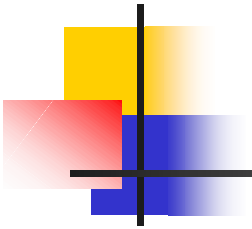


Computer Architecture
Lecture 14

Memory Hierarchy

Reference material:

Computer Organization and Design: The Hardware and Software Interface by Patterson and Hennessy, 4th Edition, Morgan and Kauffman Publications, 2008 (chap-07)

Computer System Architecture by Morris Mano, 3rd ed. (chap-12)

Recommended YouTube links

- https://www.youtube.com/watch?v=LzsiFYVMqi8
- https://www.youtube.com/watch?v=Ref231Eg9QE
- https://www.youtube.com/watch?v=7IVE6vXqEoc

Replacement Algorithms

- With direct mapping, it is no need because each block has a pre-defined location where it can be placed and it replaces the old one.
- With associative and set-associative mapping, a replacement algorithm is needed in order to determine which block to replace:
  - First-in-first-out (FIFO).
  - Least-recently used (LRU) – replace the block that has been in the cache longest with not reference to it.
  - Lest-frequently used (LFU) – replace the block that has experienced the fewest references.
  - Random.

Write Policy

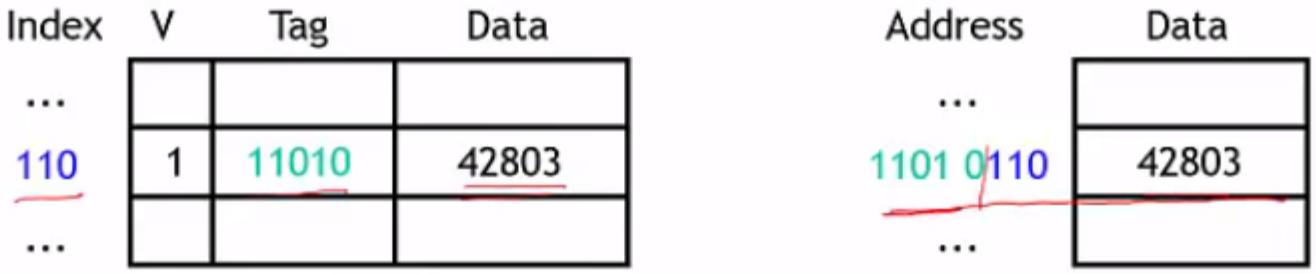
- **The problem:**
  - How to keep cache content and main memory content consistent without losing too much performance?
- **Write through:**
  - All write operations are passed to main memory: If the addressed location is currently in the cache, the cache is updated so that it is coherent with the main memory.
  - For writes, the processor always slows down to main memory speed.
  - Since the percentage of writes is small (ca. 15%), this scheme doesn't lead to large performance reduction.

Write Policy

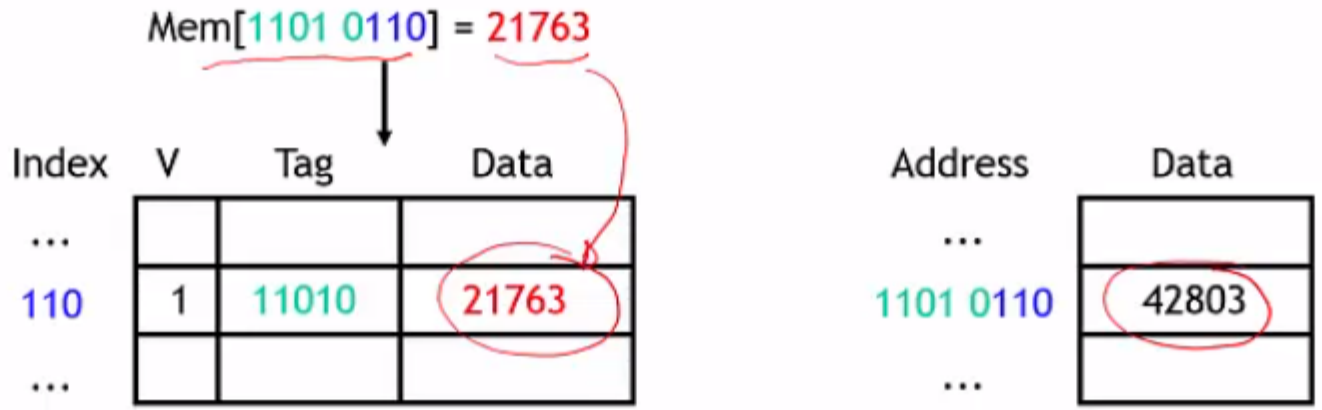
- **Write through with buffered write:**
  - The same as write-through, but instead of slowing the processor down by writing directly to main memory, the write address and data are stored in a high-speed write buffer; the write buffer transfers data to main memory while the processor continues its task.
  - Higher speed, but more complex hardware.
- **Write back:**
  - Write operations update only the cache memory which is not kept coherent with main memory. When the slot is replaced from the cache, its content has to be copied back to memory.
  - Write-back (or copy back) writes only to cache but sets a “dirty bit” in the block where write is performed.
  - When a block with dirty bit “on” is to be overwritten in the cache, it is first written to the memory.
  - Good performance (usually several writes are performed on a cache block before it is replaced), but more complex hardware is needed.

Writing to a cache

- Writing to a cache raises several additional issues
- First, let's assume that the address we want to write to is already loaded in the cache. We'll assume a simple direct-mapped cache:

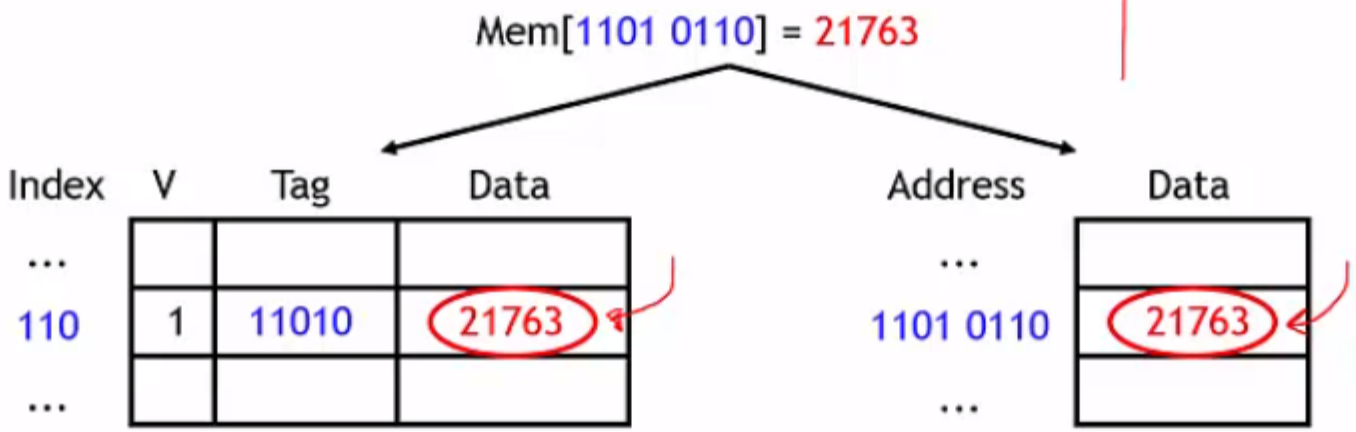


- If we write a new value to that address, we can store the new data in the cache, and avoid an expensive main memory access [but inconsistent]



Write-through caches

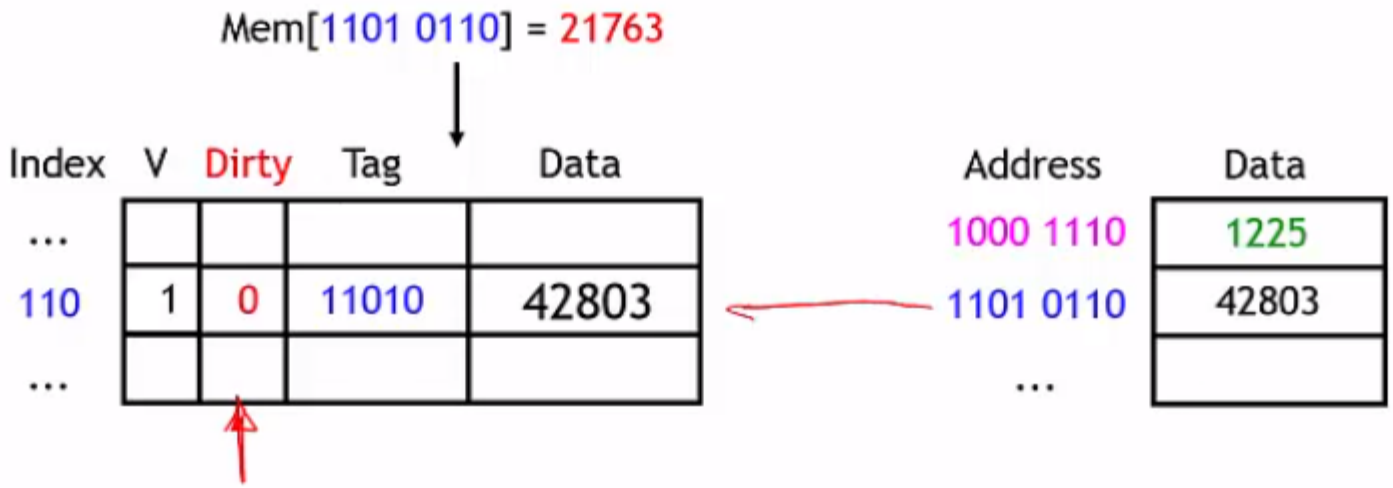
- A write-through cache solves the inconsistency problem by forcing all writes to update both the cache *and* the main memory



- This is simple to implement and keeps the cache and memory consistent
- Why is this not so good?

Write-back caches

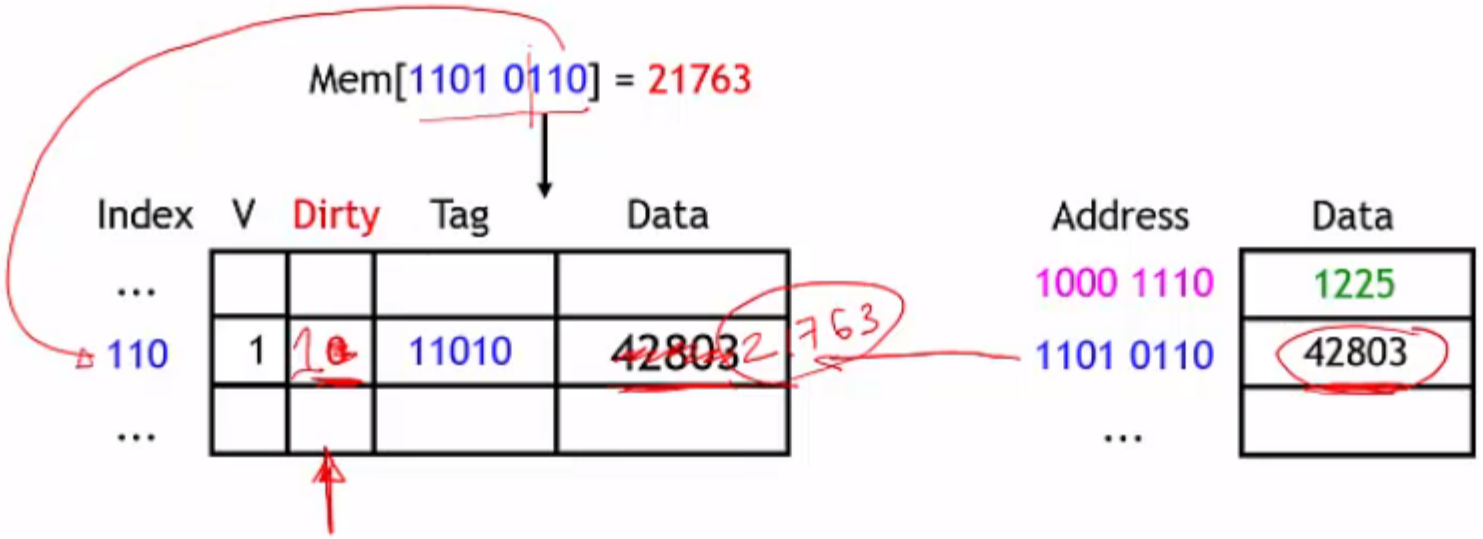
- In a write-back cache, the memory is not updated until the cache block needs to be replaced (*e.g.*, when loading data into a full cache set)
- For example, we might write some data to the cache at first, leaving it inconsistent with the main memory as shown before
  - The cache block is marked "dirty" to indicate this inconsistency



- Subsequent reads to the same memory address will be serviced by the cache, which contains the correct, updated data

Write-back caches

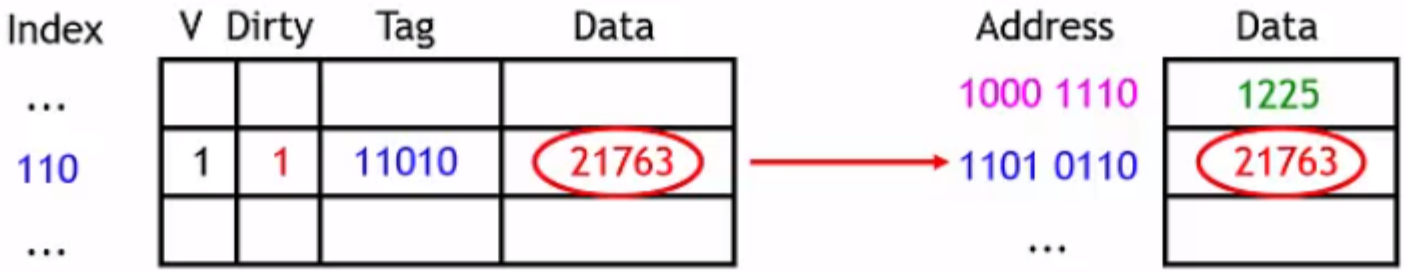
- In a write-back cache, the memory is not updated until the cache block needs to be replaced (*e.g.*, when loading data into a full cache set)
- For example, we might write some data to the cache at first, leaving it inconsistent with the main memory as shown before
  - The cache block is marked "dirty" to indicate this inconsistency



- Subsequent reads to the same memory address will be serviced by the cache, which contains the correct, updated data

Finishing the write back

- We don't need to store the new value back to main memory unless the cache block gets replaced
- e.g. on a read from Mem[1000 1110], which maps to the same cache block, the modified cache contents will first be written to main memory



- Only then can the cache block be replaced with data from address 142