Principi di architetture dei calcolatori: la memoria cache

Mariagiovanna Sami

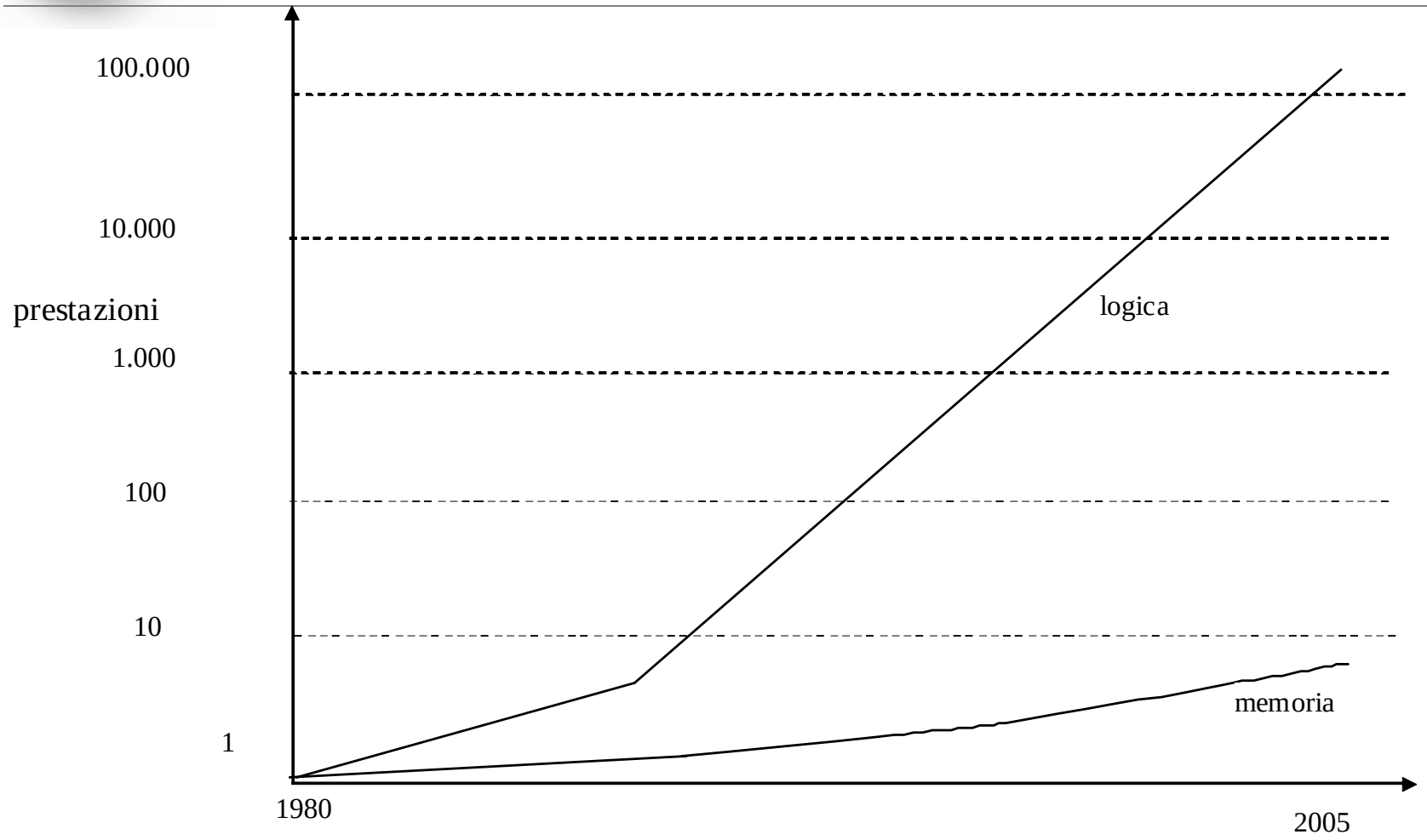
IL PROBLEMA DELLA MEMORIA

- Da sempre, il programmatore desidera una memoria *veloce* e *grande* (illimitata…)
- Il problema: le memorie sono tanto più costose quanto più sono veloci!
- D'altra parte, la differenza fra la velocità della logica (tipicamente, della CPU) e quella della memoria DRAM (la classica *memoria di lavoro,* in genere detta *RAM*) va continuamente aumentando – si tratta del *memory-logic gap.* Le (insoddisfacenti) prestazioni della RAM diventano il collo di bottiglia per le prestazioni di un sistema di elaborazione: la CPU deve "aspettare" che la RAM fornisca informazione senza poter fare nulla durante l'attesa…

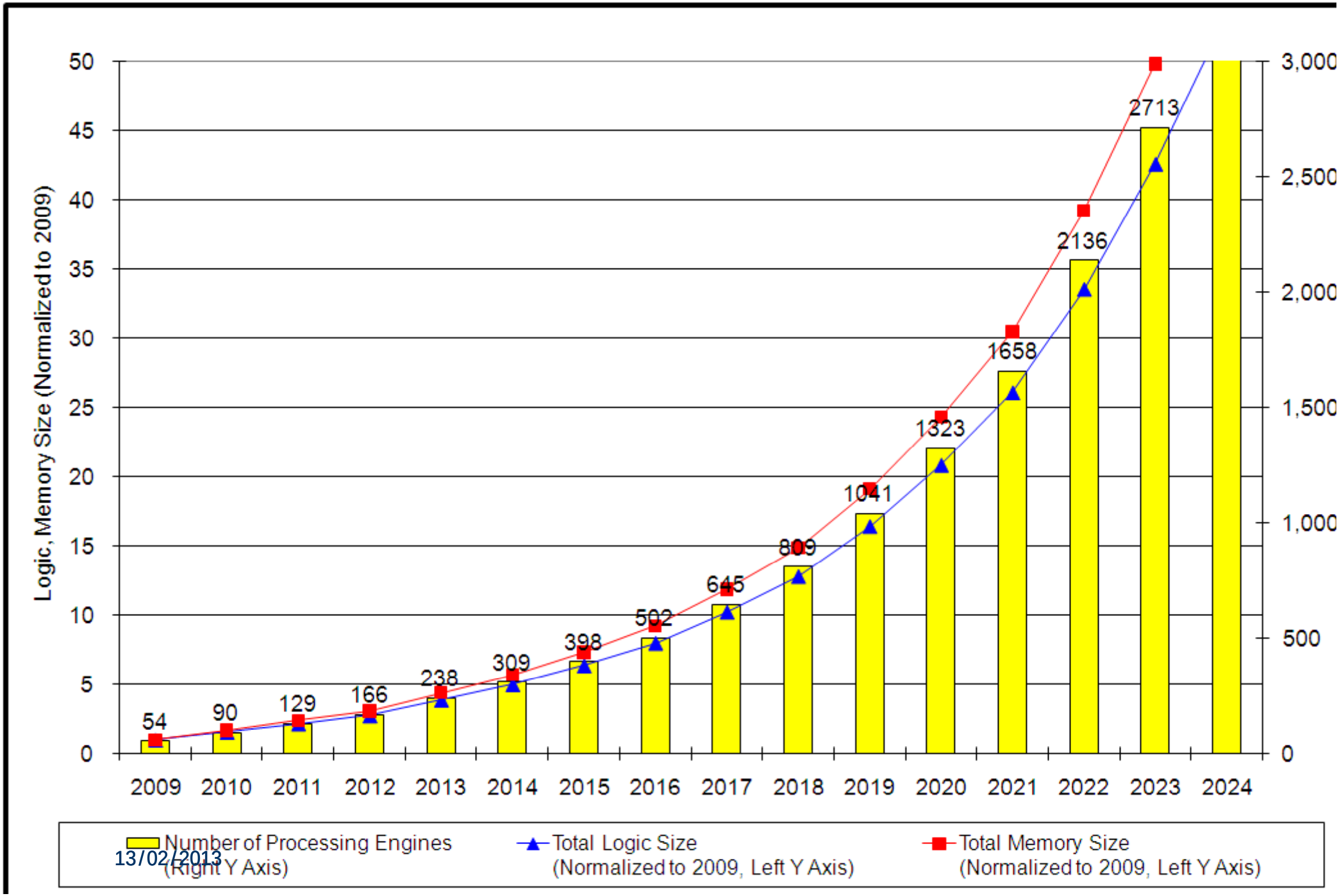
IL PROBLEMA DELLA MEMORIA

- La soluzione? Dare l'*illusione* di una memoria grande e veloce – basandosi sul fatto che in un qualsiasi passo della sua esecuzione un programma non accede a tutte le istruzioni o a tutti i dati con uguale probabilità;
- Le previsioni di crescita in complessità per quanto riguarda la logica e in dimensioni per quanto riguarda la memoria hanno un andamento esponenziale, anche per i dispositivi "mobili" della classe "consumer".

IL GAP MEMORIA-LOGICA



Andamento relativo delle prestazioni di logica e DRAM


Logic, Memory Size (Normalized to 2009)
50
45
40
35
30
25
20
15
10
5
0
3,000
2,500
2,000
1,500
1,000
500
0
54
90
129
166
238
309
398
502
645
809
1041
1323
1658
2136
2713
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Number of Processing Engines (Right Y Axis)
Total Logic Size (Normalized to 2009, Left Y Axis)
Total Memory Size (Normalized to 2009, Left Y Axis)

IL PROBLEMA DELLA MEMORIA

- Indicativamente, si riportano costi e prestazioni di varie tecnologie di memoria al 2010

tecnologia	Tempo tipico di accesso	Costo per GigaByte
SRAM	0.5-5 ns	$4.000-10.000
DRAM	40-50 ns	$100-200
Disco magnetico	3.600.000- 20.000.000 ns	$0,50-2

IL PROBLEMA DELLA MEMORIA

- Per realizzare un'organizzazione della memoria che consenta di accedere *apparentemente* a una memoria al tempo stesso molto grande e molto veloce si ricorre a una *gerarchia di memoria*, che sostituisce alla singola unità di memoria presente nella "classica" architettura di Von Neumann un *insieme di memorie caratterizzate da diverse tecnologie, costi, prestazioni e modalità di accesso*.

IL PROBLEMA DELLA MEMORIA

- In una gerarchia di memoria:
  - Livelli più elevati = più vicini alla CPU, con prestazioni più prossime a quelle prestazioni della logica (⇨ più costosi, di dimensioni più ridotte);
  - Livelli più bassi = più lontani dalla CPU, in tecnologie nettamente diverse da quelle della logica (⇨ meno costosi, capaci di ospitare quantità maggiori di informazione).
- Affronteremo per primo il problema di migliorare la *velocità media di accesso alla memoria da parte della CPU*.

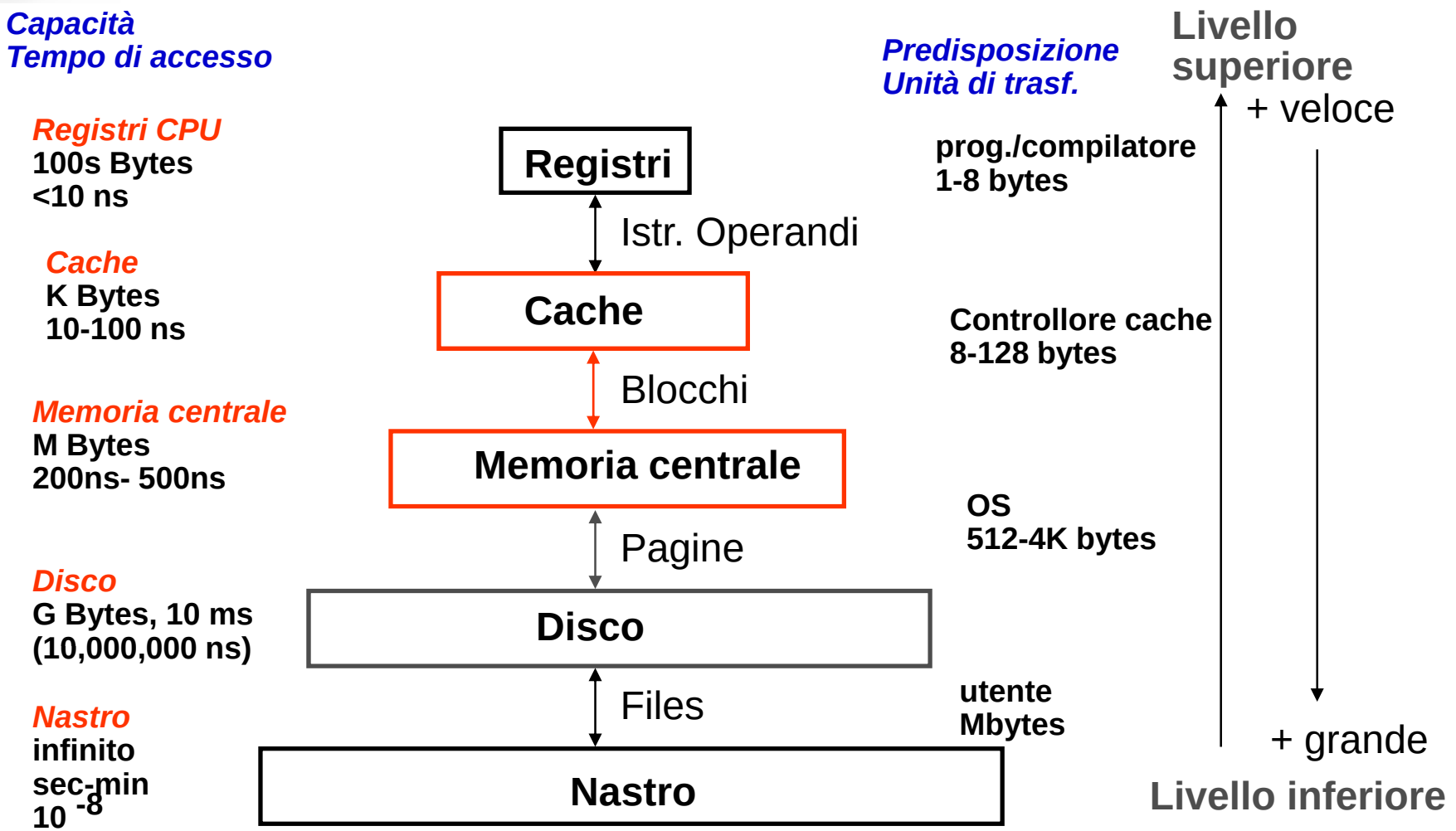
IL PROBLEMA DELLA MEMORIA

- Livello più alto della gerarchia: costituito dai registri interni alla CPU “visibili al programmatore”:
  - Ospitano solamente *dati*
  - I trasferimenti ai/dai registri vengono totalmente gestiti dal programma
  - Il numero dei registri è molto limitato (da 32 a un massimo di 256 in un banco di registri)
  - La tecnologia è la stessa della logica (il banco di registri *è* un componente logico), quindi la velocità è la medesima.

IL PROBLEMA DELLA MEMORIA

- Per ridurre il tempo medio di accesso alla memoria (sia per quanto riguarda l'accesso alle istruzioni, quindi la fase di lettura o *fetch*, sia per quanto riguarda l'accesso ai dati) si introduce fra la CPU e la memoria di lavoro in tecnologia DRAM una memoria relativamente piccola e molto più veloce della DRAM (in tecnologia SRAM), detta *cache.*

IL PROBLEMA DELLA MEMORIA



IL PROBLEMA DELLA MEMORIA

- Di norma, un livello di memoria più prossimo alla CPU contiene un sottoinsieme delle informazioni (sia dati sia istruzioni) contenute nelle memorie di livello inferiore.
- Dato che una memoria di livello elevato è troppo piccola per contenere tutta l'applicazione (con i suoi dati) si deve ammettere che di tanto in tanto parte dell'informazione in essa presente debba essere sostituita con altra che si rende necessaria e si trova in una memoria di livello inferiore;
- Di norma, l'informazione viene trasferita fra coppie di livelli adiacenti; si farà qui sempre riferimento a tale caso, trattando il caso di cache e RAM.

IL PROBLEMA DELLA MEMORIA: LA CACHE

- La cache è *molto* più piccola della RAM ⇨ in genere, troppo piccola per contenere anche uno solo dei programmi che si vogliono eseguire…
- Si deve quindi prevedere di trasferire istruzioni e dati – mano a mano che si rendono necessari – dalla RAM alla cache, riportando eventualmente in RAM i risultati calcolati nel frattempo e memorizzati in cache e le istruzioni che si prevede non "servano" più:
- Ma allora, come può convenire l'introduzione della cache, se comunque sono necessari dei trasferimenti (che ovviamente avvengono alla velocità della RAM)??

IL PROBLEMA DELLA MEMORIA

- Alla base dell'organizzazione della memoria secondo una gerarchia che consenta di accedere *apparentemente* a una memoria al tempo stesso molto grande e molto veloce c'è il *principio di località* (la cui validità è dimostrata in termini statistici su un numero molto elevato di programmi).
- *Località temporale*: se si è fatto riferimento a un elemento in memoria, con alta probabilità si farà nuovamente riferimento entro breve tempo allo stesso elemento;
- *Località spaziale*: se si è fatto riferimento a un elemento in memoria, con alta probabilità entro breve tempo si farà riferimento a elementi "vicini" (= indirizzi prossimi).

LA LOCALITÀ

- Località: nasce da strutture di programma e di dati semplici e naturali: es.:
  - In molti programmi si trovano *cicli* in cui le istruzioni vengono richiamate ripetutamente nelle successive iterazioni del ciclo ⇨ le istruzioni del *corpo del ciclo* hanno elevata località temporale; se il programma a ogni iterazione “aggiorna” valori intermedi dei risultati, questi avranno anch’essi alta località temporale;
  - Le istruzioni vengono eseguite di norma in sequenza ⇨ elevata località spaziale per le istruzioni;
  - Spesso si trovano dati organizzati in vettori ⇨ elevata località spaziale per tali dati

Organizzazione di un sistema dotato di cache

- La memoria centrale e la memoria cache sono organizzate in **blocchi** di parole o di byte, di uguale dimensione; il blocco è *l'unità di trasferimento* fra cache e RAM;
- I blocchi della memoria cache contengono **copie di blocchi** della memoria centrale, oppure sono **blocchi liberi.**
  - ad ogni blocco della cache è associato un bit – detto bit di validità, o **valid bit** - che indica se l'informazione contenuta è significativa (il blocco è valido) oppure se il blocco è *invalido*, quindi libero (cioè non è stato utilizzato in precedenza per trasferirvi informazione oppure contiene informazione che ha perso la sua validità – come si vedrà più avanti).

Organizzazione di un sistema dotato di cache

- Il **sistema di gestione della cache** (attenzione: fa parte *del controllore della cache*, non della CPU!) è in grado di
  - copiare (caricare) blocchi dalla memoria centrale alla memoria cache, oppure di
  - Scaricare (ricopiandoli) blocchi dalla memoria cache alla memoria centrale tramite un'apposita unità funzionale
- Quando il processore vuole accedere a un'informazione in memoria (istruzione o dato) **accede sempre e comunque innanzitutto alla memoria cache**, usando l'indirizzo (definito relativamente *alla RAM*) dell'informazione cercata.

Organizzazione di un sistema dotato di cache

- Il controllore della cache è dotato della capacità di "tradurre" dall'indirizzo in RAM dell'informazione cercata al meccanismo che consente di reperire l'informazione nella cache (o di stabilire che non è presente in cache).

Basi del funzionamento di un sistema dotato di cache

Primo caso: accesso alla memoria per la lettura di una **istruzione**:

- Il processore invia alla cache l'indirizzo dell'istruzione;
- Se il blocco che contiene l'istruzione da prelevare **si trova in cache**, l'istruzione viene letta ed eseguita
- Se l'istruzione **non si trova in cache**:
  - il processore **sospende** l'esecuzione
  - il blocco contenente l'istruzione viene **caricato** dalla memoria centrale in un blocco libero della memoria cache (*)
  - A trasferimento avvenuto, il processore **preleva** l'istruzione dalla cache e **riprende** l'esecuzione.
    - *Se non ci sono blocchi liberi si deve procedere a una sostituzione – che verrà discussa più avanti.*

Basi del funzionamento di un sistema dotato di cache

In modo analogo si vedono le operazioni per accesso a **dati**:

- Il processore deve leggere un dato dalla memoria cache: si procede come visto per la lettura di istruzioni
- Il processore deve scrivere un dato in memoria: si procede in modo simile ma nasce un nuovo problema, e cioè quello della *coerenza* tra memoria cache e memoria centrale (la *vista* del dato che il processore ha "guardandolo" in cache o in RAM deve essere la stessa).

Cache: definizioni

Se la CPU trova in cache l'informazione cui vuole accedere (identificata dal suo *indirizzo in memoria*) si dice che l'accesso si conclude con uno **Hit** (**successo**). Si definiscono

- **Hit Rate** (**tasso di successo**): percentuale degli accessi a memoria che trovano il dato in cache (valutata come il rapporto fra il numero di accessi che hanno successo e il numero totale di accessi);
- **Hit Time** (**tempo di successo**): tempo necessario per accedere al dato se questo si trova in cache (= t*empo di accesso a cache* + *tempo per determinare successo o fallimento del tentativo).*

Cache: definizioni

- Se la CPU *non* trova in cache l'informazione cui vuole accedere, si verifica un **Miss** (**fallimento**): l'accesso alla cache non ha fornito l'informazione cercata, che deve quindi essere recuperata nella memoria centrale e trasferita nella cache. Si definiscono:
  - *Miss Rate* (**tasso di fallimento**) = 1 - (Hit Rate)
  - *Miss penalty*: tempo necessario per sostituire un blocco in cache con un altro blocco dalla memoria di livello inferiore (si usa di norma un valore medio – se il blocco da sostituire è un blocco di dati, la sostituzione può coinvolgere prima la scrittura da cache a RAM e poi la lettura da RAM a cache);

Cache: definizioni

- Si definisce infine
  - *Miss time*: = *miss penalty* + *hit time*, tempo necessario perché la CPU ottenga l'elemento richiesto in caso di miss.
- Se tutti i blocchi in cache sono occupati, in caso di miss sono necessarie due operazioni successive:
  1. Identificazione del blocco di cache in cui verrà portata l'informazione voluta, eventuale riscrittura da cache a RAM dell'informazione ivi finora contenuta
  2. Caricamento da RAM a cache del blocco voluto.

Cache: definizioni

- Prestazioni: motivo principale per introdurre la gerarchia di memoria ⇨ è importante calcolare il tempo necessario per servire hit e miss!
- Hit time: molto minore del tempo di accesso alla memoria di livello inferiore (il tempo di accesso alla memoria inferiore è la componente principale della miss penalty).
- Si vorrebbe che il tempo *medio* di accesso all'informazione da parte della CPU fosse *molto prossimo allo hit time* – dato che hit time e miss penalty dipendono in buona misura dalla tecnologia, per avvicinarsi a tale risultato diventa determinante ridurre al massimo il miss rate.

Strutture e funzionamento della cache

- La CPU indirizza l'informazione – istruzione o dato – mediante il suo *indirizzo fisico in memoria primaria*: come si traducono tali indirizzi nella cache?
  - Metodo di **indirizzamento:** come scegliere il blocco della cache in cui copiare un blocco di memoria centrale *(*problema del *placement* o *mapping)*
  - Metodo di **identificazione:** come localizzare un indirizzo di memoria all'interno della cache
  - Metodo di **sostituzione:** come scegliere i blocchi della cache il cui contenuto va sostituito per liberare spazio
  - Metodo di **scrittura:** come comportarsi quando si deve scrivere una parola contenuta in un blocco della cache
- L'ultimo problema è in buona misura "ortogonale" ai primi tre, e verrà trattato separatamente.

Strutture e funzionamento della cache

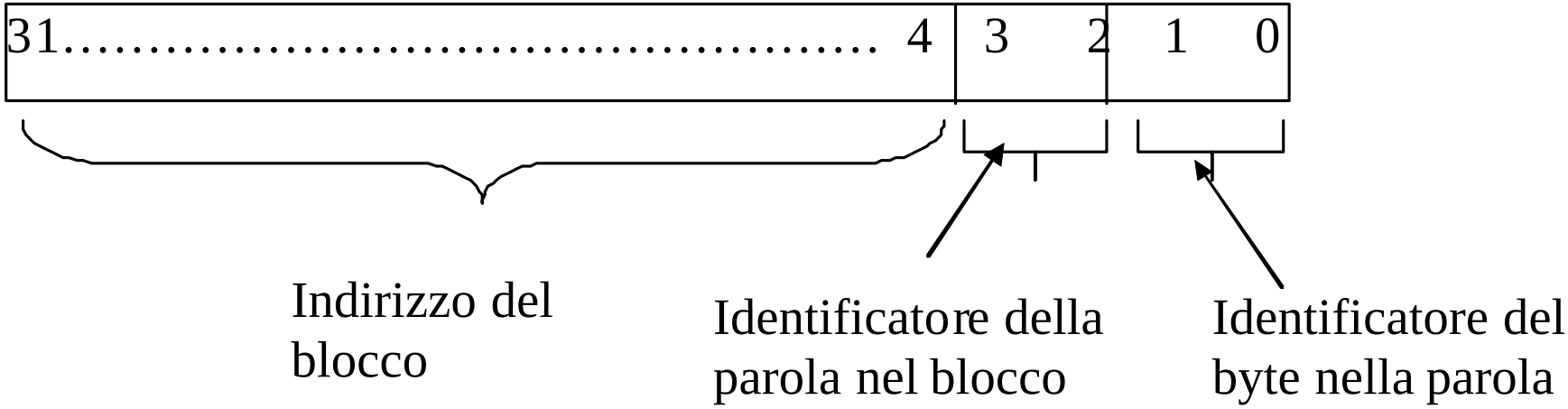
- *Per supportare la località spaziale*, il blocco (o "linea") della cache ha necessariamente dimensione superiore a una sola parola – tipica dimensione di un blocco: quattro o otto parole. (Blocchi di una sola parola non sfrutterebbero la località spaziale!);
- Anche la RAM deve essere pensata organizzata in blocchi della stessa lunghezza – i trasferimenti vengono effettuati per blocchi interi.
- L'indirizzo di una parola a questo punto deve essere scomposto in modo da identificare il blocco e la parola all'interno del blocco.

Strutture e funzionamento della cache

- Es.: blocco di quattro parole. La parola di memoria e l'indirizzo sono di trentadue bit (quattro byte):
  1. I due bit meno significativi (0 e 1) dell'indirizzo identificano il byte all'interno della parola
  2. I due bit immediatamente più significativi (2 e 3) identificano la parola all'interno del blocco;
  3. I rimanenti ventotto bit identificano il blocco.

Strutture e funzionamento della cache

- Struttura logica di un indirizzo di memoria:



Strutture e funzionamento della cache

- Architettura di cache più semplice: *cache direct-mapped* ("cache a indirizzamento diretto") - *ogni blocco della RAM può essere trasferito ("mappato") esattamente in una e una sola posizione della cache* (dato che la RAM è molto più grande della cache, ne segue che *molti diversi blocchi della RAM* potranno trasferirsi *sullo stesso blocco della cache*);
- Modo più semplice per generare la corrispondenza: basato sull'indirizzo dell'informazione in memoria.

Cache Direct Mapped

- I blocchi della cache vengono indirizzati in modo “diretto” come in qualsiasi memoria (il numero d’ordine del blocco in cache, detto “indice”, viene inviato alla porta di indirizzamento della cache). Sia N_b il numero di blocchi della cache: $\log_2 N_b$ è il numero di bit necessari per indirizzare un blocco in cache (quindi è la lunghezza dell’indice);
- Data una generica configurazione C_b dell’indice , sia B il blocco della cache indirizzato da C_b; ogni blocco di RAM il cui indirizzo ha i $\log_2 N_b$ bit meno significativi con la stessa configurazione C_b *può trasferirsi solamente in B* (ovviamente si fa riferimento ai bit che indirizzano il blocco – “depurati” da quelli che indirizzano la parola nel blocco e il byte nella parola).

Cache Direct Mapped

Da quanto detto segue che:

- Ogni blocco della memoria centrale è caricabile in **un solo** blocco della cache, identificato univocamente dalla configurazione dei $\log_2 N_b$ bit meno significativi dell'indirizzo di blocco;
- Lo *stesso* blocco di cache è la destinazione in cui si possono trasferire *tutti i blocchi della memoria centrale che condividono la stessa configurazione dei $\log_2 N_b$ bit meno significativi dell'indirizzo* (è possibile che nascano *conflitti* se un blocco che si vuole caricare deve sostituirne uno caricato da poco).

Cache Direct Mapped

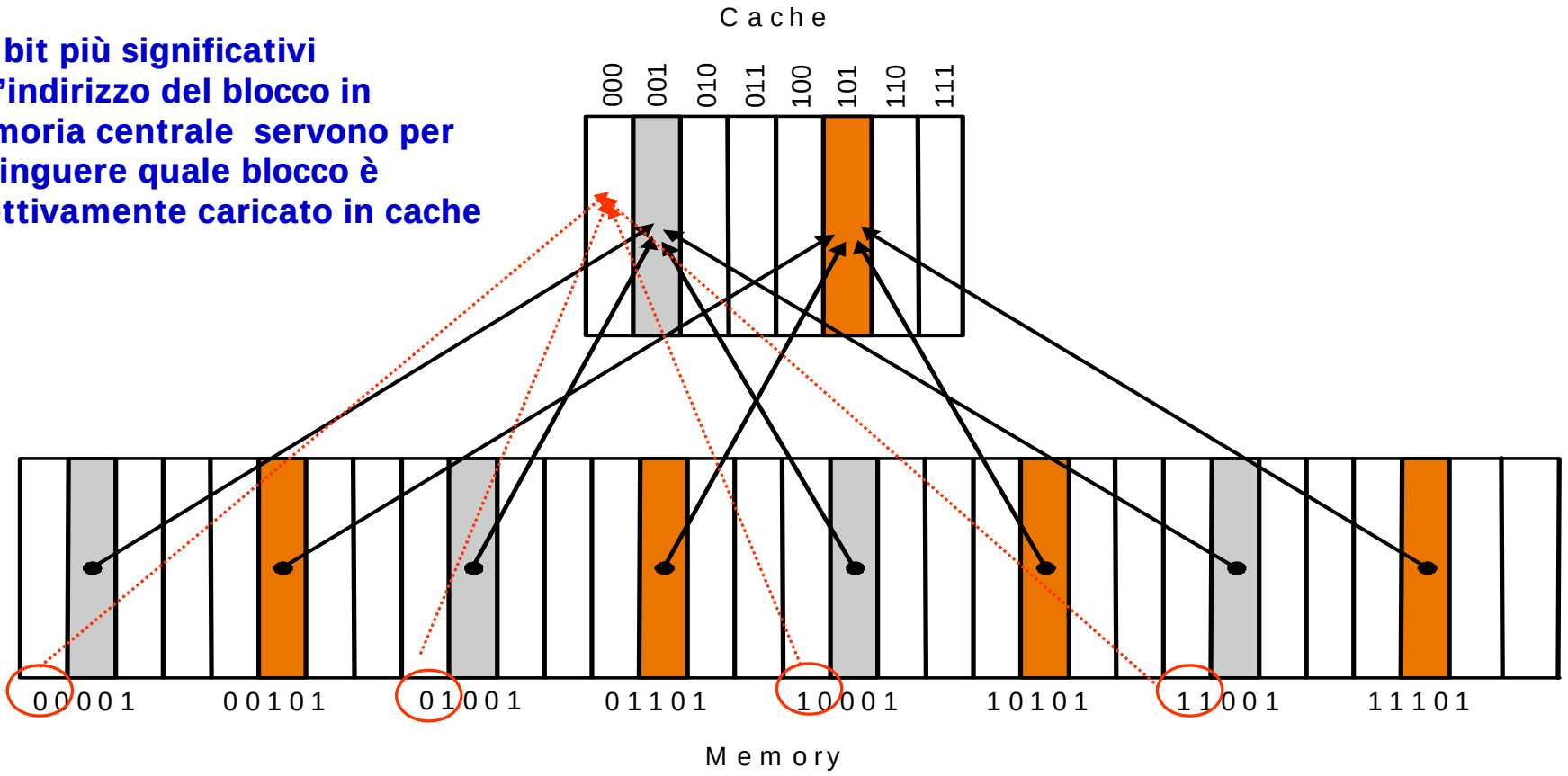
- Il metodo di indirizzamento – **mapping** – è banale: il blocco della memoria centrale con indirizzo *j* è caricabile solo nel blocco di cache il cui *indice* (numero d'ordine) si calcola come

  resto (div intera di j/n° blocchi della cache)

- Es.: RAM di 32 blocchi, cache composta di otto blocchi – l'indice del blocco in cache è di tre bit;
- 2^5 blocchi memoria centrale: l'indirizzo (indice) del blocco in RAM è di 5 bit;
- 2^2 blocchi di memoria centrale si mappano su un identico blocco di cache (= $2^5 / 2^3$)

Cache Direct Mapped

I 2 bit più significativi dell'indirizzo del blocco in memoria centrale servono per distinguere quale blocco è effettivamente caricato in cache



Cache Direct Mapped

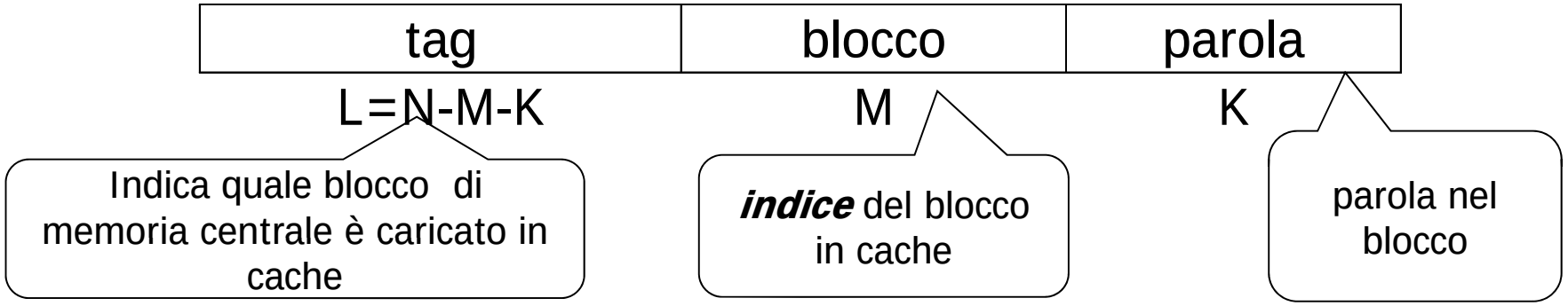
- Accedendo a un blocco in cache, come distinguere *se il blocco di RAM* che vi si trova è effettivamente quello cercato oppure un altro di quelli "aventi diritto"?
- ⇨ nella cache a ogni blocco si aggiunge un *tag* (etichetta) in cui si scriveranno i *bit più significativi dell'indirizzo del blocco in RAM*: se N_i sono i bit di indirizzamento previsti nell'architettura per il blocco, il tag sarà costituito da $N_i - \log_2 N_b$ bit. Quindi:
  - I $\log_2 N_b$ bit meno significativi dell'indirizzo – che costituiscono l'*indice* della cache – vengono utilizzati direttamente per estrarre il blocco;
  - I restanti bit (più significativi)dell'indirizzo corrispondono al tag e vengono registrati nello spazio corrispondente associato al blocco in cache.

Struttura degli indirizzi ed etichette

- Memoria centrale di 2^N parole *indirizzata alla parola*: N bit di indirizzo
- Blocchi di 2^K parole: K bit di indirizzo per identificare una parola nel blocco
- Cache di 2^M blocchi: M bit di indirizzo per identificare un blocco nella cache
- La memoria centrale consta di $2^{(N-K)}$ *blocchi* ($2^N/2^K$)
- $2^{L=(N-M-K)}$ blocchi di memoria centrale si mappano su uno stesso blocco di cache: si usano gli L bit più significativi dell'indirizzo per identificare il blocco effettivamente caricato in cache (**tag** o etichetta)

Struttura degli indirizzi ed etichette

N bit di indirizzo in memoria centrale



Esempio

- Memoria centrale da 4Gbyte (indirizzo di 32 bit), cache a indirizzamento diretto da 1KByte, parole da 32 bit (4 byte), blocchi da 32 byte (8 parole)
- *Accesso a memoria a byte*:
  - blocchi da 32 byte: 5 bit di indirizzo per identificare il singolo byte (scomponibili in 2 per indirizzare il byte nella parola e 3 per indirizzare la parola nel blocco);
  - cache di 32 blocchi (1Kbyte/32byte): 5 bit di *indice* per identificare il blocco in cache;
  - 22 bit di etichetta = 32 – 5 - 5

Identificazione

- Come detto, occorre memorizzare in cache insieme al blocco anche l'**etichetta** ricavata dall'indirizzo di memoria centrale che lo riguarda
- A questo punto, ogni "posizione" della cache include:
  - **Bit di validità (Valid bit)** che indica se questa posizione contiene o meno dati validi. All'accensione, tutte le posizioni della cache sono NON valide (bit di validità = 0);
  - **Campo etichetta** dove si registra il valore che permette di identificare univocamente l'indirizzo di memoria corrispondente ai dati memorizzati;
  - **Campo dati** che contiene una copia dei dati (il contenuto vero e proprio del blocco di memoria).

Identificazione

- Dato un **indirizzo di memoria centrale**, **l'accesso a cache** avviene nel modo seguente:
  - la parte di indirizzo che contiene l'indice del blocco in cache viene usata per selezionare il blocco; la parte di indirizzo che specifica il byte (o la parola) nel blocco viene utilizzata per accedere al byte (o alla parola) voluto
  - i bit più significativi dell'indirizzo (corrispondenti all'etichetta) vengono confrontati con l'etichetta del blocco selezionato, e contemporaneamente si verifica se il blocco è valido (bit di validità =1),
  - se il confronto è positivo e il blocco è valido si deduce che il dato è in cache e si ha *hit* – l'informazione viene trasferita alla CPU.

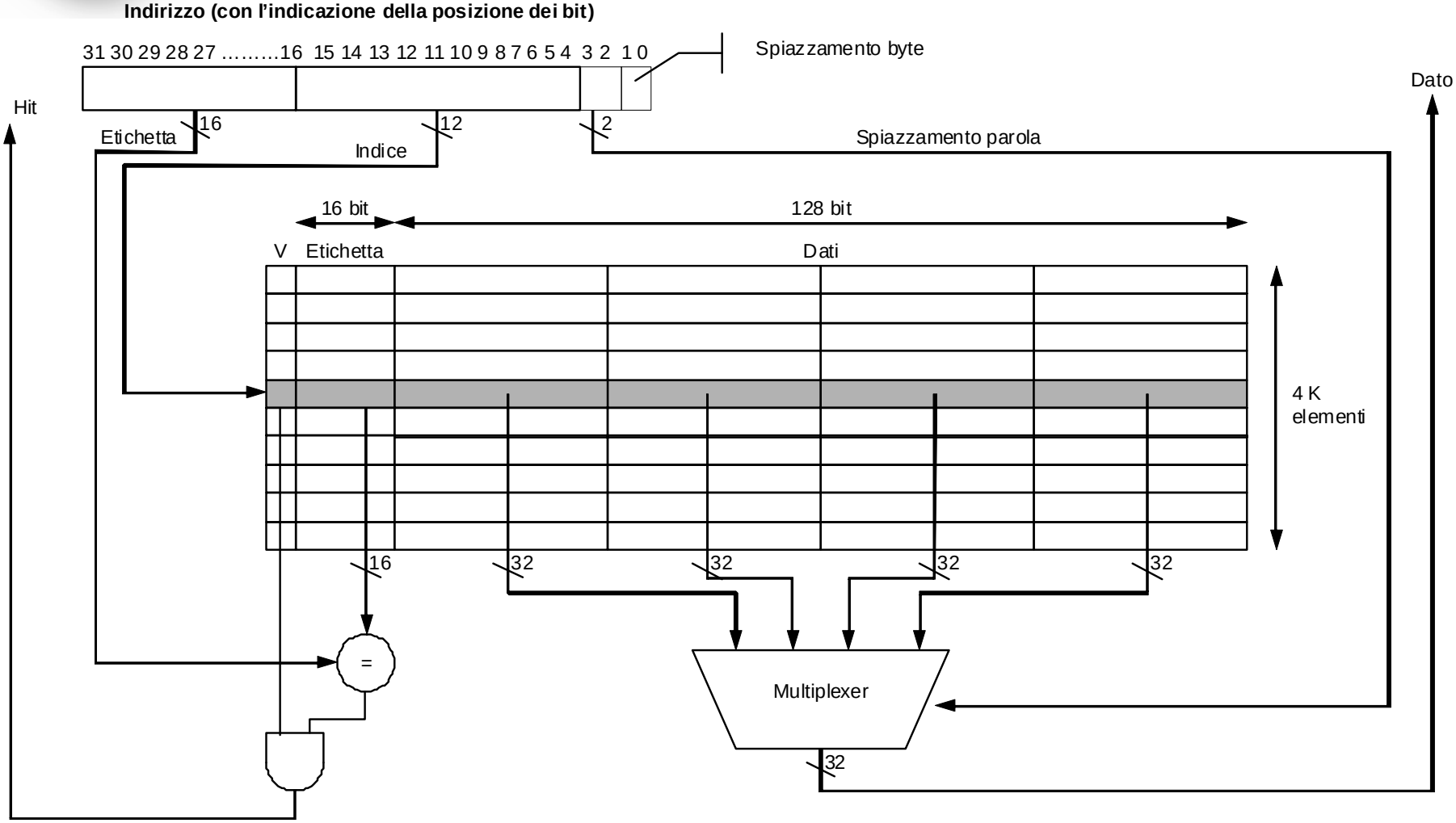
Cache direct-mapped

- Nel caso più generale (ammettendo che la memoria sia indirizzabile al byte): indirizzo di memoria di N bit diviso in 4 campi
  - B bit meno significativi permettono di individuare il singolo byte della parola nel blocco di cache
    - (Se la parola non è indirizzabile per byte è B= 0)
  - K bit per identificare la parola all'interno del blocco di cache
  - M bit per individuare la posizione del blocco in cache
  - N-M-K-B bit di etichetta per verificare che il blocco di cache contenga esattamente l'indirizzo cercato

Cache direct-mapped: esempio

- Esempio: memoria centrale 4Gigabyte (32 bit indirizzo) , con parole di 32 bit
- Memoria indirizzabile al *byte* (quattro byte in una parola); i due bit meno significativi dell'indirizzo identificano il byte nella parola;
- Blocchi da 4 parole (16 byte): 2 bit per identificare la parola nel blocco,
- Memoria cache da 16K byte (= 2^{14} parole): occorrono 12 bit per indirizzare un blocco (n° blocchi = $2^{14}/2^{2}$)
- Etichetta di 16 bit (= 32 – 12 – 4)

Cache direct-mapped: esempio



Cache direct-mapped: placement

- Per *trasferire* un blocco in cache:
  - si accede al blocco di cache il cui indice corrisponde ai $\log_2 N_b$ bit meno significativi dell'indirizzo del blocco in memoria;
  - nel tag si scrivono gli N-M-K-B bit più significativi dell'indirizzo e nella parte "dato" si scrive il contenuto del blocco;
  - Si pone a 1 il bit di validità.
- Si noti: la *sostituzione* è banale, dato che ogni blocco di RAM può avere un'unica destinazione in cache!
- Si veda una sequenza di operazioni su una cache di otto blocchi (per semplicità, blocchi di una parola) disponendo di una RAM di 32 blocchi:

Cache direct-mapped: esempio di sequenza di accessi

- Inizialmente: cache “vuota”, bit di validità tutti a 0 (il contenuto degli altri campi è privo di significato!):

Indice	V	Tag	dati
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	0		
111	0		

Cache direct-mapped: esempio di sequenza di accessi

Prima Operazione: si vuole leggere la parola di memoria con indirizzo 10110 ⇨ miss seguito da trasferimento

Indice	V	Tag	dati
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache direct-mapped: esempio di sequenza di accessi

Situazione intermedia dopo un certo numero di letture

Indice	V	Tag	dati
000	1	10	Mem[10000]
001	0		
010	1	11	Mem[11010]
011	1	00	Mem[00011]
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache direct-mapped: esempio di sequenza di accessi

- Si vuole ora leggere la parola con indirizzo 10010 ⇨ il tag contenuto nel blocco con indice 010 NON è 10, si genera miss seguito da un nuovo trasferimento

Indice	V	Tag	dati
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[101110]
111	N		

Cache direct-mapped

- Numero totale di bit in una cache D.M.: dipende sia dalla dimensione della cache sia da quella dell'indirizzo ⇨ occorre lo spazio per memorizzare dati, tag e bit di validità.
- Es.: indirizzo di 32 bit orientato al byte (i due bit meno significativi dell'indirizzo non riguardano l'indirizzo in cache), blocchi di 2^m parole, cache di 2^n blocchi ⇨ tag lungo $32 - (n+m+2)$ bit; cache di $2^n * (m * 32 + (32 - n - m - 2) + 1)$ bit.
- *Di norma, si indicano solo le dimensioni dell'area dati.*

Cache direct-mapped

- Come scegliere la dimensione del blocco?
  - Blocchi più grandi ⇨ si sfrutta meglio la località spaziale, però
  - A parità di numero di bit nella cache, se il blocco cresce troppo il numero dei blocchi diminuisce eccessivamente – *aumenta la probabilità di dover sostituire un blocco in cache per trasferirvi l'informazione necessaria*;
  - Inoltre: blocchi troppo grandi aumentano il *costo di un miss* – aumenta la penalità del trasferimento dal livello inferiore, il cui tempo è la somma del tempo dovuto al trasferimento della prima parola e di quello dovuto al resto del trasferimento (si deve tener conto della dimensione del bus fra la RAM e la cache!)
- Sperimentalmente: dimensioni utili del blocco per ottenere miss rate accettabili sono da 16 a 128 *byte* (= da 4 a 32 parole);
- Dimensione più frequente del blocco: 32 byte.

Cache direct-mapped

Si noti:

- Nella cache direct mapped si sfrutta il principio di località spaziale ma non quello di località temporale – al momento della sostituzione non si verifica da quanto tempo un blocco si trovi in cache e quindi quanto sia probabile che si tenti in breve tempo di accedere nuovamente a un blocco che è stato appena sostituito!

Gestione di un cache miss in lettura

- La memoria cui la CPU accede direttamente è ora la cache:
  - L'unità di controllo chiede il dato alla CPU; se la risposta è uno hit, l'informazione viene inviata da cache a CPU;
  - Se è un miss, il *controllore della cache* (una specifica unità hardware associata alla cache) dà inizio alla lettura in RAM e al trasferimento in cache; la CPU viene bloccata (situazione di *stallo*, ben diversa da un'eccezione! La CPU "si ferma");
  - Si consideri ad esempio un cache miss durante la lettura di un'istruzione:

Gestione di un cache miss in lettura

1. Il miss viene identificato durante la fase di lettura dell'istruzione – si noti, *il PC è già stato incrementato*.
2. Il PC dell'istruzione che ha generato miss (cioè il PC attuale *decrementato*!) viene inviato alla RAM insieme a un segnale di lettura;
3. L'informazione letta dalla RAM viene trasferita nella sezione dati dell'opportuno blocco di cache (nota: *si legge l'intero blocco*), provocando la *sostituzione del blocco ivi presente se valido*, i bit più significativi dell'indirizzo vengono scritti nel campo tag del blocco, il bit di validità viene posto a 1;
4. Si ripete il primo passo dell'istruzione, cioè la lettura.

Gestione della cache: la scrittura

- Si supponga che la scrittura riguardi una parola presente in cache; se la *scrittura* modificasse solo la cache, si avrebbe *inconsistenza* fra RAM e cache (allo stesso indirizzo sono associati due valori diversi dell'informazione contenuta) ⇨ è indispensabile mantenere la consistenza fra le due memorie!
- Soluzione più semplice: la scrittura scrive sempre *simultaneamente* in RAM *e* in cache – *soluzione write-through*;
- Se all'atto della scrittura si genera miss (si vuole scrivere in una parola che non è presente in cache) prima si esegue una lettura, portando da RAM in cache il blocco che si vuole modificare, poi si procede alla scrittura.

Gestione della cache: la scrittura

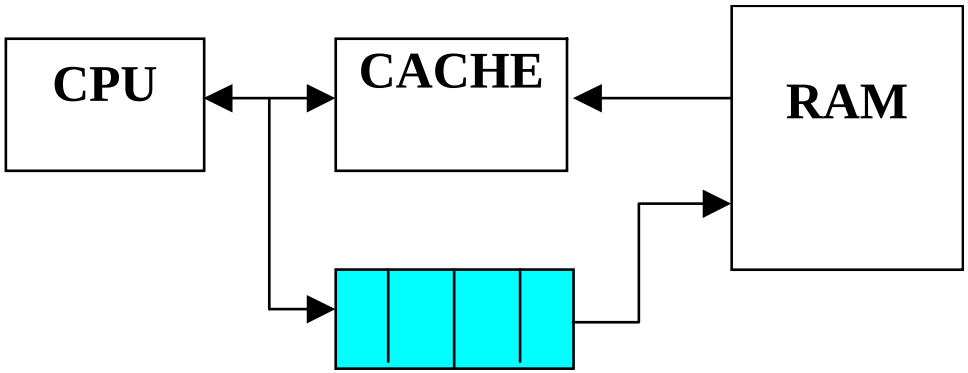
- Soluzione indicata: semplice, ma con molti difetti:
  1. la scrittura avviene alla velocità della RAM, cioè è *lenta*!
  2. Se si operano più modifiche successive della stessa parola prima che si renda necessario sostituirla, si eseguono altrettante scritture in RAM (le prestazioni diminuiscono drasticamente – una scrittura in RAM può prendere 100 cicli contro 1 dell'operazione sulla cache…)
- Primo miglioramento: si introduce un *write buffer* che all'atto di una scrittura memorizza i dati provenienti dalla CPU in attesa di trasferirli alla RAM (simultaneamente, i dati vengono scritti in cache).
  - La scrittura nel buffer avviene alla velocità della CPU, quella dal buffer alla RAM alla velocità della RAM - si disaccoppiano RAM e CPU; se le scritture sono relativamente rare, si maschera per la CPU la lentezza delle scritture in RAM

Gestione della cache: la scrittura

- Write buffer: piccola memoria first in – first out (FIFO) (gestita come un registro a scorrimento):
  1. Elemento del write buffer: contiene indirizzo e dato;
  2. In caso di scrittura, la CPU invia il dato modificato a cache e write buffer, insieme al comando di scrittura;
  3. Il *controllore del write buffer*, in modo indipendente dalle successive operazioni della CPU, completa il trasferimento in RAM; la posizione nel write buffer viene "svuotata" (il contenuto del buffer viene "fatto scorrere" verso destra);
  4. Se all'atto di una scrittura il Write Buffer è pieno, la CPU viene posta in stallo in attesa che si liberi una posizione nel Write Buffer.
- Dimensione ottima del write buffer: definita sperimentalmente sulla base di statistiche (valore tipico: 4-8 posizioni).

Gestione della cache: la scrittura

- Write buffer fra cache and RAM:
  - La CPU scrive nella cache *e* nel write buffer
  - Il controllore di memoria scrive dal buffer alla RAM



Write buffer

Gestione della cache: la scrittura

- Soluzione alternativa alla write-through: *write back*. Al momento di una scrittura si scrive *solo in cache*. *Il blocco modificato viene copiato in RAM solo al quando deve essere sostituito da un altro blocco*. Si noti: nell'intervallo, c'è inconsistenza fra cache e RAM.
- Grazie alla località, è probabile che *si scriva più volte nello stesso blocco in cache prima che questo venga sostituito* - le scritture in RAM sono meno frequenti che nel caso precedente. Al momento della sostituzione si scrive l'intero blocco, non una sola parola.
- Problema: ogni sostituzione implicherebbe prima una scrittura in RAM e poi una lettura da RAM – la miss penalty raddoppierebbe.

Gestione della cache: la scrittura

- Per migliorare l'efficienza dello schema write-back e riscrivere in RAM non tutte le volte che si fa una sostituzione ma solo se il blocco da togliere dalla cache ha subito modifiche, si aggiunge al blocco di cache un ulteriore bit, il *dirty bit*. Al momento in cui un blocco viene trasferito in cache, il dirty bit viene posto a 0; lo si porta a 1 nell'istante in cui si effettua una scrittura. Si *scrive* un blocco dalla cache alla RAM al momento di una sostituzione (quindi prima di leggere il nuovo blocco dalla RAM alla cache) *se e solo se il dirty bit del blocco vale 1*.
- Lo schema write-back ha implementazione più complessa del write-through, ma migliori prestazioni.

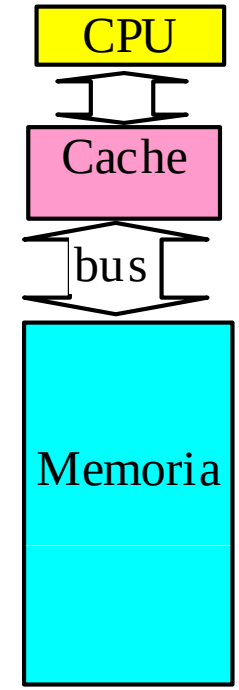
Il progetto del sistema di memorie per supportare le cache

- La velocità del bus di memoria influenza la miss penalty; la frequenza del clock di bus di memoria è di norma molto più bassa di quella del clock di CPU;
- Si può ridurre la miss penalty se ampliando la larghezza di banda dalla RAM alla cache;
- Esempio: si supponga che occorrano:
  1. Un ciclo di clock *del bus di memoria* per inviare l'indirizzo;
  2. 15 cicli per ogni accesso alla DRAM iniziato;
  3. Un ciclo per trasferire una parola di dati da RAM a cache.

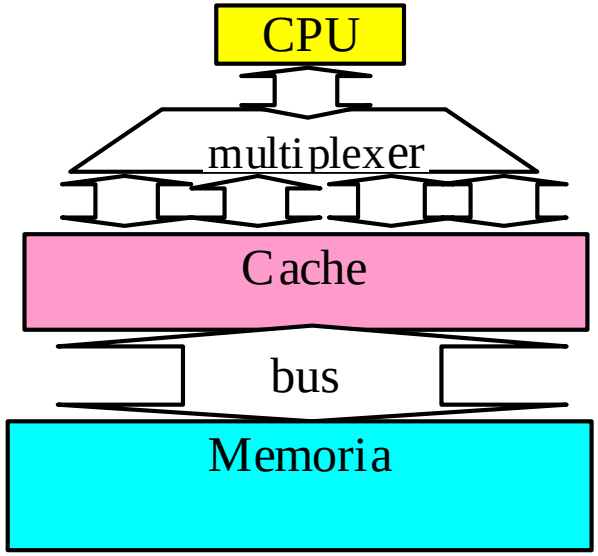
Il progetto del sistema di memorie per supportare le cache

- Se il blocco di cache è di quattro parole e la DRAM trasferisce una parola per volta(accesso *sequenziale)*, la miss penalty è pari a 1 + 4x15 + 4x1 = 65 cicli di clock del bus di memoria. Il numero di byte trasferiti per ciclo di bus è quindi (4x4)/65 = 0.25.
- Si possono immaginare tre opzioni per progettare il sistema di memorie:
  1. Soluzione vista ora: DRAM larga una parola, accessi sequenziali;
  2. Banda di memoria e di bus ampie quanto il blocco: si ammettono accessi paralleli a tutte le parole del blocco. Una soluzione intermedia porta a una banda pari a *parte* del blocco.
  3. Si amplia la memoria (organizzata in banchi paralleli) ma non il bus – si paga una volta sola il costo della latenza di accesso, si trasmette ancora una parola alla volta.

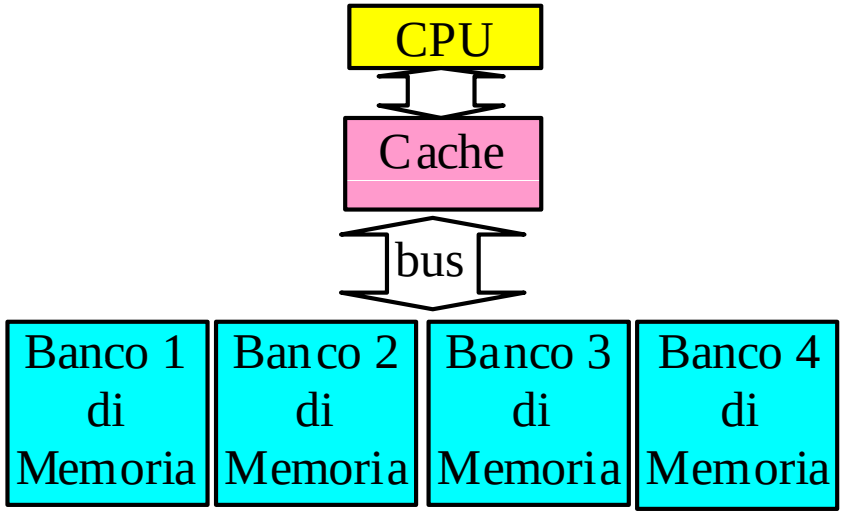
Il progetto del sistema di memorie per supportare le cache



Memoria larga una parola



Organizzazione a Memoria e bus larghi



Organizzazione a Memoria *interleaved*

Il progetto del sistema di memorie per supportare le cache

- Nel primo caso, gli accessi alle parole di un blocco avvengono in sequenza,
- Nel secondo, aumentando larghezza di bus e di memoria si aumenta in proporzione la *banda* della memoria ⇨ diminuiscono le componenti della penalità di memoria dovute a tempo di accesso e a tempo di trasferimento. Per l'esempio già visto, si aumenti la banda di memoria e di bus a due parole ⇨ la penalità di memoria diventa ora 1 + 2x15 + 2x1 = 33 cicli di clock del bus di memoria (la banda per un miss è pari a 0.48 byte per ciclo). Se la banda sale a *quattro* parole, la penalità è di soli 17 cicli (banda di 0.94 byte per ciclo).

Il progetto del sistema di memorie per supportare le cache

- Nel terzo caso, la larghezza di bus resta pari a una parola, ma si organizzano quattro banchi di memoria in parallelo (ognuno largo una parola) – soluzione detta *interleaved* (intrallacciata). Si manda l'indirizzo simultaneamente a tutti i quattro banchi, e si comandano le quattro letture simultaneamente; i trasferimenti avvengono uno per volta. Occorrono ora 1 ciclo per inviare l'indirizzo e il comando di lettura ai quattro banchi, 15 cicli per leggere in parallelo i dati dai quattro banchi, quattro cicli per trasmettere in sequenza le quattro parole sul bus ⇨ penalità pari a 20 cicli di bus di memoria (banda per miss pari a 0.80 byte per ciclo).

Il progetto del sistema di memorie per supportare le cache

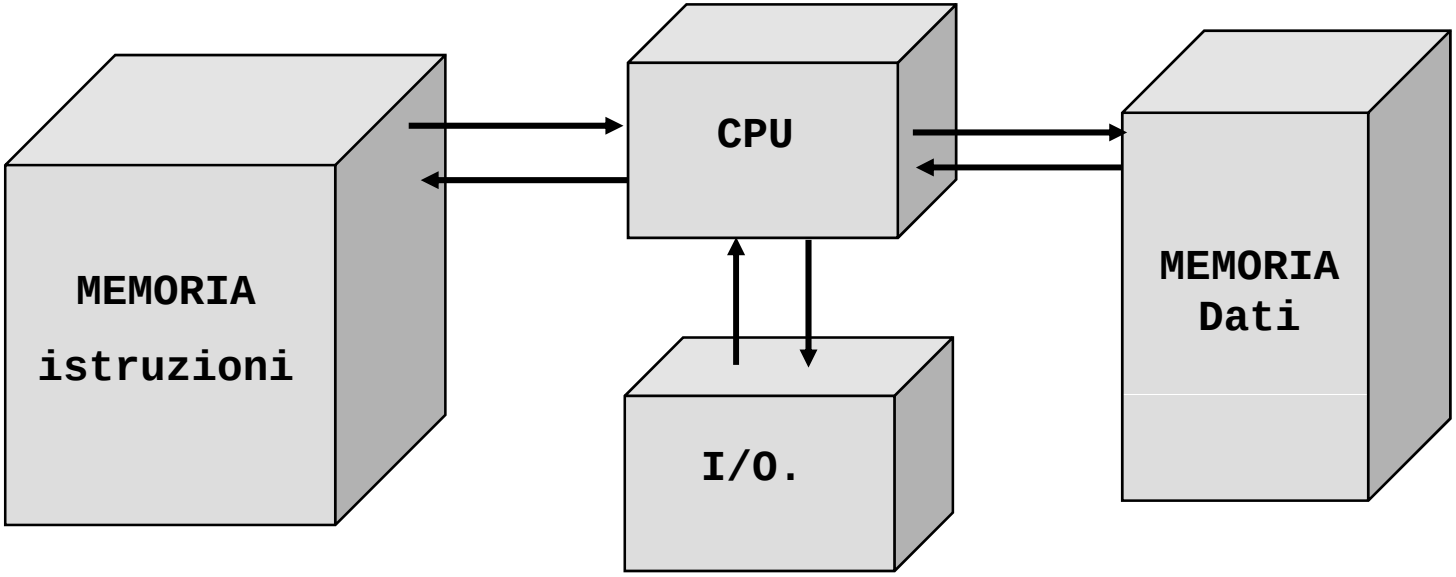
- La soluzione a banchi è molto attraente (il costo del bus è molto più ridotto); è utile anche in scrittura (si scrive indipendentemente nel singolo banco – utile in particolare per la strategia di scrittura *write-through*).

Come migliorare le prestazioni della cache

- Aumentando le dimensioni del blocco di cache, il numero di miss si riduce fino a un certo punto – superato un valore limite (tipicamente, 8-16 parole) in realtà le prestazioni peggiorano perché nella sequenza di istruzioni sono presenti istruzioni di salto che riducono l'efficienza in termini di località spaziale;
- Conviene introdurre cache *separate* per istruzioni e dati (**split cache**)
  - Le operazioni di lettura/scrittura possono essere svolte in modo indipendente in ognuna delle due cache ⇨ di fatto si raddoppia la larghezza di banda della memoria
  - Le caratteristiche di località sono molto diverse per istruzioni e per dati.
  - Il processore ora deve avere due porte di collegamento alla memoria

Come migliorare le prestazioni della cache

- La *split cache* porta a uno schema *fisico* simile alla cosiddetta “architettura di Harvard” – abbiamo ancora un unico spazio logico degli indirizzi, ma due aree fisiche distinte:



Come migliorare le prestazioni della cache

- Soluzioni più brillanti: si passa ad architetture della cache diverse da quella direct-mapped, che sfruttino anche la località temporale oltre a quella spaziale!
- Soluzione estrema: un qualsiasi blocco di RAM può mapparsi su *qualunque* blocco di cache ⇨ soluzione *totalmente associativa*.

Cache totalmente associativa

- L'elemento in base a cui si reperisce un elemento in cache è *l'indirizzo che esso ha nella memoria primaria* ⇨ l'elemento in cache ora deve essere "indirizzabile mediante parte dell'informazione contenuta nel blocco" (la cache è ora una *Content Addressable Memory* – CAM – detta anche *memoria associativa*);
- La linea di cache è ora costituita dal blocco di dati, dal bit di validità, e *dall'intero indirizzo in memoria primaria* (che costituisce il *tag*): la ricerca dell'informazione avviene confrontando *in parallelo* l'indirizzo cercato con tutti i tag.
- La libertà di *placement* è ora massima.

Cache totalmente associativa

- Al momento in cui si deve sostituire un blocco in cache, trasferendone uno dalla memoria di livello inferiore, occorre una *politica di sostituzione*: per garantire la località temporale, si può sostituire il blocco di cache *meno recentemente usato* (politica *LRU* – least recently used)- in pratica a ogni blocco della cache si associa un contatore verso il basso che viene portato al valore massimo ogni volta che si accede al blocco e diminuito di uno ogni volta che si accede ad un blocco diverso: il candidato per la sostituzione è un blocco il cui contatore ha valore minimo, a cui quindi non si è fatto riferimento da più tempo.

Cache totalmente associativa

- Soluzione LRU: teoricamente ottima, in pratica molto costosa (l'esperienza indica comunque contatori saturanti con modulo piuttosto piccolo).
- Alternativa "approssimata": si sostituisce uno dei blocchi *scritti* meno recentemente (si considera per la località temporale solo il momento del placement, non l'attività di lettura); a ogni blocco si associa un registro FIFO in cui si inserisce un 1 al momento della scrittura e che viene fatto scorre verso destra a ogni scrittura in un blocco diverso: al momento della sostituzione, sono candidati i blocchi che hanno il bit 1 nella posizione più a destra.

Cache totalmente associativa

- Soluzione ancora più banale – sostituzione casuale (politica *random*).
- In pratica, la soluzione totalmente associativa non viene usata per cache di dimensioni ragionevolmente ampie (dalle decine di migliaia di byte in su) a causa del costo elevato in termini circuitali e di consumo di potenza (il tag è molto lungo, la circuiteria di confronto sull'intera etichetta per ogni blocco occupa molta area, la ricerca parallela su tutti i blocchi a ogni accesso consuma molta potenza).

Cache set-associativa

- Compromesso fra soluzione direct-mapped e soluzione totalmente associativa: *cache set-associativa a n vie*.
- La cache è divisa in un certo numero di *insiemi* (lo stesso vale per la RAM): un *insieme* è *indirizzato mediante i bit meno significativi dell'indirizzo*, *secondo la tecnica direct-mapped*;
- Ogni insieme contiene un certo numero di blocchi – *n blocchi per ogni insieme di una cache set-associativa a n vie (n almeno = 2)*. Ogni insieme della RAM contiene un numero di blocchi molto più elevato: ogni blocco di un insieme della RAM può mapparsi su qualsiasi blocco del corrispondente insieme della cache.

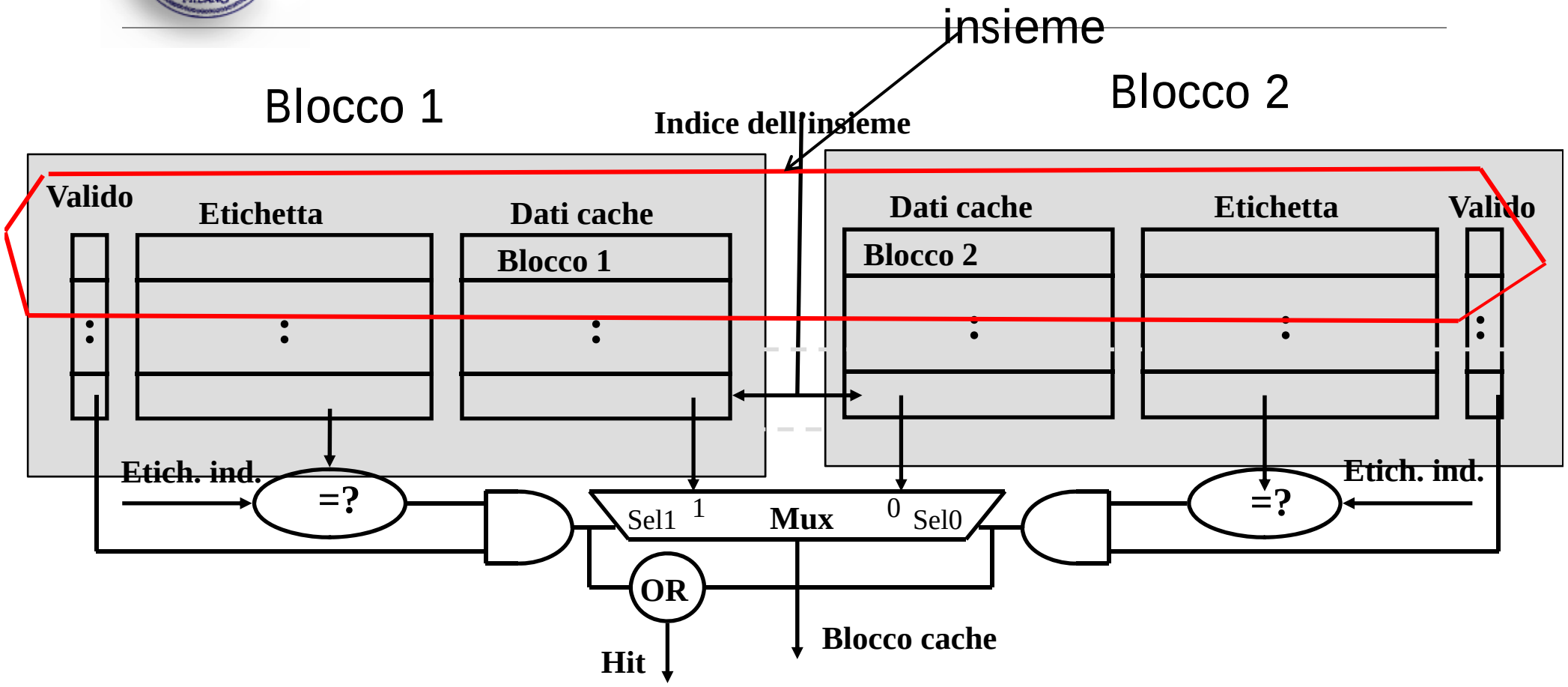
Cache set-associativa

- Ogni blocco della memoria centrale può essere portato in un unico *insieme* della cache, ma il blocco può essere poi scritto in uno *qualsiasi* dei blocchi di questo insieme
  - Combina la modalità a indirizzamento diretto per gli insiemi della cache, e la modalità completamente associativa per i blocchi all'interno dell'insieme
- Un indirizzo di memoria di N bit è suddiviso in 4 campi:
  - B bit meno significativi per individuare il byte all'interno della parola
  - K bit per individuare la parola all'interno del blocco
  - M bit per individuare l'indice dell'insieme (*indirizzamento diretto per l'insieme*)
  - N-(M+K+B) come etichetta, identifica il blocco nell'insieme (*ricerca completamente associativa nell'insieme*)

Cache set-associativa a due vie

- Cache a due vie: ogni insieme è costituito da 2 blocchi
- La parte di indirizzo che individua l'insieme seleziona simultaneamente i due possibili blocchi della cache
- L'etichetta dell'indirizzo cercato viene confrontata in parallelo con le due etichette dei due blocchi dell'insieme
- Il blocco viene identificato in base al risultato dei due confronti (o eventualmente viene dichiarato miss se ambedue i confronti danno esito negativo).

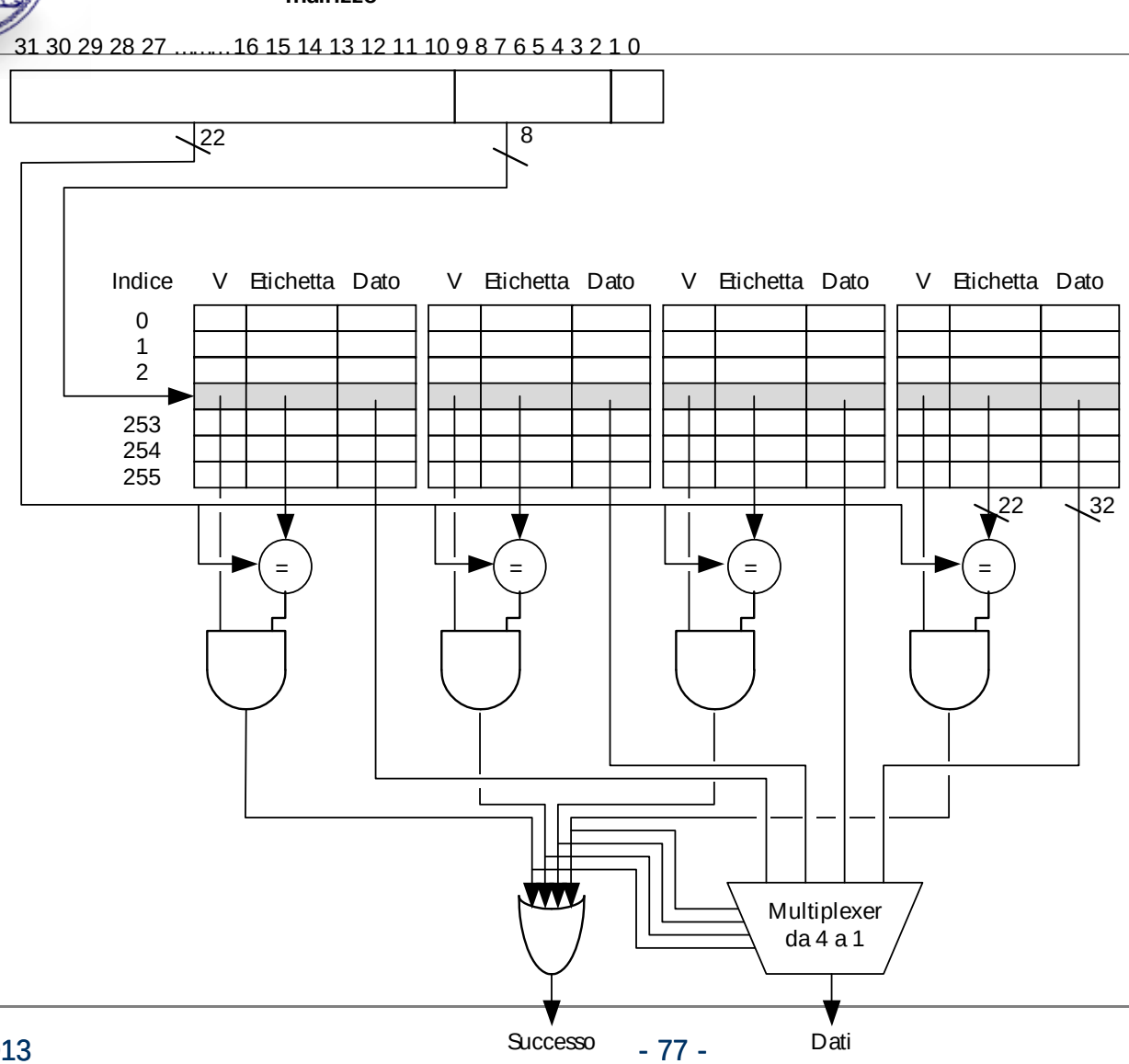
Cache set-associativa a due vie



Cache set-associativa a quattro vie

- A 4 vie: ogni blocco di memoria centrale può essere caricato in uno fra 4 blocchi di cache: ogni insieme è di 4 blocchi
- Memoria centrale 4Gbyte: indirizzo di 32 bit
- Memoria cache 1K parole (1 parola=4byte), blocchi da 1 parola:
- Organizzazione dell'indirizzo:
  - Bit 0 e 1 per indirizzare i byte
  - Numero blocchi nella cache = dimensioni della cache/dimensioni del blocco= 1024/1 = 1024 ($=2^{10}$)
  - Numero di insiemi nella cache= numero di blocchi/ dimensioni dell'insieme = 1024/4 = 256 ($= 2^{8}$)
  - Bit 2-9 indice dell'insieme nella cache
  - Bit 31-11 etichetta

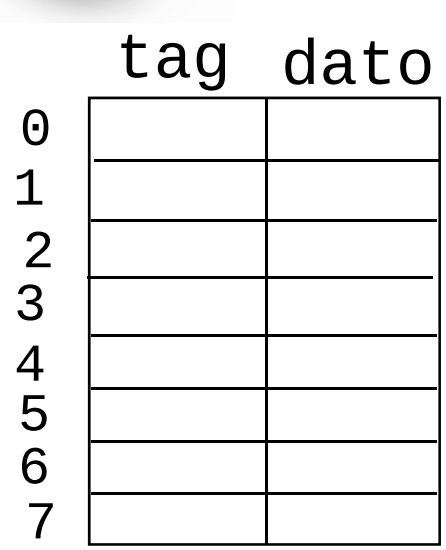
Cache set-associativa a quattro vie



Cache set-associativa: esempio

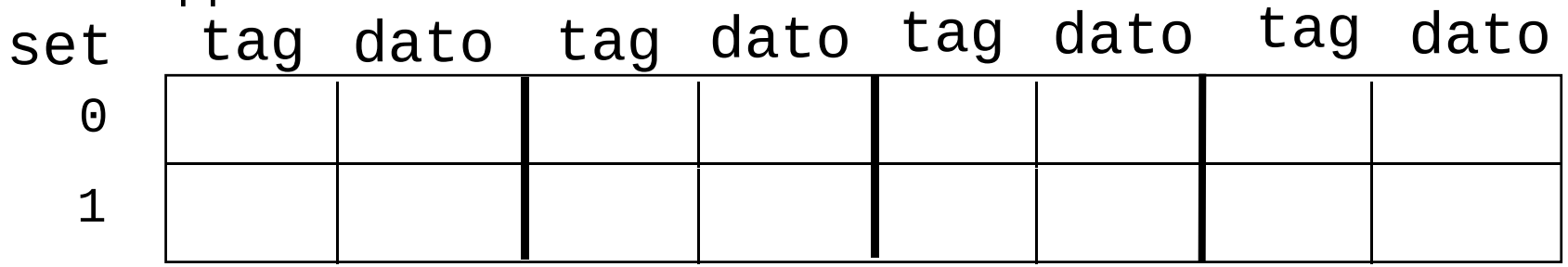
- Si considerino le organizzazioni di una cache di otto blocchi nelle versioni direct mapped (che può essere interpretata come set-associativa a una via), set-associativa a due vie, set-associativa a quattro vie.

Cache set-associativa: esempio



Direct-mapped

Set-associativa a due vie



Set-associativa a quattro vie

Cache miss e associatività: esempio

- Si abbiano tre piccole cache di quattro blocchi ciascuna; la prima totalmente associativa, la seconda set-associativa a due vie, la terza direct mapped. Si trovi il numero di miss per ognuna delle tre organizzazioni data questa sequenza di indirizzi di blocchi: 0, 8, 0, 6, 8
- Il caso più semplice è quello direct-mapped. Si valuta dapprima su quale blocco della cache si mappa ogni indirizzo:

Indirizzo del blocco	Blocco di cache
0	0
6	2
8	0

Cache miss e associatività: esempio

- Si consideri ora la dinamica degli accessi in una cache direct-mapped (la cella vuota è “invalida”):

Indirizzo del blocco in RAM	Hit o miss	Contenuto dei blocchi di cache dopo il riferimento a memoria			
		0	1	2	3
0	Miss	M[0]			
8	Miss	M[8]			
0	Miss	M[0]			
6	Miss	M[0]		M[6]	
8	Miss	M[8]		M[6]	

Cache miss e associatività: esempio

- La cache direct-mapped ha generato cinque miss in cinque accessi (evidentemente, è un caso sfortunato!)
- Si consideri ora una cache set-associativa con due set (0 e 1) ognuno dei quali contiene due blocchi. Le regole di possibile mapping da RAM a cache sono le seguenti (i blocchi con indirizzo che termina per 0 si mappano sul set 0, quelli il cui indirizzo termina per 1 sul set 1):

Indirizzo del blocco		Insieme della cache
0	⇨	0
6	⇨	0
8	⇨	0

Cache miss e associatività: esempio

- Occorre scegliere la regola di sostituzione: si usa quella del “blocco meno recentemente usato”. La successione di operazioni sulla cache è ora la seguente:

Indirizzo del blocco in RAM	Hit o miss	Contenuto dei blocchi di cache dopo il riferimento a memoria			
		Set 0	Set 0	Set 1	Set 1
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[6]		
8	Miss	M[8]	M[6]		

Cache miss e associatività: esempio

- Ancora un caso sfortunato, ma il numero di miss è sceso a 4.
- infine, cache totalmente associativa di quattro blocchi; ogni blocco di memoria si mappa indifferentemente su qualsiasi blocco di cache; si usa la politica di sostituzione LRU.

Indirizzo del blocco in RAM	Hit o miss	Contenuto dei blocchi di cache dopo il riferimento a memoria			
		blocco 0	Blocco 1	Blocco 2	Blocco 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[8]	M[6]	
8	hit	M[0]	M[8]	M[6]	

Cache miss e associatività: esempio

- Il numero di miss è sceso a tre. Come si vede, dimensioni della cache e grado di associatività influiscono sulle prestazioni della cache; il costo in termini circuitali però cambia anch'esso in modo rilevante.
- Sperimentalmente: associatività a quattro vie dà buoni risultati, a otto vie si raggiungono quasi le prestazioni di una cache totalmente associativa.

Confronto fra diverse architetture di cache

- Rispetto alla Cache a indirizzamento diretto, la Cache set-associativa a N vie:
  - Richiede N comparatori invece di 1
  - Comporta un maggior ritardo dovuto all'ulteriore MUX aggiunto sul percorso di lettura dei dati
  - I dati sono disponibili solo **dopo** la generazione del segnale di Hit/Miss

- In una cache a indirizzamento diretto, il blocco di cache richiesto è disponibile prima del segnale di Hit/Miss:
  - Possibile ipotizzare un successo e quindi leggere comunque il dato, che successivamente non verrà instradato alla CPU se si trattava in realtà di un fallimento.

Confronto fra diverse architetture di cache: mapping

- **Indirizzamento diretto**:
  - Posizione univoca, ricavata dalla divisione “indirizzo di memoria modulo numero dei blocchi in cache”
- **Completamente associativa**:
  - Posizione qualunque all’interno della cache
- **Set associativa**
  - Posizione univoca dell’insieme
  - Posizione qualsiasi del blocco all’interno dell’insieme prefissato

Confronto fra diverse architetture di cache: Identificazione del blocco

- **Indirizzamento diretto**:
  - Identifica blocco
  - Verifica etichetta e verifica bit di validità
- **Completamente associativa**:
  - Confronta etichetta in ogni blocco e verifica bit di validità
- **Set-associativa**
  - Identifica l'insieme
  - Confronta etichette dell'insieme e verifica bit di validità

Confronto fra diverse architetture di cache: sostituzione del blocco

- **Cache a indirizzamento diretto**: blocco da sostituire definito in base all'indirizzo
- **Cache set associative** (o completamente associative): scelta del blocco da sostituire
  - Casuale
  - LRU (Least Recently Used)
    - **Cache a 2 vie**. Esiste un **bit di uso** per ogni blocco dell'insieme: quando un blocco viene utilizzato il suo bit viene messo a 0, l'altro a 1
    - **Cache a 4 vie**. Esiste un **contatore di uso** per ogni blocco dell'insieme: quando un blocco viene utilizzato il suo contatore viene messo a 0, gli altri incrementati di 1. Per limitare i ritardi nell'accesso si introduce un bit di uso per ogni blocco

Calcolo del tempo medio di accesso

- Indipendentemente dalla particolare organizzazione della cache, il tempo medio di accesso T_m può essere valutato partendo dalle cifre caratteristiche della cache:
  - In caso di *hit*, il tempo di accesso è quello della cache;
  - In caso di *miss*, occorre portare il blocco in cache e poi leggerlo

Calcolo del tempo medio di accesso

$T_m = hit\_rate*hit\_time + miss\_rate*miss\_time$

Dove *miss_time* è la somma della *miss penalty* e del tempo di lettura dalla cache, cioè *hit_time* + *miss_penalty*, quindi

$T_m = hit\_rate*hit\_time + miss\_rate*(hit\_time + miss\_penalty) = hit\_time*(hit\_rate + miss\_rate) + miss\_rate*miss\_penalty = hit\_time + miss\_rate*miss\_penalty$

Calcolo del tempo medio di accesso

- Per ridurre T_m a parità di tecnologia si può:
  - Aumentare lo hit rate ⇨ si ottiene aumentando la dimensione della cache e ricorrendo ad architetture più sofisticate
  - Ridurre la miss penalty ⇨ tipicamente si ottiene introducendo più livelli di cache - almeno una seconda cache, detta "di secondo livello" o "L_2", per distinguerla da quella più vicina alla CPU che è "di primo livello" o "L_1", inserita fra cache di primo livello e RAM, con dimensioni maggiori della L_1, costo per byte minore e hit time più elevato rispetto alla cache di primo livello ma sempre molto minore del tempo di accesso in RAM.

Note conclusive

- È molto importante notare che::
  - La gestione della cache è effettuata totalmente da hardware dedicato (*controllore di cache*);
  - La cache è di norma “trasparente” al programmatore – che, salvo nel caso di architetture molto particolari, non ha modo di controllare i trasferimenti a/da cache.