Gerarchia di Memorie

Corso di Calcolatori Elettronici A

2007/2008

Sito Web:http://prometeo.ing.unibs.it/quarella

Prof. G. Quarella

prof@quarella.net

Gerarchia di Memorie

Obiettivo: creare l'illusione di avere a disposizione una memoria veloce ed illimitata. In pratica si vuole poter disporre di tanta memoria quanta ne sia disponibile con la tecnologia meno costosa con una velocità di accesso pari a quella della tecnologia più veloce.

Tecnologia di memorizzazione	Tempo di accesso tipico	$ per GByte nel 2004
SRAM	0,5-5 ns	$4.000-$10.000
DRAM	50-70 ns	$100 - $200
Dischi magnetici	5-20 milioni ns	$0,5 - $2

Gerarchia di Memorie

Principio di località

- **Località temporale**

  Tendenza a riusare i dati a cui si è fatto riferimento di recente.

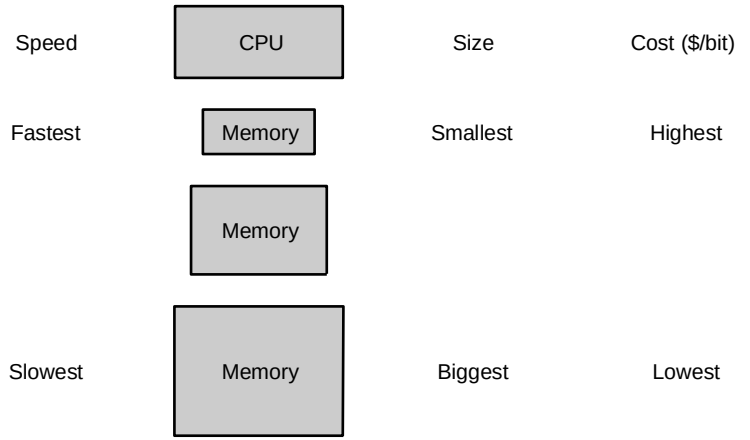
  - Istruzioni e dati all'interno di un ciclo di un programma.

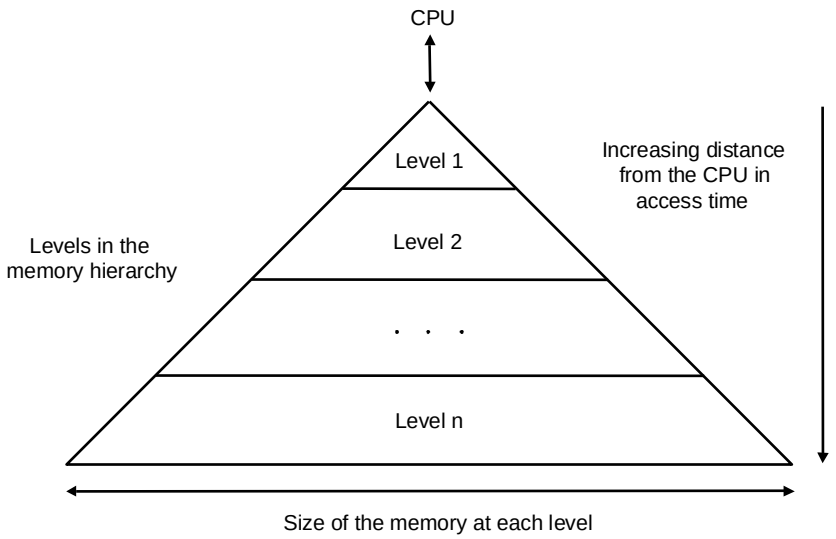
- **Località spaziale**

  Tendenza a far riferimento a dati vicini a quelli a cui si è fatto riferimento di recente.

  - Sequenzialità delle istruzioni e dei dati (vettori, campi di una struttura)

Gerarchia di Memorie

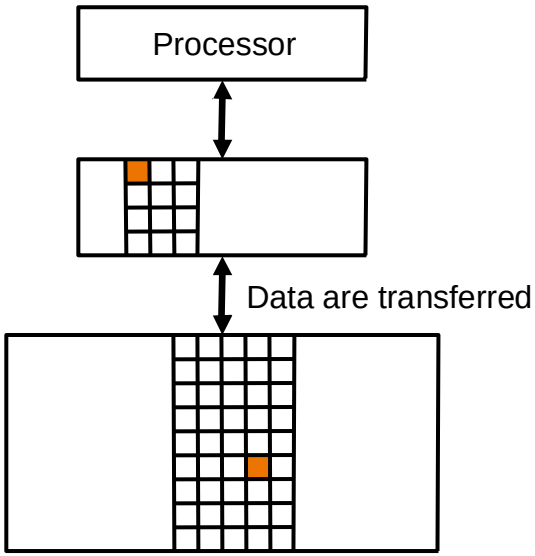




Gerarchia di Memorie

La CPU comunica solo col livello superiore, gli altri trasferimenti dati avvengono solo fra livelli adiacenti.

<u>Blocco</u>: unità minima di informazione che può essere presente o assente in una gerarchia a due livelli.



Gerarchia di Memorie

Prestazioni

Hit – **Successo nell'accesso**: il blocco è presente nel livello superiore.

Miss – **Fallimento nell'accesso**: il blocco non è presente nel livello superiore e deve essere recuperato dal livello inferiore della gerarchia.

Hit rate – **Hit ratio** (frequenza, frazione degli hit): frazione degli accessi alla memoria che si sono risolti al livello superiore.

Miss rate (frequenza di miss = 1 - frequenza di hit): frazione degli accessi alla memoria che non sono stati soddisfatti al livello superiore.

Gerarchia di Memorie

Prestazioni

Tempo di hit: tempo necessario ad accedere al livello superiore della gerarchia delle memorie.

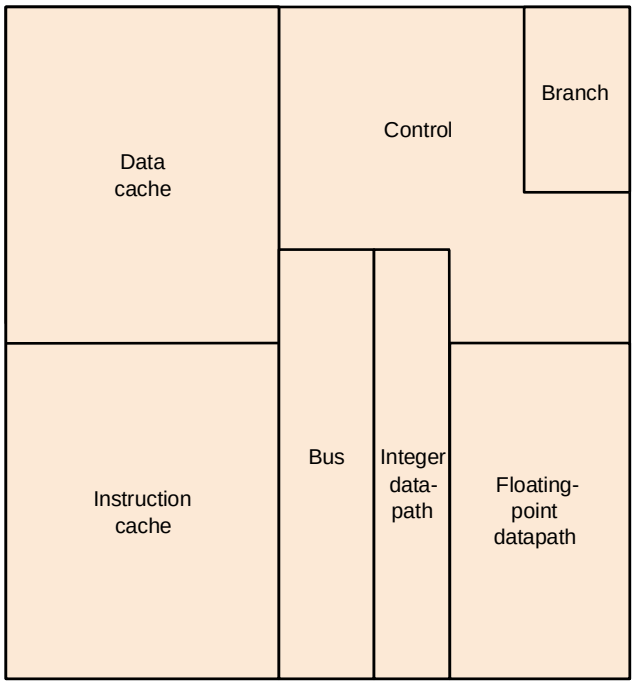
Penalità di miss: tempo necessario per sostituire un blocco nel livello superiore con il blocco corrispondente del livello inferiore, più il tempo necessario per consegnare il dato al processore.

Per migliorare le prestazioni i programmatori / compilatori / sistema operativo devono essere coscienti del funzionamento della gerarchia delle memorie.

Cache

(nascondiglio)

Livello della gerarchia tra la CPU e la memoria principale ed in senso lato gestione della memoria che tragga vantaggio dalla località degli accessi.



Pentium

Cache a corrispondenza diretta

(direct mapped)

Ogni blocco di memoria può risiedere in una sola locazione della cache.

(indirizzo del blocco) = (indirizzo del byte) / (byte per blocco)

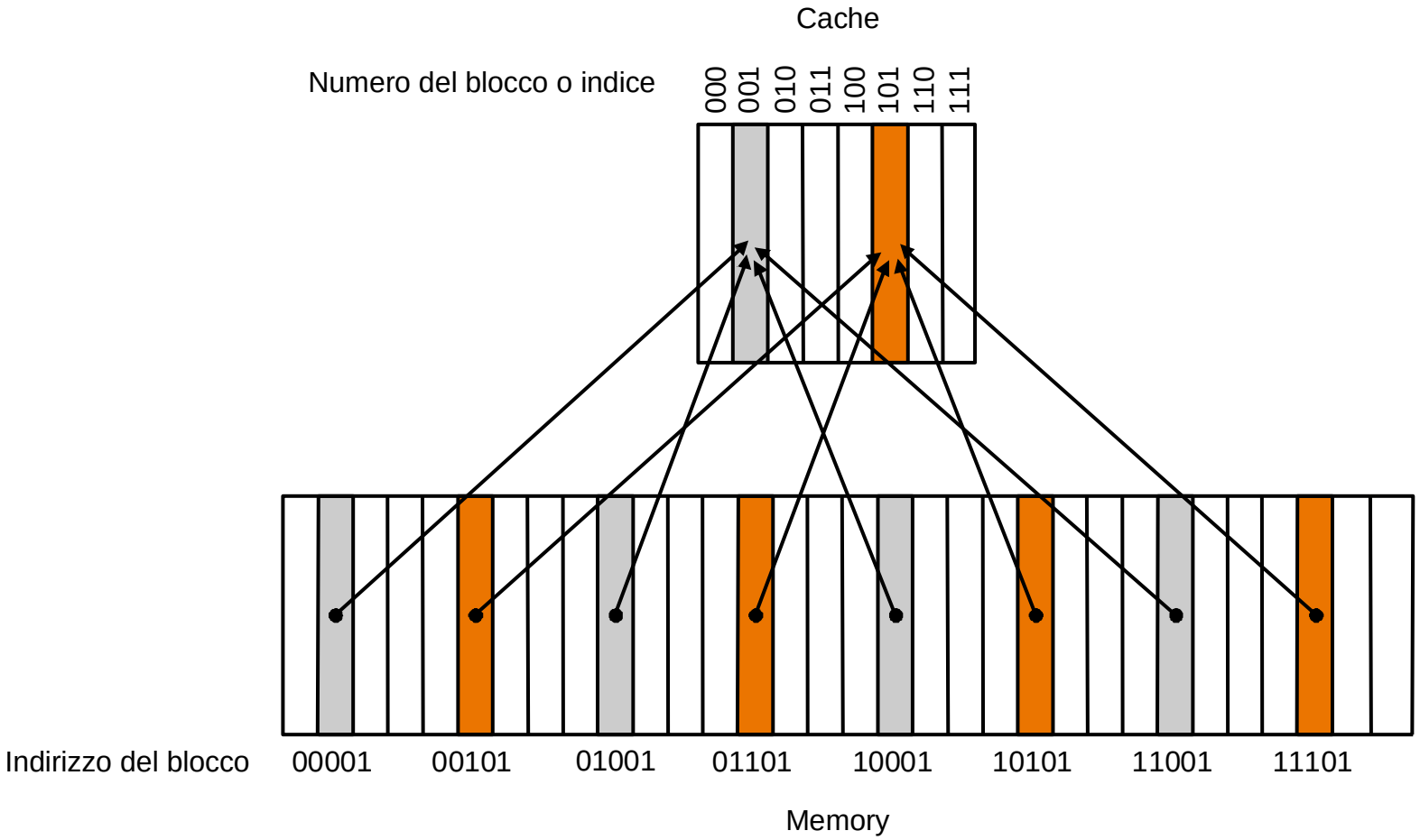
(numero del blocco) = (indirizzo del blocco) modulo (numero di blocchi nella cache)

Se il numero di blocchi nella cache è una potenza di 2, l'operazione di modulo si può calcolare semplicemente utilizzando gli n bit meno significativi dell'indirizzo del blocco, dove

$n = \log_2$(numero di blocchi nella cache)

Cache a corrispondenza diretta

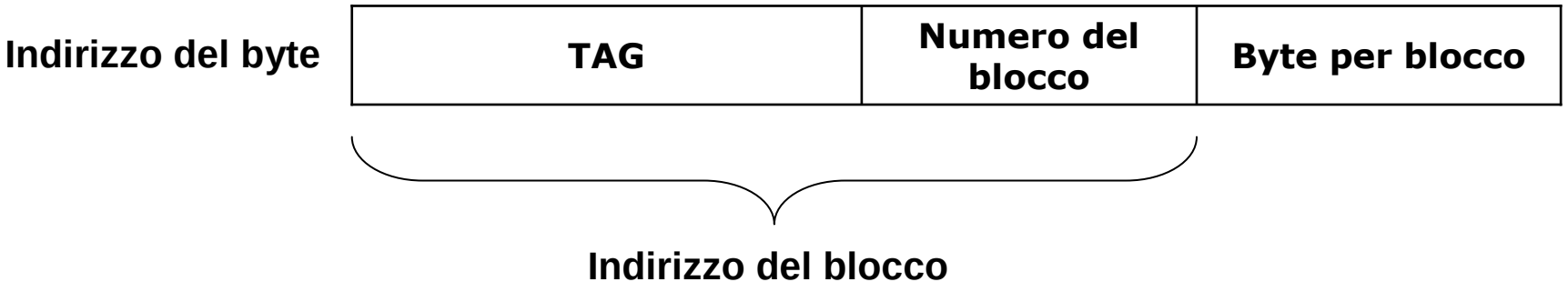
Esempio: cache con 8 blocchi



Cache a corrispondenza diretta

Per ciascuna locazione della cache serve un campo chiamato **tag** che identifichi quale blocco di memoria è contenuto nella cache stessa. Si utilizzano pertanto i bit più significativi dell'indirizzo del blocco.

In generale i bit dell'indirizzo di un byte di memoria possono essere raggruppati per identificare le grandezze introdotte sinora.



Cache a corrispondenza diretta

Oltre al campo tag, serve un bit che indichi la validità del blocco nella cache. Ad esempio la cache inizialmente sarà vuota e tali bit saranno a zero.

La cache contiene dunque sia dati sia porzioni di indirizzi ovvero i campi tag e i bit di validità.

Dimensione

(bit nella cache) = (numero dei blocchi della cache) • ((dimensione del blocco) + (dimensione del tag) + (dimensione del campo di validità))

Esempio: cache con 64KB di dati, blocchi di una parola ed indirizzi di 32 bit:

(bit nella cache) = 2^{14} • (32 + (32 – 2 - 14) + 1) = 784 Kbit = 98 KB

Cache a corrispondenza diretta
Accesso in lettura

Indirizzo del riferimento	Hit o miss nella cache	Blocco della cache corrispondente
10110 (22)	miss	110 (22 mod 8)
11010 (26)	miss	010 (26 mod 8)
10110 (22)	hit	110 (22 mod 8)
11010 (26)	hit	**010 (26 mod 8)**
10000 (16)	miss	000 (16 mod 8)
00011 (3)	miss	011 (3 mod 8)
10000 (16)	hit	000 (16 mod 8)
10010 (18)	miss	**010 (18 mod 8)**

Indice	V	Tag	Dato
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Indice	V	Tag	Dato
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10	Mem(10110)
111	N		

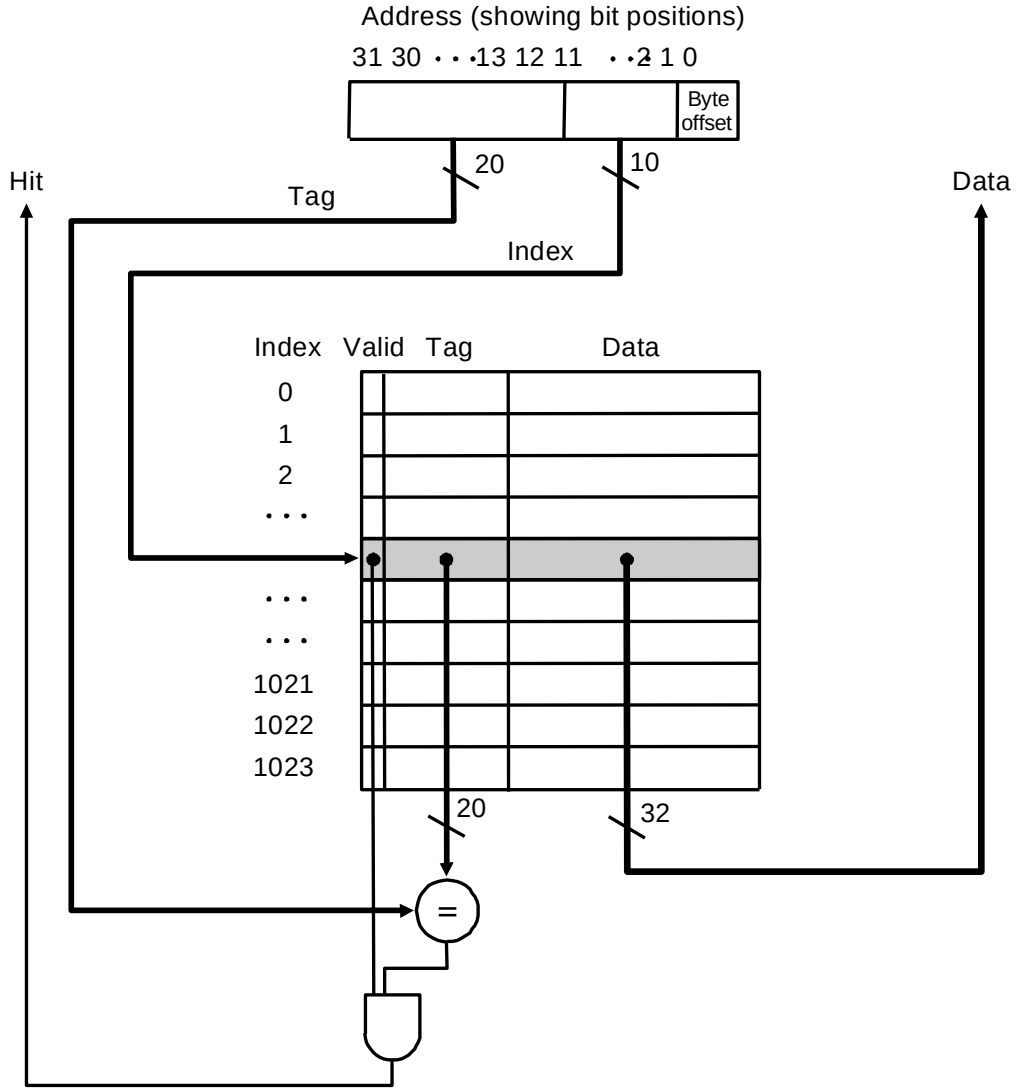
Indice	V	Tag	Dato
000	N		
001	N		
010	S	11	Mem(11010)
011	N		
100	N		
101	N		
110	S	10	Mem(10110)
111	N		

Indice	V	Tag	Dato
000	S	10	Mem(10000)
001	N		
010	S	11	Mem(11010)
011	N		
100	N		
101	N		
110	S	10	Mem(10110)
111	N		

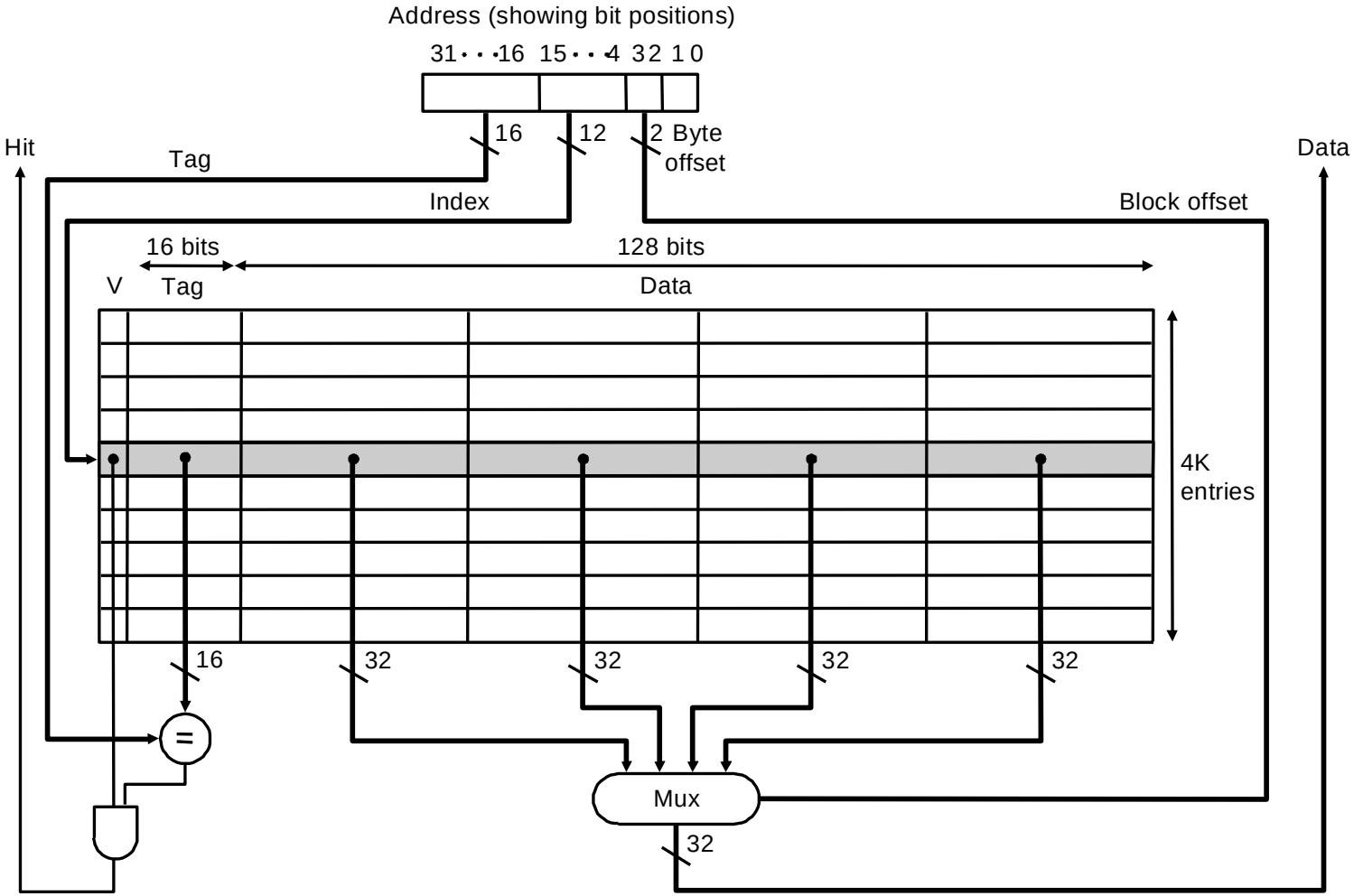
Indice	V	Tag	Dato
000	S	10	Mem(10000)
001	N		
010	S	11	Mem(11010)
011	S	00	Mem(00011)
100	N		
101	N		
110	S	10	Mem(10110)
111	N		

Indice	V	Tag	Dato
000	S	10	Mem(10000)
001	N		
010	S	10	Mem(10010)
011	S	00	Mem(00011)
100	N		
101	N		
110	S	10	Mem(10110)
111	N		

Cache a corrispondenza diretta
Blocco di una parola



Cache a corrispondenza diretta
Blocco di 4 parole



Cache e località

Ogni tipo di cache si avvantaggia naturalmente della **località temporale** visto che contiene i dati a cui si è avuto accesso di recente.

Si avvantaggia invece della **località spaziale** solo se il blocco contiene almeno due parole adiacenti.

L'efficienza di utilizzo della memoria della cache migliora in quanto i tag e i bit di validità occupano in percentuale meno spazio rispetto ai dati.

Località e prestazioni

La **frequenza di miss** si riduce all'aumentare della dimensione dei blocchi e questo in particolare si verifica per la cache delle istruzioni, visto che le istruzioni hanno una maggiore località spaziale.

Se la dimensione del blocco è una frazione significativa della dimensione della cache, significa che i blocchi sono pochi e quindi la frequenza di miss può aumentare.

La **penalità di miss** aumenta dato che aumenta il numero di parole da trasferire dal livello inferiore della gerarchia e quindi il miglioramento della frequenza di miss diviene meno significativo al crescere della dimensione dei blocchi. Il tempo di latenza dovuto alla prima lettura è difficile da limitare, mentre si può nascondere il tempo di trasferimento delle parole successive riprendendo l'esecuzione senza attendere il trasferimento dell'intero blocco (*ripresa immediata – early restart*). Tale tecnica è efficace soprattutto per la cache delle istruzioni.

Gestione dei miss in lettura

Se un blocco non è presente nella cache bisogna mettere in stallo l'intera CPU, mentre un'unità di controllo separata preleva il blocco dal livello inferiore della gerarchia e lo trasferisce nella cache. Se il miss non riguarda istruzioni ma dati, l'esecuzione può continuare finché il dato mancante non deve essere utilizzato (stallo all'utilizzo).

In generale al verificarsi di un miss nella cache delle istruzioni (o dei dati) sono necessari i seguenti passi:

1. Inviare PC – 4 (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.

Si hanno due tecniche per risolvere l'incoerenza:

- **Write-through**: i dati sono scritti sia nel blocco della cache che nel blocco del livello inferiore della gerarchia.
- **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.

Cache
Miss in scrittura

- **Write-through**: se i blocchi della cache contengono una sola parola, il miss non richiede di leggere la memoria, ma genera solo due scritture una nella cache ed una nella memoria. Diversamente bisogna prima trasferire il blocco mancante nella cache e poi procedere con le suddette due scritture.
- **Write-back**: prima di sovrascrivere un blocco nella cache è necessario controllare se è stato modificato (dirty bit) e nel caso copiarlo nel livello inferiore della gerarchia (memoria principale); si procede quindi a trasferire il blocco mancante nella cache e a modificarlo.

Confronto fra le due tecniche

Write-through

- I miss sono più semplici e meno costosi poiché non richiedono mai di scrivere un intero blocco al livello inferiore.
- È più semplice implementare la tecnica write-through rispetto alla write-back, può però servire un buffer delle scritture.

Write-back

- La velocità in scrittura, in caso di hit, è dettata dalla cache anziché dalla memoria.
- Le scritture di più parole nello stesso blocco richiedono una sola operazione di scrittura al livello inferiore della gerarchia.
- Viene sfruttata tutta la larghezza di banda del sistema visto che durante la scrittura nel livello inferiore viene ricopiato un blocco intero.

Confronto fra le due tecniche

Se la latenza delle scritture al livello inferiore della gerarchia è particolarmente elevata è più conveniente la tecnica write-back, perché limita il numero delle scritture.

Per aumentare le prestazioni con la tecnica write-through si utilizza un **buffer delle scritture**, che memorizza i dati in attesa di essere scritti nella memoria.

Il processore entra in stallo solo se il buffer è pieno.

Più la frequenza delle scritture è elevata e tanto più capiente deve essere il buffer.

Progetto di un sistema di memoria con cache

Obiettivo: progettare un sistema di memoria in modo da minimizzare la penalità di miss

La memoria principale è implementata con DRAM caratterizzata da un'elevata densità più che da un tempo di accesso piccolo.

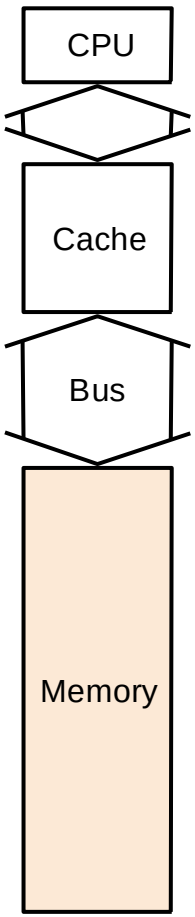
Parametri su cui agire:

- Latenza di accesso
- Throughput

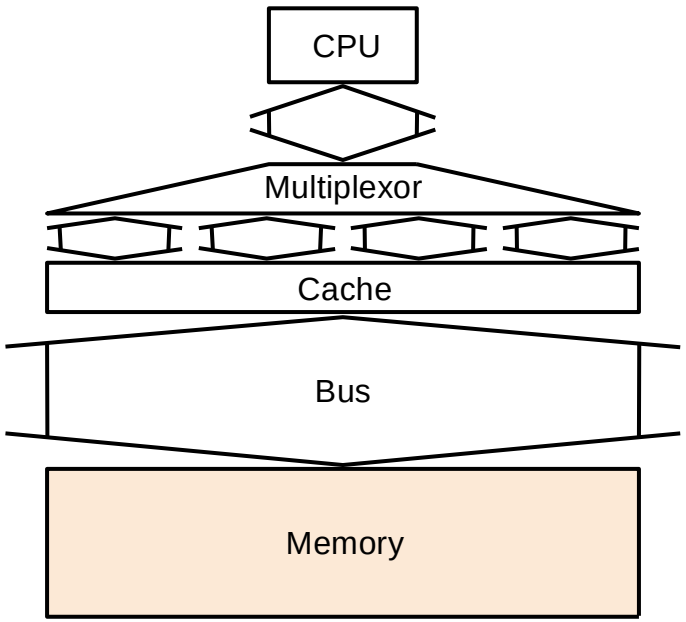
Il bus connette il processore alla memoria e tipicamente è caratterizzato da una frequenza di clock almeno 10 volte inferiore

⇒ Aumentare la banda di trasmissione

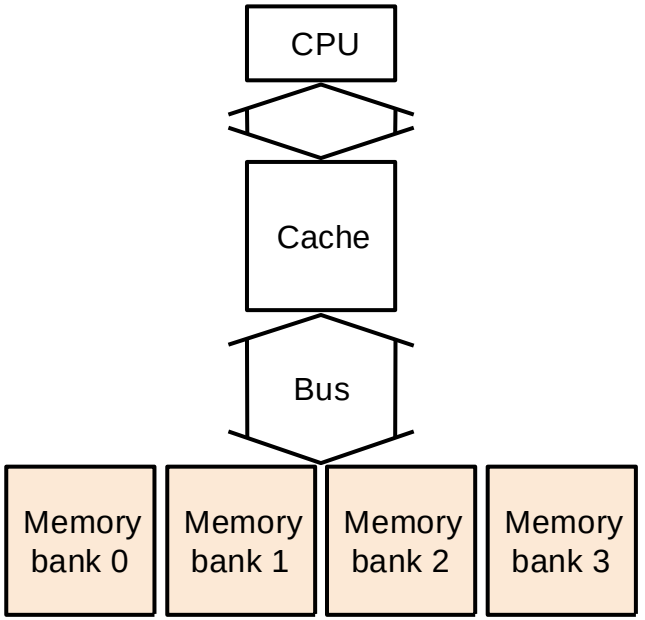
Progetto di un sistema di memoria con cache



a. One-word-wide memory organization



b. Wide memory organization



c. Interleaved memory organization

Progetto di un sistema di memoria con cache

Es.:

- 1 ciclo di clock (bus) per inviare l'indirizzo
- 15 cicli di clock per l'accesso alla DRAM
- 1 ciclo di clock per ogni parola di dato trasferita

<u>Penalità di miss per un blocco di 4 parole</u>

a) 1 + 4 x 15 + 4 x 1 = 65 cicli di clock

ampiezza di banda (byte per ciclo di clock) per un singolo miss = 4 x 4 / 65 = 0,25

b) 1 + 2 x 15 + 2 x 1 = 33 cicli di clock (memoria ampia 2 parole)

ampiezza di banda (byte per ciclo di clock) per un singolo miss = 4 x 4 / 33 = 0,48

1 + 1 x 15 + 1 x 1 = 17 cicli di clock (memoria ampia 4 parole)

ampiezza di banda (byte per ciclo di clock) per un singolo miss = 4 x 4 / 17 = 0,94

c) 1 + 1 x 15 + 4 = 20 cicli di clock

ampiezza di banda (byte per ciclo di clock) per un singolo miss = 4 x 4 / 20 = 0,8

Progetto di un sistema di memoria con cache

Soluzione b

- Maggior costo
- Aumento del tempo di hit

Soluzione c: **Interleaving**

Cache e bus come in a)

Con un solo accesso trasferisco più parole

Consente letture simultanee e scritture indipendenti

L'ampiezza di banda è ottenuta sfruttando le moderne DRAM

- **SDRAM (Synchronous DRAM) – modalità burst**
- **DDR SDRAM (Double Data Rate SDRAM)**

Valutazione delle prestazioni

Si ipotizza innanzitutto che il costo degli accessi alla cache corrispondenti a degli hit faccia parte dei cicli di esecuzione normali della CPU.

tempo di CPU = (cicli di esecuzione della CPU + cicli di stallo per la memoria) • periodo del clock

cicli di stallo per la memoria = cicli di stallo in lettura + cicli di stallo in scrittura

cicli di stallo in lettura = numero di letture per programma • frequenza di miss in lettura • penalità per i miss in lettura

Considerando uno schema write-through si ha:

cicli di stallo in scrittura = numero di scritture per programma • frequenza di miss in scrittura • penalità per i miss in scrittura + stalli sul buffer delle scritture

Valutazione delle prestazioni

Se il buffer è profondo almeno quattro parole e la memoria è in grado di accettare le scritture ad una frequenza almeno doppia rispetto alla frequenza media con cui il programma effettua le scritture, supponiamo di trascurare gli stalli sul buffer delle scritture.

Trascurando quindi gli stalli sul buffer e considerando uguali le penalità per i miss in lettura e per i miss in scrittura, si ha:

cicli di stallo per la memoria = numero di accessi in memoria per programma • frequenza di miss • penalità per i miss

o

cicli di stallo per la memoria = istruzioni per programma • numero di miss per istruzione • penalità per i miss

Valutazione delle prestazioni
Tempo medio di accesso alla memoria

Il tempo medio di accesso alla memoria (average memory access time - AMAT) è una metrica alternativa per valutare i progetti delle cache, che tiene conto sia degli hit che dei miss e della relativa frequenza.

AMAT = tempo necessario per un hit + frequenza di miss • penalità di miss

Sia h la frequenza di hit, C il tempo di accesso per la cache, M la penalità di miss rispetto alla memoria principale. La formula diventa:

$$T_{AMAT} = hC + (1-h)M$$

Esempio: calcolo del tempo medio di accesso in un calcolatore con cache

- Supponiamo che il 30% delle istruzioni in un programma tipico effettui un'operazione di scrittura o di lettura dati
- Supponiamo che la frequenza di successo sia il 95% per le istruzioni e il 90% per i dati
- Supponiamo che siano sempre necessari **17 cicli** di clock per caricare un blocco nella cache dalla memoria principale (sia per le operazioni di lettura che di scrittura)
- Supponiamo che l'accesso alle informazioni nella cache richieda **1 ciclo** di clock
- Si otterrà un **tempo medio di accesso** (riferito a 1 istruzione) pari a

$$t_{ave} = [\underbrace{(0{,}95\times 1+0{,}05\times 17)}_{\text{per le istruzioni}}+\underbrace{0{,}3\times(0{,}9\times 1+0{,}1\times 17)}_{\text{per i dati}}] \times \text{periodo del clock}$$

Esempio: miglioramento delle prestazioni grazie alla presenza della cache

- Con un processore senza cache che impiega 10 cicli di clock per accedere alla memoria sarebbero necessari $(1 + 0{,}3) \times 10$ cicli di clock e quindi il tempo medio di accesso sarebbe

$$t_{senza\ cache} = 1{,}3 \times 10 \times \text{periodo del clock}$$

- Da cui

$$\frac{t_{senza\ cache}}{t_{con\ cache}} = \frac{13 \times \text{periodo di clock}}{[(0{,}95 \times 1 + 0{,}05 \times 17) + 0{,}3 \times (0{,}9 \times 1 + 0{,}1 \times 17)] \times \text{periodo di clock}} =$$

$$= \frac{13}{(0{,}95 \times 1 + 0{,}05 \times 17) + 0{,}3 \times (0{,}9 \times 1 + 0{,}1 \times 17)} = \frac{13}{1{,}8 + 0{,}78} \approx 5$$

Ovvero il calcolatore dotato della cache funziona a una velocità 5 volte superiore a quella del calcolatore senza cache

Esempio: calcolo del tempo di CPU

- Sia **I** il numero delle istruzioni di un programma
- Nell'esempio precedente i cicli di stallo della memoria sono dati da:

accessi ai dati

$$I \times 0{,}05 \times 17 + 0{,}3 \times I \times 0{,}1 \times 17 = (0{,}85 + 0{,}51)I = 1{,}36\ I$$

Cicli di stallo per le istruzioni

Cicli di stallo per i dati

- Supponiamo che il calcolatore abbia un CPI = 2 in assenza di stalli per la memoria e che il periodo di clock sia 2 ns

$$\text{tempo di CPU} = (2 \times I + 1{,}36 \times I) \times 2\ \text{ns} = 6{,}72 \times I\ \text{ns}$$

Esempio: confronto con sistema con cache ideale

- Il valore di CPI per un sistema con cache ideale, cioè in cui non si verificano mai dei miss, è 2 per ipotesi
- Il numero dei cicli di stallo per la memoria (per istruzione) era 1,36
- E quindi, il valore di CPI totale (in presenza di stalli) è dato da:

  2 + 1,36 = 3,36
- Confrontando il sistema con cache ideale e il sistema che presenta stalli si ha:

$$\frac{\text{tempo di CPU con gli stalli}}{\text{tempo di CPU con cache ideale}} = \frac{\text{I} \times \text{CPI}_{\text{con stalli}} \times \text{periodo di clock}}{\text{I} \times \text{CPI}_{\text{ideali}} \times \text{periodo di clock}} = \frac{3{,}36}{2} = 1{,}68$$

- In pratica le prestazioni del sistema con cache ideale sarebbero migliori di un fattore pari a 1,68 rispetto al sistema con la cache con le frequenze di successo ipotizzate.

Esempio: prestazioni della cache al crescere della frequenza di clock

- Supponiamo di voler aumentare le prestazioni del nostro sistema raddoppiando la frequenza di clock
- Ovvero il periodo di clock, che avevamo supposto essere pari a 2 ns, diventa 1 ns
- Supponiamo che la velocità della memoria principale rimanga uguale e che quindi la quantità di tempo dovuta agli stalli rimanga invariata
- Con un clock più veloce, a parità di tempo di accesso alla memoria principale, saranno necessari 34 cicli di clock invece di 17
- Per cui **i cicli di stallo totali** saranno dati da:

$$I \times 0{,}05 \times \mathbf{34} + 0{,}3 \times I \times 0{,}1 \times \mathbf{34} = (1{,}7 + 1{,}02) \times I = 2{,}72 \times I$$

- Ricordando che abbiamo ipotizzato un CPI (senza stalli) pari a 2 si ha:

$$\frac{\text{tempo CPU con clock lento}}{\text{tempo CPU con clock veloce}} = \frac{(2+1{,}36)\times I \times 2\,\text{ns}}{(2+2{,}72)\times I \times 1\,\text{ns}} = \frac{6{,}72}{4{,}72} \approx 1{,}4$$

Esempio: considerazioni

- Nell’esempio precedente, il calcolatore con il clock più veloce è circa 1,4 volte più veloce e non 2 volte, per effetto della maggiore influenza dei fallimenti nella cache (legge di Amdahl)
- Ciò significa che l’importanza delle prestazioni della cache è maggiore per CPU con basso CPI e con elevata frequenza di clock
- Si era inoltre ipotizzato che il tempo di accesso alla cache nelle situazioni di hit non fosse rilevante. In realtà, con cache di dimensioni molto grandi, il tempo di accesso potrebbe compensare o essere dominante rispetto al miglioramento della frequenza di successo
- È importante la tipologia di cache adottata per ottenere una diminuzione della frequenza di miss

Tecniche per migliorare le prestazioni

- Cache set-associative
  - Per ridurre la frequenza di miss
- Cache a più livelli
  - Per ridurre la penalità di miss

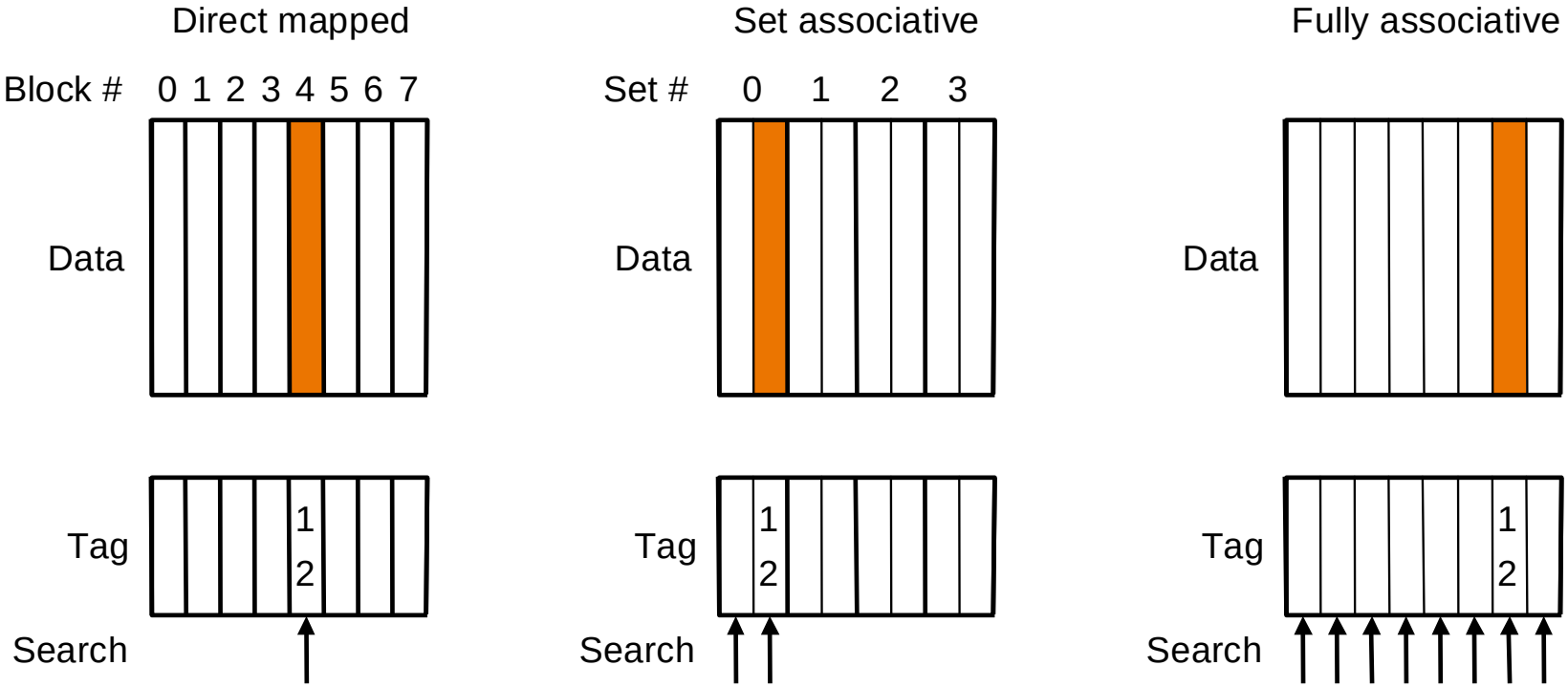
Cache set-associative

L’associazione fra blocco di memoria e locazione occupata nella cache influenza la frequenza di miss visto che più blocchi di memoria competono per un ristretto numero di insiemi (set) della cache in cui essere caricati.

Nelle **cache a corrispondenza diretta** ogni blocco di memoria può essere caricato in un sola locazione (i set in cui è divisa la cache possono contenere solo un blocco – **cache set-associative ad una via**), per ridurre i miss gli insiemi in cui è suddivisa la cache devono contenere almeno due blocchi (**cache set-associative a due vie**).

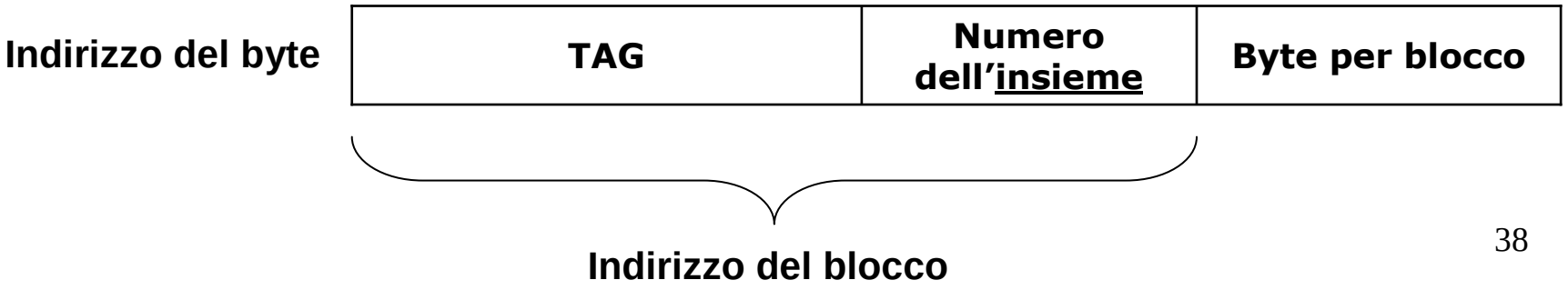
Una **cache set-associative a n vie**, dove n è il numero totale dei blocchi che la cache può contenere, si dice **completamente associativa** dato che ogni blocco può occupare una posizione qualsiasi all’interno della cache.

Cache set-associative



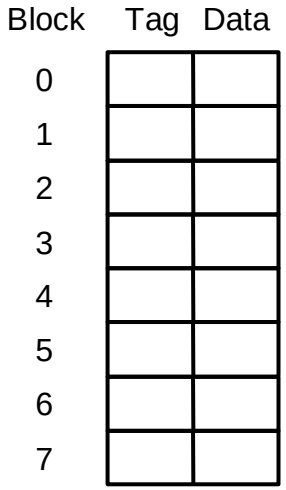
(indirizzo del blocco) = (indirizzo del byte) / (byte per blocco)

(numero dell'insieme) = (indirizzo del blocco) modulo (numero di insiemi nella cache)

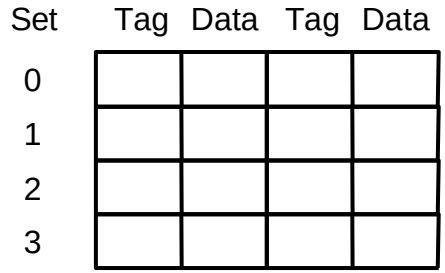


Cache set-associative

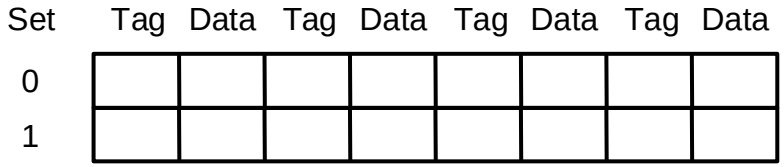
One-way set associative
(direct mapped)



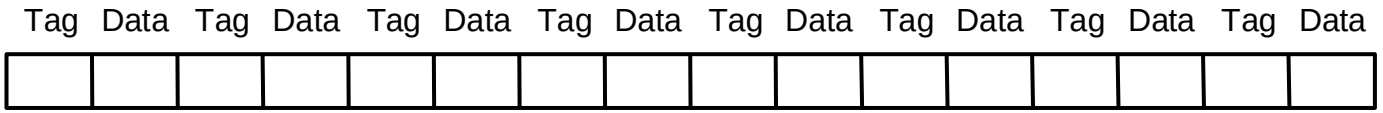
Two-way set associative



Four-way set associative



Eight-way set associative (fully associative)

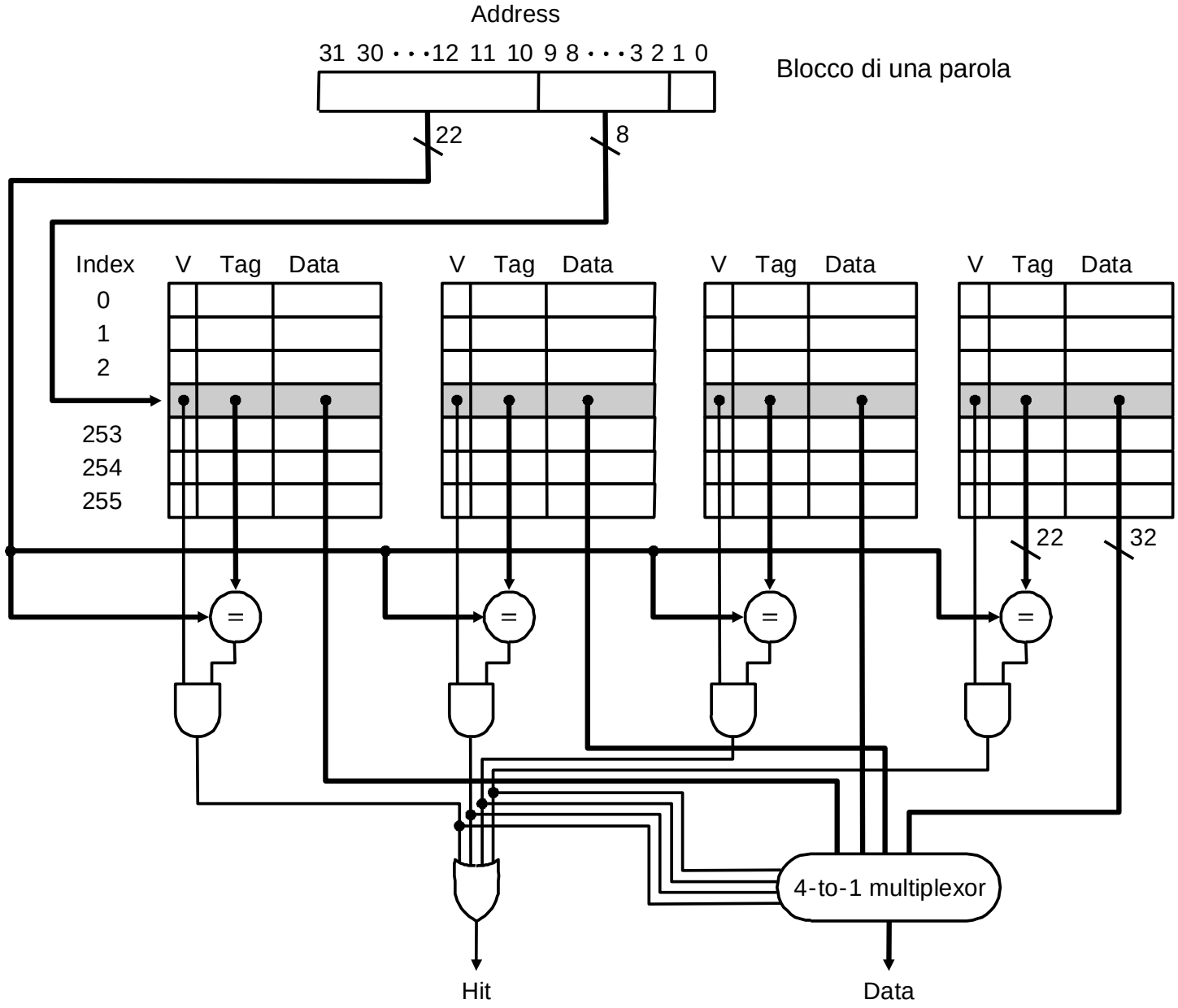


Cache set-associative

La collocazione più flessibile dei blocchi introduce le seguenti problematiche:

- **Ricerca di un blocco all'interno della cache**
  - **Costo implementazione, hit rallentati**
- **Scelta del blocco da sostituire in un set**
  - **Costo implementazione**

Ricerca di un blocco all'interno della cache



Ricerca di un blocco all'interno della cache

Per trovare un blocco all'interno di un set di una cache bisogna esaminare in parallelo, per aumentare le prestazioni, tutti i tag associati ai blocchi contenuti in un determinato insieme. Per questo servono tanti comparatori quanti sono i tag da controllare all'interno di un insieme e dei multiplexer per restituire la parola cercata.

In generale per una cache set-associative a n vie servono n comparatori.

Il costo di una tale implementazione sta quindi nei comparatori aggiuntivi e nel ritardo che le operazioni di confronto comportano. La scelta fra un tipo di cache e l'altra dipenderà pertanto dal confronto tra il costo di un miss rispetto al costo associato all'implementazione dell'associatività.

Dimensione dei campi tag rispetto all'associatività della cache

Aumentare l'associatività richiede più bit per i tag di ciascuno blocco. Sia data una cache di 4K blocchi, blocchi di 4 parole e indirizzi di 32 bit. Trovare il numero totale di insiemi ed il numero totale di bit per i tag nel caso di cache a corrispondenza diretta, set-associative a 2 e a 4 vie e completamente associativa.

4K blocchi significa che la cache ha 2^{12} blocchi, nel caso di **cache a corrispondenza diretta** questo numero coincide con il numero degli insiemi. Blocchi di 4 parole significa dividere la memoria in blocchi di 16 byte, quindi 4 bit dei 32 individuano i byte per blocco. Di conseguenza per il campo tag rimangono **32-12-4=16 bit**, in totale nella cache **16x4K=64Kbit** sono dedicati ai campi tag.

In una **cache set-associative a 2 vie**, ci sono due blocchi per insieme quindi gli insiemi sono 4K/2=2K. Il numero dei tag rimane comunque pari al numero dei blocchi cioè 4K. Per identificare un insieme servono quindi 11 bit e il campo tag ha **32-11-4=17bit** in totale **17x4K=68Kbit** sono dedicati ai campi tag.

Dimensione dei campi tag rispetto all'associatività della cache

In una **cache set-associative a 4 vie**, ci sono 4 blocchi per insieme quindi gli insiemi sono 4K/4=1K. Il numero dei tag rimane comunque pari al numero dei blocchi cioè 4K. Per identificare un insieme servono quindi 10 bit e il campo tag ha **32-10-4=18bit** in totale **18x4K=72Kbit** sono dedicati ai campi tag.

In una **cache completamente associativa** (in questo caso equivale ad una cache set-associative a 4K vie), c'è un solo insieme quindi non serve un campo per identificare gli insiemi. Il numero dei tag rimane comunque pari al numero dei blocchi cioè 4K. Il campo tag ha **32-0-4=28bit** in totale **28x4K=112Kbit** sono dedicati ai campi tag.

	32 bit di indirizzo così ripartiti		
Cache di 4K blocchi	TAG	Numero di insiemi	Byte per blocco
set-associative a 1 via = a corrispondenza diretta	16	12	4
set-associative a 2 vie	17	11	4
set-associative a 4 vie	18	10	4
set-associative a 4K vie = completamente associativa	28	0	4

Attenzione!

Gli indirizzi di 32 bit utilizzati finora indirizzano il byte; nello svolgimento degli esercizi gli indirizzi utilizzati potrebbero invece indirizzare parole o blocchi.

Con indirizzi di byte su 32 bit, l'ultimo campo di bit specifica i byte all'interno di un blocco. Tale campo può essere suddiviso in un offset del byte di 2 bit che specifica il numero del byte all'interno di una parola di 4 byte e in un offset che specifica il numero della parola all'interno di un blocco.

Con indirizzi di parole non si ha l'offset del byte e nel caso suddetto i 30 bit più significativi si possono appunto considerare come l'indirizzo di una parola.

Scelta del blocco da sostituire in un set

Tranne che per le cache a corrispondenza diretta negli altri casi bisogna scegliere il blocco da sovrascrivere all'interno di un set.

Per ridurre la frequenza di miss bisognerebbe sostituire il blocco **usato meno recentemente (least recently used - LRU)**.

La tecnica LRU è costosa e viene implementata solo per gerarchie di memoria con basso grado di associatività (tipicamente da 2 a 4). Per una cache set-associative a 2 vie basta un bit per tenere traccia del blocco usato più di recente, all'aumentare dell'associatività tale tecnica viene approssimata oppure si utilizza la **sostituzione casuale**.

Esempio: miss della cache e collocazione dei blocchi nella cache

Sia data la seguente **sequenza di indirizzi** a cui si intende fare accesso, espressi come **indirizzi di parola**:

1, 4, 8, 5, 33, 66, 32, 56, 9, 11, 4, 43, 88, 6, 32

Sia data una cache con **blocchi di 4 parole** e una dimensione totale di **32 parole**.

Determiniamo se ciascuno degli accessi è un **hit** o un **miss** assumendo che la cache sia inizialmente vuota, nei 3 casi seguenti:

1. Cache a corrispondenza diretta
2. Cache totalmente associativa, ipotizzando una sostituzione LRU
3. Cache set-associativa a 2 vie, ipotizzando una sostituzione LRU

Assumendo indirizzi a 16 bit, determiniamo anche il numero di bit riservati all'etichetta (tag), al blocco (o insieme) e alla parola.

Trasformazione degli indirizzi

- Gli indirizzi a cui dobbiamo fare accesso sono espressi come indirizzi di parola
- Siccome, in ogni caso, dobbiamo caricare nella cache un blocco di 4 parole, trasformiamo gli **indirizzi di parola** in **indirizzi di blocco**
- Usiamo la seguente equazione

$$\text{indirizzo di blocco} = \left\lfloor \frac{\text{indirizzo di parola}}{\text{ampiezza di un blocco}} \right\rfloor = \left\lfloor \frac{\text{indirizzo di parola}}{4} \right\rfloor$$

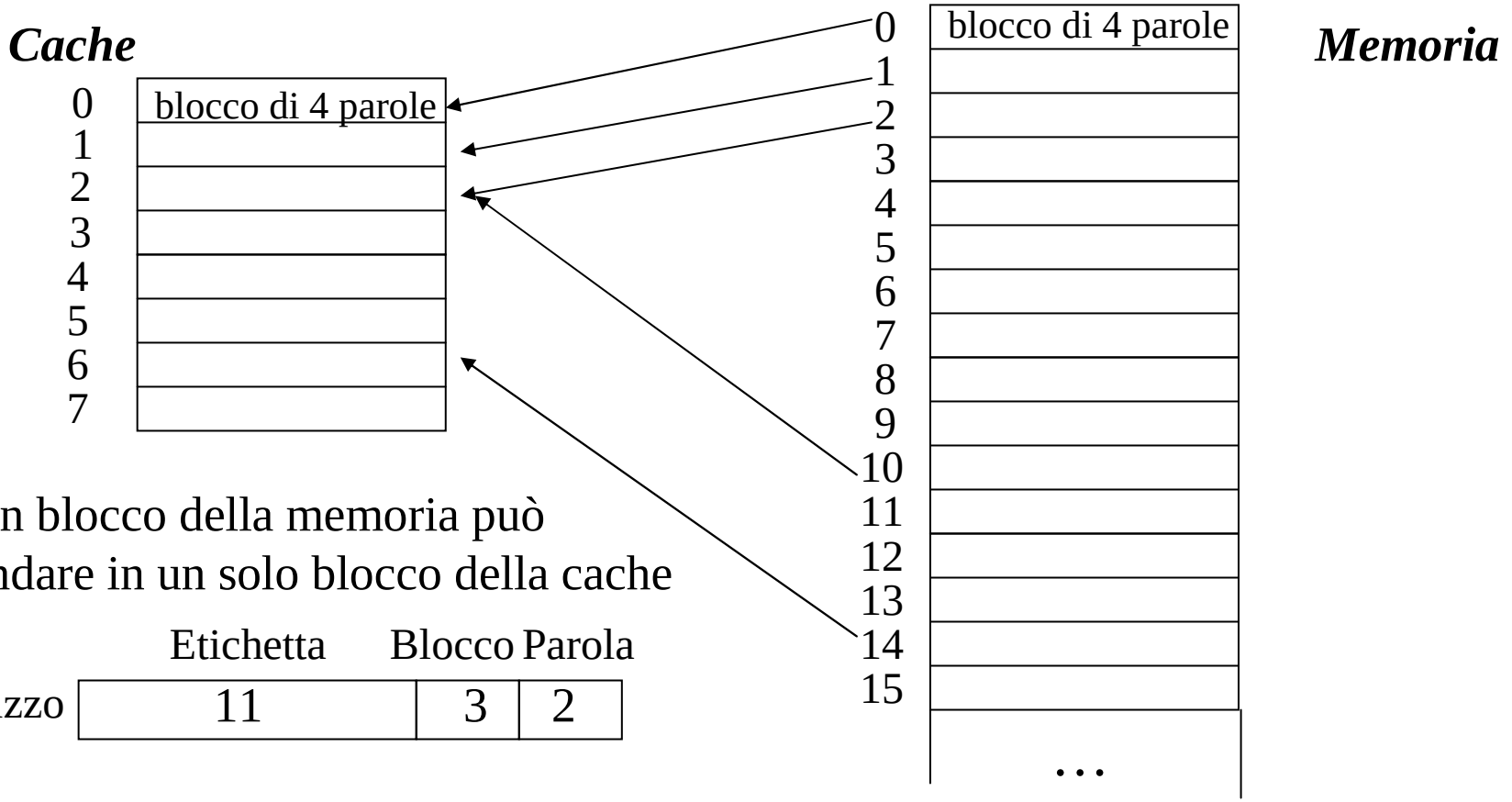
- E quindi troviamo le seguenti corrispondenze

indirizzo di parola	1,	4,	8,	5,	33,	66,	32,	56,	9,	11,	4,	43,	88,	6,	32
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
indirizzo di blocco	0,	1,	2,	1,	8,	16,	8,	14,	2,	2,	1,	10,	22,	1,	8

Caso 1: cache a corrispondenza diretta

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

Blocco della cache 0, 1, 2, 1, 0, 0, 0, 6, 2, 2, 1, 2, 6, 1, 0



Un blocco della memoria può andare in un solo blocco della cache

Caso 1: cache a corrispondenza diretta

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

blocco della cache = (indirizzo di blocco) modulo (numero blocchi in cache)

Blocco	1	2	3	4	5	6	7	8
0	**mem[0]**	mem[0]	mem[0]	mem[0]	**mem[8]**	**mem[16]**	**mem[8]**	mem[8]
1		**mem[1]**	mem[1]	**mem[1]**	mem[1]	mem[1]	mem[1]	mem[1]
2			**mem[2]**	mem[2]	mem[2]	mem[2]	mem[2]	mem[2]
3								
4								
5								
6								**mem[14]**
7								
	miss	miss	miss	hit	miss	miss	miss	miss

Caso 1: cache a corrispondenza diretta

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

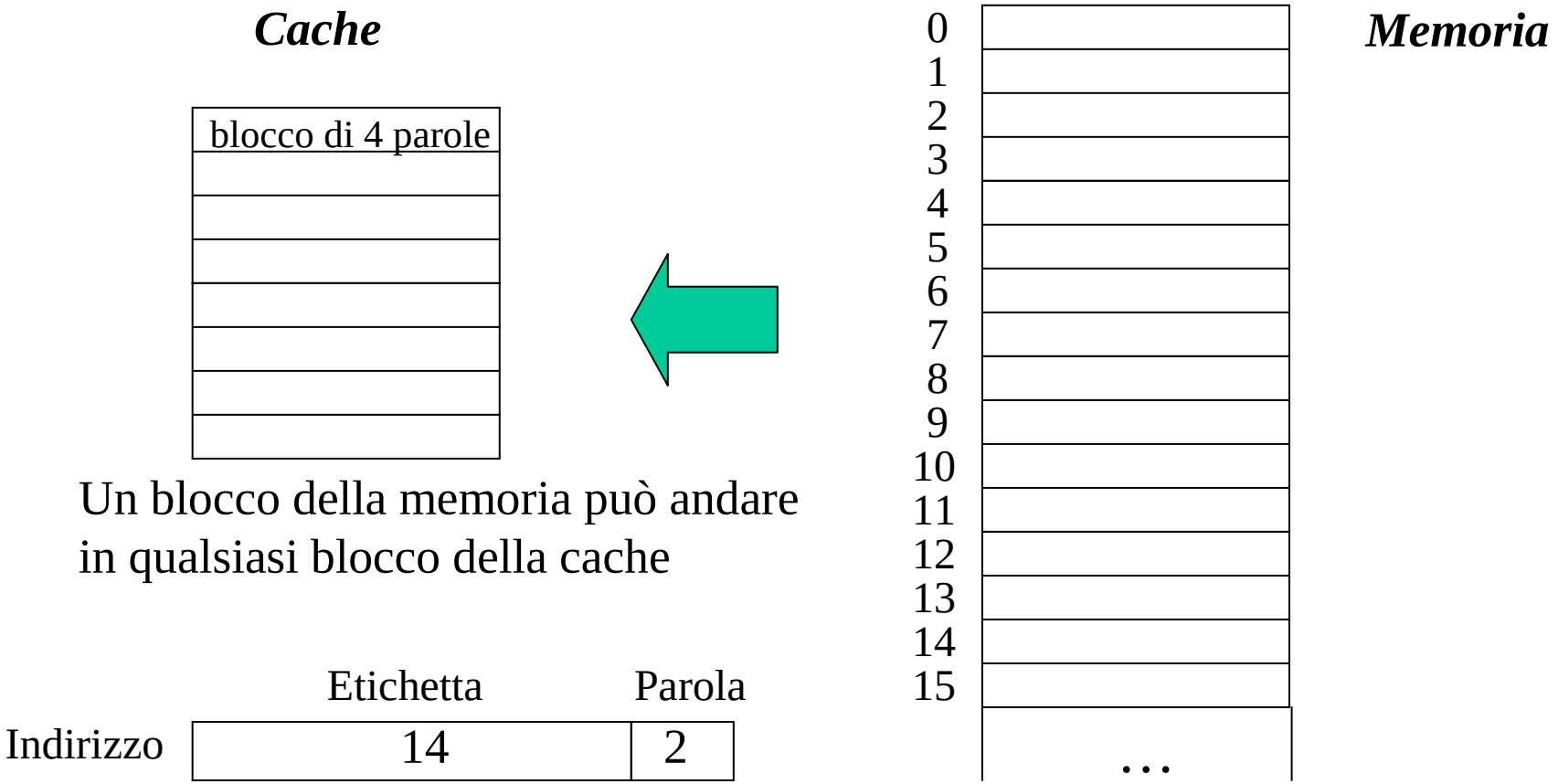
blocco della cache = (indirizzo di blocco) modulo (numero blocchi in cache)

Blocco	9	10	11	12	13	14	15
0	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	**mem[8]**
1	mem[1]	mem[1]	**mem[1]**	mem[1]	mem[1]	**mem[1]**	mem[1]
2	**mem[2]**	**mem[2]**	mem[2]	**mem[10]**	mem[10]	mem[10]	mem[2]
3							
4							
5							
6	mem[14]	mem[14]	mem[14]	mem[14]	**mem[22]**	mem[22]	mem[22]
7							
	hit	hit	hit	miss	miss	hit	hit

Risultato totale degli accessi alla cache = 9 miss e 6 hit

Caso 2: cache completamente associativa

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8



Un blocco della memoria può andare in qualsiasi blocco della cache

Caso 2: cache completamente associativa

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

Blocco	1	2	3	4	5	6	7	8
0	**mem[0]**	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]
1		**mem[1]**	mem[1]	**mem[1]**	mem[1]	mem[1]	mem[1]	mem[1]
2			**mem[2]**	mem[2]	mem[2]	mem[2]	mem[2]	mem[2]
3					**mem[8]**	mem[8]	**mem[8]**	mem[8]
4						**mem[16]**	mem[16]	mem[16]
5								**mem[14]**
6								
7								
	miss	miss	miss	hit	miss	miss	hit	miss

Caso 2: cache completamente associativa

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

Blocco	9	10	11	12	13	14	15
0	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]	mem[0]
1	mem[1]	mem[1]	**mem[1]**	mem[1]	mem[1]	**mem[1]**	mem[1]
2	**mem[2]**	**mem[2]**	mem[2]	mem[2]	mem[2]	mem[2]	mem[2]
3	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	**mem[8]**
4	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]
5	mem[14]	mem[14]	mem[14]	mem[14]	mem[14]	mem[14]	mem[14]
6				**mem[10]**	mem[10]	mem[10]	mem[10]
7					**mem[22]**	mem[22]	mem[22]
	hit	hit	hit	miss	miss	hit	hit

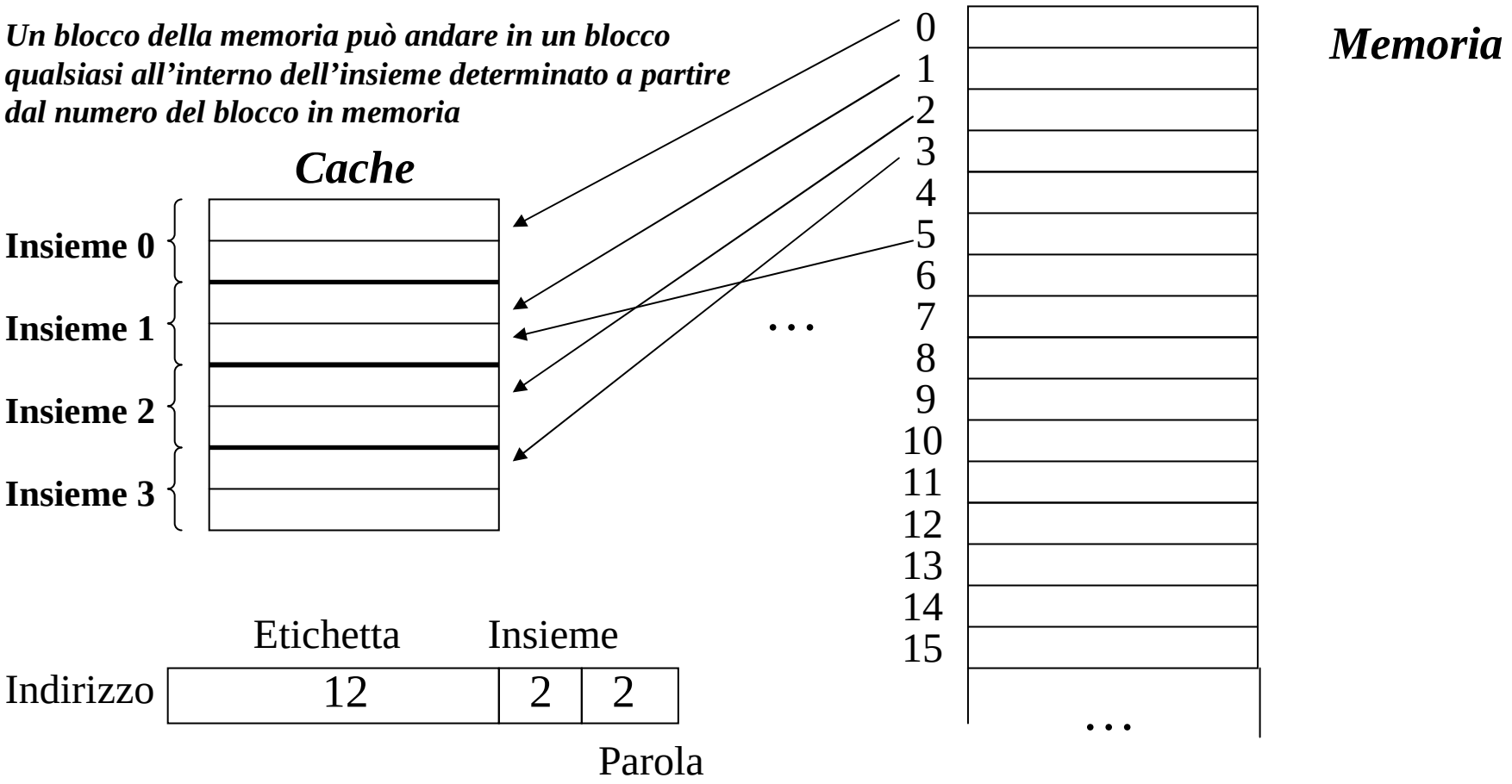
Risultato totale degli accessi alla cache = 8 miss e 7 hit

Caso 3: cache set-associativa a 2 vie

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

Indirizzi di insieme 0, 1, 2, 1, 0, 0, 0, 2, 2, 2, 1, 2, 2, 1, 0

Un blocco della memoria può andare in un blocco qualsiasi all'interno dell'insieme determinato a partire dal numero del blocco in memoria



Caso 3: cache set-associativa a 2 vie

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

insieme della cache = (indirizzo di blocco) modulo (numero insiemi in cache)

Insieme	1	2	3	4	5	6	7	8
0	**mem[0]**	mem[0]	mem[0]	mem[0]	mem[0]	**mem[16]**	mem[16]	mem[16]
					mem[8]	mem[8]	**mem[8]**	mem[8]
1		**mem[1]**	mem[1]	**mem[1]**	mem[1]	mem[1]	mem[1]	mem[1]
2			**mem[2]**	mem[2]	mem[2]	mem[2]	mem[2]	mem[2]
								mem[14]
3								
	miss	miss	miss	hit	miss	miss	hit	miss

Caso 3: cache set-associativa a 2 vie

Indirizzi di blocco 0, 1, 2, 1, 8, 16, 8, 14, 2, 2, 1, 10, 22, 1, 8

insieme della cache = (indirizzo di blocco) modulo (numero insiemi in cache)

Insieme	9	10	11	12	13	14	15
0	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]	mem[16]
	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	mem[8]	**mem[8]**
1	mem[1]	mem[1]	**mem[1]**	mem[1]	mem[1]	**mem[1]**	mem[1]
2	**mem[2]**	**mem[2]**	mem[2]	mem[2]	**mem[22]**	mem[22]	mem[2]
	mem[14]	mem[14]	mem[14]	**mem[10]**	mem[10]	mem[10]	mem[10]
3							
	hit	hit	hit	miss	miss	hit	hit

Risultato totale degli accessi alla cache = 8 miss e 7 hit

Note

- Nonostante il risultato nel caso specifico, normalmente la **cache completamente associativa** presenta le migliori prestazioni in termini di frequenza di miss.
- Il problema delle cache completamente associative è tuttavia **l'aumento del tempo e/o del costo** di accesso agli elementi.
- Infatti, il campo tag (etichetta) dell'indirizzo del blocco richiesto dalla CPU va confrontato con il campo tag degli elementi della cache.
- Nel caso di **cache a corrispondenza diretta** il confronto va fatto con un solo blocco della cache.
- Nel caso di **cache set-associativa a *n* vie** il confronto va fatto con gli *n* blocchi della cache presenti nell'**insieme** individuato dall'indirizzo del blocco richiesto dalla CPU. Questo può essere fatto con *n* comparatori che lavorano in parallelo.
- Nel caso di **cache completamente associativa**, il confronto va fatto con tutti i blocchi della cache… si avrebbe bisogno di molti comparatori.

Un programma in assembler MIPS effettua delle letture in corrispondenza degli indirizzi cha vanno da 0x188EEABC fino a 0x188EEB48 e da 0x200C1DEF fino a 0x200C1E98. Il computer che esegue il codice dispone di una cache per i dati di 256 byte, set-associative a 2 vie e con blocchi di 8 parole.
Supponendo la cache inizialmente vuota:
Specificare per quali letture si ha un miss.
Rappresentare graficamente le variazioni che avvengono nella cache evidenziando i tag dei blocchi caricati. Per l'eventuale sostituzione dei blocchi utilizzare la tecnica LRU.

Nel MIPS parola = 4 byte

Identificare quindi i campi in cui suddividere gli indirizzi

$$\underbrace{\frac{2^8}{2^3 \cdot 2^2}}_{} = 2^3$$ =Numero dei blocchi

$$\frac{2^3}{2} = 2^2$$ =Numero dei set

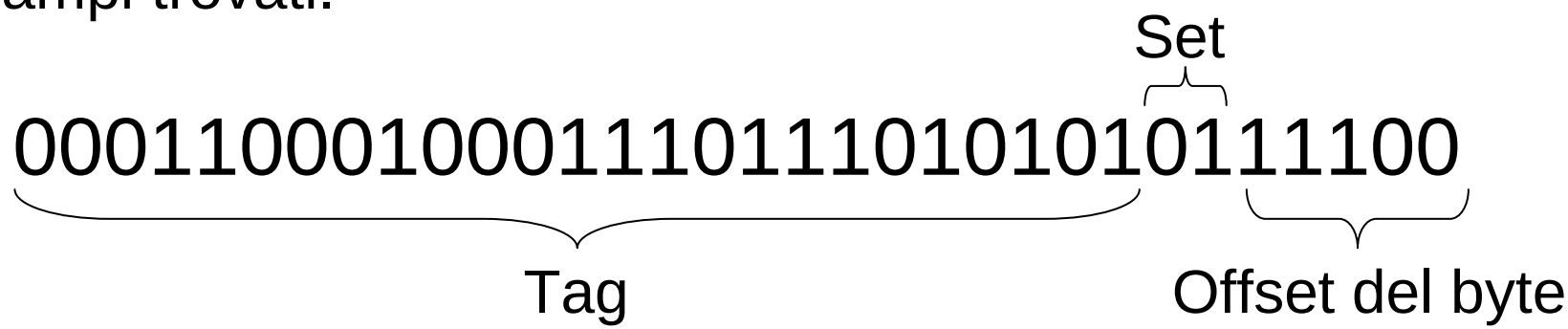
5 bit per individuare il byte all'interno di un blocco

2 bit per individuare il set che potrebbe contenere il blocco

32 – 5 – 2 = 25 bit di tag

I campi in cui scomporre gli indirizzi sono quindi 25 – 2 – 5

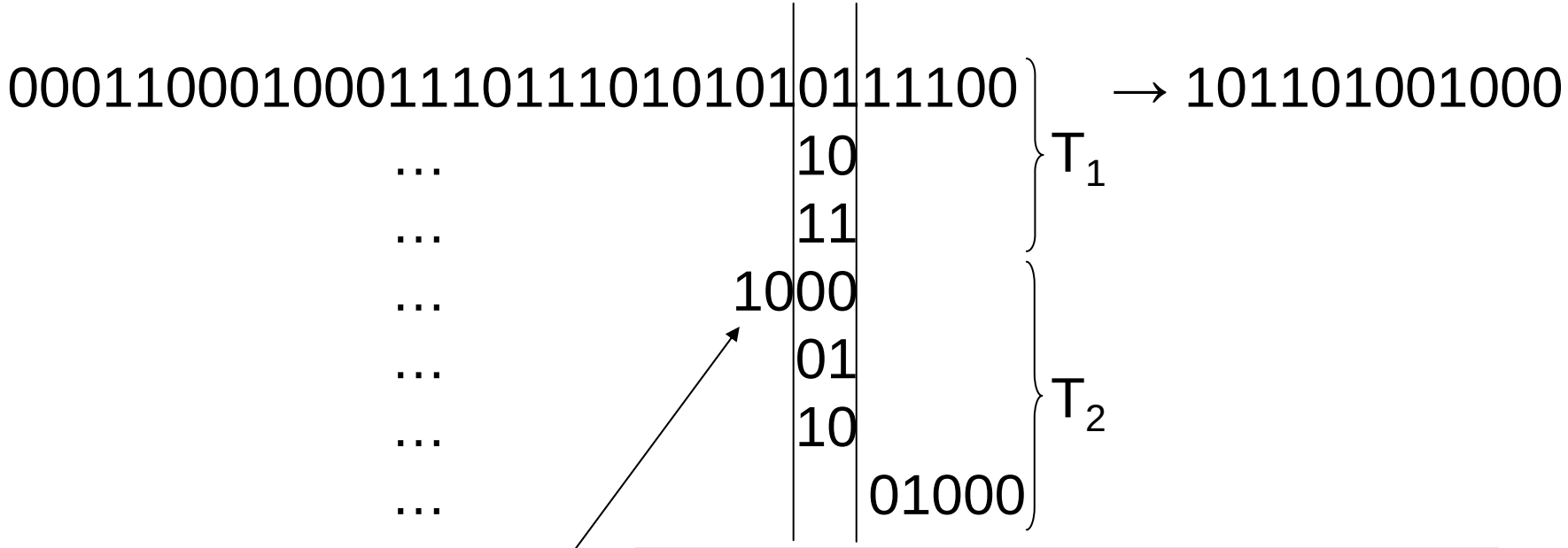
Scrivere in binario l'indirizzo 0x188EEABC ed individuare i campi trovati.



Nota: la lettura all'indirizzo 0x188EEABC consiste nel recuperare il byte 28 (11100) all'interno di un blocco di 32 byte che potrebbe essere contenuto nel set 1 (01) della cache.

La presenza del blocco nella cache è verificata tramite il bit di validità e il tag **0x311DD5** (00011000100011101110101011) .

0x188EE**ABC** → 0x0x188EE**B48**

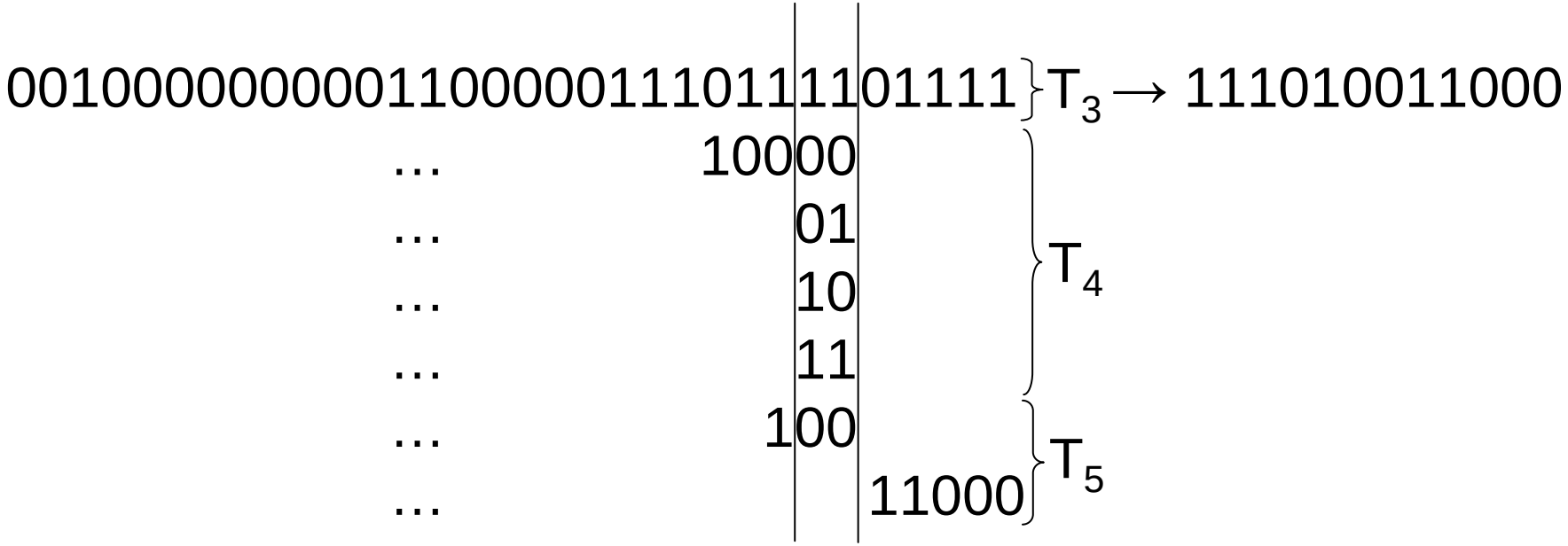


T_1 = 0x311DD**5**

T_2 = 0x311DD**6**

Miss		
0x188EE**ABC**	**T_1**	**set 1**
0x188EE**AC0**	**T_1**	**set 2**
0x188EE**AE0**	**T_1**	**set 3**
0x188EE**B00**	**T_2**	**set 0**
0x188EE**B20**	**T_2**	**set 1**
0x188EE**B40**	**T_2**	**set 2**

0x200C1**DEF** → 0x200C1**E98**

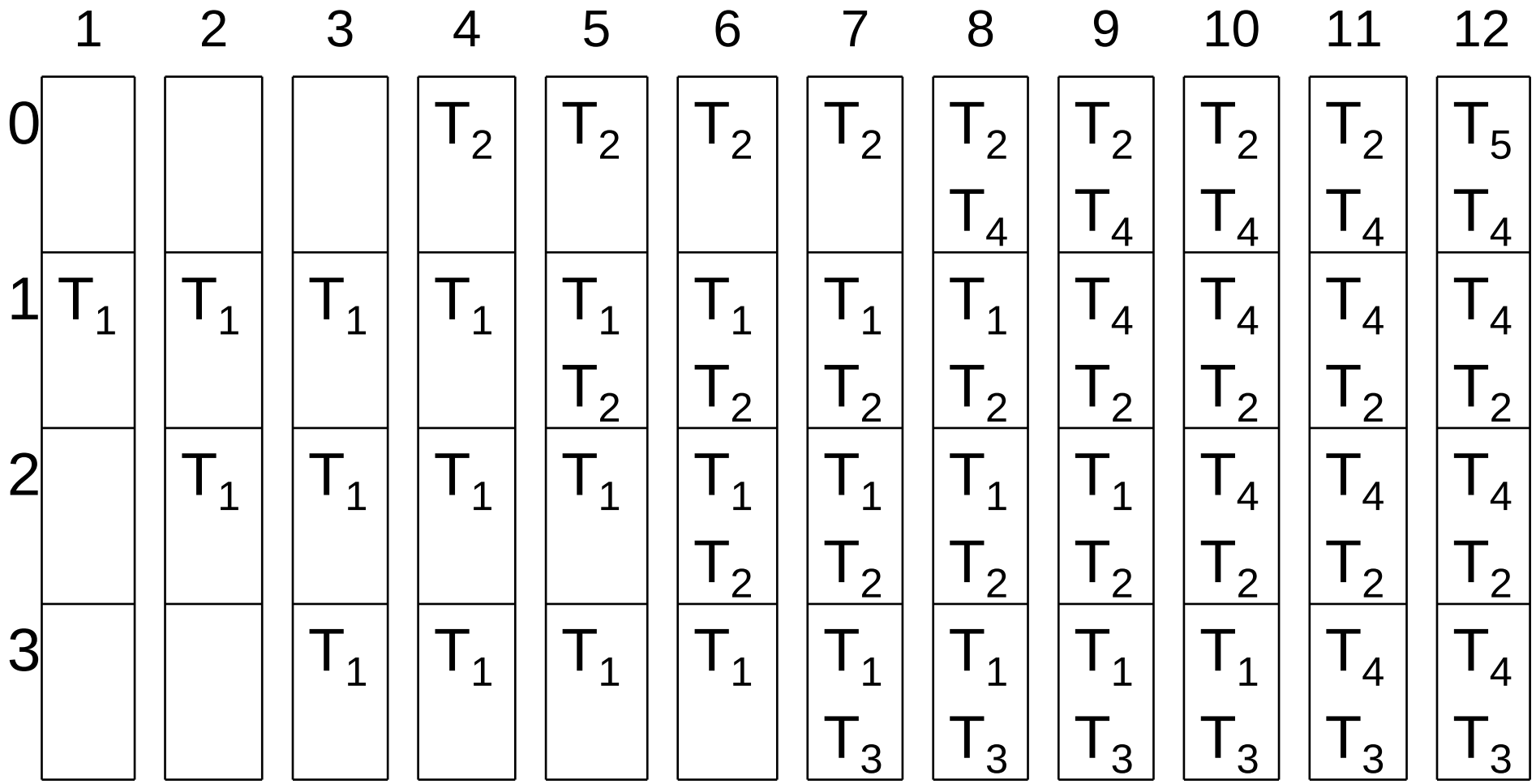


T_3 = 0x40183**B**

T_4 = 0x40183**C**

T_5 = 0x40183**D**

	Miss	
0x200C1**DEF**	**T_3**	**set 3**
0x200C1**E00**	**T_4**	**set 0**
0x200C1**E20**	**T_4**	**set 1**
0x200C1**E40**	**T_4**	**set 2**
0x200C1**E60**	**T_4**	**set 3**
0x200C1**E80**	**T_5**	**set 0**


1 2 3 4 5 6 7 8 9 10 11 12
0
1
2
3
T1
T2
T3
T4
T5

Esempio: confronto fra calcolatori con tipologie di cache diverse

- Calcolatore 1 con periodo di clock pari a 2 ns e la cache seguente:
  - A corrispondenza diretta con blocchi di una parola
  - Frequenza di miss per le istruzioni 4% e frequenza di miss per i dati 8%
- Calcolatore 2 con periodo di clock pari a 2,4 ns e la cache seguente:
  - Set-associativa a due vie con blocchi di 4 parole
  - Frequenza di miss per le istruzioni: 2% e frequenza di miss per i dati: 4%
- In questi calcolatori, metà delle istruzioni contengono un accesso ai dati, la penalità di miss è pari a 6 più la dimensione del blocco in parole, il valore di CPI in assenza di fallimenti è pari a 2.
- Quale calcolatore è più veloce?

Soluzione

- Per il calcolatore 1

  penalità di miss = 6 + 1 = 7 cicli

  cicli di miss = cicli di miss per istruzioni + cicli di miss per i dati =

  $$I \times 0{,}04 \times 7 + I \times 0{,}50 \times 0{,}08 \times 7 = 0{,}56 \times I$$

  tempo CPU$_1$ = $(2 + 0{,}56) \times I \times 2$ ns = $5{,}12 \times I$ ns

- Per il calcolatore 2

  penalità di miss = 6 + 4 = 10 cicli

  cicli di miss = cicli di miss per istruzioni + cicli di miss per i dati =

  $$I \times 0{,}02 \times 10 + I \times 0{,}50 \times 0{,}04 \times 10 = 0{,}40 \times I$$

  tempo CPU$_2$ = $(2 + 0{,}40) \times I \times 2{,}4$ ns = $5{,}76 \times I$ ns

$$\frac{\text{tempo di CPU}_2}{\text{tempo di CPU}_1} = \frac{5{,}76}{5{,}12} = 1{,}125$$

Il calcolatore 1 è 1,125 volte più veloce del calcolatore 2

Cache a più livelli

All’aumentare delle prestazioni dei processori, l’accesso alla memoria principale (DRAM) diventa sempre più un collo di bottiglia significativo, per ridurre la penalità di miss si introduce una cache di secondo livello (SRAM) solitamente all’esterno del chip del processore. Con tale cache la penalità di miss della cache di primo livello è relativa al tempo di accesso alla cache di secondo livello, molto inferiore al tempo di accesso alla memoria principale.

Il tempo medio di accesso alla memoria può dunque essere ricalcolato in base all’introduzione di questo nuovo livello nella gerarchia di memorie.

Cache a più livelli

Tempo medio di accesso alla memoria

$T_{AMAT} = hC + (1-h)M$ con una cache

$T_{AMAT} = h_1C_1 + (1-h_1)T_{AMAT2}$ con 2 cache

$T_{AMAT2} = h_2C_2 + (1-h_2)M$

$T_{AMAT} = h_1C_1 + (1-h_1)(h_2C_2 + (1-h_2)M) =$

$= h_1C_1 + (1-h_1)h_2C_2 + (1-h_1)(1-h_2)M$

Cache a più livelli

Con due livelli di cache, il primo livello può concentrarsi sulla riduzione del tempo associato agli hit, mentre il secondo sulla frequenza di miss.

La cache secondaria è solitamente almeno 10 volte più ampia della principale e il suo tempo di accesso è dell'ordine dei 10 cicli di clock contro gli almeno 40 richiesti per accedere alla memoria principale.

Per l'implementazione di un tale sottosistema di memoria bisogna tenere in considerazione le tecnologie adottate ed è necessario utilizzare dei modelli per simulare il comportamento dei due livelli di cache.

Esempio: prestazioni in presenza di cache secondaria

- Calcolatore dotato di una cache primaria e di una cache secondaria.
- Frequenza di successi uguale per entrambe le cache: 95% per le istruzioni e 90% per i dati.
- Il 30% delle istruzioni richiede un accesso ai dati.
- Sono necessari 1 ciclo di clock per accedere alla cache primaria e 10 cicli di clock per accedere alla cache secondaria.
- Penalità di fallimento per accesso alla memoria pari a 38 cicli di clock.
- Confrontare il tempo medio di accesso sperimentato dalla CPU di questo sistema con il tempo medio di accesso nel caso di un processore con la sola cache primaria.

  Ricordare che vale:

$$\mathbf{t_{ave} = h_1C_1 + (1-h_1)h_2C_2+(1-h_1)(1-h_2)M}$$

  dove h_1 e h_2 sono le rispettive frequenze di successo delle 2 cache, C_1 e C_2 sono i rispettivi tempi di accesso alle cache, M è il tempo per l'accesso alla memoria principale.

Soluzione

- Si deve calcolare il rapporto fra i tempi medi di accesso
- Sia *p* il periodo di clock
- Si ha:
  - $h_1 = h_2 = 0{,}95$ per le istruzioni e 0,90 per i dati
  - $C_1 = 1 \times p \qquad C_2 = 10 \times p \qquad M = 38 \times p$

$$\frac{\text{tempo con 1 cache}}{\text{tempo con 2 cache}} = \frac{h_1 C_1 + (1 - h_1) M}{h_1 C_1 + (1 - h_1) h_2 C_2 + (1 - h_1)(1 - h_2) M} =$$

$$= \frac{\overbrace{[(0{,}95 \times 1 + 0{,}05 \times 38)}^{\text{Per le istruzioni}} + \overbrace{0{,}3 \times (0{,}9 \times 1 + 0{,}1 \times 38)}^{\text{Per i dati}}] \times p}{\underbrace{[(0{,}95 \times 1 + 0{,}05 \times 0{,}95 \times 10 + 0{,}05 \times 0{,}05 \times 38)}_{\text{Per le istruzioni}} + \underbrace{0{,}3 \times (0{,}9 \times 1 + 0{,}1 \times 0{,}9 \times 10 + 0{,}1 \times 0{,}1 \times 38)}_{\text{Per i dati}}] \times p} =$$

$$= \frac{4{,}26}{2{,}174} \approx 1{,}96$$

Il tempo medio di accesso con 2 cache (primaria e secondaria) è circa la metà

La memoria virtuale

Uno spazio d'indirizzamento virtuale permette di raggiungere i seguenti obiettivi:

- Condivisione in maniera efficiente sicura della memoria e della CPU tra molti programmi.
- Eliminazione delle difficoltà di programmazione legate ad una quantità piccola e limitata di memoria principale.

Un programma, compilato in uno spazio di indirizzamento virtuale, ha quindi un proprio spazio di indirizzamento separato da quello degli altri programmi consentendo la condivisione e protezione di cui sopra. Può superare le dimensioni della memoria principale e non è necessario che occupi un blocco contiguo di memoria.

La gestione della memoria da assegnare ai programmi è quindi semplificata (es.: problemi di rilocazione, overlay).

La memoria virtuale

La memoria principale contiene solo le parti attive dei programmi, così come una cache contiene i blocchi attivi del programma in esecuzione.

La memoria principale (*memoria fisica*) e l'hard disk possono dunque essere visti come due livelli di cache rispetto ad una memoria molta grande, ma inesistente ovvero *virtuale*. La terminologia è però differente:

Pagina invece che *blocco di memoria virtuale*.

Page fault (**mancanza di pagina**) invece che *miss*.

La memoria virtuale è implementata sia dall'hardware sia tramite il sistema operativo.

Sfide nel progetto delle gerarchie di memoria

In generale i miss possono essere classificati in base alle seguenti categorie:

- **Miss Certi** (miss obbligatori) o miss di partenza a freddo.
- **Miss per Capacità**
- **Miss per Conflitti**

Ogni modifica che migliora potenzialmente la frequenza di miss può avere degli effetti negativi sulle prestazioni.

Modifica progettuale	Effetto sulla frequenza di miss	Possibile effetto negativo sulle prestazioni
Aumento delle dimensioni	Riduzione dei miss per capacità	Può aumentare il tempo di accesso
Aumento dell'associatività	Riduzione della frequenza di miss dovuta ai miss per conflitto	Può aumentare il tempo di accesso
Aumento della dimensioni dei blocchi	Riduzione della frequenza di miss per maggiore località spaziale	Può aumentare la penalità di miss

Riferimenti

Computer Organization and Design

The Hardware/Software Interface 3rd Edition

David A. Patterson, John L. Hennessy

Capitolo 7

Versione italiana:

Struttura e Progetto dei Calcolatori

L'Interfaccia Hardware-Software

2ª edizione Zanichelli

http://en.wikipedia.org/ o http://it.wikipedia.org/