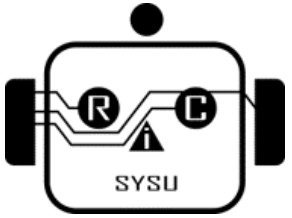


计算机组成原理

第五章：存储器层次结构

中山大学计算机学院

陈刚

2022年秋季

上讲回顾

回顾内容

- 5.3 高速缓冲存储器
  - 什么是程序访问的局部性
  - 具有Cache机制的CPU的基本访存过程
  - Cache和主存之间的映射方式

回顾——5.3.1 程序访问局部性

什么是程序访问局部性？

在较短时间间隔内，程序产生的地址/所访问数据往往集中在存储器的一个很小范围内

- **时间局部性(Temporal Locality)：刚被访问过的存储单元很可能不久又被访问**

  让最近被访问过的信息保留在靠近CPU的存储器中

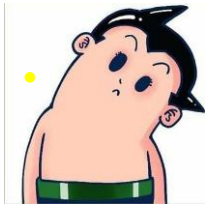
- **空间局部性 (Spatial Locality)：刚被访问过的存储单元的邻近单元很可能不久被访问**

  将刚被访问过的存储单元的邻近单元调到靠近CPU的存储器中

回顾——Cache - 主存层次的平均访问时间

- 命中（Hit）：要访问的信息在Cache中
  - Hit Rate(命中率 *p*)： 在Cache中的概率
  - Hit Time (命中时间 Tc)：访问Cache所需时间，包括：判断时间 + Cache访问
- 失效（Miss）：要访问的信息不在Cache中
  - Miss Rate (失靶率/失效率 *1 - p*) = 1 - (Hit Rate)
  - Miss Penalty (失效损失 Tm)：从主存将一块信息替换到Cache所需时间，包括访问主存块，向上逐层传输块直至将数据块放入发生缺失的那一层所需时间。

命中时间<<失效损失



- 平均访问时间 = $p\times Tc+(1-p)\times(Tm+Tc)=Tc+(1-p)\times Tm$

提高平均访问速度，必须提高命中率！

回顾——5.3.2 Cache(高速缓存)是什么样?

Cache结构

- Cache是小容量、高速缓冲存储器，由SRAM组成
- Cache直接制作在CPU芯片内，速度几乎与CPU一样快
- 一般将Cache和主存的存储空间都划分为若干大小相同的块（主存中称为：块Block、Cache中称为：行line)

实现Cache机制需要解决的问题

- 主存块和Cache之间如何映射?
- 给出的主存地址怎么样转换为Cache地址?
- 写数据时，怎样保证Cache和MM一致性?
- Cache已满时，怎么办?

5.3.4 哪些因素影响了Cache的失效率?

5.3.4 哪些因素影响了Cache的失效率

Miss Rate和Cache大小、Block大小的关系

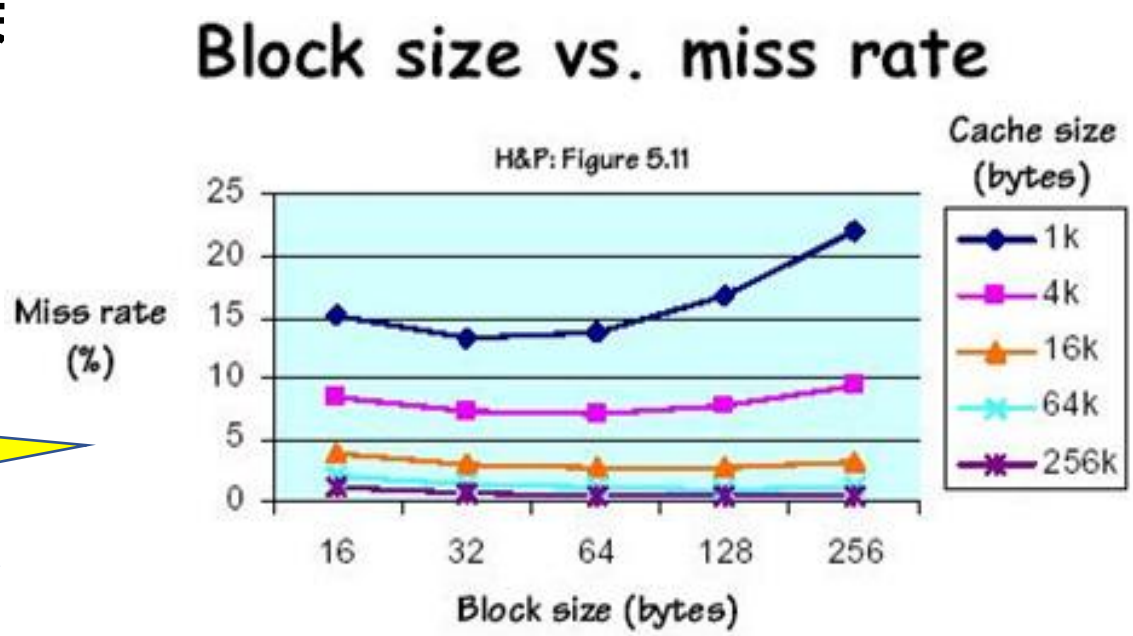
Cache性能由Miss Rate确定，而Miss Rate与Cache大小、Block大小、映射方式、Cache级数等有关

- **Cache大小：Cache越大，Miss Rate越低，但成本越高！**
- **Block大小：**
  - **Block越大，Miss Rate越低，因为空间局部性充分发掘**
  - **Block大小在Cache大小中所占的比例增加到一定程度时，Miss Rate也会随之增加**
    - **因为Cache Block的总数变少了**

5.3.4 哪些因素影响了Cache的失效率

Miss Rate和Cache大小、Block大小的关系

- 单纯增加Block大小带来一个更严重的问题是缺失损失增大
- 当Block较大时，缺失损失的上升超过了缺失率的降低，故Cache性能也相应降低

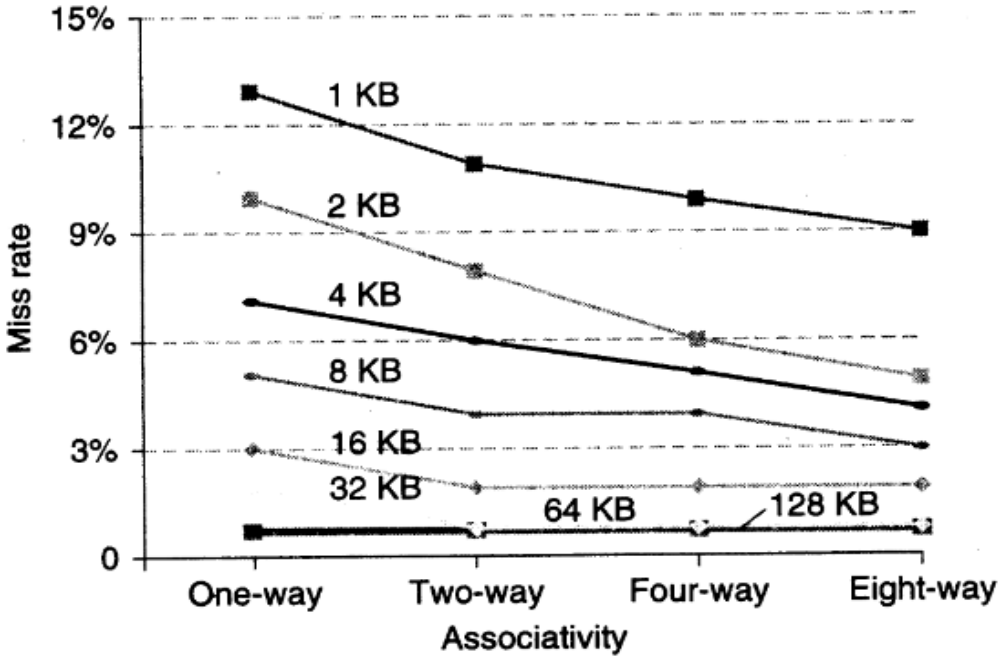


Block不能太大，也不能太小!

5.3.4 哪些因素影响了Cache的失效率

Miss Rate和Cache映射方式的关系

- Cache映射方式
  - Cache容量小时， Cache映射方式对Miss Rate有影响
  - Cache容量大时，Cache映射方式对Miss Rate影响不大
    - 映射到同一组的概率降低



5.3.4 哪些因素影响了Cache的失效率

Cache失效类型

强制失效(Compulsory misses)

- **首次访问某数据块时，必然引起的Cache失效**
- **增加Block大小，有利于减少此类不命中**

容量失效(Capacity misses)

- **Cache不能存放程序运行所需的所有块，替换后再次被使用所引起的Cache失效**
- **增加Cache大小，有利于减少此类不命中**

冲突失效(Conflict misses)

- **映射到同一组的数据块个数超过组内可容纳的块时，竞争所引起的Cache失效**
- **全相联没有此类失效，但价格贵且访问速度慢**

5.3.4 哪些因素影响了Cache的失效率

Cache失效类型

- **强制失效(Compulsory misses)**
- 容量失效(Capacity misses)
- 冲突失效(Conflict misses)

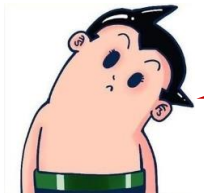
Cache抖动可能由哪些Cache失效引起的?

容量失效和冲突失效可能引起Cache抖动

- **某些块在主存和Cache之间频繁传送**
- **增加容量和相联性，有助于缓解这种现象**

5.3.5 Cache的一致性问题

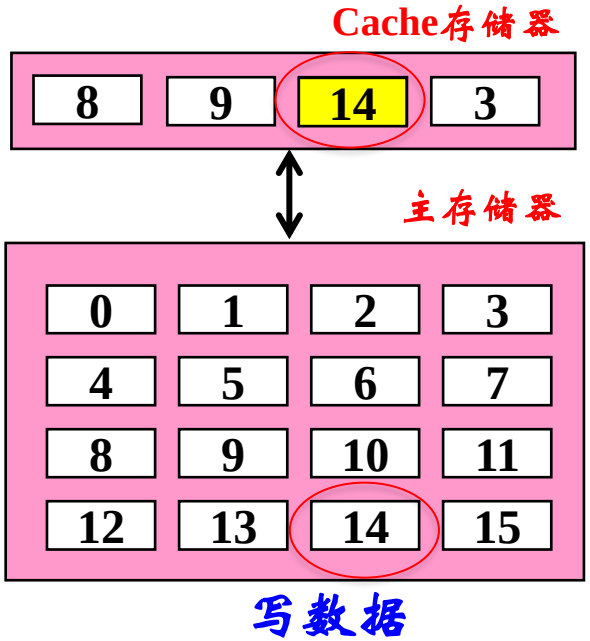
5.3.5 Cache的一致性问题



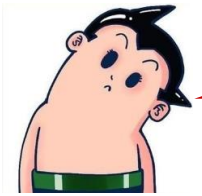
Cache的一致性问题指什么?

- Cache中的内容是主存的副本

情况1：当Cache中的内容进行更新时，而没有改变主存中的相应内容时，Cache和主存之间产生了不一致(inconsistent)



5.3.5 Cache的一致性问题



Cache的一致性问题指什么?

情况2：当多个设备都允许访问主存时

例：I/O设备可通过DMA 方式直接读写内存时，如果Cache中的内容被修改，则I/O设备读出的对应主存单元的内容无效；若I/O设备修改了主存单元的内容，则对应Cache行中的内容无效。

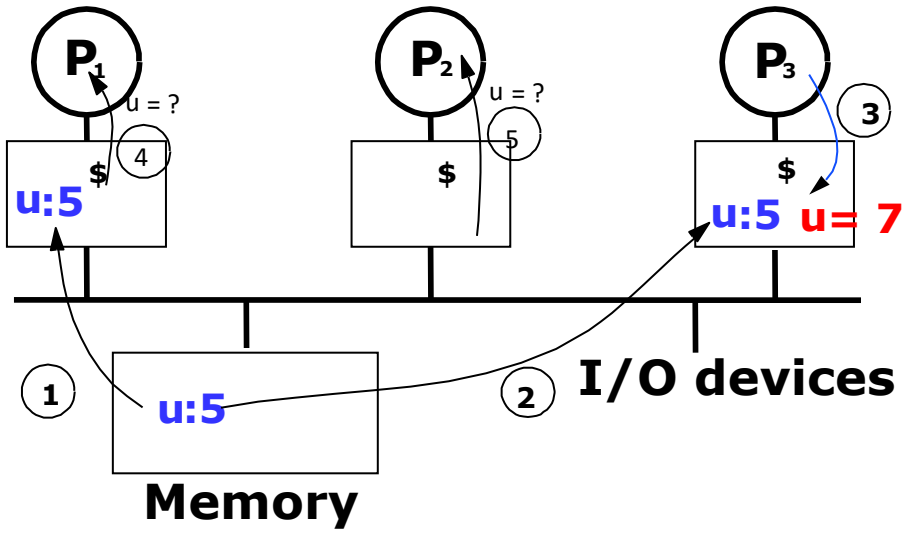
情况3：当多个CPU都有各自私有的Cache并且共享主存时

例：某个CPU修改了自身Cache中的内容，则对应的主存单元和其他CPU中对应的Cache行的内容都要变为无效。

5.3.5 Cache的一致性问题

处理器私有Cache出现的问题

- 同一变量拷贝可能出现在多个处理器私有Cache中
- 某处理器写操作可能对其它处理器是不可见的



P3私有Cache中的块u被更新后，各处理器读到的是不同的u值

程序不能容忍这样的错误， 但这种现象却很常见!

5.3.5 Cache的一致性问题

如何保持Cache一致性呢？

需要研究Cache写机制

- **Write Through (写直达、写通过、直写)**
- **Write Back (写回、一次性写、回写)**

5.3.5 Cache的一致性问题

Cache的写机制：Write Through（写直达、写通过、直写）

- 当一个写操作产生时，新值同时写到Cache和主存的块中
- 写直达会带来什么影响?

  Memory is too slow(>100Cycles)

  10%的存数指令会使CPI增加到：1.0+100×10%=11
- 在Cache和主存之间使用写缓冲(Write Buffer)
  - 当一个数据等待被写入主存时，先将其存入写缓冲；
  - 在把数据写入Cache和写缓冲后，处理器继续执行命令；
  - 当写主存操作结束后，写缓冲里的数据释放

5.3.5 Cache的一致性问题

Cache的写机制：Write Back (写回、一次性写、回写)

- 当一个写操作产生时，新值仅被写到Cache中，而不被写入主存
- 写回方式会带来什么影响?
  - 大大降低主存带宽需求，提高系统性能，控制可能很复杂
  - 每个Cache行都设置一个修改位（“dirty bit-脏位”），如果对应Cache行中的主存块被修改，就同时置修改位为“1”，如果修改位为“0”，则说明对应主存块没有被修改过
  - 只有当修改位为“1”的块从cache中替换出去时，才把它写回主存

5.3.5 Cache的一致性问题

如何保持Cache一致性呢？

写命中(Write Hit)
要写的单元已经在Cache中

- **Write Through**
  (写通过、写直达、直写)
- **Write Back**
  (一次性写、写回、回写)

写不命中(Write Miss)
要写的单元不在Cache中

- **Allocate-on-miss (写分配)**
- **No-allocate-on-write (写不分配)**

5.3.5 Cache的一致性问题

写不命中时如何处理?

- **Allocate-on-miss (写分配)**

  更新主存块中相应单元，再将该主存块装入Cache；
  试图利用空间局部性，但增加了从主存读入一个块的开销

- **No-allocate-on-write (写不分配)**

  直接写主存，不把被写数据放入Cache；
  减少了从主存读一个块的开销，但没有利用空间局部性

1、写直达Cache可用写分配或写不分配
2、写回Cache通常用写分配

5.3.5 Cache的一致性问题

问题1：以下描述的是哪种写策略?

写直达 、写分配!

问题2：如果用写不分配， 则如何修改算法?

ON REFERENCE TO Mem[X]: Look for X among tags...

HIT: *X == TAG(i) , for some cache line i*

READ: return DATA[I]
WRITE: change DATA[I]; Start Write to Mem[X]

MISS: *X not found in TAG of any cache line*

REPLACEMENT SELECTION:
Select some line k to hold Mem[X]

READ: Read Mem[X]
Set TAG[k] = X, DATA[k] = Mem[X]

WRITE: Start Write to Mem[X]
~~Set TAG[k] = X, DATA[k] = new Mem[X]~~

5.3.5 Cache的一致性问题

问题3：以下算法描述的是哪种写策略?

写回、写分配!

ON REFERENCE TO Mem[X]: Look for X among tags...

HIT: *X = TAG(i) , for some cache line I*

READ: return DATA(i)
WRITE: change DATA(i); ~~Start Write to Mem[X]~~

MISS: *X not found in TAG of any cache line*

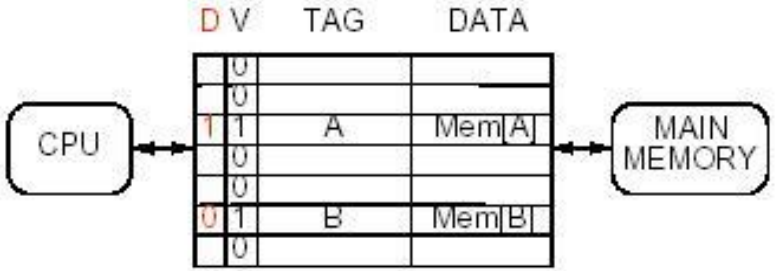
REPLACEMENT SELECTION:
~~Select some line k to hold Mem[X]~~
Write Back: Write Data(k) to Mem[Tag[k]]

READ: Read Mem[X]
Set TAG[k] = X, DATA[k] = Mem[X]

WRITE: ~~Start Write to Mem[X]~~
Set TAG[k] = X, DATA[k] = new Mem[X]

5.3.5 Cache的一致性问题

Write-back w/ "Dirty" bits



ON REFERENCE TO Mem[X]: Look for X among tags...

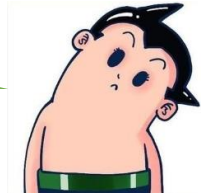
HIT: *X = TAG(i) , for some cache line I*
READ: return DATA(i)
WRITE: change DATA(i); ~~Start Write to Mem[X]~~ D[i]=1

MISS: *X not found in TAG of any cache line*
REPLACEMENT SELECTION:
Select some line k to hold Mem[X]
If D[k] == 1 (Write Back) Write Data(k) to Mem[Tag[k]]
READ: Read Mem[X]; Set TAG[k] = X, DATA[k] = Mem[X], D[k]=0
WRITE: ~~Start Write to Mem[X]~~ D[k]=1
Set TAG[k] = X, DATA[k] = new Mem[X]

5.3.6 Cache替换算法

5.3.6 Cache替换算法

什么时候需要进行Cache替换?

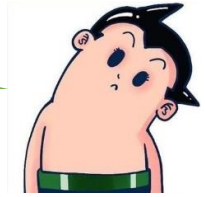


- Cache行数<<主存块数;
- 主存块和Cache行：多对1;
- 当一个新的主存块需要复制到Cache中时，如果Cache中的对应行已经全部被占满，怎么办?

选择淘汰掉一个Cache行中的块

5.3.6 Cache替换算法

什么时候需要进行Cache替换?



例：某Cache采用二路组相联映射，其数据区容量为16块。假定第0组的两个行分别存放了主存第0块和第8块，此时需调入主存第16块，根据映射关系，它只能放到Cache的第0组。第0组中必须调出一块，如何选择调出哪一块?

淘汰策略问题/替换算法

5.3.6 Cache替换算法

什么时候需要进行Cache替换?

直接映射（Direct Mapped）Cache

- 映射唯一，无条件用新信息替换老信息

N路组相联（N-way Set Associative）Cache

- 每个主存数据有N个Cache行可选择，需考虑替换哪一行

全相联（Fully Associative）Cache

- 每个主存数据可存放到Cache任意行中，需考虑替换哪一行

常用替换算法：

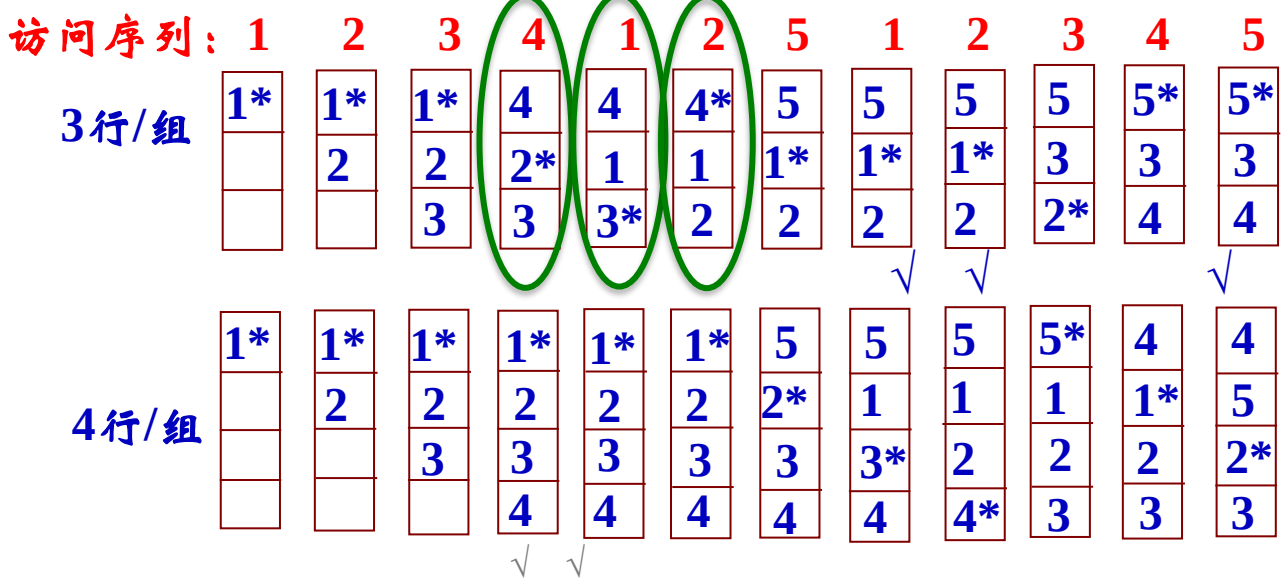
先进先出FIFO、最近最少用LRU、最不经常使用LFU、随机替换

5.3.6 Cache替换算法

先进先出（First In First Out，FIFO）算法

基本思想：总是把最先进入的那一块替换掉

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一访存序列，考察3行/组、 4行/组的情况。

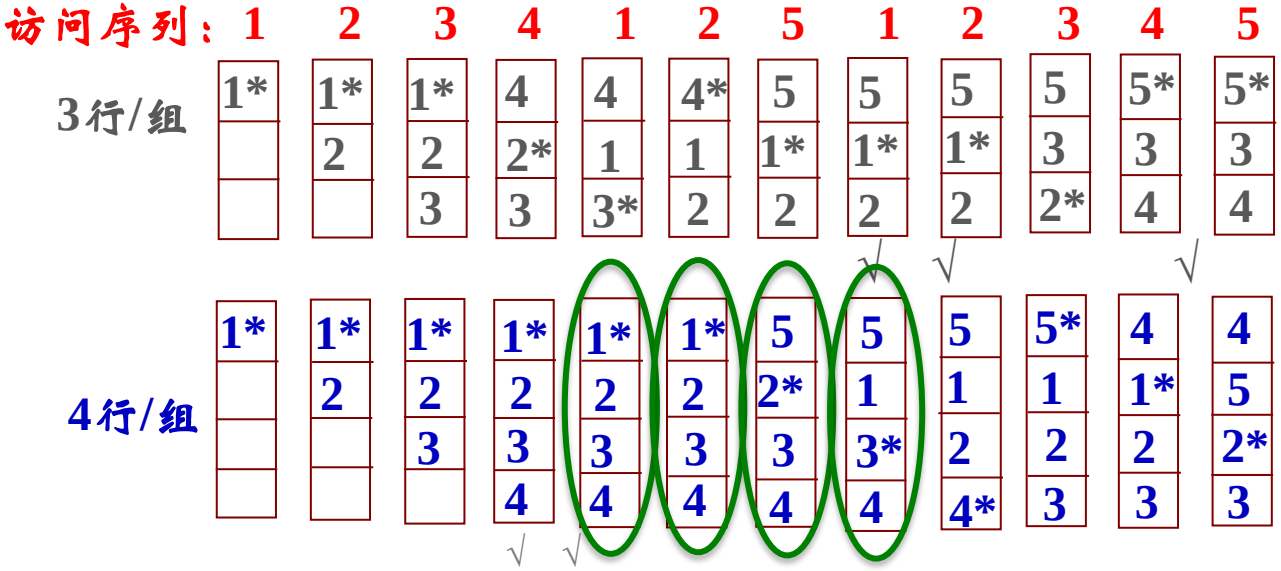


5.3.6 Cache替换算法

先进先出FIFO算法

基本思想：总是把最先进入的那一块替换掉

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一访存序列，考察3行/组、 4行/组的情况。



5.3.6 Cache替换算法

最近最少使用（Least Recently Used，LRU）算法

基本思想：总是把最近最少用的那一块淘汰掉，利用时间局部性

例：假定主存中的5块{1,2,3,4,5}同时映射到Cache同一组中，对于同一地址流，考察3行/组、 4行/组、 5行/组的情况。

访问序列：

1	2	3	4	1	2	5	1	2	3	4	5
1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4
		1	2	3	4	1	2	5	1	2	3

LRU算法的命中率随组中行数的增大而提高

3行/组　　√　√

5.3.6 Cache替换算法

最近最少使用LRU算法

- 当分块局部化范围(即：某段时间集中访问的存储区)超过了Cache存储容量时，命中率会变得很低。极端情况下，假设地址流是1,2,3,4,1,2,3,4,1,……，而Cache每组只有3行，那么，不管是FIFO，还是LRU算法，其命中率都为0。这种现象称为抖动(Thrashing / PingPong)
- LRU算法具体实现：通过给每个Cache行设定一个计数器，根据计数值来记录这些主存块的使用情况。这个计数值称为LRU位

5.3.6 Cache替换算法

最近最少使用LRU算法

- **计数器变化规则**
  - **每组4行时，计数器设2位。计数值越小，则说明越被常用**
  - **命中时，被访问行的计数器置0，其他行计数器加1，其余不变**
  - **未命中且该组未满时，新行计数器置为0，其余全加1**
  - **未命中且该组已满时，计数值最大的那一行中的主存块被淘汰，新行计数器置为0，其余加1**

访问序列：1 2 3 4 1 2 5 1 2 3 4 5

0	1	1	1	2	1	3	1	0	1	1	1	2	1	0	1	1	1	2	1	3	1	0	5
		0	2	1	2	2	2	3	2	0	2	1	2	2	2	0	2	1	2	2	2	3	4
				0	3	1	3	2	3	3	3	0	5	1	5	2	5	3	5	0	4	2	3
						0	4	1	4	2	4	3	4	3	4	3	4	0	3	1	3	1	2

注：表中蓝色表示计数器的值，红色表示Cache中存放的数据

5.3.6 Cache替换算法

随机（Random）替换算法

基本思想：随机地从候选的槽中选取一个淘汰，与使用情况无关

模拟试验表明，随机替换算法在性能上只稍逊于LRU算法，而且代价低！

5.3.6 Cache替换算法

例：假定计算机系统有一个容量为32K×16位的主存，且有一个4K字的4路组相联Cache，主存和Cache之间的数据交换块的大小为64字。

1、试分析Cache的结构和主存地址的划分

答：假定主存按字编址，每字为16位。

Cache：4K字=64*64字/行＝16组 × 4行 / 组 × 64 字 / 行

主存：32K字=512块 × 64字 / 块 = = 2^5 ×2^4块× 64字 / 块

主存地址划分为：

标志位	组号	字号
5	4	6

5.3.6 Cache替换算法

例：假定计算机系统有一个容量为32K×16位的主存，且有一个4K字的4路组相联Cache，主存和Cache之间的数据交换块的大小为64字。

设Cache开始为空，处理器顺序地从存储单元0、1、…、4351中取数，一共重复10次。Cache比主存快10倍，**设采用LRU策略**。

2、分析：采用Cache后速度提高了多少?

答：处理器顺序地从存储单元0、1、…、4351中取数

4352/64=68，

处理器的访问过程是对前68块连续访问10次

5.3.6 Cache替换算法

	第0 行	第1 行	第2 行	第3 行
第0组	0/64	16/	32	48
第1组	1/65	17	33	49
第2组	2/66	18	34	50
第3组	3/67	19	35	51
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

处理器的访问过程是对前68块连续访问10次

LRU算法分析：

第一次循环：对于每一块只有第一字未命中，其余都命中，缺失68次。

5.3.6 Cache替换算法

	第0 行	第1 行	第2 行	第3 行
第0组	0/64	16/0	32	48
第1组	1/65	17	33	49
第2组	2/66	18	34	50
第3组	3/67	19	35	51
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

处理器的访问过程是对前68块连续访问10次

LRU算法分析：

第一次循环：对于每一块只有第一字未命中，其余都命中，缺失68次。

5.3.6 Cache替换算法

	第0 行	第1 行	第2 行	第3 行
第0组	0/64	16/0	32	48
第1组	1/65	17/1	33	49
第2组	2/66	18/2	34	50
第3组	3/67	19/3	35	51
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

LRU算法分析：

第一次循环：对于每一块只有第一字未命中，其余都命中，缺失68次。

5.3.6 Cache替换算法

	第0 行	第1 行	第2 行	第3 行
第0组	0/64/48	16/0/64	32/16	48/32
第1组	1/65/49	17/1/65	33/17	49/33
第2组	2/66/50	18/2/66	34/18	50/34
第3组	3/67/51	19/3/67	35/19	51/35
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

LRU算法分析：

第一次循环：对于每一块只有第一字未命中，其余都命中，缺失68次。

后9次循环：有20块的第一字未命中，其余都命中。

5.3.6 Cache替换算法

	第0 行	第1 行	第2 行	第3 行
第0组	0/64/48	16/0/64	32/16	48/32
第1组	1/65/49	17/1/65	33/17	49/33
第2组	2/66/50	18/2/66	34/18	50/34
第3组	3/67/51	19/3/67	35/19	51/35
第4组	4	20	36	52
……	……	……	……	……
……	……	……	……	……
第15组	15	31	47	63

LRU算法分析：

第一次循环：对于每一块只有第一字未命中，其余都命中，缺失68次。

后9次循环：有20块的第一字未命中，其余都命中。

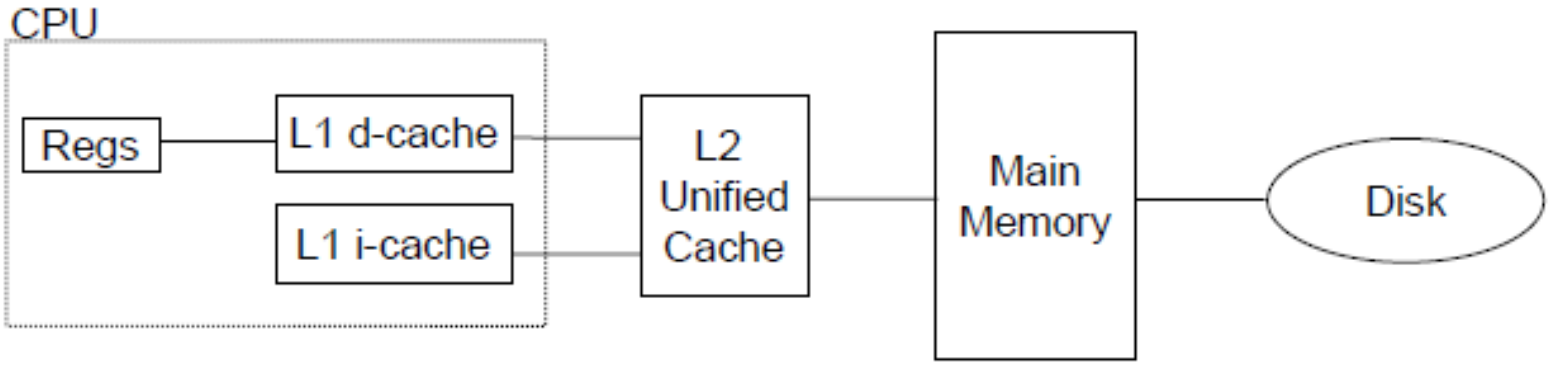
命中率p： (43520−68−9×20)/43520=99.43%

速度提高：$t_m/t_a=t_m/(t_c+(1-p)t_m)=10/(p+10\times(1-p))=9.5$倍

5.3.7 多级Cache

5.3.7 多级Cache

多级Cache系统成为主流：在Cache-Memory 系统中使用更多的层次结构，以掩盖CPU访存延迟，提高处理器的执行效率



一个典型的多级cache组织结构

5.3.7 多级Cache

多Cache系统设计的主要考虑因素：

(1) 单级/多级

- 片内(On-chip)Cache：将Cache和CPU作在一个芯片上
- 外部(Off-chip)Cache：不做在CPU内而是独立设置一个Cache
- 单级Cache：只用一个片内Cache
- 多级Cache：同时使用L1 、L2 Cache，有些系统还有L3 Cache

L1 Cache更靠近CPU，其速度比L2快，其容量比L2小

5.3.7 多级Cache

多Cache系统设计的主要考虑因素：

(2) 联合/分立

- **分立：数据和指令分开存放在各自的数据和指令Cache中**
- **联合：数据和指令都放在一个Cache中**

一般L1 Cache都是分立Cache，为什么？

L1 Cache的命中时间比命中率更重要！减少命中时间以获得较短的时钟周期

防止结构冒险

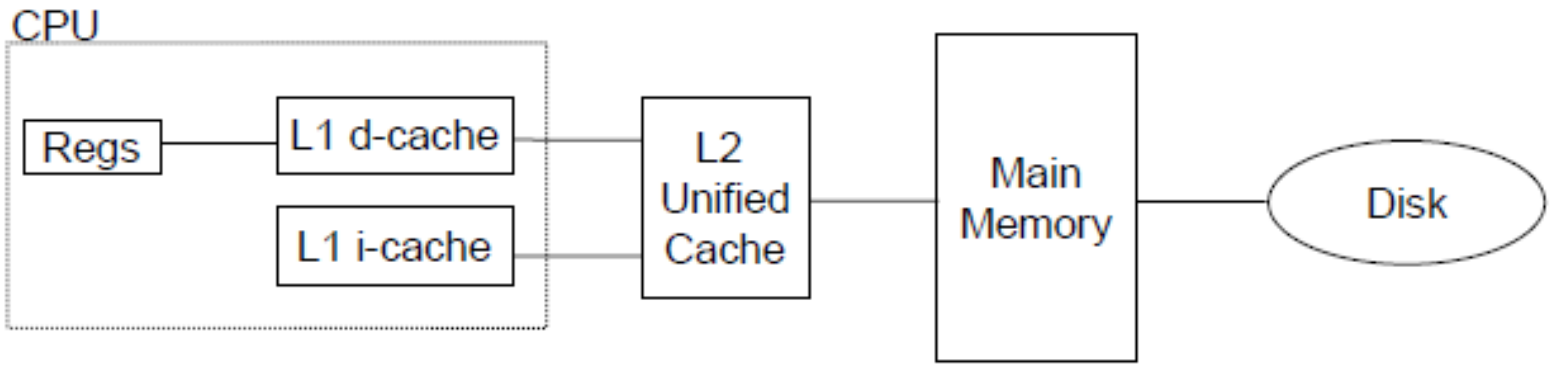
一般L2 Cache都是联合Cache，为什么？

L2 Cache的命中率比命中时间更重要！降低缺失率以减少访问主存缺失损失

5.3.7 多级Cache

两级Cache 系统的缺失损失(Miss Penalty)分析：

- **若L2 Cache包含所请求信息，则缺失损失为L2 Cache的访问时间**
- **否则要访问主存，并同时取到L1 Cache和L2 Cache(缺失损失更大)**



5.3.7 多级Cache

例：某处理器的CPI为1（如果所有访问能在L1 Cache命中），时钟频率为5GHz。假设访问一次主存的时间为100ns(包括所有的缺失处理)，设平均每条指令在L1 Cache中的缺失率为2%；如果增加一个L2 Cache，访问时间为5ns，而且容量大到使L2 Cache缺失率减为0.5%，问处理器速率提高了多少?

解：如果只有一级Cache，则缺失只有一种：

L1缺失(访问主存)，其缺失损失为：100ns×5GHz=500个时钟周期

总的CPI = CPI +每条指令中存储器停顿的时钟周期

= 1+ 500×2% = 11.0

5.3.7 多级Cache

例：某处理器的CPI为1（如果所有访问能在L1 Cache命中），时钟频率为5GHz。假设访问一次主存的时间为100ns(包括所有的缺失处理)，设平均每条指令在L1 Cache中的缺失率为2%；如果增加一个L2 Cache，访问时间为5ns，而且容量大到使L2 Cache缺失率减为0.5%，问处理器速率提高了多少?

解：如果只有一级Cache，则缺失只有一种：

L1缺失(访问主存)，其缺失损失为：100ns×5GHz=500个时钟周期

总的CPI = CPI +每条指令中存储器停顿的时钟周期

= 1+ 500×2% = 11.0

如果有二级Cache，则有两种缺失：

L1缺失(访问L2Cache)：5ns×5GHz = 25个时钟周期

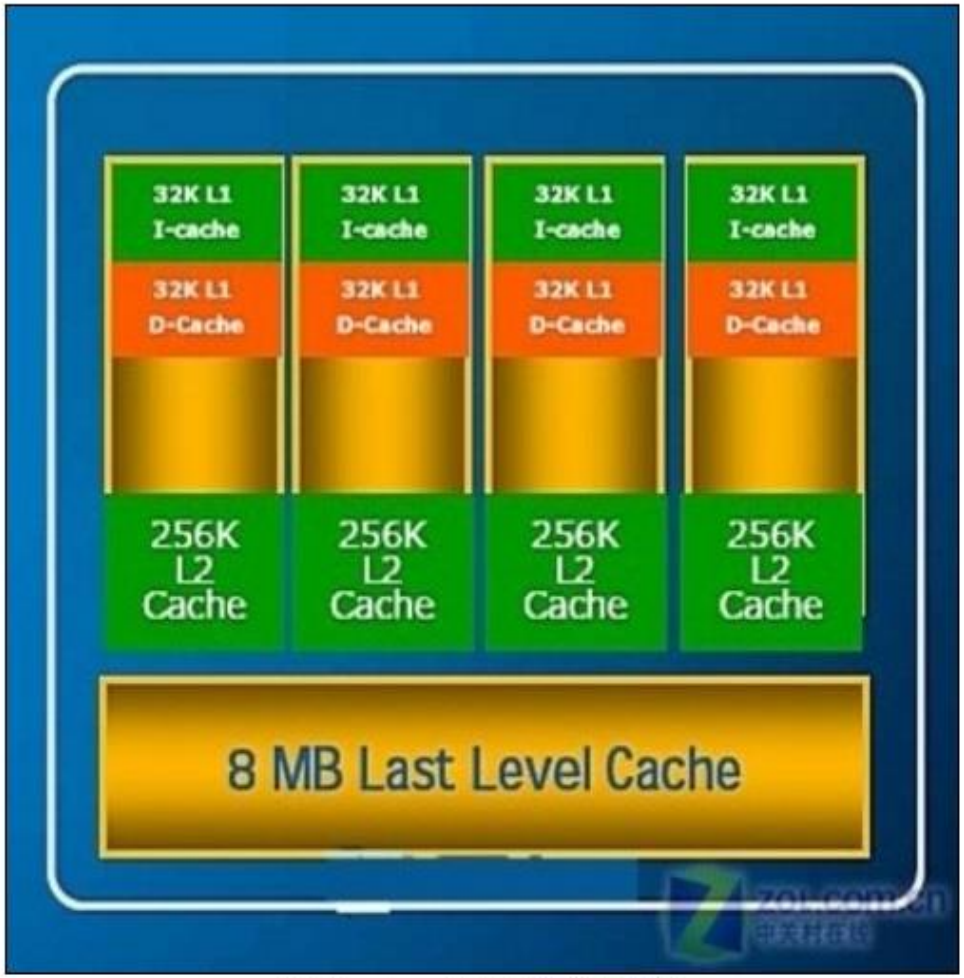
L2缺失(访问主存)：100ns×5GHz=500个时钟周期

总的CPI = CPI + 每条指令的一级停顿时钟周期+二级停顿的时钟周期

= 1+ 25×2% + 500×0.5% = 4.0

因此，二者的性能比为11.0/4.0=2.8倍

5.3.7 多级Cache



Nehalem Core i7处理器缓存结构图

Per core:

-32KB, 4-way L1 $I

-32KB, 8-way L1 $D

-256KB, 8-way L2

Shared

- 8 MB, 16-way L3

5.3.7 多级Cache

缓存技术的应用很广泛

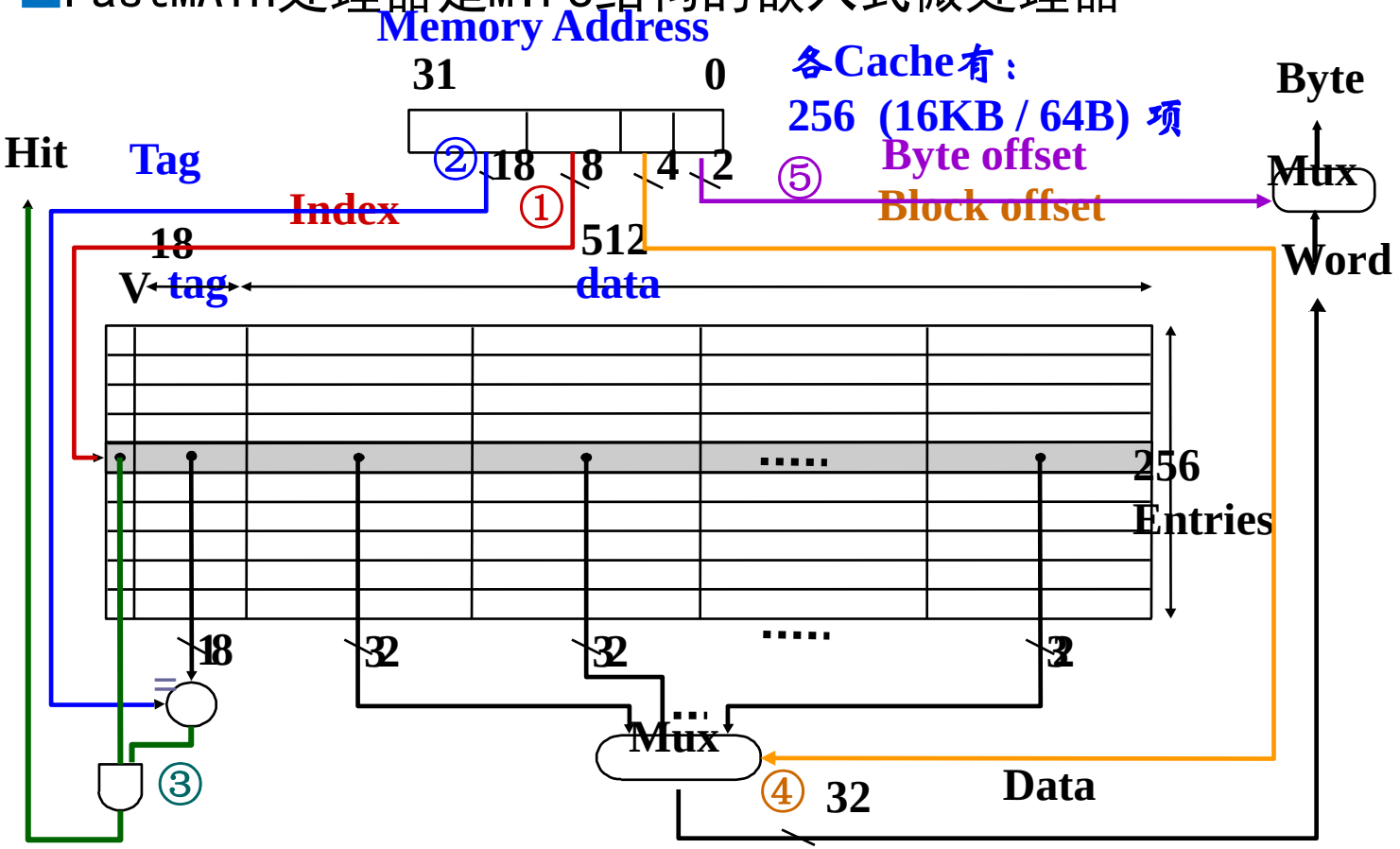
Type	What cached	Where cached	Latency (cycles)	Managed by
CPU registers	4-byte word	On-chip CPU registers	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware
L1 cache	32-byte block	On-chip L1 cache	1	Hardware
L2 cache	32-byte block	Off-chip L2 cache	10	Hardware
Virtual memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

缓存技术的基本思想

充分利用程序访问的局部性特点，将大容量、慢速存储器中当前刚用过的局部数据复制或暂存在小容量、快速存储器中，提高计算机系统访问效率

实例：内置FastMATH处理器

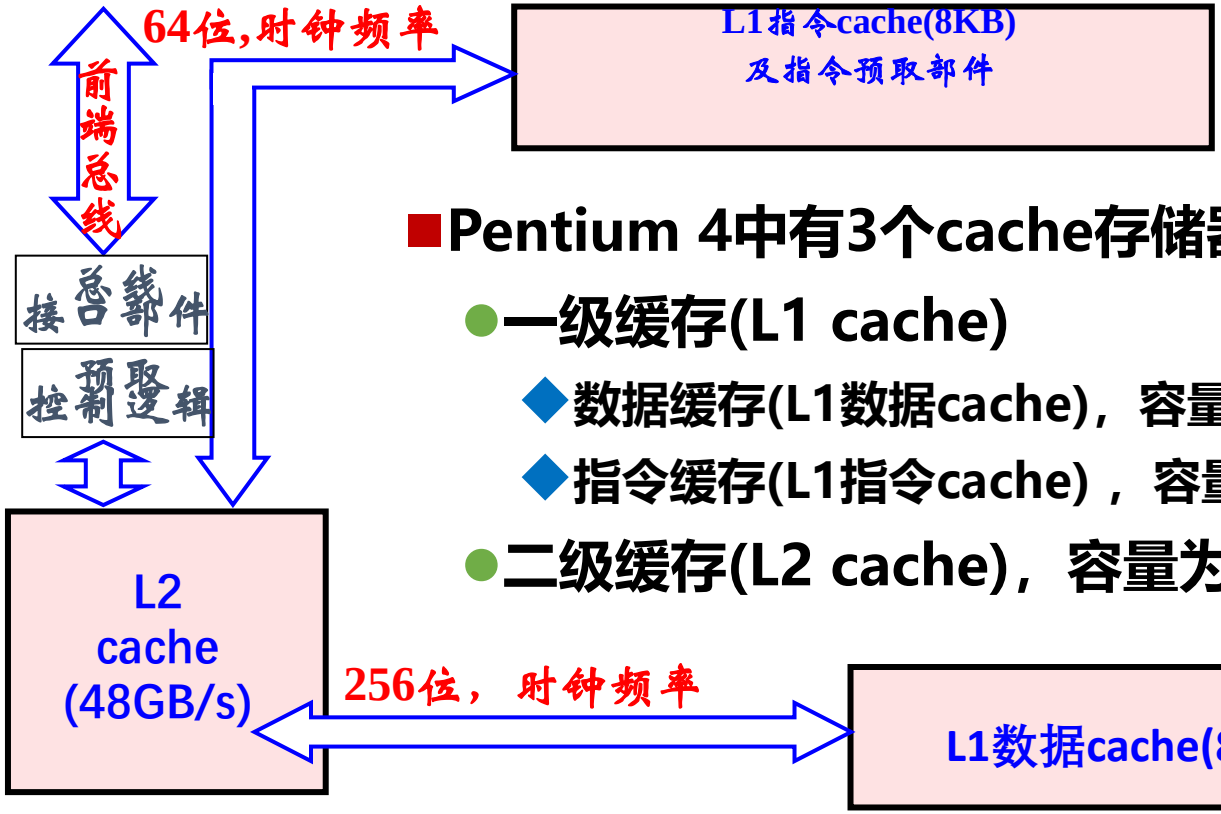
- FastMATH处理器是MIPS结构的嵌入式微处理器



■采用12级流水线结构，指令/数据Cache分开

■处理器提供写直达和写回两种方式，由OS决定

实例：Pentium 4的cache存储器



- Pentium 4中有3个cache存储器，分成：
  - 一级缓存(L1 cache)
    - 数据缓存(L1数据cache)，容量为8KB
    - 指令缓存(L1指令cache)，容量为8KB
  - 二级缓存(L2 cache)，容量为256 KB～2MB

联系方式

- Acknowledgements:
- This slides contains materials from following lectures:
  - Computer Architecture (ETH，NUDT，USTC，SYSU)

- Research Area：
  - 计算机视觉与机器人应用计算加速，
  - 人工智能和深度学习芯片及智能计算机

- Contact：
  - 中山大学计算机学院
  - 管理学院D101（图书馆右侧）
  - 机器人与智能计算实验室
  - cheng83@mail.sysu.edu.cn