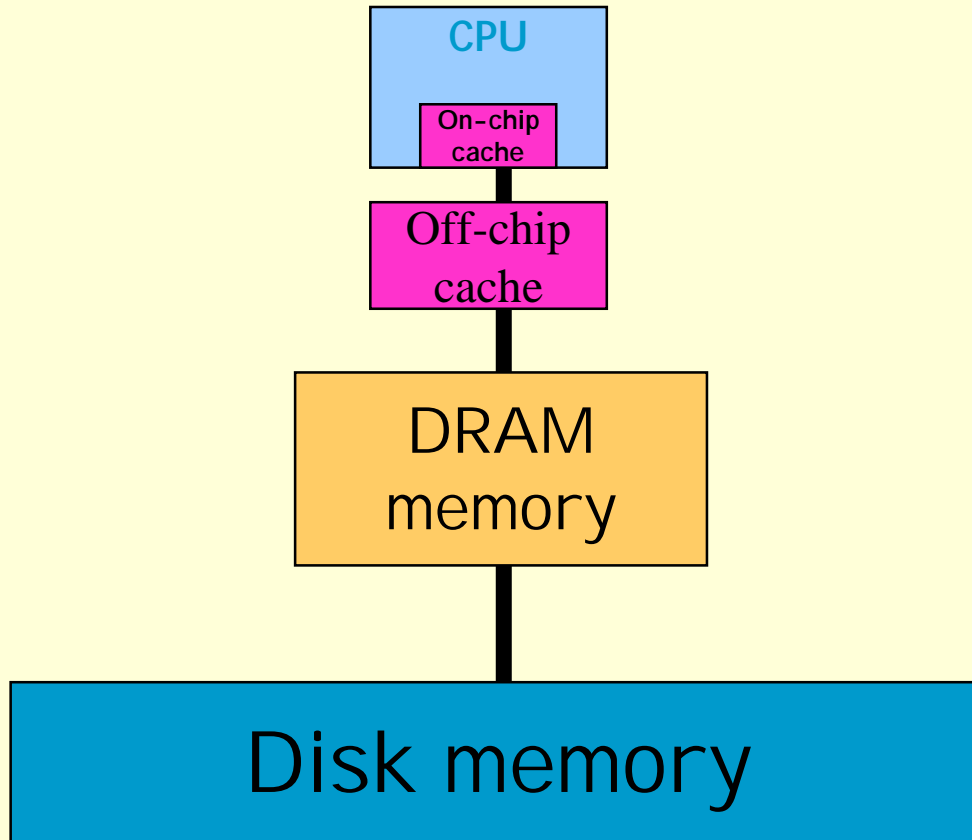


# Caches, part II

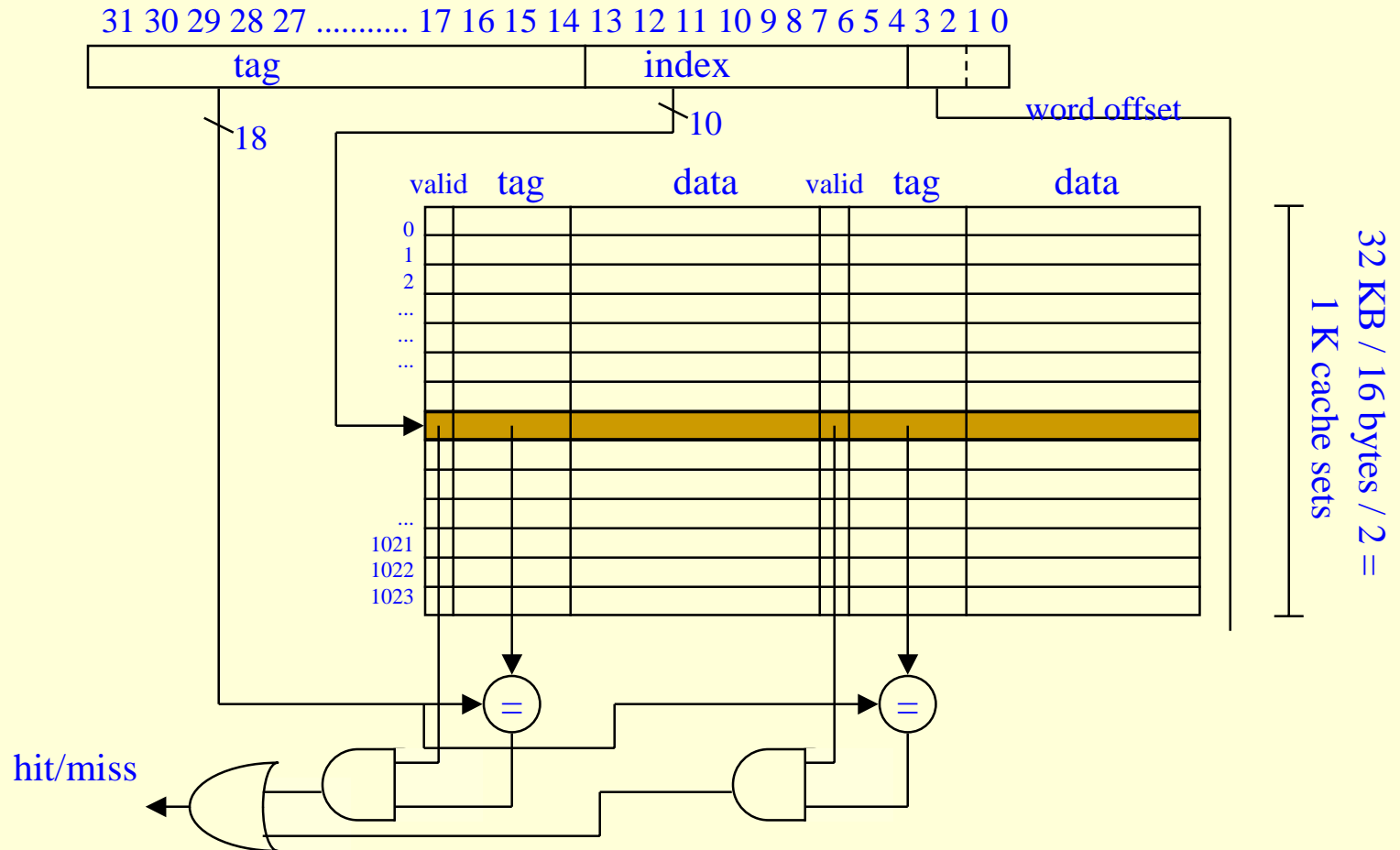


# Details

- X • What bits should we use for the index?
- X • How do we know if a cache entry is empty?
  - Are stores and loads treated the same?
  - What if a word overlaps two cache lines??
- X • How does this all work, anyway???

# Accessing a Sample Cache

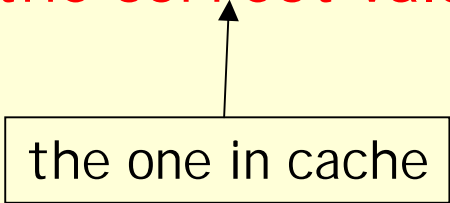
- 32 KB cache, 2-way set-associative, 16-byte block size



# 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

Creates a memory consistency question ... how do you ensure memory gets the correct value?



the one in cache

# Policy decisions for stores

- Do you keep memory and cache identical?
  - write-through cache: all writes go to both cache and main memory
  - write-back cache: writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
- Do you make room in cache for store miss?
  - write-allocate: on a store miss, bring target line into the cache.
  - write-around: on a store miss, ignore cache

# Dealing with stores

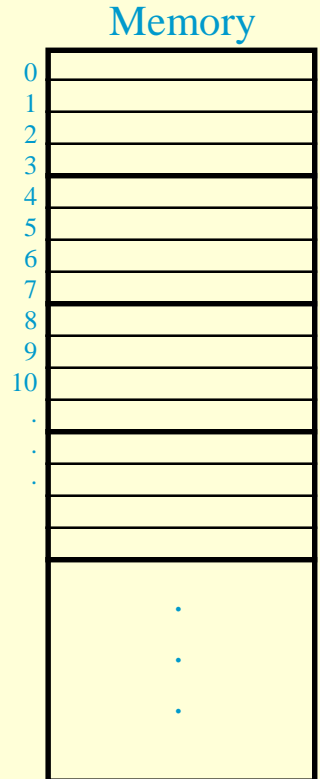
- On a store hit, write the new data to cache.
  - In a *write-through* cache, write the data immediately to memory.
  - In a *write-back* cache
    - Mark the line as dirty.
      - means cache has correct value, but memory doesn't
    - On any cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.
- On a store miss,
  - In a *write-allocate* cache,
    - Initiate a cache block load from memory.
  - In a *write-around* cache,
    - Write directly to memory.

# Cache Alignment

memory address: 

tag	index	offset
-----	-------	--------

- A cache line is all the data whose address share the tag and index.
  - Example: Suppose offset is 5 bits,
    - Bytes 0-31 form the first cache line
    - Bytes 32-63 form the second, etc.
  - When you load location 40, cache gets Bytes 32-63
- This results in
  - no overlap of cache lines
  - easy to find if address is in cache (no additions)
  - easy to find the data within the cache line
- Think of memory as organized into cache-line sized pieces (because in reality, it is!)



# Can a word overlap two cache lines?

- Depends on the architecture ...
  - Some require words to be word-aligned ...
    - and double words to be double-word aligned
    - Every load and store is to a single cacheline
  - Others allow data to span cache lines
    - Can be two cache misses for a single reference!
    - Requires more shifting logic
    - But allows more compact data structures.

# Cache Vocabulary

- miss penalty: extra time required on a cache miss
- hit rate: fraction of accesses that are cache hits
- miss rate:  $1 - \text{hit rate}$

# A Performance Model

- $TCPI = BCPI + MCPI$ 
  - $TCPI = \text{Total CPI}$
  - $BCPI = \text{Base CPI} = \text{CPI assuming perfect memory}$
  - $MCPI = \text{Memory CPI} = \text{cycles waiting for memory per instruction}$
- $BCPI = \text{peak CPI} + PSPI + BSPI$ 
  - $PSPI = \text{pipeline stalls per instruction}$
  - $BSPI = \text{branch hazard stalls per instruction}$
- $MCPI = \text{accesses/instruction} * \text{miss rate} * \text{miss penalty}$ 
  - 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.
  - If the miss penalty or miss rate is different for I-cache and D-cache (which is common), then
$$MCPI = InstMR * InstMP + DataAccesses/inst * DataMR * DataMP$$

# 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?

# Three types of cache misses

- Compulsory misses
  - number of misses needed to bring every cache line referenced by program into an infinitely large cache.
- Capacity misses
  - number of misses in a fully associative cache of the same size as the cache in question minus the compulsory misses.
- Conflict misses
  - number of misses in actual cache minus number there would be in a fully-associative cache of the same size.

Total misses = (Compulsory + Capacity + Conflict) misses

- Example: 4-word direct-mapped, 1 word per cacheline
  - Reference sequence: 4, 8, 12, 4, 8, 20, 4, 8, 20, 24, 12, 8, 4
    - Compulsory misses:
    - Capacity misses:
    - Conflict misses:

# So, then, how do we decrease...

- Compulsory misses?
- Capacity misses?
- Conflict misses?

# LRU replacement algorithms

- Not needed for direct-mapped caches
- Requires one bit per set for 2-way set-associative, 8 bits per set for 4-way (2 bits per entry), 24 bits per set for 8-way, etc.
- *Can be approximated with  $k-1$  bits per set*
  - *One bit says which half was less recently used*
  - *Each half has one bit says which quarter is LRU, ...*
- Cheaper approach is to use random replacement within a set. Miss rate is about 10% higher than LRU.
- Highly associative caches (like page tables, which we'll get to) use a different approach.

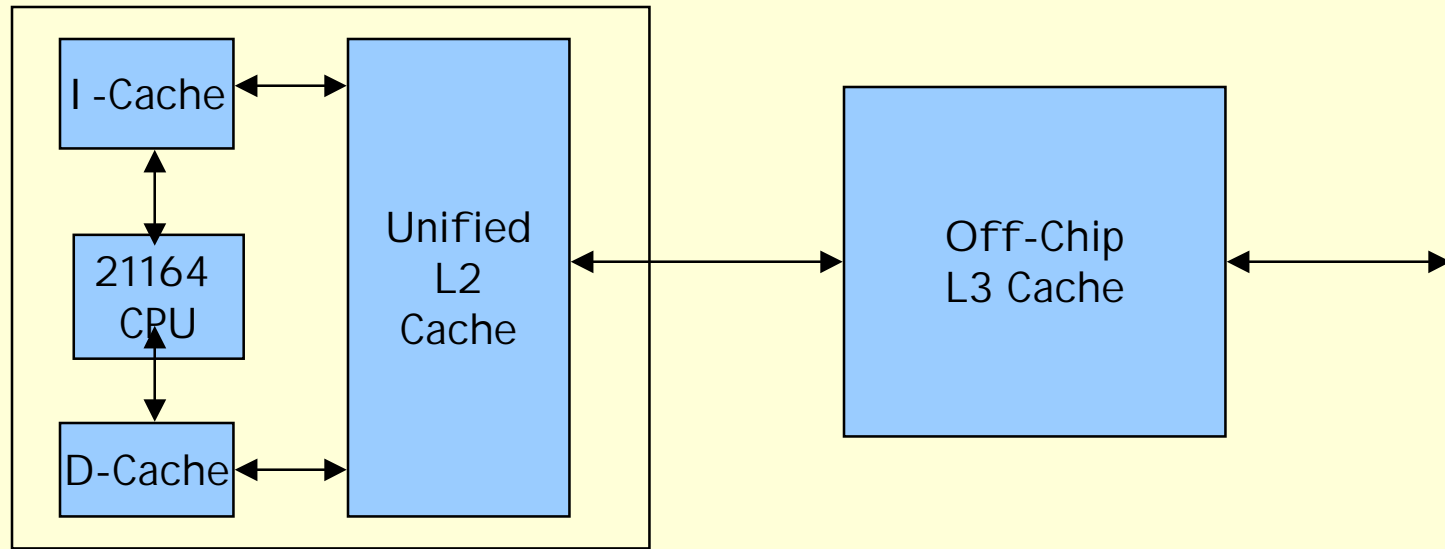
# Caches in Current Processors

- Often direct mapped level 1 cache (closest to CPU), associative further away
- Split I and D level 1 caches (for throughput rather than miss rate), unified further away.
- Write-through and write-back are both common, but never write-through all the way to memory.
- Cache line size at least 32, getting larger.

*Usually cache is non-blocking*

- *processor doesn't stall on a miss, but only on the use of a miss (if even then)*
- *this means the cache must be able to handle multiple outstanding accesses.*

# Example -- DEC Alpha 21164 Caches



I Cache and DCache -- 8 KB, Direct Mapped, 32-Byte lines

L2 cache -- 96 KB, 3-way Set Associative, 32-Byte lines

L3 cache -- 1 MB, Direct Mapped, 32-byte lines (but different L3's can be used)

# Cache Review

memory address: 

tag	index	offset
-----	-------	--------

DEC Alpha 21164's L2 cache:

96 KB, 3-way set associative, 32-Byte lines

64 bit addresses

How many "offset" bits?

How many "index" bits?

How many "tag" bits?

Draw cache picture – how do you tell if it's a hit?

# Computer of the Day

- The first Macintosh
- “1984” Superbowl advertisement shown only once
  - see [www.apple-history.com/quickgallery.html](http://www.apple-history.com/quickgallery.html)
- Motorola 68000 processor – 8 MHz, 128 KB memory
- 13.5" x 9.5" x 11". 16.5 pounds
- Had mouse, small b/w monitor, built-in floppy drive, and backpack. Cost \$2,495.
- First commercial computer with a GUI .
  - Incredibly easy to use compared to DOS, ...**