The 3 C’s

Taxonomy of misses --- misses can be classified as:

**Compulsory** (or cold)
- caused by the first reference to a block
- how to eliminate: increase block size

**Capacity**
- the cache is not big enough to hold all the blocks that were referenced
- how to eliminate: increase cache size

**Conflict**
- two blocks map to the same set & all blocks in the set are valid
- how to eliminate: increase associativity

Design Trade-offs

**Cache size**
- the bigger the cache
  - the higher the hit ratio (fewer capacity misses)
  - the longer the access time

Calculating cache size
- cache size: # bytes of data
  - different than the number of bits of memory needed to build a cache (data, tags, state)
- # bytes/block * associativity * # sets
- example: 64KB cache, 32B blocks, 2-way set-associative
  - what is the cache size?
  - how many bits does it take to implement the cache?
Design Tradeoffs

Block size

large blocks
+ can take better advantage of spatial locality (& reduce compulsory misses)
+ have better transfer efficiency when transferring data from memory
  • less block transfer overhead per block
    the overhead: 1 address transmission & 1 memory lookup for a block of any size
    • amortize the overhead by transferring many words in 1 memory request
+ have less tag overhead
  • 64KB, direct-mapped, 4B blocks: $2^{14}$ tags
  • 64KB, direct-mapped, 16B blocks: $2^{12}$ tags
- have larger transfer latency
- might not access all the bytes in the block
  • have transferred the words from memory for nothing
- might increase capacity misses because bigger blocks means fewer blocks in the cache

Design Tradeoffs

Associativity

with a larger associativity:
+ higher hit ratio
  • more places to put data for a particular index value
    (reduces conflict misses)
  • more important for small caches
    gains diminish as increase associativity
- slightly larger hardware cost
  • comparator for each block in the set
- increase in tag bits (decrease index bits)
- slightly larger cache access time (a MUX or a larger MUX)
- need block replacement hardware
Design Tradeoffs

Block replacement: which block you replace on a cache miss

- LRU (least recently used)
  - toggle between bit values for 2-way
  - counters for more associativity
- random
  - for example, low bits of the cycle counter
- little impact on performance

Design Tradeoffs

Memory update policy

- write through
  - update memory on each store (one word is written to memory)
  - memory is always coherent (memory has the same value as the cache)
  - performance depends on the # of writes

Hiding write latency with write through

- write buffer:
  - contains data & its address until data is written to memory
  - CPU writes data into the cache & write buffer & then continues execution
  - memory is updated when the processor-memory bus is free
  - CPU must stall if the write buffer is full
  - must check write buffer on a cache miss: why?
Design Tradeoffs

Memory update policy
- write back
  - update memory when the block is replaced from the cache
  + performance depends on the # of block replacements
  - add a dirty bit to each block’s state
    - clear dirty bit when a block is read into the cache (clean block)
    - set dirty bit when the block is updated (dirty block)
    - on a block replacement, if the dirty bit is set, write the block to memory
  - must do a tag check before the write to make sure this is the right block
    - if the block is one word, write-through can just write
  - memory is not coherent
    - must flush blocks from the cache before writing to disk
  - slightly longer cycle than write-through
    - must check the tag before updating the cache

Hiding write latency with write back
- the dirty block can be buffered while the new block is being fetched
- replaced after control returns to the CPU

Design Tradeoffs

Cache contents
- separate instruction & data caches
  + separate access ⇒ double the bandwidth
    - not have to stall the fetch if a load is at the same time
    + shorter access time than 1 larger cache for both
    + different configurations for I & D: *why?*

- implemented as the first-level, on-chip cache
- unified cache
  + lower miss rate if same size as the sum of the separate caches
    - more flexibility in where blocks can be placed
  + less cache controller* hardware since only 1 cache
    (* logic that implements cache accesses, hits, misses)
**Cache Performance**

**Hit (miss) rate**

\[
\text{Hit (miss) rate} = \frac{\text{# references that hit (miss)}}{\text{# references}}
\]

- miss rate = 1 - hit rate
- measures how well the cache functions
- useful for understanding cache behavior relative to the number of references
- intermediate (component) metric

**Average memory access time**

\[
\text{Average memory access time} = \text{HitTime} + \text{MissRatio} \times \text{MissPenalty}
\]

- always incur hit time because either:
  - a reference hits or
  - on a miss you access the cache again after the block is brought into the cache (this time it does hit!)
- (rough) average time it takes to do a memory reference
- performance of the memory system, including factors that depend on the implementation (the miss penalty)
- intermediate (component) metric

**Miss penalty**

\[
\text{Miss penalty} = \text{send the address to memory} + \text{access the memory} + \text{transfer the data (block size (in words) \times transfer time for 1 word)} \quad \text{(assuming bus is 1 word wide)}
\]
Execution Time

Including the memory system in execution time

CPU time = (CPU execution cycles + memory stall cycles) × cycle time
(hit time is included in CPU execution cycles)

\[
\text{memory stall cycles} = \frac{\text{memory accesses}}{\text{program}} \times \text{miss rate} \times \text{miss penalty}
\]
\[
= \frac{\text{instructions}}{\text{program}} \times \frac{\text{misses}}{\text{instruction}} \times \text{miss penalty}
\]

- might have different miss ratios for data & instructions
  \[
  \frac{\text{misses}}{\text{instruction}} = \text{instruction miss rate} + \left( \text{data miss rate} \times \frac{\text{data references}}{\text{instruction}} \right)
  \]
- memory stall cycles are measured in CPU cycles
  - the faster the clock rate, the larger the miss penalty
  - the lower the CPI, the greater the impact of memory stall cycles

Look at the performance examples on pp. 565-568