第五章

大容量和高速度：开发存储器层次结构

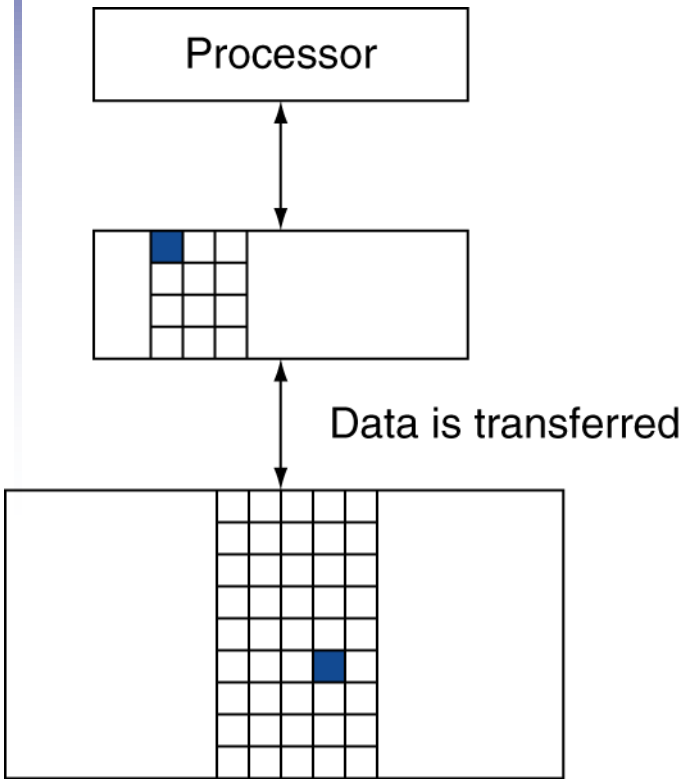
局部性原理

- 在任何时间内，程序访问的只是地址空间相对较小的一部分内容
- 时间局部性 Temporal locality
  - 如果某个数据项被访问，在不久的将来它可能再次被访问
- 空间局部性 Spatial locality
  - 如果某个数据项被访问，与它地址相邻的数据项可能很快也将被访问

利用局部性原理

- 存储器层次结构 Memory hierarchy
- 将全部数据存在磁盘上
- 将最近访问的（和邻近的）数据
  - 从磁盘拷贝到DRAM主存
  - 从DRAM主存拷贝到SRAM缓存
  - 从SRAM缓存拷贝到CPU

存储器层次结构



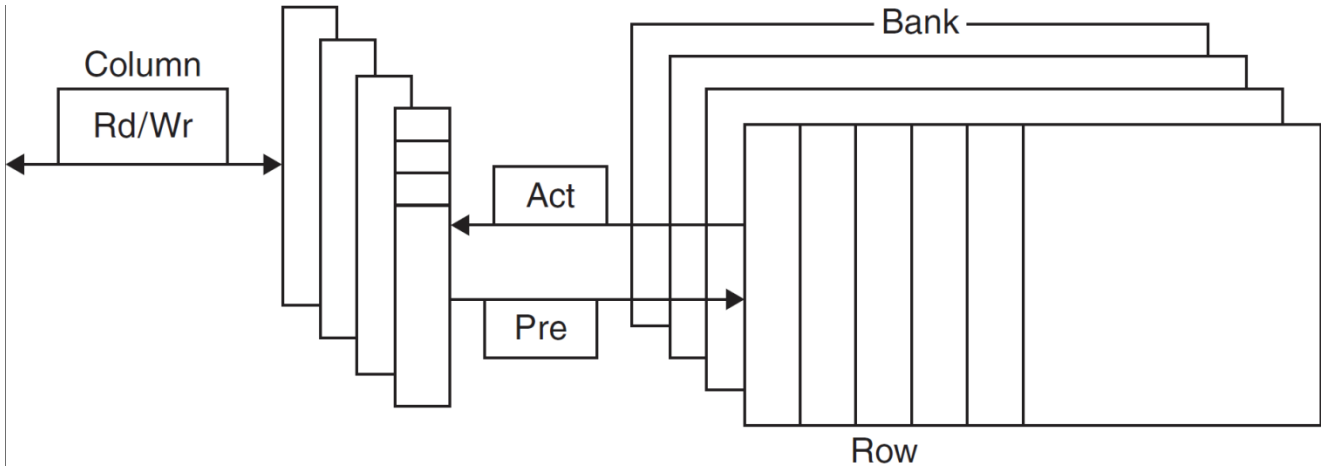
- 块 Block (行line): 拷贝的最小单元
- 如果CPU需要的数据存放在更高一级的存储器里
  - 命中 Hit:
    - 命中率 Hit ratio: hits/accesses
- 如果不在更高一级的存储器里
  - 缺失 Miss: 将相应的数据块从低层存储器拷贝到高一级的存储器里
    - 缺失率: misses/accesses
      = 1 – hit ratio
- 命中时间/缺失代价

存储器技术

- 缓存 Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- 主存 Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- 磁盘
  - 5ms – 20ms, $0.20 – $2 per GB
- 理想存储器 Ideal memory
  - 访问时间近似于SRAM
  - 存储容量和价格近似于磁盘

DRAM

- 在电容上保存电荷
  - 使用单个晶体管对电容进行访问
  - 必须周期性的刷新
    - 读出-写回
    - 两级译码结构-读周期/写周期
    - 一次刷新一整行 “row”

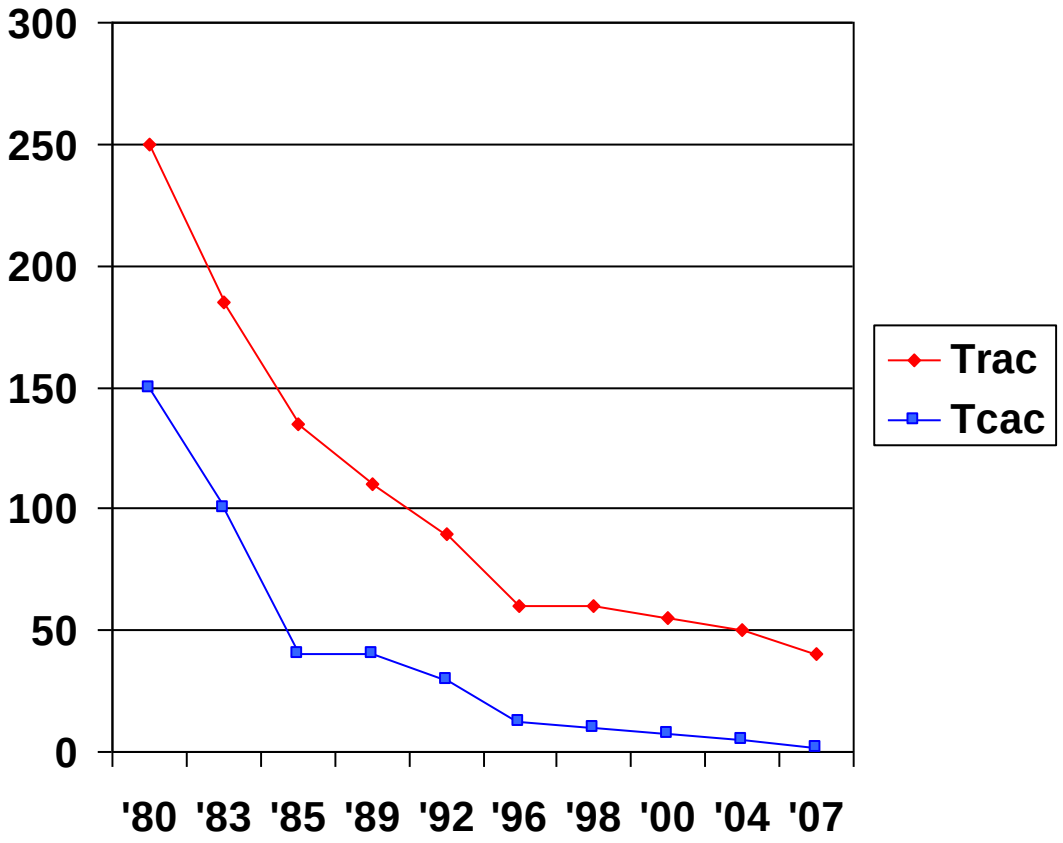


DRAM组织结构

- DRAM中的位被组织成一个矩阵
  - DRAM每次访问一整行 （row）
- 双倍数据率Double data rate (DDR) DRAM
  - 时钟的上升沿和下降沿都要传送数据
- 四倍数据率Quad data rate (QDR) DRAM
  - 将 DDR 输入和输出分离

DRAM 演化

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



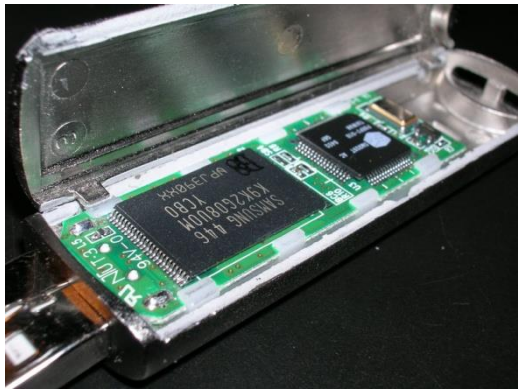
Trac：访问新的一行/一列的时间
Tcac：访问已经在CACHE的行中一列的时间

DRAM性能优化

- 行缓冲 Row buffer
  - 允许多个字并行地读和刷新
- 同步DRAM (Synchronous DRAM)
  - 允许以突发模式，连续访问数据，而不需要每次发送地址
  - 提高带宽
- 地址交叉
  - DRAM内部可以组织成对多个BANK进行读或写操作
  - 每个BANK有自己的行缓冲器
  - 向不同的BANK发送一个地址可以允许同时对他们进行读或写操作

闪存

- 非易失型半导体存储器
  - 比磁盘快100× – 1000×
  - 体积小、低功耗、抗震动
  - 价格 $/GB 介于磁盘和 DRAM之间

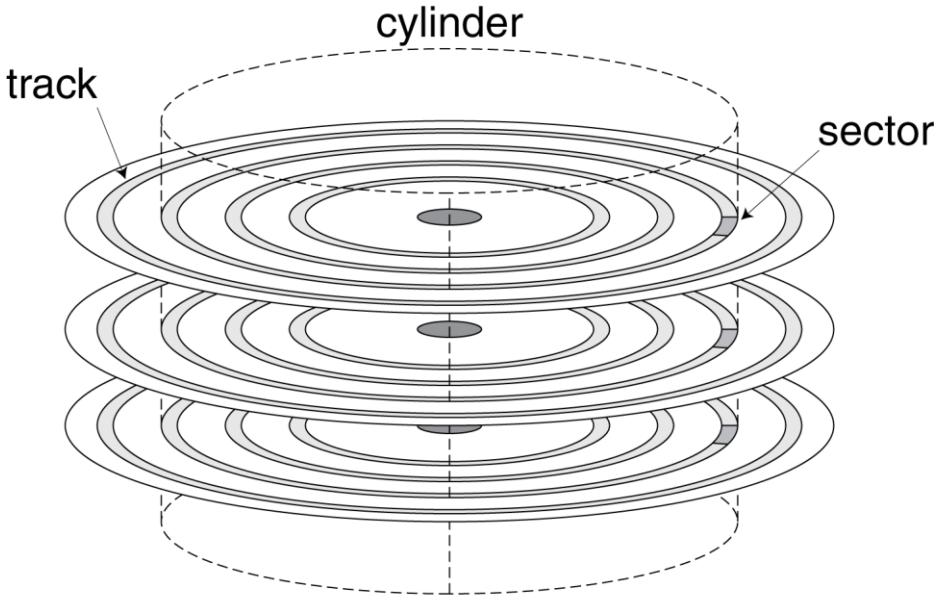


闪存简介

- NOR型闪存: bit cell like a NOR gate
  - 随机读、写访问
  - 用于嵌入式系统中的指令存储器
- NAND型闪存: bit cell like a NAND gate
  - 存储密度高 (bits/area), 价格低
  - 读写单位：物理页
  - 擦除单位：物理块
  - 用于U盘、SD卡、SSD固态硬盘
- 闪存10000+次使用后会失效
  - 不可以直接替换DRAM或者磁盘
  - 需要损耗平衡 Wear leveling: 重新映射数据到使用较少的物理块

硬盘

- 非易失型转动电磁存储介质



磁盘存储器的访问

- 扇区
  - 扇区 ID
  - 数据 (默认512 bytes, 推荐4096 bytes)
  - 纠错码 Error correcting code (ECC)
    - 用于隐藏制作工艺缺陷或记录错误
- 扇区的访问
  - 排队等待时间
  - 寻道时间 Seek
  - 旋转延迟 Rotational latency
  - 数据传输 Data transfer
  - 控制器的额外开销 Controller overhead

【例】磁盘访问

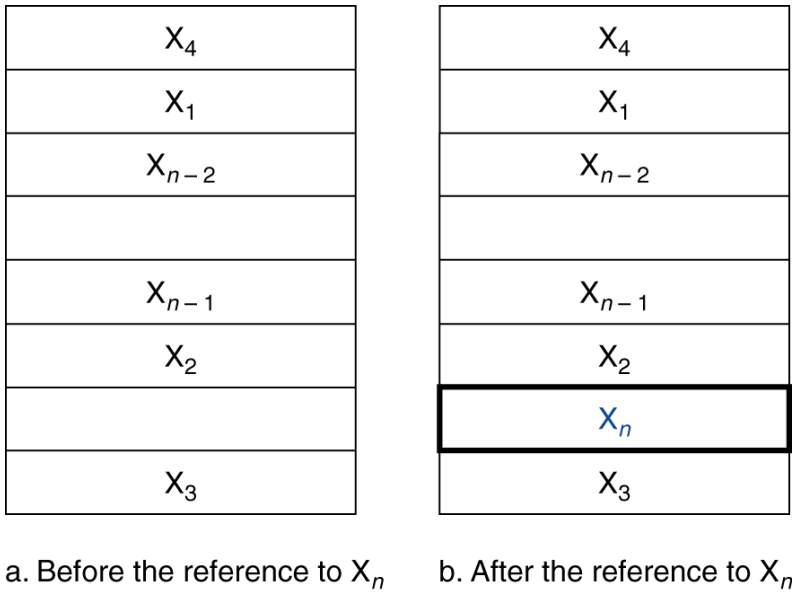
- 假设
  - 扇区大小512B, 转速15,000rpm, 平均寻道时间4ms, 数据传输率100MB/s , 控制器延时0.2ms
- 平均读时间
  - 4ms 寻道时间
+ ½ / (15,000/60) = 2ms 旋转延迟
+ 512 / 100MB/s = 0.005ms 传输时间
+ 0.2ms 控制器延时
= 6.2ms
- 假设平均寻道时间 1ms
  - 则平均读时间为 3.2ms

磁盘性能优化

- 制造商所声称的平均寻道时间
  - 对所有可能的寻道时间取平均值
  - 局部性和OS调度都会导致更短的平均寻道时间
- 智能硬盘控制器分配磁盘扇区的使用
  - 为上层用户提供逻辑扇区接口
  - SCSI, ATA, SATA
- 磁盘包含了缓存
  - 根据预测，预取部分扇区至DISK缓存
  - 避免寻道延迟和旋转延迟

高速缓存 Cache

- 距离CPU最近一级的存储层次
- 给定访问 $X_1, \ldots, X_{n-1}, X_n$



a. Before the reference to X_n b. After the reference to X_n

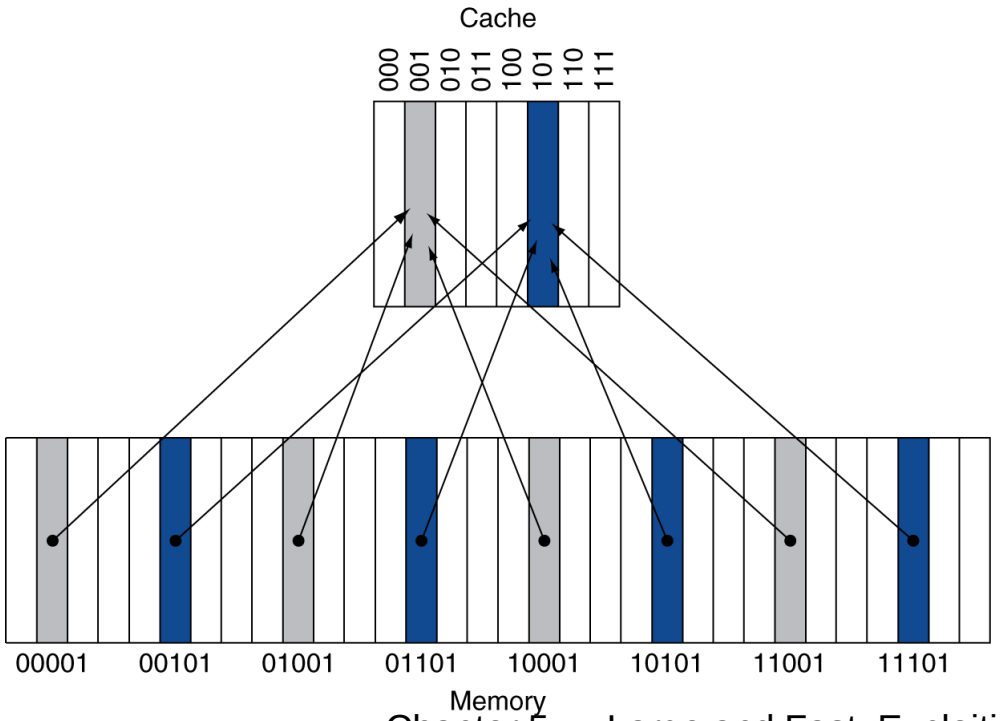
- 如何知道数据在不在?
- 如何找到数据?

直接映射

- 每个存储器地址仅仅对应到CACHE中的唯一的一个位置：

  块号(Index) = (块地址) mod (cache中的块数)

  - cache块数为2^n时，可以直接用内存地址的低位表示



辨识：
Cache块号
主存块地址

区/区号

标记和有效位

- 如何知道某个cache位置上记录的是哪块（内存）数据?
  - 存储标记tag （对于上图可以用地址高两位）
- 如何保证是否包含有效信息？
  - 有效位Valid bit: 1 = 在, 0 = 不在
  - 初始值为 0

【例】Cache

- 容量8块, 1 字/块, 直接映射
- 初始状态

字地址(十进制)：22，26，22，16，3，16，18，16

Index 索引	V 有效位	Tag 标记	Data 数据
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

【例】Cache

字地址(十进制)：22，26，22，16，3，16，18，16

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

【例】Cache

字地址(十进制)：22，26，22，16，3，16，18，16

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

【例】Cache

字地址(十进制)：22，26，22，16，3，16，18，16

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
16	10 000	Miss	000

Index	V	Tag	Data
000	N	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

【例】Cache

字地址(十进制)：22，26，22，16，3，16，18，16

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

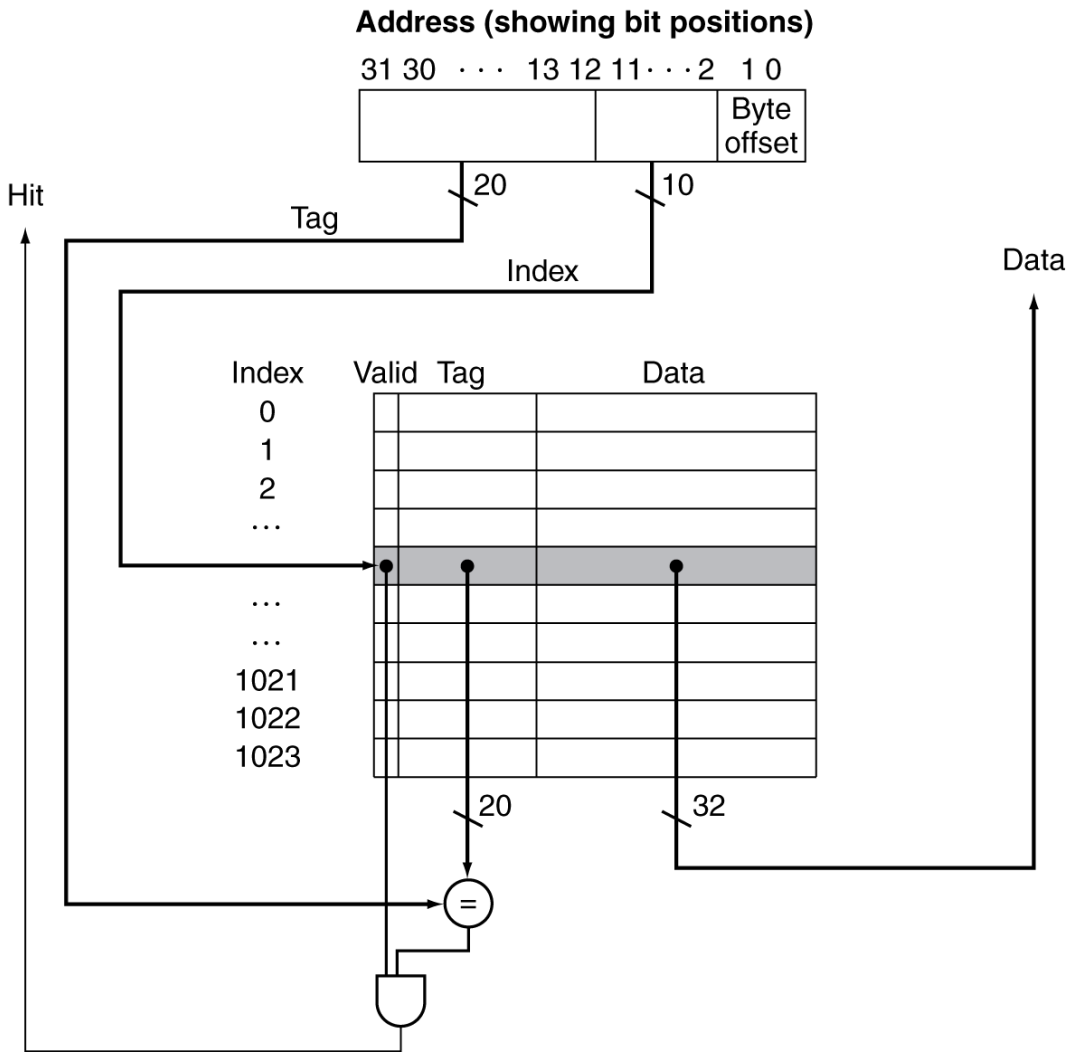
【例】Cache

字地址(十进制)：22，26，22，16，3，16，18，16

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

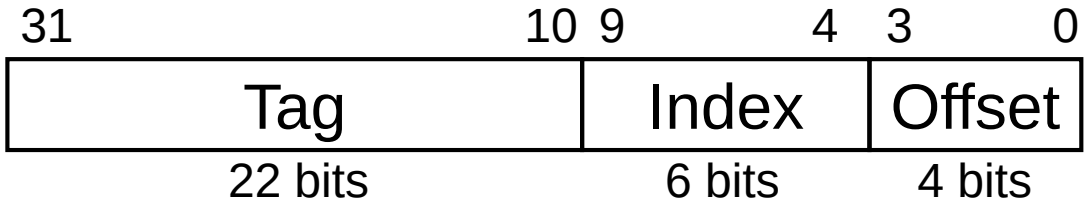
地址划分


Address (showing bit positions)
31 30 · · · 13 12 11 · · · 2 1 0
Byte offset
20
10
Hit
Tag
Index
Data
Index
Valid
Tag
Data
0
1
2
· · ·
· · ·
· · ·
1021
1022
1023
20
32
=

【例】将一个地址映射到多字大小的cache块中

- 一个cache有64个块, 每块16字节。则字节地址1200将被映射到cache中的哪一块？

- 块地址 = ⌊1200/16⌋ = 75
- Cache中的块号 = 75 modulo 64 = 11



Cache缺失处理

- Cache 命中，则CPU正常处理
- Cache 缺失
  - 造成CPU流水线阻塞
  - 从下一级存储器中取数据块
  - 指令cache 缺失
    - 重新进行指令获取
  - 数据cache 缺失
    - 完成数据的访问

写直达 Write-Through

- 当数据写命中时，只更新cache中的数据块
  - 但是会导致cache和主存的不一致
- 写直达：同时更新cache和主存
- 但是，会令写操作时间变长
  - e.g., 如果基本CPI = 1, 10%的指令是存储，写入主存需要 100个指令周期
    - 有效 CPI = 1 + 0.1×100 = 11
- 解决方案: 写缓冲 write buffer
  - 将等待写入主存的数据暂存在写缓冲内
  - CPU可以立即继续执行
    - 仅当写缓冲已经满时，写操作必须停下来，直到写缓存中有一个空位置

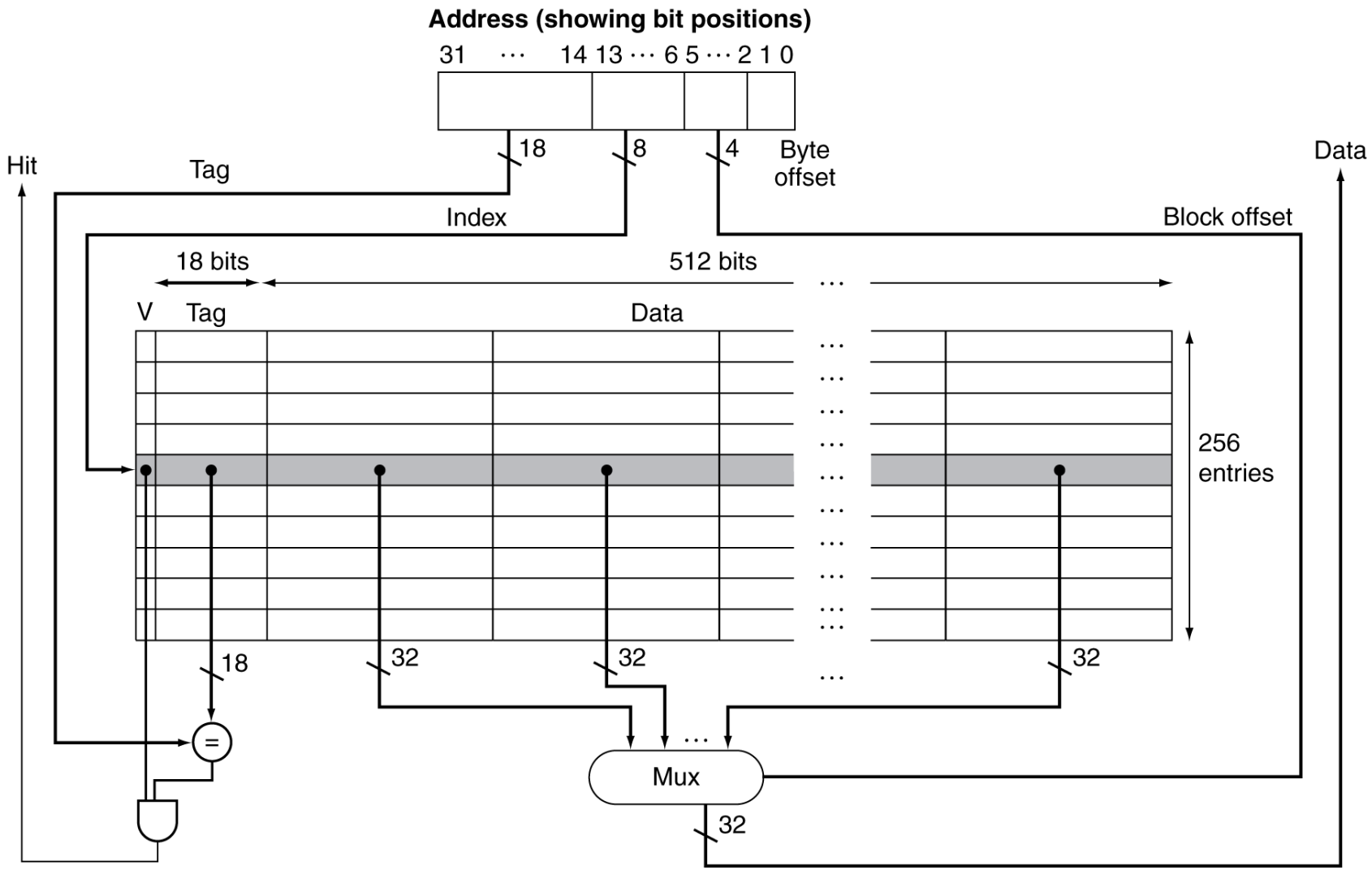
写回 Write-Back

- **写回: 仅更新cache中的数据块，修改过的块被替换时才写较低层次的存储结构。**
  - 必须记录每个数据块是否被修改过（脏 dirty）
- 当一个被修改过的块被替换时
  - 将被替换的块写回至主存
  - 可以利用写缓冲，来优先安排读出被更新的数据块

内置 FastMATH 处理器

- 嵌入式 MIPS 处理器
  - 12级流水线结构
  - 每个时钟周期可以请求一个指令字和一个数据字
- 分离 I-cache 和 D-cache
  - 每个 16KB: 256 blocks × 16 words/block
  - D-cache: 写直达 或 写回
- SPEC2000 缺失率
  - I-cache: 0.4%
  - D-cache: 11.4%
  - 综合缺失率: 3.2%

内置 FastMATH处理器



Cache性能的评估和改进

- CPU时间的组成
  - 程序执行时钟周期数
    - 包括 cache 命中的时间
  - 存储器阻塞的时钟周期数
    - 主要由于 cache 缺失
- 简化假设（写直达的读、写操作）：

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

【例】计算Cache性能

- 假设
  - I-cache 失效率 = 2%
  - D-cache 失效率 = 4%
  - 缺失代价 = 100 个时钟周期
  - 理想 CPI = 2
  - Load & stores 占全部36%的指令
- 指令缺失的时钟周期数
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- 实际 CPI = 2 + 2 + 1.44 = 5.44
  - 理想 CPU 是 5.44/2 =2.72 倍

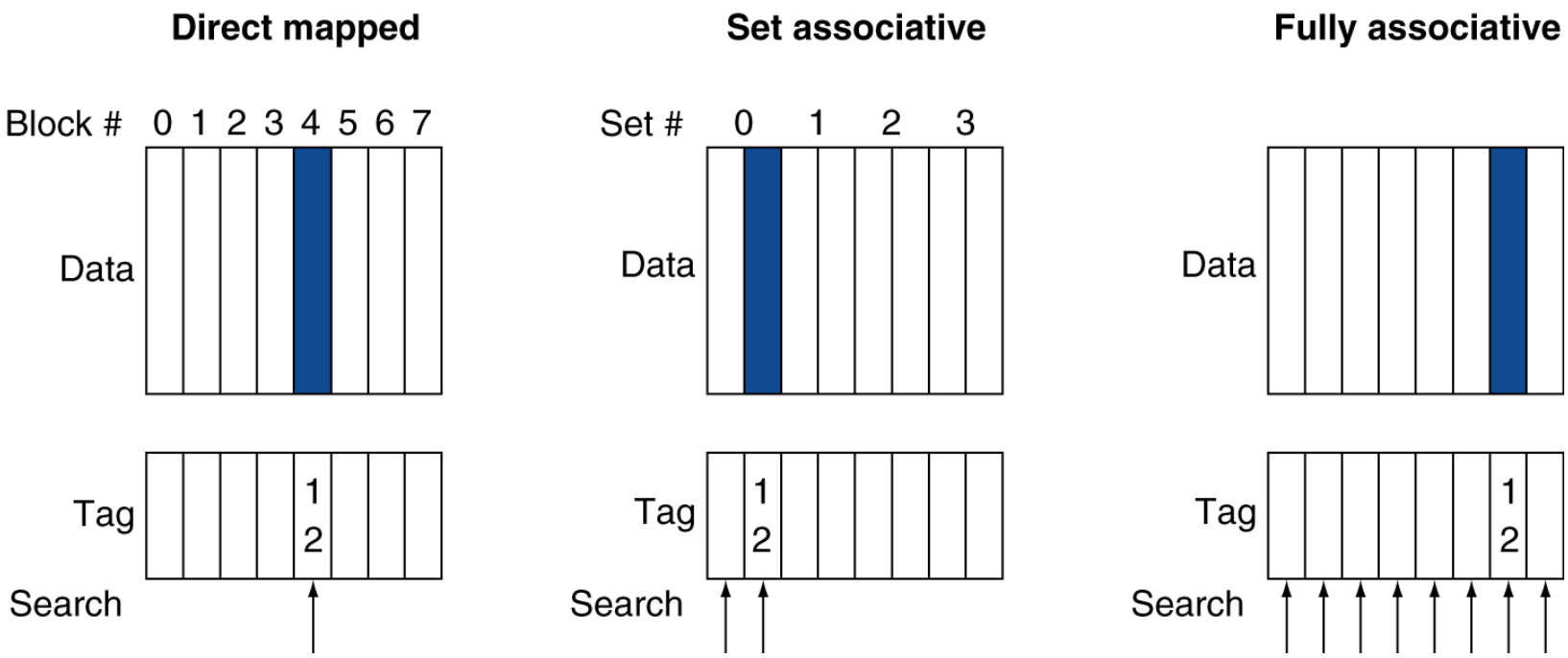
平均访问时间

- 命中时间也对性能也有重要影响
- 平均存储器访问时间Average memory access time (AMAT)
  - AMAT = 命中时间+ 缺失率 × 缺失代价
- 例子
  - CPU时钟周期时间为1ns, 命中时间为1个时钟周期, 缺失代价为20 时钟周期, I-cache的缺失率为 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 即每个指令需要2个时钟周期

Cache相联

- 全相联
  - 一个块可以被放置在cache中的任何位置
  - 需要检索cache中的所有项
  - 每个cache项都需要比较器 (开销大)
- 多路组相联
  - 每个组（set）包含 *n* 块
  - 块号决定了组号
    - (块号) modulo (cache中的组数)
  - 可能需要查找组中所有块
  - 只需要*n* 个比较器 (相比于直接映射减小了开销)

Cache相联


Direct mapped
Block # 0 1 2 3 4 5 6 7
Data
Tag
1
2
Search
Set associative
Set # 0 1 2 3
Data
Tag
1
2
Search
Fully associative
Data
Tag
1
2
Search

相联度

- 一个拥有8个块的cache被配置成直接映射、两路组相联、四路组相联、八路组相联的结构

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

三种Cache映射比较

- Cache包含4块，1字/块
  - 直接映射，2路组相联，全相联
  - 块访问顺序: 0, 8, 0, 6, 8

三种Cache映射比较

- 直接映射　　块访问顺序: 0, 8, 0, 6, 8
  - 块号（index）

0 / 4 = 0, 6 / 4 = 2, 8 / 4 = 0

块地址	块号
0	0/4=0
6	6/4=2
8	8/4=0

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

三种Cache映射比较

- 2路组相联　　块访问顺序: 0, 8, 0, 6, 8
  - 组数=块数/2=2
  - 组号（index）= 块地址 mod组数

0 / 2 = 0, 6 / 2 = 0, 8 / 2 = 0

替换规则：最近最少使用的块

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

三种Cache映射比较

- 全相联　　块访问顺序: 0, 8, 0, 6, 8

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

三种Cache映射比较

- Cache包含8块，1字/块
  - 块访问顺序: 0, 8, 0, 6, 8
  - 直接映射，2路组相联，全相联
  - 缺失次数？
- Cache包含16块
  - 缺失次数？

三种Cache映射比较

- Cache包含8块，2字/块
  - 字地址：

1, 134, 212, 1, 135, 162, 2, 161

  - 直接映射，2路组相联，全相联
  - 缺失次数？

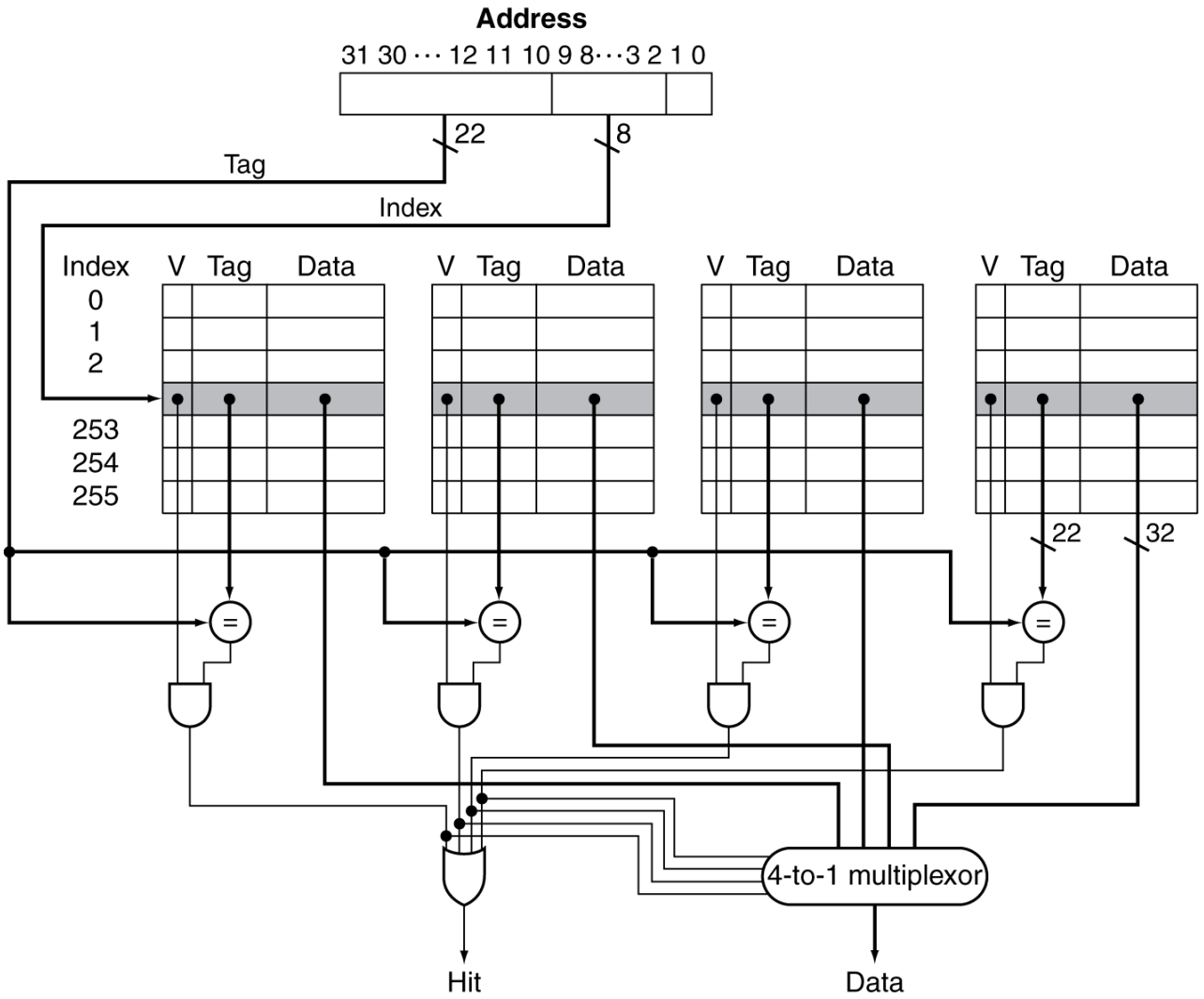
相联度与缺失率的关系

- 增加相联度可以降低缺失率
  - 但是收益是逐级递减的

- 系统仿真

  64KB D-cache, 16-word blocks, SPEC2000
  - 1路: 10.3%
  - 2路: 8.6%
  - 4路: 8.3%
  - 8路: 8.1%

多路组相联Cache结构



替换策略

- 直接映射: 别无选择，想选也没得选
- 组相联
  - 倾向选择储存非有效信息（non-valid）的块
  - 如果没有non-valid块，选择组中的一块
- 最近最少使用 Least-recently used (LRU)
  - 选择最长未被使用的
    - 2路组相联相对容易实现，4路组相联仍可控制，超过4路很难控制
- 随机替换
  - 对于相联度较高的情况，和LRU获得近似相同的结果

多级 Cache

- L1-cache 直接连接 CPU
  - 容量小，但速度快
- L-2 cache 处理L1-cache 的失效
  - 容量较大，速度较慢，但仍比主存要快
- 主存用于处理L2-cache 的失效
- 一些高端系统包含 L-3 cache

多级 Cache 举例

- 假设
  - CPU 基准 CPI = 1, 时钟频率 = 4GHz
  - 缺失率/指令 = 2%
  - 主存访问时间 = 100ns
- 仅有L1-cache
  - 缺失代价 = 100ns/0.25ns = 400 时钟周期
  - 有效 CPI = 1 + 0.02 × 400 = 9

P277

多级Cache举例 (cont.)

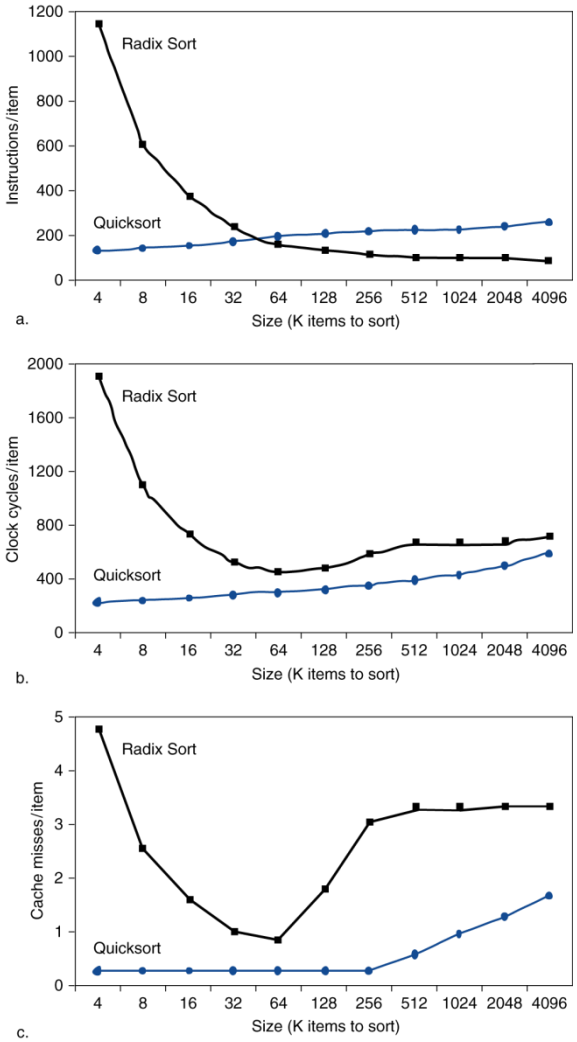
- 如果增加 L2-cache
  - 访问时间 = 5ns
  - 对于主存的整体失效率 = 0.5%
- L1失效& L2 命中
  - 缺失代价 = 5ns/0.25ns = 20 时钟周期
- L1失效& L2 失效
  - 额外缺失代价 = 500 时钟周期
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- 性能比 = 9/3.4 = 2.6

多级 Cache 的考虑因素

- L1-cache
  - 集中在最小化命中时间
- L2-cache
  - 集中在低的缺失率，以避免主存的访问
  - 命中时间对整体性能影响较小
- 结果
  - L1-cache通常较独立cache的容量小
  - L1-cache的块大小通常小于L2-cache的块大小

与软件交互

- 缺失与内存访问模式有关
  - 算法的性能
  - 面向内存访问的编译器优化



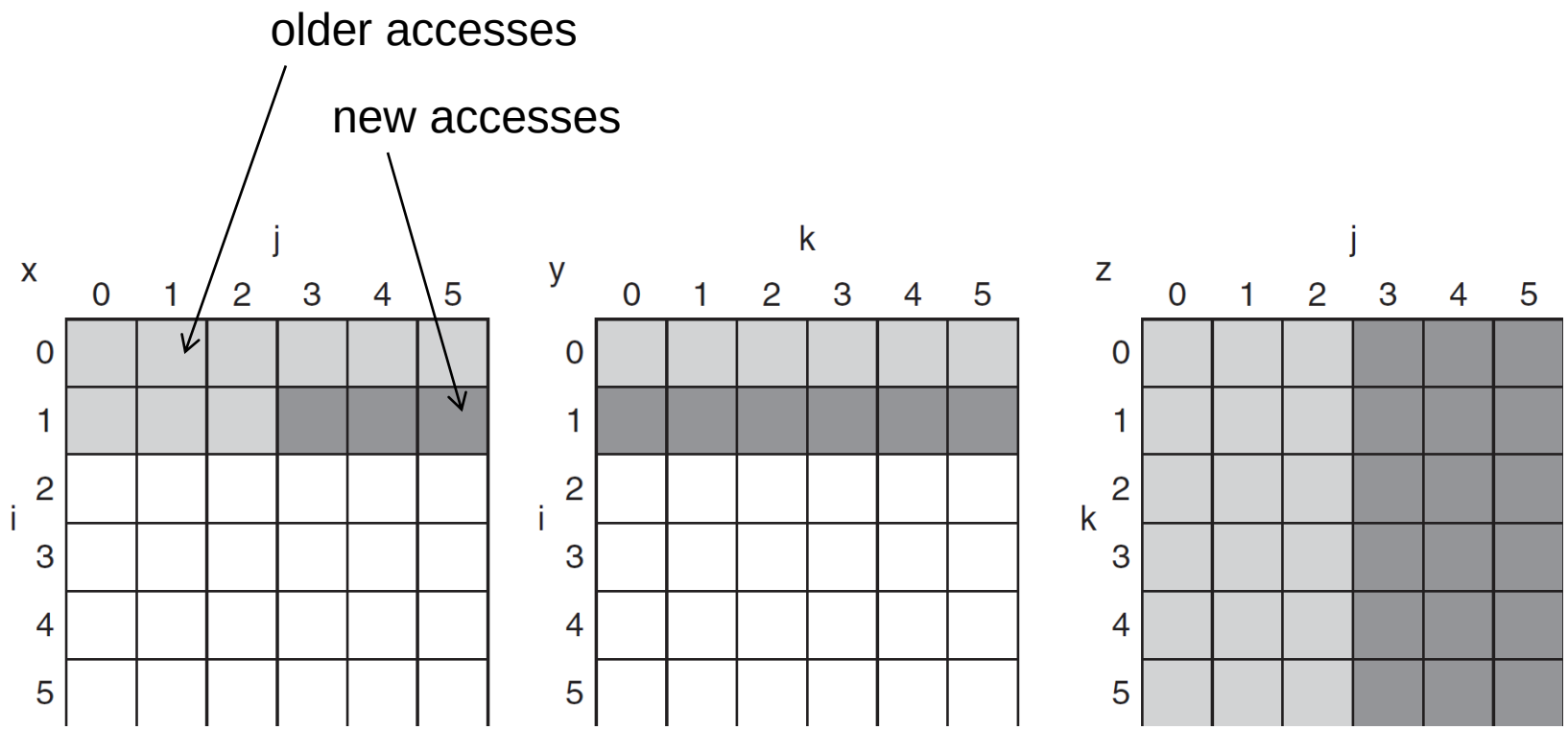
通过分块进行软件优化

- 目标：通过对cache中的数据进行重用，提升数据的时间局部性并因此降低缺失率
- 例如 DGEMM 的内循环:

```
for (int j = 0; j < n; ++j)
{
  double cij = C[i+j*n];
  for( int k = 0; k < n; k++ )
    cij += A[i+k*n] * B[k+j*n];
  C[i+j*n] = cij;
}
```

DGEMM 访问模式

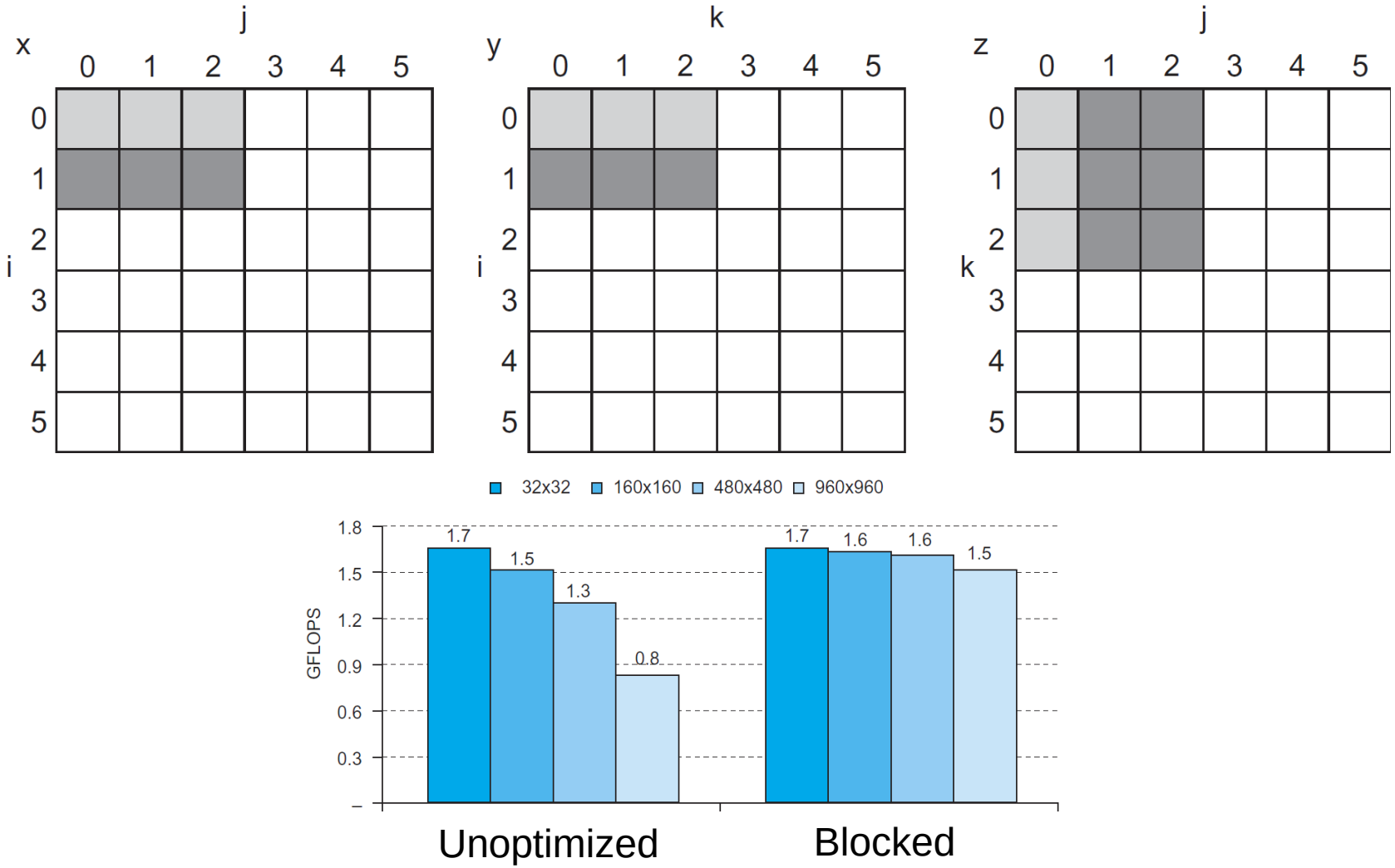
- 三个数组C, A, B的访问情况



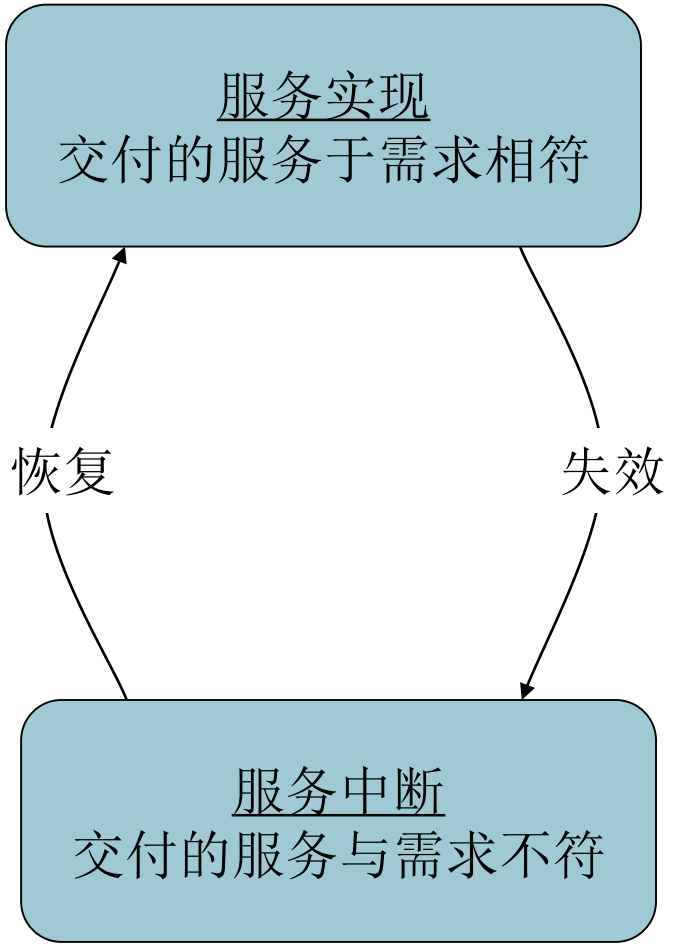
DGEMM的cache分块版本

```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5  for (int i = si; i < si+BLOCKSIZE; ++i)
6   for (int j = sj; j < sj+BLOCKSIZE; ++j)
7   {
8    double cij = C[i+j*n];/* cij = C[i][j] */
9    for( int k = sk; k < sk+BLOCKSIZE; k++ )
10    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11   C[i+j*n] = cij;/* C[i][j] = cij */
12  }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17   for ( int si = 0; si < n; si += BLOCKSIZE )
18    for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19     do_block(n, si, sj, sk, A, B, C);
20 }
```

分块后 DGEMM 的访问模式



可信存储器层次



- 失效: 一个组成器件发生的错误
  - 可能导致整个系统的错误

可信存储的度量标准

- 可靠性: 平均无故障时间 mean time to failure (MTTF)
- 服务中断: 平均维修时间 mean time to repair (MTTR)
- 失效间隔平均时间 Mean time between failures
  - MTBF = MTTF + MTTR
- 可用性 = MTTF / (MTTF + MTTR)
- 提高可用性的手段
  - 提高MTTF: 故障避免、故障容忍、故障预测
  - 减少MTTR: 改良工具、诊断处理过程、修复过程

一位纠错两位检错汉明编码SEC/DED

- 汉明距离
  - 两个等长二进制数的汉明距离是两个数对应位置不同的位的数量
- 最小距离 = 2 提供了一位纠错
  - E.g. 奇偶校验码 parity code
- 最小距离 = 3 提供了一位纠错、两位检错

SEC编码

- 计算汉明码:
  - 对数据部分从左到右开始依次编号
  - 将所有编号为2的整数次幂的位标记为奇偶校验位（1,2,4,8,16）
  - 其他剩余位置用作数据位
  - 奇偶校验位的位置决定了其对应的数据位

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

SEC解码

- 校验位的值指出错误位
  - E.g.
    - 校验位 Parity bits = 0000 说明没有错误
    - 校验位 Parity bits = 1010 说明 bit 10 发生翻转

SEC/DEC码

- 增加一位的奇偶校验码，对整个字 (p_n)进行计算校验
- 最小汉明距离变为 4
- 解码过程:
  - 当 H = SEC 的校验位
    - H 为偶数, p_n 为偶数, 没有错误发生
    - H 为奇数, p_n 为奇数, 表明出现了一位可纠正错误
    - H 为偶数, p_n 为奇数, p_n 位错误
    - H 为奇数, p_n 为偶数, 表明出现了两位错误
- Note: DRAM 用每64位做 8位SEC/DEC的ECC来保护数据

虚拟机

- 主机模拟客户操作系统和机器资源
  - 提升了多个用户之间的隔离
  - 规避了安全和可靠性的问题
  - 有助于资源共享
- 虚拟化对性能有影响
  - 适合于现代高性能计算机
- 举例
  - IBM VM/370 (70年代的技术!)
  - VMWare
  - Microsoft Virtual PC

虚拟机监视器

- 虚拟资源映射到物理资源
  - 内存, I/O设备, CPUs
- 客户端代码在用户模式运行在本机
  - Traps to VMM on privileged instructions and access to protected resources
- 客户端 OS 可能不同于主机 OS
- VMM 管理实际的 I/O 设备
  - 为客户端模拟通用的虚拟 I/O 设备

举例: 时钟的虚拟化

- 本机发生时钟中断
  - OS 挂起当前进程，处理中断，选择和恢复下一个进程
- 采用虚拟机监视器
  - VMM挂起当前VM, 处理中断, 选择和恢复下一个虚拟机
- 如果一个VM 需要时钟中断
  - VMM模拟一个虚拟时钟
  - 当物理时钟中断发生时，为VM模拟中断

指令集支持

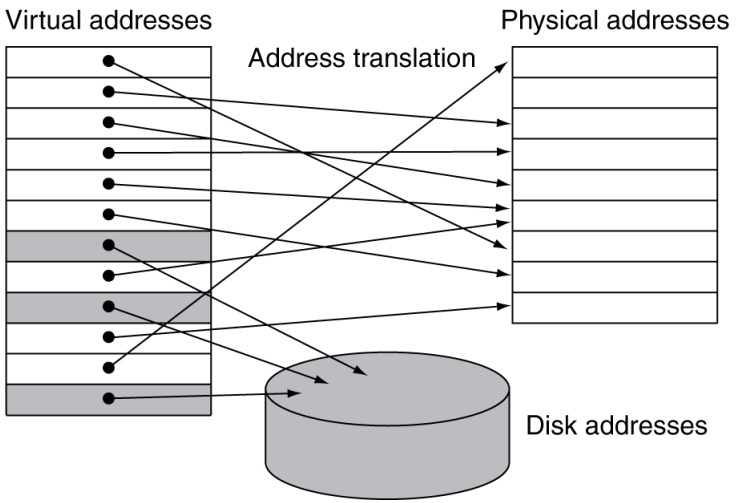
- 用户和系统模式
- 特权级指令智能在系统模式下使用
  - 如果在用户模式下执行将会产生trap中断
- 所有物理资源只能通过特权指令访问
  - 包括页表、中断控制、I/O寄存器等
- 大部分指令集在创建时没有考虑虚拟化思想
  - X86和大部分RISC系统结构都是如此

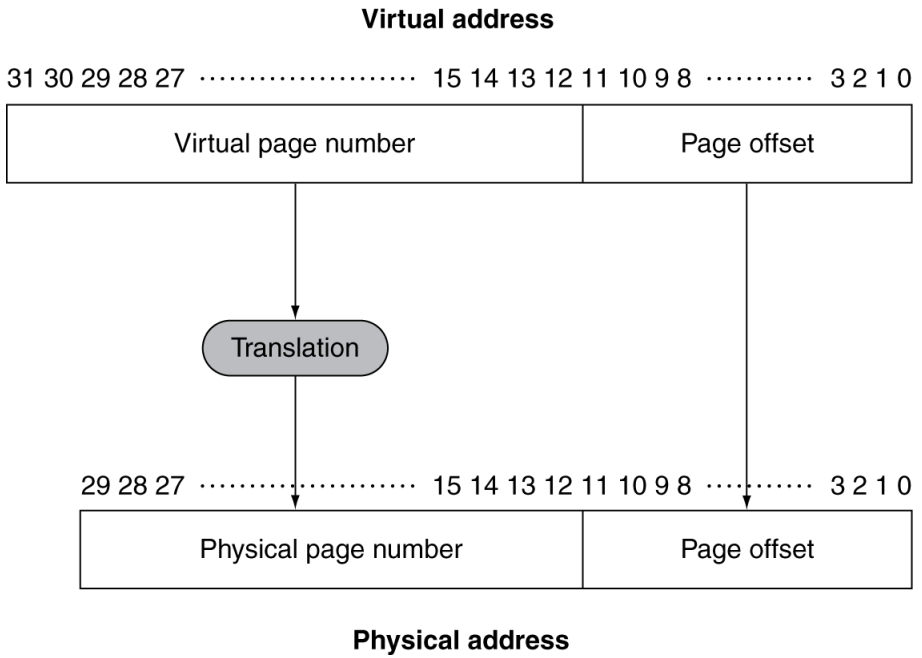
虚拟存储器

- 利用主存作为二级存储（磁盘）的 “cache”
  - CPU硬件和操作系统 (OS) 共同管理
- 程序共享主存
  - 每个程序获得私有的虚拟地址空间，用来保存经常访问的代码和数据
  - 与其他程序隔离
- CPU和 OS 翻译虚拟地址到物理地址
  - VM “块（block）”称为页（page）
  - VM 映射的 “缺失（miss）”称为 缺页（page fault）

地址映射

- 固定页大小 (e.g., 4K)





虚拟存储器：
虚页号（Virtual Page Number）
页偏移（Page Offset）

物理虚拟存储器：
物理页号（ Physical Page Number）
页偏移（Page Offset）

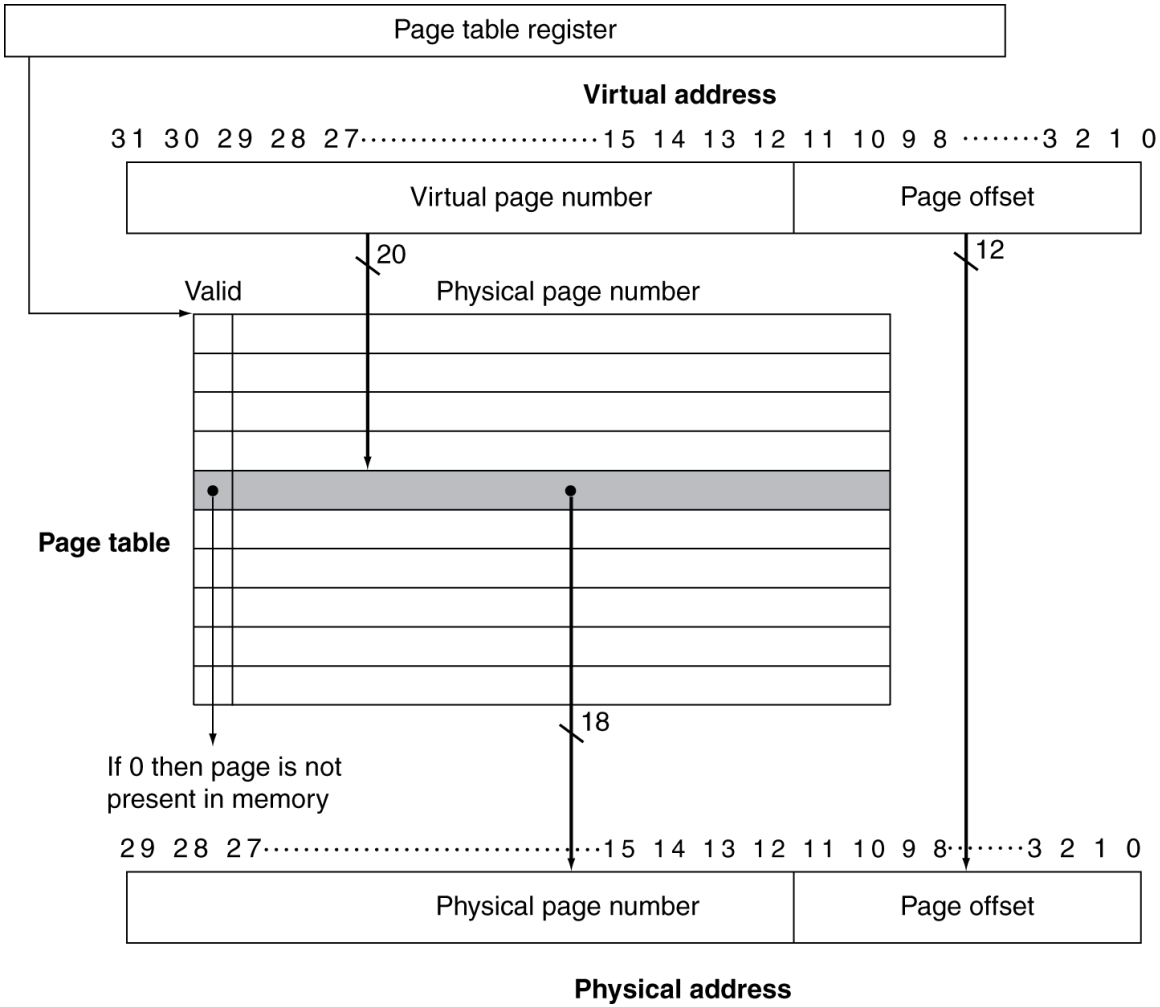
缺页的代价

- 如果发生了缺页，必须从磁盘上取回该页
  - 消耗几百万个时钟周期
  - 由OS代码管理
- 尽量降低缺页率
  - 存储器中的页按照全相联的方式放置
  - 智能替换算法

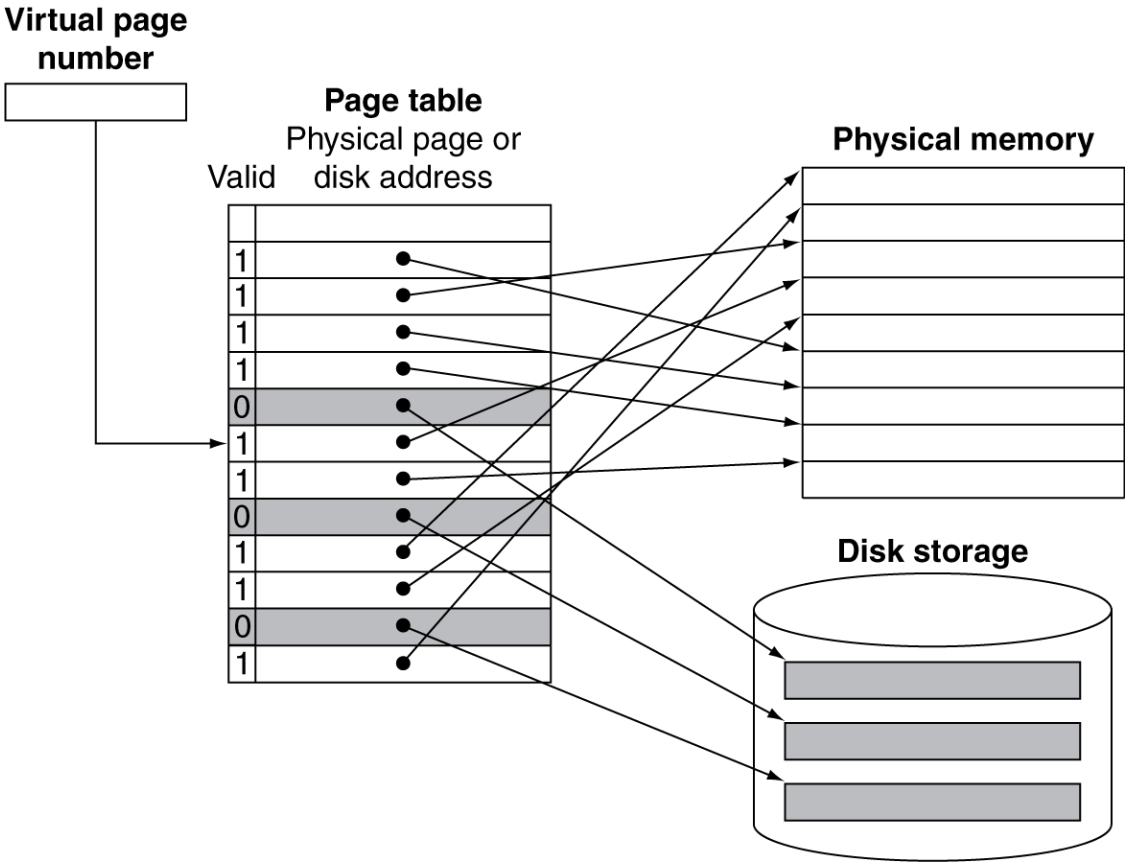
页表（PTE）

- 储存布局的信息
  - 保存着虚拟地址和物理地址之间转换关系的表。页表保存在主存中，通常使用虚页号来索引
  - CPU中的页表寄存器指向物理内存中存储的页表首地址
- 如果页在主存中
  - PTE存储了物理页号
  - 和其他状态位 (referenced, dirty, …)
- 如果页不在主存中
  - 由于无法提前获知存储器中某一页被替换出去的时间，操作系统在创建进程时，通常在磁盘中为进程中的所有页创建空间（称为交换区）
  - PTE需要利用数据结构指向每个虚拟页在交换区中存放的位置

页表（PTE）—页表转换


Page table register
Virtual address
31 30 29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not present in memory
29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Physical page number
Page offset
Physical address

页表—页表到存储的映射


Virtual page number
Page table
Physical page or disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

页表（PTE）—替换和写入

- 为了降低缺页率，推荐使用LRU替换算法
  - 某一页被访问时，PTE中的引用位 (aka 使用位)设置为1
  - 周期性地被OS清零
  - 某一页的引用位为0：说明最近未被使用
- 磁盘写入需要花费几百万个时钟周期
  - 成块写入，不单独写某个地址
  - 写直达在实际应用中不现实
  - 利用写回 (write-back)

页表（PTE）—例子

虚拟地址访问顺序：

4095，30860，15000，12789，4098，19215

系统页大小4KB， LRU替换算法，页表初态如下，问：

（1）是否命中，（2）页表访问完的状态

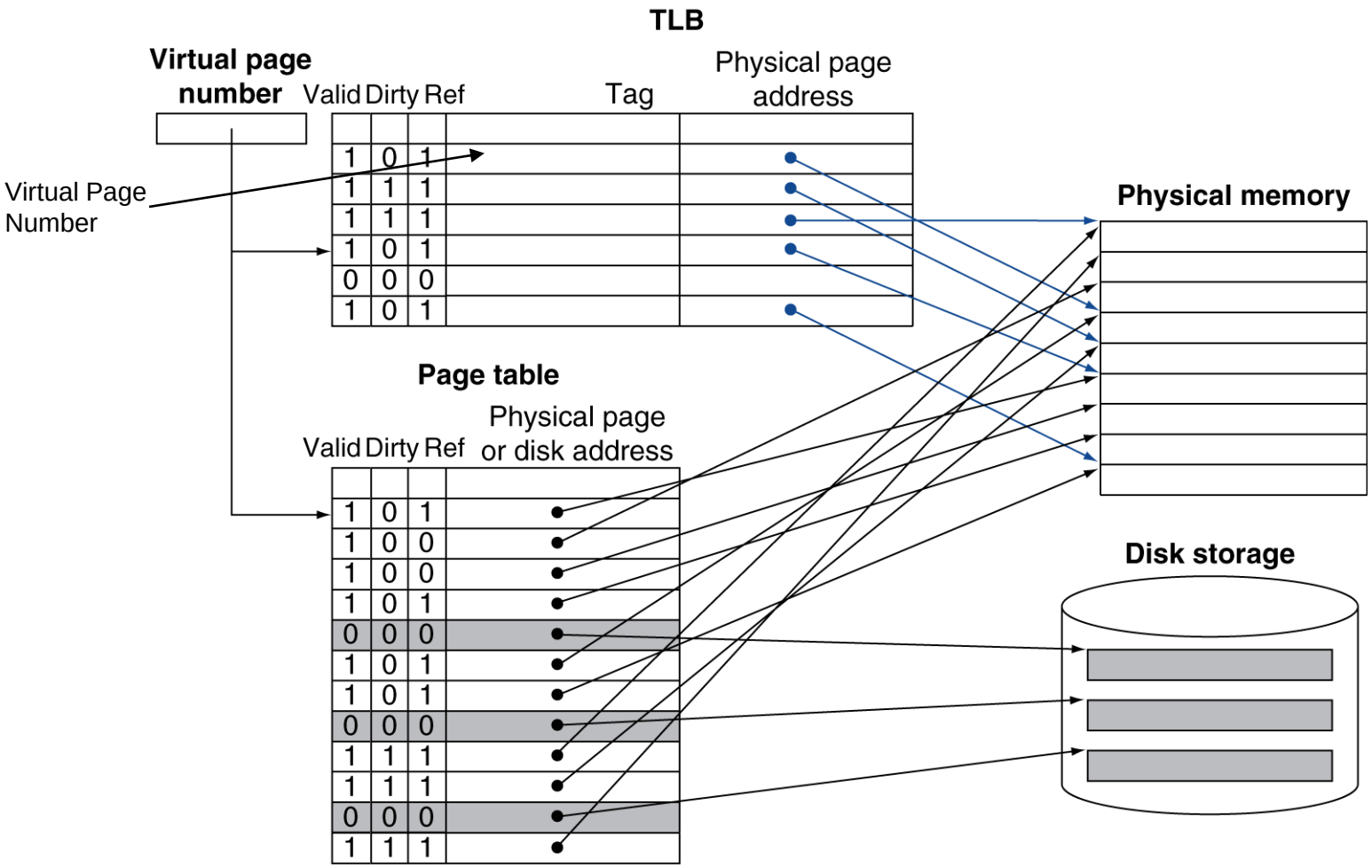
有效位	物理页/硬盘上
1	2
1	5
0	硬盘
0	硬盘
1	9
1	11
0	硬盘
1	4
0	硬盘
1	3

4095	0
30860	7
15000	3
12789	3
4098	1
19215	4

利用TLB进行快速地址转换

- 地址映射需要额外的访存开销
  - 第一次访问 PTE
  - 第二次访问实际内存地址
- 提高访问性能的关键在于依靠页表的访问局部性
  - 现代CPU都包含了一个特殊的cache以跟踪最近使用过的地址变换
  - 称为快表 或 地址变换高速缓存 Translation Look-aside Buffer (TLB)
  - 典型值: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - 可以由硬件或软件来管理缺失

利用TLB进行快速地址转换



TLB 缺失

- 如果页在主存
  - 从主存中加载 PTE 然后重试
  - 可以利用硬件来处理
    - 可以采用更为复杂的页表结构
  - 或者可以利用软件来处理
    - 使用更为先进的算法选择替换页
- 如果页不在主存 (缺页page fault)
  - OS 来提取页，并更新页表
  - 随后重新开始执行缺页的指令

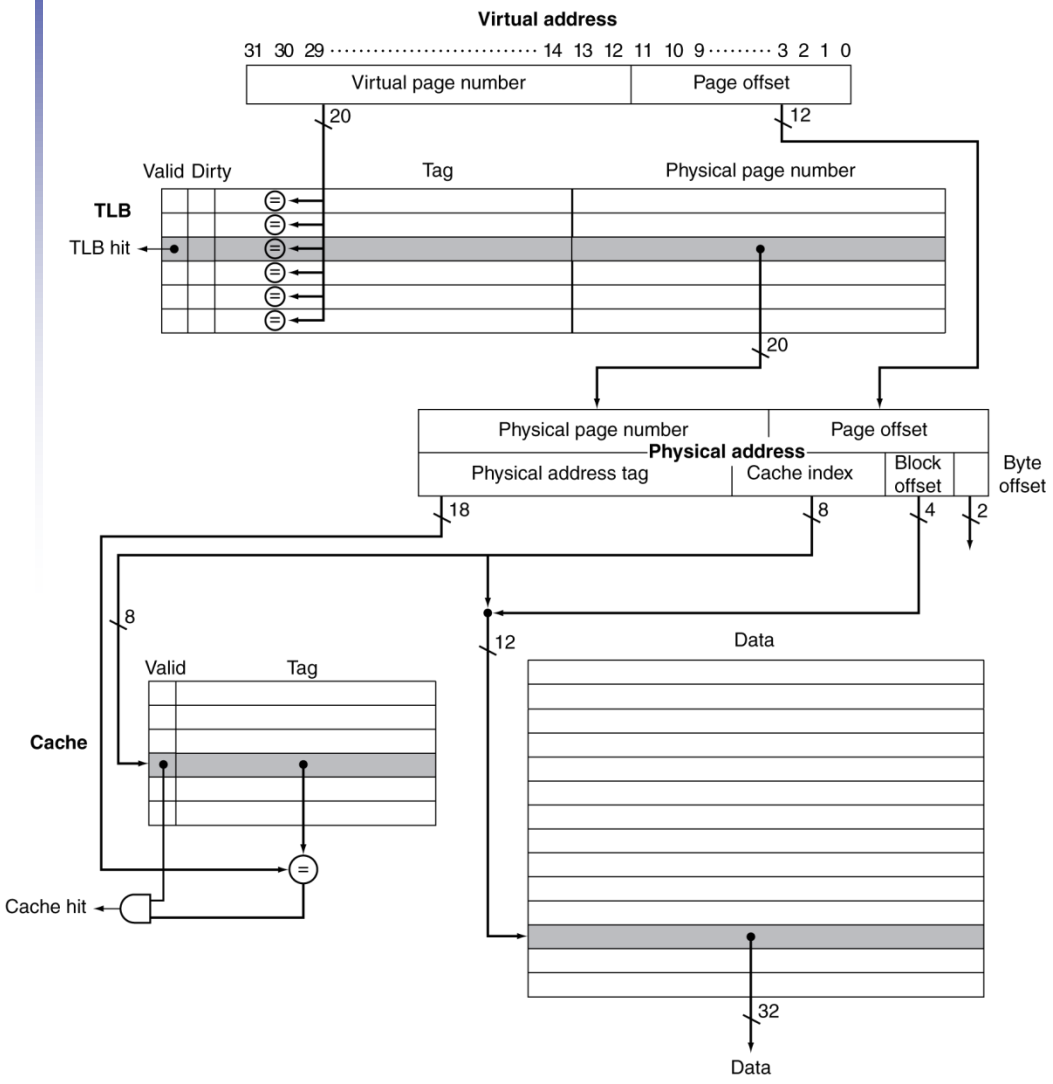
TLB 缺失处理

- TLB 的缺失说明
  - 页存在，但是 PTE 不在 TLB
  - 页不存在
- 必须在目标寄存器被覆盖前发现 TLB缺失
  - 异常处理
- 处理器从主存拷贝PTE到TLB
  - 随后重启指令
  - 如果不存在该页，会发生缺页（page fault）

缺页处理

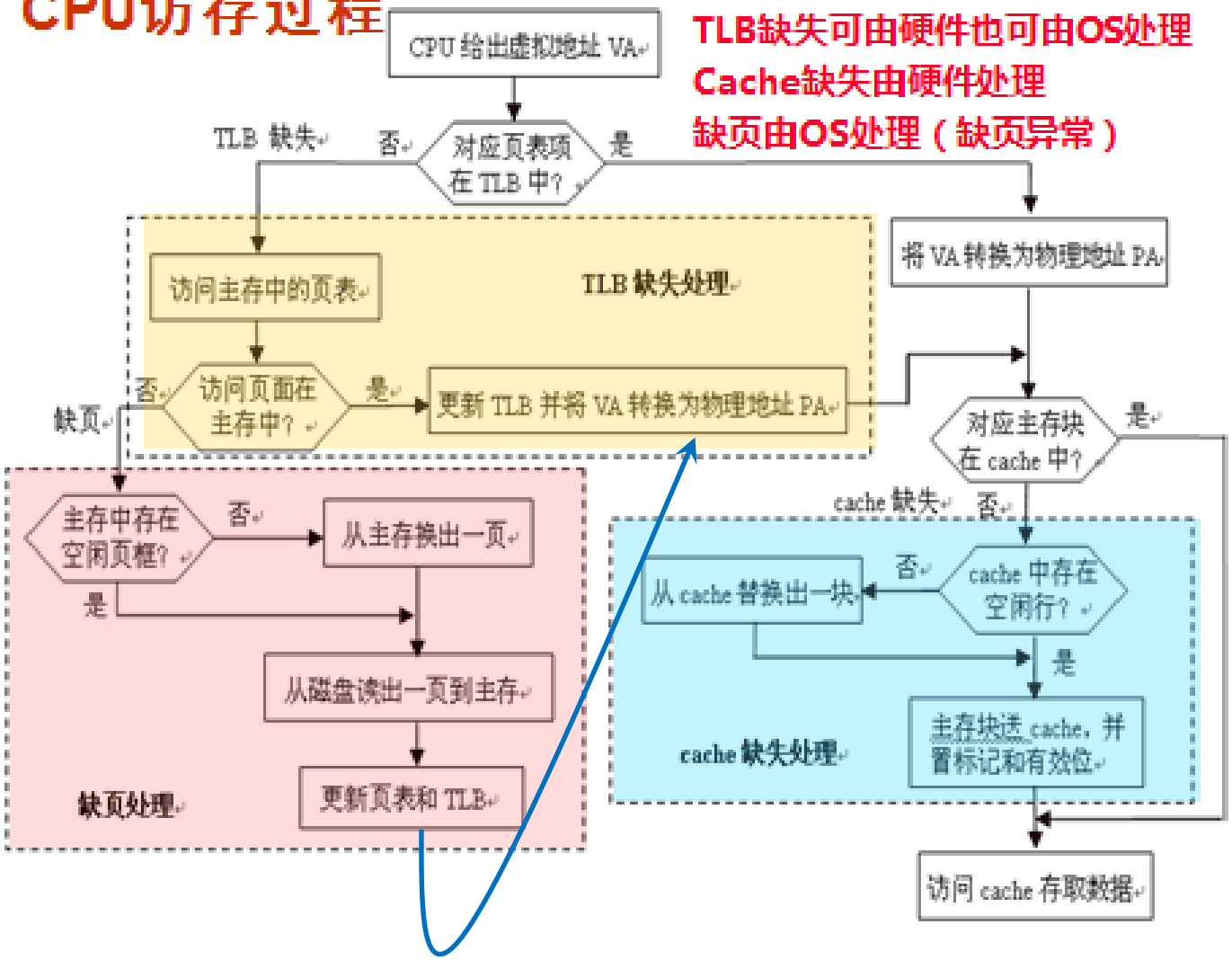
- 利用发生缺失的虚拟地址来查找 PTE
- 定位磁盘上的页
- 选择要替换的页
  - 如果是页的状态是dirty,先写到磁盘上
- 将页读到主存并更新页表
- 重新运行进程
  - 从发生缺失的指令开始重新运行指令

TLB与Cache的交互



- 如果cache 标志tag用了物理地址
  - 在 cache 查找前需要进行地址转换
- 另一种方法: 利用虚拟地址的标志 tag
  - 由于不同地址混淆造成的问题
    - 不同的虚拟地址可能共享物理地址

CPU访存过程



TLB缺失可由硬件也可由OS处理

Cache缺失由硬件处理

缺页由OS处理（缺页异常）

TLB讨论（P301）

- 存储器层次结构如图5-30所示，由一个TLB和一个Cache组成。一次存储器访问可能遇到三种不同类型的缺失：TLB缺失、缺页以及Cache缺失。考虑这三种缺失发生一个或多个时所有可能的组合（7种可能性），说明这些情况是否会真的发生，在什么条件下发生。

TLB例题

- 假设某系统采用4KB的页，一个4项的全相联TLB，使用LRU。如果必须从磁盘中取回页，那么增加下一次能取的最大页数，下面是该系统所看见的虚拟地址流、TLB和页表的初始状态
- 4669,2227,13916,34587,48870,12608,49225

TLB的初始状态

有效位	标记位	物理页号
1	11	12
1	7	4
1	3	6
0	4	9

页表的初始状态

有效位	物理页/硬盘上
1	5
0	硬盘
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
0	硬盘
0	硬盘
1	3
1	12

请给出系统的最终状态？
对于每次访问，需要列出是否在TLB中命中，是否在页表中命中或者发生缺页？

TLB例题(cont.)

虚拟地址　4669
虚页　　1　=>　PF

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	11	12
1	7	4
1	3	6
0 → 1	4 → 1	9 → 13

页表的状态

有效位	物理页/硬盘上
1	5
0 → 1	硬盘 → 13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
0	硬盘
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　2227
虚页　　0　=>　M

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	11	12
1	7	4
1	3	6
1	1	13
1	0	5

替换

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
0	硬盘
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　13916
虚页　　3　=>　H

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	7	4
1	3	6
1	1	13
1	0	5
1	3	6

改变LRU顺序

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
0	硬盘
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　34587
虚页　　8　=>　PF

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	7	4
1	1	13
1	0	5
1	3	6
1	8	14

替换

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
0-> 1	硬盘 -> 14
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　48870
虚页　　11　=>　M

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	1	13
1	0	5
1	3	6
1	8	14
1	11	12

替换

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
1	14
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　12608
虚页　　3　=>　H

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	0	5
1	3	6
1	8	14
1	11	12
1	3	6

改变LRU顺序

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
1	14
0	硬盘
1	3
1	12

TLB例题(cont.)

虚拟地址　49225
虚页　　12　=>　PF

- H:　Hit in TLB;
- M:　Miss in TLB hit in PT;
- PF:　Page Fault

TLB的状态

有效位	标记位	物理页号
1	0	5
1	8	14
1	11	12
1	3	6
1	12	15

替换

页表的状态

有效位	物理页/硬盘上
1	5
1	13
0	硬盘
1	6
1	9
1	11
0	硬盘
1	4
1	14
0	硬盘
1	3
1	12
1	15

存储器的保护

- 不同的任务可以共享各自一部分的虚拟地址空间
  - 但是需要防护恶意进程
  - 需要 OS 的帮助
- 硬件支持的 OS 保护
  - 超级用户管理模式 (aka kernel mode)
  - 特权指令
  - 只能在超级用户模式下访问的页表和其他状态信息
  - 特殊系统调用 (e.g., MIPS指令集中的syscall)

存储器体系结构

The BIG Picture

- 在存储器体系结构中，不同层次采用通用的原则
  - 主要使用cache中的术语
- 在每一个存储器层次
  - 块的放置
  - 寻找一个块
  - 缺失时替换
  - 写策略

块的放置

- 由相联度所决定
  - 直接映射 (1路相联)
    - 每个组中只有1个块可以放置
  - n路组相联
    - 每组的块数有n个选择
  - 全相联
    - 任意位置
- 提高关联度可以降低缺失率
  - 增加复杂性, 开销, 和访问时间

查找一个块

机制	定位方法	需要比较的次数
直接映射	索引	1
n路组相联	索引组，再查找组中的元素	n
全相联	查找所有的cache项	Cache的项数
	独立的查找表	0

- Cache 的硬件化
  - 减少比较的开销
- 虚拟存储器
  - 独立的查找表使得全相联成为可能
  - 有效降低缺失率

替换算法

- 缺失时的替换策略
  - 最近最少使用 (LRU)
    - 对于相联度高的映射机制，实现复杂且需要较贵的硬件开销
  - 随机
    - 和LRU性能相似,易于实现
- 虚拟内存
  - 提供引用位或者其他等价的功能使操作系统更方便地追踪一组最近最少使用的项。

写策略

- 写直达
  - 同时更新上层和下层
  - 简化了替换，但是需要写缓冲
- 写回
  - 仅更新上层
  - 当块被更新时，更新下层
  - 需要记录多个状态
- 虚拟内存
  - 写到磁盘的延迟很大，只有写直达可行

缺失的原因：3C

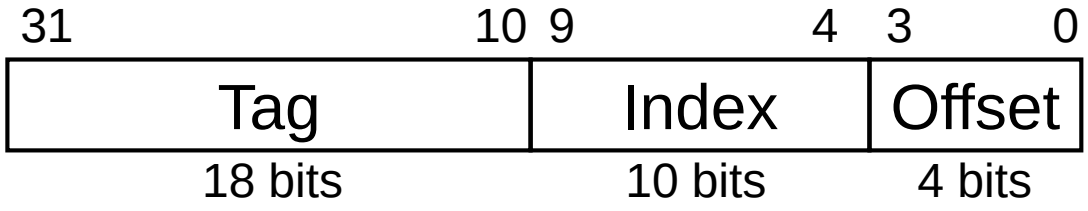
- 强制缺失 Compulsory misses
  - 对从没有在cache中出现的块第一次访问引起的缺失，或称为冷启动缺失
- 容量缺失 Capacity misses
  - Cache的容量不足以容纳一个程序执行所需的所有块所引起的缺失
  - 一个被替换的块，随后又一次被访问
- 冲突缺失 Conflict misses
  - 在组相联或直接映射的 cache 中
  - 多个块竞争同一个组时而引起的cache缺失
  - 同样容量的全相联cache中不存在这种情况

Cache设计的折中

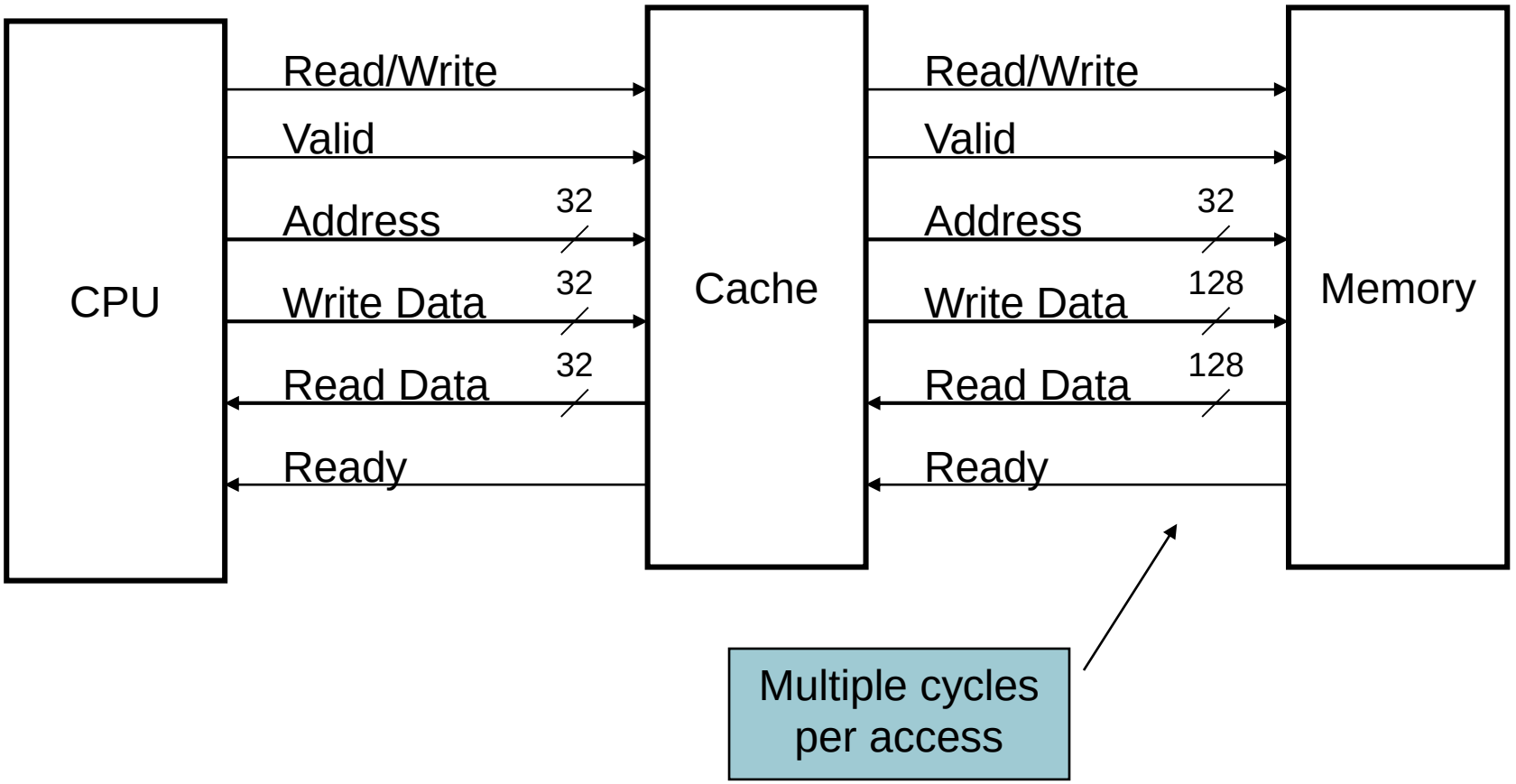
设计变化	对缺失率的影响	可能对性能产生的负面影响
增加cache容量	减少了容量缺失	可能增加访问时间
提高相联度	由于减少了冲突缺失，因此降低了缺失率	可能增加访问时间
增加块的容量	由于空间局部性，因此对很宽范围内变化的块大小，都能降低缺失率	增加缺失代价，块太大还会增加缺失率

使用有限状态机控制简单的Cache

- Cache的关键特征
  - 直接映射、写回机制、写分配策略
  - 块大小: 4个字 (16 字节或者128位)
  - Cache容量: 16 KB (1024 个块)
  - 32-bit 字节地址
  - 每个块包含有效位和写入位（dirty bit）

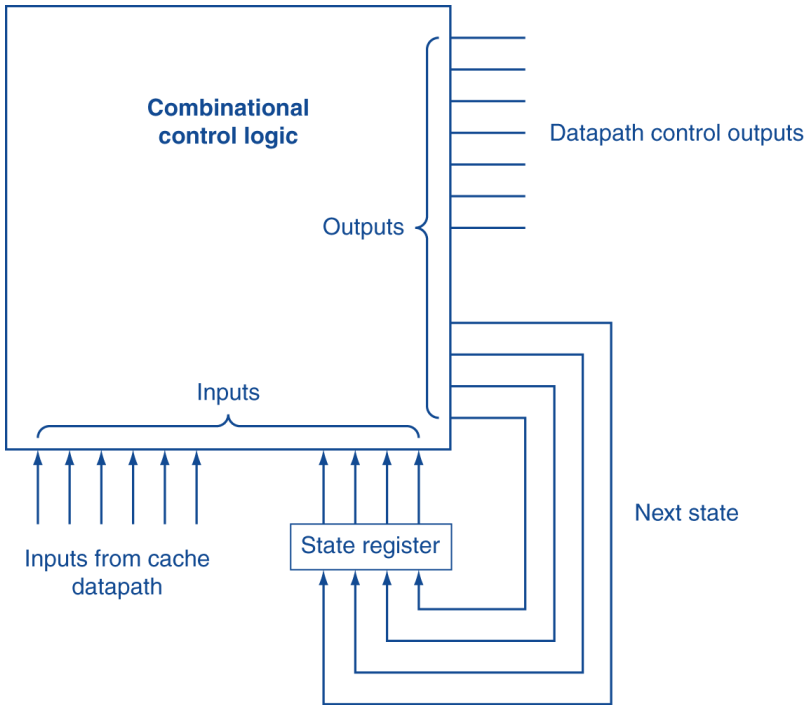


接口信号

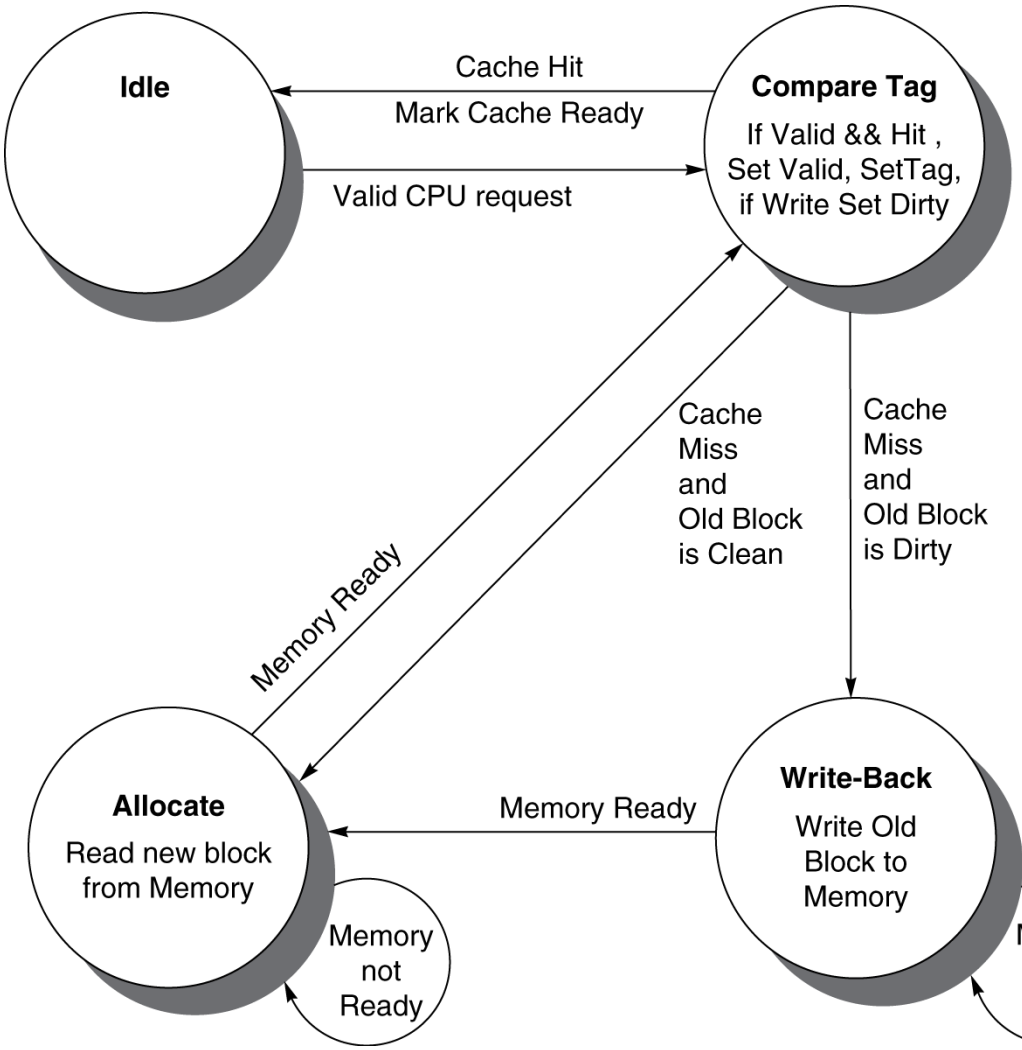

CPU
Read/Write
Valid
Address
32
Write Data
32
Read Data
32
Ready
Cache
Read/Write
Valid
Address
32
Write Data
128
Read Data
128
Ready
Memory
Multiple cycles per access

有限状态机

- 利用FSM顺序控制步骤
- 一组状态，在每个时钟边沿转换状态
  - 状态值由二进制编码
  - 当前状态储存在寄存器里
  - 下一个状态 = f_n (当前状态,当前输入)
- 控制输出信号 = f_o (当前状态)



FSM Cache控制器



可以分成多个独立的状态，以减少时钟周期时间

Cache一致性问题

- 假设两个CPU核心共享同一个物理地址空间
  - 写直达caches

时间	事件	CPU A的cache内容	CPU B的cache内容	存储器的内容
0				0
1	CPU A 读 X	0		0
2	CPU B 读 X	0	0	0
3	CPU A 写 1 到 X	1	0	1

一致性的定义

- 非正式定义: 读操作返回的内容和写一致
- 正式定义:
  - P 对地址X写入; P读出地址X中的内容 (之间不包括其他写入)
    ⇒ 读返回了所写的内容
  - P_1对地址X写入; P_2读出地址X中的内容 (有足够的时间间隔)
    ⇒ 读返回了所写的内容
    - c.f. CPU B在第3步之后，读了地址X
  - P_1对地址X写入, P_2 对地址X写入
    ⇒ 所有的处理器看到些操作在相同的顺序
    - 结果X具有相同的最终值

Cache一致性协议

- 在支持cache一致性的多核处理器系统中，实现一致性的方案
  - 数据迁移到本地cache
    - 减少了访问共享内存的带宽
  - 复制“读共享”数据
    - 减少内容的访问
- 监听（Snooping）协议
  - 每个cache监听总线的读和写
- 目录（Directory）协议
  - Cache和内存在目录中记录共享块的状态

多级片上Caches

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

两级 TLB 架构

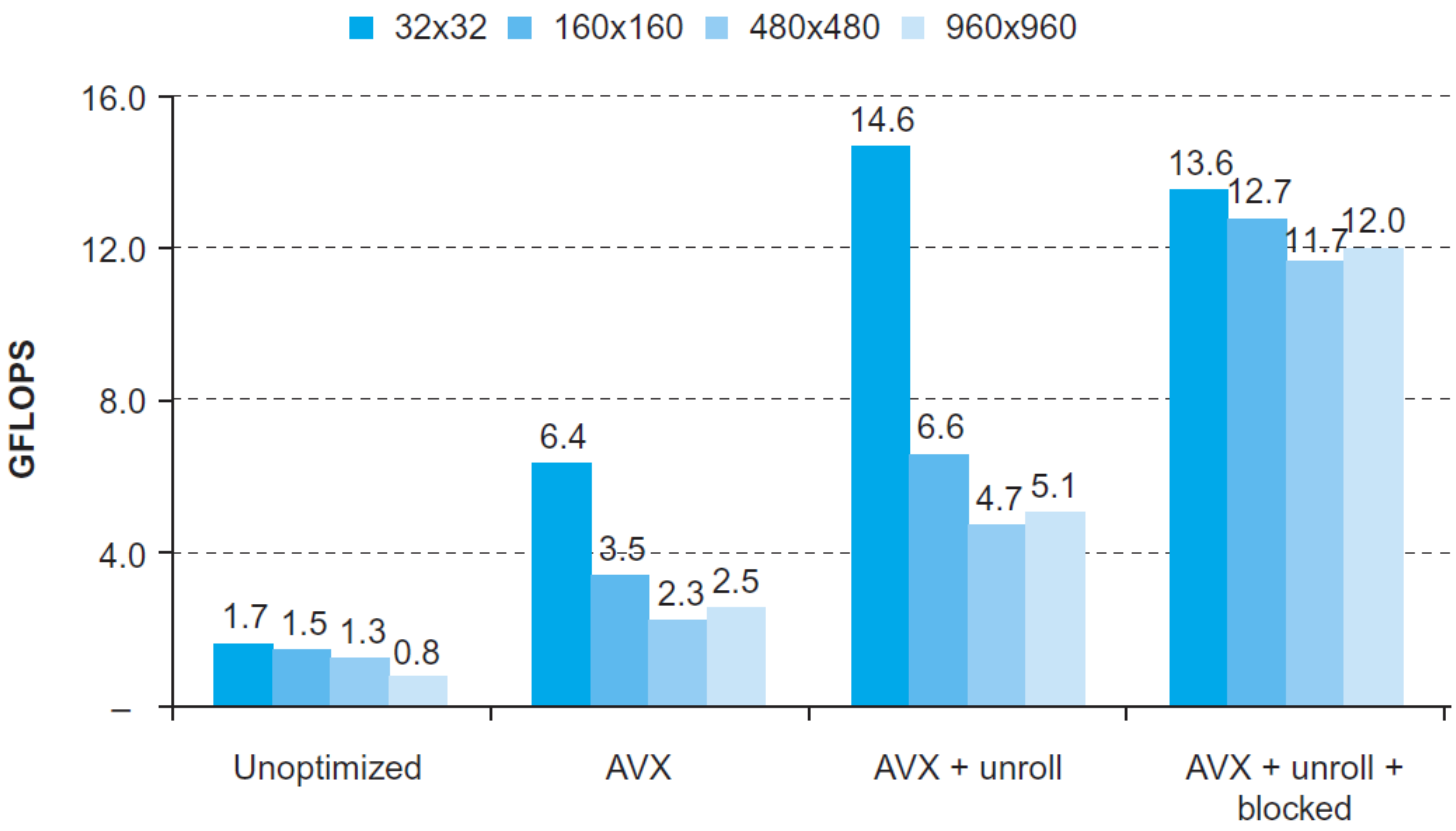
Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

多种优化措施

- A8和Core i7都需要支持每个时钟周期执行一条以上的访存指令
- 将cache分成多个体(bank),在不发生体冲突时，能对多个体并行进行访问
- Core i7 cache优化
  - 请求字优先策略
  - 非阻塞 cache
    - 缺失命中 （Hit under miss）
    - 缺失情况下的缺失（Miss under miss）
  - 数据预取

DGEMM

- 当矩阵规模从32*32增加到960*960时4种版本的DGEMM的性能。对于规模最大的矩阵，完全优化的代码的性能几乎是第三章图3-21中未优化代码的15倍



陷阱

- 在写程序或编译器生成代码时忽略存储系统的行为
  - 举例: 循环访问数组的多个行或列
  - 存储地址跨度较大造成很差的局部性
- 字节编址 vs. 按字编址
  - 举例: 32字节的直接映射cache, 块大小是4字节
    - 字节地址 36映射到块 1
    - 字地址 36映射到块 4

陷阱

- 对于共享cache,组相联度少于核的数量或者共享该cache的线程数
  - 较低的相联度将导致冲突缺失
  - 更多核心 ⇒ 需要增加相联度
- 利用平均访问时间AMAT来评估乱序处理器的存储器层次结构
  - 忽略了访问非阻塞块的效果
  - 解决方法：模拟乱序处理器和存储器结构

陷阱

- 通过在未分段地址空间的顶部增加段来扩展地址空间
  - 举例, Intel 80286
  - 但是段并不总是足够大
  - 使得地址逻辑变得非常复杂
- 在不为虚拟化设计的指令集体系结构上实现虚拟机监视器
  - 举例, 非特权指令访问硬件资源
  - 解决方法：要么扩展ISA, 要么要求用户OS不去访问可能出现问题的指令

结论

- 多快好省在这个世界上不存在
- 快的存储器容量小, 容量大的存储器速度慢
  - 又快又大的存储器是人类的目标 ☹
  - 多级缓存使人们看到了梦想的美好 ☺
- 局部性原理
  - 程序频繁访问一小部分的存储空间
- 存储器层次结构
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM 内存 ↔ 磁盘
- 存储器系统的设计对于多核处理器至关重要

习题一

- 对于虚地址13048，如果页的大小是2KB，那么请问相应的虚页号和页内偏移各自是什么？

- 对于虚地址13048，如果页的大小是2KB，那么请问相应的虚页号和页内偏移各自是什么？

- 6/760　110 0010 1111 1000

- 对于虚地址13048，如果页的大小是2KB，那么请问相应的虚页号和页内偏移各自是什么？
- 6/760　110 0010 1111 1000
- 如果上述的虚页映射到12号物理页，请问其物理地址是什么？

- 对于虚地址13048，如果页的大小是2KB，那么请问相应的虚页号和页内偏移各自是什么？
- 6/760　110 0010 1111 1000

- 如果上述的虚页映射到12号物理页，请问其物理地址是什么？
- 25336　1100　0010 1111 1000

习题二

- 对于4个块的cache，如果采用2路组相联方式，并假定刚开始cache没有有效数据，当访问0，1，2，3，3，6，7，6，7，0，1块时，请问此时命中率是多少？

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	m	1	0		
1	1	m	1	1		
2	2	m	1	2	1	6
3	3	m	1	3	1	7
3	3	h				
6	2	m				
7	3	m				
6	2	h				
7	3	h				
0	0	m				
1	1	m				

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	0			
1	1	miss	0		1	
2	0	miss	0	2	1	
3	1	miss	0	2	1	3
3	1	hit	0	2	1	3
6	0	miss	6	2	1	3
7	1	miss	6	2	7	3
6	0	hit	6	2	7	3
7	1	hit	6	2	7	3
0	0	miss	6	0	7	3
1	1	miss	6	0	7	1

习题三

- 如果有一个处理器的理想CPI=1，结构设计师正在为它选择存储部件，
- 方案1: I-cache 缺失率2.2% D-cache缺失率3.9%，缺失代价100个时钟周期；
- 方案2: I-cache缺失率2.6% D-cache缺失率3.6%，缺失代价85个时钟周期。

目标程序中访问内存的指令占38%。请问选择哪个方案会获得更好的性能？

- 如果有一个处理器的理想**CPI=1**，结构设计师正在为它选择存储部件，方案**1: I-cache** 缺失率**2.2% D-cache**缺失率**3.9%**，缺失代价**100**个时钟周期；方案**2 I-cache**缺失率**2.6% D-cache**缺失率**3.6%**，缺失代价**85**个时钟周期。目标程序中访问内存的指令占**38%**。请问选择哪个方案会获得更好的性能？

- CPI1=1+2.2%*100+3.9%*38%*100=1+2.2+1.482=4.682
- CPI2=1+2.6%*85+3.6%*38%*85=1+2.21+1.163=4.373
- 故选方案2

习题四

- 如果处理器中$t0=4100、$t1=200当程序发出lw $t1,100($t0)指令时，程序访问的物理内存是那个单元？假设此时页表（页的大小为4KB）的部分内容如下

有效位	物理页/硬盘上
1	5
1	2
0	硬盘
1	6

- **如果处理器中$t0=4100、$t1=200当程序发出lw $t1,100($t0)指令时，程序访问的物理内存是那个单元？假设此时页表（页的大小为4KB）的部分内容如下**

有效位	物理页/硬盘上
1	5
1	2
0	硬盘
1	6

该指令访问的编程地址（虚地址）是4100+100=4200，4200=4096+104得出该地址对应于1号虚页、页内偏移104，经页表转换后为2号物理页，则物理地址为2*4096+104=8296