中国科学技术大学
University of Science and Technology of China

Ch 6 存储系统

王 超

中国科学技术大学计算机学院
高能效智能计算实验室

2022年春

Arch 2030 in ISCA 2016

中国科学技术大学
University of Science and Technology of China

Arch2030: A Vision of Computer Architecture Research over the Next 15 Years

CCC
Computing Community Consortium
Catalyst

http://arch2030.cs.washington.edu/

专栏 中国计算机学会通讯 第13卷 第7期 2017年7月

计算机体系结构2030：未来15年的研究愿景*

作者：路易斯·塞泽(Luis Ceze)
马克·希尔(Mark D. Hill)
托马斯·维尼施(Thomas F. Wenisch)
译者：鄢贵海　王　颖　刘宇航
中国科学院计算技术研究所

关键词：计算机体系结构　愿景规划

编者按：过去10年，很多计算机体系结构研究学者都在感叹体系结构的研究步履维艰，在很多国际会议的讨论中出现了类似“体系结构正在消亡(Architecture is dying)”，“永生的体系结构(Long lived architecture)”等有趣的争论，其背后的动因是对传统的冯·诺伊曼体系结构在应对多样化应用的局限的深刻认识和对摩尔定律的放缓甚至终结的担心。这篇文章源于2016国际体系结构年会(ISCA 2016)上的Workshop讨论和公开问卷。然而，主旨并不是回答体系结构是否还值得无数的学者前赴后继，而是以一个更加开放的心态去探讨体系结构如何求变以为新应用提供更好服务。本文的贡献者达到近40人，大部分来自具有悠久体系结构研究历史的美国一流大学，学术背景也很多样化。不同领域的思想碰撞为我们勾勒出未来15年体系结构研究可能的图谱。

应用趋势、器件技术和系统结构的发展驱动了信息技术的进步。然而，这一进步的早期引擎——摩尔定律和登纳德缩放定律(Dennard scaling)——收益正在快速地变得越来越小。计算机界已经直接面临新的挑战：如何确保信息技术有一个坚实的未来。

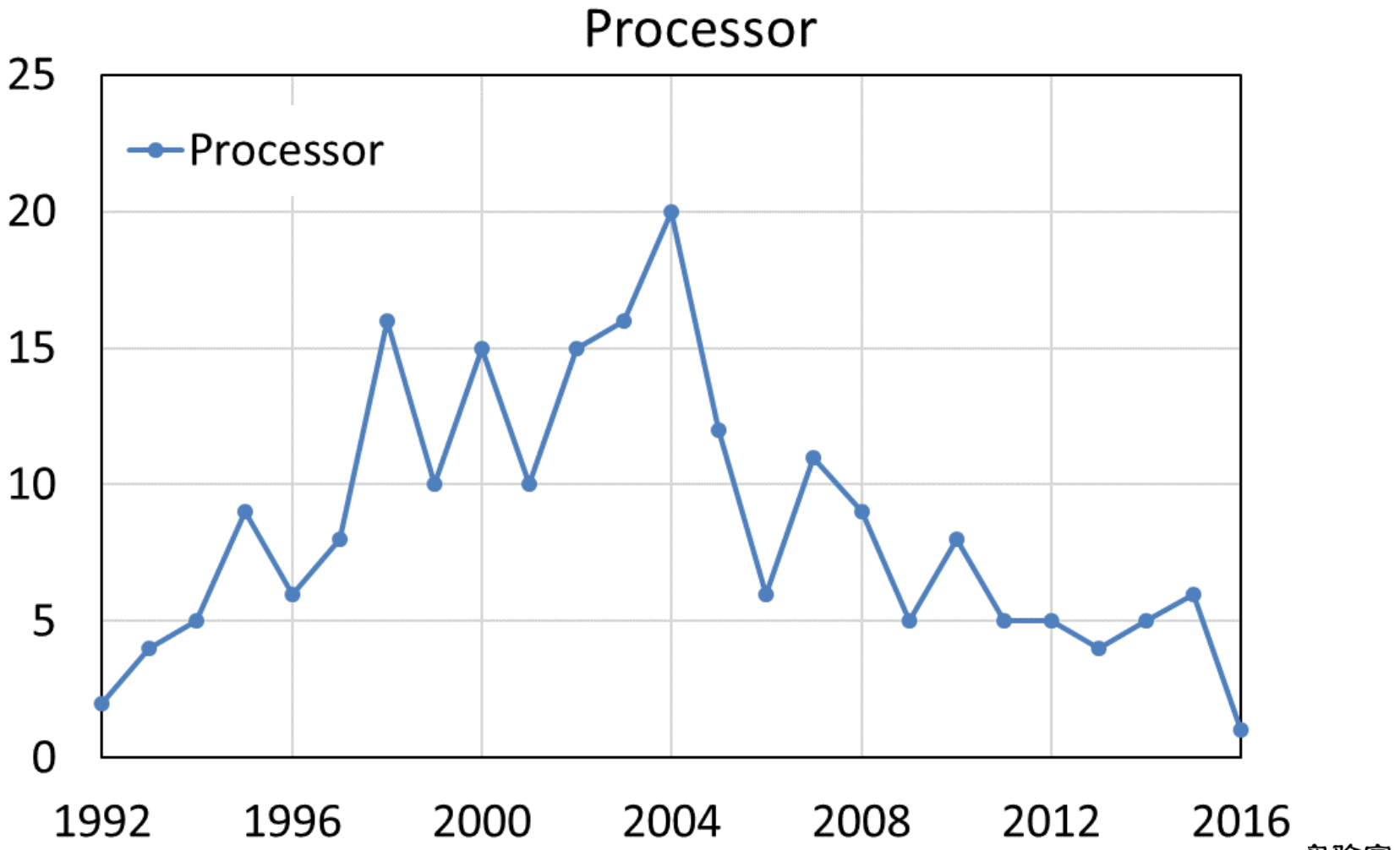
在过去几年，计算机体系结构领域的学者进

世纪的计算机架构》白皮书，影响了学术界和工业界的项目资助计划。最近，《IEEE重启计算倡议》一文又从体系结构、器件和电路等方面探索计算系统的未来。

本文将努力延续这一话题的讨论，深入接触应用和器件/电路学界，了解他们的趋势和愿景。我

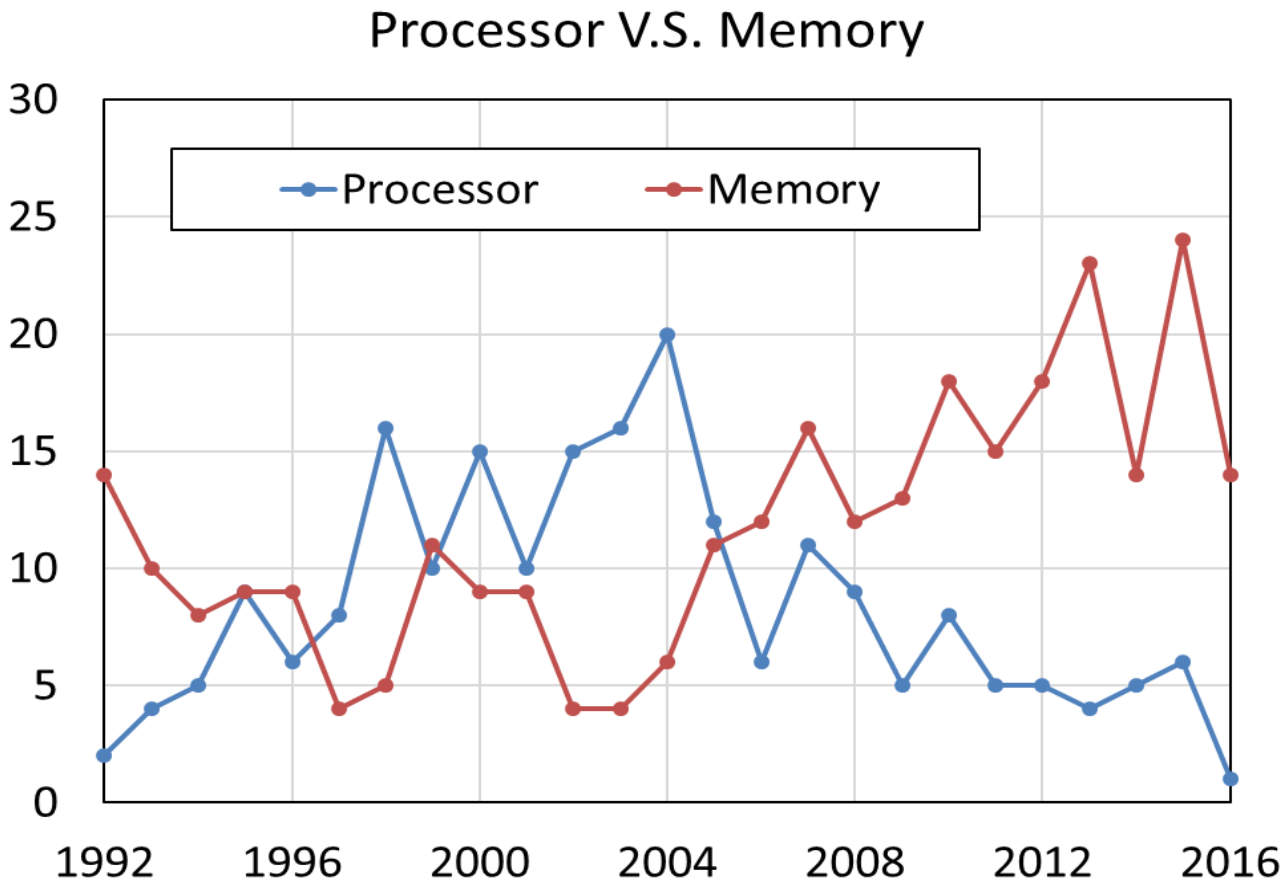
嵌入式系统实验室
Embedded System Laboratory
Suzhou Institute for Advanced Study of USTC

ISCA 1992-2016



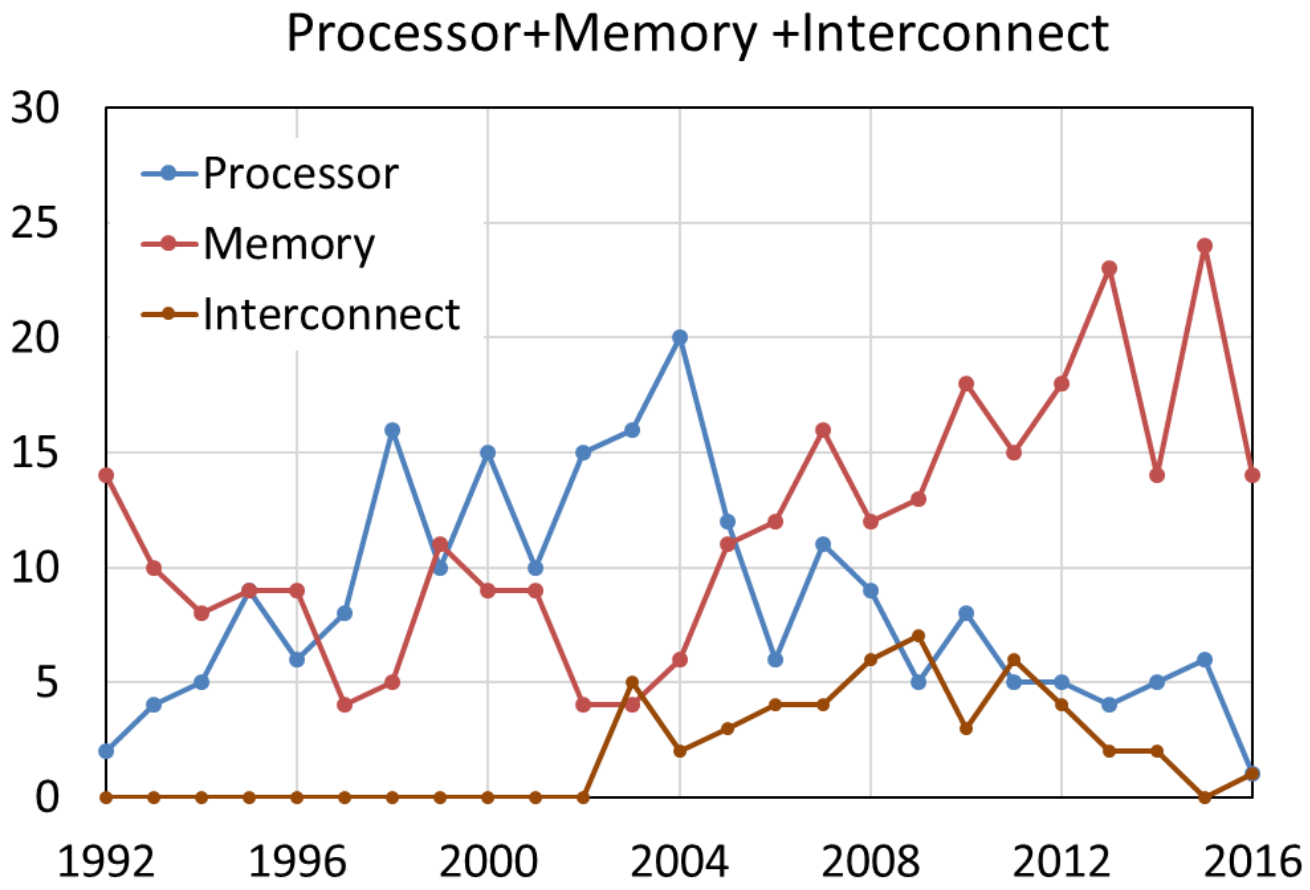
Source：Prof. Yuan Xie (UCSB)

ISCA 1992-2016



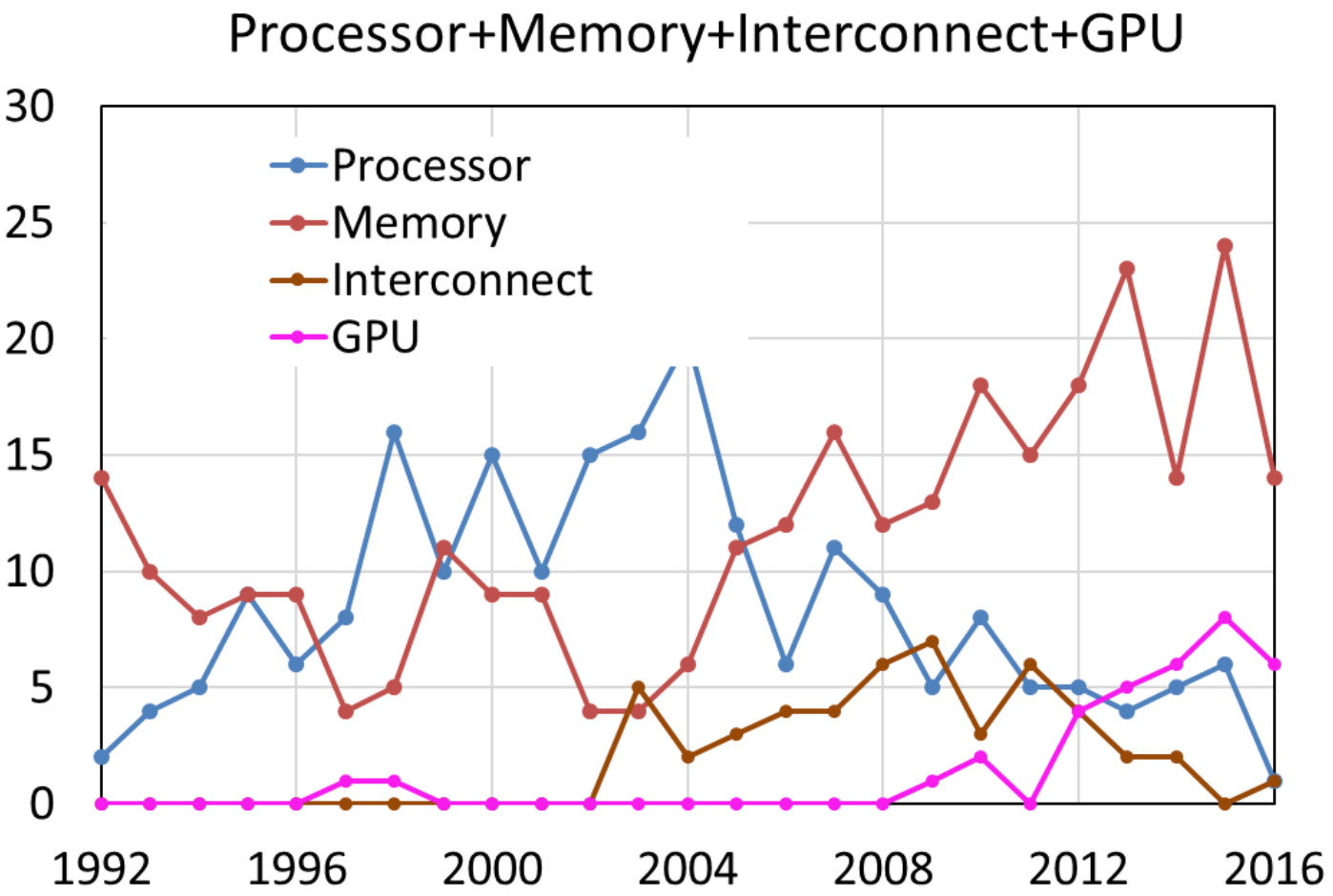
Source：Prof. Yuan Xie (UCSB)

ISCA 1992-2016



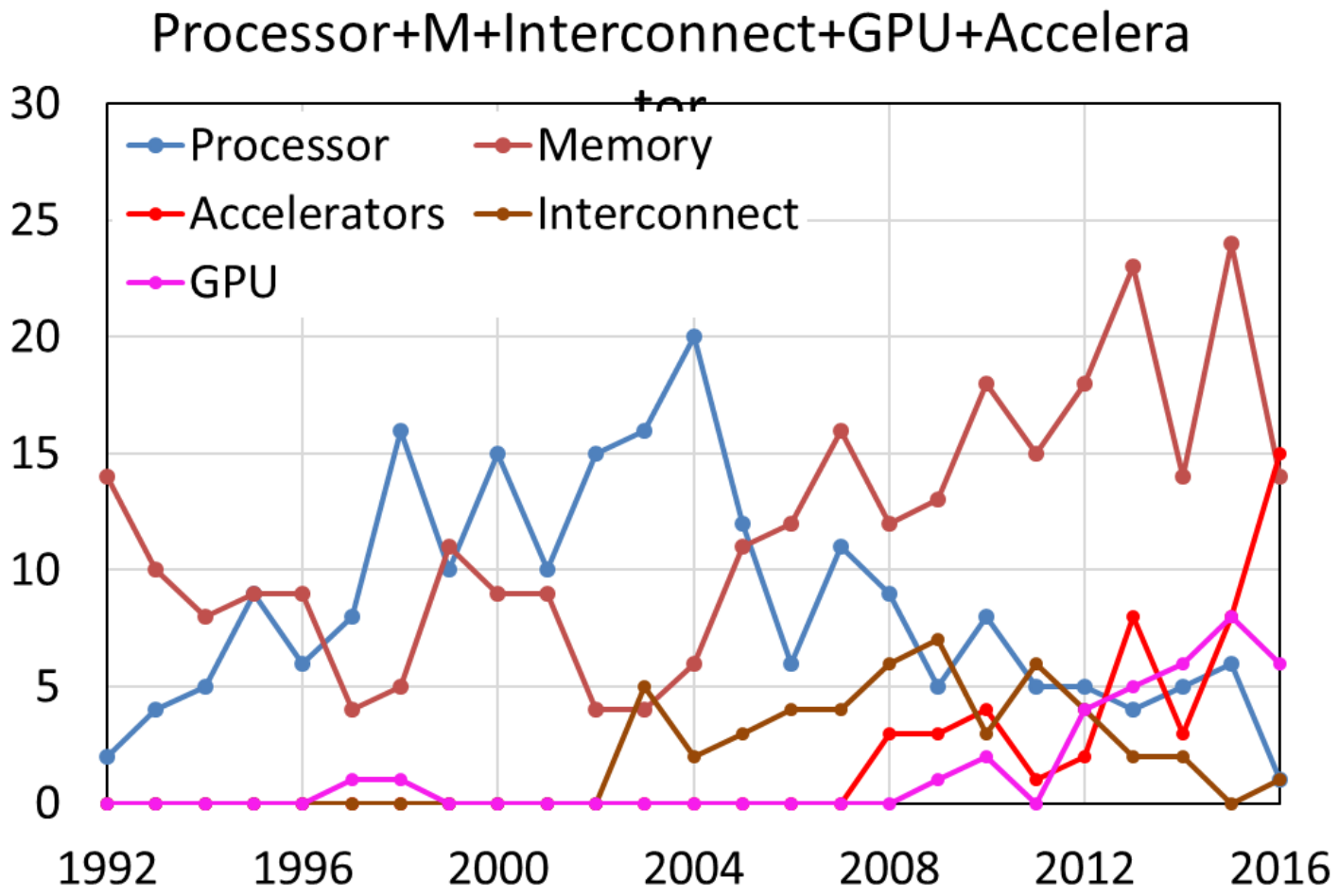
Source: Prof. Yuan Xie (UCSB)

ISCA 1992-2016



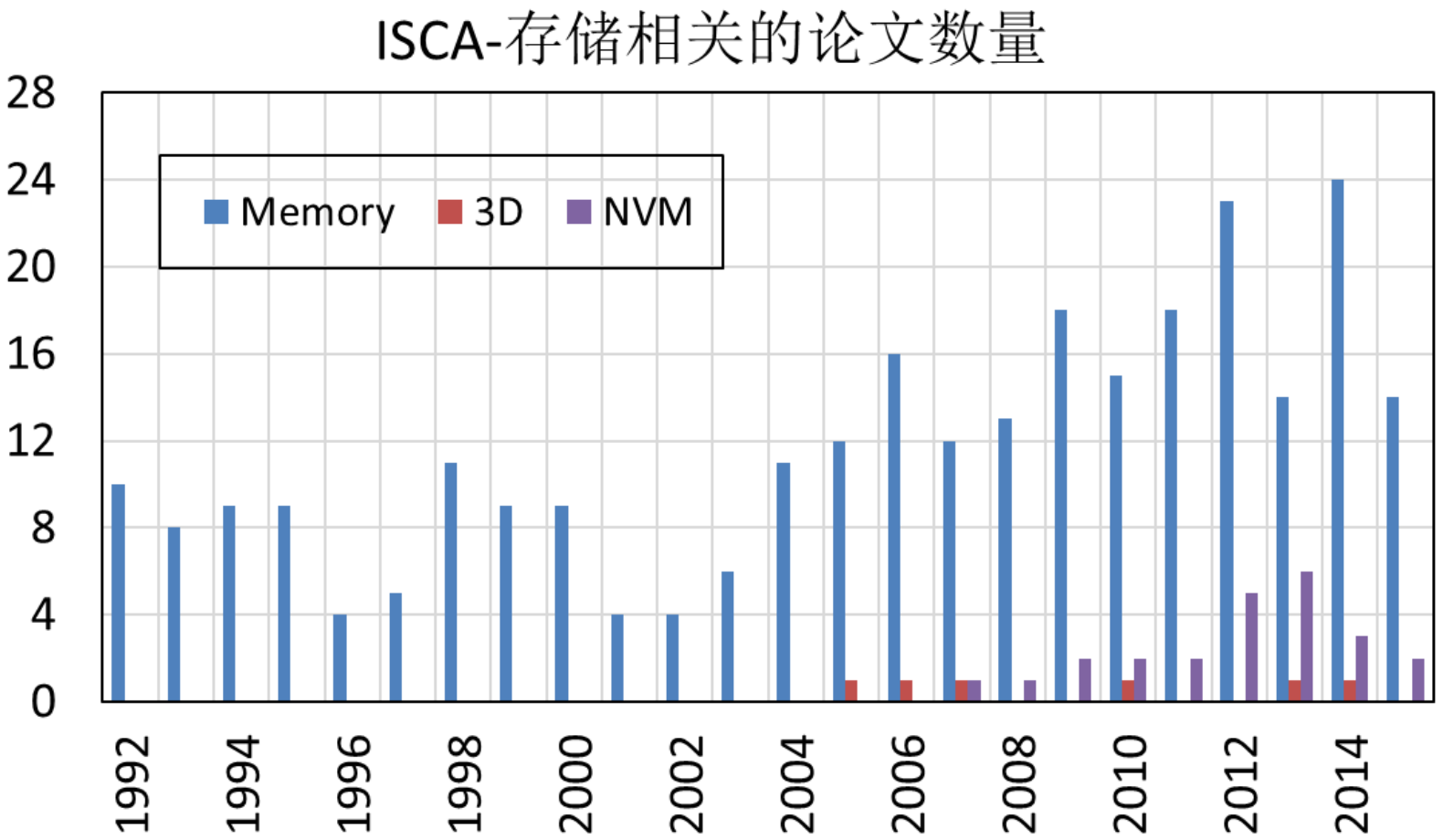
Source: Prof. Yuan Xie (UCSB)

ISCA 1992-2016



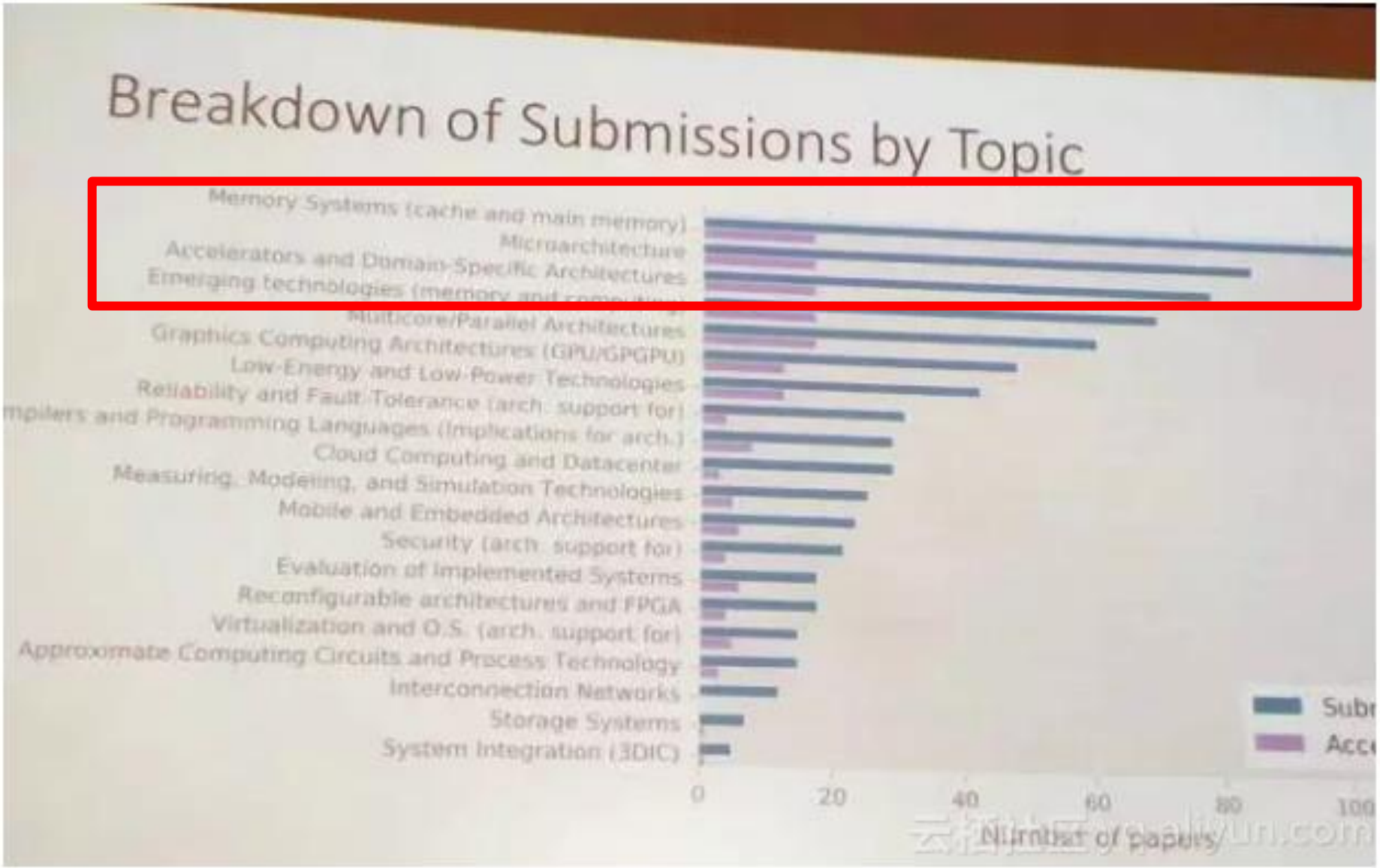
Source: Prof. Yuan Xie (UCSB)

ISCA 1992-2016

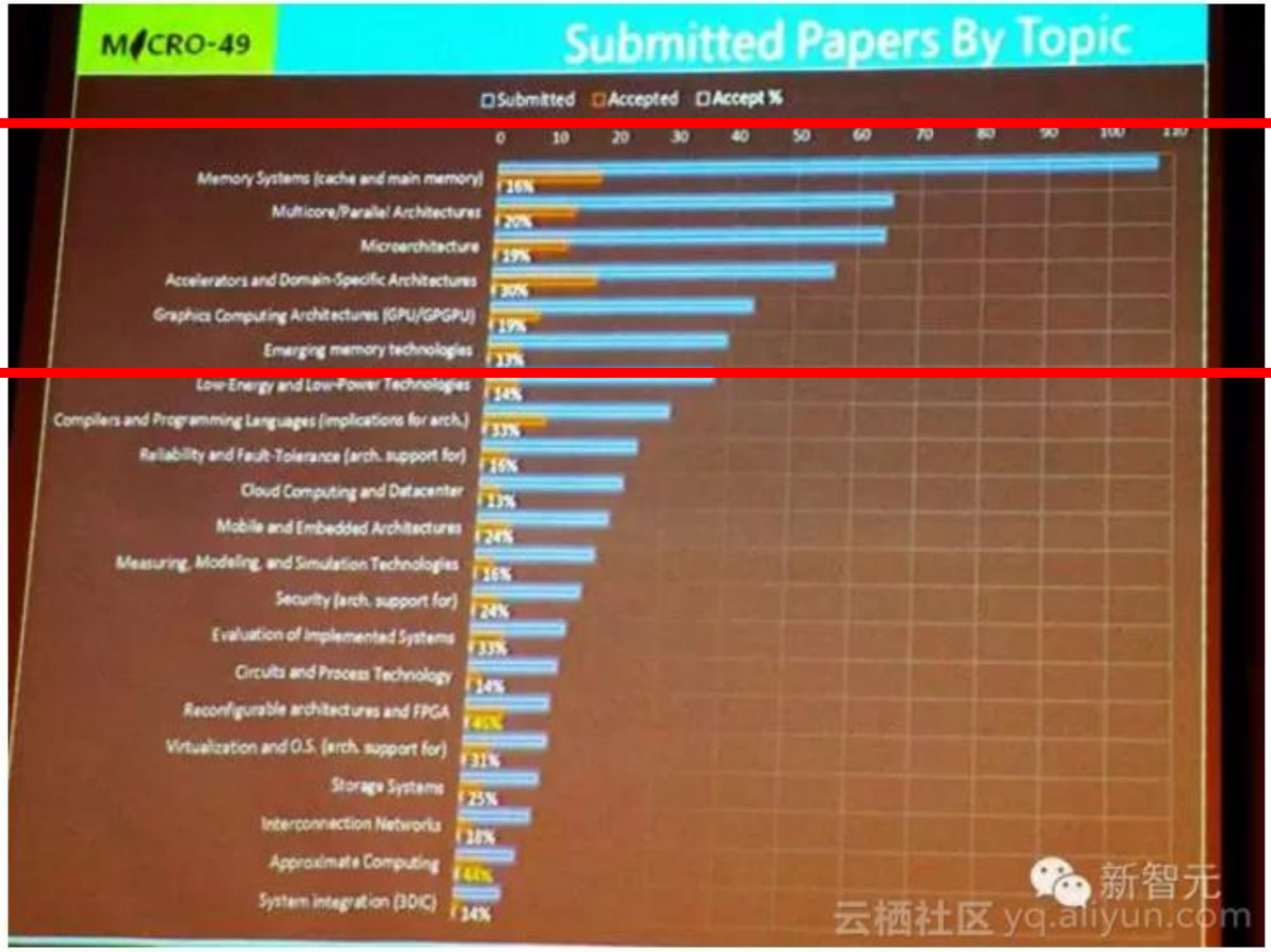


Source：Prof. Yuan Xie (UCSB)

MICRO 2017


Breakdown of Submissions by Topic
Memory Systems (cache and main memory)
Microarchitecture
Accelerators and Domain-Specific Architectures
Emerging technologies (memory and
Multicore/Parallel Architectures
Graphics Computing Architectures (GPU/GPGPU)
Low-Energy and Low-Power Technologies
Reliability and Fault Tolerance (arch. support for)
mpilers and Programming Languages (Implications for arch.)
Cloud Computing and Datacenter
Measuring, Modeling, and Simulation Technologies
Mobile and Embedded Architectures
Security (arch. support for)
Evaluation of Implemented Systems
Reconfigurable architectures and FPGA
Virtualization and O.S. (arch. support for)
Approximate Computing Circuits and Process Technology
Interconnection Networks
Storage Systems
System Integration (3DIC)
0
20
40
60
80
100
Number of papers

MICRO 2016



主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构
2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉
3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例
4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法
5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

本章主要参考书：
唐朔飞《计算机组成原理》、白中英《计算机组成原理》、COD5

1.1 存储器的分类

- 存储器
  - ✓ 计算机系统中的记忆设备，用于存放程序和数据
- 存储器的分类
  - ✓ 按存储介质分类：半导体存储器、磁存储器和光盘存储器等
  - ✓ 按存取方式分类：随机存储器和顺序存储器
  - ✓ 按存储内容可变性分类：只读存储器和读写存储器
  - ✓ 按信息易失性分类：易失性存储器和非易失性存储器
  - ✓ 按在计算机系统中的作用分类：
    - 高速缓冲存储器
    - 主存储器
    - 辅助存储器

1.1 存储器的分类（2）

- 存储器
  - 主存储器
    - 随机存储器（RAM）
      - 静态RAM(SRAM)
      - 动态RAM(DRAM)
    - 只读存储器（ROM）
      - 掩膜式MROM
      - 可编程式PROM
      - 可擦写式EPROM
      - 电擦写式EEPROM
  - 缓冲存储器
  - 辅存储器
    - 磁盘
    - 磁带
    - 光盘
  - 快擦型存储器Flash Memory(NAND,NOR)

新型存储器 MRAM(磁存储器),FeRAM（铁电存储器）,STT-RAM(自旋存储器)，OXRAM(氧化物电阻存储器)，ReRAM(阻变式存储器),PCM(相变存储器)，Memristor（忆阻器）

半导体存储器的类型和特点

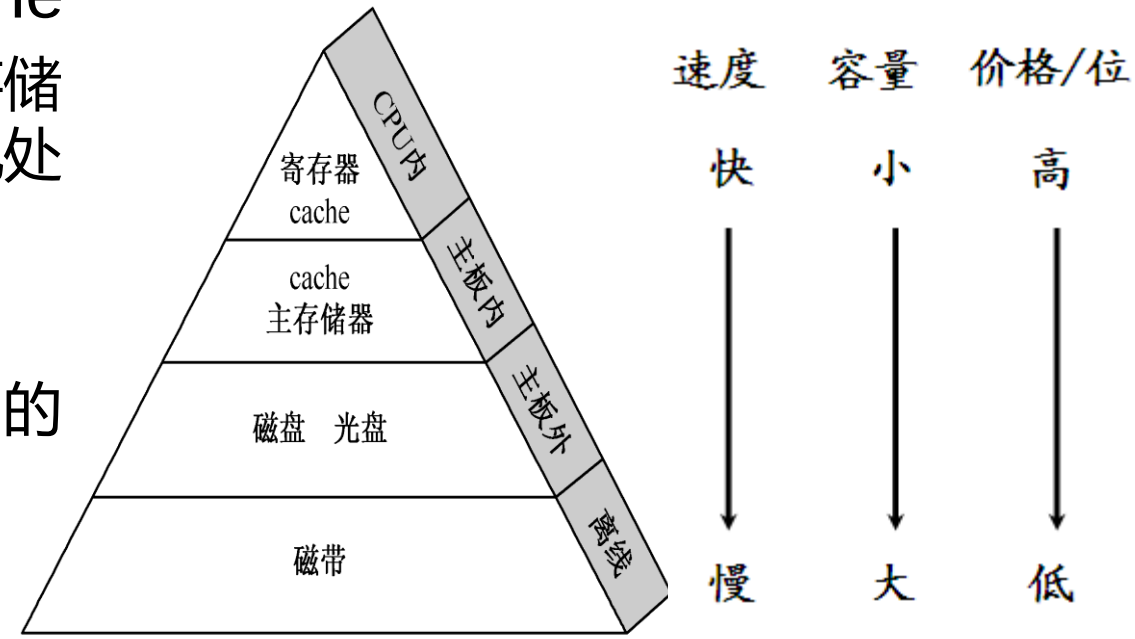
存储器类型	种类	可擦除性	写机制	易失性
随机存储器RAM	读写存储	电，字节级	电	易失
只读存储器ROM	一次编程只读存储	不能	掩膜	非易失
可编程PROM			电	
光擦可编程EPROM	多次编程只读存储	紫外线，字节级		
电擦可编程EEPROM		电，字节级		
闪速存储器		电，块级		

新型存储器 MRAM(磁存储器),FeRAM（铁电存储器）,STT-RAM(自旋存储器),OXRAM(氧化物电阻存储器)，ReRAM(阻变式存储器),PCM(相变存储器)，Memristor（忆阻器）

课后思考：调研新型存储器的典型特征，如原理、密度、易失性、速度、容量等

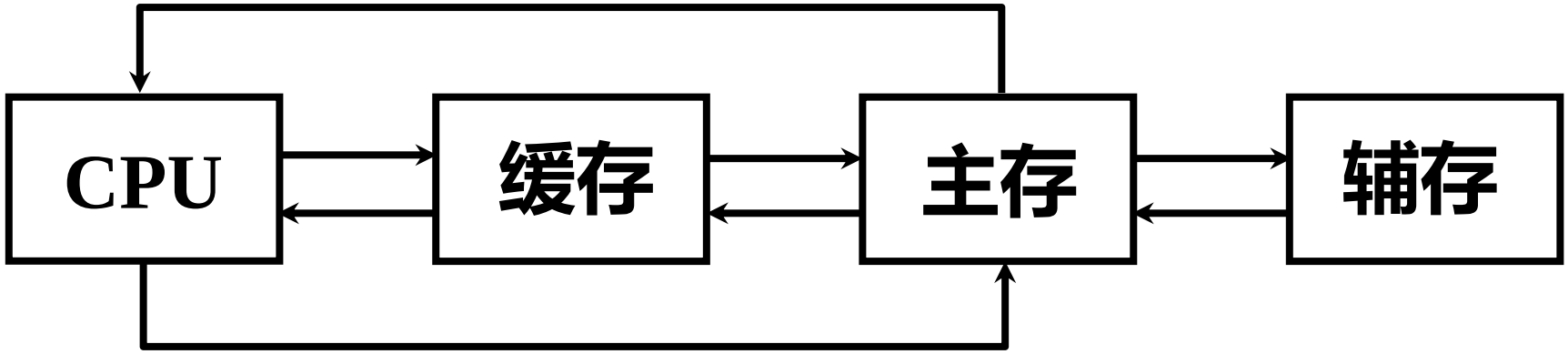
1.2 存储器的层次结构

- 多级存储器体系结构
  - ✓高速缓冲存储器cache
    - 高速小容量半导体存储器，用于提高计算机处理速度
  - ✓主存储器
    - 存放计算机运行期间的大量程序和数据
  - ✓外存储器
    - 大容量辅助存储器
      满足计算机的存储需求



1.2 存储器的层次结构（2）

- 缓存-主存层次和主存-辅存层次

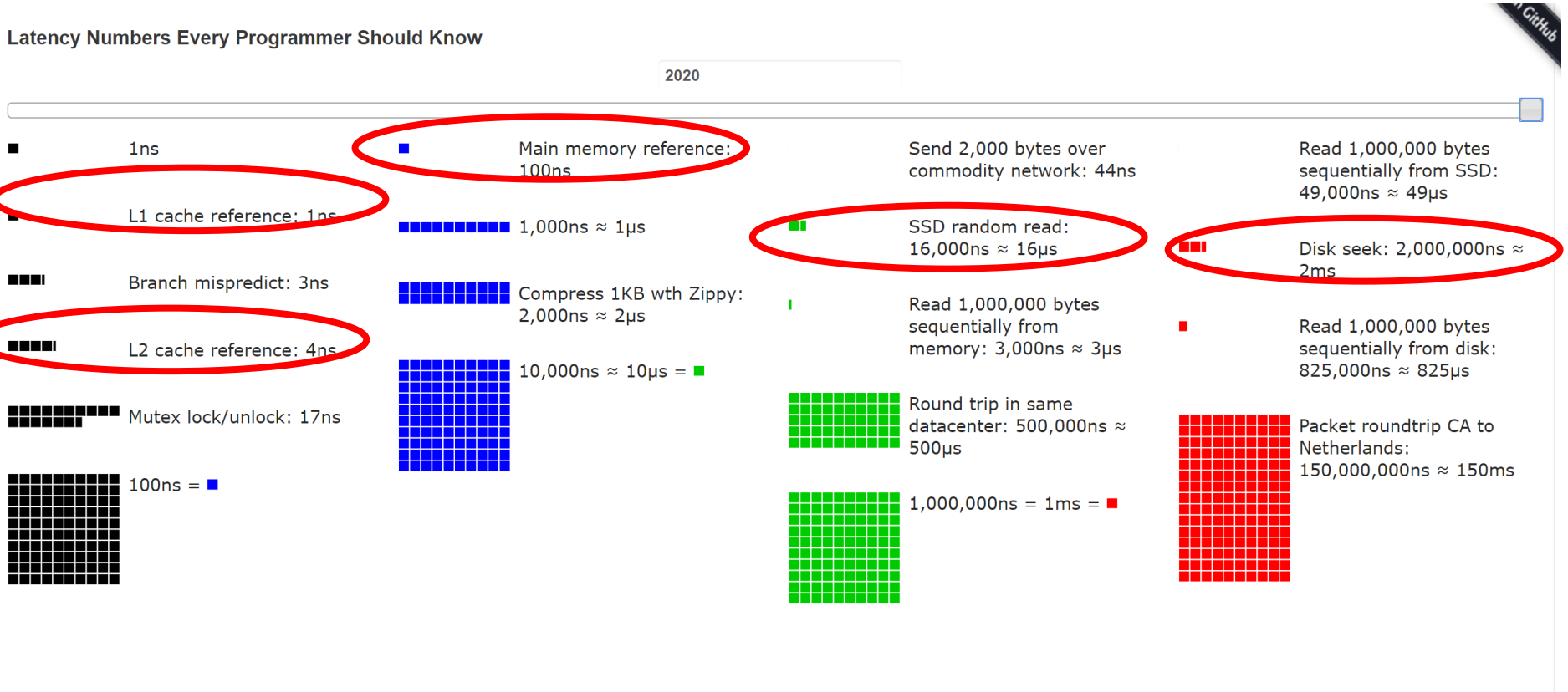


（速度）　　（容量）

缓存－主存　　主存－辅存

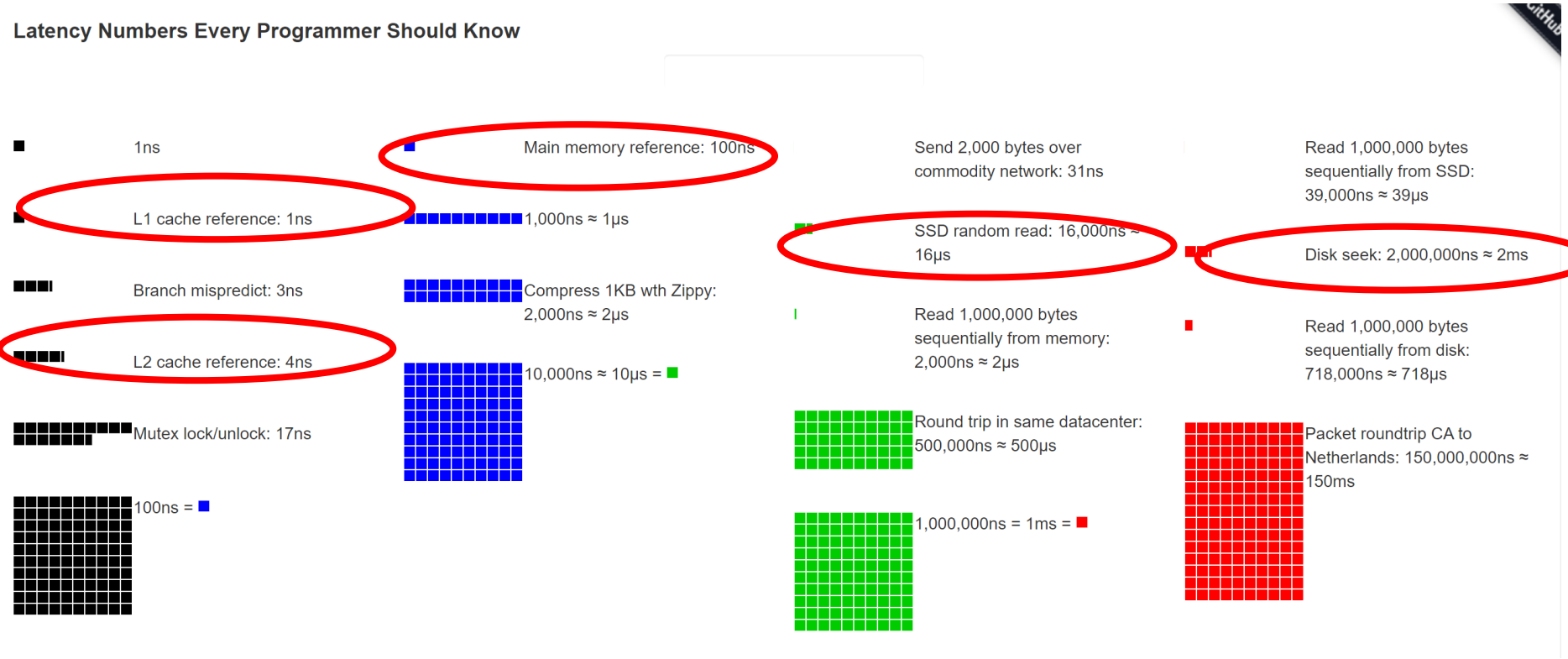
主存储器	虚拟存储器
实地址	虚地址
物理地址	逻辑地址

复习：典型延迟



http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

复习：典型延迟

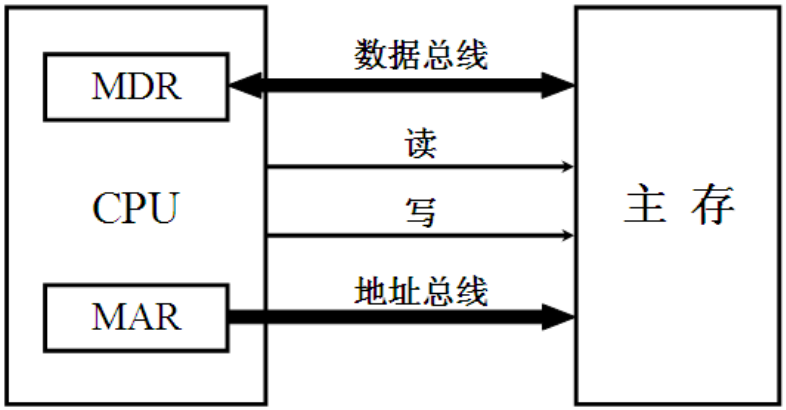


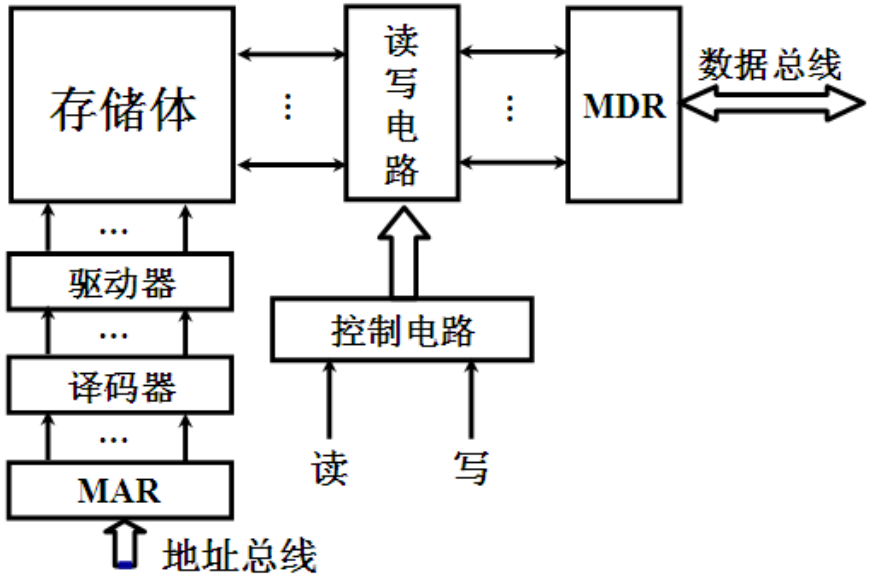
http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构
2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储
3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例
4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法
5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

主存的基本组成





驱动器、地址译码器、读写电路都制作在存储芯片中

MAR MDR在CPU中

2.1 主存储器概述（2）

- 字寻址与字节寻址
  - ✓ 字存储单元——存放一个机器字的存储单元
    字节存储单元——存放一个字节的存储单元
  - ✓ 字地址——字存储单元对应的单元地址
    字节地址——字节存储单元对应的单元地址
  - ✓ 字寻址计算机——计算机中可编址的最小单位是字存储单元
    字节寻址计算机——计算机中可编址的最小单位是字节存储单元

高位字节 地址为字地址

字地址	字节地址			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11

低位字节 地址为字地址

字地址	字节地址	
0	1	0
2	3	2
4	5	4

复习：大尾端、小尾端

◘字节顺序的问题

✓多字节数据的存储，涉及至

大尾端与小尾端 0X0123

字地址	31 24	23 16	15 8	7 0
0	0X0	0X1	0X2	0X3
4				

字地址

0

4

小尾端——低位在低地址

如下数据00000000 00000001 00000010 00000011

小尾端： 00000000 00000001 00000010 000000

addr+3 addr+2 addr+1 addr

//低位在低地址

大尾端： 00000011 00000010 00000001 0000000

addr+3 addr+2 addr+1 addr+0

//高位在低地址

```
#include <stdio.h>
int main()
{
    int tt = 1;
    char *c = (char*)(&tt);
    if(*c == 1)
    {
        printf(      endian\n");
    }
    else
    {
        printf(      endian\n");
    }
    return 0;
}
```

2.1 存储器的层次结构（3）

- 主存储器的技术指标
  - ✓ 存储容量
    - 一个存储器中存储单元的总数，一般用字数(W)或字节数(B)表示
  - ✓ 存取时间
    - 存储器访问时间，指一次读操作命令发出到该操作完成，将数据送到数据总线上所经历的时间
    - 写操作时间通常等于读操作时间
  - ✓ 存储周期
    - 连续两次读写操作所需的最小间隔时间
    - 存储周期一般大于存取时间
  - ✓ 存储器带宽
    - 单位时间内存储器存取的信息量，用字/秒、字节/秒或位/秒表示
    - 衡量数据传输速率的重要指标，决定了以存储器为中心的计算机系统获得信息的速度，是改善机器性能瓶颈的一个关键因素

提高存储器带宽的措施有：

1）缩短存取时间、存储周期

2）增加存储字长，使每个存取周期可读写更多二进制位数

3）增加存储体

复习：评价指标

- CPU时间
  - ✓CPI
  - ✓IC
  - ✓主频
- 加速比（Amdahl定律）
  - ✓部件加速比
  - ✓可改进比例
- 流水线
  - ✓吞吐率
  - ✓加速比
  - ✓效率

指令字长
机器字长
存储字长

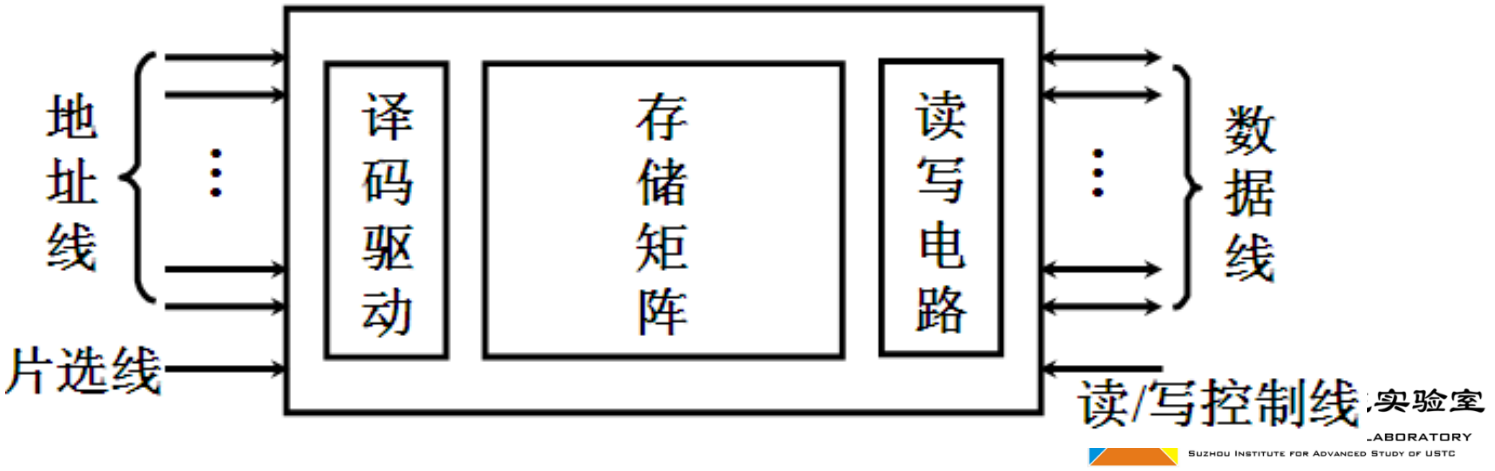
存储器带宽
总线带宽

机器周期
时钟周期
指令周期

2.2 存储芯片简介

半导体存储芯片的基本结构

- ✓ 采用超大规模集成电路制造工艺，在一个芯片内集成具有记忆功能的存储矩阵、译码驱动电路和读/写电路等
- ✓ 译码驱动
  - 把地址总线送来的地址信号翻译成对应存储单元的选择信号
- ✓ 读/写电路
  - 包括读出放大器和写入电路，用于完成读/写操作



2.2 半导体存储芯片简介（2）

- 存储芯片的连线
  - ✓存储芯片通过地址总线、数据总线和控制总线与外部连接
  - ✓地址线
    - 单向输入，其位数与芯片存储单元的数量有关
  - ✓数据线
    - 双向输入/输出，其位数与芯片可一次读出或写入的数据位数有关

地址线和数据线的位宽，共同反映了存储芯片的容量

例如，地址线10根，数据线4根，则表示存储芯片的存储单元数为2^{10}，每个单元的数据宽度为4位，则其芯片容量为$2^{10} \times 4b = 4Kb$

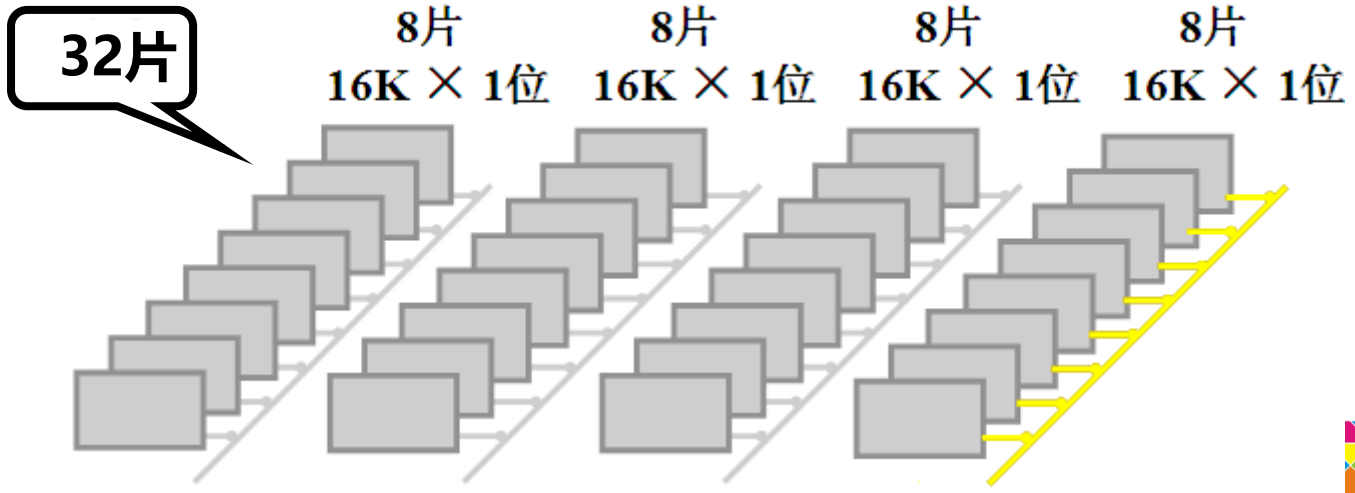
2.2 半导体存储芯片简介（3）

✓控制线：主要有读/写控制线和片选线两种

- 读/写控制线：决定芯片进行读/写操作
  可以读、写分开两根线（6264），也可以合用一根线（2114）
- 片选线：用来在存储矩阵中选择存储芯片，可能有多根线

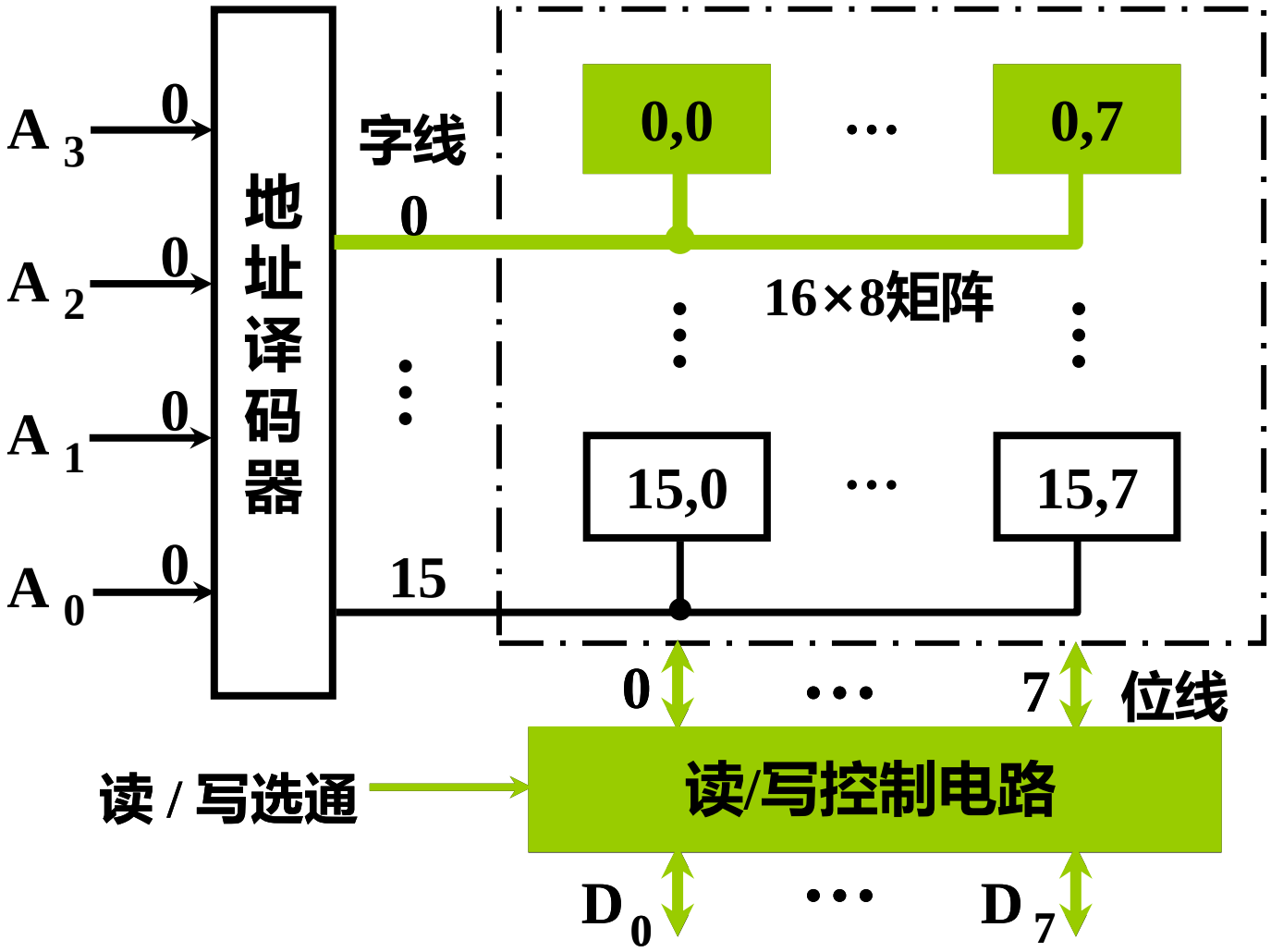
存储芯片片选线的作用

用 16K × 1位 的存储芯片组成 64K × 8位 的存储器

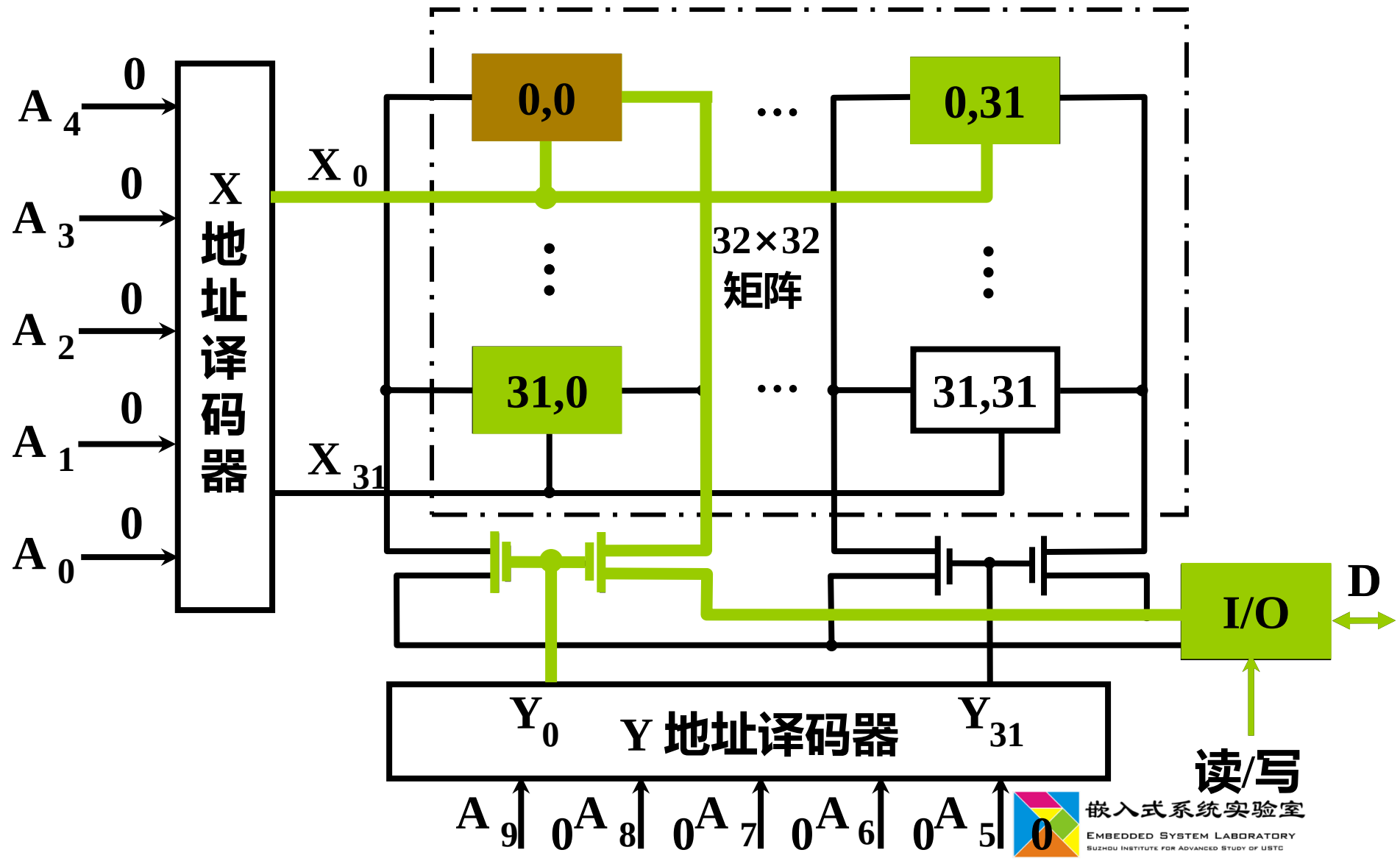


进行一次读/写时，将同时选中8片存储芯片，从每个存储芯片的同一地址单元取出/写入1位数据，组成一个8位的数据

半导体存储芯片的译码驱动方式——线选法

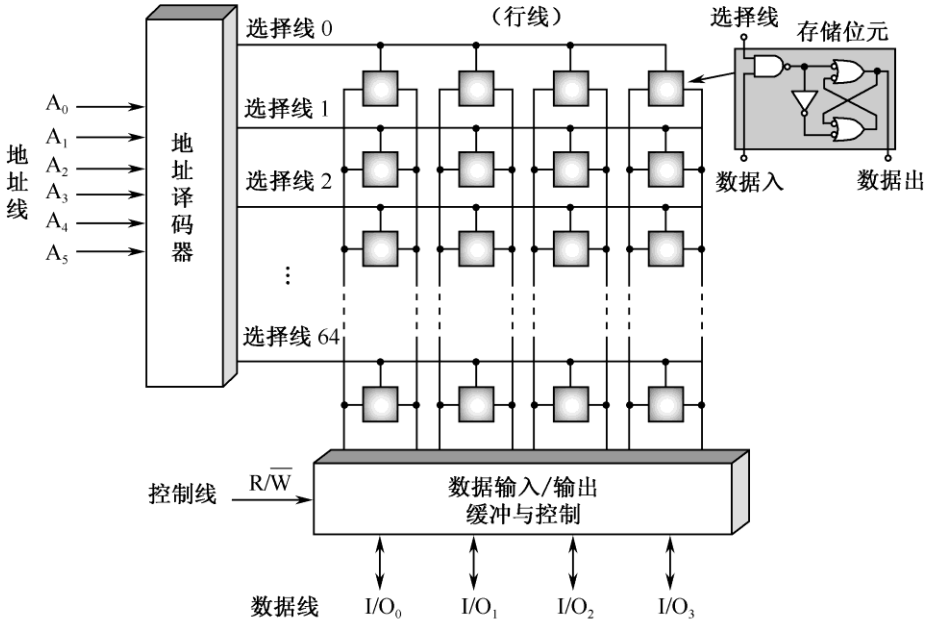


半导体存储芯片的译码驱动方式——重合法



2.3 SRAM随机存储器

- 基本的静态存储元阵列
  - ✓用**锁存器**作为存储元——SRAM的特征
  - ✓**易失性**存储：不断电则持久保存信息
  - ✓信号线
    - 地址线 A_0~A_5
    - 数据线 I/O_0~I/O_3
    - 控制线 R/$\overline{W}$，读写不会同时发生
    - 64条地址选择线，打开每个存储元的输入与非门



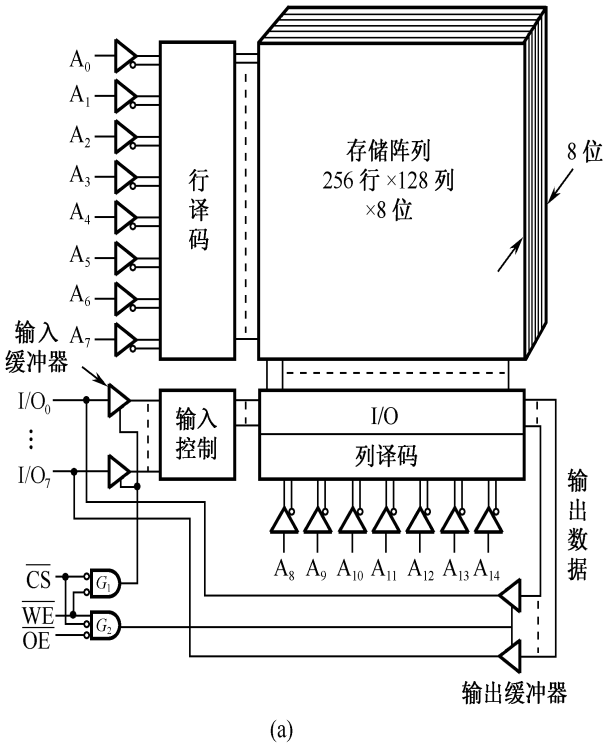
2.3 SRAM随机存储器（2）

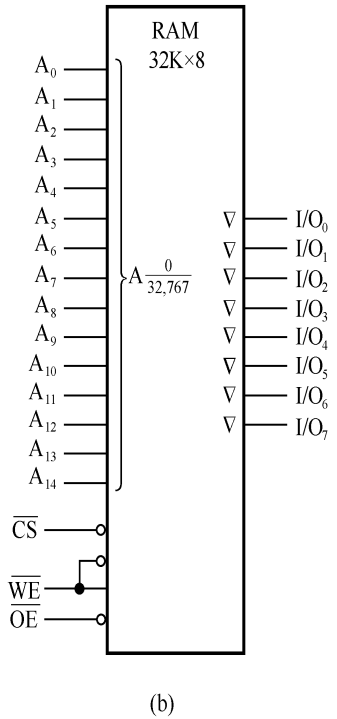
◘基本的SRAM逻辑结构

✓重合法双译码，便于扩展组织更大的存储容量

- **32K×8位的SRAM**
  - **8块存储芯片组成**
  - **每片有32K个存储单元，排成256×128的矩阵**
  - **每个存储单元保存1b**
  - **一个字节（8b）数据，保存在8块芯片的同一地址单元**

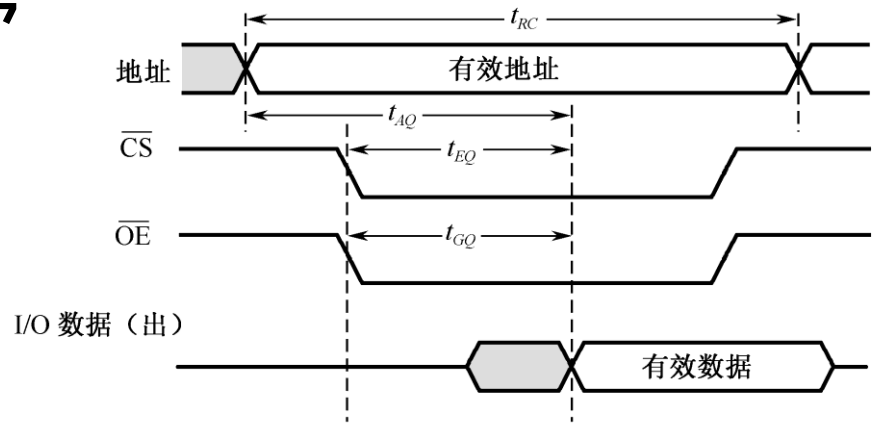
- **地址译码**
  - **15条地址线（32K = 2^{15}）**
  - **采用双译码方式**
  - **A0～A7，行地址译码**
    A8～A14，列地址译码



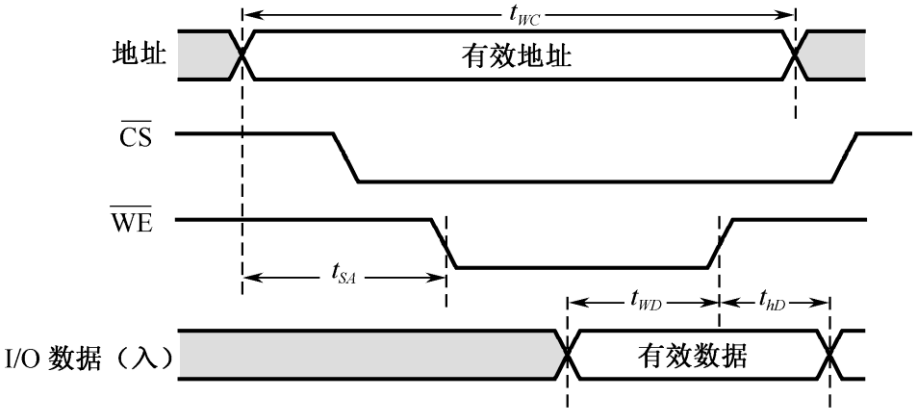


SRAM的读写周期时序

- ✓读周期
  - 读出时间t_{AQ}
  - 读周期时间t_{RC}
- ✓写周期
  - 写入时间t_{WD}
  - 维持时间t_{hD}
  - 写周期时间t_{WC}
- ✓存取周期
  - 读周期t_{RC} = 写周期t_{WC}



(a) 读周期 ($\overline{WE}$ 高)

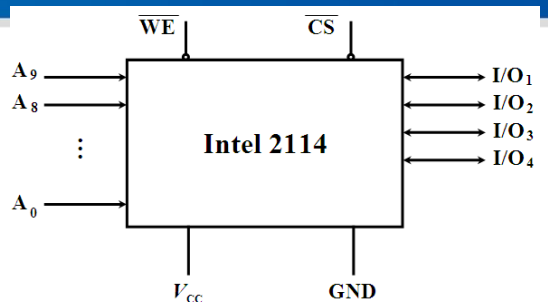


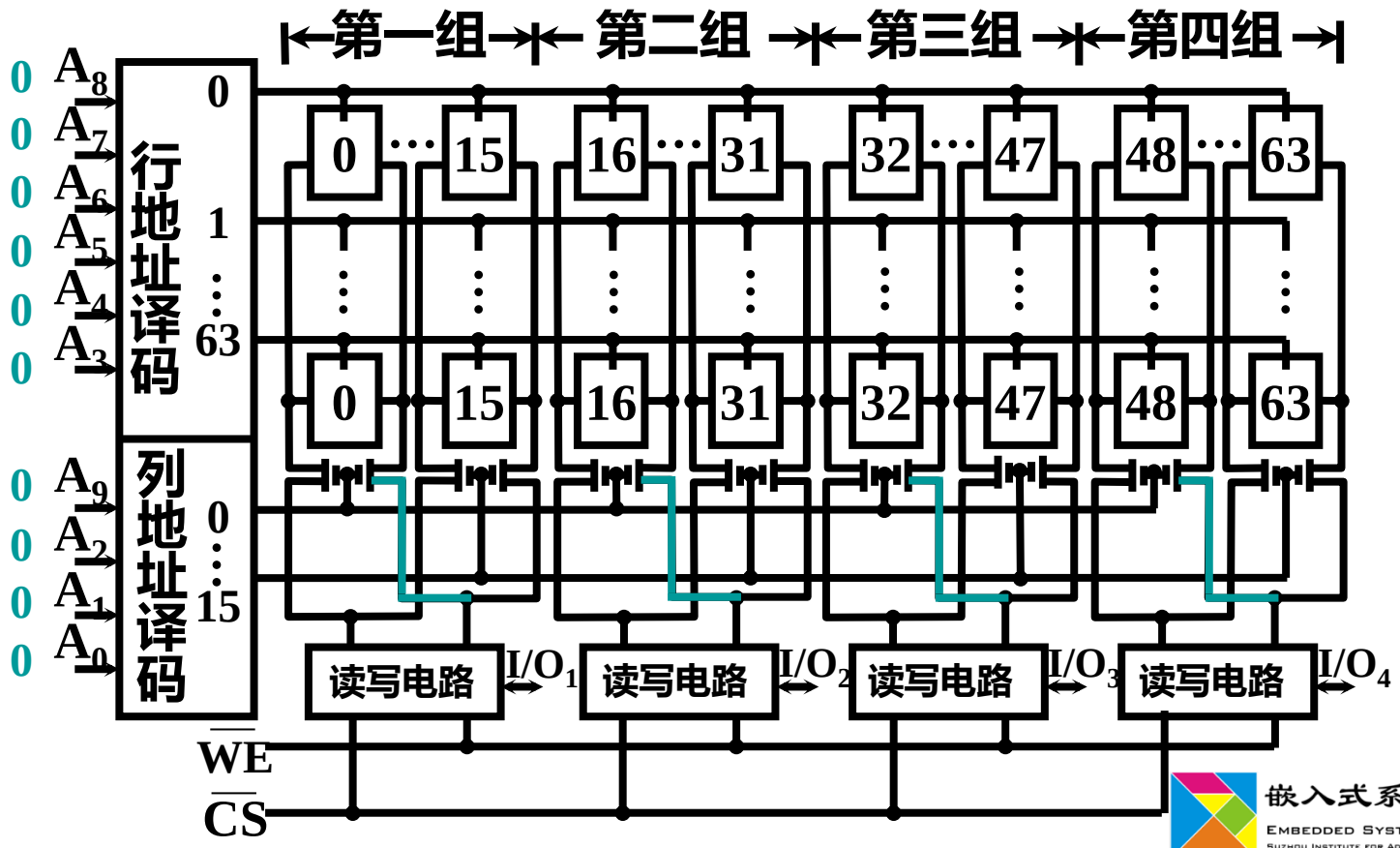
(b) 写周期 ($\overline{WE}$ 低)

CS，片选信号　OE，读出使能信号　WE，写入使能信号

SRAM芯片示例

- Intel 2114 SRAM芯片
  - ✓1K×4b的2114外特性
  - ✓结构示意（64×64）

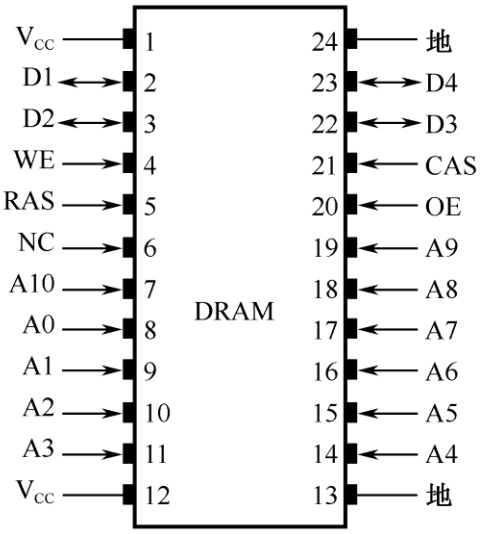




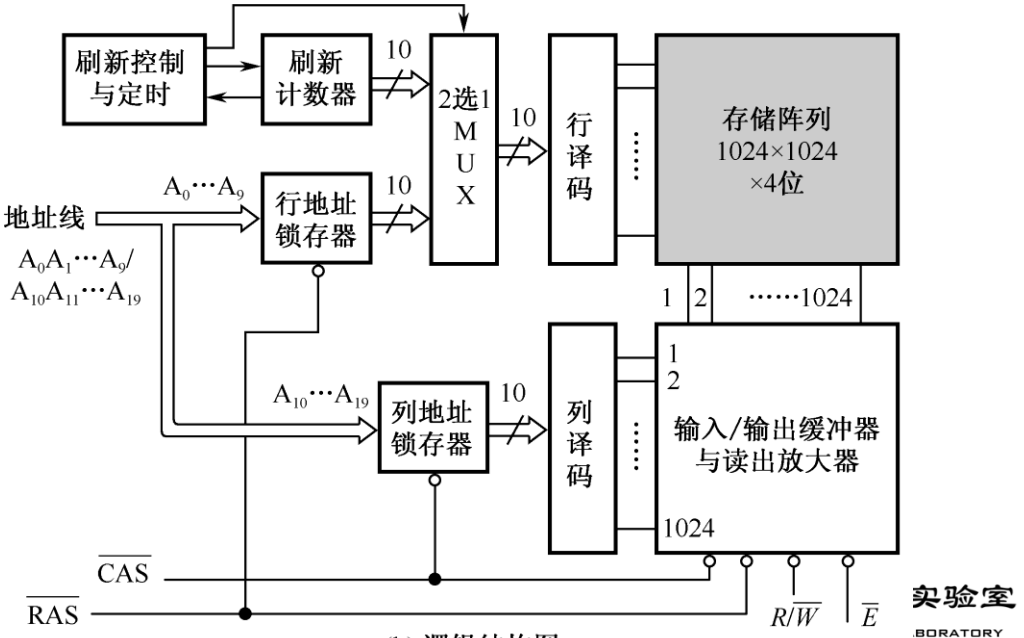
2.4 DRAM随机存储器

DRAM芯片的逻辑结构

- ✓ 由MOS晶体管（开关）和**电容**（信息）组成的记忆电路。**电容信息存1~2ms**
- ✓ 增加了行地址锁存器和列地址锁存器
- ✓ 增加了**刷新计数器**和相应的控制电路



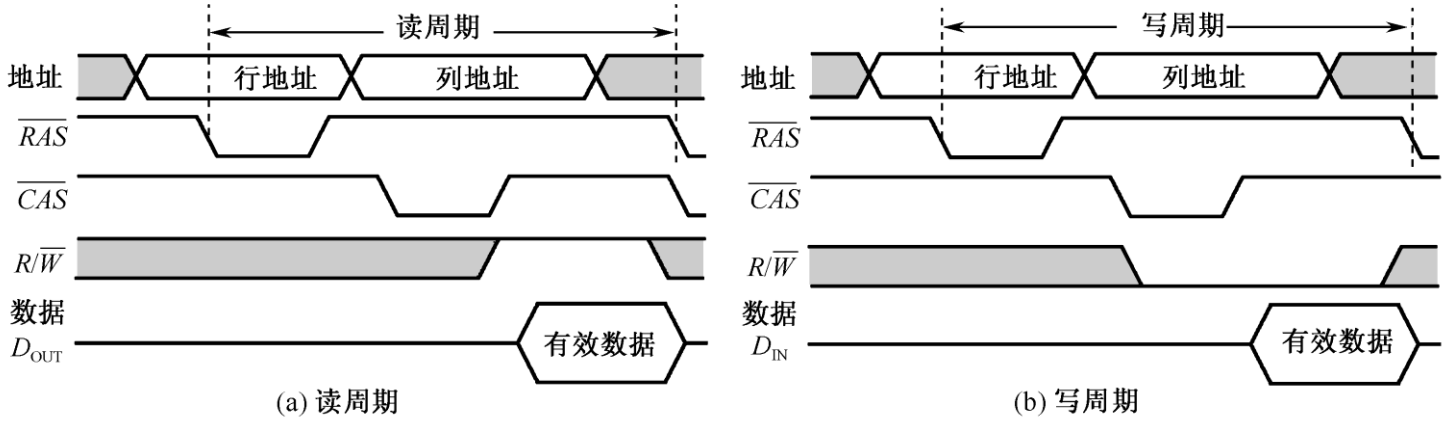
(a) 管脚图



(b) 逻辑结构图

2.4 DRAM随机存储器（3）

- DRAM的读写周期
  - ✓ 读、写周期的界定——从行选信号$\overline{RAS}$的下降沿开始，到下一个$\overline{RAS}$信号的下降沿为止

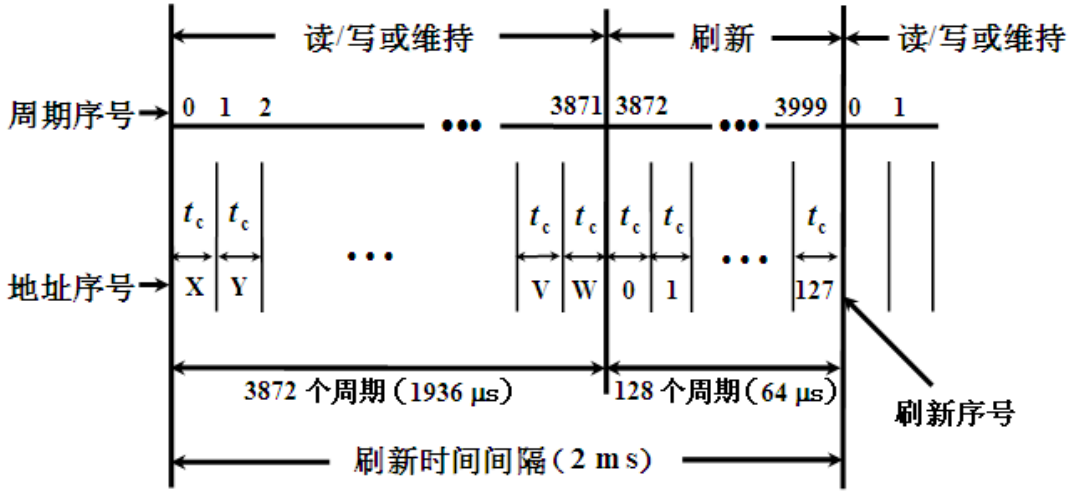


- DRAM的刷新
  - ✓ 过程实质：将原存储信息读出，由刷新放大器形成原信息并重新写入
  - ✓ 刷新机制：在一定时间内，使用专门的刷新电路，对DRAM的全部存储元按行进行一次刷新。刷新间隔一般取2ms，即刷新周期
  - ✓ 刷新方式：集中刷新、分散刷新、异步刷新

DRAM的刷新方式—集中刷新

□集中刷新

✓DRAM的所有行在每一个刷新周期都被刷新，此时必须停止所有的读/写操作



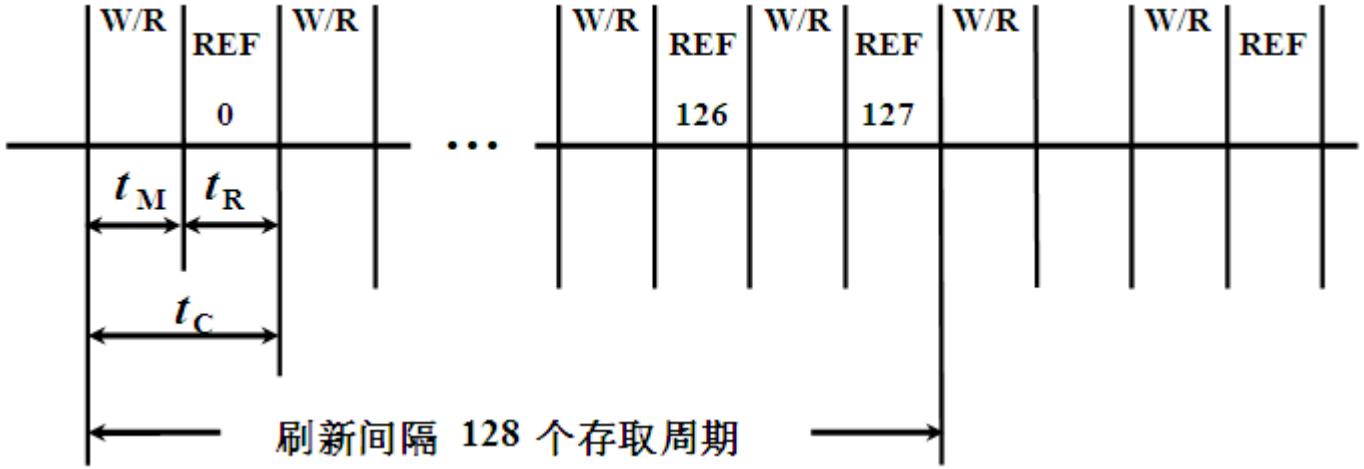
以128×128矩阵的存储芯片为例，存取周期0.5us

“死区”： 0.5us×128 = 64us　　“死时间率”： 128/4000 = 3.2%

DRAM的刷新方式—分散刷新

分散刷新

- ✓对每一行的刷新插入在正常的读/写周期中完成
- ✓不停止读/写操作，但存取周期变长



以128×128矩阵的存储芯片为例

$t_C = t_M + t_R$，无“死区”

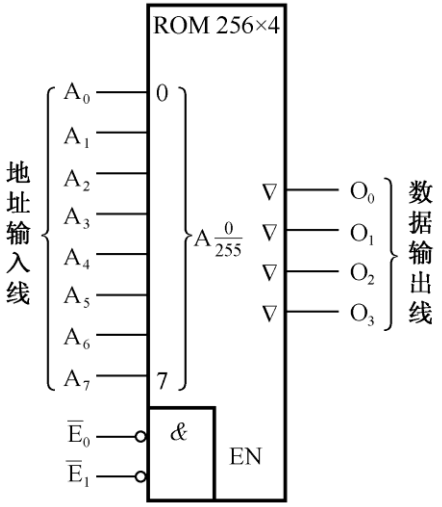
读写+刷新　　**存取周期为0.5us+0.5us = 1us**

DRAM VS. SRAM

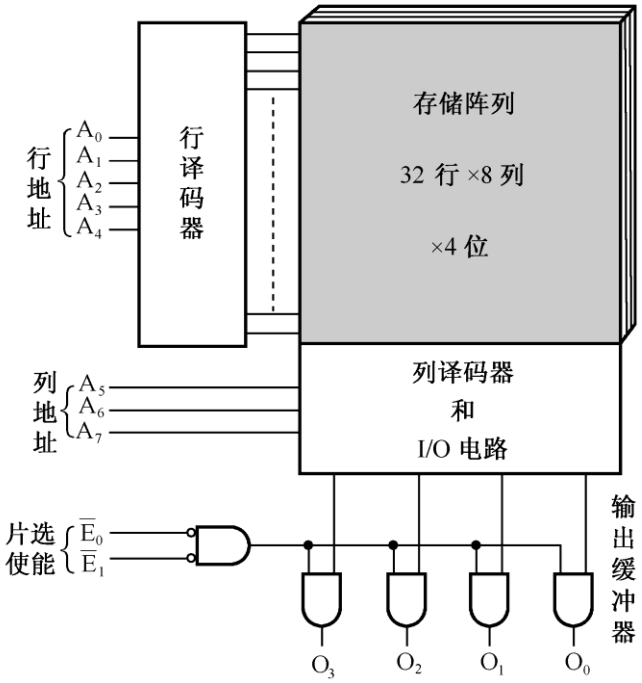
	DRAM	SRAM
存储原理	电容	锁存器
集成度	高	低
芯片引脚	少	多
功耗	低	高
价格	低	高
速度	慢	快
刷新	需要	不需要
应用	主存	缓存

2.5 只读存储器ROM

- ROM（Read-Only Memory）
  - ✓ 存储信息必须在工作前写入；在工作的时候只能读出，不能写入
  - ✓ 类型：掩膜ROM；可编程ROM（PROM、EPROM、E^2PROM）
- 掩膜ROM
  - ✓ 存储内容固定，由生产厂家提供



(a) 掩模 ROM 逻辑符号



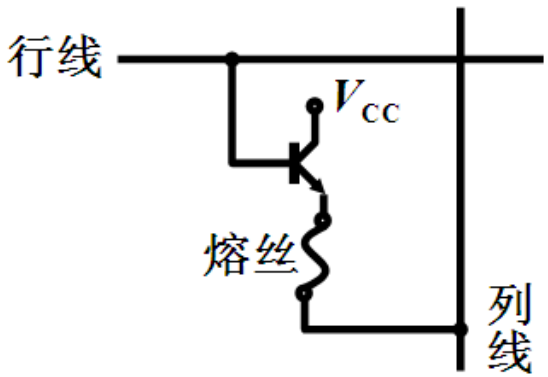
(b) 内部逻辑框图

2.5 只读存储器ROM（2）

可编程ROM

✓ PROM：实现一次性编程的ROM

- 熔丝断，为0；熔丝不断，为1

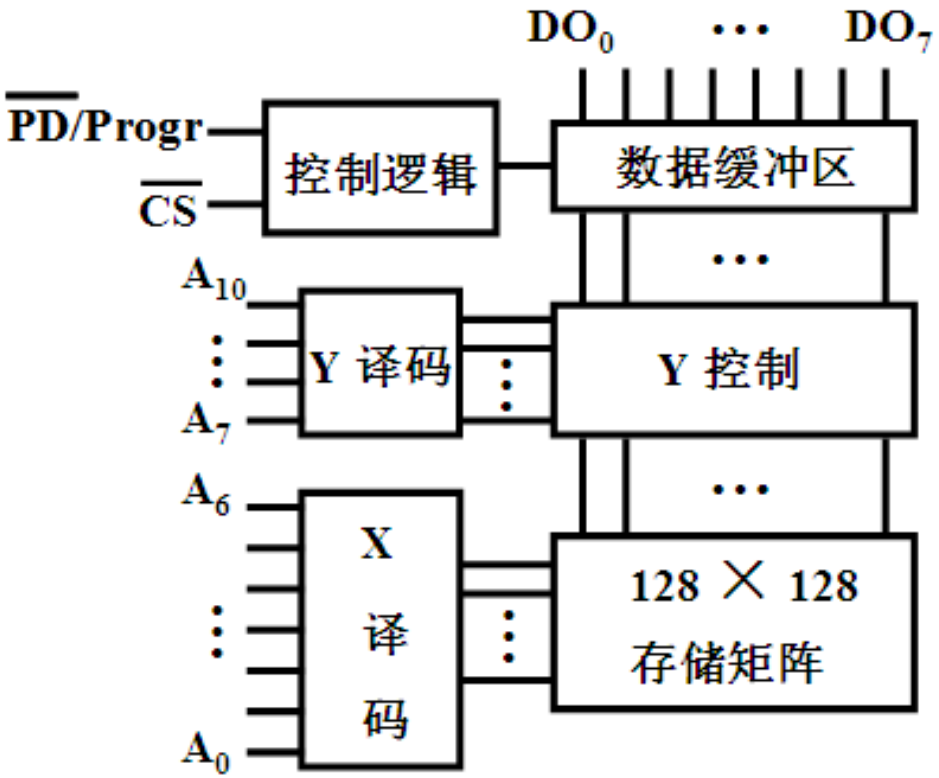


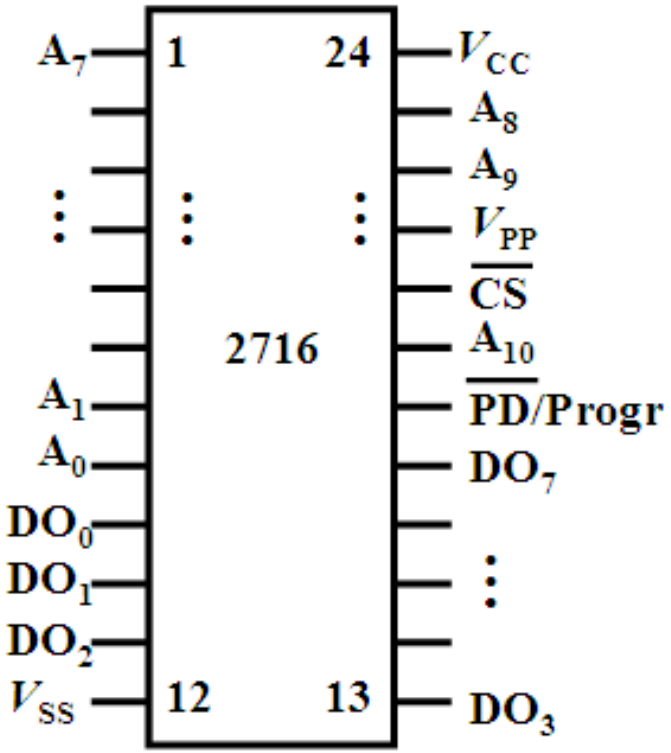
✓ EPROM：可擦除可编程存储器

- 可以对存储的信息作任意次改写
- 擦写方式：
  1）紫外线照射：时间相对较长，全局性擦写
  2）电气方法（E^2PROM）：电擦除，支持全部或局部擦写

EPROM的逻辑图

2716型EPROM芯片





$\overline{PD}$/Progr 功率下降 / 编程输入端　读出时 为 低电平

主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构

2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉

3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例

4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法

5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

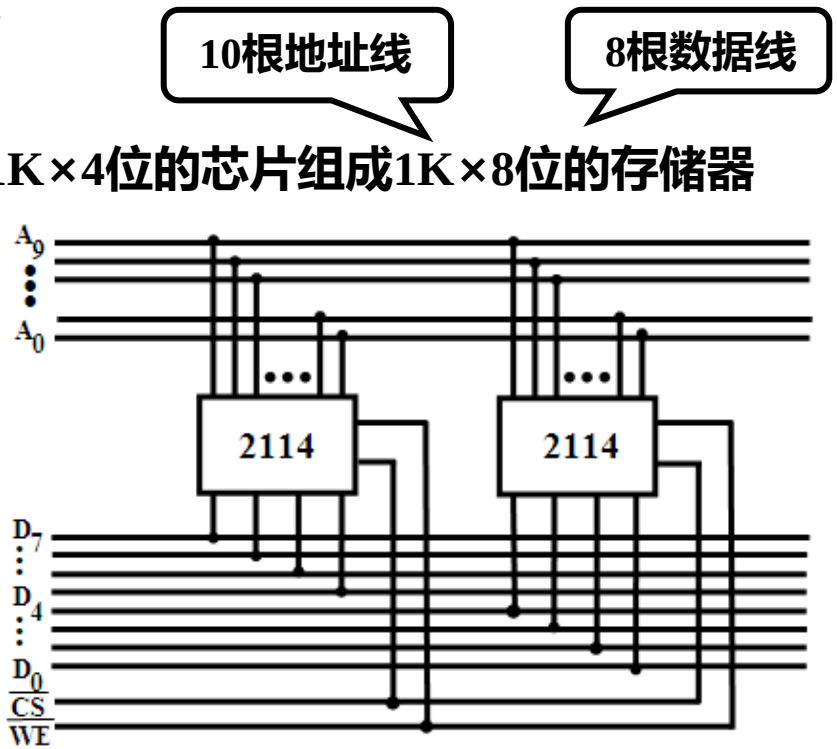
2.6 存储器与CPU的连接

存储容量的扩展

- ✓ 单片存储芯片的容量有限，需要将若干存储芯片连在一起组成具有足够容量的存储器
- ✓ 位扩展：增加存储器的横向容量
- ✓ 字扩展：增加存储器的纵向容量

位扩展

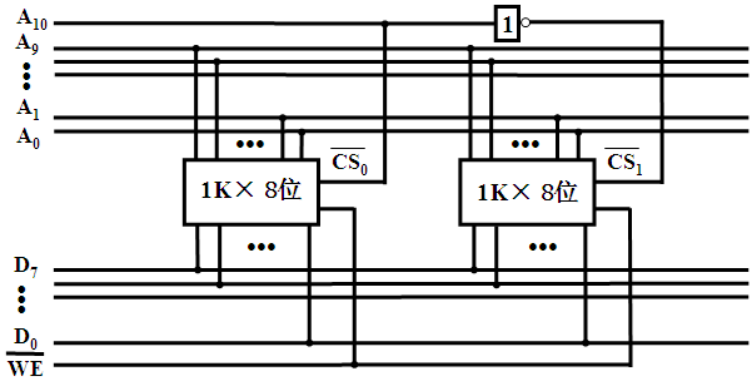
- ✓ 增加存储字长
- ✓ 多个存储芯片的三组信号线关系
  - 地址线和控制线公用
  - 数据线单独分开



2.6 存储器与CPU的连接（2）

字扩展

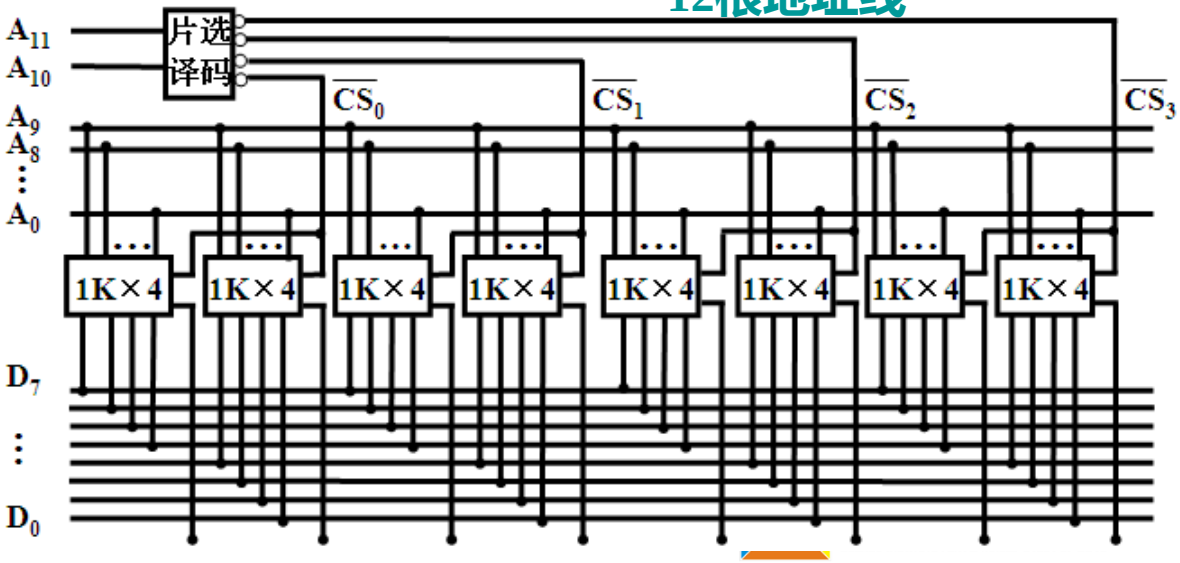
- ✓增加存储单元的数量
- ✓多个存储芯片的三组信号线关系
  - 地址线和数据线公用
  - 读写控制线公用
  - 片选使能控制线独立，通过地址的高位字段进行译码决定



2片1K×8位的芯片组成2K×8位的存储器

字、位扩展

8片1K×4位的芯片组成4K×8位的存储器



2.6 存储器与CPU的连接（3）

- 存储器与CPU的连接
  - ✓存储芯片与CPU芯片相连，主要注意三种信号线的连接
  - ✓地址线的连接
    - CPU的地址线一般比存储芯片的地址线多
    - **CPU地址线的低位线与存储芯片的地址线相连**
      CPU地址线的高位线或用于存储扩展，或用于片选等其他用途
  - ✓数据线的连接
    - **存储芯片的数据线应与CPU的数据线数量相同，若存储芯片数据线不够，需对其进行位扩展**
  - ✓读/写命令控制线的连接
    - **CPU的读/写命令线一般可以直接连到存储芯片的读/写控制端**，通常高电平为读，低电平为写
    - 若读/写命令线分开，则分别连到对应的存储芯片控制端

2.6 存储器与CPU的连接（4）

- 片选控制线的连接
  - ✓ CPU与存储芯片正确工作的关键
  - ✓ 片选有效信号与CPU的**访存控制信号MREQ**（低电平有效）有关，还与**地址线（一般为高位线）**有关
  - ✓ 通常需要用到一些逻辑电路来产生片选信号，如译码器和门电路
- 合理选择存储芯片
  - ✓ 存储芯片类型（RAM或ROM）和数量的选择
  - ✓ ROM：通常用于存放系统程序、标准函数库和各类常数等

    RAM：用于用户程序使用，为用户编程而设置
  - ✓ 考虑芯片数量时，应尽量使连线简单方便
- 其他考虑：如时序配合、速度、负载等

存储器与CPU的连接示例一（1）

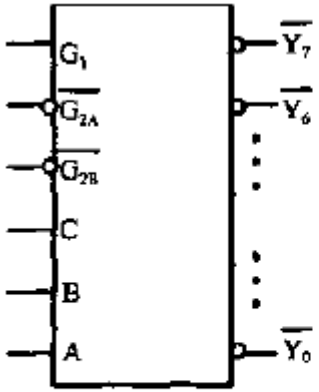
- 例 设CPU有16根地址线、8根数据线，并用$\overline{MREQ}$作为访存控制信号（低电平有效），用WR作为读/写控制信号（高电平为读，低电平为写）。现有下列存储芯片：1K×4位RAM、4K×8位RAM、8K×8位RAM、2K×8位ROM、4K×8位ROM、8K×8位ROM及74138译码器和各种门电路。画出CPU与存储器的连接图，要求如下：

  1）主存地址空间分配为：

  6000H ~ 67FFH为系统程序区；6800H ~ 6BFFH为用户程序区

  2）合理选用上述存储芯片，说明各选几片

  3）详细画出存储芯片的片选逻辑图



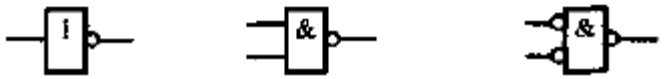
74 138 译码器

G_1, $\overline{G_{2A}}$, $\overline{G_{2B}}$为控制端

C, B, A 为变量输入端

$\overline{Y_0}$…$\overline{Y_7}$为变量输出端

必须保证控制端G_1为高电平，G_{2A}和G_{2B}为低电平，译码器才能正常工作



存储器与CPU的连接示例一（2）

解： 6000H ~ 67FFH为系统程序区；6800H ~ 6BFF

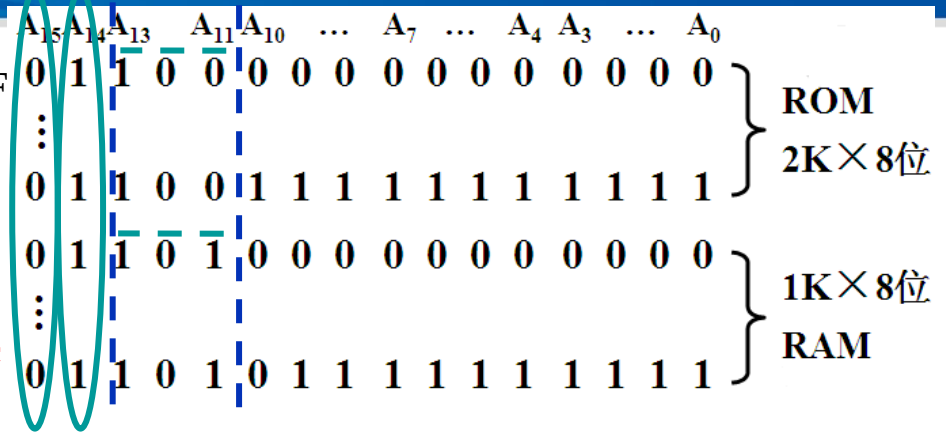
第一步，确定所要求的主存空间各区域容量

第二步，根据容量和用途，选片

系统区：1片2K×8位的ROM

用户区：2片1K×4位的RAM

位扩展



第三步，分配CPU地址线

CPU的低11位地址线A_{10}~A_0与2K×8位ROM的地址线相连；低10位地址线A_9~A_0与2片1K×4位RAM的地址线相连。

剩下的高位地址线与访存控制信号一起产生存储芯片的片选信号

第四步，片选信号的形成

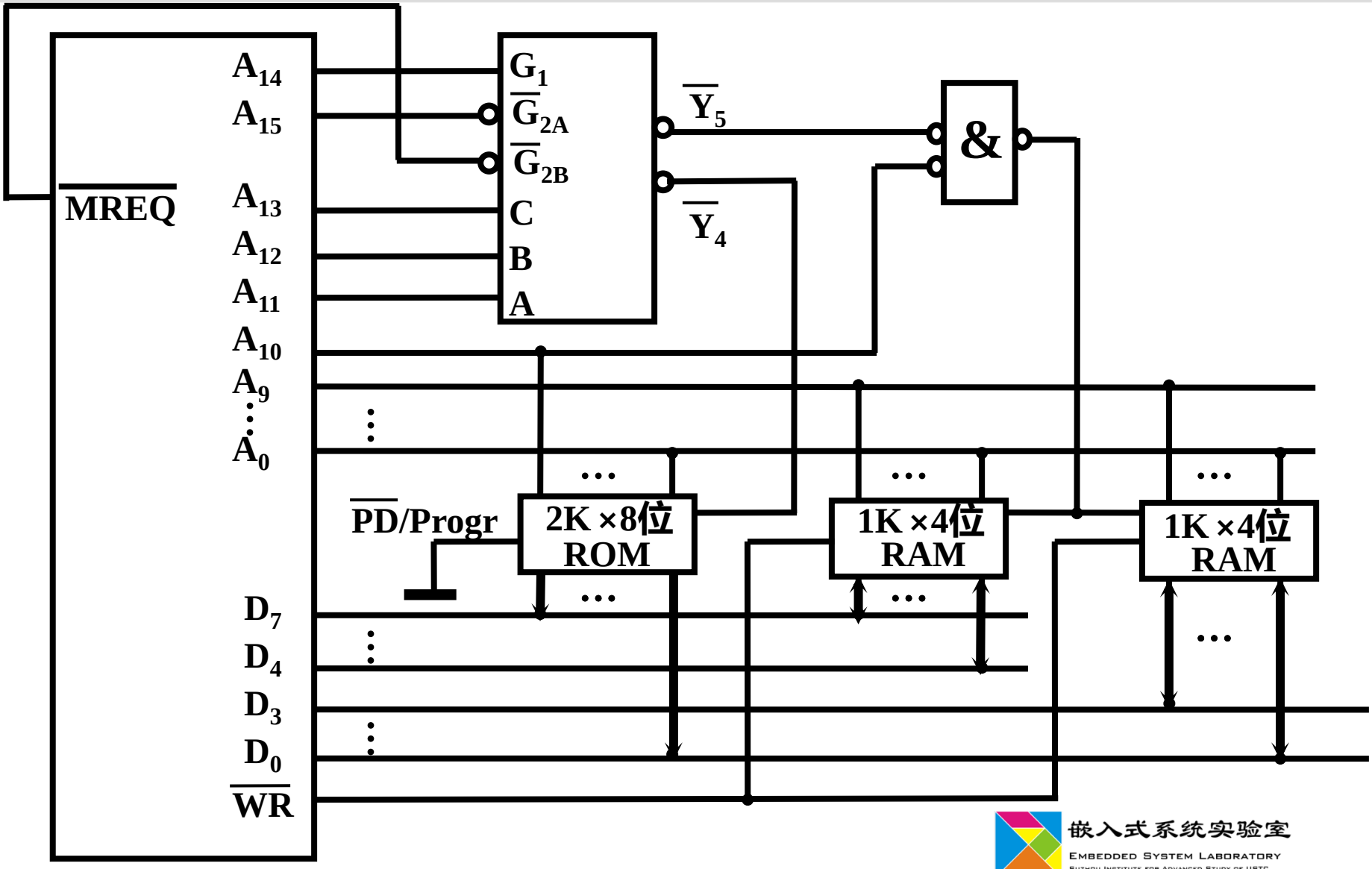
根据74138译码逻辑关系，必须保证控制端G_1为高电平，G_{2A}和G_{2B}为低电平，译码器才能正常工作

A_{15}始终为低、A_{14}始终为高，分别接$\overline{G_{2A}}$和$\overline{G_1}$；而$\overline{MREQ}$低电平有效，接$\overline{G_{2B}}$

剩下A_{13}、A_{12}、A_{11}，分别接译码器的输入端C、B、A

译码结果为4（Y4有效），选中ROM；译码结果为5（Y5有效），与A_{10}一起通过与门，都为低电平有效时，选中2片RAM

存储器与CPU的连接示例一（3）



存储器与CPU的连接示例二（1）

- 例 设CPU有16根地址线、8根数据线，并用$\overline{MREQ}$作为访存控制信号（低电平有效），用WR作为读/写控制信号（高电平为读，低电平为写）。现有下列存储芯片：1K×4位RAM、4K×8位RAM、8K×8位RAM、2K×8位ROM、4K×8位ROM、8K×8位ROM及74138译码器和各种门电路。画出CPU与存储器的连接图。

  要求主存的地址空间：最小8K地址为系统程序区，与其相邻的16K地址为用户程序区，最大的4K地址空间为系统程序工作区。详细画出存储芯片的片选逻辑并指出存储芯片的种类及片数。

解：

第一步，根据题目的地址范围写出对应的二进制地址码

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	最小 8K×8 位系统程序区
…	…															
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	相邻 16K×8 位用户程序区
…	…															
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
…	…															
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	最大 4K×8 位系统程序工作区
…	…															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

第二步，根据容量及作用，选片

系统程序区：1片8K×8位ROM

用户程序区：2片8K×8位RAM

系统程序工作区：1片4K×8位RAM

存储器与CPU的连接示例二（2）

第三步，分配地址线

A_{12}~A_0，与1片8K×8位ROM和2片8K×8位RAM的地址线相连

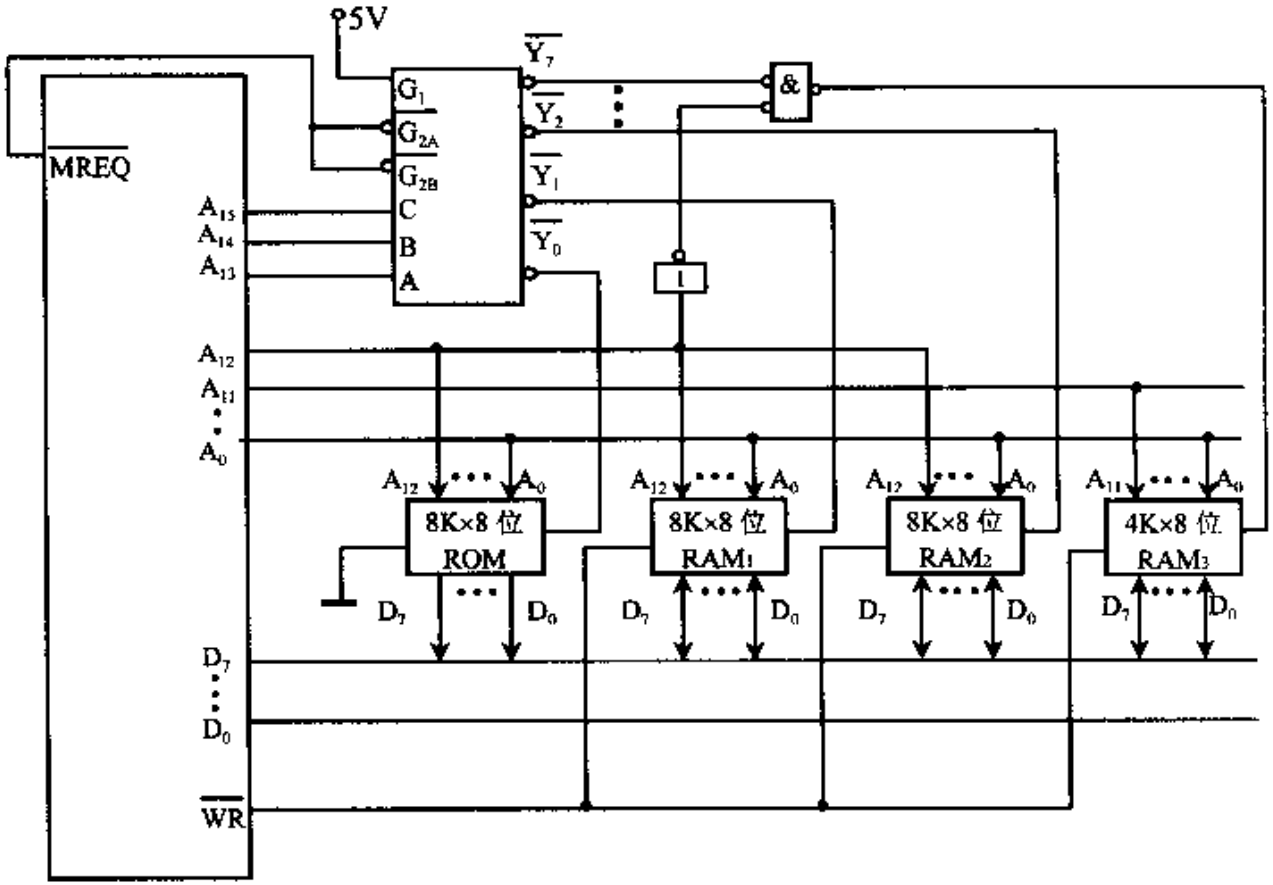
A_{11}~A_0，与1片4K×8位RAM的地址线相连

第四步，形成片选信号

G_1接+5V，$\overline{G}_{2A}$和$\overline{G}_{2B}$接$\overline{MREQ}$，保证译码器正常工作

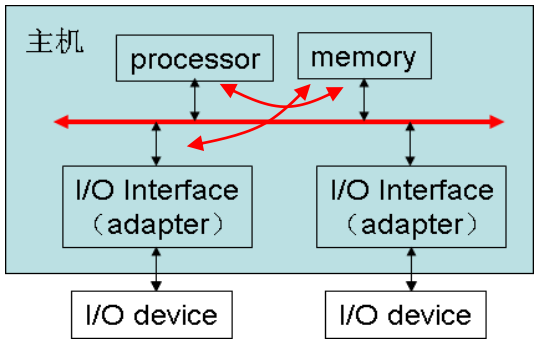
$A_{15}A_{14}A_{13}$分别接译码输入端C、B、A，其译码输出$\overline{Y}_0$、$\overline{Y}_1$、$\overline{Y}_2$、$\overline{Y}_7$分别作为三类存储芯片的片选信号

最大4K地址的片选，需要结合$\overline{Y}_7$和A_{12}



2.7 提高访存速度

- 主存的存取速度已成为计算机系统的性能瓶颈
- CPU不断增强，运算速度提升远快于存取速度提升
  - ✓ 100MHz的Pentium处理器平均10ns就执行一条指令，而DRAM的典型访问时间是60～120ns。（1-4-100ns）
  - ✓ 流水线：单周期访存
- 结构冲突
  - ✓ 访存冲突：指令预取与数据读写
  - ✓ 总线占用：CPU和I/O争抢访问主存 →减少访存

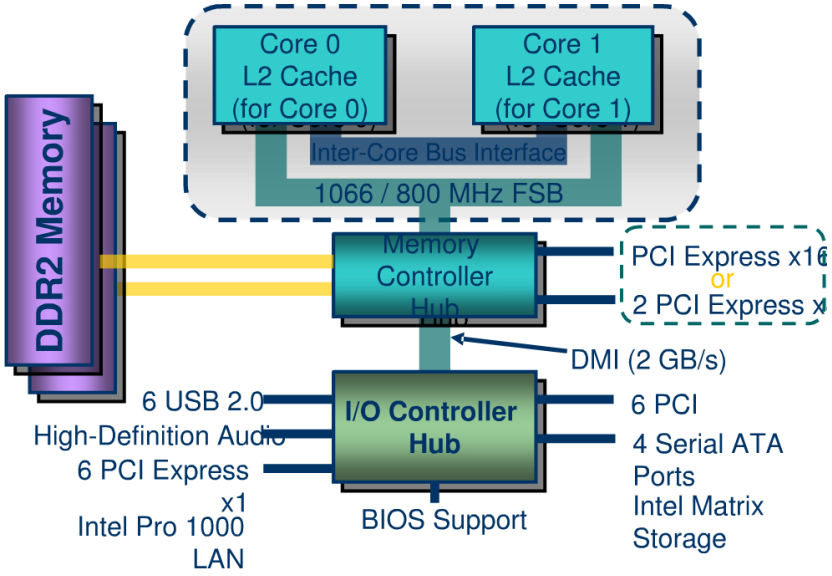


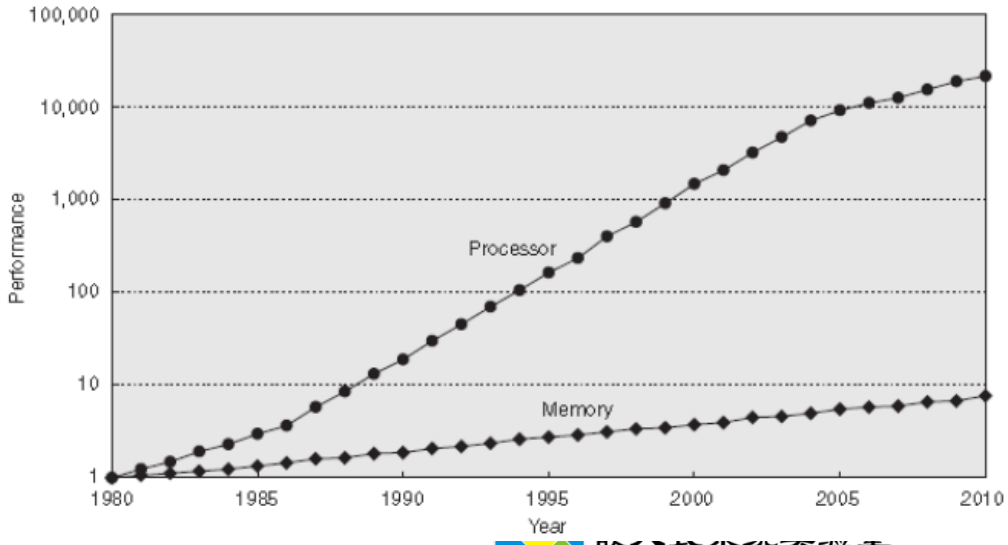
存储器带宽需求

- 例：对于200MIPS系统，总存储器带宽最多需要505MW/s
  - ✓ 取指：200MW/s（200MIPS）
  - ✓ 取数和写回：300MW/s(两个opr，一个结果)
  - ✓ I/O：5MW/s

奔腾2：800MIPS
Core i9: 81666 MIPS

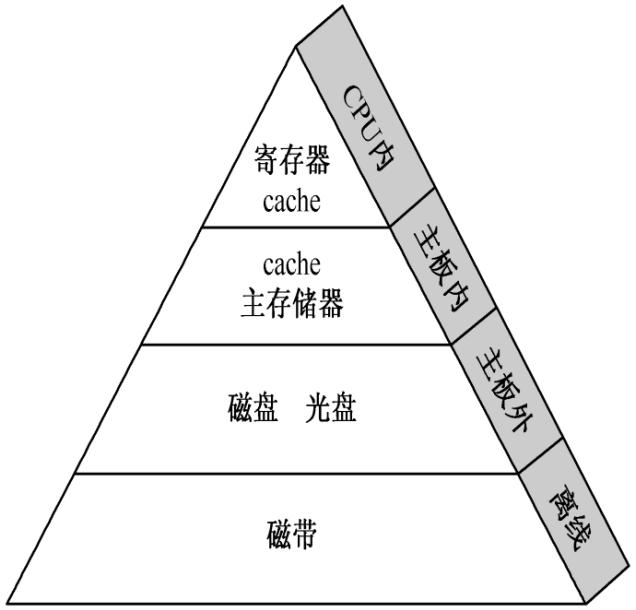
- 要求访存周期19.8ns，而一般的DRAM访存周期约100~200ns





Memory Wall

- 处理器速度增长对系统性能的贡献将被DRAM性能所屏蔽
- 提高访存速度的措施
  - ✓ 采用双端口存储器
  - ✓ 采用高速主存（SDRAM/CDRAM等）、新型存储部件
  - ✓ 并行访问存储器：采用交叉访问：用低带宽器件构成高带宽存储总线 RAS/CAS轮流送各个bank，循环输出各个字
    - 单体多字系统
    - 多体并行系统（交叉存储器）
  - ✓ 采用层次化存储系统结构
    - 优化：隐藏访存延迟
      - 减少访存频率，减少Cache miss
      - 会增加失效情况下的延迟

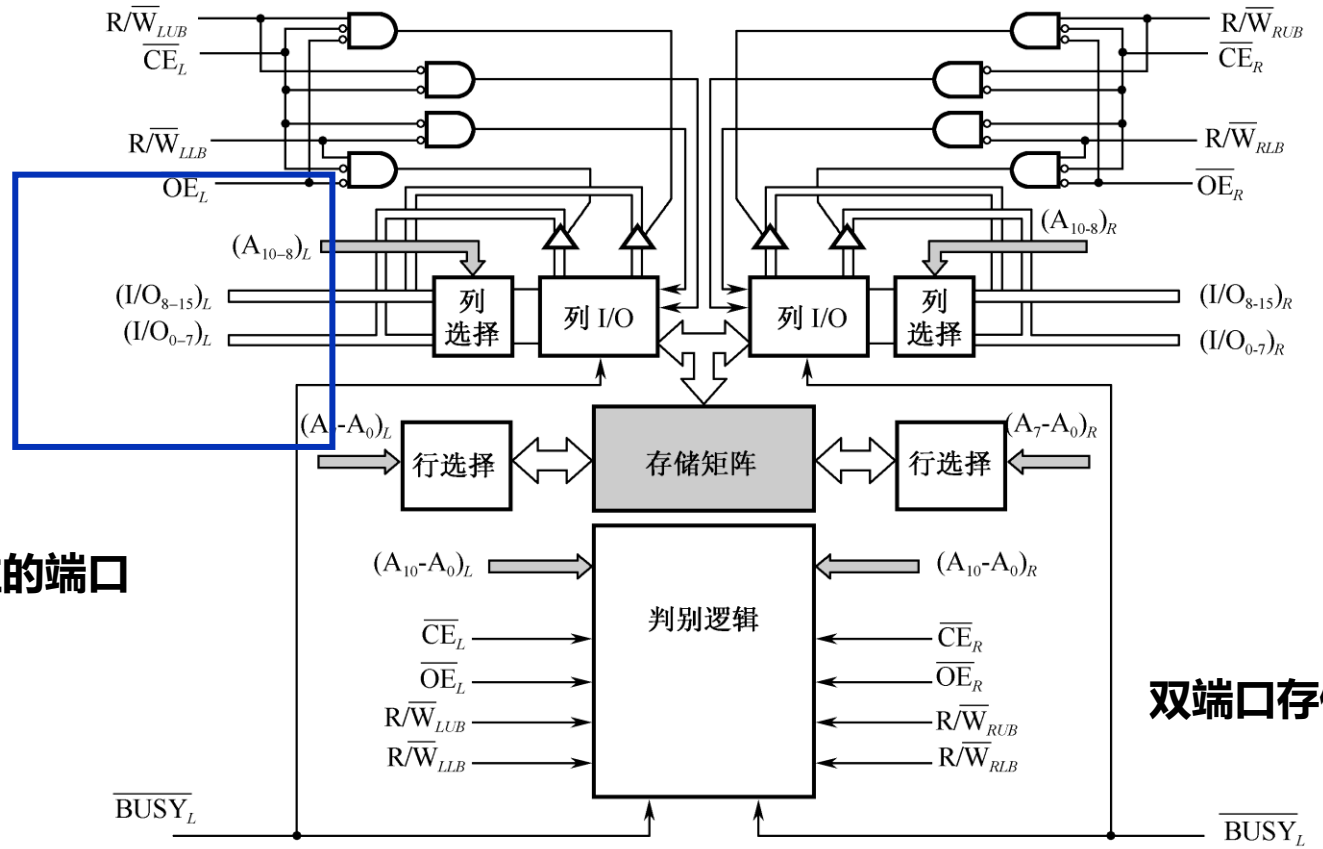


Wm. A. Wulf , Sally A. McKee, Hitting the memory wall: implications of the obvious, ACM SIGARCH Computer Architecture News, v.23 n.1, p.20-24, March 1995

双端口存储器的结构

- ✓一个存储器具有两组相互独立的读写控制电路

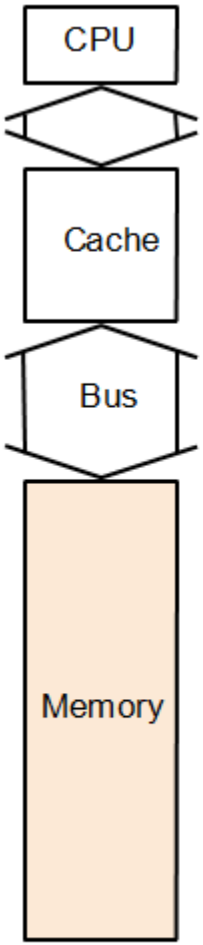
存储容量2K
字长16位



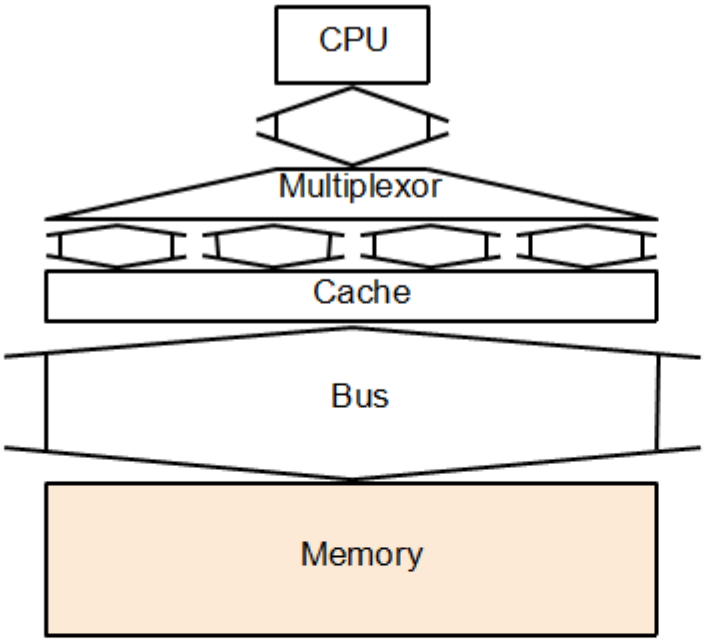
左右两套独立的端口

双端口存储器

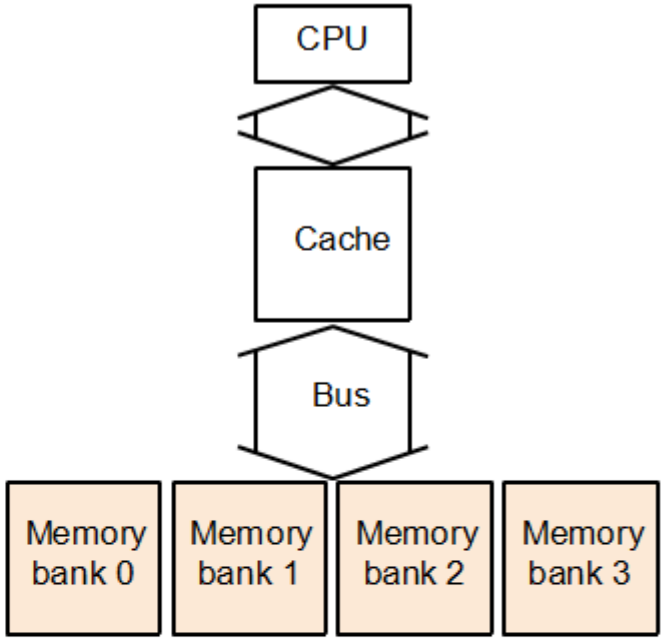
并行存储的手段



one-word wide
memory organization



wide memory organization

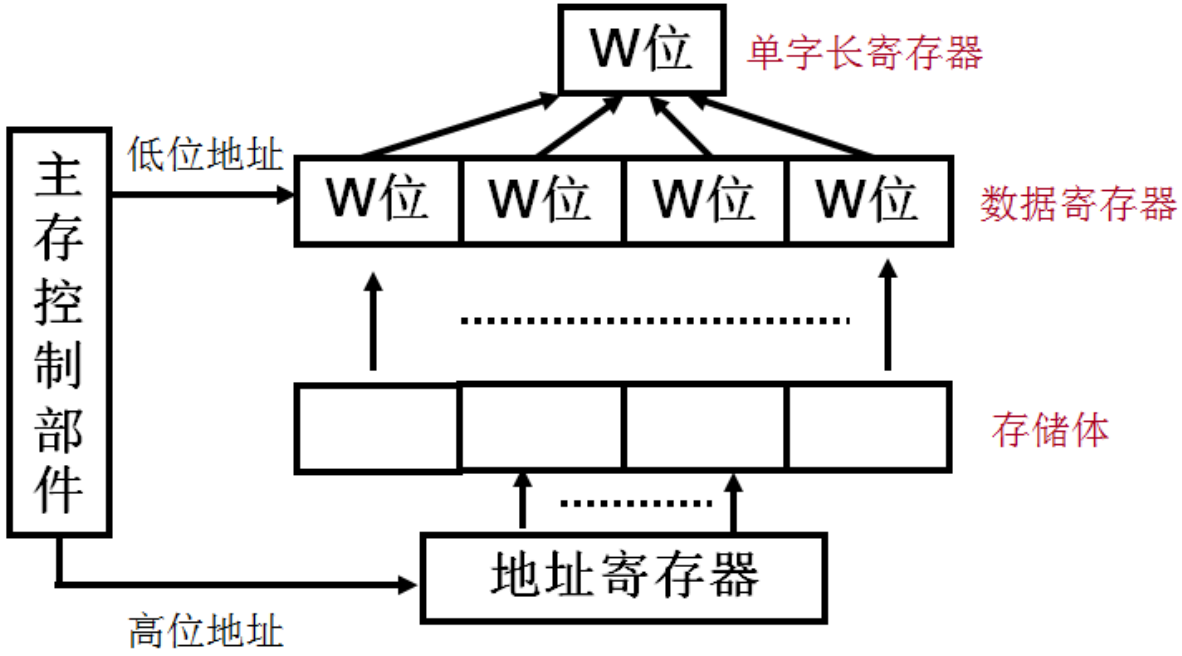


interleaved
memory organization

- 单体单字
- 单体多字
- 多体并行

单体多字系统

- 系统中只有一个存储体
  - ✓ 由单个存储芯片或多个存储芯片构成的一个独立存储器
  - ✓ 多字：如在一个周期内，从同一地址开始顺序读出4条指令字，再逐条将指令送至CPU执行。

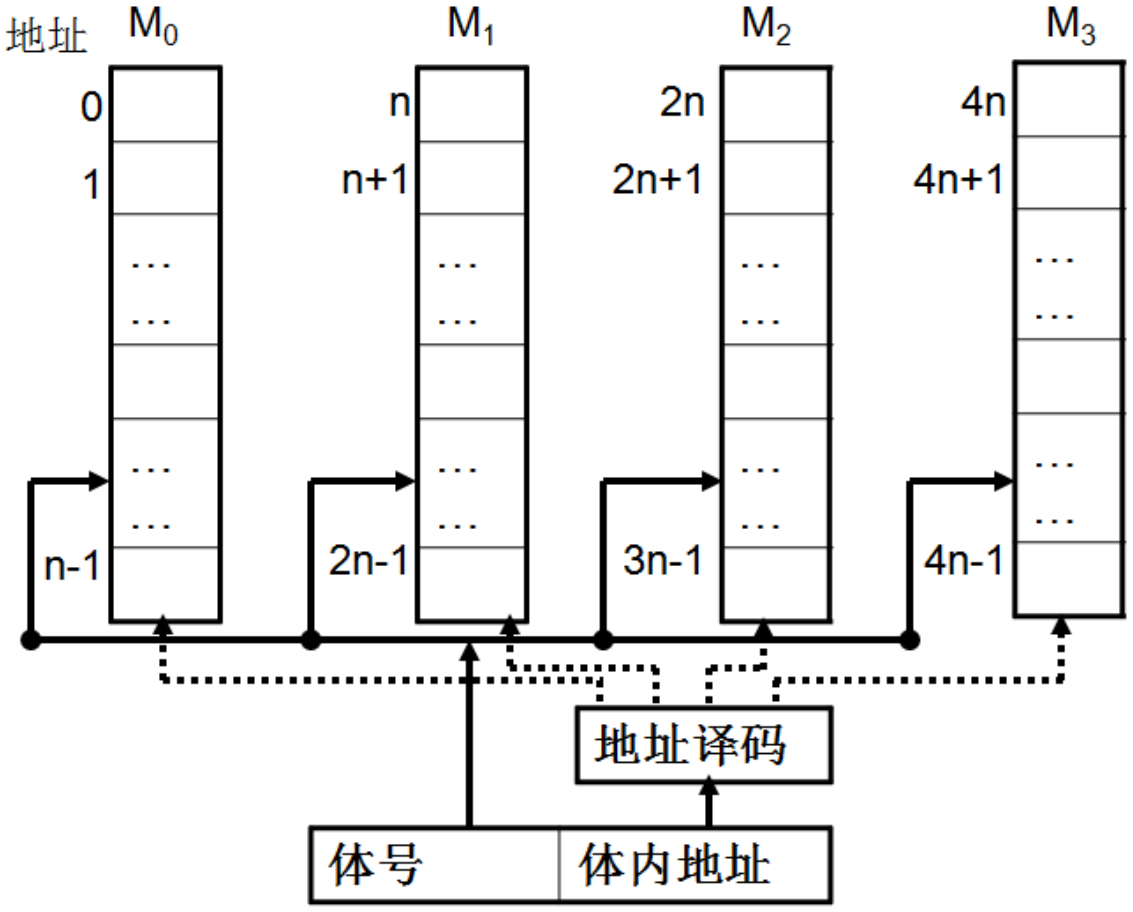


多体并行系统

- 系统含多个存储体（bank，module）
  - ✓每个存储体都有自己的读写线路、地址寄存器和数据寄存器。
- 两种编址方式
  - ✓高位交叉编址：扩容，不同应用空间➔顺序存储
  - ✓低位交叉编址：多体交叉访问➔交叉存储
- 特点
  - ✓M个模块按一定的顺序轮流启动各自的访问周期
  - ✓启动两个相邻模块的最小时间间隔等于单模块访问周期1/M（带宽提高M倍）

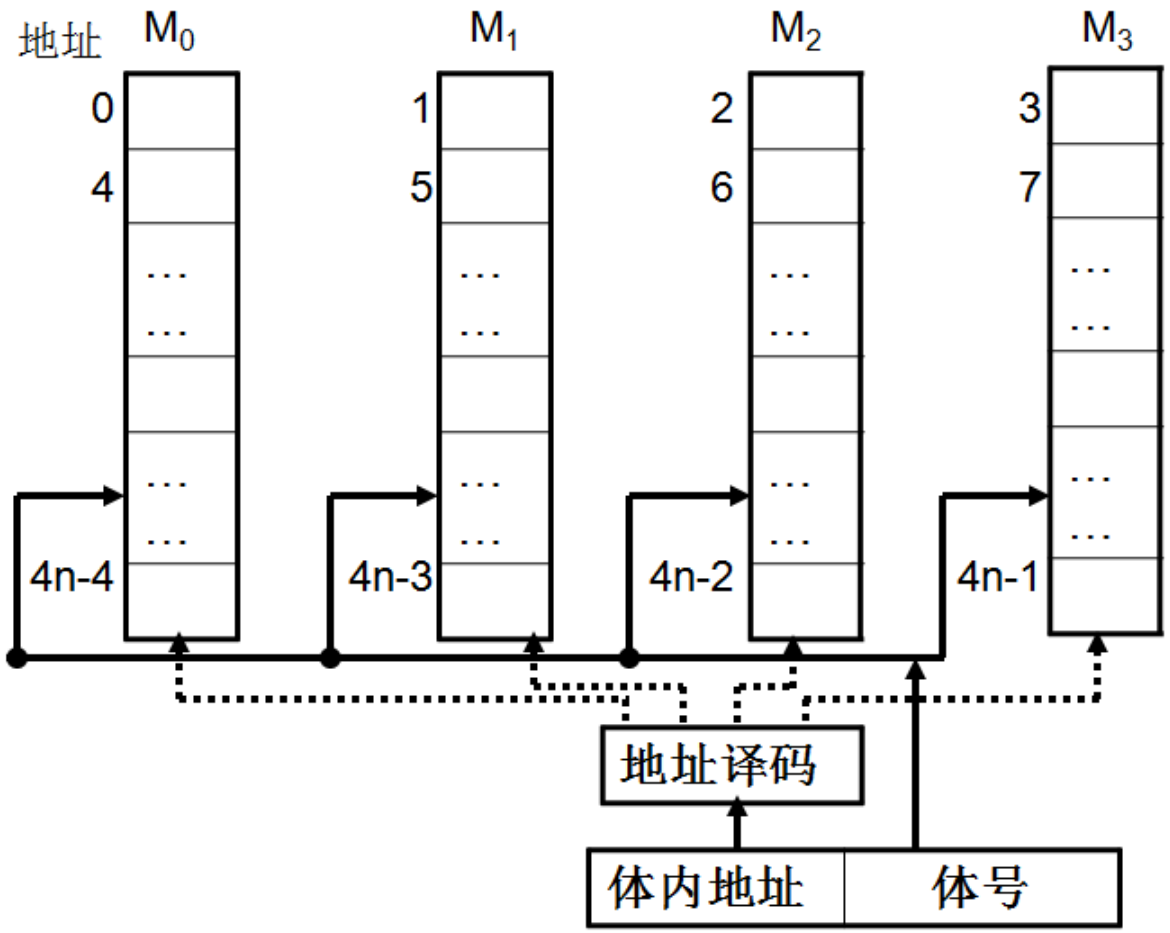
多体并行系统—高位交叉编址

- 存储体选则
  - ✓高位地址
- 顺序编址
- 方便扩容
  - ✓不同内存插槽对应不同存储体
- 支持并行访问
  - ✓多任务、多处理器系统，多个任务并发执行
  - ✓某体用于程序执行，某体用于I/O



多体并行系统—低位交叉编址

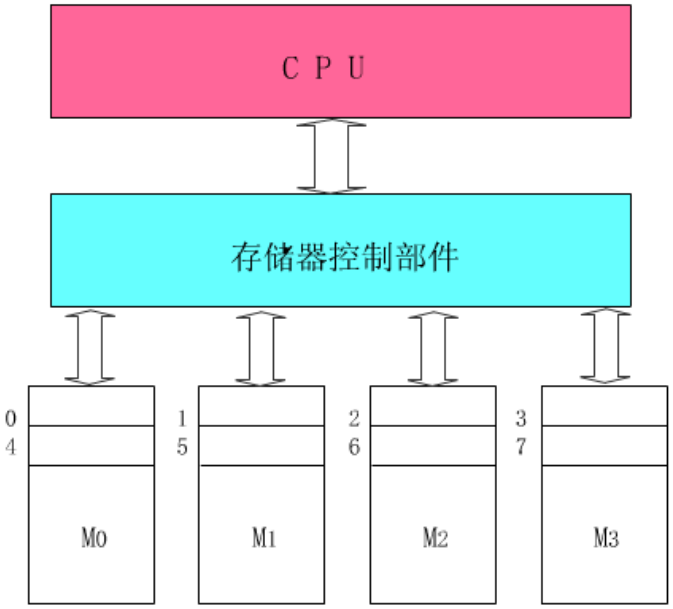
- 存储体选择
  - ✓低位地址
- 交叉编址
- 访问模式
  - ✓并行访问
    - 同时输出多个字
  - ✓交叉访问
    - 提高带宽



多体并行系统—低位交叉

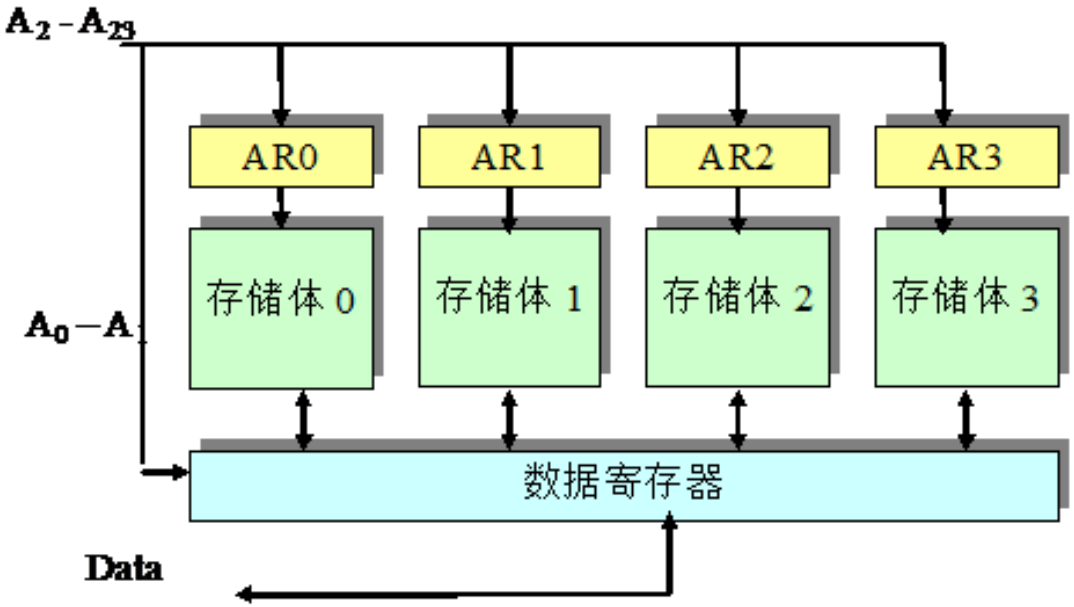
例：模四交叉个模块的编制序列

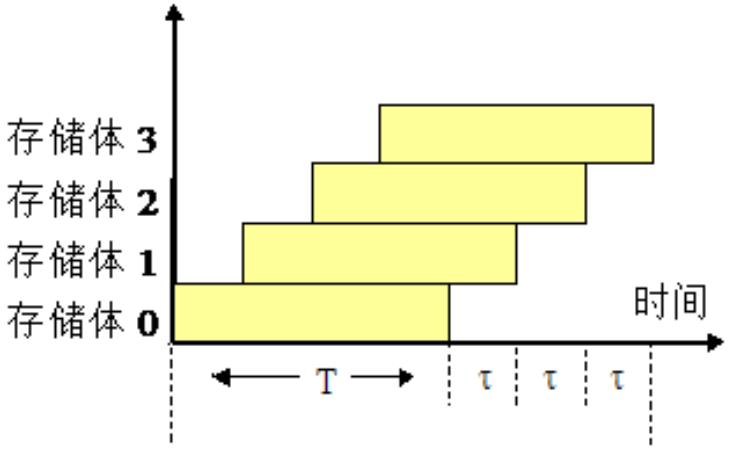
体号	体内编址序列	对应二进制地址最低二位
M1	0, 4, 8, 12, …, 4 j+0, …	0　0
M2	1, 5, 9, 13 , …, 4 j+1, …	0　1
M3	2，6，10，14 ,…,4j+2,…	1　0
M4	3，7，11，15 ,…,4j+3,…	1　1



多体并行系统—低位交叉编址

- 地址依次送各个存储体，交叉访问
  - ✓效果：两个连续地址字的读取之间不必插入等待状态
- 数据总线顺序输出（带宽提高M倍）
- 总时间=T+(m-1)*t





多模块交叉存储器例题

在一个具有4个存储体的低位多体交叉存储器中，如果处理器的访存地址为以下十进制值。求该存储器比单体存储器的平均访问速率提高多少（忽略初启时的延迟）？

(1) 0001、0002、0003、…、0100

(2) 0002、0004、0006、…、0200

(3) 0003、0006、0009、…、0300

解：(1) 各个访问操作可以交叉进行，访问速率可达到单体存储器的4倍。

(2) 只有2个存储体交叉访问（2、0、2），访问速率可达到单体存储器的2倍。

(3) 访问的存储体分别是3、2、1、0、3、…，各属于不同的存储体，访问速率可达到单体存储器的4倍。

多模块交叉存储器例题

- 设存储器容量为32字，字长64位，模块数m=4，分别用多体高位交叉即顺序方式和多体低位交叉方式进行组织。存储周期T=200ns，数据总线宽度为64位，总线传送周期t=50ns。若连续读出4个字，问顺序存储器和交叉存储器的带宽各是多少?

解：

顺序存储器和交叉存储器连续读出m=4个字的信息总量都是：

$q=64b\times4=256b$

顺序存储器和交叉存储器连续读出4个字所需的时间分别是：

$t_1=T+(m-1)t=200ns+3*50ns=350ns=3.5\times10^{-7}s$

$t_2=mT=4\times200ns=800ns=8\times10^{-7}s$

顺序存储器和交叉存储器的带宽分别是：

$W_1=q/t_1=256b\div(3.5\times10^{-7})s=730Mb/s$

$W_2=q/t_2=256b\div(8\times10^{-7})s=320Mb/s$

主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构

2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉
   - **2.9 容错与校验**

3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例

4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法

5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

Google2009：数据存储设备可靠性

- DRAM错误率超出人们预想
  - ✓DIMM(Dual-Inline-Memory-Modules)，双列直插式存储模块,中有约8.2%受到了可修正错误的影响
  - ✓平均一个DIMM每年发生3700次可修正错误
  - ✓错误类型：
    - 由电磁干扰或者硬件故障所导致
    - 软错误：很少损坏字位，是可修正的；
    - 硬错误：会损坏字位而成为物理缺陷，从而造成数据错误的反复发生。
- 硬盘：数据失效率高达6%
  - ✓错误类型：位跳变，物理损坏

容错的解决方案：Redundancy

Main approach: mask failures using redundancy（冗余）

主要解决方案：通过冗余掩盖失效

- Information redundancy （信息冗余）
  - ✓海明码、CRC码、奇偶校验码
- Time redundancy（时间冗余）
  - ✓回滚
- Physical redundancy（空间冗余）
  - ✓复用
  - ✓磁盘冗余阵列

奇偶编码校验（Parity Check Code）

奇偶编码校验（Parity Check Code）

- 一个编码系统中任意两个合法编码（码字）之间不同的二进数位（bit）数叫这两个码字的码距,也称为汉明距离，用d表示。
  - ✓例如码字10010和01110，有3个位置的码元不同，所以d=3。
- 整个编码系统中任意两个码字的的最小距离就是该编码系统的码距。任何一种编码是否具有检测能力或纠错能力，都与编码的最小距离有关。
  - ✓在一个码组内为了检测D个误码，需要最小码距L>=D+1;
  - ✓在一个码组内为了纠正C个误码，需要最小码距L>=2C+1;

编码纠错理论

- L-1 >= D+C 且 D>=C
  - ✓ 即编码最小距离L越大，则其检测错误的位数D也越大，纠正错误位数C也越大，且纠错能力恒小于或等于检测能力。
    - 例如，L = 2，则D = 1，C = 0。码距=2才能检测1位错
    - 例如，L = 3，则D = 2，C = 0；或D=1，C=1。码距=3才能检测2位错，或者检测1位错，纠错1位。
- 应用
  - ✓ 数据通信：奇偶校验（串行），CRC（网络）
  - ✓ 硬盘：CRC
  - ✓ 内存：ECC（错误检查和纠正）校验

奇偶编码校验（Parity Check Code）

- **在被传送的n位代码($b_{n-1}b_{n-2}...b_1b_0$)上增加一位校验位P（Parity），将原数据和得到的奇（偶）校验位一起进行存取或传送(即传送P$b_{n-1}b_{n-2}...b_1b_0$)。**
  - ✓奇校验：使“1”的个数为奇数
    - 0000 0000 - >0000 0000 1
    - 0000 0001 - >0000 0001 0
  - ✓偶校验：使“1”的个数为偶数
    - 0000 0000 - >0000 0000 0
    - 0000 0001 - >0000 0001 1
- 为什么能容错？具有什么容错能力？
  - ✓L-1 >= D+C 且 D>=C
  - ✓D=1 C=0

能够纠错一位的Hamming码

- **海明码需要几位校验码?**
- **设有k位数据，r位校验位,r位校验位有2^r个组合。**
  - ✓ **若用0表示无差错，则剩余2^r-1个值表示有差错，并指出错在第几位。**
- **由于差错可能发生在k个数据位中或r个校验位中，因此有：$2^r-1 \geq r+k$**

数据位*k*	校验位*r*	总位数*n*
1	2	3
2~4	3	5~7
5~11	4	9~15
12~26	5	17~31
27~57	6	33~63
58~120	7	65~127

海明校验码(校验位置)

校验位和数据位是如何排列的?

校验位排列在 2^{i-1} (i =0,1,2,…)的位置上

例1：有一个4位数为$D_4D_3D_2D_1$,需要3位校验码$P_3P_2P_1$，由此生成一个海明码

7	6	5	4	3	2	1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
			2^2		2^1	2^0

例2： 有一字节的信息需生成海明码

12	11	10	9	8	7	6	5	4	3	2	1
D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
				8				4		2	1

校验位取值公式及计算举例

海明码的校验位P_i和数值位D_i的关系?

- ✓ 设k位数据，r位校验码，把k+r=m个数据记为$H_mH_{m-1}\ldots H_2H_1$(海明码），每个校验位P_i在海明码中被分配在2^{i-1}位置上。
- ✓ **H_i由多个校验位校验：每个海明码的位号要等于参与校验它的几个检验位的位号之和。**

$$\text{分解}i = 2^{j_1} + 2^{j_2} + \ldots + 2^{j_x}\ (j_1 \neq j_2 \neq \ldots \neq j_x)$$

得：H3由P1和P2校验，H5由P1和P3校验， H6由P2和P3校验，H7由P1、P2和P3校验，。。。

即：P_i参与第j_1、j_2、…、j_x个校验位的计算(P1参与H3、H5、H7、…)

海明码位号	H_{12}	H_{11}	H_{10}	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
数据位/校验位	D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
参与校验的校验位位号	4, 8	1, 2, 8	2, 8	1, 8	8	1, 2, 4	2, 4	1, 4	4	1, 2	2	1

例:

每个数据位至少出现在两个Pi值的形成关系中。当任一数据位发生变化时，必将引起二或三个Pi值跟着变化。

数据位为1011（偶校验）

P3= D4⊕D3 ⊕D2

0 = 1 ⊕ 0 ⊕ 1

P2= D4 ⊕D3 ⊕D1

0 = 1 ⊕ 0 ⊕ 1

P1 = D4 ⊕D2 ⊕D1

1 = 1 ⊕ 1 ⊕ 1

最后，海明码为1010101

H	7	6	5	4	3	2	1
	D_4	D_3	D_2	P_3	D_1	P_2	P_1
2^2	D_4	D_3	D_2	P_3			
2^1	D_4	D_3			D_1	P_2	
2^0	D_4		D_2		D_1		P_1
	1	0	1		1		

海明码的纠错原理

- 海明码的接收端的公式:
  - ✓$S_3= P_3\oplus D_4\oplus D_3 \oplus D_2$

    $S_2= P_2\oplus D_4 \oplus D_3 \oplus D_1$

    $S_1= P_1\oplus D_4 \oplus D_2 \oplus D_1$
  - ✓假定 海明码1010101在传送中变成了1000101

    $S_3= P_3\oplus D_4\oplus D_3 \oplus D_2=0\oplus 1\oplus 0 \oplus 0 =1$

    $S_2= P_2\oplus D_4 \oplus D_3 \oplus D_1=0\oplus 1\oplus 0 \oplus 1=0$

    $S_1= P_1\oplus D_4 \oplus D_2 \oplus D_1=1\oplus 1\oplus 0 \oplus 1=1$

  因此,由$S_3S_2S_1$= 101,指出第5位错,应由0变1

CRC循环冗余校验码

CRC（Cyclic Redundancy Check）

- **CRC：存储器、数据通信**
- **基于模2运算：不考虑进位和借位**
  - ✓模2加减运算：异或（相同为“0”，不同为“1”）
  - ✓模2乘：按模2加求部分积之和
  - ✓模2除：按模2减求部分余数

模2运算举例

$0 \pm 1 = 1 \quad 0 \pm 0 = 0 \quad 1 \pm 0 = 1 \quad 1 \pm 1 = 0$

$$
\begin{array}{r}
1010 \\
\times \quad 101 \\
\hline
1010 \\
0000 \\
1010 \\
\hline
100010
\end{array}
$$

$$
\begin{array}{r}
101 \\
\hline
101 \overline{)\,10000} \\
101 \\
\hline
0010 \\
000 \\
\hline
0100 \\
101 \\
\hline
001
\end{array}
$$

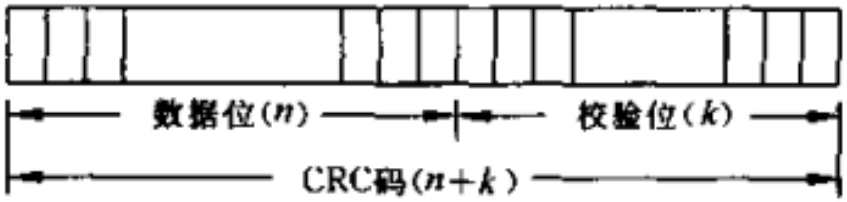
余数＝001

CRC校验步骤

□CRC生成

- ✓将n位数据$D_{n-1},\ldots D_0$用n-1次多项式M(x)表示，即 $M(x) = D_{n-1}x^{n-1} + D_{n-2}x^{n-2} +\ldots+ D_1 x^1 + D_0 x^0$
- ✓将M(x)左移k位（补0），即得：$M(x)*x^k$
- ✓将$M(x)*x^k$除以k+1位的**生成多项式G(x)**，余数即为k位的CRC校验位
- ✓将CRC校验位拼装在$D_{n-1},\ldots D_0$之后，成为n+k位数据，也称（n+k，n）码。

CRC码的组成



CRC码—编码举例

例:有效信息为1100,生成多项式G(x)=1011,将其编成CRC

- 解: 数据位数n=4，校验位数k=G(x)位数-1=3

$M(x)= x^3 + x^2 = 1100$

$M(x)\cdot x^3 = x^6 + x^5 = 1100000$

$G(x)= x^3 + x + 1 = 1011$

$$\frac{M(x)\cdot x^3}{G(x)} = \frac{1100000}{1011} = \mathbf{1110} + \frac{010}{1011}$$

$M(x)\cdot x^3 + R(x) = 1100000 + 010$

$= 1100010$

编好的循环校验码称为(7,4)码,即n+k=7, n=4

CRC的译码与纠错

- 将收到的n+k位CRC码用**约定**的生成多项式G(x)去除
  - ✓ 正确，则余数＝0。
  - ✓ 如果某一位出错,则余数不为0。不同位出错，余数不同。
- 余数和出错位之间的对应关系不变。
  - ✓ 与待测码无关，与**生成多项式**有关

对应G(x)=1011的(7,4)码的**出错模式**

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	余数
正确	1	1	0	0	0	1	0	0 0 0
A_7错	1	1	0	0	0	1	1	0 0 1
A_6错	1	1	0	0	0	0	0	0 1 0
A_5错	1	1	0	0	1	1	0	1 0 0
A_4错	1	1	0	1	0	1	0	0 1 1
A_3错	1	1	1	0	0	1	0	1 1 0
A_2错	1	0	0	0	0	1	0	1 1 1
A_1错	0	1	0	0	0	1	0	1 0 1

CRC的译码与纠错（con't）

口余数循环

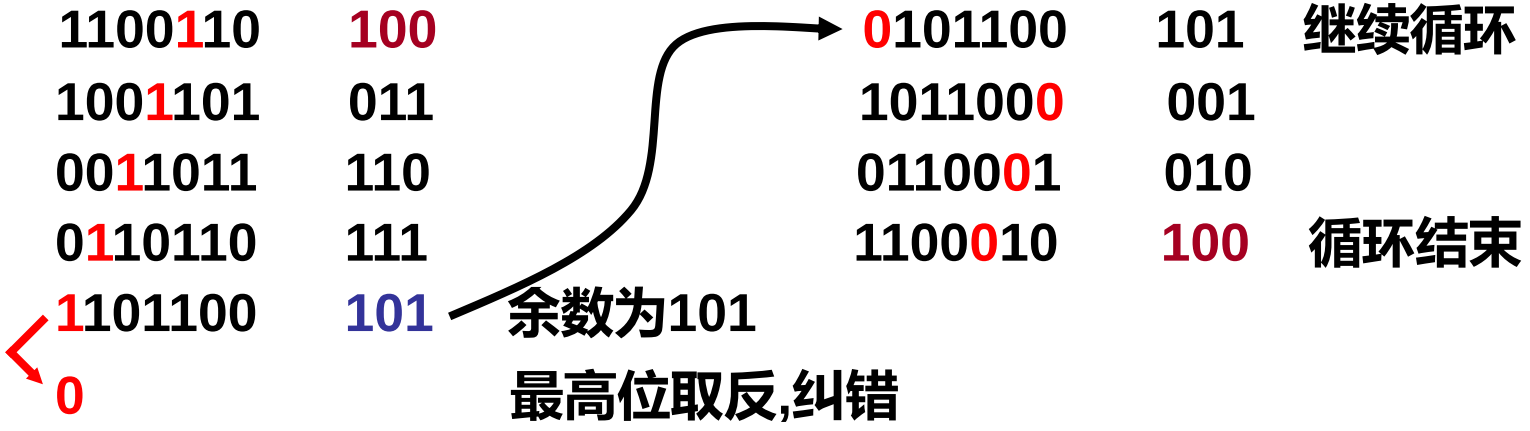
- ✓ 如果对余数补0，除以G(x)，得下一余数。
- ✓ 继续除下去，各次余数将按右表顺序循环。
- ✓ 特定生成多项式的余数模式固定

$$1011\overline{)\ 0010}$$

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	余数
正确	1	1	0	0	0	1	0	0 0 0
A_7错	1	1	0	0	0	1	1	0 0 1
A_6错	1	1	0	0	0	0	0	0 1 0
A_5错	1	1	0	0	1	1	0	1 0 0
A_4错	1	1	0	1	0	1	0	0 1 1
A_3错	1	1	1	0	0	1	0	1 1 0
A_2错	1	0	0	0	0	1	0	1 1 1
A_1错	0	1	0	0	0	1	0	1 0 1

CRC码的纠错方法 - 循环移位法

- 将CRC码进行左循环移位，至出错位被移至最高位
  - ✓ 余数添0继续除法，当余数为101时，出错位被移到最高位
- 对最高位取反，纠错
- 继续循环移位，直至循环一周
  - ✓ 继续余数除法，直至余数变成第一次的余数。
- 例A_5出错（生成多项式1011）



生成多项式G(x)

- 生成多项式G(x)应能满足下列要求:
  - ✓ 任何一位发生错误都应使余数不为0.
  - ✓ 不同位发生错误应当使余数不同.
  - ✓ 对余数继续作模2除,应使余数循环.
  - ✓ 例:
    - G(x)=x+1=11 (7,6)码,判一位错
    - G(x)= $x^3+x+1=1011$ (7,4)码,判二位错或纠一位错
    - G(x)= $x^3+x^2+1=1101$ (7,4)码,判二位错或纠一位错
    - G(x)=(x+1)(x^3+x+1)=11101 (7,3)码,判二位错并纠一位错

课后思考

- 理解“码距”
  - ✓具有1位纠错能力的编码系统最小码距是多少?
  - ✓单纠错海明码码距是多少?
  - ✓（n+k, n）CRC码码距是多少?
- 比较海明码与CRC码的容错能力
- 作业：无

课堂练习

例:有效信息为1010,生成多项式G(x)=1011,将其编成CRC码。

- 解: 数据位数n=4，校验位数k=G(x)位数-1=3

$M(x)= x^3 + x^1 = 1010$

$M(x)\cdot x^3 = x^6 + x^4 = 1010000$

$G(x)= x^3 + x + 1 = 1011$

$$\frac{M(x)\cdot x^3}{G(x)} = \frac{1010000}{1011} = \mathbf{1110} + \frac{011}{1011}$$

$M(x)\cdot x^3 + R(x) = 1010000 + 011$

$= 1010011$

主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构
2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉
   - 2.9 汉明校验与CRC校验
3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例
4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法
5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

3.1 Cache的基本原理

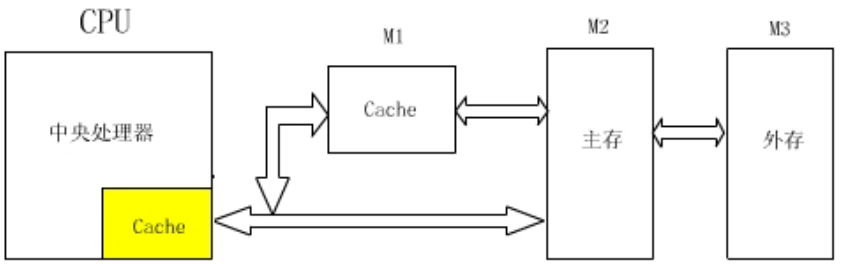
引入Cache的原因

- ✓解决CPU和主存之间速度不匹配的问题
  - 据统计，CPU的速度平均每年改进60%，而组成主存的DRAM的速度平均每年只改进7%
- ✓避免CPU与I/O设备的访存冲突
  - 一旦主存需与I/O设备交换信息时，CPU可以访问Cache获取信息

Cache的组成

- ✓一般使用SRAM构成
  - 典型大小在几百KB到几MB
- ✓多级结构
  - CPU内——主板上
- ✓全硬件实现，对用户透明

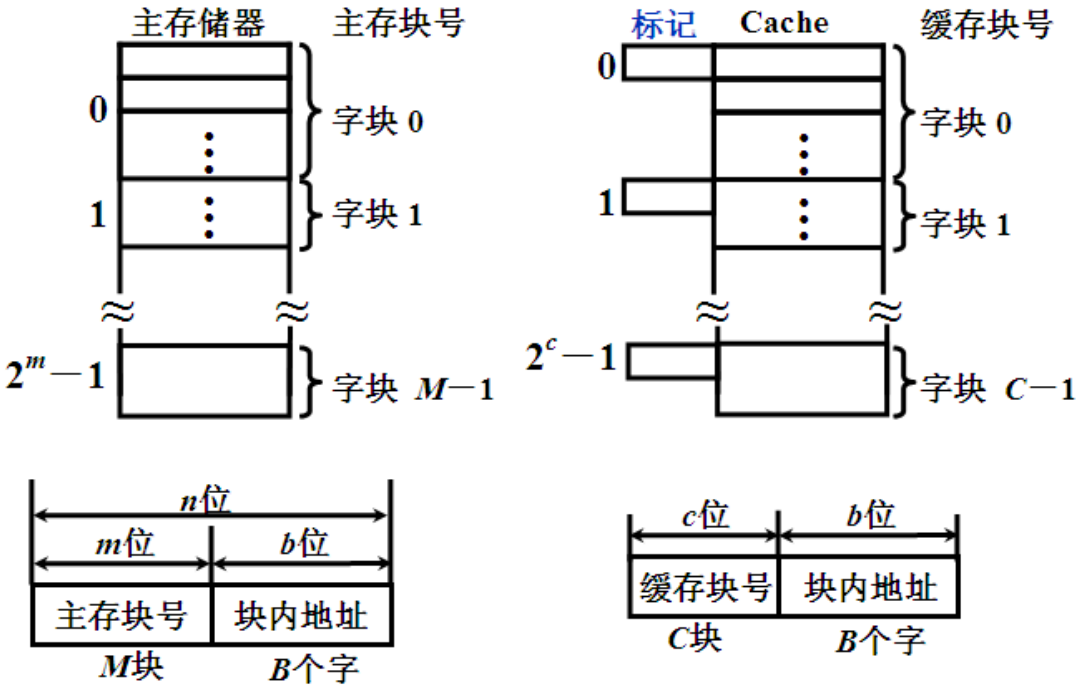
利用了程序的局部性原理



3.1 Cache的基本原理（2）

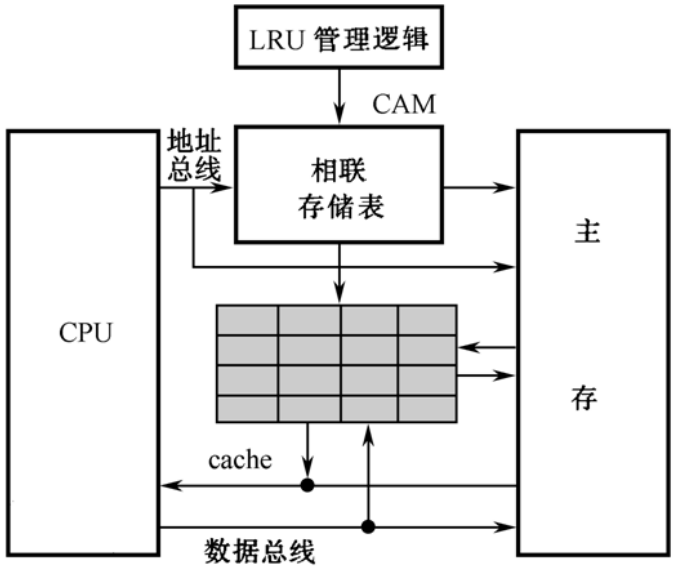
Cache-主存空间的基本结构

- ✓对主存和Cache分块，每块中包含同样数目的存储字
  - Cache容量远小于主存，因此Cache块数远少于主存块数
- ✓Cache与主存之间的数据交换以块为单位



3.1 Cache的基本原理（3）

Cache的工作原理

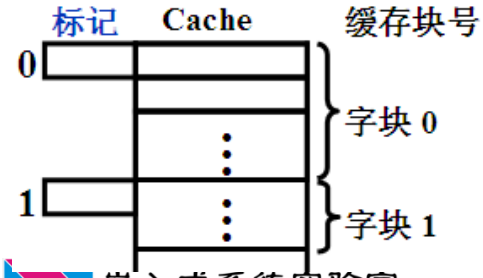


- ✓控制逻辑
  - 片外Cache，主存/Cache控制器
  - 片内Cache，CPU控制
- ✓数据交换过程
  - CPU将内存地址发给Cache和主存
  - 控制逻辑判断该地址是否在Cache中

    1）命中，Cache将存储字提供给CPU（按字传送）

    2）未命中，主存将存储字提供给CPU，同时将该字所在的整个块调入Cache（按块传送）——主存块与Cache块建立对应关系

判断是否命中——利用Cache块的标记



3.1 Cache的基本原理（4）

Cache命中率

- ✓衡量Cache的性能效率的指标
- ✓与Cache的容量和块长有关

设程序执行过程中，N_c表示访问Cache完成存取的总次数，N_m表示访问主存完成存取的总次数，则命中率h为

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

设t_c为命中时的Cache访问时间，t_m为未命中时的主存访问时间，则Cache/主存平均访问时间t_a为

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

用e表示访问效率，有

$$e = \frac{t_c}{t_a} \times 100\% = \frac{t_c}{ht_c + (1-h)t_m} \times 100\%$$

t_a越接近t_c，h越接近1，效率越高

Cache命中率计算例题

- 例 假设CPU执行某段程序时，共访问Cache命中2000次，访问主存50次。已知Cache存取周期为50ns，主存存取周期为200ns。求Cache-主存系统的命中率、效率和平均访问时间。

  解：

  Cache命中率 h = 2000 / (2000 + 50) = 0.97

  平均访问时间为

  $$50\ \text{ns} \times 0.97 + 200\ \text{ns} \times (1 - 0.97) = 54.5\ \text{ns}$$

  访问效率

  e = 访问Cache的时间/ 平均访问时间 × 100%

  = 50ns / 54.5ns ×100%

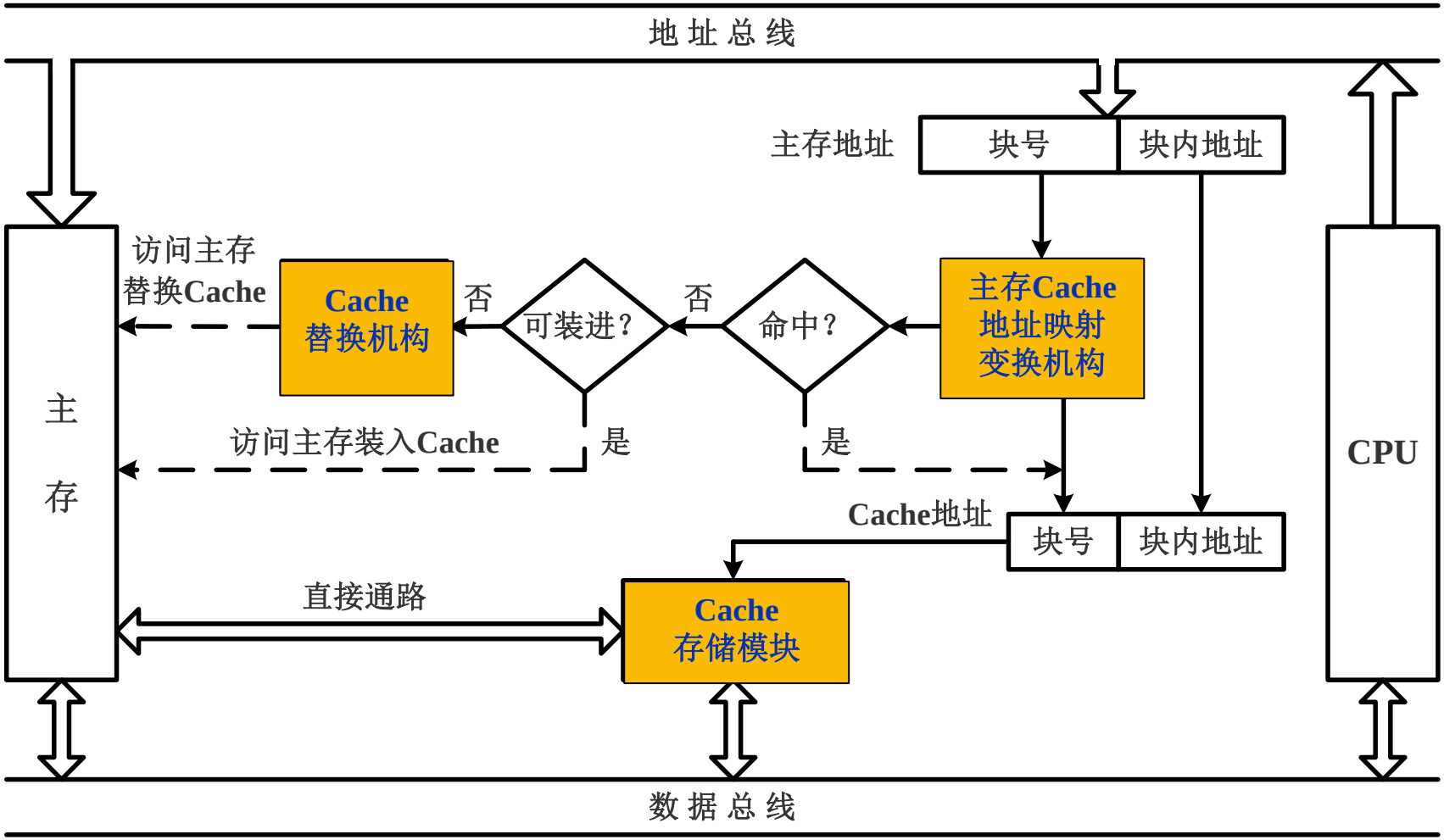
  = 91.7%

Cache命中率的讨论

- Cache命中率与容量
  - ✓Cache容量越大，命中率越高
    - 当容量达到一定值时，命中率将不再明显提高
  - ✓Cache容量越大，成本越高

- Cache命中率与块长
  - ✓取决于程序的局部特性
  - ✓随着块由小到大增长，命中率最初将提高，而后下降
  - ✓块长一般取每块4~8个字

3.2 Cache的基本结构



3.2 Cache的基本结构（2）

- Cache存储模块
  - ✓以块为单位与主存交换信息
- 地址映射变换机构
  - ✓将CPU送来的主存地址转换为Cache地址
  - ✓地址映射函数
- 替换机构
  - ✓Cache内容已满，无法容纳新的主存块时，确定Cache内哪个块移出到主存
  - ✓替换算法

Cached的存储结构

- 单一Cache和多级Cache
  - ✓单一Cache
    - CPU和主存之间只设一个Cache
    - 随着技术发展，与CPU制作在同一芯片中——片内Cache
    - 存取速度快，不占用片外系统总线，但容量受限
  - ✓多级Cache
    - 在片内Cache基础上，再增加一（多）级片外Cache
    - 与CPU之间使用独立数据通路
- 统一Cache和分离Cache
  - ✓统一Cache，指令和数据存放在同一Cache内
  - ✓分离Cache，指令和数据分别存放——指令Cache和数据Cache

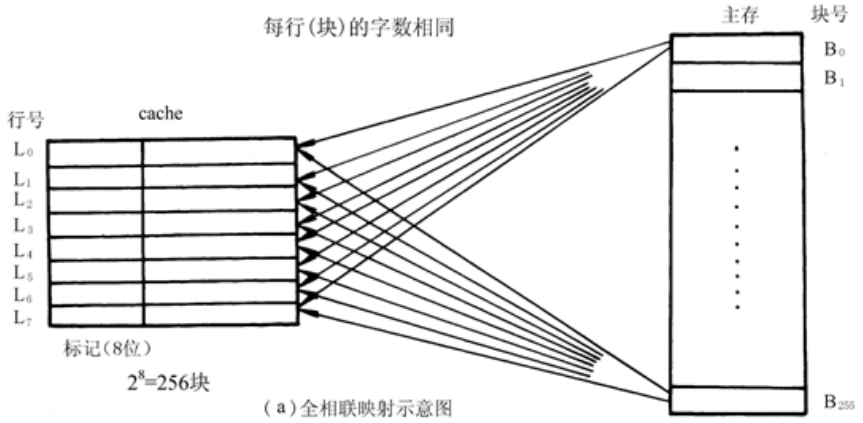
地址映射

- ✓将主存地址定位到Cache地址的方法
- ✓地址映射的方式（Cache组织方式）
  - 全相联映射——灵活性大的映射关系
  - 直接映射——固定的映射关系
  - 组相联映射——前两种的折中
- ✓选择映射方式需考虑的因素
  - 硬件是否容易实现
  - 地址变换的速度是否快
  - 发生冲突的概率是否低

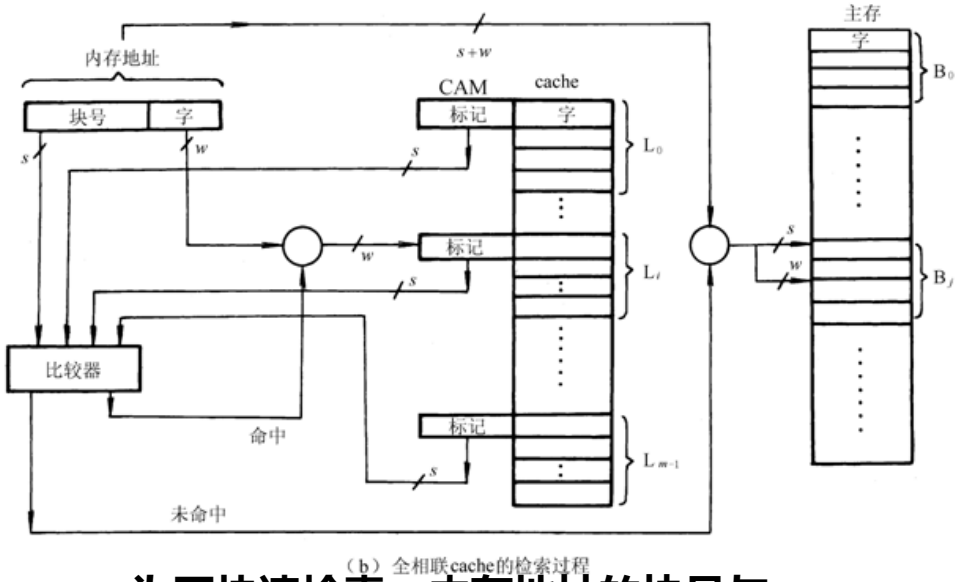
3.3 Cache与主存的地址映射——全相联映射

全相联映射

- ✓主存中的每一个字块可以映射到Cache中的任何一块位置上
- ✓利用标记来记录映射关系，标记构成了一张目录表



(a)全相联映射示意图



(b) 全相联cache的检索过程

将地址分为两部分（块号和字），在内存块写入Cache时，同时写入块号标记

主存地址长度= (s+w)位，块为s位，主存块数为2^s；字为w位，块大小为2^w个字或字节

适用于小容量的Cache

为了快速检索，内存地址的块号与Cache中所有行的标记同时进行比较

优点：冲突概率小，Cache的利用高。
缺点：比较器难实现，需要一个访问速度很快代价高的相联存储器

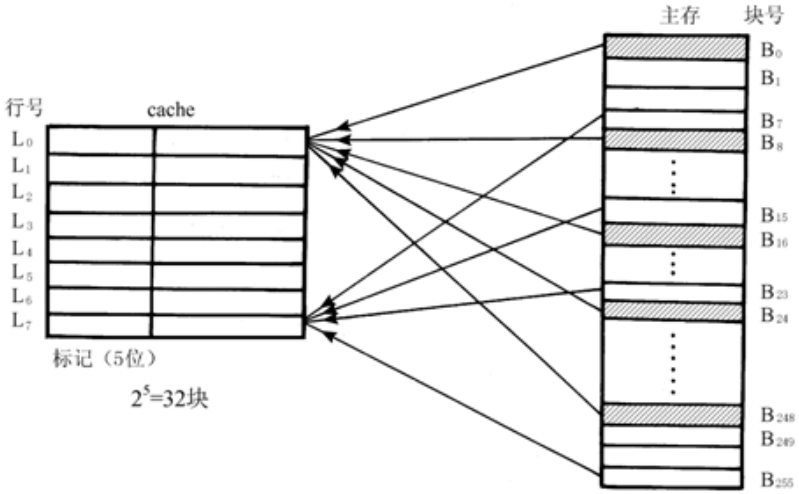
3.3 Cache与主存的地址映射——直接映射

- 直接映射
  - ✓每个主存块只能映射到Cache的一个特定块位置
    每个Cache块可以和若干个主存块对应
  - ✓映射关系式
    - i表示Cache块号，j表示主存块号
      m表示Cache的总块数（一般为2的幂数）

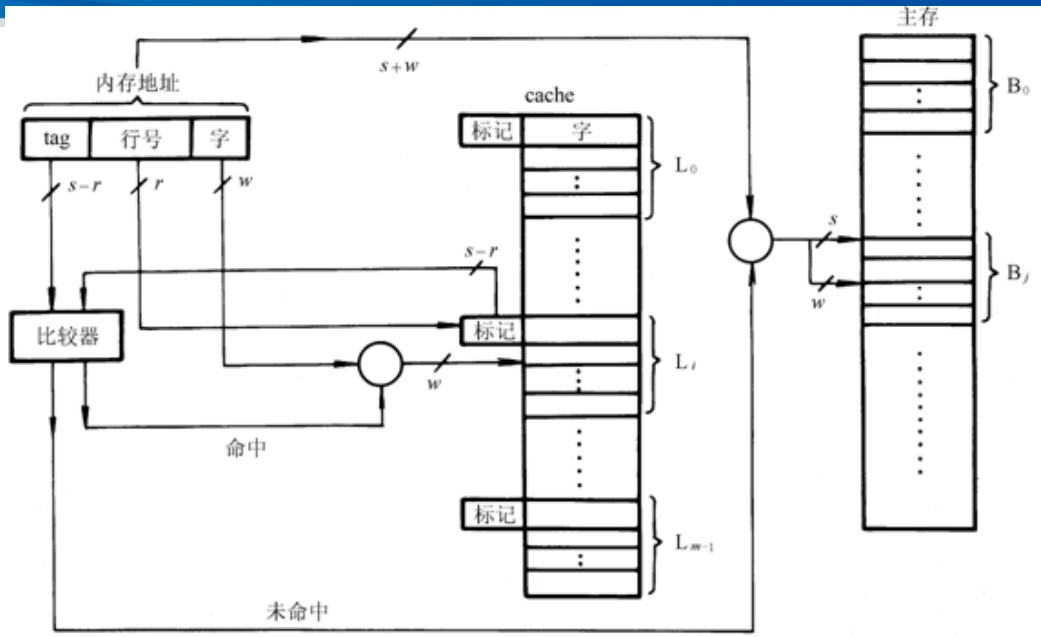
$$i = j \bmod m$$

  - ✓优点：比较电路少，硬件实现简单，Cache地址为主存地址的低几位，不需变换。
    缺点：冲突概率高
  - ✓适合大容量Cache

直接映射示意



(a) 直接映射示意图



(b) 直接映射cache的检索过程

主存地址长度为(s+w)位
其中s又分为两部分：
r位，作为选择Cache块的行号
s-r位，用作标记

利用主存地址的行号选择Cache的相应块
利用主存地址的标记进行一次比较以确定是否命中

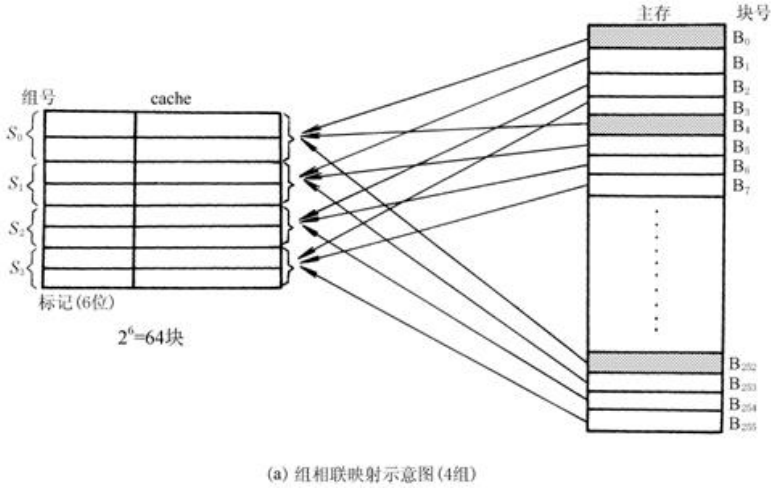
3.3 Cache与主存的地址映射——组相联映射

组相联映射

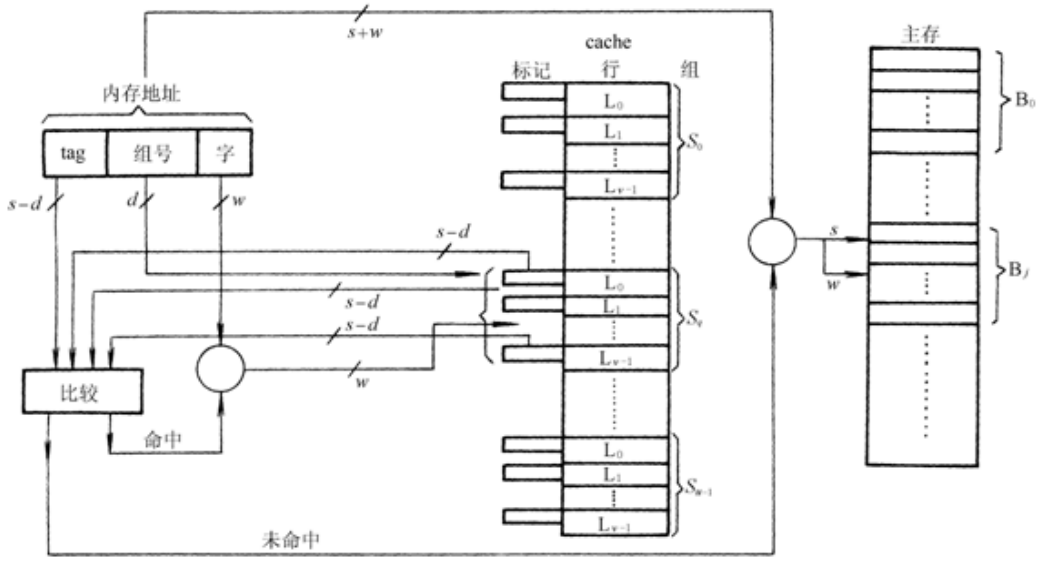
- ✓将Cache块分为u组，每组有v块
  - 主存块存放在哪个组是固定的，而存到组内哪个块则是灵活的
  - 组间直接映射，组内全相联映射
- ✓函数关系

  Cache块数 $m = u \times v$； 组号 $q = j \bmod u$
  - 主存的第j块内容映射到Cache的第q组中的某块
- ✓特点分析
  - 是全相连和直接映射的折中
  - v = 1，直接映射；u = 1，全相联映射
- ✓v的取值一般是2的幂，称之为v路组相联cache

组相联映射示意



(a) 组相联映射示意图(4组)



(b) 组相联cache的检索过程

相联度越高，Cache空间的利用率就越高，块冲突概率就越低，失效率也就越低。

Cache访问示例

访存地址：00010101

64个数据块的主存空间
（块大小4字节）

00 01 10 11

000000——
000001——
111111——

256个字节的
主存空间

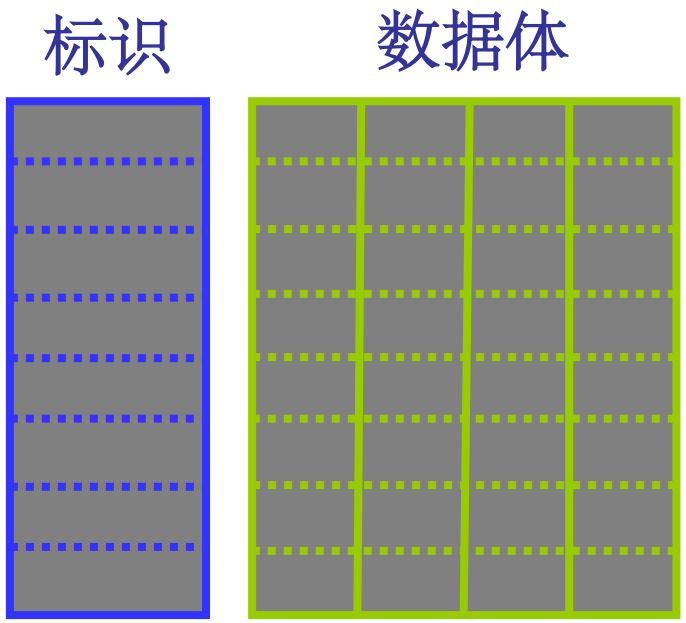
00000000
00000001
11111111

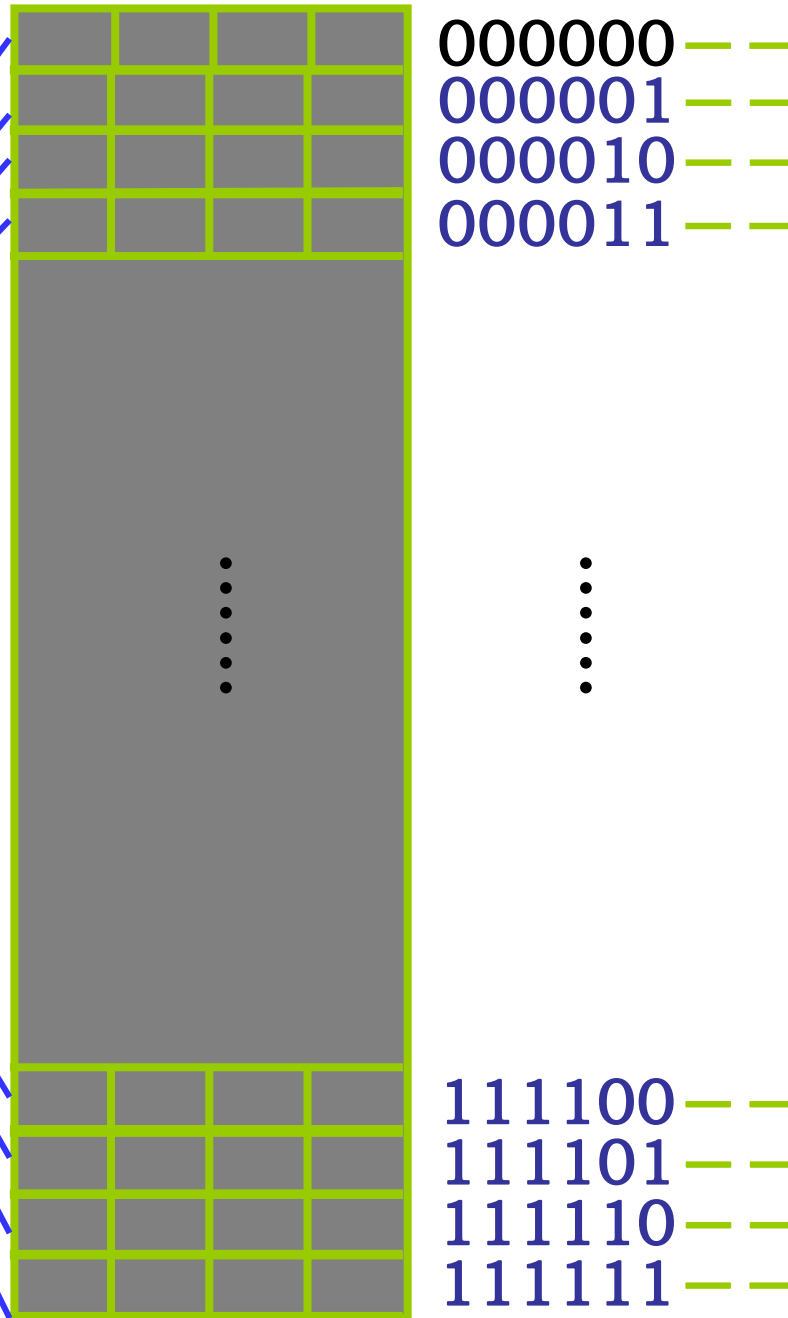
访存地址：000101 01

块地址　　块内地址

全相联

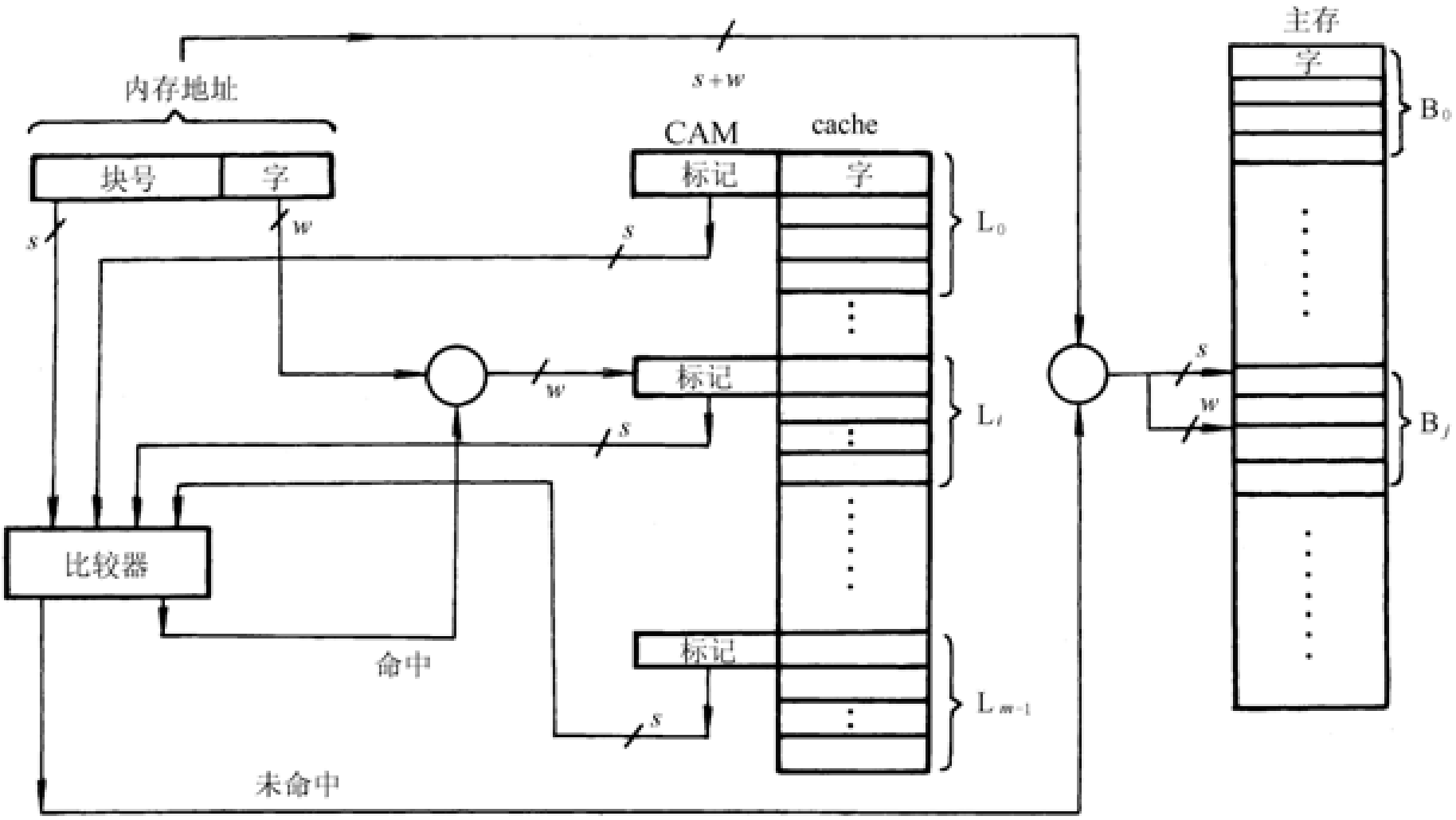
8个数据块的cache:





访存地址：

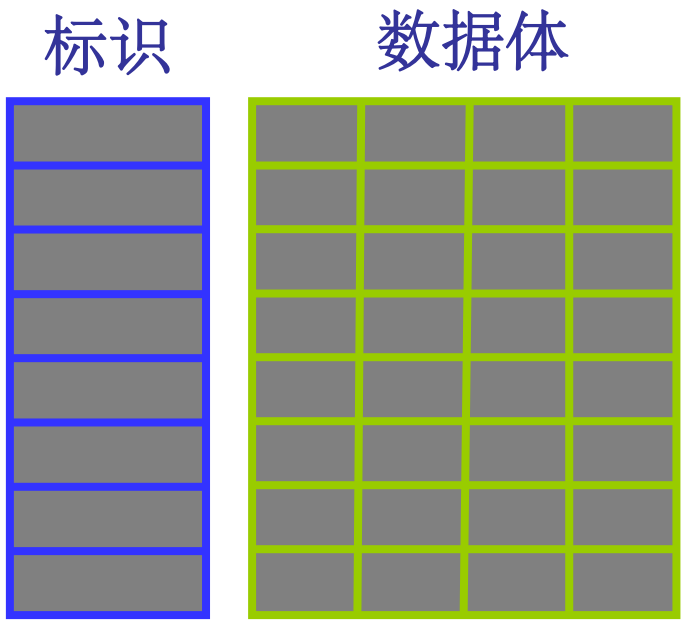
6	2
tag	Offset
XXXXXX	— —



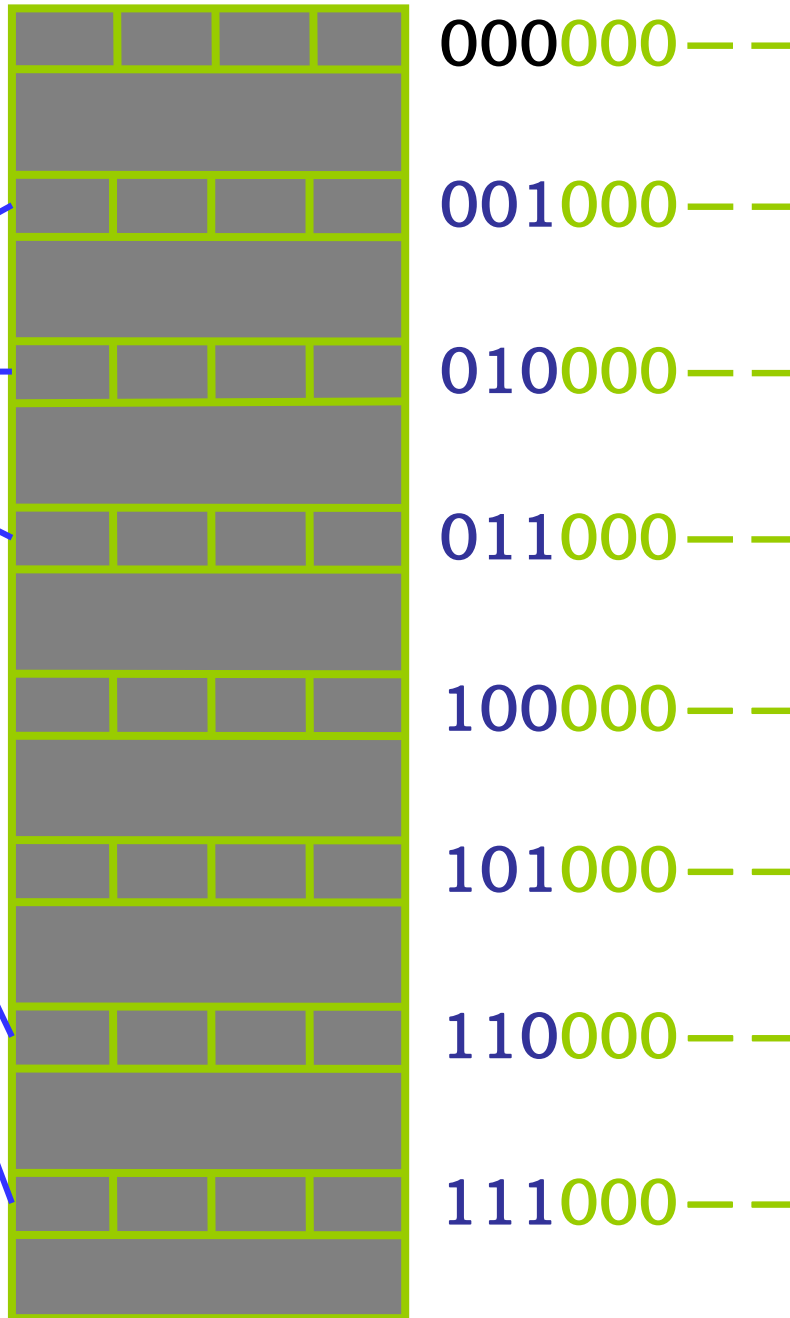
（b） 全相联cache的检索过程

直接映射

8个数据块的cache:

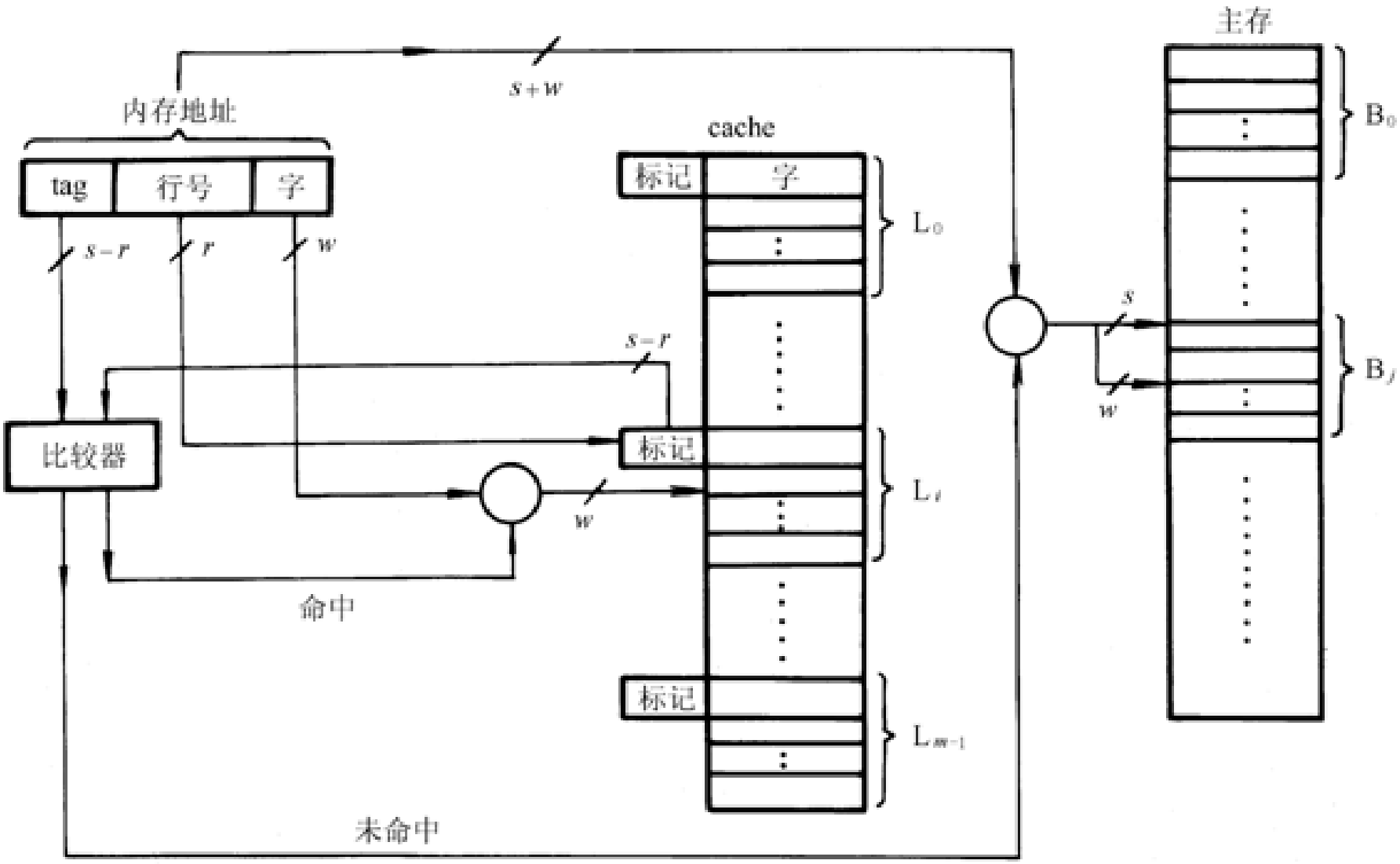


64个数据块的主存空间



访存地址:

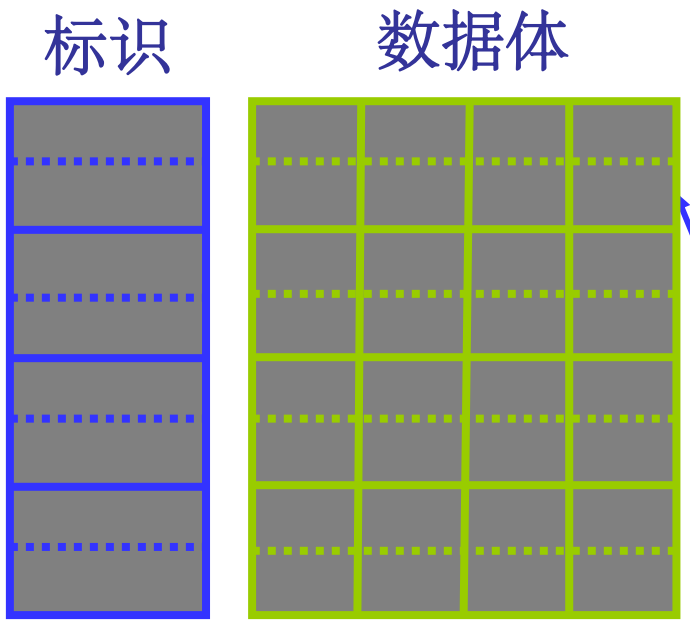
3	3	2
tag	index	Offset
XXX	000	– –



(b) 直接映射cache的检索过程

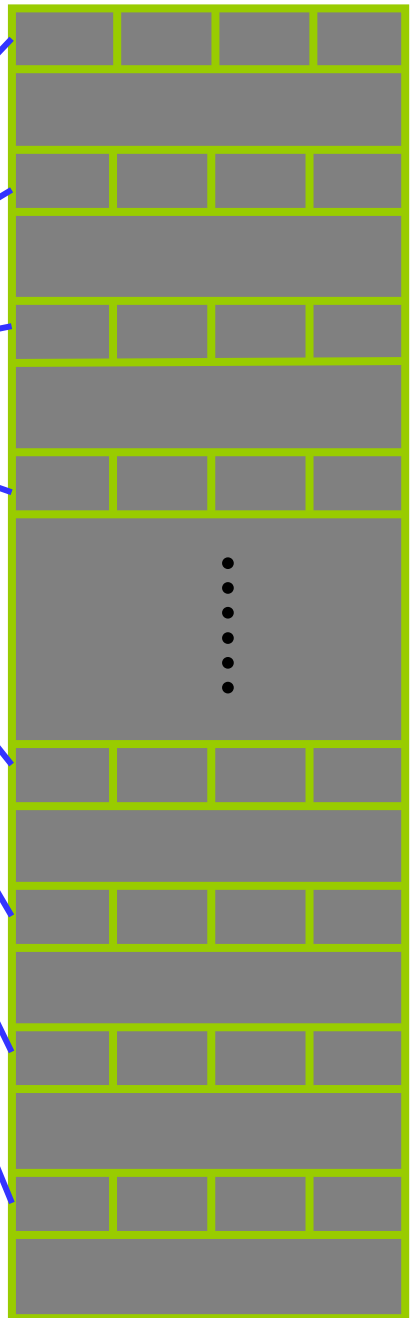
2路组相联

8个数据块的cache:



访存地址:

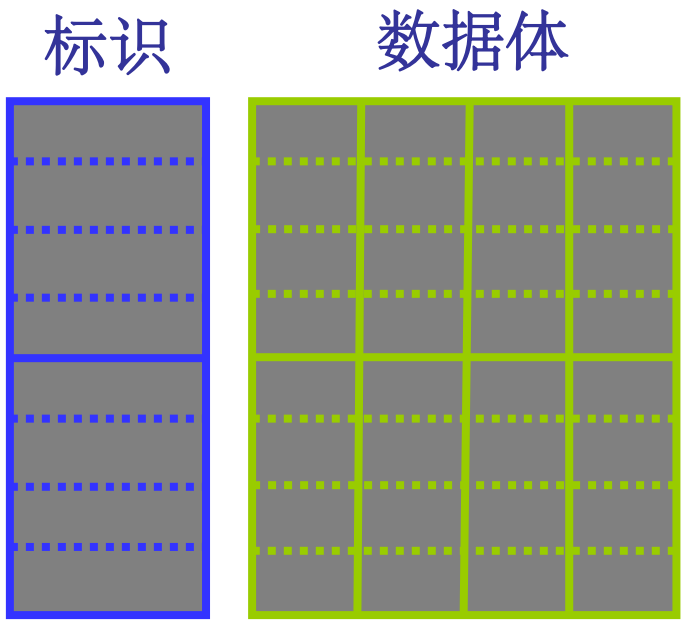
4	2	2
tag	index	Offset
XXXX	00	– –

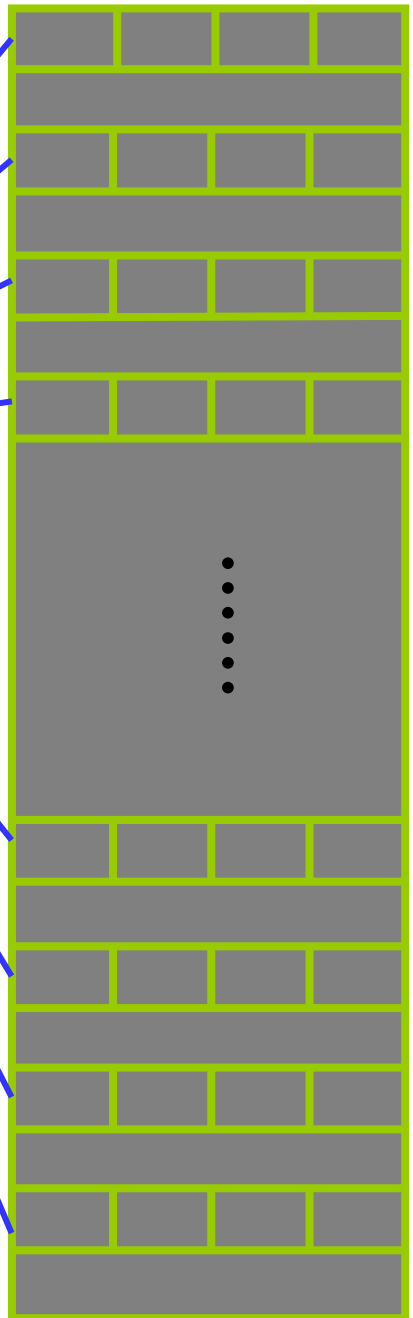


000000 – –
000100 – –
001000 – –
001100 – –
⋮
110000 – –
110100 – –
111000 – –
111100 – –

4路组相联

8个数据块的cache:



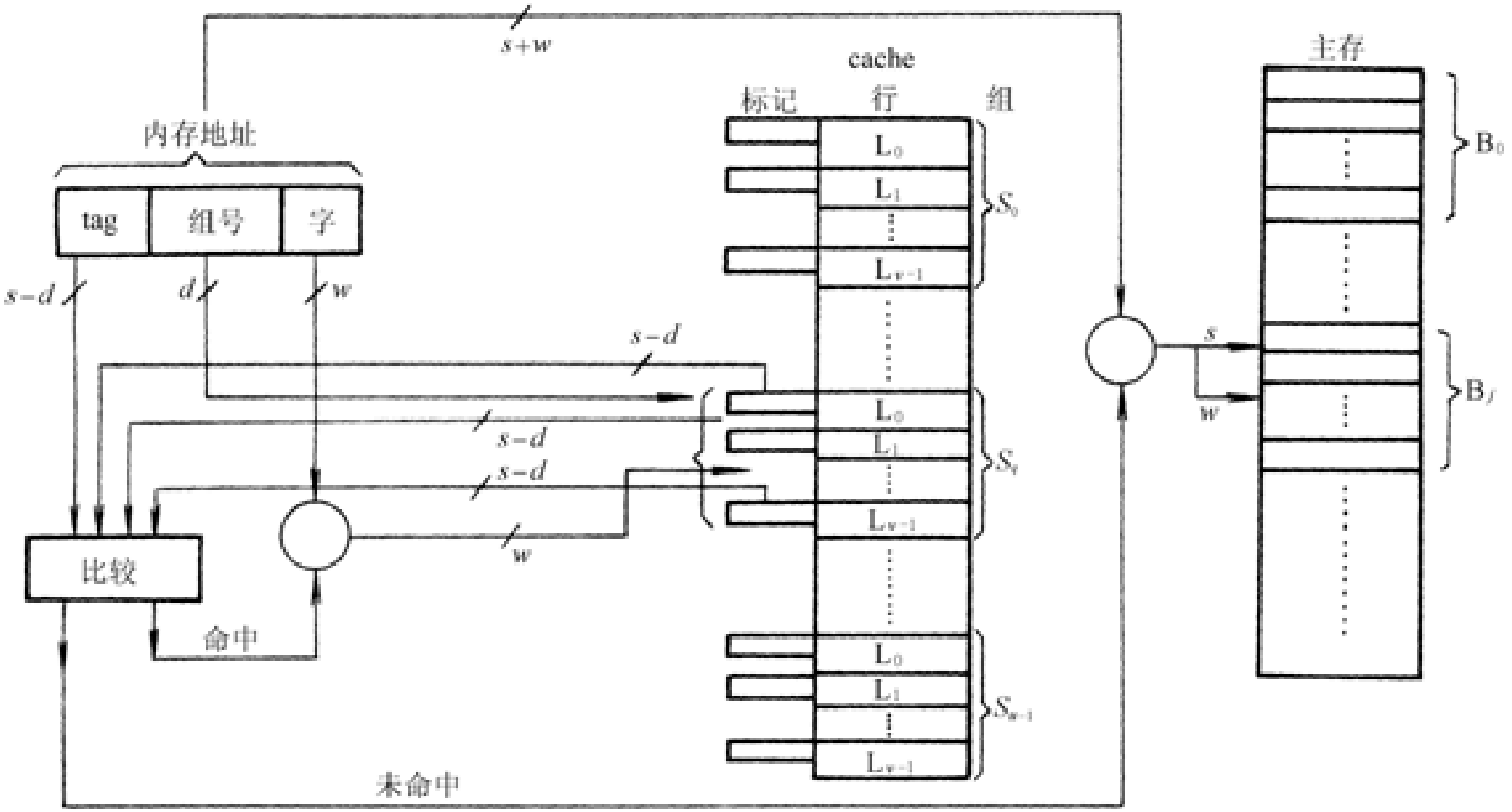


64个数据块的主存空间

000000——
000010——
000100——
000110——
111000——
111010——
111100——
111110——

访存地址：

5	1	2
tag	index	Offset
XXXXX	0	——



(b) 组相联cache的检索过程

地址映射方式例题（1）

- 例 假设主存容量为512KB，Cache容量为4KB，每个块为16个字，每个字32位。

  1）Cache地址多少位？可以容纳多少块？

  2）主存地址多少位？可以容纳多少块？

  3）在直接映射方式下，主存的第几块映射到Cache的第5块（设起始字块为第1块）？

  4）画出直接映射方式下主存地址字段中各段的位数

  解:

  1）Cache容量4KB = 2^{12}B，地址为12位。块数m = 4KB / (16×4B) = 64块

  2）主存容量512KB = 2^{19}B，地址为19位。块数 n = 512KB / (16×4B) = 8192块

  3）主存第5、2^6+5、2*2^6+5、…、2^{13}-2^6+5块映射到Cache第5块

  4）

主存块标记	Cache块地址	块内地址
7位	6位	6位

3.4 Cache替换策略

替换策略

✓ 替换产生的原因

- Cache容量比主存小得多，某个新的主存块要调入Cache时，其对应可存放的Cache块可能都已经被使用

✓ 替换策略与Cache组织方式密切相关

- 直接映射，直接替换对应Cache块即可
- 全相联和组相联映射，选择某一块——替换算法

✓ 常用的替换算法

- **先进先出FIFO算法**——选择最早调入Cache的块进行替换
- **随机法**——利用随机数产生器，随机选择被替换的块
- **最少使用LFU算法**——被访问的块计数器**增加1**，替换时选择计数值**最小**的块（同时清零其计数器）缺点：不能反映cache近期访问情况
- **最近最少使用LRU算法**——被访问的块计数器**置0**，其他块的计数器**增加1**，替换时选择计数值**最大**的块，符合cache的工作原理

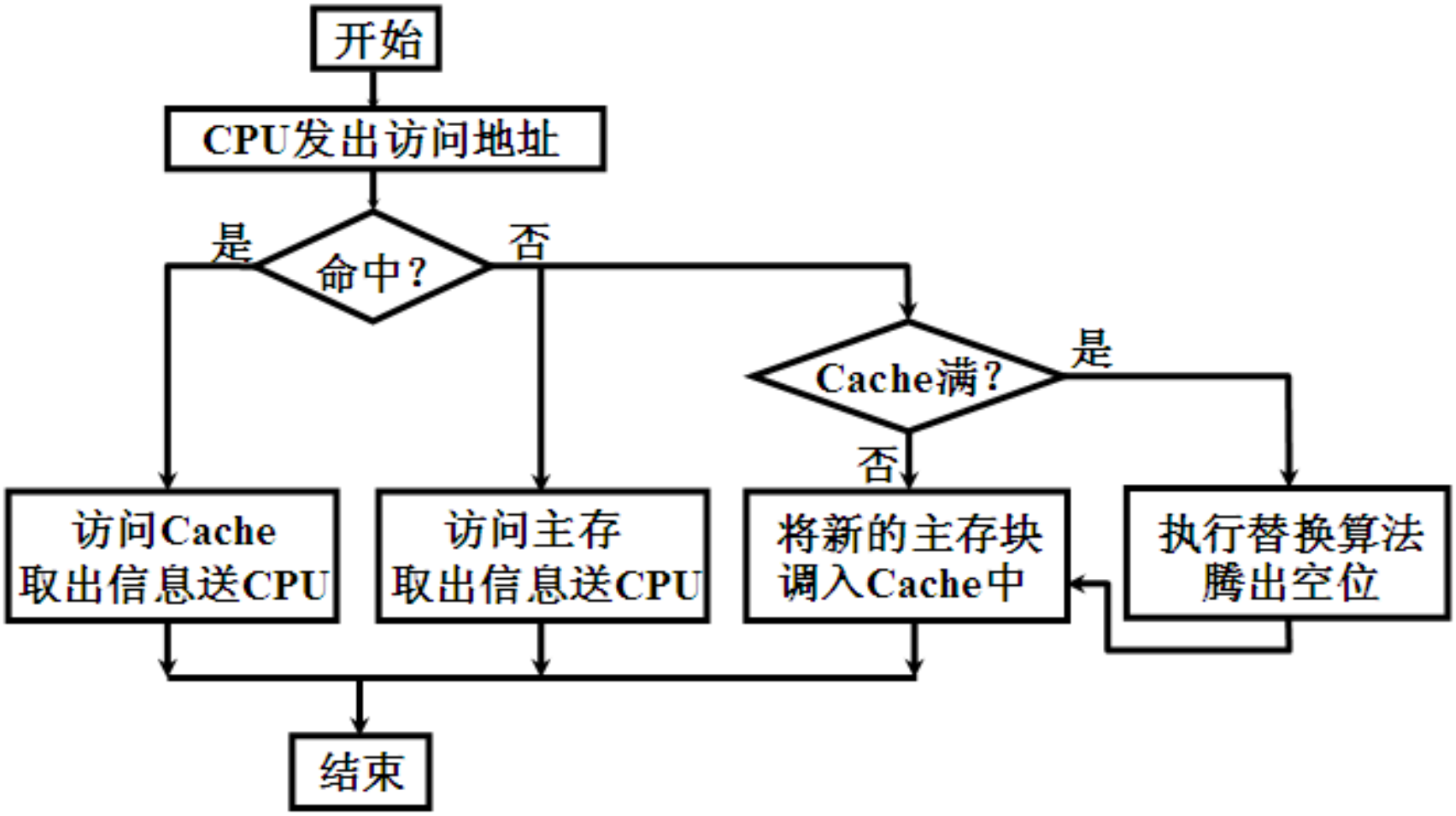
3.4 Cache替换策略

- 例 设cache有1、2、3、4共4个块，a、b、c、d等为主存中的块,访问顺序一次如下：a、b、c、d、b、b、c、c、d、d、a,下次若要再访问e块。
问，采用LFU和LRU算法替换结果是不是相同？

	LFU（最不经常使用）					LRU（近期最少使用）				
	说明	1块	2块	3块	4块	说明	1块	2块	3块	4块
a	a进入	1	0	0	0	a进入	0	1	1	1
b	b进入	1	1	0	0	b进入	1	0	2	2
c	c进入	1	1	1	0	c进入	2	1	0	3
d	d进入	1	1	1	1	d进入	3	2	1	0
b	命中	1	2	1	1	命中	4	0	2	1
b	命中	1	3	1	1	命中	5	0	3	2
c	命中	1	3	2	1	命中	6	1	0	3
c	命中	1	3	3	1	命中	7	2	0	4
d	命中	1	3	3	2	命中	8	3	1	0
d	命中	1	3	3	3	命中	9	4	2	0
a	命中	2	3	3	3	命中	0	5	3	1
e	替换a	1	0	0	0	替换b	1	0	4	2

3.5 Cache的读写策略

Cache的读操作过程



3.5 Cache的读写策略（2）

- Cache的存储一致性问题
  - ✓ Cache块内写入的信息，必须与被映射的主存块内的信息完全一致
- Cache的写策略（写命中）
  - ✓ 写直达法
    - 写操作时，数据既写入Cache，又写入主存
    - 写操作时间就是访问主存的时间
    - 优点：读操作时不涉及对主存的写操作，更新策略比较容易实现
  - ✓ 写回法
    - 写操作时，只把数据写入Cache而不写入主存
    - 当被写过的Cache块要被替换时才写回主存
    - 写操作时间就是访问 Cache 的时间，读操作 Cache 失效发生数据替换时，被替换的块需写回主存，增加了 Cache 的复杂性

3.5 Cache的读写策略（3）

- Cache的写策略（写失效）
  - 按写分配(写时取)：写失效时，先把所写单元所在的**块调入Cache**，再行写入。
  - 不按写分配(绕写法)：写失效时，直接写入下一级存储器而**不调块**。

- 写策略与调块

  写回法 —— 按写分配

  写直达法 —— 不按写分配

Cache的两种写策略（命中+失效）

- 写回 - 按写分配：
  - ✓命中：只写cache；
  - ✓失效：调块，只写cache；
- 写回 - 不按写分配
  - ✓命中：只写cache；
  - ✓失效：只写主存；
- 写直达 - 按写分配：
  - ✓命中：写cache写主存；
  - ✓失效：调块，写cache写主存；
- 写直达 - 不按写分配：
  - ✓命中：写cache写主存；
  - ✓失效：只写主存；

思考：Cache写策略各自在什么场景下有优势？

主要内容

1. 存储器概述
   - 1.1 存储器的分类
   - 1.2 存储系统的层次结构
2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉
3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
   - 3.2 Cache的基本结构
   - 3.3 Cache与主存的地址映射
   - 3.4 Cache存储块的替换策略
   - 3.5 Cache写策略
   - 3.6 Cache组织举例
4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法
5. 辅助存储器
   - 5.1 辅存概述
   - 5.2 磁记录原理与记录方式
   - 5.3 磁盘存储器
   - 5.4 光盘存储器
   - 5.5 FLASH存储器

4.1 虚拟存储器的基本概念

实地址与虚地址

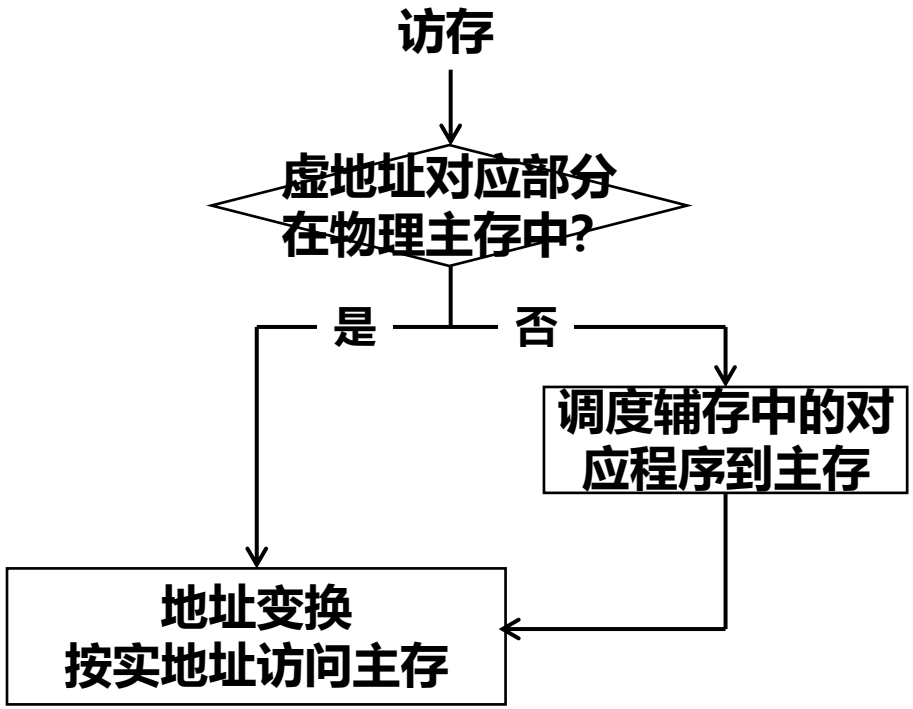
- ✓产生虚地址的原因
  - 程序所需的存储容量超过了实际的物理内存容量
  - 多用户多任务系统中，多用户或多任务共享内存并行执行，每个程序占用的实际内存在编程时无法确定，需运行时动态分配
- ✓虚地址：也称逻辑地址，指用户编程时使用的地址
  - 对应的存储空间称为虚存空间或逻辑地址空间
- ✓实地址：也称物理地址，指实际的物理主存的地址
  - 对应的存储空间称为主存空间或物理存储空间
- ✓虚地址到实地址的转换过程称为程序的重定位

编程时无需考虑物理主存的大小，也无需考虑程序如何存放

4.1 虚拟存储器的基本概念（2）

虚存访问过程

- ✓ 用户按虚地址编程并保存在辅存中。程序执行时，分配给每个程序一定的物理主存空间，由地址转换部件完成重定位；若分配的物理主存不够，则只调入当前正运行或将要运行的程序或数据，其他部分暂存在辅存中



4.1 虚拟存储器的基本概念（3）

- 虚地址空间与实地址空间
  - ✓ 虚地址空间可以远大于实地址空间
    - 用于提高存储容量
  - ✓ 虚地址空间可以远小于实地址空间
    - 通常出现在多用户多任务系统中，单个任务不需要很大的实地址空间，使用较小的虚存空间可以缩短指令中地址字段长度（有什么好处？）
  - ✓ 虚存空间大小仅依赖于辅存的大小
  - ✓ 在主存命中率较高时，虚存的访问时间接近于主存访问时间

每个程序可以拥有一个具有辅存的存储容量和近似主存访问速度的虚拟存储器

虚存与Cache

- 三级存储体系的两个层次

Cache ——— 主存 ——— 辅存

  - ✓ 两个层次间分别有**辅助硬件**和**辅助软硬件**负责地址变换和管理
  - ✓ Cache和主存构成**系统内存**，主存和辅存构成**虚拟存储器**

- 虚存与Cache的相同点
  - ✓ 出发点相同:为了提高存储系统的性价比，利用了程序运行时的局部性原理，使其性能接近高速存储器，而价格和容量接近低速存储器

虚存与Cache（2）

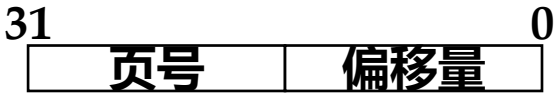
- 两个存储层次的不同点
  - ✓侧重点不同
    - Cache主要解决主存与CPU的速度差异问题
    - 虚存主要解决主存与辅存的存储容量的问题
  - ✓数据通路不同
    - CPU与Cache和主存之间均有直接访问通路，Cache不命中时可以直接访问主存
    - 虚存依赖的辅存与CPU之间没有直接数据通路，主存不命中时只能通过换页解决
  - ✓透明性不同
    - Cache完全由硬件管理，对系统程序员和应用程序员均透明
    - 虚存由软件（操作系统）和硬件共同管理，对系统程序员不透明
  - ✓未命中时的代价不同
    - 主存未命中时的系统性能损失要**远大于**Cache未命中时的损失

虚存要解决的关键问题

- 虚存机制要解决的关键问题
  - ✓地址映射问题
    - 虚地址如何变为主存物理地址（内地址变换）或辅存物理地址（外地址变换），以便换页
    - 主存分配、存储保护和程序重定位等问题
  - ✓替换问题
    - 决定哪些程序和数据被调出主存
  - ✓更新问题
    - 确保主存和辅存相应数据和程序的一致性

4.2 页式虚拟存储器

- 虚拟存储器可以分为页式、段式、段页式存储器
- 页式虚存地址概念
  - ✓ 逻辑页：虚地址空间被分成的等长大小的页
    物理页：主存空间被分成的与逻辑页同样大小的页
  - ✓ 虚地址：分为两个字段，高字段为逻辑页号，低字段为页内偏移量
    实地址：分为两个字段，高字段为物理页号，低字段为页内偏移量

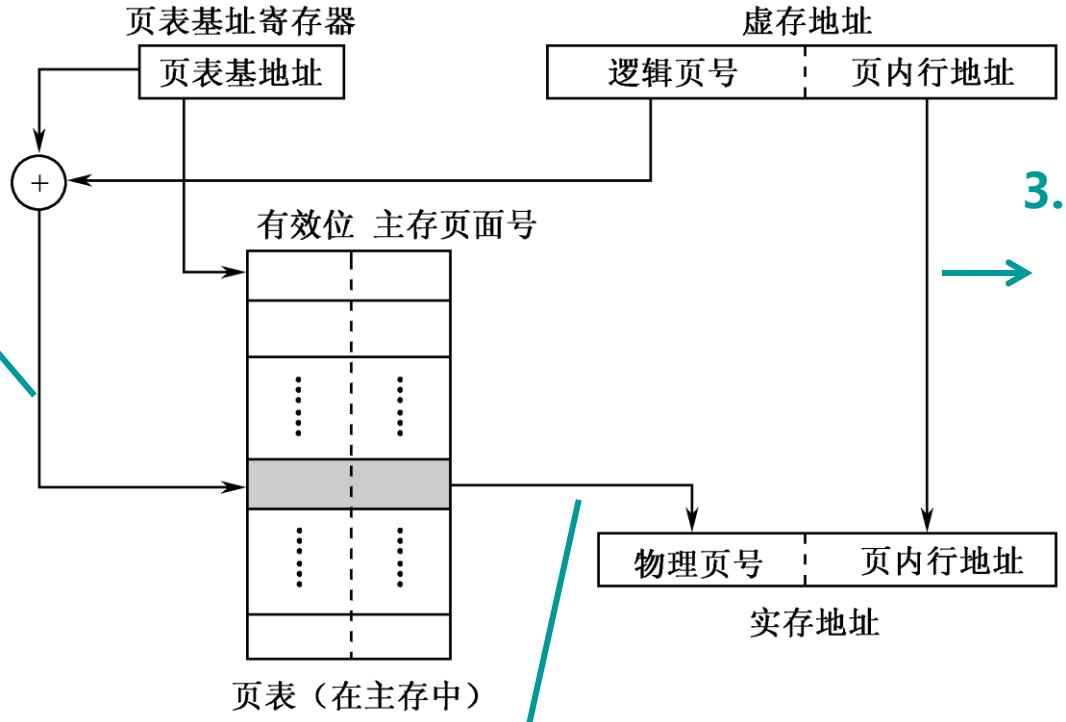


- 页表
  - ✓ 用于将虚地址（逻辑地址）转换成实地址（物理地址）
  - ✓ 每个进程对应一个页表
  - ✓ 页表的一个表项对应一个逻辑页
  - ✓ 表项的内容
    - 对应的逻辑页所在的物理页号
    - 指示该逻辑页是否已调入主存的有效位

页式虚存地址映射

□ 页式虚存地址映射

✓ 现代CPU通常有专门的硬件支持地址映射转换



1. 用逻辑页号作为页表内的偏移地址索引页表

2. 找到相应的物理页号，作为实地址的高字段

3. 虚地址的页内偏移量作为实地址的页内偏移量，构成完整的物理地址

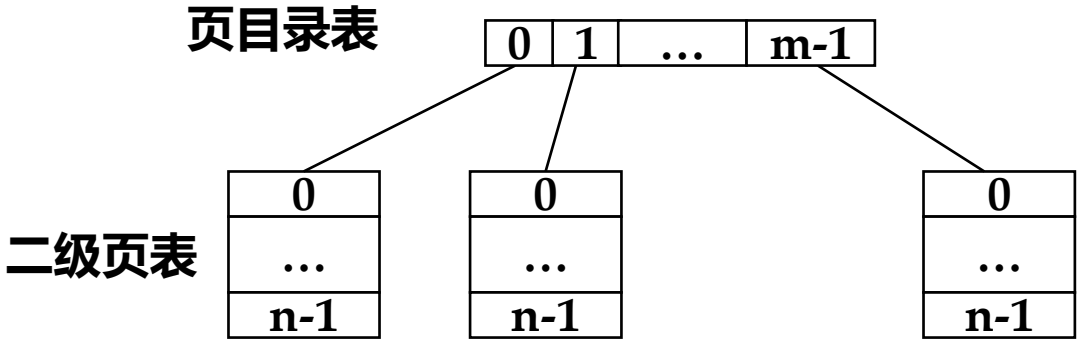
页表基址与页表分页

- 页表基址
  - ✓ 将页表基址保存在页表基址寄存器中，页表本身保存在主存中
  - ✓ 每个进程需要的页数不同，页表长度可变
  - ✓ 地址映射：页表基址 + 逻辑页号
- 页表的分页
  - ✓ 每个进程的页表可能很长
    - 设一个进程的虚地址空间为2GB，每个逻辑页大小为512B，则页表表项数（逻辑页数目）为 $2^{31} / 2^{9} = 2^{22} = 4M$;
  - ✓ 将页表存储在虚存中，通过对页表分页节省页表占用的主存空间
    - 进程运行时，其页表一部分在主存中，一部分在辅存中

二级页表

二级页表

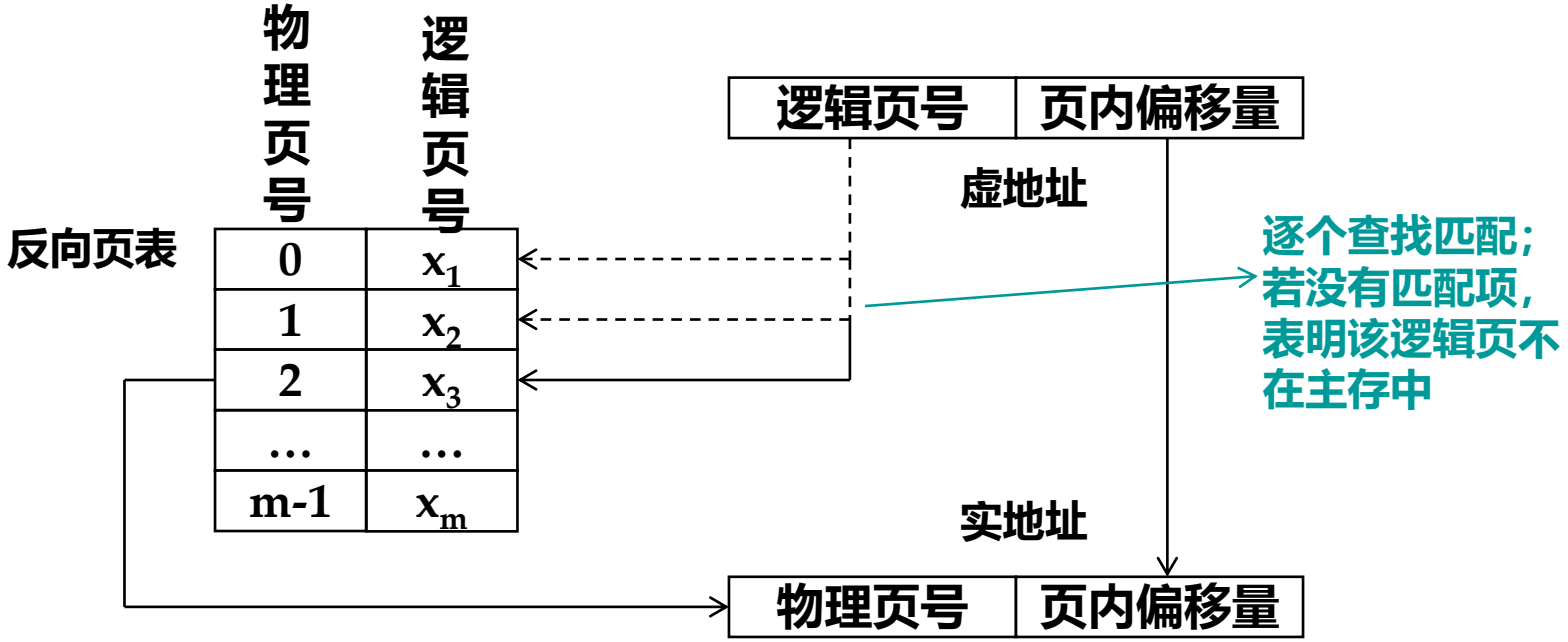
- ✓设置一个页目录表，其中的每个目录表项指向一个页表



一个进程最多可以有m×n个逻辑页

反向页表

- 反向页表
  - ✓ 一般情况：逻辑页少（主存空间），物理页多（辅存空间）
  - ✓ 在页表较大的系统（？）中，实现物理页号(少)到逻辑页号(多)的映射

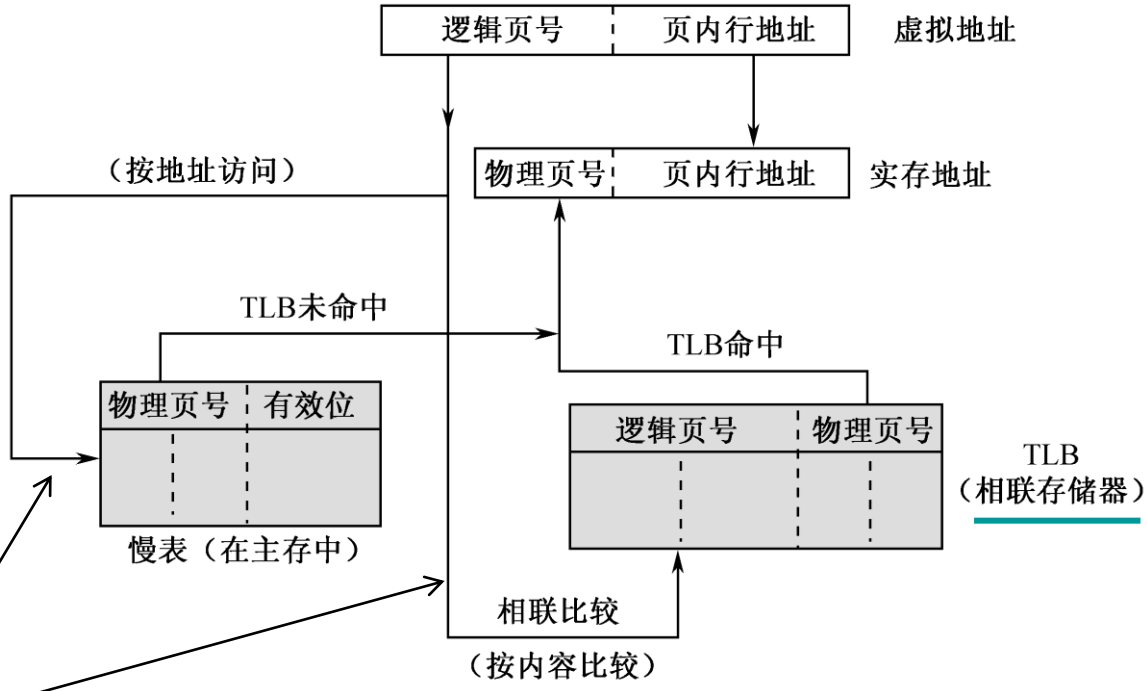


  - ✓ 主存的物理页数目决定了反向页表的长度（表项数）
  - ✓ 优点：页表所占空间大大缩小
    缺点：反向查找匹配的时间很长，性能受限于查找算法

快速地址变换——快表TLB

- TLB的使用（Translation Look-aside Buffer）
- 转换检测缓冲区
  - ✓为什么要使用快表?
    - 页表一般保存在主存中，即使逻辑页已经在主存中，也至少需要访问两次物理主存才能完成需要的访存操作，这使得虚存的存取时间加倍
  - ✓为减少访存次数，对页表进行二级缓存，将页表中最活跃的部分存放在高速存储器（如Cache）中，组成快表TLB
    - TLB：专用于页表缓存的高速存储部件
    - 保存在主存中的完整页表称为**慢表**

TLB的地址映射



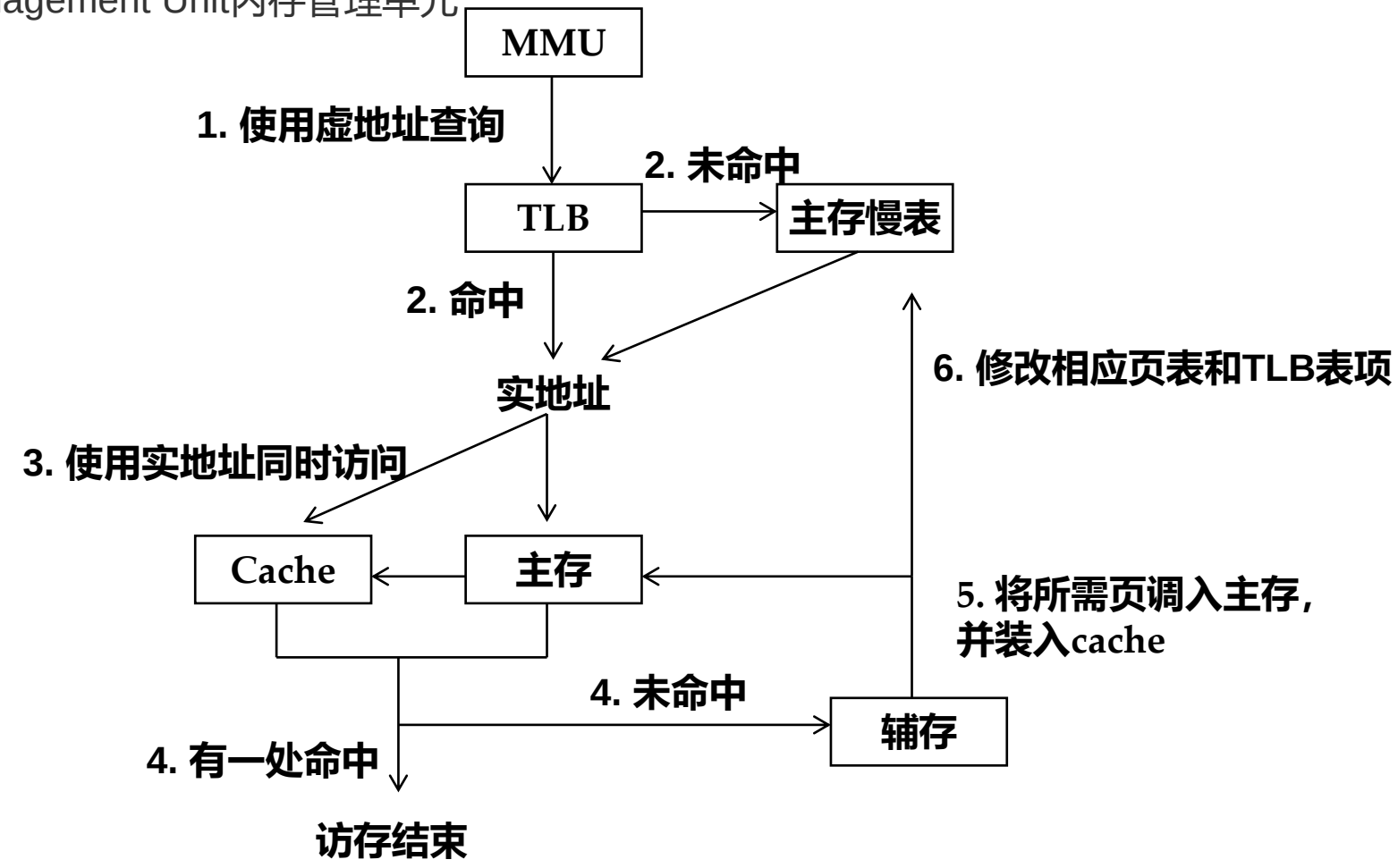
根据逻辑页号同时查找快表和慢表

- **按地址访问：将逻辑页号作为慢表表项的索引**
- **按内容比较：将逻辑页号作为匹配关键字在快表表项内容中进行查找**

根据程序局部性原理，多数虚存访问都将通过TLB完成，从而能够有效降低访存的时间延迟

完整的一次访存过程

Memory Management Unit内存管理单元



- 分页的优缺点
  - ✓优点：页长固定，便于构造页表和管理，不存在碎片
  - ✓缺点：页长与程序逻辑大小无关，不利于编程的独立性，以及复杂的存储管理操作
- 段
  - ✓按程序的自然分界划分的长度**可以动态改变**的区域
    - 通常将代码、操作数和常数等不同类型数据划分到不同的段中
  - ✓每个程序可以有多个相同类型的段
  - ✓段的虚地址组成：段号 + 段内偏移量
  - ✓分段为组织程序和数据提供了方便
    - 分页对程序员不可见，而分段通常对程序员是可见的

4.3 段式虚拟存储器（2）

段表

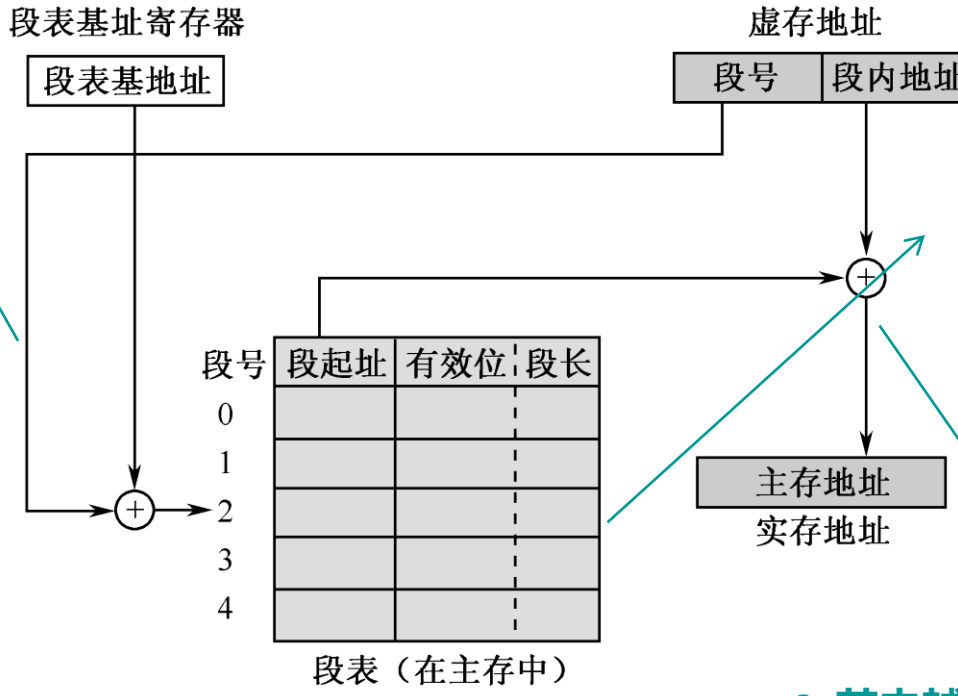
- ✓用于虚地址到主存实地址的变换
- ✓本身也是一个段，一般保存在主存中
- ✓每个程序设置一个段表，其中每个表项对应一个段
- ✓段表表项字段（和页表的表项页号+有效位区别）
  - 有效位——指明该段是否调入主存
  - 段起始地址——指明该段已在主存中时的首地址
  - 段长——记录该段的实际长度

段长字段用于保证访问某段的地址空间时，段内地址不会超过该段长度而导致地址越界，破坏其他段

4.3 段式虚拟存储器（3）

□段式虚存的地址映射

1. 以段号s为索引访问段表的第s个表项



2. 若有效，则比较段内地址d与该项的段长；若无效，则产生缺页中断，从辅存调入该页，并修改段表

3. 若未越界，则将段起址与段内地址相加，得到主存实地址；若越界，则产生地址越界中断

4.3 段式虚拟存储器（4）

- 段式虚存的优点
  - ✓段具有逻辑独立性，便于编译、管理、修改、保护和多程序共享
  - ✓段长可以按需动态改变，允许自由调度，能有效利用主存空间
- 段式虚存的缺点
  - ✓主存空间分配比较复杂
  - ✓容易在段间产生许多碎片，造成存储空间利用率降低
  - ✓段长不一定为2的整数次幂，必须进行段起始地址与段内偏移量的求和运算来得到物理地址
    - 相比于页式存储管理，段式存储管理需要更多的硬件支持

问：段表和页表哪个占用空间大?

4.4 段页式虚拟存储器

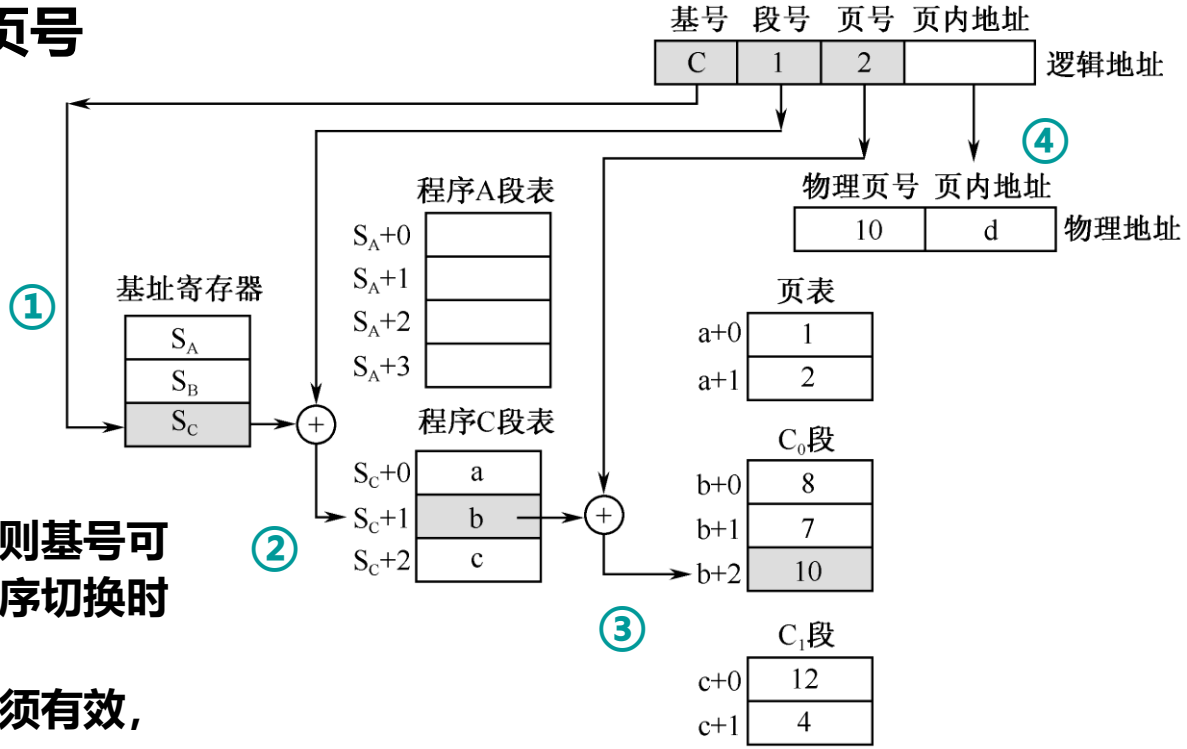
- 段式虚存和页式虚存的结合
  - ✓ 物理主存被等分为页
  - ✓ 程序先按逻辑结构分段，每个段再按照主存的页大小分页
  - ✓ 程序按页调入调出，但按段进行编程、保护和共享
- 段页式虚存管理
  - ✓ 每个程序通过一个段表和多个页表进行两级定位
  - ✓ 段表的每个表项对应一个段，其中有一个指针指向该段的页表
  - ✓ 页表指明该段各页对应的主存物理页，以及是否装入、是否修改等状态信息
- 段页式虚地址

基号N	段号S	段内逻辑页号P	页内地址偏移量D

多任务系统中，操作系统用来表明该程序在系统中的序号

段页式虚存的地址变换示例

三个程序A、B、C，基址分别为S_A、S_B、S_C
程序A有4个段，程序C有3个段
每个程序对应一张段表，每个段对应一张页表
段表的表项内容为相应页表的起始位置，页表的表项内容为相应的物理页号



说明：

1. 若系统只有一个基址寄存器，则基号可以不要，由操作系统负责在程序切换时修改基址寄存器内容
2. 每个找到的段表和页表表项必须有效，否则需中断操作

4.5 虚存的替换算法

- 替换的时机——需从辅存调页而主存已满时
- 主要的替换算法
  - ✓FIFO算法：First In First Out，先入先出
  - ✓LRU算法：Least Recently Used，最近最少使用
  - ✓LFU算法：Least Frequently Used，最不经常使用
- 虚存替换算法与Cache替换算法的不同
  - ✓Cache替换全部由硬件实现，虚存替换有操作系统的支持
  - ✓虚存缺页对系统性能的影响比Cache未命中要大很多

4.5 虚存的替换算法（2）

示例

- ✓ 假设主存只允许存放a、b、c三个页面，逻辑上构成a进c出的FIFO队列。某次操作中进程访存的序列是0,1,2,4,2,3,0,2,1,3,2（虚页号）。若分别采用FIFO算法、FIFO+LRU算法，请用列表法分别求两种替换策略情况下主存的命中率。

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

说明：

1. **FIFO算法中，页面都从a位置进入队列，在c位置换出队列**
2. **FIFO+LRU算法中，每当某个页面命中时，将其移动到FIFO队列入口位置a，将其被替换的时间延后**

存储系统主要内容

1. 存储器概述
   - 1.1 存储器的分类
     （缓存，主存，辅存）
   - 1.2 存储系统的层次结构
     （容量、速度权衡，局部性原理）

2. 主存储器
   - 2.1 主存概述
   - 2.2 半导体存储芯片简介
   - 2.3 SRAM存储器
   - 2.4 DRAM存储器
   - 2.5 ROM只读存储器
   - 2.6 存储器与CPU的连接
   - 2.7 并行存储（1）— 双端口存储器
   - 2.8 并行存储（2）— 多模块交叉

3. 高速缓冲存储器Cache
   - 3.1 Cache的基本原理
     （局部性原理）
   - 3.2 Cache的基本结构
     （Cache存储、替换、地址映射）
   - 3.3 Cache与主存的地址映射
     （全相连、直接映射、组相连）
   - 3.4 Cache存储块的替换策略
     （FIFO,随机法，LRU,LFU）
   - 3.5 Cache写策略
     (写命中-写直达&写回，写失效-按写分配&不按写分配)

4. 虚拟存储器
   - 4.1 虚拟存储器的基本概念
   - 4.2 页式虚拟存储器
     (TLB)
   - 4.3 段式虚拟存储器
   - 4.4 段页式虚拟存储器
   - 4.5 虚存的替换算法

5. 辅助存储器（IO）

本章总结

- 如何设计实现一个层次化存储系统?
  - ✓存储系统的功能。
  - ✓存储性能的优化。动机、方法（并行、层次化）
    - 缓存=>主存=>虚存
  - ✓存储容错机制（通用方法）
- 作业
- COD RISC-V edition: 5.5，5.11（Cache)

"study the past if you would define the future."

by Confucius