Corso di Laurea in Informatica
Architettura degli elaboratori
a.a. 2025-2026

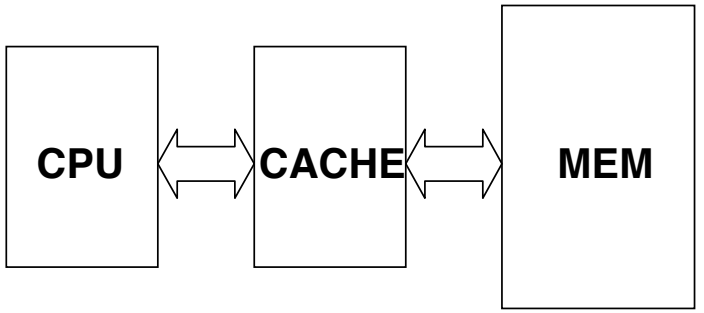
Architettura degli Elaboratori 2025-2026

Gestione della cache

Prof. Elisabetta Fersini
elisabetta.fersini@unimib.it

Gestione della cache

4 decisioni da prendere



1. Dove posizionare un blocco ?
2. Come reperire un blocco ?

} **Tecnica di indirizzamento**

3. Quale blocco sostituire in corrispondenza di un miss ?

Algoritmo di sostituzione

4. Come gestire un'operazione di scrittura ?

Strategia di aggiornamento

Algoritmi per la sostituzione di blocchi

- Nella cache ad accesso diretto: se il blocco di memoria è mappato in una linea di cache già occupata (conflict miss), si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco
- Quale blocco sostituire in caso di (capacity) miss nelle cache set- o fully- associative?
  - In caso di cache completamente associativa: ogni blocco è un potenziale candidato per la sostituzione
  - In caso di cache set-associativa a N vie: bisogna scegliere tra gli N blocchi del set

Algoritmi per la sostituzione di blocchi

- Politica di sostituzione **Random** – Scelta casuale
- Politica di sostituzione **Least Recently Used** (**LRU**)
  - Sfruttando la località temporale, LRU rimuove l'**oggetto usato meno recentemente**
  - Ad ogni blocco si associa un timestamp (o contatore) che viene utilizzato per tenere traccia dell'accesso a ciascun elemento
  - L'elemento con il timestamp più vecchio (quello a cui si è avuto accesso meno di recente) viene eliminato quando la cache è piena.

LRU

- **Cache Fully Associative**
  - **Sostituzione**: quando la cache è piena e si deve inserire un nuovo blocco, si applica una politica di sostituzione.
  - **Come funziona:**
    - Si tiene traccia dell'ordine d'uso dei blocchi.
    - Quando è necessario rimpiazzare un blocco, si sceglie quello **usato meno di recente**.
    - Implementazione tipica: **una lista ordinata**, oppure **contatori di tempo** o **bit di utilizzo**.

LRU

- Il **timestamp** (contatore) serve per registrare l'**istante dell'ultimo accesso** a ogni blocco della cache.
- Come funziona in pratica:
  - Il gestore della cache si occupa di aggiornare **due contatori**:
    - Si mantiene un contatore **globale** che aumenta di 1 ad ogni accesso alla cache.
    - Quando un blocco viene letto o scritto, si aggiorna il suo contatore **locale** con il valore attuale del contatore globale.

LRU

- **Cache Set Associative**
  - **Sostituzione:** avviene solo all'interno del set dove il nuovo blocco deve essere inserito.
  - **Come funziona:**
    - Si applica la LRU **all'interno del set**: si tiene traccia di quali blocchi del set sono stati usati più o meno recentemente.
    - Se il set è pieno, si sostituisce il blocco meno recentemente usato **di quel set**.
    - Per set piccoli (es. 2 vie), si può usare un solo bit per indicare quale dei due blocchi è il meno recente.

FIFO

- Politica di sostituzione **First In First Out** (FIFO)
  - Si approssima la strategia LRU selezionando il blocco più vecchio anziché quello non usato da più tempo
- **Cache Fully Associative:**
  - Viene sostituito il blocco **più vecchio** (caricato per primo).
  - Implementazione: una **coda** circolare.
  - **Facile da gestire**, ma può non rispettare il reale utilizzo dei dati.
- **Cache Set Associative:**
  - In ogni set si tiene traccia dell'**ordine di arrivo** dei blocchi.
  - Semplice da implementare (es. con una piccola coda per ogni set).

LFU

LFU (Least Frequently Used): rimuove l'elemento meno **frequentemente** utilizzato.

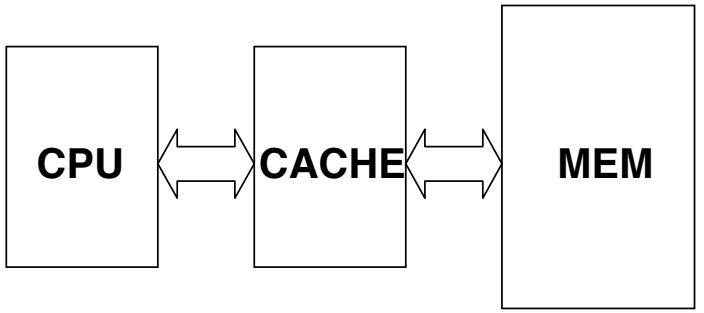
- **Cache Fully Associative:**
  - Viene sostituito il blocco **meno usato nel tempo**.
  - Richiede un **contatore** per ciascun blocco.
  - Più **costosa da implementare** (in hardware), ma potenzialmente efficace.
- **Cache Set Associative:**
  - Ogni riga del set ha un **contatore di accessi**.
  - Si sostituisce la riga **meno frequentemente usata** del set.
  - Più complesso → usato più spesso in software (es. cache disco) che in hardware.
- **Nota**: Problema dello "stacking", i.e. un oggetto usato molto tempo fa ma tante volte potrebbe restare in cache per sempre.

MRU

- **MRU (Most Recently Used):** Viene sostituito il blocco **più recentemente usato**.
- **Cache Fully Associative:**
  - Adatta in **pattern dove i dati appena usati non verranno riutilizzati presto**.
  - Meno comune, ma utile in alcuni contesti (ad es. strutture dati tipo stack).
- **Cache Set Associative:**
  - Si tiene traccia dell'ultimo blocco usato **in quel set**.
  - Quando necessario, si rimuove **quello**.
  - Anche qui, può avere senso in **casi specifici**, ma è raro in cache CPU tradizionali.

Gestione della cache

4 decisioni da prendere



1. Dove posizionare un blocco ?
2. Come reperire un blocco ?

Tecnica di indirizzamento

3. Quale blocco sostituire in corrispondenza di un miss ?

Algoritmo di sostituzione

4. Come gestire un'operazione di scrittura ?

Strategia di aggiornamento

Gestione dei miss in lettura

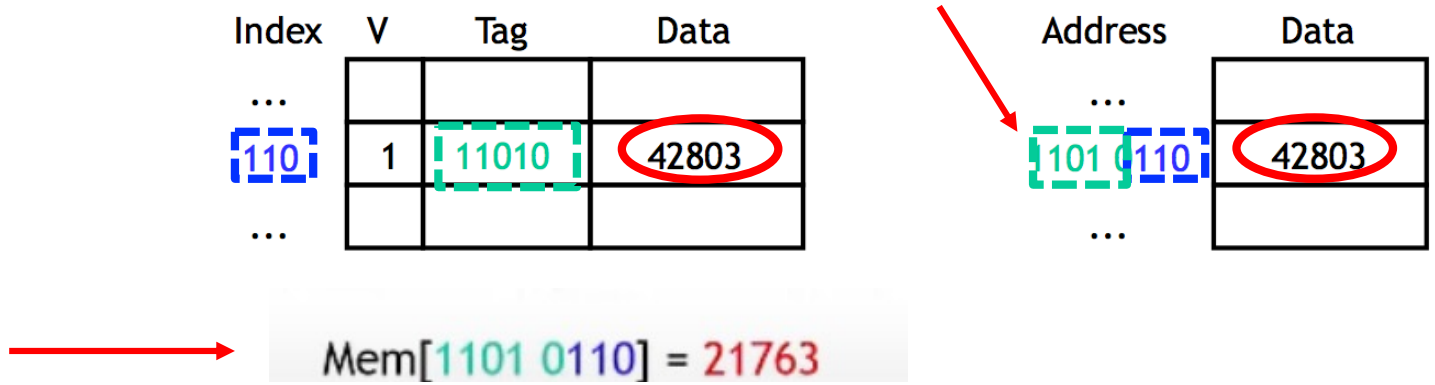
- Se un blocco non è presente nella cache bisogna mettere in stallo l'intera CPU
- In generale al verificarsi di un miss nella cache delle istruzioni sono necessari i seguenti passi:
  1. Inviare PC (uscita della ALU) alla memoria
  2. Lettura dalla memoria
  3. Scrittura nella cache (dato, tag e bit di validità)
  4. Riavviare l'esecuzione dell'istruzione che ha causato il miss.

Cache - Accesso in scrittura

- Scrivere un dato nella cache significa creare un'incoerenza, se non si aggiornano i livelli inferiori della gerarchia di memorie.
- Tale aggiornamento richiede lo stallo della CPU.
- Due tecniche risolutive:
  - **Write-through**:
  - **Write-back**
  - Utilizzo di un **write buffer**

Cache - Accesso in scrittura

- Scrivere un dato nella cache significa creare un'**incoerenza**, se non si aggiornano i livelli inferiori della gerarchia di memorie.



Cache: Write-through

- Write-through: i dati sono scritti nel blocco della cache e nel blocco del livello inferiore.
- Vantaggi
  - E' la soluzione più semplice da implementare
  - Si mantiene la coerenza delle informazioni nella gerarchia di memorie
- Svantaggi
  - Le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore → diminuiscono le prestazioni
  - Aumenta il traffico sul bus di sistema

Cache: Write-back

- **Write-back** (o copy-back): i dati sono scritti solo nel blocco della cache. Il blocco modificato viene scritto nel livello inferiore della gerarchia solo quando deve essere sostituito.
  - subito dopo la scrittura, cache e memoria di livello inferiore sono ***inconsistenti***
  - Viene mantenuto un **dirty bit**
- Vantaggi
  - Le scritture avvengono alla velocità della cache
  - Scritture successive sullo stesso blocco alterano solo la cache
- Svantaggi
  - Ogni sostituzione del blocco può provocare un trasferimento in memoria

Validity Bit vs Dirty Bit

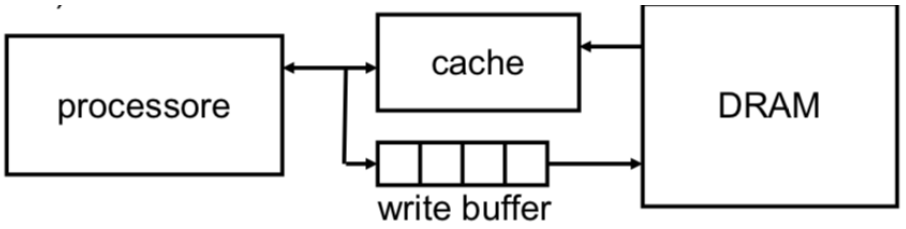
- Attenzione a non confonderli: hanno scopi diversi e indicano condizioni differenti nella cache.
  - **Validity bit = 0** → i dati non sono validi, la cache non deve usarli
  - **Validity bit = 1** → i dati sono validi, la cache può usarli.
  - **Dirty bit = 0** → i dati sono coerenti con la RAM.
  - **Dirty bit = 1** → i dati sono modificati in cache e devono essere scritti in RAM prima di essere rimpiazzati.
- Cosa succede ad entrambi:
  - Quando una riga viene caricata dalla memoria, valid bit=1 e dirty bit=0.
  - Se la CPU modifica i dati nella cache, allora il dirty bit viene impostato a 1 ma il validity bit resta 1.
  - Se la riga viene invalidata, il validity bit diventa 0, ma il dirty bit non ha senso finché la riga non è valida di nuovo.

Cache: Write-back

- **Vantaggi**:
  - **Prestazioni migliori**: riduce il traffico verso la memoria.
  - Molte scritture consecutive sullo stesso blocco causano **una sola scrittura in RAM**.
- **Svantaggi**:
  - Cache e memoria possono essere **temporaneamente incoerenti**.
  - Richiede logica più complessa (bit dirty, gestione della scrittura alla rimozione).
  - Più complicato in sistemi **multi-core** (rischio di inconsistenza tra più cache).

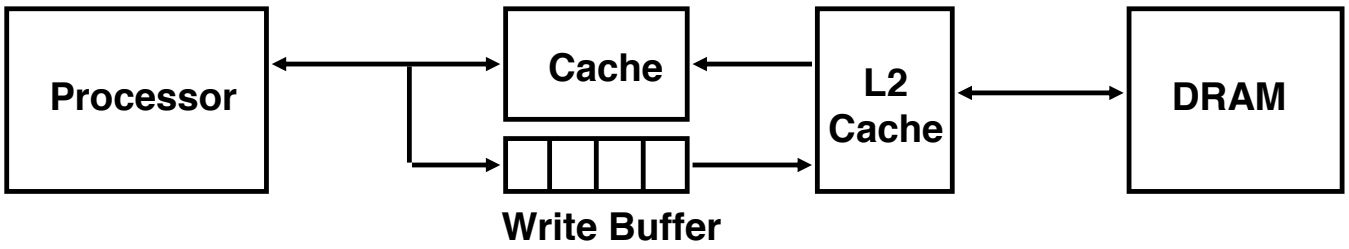
Cache: write-through con write buffer

- *Write buffer* interposto tra la cache e la memoria di livello inferiore
  - Il processore scrive i dati sia nella cache e nel write buffer
  - Il controller della memoria scrive il contenuto del write buffer in memoria
- Il write buffer è gestito in modalità FIFO
  - Numero tipico di elementi del buffer: 4
  - Efficiente se la frequenza di scrittura << 1/write cycle della DRAM
  - Altrimenti, il buffer può andare in saturazione ed il processore deve aspettare che le scritture giungano a completamento (*write stall*)



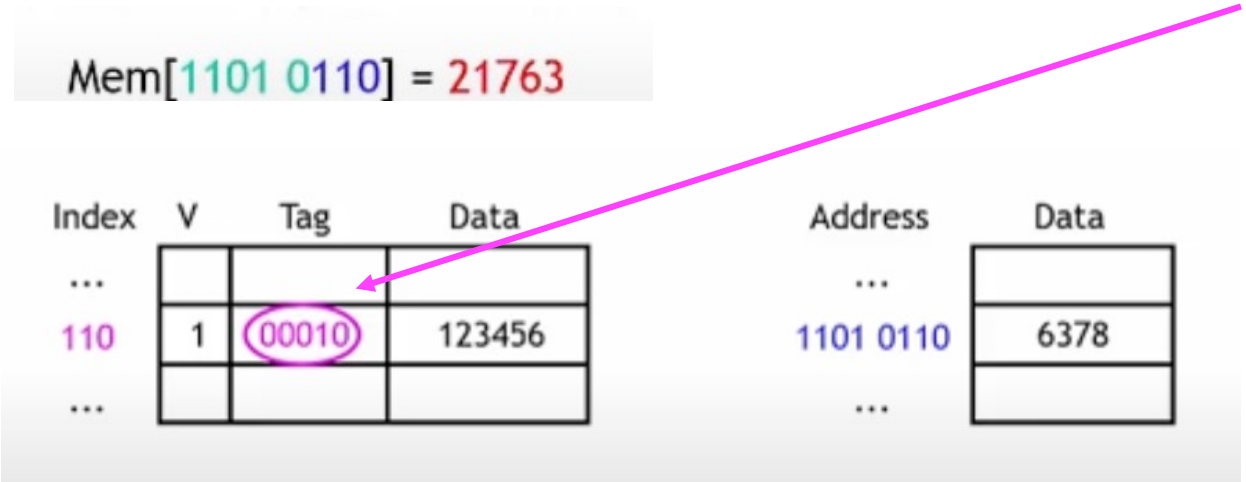
Cache: write-through con write buffer

- *Write buffer* interposto tra la cache e la memoria di livello inferiore
  - Il processore scrive i dati sia nella cache e nel write buffer
  - Il controller della memoria scrive il contenuto del write buffer in memoria
- Il write buffer è gestito in modalità FIFO
  - Numero tipico di elementi del buffer: 4
  - Efficiente se la frequenza di scrittura << 1/write cycle della DRAM
  - Altrimenti, il buffer può andare in saturazione ed il processore deve aspettare che le scritture giungano a completamento (*write stall*)



Write miss

- Un secondo scenario di inconsistenza potrebbe accadere se cerchiamo di scrivere in un indirizzo che non è (ancora) contenuto in cache: **write miss**
- Immaginiamo di voler memorizzare 21763 all'indirizzo 11010110



Write miss

- Le scritture possono indurre **write miss**
- Soluzioni possibili:
  - **Write allocate**: il blocco viene caricato in cache e si effettua la scrittura
  - **No-write allocate**: il blocco viene scritto direttamente nella memoria di livello inferiore, senza essere trasferito in cache

- Tipicamente abbiamo le seguenti combinazioni:
  - Write back con Write allocate
  - Write through con Write Not allocate

Osservazione (I)

- Una cache con blocchi di dimensioni maggiori sfruttano maggiormente la località spaziale diminuendo la frequenza di miss
- La frequenza di miss torna a crescere se la dimensione dei blocchi diventa troppo grande rispetto alla dimensione della cache
- Quindi i blocchi vengono scaricati dalla cache prima ancora che molti dati in essi contenuti siano stati utilizzati
- Quindi la località spaziale tra le parole di un blocco diminuisce e il miglioramento legato alla frequenza di miss si riduce

Osservazione (II)

- Inoltre, cresce anche il costo di una miss: la penalità di una miss è determinata dal tempo necessario a prelevare un blocco dal livello sottostante e a scriverlo nella cache
- Il tempo per prelevare un blocco è dato dalla somma tra la latenza per ottenere la prima parola del blocco e il tempo di trasferimento del resto del blocco