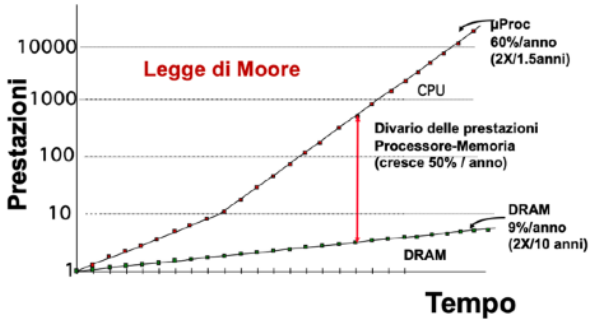
04 - MEMORIA CACHE

OBBIETTIVO

Migliorare le prestazioni di un calcolatore attraverso il sistema di memoria in modo da:
- fornire agli utenti l'illusione di poter usufruire di una memoria contemporaneamente grande e veloce
- fornire al processore istruzioni e dati alla velocità con cui è in grado di elaborarli

Problema: divario delle prestazioni processore-memoria



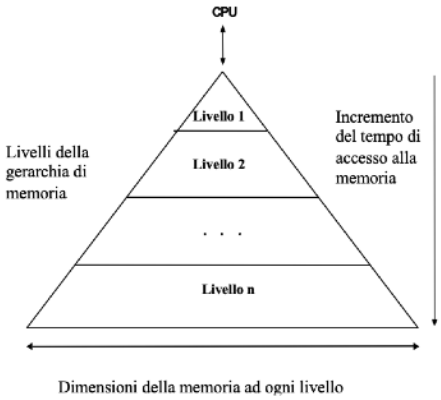
Soluzione: sfruttare il principio di località dei riferimenti

- **Località temporale:** quando si fa riferimento a un elemento di memoria, c'è la tendenza a far riferimento allo stesso elemento entro breve.
  - Esempio: riutilizzo di istruzioni e dati contenuti nei cicli.
- **Località spaziale:** quando si fa riferivano a un elemento di memoria, c'è la tendenza a far riferimento entro breve tempo ad altri elementi che hanno indirizzo vicino a quello dell'elemento corrente.
  - Esempio: sequenzialità delle istruzioni e accesso a dati organizzati in vettori o matrici.

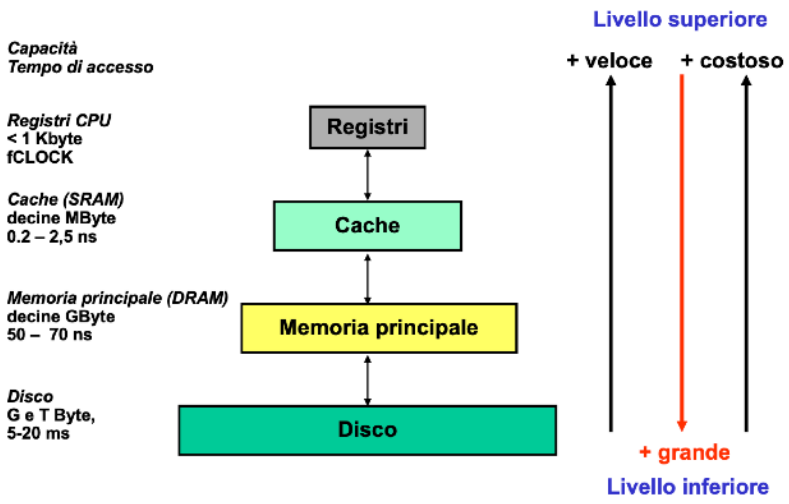
Soluzione: gerarchia di memoria

- Utilizzare diversi livelli di memoria, ciascuno di diversa velocità e dimensione, realizzati con tecnologia diverse in modo da ottenere un buon compromesso costo/prestazioni

- Obbiettivo: fornire all'utente una quantità di memoria pari a quella disponibile nella tecnologia più economica, consentendo allo stesso tempo una velocità di accesso pari a quella garantita dalla tecnologia più veloce

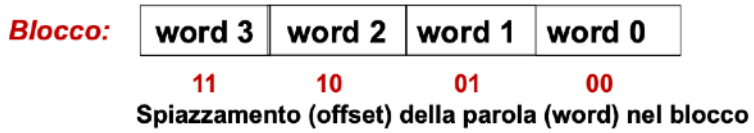


LIVELLI DELLA GERARCHIA DI MEMORIA



MEMORIA CACHE: CONCETTI BASE

- Una **gerarchia di memoria è composta da più livelli**, ma i dati vengono di volta in volta copiati solo tra **due livelli adiacenti**
- Consideriamo i due livelli: **cache** e **memoria principale**
- La **cache** (**livello superiore**) è più piccola, veloce e costosa rispetto alla **memoria principale** (**livello inferiore**)
- La minima quantità di inform. che può essere presente o assente nella cache è il ***blocco*** o ***cache line***
- Per sfruttare la **località spaziale** è necessario che la dimensione del blocco della cache si Sun multiplo della dimensione della parola (word) di memoria:
  - esempio: dimensione blocco = 128 bit = 4 parole da 32 bit



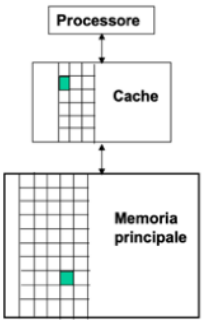
- il numero di blocchi presenti in cache è dato da:

NUMERO BLOCCHI IN CACHE = DIMENSIONE CACHE / DIMENSIONE BLOCCO

esempio: dim cache = 64K byte e blocco da 128 bit cioè 16 byte → la cache contiene: 64 Kbyte / 16 byte = 4 K blocchi

Hit: successo nell'accesso alla cache

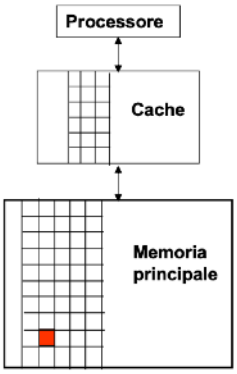
Se il dato (parola) richiesto dal processore **compare in uno dei blocchi presenti nel livello superiore**, si dice che la **richiesta ha successo (hit).**



Miss: fallimento nell'accesso alla cache

- se il dato (parola) **non si trova nel livello superiore**, si dice che **la richiesta fallisce (miss)** → per trovare il blocco che contiene il dato richiesto, bisogna accedere al livello inferiore della gerarchia

- In caso di fallimento (miss) nell'accesso ad un dato:
  - **Stallo** della CPU
  - Richiesta del blocco contenente il dato cercato alla memoria principale
  - Copia del blocco in cache
  - **Ripetizione dell'accesso in cache** (hit)

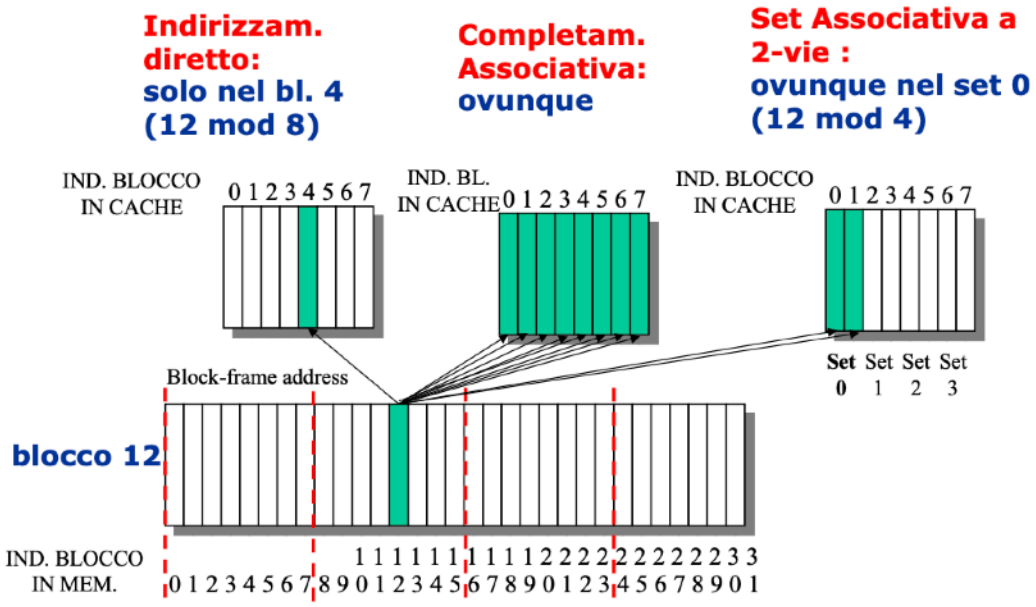


Problema del piazzamento di un blocco

- **Problema**: dato un indirizzo di un blocco nella memoria principale, determinare la sua posizione nella cache.
- Occorre stabilire una corrispondenza tra l'indirizzo in mem del blocco e l'indirizzo del blocco in cache.
- Questa corrispondenza dipende dall'architettura della cache:
  1. **Cache a indirizzamento diretto**
  2. **Cache completamente associativa**
  3. **Cache set-associativa a *n* vie**

Piazzamento del blocco

Data una memoria da **32 blocchi**, dove può essere piazzato il blocco **12** di memoria in una cache da **8 blocchi?**



STRUTTURA DI UNA CACHE

Ogni posizione **(entry) o riga** di una memoria cache include:

1. **Valid bit** che indica se l'entry contiene o meno dati validi. Quando il calcolatore viene acceso tutte le posizioni della cache sono segnalate come NON valide → *cold start misses*

2. **Campo etichetta (tag)** che contiene il valore che identifica univocamente l'indirizzo di memoria corrispondente ai dati memorizzati nella entry

3. **Campo dati** che contiene una copia dei dati **blocco o cache line**

V	TAG	BLOCCO

esempio: cache a ind. diretto da 8 blocchi

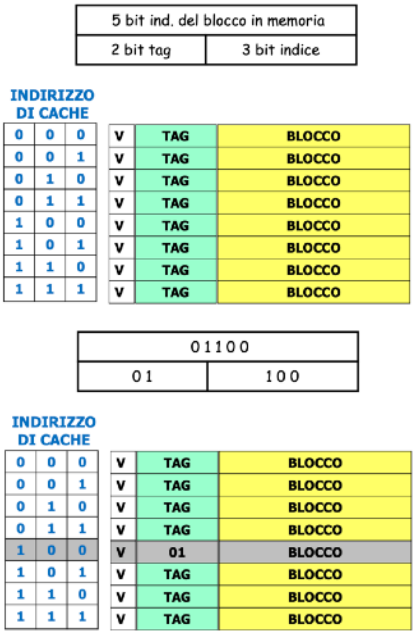
3 bit indirizzo di cache		
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO
V	TAG	BLOCCO

esempio: memoria da 32 blocchi e cache ad ind. diretto da 8 blocchi
Problema: data una memoria da 32 blocchi (5 bit di ind. di memoria) e una cache 8 blocchi (3 bit di ind cache) come realizziamo il zapping tra indirizzi di memoria e indirizzi di cache?

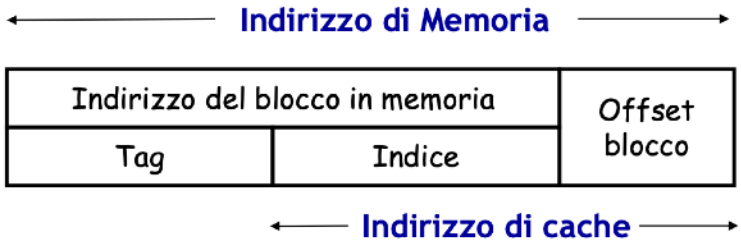
MAPPING DEGLI INDIRIZZI DEI BLOCCHI DI MEMORIA

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31



Consideriamo in blocco di memoria di ind. 12 →

Indirizzo di memoria e indirizzo di cache



*Indirizzo di memoria di **N** bit diviso in campi:*

- **Spiazzamento (offset) del byte** nel blocco individua il byte desiderato all'interno del blocco
- **Indice (index)** serve a identificare l'indirizzo del blocco in cache
- **Etichetta (tag)** da confrontare con il contenuto del campo etichetta del blocco di cache selezionato dall'indice

---

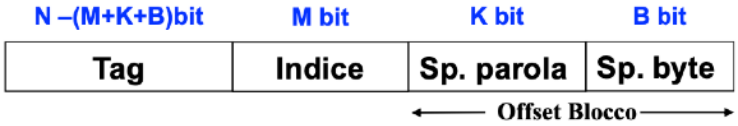
CACHE A INDIRIZZAMENTO DIRETTO (DIRECT MAPPED)

Ogni indirizzo di memoria corrisponde ad uno e un solo indirizzo della cache dato da:

$$(\text{Ind. Blocco})_{\text{cache}} = (\text{Ind. Blocco})_{\text{mem}} \ \text{modulo} \ (\text{Num. Blocchi in cache})$$

esempio: dato l'indirizzo 12 del blocco in memoria, calcolare l'indirizzo del blocco in una cache da 8 blocchi: $(\text{Ind. Blocco})_{\text{cache}} = (12)_{\text{mem}} \ \text{modulo} \ (8) = 4$

Indirizzamento nella cache a indirizzamento diretto



*Indirizzo di memoria di **N** bit diviso in 4 campi:*

- **Spiazzamento (offset) del byte** nella parola individua il byte desiderato all'interno della parola: B bit

    - Se la memoria non è indirizzabile al byte **B = 0**
- **Spiazzamento (offset) della parola** nel blocco individua la parola voluta all'interno del blocco: K bit
    - Se il blocco contiene un sola parola **K = 0**
- **Indice (index)** serve a identificare l'indirizzo del blocco in cache: **M** bit
- **Etichetta (tag)** da confrontare con il contenuto del campo etichetta del blocco di cache selezionato dall'indice: **N – (M + K + B)** bit

esempio 1: cache a indirizzo diretto da 4K Byte e blocco da 32 bit

- Indirizzo di memoria: **N = 32 bit**
- Dimensione della cache **4 K Byte**
- Dimensione del blocco = **1 parola**
    - Sp. (offset) della parola: **K = 0 bit**
- Dimensione della parola = 32 bit = 4 Byte = 22 Byte
    - Sp. (offset) del byte: **B = 2 bit**
- Numero di blocchi = Dim. Cache / Dim. Blocco = 4 KByte / 4 Byte = **1 K blocchi = 2^10 blocchi** → Indice **M = 10 bit**
- **Tag** = (32 – 10 – 2) bit = **20 bit**

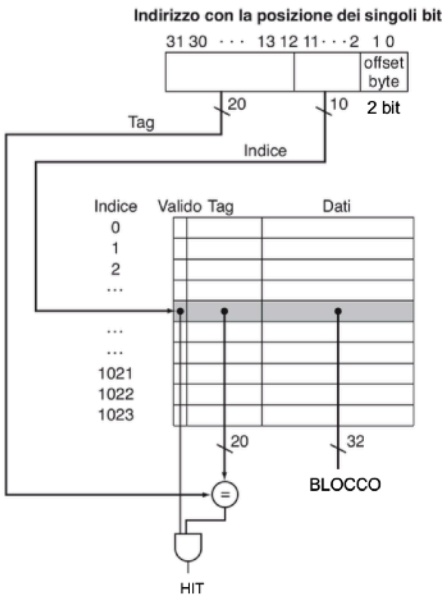
Struttura dell'indirizzo di memoria composto da N=32 bit

20 bit	M = 10 bit	B = 2 bit
Tag	**Indice**	**Sp. byte**

Struttura dell'indirizzo di memoria:

- Un blocco contiene una sola parola da 4 byte => **2 bit offset**
- Cache da 1024 blocchi (2^{10}) => **10 bit di indice**

➢ Restanti **20 bit** per indicare **l'etichetta (tag)** da confrontare

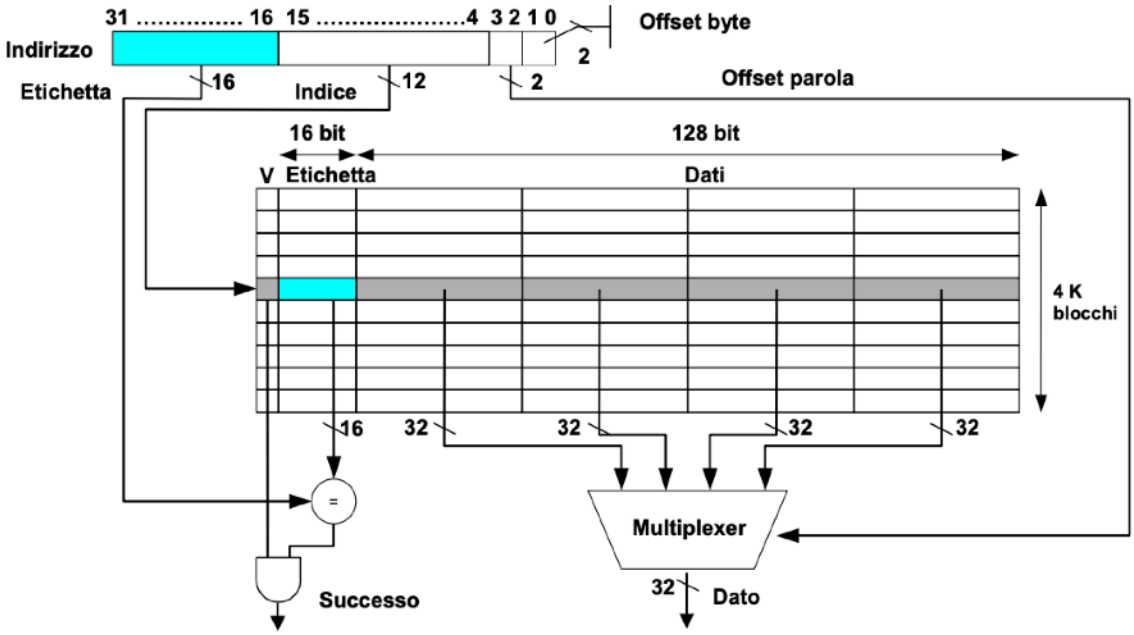


esempio 2: cache a indirizzamento diretto da 64 K Byte e blocco da 128 bit

- Indirizzo di memoria: **N = 32 bit**
- Dimensione della cache **64 K Byte**
- Dimensione del blocco **128 bit** = 4 parole = 22 parole
    - Offset della parola: **K = 2 bit**
- Dimensione della parola = 32 bit = 4 Byte = 22 Byte
    - Offset del byte: **B = 2 bit**
- Numero di blocchi = Dim. Cache / Dim. Blocco = 64 KByte / 16 Byte = **4 K blocchi = 2^12 blocchi** → Indice **M = 12 bit**
- **Tag** = (32 – 12 – 2 – 2) = **16 bit**

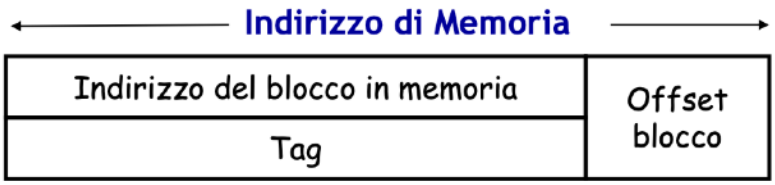
Struttura dell'indirizzo di memoria composto da N=32 bit

16 bit	M = 12 bit	K = 2 bit	B = 2 bit
Tag	**Indice**	**Sp. parola**	**Sp. byte**



CACHE COMPLETAMENTE ASSOCIATIVA

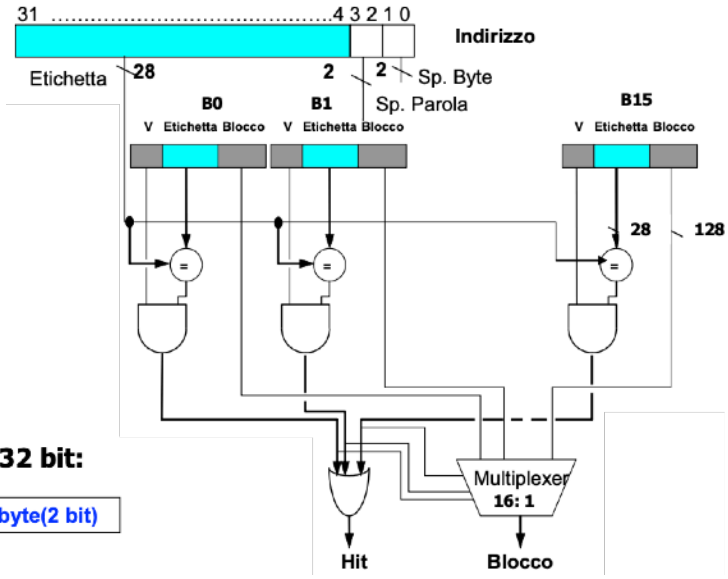
- In una cache **completamente associativa (fully associative)** un blocco di memoria può essere messo in una qualsiasi posizione della memoria cache.
- Poiché il blocco può essere messo in un posto qualsiasi della cache → tutti i blocchi della cache debbono essere esaminati durante la ricerca tramite i tag (etichette)
- Nell'indirizzo di cache **non esiste il campo indice**, ma solo i campi tag e offset:



Mi serviranno 8 comparatori perché ho 8 posizioni della cache da controllare.

cache completamente associativa da 256 byte e blocco do 128 bit

- indirizzo di memoria N = 32 bit
- Dim. Cache 256 Byte
- Dim. Blocco 128 bit = 16 B
- Numero di blocchi = Dim Cache / Dim Blocco = 256 Byte / 16 Byte = 16 blocchi
- Non esiste il campo indice
- Etichetta (tag) = (32 - 2 - 2) bit = 28 bit



Struttura dell'indirizzo di memoria da N = 32 bit:

Tag (28 bit)	Sp. parola (2 bit)	Sp. byte(2 bit)

CACHE SET-ASSOCIATIVA A *N* VIE

- Cache suddivisa in **insiemi (set)** ognuno dei quali da ***n*** blocchi:

Numero di blocchi = Dim. Cache / Dim. Blocco

Numero di set = Dim. Cache / (Dim. Blocco * n)

- Ogni blocco della memoria corrisponde ad un unico insieme della cache di indirizzo

$$(\text{Ind. Set})_{\text{cache}} = (\text{Ind. Blocco})_{\text{mem}} \text{ modulo } (\text{Num. Set in cache})$$

- Il blocco può essere messo in **uno qualsiasi degli n blocchi dell'insieme (set)** → la ricerca del blocco deve essere effettuata su tutti i blocchi dell'insieme individuato dall'indice

Indirizzamento nelle cache set associative a n vie

N –(M+K+B)bit	M bit	K bit	B bit
Tag	**Indice**	**Sp. parola**	**Sp. byte**

Indirizzo di memoria di N bit diviso in 4 campi:

- **Spiazzamento (offset) del byte** nella parola individua il byte desiderato all'interno della parola: **B bit**
- **Spiazzamento (offset) della parola** nel blocco individua la parola voluta all'interno del blocco: **K bit**
- **Indice (index)** serve a identificare l'insieme: **M bit**
- **Etichetta (tag)** da confrontare con il contenuto del campo etichetta dei blocchi dell'insieme selezionato dall'indice: **N – (M + K + B) bit**

Cache set-associativa a 4 vie da 4 KB e blocco da 32 bit

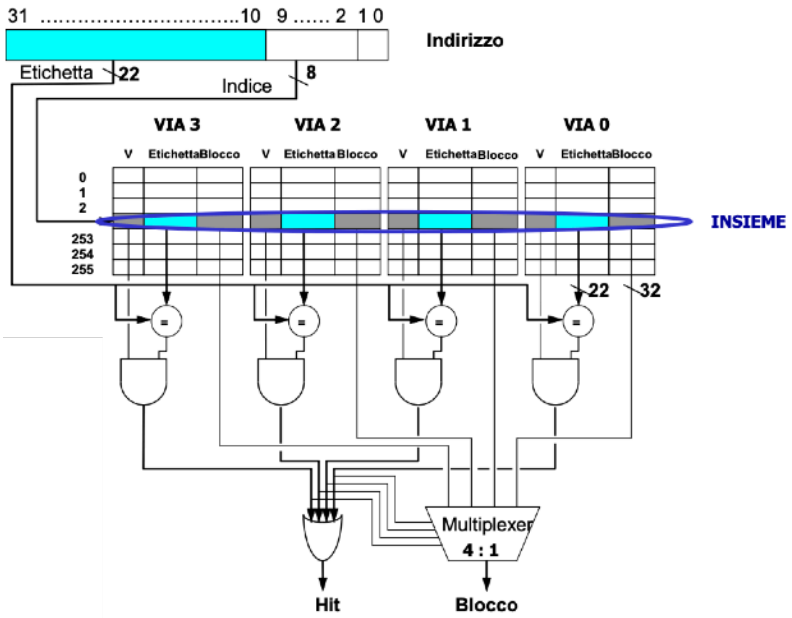
- Indirizzo di memoria: **N = 32 bit**
- Cache set associativa a **n = 4 vie**
- Dim. Cache **4 KByte**
- Dim. Blocco **32 bit** (1 parola = 4 Byte):
  - Sp. della parola: **K = 0 bit**
  - Sp. del byte: **B = 2 bit**

Struttura dell'indirizzo di memoria:

22 bit	M = 8 bit	B = 2 bit
Tag	**Indice**	**Sp. byte**

- Numero di blocchi= Dim. Cache / Dim. Blocco = 4 K Byte / 4 Byte = 1 K blocchi
- Numero di insiemi: Dim. Cache / (Dim. Blocco x n)= 4 K Byte / (4 Byte x 4) = **2^8 insiemi → indice dell'insieme M = 8 bit**
- Tag = (32 – 8 – 2) bit = 22 bit

Struttura simile a 4 cache a indirizzamento diretto affiancate → e usate in parallelo



esempio: cache set associativa a 4 vie

- ogni insieme (set) è costituito da 4 blocchi
- La parte dell'indirizzo di memoria viene confrontata in parallelo con le 4 etichette dei 4 blocchi dell'insieme
- L'etichetta (tag) dell'indirizzo di memoria viene confrontata in parallelo con le 4 etichette dei 4 blocchi dell'insieme
- Il blocco viene selezionato in base al risultato dei 4 confronti

→ in pratica è simile a 4 cache a indirizzamento diretto affiancate e usate in parallelo, accedendo ai 4 blocchi corrispondenti nelle 4 cache, confrontando i 4 campi tag e scegliendo no dei 4 blocchi (oppure nessuno nel caso di miss)

ESERCIZIO SULLA STRUTTURA DEGLI INDIRIZZI

Si consideri una gerarchia di memoria costituita da una cache e una memoria principale caratterizzata da indirizzo di memoria da 32bit

- dimensione cache 256 K Byte
- blocco da 128 bit

Nota: la memoria è indirizzata a livello di byte

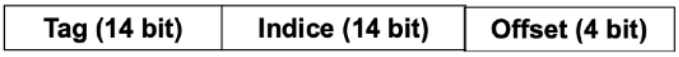
Calcolare la struttura degli indirizzi di memoria nei seguenti casi:

1. Cache a indirizzamento diretto (direct mapped)
2. Cache completamente associativa
3. Cache set-associativa a 4 vie

Soluzione

- **cache a indirizzamento diretto (direct mapped)**

Dimensione del blocco 128 bit = 4 parole = 16 Byte → Offset: 4 bit
Numero di blocchi = dim. cache / dim. blocco = 256 K Byte / 16 Byte = 16 K blocchi = 2^14 blocchi → Indice 14 bit
→ Tag = (32 – 14 – 4) = 14 bit

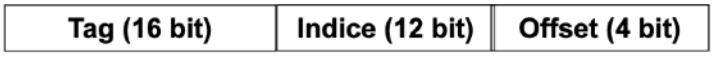


- **cache completamente associativa**

→ non ci sono bit di indice → Tag = (32 - 4) = 28 bit

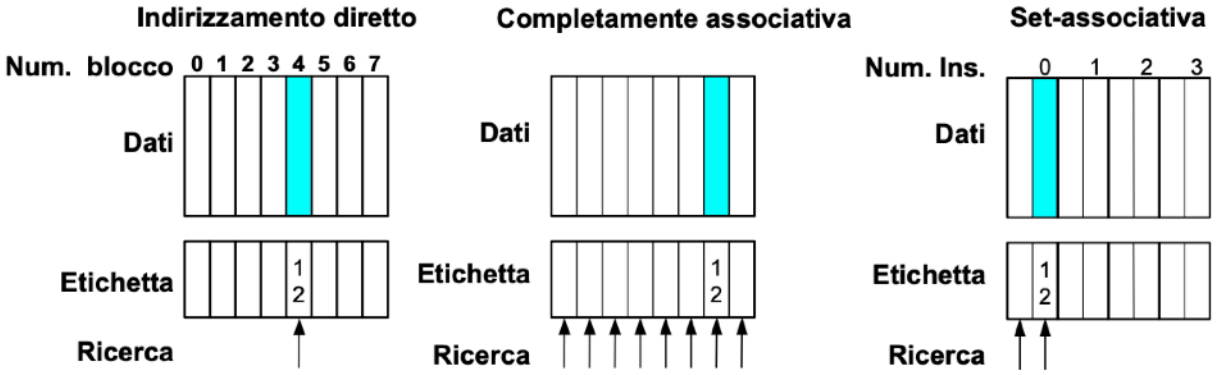
- **cache set-associativa a 4 vie**

Num. set in cache = Dim. Cache / (4 * Dim. Blocco) = 256 K Byte / 64 Byte = 4 K blocchi = 2^12 blocchi
→ Indice 12 bit (serve per l'indirizzare un set)
→ Tag = (32 – 12 – 4) = 16 bit



Identificazione del blocco in cache: sommario

Data una memoria da 32 blocchi, come ricerco il blocco 12 di memoria in una cache da 8 blocchi?



- **indirizzamento diretto** ⟶ identifico posizione con indice, verifico 1 etichetta (tag) e verifico valid bit
- **indirizzamento completamente associativo** ⟶ confronto tutti i tag in ogni blocco e verifico valid bit
- **indirizzamento set-associativo a 2 vie** ⟶ identifico insieme con indice, confronto 2 etichette dell'insieme e verifico valid bit

Effetto dell'incremento dell'associatività sulle prestazioni di una cache

- Esempio di valutazione sperimentale della frequenza di miss considerando una cache di dimensione fissata all'aumentare dell'associatività (numero di vie)
- Si osserva una riduzione della frequenza di miss (miss rate) all'aumentare dell'associatività

Associatività	Frequenza di miss
1	10,3%
2	8,6%
4	8,3%
8	8,1%

Non sembra utile spingere l'associatività a livelli elevati (> 8) perché sarebbe molto costoso da fare.

Incremento dell'associatività

- Principale vantaggio ⟶ riduzione della frequenza di miss (miss rate)
- Principali svantaggi:
  - Maggiore costo implementativo
  - Incremento del tempo di hit (hit time).
- Inoltre all'aumentare dell'associatività diminuiscono i bit di indice e si espandono i bit di tag:

⟶ la scelta tra cache a indirizzamento diretto, completamente associative e set associative dipende dal costo di implementazione dell'associatività (sia in tempo sia in hardware addizionale) rispetto alla riduzione del miss rate

PROBLEMA DELLA SOSTITUZIONE DI UN BLOCCO

Quando si verifica un fallimento (miss) in una cache:

1. Se la cache è a indirizzamento diretto → c'è un solo candidato alla sostituzione
2. Se la cache è completamente associativa → sorge il problema di scegliere quale blocco sostituire: ogni blocco è un potenziale candidato per la sostituzione
3. Se la cache è set-associativa → sorge il problema di scegliere quale blocco sostituire: ogni blocco all'interno dell'insieme (set) è un potenziale candidato per la sostituzione

→ bisogna definire delle politiche per la sostituzione del blocco nelle cache completamente associative o set-associative

Politiche di sostituzione di un blocco in cache associative

1. **Casuale**
2. **Least Recently Used (LRU):** sostituisce il **blocco utilizzato meno di recente**, perché, in base al principio di **località temporale** degli accessi, questo ha la probabilità più bassa di essere richiesto nel prossimo futuro.
3. **FIFO (First In First Out):** sostituisce sempre il blocco caricato per primo in cache, indipendentemente da quando sia stato fatto accesso a tale blocco.

Sebbene il comportamento migliore si ottiene con la politica **LRU**, la sua realizzazione in hardware è piuttosto complessa.

Problema della strategia di scrittura

Possibili strategie per la gestione delle scritture:

1. **Write-through**: l'informazione viene scritta sia nel blocco della cache sia nel blocco della memoria principale.
2. **Write-back**: l'informazione viene scritta **solo** nel blocco della cache. Il blocco modificato sarà scritto nella memoria principale solo quando sarà sostituito a causa di un successivo miss (di tipo read miss oppure write miss):
    - Al termine della scrittura nella cache il blocco di cache diventa **dirty (modified)** e la memoria principale conterrà un valore vecchio rispetto a quello presente nella cache => memoria principale e cache **non sono coerenti**.
    - Solo quando il blocco sarà sostituito a causa di un miss e sarà scritto in memoria, la memoria sarà aggiornata e resa **coerente** rispetto alla cache.

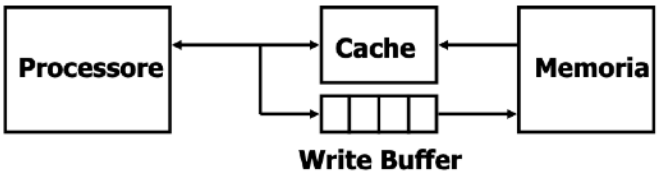
Vantaggi della politica write-through:

- Politica più semplice da realizzare
- Per essere efficace, una cache write-through deve essere dotata di un **buffer di scrittura (write buffer)** in modo da non dove accedere al livello inferiore di memoria che è più lento.
- I fallimenti in lettura sono meno costosi, infatti non richiedono mai la scrittura nel livello inferiore.

Vantaggi della politica write-back:

- I singoli blocchi possono essere scritti dal processore alla frequenza a cui la cache, e non la memoria principale, è in grado di accettarle.
- Scritture multiple all'interno dello stesso blocco richiedono una sola scrittura nella memoria principale.

Buffer di scrittura (write buffer)



- **Idea base**: Inserire un **buffer FIFO** (da 4 oppure 8 posizioni) in modo che il processore non debba aspettare il tempo di accesso del livello inferiore (memoria principale)
    - Il processore scrive i dati nella cache e nel write buffer in parallelo
    - Il controllore di memoria successivamente scriverà dati presenti nel write buffer in memoria
- **Problema principale**: Saturazione del write buffer
- Tipicamente una cache write through è sempre accoppiata ad un write buffer, però anche una cache write back può essere accoppiata ad write buffer.

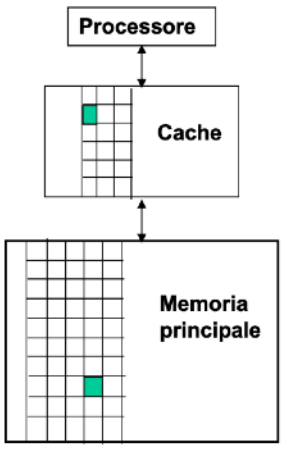
HIt & Miss: Read & Write

- **Read Hit:**
    - Lettura blocco in cache
- **Read Miss:**
    - Stallo di CPU e richiesta del blocco in memoria
    - **Scrittura** di una copia del blocco in cache
    - Ripetizione dell'operazione di **lettura** del blocco in cache

- **Write Hit:**
    - **Write-through:** Scrittura del blocco sia in cache sia in memoria
    - **Write-back:** Scrittura del dato solo in cache. La copia in memoria avverrà solo quando il blocco sarà rimpiazzato a causa di un miss (read/write)
- **Write Miss:**
    - Stallo di CPU e richiesta del blocco in memoria
    - **Scrittura** di una copia del blocco in cache
    - Ripetizione dell'operazione di **scrittura** del blocco in cache (ed eventualmente riscrittura in memoria se **write-through**)

GERARCHIA DI MEMORIA: PRESTAZIONI

- consideriamo una gerarchia di memoria su due livelli (cache e memoria principale) nel caso di **hit**



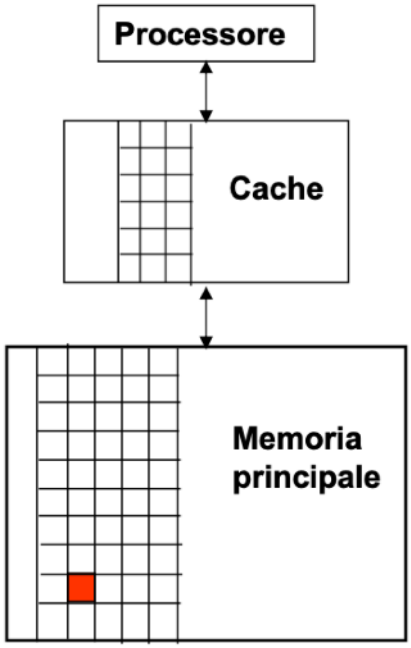
- **Hit (successo):** dati presenti in un blocco del livello superiore della gerarchia.

- **Hit Rate (frequenza dei successi)**: numero di accessi a memoria che trovano il dato nel livello superiore rispetto al numero totale di accessi a memoria

  Hit Rate = (# successi) / (# accessi a memoria)

- **Hit Time (tempo di successo)**: tempo per accedere al dato nel livello superiore della gerarchia, che comprende anche il tempo necessario per stabilire se il tentativo di accesso si risolve in un successo o in un fallimento.

- se il dato non si trova nel livello superiore, la richiesta fallisce (**miss**) → per trovare il blocco che contiene i dati richiesti, bisogna accedere al livello inferiore della gerarchia



- **Miss (fallimento)**: i dati devono essere recuperati dal livello inferiore.
- **Miss Rate (frequenza di fallimento)**: numero di accessi a memoria che falliscono rispetto al numero totale di accessi a memoria

  Miss Rate = (# fallimenti) / (# accessi a memoria)
- Chiaramente: **Hit Rate + Miss Rate = 1**
- **Miss Penalty (penalizzazione di fallimento)**: tempo necessario a sostituire un blocco nel livello superiore caricandolo da un blocco di livello inferiore più lento
- **Miss Time (tempo di fallimento):**

  Miss Time = Hit Time + Miss Penalty
- Notare che: **Hit Time << Miss Penalty**

Tempo medio di accesso alla memoria

Per definizione:

Tm = Hit Rate ∗ Hit Time + Miss Rate ∗ Miss Time

Essendo: Miss Time = Hit Time + Miss Penalty si ha:

→ Tm = Hit Rate ∗ Hit Time + Miss Rate * (Hit Time + Miss Penalty)

→ Tm = Hit Rate ∗ Hit Time + Miss Rate ∗ Hit Time + Miss Rate ∗ Miss Penalty =

→ Tm = (Hit Rate + Miss Rate) ∗ Hit Time + Miss Rate ∗ Miss Penalty =

Essendo: Hit Rate + Miss Rate = 1 si ha:

→ Tm = Hit Time + Miss Rate ∗ Miss Penalty

Esercizio sulle prestazioni di una cache

Data una gerarchia di memoria con frequenza di successi alla cache pari al 95% e con tempo di accesso alla cache pari ad un ciclo di clock di 2 ns, calcolare il tempo medio di accesso alla memoria sapendo che la penalizzazione in caso di miss è pari a 25 cicli di clock.

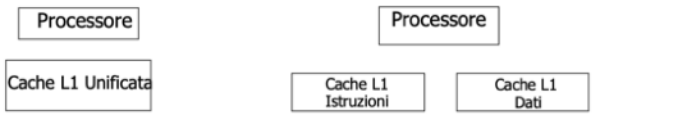
Soluzione

Dati: Hit Rate = 0,95
Hit Time = 2 ns
Miss Penalty = 50 ns

Essendo: Hit Rate + Miss Rate = 1 → Miss Rate = 1 – 0,95 = 0,05

Calcoliamo: Tm = Hit Time + Miss Rate ∗ Miss Penalty = = 2 ns + 0,05 ∗ 50 ns = = 4,5 ns

Cache unificata vs separata I$ & D$ (Harvard architecture)



- *Per sfruttare meglio il principio di località*
- *Per evitare conflitti strutturali nella pipeline*

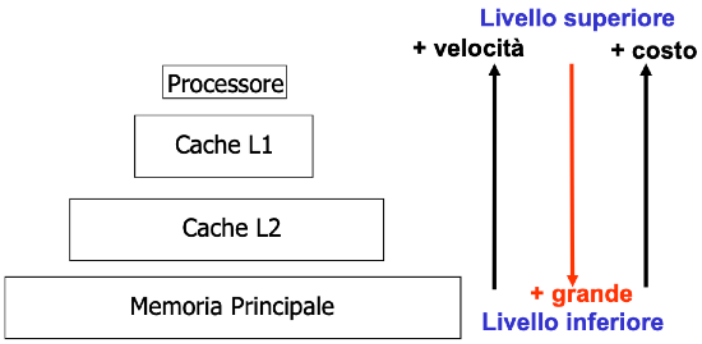
Tempo medio di accesso alla memoria per cache separate istruzioni e dati:

AMAT = % Istr. (Hit Time + I$ Miss Rate * Miss Penalty) + % Dati (Hit Time + D$ Miss Rate * Miss Penalty)

Dove tipicamente: **I$ Miss Rate << D$ Miss Rate**

Gerarchia a più livelli di cache

- Attualmente i programmi utilizzano spazi di indirizzamento così grandi e i processori hanno bisogno di memorie così veloci che una gerarchia a due livelli (cache e memoria principale) non è sufficiente per soddisfare i requisiti in termini di prestazioni.
- Si costruiscono quindi gerarchie di memoria con più di 2 livelli: ad esempio **cache di livello L1, cache di livello L2 e memoria principale.**
- Per ogni coppia di livelli si applicano le stesse considerazioni fatte per la gerarchia a due livelli (cache L1 + memoria)



Tempo medio di accesso alla memoria con cache L1 e cache L2

Dato: $Tm = Hit\ Time_{L1} + Miss\ Rate_{L1} * Miss\ Penalty_{L1}$
Analogamente si ha che:

$$\mathbf{Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} * Miss\ Penalty_{L2}}$$

$$\rightarrow Tm = Hit\ Time_{L1} + Miss\ Rate_{L1} * \mathbf{(Hit\ Time_{L2} + Miss\ Rate_{L2} * Miss\ Penalty_{L2})}$$

Esercizio con cache L1 e cache L2

Data una gerarchia di memoria con cache L1, cache L2 e memoria principale:

- frequenza di hit alla cache L1 pari al 95% e tempo di accesso alla cache L1 pari ad un ciclo di clock di 2 ns;
- frequenza di hit alla cache L2 pari al 90% e tempo di accesso alla cache L2 pari ad 5 cicli di clock;

Calcolare il tempo medio di accesso alla memoria sapendo che la penalizzazione in caso di miss in L2 è pari a 15 cicli di clock.

Soluzione

Hit Rate L1 = 0,95 → Miss Rate L1 = 0,05
Hit Rate L2 = 0,90 → Miss Rate L2 = 0,10
Hit Time L1 = 2 ns Hit Time L2 = 10 ns Miss Penalty L2 = 30 ns

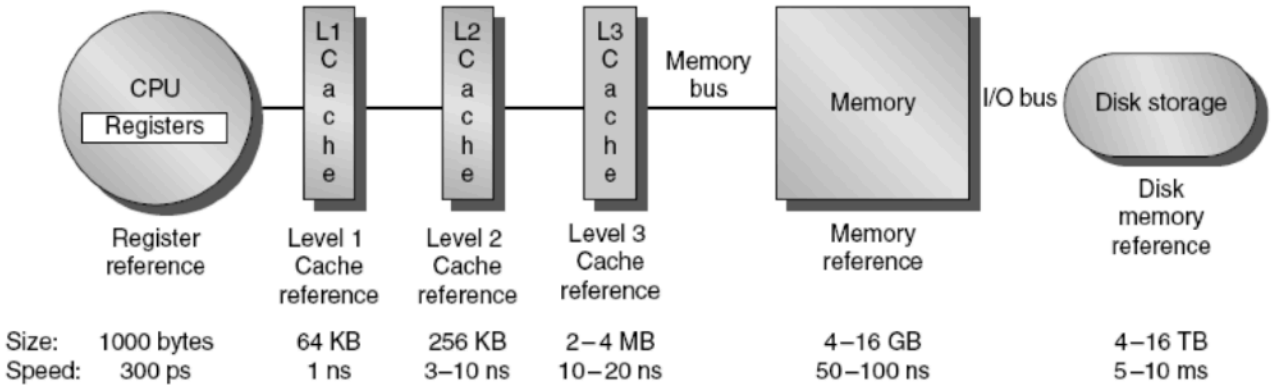
Essendo Miss Penalty L1 = Hit Time L2 + Miss Rate L2 * Miss Penalty L2
→ Tm = Hit Time L1 + Miss Rate L1 * (Hit Time L2 + Miss Rate L2 * Miss Penalty L2) = 2 ns + 0,05 (10 ns + 0,10 * 30 ns) = 2,65 ns

Tecnologie di memoria

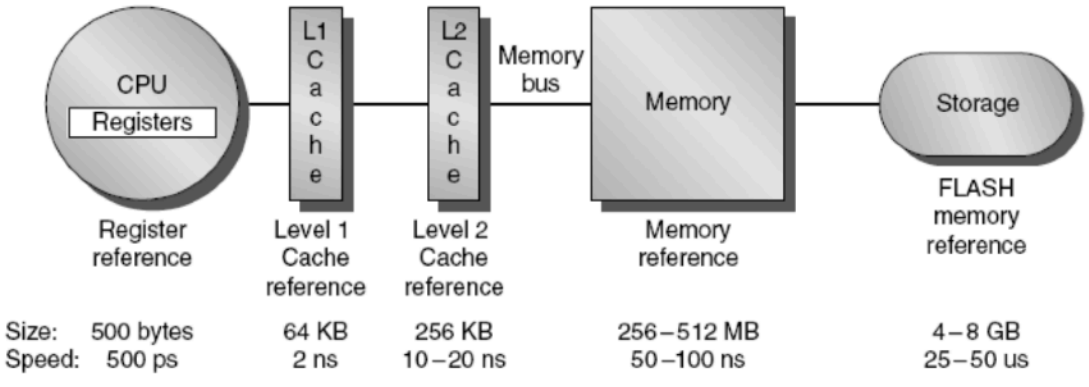
Attualmente le tipiche tecnologie utilizzate nel costruire la gerarchia di memoria nei calcolatori sono, in ordine di velocità decrescente e dimensione crescente:

Livello	Tecnologia	Tipo	Tempo di accesso	Dimensioni tipiche
Cache L1	SRAM (Static Random Access Memories)	Volatile	0,2 – 2,5 ns	16KB - 64 KB
Cache L2	SRAM (Static Random Access Memories)	Volatile	0,2 – 2,5 ns	128 KB – 8 MB
Memoria Principale	DRAM (Dynamic Random Access Memories)	Volatile	50 – 70 ns	decine Gbyte
Memoria permanente	Flash	Non volatile	0,005 – 0,05 ms	centinaia Gbyte
Memoria permanente	Dischi	Non volatile	5 – 20 ms	molti Tbyte

Livelli di gerarchia di memoria



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device