

Inquadramento del problema

L'analisi di un congruo numero di programmi scritti da programmatori professionisti ha dimostrato che la maggior parte del tempo di esecuzione di una CPU è, di solito, impegnato da procedure in cui vengono eseguite ripetutamente le stesse istruzioni.

In realtà, non è importante conoscere lo schema dettagliato della sequenza di istruzioni, il punto chiave è che molte istruzioni in aree ben localizzate del programma vengono eseguite ripetutamente in un determinato periodo, e si accede al resto del programma relativamente di rado.

Questa proprietà viene chiamata *località dei riferimenti*. Si manifesta in due modi: *località temporale* e *località spaziale*.

La *località temporale* rappresenta la probabilità che un'istruzione eseguita di recente venga eseguita nuovamente, entro breve tempo.

La *località spaziale* rappresenta invece la probabilità che istruzioni vicine ad un'istruzione eseguita di recente (dove la vicinanza è espressa in termini di indirizzi delle istruzioni) siano anch'esse eseguite nel prossimo futuro.

Utilità di avere una "cache memory"

La velocità con cui la memoria risponde alle richieste di istruzioni e dati della CPU ha un peso determinante sulle prestazioni di un sistema,

Se tra la memoria principale e la CPU si potesse **interporre una memoria molto veloce, contenente le parti di programma e di dati che, volta per volta, interessano l'elaborazione**, il tempo totale di esecuzione verrebbe ridotto in modo significativo.

Questa è esattamente la funzione della "*cache memory*" che in inglese significa letteralmente "*schermo della memoria*". Si tratta quindi di un elemento che inserito tra CPU e memoria principale impedisce alla prima di "*vedere*" i tempi di risposta reali della memoria. La struttura della CPU, infatti, non viene influenzata dalla presenza o meno di una *cache memory*. L'interfaccia verso la memoria applica il "*protocollo*" di trasferimento *dalla* memoria o *verso la* memoria ignorando la presenza di una struttura intermedia.

Agli albori di questa tecnica, infatti, la cache memory era solo un accessorio a pagamento.

Una traduzione possibile in italiano del concetto è "memoria tampone".

Principio di funzionamento

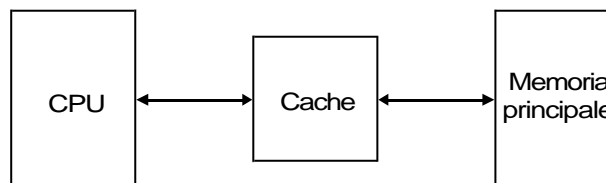
Concettualmente, le operazioni svolte da una memoria cache sono molto semplici.

- I circuiti di controllo della memoria cache sono progettati per avvantaggiarsi della proprietà **della località dei riferimenti**.
- **L'aspetto temporale** di questa proprietà suggerisce di portare un elemento (istruzioni o dati) nella cache quando viene richiesto per la prima volta, in modo tale che rimanga a disposizione nel caso di una nuova richiesta.
- **L'aspetto spaziale** suggerisce che, invece di portare dalla memoria principale alla cache un elemento alla volta, è conveniente portare un insieme di elementi che risiedono in indirizzi adiacenti.

Si farà riferimento al termine **blocco** per indicare un insieme di indirizzi contigui di una qualche dimensione.

Un altro termine utilizzato di frequente per indicare un blocco della cache è **linea di cache**.

Dimensione della cache



Quando si riceve una richiesta di lettura fatta dalla CPU, il contenuto di un blocco di parole di memoria contenenti la locazione specificata viene trasferito nella cache, una parola alla volta. In seguito, ogniqualvolta il programma fa riferimento a una di queste locazioni del blocco, i valori desiderati vengono letti direttamente dalla cache.

E' chiaro che perché la cache svolga efficacemente il suo compito deve avere **tempi di accesso molto più brevi di quelli della memoria principale** e, quindi, a parità di tecnologia **deve essere molto più piccola**.

Se è piccola non potrà contenere una frazione ridotta delle istruzioni ed i dati della memoria principale.

Posizionamento e sostituzione

La corrispondenza tra i blocchi della memoria principale e quelli della cache è specificata dalla funzione di *posizionamento (mapping)*.

Quando la cache è piena e si fa riferimento a una parola di memoria (istruzione o dato) che non è presente nella cache, l'hardware di controllo della cache deve decidere quale blocco della cache debba essere rimosso per far spazio al nuovo blocco, che contiene la parola a cui si fa riferimento. L'insieme di regole in base alle quali viene fatta questa scelta costituisce *l'algoritmo di sostituzione*.

La struttura dei sistemi è, di solito, organizzata in modo che la presenza di una cache non influenzi il funzionamento della CPU. Quest'ultima, infatti, nell'esecuzione del programma effettua le richieste di lettura e scrittura utilizzando gli indirizzi delle locazioni nella memoria principale. È la logica di controllo della cache che si fa carico di determinare se la parola richiesta è presente o meno nella cache. Se è presente, viene effettuata l'operazione di lettura o scrittura della locazione di memoria appropriata. In questo caso si dice che *l'accesso in lettura o in scrittura ha avuto successo (read hit o write hit)*.

Read hit e write hit

Se l'accesso alla cache ha avuto successo bisogna distinguere due tipi di situazioni. Nel caso di **un'operazione di lettura**, la memoria principale non viene coinvolta, mentre per **un'operazione di scrittura** il sistema può procedere in due modi. Nel primo (*write-through*), la locazione della cache e quella della memoria principale vengono entrambe aggiornate.

Nel secondo si aggiorna soltanto la locazione della memoria cache, segnandola come aggiornata con un bit *di modifica o dirty*.

In questo caso, la locazione della memoria principale viene aggiornata in seguito, quando il blocco contenente la parola marcata deve essere rimosso dalla memoria cache per far posto a un nuovo blocco.

Tale tecnica viene chiamata spesso *write-back*, o *copy back*.

Il *write-through* è più semplice, ma implica **inutili operazioni di scrittura** nella memoria principale, se una parola viene aggiornata più volte durante il periodo in cui risiede nella cache. Ma anche il protocollo *write-back* può causare **inutili scritture nella memoria principale**, visto che, quando si procede con la scrittura nella memoria principale di un blocco, **tutte le parole del blocco vengono scritte**, anche se solo una delle parole del blocco della cache è stata modificata.

Read miss e write miss

Quando la parola indirizzata durante un'operazione di lettura non è presente nella cache si dice che *l'accesso in lettura è fallito (read miss)*. Il blocco di parole contenente la parola richiesta viene copiato dalla memoria principale nella memoria cache. Dopo aver caricato l'intero blocco nella memoria cache, la parola richiesta viene inviata alla CPU.

In alternativa, è possibile inviare immediatamente la parola alla CPU, non appena la si legge dalla memoria principale. Quest'ultimo approccio, chiamato *load-through*, o anche *early restart*, **riduce in qualche modo il tempo di attesa della CPU, a discapito di una maggiore complessità del circuito di controllo.**

Durante un'operazione di scrittura, se la parola indirizzata non è nella cache si dice che *l'accesso in scrittura è fallito (write miss)*. Quindi, se si utilizza un protocollo *write-through*, le informazioni vengono scritte direttamente nella memoria principale, nel caso invece di un protocollo *write-back*, il blocco contenente la parola indirizzata viene prima caricato nella cache, poi viene sovrascritto con le nuove informazioni.

Posizionamento: indirizzamento diretto

Si consideri una memoria cache costituita da **128 blocchi** (numerati da 0 a 127) di **16 parole ciascuno, per un totale di 2048 (2K) parole.**

Per individuare una parola nella cache è necessario un indirizzo di 11 bit.

Se supponiamo che la memoria principale del sistema abbia una capacità di **64K parole (indirizzi di 16 bit)**, essa può essere immaginata suddivisa in **4K blocchi da 16 parole ciascuno**, numerati da 0 a 4095.

La tecnica di *indirizzamento diretto* associa:

- il blocco 0 della memoria principale, che comprende le 16 locazioni da 0000_{16} a $000F_{16}$, insieme al blocco 128 ($03FF_{16}$ - 0400_{16}), al blocco 256, e così via fino al blocco 3967 al blocco 0 della cache.
- Il blocco 1 (0010 - $001F$), il blocco 129, il blocco 257 e così via fino a quello 3968 al blocco 1 della cache.
- In generale il blocco *i-esimo* della memoria principale, insieme ai blocchi di indice $i + n128$, con n che varia da 0 a 32 sono associati al blocco *i-esimo* della cache.

I bit di *etichetta*

Se si confrontano i 16 bit di indirizzo delle locazioni della memoria principale con gli 11 bit d'indirizzo della cache si vede che i quattro bit meno significativi di entrambi (una cifra esadecimale) individuano sempre la parola all'interno di un blocco, sia in memoria che nella cache.

I rimanenti 12 bit dell'indirizzo della memoria principale individuano uno dei 4096 blocchi in cui essa è articolata.

I **7 bit meno significativi** individuano l'**indice del blocco della cache** a cui il blocco della memoria principale è associato.

I **5 bit più significativi** rappresentano, invece, una sorta di **indice di molteplicità dell'associazione**.

In pratica i 5 bit più significativi sono il valore dell'indice *n* che stabilisce quale dei 32 blocchi di memoria associabili ad un certo blocco della cache è effettivamente presente nella stessa. Essi sono contenuti in un registro che la struttura hardware della cache mette a disposizione e che viene di solito chiamato *etichetta (tag)*.

Nella figura allegata, il blocco 4095 della memoria principale, l'ultimo, ha come etichetta 31 ed è associato al blocco 127 della memoria cache.

La ricerca dell'indirizzo nella cache

Memoria principale			Memoria cache			B
Indirizzo binario	ind. esadec.	Locazione	Blocco	Parola	Locazione	dec.
00000 0000000 0000	000 0	XX.....X	0000000	0000	XX.....X	0
00000 0000000 0001	000 1	XX.....X	0000000	0001	XX.....X	
.....	
00000 0000000 1111	000 F	XX.....X	0000000	1111	XX.....X	1
00000 0000001 0000	001 0	XX.....X	0000001	0000	XX.....X	
.....	
00000 0000001 1111	001 F	XX.....X	0000001	1111	XX.....X	127
.....	
.....	
00000 1111111 0000	07F 0	XX.....X	1111111	0000	XX.....X	0
00000 1111111 1111	07F F	XX.....X	1111111	1111	XX.....X	
00001 0000000 0000	080 0	XX.....X	0000000	0000	XX.....X	
00001 0000000 1111	080 F	XX.....X	0000000	1111	XX.....X	1
00001 0000001 0000	081 0	XX.....X	0000001	0000	XX.....X	
.....	
00001 0000001 1111	081 F	XX.....X	0000001	1111	XX.....X	127
.....	
.....	
11111 1111111 0000	FFF 0	XX.....X	1111111	0000	XX.....X	127
.....	
11111 1111111 1111	FFF F	XX.....X	1111111	1111	XX.....X	

Tag

Funzionamento dell'indirizzamento diretto

In pratica quando un blocco della memoria principale deve essere trasferito nella cache **la sua posizione è già predeterminata**.

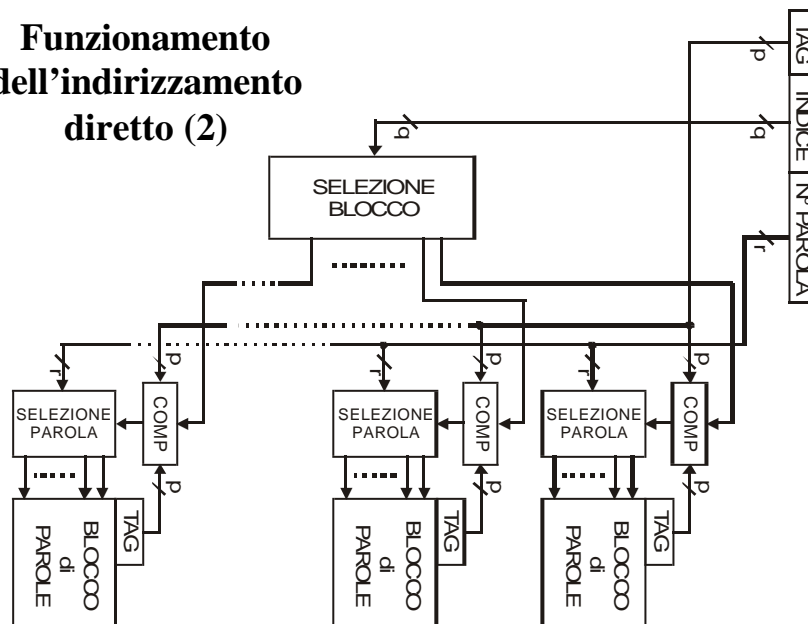
- Esso andrà nella posizione individuata dai 7 bit appena più significativi rispetto ai 4 che contraddistinguono le singole parole del blocco.
- Il blocco sarà accompagnato, nel **campo etichetta** della cache, dai 5 bit più significativi dell'indirizzo, che serviranno negli accessi alla cache, per capire quale dei 32 blocchi che potrebbero essere in quella posizione, vi si trova effettivamente.

Durante l'esecuzione del programma l'indirizzo a 16 bit, da cui bisogna prelevare l'istruzione o il dato, viene spezzato in tre parti.

I 5 bit più significativi vengono confrontati con l'**etichetta** del blocco della cache individuato dai successivi 7 bit, per vedere se quel particolare blocco di informazioni è effettivamente presente nella cache.

Se l'**etichetta** corrisponde, si legge la parola del blocco a cui puntano i 4 bit meno significativi.

Funzionamento dell'indirizzamento diretto (2)



Considerazioni sull'algoritmo

Poiché, in generale ad uno stesso blocco della cache è associato un numero di blocchi della memoria principale superiore a uno, **può insorgere un conflitto per occupare un determinato blocco della cache, anche se questa non è piena.**

Per esempio, le istruzioni di un programma possono iniziare nel blocco 1 e continuare nel blocco 512, magari in seguito a un salto.

L'esecuzione del programma implica che entrambi i blocchi siano trasferiti nella cache, ma entrambi corrispondono al blocco 1 della stessa.

Il problema viene risolto consentendo al nuovo blocco di sovrascrivere quello attualmente residente nella cache. **In questo caso, l'algoritmo di sostituzione è banale.**

Il metodo di indirizzamento diretto è facile da realizzarsi, ma non è molto flessibile.

Cache *completamente associative*

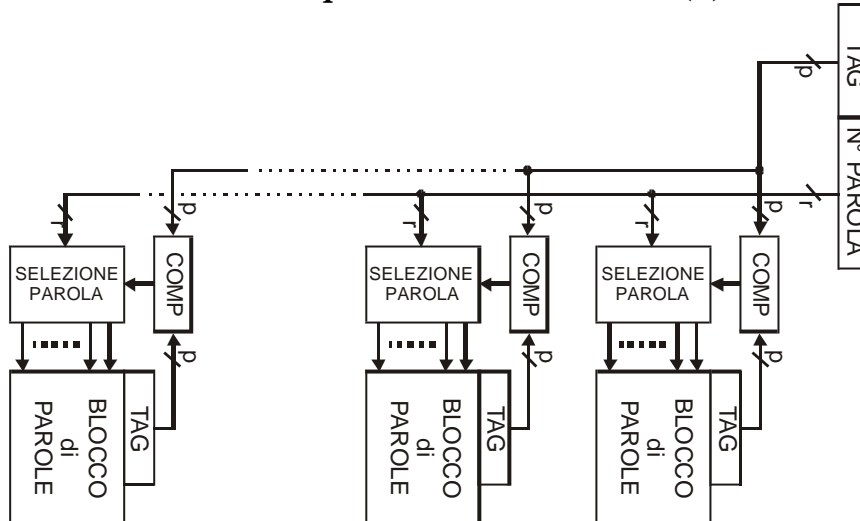
Esiste un modo molto più flessibile di posizionamento, in cui **un blocco della memoria principale può essere posizionato in un qualsiasi punto della cache.**

Con 2048 locazioni della cache divise in 128 blocchi da 16 locazioni, **sono necessari 12 bit di etichetta per identificare uno dei 4096 blocchi della memoria principale**, quando questo risiede nella cache.

I bit di etichetta di un indirizzo provenienti dalla CPU sono confrontati con l'etichetta di ogni blocco della memoria cache per vedere se il blocco cercato è presente.

Si tratta della cosiddetta tecnica di ***indirizzamento completamente associativo***; essa consente la massima libertà nella scelta della locazione della cache in cui posizionare il blocco di memoria, quindi lo spazio della cache può essere utilizzato in modo molto più efficiente. Un nuovo blocco che deve essere copiato nella cache ne fa uscire un altro già residente solo se la cache è già piena. In tal caso, è però necessario **utilizzare un algoritmo per selezionare il blocco che deve essere sostituito**. Come discuteremo più avanti, ci sono numerosi ***algoritmi di sostituzione*** che si possono usare.

Cache completamente associative (2)



Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

15

Le CAM (Content Addressable Memory)

Il costo di una memoria cache completamente associativa è maggiore rispetto a quello di una cache a indirizzamento diretto, poiché è necessario esaminare 128 etichette per determinare la presenza o meno di un blocco in memoria. Una ricerca di questo tipo si chiama *ricerca associativa* o *per contenuto*, in quanto corrisponde all'operazione che si fa quando si vuole sapere se un oggetto od un cognome sono presenti in un elenco. Se si adoperasse **una logica seriale**, il tempo di ricerca sarebbe moltiplicato per il numero di confronti da effettuare e ciò porterebbe a **tempi di accesso inammissibili**. La ricerca viene effettuata con un **algoritmo parallelo** del tipo visto nello schema a blocchi precedente.

Le cache di questo tipo si chiamano anche "*memorie indirizzabili per contenuto*" ed, infatti, hanno in inglese un nome dettato dalla loro struttura: **C.A.M. (Content Addressable Memory)**. Sono ormai disponibili anche come componenti di sistemi non di calcolo. Recentemente la MOSAID Technologies Inc. ha presentato una CAM da **64k locazioni da 36 bit** con un **tempo di accesso massimo di 60 ns**, in un contenitore da 304 pin.

Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

16

Le cache *set-associative*

Esiste **una combinazione delle due tecniche di indirizzamento**.

I blocchi della cache sono raggruppati in *insiemi (set)*, ed il meccanismo di associazione è congegnato in modo che ciascun blocco della memoria principale è associato ad un particolare *insieme* della cache ma può risiedere in una qualsiasi posizione all'interno di esso.

Avere più possibilità di posizionamento di un blocco in un insieme riduce la possibilità di dover attivare meccanismi di sostituzione di blocchi nella cache, anche quando essa è non è completa, e riduce il numero di etichette tra cui effettuare la ricerca associativa.

Con una cache da 2048 parole, suddivisa in 128 blocchi da 16 locazioni ciascuna, se si sceglie di avere **4 blocchi per insieme**, si hanno 32 insiemi. Se la memoria principale ha i soliti 4096 blocchi (64k locazioni) essi sono individuabili attraverso i 12 bit più significativi di indirizzo comuni alle 16 parole.

Ad ogni insieme della cache corrispondono, allora, 128 blocchi della memoria principale ma solo fino ad un massimo di 4 di essi possono trovare effettivamente posto contemporaneamente nella cache.

L'algoritmo di posizionamento

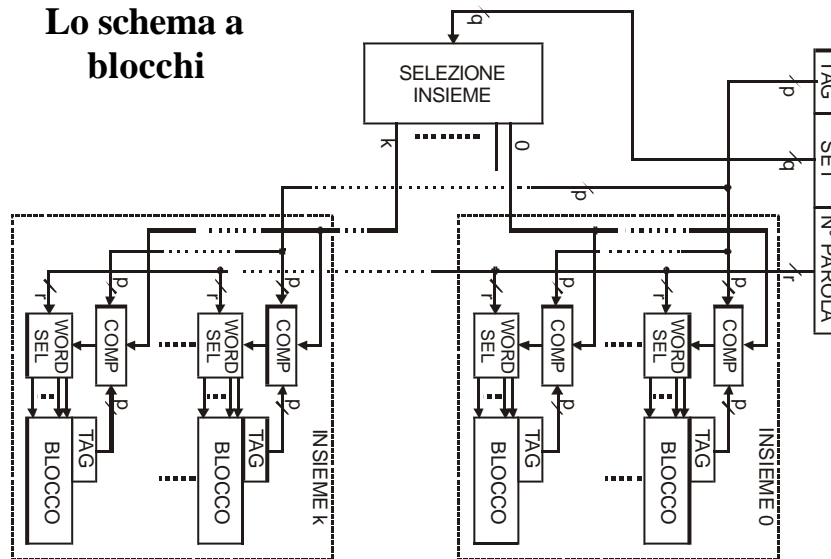
I 128 blocchi della memoria 0, 32, 64, 96, 128, ..., 4064 puntano all'insieme 0 della cache, in una qualsiasi delle quattro posizioni.

I blocchi 1, 33, 65, 97, 129, ... 4065, sono associati all'insieme 1 della cache e così via per gli insiemi 2, 3 4 e fino a quello 31.

Quando un blocco viene caricato nella cache, i 5 bit meno significativi dei 12 bit d'indirizzo individuano l'insieme della cache dove esso deve essere posizionato; in un blocco vuoto. I successivi 7 bit più significativi dell'indirizzo vengono trascritti nel registro della cache associato al blocco, e le 16 parole di programma memorizzate nelle locazioni.

Quando si effettua un accesso, i 4 bit meno significativi dell'indirizzo sono inizialmente ignorati, serviranno successivamente per individuare su quale delle 16 parole del blocco si deve operare. I successivi 5 bit (*campo set*) indicano l'insieme dove cercare, mentre i 7 bit più significativi indicano il blocco cercato. Questi bit debbono, quindi, essere comparati associativamente con le etichette dei quattro blocchi dell'insieme, per controllare se il blocco desiderato è presente o meno nell'insieme. Questa ricerca associativa su solo 4 etichette è certamente molto più semplice che in una memoria completamente associativa.

Lo schema a blocchi



Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

21

La cache *set-associativa* è un compromesso

La condizione limite, di 128 blocchi per insieme, non richiede alcun bit per il **campo set** e corrisponde alla tecnica **completamente associativa**, con 12 bit di etichetta.

L'altra condizione limite è costituita dalla presenza di un blocco per insieme, che corrisponde alla **cache ad indirizzamento diretto**.

Un ulteriore bit di controllo, chiamato **bit di validità**, è necessario per ogni blocco. Questo bit indica se il blocco contiene o meno dati validi.

Non deve però essere confuso con il **bit di modifica**, menzionato in precedenza, il bit di modifica, che indica se il blocco è stato modificato o meno durante il periodo in cui è restato nella cache, serve soltanto nei sistemi che non fanno uso del metodo **write-through**.

I bit di validità vengono tutti posti a 0 nell'istante iniziale di accensione del sistema e, in seguito, ogni volta che nella memoria principale vengono caricati programmi e dati nuovi dal disco.

Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

22

Il bit di validità

I trasferimenti dal disco alla memoria principale vengono effettuati mediante un meccanismo di DMA (Direct Memory Access).

Di solito, programmi e dati nuovi, provenienti da disco, vanno direttamente nella memoria principale.

Il bit di validità di un certo blocco della cache viene posto a 1 la prima volta che le 16 parole di dati o programma vengono caricate; poi, ogni volta che un blocco della memoria principale viene sovrascritto con codice sorgente che non è passato attraverso la cache, viene effettuato un controllo per determinare se gli indirizzi del blocco che sta per essere cancellato siano presenti o meno nella cache.

Se il blocco precedente della memoria principale aveva una copia nella cache, il corrispondente bit di validità viene posto a 0; in tal modo, si garantisce che nella cache non ci siano dati obsoleti.

Il bit di validità a 0 candida immediatamente il blocco ad essere sovrascritto

Coerenza della cache

Una situazione dello stesso tipo si determina quando viene effettuato un trasferimento tramite DMA dalla memoria al disco e la cache utilizza un protocollo *write-back*.

In questo caso, i dati nella memoria potrebbero non riflettere i cambiamenti che possono essere stati fatti sulla copia dei dati presenti nella cache.

Una possibile soluzione a tale problema consiste nello ***svuotamento (flush)*** della cache forzando i dati con il bit di modifica attivo a essere ricopiati nella memoria principale prima che venga effettuato il trasferimento tramite DMA.

Il sistema operativo è in grado di fare tutto ciò in modo molto semplice, senza peggiorare significativamente le prestazioni, visto che le operazioni di ingresso/uscita di scrittura non sono molto frequenti.

Questa necessità di garantire che due diverse entità (i sottosistemi della CPU e del DMA nel caso presente) utilizzino la stessa copia dei dati viene chiamata problema della ***coerenza della cache***

Algoritmi di sostituzione

In una cache ad indirizzamento diretto, la posizione di ogni blocco è predefinita, quindi non esiste alcuna strategia di sostituzione.

Nelle memorie cache *associative* e *set-associative*, esiste invece una certa flessibilità. **Quando un nuovo blocco deve essere portato nella cache, e tutte le posizioni che potrebbe occupare contengono dati validi, il controllore della cache deve decidere quale blocco, tra quelli esistenti, sovrascrivere.** Questa è una decisione importante, perché la scelta potrebbe essere determinante per le prestazioni del sistema.

In generale, l'obiettivo è quello di **mantenere nella cache quei blocchi che hanno una maggior possibilità di essere nuovamente utilizzati nel prossimo futuro.** ma, non è facile prevedere i blocchi a cui si farà riferimento.

Una possibile strategia è di tener presente che la *località dei riferimenti* suggerisce che i blocchi a cui c'è stato accesso di recente hanno elevata probabilità di essere utilizzati nuovamente entro breve tempo.

Sostituzione con algoritmo LRU

Quando bisogna eliminare dalla cache un blocco, è **sensato sovrascrivere quello a cui non si accede da più tempo.** Tale blocco prende il nome di blocco *utilizzato meno di recente (Least Recently Used, LRU)*, e la tecnica si chiama *algoritmo di sostituzione LRU*.

Per utilizzare l'algoritmo LRU, il controllore della cache deve mantenere traccia di tutti gli accessi ai blocchi mentre l'elaborazione prosegue.

Il blocco LRU di una memoria cache *set-associativa* con insiemi di quattro blocchi, può, ad esempio, essere indicato da un contatore di 2 bit associato a ciascun blocco, il cui contenuto sia opportunamente gestito secondo un protocollo automatico

Gestione contatore di accessi

Quando si ha un successo nell'accesso al blocco (*hit*):

- il contatore di quel blocco viene posto a 0. I contatori con valori originariamente inferiori a quelli del blocco a cui si accede, vengono incrementati di uno, mentre tutti gli altri rimangono invariati.

Quando, invece, l'accesso fallisce (*miss*) si aprono due possibilità:

- l'insieme non è pieno, il contatore associato al nuovo blocco caricato dalla memoria principale viene posto a 0 e il valore di tutti gli altri contatori viene incrementato di 1.
- l'insieme è pieno, si rimuove il blocco il cui contatore ha valore 3 e si pone il nuovo blocco al suo posto, mettendo il contatore a 0. Gli altri tre contatori dell'insieme vengono incrementati di un'unità.

E' facile verificare che i valori dei contatori occupati sono sempre diversi.

Alternative all' algoritmo LRU

L' algoritmo LRU è stato utilizzato ampiamente, con buone prestazioni in numerosi schemi di accesso. Ma in alcune situazioni, per esempio, nel caso di accessi sequenziali a un vettore di elementi che è leggermente troppo grande per poter stare tutto nella cache, le prestazioni sono molto scadenti

Le prestazioni dell' algoritmo LRU possono essere migliorate introducendo una piccola dose di casualità nella scelta del blocco da sostituire.

Sono stati proposti molti altri algoritmi di sostituzione più o meno complessi, che richiedono talvolta anche notevoli complicazioni hardware, ma **l' algoritmo più semplice, e spesso anche piuttosto efficace, consiste nello scegliere in modo completamente casuale il blocco da sostituire.**

Considerazioni generali sulle prestazioni

Le prestazioni di un computer dipendono dalla velocità con cui possono essere portate nella CPU le istruzioni macchina e dalla velocità con cui queste possono essere eseguite.

La velocità dell'esecuzione può essere aumentata agendo sulla struttura dei controlli della CPU ma la velocità di accesso alle istruzioni è almeno altrettanto importante.

La gerarchia di memoria che vede la cache come elemento tampone tra CPU e memoria principale che, a sua volta è un elemento di raccordo verso la memoria di massa (disco fisso) nasce dalla necessità di ottenere il miglior rapporto possibile prezzo/prestazioni.

Il principale scopo di questa gerarchia è quello di creare una memoria che la CPU possa vedere come caratterizzata da un breve tempo di accesso e da una grande capacità.

Ogni livello della gerarchia gioca un ruolo importante. Anche la velocità e l'efficienza dei trasferimenti dei dati fra i vari livelli della gerarchia sono fattori di enorme importanza.

Frazionamento della memoria in moduli

È utile che i trasferimenti da o verso le unità più veloci possano essere effettuati alla stessa frequenza dell'unità più veloce. Ciò non può avvenire se si accede nello stesso modo all'unità più lenta e a quella più veloce. Tale risultato può invece essere raggiunto se si sfrutta il parallelismo nell'organizzazione dell'unità più lenta.

Un modo efficiente per introdurre il parallelismo nella gestione della memoria principale consiste nell'impiego di *un'organizzazione interallacciata*.

Se la memoria principale di un calcolatore è strutturata come una collezione di moduli fisicamente separati, ognuno con il proprio registro buffer degli indirizzi (Address Buffer Register, ABR) e registro buffer dei dati (Data Buffer Register, DBR), le operazioni di accesso alla memoria possono procedere contemporaneamente in più di un modulo. Così, si può incrementare la frequenza aggregata di trasmissione delle parole da o verso il sistema di memoria principale.

Distribuzione degli indirizzi nei moduli (1/3)

Il modo in cui gli indirizzi individuali sono distribuiti attraverso i moduli è importante per stabilire il numero medio di moduli che può essere tenuto occupato mentre le elaborazioni procedono.

Esistono due metodi di distribuzione degli indirizzi.

Nel primo, l'indirizzo di memoria generato dalla CPU viene decodificato nel seguente modo:

I k bit più significativi indicano uno degli n moduli, e gli m bit meno significativi indicano una particolare parola all'interno del modulo. Quando si accede a locazioni consecutive, come accade quando si copia un blocco di dati nella cache, viene coinvolto soltanto un modulo. Nello stesso tempo, tuttavia, dispositivi con possibilità di accesso diretto alla memoria (DMA) possono aver accesso alle informazioni contenute in altri moduli della memoria.

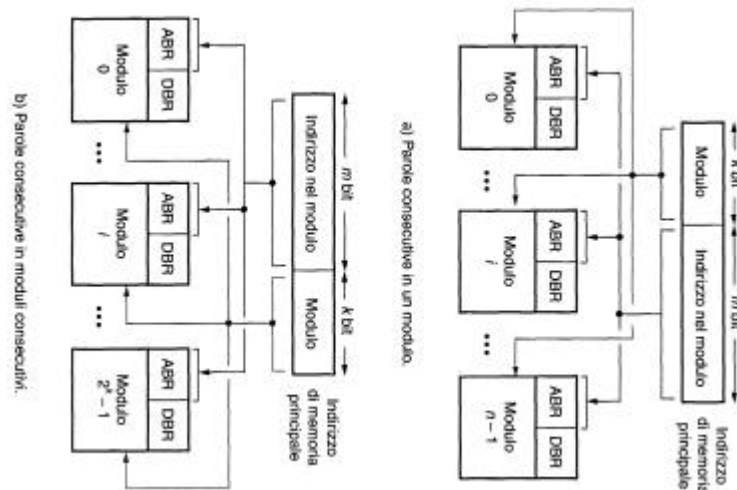
Distribuzione degli indirizzi nei moduli (2/3)

Il secondo e più efficiente modo di indirizzare i moduli è chiamato *interallacciamento della memoria*.

I k bit meno significativi dell'indirizzo di memoria selezionano un modulo, mentre gli m bit più significativi indicano una locazione all'interno del modulo. In questo modo, indirizzi consecutivi sono posizionati in moduli successivi; così facendo, ogni componente del sistema che faccia richiesta di accesso a locazioni di memoria consecutive può tenere occupati vari moduli in un qualsiasi istante. Ciò consente di ottenere sia accessi a un blocco di dati più veloci, sia una più elevata media complessiva di utilizzo della memoria.

Per realizzare la struttura interallacciata, devono esserci 2^k moduli. In caso contrario, ci sono degli stati vuoti associati a locazioni inesistenti nello spazio di indirizzamento della memoria.

Distribuzione degli indirizzi nei moduli (3/3)



Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

33

Effetti della memoria *interallacciata* (1/3)

Analizziamo il tempo necessario per trasferire un blocco di dati dalla memoria principale alla cache nel caso in cui un'operazione di lettura fallisca. Supponiamo di avere una cache con **blocchi di 8 parole**.

Nel caso di un fallimento in fase di lettura, il blocco che contiene la parola desiderata deve essere trasferito nella memoria cache.

Si supponga che sia necessario un ciclo di clock per inviare un indirizzo alla memoria principale. La memoria sia realizzata con chip DRAM che consentono di accedere alla prima parola in 8 cicli di clock, mentre per le parole successive del blocco, sono richiesti soltanto 4 cicli di clock per parola (quando si leggono locazioni consecutive in memorie DRAM, infatti, a partire da una riga di celle, l'indirizzo di riga viene decodificato una sola volta, operazione che consente di risparmiare metà del tempo per ogni accesso successivo). Inoltre, è necessario un altro ciclo di clock per inviare una parola alla cache.

Se la memoria è composta da un unico modulo, il tempo necessario per caricare il blocco desiderato nella cache è

$$1 + 8 + (7 \times 4) + 1 = 38 \text{ cicli}$$

Maggio 2002

Architettura degli elaboratori I - Mod. B - 3. Cache memory

34

Effetti della memoria *interallacciata* (2/3)

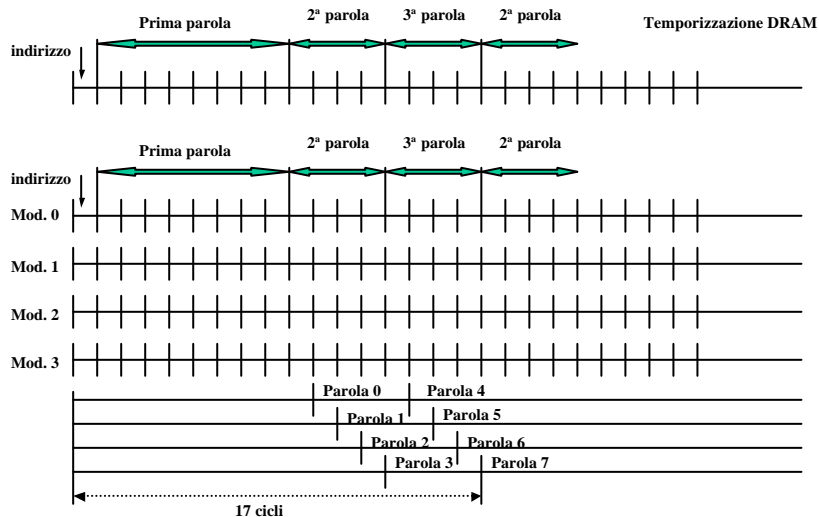
Si supponga ora che la memoria sia costituita da quattro moduli interallacciati.

Quando l'indirizzo iniziale del blocco arriva alla memoria, si accede ai dati richiesti in tutti e quattro i blocchi, utilizzando in ciascuno di essi i bit più significativi dell'indirizzo. Dopo 8 cicli di clock ogni modulo ha pronta una parola di dati nel buffer DBR. Queste parole vengono trasferite nella cache, una parola alla volta, nei 4 cicli di clock successivi. Durante questo tempo, si accede alla parola successiva di ogni modulo, e i 4 cicli di clock seguenti consentono di portare i dati letti nella cache. Di conseguenza, il tempo complessivo necessario per caricare il blocco dalla memoria principale interallacciata alla cache è

$$1 + 8 + 4 + 4 = 17 \text{ cicli}$$

Quindi, l'utilizzo della *struttura interallacciata* consente di ridurre il tempo di trasferimento di un fattore superiore a 2.

Effetti della memoria *interallacciata* (3/3 grafico)



Frequenza di successo e frequenza di fallimento

Un indicatore eccellente dell'efficacia della particolare realizzazione adottata nella gerarchia di memoria è la percentuale di successo ottenuta nell'accesso alle informazioni ai vari livelli della gerarchia che raccorda la CPU al disco rigido, passando per la cache, e la memoria principale.

Il numero di successi indicati come frazione di tutti gli accessi provati viene chiamato ***frequenza di successo***, mentre la ***frequenza di fallimento*** rappresenta il numero di fallimenti in rapporto al numero totale di accessi effettuati.

Teoricamente, se la frequenza di successo fosse 1 a tutti i livelli, l'intera gerarchia di memoria, dal punto di vista della CPU, apparirebbe come una singola unità, **con il tempo di accesso di una cache sul chip del processore ma con le dimensioni di un disco magnetico.**

Quanto ci si avvicini a questo ideale dipende in misura notevole dalla frequenza di successo che si ottiene ai diversi livelli della gerarchia.

Frequenze di successo elevate, ben oltre il 90%, sono essenziali per calcolatori a prestazioni elevate.

Penalità di fallimento

Le prestazioni sono influenzate negativamente dalle azioni che devono essere effettuate quando l'accesso a un dato fallisce.

Il tempo addizionale necessario per portare le informazioni desiderate nella cache è chiamato ***penalità di fallimento***.

Questa penalità si riflette, in ultima analisi, sull'intervallo di tempo durante il quale la CPU rimane in una situazione di stallo a causa della mancata disponibilità delle istruzioni o dei dati necessari per l'esecuzione.

In generale, **la penalità di fallimento è il tempo richiesto per portare il blocco di dati desiderato da un'unità più lenta a una più veloce nella gerarchia di memoria.**

Essa viene ridotta se si realizzano e si adottano meccanismi efficienti per il trasferimento dei dati fra i vari livelli della gerarchia di memoria. Abbiamo visto come una memoria interallacciata possa ridurre in modo significativo la penalità di fallimento tra cache e memoria principale.

Incremento di prestazioni con la cache (1/3)

Valutiamo ora l'impatto della cache sulle prestazioni complessive del sistema di calcolo. Sia h la frequenza di successo, M la penalità di fallimento, ossia il tempo necessario per accedere alle informazioni nella memoria principale, e C il tempo per accedere alle informazioni nella memoria cache. Il tempo medio di accesso sperimentato dalla CPU è

$$t_{ave} = hC + (1 - h)M$$

Consideriamo un *processore veloce* senza cache e con *memoria principale* DRAM, per il quale sono necessari 10 cicli di clock per ogni accesso in lettura alla memoria.

Si supponga ora che allo stesso sistema sia aggiunta una memoria cache che contiene blocchi di otto parole e che la memoria principale sia interallacciata. In questo caso, come abbiamo calcolato in precedenza, sono necessari 17 cicli per caricare un blocco nella cache.

Incremento di prestazioni con la cache (2/3)

Si ipotizzi che il 30% delle istruzioni in un programma tipico effettuino un'operazione di scrittura o di lettura, e che la frequenza di successo sia il 95% per le istruzioni e il 90% per i dati.

Si supponga, inoltre, che la penalità di fallimento sia la stessa per operazioni di scrittura e operazioni di lettura.

Con le ipotesi fatte, i tempi nei due casi sono:

$$\frac{\text{Senza - cache}}{\text{Con - cache}} = \frac{130 \times 10}{100(0,95 \times 1 + 0,05 \times 17) + 30(0,9 \times 1 + 0,1 \times 17)} = 5,04$$

Incremento di prestazioni con la cache (3/3)

Possiamo anche valutare quanto sia efficace la memoria cache con le prestazioni ipotizzate se confrontata con una cache ideale con una frequenza di successo pari al 100% (in tal caso, tutti gli accessi alla memoria richiedono un solo ciclo di clock).

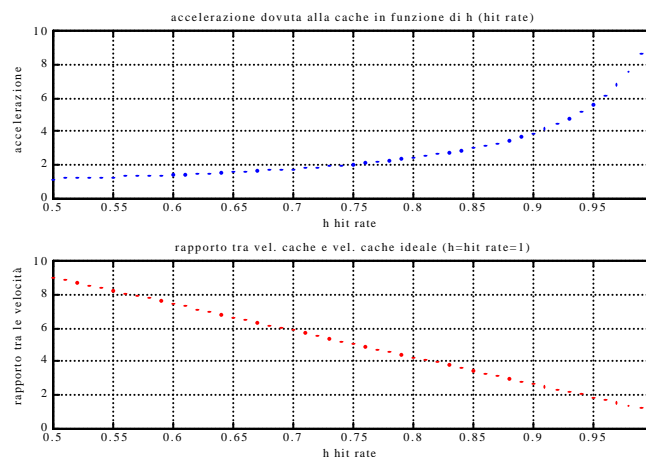
$$\frac{100(0,95 \times 1 + 0,05 \times 17) + 30(0,9 \times 1 + 0,1 \times 17)}{130} = 1,98$$

In pratica la cache ipotizzata permette alla CPU di funzionare come se avesse una memoria principale, basata su DRAM, con tempi d'accesso che sono solo il doppio di quelli della cache.

Nell'esempio abbiamo supposto la frequenza di successo diversa per istruzioni e dati. Sebbene frequenze di successo con valori superiori a 0.9 siano ottenibili per entrambi, la frequenza di successo per le istruzioni è di solito maggiore di quella per i dati. Questi valori dipendono dal progetto della cache e dal profilo di accesso a istruzioni e dati che caratterizza il programma in esecuzione.

Accelerazione dovuta alla memoria cache

Facendo variare il tasso di successo h (uguale per dati ed istruzioni) le due formule precedenti danno i seguenti diagrammi



Una cache per le istruzioni ed una per i dati?

Per i motivi che abbiamo più volte ricordato il posto migliore in cui collocare una cache è il chip della CPU. Sfortunatamente, però, si trova posto solo per cache piccole.

Tutti i processori con prestazioni elevate hanno una cache "on chip". Alcuni produttori hanno deciso di realizzare due cache separate, una per le istruzioni e un'altra per i dati, come nei processori 68040 e PowerPC 604. In altri casi è stata adottata una sola cache per istruzioni e dati, come nel processore PowerPC 601.

Una sola cache per istruzioni e dati in teoria dovrebbe avere una frequenza di successo superiore, poiché offre una maggiore flessibilità nella memorizzazione di nuove informazioni. Tuttavia, se si utilizzano cache separate, è possibile accedere contemporaneamente a entrambe le prestazioni cache, aumentando così il parallelismo e, di conseguenza, le prestazioni della CPU. **Lo svantaggio delle cache separate è che l'incremento del grado di parallelismo è accompagnato da circuiti molto più complessi.**

Una cache secondaria

Poiché le dimensioni della memoria cache sul chip della CPU sono limitate, una buona strategia consiste nell'utilizzare tale cache come **cache primaria**.

Una cache secondaria esterna, costruita con chip SRAM, viene poi aggiunta per ottenere la capacità desiderata.

In questo caso la cache primaria deve essere progettata in modo tale da consentire accessi molto veloci da parte della CPU, visto che il suo tempo di accesso ha un'influenza significativa sulla frequenza di clock della CPU. Ma non si può accedere a una memoria cache alla stessa velocità con cui si accede al banco di registri interni della CPU, perché la cache ha dimensioni maggiori ed è più complessa.

Un modo pratico di accelerare l'accesso alla cache è quello di **consentire l'accesso a più di una parola contemporaneamente**. Questa tecnica è utilizzata sia nel Processore 68040 sia nel PowerPC. Si accede alla cache del Processore 68040 leggendo (o scrivendo) 8 byte per volta. Per quella del PowerPC l'accesso ai dati avviene leggendo o scrivendo 16 byte per volta.

Considerazioni generali sulla cache secondaria

La cache secondaria può essere anche notevolmente più lenta, ma dovrebbe essere molto più grande di quella primaria in modo tale da garantire una frequenza di successo elevata negli accessi. La velocità di questa cache è meno critica perché influenza soltanto la penalità di fallimento della cache primaria.

Una workstation può avere una *cache primaria* con una capacità di decine di kilobyte e una *cache secondaria* di diversi megabyte.

La presenza di una cache secondaria riduce ulteriormente l'impatto della velocità della memoria principale sulle prestazioni del calcolatore.

Accesso in un sistema con due livelli di cache

Il tempo medio di accesso visto dalla CPU è:

$$t_{\text{ave}} = h_1 C_1 + (1 - h_1) h_2 C_2 + (1 - h_1)(1 - h_2)M$$

dove i parametri sono definiti come segue:

h_1 = frequenza di successo nella cache primaria;

h_2 = frequenza di successo nella cache secondaria;

C_1 = tempo di accesso alle informazioni nella cache primaria;

C_2 = tempo di accesso alle informazioni nella cache secondaria;

M = tempo di accesso alle informazioni nella memoria principale.

Il numero di fallimenti di accesso nella cache secondaria, dato dal termine $(1 - h_1)(1 - h_2)$, dovrebbe essere basso. Se h_1 e h_2 raggiungono entrambi valori dell'ordine del 90%, allora il numero di fallimenti sarà inferiore all'1% degli accessi alla memoria da parte della CPU. Quindi, la penalità di fallimento M sarà meno critica dal punto di vista delle prestazioni.

CACHE - trasferimento da memoria

- Ipotesi: memoria che impiega $2T_L$ CC per trasferire la prima parola al MBR e solo T_L per trasferire le successive; cache con blocchi di n parole.
- Per scrivere 1 blocco sono necessari:
 - 1 CC per trasferire l'indirizzo sul MAR,
 - $2T_L$ CC per mettere il contenuto della prima parola sul MBR,
 - $1T_L$ CC per ognuna delle successive $n-1$ parole.
- In totale:
 - $T_A = 1 + 2T_L + (n-1)T_L + 1 = 2 + (n + 1) T_L$

CACHE - memoria interallacciata (1/2)

- Ipotesi: ora la memoria è interallacciata, siano 2^k moduli da 2^m parole (indirizzo suddiviso in $k + m$ bit).
- Per scrivere 1 blocco sono necessari:
 - 1 CC per trasferire l'indirizzo sul MAR
 - $2T_L$ CC per mettere il contenuto della prime 2^k parole sugli MBR di ciascun modulo
 - 2^k CC per trasferirle alla cache mentre contemporaneamente si leggono le successive 2^k in T_L CC, che poi verranno trasferite ($T_L \geq 2^k$).
- In totale:
 - $T_B = 1 + 2T_L + [(n-2^k)/2^k] T_L + 2^k = 1 + 2^k + [(n + 2^k) / 2^k] T_L$

CACHE - memoria interallacciata (2/2)

- Conclusione
 - la presenza di memoria interallacciata consente un incremento delle prestazioni per quanto riguarda il trasferimento delle informazioni dalla memoria che può essere valutata definendo il rapporto:

$$• S = T_A / T_B$$

CACHE - penalità di fallimento (1/3)

- Definiamo:
 - p_h probabilità di successo nell'accesso alla cache
 - T_C tempo trasferimento informazioni da cache a CPU
 - T_M tempo trasferimento informazioni da memoria a CPU

- Allora il tempo medio di accesso sarà:

$$T_{av} = p_h T_C + (1 - p_h) T_M .$$

- Da confrontare col tempo che si avrebbe:

- senza cache T_M
- con cache, senza fallimenti T_C

CACHE - penalità di fallimento (2/3)

- Ipotesi:
 - p_{hi} probab. di successo nell'accesso a un'istruzione nella cache
 - p_{hd} probab. di successo nell'accesso a un dato nella cache
 - p_d probab. che un'istruzione richieda l'accesso ad un dato
 - T_C tempo accesso informazioni esistenti nella cache
 - T_M tempo accesso informazioni in memoria
- Allora il tempo medio di accesso sarà:
$$T_{av} = \{ [p_{hi} T_C + (1 - p_{hi}) T_M] + p_d [p_{hd} T_C + (1 - p_{hd}) T_M] \} .$$
- Da confrontare col tempo che si avrebbe:
 - senza cache $(1 + p_d) T_M$
 - con cache, senza fallimenti $(1 + p_d) T_C$

CACHE - penalità di fallimento (3/3)

- Ipotesi: inseriamo una seconda cache
- Definiamo:
 - p_{1h} probabilità di successo nell'accesso alla cache primaria
 - p_{2h} probabilità di successo nell'accesso alla cache secondaria
 - T_{1C} tempo accesso informazioni nella cache primaria
 - T_{2C} tempo accesso informazioni nella cache secondaria
 - T_M tempo accesso informazioni in memoria
- Allora il tempo medio di accesso sarà:

$$T_{av} = p_{1h} T_{1C} + (1 - p_{1h}) p_{2h} T_{2C} + (1 - p_{1h})(1 - p_{2h}) T_M$$