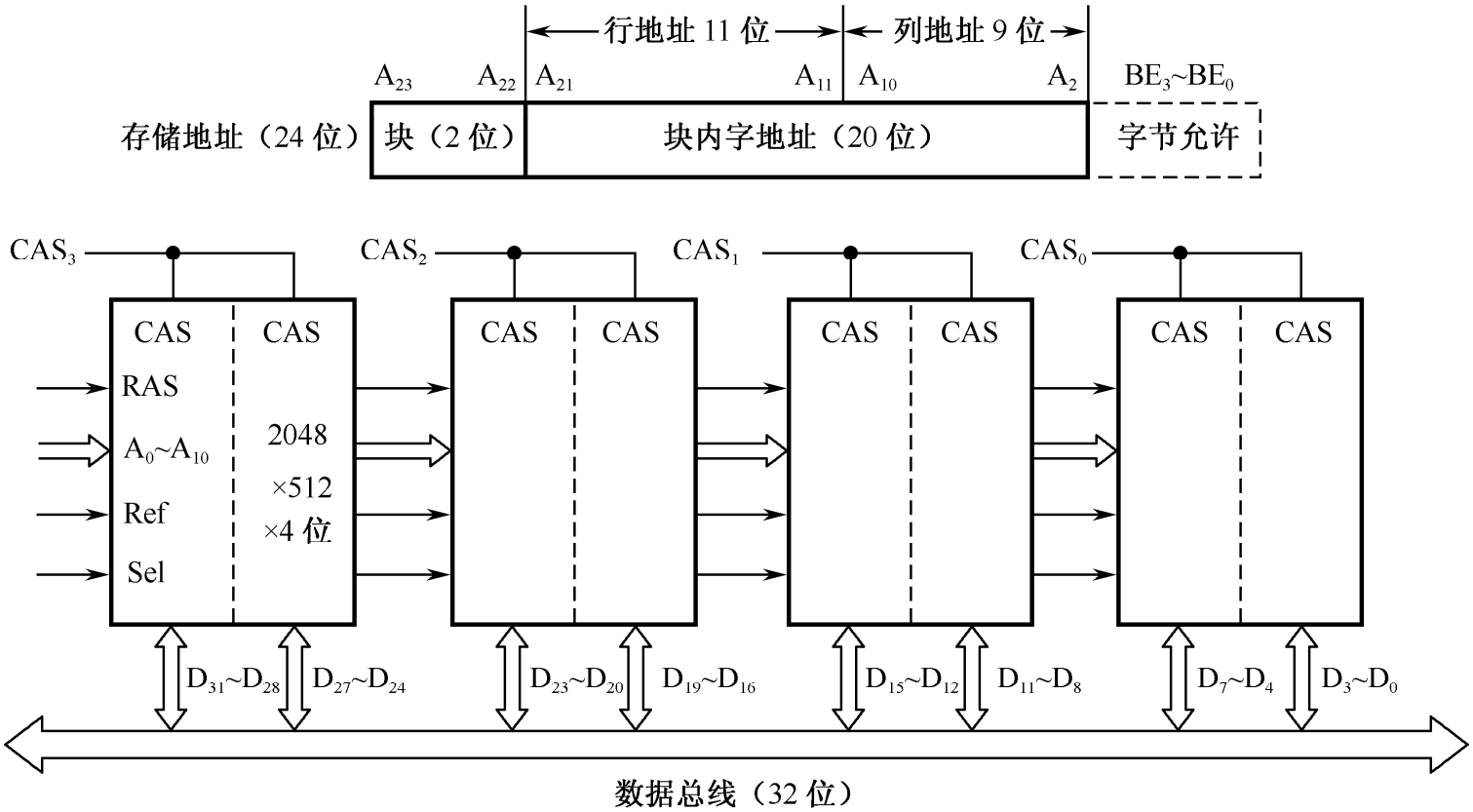

(a) SDRAM 内部结构




(b) SDRAM 读操作时序（猝发长度 =4　CAS 延时 =2）

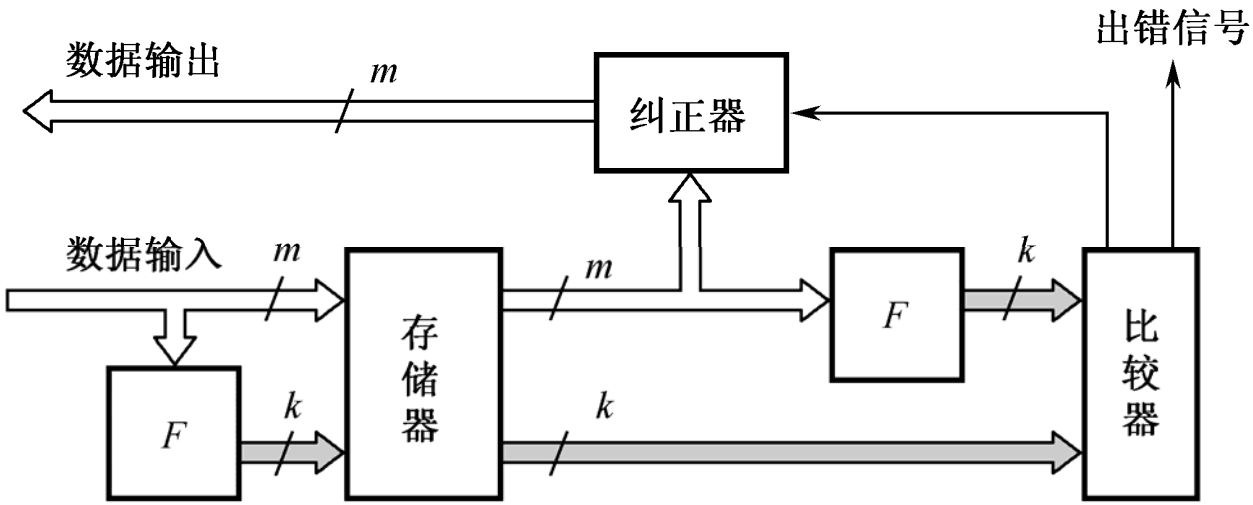
# 3.3.5 高级DRAM结构

- 例4 CDRAM内存条组成实例 ：
  - 一片CDRAM的容量为1M×4位，8片这样的芯片可组成1M×32位4MB的存储模块，其组成如下图所示。P76 。



# 3.3.6 DRAM读/写的正确性校验

- DRAM通常用做主存储器，其读写操作的正确性与可靠性至关重要。为此除了正常的数据位宽度，还增加了附加位，用于读/写操作正确性校验。增加的附加位也要同数据位一起写入DRAM中保存。其原理如图所示。
  - F为奇偶检验异或电路。如果读写正确，数据写入存储器前和读出存储器后两部分F运算结果应该相同，否则，错误。
  - 奇偶检验：增加1位附加位做检验。只能检测，无法纠正。
  - 汉明码检验。纠正码的要求：P78 表3.2



# 3.4 只读存储器与闪速存储器 / 了解

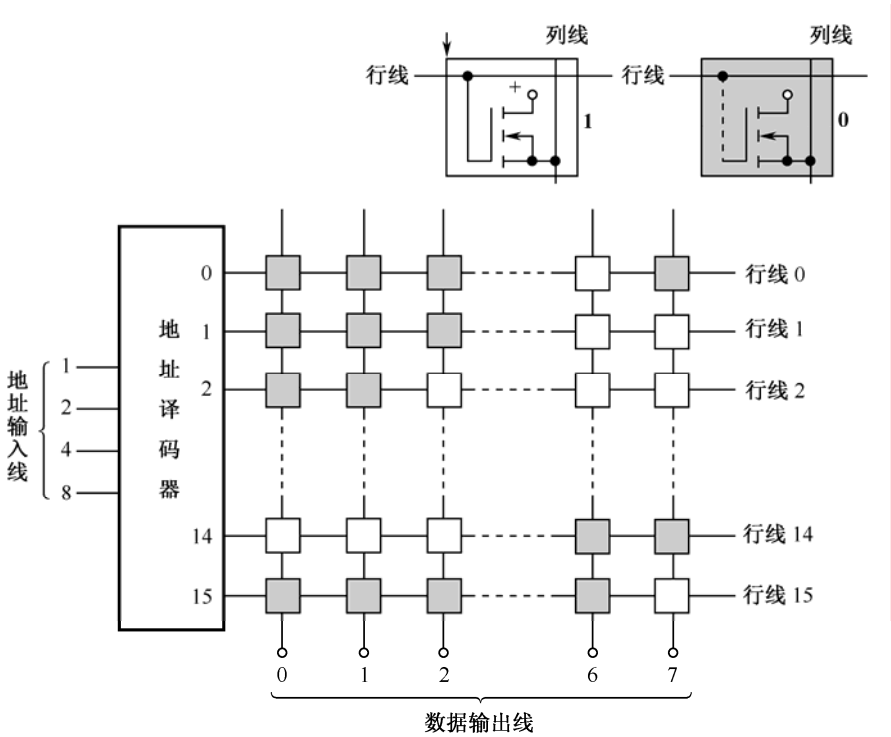
- SDRAM/DRAM是随机读写存储器，数据可读可写。

## 3.4.1 只读存储器ROM / 了解

- ROM叫做只读存储器。数据只能读出，不能写入，其存储的原始数据，必须在它工作以前写入。主要有两类：
  - **掩模ROM**：掩模ROM实际上是一个存储内容固定的ROM，由生产厂家提供产品。
  - **可编程ROM**：用户后写入内容，有些可以多次写入。
    - 一次性编程的PROM
    - 多次编程的EPROM和$E^2$PROM

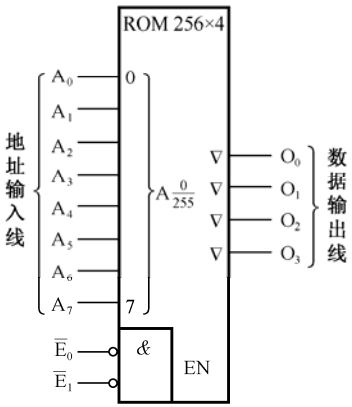
# 3.4.1 只读存储器ROM / 了解

- 掩模ROM / MROM
  - 掩模ROM的阵列结构和存储元
    - 16×8位的ROM阵列结构。16个字，字长8位。
    - 地址线4位，数据线8位。对于某个确定的4位地址，可以唯一的确定1行数据，进行读写操作。示意图中，白色MOS管存储1，黑色MOS管存储0。

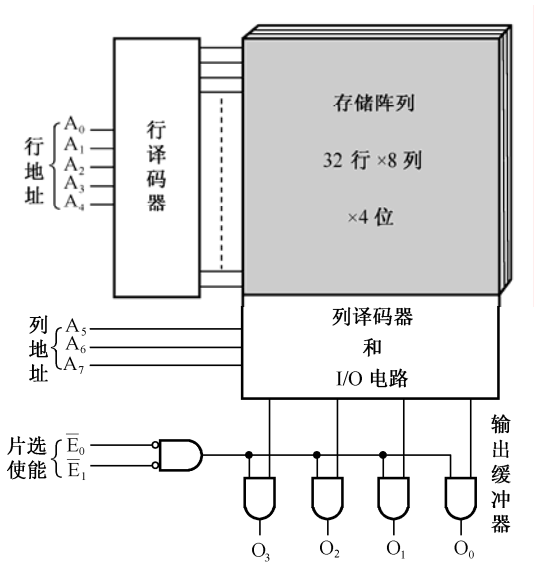


# 3.4.1 只读存储器ROM / 了解

- 掩模ROM / MROM
  - 掩模ROM的逻辑符号和内部逻辑框图
    - 32×8×4位的ROM阵列结构。32×8个字，字长4位。
    - 地址线8位，数据线4位。对于某个确定的8位地址，可以唯一的确定1个存储单元，进行读写操作。
    - 行地址5位，列地址3位，行地址和列地址在32×8存储阵列中确定唯一一个有效的存储单元。




(a) 掩模 ROM 逻辑符号




(b) 内部逻辑框图

# 3.4.1 只读存储器ROM / 了解

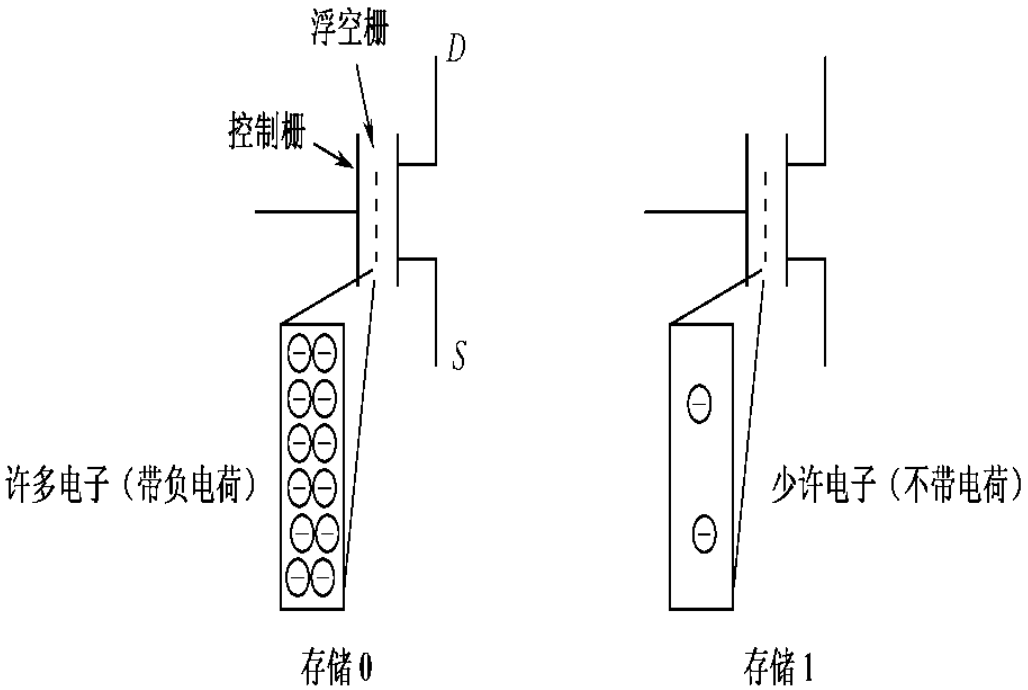
- 可编程**ROM**
  - PROM（一次性编程ROM）
    - 允许用户进行一次性编程
  - EPROM（可擦除可编程ROM）
    - 紫外光擦除、并可重复编程的ROM
  - EEPROM（电擦除可编程ROM） **$E^2$PROM**
    - 擦除和编程（擦写）通过加电进行

# 3.4.2 FLASH存储器 / 了解

- FLASH存储器也翻译成闪速存储器，它是高密度非失易失性的读/写存储器。
  - 高密度意味着它具有巨大比特数目的存储容量。
  - 非易失性意味着存放的数据在没有电源的情况下可以长期保存。
  - 它既有RAM的优点，又有ROM的优点，称得上是存储技术划时代的进展。

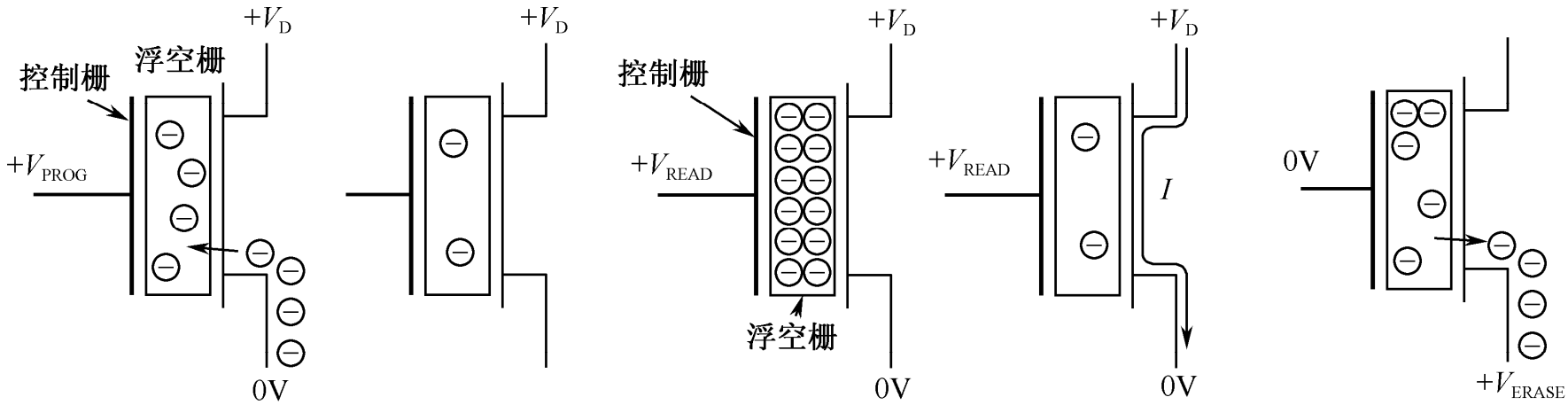
# 3.4.2 FLASH存储器 / 了解

- **FLASH存储元 / 了解**
  - 在EPROM存储元基础上发展起来的，增加了芯片的电擦除和重新编程能力,由此可以看出创新与继承的关系。
  - 如右图所示为闪速存储器中的存储元，由单个MOS晶体管组成，除漏极D和源极S外，还有一个控制栅和浮空栅。



# 3.4.2 FLASH存储器 / 了解

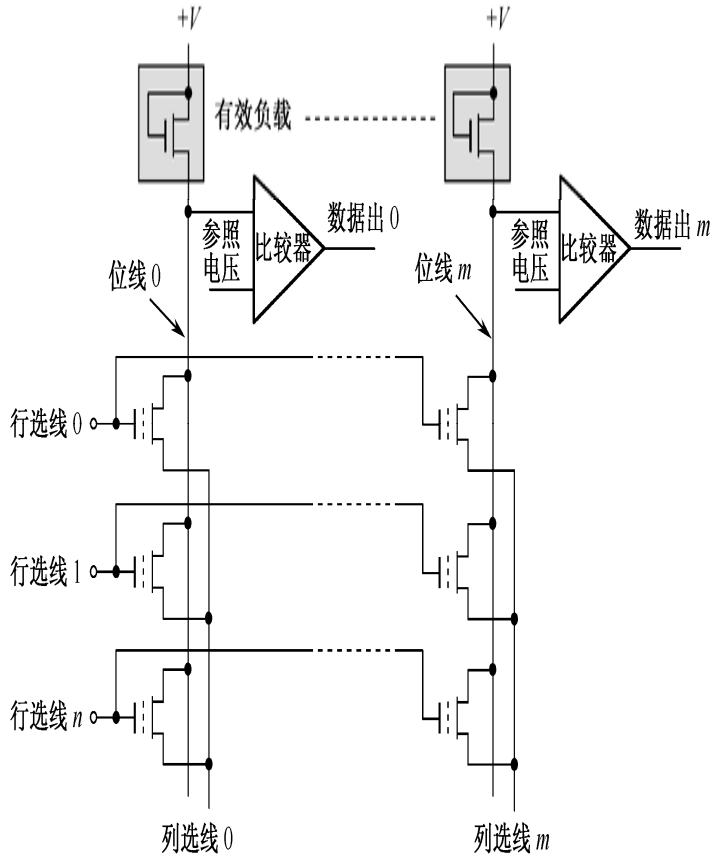
- **FLASH存储器基本操作**
  - 编程操作：写操作→控制栅C加正电压，写0，否则默认存1；
  - 读取操作：读操作→控制栅C加正电压，浮空栅确定读0/1；
  - 擦除操作：电擦出方式；




(a) 编程操作　(b) 读出操作　(c) 擦除操作

# 3.4.2 FLASH存储器 / 了解

- **FLASH存储器阵列结构**
  - FLASH存储器的阵列结构如图所示。
  - 在某一时间只有一条行选择线被激活。
  - 读操作时，假定某个存储元原存1，那么晶体管导通，与它所在位线接通，有电流通过位线，所经过的负载上产生一个电压降。这个电压降送到比较器的一个输入端，与另一端输入的参照电压做比较，比较器输出一个标志为逻辑1的电平。如果某个存储元原先存0，那么晶体管不导通，位线上没有电流，比较器输出端则产生一个标志为逻辑0的电平。

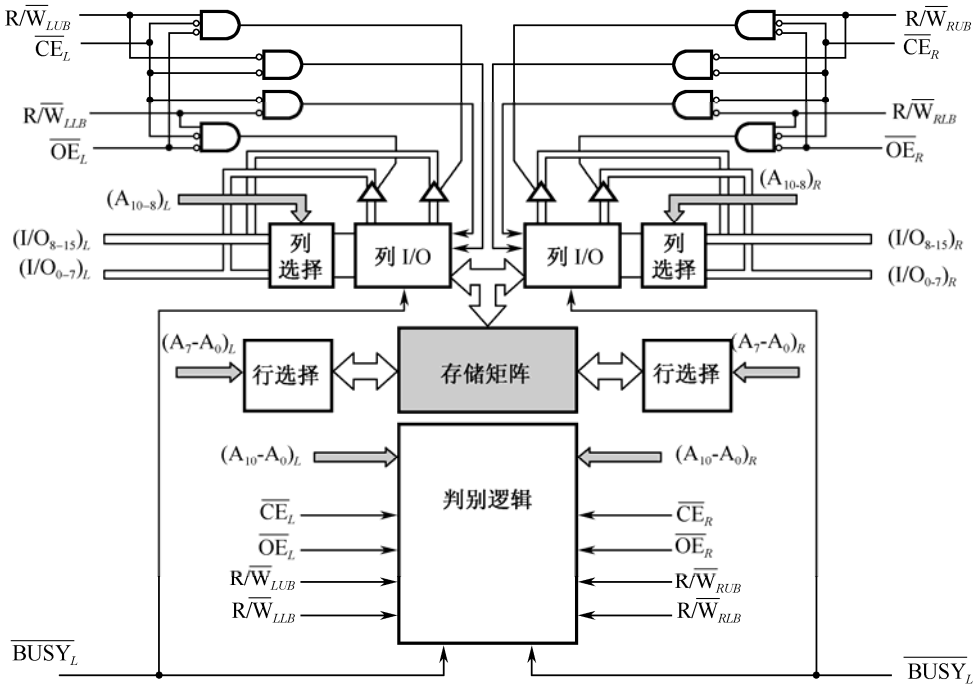


# 3.5 并行存储器

- 解决速度问题
  - 由于CPU和主存储器之间在速度上是不匹配的，这种情况便成为限制高速计算机设计的主要问题。为了提高CPU和主存之间的数据传输率，除了主存采用更高速的技术来缩短读出时间外，还可以采用并行技术的存储器。
  - 空间并行技术：双端口技术
  - 时间并行技术：多模态交叉技术

# 3.5.1 双端口存储器

- 双端口存储器的逻辑结构
  - 双端口存储器：同一个存储器具有两组相互独立的读写控制电路。可以进行并行的读写操作，是一种高速工作的存储器。
  - 双端口存储器IDT7133的逻辑框图如图。
    - 2K×16位SRAM。有左右两组端口，具有独立的地址线$A_0$~$A_{10}$，数据线$I/O_0$~$I/O_{15}$，和控制线(R/W,CE,OE,BUSY)

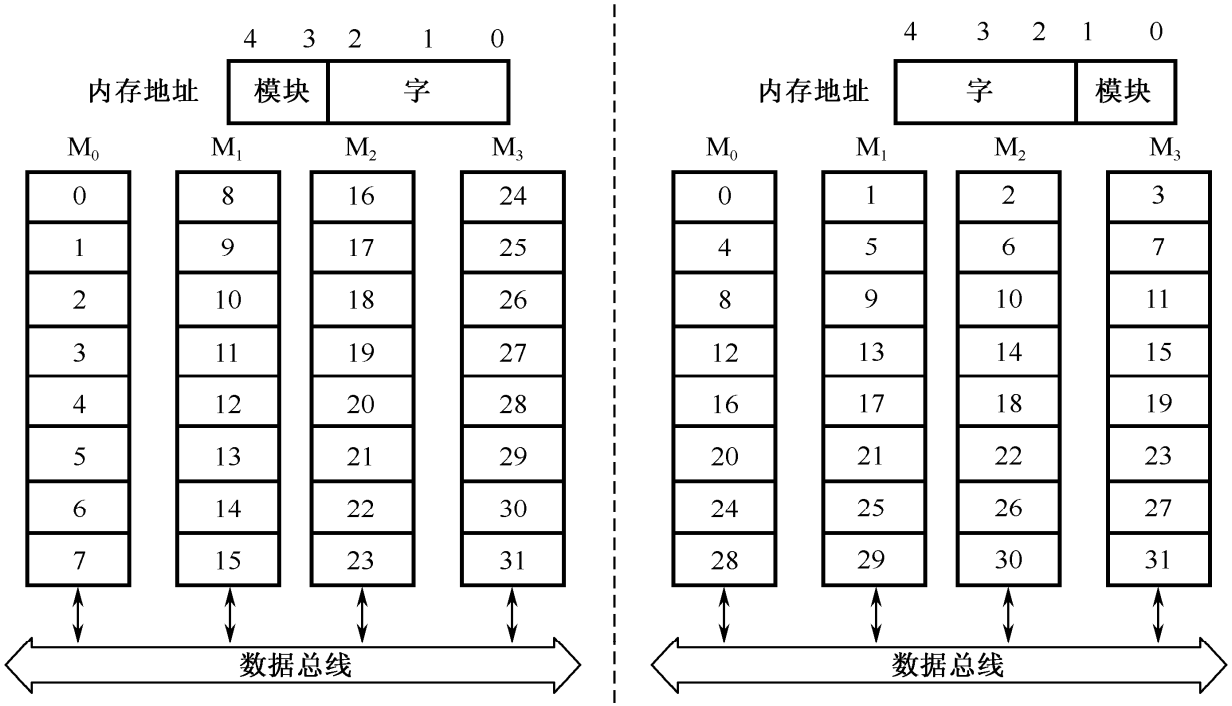


# 3.5.1 双端口存储器

- 无冲突读写控制
  - 两个端口的地址不相同时，在两个端口上进行读写操作，一定不会发生冲突。
    - 当任一端口被选中驱动时，就可对整个存储器进行存取，每一个端口都有自己的片选控制(CE)和输出驱动控制(OE)。读操作时，端口的OE(低电平有效)打开输出驱动器，由存储矩阵读出的数据就出现在I/O线上。
    - 无冲突读写控制如表3.4 P85
- 有冲突读写控制
  - 当两个端口地址相同时，便发生读写冲突。
    - 使用BUSY标志为解决此问题。在这种情况下，片上的判别逻辑可以决定对哪个端口优先进行读写操作，而对另一个被延迟的端口置BUSY标志(BUSY变为低电平)，即暂时关闭此端口。
    - 左右端口读写读写操作功能判断 如表3.5 P86 判别电路
  -

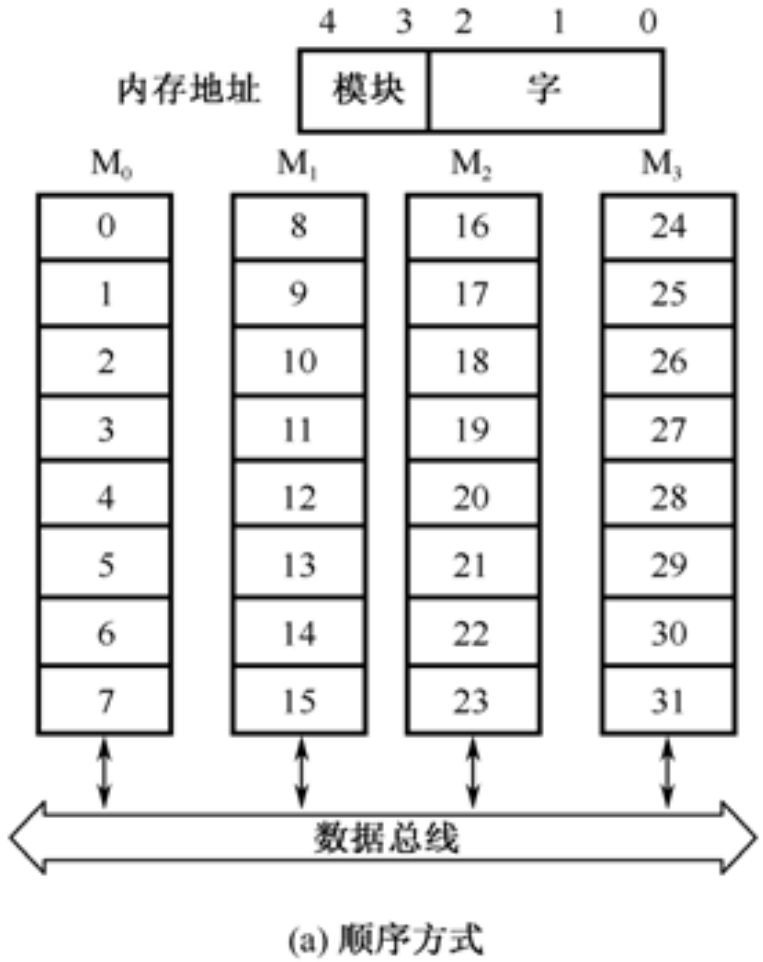
# 3.5.2 多模块交叉存储器

- 存储器的模块化组织
  - 一个由若干个模块/存储单元组成的主存储器是线性编址的。为模块分配地址可以行优先或列优先，即分配地址有两种方式：
    - 顺序方式：常规方式，列优先，每个模块的地址依次分配。
    - 交叉方式：行优先，所有模块地址同时分配。



# 3.5.2 多模块交叉存储器

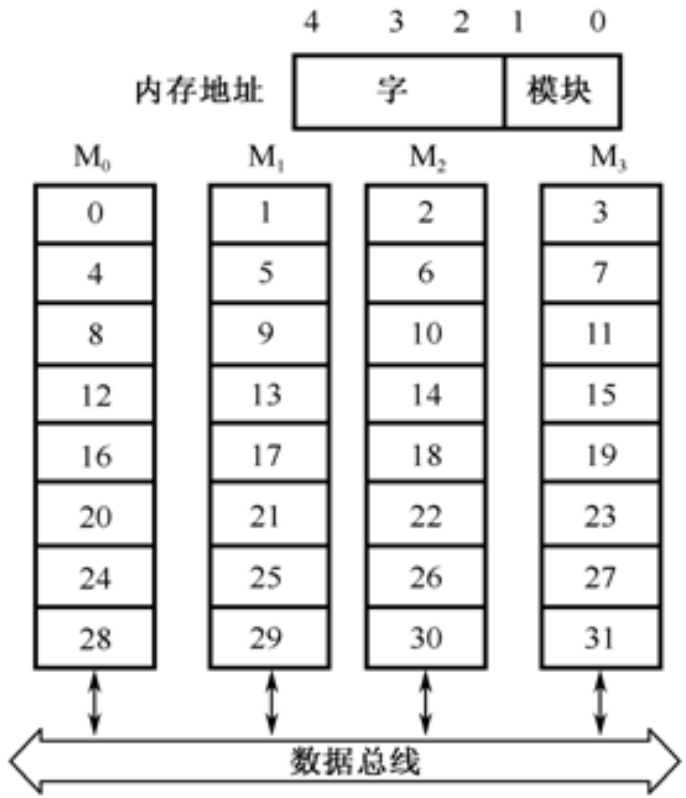
- 顺序方式
  - 使用模块按顺序方式组织容量为32个字的存储器。
  - 设计方案：
    - 32个字分4个模块$M_0$，$M_1$，$M_2$，$M_3$，每个模块8个字。
    - 访问地址按顺序方式依次分配给每个模块中的每个字。
    - 需要**5位地址**，**高2位**用于选择模块**低3位**用于选择模块中的字。
    - 特点：某个模块进行存取时，其他模块不工作，优点是某一模块出现故障时，其他模块可以照常工作，通过增添模块来扩充存储器容量比较方便。缺点是各模块串行工作，存储器的带宽受到了限制。




(a) 顺序方式

# 3.5.2 多模块交叉存储器

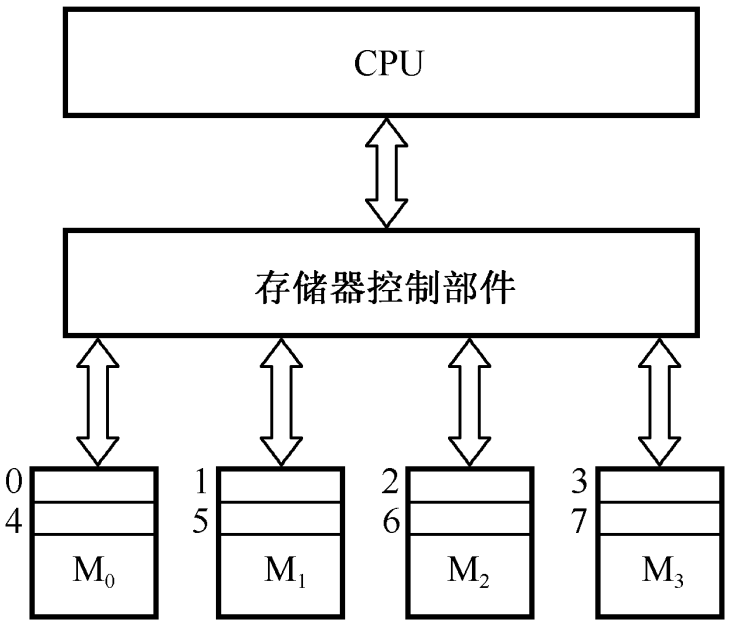
- 交叉方式
  - 使用模块按顺序方式组织容量为32个字的存储器。
  - 设计方案：
    - 32个字分4个模块$M_0$，$M_1$，$M_2$，$M_3$，每个模块8个字。
    - 访问地址按顺序方式依次分配给4个模块中的每个字。
    - 需要**5位地址**，**低2位**用于选择模块，**高3位**用于选择模块中的字。
    - 特点：连续地址分布在相邻的不同模块内，同一个模块内的地址都是不连续的。优点是对连续字的成块传送可实现多模块流水式并行存取，大大提高存储器的带宽。使用场合为成批数据读取。

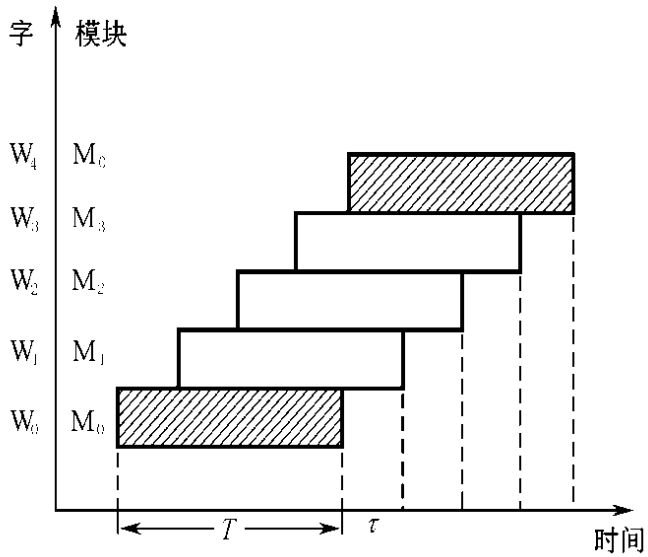


(b) 交叉方式

# 3.5.2 多模块交叉存储器

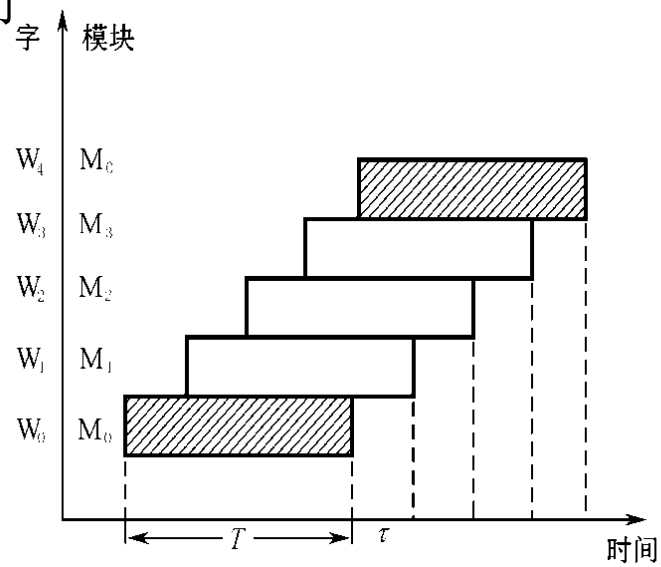
- 多模块交叉存储器的基本结构
  - 如图为四模块交叉存储器结构框图。
    - 流水线读写技术：主存被分成4个相互独立、容量相同的模块$M_0$，$M_1$，$M_2$，$M_3$，每个模块都有自己的读写控制电路、地址寄存器和数据寄存器，各自以等同的方式与CPU传送信息/。在理想情况下，如果程序段或数据块都是连续地在主存中存取，那么将大大提高主存的访问速度。



# 3.5.2 多模块交叉存储器

- 性能定量分析
  - 模块的字长等于数据数据总线宽度，模块存取一个字的存储周期为$T$，总线传输周期为 $\tau$，存储器的交叉模块数为$m$，为了实现流水线存取，要求$T=m\times\tau$。
  - $m=T/\tau$，交叉存取度。
  - 多模块交叉存储器连续读取m个字的时间
    - $t_1=T+(m-1)\times\tau$
  - 顺序方式存储器连续读取m个字的时间
    - $t_2=m\times T$
  - $t_1<t_2$，交叉存储器的带宽提高。

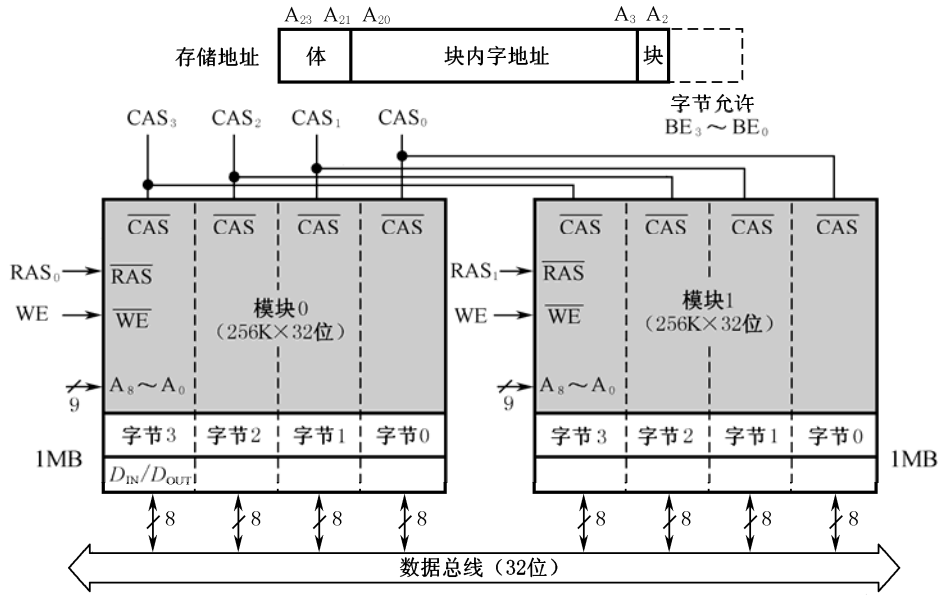


# 3.5.2 多模块交叉存储器

- 例：设存储器容量为32字，字长64位，模块数m=4，分别用顺序方式和交叉方式进行组织。存储周期T=200ns，数据总线宽度为64位，总线传送周期=50ns。若连续读出4个字，问顺序存储器和交叉存储器的带宽各是多少?
  - 解：顺序存储器和交叉存储器连续读出m=4个字的信息总量都是：q=64b×4=256b
  - 顺序存储器和交叉存储器连续读出4个字所需的时间分别是：
    - $t2=mT=4\times200ns=800ns=8\times10^{-7}s$
    - $t1=T+(m-1)=200ns+350ns=350ns=35\times10^{-7}s$
  - 顺序存储器和交叉存储器的带宽分别是：
    - $W2=q/t2=256b\div(8\times10^{-7})s=320Mb/s$
    - $W1=q/t1=256b\div(35\times10^{-7})s=730Mb/s$

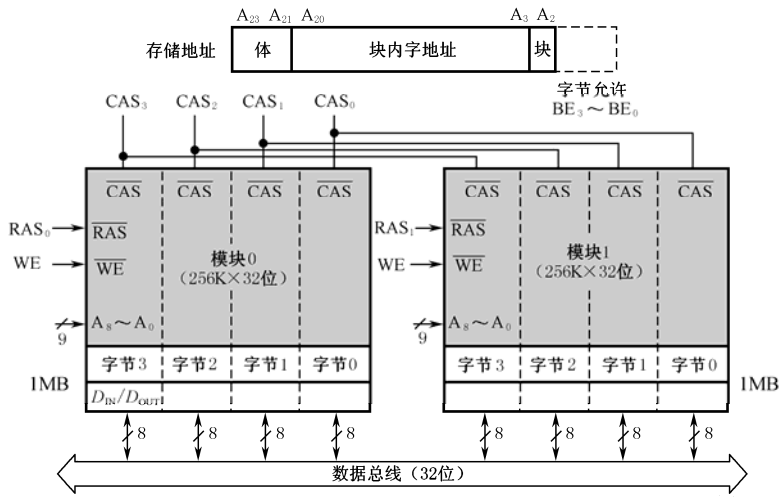
# 3.5.2 多模块交叉存储器

- **2模块交叉存储器举例**
  - 使用**256K×4**的DRAM芯片组织交叉存储器**4M×32位**
    - 每个虚线框表示2片DRAM。
    - 做位扩展：每个模块8片DRAM，模块容量＝1MB＝256K×32位。
    - 做字扩展：组织2个模块交叉存储，容量＝2MB＝512K×32位。
    - 做体*/*层扩展：总共16个模块， 2个1组 组织为2模块的交叉存储体，总共8个2模块交叉存储结构，总容量＝16MB＝2MB×8＝512K×32×8位
    - 地址总线24位。
    - 数据总线32位。



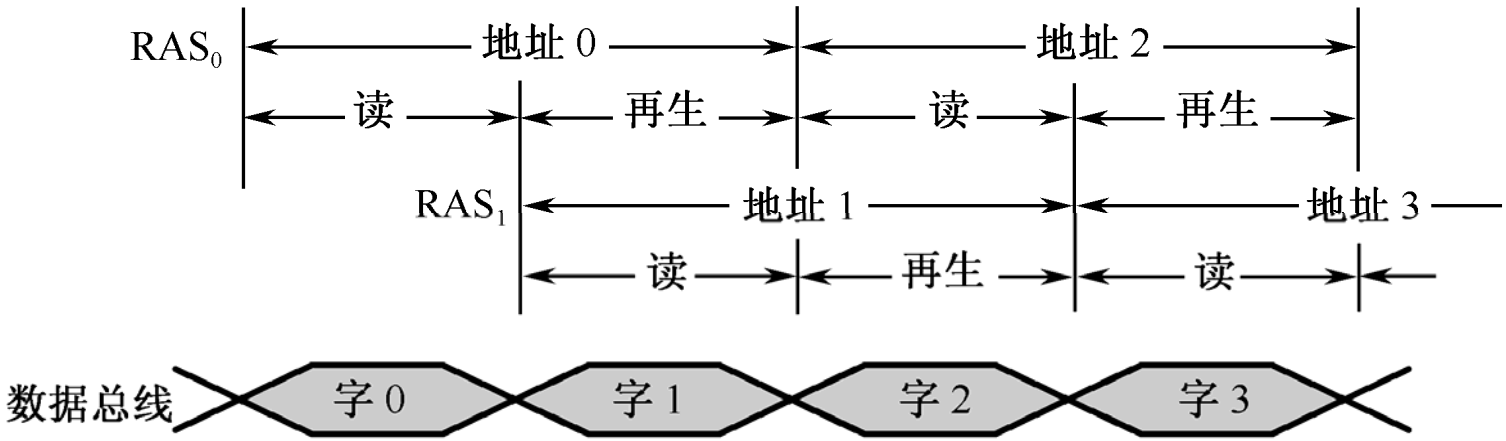
# 3.5.2 多模块交叉存储器

- **2模块交叉存储器举例**
  - 3组信号线分配：**256K×4→4M×32位**
    - **32位数据线**：32位数据线分4组，每组8位连每个模块中1个虚线框中的两片芯片。
    - **24位地址线**：
      - $A_{23}$~$A_{21}$：3位，用于8个2模块交叉存储器的选择信号。
      - $A_{20}$~$A_3$：18位，用于每个模块256K×32位 字数据的寻址。使用9位地址线分两次传送。
      - $A_2$：1位，用于2模块结构中模块的选择。
      - $A_2$~$A_0$：2位，没有使用。
    - **控制线**：
      - RAS:同1模块内的所有芯片一起连接。
      - CAS:每个模块虚线框中的2片芯片与对应字节允许信号连接。
      - 4字节允许信号：$BE_3$~$BE_0$



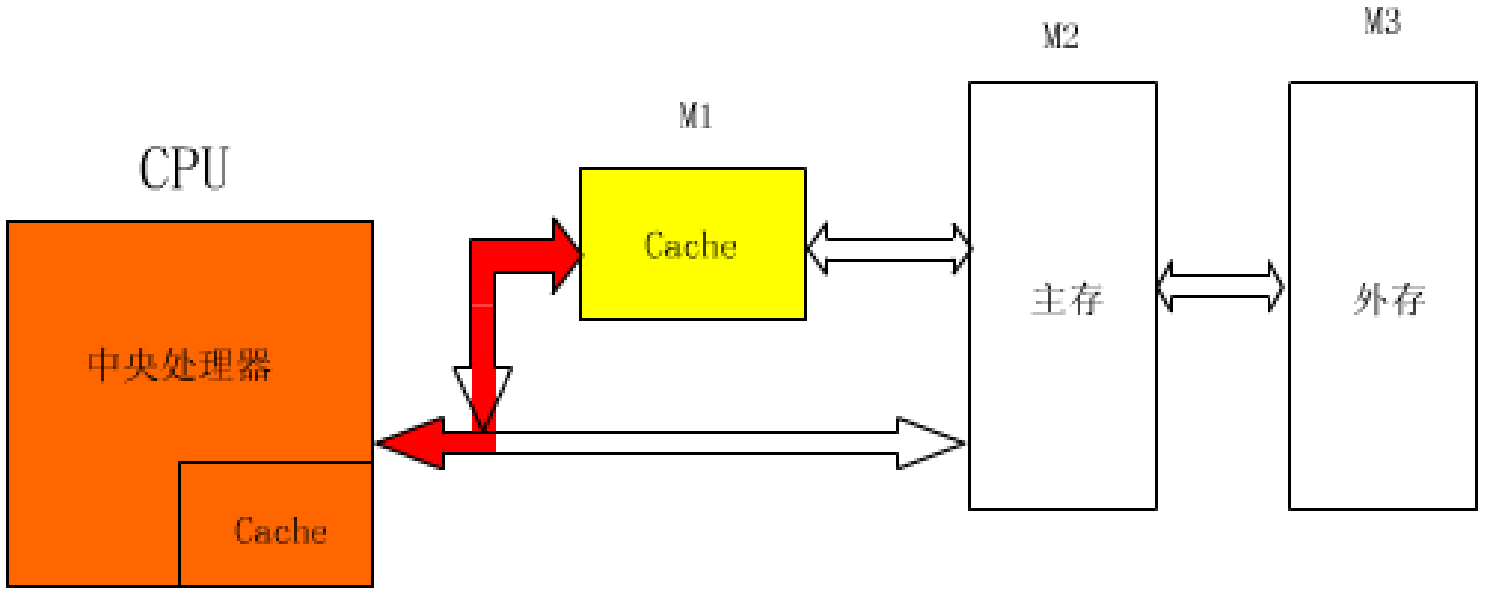
# 3.5.2 多模块交叉存储器

- **2模块交叉存储器举例**
  - 零等待存取：连续2个字的读取之间不需要等待时间。
    - 理解：在模块1中读第1个字，再生的时间/等待时间可以在模块2中读第2个字，依次类推.....在前1个字的再生时间开始读下一个字。



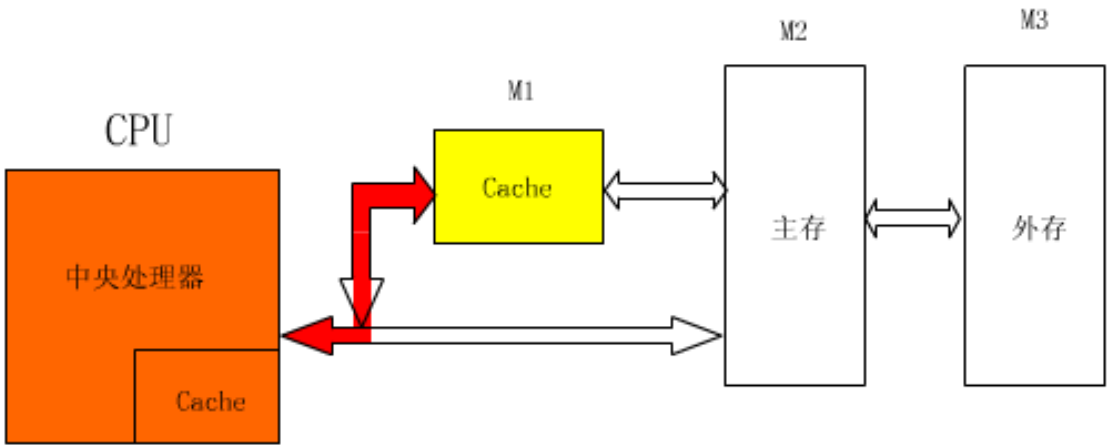
# 3.6 Cache存储器

- Cache是一种高速缓冲存储器，是为了解决cpu与主存之间速度不匹配而采用的一项技术，其基本原理基于程序运行过程中具有的空间局部性和时间局部性特征。



# 3.6.1 Cache基本原理

- **Cache的功能**
  - cache是介于CPU和主存之间的小容量存储器，由高速的SRAM 组成，存取速度比主存快，是主存的缓冲存储器。
  - 它能高速地向CPU提供指令和数据，加快程序的执行速度。
  - 它是为了解决CPU和主存之间速度不匹配而采用的一项常用技术。
  - 两级或多级Cache系统
    - 早期的一级Cache在CPU内，二级在主板上，现在的CPU内带L1 Cache和L2 Cache

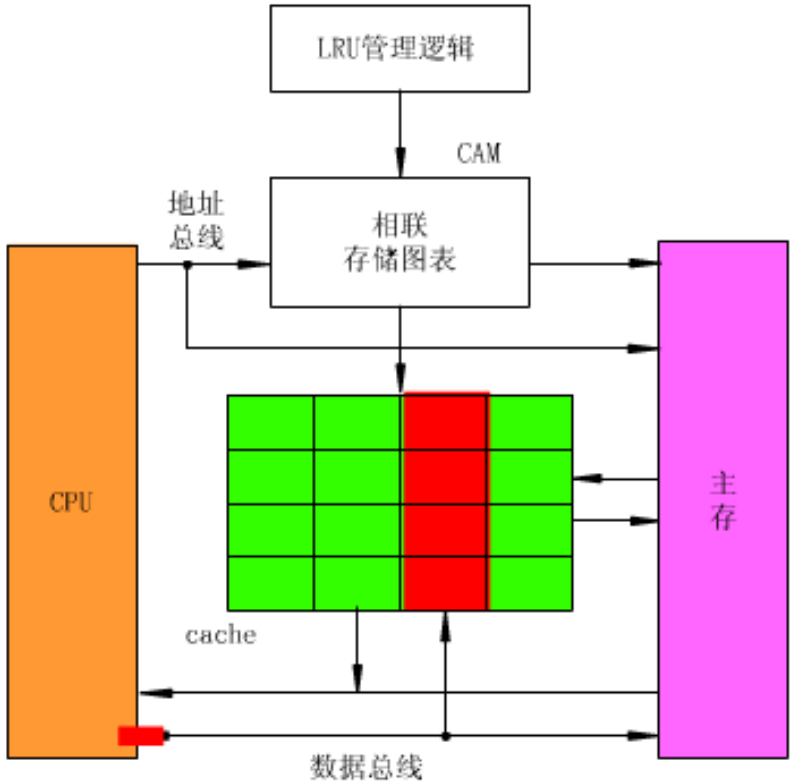


# 3.6.1 Cache基本原理

- **Cache的基本原理**
  - Cache中有SRAM和相应的控制逻辑
    - 若cache在cpu外，cache的控制逻辑与主存的控制逻辑集合在一起，主存/cache控制器。
    - 若cache在cpu内，由cpu提供它的控制逻辑。
  - CPU与cache之间的数据交换以字为单位，Cache与主存间的数据传送以块为单位，一个块(Block)由若干字组成。
  - 当cpu给出内存地址读内存字时，cache控制逻辑判断此字是否在cache中：
    - 在，cache高速传送此字到cpu中；
    - 不在，主存传送此字到cpu中，同时把包含此字的内存块传送到cache中。
  - 如果cpu要使用到的所有数据都在cache中，是最优的。

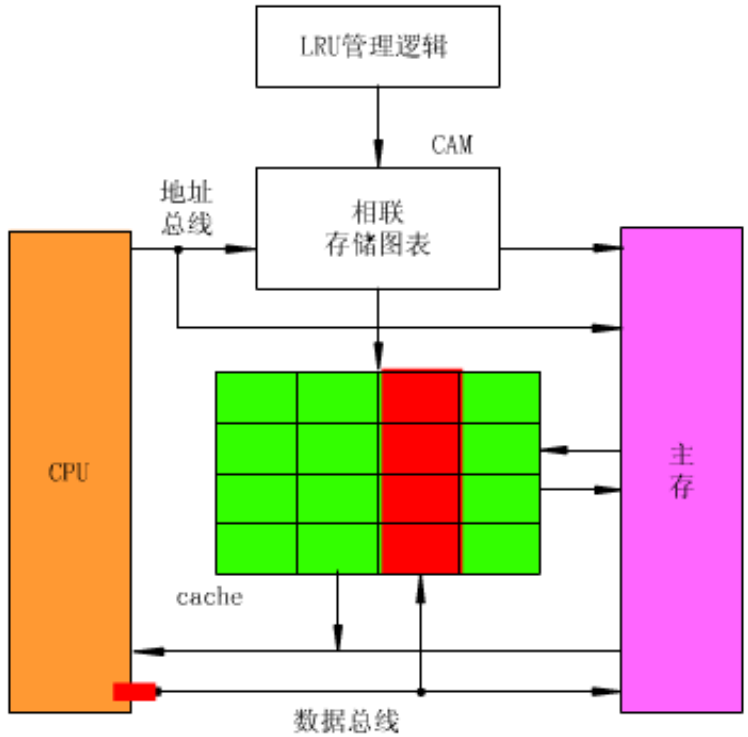
# 3.6.1 Cache基本原理

- **Cache的基本原理**
  - cache中16个字，每行4个；
  - cpu执行访问指令，把所要访问的字W的地址送入CAM(相连存储表)，判断字W是否在cache中，cache执行传送或替换操作。



# 3.6.1 Cache基本原理

- **Cache的基本结构**
  - 存储体：以字为单位，若干字构成块；
  - 相连存储器CAM：cache的块表/标记表，快速指示所要访问的信息是否在Cache中；
    - 根据不同的地址映射方式，CAM表中保存不同的块标记，检索要访问的数据是否在cache中。
  - 读写控制逻辑；
  - 地址映射变换结构：
    - 实现主存地址→cache地址；
  - 替换结构：
    - 更新cache中数据结构；



# 3.6.1 Cache基本原理

- **Cache的命中率**
  - 命中率：指CPU要访问的信息在cache中的比率；命中率→1， cpu从内存读出数据的时候→从cache读出数据的时间。
  - 命中率**h**：$h = N_c/(N_c+N_m)$
    - $N_c$ ： cache完成存取的总次数。
    - $N_m$ ： 主存完成存取的总次数。
  - **cache/主存系统的平均访问时间$t_a$**： $t_a = h \times t_c + (1-h) \times t_m$
    - $t_c$ ： cache访问时间。
    - $t_m$： 主存访问时间。
    - 失效率＝**1-** 命中率$= 1-h$
  - 期望$t_a \rightarrow t_c$ ：
    - 访问效率e： $e = t_c/t_a = 1/(h+(1-h)r) \rightarrow 1$
    - $r = t_m/t_c$： 内存与cache的速度比

# 3.6.1 Cache基本原理

- 例：CPU执行一段程序时，cache完成存取的次数为1900次，主存完成存取的次数为100次，已知cache存取周期为50ns，主存存取周期为250ns，求cache/主存系统的效率和平均访问时间。
- 解：
  - 命中率
    - $h=N_c/(N_c+N_m)=1900/(1900+100)=0.95$
  - 主存与**Cache**的速度倍率
    - $r=t_m/t_c=250ns/50ns=5$
  - 访问效率
    - $e=1/(r+(1-r)h)=1/(5+(1-5)\times0.95)=83.3\%$
  - 平均访问时间
    - $t_a=t_c/e=50ns/0.833=60ns$

# 3.6.2 主存与Cache的地址映射

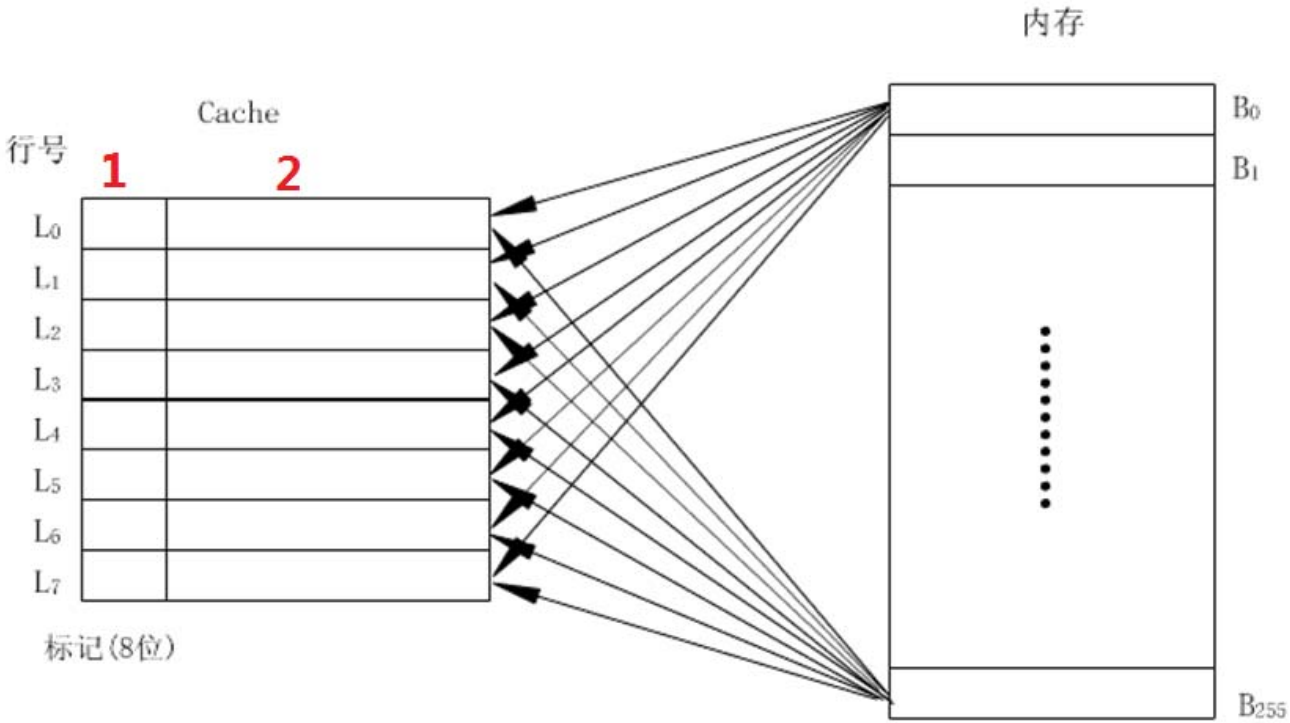
- Cache的数据块称为行（线Line，槽Slot），主存的数据块称为块（Block），行与块是等长的，均包含相同的 字 单元。
- 地址映射：确定主存块与Cache行之间的对应关系，确定一个主存块应该存放到哪个Cache行中，即将CPU访问提供的主存地址按某种映射函数关系变换成**Cache地址**的过程；
- 地址映射的方式
  - 全相联映射：将一个主存块存储到任意一个Cache行
  - 直接映射：将一个主存块存储到唯一的一个Cache行
  - 组相联映射：将一个主存块存储到唯一的一个Cache组中任意一个行
- 选择哪种映射方式，要考虑：
  - 硬件是否容易实现
  - 地址变换的速度是否快
  - 主存空间的利用率是否高
  - 主存装入一块时，发生冲突的概率

# 3.6.2 主存与Cache的地址映射

- 全相联映射方式
  - Cache的数据块是行，用$L_i$表示，i=0,1,2,.....,m-1，共m=$2^r$行。
  - 主存的数据块是块，用$B_j$表示，j=0,1,2,.....,n-1，共n=$2^s$块。
  - 每行/每块均有连续的k=$2^w$个字数据，字是cpu访问存储器的最小存取单位。**r＜s**
  - **Cache地址**：r+w位，r：行号，w：行内地址；
  - 主存地址：s+w位，s：块号，w：块内地址；
  - 全相联映射：将主存中的一个块的地址(块号s)和内容(数据)一起保存在cache中，**块与行对应关系是随机的**。
  - 将主存中第j块写入cache中第i行：
    - 在CAM中写入标记：块号j就是标记，将块号j写入cache中CAM中第i行的位置。
    - 写入数据：第j块中的数据写入cache中第i行存储体。
  - 特点：
    - 内存块号s位，**s位块号→标记**。
    - 主存中的块可以对应cache中的任意行。是一种多对1的关系。
    - CAM表占用较大的cache空间(大小应为$2^r$×s位)，电路设计难。

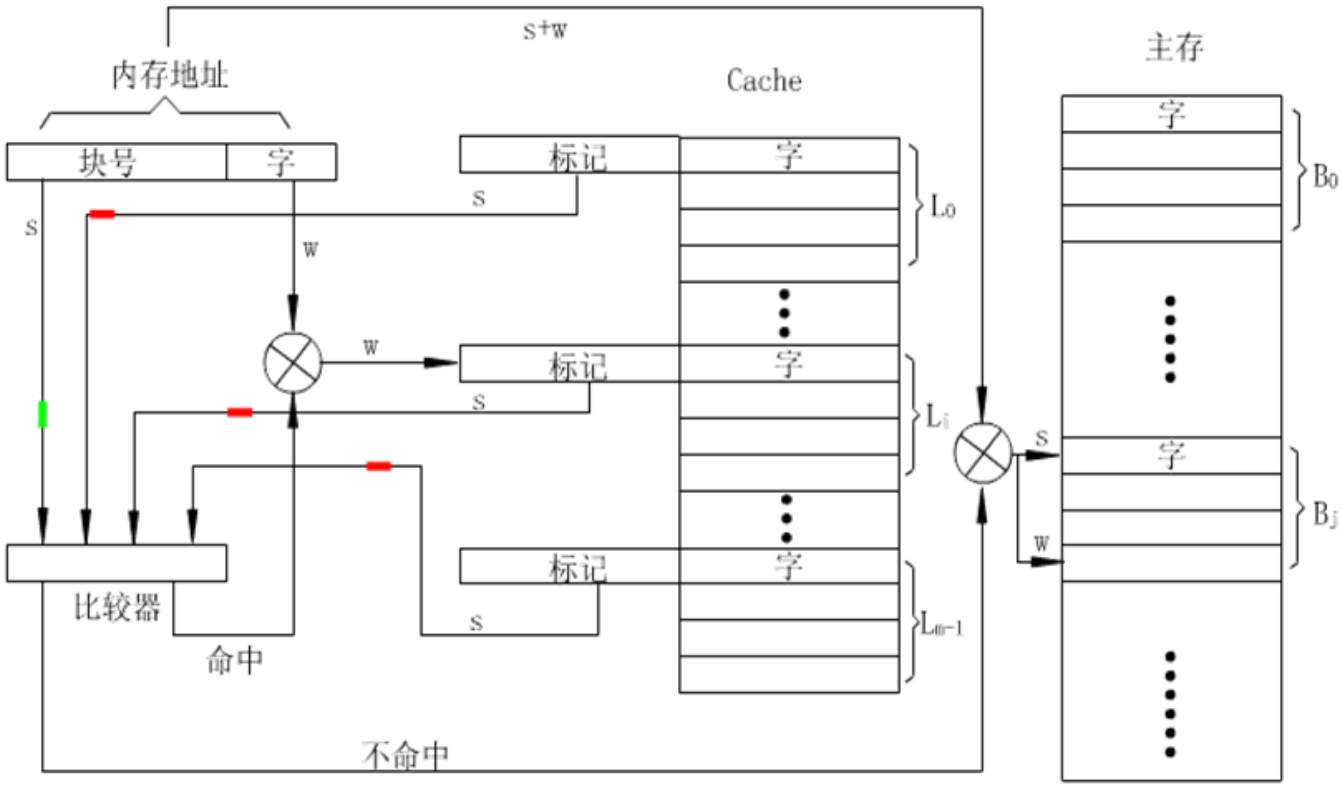
# 3.6.2 主存与Cache的地址映射

- 全相联映射方式
  - Cache：8行。主存：256块，块号8位表示。256块可以映射到8行中的任意1行。
  - 被映射的块其块号写入cache中第1部分CAM/标记表中。数据写入cache中第2部分/cache行。
  - CAM表中标记就是块号，标记8位。



# 3.6.2 主存与Cache的地址映射

- 全相联映射方式
  - 检索：
    - 访问指令提供的内存地址(块号+块内偏移 / 字)
    - 块号与CAM中所有标记做比较；
    - 块号命中，从cache命中行中读数据；否则从内存中读数据。



# 3.6.2 主存与Cache的地址映射

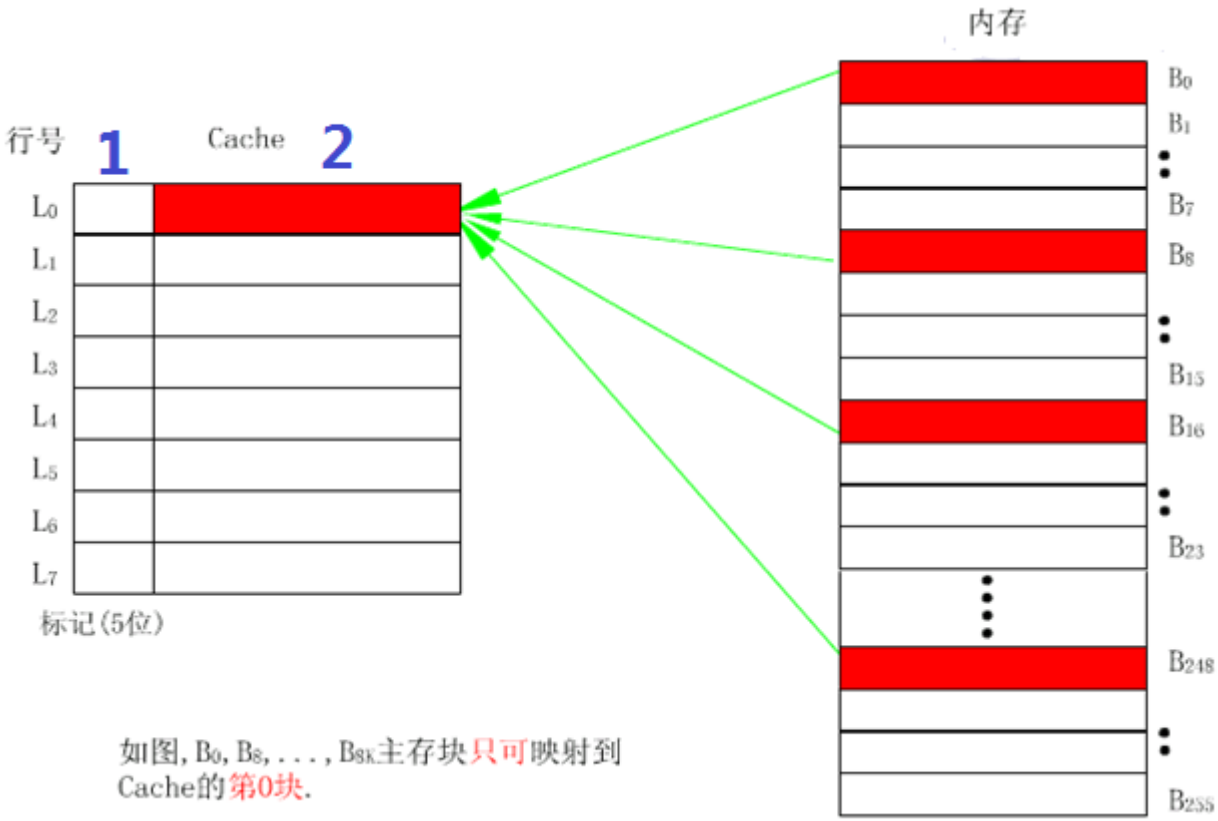
- 全相联映射方式
  - 映射公式：
    - 主存地址长度＝(s＋w)位
    - 寻址单元数＝$2^w$个字或字节
    - 块大小＝行大小＝$2^w$个字或字节
    - 主存的块数＝$2^s$
    - 标记大小＝s位
    - cache的行数＝不由地址格式确定

# 3.6.2 主存与Cache的地址映射

- 直接映射方式
  - Cache中共m=$2^r$行，内存共n=$2^s$块。每行/每块有相同个数字数据。
  - 直接映射：主存中的块映射到cache中的特定行，是一种多对1的映射。
  - Cache的行号i和主存的块号j的关系：
    - ***i=j mod m*** m：cache的总行数
    - 主存中的第0，m，2m，.....映射到cache中的第0行；
    - 主存中的第1，m+1，2m+1，.....映射到cache中的第1行；
  - 将主存中第j块写入cache中第i行：
    - 在CAM中写入标记：块号j的高s-r位作为标记，写入到cache中相联存储表CAM中第i行的位置。
    - 写入数据：第j块中的数据写入cache中第i行。
  - 说明：
    - 内存块号s位，分为高**s-r位→标记**和低**r位→cache行号**。

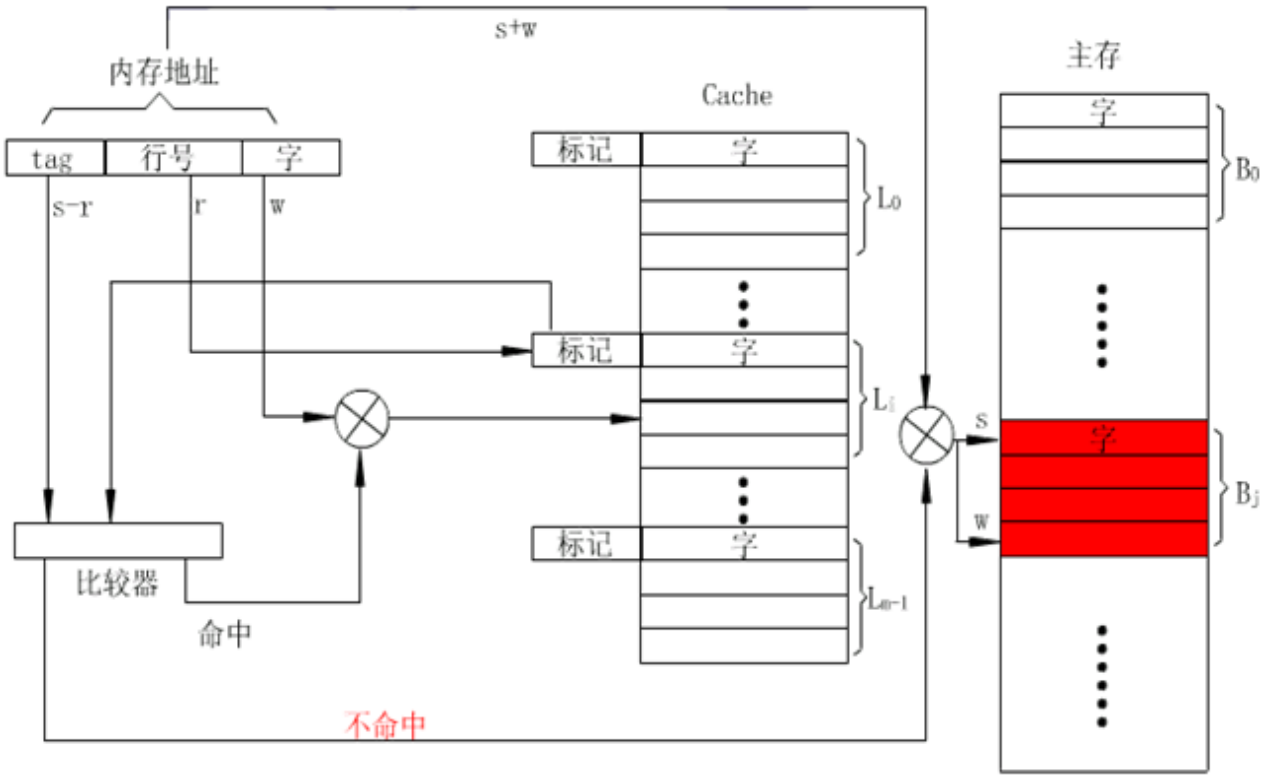
# 3.6.2 主存与Cache的地址映射

- 直接映射方式
  - Cache：8行，行号3位。主存：256块，块号8位表示。标记5位；
  - 以8为模进行映射：i=j mod 8，第j块只能映射到第i行。
  - 被映射的块其块号高5位作为标记写入到cache第1部分CAM表第i行。数据写入cache中第i行对应的第2部分/cache行。



# 3.6.2 主存与Cache的地址映射

- 直接映射方式
  - 检索：
    - 访问指令提供的内存地址(标记+行号+块内偏移/字)
    - 内存地址标记与行号指定标记做比较；
    - 命中，从cache命中行中读数据；否则从内存中读数据。



# 3.6.2 主存与Cache的地址映射

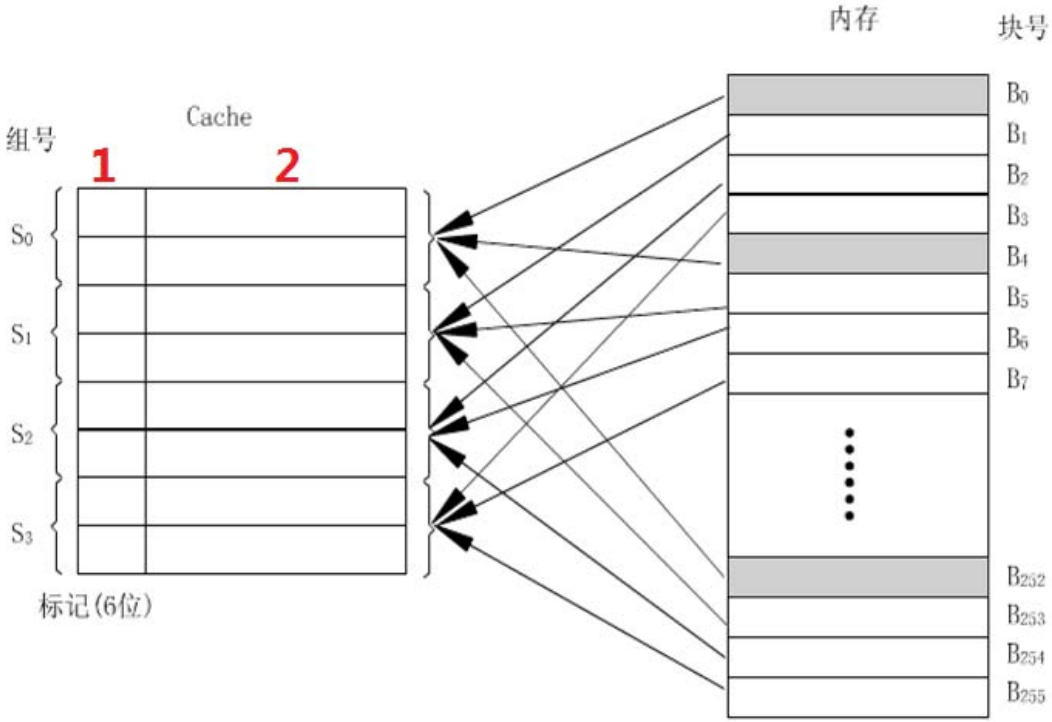
- 直接映射方式
  - 映射公式：
    - 主存地址长度＝(s+w)位
    - 寻址单元数＝$2^{s+w}$个字或字节
    - 块大小＝行大小＝$2^{w}$个字或字节
    - 主存的块数＝$2^{s}$
    - cache的行数＝m＝$2^{r}$
    - 标记大小＝(s-r)位

# 3.6.2 主存与Cache的地址映射

- 组相联映射方式
  - Cache分u=$2^d$组，每组v行，共m=u×v行，内存共n=$2^s$块。每行/每块有 相同个数 字数据。
  - 组相联映射：主存中的块映射到cache中的特定组，在组内的位置是随机的，组间直接/组内全相联。
  - Cache的组号q和主存的块号j的关系：
    - ***q=j mod u*** u：cache的组数
    - 主存中的第0，u，2u，……映射到cache中的第0组；
    - 主存中的第1，u+1，2u+1，……映射到cache中的第1组；
  - 将主存中第j块写入cache中第q组第i行：
    - 在CAM中写入标记：块号j的高s-d位作为标记，写入到cache中相联存储表CAM中第q组第i行的位置。
    - 写入数据：第j块中的数据写入cache中第q组第i行。
  - 说明：
    - 内存块号s位，分为高**s-d**位→标记和低d位→cache组号。

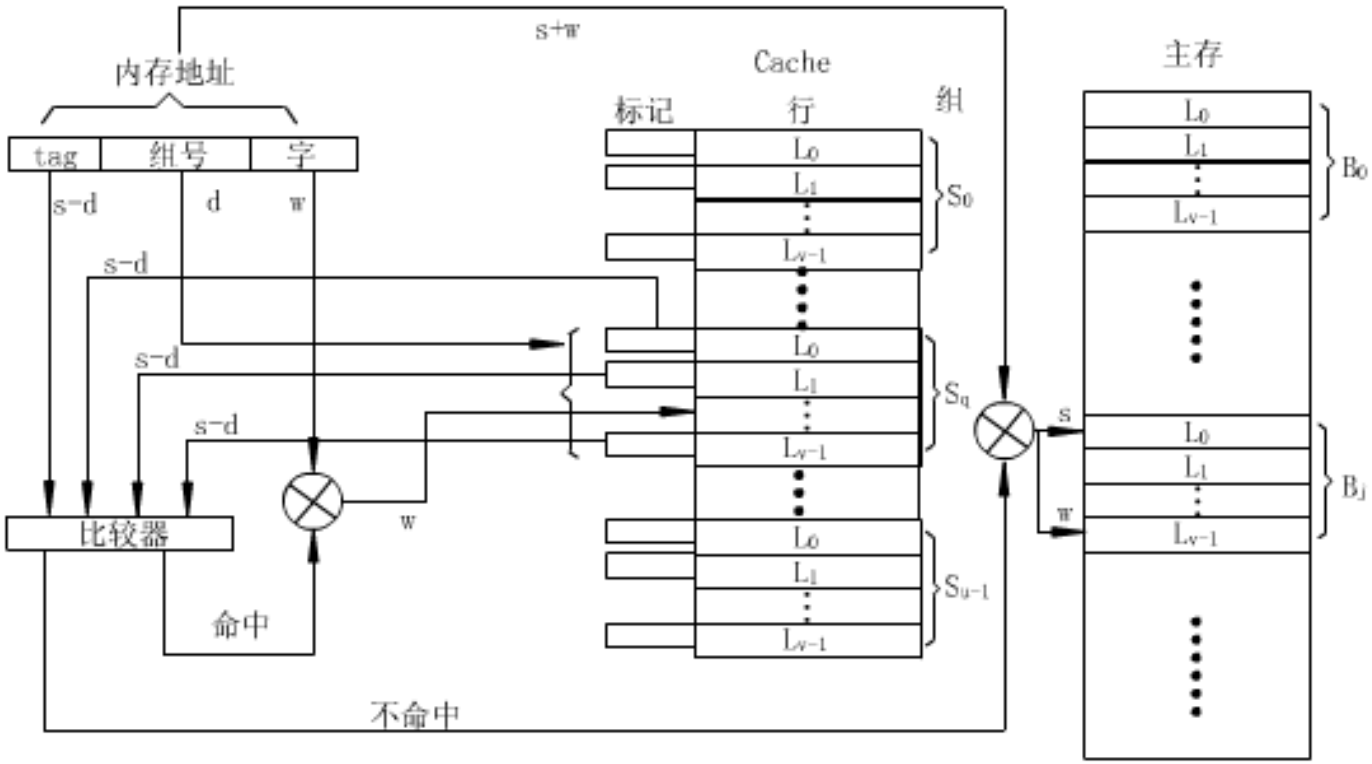
# 3.6.2 主存与Cache的地址映射

- 组相联映射方式
  - Cache：4组，每组2行，共8行，组号2位。主存：256块，块号8位表示。标记6位；
  - 首先以4为模进行组映射：q=j mod 4，再次进行全相联映射到第q组的第i行。
  - 被映射的块其块号高6位作为标记写入到cache第1部分CAM表第q组第i行。数据写入cache中第q组第i行对应的第2部分/cache行。



# 3.6.2 主存与Cache的地址映射

- 组相联映射方式
  - 检索：
    - 访问指令提供的内存地址(标记+组号+块内偏移/字)
    - 内存地址标记与组号指定的所有标记做比较；
    - 命中，从cache对应组命中行中读数据；否则从内存中读数据。



# 3.6.2 主存与Cache的地址映射

- 组相联映射方式
  - 映射公式：
    - 主存地址长度＝(s+w)位
    - 寻址单元数＝$2^{s+w}$个字或字节
    - 块大小＝行大小＝$2^{w}$个字或字节
    - 主存的块数＝$2^{s}$
    - 每组的行数＝k
    - 每组的v＝$2^{d}$
    - cache的行数＝u×v
    - 标记大小＝(s-d)位

# 3.6.2 主存与Cache的地址映射

- 3中映射方式比较
  - 全相联映射：
    - 内存地址=块号+偏移地址/字地址；
    - 标记=块号
    - 多对多的映射
  - 直接映射：
    - 内存地址=tag/标记+行号+偏移地址/字地址；
    - 标记=块号高s-r位，s块号位数，r行号位数；
    - 多对1的映射
  - 组相联映射：
    - 内存地址= tag/标记+组号+偏移地址/字地址；
    - 标记=块号高s-d位，s块号位数，d组号位数；
    - 多对多的映射
    - v=1，是直接映射
    - u=1，是全相联映射

# 3.6.2 主存与Cache的地址映射

- 例1：设主存容量1MB，cache容量16KB，块的大小为512B，采用全相联映射方式。

① 写出cache的地址格式。

② 写出主存的地址格式。

③ 块表/标记表/ACM表的容量多大？

④ 主存地址为CDE8FH的单元，在cache中的什么位置？

⑤ 画出地址映射及变换示意图。

- 解：
  - cache的地址格式： **14位地址**


    - cache的容量16KB→**14位地址**
    - 块（行）的大小为512B →块/行内地址 **9位**
    - 行号/行地址为14－9＝**5位**　　$2^5=32$行

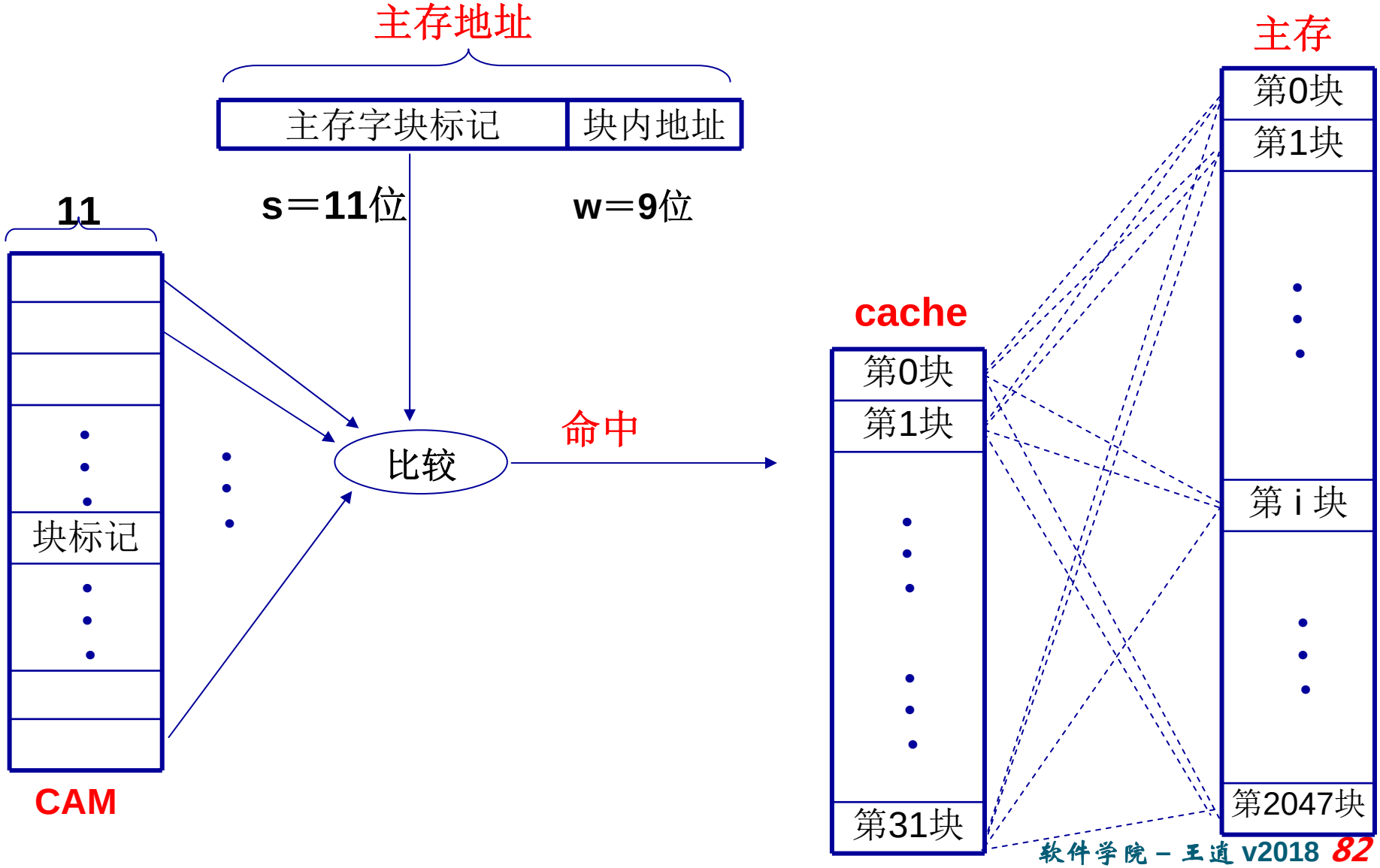
# 3.6.2 主存与Cache的地址映射

- 解：
  - 主存的地址格式：


    - 主存的容量1MB→**20位地址**
    - 块（行）的大小为512B →块/行内地址 **9位**
    - 块号/块地址为20－9＝**11位**　$2^{11}=2K=2048$块
    - **tag＝块号＝11位**
  - 块表/标记表/ACM表的容量：
    - 行数×tag/块号位数＝$2^5$×**11位**
  - 主存地址为CDE8FH的单元，在cache中的位置：
    - 可以映射到cache中的任意行
    - **CDE8FH= 1100 1101 1110 1000 1111** B
    - 在映射行中的位置是/行内地址/块内地址/偏移量：**010001111**

# 3.6.2 主存与Cache的地址映射

- 解：地址映射及变换示意图：



# 3.6.2 主存与Cache的地址映射

- 例2：设主存容量1MB，cache容量16KB，块的大小为512B，采用直接映射方式。

① 写出cache的地址格式。

② 写出主存的地址格式。

③ 块表/标记表/ACM表的容量多大？

④ 主存地址为CDE8FH的单元，在cache中的什么位置？

⑤ 画出地址映射及变换示意图。

- 解：
  - cache的地址格式： **14位地址**

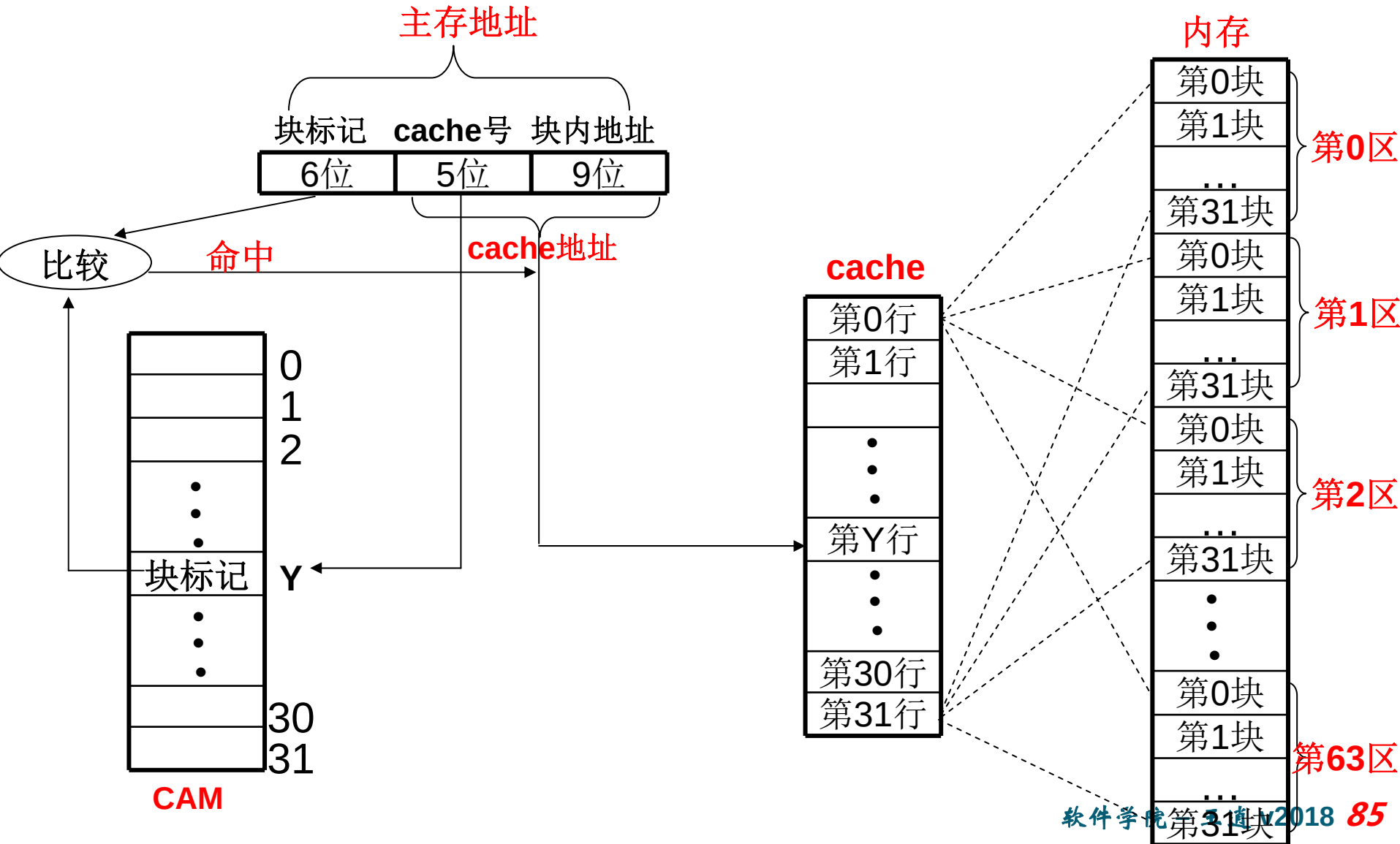

    - cache的容量16KB→**14位地址**
    - 块（行）的大小为512B →块/行内地址 **9位**
    - 行号/行地址为14－9＝**5位**　　$2^5=32$行

# 3.6.2 主存与Cache的地址映射

- 解：
  - 主存的地址格式：
    - 主存的容量1MB→**20位地址**
    - 块（行）的大小为512B →块/行内地址 **9位**
    - 块号/块地址为20－9＝**11位**　$2^{11}$=2K=2048块
    - **块号=tag+行号，tag=6位，行号=5位**
  - 块表/标记表/ACM表的容量：
    - 行数×块号位数=$2^5$×**6位**
  - 主存地址为CDE8FH的单元，在cache中的位置：
    - **CDE8FH= 1100 1101 1110 1000 1111** B
    - Tag= **1100 11**；行号=**01 111**，即cache中第15行；
    - 在映射行中的位置是/行内地址/块内地址/偏移量：**010001111**

# 3.6.2 主存与Cache的地址映射

- 解：地址映射及变换示意图：



# 3.6.2 主存与Cache的地址映射

- 例3：某系统的存储器为2MB，每字块为8个字，每字32位，若cache为16KB，采用字节编址方式。

① 采用直接映射，主存地址格式是什么？

② 采用全相联映射，主存地址格式是什么？

③ 采用16路组相联映射，主存地址格式是什么？

- 解：
  - **2MB**的存储器：**21位地址**
    - 块地址/块号：**16位**　　$2^{16}=64K$块
    - 块/行内地址→8个字：**3位**
    - 字节地址→每字32位/4个字节：**2位**
  - **16KB**的cache：**14位地址**
    - 行地址/行号：**9位**　　$2^{9}=512$行
    - 块/行内地址→8个字：**3位**
    - 字节地址→每字32位/4个字节：**2位**

# 3.6.2 主存与Cache的地址映射

- 例3：某系统的存储器为2MB，每字块为8个字，每字32位，若cache为16KB，采用字节编址方式。

① 采用直接映射，主存地址格式是什么？

② 采用全相联映射，主存地址格式是什么？

③ 采用16路组相联映射，主存地址格式是什么？

- 解：
  - 直接映射主存地址格式：

| 7位标记 | 9位行地址 | 3位块内地址 | 2位字节地址 |
|---|---|---|---|

  - 全相联映射主存地址格式：

| 16位标记 | 3位块内地址 | 2位字节地址 |
|---|---|---|

  - 组相联映射主存地址格式：

| 12位标记 | 4位组地址 | 3位块内地址 | 2位字节地址 |
|---|---|---|---|

# 3.6.2 主存与Cache的地址映射

- 例4：P96 一个组相联cache由64个行组成，每组4行。主存包含4K个块，每块128字。请表示内存地址的格式。
- 解：
  - 块大小＝行大小＝$2^w$个字＝128＝$2^7$　　∴ w＝7
  - 每组的行数k＝4
  - cache的行数＝kv＝K×$2^d$＝4×$2^d$＝64　　∴ d＝4
  - 组数v＝$2^d$＝$2^4$＝16
  - 主存的块数$2^s$＝4K＝$2^2$×$2^{10}$＝$2^2$　　∴ s＝12
  - 标记大小(s-d)位＝12-4＝8位
  - 主存地址长度(s+w)位＝12+7＝19位
  - 主存寻址单元数$2^{s+w}$＝$2^{19}$
  - 故 k＝4各组相联的内存地址格式如下所示：
  - **组相联映射主存地址格式：**

| s-d=8位标记 | d=4位组号 | 字号=7位 |
|---|---|---|

# 3.6.3 替换策略

- 替换问题：
  - cache工作原理要求它尽量保存最新数据，当新主存块要进入cache行时，如果cache行已经被占用，则要产生替换问题。
  - 对直接映射的cache来说，只要把此特定位置上的原主存块换出cache即可。
  - 对全相联和组相联cache来说， 就要从允许存放新主存块的若干特定行中选取一行换出。即需要选择替换策略（算法）
- 常用替换算法有：
  - 最不经常使用(**LFU**：Least frequently used )算法
    - 替换使用次数最少的块
  - 近期最少使用(**LRU**：Least Recently Used )算法
    - 本指替换近期最少使用的块，实际实现的是替换最久没有被使用的块
  - 随机替换(random)
    - 随意选择被替换的块，不依赖以前的使用情况

# 3.6.3 替换策略

- 最不经常使用/LFU算法：
- 替换原则
  - 将一段时间内被访问次数最少的那行数据替换出去；
- 使用方法
  - 每行设置一个计数器，从0开始计数；
  - 每访问一次， 被访问行的计数器增1；
  - 当需要替换时，将计数值最小的行换出，同时将这些行的计数器都清零。
- 特点
  - 这种算法将计数周期限定在对这些特定行两次替换之间的间隔时间内，不能严格反映近期访问情况。

# 3.6.3 替换策略

- 近期最少使用 / LRU算法：
- 替换原则
  - 将近期内长久未被访问过的行替换出去。
- 使用方法
  - 每行也设置一个计数器；
  - 每访问一次，被访问行的计数器清零，其它各行计数值1；
  - 当需要替换时，将计数值最大的行换出。
- 特点
  - 这种算法保护了刚拷贝到cache中的新数据行，使Cache的使用率较高。

# 3.6.3 替换策略

- 随机替换算法：
- 替换原则
  - 从特定的行位置中随机地选取一行换出。
- 特点
  - 在硬件上容易实现，且速度也比前两种策略快。但降低了命中率和cache工作效率。

# 3.6.4 cache的写操作策略

- 写问题：
  - cache中的数据是主存部分数据的拷贝，要求二者内容保持一致。
  - 当发生cpu写cache操作时，如何保持二者数据一致？
    - 使用cache写操作策略。
- 3中cache写操作策略：
  - 写回法(**Write Back**) /回写法
    - 换出时，对行的修改位进行判断，决定是写回还是舍掉。
  - 全写法(**Write Through**) /直写法/写透法
    - 写命中时，Cache与内存一起写
  - 写一次法(**Write Once**) /
    - 与写回法一致，但是第一次Cache命中时采用全写法。

# 3.6.4 cache的写操作策略

- 写回法：
- 写入策略
  - 只修改cache的内容，而不立即写入主存；
  - 只有当此行被换出时才写回主存。
- 优点
  - 减少了访问主存的次数
- 缺点
  - 存在Cache与主存不一致性的隐患。
- 实现该方法时，cache行必须配置一个修改位，以反映此行是否被CPU修改过。

# 3.6.4 cache的写操作策略

- 全写法：
- 写入策略
  - cache与主存同时发生写修改，因而较好地维护了cache与主存的内容的一致性。
- cache中每行无需设置一个修改位以及相应的判断逻辑。
- 缺点是降低了cache的功效。

# 3.6.4 cache的写操作策略

- 写一次法：
- 写入策略
  - 基于写回法，并结合全写法的写策略；
  - 写命中与写未命中的处理方法与写回法基本相同，只是第一次写命中时要同时写入主存。
- 第一次写命中时，启动一个主存的写周期，其目的是使其它Cache可以及时更新或废止该块内容，这便于维护系统全部cache的一致性。

# Cache中的内容总结

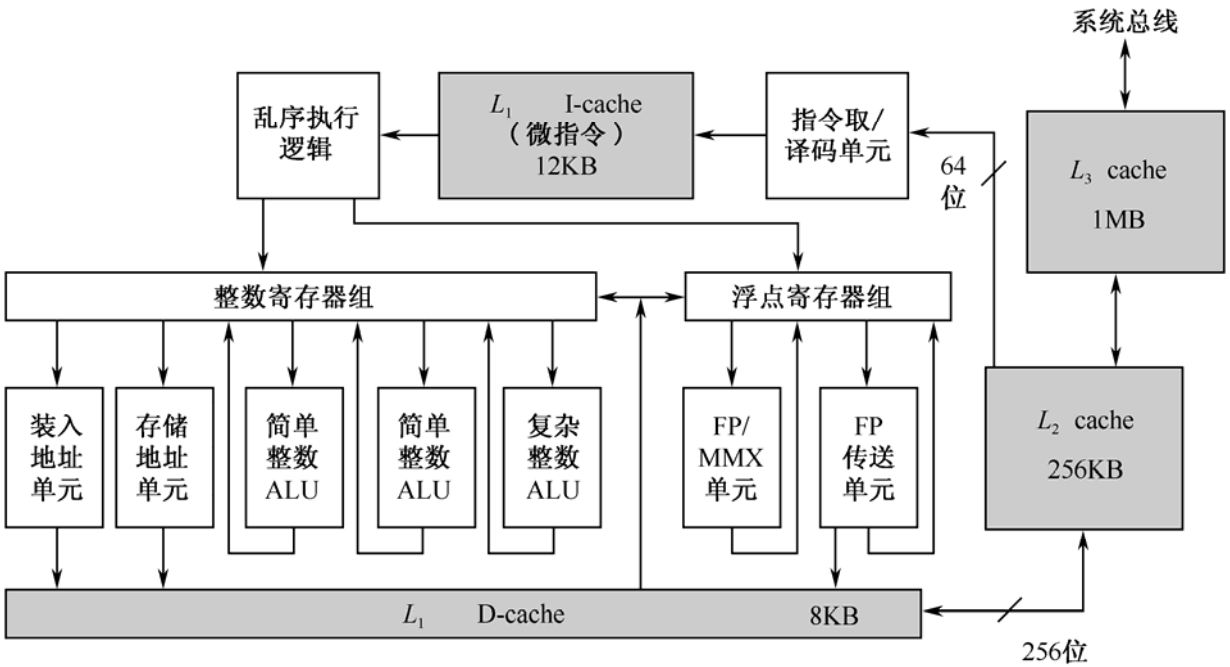
- **cache的内容包括：**
  1. 标记：与地址映射方式有关；
  2. 有效位：是否有有效数据；
  3. 数据：以块为单位；
  4. cache一致性维护控制位（是否写过）；
  5. 替换算法的控制位 （计数器）。

# 3.6.5 Pentium4的cache组织

- **Intel微处理器的cache组织**：
  - **80386**：没有片内cache。
  - **80486**：8KB的片内cache，每行16B的4路组相联结构。
  - **Pentium**：两片片内$L_1$ cache，1片D_cache，1片I_cache。
  - **Pentium2**：增加$L_2$cache，256KB，每行128B，8路组相连结构。
  - **Pentium3**：增加外部$L_3$cache。
  - **Pentium4**：增加片内$L_3$cache。

# 3.6.5 Pentium4的cache组织

- **Petium4的cache布局及主要部件：**
  - 取指/译码单元：顺序从L2cache中取程序指令，将它们译成一系列的微指令，并存入L1指令cache中。
  - 乱序执行逻辑：依据数据相关性和资源可用性，调度微指令的执行，因而微指令可按不同于所取机器指令流的顺序被调度执行。
  - 执行单元：它执行微指令，从L1数据cache中取所需数据，并在寄存器组中暂存运算结果。
  - 存储器子系统：这部分包括L2cache、L3cache和系统总线。当L1、L2cache未命中时，使用系统总线访问主存。系统总线还用于访问I/O资源。



# 3.6.6 使用多级cache减少缺失损失

- **Intel微处理器的cache组织：**
  - **L1，L2，L3**：3级缓存系统
  - 访问L1没有命中，有缺失损失。
  - 访问L2命中，缺失损失定义为访问L2的时间。
  - 访问L2没有命中，访问L3命中，访问L3的时间就是缺失损失。
  - 访问L3没有命中，访问内存，访问内存的时间就是缺失损失。
  - 例10中的 **CPI**（Clock cycle Per Instruction）表示执行某个程序的指令平均时钟周期数则：它等于 一个程序的CPU时钟周期数/该程序的指令条数

# 3.7 虚拟存储器

- 理解：
  - 多用户多任务系统：计算机同时执行多个应用程序，共享计算机的物理内存。
  - 问题：
    - 物理内存的容量较小，无法容纳执行的多个应用程序。
    - 多个应用程序共享内存，某个执行的应用程序分配到哪块物理内存？能一定分配到写程序时规定的地址对应的物理内存吗？
  - 解决思路：使用虚拟内存的概念解决上面的问题
    - 只为执行程序当前正在使用的指令和数据分配物理内存；其余部分驻留在辅存中。
    - 区分编程序时使用到的内存地址与程序执行时分配到的物理内存地址，二者之间只有在程序执行时才进行转换。

# 3.7.1 虚拟存储器的基本概念

- 实地址与虚地址：
  - 虚地址与虚拟存储空间：编制程序时使用的地址称为虚地址或逻辑地址，其对应的存储空间是虚拟存储空间或逻辑空间；
    - 虚拟存储器只是一个容量非常大的存储器的逻辑模型，不是任何实际的物理存储器。
  - 实地址与物理存储空间：计算机物理内存的访问地址则称为实地地或物理地址，其对应的存储空间称为物理存储空间或主存空间。
  - 程序进行虚地址到实地址转换的过程称为程序的再定位。地址映射的问题。**虚拟地址/逻辑地址→物理地址**
    - 物理地址由CPU地址引脚送出，用于访问主存的地址。
    - 虚拟地址由编译程序生成的，是程序的逻辑地址，其地址空间的大小受到辅助存储器容量的限制。
    - 每个程序都自己的虚拟空间，所有执行程序共享物理内存空间。

# 3.7.1 虚拟存储器的基本概念

- 虚拟存储器的访问：
  - 程序运行时，由地址变换机构将程序中由虚拟地址定位的正在使用指令和数据调入程序分配到的物理内存中。
  - 每次访问获取指令和数据时，首先判断虚拟地址定位的指令和数据是否在物理内存中？
    - 在，将执行指令中的虚拟地址转换为物理地址，使用物理地址访问物理内存，获取数据；
    - 不在，按照某种算法将虚拟空间中的程序/数据调度进物理内存，再将虚拟地址转换为物理地址访问物理主存，获取指令和数据。
  - 三级存储体系：Cache，主存/内存条，辅存/硬盘
    - cache和主存构成了系统的内存；
    - 主存和辅存依靠辅助软硬件的支持构成了虚拟存储器；

# 3.7.1 虚拟存储器的基本概念

- 区分cache和虚存：
  - 出发点相同：二者都是为了提高存储系统的性能价格比而构造的分层存储体系，都力图使存储系统的性能接近高速存储器，而价格和容量接近低速存储器。
  - 原理相同：都是利用了程序运行时的局部性原理把最近常用的信息块从相对慢速而大容量的存储器调入相对高速而小容量的存储器。
  - 侧重点不同：cache主要解决主存与CPU的速度差异问题；而就性能价格比的提高而言，虚存主要是解决存储容量问题，另外还包括存储管理、主存分配和存储保护等方面。
  - 数据通路不同：CPU与cache和主存之间均有直接访问通路，cache不命中时可直接访问主存；而虚存所依赖的辅存与CPU之间不存在直接的数据通路，当主存不命中时只能通过调页解决，CPU最终还是要访问主存。
  - 透明性不同：cache的管理完全由硬件完成，对系统程序员和应用程序员均透明；而虚存管理由软件（操作系统）和硬件共同完成，由于软件的介入，虚存对实现存储管理的系统程序员不透明，而只对应用程序员透明（段式和段页式管理对应用程序员“半透明”）。
  - 未命中时的损失不同：由于主存的存取时间是cache的存取时间的5～10倍，而主存的存取速度通常比辅存的存取速度快上千倍，故主存未命中时系统的性能损失要远大于cache未命中时的损失。

# 3.7.1 虚拟存储器的基本概念

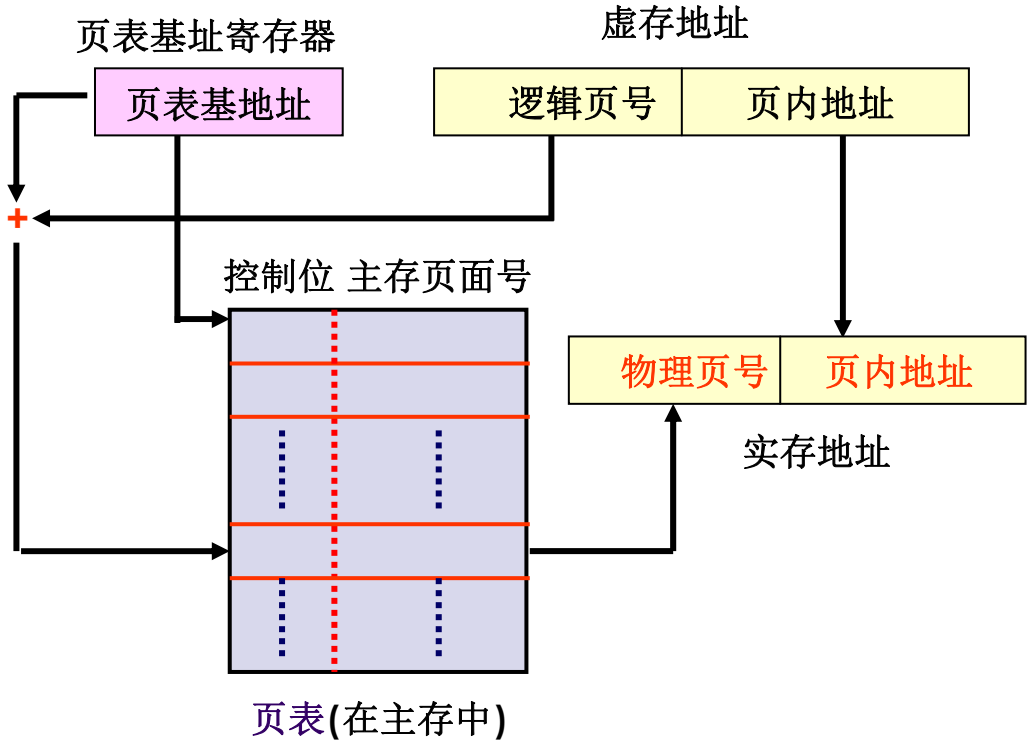
- 虚存机制要解决的关键问题
  - 调度问题：决定哪些程序和数据应被调入主存。
  - 地址映射问题：在访问主存时把虚地址变为主存物理地址（这一过程称为内地址变换）；在访问辅存时把虚地址变成辅存的物理地址（这一过程称为外地址变换），以便换页。此外还要解决主存分配、存储保护与程序再定位等问题。
  - 主存-外存层次的基本信息传送单位
    - 段：按程序逻辑划分为可变长的块，称为段
    - 页：机械地划分为大小相同的块，称为页面
    - 段页：程序按模块分段，段内分页
  - 替换问题：决定哪些程序和数据应被调出主存。
  - 更新问题：确保主存与辅存的一致性。

# 3.7.1 虚拟存储器的基本概念

- 虚拟存储器的管理
  - 段式管理：把主存按段分配的存储管理方式
    - 优点：段的界线分明，段易于编译、管理、修改和保护，便于多道程序共享
    - 缺点：段的长度各不相同，主存空间分配麻烦
  - 页式管理：以定长页面进行存储管理的方式
    - 优点：页的起点和终点地址固定,方便造页表,新页调入主存也很容易掌握，比段式空间浪费小
    - 缺点：处理、保护和共享都不及段式来得方便
  - 段页式管理：分段和分页相结合的存储管理方式
    - 优点：综合段式和页式管理方式的特点
    - 缺点：需要多次查表过程

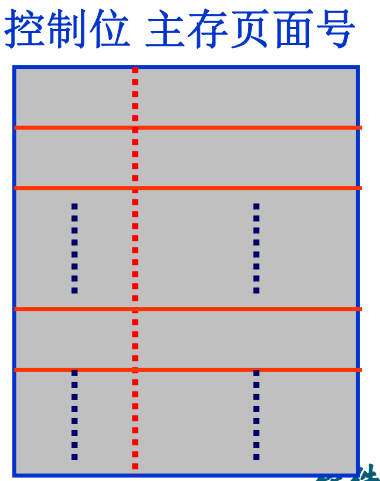
# 3.7.2 页式虚拟存储器

- 页式虚拟地址映射
  - 逻辑页：页式虚拟存储系统中，虚地址空间被分成等长大小的存储单位，每个存储单位就是一 页，每页中有固定个数的字存储单元，比如4K。
    - 虚拟地址=逻辑页号/高位+页内地址/偏移量/低位
  - 物理页：主存空间也被分成同样大小的页存储单位，称为物理页。
    - 物理地址=物理页号/高位+页内地址/偏移量/低位



# 3.7.2 页式虚拟存储器

- 页式虚拟地址映射
  - 页式虚拟地址映射：使用页表可以把虚地址（逻辑地址）转换成物理地址。页表在主存中，系统管理。
    - 每个进程对应一个页表。页表位置：页表基址寄存器
    - 页表中的行/页表项/表项：一个虚存页面的表项，表项的内容包含该虚存页面所在的主存页面的地址（物理页号），以及指示该逻辑页是否已调入主存的有效位。
    - 地址变换：用逻辑页号作为页表内的偏移地址索引页表（将虚页号看作页表数组下标）并找到相应物理页号，用物理页号作为实存地址的高字段，再与虚地址的页内偏移量拼接，就构成完整的物理地址。
      - 页表基地址+逻辑页号→物理页号
      - 物理页号+页内地址→物理地址



# 3.7.2 页式虚拟存储器

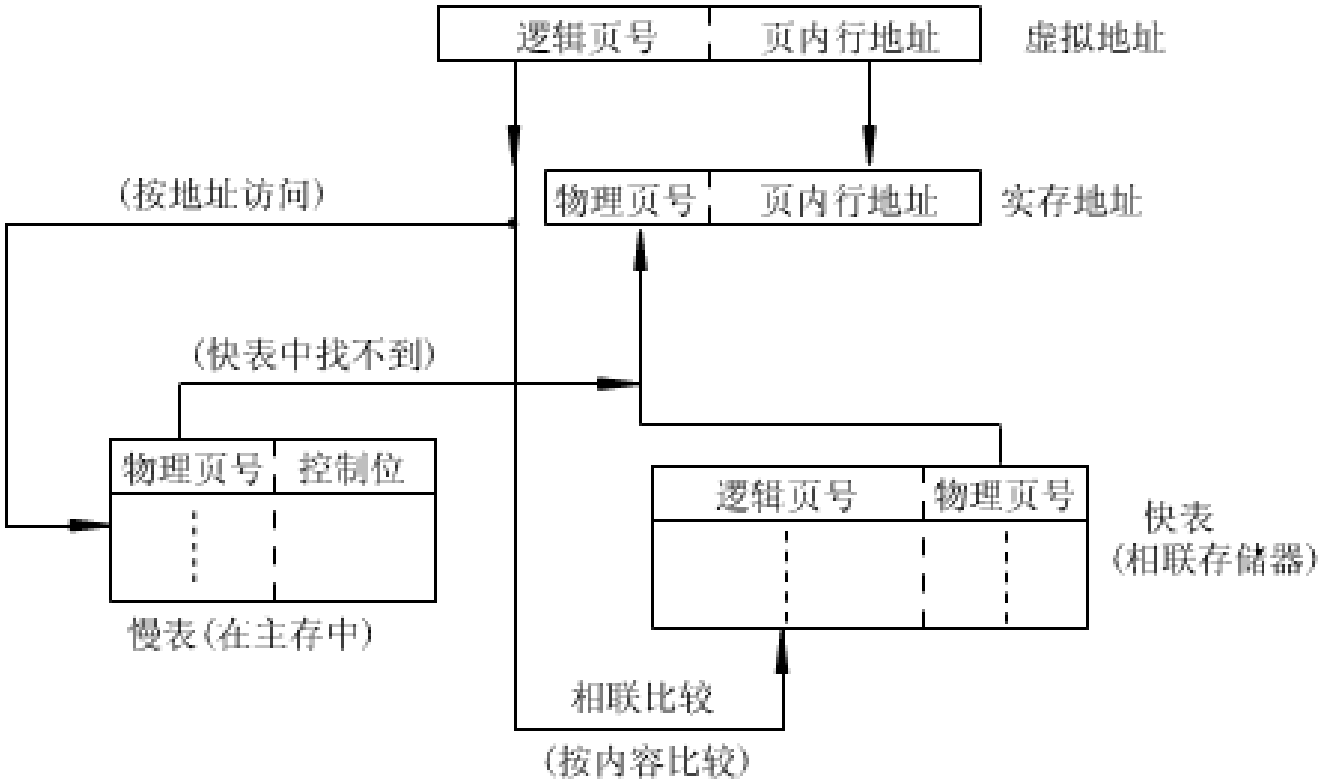
- 页式虚拟地址映射
  - 页表的大小：
    - 每个进程所需的页数并不固定，由于虚存地址空间可以很大，因而每个进程的页表有可能非常长。例如，如果一个进程的虚地址空间为2G字节，每页的大小为512字节，则总的虚页数为$2^{31}/2^{9}=2^{22}$。
    - 将页表存储在虚存中，当一个进程运行时，其页表中一部分在主存中，另一部分则在辅存中保存。
    - 采用二级页表结构。每个进程有一个页目录表，其中的每个表项指向一个页表。若页目录表的长度（表项数）是m,每个页表的最大长度（表项数）为n,则一个进程最多可以有m×n个页。
    - 在页表长度较大的系统中，还可以采用反向页表实现物理页号到逻辑页号的反向映射。

# 3.7.2 页式虚拟存储器

- 转换后援缓冲器 / TLB
  - 2次访存交换数据：时间倍增
    - 第1次：访问页表→物理页号→物理地址
    - 第2次：使用物理地址→定位/交换数据
  - 解决方法：使用TLB实现页表缓存
    - 转换后援缓冲器(TLB)：对页表本身实行二级缓存，把页表中的最活跃的部分存放在高速存储器中，组成快表。这个专用于页表缓存的高速存储部件通常称为TLB。
    - 完整的页表保存在主存中，称为慢表。

# 3.7.2 页式虚拟存储器

- 使用**TLB**实现地址映射
  - **Cache**：快表　　主存：慢表



# 3.7.2 页式虚拟存储器

- 内页表与外页表
  - 内页表：虚地址到主存物理地址的变换表
  - 外页表：虚地址与辅存地址之间的变换表
    - 当主存缺页时，调页操作首先要定位辅存，而外页表的结构与辅存的寻址机制密切相关。例如对磁盘而言，辅存地址包括磁盘机号、磁头号、磁道号和扇区号等。
- 虚拟存储器、TLB和外页Cache的协同工作
  - 操作系统管理：
  - 读出：虚地址→主存物理地址→Cache地址
  - 写回：Cache地址→主存物理地址→虚地址
  - 缺失问题：例11 P104

# 3.7.3 段式/段页式虚拟存储器

- 段式虚拟存储器
  - 逻辑段/段：段是按照程序的自然分界划分的长度可以动态改变的区域。通常把子程序、操作数和常数等不同类型的数据划分到不同的段中，并且每个程序可以有多个相同类型的段。
    - 虚拟地址=段号/高位+段内地址/偏移量/低位
  - 段式虚拟地址映射：使用段表可以把虚地址（逻辑地址）转换成物理地址。段表在主存中，系统管理。
    - 每个程序/进程对应一个段表。段表位置：段表基址寄存器
    - 段表中的行/段表项/表项：一个段的表项值，包含3个字段：

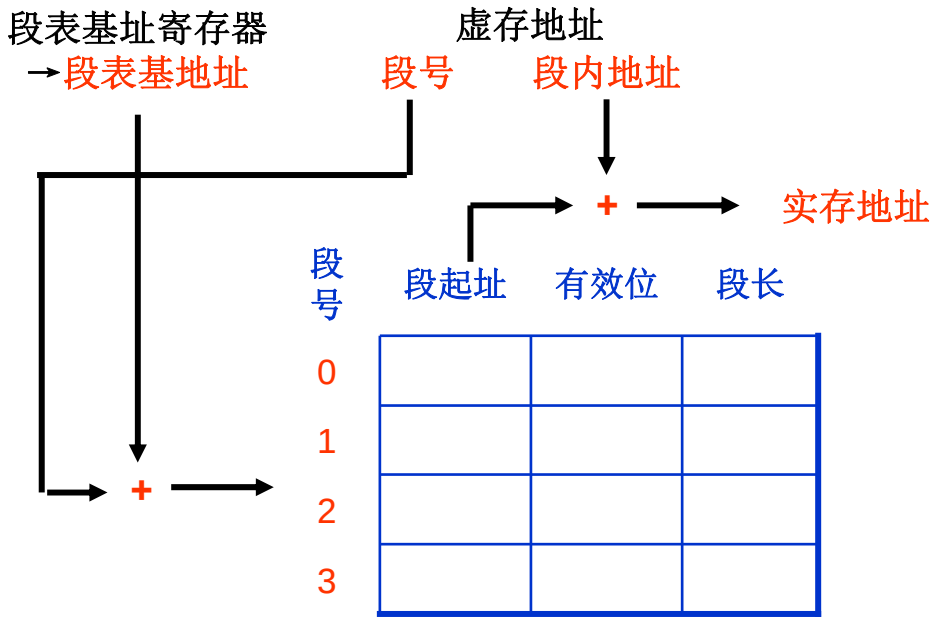
    (1)有效位：指明该段是否已经调入实存。

    (2)段起址：该段已经调入实存，指明该段在实存中的首地址。

    (3)段长：记录该段的实际长度。设置段长字段的目的是为了保证访问某段的地址空间时，段内地址不会超出该段长度导致地址越界而破坏其他段。
    - 段表本身也是一个段，可以存在辅存中，但一般是驻留在主存中。

# 3.7.3 段式/段页式虚拟存储器

- 段式虚拟存储器
  - 地址变换：用虚拟地址中的段号作为段表内的偏移地址索引页表（将段号看作段表数组下标）并找到相应段起址，用段起址作为实存地址的高字段，再与虚地址的段内偏移量拼接，就构成完整的物理地址。
    - 段表基地址＋段号→段起址
    - 段起址＋段内地址→物理地址
  - 段起址：段地址，段起始位置；




段表(在主存中)

# 3.7.3 段式/段页式虚拟存储器

- 段式虚拟存储器
  - 优点：

    ①段的逻辑独立性使其易于编译、管理、修改和保护，也便于多道程序共享。

    ②段长可以根据需要动态改变，允许自由调度，以便有效利用主存空间。
  - 缺点：

    ①因为段的长度不固定，主存空间分配比较麻烦。

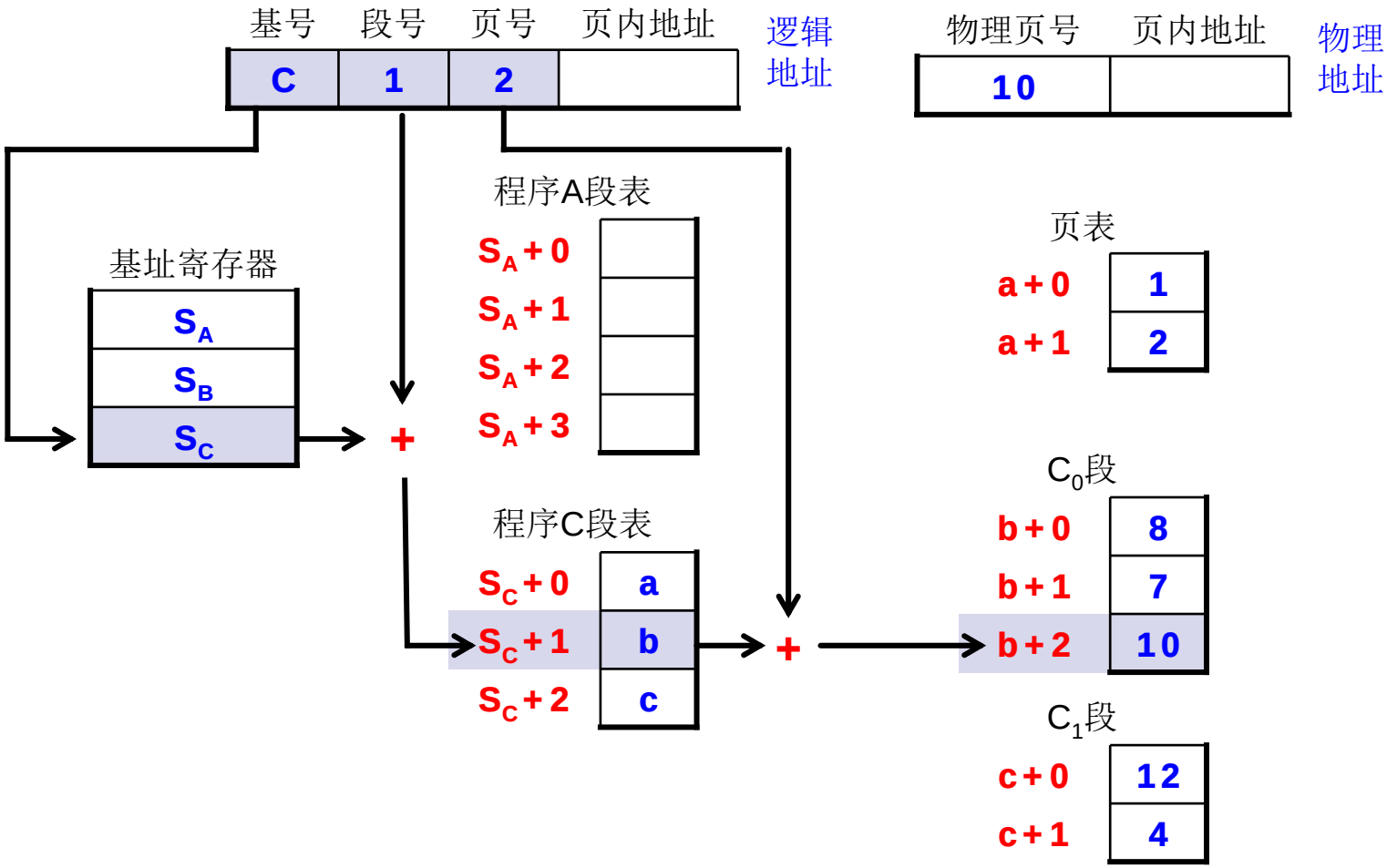
    ②容易在段间留下许多外碎片，造成存储空间利用率降低。

    ③由于段长不一定是2的整数次幂，因而不能简单地像分页方式那样用虚地址和实地址的最低若干二进制位作为段内偏移量，并与段号进行直接拼接，必须用加法操作通过段起址与段内偏移量的求和运算求得物理地址。因此，段式存储管理比页式存储管理方式需要更多的硬件支持。

# 3.7.3 段式/段页式虚拟存储器

- 段页式虚拟存储器
  - 结构：程序按逻辑结构分段，每个段再分为大小固定的页。
    - 虚拟地址=段号/高位+段内逻辑页号/低位+页内偏移/低位
  - 段式虚拟地址映射：使用1个段表和多个页表可以把虚地址（逻辑地址）转换成物理地址。段表/页表在主存中，系统管理。
    - 使用段号→定位段表项→定位页表+页号→页表项→物理页号+页内偏移
  - 多任务系统的段页式虚地址：
    - 虚拟地址=基号/高位+段号/高位+段内逻辑页号/低位+页内偏移/低位
    - 基号：程序在多任务系统中的序号。

- **段/页式虚拟存储器** 例12 P106 三个程序基号是A、B和C表，其基址寄存器的内容分别为$S_A$、$S_B$和$S_C$。程序A由4个段构成，程序C由3个段构成。段页式虚拟存储系统的逻辑地址到物理地址的变换过程如图所示。



# 3.7.4 虚存的替换算法

- 调页　当需要从辅存调页至主存而主存已满时，要进行主存页面的替换。
  - 替换算法：虚存替换算法与cache的替换算法类似，有：
    - 近期最少使用(LRU)算法
    - 最不经常使用(LFU)算法
    - 先进先出(FIFO)算法
  - 回写：
    - 对于将被替换出去的页面，假如该页调入主存后没有被修改,就不必进行处理，否则就把该页重新写入外存，以保证外存中数据的正确性。为此,在页表的每一行应设置一修改位。
  - 比较虚存替换与**cache替换**
    - cache的替换全部靠硬件实现，虚存的替换有操作系统的支持。
    - 虚存缺页对系统性能的影响比cache未命中要大得多，因为调页需要访问辅存，并且要进行任务切换。
    - 虚存页面替换的选择余地很大，属于一个进程的页面都可替换。

# 3.7.4 虚存的替换算法

- 例13 假设主存只允许存放a、b、c三个页面，逻辑上构成a进c出的FIFO队列。某次操作中进程访存的序列是0,1,2,4,2,3,0,2,1,3,2（虚页号）。若分别采用FIFO算法、FIFO+LRU算法，请用列表法分别求两种替换策略情况下主存的命中率。
  - 访存操作：如果有需要的页面号，则正常访问；如果没有，则替换。

| 页面访问序列 | | 0 | 1 | 2 | 4 | 2 | 3 | 0 | 2 | 1 | 3 | 2 | 命中率 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIFO 算法 | a | 0 | 1 | 2 | 4 | 4 | 3 | 0 | 2 | 1 | 3 | 3 | 2/11=18.2% |
| | b | | 0 | 1 | 2 | 2 | 4 | 3 | 0 | 2 | 1 | 1 | |
| | c | | | 0 | 1 | 1 | 2 | 4 | 3 | 0 | 2 | 2 | |
| | | | | | | 命中 | | | | | | 命中 | |
| FIFO+ LRU 算法 | a | 0 | 1 | 2 | 4 | 2 | 3 | 0 | 2 | 1 | 3 | 2 | 3/11=27.3% |
| | b | | 0 | 1 | 2 | 4 | 2 | 3 | 0 | 2 | 1 | 3 | |
| | c | | | 0 | 1 | 1 | 4 | 2 | 3 | 0 | 2 | 1 | |
| | | | | | | 命中 | | | 命中 | | | 命中 | |

# 3.8 奔腾序列机的虚存组织

- 基于IA-32体系结构的奔腾序列机为管理存储器提供了硬件支持，在cpu中有MMU部件，实现存储器管理功能。

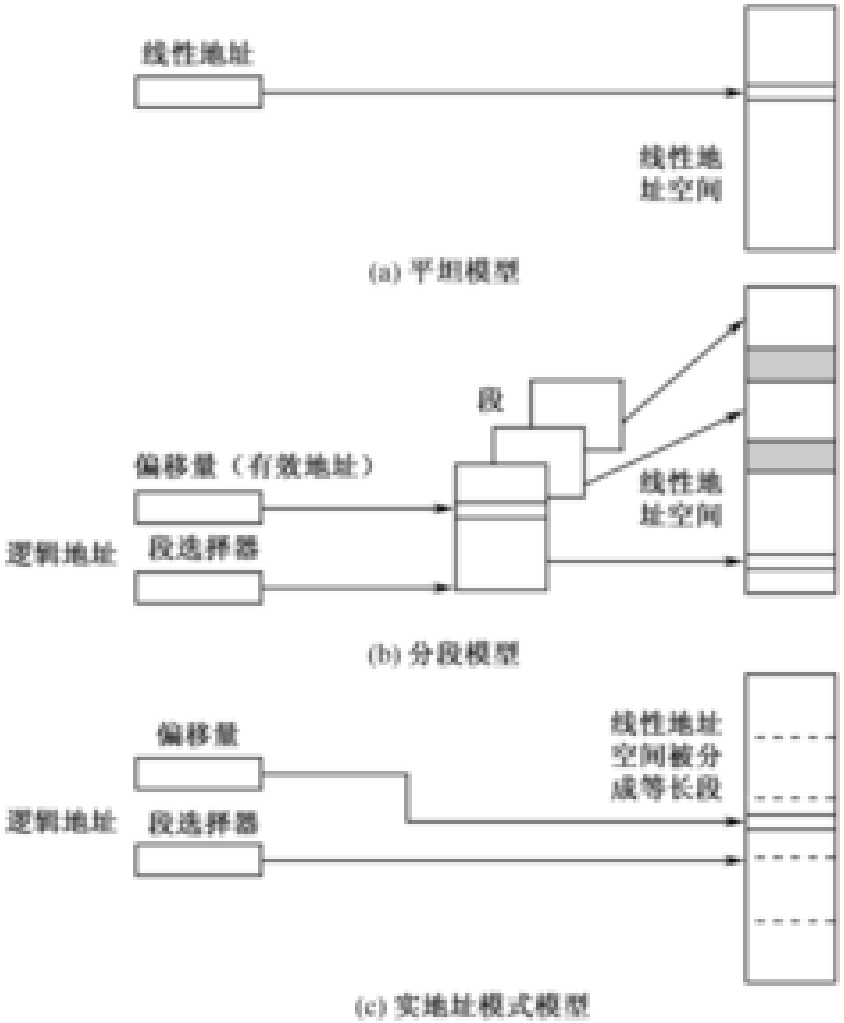
# 3.8.1 存储器模型

- **MMU** 支持3中存储器模型/内存使用模式：
  - 实地址存
  - 分段存储
  - 平坦存储

emory model,
分为逻辑段，每
$2^{20}$B，逻辑地址
由段基址

el，每个程序都使
个段，每个段的
成，MMU将逻
辑地址转

存被组织成单一
的、连续
数据和堆栈都在
该地址空



# 3.8.2 虚地址模式

- 虚存组织：
  - IA-32体系结构微处理机的虚拟存储器可以通过两种方式实现：分段和分页。
  - 存储管理部件MMU包括分段部件SU和分页部件PU两部分。
  - 分段部件将程序中使用的虚地址转换成线性地址。
  - 分页部件则将线性地址转换为物理地址。
  - 逻辑地址/虚拟地址：程序中提供的内存地址。
  - 线性地址：逻辑地址转换为物理地址的中间层。
  - 物理地址：定位物理内存单元需要的地址信号。
  - 如果没有分页，线性地址就是物理地址。

# 3.8.2 虚地址模式

- 虚存组织：
  - 不分段不分页模式：。
  - 分段不分页模式：。
  - 不分段分页模式：。
  - 分段分页模式：。

# 3.8.3 分页模式下地址转换

- 虚存组织：
  - 不分段不分页模式：。