



北京航空航天大學
BEIHANG UNIVERSITY

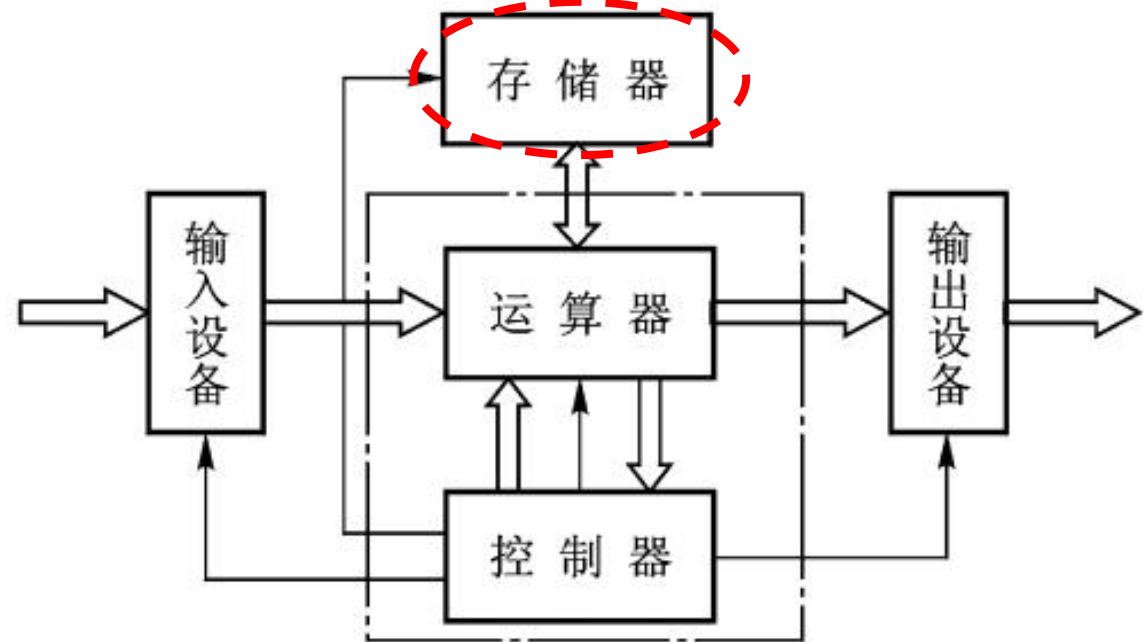
存储器

2020年11月21日星期六



概述：计算机组成

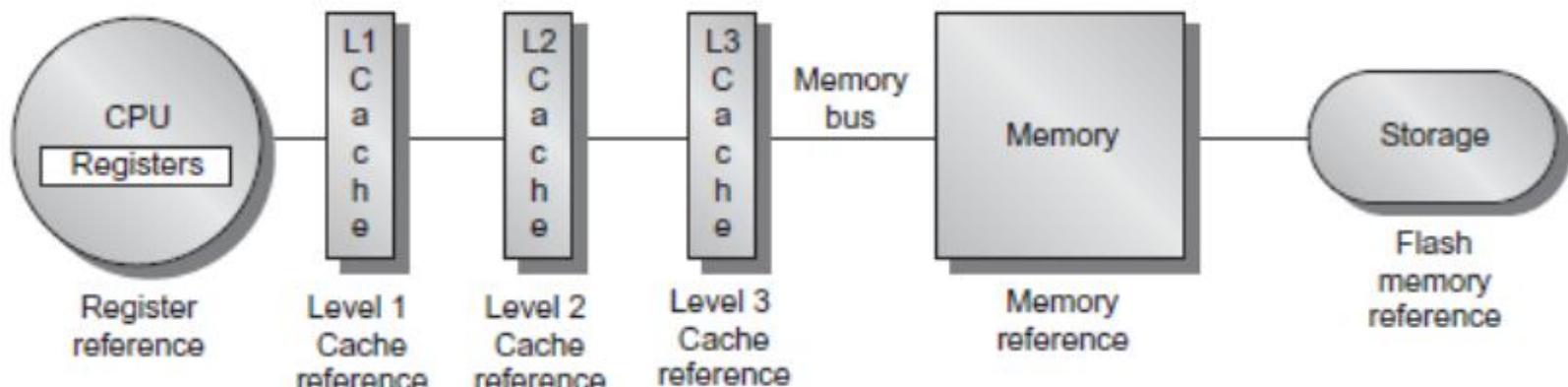
- (1) 运算器ALU（如加法器、布尔逻辑、移位器等）
- (2) 存储器
- (3) 控制器（有限状态机，时序电路）
- (4) 输入输出I/O接口
- (5) 互连





概述

一台计算机里有哪些存储器？



| | Size: | 1000 bytes | 64 KB | 256 KB | 4-8 MB | 4-16 GB | 256 GB-1 TB |
|---------|--------|------------|-------|---------|----------|-----------|-------------|
| Laptop | Speed: | 300 ps | 1 ns | 3-10 ns | 10-20 ns | 50-100 ns | 50-100 uS |
| Desktop | Size: | 2000 bytes | 64 KB | 256 KB | 8-32 MB | 8-64 GB | 256 GB-2 TB |
| | Speed: | 300 ps | 1 ns | 3-10 ns | 10-20 ns | 50-100 ns | 50-100 uS |

(B)

Memory hierarchy for a laptop or a desktop





概述

■ 存储器与寄存器（触发器）的不同之处

- 用途（相同）——存储电路的历史状态
- 电路功能不同（寄存器直接跟CPU交互，存储器需要通过寄存器才能与CPU交互）
- 器件结构不同



概述

➤ 器件结构

| | 寄存器 | 存储器 |
|------|-----|-------------------|
| 存储单元 | 触发器 | 晶体管存储单元 |
| 存储密度 | 低 | 高 |
| 访问方式 | 连线 | 地址译码结合读/ 写脉冲 |
| 访问操作 | 并发 | 每次只能读/写部 分存储数据 |



概述

➤ 存储器的地址

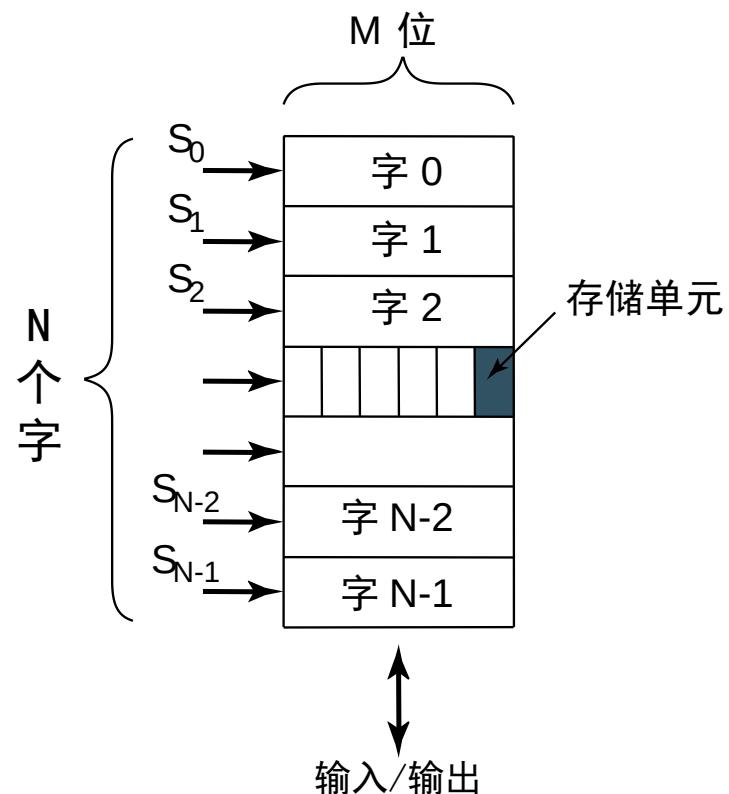
- “因为半导体存储器的存储单元数目极其庞大而器件的引脚数目有限，所以在电路结构上就不能像寄存器那样把每个存储单元的输入和输出直接引出。
- 为了解决这个矛盾，在存储器中的每个存储单元编了一个地址，只有被输入地址代码指定的那些存储单元才能与公共的输入/输出引脚接通，进行数据的读出或写入。 ”
- 地址、数据、读写控制 三总线结构



存储容量与数据结构

存储容量：位(bit)，字节(byte)，字(word)，千字节(Kbyte)，兆字节 (Mbyte)，吉字节(Gbyte)，太字节(Tbyte)；
存储器容量 = 字数×位数 (字长)

N × M 线性结构



字数目 == N 选择信号数目



存储器基本结构

阵列结构

- 采用行译码器和列译码器的二维译码结构可减小译码器的规模，两个译码器交叉译码选中某存储单元

行地址选择信号

- 行地址选择某一行进行读/写操作，列地址选取进行读/写的字单元。

列地址选择信号



存储器基本结构

- 使各存储块内部的字线和位线保持在一定长度范围内，提高存取速度；
- 块地址用于选中唯一存储块进行读写操作，未被选中的块将被置于省电模式，大大节省了整个存储器的功耗。



半导体存储器

■ 半导体存储器的分类

只读 ROM：Read-Only Memory

掩模ROM、PROM

EPROM、E²PROM

随机存取 RAM：Random-Access Memory

SRAM

DRAM

外部存储器：External Memory

机械硬盘，固态硬盘/U盘(Flash)等



只读存储器

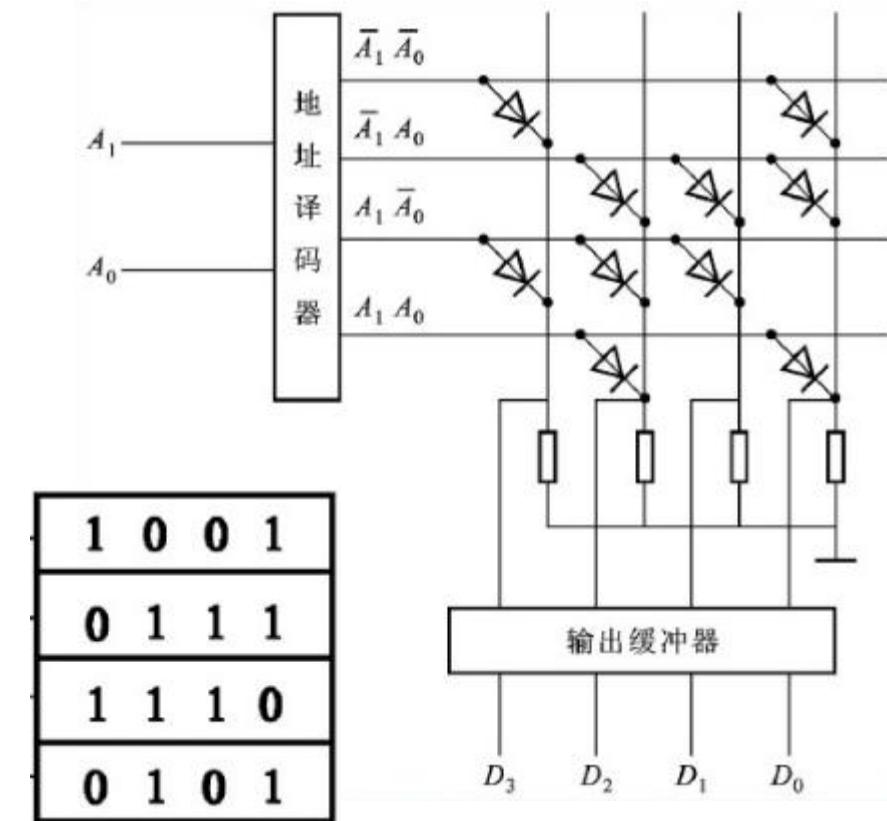
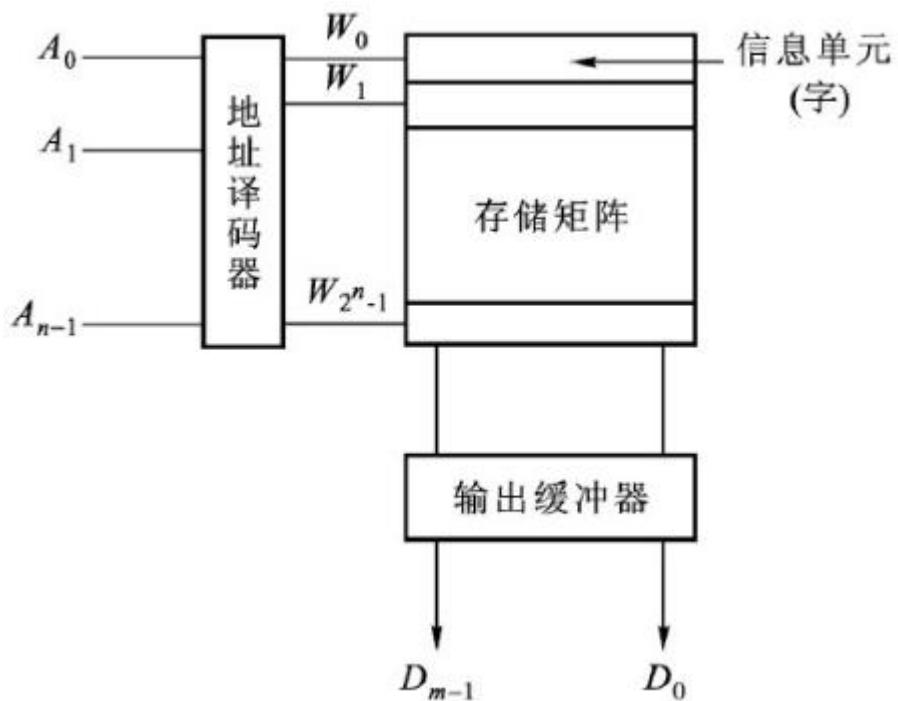
只读存储器 (Read-Only Memory, 简称: ROM)。ROM所存数据，一般是装入整机前事先写好的，整机工作过程中只能读出，而不像随机存储器那样能快速地、方便地加以改写。ROM所存数据稳定，断电后数据也不会改变；常用于存储各种固定程序和系统数据。

- 不可编程只读存储器 (ROM) ； 生产后数据就固定，不可更改
- 可编程只读存储器 (Programmable ROM, 简称: PROM) ；
一般可编程一次，不可逆
- 可擦可编程只读存储器 (Erasable Programmable Read Only Memory, 简称: EEPROM) ； 通常通过紫外线可多次编程
- 电可擦可编程只读存储器 (Electrically Erasable Programmable Read-Only Memory, 简称: EEPROM) ； 电可多次编程



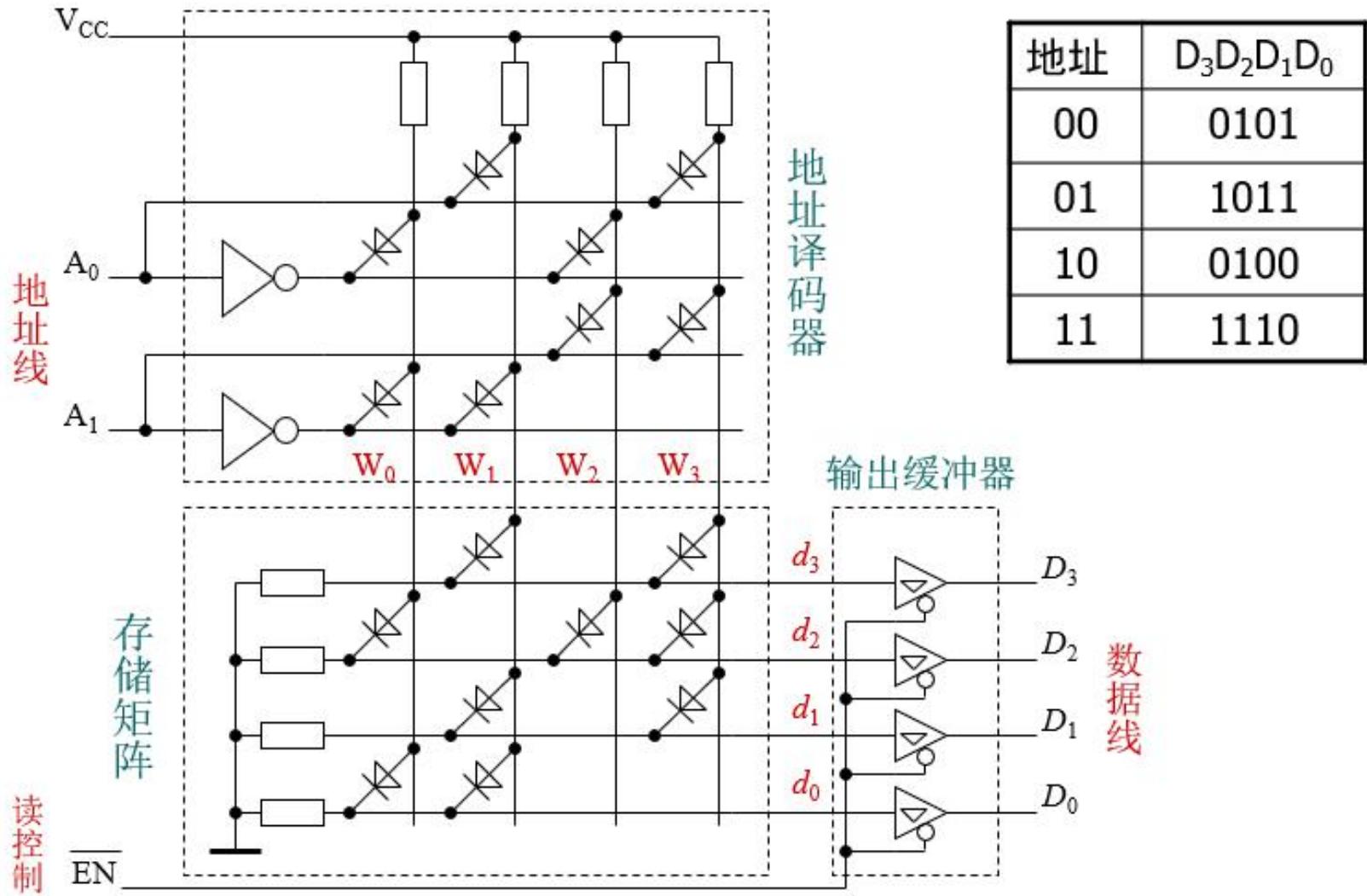
不可编程只读存储器 (ROM)

在制造过程中，将数据以一特制光罩 (mask) 烧录于线路中，其资料内容在写入后就不能更改。典型产品是掩膜ROM。目前较少使用。





不可编程只读存储器 (ROM)





不可编程只读存储器 (ROM)

电脑主板上固化的一个基本输入/输出系统，称为**BIOS**（基本输入输出系统），早期电脑的BIOS就是存储在ROM里。其主要作用是完成对系统的加电自检、系统中各功能模块的初始化、系统的基本输入/输出的驱动程序及引导操作系统。

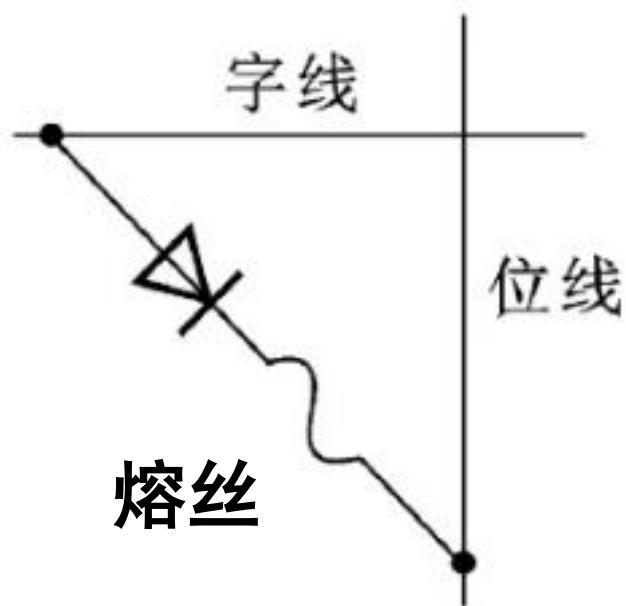
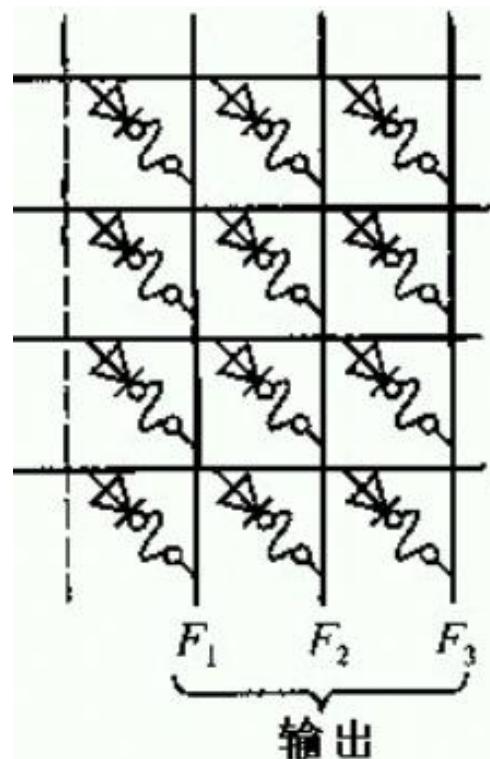


**CD-ROM
DVD-ROM**



一次可编程只读存储器 (PROM)

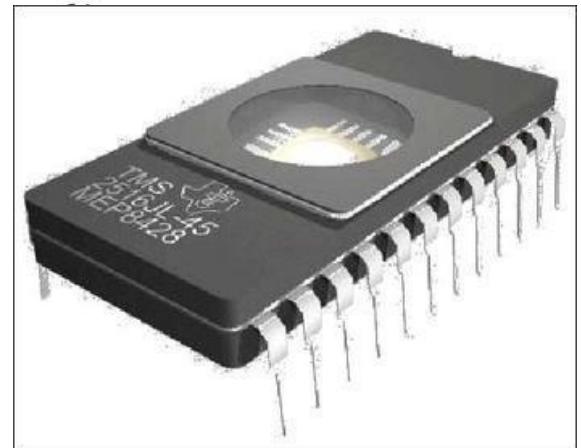
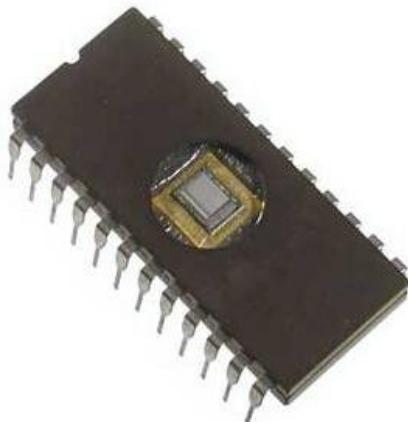
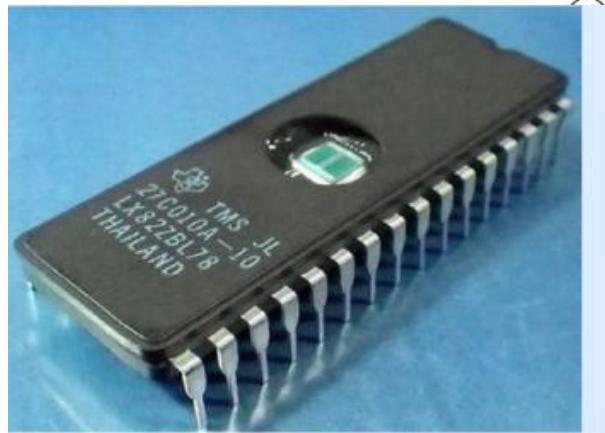
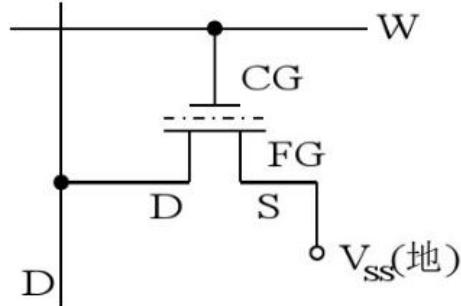
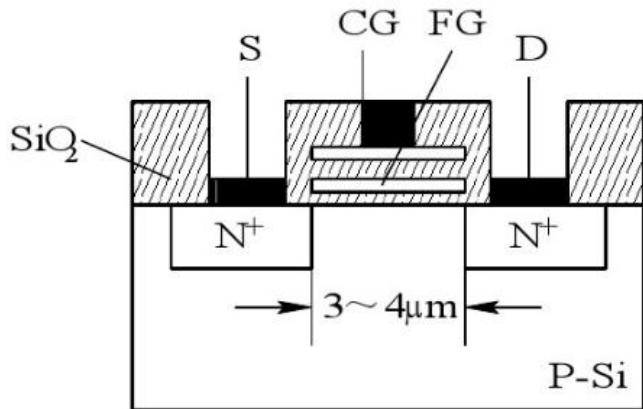
典型产品是“熔丝结构”与“PN结型结构”，可一次编程，通常芯片生产后，所有单元都是数据“1”，然后在使用过程中通过熔断熔丝来得到数据“0”；可通过紫外线、或者大电流熔断，不可恢复；目前较少使用。





可擦可编程只读存储器 (EPROM)

可用户根据需要来写入，并通常能通过紫外线等把已写入的内容擦去后再改写，即是一种多次改写的ROM；目前也较少使用。



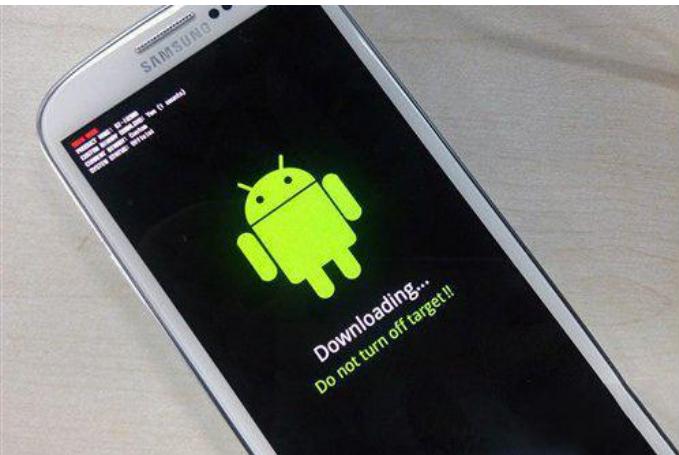
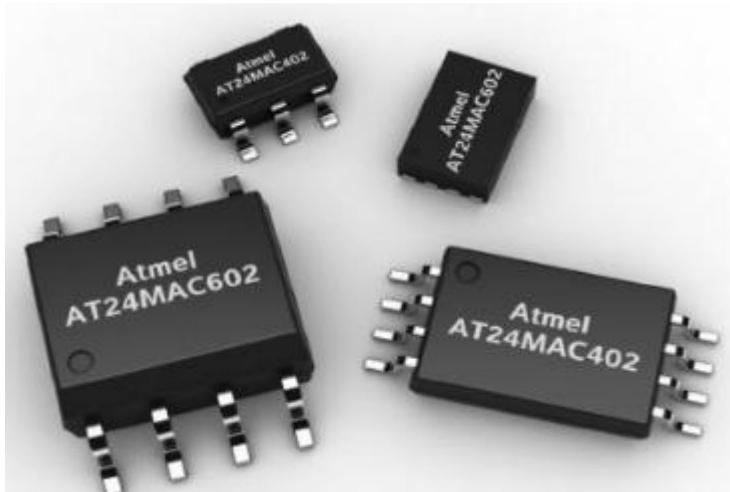
EPROM通常用紫外线擦除，它的信息擦除是整个芯片的擦除



电可擦可编程只读存储器 (EEPROM)

可用户根据需要来写入，并能通过电流电压把已写入的内容擦去后再改写，即是一种多次改写的ROM；目前使用比较多；EEPROM与EPROM结构差不多，只是访存晶体管不太一样，可以支持电流来进行编程与擦除

目前电脑主板上BIOS ROM芯片以及手机内置ROM大部分都采用EEPROM



问题1：既然EEPROM也可以编程，那跟RAM有什么区别？

问题2：手机刷机时为什么通常提示要接电源，且不能断电？

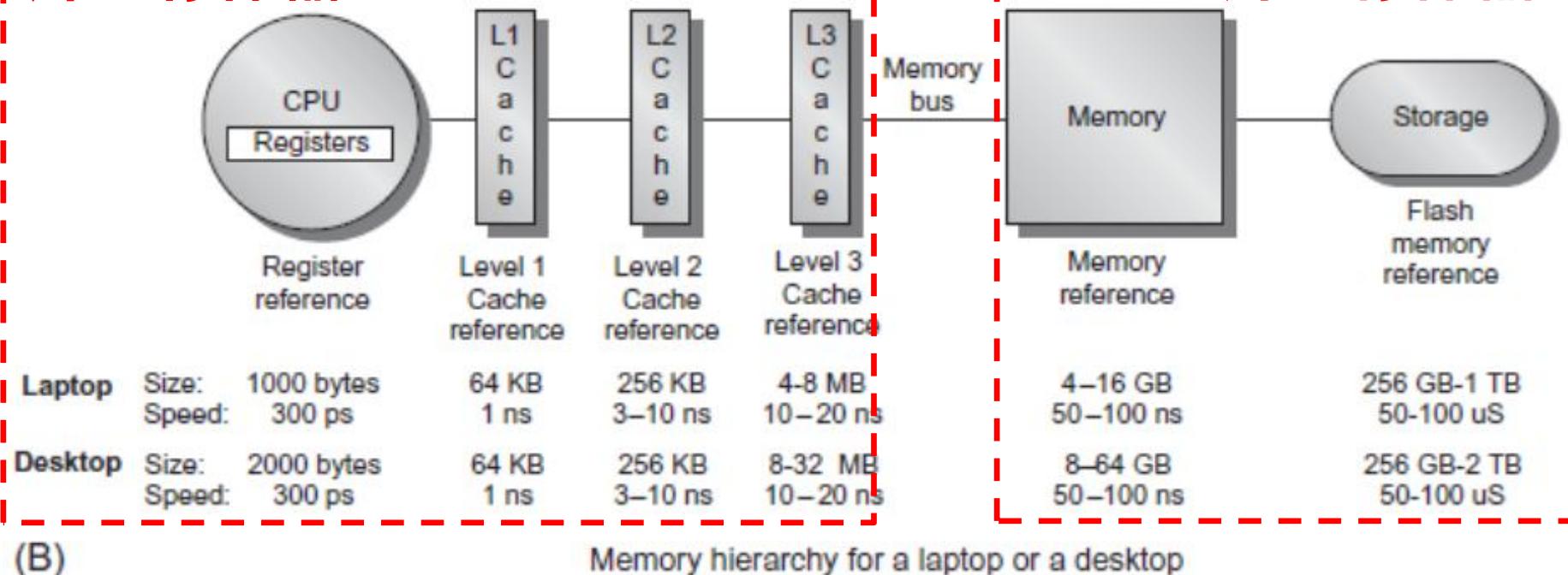


背景：回顾计算机的存储器层次结构

片上存储器

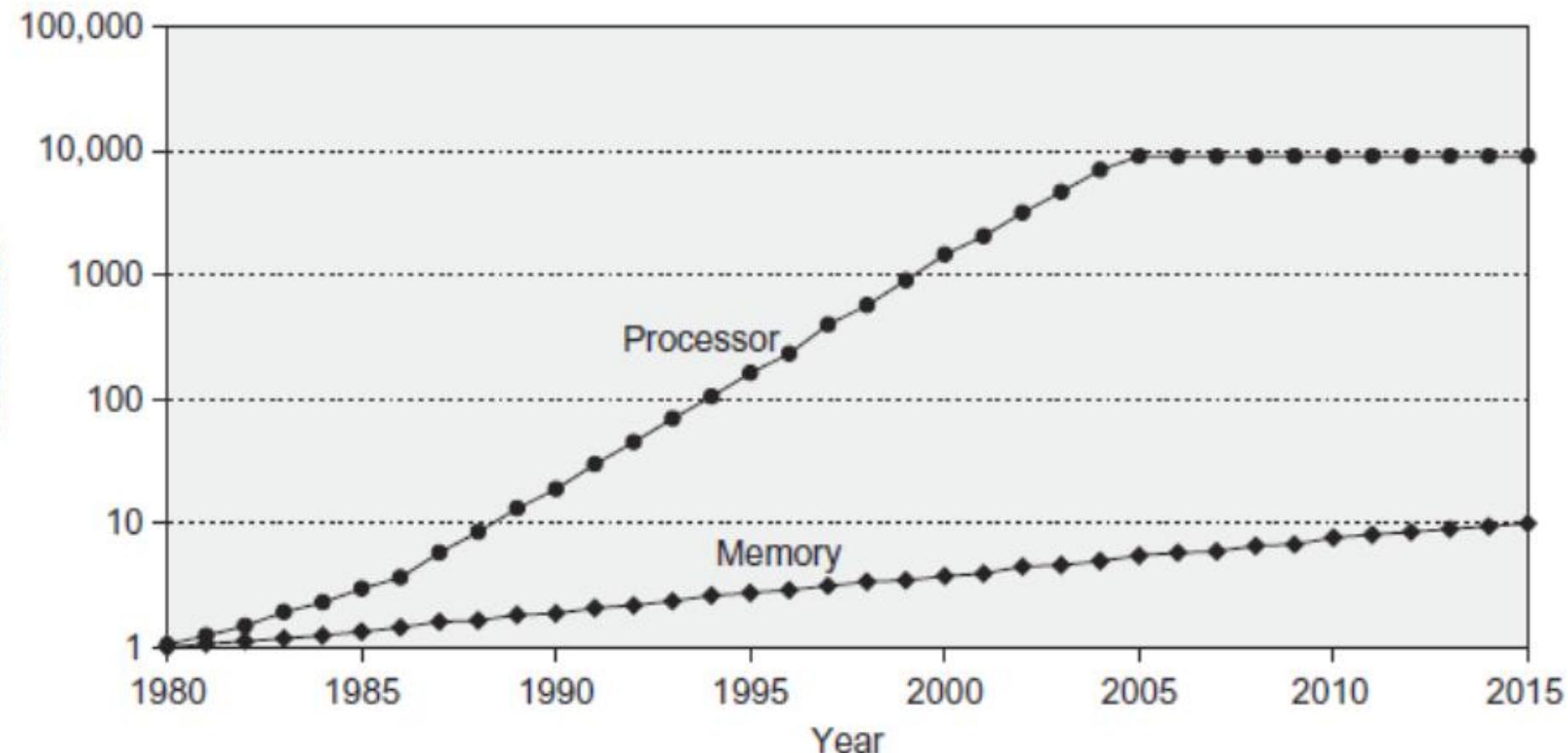
数据
总线

片外存储器





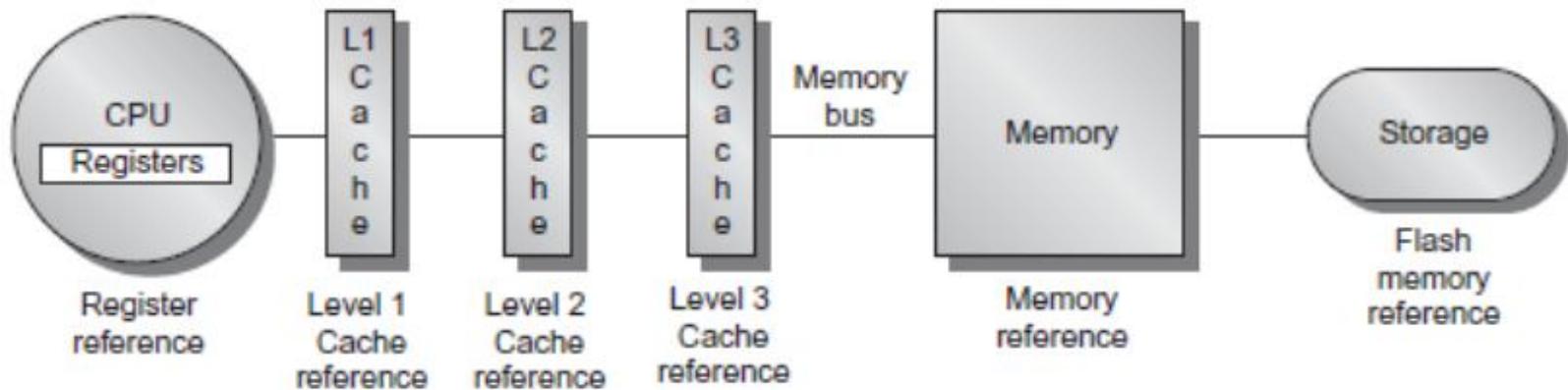
背景：处理器与存储器性能鸿沟



怎么解决这个问题？



存储器层次结构



| | Size: | 1000 bytes | 64 KB | 256 KB | 4-8 MB | 4-16 GB | 256 GB-1 TB |
|---------|--------|------------|-------|---------|----------|-----------|-------------|
| Laptop | Speed: | 300 ps | 1 ns | 3-10 ns | 10-20 ns | 50-100 ns | 50-100 uS |
| Desktop | Size: | 2000 bytes | 64 KB | 256 KB | 8-32 MB | 8-64 GB | 256 GB-2 TB |
| Desktop | Speed: | 300 ps | 1 ns | 3-10 ns | 10-20 ns | 50-100 ns | 50-100 uS |

Memory hierarchy for a laptop or a desktop

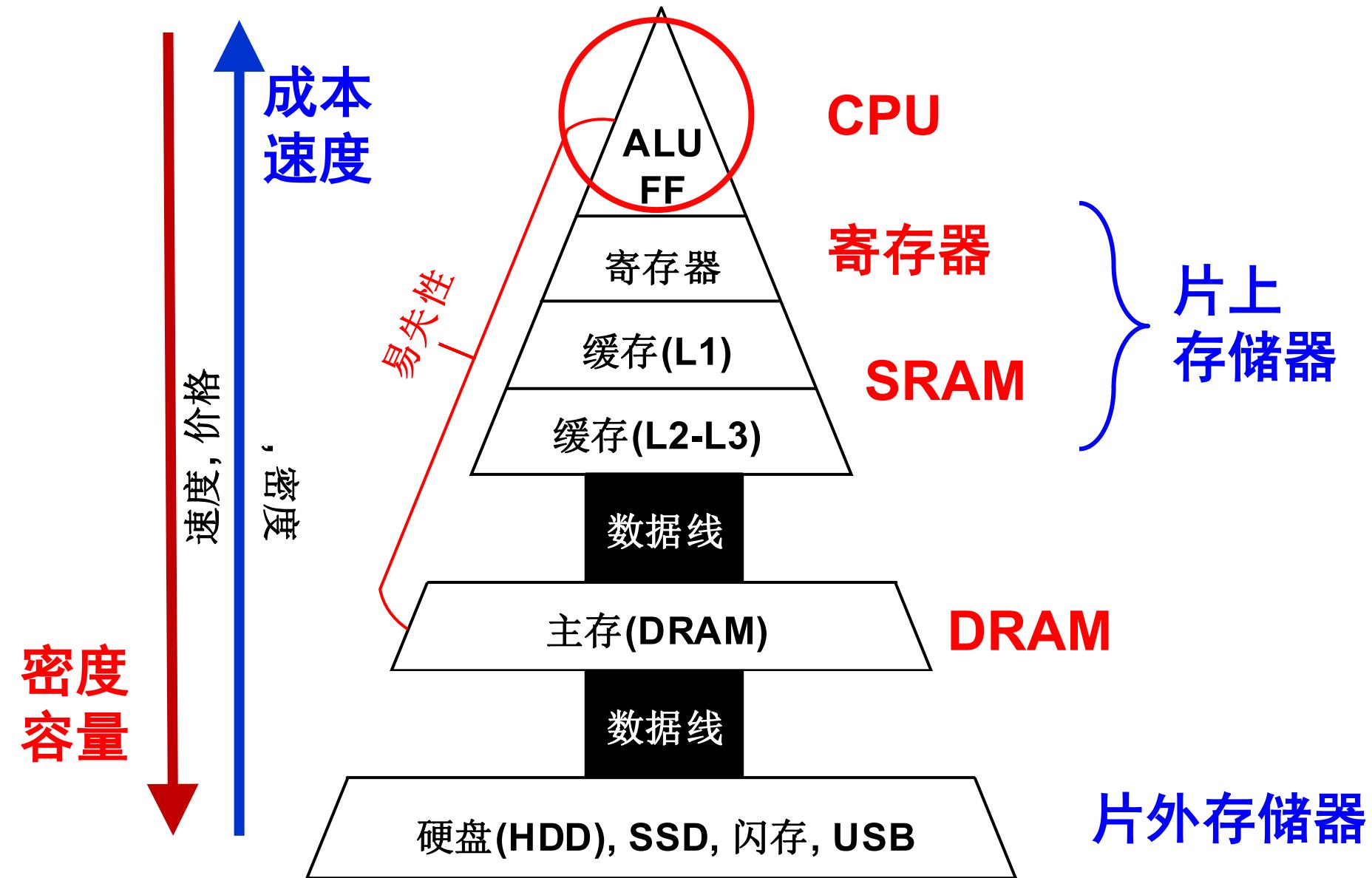


| Operation | Energy [pJ] | Relative Cost |
|--------------------|-------------|---------------|
| 32 bit int ADD | 0.1 | 1 |
| 32 bit float ADD | 0.9 | 9 |
| 32 bit int MULT | 3.1 | 31 |
| 32 bit float MULT | 3.7 | 37 |
| 32 bit 32KB SRAM | 5 | 50 |
| 32 bit DRAM | 640 | 6400 |

数据搬运功耗远
远大于计算功耗

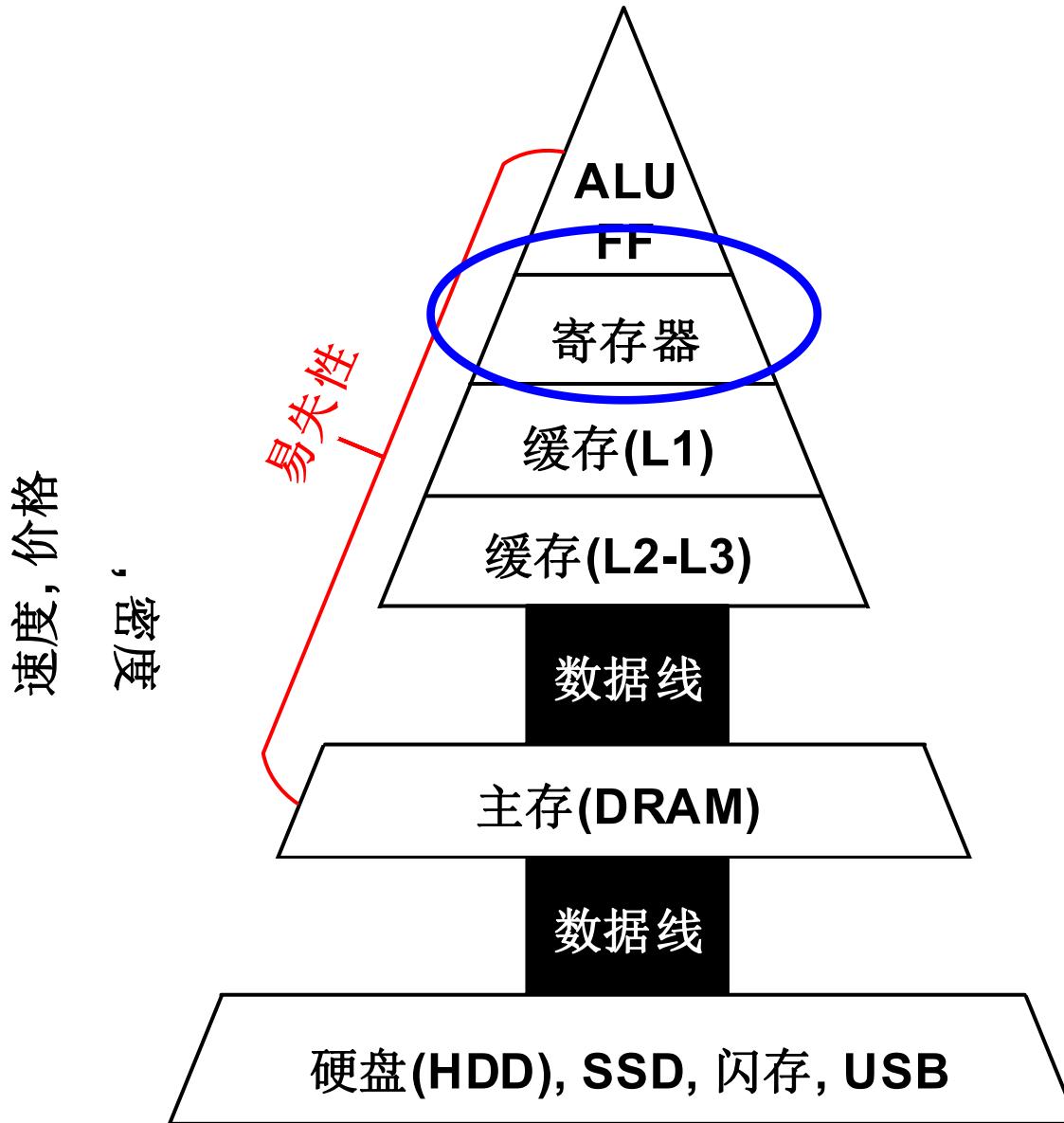


存储器层次结构





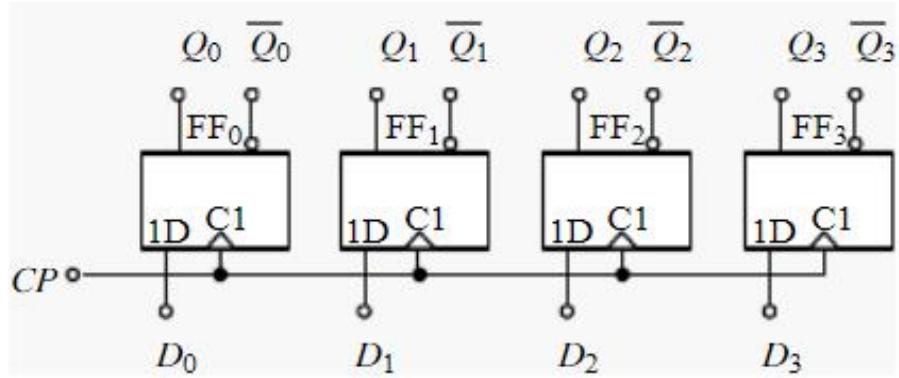
存储器层次结构





寄存器 VS 触发器

寄存器是中央处理器内的组成部分，用来暂存指令、数据和地址，由触发器组成。寄存器数量很少，通常Kbytes
所有数据通常都必须通过寄存器与CPU进行交互。
触发器按电路结构可分为 RS触发器、D触发器、JK触发器、T触发器等。



问题？

查看有关计算机的基本信息

Windows 版本

Windows 10 家庭中文版

© 2019 Microsoft Corporation。保留所有权利。

系统

处理器: Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz

已安装的内存(RAM): 8.00 GB (7.89 GB 可用)

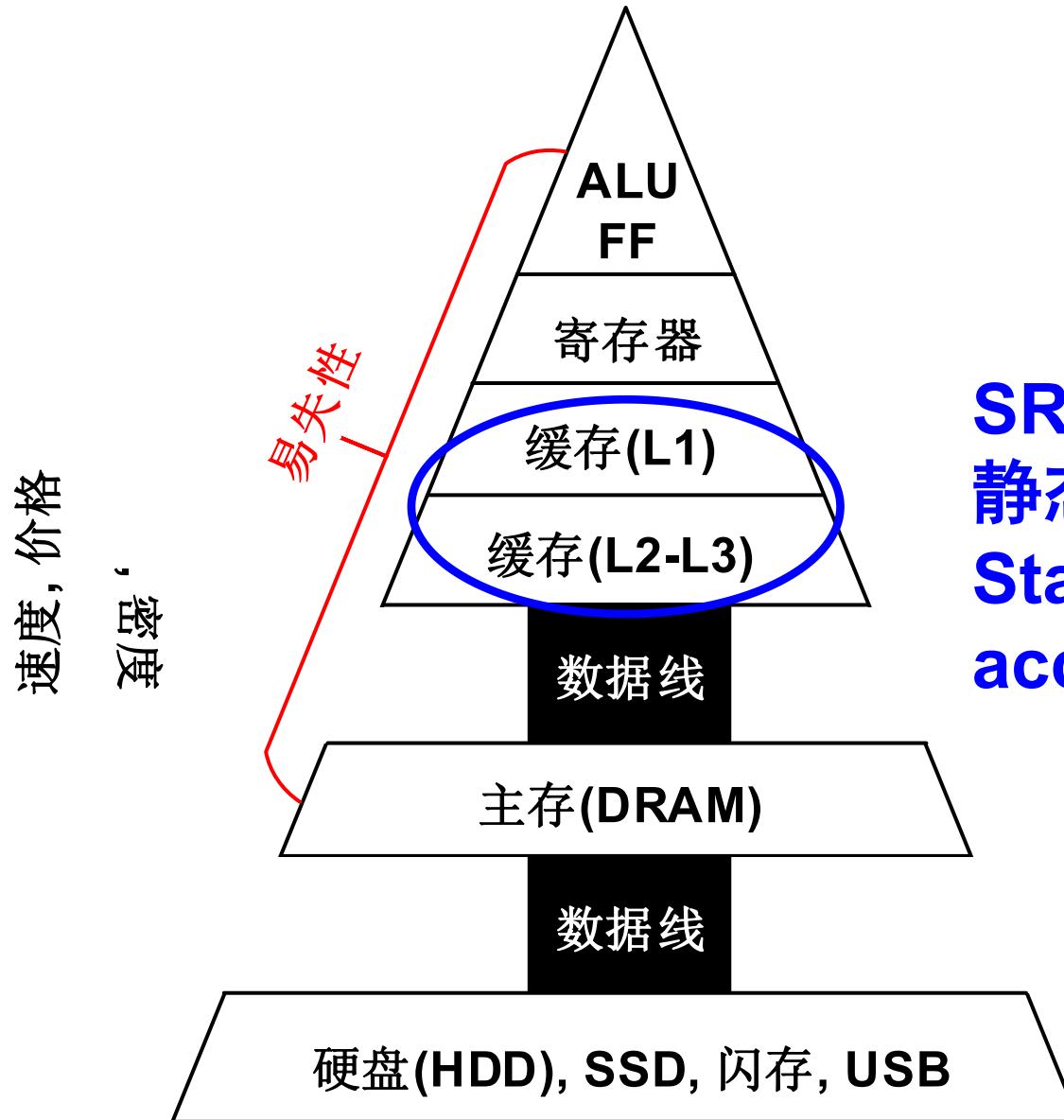
系统类型: 64 位操作系统，基于 x64 的处理器

笔和触控: 没有可用于此显示器的笔或触控输入

32位计算机，64位计算机，什么意思？



存储器层次结构



SRAM
静态随机存取存储器
Static random-access memory



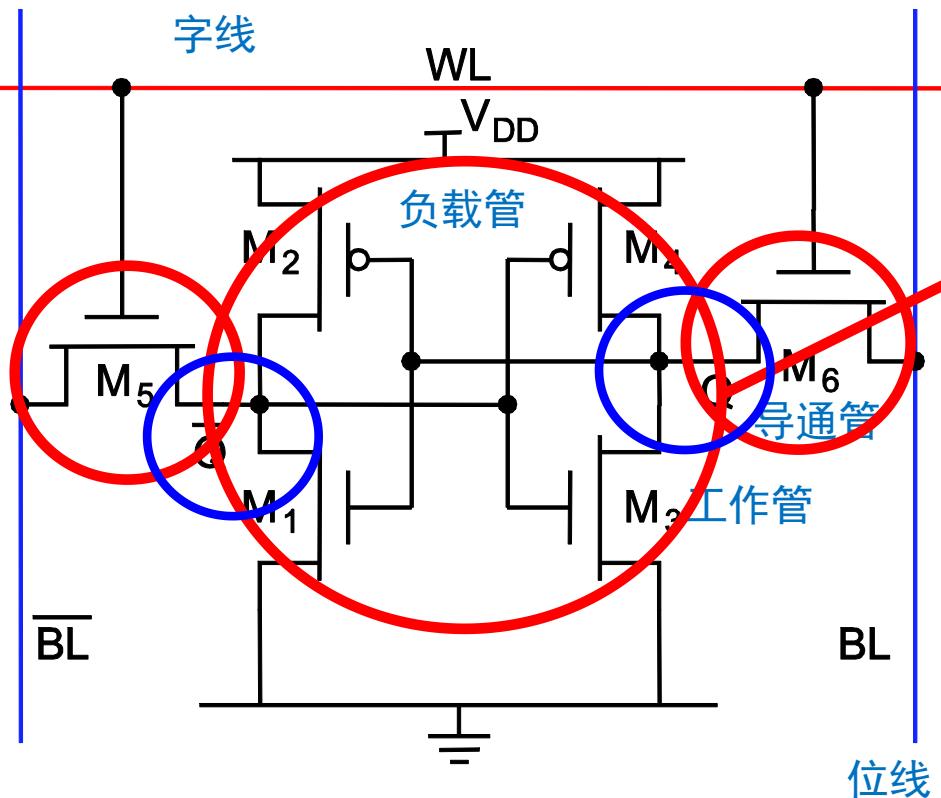
静态随机存取存存储器 (SRAM)

■ 特点

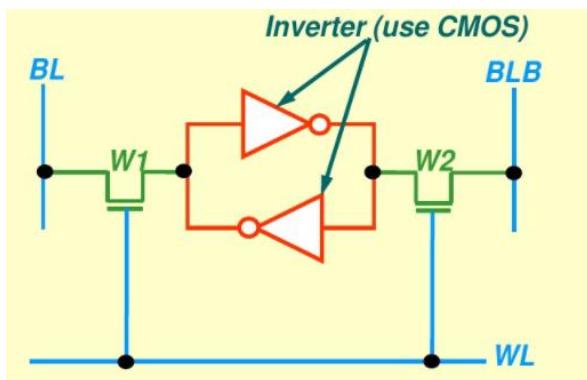
- 可以随时从任何一个指定的地址读出数据，也可以随时将数据写入任何一个指定的存储单元中去
- 掉电后存储的数据丢失



静态随机存取存储器 (SRAM)

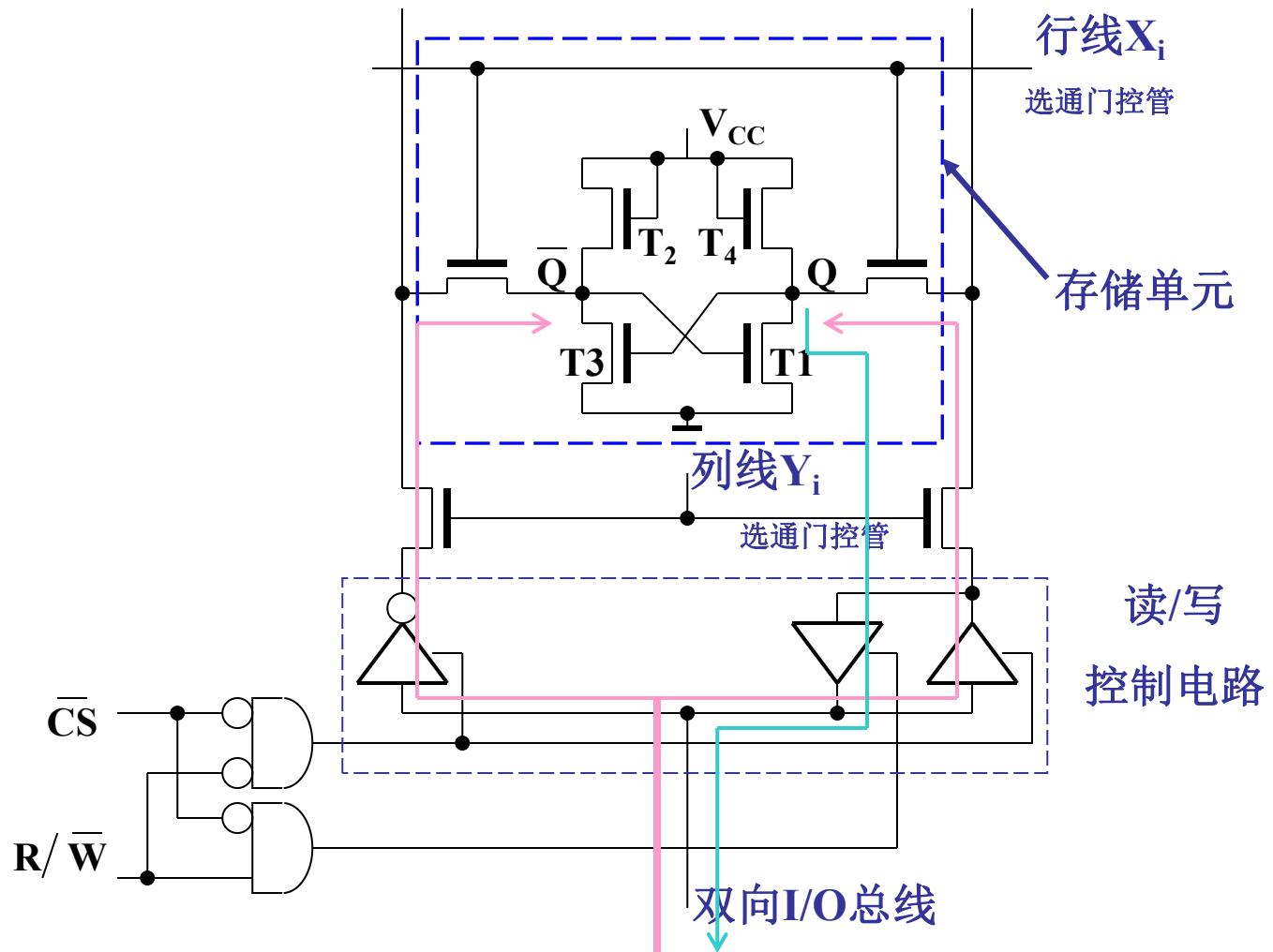


- 两个MOS反相器交叉耦合
- 每个基本的存储单元由六个MOS管构成，所以，静态存储电路又称为六管静态存储电路。
- 字线(WL)控制导通管(M5,M6)
- 极性相反的位线(BL,/BL)传输存储信息
- 信息存储于节点Q与Qbar





静态随机存取存储器 (SRAM)

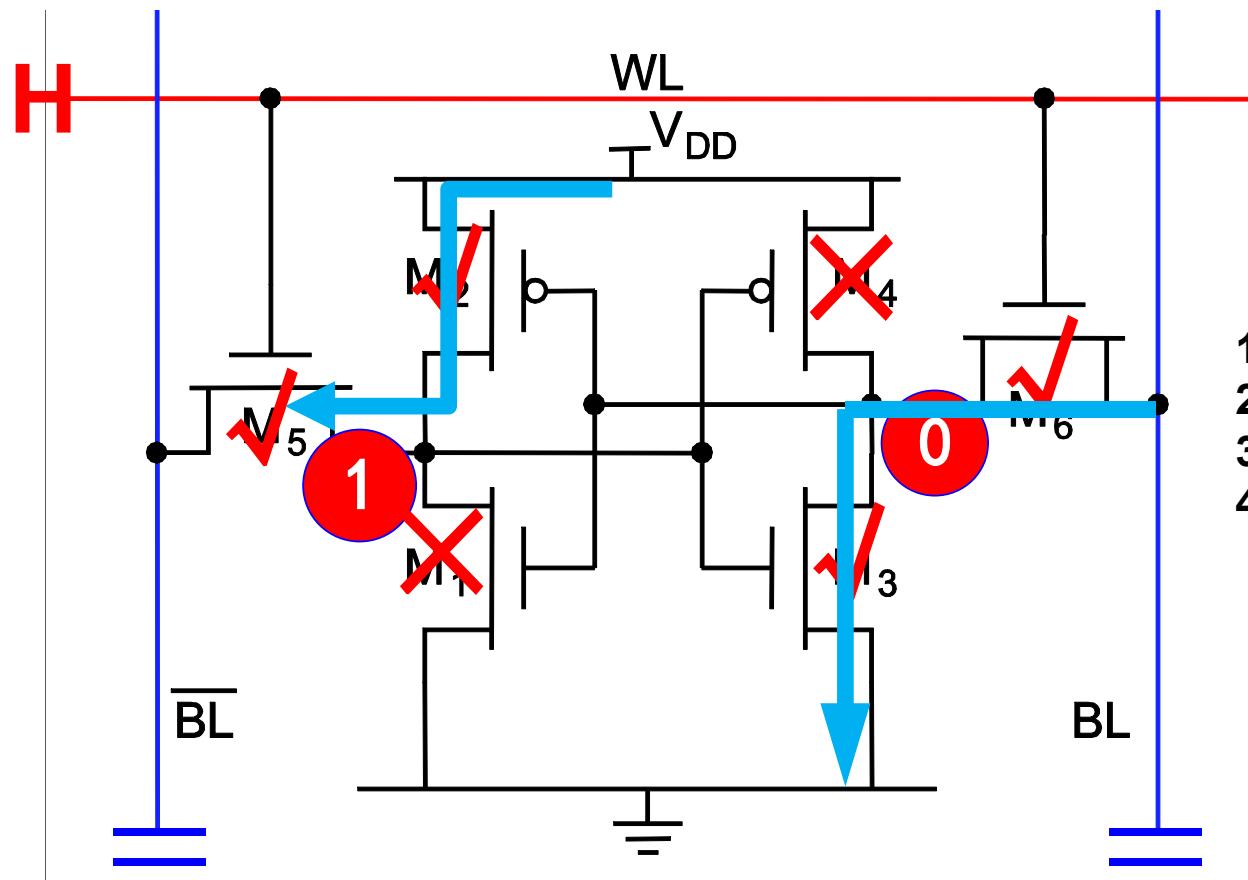




六管SRAM存储单元—读(1,0)操作

充电维持高电平

放电到低电平



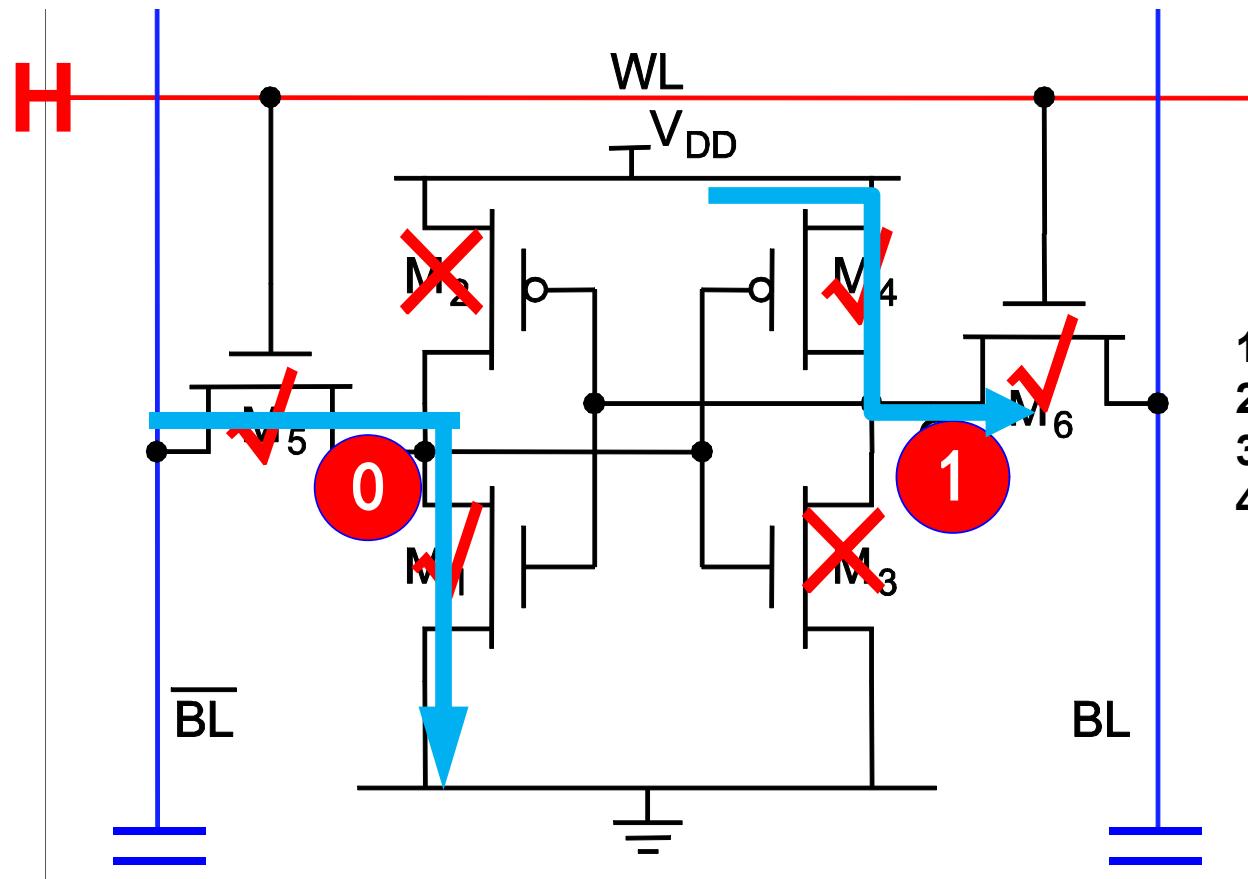
- 1、BL BL_bar为高，电容充电
- 2、WL位高, M5,M6导通
- 3、假设Q=0,
- 4、Q和Q_bar通过BL和BL_bar读出



六管SRAM存储单元—读(0,1)操作

放电到低电平

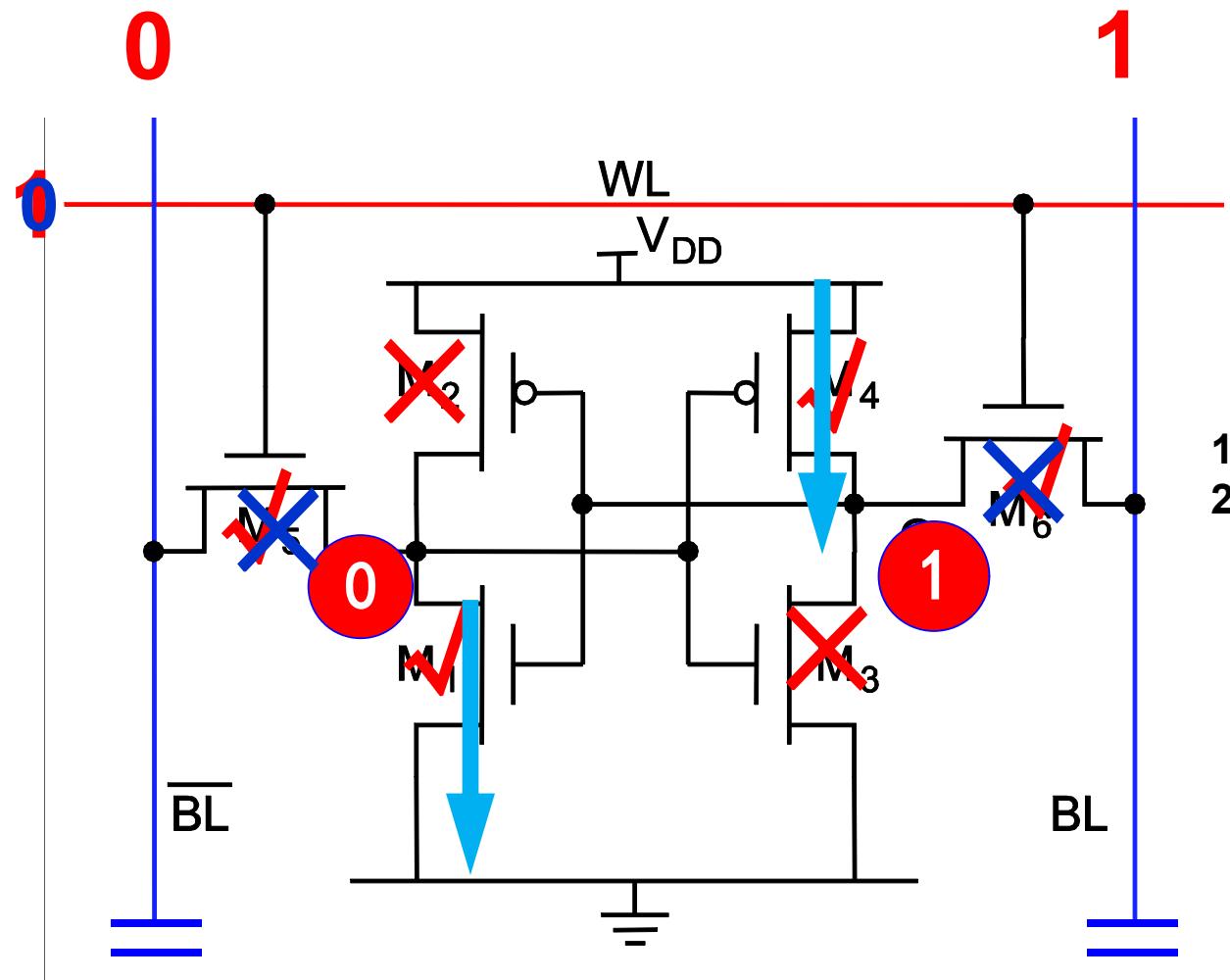
充电到高电平



- 1、BL BL_bar为高，电容充电
- 2、WL位高, M5,M6导通
- 3、假设Q=1,
- 4、Q和Q_bar通过BL和B_bar读出



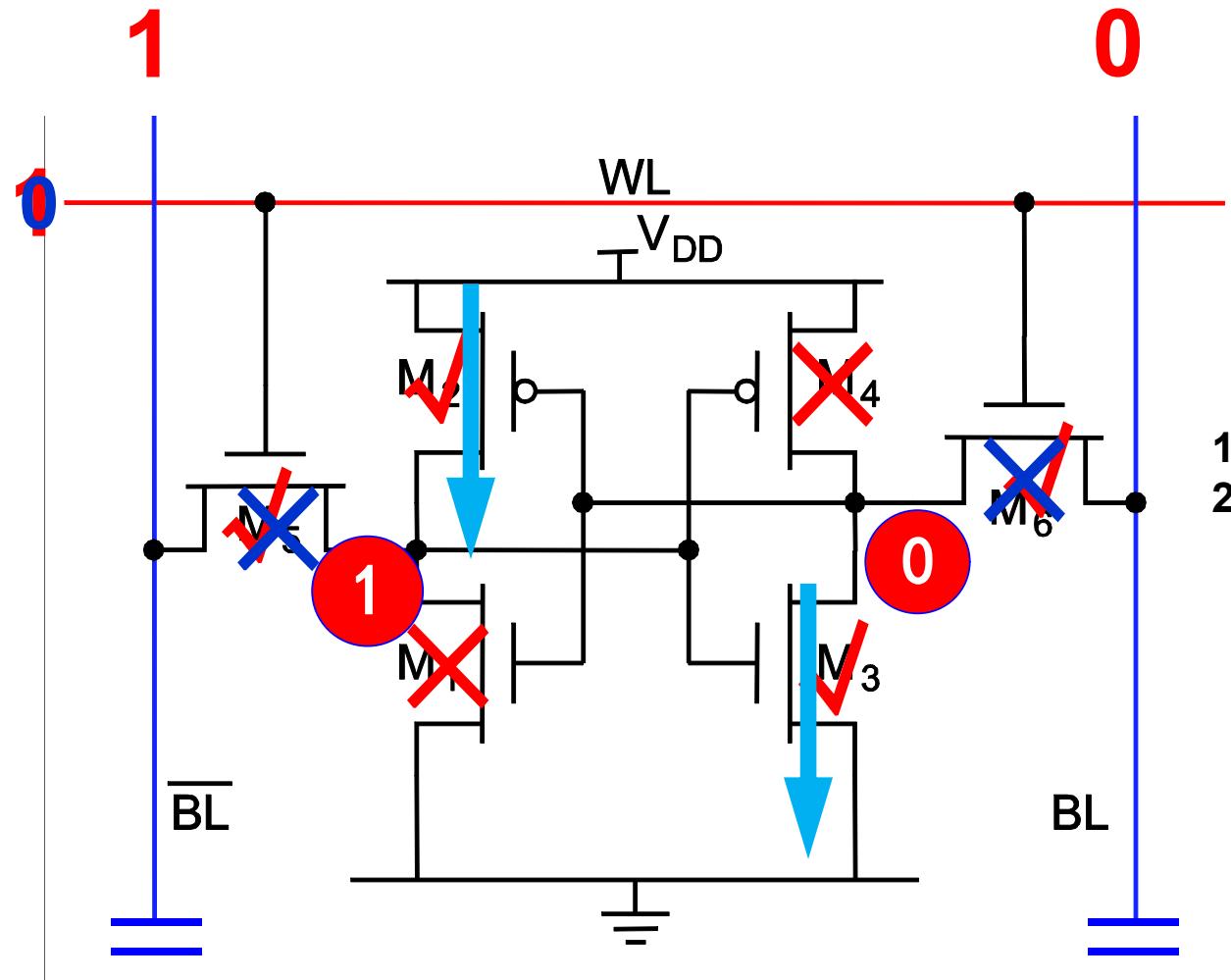
六管SRAM存储单元—写(0,1)操作



- 1、WL位高, M₅,M₆导通
- 2、假设写1, 即BL=1, BL_bar = 0

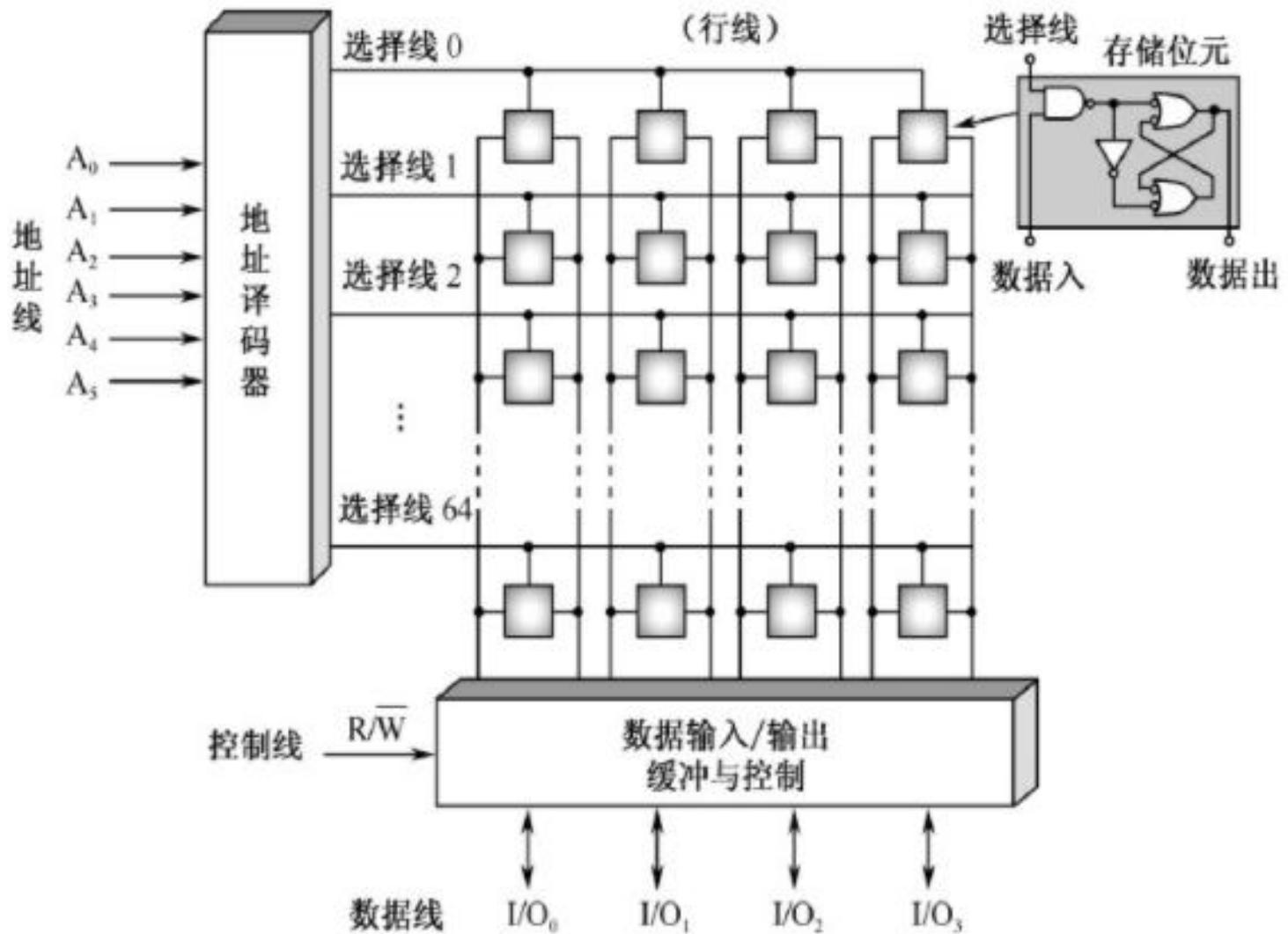


六管SRAM存储单元—写(1,0)操作



- 1、WL位高, M₅,M₆导通
- 2、假设写0, 即BL=0, BL_bar = 1

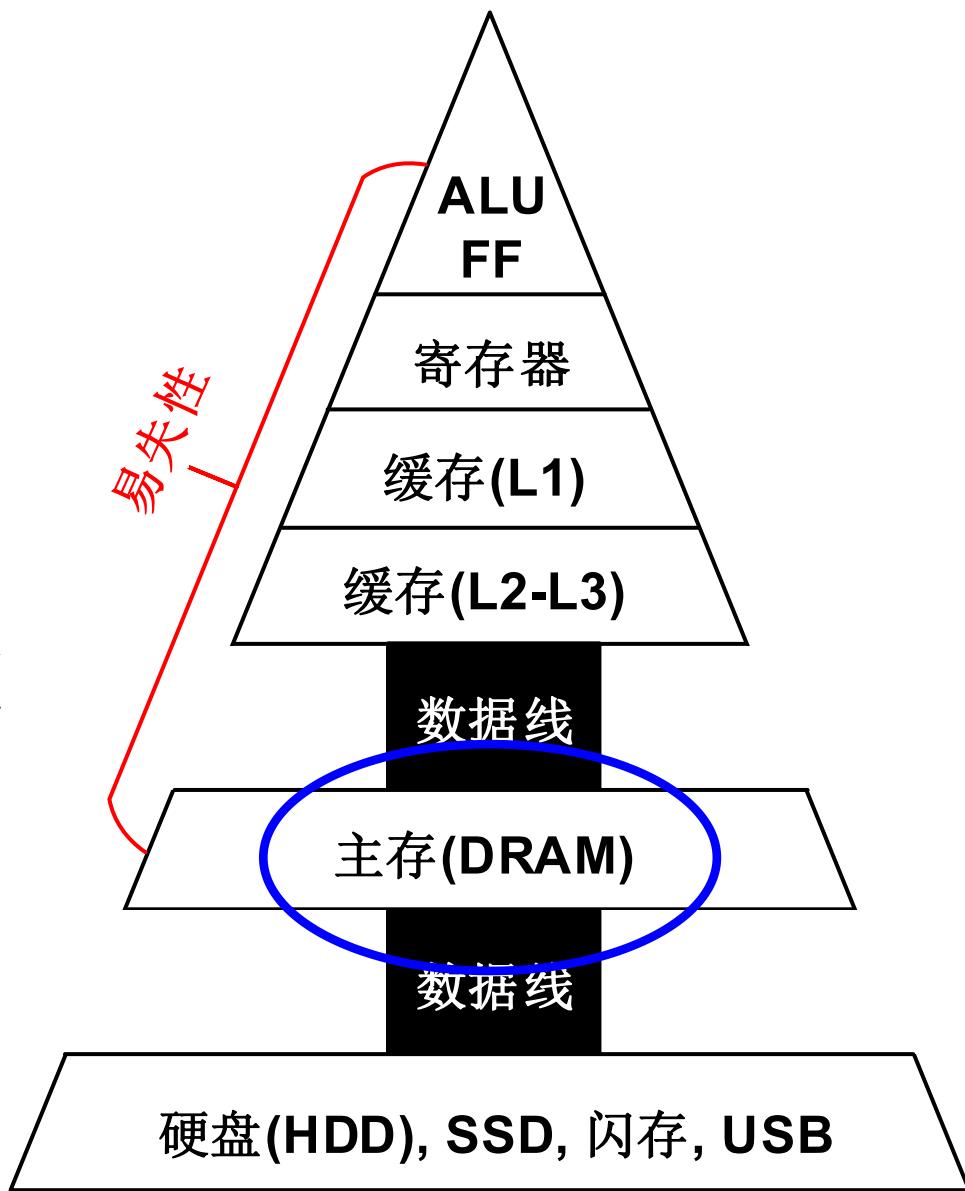
SRAM存储阵列





存储器层次结构

速度，价格
密度

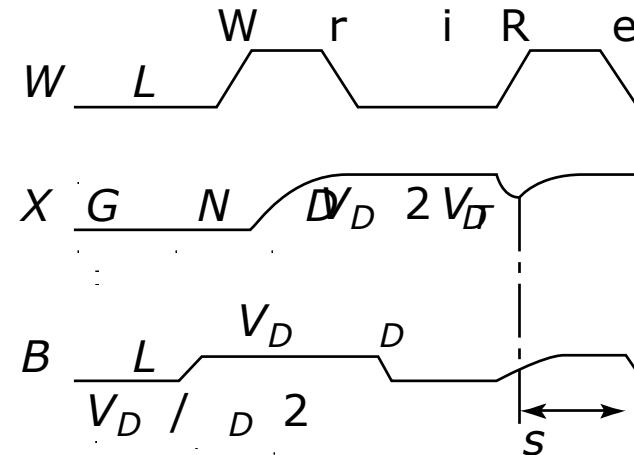
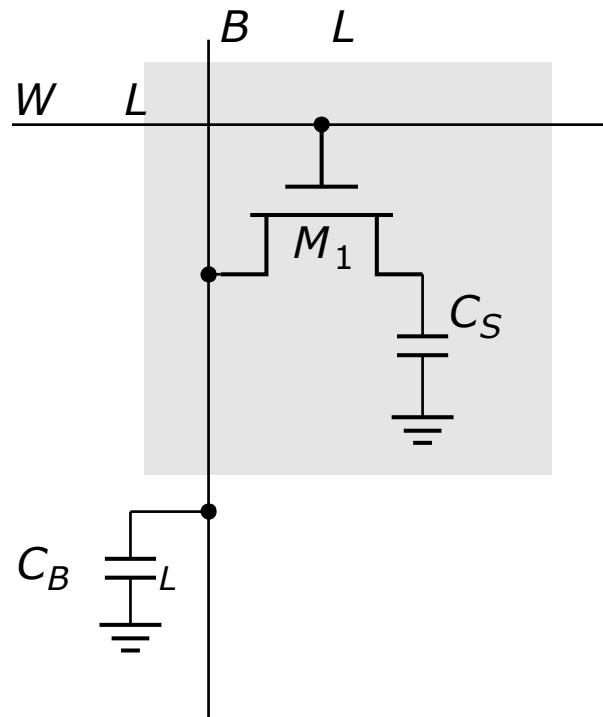


DRAM
动态随机存取存储器
**Dynamic random
access memory**



动态随机存取存储器 (DRAM)

利用电容存储的电荷多少来存储信息



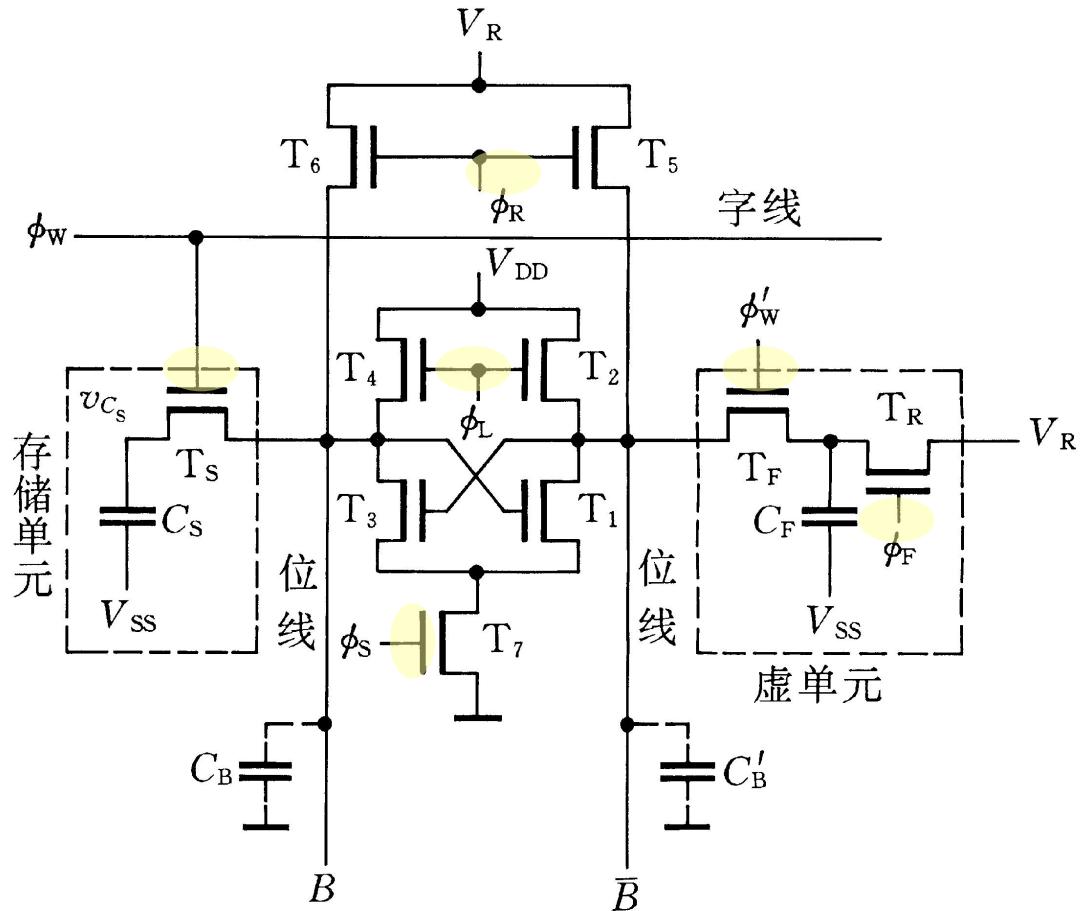
- 写操作：字选择线 (WL) 为高电平， M_1 管导通，写信号存入电容 C_S 中；
- 读操作：字选择线为高电平，存储在电容 C_S 上的电荷，通过 M_1 输出到数据线上，经过读放放大器读出所保存的信息。

注意：破坏性读取，每次读取数据后，需要回写刷新



DRAM灵敏恢复/读出电路

读出操作在顺序产生的时钟信号控制下进行
 $(\varphi_F \varphi_R) \rightarrow (\varphi_W) \rightarrow$ 时钟信号(φ_S) $\rightarrow \varphi_L$

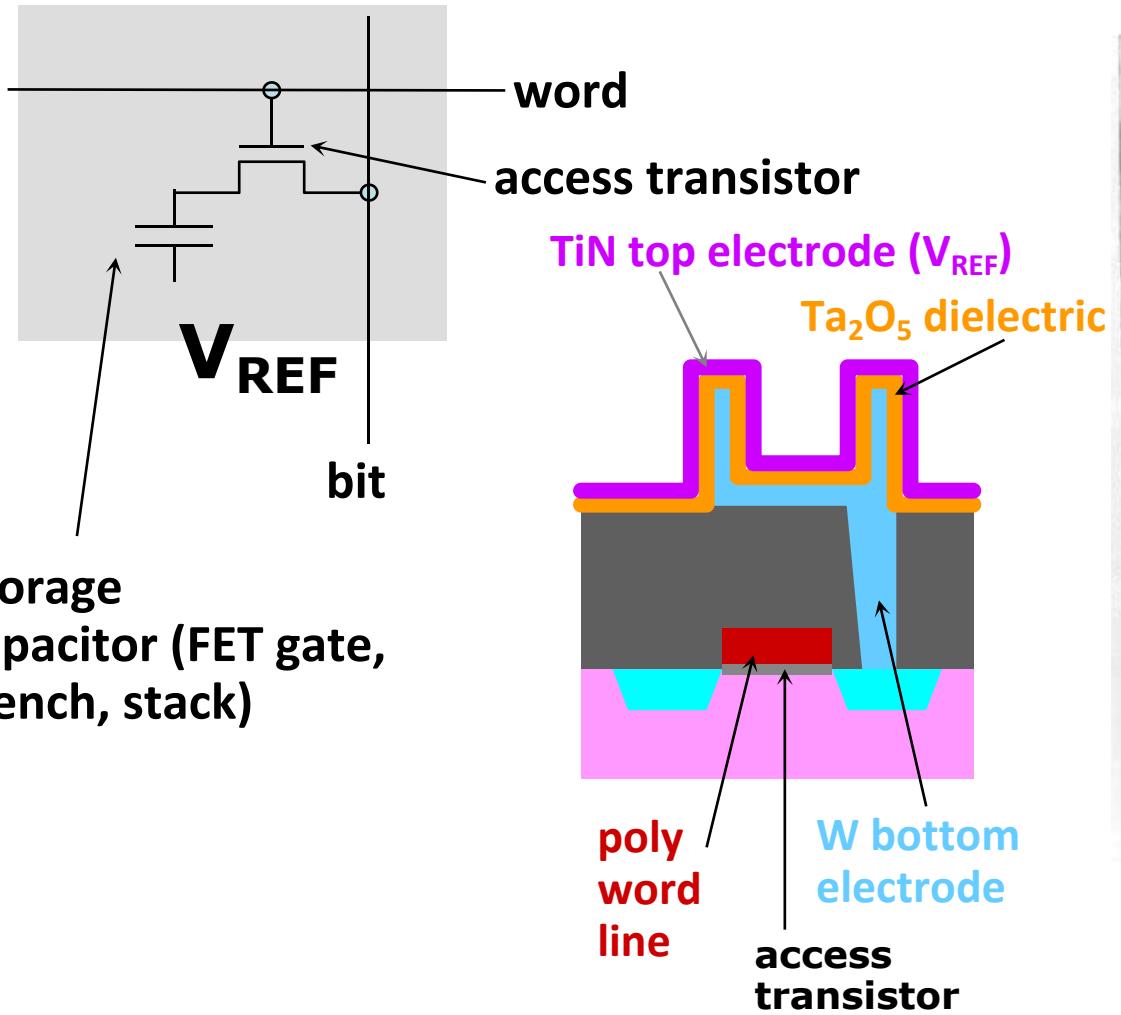


- 正脉冲 $\varphi_F \varphi_R$, B, B' (位线) 和 C_F 被预充电到参考电位 V_R ;
- 正反馈电路, 对位线 B 和 B' 电位的微小差别进行放大, 形成正常的高低电平
- 如果电容 C_S 存储有电荷, 将 B (B') 置为正常的高电平(低电平), 同时为 C_S 恢复到高电平;
- 如果电容 C_S 没有存储电荷, 则...

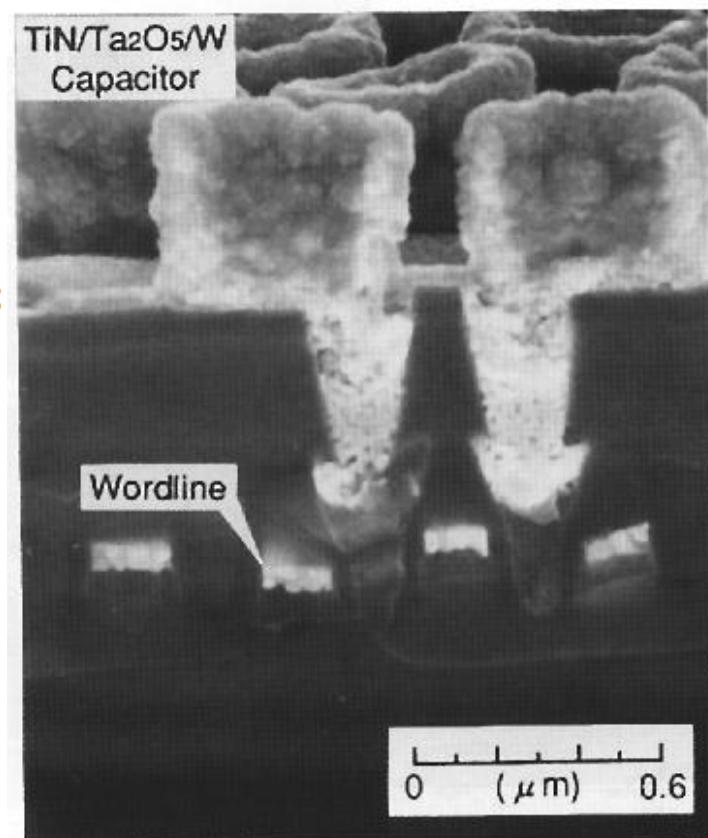
单管DRAM [IBM]



1-T DRAM Cell



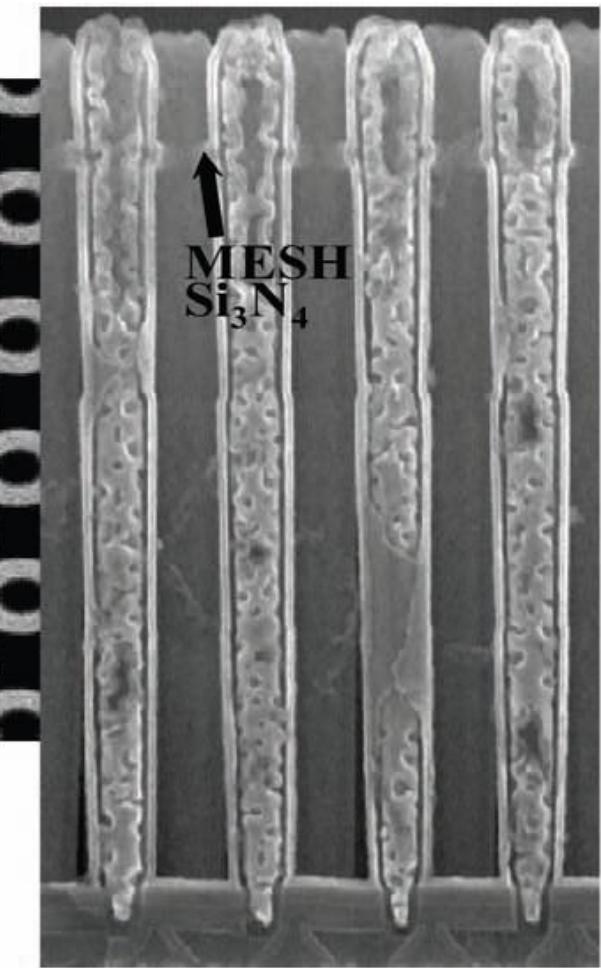
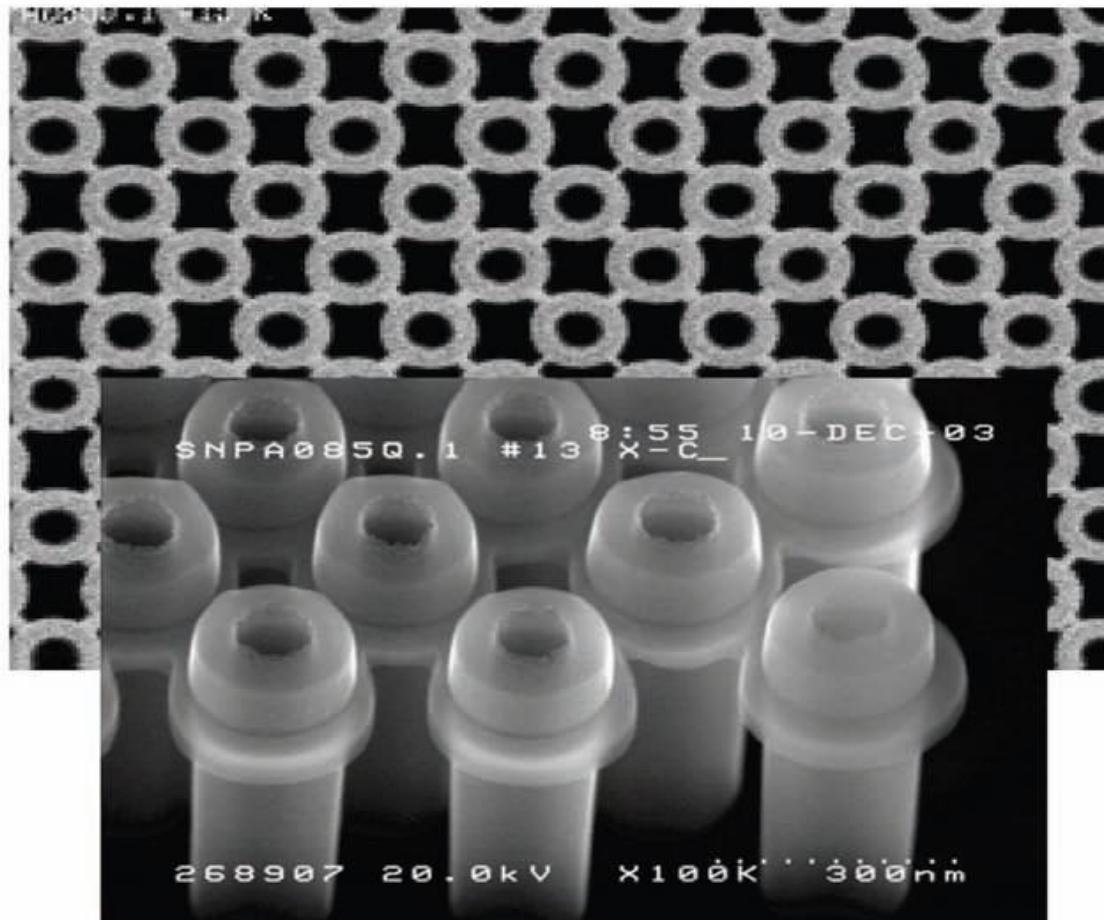
Storage
capacitor (FET gate,
trench, stack)





现代DRAM结构

电容漏电，需刷新操作



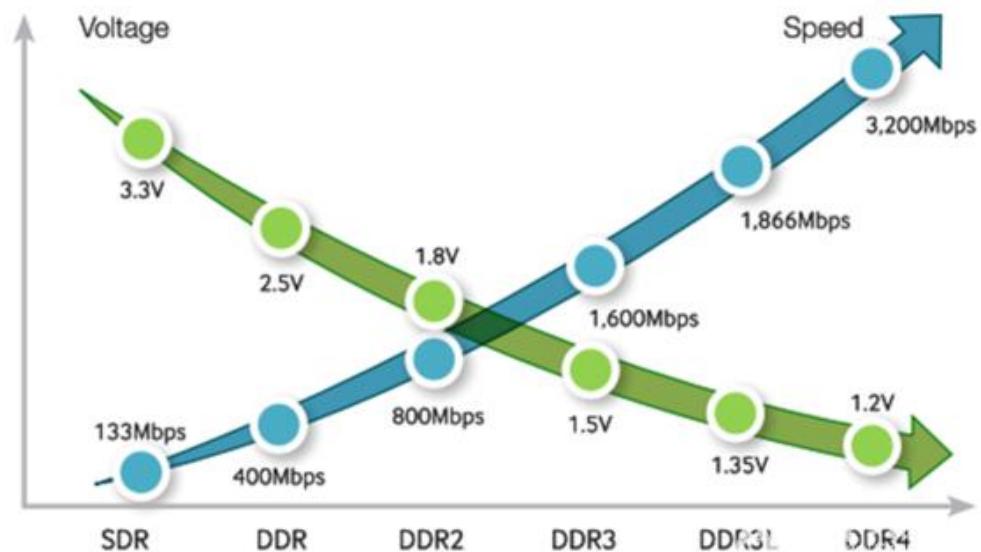
[Samsung, sub-70nm DRAM]

DDR (双倍速率同步动态随机存储器)



DRAM数据传输标准

| Standard | I/O clock rate | M transfers/s |
|----------|----------------|---------------|
| DDR1 | 133 | 266 |
| DDR1 | 150 | 300 |
| DDR1 | 200 | 400 |
| DDR2 | 266 | 533 |
| DDR2 | 333 | 667 |
| DDR2 | 400 | 800 |
| DDR3 | 533 | 1066 |
| DDR3 | 666 | 1333 |
| DDR3 | 800 | 1600 |
| DDR4 | 1333 | 2666 |



为什么叫双倍速率?

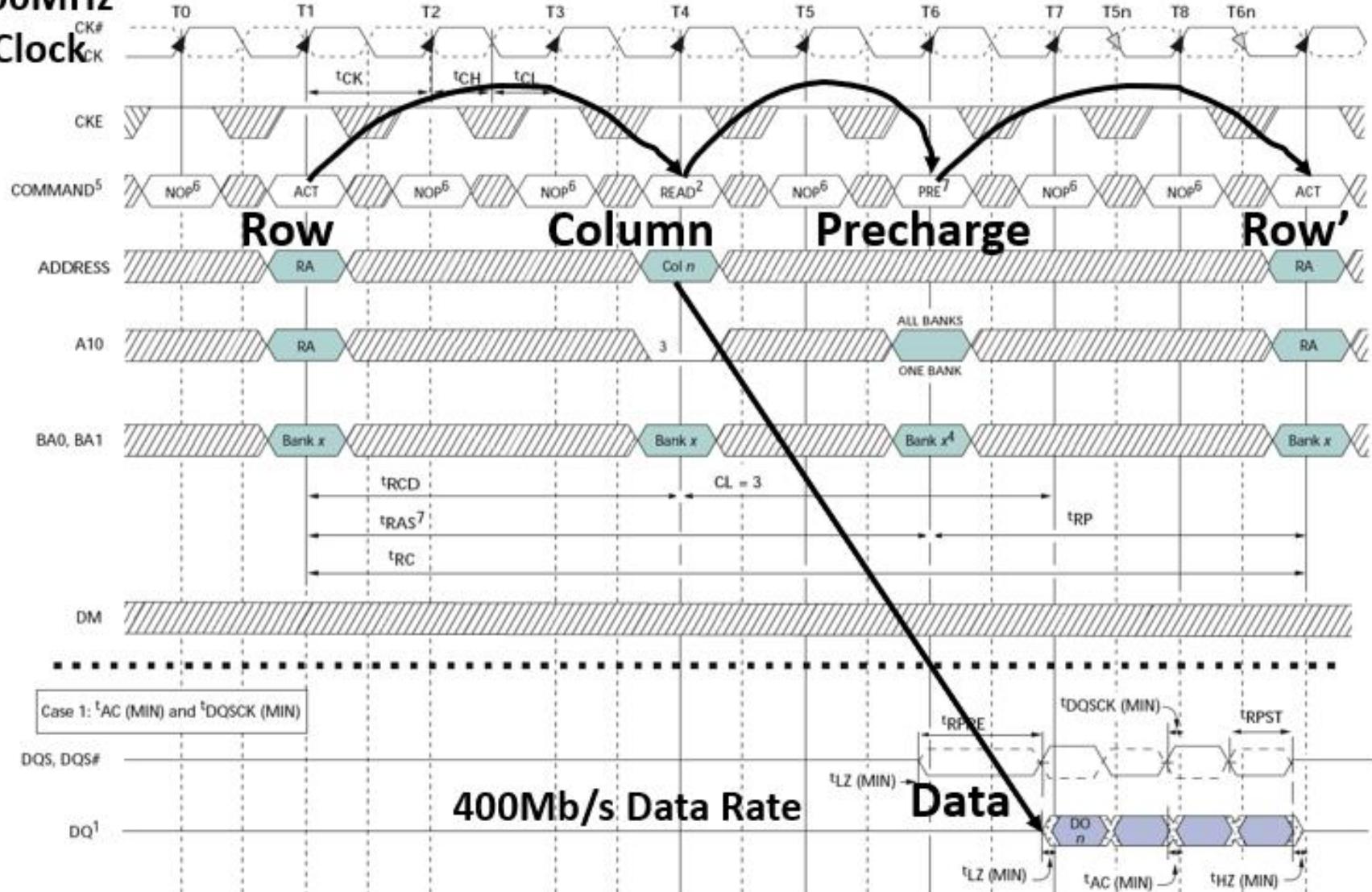




DDR 操作 (三步)

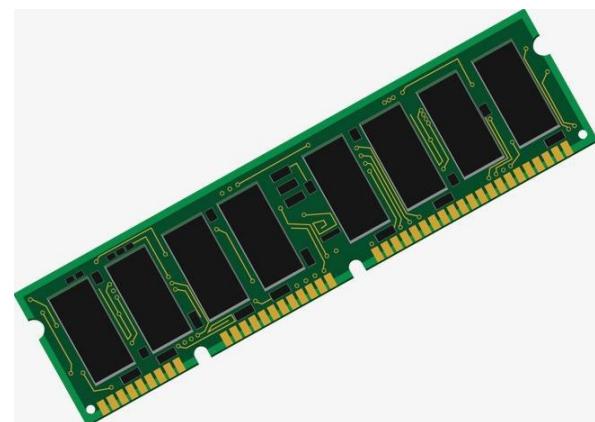
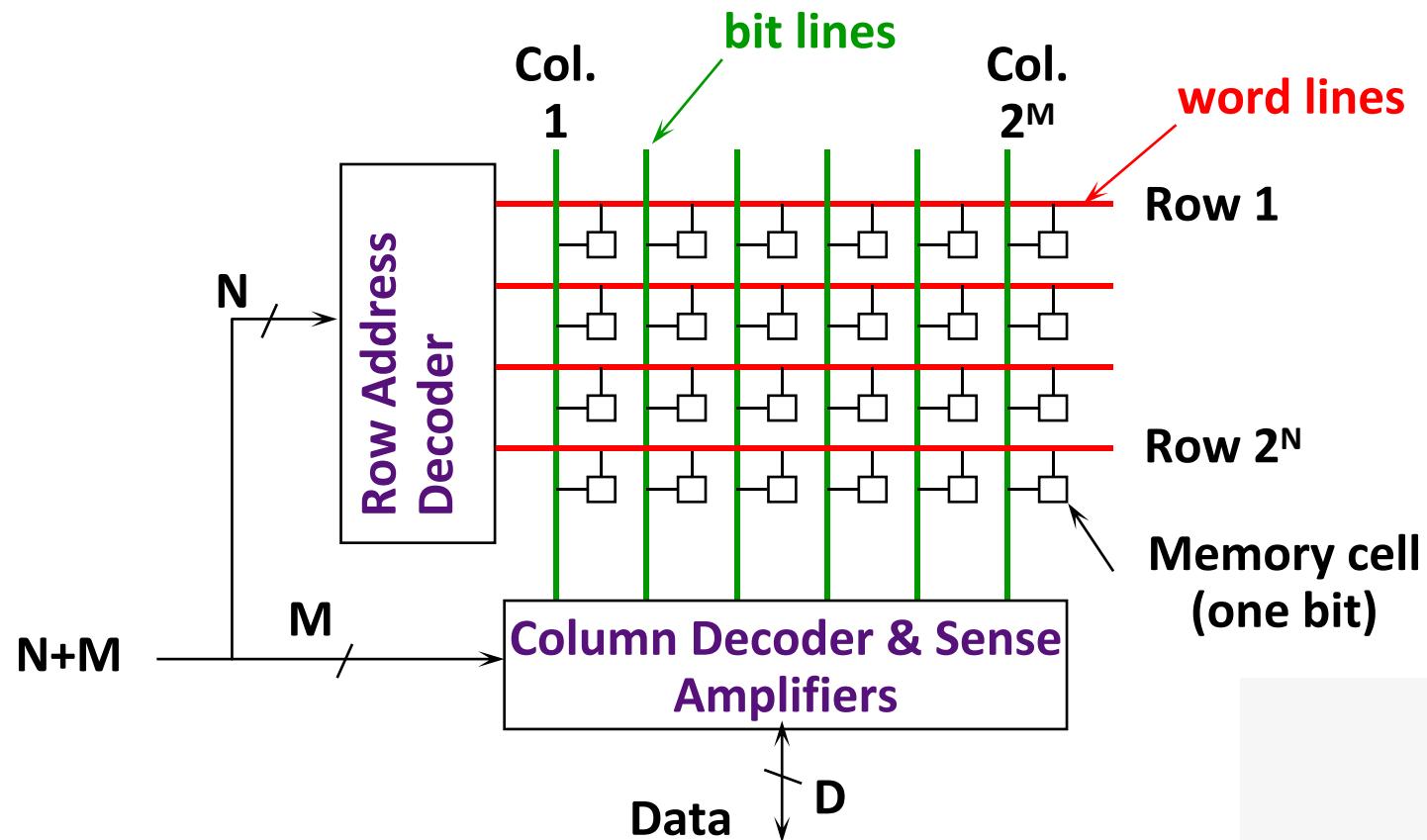
200MHz

Clock_{CK}





DRAM存储阵列





存储器容量的扩展

- 位扩展方式
- 字扩展方式



存储器容量的扩展

存储器容量的扩展

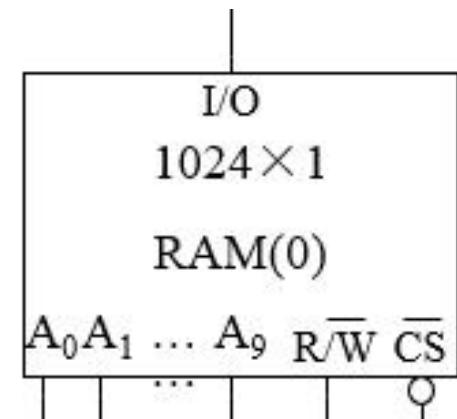
位扩展

8位、16位、64位、32位、64位...

➤ 位扩展方式：字数（存储单元的个数）够用，而位数（字的宽度）不够

➤ 扩展连线方法：

- 地址线并联
- 控制线并联 (R/\bar{W} 、 \bar{CS})
- 每一片的双向数据线 (I/O) 按位的高低排列好

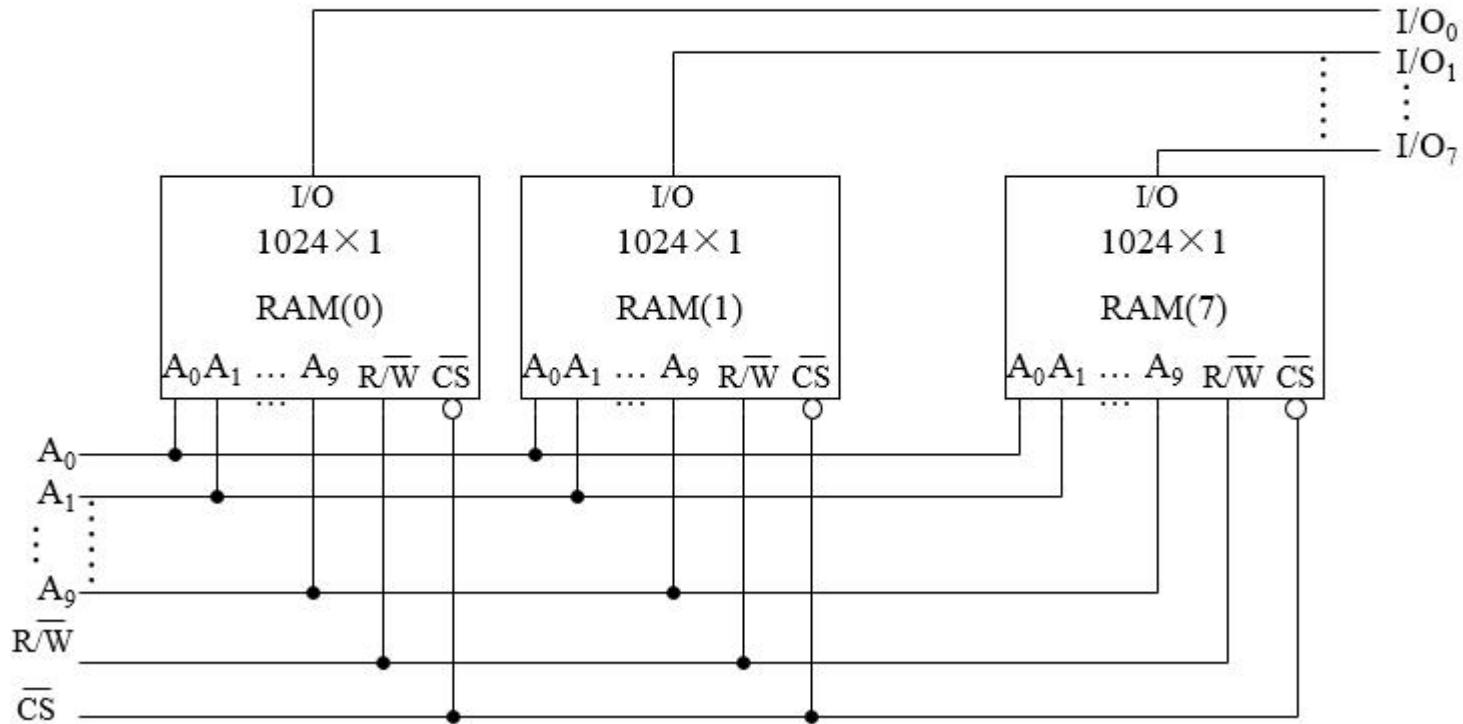


举例：将一个 $1024*1$ 的RAM扩展成 $1024*8$

存储器容量的扩展

位扩展

例：将 1024×1 的RAM扩展成 1024×8





存储器容量的扩展

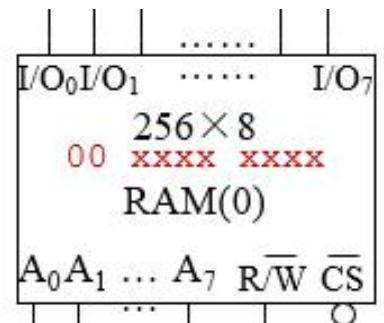
字扩展

- 将RAM (ROM) 接成字数更多的存储器
- 地址扩展

例：

用 256×8 位RAM接成 1024×8 位

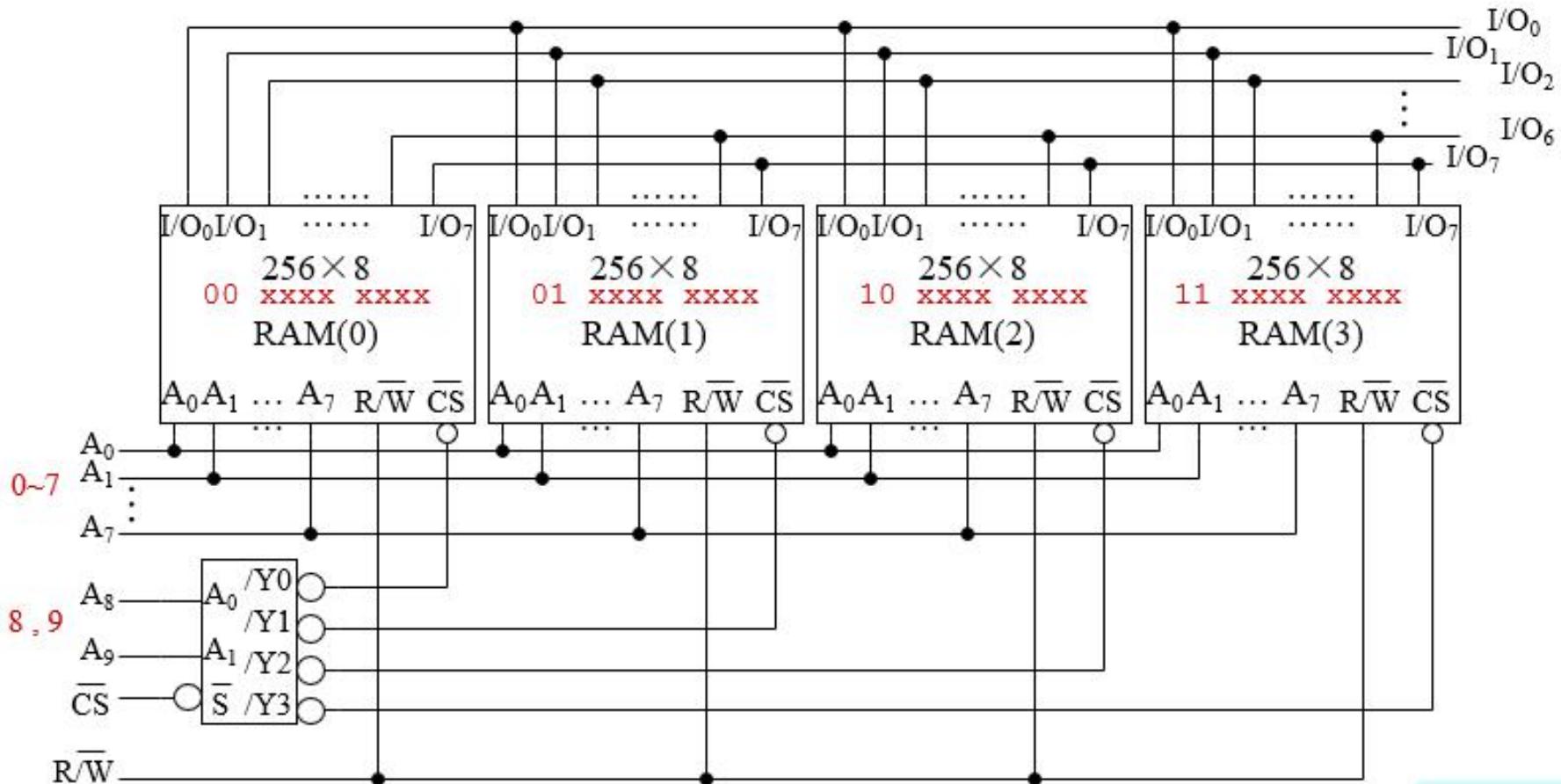
- 用4片
- “然而每片RAM上的地址输入范围只是8位，给出的地址范围全是0~255，无法区分4片中同样的地址单元”
- 回顾：寻址——译码 · 产生新的译码信号实现选片逻辑





存储容量的扩展

字扩展





存储容量的扩展

字扩展

■ 字扩展的连线方法

➤ 地址线

- 原有的地址线并联；
- 用译码器对增加的外部地址线进行译码，译码后的 2^{n-m} 根线分别连接RAM的片选，实现选片功能

➤ 数据线并联

➤ 控制线：

- 读写脉冲 R/\overline{W} 并联
- 新的 \overline{CS} 信号（利用译码器上的选通信号）



存储容量的扩展

存储器扩展

如果RAM(ROM)的位数和字数都不够用，就需要同时采用位扩展和字扩展

- CPU的地址线——
(要考虑实际应用中需要的地址空间的大小)
- CPU的数据线——
- 地址空间 (编址) —— (映像) —— 存储器

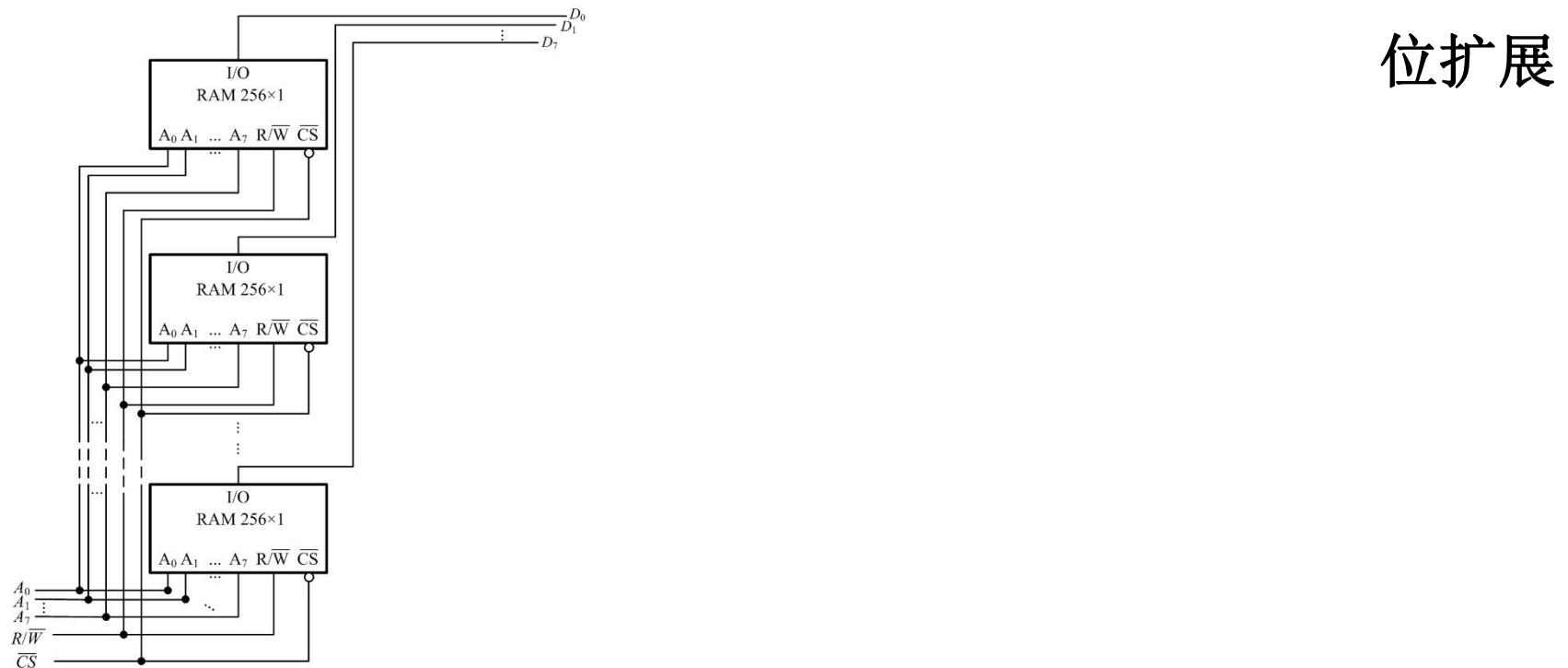
存储器容量的扩展



- 例：以 256×1 的RAM芯片扩展为1k的内存，且数据线宽度为8。

➤ 分析：

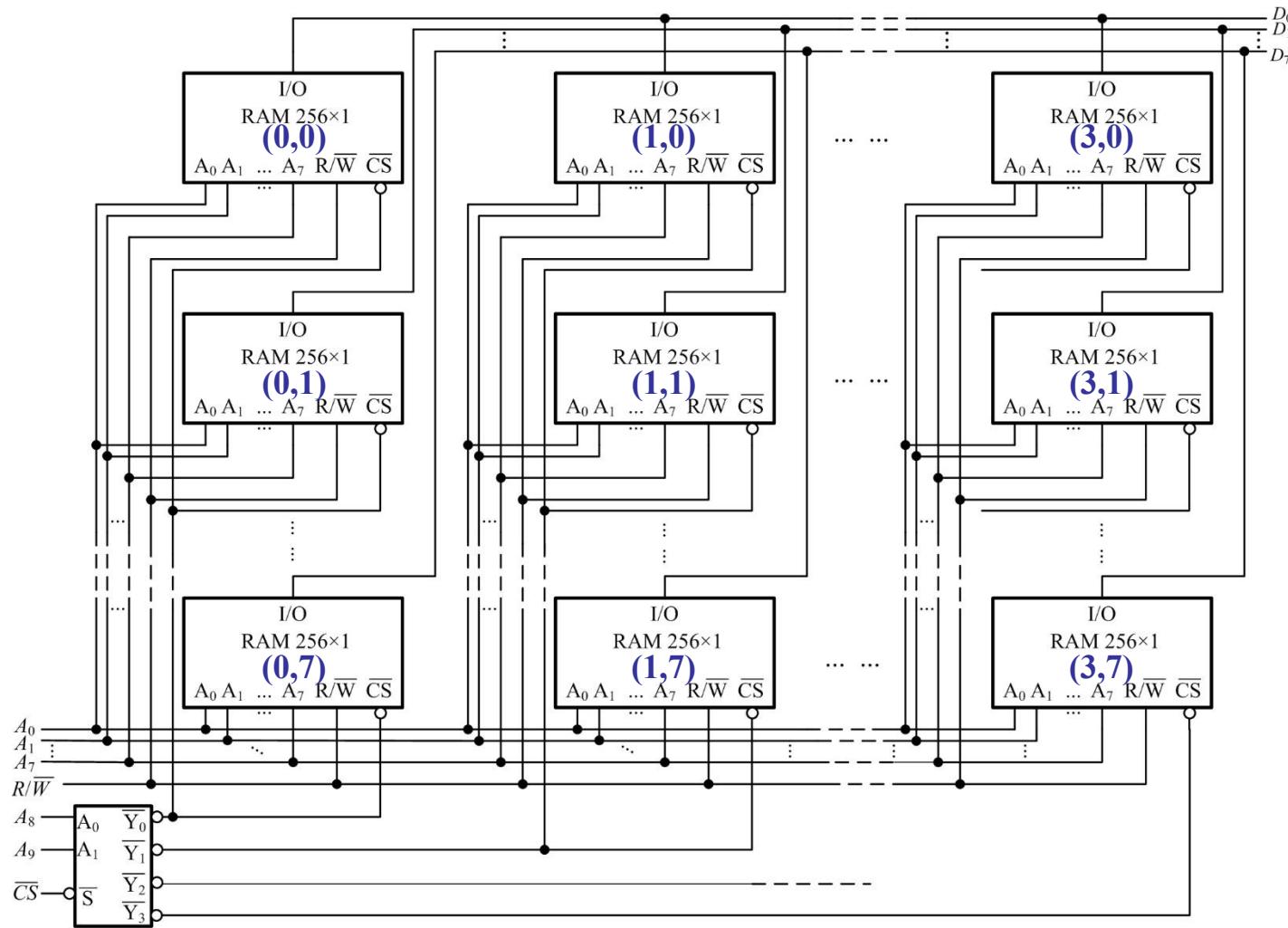
- 位扩展 $1\text{bit} \rightarrow 8\text{bit}$
- 字扩展 考虑1k的存储器需要几根地址线？——10根





存储器容量的扩展

例：以 256×1 的RAM芯片扩展为 $1k$ 的内存，且数据线宽度为8。



字扩展



存储器实现组合逻辑函数

■ ROM的主要用途

- 程序存储器

■ ROM的逻辑电路用途

- 实现组合逻辑函数



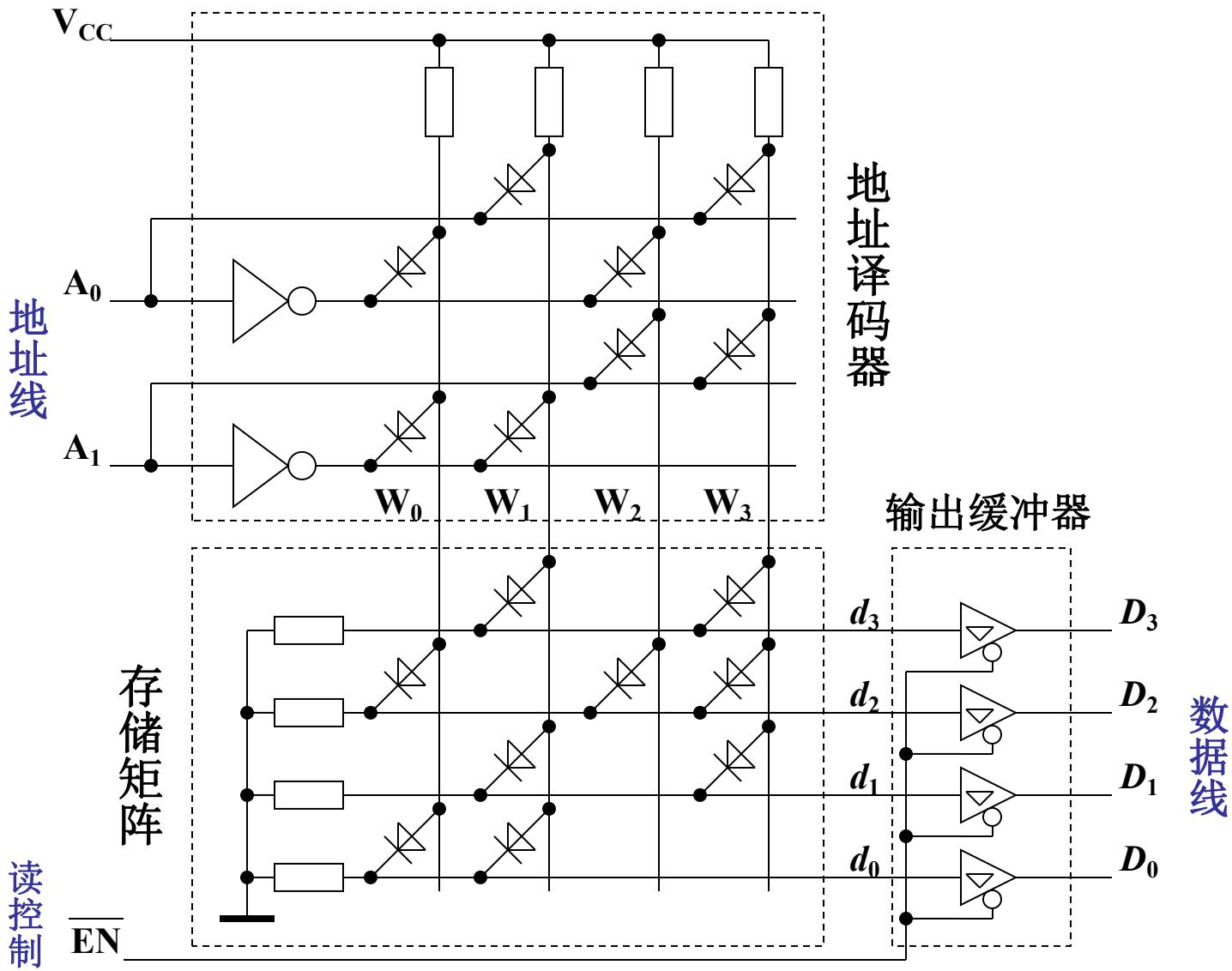
存储器实现组合逻辑函数

■ 回顾：ROM的电路特点

- ROM的地址译码器由许多“与门”组成，这些与门网络称为“与阵列”，若将地址变量看成输入逻辑变量，则地址译码器的输出（各条字线）便可代表输入变量的全部各个最小项。
- 存储网络是位线和字线在交叉点上的耦合（连接、不连接）形成，位线的输出为这些耦合元件的逻辑“或”，这些或门网络实现了最小项的或的逻辑运算。

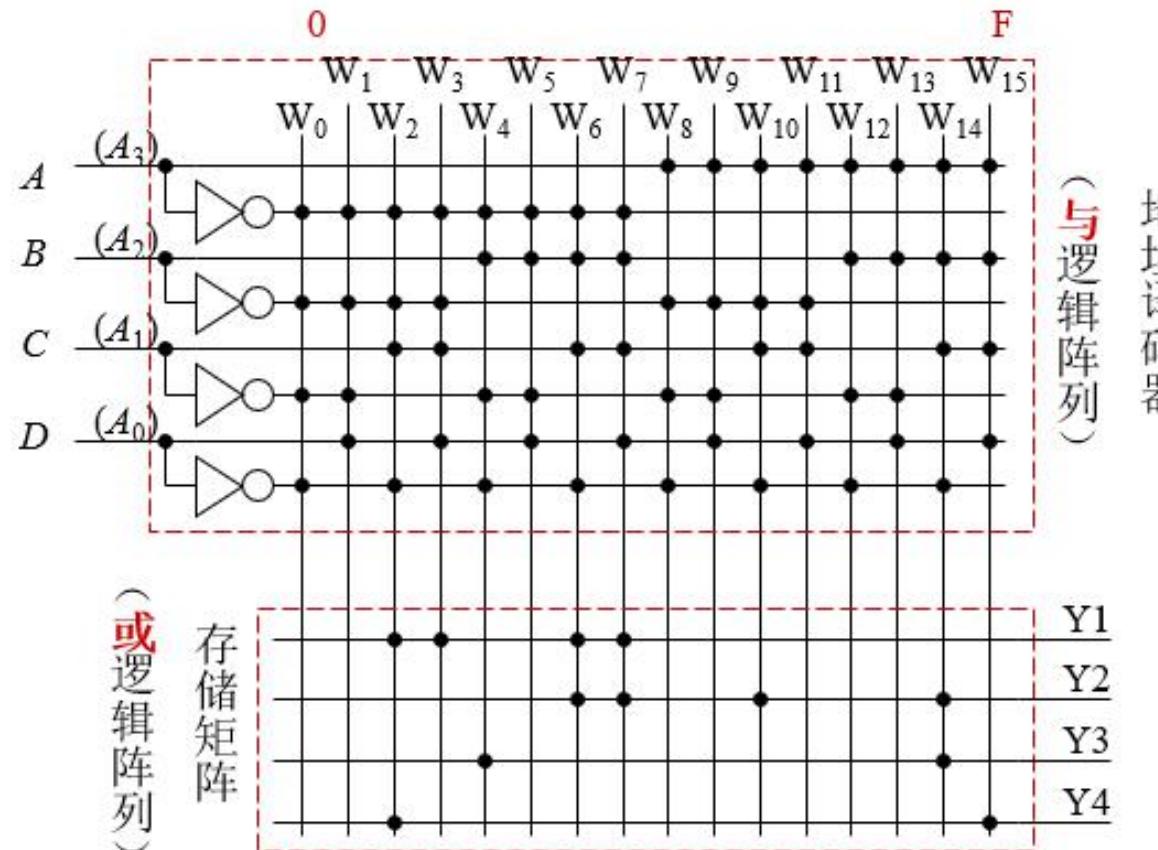


存储器实现组合逻辑函数





存储器实现组合逻辑函数



■ ROM内部简化作图

- **存储矩阵**
 - 不用画出晶体管，在接入存储器件的交叉点上画一个圆点——般以有圆点为1，没有为0。
 - “或”阵列。

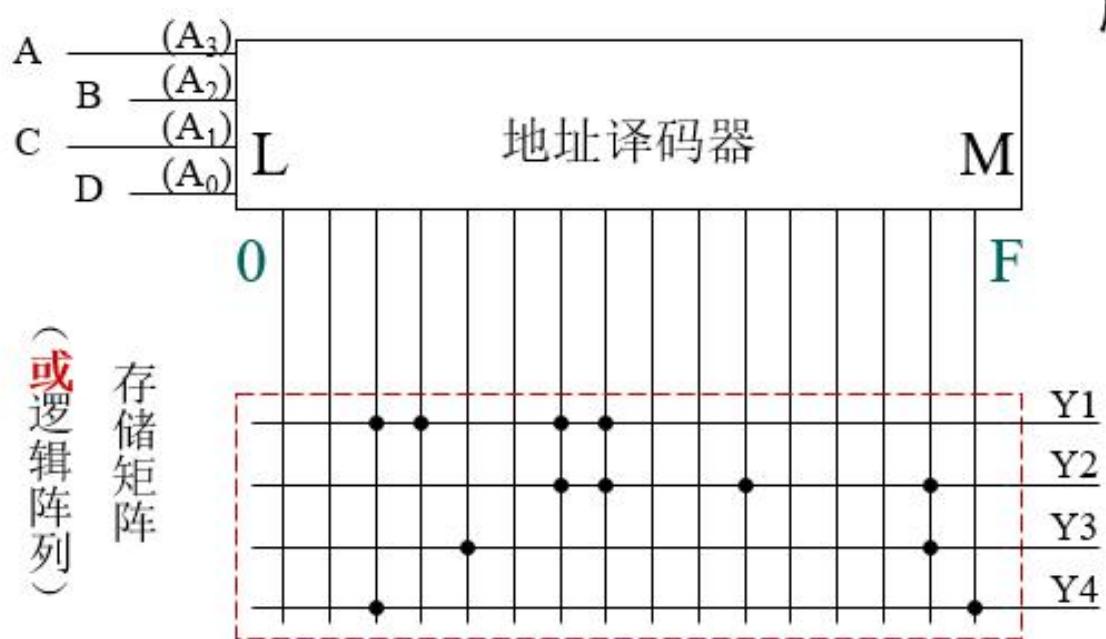
- **地址译码矩阵**
 - 也画圆点，但实际上它是“与”阵列。



存储器实现组合逻辑函数

■ ROM内部简化作图 (续)

- 不具体画出译码矩阵：只要明确最小项与地址输入的对应关系





存储器实现组合逻辑函数

■ 总之...

- ROM的地址译码器——实现了地址输入的全译码.....
- ROM可以实现个数与它的地址位数一样多的逻辑变量的逻辑函数
- 逻辑函数的个数 等于 其数据位宽度数



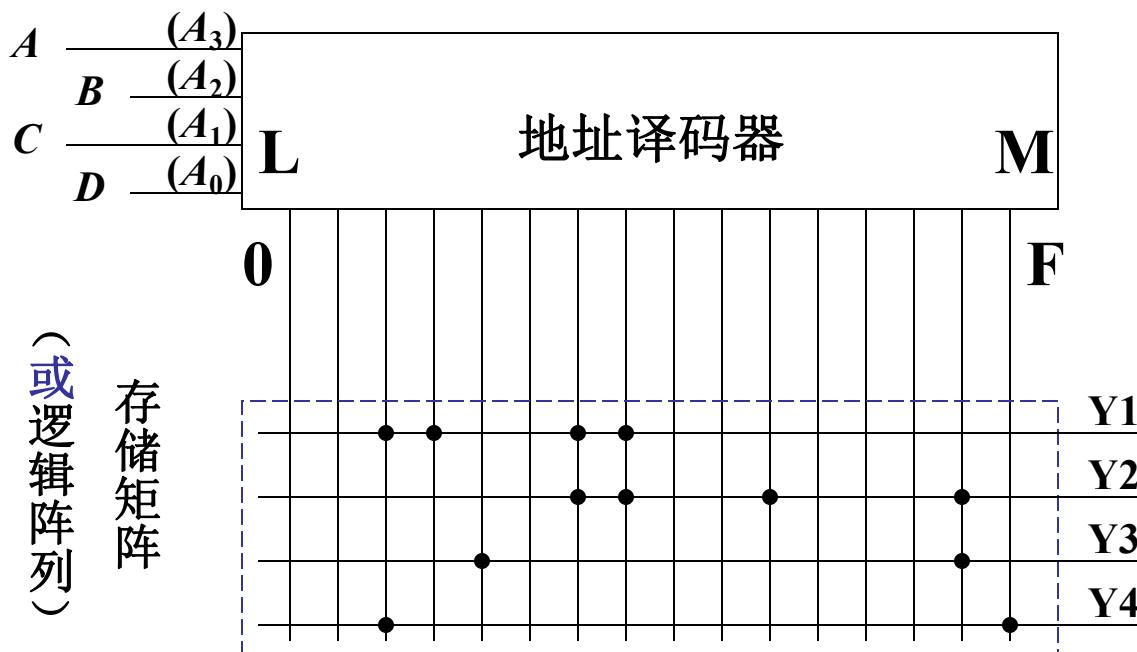
存储器实现组合逻辑函数

■ ROM组合逻辑分析

- 最小项和，化简

例：前页，图中的逻辑

$$\begin{aligned}Y_1 &= A'C \\Y_2 &= A'BC + ACD' \\Y_3 &= A'BC'D' + ABCD' \\Y_4 &= A'B'CD' + ABCD\end{aligned}$$





存储器实现组合逻辑函数

■ ROM组合逻辑设计

- 设计需求：
 - 给出的是功能定义，或者
 - 给出的是逻辑函数式，或者
 - 给出的是真值表
- 设计方法...



存储器实现组合逻辑函数

■ ROM组合逻辑设计（续）

- 逻辑表达设计需求
- 指定输入轨与地址线之间的关系
- 转换成逻辑函数的最小项之和表达式
- 根据逻辑函数的输入、输出变量数目，确定ROM的容量，选择合适的ROM，画出ROM的阵列图
- 根据阵列图对ROM进行编程



存储器实现组合逻辑函数

■ ROM组合逻辑设计（续）

- 例：用ROM构成 $y=x^2$ 运算电路， x 取值范围为 0~15正整数





存储器实现组合逻辑函数

✓ $Y_7 = m_{12} + m_{13} + m_{14} + m_{15}$

✓ $Y_6 = m_8 + m_9 + m_{10} + m_{11} + m_{14} + m_{15}$

✓ $Y_5 = m_6 + m_7 + m_{10} + m_{11} + m_{13} + m_{15}$

✓ $Y_4 = m_4 + m_5 + m_7 + m_9 + m_{11} + m_{12}$

✓ $Y_3 = m_3 + m_5 + m_{11} + m_{13}$

✓ $Y_2 = m_2 + m_6 + m_{10} + m_{14}$

✓ $Y_1 = 0$

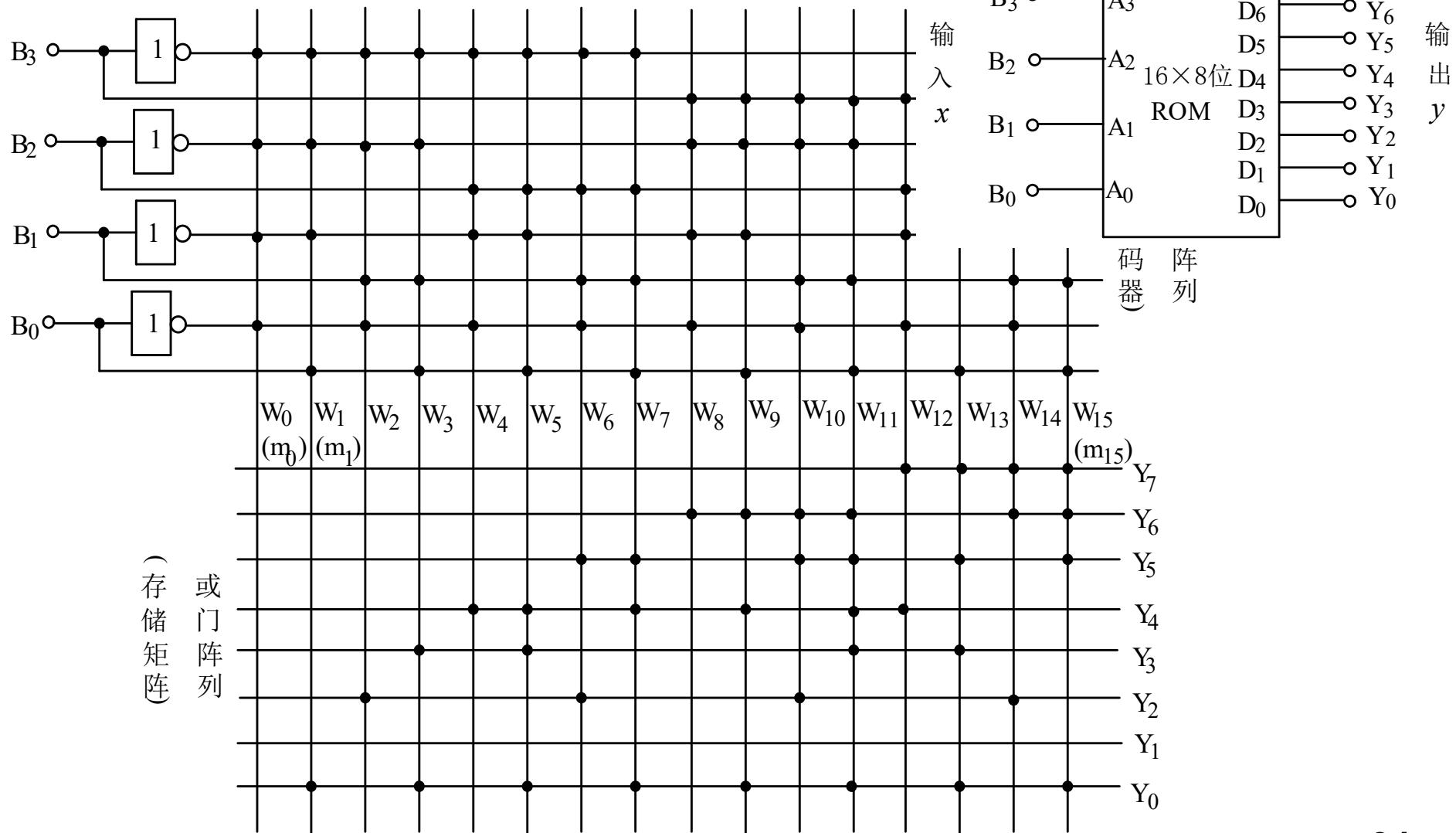
✓ $Y_0 = m_1 + m_3 + m_5 + m_7 + m_9 + m_{11} + m_{13} + m_{15}$

| X_3 | X_2 | X_1 | X_0 | Y_7 | Y_6 | Y_5 | Y_4 | Y_3 | Y_2 | Y_1 | Y_0 | N_{10} |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 9 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 16 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 36 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 49 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 64 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 81 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 100 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 121 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 144 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 169 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 196 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 225 |



存储器实现组合逻辑函数

✓ ROM存储矩阵连接图





存储器实现组合逻辑函数

■ ROM组合逻辑设计（续）

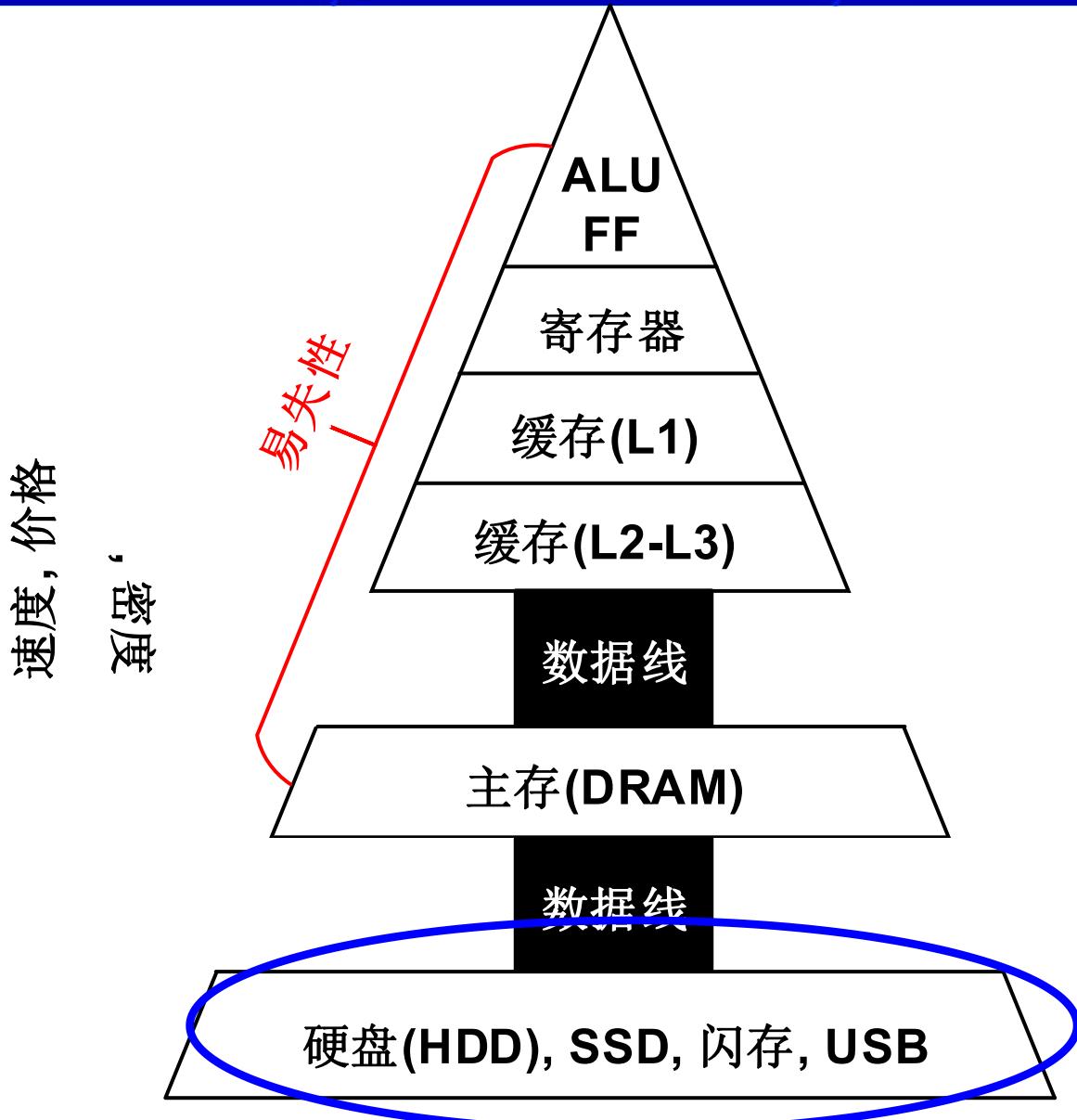
- 例：用ROM构成代数运算 $z = x \cdot y$ 的电路

- ◆ 思考：如果乘数宽度都是 n -bit，所需ROM的存储量是多少？
 - ◆ 回答：输入地址线 $2n$ ，但乘积数目是 2^{2n} ，输出位线 $2n$ 。需要容量 $2^{2n} \times 2n$ 的ROM。
 - » 如两个4-bit相乘，需要： $2^8 \times 8 = 2048$

- 问题：资源消耗随问题规模 n 指数性增长。



存储器层次结构



机械硬盘



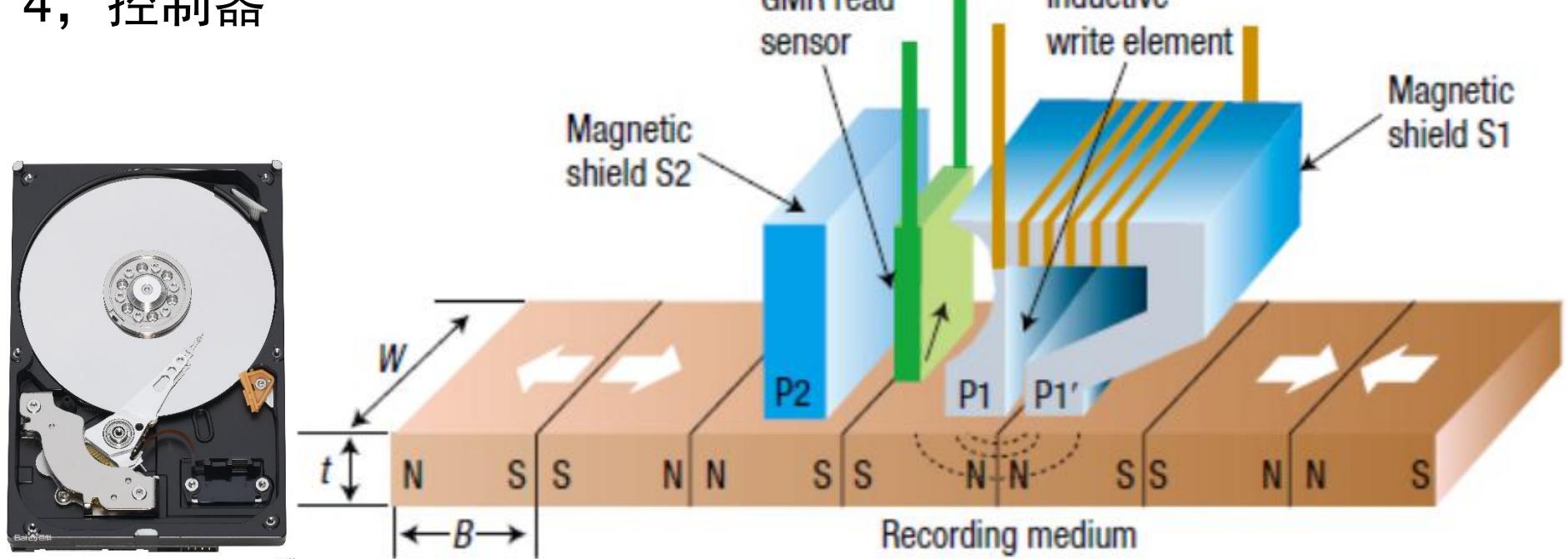
From baidu



硬盘组成与原理

机械硬盘：

- 1, 盘片, 数据存储媒介, 磁粉附在铝合金/玻璃圆盘基上
- 2, 磁头, 最精密的部分, 读写数据信息
- 3, 机械传功: 提供盘片运转动力
- 4, 控制器





硬盘组成与原理

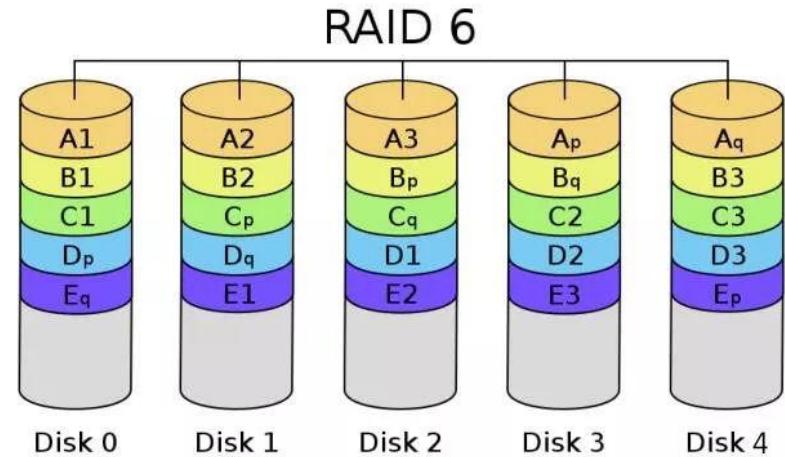
YOUKU





磁盘阵列（RAID）

磁盘阵列（Redundant Arrays of Independent Drives, RAID），有“独立磁盘构成的具有冗余能力的阵列”之意。



- **镜像：**冗余技术，防止磁盘发生故障而数据丢失；
- **数据条带：**将数据以块的方式分布存储在多个磁盘中，进行并发处理；
- **数据校验：**配合数据条带，提供错误校验恢复功能；比如奇偶校验



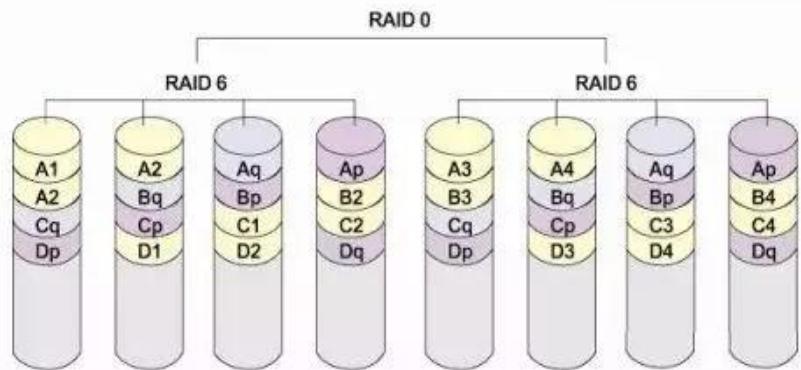
磁盘阵列 (RAID)

| RAID 等级 | RAID0 | RAID1 | RAID5 | RAID6 | RAID10 |
|---------|------------|-----------------|------------|------------|-----------------------|
| 别名 | 条带 | 镜像 | 分布奇偶校验条带 | 双重奇偶校验条带 | 镜像加条带 |
| 容错性 | 无 | 有 | 有 | 有 | 有 |
| 冗余类型 | 无 | 有 | 有 | 有 | 有 |
| 热备盘 | 无 | 有 | 有 | 有 | 有 |
| 读性能 | 高 | 低 | 高 | 高 | 高 |
| 随机写性能 | 高 | 低 | 一般 | 低 | 一般 |
| 连续写性能 | 高 | 低 | 低 | 低 | 一般 |
| 需要磁盘数 | $n \geq 1$ | $2n (n \geq 1)$ | $n \geq 3$ | $n \geq 4$ | $2n(n \geq 2) \geq 4$ |
| 可用容量 | 全部 | 50% | $(n-1)/n$ | $(n-2)/n$ | 50% |

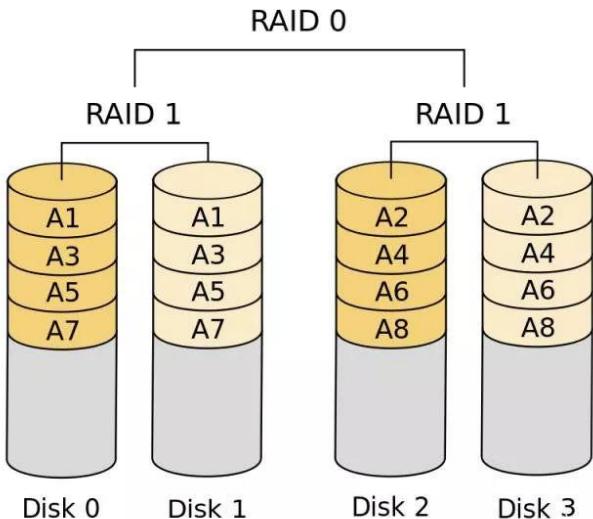


磁盘阵列 (RAID)

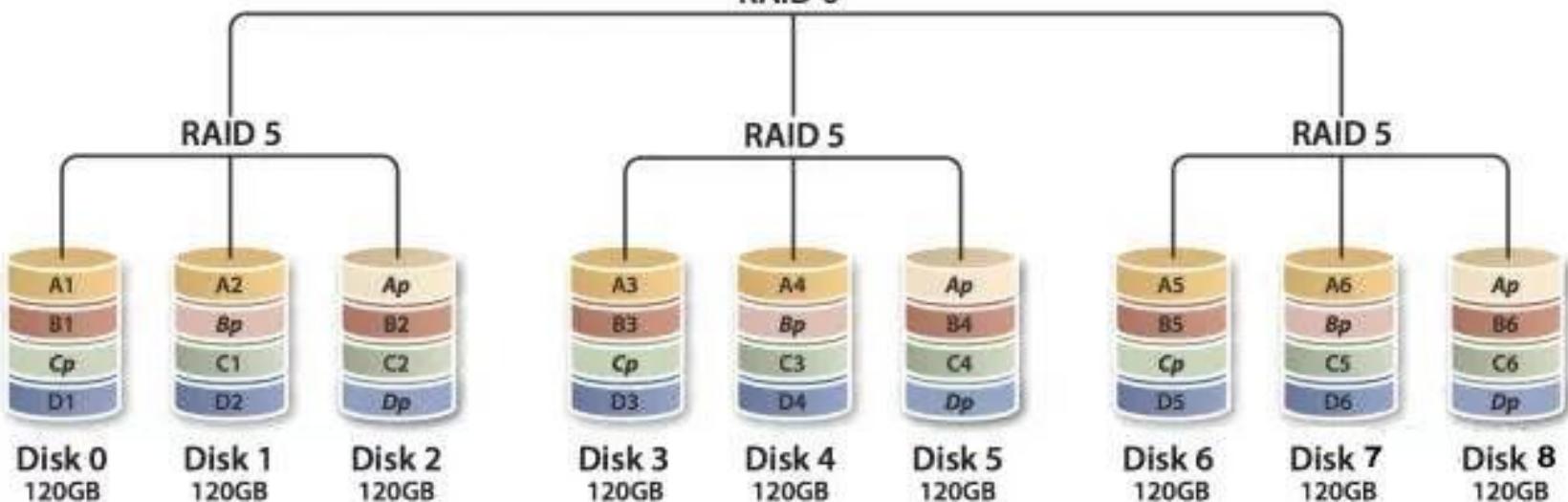
RAID 60



RAID 1+0



RAID 0

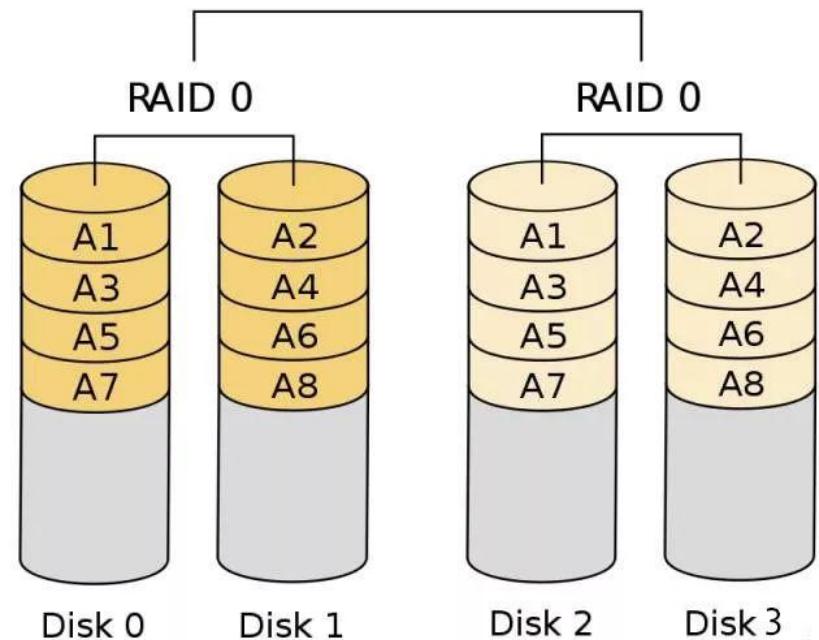




磁盘阵列 (RAID)

RAID 0+1

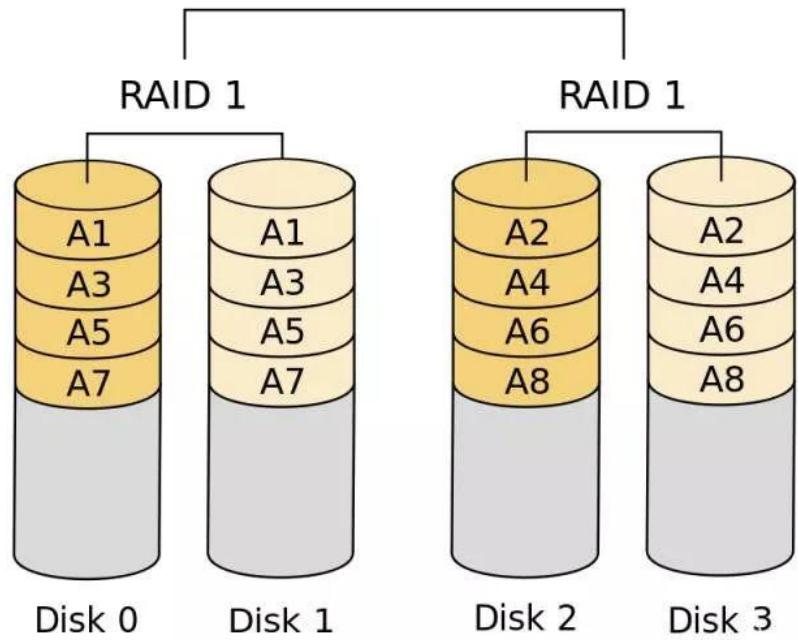
RAID 1



RAID 1+0

RAID 0

VS



RAID 0: 条带

RAID 1: 镜像



自己动手组建 RAID0 与 RAID1

The screenshot shows the BIOS SETUP UTILITY interface. The main menu bar includes Extreme Tweaker, Main, Advanced, Power, Boot, Tools, and Exit. The Main tab is selected. On the left, a list of options includes System Time, System Date, Language, SATA 1 through SATA 6, Storage Configuration, and System Information. The Storage Configuration option is highlighted with a red rectangle. The right panel displays system information: System Time (14:49:48, Wed 08/18/2010, English), and a table for SATA drives (SATA 1: Hitachi HDS7220200, SATA 2: Hitachi HDS7220200, SATA 3: Not Detected, SATA 4: Not Detected, SATA 5: Not Detected, SATA 6: Not Detected). A legend on the right defines keyboard shortcuts: F10 for Save and Exit, ESC for Exit, Enter for Go to Sub Screen, F1 for General Help, T1 for Select Item, and F5 for Select Screen.

<https://jingyan.baidu.com/article/2fb0ba40e5bce800f2ec5fa1.html>

<https://jingyan.baidu.com/article/20095761e3e568cb0721b401.html>



固态硬盘：NAND型



NAND存储阵列 + 控制器



Toshiba&
Sandisk

TOSHIBA
SanDisk

Intel&
Micron

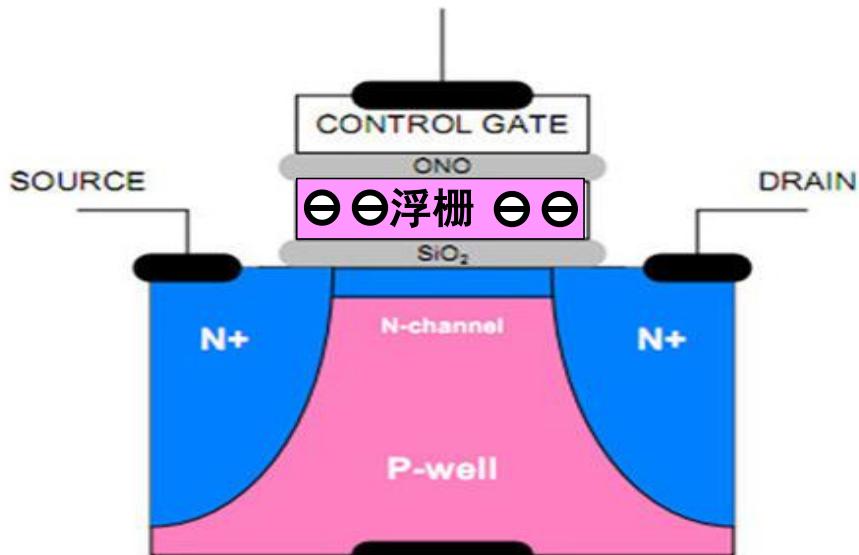
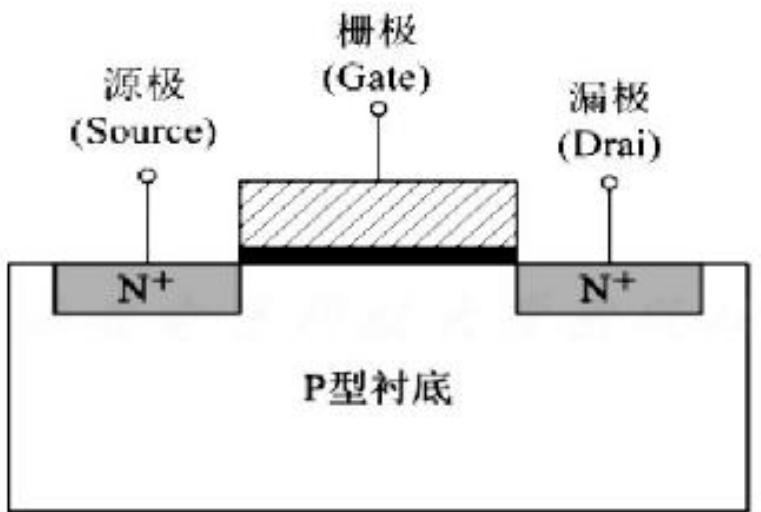
Micron
intel

SK-hynix

SK hynix

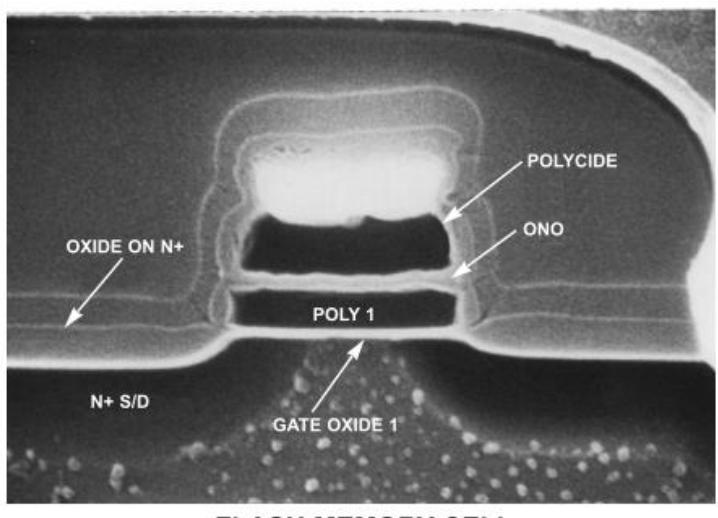


固态硬盘基本器件：浮栅晶体管



晶体管

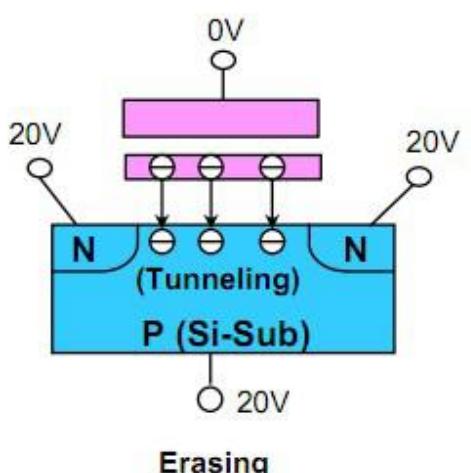
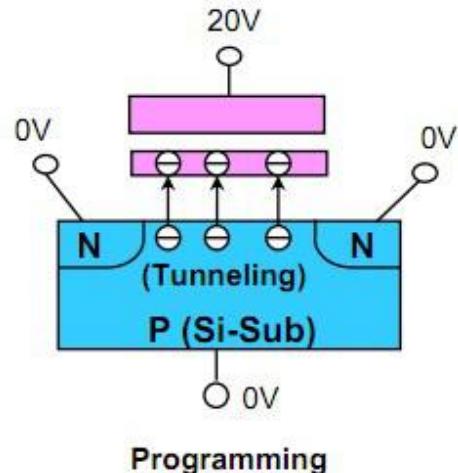
浮栅晶体管，相比普通晶体管多了一层浮栅，能够存储电子。浮栅中有电子，则代表存储1；浮栅中无电子，则代表存储0。



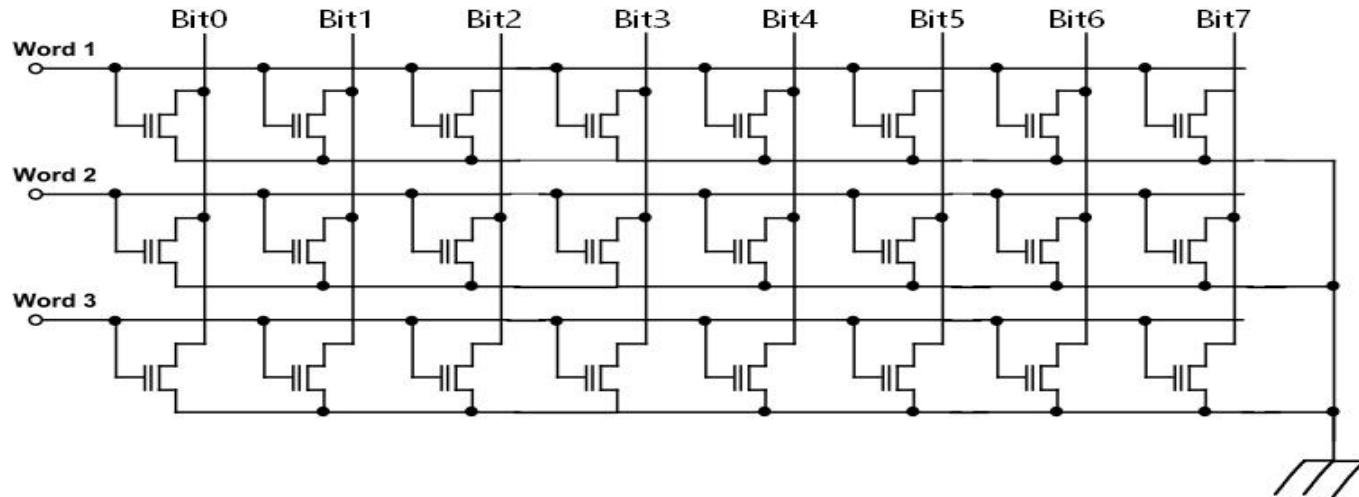
FLASH MEMORY CELL



浮栅晶体管单元与Nor Flash 阵列



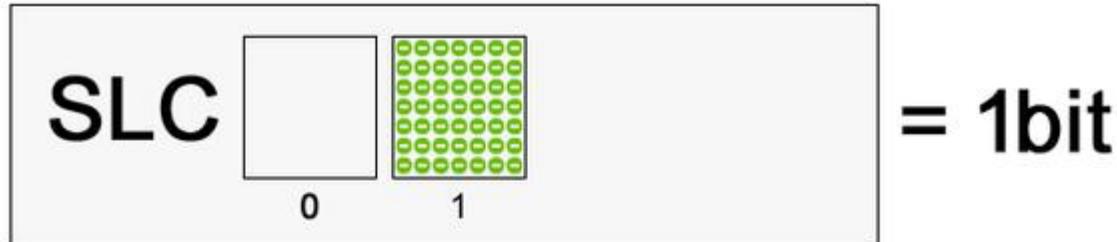
- 编程：栅极加上高电压，源极和衬底接地
- 擦除：栅极接地，源极漏极和衬底加高电压



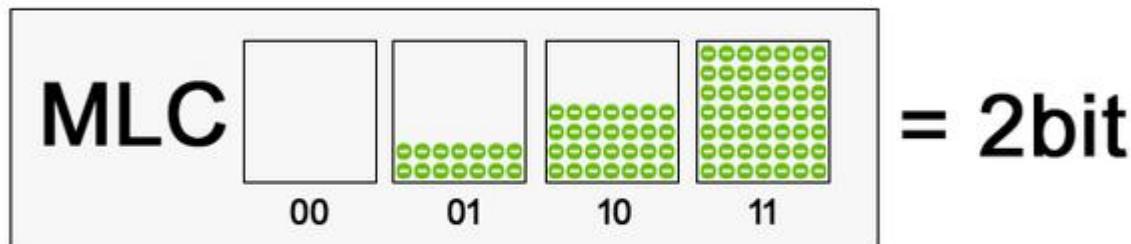


浮栅晶体管：单比特 VS 多比特

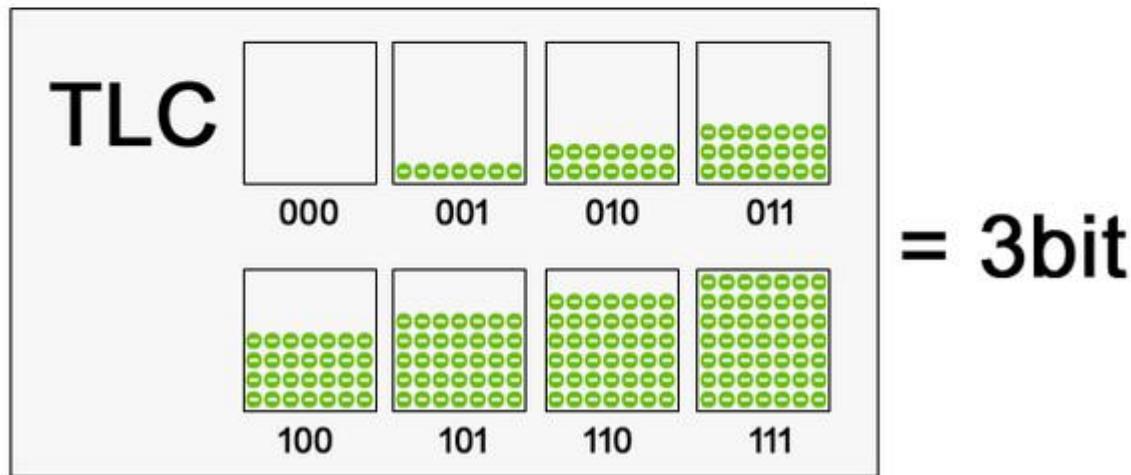
Sigle level cell



Multi level cell



Triple level cell



能够区分出浮栅中不同电子数目

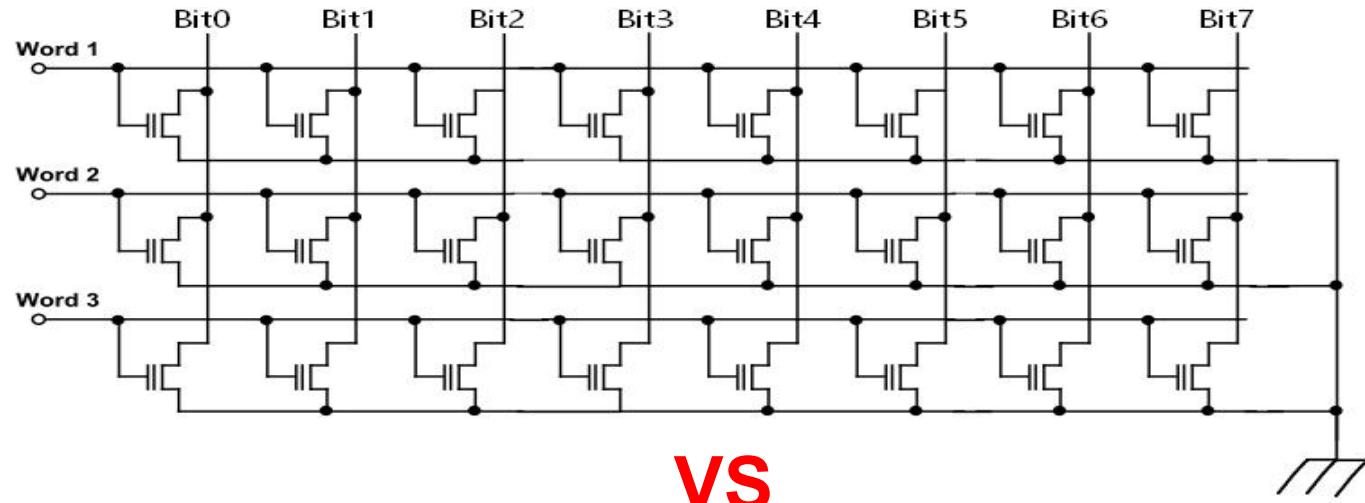


浮栅晶体管：单比特 VS 多比特

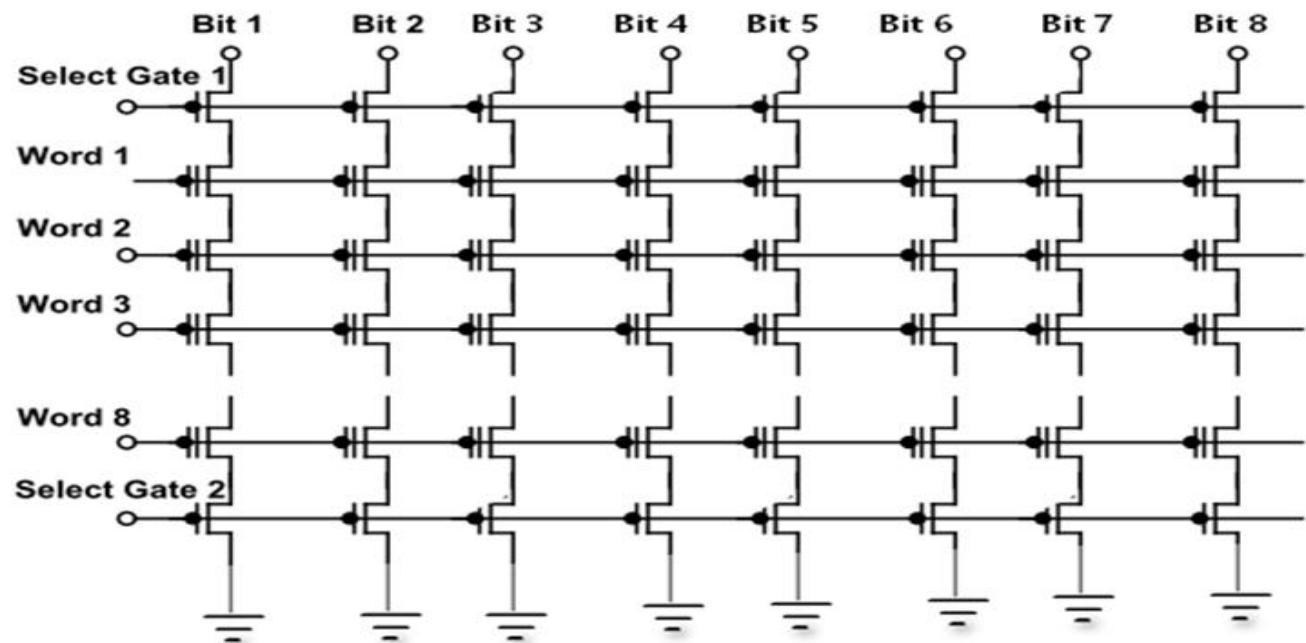
| Parameter | SLC | MLC | TLC |
|------------------------|-------------|-----------|-------------|
| Bit per cell (存储密度) | 1 | 2 | 3 |
| P/E cycle(寿命) | 100000 | 3000~5000 | 500~1000 |
| Read time (读取时间) | 25μs | 50μs | 75μs |
| Program time (写入时间) | 200~300μs | 600~900μs | 900~16000μs |
| Erase time (擦除时间) | 1500~2000μs | 3000μs | 4500μs |



闪存：NOR型 VS. NAND型



VS



NOR型：
每个存储
单元可以
单独访问

NAND型：
同一行必须
同时访问



闪存：NOR型 VS. NAND型

| | NOR Flash | NAND Flash |
|------|----------------|--------------------------|
| 密度 | 小 | 大 |
| 容量 | 小 | 大 |
| 写入速度 | 慢 | 快 |
| 读取速度 | 快 | 慢 |
| 成本价格 | 高 | 低 |
| 应用场景 | 程序存储器 嵌入式应用 | 大容量数据存储器 如 U 盘, SSD 等 |



Flash 微缩问题

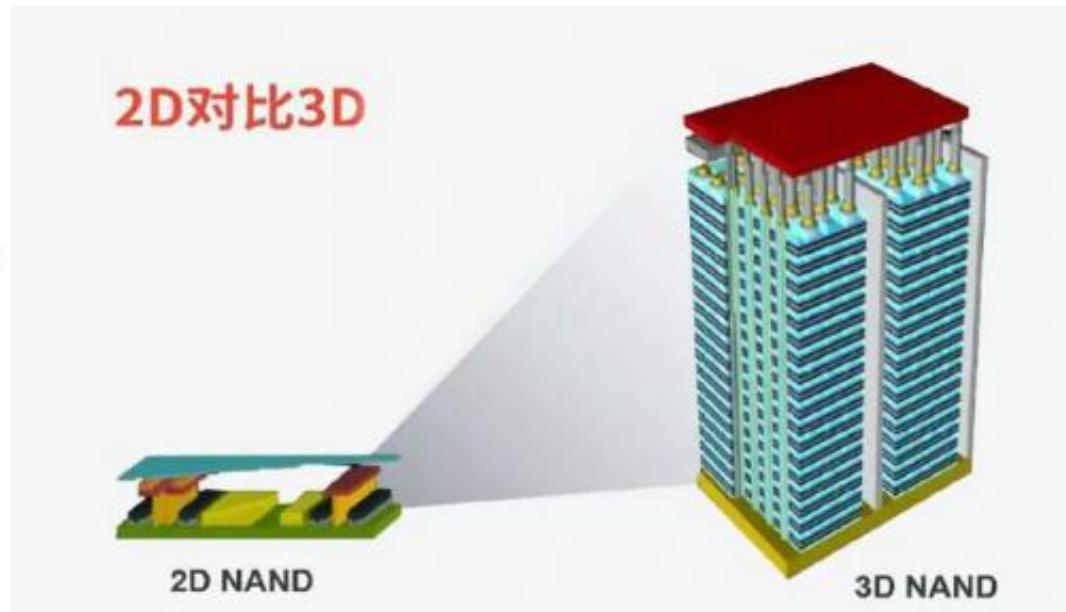
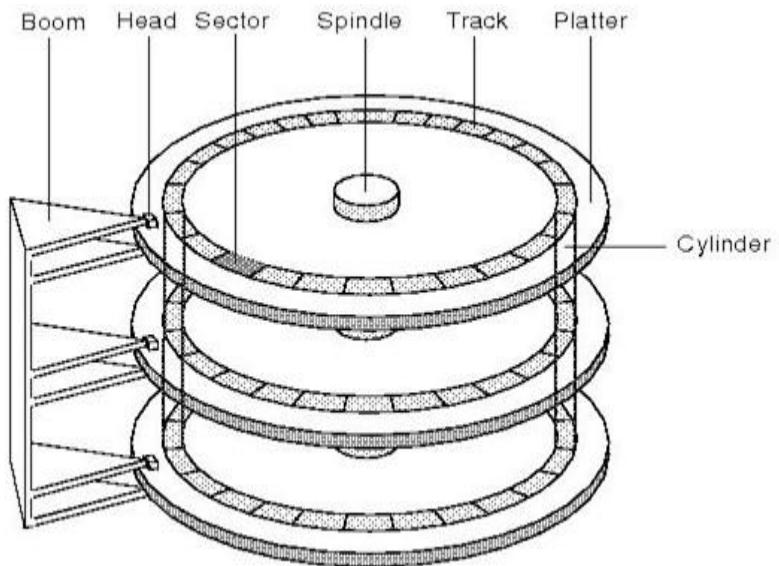
How flash technology scales:

scaling > less electrons > performance degrade



尺寸缩小，浮栅存储的电子数量减小，带来可靠性问题，
数据保持时间问题等，目前主流是28nm，往下微缩较难

2D Flash VS. 3D Flash



硬盘多层盘片叠加

3D NAND多层叠加



3D NAND Flash

3D NAND Architecture

垂直贯穿漏端选择栅SGD

SGD

WL

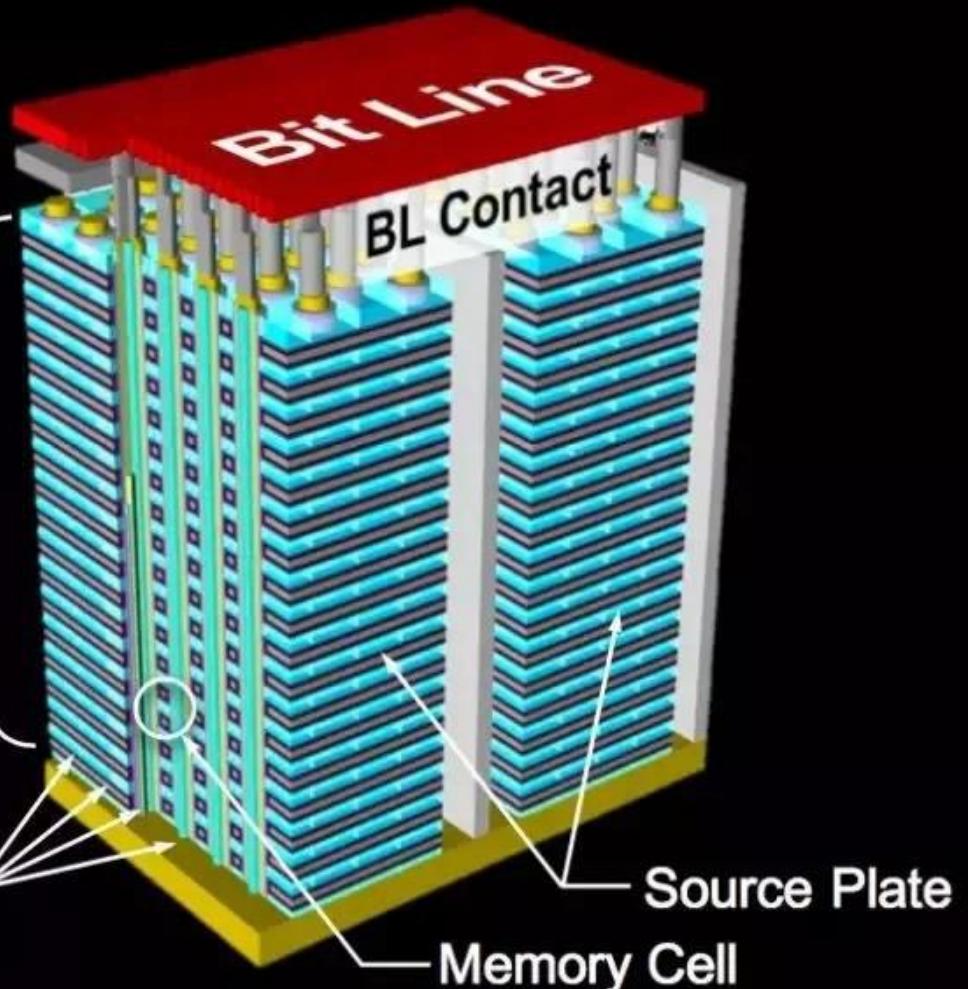
垂直贯穿源端选择栅SGS

SGS

Memory Holes

Source Plate

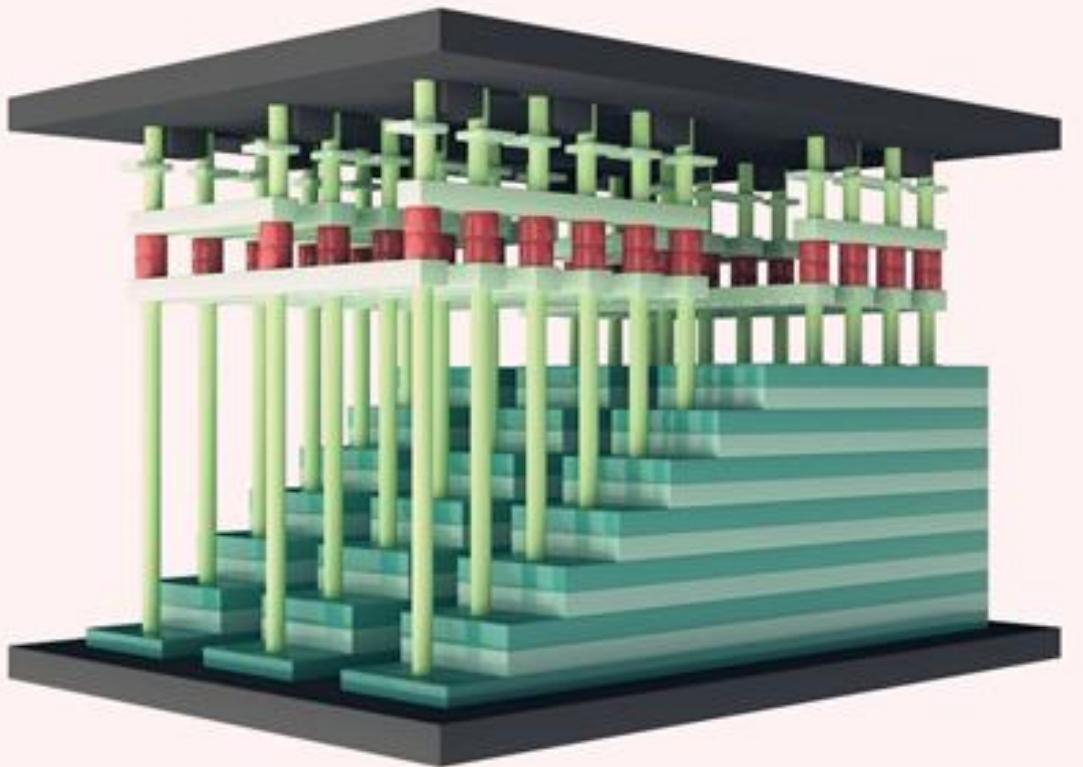
Memory Cell



东芝和西部数据已研发128层3D NAND Flash



长江存储 (YMTC) : 3D NAND



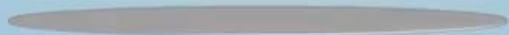
YMTC Xtacking™ Architecture



长江存储2019年9月量产64层3D NAND Flash



长江存储 (YMTC) : Xtacking



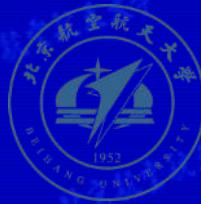
国际首创，自主知识产权



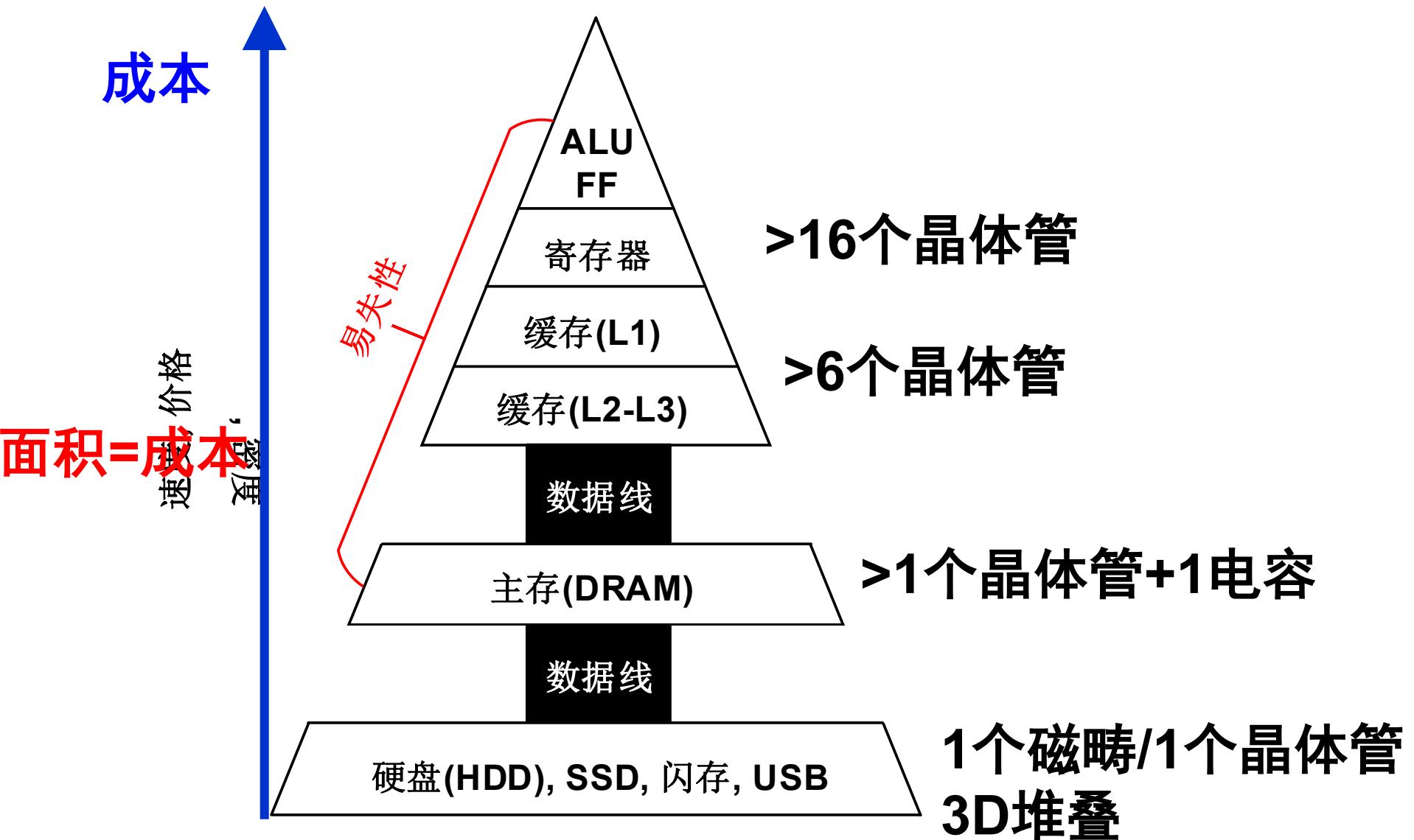
固态硬盘 VS 机械硬盘

| 参数 | SSD(主流256G) | 机械硬盘(2.5" 500G) |
|--------|-------------|-------------------------|
| 更快 | 连续读取 | 450~500 Mb/S |
| | 连续写入 | 300~500 Mb/S |
| | 4K随机读取 | 60,000~10,000 IOPS |
| | 4K随机写入 | 40,000~90,000 IOPS |
| | 读取时间 | 0.04~0.2ms |
| | 写入时间 | 0.04~0.2ms |
| 更轻巧节能 | 尺寸 | Min: 42x22x2.75(3.85)mm |
| | 重量 | Min 20g |
| | 运行功耗 | 2~4.5W |
| 更宽适用范围 | 操作温度 | 0~70 °C |
| | 抗震/噪音 | 抗震, 无噪音 |
| 价格较贵 | 价格 | 359~699 |
| 寿命短 | 寿命 | 1000~3000P/E |

服务器居多

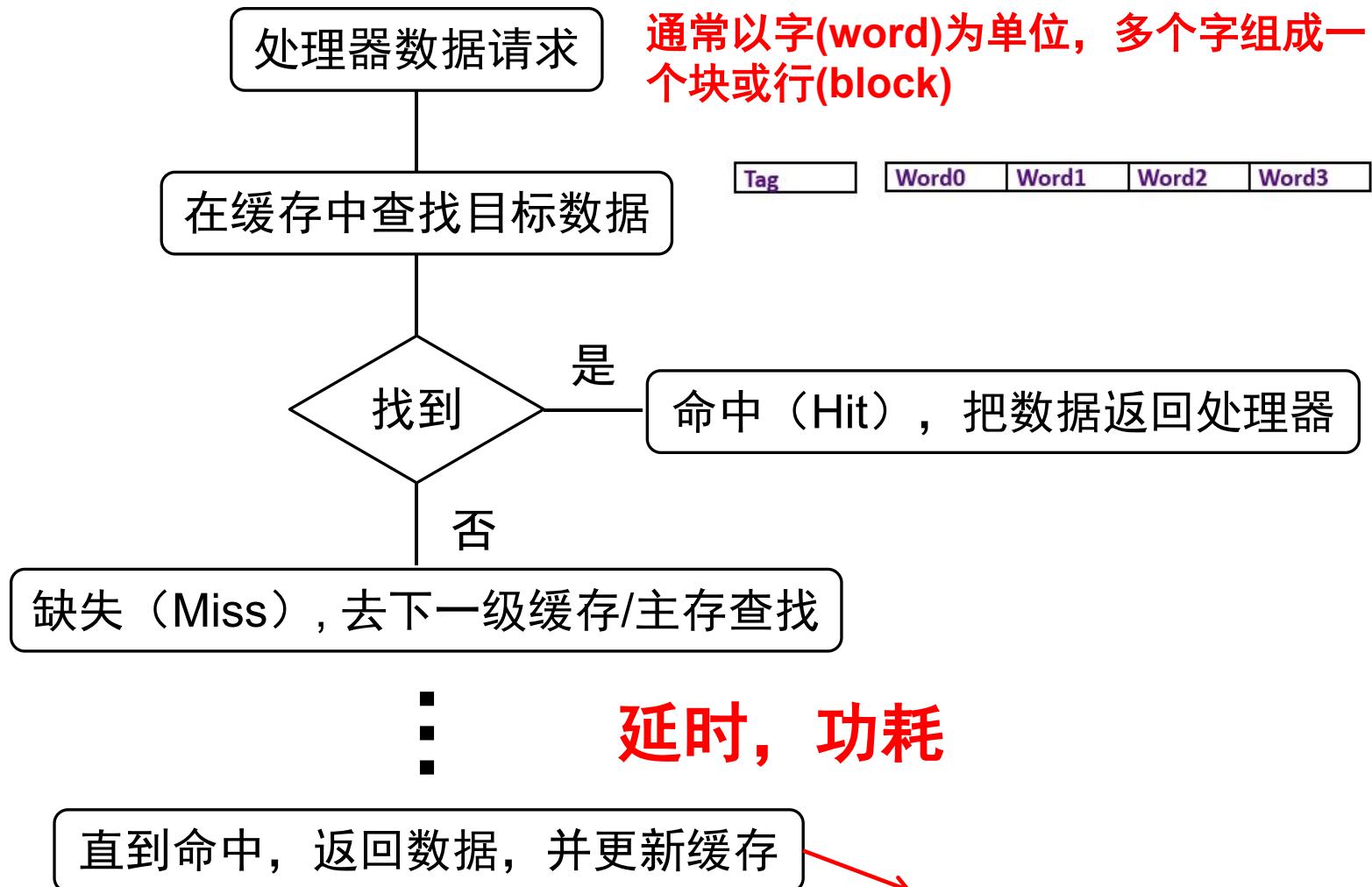


为什么价格差别这么大



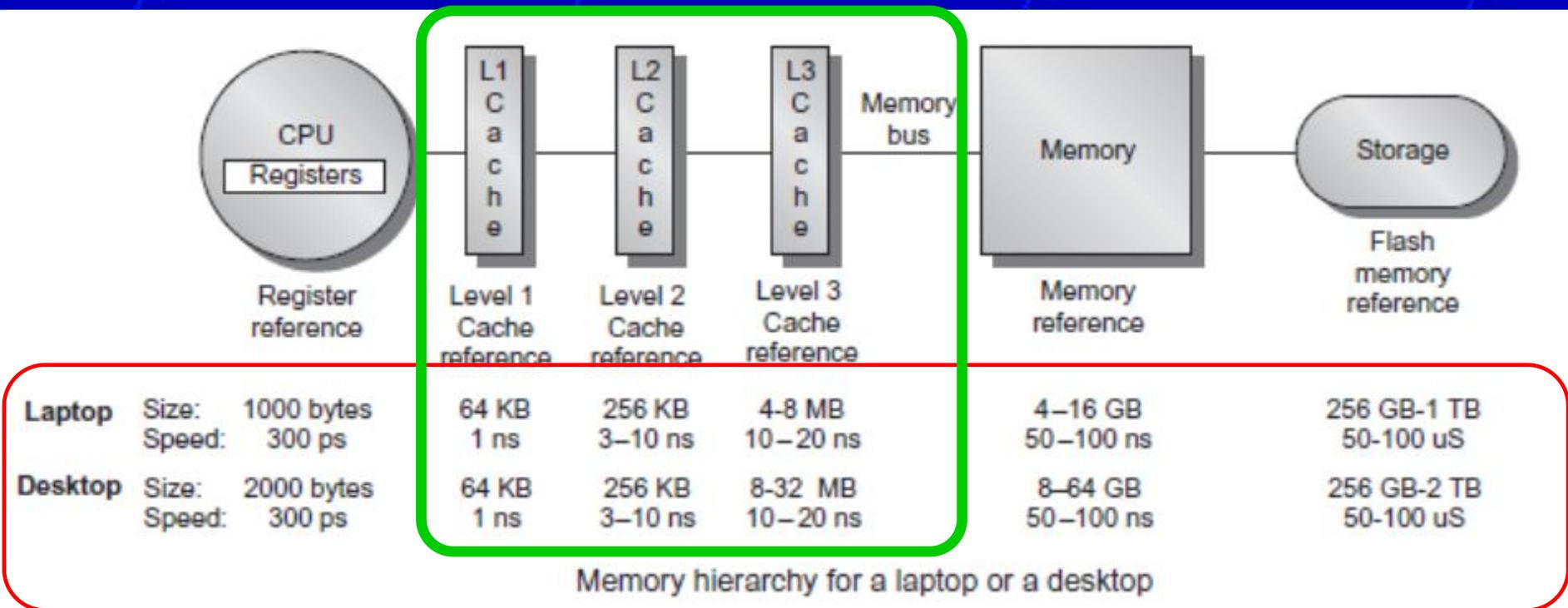


存储器数据访问基本原理





存储器层次结构优化



问题：因为价格/面积问题，**高速缓存容量有限**，每一层相比前一层容量更大，但速度更慢，怎么解决？**优化策略：局域性原理！**

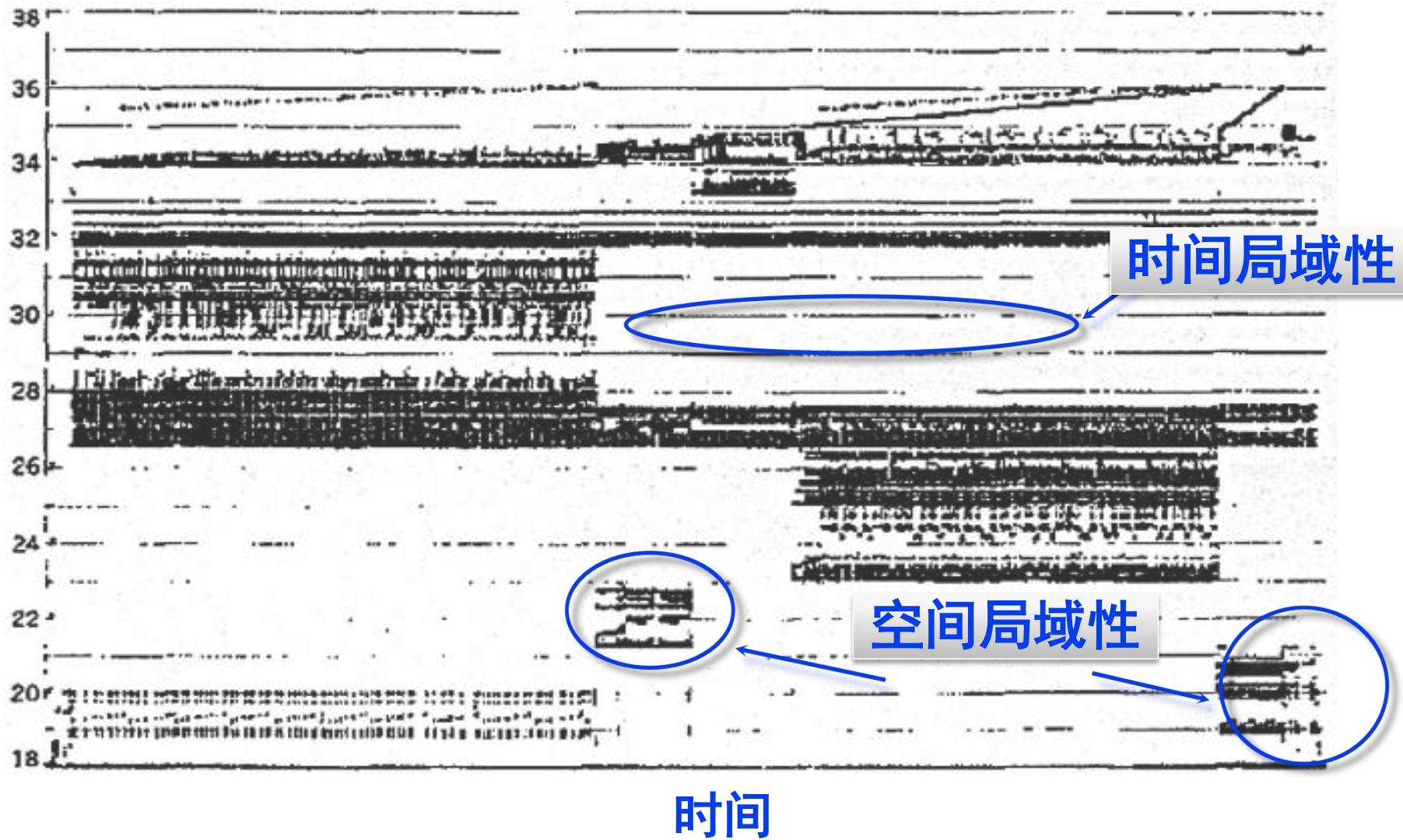
时间局域性：如果某个数据被访问，则很可能在不久的将来再次访问该数据

空间局域性：如果某个数据被访问，则很可能在不久的将来访问该数据附近数据



举例

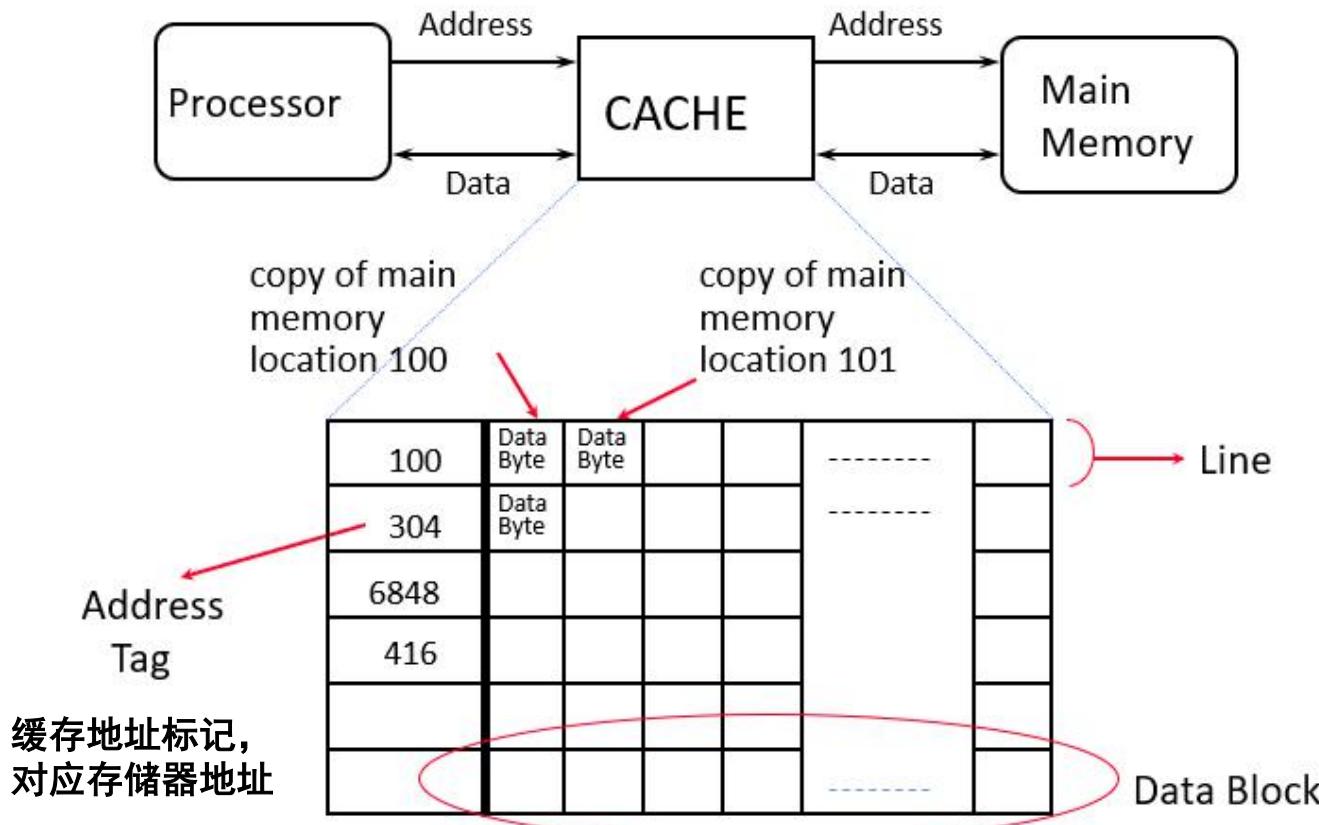
存储器位置，每个点代表一个地址





缓存结构

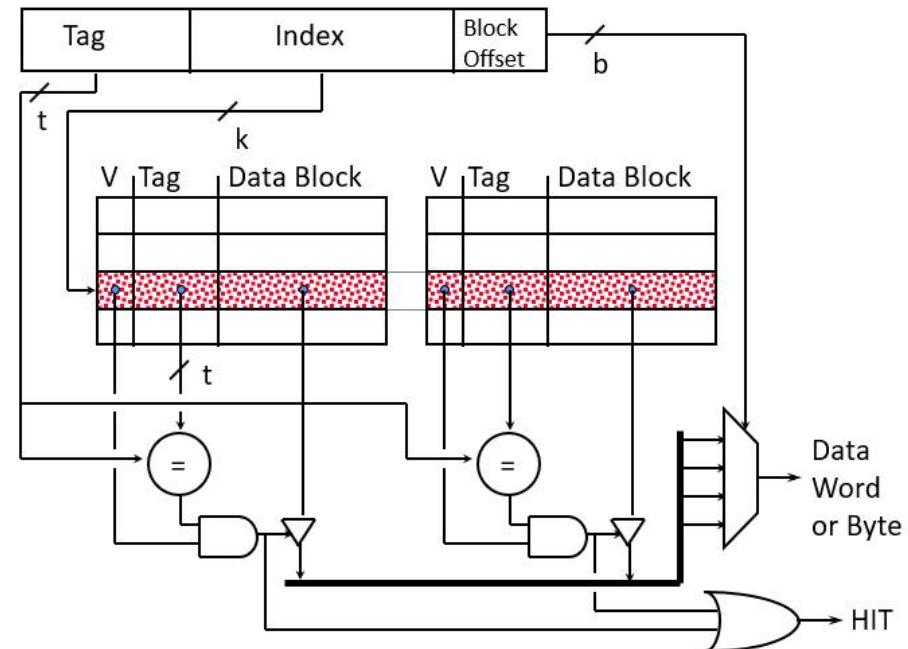
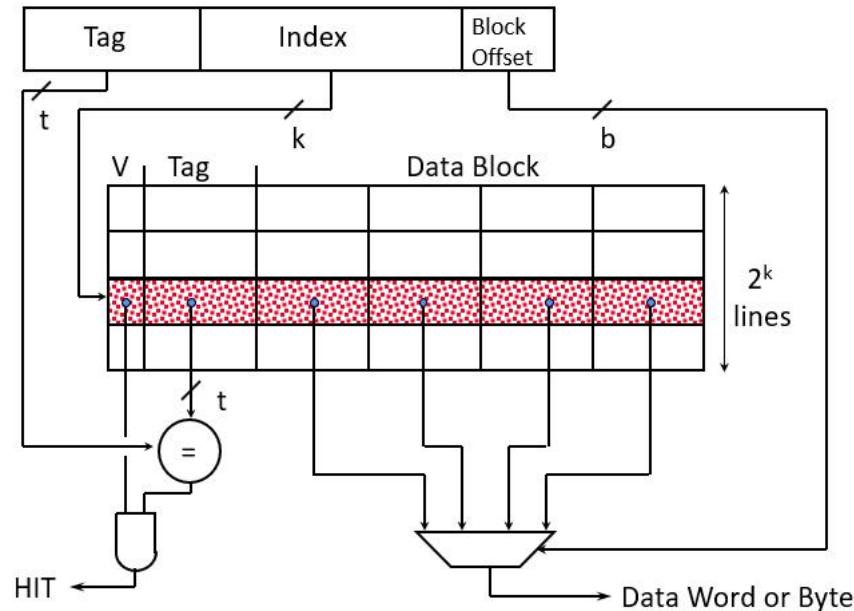
组相联：组（set）指的是缓存中的多个块(block)，一个块首先被映射到一个组块里，然后在该组块里查找；





缓存结构

如果set中有n个block，则缓存布局称为n路组相联
直接映射缓存每个set只有一个block；全相联缓存只有一个组；

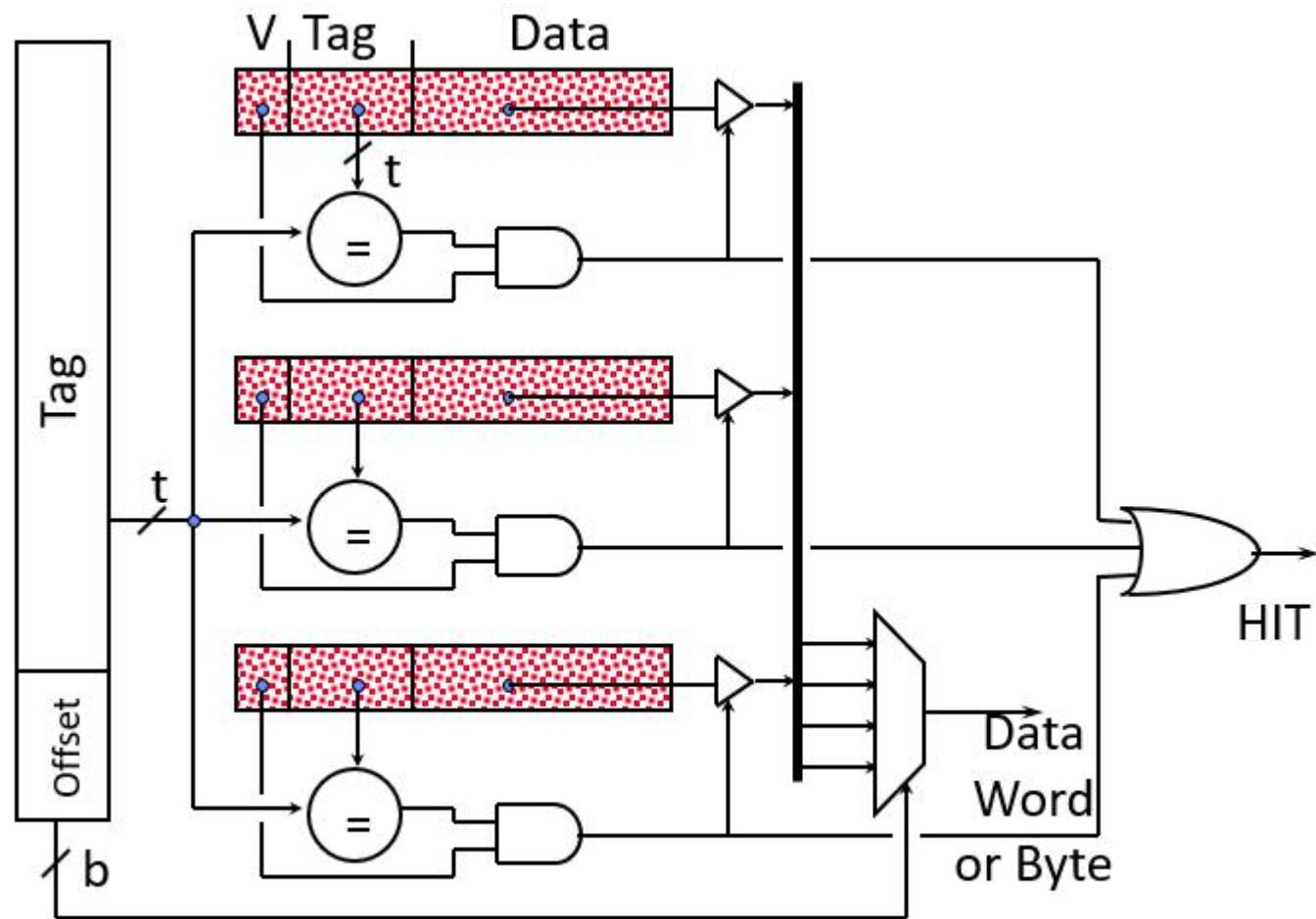


直接映射缓存

2路组相联缓存



缓存结构



全相联缓存，所有block在一个set



缓存写入/读取

缓存读取：缓存与主存有相同拷贝副本，直接读取，比较简单

缓存写入：需保证缓存与主存的副本一致性

- (1) 直接写回(write-through)策略: 同时更新缓存与主存；
- (2) 回写缓存(write-back): 只更新缓存副本，在需要替换该缓存时，再更新主存副本，缓存被擦除

两种策略都利用写缓冲区来实现异步主存更新操作，节约时间



几个关键指标

缺失率(miss rate): 未能找到预期目标的缓存访问数所占总访问次数的比率

强制缺失: 数据块的第一次访问，必然缺失；

容量缺失: 缓存不能包括程序运行期间的所有块；

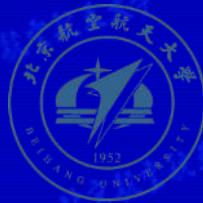
冲突缺失: 多个块映射到同一个块的组时，混杂访问发生冲突

命中时间: 在缓存中命中目标所花费的时间

缺失代价: 从主存中获取到替代块的时间

数据平均获取时间= 命中时间+ 缺失率 x 缺失代价

怎么提升性能？



缓存优化的基本原则与策略

- (1) 缩短命中时间：小而简单的第一级缓存和路预测
- (2) 增大缓存带宽：流水线缓存，多组缓存，无阻塞缓存
- (3) 降低缺失代价：关键字优化，合并写缓冲区
- (4) 降低缺失率：编译器优化
- (5) 通过并行降低缺失率：硬件预取和编译器预取

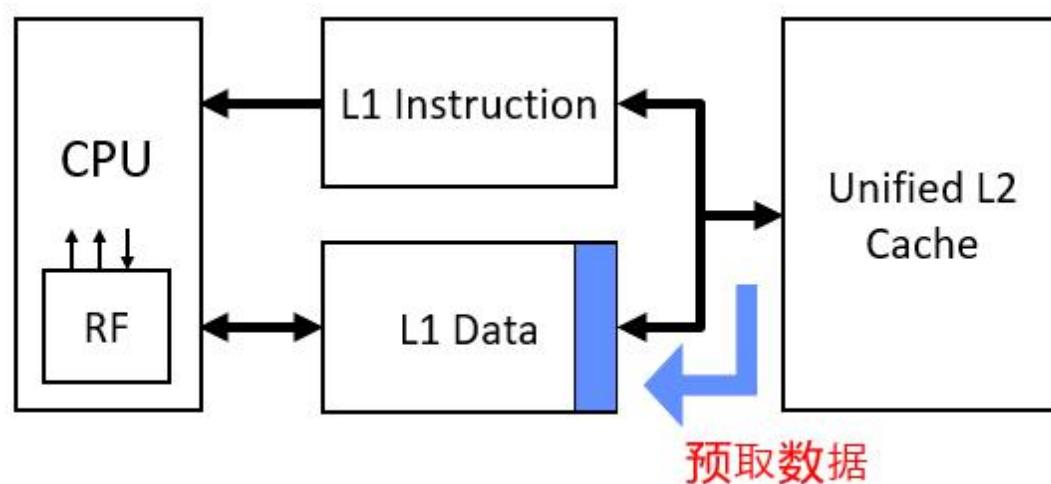
通过功耗&成本代价，带来性能的提升



举例：预取

根据程序运行规律，通过软件/硬件，从主存预先获取数据存入缓存

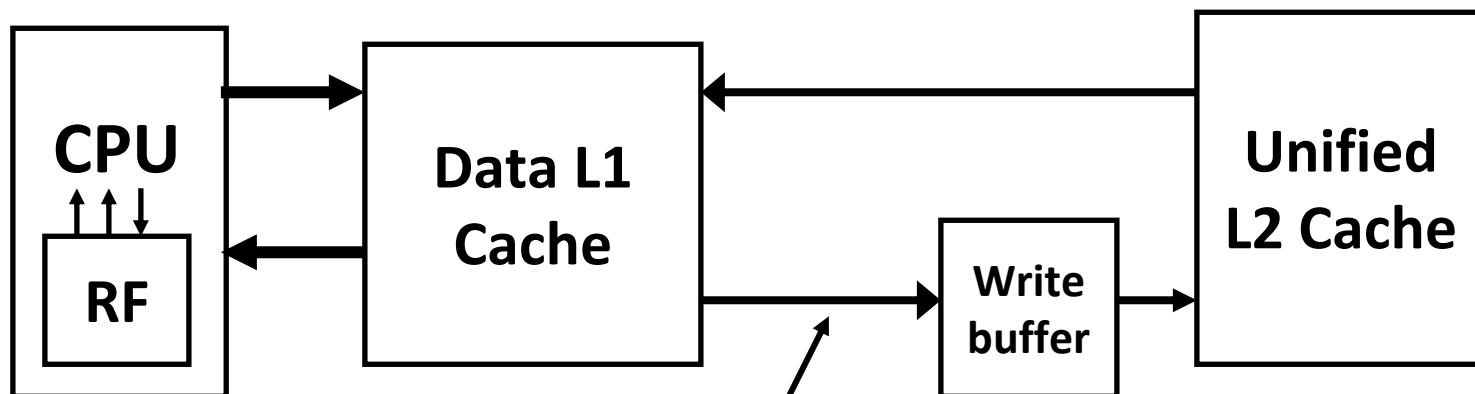
- 预测的准确性（保证命中率）；
- 预取时效性（预取过早，占缓存容量，预期过晚，无意义）
- 折中考虑缓存的容量以及数据带宽





怎么减小缓存缺失率？

- (1) 增大缓存块容量； (2) 增大总缓存容量； (3) 提高相联度
- (4) 多级缓存；
- (5) 利用写缓冲区，提高读取操作的优先级，在写入主存之前，且未清空写缓冲区之前进行读取
- (6) 缓存索引期间避免地址转换



举例：写回之前进行读取



当相联缓存满载时，替换策略？

注意：考虑时间局部性，只有发生缓存缺失miss的时候，才会替换

- (1) 随机替换
- (2) 先入先出
- (3) 最不常用的被替换（时间局部性）
- (4) 最近被更新的（下次还会被更新）

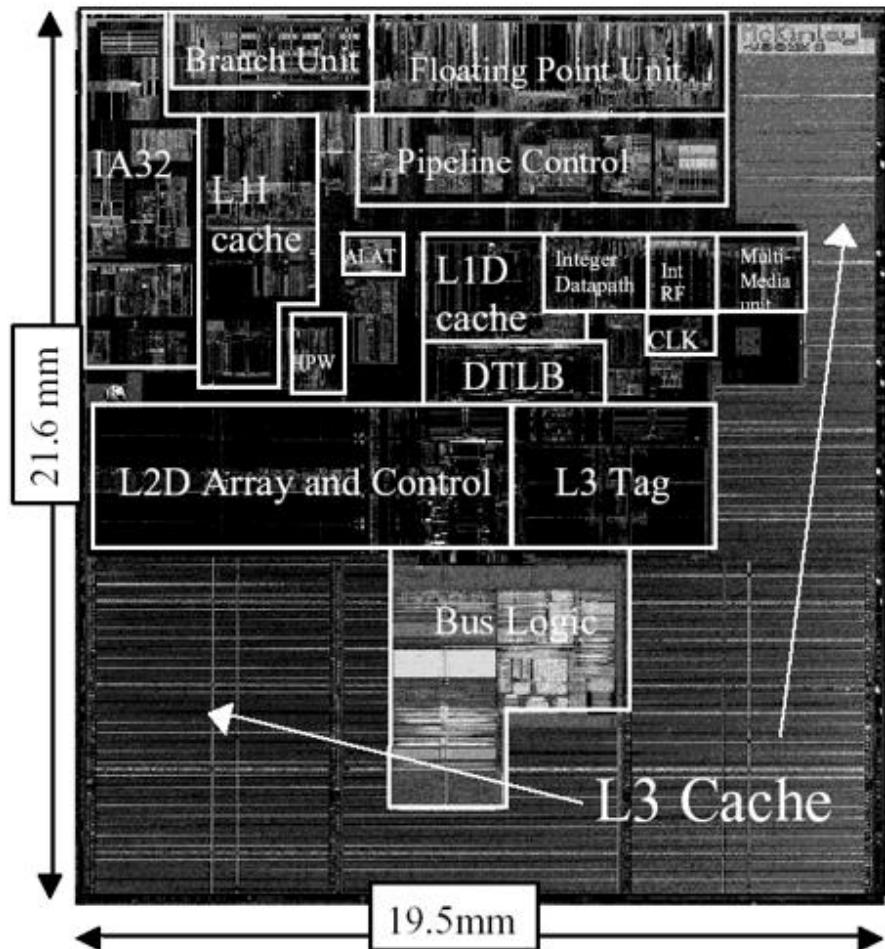
各有各的问题，需根据实际应用场景制定缓存替换策略

举例：影音播放用什么策略；图像编辑用什么策略



举例

Itanium-2 On-Chip Caches (Intel/HP, 2002)



Level 1: 16KB, 4-way s.a., 64B line, quad-port (2 load+2 store), single cycle latency

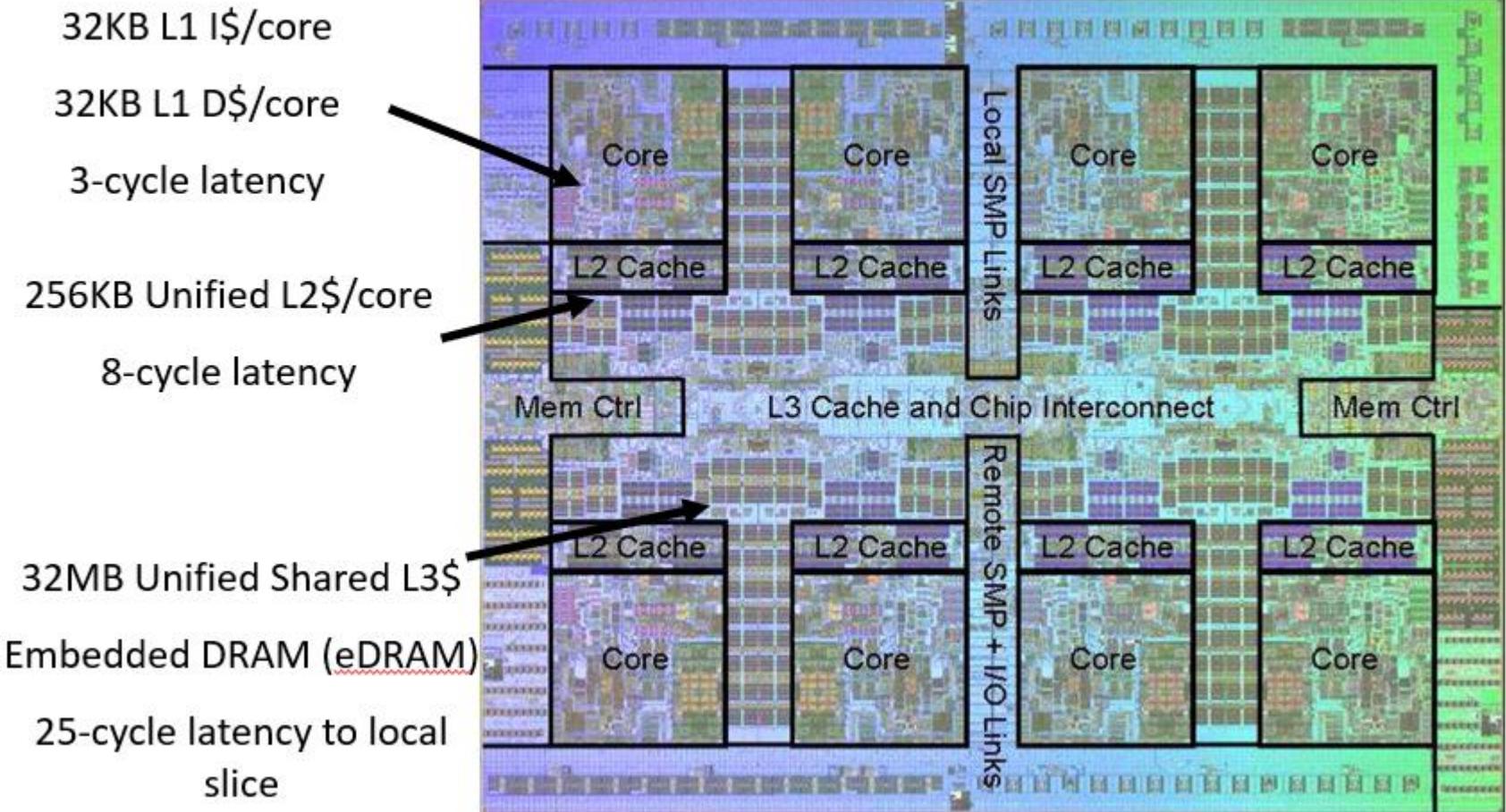
Level 2: 256KB, 4-way s.a., 128B line, quad-port (4 load or 4 store), five cycle latency

Level 3: 3MB, 12-way s.a., 128B line, single 32B port, twelve cycle latency



举例

Power 7 On-Chip Caches [IBM 2009]

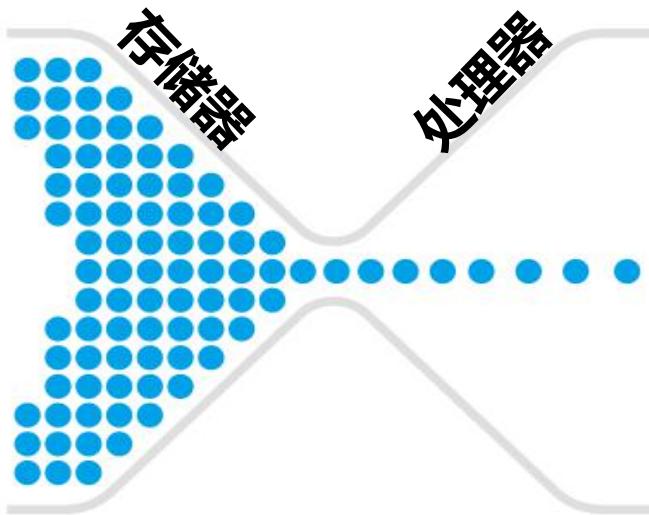


作业

7.7, 7.8, 7.10



问题？



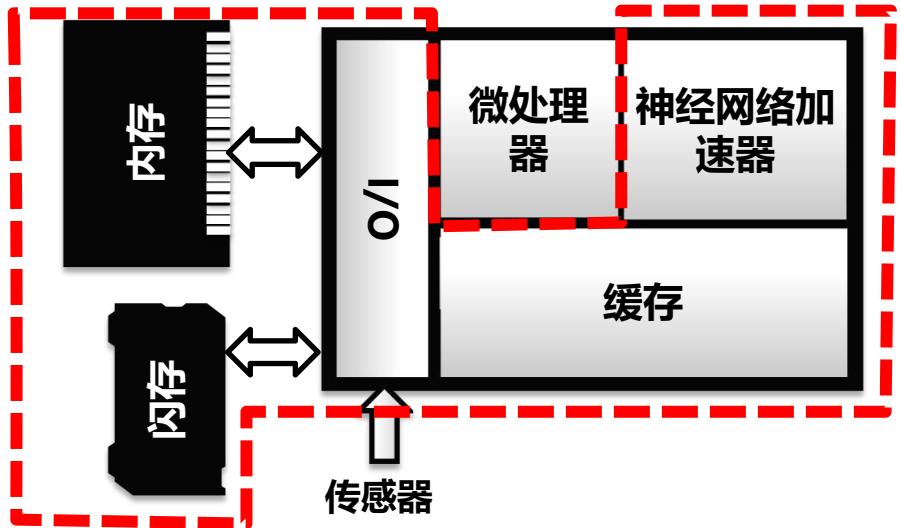
| Operation | Energy [pJ] | Relative Cost |
|--------------------|-------------|---------------|
| 32 bit int ADD | 0.1 | 1 |
| 32 bit float ADD | 0.9 | 9 |
| 32 bit int MULT | 3.1 | 31 |
| 32 bit float MULT | 3.7 | 37 |
| 32 bit 32KB SRAM | 5 | 50 |
| 32 bit DRAM | 640 | 6400 |



存算一体

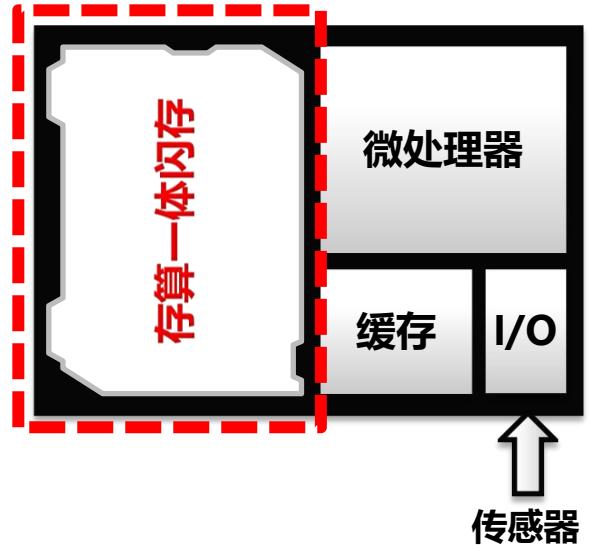
存算一体架构的目标（举例）

传统架构



VS

存算一体架构



- 1、网络权重数据直接存储在闪存中，不需要片外存储；
- 2、计算直接通过闪存完成，节省数据I/O延时与功耗；