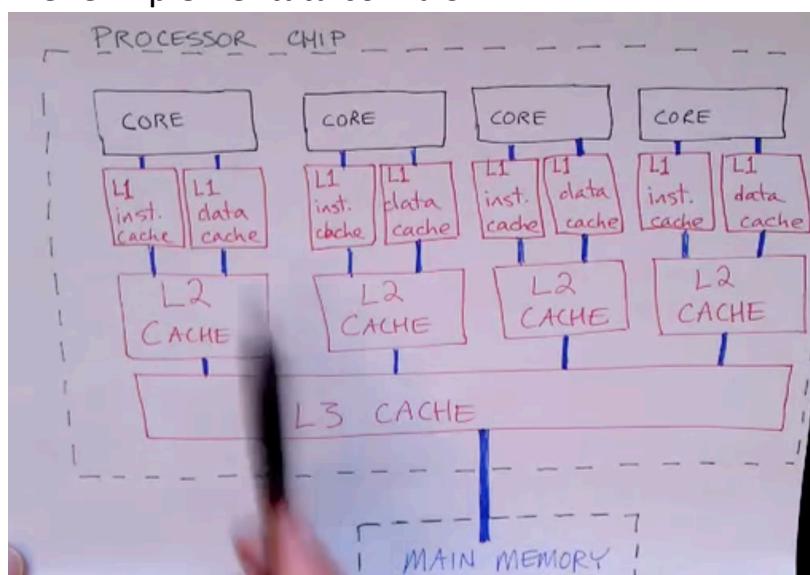


Caches

Thursday, 24 August 2023 09:50

- La main memory è lenta, quindi si usa una cache
 - o La cache è tra la CPU e la main memory
 - o La CPU manda un address alla Cache
 - Se c'è nella cache, allora ritorna alla CPU
 - Senò lo reindirizza alla main memory
 - o Nella cache si mettono informazioni molto frequenti e critiche
 - Facendo così possiamo dare un accesso più veloce alla cpu
 - Si possono mettere caches in tutti i casi dove vogliamo avere blocchi di dati più velocemente
 - o La cache è locata nella CPU ed è tra la CPU e la DRAM
 - o Viene implementata con la SRAM



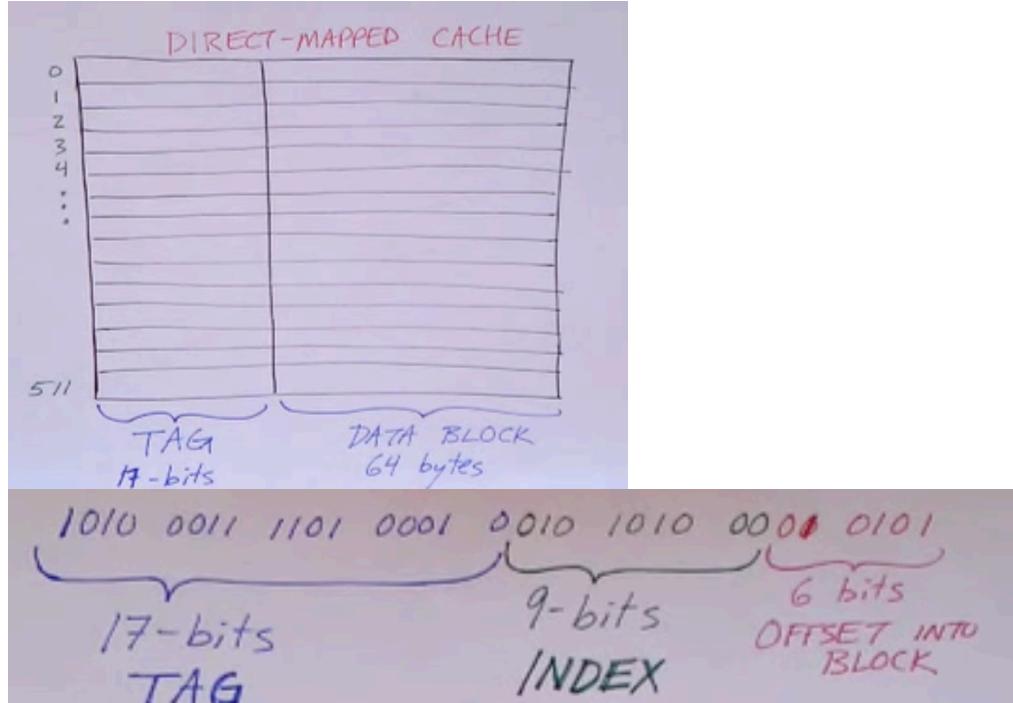
L'immagine è abbastanza autoesplicativa



- Ci sono 2 modi per fare funzionare una cache:
 - o Set associative
 - Abbiamo gruppi di memoria
 - Quindi sta volta controlliamo i gruppi chiamati set
 - o Fully associative cache

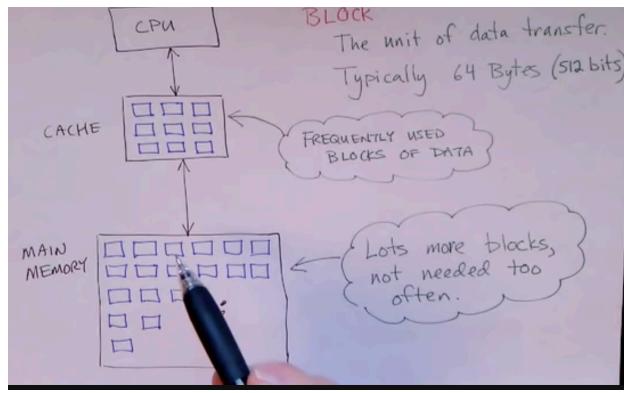
- Non sono utilizzati tanto
- Ogni blocco può andare in qualunque cache line
- Quindi dobbiamo tenere informazioni di quale blocco lo contiene, e quindi iterare per tutta la cache
- Directed-mapped cache
 - Ogni blocco può andare in solamente 1 cache line
 - Ogni linea può contenere solamente determinati blocchi
 - E quindi possiamo saltare tante linee e controllare solamente 1 linea
 - Noi lavoreremo con questa
- Esempio con $2^9 = 512$ bits cache

Nota: manca 1 bit, alla fine della colonna, chiamato bit di validità



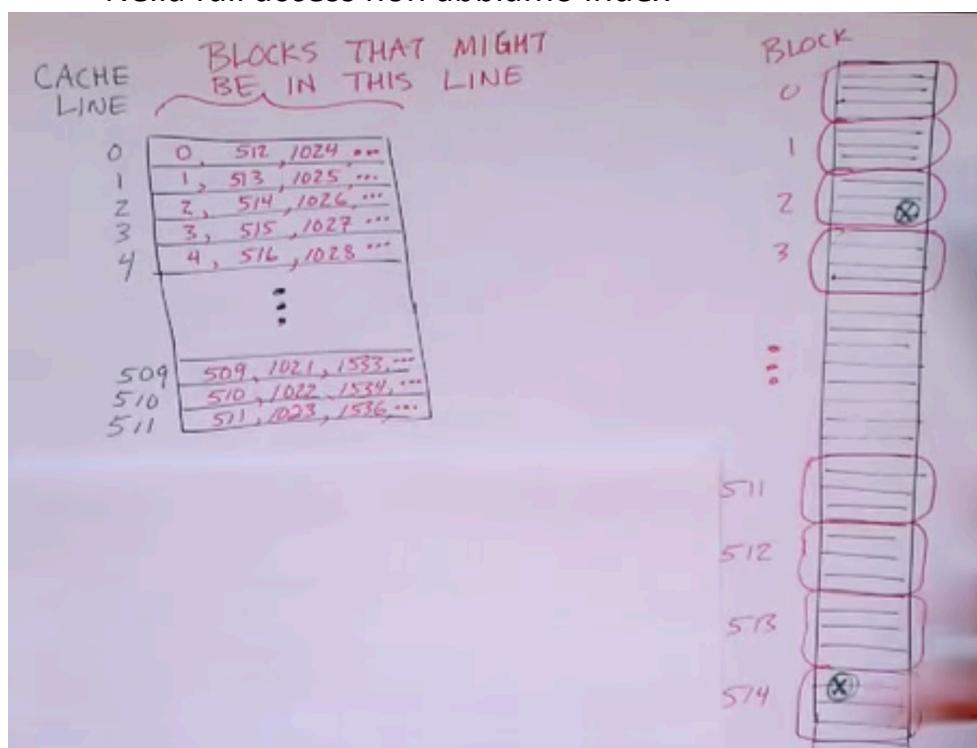
Allora, spiegazione:

- Noi con l'index scendiamo giù alla riga che ci interessa
- Una volta scesi, controlliamo se il tag è uguale, se si allora la cache possiede l'informazione che ci serve
 - Se non è stato trovato, abbiamo fatto una cache miss
 - L'accesso è lento
 - Il nuovo blocco viene salvato poi nella cache
 - ◆ Avviene un Eviction/Replacements
 - ◊ Eviction = Blocco da rimuovere
 - ◊ Ovviamente un blocco viene rimosso
- In talcaso, ci spostiamo di offset bit nel data blocks
- Abbiamo fatto una cache hit
 - L'accesso è veloce
- Ogni riga viene chiamato "blocco"



Nota:

- Abbiamo 512 righe = 2^9 ed il nostro index è 9 bits
- Il data block è grande 64 bytes, il nostro offset di block è fatto da 6 bits, $2^6 = 64$
- Nella full access non abbiamo index



- Possiamo notare che la linea 1 può avere o blocco 1, Oppure blocco 513 oppure 1025
 - Siccome stiamo lavorando con 9 bit, i bits dopo il 9 vengono troncati
 - Quindi $0000\ 0001\ 0 = 1\ 0000\ 0001\ 0 = 10\ 0000\ 0001\ 0$
- Da questo possiamo notare un problema: le linee possono contenere troppi blocchi, e quindi si condividono troppo linee di caches
 - Ci sono conflitti, e quindi la cache può andare in trashing

- Cache misses

- Avviene quando stiamo cercando un valore nella cache ma non esiste, e ci sono 3 casi di cache misses.

esiste, e ci sono 3 casi di cache misses.

- Compulsory
 - All'inizio quando il programma è appena iniziato
 - Ci sono sempre questi all'inizio
 - La cache all'inizio è cold
- Capacity
 - La cache è troppo piccola e quindi non possiamo tenere tutto, e quindi avremo tante misses
- Conflict
 - Abbiamo tanti spazi liberi però 2 variabili frequenti vogliono mapparsi nella stessa linea
- Se abbiamo troppi misses può capitare trashing
 - Due variabili combattono per la stessa linea
 - Ogni accesso è una cache miss
 - La performance muore
 - Hot spots = Linee dove ci sono tante variabili
- Locality
 - Si basa sul fatto che certe variabili vengono usate molto ed altre raramente, e le variabili che vengono usate molto vengono chiamate hotspot
 - Dentro ad un loop abbiamo un hotspot
 - Principi di località:
 - Temporal locality: se un byte è stato usato recentemente, allora è possibile verrà usato dinuovo prossimamente
 - Spatial locality: se un byte è stato usato recentemente, allora probabile che i byte vicini verranno utilizzati
 - Questo viene usato sia per le informazioni che le istruzioni
 - Le variabili che sono utilizzate più frequentemente sono chiamate working set
 - E' il set di bytes che è stato usato recentemente
 - E' il set di bytes che sarà necessario nel vicino futuro
 - Se il working set è in cache, il programma sarà più veloce
 - Quando portiamo bytes nella cache, li teniamo
 - Proviamo a tenere i bytes nella cache se è stato usato più recentemente
 - Se portiamo un byte, portiamo anche i bytes vicini
 - Infatti portiamo blocchi di bytes
- Quando la CPU vuole scrivere:
 - Wire-hit
 - La cache contiene l'informazione, ed abbiamo 2 decisioni:
 - Write through
 - Cache immediatamente e modifica il blocco in memoria

- Write back
Cache immediatamente e modifica il blocco in memoria quando block evicted
- Write-miss
 - Write-Allocate
La cache legge il memoria, la mette in cache e poi update la cache
 - Write-No-Allocate
La cache forward la modifica alla memoria centrale

16+8+4+2+1

1 0011

1 3

128

1000 0000

80