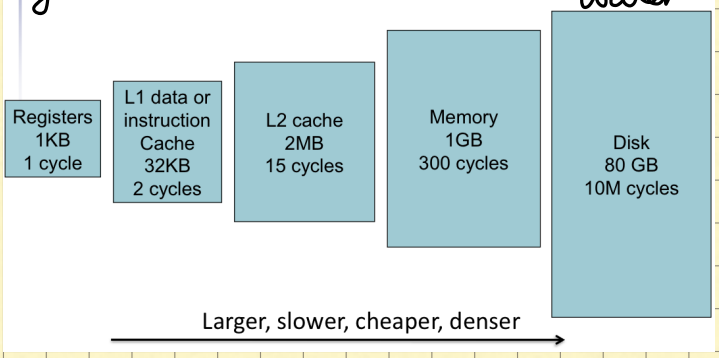
Memory Hierarchy

- Various storage devices in computers:

higher

lower



大部分capacity是在lower处的.
但总cost都是差不多的
而是平均cost越 higher 越贵.

Memory Technology

- Access time and price per bit vary widely among different technologies

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Data in 2012

- Ideal memory
  - Access time of Cache
  - Capacity and cost/GB of disk

理想的内存就是有Cache的速度 disk的大小和均价.

Cache Hierarchies

- Data and instructions are stored on DRAM chips
  - DRAM is a technology that has high bit density, but relatively poor latency
  - an access to data in memory can take as many as 300 cycles!
- Hence, some data is stored on the processor in a structure called the cache
  - caches employ SRAM technology, which is faster, but has lower bit density
- Internet browsers also cache web pages – same concept

DRAM的数据密度高,但是访问慢

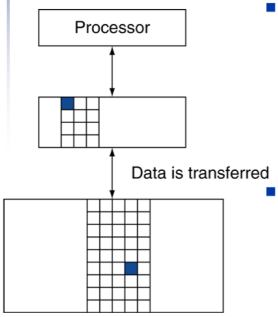
cache访问速度快,但是数据密度低

浏览器也用了cache.

Memory Hierarchy

- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels



- The memory in upper level is originally empty
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
  - Block (also called line): unit of copying
    - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses = 1 – miss ratio
  - Then accessed data supplied from upper level

上层的内存起始时是空的.
访问时若找不到，则需从memory复制
Miss：失效有惩罚.

block:每次复制移动的最小单元

尽量使命中率提高.

Locality

- Why do caches work?
  - Temporal locality: if you used some data recently, you will likely use it again
  - Spatial locality: if you used some data recently, you will likely access its neighbors
- No hierarchy:
  - average access time for data = 300 cycles
- 32KB 1-cycle L1 cache that has a hit rate of 95%:
  - average access time = 0.95 x 1 + 0.05 x (301) = 16 cycles

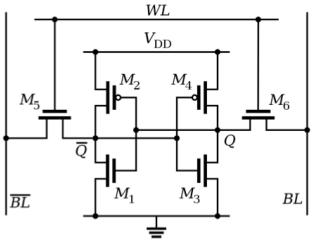
所有memory都会用这两个.

Temporal locality：最近常用的数据将很可能会被再次使用.
时间局部性

Spatial locality：最近常用的数据，其邻近数据将很可能被访问.
空间局部性

SRAM Technology

- Static RAM
  - Memory arrays with a single read/write port
- It's Volatile
  - The data will lost when SRAM is not powered
- Compared with DRAM
  - Don't need to refresh, use 6-8 transistors to install a bit
- Used in CPU cache, integrated onto the processor chip

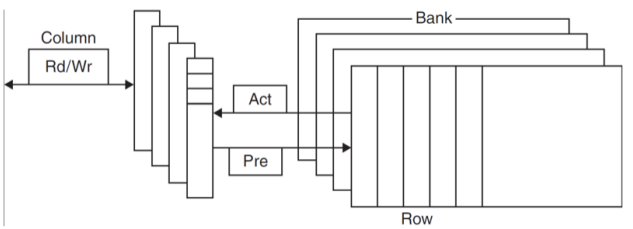


SRAM的内容不需要刷新.
需要更多电容保证不容易掉电
但断电会失去数据

用于CPU的cache，集成在处理器芯片上.

DRAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM "row"



数据以电容中的电荷存储.
仅使用单个晶体管存储该电荷.
需周期性刷新

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Synchronous DRAM
  - A clock is added, the memory and processor are synchronized
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
  - DDR4-3200 DRAM: 3200M times of transfer per second

同步DRAM
只有上升沿读

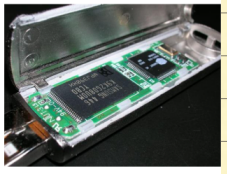
上下沿都读.

Flash Storage 闪存

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

掉电数据不消失

- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks



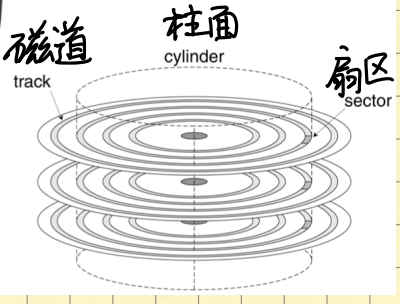
约1000次访问后flash bit就会失效.

耗损均衡: 将发生多次写操作的块重新映射到较少被写的块.

Disk Storage 磁盘

- Nonvolatile, rotating magnetic storage

掉电数据不消失

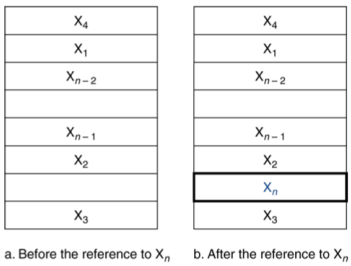


Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, …, X_{n-1}, X_n



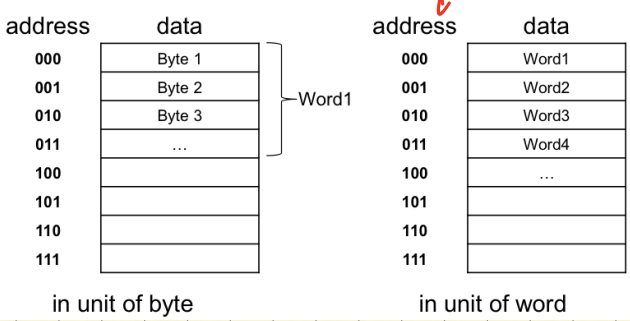
- How do we know if the data is present?
- Where do we look?

cache是离CPU最近的存储器.

Memory Structure

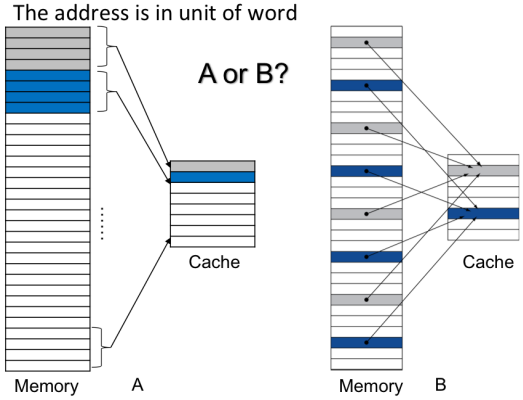
- Address and data
  - Address is the index, are not stored in memory
    - Address can be in unit of byte or in unit of word
  - Only data is stored in memory

这些地址只是索引,不存于内存中



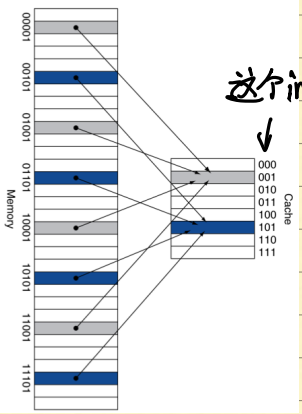
Direct Mapped Cache 直接映射

- Memory size: 32 words, cache size: 8 words, block size: 1 word
- The address is in unit of word



B会更好.
被踢出的概率更小.(空间局部性)
B有8个memory block映射到cache的同一block中.

- Memory size: 32 words, cache size: 8 words, block size: 1 word
- The address is in unit of word
- Direct mapped cache:
  - Location determined by address
  - One data in memory is mapped to only one location in cache
  - Use low-order address bits or high-order bits?
  - The lower bits defines the address of the cache
  - Index: which block to select



这个index只要看Memory中的后三位
(最低的三位)

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

tag 就是 Memory 的地址除去 index 的前面的位. 用于区分同时映射到同一 cache block 的 memory block

valid bit 用于表示某处数据是否有效.

Cache Example (以word表示)

- 8-blocks, 1 word/block, direct mapped
- Initial state

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

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		

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		

index 实际上是不用存的. 若将 tag 和 index 合起来存一个 memory 的 address, 会更耗存储空间.

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	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		

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		

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	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

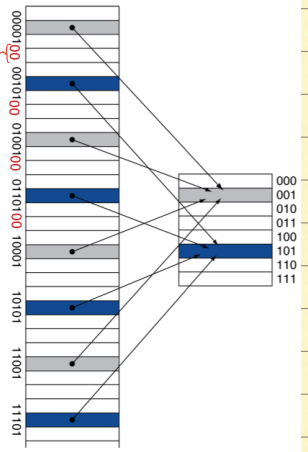
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

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		

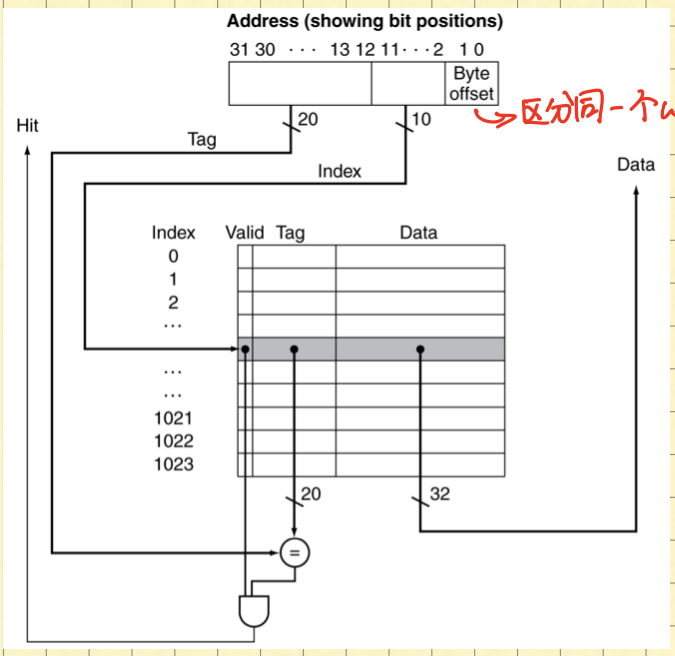
Memory in unit of byte

- How about the memory is in unit of byte? Assume:
- Memory size:
  - 32 words = 128 bytes
- Cache size:
  - 8 words = 32 bytes
- Block size:
  - 1 word = 4 bytes
- How to determine cache index and tag?

用于表示每个block中的第几个byte

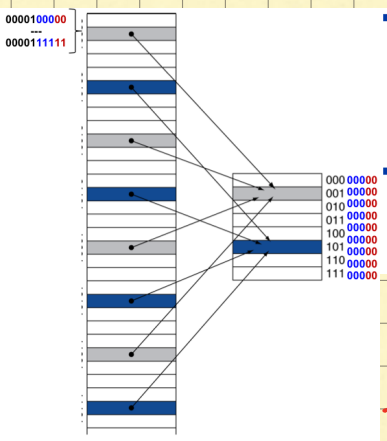


Address Subdivision

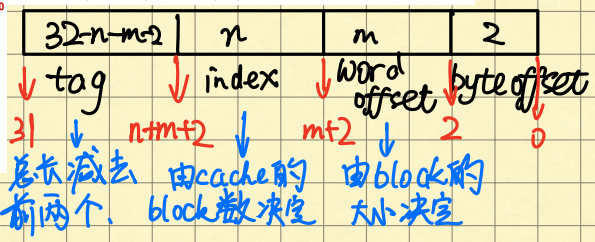


Larger Block Size

- How about the block size is larger? Assume:
- Memory size:
  - 256 words = 1024 bytes
- Cache size:
  - 64 words = 256 bytes
- Block size:
  - 8 word = 32 bytes
- How to determine cache index and tag?

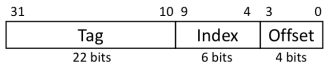


- Assume:
  - 32-bit address
  - Direct mapped cache
  - 2^n number of blocks, so n bit for index
  - Block size: 2^m words, so m bit for the word within the block
- Calculate:
  - Size of tag field: 32-(n+m+2)
  - Size of cache: 2^n*(block size + tag size +valid field size)
    $=2^n*(2^m*32+(32-n-m-2)+1)$



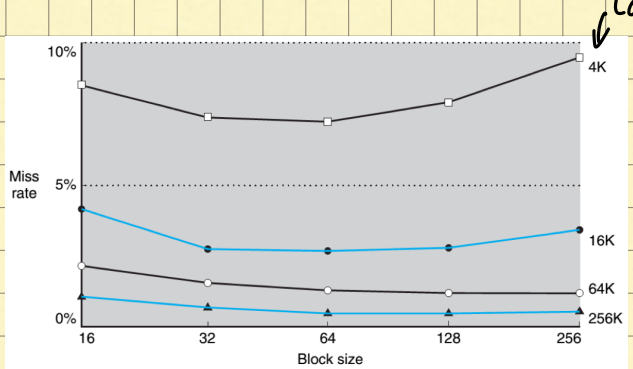
Example

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address = ⌊1200/16⌋ = 75 (相当于右移4位，即去除offset)
- Block number = 75 modulo 64 = 11 (相当于取后6位，即取index)



Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help



Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Read miss vs. write miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through 写穿透

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer 写缓冲
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

若在写入时修改cache的同时修改memory中的内存，那写入的时间将会很长

用一个buffer积累写入，直到buffer满了再一次性写入内存.

Write-Back 写返回

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

在buffer中积累写入，直到该block被踢出时才更新.

Write Allocation 写分配

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Write Policies Summary

- If that memory location is in the cache?
  - Send it to the cache
  - Should we also send it to memory right away? (write-through policy)
  - Wait until we kick the block out (write-back policy)
- If it is not in the cache?
  - Allocate the line (put it in the cache)? (write allocate policy)
  - Write it directly to memory without allocation? (no write allocate policy)

在写cache同时写到memory

在cache中要被踢出时写到memory } 都可用 buffer

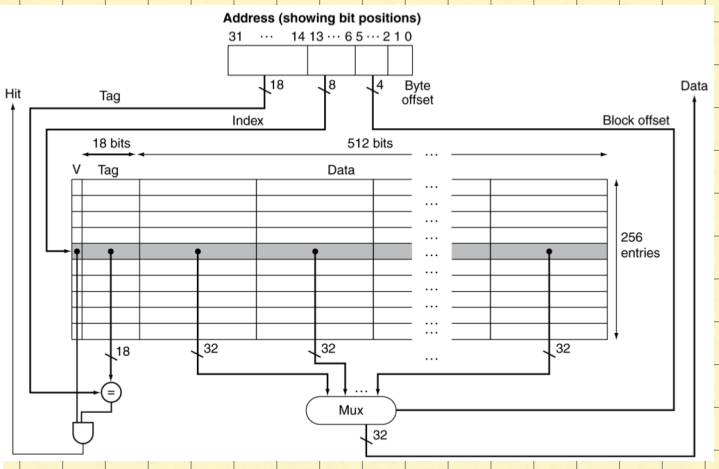
先放入 cache

直接 memory

Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache (指令缓存) and D-cache (数据缓存)
  - Each 16KB: 2^8 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

由于指令是按顺序访问的而数据不是



Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

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

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

程序执行周期

内存停顿周期.

Cache Performance Example

- Calculate actual CPI, given that
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction (assume *N* ins. In total)
  - I-cache: N × 0.02 × 100/N = 2
  - D-cache: N × 0.36 × 0.04 × 100/N = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time 平均访问时间.

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

AMAT = 命中时间 + 失误率 × 失误惩罚.

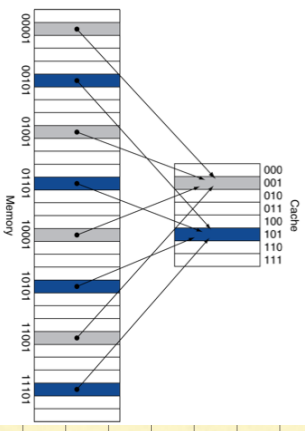
Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
  - CPI=2, Miss=3.44, % of memory stall: 3.44/5.44=63%
  - CPI=1, Miss=3.44, % of memory stall: 3.44/4.44=77%
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

cache的miss ratio是非常重要的.

Recall: Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - Capacity of cache is not fully exploited
  - Miss rate is high



Cache Example

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

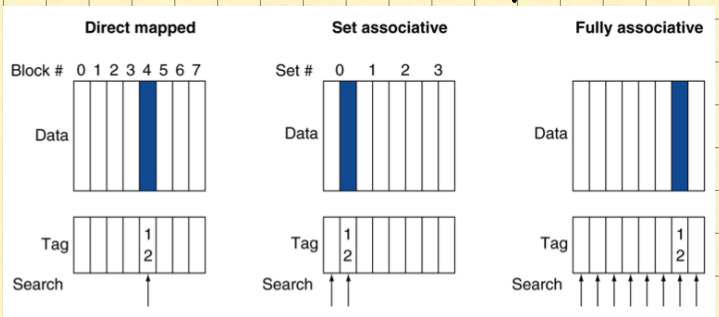
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010
26	11 010	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		

会踢掉目标地址中原有的数据.
但还有空的地方，没必要踢掉

Associative Cache Example



direct map: 一个entry即是一个block.

set: 每个set中可有多个block

Fully: 只要有一个地方不满,都可以放

Associative Caches

- Fully associative 全相连
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative 组相连
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

可以让给定的block进入任意entry

每次对全部entry都比较看是否空

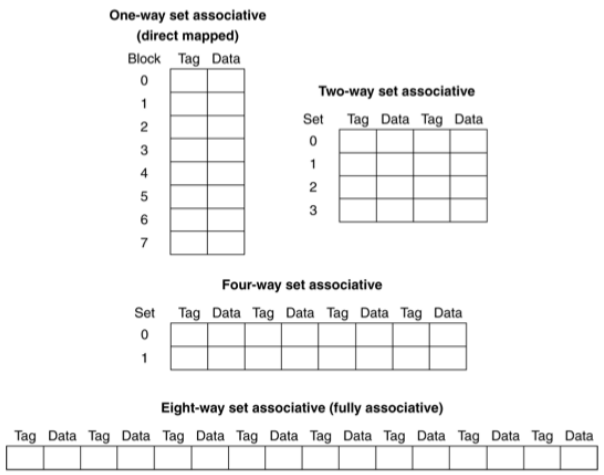
因此会使成本上升. 全相连一般只用于较小容量的cache.

每个set中有n个entry

对同一个set内看是否有空.

Spectrum of Associativity

- For a cache with 8 blocks



directed mapped的miss rate最高

fully的miss rate最低.

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

5次访问5次miss

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]	

- 2-way set associative

5次访问4次miss

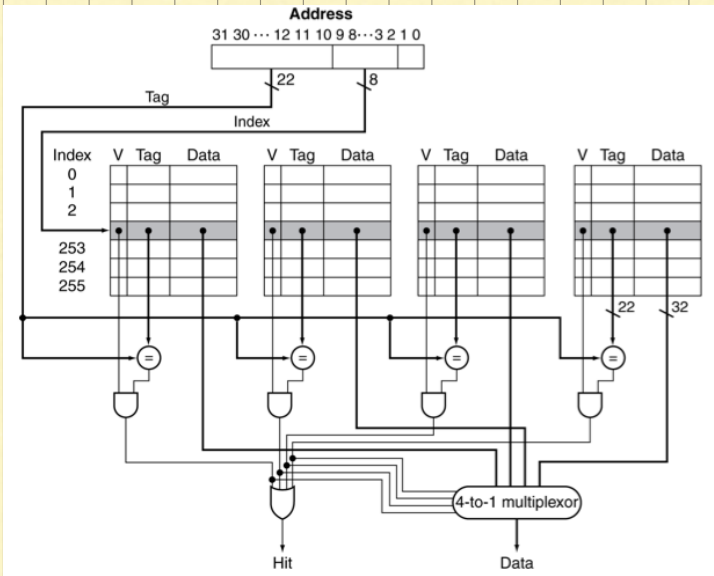
Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

- Fully associative

5次访问3次miss

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]	

Set Associative Cache Organization



比较时4个都拿出来比较.

How much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

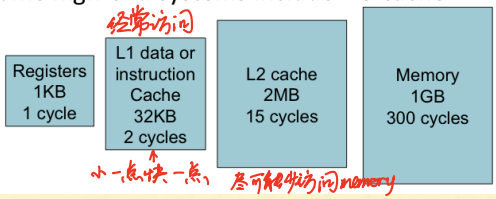
LRU: 将最少用到的踢出去.

Random: 随机踢出.

越高的相连度, random的效果与LRU越相似.
(全相连中一样效果)

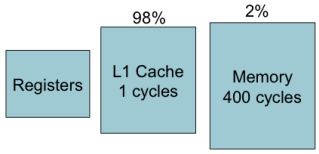
Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

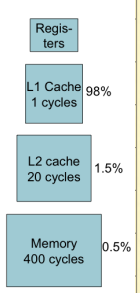


Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9



- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6



Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

第一级缓存：减少命中用时

第二级缓存：减少访问内存的次数

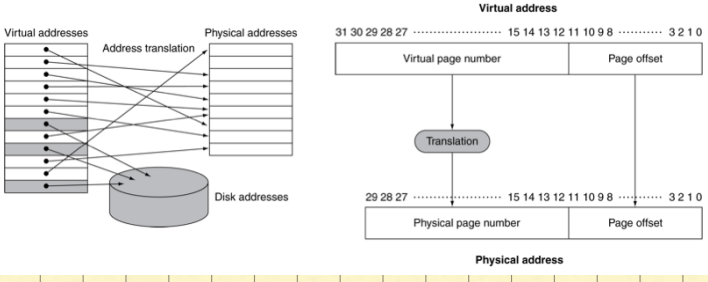
Virtual Memory

- Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page
  - VM "miss" is called a page fault

VM的block即page(页)
VM的miss即page fault(页错误)

Address Translation

- Fixed-size pages (e.g., 4K)



virtual memory不真正存东西，只表示一种地址映射。

Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

在出现page fault后，若重新访问disk，要花费很多时间。

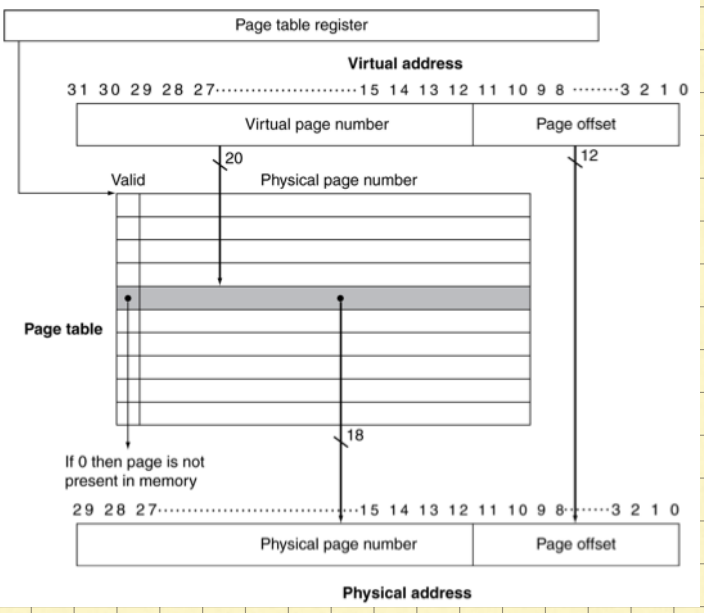
Page Tables

- Where is the placement information? Page Table
  - Array of page table entries (PTE), indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- Each program has its page table. Page table is in memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, ...)
- If page is not present
  - PTE can refer to location in swap space on disk

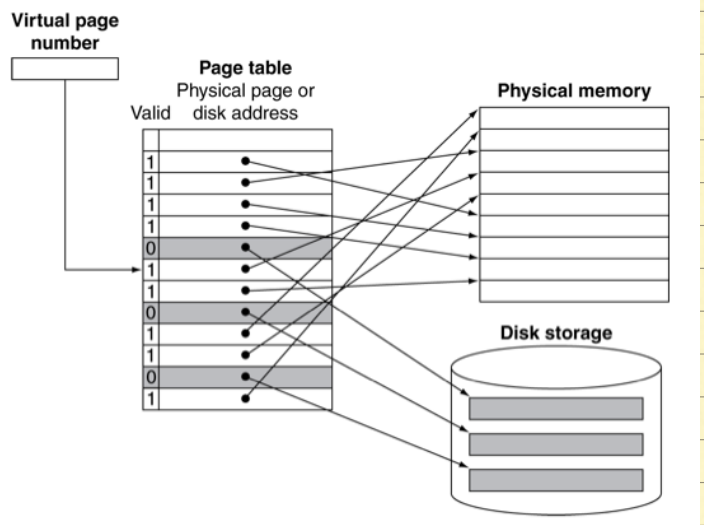
page table 内定义了从 virtual page number 到 physical memory 的映射.

每个程序都有 page table，且存于内存中.

Translation Using a Page Table



Mapping Pages to Storage



可以在 cache 内放一个小表，用于存常用的物理内存地址.

Replacement and Writes

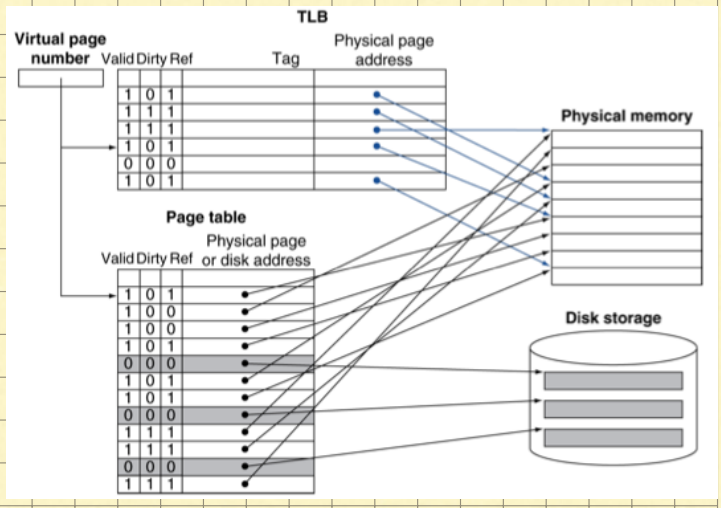
- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Use write-back, because write through is impractical
  - Dirty bit in PTE set when page is written

在表前加一列 reference bit，用于 LRU
会周期性清零.

在表前加一列 dirty bit，用于 write-back
(在被踢出去时是否要写回 disk)

Fast Translation Using a TLB

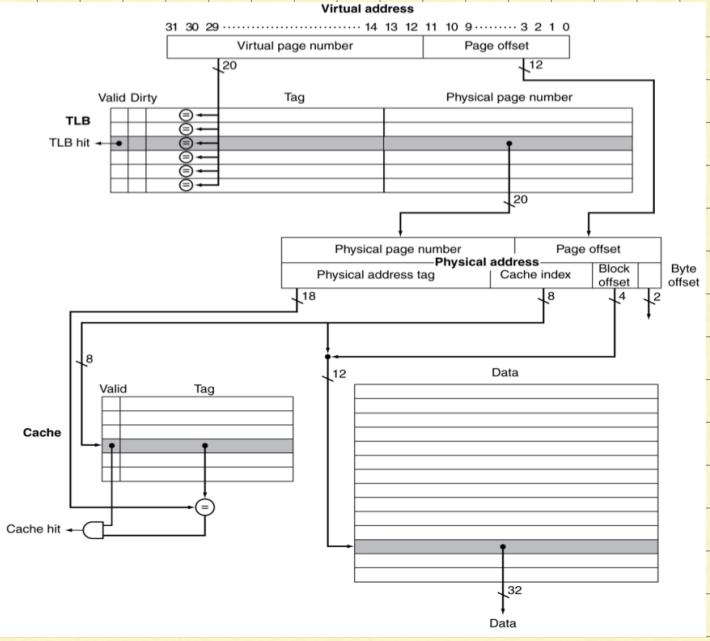
- Since page table is in memory, every memory access by a program requires two memory accesses
  - One to access the page table entry
  - Then the actual memory access
- Can we move the page table to CPU?
  - Yes, use a fast cache in CPU to store recently used PTEs, because access to page tables has good locality
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

TLB and Cache Interaction



TLB
无 | 有
page table 直接找到内存
虚拟 | 物理
找 disk 找物理内存.

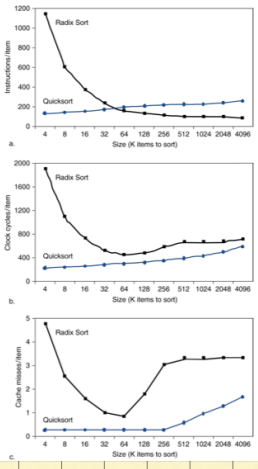
Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

- L1 cache → A cache for a cache
- L2 cache → A cache for disks
- Main memory → A cache for a main memory
- TLB → A cache for page table entries

Interations with Software

- Compare two algorithms: Radix sort & Quicksort
- When size is large,
  - Radix sort has less instructions
  - But quicksort has less clock cycles
  - Because miss rate of radix sort is higher



Software Optimization via Blocking

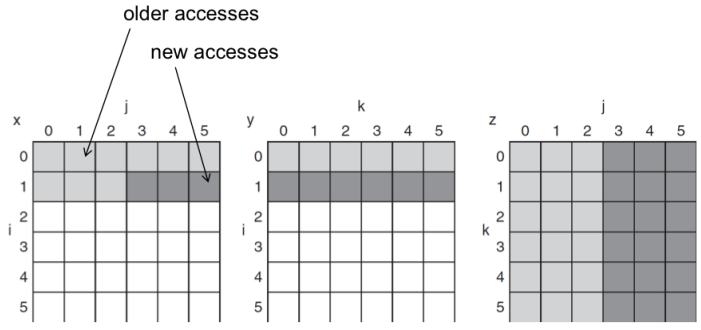
- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
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;
}
```

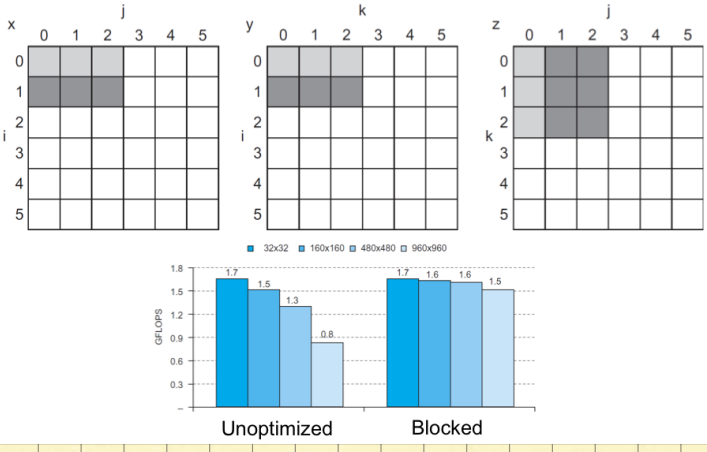
淦4 3 Cache miss

DGEMM Access Pattern

- C, A, and B arrays



Blocked DGEMM Access Pattern



The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate
- Cache and TLB
  - Set-associative, some cache uses direct map

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support
- Cache
  - Both LRU and random is ok

基本上用random

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

对虚拟内存只用write-back

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

cache已经满了

cache还未满
只发生在非fully associative cache里.

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

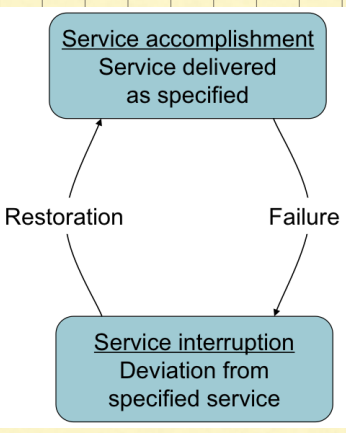
Multilevel On-chip Caches

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

2-level TLB Organization

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

Dependability



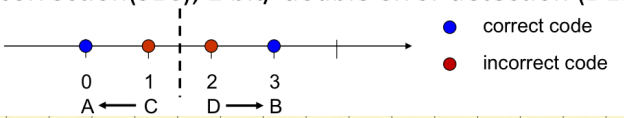
- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: fault detection, fault diagnosis and fault repair

The Hamming SEC Code

- Hamming distance
  - Number of bits that are different between two bit patterns
  - E.g. use 111 to represent 1, use 000 to represent 0, hamming distance (d) is 3, d=3.
- Minimum distance = 2 provides single bit error detection
  - E.g. odd-parity code: 10 → 101, 11 → 110, d = 2
- Minimum distance = 3 provides single error correction(SEC), 2 bit/ double error detection (DED)



Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power of 2 are parity bits (bit 1 2 4 8 are parity bits)
  - Each parity bit checks certain data bits:

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	1	0	0	1	0	1	0	1	0
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	
Parity bit coverate p2		X	X			X	X			X	X	
Parity bit coverate p4				X	X	X	X					X
Parity bit coverate p8								X	X	X	X	X

Decoding SEC

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - E.g.
    - Parity bits = 0000 indicates no error
    - Parity bits = 0101 indicates bit 10 was flipped

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	
	0	1	1	1	0	0	1	0	1	1	1	0	
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		√ 0
Parity bit coverate p2		X	X			X	X			X	X		X 1
Parity bit coverate p4				X	X	X	X					X	√ 0
Parity bit coverate p8								X	X	X	X	X	X 1

SEC / DED Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4
- Decoding:
  - Let H = SEC parity bits
    - H even, p_n even, no error
    - H odd, p_n odd, correctable single bit error
    - H even, p_n odd, error in p_n bit
    - H odd, p_n even, double error occurred
- Note: ECC DRAM uses SEC/DED with 8 bits protecting each 64 bits

Summary

- Cache Performance
  - Mainly depends on miss rate and miss penalty
- To improve cache performance:
  - Fully associative cache
  - Set-associative cache
  - Replacement policy
  - Multilevel cache
- Dependability
  - MTTF, MTTR, reliability, availability
  - Hamming code: SEC/DED code