

# **处理器体系结构**

## **第五章 高速缓存存储器A**

### **--Cache**

#### **(Micro-processor Architecture)**

---

# 目录

---

## 5.1 高速缓存存储器

### 5.1.1 存储器组织结构

### 5.2.2 Cache的基本结构




















### 5.2.3 改善Cache性能

### 5.2.4 改善DRAM性能

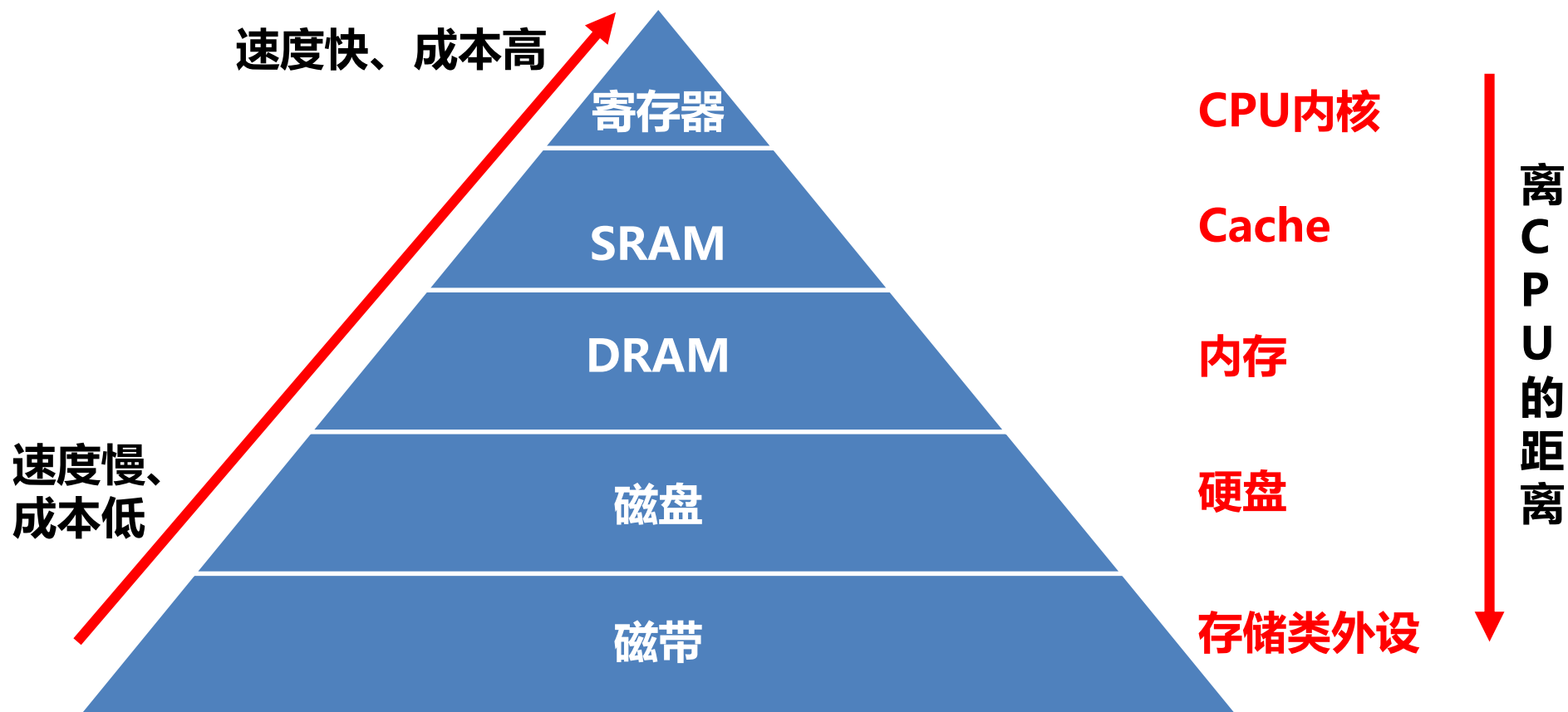
## 5.1.1 存储器组织结构--什么是Cache?

### 维基百科的定义:

In computing, a **cache** is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere.

 <b>Cache</b> 修改日期: 2019/4/14 20:53	C:\用户\min\AppData\Roaming\360se6\User Data\Default
 <b>Cache</b> 修改日期: 2019/4/14 20:51	C:\用户\min\AppData\Local\Netease\CloudMusic\Cache
 <b>Caches</b> 修改日期: 2019/4/14 20:20	C:\用户\min\AppData\Local\Microsoft\Windows
 <b>cache</b> 修改日期: 2019/4/14 20:09	C:\用户\min\AppData\Roaming\youku
 <b>Cache</b> 修改日期: 2019/4/14 19:30	C:\用户\min\AppData\Roaming\360Safe\SoftMgr
 <b>Cache</b> 修改日期: 2019/4/14 17:02	C:\用户\min\AppData\Local\Microsoft\TokenBroker
 <b>Cache</b> 修改日期: 2019/4/14 17:01	C:\用户\min\AppData\Local\Temp\chromeshell\Default
 <b>CachedFiles</b> 修改日期: 2019/4/14 16:59	C:\用户\min\AppData\Roaming\Microsoft\Windows\Themes
 <b>cache</b> 修改日期: 2019/4/14 15:51	C:\用户\min\AppData\Roaming\360mobilemgr
 <b>Media Cache</b> 修改日期: 2019/4/14 15:09	C:\用户\min\AppData\Roaming\360se6\User Data\Default
 <b>cache</b> 修改日期: 2019/4/14 9:05	C:\用户\min\AppData\Roaming\Macromedia\Flash Player\openssl
 <b>NV_Cache</b> 修改日期: 2019/4/13 21:09	C:\用户\min\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\AC\Temp\NVID...
 <b>NV_Cache</b> 修改日期: 2019/4/13 21:09	C:\用户\min\AppData\Local\Packages\Microsoft.LockApp_cw5n1h2txyewy\AC\Temp\NVIDIA Corpo...
 <b>dict.cache</b> 修改日期: 2019/4/13 21:08	C:\用户\min\AppData\Local\Yodao\DeskDict
 <b>Cache</b> 修改日期: 2019/4/10 18:08	C:\用户\min\AppData\Local\Packages\Microsoft.Windows.Cortana_cw5n1h2txyewy\AC\TokenBroker
 <b>Doc Cache</b> 修改日期: 2019/4/10 4:42	C:\用户\min\AppData\Roaming\jisutodo
 <b>cache</b> 修改日期: 2019/4/9 16:55	C:\用户\min\AppData\Roaming\MathWorks\MATLAB\R2015b\sim scape
 <b>cache</b> 修改日期: 2019/4/7 23:56	C:\用户\min\AppData\Roaming\IQIYI Video\PluginConfig\faIcon_qyfragment\data
 <b>Cache</b>	C:\用户\min\AppData\Roaming\IQIYI Video\PluginConfig\faIcon_qyfragment\data\cache

## 5.1.1 存储器组织结构--存储器的单位容量成本

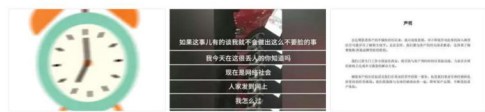


为什么这样安排存储结构?

## 5.1.1 存储器组织结构--程序的局部性原理

### 身处大数据时代：

官方新回应：退车退款！女车主：被奔驰一句话伤了自尊！涉事4S店被曝多次坑骗消费者



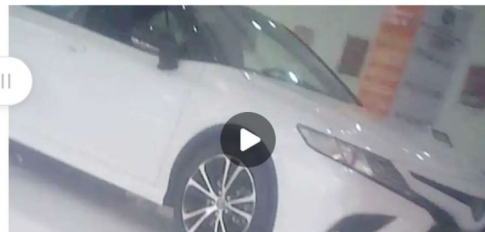
21世纪经济报道 1.2万评论 1小时前

奔驰大哥有难，宝马想也没想，连忙送出一波死亡助攻



爱开车的山羊哥 517评论 1小时前

暗访丰田4S店，亚洲龙上市后凯美瑞能优惠多少？



西瓜视频 小视频 未登录

### 思考：程序的语言中的loop 语句的执行特点

- Loop 中的指令和数据，在短的时间内，被重复的执行（满足时间局部性原理）
- Loop 中的数据通常以数组的格式存放，数组的格式都是以相邻的单元存放（满足空间局部性原理）

**时间的局部性：**如果一个数据块被访问，则短时间内这个数据块很可能被再访问

**空间的局部性：**如果一个数据块被访问，则与其相邻数据块也会不久被访问

-> 可用于解决存储器**速度和成本的矛盾**

### 5.1.1 存储器组织结构--存储器组织结构概念 (Memory hierarchy)

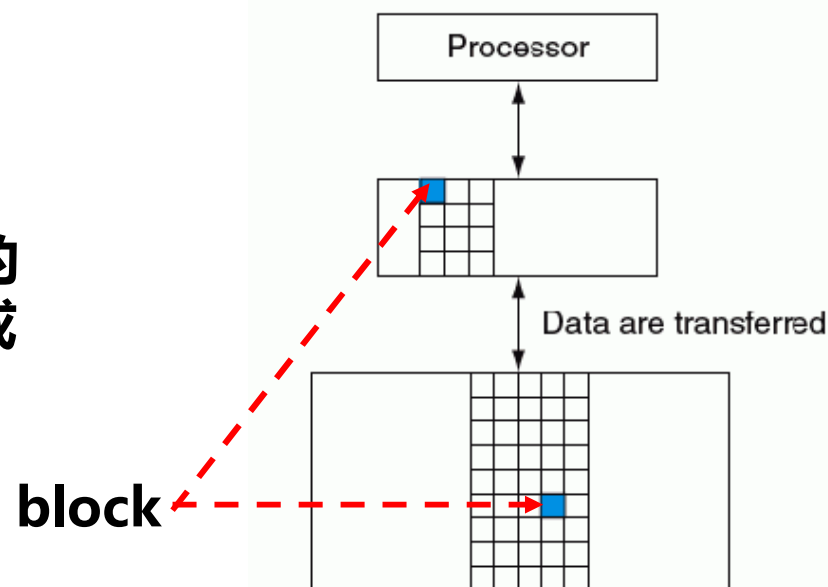
**存储器组织结构：**使用多级不同速度和容量的存储器，存储器与处理器距离越远，则容量越大，但访问时间越长

- 紧挨着处理器的存储器称之为cache，由SRAM 来实现 (典型访问时间：0.5~2.5ns)
- 主存储器 (main memory) 由DRAM 来实现 (典型访问时间：50~70ns)
- 离处理器最远，访问时间最长，容量最大的由磁盘来实现 (典型访问时间：5~20ms)

**数据的访问：**存储器的组织结构包含多级存储器，但数据的复制仅发生在相邻的2 级之间

**Block块：**

**Block 是两级存储器中交换的最小信息单元，其中或存信息或不存信息**



---

### 5.1.1 存储器组织结构--存储器组织结构概念 (Memory hierarchy)

---

Year introduced	Chip size	\$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	\$1,500,000	250 ns	150 ns
1983	256 Kibibit	\$500,000	185 ns	100 ns
1985	1 Mebibit	\$200,000	135 ns	40 ns
1989	4 Mebibit	\$50,000	110 ns	40 ns
1992	16 Mebibit	\$15,000	90 ns	30 ns
1996	64 Mebibit	\$10,000	60 ns	12 ns
1998	128 Mebibit	\$4,000	60 ns	10 ns
2000	256 Mebibit	\$1,000	55 ns	7 ns
2004	512 Mebibit	\$250	50 ns	5 ns
2007	1 Gibibit	\$50	45 ns	1.25 ns
2010	2 Gibibit	\$30	40 ns	1 ns
2012	4 Gibibit	\$1	35 ns	0.8 ns

**FIGURE 5.5 DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter considerably slower.** The improvements in access time have been slower but continuous, and cost roughly tracks density improvements, although cost is often affected by other issues, such as availability and demand. The cost per gibibyte is not adjusted for inflation.

---

## 5.1.1 存储器组织结构--存储器访问的几个概念

---

- ◆ hit (命中) : 在当前级的存储器中找到目标
- ◆ Miss (缺失) : 在当前级存储器中没找到目标
- ◆ Hit rate (命中率) : 在存储器中找到目标的比率
- ◆ Miss rate (缺失率) : 在存储器中未找到目标的比率
- ◆ Hit time (命中时间) : 访问某层存储器的时间, 包括判断访问是否命中的时间
- ◆ Miss penalty (缺失弥补时间) : 发生缺失时, 将目的块从底层取出传送到发生缺失的层, 并插入到发生缺失的层所花的时间



---

# 目录

---

## 5.1 高速缓存存储器

### 5.1.1 存储器组织结构

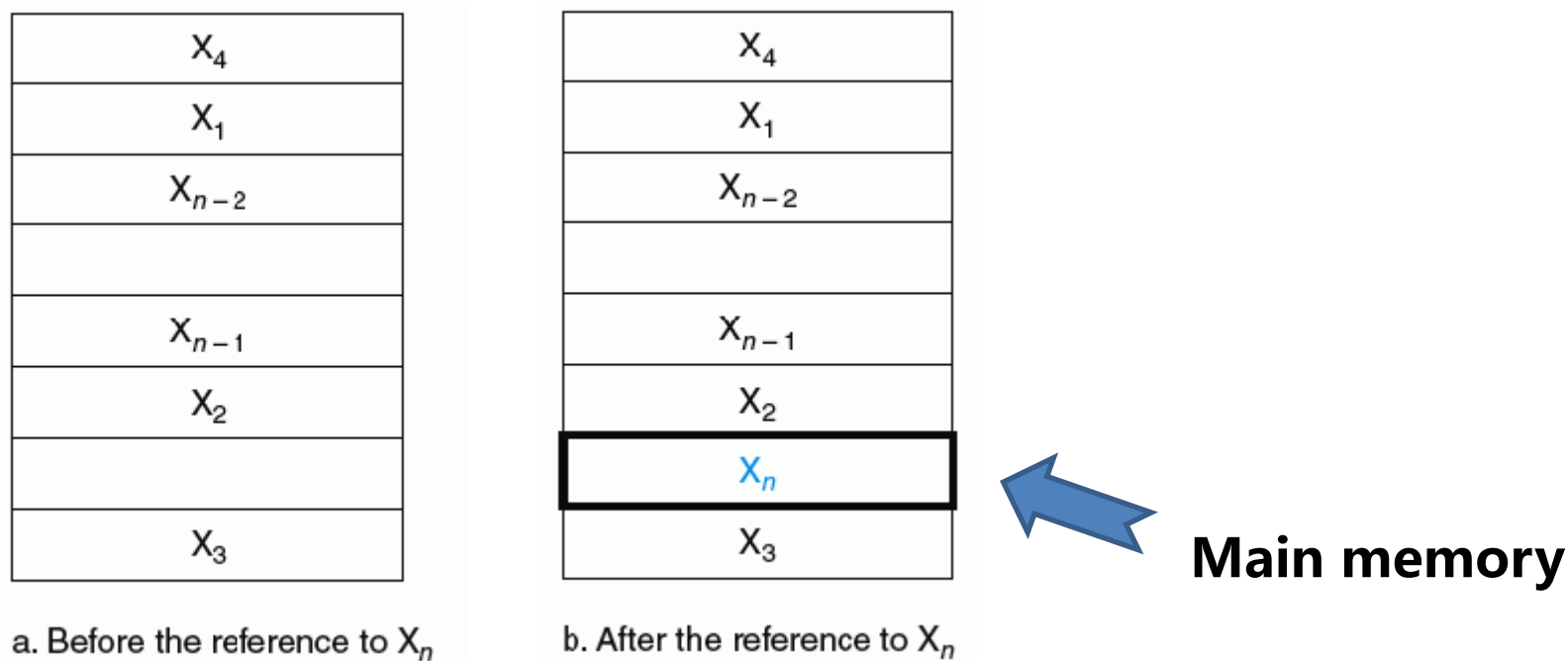
### 5.2.2 Cache的基本结构

### 5.2.3 改善Cache性能

### 5.2.4 改善DRAM性能

## 5.2.2 Cache的基本结构--Cache概念及查找

按照离CPU距离将存储器分层，则如果一个数据块不在第 $i + 1$ 层，第 $i$ 层那么肯定就不会在第 $i$ 层



图a: 在Cache中寻找 $X_n$ 未找到，于是就产生了缺失

图b: 启动主存储器(DRAM)，查找 $X_n$ 并传送到Cache

---

## 5.2.2 Cache的基本结构--在Cache 中数据定位

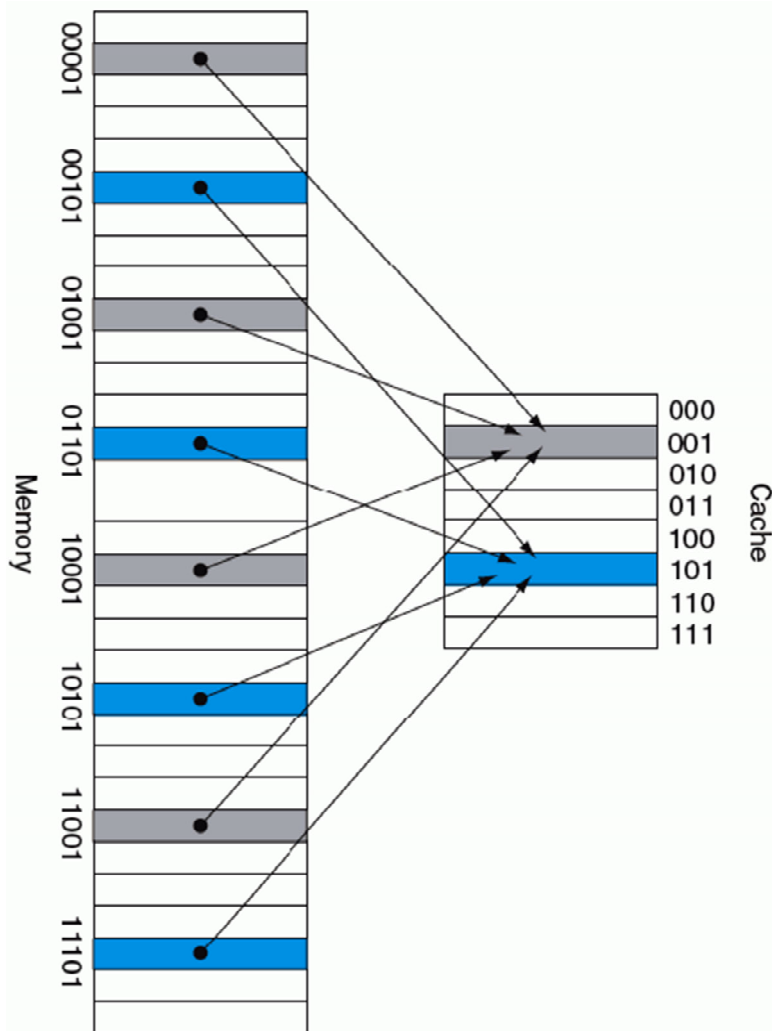
---

**问题1: 怎样知道一个数据项是否在cache 中?**

**问题2: 如果在cache中, 又怎样能找到数据?**

- Cache中的每个数据单元都有固定的位置和编址, 地址范围为2的幂次方, 远少于内存范围
- Cache与内存地址的关系为**直接映射**: 取内存地址的低位部分表示数据地址

## 5.2.2 Cache的基本结构--在Cache 中数据定位



问题3:

由于从大空间到小空间的映射，会导致多个地址映射到cache 中同一个单元，那么**Cache中的数据到底来自哪个内存单元？**

解决方案？：

- 增加标志位tag，保存数据项的高位地址
- 增加有效位V，表示数据项的地址是否有效

在处理器上电reset 或初始化时，用V指明tag地址位的有效或无效

## 5.2.2 Cache的基本结构-- Cache的寻址过程

1.产生Miss,  
导致Load

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

装入

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory(10110 <sub>two</sub> )
111	N		

b. After handling a miss of address (10110<sub>two</sub>)

## 5.2.2 Cache的基本结构-- Cache的寻址过程

### 2. 产生Miss, 导致Load

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	<b>010</b>	<b>miss</b>

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory(10110 <sub>two</sub> )
111	N		

b. After handling a miss of address (10110<sub>two</sub>)

装入

Index	V	Tag	Data
000	N		
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	N		
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memorily (10110 <sub>two</sub> )
111	N		

c. After handling a miss of address (11010<sub>two</sub>)

## 5.2.2 Cache的基本结构-- Cache的寻址过程

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	010	miss
10110	<b>110</b>	<b>hit</b>
11010	<b>010</b>	<b>hit</b>

3. 命中

4. 命中

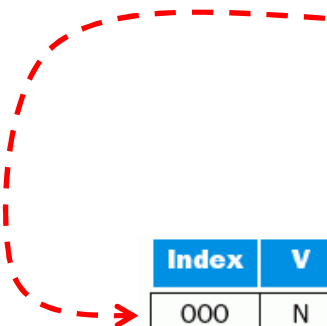
Index	V	Tag	Data
000	N		
001	N		
010	Y	$11_{\text{two}}$	Memory ( $11010_{\text{two}}$ )
011	N		
100	N		
101	N		
110	Y	$10_{\text{two}}$	Memory ( $10110_{\text{two}}$ )
111	N		

c. After handling a miss of address ( $11010_{\text{two}}$ )

## 5.2.2 Cache的基本结构-- Cache的寻址过程

### 5. 产生Miss, 导致Load


数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	010	miss
10110	110	Hit
11010	010	Hit
10000	000	miss



Index	V	Tag	Data
000	N		
001	N		
010	Y	$11_{two}$	Memory ( $11010_{two}$ )
011	N		
100	N		
101	N		
110	Y	$10_{two}$	Memory ( $10110_{two}$ )
111	N		

c. After handling a miss of address ( $11010_{two}$ )

装入



Index	V	Tag	Data
000	Y	$10_{two}$	Memory ( $10000_{two}$ )
001	N		
010	Y	$11_{two}$	Memory ( $11010_{two}$ )
011	N		
100	N		
101	N		
110	Y	$10_{two}$	Memory ( $10110_{two}$ )
111	N		

d. After handling a miss of address ( $10000_{two}$ )



## 5.2.2 Cache的基本结构-- Cache的寻址过程

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	010	miss
10110	110	Hit
11010	010	Hit
10000	000	miss
00011	011	miss

6. miss

Index	V	Tag	Data
000	Y	$10_{two}$	Memory ( $10000_{two}$ )
001	N		
010	Y	$11_{two}$	Memory ( $11010_{two}$ )
011	N		
100	N		
101	N		
110	Y	$10_{two}$	Memory ( $10110_{two}$ )
111	N		

d. After handling a miss of address ( $10000_{two}$ )

装入


Index	V	Tag	Data
000	Y	$10_{two}$	Memory ( $10000_{two}$ )
001	N		
010	Y	$11_{two}$	Memory ( $11010_{two}$ )
011	Y	$00_{two}$	Memory ( $00011_{two}$ )
100	N		
101	N		
110	Y	$10_{two}$	Memory ( $10110_{two}$ )
111	N		

e. After handling a miss of address ( $00011_{two}$ )

## 5.2.2 Cache的基本结构-- Cache的寻址过程

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	010	miss
10110	110	Hit
11010	010	Hit
10000	000	miss
00011	011	miss
10000	000	hit

7. hit



Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	Y	00 <sub>two</sub>	Memory (00011 <sub>two</sub> )
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

e. After handling a miss of address (00011<sub>two</sub>)

## 5.2.2 Cache的基本结构-- Cache的寻址过程

数据地址	映射Cache的位置	Cache访问结果
10110	110	miss
11010	010	miss
10110	110	Hit
11010	010	Hit
10000	000	miss
00011	011	miss
10000	000	hit
10010	010	<b>miss</b>

8. Tag不相等  
,miss

Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	11 <sub>two</sub>	Memory (11010 <sub>two</sub> )
011	Y	00 <sub>two</sub>	Memory (00011 <sub>two</sub> )
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

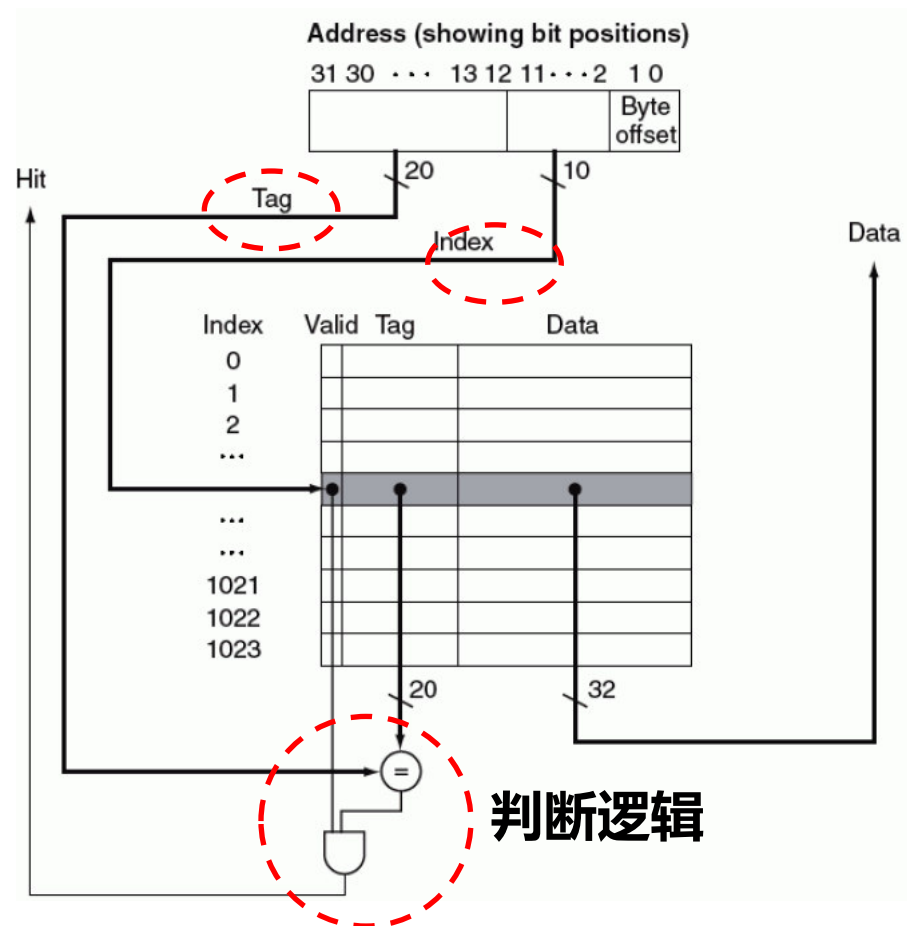
e. After handling a miss of address (00011<sub>two</sub>)

装入

Index	V	Tag	Data
000	Y	10 <sub>two</sub>	Memory (10000 <sub>two</sub> )
001	N		
010	Y	10 <sub>two</sub>	Memory (10010 <sub>two</sub> )
011	Y	00 <sub>two</sub>	Memory (00011 <sub>two</sub> )
100	N		
101	N		
110	Y	10 <sub>two</sub>	Memory (10110 <sub>two</sub> )
111	N		

f. After handling a miss of address (10010<sub>two</sub>)

## 5.2.2 Cache的基本结构-- Index、tag、valid三者的关系



每个block有一个Tag，可以包含1到多个word

---

## 5.2.2 Cache的基本结构-- Cache 的宽度和容量

---

MIPS 有32 位的地址，设cache 有 $2^n$ 个block，每个block 有 $2^m$ 个word（或者 $2^{m+2}$ bytes）。则tag的位长有多少？每个Block的容量？整个Cache的容量？

tag 的位长： $32-(n+m+2)$

- n 位用来在cache 中寻址block
- m 位用来在block 中寻址word
- 2 位用来在word 中寻址byte

每个block的容量： $(2^m \times 32) + 32-(n+m+2)+1$  (valid位)

整个cache的容量：

$$2^n \times (2^m \times 32 + 32-(n+m+2)+1)$$

Tag位，即地址的高位部分

---

## 5.2.2 Cache的基本结构-- Cache 的宽度和容量

---

例1：某cache 要存储16KB 数据，每个block 有4 个word ，地址位为32bit ，问cache 总的容量需要多少？

解：

16KB /4 byte=4K word

4K word/4=1K blocks= $2^{10}$  blocks

每个block有数据位：  $4 \times 32 = 128$  bits

每个block有tag位：  $32-(10+2+2)=18$  bits

每个block有valid位： 1 bits

Cache的总存储量需要：  $1K \times (128+18+1) = 147K$  bit

---

## 5.2.2 Cache的基本结构-- Cache 的宽度和容量

---

**例2：对于一个拥有64 个blocks，每个block 有16 bytes的Cache，则字节地址1200映射的块号是多少？**

**解：**

$$1200/16=75>64$$

**映射的块号为11**

**p.s. Tag为： 00 0000 0000 0000 0000 0001**

---

# 目录

---

## 5.1 高速缓存存储器

### 5.1.1 存储器组织结构

### 5.2.2 Cache的基本结构

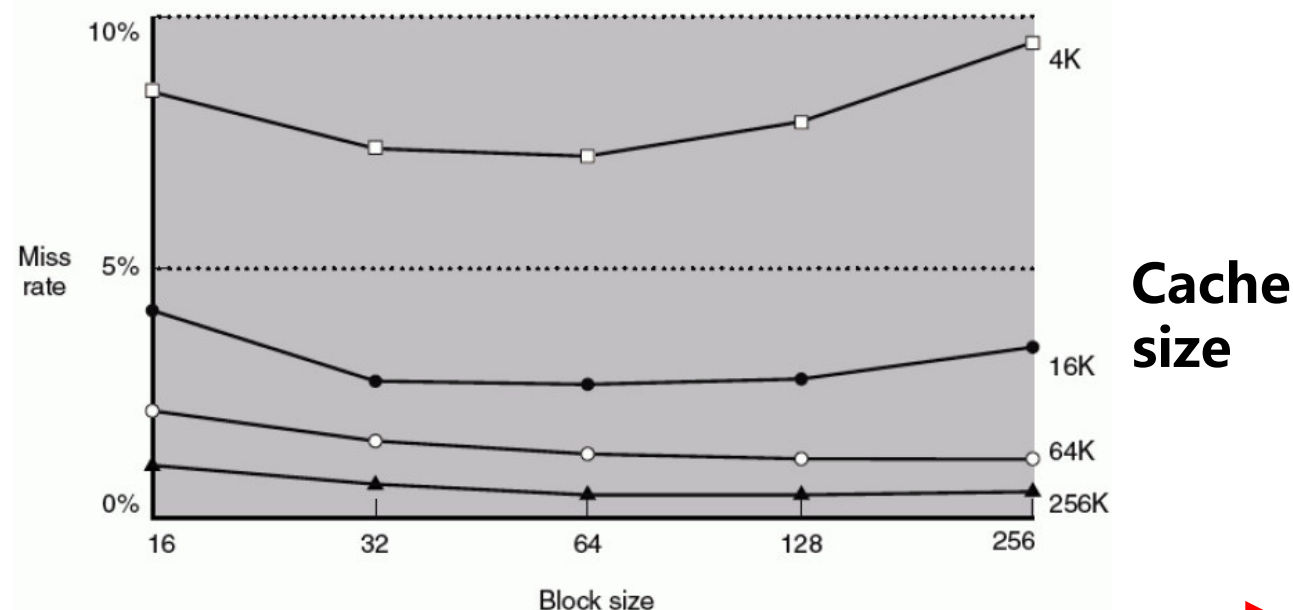
### 5.2.3 改善Cache性能

### 5.2.4 改善DRAM性能



## 5.2.3 改善Cache性能--Block 的容量与缺失率的关系

Block的容量适中时，缺失率较低；Block容量过大时，缺失率较高，因为block 容量过大，就会使cache中所容纳的Block的总数目减少



Block size

空间局部性不足

Block数量减少，产生大量竞争  
空间局部性带来的性能改善有限  
缺失时间变长

缺失时间 (miss penalty) 组成：从相邻存储器级中取出该块的时间 + 把该块装入到Cache的时间

缺失时间 = 取第一个字 (word) 的延迟 (latency) + 传送块内其它字(word) 的时间

---

## 5.2.3 改善Cache性能--缺失时间的改进措施

---

**尽早启动法：**一旦块中所请求的字取回到cache，处理器就立即启动恢复运行，而不必等待整个块传送完毕

- 采用上述方法对指令cache 采用效果较好。因为指令的执行多是串行的，若是每个周期传送一条指令，处理器的执行就会持续下去。
- 这个方法对数据cache 效果不好。因为所要求的数据字不可能包含在同一个block 中，当一个请求其它block 块字时，由于当前的block块正在传送，于是，处理器不得不停顿

**请求字优先方法：**请求的字被优先传送到Cache，而块中其它的字其后以请求字的地址为起始地址，延续传送直到块的开始地址

- 这种方法同样对数据cache的传送受到局限

**-> 缺失后如何处理？(读缺失、写缺失)**

---

## 5.2.3 改善Cache性能--读指令缺失 (Read Miss ) 的处理步骤

---

发生指令缺失时的处理:

- 发送PC-4的值 (PC的原始值) 给主存储器
- 指示主存储器(DRAM) 读, 并等待主存储器完成
- 将主存储器(DRAM) 返回的数据写cache的数据单元, 并把地址的高位填入到cache的tag字段, 同时置相应的有效位valid
- 恢复处理器的执行, 重取指令, 这次就会命中

数据cache读缺失时的处理基本相同

### 5.2.3 改善Cache性能--写cache的处理方法

**问题1：**仅写cache，而不写主存储器(DRAM) 就会导致Cache 和主存储器(DRAM) 的数据不一致！

**Write-through** (写直达)策略：写cache的时候，同时直接写入到主存储器(DRAM)中，以保持cache和主存储器(DRAM)数据的一致性

**问题2：**如果在cache 中找不到要写的地址，就会发生写缺失 (Write-Miss )

**解决方案：**当发生写缺失时，首先从主存储器(DRAM) 取回相应的block放置到cache 中，再把缺失的字同时写入到cache 和主存储器(DRAM)

**问题1：为什么写2次？**

**问题2：以上策略有什么缺点？**

➤ Write-through会导致处理器的性能下降。

例如，store指令，其CPI在不发生写缺失时是1个周期，而“写直达”却需要额外100个周期进行写主存储器(DRAM)。在SPEC2000 中store指令的出现比例10%，于是得综合CPI 为： $1 + 100 \times 10\% = 11$ ，性能降低十倍

**Write-around**(写绕回)策略：写的时候绕过cache，直接写入主存储器(DRAM)

**写绕回的好处：**有时候程序会连续写多次块(但不读取该块)，那么在处理缺失的时候，写绕回可以避免重复读取DRAM

---

## 5.2.3 改善Cache性能--写cache的处理方法

---

思考：写操作的关键“瓶颈”在哪里？ -> 写DRAM的延迟

- 为解决直接写主存储器(DRAM)的延迟，使用write-buffer：先把数据写入write-buffer，不用等待写主存储器(DRAM)完成，就恢复处理器的执行；直到主存储器(DRAM)写入完毕才释放write-buffer；(可缩短缺失时间)
  - 若存储器完成写操作的速度 < 处理器产生写操作的速度，那么再多的buffer也没用
  - 通常，增加write-buffer的深度，使之能容纳多个条目，可以减缓处理器的停顿

-> 更好的解决方案是减少写DRAM操作

---

## 5.2.3 改善Cache性能-- Write-back的处理策略

---

**Write-back** (写回)策略: 写的时候仅写cache 中对应的block , 仅当这个block 被替换时, 才写回主存储器(DRAM)

Note: We **must write the block back to memory if the data in the cache is dirty** and we have a cache miss. If we simply overwrote the block on a store instruction before we knew whether the store had hit in the cache (as we could for a write-through cache), we would destroy the contents of the block, which is not backed up in memory. A write-through cache can write the data into the cache and read the tag; if the tag mismatches, then a miss occurs. Because the cache is write-through, the overwriting of the block in the cache is not catastrophic since memory has the correct value

程序Cache不会dirty, 因为不会被修改; 而数据Cache需要增加一个dirty位来检查数据一致性:

Dirty	valid	Tag	block
-------	-------	-----	-------

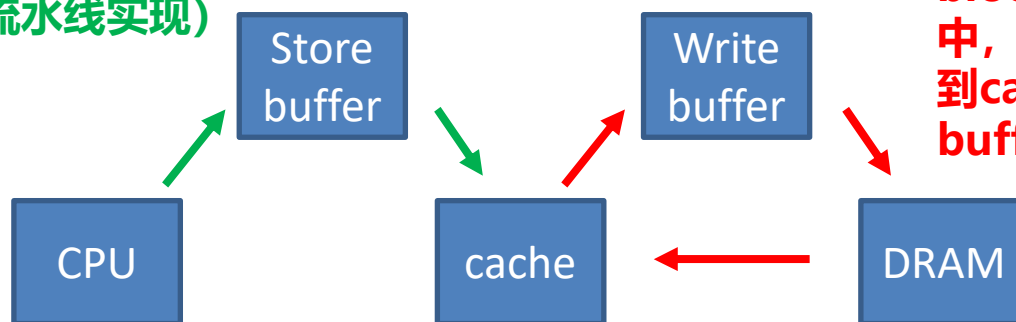
“dirty” 意味着cache中数据与主存不一致, 此时若简单地overwrite, 则会丢失数据

## 5.2.3 改善Cache性能-- Write-back的处理策略

对于“写回”，由于不能overwrite数据，所以store需要两个周期进行存数：第一个周期检查是否命中，第二个周期进行写操作；

相比而言，“写直达”可以overwrite数据，若不发生写缺失，则只需要一个周期写cache；(但这并不意味着写直达跟快，因为写直达还需要写DRAM，这会花费更多的周期)

为解决上述问题，“写回”还可以采用额外的store buffer，用于保存store的数据；此时若命中，则store可以在一个周期内完成写数（流水线实现）

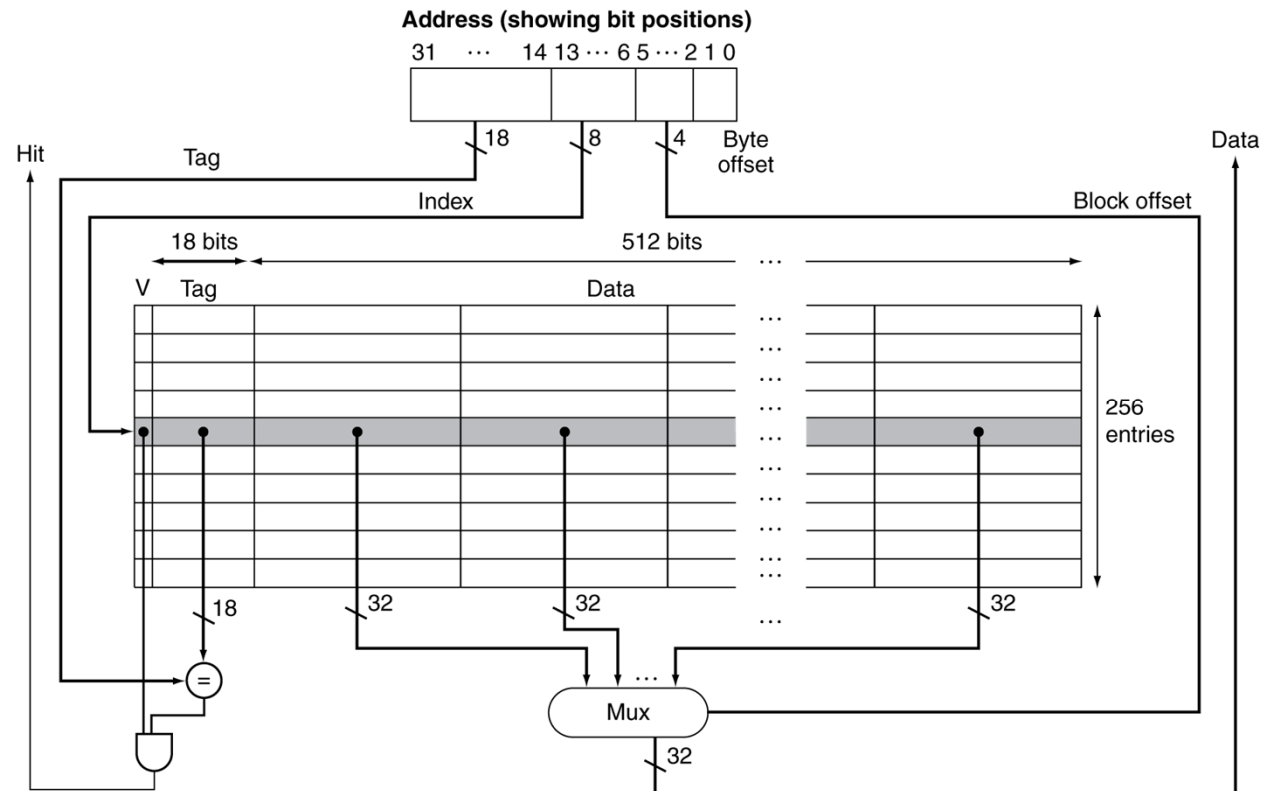


Block被替换时，也可以使用write-buffer，减少miss-penalty时间。方法是先将原block数据搬到write-buffer中，紧接着从主存储器读数据到cache，最后把write-buffer中数据写回主存

即使处理器产生“write”速度高于主存储器(DRAM)接收速度，write-back方法也表现出优越的性能，但其硬件实现要比write-through复杂和困难

### 5.2.3 改善Cache性能--一个Cache 组织结构实例：FastMATH Processor

- 16 KB
- 16 words/block
- 256 blocks



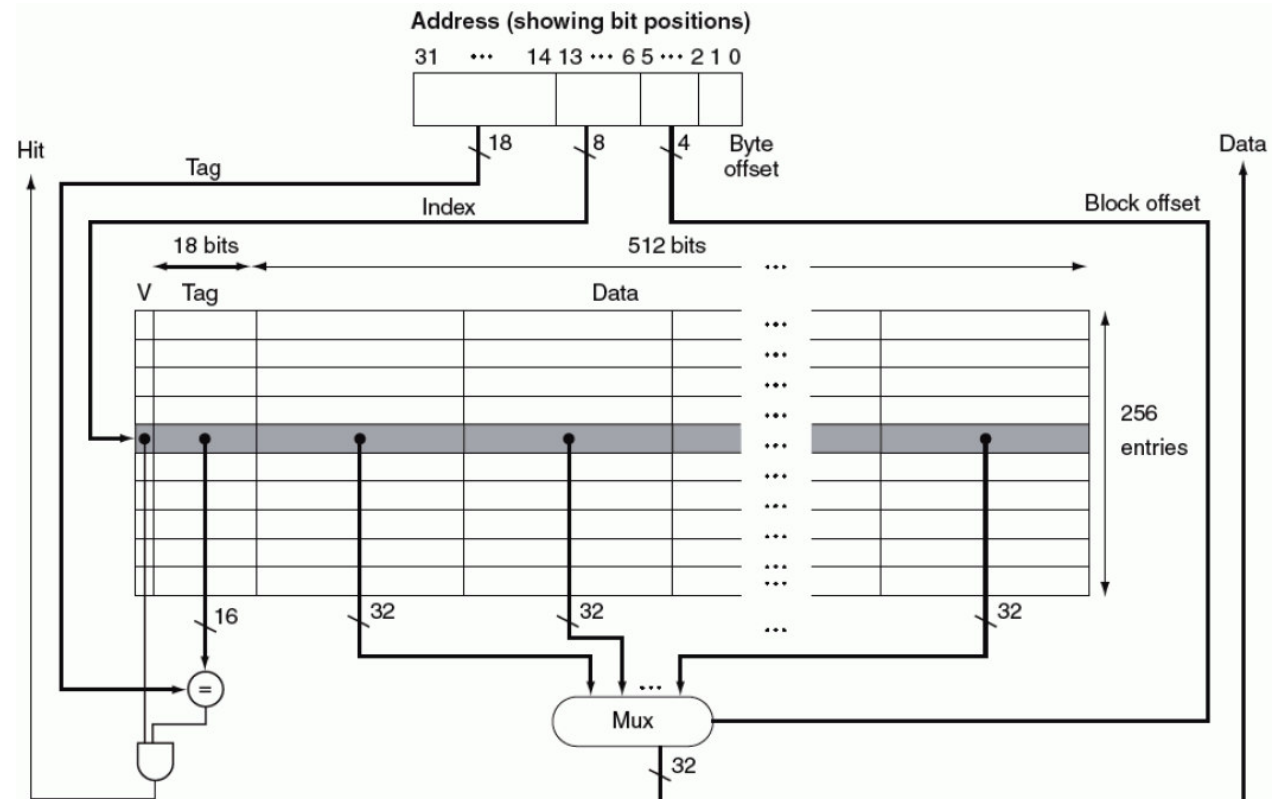
- 实际实现时，每个cache又细分为大RAM和小RAM，大RAM存放数据，小RAM存放tag  
->避免了使用大的数据选择器



## 5.2.3 改善Cache性能--一个Cache 组织结构实例：FastMATH Processor

### 读写Cache的流程：

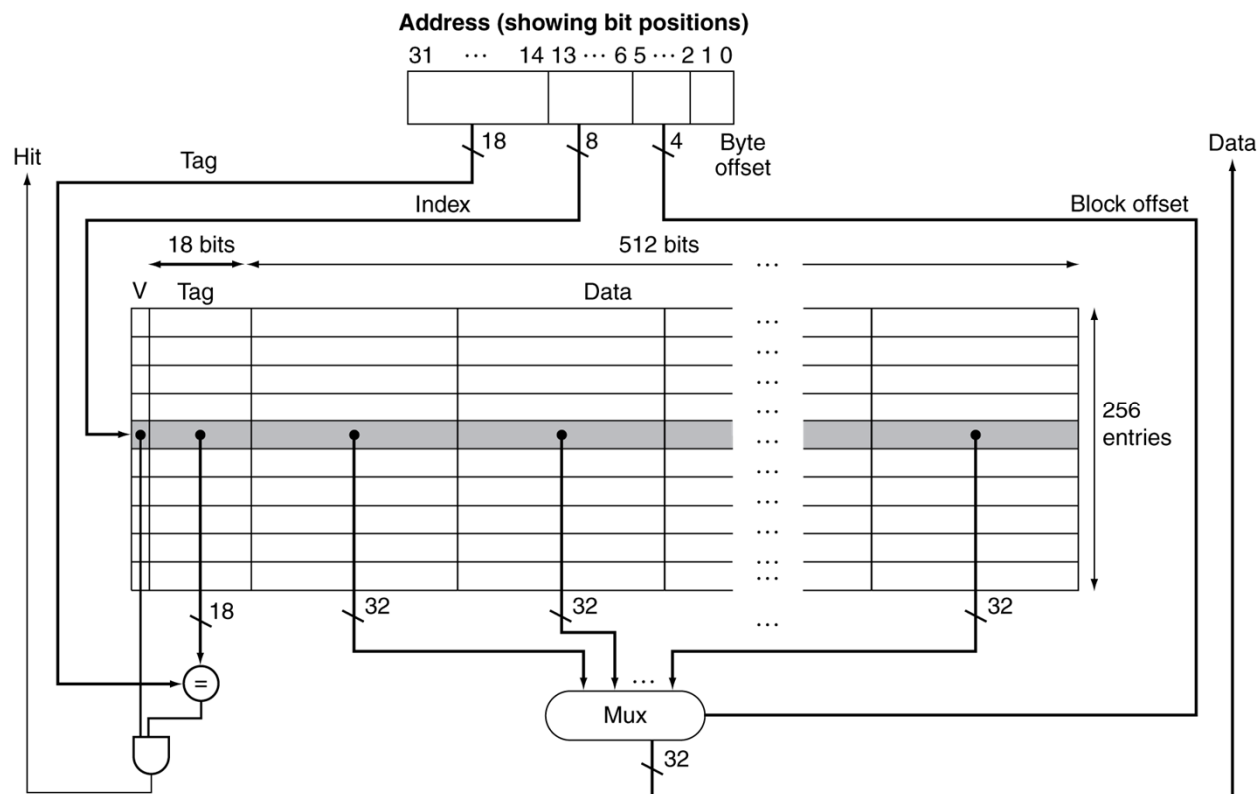
1. 将地址送给Cache；这个地址或来自PC（指令）或来自ALU（数据地址）
2. 若cache hit（命中），说明所请求的块在cache中，用block offset 选定请求的字
3. 若cache miss（缺失），发送地址给主存储器(DRAM)，一旦主存储器(DRAM) 返回数据，就会写入数据或读出数据



- 操作系统决定采用write-through或write-back
- FastMATH Processor 提供了一个block 大小的write-buffer

## 5.2.3 改善Cache性能--一个Cache 组织结构实例：FastMATH Processor

- 为了避免结构冒险，采用分离的数据Cache和指令Cache



-> 分离式Cache和混合式Cache各有哪些优缺点?

---

### 5.2.3 改善Cache性能--实例运行SPEC2000的缺失率

---

- 混合式Cache不对数据和指令进行区分，所以比分离式Cache具有稍好的命中率
- 分离式Cache比混合式Cache具有更高的带宽，且更有利于流水线

分别对FastMATH 采用混合式cache 和分离式cache，测试情况：

- Total size: 32 KB
  - Split cache miss rate: 3.24%
  - Combined cache miss rate: 3.18%
- 分离式cache 的缺失率稍微比混合式cache高

但分离式cache的带宽，很容易弥补缺失率的落差

**-> 不能把缺失率作为cache性能评价的唯一指标，衡量Cache性能的最终指标是存储器系统对程序执行时间的影响**

---

# 目录

---

## 5.1 高速缓存存储器

### 5.1.1 存储器组织结构

### 5.2.2 Cache的基本结构

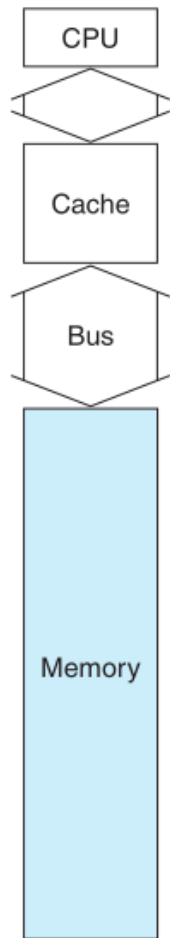
### 5.2.3 改善Cache性能

### 5.2.4 改善DRAM性能

---

## 5.2.4 改善DRAM性能--存储器带宽对缺失时间的影响

---

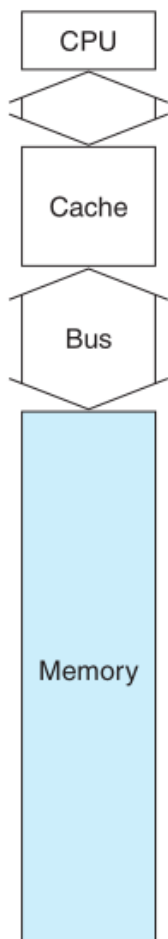


The processor is typically connected to memory over a bus. The clock rate of the bus is usually much slower than the processor, by as much as a factor of 10. The speed of this bus affects the miss penalty.

**假设:**

- 发送一个地址需要1个存储器总线访问周期
- 每次访问DRAM需要15个存储器总线访问周期
- 发送一个word需要1存储器总线访问周期
- 每个block有4个word

## 5.2.4 改善DRAM性能--带宽为1个word的情况



假设：发送一个地址需要1个存储器总线访问周期，每次访问DRAM需要15个存储器总线访问周期，发送一个word需要1存储器总线访问周期，每个block有4个word

左图中主存储器的带宽为1个word，因而一次Miss Penalty时间为：

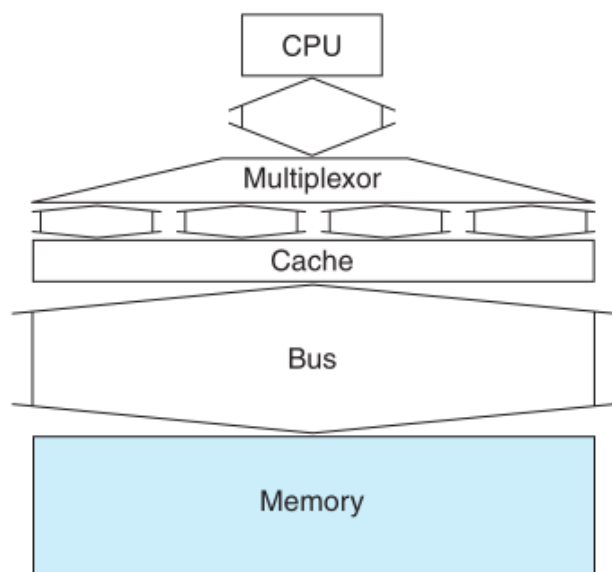
$$1 + 4 \times 15 + 4 \times 1 = 65 \text{ 个存储器总线访问周期}$$

发生缺失时，每个周期传送的字节数为：

$$(4 \times 4) / 65 = 0.25 \text{ 字节/周期}$$

## 5.2.4 改善DRAM性能--带宽为4个word的情况

假设：发送一个地址需要1个存储器总线访问周期，每次访问DRAM需要15个存储器总线访问周期，发送一个word需要1存储器总线访问周期，每个block有4个word



b. Wide memory organization

左图中主存储器的带宽为4个word，因而一次Miss Penalty 时间为：

$$1 + 15 + 1 = 17 \text{ 个存储器总线访问周期}$$

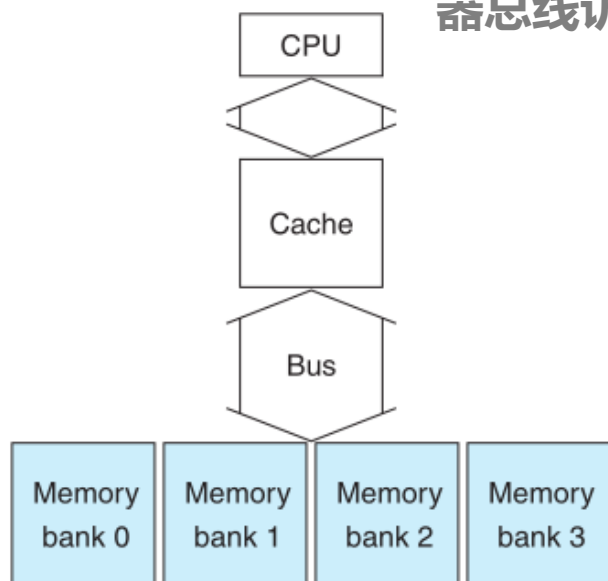
发生缺失时，每个周期传送的字节数为：

$$(4 \times 4) / 17 = 0.94 \text{ 字节/周期}$$

缺点：总线太宽、Cache速度降低(多路选择器和逻辑控制)

## 5.2.4 改善DRAM性能--以Bank方式组织存储器结构，一次启动连续读或写多个字

假设：发送一个地址需要1个存储器总线访问周期，每次访问DRAM需要15个存储器总线访问周期，发送一个word需要1存储器总线访问周期，每个block有4个word



c. Interleaved memory organization

左图4个bank需要一个地址，允许存储器连续地读出或写入4个bank，因此一次Miss Penalty时间为：

$$1 + 15 + 4 \times 1 = 20 \text{ 个存储器总线访问周期}$$

发生缺失时，每个周期传送的字节数为：

$$(4 \times 4) / 20 = 0.80 \text{ 字节/周期}$$



## 5.2.4 改善DRAM性能-- DRAM技术发展历程

DRAM诞生：1968年，IBM申请DRAM(Dynamic Random Access- Memory)专利；1970年，Intel研发第一款商用DRAM，大小为1KB；

“远古阶段”：1968~1992，DRAM的大小由1KB发展到16MB，速度基本在50M以下（486的主频也只有33M）；

“第一次DRAM革命”：1993年，三星引入**SDRAM**技术，频率飙升到133M

“第二次DRAM革命”：1996年，三星提出**DDR SDRAM**

(核心频率133M、总线频率133M、数据速率266M)

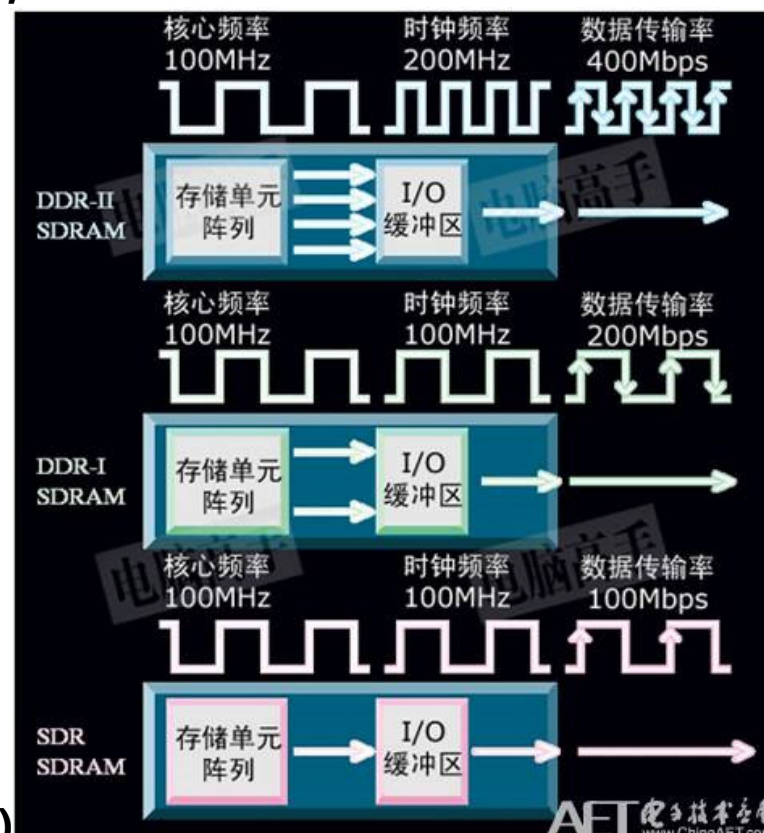
新世纪的DRAM：

2001，DDR2 (核心133M、总线266M、数据533M)

2005，DDR3 (核心133M、总线533M、数据1066M)

2011，DDR4 (核心266M、总线1066M、数据2166M)

上述核心频率可变，但变化不会太大，DDR4的核心频率范围：200~400M



---

## 小结

---

### 内容:

5.1.1 存储器组织结构

5.2.2 Cache的基本结构

5.2.3 改善Cache性能

5.2.4 改善DRAM性能

写Cache的处理方法(写直达、写绕回、写回)  
缺失率和缺失时间的改善措施

### 重点:

1. 尽管增加block的容量可以降低缺失率，但会导致miss penalty (block替换时间) 时间变长
2. 增加带宽可以降低miss penalty