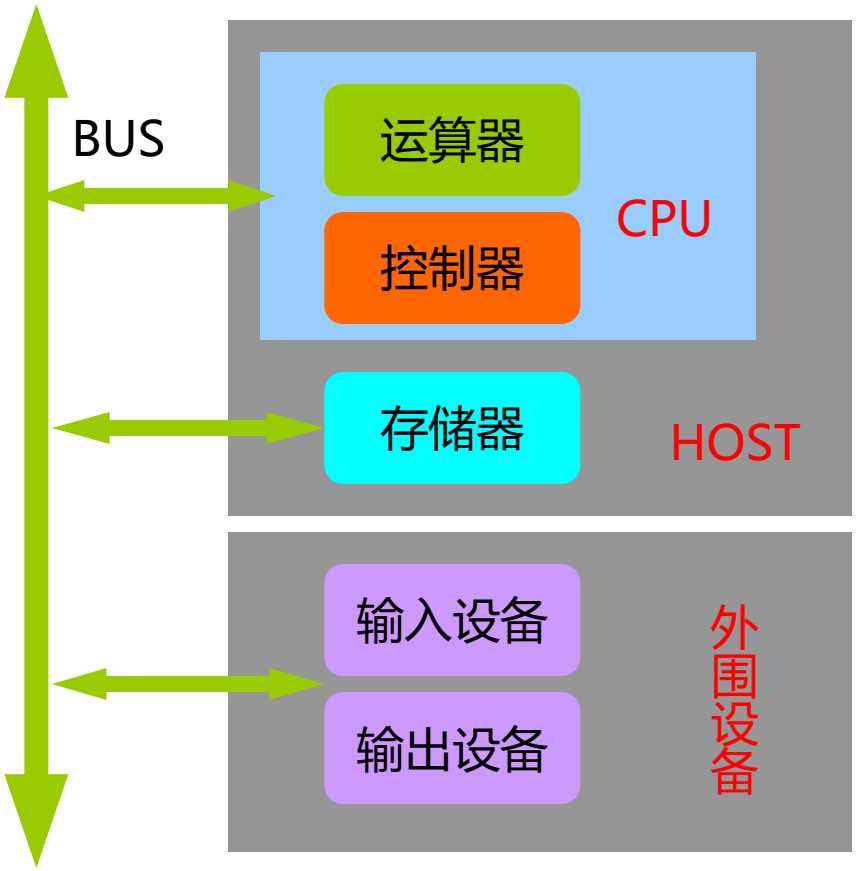
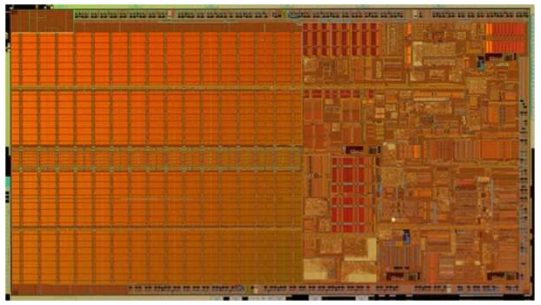
CPU

# 计算机组成原理


# 四、存储系统


BUS
运算器
控制器
CPU
存储器
HOST
输入设备
输出设备
外围设备

# 花样繁多的存储器




Transcend
aXeRam
DDR2 800MHz 1GB


MO
1.3 GB

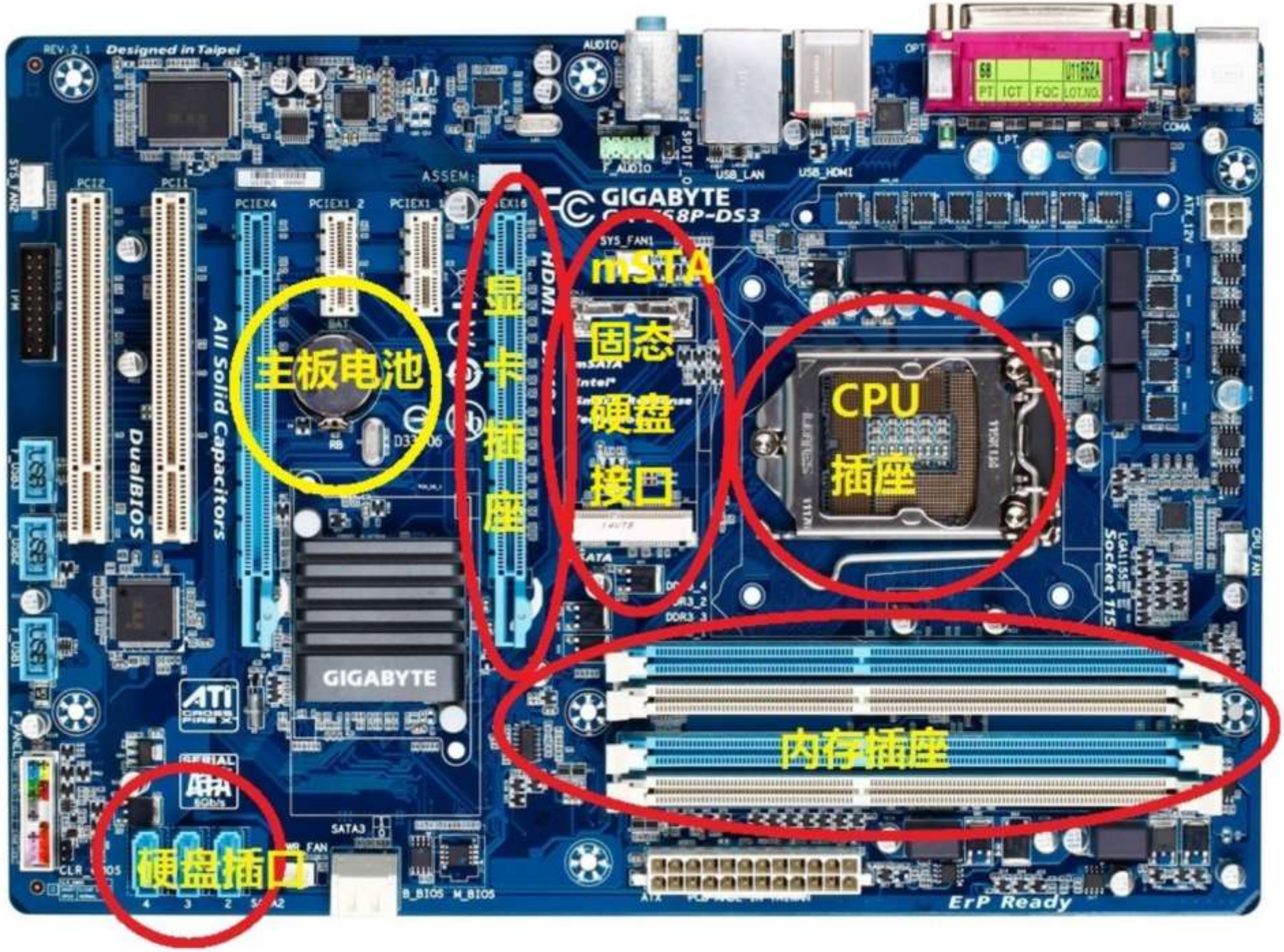

FUJIFILM
LTO
Ultrium 6
DATA CARTRIDGE
2.50
6.25


SSD

# 内存


Kingston
KVR
9905216-048.A00LF
2147992-2131516
ASSY IN CHINA (1)
KVR400X64C3A/512
2.6V
Warranty Void If Removed
GENUINE KINGSTON

# 外存


主板电池
显卡插座
mSTA
固态
硬盘
接口
CPU
插座
内存插座
硬盘插口
GIGABYTE
Designed in Taipei
All Solid Capacitors
DualBIOS
ErP Ready

| 配件 | 详细型号 |
| --- | --- |
| CPU | i7-12700K/12700KF/12700/12700F |
| 主板 | 微星(MSI)PRO Z690-A WIFI DDR4电脑主板 |
| 显卡 | ASUS TUF-GeForce RTX3080TI-12G-GAMING |
| 内存 | 宏碁掠夺者 雷霆战甲系列<br>32G(16G×2)套 DDR4 3600 B-die颗粒 |
| 硬盘 | 三星（SAMSUNG）1TB SSD 980 固态硬盘 |
| 机箱 | 安钛克复仇者、追风者P500A |
| 电源 | 安钛克 NE 850(白色)/巨龙GW-EPS1000DA |
| 散热 | 利民（Thermalright）水冷<br>Frozen MAGIC 360 SCENIC |

# 本章主要内容

- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 4.1 存储器概述

- 4.1.1 存储器分类
- 4.1.2 存储器技术指标
- 4.1.3 存储系统层次结构
- 4.1.4 主存的基本结构
- 4.1.5 主存中数据的存放

# 存储器分类

- 按存储介质分
- 按存取方式分
- 按读写功能分
- 按信息的可保存性分
- 按在计算机系统中的作用分

# 按存储介质分

- 半导体存储器
  - 双极型存储器 MOS存储器
  - 速度快、功耗低
- 磁存储器
  - 磁芯、磁带、磁盘
  - 容量大，速度慢、体积大
- 激光存储器
  - CD-ROM CD-RW CD-R
  - DVD-ROM DVD-RW DVD-R
  - 便于携带，廉价，易于保存

# 按存取方式分

- 随机存储器
  - 存取时间与物理位置无关
  - 磁芯、半导体存储器
- 顺序存储器
  - 存取时间与物理位置有关
  - 磁带、激光存储器
- 直接存储器
  - 不必经过顺序搜索就能在存储器中直接存取信息的存储器
  - 磁盘存储器

# 按读/写功能分

- 只读存储器 (ROM)
  - 存储器内容是预置的，固定的，无法改写
- 读/写存储器
  - 既能读出也能写入的存储器
  - 随机存储器RAM

# 按信息的可保存性分

- 易失性存储器 *Volatile Memories*
  - 断电后信息消失
  - SRAM
  - DRAM
- 非易失性存储器*Non-Volatile Memories*
  - 断电后仍能保存信息
  - 磁存储器、激光存储器、NVRAM

# 按在计算机系统中的作用分

- **寄存器存储器**：多个寄存器组成的存储器，一般由几个或几十个寄存器组成，其字长一般与计算机字长相同，主要用来存放**地址**、**数据**及**运算的中间结果**，速度与CPU匹配，容量很小。
- **高速缓冲存储器Cache**：隐藏在寄存器和主存之间的一个高速小容量存储器，用于**存放CPU 即将或经常要使用的指令和数据**。它一般采用静态RAM 构成，用于缓冲CPU与慢速主存之间的性能差异，**提高存储系统的访问速度**。

# 按在计算机系统中的作用分

- **主存储器**：简称主存，是CPU 中除寄存器外唯一能直接访问的存储器，用于**存放指令和数据**。CPU 通过主存地址直接、随机地读写主存储器。主存一般由半导体存储器构成，但注意主存并不是单一的内存，还包括BIOS、硬件端口等。
- **外存储器**：简称外存或辅助存储器。外存容量很大，但存取速度相对较低。目前广泛使用的外存储器包括**磁盘**、**磁带**、**光盘存储器**、**磁盘阵列**和**网络存储系统**等。外存用来存放当前暂不参与运行的程序和数据，以及一些需要永久性保存的数据信息。

# 存储系统主要技术指标-存储容量

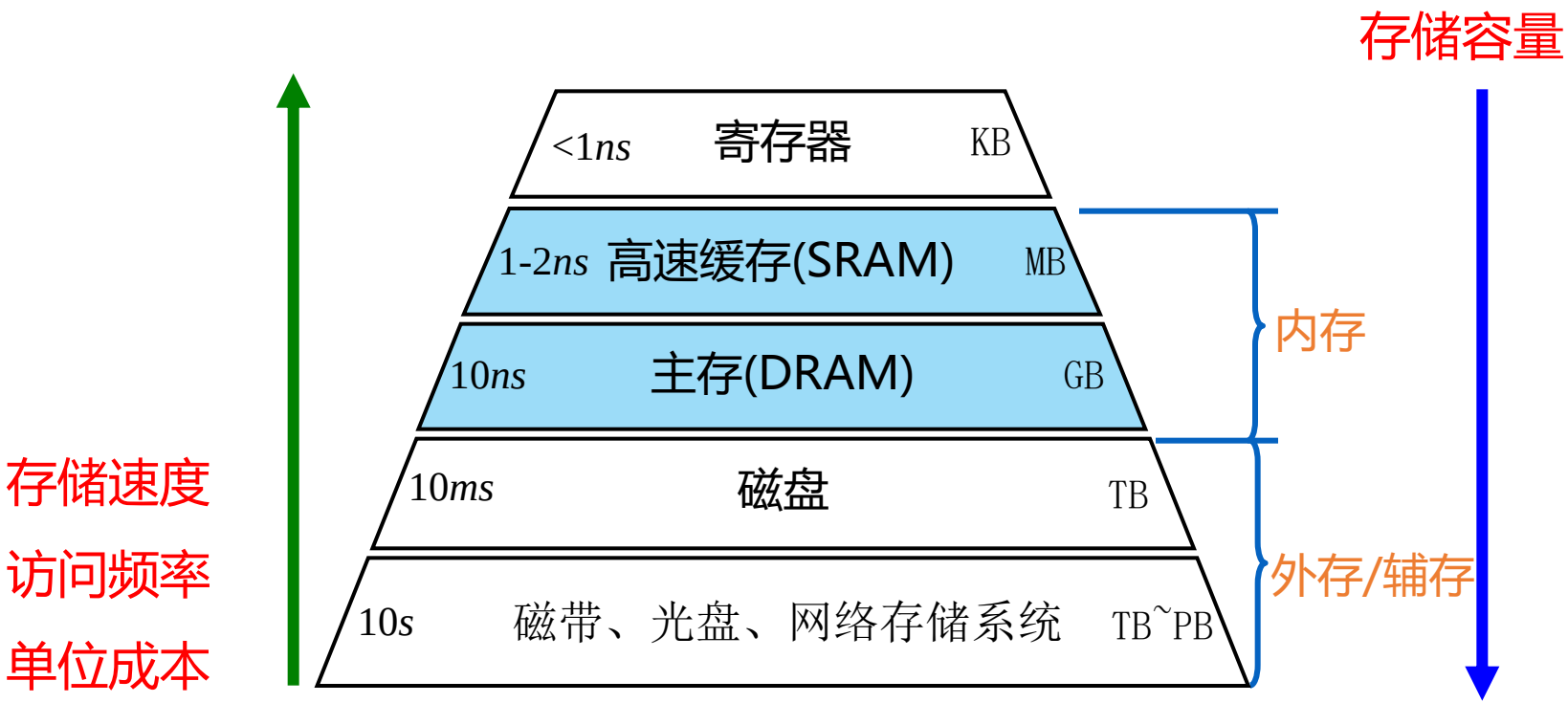
- 存储器可以存储的二进制信息总量称为存储容量。
- 存储容量可以采用**比特位**或者**字节**来表示。

1B(Byte 字节)=8bit，
1KB (Kilobyte 千字节)=1024B，
1MB (Megabyte 兆字节 简称“兆”)=1024KB，
1GB (Gigabyte 吉字节 又称“千兆”)=1024MB，
1TB (Trillionbyte 万亿字节 太字节)=1024GB，
其中1024=2^10 ( 2 的10次方)，
1PB（Petabyte 千万亿字节 拍字节）=1024TB，
1EB（Exabyte 百亿亿字节 艾字节）=1024PB，
1ZB (Zettabyte 十万亿亿字节 泽字节)= 1024 EB，

# 存储系统主要技术指标-存取速度

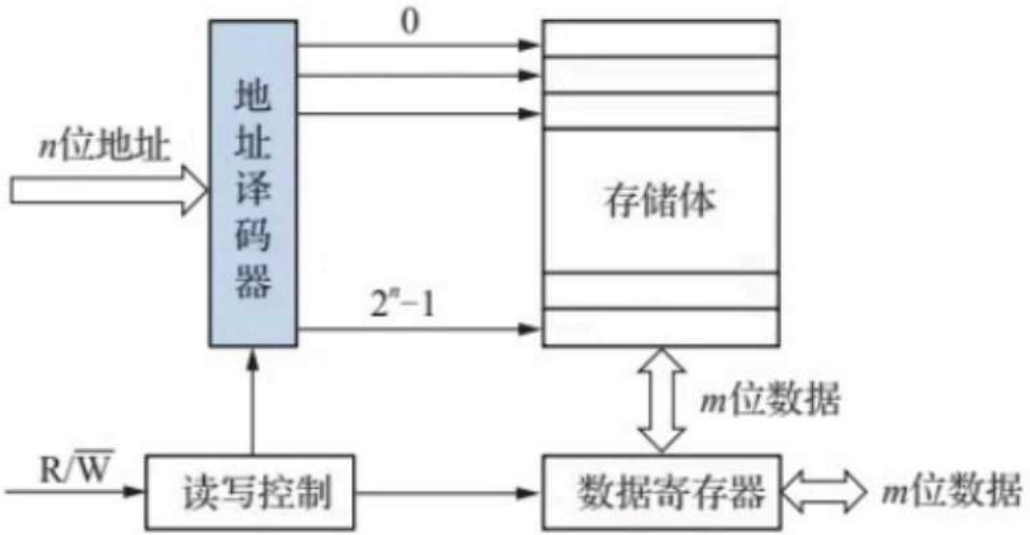
- 存储时间
  - 接受到读写命令到从存储器中读出或写入信息所经历的时间
- 存储周期
  - 连续两次访问存储器所需要的最小时间间隔
- 存储器带宽
  - 单位时间内存储器存取的信息量
  - Byte/s
  - **光盘、磁盘、U盘、固态盘的读写带宽?**

# 存储系统分层结构


存储容量
存储速度
访问频率
单位成本
<1ns 寄存器 KB
1-2ns 高速缓存(SRAM) MB
10ns 主存(DRAM) GB
10ms 磁盘 TB
10s 磁带、光盘、网络存储系统 TB~PB
内存
外存/辅存

| | | 1980 | 1990 | 2000 | 2010 | 2010:1980 |
|---|---|---|---|---|---|---|
| CPU | Name | 8080 | 386 | Pentium II | Core i7 | / |
| | Clock rate(MHz) | 1 | 20 | 600 | 2,500 | 2,500 |
| | Cycle time(ns) | 1,000 | 50 | 1.6 | 0.4 | 2,500 |
| | Cores | 1 | 1 | 1 | 4 | 4 |
| | Effective Cycle time(ns) | 1,000 | 50 | 1.6 | 0.1 | 10,000 |
| SRAM | $/MB | 19,200 | 320 | 100 | 60 | 320 |
| | access time(ns) | 300 | 35 | 3 | 1.5 | 200 |
| DRAM | $/MB | 8,000 | 100 | 1 | 0.06 | 130,000 |
| | access time(ns) | 375 | 100 | 60 | 40 | 9 |
| | typical size(MB) | 0.064 | 4 | 64 | 8,000 | 125,000 |
| Disk | $/MB | 500 | 8 | 0.01 | 0.0003 | 1,600,000 |
| | access time(ms) | 87 | 28 | 8 | 3 | 29 |
| | typical size(MB) | 1 | 160 | 20,000 | 1,500,000 | 1,500,000 |

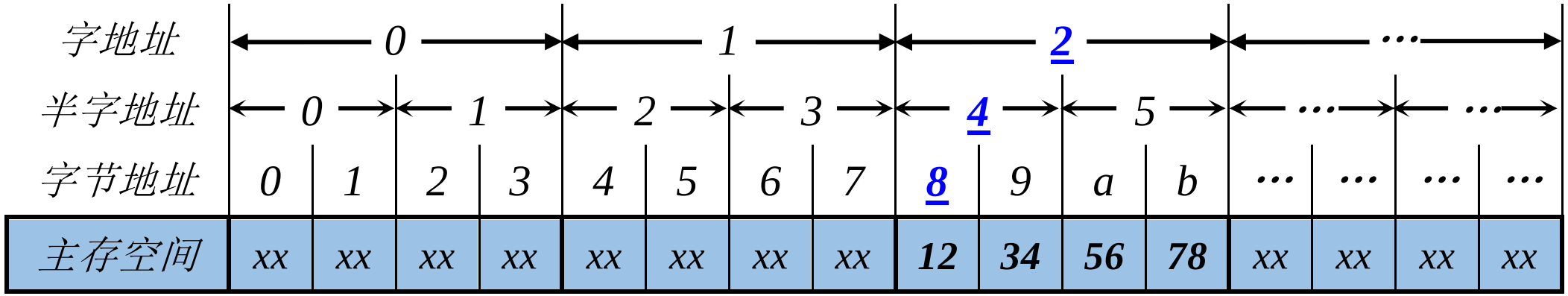
# 主存储器




- 主存是**机器指令直接操作的存储器**，采用主存地址进行随机访问，整个主存从空间逻辑上可以看作一个一维数组mem[]，每个数组元素存储一个m 位的数据单元，主存地址 addr就是数组的下标索引，数组元素的值mem[addr]就是主存地址对应的存储内容。
- **主存储器**：由存储体加上一些外围电路构成，外围电路包括地址译码器、数据寄存器和读写控制电路。
  - **地址译码器**：接收来自CPU的"位地址信号，经译码、驱动后形成2^n根地址译码信号，每一根地址译码信号连接一个存储单元。
  - **数据寄存器**：暂存 CPU送来的数据，或暂存从存储体中读出的 数据。
  - **读写控制电路**：接收 CPU的读写控制信号后产生存储器内部的控制信号，将指定地址的信息从存储体中读出并送到数据寄存器中供CPU使用，或将来自CPU并已存入数据寄存器的信息写入存储体中的指定单元。

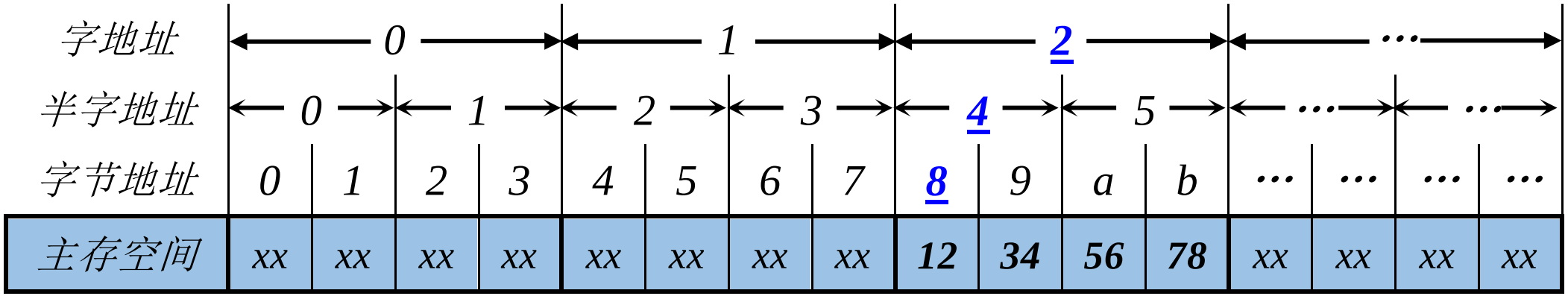
# 主存中数据的存放

- **存储字长**: 主存的一个存储单元所存储的二进制位数。
- **数据字长**(简称字长): 计算机一次能处理的二进制数的位数。
- 存储字长与数据字长不一定相同，如字长为32位的计算机所采用的存储字长可以是16位、32位或64位。
- 存储字长都是字节的整倍数，主存通常按字节进行编址。以32位计算机为例，主存既可以按字节访问,也可以按16位半字访问,还可以按照32位的字进行访问。按照访问存储单元的大小,主存地址可以分为**字节地址**、**半字地址**、**字地址**。

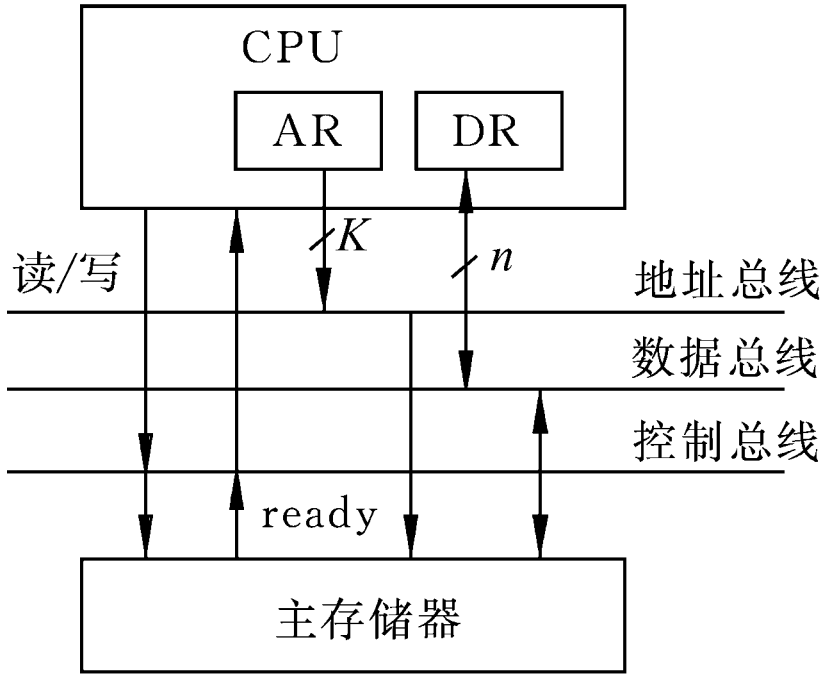


# 主存中数据的存放

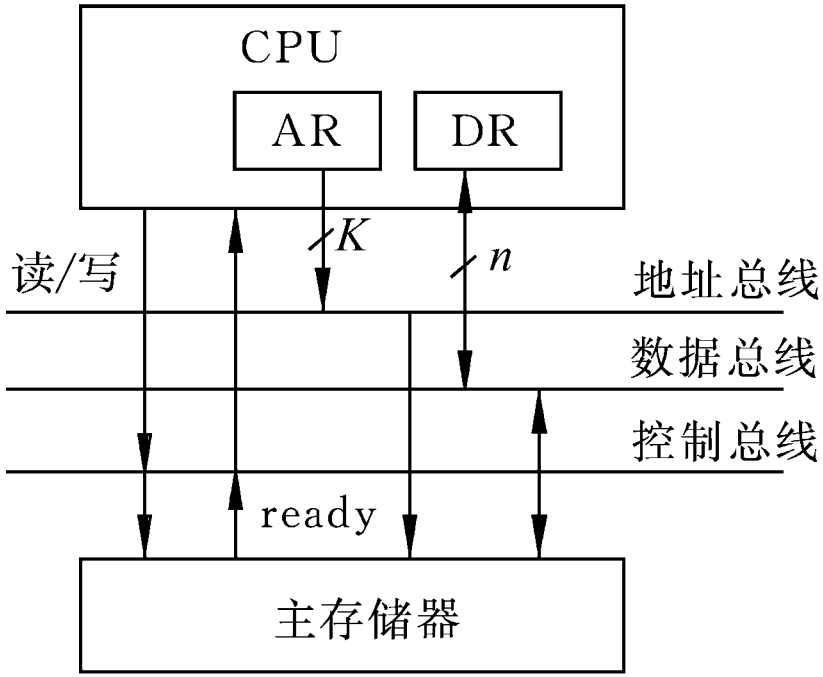
- **大端和小端方式**：采用多字节方式访问主存时，主存中的字节顺序非常重要，不同的顺序访问得到的数据完全不一样。
  - **小端(Little-Endian)方式**：存储器的低字节地址单元中存放的是数据的最低字节，比如访问2号字存储单元时，如果按小端方式访问得到的数据是0x78563412
  - **大端(Big-Endian)方式**：存储器的低字节地址单元中存放的是数据的最高字节，按照大端方式访问得到的数据则是0x12345678



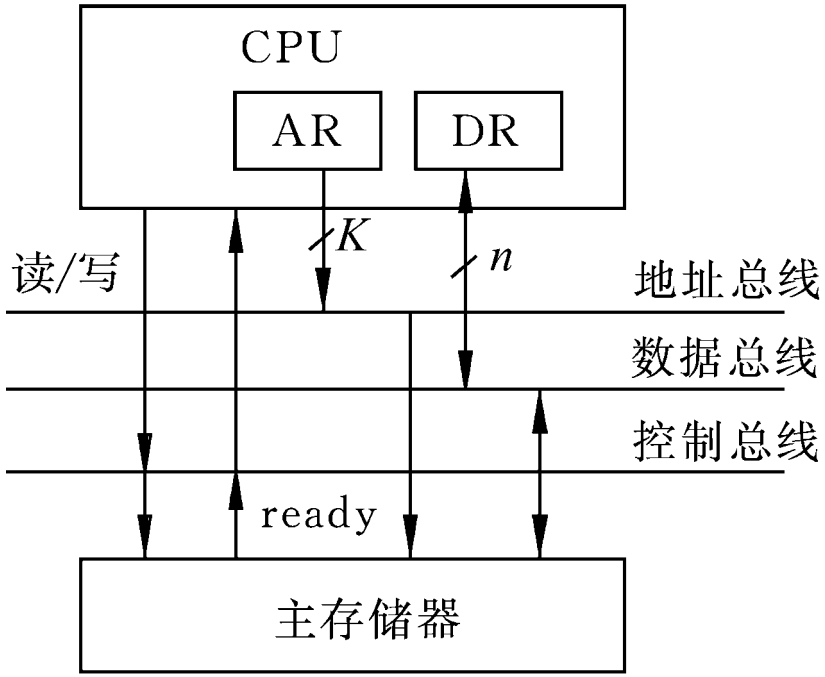
- 主存储器和CPU是由总线连接的。CPU通过使用AR（地址寄存器）和DR（数据寄存器）个主存储器进行数据传送。
- 在一个周期内，CPU和主存储器之间进行n位数据传送。



- 读操作：存储器→CPU
  - CPU把信息字的地址送到AR,经地址总线送往主存储器
  - CPU通过控制总线发读(Read)命令
  - CPU等待主存储器的Ready回答信号，Ready为 1，表示信息已读出经数据总线,送入DR。



- 写操作：CPU→存储器
  - CPU把信息字的地址送到AR，经地址总线送往主存储器,并将信息字送往DR
  - CPU通过控制总线发写(Write)命令
  - CPU等待主存储器的Ready回答信号，Ready为 1，表示信息已从DR经数据总线写入主存储器



# 本章主要内容

- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 半导体存储器

- 半导体存储器的主要特点
  - 存取速度快、体积小、性能可靠
  - 已成为实现主存的首选器件。
- 半导体存储器：
  - 随机存储器（Random Access Memory， RAM）
  - 只读存储器

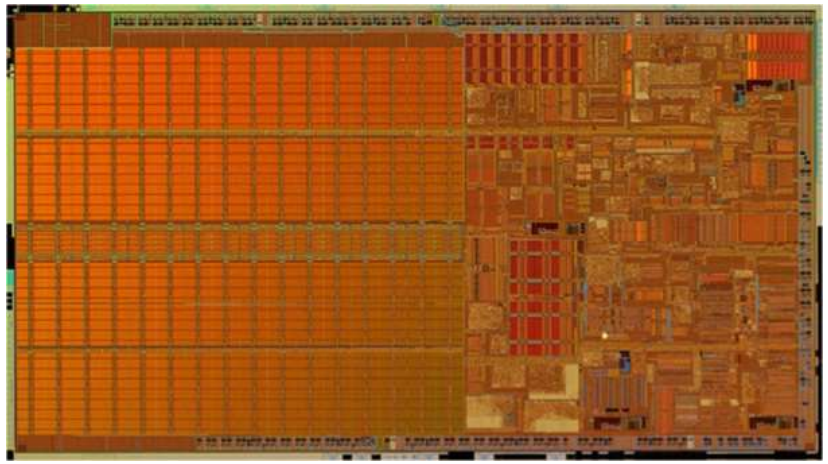
# 4.2 半导体存储器

- 4.2.1 静态MOS存储器
- 4.2.2 动态MOS存储器
- 4.2.3 只读存储器
- 4.2.4 DRAM的发展*

# 随机存取存储器（Random Access Memory）

- 静态MOS存储器
  - SRAM
- 动态MOS存储器
  - DRAM

# 半导体存储器如何存储数据?



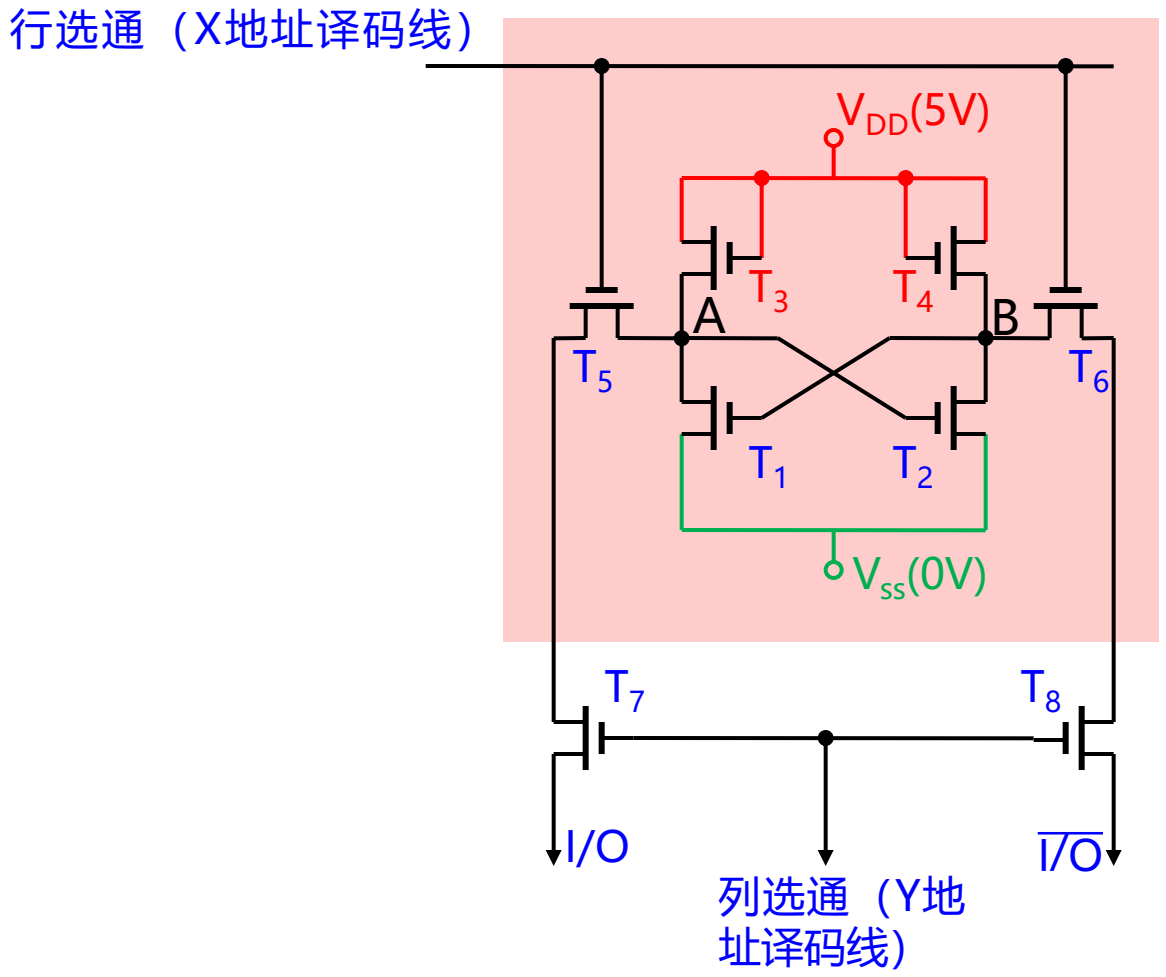
SRAM（CPU缓存）


DRAM 内存条

# 随机存取存储器（Random Access Memory）

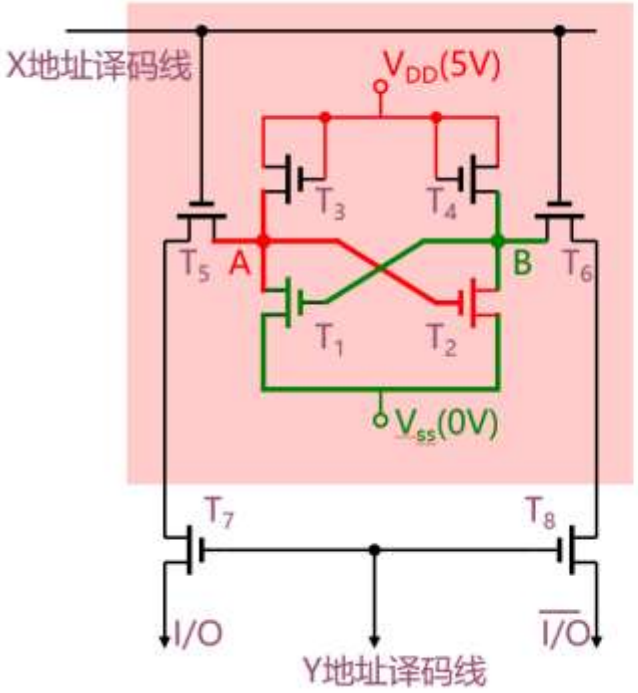
- 静态MOS存储器(SRAM)
  - 存储体以静态MOS存储元为基本单元组成的存储器
- 静态MOS存储单元
  - 存储单元是存储器中的最小存储单位，也称为存储元
  - 存储一位二进制信息。
- 静态MOS存储单元需具备以下基本功能
  - 具有两种稳定状态
  - 两种稳定状态经外部信号控制可以相互转换。
  - 经控制后能读出其中的信息。
  - 无外部原因，其中的信息能长期保存

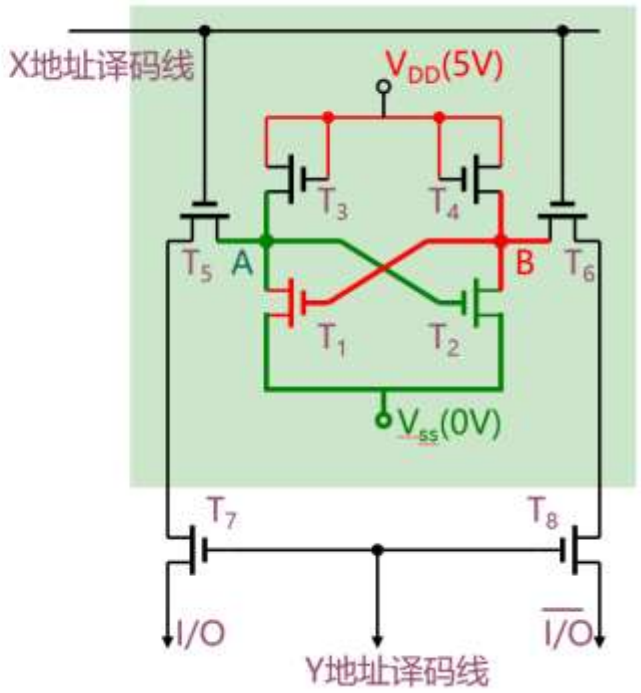
# 六管SRAM存储器 (SRAM Cell)




- 工作管$T_1$ $T_2$
  - 存储数据
- 负载管$T_3$ $T_4$
  - 补充电荷
- 门控管$T_5$ $T_6$ $T_7$ $T_8$
  - 开关作用
  - 控制存储元数据信息与外界的连通或隔离

# 六管SRAM存储器两种状态



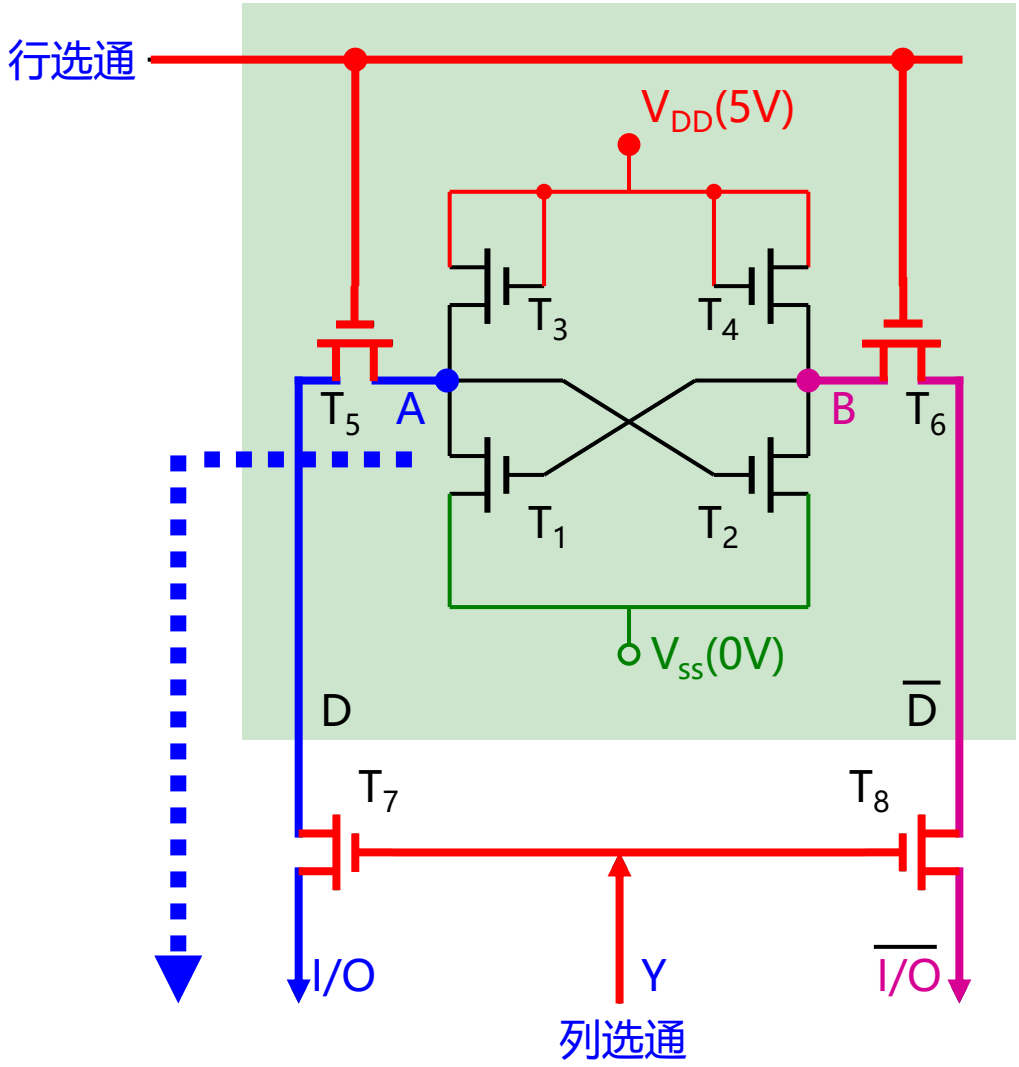




**耦合电路MOS管导通截止状态存储数据**

## ■ 电路状态

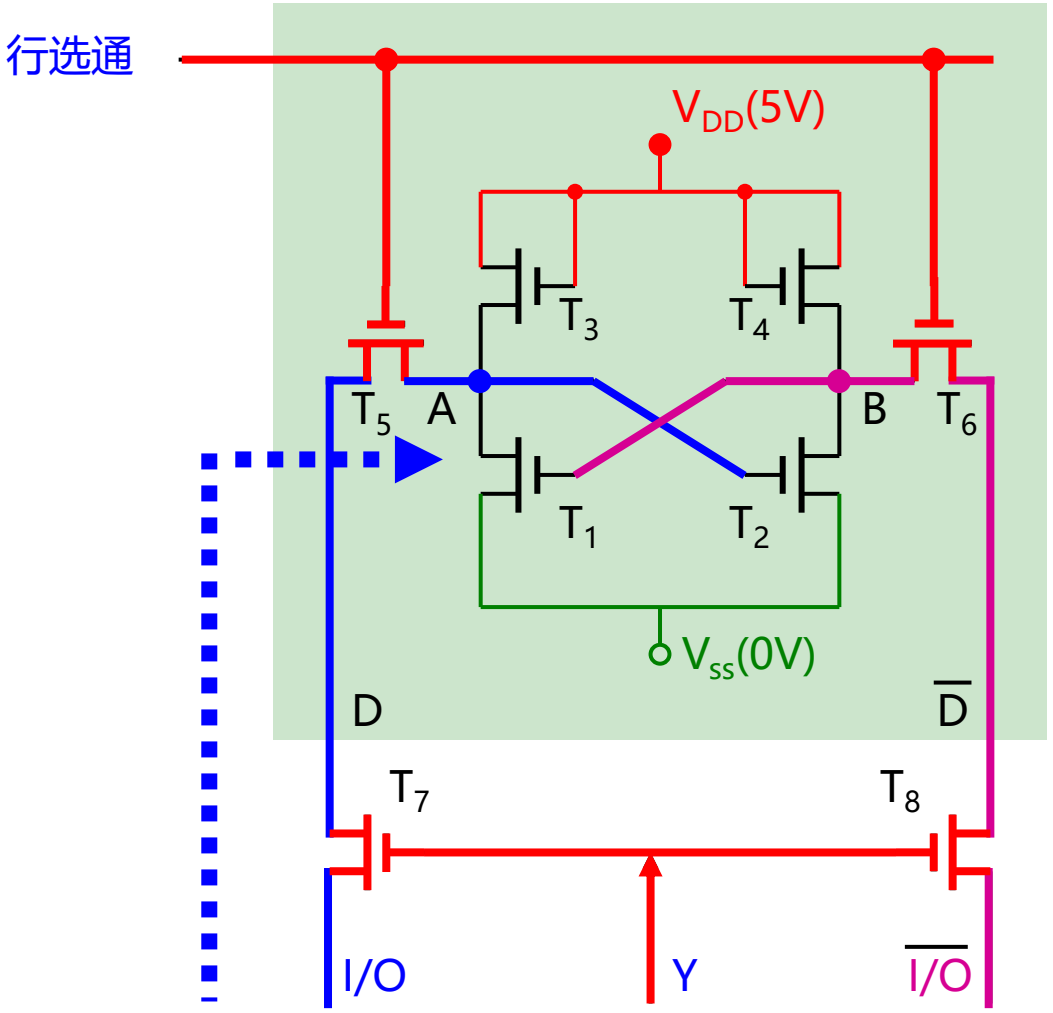
- A 点高电位使**T2管**导通
  - A 点高电位，栅极高电位则源级和漏极导通，低电位则截止
- B点与接地端导通，变成低电位
  - B点的低电位又使T1管截止A 点的高电位和漏极接地端隔离。
- T1、T2管一个截止、一个导通，
  - A点高电位，B 点低电位，形成一个稳定的状态，用来存储数据“1”
  - 当B点为高电位时A 点则为低电位，也可以构成另一个稳态，可用这个状态表示数据“0”
- A、B点电平信号相同时的状态是不稳定状态

# 六管SRAM存储器



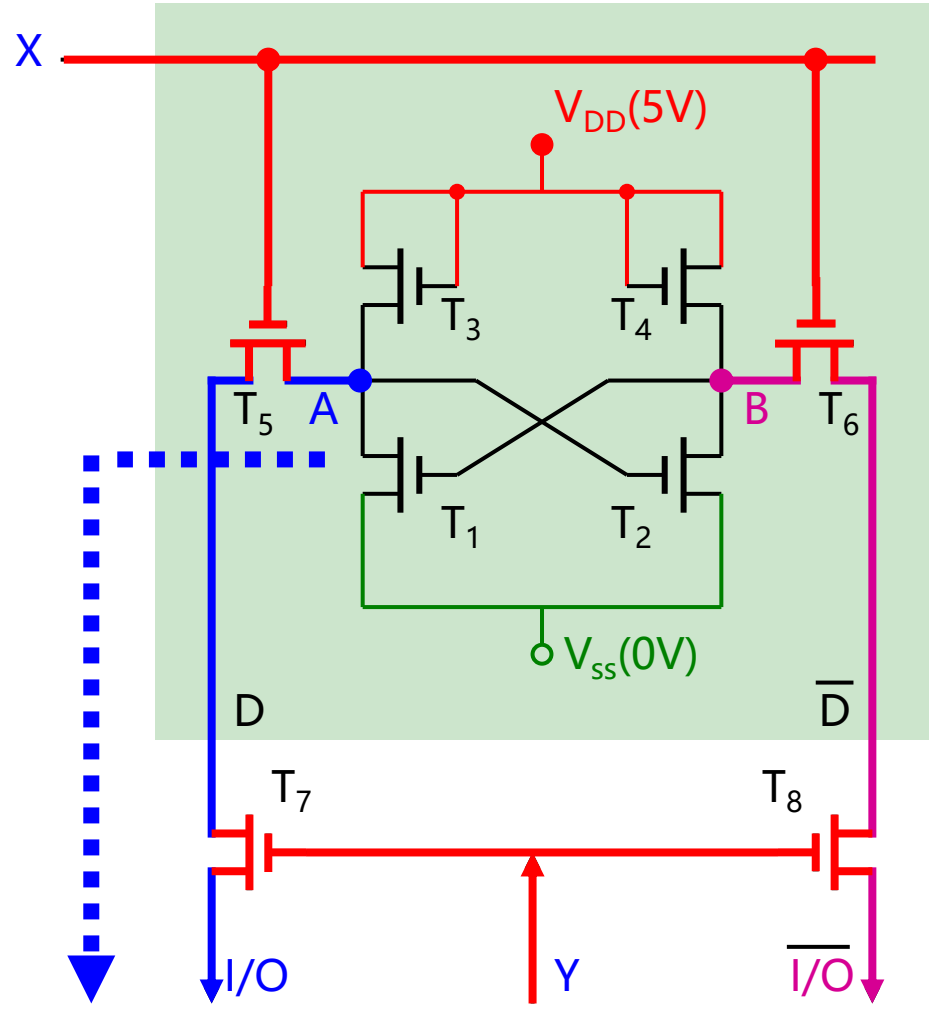

- 行选通
  - T5、T6管导通
  - A点与位线D相连
- 列选通
  - T7、T8管导通
  - A点电位输出到I/O端

# 六管SRAM存储器写操作



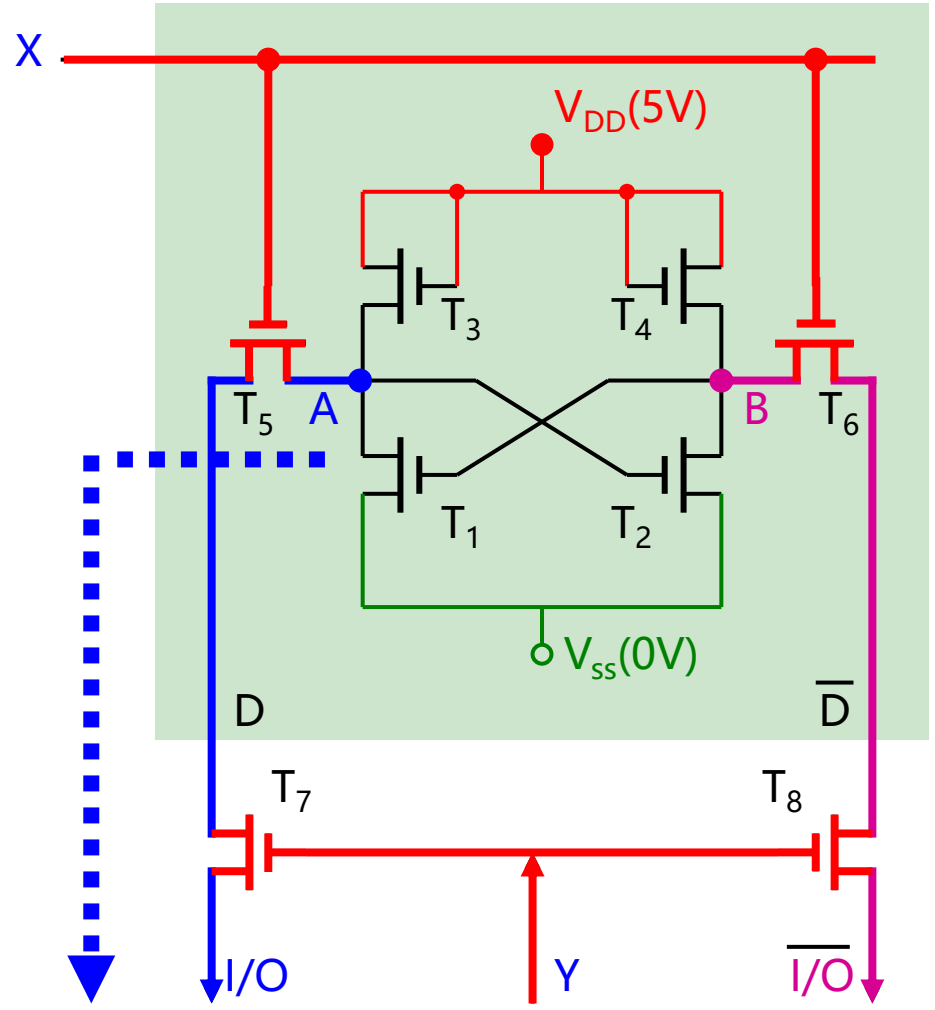

- 写操作：
  - 通过给出行选通、列选通的高电平信号打开4个门控管T5~T8
  - A 点与位线D相通，B点与位线D’相通
  - 将**D线加高电位**、同时给**D’线加低电位**，则可将A点和B点的电位变成1和0,从而写入数据“1”
  - 为D线加低电位、D线加高电位，则可写入数据“0”

# 六管SRAM存储器读操作



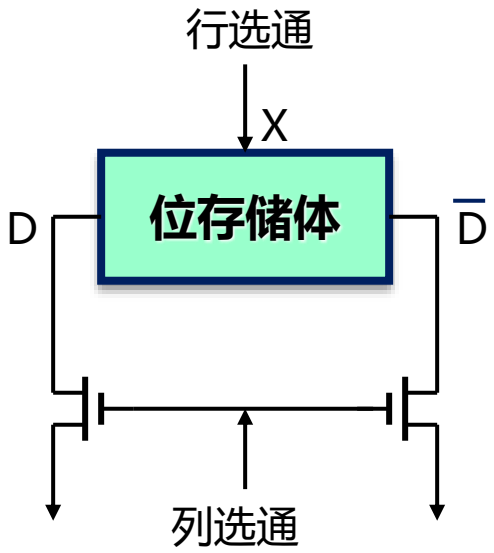

- 打开门控管T5~T8
- A点与位线D相通，B 点与位线D’相通
- 将两根位线的信号经差分放大器放大后，根据电流方向的不同输出不同的数据信息

# 六管SRAM存储器信息保持



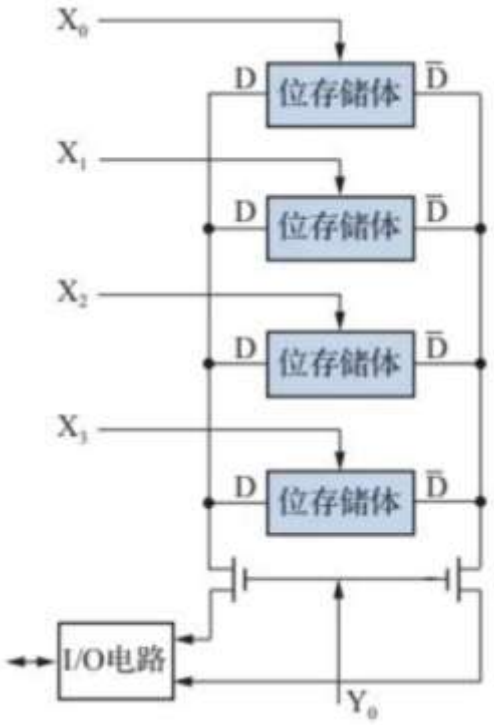

- 不给出行、列选通信号，门控管T5~8T截止
- 电源Vcc通过负载管 T3、T4,不断为T1 或T2提供电流，以保存信息
- 只要电源不断，存储单元的状态就一直保持不变，并且在读出时也不破坏原有数据。

# 位存储体封装与扩展



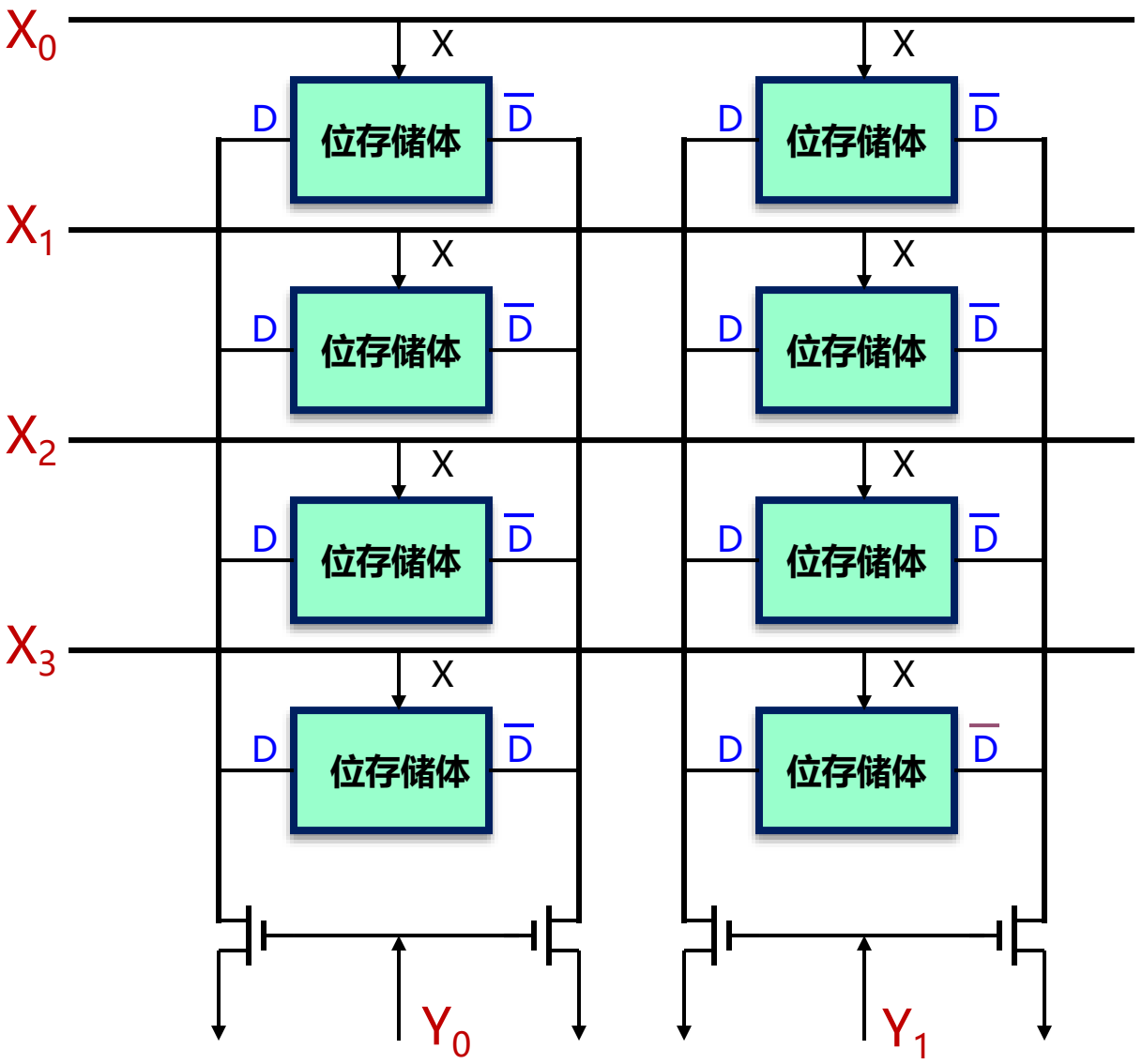

- 输入：X　行选择线
- 输出：D　数据输出口
- T1~T6管封装在一起构成一个6管SRAM存储单元
- 6管SRAM 存储单元只能保存一位数据
- 行选通信号即可将其内部存储的数据输出到位线D上

# 位存储体封装与扩展



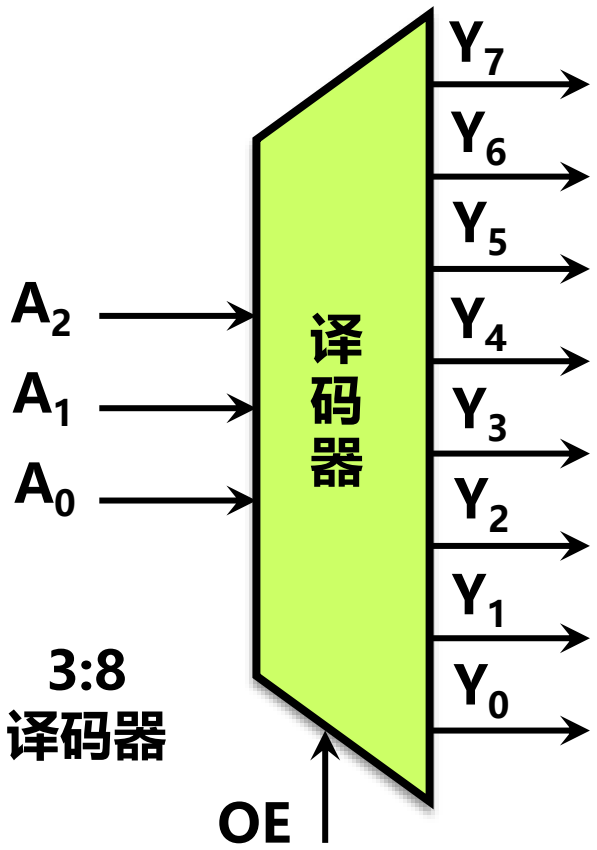

- 将4个位存储体的**位线并联**在一起，且各自连接不同的行选通地址 X0~X3，就可以得到一个**4位的存储单元**
- **同一时刻最多有一根行选通信号为高电平**，这样才能选中某一行存储单元，并输出到位线
- 当列选通信号Y0有效时，内部数据会与1/0电路连通，通过控制电路即可进行读取或写入操作

# 位存储体封装与扩展




- 行列两个方向都进行扩展
  - 所有列的位线都并联在一起
- 同一时刻列选通信号也只允许一根有效
  - 通过行、列地址可以选通其中一个位存储体进行访问

# 地址译码器




$$Y_0 = \bar{A}_2\bar{A}_1\bar{A}_0OE \qquad Y_4 = A_2\bar{A}_1\bar{A}_0OE$$

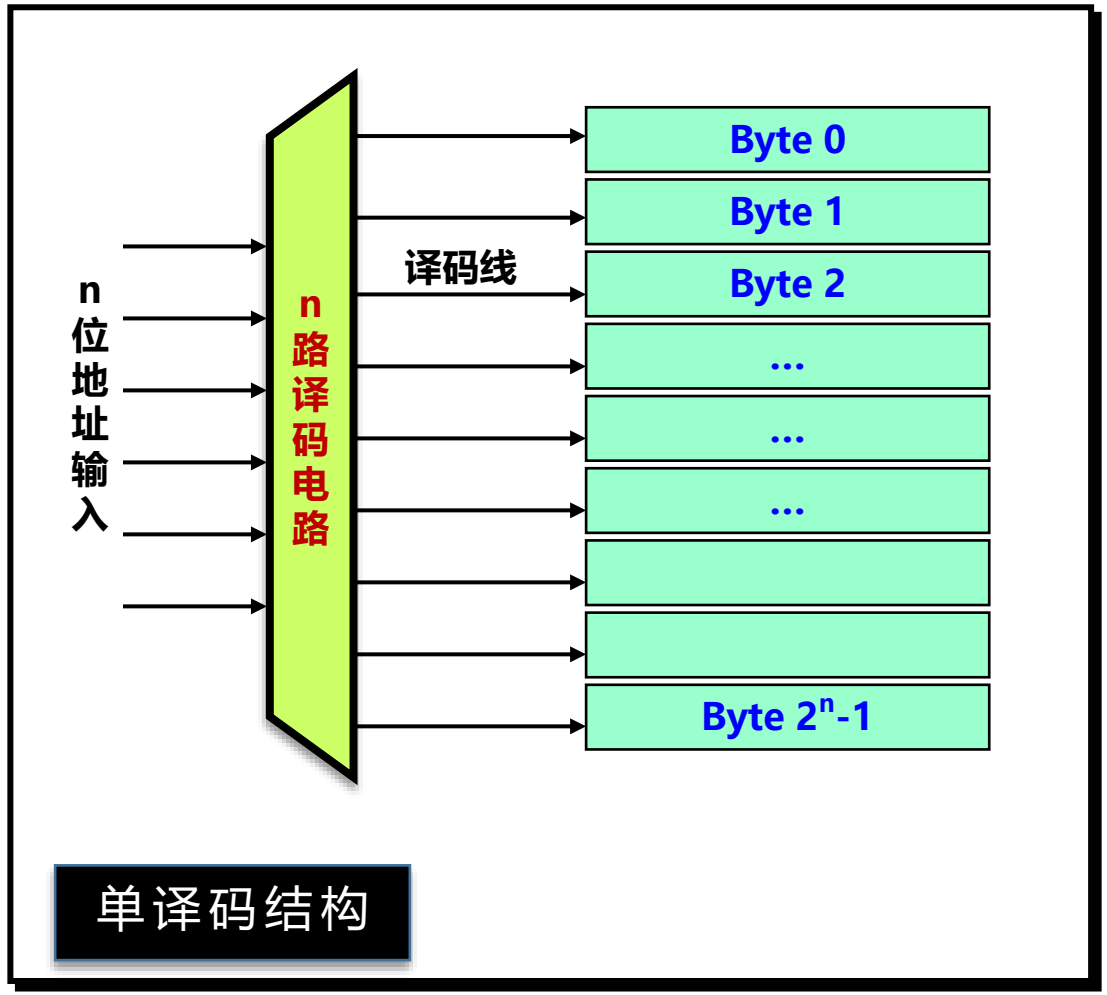
$$Y_1 = \bar{A}_2\bar{A}_1A_0OE \qquad Y_5 = A_2\bar{A}_1A_0OE$$

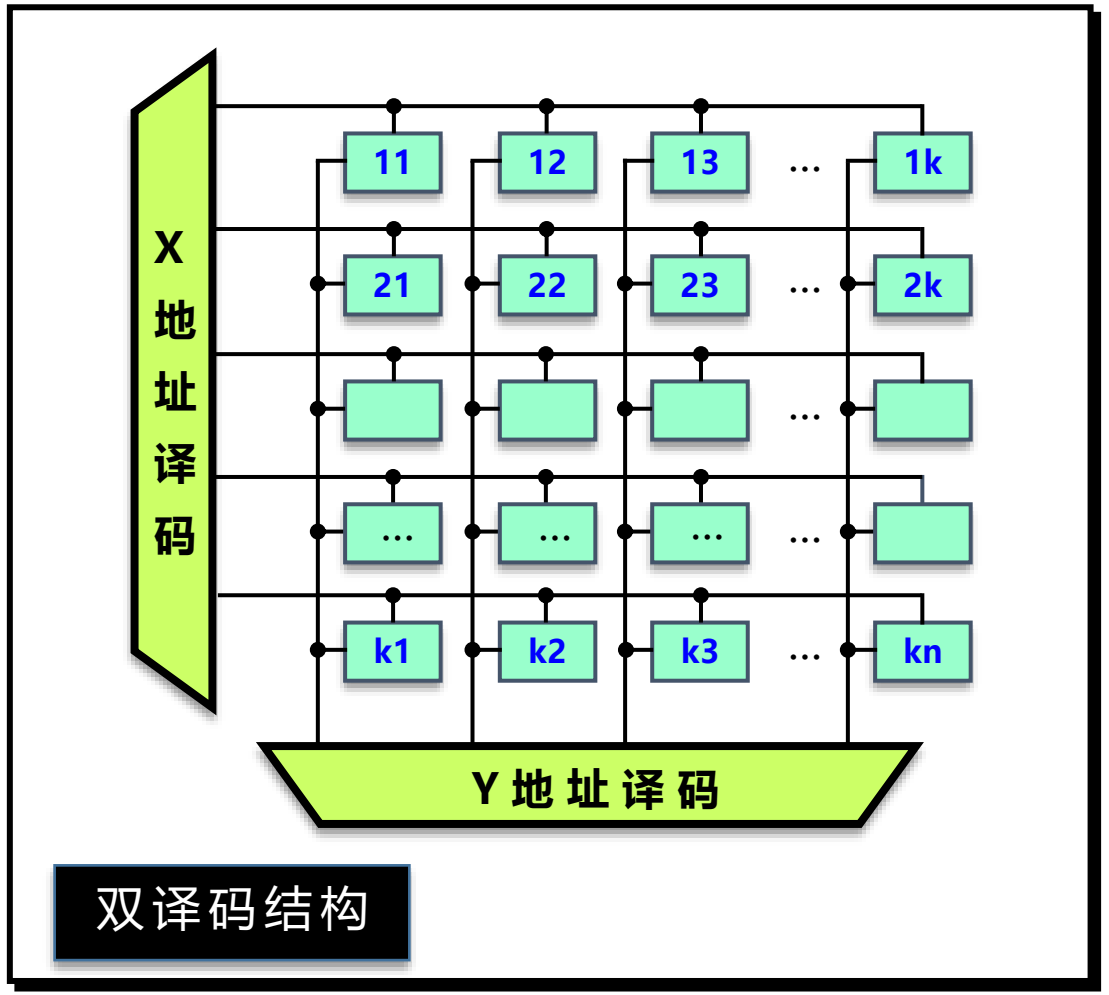
$$Y_2 = \bar{A}_2A_1\bar{A}_0OE \qquad Y_6 = A_2A_1\bar{A}_0OE$$

$$Y_3 = \bar{A}_2A_1A_0OE \qquad Y_7 = A_2A_1A_0OE$$

- **译码器**实现行、列选通信号的控制

# 译码方式



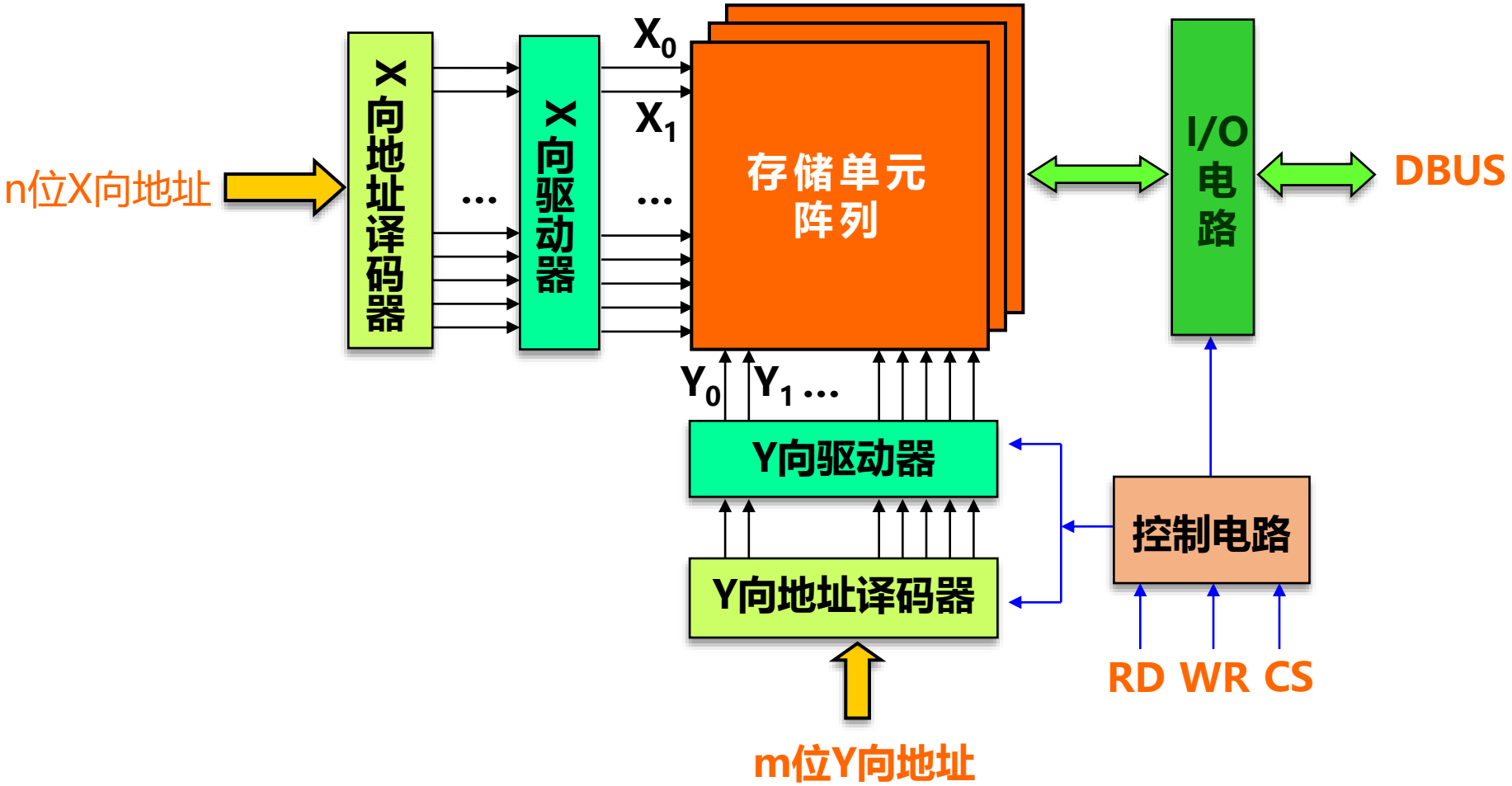




n位地址，寻址$2^n$个存储单元，$2^n$根译码线

n位地址，寻址$2^n$个存储单元，$2*2^{n/2}$根译码线

# 静态存储器芯片结构

■ 静态 MOS 存储器一般由**存储体**、**地址译码电路**、**I/0 电路**和**控制电路**等组成




4096x4位的静态 MOS 存储器结构框图

# 存储体与地址译码器

- 存储体
  - 4组存储阵列地址线并联构成
  - X、Y方向各6根地址线
  - 经过译码器后各产生64根选择线，存储矩阵为64x64位。
- 地址译码器
  - 将地址翻译成驱动存储单元门控管的行、列选通信号，并选中相应的存储单元

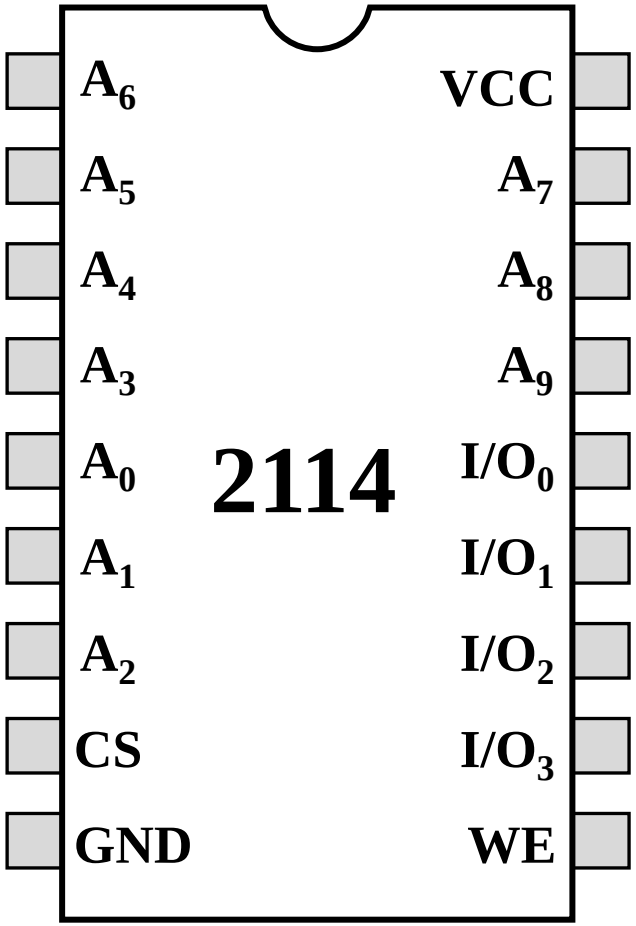
# 驱动器与I/O电路

- 驱动器
  - 一条选择线带很多存储位时负载过大
  - 在地址译码器输出端增加驱动电路
  - 保证每一个存储位都能正常工作
- I/O电路
  - 存储体与数据总线之间的电路
  - 读出时具有放大信号的作用

# 驱动器与I/O电路

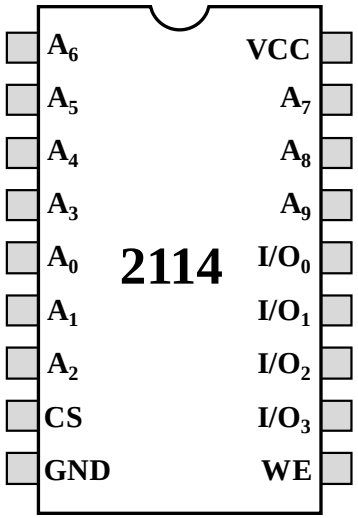
- 片选和读写控制电路
  - 一块集成芯片的容量有限，往往需要将多块芯片连接起来，组成一个大容量的存储器。
  - 存储器被访问时并不是所有的芯片都会同时工作，通过片选信号(CS)可以很好地解决存储芯片的选择问题。只有片选信号(CS)有效的存储芯片才能进行读或写操作。
  - 读或写操作通过CPU发出读写命令控制(WE)来实现。

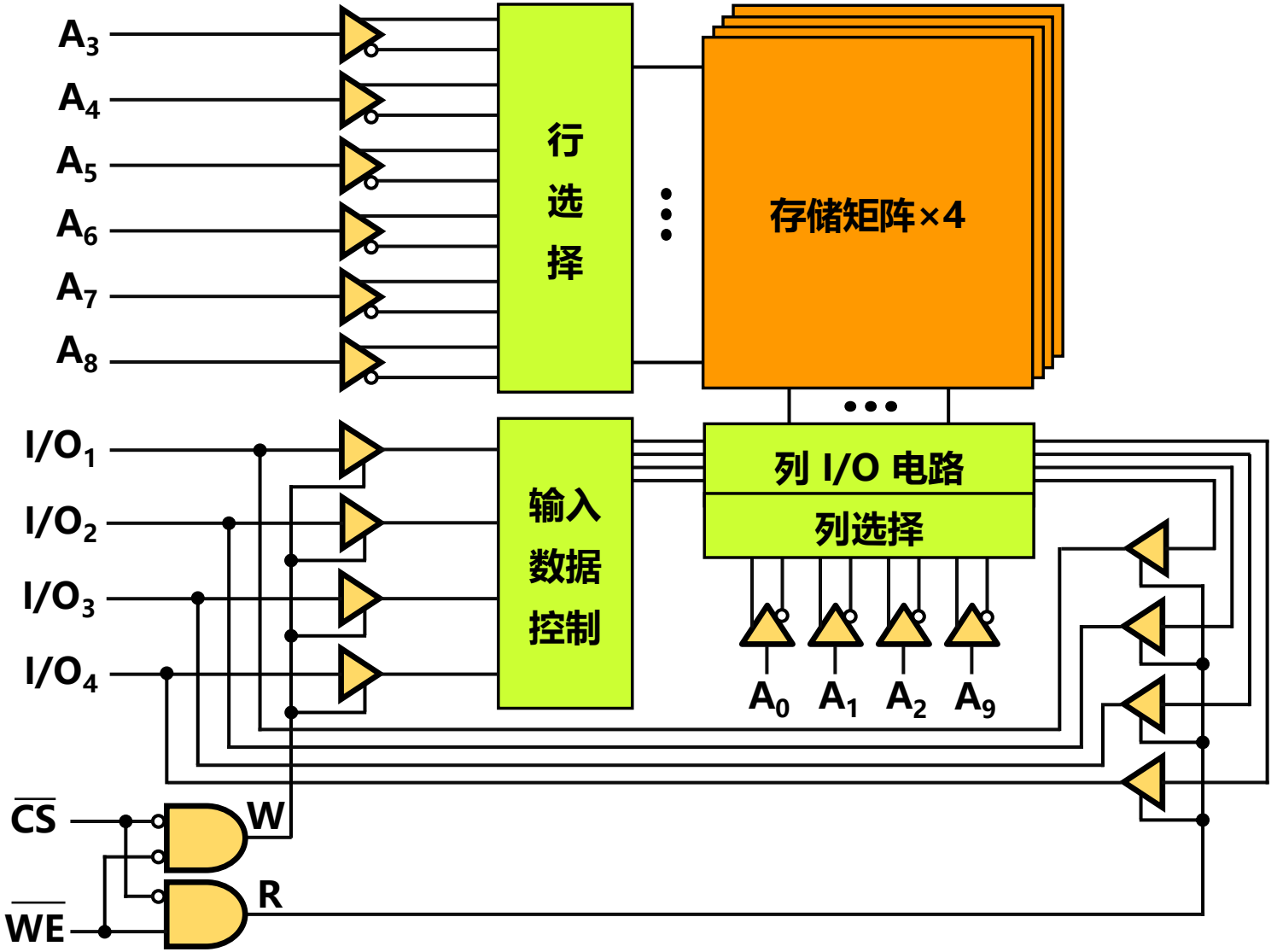
# 2114引脚图




- Intel 2114 是Intel 公司推出的一款1Kx4位的 SRAM 芯片
- 包括1024个字，字长4位
- 电源和接地引脚
- 地址输入引脚10个
- 数据双向输入输出引脚4个
- 片选信号CS和写使能信号WE两个引脚。

# 2114 SRAM内部结构



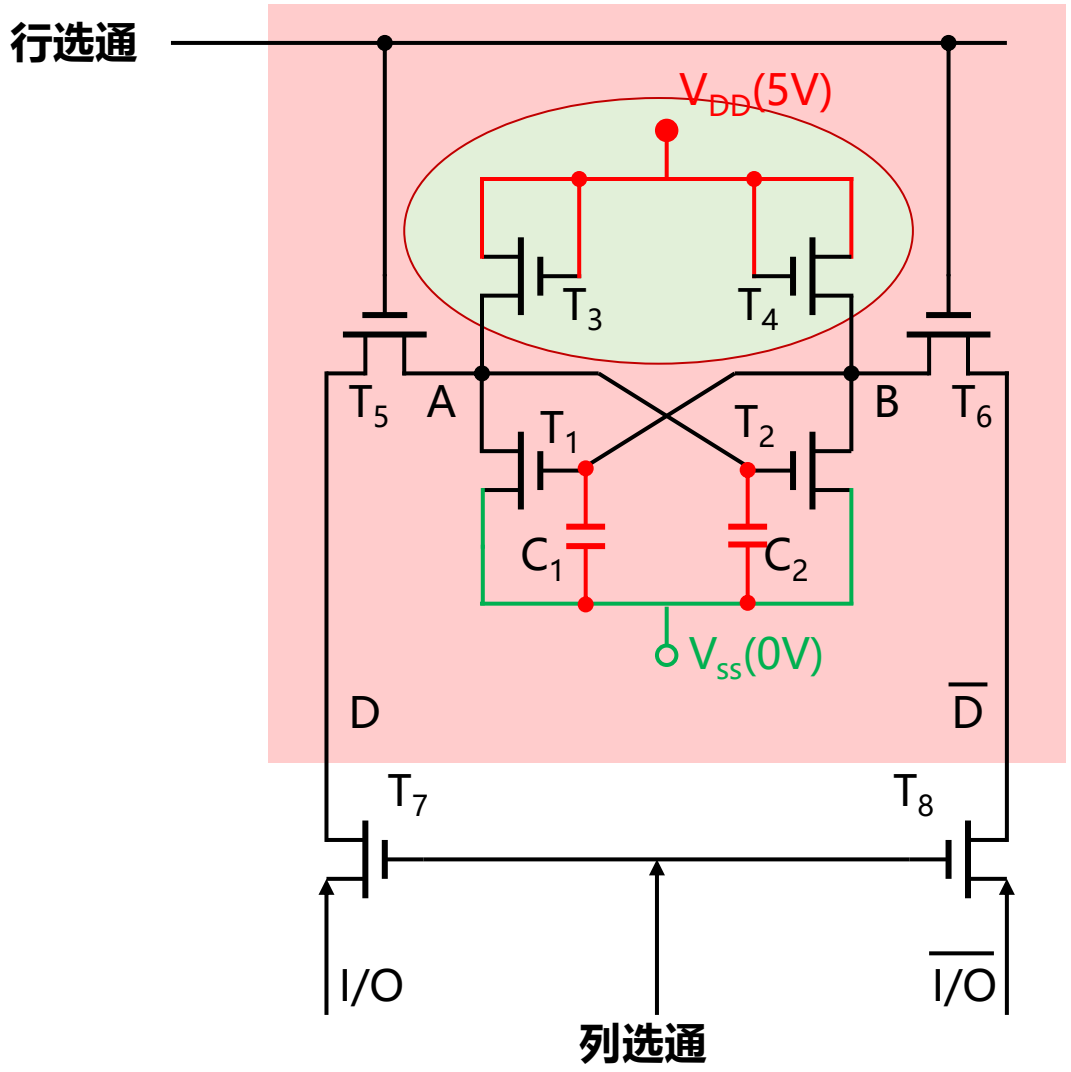
# 4.2 半导体存储器

- 4.2.1 静态MOS存储器
- 4.2.2 动态MOS存储器
- 4.2.3 只读存储器
- 4.2.4 DRAM的发展*

# 动态MOS存储器

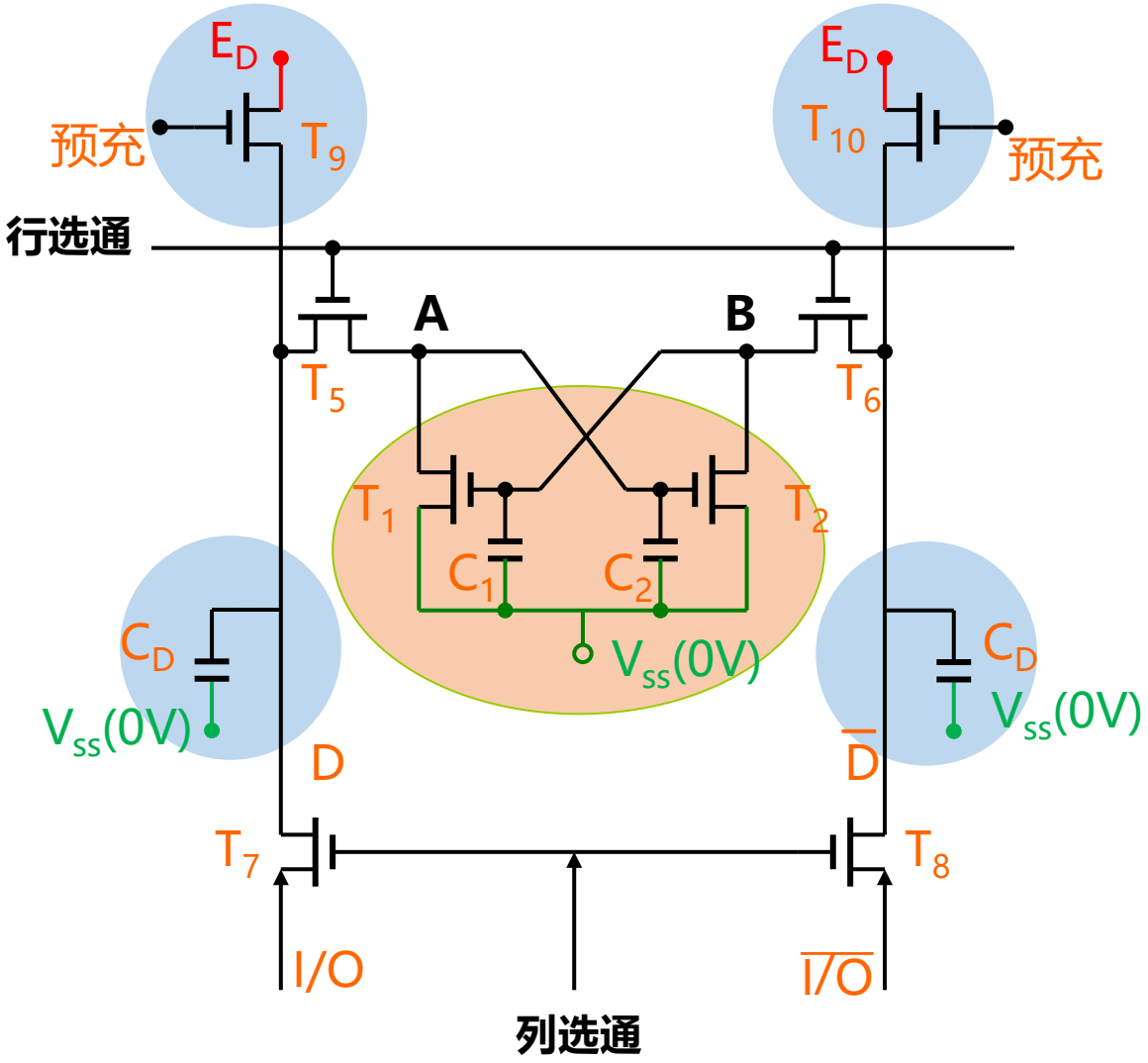
动态MOS存储器：存储体以**动态 MOS 存储元**为基本单元组成的存储器

# 如何提高存储密度



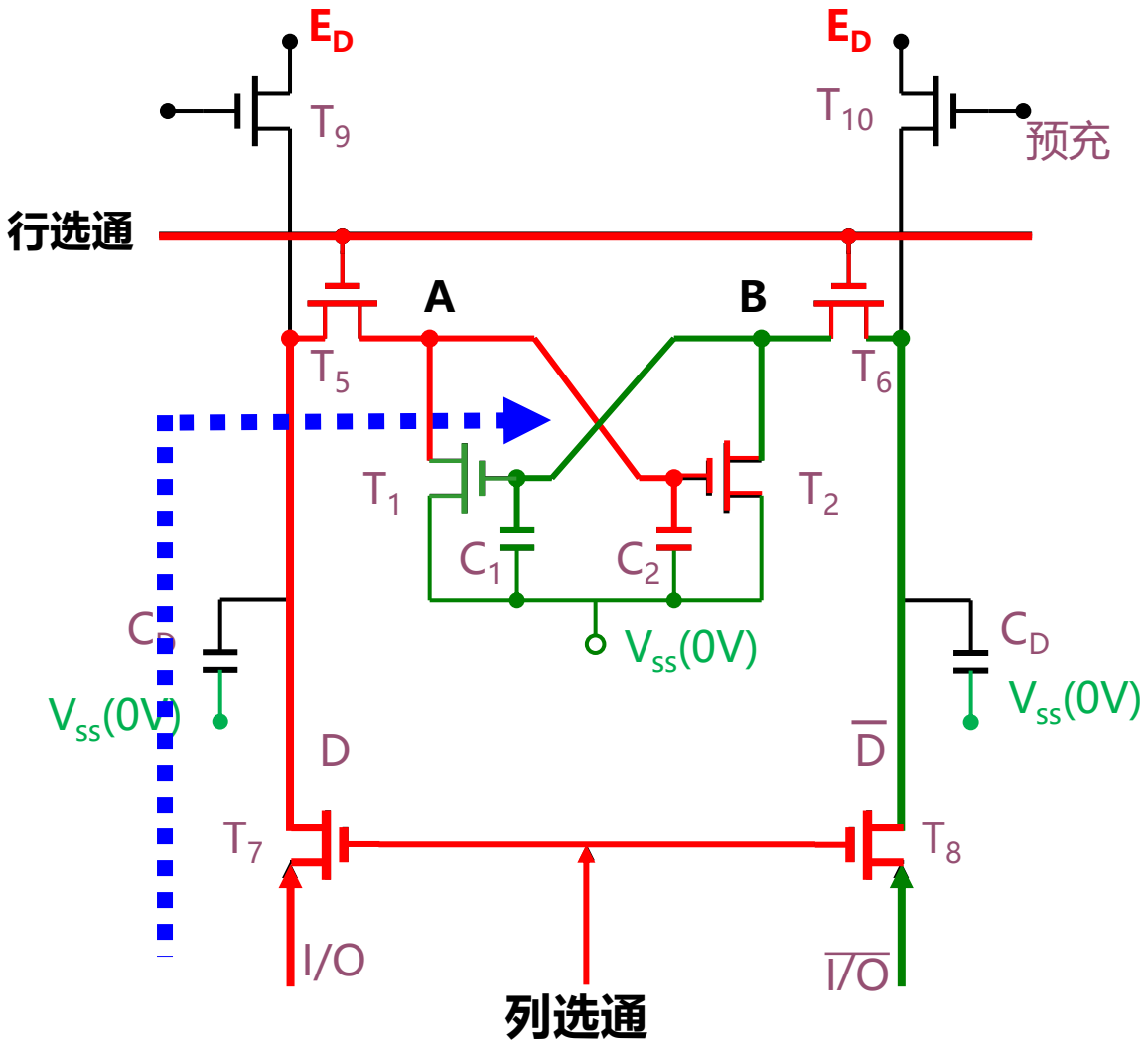

- SRAM
  - 6个MOS管才能存储1位数据，存储密度较低
  - 存储单元不进行读写，功耗管和导通的工作管之间也有电流存在，其功耗较大
- 去掉两个负载管
  - 提升存储密度
  - 减少功耗
  - 降低成本
- 增加两个电容缓存电荷
  - 电容不能永久保存电荷
  - 增加额外电路补充

# 四管DRAM存储器



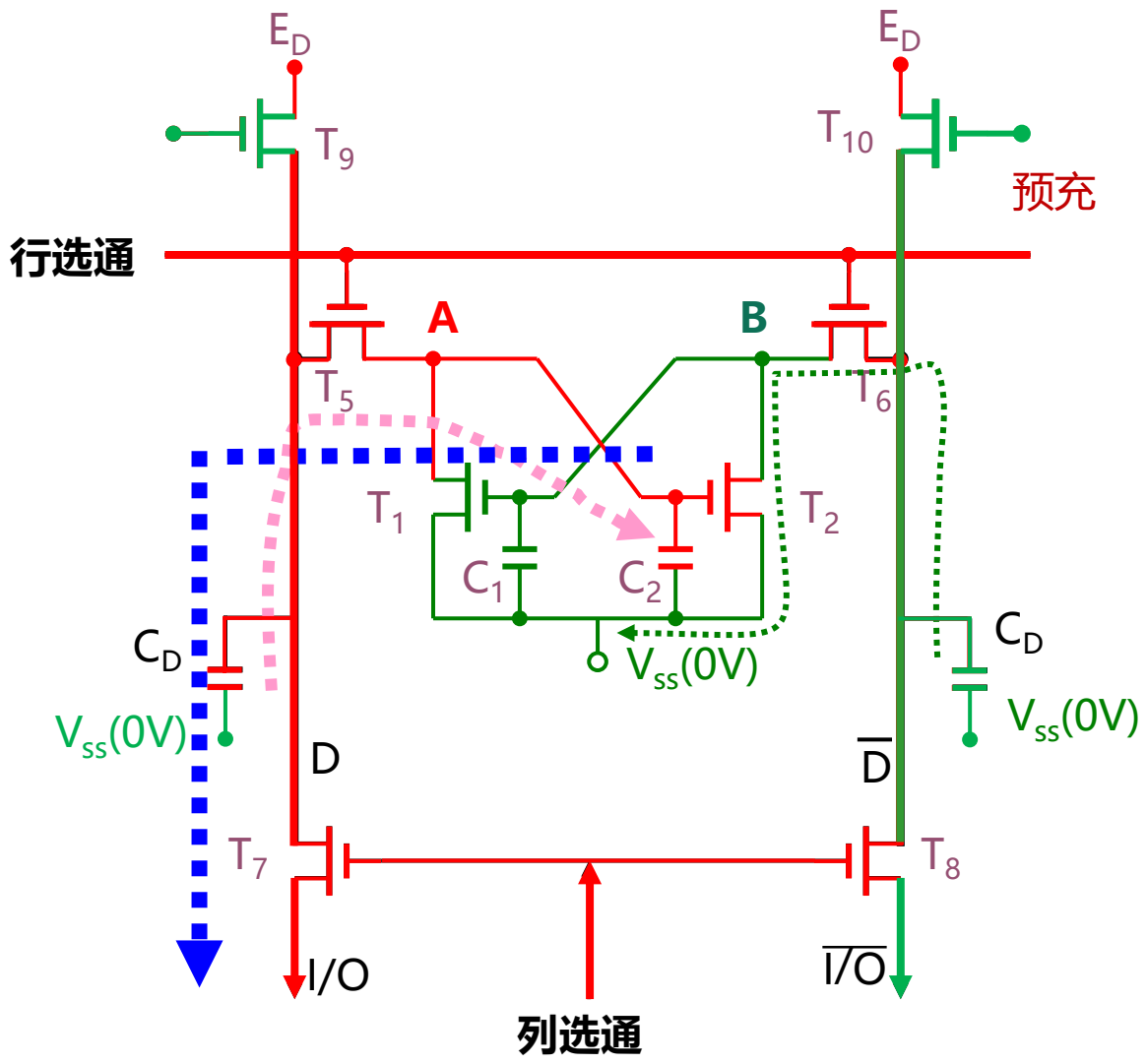

- **DRAM存储原理**
  - 利用电容电荷存储数据
  - 电容不能永久保存电荷
  - 必须增加额外电路补充

# 四管DRAM存储器写操作



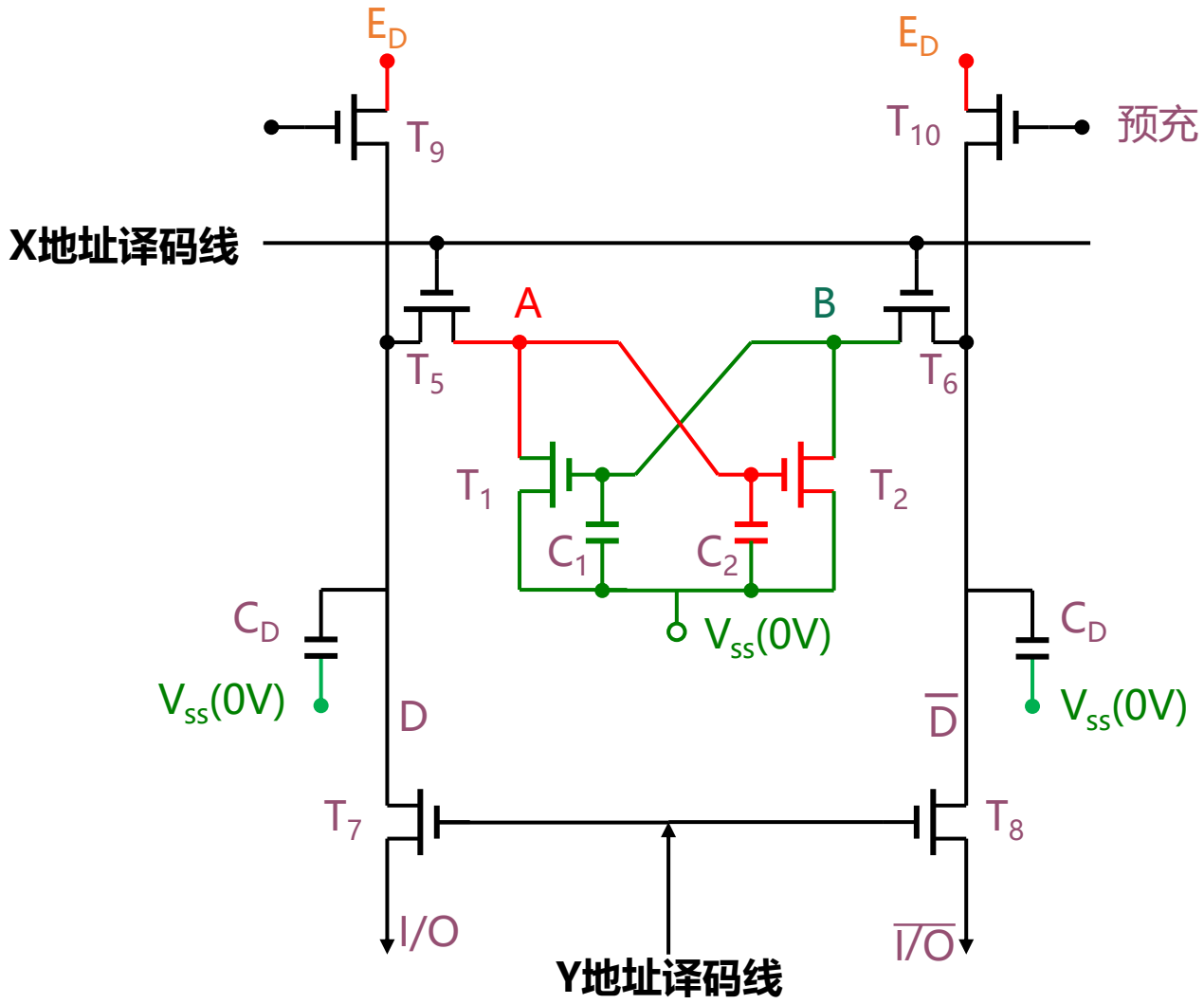

- 列选通
  - T7、T8管导通
  - I/O端数据写入到位线
- 行选通
  - T5、T6管导通
  - 位线相连与C2相连

# 四管DRAM存储器读操作



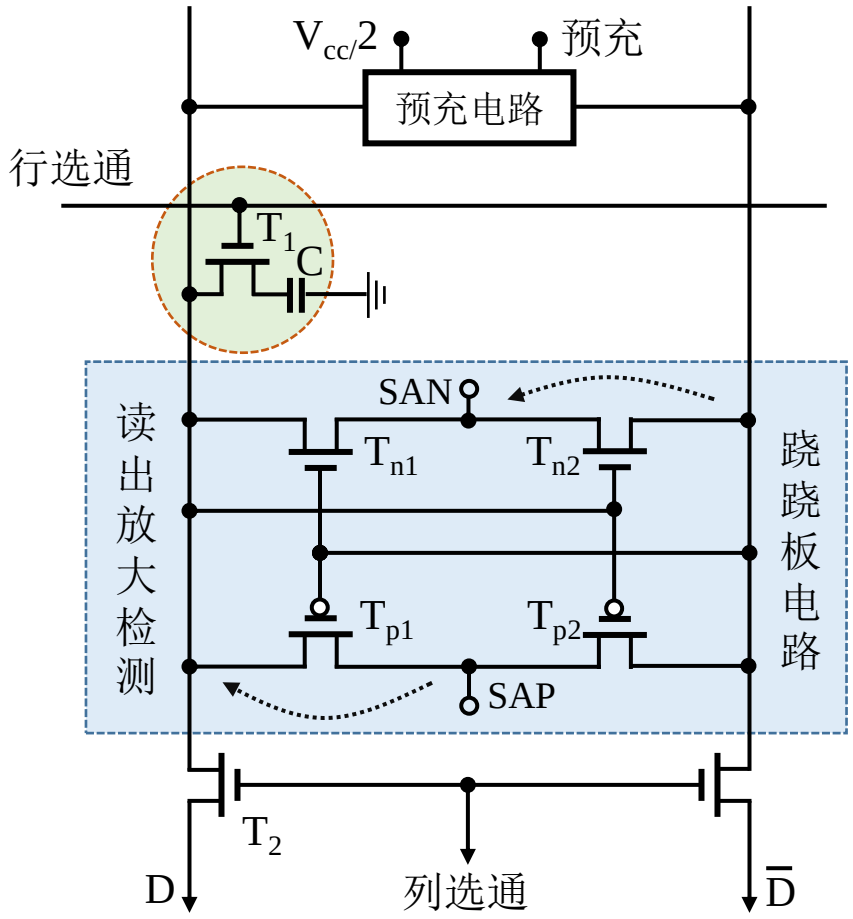

- **给出预充信号**
  - T9、T10导通
  - 充电电压给$C_D$充电
- **撤除预充信号**
- **行选通**
  - T5、T6管导通
  - 位线相连与$C_2$相连
  - $C_D$给$C_2$充电
  - 补充电荷
- **列选通**
  - T7、T8管导通
  - $C_2$数据读出到I/O
- **读过程比写复杂、速度慢**

# 进一步提高存储密度



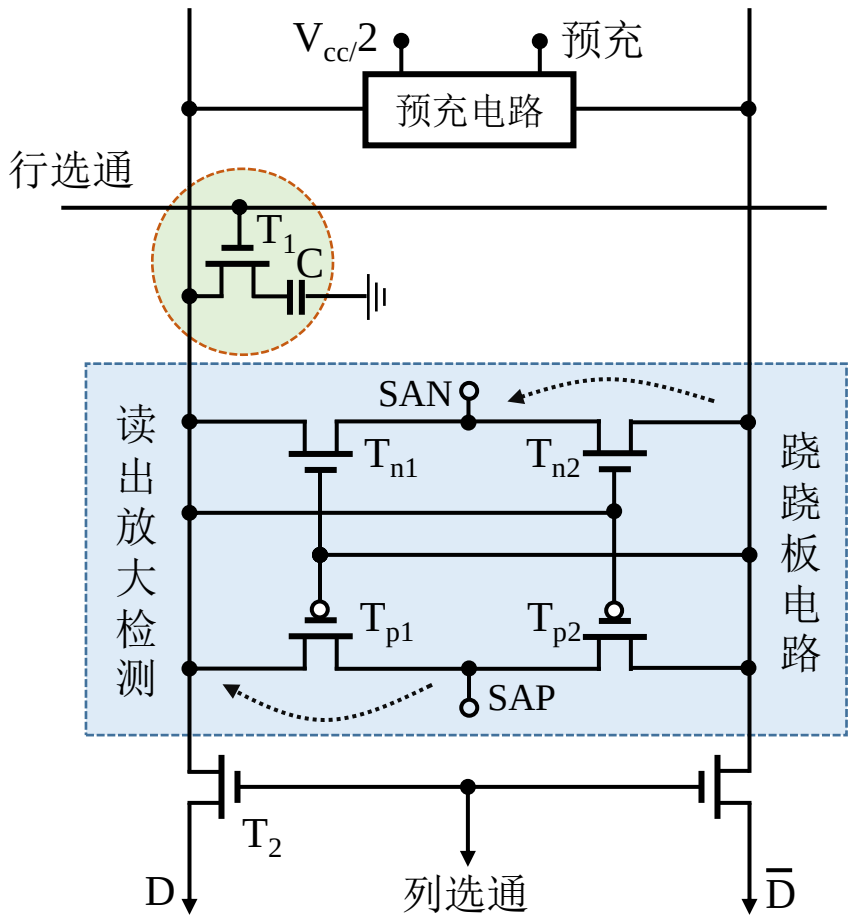

- 进一步提高存储密度
  - 核心是电容
  - 裁剪冗余电路

# 单管DRAM存储



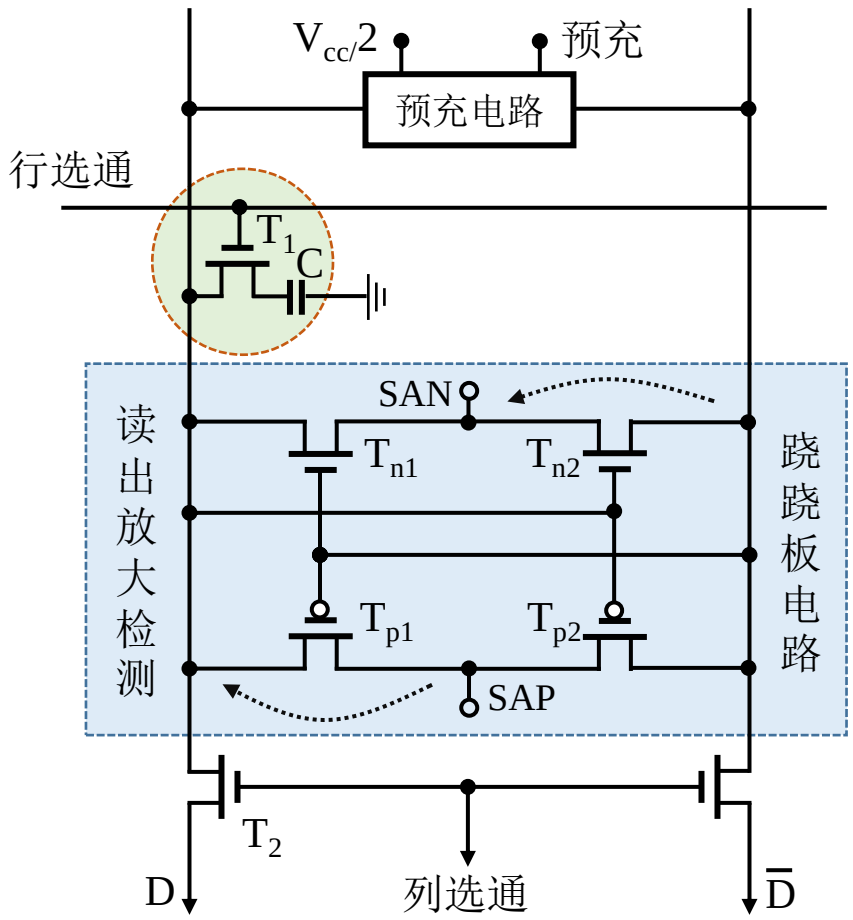

- 单管动态 MOS 存储单元
  - 一个 MOS 管
  - 一个电容
- 目前内存中基本存储单元的结构
- 利用存储电容C是否带电荷来表示数据
  - 有电荷表示数据“1”无电荷表示数据“0”
  - 读出时给出行、列选通信号使得T1、T2导通
  - 存储电容C上的电荷会与位线上的寄生电容进行电荷重分配，形成微弱的电流
  - 再由读出放大器根据是否存在电流输出数据“1”或“0”
  - 写入时通过位线上的高电平对电容C进行充电写入“1”，低电平对电容放电写入“0”

# 单管DRAM存储的工作原理




- 预充操作 (Precharge)
  - 预充电路将位线D和D’预充到 Vc/2的电压后撤除预充信号，此时位线上的寄生电容保持Vcc/2的电压;
  - 目的是加速读取的过程
- 访问操作 (Access)
  - 行选通，$T_1$管导通
  - 存储电容和位线寄生电容电荷重分配，假设存储电容上有电荷，存储数据“1”则位线D上的电压将略大于Vcc/2;反之如果存储电容上没有电荷，则位线D上的电压将略低于 Vcc/2。

# 单管DRAM存储的工作原理



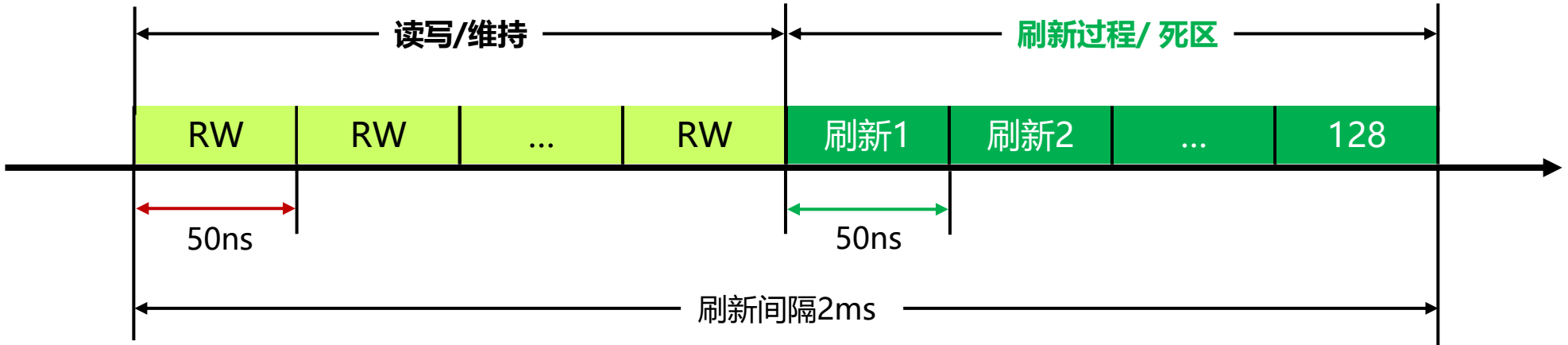

- 信号检测 (Sense)
  - 电压略高的一侧拉升到逻辑1，另一侧为0
- 数据恢复 (Restore)
  - 如数据为1，位线上的逻辑1给存储电容进行充电
- 数据输出(Output)
  - 给出列选通信号，数据输出到外部。
  - 行列选通信号分时给出，行列地址复用减少引脚
  - 撤除行选通信号，关闭读出放大检测电路

# DRAM 刷新

- **刷新：**信息以电荷形式存储在工作管的栅极电容中，但电容容量较小，所存电荷会在一段时间后逐渐泄漏。为使所存信息能长期保存，需要在电容中电荷泄漏完之前定时地补充电荷
- **刷新周期：**存储器两次完整刷新之间的时间间隔
  - 信息存储到泄漏之间必须完成刷新，称为**最大刷新周期**
- **按行刷新**
  - 存储体采用双译码结构，刷新地址计数器给出刷新行地址
- **刷新方式**
  - CPU与刷新控制器对DRAM的争用问题
  - 集中式、分散式、异步式

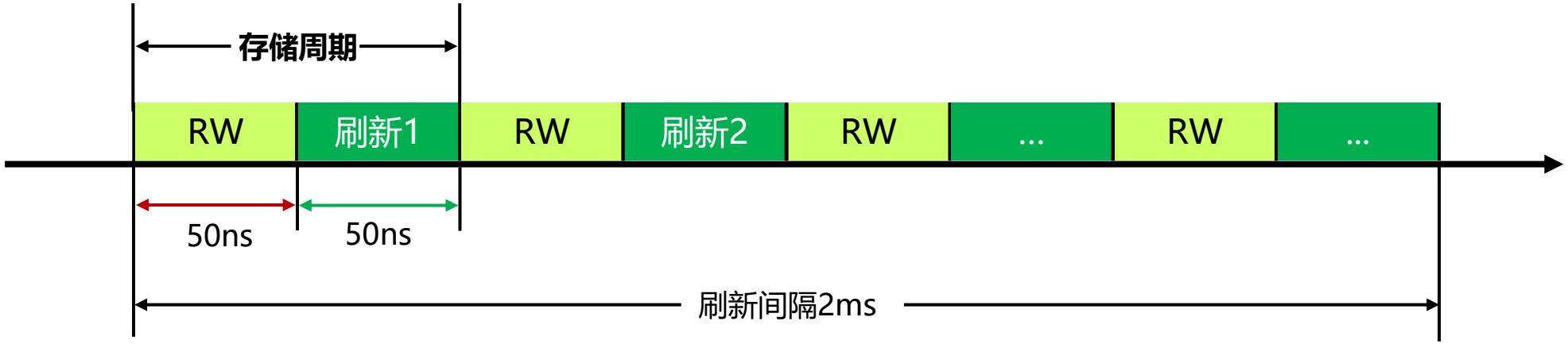
# 集中刷新方式

- 最大刷新周期：2ms
- 在数据丢失之前集中刷新所有行，2ms内
  - 进行 4000次读写或保持操在集中刷新方式下
  - 前3872个读写周期都用来进行读写或保持
  - 最后128个读写周期集中用于刷新
- 存在较长时间的“死区”
  - 即在集中刷新的128 个读写周期内，CPU长时间不能访问存储器
  - 用在**实时要求不高**的场合



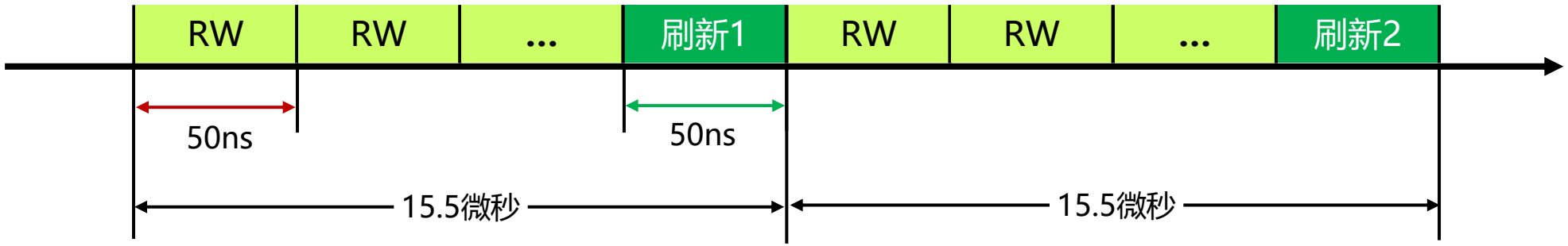
# 分散刷新方式

- 最大刷新周期：2ms
- **存储周期**：读写+刷新　各刷新周期**分散**安排在存取周期中
- 刷新次数 2ms/100ns=20000次，不存在“死区”
- 刷新过于频繁，严重影响了系统的速度，故不适合应用于高速存储器。

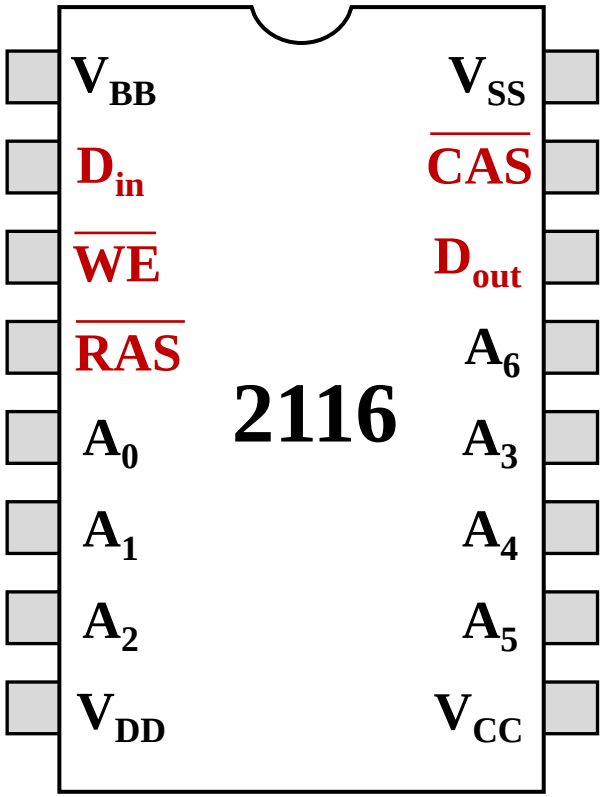


# 异步刷新方式

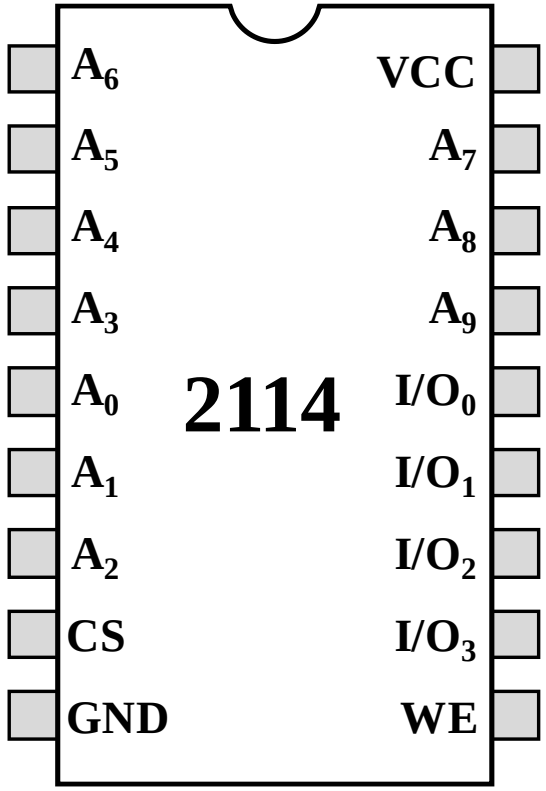
- 刷新周期：2ms，各刷新周期分散安排在2ms内
- 每隔2ms/128=15.5微秒刷新一行,将128次刷新分散
- **最常用**



# 2116引脚图




- 地址线
- 数据线
- 读写控制线
- RAS　CAS
- 电源线
- 地线



# 4.2 半导体存储器

- 4.2.1 静态MOS存储器
- 4.2.2 动态MOS存储器
- **4.2.3 只读存储器**
- 4.2.4 DRAM的发展*

# 只读存储器(ROM)

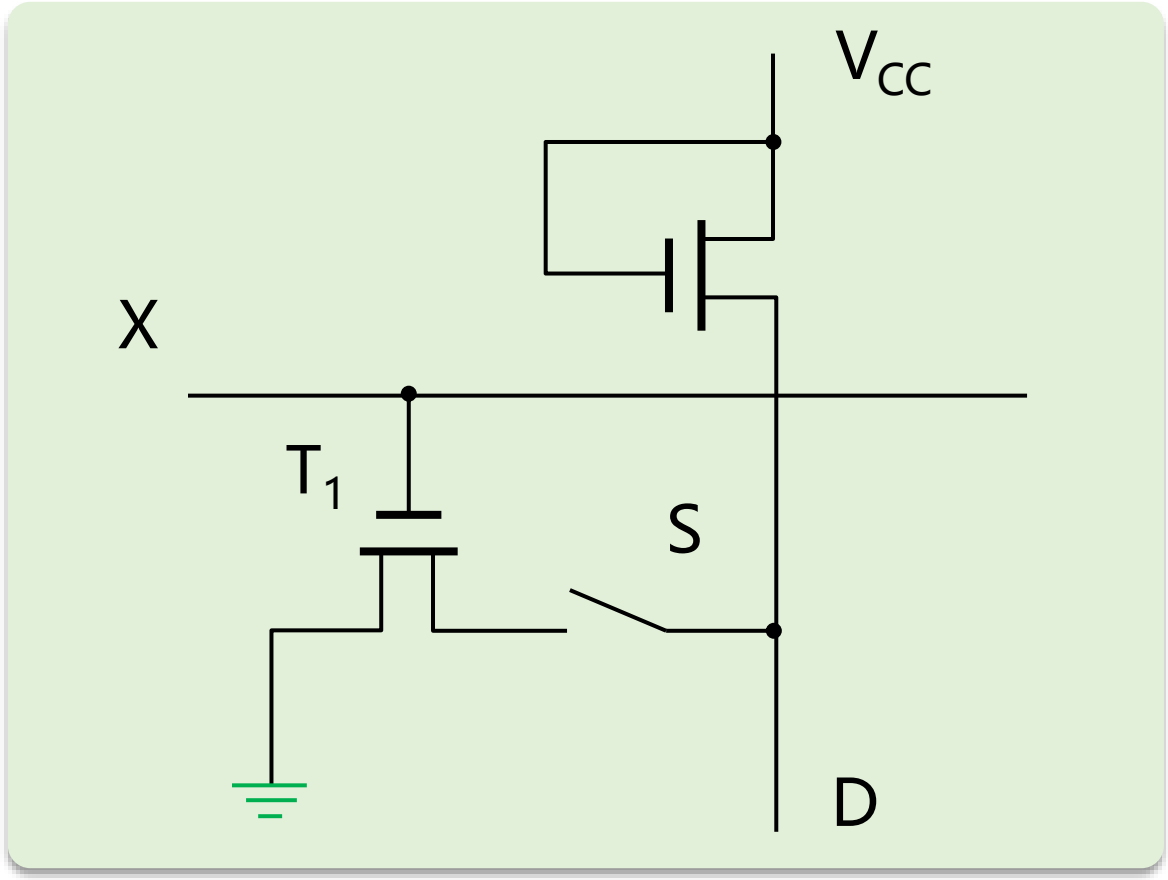
- 信息只能读出、不能随意写入的存储器称为只读存储器，记为ROM
- 将信息写入之后，信息就固定在其中
- 非易失性，即使电源断电，保存的信息也不会丢失
- 用来存放一些不需要修改的程序
  - 微程序
  - 子程序
  - 某些系统软件
  - 用户软件

# 只读存储器(ROM)

- 掩模式只读存储器
- 可编程只读存储器（PROM）
- 可擦除可编程只读存储器（EPROM, EEPROM）
- 电可擦除可编程只读存储器（EEPROM）
- 闪存（Flash）

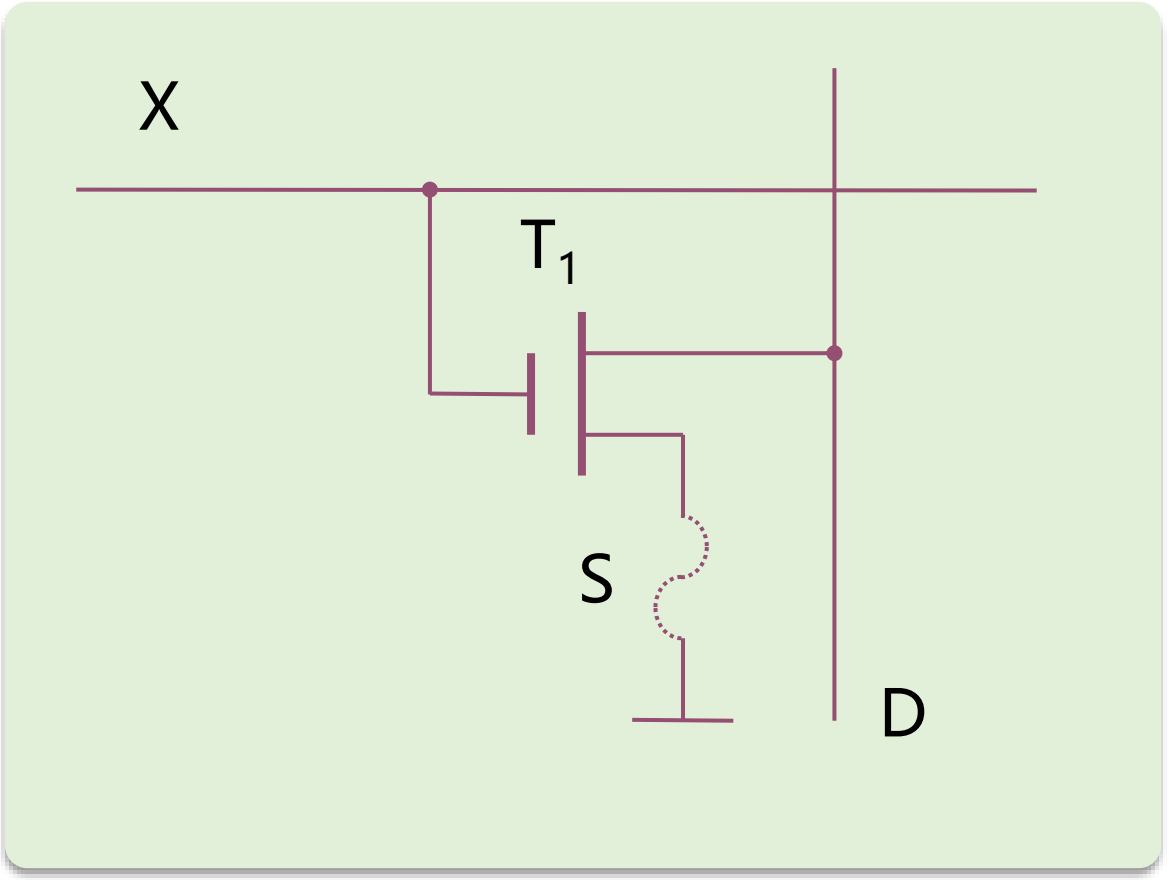
# 掩模式只读存储器

- 存储单元
  - MOS 管T1
  - 开关S
- X为行选通线，D为数据线，Vcc为电源
- 另一个 MOS 管为共享的功耗管，起电阻的作用。
- 当行选通时，
  - T1管导通，如果S是断开的，则位线D上将输出信息1;
  - 如果S是闭合的，位线D与接地端相连，则输出信息0。

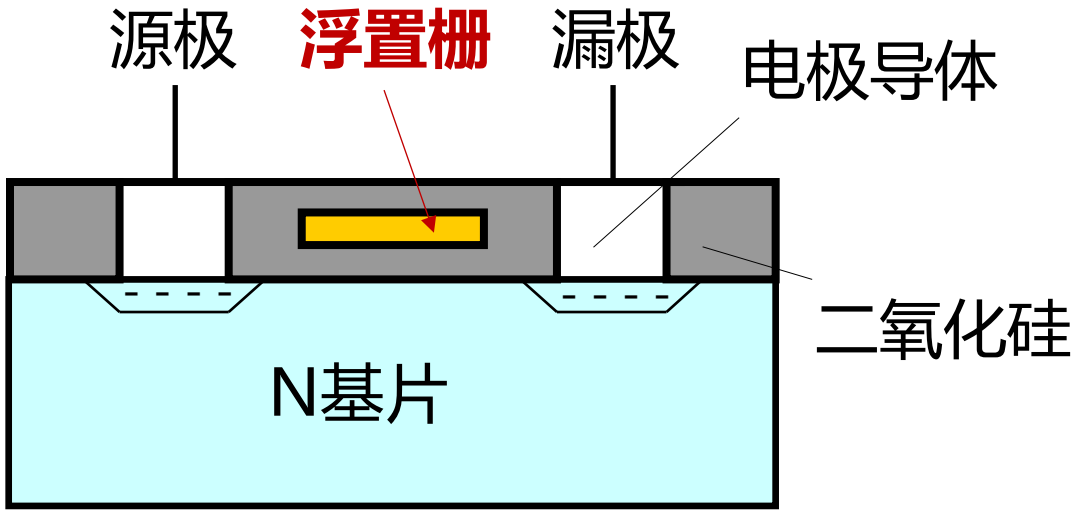


# 可编程只读存储器

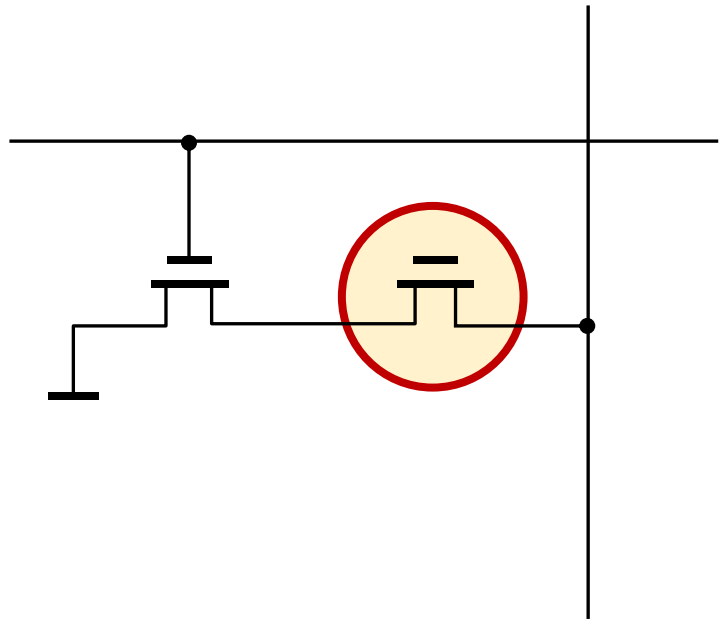
- 一种用户可**写一次**的ROM
- 采用熔丝代替了开关S
- 该存储器出厂时每个存储单元的熔丝都是连接状态，存储数据为全“0”。
- PROM 可利用编程器进行一次改写,具体写入时可以通过辅助电路有选择性地将某些存储单元的熔丝高压熔断再写入数据“1”



# 可擦写ROM——EPROM




(a) 单元结构

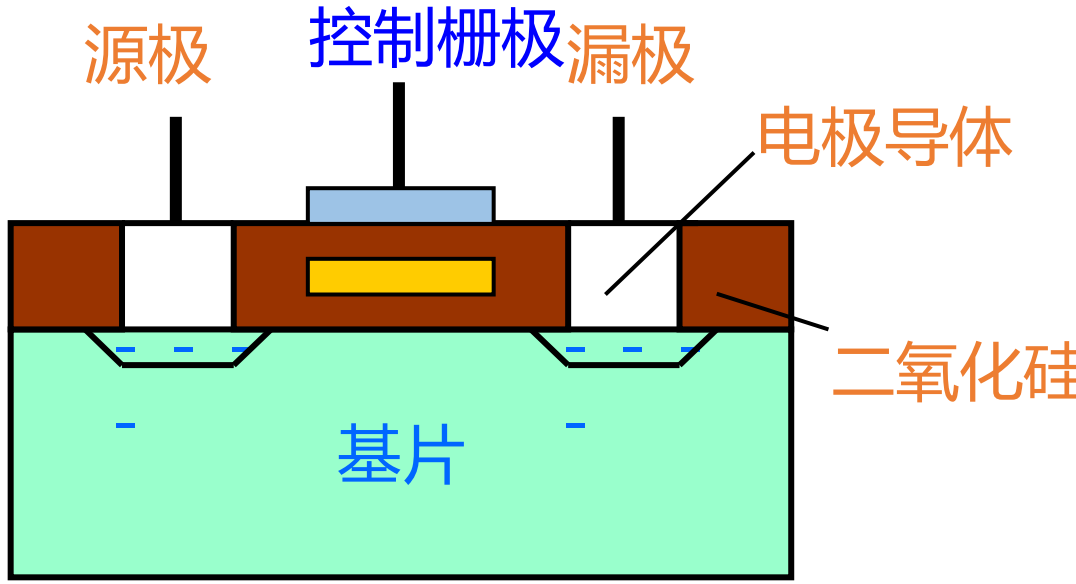


(b) 电路结构

- 可多次写入的ROM
- 写入的信息可长期保存
- 不需要这些信息或希望进行修改时，可擦除后再进行重写
- 擦除
  - EPROM芯片正面有一个石英玻璃窗口，用紫外线持续照射该窗口一段时间，会形成光电导现象，浮置栅上的电荷会完全泄漏，所有存储单元都恢复到原始状态

# 电可擦写ROM——EEPROM $E^2$PROM

- 电可擦除可编程只读存储器(Electrically Erasable Programmable ROM，EEPROM)
- 其在EPROM的浮置栅上方增加了一个控制栅极，写入方式与EPROM 相同
- 擦除时不需要用紫外线照射，只需将控制栅极加上高电平，就可以将浮置栅中的电荷泄漏掉
- 可以精准地删除某一存储单元，而不是一次性擦除芯片上的所有数据



# 闪速存储器 Flash Memory

- 快速擦写、非易失性存储器，可以在线进行擦除和重写。
- 其逻辑结构与 EPROM的相似，二者最主要的区别在于存储单元的结构和工艺。
- 工作方式：
  - 读工作方式
  - 编程工作方式
  - 擦除工作方式
  - 功耗下降方式
- 编程和擦除方式
  - 写命令到命令寄存器的方法来管理编程和擦除。
- 闪存芯片主要应用于微机主板上的 BIOS 和移动存储器中。
  - U盘、SSD均采用这种存储单元。

# 半导体存储器对比

| SRAM | DRAM | ROM | PROM | EPROM | EEPROM |
| --- | --- | --- | --- | --- | --- |
| MOS管 | 电容 | 开关 | 熔丝 | 浮置栅 | 浮置栅 |
| 快 | 慢 | 只读 | 写一次 | 高压写入 | 高压写入 |
| 6MOS | 1MOS+1C | | | 紫外线擦 | 控制栅极 |
| 功耗高 | 价格便宜 | | | 离线擦除 | 在线电擦 |
| | 动态刷新 | | | 擦后写 | 擦后写 |
| | 行列分开 | | | | |

# 本章主要内容

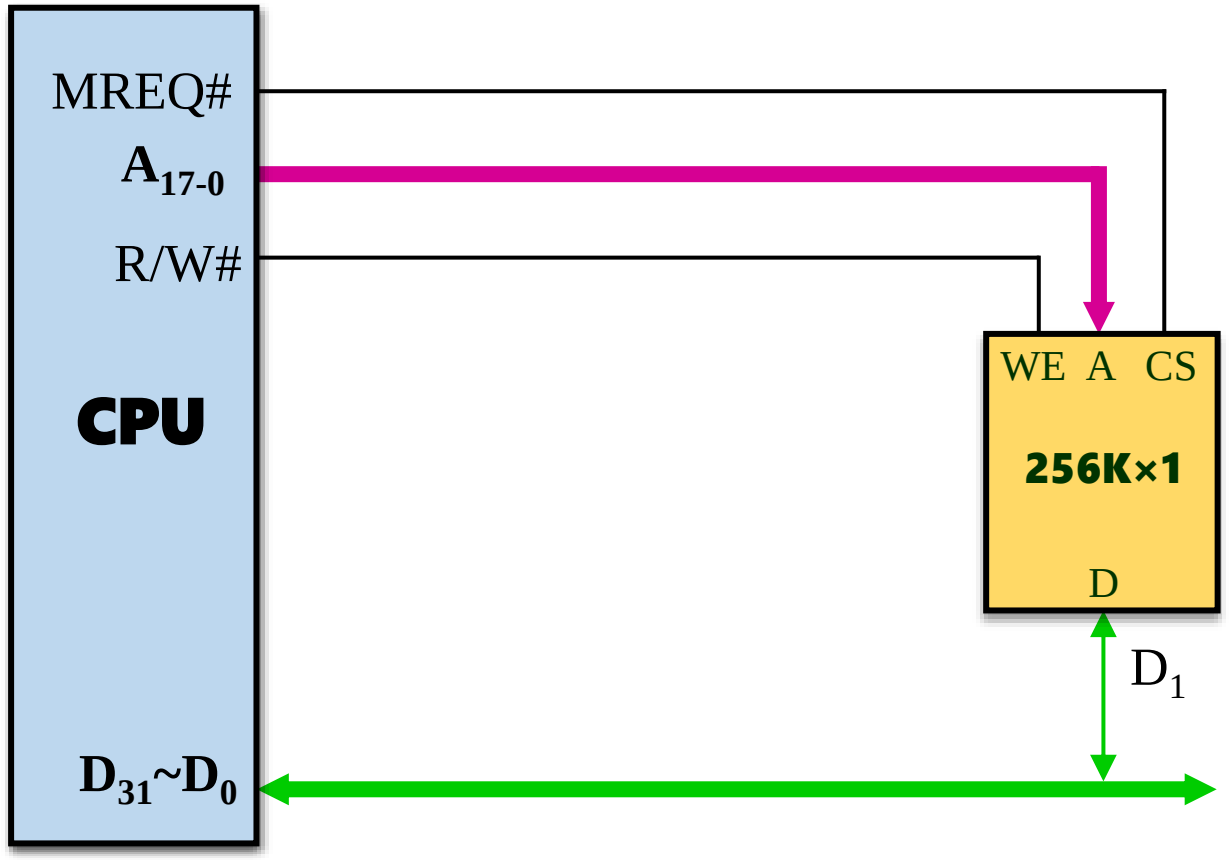
- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 主存储器

- 基本概念
- 随机存储器
- 只读存储器
- **主存储器与CPU的连接**
- 高速主存储器

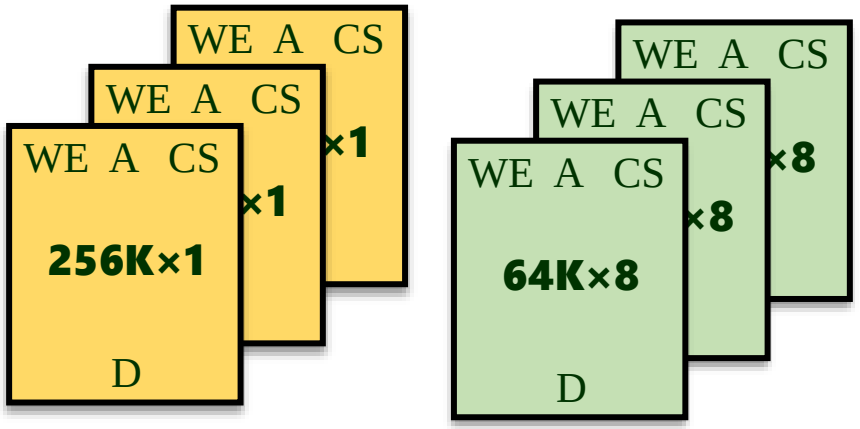
# 主存储器与CPU的连接

- 地址线的连接
- 数据线的连接
- 控制信号线的连接
- 存储扩展



# 存储器扩展

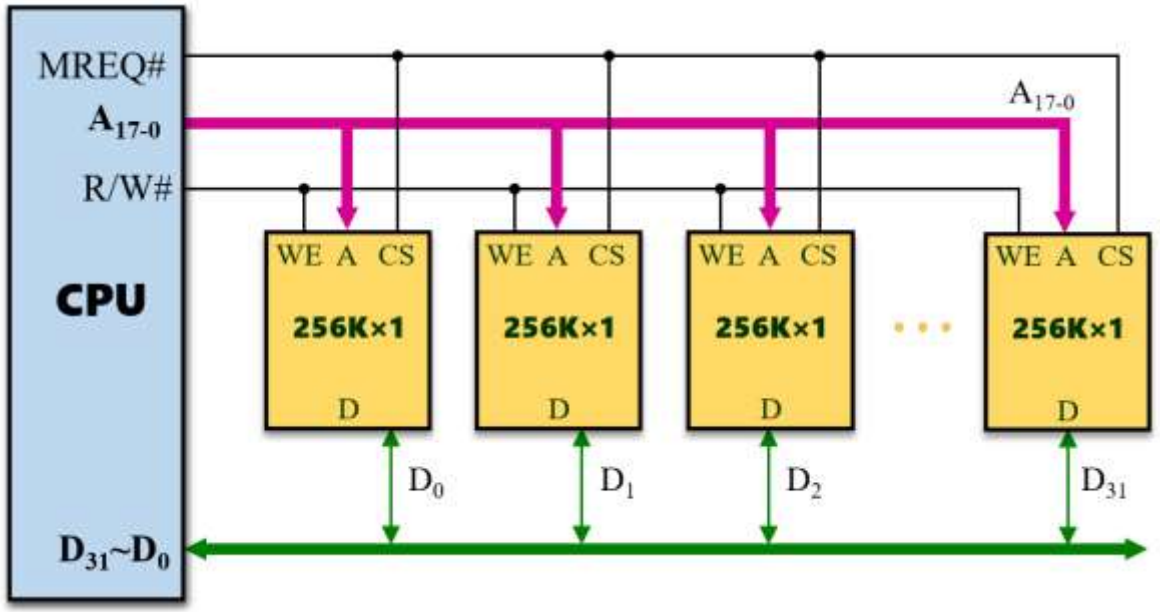
- 位扩展(数据总线扩展)
  - 各芯片并行工作
- 字扩展(地址总线扩展)
  - 同一时刻仅一芯片工作
- 字位同时扩展



# 位扩展（DBUS）

- **位扩展**：又称为**字长扩展**或**数据总线扩展**，当存储芯片的数据总线位宽小于CPU数据总线位宽时，采用位扩展的方式进行扩展。
- 所有存储芯片的地址线、读写控制线**并联**后分别与CPU的地址线和读写控制线连接:
- 存储芯片的数据线依次与CPU的数据线相连
- 所有芯片的片选控制线并联后与CPU的访存请求信号MREQ#相连。

# 位扩展（DBUS）

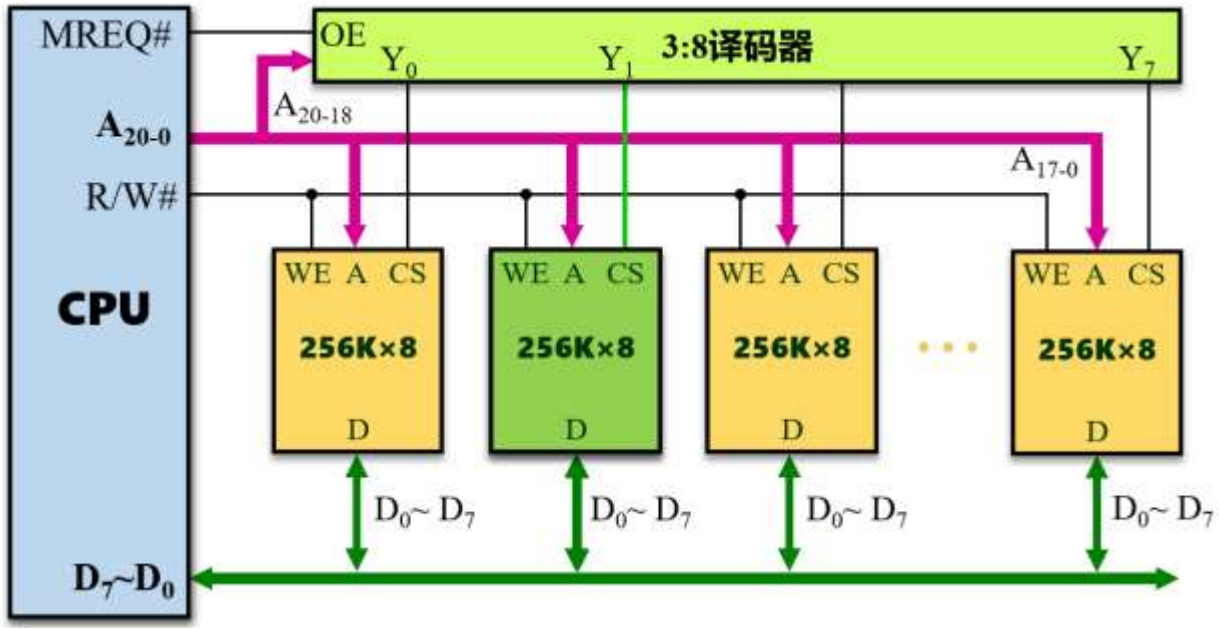



- 存储器数据位宽为N，存储芯片的数据位宽为k，N>k，则需要N/k个芯片进行存储扩展，利用256Kx1位的SRAM 存储芯片组成256Kx32位的存储器并与CPU连接，需要32/1=32个SRAM 芯片
- 与CPU连接时，将32个存储芯片的地址线(18根)、读写控制线各自并联，分别与CPU的地址线和读写控制线相连
- 同时将所有存储芯片的片选端均与CPU的MREO#信号相连，只有这样才能保证 32 个芯片同时被选中
- 32个存储芯片的数据线分别连到 CPU的 32 位数据线上

# 字扩展(ABUS)

- **字扩展**：**容量扩展**或**地址总线扩展。**
- 存储芯片的存储容量不能满足存储器对存储容量的要求时，采用字扩展方式来扩展存储器。
- 所有存储芯片的数据线、读写控制线各自并联，同时分别与CPU的数据线和读写控制线连接
- 存储芯片的片选信号可以由CPU多余的地线通过译码器译码产生。

# 字扩展(ABUS)

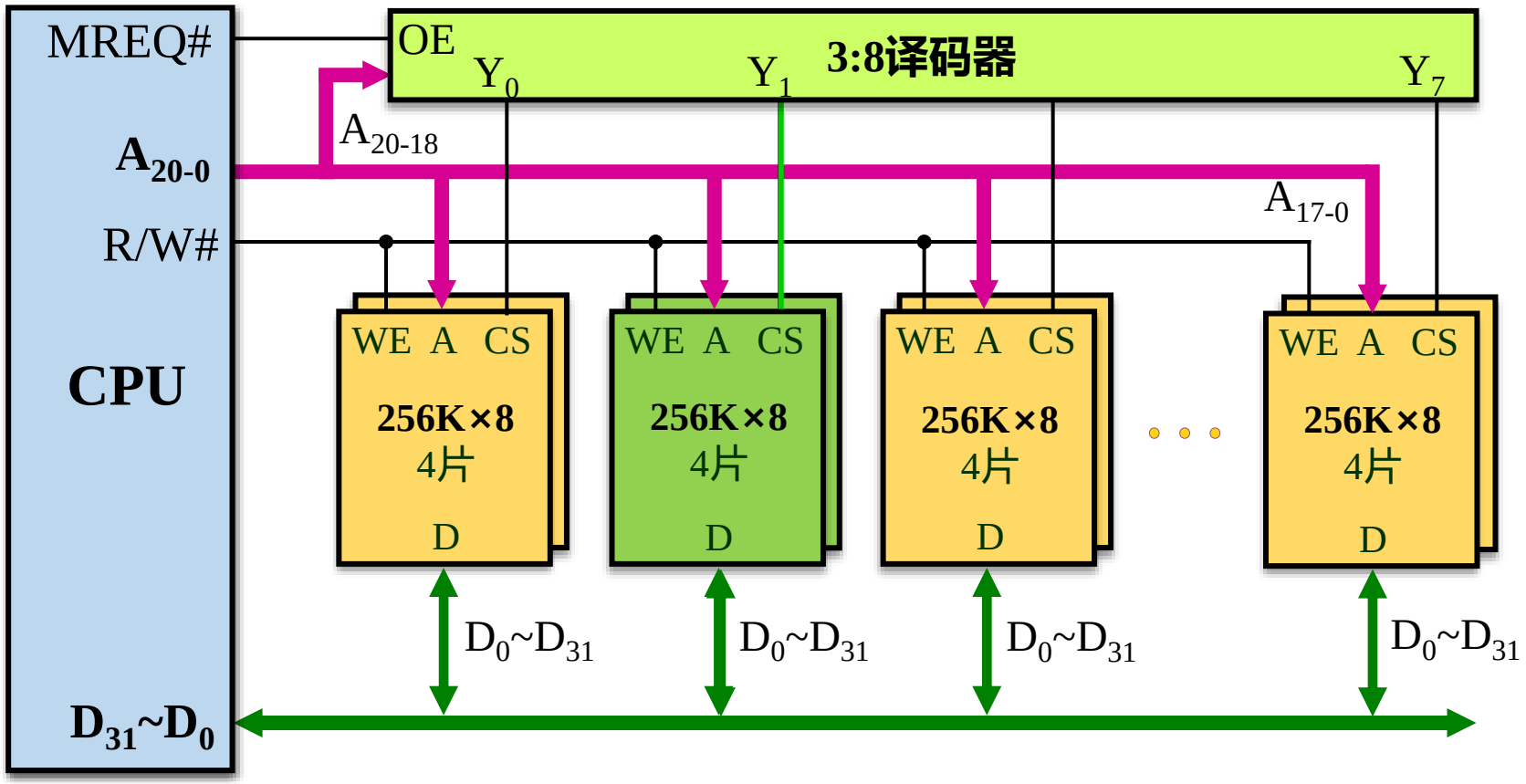



- 假设存储器容量为M，存储芯片的容量为I。若M>1,则需要 M/I个芯片进行存储扩展,
  - 利用256Kx8位的 SRAM 存储芯片组成2Mx8位的存储器并与CPU连接，需要 2M/256K=8个SRAM芯片。
  - 与CPU进行连接时，256K的芯片对应18根地址线，CPU访问2M的主存容量需要21根地址线
- 可以将高3位地址 A（20-18）送入3:8译码器输入端，将3:8译码器的8个输出分别连接到8个SRAM 芯片的片选信号CS端
- 将CPU内存请求信号MREQ#连接到译码器使能端，只有进行存储访问时，译码器才能进行工作，否则译码器输出全0(假设高电平有效)，所有存储芯片均不被选中，输出为高阻态。

# 字位同时扩展

- 存储芯片的数据位宽和存储容量均不能满足存储器的数据位和存储总容量要求时，可以采用字位同时扩展方式来组织存储器
- 首先通过位扩展满足**数据位**的要求，再通过字扩展满足**存储总容量**的要求。

# 字位同时扩展




存储系统 M*N位，若使用l*k位的芯片，l<M,k<N，需（M/l）*(N/k)个芯片

# 主存储器

- 基本概念
- 随机存储器
- 只读存储器
- 新型存储器
- 主存储器与CPU的连接
- **高速主存储器**

# 本章主要内容

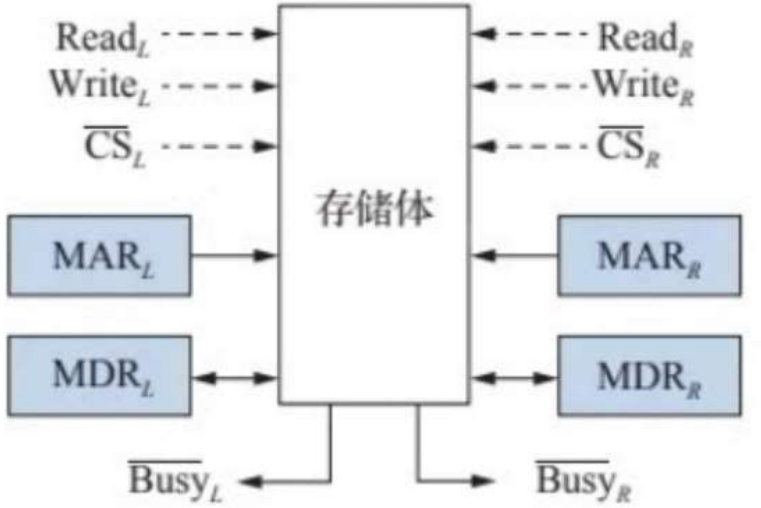
- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 高速存储器

- CPU与存储器之间的速度无法匹配
- 解决之道
  - 增加Cache；
  - 采用高速器件提高速度；
  - 采用双端口存储器；
  - 增加字长，在每个存储周期中存取多个字。
  - 将主存划分为多个模块，多模块并行

# 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端口
- 具有两组相互独立的读写控制线路
- 两组读写控制线路可以并行操作
- 端口地址不相同，无冲突，并行存取
- 端口地址相同，读写冲突，无法并行存取

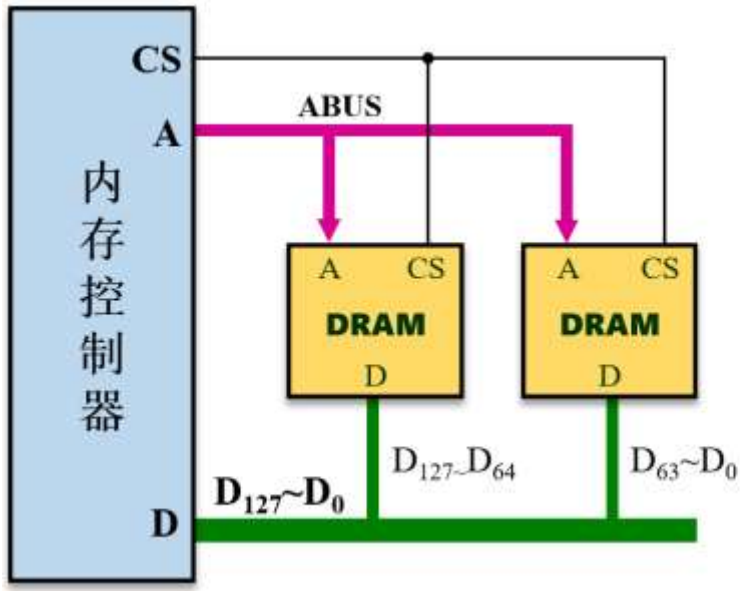


# 单体多字存储器

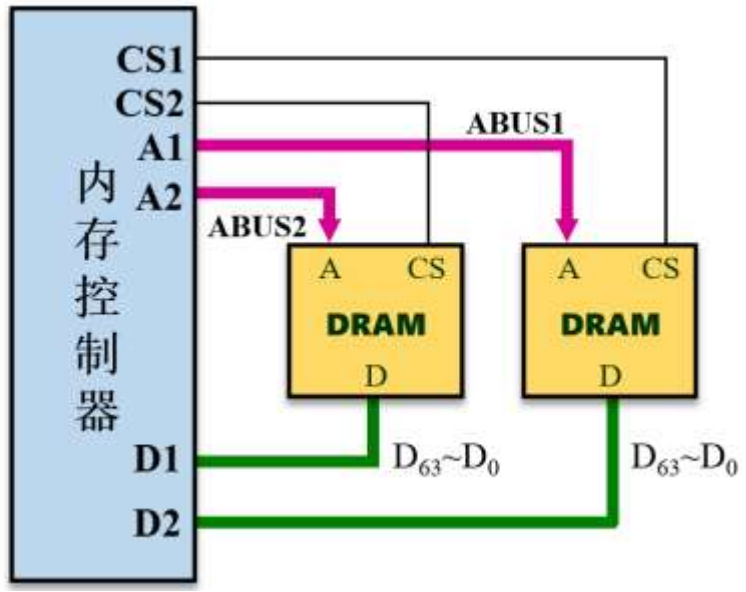
- 单体多字存储器的构造与存储器位扩展方式完全相同，
  - 多个存储模块共享地址总线
  - 按同一地址并行访问不同存储模块的同一单元
- 多通道内存技术采用的单体多字技术，
  - 双通道
  - 三通道
  - 四通道

# 多通道内存

- **联动模式**：
  - 一种双通道内存技术，
  - 同一时刻两根内存并发工作，各自访问同一地址单元中的 64 位数据
  - 两根内存的容量、频率、时序只有完全一致才能同步工作
- **非联动模式**：
  - 内存控制器通过独立的片选信号、地址总线、读写控制线连接两根内存，数据总线也是独立的两条64位总线
  - 该方式中两根内存也可以并发工作，但二者地址、读写命令并不需要同步
  - 频率相同即可，容量和时序特性并不要求一致



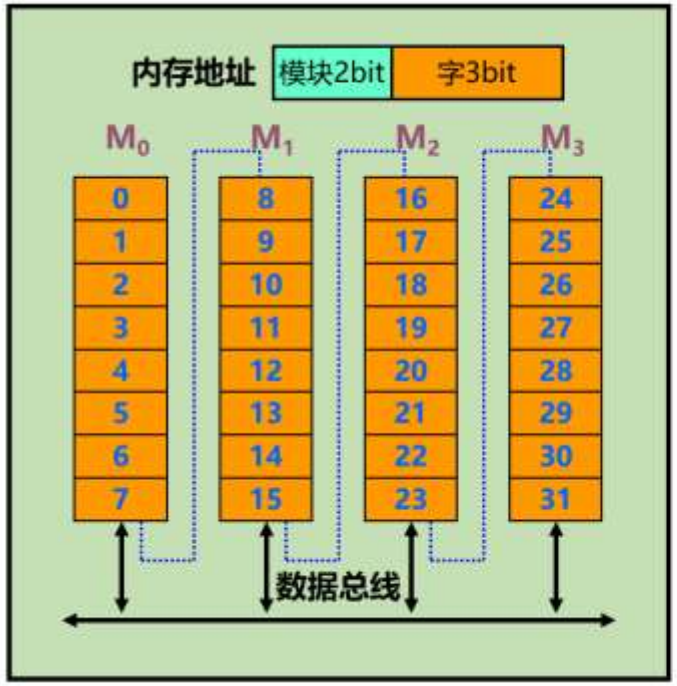

联动模式

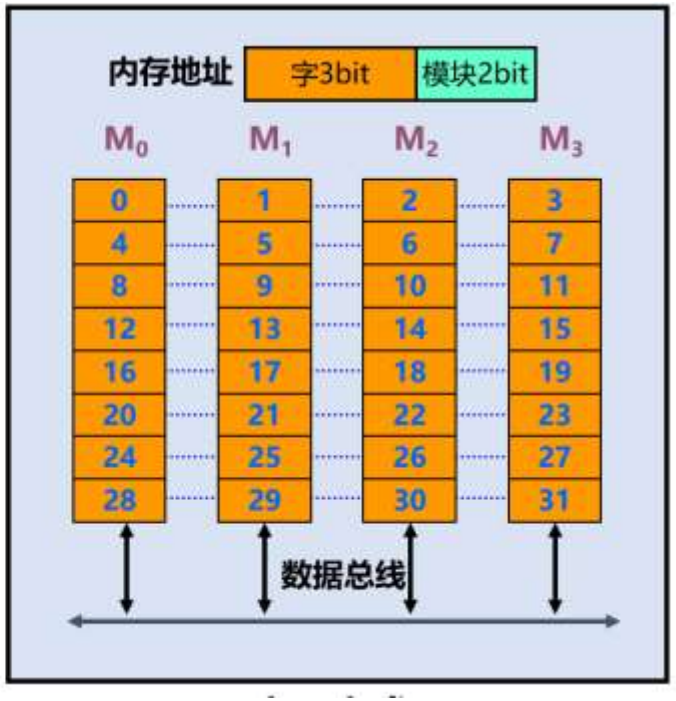



非联动模式

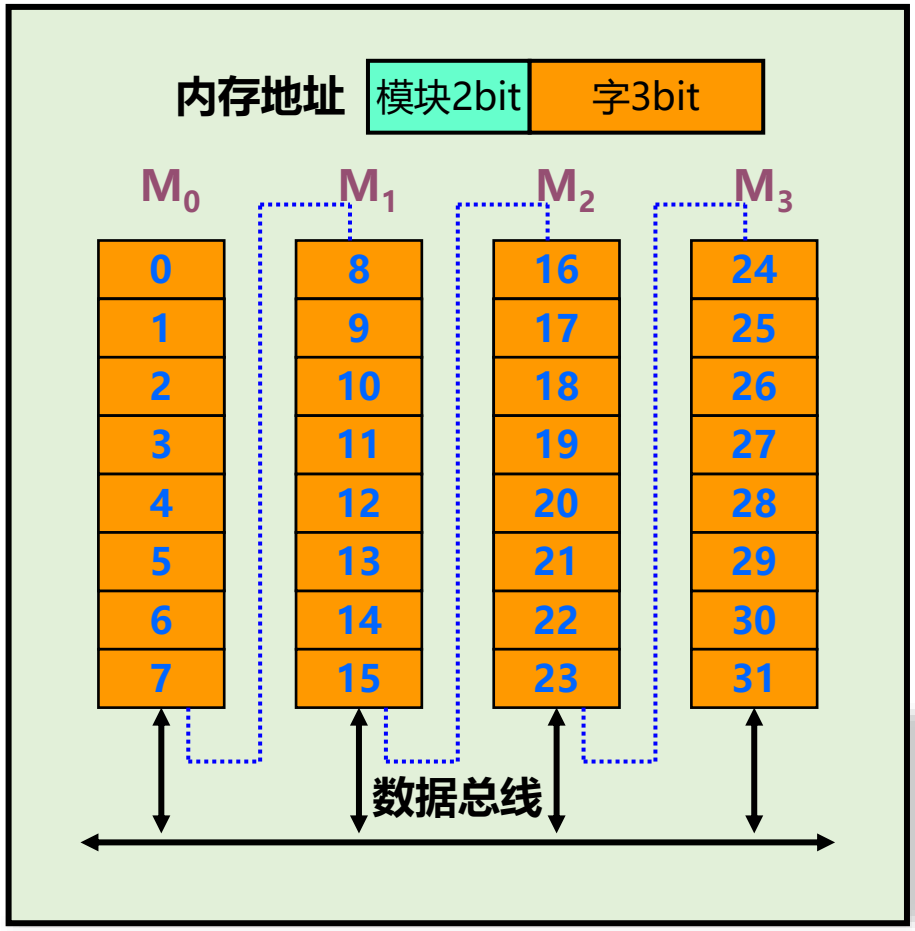
# 多体并行存储器 （多体交叉存储器）

- 多体交叉存储器也由多个存储模块构成，这些模块的容量和存取速度相同。
  - 高位多体交叉
  - 低位多体交



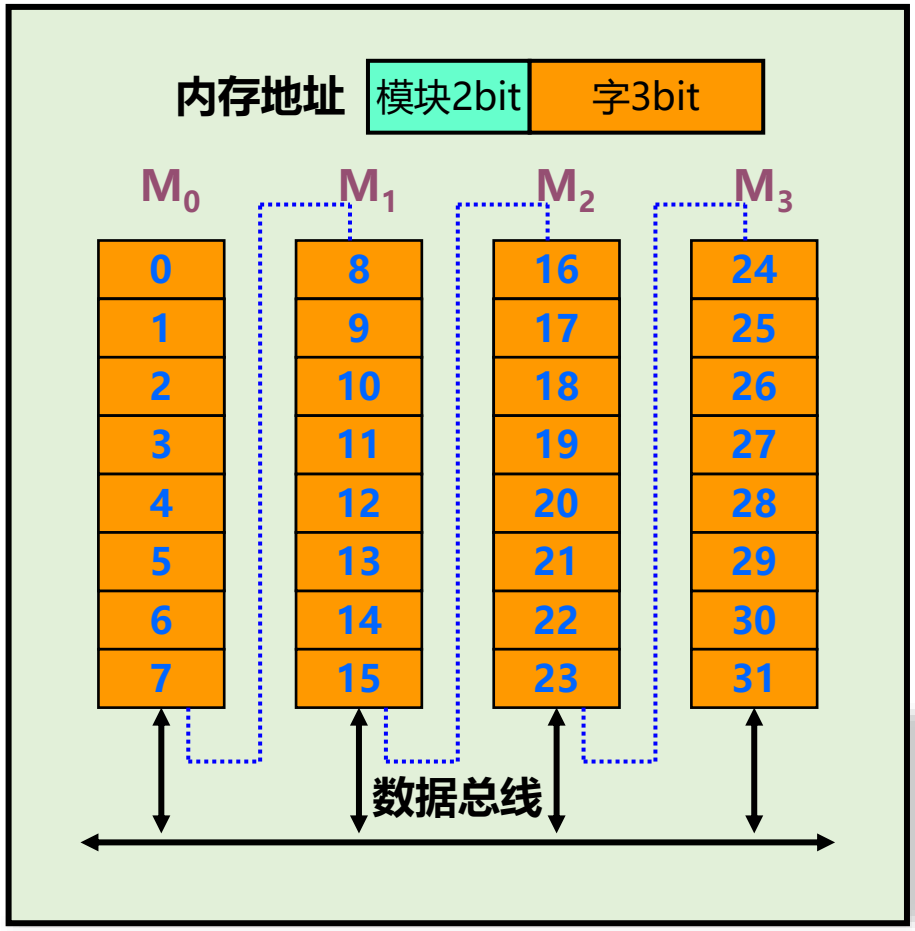
# 高位多体交叉




顺序方式

- 高位地址译码产生片选信号，选择不同的存储模块
- 低位地址直接选择一个存储模块内的不同存储单元
- 高位交叉方式中不同存储模块对应不同的地址区间，将地址顺序分配给一个模块后，按顺序为下一个模块分配地址，又称为顺序编址模式

# 高位多体交叉

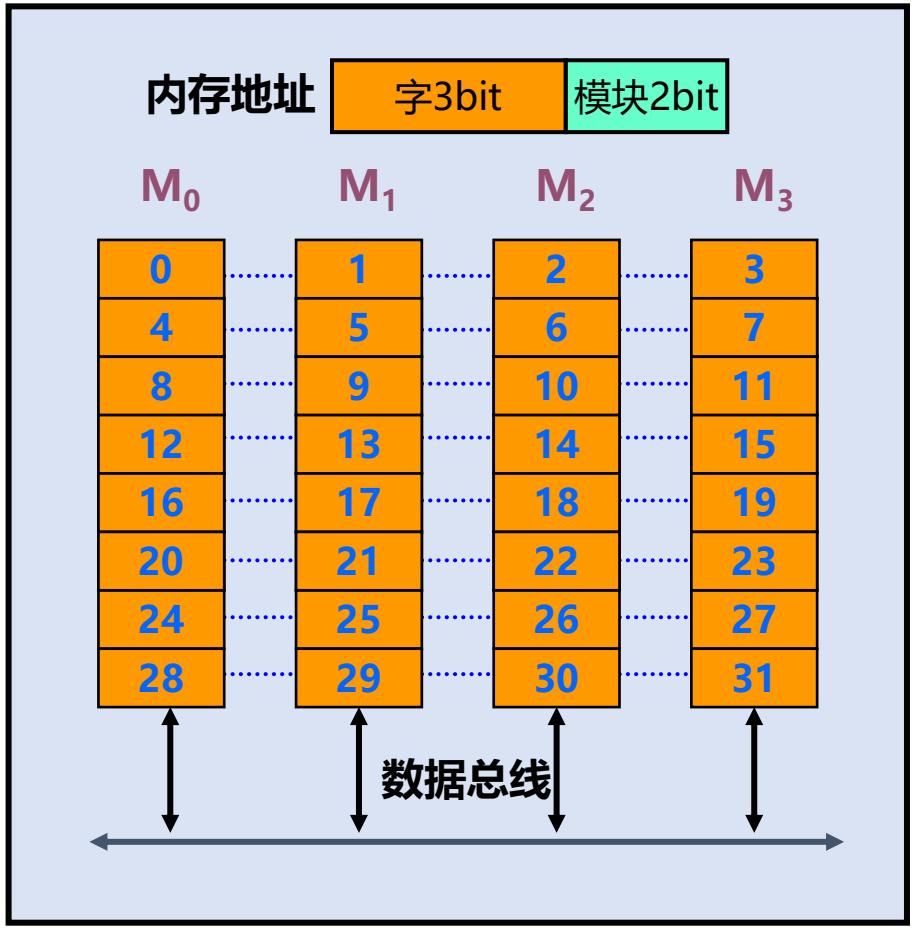



顺序方式

- 相邻的地址在同一存储体内
- 不同存储体中的地址不相邻

程序执行过程中的指令和数据基本分布在同一个存储体中，往往会导致**一个存储体访问频繁，而其他存储体基本处于空闲状态**，无法实现多个存储体的并行工作。

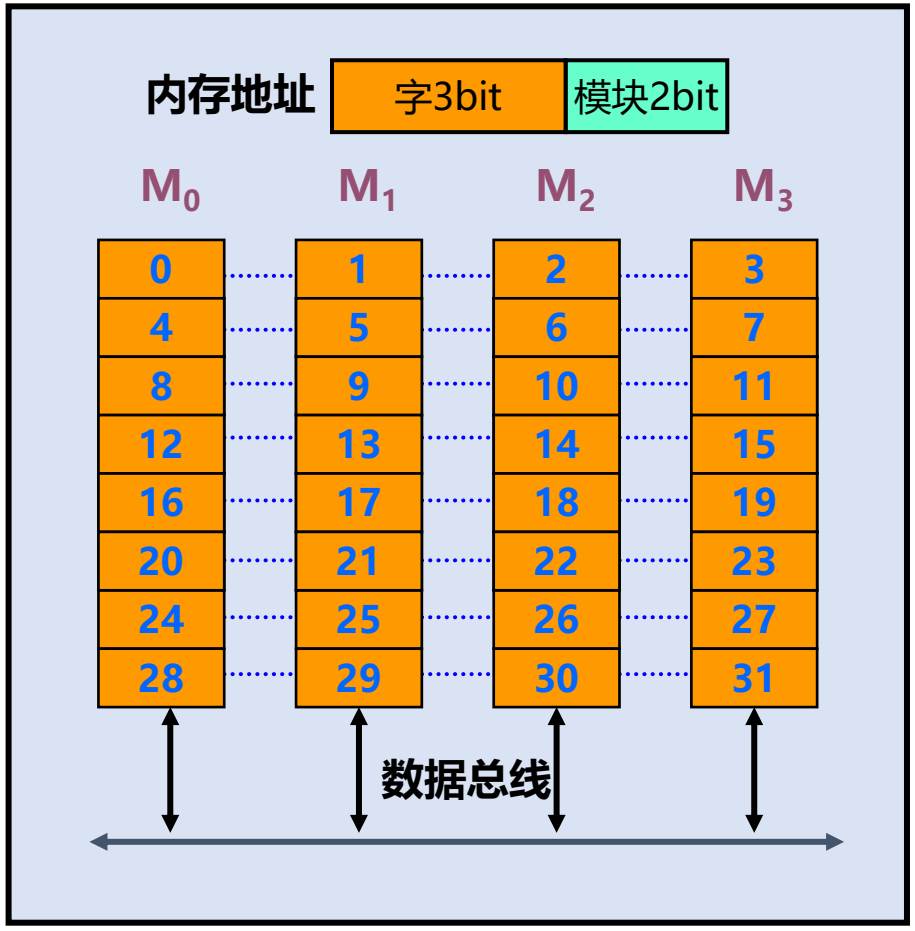
# 低位交叉存储器




交叉方式

- 低位地址译码进行片选，而用高位地址选择存储模块内的不同存储单元
- 又称为交叉编址模式

# 低位交叉存储器



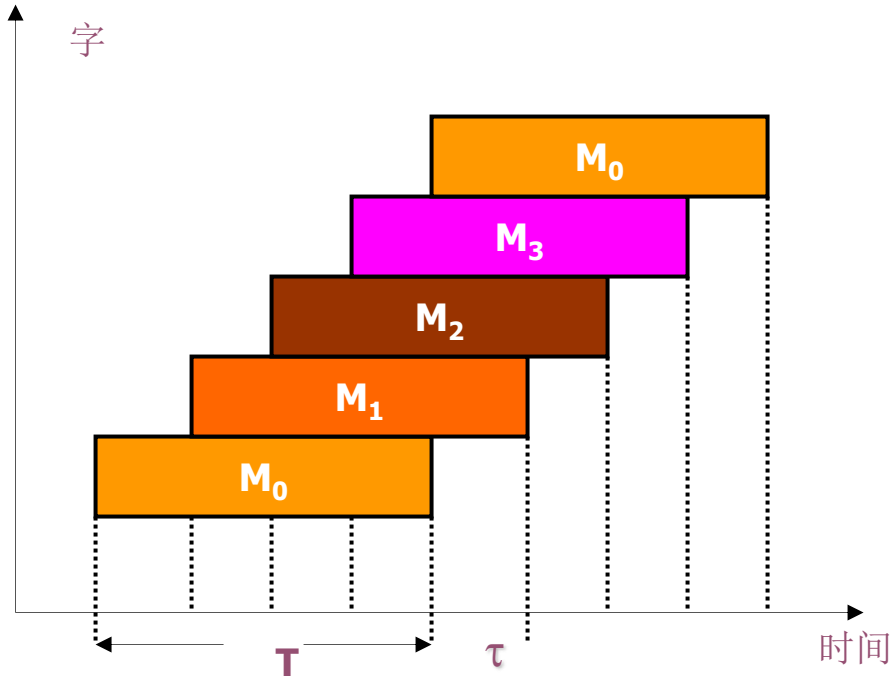

交叉方式

- 相邻的地址处在不同存储体内
- 同一存储体中的地址不相邻

表 4.3　4 体低位交叉存储体编址序列

| 模块号 | 地址编址序列 | 最低两位地址 |
|---|---|---|
| 0 | 0，4，8，…，4*i* | 00 |
| 1 | 1，5，9，…，4*i*+1 | 01 |
| 2 | 2，6，10，…，4*i*+2 | 10 |
| 3 | 3，7，11，…，4*i*+3 | 11 |

# 交叉编址顺序访问时可按流水方式存取




T: 模块存取周期 τ: 总线传输周期
m: 存储器交叉模块数

- $T = m\tau$
- $m = T/\tau$ 交叉存取度

连续读取n个字的时间

- $t_1 = T + (n-1)\ \tau$

# 习题

- 4.4、4.6

# 习题答案

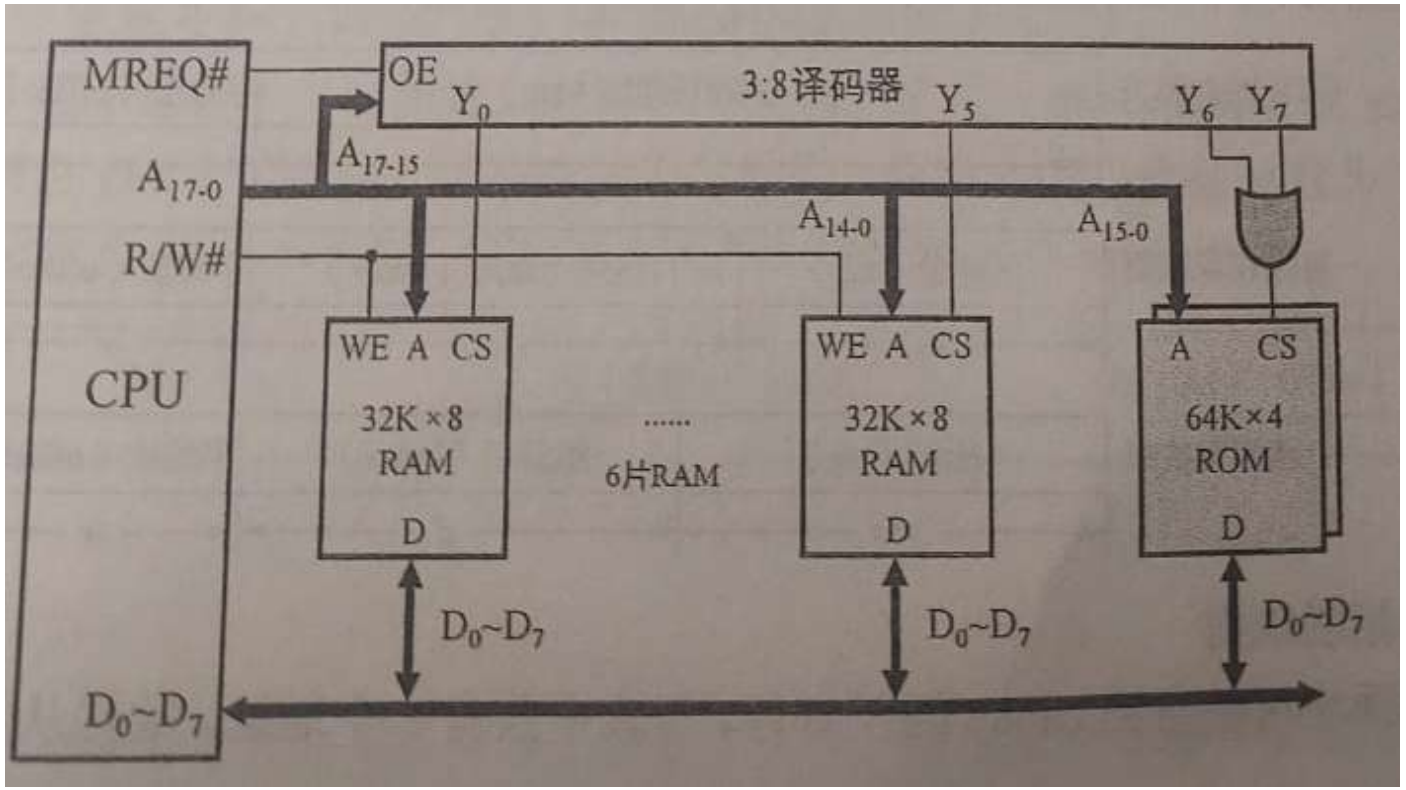
## 4.4

总容量：32KB = 32 × 1024 字节
每个地址：16位 = 2字节
地址位数： 32 × 1024/2=2^14，14位
数据寄存器位数直接由字长决定，16位

# 习题答案

## 4.6

- 256KB=2^18 B，00000H-3FFFFH
- 30000H~3FFFFH容量为64KB的高地址区间，其需要64KBx4位ROM 芯片2片进行位扩展
- 00000H~2FFFFH 存储空间为 RAM 区间，容量为192KB, 其需要32KBx8位RAM芯片 192KB/32KB=6片进行字扩展



# 本章主要内容

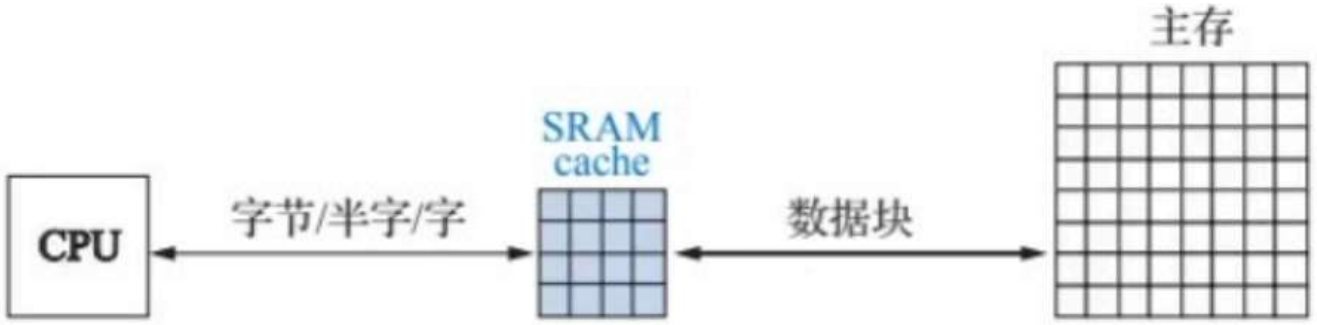
- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 4.5 高速缓冲存储器

- **4.5.1 cache工作原理**
- **4.5.2 程序局部性**
- **4.5.3 cache的基本概念**
- **4.5.4 cache读写流程与关键技术**
  - 相联存储器、地址映射、 替换算法、 写入策略
- **4.2.9 cache应用**

# Cache工作原理

- Cache：在 **CPU与主存之间**增加一个隐藏的小容量的快速的 SRAM




- 将主存中**经常访问或即将访问的数据的副本**调度到小容量的SRAM 中，使得大部分数据访问都可以在快速的Cache中进行
- CPU 执行的程序具有较强的**程序局部性。**

# 4.5 高速缓冲存储器

- 4.5.1 cache工作原理
- 4.5.2 程序局部性
- 4.5.3 cache的基本概念
- 4.5.4 cache读写流程与关键技术
  - 相联存储器、地址映射、 替换算法、 写入策略
- 4.2.9 cache应用

# 数据访问局部性（程序局部性）

- **程序局部性**---程序在执行时呈现出局部性规律，
  - 整个程序的执行仅限于程序中的某一部分
  - 执行程序所需的指令和数据也仅局限于某个存储区域内
- **空间局部性:** 某内存区域刚被访问，很快其相邻区域有可能被访问
- **时间局部性:** 某内存区域刚被访问，很快该区域可能会被重复访问

# 程序局部性举例

- **数据**
  - 数组元素访问 (空间)
  - 结构体、数据库记录访问(空间)
  - 局部变量，计数器，指针等被重复使用 (时间)
- **指令**
  - 顺序访问的指令 (空间)
  - 重复使用的循环体 (时间)
  - 子函数 (时间)

```
sum = 0;
for (i = 0; i < 1000; i++)
   sum += a[i];
return sum;
```

- for 循环体中的指令序列在主存中顺序存放，具有空间局部性
- 循环将被执行1000次，时间局部性
- 变量i、sum，在每次执行循环代码时都会被用到，时间局部性
- 数组包含 1000个数据元素，这些数据在内存中顺序存放，空间局部性

# 4.5 高速缓冲存储器

- 4.5.1 cache工作原理
- 4.5.2 程序局部性
- 4.5.3 cache的基本概念
- 4.5.4 cache读写流程与关键技术
  - 相联存储器、地址映射、 替换算法、 写入策略
- 4.2.9 cache应用

# cache 术语

- **命中率** （ hit rate ）
  - 主存访问中cache命中比例
- **缺失率** （miss rate）
  - 1 – 命中率
- **命中访问时间**: （hit time）
  - 数据查找时间、cache访问时间、总线传输时间
- **缺失损失** （miss penalty）
  - 主存块调入cache，数据传输到处理器的时间
  - 包括数据查找时间、主存访问时间、cache访问时间
  - 远大于命中时间

# cache 术语

- **块（block）：** cache 和主存都被分成若干个固定大小的数据块
  - 每个数据块又包含若干个字，数据缺失时需要将访问数据所在的块从慢速主存载入 cache 中，这样相邻的数据也随着数据块一起载入 cache。
  - 充分利用空间局部性，提高顺序访问的命中概率。
  - 块过小无法利用预读策略优化空间局部性，块过大将使得替换算法无法充分利用时间局部性。
- 主存地址和cache地址都可以分为块地址和块内偏移地址(Ofset)


图 4.32　地址格式

# cache命中率

- **命中率**
  - $N_c$表示cache完成存取访问的总次数
  - $N_m$表示主存完成存取访问的总次数

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

- **$t_a$平均访问时间**
  - $t_c$表示命中cache时的访问时间
  - $t_m$表示缺失访问时间

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

- **访问效率**=$t_c / t_a$
- 影响命中率的几个因素
  - 程序行为（局部性）　　cache容量
  - 组织方式　　块大小

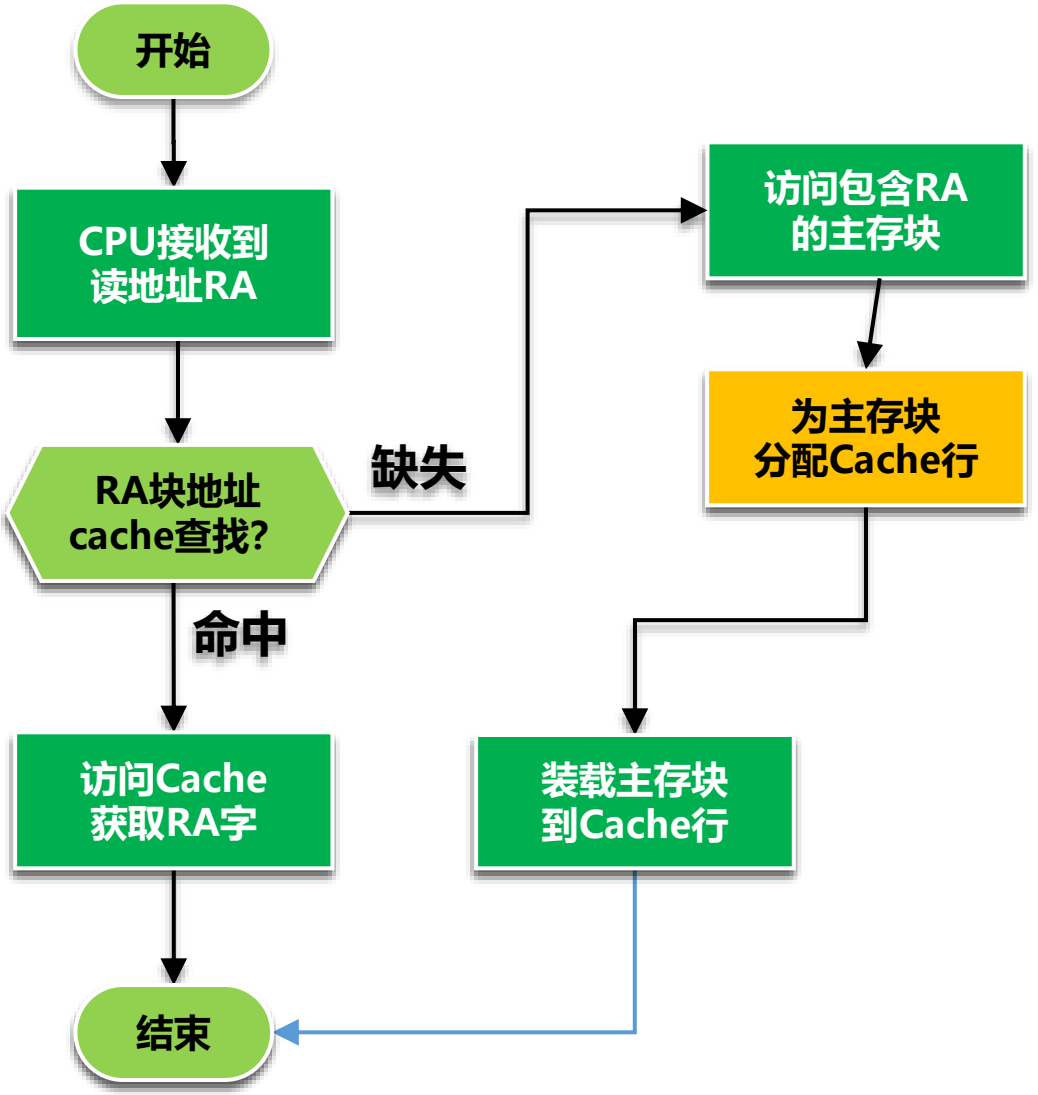
# 4.5 高速缓冲存储器

- 4.5.1 cache工作原理
- 4.5.2 程序局部性
- 4.5.3 cache的基本概念
- 4.5.4 cache读写流程与关键技术
  - 相联存储器、地址映射、 替换算法、 写入策略
- 4.2.9 cache应用

# CPU cache系统读过程

- CPU给出主存地址（块地址，块内地址）
- 主存**块地址**为**关键字**进行查找
- 如相符表示**副本**在cache中，命中，访问cache
- 否则数据缺失，访问主存
  - 将数据所在块副本调入cache
  - 载入副本过程可能引起替换
  - 更新查找表，记录当前数据块地址
  - cache缺失时系统等待数据调入

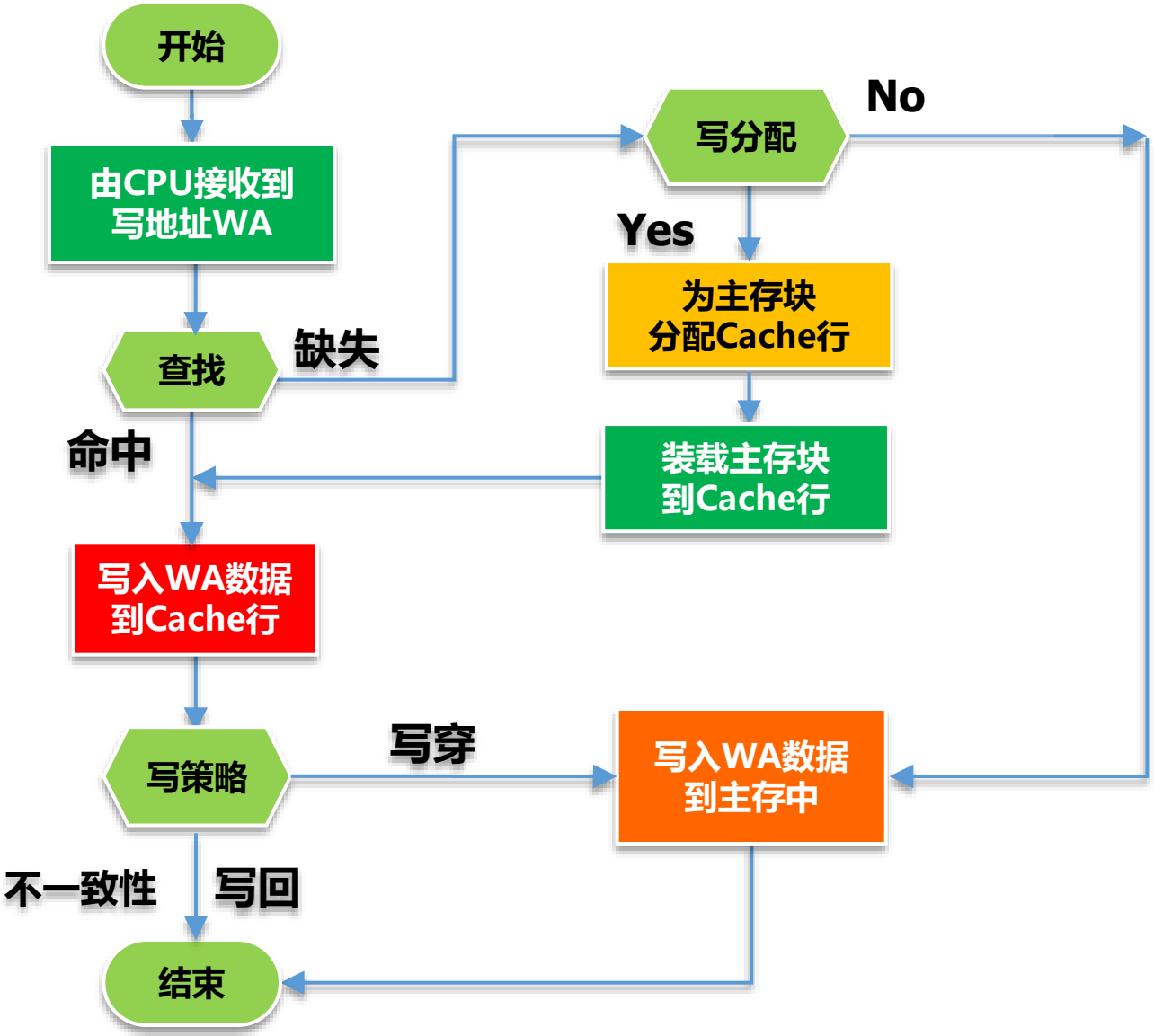
# CPU cache 读操作



# CPU cache系统写过程

- CPU给出主存地址
- 主存块地址为关键字进行查找
  - 相符则表示命中，数据副本在cache中
  - 缺失根据写分配策略决定是否将该主存地址对应数据块调入
- 写入数据到cache
- 根据写策略决定是否写入主存

# CPU cache 写操作



# cache关键技术

- **数据查找　Data Identification**
  - 如何判断数据在cache中
- **地址映射　Address Mapping**
  - 主存数据如何放置到cache行/槽中
- **替换策略　Placement Policy**
  - cache满后如何处理
- **写入策略　Write Policy**
  - 如何保证cache与memory的一致性

# 如何查找

- **数据查找**　地址映射　替换策略　写入策略
- 主存地址 → cache 地址 → cache 数据
  - 程序员软件思路
    - **什么数据结构，如何快速查找**
  - 架构师硬件思路
    - **如何硬件存储，如何快速查找**
- **相联存储器**
  - 按内容进行访问的存储器

**cache查找表**

| 主存块号 | cache块号 |
| --- | --- |
| 001 | 2 |
| 021 | 4 |
| … | … |
| 091 | 7 |

# 相联存储器 Content addressable memory（CAM）

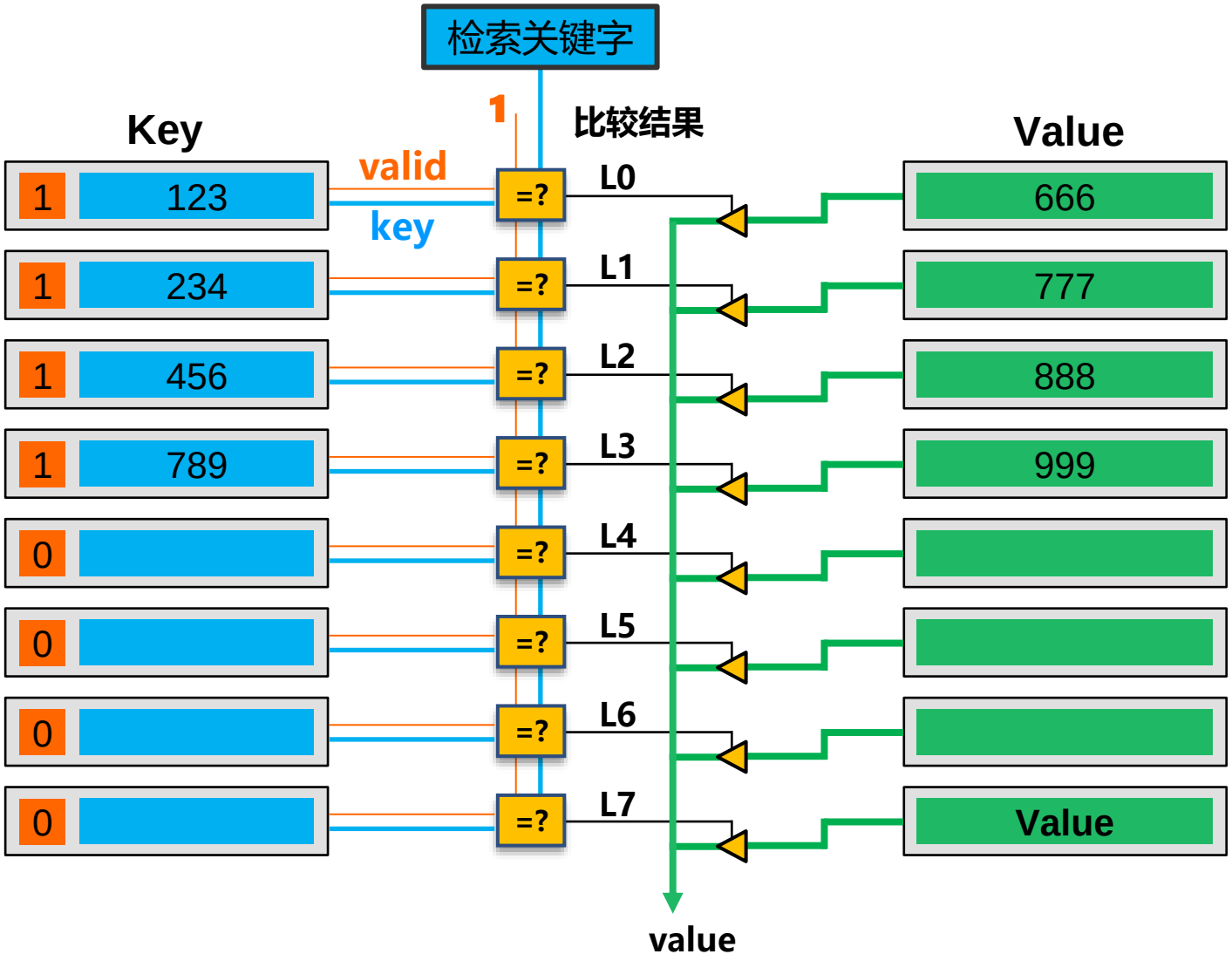
按内容进行访问的存储器　　（Key,Value）

| 物理地址 | 工号 | 姓名 | 出生年月 | 工资数 |
|---|---|---|---|---|
| N | 001 | 张帅 | 1976/7 | 4000 |
| N+1 | 021 | 李猛 | 1978/9 | 2000 |
| N+2 | 023 | 郝牛 | 1977/6 | 3000 |

| 主存块号 | Cache地址 |
|---|---|
| 001 | 00001000 |
| 021 | 00001010 |
| 023 | 00001001 |

用关键字去检索存储器中部分字段，对包含关键字的存储单元进行读写，以内容作为地址访问的存储器称为相联存储器， CAM的输入不是地址，而是检索关键字 key，输出则是该关键字对应的 value 值。

# 相联存储器


检索关键字
1
比较结果
Key
valid
key
Value
1
123
1
234
1
456
1
789
0
0
0
0
=?
=?
=?
=?
=?
=?
=?
=?
L0
L1
L2
L3
L4
L5
L6
L7
666
777
888
999
Value
value

# 相联存储器

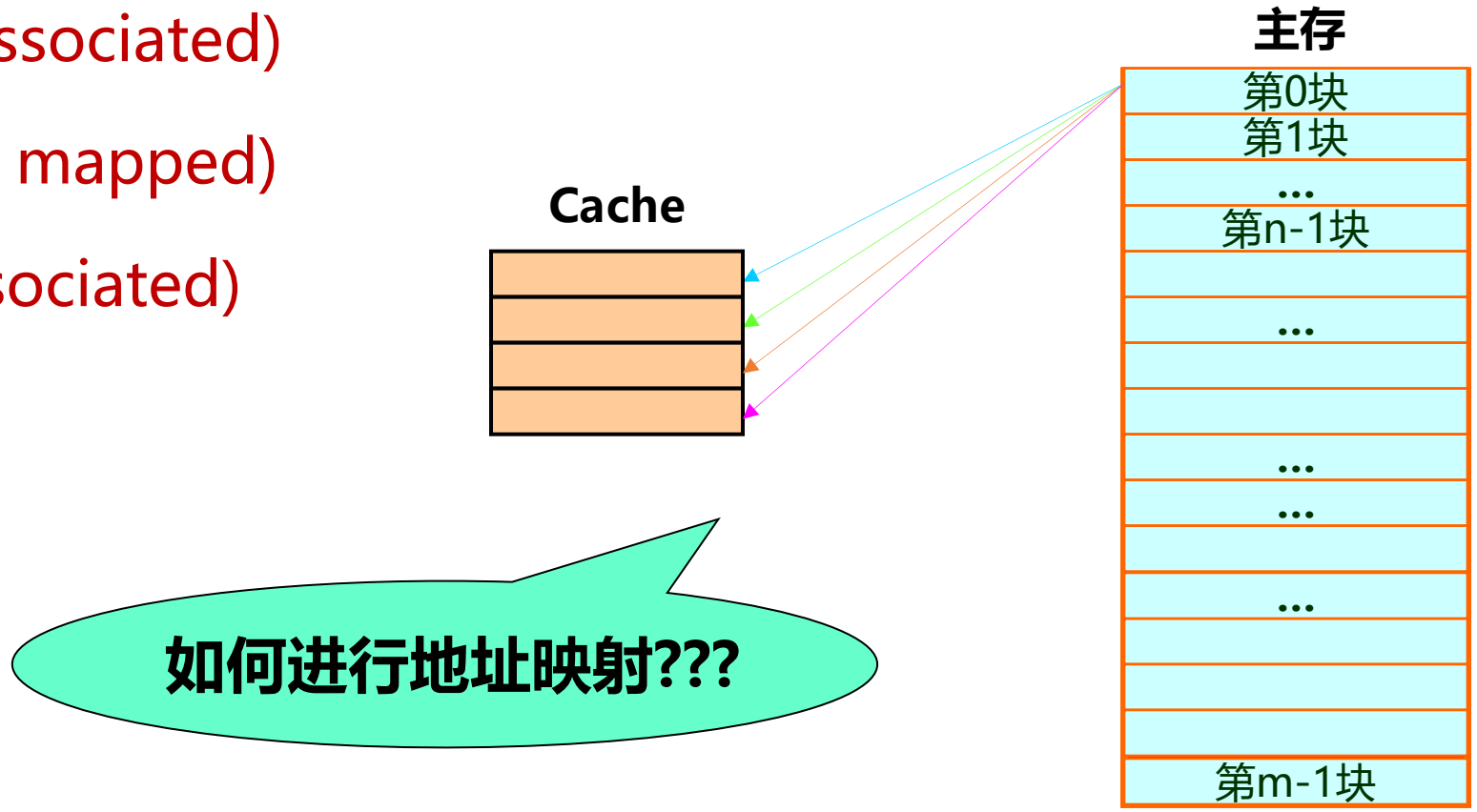
- 至少按内容进行访问（Key， Value）
- 以关键字作全局并发比较
  - 硬件成本高（比较器多）
- 存放查找表
- 存储容量 = 查找表容量 = 表项数 * 表项大小
  - cache中用于存放**块表**，虚拟存储器中存放**段表、页表**
    - (valid, Key, Value )
    - ( 有效位, 主存块地址 ， cache块地址)
    - ( 有效位, VPN ， PPN )

# cache关键技术

- **数据查找　Data Identification**
  - 如何判断数据在cache中
- **地址映射　Address Mapping**
  - 主存数据如何放置到Cache行/槽中
- **替换策略　Placement Policy**
  - Cache满后如何处理
- **写入策略　Write Policy**
  - 如何保证cache与memory的一致性

# 主存与cache地址映射关系

- **地址映射：**将主存地址空间映射到cache的地址空间，即把存放在主存中的程序或数据块载入 cache 块的规则
  - 全相联　(fully-associated)
  - 直接相联 (direct mapped)
  - 组相联　(set-associated)

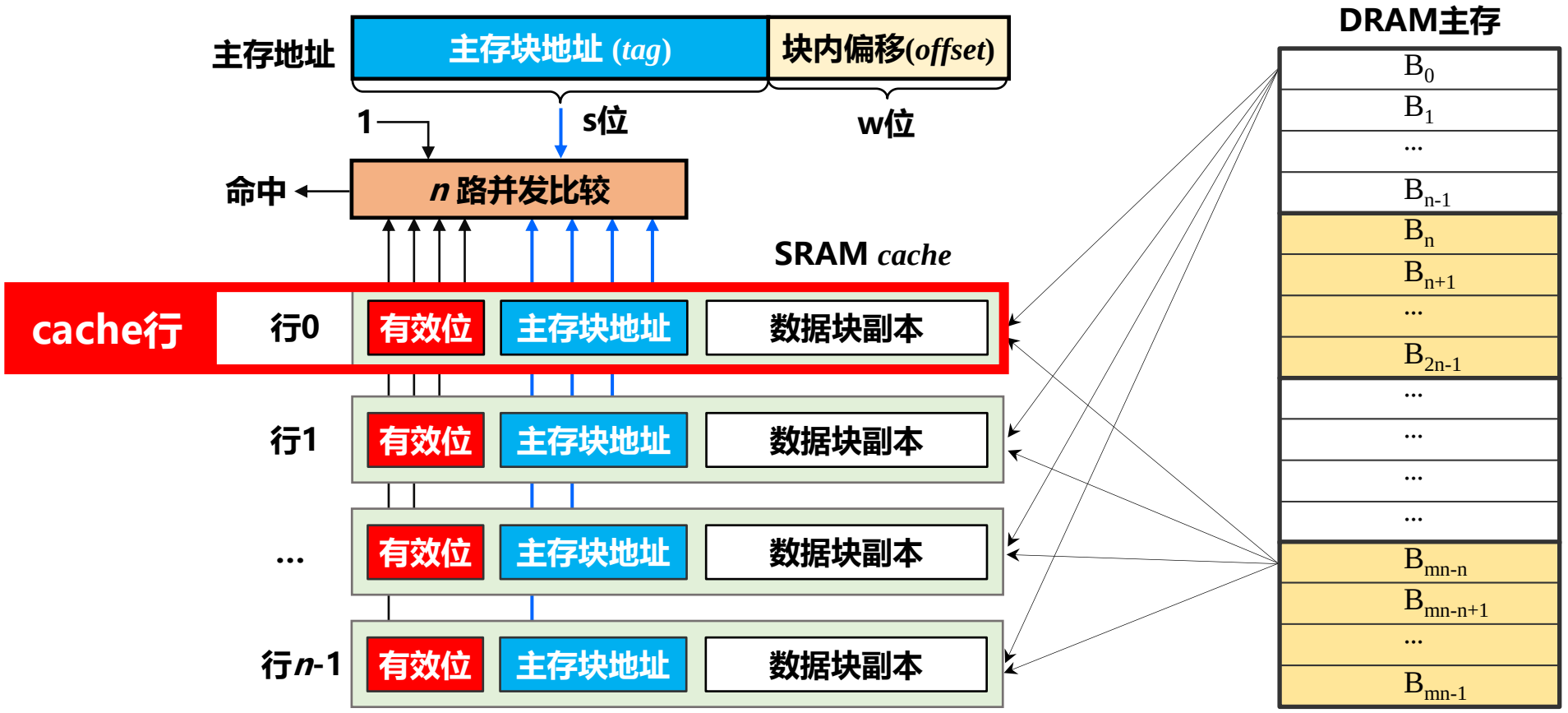


# 全相联映射的基本原理

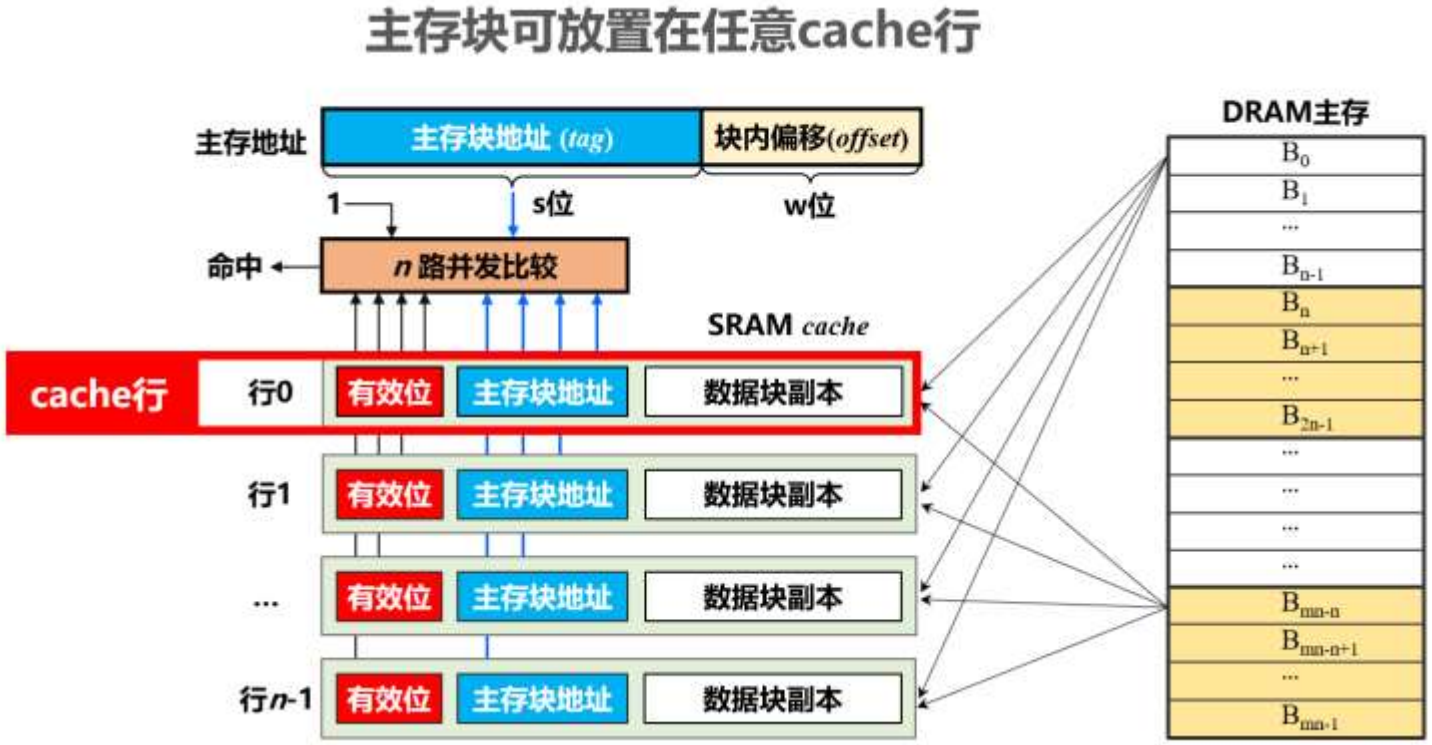
- 主存中的每一个数据块都可以放置到cache的任意一个数据块中，是一对多的映射关系。
- 新的主存数据块可以载入cache 中任何一个空位置，只有 cache 满时才需要进行数据块置换。
- 全相联映射时cache利用率最高，但查找成本较高，需要CAM 提供快速的查找功能。

# 全相联映射

## 主存块可放置在任意cache行



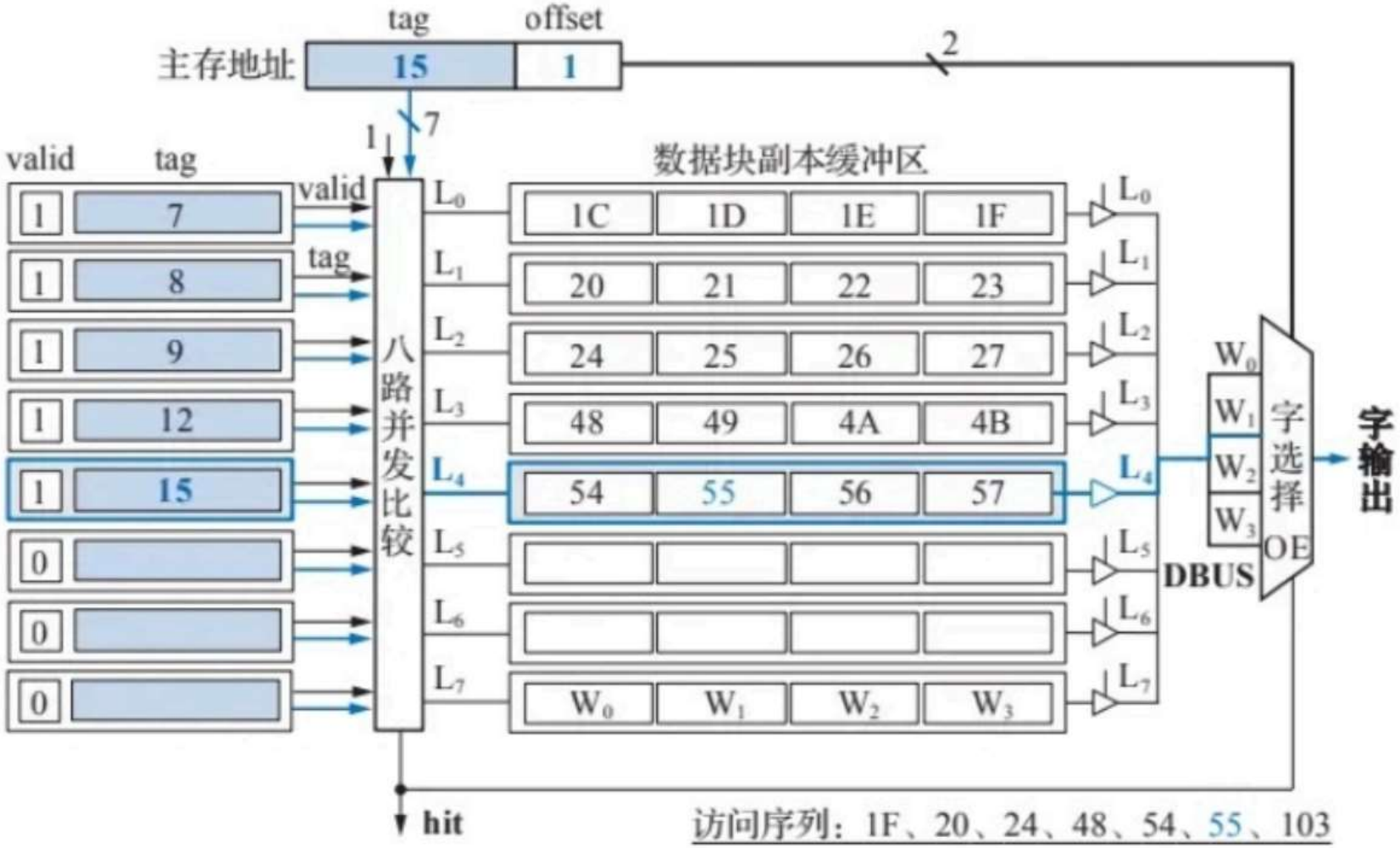
# 全相联映射




- cache块大小=2^w字节
- cache副本缓冲区容量=nx2^w字节
- 主存容量= 2^(s+w)字节，不考虑脏数据位和淘汰计数
- cache的实际容量nx(1+s+8x2^w)位

# 全相联映射动态载入过程

Cache 8行，块大小4W，主存$2^9$ W　Cache总容量=（valid+标记位+标志位+副本容量）*总行数




| 读访问序列 | 1F | 20 | 24 | 48 | 54 | 55 | 103 |
|---|---|---|---|---|---|---|---|

# 全相联映射动态载入过程

| 访问序列 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 十六进制 | 1F | 20 | 24 | 48 | 54 | 55 | 103 |
| 二进制 | 0000111 11 | 0001000 00 | 0001001 00 | 0010010 00 | 0010101 00 | 0010101 01 | 1000000 11 |
| ( tag,offset ) | (7,3) | (8,0) | (9,0) | (12,0) | (15,0) | (15,1) | (40,3) |
| cache 行 | 0 | 1 | 2 | 3 | 4 | 4 | 5 |
| 访问情况 | 载入 | 载入 | 载入 | 载入 | 载入 | 命中 | 载入 |

- 假设初始状态为空，所有cache 行中的有效位都为0
- 第1次主存访问地址为1F，访问缺失，载入1F所在的主存数据块（1C-1F）存到cache的第0行，同时将主存块地址7作为标记存放在第0行的标记字段中，并将有效位valid置1,方便后续查找。
- 第2~5次访问的主存块地址均不相同，所以访问均为缺失状态，分别载入对应的数据块到第1、2、3、4行的空行中。
- 第6次访问的主存字地址为55，主存块地址为15，第4行中的标记字段与之相同且有效位为1，访问命中，由块内偏移offset=1选择 cache 行中的 W1输出。
- 第7次访问 103 结果也是缺失，由于cache 仍然有空行，因此继续载入数据。

# 全相联应用场合

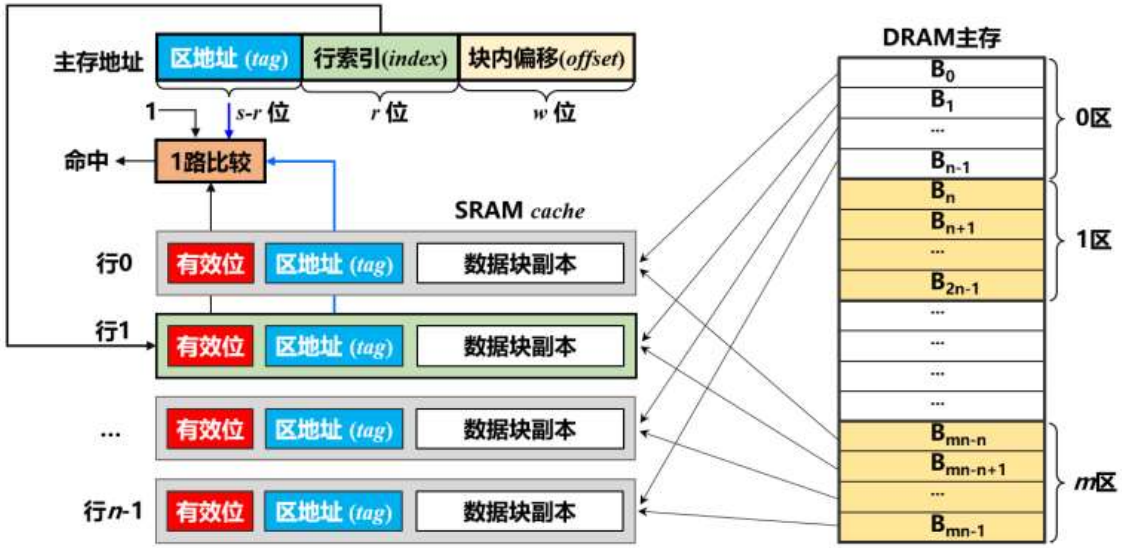
- 块映射灵活，一对多映射
- cache全部装满后才会出现块冲突
- 块冲突的概率低，cache利用率高
- 淘汰算法复杂
- 命中率高

# 直接相联映射

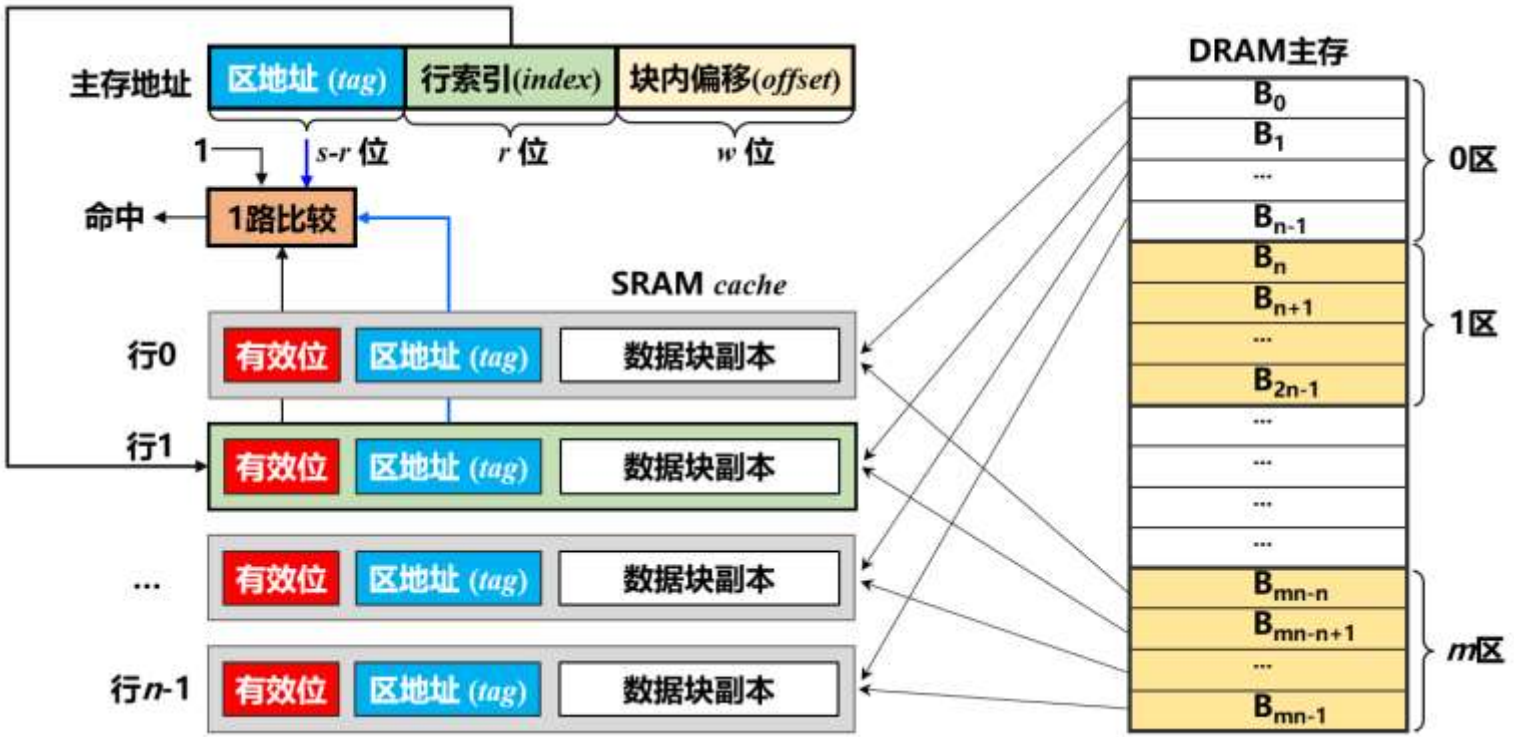
- 直接相联映射中每一个主存块地址只能映射到cache 中**固定的行块**

**cache块号i， 共n块，主存块号j，i=j mod n**

- 以上规则等效于将主存按照 cache 大小进行分区，每个分区中包含的块数与 cache 的行数相同
- 主存地址可细分为**区地址(tag)**、**区内行索引(index)**、**块内偏移(ofset)**三部分，这里 index字段就是数据块映射到 cache 中的行号，和上式中的余数完全相同。

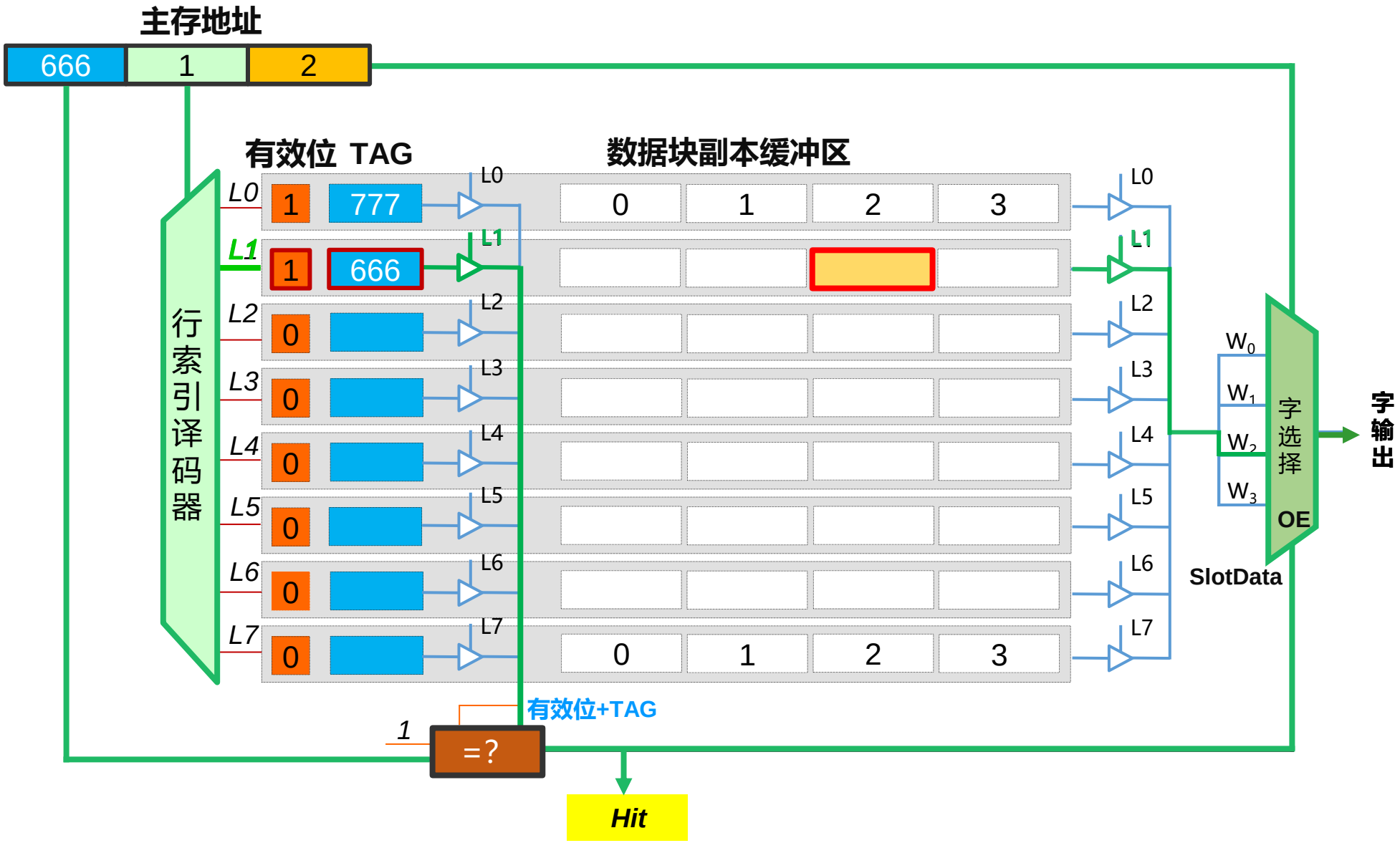


# 直接相联映射



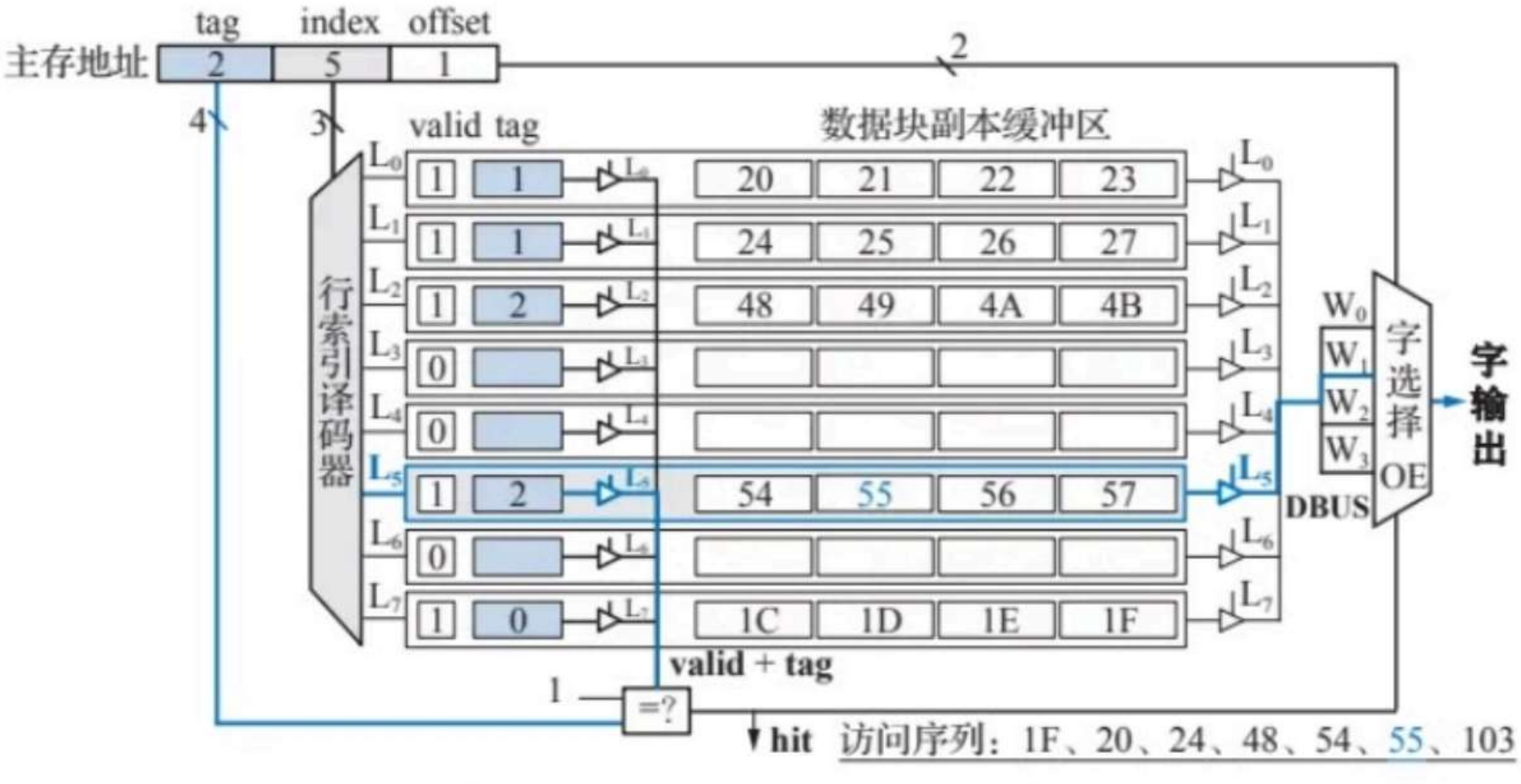

- cache块大小=2^w字节
- cache行数n=2^r
- 主存容量=2^(s+w)字节，如果不考虑脏数据位和淘汰计数
- cache的实际容量为n(1+s-r+8x2^w)位。
- 由于主存块只能放置在 index 对应的 cache 行中，因此直接相联映射并不需要全相联查找，查找表不需要存放在相联存储器中，各 cache 行中的元数据信息(tag,valid,dirty)只需要和数据块副本存放在一起即可

# 直接相联映射逻辑实现（构造观）



# 直接相联映射载入过程

Cache 8行，块大小4W，主存$2^9$ W　Cache总容量=（valid+标记位+标志位+副本容量）*总行数




**读访问序列**

| |
|---|
| 1F |
| 20 |
| 24 |
| 48 |
| 54 |
| 55 |
| 103 |

| 访问序列 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 十六进制 | 1F | 20 | 24 | 48 | 54 | 55 | 103 |
| 二进制 | 0000 111 11 | 0001 000 00 | 0001 001 00 | 0010 010 00 | 0010 101 00 | 0010 101 01 | 1000 000 11 |
| ( tag,index,offset ) | (0,7,3) | (1,0,0) | (1,1,0) | (2,2,0) | (2,5,0) | (2,5,1) | (8,0,3) |
| cache 行 | 7 | 0 | 1 | 2 | 5 | 5 | 0 |
| 访问情况 | 载入 | 载入 | 载入 | 载入 | 载入 | 命中 | 替换 |

- 假设初始状态为空
- 第1次主存访问地址为1F，第7行中有效位 valid=0，访问缺失，载入IF所在的主存数据块(1C~1F)到 cache 的第7行;同时将主存区地址0作为标记存放在第7行的标记字段中，并将有效位valid置1，方便后续查找。
- 第2~5次访问均为缺失状态，分别将对应主存数据块载入第0、1、2、5行中。
- 第6次访问的主存字地址为55，行索引index=5，第5行的标记字段与主存地址标记字段相同，且valid=1，访问命中，由块内偏移ofset=1选择 cache 行中的 W,输出。
- 第7次访问的主存地址为103，行索引index=0，而第0行的有效位 valid=1，表明该行已经包含数据，但标记字段tag=1，和当前地址的tag字段不相符，数据缺失。需要载入 103 所在的主存块到第0行中，并将原有数据替换。

# 直接相联应用场合

- 块映射速度快，一对一映射，无须查表
  - 利用索引字段直接对比相应标记位即可
  - 查找表可以和副本一起存放，无需相联存储器
- cache容易冲突，cache利用率低
- 淘汰算法简单
- 命中率低，适合大容量cache

# 组相联映射

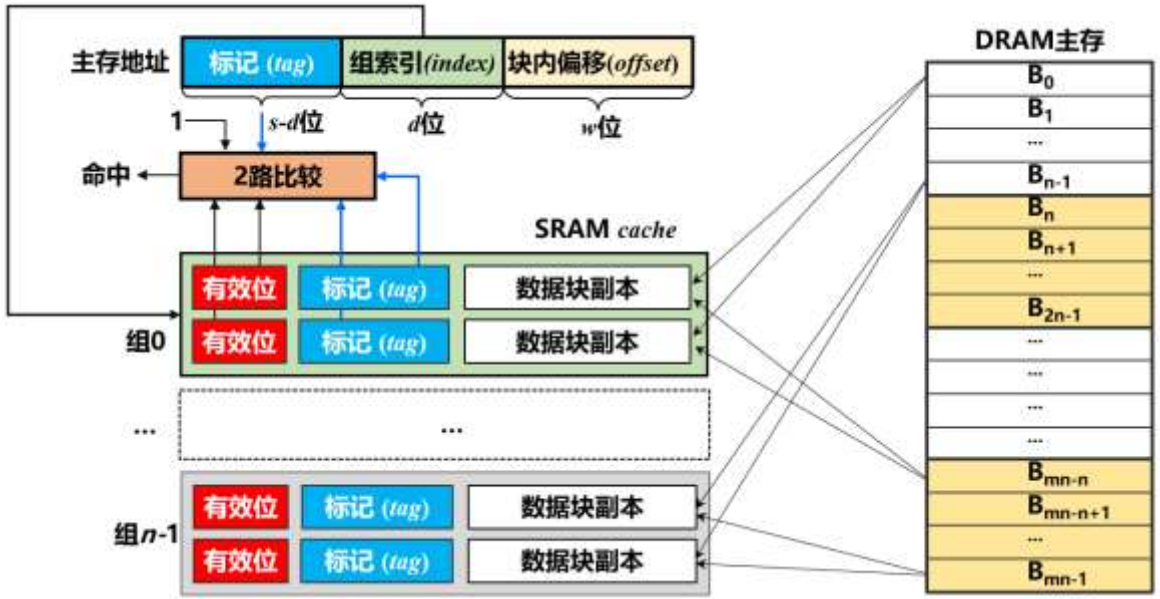
- 全相联映射命中率高，但查找硬件成本高
- 直接相联映射查找成本低但命中率低
- 组相联映射是直接相联映射和全相联映射两种方式的折中，既能提高命中率，又能降低查找硬件的开销。

# 组相联映射

- 组相联映射：cache分成固定大小的组，每组有k行，称为k-路组相联，主存数据块**首先采用直接相联映射**的方式定位到 cache 中固定的组，然后**采用全相联映射**的方式映射到组内任何cache 行

cache组号=主存块号mod(cache 组数)

# 组相联映射

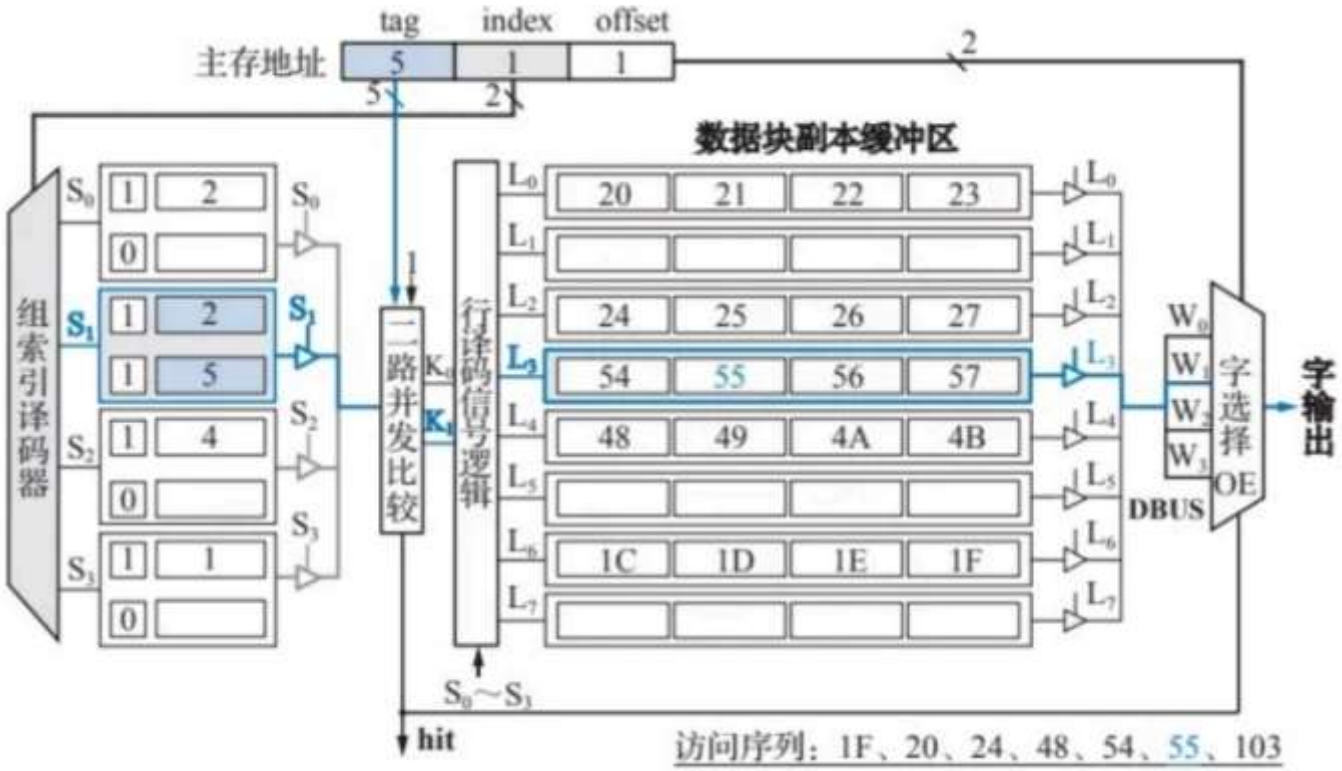



- 主存地址可细分为标记字段(tag)、组索引(index)、块内偏移(ofset )3部分，这里的组索引实际就是上式中的余数。

  cache组号=主存块号 mod (cache 组数)

- cache块大小=2^w字节
- 组数n=2^d
- 主存容量=2^(s+w)字节
- cache容量=2nx(1+s-d+8x2^w)位

# 组相联映射硬件实现



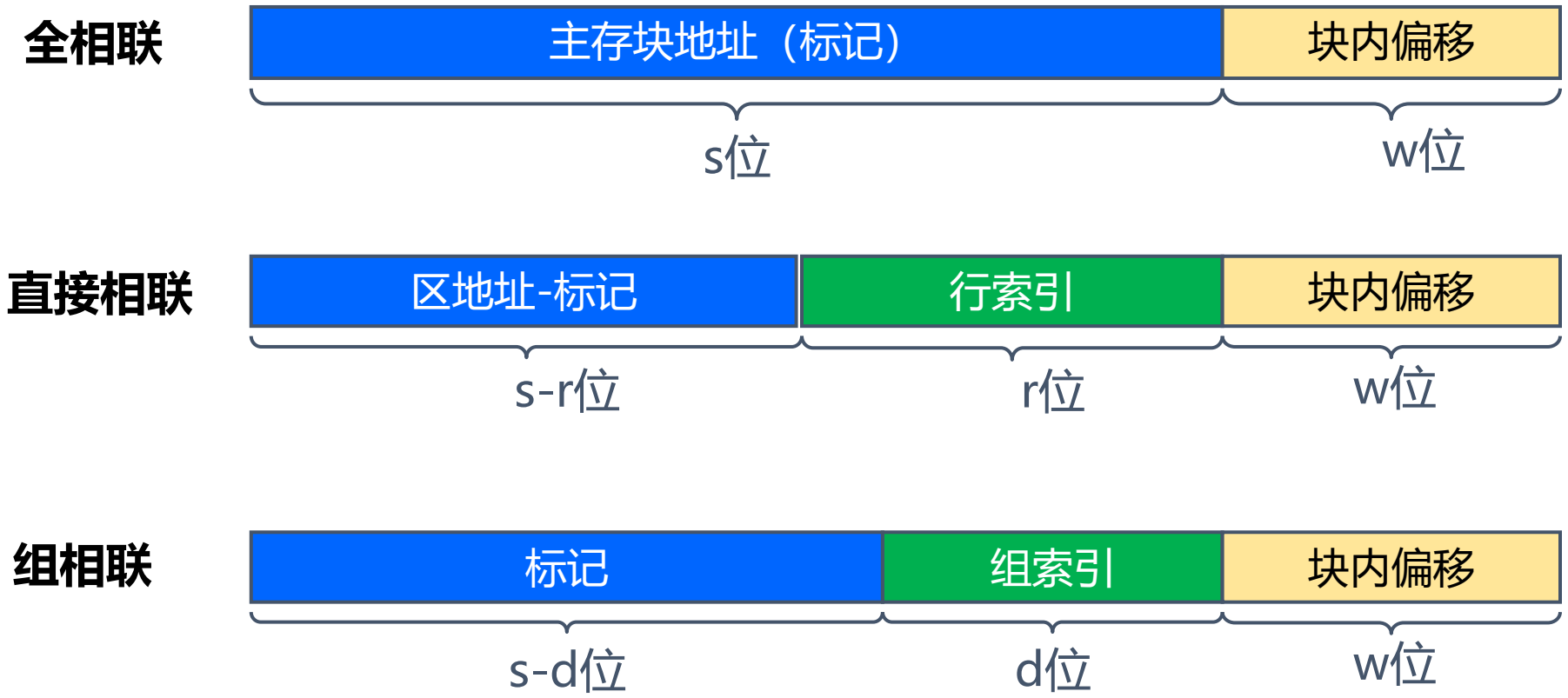

| 访问序列 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 十六进制 | 1F | 20 | 24 | 48 | 54 | 55 | 103 |
| 二进制 | 00001 11 11 | 00010 00 00 | 00010 01 00 | 00100 10 00 | 00101 01 00 | 00101 01 01 | 10000 00 11 |
| (tag,index,offset) | (1,3,3) | (2,0,0) | (2,1,0) | (4,2,0) | (5,1,0) | (5,1,1) | (10,0,3) |
| cache 组 / 行 | $S_3$/6 | $S_0$/0 | $S_1$/2 | $S_2$/4 | $S_1$/3 | $S_1$/3 | $S_0$/1 |
| 访问情况 | 载入 | 载入 | 载入 | 载入 | 载入 | 命中 | 载入 |

- 第1次主存访问地址为1F，第3组中两个 cache 行的有效位valid均为0，访问缺失，载入1F所在的主存数据块(1C~1F)到 cache第3组的第一个空行中(cache的第6行);同时将标记字段tag=1存放在第6行的标记字段中并将有效位 valid 置1，方便后续查找。
- 第2~5次访问均为缺失状态，分别将对应主存数据块载入cache的第0、2、4、3 行中。
- 第6次访问的组索引字段index=1，第1组的标记信息和有效位信息与主存地址标记字段进行二路并发比较，组内第1行相符，其有效位为1，访问命中，K1输出为1，经过行选通信号逻辑输出L3=1，选中第3行数据输出，由块内偏移offset=1选择 cache 行中的 W1字输出。
- 第7次访问的主存地址为103，组索引index=0，第0组的第0行数据有效位valid=1，但标记字段和主存地址标记字段不相符，第0组的第1行有效位 valid=0，为空行，数据缺失，将103所在的主存块载入第0组的第1行中即可。

# 组相联存储器硬件开销

- SRAM
  - 存放数据副本
- 多个相联存储器共享一个多路比较器
  - 相对于全相联　多路比较器复杂度低
  - 查找表表项内容（valid位，标记，dirty位，置换标记位）
  - 相联存储器总容量
    - cache行数*（1+标记宽度+1+置换标记位）
- 片外缓存如果查找表在CPU内部?
  - 查找表中必须增加cache行地址？　why

# 不同映射方式主存地址划分



# 组相联应用场合

- 容量小的cache可采用全相联映射或组相联映射
  - Pentium CPU  L1 L2 cache
- 容量大的可采用直接映射方式
  - 查找速度快，命中率相对低
  - 但cache容量大可提高命中率
  - 块设备缓存

# 例题1

- 某计算机字长32位，采用直接映射Cache,主存容量4MB，Cache数据存储体容量为4KB，字块长度为8个字。
  1. 画出直接映射方式下主存地址划分情况。
  2. 设cache初始状态为空，若CPU顺序访问0-99号单元，并从中读出100个字，假设主存一次读一个字，并重复此顺序10次，请计算cache命中率。
  3. 如果cache的存取时间是2ns，主存访问时间是20ns，平均访问时间是多少。
  4. Cache-主存系统访问效率。

# 例题2

- 主存地址空间大小为256MB，按字节编址。指令数据Cache，均有8行，Cache行大小为64B，数据Cache直接相联。现有两功能相同的程序A，B，其伪代码如下所示：
- 假定int型数据为32位补码，程序编译时i,j,sum均分配在寄存器中，数组a按行优先方式存放，首地址为320（十进制）。

```
int a[256][256];
for (i = 0; i < 256; i++)
 for (j = 0; j < 256; j++)
   sum += a[i][j];
```

程序A

```
int a[256][256];
for (j = 0; j < 256; j++)
  for (i = 0; i < 256; i++)
    sum += a[i][j];
```

程序B

**1）若不考虑用于Cache一致性维护和替换算法的控制位，数据cache的总容量是多少?**

**2）数组元素a[0][31],a[1][1]所在主存块对应的cache行分别是多少，行号从零开始。**

**(3)程序A，B的数据访问命中率各是多少？那个程序的执行时间更短?**

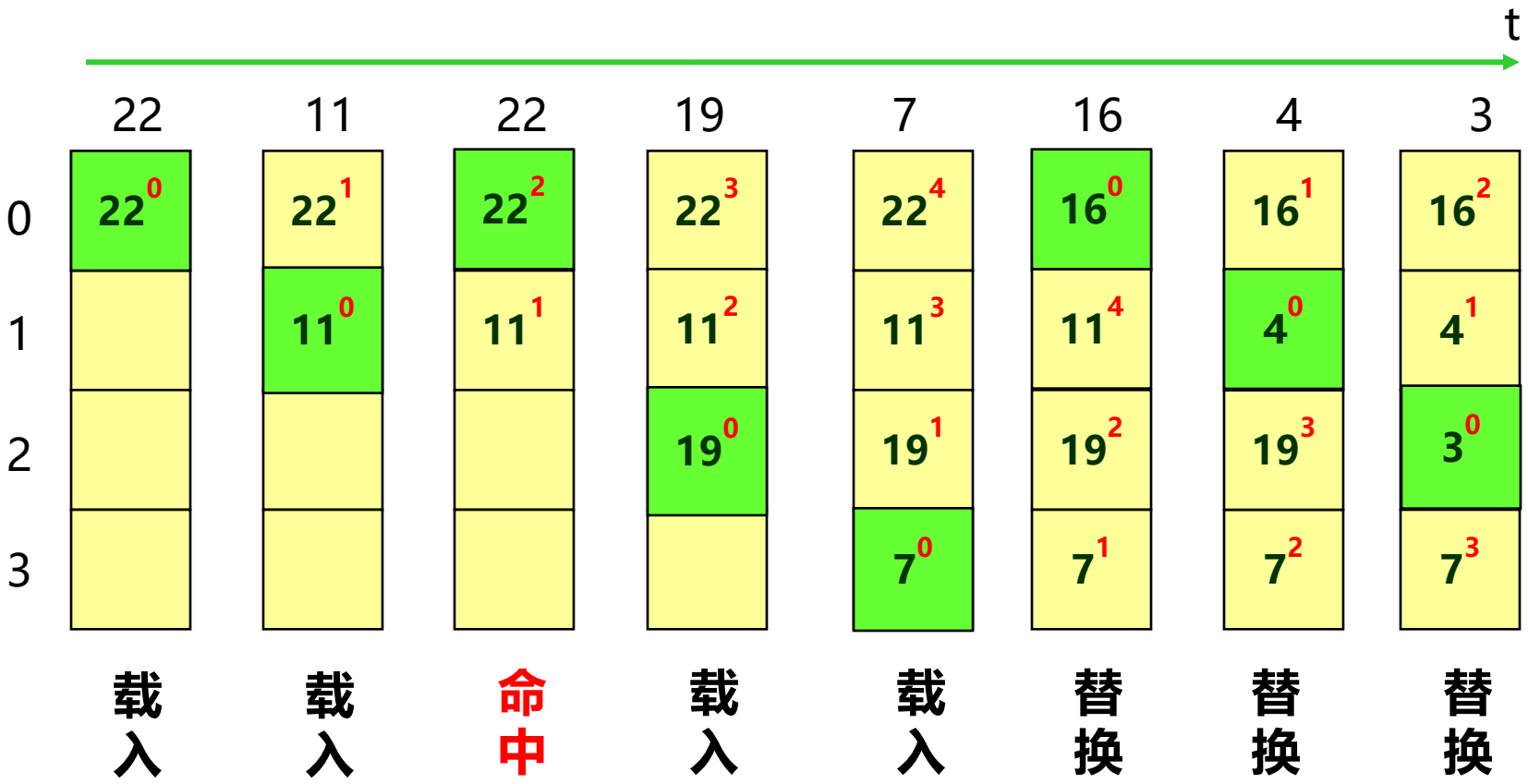
# cache关键技术

- **数据查找　Data Identification**
  - 如何判断数据在cache中
- **地址映射　Address Mapping**
  - 主存数据如何放置到Cache行/槽中
- **替换策略　Placement Policy**
  - Cache满后如何处理
- **写入策略　Write Policy**
  - 如何保证cache与memory的一致性

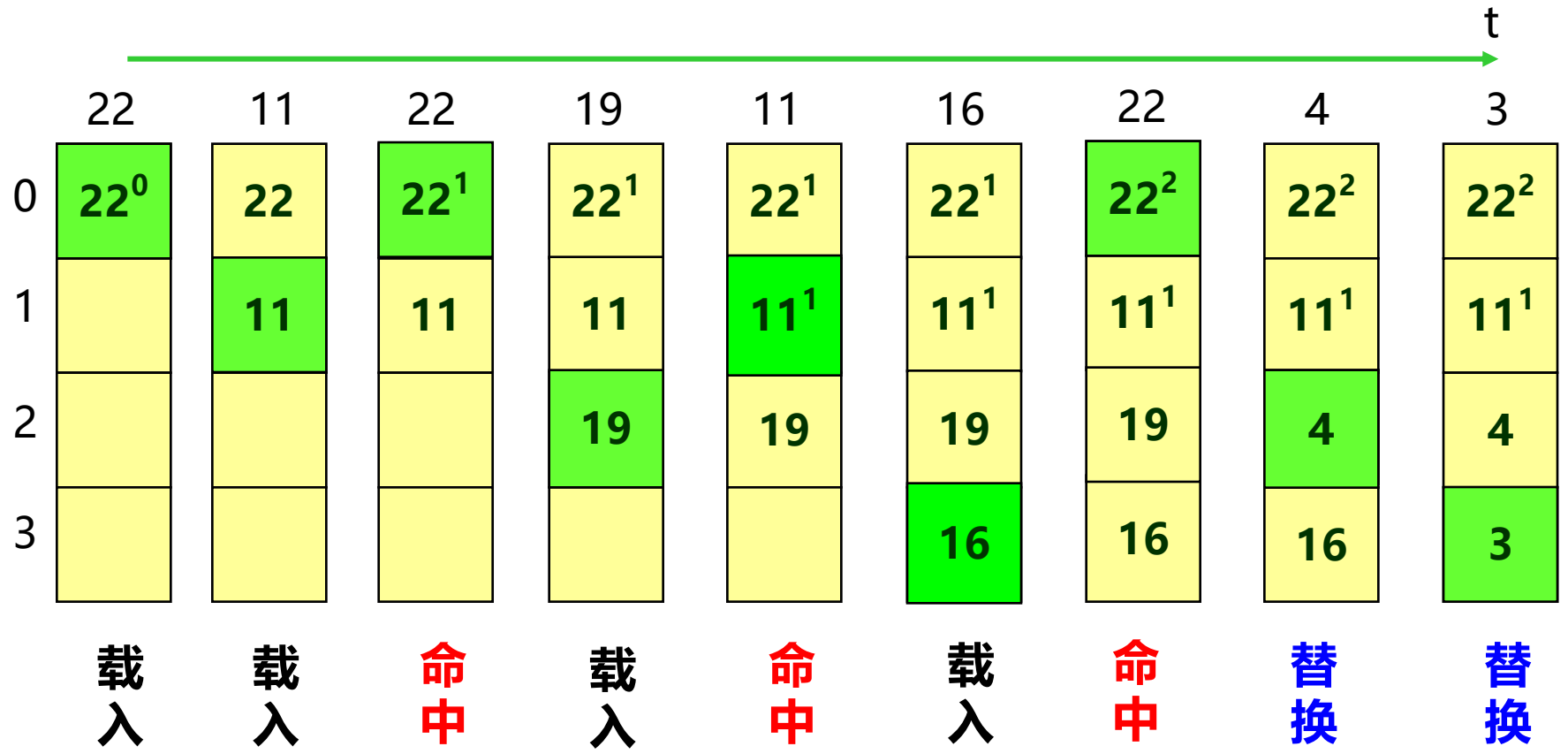
# 替换策略与写操作策略

- **替换策略**
  - 先进先出法
  - 最近最不经常使用方法---LFU
  - 近期最少使用法--- LRU
  - 随机替换法

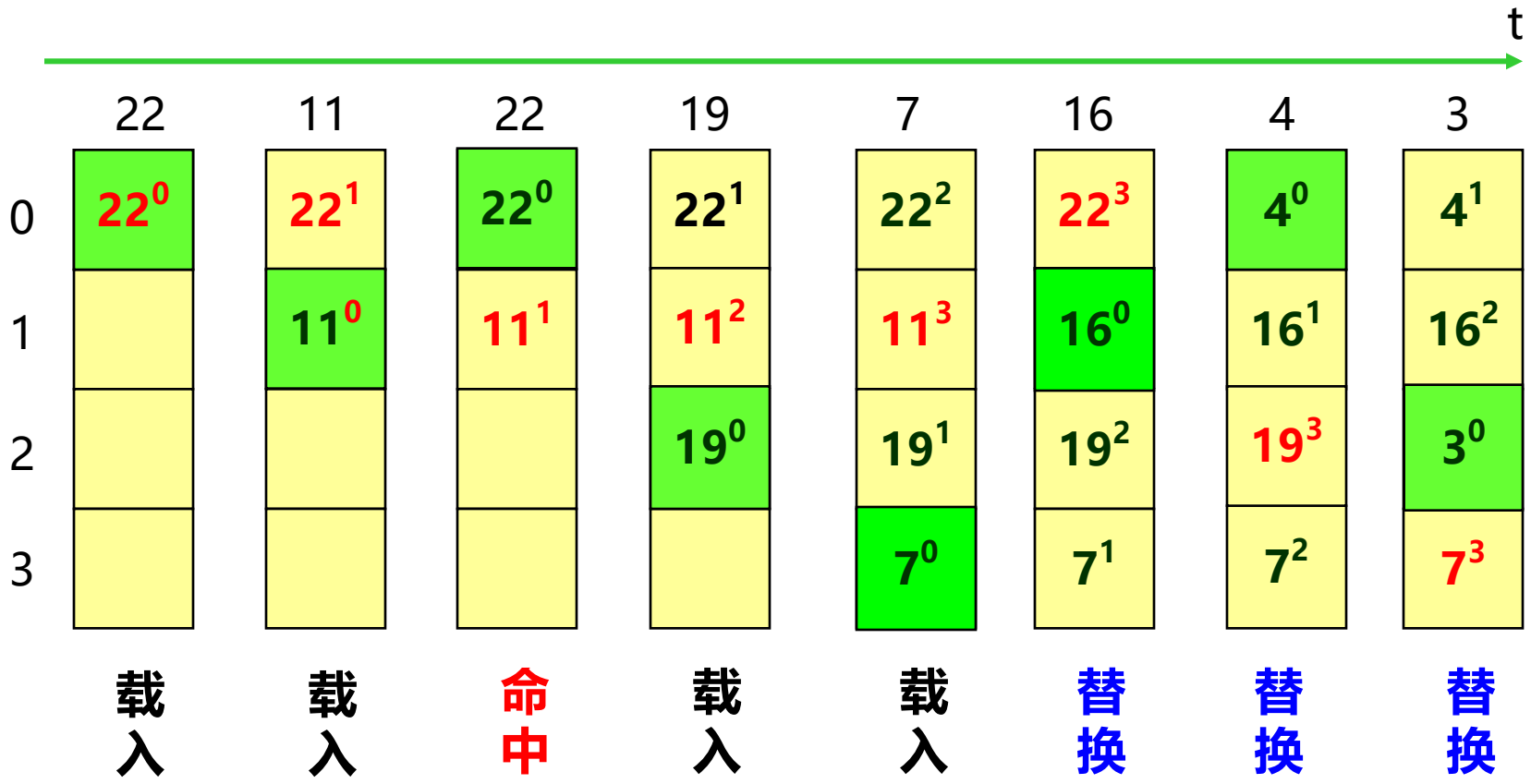
# Cache先进先出替换策略(FIFO)



# Cache最不经常使用算法(LFU)



# Cache近期最久未使用算法(LRU)

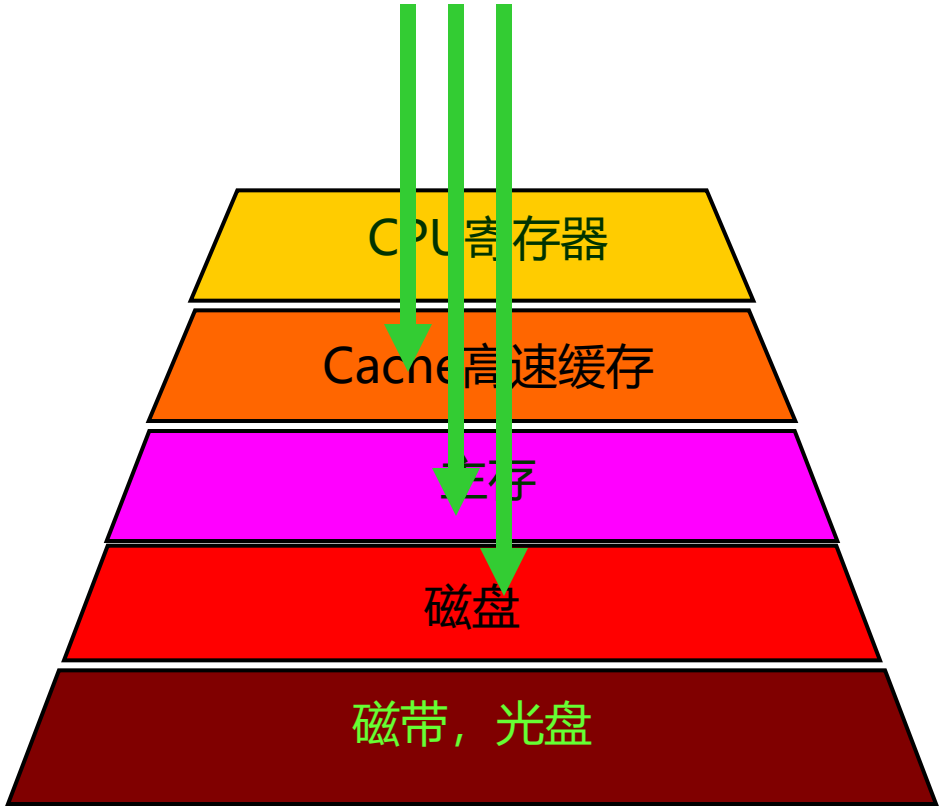


# cache关键技术

- **数据查找　Data Identification**
  - 如何判断数据在cache中
- **地址映射　Address Mapping**
  - 主存数据如何放置到Cache行/槽中
- **替换策略　Placement Policy**
  - Cache满后如何处理
- **写入策略　Write Policy**
  - 如何保证cache与memory的一致性

# 写入策略

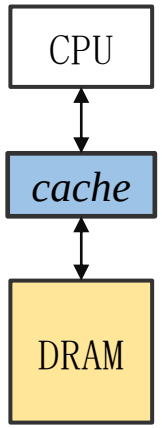
- 写回法(write back)
- 写穿法 (write through)



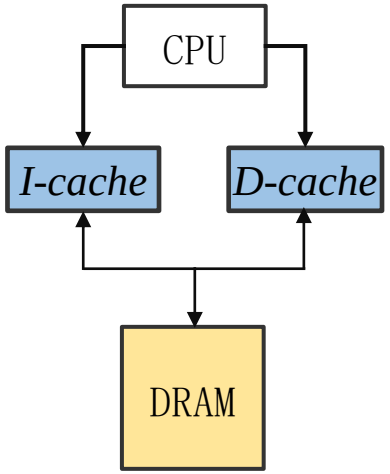
# 4.5 高速缓冲存储器

- 4.5.1 cache工作原理
- 4.5.2 程序局部性
- 4.5.3 cache的基本概念
- 4.5.4 cache读写流程与关键技术
  - 相联存储器、地址映射、 替换算法、 写入策略
- 4.2.9 cache应用

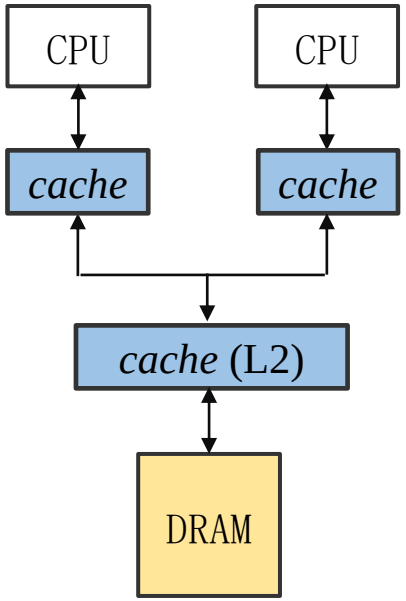
# cache实际应用



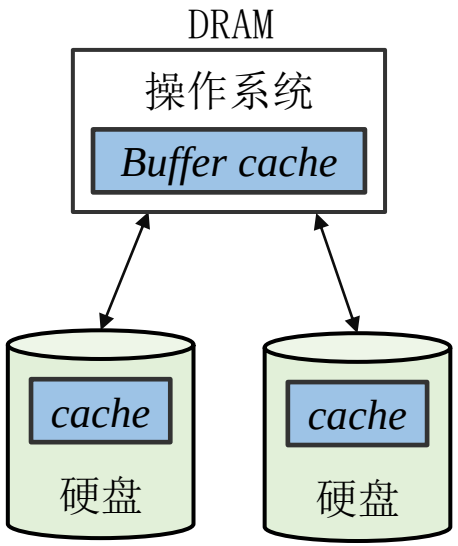

(a) 统一*cache*



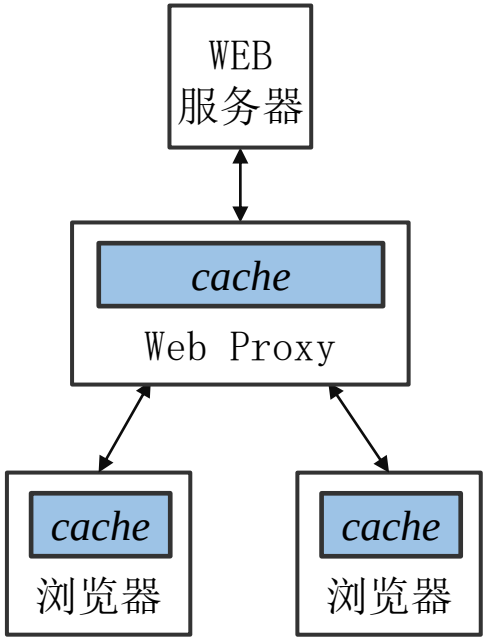

(b) 分离*cache*




(c) 多级*cache*




(d) Buffer *cache*和硬盘缓存芯片




(e) WEB *cache*

# 本章主要内容

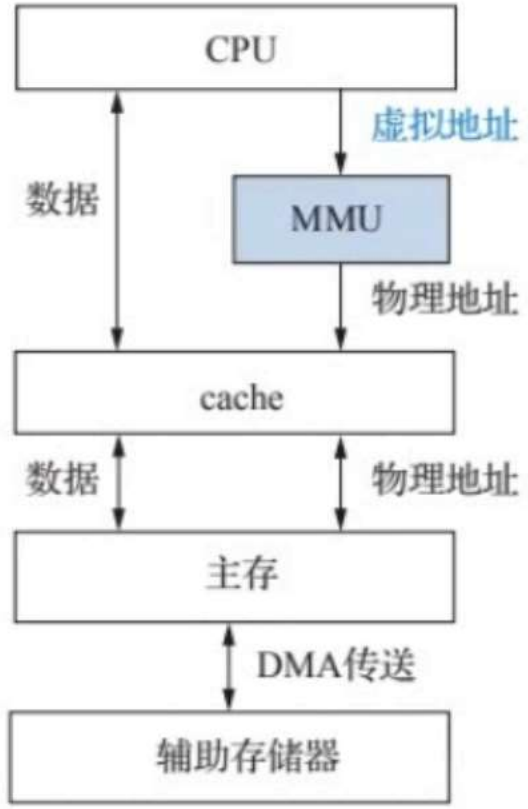
- 4.1 存储器概述
- 4.2 半导体存储器
- 4.3 主存的组织及与CPU的连接
- 4.4 并行主存系统
- 4.5 高速缓冲存储器
- 4.6 虚拟存储器

# 4.6 虚拟存储器

- 4.6.1 虚拟存储器工作原理
- 4.6.2 虚拟存储器地址映射与变换
- 4.6.3 页式虚拟存储器

# 虚拟存储器

- 如何让更多的程序在有限的主存空间中高效并发运行且互不干扰是虚拟存储器需要解决的问题?
- 虚拟存储器：
  - 英国曼切斯特大学的基尔伯恩(Kilbum)等人于1961年提出
  - 目前几乎所有的计算机中都采用了虚拟存储器系统。
  - 在主存和辅存之间增加部分软件和必要的硬件，使**辅存和主存构成一个有机的整体**，就像一个单一的、可供CPU直接访问的大容量主存一样。
  - 用虚拟存储器提供的地址(虚拟地址)进行编程，在实际主存空间大小没有增加的情况下，编程不再受实际主存空间大小的限制



# 虚拟存储器

- 虚存空间是靠辅存（磁盘）来支持的，用户感觉到的是一个速度接近主存而容量极大的存储器
- 利用了程序的局部性，采用按需加载的方式加载程序代码和数据。
  - 加载程序时并不直接将程序和代码载入主存，而仅仅在相应的虚拟地址转换表(段表、页表)中登记虚拟地址对应的磁盘地址
  - 程序执行并访问该虚拟地址对应的程序或数据时，会产生缺页异常，操作系统会调用异常处理程序并载入实际的程序和代码

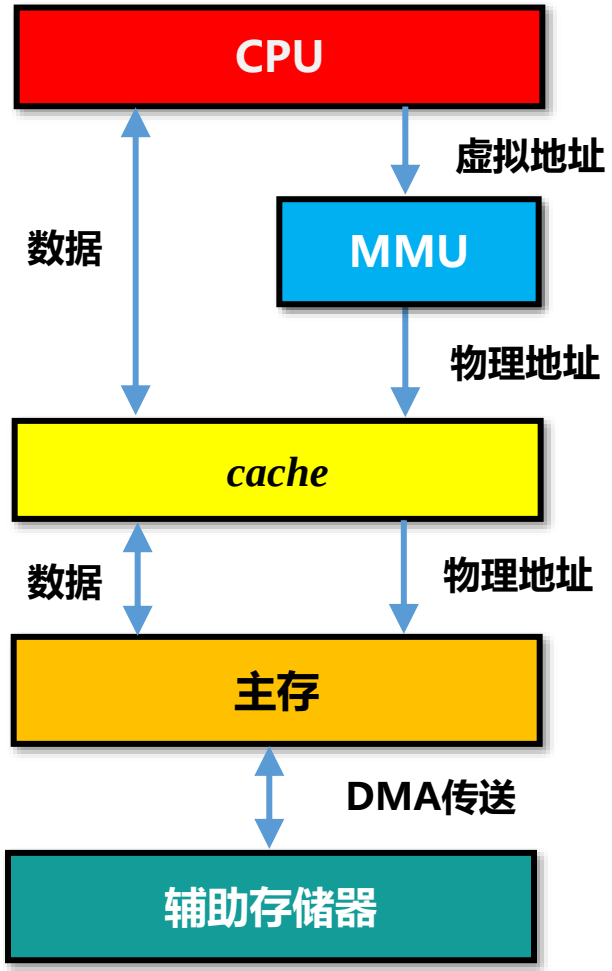
# 4.6 虚拟存储器

- 4.6.1 虚拟存储器工作原理
- 4.6.2 虚拟存储器地址映射与变换
- 4.6.3 页式虚拟存储器

# 虚拟存储器

- 虚拟存储器的地址空间
  - 第一种是虚拟地址空间，也称为虚拟空间或虚地址空间，它是程序员用来编写程序的地址空间
  - 第二种地址空间是主存的地址空间，也称物理地址空间或实地址空间
  - 第三种地址空间是辅存地址空间，也就是磁盘存储器的地址空间
- 虚拟地址(虚地址)、主存物理地址(实地址)和磁盘存储器地址(磁盘地址或辅存地址)。

# 层次性结构




- 存储管理控制部件MMU(Memory Management Unit)：找出虚拟地址和物理地址之间的对应关系，并判断这个虚拟地址对应的内容是否已经在主存中。
- 如果已经在主存中，则通过MMU将虚拟地址转换成物理地址，CPU直接访问主存单元
- 如果不在主存中，则把包含这个字的一页或一个程序段(与虚拟存储器的类型有关)调入主存，并在 MMU 中填写相关的标记信息。

# 地址映射

- 虚拟存储器中的地址映射
  - 虚拟地址空间映射到主存空间
  - 利用虚拟地址访问的内容按照某种规则从辅存装入主存储器中
  - 并建立虚地址与实地址之间的对应关系
  - 地址转换则是在程序被装入主存后
  - 在实际运行时，把虚拟地址转换成实地址或磁盘地址，以便 CPU 从主存或磁盘中读取相应的信息。

# 与CPU cache的相似之处

- 将程序中常用的部分驻留在高速存储器
  - 程序载入按需载入（不是全部载入）
  - 高速存储器分块或者分页（粒度问题）
  - 主存空间满，将不常用程序或数据**淘汰**或**交换**到辅存中
- 数据的换入换出由硬件或操作系统完成，对用户透明
- 利用程序局部性，使存储系统的性能接近于高速存储器；价格接近于低速存储器，并扩充主存容量

# 与CPU cache的差异

- 虚存用于扩大主存容量，CACHE用于加速主存性能
- 虚存中未命中性能损失远大于CACHE系统
  - 全相联提升命中率
  - 更大的交换单位页
  - 近似LRU算法（CLOCK算法）
- 虚存由硬件和OS联合管理，CACHE由硬件管理

# 虚拟存储器分类

- 根据虚拟存储器中对主存逻辑结构划分的粒度不同，虚拟存储器
  - 页式
  - 段式
  - 段页式

# 4.6 虚拟存储器

- 4.6.1 虚拟存储器工作原理
- 4.6.2 虚拟存储器地址映射与变换
- 4.6.3 页式虚拟存储器

# 页式虚拟存储器

- 以页(Page)为逻辑结构划分信息传送单位的虚拟存储器称为页式虚拟存储器。
- 在页式虚拟存储器中，虚拟空间和主存空间均被划分成固定大小的页。不同类型的计算机对页大小的划分不同，常见的页大小为 4KB，也有更大容量的页。

# 页式虚拟存储器

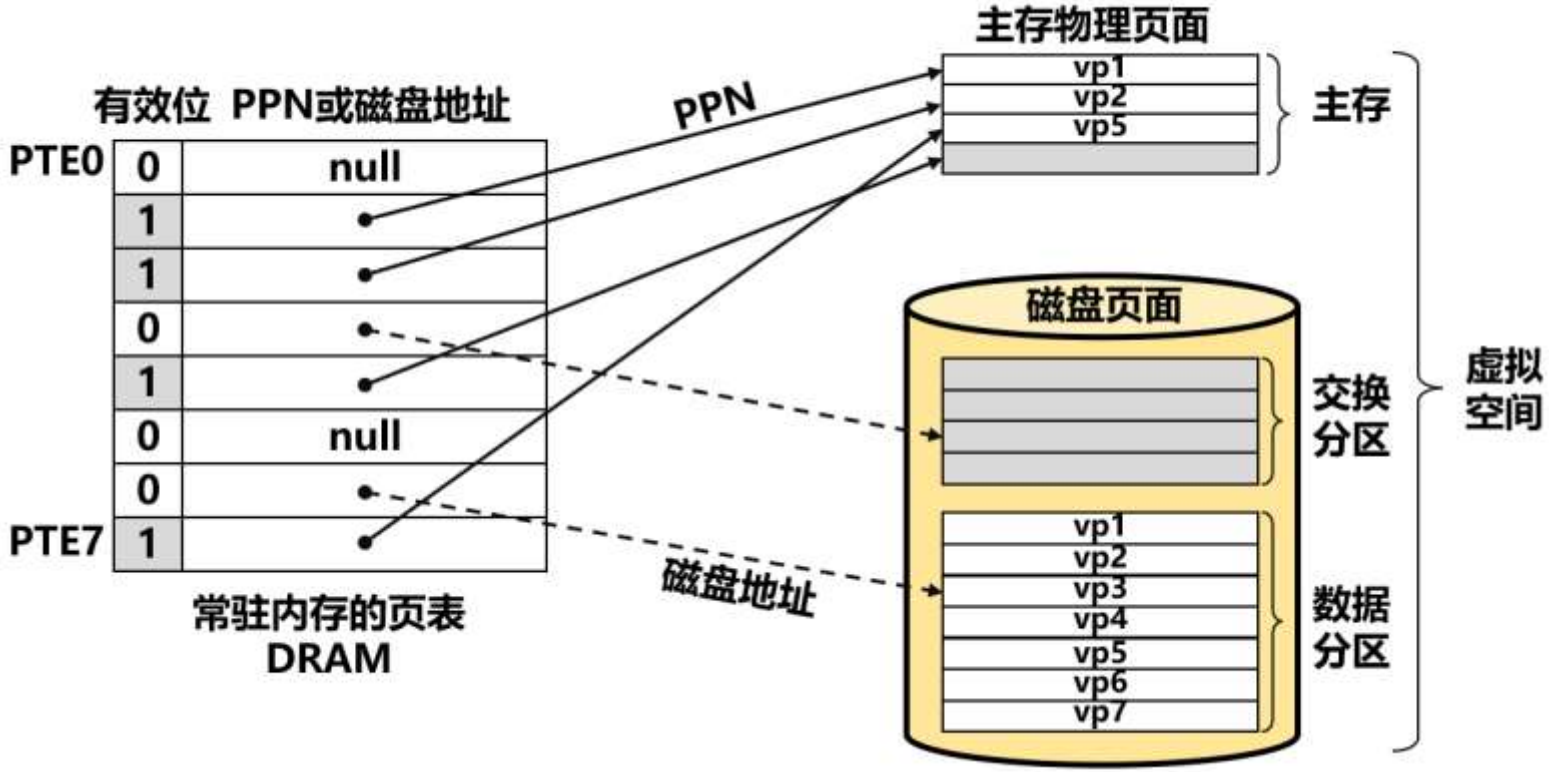
- 虚拟地址被划分成虚拟页号(VirtualPage Number，VPN)和虚拟页偏移(Virtual Page Ofset，VPO)
- 物理地址也被划分成物理页号(Physical Page Number,PPN)和物理页偏移(PhysicalPage Ofset，PPO)两部分，其中虚拟页号又称为虚页号，物理页号又称为页框号或实页号

# 页式虚拟存储器

- 虚拟地址到物理地址的映射本质上就是将 VPN 转换成对应的 PPN
- 页式虚拟存储器中虚拟地址与物理地址之间的转换是基于页表
  - 页表是一张保存虚拟页号 VPN 和物理页号 PPN对应关系的查找表，是一个由若干个表项组成的数组
  - 采用 VPN 作为索引进行访问，每一个表项主要包括有效位和物理页号，另外还包括修改位、使用位、权限位等信息

# 页式虚拟存储器

- 主存和磁盘都分为固定大小的页面
- 交换分区用于存放主存页面换出的动态修改数据，数据分区用于存储用户程序和数据



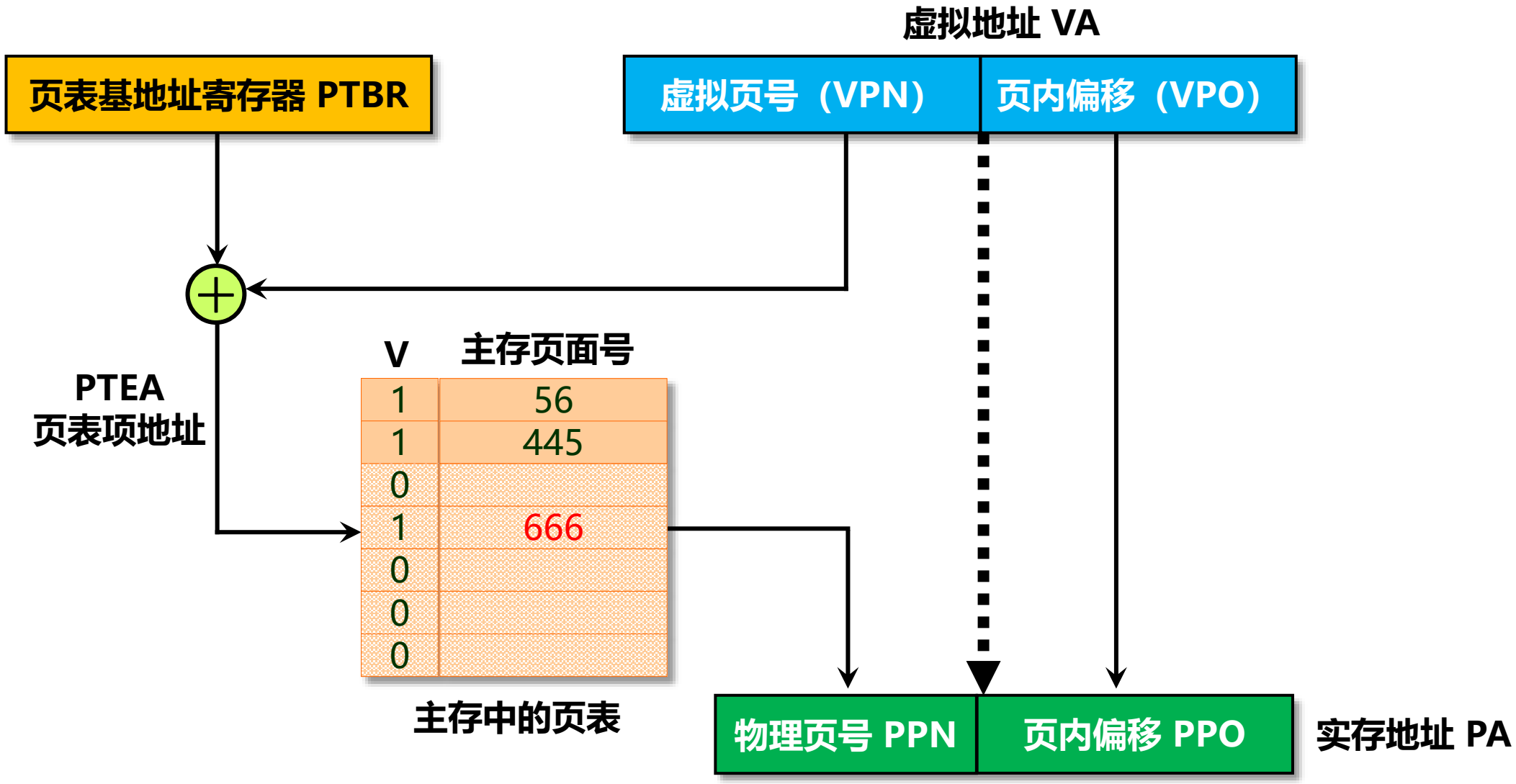
# 页式虚拟存储器

- VA( VirtualAddress):虚拟地址
- PA(Physical Address):物理地址
- PTE(Page Table Entry):页表项
- PTBR(Page Table Base Register,)：页表基址寄存器，页表主存中的首地址

# 页式虚拟存储器

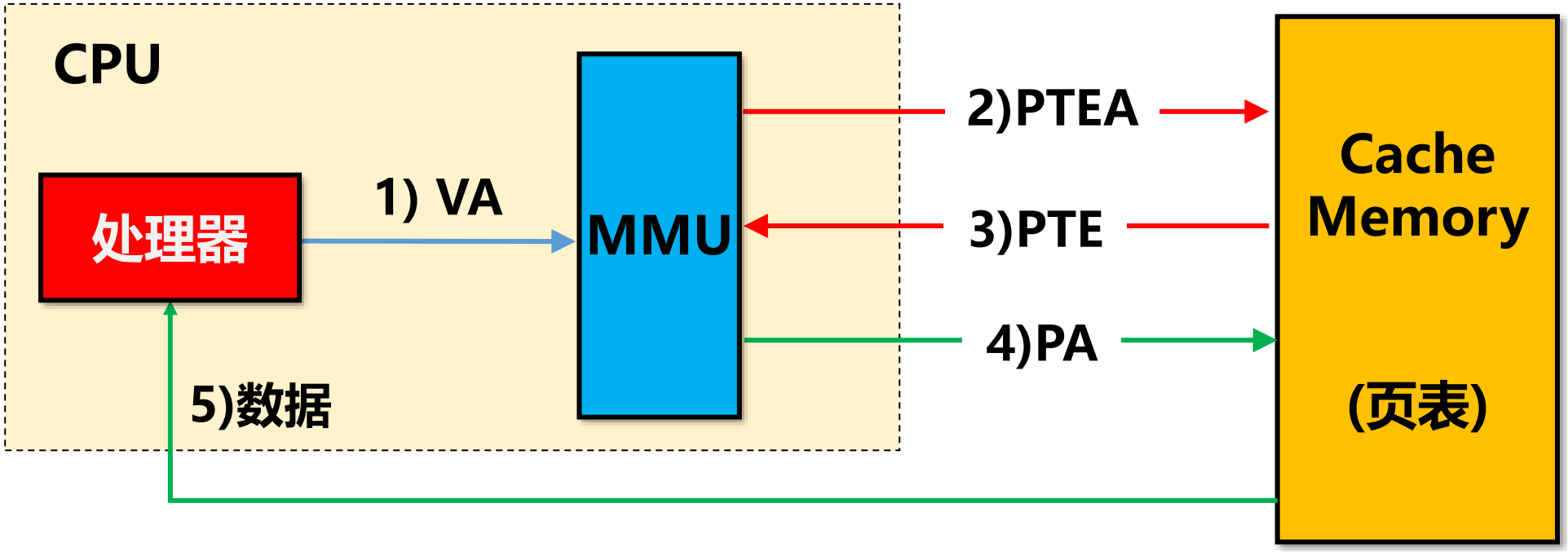
- 虚拟地址对应页表项中的有效位为1，表明当前页的数据在主存中，直接利用页表中的物理页地址 PPN与VPO一起生成物理地址即可访问主存数据。
- 有效位为0时，对应虚拟页可能是暂未分配页，如图中标记为nul 的页表项;没有标记为null的页表项则表示对应页在磁盘中，访问对应页时会触发缺页异常，由操作系统的缺页异常处理程序负责将磁盘上对应的页载入主存中，并同时更新页表项，方便后续访问

# 虚拟地址如何转换成物理地址

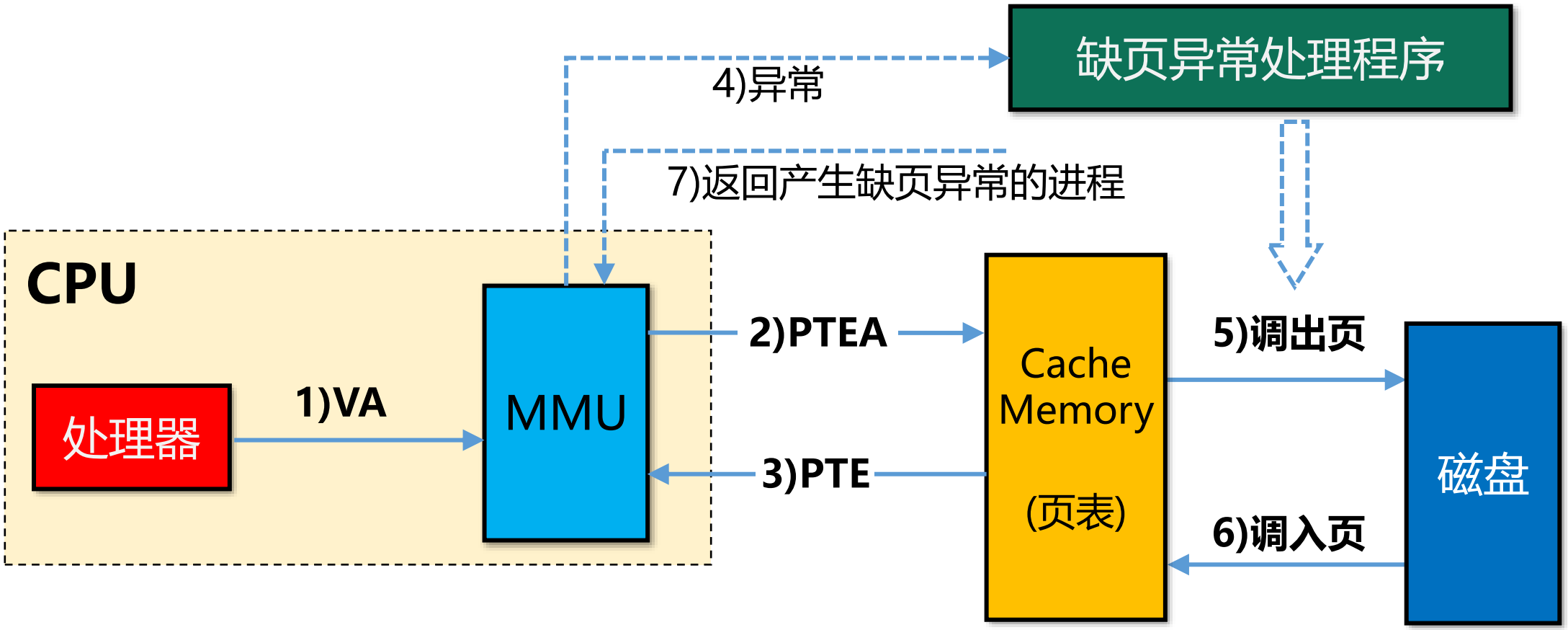



页表项地址 PTEA=PTBR+VPNx页表项字节大小

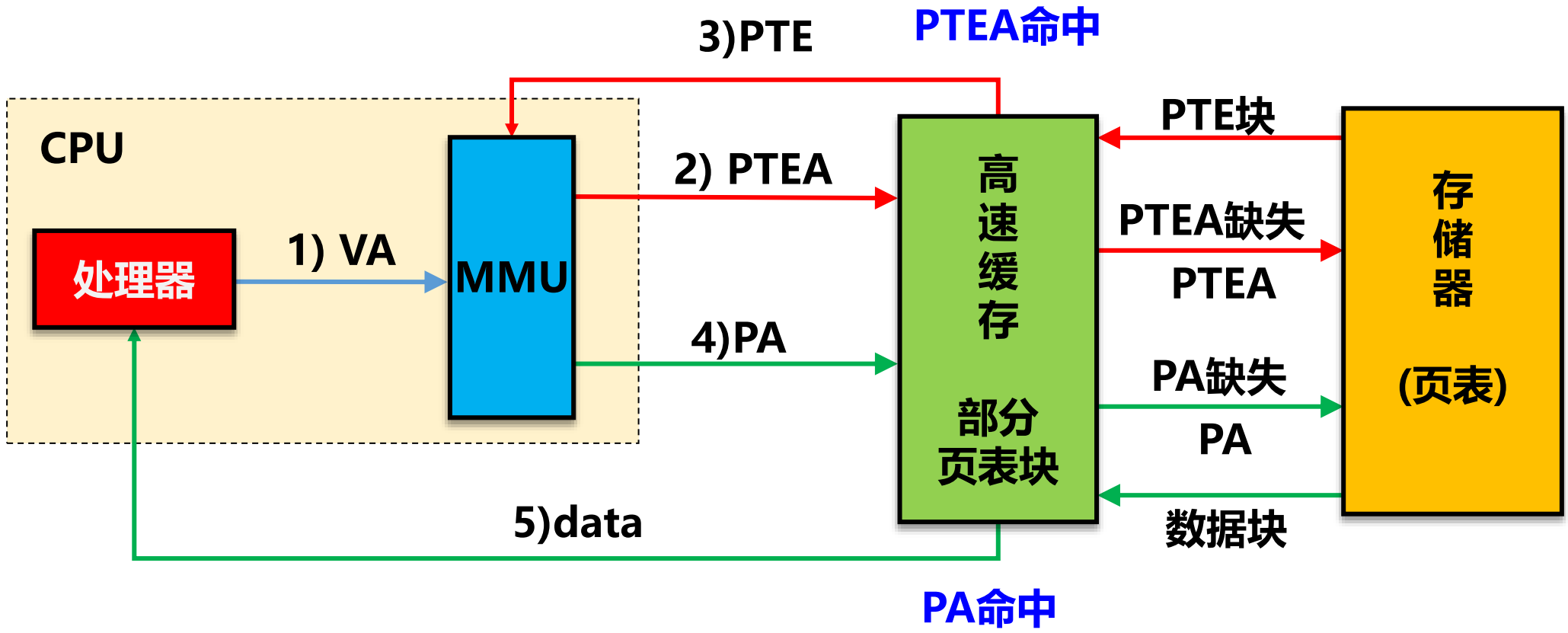
# 虚拟地址→物理地址　(页命中)



# 虚拟地址→物理地址（缺页）


缺页异常处理程序
4)异常
7)返回产生缺页异常的进程
CPU
处理器
1)VA
MMU
2)PTEA
3)PTE
Cache
Memory
(页表)
5)调出页
6)调入页
磁盘

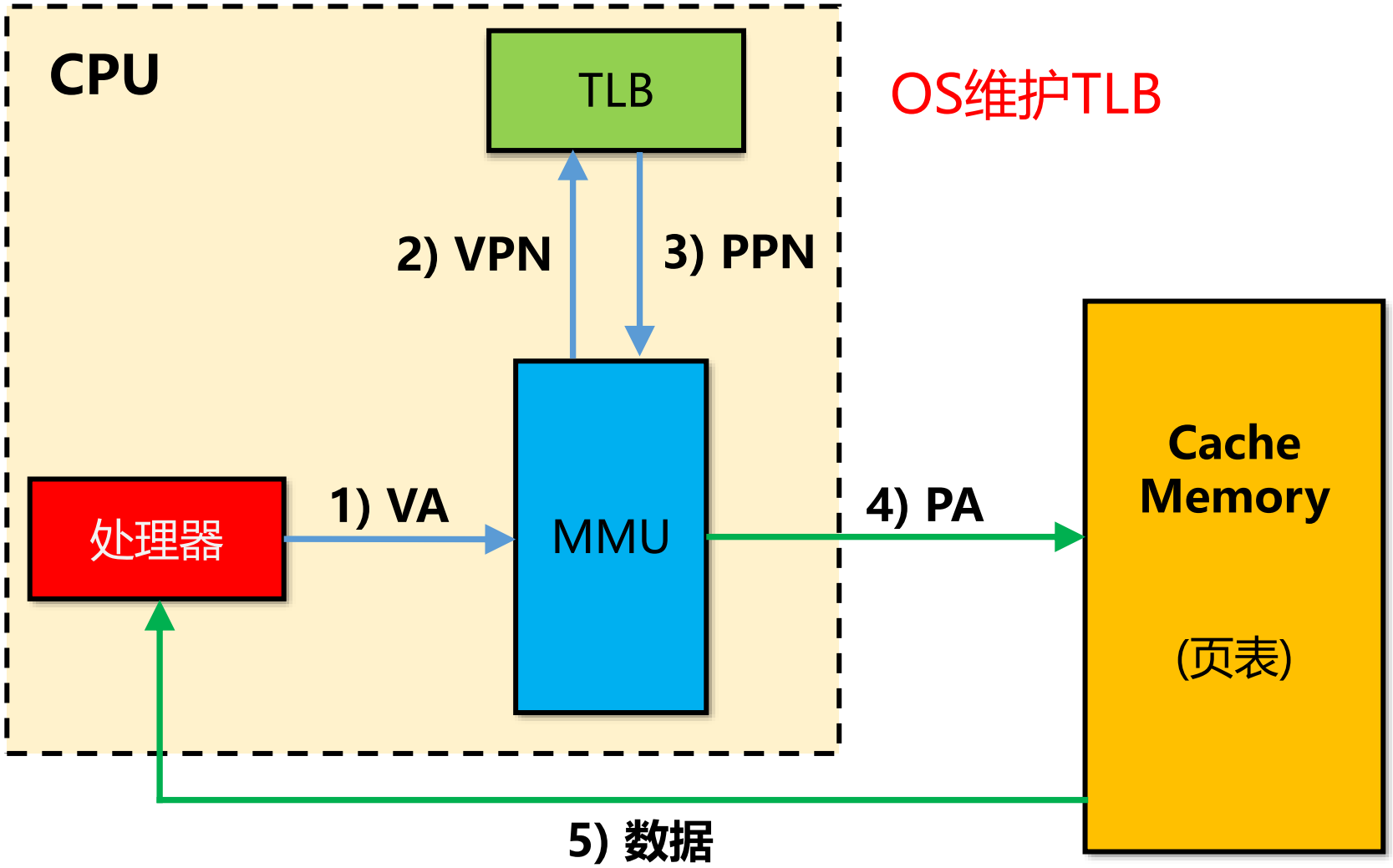
# 虚存，cache，主存层次



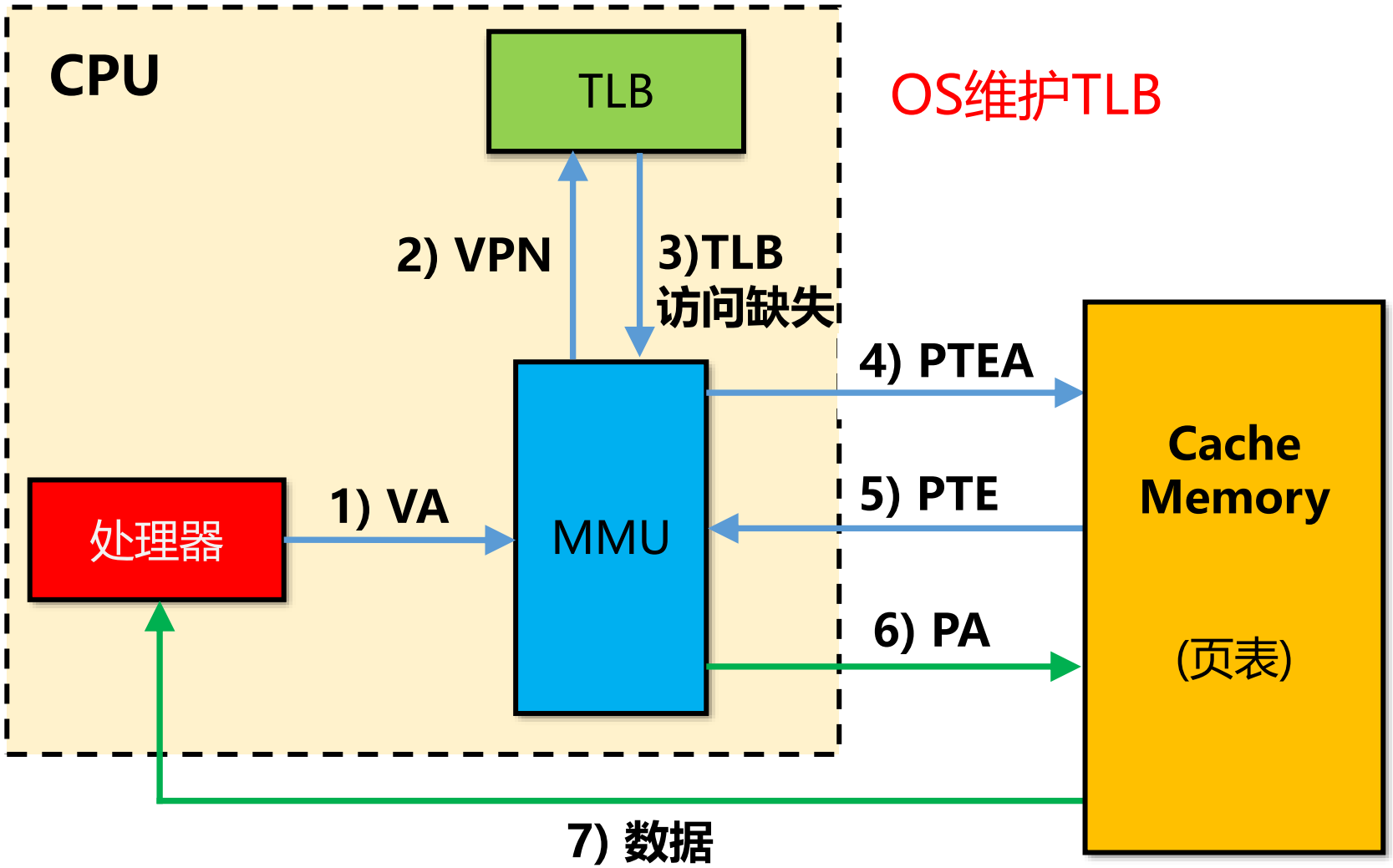
# TLB 加速虚拟存储器地址转换

- 转换旁路缓冲区(Translation Look-aside Bufer，TLB)
  - 缓冲经常访问的页表项 PTE
  - TLB 本质上就是一个容量较小的 cache，为提高查找速度，大多采用全相联或组相联方式，且采用随机替换算法。
- 当采用组相联方式时,按照组相联cache的地址划分方法,将虚页号划分成TLB标记(TLBT)和TLB索引(TLBI)两部分，以便于快速判断所要访问的页面是否在主存中

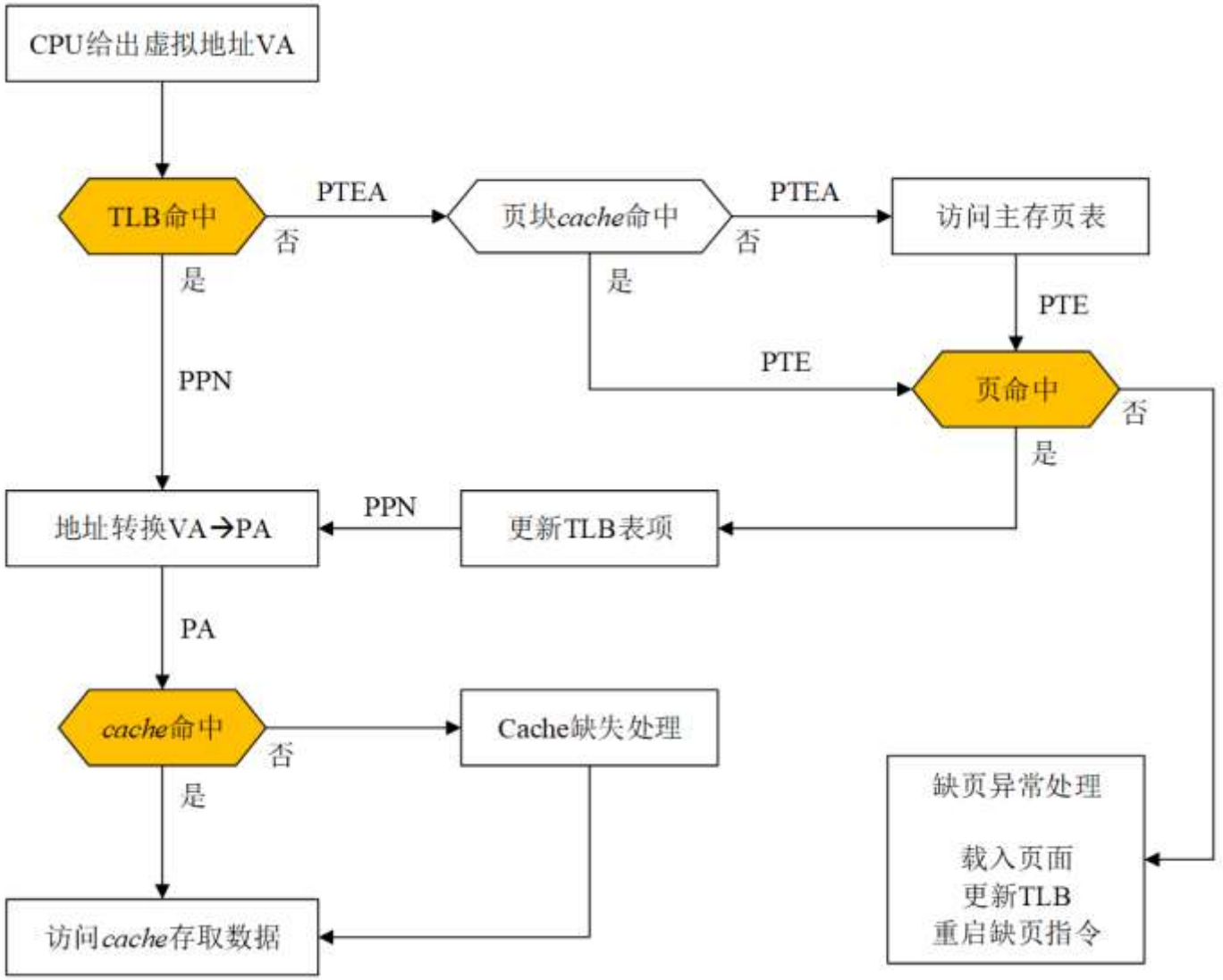
# TLB 命中



# TLB 缺失

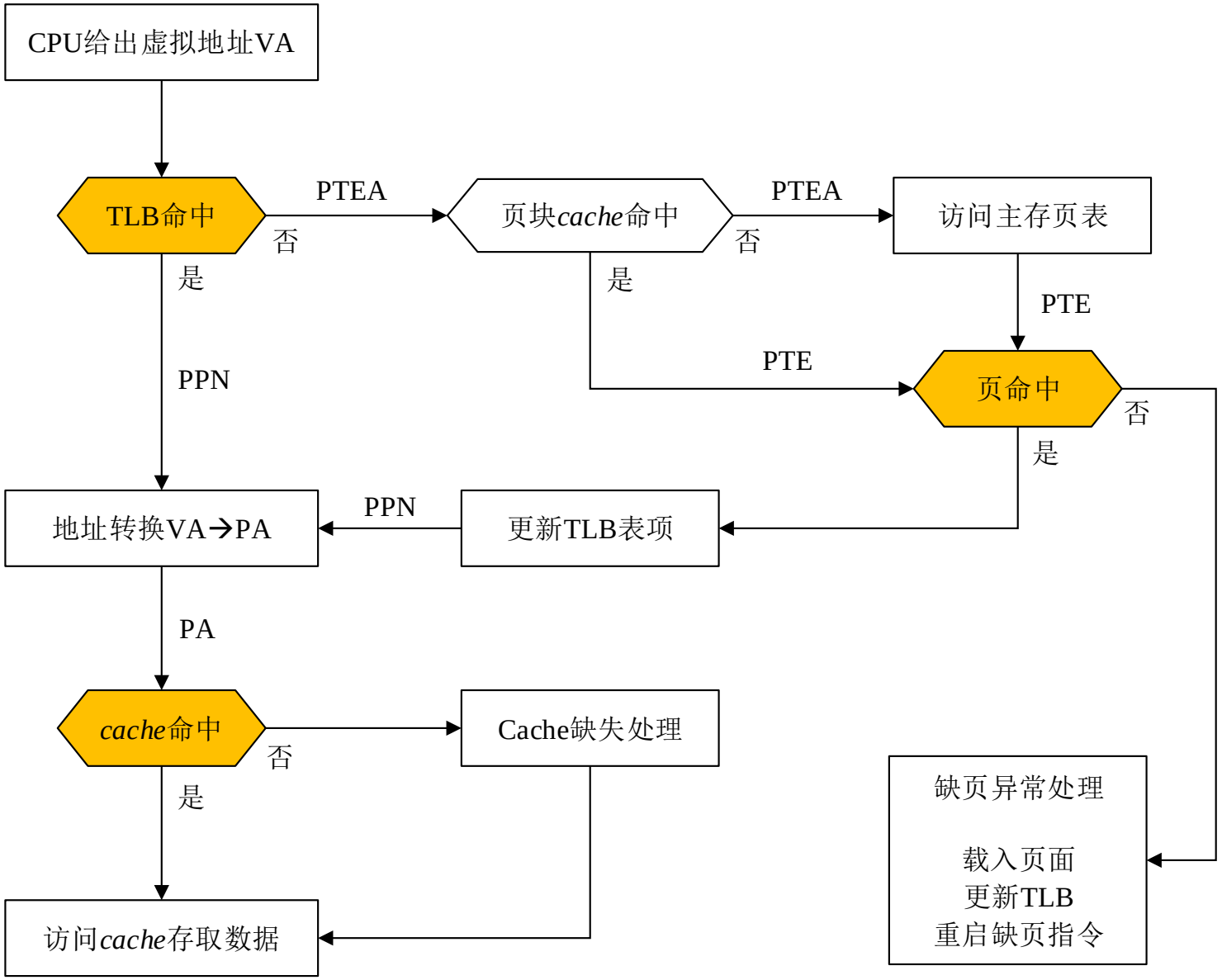

CPU
TLB
OS维护TLB
2) VPN
3)TLB
访问缺失
4) PTEA
Cache
Memory
处理器
1) VA
MMU
5) PTE
6) PA
(页表)
7) 数据

# CPU访存




- 虚拟存储器需要硬件和操作系统协同工作
  - 在操作系统引导完成之前，CPU只能用物理地址访问主存(称为实地址模式或实模式)
  - 引导完成后则进入保护模式(虚地址模式)，此时CPU只能使用虚拟地址访问主存

# 虚拟地址→物理地址流程


CPU给出虚拟地址VA
TLB命中
PTEA
否
页块cache命中
PTEA
否
访问主存页表
是
是
PTE
PPN
PTE
页命中
否
是
地址转换VA→PA
PPN
更新TLB表项
PA
cache命中
否
Cache缺失处理
是
缺页异常处理
载入页面
更新TLB
重启缺页指令
访问cache存取数据

- 例题4.7