



La Sapienza

Università degli Studi di Roma

Dipartimento di Informatica e Sistemistica

CALCOLATORI ELETTRONICI

Gerarchie di memoria

Emiliano Trevisani

trevisani@dis.uniroma1.it

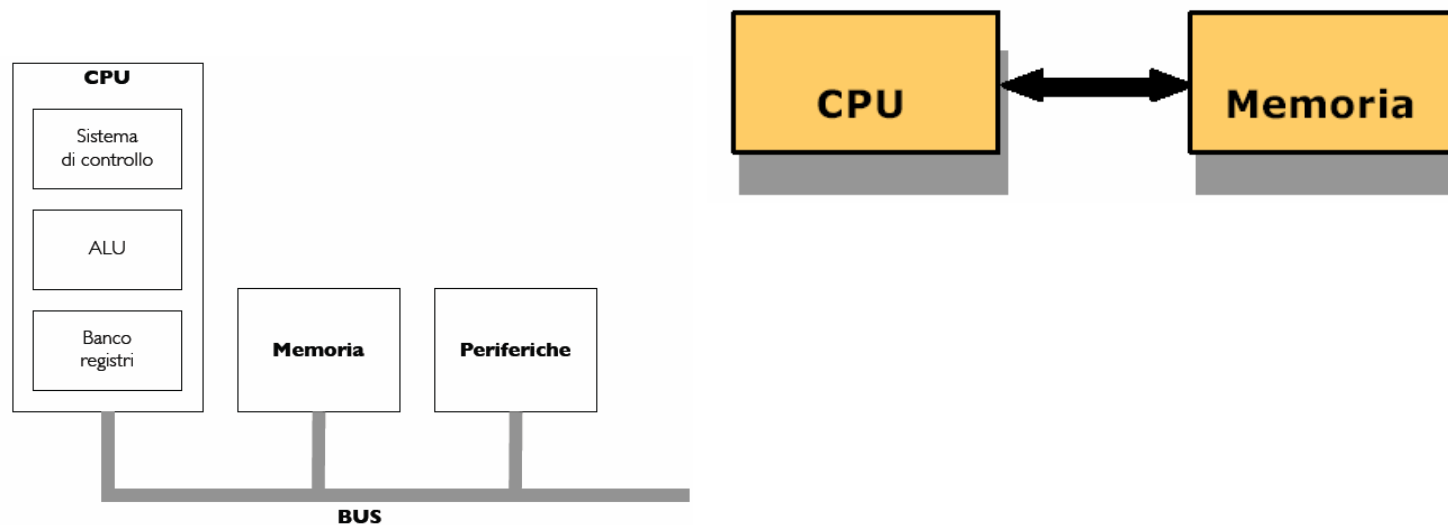
A.A. 2007/2008

Gerarchie di memoria



□ Con riferimento al modello di Von Neumann già discusso consideriamo la comunicazione **CPU ↔ Memoria**

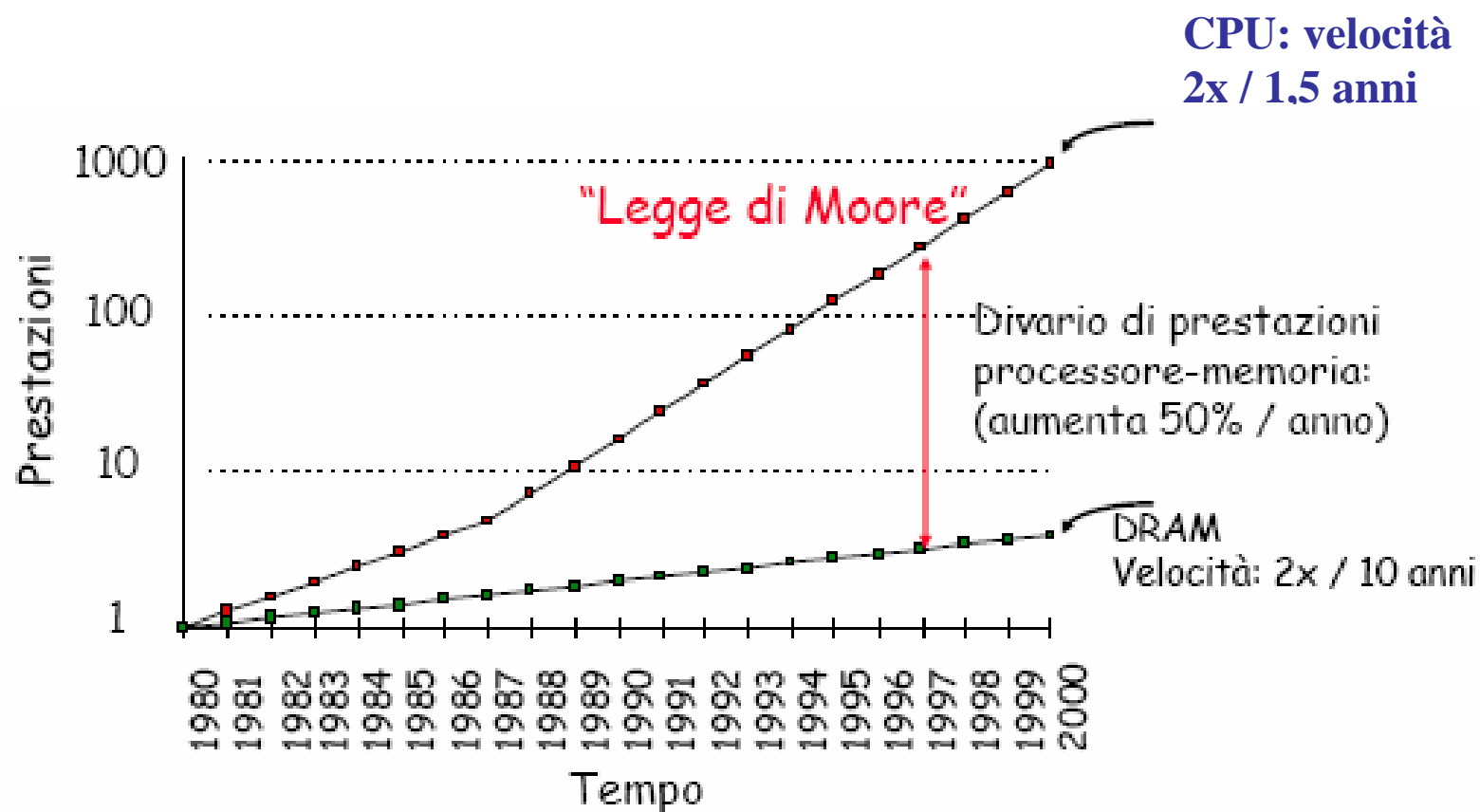
- Il canale di comunicazione tra la CPU e la memoria è il punto critico (collo di bottiglia) del sistema a causa della differenza di velocità dei due sistemi
 - La tecnologia consente di realizzare CPU sempre più veloci
 - Il tempo di accesso delle memorie **non cresce così rapidamente**



Gerarchie di memoria



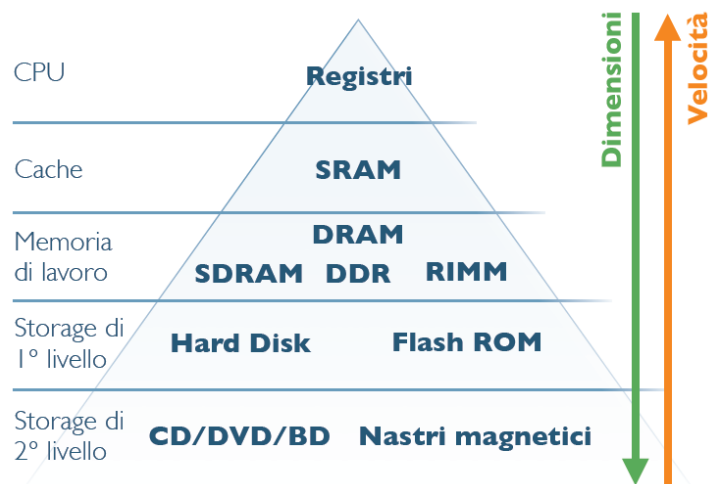
- Per avere un'idea del **divario di prestazioni CPU-Memoria**:



Gerarchie di memoria



- ❑ Un sistema di memoria **ideale** dovrebbe soddisfare i seguenti requisiti:
 - Costo basso per bit
 - Capacità elevata
 - Tempi di accesso ridotti [riferimento: i tempi delle CPU]
- ❑ Ma, con riferimento ai sistemi di memoria **reali**
 - Le memorie con capacità elevata [costo per bit ridotto] sono lente
 - Le memorie veloci hanno una capacità ridotta [costo per bit ridotto]





□ Idea \Rightarrow Soluzione approssimata: concetto di **GERARCHIA DI MEMORIE**

- Tenuto conto delle differenze di costo e di velocità, diventa conveniente costruire una gerarchia di livelli di memoria, con la memoria più veloce posta più vicino al processore e quella più lenta, meno costosa, posta più distante
- L'obiettivo è quello di 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
- Tecnologie diverse possono soddisfare al meglio ciascuno dei requisiti.
- Una gerarchia cerca di ottimizzare **globalmente** i parametri.
- Offre l'illusione di una memoria capiente, veloce ed economica:

Gerarchie di memoria



- Un sistema di memoria gerarchico può essere reso **efficiente** se la modalità di accesso ai dati ha caratteristiche prevedibili
 - Le tecniche che realizzano gerarchie di memoria sono tutte basate sul **principio di località**, verificato analizzando l'esecuzione di un gran numero di programmi particolarmente significativi:
 - **Località temporale**: quando si fa riferimento a un elemento di memoria, c'è la tendenza a far riferimento allo **stesso** elemento entro breve (un tipico esempio è costituito dal riutilizzo di istruzioni e dati contenuti nei cicli dei programmi);
 - **Località spaziale**: quando si fa riferimento 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 (tipicamente, eseguita un'istruzione si tende ad eseguire quella immediatamente successiva; quando si accede a dati organizzati in vettori o matrici, l'accesso a un dato è seguito dall'accesso al dato immediatamente successivo, etc.).
 - 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**

Gerarchie di memoria



□ La memoria di un calcolatore è implementata quindi come una **gerarchia di memoria**

- Differenti tempi di accesso e di costo corrispondenti ai diversi livelli di memoria
- Lo scambio di informazione avviene solo tra due livelli adiacenti. L'unità di informazione che viene scambiata si dice **blocco**.

Aumenta il tempo di accesso

Aumenta la capacità di memorizzazione

Diminuisce il costo per bit

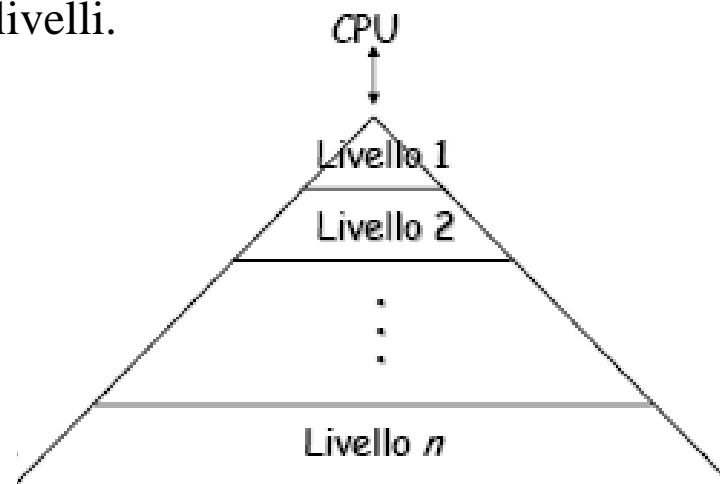


Gerarchie di memoria



□ 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.

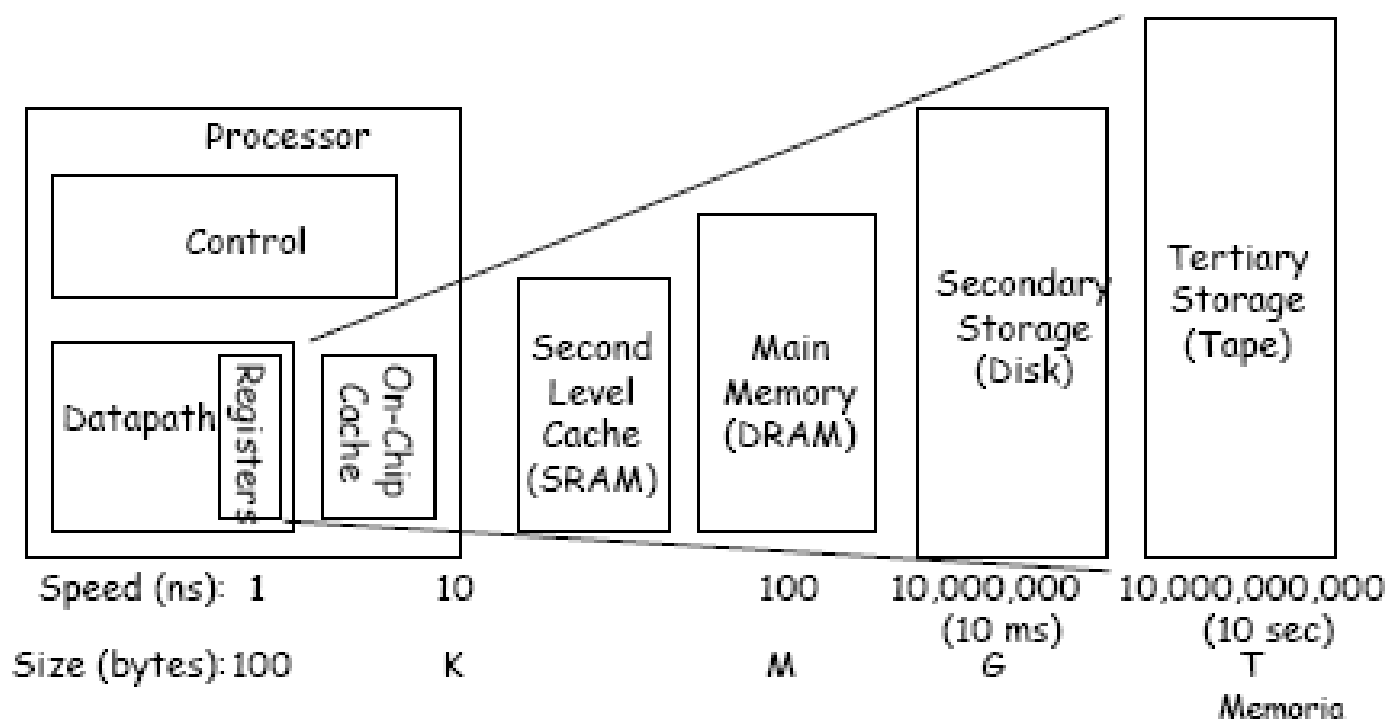


Gerarchie di memoria



□ Obiettivi della gerarchia di memoria:

- Fornire all'utente una quantità di memoria pari a quella disponibile nella tecnologia più economica
- Fornire una velocità di accesso pari a quella garantita dalla tecnologia più veloce

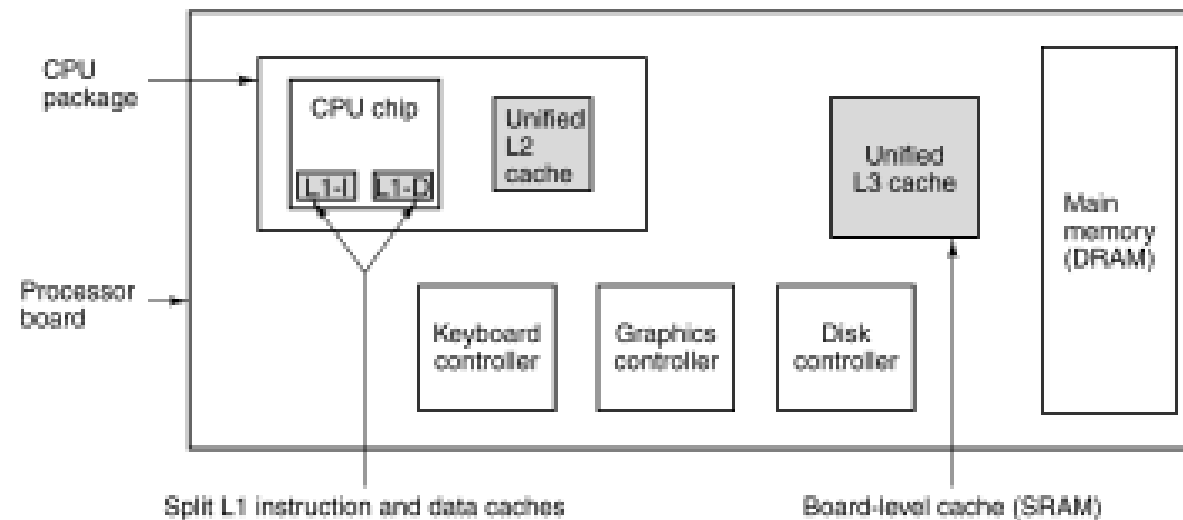


Gerarchie di memoria



□ Il concetto di gerarchia può essere applicato iterativamente: **gerarchie di cache**

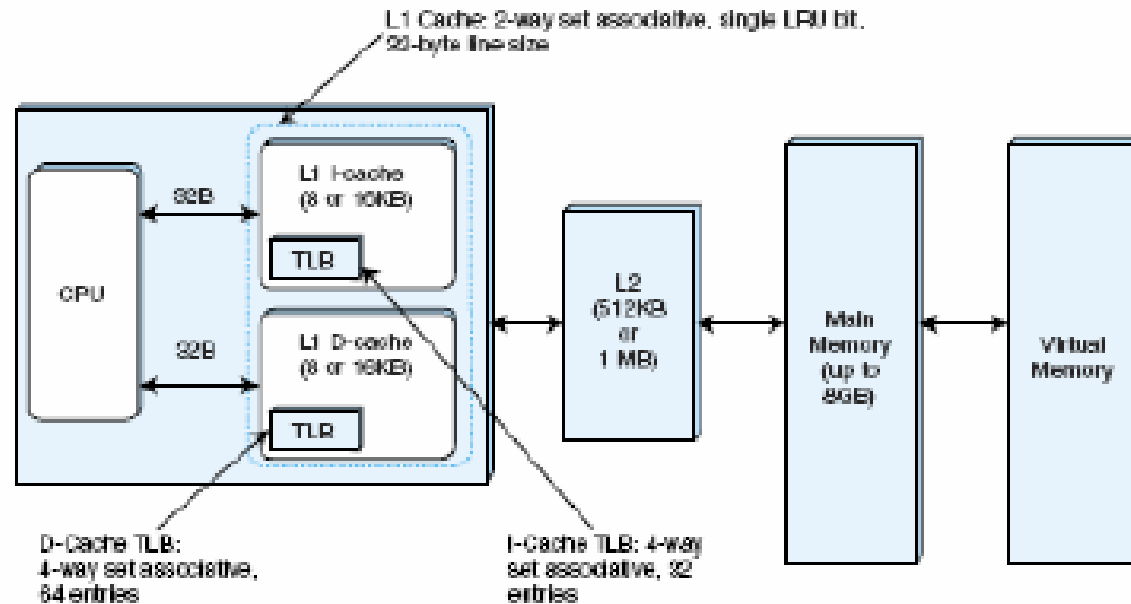
- 2 o 3 livelli:
 - La cache di livello 1 (dialogo **diretto** con la CPU, **velocità massima**) in genere è separata per dati e istruzioni
 - Concetto di **Cache inclusive**: ciascuna cache contiene sempre quella del livello superiore



Gerarchie di memoria



□ Esempio: CPU Intel Pentium®



Il concetto di **memoria virtuale** si riferisce ad una architettura di sistema capace di simulare uno spazio di memoria centrale maggiore di quello fisicamente presente; questo risultato si raggiunge utilizzando spazio di memoria secondaria su altri dispositivi, di solito le unità a disco

TLB: Translation Lookaside Buffer (TLB) è un'ulteriore cache della CPU che contiene un frazione della *page table*, una struttura dati che traduce indirizzi di memoria virtuale in indirizzi fisici

Gerarchie di memoria



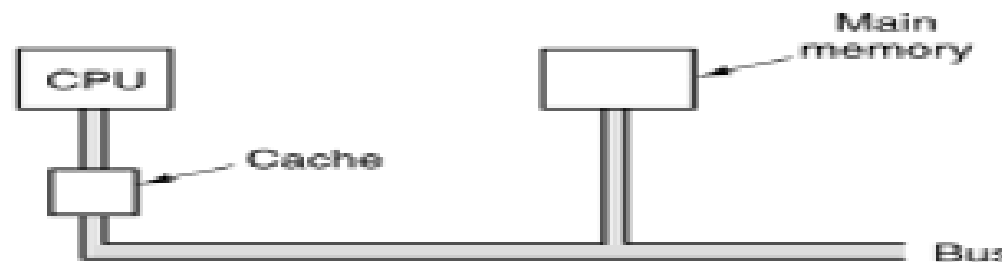
□ Hit & Miss

- Si ottiene un **hit** (successo nell'accesso) quando i dati richiesti dal livello superiore (ad es. il processore) compaiono in qualche blocco del livello inferiore.
- Si ottiene un **miss** (fallimento nell'accesso) se il dato non è presente nel livello immediatamente inferiore e la gerarchia dovrà recuperarlo dai livelli inferiori.
- 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 (cache).
- Il **miss ratio** ($= 1 - \text{hit ratio}$) indica la frazione degli accessi che non sono stati risolti al 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 complessivo necessario a sostituire un dato nel livello più vicino con il blocco corrispondente del livello inferiore ed a consegnare il dato al livello richiedente (ad es. il processore).

Memoria cache



- ❑ La cache opera alla velocità del processore, e quindi nasconde la “lentezza della memoria
- ❑ Scopo della cache: **disaccoppiare** le velocità di processore e RAM
- ❑ Le cache sfruttano il principio di **località dei riferimenti a memoria**:
 - contengono le ultime porzioni di memoria acceduta: se il processore richiede l’accesso ad una di esse evita un accesso alla memoria [località temporale]
 - ogni “riga” di cache in genere contiene blocchi contigui di parole di memoria [località spaziale]



Memoria cache



□ Strategia di utilizzo di una cache:

- La prima volta che il processore richiede dei dati si ha un cache miss
 - I dati vengono caricati dalla memoria principale e vengono copiati anche nella cache
 - Si legge un blocco di parole contigue
- Le volte successive, quando il processore richiede l'accesso ad una cella di memoria
 - Se il dato è presente in un blocco contenuto nella cache, la richiesta ha successo ed il dato viene passato direttamente al processore (**cache hit**)
 - Altrimenti la richiesta fallisce ed il blocco contenente il dato viene anche caricato nella cache e passato al processore (**cache miss**)
- Obiettivo: **aumentare quanto più possibile il tasso di cache hit**

Memoria cache



□ Strategia di utilizzo di una cache:

- Tempo medio di accesso a memoria (AMAT - Average Memory Access Time)

- hit rate [h]: $h = \frac{\#hit}{\#accessi}$

$$AMAT = hc + (1 - h)m$$

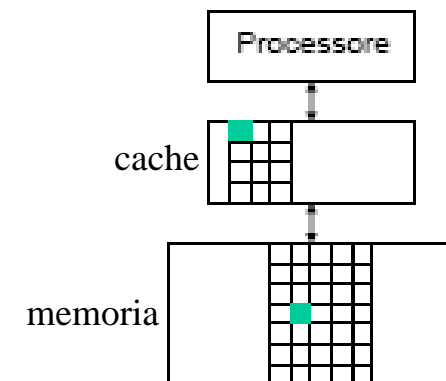
- miss rate: $miss_rate = 1 - h$

- hit time [c]: tempo di accesso alla cache

- miss penalty [m]: (access time + transfer time: tempo per accedere al livello inferiore della gerarchia di memoria più tempo per trasferire il blocco dal livello inferiore della gerarchia)

- $c \ll m$ [**hit time** \ll **miss penalty**]

- AMAT tende a migliorare [$\Leftrightarrow AMAT \rightarrow c$] se $h \rightarrow 1$



Memoria cache



□ Organizzazione della memoria e della cache

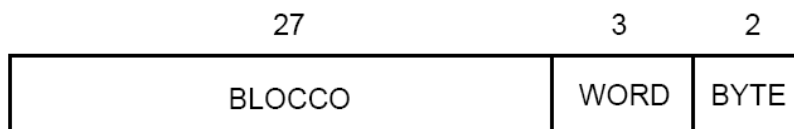
- Lo spazio di memoria è organizzato in **blocchi** (da 4 a 64 byte), anche detti **linee di cache**
 - Ciascuna linea di cache contiene più **word**
 - Ciascuna word contiene più byte
- Le cache sono organizzate in **righe** (o **locazioni** o **slot**): ciascuna contiene una linea di cache, cioè un blocco di memoria
- Tutti i trasferimenti Cache \leftrightarrow Memoria avvengono **a livello di blocco**
- Quando una word non viene trovata in cache, si trasferisce l'intera linea dalla memoria [o dalla cache di livello più basso]

Memoria cache



□ Organizzazione della memoria e della cache

- Esempio:
 - Indirizzi a 32 bit (spazio di indirizzamento di 2^{32} byte)
 - Linee di cache (blocchi) di 32 byte
 - Word di 4 byte
 - Struttura dell'indirizzo [32 bit]:
 - I 27 bit più significativi rappresentano il numero di blocco
 - I successivi 3 bit specificano la word all'interno del blocco
 - I 2 bit meno significativi specificano il byte all'interno della word



Struttura degli indirizzi

Memoria cache



- ❑ Poiché la cache ha dimensione inferiori rispetto alla memoria, **essa conterrà, istante per istante, solo un sottoinsieme dei blocchi di memoria**
- ❑ Di conseguenza i seguenti aspetti devono essere considerati:
 - In caso di miss, in quale riga della cache devo copiare il blocco trasferito dalla memoria? [**Problema del posizionamento di un blocco**]
 - A fronte della richiesta da parte della CPU di una word, in quale riga della cache cercare il blocco corrispondente? Individuata la riga, come capire se il blocco è in cache? [**Problema della ricerca di un blocco**].
 - Nel caso fosse possibile scegliere la riga della cache dove copiare il blocco di memoria a seguito di una miss e la cache è piena, come selezionare il blocco da sostituire nella cache? [**Problema della sostituzione di un blocco**]
 - Cosa succede se una riga di cache che contiene almeno una word modificata [es scrittura in memoria] deve essere sostituita? [**Problema della strategia di scrittura**].

Memoria cache



- Ciascuna locazione di cache, oltre all'eventuale blocco di memoria, contiene in generale altre informazioni:
- **bit di validità [V]**: indica se la locazione contiene (=1) o meno (=0) un blocco di memoria
 - **etichetta [tag]**: se V=1 consente di risalire all'indirizzo completo del blocco contenuti nella locazione della cache; in genere contiene i bit più significativi dell'indirizzo
 - **bit di modifica (o dirty-bit) [D]**: se V=1 indica se il blocco contenuto nella locazione è stato modificato (=1) o meno (=0) dal momento in cui è stato caricato in cache; a seconda della strategia di gestione delle scritture può essere presente o meno

V	D	Tag	Value



□ Problema del posizionamento di un blocco

- In fase di esecuzione, la CPU può a priori tentare di accedere a una qualunque word nello spazio totale di indirizzamento: spazio che può essere qui visto come corrispondente all'intera memoria RAM
- Questo spazio (livello più basso nelle gerarchie della memoria di lavoro), certamente eccede le dimensioni della cache.
- E' necessario definire le possibili modalità per consentire ad ogni parola della memoria indirizzabile di trovare (ove necessario) una posizione della cache in cui possa essere trasferita:
- In sintesi occorre definire una corrispondenza tra l'indirizzo in memoria della word e la riga della cache \Rightarrow 3 strategie:
 - **Cache ad indirizzamento diretto**
 - **Cache set-associativa ad n vie**
 - **Cache completamente associativa**

Memoria cache



□ Cache ad indirizzamento diretto [direct mapping]

- Ogni blocco di memoria corrisponde esattamente ad una locazione della cache. La corrispondenza tra indirizzo di blocco e locazione della cache è data dalla seguente relazione
- $(\text{Ind. blocco})_{\text{cache}} = (\text{Ind. blocco})_{\text{mem}} \bmod (\# \text{ locazioni nella cache})$
- Indicando con $N=2^m$ il numero di locazioni della cache, essendo questo una potenza di 2 l'operazione modulo può essere effettuata considerando semplicemente gli m bit meno significativi dell'indirizzo, che sono pertanto utilizzati come *indice* della cache.
 - Abbiamo tacitamente ipotizzato che 1 blocco di memoria = 1 word = 1 byte [ossia indirizzo di blocco = indirizzo di memoria]; l'estensione al caso generale è banale
- Il problema della sostituzione di un blocco ha soluzione banale
- Vantaggi:
 - Semplicità realizzativa / Poco costosa / Accesso veloce
- Svantaggi:
 - Poco efficiente [spreco di risorse]

Memoria cache



Cache ad indirizzamento diretto [direct mapped]

- Esempio: Cache a indirizzamento diretto con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte \Rightarrow i 2 bit meno significativi dell'indirizzo indicizzano la cache

MEMORIA CENTRALE

Address				Value
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Ogni locazione della cache può contenere 4 possibili blocchi

V	D	Tag	Value
			00
			01
			10
			11

CACHE

Memoria cache



Cache ad indirizzamento diretto [direct mapped]

- Esempio: Cache a indirizzamento diretto con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte \Rightarrow i 2 bit meno significativi dell'indirizzo indicizzano la cache

MEMORIA CENTRALE

Address				Value
0	0	0	0	A
0	0	0	1	B
0	0	1	0	C
0	0	1	1	D
0	1	0	0	E
0	1	0	1	F
0	1	1	0	G
0	1	1	1	H
1	0	0	0	I
1	0	0	1	L
1	0	1	0	M
1	0	1	1	N
1	1	0	0	O
1	1	0	1	P
1	1	1	0	Q
1	1	1	1	R

Ogni locazione della cache può contenere 4 possibili blocchi

V	D	Tag	Value
1	0	01	E
1	0	00	B
1	0	11	Q
0	-	-	-

00
01
10
11

CACHE

Memoria cache



❑ Cache completamente associativa [fully associative]

- Un blocco di memoria può essere caricato in una qualsiasi locazione della memoria cache;
 - Al momento della ricerca tutte le locazioni della cache dovranno essere esaminate.
- Il problema della sostituzione di un blocco ha soluzione non banale
- Vantaggi:
 - Massima efficienza [no spreco di risorse]
- Svantaggi:
 - Complessità realizzativa
 - Più costosa
 - Può essere più lenta

Memoria cache



Cache completamente associativa [fully associative]

- Esempio: Cache fully associative con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte

MEMORIA CENTRALE

Address				Value
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Ogni locazione della cache può contenere uno qualsiasi dei 16 possibili blocchi

V	D	Tag	Value

00
01
10
11

CACHE

Memoria cache



Cache completamente associativa [fully associative]

- Esempio: Cache fully associative con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte

MEMORIA CENTRALE

Address				Value
0	0	0	0	A
0	0	0	1	B
0	0	1	0	C
0	0	1	1	D
0	1	0	0	E
0	1	0	1	F
0	1	1	0	G
0	1	1	1	H
1	0	0	0	I
1	0	0	1	L
1	0	1	0	M
1	0	1	1	N
1	1	0	0	O
1	1	0	1	P
1	1	1	0	Q
1	1	1	1	R

Ogni locazione della cache può contenere uno qualsiasi dei 16 possibili blocchi

V	D	Tag	Value	
00	1	0	0010	C
01	1	0	1001	L
10	0	-	-	-
11	0	-	-	-

CACHE

Memoria cache



□ Cache set associativa a n vie [n -way associative]

- Le locazioni della cache sono suddivise in insiemi (set) ciascuno di n elementi.
- Ad un blocco di memoria corrisponde un solo insieme della cache; tuttavia il blocco può essere caricato in una qualsiasi delle n locazioni dell'insieme stesso
- In altre parole, in una cache set-associativa a n vie, ogni blocco della memoria può essere trasferito in un numero prefissato n di locazioni della cache
- Si utilizzano:
 - Indirizzamento diretto per il mapping indirizzo di blocco → insieme
 - Associatività completa all'interno dell'insieme
- Il problema della sostituzione di un blocco ha soluzione non banale

Memoria cache



□ Cache set associativa a n vie [n -way associative]

- L'idea è quella mediare i vantaggi dell'associatività con quelli dell'indirizzamento diretto
- In realtà l'associatività ad n vie è il meccanismo generale che comprende, come casi particolari:
 - la corrispondenza diretta [$n=1$]
 - l'associatività completa [$n = \text{\#locazioni della cache}$]

Memoria cache



Cache set associativa a n vie [n -way associative]

- Esempio: Cache 2-way associative con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte; \Rightarrow il bit meno significativo dell'indirizzo indicizza il set

MEMORIA CENTRALE

Address				Value
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Ogni blocco di memoria può essere caricato in una qualsiasi delle 2 possibili locazioni del set associato

V	D	Tag	Value

0
1

CACHE

Memoria cache



Cache set associativa a n vie [n -way associative]

- Esempio: Cache 2-way associative con 4 locazioni; indirizzi a 4 bit, blocco = 1 word = 1 byte; \Rightarrow il bit meno significativo dell'indirizzo indicizza il set

MEMORIA CENTRALE

Address				Value
0	0	0	0	A
0	0	0	1	B
0	0	1	0	C
0	0	1	1	D
0	1	0	0	E
0	1	0	1	F
0	1	1	0	G
0	1	1	1	H
1	0	0	0	I
1	0	0	1	L
1	0	1	0	M
1	0	1	1	N
1	1	0	0	O
1	1	0	1	P
1	1	1	0	Q
1	1	1	1	R

Ogni blocco di memoria può essere caricato in una qualsiasi delle 2 possibili locazioni del set associato

V	D	Tag	Value
1	0	101	M
1	0	000	A
0	-	-	-
0	-	-	-

0
1

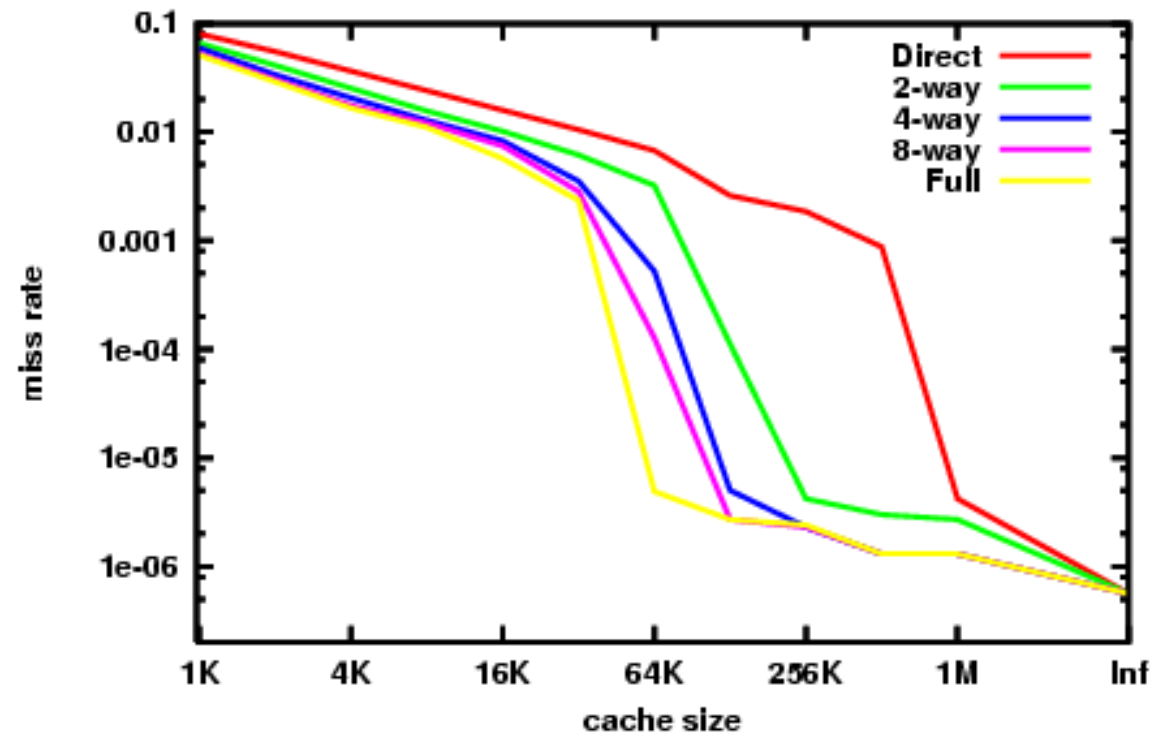
CACHE

Memoria cache



□ Cache set associativa a n vie [n -way associative]

- Analisi delle prestazioni al variare del parametro n :
 - Se $n \rightarrow \#$ locazioni della cache [n -way associative \rightarrow fully associative]:
tendo a sfruttare completamente la size della cache \Rightarrow probabilità di miss diminuisce; tuttavia, ad ogni accesso, la ricerca del blocco coinvolge un numero crescente di locazioni





□ Problema della ricerca di un blocco

- Ogni locazione della cache può contenere parole memorizzate in diverse locazioni di memoria \Rightarrow si utilizza un'etichetta [**tag**] per indicare quale delle possibili parole di memoria si trova nella cache
- Il campo tag viene utilizzato in fase ricerca per verificare se la parola cercata è presente in cache
- È necessario disporre di un metodo per riconoscere se un blocco della cache contiene informazioni valide:
 - Quando un processore viene acceso la cache è vuota e le informazioni in ciascuna locazione [es. tag] non sono significative.
 - Per questo scopo si aggiunge ad ogni locazione della cache un bit di validità (**validity bit**) ad indicare se la locazione stessa contiene effettivamente un blocco di memoria



□ Problema della sostituzione di un blocco

- Quando si verifica un fallimento nell'accesso alla cache, nel caso di cache a indirizzamento diretto c'è un solo candidato alla sostituzione: il problema ha soluzione banale.
- In una cache completamente associativa occorre **decidere** quale blocco sostituire: ogni blocco è un potenziale candidato per la sostituzione.
- In una cache set-associativa, l'insieme interessato dalla sostituzione è identificato immediatamente ma occorre stabilire una politica di sostituzione limitatamente ai blocchi dell'insieme.
- Pertanto, le cache associative e set-associative necessitano di una politica di sostituzione [politica di **eviction** della cache]
- Una politica di eviction ragionevole deve ispirarsi al principio della località temporale



❑ Problema della sostituzione di un blocco

- Esempi di politiche di eviction [richiedono in generale complessità aggiuntiva]:
 - Least Recently Used [**LRU**]: sostituisco il blocco che è stato usato meno di recente [= inutilizzato da più lungo tempo]
 - Least Frequently Used [**LFU**]: sostituisco il blocco che è stato usato meno di frequente
 - First In First Out [**FIFO**]: sostituisco il blocco che stato caricato in cache meno di recente indipendentemente dal fatto che è stato eventualmente utilizzato di recente [poco ispirato al principio di località temporale]
 - **Random**: seleziono casualmente il blocco da sostituire
 - **Ideale**: sostituisco il blocco che **sarà** utilizzato più tardi di tutti gli altri [eventualmente mai più]; la politica non è implementabile



□ Problema della sostituzione di un blocco

- Esempio: analisi dell'hit rate per cache 2 way associative: confronto fra politica LRU a Random

Cache size	LRU	Random
16K	94,6 %	94,1 %
64K	98,2 %	97,9 %
256K	98,81 %	98,80 %

- Oss: non necessariamente la politica Random deve far pensare ad un comportamento prestazionale peggiore della cache



❑ Problema della strategia di scrittura

- A seguito di un'operazione di scrittura in memoria, sarebbe ideale avere:
 - un'operazione di scrittura eseguita velocemente (accedendo solo alla cache)
 - l'informazione contenuta nella cache allineata (consistente) con quella corrispondente nella RAM.
- Due possibili approcci:
 - **Write-through**: il blocco viene modificato in cache **ed** in memoria principale
 - Operazioni di gestione della scrittura più semplici ma scrittura più lenta (la scrittura in memoria non trae vantaggio dalla presenza della cache)
 - **Write-back** (o **Copy back**): il blocco viene modificato **solo** in cache attivando il flag **dirty-bit**; il blocco modificato viene scritto nel livello inferiore della gerarchia (es. memoria) solo quando se ne decide la sostituzione.
 - Operazioni di gestione della scrittura più complesse ma scrittura **immediata**

Memoria cache



- Esempio 1: cache a indirizzamento diretto con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [3 bit meno significativi dell'indirizzo indicizzano la cache]
 - Bootstrap del sistema

	V	D	Tag	Value
000	0	-	-	-
001	0	-	-	-
010	0	-	-	-
011	0	-	-	-
100	0	-	-	-
101	0	-	-	-
110	0	-	-	-
111	0	-	-	-

Memoria cache



- Esempio 1: cache a indirizzamento diretto con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [3 bit meno significativi dell'indirizzo indicizzano la cache]
 - Dopo la gestione di un miss (fallimento di accesso) per l'indirizzo **10110**.

	V	D	Tag	Value
000	0	-	-	-
001	0	-	-	-
010	0	-	-	-
011	0	-	-	-
100	0	-	-	-
101	0	-	-	-
110	1	0	10	Mem(10110)
111	0	-	-	-

Memoria cache



- Esempio 1: cache a indirizzamento diretto con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [3 bit meno significativi dell'indirizzo indicizzano la cache]
 - A seguito di diversi trasferimenti da memoria

	V	D	Tag	Value
000	1	0	11	Mem(11000)
001	0	-	-	-
010	1	0	10	Mem(10010)
011	1	0	00	Mem(00011)
100	1	0	00	Mem(00100)
101	1	0	01	Mem(01101)
110	1	0	10	Mem(10110)
111	0	-	-	-

Memoria cache



- Esempio 1: cache a indirizzamento diretto con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [3 bit meno significativi dell'indirizzo indicizzano la cache]
 - Dopo un 'operazione di scrittura all'indirizzo 10110 [Write back]

	V	D	Tag	Value
000	1	0	11	Mem(11000)
001	0	-	-	-
010	1	0	10	Mem(10010)
011	1	0	00	Mem(00011)
100	1	0	00	Mem(00100)
101	1	0	01	Mem(01101)
110	1	1	10	Mem(10110)
111	0	-	-	-

Memoria cache



- Esempio 1: cache a indirizzamento diretto con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [3 bit meno significativi dell'indirizzo indicizzano la cache]
 - Dopo la gestione di un miss per l'indirizzo **11110**. Il blocco con indirizzo 10110, essendo D=1, deve essere scritto in memoria

	V	D	Tag	Value
000	1	0	11	Mem(11000)
001	0	-	-	-
010	1	0	10	Mem(10010)
011	1	0	00	Mem(00011)
100	1	0	00	Mem(00100)
101	1	0	01	Mem(01101)
110	1	0	11	Mem(11110)
111	0	-	-	-

Memoria cache



- Esempio 2: cache 2-way associative con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [2 bit meno significativi dell'indirizzo indicizzano i 4 set della cache]; politica di eviction LRU
 - Bootstrap del sistema

	V	D	Tag	Value
00	0	-	-	-
	0	-	-	-
01	0	-	-	-
	0	-	-	-
10	0	-	-	-
	0	-	-	-
11	0	-	-	-
	0	-	-	-

Memoria cache



- Esempio 2: cache 2-way associative con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [2 bit meno significativi dell'indirizzo indicizzano i 4 set della cache]; politica di eviction LRU
 - Dopo la gestione di un miss (fallimento di accesso) per l'indirizzo 10110.

	V	D	Tag	Value
00	0	-	-	-
	0	-	-	-
01	0	-	-	-
	0	-	-	-
10	1	0	101	Mem(10110)
	0	-	-	-
11	0	-	-	-
	0	-	-	-

Memoria cache



- Esempio 2: cache 2-way associative con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [2 bit meno significativi dell'indirizzo indicizzano i 4 set della cache]; politica di eviction LRU
 - Dopo la gestione di un miss per l'indirizzo 00110.

	V	D	Tag	Value
00	0	-	-	-
	0	-	-	-
01	0	-	-	-
	0	-	-	-
10	1	0	101	Mem(10110)
	1	0	001	Mem(00110)
11	0	-	-	-
	0	-	-	-

Memoria cache



- Esempio 2: cache 2-way associative con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [2 bit meno significativi dell'indirizzo indicizzano i 4 set della cache]; politica di eviction LRU
 - Dopo la gestione di un hit (successo nell'accesso) per l'indirizzo 10110.

	V	D	Tag	Value
00	0	-	-	-
	0	-	-	-
01	0	-	-	-
	0	-	-	-
10	1	0	101	Mem(10110)
	1	0	001	Mem(00110)
11	0	-	-	-
	0	-	-	-

Memoria cache



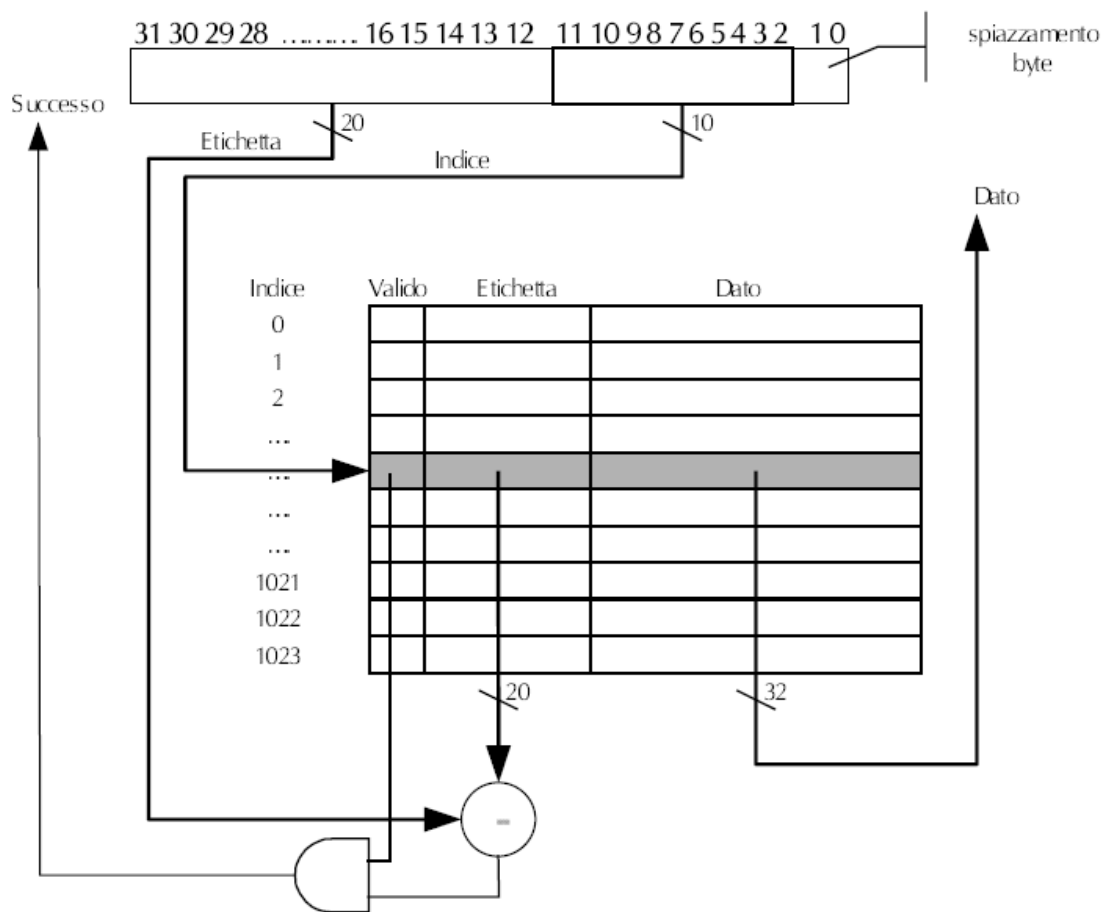
- Esempio 2: cache 2-way associative con 8 locazioni; indirizzi a 5 bit, blocco = 1 word = 1 byte [2 bit meno significativi dell'indirizzo indicizzano i 4 set della cache]; politica di eviction LRU
 - Dopo la gestione di un miss per l'indirizzo 11110. Utilizzando la politica LRU, fra i 2 blocchi di indirizzo 10110 e 00110 il secondo è quello usato meno di recente (malgrado sia entrato per ultimo nella cache) e viene pertanto sostituito

	V	D	Tag	Value
00	0	-	-	-
	0	-	-	-
01	0	-	-	-
	0	-	-	-
10	1	0	101	Mem(10110)
	1	0	111	Mem(11110)
11	0	-	-	-
	0	-	-	-

Memoria cache



- Esempio 3: cache da 4 Kbyte a indirizzamento diretto con blocco = 1 word = 4 byte



1 blocco = 1 word

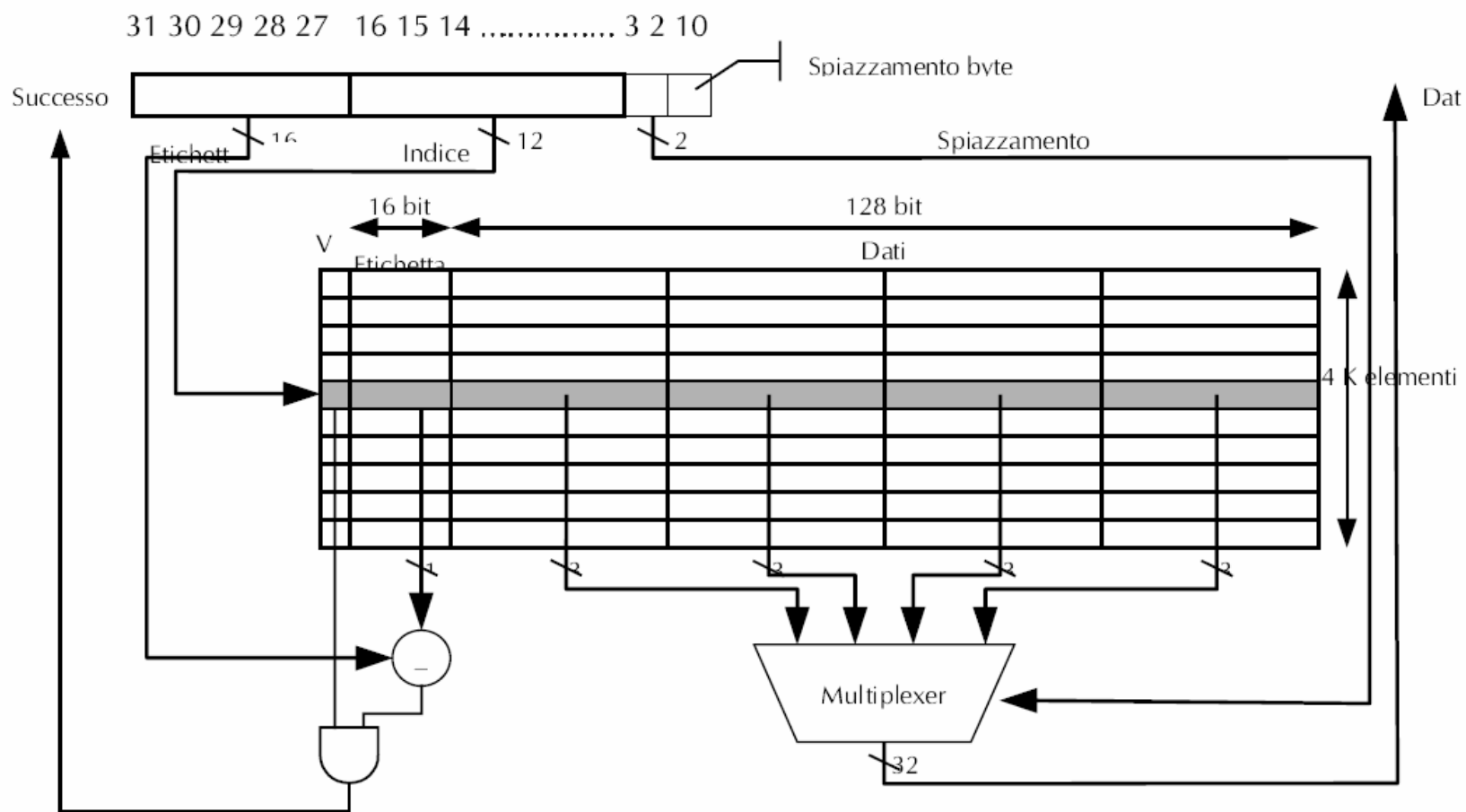


Non sfrutto la
località spaziale
degli accessi

Memoria cache



- Esempio 4: cache da 64 Kbyte a indirizzamento diretto con blocco = 4 word = 16 byte [sfrutto la località spaziale degli accessi]



Memoria cache



- **Esercizio:** Si consideri una cache 2-way associative con 8 locazioni e politica di sostituzione LFU. Supponendo la cache inizialmente vuota, mostrare l'evoluzione del contenuto nel tempo ed i corrispondenti hit / miss a seguito della seguente sequenza di accessi in memoria: 10001, 01001, 01001, 11101

	T ₁			T ₂			T ₃			T ₄		
	V	Tag	Value	V	Tag	Value	V	Tag	Value	V	Tag	Value
00	0	-	-	0	-	-	0	-	-	0	-	-
	0	-	-	0	-	-	0	-	-	0	-	-
01	1	100	Mem(10001)	1	100	Mem(10001)	1	100	Mem(10001)	1	111	Mem(11101)
	0	-	-	1	010	Mem(01001)	1	010	Mem(01001)	1	010	Mem(01001)
10	0	-	-	0	-	-	0	-	-	0	-	-
	0	-	-	0	-	-	0	-	-	0	-	-
11	0	-	-	0	-	-	0	-	-	0	-	-
	0	-	-	0	-	-	0	-	-	0	-	-
MISS			MISS			HIT			MISS			