Introduzione all'Architettura del Calcolatore

lezione 4

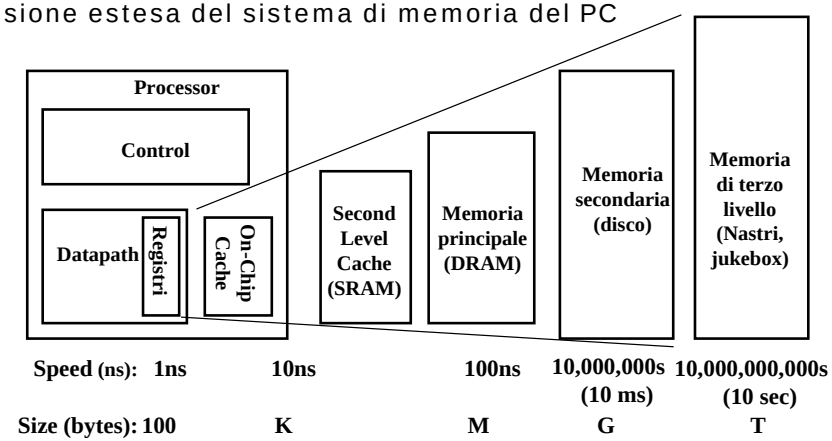
Luciano Bononi

bononi@cs.unibo.it

http://www.cs.unibo.it/~bononi/

Figure credits: some figures have been taken from web presentations on the web: Larry and Nancy Long, Mike Schulte

Visione estesa del sistema di memoria del PC

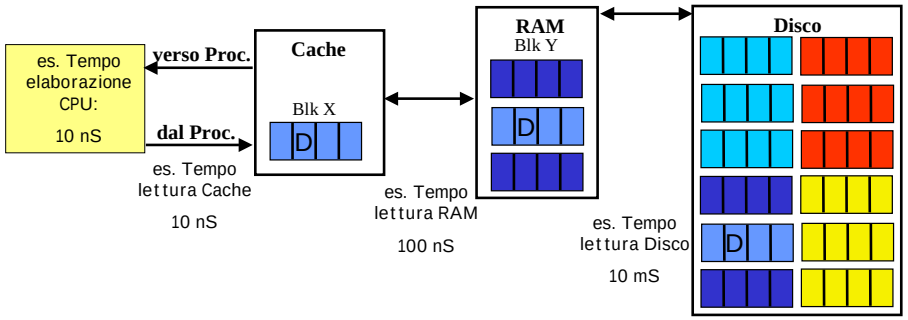


Velocità: massima
Capacità: inferiore
Costo: elevato

Velocità: minima
Capacità: superiore
Costo: ridotto

Come funziona la gerarchia di memoria?

- **I dati vengono cercati da sinistra (lato della CPU) verso destra (lato delle unità di memoria via via più capienti e più lente), finchè non vengono trovati. Prima vengono trovati, minore è il tempo richiesto.**
- **I blocchi di dati (più grandi sul lato destro, più piccoli a sinistra) contenenti il dato cercato vengono spostati verso sinistra, per favorire gli accessi futuri (vedi principi di locality tra due slide)**



Gerarchia di memoria: terminologia

- **Hit: quando il dato richiesto si trova nella memoria di alto livello (es. cache)**
  - Hit Rate: frazione di accessi in memoria il cui dato si trova in cache
  - Hit Time: tempo di accesso al livello superiore (cache)= tempo di accesso a memoria + tempo per rilevare HIT/MISS
- **Miss: quando il dato richiesto non si trova nella memoria di alto livello (es. cache) ma deve essere caricato dalla memoria di basso livello**
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: tempo necessario a sostituire un blocco + tempo per recapitare il blocco al processore
- **Hit Time << Miss Penalty**

Gerarchia di memoria: prestazioni

- **Prestazioni della gerarchia (1 solo livello: cache-RAM)**
  - h=hit rate in cache, e quindi (1-h)=miss rate
  - m=tempo di accesso dato in RAM (miss penalty)
  - c=tempo di accesso dato in cache (hit time)
- **Tempo medio di accesso (TMA) a un dato memorizzato**
  - TMA = c*h + (c+m)*(1-h) = c + (1-h)*m
  - es. se h=90%, c=5 ns e m=30 ns, TMA=8 ns
  - Domanda: conviene investire 1000 euro in cache arrivando a h=99%?
    - TMA = 5 + (1-0.99)*30 = 5.3 ns contro gli 8 ns precedenti.
  - Risposta... dipende!

Come viene gestita la gerarchia?

- **Registri <-> Memoria**
  - es. attraverso il compilatore (e il programmatore)
- **cache <-> memoria**
  - controllo hardware (+ “arte” programmazione)
- **memoria <-> dischi**
  - controllo hardware
  - supporto del sistema operativo (es. memoria virtuale)
  - programmatore (definizione dei files)

Tecnologia di memoria

- **Accesso casuale (Random Access):**
  - “Random”: tempo di accesso costante per ogni dato in ogni indirizzo
  - DRAM: Dynamic Random Access Memory
    - alta densità, basso consumo, costa poco, lenta
    - Dynamic: deve essere “rinfrescata” regolarmente
  - SRAM: Static Random Access Memory
    - bassa densità, alto consumo, costosa, veloce
    - Static: mantiene il contenuto senza necessità di “refresh” (se alimentata)
- **“Non-così-random” (mixed):**
  - il tempo di accesso varia a seconda della posizione dei dati e del momento in cui accedo
  - Es.: Dischi, CDROM
- **Sequenziale (Sequential Access):**
  - tempo di accesso dipende linearmente dalla locazione dati (nastro)

Miglioramenti delle DRAM (tempo di accesso m)

- *Page mode e EDO* **(Extended Data Out) DRAM:**
  - si accede la RAM per “righe”, memorizzate in buffer che funziona come una SRAM (veloce).
  - a questo punto, mediante gli indirizzi di colonna, i diversi bit della riga possono essere recuperati in parallelo.
  - Prestazioni: 120 ns -> 60 ns (page mode) -> 25 ns (EDO)
- *SDRAMs* **(S =synchronous): i bit sono espulsi dal buffer secondo un clock (~100 MHz).**
  - questo evita il bisogno di passare gli indirizzi di colonna dei bit. E’ utile per accessi burst. Il tempo per accedere ai bit ***dopo che si è fatto l’accesso alla riga è*** = 8-10 ns !
- **usano tecnologia DRAM, poco costo aggiuntivo porta vantaggi consistenti**

Quadro riassuntivo

Level	Memory Technology	Typical Size	Typical Access Time	Cost per Mbyte
Registers	D Flip-Flops	64 32-bit	2 -3 ns	N/A
L1 Cache (on chip)	SRAM	16 Kbytes	5 - 25 ns	$100 - $250
L2Cache (off chip)	SRAM	256 Kbytes	5 - 25 ns	$100 - $250
Main Memory	DRAM	256 Mbytes	60 - 120 ns	$5 - $10
Secondary Storage	Magnetic Disk	8 Gbytes	10 - 20 ms	$0.10-$0.20

Domande critiche per il progettista

- **D1: Come faccio a essere sicuro che il dato cercato è nel blocco del livello superiore? (identificazione del blocco)**
- **D2: dove vado a memorizzare il blocco dati che viene copiato dal livello inferiore al livello superiore di memoria? (scelta dello slot)**
  - sappiamo che ad ogni accesso al dato, il suo sotto-blocco sale di livello fino alla cache
- **D3: Quale "blocco superiore" deve essere sostituito da uno "inferiore" se lo spazio vuoto della cache è finito? (rimpiazzamento del blocco)**
- **D4: Cosa succede se devo aggiornare (scrivere) un nuovo dato? (politica di "scrittura")**
  - scrivo il dato in cache? oppure in tutte le copie del dato presenti nella gerarchia? Corro il rischio di avere copie di dati inconsistenti?

Locality: permette di sfruttare la gerarchia

- **Sebbene le memorie permettano accesso casuale a larghi spazi di indirizzamento, i programmi in esecuzione accedono solo a *piccole porzioni* dei dati in ogni istante.**
- **Se un dato viene richiesto a un momento dell'esecuzione**
  - esso tende a essere richiesto di nuovo, entro poco tempo (***Temporal locality)***
  - i dati a esso vicini tendono a essere richiesti, entro poco tempo (***Spatial locality)***
- *Trucco:* **copiamo queste piccole porzioni di dati in una piccola e veloce memoria vicina alla CPU.**
- **Programmatori e compilatori lavorano insieme sulle ipotesi della gerarchia per cooperare ad aumentare l'effetto del principio di locality, e quindi a trarre vantaggio dalla gerarchia.**

Principi generali della gerarchia di memoria

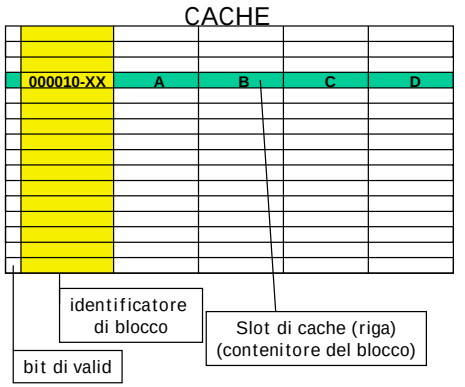
- **Locality**
  - *Locality Temporale*: il dato al quale si accede in memoria viene di solito richiesto nuovamente entro poco tempo (es. ciclo ripetuto)
  - *Locality Spaziale*: i dati vicini a quello appena richiesto di solito sono richiesti entro poco tempo (es. scansione di un vettore)
    - Questi due principi sono alla base dello sfruttamento efficace della gerarchia di memoria!
    - Sappiamo che i dati richiesti vanno portati "vicino al processore", possibilmente con i dati a loro "vicini", in quanto in questo modo siamo in grado di ottenere h elevato, e quindi TMA ridotto, anche se la cache è molto inferiore alla RAM, la RAM è inferiore al disco, ecc.

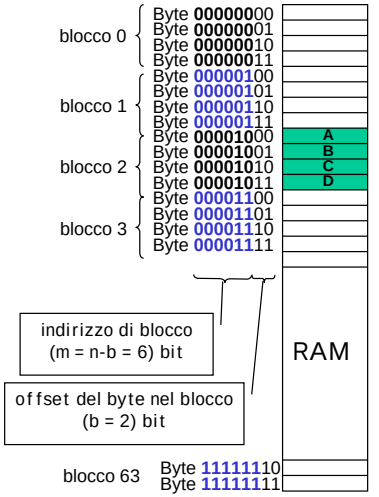
Cache

- **Due problemi:**
  - Come sappiamo se il dato cercato è in Cache? (hit check)
  - Se il dato è in cache, come lo troviamo?
- **Dato, blocco di RAM e slot di cache**
  - supponiamo il dato sia pari a un unità di memoria indirizzabile della RAM (es. 1 Byte)
  - un blocco è un gruppo di 2^b dati (b intero)
    - es. 1 blocco da 2 (b=1) ,4 (b=2),8 (b=3)... Byte
  - uno slot è un contenitore (in cache) per un blocco.

Cache

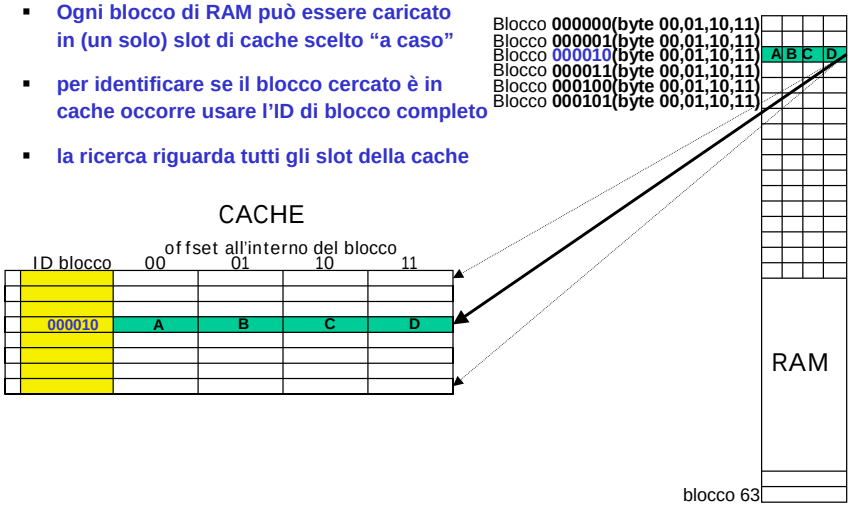
- **es. spazio di indirizzamento di RAM:**
  indirizzi da n=8 bit (256 Byte di RAM)
  dato= 1 byte, blocco=2^b (b=2) = 4 byte
  indirizzo di blocco = m = n-b = 6 bit
- **slot di cache = 2^s (s = 4) = 16**





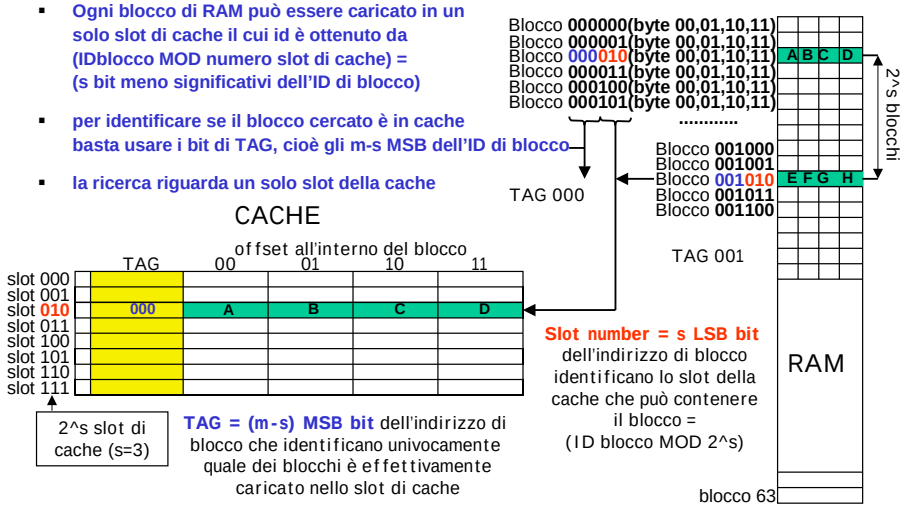
Cache di tipo “Associative”

- **Ogni blocco di RAM può essere caricato in (un solo) slot di cache scelto “a caso”**
- **per identificare se il blocco cercato è in cache occorre usare l’ID di blocco completo**
- **la ricerca riguarda tutti gli slot della cache**



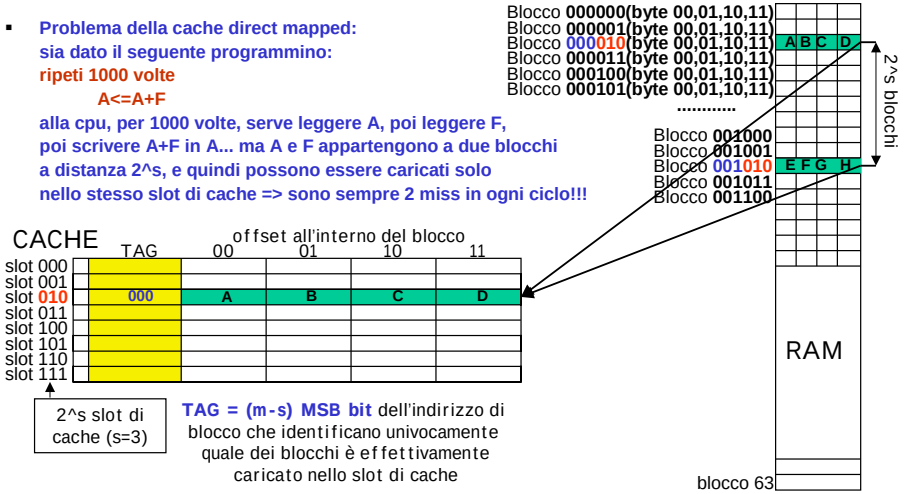
Cache di tipo “Direct Mapping”

- **Ogni blocco di RAM può essere caricato in un solo slot di cache il cui id è ottenuto da (IDblocco MOD numero slot di cache) = (s bit meno significativi dell’ID di blocco)**
- **per identificare se il blocco cercato è in cache basta usare i bit di TAG, cioè gli m-s MSB dell’ID di blocco**
- **la ricerca riguarda un solo slot della cache**



Cache di tipo “Direct Mapping”

- **Problema della cache direct mapped:**
  sia dato il seguente programmino:
  ripeti 1000 volte
  A<=A+F
  alla cpu, per 1000 volte, serve leggere A, poi leggere F, poi scrivere A+F in A... ma A e F appartengono a due blocchi a distanza 2^s, e quindi possono essere caricati solo nello stesso slot di cache => sono sempre 2 miss in ogni ciclo!!!

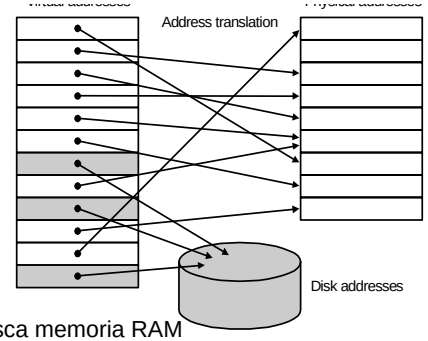


Cache di tipo “Direct Mapping”

- **Esercizi**
- **Un ciclo usa due Byte di indirizzo esadecimale $1234 e $FAF4. Se il blocco dati contenuto in uno slot di cache è di 2^b Byte (b=3) e il numero di slot della cache è di 2^s (s=4) è utile avere una cache direct mapped? e se (s=3)? e se invece s=3 ma b=4? Perchè? Che dimensione ha il campo TAG in bit nei tre casi? Che dimensione avrebbe il campo ID di blocco se la cache fosse Associativa, nei tre casi?**
- **Assumendo accessi completamente casuali alla RAM di 256 Byte, e assumendo di avere una cache con capacità di 32 Byte (senza contare il bit di valid e il campo ID del blocco), che valore di hit ratio h potrei aspettarmi, nell'esecuzione di un programma molto lungo?**
- **Assumere una RAM da 1024 Bytes, blocchi dati da 2^b (b=3)bytes, con una cache da 2^s slot (s=4) ognuno in grado di contenere un blocco dati. Che capacità di memorizzazione totale avrà la cache (in bit) assumendo che si tratti di una cache associativa? e se si tratta di una cache direct mapped? (contare un bit di valid, il campo ID o TAG e il blocco dati per ogni slot)**
- **In quale slot di cache direct mapped, con b=3 e s=4, verrebbe inserito il BYTE di indirizzo esadecimale $F3C? e se b=4 e s=3? e se b=4 e s=4? e se $F3C fosse l'indirizzo del blocco?**

Memoria Virtuale

- **La memoria RAM può funzionare come una cache rispetto alla memoria secondaria (disco):**



- **Obiettivi:**
  - ***Illusione*** di avere una gigantesca memoria RAM
  - ***Rilocazione dei programmi (tanti programmi in RAM)***
  - ***Protezione dei programmi (spazi di indirizzi separati)***

Memoria Virtuale: terminologia

- *Pagina:* **equivalente del “*blocco*” di dimensione fissata.**
- *Page faults:* **equivalente del “*miss*”**
- *Virtual address:* **equivalente del** *“tag.”*
  - ***N.B. gli indirizzi virtuali sono a molti bit. La posizione in RAM fisica di una pagina viene mantenuta attraverso tabelle di paginazione (non attraverso i campi ID o TAG).***
- *Physical address:* **è l'indirizzo effettivo nella RAM fisica del dato puntato dal virtual address. Non ha l'equivalente per la cache.**
- *Memory mapping (address translation):* **è la conversione degli indirizzi virtuali in indirizzi fisici. Non ha equivalente per cache.**