Memorie Cache

Introduzione

Le memorie cache si inseriscono tra la CPU e la memoria centrale con lo scopo di diminuire il tempo medio di accesso ai dati memorizzati. Questo obiettivo viene conseguito sia realizzando memorie cache di ridotta capacità ma con tempi di accesso bassi rispetto alle memorie centrali, sia sfruttando il fatto, statistico, che accessi consecutivi (o comunque ravvicinati nel tempo) tendono ad essere fatti su locazioni adiacenti (o comunque ravvicinate nello spazio di indirizzamento della memoria).

Lo sfruttamento di questo comportamento avviene trasferendo nella cache, in occasione di un primo accesso alla memoria centrale, non soltanto il dato interessato dalla lettura ma un intero suo "intorno", cioè anche il contenuto delle locazioni adiacenti a quella verso la quale l'accesso era indirizzato. Tale "intorno" viene memorizzato in cache in quello che prende il nome di "linea" o "blocco" e la cui dimensione, solitamente di poche decine di byte, è fissa e costante per ciascuna cache.

Una volta riempito un blocco di cache nel modo appena descritto, una successiva lettura in memoria, che interessi una locazione il cui contenuto risulti già presente nel blocco appena letto, avrà un tempo di accesso inferiore rispetto ad un normale accesso in memoria centrale poiché, come accennato prima, le cache hanno tempi di accesso più brevi.

Il prezzo da pagare per questo miglioramento di prestazioni consiste nel maggior tempo di accesso necessario nel caso in cui il dato a cui si vuole accedere non sia memorizzato in cache: in tal caso, infatti, come visto prima, l'accesso alla memoria centrale non si limita ad una sola lettura del dato che ci interessa, ma consiste in una lettura dell'intero blocco di dati adiacenti a quello voluto, per cui in tal caso il tempo impiegato sarà in realtà più lungo di quello proprio per una singola lettura dalla memoria centrale.

Un accesso alla memoria fatto verso un dato che risulta già memorizzato in cache prende il nome di "hit" (= colpito!); similmente, un accesso alla memoria fatto verso un dato che non è memorizzato in cache, che quindi richiede il riempimento di una linea della stessa, prende il nome di "miss" (= mancato!).

Detti t_{hit} e t_{miss} i corrispondenti tempi di accesso, detto t_{mc} il tempo di accesso proprio della memoria centrale, per quanto detto fin qui vale la disuguaglianza $t_{hit} < t_{mc} < t_{miss}$.

Esempio applicativo: cache fully associative

Consideriamo una memoria centrale di 128 byte, contenente i dati rappresentati di seguito, organizzati per comodità in righe di 4 byte. A fianco di ciascuna riga è indicato, in corsivo, l'indirizzo del primo byte della stessa.

```
00h = 0000000b   01h 05h 7Fh 95h
04h = 0000100b   03h 62h ADh 44h
08h = 0001000b   89h 54h 22h 78h
0Ch = 0001100b   37h 12h 72h FFh
10h = 0010000b   00h 48h 33h 6Eh
14h = 0010100b   FEh DEh C1h A0h
18h = 0011000b   00h 48h 33h 6Eh
1Ch = 0011100b   03h 62h 48h 33h
20h = 0100000b   48h 89h 54h 22h
24h = 0100100b   37h 12h 72h 41h
28h = 0101000b   CFh 62h 48h 33h
2Ch = 0101100b   DEh C1h 62h 48h
30h = 0110000b   E0h 48h 89h D1h
34h = 0110100b   00h 48h 33h 1Dh
38h = 0111000b   BCh 37h 12h 72h
3Ch = 0111100b   03h 62h 48h EDh
40h = 1000000b   A5h 89h 54h 22h
44h = 1000100b   03h 62h B7h CCh
48h = 1001000b   48h 86h 52h 23h
4Ch = 1001100b   11h 03h 62h 48h
50h = 1010000b   03h 62h ADh 69h
54h = 1010100b   B0h 01h 05h 7Fh
58h = 1011000b   12h 72h 01h 05h
5Ch = 1011100b   FEh DEh C1h 48h
60h = 1100000b   12h 72h Ffh 52h
64h = 1100100b   62h ADh 44h 00h
68h = 1101000b   03h 62h 07h 91h
6Ch = 1101100b   33h 89h 54h 22h
70h = 1110000b   34h 97h 77h 58h
74h = 1110100b   33h 89h 07h 50h
78h = 1111000b   05h 93h 06h 32h
7Ch = 1111100b   31h 41h 59h 26h
```

Niente paura, non si tratta di dati significativi, anzi li ho messi a caso e francamente non passava più... neanche ci serviranno tutti... vabbè, andiamo avanti.

Una piccola cosa che ci fa comodo notare a questo punto: per come sono stati strutturati i dati, in righe da 4 byte ciascuna, si può pensare l'indirizzo a 7 bit di ciascun byte come costituito da una parte che identifica la riga e da una parte che identifica la colonna in cui il dato si trova. La parte che identifica la riga è costituita dai bit più significativi, tanti quanti ne sono necessari per codificare tutte le righe: in questo caso abbiamo 128/4 = 32 = 2^5 righe, quindi i 5 bit più significativi identificano la riga. I rimanenti 2 bit (i meno significativi) identificheranno la colonna, ed infatti 2^2 = 4 colonne. Tutto questo è abbastanza evidente considerando gli indirizzi dei primi byte di ciascuna riga scritti in binario nella tabella precedente. Appartenendo tutti alla prima colonna, è logico che gli ultimi 2 bit siano sempre a 0, mentre i 5 bit più significativi assumono tutte le configurazioni da 00000b a 11111b man mano che si scende nella tabella.
Se la tabella fosse stata organizzata in righe da 8 byte, queste sarebbero state soltanto 16, portando l'indirizzo a 7 bit di ciascuna locazione ad essere costituito da 4 bit di riga e 3 di colonna.

Consideriamo ora una cache fully associative costituita da 8 linee da 4 byte ciascuna, per una capacità complessiva di 32 byte. Inizialmente sarà vuota, ce la possiamo immaginare così (le X rappresentano cifre esadecimali ancora da riempire e i puntini cifre binarie ancora non assegnate):

(cache directory)	(linee)
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh

Immaginiamo ora che il programma in esecuzione sulla CPU voglia leggere il byte di indirizzo 0000000b, il primo in assoluto della memoria centrale. Poiché la cache è vuota, questa lettura darà luogo ad una condizione di "miss", cioè renderà necessario il riempimento di una linea di cache con il dato letto e quelli ad esso adiacenti andandoli a prendere in memoria centrale. In particolare, nella prima linea della cache andrà a finire la prima riga di 4 byte che avevamo scritto nel contenuto della memoria centrale, cioè:

(tag)	(linee)
00000b	01h 05h 7Fh 95h
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh
.....*b*	XXh XXh XXh XXh

Nella riga della cache directory corrispondente alla prima linea della cache andrà a finire il cosiddetto "tag", ossia un insieme di bit necessario e sufficiente alla cache per capire da che parte della memoria centrale è stato prelevato il contenuto della linea. Nel caso che stiamo considerando questi bit sono i 5 bit più significativi dell'indirizzo a cui abbiamo acceduto, gli stessi che, nella tabella della memoria centrale, venivano usati per identificare la riga.
Dopo il riempimento della linea di cache, alla CPU verrà passato soltanto il dato che le interessava, cioè il valore 01h (evidenziato in verde nella tabella)

A questo punto supponiamo che la CPU effettui una lettura all'indirizzo 0000010b = 02h. I 5 bit più significativi dell'indirizzo (00000b, che costituiscono il tag, come abbiamo detto) vengono confrontati con tutti tag memorizzati nella cache directory, per controllare se il blocco di memoria centrale di cui fa parte il dato che vogliamo leggere è già stato salvato in cache a causa di letture precedenti. In tal caso avremmo una condizione di "hit", perché potremo leggere il dato voluto direttamente dalla cache senza dover accedere alla memoria centrale. Il tag cercato (00000b) viene individuato nella prima linea di cache, che infatti contiene la copia dei primi 4 byte della memoria centrale e quindi anche quello voluto, di indirizzo 02h, cioè il terzo della linea di cache (evidenziato in giallo), che viene restituito alla CPU come risultato della lettura.

Più precisamente: una volta individuata, mediante il tag, la linea di cache dalla quale il dato deve essere prelevato, la posizione del byte da leggere tra i 4 che costituiscono la linea è data dai rimanenti 2 bit dell'indirizzo, quelli meno significativi che non sono stati considerati per la determinazione del tag. Nel caso di accesso all'indirizzo 0000000b il tag è, come abbiamo visto, 00000b mentre la parte rimanente, che prende il nome di offset, vale 00b = 0 e quindi corrisponde al primo byte della linea. Analogamente, con riferimento all'indirizzo 0000010b, il tag rimane lo stesso, cambia però l'offset che diventa 10b = 2, cioè corrispondente al terzo byte della linea.

Immaginiamo ora di voler leggere il byte di indirizzo 57h = 1010111b. Il tag di questo indirizzo è 10101b , che non è presente nella cache directory nella situazione attuale. Avremo allora una nuova condizione di "miss" che provocherà il riempimento di una nuova linea di cache come segue, prelevandola dalla memoria a partire dall'indirizzo 1010100b = 54h, cioè dal primo byte della riga di 4 che andremo a copiare in cache.
La configurazione della cache dopo questa nuova lettura sarà la seguente, dove il dato evidenziato in verde è quello ritornato alla CPU come risultato della lettura, il quarto della nuova linea memorizzata in cache, cioè quello di offset = 11b = 3.

(tag)	(linee)
00000b	01h 05h 7Fh 95h
10101b	B0h 01h 05h 7Fh
.....b	XXh XXh XXh XXh
.....b	XXh XXh XXh XXh
.....b	XXh XXh XXh XXh
.....b	XXh XXh XXh XXh
.....b	XXh XXh XXh XXh
.....b	XXh XXh XXh XXh

Analogamente a prima, data questa situazione della memoria cache, una lettura dall'indirizzo 1010101b = 55h produrrà una condizione di "hit" poiché il dato (evidenziato in giallo) è già memorizzato in cache con tag = 10101b e offset = 01b = 1, cioè è il secondo della linea di cache di cui fa parte.

N.B. La lunghezza dei tag da memorizzare nella cache directory NON dipende dalla dimensione della cache, ma dipende dalla dimensione della memoria centrale con la quale la cache lavora. Se dovessimo usare la stessa cache vista finora per lavorare con una memoria centrale di 256 byte, quindi di capacità raddoppiata rispetto a quella considerata fino qui, le righe da 4 byte di suddetta memoria centrale sarebbero 256/4 = 64, corrispondenti alla necessità di 6 bit di tag (non più soltanto 5) per essere identificate una volta salvate in cache.

Le linee di cache si riempiono man mano che vengono effettuati accessi in memoria che producono "miss". Dopo altre 6 di esse, avremo riempito tutte le linee della cache, per esempio come illustrato di seguito (spero di non essermi sbagliato a copiare i valori, dovrebbero essere giusti...)

(tag)	(linee)
00000b	01h 05h 7Fh 95h
10101b	B0h 01h 05h 7Fh
10000b	A5h 89h 54h 22h
10010b	48h 86h 52h 23h
00100b	00h 48h 33h 6Eh
00010b	89h 54h 22h 78h
10011b	11h 03h 62h 48h
00011b	37h 12h 72h Ffh

Cosa succede a questo punto se ho di nuovo una lettura che produce una "miss"? Niente di particolare, semplicemente rimpiazzerò una delle linee di cache (e relativo tag) con il nuovo blocco letto dalla memoria centrale.
E se l'istruzione successiva mi richiede un dato che era memorizzato nella linea di cache che ho appena “buttato via”, sovrascrivendola col nuovo blocco? Che sfi... ehm... pazienza, vorrà dire che ci sarà un'altra "miss" e un'altra delle linee della cache verrà rimpiazzata con quella da cui si ha bisogno di leggere, indipendentemente dal fatto che sia un blocco che era già stato in cache o meno. Diciamo che, piuttosto che rimpiazzare semplicemente le linee una per volta ripartendo dall'inizio, risulta più sagace sostituire, ad esempio, la linea di cache che è stata acceduta meno di frequente o meno di recente... le cache più evolute implementano questo tipo di algoritmi per la sostituzione delle linee, in modo da cercare di minimizzare il numero di "miss", tenendo in cache le linee più utili e sbarazzandosi delle altre al momento di doverle rimpiazzare.

Una volta compreso tutto quanto fino a qui, è importante ricordare che, data una memoria centrale con N_A bit di indirizzo e una cache fully associative con L linee da N byte ciascuna, si ha:
- capacità della cache (in byte): N*L
- numero di bit di offset: $N_O = \log_2 N$
- numero di bit di tag: $N_T = N_A - N_O$
- capacità necessaria per la cache directory (in bit) = N_T*L

Con il crescere delle dimensioni delle cache e delle memorie centrali, la capacità necessaria per la cache directory è divenuta sempre maggiore, al punto da rendere necessarie nuove soluzioni per rendere più rapide le ricerche dei tag al loro interno. Per questa ragione sono state introdotte le cache one way associative, nelle quali, a scapito di un tasso di "miss" più elevato, la rapidità di ricerca nella cache directory è massima.

Esempio applicativo: cache one way associative

La rappresentazione astratta di una cache one way associative è leggermente diversa rispetto a quella di una fully associative. Ispirandoci a quella considerata fin qui, con 8 linee da 4 byte ciascuna, possiamo pensare che la cache one way associative corrispondente sia fatta così (quando è vuota):

(tag)	(index)	(linee)			
..b	*000b*	XXh	XXh	XXh	XXh
..b	*001b*	XXh	XXh	XXh	XXh
..b	*010b*	XXh	XXh	XXh	XXh
..b	*011b*	XXh	XXh	XXh	XXh
..b	*100b*	XXh	XXh	XXh	XXh
..b	*101b*	XXh	XXh	XXh	XXh
..b	*110b*	XXh	XXh	XXh	XXh
..b	*111b*	XXh	XXh	XXh	XXh

La differenza che salta subito all'occhio è che i 5 bit associati a ciascuna linea non sono più tutti variabili: alcuni di essi (i meno significativi) sono fissi e costituiscono il cosiddetto "index". Essi sono in numero sufficiente ad identificare tutte le linee della cache (in questo caso $8 = 2^3$ linee di cache => 3 bit di index). Tutto ciò comporta che, anche nel caso in cui la cache sia completamente vuota, quando andiamo a riempirne una linea non siamo liberi di decidere in quale di esse poter depositare i dati, ma siamo costretti a salvarli in quella la cui parte di index corrisponde alla parte di index dell'indirizzo da cui è stata fatta la lettura.
Mi spiego subito meglio impiegando le letture di esempio fatte in precedenza: se leggiamo dall'indirizzo 0000000b, i 5 bit più significativi (00000b) sono quelli che identificano la linea di cache nel quale sarà memorizzato il blocco e la linea di memoria centrale dalla quale li andremo a prelevare. Di questi 5, i 3 meno significativi (000b) costituiscono l'index, per cui saremo costretti a salvare i 4 byte letti dalla memoria centrale nella prima linea della cache, il cui index è appunto 000b, scrivendo nella parte di tag solamente i 2 bit rimanenti che servono per identificare completamente la riga di memoria centrale da cui proviene la linea di cache appena letta.
Avremo quindi questa situazione (in verde il byte restituito alla CPU come risultato della lettura):

(tag)	(index)	(linee)			
00b	*000b*	01h	05h	7Fh	95h
..b	*001b*	XXh	XXh	XXh	XXh
..b	*010b*	XXh	XXh	XXh	XXh
..b	*011b*	XXh	XXh	XXh	XXh
..b	*100b*	XXh	XXh	XXh	XXh
..b	*101b*	XXh	XXh	XXh	XXh
..b	*110b*	XXh	XXh	XXh	XXh
..b	*111b*	XXh	XXh	XXh	XXh

La lettura successiva, all'indirizzo 02h = 0000010b, genera una "hit" in cache in quanto i 5 bit più significativi dell'indirizzo voluto (00000b) coincidono con la concatenazione di tag e index della prima linea di cache. Da notare che la verifica della presenza o meno dei dati in cache in questa situazione è immediata: noto l'index dell'indirizzo voluto (000b, come prima), se il relativo tag nella linea di cache corrisponde ai 2 bit più significativi dell'indirizzo, allora il blocco memorizzato in quella linea è quello dal quale possiamo prelevare il terzo byte (offset = 10b = 2) per restituirlo come risultato della lettura (evidenziato in giallo). Se il tag non corrispondesse si avrebbe automaticamente una "miss", in quanto nessuna delle altre linee della cache potrebbe contenere quello stesso blocco avendo tutte index diverso da quello cercato..

Proseguendo con le letture dell'esempio precedente, segue l'accesso alla memoria all'indirizzo 57h = 1010111b. Il tag di

questo indirizzo è 10b (bit più significativi), l'index è 101b (bit centrali) e l'offset è 11b (bit menò significativi).
Poiché la linea di cache associata all'index 101b è vuota, si ha una "miss" con la quale viene riempita ed il suo tag viene impostato al tag dell'indirizzo acceduto, avendo così la seguente configurazione:

(tag)	(index)	(linee)			
00b	*000b*	01h	05h	7Fh	95h
..b	*001b*	XXh	XXh	XXh	XXh
..b	*010b*	XXh	XXh	XXh	XXh
..b	*011b*	XXh	XXh	XXh	XXh
..b	*100b*	XXh	XXh	XXh	XXh
10b	*101b*	B0h	01h	05h	7Fh
..b	*110b*	XXh	XXh	XXh	XXh
..b	*111b*	XXh	XXh	XXh	Xxh

Al solito, in verde è evidenziato il byte restituito alla CPU come risultato della lettura. Seguendo lo stesso meccanismo precedentemente esposto, una lettura all'indirizzo 55h = 1010101b ritorna il byte evidenziato in giallo, esattamente come accadeva con la cache fully associative.

Esaminiamo ora con attenzione la configurazione della cache fully associative piena, riportata qualche pagina addietro: si nota che il tag della terza linea di cache è 10000b. Poniamo il caso che tale blocco sia stato caricato in cache a causa di una lettura all'indirizzo 1000001b. Se la stessa lettura fosse stata invece effettuata impiegando la cache one way associative così come si trova nella situazione descritta fino a qui, il blocco letto dalla memoria centrale come risultato della "miss" sarebbe stata memorizzata, in cache, NON in una delle linee libere, sebbene ce ne siano, ma avrebbe dovuto essere memorizzata forzatamente nella prima linea della stessa, rimpiazzando quella già esistente, in quanto l'index dell'indirizzo è 000b.

Analizziamo in dettaglio quello che accade: l'indirizzo da leggere è 1000001b, quindi tag=10b, index = 000b, offset = 01b. In corrispondenza dell'index = 000b nella cache directory abbiamo come tag il valore 00b, che non corrisponde a quello dell'indirizzo che vogliamo leggere. Siamo allora in condizione di "miss": dobbiamo leggerci dalla memoria centrale i 4 byte a partire dall'indirizzo 1000000b e rimpiazzarli alla prima linea della cache, sostituendone anche il tag, che assume il valore 10b in accordo con l'indirizzo dal quale sono stati prelevati i dati del blocco. Nel complesso siamo arrivati alla situazione che segue:

(tag)	(index)	(linee)			
10b	*000b*	A5h	89h	54h	22h
..b	*001b*	XXh	XXh	XXh	XXh
..b	*010b*	XXh	XXh	XXh	XXh
..b	*011b*	XXh	XXh	XXh	XXh
..b	*100b*	XXh	XXh	XXh	XXh
10b	*101b*	B0h	01h	05h	7Fh
..b	*110b*	XXh	XXh	XXh	XXh
..b	*111b*	XXh	XXh	XXh	Xxh

Senza continuare oltre con l'esempio, il concetto fondamentale è che, a fronte di una individuazione immediata dell'eventuale linea di cache che contiene il blocco desiderato, il cui effetto è quindi quello di tendere ad abbassare il tempo di accesso, corrisponde una maggiore frequenza di "miss" e rimpiazzamenti di linee di cache che tende ad innalzarlo. Una soluzione di compromesso tra le cache fully associative e quelle one way associative è quella rappresentata dalle cache n-way associative, trattate di seguito.

In merito alle cache one way associative, detto N_A il numero di bit di indirizzo della memoria centrale, L il numero di linee della cache ed N il numero di byte memorizzati in ciascuna di esse, è utile ricordare che:
- capacità della cache (in byte): N*L
- numero di bit di offset: $N_O = \log_2 N$
- numero di bit di index: $N_I = \log_2 L$
- numero di bit di tag: $N_T = N_A - N_I - N_O$
- capacità necessaria per la cache directory (in bit) = N_T*L

Esempio applicativo: cache n-way associative

Le cache n-way associative cercano di salvare i pregi dei due tipi di cache visti prima limitandone i difetti. Si tratta di una soluzione di compromesso, dove anziché esistere una sola linea per ciascun index come nelle one way associative, le linee con lo stesso index sono appunto n (dove n è una potenza di 2).
Ogni gruppo di n linee identificate dallo stesso index prende il nome di "set", sempre riconducendoci alla cache di 8 linee da 4 byte, questa volta considerandola 2-way associative, avremo la seguente struttura:

(tag)	(index)	(linee)
...b	*00b*	XXh XXh XXh XXh
...b	*00b*	XXh XXh XXh XXh
...b	*01b*	XXh XXh XXh XXh
...b	*01b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh

Come si nota, ora a ciascun index non corrisponde più una sola linea di cache ma un set di n linee, gestite tra loro come se si trattasse di una cache fully associative, quindi in questo caso potrò fare 2 letture ad indirizzi con stesso index ma tag diverso prima di dover rimpiazzare una linea di cache. Andiamo con ordine e, per capirci meglio, riprendiamo brevemente le solite letture di esempio dalla memoria centrale. Siete ancora lì o c'era qualcosa di più interessante in TV?

Poniamo prima attenzione al fatto che, se prima i bit di index erano 3 perché dovevano identificare ciascuna delle 8 linee della cache, questa servono soltanto per identificare uno dei 4 set da 2 linee in cui la cache è suddivisa, quindi sono ridotti a 2 bit di index. I bit di offset sono rimasti 2 in quanto le linee della cache sono ancora da 4 byte, per cui i 3 bit rimanenti per arrivare a formare i 7 di indirizzo sono i bit di tag.

La lettura all'indirizzo 0000000b (tag = 000b, index = 00b, offset = 00b) riempie una linea del primo set (quello di index 00b), ad esempio la prima, come illustrato di seguito, e ritorna il byte evidenziato in verde. La successiva lettura all'indirizzo 0000010b (tag = 000b, index = 00b, offset = 10b), risulta in una "hit" in cache che interessa la linea appena riempita, ritornando il byte evidenziato in giallo. Il tag di detta linea viene logicamente impostato a 000b dalla prima lettura, in accordo col tag dell'indirizzo a cui si è acceduto.

(tag)	(index)	(linee)
000b	*00b*	01h 05h 7Fh 95h
...b	*00b*	XXh XXh XXh XXh
...b	*01b*	XXh XXh XXh XXh
...b	*01b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh

La lettura seguente, all'indirizzo 57h = 1010111b (tag = 101b, index = 01b, offset = 11b) risulta in una "miss" a seguito della quale viene riempita la prima linea caratterizzata da index = 01b, il cui tag viene impostato a 101b (quello dell'indirizzo a cui si è acceduto). La successiva lettura all'indirizzo 1010101b (tag = 101b, index = 01b, offset = 01b) genera una "hit" nella stessa linea di cache e come visto anche nei casi precedenti. La struttura della cache a seguito di queste due nuove letture è quella illustrata di seguito, in verde e giallo sono sempre evidenziati i due byte letti nel rispettivo ordine.

(tag)	(index)	(linee)
000b	*00b*	01h 05h 7Fh 95h
...b	*00b*	XXh XXh XXh XXh
101b	*01b*	B0h 01h 05h 7Fh
...b	*01b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*10b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh
...b	*11b*	XXh XXh XXh XXh

Abbiamo ora la lettura all'indirizzo 41h = 1000001b (tag = 100b, index = 00b, offset = 01b), che nel caso della cache one way associative causava il rimpiazzamento del blocco letto in occasione della prima lettura. In questo caso invece, sebbene sia già stata memorizzata in cache una linea avente index = 00b, dato che ve ne è un'altra con lo stesso index ancora non allocata, questa nuova "miss" può salvare i dati in cache nella seconda linea a disposizione del primo set, quello di index 00b, ed impostarne i bit del tag di conseguenza, come illustrato di seguito:

(tag)	(index)	(linee)			
000b	*00b*	01h	05h	7Fh	95h
100b	*00b*	A5h	89h	54h	22h
101b	*01b*	B0h	01h	05h	7Fh
...b	*01b*	XXh	XXh	XXh	XXh
...b	*10b*	XXh	XXh	XXh	XXh
...b	*10b*	XXh	XXh	XXh	XXh
...b	*11b*	XXh	XXh	XXh	XXh
...b	*11b*	XXh	XXh	XXh	XXh

Va da sé che, sebbene questo funzionamento riduca il numero di "miss" rispetto ad una cache one way associative, esso risulta comunque sempre maggiore rispetto a quello di una cache fully associative: considerando la situazione a cui siamo arrivati con questa cache 2 way associative, una successiva lettura all'indirizzo 1110000b (tag = 111b, index = 00b, offset = 00b) richiederebbe comunque il rimpiazzamento di una delle due linee del primo set, mentre se si trattasse di una fully associative il nuovo blocco potrebbe essere tranquillamente memorizzato in una delle 5 linee rimaste libere avendo cura di salvarne il tag.

Raddoppiando ulteriormente il numero di vie associative, passando cioè da 2 a 4, ci si avvicina ulteriormente al comportamento di una cache fully associative allontanandosi da quello di una one way associative. La struttura risultante (vuota) sarebbe infatti:

(tag)	(index)	(linee)			
....b	*0b*	XXh	XXh	XXh	XXh
....b	*0b*	XXh	XXh	XXh	XXh
....b	*0b*	XXh	XXh	XXh	XXh
....b	*0b*	XXh	XXh	XXh	XXh
....b	*1b*	XXh	XXh	XXh	XXh
....b	*1b*	XXh	XXh	XXh	XXh
....b	*1b*	XXh	XXh	XXh	XXh
....b	*1b*	XXh	XXh	XXh	Xxh

I bit di tag sono passati da 3 a 4, mentre quelli di index sono diminuiti ad 1. E' evidente che un ulteriore raddoppio delle vie associative ci porterebbe ad avere 5 bit di tag e nessuno di index, situazione esattamente corrispondente a quella della cache fully associative dalla quale siamo partiti.
Volendo generalizzare, cache one way associative e fully associative possono essere pensate come casi particolari della cache n-way associative, la prima con n = 1, la seconda con n = L, dove L è il numero di linee della cache.

I dati che è utile memorizzare relativamente alle cache n-way associative sono, detto N_A il numero dei bit di indirizzo della memoria centrale, L il numero di linee della cache, N il numero di byte per ciascuna linea di cache ed n il numero di vie associative, cioè il numero di linee che costituiscono un set:
- capacità della cache (in byte): N*L
- numero di bit di offset: $N_O = \log_2 N$
- numero di set $N_S = L/n$
- numero di bit di index: $N_I = \log_2 N_S = \log_2 L - \log_2 n$
- numero di bit di tag: $N_T = N_A - N_I - N_O$
- capacità necessaria per la cache directory (in bit) = N_T*L

Ok, credo di aver detto tutto quel che dovevo dire, adesso mi prendo una pausa, faccio un giretto poi rileggo...
Fede