

存储器

彭飞

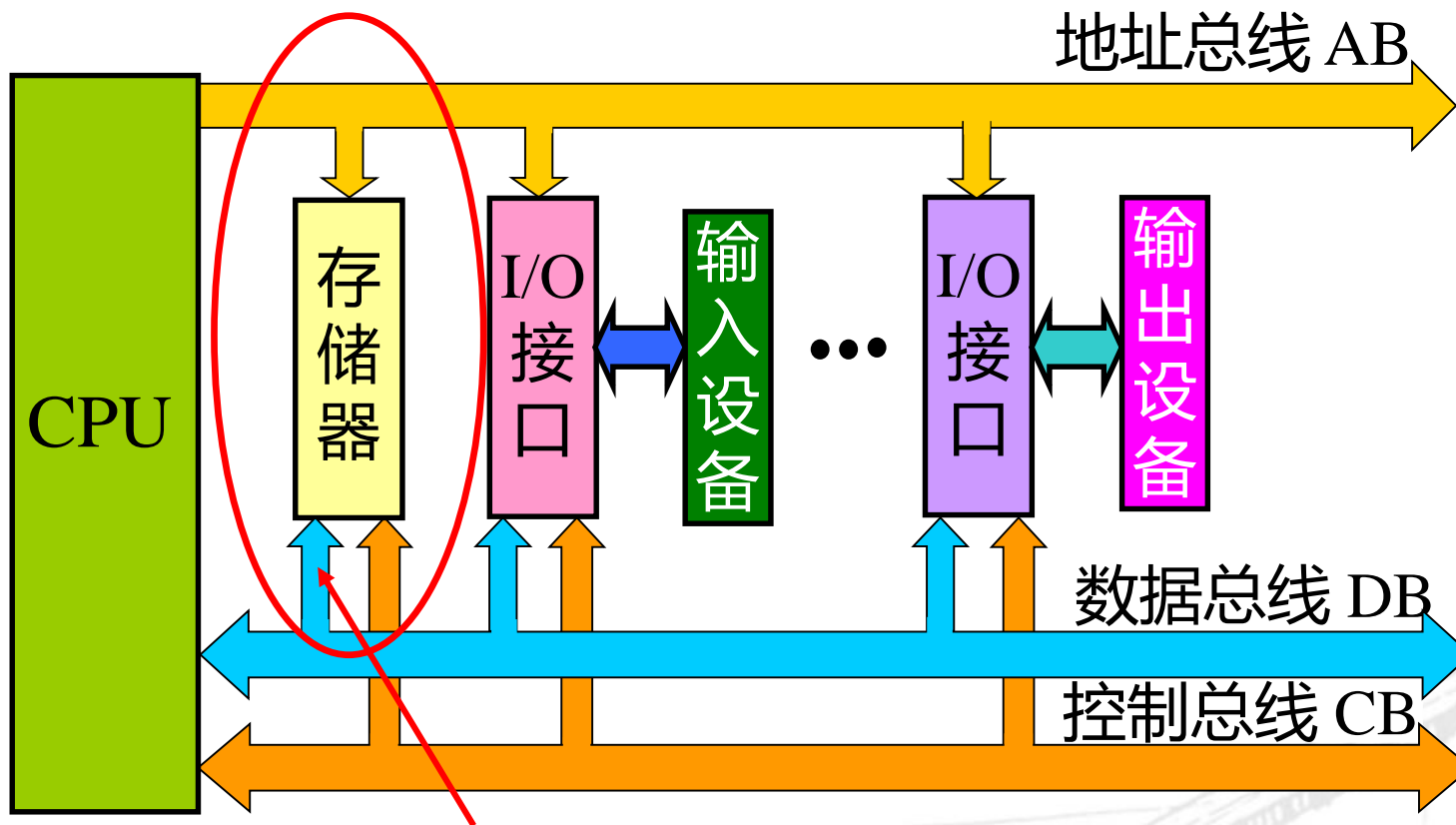
东南大学电气工程学院



- **存储器的基本概念、种类和组成**
- **SRAM的工作原理**
- **DRAM的工作原理**
- **紫外线擦除可编程ROM (EPROM) 介绍**
- **电可擦可编程只读存储器**
- **TM4C控制器中的存储器**
- **字节顺序**



- 存储器是计算机及嵌入式系统中的记忆设备。



- 半导体存储器
- 磁带光盘等

半导体
存储器

随机存取存储器
(RAM)

静态RAM (SRAM)

动态RAM (DRAM)

非易失RAM (NVRAM)

只读存储器
(ROM)

掩膜式ROM

一次性可编程ROM (PROM)

紫外线擦除可编程ROM (EPROM)

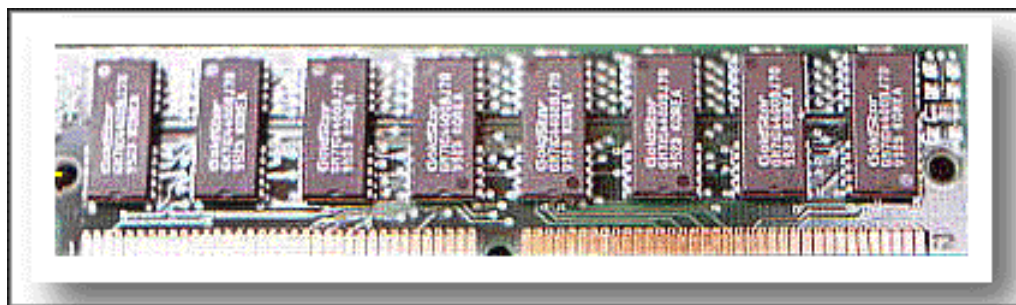
电擦除可编程ROM (EEPROM)

闪速存储器 (Flash Memory)



随机存储器RAM

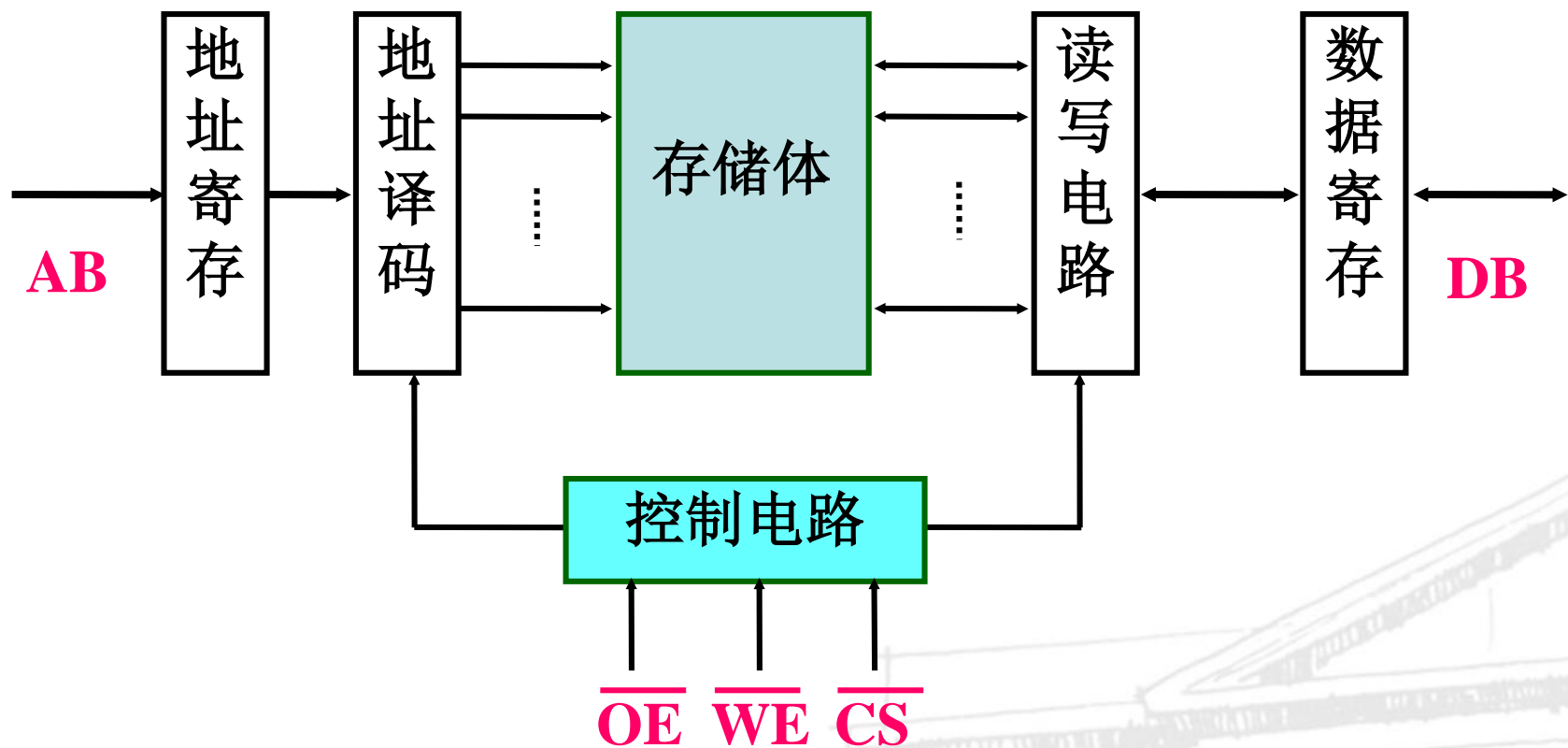
	组成单元	速度	集成度	应用
SRAM	触发器	快	低	小容量系统
DRAM	极间电容	慢	高	大容量系统
NVRAM	铁电材料，磁阻材料， 液晶	慢	低	小容量非易失



- **掩膜ROM:**
 - 信息制作在芯片中，不可更改
- **PROM:**
 - 允许一次编程，此后不可更改
- **EPROM:**
 - 用紫外光擦除，擦除后可编程；并允许用户多次擦除和编程
- **EEPROM (E²PROM) :**
 - 采用加电方法在线进行擦除和编程，也可多次擦写
- **Flash Memory (闪存) :**
 - 能够快速擦写的，但只能按块 (Block) 擦除



半导体存储器芯片的结构



一、存储体

- 存储器芯片的主要部分，用来存储信息

二、地址译码电路

- 根据输入的地址编码来选中芯片内某个特定的存储单元

三、片选和读写控制逻辑

- 选中存储芯片，控制读写操作

四、数据缓冲

- 存储器数据与总线的接口



- 一个**基本存储单元**，只能存储1位数据
- 每个**存储单元**具有一个唯一的地址，可存储1位（位片结构）或多位（字片结构）二进制数据
- 存储容量与地址、数据线个数有关：

芯片的存储容量

$$= 2^M \times N \text{ bits}$$

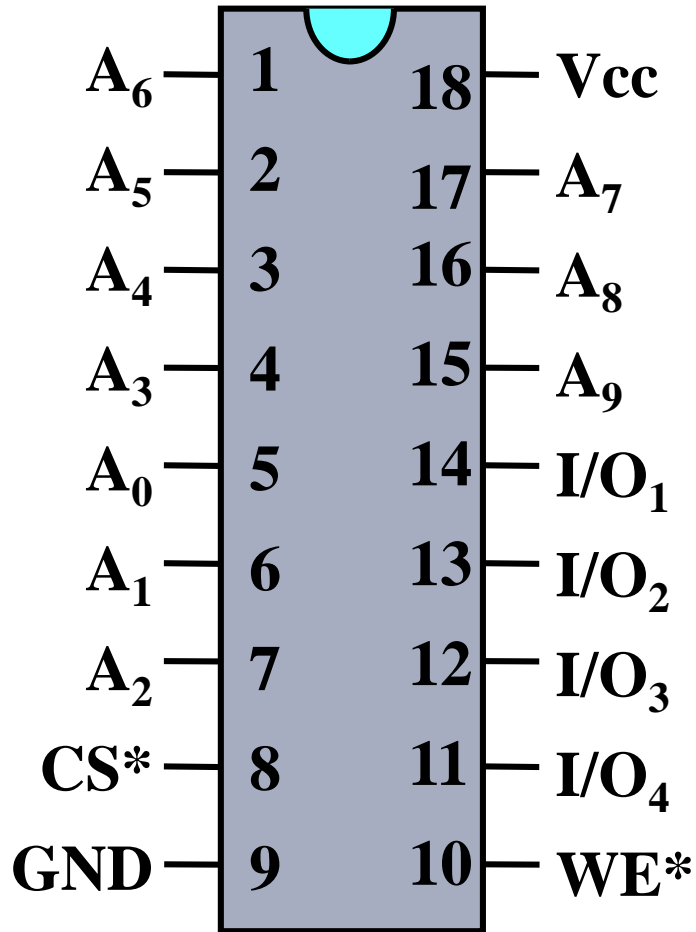
$$= \text{存储单元数} \times \text{存储单元的位数}$$

M：芯片的**地址线根数**

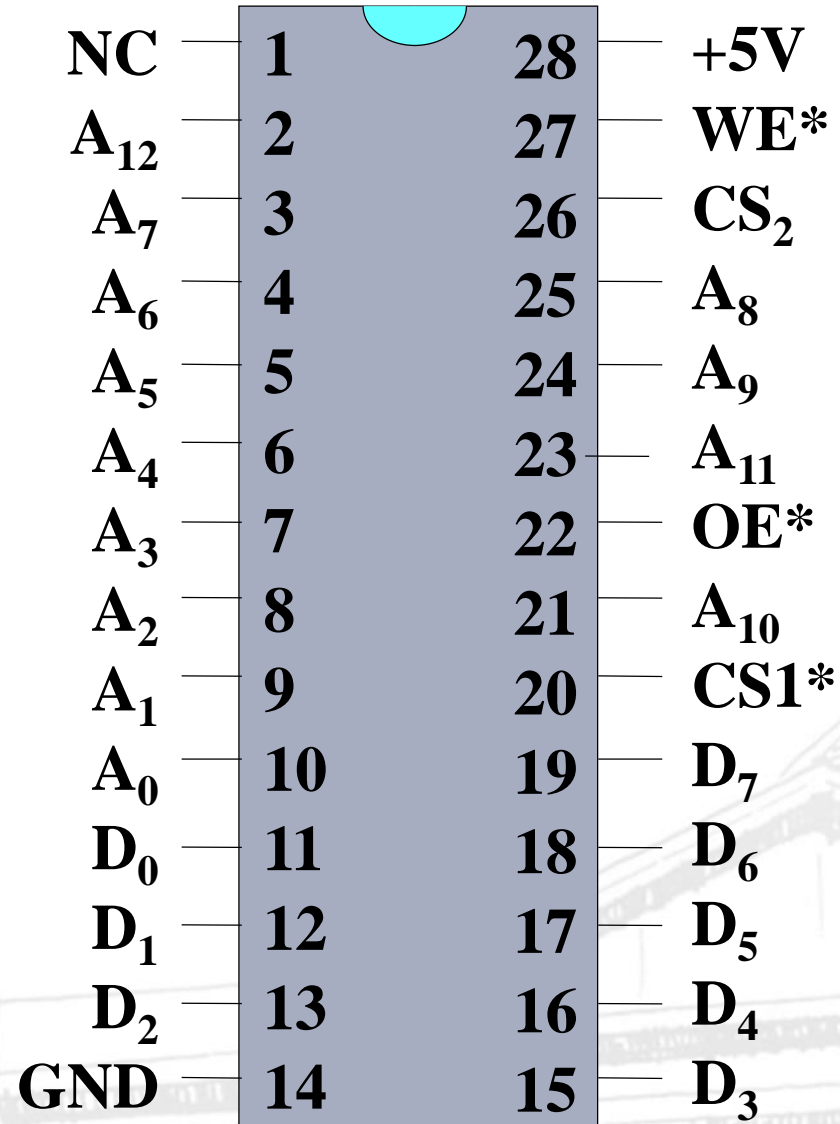
N：芯片的**数据线根数**



静态RAM



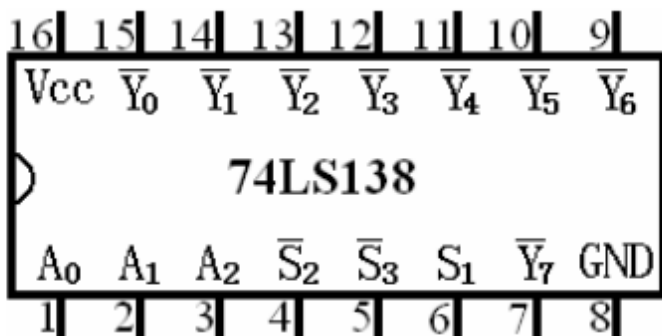
SRAM芯片2114



SRAM芯片6264

• 译码电路：

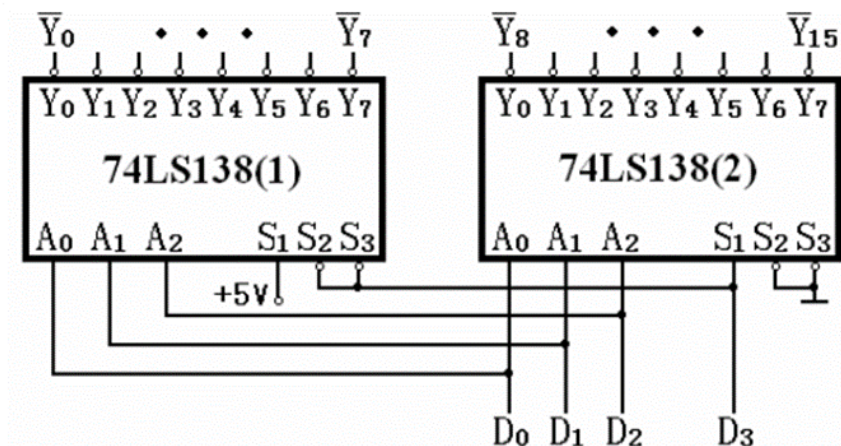
- 根据地址，选中其唯一对应的存储单元，如典型的74LS138译码电路



3线-8线译码器（74LS138）功能表

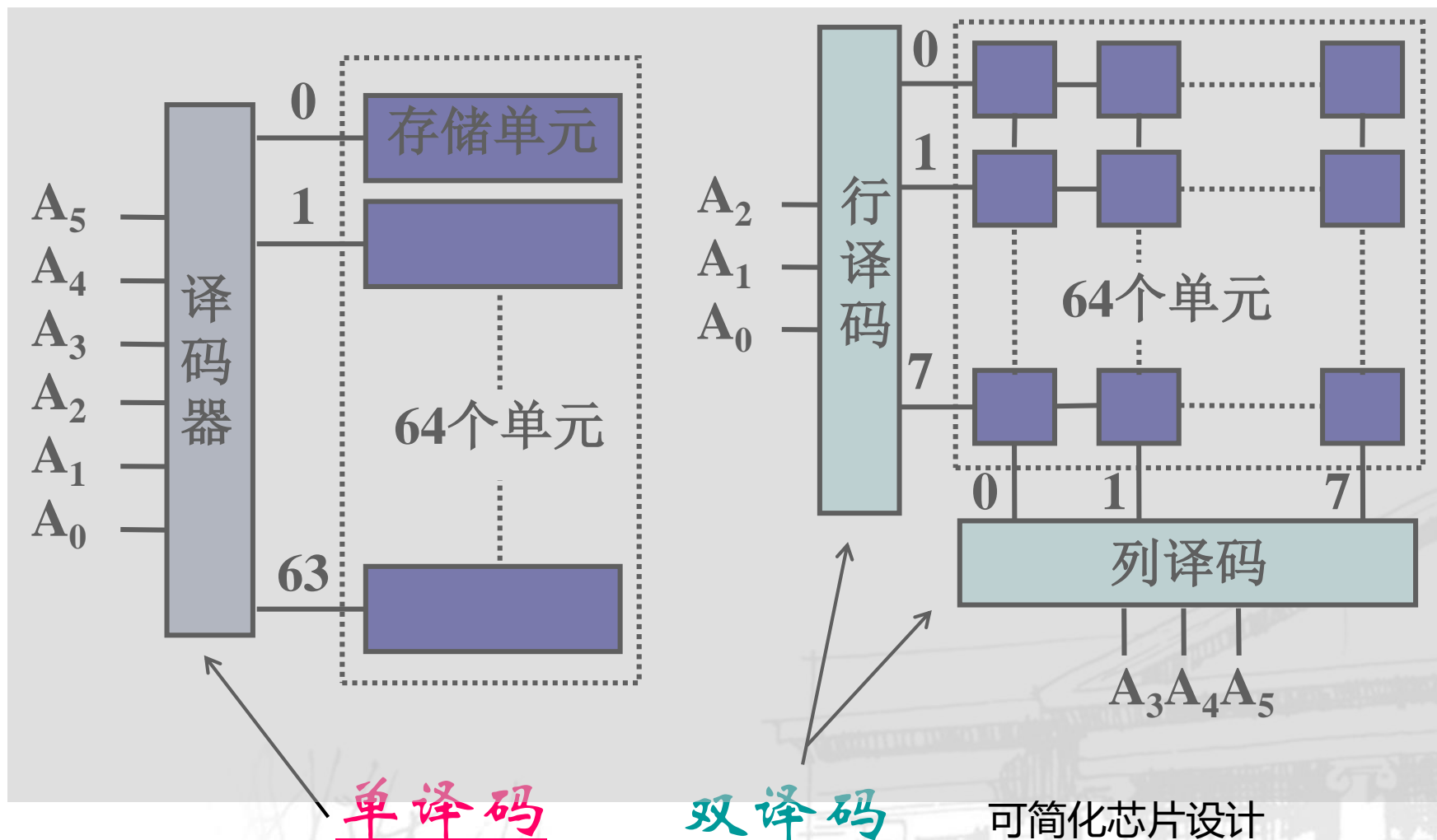
输入				输出								
S ₁	$\overline{S}_2+\overline{S}_3$	A ₂	A ₁	A ₀	\overline{Y}_0	\overline{Y}_1	\overline{Y}_2	\overline{Y}_3	\overline{Y}_4	\overline{Y}_5	\overline{Y}_6	\overline{Y}_7
0	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

• 级联使用:



D ₃ D ₂ D ₁ D ₀	\bar{Y}_0	\bar{Y}_1	\bar{Y}_2	\bar{Y}_3	\bar{Y}_4	\bar{Y}_5	\bar{Y}_6	\bar{Y}_7	\bar{Y}_8	\bar{Y}_9	\bar{Y}_{10}	\bar{Y}_{11}	\bar{Y}_{12}	\bar{Y}_{13}	\bar{Y}_{14}	\bar{Y}_{15}
0 0 0 0																
0 0 0 1																
0 0 1 0																
0 0 1 1																
0 1 0 0																
0 1 0 1																
0 1 1 0																
0 1 1 1																
1 0 0 0																
1 0 0 1																
1 0 1 0																
1 0 1 1																
1 1 0 0																
1 1 0 1																
1 1 1 0																
1 1 1 1																

• 存储器内部的译码电路



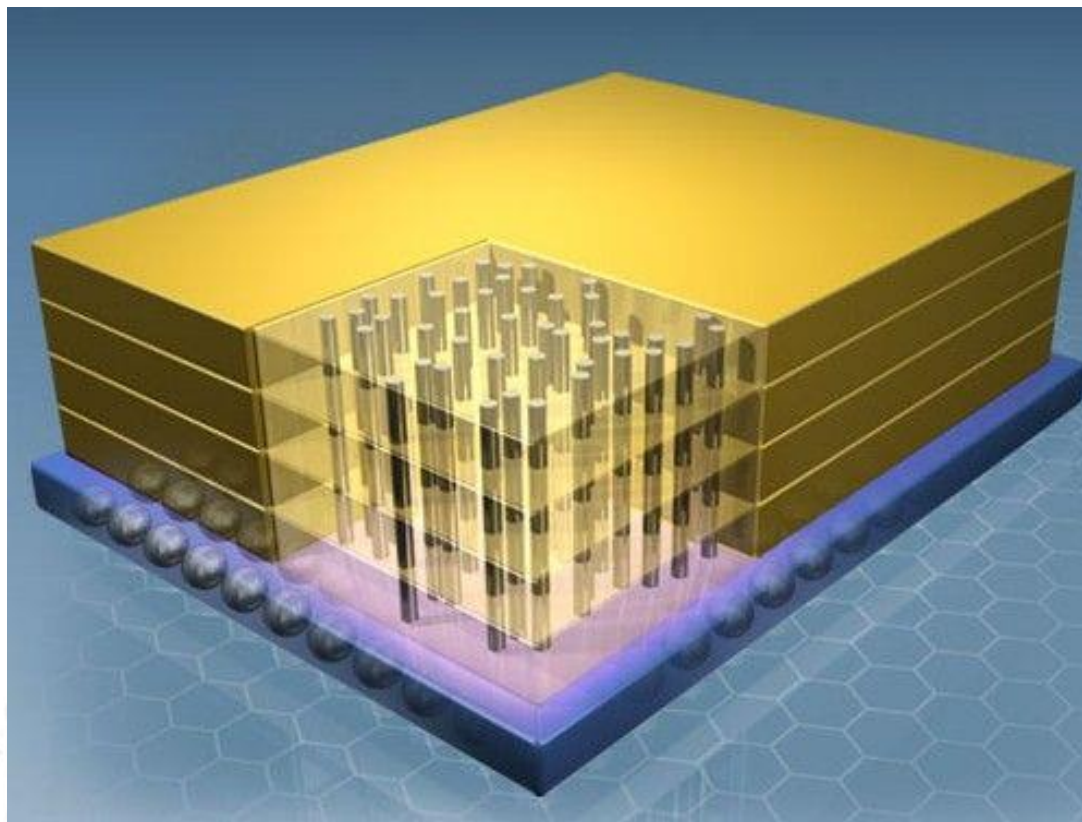
可简化芯片设计
主要采用的译码结构

南京 四牌楼2号 <http://ee.seu.edu.cn>



3D存储技术：

普通的存储芯片多为平面结构，数据只能前后左右移动，而3D存储芯片可实现数据在三维空间中的存储和传递，将大幅提高存储设备的存储能力。



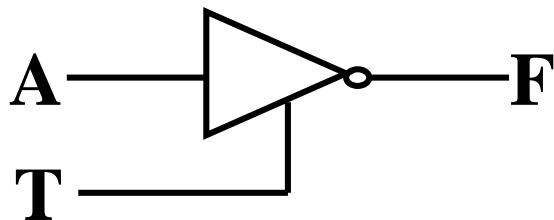
- **片选端CS*或CE***
 - 有效时，可以对该芯片进行读写操作
- **输出OE***
 - 控制读操作。有效时，芯片内数据输出
 - 该控制端对应系统的读控制线
- **写WE***
 - 控制写操作。有效时，数据进入芯片中
 - 该控制端对应系统的写控制线



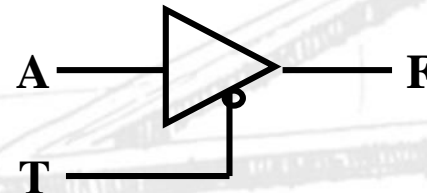
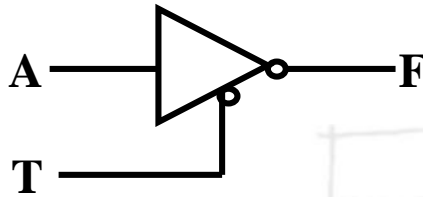
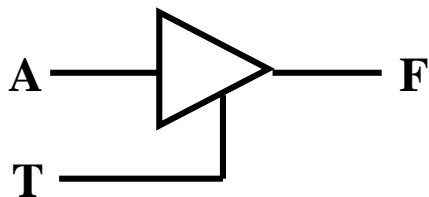
• 数据缓冲:

- 存储器数据与总线的接口，本存储器工作时，既要**输入**，又要**输出**。本存储器不工作时，要呈现**高阻态**，不影响其他存储器工作。

• 三态门：功率放大、导通开关

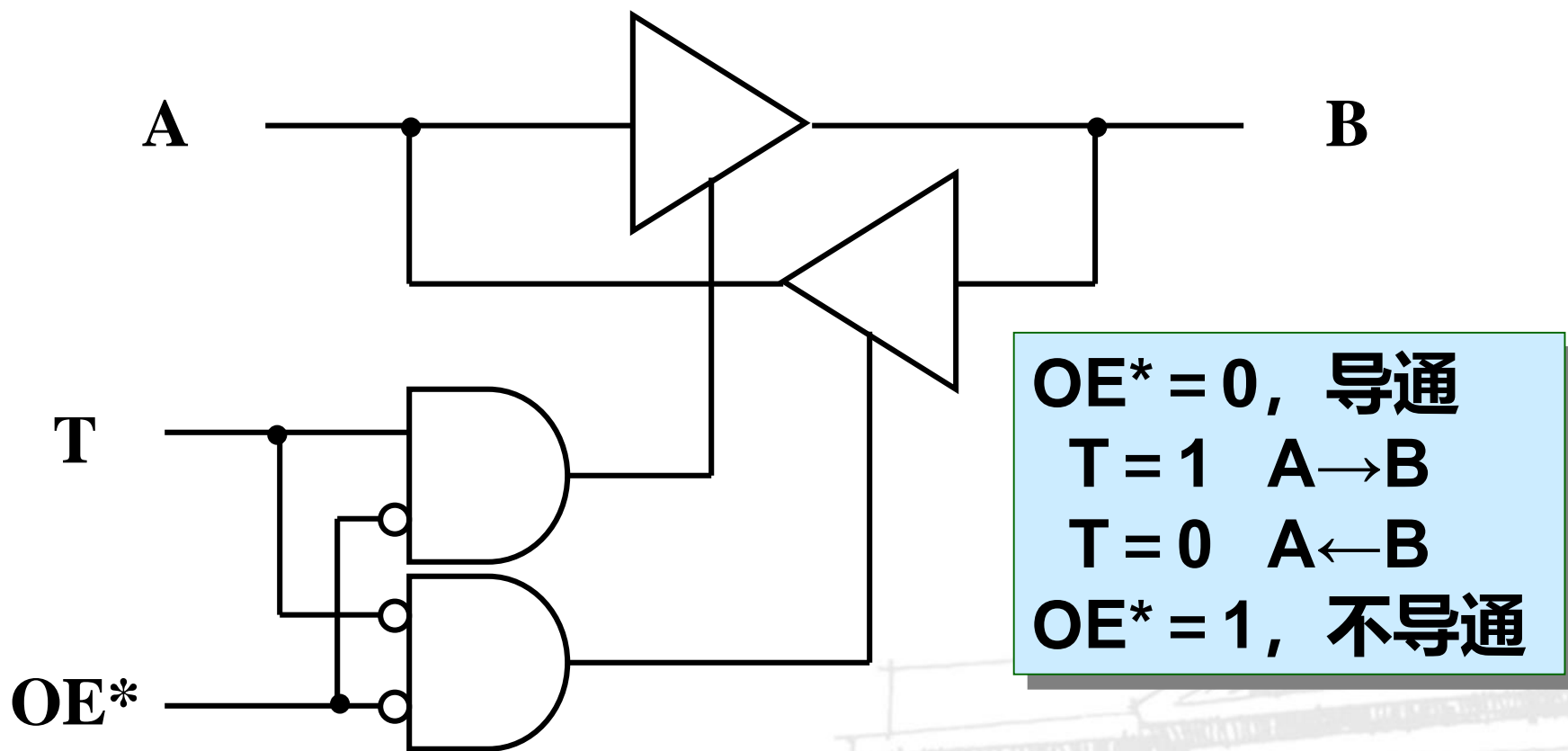


T为低平时：
输出为高阻抗（三态）
T为高电平时：
输出为输入的反相



表示反相或低电平有效

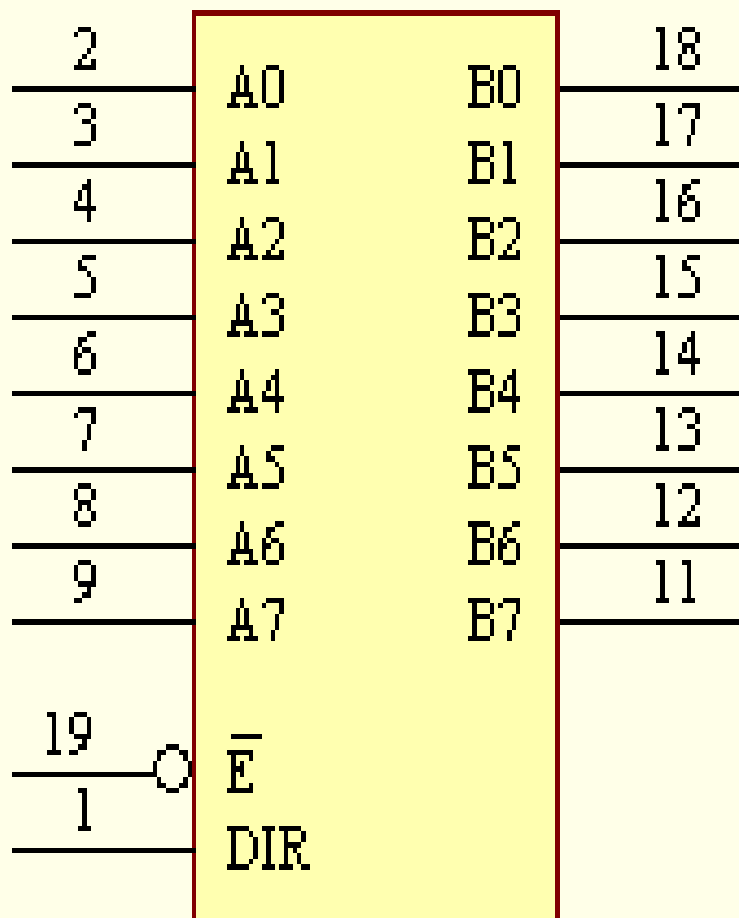
• 双向缓冲



具有双向导通和三态的特性



74LS245



74LS245

8位双向缓冲器

- 控制端连接在一起，低电平有效
- 可以双向导通
- 输出与输入同相

$E^* = 0$, 导通

$DIR = 1$ $A \rightarrow B$

$DIR = 0$ $A \leftarrow B$

$E^* = 1$, 不导通

- SRAM的**基本存储单元**是**触发器**电路
- 每个**基本存储单元**存储二进制数一位
- 许多个基本存储单元形成行列存储矩阵
- SRAM一般采用“字结构”存储矩阵：
 - 每个**存储单元**存放多位（4、8、16等）
 - 每个存储单元具有一个地址



• SRAM六管结构的工作原理

保持 (Standby)

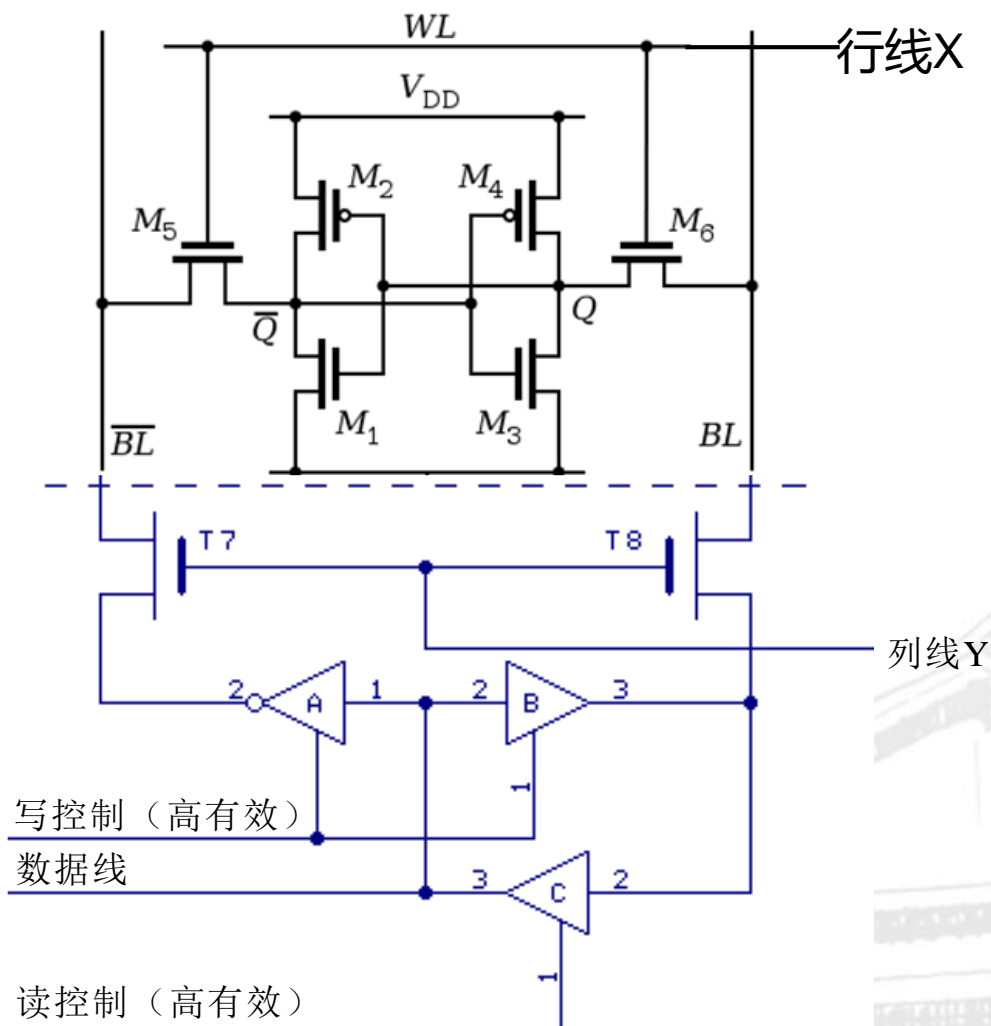
如果字线 (行线, Word Line) 没有被选为高电平, 那么作为控制用的M5与M6两个晶体管处于断路, 把基本单元与位线隔离。由M1 – M4组成的两个反相器继续保持其状态, 只要保持与高、低电平的连接。数据线呈**高阻态**。

读 (Reading)

行线和列线被选中, 读使能。写无效。M5, M6, M7, M8导通, Q中保存的值传送的数据线上。

写 (Writing)

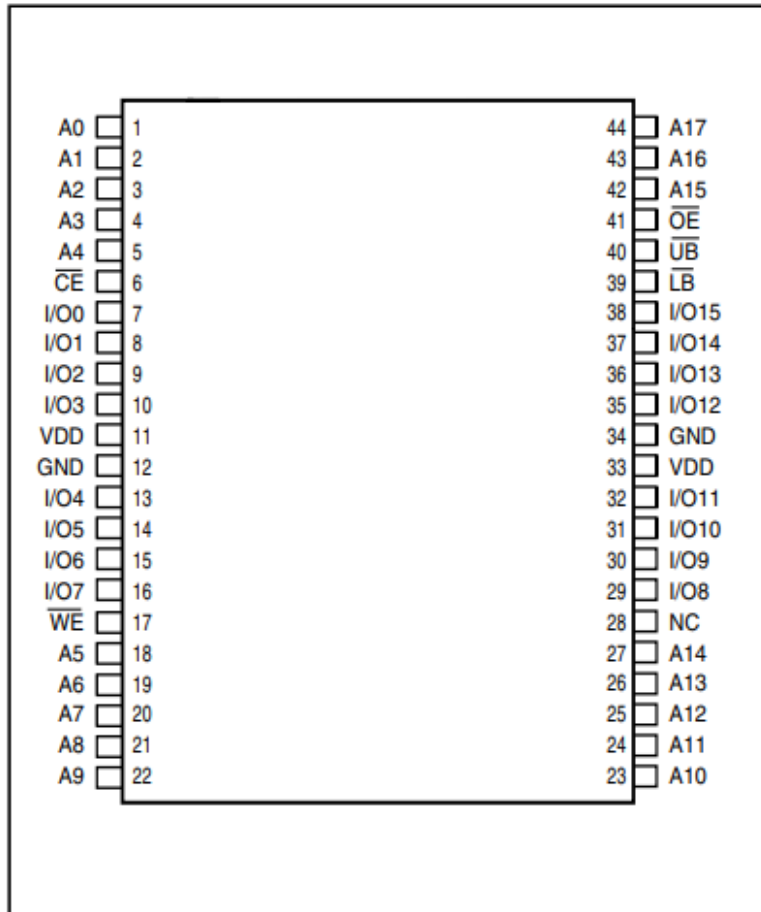
行线和列线被选中, 写使能。读无效。M5, M6, M7, M8导通, 数据线上为需要写入的数据, 且具有较强的**驱动能力**, 强制存储单元改变状态。



- **IS61LV25616AL**
- 共有 4,194,304-bit 个存储单元 262,144 × 16 bits

PIN CONFIGURATIONS

44-Pin TSOP (Type II) and SOJ



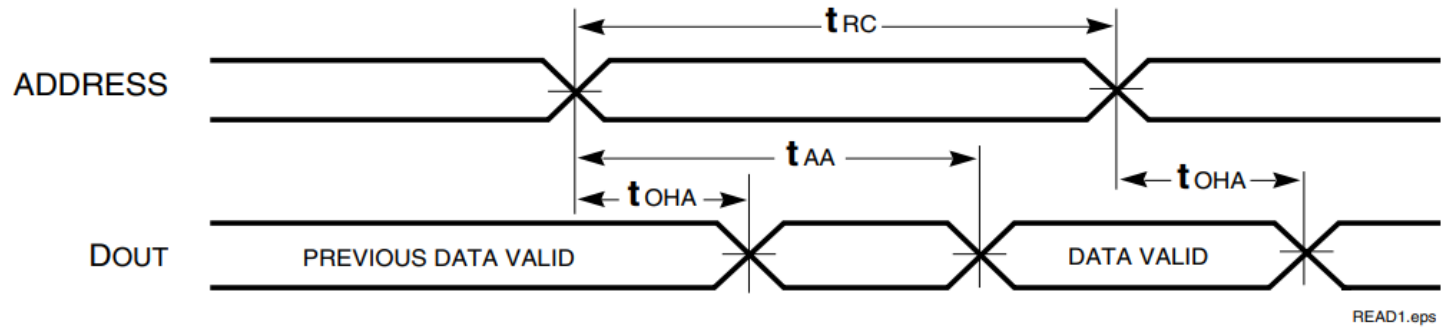
PIN DESCRIPTIONS

A0-A17	Address Inputs
I/O0-I/O15	Data Inputs/Outputs
\overline{CE}	Chip Enable Input
\overline{OE}	Output Enable Input
\overline{WE}	Write Enable Input
\overline{LB}	Lower-byte Control (I/O0-I/O7)
\overline{UB}	Upper-byte Control (I/O8-I/O15)
NC	No Connection
V _{DD}	Power
GND	Ground



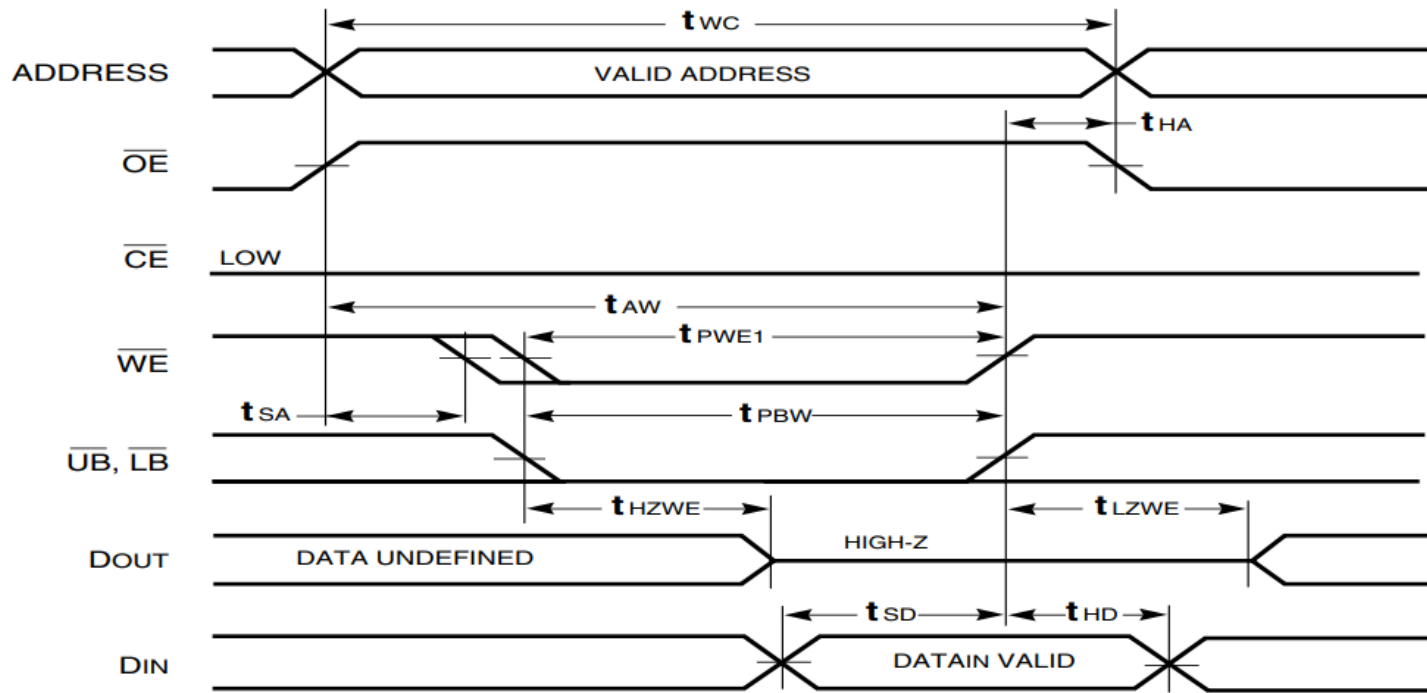
• 读周期1

READ CYCLE NO. 1^(1,2) (Address Controlled) ($\overline{CE} = \overline{OE} = V_{IL}$, \overline{UB} or $\overline{LB} = V_{IL}$)



• 写周期2

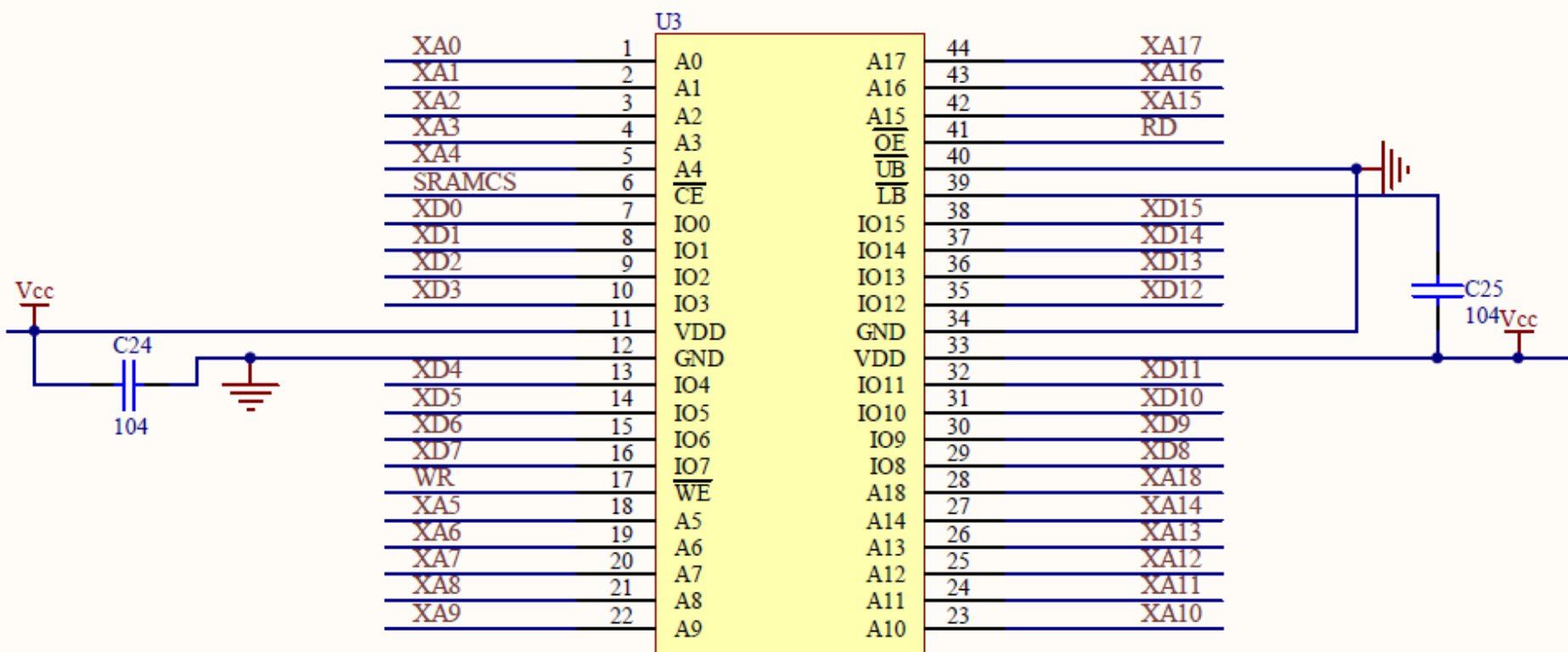
WRITE CYCLE NO. 2 ($\overline{\text{WE}}$ Controlled. $\overline{\text{OE}}$ is HIGH During Write Cycle) ^(1,2)



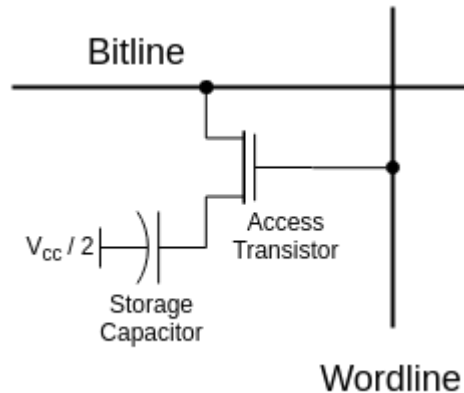
UB_CEW2R2.eps



• 与CPU的连接



- **DRAM (Dynamic Random Access Memory)**，即动态随机存储器，也就是我们常说的计算机内存，在现代计算机系统和SOC系统中有很重要的作用。



存储电容(Storage Capacitor),它通过存储在其中的电荷的多和少，或者说电容两端电压差的高和低，来表示逻辑上的 1 和 0。**存储电容**的 Common 端接在 $V_{cc}/2$ 。

- 当**存储电容**存储的信息为 1 时，另一端电压为 V_{cc} ，此时其所存储的电荷
- $Q = +V_{cc}/2 / C$
- 当**存储电容**存储的信息为 0 时，另一端电压为 0，此时其所存储的电荷
- $Q = -V_{cc}/2 / C$

访问晶体管(Access Transistor)，它的导通和截止，决定了允许或禁止对**存储电容**所存储的信息的读取和改写。

字线(Wordline)，它决定了**访问晶体管**的导通或者截止。

位线(Bitline)，它是外界访问**存储电容**的唯一通道，当**访问晶体管**导通后，外界可以通过**位线**对**存储电容**进行读取或者写入操作。

• 数据读写原理

- 读数据时，**字线**设为逻辑高电平，打开**访问晶体管**，然后读取**位线**上的状态
- 写数据时，先把要写入的电平状态设定到**位线**上，然后打开**访问晶体管**，通过**位线**改变**存储电容**内部的状态。

• 问题：

• 外界的逻辑电平与 Storage Capacitor 的电平不匹配

- 由于**位线**的电容值比**存储电容**要大的多（通常为 10 倍以上），当**访问晶体管**导通后，如果**存储电容**存储的信息为 1 时，**位线**电压变化非常小。外界电路无法直接通过**位线**来读取**存储电容**所存储的信息。

• 进行一次读取操作后，存储电容存储的电荷会变化

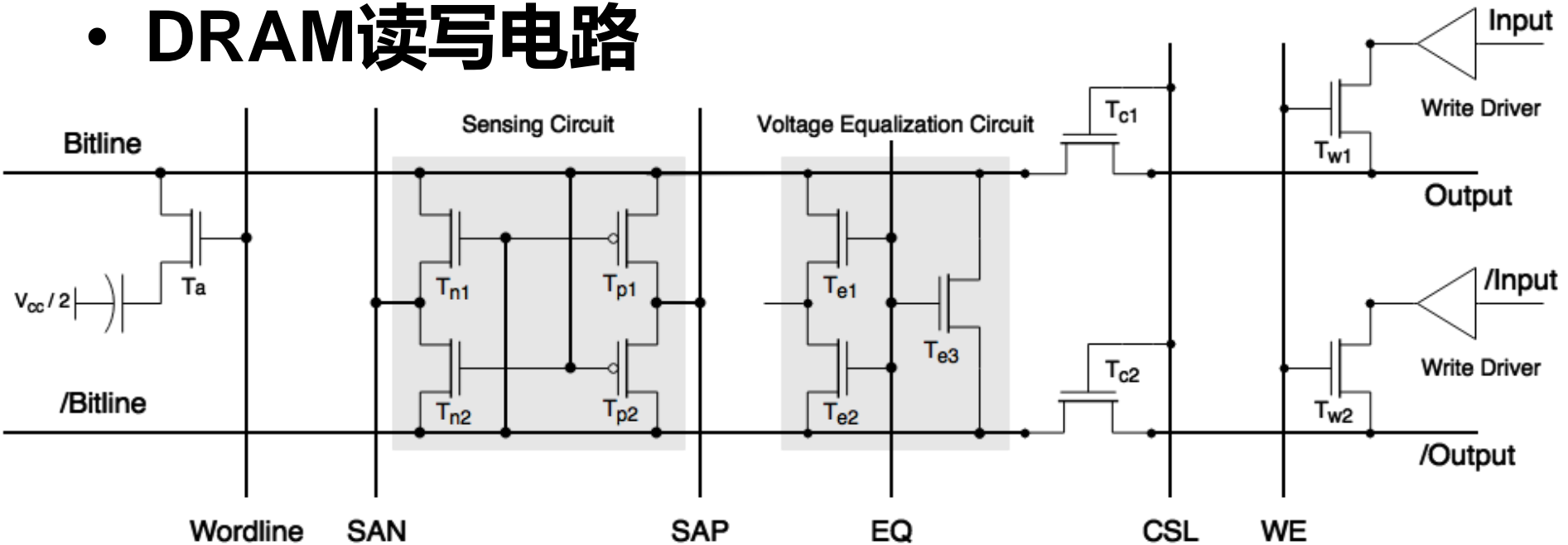
- 在进行一次读取操作的过程中，**访问晶体管**导通后，由于**位线存储电容**端的电压不一致，会导致**存储电容**中存储的电荷量被改变。最终可能会导致在下一次读取操作过程中，无法正确的判断**存储电容**内存储的信息。

• 由于存储电容的物理特性，即使不进行读写操作，其所存储的电荷都会慢慢变少

- 这个特性要求 DRAM 在没有读写操作时，也要主动对**存储电容**进行电荷恢复的操作。



• DRAM读写电路

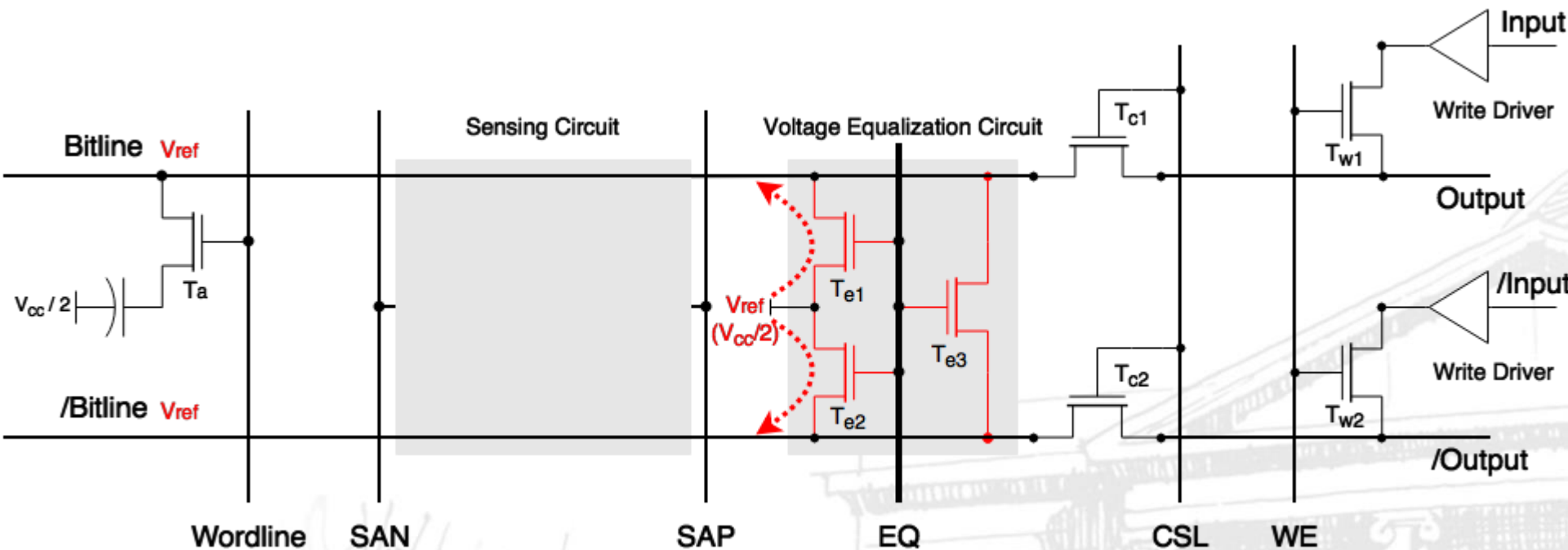


差分读取放大器（Differential Sense Amplifier）包含读出电路（Sensing Circuit）和电压均衡电路（Voltage Equalization Circuit）两个主要部分。

它主要的功能就是

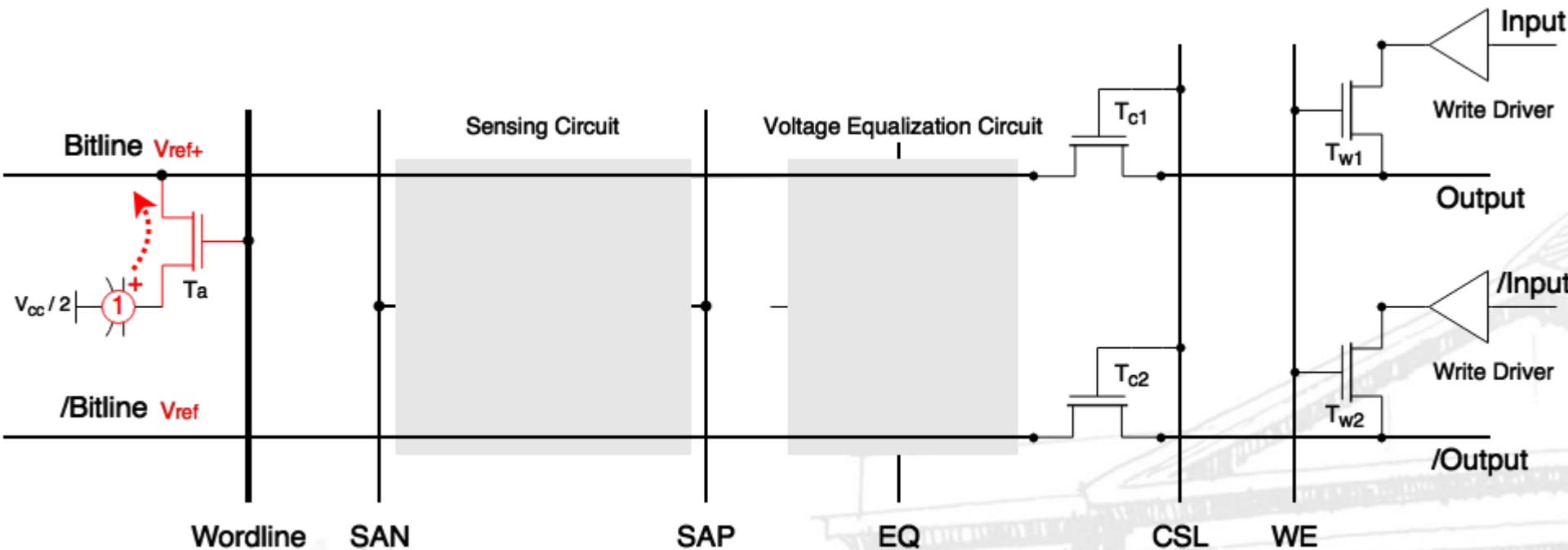
1. 将**存储电容**存储的信息转换为逻辑 1 或者 0 所对应的电压，并且呈现到**位线**上。
2. 同时，在完成一次读取操作后，通过**位线**将**存储电容**中的电荷恢复到读取之前的状态。

- 一个完整的读操作(Read Operation) 包含了, **Precharge**、**Access**、**Sense**、**Restore** 四个阶段。
- **Precharge:**
 - 在这个阶段, 首先会通过控制 EQ 信号, 让 T_{e1} 、 T_{e2} 、 T_{e3} 晶体管处于导通状态, 将 Bitline 和 /Bitline 线上的电压稳定在 V_{ref} 上, $V_{ref} = V_{cc}/2$ 。然后进入到下一个阶段。



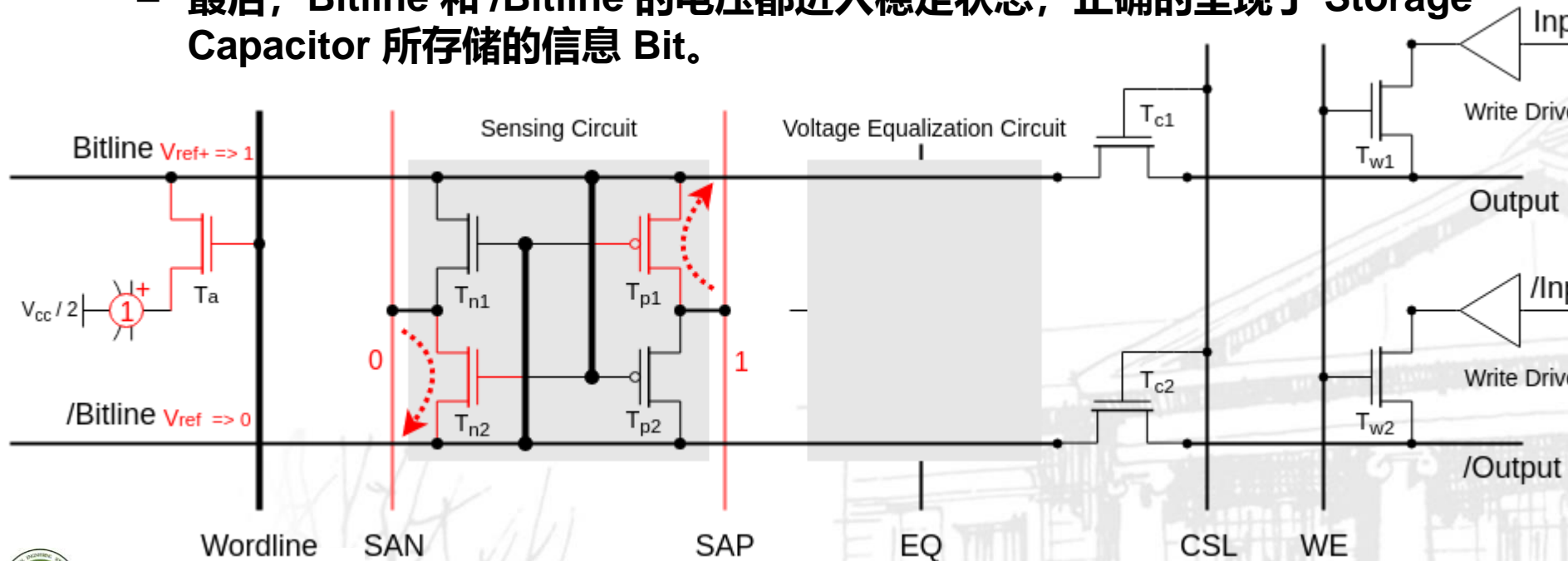
- Access:

- 经过 Precharge 阶段，Bitline 和 /Bitline 线上的电压已经稳定在 V_{ref} 上了，此时，通过控制 Wordline 信号，将 Ta 晶体管导通。Storage Capacitor 中存储正电荷会流向 Bitline，继而将 Bitline 的电压拉升到 $V_{ref}+$ 。然后进入到下一个阶段。



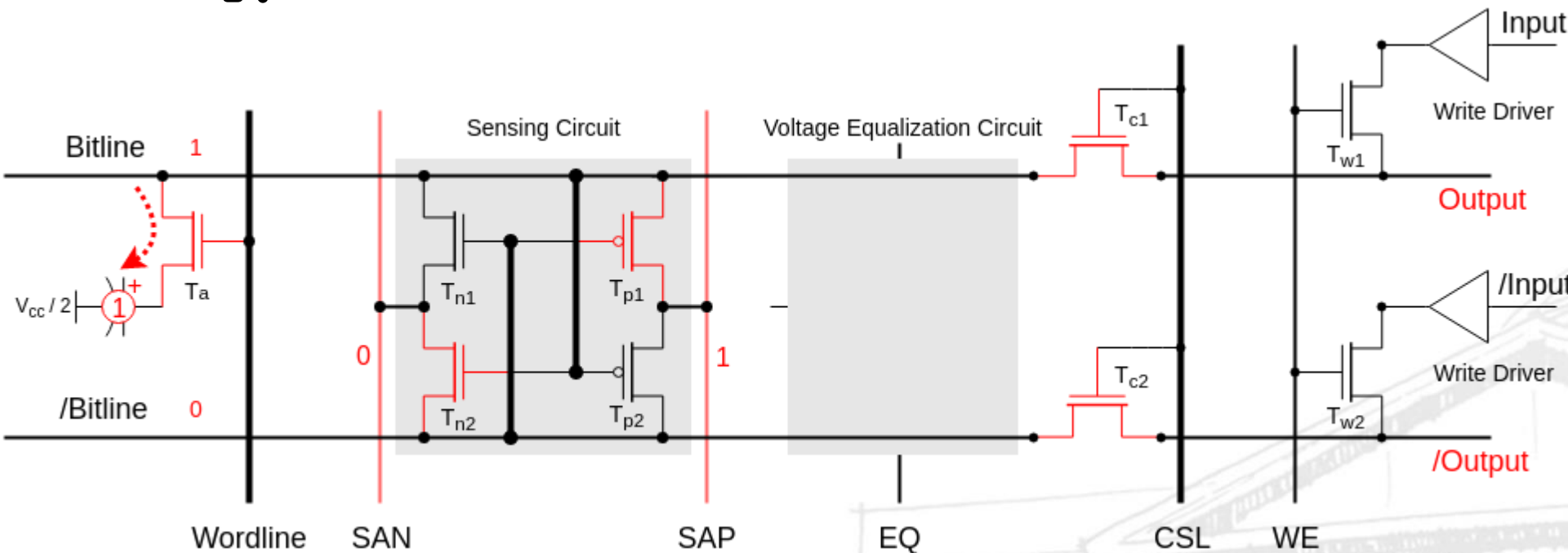
Sense

- 由于在 Access 阶段，Bitline 的电压被拉升到 V_{ref+} ， T_{n2} 会比 T_{n1} 更具导通性， T_{p1} 则会比 T_{p2} 更具导通性。
- 此时，SAN (Sense-Amplifier N-Fet Control) 会被设定为逻辑 0 的电压，SAP (Sense-Amplifier P-Fet Control) 则会被设定为逻辑 1 的电压，即 V_{cc} 。由于 T_{n2} 会比 T_{n1} 更具导通性，/Bitline 上的电压会更快被 SAN 拉到逻辑 0 电压，同理，Bitline 上的电压也会更快被 SAP 拉到逻辑 1 电压。接着 T_{p1} 和 T_{n2} 进入导通状态， T_{p2} 和 T_{n1} 进入截止状态。
- 最后，Bitline 和 /Bitline 的电压都进入稳定状态，正确的呈现了 Storage Capacitor 所存储的信息 Bit。

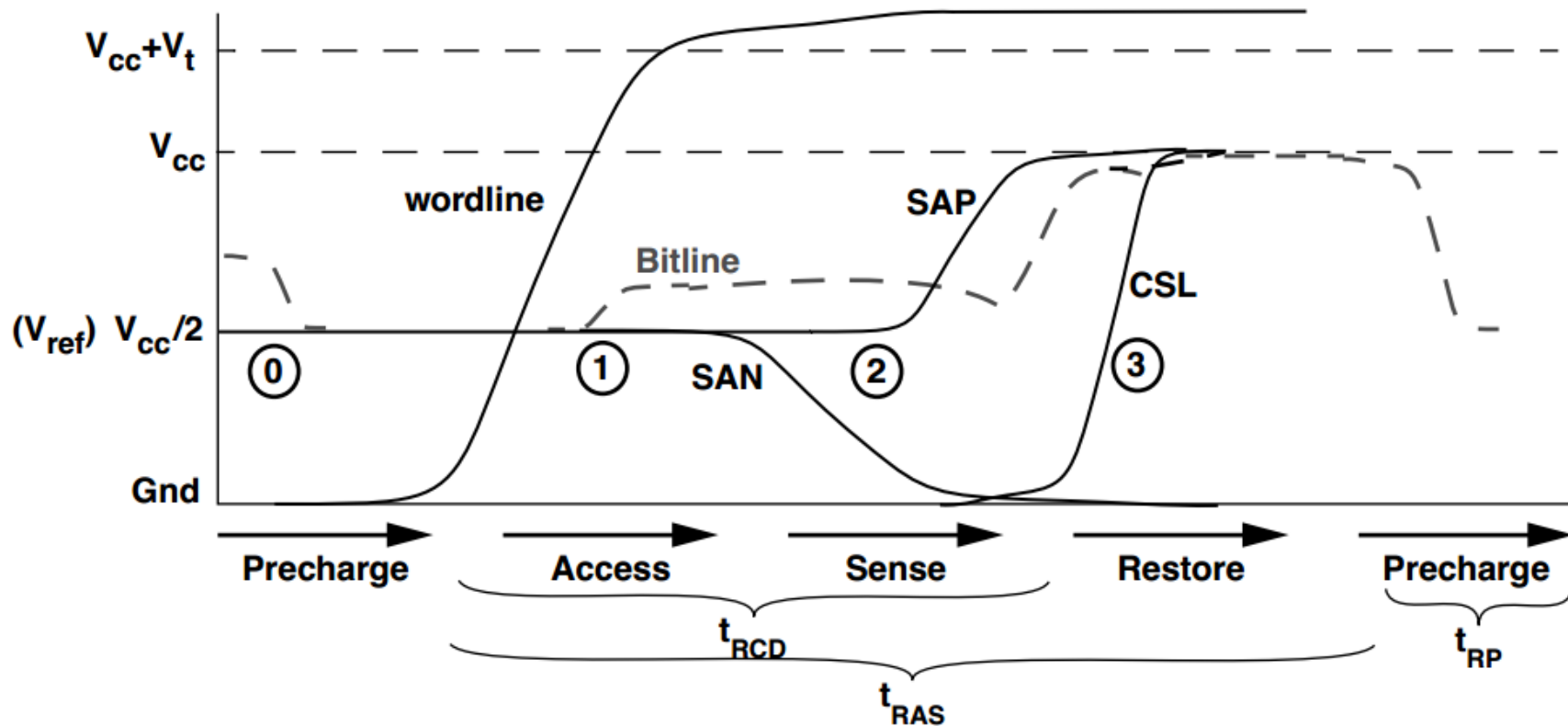


• Restore

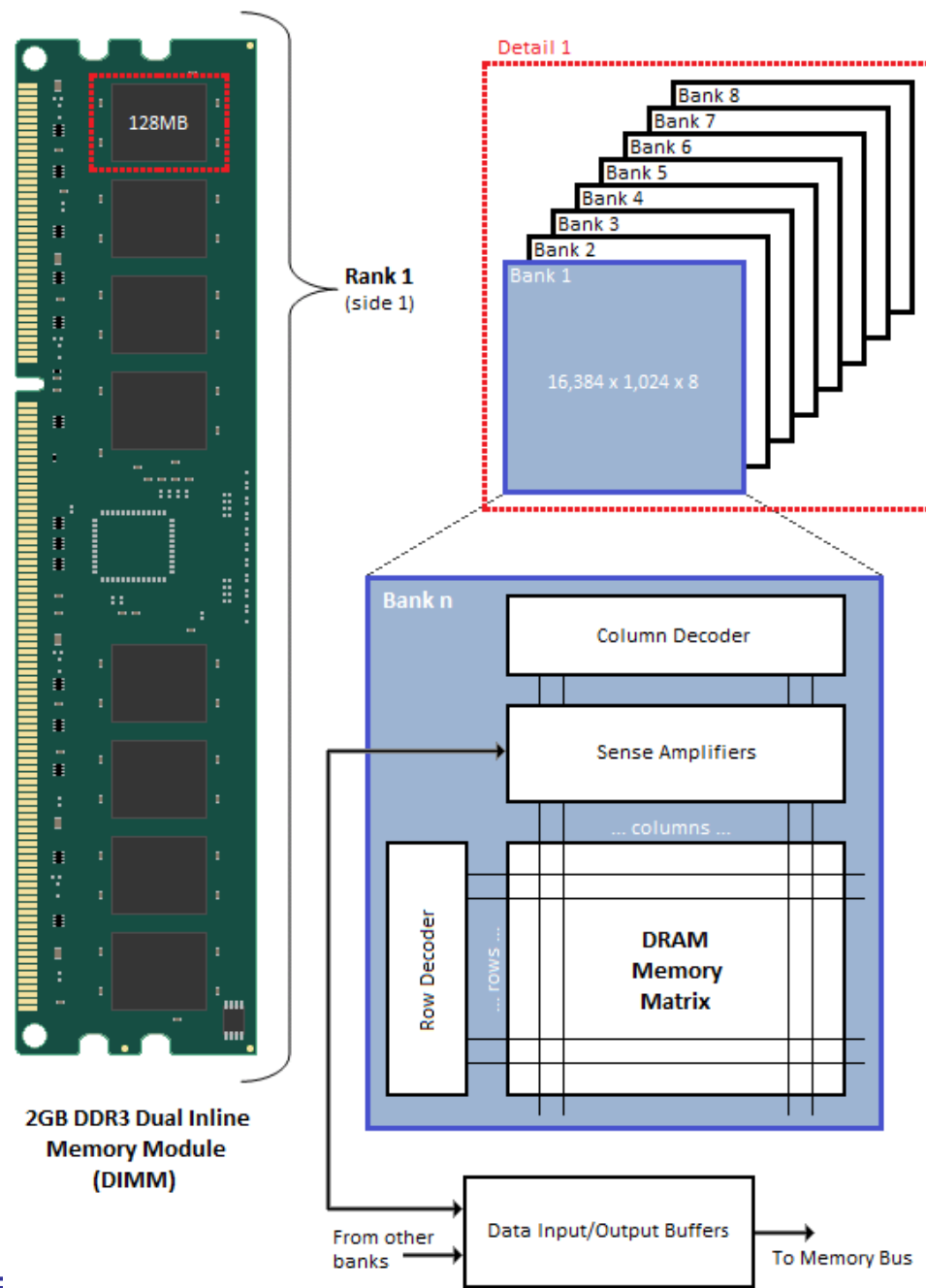
- 在完成 Sense 阶段的操作后，Bitline 线处于稳定的逻辑 1 电压 V_{cc} ，此时 Bitline 会对 Storage Capacitor 进行充电。经过特定的时间后，Storage Capacitor 的电荷就可以恢复到读取操作前的状态。



• 时序



• DRAM存储矩阵的结构



- **集中刷新**

- 在刷新周期内，对全部存储单元集中一段时间进行刷新（逐行进行），此时必须停止读写操作。

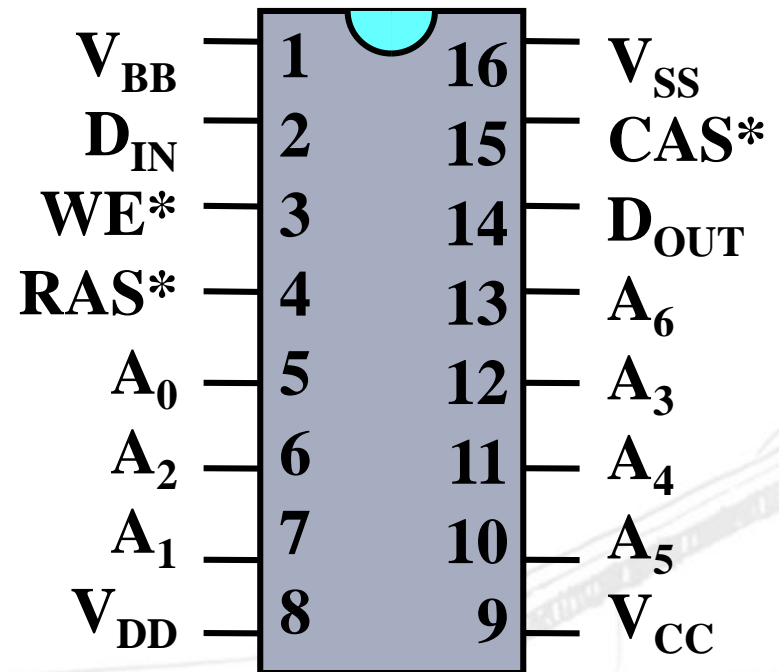
- **异步刷新**

- 在刷新周期内对所有行各刷新一次，即每隔（刷新周期/总行数） μs ，刷新一次。能充分利用刷新周期，提高刷新频率。

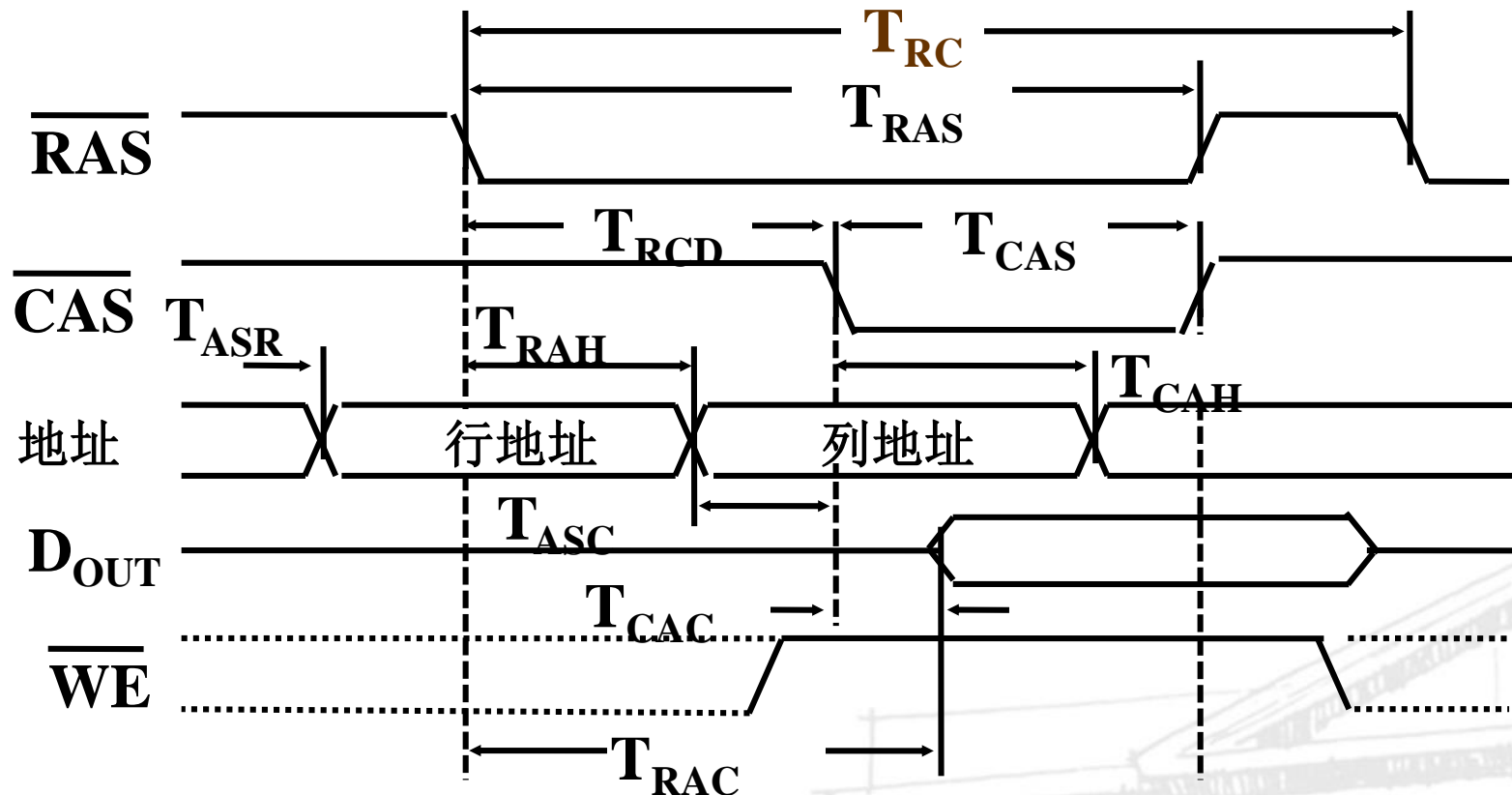


- 与SRAM相比，DRAM所用的mos管少，占硅面积小，因而功耗小，集成度高
- 但是因为采用了**电容存储电荷的原理**来寄存信息，会发生漏电现象，所以要**保持状态不变**，需要定时刷新，因为读操作会使得状态发生改变，故需要**读后再生**。且速度比SRAM慢。
- 但是由于其功耗小，集成度高，被广泛应用于计算机中。
- 需要有专门的控制器与CPU连接

- 存储容量为 $16K \times 1$
- 16个引脚：
 - 7根地址线 $A_6 \sim A_0$
 - 1根数据输入线 D_{IN}
 - 1根数据输出线 D_{OUT}
 - 行地址选通 RAS^*
 - 列地址选通 CAS^*
 - 读写控制 WE^*



• DRAM读

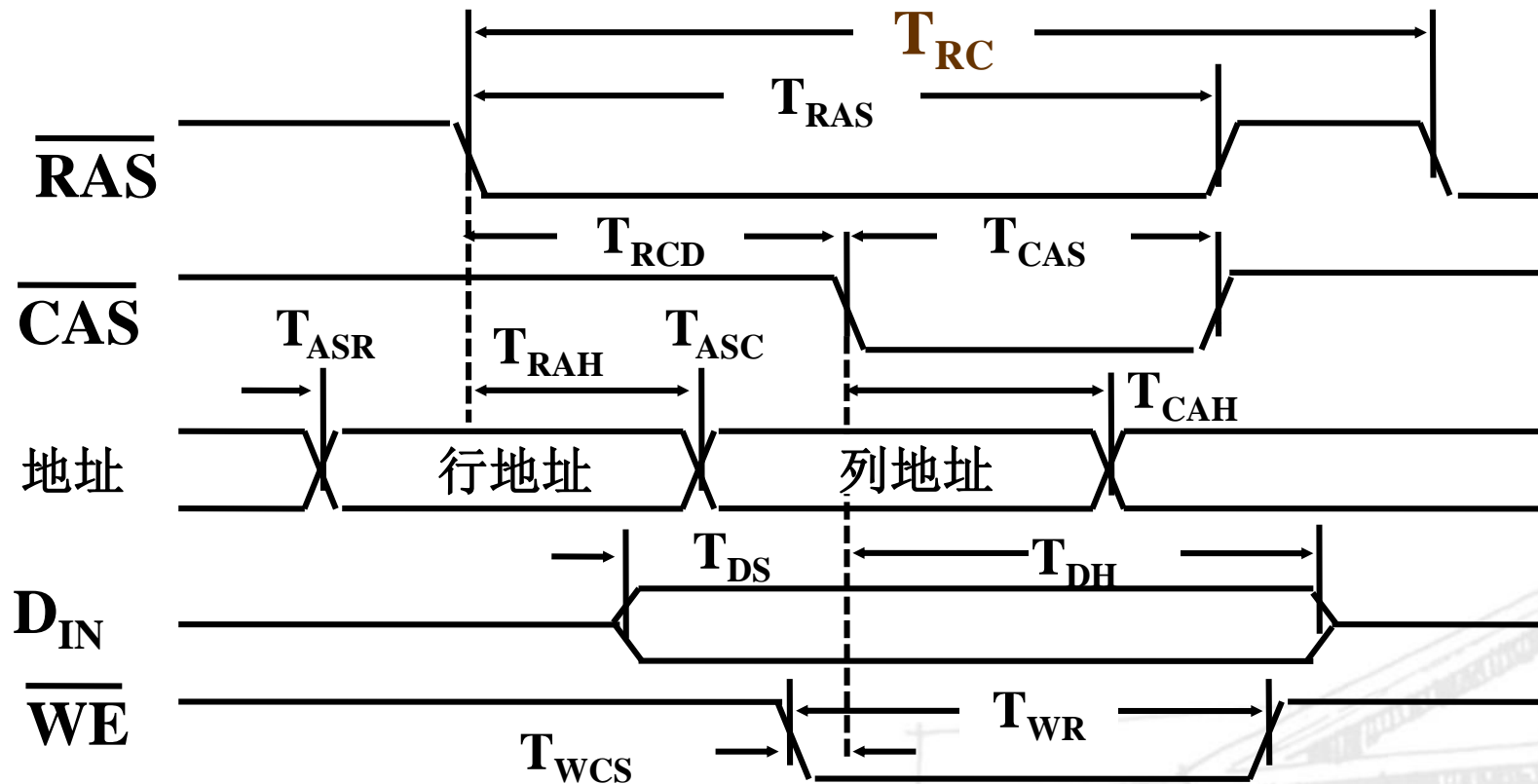


存储地址需要分两批传送

- 行地址选通信号RAS*有效，开始传送行地址
- 随后，列地址选通信号CAS*有效，传送列地址，CAS*相当于片选信号
- 读写信号WE*读有效
- 数据从D_{OUT}引脚输出



• DRAM写

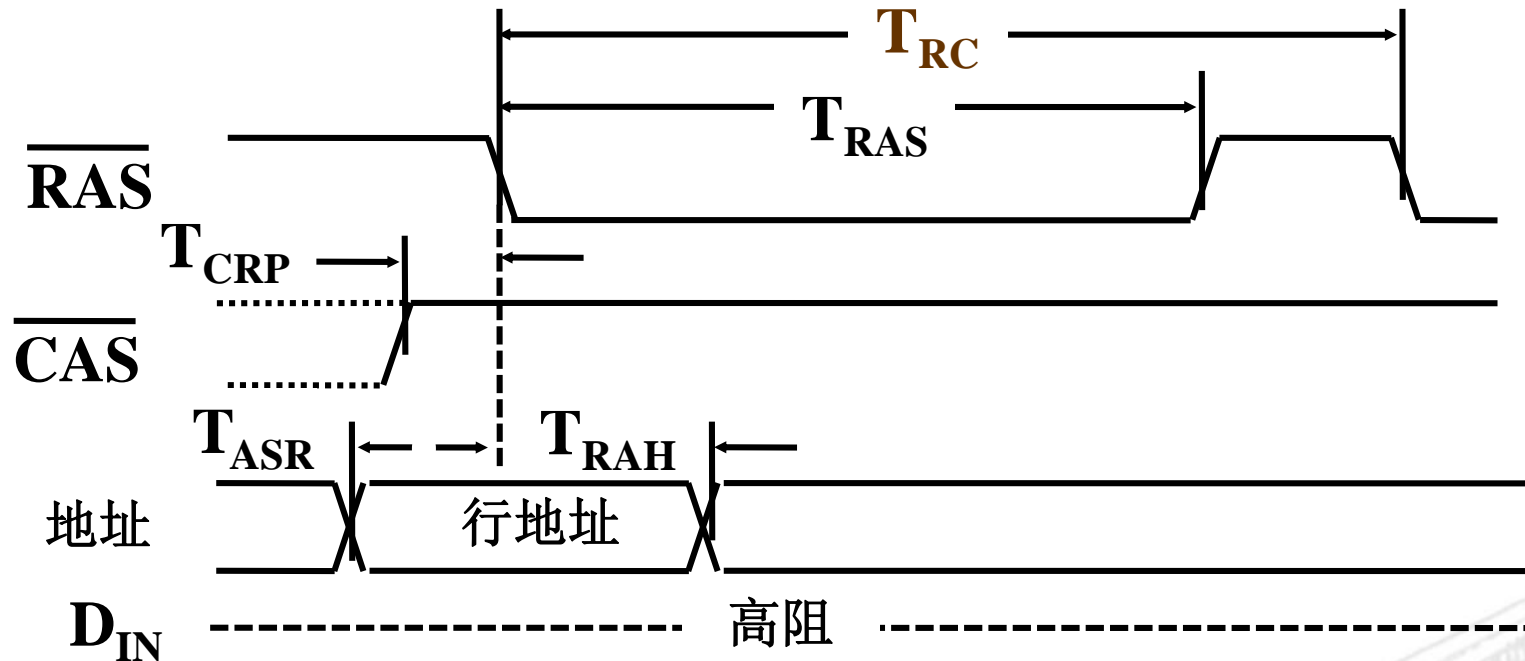


存储地址需要分两批传送

- 行地址选通信号RAS*有效，开始传送行地址
- 随后，列地址选通信号CAS*有效，传送列地址
- 读写信号WE*写有效
- 数据从D_{IN}引脚进入存储单元



• DRAM刷新

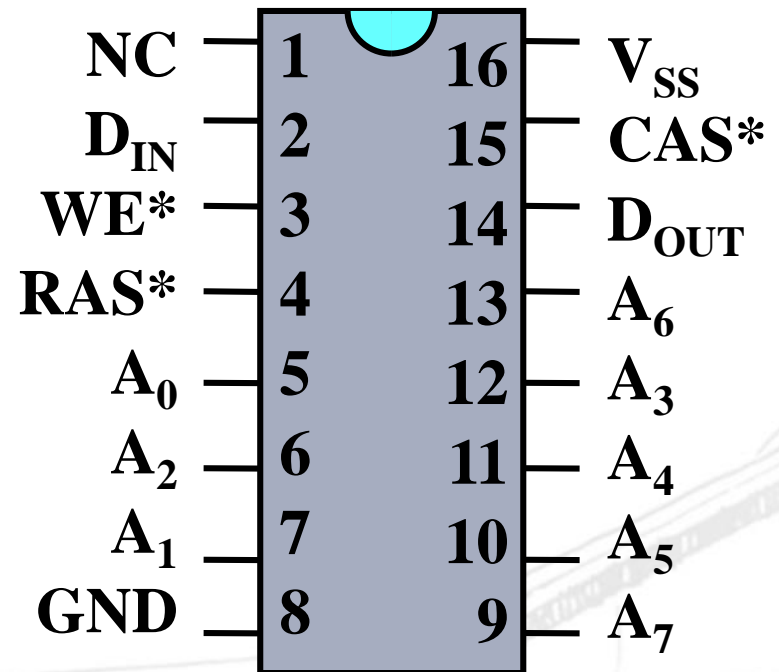


采用“仅行地址有效”方法刷新

- 行地址选通RAS*有效，传送行地址
- 列地址选通CAS*无效，没有列地址
- 芯片内部实现一行存储单元的刷新
- 没有数据输入输出
- 存储系统中所有芯片同时进行刷新
- DRAM必须每隔固定时间就刷新



- 存储容量为 $64K \times 1$
- 16个引脚：
 - 8根地址线 $A_7 \sim A_0$
 - 1根数据输入线 D_{IN}
 - 1根数据输出线 D_{OUT}
 - 行地址选通 RAS^*
 - 列地址选通 CAS^*
 - 读写控制 WE^*



-
- **存储器是由_____， _____， _____， _____四个基本结构组成的**
 - **SRAM的基本存储单元是什么？**
 - **DRAM的基本存储单元是什么？**
 - **SRAM与DRAM的相同点和不同点**

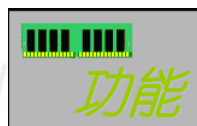
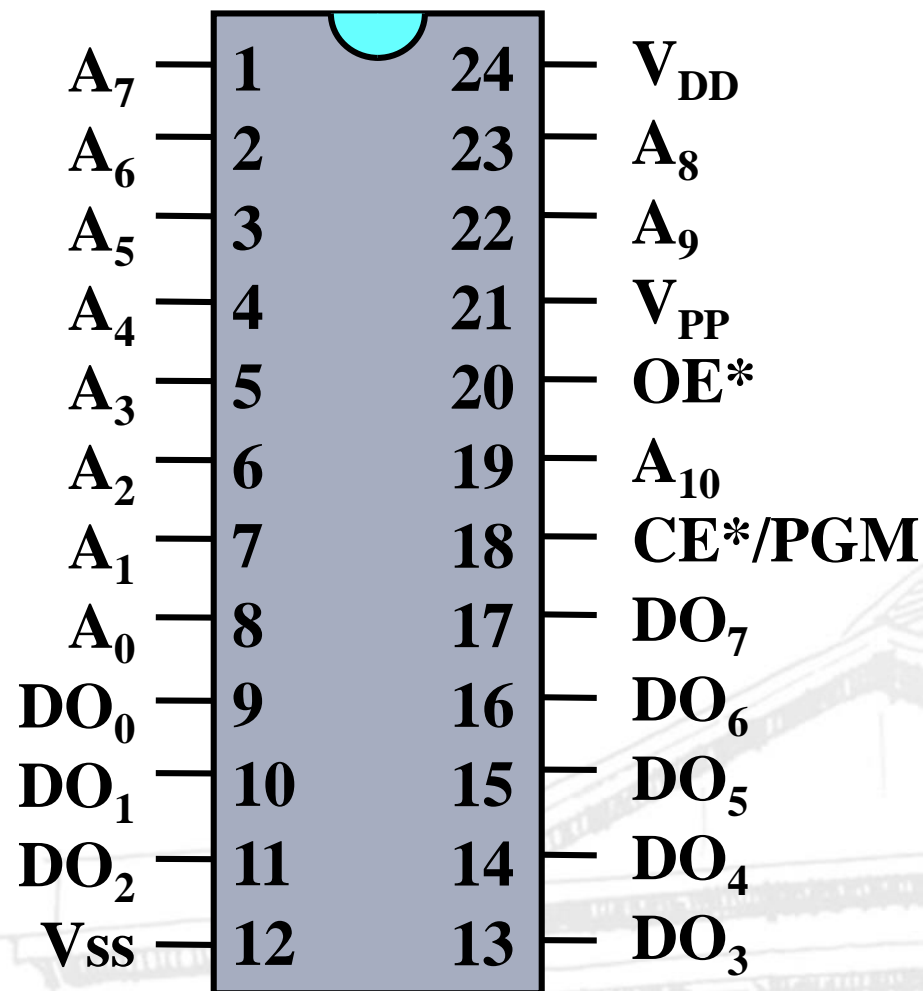
<https://www.bilibili.com/video/BV1PY411N7SQ>



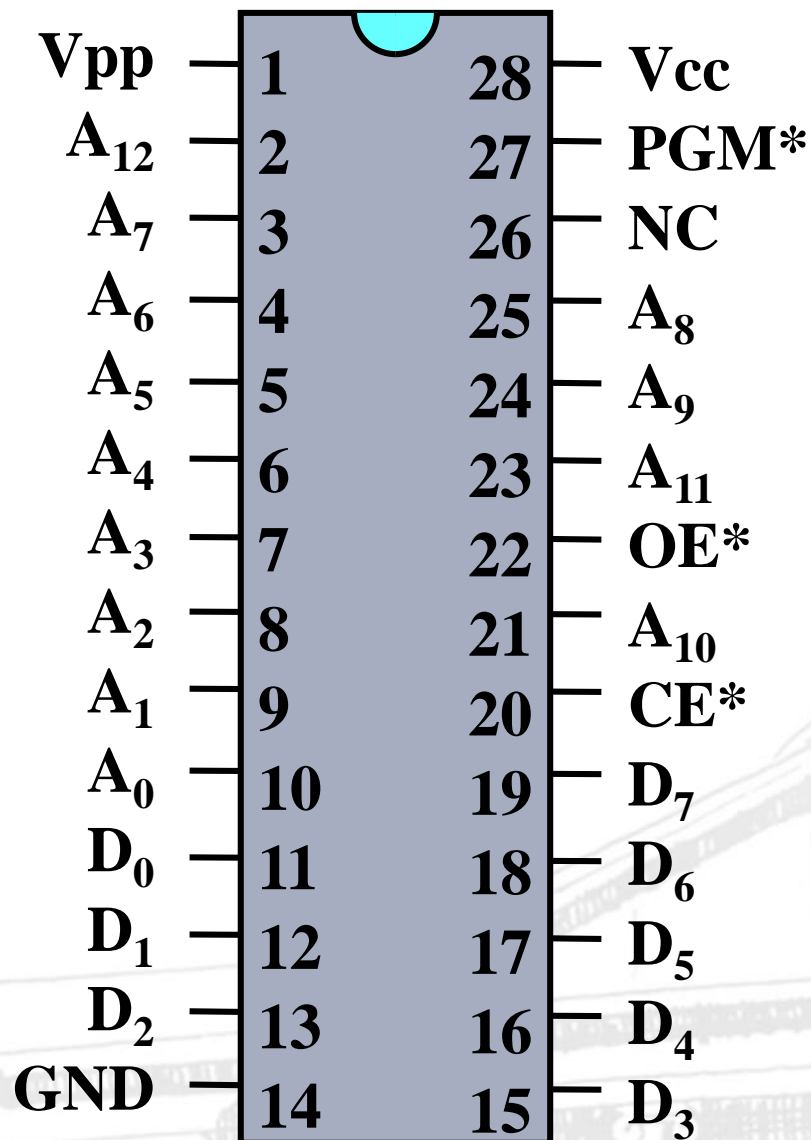
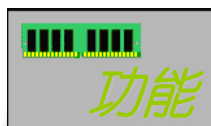


- **顶部开有一个圆形的石英窗口，用于紫外线透过擦除原有信息**
- **一般使用专门的编程器（烧写器）进行编程**
- **编程后，应该贴上不透光封条**
- **出厂未编程前，每个基本存储单元都是信息1**
- **编程就是将某些单元写入信息0**

- 存储容量为 $2K \times 8$
- 24个引脚：
 - 11根地址线 $A_{10} \sim A_0$
 - 8根数据线 $DO_7 \sim DO_0$
 - 片选/编程 CE^*/PGM
 - 读写 OE^*
 - 编程电压 V_{PP}

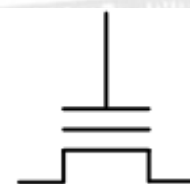
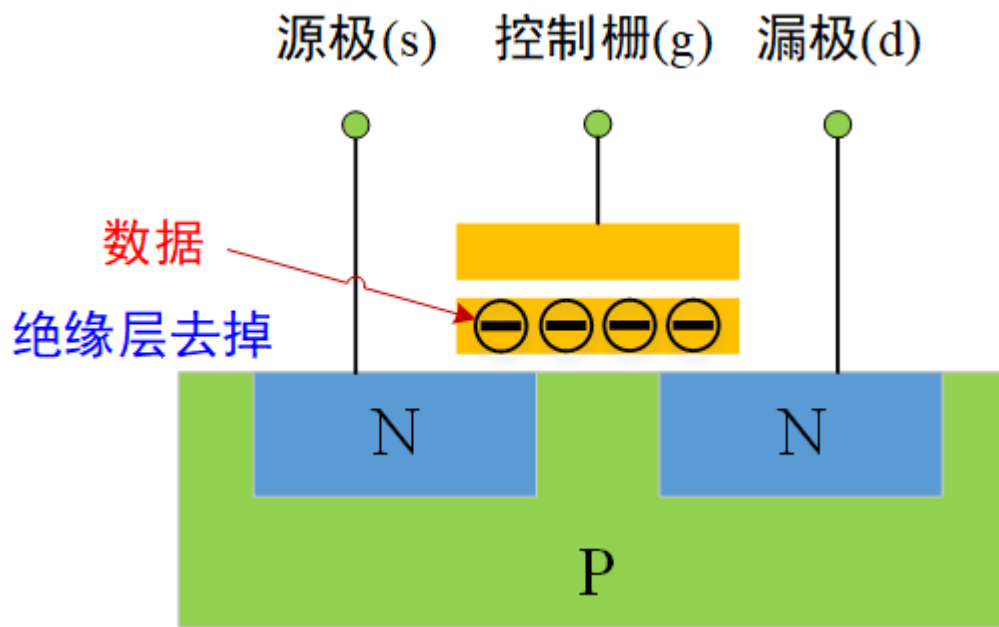
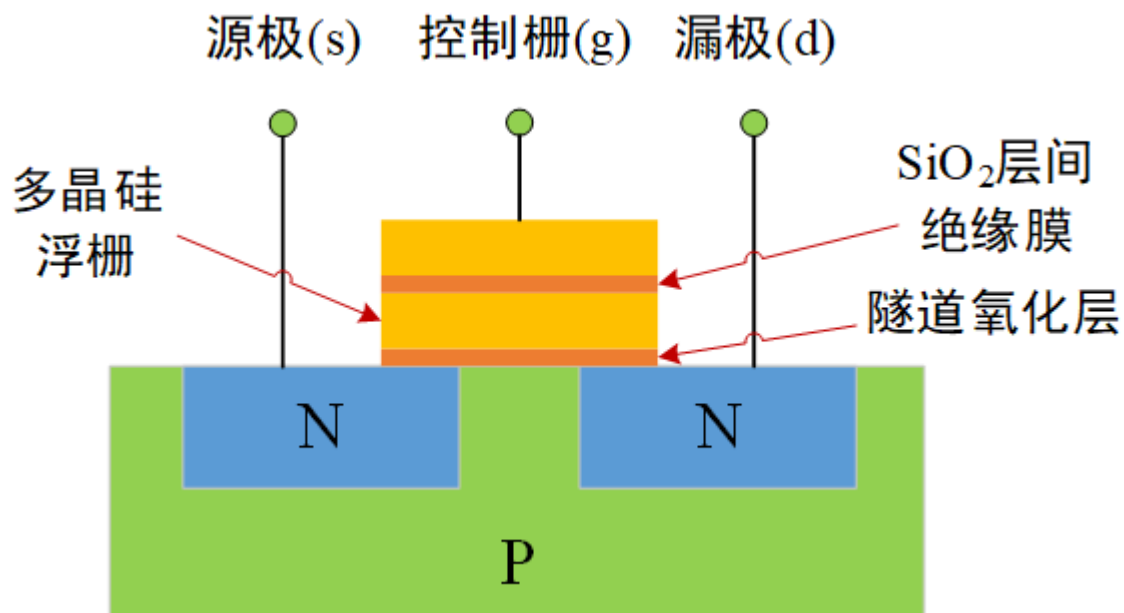


- 存储容量为 $8K \times 8$
- 28个引脚：
 - 13根地址线 $A_{12} \sim A_0$
 - 8根数据线 $D_7 \sim D_0$
 - 片选 CE^*
 - 编程 PGM^*
 - 读写 OE^*
 - 编程电压 V_{PP}

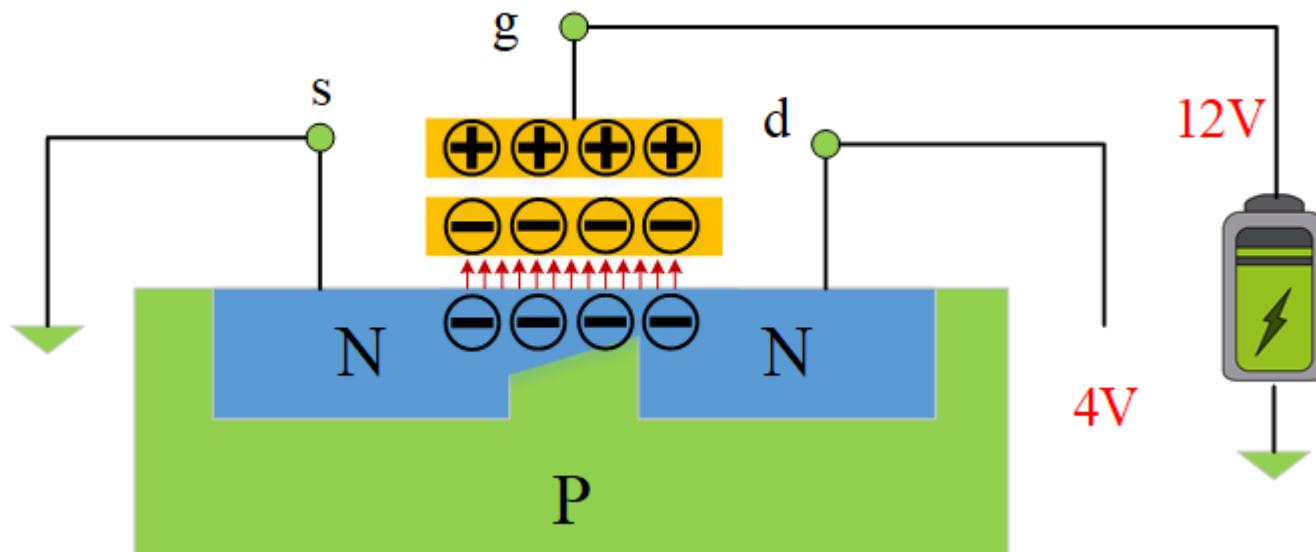


- **EEPROM (Electrically Erasable Programmable read only memory)**
 - 即电可擦可编程只读存储器，是一种掉电后数据不丢失（不挥发）存储芯片。
- **快闪存储器（英语：Flash Memory）**
 - 全名叫Flash **EEPROM** Memory，是一种电子式可清除程序化只读存储器的形式，允许在操作中被多次擦或写的存储器。
 - Flash又分为NAND flash和NOR flash二种。
- **EEPROM使用浮栅场效应管(Floating Gate FET)作为基本存储单元来存储数据**

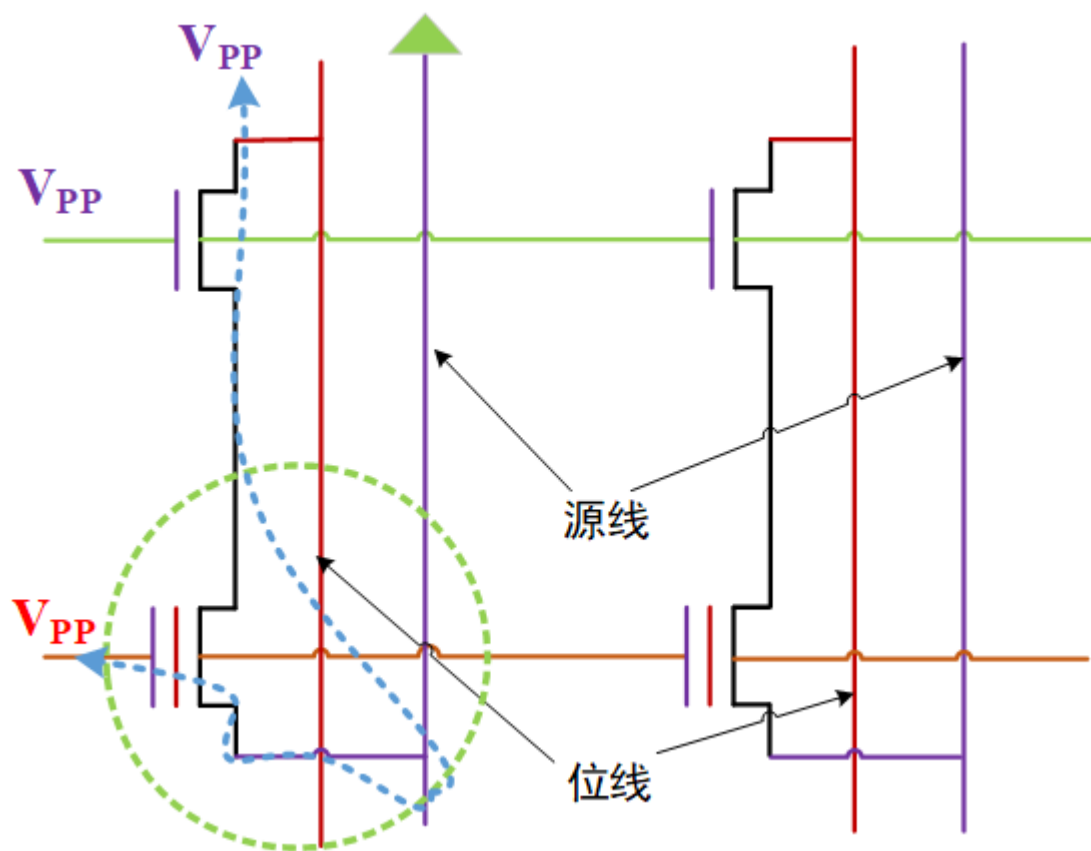
浮栅场效应管



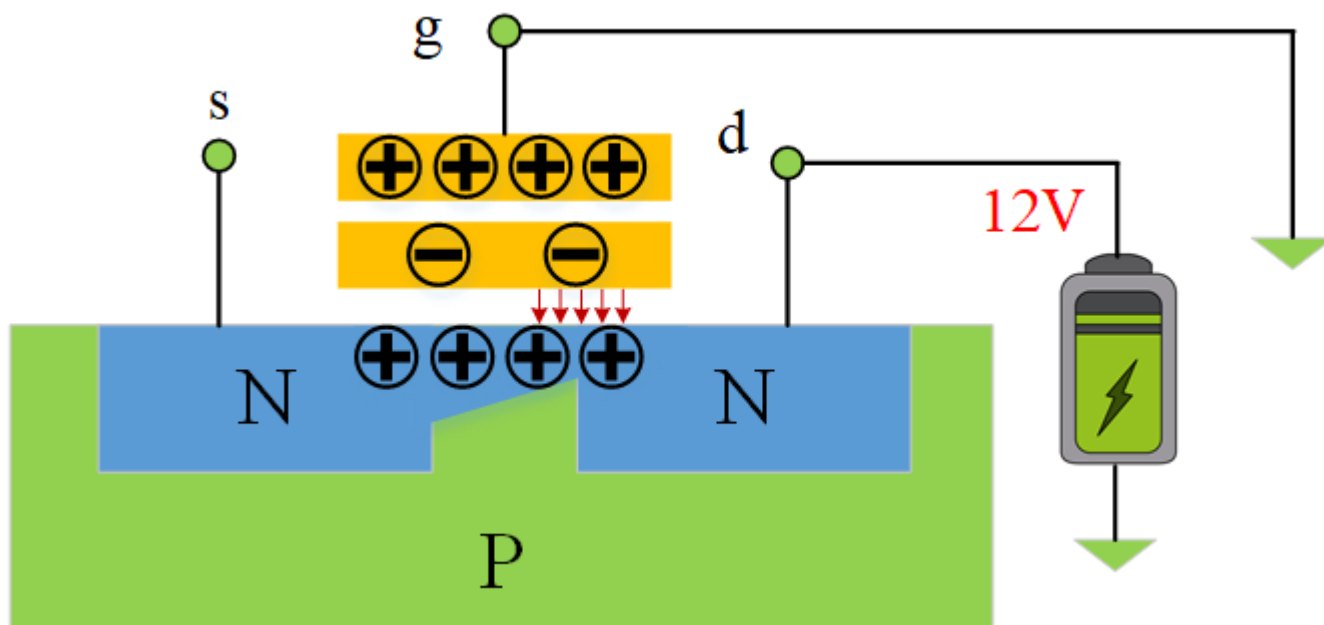
- 通常利用F-N隧道效应 (Fowler-Nordheim tunneling) 或热电子注入对存储单元进行“擦除”或“写入”操作
- 对存储单元进行“编程 (写入)”操作，代表逻辑“0”。



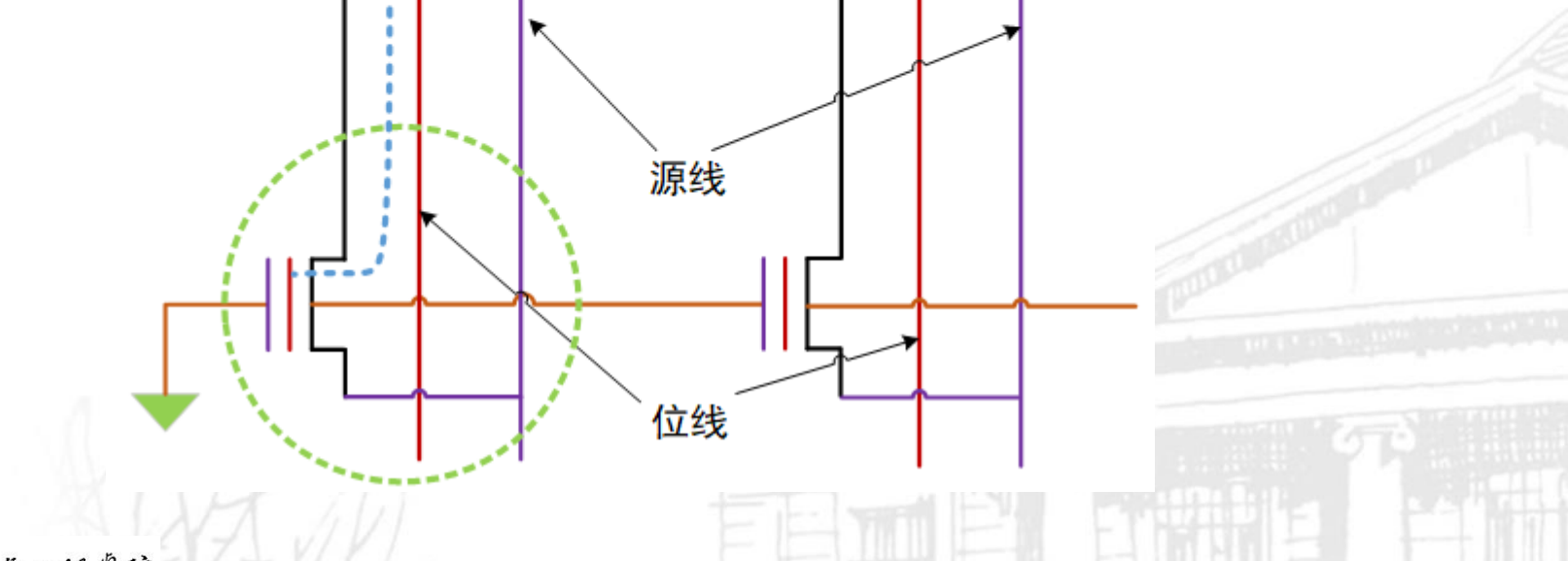
- 进行存储单元的“编程”操作时，源线（Sources Lines, SL）与位线（Bit Lines, BL）均为低电平，而控制栅线（CL）为高电平，当对应存储单元的选通管打开时（Word Line, $WL=V_{PP}$ ），如下图所示



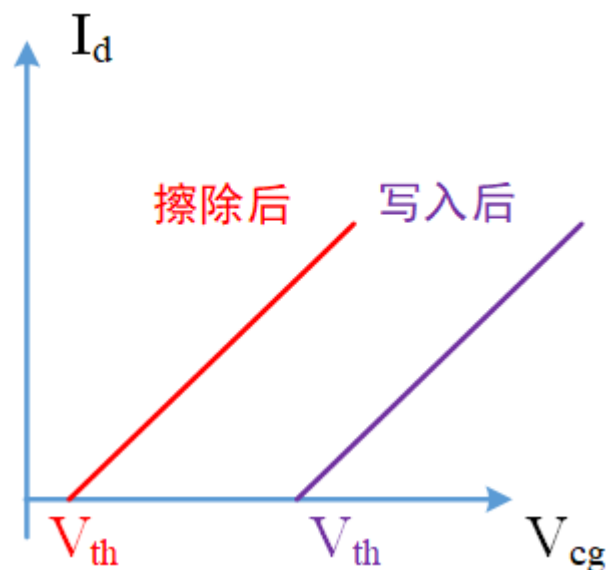
- 对存储单元进行“擦除”操作，就是将浮栅中电子释放的过程



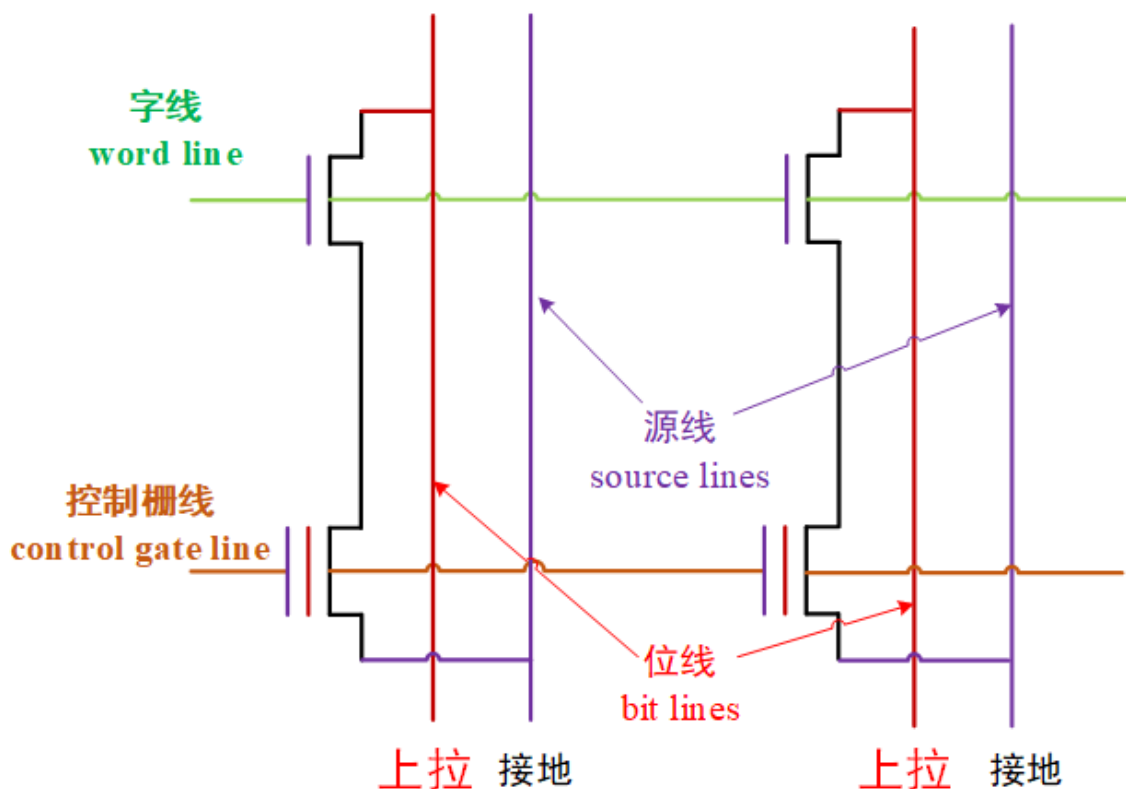
-



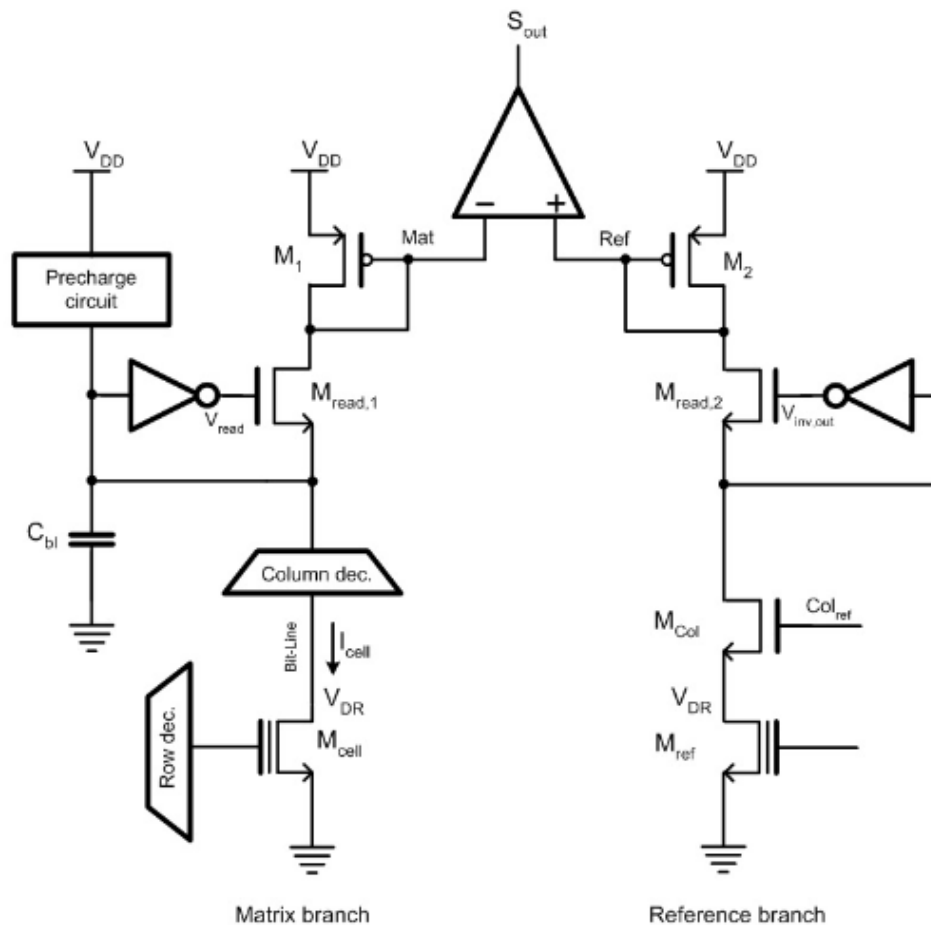
- 读取时位线被上拉，给控制栅极加一个中间电压，如果浮动栅极有电荷，DS不导通，位线读出1。否则位线读出0。
- 位线上的电压，不能直接输出到数据总线上。为了正确的读取数据，并与逻辑电平匹配，还需要差分感知放大电路和输出缓冲电路，差分感知放大电路或输出缓冲电路中有**反相器**，所以数据总线上的数据，与位线上的数据**相反**。



擦除和写入后，控制栅极的开启发生变化



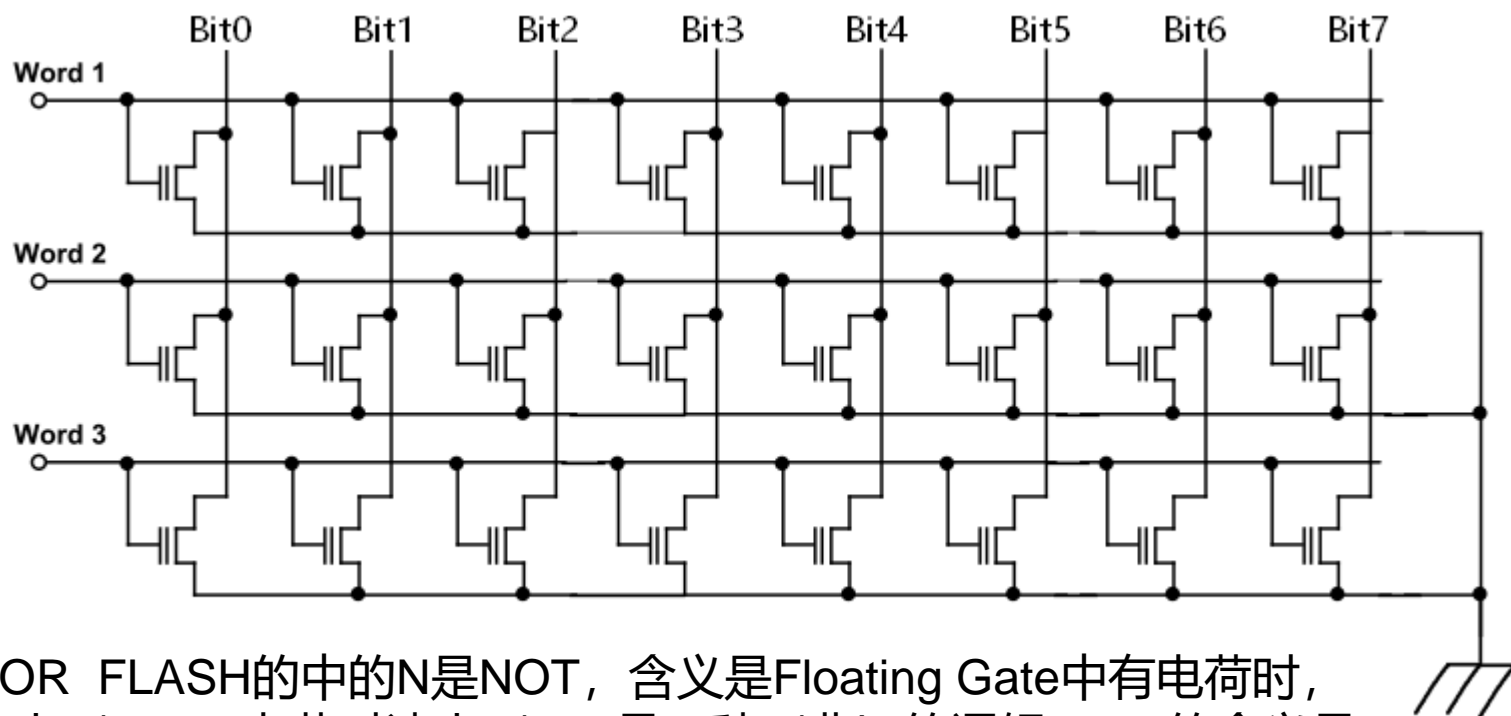
• 实际使用的感知放大和数据缓冲



擦除----FG不带电子----逻辑“1”----mos管容易导通，导通电流大----位线为0----数据总线位:1
 写入（编程）----FG带电子----逻辑“0”----mos管难导通，导通电流小----位线为1----数据总线位:0



- **每个Bit Line下的基本存储单元是并联的，当某个Word Line被选中后，就可以实现对该Word的读取，也就是可以实现位读取（即Random access），且具有较高的读取速率。**



NOR FLASH中的N是NOT，含义是Floating Gate中有电荷时，读出 '0'，无电荷时读出 '1'，是一种 '非' 的逻辑；OR的含义是同一个Bit Line下的各个基本存储单元是并联的，是一种 '或' 的逻辑，这就是NOR 的由来。

- 基本存储单元的并联结构决定了金属导线占用很大的面积, 因此NOR FLASH的存储密度较低, 无法适用于需要大容量存储的应用场合, 即适用于code-storage, 不适用于data-storage
- 基本存储单元的并联结构决定了NOR FLASH具有存储单元可独立寻址且读取效率高的特性, 因此适用于code-storage, 且程序可以直接在NOR 中运行 (即具有RAM的特性)
- NOR FLASH写入采用了热电子注入方式, 效率较低, 因此NOR写入速率较低, 不适用于频繁擦除/写入场合。

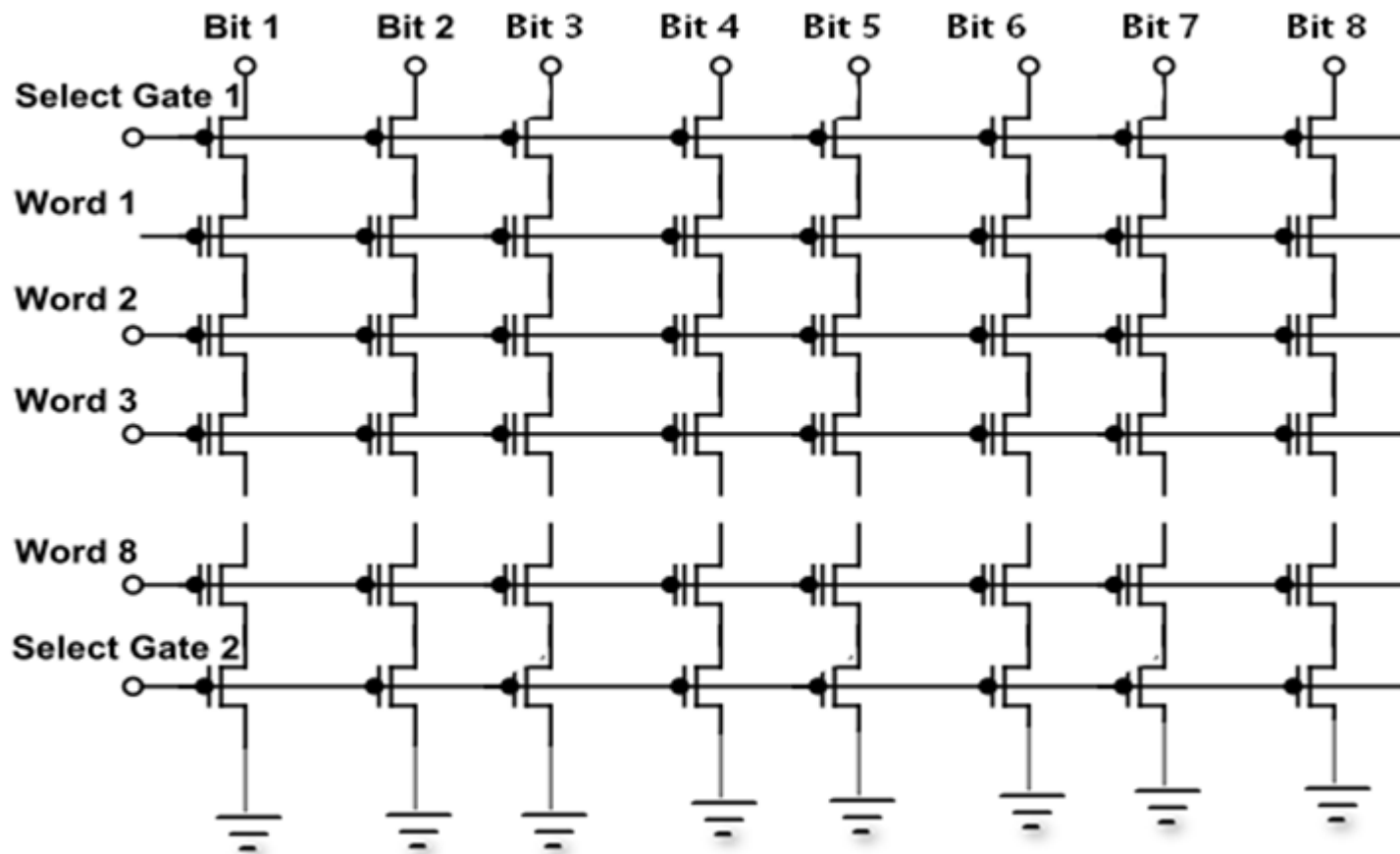


NAND型 FLASH

每个Bit Line下的基本存储单元是串联的，NAND读取数据的单位是Page。

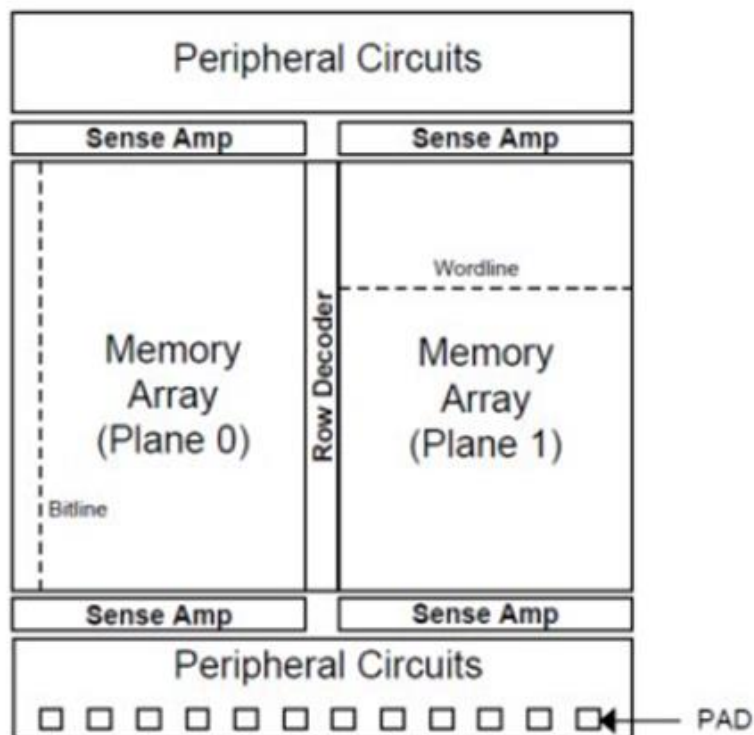
NAND读写是以页为单位的，擦除是以块为单位的。

NAND无法实现位读取（即Random access），程序代码也就无法在NAND上运行。

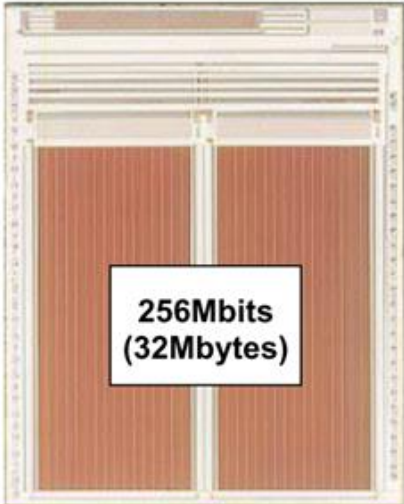
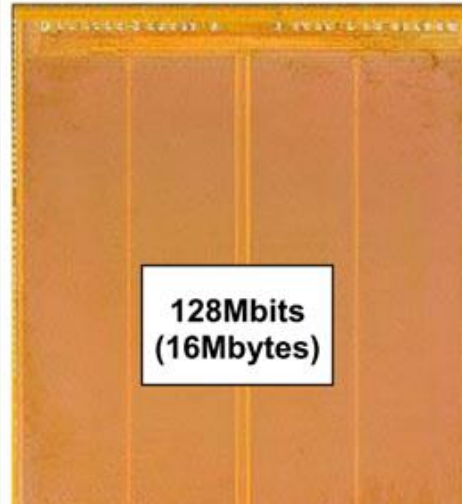


- 基本存储单元的串联结构减少了金属导线占用的面积，Die的利用率很高，因此NAND FLASH存储密度高，适用于需要大容量存储的应用场合，即适用于data-storage。
- 基本存储单元的串联结构决定了NAND FLASH无法进行位读取，也就无法实现存储单元的独立寻址，因此程序不可以直接在NAND 中运行,因此NAND是以Page为读取单位和写入单位，以Block为擦除单位。
- NAND FLASH写入采用F-N隧道效应方式，效率较高，因此NAND擦除/写入速率很高，适用于频繁擦除/写入场合。同时NAND是以Page为单位进行读取的，因此读取速率也不算低（稍低于NOR）

NAND Flash与NOR Flash对比



NAND Flash memory floorplan

32MBytes (256Mbits) DiskOnChip Millennium Plus (NAND-based)	16MBytes (126Mbits) NOR Flash
 <p>256Mbits (32Mbytes)</p>	 <p>128Mbits (16Mbytes)</p>
244% better than NOR	

• 存储器地址空间

- 存储器中每一个存储单元 (byte或Word) 的存储位置, 都对应地址线上电平一种组合方式, 这些地址线电平的集合, 是存储器的片内地址空间。

• 总线地址空间

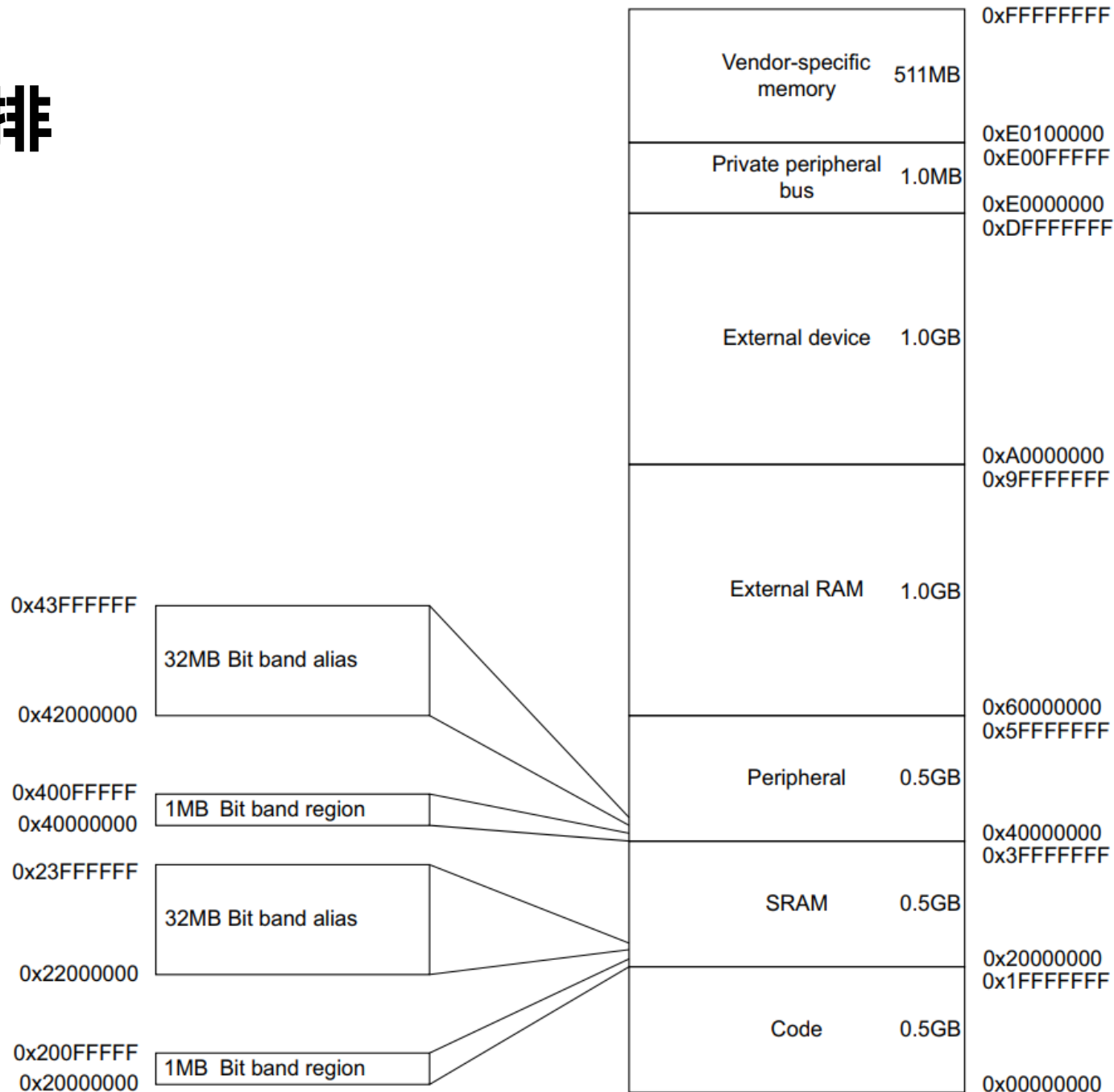
- 存储器与CPU相连后, 存储器中每一个存储单元 (byte或Word), 都对应CPU总线上电平一种组合方式, 这些地址线电平的集合, 是存储器在总线上的地址空间。
- 存储器与CPU相连的连接方式不同, 总线地址空间也不一样。
- 统一编址、独立编址



- **STM32F401的存储器：**
 - 统一编址
 - 4G的内存空间（32位）
 - 小端模式
 - 64KB SRAM
 - 128KB Flash memory



• 地址编排



- **FLASH、ROM、RAM是如何与CPU相连的？**



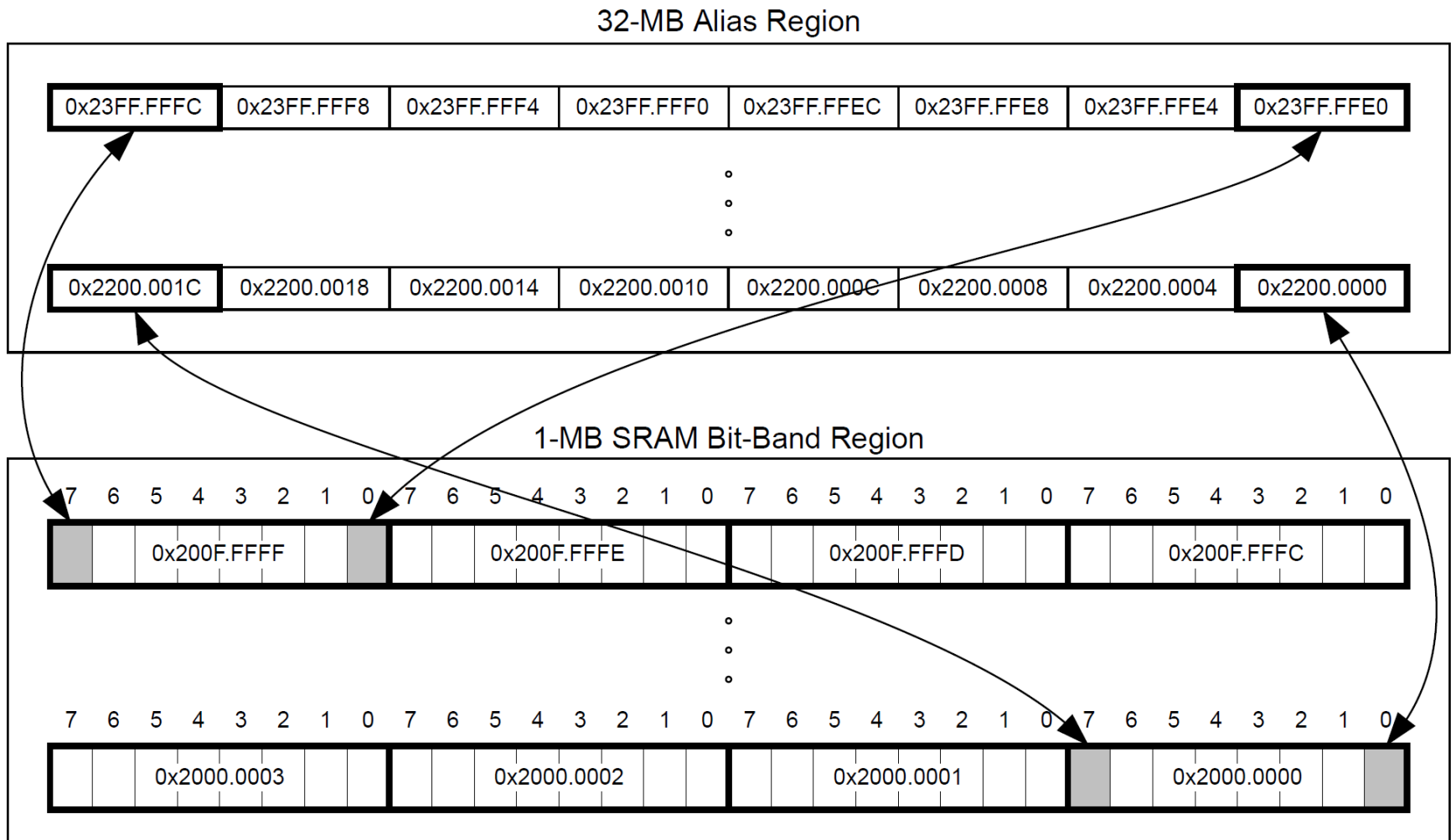
- **SRAM起始地址0x2000.0000**
 - Bit-band 映射的起始地址0x2200.0000
- **使用方法:**
 - bit-band alias = bit-band base + (byte offset * 32) + (bit number * 4)
- **例: 0x2000.1000 bit 3**
 - $0x2200.0000 + (0x1000 * 32) + (3 * 4) = 0x2202.000C$

```
#define HWREG(x)  
    (*((volatile uint32_t *)(x)))
```

```
#define HWREGBITW(x, b)  
    HWREG(((uint32_t)(x) & 0xF0000000) | 0x02000000 |  
          (((uint32_t)(x) & 0x000FFFFF) << 5) | ((b) << 2))
```



• Bit-band 示意图

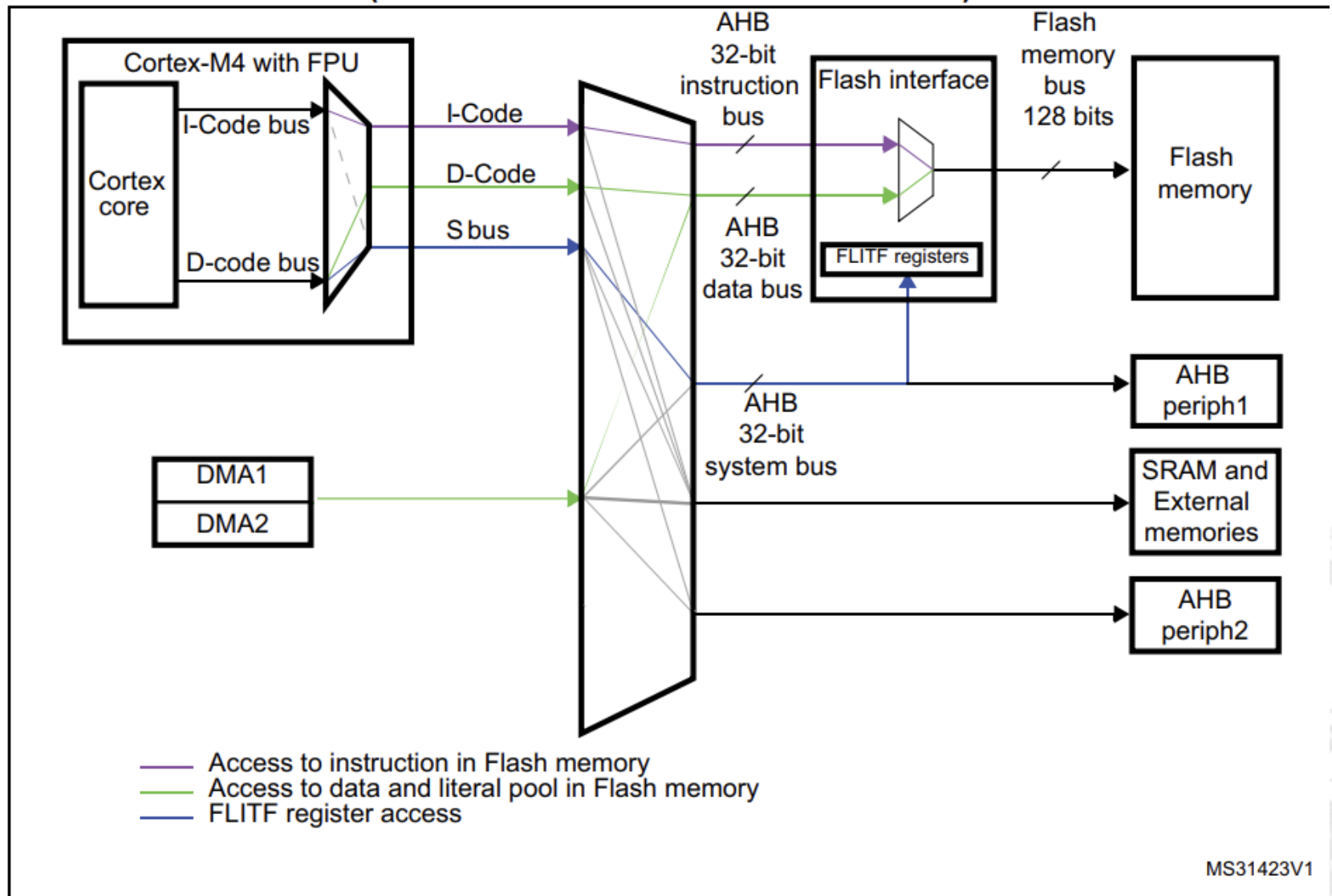


- **Flash memory**

- 通过I-Code bus和D-Code bus与Flash interface读取数据
- 通过S bus读写Flash interface的寄存器擦除和写入数据
- 256KB 容量
- 128位读取
- 8位、16位、32位、64位编程
- 可以按照块擦除，也可以整体擦除
- 四个16KB的块，一个64KB的块、一个128KB的块，总共256KB。
- 地址从0x08000000开始
- 被重新映射到了从0x00000000开始到0x0007FFFF的区域



- Flash memory



MS31423V1

- Flash memory

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

- **Flash 的编程**

- 擦除函数HAL_FLASHEx_Erase

```
HAL_StatusTypeDef HAL_FLASHEx_Erase(FLASH_EraseInitTypeDef *pEraseInit,  
uint32_t *SectorError)
```

- pEraseInit为一个结构体指针

```
typedef struct  
{  
    uint32_t TypeErase;  
    uint32_t Banks;  
    uint32_t Sector;  
    uint32_t NbSectors;  
    uint32_t VoltageRange;  
}  
FLASH_EraseInitTypeDef
```



• Flash 的编程

- 擦除函数HAL_FLASHEx_Erase
- pEraseInit为一个结构体指针

- TypeErase 擦除的方式，可以是块擦拭，也可以是整体擦除

```
#define FLASH_TYPEERASE_SECTORS      0x00000000U /*!< Sectors erase only */
#define FLASH_TYPEERASE_MASSERASE    0x00000001U /*!< Flash Mass erase activation
```

- Banks 对于STM32F401RB芯片，Banks固定为FLASH_BANK_1

```
#define FLASH_BANK_1    1U /*!< Bank 1 */
```

- Sector为要块擦除模式下，要擦除的块

```
#define FLASH_SECTOR_0    0U /*!< Sector Number 0 */
#define FLASH_SECTOR_1    1U /*!< Sector Number 1 */
#define FLASH_SECTOR_2    2U /*!< Sector Number 2 */
#define FLASH_SECTOR_3    3U /*!< Sector Number 3 */
#define FLASH_SECTOR_4    4U /*!< Sector Number 4 */
#define FLASH_SECTOR_5    5U /*!< Sector Number 5 */
```

- NbSectors 擦除的块的数量，一般为1，如果大于1，则从Sector块开始依次擦除
- VoltageRange 芯片的供电电压 填FLASH_VOLTAGE_RANGE_3



- **Flash 的编程**
 - 编程函数HAL_FLASH_Program

```
HAL_StatusTypeDef HAL_FLASH_Program(uint32_t TypeProgram,  
uint32_t Address, uint64_t Data)
```

- **TypeProgram 编程的字长**

FLASH_TYPEPROGRAM_BYTE, 运行一次本函数写入8位数据;

FLASH_TYPEPROGRAM_HALFWORD, 运行一次本函数写入16位数据;

FLASH_TYPEPROGRAM_WORD, 运行一次本函数写入32位数据;

FLASH_TYPEPROGRAM_DOUBLEWORD, 运行一次本函数写入64位数据;

- **Address 要编程的FLASH地址**
 - **Data 要写入的数据**



- Flash 的编程示例

- 擦除SECTOR 4

```
FLASH_EraseInitTypeDef EraseInit;  
EraseInit.VoltageRange=FLASH_VOLTAGE_RANGE_3;  
EraseInit.Banks=FLASH_BANK_1;  
EraseInit.TypeErase=FLASH_TYPEERASE_SECTORS;  
EraseInit.NbSectors=1;  
EraseInit.Sector=FLASH_SECTOR_4;  
uint32_t FLASH_EraseStatus = HAL_ERROR;  
HAL_FLASHEx_Erase(&EraseInit, &FLASH_EraseStatus);
```

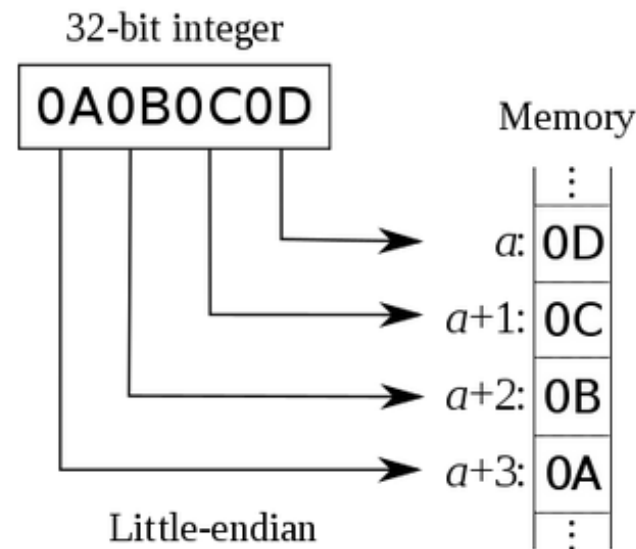
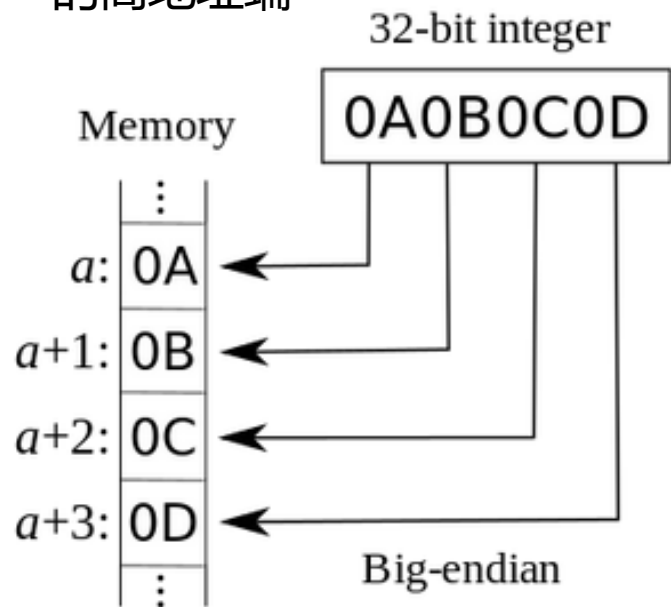
- 在地址0x08010000处依次写入0、1、2、3到100

```
int i;  
for(i=0;i<100;i++){  
    HAL_FLASH_Program(FLASH_TYPEPROGRAM_BYTE, 0x08010000+i, i);  
}
```

- **字节顺序，又称端序或尾序（Endianness），在计算机科学领域中，指存储器中或在数字通信链路中，组成多字节的字的字节的排列顺序。**

1) Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

2) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端

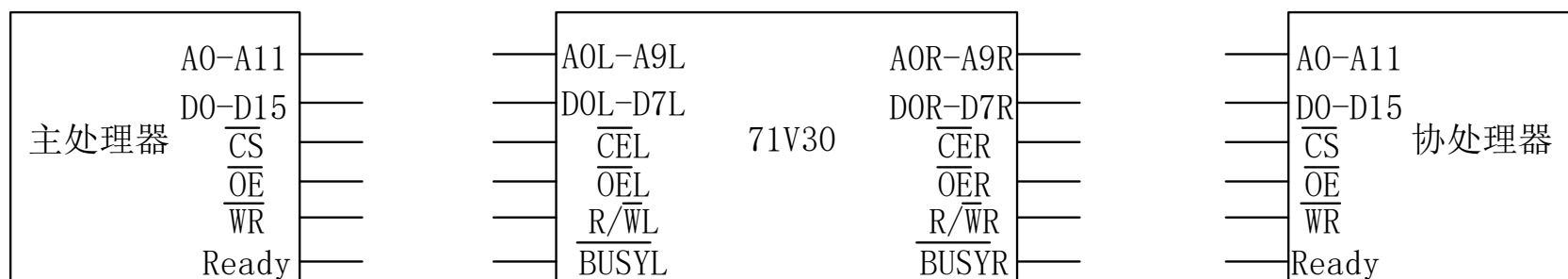


- **uint32_t a=100000, &a=0x20000010, 那么, 存储变量a是如何存储的? 存储变量a所用的存储单元的地址分别是哪些? 这些存储单元里各存储了什么? (存储器以字节为单位编址)**



示例

- 假设一个高精度数据采集系统，由一个主处理器和一个协处理器组成。主处理器负责数据处理，协处理器负责显示和通信。
- 主处理器和协处理器使用双口RAM（71V30 1k*8bit）通信。



CS的起始地址为0x6000.0000

- 如果每次传输16bits的数据，该如何连接？
- 如果共享内存需要2k*16 bits，该如何连接？所占用的地址空间是什么？

