



北京邮电大学

BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS



Computer Organization and Architecture

Chapter 4

Cache Memory

School of Computer Science (National Pilot Software Engineering School)

AO XIONG (熊翱)

xiongao@bupt.edu.cn





Preface

We have learned:

- Basic Concepts and Computer Evolution 基本概念和计算机发展历史
- Performance Issues 性能问题
- Top level view of computer function and interconnection 计算机功能和互联结构顶层视图
 - Computer Components 计算机部件
 - Computer Function 计算机功能
 - Interconnection Structures 互联结构
 - Bus Interconnection 总线互连
 - QPI and PCIe QPI和PCIe



Review

- 问题1：总线为什么需要仲裁？仲裁方式一般有哪些？
- 问题2：计算机中具有存储功能的部件有哪些？存储器的存取方式有哪几种？
- 问题3：为什么需要cache？cache的基本原理是什么？



Preface

We will focus the following contents today:

- Cache Memory **cache存储器**
 - What are the characteristics of memory **存储系统的特点**
 - Why do we use cache? **为什么要用cache**
 - How does caching work? **Cache怎么工作**



Outline

- Key Terms
- Computer Memory System Overview 计算机存储系统概述
- Cache Memory Principles cache原理
- Elements of Cache Design cache设计要素
- Pentium 4 Cache Organization 奔4cache组织



Key terms (1)

- high-performance computing(HPC)高性能计算
- memory hierarchy: 存储器层次架构
- direct access: 直接访问
- random access: 随机存取
- access time: 存取时间, 访问时间
- cache memory: cache存储
- multilevel cache: 多级cache
- L1 cache: 一级cache
- L2 cache: 二级cache
- L3 cache: 三级cache



Key terms (2)

- Locality: 局部性
- spatial locality: 空间局部性
- temporal locality: 时间局部性
- cache hit: cache命中
- hit ratio: 命中率
- cache miss: cache缺失, cache未命中
- logical cache: 逻辑cache
- physical cache: 物理cache
- direct mapping: 直接映射
- associative mapping: 全相联映射
- set-associative mapping: 组相联映射
- Tag: 标记
- cache line: 和Tag关联的cache行



Key terms (3)

- split cache: 分立cache
- unified cache: 统一cache
- data cache: 数据cache
- instruction cache: 指令cache
- virtual cache: 虚拟cache
- replacement algorithm: 替换算法
- write back: 写回
- write once: 写一次
- write through: 写直达



Outline

- Key Terms
- Computer Memory System Overview 存储系统综述
- Cache Memory Principles cache原理
- Elements of Cache Design cache设计要素
- Pentium 4 Cache Organization Pentium4的cache组织



Memory

- Memory is a component used to store instructions and data 存储器是用于存放指令和数据的部件
- In a Von Neumann structured computer, memory does not distinguish between instructions and data 冯诺依曼结构的计算机中，存储器不区分指令和数据
- The performance of memory is constantly improving, but it is far from the speed of CPU 内存的性能也在提高，但是远远达不到CPU的速度
- Leading to a growing performance difference between them, which seriously affects the overall performance of the computer 导致它们之间的性能差异越来越大，严重影响了计算机的整体性能
- How to solve this problem? 如何解决这个问题？



Computer Memory System Overview

- Characteristics of Memory Systems

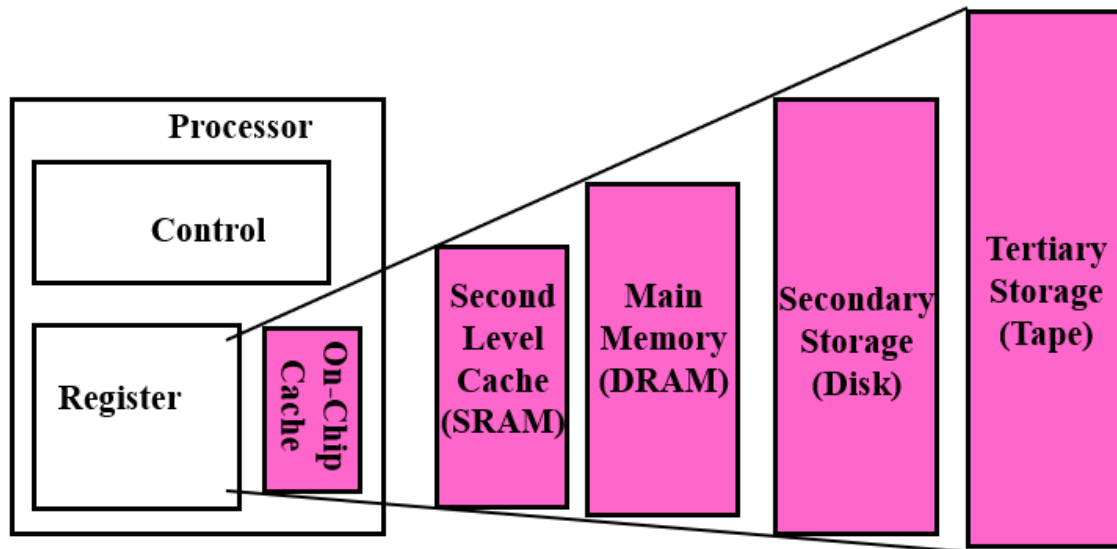
(存储系统的特性)

- The Memory Hierarchy

(存储系统层次结构)



Memory types 存储系统类型



- 和CPU的距离不一样
- 采用的存储技术不同
- 容量不一样
- 访问速度不同

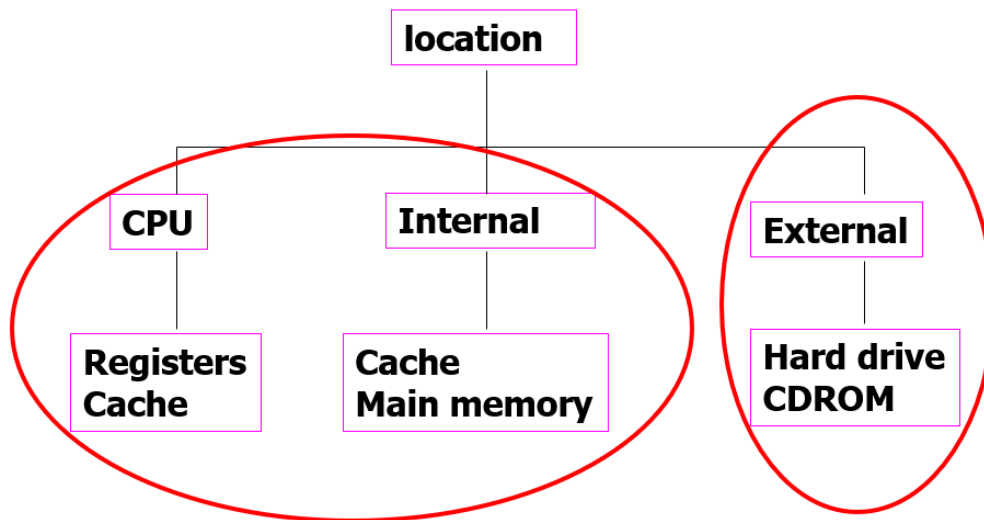


Characteristics of memory 存储系统的特征

- Location 位置
- Capacity 容量
- Unit of transfer 传送单位
- Access method 访问方法
- Performance 性能
- Physical type 物理类型
- Physical characteristics 物理特征
- Organisation 组织形式



Location 位置



- 存储位置主要是指存储器是在计算机的内部还是外部。
- 内部存储器通常包括：
 - CPU片上的寄存器以及片上Cache;
 - 内部存储器，以及片外cache。
- 外部存储器一般包括磁盘，磁带，CDROM等。处理器需要通过I/O控制器来访问这些存储，所以称为外部存储器



Capacity Unit 容量单位

- Bit: 0 or 1
- Byte: 8 bit
- Word size
 - The natural unit of organisation 主存中存储的自然单位
 - Not a fixed length 不是固定长度
 - Now it is generally 16 bit, 32 bit, or 64 bit 一般是16位，32位或64位
 - Most registers are one word 大部分寄存器是1个字长
 - Integers may be in words 整数一般也是一个字长
 - Instruction length is generally the same as the word 指令长度也和字长相同
 - Bus width 总线宽度



Capacity 容量

- Capacity: the amount of information can be stored 存储数据的数量
 - Number of addressable units 可寻址单元的数量
 - Unit size 单元大小
- Addressable unit: refers to a storage unit that can be uniquely addressed in memory 可寻址单元：指的是在内存中能够唯一寻址的存储单元
- Capacity: Addressable unit number*unit size 容量：可寻址单元数量*单元大小
 - Number of words 字的数量
 - Number of Bytes 字节数量



Unit of transfer 传送单位

- Internal
 - Usually governed by data bus width, word 由数据总线的带宽控制，一般跟字的bit数一样
- External
 - Usually a block which is much larger than a word 通常用块的方式，比字会大很多
- Addressable unit 可寻址单元
 - Smallest location which can be uniquely addressed 可唯一寻址的最小存储单位
 - Word internally 内部一般是字
 - Cluster on external disks 磁盘一般是用块为单位



Access methods – 1 访问方式1

- Sequential 顺序存取
 - Start at the beginning and read through in order 按照顺序从头到尾进行存储，读也需要按照顺序来读
 - Access time depends on location of data and previous location 存取时间依赖于数据的位置和当前所在位置的距离
 - e.g. tape
- Direct 直接存取
 - Individual blocks have unique address 每一个块都有一个唯一的地址
 - Access is by jumping to vicinity plus sequential search 先跳到临近的位置，然后再顺序检索
 - Access time depends on location and previous location 访问时间依赖于当前的位置和数据所在位置的距离
 - e.g. disk



Access methods – 2 访问方式2

- Random 随机存取
 - Individual addresses identify locations exactly 每个地址都有一个唯一的存储单元
 - Access time is independent of location or previous access 存取时间不依赖于数据的位置和前一次的存取
 - e.g. RAM
- Associative 关联存取
 - Data is located by a comparison with contents of a portion of the store 先比较再存取
 - Access time is independent of location or previous access 存取时间不依赖于数据的位置和前一次的存取
 - e.g. cache



Performance -1 性能1

- Access time 访问时间
 - Time between presenting the address and getting the valid data 从给出地址到得到有效数据的时间
 - For RAM: time to address the unit and perform the transfer 对于RAM，一般指的是将存储地址发给存储器到完成传输所用的时间
 - For non-RAM: time to position the R/W head over the desired location 对于非随机存取存储器，指的是将读写头定位到数据所在位置的时间。
- Memory Cycle time 存储周期
 - Applied to RAM 主要针对RAM
 - Time may be required for the memory to “recover” before next access 内存存在下一次访问之前需要一定的“恢复”时间
 - = access time + additional time required before next access 存储周期=访问时间+下一次存取开始之间需要的附加时间，用于进行存储器刷新操作



Performance -2 性能2

- Transfer Rate 传输速率
 - Rate at which data can be moved 数据移动的速率
 - For RAM: $R=1/(\text{cycle time})$ 对于RAM，等于存储周期时间的倒数。R指的是传输速率。
 - For non-RAM: $R=N/(T_N-T_A)$ 对于非RAM，等于读写的位数，除以读写的总时间-存取时间，也就是定位的时间。
 - N: number of bits 传输的总位数
 - T_N : time to r/w N bits; 读写N位的总时间
 - T_A : average access time 平均存取时间，也就是定位时间。



Physical types 物理类型

- Semiconductor 半导体
 - RAM
- Magnetic 磁介质
 - Disk & Tape
- Optical 光学材料
 - CD & DVD
- Others
 - Bubble 磁鼓
 - Hologram 全息照相



Physical characteristics 物理特性

- Decay or not 是否衰减
 - Decay: Flash disk
 - No decay: Hard disk
- Volatility or not 是否易失性
 - Volatility: Memory
 - No volatility: Hard disk
- Erasable or not 是否可擦除性
 - Erasable : Hard disk
 - No erasable: CD-ROM
- Power consumption 能耗



Organization 组织方式

- Memory organization
 - Physical arrangement of bits into words 如何将多个位组织形成字
 - How to organize multiple memory chips into a large capacity storage system 如何用多个存储器芯片，组织成大容量的存储系统
 - Discuss later



Computer Memory System Overview

- Characteristics of Memory Systems

(存储系统的特性)

- The Memory Hierarchy

(存储器层次结构)



Design Objectives 存储系统设计目标

- Major design objective of any memory system 存储系统设计的主要目标
 - To provide adequate **storage capacity** 提供足够的存储容量
(**how big?**) 能够达到多大的存储空间
 - An acceptable level of **performance** 可接受的存储器速度
(**how fast?**) 能达到什么速度
 - At a reasonable **cost** 以一个合理的代价
(**how much?**) 需要花费多少钱才能达到设计目标



Design methods 设计方法

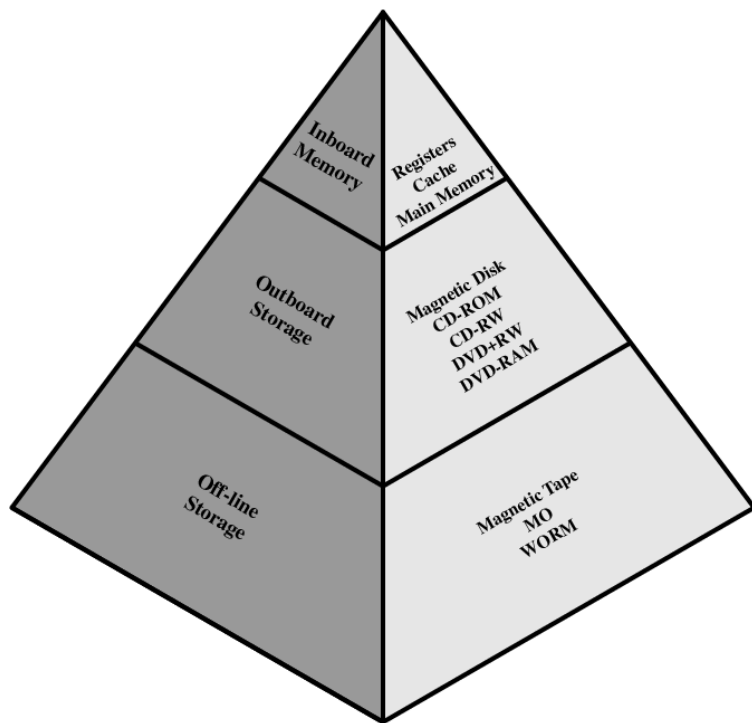
- There is always a trade-off among cost, capacity, access time
总是需要在代价、容量和存取时间这三个要素之间进行平衡
 - Faster access time \rightarrow greater cost per bit 速度越快，价格越贵
 - Greater capacity \rightarrow smaller cost per bit 容量越大，每位越便宜
 - Greater capacity \rightarrow slower access time 容量越大，访问时间越慢
- To meet higher capacity and lower cost requirements, memory speed should not be too high 为了达到大容量、低代价的要求，存储器速度就不能要求太高
- To meet performance requirement, capacity and price should not be too high 为了达到高性能的要求，容量和价格方面不能要求太高



The hierarchy 存储层次结构

- Single technology can not solve this problem 单一技术无法解决
- Use memory hierarchy 采用存储器层次架构
 - Consists of distinct levels of memory components 包括不同的层次的内存单元
 - Each level characterized by its size, access time, and cost per bit 每一层的存储单元都有一定的容量、访问时间和价格
 - Different levels of memory interact in some way 不同层次的存储器之间通过某种方式交互
- The Goal of memory hierarchy 存储层次设计的目标
 - Try to basically match the speed of the CPU and the memory closest to it 尝试使CPU和最靠近它的存储器的速度基本匹配

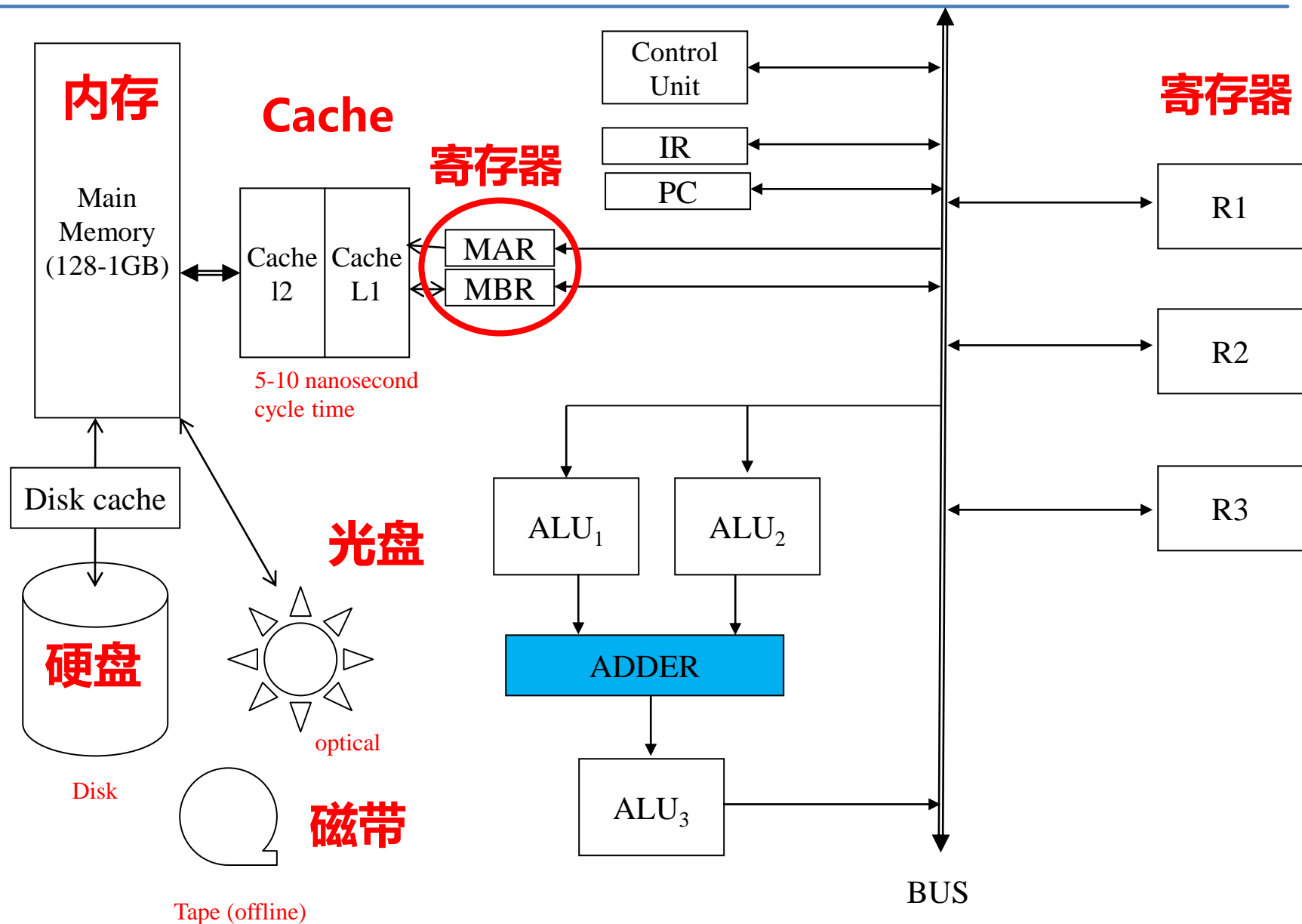
Memory Hierarchy – Diagram 存储体系框图



- 典型的存储层次结构包含三层：
 - 最接近CPU的是主板上的存储系统，包括寄存器、cache和主存
 - 第二层是板外存储系统，比如磁盘、CDROM，DVD等
 - 最外层是离线存储，一般磁带就属于这种存储层次



Memory Hierarchy 存储体系结构





Hierarchy List 存储器架构中的各类存储设备

- Registers 寄存器
- L1 Cache 第一级cache
- L2 Cache 第二级cache
- Main memory 内存，也叫主存，存储器
- Disk cache 磁盘缓存
- Disk 磁盘
- Optical 光盘
- Tape 磁带



Summary 小结

- There are many types of storage device in computer, including registers, memory, cache, hard drives, etc. Their storage location, capacity, performance, price, and other characteristics have their own 计算机内部的存储器种类很多，包括寄存器、内存、**cache**、硬盘等，他们的存储位置、容量、性能、价格等都有各自的特点
- The goal of storage systems is to have large capacity, high performance, and affordable prices 存储系统的目标是容量大、性能高、价格便宜
- A single technology cannot achieve these three goals, balance is ed 单一技术无法达到这三个目标，需要在三者之间寻找一个平衡
- Using multi-level storage systems is an effective way to achieve goals 采用多层级的存储系统是实现目标的有效方法



Outline

- Key Terms
- Computer Memory System Overview 存储系统综述
- Cache Memory Principles cache原理
- Elements of Cache Design cache设计要素
- Pentium 4 Cache Organization Pentium4的cache组织

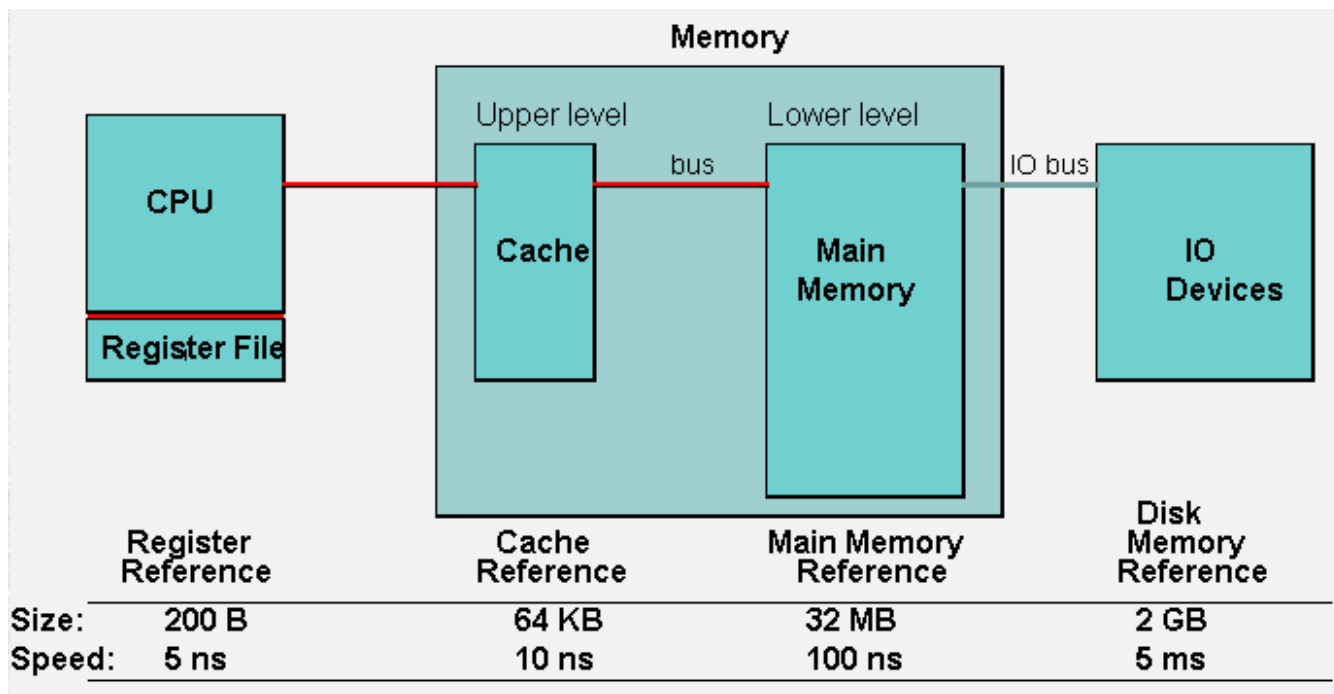


So you want fast? 你需要多快的存储系统呢?

- What is cache?
 - A high-speed memory faster than general RAM 比一般的RAM快的一种高速存储器
 - Using expensive but faster SRAM technology 用昂贵但较快速的SRAM技术
- How to use cache?
 - It is possible to build a computer which uses only static RAM 你可以用SRAM来构造你的计算机
 - This would be very fast 这会很快
 - This would cost a very large amount 而且这个会非常昂贵
- Hierarchy is the solution 采用层次架构



Reference 参考架构



- CPU中有一组寄存器，寄存器很小，存取速度为5ns。
- 直接和CPU相连的是高速缓存，大小在几十KB，存取速度10ns
- 再往上为主存，容量为几十MB，存取速度为100ns级别
- 再往上就是通过IO模块连接的磁盘存储了。容量GB级别，存取速度毫秒级



Locality of Reference 局部性引用

- The cache capacity is small, but the memory is large. Why Cache Works? **cache**容量很小，而内存很大。为什么**Cache**能发挥作用
- The memory hierarchy works because of Locality of reference: 存储器层次架构之所以有效，是因为局部性引用的原理
 - Most programs are highly sequential; the next instruction usually comes from the next memory location. 程序是高度顺序的，下一条指令通常来自下一个内存位置
 - Data is usually structured, and data in these structures normally are stored in continuous memory locations 数据通常是结构化的，这些结构中的数据通常存储在连续的内存位置
 - Short loops are a common program structure, especially for the innermost sets of nested loops. This means that the same small set of instructions is used over and over. 短循环是一种常见的程序结构，特别是对于嵌套循环的最内层集合。这意味着同一小部分指令被反复使用



Cache Memory Principles **cache原理**

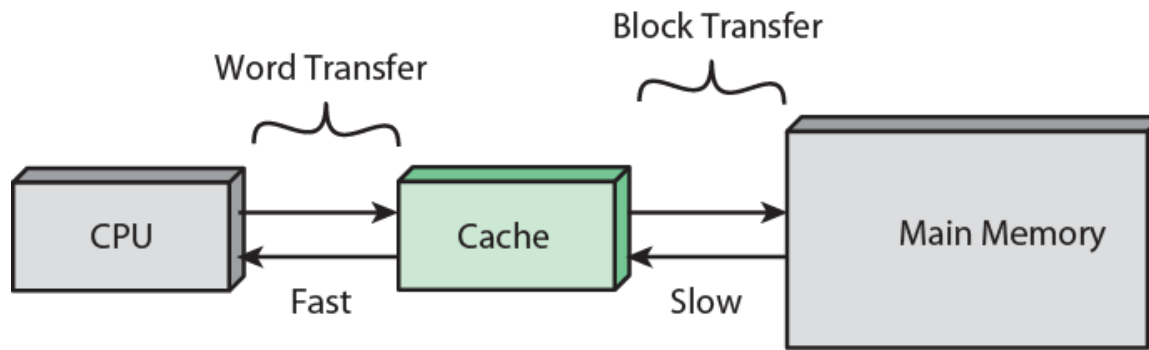
- Large and slow main memory together with a smaller, faster cache memory to improve the performance **将容量小、速度快的cache, 和容量大、速度慢的主存相结合, 来提高存储体系的性能**
- Cache stores a small part of memory data **cache上保存的是内存一小部分的数据**
 - If the data required by the CPU is in the cache, it will read the cache directly without reading the memory **如果CPU需要的数据在cache上, 就直接读取cache, 不需要读取内存**
 - If the required data is not in the cache, read the memory and update the cache **如果需要的数据不在cache上, 需要读取内存, 并更新cache**



Characteristic 特点

- Compare to the memory, cache memory: 相对于主存, cache具有如下特点:
 - Small amount 容量小
 - Fast access - Operate at nearly the speed of the processor 速度快,基本上和处理器差不多
 - Sits between normal main memory and CPU 逻辑上在主存和CPU之间
 - May be located on CPU chip or module 物理位置可能在CPU片上
 - Very expensive 价格比较贵

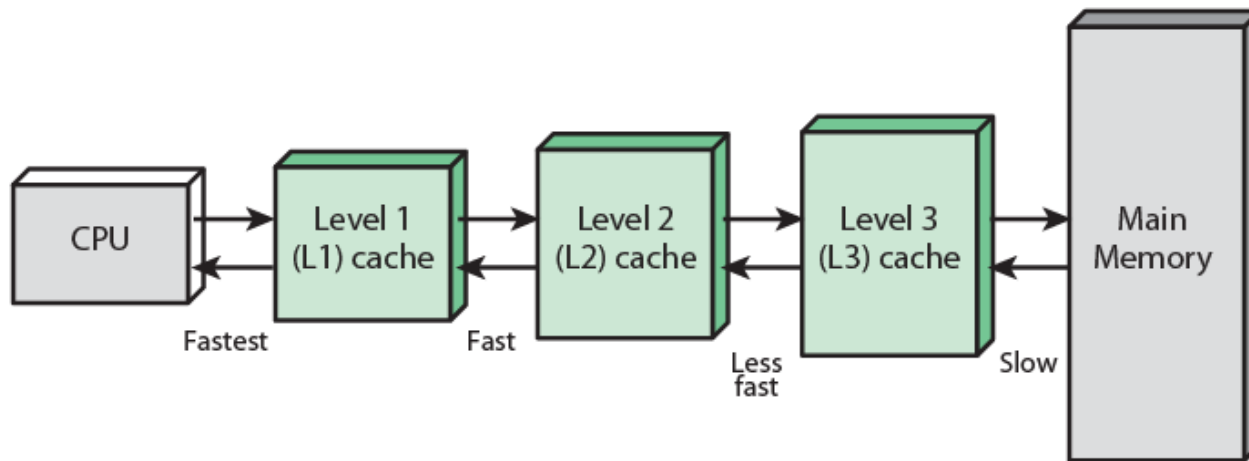
Single cache



(a) Single cache

- 一个容量大、速度较慢的内存，同时还有一个容量小但是速度快的cache。
- Cache中存放的是主存中一部分数据的副本
- cache和内存之间是通过块来交换数据的。
- 当CPU需要访问主存时，它先去查看cache中是否有这个数据，
 - 如果有，就把这个字从cache取过来。
 - 如果不在cache中，就把内存中包含这个字的块传送给cache，再从cache传给CPU。

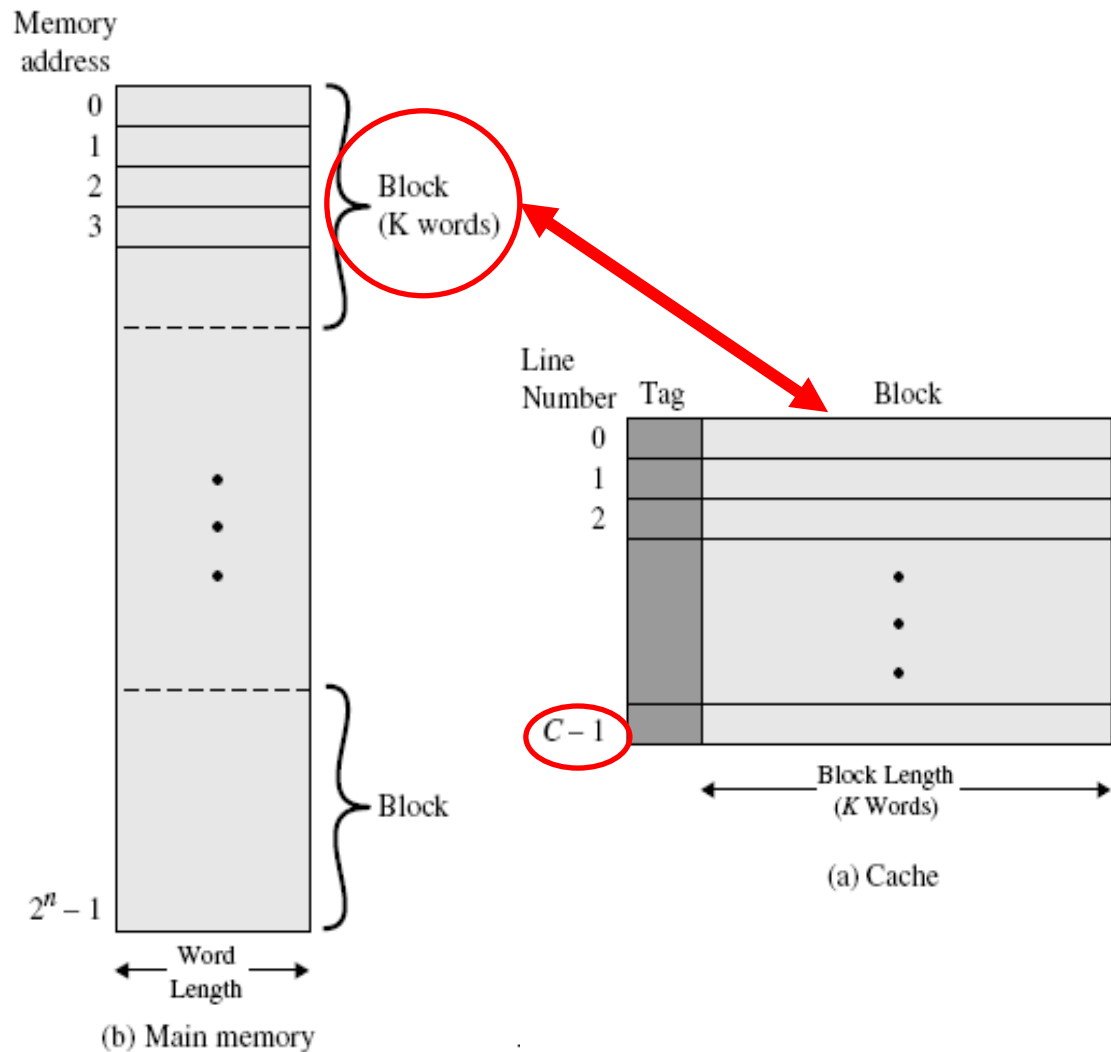
Multi-level cache



(b) Three-level cache organization

- 三级cache的方案
- 第二级cache比第一级cache的容量大，速度慢
- 第三级cache的容量比第二级又要大一些，速度则更慢一点
- 每一级是下一级的cache

Cache/memory structure **cache和主存的映射**



- cache和内存之间通过块来交换数据
- 主存中每行是一个字，连续k行组织成一个块
- Cache中按照行来组织，每个行的大小等于主存中一个块的大小，再加上一些标志位
- Cache的行映射到内存中的一个块

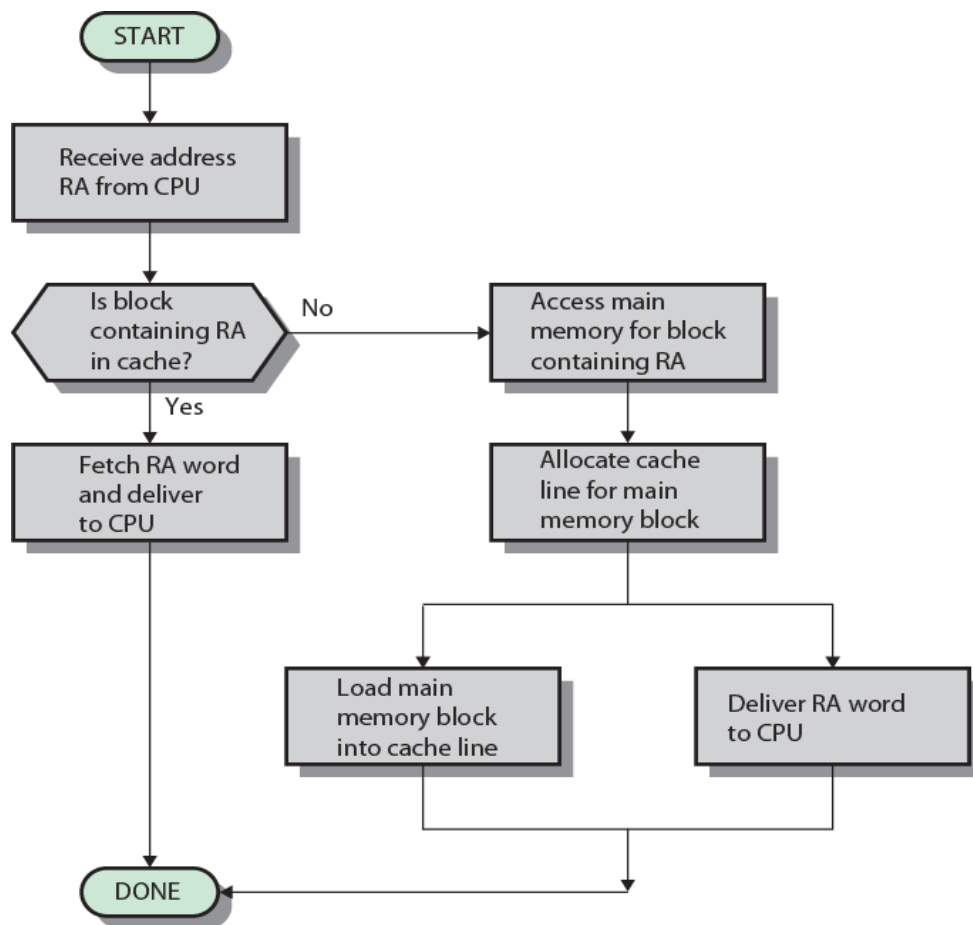


Cache operation **cache操作**

- CPU requests contents of memory location **CPU向存储器请求内容**
- Check cache for this data **检查数据是否在cache中**
 - If present, get from cache (fast) **如果在的话，直接从cache中取数据就可以了。这样速度就很快**
 - If not present, read required block from main memory to cache **如果不在的话，就需要从存储器中读取块到cache中**
 - Then deliver from cache to CPU **然后再把数据从cache传递给CPU**
 - May also be transmitted to the cache and CPU at the same time **也可能是同时给cache和CPU**
- Cache includes tags to identify which block of main memory is in each cache slot **所以，cache中需要包括一个标识，用来标识内存中的哪个块在cache中**

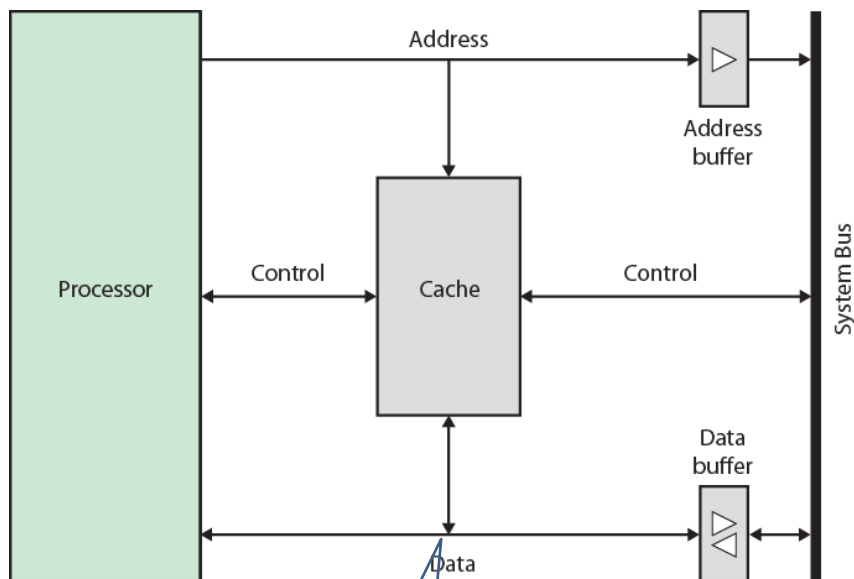


Cache read operation – flowchart 流程图



- 从CPU中得到需要访问的数据地址
- 根据数据地址，分析数据所在的块是否在cache中
- 如果在cache中，直接访问cache，把数据传送给CPU
- 如果不在Cache中，需要访问数据所在的块，并在cache中分配一行给这个数据块
- 把数据所在的块存到cache中，同时把数据传给CPU
- 存Cache和传数据给CPU，可以并行完成。

Typical cache organization



**内存来的数据同时
给Cache和CPU**

- Cache通过地址线、数据线和控制线与CPU相连，地址线和数据线同时和地址缓存、数据缓存相连，缓存连到系统总线上，主存也连到系统总线上
- 当Cache命中的时候，地址缓存和数据缓存都不启用，数据直接从cache传到CPU。系统总线上没有数据。
- 当cache没有命中的时候，地址放到系统总线上，存储器将数据通过数据缓存提交给cache和CPU
- 有的结构中，数据从内存到cache之后，必须要通过cache才能提交给CPU



Summary 小结

- The principle of locality in programs provides possibilities for the use of cache 程序的局部性原理给cache的使用提供了可能性
- Insert fast and small capacity cache into CPU and memory 在CPU和内存中插入速度快、容量小的cache
- Save a copy of some data in memory in the cache Cache中保存内存中部分数据的副本
- Before the CPU accesses memory, first check if the data is in the cache. If it is in the cache, access the cache to obtain the data, otherwise access the memory CPU访问内存前，先查看数据是否在cache中，如果在cache中，访问cache获得数据，否则访问内存
- If most of the access can be in the cache, it can greatly improve the performance of the storage system 如果大部分的访问都能在cache中，可以大幅度提高存储系统的性能



Outline

- Key Terms
- Computer Memory System Overview 存储系统综述
- Cache Memory Principles cache原理
- Elements of Cache Design cache设计要素
- Pentium 4 Cache Organization Pentium4的cache组织



Elements of cache design **cache设计要素**

- Cache Addresses **Cache地址**
- Cache Size **Cache 容量**
- Mapping Function **映射功能**
- Replacement Algorithm **替换算法**
- Write Policy **写策略**
- Line Size **行大小**
- Number of Caches **Cache 数目**



Virtual address 虚拟地址

- What is virtual address?
 - A technology of Memory Management 内存管理的一种技术
 - Allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available. 允许程序从逻辑角度去访问内存，不考虑物理内存的大小
 - A memory expansion technology 一种内存扩展技术
 - It will not change the size of the physical space of memory, but it can make the way of program accessing memory more flexible. 不改变物理空间大小，但是使得程序能够更灵活地访问内存



Virtual address 虚拟地址

- How to use virtual address?
 - Address fields of machine instructions contain virtual addresses.
机器指令中的地址域包含的是虚拟地址
 - Memory access requires an actual physical address 内存访问需要有实际的物理地址
 - Hardware memory management unit (MMU) translates each virtual address into a physical address. 有一个硬件的内存管理工具 MMU负责翻译地址



Cache addresses **cache地址**

- Where does cache sit?
 - Between processor and virtual memory management unit **位于处理器和虚拟内存管理单元之间，称为逻辑cache**
 - Between MMU and main memory **位于虚拟内存管理单元和主存之间，称为物理cache**

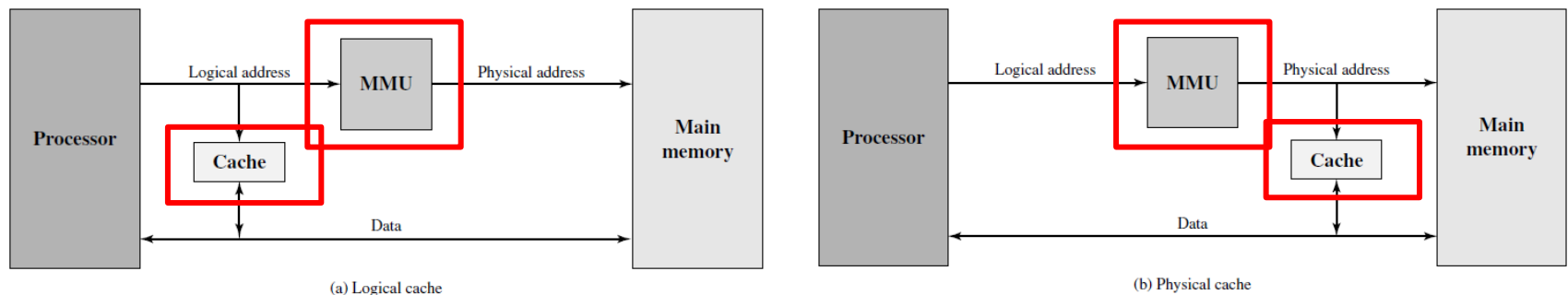


Figure 4.7 Logical and Physical Caches

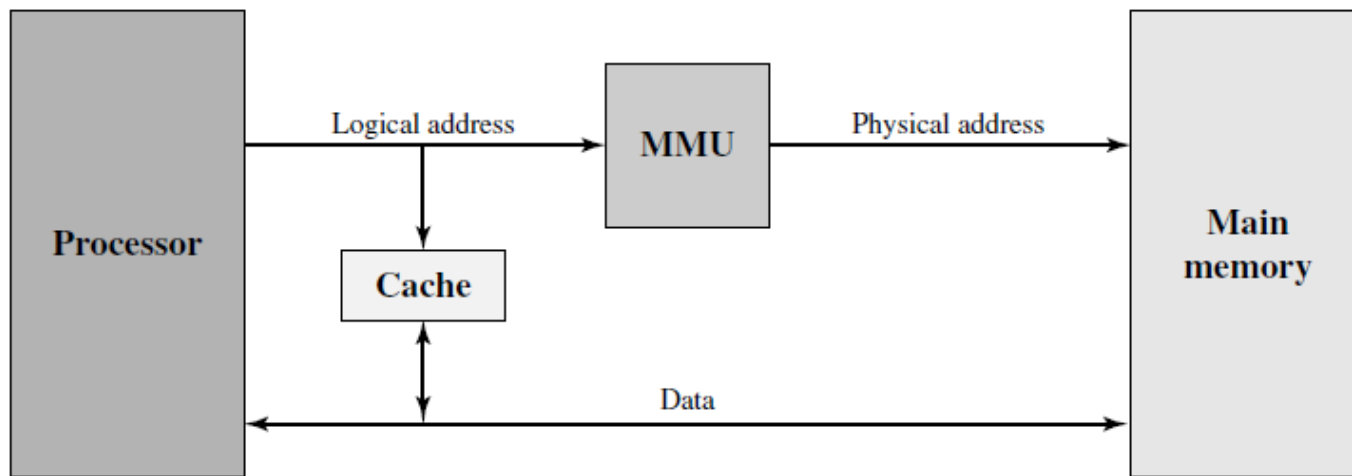


Logical cache 逻辑cache

- Logical cache (virtual cache) stores data using virtual addresses 逻辑cache 也叫做虚拟cache，用虚拟地址来保存数据
 - Processor accesses cache directly, without going through the MMU(Memory management Unit) 处理器直接访问cache，不经过存储器管理单元
 - Cache access faster, before MMU address translation cache访问快，在MMU翻译地址之前就可以访问
 - Virtual addresses use same address space for different applications 虚拟地址对于不同的应用使用相同的地址空间
 - Must flush cache on each context switch 在程序切换的时候，必须要更新cache



Logical cache 逻辑cache



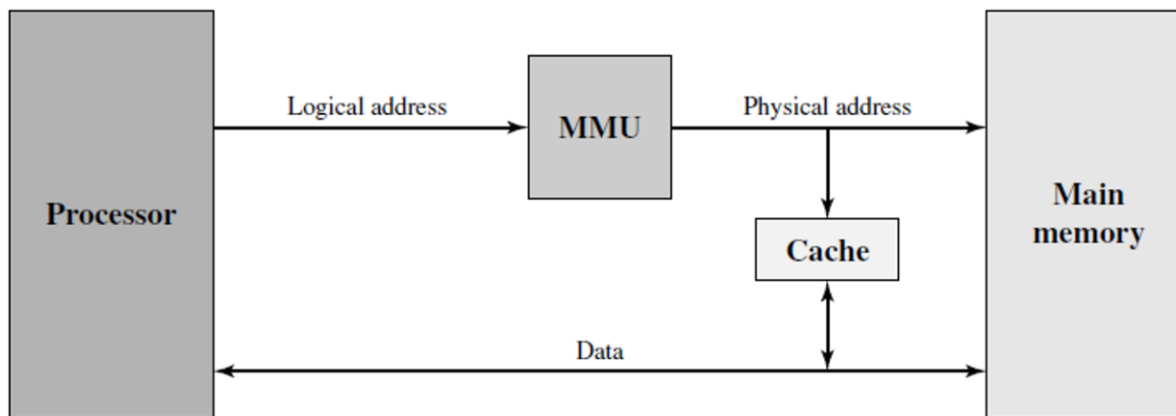
(a) Logical cache

- 处理器请求数据的逻辑地址直接发给了cache，不经过MMU
- 如果Cache命中，速度很快
- 如果cache没有命中，再通过MMU，转换成物理地址，到主存中去取数据，然后更新cache



Physical cache 物理cache

- Physical cache stores data using main memory physical addresses
物理cache使用主存的物理地址来保存数据
- Logical address is translated into the physical address by the MMU, and then the cache is accessed 逻辑地址先翻译, 再访问cache



(b) Physical cache



Cache size does matter **cache的容量很重要**

- Capacity has a great influence on performance **cache容量对性能影响很大**
- Why not very big cache?
- Cost **价钱**
 - More cache is expensive **cache越大，价钱越贵**
- Speed **速度**
 - More cache is faster (up to a point) **cache越大，速度越快**
 - Checking cache for data takes time **同时，cache的检查花的时间也越多**



Cache size of main computer 主要计算机的cache大小

Processor	Type	Year of Introduction	L1 cache	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA _b	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—



Cache hits and misses **cache命中和失效**

- A cache hit **cache 命中**
 - The CPU requested data is in cache, thus cache can pass the contents to the CPU **CPU请求的数据在cache中，cache直接把数据给CPU**
- A cache miss **cache失效**
 - The requested data is not in cache, it must be read from the memory **CPU请求的数据不在cache中，需要从存储器中读数据**
- Cache performance is often measured in terms of the hit/miss ratio **cache的性能经常用命中率/失效率来衡量**



Ratio of hit and miss 命中率和失效率

- Hit 命中
 - Hit Ratio: The fraction of memory access found in the upper level. 命中率: 上层访问数据的需求在本层完成的比例
 - Hit Time: Time to access the cache which consists of 访问cache的时间, 包括:
 - Access time + Time to determine hit/miss 访问时间+决定是否命中的时间
- Miss 失效
 - Miss Ratio = $1 - (\text{Hit Ratio})$ 失效率 = $1 - \text{命中率}$
 - Miss Penalty(惩罚): Time to replace a block in the cache + Time to deliver the missed data to the processor 失效惩罚: 替换cache块的时间+传送数据给处理器的时间



Example 1

- Suppose that the processor has access to two levels of memory.

假设处理器有两级存储器

- Level 1 contains 1000 words and has an access time of $0.01 \mu\text{s}$. 第一级包含1000个字，速度0.01us
- Level 2 contains 100,000 words, access time of $0.1 \mu\text{s}$ 第二级有100,000个字，速度0.1us
- Assume that if a word to be accessed is in level 1, then the processor accesses it directly 假定如果数据在第一级，处理器直接就可以得到



Example 1

- If it is in level 2, then the word can be transferred to level 1 and processor 如果在第二级，同时给第一级和处理器
 - H: hit ratio, the word is found in the faster memory H是命中率，表示字在第一级存储器
 - T1: access time to level 1 T1: 访问Level1的时间
 - T2: access time to level 2 T2: 访问Level2的时间
- If $H=95\%$ 假定命中率H为95%
- What is the average access time? 平均访问时间



Answer 1

- The average time to access a word is 访问一个字的平均访问时间为:

$$- 0.95 \times 0.01 \mu s + 0.05(0.01 \mu s + 0.1 \mu s) = 0.0095 + 0.0055 = 0.015 \mu s$$

数据在L1上，直接访问L1的时间

数据不在L1，需要先检索L1，然后再去L2上取数据



Example 2

- Suppose that the processor has access to cache and memory.
假定处理器需要访问**cache**和存储器
 - The cache access time: 100ns **cache**的访问时间是**100ns**
 - The memory access time: 1000ns 存储器的访问时间是**1000ns**
- If we want the average access not to exceed 15% of cache access time, what minimum cache hit ratio should be? 如果我们需要的平均访问速度不能超过**cache**访问速度的**115%**，那么最小的命中率必须达到多少？



Answer 2

- The average access time 平均访问时间

$$= H \cdot T_c + (1-H)(T_c + T_m) < 1.15 \cdot 100$$

$$= H \times 100 + (1-H) \times 1100 < 115$$

- So, $H > 98.5\%$

为什么需要 $T_c + T_m$?



Cache mapping **cache映射**

- Cache capacity is generally much smaller than the main memory capacity **cache的容量一般都会远远小于主存的容量**
- Need some functions to map memory blocks to cache rows **需要某种功能实现内存的块到cache行的映射**
- Three different types of mapping functions: **三种映射类型**
 - Direct mapping **直接映射**
 - Associative mapping **相联映射**
 - Set associative mapping **组相联映射**
- Mapping function is implemented in hardware **映射由硬件实现**



Mapping parameter of example 映射参数

- Following parameters are used for the discussion of the following examples 以下参数用于后面例题的讨论
- Cache
 - 64k Byte, line of 4 bytes 64K字节，每行4个字节
 - 16k (2^{14}) lines, need 14 bit address 总共16k行，需要14位地址寻址
- Memory
 - 16M Bytes, block of 4 bytes 16M字节，每块4个字节
 - Byte Addressable, need 24 bit address 字节寻址，需要24位地址
- Memory size is 64 times of cache size 内存大小是cache的256倍



Direct mapping 直接映射

- Each block of main memory maps to only one cache line 存储器中的每个块只能映射到cache中唯一的一行
 - If a block is in cache, it must be in one specific place 如果内存中的块在cache中，那么它只能在cache中的特定的一行
- Address is in two parts 地址包括两部分
 - Least Significant w bits identify unique word 低 w 位确定块中的字
 - Most Significant s bits specify one memory block 高 s 位确定内存中的块
- The MSBs are split into a cache line field r and a tag of $s-r$ (most significant) 高 s 位在cache行中分为行域 r 和标记域 $s-r$



Direct mapping address structure 直接映射地址结构

- 16M Bytes main memory need 24 bit address 24位地址
- 2 bit word identifier (4 byte block) 2位字标识, 每块有4个字节
- 22 bit block identifier 22位块标识
 - 14 bit slot or line 14位行或者块地址
 - 8 bit tag (=22-14) 8位标志
- No two blocks in the same line have the same Tag field 映射到同一cache行中的所有内存块, 不会有相同的标志
- Check contents of cache by finding line and checking Tag 通过找到行并检查标记, 确定cache是否命中

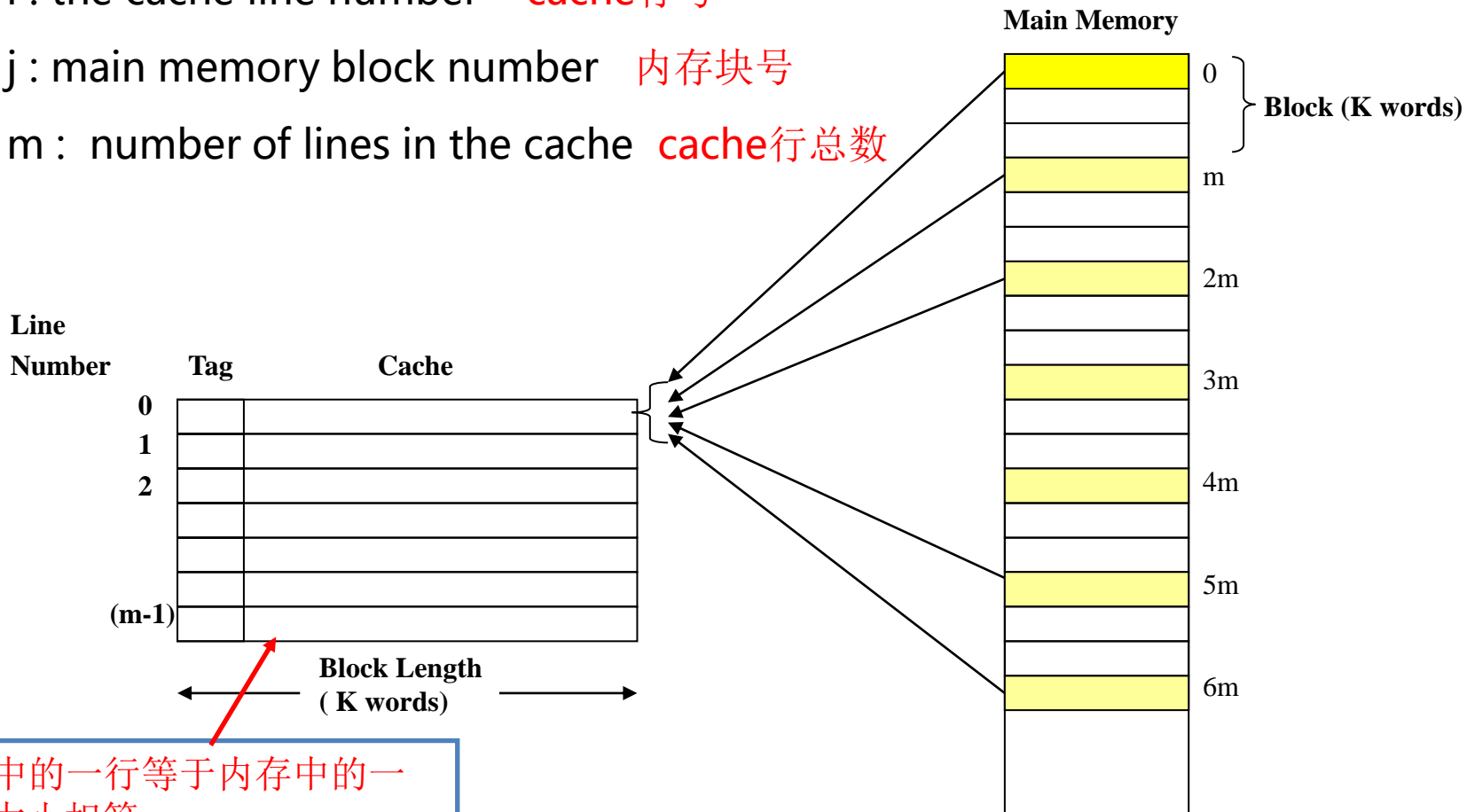
Tag s-r	Line or Slot r	Word w
8	14	2



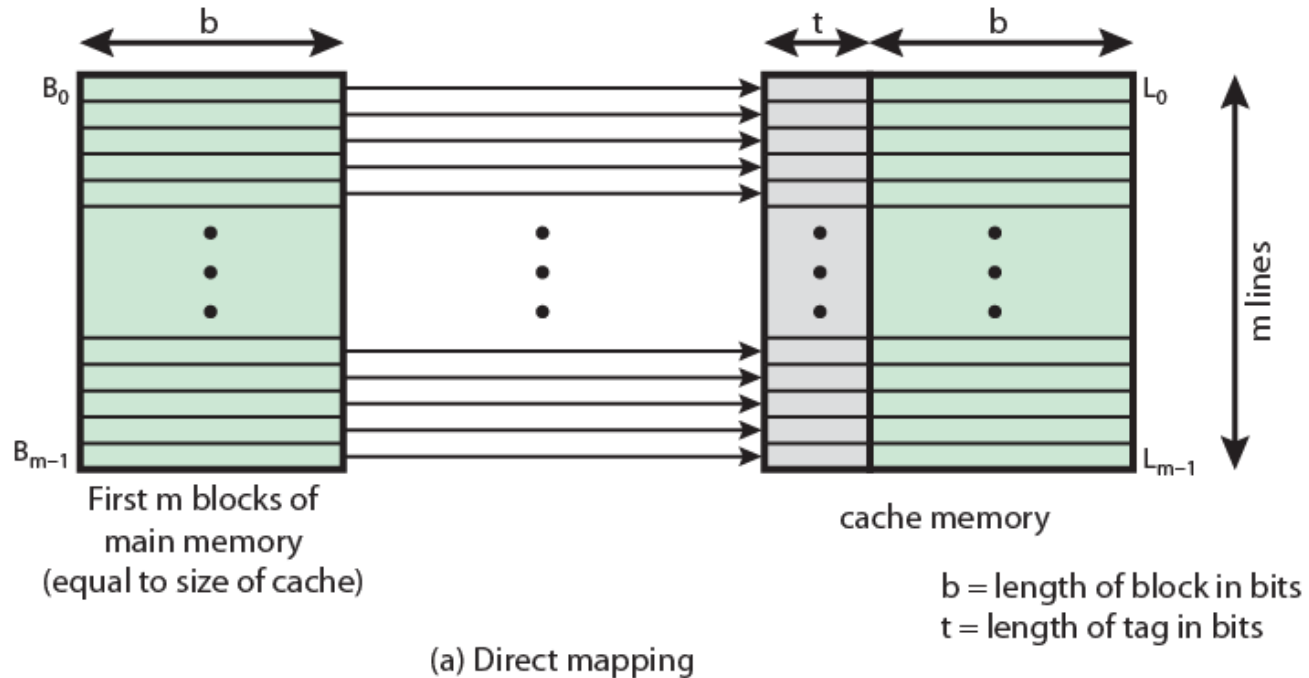
Direct mapping 直接映射

- $i = j \bmod m$

- i : the cache line number **cache行号**
- j : main memory block number **内存块号**
- m : number of lines in the cache **cache行总数**



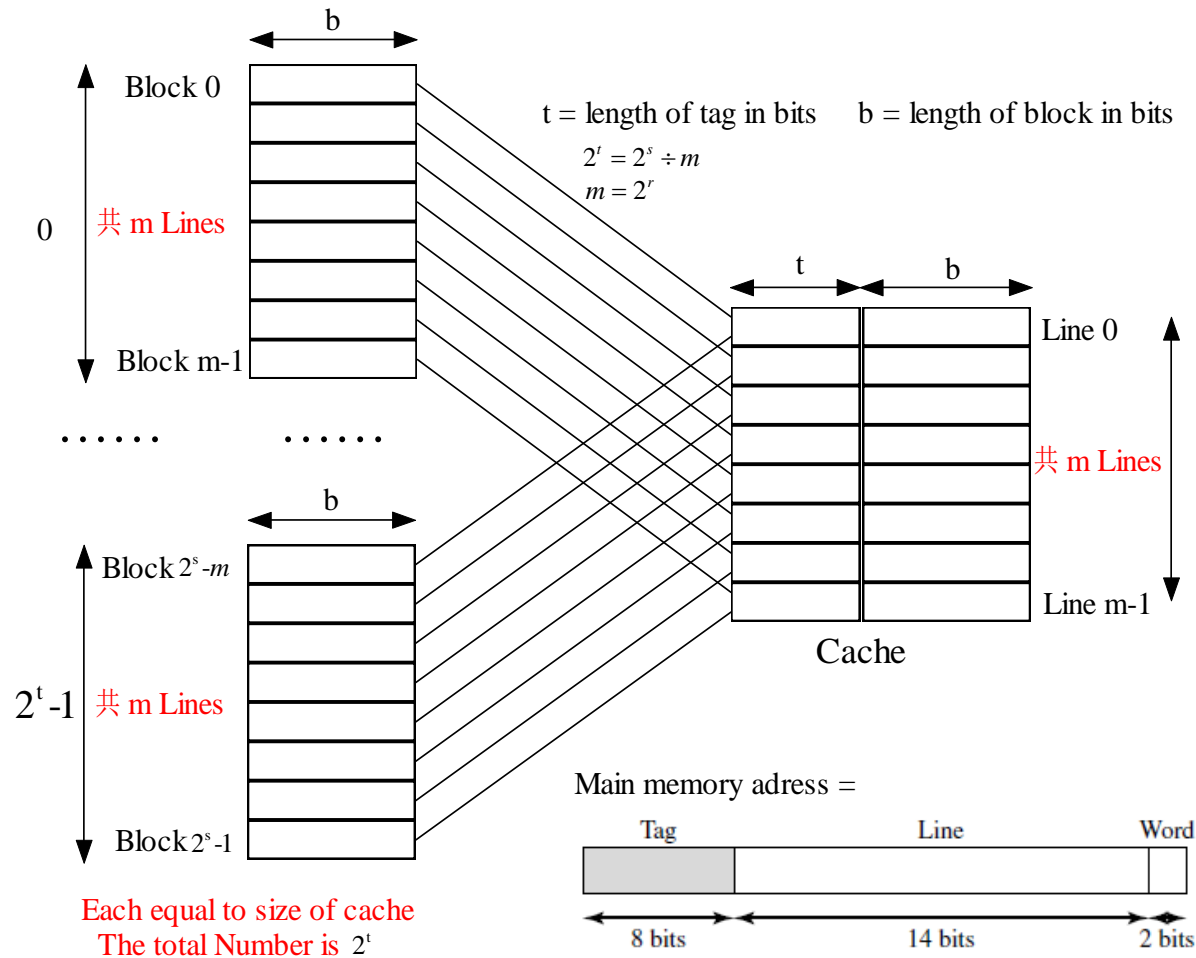
Direct mapping from cache to memory 直接映射图示



- 主存中前 m 个块一对一映射到cache中的 m 行
- Cache中，每一行包括 b 位数据位和 t 为标记位



Direct mapping from cache to memory 直接映射图示



- 内存中的块从头开始，按照cache的行数进行分组，每组都有m块
- 每一组都按照顺序映射到cache的行
- 内存中的一个块只能映射到cache中的唯一一行
- Cache中的一行对应内存中的 2^t 块



Direct Mapping Cache Line Table 直接映射行表

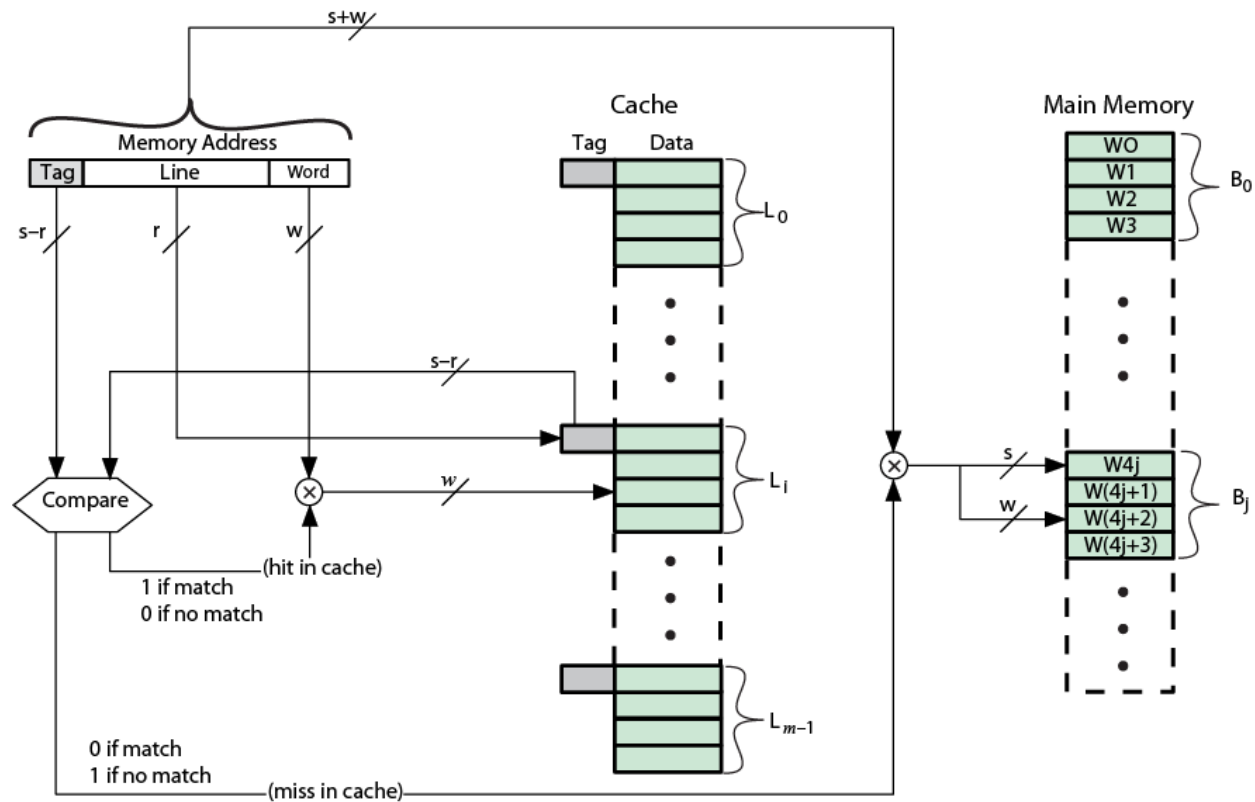
Cache的行

对应的内存块号

Cache line	Main Memory blocks held
0	0, m, 2m, 3m, ...
1	1, m+1, 2m+1, 3m+1, ...
...	
m-1	m-1, 2m-1, 3m-1, 4m-1, ...



Direct mapping cache organization 直接映射cache组织

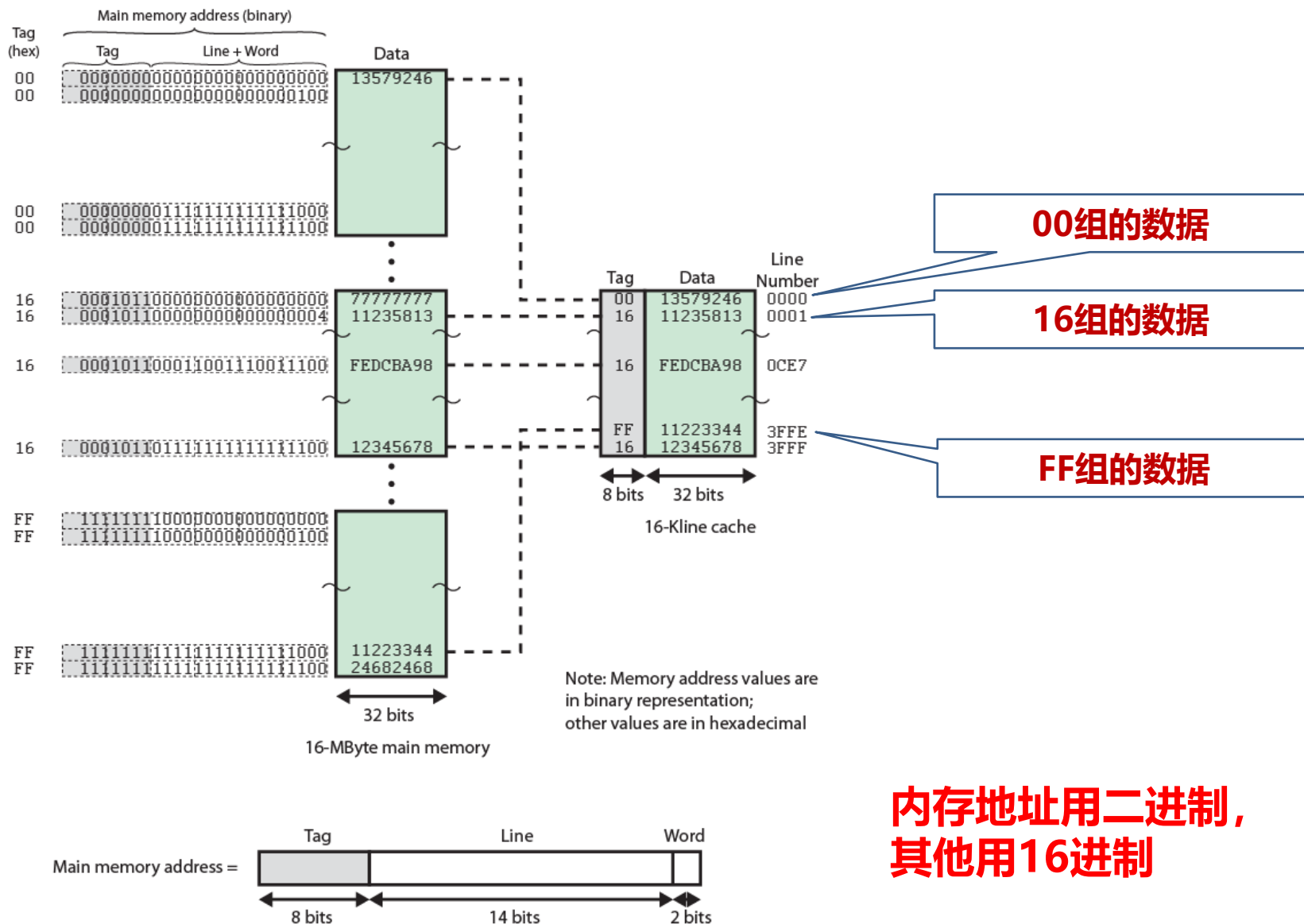


- 根据内存地址中的 cache 行地址去找到 cache 的对应行
- cache 中行的标志位 s-r, 和内存地址中的高位 s-r 进行对比
- 如果一致, 说明命中, 根据地址中的字序号, 获取数据返回给 CPU
- 如果不一致, 说明不在 cache 中。去主存中获取数据并处理



Example of direct mapping

直接映射举例



内存地址用二进制，
其他用16进制



参数分析-内存

内存单元号	内存地址
0	0000 000 0
1	0000 000 1
2	0000 001 0
3	0000 001 1
4	0000 010 0
5	0000 010 1
6	0000 011 0
.....	
253	1111 110 1
254	1111 111 0
255	1111 111 1

块

块号S

块内地址w

- 内存总共有256个存储单元（字）
- 需要8位地址来表示每一个字的位置
- 假定每2个字组成一组（块），总共有128块
- 块号需要7位来表示， $s=7$ ，块内标识每个字，需要1位， $w=1$
- 对每个字来说
 - 每个字所在的块号，正好是地址的前7位
 - 块内的位置，就是地址的最后一位



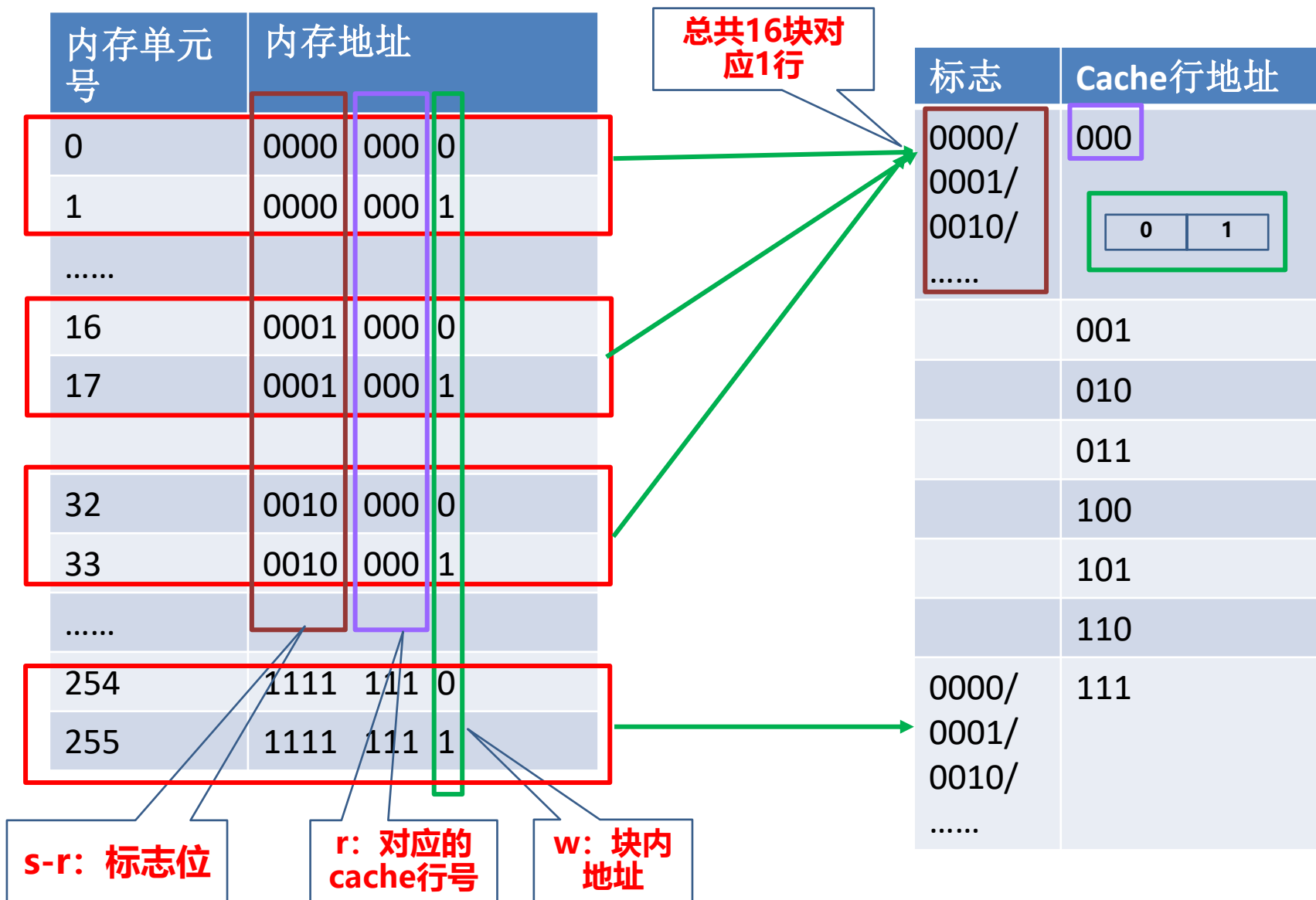
参数分析-cache

Cache行号	Cache行地址
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

- Cache总共有8行，每行2个字（对应内存中每个块2个字）
- 标识一个cache需要3位， $r=3$
- 标识cache行内的每个字，需要1位。跟内存一样， $w=1$
- 内存容量是cache的16倍
- 一个cache行需要对应16个内存块
- Cache行中是不是需要4位来标识存的这16个内存块中的哪一个呢？



参数分析-映射





Direct mapping summary 直接映射小结

- Memory address length = $(s + w)$ bits 内存地址长度= $s+w$ 位
 - Block size = 2^w words or bytes 内存块大小=cache的行大小= 2^w
 - Number of blocks in main memory = $2^{(s+w)}/2^w = 2^s$ 内存块的数量
 $=2^s$
 - Number of addressable units = 2^{s+w} words or bytes 寻址单元数
 $=2^{s+w}$
- Number of lines in cache = $m = 2^r$ cache的行数= 2^r
- Size of tag = $(s - r)$ bits 标记位的长度= $s-r$

Tag $s-r$	Line or Slot r	Word w
8	14	2



Characteristics of direct mapping 直接映射的特点

- Simple implementation logic 实现逻辑简单
- Inexpensive 便宜
- Fixed location for given block 映射方式固定
- Frequent replacement operations, low hit rate 替换操作频繁，命中率比较低
 - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high 如果程序访问的2个块重复地映射到同一个cache行，cache失效率高
 - Called jitter 称为抖动



Victim cache – a solution to direct mapping 一种直接映射的补丁

- A method to lower miss penalty 降低失效惩罚
- Remember and save what was discarded 记住并保存被丢弃的行
 - Already fetched 被访问过的行
 - Use again with little penalty 再次使用会降低惩罚
- Victim cache
 - Fully associative 全关联映射
 - 4 to 16 cache lines 只有4-16个cache行的大小
 - Between direct mapped L1 cache and next memory level 在一级cache和下一级存储层级之间



Associative mapping 全相联映射

- A mapping method proposed to solve the problem of direct mapping 为解决直接映射存在的问题而提出的一种映射方法
- A main memory block can load into any line of cache 主存中的块可以映射到cache中的任意一行
 - Memory address is interpreted as tag and word 内存地址分解为tag和块内的字地址
 - Tag uniquely identifies block of memory tag唯一确定内存中的一行
 - Every line's tag is examined for a match cache行的tag用来检查是否匹配
- Mapping is quite flexible 映射非常灵活
- Cache searching gets complex cache检索复杂



Associative mapping address structure 全相联映射地址

- Address length = $(s + w)$ bits 地址长度= $s+w$ 位
- Number of addressable units = 2^{s+w} words or bytes 寻址空间= 2^{s+w}
- Block size = line size = 2^w words or bytes 块大小 = 2^w
- Number of blocks in main memory = $2^{s+w}/2^w = 2^s$ 块数量 = 2^s
- Number of lines in cache m cache的行数是 m
- Size of tag = s bits cache中行的tag标记位为 s 位



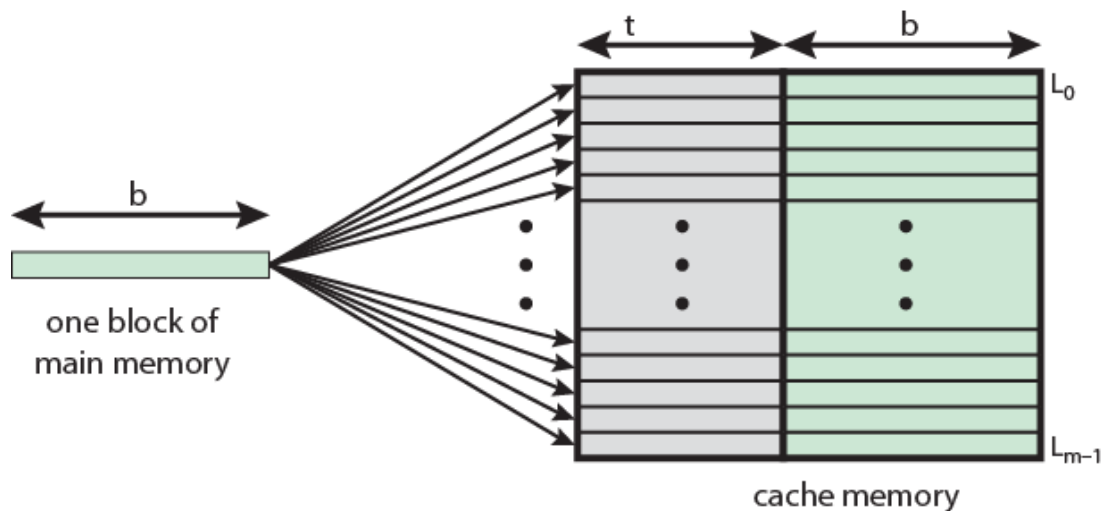
Associative mapping address structure 全相联映射地址

- 22 bit tag stored with each 32 bit block of data 22个标记位，
每个块有4个字节32bit的数据
- When the memory needs data, compare tag in address field
with tag entry in cache to check for hit 需要数据时，检查地址中的
tag位和cache中的tag位是否一致，来判断是否命中
- Least significant 2 bits of address identify which word is
required from data block 最低的2位确定cache中的块数据中，哪
个字是需要的数据。

Tag 22 bit	Word 2 bit
------------	---------------



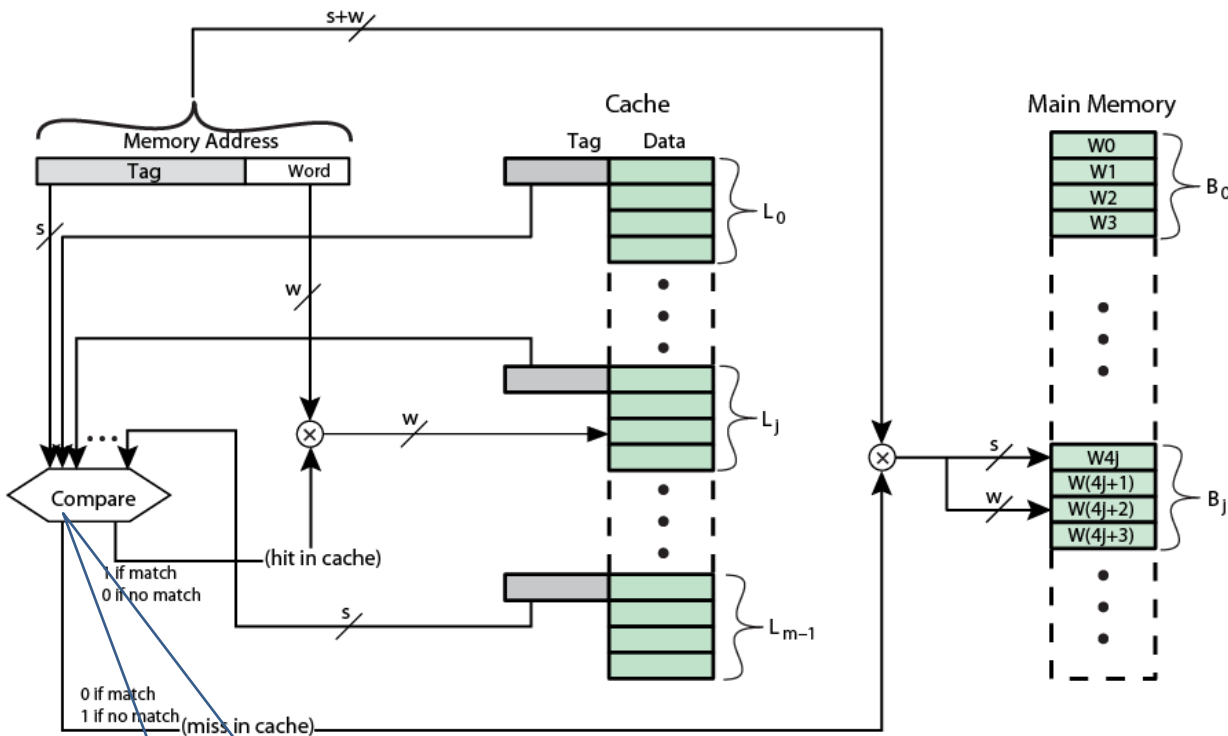
Associative mapping 全相联映射



- 内存中一个块可以映射到cache中的任意一行，没有任何限制
- 全相联映射中，cache中每行的标志位比直接映射要长很多
- 灵活性最高，复杂度也最高



Fully associative cache organization 全相联映射cache组织



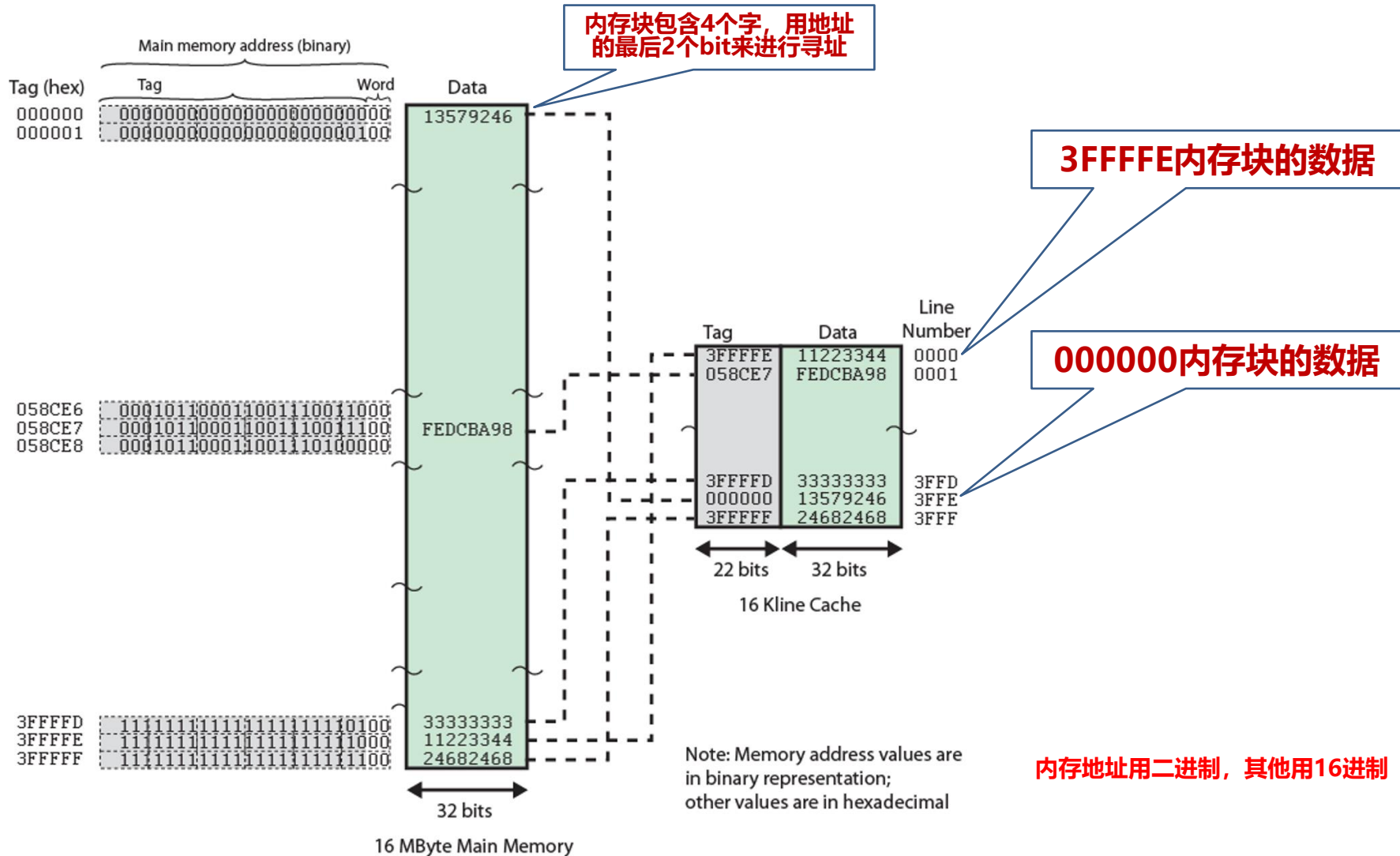
需要比较内存的tag和
所有cache行的tag

- 内存地址包括Tag部分和word部分
- Cache每行包括Tag部分和数据
- CPU访问数据时，将内存中的tag和全部cache中的tag进行比较
- 如果有一个匹配，表示命中，直接访问数据
- 如果全部不匹配，表示需要的数据不在cache中，需要到内存中去取数据



Example of associative mapping

全相联映射举例





Question

- **全相联映射的最大问题是什么？**



- Advantage
 - Flexibility 灵活
- Disadvantage 缺点
 - Large tag memory tag很长
 - The complex circuitry required to examine the tags of all caches lines in parallel 需要设计复杂的电路来并行比较tag
- A mechanism is needed to improve the hit rate 需要有一种机制来提高命中率
- Replacement algorithms are important to maximize the hit ratio 替换算法很重要



Associative Mapping Summary 全相联映射小结

- Address length = $(s + w)$ bits 地址长度= $s+w$ 位
 - Number of addressable units = 2^{s+w} words or bytes 寻址空间= 2^{s+w}
 - Block size = line size = 2^w words or bytes 块大小 = 2^w
 - Number of blocks in main memory = $2^{s+w}/2^w = 2^s$ 块数量 = 2^s
- Number of lines in cache m cache的行数是 m
- Size of tag = s bits cache中行的tag标记位为 s 位



Question

- **直接映射和全相联映射都有问题，如何解决？**



Set Associative Mapping 组相联映射

- Combination of Direct Mapping and Fully Associated Mapping
直接映射和全相联映射的结合
- Cache is divided into a number of sets **cache**分为若干个组
 - Each set contains a number of lines 每个组包含一些行
 - A given block maps to any line in a given set 内存块映射到某个组中的任意一行
- One memory block can only be mapped to a fixed cache set, which greatly reduces the difficulty of examine **内存块只能映射到cache组中的固定组，比较难度大大降低了**
- Since memory blocks can be mapped to any row in the cache group, it has certain flexibility 由于内存块可以映射到**cache组中的任意一行，具有一定的灵活性**



Example of set associative mapping 组相联映射举例

- e.g. 2 lines per set 假如每组cache有2行
 - 2 way associative mapping 2路组相联映射
 - A given block can be in one of 2 lines in only one set 任意一个给定的块可以映射到确定的唯一一组的两行中的任意一行
- 13 bit set number 13位组号
- Block number in main memory is modulo 2^{13} 内存中的块号与映射的cache组的关系是mod 2^{13}
- 000000, 008000, 010000, ... FF8000 map to same set 例如, 这 2^9 个内存块映射到同一个cache组



Set associative mapping address structure 组相联映射地址结构

- Use set field to determine cache set to look in 使用组号来决定是cache的哪个组
- Compare tag field to see if we have a hit 比较tag标记来确定是否命中

Tag 9 bit	Set 13 bit	Word 2 bit
-----------	------------	------------

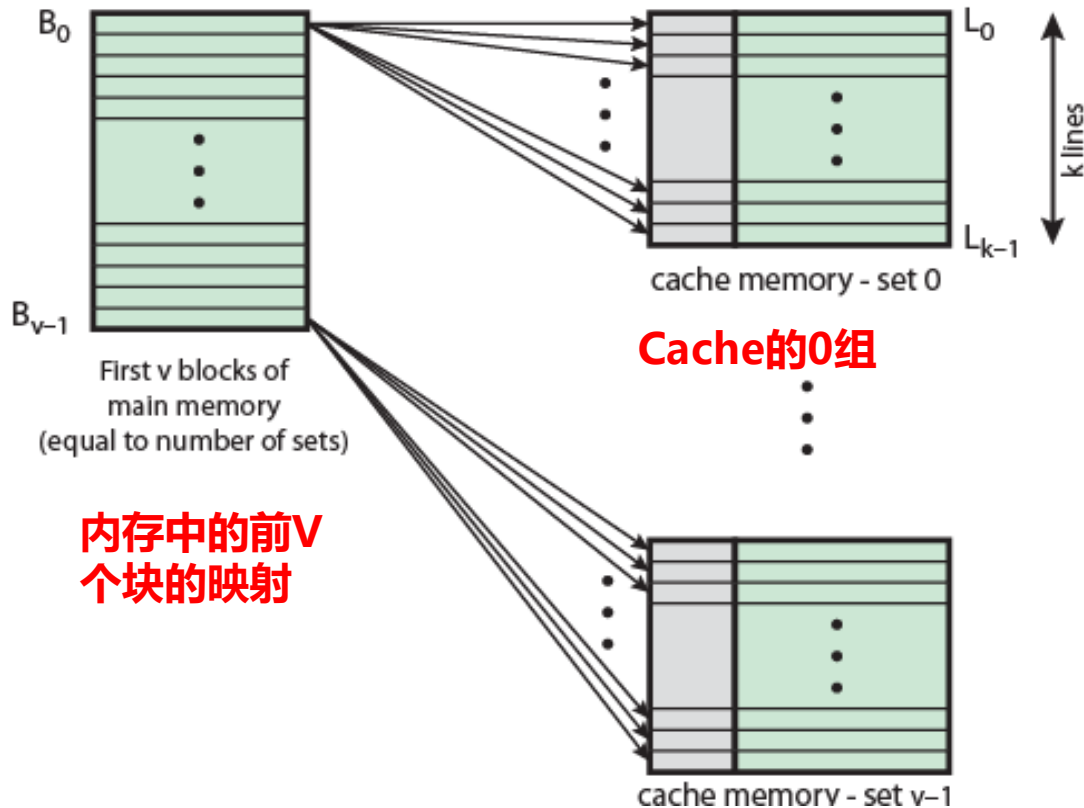
9位标记

13位组号



Mapping from memory to cache: v associative **V组相联映射**

Cache分为V组，每组K行



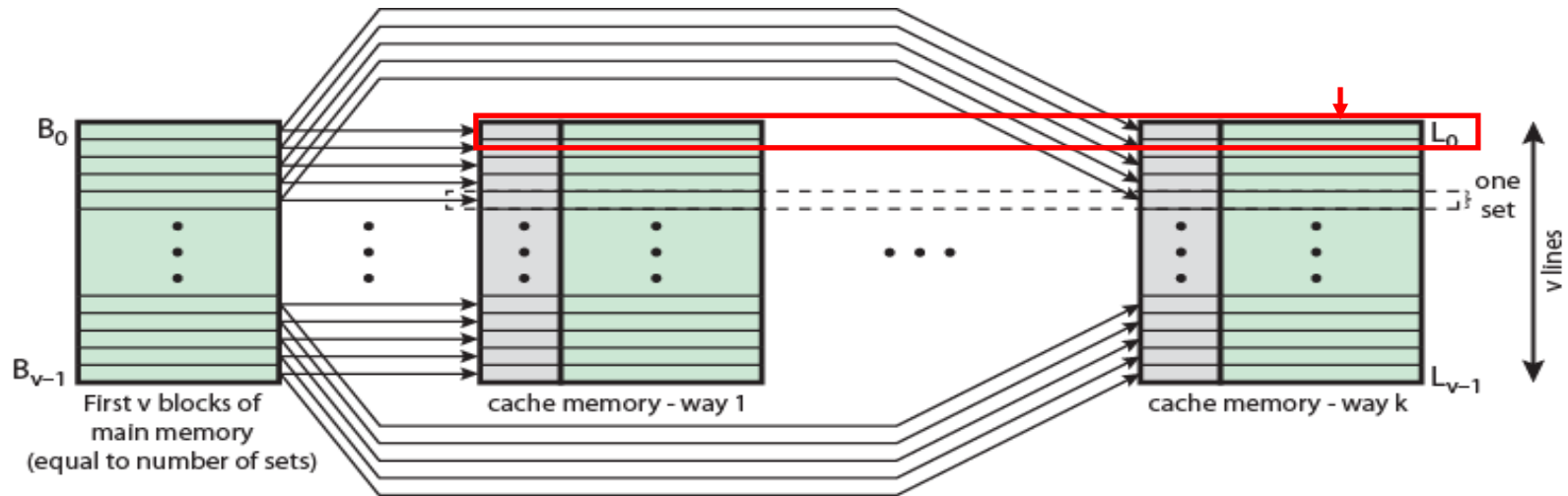
内存中的前V个块的映射

Cache的0组

Cache的v-1组

- cache分为v个组，每个组包含k行
- 内存的前V块，每块按顺序映射到右边的V个组的一个组
- 内存中的每一块可能会在组中的任一行
- 内存中后面的块按照同样的方法，映射到cache中某一组

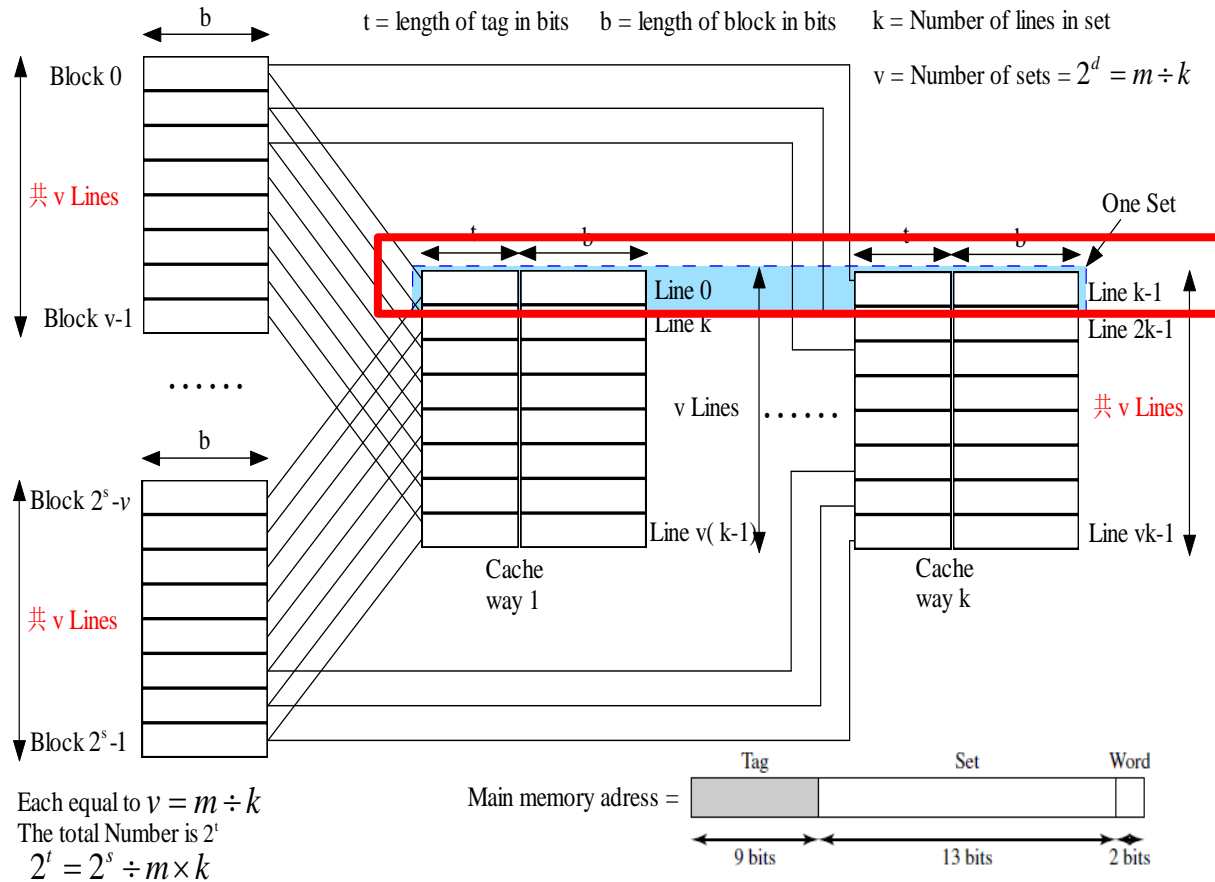
Mapping from memory to cache: k associative k路相联映射



- V 组相联映射同时也可以看做是 k 个直接映射的同时使用
- cache被分成大小相等的组，每组有 k 行，总共有 V 个这样的组。称为 K 路相联映射
- 内存中的 $0 \sim v-1$ 行，通过直接映射方式，映射到cache中每一路的 v 行中。然后，内存中的 $v \sim 2v-1$ 也同样映射到每一路中的 v 行
- 内存中的每个块，可以映射到内存中的 k 路，也就是 k 行



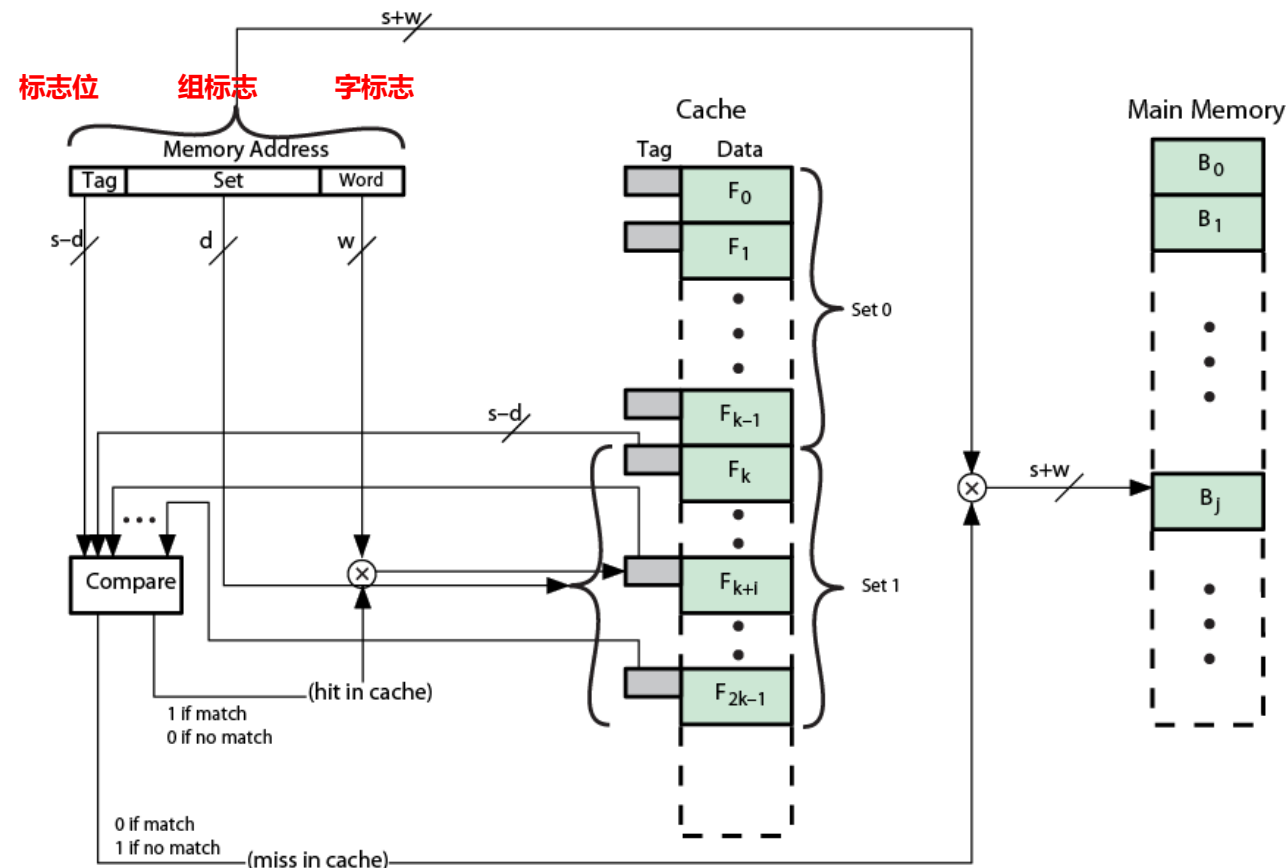
Mapping from memory to cache: k associative k路相联映射



- cache采用v组k路组关联映射方法。每一路有v行
- 主存从0~v-1块映射到cache的所有k路的v行
- v~2v-1块同样映射到cache的所有k路的v行，如此类推
- 内存中的每一个块都对应cache中的组中的k行



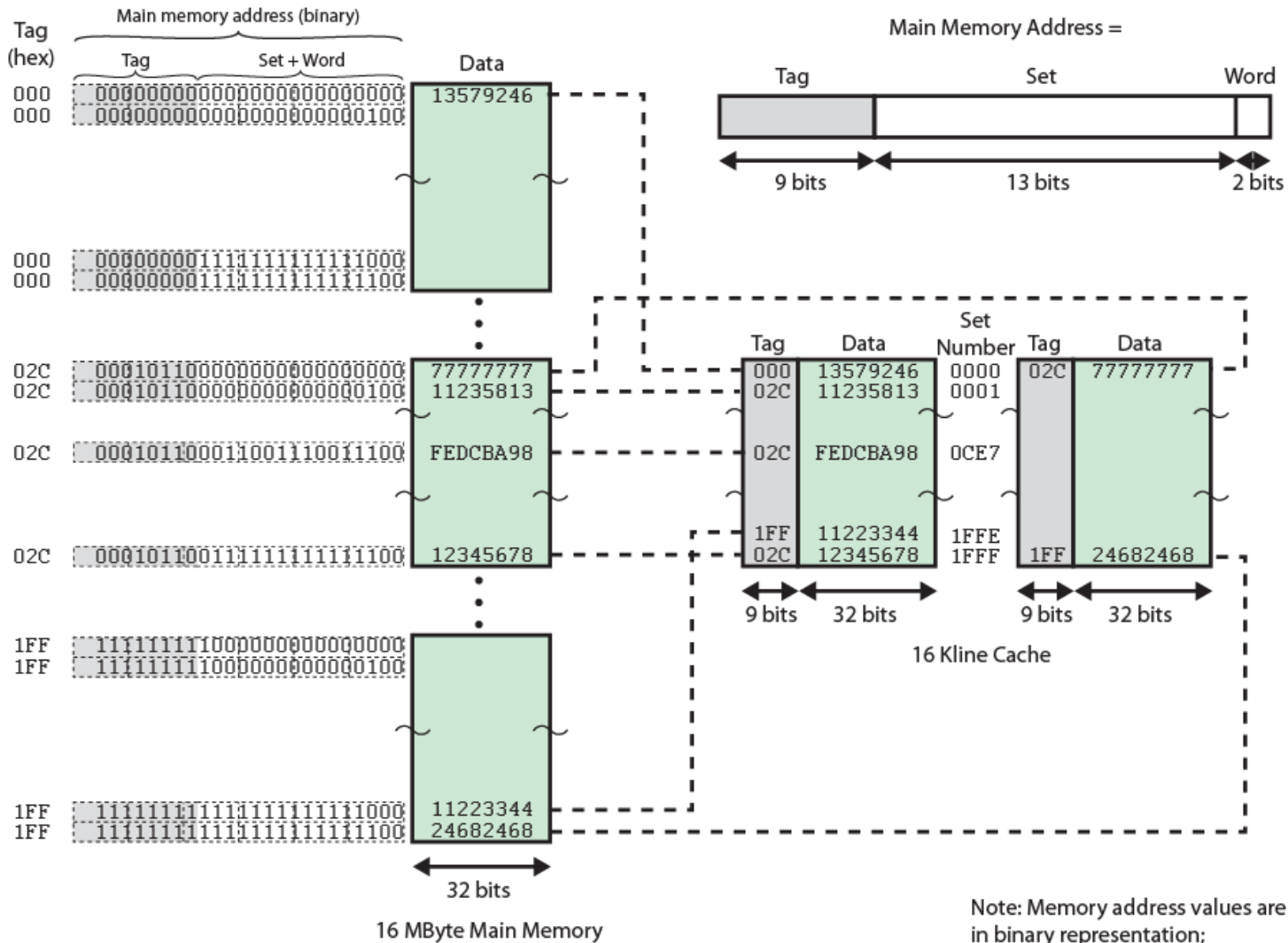
K-way set associative cache organization **k路组相联cache组织**



- 内存地址分解成标志位、组标识和字标识。
- 根据组标识，找到cache中对应的组，然后将cache中该组中所有行的tag和主存地址的tag行进行对比
- 如果能对比成功，表示命中，再根据字标识，取到处理器需要的数据，并返回给处理器
- 如果没有命中，到主存中获得存储器地址所在的块，并根据算法替换到cache中，然后再返回给处理器



2-way set associative mapping example 2路组相联举例





Set associative mapping summary 组相联映射总结

- Address length = $(s + w)$ bits 内存地址长度为 $s+w$
 - Number of addressable units = 2^{s+w} words or bytes 可寻址内存空间为: 2^{s+w} 个字或字节
 - Block size = line size = 2^w words or bytes 内存块或cache行大小为 2^w 个字或字节
 - Number of blocks in main memory = 2^s 内存中的块数为 2^s
- Number of lines in set = k cache中每组有 k 行
 - Number of sets = $v = 2^d$ cache 总共分为 v 组, 需要用 d 位来标识
 - Number of lines in cache = $kv = k * 2^d$ cache总行数为 kv
- Size of tag = $(s - d)$ bits 标志位等于 $s-d$



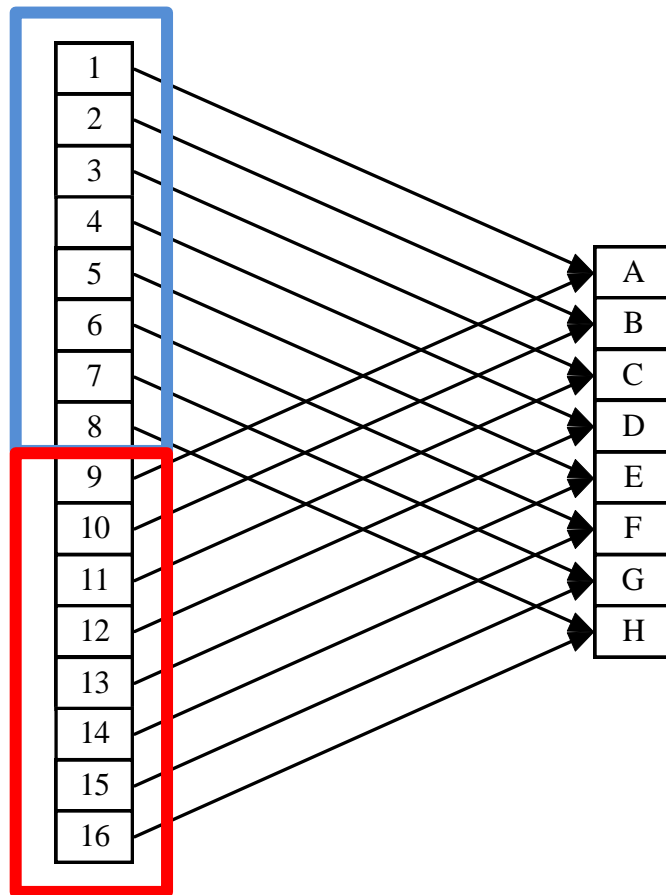
Example 1

- Students: 1~16 as memory blocks 学生1~16是内存块
- Computers: A,B,C,D,E,F,G,H as Cache lines 计算机A~H是cache行
- **Question:** how to assign computers to students? 如何给学生分配计算机
 - Direct mapping
 - Associative mapping
 - Set associative mapping



Direct mapping 直接映射

- 1,9 -->A
- 2,10-->B
- 3,11-->C
- 4,12-->D
- 5,13-->E
- 6,14-->F
- 7,15-->G
- 8,16-->H



- 将学生分为2组，每组8个学生。1~8号为第一组，9~16号为第二组
- 第一组的8个学生，一一映射到A~H这8台机器。第二组的8个学生，也一一映射到A~H这8台机器
- 如果1号学生用了A，那么9号学生就不能用了。9号要用的话，就要把1号给替换出去
- 如果要找学生1，只需要到A机器那里看他在不在



Associative mapping 全相联映射

1, 2, 3,
4, 5, 6,
7, 8,
9, 10,
11, 12,
13, 14,
15, 16



A B C D
E F G H

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16



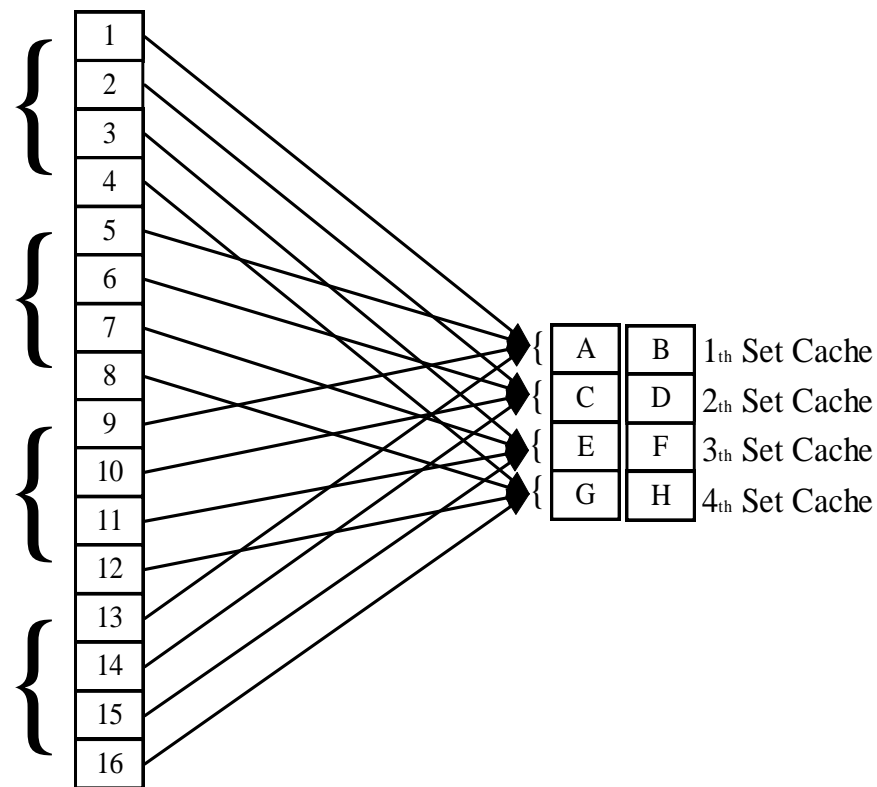
A
B
C
D
E
F
G
H

- 所有的学生都可以使用任意一台计算机
- 如果所有的计算机都占满了，新来的学生要用计算机，只能把现在用计算机的学生调出去一个
- 如果要看某个学生是不是在使用计算机，需要从A开始挨个检查，看该学生是不是在使用这个计算机



Two way set associative mapping 2路组关联映射

- 2路，cache的每组有2行。总共分为4组
- 第一组包含AB两台机器，第二组包含CD，第三组包含EF，第四组包含GH。
- 学生按4个一组，分为4组。每一组的4个学生按顺序映射到4组计算机
- 1、5、9、13，这4个学生共用AB这一组计算机，其他类似
- 如果要看1号学生是不是在用计算机，只需要看第一组计算机中是否有1号学生就可以
- 组相联映射既有一定的灵活性，同时也减少了对比的复杂度



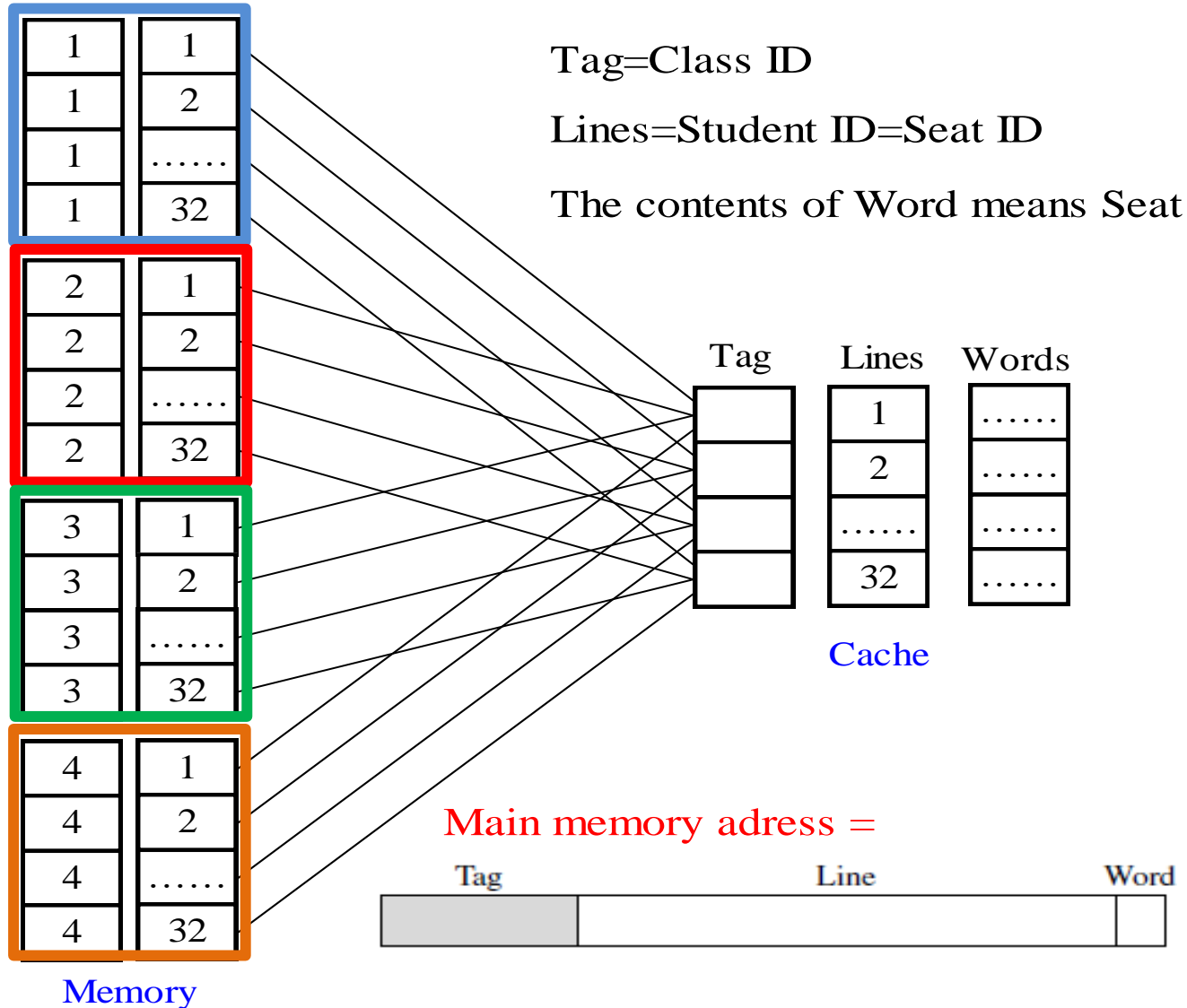


Example 2

- Students as main memory: 学生是内存
 - There are four classes 有4个班
 - Class ID:1-4 班号从1~4
 - There are 32 students in each class, 每个班有32个学生
 - Student ID: 1-32 学生ID从1~32
- Seat as Cache : 教室的座位是cache
 - There are only 32 seats for all students, 总共只有32个座位
 - Seat ID: 1-32 座位号为1~32
- Question: how to assign seats to students? 如何给学生分配座位?

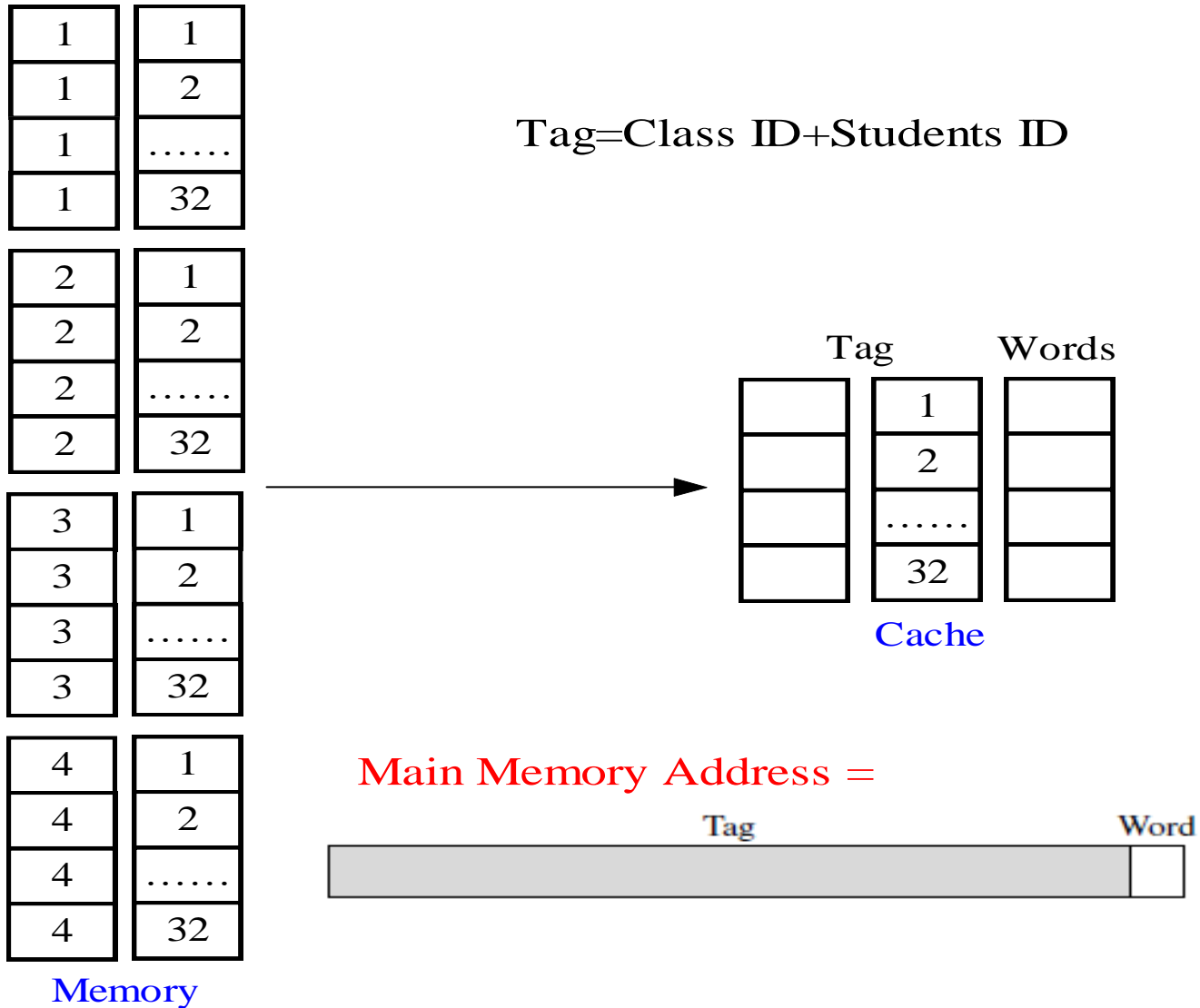


Direct mapping 直接映射





Associative mapping 全相联映射



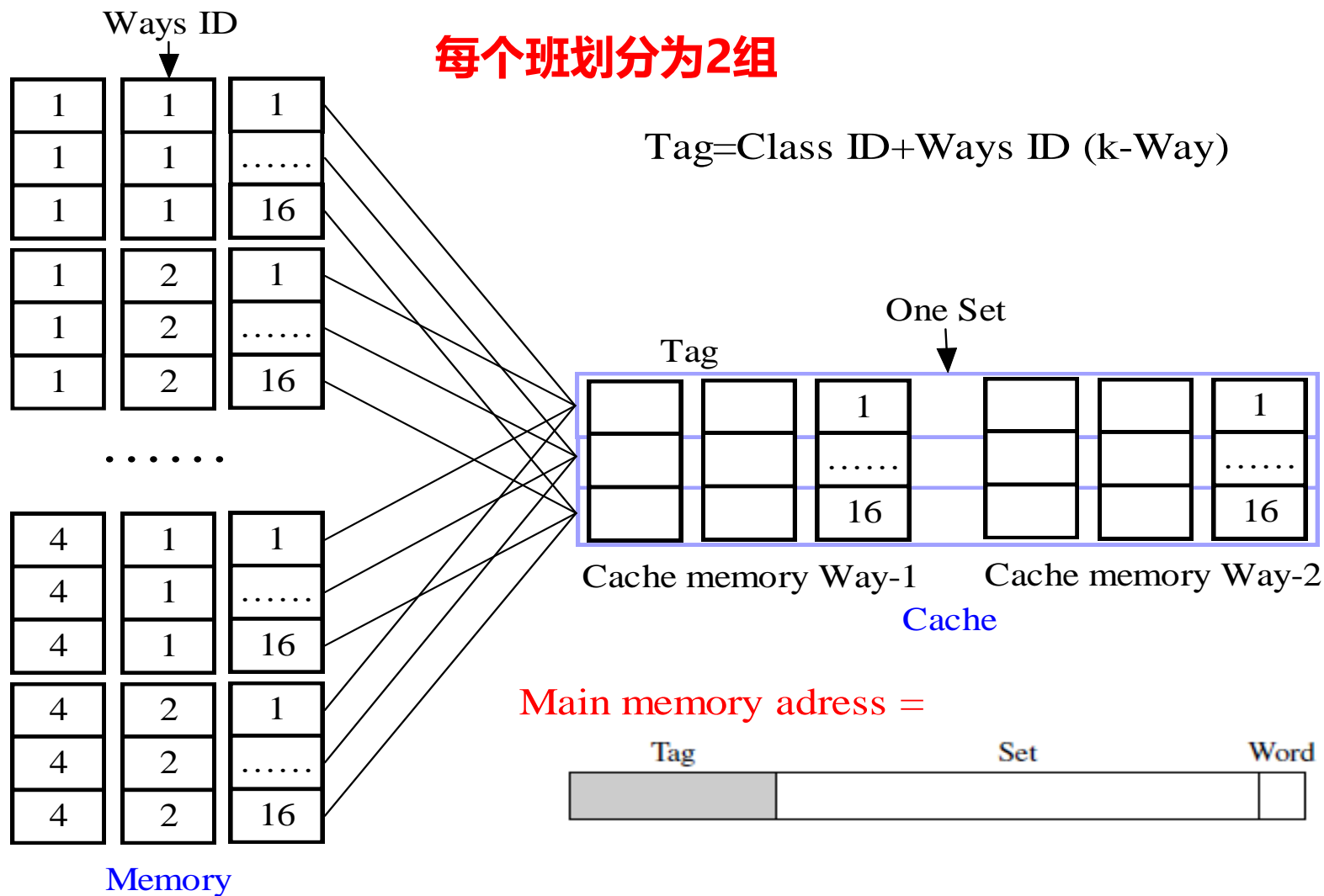


2-way associative mapping 2路组关联映射

Each class is divided to 2 group

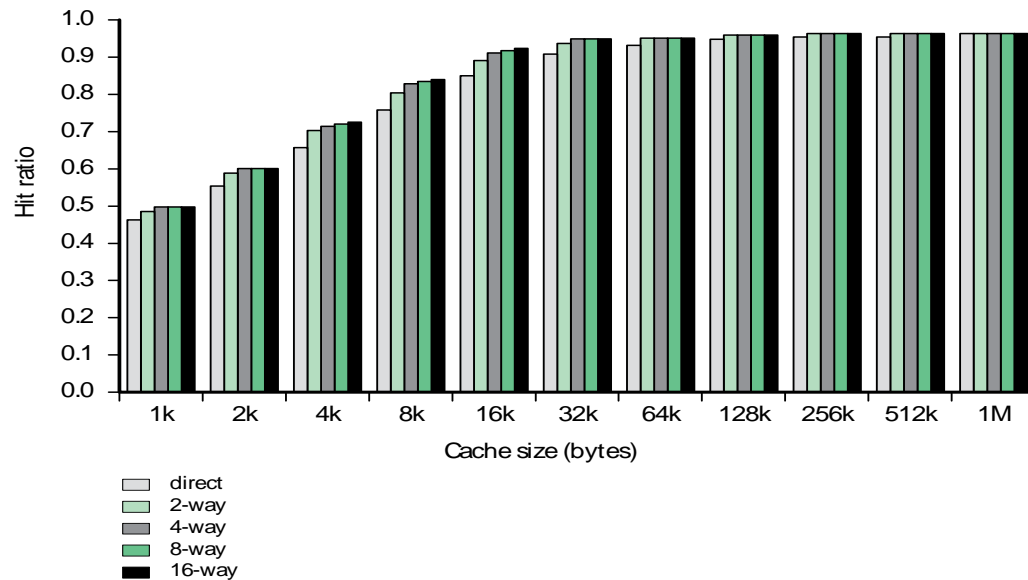
每个班划分为2组

Tag=Class ID+Ways ID (k-Way)





Direct and set associative 直接映射和组关联映射



- cache容量比较小的时候，增加cache的大小，命中率提高很快
- cache达到32k的时候，命中率提高不明显，cache不是越大越好。
- cache的容量达到32k之后，提高路数基本没有效果了，但是电路复杂度提高了，提高路数意义不大
- 当cache达到1M之后，组相联映射和直接映射的性能基本上差不多



Replacement algorithms 替换算法

- Only a small part of main memory data can be saved in cache
cache中只能保存一小部分主存数据
- When required data is not in cache, it is necessary to fetch the block from memory 当需要的数据不在**cache**的时候，需要从主存取过来
- May need to move a row from the cache 可能需要将**cache**中的某一行移走
- Which line will be replaced? 替换掉哪一行呢？
- Different memory mapping methods adopt different replacement strategies 不同的内存映射方法采用不同的替换策略



For direct mapping 对于直接映射

- Each block in main memory corresponds to a fixed row in cache 主存中的每个块对应的是cache中的固定的一行
- no choice 没有选择
- When a new block in the memory needs to be written to the cache, you can only replace the data in the previous cache 内存中新的一块需要写到cache时，只能将之前cache中的数据替换掉
- write the new block data to the cache row 写入新的块数据到cache行



For associative mapping 对于相联映射

- Block in memory can correspond to multiple rows in the cache 内存中的一行对应cache中的多行
- If there are blank lines in the corresponding multiple lines, write them directly 如果对应的多行中有空行，直接写入
- If there is no blank row in the corresponding multiple rows, you need to determine which row to overwrite 如果对应的多行中没有空行，需要确定将哪一行的数据覆盖
- Hardware implemented algorithm for speed 为了速度，用硬件实现替换算法



Replacement strategies 替换策略

- Least Recently used (LRU) 最近使用原则
 - usage time of the cache line needs to be recorded 需要记录
cache行的使用时间
- First in first out (FIFO) 先进先出原则
 - replace block that has been in cache longest 在cache中时间最
长的替换出去
- Least frequently used 最少访问频率原则
 - replace block which has had fewest hits 替换掉最少使用的块
- Random 随机原则



Question

思考：

- **我们已经了解了内存和cache的映射规则，还有替换算法。**
- **这样是不是够了？**



Why need write policy? 为什么需要写策略

- Data in cache is only a copy of memory **cache**中的数据是存储器数据的备份
- If data in the cache is updated, it is inconsistent with the data in memory 如果**cache**的数据发生了更新，就和存储器中的不一致了
- How and when to maintain data consistency is a problem 如何，以及什么时候去维护数据的一致性，是一个问题，也是写策略需要解决的问题
- 15% of memory references are writes 写操作占了**15%**
- Need to design a write policy 需要设计一个写策略



Write policy 写策略

- If the contents in cache are updated 如果cache中更新了内容
 - Cache cannot be overwritten before the memory data corresponding to the cache is updated 在cache对应的内存数据更新之前，cache不能被覆盖
- If there are multiple CPUs in the system 如果系统中有多cpu
 - Each may has an independent cache 每个CPU可能有独立的cache
 - Cache consistency needs to be maintained. 需要维护cache的一致性
 - This will make the problem more 这样问题会更复杂
- I/O may address main memory directly. Bringing data consistency problems I/O可能直接访问内存，带来数据一致性问题



Write through 写直达

- All writes go to main memory as well as cache 所有写cache的操作同时会写存储器
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date 多个CPU时，能够监视存储器，以保证本地cache的数据及时更新
- Lots of traffic 会带来大量的流量
- Slows down writes 减慢写的速度



Write back 写回法

- Updates initially made in cache only 更新操作最开始只在cache中进行
- Update bit for cache slot is set when update occurs 当cache更新的时候，设置“更新位”
- If block is to be replaced, write to main memory only if update bit is set 当cache块要替换的时候，如果更新位设置，就更新存储器的数据
- Part of data in memory is invalid 部分存储器中的数据是无效的
- I/O must access main memory through cache I/O需要通过cache来访问存储器



Example

- Consider a cache with a line size of 32 bytes and a main memory that requires 30 ns to transfer a 4-byte word. 考虑具有32字节的行大小的缓存，需要30 ns来传送4字节字的主存储器
- For any line that is written at least once before being swapped out of the cache, if we want a write-back cache to be more efficient than a writethrough cache. 对于在交换出缓存之前至少写入一次的任何行，并且假定写回法比写直达更有效
- Then what is the average number of times that the line must be written before being swapped out for? 在交换出缓存之前必须写入该行的平均次数是多少？



Example

- **For the write-back case**, each dirty line is written back once, at swap-out time, taking $8 \times 30 = 240$ ns. 对于写回法，每个脏行在交换时回写一次，时间为 $8 \times 30 = 240$ ns。
- **For the write-through case**, each update of the line requires that one word be written out to main memory, taking 30 ns. 对于写直达，每次更新都需要将一个字写入主内存，耗时30 ns
- Therefore, if the average line that gets written at least once gets written more than 8 times before swap out, then write back is more efficient.
因此，如果至少写入一次的平均行在调出之前被写入8次以上，那么回写效率更高。



Why need Cache Coherency?

- In multi CPU architecture, there is a cache consistency problem when sharing memory 在多CPU架构中，共享内存时，存在cache一致性问题
 - Each CPU has its own cache 每个CPU都有自己的cache
 - If the cache on a processor is modified and not written back to the memory, the data in the memory is invalid, as is the data in other caches 某个处理器上的cache做了修改并且没有回写到存储器，那内存中的数据无效，其他cache上该数据也无效
- DMA may also have consistency problems DMA也可能会存在一致性问题
 - The data read through DMA is not the latest data 通过DMA读取的数据不是最新的数据



Cache Coherency **cache一致性1**

- Bus watching with write through **写直达的总线检测**
 - Each cache controller monitors the address lines **每个cache控制器监视地址线**
 - If another master writes to a memory location, then the corresponding cache entry is invalid **如果其他的处理器写存储器，设置相应的cache行为无效**
- Hardware transparency **硬件透明**
 - The hardware method is used to ensure that all changes to main memory through cache will be reflected in all caches at the same time. **用硬件的方法来保证所有通过cache对主存的修改，同时都会反映到所有的cache中**
 - If a processor modifies a word in the cache, word in main memory and other caches will change through hardware at the same time **如果一个处理器修改了cache上的一个字，那么通过硬件的方式，主存和其他cache上的的字也会同时修改**



Cache Coherency **cache一致性2**

- Noncacheable memory **非cache存储器**
 - Part of the main memory is shared by multiple processors **一部分主存内容为多个处理器共享**
 - All accesses to shared main memory will result in cache invalidation **对共享主存的访问都会导致cache失效**
 - The shared data in the memory will not be copied to the cache **存储器中共享部分的数据不会复制到cache中**



Line size – 1 行大小1

- Retrieve not only desired word but a number of adjacent words as well 检索数据块到cache时，不仅需要的数据，邻近的数据也会进cache
- Increased block size will increase hit ratio at first 增加块大小开始会提高命中率
 - the principle of locality 局部性原理
- Hit ratio will decrease as block becomes even bigger 当块持续变大时，命中率会下降



Line size – 2 行大小2

- Larger blocks 大cache块的问题
 - Reduce number of blocks that fit in cache 减小了cache中的行数
 - Data overwritten shortly after being fetched 很快就替换出去
 - Each additional word is less local so less likely to be needed 附加的字不够局部性，不容易被需要
- No definitive optimum value has been found 没有绝对最优的方法
- 8 to 64 bytes seems reasonable 8~64个字节有道理一些
- For HPC systems, 64- and 128-byte most common 对于HPC系统，64和128字节比较常用



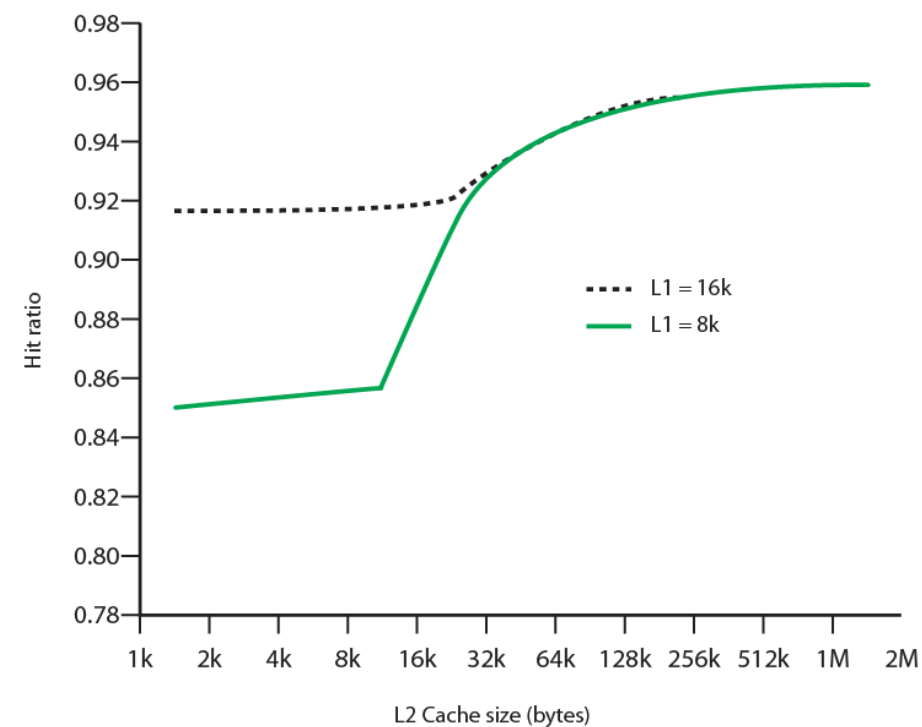
Multilevel Caches 多级cache

- High logic density enables caches on chip 集成度的提高使得cache可以放在CPU芯片上
 - Faster than bus access 比总线访问更快
 - Frees bus for other transfers 减少了总线的访问
- Common to use both on and off chip cache 常用的是既使用片上cache，也使用片外cache
 - L1 on chip, L2 off chip in static RAM L1在片上，L2在片外，用SRAM
 - L2 access much faster than DRAM or ROM L2访问速度比DRAM快
 - L2 often uses separate data path L2使用独立的数据通道
 - L2 may now be on chip 现在L2都有可能在片上
 - Resulting in L3 cache 这样会导致L3 cache的出现



Hit Ratio (L1 & L2) For 8 k bytes and 16 k byte L1

两级cache时的命中率



- L1为8k
 - L2比L1小，L2对命中率的影响不大
 - L2从8k上升到16k的时候，命中率会迅速上升
 - L2到达256k之后，再增加L2的大小，命中率也不会有大的改进
- L1为16k
 - L2小于L1，对命中率没有影响
 - L2超过L1之后，命中率迅速上升
 - L2到达256k之后，再增加意义也不大



Unified v Split Caches 统一与分立cache

- Initially, instructions and data are not distinguished, and they are cached in the same cache 最初并不区分指令和数据，指令和数据都缓存到同一个cache
- Later, there was an architecture using two L1 caches, one for storing data and the other for storing instructions 后来出现了采用两个L1 cache的架构，一个cache用于存放数据，另一个用于存放指令
 - When instructions are needed, access the instruction cache 需要指令的时候，访问指令cache
 - Access the data cache when data is needed 需要数据的时候，访问数据cache



Unified v Split Caches 统一与分立cache

- Advantages of unified cache 统一cache的好处
 - Easy design & implement 设计和实现容易
 - Balances load of instruction and data fetch 自动平衡数据和指令的获取
 - Higher hit rate 高命中率
- Advantages of split cache 分立cache的好处
 - Eliminates cache contention between instruction fetch/decode unit and execution unit 减少了取指、解码、执行中的cache竞争
 - Important in pipelining 在流水线中很重要



Summary 小结

- Cache Addresses Cache地址
- Cache Size Cache 容量
- Mapping Function 映射功能
- Replacement Algorithm 替换算法
- Write Policy 写策略
- Line Size 行大小
- Number of Caches Cache 数目



Outline

- Key Terms
- Computer Memory System Overview 存储系统综述
- Cache Memory Principles cache原理
- Elements of Cache Design cache设计要素
- Pentium 4 Cache Organization Pentium4的cache组织

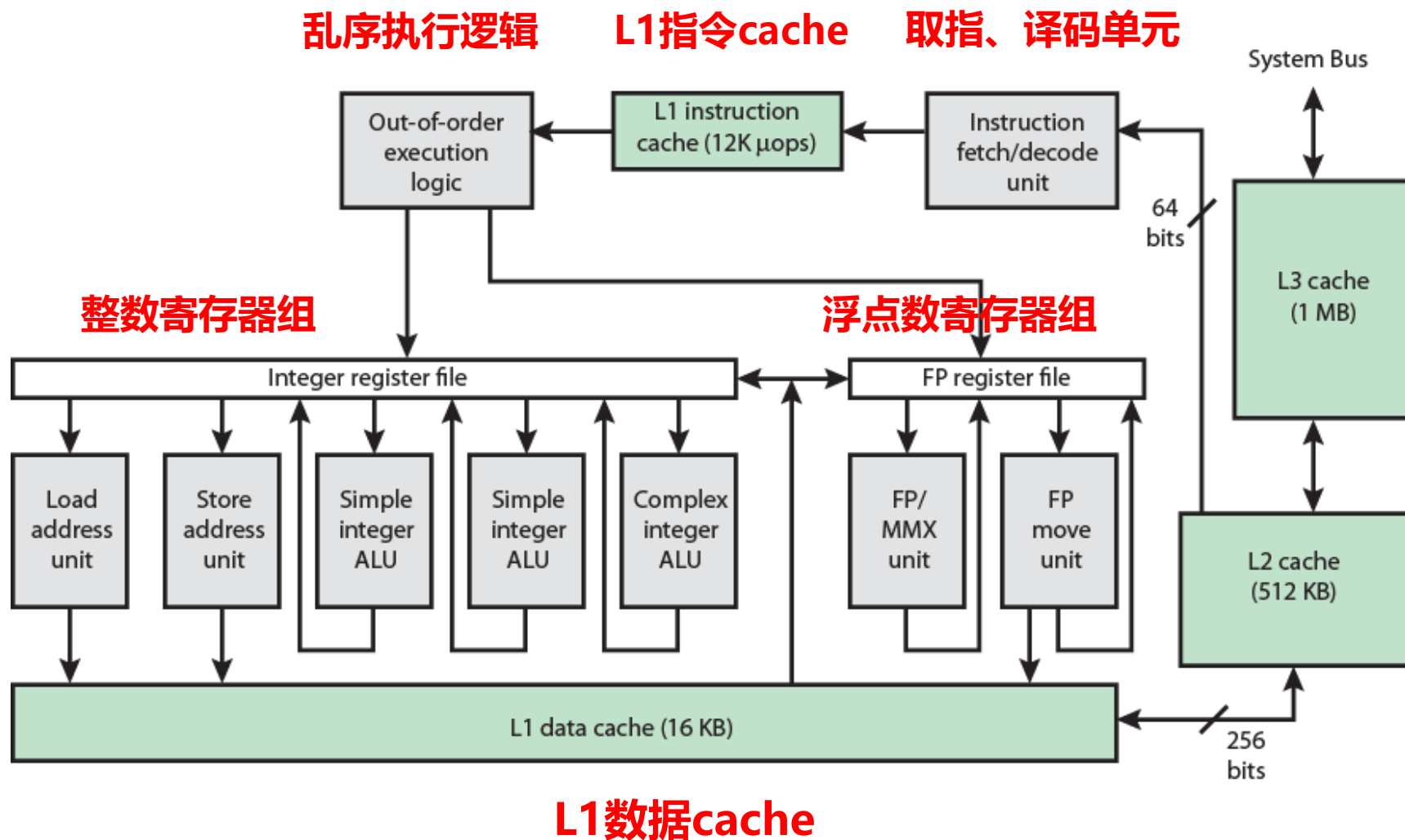


Intel cache evolution Intel cache的演进

- 80386 – no on chip cache 80386 没有片上cache
- 80486 – 8k using 16 byte lines and four way set associative organization 80486采用8kcache，每行16字节，4路组相联
- Pentium (all versions) – two on chip L1 caches 奔腾两个L1 cache
 - Data & instructions 数据和指令cache分开
- Pentium III – L3 cache added off chip PIII, 增加了片外L3 cache
- Pentium 4 P4处理器
 - L1 caches: 8k bytes, 64 byte lines, four way set associative
 - L2 cache: Feeding both L1 caches, 256k, 128 byte lines, 8 way set associative
L2cache 为2个L1 cache提供数据
 - L3 cache : on chip, 1M L3 cache也放在片上了，容量1M



Pentium 4 block diagram P4处理器框架图





Summary and Question

- 小结
 - 对计算机存储系统的结构进行了分析
 - 详细讨论了cache存储器的工作原理、设计要素等内容
 - 对Pentium和ARM的cache组织方式进行了了解
- 问题
 - 问题1：数据存取的四种方式，以及典型的存储设备
 - 问题2：cache和主存的映射方法有哪几种？简单描述一下映射方法。
 - 问题3：为什么会存在cache的写策略问题？有哪几种写策略？



Assignments

- Review questions
 - 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8
- Problems
 - 4.1, 4.2, 4.3, 4.7, 4.8, 4.10, 4.11, 4.12



谢谢大家!

