1 Lezione del 04-03-25

1.1 Hard disk

Gli **hard disk** (*dischi rigidi*) sono effettivamente, seppur memorie, **periferiche**, collegate al bus attraverso la loro interfaccia. La CPU non puo' eseguire programmi direttamente dall'hard disk, ma deve prima caricarli in memoria principale (memoria RAM).

Questo perche' letture e scritture in hard disk vengono effettuate per **blocchi** (storicamente di 512 byte), e richiedono molto piu' tempo di quanto sia possibile aspettare al prelievo di istruzioni o operandi.

Nello standard PC AT gli hard disk usano interfacce **SATA**.

Dal punto di vista elettromeccanico venivano realizzati attraverso dischi di materiale ferromagnetico imperniati ad un asse centrale, con testine mobili che scandivano il raggio dei dischi, rilevando o modificando la loro magnetizzazione per accedere all'informazione. Il complesso di dischi e testine viene detto **drive**.

L'informazione viene disposta su ogni disco in **settori** e **tracce**. Le tracce sono concentriche e i settori formano degli "spicchi" di ogni faccia. Notiamo che entrambe le facce di ogni disco possono memorizzare informazione. Un **blocco** e' quindi formato dalla regione di una traccia compresa in un certo sensore.

I dischi vengono tenuti continuamente in rotazione (negli ordini delle centinaia/migliaia di RPM). Il tempo che la testina impiega a raggiungere una tracca viene detto **tempo di seek**, t_{seek}, il tempo che alla velocita' di rotazione del disco l'informazione si trovi sotto la testina **latenza** $t_{latency}$ e il tempo necessario ad effettuare l'operazione vera e propria **tempo di lettura/scrittura** $t_{r/w}$, per cui il tempo di lettura/scrittura complessivo risulta:

$$t_{seek} + t_{latency} + t_{r/w} \sim 1\,\mathrm{ms}$$

nell'ordine del millisecondo, per la CPU estremamente (milioni di volte) piu' lento della RAM.

Quello che accade al tempo di lettura e' che il blocco viene copiato in un buffer di memoria nell'interfaccia che viene poi reso disponibile alla CPU. Viceversa, al tempo di scrittura il buffer viene riempito dalla CPU, e l'interfaccia si occupa poi di copiarlo all'interno del settore giusto.

Per effettuare un operazione dobbiamo quindi sapere:

- Quale *testina* individuare;
- Quale *traccia* individuare;
- Quale *regione* (quindi quale *blocco*) individuare.

Storicamente queste informazioni erano gestite lato software, concedendo la possibilita' di alterare la *formattazione* del disco. Oggi la formattazione e' definita in fabbrica, e l'interfaccia offre una sua astrazione. In questa astrazione ogni blocco e' quindi indirizzato da un indirizzo logico, il **Logical Block Address**, **LBA**.

1.1.1 Interfaccia SATA

L'interfaccia del PC IBM e' dotata di diversi registri a 8 bit e uno a 16 bit:

- **Registri di selezione** del blocco:
  - **SNR** (Sector Number);

  - **CNL** (Cylinder Number Low);
  - **CNH** (Cylinder Number High);
  - **HND** (Head And Drive): solo gli ultimi 4 bit di questo registro formano l'informazione sulla testina da utilizzare. Gli altri bit vengono usati diversamente, ad esempio per selezionare quale drive usare in configurazioni master/slave, o per abilitare il LBA, usando quindi i registri di selezione per specificare un indirizzo logico (su $3 \cdot 8 = 4 = 28$ bit) anziche' un informazione geometrica sulla posizione del blocco desiderato.

  Vediamo che dalla dimensione dell'LBA (assumiamo che per indirizzamento geometrico si trova la stessa cosa) si ha una dimensione del disco:

$$2^{28} \cdot 2^9 = 2^{37} = 128\,\text{GB}$$

  Per questo si puo' abilitare la modalita' **LBA48** (che non e' un gruppo di idol giapponesi), dove ci si aspetta il LBA venga specificato in due passate, una da 24 bit e una da 20 bit sugli stessi registri.

- **SCR** (Section Counter): permette di specificare su quanti settori contigui a partire da quello specificato prima eseguire l'operazione;
- **BR** (Buffer Register): l'unico registro a 16 bit, permette di accedere al buffer 2 byte alla volta;
- **STS** (Status Register): il classico registro di stato che ci notifica se un'operazione e' conclusa o si puo' effettuare;
- **CMD** (Command): serve a specificare l'operazione da effettuare (lettura, scrittura, ecc...).

1.2 Caching

Abbiamo detto che la memoria RAM e' molto piu' veloce dei dischi rigidi. Questo e' vero, ma non significa che non ci sia comunque un certo dislivello tra la velocita' della CPU e la velocita' della RAM: un operazione puo' comunque richiedere nel'ordine dei ~ 100 circa cicli di clock.

Per questo motivo si inframezzano fra la CPU e la RAM piu' memorie, relativamente piccole ma veloci, dette **memorie di cache**.

L'idea e' che la RAM in se' e' costituita da memoria dinamica (DRAM), quindi a condensatori, relativamente lenta e con tempo di refresh, mentre le memorie di cache vengono implementate con memorie statiche, piu' veloci ma piu' costose da realizzare su larga scala (per cui le dimensioni ridotte).

1.2.1 Principi di localita'

Le piccole dimensioni delle memorie vengono aiutate dalla **localita'** del codice in memoria: istruzioni che compongono le stesse funzioni avranno istruzioni vicine fra di loro, le strutture definite dal programmatore conterranno dati locali, ecc... In particolare, potremo distinguere fra due **principi di localita'**:

- **Localita' temporale:** una volta visto un indirizzo, e' probabile che questo o indirizzi ad esso vicini siano visti di nuovo;

- **Localita' spaziale:** solitamente si accede ad indirizzi vicini fra di loro.

La cache avra' quindi il compito di memoizzare i valori prelevati con frequenza dalla DRAM. Possiamo immaginare che la prima lettura di un dato richiedera' il tempo completo di accesso, ma la lettura successiva, ammesso che quel dato sia stato salvato nella cache, richiedera' un tempo di accesso significativamente minore.

L'importante e' che questo processo sia **trasparente** per la CPU, cioe' che questa non si debba preoccupare di quali indirizzi sono stati visti dalla cache e memoizzati e quali no. Il risultato finale e' la velocizzazione di un qualsiasi programma senza dover agire in nessun modo sul programma stesso. Di contro, non e' detto che il programmatore non possa sfruttare la presenza della memoria cache, cercando di sviluppare algoritmi e strutture dati che rispettano il piu' possibile i principi di localita' (tecniche *data driven*).

1.2.2 Cache ad indirizzamento diretto

Vediamo un primo esempio di memoria cache. Abbiamo che lato processore ci arriveranno le linee di byte enable (BE) e le linee di indirizzo (A). Inoltre avremo a disposizione un bus dati (D) di un certo numero di linee.

Vorremo porre fra CPU e DRAM una cache, connessa a quest'ultima da una **cacheline** da 64 byte.

In fase di lettura, invece di leggere l'unica riga richiesta dal processore, si procedera' alla lettura di un certo numero di righe (poniamo 8). Questo significa che per un tempo di lettura di riga di t, ci vorra' un tempo $\sim 8t$ (solitamente meno). La speranza e' che queste righe verranno lette successivamente dal processore.

Inoltre, ad ogni "blocco" di memoria letto dalla cache si dovra' associare dell'informazione riguardo alla posizione in memoria (informazione che viene contenuta nella **memoria delle etichette**). E' quindi piu' conveniente leggere regioni relativamente piu' grandi di memoria, in modo da non sprecare *overhead* per piccole quantita' di dati.

La divisione della DRAM in cacheline e' quindi realizzata giocando sulle scomposizioni degli indirizzi. Per ottenere la regione corrispondente ad un indirizzo (il numero di cacheline) si realizza una sorta di *funzione di hash*, prendonendo gli n bit piu' significativi della linea di indirizzo A e usandoli come chiave per la regione di dati corrispondente. Inoltre, alla regione selezionata si associa un singolo bit di validita'. Un comparatore fra etichetta e gli n bit piu significativi messo in AND a questo bit di validita' ci assicurera' quindi la presenza nella cacheline del dato richiesto, detta **hit/miss**. A questo punto bastera' ricavare una linea di offset dai bit meno significativi di A, e leggere dalla memoria cache a tale offset, all'indice indicato dall'etichetta. riguarda bene

Notiamo che questa cache soffre di problemi di **collisione**: infatti ci sara' un numero di regioni con lo stesso indice (etichetta ?) pari alla dimensione della RAM fratto la dimensione della cache.