

## Memorie

### *Caratteristiche principali*

- Locazione: processore, interna (principale), esterna (secondaria)
- Capacità: dimensione parola, numero di parole
- Unità di trasferimento: parola, blocco
- Metodo di accesso: sequenziale, diretto, casuale, associativo
- Prestazioni: tempo di accesso, tempo di ciclo, velocità trasferimento
- Modello fisico: a semiconduttore, magnetico, ottico, magnetico-ottico
- Caratteristiche fisiche: volatile/non volatile, riscrivibile/non riscrivibile
- Organizzazione

Introduzione, Struttura e funzione CPU      Architettura degli elaboratori -1      Pagina 139

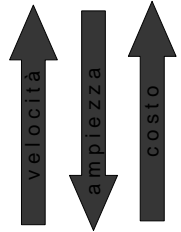
## Gerarchie di memoria

### *Tecnologie di memoria*

L'ideale sarebbe una memoria molto **ampia**, molto **veloce** e molto **economica**

Tecnologia

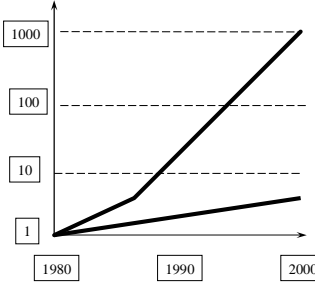
registro  
cache  
SRAM  
DRAM  
disco  
CD/DVD-ROM [meno capace di disco!]  
nastro



Introduzione, Struttura e funzione CPU      Architettura degli elaboratori -1      Pagina 140

## Gerarchie di memoria

### *Prestazioni CPU/memoria*



- Le CPU hanno avuto un aumento di prestazioni notevole, dovuto ad innovazioni tecnologiche ed **architetturali**
- Le memorie sono migliorate **solo** grazie agli avanzamenti tecnologici

Introduzione, Struttura e funzione CPU      Architettura degli elaboratori -1      Pagina 141

## Gerarchie di memoria

### *Proprietà dei programmi*

- Proprietà **statiche** (*dal file sorgente*)
- Proprietà **dinamiche** (*dall'esecuzione*)
  - **Linearità** dei riferimenti
    - Gli indirizzi acceduti sono spesso consecutivi
  - **Località** dei riferimenti
    - Località **spaziale**
      - Gli accessi ad indirizzi **contigui** sono **più probabili**
    - Località **temporale**
      - La **zona di accesso più recente** è quella di **permanenza più probabile**

Introduzione, Struttura e funzione CPU      Architettura degli elaboratori -1      Pagina 142

## Gerarchie di memoria

### *La congettura 90/10*

Un programma impiega mediamente il **90%** del suo tempo di esecuzione alle prese con un numero di istruzioni pari a circa il **10%** di tutte quelle che lo compongono.

## Gerarchie di memoria

### *Divide et impera*

- Conviene organizzare la memoria su più livelli gerarchici:
  - **Livello 1 (cache)**: molto veloce e molto costosa  
⇒ dimensioni ridotte, per i dati ad accesso più probabile [anche più livelli di cache]
  - **Livello 2 (memoria centrale)**: molto ampia e lenta  
⇐ costo contenuto, per tutti i dati del programma

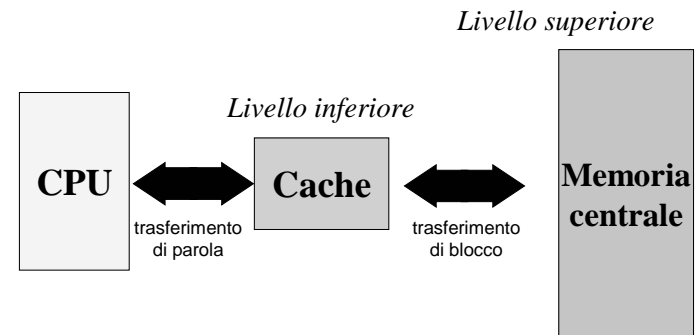
## Gerarchie di memoria

### *Organizzazione gerarchica*

- Memoria a livelli
  - Al livello più alto (**superiore**) stanno i ‘supporti di memoria’ più capaci, più lenti e meno costosi
  - Ai livelli più bassi (**inferiori**) si pongono supporti più veloci, più costosi e meno capaci
  - La CPU usa direttamente il **livello più basso**
  - Ogni livello superiore deve contenere tutti i dati presenti ai livelli inferiori (ed altri)

## Gerarchie di memoria

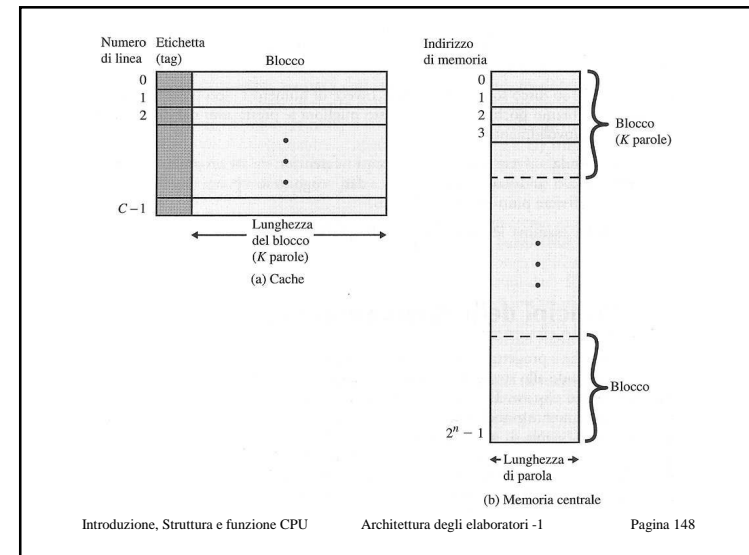
### *Schema concettuale*



## Gerarchie di memoria

### Suddivisione in blocchi

- Per realizzare un'organizzazione gerarchica conviene suddividere la memoria in **blocchi**
- La **dimensione** di un blocco è la **quantità minima indivisibile** di dati che occorre prelevare (copiare) dal livello superiore
- L'**indirizzo** di un dato diviene l'indirizzo del blocco che lo contiene *sommato* alla posizione del dato all'interno del blocco



## Gerarchie di memoria

### Hit e miss

Un dato richiesto dalla CPU può essere presente in cache (**hit**) oppure mancante (**miss**)

- Un **hit**, *successo*, deve essere molto probabile (>90%) se si vuole guadagnare efficienza prestazionale
- Un **miss**, *fallimento*, richiede l'avvio di una procedura di scambio dati (**swap**) con il livello superiore

## Gerarchie di memoria

### Tempo medio di accesso

$T_a$  : Tempo medio di accesso ad un dato in memoria

$$T_a = T_h \times P_h + T_m \times (1 - P_h)$$

$T_h$  = tempo di accesso ad un dato **presente** in cache

$T_m$  = tempo medio di accesso ad un dato **non** in cache

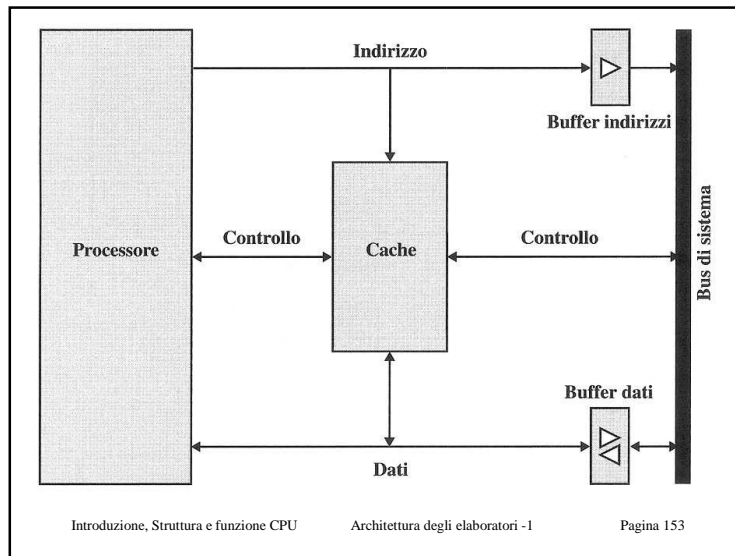
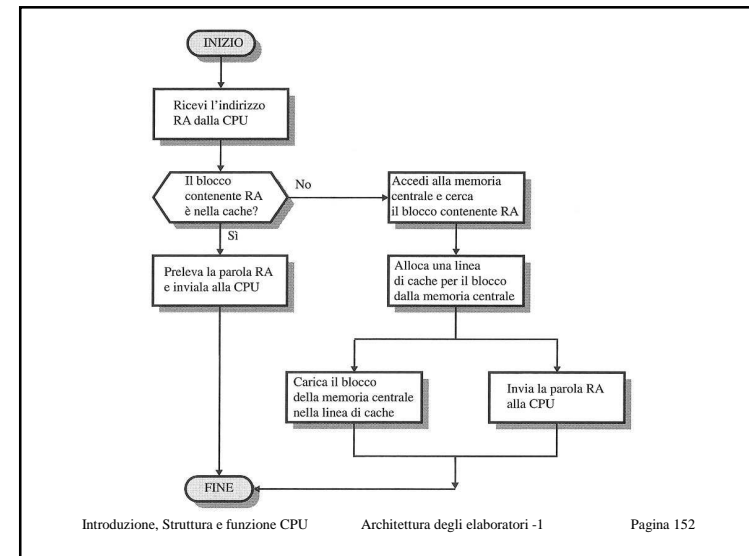
(funzione della dimensione del blocco)

$P_h$  = probabilità di **hit**

(funzione della dimensione del blocco e della politica di gestione)

## Gerarchie di memoria *Tecnica generale*

- Suddivisione della memoria centrale in blocchi *logici*
- Dimensionamento della cache in **multiplo** di blocchi
- Per ogni indirizzo emesso dalla CPU
  - **Hit** ⇒ Il dato richiesto viene fornito **immediatamente** alla CPU
  - **Miss** ⇒ La cache richiede il dato al livello superiore  
Il blocco contenente il dato viene posto in cache  
Il dato richiesto viene fornito alla CPU



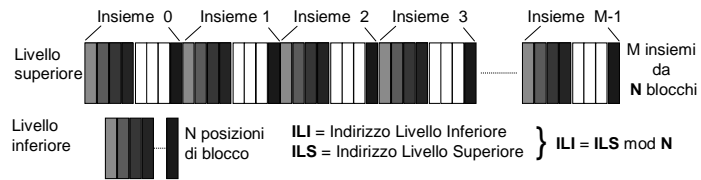
## Gerarchie di memoria *Problematiche*

- Organizzazione della cache e tecniche di allocazione
- Individuazione di hit o miss
- Politica di rimpiazzo dei blocchi
- Congruenza dei blocchi

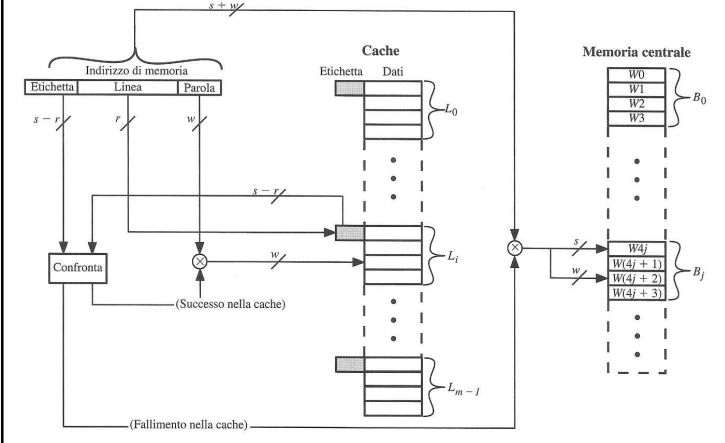
## Gerarchie di memoria Associazione diretta

Tecnica nota come **direct mapping**

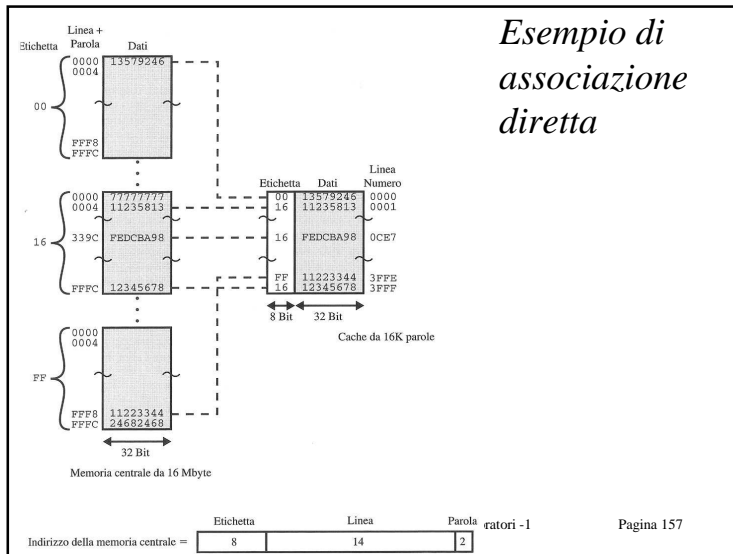
- Ogni blocco del livello superiore può essere allocato *solo* in una specifica posizione (detta **linea** o **slot**) del livello inferiore



## Associazione diretta



## Esempio di associazione diretta



## Gerarchie di memoria Associazione diretta

### Vantaggi

- Semplicità di traduzione da indirizzo ILS (memoria) ad indirizzo ILI (cache)
- Determinazione veloce di hit o miss

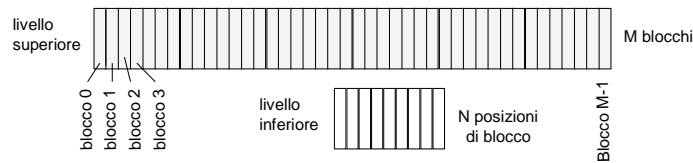
### Svantaggi

- Necessità di contraddistinguere il blocco presente in ILI (introduzione di un'etichetta, 'tag')
- Swap frequenti per accesso a dati di blocchi adiacenti

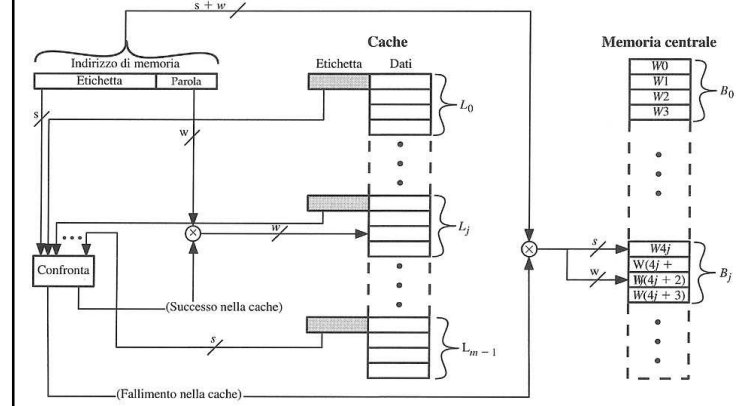
## Gerarchie di memoria Associazione completa

Tecnica nota come *fully associative*

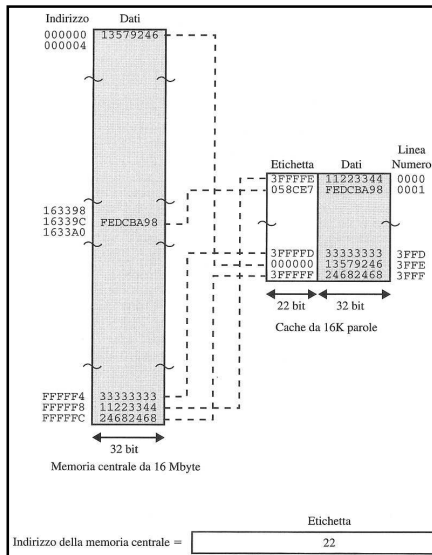
- Ogni blocco del livello superiore può essere posto in *qualsunque* posizione del livello inferiore



## Associazione completa



## Esempio di associazione completa



## Gerarchie di memoria Associazione completa

Alla cache capace di N blocchi viene associata una tabella di N posizioni, contenenti il numero di blocco effettivo (**tag**) in essa contenuto.

### Vantaggi

- Massima efficienza di allocazione

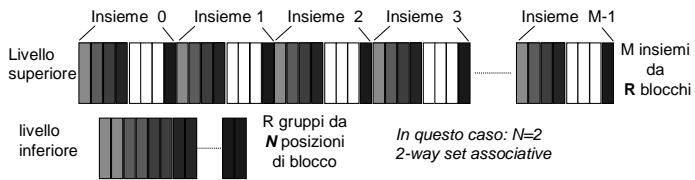
### Svantaggi

- Determinazione onerosa della corrispondenza ILL-ILS e della verifica di hit/miss

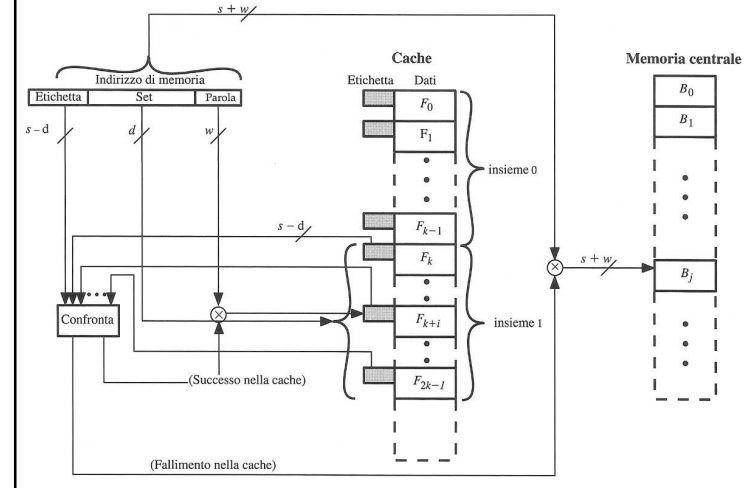
## Gerarchie di memoria Associazione a gruppi

Tecnica nota come *N-way set associative*

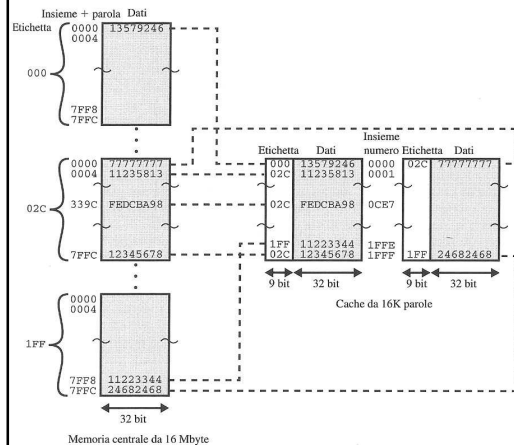
- Ogni blocco di un certo insieme di blocchi del livello superiore può essere allocato liberamente in uno specifico gruppo di blocchi del livello inferiore



## Associazione a gruppi



## Esempio di associazione a gruppi



## Gerarchie di memoria Associazione a gruppi

- Alla cache, composta da R gruppi di N posizioni di blocco ciascuno, si affiancano R tabelle di N elementi, contenenti le etichette (**tag**) che designano i blocchi effettivi posti nelle posizione corrispondente.
  - **Valutazione:** buona efficienza di allocazione a fronte di una sopportabile complessità di ricerca

## Gerarchie di memoria

### *Politiche di rimpiazzo dei blocchi*

Quale blocco conviene sostituire in cache per effettuare uno swap? (*Penalità di miss*)

- **Casuale**, per occupazione omogenea dello spazio
- **Least Recently Used (LRU)**, per preservare *località temporale*

P(miss)	N-way	rimpiazzo casuale			rimpiazzo LRU		
		2	4	8	2	4	8
16 KB	5,69	5,29	4,96	5,18	4,67	4,39	
64 KB	2,01	1,66	1,53	1,88	1,54	1,39	
256 KB	1,17	1,13	1,12	1,15	1,13	1,12	

Introduzione, Struttura e funzione CPU

Architettura degli elaboratori -1

Pagina 167

## Gerarchie di memoria

### *Il problema della scrittura*

La scrittura dei dati determina *incoerenza* tra il blocco in cache e quello nei livelli superiori

- **‘Write through’**
  - Scrittura *contemporanea* in cache e nel livello di memoria superiore
  - Aumento di traffico per frequenti scritture nel medesimo blocco, ma i dati sono sempre coerenti tra i livelli
  - Si ricorre a buffer di scrittura *asincroni* (differiti) verso la memoria.

*N.B.: La memoria contiene istruzioni e dati, e solo il 50% delle operazioni sui dati sono scritture (circa 12 % del totale)*

Introduzione, Struttura e funzione CPU

Architettura degli elaboratori -1

Pagina 168

## Gerarchie di memoria

### *Il problema della scrittura*

- **‘Write back’**
  - Scrittura in memoria superiore *differita* al rimpiazzo del blocco di cache corrispondente
  - Occorre ricordare se sono avvenute operazioni di scrittura nel blocco
  - Consente ottimizzazione del traffico tra livelli
  - Causa periodi di *incoerenza*

Introduzione, Struttura e funzione CPU

Architettura degli elaboratori -1

Pagina 169

## Gerarchie di memoria

### *Il problema dei ‘miss’*

- Miss di **primo accesso**, *inevitabile* e non riducibile
- Miss per **capacità insufficiente**, quando la cache *non può* contenere tutti i blocchi necessari all’esecuzione del programma
- Miss per **conflitto**, quando *più* blocchi possono essere mappati (con associazione diretta o a gruppi) su *uno* stesso gruppo

Introduzione, Struttura e funzione CPU

Architettura degli elaboratori -1

Pagina 170



## Gerarchie di memoria Il problema dei 'miss'

- Tecniche 'classiche' di soluzione
  - Maggior dimensione di blocco
    - Buona per fruire di *località spaziale*
    - Causa incremento di miss per conflitto (meno blocchi disponibili)
  - Maggiore associatività
    - Causa incremento del tempo di localizzazione in gruppo (hit)
    - Soggetta alla 'regola del 2:1'
      - Una cache ad N blocchi con associazione diretta ha una probabilità di miss pressoché uguale ad una cache di dimensione N/2 con associazione a 2 vie

## Gerarchie di memoria Il problema dei 'miss'

- Altre tecniche
  - Separazione tra cache *dati* e cache *istruzioni* (già visto in pipeline)
  - Ottimizzazione degli accessi mediante compilatori
    - Posizionamento accurato delle procedure ripetitive
    - Fusione di vettori in strutture (località spaziale)
    - Trasformazioni di iterazioni annidate (località spaziale)
    - ...

## Gerarchie di memoria Es.: Fusione di vettori in strutture

```

/* prima della ottimizzazione */
int val[SIZE];
int key[SIZE];

/* dopo l'ottimizzazione */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
                    
```

MEMORIA

merged\_array[0]

MEMORIA

## Gerarchie di memoria Es.: Iterazioni annidate

```

/* prima della ottimizzazione */
for (j=0;j<100;j=j+1)
    for (i=0;i<5000;i=i+1)
        x[i][j] = 2*x[i][j];

/* dopo l'ottimizzazione */
for (i=0;i<5000;i=i+1)
    for (j=0;j<100;j=j+1)
        x[i][j] = 2*x[i][j];
                    
```

controllo prima

controllo dopo