第三章 多层次的存储器

谢在鹏
勤学楼4203
Email: zxiehhu@163.com
QQ：29780805

内容简介

- 存储器分类、分级，存储器技术指标
- SRAM存储器、DRAM存储器、只读存储器
- 双端口存储器、交叉存储器
- 高速缓冲存储器cache技术
- 虚拟存储器技术

重点内容

- 基本概念

  存储容量、存取时间、存储周期、虚拟存储器
- 存储器的分级结构
- 主存储器的逻辑设计
- 顺序存储器和交叉存储器的定量分析
- 高速缓冲存储器cache的基本原理，cache命中率相关计算

课后作业

P115：1、5、7、8、9

3.1 存储器概述

- 存储器的分类
- 存储器的分级
- 存储器技术指标

3.1.1 存储器分类

—— 存储介质

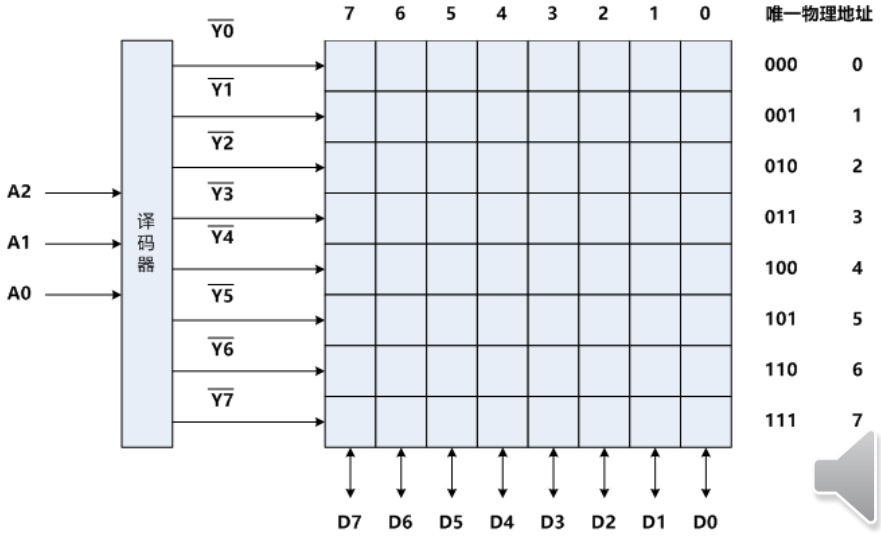
半导体存储器：半导体器件组成。

磁表面存储器：磁性材料组成。

3.1.1 存储器分类

—— 存取方式

随机存储器：任何存储单元的内容都能被随机存取，且存取时间与存储单元的物理位置无关，如半导体存储器。

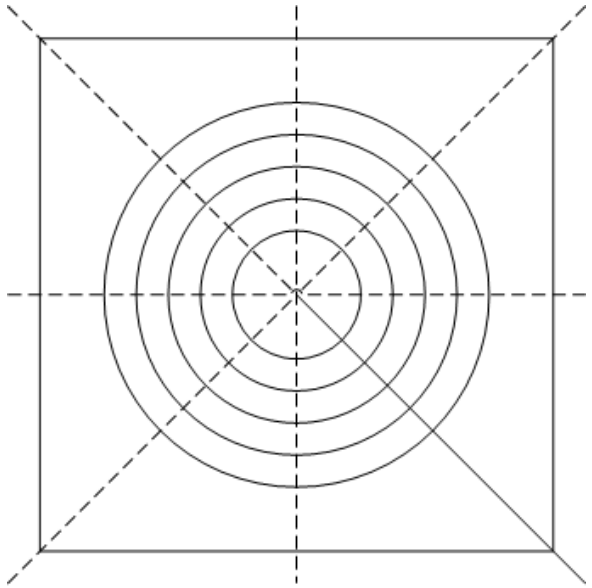


3.1.1 存储器分类

—— 存取方式

顺序存储器：只能按照固定的顺序存取，存取时间与存储单元的物理位置有关，如磁带。

半顺序存储器：随机与顺序相结合，如磁盘，磁道、扇区是随机存取，而一个确定的磁道、扇区内是顺序存取的。



软磁盘的磁道、扇区

3.1.1 存储器分类

—— 存储内容可变性

可读可写存储器（RAM）：既能读出又能写入的半导体存储器。

只读存储器（ROM）：存储的内容是固定不变的，正常工作时只能读出而不能写入的半导体存储器。只有通过特殊手段（如紫外线照射、高压）才能写入、改变内容。

3.1.1 存储器分类

—— 信息易失性

易失性存储器：断电后信息消失，半导体存储器。

非易失性存储器：断电后仍能保存信息，磁表面存储器。

3.1.1 存储器分类

—— 作用地位

内部存储器、外部存储器。

高速缓冲存储器、主存储器、辅助存储器、控制存储器。

3.1.2 存储器的分级

对存储器的要求是容量大、速度快、成本低，但是在一个存储器中要求同时兼顾这三个方面的要求是困难的。

为了解决这方面的矛盾，目前在计算机系统中通常采用多级存储器体系结构（多层次存储器），即高速缓冲存储器、主存储器、辅助存储器。

3.1.2 存储器的分级

CPU内部寄存器组

CPU片内高速缓冲存储器（cache）

CPU片外高速缓冲存储器（cache）

内部存储器（内存、主存）

外部存储器（外存、辅存）

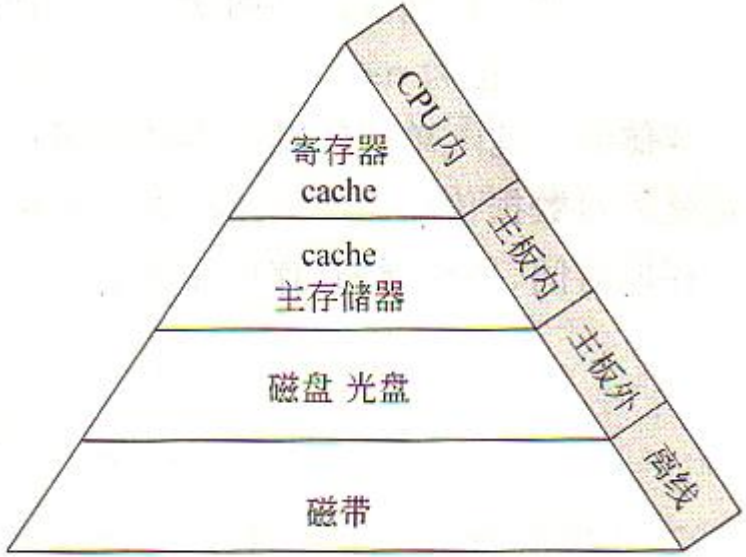
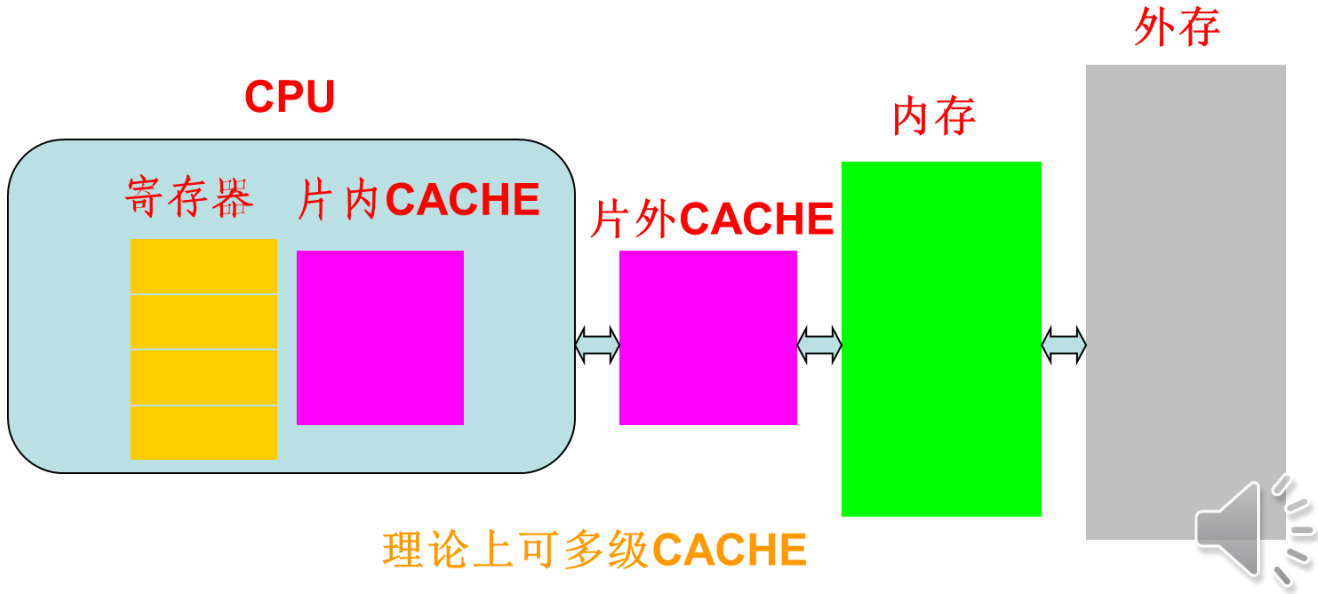


图 3.1　存储器的分级结构

3.1.2 存储器的分级

CPU能直接访问高速缓冲存储器cache和内存；外存信息必须调入内存后才能为CPU进行处理。



3.1.2 存储器的分级

高速缓冲存储器：高速小容量半导体存储器，强调快速存取指令和数据。

主存储器：介于cache与外存储器之间，用来存放计算机运行期间的大量程序和数据。要求选取适当的存储容量和存取周期，使它能容纳系统的核心软件和较多的用户程序。

辅助存储器：大容量辅助存储器，强调大的存储容量，以满足计算机的大容量存储要求，用来存放系统程序、应用程序、数据文件、数据库等。

3.1.3 存储器技术指标

存储容量：指一个存储器中可以容纳的存储单元总数。典型的存储单元存放一个字节，因此通常用字节数来表示。

1KB $= 2^{10}$ B　　1MB $= 2^{20}$ B

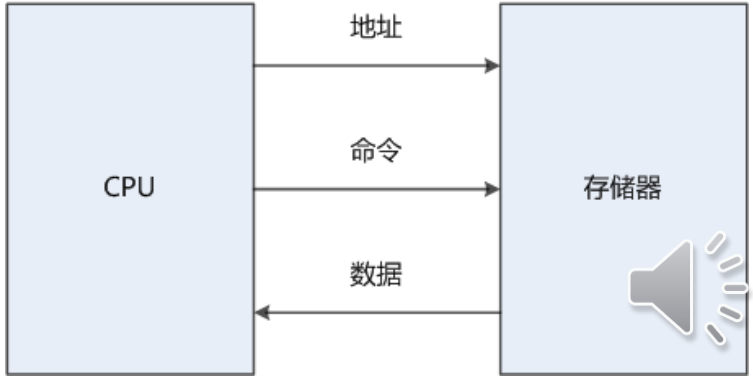
1GB $= 2^{30}$ B　　1TB $= 2^{40}$ B

3.1.3 存储器技术指标

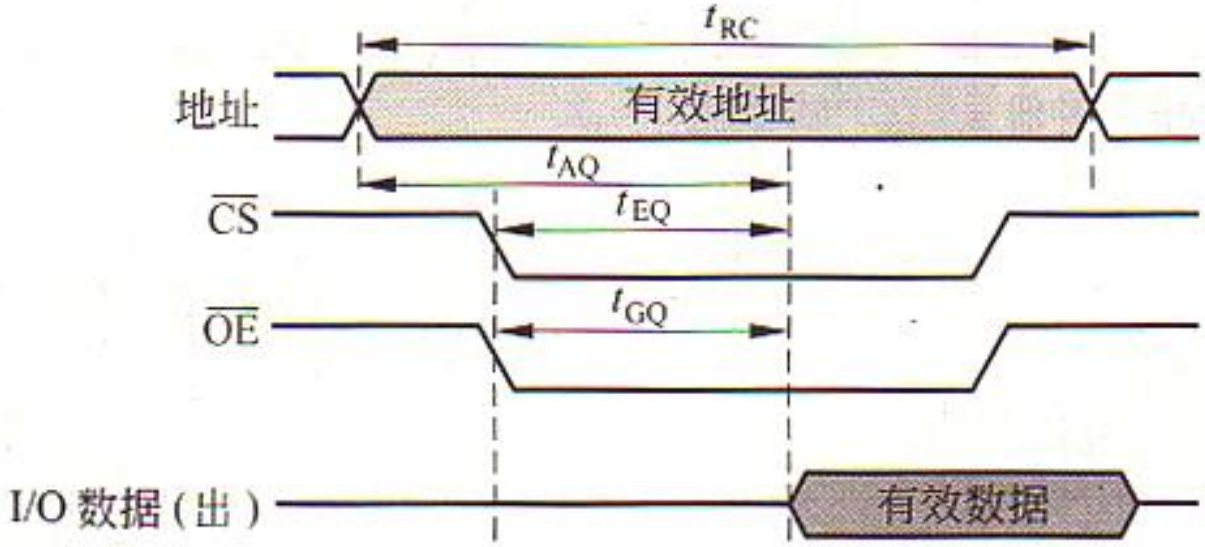
存取时间、存取周期：存取时间也称为读写时间，包括读时间、写时间。存取周期也称为读写周期，包括读周期、写周期。存取周期略大于存取时间。

3.1.3 存储器技术指标

以存储器读操作为例说明（读时间、读周期）：读时间指CPU输出地址、发出读操作命令，到存储器读出数据到数据总线上的时间。但读出的数据还需要经过数据总线、CPU内部数据通路，到运算器/控制器，然后才能启动下一次读操作。读周期是连续启动两次读操作所需最小间隔时间。



3.1.3 存储器技术指标



(a) 读周期 ($\overline{WE}$ 高)

读时间：t_{AQ} 反映存储器本身的存取速度。

读周期：t_{RC} 反映存储器和CPU协同的系统级存取速度。

3.1.3 存储器技术指标

存储器带宽：单位时间里存储器所能存取的信息量，通常以位/秒或字节/秒做度量单位。

3.2 SRAM存储器

- 基本的静态存储元阵列
- 基本的SRAM逻辑结构
- 读写周期波形图

3.2.1 基本的静态存储元阵列

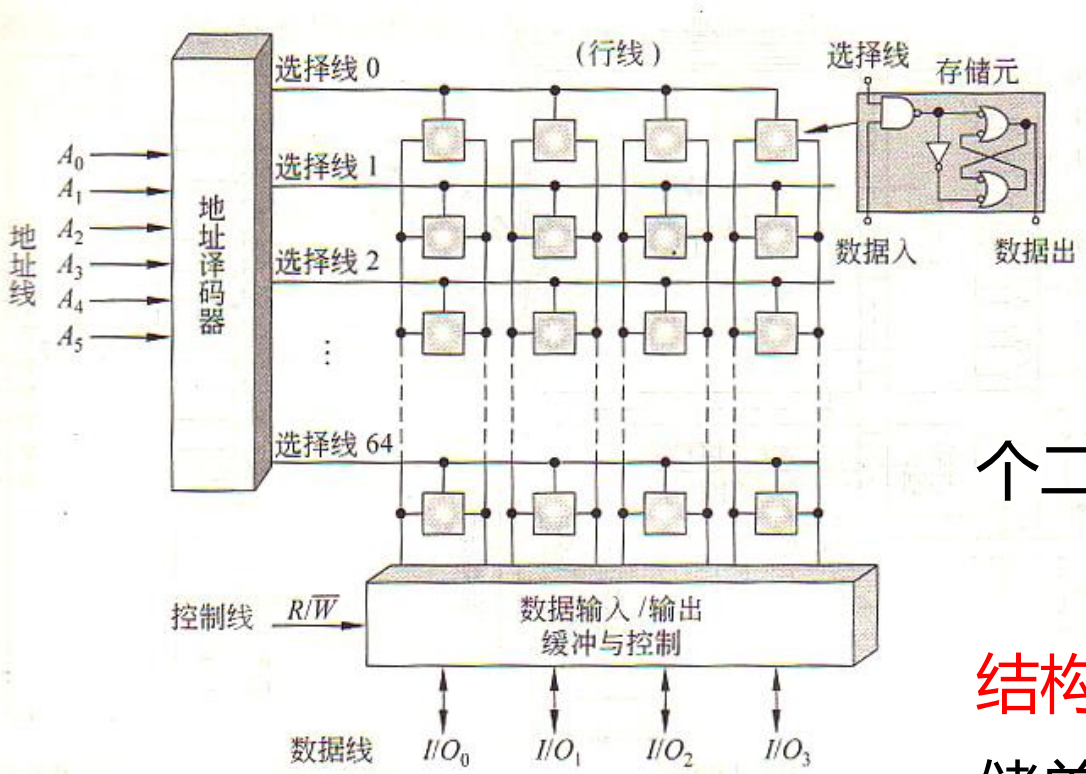
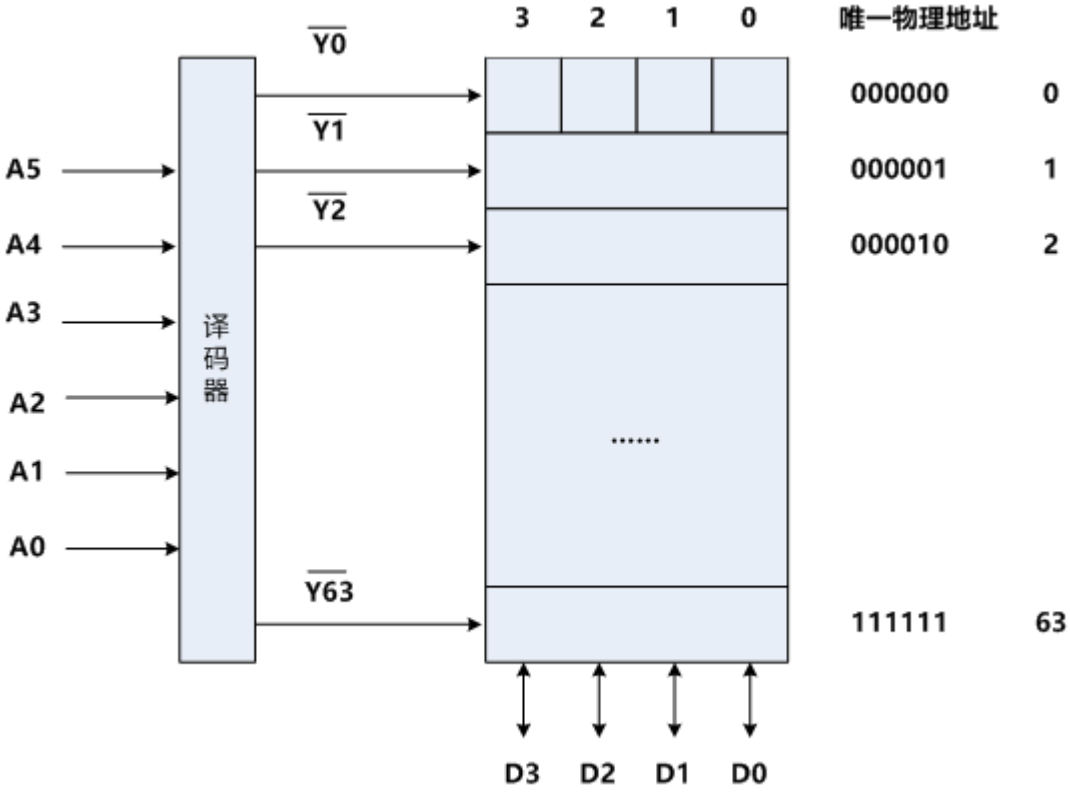


图 3.2 基本的静态存储元阵列

存储元：触发器，存放一个二进制位。

存储元阵列：存储元组织结构，64个存储单元，每个存储单元可存放4个二进制位。

3.2.1 基本的静态存储元阵列



3.2.1 基本的静态存储元阵列

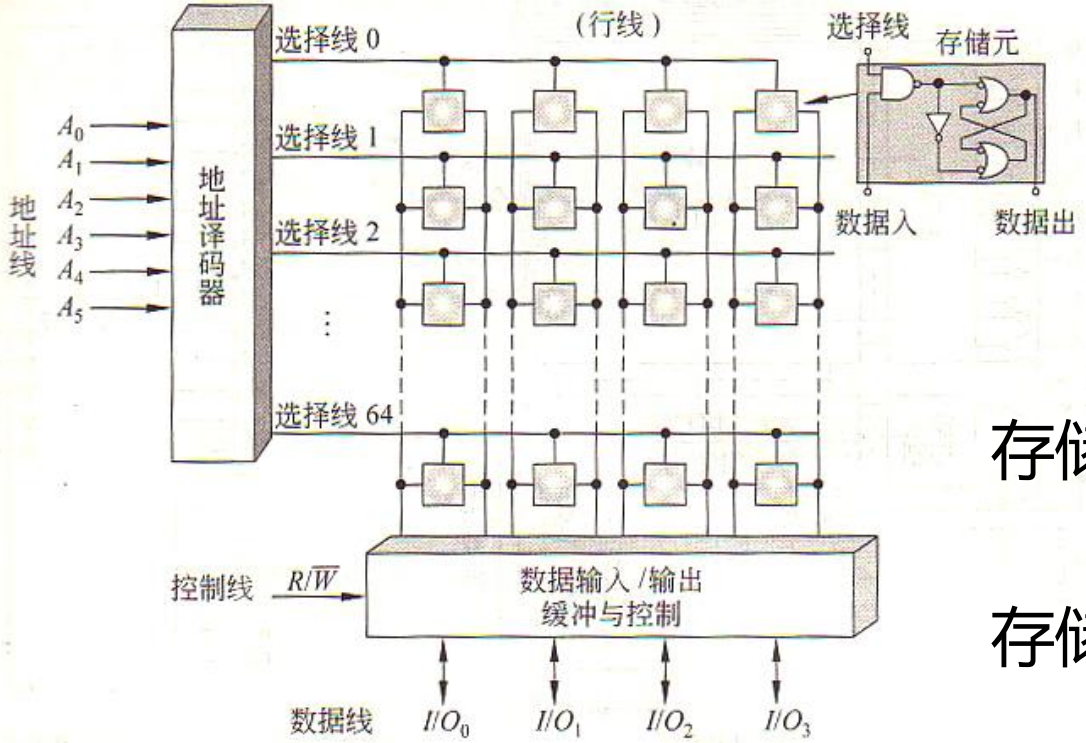


图 3.2 基本的静态存储元阵列

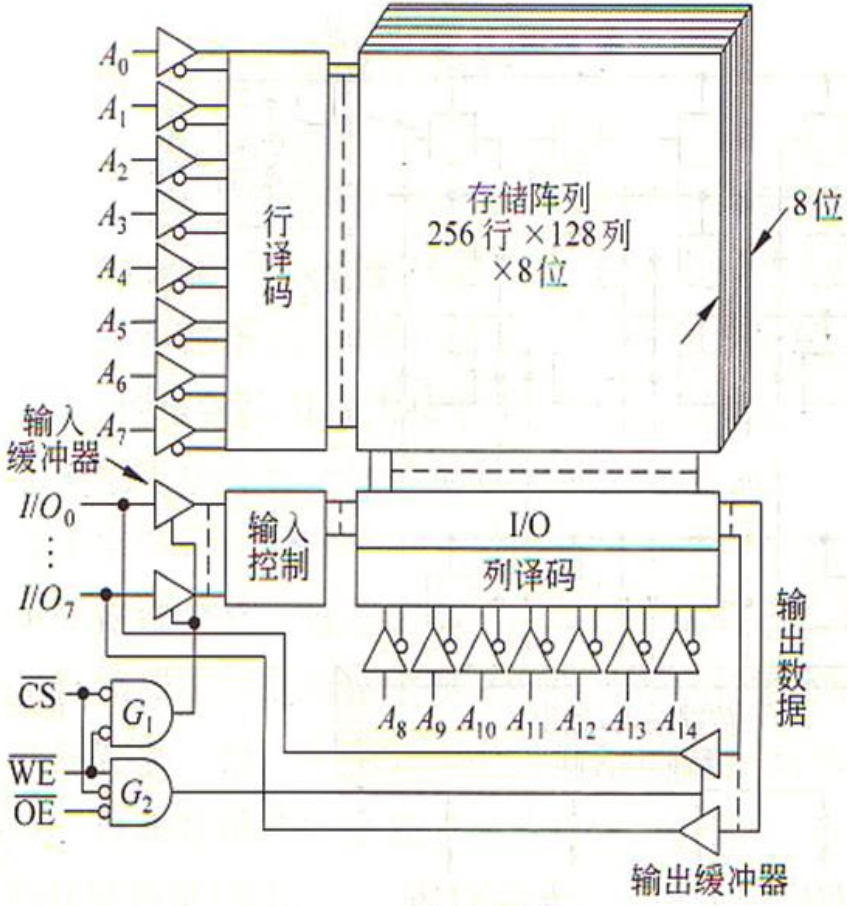
地址线：6位，2^6=64个存储单元，CPU ->存储器。

数据线：4位，CPU <-> 存储器。

控制线：1位，读写控制信号$R/\overline{W}$。

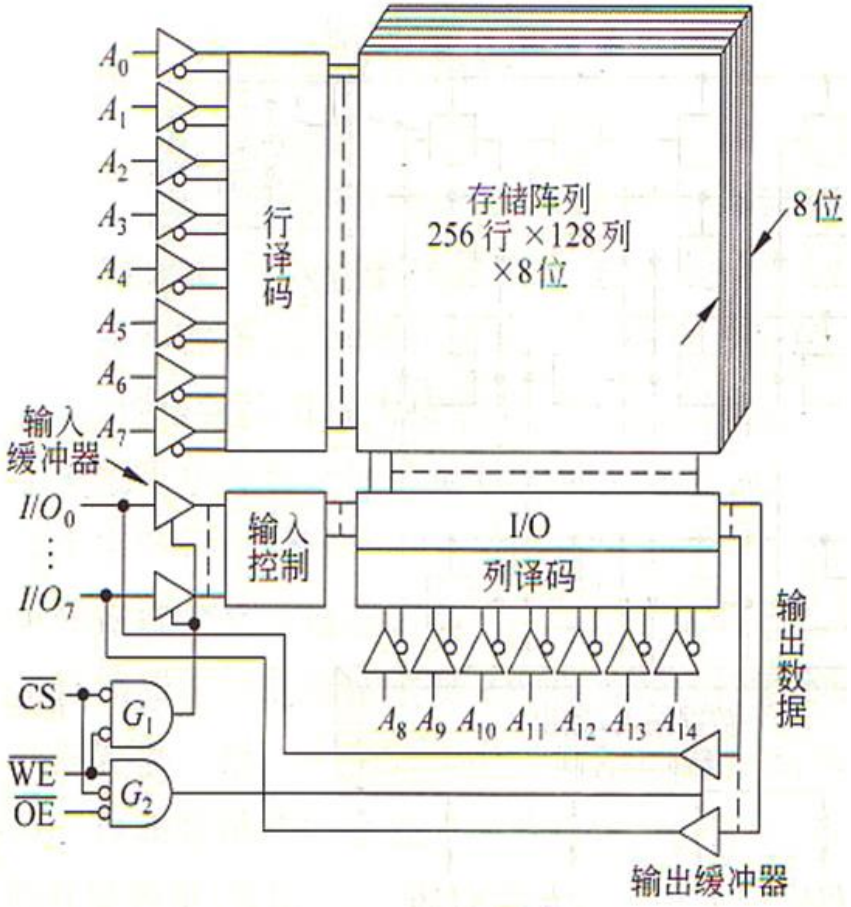
3.2.2 基本的SRAM逻辑结构

存储器容量：32K个存储单元，每个存储单元可存放8个二进制位，即32KB。



3.2.2 基本的SRAM逻辑结构

存储元阵列：8个256行*128列存储元阵列面，相同点元同时I/O。

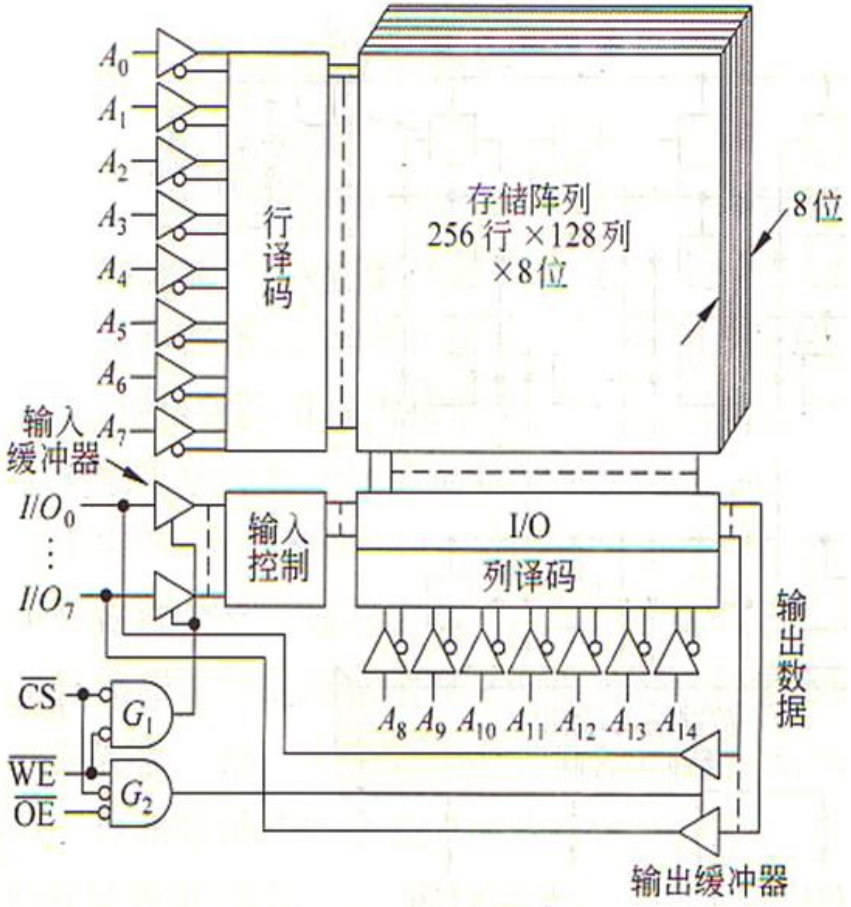


3.2.2 基本的SRAM逻辑结构

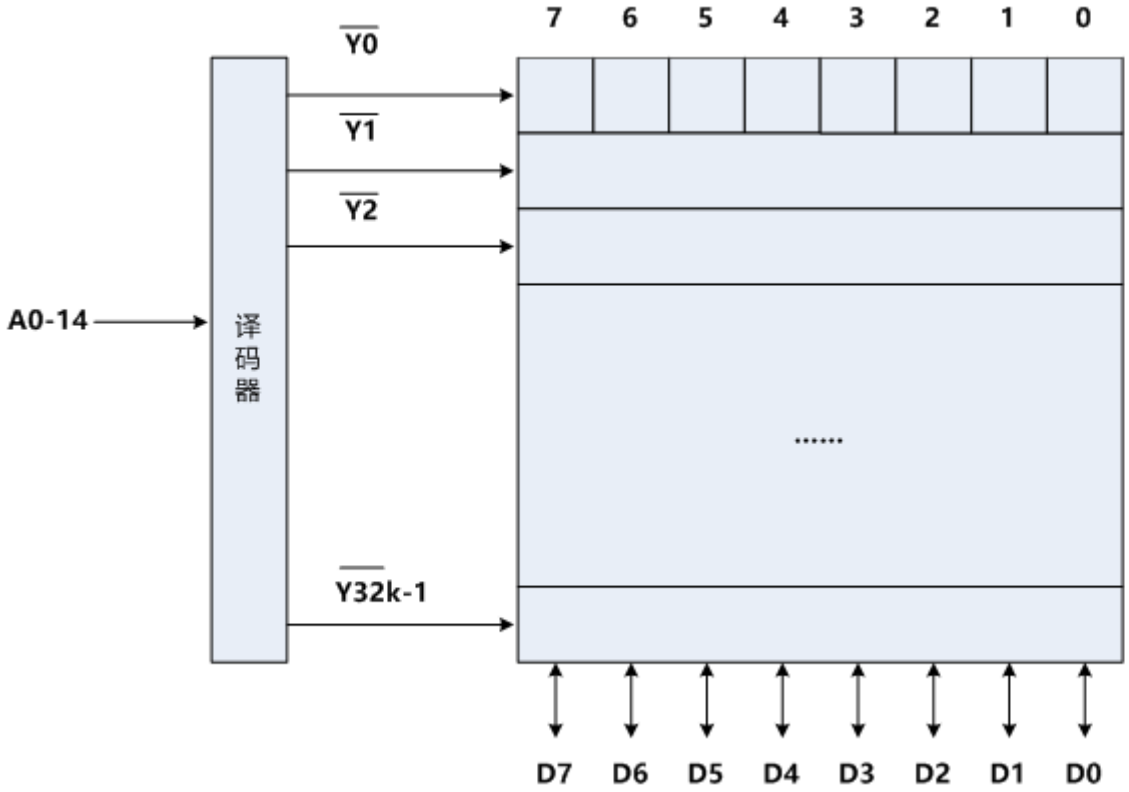
选择存储单元 =

选择存储元阵列面上的点

+8各面相同点同时输出。



3.2.2 基本的SRAM逻辑结构



唯一物理地址

000 0000 0000 0000 0

000 0000 0000 0001 1

000 0000 0000 0010 2

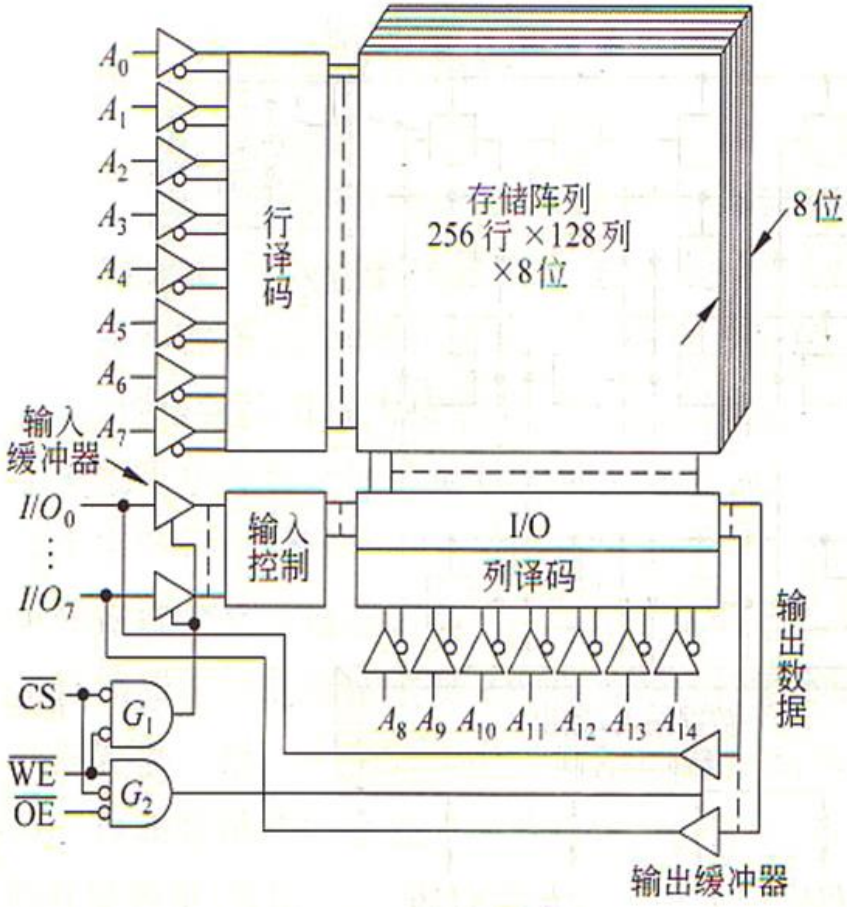
111 1111 1111 1111 32k-1

3.2.2 基本的SRAM逻辑结构

地址线：15根$A_{0\text{-}14}$、双译码方式（行8位 * 列7位）。

数据线：$I/O_{0\text{-}7}$。

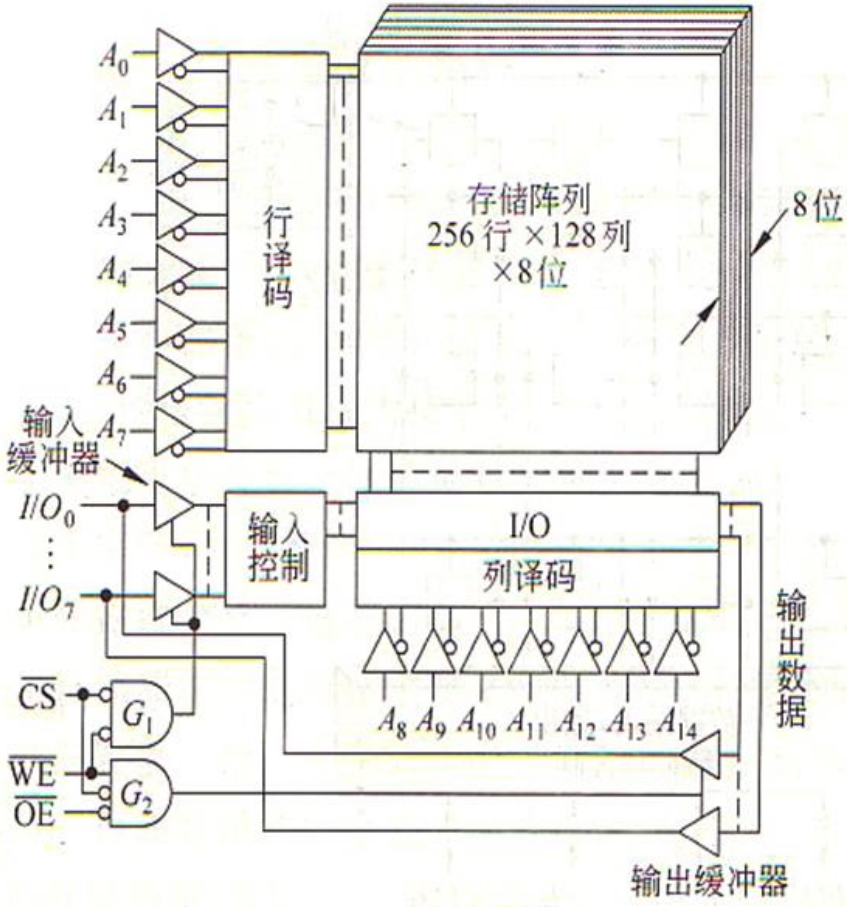
控制线：$\overline{CS}$（片选）、$\overline{WE}$（写有效）、$\overline{OE}$（读有效）。



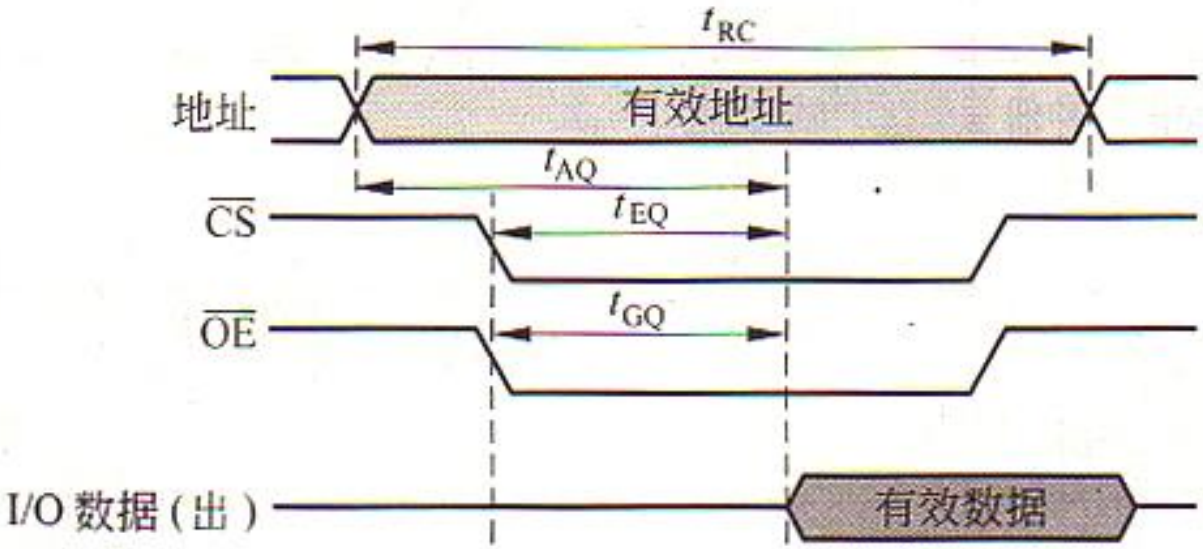
3.2.2 基本的SRAM逻辑结构

单译码方式：输入线15根，输出线2^{15}=32K根。

双译码方式：输入线15根，输出线2^8=256、2^7=128，合计384根。输出线少，便于集成，提高存储元阵列密度。



3.2.3 读写周期波形图

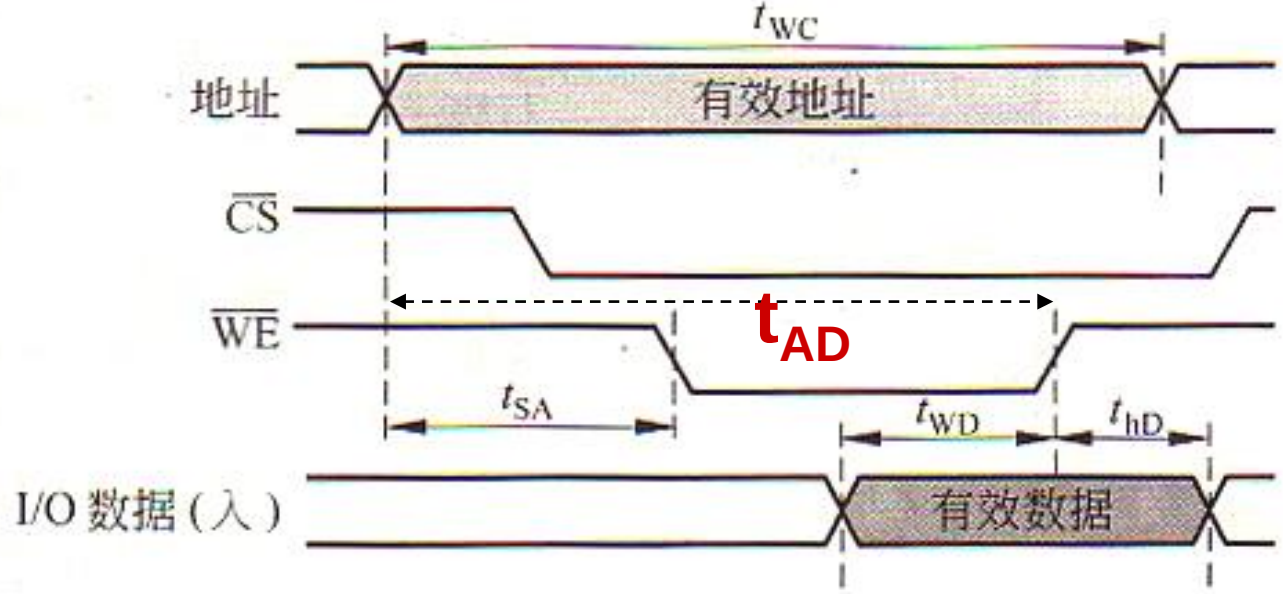


(a) 读周期 ($\overline{WE}$ 高)

读操作的过程：地址稳定输出到地址总线上即地址信号有效→片选信号、读信号有效→数据稳定输出到数据总线上。

读时间：t_{AQ}　　读周期：t_{RC}

3.2.3 读写周期波形图



(b) 写周期（$\overline{WE}$ 低）

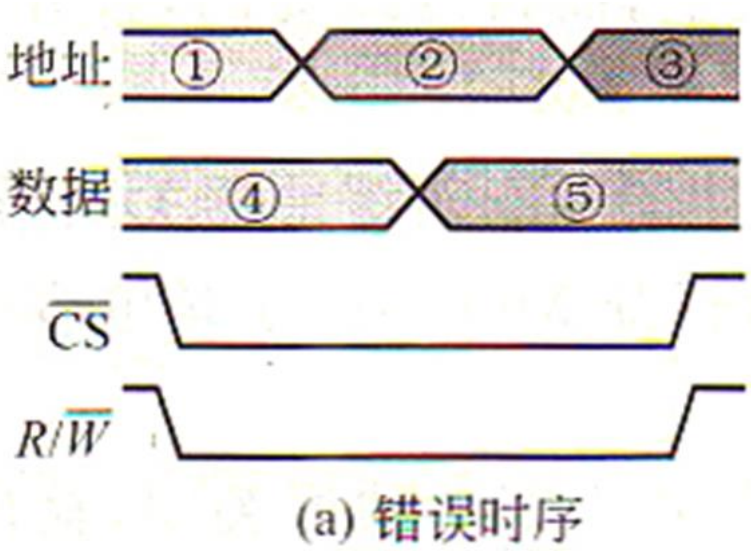
写操作的过程：地址稳定输出到地址总线上即地址信号有效→片选信号、写信号有效→写数据稳定出现在数据总线上→数据可靠写入存储器。

写时间：t_{AD}　　写周期：t_{WC}

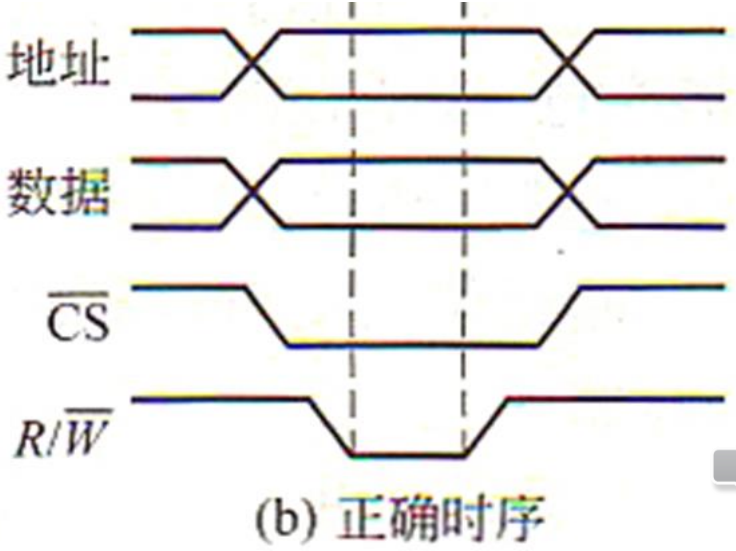
3.2.3 读写周期波形图

[例]下图是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。请指出下图写入时序中的错误，并画出正确的写入时序图。

[解]在写周期内，地址信号、数据信号、片选信号、写信号应该是稳定不变的。

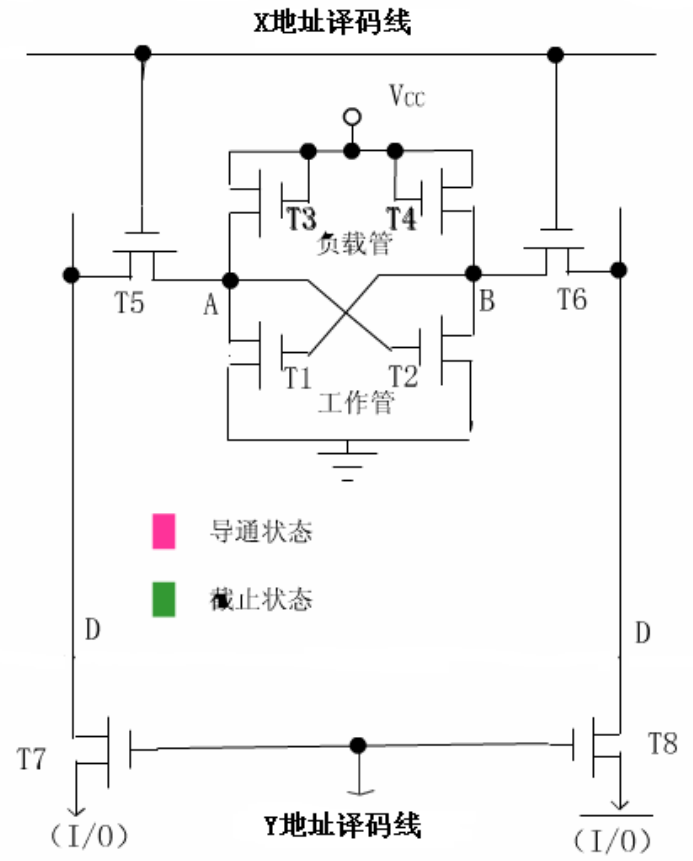


(a) 错误时序



(b) 正确时序

3.2 SRAM存储器

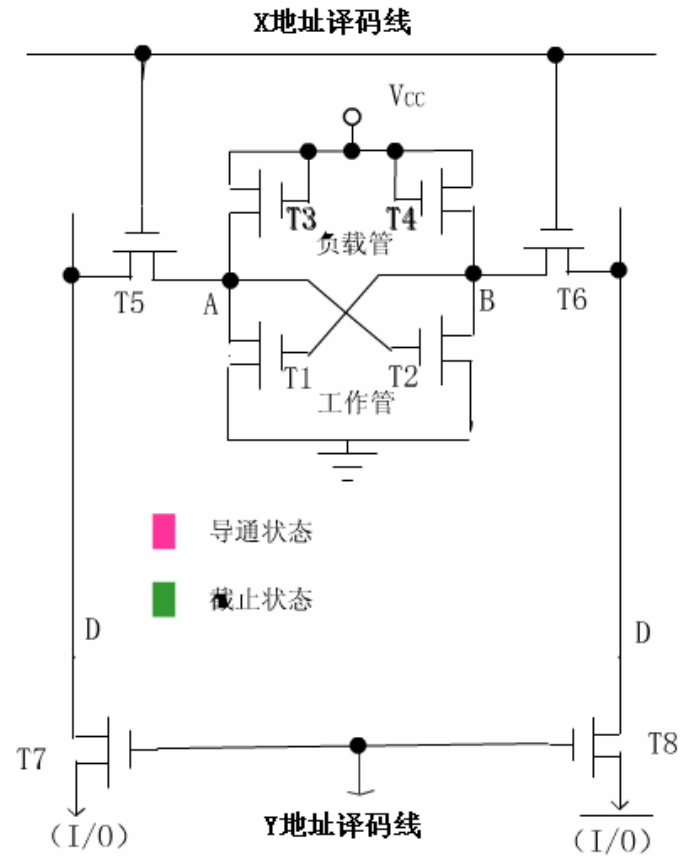


SRAM存储元采用双稳态交叉反馈电路，可以靠自身维持信息不变，不需要外部电路定期刷新、补充电荷。

“1”：A点高电位、B点低电位

“0”：A点低电位、B点高电位

3.2 SRAM存储器



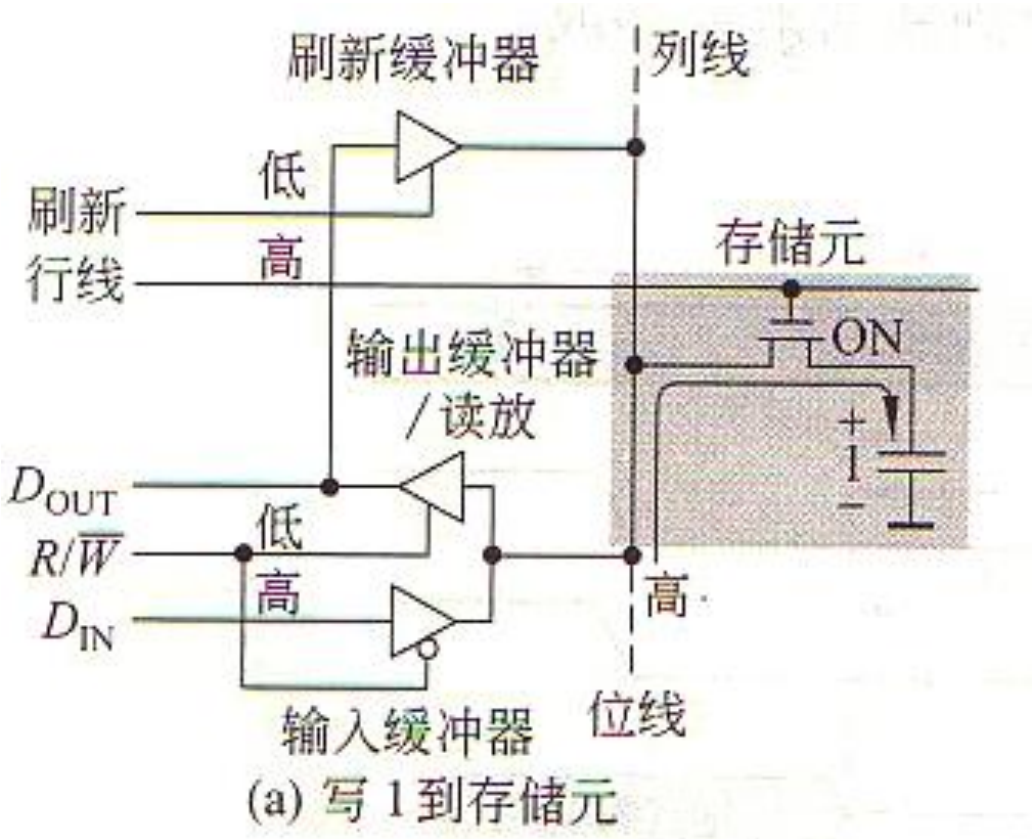
SRAM存储器特点：不掉电情况下，可稳定维持信息不变。

SRAM存储器缺点：MOS管多，不利于大容量集成。

3.3 DRAM存储器

- DRAM存储元的记忆原理
- DRAM芯片的逻辑结构
- 读写周期
- 存储器容量的扩充（主存储器逻辑设计）

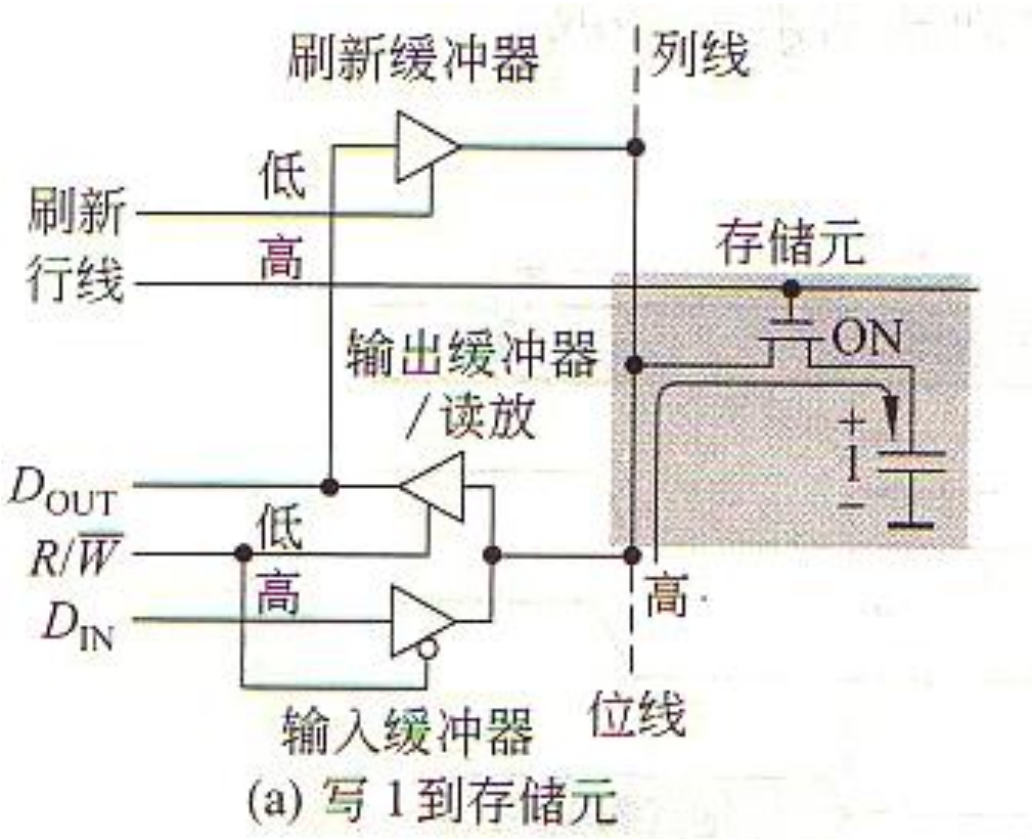
3.3.1 DRAM存储元的记忆原理



(a) 写 1 到存储元

- 1个MOS管+1个电容
- MOS管一个开关

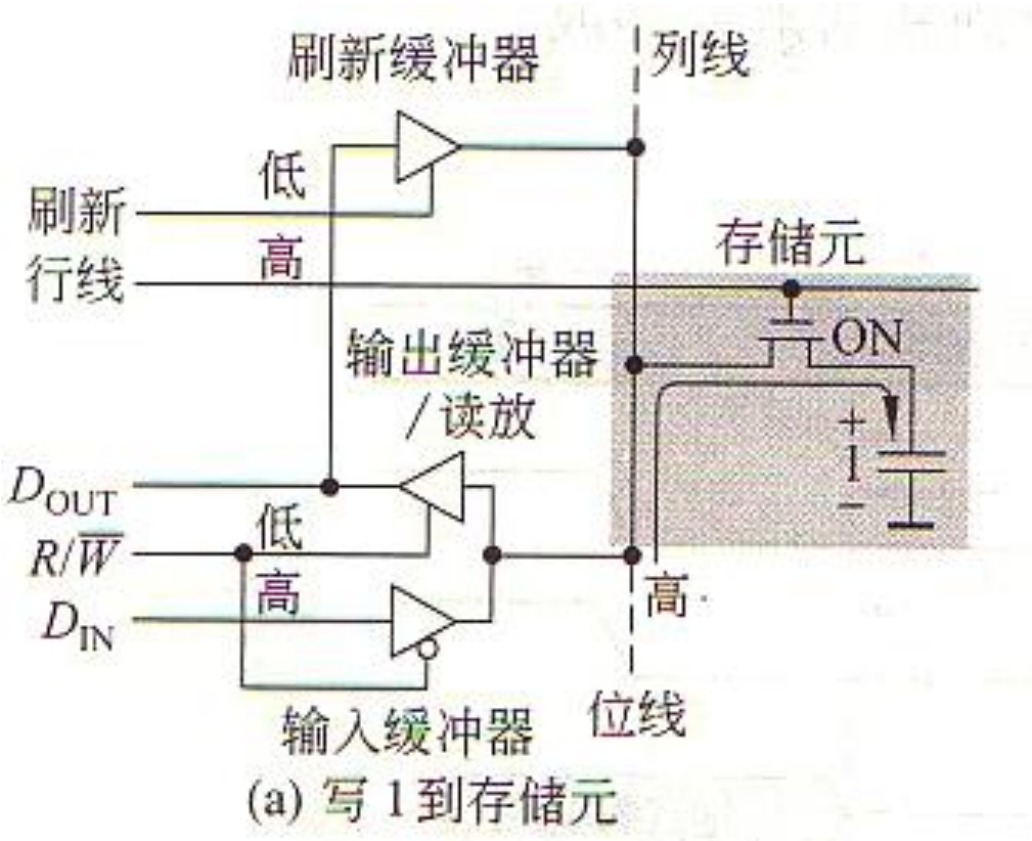
3.3.1 DRAM存储元的记忆原理



(a) 写 1 到存储元

写1：输入缓冲器打开，输入数据Din=1送到存储元位线上，而行选线=1打开MOS管，位线上的高电平给电容充电、存储电荷，表示存储了“1”。

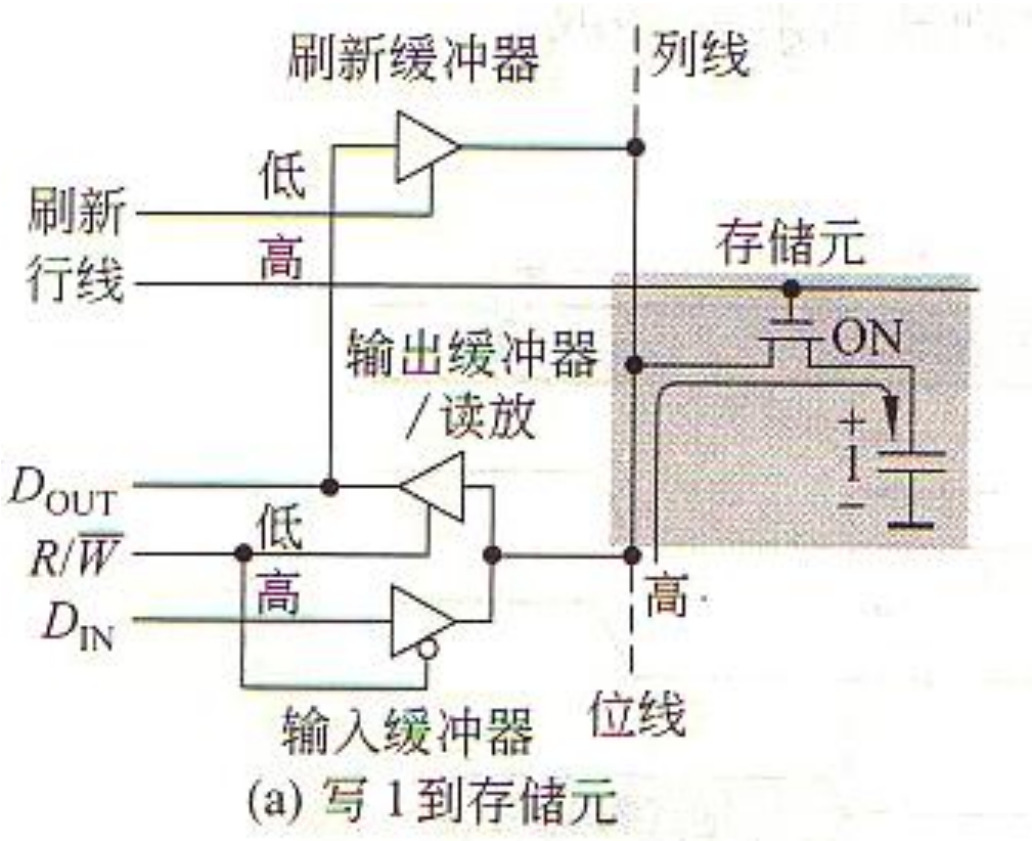
3.3.1 DRAM存储元的记忆原理



(a) 写 1 到存储元

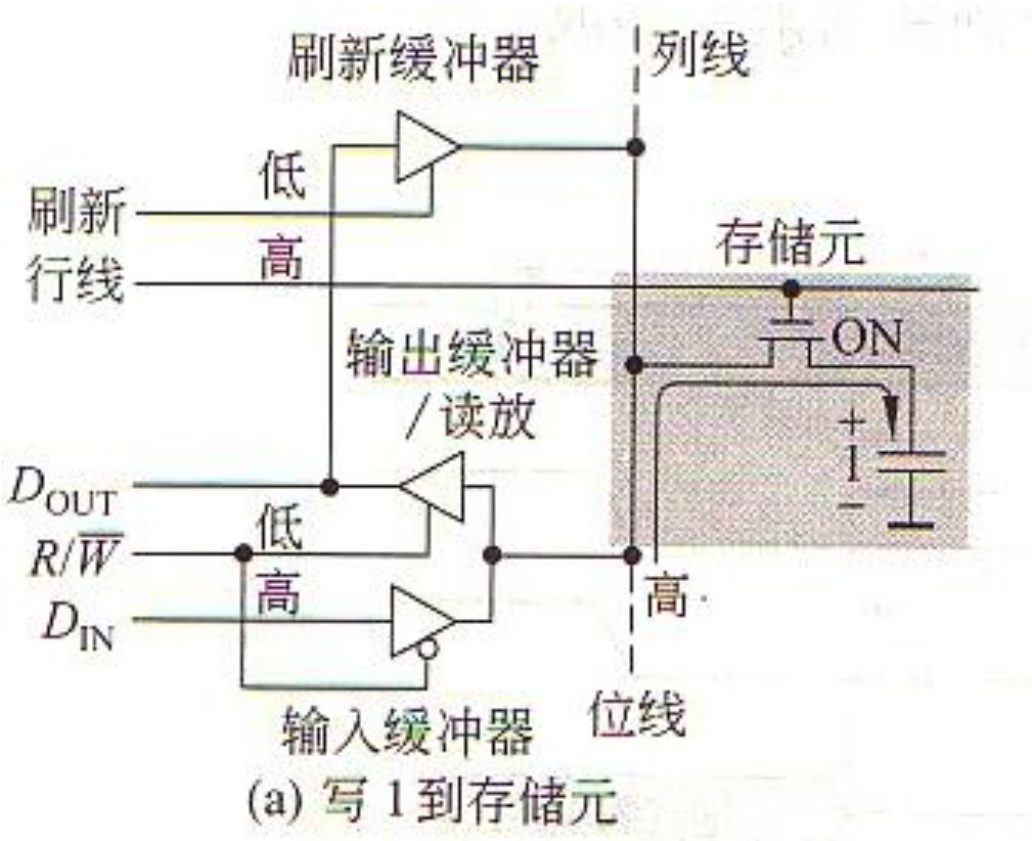
写0：输入缓冲器打开，输入数据Din=0送到存储元位线上，而行选线=1打开MOS管，电容上的电荷通过MOS管和位线放电，表示存储了“0”

3.3.1 DRAM存储元的记忆原理



读操作：输出缓冲器/读出放大器打开，行选线=1打开MOS管。如果存储的“1”，则电容上存储了足够的电荷，将通过位线、输出缓冲器/读出放大器发送到Dout上，即Dout=1；如果存储的“0”，则电容上无足够的电荷输出，即Dout=0。

3.3.1 DRAM存储元的记忆原理



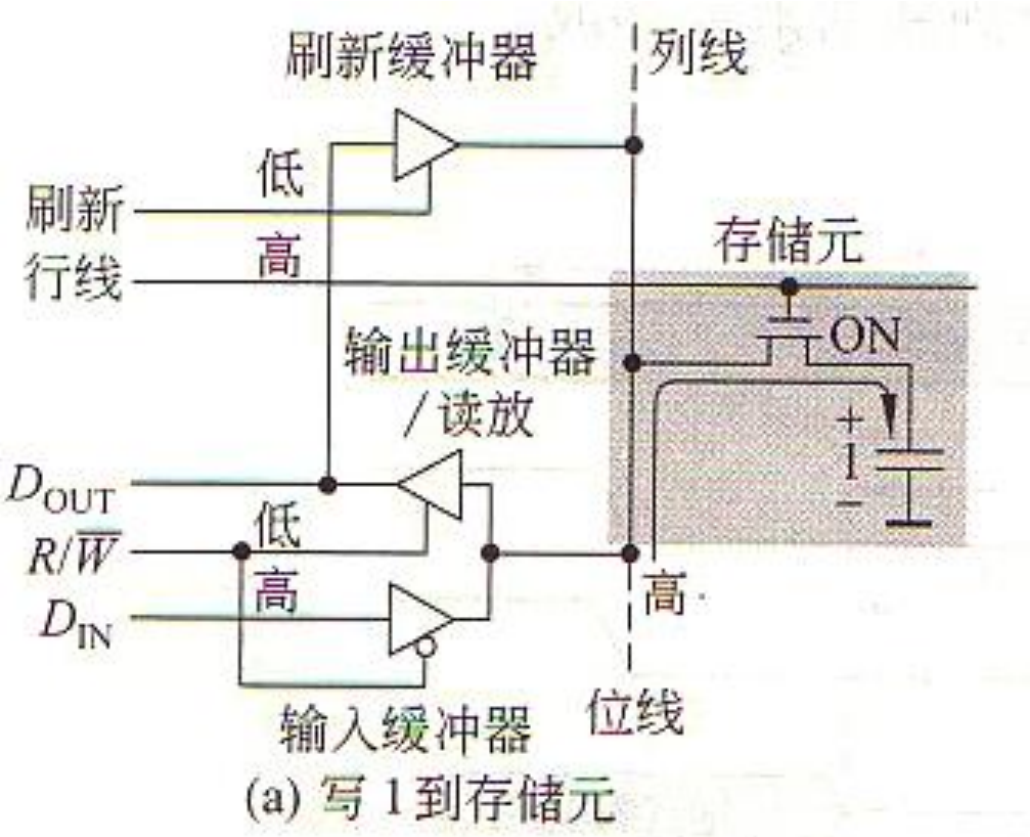
(a) 写 1 到存储元

为什么要刷新存储元的“1”？

（1）存储“1”，电容上驻留足够的电荷，但是MOS管、电容总会有泄漏，不能长时间维持足够的电荷；

（2）读“1”是破坏性读出，因为驻留电荷经位线、输出缓冲器/读出放大器、释放电荷。

3.3.1 DRAM存储元的记忆原理

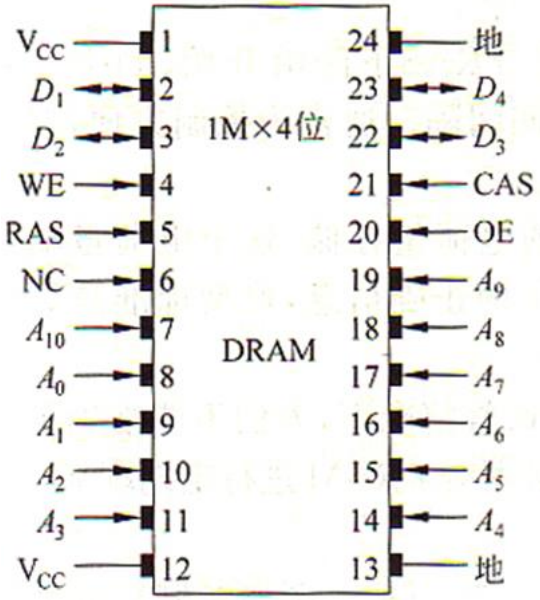


如何刷新存储元的“1”？

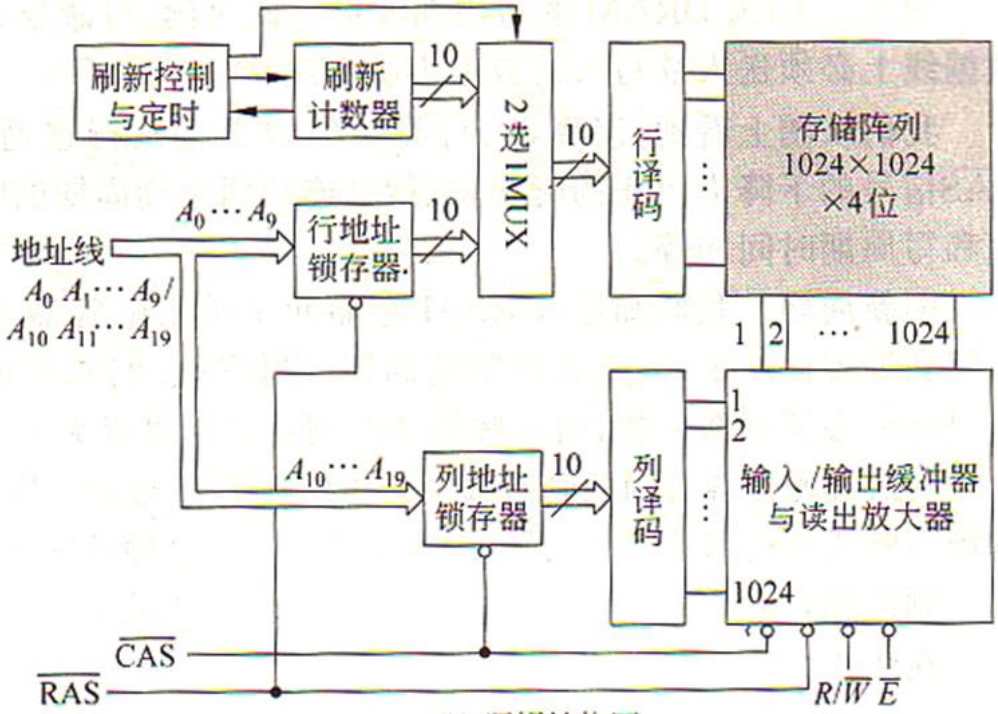
（1）读出Dout=1时刷新，Dout=1经过刷新缓冲器送到位线上，再经MOS管写到电容上。

（2）定时刷新，按行刷新、规定时间范围内刷新存储阵列一行中全部存储元。

3.3.2 DRAM芯片的逻辑结构



(a) 管脚图

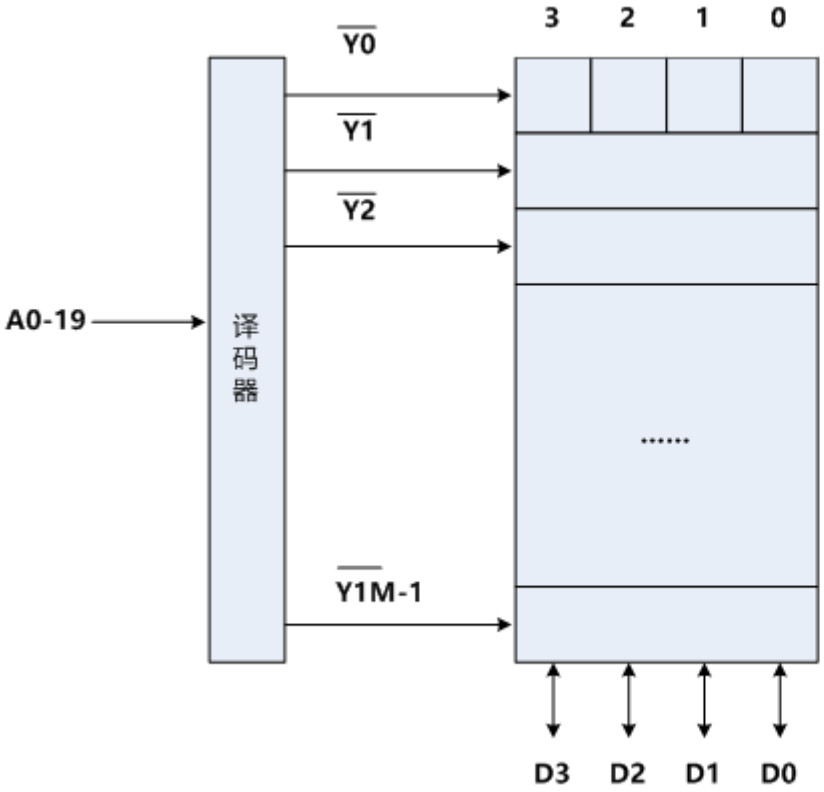


(b) 逻辑结构图

存储容量：1M*4位，1M个存储单元，每个存储单元4个二进制位。

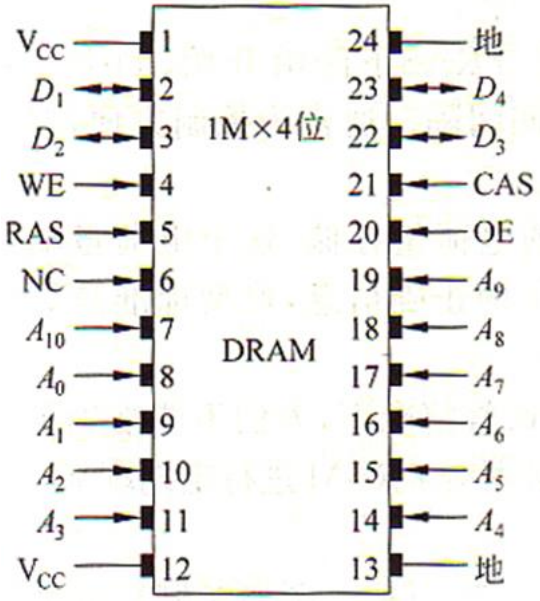
存储元阵列：4个1024*1024存储元阵列面，每个面相同点元同时输入/输出。

3.3.2 DRAM芯片的逻辑结构

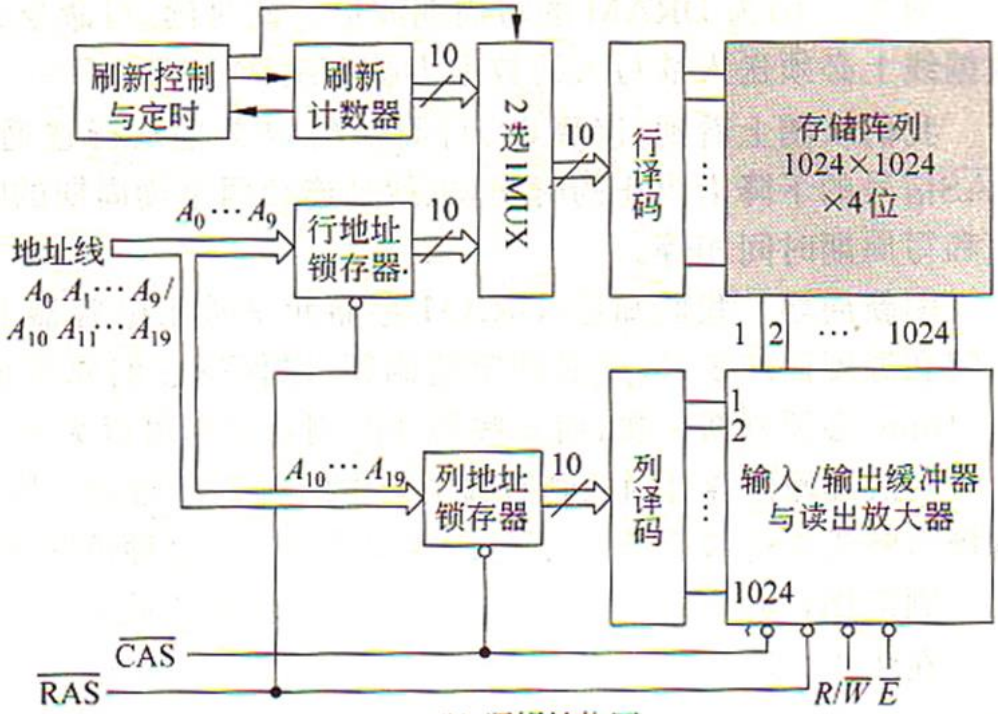


唯一物理地址	
0000 0000 0000 0000 0000	0
0000 0000 0000 0000 0001	1
0000 0000 0000 0000 0010	2
1111 1111 1111 1111 1111	1M-1

3.3.2 DRAM芯片的逻辑结构



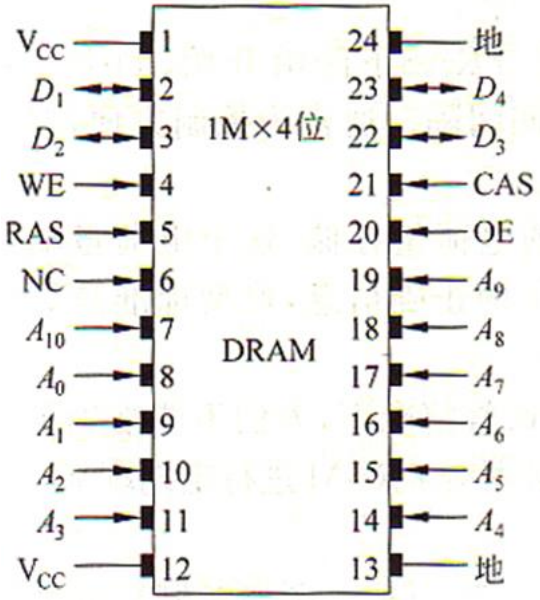
(a) 管脚图



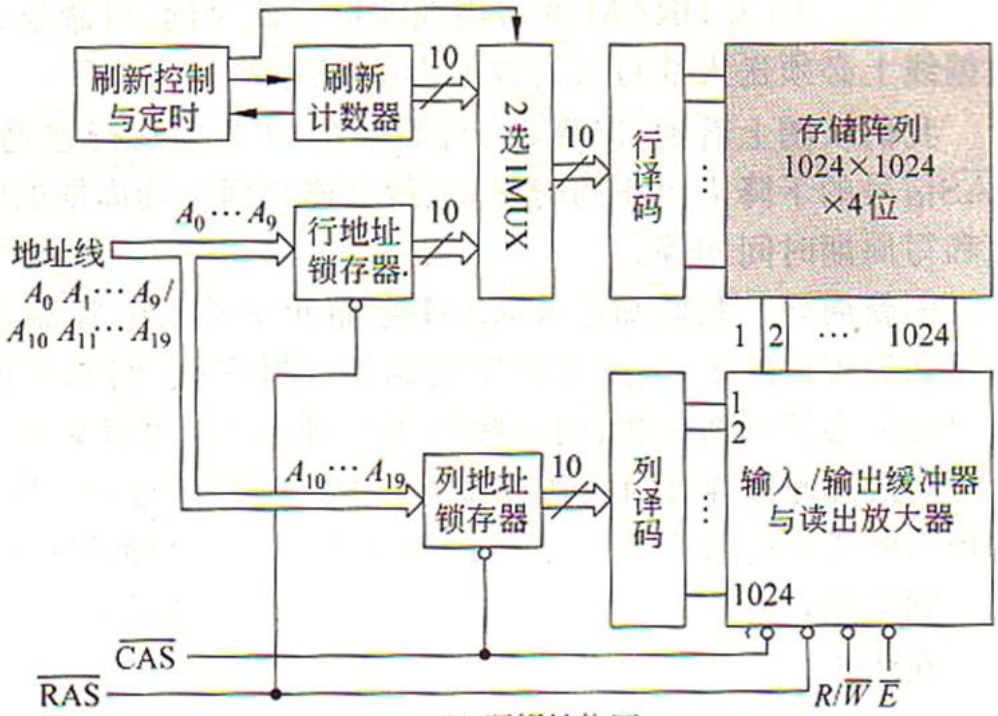
(b) 逻辑结构图

地址线：应该20根，实际上只有10根A0-9，20位地址分2次输入。$\overline{RAS}$有效时，通过A0-9输入10位行地址；$\overline{CAS}$有效时，通过A0-9输入10位列地址。

3.3.2 DRAM芯片的逻辑结构



(a) 管脚图

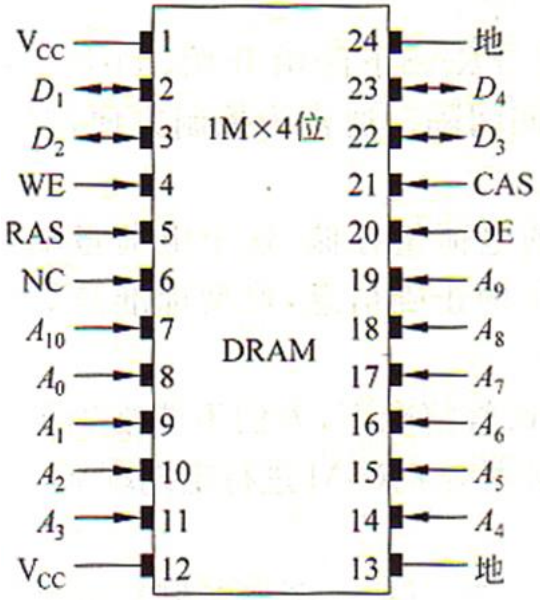


(b) 逻辑结构图

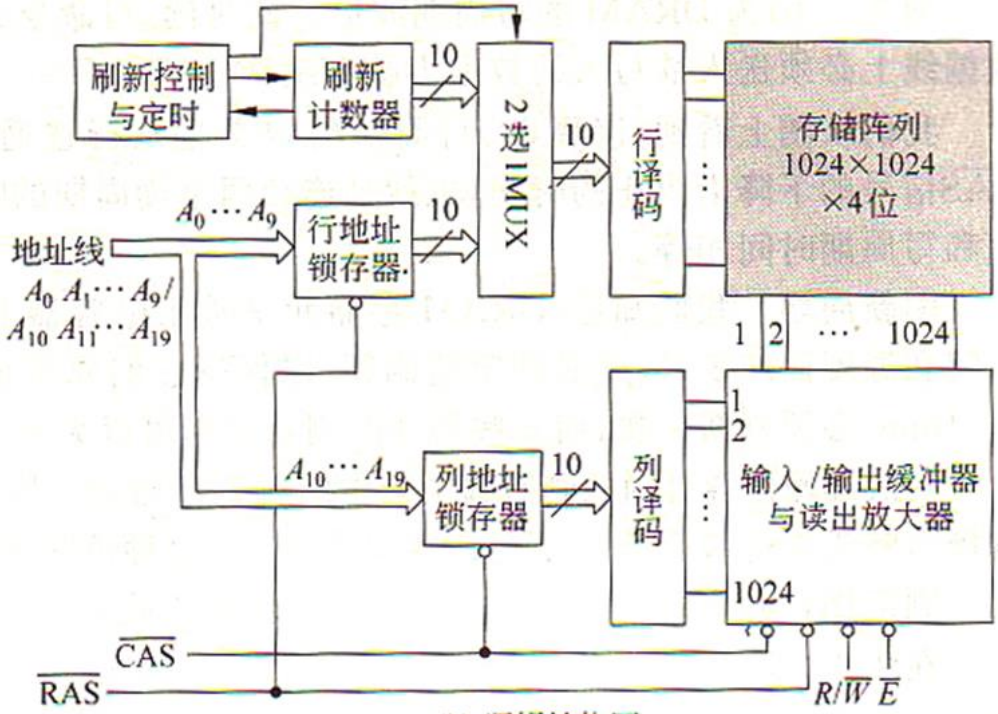
数据线：4根，$D_{1\text{-}4}$

控制线：$\overline{RAS}$、$\overline{CAS}$；WE、OE；

3.3.2 DRAM芯片的逻辑结构



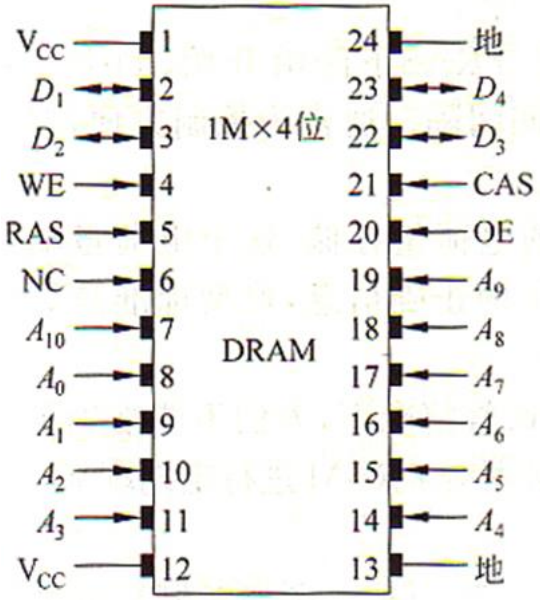
(a) 管脚图



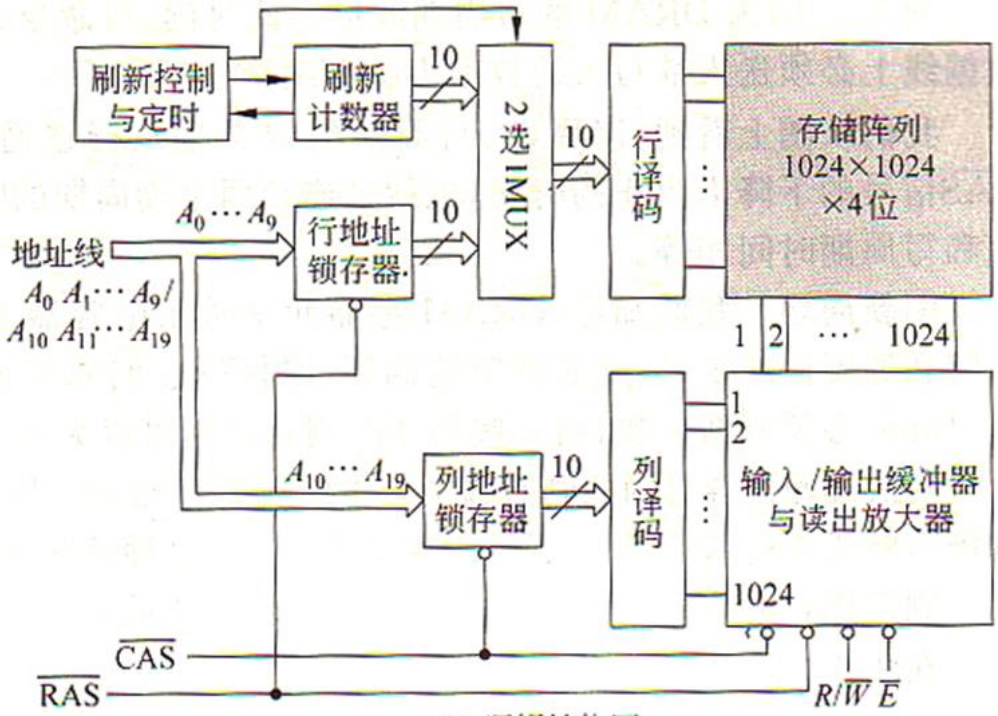
(b) 逻辑结构图

按行刷新：典型是每隔几个ms必须刷新一次，一次刷新一行，即1024*4个存储元。

3.3.2 DRAM芯片的逻辑结构



(a) 管脚图

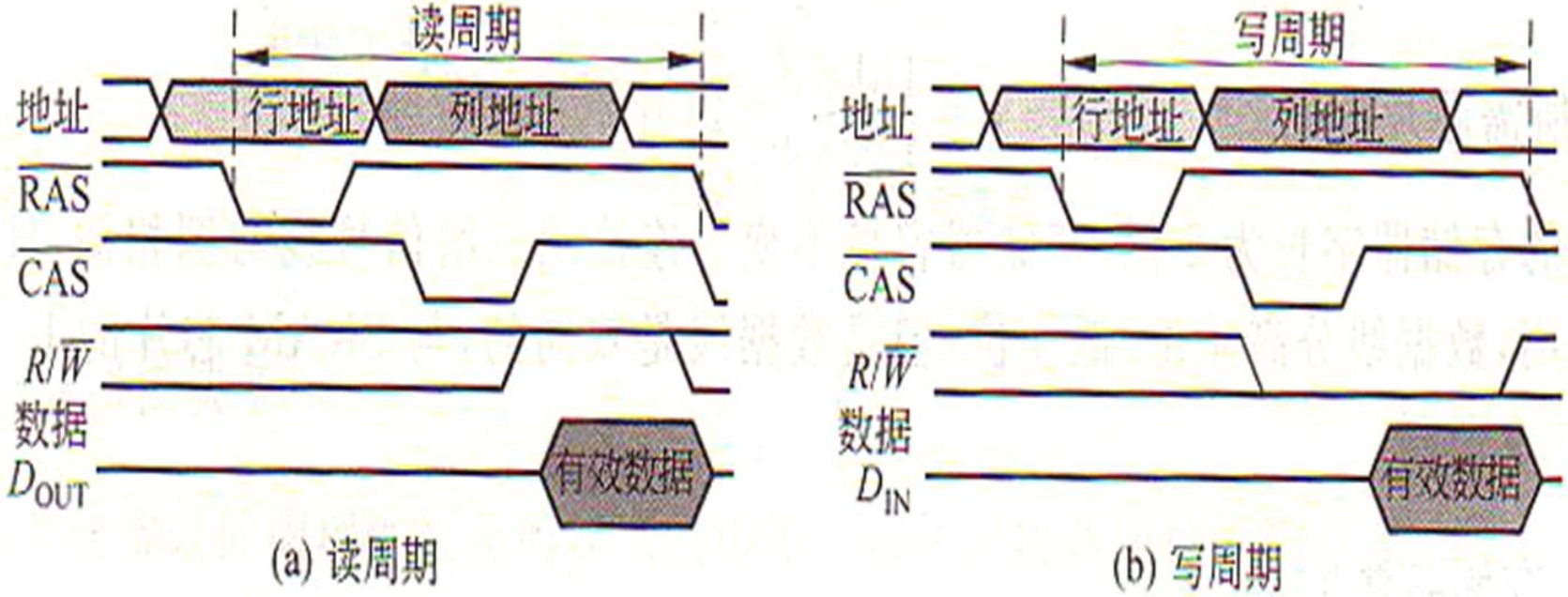


(b) 逻辑结构图

刷新计数器：每刷新一行，自动加1，下一次刷新的行地址。

刷新控制与定时器：刷新周期，即每行的刷新间隔时间。

3.3.3 读写周期



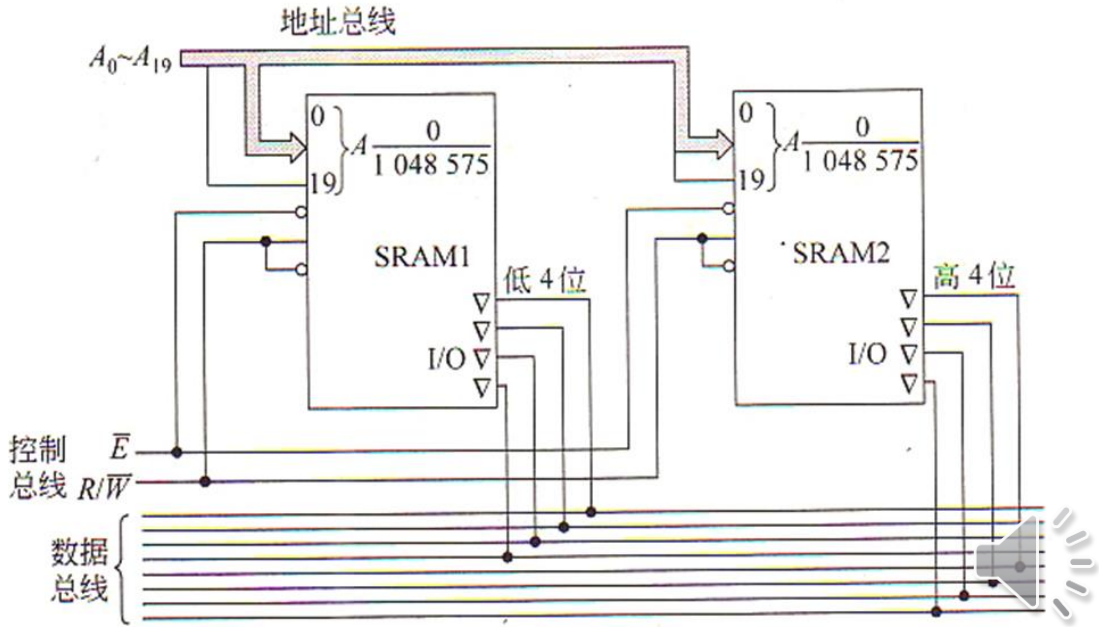
读写周期：注意行地址、列地址分2次输入的控制机制。

3.3.4 存储器容量的扩充 —— 位数扩展

[例] 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。

$\overline{E}$：允许CPU访问内存

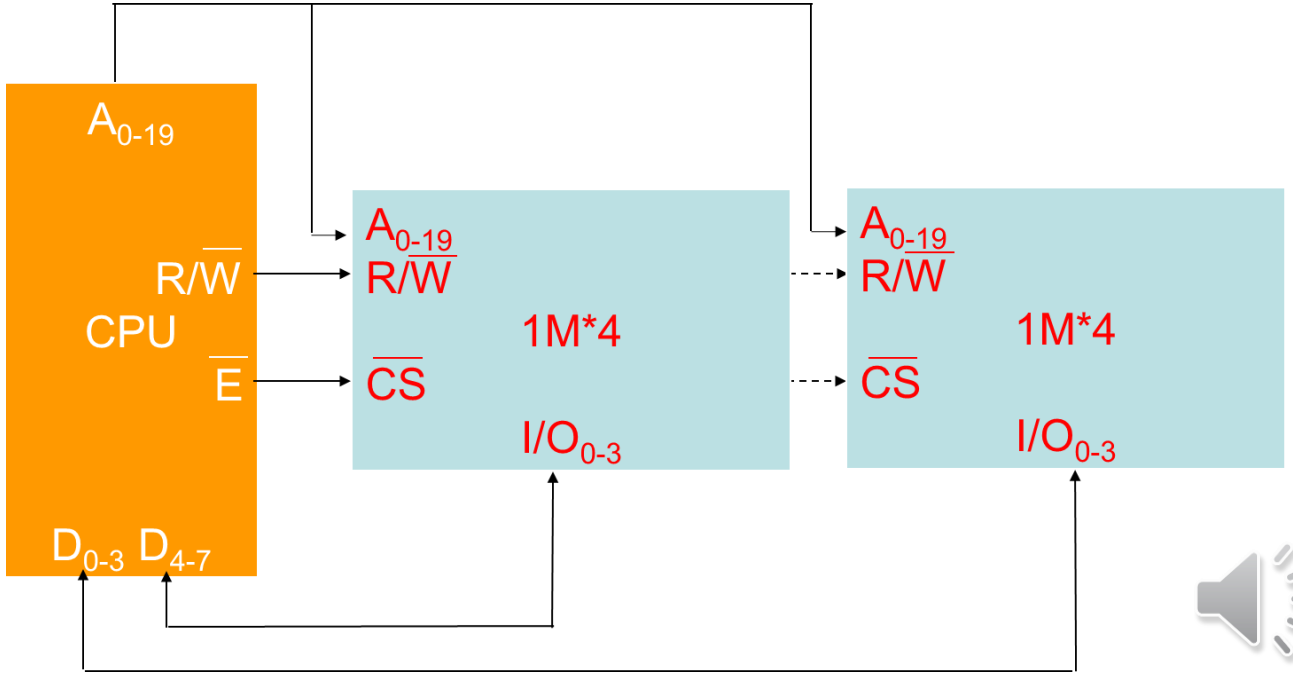
R/$\overline{W}$：读写信号



3.3.4 存储器容量的扩充

—— 位数扩展

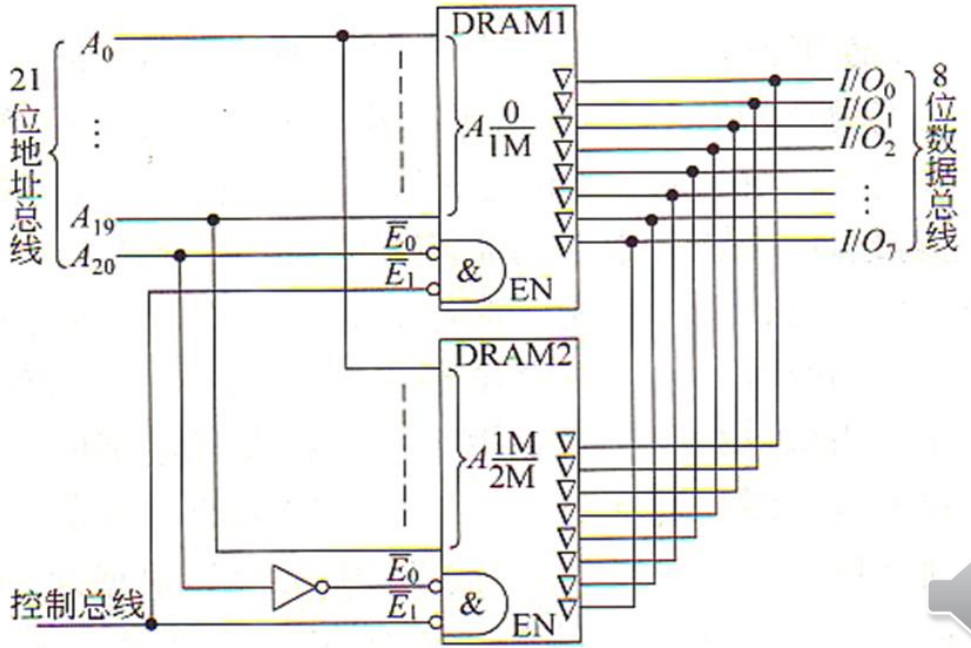
[例] 利用1M*4位的SRAM芯片，设计一个存储容量为1M*8位的SRAM存储器。



3.3.4 存储器容量的扩充

—— 字数扩展

[例] 利用1M*8位的DRAM芯片，设计一个存储容量为2M*8位的SRAM存储器。



地址线A20：接$\overline{E0}$

=0，选择DRAM1片

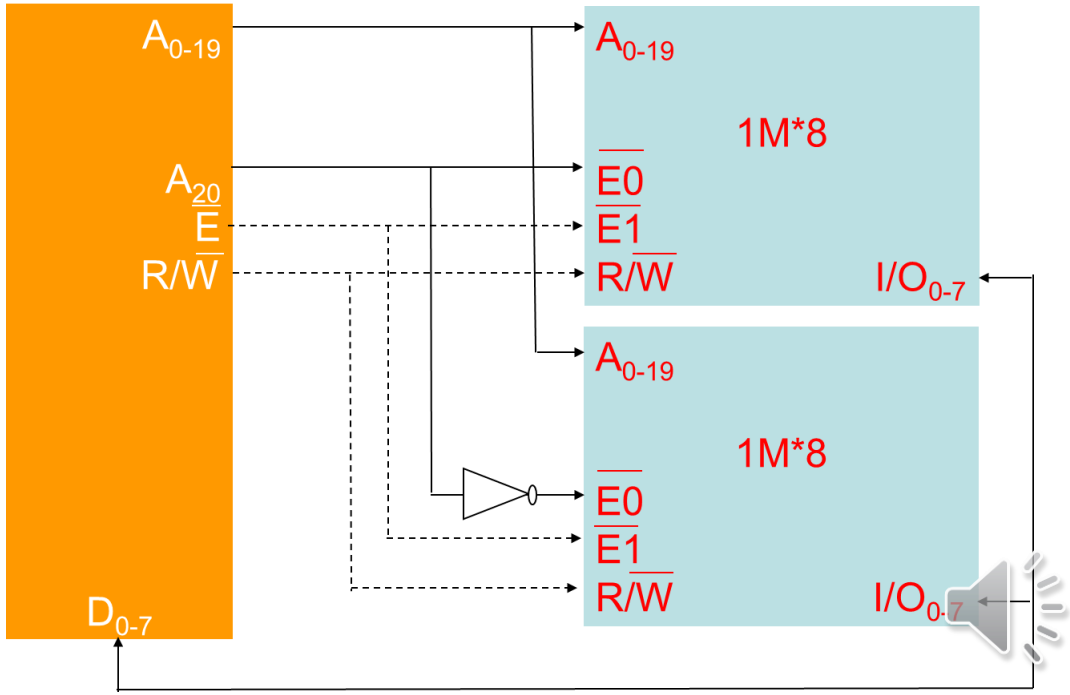
=1，选择DRAM2片

控制线$\overline{E}$：接$\overline{E1}$

3.3.4 存储器容量的扩充

—— 字数扩展

[例] 利用1M*8位的DRAM芯片，设计一个存储容量为2M*8位的SRAM存储器。



3.3.4 存储器容量的扩充

[例] 主存储器逻辑设计。

设CPU的地址总线16根(A0-15，A0为低位)，双向数据总线16根(D0-15)，控制总线中与主存有关的信号有$\overline{MREQ}$(允许访存，低电平有效)，R/$\overline{W}$(高电平为读命令，低电平为写命令)。

主存地址空间分配如下（下述地址为十进制数）：

0—8191为系统程序区（8K），由只读存储芯片组成;

8192—32767为用户程序区（24K）；

63487-65535为系统程序工作区（2K）。

3.3.4 存储器容量的扩充

[例] 主存储器逻辑设计。

现有如下存储器芯片：

EPROM：8K×8位(控制端仅有$\overline{CS}$);

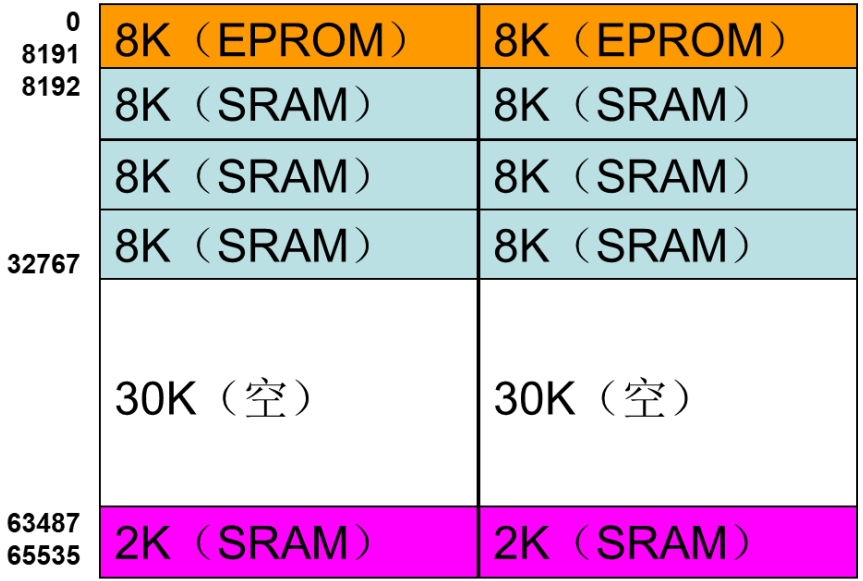
SRAM：2K×8位，8K×8位（控制端有$\overline{CS}$、R/$\overline{W}$）。

请从上述芯片中选择适当芯片设计该计算机主存储器，画出主存储器逻辑框图，注意画出选片逻辑(可选用门电路及3：8译码器74LS138)与CPU 的连接，说明选哪些芯片，选多少片。

3.3.4 存储器容量的扩充

[例] 主存储器逻辑设计方法

第一步：根据设计容量、提供的芯片容量构建地址空间分布图（类似搭积木），可能需要字数扩展、位数扩展。



芯片选择：

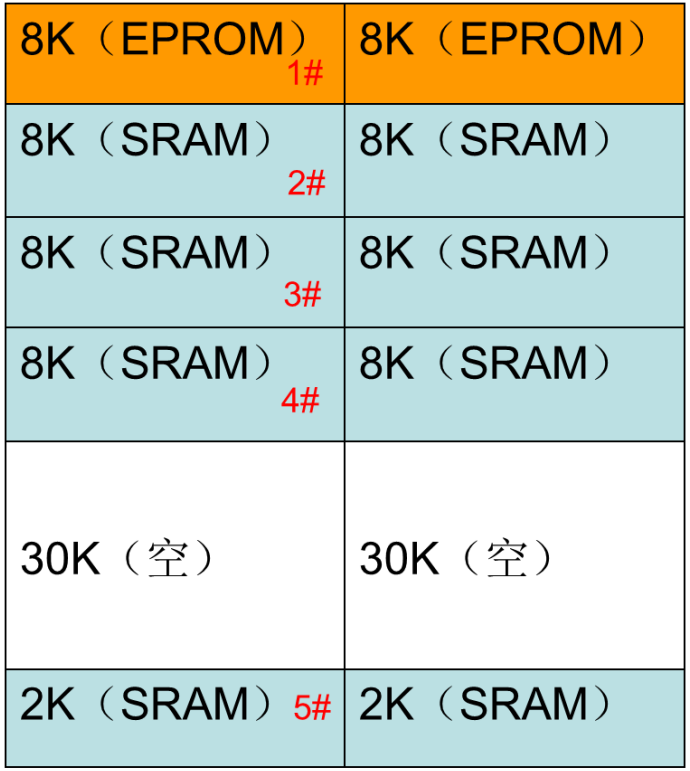
2片8K*8位EPROM芯片；

6片8K*8位SRAM芯片：

2片2K*8位SRAM芯片。

3.3.4 存储器容量的扩充

第二步：用二进制写出连续的地址空间范围



0000 0000 0000 0000 0000H
0001 1111 1111 1111 1FFFH
0010 0000 0000 0000 2000H
0011 1111 1111 1111 3FFFH
0100 0000 0000 0000 4000H
0101 1111 1111 1111 5FFFH
0110 0000 0000 0000 6000H
0111 1111 1111 1111 7FFFH
1000 0000 0000 0000 8000H

1111 0111 1111 1111 F7FFH
1111 1000 0000 0000 F800H
1111 1111 1111 1111 FFFFH

3.3.4 存储器容量的扩充

第三步：写出各片组的片选逻辑表达式。

从二进制地址空间范围可以明显看出，采用A15、A14、A13三根地址线译码（74LS138），可以从地址上区分出各片组。

1#：$\overline{CS1}=\overline{A15}\,\overline{A14}\,\overline{A13}=000=\overline{Y0}$

2#：$\overline{CS2}=\overline{A15}\,\overline{A14}A13=001=\overline{Y1}$

3#：$\overline{CS3}=\overline{A15}A14\overline{A13}=010=\overline{Y2}$

4#：$\overline{CS4}=\overline{A15}A14A13=011=\overline{Y3}$

5#：$\overline{CS5}=A15A14A13A12A11=111\ 11=\overline{Y7}\ 11$

3.3.4 存储器容量的扩充

第四步：按三总线分析CPU和选用存储器芯片的数据线、地址线、控制线，以便设计CPU与存储器的连接。

8K×8位 EPROM（#1）

数据线$D^{0\text{-}7}$，地址线$A_{0\text{-}12}$，片选$\overline{CS}$

8K×8位 SRAM（#2、#3、#4）

数据线$D_{0\text{-}7}$，地址线$A_{0\text{-}12}$，片选$\overline{CS}$、读写$R/\overline{W}$

2K×8位 SRAM（#5）

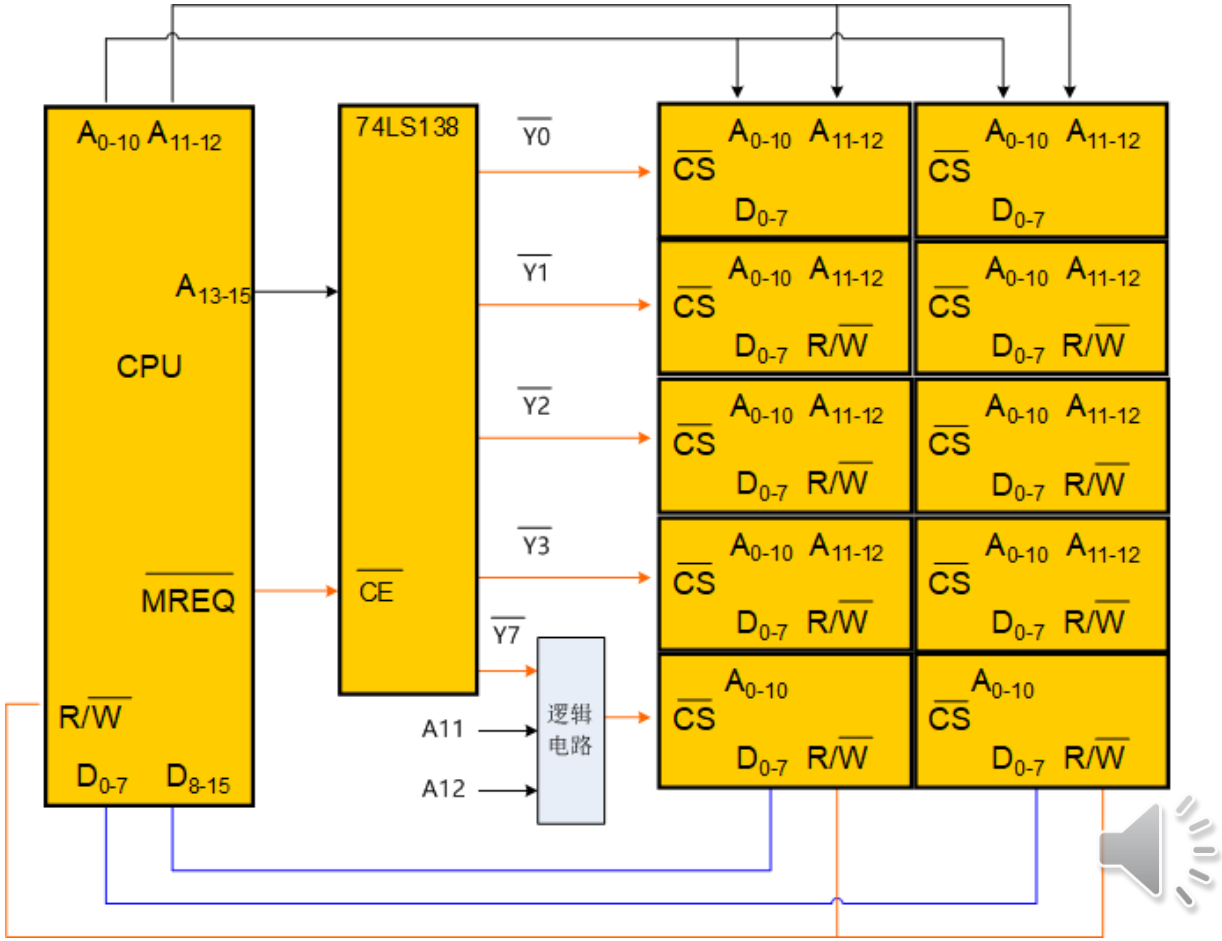
数据线$D_{0\text{-}7}$，地址线$A_{0\text{-}10}$，片选$\overline{CS}$、读写$R/\overline{W}$

CPU

数据线$D_{0\text{-}15}$，地址线$A_{0\text{-}15}$ ，访问存储器$\overline{MREQ}$、读写信号线$R/\overline{W}$

3.3.4 存储器容量的扩充

第五步：设计CPU与存储器连接的逻辑结构图。



3.4 只读存储器ROM

- 一次性掩膜ROM
- EPROM存储元

3.4 只读存储器ROM

—— 特点

正常工作情况下，只能读、不能写，读出的是事先存入的确定信息；

通过特定方式擦除，然后写入信息；

只读存储器由于工作可靠，保密性强，在计算机系统中得到广泛应用；

技术和工艺的改进：ROM → PROM → EPROM → E^2PROM → KEPROM →闪速存储器。

3.4 只读存储器ROM —— 一次性掩膜ROM

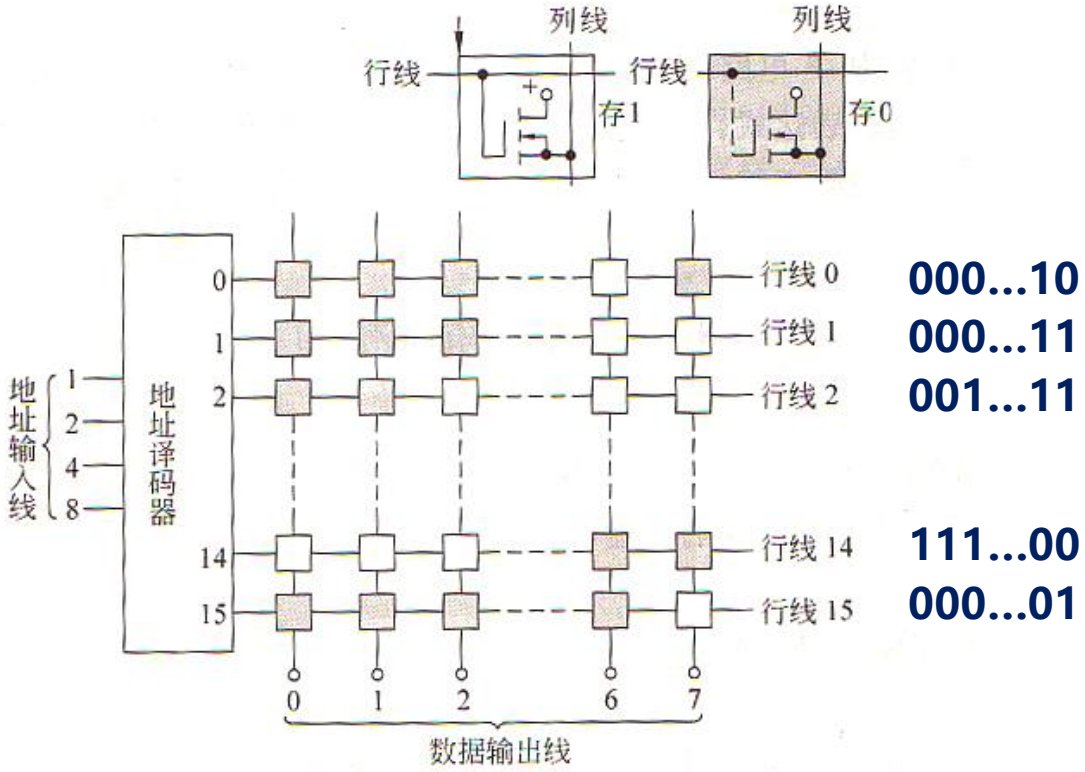


图 3.17　16×8 位 ROM 阵列结构示意图

①出厂前一次性烧成；

②一旦烧成，不能重写。

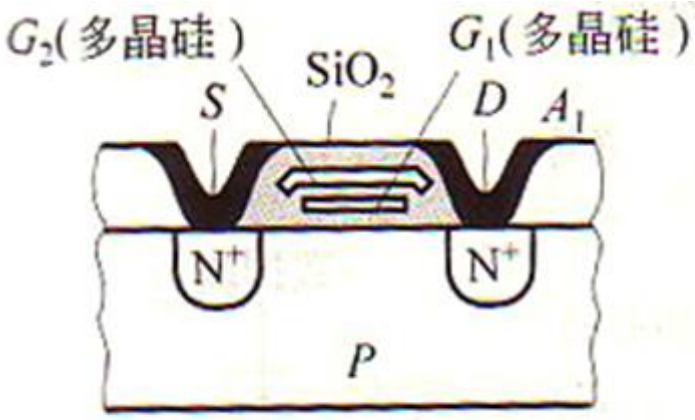
3.4 只读存储器ROM —— 可编程ROM

PROM：一次编程ROM。出厂时全“0”或全“1”，可写一次，不能擦除重写。

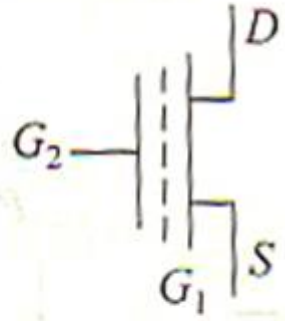
EPROM：光擦除可编程ROM，可多次擦除重写。

E^2PROM：电擦除可编程ROM，可多次擦除重写。

3.4 只读存储器ROM —— EPROM存储元



浮栅雪崩注入型 MOS 管结构



逻辑符号

D：漏极

S：源极

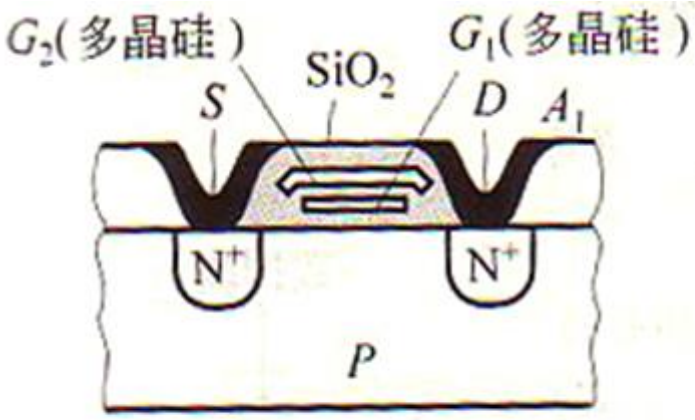
G1、G2：栅极

写“0”：在D端加上几十伏的脉冲电压，使G1上累积电荷，由于G1周围都是绝缘的二氧化硅层，泄漏电流极小，能长期保存累积电荷。

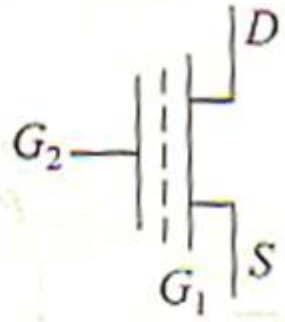
出厂时为全“1”状态：G1上无累积电荷。

3.4 只读存储器ROM

—— EPROM存储元



浮栅雪崩注入型 MOS 管结构



逻辑符号

D：漏极

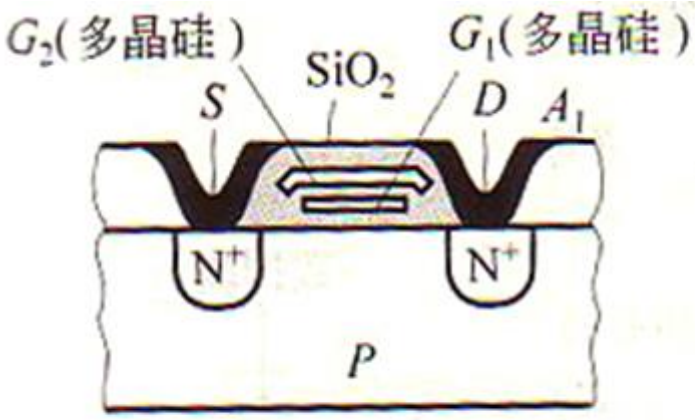
S：源极

G1、G2：栅极

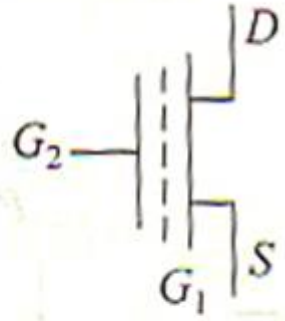
读“1”：G1上无累积电荷，在正常工作状态下，G2加上高电平，MOS管导通，表示读出“1”；

3.4 只读存储器ROM

—— EPROM存储元



浮栅雪崩注入型 MOS 管结构



逻辑符号

D：漏极

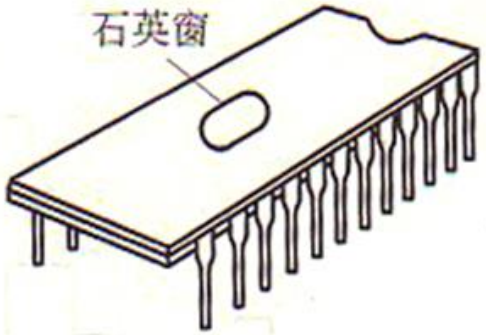
S：源极

G1、G2：栅极

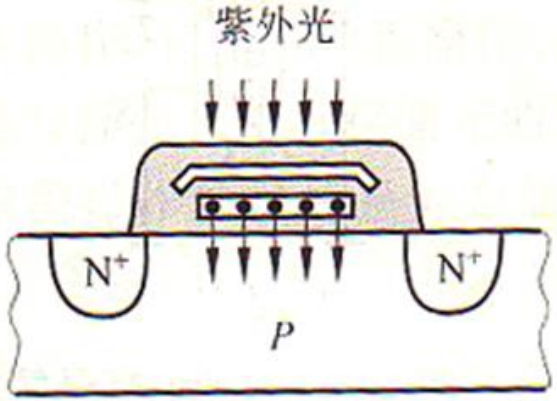
读“0”： G1上有累积电荷，在正常工作状态下，G2加上高电平，达不到MOS管的开启电压，MOS管截止，表示读出“0”。

3.4 只读存储器ROM

—— EPROM存储元



存储器外形图



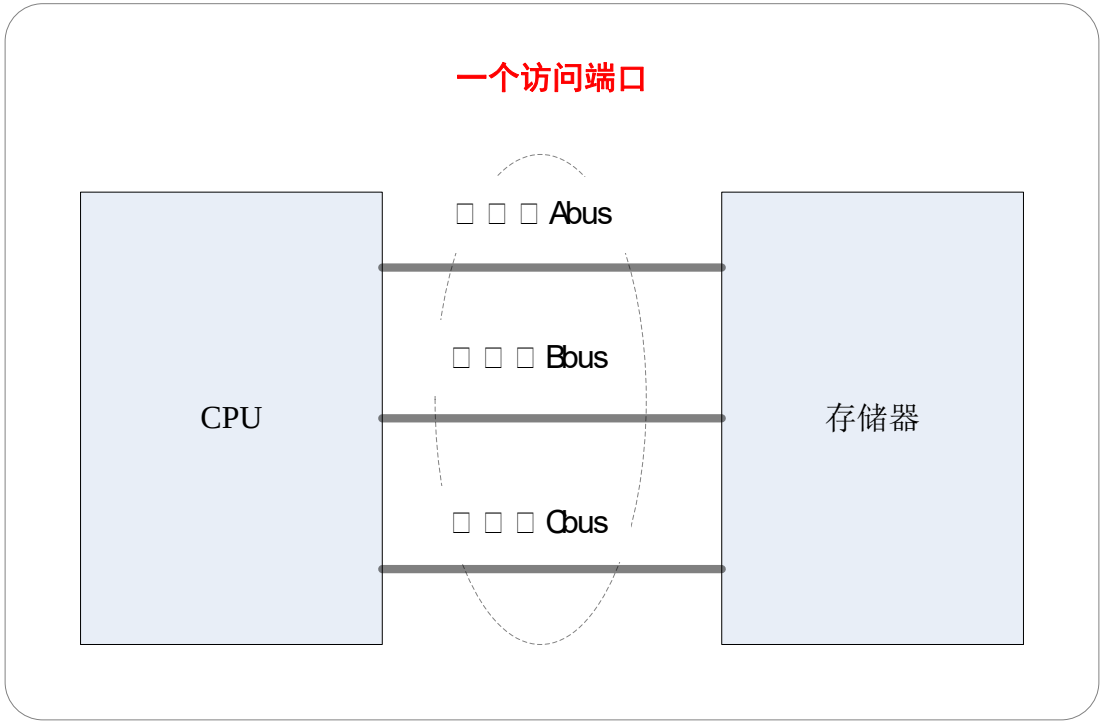
光抹成全“1”

擦除： 紫外光通过石英窗口照射，G1上的电荷获得足够能量，穿过氧化层回到衬底，从而使G1上无累积电荷，恢复为全“1”状态，然后可以进行重写。

3.5 并行存储器

- 双端口存储器
- 多模块交叉存储器

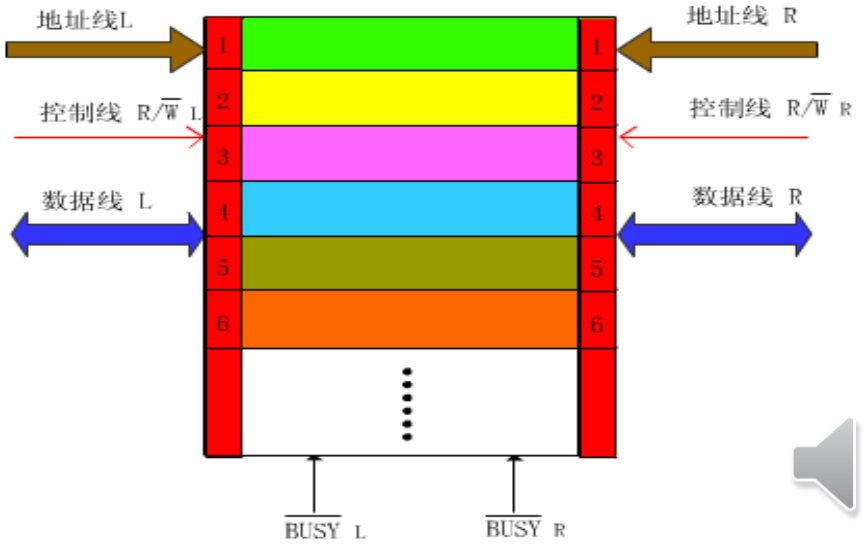
3.5.1 双端口存储器



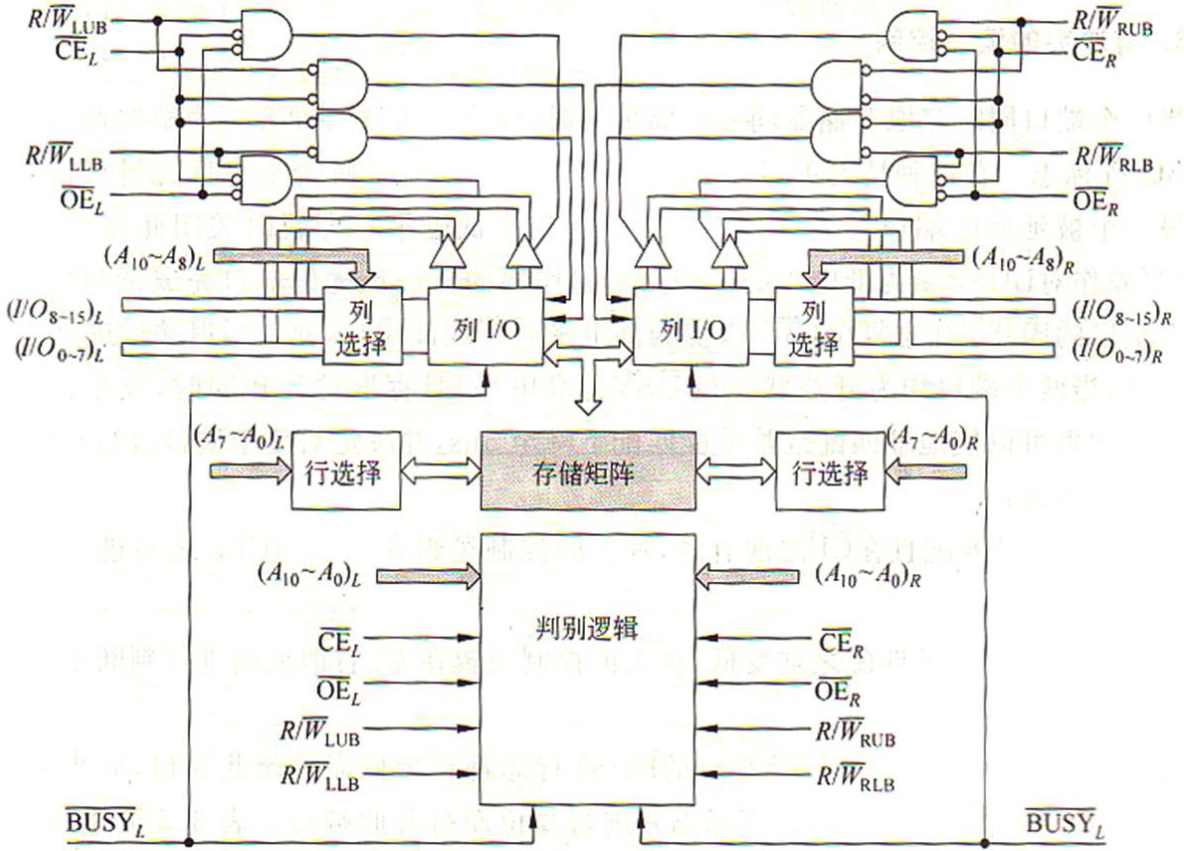
存储器访问端口的三总线结构：地址总线、控制总线、数据总线

3.5.1 双端口存储器

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线路（即两个相互独立的访问端口），它们分别具有各自的地址线、数据线和控制线，可以进行并行的独立操作。



3.5.1 双端口存储器


$R/\overline{W}_{LUB}$
$\overline{CE}_L$
$R/\overline{W}_{LLB}$
$\overline{OE}_L$
$(A_{10}\sim A_8)_L$
$(I/O_{8\sim15})_L$
$(I/O_{0\sim7})_L$
列选择
列 I/O
列 I/O
列选择
$R/\overline{W}_{RUB}$
$\overline{CE}_R$
$R/\overline{W}_{RLB}$
$\overline{OE}_R$
$(A_{10}\sim A_8)_R$
$(I/O_{8\sim15})_R$
$(I/O_{0\sim7})_R$
$(A_7\sim A_0)_L$
行选择
存储矩阵
行选择
$(A_7\sim A_0)_R$
$(A_{10}\sim A_0)_L$
$\overline{CE}_L$
$\overline{OE}_L$
$R/\overline{W}_{LUB}$
$R/\overline{W}_{LLB}$
判别逻辑
$(A_{10}\sim A_0)_R$
$\overline{CE}_R$
$\overline{OE}_R$
$R/\overline{W}_{RUB}$
$R/\overline{W}_{RLB}$
$\overline{BUSY}_L$
$\overline{BUSY}_L$

3.5.1 双端口存储器

无冲突读写控制：

当只有任意一个端口访问存储器时，当然它可以对整个存储器的任何单元进行存取，而且不存在冲突的问题。

当两个端口同时访问存储器，而访问地址不相同（不同存储单元）时，在两个端口上可以同时各自独立地并行进行读写操作，不会发生冲突。

3.5.1 双端口存储器

有冲突读写控制：

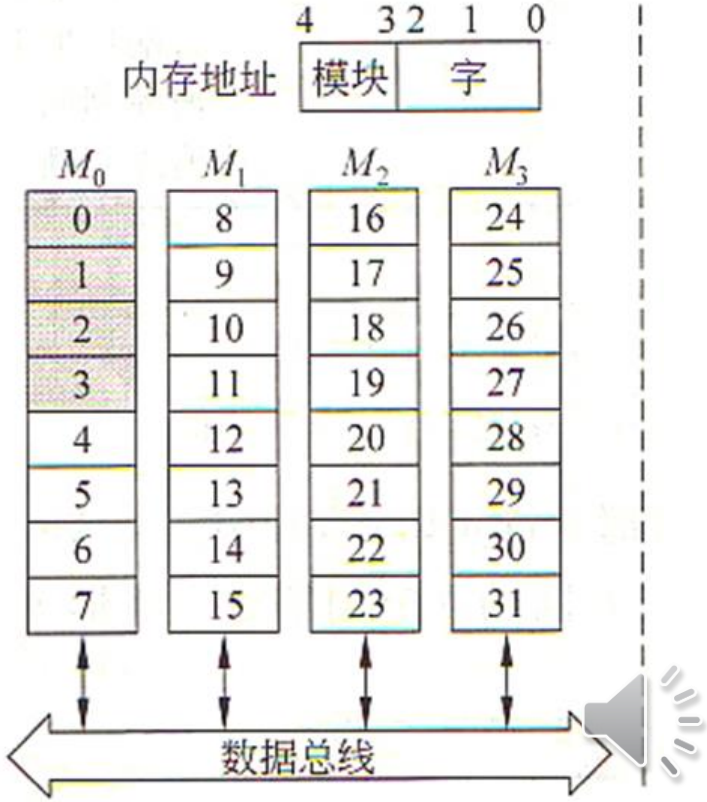
当两个端口同时存取（访问）存储器同一存储单元时，便发生读写冲突。

为解决此问题，特设置了$\overline{\text{BUSY}}$标志。

在这种情况下，片上的判断逻辑可以决定哪个端口优先进行读写操作，而对另一个被延迟的端口置$\overline{\text{BUSY}}$标志。

3.5.2 多模块交叉存储器 —— 存储器的模块化组织

顺序方式：各存储体依次顺序定义地址空间，即字扩展方式，每个存储体中的地址是连续的。



地址范围：

M0：00 000-00 111

M1：01 000-01 111

M2：10 000 10 111

M3：11 000-11 111

3.5.2 多模块交叉存储器 —— 存储器的模块化组织

顺序方式：各存储体依次顺序定义地址空间，即字扩展方式，每个存储体中的地址是连续的。

特点：

(1) 某模块进行存取时，其他模块不工作。

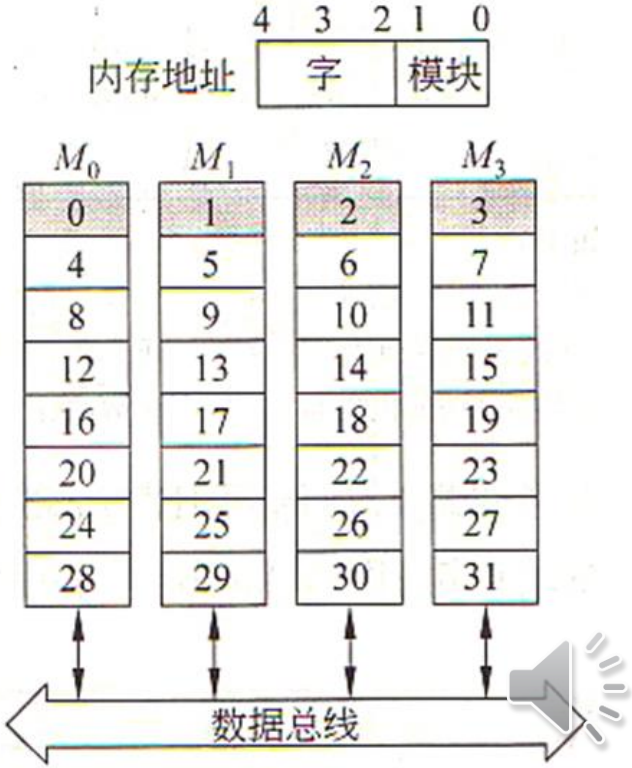
(2) 某模块出现故障时，其他模块可以照常工作。

(3) 增添模块扩充容量比较方便。

(4) 各模块串行工作，无法采用流水线技术。

3.5.2 多模块交叉存储器 —— 存储器的模块化组织

交叉方式：连续地址交叉分配在各个存储体中，每个存储体中的地址是不连续的。



地址范围：

M0：000 00-111 00

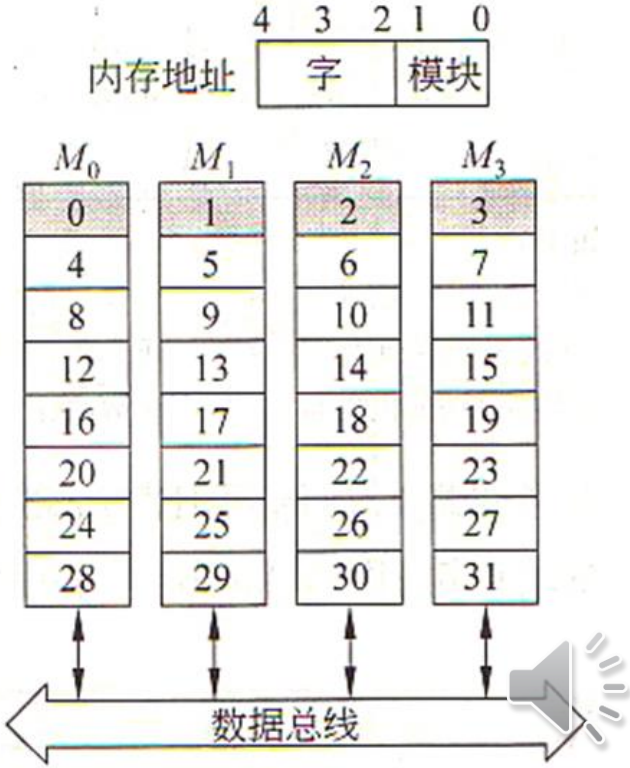
M1：000 01-111 01

M2：000 10-111 10

M3：000 11-111 11

3.5.2 多模块交叉存储器 —— 存储器的模块化组织

交叉方式：连续地址交叉分配在各个存储体中，每个存储体中的地址是不连续的。



特点：

（1）同一个模块内的地址都是不连续的。

（2）对连续的存储器访问可实现流水线并行存取提高存储器的带宽。

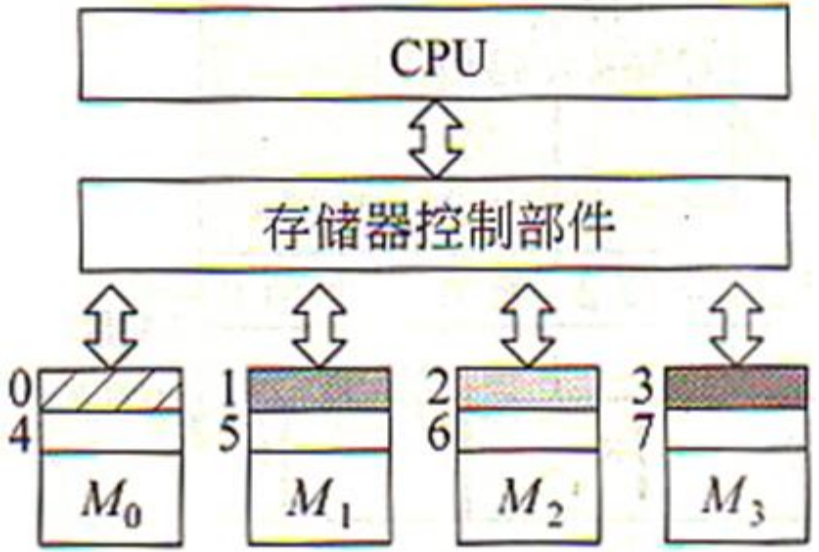
3.5.2 多模块交叉存储器 —— 多模块交叉存储器结构

（1）设模块存取一个存储单元的时间为T；

（2）总线传递一个数据到CPU，然后到运算器/控制器的时间为τ；

（3）流水线方式存取：M0取出一个数据D0，然后经过总线传递；当D0总线传递完成，**马上**就有M1中的一个数据D1，送给总线传递；依此类推……，充分利用总线，只要总线传递不重叠即可。

（4）为了充分利用总线，由于T> τ，必然有四个存储体存取的重叠操作，即交叉存储器的流水线方式存取。



四模块交叉存储器结构框图

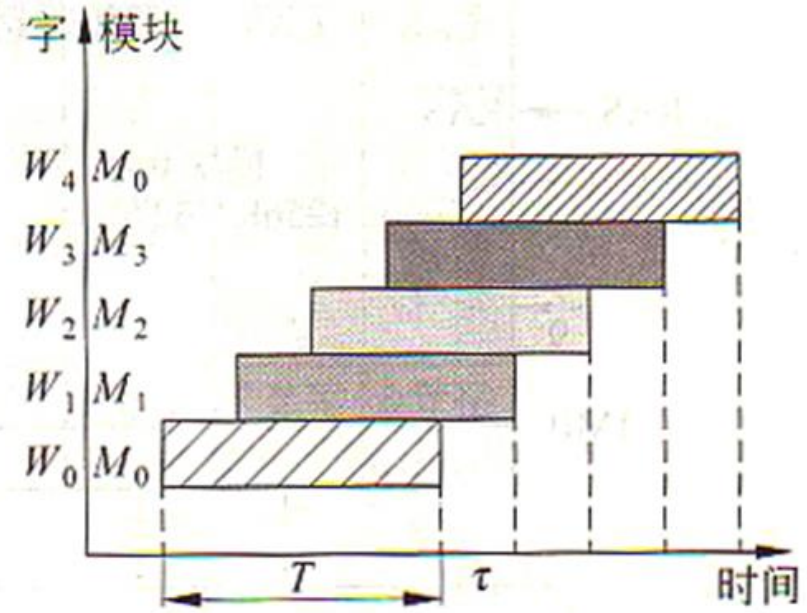
3.5.2 多模块交叉存储器 —— 多模块交叉存储器结构

（1）流水线的时钟周期为τ；

（2）采用交叉流水线完成n个任务所需要的时间为（T+τ）+(n-1)τ，为了和流水线时间公式形式上一致，为T+(n-1)τ；

（3）采用非流水线完成n个任务所需要的时间为n（T+τ），可约为nT；

（4）T=4 τ。



流水线方式存取示意图

3.5.2 多模块交叉存储器 —— 多模块交叉存储器结构

【例5】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τ=50ns。问顺序存储器和交叉存储器的带宽各是多少?

【解】顺序存储器和交叉存储器连续读出m=4个字的信息总量都是：

q=64位×4=256位。

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：

$t2=mT=4\times200ns=800ns=8\times10^{-7}s$

$t1=T+(m-1)\tau=200ns+3\times50ns=350ns=3.5\times10^{-7}s$

顺序存储器和交叉存储器的带宽分别是：

$W2=q/t2=256\div(8\times10^{-7})=320M\ bps$

$W1=q/t1=256\div(3.5\times10^{-7})=730M\ bps$

3.6 cache存储器

- Cache基本原理
- 主存与cache的地址映射
- 替换策略
- Cache的写操作策略
- Pentium 4 的cache组织

3.6.1 cache基本原理

—— cache的功能

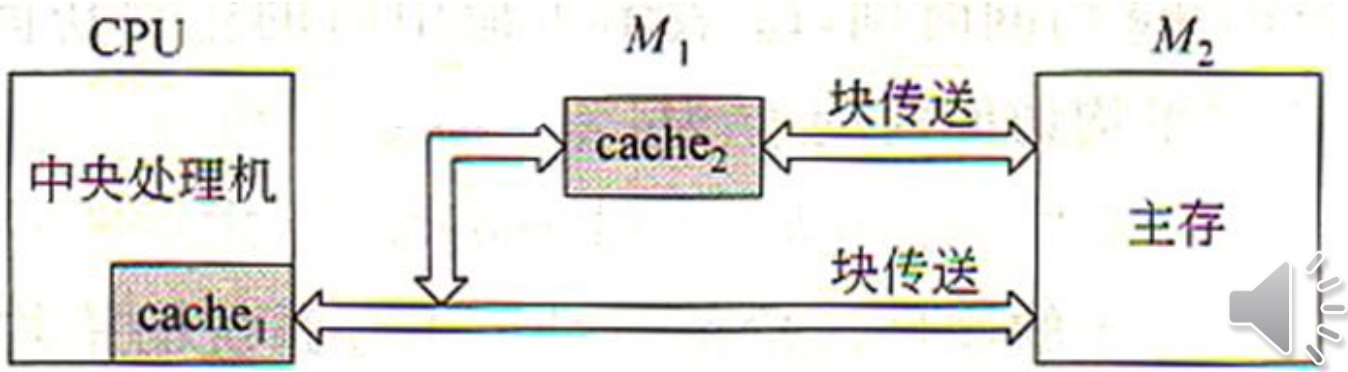
cache高速缓冲存储器，是为了解决CPU和主存之间速度不匹配而采用的一项重要技术。

（1）可高速存取的小容量的存储器，片内cache已经接近于CPU处理速度；

（2）CPU可直接访问cache；

（3）可构造2级以上的cache系统；

（4）cache和主存、cache和cache之间的块传送，对用户是透明的。



3.6.1 cache基本原理

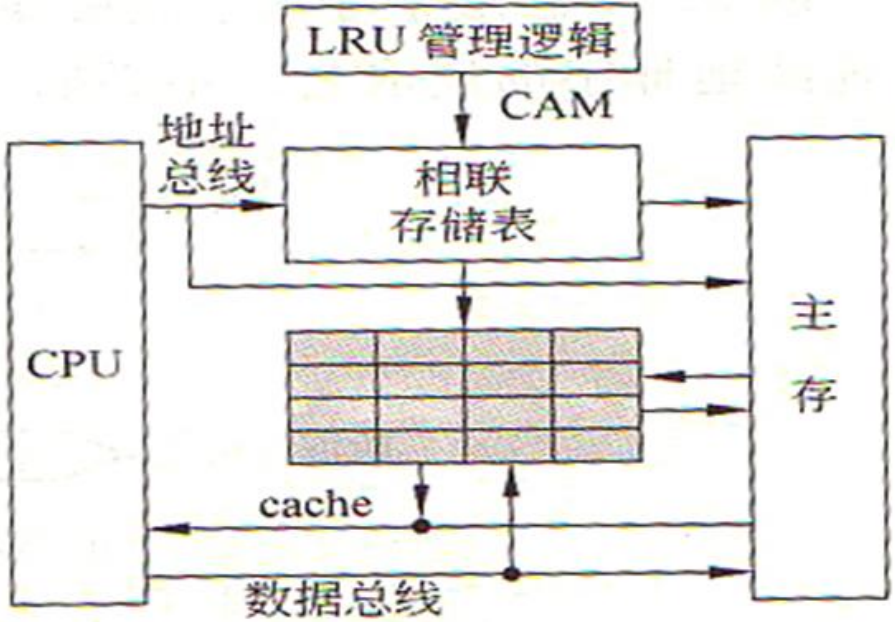
—— 存储器分块

块地址（块编号、11位）	块内地址（偏移地址、9位）
0000 0000 000	0 0000 0000
0000 0000 000	1 1111 1111
0000 0000 001	0 0000 0000
0000 0000 001	1 1111 1111
1111 1111 111	0 0000 0000
1111 1111 111	1 1111 1111

3.6.1 cache基本原理

（1）主存和cache均按照约定长度划分为若干块，存储单元物理地址=块地址（块编号）+ 块内地址；

（2）主存中一个存储块调入（交换）到cache中，则将存储块地址（块编号）存放到相联存储表CAM中，将数据块内容存放在cache中；

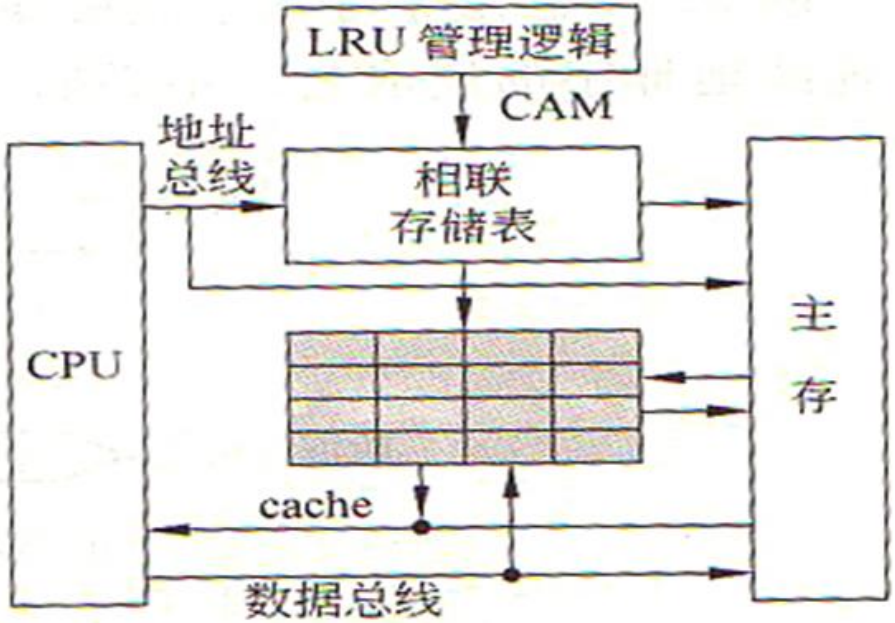


3.6.1 cache基本原理

（3）当CPU访问主存时输出物理地址，然后根据物理地址的高位部分（块地址、块编号），判断是否在相联存储器CAM中?

若在，则命中，直接访问cache；

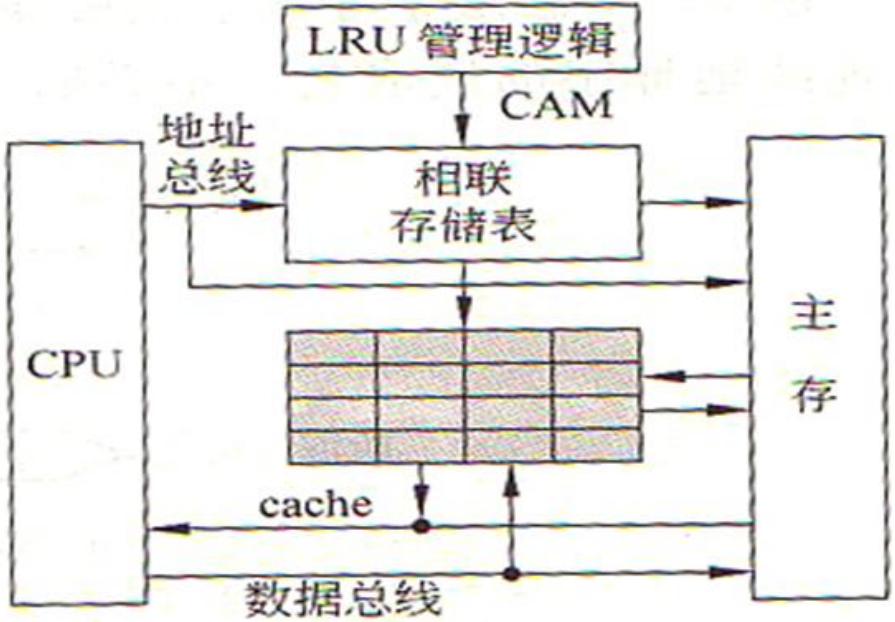
若不在，则未命中，直接访问主存，并将该单元所在存储块交换到cache中，后续对该存储块的访问则在cache命中。



3.6.1 cache基本原理

（4）通过cache和主存之间的存储块动态交换，尽量争取CPU访存操作在cache命中，从而总体提高访存速度；

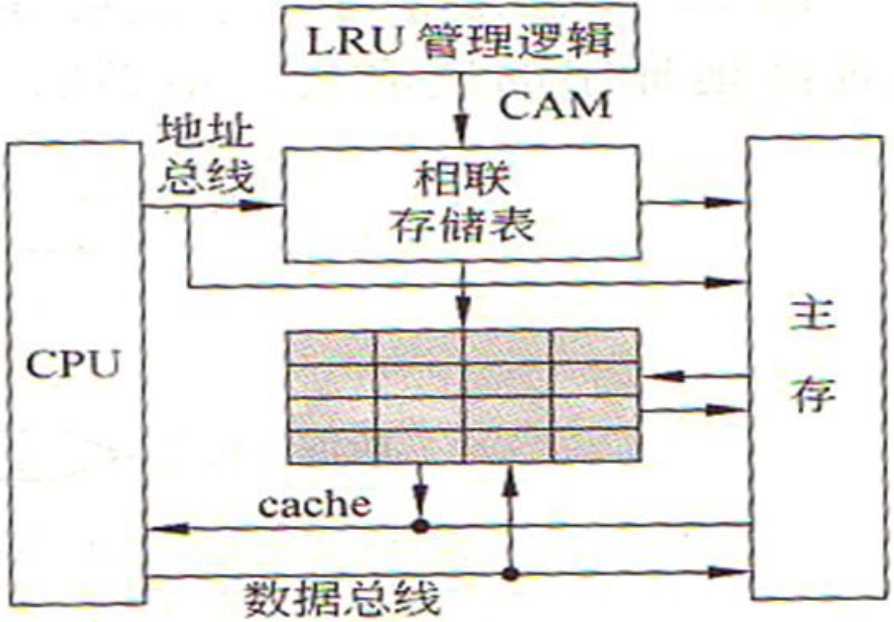
（5）基于程序和数据的局部性访问原理，cache实际上是主存的当前最活跃部分，即主存的一个子集。



3.6.1 cache基本原理

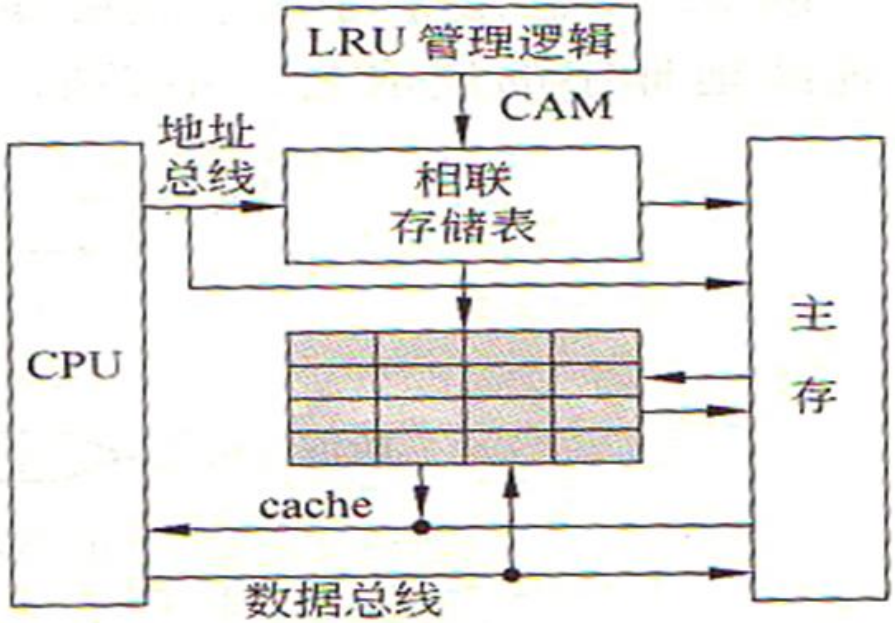
相联存储器：按内容访问；

普通存储器：按地址访问。



3.6.1 cache基本原理

LRU管理逻辑：是一种替换策略，当cache已满、且有新的存储块需要交换到cache时，将cache中最近最少使用的存储块替换出去。



(1) 命中率

增加cache以后，就应该尽量争取在cache中命中越多越好。

在一个程序执行期间，设Nc表示cache命中完成存取的总次数，Nm表示未命中、主存完成存取的总次数，h定义为命中率，则有：

$$h=\frac{Nc}{Nc+Nm}$$

（2） cache/主存系统的平均访问时间

设tc表示命中时的cache访问时间，tm表示未命中时的主存访问时间，1-h表示未命中率，则cache/主存系统的平均访问时间ta为：

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

（3） 访问效率

设r=tm/tc表示主存慢于cache的倍率，e表示访问效率，则有：

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

(4) 命中率的影响因素

程序行为，顺序程序比分支程序命中率高；

cache的容量；

组织方式；

存储块大小。

3.6.1 cache基本原理

—— cache的命中率

【例】CPU执行一段程序时，cache完成存取次数为1900次，主存完成存取次数为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】 h=Nc/(Nc+Nm)=1900/(1900+100)= 0.95

r=tm/tc=250ns/50ns= 5

e=1/(r+(1-r)h)=1/(5+(1-5)×0.95)= 83.3%

ta=tc/e=50ns/0.833=60ns

3.6.2 主存与cache的地址映射

如何把主存块放到cache中合适的位置？如何根据CPU输出的物理地址确定是否在cache命中？如何在cache中定位到访问的存储块和存储单元？必须有一套对应的策略，称为地址映射。

地址映射有全相联映射方式、直接映射方式和组相联映射方式三种。

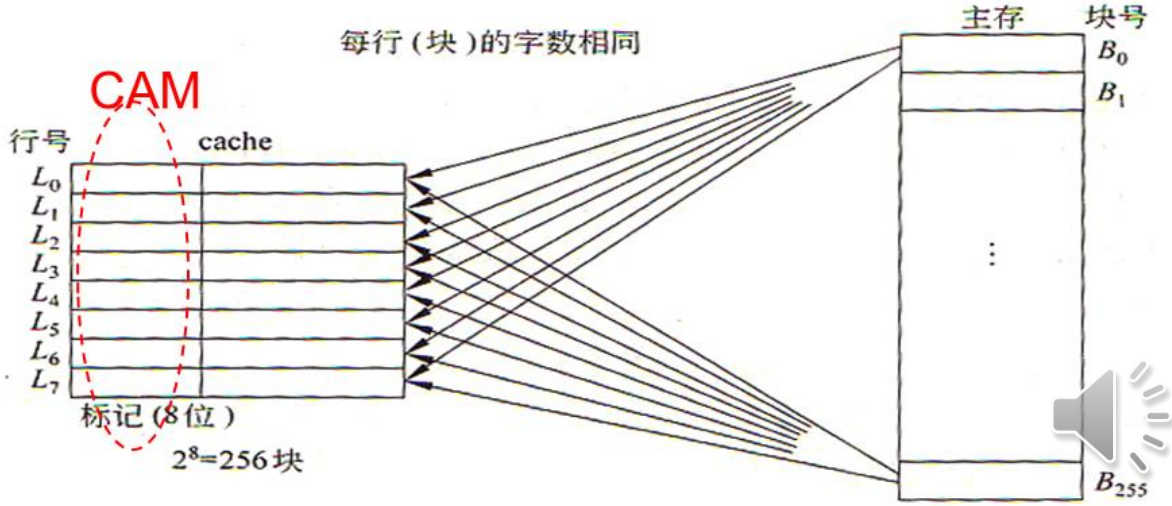
3.6.2 主存与cache的地址映射 —— 全相联映射方式

主存和cache都按照相同的长度划分若干个块（行），当主存中某块（行）交换到cache的一个块（行）中时，将块地址集中存放在相联存储器CAM中。

（1）主存256块，cache8块；

（2）设块的长度为128B，则主存物理地址为：块地址8位 + 块内地址7位；

（3）主存中的任意一块（行）可以载入到cache中任意一块（行）位置上。



3.6.2 主存与cache的地址映射 —— 全相联映射方式

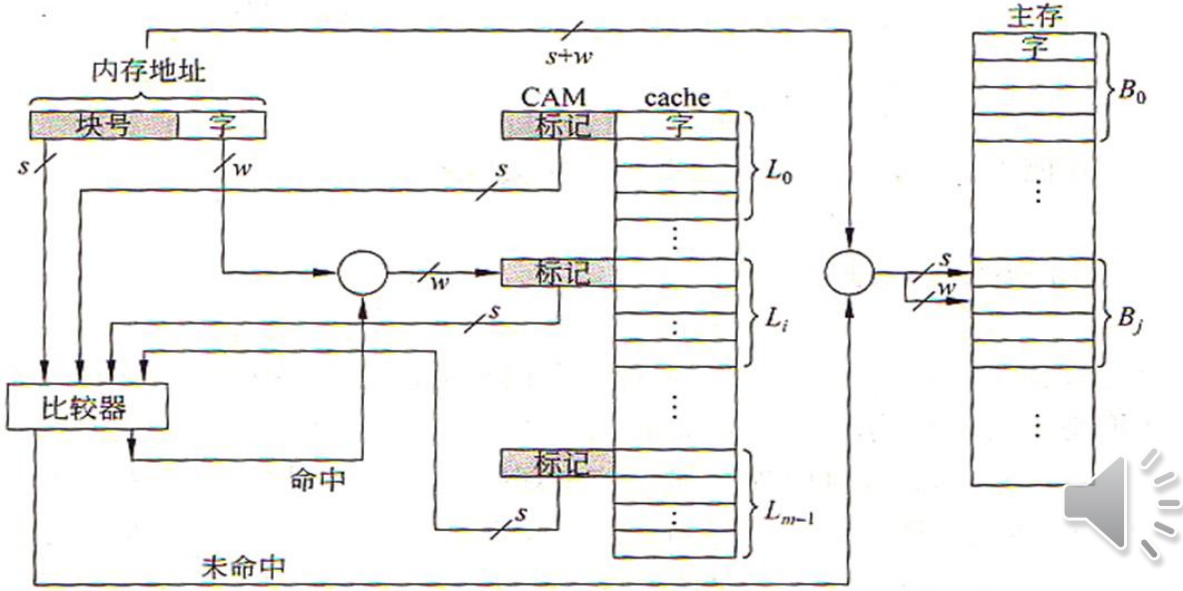
根据物理地址中的块地址（块号）s和CAM中已存放的块地址（标记）进行比较：

若命中，则根据块内地址（字）w访问cache指定单元；

若未命中，则根据s+w访问主存单元，并将该块载入cache中。

特点：

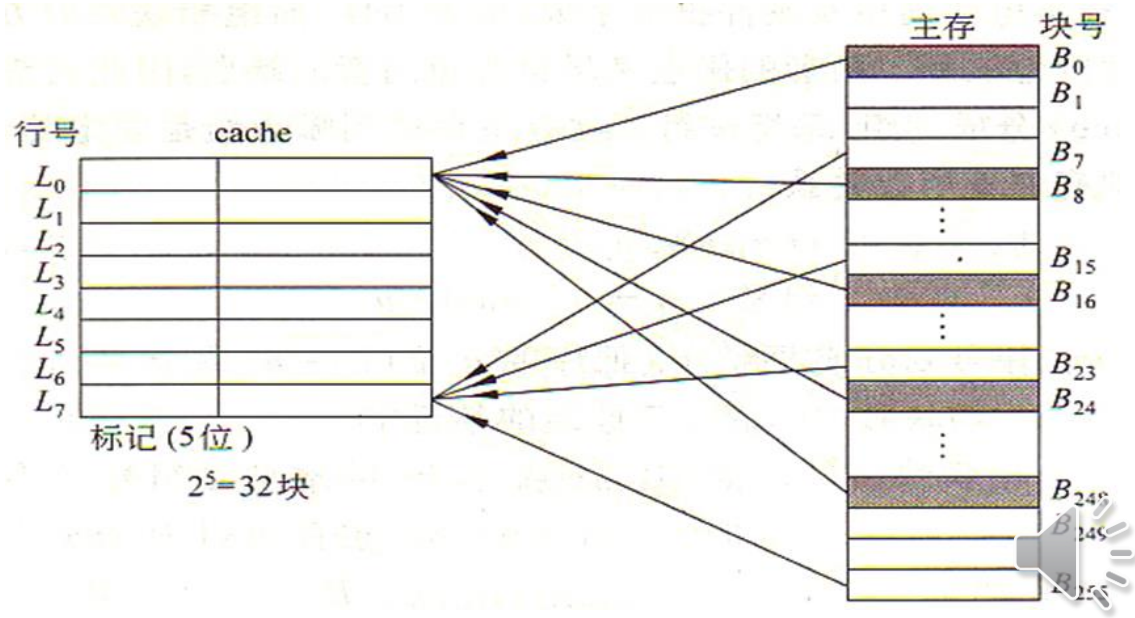
灵活；比较器耗时。



3.6.2 主存与cache的地址映射 —— 直接映射方式

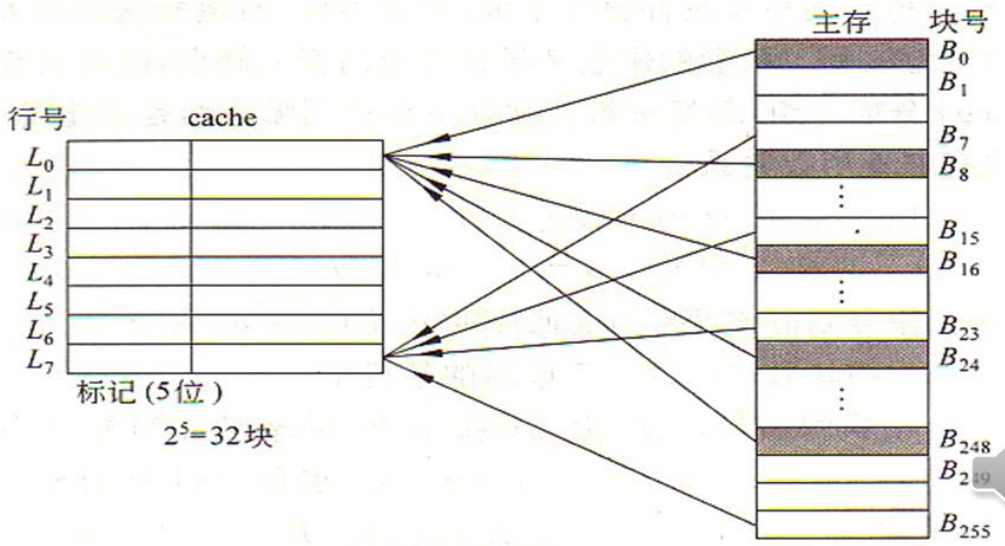
（1）主存按照cache的块数进行分组，如图主存中的256个数据块，按照8块一组，划分为32个组；

（2）主存物理地址=组号5位+组内块编号3位+块内地址7位。



3.6.2 主存与cache的地址映射 —— 直接映射方式

主存中每组的第0块（B0、B8、B16...B248）只能载入到L0块（行）；
主存中每组的第1块（B1、B9、B17...B249）只能载入到L1块（行）；
……；
主存中每组的第7块（B7、B15、B23...B255）只能载入到L7块（行）。



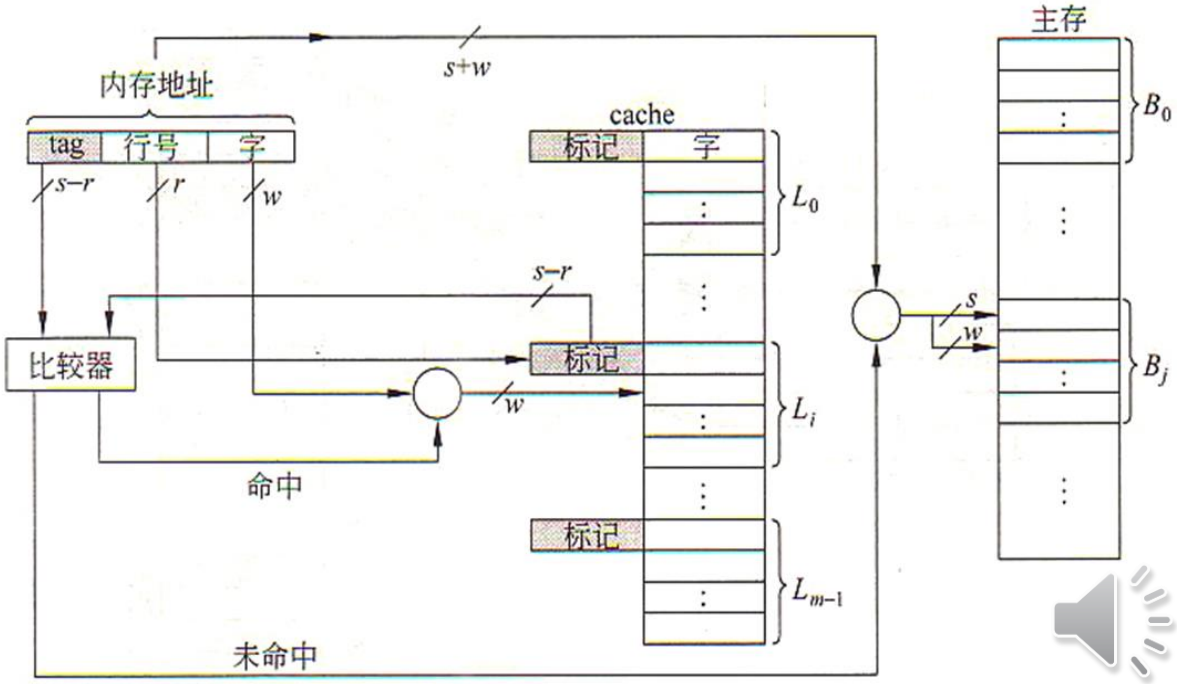
3.6.2 主存与cache的地址映射 —— 直接映射方式

（1）根据物理地址组内块编号r，直接对应cache中同编号块;

（2）比较组号s-r是否一致；若一致，则命中，否则未命中。

特点：

——对应、灵活性差。



3.6.2 主存与cache的地址映射 —— 组相联映射方式

全相联映射方式太灵活、比较工作量大；

直接映射方式——对应太死；

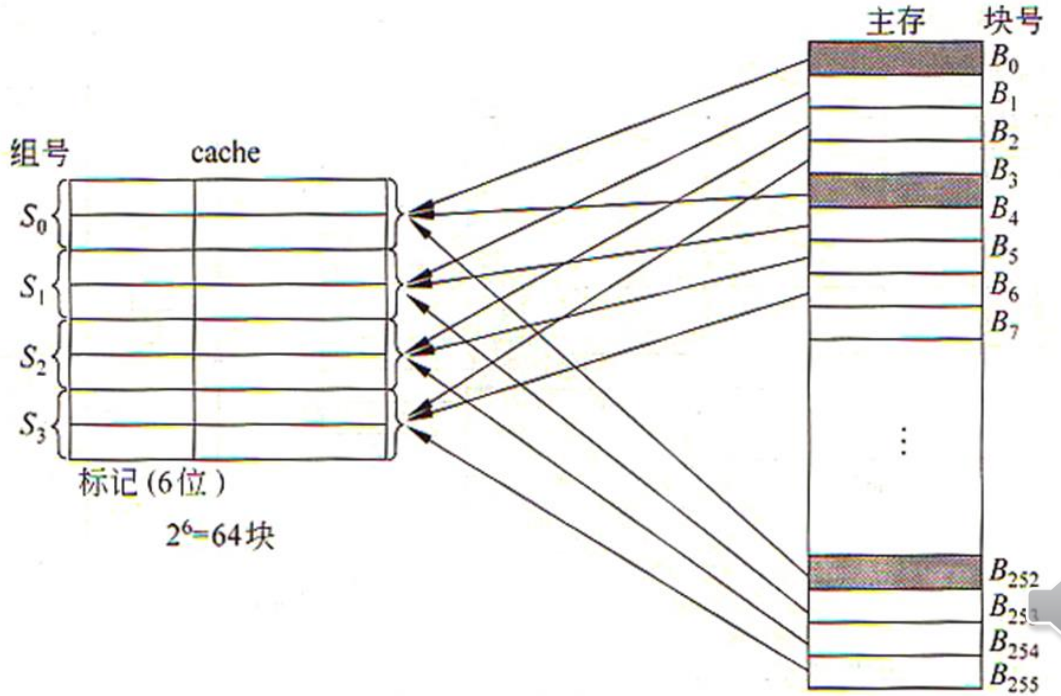
组相联映射方式是前两种方式的折衷方案，即将cache、主存都分组，适当提高灵活性；

最常采用组相联映射方式。

3.6.2 主存与cache的地址映射 —— 组相联映射方式

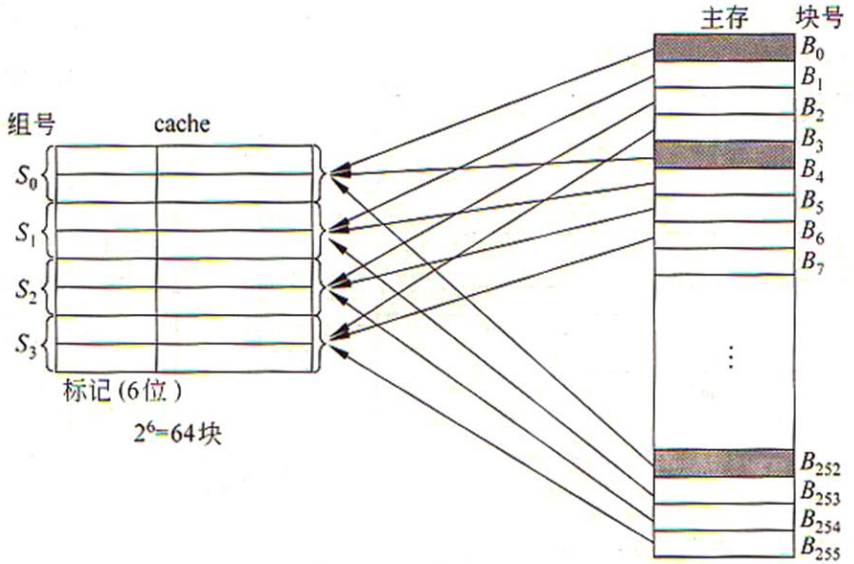
（1）cache中每两个块（行）为一组，共划分为4组；

（2）主存按照cache中的组数进行分组，共划分为64个组，每个组4块。



3.6.2 主存与cache的地址映射 —— 组相联映射方式

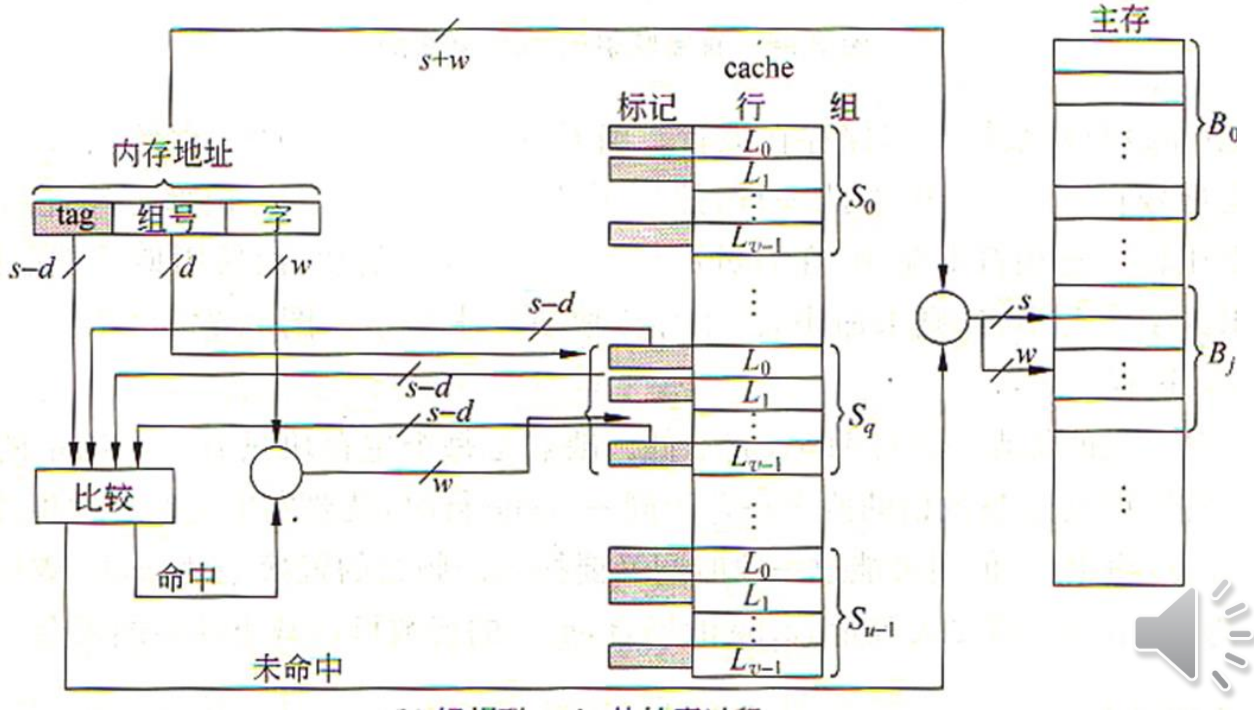
主存中每组的第0块（B0、B4、...B252）可以载入到S0组中任意一块；
主存中每组的第1块（B1、B5、...B253）可以载入到S1组中任意一块；
主存中每组的第2块（B2、B6、...B253）可以载入到S1组中任意一块;
主存中每组的第3块（B3、B7、...B255）可以载入到S3组中任意一块。



3.6.2 主存与cache的地址映射 —— 组相联映射方式

（1）根据物理地址组内块编号d，直接对应cache中同编号组;

（2）比较cache组中所有块的s-d是否一致；若一致，则命中，否则未命中。



3.6.3 替换策略

替换策略：当一个新的主存块需要交换到cache、而允许存放此块（行）的位置已满时，需要选择哪一块（行）被替换出cache，这需要一种策略机制。

3.6.3 替换策略

直接映射方式：内存中每组的第i块只能映射到cache中的第i块，是固定的一对一关系，不需要替换策略。

全相联映射方式：内存中的每一块可以映射到cache中的任意块（行），存在替换策略的问题。

组相联映射方式：内存中每组的第i块可映射到cache的第i组中的任意一块（行），存在替换策略的问题。

3.6.3 替换策略

最不经常使用(LFU)算法：上次替换到本次替换之间的一段时间内被访问次数最少的块替换出去，不能严格反映近期访问情况。

近期最少使用(LRU)算法：将近期内（未限定在两次替换之间）长久未被访问过的块换出。

随机替换策略：随机地选择一块替换出去。

3.6.4 cache的写操作策略

写操作策略：由于cache的内容只是主存部分内容的复制，它应当与主存内容保持一致。而CPU对cache的写操作，更改了cache的内容，但并不意味着更改了主存的内容，必须有一种策略机制来保证cache和主存中的内容保持一致。

3.6.4 cache的写操作策略

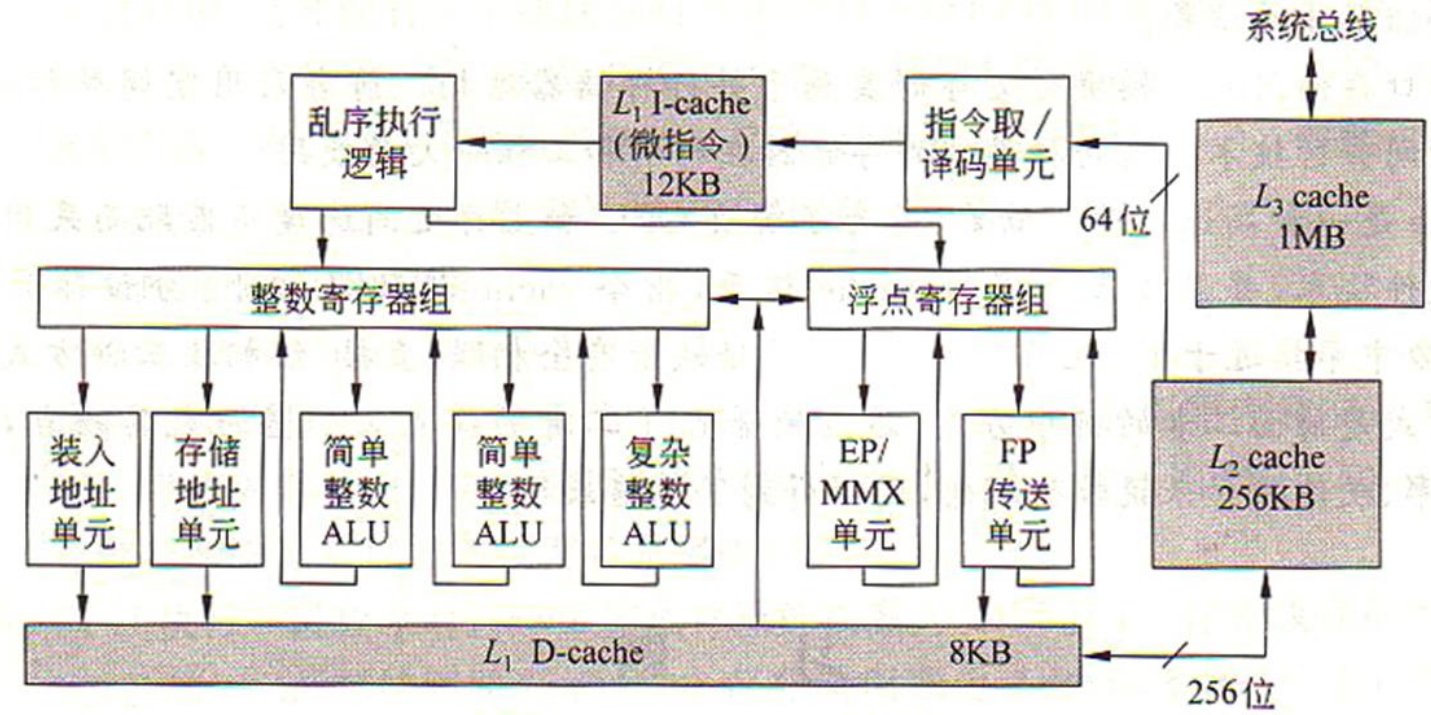
写回法：当CPU写cache命中时，只修改cache的内容，而不立即写入主存。只有当此块被替换出去时才写回主存。这种方法减少了访问主存的次数，但是存在数据不一致性的隐患。

全写法：当写cache命中时，cache与主存同时发生写操作；当写cache未命中时，直接写主存。该方法维护了cache与主存的内容一致性，但是效率低。

3.6.4 cache的写操作策略

写一次法：是指第一次写cache命中时要同时写入主存，以后的写操作只写cache，替换出去时再写主存。第一次写cache命中时同时写入主存，目的是通知主存和其它cache，这个数据块发生了写操作，应实施数据一致性策略。

3.6.5 Pentium 4 的cache组织



Pentium 4 的 cache 布局图

片内三级cache：L1、L2、L3。 L1 cache：L1 I-cache、L1 D-cache。

3.7 虚拟存储器

- 虚拟存储器的基本概念
- 页式虚拟存储器
- 段式虚拟存储器
- 段页式虚拟存储器
- 替换算法与写策略
- 虚拟存储器实例

3.7.1 虚拟存储器的基本概念

虚拟存储器的引入：

（1）是否放得下？希望提供一个足够大的存储器地址空间（编程空间），不需编写程序时考虑实际内存是否放得下；

（2）放什么位置？在多用户多任务系统中，多用户/多任务共享全部主存，同时执行多道程序。在编制程序时无法确定内存位置，必须等到程序运行时才会动态分配。

3.7.1 虚拟存储器的基本概念

虚拟存储器：

基于程序的局部性访问原理（在一段时间范围内，执行的程序是一个大程序中相对集中的一部分程序模块，也称为当前活跃部分），通过硬件/操作系统实现主存-外存之间的信息部分调入调出，使用户感觉有一个足够大的存储器地址空间，可以为更大或更多的程序所使用。

3.7.1 虚拟存储器的基本概念

技术意义：

虚拟存储器技术的目的是解决主存容量不足，属于主存和外存之间的问题。

cache技术的目的是解决主存速度问题，属于CPU和主存之间的问题。

3.7.1 虚拟存储器的基本概念

物理地址空间、物理地址：

物理存储器的实际地址空间称为物理地址空间；

物理地址空间受CPU外部地址总线控制，设地址总线为n根、则物理地址空间最大为2^n；

物理地址空间中的每一个存储单元都必须有一个唯一的地址编码，称为物理地址。

3.7.1 虚拟存储器的基本概念

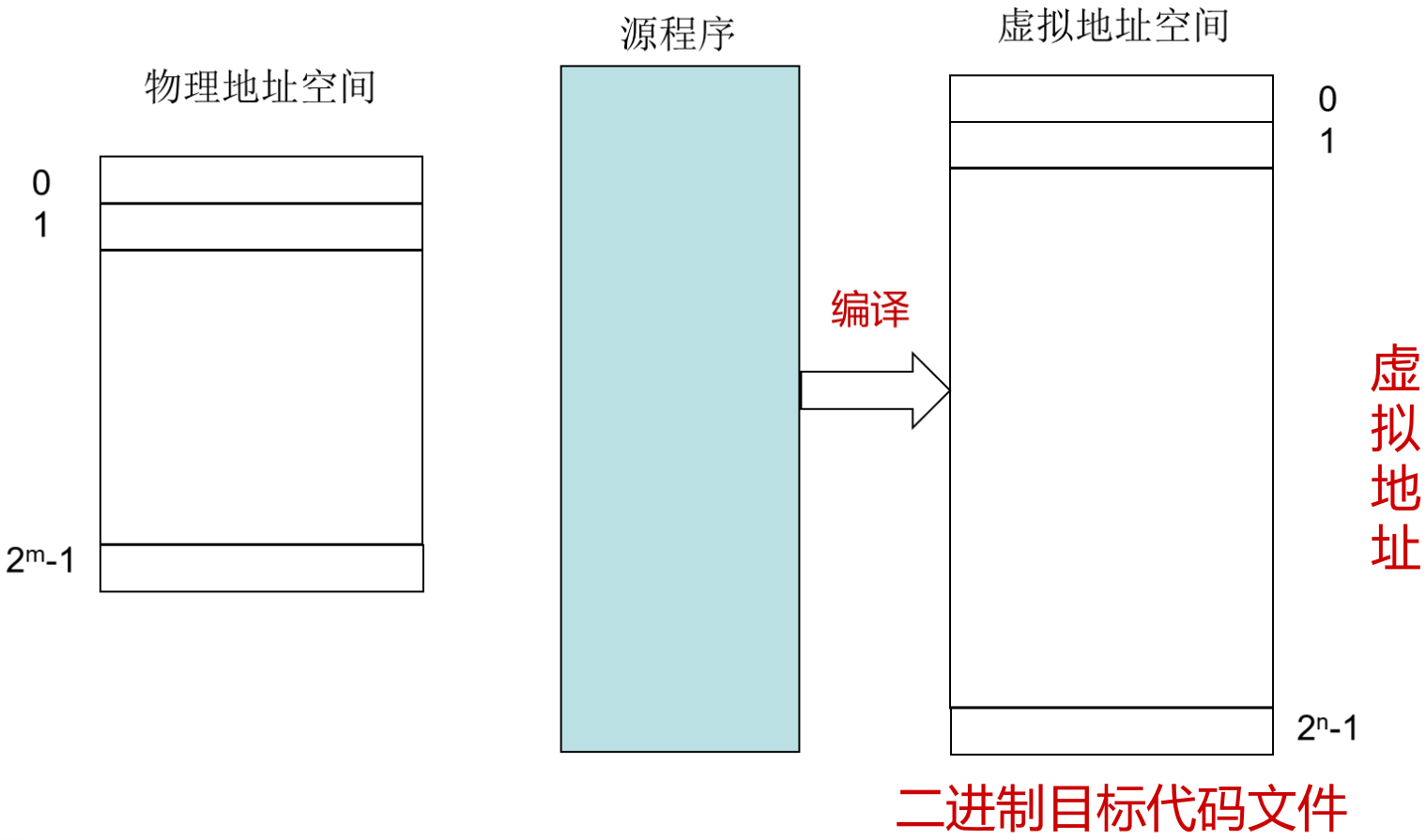
虚拟地址空间、虚拟地址：

通过虚拟存储器技术为用户提供的比实际物理地址空间大、且足够使用的存储器地址空间，称为虚拟（逻辑）地址空间；

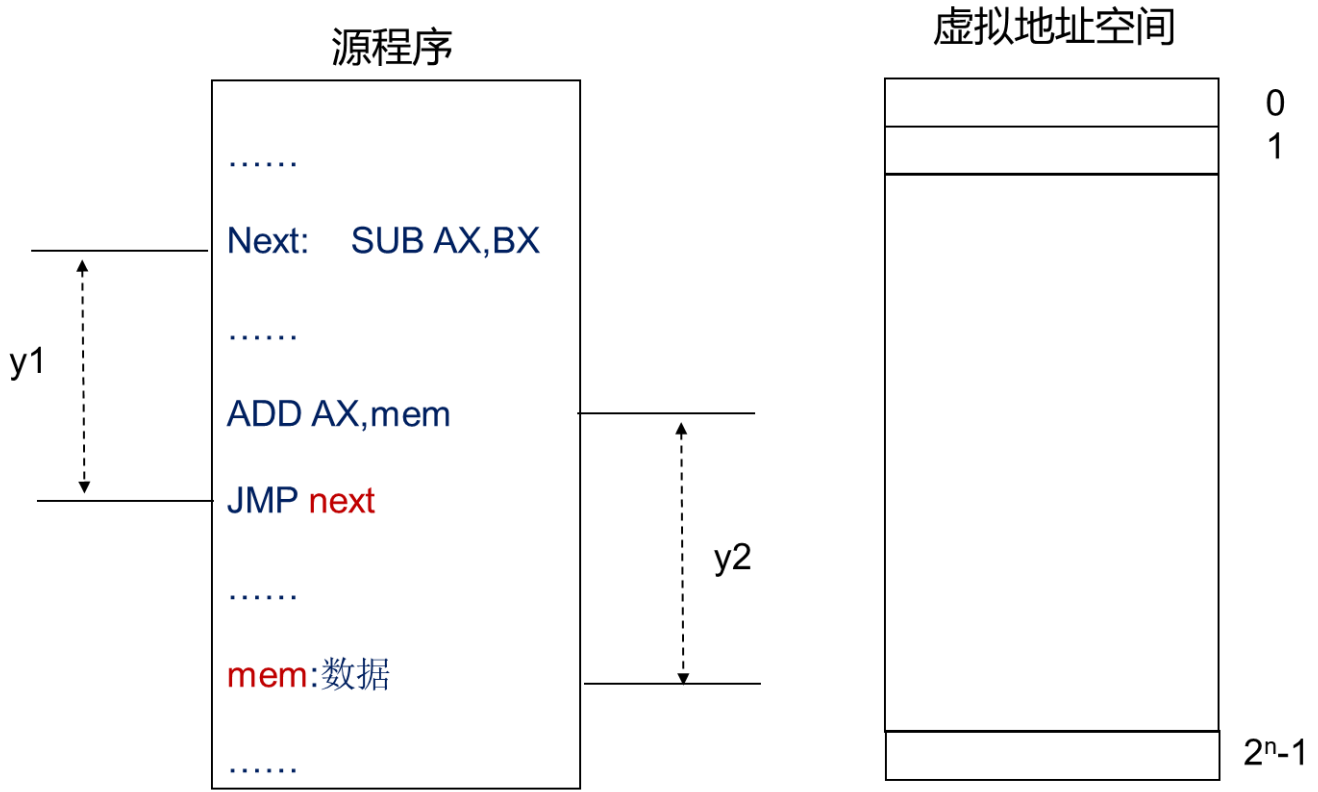
编译程序在虚拟地址空间上生成的逻辑地址，称为虚拟地址；

工作在虚拟地址模式下的CPU负责解释虚拟地址，并通过相应的策略机制转换为物理地址、访问物理内存。

3.7.1 虚拟存储器的基本概念



3.7.1 虚拟存储器的基本概念



3.7.1 虚拟存储器的基本概念

信息交换基本单位：

主存-外存之间信息交换的基本传输单位可采用几种不同的方案：段、页、段页，相应地分别称为段式虚拟存储器、页式虚拟存储器、段页式虚拟存储器。

3.7.1 虚拟存储器的基本概念

—段

段是利用程序的模块化性质，按照程序的逻辑结构划分成的多个相对独立部分。通常是指独立的功能模块、数据模块，如过程、子程序、数据文件等。

段作为独立的逻辑单位，可以被其它段调用，段间连接可形成规模更大的程序。

3.7.1 虚拟存储器的基本概念

—— 段

段的优点：段的分界与程序的自然分界相对应，段的逻辑独立性使它易于编译、管理、修改和保护，以段为单位在主存-外存之间交换不会改变程序的结构性质，可保证程序的完整性、一致性。

段的缺点：由于段的长度各不相同，给主存空间的预留、分配带来麻烦。

3.7.1 虚拟存储器的基本概念

—— 页

物理地址空间、虚拟地址空间（待执行程序的二进制目标代码文件）均按约定长度划分为若干数据块，一个数据块称为一页。

优点：页的大小是固定的，新页调入主存容易合理地预留、分配存储空间。

缺点：因为页不是独立的逻辑单位，处理、保护和共享都不及段方便，可能破坏程序的逻辑结构。

3.7.1 虚拟存储器的基本概念

——段页

采用分段和分页相结合的方法。

程序按模块分段，段内再分页，外存和主存之间信息交换以页为单位。

3.7.2 页式虚拟存储器 ——物理页、逻辑页

将物理地址空间、虚拟地址空间均按照约定长度划分为若干页，主存和外存之间的调入调出以页为基本单位。

物理地址空间的页称为物理页（实页），物理地址 = 物理页号 + 页内偏移地址。

虚拟地址空间（待执行程序的二进制目标代码文件）的页称为逻辑页（虚页），虚拟地址 = 虚拟页号 + 页内偏移地址。

3.7.2 页式虚拟存储器 ——基于页表的地址变换

页表：由”虚页号+装载标志+实页号+修改标志位+……”构成，每个虚页在页表中占一行。

当某个虚页交换到内存时，在页表中对应的虚页行上记载实页号、装载标志等信息。

虚页号	装载标志	实页号	修改标志	……
0	0			……
1	1	5	0	……
2	0			……
3	0			……
……	……	……	……	……
2^n-1	0			……

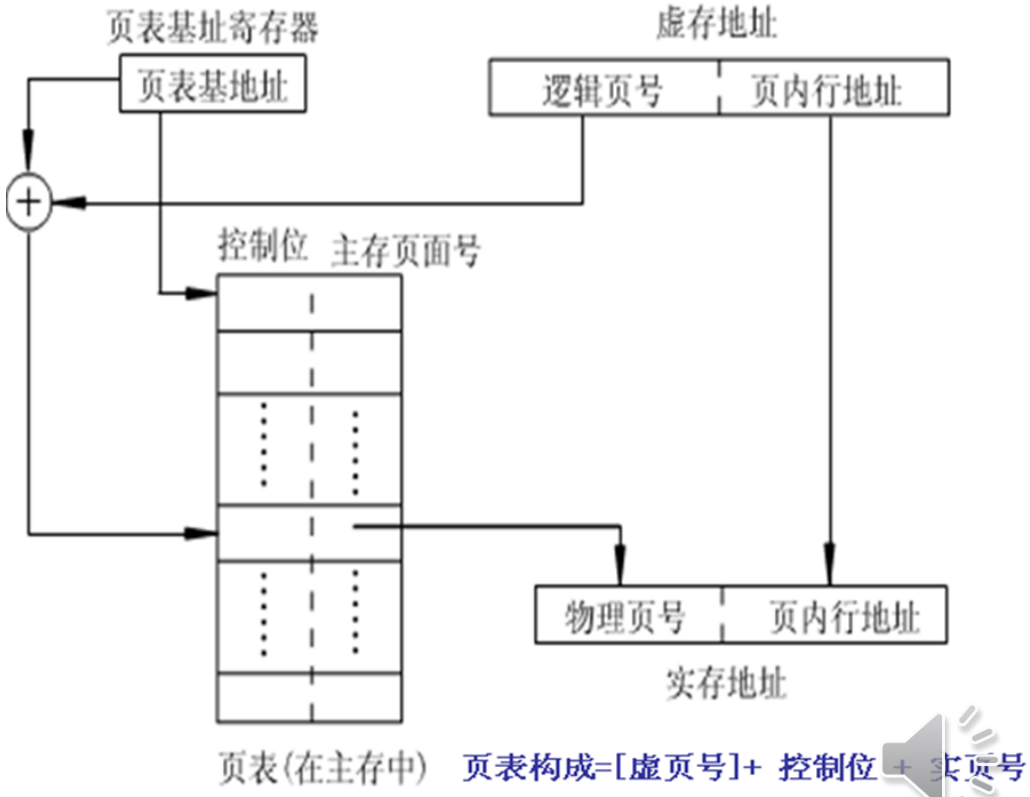
页表构成示意图

3.7.2 页式虚拟存储器 ——基于页表的地址变换

查表法：当CPU访问主存时，根据虚拟地址中的虚页号查页表。

若已装入，则取实页号，物理地址=实页号+页内地址；

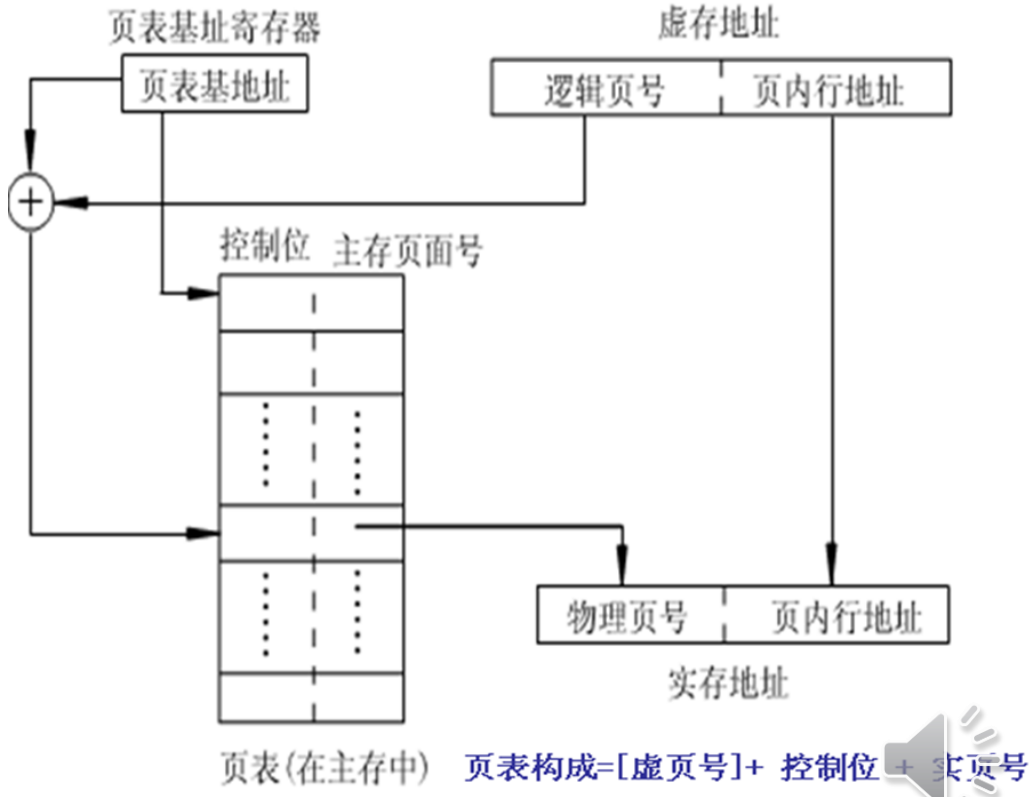
若未装入，则产生缺页中断，从外存调入页到内存。



3.7.2 页式虚拟存储器 ——基于页表的地址变换

页表基址寄存器：存放页表基地址（页表中首单元在内存中的物理地址）。

访问的页表行的地址=页表基地址+虚页号。

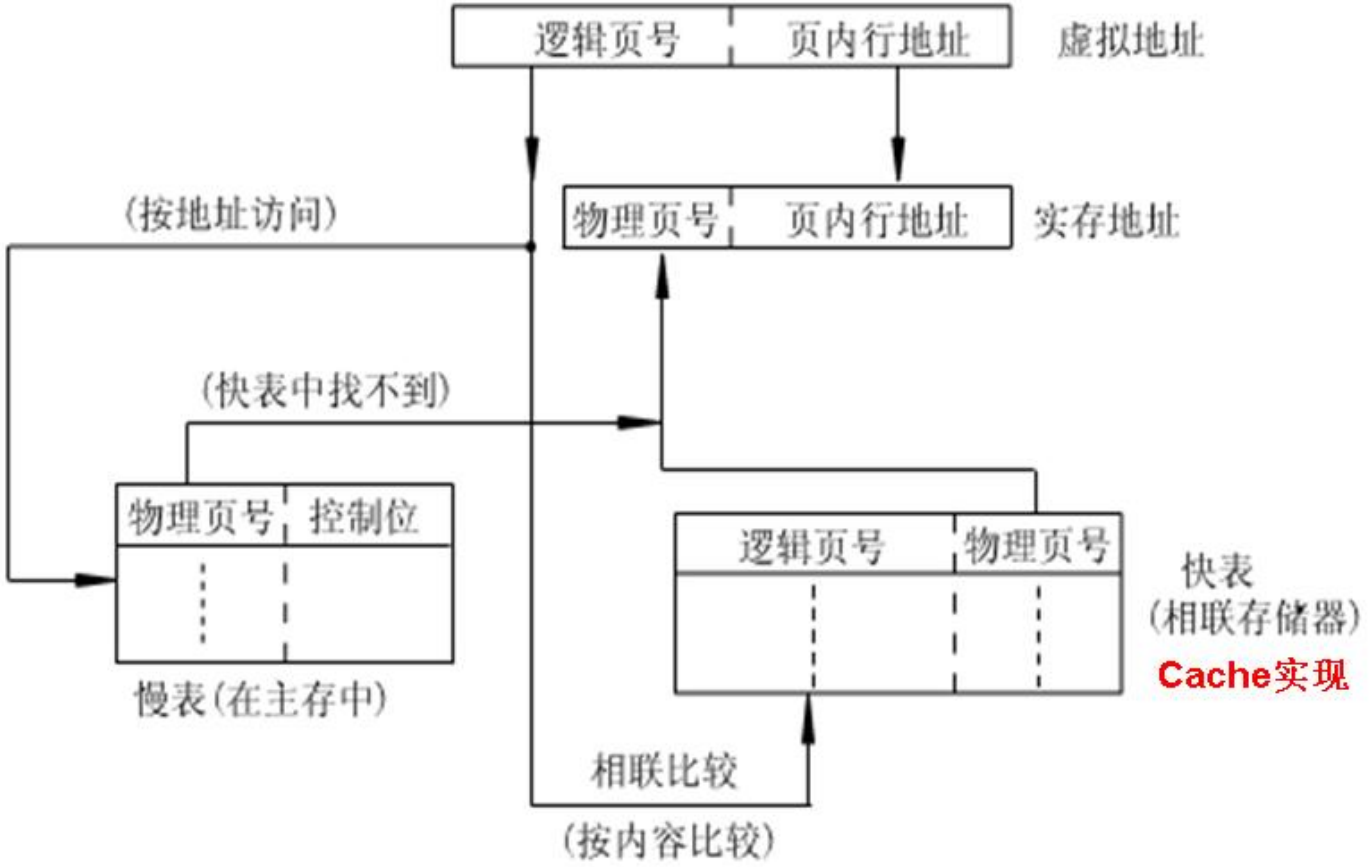


3.7.2 页式虚拟存储器 ——快表、慢表

在页式虚拟存储器中，虚拟地址通过查询页表转换为物理地址，而页表在主存中，因此进行一次访问主存储器操作必须要两次访问主存（一次根据虚拟地址查页表，获得物理地址；一次根据物理地址访问具体存储单元）。

显然效率不高，可以考虑把页表的当前最活跃部分存放在高速缓冲存储器cache中，即快表，从而提高页表的查询效率。

3.7.2 页式虚拟存储器 ——快表、慢表



3.7.3 段式虚拟存储器

基于段表的管理和地址变换：

段表 = 段号+段起始地址+段长度+装载标志+修改标志+……；

根据虚拟地址中的逻辑段号、查询段表、取得段起始地址、与虚拟地址中的段内地址组合形成物理地址。

3.7.3 段式虚拟存储器

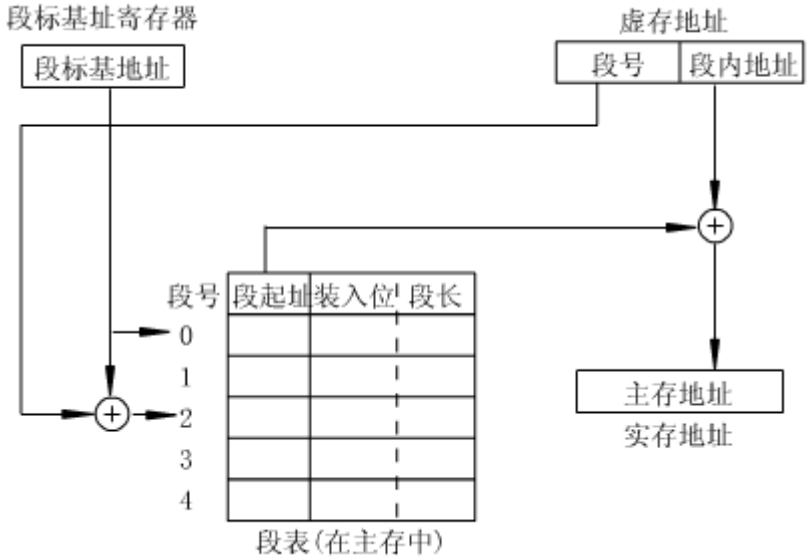
基于段表的管理和地址变换：

段表 = 段号+段起始地址+段长度+装载标志+修改标志+……；

根据虚拟地址中的逻辑段号、查询段表、取得段起始地址、与虚拟地址中的段内地址组合形成物理地址。

段表基址寄存器：

存放段表首单元的物理地址。

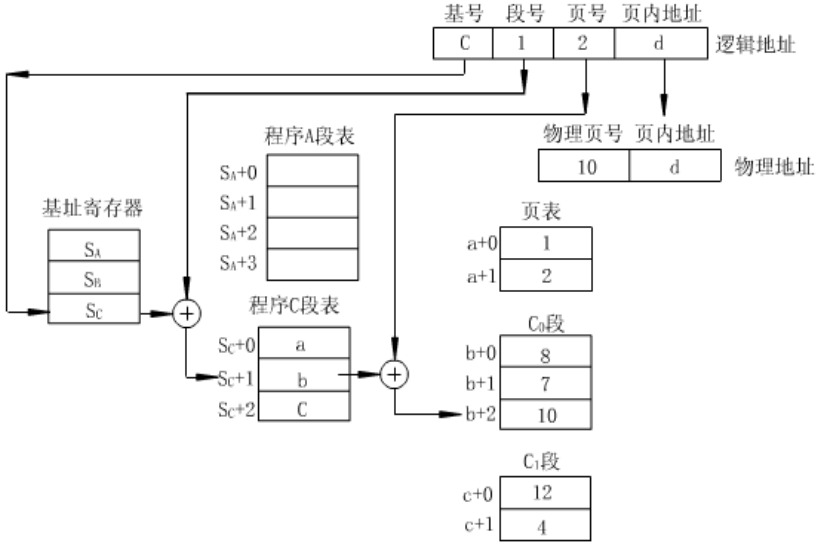


3.7.4 段页式虚拟存储器

段表、页表：先分段、然后分页，因此由虚拟地址变换为物理地址时需要先查段表、后查页表。

基号：在多道程序运行的情况下、每道程序都有一个标识号，称为基号。

基址寄存器：每个基址寄存器存放每道程序所对应的段表的首单元的物理地址，根据基号确定基址寄存器。



3.7.5 替换算法与写策略

访问的页（或段）不在主存中，同时主存已满，采用什么样的规则来将某一页（或段）交换出主存，即替换算法。

LRU：近期最少使用算法

LFU：最不经常使用算法

FIFO：先进先出算法

3.7.5 替换算法与写策略

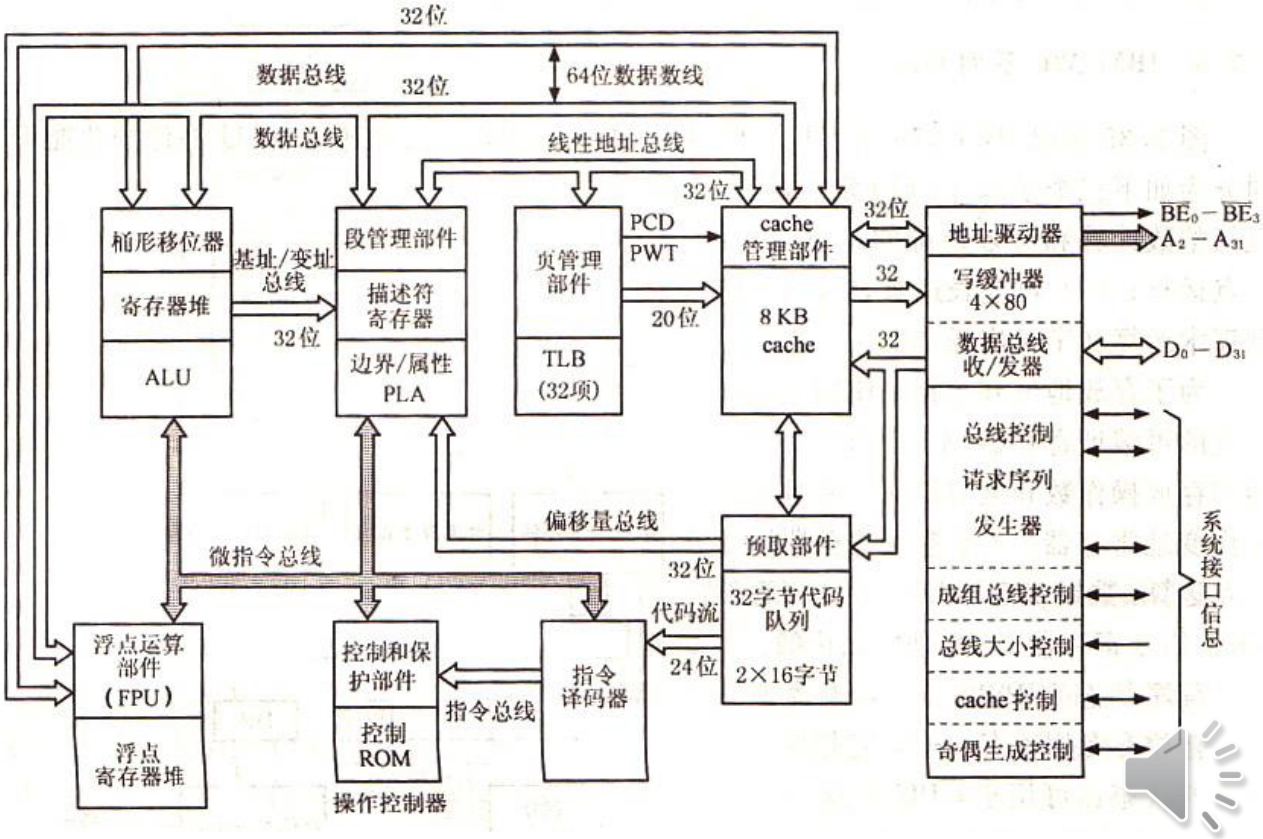
对于将被替换出去的页（或段），假如该页（或段）调入主存后没有被修改，就不必进行处理，否则就把该页（或段）重新写入外存，以保证外存中数据的正确性、一致性。

为此，在页表/段表中，需要设置修改标志位。

3.7.6 虚拟存储器实例

cache管理部件：

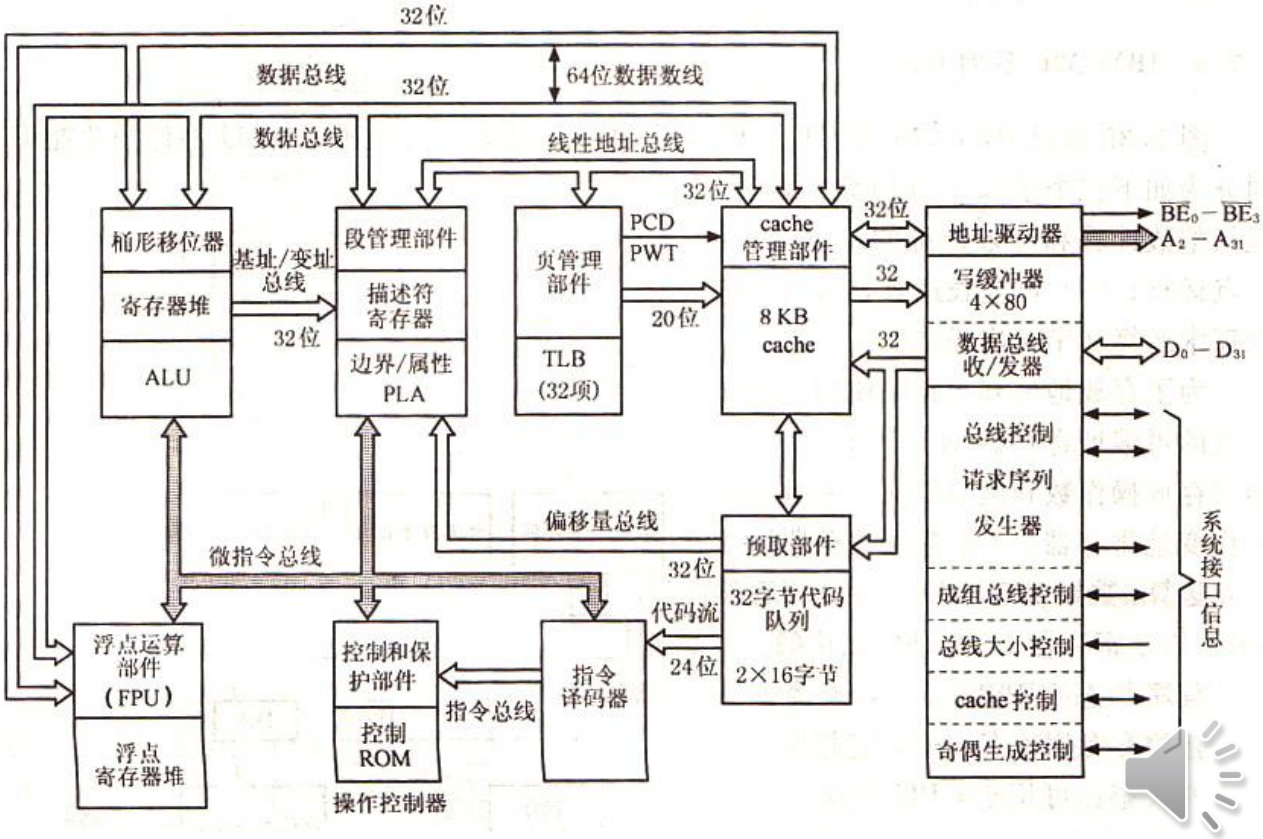
块交换、地址映射、替换策略、写策略等。



3.7.6 虚拟存储器实例

段管理部件SU：

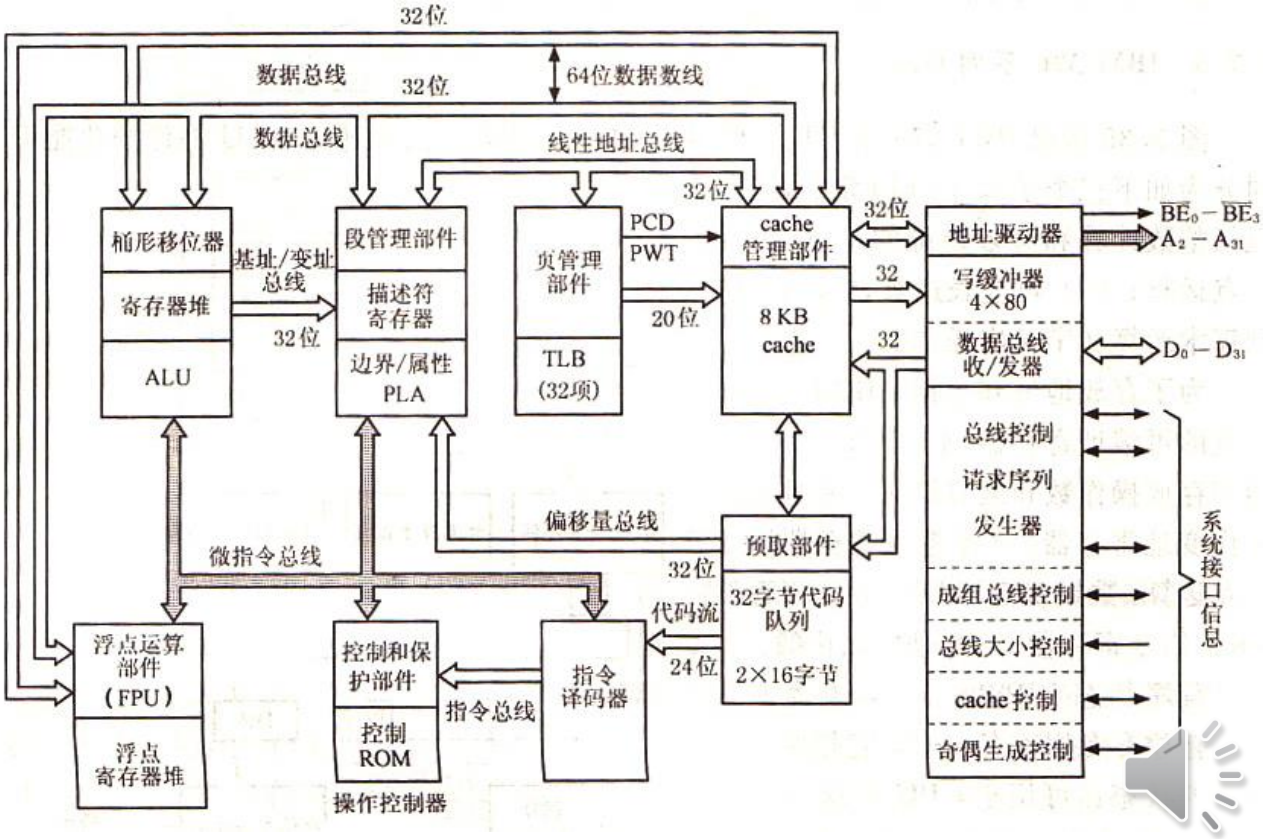
段表维护、基于段表的地址变换、替换策略、写策略等。



3.7.6 虚拟存储器实例

页管理部件PU：

页表维护、基于页表的地址变换、替换策略、写策略等。

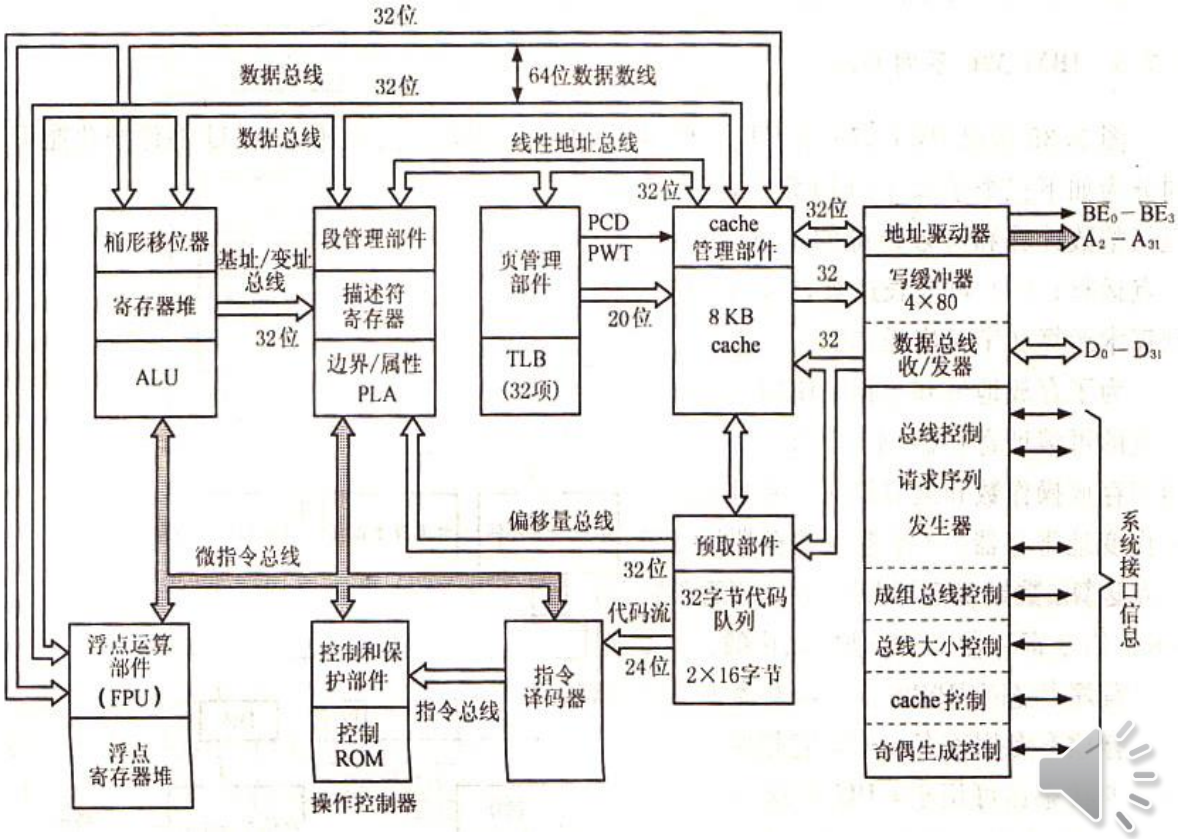


3.7.6 虚拟存储器实例

分段不分页模式：段式虚拟存储器。

分段分页模式：段页式虚拟存储器。

不分段分页模式：页式虚拟存储器。



3.7.6 虚拟存储器实例

分段不分页模式：段式虚拟存储器。

分段分页模式：段页式虚拟存储器。

不分段分页模式：页式虚拟存储器。