

Memory Hierarchies II

◆ Last Time

- Locality and Notion of a memory hierarchy
- Implicit management
- Amortized access cost (AMAT)

◆ This Time

- Cache organization
- Dealing with Stores

◆ Reminders and Announcements

- HW #5 assigned (on the web late today), due 3/14/2000 at beginning of class
- Final exam is 3/20/2000, 3-6pm, Peterson Hall 110

CS 141 Chien

1

March 7, 2000

Typical Cache Design

◆ Lowest Address bits select word in line

◆ Next lowest address bits index the cache memory (interleaved mapping)

◆ Remaining high bits form the cache tag values

◆ Example Sizes:

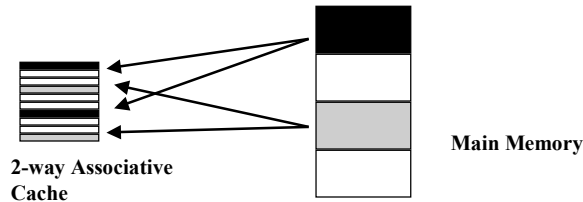
- 32KB total data storage
- 32 bit address space
- 16 byte line size
- tag:index:select = 17:11:4

CS 141 Chien

2

March 7, 2000

Associative Caches



- ◆ What if two important pieces of data for one computation map to the same cache location?
 - thrashing of data in and out of cache
- ◆ Associative caches allow data to be in several places. n-way associative cache allows n places.
- ◆ This flexibility is expensive! (2-way is possible)

CS 141 Chien

3

March 7, 2000

Real Benefits and Limitations of Caches

- ◆ Work well for many applications, but not ALL!
- ◆ Performance depends on application structure, amount of locality present
- ◆ Effectively reduce the average access time.
- ◆ Don't work well if there is no locality (pathological access patterns)
- ◆ Don't work well if a computation isn't run long enough to "fill up the cache" with its data, limitations on multitasking
- ◆ A probabilistic way to improve performance.

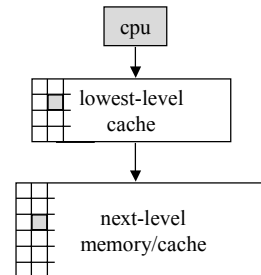
CS 141 Chien

4

March 7, 2000

Cache Fundamentals

- ◆ *cache hit* -- an access where the data is found in the cache.
- ◆ *cache miss* -- an access which isn't
- ◆ *hit time* -- time to access the cache
- ◆ *miss penalty* -- time to move data from further level to closer, then to cpu
- ◆ *hit ratio* -- percentage of time the data is found in the cache
- ◆ *miss ratio* -- (1 - hit ratio)



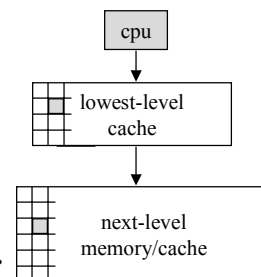
CS 141 Chien

5

March 7, 2000

Cache Fundamentals, cont.

- ◆ *cache block size or cache line size*-- the amount of data that gets transferred on a cache miss.
- ◆ *instruction cache* -- cache that only holds instructions.
- ◆ *data cache* -- cache that only caches data.
- ◆ *unified cache* -- cache that holds both.



CS 141 Chien

6

March 7, 2000

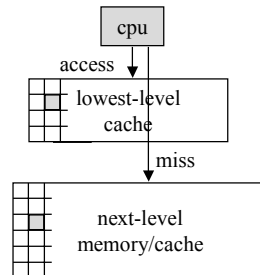
Cacheing Issues

On a memory access -

- ◆ How do I know if this is a hit or miss?

On a cache miss -

- ◆ where to put the new data?
- ◆ what data to throw out?
- ◆ how to remember what data this is?

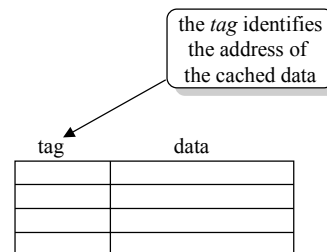


A simple cache

address string:

```

4      0000100
8      00001000
12     00001100
4      00000100
8      00001000
20     00010100
4      00000100
8      00001000
20     00010100
24     00011000
12     00001100
8      00001000
4      00000100
  
```



4 entries, each block holds one word, any block can hold any word.

- ◆ A cache that can put a line of data anywhere is called *fully associative*
- ◆ The most popular replacement strategy is *LRU* (least recently used).

A simpler cache

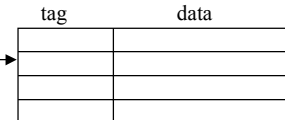
address string:

```

4      00000100
8      00001000
12     00001100
4      00000100
8      00001000
20     00010100
4      00000100
8      00001000
24     00011000
12     00001100
8      00001000
4      00000100
    
```

an index is used
to determine
which line an address
might be found in

00000100



4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- ◆ A cache that can put a line of data in exactly one place is called *direct-mapped*
- ◆ Advantages/disadvantages vs. fully-associative.

CS 141 Chien

9

March 7, 2000

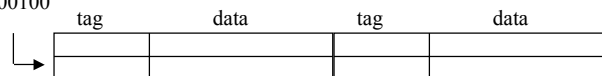
A set-associative cache

address string:

```

4      00000100
8      00001000
12     00001100
4      00000100
8      00001000
20     00010100
4      00000100
8      00001000
24     00011000
12     00001100
8      00001000
4      00000100
    
```

00000100



4 entries, each block holds one word, each word in memory maps to one of a set of n cache lines

- ◆ A cache that can put a line of data in exactly n places is called *n -way set-associative*.
- ◆ The cache lines that share the same index are a cache *set*

CS 141 Chien

10

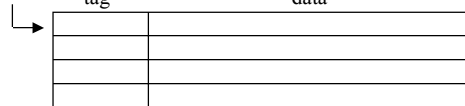
March 7, 2000

Longer Cache Blocks

address string:

4 00000100
 8 00001000
 12 00001100
 4 00000100
 8 00001000
 20 00010100
 4 00000100
 8 00001000
 20 00010100
 24 00011000
 12 00001100
 8 00001000
 4 00000100

00000100



4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

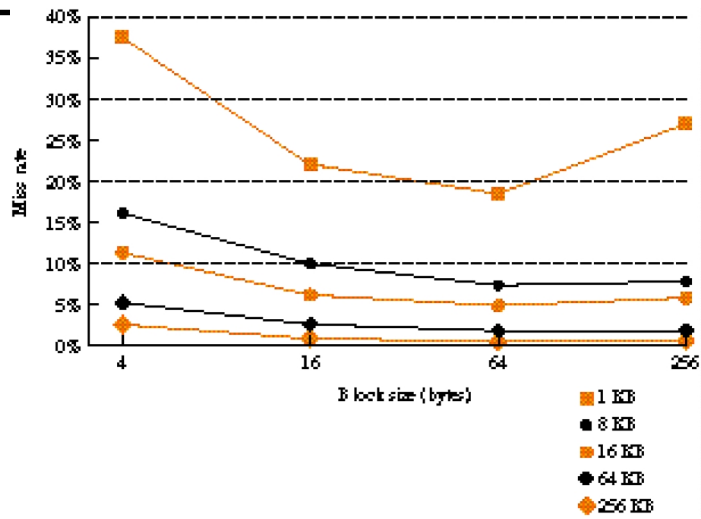
- ◆ Large cache blocks take advantage of *spatial locality*.
- ◆ Too large of a block size can waste cache space.
- ◆ Longer cache blocks require less tag space

CS 141 Chien

11

March 7, 2000

Block Size and Miss Rate



CS 141 Chien

12

March 7, 2000

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size = ?

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

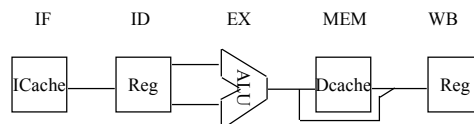
CS 141 Chien

13

March 7, 2000

Handling a Cache Access

1. Use index and tag to access cache and determine hit/miss.
2. If hit, return requested data.
3. If miss, select a cache block to be replaced, and access memory or next lower cache (possibly stalling the processor).
 - load entire missed cache line into cache
 - return requested data to CPU (or higher cache)
4. If next lower memory is a cache, goto step 1 for that cache.



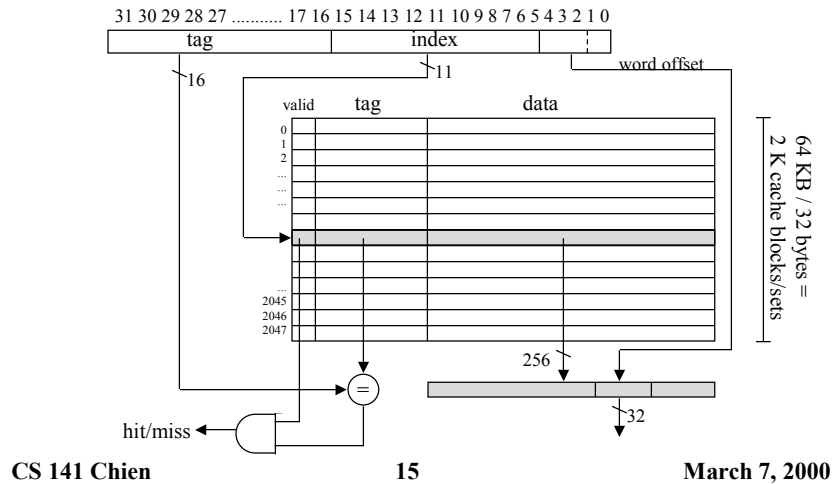
CS 141 Chien

14

March 7, 2000

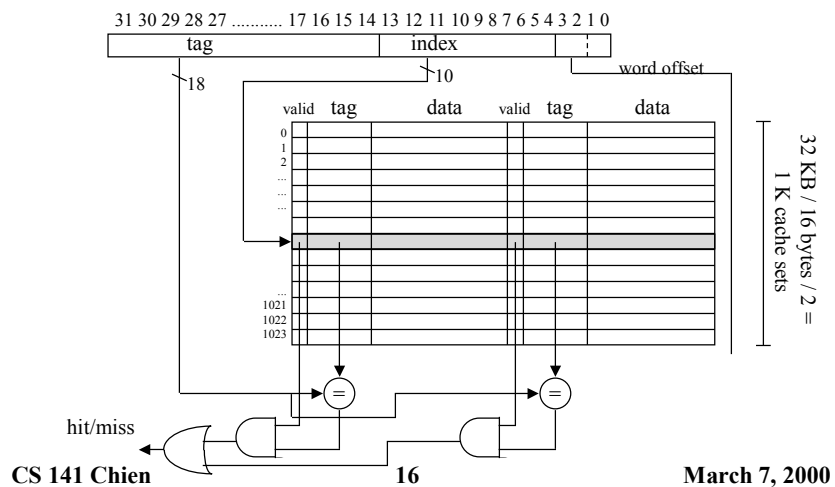
Accessing a Sample Cache

- ◆ 64 KB cache, direct-mapped, 32-byte cache block size



Accessing a Sample Cache

- ◆ 32 KB cache, 2-way set-associative, 16-byte block size



Associative Caches

- ◆ Higher hit rates, but...
- ◆ longer access time (longer to determine hit/miss, more muxing of outputs)
- ◆ more space (longer tags)
 - 16 KB, 16-byte blocks, dm, tag = ?
 - 16 KB, 16-byte blocks, 4-way, tag = ?

Dealing with Stores

- ◆ Stores must be handled differently than loads, because...
 - they don't necessarily require the CPU to stall.
 - they change the content of cache/memory (creating memory *consistency* issues)
 - may require a load and a store to complete

Policy decisions for stores

- ◆ **Keep memory and cache identical?**
 - *write-through* => all writes go to both cache and main memory
 - *write-back* => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- ◆ **Make room in cache for store miss?**
 - *write-allocate* => on a store miss, bring written line into the cache
 - *write-around* => on a store miss, ignore cache

Dealing with stores

- ◆ **On a store hit, write the new data to cache. In a *write-through* cache, write the data immediately to memory. In a *write-back* cache, mark the line as dirty.**
- ◆ **On a store miss, initiate a cache block load from memory for a *write-allocate* cache. Write directly to memory for a *write-around* cache.**
- ◆ **On any kind of cache miss in a *write-back* cache, if the line to be replaced in the cache is dirty, write it back to memory.**

Cache Performance

$$\text{TCPI} = \text{BCPI} + \text{MCPI}$$

- where TCPI = Total CPI;
- BCPI = base CPI, which means the CPI assuming perfect memory;
- MCPI = the memory CPI, the number of cycles (per instruction) the processor is stalled waiting for memory.

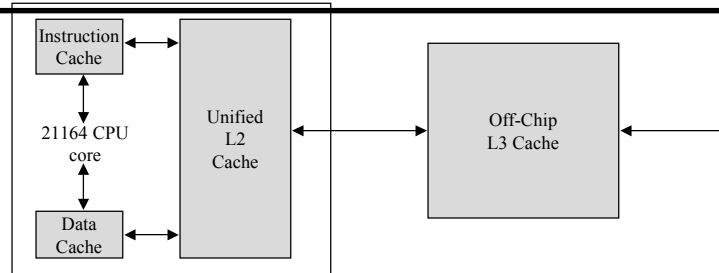
$$\text{MCPI} = \text{accesses/instruction} * \text{miss rate} * \text{miss penalty}$$

- this assumes we stall the pipeline on both read and write misses, that the miss penalty is the same for both, that cache hits require no stalls.

Cache Performance

- ◆ Instruction cache miss rate of 4%, data cache miss rate of 9%, BCPI = 1.0, 20% of instructions are loads and stores, miss penalty = 12 cycles, TCPI = ?
- ◆ Unified cache, 25% of instructions are loads and stores, BCPI = 1.2, miss penalty of 10 cycles. If we improve the miss rate from 10% to 4% (e.g. with a larger cache), how much do we improve performance?
- ◆ BCPI = 1, miss rate of 8% overall, 20% loads, miss penalty 20 cycles, never stalls on stores. What is the speedup from doubling the cpu clock rate?

Example -- DEC Alpha 21164 Caches



- ◆ ICache and DCache -- 8 KB, DM, 32-byte lines
- ◆ L2 cache -- 96 KB, 3-way SA, 32-byte lines
- ◆ L3 cache -- 1 MB, DM, 32-byte lines

CS 141 Chien

23

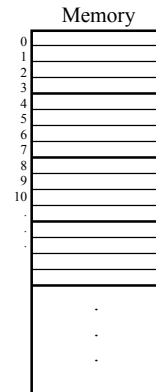
March 7, 2000

Cache Alignment

memory address



- ◆ The data that gets moved into the cache on a miss are all data whose addresses share the same tag and index (regardless of which data gets accessed first).
- ◆ This results in
 - no overlap of cache lines
 - easy mapping of addresses to cache lines (no additions)
 - data at address X always being present in the same location in the cache block (at byte $X \bmod \text{blocksize}$) if it is there at all.
- ◆ Think of memory as organized into cache-line sized pieces (because in reality, it is!).

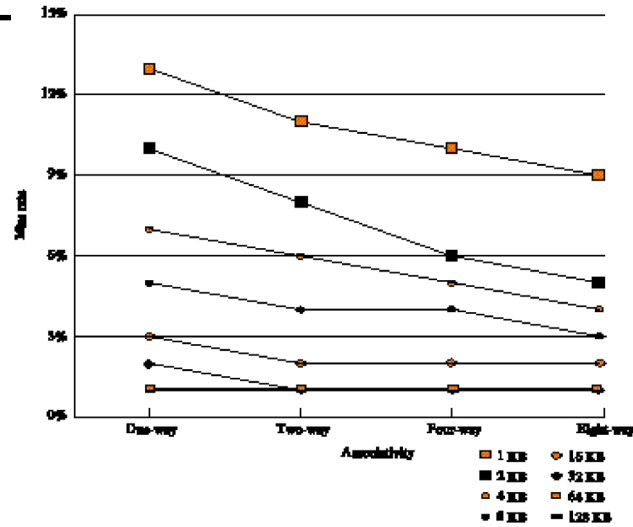


CS 141 Chien

24

March 7, 2000

Cache Associativity



CS 141 Chien

25

March 7, 2000

Summary

- ◆ Caches give illusion of a large, cheap memory with the access time of a fast, expensive memory.
- ◆ Caches take advantage of memory locality, specifically temporal locality and spatial locality.
- ◆ Cache design presents many options (block size, cache size, associativity, write policy) that an architect must combine to minimize miss rate and access time to maximize performance.

CS 141 Chien

26

March 7, 2000