



Memory Hierarchy and Management

Ingo Sander
ingo@imit.kth.se



Caches

- The access to the main memory is time consuming
- A cache is a small, but fast memory that is located near the CPU to reduce memory access times
- Ideally the processor does only need to access the cache and not the memory

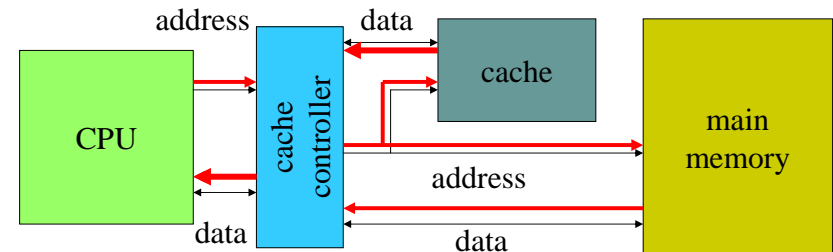
Memory Access Times



Memory Technology	Typical Access Time	\$ per GB in 2004
SRAM	0.5 ns - 5 ns	\$4000 - \$10000
DRAM	50 ns – 70 ns	\$100 - \$200
Magnetic disk	5,000,000 ns – 20,000,000 ns	\$0.5 - \$2

Source: Patterson and Hennessy, 2004

Caches and CPUs



© 2000 Wolf (Morgan Kaufman)



Cache operation

- Many main memory locations are mapped onto one cache entry.
- May have caches for:
 - instructions;
 - data;
 - data + instructions (**unified**).
- Memory access time is no longer deterministic.

© 2000 Wolf (Morgan Kaufman)



Terms

- **Cache hit**: required location is in cache.
- **Cache miss**: required location is not in cache.
- **Working set**: set of locations used by program in a time interval.

© 2000 Wolf (Morgan Kaufman)



Types of misses

- **Compulsory (cold)**: location has never been accessed.
- **Capacity**: working set is too large.
- **Conflict**: multiple locations in working set map to same cache entry.

© 2000 Wolf (Morgan Kaufman)



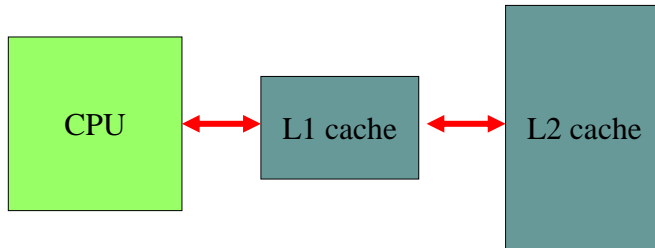
Memory system performance

- h = cache hit rate.
- t_{cache} = cache access time, t_{main} = main memory access time.
- Average memory access time:
 - $t_{\text{av}} = ht_{\text{cache}} + (1-h)t_{\text{main}}$

© 2000 Wolf (Morgan Kaufman)



Multiple levels of cache



© 2000 Wolf (Morgan Kaufman)

September 7, 2004

2B1447 Embedded Systems

9



Multi-level cache access time

- h_1 = cache hit rate.
- h_2 = rate for miss on L1, hit on L2.
- Average memory access time:
 - $t_{av} = h_1 t_{L1} + h_2 t_{L2} + (1 - h_2 - h_1) t_{main}$

© 2000 Wolf (Morgan Kaufman)

September 7, 2004

2B1447 Embedded Systems

10



Replacement policies

- **Replacement policy**: strategy for choosing which cache entry to throw out to make room for a new memory location.
- Two popular strategies:
 - Random.
 - Least-recently used (LRU).

© 2000 Wolf (Morgan Kaufman)

September 7, 2004

2B1447 Embedded Systems

11



Cache organizations

- **Fully-associative**: any memory location can be stored anywhere in the cache (almost never implemented).
- **Direct-mapped**: each memory location maps onto exactly one cache entry.
- **N-way set-associative**: each memory location can go into one of n sets.

© 2000 Wolf (Morgan Kaufman)

September 7, 2004

2B1447 Embedded Systems

12



Cache performance benefits

- Keep frequently-accessed locations in fast cache.
- Cache retrieves more than one word at a time.
 - Sequential accesses are faster after first access.

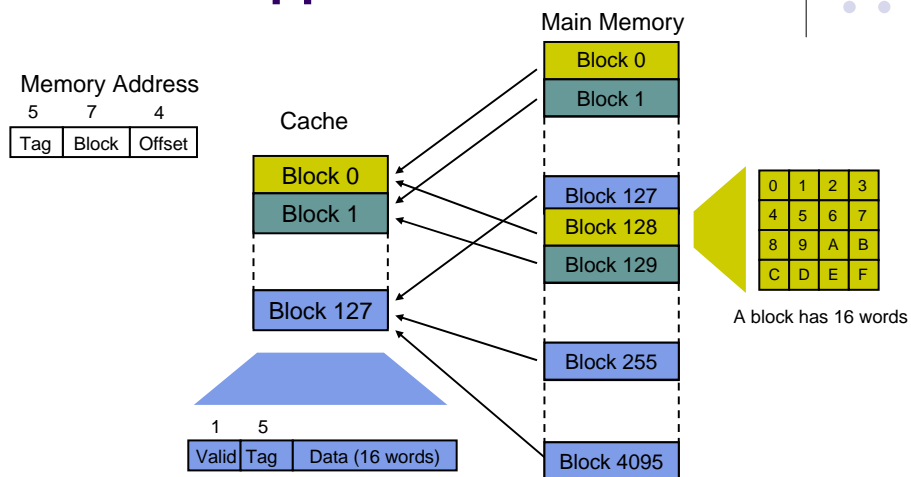
© 2000 Wolf (Morgan Kaufman)



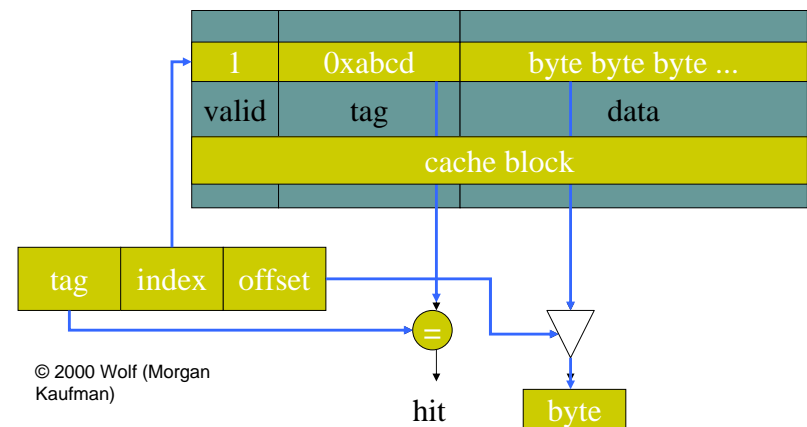
Example Direct Mapped Cache

- Cache has 2K words, organized as 128 blocks of 16 words
- Memory has 64 K words treated as 4K blocks of 16 Words
- Address size is 16 bits
- The direct map technique uses the *modulo* operation to map on a cache block
 - Block 0, 128, 256, ... is mapped on Block 0 in the cache

Example Direct Mapped Cache



Direct-mapped cache



© 2000 Wolf (Morgan Kaufman)



Write operations

- **Write-through**: immediately copy write to main memory.
- **Write-back**: write to main memory only when location is removed from cache.

© 2000 Wolf (Morgan Kaufman)

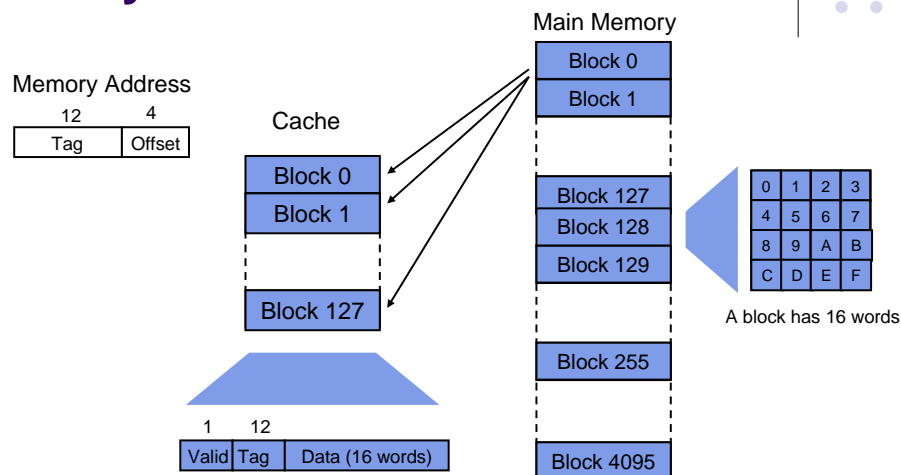


Direct-mapped cache locations

- Many locations map onto the same cache block.
- Conflict misses are easy to generate:
 - Array a[] uses locations 0, 1, 2, ...
 - Array b[] uses locations 1024, 1025, 1026, ...
 - Operation $a[i] + b[i]$ generates conflict misses.

© 2000 Wolf (Morgan Kaufman)

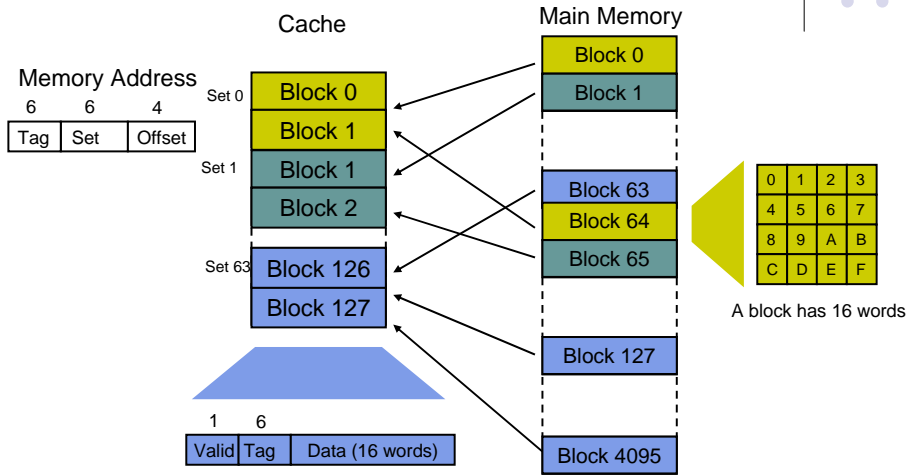
Fully associative cache



Fully associative cache

- There is a complete freedom, where to place a block in the cache
- But all blocks have to be searched for the correct tag pattern
- In order to have an acceptable performance, the tags must be searched in parallel

Example Set-associative-mapped cache



Example: direct-mapped vs. set-associative

address	data
000	0101
001	1111
010	0000
011	0110
100	1000
101	0001
110	1010
111	0100

© 2000 Wolf (Morgan Kaufman)

Direct-mapped cache behavior

• After 001 access:			• After 010 access:		
block	tag	data	block	tag	data
00	-	-	00	-	-
01	0	1111	01	0	1111
10	-	-	10	0	0000
11	-	-	11	-	-

Direct-mapped cache behavior, cont'd.

• After 011 access:			• After 100 access:		
block	tag	data	block	tag	data
00	-	-	00	1	1000
01	0	1111	01	0	1111
10	0	0000	10	0	0000
11	0	0110	11	0	0110

© 2000 Wolf (Morgan Kaufman)

© 2000 Wolf (Morgan Kaufman)

Direct-mapped cache behavior, cont'd.



- After 101 access:

block	tag	data
00	1	1000
01	1	0001
10	0	0000
11	0	0110

- After 111 access:

block	tag	data
00	1	1000
01	1	0001
10	0	0000
11	1	0100

© 2000 Wolf (Morgan Kaufman)

2-way set-associative cache behavior



- Final state of cache (twice as big as direct-mapped):

set	blk 0 tag	blk 0 data	blk 1 tag	blk 1 data
00	1	1000	-	-
01	0	1111	1	0001
10	0	0000	-	-
11	0	0110	1	0100

© 2000 Wolf (Morgan Kaufman)

2-way set-associative cache behavior



- Final state of cache (same size as direct-mapped):

set	blk 0 tag	blk 0 data	blk 1 tag	blk 1 data
0	01	0000	10	1000
1	10	0111	11	0100

© 2000 Wolf (Morgan Kaufman)

Example caches



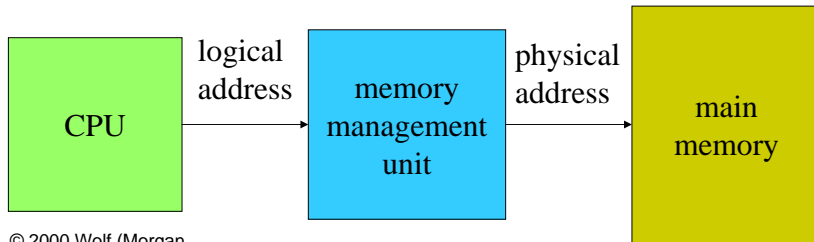
- StrongARM:
 - 16 Kbyte, 32-way, 32-byte block instruction cache.
 - 16 Kbyte, 32-way, 32-byte block data cache (write-back).

© 2000 Wolf (Morgan Kaufman)



Memory management units

- Memory management unit (MMU) translates addresses:



© 2000 Wolf (Morgan Kaufman)



Memory management tasks

- Allows programs to move in physical memory during execution.
- Allows **virtual memory**:
 - memory images kept in secondary storage;
 - images returned to main memory on demand during execution.
- **Page fault**: request for location not resident in memory.

© 2000 Wolf (Morgan Kaufman)



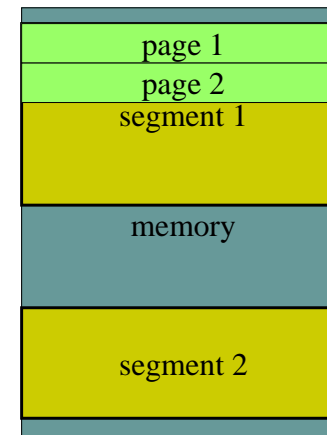
Address translation

- Requires some sort of register/table to allow arbitrary mappings of logical to physical addresses.
- Two basic schemes:
 - **segmented**;
 - **paged**.
- Segmentation and paging can be combined (x86).

© 2000 Wolf (Morgan Kaufman)



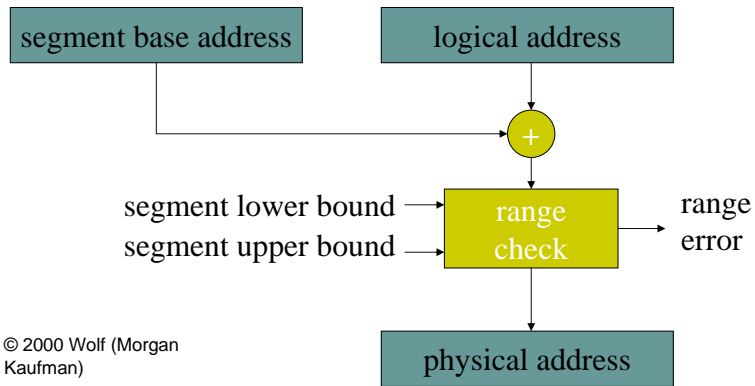
Segments and pages



© 2000 Wolf (Morgan Kaufman)



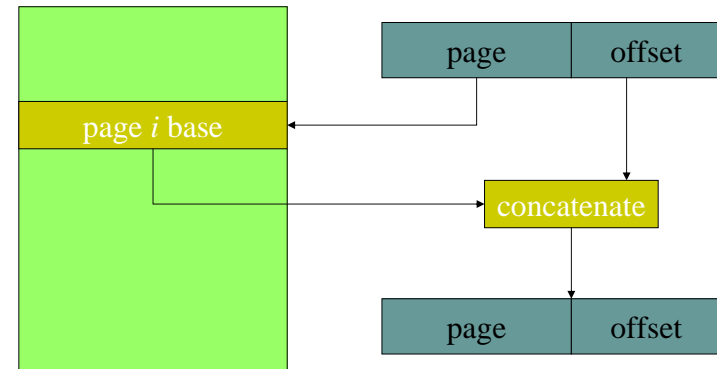
Segment address translation



© 2000 Wolf (Morgan Kaufman)



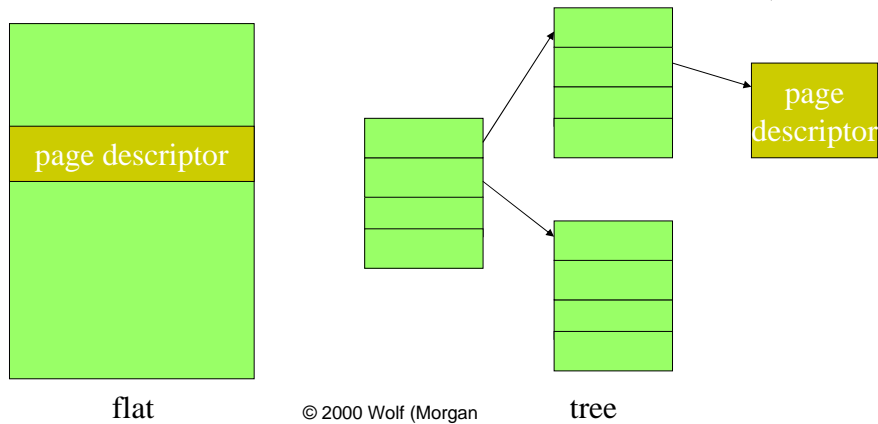
Page address translation



© 2000 Wolf (Morgan Kaufman)



Page table organizations



© 2000 Wolf (Morgan Kaufman)



Caching address translations

- Large translation tables require main memory access.
- **TLB (Translation Lookaside Buffer)**: cache for address translation.
 - Typically small.

© 2000 Wolf (Morgan Kaufman)



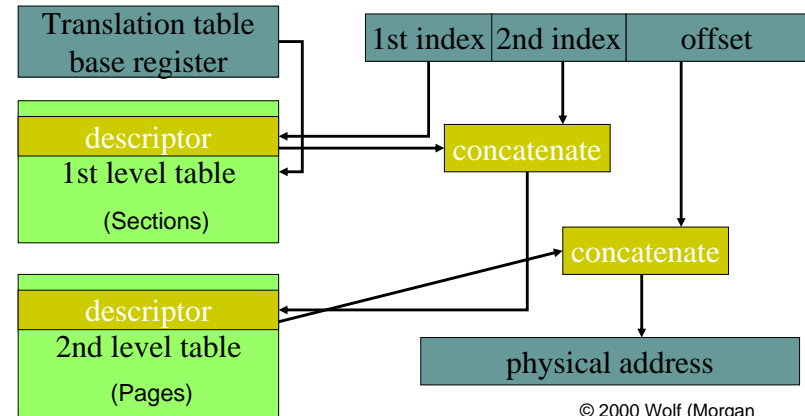
ARM memory management

- Memory region types:
 - section: 1 Mbyte block;
 - large page: 64 kbytes;
 - small page: 4 kbytes.
- An address is marked as section-mapped or page-mapped.
- Two-level translation scheme.

© 2000 Wolf (Morgan Kaufman)



ARM address translation



© 2000 Wolf (Morgan Kaufman)