Gerarchie di Memoria
e
Cache

La memoria

I sistemi di memoria di un elaboratore possono essere suddivisi in:

- Memoria interna al processore
- Memoria principale
- Memoria secondaria

La memoria interna

- Registri interni alla CPU
  - Visibili o no al programmatore
  - Memorizzano temporaneamente dati e istruzioni
  - Dimensioni: decine di bytes
  - Tempo di accesso: qualche ns

  Nelle CPU più recenti cresce la quantità di risorse dedicate alla memoria:
  - memorie cache nella CPU:
    - 1980: processori senza cache (I386)
    - 1995: Alpha 21164 55% dei transistors
    - 2000: Merced (Intel-HP) 85% dei transistors

La Memoria principale

- Memorizza dati e istruzioni che servono per il funzionamento dell'unità centrale.
- La CPU vi accede direttamente.
- Dimensioni: centinaia di Mbytes su un personal computer, nell'ordine dei Gigabytes su supercalcolatori.
- Velocità: attorno ai 100 ns.

E' la memoria nella quale sono contenuti i programmi che la CPU esegue e i dati su cui la stessa CPU può accedere direttamente.

La Memoria Secondaria

- Dimensioni: nell'ordine dei Gbytes/Therabytes.
- Velocità: nell'ordine dei milioni di ns (millisecondi)

Tecnologie e caratteristiche

I vari tipi di memoria sono realizzati con tecnologie con valori diversi di:

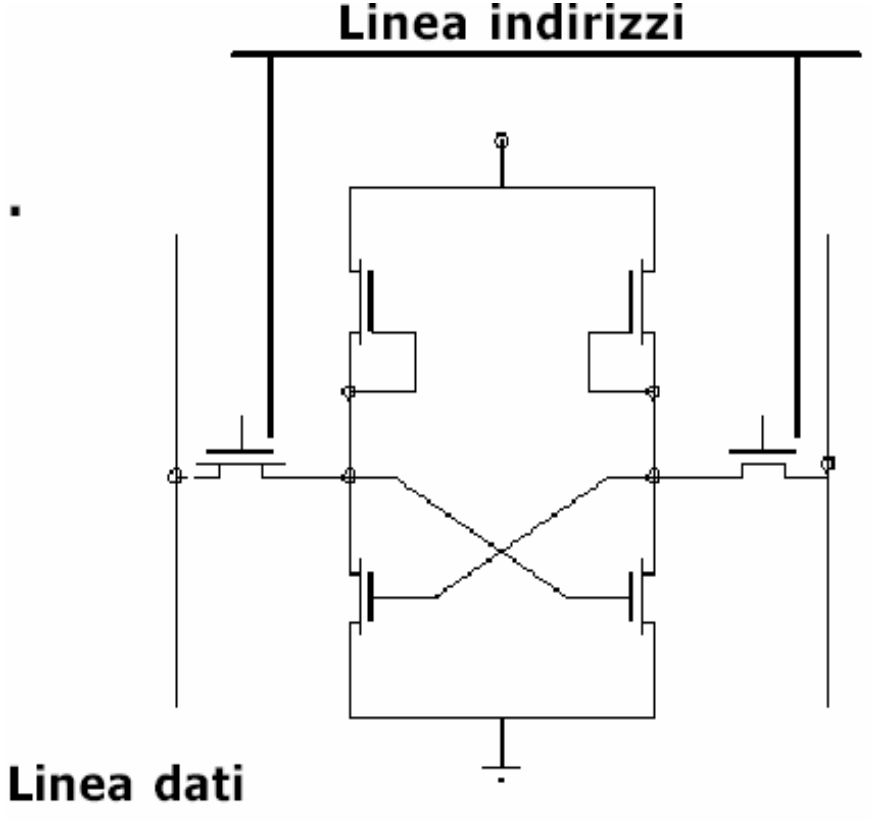
- Costo per singolo bit immagazzinato.
- Tempo di accesso (ritardo fra l'istante in cui avviene la richiesta e l'istante in cui il dato è disponibile al richiedente)
- Modo di accesso (seriale o casuale).

TECNOLOGIA DELLE MEMORIE

Memorie a semiconduttore con tecnologia VLSI (memoria principale).
Memorie magnetiche (memoria secondaria).
Memorie ottiche (memoria secondaria).

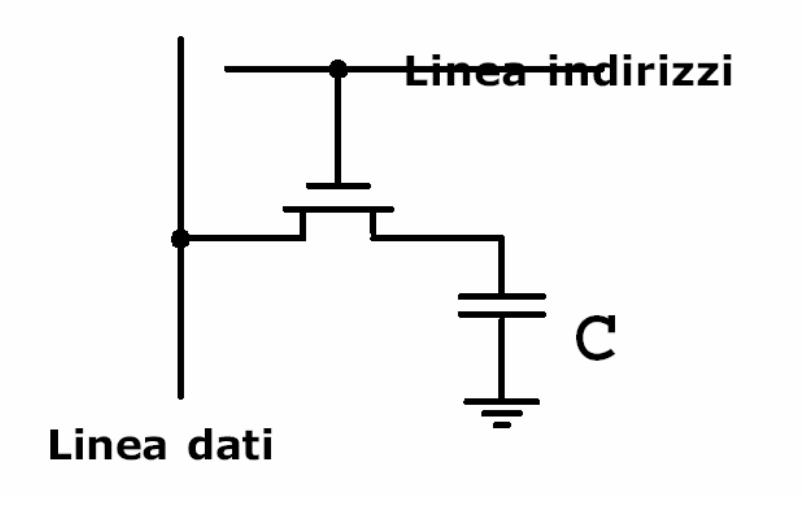
Le memorie RAM statiche

- La cella elementare è costituita da 6 transistori mos che formano un FLIP-FLOP.
- L'informazione permane stabile in presenza della tensione di alimentazione
- Tempi di accesso rapidi.
- Costi elevati.



Le memorie RAM dinamiche

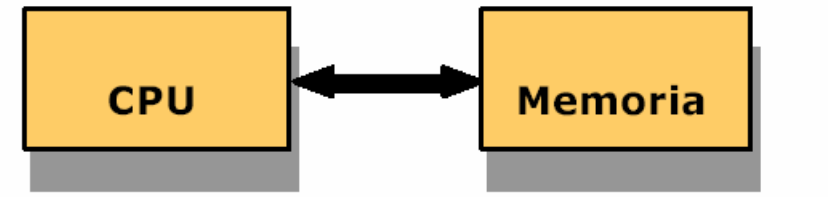
- La cella elementare è costituira da un condensatore che viene caricato (1) o scaricato (0).
- La tensione sul condensatore tende a diminuire (millisecondi) e quindi deve essere ripristinata o rinfrescata.



La semplicità della cella consente capacità molto elevate in spazi (e costi) contenuti

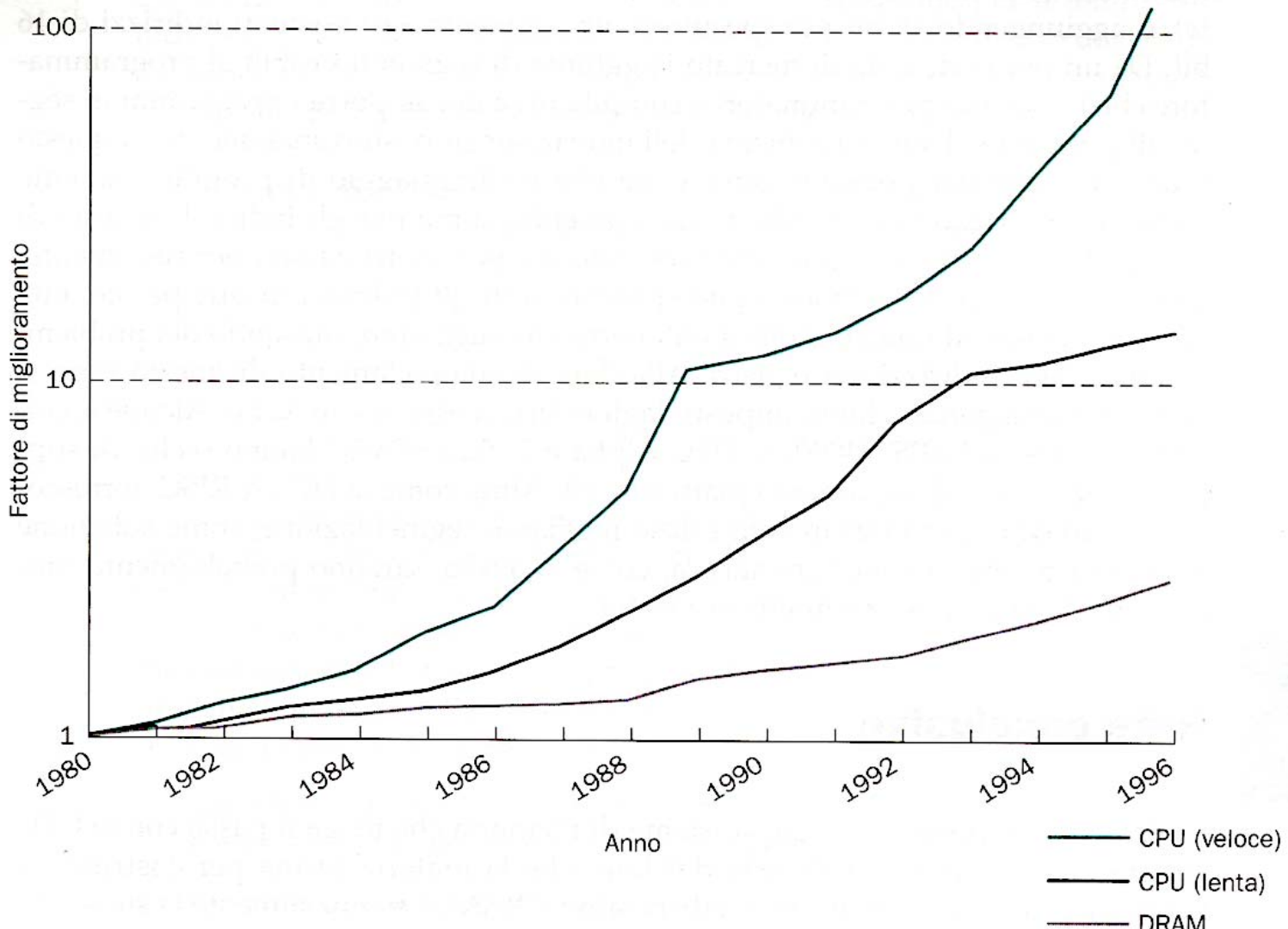
CPU-MEMORIA

Nell'architettura VonNeuman il canale di comunicazione tra la CPU e la memoria è il punto critico (collo di bottiglia) del sistema.



- La tecnologia consente di realizzare CPU sempre più veloci
- Il tempo di accesso delle memorie non cresce così rapidamente

Confronto tra la velocità delle DRAM e delle CPU



La gerarchia delle memorie

La soluzione ottimale per un sistema di memoria è:

- Costo minimo
- Capacità massima
- Tempi di accesso minimi

Soluzione approssimata: GERARCHIA

Tecnologie diverse possono soddisfare al meglio ciascuno dei requisiti.

Una gerarchia cerca di ottimizzare globalmente i parametri.

Esempio di gerarchia

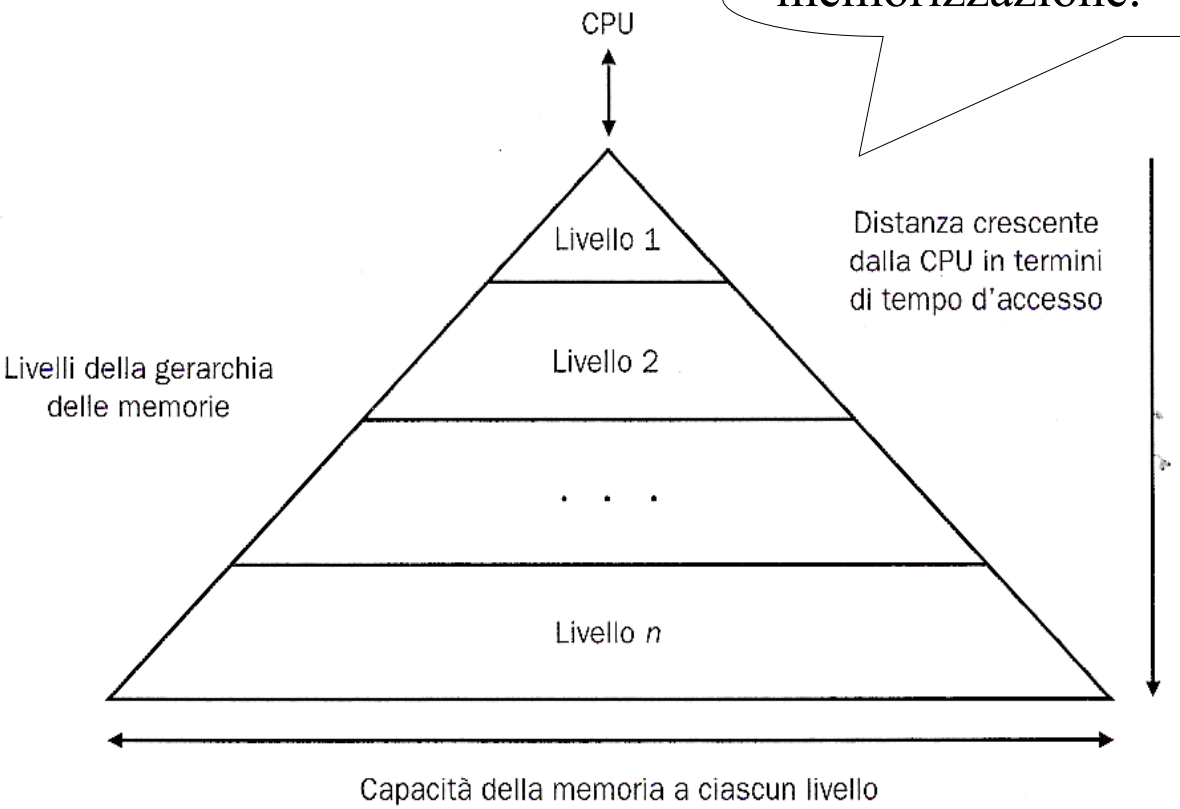
Il sistema di memoria di uno studente ha una struttura gerarchica:

- La propria memoria.
- La propria scrivania.
- Lo scaffale di casa.
- La libreria o la biblioteca di Facoltà.
- Depositi casa editrice.

La gestione del sistema di memoria globale di uno studente è molto complessa e richiede la conoscenza preventiva delle attività che si svolgeranno.

Struttura fondamentale della gerarchia della memoria

Al crescere della distanza dalla CPU cresce anche il costo e la capacità di memorizzazione.



La gerarchia può consistere di più livelli, ma in ogni istante di tempo i dati sono copiati solamente tra ciascuna coppia di livelli adiacenti, per cui ci si può concentrare su due soli livelli.

Principio di località

Un sistema di memoria gerarchico può essere reso efficiente se la modalità di accesso ai dati ha caratteristiche prevedibili.

Il principio di località si basa sul fatto che in un dato istante i programmi fanno accesso ad una porzione relativamente piccola del loro spazio di indirizzamento.

Si distinguono 2 tipi diversi di località:

- *Località temporale*: è probabile che un oggetto a cui si è fatto riferimento venga nuovamente richiesto in tempi brevi. Es: cicli in un programma, le istruzioni sono richieste ripetutamente
- *Località spaziale*: è probabile che gli oggetti che si trovano vicini ad un oggetto a cui si è fatto riferimento vengano richiesti in tempi brevi .Es: esecuzione sequenziale in un programma.

Criteri di Gestione

- I dati utilizzati più spesso vanno posti in memorie facilmente accessibili.
- I dati utilizzati più raramente sono posti in memorie con tempi di accesso elevato.
- Allocazione dinamica per utilizzare gli spazi disponibili con la massima efficienza.
- Spostamento automatico dei dati tra i livelli.
- Canali di comunicazione veloci fra i livelli.

La politica di gestione mira ad offrire una memoria che abbia:

- i tempi di accesso della più veloce,
- le dimensioni della maggiore,
- i costi della più economica.

Hit & Miss

Si ha un **hit** (successo nell'accesso) quando i dati richiesti dal livello superiore (ad es. il processore) compaiono in qualche blocco nel livello inferiore.

Si ha un **miss** (fallimento nell'accesso) se il dato non è presente nel livello immediatamente inferiore ed occorre accedere al livello più distante.

Uno dei parametri principali per la valutazione delle prestazioni di una gerarchia di memoria è l'**hit ratio** (tasso di hit), ovvero la frazione degli accessi in ram che si sono risolti al livello più vicino. Il **miss ratio** è definito come *1-hit ratio* ed indica la percentuale degli accessi che non sono stati soddisfatti dal livello più vicino nella gerarchia.

Tempo di hit: tempo necessario a prelevare il dato dal livello più vicino, comprendendo il tempo necessario a determinare se l'accesso è un hit o un miss.

Tempo di miss: tempo necessario a sostituire un dato nel livello più vicino con il blocco corrispondente nel livello inferiore, più il tempo necessario per consegnare il dato al livello richiedente (ad es. il processore).

La cache

Il termine cache è stato usato per la prima volta per indicare il livello della gerarchia tra la CPU e la memoria principale, ma è oggigiorno utilizzato per indicare qualsiasi tipo di gestione della memoria che tragga vantaggio dalla località degli accessi.

Il meccanismo è semplice: Il processore interagisce direttamente SOLO con la cache. Quando il processore richiede una parola non presente nella cache (miss) la parola viene trasferita dalla memoria nella cache.

Occorre definire dei meccanismi per:

1. Conoscere se un dato è nella cache
2. Nel caso in cui il dato sia presente, conoscere la sua posizione ed accedervi.

Queste due operazioni devono essere eseguite nel minor tempo possibile, poiché la velocità con cui si riesce ad accedere ai dati nella cache influisce drasticamente sulle prestazioni dell'intero sistema di memoria.

Cache a corrispondenza diretta

A ciascuna parola di memoria corrisponde esattamente una locazione della cache.

posizione nella cache=

(indirizzo della parola) mod (numero di posizioni nella cache)

Se il numero di blocchi nella cache è una potenza di 2, la posizione corrispondente della parola in cache è data dai $\log_2$(numero elementi nella cache) bits meno significativi dell'indirizzo in memoria principale.

Esempio:

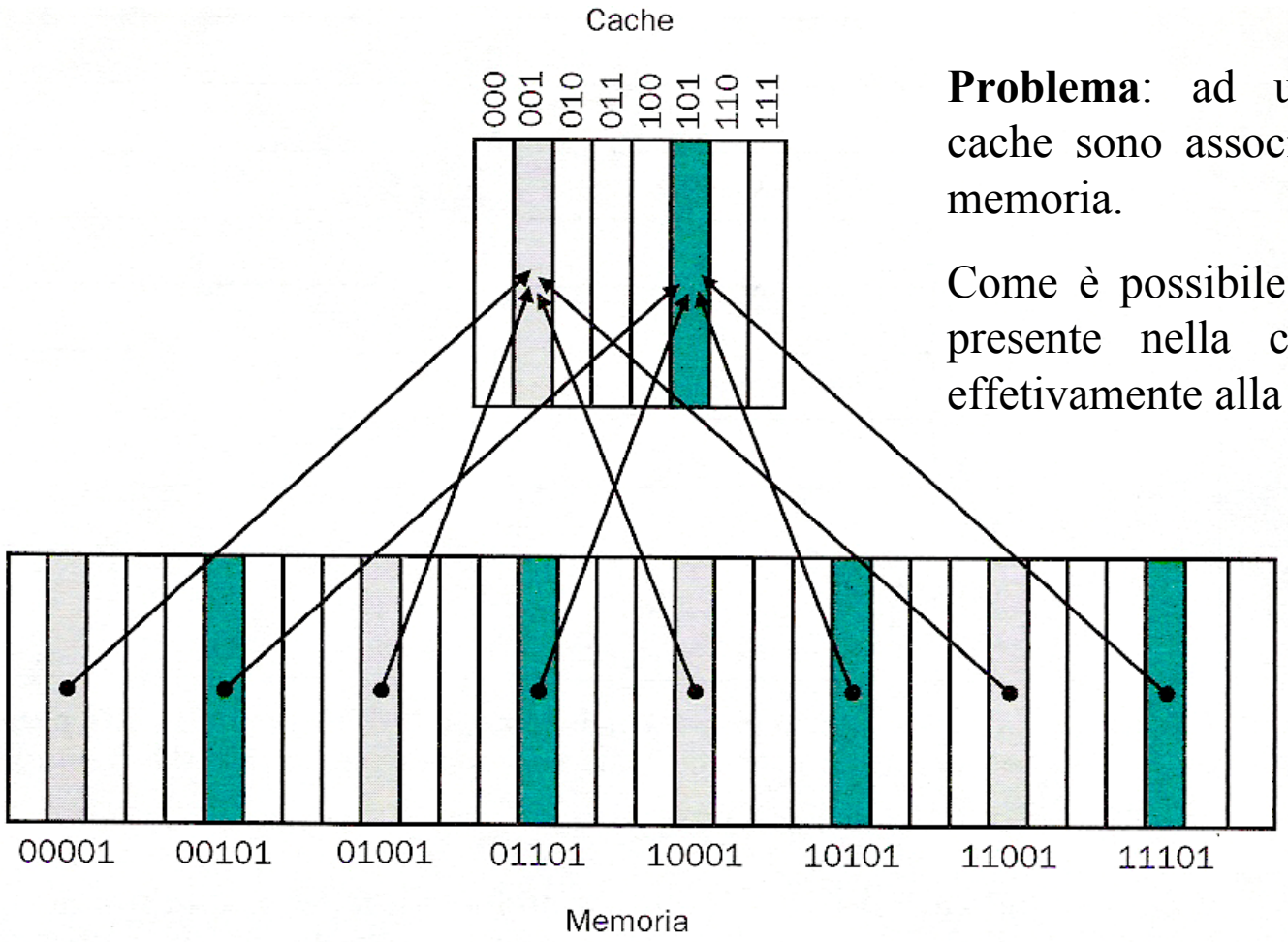
Numero di elementi nella cache: 8,

Bit per indirizzare una locazione della cache: $\log_2(8)=3$

Indirizzo della parola di memoria= 0111 01010 0010 0100

Posizione in cache: 100 (4)

Corrispondenza tra indirizzi in memoria e in cache



Problema: ad una posizione in cache sono associate più parole di memoria.

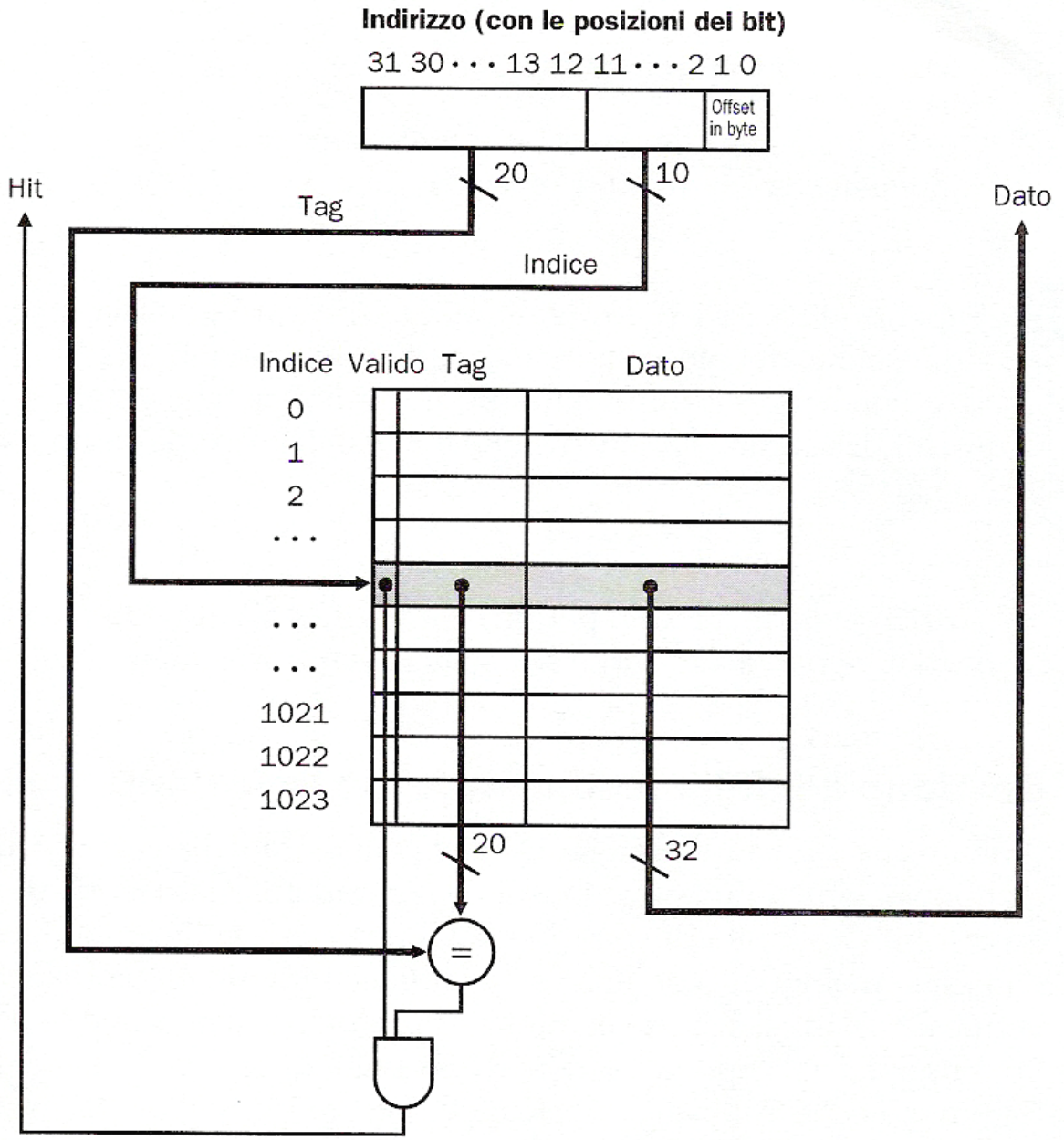
Come è possibile sapere se il dato presente nella cache corrisponde effetivamente alla parola richiesta?

I tag

Oltre ai dati richiesti, la cache deve permettere la memorizzazione di una serie di informazioni aggiuntive (dette tags) che permettano la corretta gestione della gerarchia di memorie.

Per associare univocamente una parola di memoria ad una posizione in una cache direct mapped è sufficiente memorizzare, assieme al dato, anche la parte più significativa dell'indirizzo, quella che non corrisponde ai bit utilizzati come indice nella cache.

Per distinguere le posizioni della cache contenenti dati validi (es.all'avvio del processore la cache è vuota ed il suo contenuto non è significativo), il metodo più comune è aggiungere un *bit di validità*.


Indirizzo (con le posizioni dei bit)
31 30 · · · 13 12 11 · · · 2 1 0
Offset in byte
20
10
Hit
Tag
Indice
Dato
Indice Valido Tag
Dato
0
1
2
· · ·
· · ·
· · ·
1021
1022
1023
20
32
=

Calcolo della dimensione totale di una cache a corrispondenza dirtetta

Assumendo che la cache abbia una capacità di 2^n parole, e blocchi di una parola (4 bytes) il campo tag ha dimensione: 32 – (n+2) bit, perché 2 bit sono usati per l'offset del byte.

Il numero totale di bit in una cache a corrispondenza diretta contenente 2^n blocchi è maggiore di:

2^n x dim.blocco= 2^n x 32 bits,

ed è invece pari a:

2^n x (dim.blocco+dim.tag+dim.bit validità)=

2^n x (32+32-(n+2)+1)= 2^n x (63-n) bits

Gestione delle operazioni di scrittura

Le operazioni di scrittura devono essere gestite in maniera più complessa rispetto alle letture. Se le scritture alterassero solo lo stato della cache e non venissero propagate fino alla memoria principale si creerebbero situazioni di incoerenza.

Il modo più semplice per assicurare la coerenza di cache e memoria è di scrivere sempre i dati sia nella memoria che nella cache. Tale tecnica è nota come *write-through*.

Write through

1. Accedere alla cache usando come indice i bit meno significativi dell'indirizzo.
2. Scrivere i rimanenti bit dell'indirizzo nel campo tag, scrivere la parola di dato e forzare ad uno il bit di validità
3. Scrivere la parola nella memoria principale utilizzando l'indirizzo completo.

Svantaggi: tutte le operazioni di scrittura implicano scrittura nella memoria principale, durante le quali il processore rimane in fase di stallo.

Possibile soluzione: *buffer delle scritture*. Limiti: processore è in grado di generare richieste di scritture ad un tasso maggiore di quello con cui la memoria porta a termine le scritture (crescita infinita della coda).

Nota: per scrivere non è necessario effettuare accessi in lettura alla memoria principale.

Write back

Una tecnica alternativa che offre prestazioni in genere superiori è il write-back.

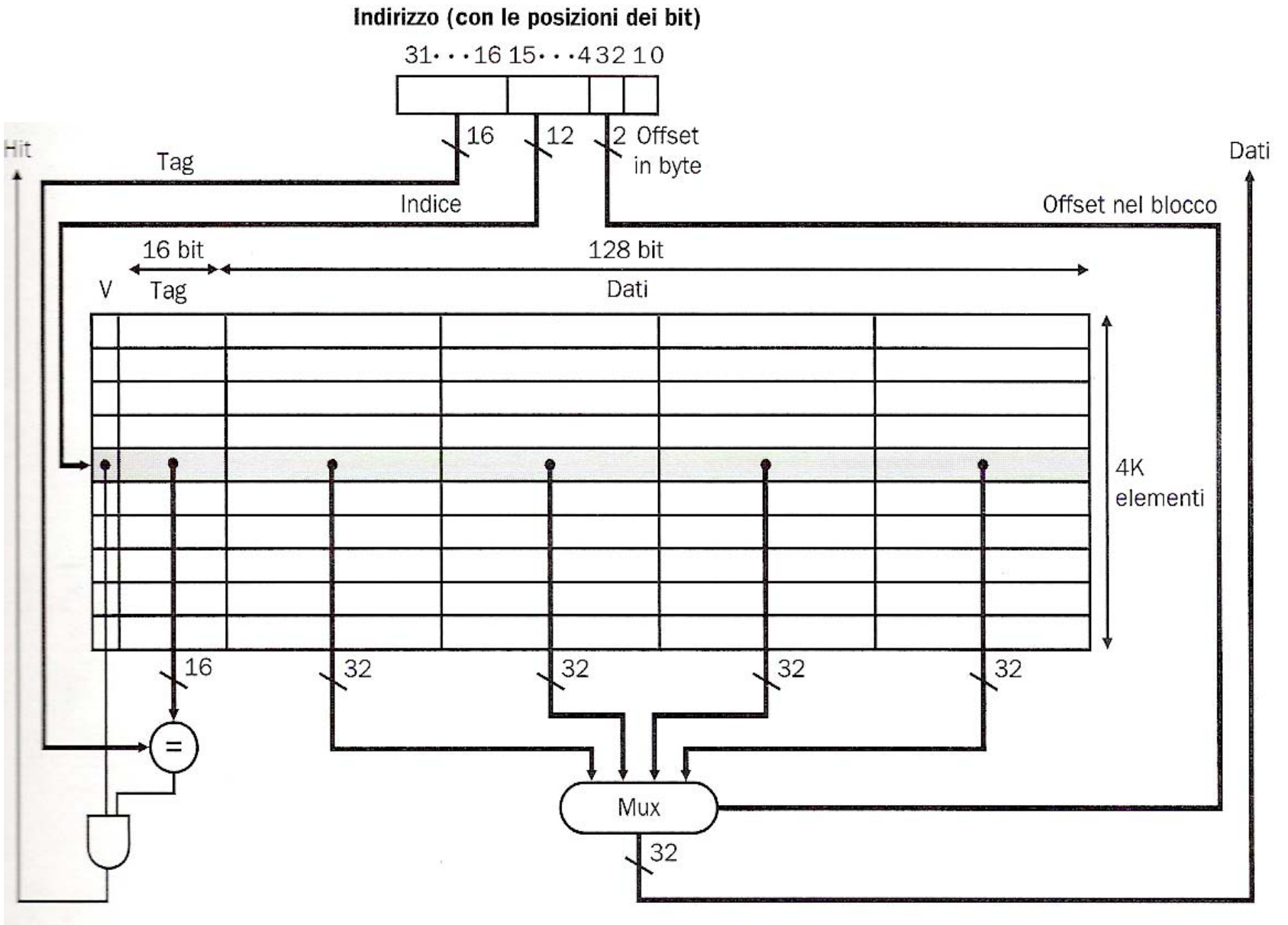
I blocchi sono effettivamente scritti in memoria principale solo quando è necessario rimpiazzarli nella cache.

Tale soluzione ha naturalmente una maggiore complessità implementativa rispetto al write-through.

Sfruttare la località spaziale

Per trarre vantaggio del principio di località spaziale, è opportuno prevedere che i blocchi della cache siano più ampi di una sola parola. In caso di miss saranno così caricate diverse parole adiacenti che hanno un'elevata probabilità di essere richieste nel prossimo futuro.

Ogni linea di cache memorizza più parole e si usa il campo "Offset del blocco" per selezionare la parola richiesta tra quelle presenti nella linea.


Indirizzo (con le posizioni dei bit)
31 · · · 16 15 · · · 4 3 2 1 0
16
12
2 Offset in byte
Hit
Tag
Indice
Offset nel blocco
Dati
16 bit
128 bit
V
Tag
Dati
4K elementi
16
32
32
32
32
=
Mux
32
Dati

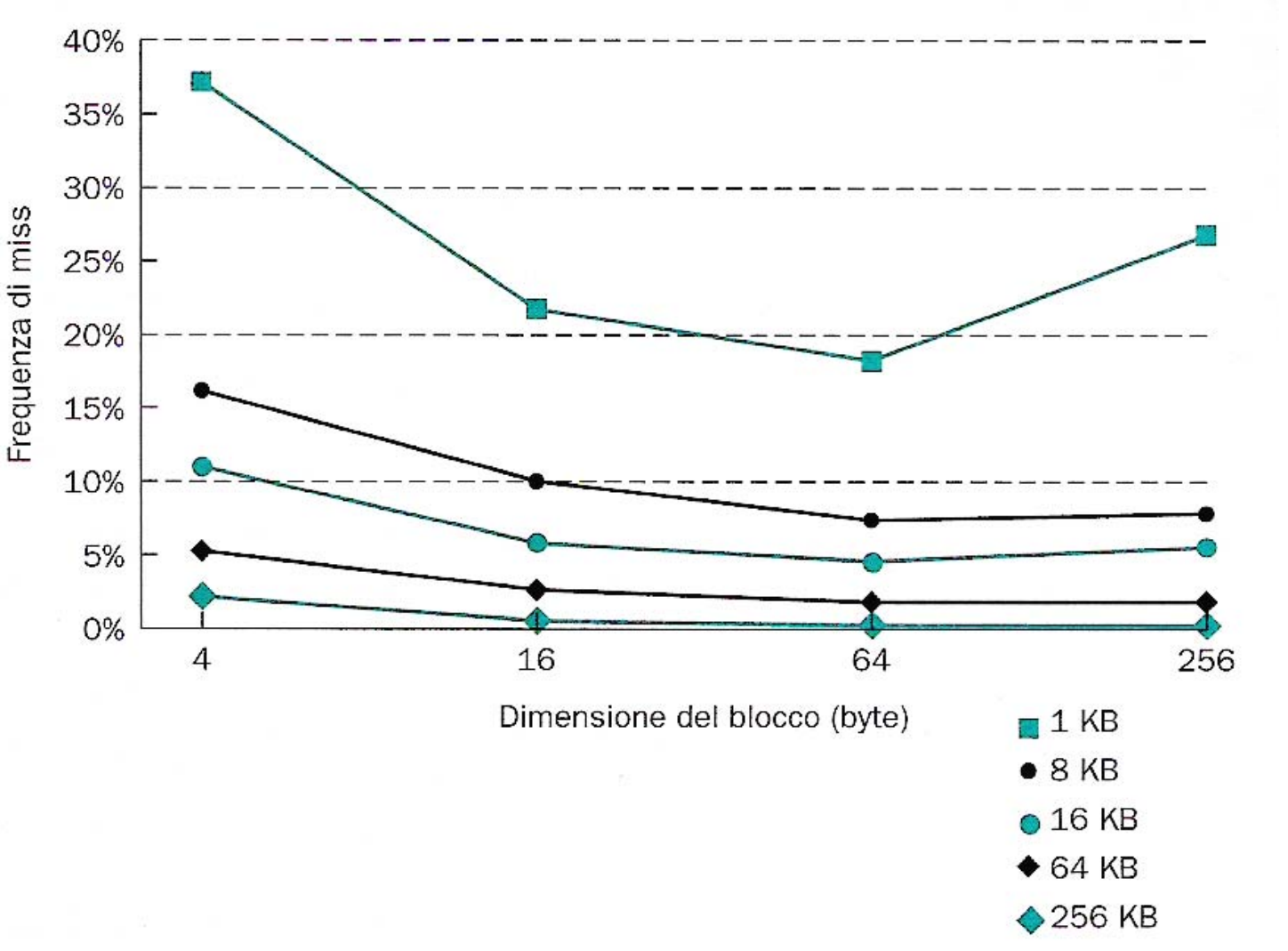
Gestione delle scritture

Se il blocco contiene più di una parola, non è possibile scrivere solo il dato e il tag: siano X ed Y 2 indirizzi associati alla stesso blocco C (di 4 parole) in cache, e si assuma che C contenga inizialmente Y e le tre parole adiacenti. Se si scrivesse semplicemente X e il suo tag, il blocco in cache conterrebbe una parola di X e tre parole di Y!

All'atto della scrittura si confronta il tag in cache con quello associato all'indirizzo in cui scrivere. Se sono uguali è sufficiente aggiornare la parola in cache. Altrimenti è necessario caricare l'intero blocco in cache e quindi riscrivere la parola che ha causato il miss.

Nota: a differenza del caso con blocchi di una parola, i miss nelle scritture richiedono di leggere la memoria.

Dipendenza della frequenza di miss dalla dimensione dei blocchi



Fully/set associative cache

Nelle cache **direct mapped** a ciascun blocco della memoria corrisponde una specifica locazione nella cache.

Pro: semplicità implementativa.

Contro: possibile sotto-utilizzazione della cache.

In una cache **fully associative** ogni blocco può essere collocato in qualsiasi locazione della cache.

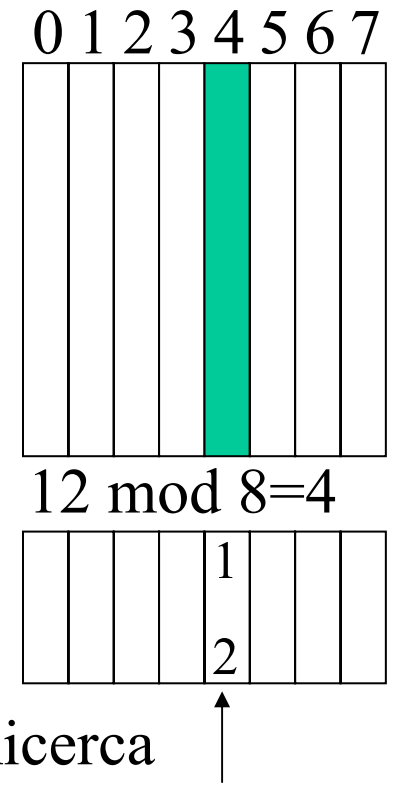
Pro: massima utilizzazione della cache.

Contro: per ricercare un blocco nella cache è necessario cercarlo in tutte le celle. Per velocizzare la ricerca è necessario effettuarla in parallelo, associando un comparatore a ciascuna posizione della cache. COSTO MOLTO ELEVATO.

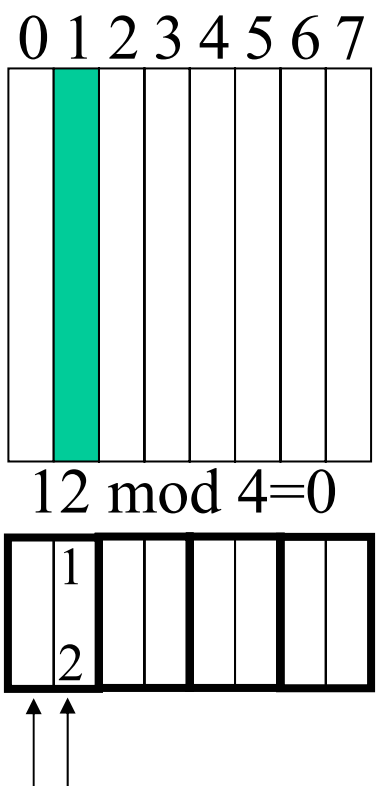
Un buon compromesso è costituito dalle cache **set-associative**: ciascun blocco di memoria ha a disposizione un numero fisso *n* (>=2) di locazioni in cache. Tali cache sono dette anche **n-way set associative**.

Posizione di un blocco di memoria nel caso di cache:

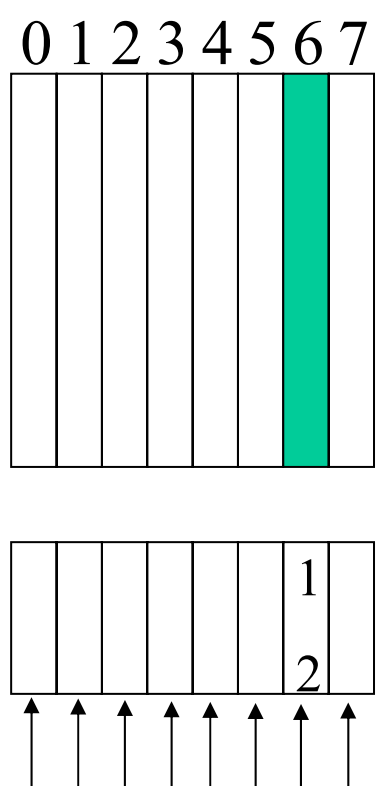
A corrispondenza diretta



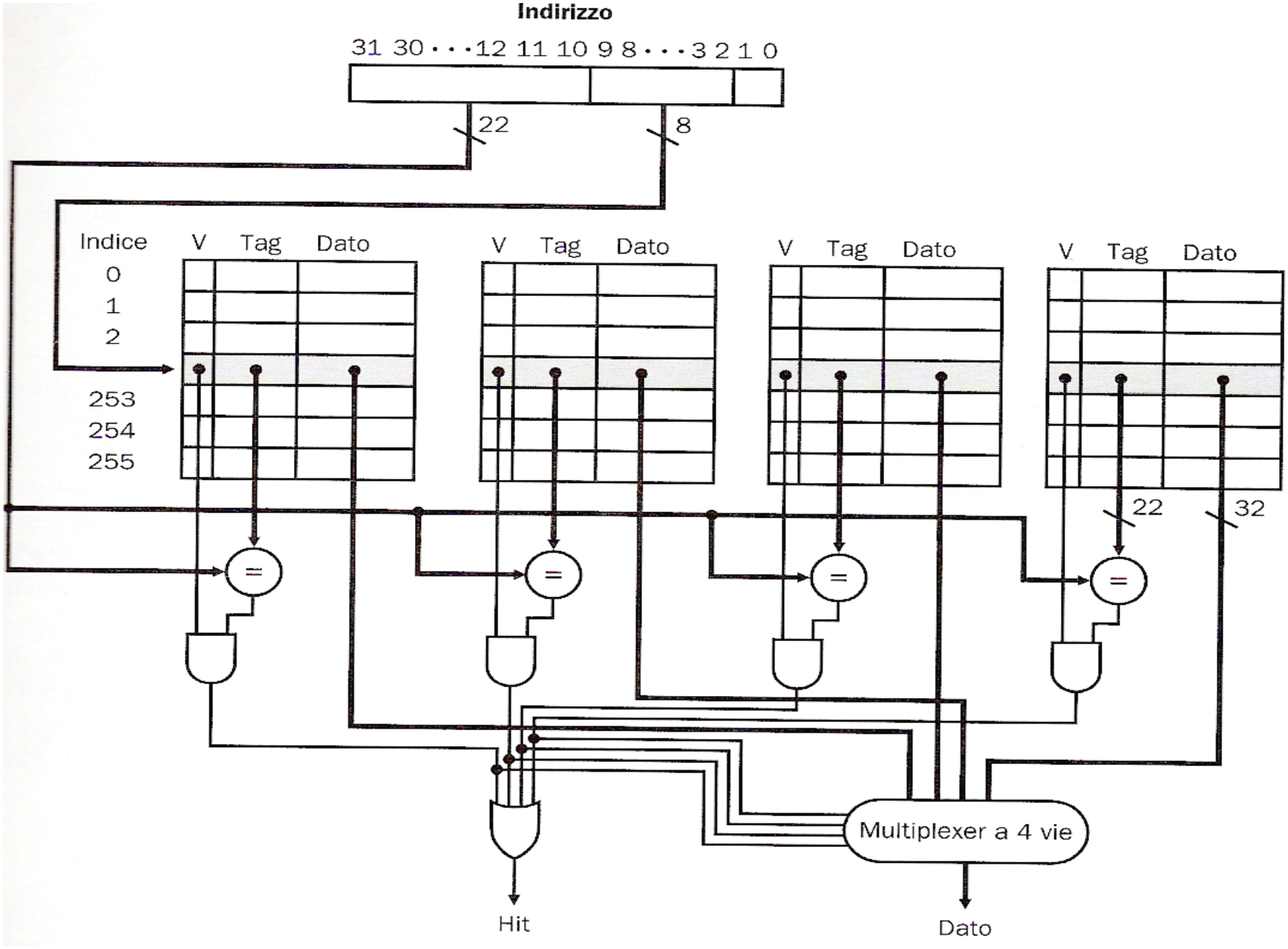
Set-associative



Fully associative



Ricerca di un blocco all'interno di una cache set-associative


Indirizzo
31 30 · · · 12 11 10 9 8 · · · 3 2 1 0
22
8
Indice
V
Tag
Dato
0
1
2
253
254
255
=
22
32
Multiplexer a 4 vie
Hit
Dato

Scelta del blocco da sostituire

In caso di miss in lettura, nelle cache set-associative occorre scegliere quale elemento dell'insieme rimpiazzare.

La tecnica usata comunemente è detta LRU (Least recently used), che prevede di sostituire l'elemento usato meno di recente. Nel caso di caches set-associative a 2 vie è possibile implementare tale politica semplicemente memorizzando un bit di informazione aggiuntiva che indichi l'elemento usato più (o meno) recentemente.