Cache

Fri, 27 May

La **cache** è quel tipo di memoria che si trova tra la *CPU* e la *memoria centrale*. La sua dimensione è nell'ordine del *MegaByte* e la sua velocità d'accesso è molto elevata.

La gestione della cache è *trasparente* al programmatore, tale memeoria viene gestita da un **algoritmo di caching**, basato sui *principi di località*.

Tecniche di indirizzamento

Esistono *tre* tipi di cache in base alla loro **tecnica di indirizzamento**:

- **Direct Mapped**
- **Fully Associative**
- **Set Assocaitve**

Direct Mapped

A ciasciun blocco della memoria corrisponde una specifica locazione della cache.

Per questioni di dimensione, un blocco della cache può mappare più blocchi di memoria.

L'indirizzo che indica un blocco di cache è composto da tre elementi:

- **Tag**

  Contiene informazioni per verificare se la parola cercata sia effettivamente quella nella cache.
- **Indice**

  Utilizzato per sezionare il blocco della cache.
- **Bit di validità**

  Un campo che indica se il blocco si memoria associato contiene un dato valido o meno.

Fully Associative

Ogni blocco può essere collocato in qualsiasi locazione della cache. C'è una corrispondenza uno ad uno tra memoria centrale e cache, per cui non tutte le locazioni della memoria centrale sono mappate.

Per ricercare un blocco è necessaria una *ricerca sequenziale* molto lenta, a cui si può ovviare con una molto più costosa *ricerca parallela*.

Set Associative

È una soluzione intermedia tra le *Direct Mapped* e la *Fully Associative*. Ciascun blocco della memoria ha a disposizione un numero fisso (≥2) di locazioni di cache.

Algoritmi di sostituzione

È necessario un *algoritmo* che scelga quali blocchi presenti nella cache *sostituire* per introurre i nuovi blocchi *necessari*.

Esisono tre tipi di politiche di sostituzione:

- *Casuale* [**Random**]

Si sceglie a caso un elemento da rimuovere, è semplice da implementare ed è molto rapida.

- ▼ *Least Recently Used* [**LRU**]

  Si rimuove l'elemento non utilizzato da più tempo, è necessario tenere traccia di quando un blocco è stato referenziato. A ogni blocco viene assegnato il valore massimo di importanza, valore che viene diminuito di uno ogni volta che si accede ad un blocco differente.

- ▼ *First in First Out* [**FIFO**]

  Simile all'*LRU*, si rimuove il blocco più vecchio, staccandosi però dal concetto di utilizzo.

Gestione dei miss in lettura

Se il blocco richiesto non è presente in cache, l'intera CPU entra in una fase di *stallo*.

Verificatosi un miss, sono necessari quattro passaggi:

- ▼ Inviare `PC−4` in uscita dall'ALU alla memoria

  Per poter riprendere l'esecuzione dall'istruzione che ha causato il miss

- Leggere della memoria del dato necessario
- Scrivere nella cache tale dato
- Riavviare l'esecuzione dell'istruzione che ha causato il miss

Accesso alla cache in scrittura

Scrivere un dato in un livello superiore della gerarchia, come la cache, genera un'incoerenza per cui si devono aggiornare i livelli inferiori, e ciò causa uno *stallo* della CPU.

Esistono tre *tecniche* risolutive:

- **Write-through**
- **Write-back**
- Usare un **write buffer**

Write-through

Il dato necessario viene scritto simultaneamente sia nella cache che nel livello inferiore.

È semplice da implementare e si mantiene la coerenza nella gerarchia.

Le operazioni di scrittura devono però essere effettuate alla velocità di scrittura della memoria a livello inferiore, aumentando anche il traffico sul BUS di sistema.

Write-back

I dati vengono scritti solamente nel blocco della cache, il blocco modificato viene aggiornato anche nel livello inferiore solo quando deve essere sostituito.

Le scritture avvengono alla velocità della cache, e eventuali scritture successive sullo stesso blocco alterano solo la cache.

Ogni sostituzione di blocco potrebbe implicare un trasferimento di memoria sui livelli inferiori

Write-through con write buffer

Si utlizza un buffer *FIFO* (**write buffer**) intermedio tra la cache e il livello inferiore.

Vengono scirtti i dati sia nella cache che nel *buffer*, e il controller della memoria aggiorna il livello inferiore *asincornamente* prendendo i dati dal *write buffer*.

È possibile utilizzare più livelli di cache (L1, L2, L3) per evitare di mandare in overflow il write buffer.