4 - Organizzazione dei sistemi di calcolo

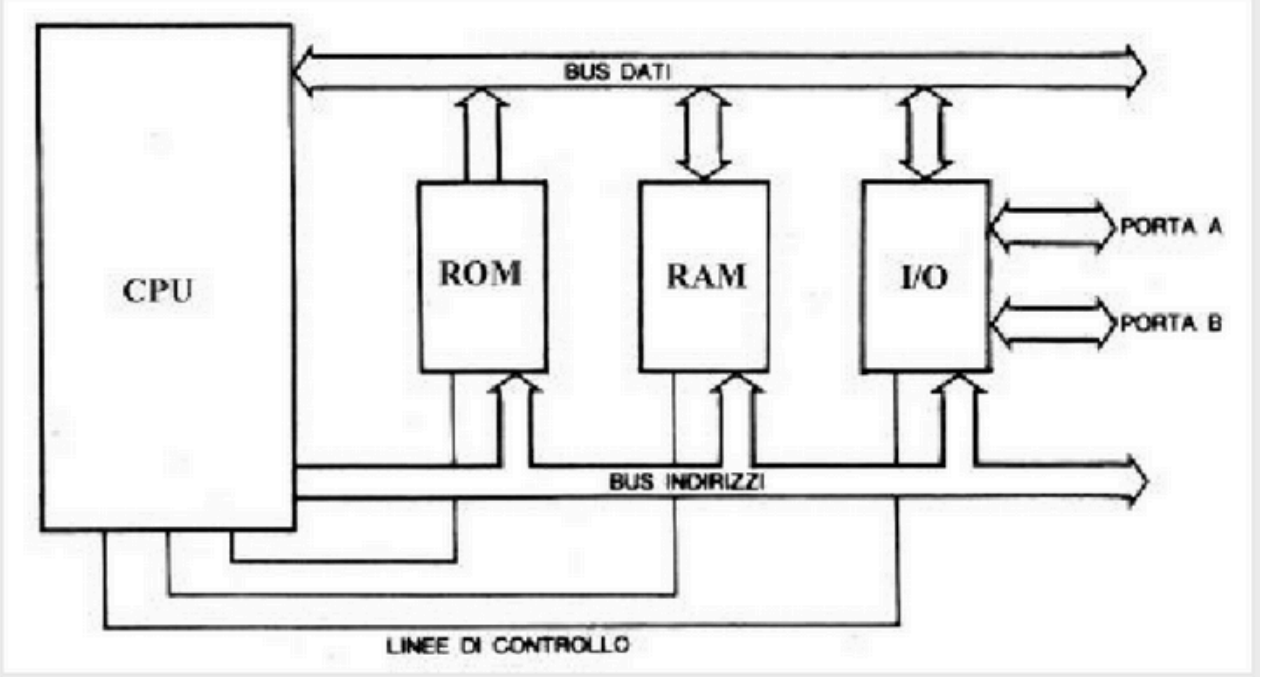
Ci sono da fare delle distinzioni nei calcolatori:

- La potenza di calcolo e capacità di memorizzazione
- Ambiente e scopo per cui sono utilizzati
  Una distinzione ulteriore è il loro scopo principale:
- Personal Computer (PC): usati come elaboratori di testo, Internet, banche dati, strumenti da ufficio, etc.
- Workstation: usati per il calcolo e la programmazione, per la grafica avanzata e la ricerca
- Mainframe: grandi aziende, banche, gestione di complesse reti di computer e di apparecchiature, applicazioni gestionali
- Network computer: computer collegati in rete condividendo dati (dischi) e altre risorse. I "terminali" sono postazioni prive di capacità di elaborazione, dotate solo di monitor e tastiera e collegate ad un computer centrale di cui sfruttano la CPU e la memoria

Architettura dei calcolatori e dispositivi I/O

L'architettura dei PC nell'hardware è un Von Neumann-like, molto simile ma non uguale, basata su questi componenti:

- CPU
- ROM
- RAM
- Dispositivi I/O:
  - Monitor
  - Hard Disk
  - Drive CD/DVD
  - Stampanti
- Bus
  Tutto questo è l'hardware, la parte più importante di tutte
  Lo schema logico di queste componenti chiamate Von-Neumann estesa è il seguente:



I bus sono bidirezionali, cioè possono avere sia un input che un output, altri sono unidirezionali, cioè utilizzabili da una sola parte

Scheda madre

La scheda madre è la componente che raccoglie la circuiteria elettronica di interfaccia fra le componenti principali e i bus di espansione.
Fisicamente possono essere montati alcuni componenti tramite degli slot dedicati come CPU, RAM, ROM(BIOS), GPU(Scheda video) e altri invece tramite cavi come Hard disk, lettore DVD, Alimentatore, USB etc.
Alcune CPU possono essere saldate per questioni di spazio ed economiche.

Bus

I bus sono piste su un circuito stampato utilizzate per trasportare segnali elettrici che rappresentano i bit. I bus hanno anche il loro clock
Esistono diversi tipi di bus:

- **Bus dati**: canale attraverso il quale "viaggiano"(0 e 1) i dati, sono usufruibili da tutti i componenti del sistema bidirezionalmente e ampi quanto l'architettura permette (32 o 64 bit)
- **Bus indirizzi**: canale attraverso quale la CPU specifica in quale indirizzo (celle di memoria RAM o periferiche di I/O) andare a scrivere/leggere dati. Questi bus sono fruibili sono il scrittura dalla CPU e in lettura dagli altri componenti monodirezionalmente
- **Bus controlli**: collegamenti tra i quali è possibile coordinare le attività del sistema, tramite questo la CPU decide quale componente deve scrivere sul bus dati in quell'esatto momento, quale deve leggere l'indirizzo sul bus indirizzi, quali celle di memoria devono scrivere e quali devono leggere, si deve assicurare che solo i componenti autorizzati possano accedere e modificare i dati.

CPU

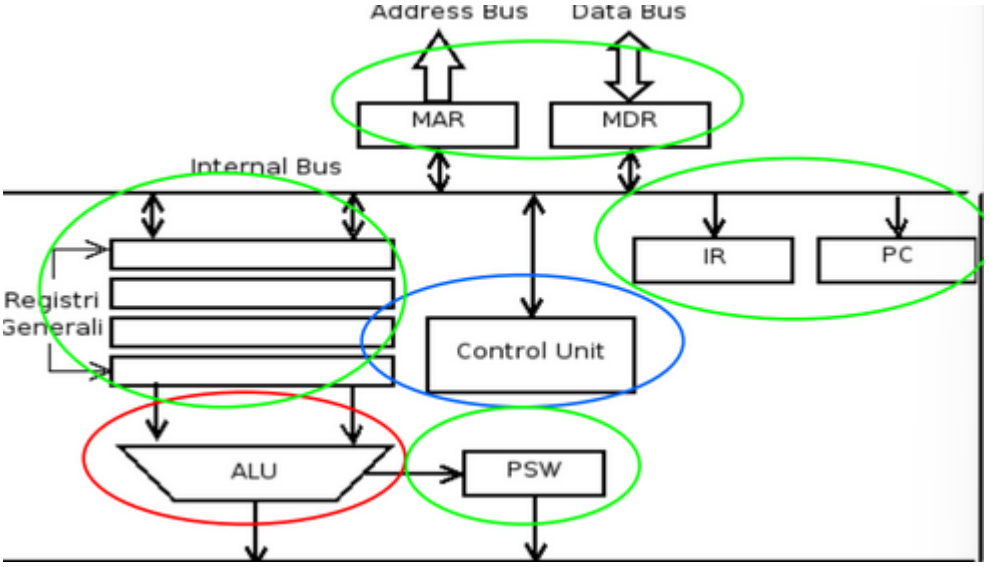
La CPU esegue le istruzioni di un programma che devono essere presenti in memoria RAM o ROM (la RAM serve per contenere i programmi che sono attualmente in esecuzione). Durante l'esecuzione del programma la CPU legge o scrive dati in memoria, non può fare altri azioni, il risultato dell'esecuzione dipende dal dato su cui opera e dallo stato interno della CPU stessa che tiene traccia delle passate operazioni
Gli elementi che costituiscono una CPU son 3:

- Registri: locazioni di memoria interne alla CPU, molto veloci(si parla sempre di nanosecondi), a cui è possibile accedere molto più rapidamente che alla memoria centrale, il loro valore complessivo dei registri costituisce lo stato in cui essa si trova in un determinato instante. I registri di una CPU sono lunghi quanto è lunga l'architettura della CPU (32 o 64 bit).
- Arithmetic Logic Unit: esegue le operazioni logiche e aritmetiche
- Control Unit (unità di controllo): "legge" (da l'impulso di lettura) dalla memoria le istruzioni e le decodifica, se occorre legge i dati per l'istruzione "esegue" l'istruzione e memorizza il risultato in memoria o registro della CPU

Altri componenti di una CPU sono:

- il PSW (Process Status Word), contiene le informazioni di stato (abilitazione/disabilitazione di interrupt, bit selezione SU o utente), alcune PSW (e processori) hanno delle istruzioni specificate per un sistema operativo
- MAR (Address Bus): indirizzo di riferimento alla memoria da cui vogliamo leggere o scrivere (ogni cella di memoria ha un suo indirizzo e al suo interno un dato, questa è la RAM)
- MDR(Data Bus): dati provenienti/da inviare alla memoria, agisce come un buffer temporaneo
- PC(Program Counter): Indirizzo nella memoria della successiva istruzione da eseguire (il termine contatore di istruzioni è un po' ambiguo visto che tale registro non effettua alcun conteggio)
- IR (Instruction Register): Istruzione corrente in esecuzione
- I/O AR (Input Output Address Register): Specifica il dispositivo di I/O con cui la CPU vuole comunicare (es: tastiera)
- I/O (Input Output Buffer Register): Contiene i dati da scambiare con il dispositivo (es: parola della tastiera)

Un buffer è un area di memorizzazione temporanea che aiuta a gestire e regolare il flusso di dati tra due dispositivi o processi che operano a velocità diverse



La CPU esegue le istruzioni in maniera sequenziale, indipendentemente dall'architettura, grazie all'PC.
Una CPU ha diversi modi di misurare le prestazioni:

- Velocità del clock (Ghz), quante volte una CPU effettua un ciclo (un impulso) al secondo
- Istruzioni al secondo (MIPS), le istruzioni che un processore può effettuare in un secondo

Registri del processore

I registri quindi si dividono in due grandi categorie:

- Visibili all'utente (L'istruzione di un programma)
  - Permettono di minimizzare i riferimenti alla memoria principale, al posto di tenerli nella memoria stanno nella CPU
  - Generalmente sono disponibili per categorie come:
    - Dati
      - General purpose: utilizzato per memorizzare temporaneamente dati e/o informazioni di controllo di vario genere
      - Dedicati: dedicati ad un unico scopo
    - Indirizzi(index register, segment pointer, stack pointer)
    - Codice di condizione (flag - parzialmente visibili)
- Di stato e controllo
  - Memorizzano l'esito delle operazioni del processore
  - Solo ad alcuni programmi è possibile accedervi per istruzioni eseguite in modalità di controllo o di super utente (sistema operativo)

I contenuti di un registro son sempre dinamici, mai statici, poiché le informazioni sono sempre in cambiamento con il loro contenuto.
Una CPU può circotuicitare, in modo da utilizzare l'esito dell'operazione precedente senza dover prenderla di nuovo dalla memoria, risparmiando questa ultima e anche tempo.

Processore

I processori possono implementare al loro interno più unità di esecuzione per eseguire più operazioni contemporaneamente, aumentando le prestazioni delle CPU ma complicando l'esecuzione, poiché per eseguire in modo efficiente più istruzioni si devono coordinare
La possibilità di disporre di più CPU permette al sistema operativo di far eseguire in parallelo più programmi aumentando notevolmente le prestazioni (multi threading)
Un giga transfer (GT/s) è un'unità di misura utilizzata per indicare il numero di operazioni di trasferimento dati che avvengono in un secondo, espressa in miliardi di trasferimenti al secondo. È comunemente usato per descrivere la velocità di trasferimento dati nei sistemi informatici, come nei bus di sistema o nelle interfacce di rete.

Istruzioni

Un programma è una sequenza di istruzioni, un istruzione è un comando che eventualmente opera sui dati
Ci sono diversi tipi di istruzioni come:

- Trasferimento
  - Processore - memoria (e viceversa)
  - Processore - modulo di I/O (e viceversa)
- Elaborazioni di dati (operazioni logico-aritmetiche)
- Controllo (modifica della sequenza di esecuzione)
  Un set di istruzioni dipende dal processore (come un I3 vs I7,AMD vs Intel)

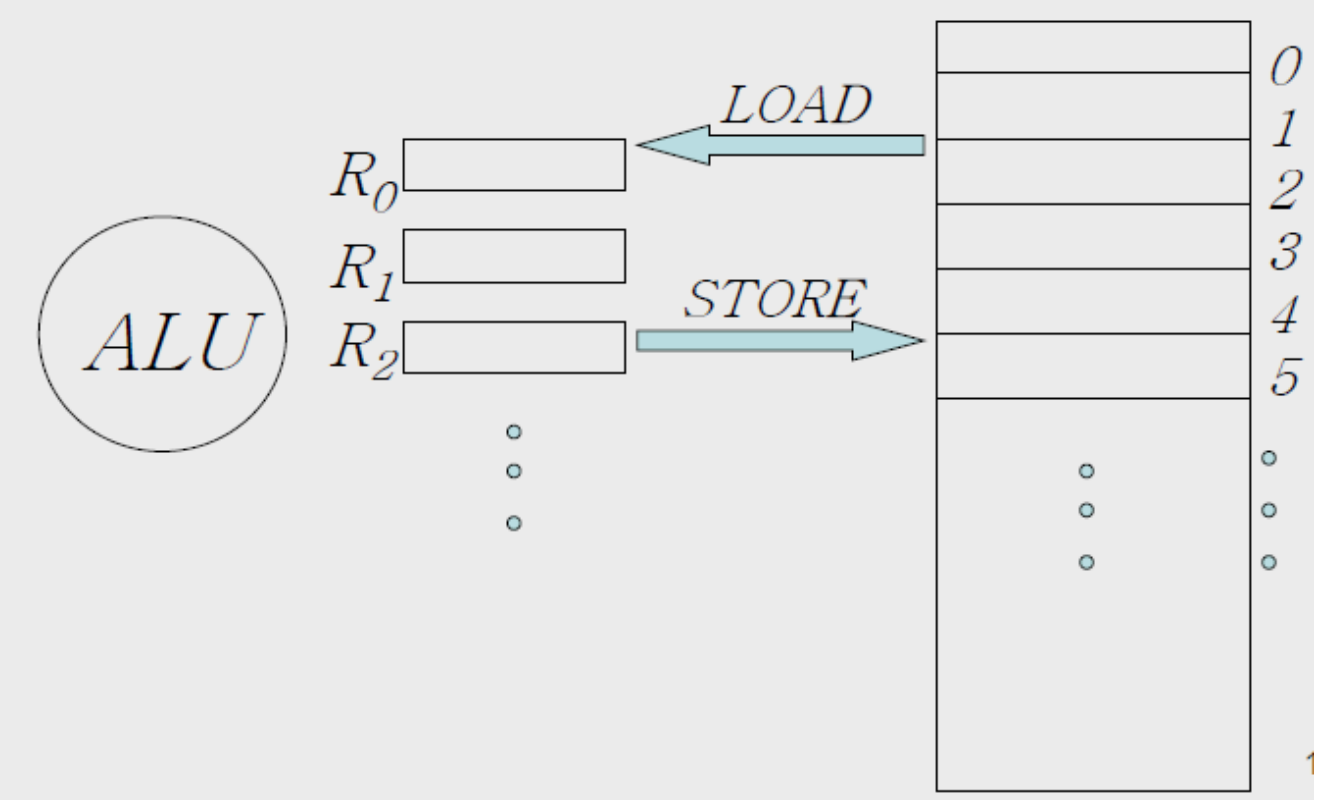
Indirizzamento

Molte istruzioni contengono operandi e si pone il problema di come specificare la posizione, per questo esiste l'indirizzamento e le sue modalità:

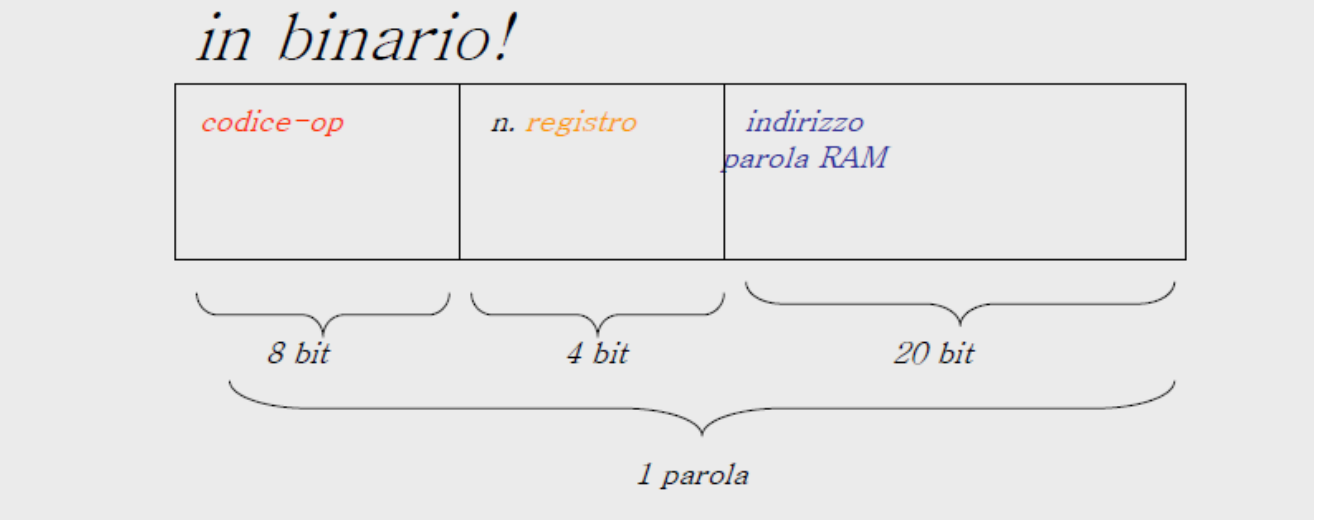
- **Indirizzamento immediato**: Contenere nel campo riservato all'indirizzo l'operando stesso piuttosto che un indirizzo o qualunque altra informazione sulla posizione. Un operando così si dice immediato. L'entità del valore è limitata dalla dimensione del campo
- **Indirizzamento diretto**: Dare l'indirizzo completo dell'operando in memoria. L'indirizzamento diretto serve solo ad accedere a variabili globali il cui indirizzo è noto in fase di compilazione
- **Indirizzamento a registro**: Analogo all'indirizzamento diretto ma si specifica un registro piuttosto di una locazione di memoria, ed è la modalità più utilizzata nei computer dato che i registri son veloci in accesso e hanno indirizzi brevi
- **Indirizzamento a registro indiretto**: L'operando proviene o è destinato alla memoria, ma il suo indirizzo non è incorporato nell'istruzione, ma bensì in un registro, prendendo il nome di puntatore

- **Indirizzamento indicizzato**: L'indirizzamento in memoria che si ottiene specificando un registro (in via implicita o esplicita) più uno spiazzamento costante
- **Indirizzamento indicizzato esteso**: L'indirizzo di memoria è calcolato sommando tra di loro il contenuto di due registri più un offset(numero di byte da aggiungere a un indirizzo di base per ottenere un indirizzo specifico), un registro funge da base e l'altro da indice
- **Indirizzamento a stack**:modalità di accesso alla memoria che utilizza una struttura a pila (LIFO), gestendo i dati con operazioni di push e pop tramite un puntatore allo stack, usata spesso per le chiamate di funzione e variabili locali.

Esempio di istruzioni

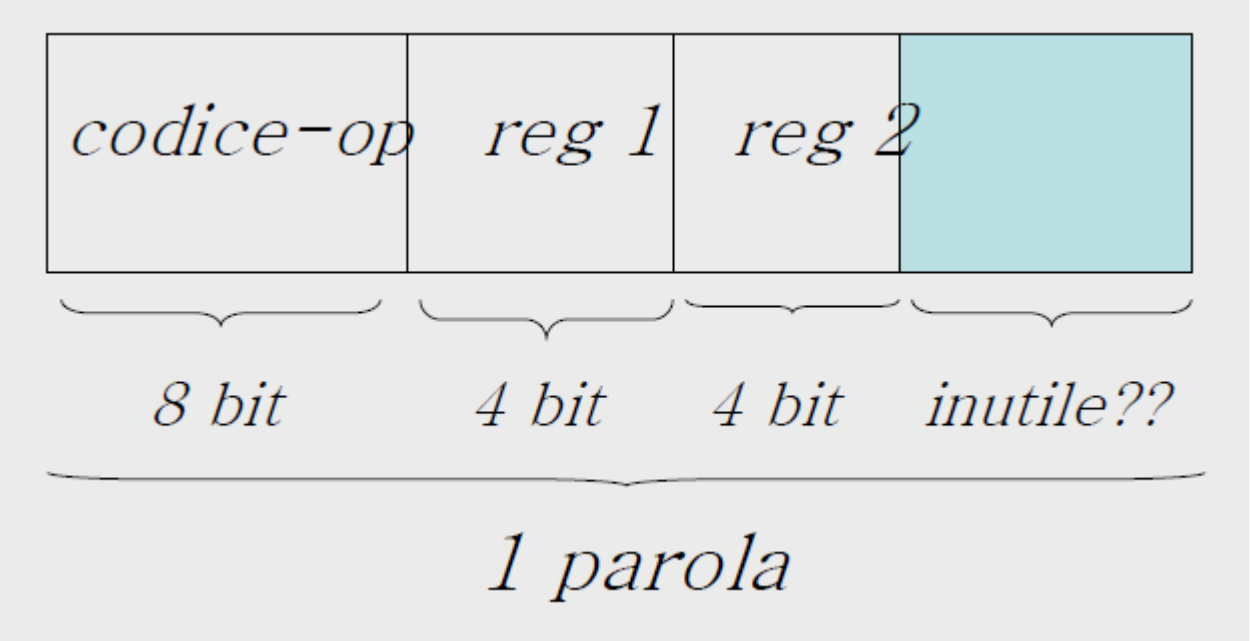


In un operazione di load e store per esempio l'ALU non viene usata, vengono usate solo RAM,IR e PC e il resto della componentistica ma non l'ALU, visto che non si sta effettuando un operazione logica aritmetica.

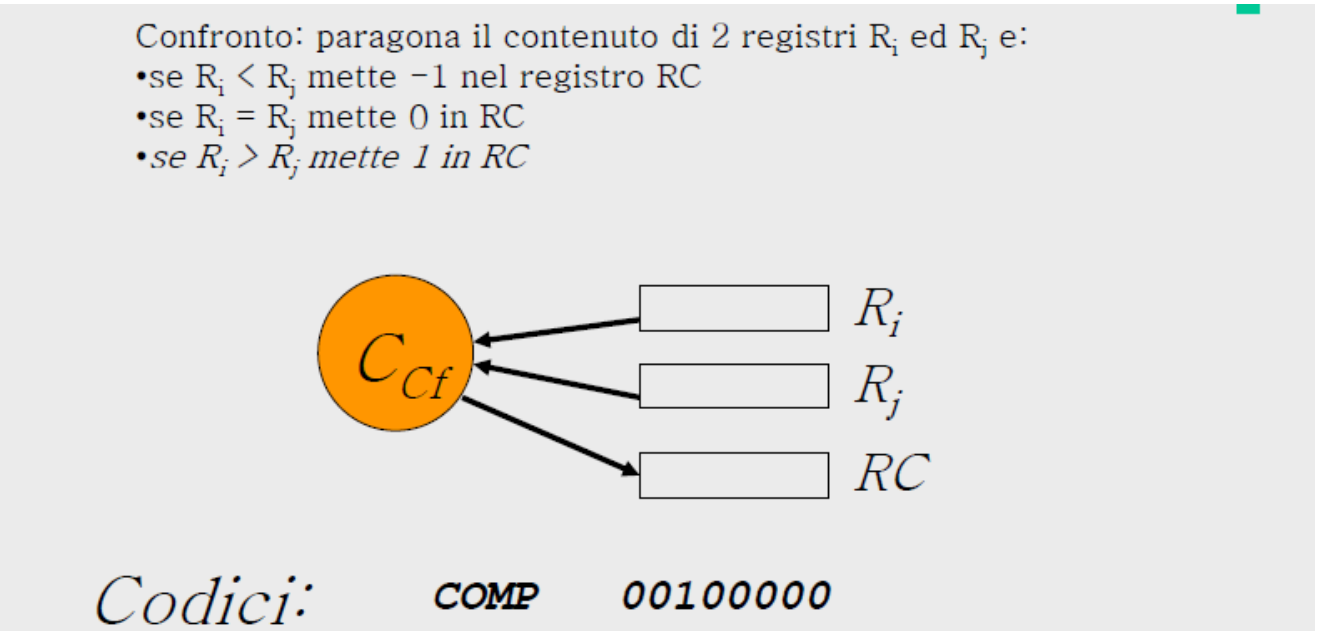


I registri usati possono essere massimo 16 nelle architettura moderne, perché alcuni sono

riservati e inutilizzabili ed ecco perché si usano unicamente 4 bit
In un architettura del genere, dove questi 32 bit formano una parola, per indirizzamento diretto alla memoria, si possono avere 2^{20} celle di memoria · 32 bit di area al suo interno (4 byte), questo calcolo corrisponde a 4 megabyte



Qui con questo registro vuoto si può utilizzare quello spazio apposta per controllo



CISC e RISC

Come si fa a compilare un linguaggio di alto livello affinché possa essere eseguito in maniera efficiente nel calcolatore?
Ci sono due scelte progettuali:

- Macchine RISC(Reduced Instruction Set Compiler): Architettura dove ci sono poche istruzioni per poi adeguarlo a quelli dell'utente
- Macchine CISC(Complex Instruction Set Compiler): Architettura molto complessa che includa un numero elevato di istruzioni e modi di indirizzamento e che includa anche istruzioni molto vicine a quelle presenti nel linguaggio ad alto livello

Queste sono 4 architetture, le due a sinistra sono CISC e quelle a destra sono RISC

VAX 11/780 Nr. of instructions: 303 Instruction format: not fixed Addressing modes: 22 Number of general purpose registers: 16	Sun SPARC Nr. of instructions: 52 Instruction format: fixed Addressing modes: 2 Number of general purpose registers: up to 520
Pentium Nr. of instructions: 235 Instruction format: not fixed Addressing modes: 11 Number of general purpose registers: 8	PowerPC Nr. of instructions: 206 Instruction format: not fixed (but small differences) Addressing modes: 2 Number of general purpose registers: 32

Ma quindi quale è la differenza tra una macchina RISC e una CISC? La differenza sostanziale sta in alcuni principi di progettazione, chiamati **principi di progettazione RISC**:

- **Tutte le istruzioni sono eseguite direttamente dall'hardware**, e non sono interpretate tramite microistruzioni, questa eliminazione di un livello di interpretazione garantisce una velocità maggiore rispetto all'architettura CISC che per eseguire operazioni più complesse le suddivide in microistruzioni (anche se maggior parte di queste son utilizzate in minor frequenza),ma meno retrocompatibilità
- **Massimizzare la frequenza di emissione**, con alcuni trucchi. I calcolatori moderni hanno bisogno di eseguire più azioni possibili in un secondo, qui il parallelismo torna utile proprio per poter eseguire più istruzioni in un intervallo minore di tempo.
- **Le istruzioni devono essere facili da decodificare**, la decodifica stessa è un limite poiché ogni istruzione deve essere decodificata in modo da dover determinare le risorse necessarie. Il modo per ottimizzare le istruzioni è renderle regolari, di lunghezza fissa e con pochi campi, in sostanza meno formati d'istruzioni ci sono meglio è.
- **Solo le istruzioni Load e Store fanno riferimento alla memoria**, uno dei modi più semplici per spezzare le operazioni in passi separati è quello di richiedere che gli operandi per maggior parte di queste operazioni vengano prelevati dai registri e memorizzati al loro interno. L'operazione dello spostamento degli operandi dalla memoria ai registri può essere compiuta separatamente tramite altre istruzioni, ma l'accesso alla memoria può richiedere un tempo considerevole e non prevedibile, per poter risolvere questo problema le operazioni, a patto che non facciano null'altro se non muovere operandi tra registri e memoria, possono essere sovrapposti all'esecuzione di altre istruzioni, in conclusione unicamente le operazioni LOAD e STORE dovrebbero far riferimento alla memoria, le altre devono operare unicamente sui registri
- **Molti registri disponibili**, sempre dato dalla questione che accedere alla memoria è molto lento occorre prevedere abbastanza registri (almeno 32) in modo da poterli tenere fin quando necessario, altrimenti si incorrerebbe in una inefficienza data dal fatto che, non avendo abbastanza registri, bisognerebbe scaricare in memoria tutti i valori per poi ricaricarli.

Esecuzione dell'istruzione

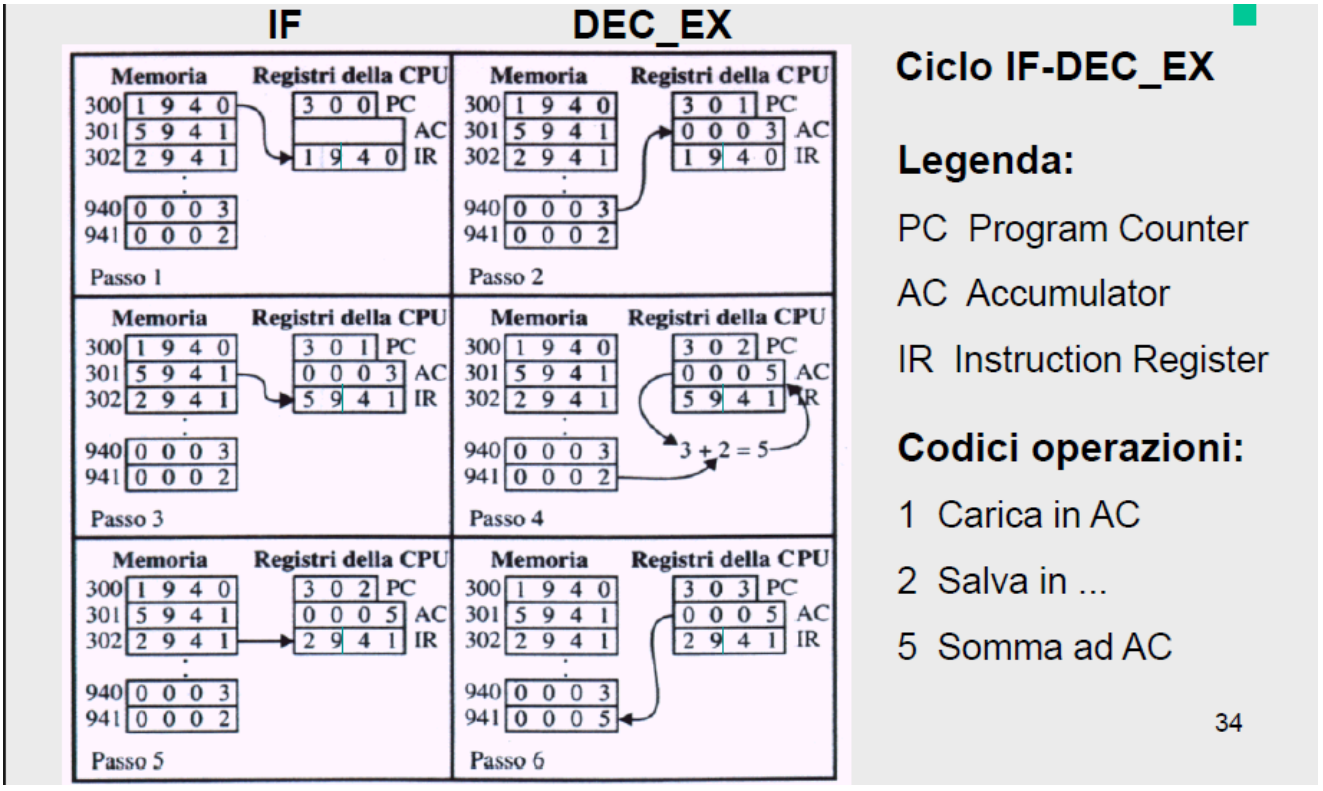
La CPU esegue ogni istruzione compiendo una serie di piccoli passi:

1. Prelevare la successiva istruzione dalla memoria per portarla nell'IR
2. Modificare il PC per farlo puntare all'istruzione seguente
3. Determinare il tipo dell'istruzione appena prelevata
4. Se l'istruzione usa una parola in memoria, determinare dove si trova
5. Se necessario, prelevare la parola per portarla in un registro della CPU
6. Eseguire l'istruzione
7. Tornare al punto 1
   Questa sequenza di passi viene chiamata **ciclo esecutivo delle istruzioni** o ciclo di **prelievo-decodifica-esecuzione**
   Gli stessi passi li facciamo comunque su un istruzione diversa

In modo più formale abbiamo queste vere istruzioni:

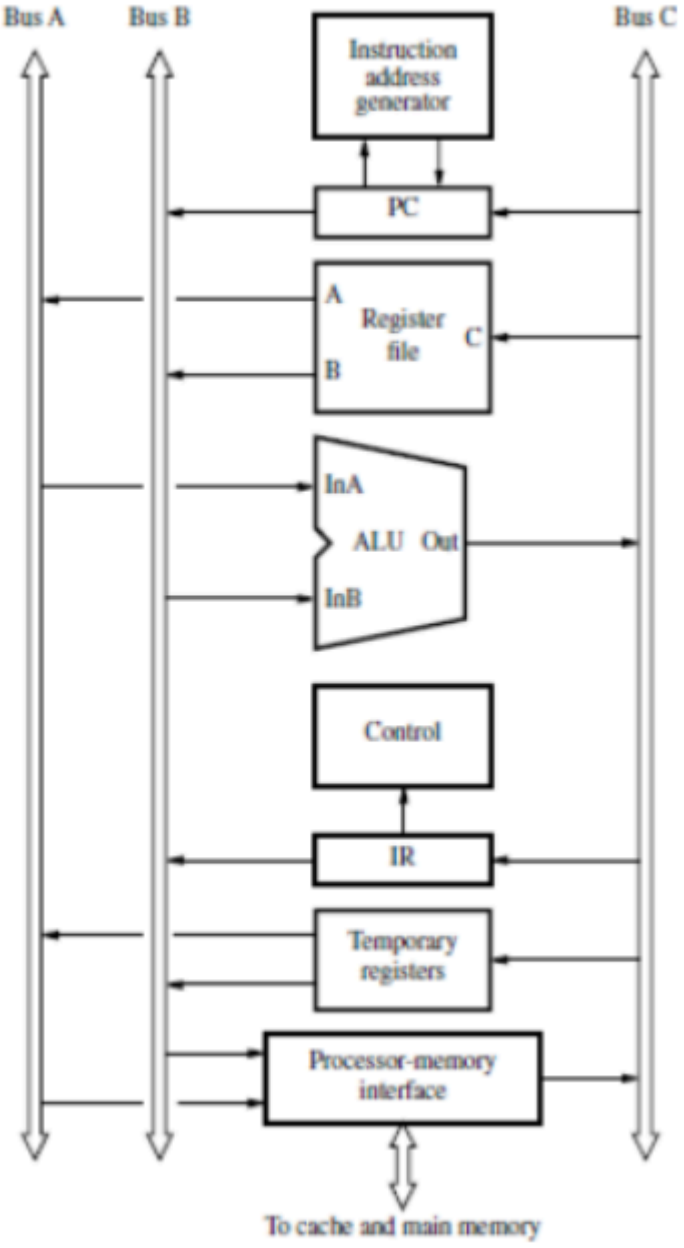
1. **Instruction Fetch**(Acquisizione dell'istruzione): il processore preleva l'istruzione dalla memoria, specificata dal PC e la carica nell'IR, per poi dover aumentare il valore del PC
2. **Instruction Decode**(Decodifica dell'istruzione): dall'istruzione prelevata viene determinata quale è l'operazione che deve essere eseguita e i gli operandi mediante la conoscenza dei codici operativi (Control Unit)
3. **Instruction Execute**(Esecuzione dell'istruzione): viene eseguita l'istruzione

Questa è un esempio di esecuzione in RISC di una ipotetica macchina ridotta all'osso



In una macchina CISC dovremmo aggiungere questi passaggi:

- **Memory Access**: nel caso in cui un istruzione richieda un accesso alla memoria, questa fase sostituisce o segue la precedente
- **Write Back**: scrittura del risultato in memoria
  Questo è un esempio di CISC dove si utilizzano tutti i componenti



ROM (Read Only Memory)

La ROM è una memoria unicamente in lettura, non sono possibili operazioni di scrittura, questa memoria viene scritta unicamente una volta dal produttore, non perde mai la memoria anche se l'alimentazione non è presente e nei computer viene tipicamente utilizzata per contenere le istruzioni di avvio del programma di avviamento (bootstrap), chiamato il BIOS su una scheda madre
Il BIOS è una sequenza di istruzioni di avvio eseguita automaticamente ad ogni accensione del PC:

1. Attivazione dell'hardware installato e test di funzionamento del sistema (Il processore esegue per la prima volta all'accensione prima le istruzioni della ROM, ancor prima

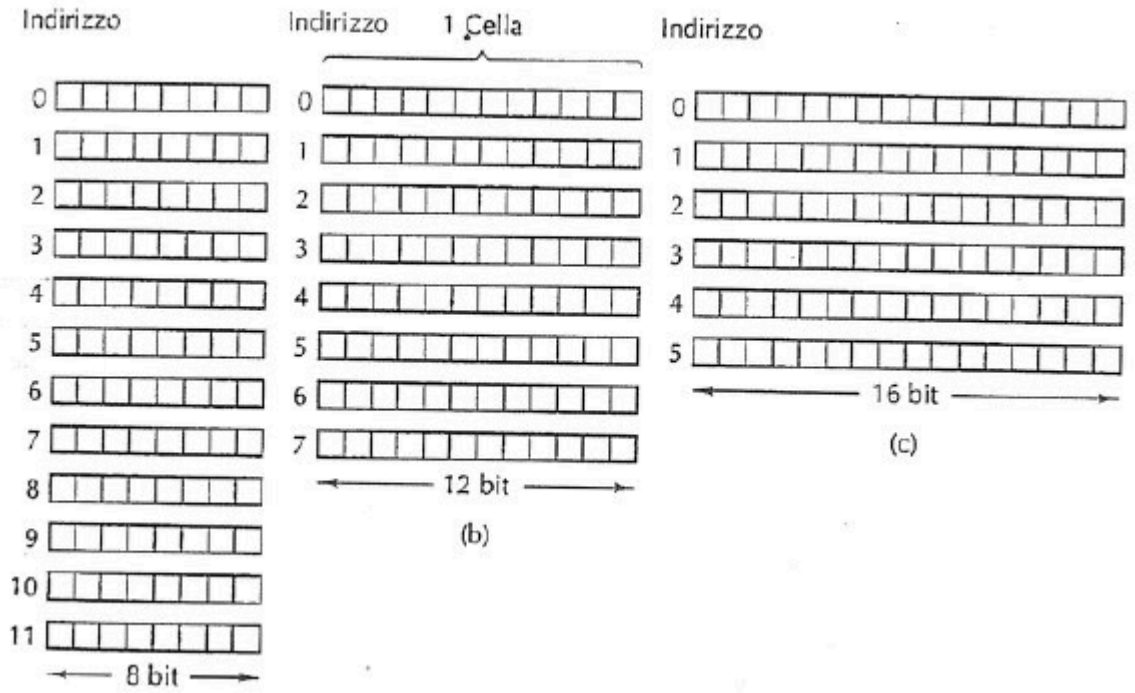
quello della RAM visto che questa è vuota)
2. Verifica della presenza del sistema operativo (porta il micro kernel nella RAM)
3. Avvio del primo processo (Sui sistemi UNIX è init)

Queste ROM sono utilizzate anche in altri ambiti come quelle automobilistiche e elettrodomestiche;
Dopo le ROM vennero inventate le **PROM** (Programmable ROM), differisce dalla ROM per il fatto che può essere programmata una volta sul posto, in modo da non essere programmata ad ordine da parte del produttore;
Dopo ancora vennero inventate le **EPROM** (Erasable PROM), in cui i campi non vengono solo programmati ma anche cancellati tramite ultravioletto;
Infine arrivarono le **EEPROM** (Eletrical EPROM), cancellabili tramite impulsi elettrici piuttosto che raggi ultravioletti;
Da queste derivano poi le **memorie flash**, cancellabili a blocchi e utilizzate per altri scopi come la memorizzazione di dati di ogni giorno

RAM (Random Access Memory)

La RAM è una memoria dove vengono conservate le istruzioni del programma attualmente in esecuzione e i suoi dati, durante la loro esecuzione i programmi e i dati devono trovarsi almeno parzialmente nella memoria centrale (Memoria centrale=RAM).
È una memoria volatile, può contenere dati e istruzioni unicamente se è alimentata, una volta spenta si cancella e non contiene più nulla, si misura in GigaByte (più grande è, più programmi possono essere contenuti) e la velocità si misura in frequenza di lavoro(la sua unità di misura è MT/s, milioni di trasferimenti al secondo), più è alta la frequenza e più velocemente sarà accessibile.
Le memorie sono costituite da un certo numero di celle (o locazioni), ciascuna delle quale si può memorizzare delle informazioni, ogni cella ha un numero, chiamato indirizzo, attraverso il quale il programma può riferirsi ad essa.
Se una memoria ha n celle i suoi indirizzi varieranno da 0 a $n-1$. Tutte le celle di memoria contengono lo stesso numero di bit, se una cella contiene k bit può contenere 2^k combinazioni diverse di bit. (Questo parametro della grandezza delle celle viene stabilito sul produttore)

Questo è un esempio di una parola a 96 bit



Altri tipi di RAM

Prima delle RAM esisteva un altro tipo di memorie, quelle ad accesso sequenziale, bisogna passare in rassegna tutte le celle fino a raggiungere quella che volevi, con la RAM si passa direttamente alla cella che si vuole.
Le RAM si distaccano in poi due tipi:

- RAM statiche (Static RAM): composte da circuiti similia Flip-Flop D, hanno la proprietà di mantenere il proprio contenuto e sono molto veloci, tipicamente utilizzate come cache di secondo livello
- RAM dinamiche (Dynamic RAM): composte da un array di celle con un transistor e un condensatore che può essere utilizzato per memorizzare 0 o 1, molto più lente delle SRAM ma di dimensioni maggiori ed elevata densità, utilizzate quindi per le memoria centrale del computer
  Da queste poi è nato un ibrido, le SDRAM (Synchronus DRAM, DRAM sincrona), da una parte statica e da una parte dinamica.

Formati fisici

In formato fisico le RAM si dividono in due formati:

- SIMM(Single Inline Memory Module), connettore a 72 pin con trasferimento di 32 bit per ciclo
- DIMM(Dual Inline Memory Module), connettore a 84 pin con trasferimento di 64 bit per ciclo

Ogni quanto avviene un operazione di lettura e scrittura? Una qualunque istruzione in esecuzione richiede accesso in memoria poiché deve essere prelevata, su una macchina CISC arrivano le 3 istruzioni precedentemente dette (fase di Memory Access e Write Back)

In un moderno sistema dotato di pipeline vi è un avvio di istruzione ad ogni colpo di clock

La RAM è collegata alla CPU tramite l'FSB (Front Side Bus), trasporta i dati tra la CPU e i vari dispositivi veloci (come la stessa RAM) e il suo throghput (o la sua banda) viene determinato moltiplicando:

- Byte delle word del processore
- Frequenza del clock del bus
- Numero di data transfer del BUS ad ogni ciclo
  Per esempio in un sistema con un processore a 32 bit (4 byte) con un FSB a 100mhz e 4 trasferimenti a ciclo otteniamo 1600 Megabyte al secondo (MB/s)
  Se un processore viene accoppiato con una RAM lenta il processore verrà anche impattato da quest'ultima

La memoria viene quindi valutata attraverso i seguenti parametri:

- Dimensione
- Velocità
- Costo
  Questi 3 parametri sono ottimizzabili a 2 a 2, non è possibile avere tutte e i parametri al loro massimo

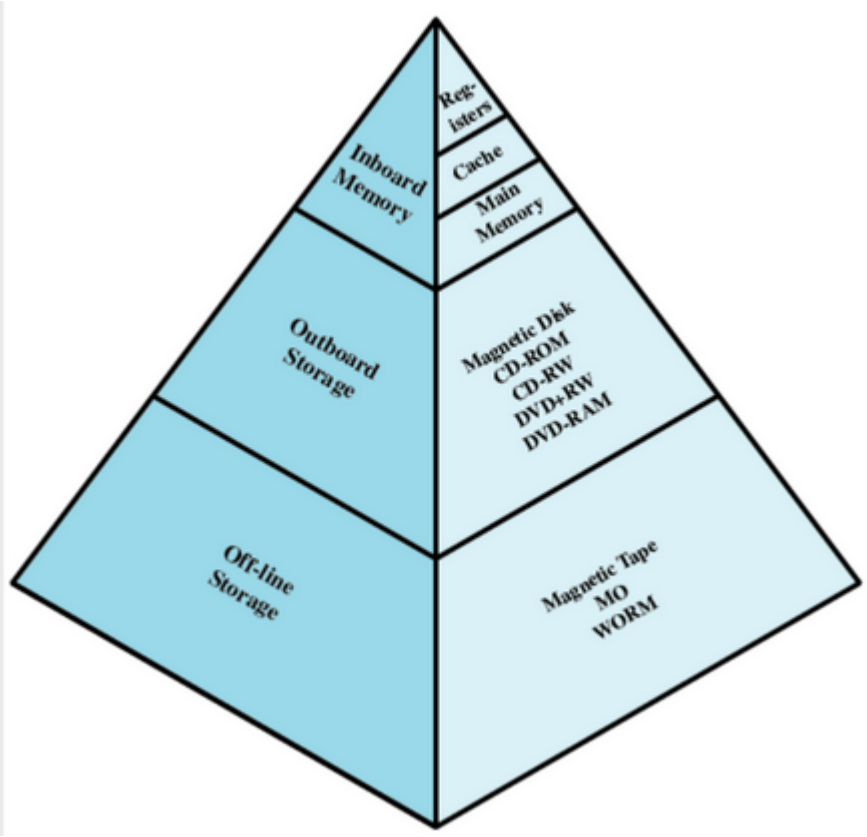
Gerarchie delle memorie

È una soluzione alla inconciliabilità dei 3 parametri e secondo la gerarchia:

- Diminuisce il costo per bit
- Aumenta la capacità
- Aumenta il tempo di accesso
- Diminuisce la frequenza di accesso del processore

La gerarchia prevede nella parte alta i registri della CPU (che si possono accedere alla stessa velocità della CPU), poi si trova la cache (che varia da diversi KB a diversi MB), poi si trova la memoria centrale, infine dischi magnetici e nastri magnetici.
Muovendosi verso il basso della gerarchia aumentano tre parametri chiavi, il tempo di accesso diventa via via più grande, la capacità di memorizzazione aumenta man mano che

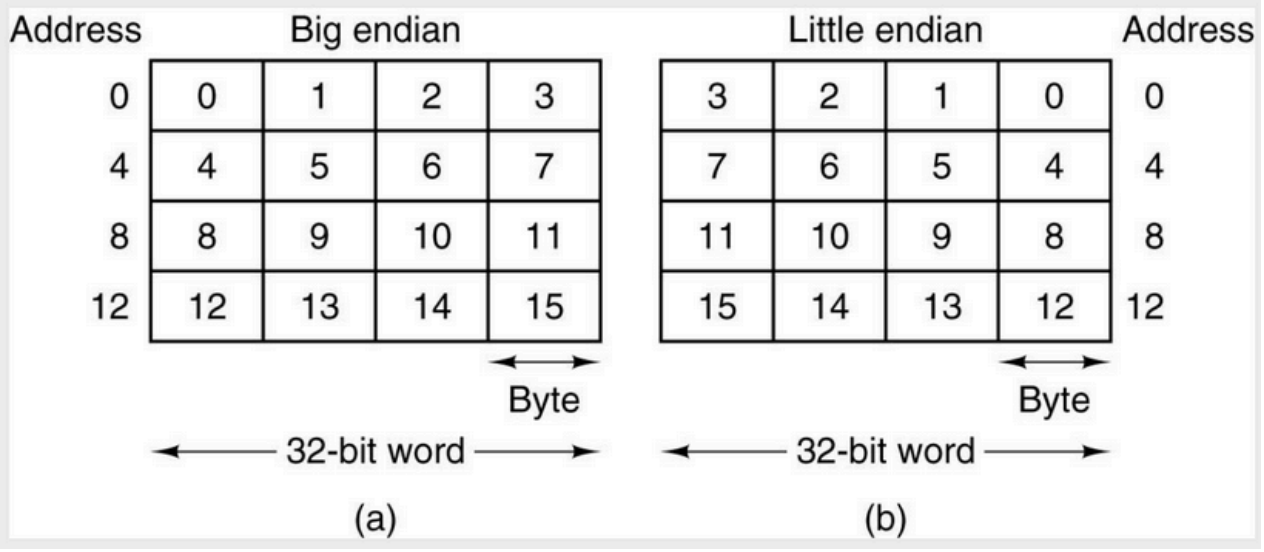
si scende e infine diminuiscono scendendo i costi unitari



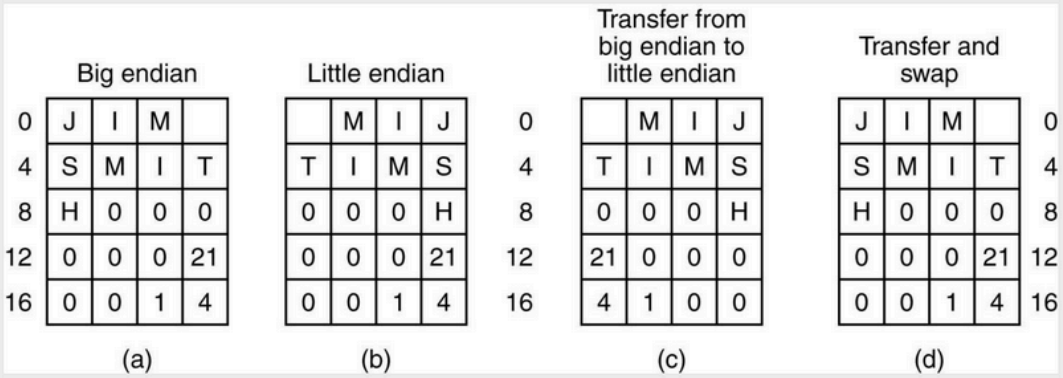
Ordinamento dei Byte

All'interno di una parola di memoria i byte possono essere numerati in due modi:

- Da sinistra a destra (Big endian, dal byte più significativo al meno significativo)
- Da destra a sinistra (Little endian, dal byte meno significativo al più significativo)



Trasferimento dei Byte



(a) Macchina “big endian”.
(b) Macchina “little endian”.
(c) RIsultati del trasferimento dati da “big endian” a “little endian”.
(d) Byte-swapping di (c).

Memoria cache

La memoria cache contiene i dati di utilizzo più ricorrente in modo che il processore non li debba cercare nelle aree di memoria centrale, è molto veloce e serve a compensare la differenza tra velocità d'accesso e trasferimento dei dati tra CPU e RAM
Osservazione: il suo buon funzionamento della cache deriva dalla capacità del sistema nel mantenere in essa le informazioni necessarie
Nella cache occorre una politica di gestione per poter essere utile al fine di velocizzare il sistema, ma come fa la cache a sapere nel futuro cosa potrebbe servire esattamente?
Ci sono due principi nella memoria cache:

- **Località spaziale**: se in un certo istante viene referenziato un indirizzo di memoria è altamente probabile che in istanti immediatamente successivi possano venire referenziati indirizzi vicini. In un istruzione per esempio viene aumentato il PC, per i dati nelle aree vicine saranno presenti
- **Località temporale**: in un certo istante (T) viene referenziato un indirizzo di memoria, è altamente probabile che in istanti immediatamente successivi lo stesso indirizzo possa essere nuovamente referenziato, per esempio in un ciclo in un linguaggio di programmazione nelle istruzioni
  I riferimenti alla memoria fatti in un breve intervallo di tempo tendono ad utilizzare solo una piccola parte della memoria locale

Motivazioni della cache

La cache permette di poter accelerare istruzioni come:

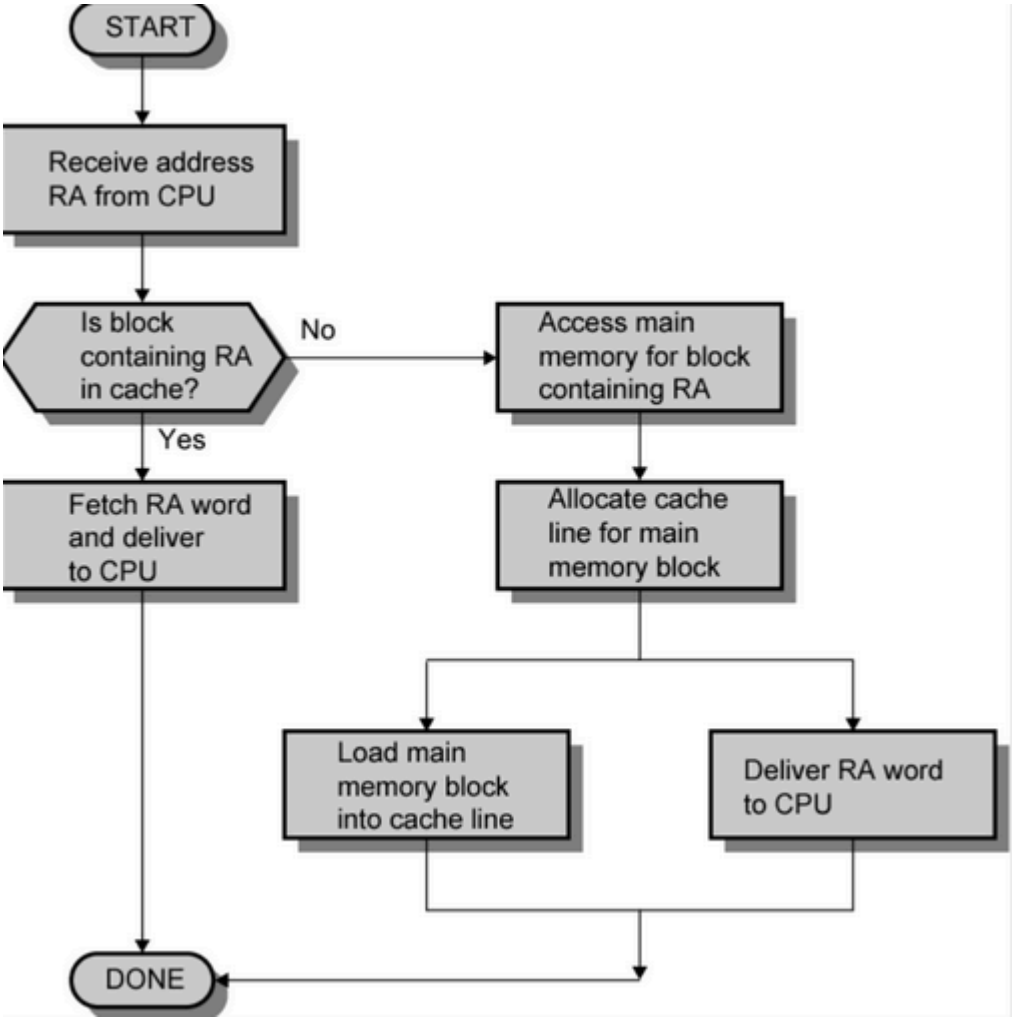
- Cicli di istruzione del processore che accedono almeno una volta alla memoria principale (prelievo dell'istruzione IF)

E poi altre motivazioni tecniche:

- La velocità del processore è limitata dalla velocità della memoria
- La velocità della memoria centrale è minore di quella del processore
  Sfruttando il principio della località si inserisce quindi una memoria piccola e veloce fra processore e memoria centrale, con l'obbiettivo di fornire una memoria con velocità prossima a quella del processore, disporre di una quantità di spazio sufficiente per non rallentare il processore e contenere i costi delle memorie del sistema
  La cache però deve essere abbastanza grande per poter contenere abbastanza informazioni da non rallentare il processore

Lettura della cache

Questo è un caso di lettura dalla cache:



Le istruzioni che vengono eseguite in ordine sono:

- La cache riceve un indirizzo di memoria dalla CPU
- Questo indirizzo che la CPU chiede è presente al mio interno?
  - Se è presente, restituisce la parola letta, lavorando alla velocità della cache (questo è il ramo HIT)
  - Se non è presente (ramo MISS)
  - Si accede alla memoria principale per recuperarlo
  - Siccome la cache a regime è sempre piena viene messa in una linea d'attesa

- Viene caricato dalla memoria principale nella linea di cache
- Viene consegnato il registro

Nel caso del MISS noi lavoriamo ad una velocità peggiore di non avere completamente una cache e prelevarlo direttamente dalla RAM, aumentano i passaggi rispetto al caso senza cache

C'è una formula specifica che permette di calcolare il vantaggio di tempo che la cache ci da in lettura e scrittura per singola istruzione:

$$\frac{t_{cache}}{t_{memory}} = \frac{(c+m)+(k-1)c}{km} = \frac{c+m}{km} + \frac{(k-1)}{k}\frac{c}{m}$$

Se il risultato è maggiore di 1 la cache sta rallentando il sistema, se 1 significa che non ci sono vantaggi veri e propri, se minore significa che la cache sta facendo il suo lavoro

C'è un modo più semplice per poter eseguire questa operazione, in modo da calcolare il tempo di accesso medio per tutte le istruzioni

Nel caso più generale, posto:

h=*hit ratio* (frequenza di successi nell'accesso alla cache)

((1-h)=*miss ratio*)

Si ottiene che il tempo medio di accesso è pari a:

$$=h\cdot c+(1-h)\cdot(c+m)=$$

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

$$=c+(1-h)m$$

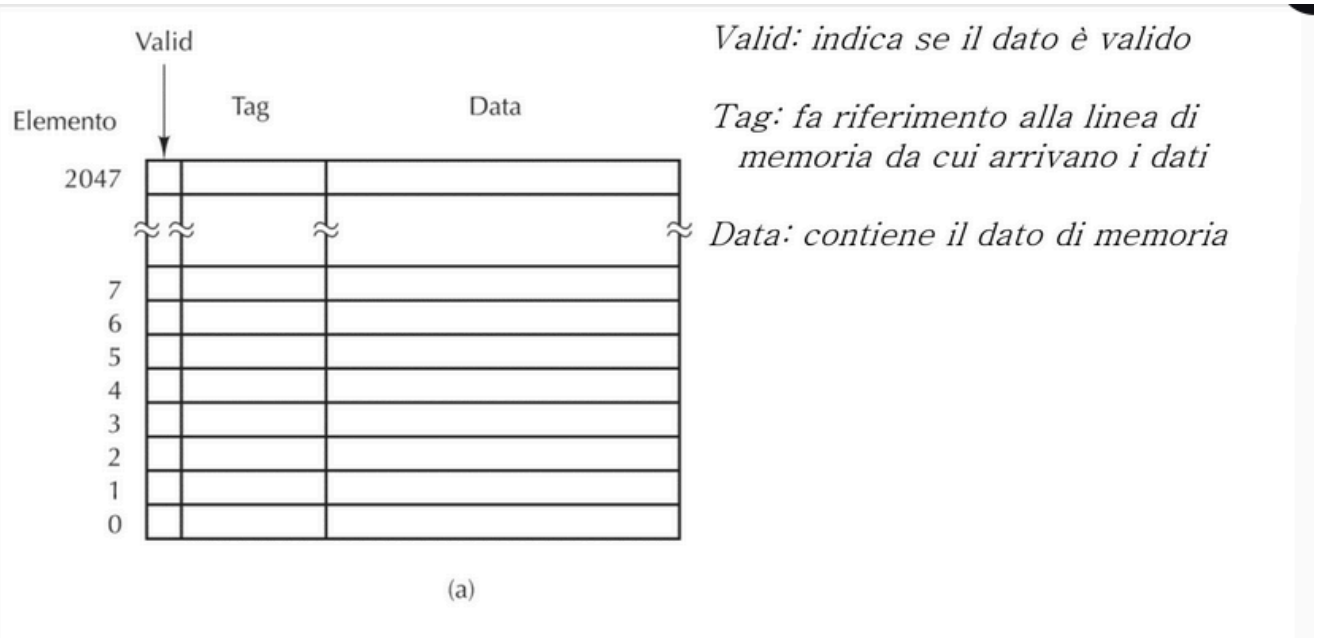
Dire che c'è un hit ratio pari a 1 significa che c'è il dato che stai cercando sempre nella cache, 0 è il suo contrario.
L'hit ratio principalmente dovrebbe essere circa dell'1% o meno per essere più efficiente della memoria centrale

Gestione della cache

Posizionamento di un blocco in cache

Indirizzamento diretto

Un blocco può essere messo in un solo punto della cache, per poterne ottenere l'indirizzo del blocco si usa il modulo (indirizzo di blocco) % (numero di blocchi nella cache)



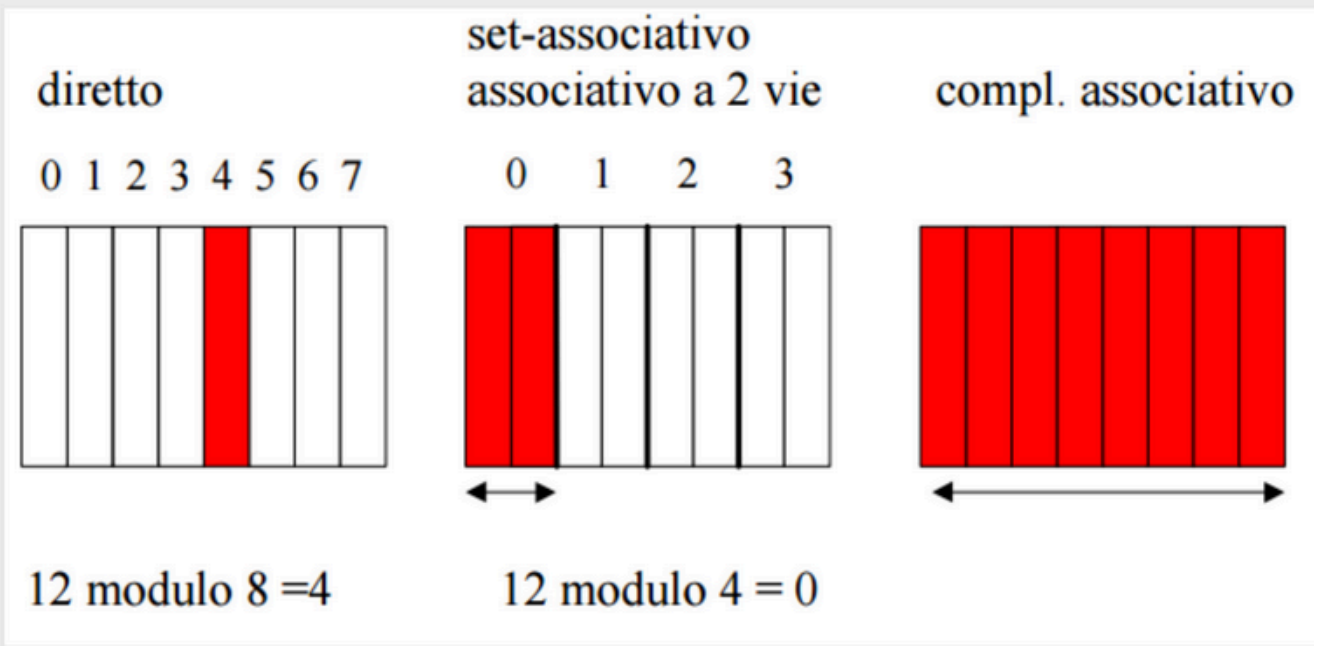
Full associative

Un blocco può essere posto ovunque nella cache, è facilissimo da implementare ma serve un altra tabella (con un altro accesso) per capire dove sono posti i vari blocchi di memoria nella cache

Set associative

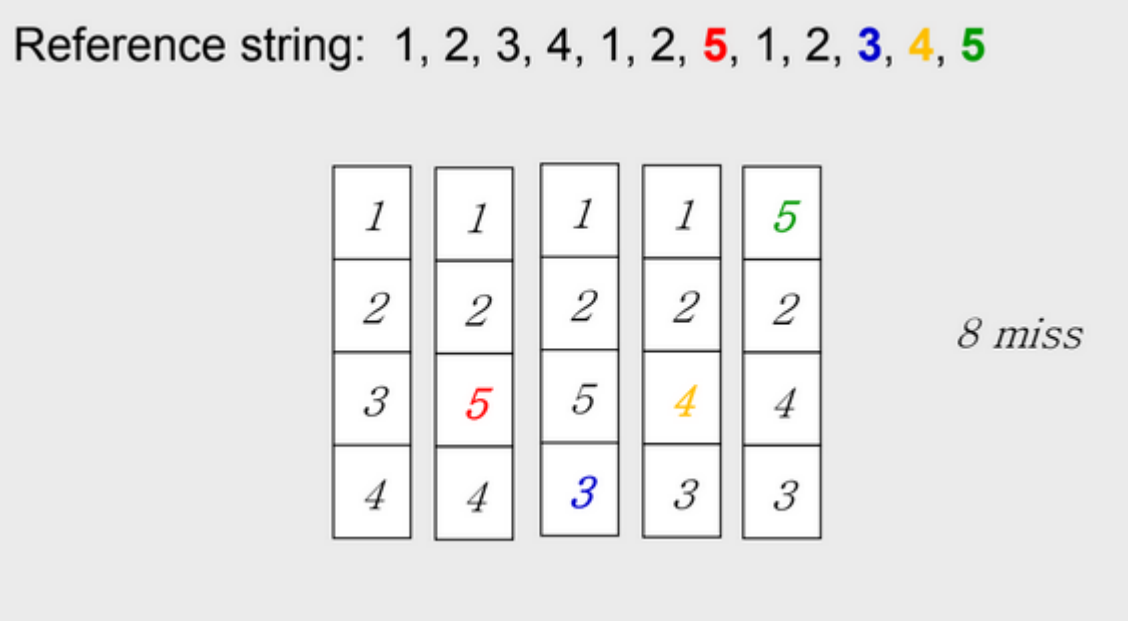
Un blocco può essere posto in un insieme ridotto di posizioni nella cache
L'insieme è formato da un gruppo di due o più blocchi di cache, un blocco prima viene messo in corrispondenza di un insieme e poi posto in qualunque posizione dell'insieme;
L'insieme viene scelto con la regola del modulo detta precedentemente.
Se ci sono n blocchi di un insieme il posizionamento definito come **set-associativo a n vie**

Immaginiamo un blocco di indirizzo 12 e una cache che può contenere 8 blocchi



Algoritmi di replacement: Least Recently Used, LRU

Questo algoritmo prova ad approssimare il futuro dell'istante dopo T (T+1) e la sua istruzione prevedendo che sia riguardante sempre l'istruzione precedente;
Nel formale quindi si sostituiscono i blocchi meno recentemente utilizzati, con l'ipotesi che il blocco meno utilizzato recentemente è quello con la minor probabilità di essere referenziato, approssimando il futuro prossimo al passato recente
Questo è un esempio dove noi sappiamo già la sequenza (futuro) (le colonne vanno lette in verticale)



Algoritmo di replacement: Random

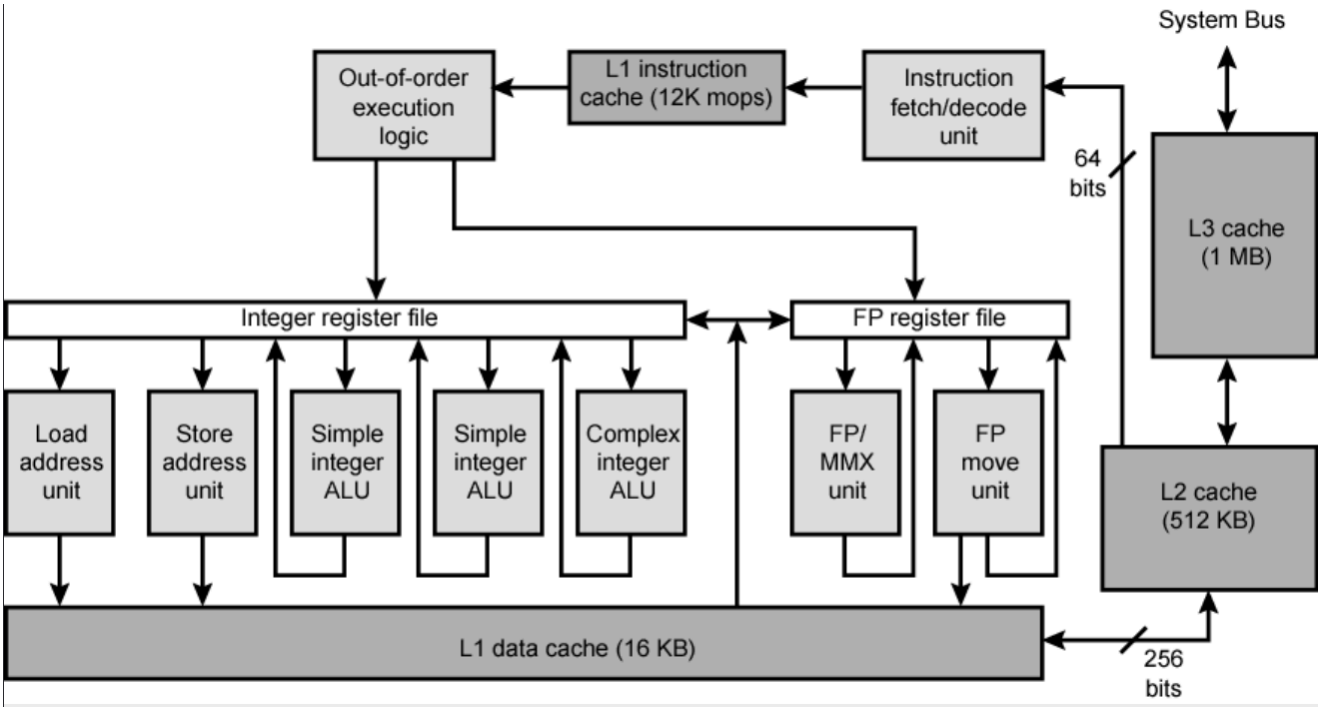
I blocchi candidati per la sostituzione sono scelti in modo casuale, facilità la realizzazione rispetto all'LRU

Associatività	A 2 vie		A 4 vie		A 8 vie	
Dimensione	LRU	Casuale	LRU	Casuale	LRU	Casuale
16KB	5,18%	5,69%	4,67%	5,29%	4,39%	4,96%
64KB	1,88%	2,01%	1,54%	1,66%	1,39%	1,53%
256KB	1,15%	1,17%	1,13%	1,13%	1,12%	1,12%

Empiricamente nel crescere della dimensione della cache lo scarto di performance è minimo o nullo

Funzionamento di un Pentium 4

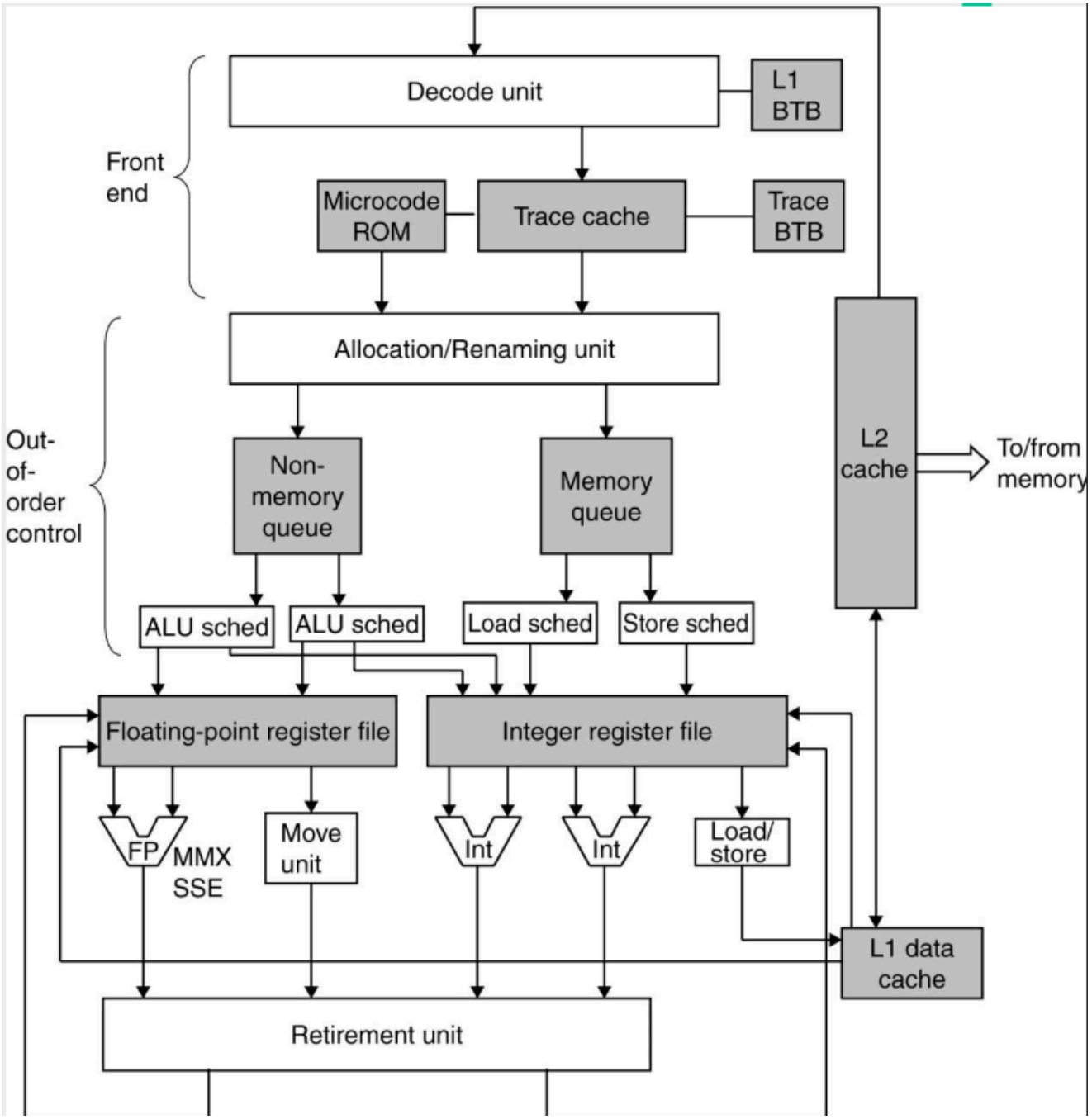
In un Pentium 4 vediamo tutte le componenti di esecuzione



Alcune componenti aggiuntive che non si sono viste sono:

- **Unità di Fetch/Decode delle Istruzioni**: Recupera e decodifica le istruzioni dalla memoria cache L1.
- **Unità di Esecuzione**: Esegue le istruzioni recuperate, utilizzando i dati dalla cache L1 e, se necessario, dalla cache L2 o L3.
- **Esecuzione Out-of-Order**: Consente al processore di eseguire le istruzioni non necessariamente nell'ordine in cui appaiono nel programma, migliorando l'efficienza e la velocità di esecuzione.

Funzionamento di un Intel i7



Ogni core contiene:

- L2, L1 e la logica per accedere a L3 condivisa
- Cache di pre-fetch che cercano di caricare istruzioni prima che siano referenziate.
- Front-end:
  - Prelievo, decodifica in formato RISC e salvataggio in $ delle istruzioni
  - Le istruzioni prelevate da $ L1 vengono passate ai decodificatori che determinano le operazioni da svolgere nella pipeline unità di controllo fuori sequenza:
- Unità di controllo fuori sequenza:
  - Tiene traccia delle micro-operazioni, se le risorse sono disponibili viene inserita in una delle due code altrimenti ritardata
  - Il fuori sequenza è dovuto alla pipeline

  Il pre-fetch è una tecnica utilizzata nei microprocessori per migliorare le prestazioni riducendo i tempi di attesa, in pratica, consiste nel precaricare le istruzioni o i dati dalla memoria prima che siano effettivamente necessari e questo permette al processore di

avere già a disposizione le informazioni quando servono, evitando ritardi dovuti alla velocità inferiore della memoria rispetto al processor

Write policy

Quando la CPU comincia ad aggiornare un valore in cache, il dato viene unicamente aggiornato nella cache e non nella memoria centrale, questo problema si chiama **perdita di aggiornamento**;
C'è una politica che permette di poter gestire questo caso chiamata **Write Back**, quando la CPU modifica i dati, questi vengono scritti solo nella cache (ad esempio, L1 o L2) e non immediatamente nella memoria principale, questo riduce la latenza, poiché la cache è molto più veloce della memoria principale;
Ogni blocco di dati nella cache ha un flag chiamato “dirty bit” e questo bit viene impostato a 1 quando i dati nel blocco sono stati modificati rispetto alla copia presente nella memoria principale;
I dati modificati vengono scritti nella memoria principale solo quando il blocco di cache che li contiene deve essere sostituito (ad esempio, quando un nuovo blocco di dati deve essere caricato nella cache e non c’è spazio disponibile). A questo punto, se il dirty bit è impostato, i dati vengono copiati dalla cache alla memoria principale

Nelle architetture moderne ci sono più CPU (con più cache) e la memoria RAM è condivisa, in questo caso viene usata la politica di **Write Through**, vengono aggiornate sia la memoria cache che la RAM con lo svantaggio di creare più traffico sui bus e rallentare il processo di scrittura

Quale delle due politiche oggi viene utilizzata?
La politica di scrittura più comunemente utilizzata è il write-back, questo perché il write-back tende a ridurre il numero di accessi alla memoria principale, migliorando così le prestazioni complessive del sistema.

In una CPU esistono diverse cache a livelli:

- Cache L1-I: contiene solo ed esclusivamente istruzioni
- Cache L1-D: contiene solo ed esclusivamente dati
  Di solito sono grandi tra i 16-64KB, poco più grande dei registri e direttamente sul chip, quindi molto veloci, ma troppo piccola per certe operazioni
- Cache L2: contiene sia dati che istruzioni, localizzata sul CPU package, condivisa da tutti i core, grande circa 0.5-2MB
- Cache L3: contiene sia dati che istruzioni, condivisa da tutti i processori ed esterna dalla CPU

3 livelli di cache sono necessari per avere un hit ratio molto elevato, riducendo il miss ratio al 0,001%

Domanda da esame, cosa si nota se la formula per il calcolo del tempo medio di accesso viene applicata per tutti e 3 i livelli?

Quando applichiamo la formula per il calcolo del tempo medio di accesso (Average Memory Access Time, AMAT) a una gerarchia di memoria con tre livelli di cache, emergono alcune osservazioni interessanti:

Abbiamo i livelli di cache L1, L2 e L3, ed ogni livello ha un suo tempo di accesso (hit time) e una probabilità di mancare (miss rate). La formula dell'accesso medio tiene conto di questi fattori per calcolare il tempo medio necessario per accedere ai dati.

La formula si espande in modo da considerare il tempo di accesso di ciascun livello e il miss rate che porta al livello successivo. Ad esempio, se un dato non è presente nella cache L1, viene cercato nella cache L2, e se non è presente nemmeno lì, si passa alla cache L3, e infine alla memoria principale. Questo processo riduce il miss penalty, cioè il tempo perso per accedere ai dati mancanti, perché ogni livello di cache è più veloce della memoria principale.

Un altro aspetto importante è l'effetto cumulativo dei miss rate. Se la cache L1 ha un alto miss rate, ci saranno più accessi alla cache L2, e così via. Pertanto, migliorare il miss rate della cache L1 può avere un impatto significativo sull'accesso medio complessivo.

Inoltre, il tempo di accesso (hit time) di ciascun livello di cache è cruciale. Le cache più vicine alla CPU, come la L1, sono molto veloci ma piccole, mentre le cache più lontane, come la L3, sono più lente ma più grandi. Questo bilanciamento tra velocità e dimensione è fondamentale per ottimizzare la gerarchia di memoria.

In sintesi, la progettazione della gerarchia di cache deve trovare un equilibrio tra ridurre il miss rate e mantenere tempi di accesso rapidi. Questo equilibrio determina l'efficienza complessiva della gerarchia di memoria e il tempo medio di accesso finale.

Interruzioni

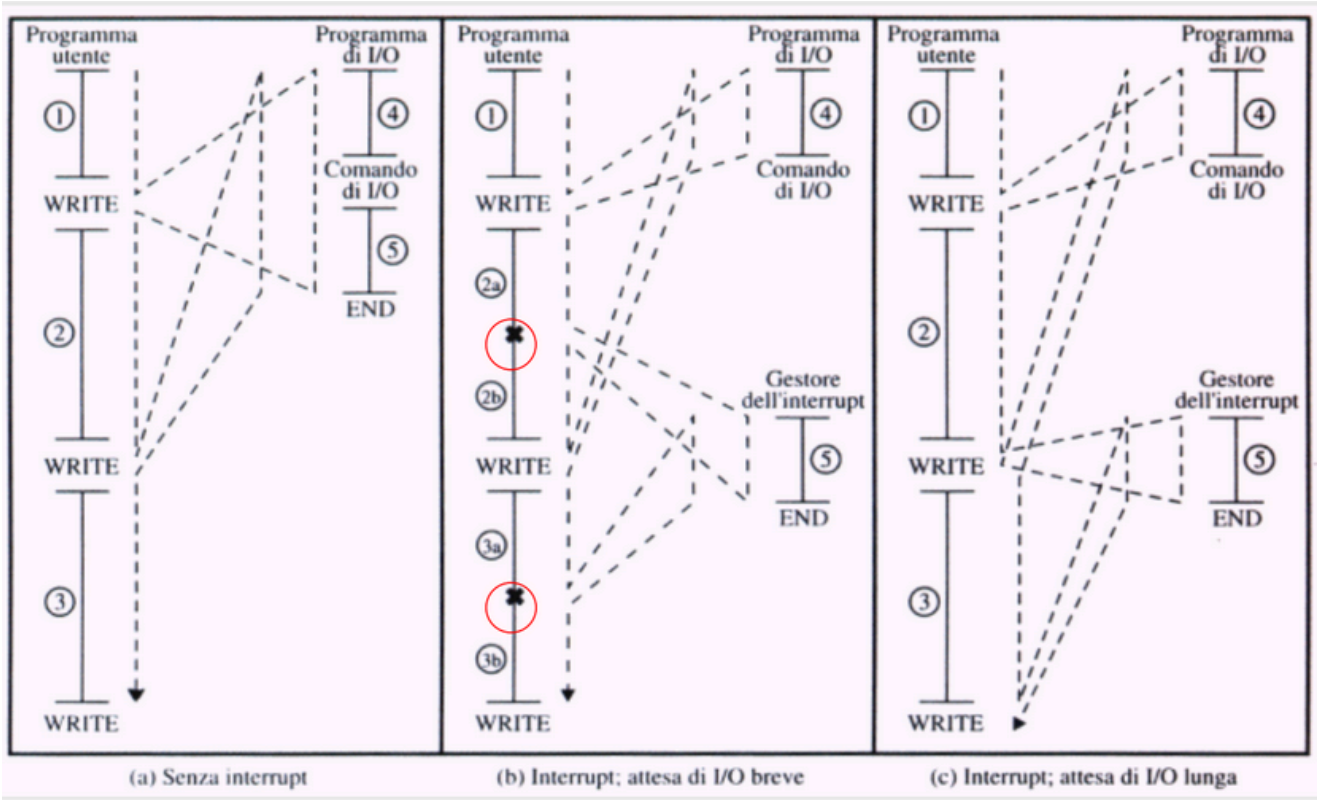
Le interruzioni consentono di interrompere la normale elaborazione del processore, con lo scopo di migliorare l'efficienza dell'elaborazione (rispetto ai dispositivi periferici molto più lenti del processore come le stampanti o i processi come un browser) e permettono al modulo di I/O, al termine del comando di quest'ultimo, di segnalare l'evento al processore.
Esistono due tipologie di interruzioni:

- **Interrupt**: generate da un dispositivo hardware, non connesso o dipendente a nessuna istruzione (es. stampante che finisce l'inchiostro)
- **Trap**: generate da un programma in esecuzione, viene generata proprio dal programmatore (es. overflow)
  L'Interrupt o la Trap si concretizzano con un bit che va ad 1 nella PSW
  Esistono diverse classi di interruzioni:
- Programma: errore di esecuzione (overflow, divisione per zero, violazione spazio di memoria)
- Timer: operazioni pianificate (ogni tot. istanti di tempo esiste un interrupt)
- I/O: tutti gli interrupt date dall'I/O

- Errori hardware: problemi fisici (cali di tensione)

Ma le interrupt dove viaggiano? Attraverso le linee di controllo
E le Trap? Non viaggiano sulle linee di controllo ma essendo un istruzioni arrivano direttamente nel bus della CPU (e nella PSW)

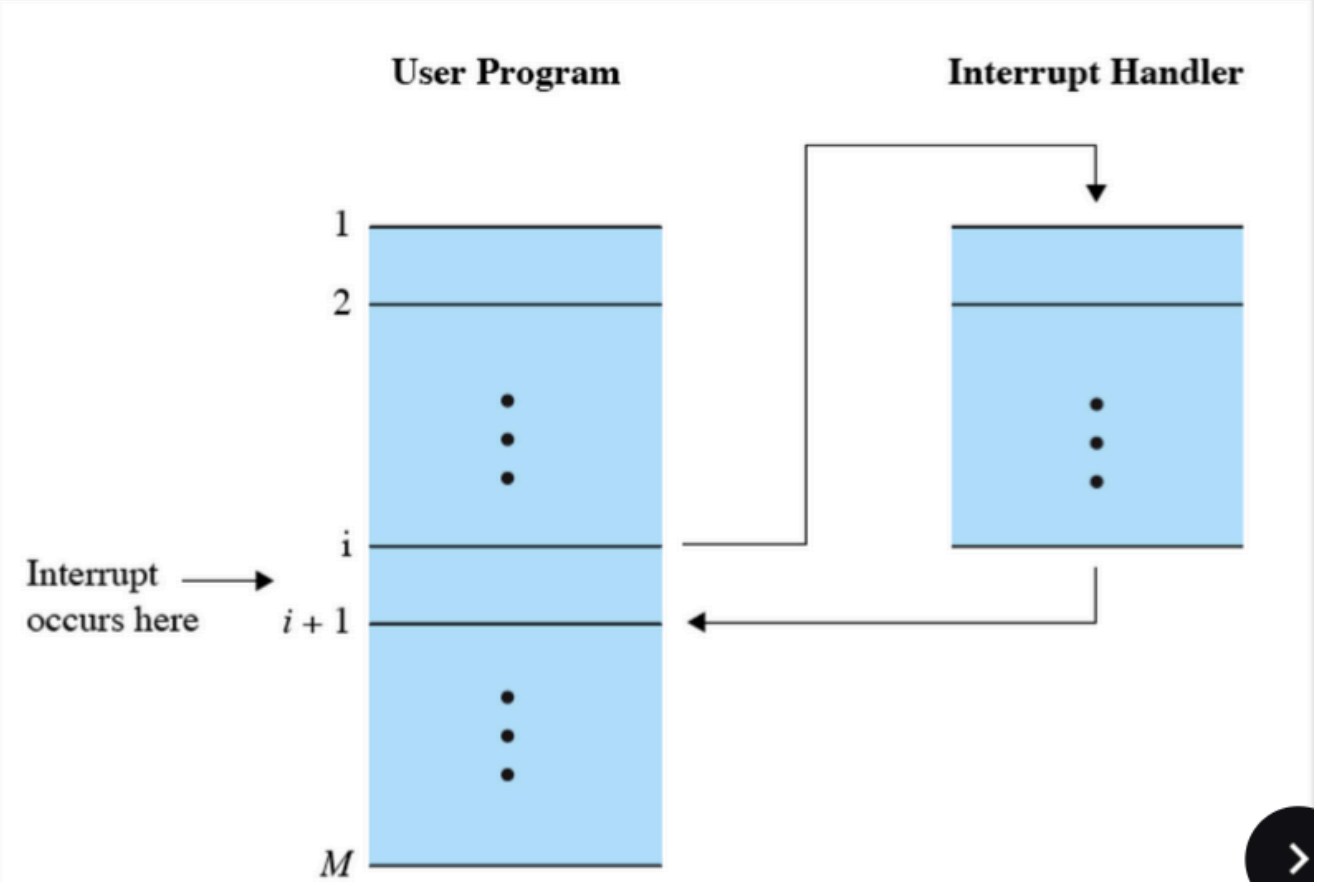
Questi sono processi con 3 esempi, senza interrupt, con interrupt di attesa breve I/O e con interrupt di attesa lunga I/O:



Gestione dell'interrupt

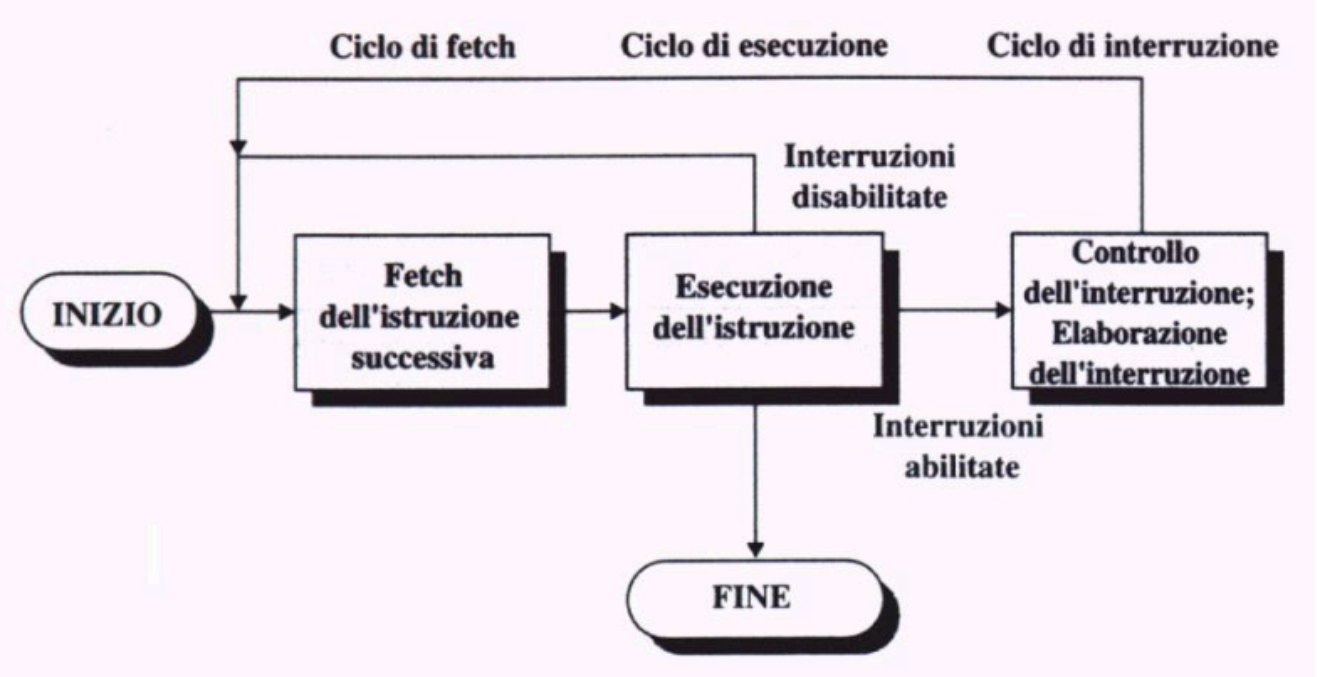
Il programma utente non contiene alcun comando speciale per la gestione della operazione I/O, quindi l'SO utilizza un **interrupt handler**, che ha il compito di determinare chi genera l'interruzione e, se conosce il dispositivo, chiama il programma (o driver) per gestire tale

evento



Ciclo di esecuzione con l'interruzione

Se l'interrupt arriva prima dell'istruzione (tra la fine prima istruzione e l'inizio della seconda istruzione) può essere gestito dal SO nello stack di sistema.
Il ciclo di esecuzione con interruzioni permette al processore di gestire eventi esterni o situazioni eccezionali durante l’esecuzione delle istruzioni. Normalmente, il processore segue un ciclo di esecuzione sequenziale, cioè legge l’istruzione successiva, la decodifica, la esegue e poi passa alla seguente. Tuttavia, con il ciclo di esecuzione interrotto, il processore verifica la presenza di eventuali richieste di interruzione alla fine di ogni ciclo di esecuzione.
Quando viene rilevata un’interruzione, il processore interrompe l’esecuzione normale e avvia un processo per gestire l'evento segnalato. A questo scopo, salva lo stato corrente (come l'indirizzo dell'istruzione in esecuzione e lo stato del registro di stato del programma) per poter riprendere il lavoro al termine della gestione dell'interruzione.



Domanda ipotetica ,se si immagina una macchina CISC? Rimane lo stesso

Elaborazione delle interruzioni

Una volta rilevata l'interruzione, il processore invia un segnale di riconoscimento (acknowledgement) al dispositivo che ha generato l'interruzione. Questo serve a confermare la ricezione della richiesta. Il sistema operativo contiene il programma specifico per gestire l'interruzione: identifica la causa dell'interruzione, determina quale modulo del sistema operativo la deve gestire e avvia il programma corrispondente. Prima di eseguire il programma di gestione dell'interruzione, il processore salva alcune informazioni cruciali, come il contatore di programma (PC), lo stato del programma (PSW), e i registri che potrebbero essere modificati.

In breve:

1. **Individuazione dell'interruzione:** Il processore la rileva alla fine del ciclo di esecuzione.
2. **Riconoscimento:** Viene inviato un segnale di riconoscimento al dispositivo.
3. **Gestione:** Viene identificato il tipo di interruzione e avviato il programma di gestione appropriato.
4. **Salvataggio dati:** Le informazioni necessarie vengono salvate per consentire la ripresa del lavoro.

In questo modo, il processore riesce a rispondere in modo efficiente e a tornare allo stato precedente dopo aver gestito l'interruzione.

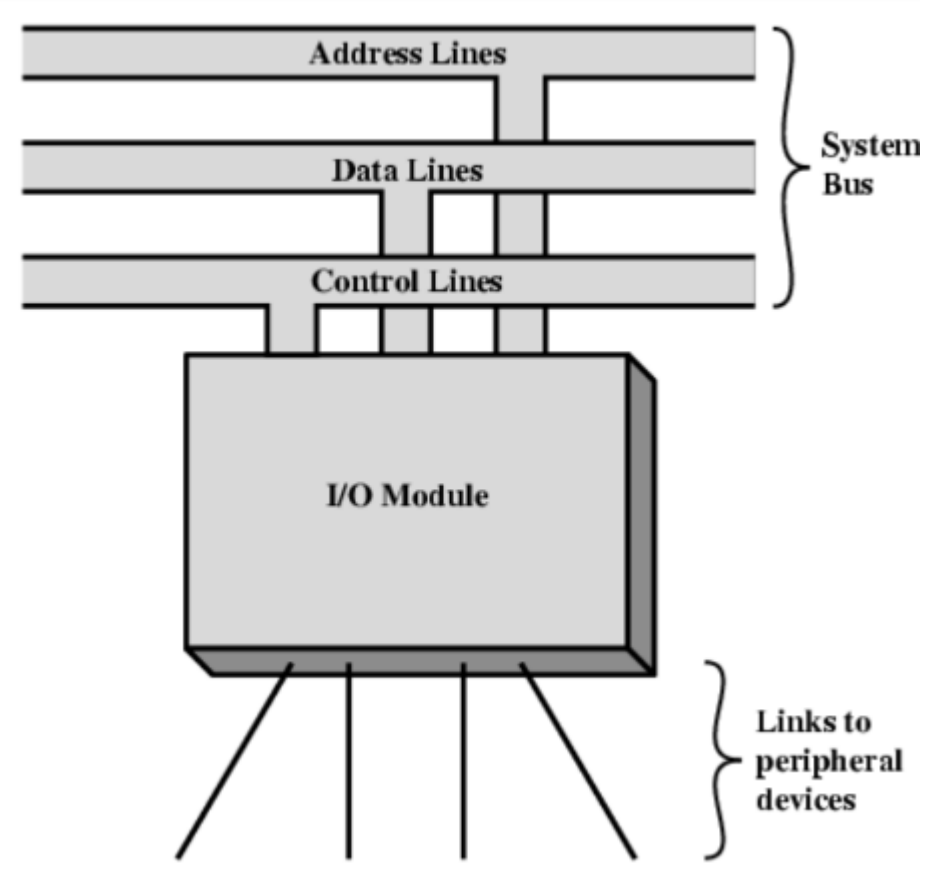
Dispositivo di I/O

I dispositivi di input e output sono molteplici

- Principali dispositivi di input:
  - tastiera
  - mouse
- Principali dispositivi di output:
  - monitor
  - stampante
  - casse acustiche
- Collegamenti tramite:
  1. porte di comunicazione (tastiera e mouse),
  2. schede montate sulla motherboard.
     - Monitor - scheda video,
     - Casse - scheda audio.
       NB: queste due schede possono anche essere integrate nella scheda madre.

Modulo di I/O

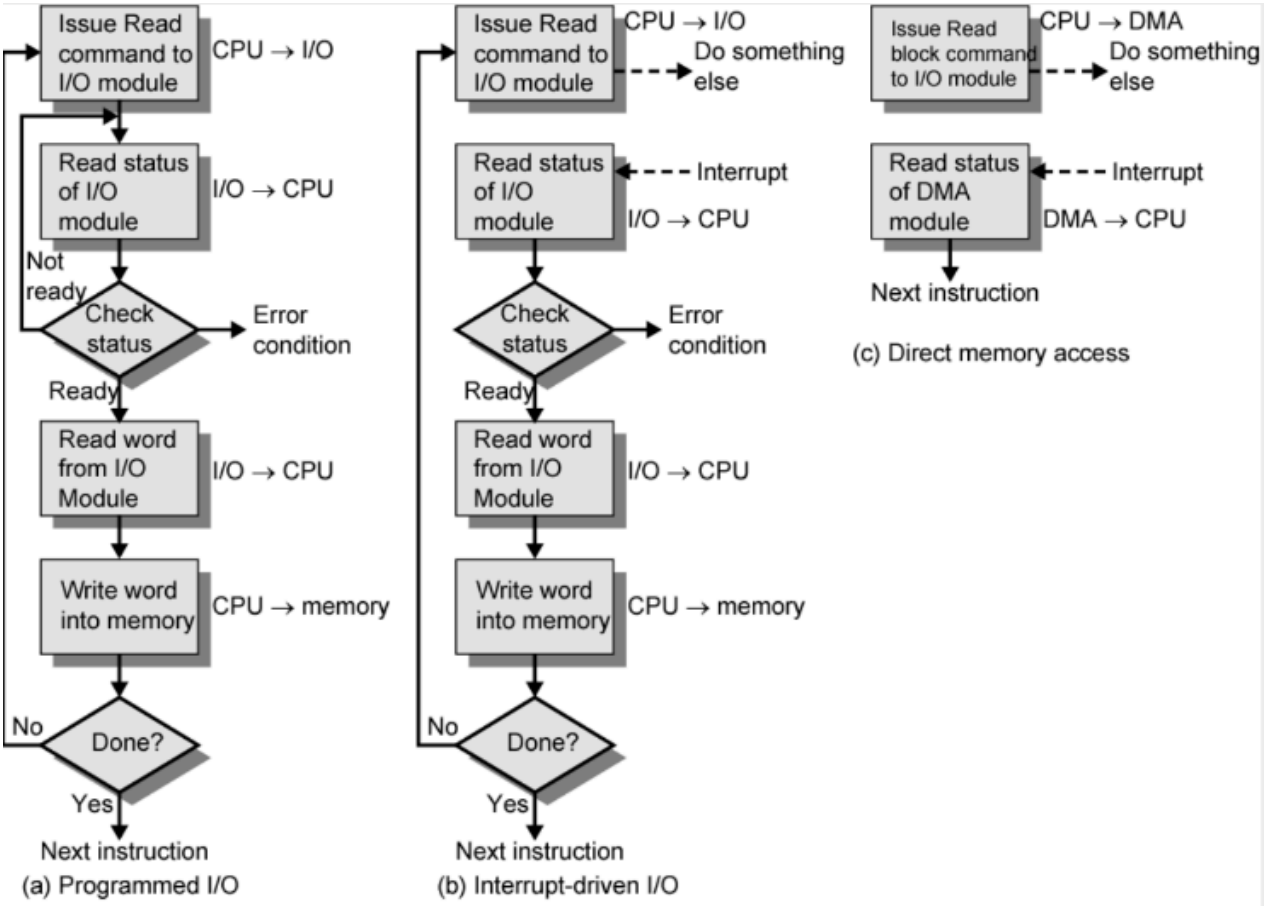
Ma come fanno questi dispositivi a parlare? Tramite un modulo dedicato che arrivano ai bus.



Tecniche di I/O

Esistono 3 modi per poter far parlare i dispositivi tra di loro:

- Programmato: legacy, pochissimo utilizzato
- Interrupt driven: utilizzato nel mondo consumer
- Direct Memory Access (DMA): di nicchia, principalmente mondo server



I/O programmato

Nel I/O programmato i comandi vengono inviati in questo modo:

- La CPU comunica al dispositivo I/O
- La CPU legge lo stato del dispositivo I/O
- Controlla lo stato, vede se è presente e se sì procede, altrimenti la CPU torna al passaggio precedente
- Si legge la parola dal dispositivo I/O che arriva alla CPU
- La parola viene scritta in memoria
- Fine
  Questo approccio ha due problemi:

1. La CPU passa tutto il tempo a interrogare il dispositivo I/O sulla linea di controllo (attesa attiva, busy wait)
2. Tutto il transito dei dati passa dalla CPU

I/O Interrupt Driven

Lo stesso approccio dell'I/O programmato ma presenta delle interruzioni

- La CPU comunica al dispositivo I/O, ma continua ad eseguire altre istruzioni
- La CPU legge lo stato del dispositivo I/O e manda un interrupt
- Controlla lo stato, vede se è presente e se sì procede, altrimenti avviene una condizione di errore
- Si legge la parola dal dispositivo I/O che arriva alla CPU
- La parola viene scritta in memoria
- Fine
  Questo approccio toglie il problema dell'attesa attiva ma rimane ancora il secondo:
- Tutto il transito dei dati passa dalla CPU

DMA(Direct Memory Access)

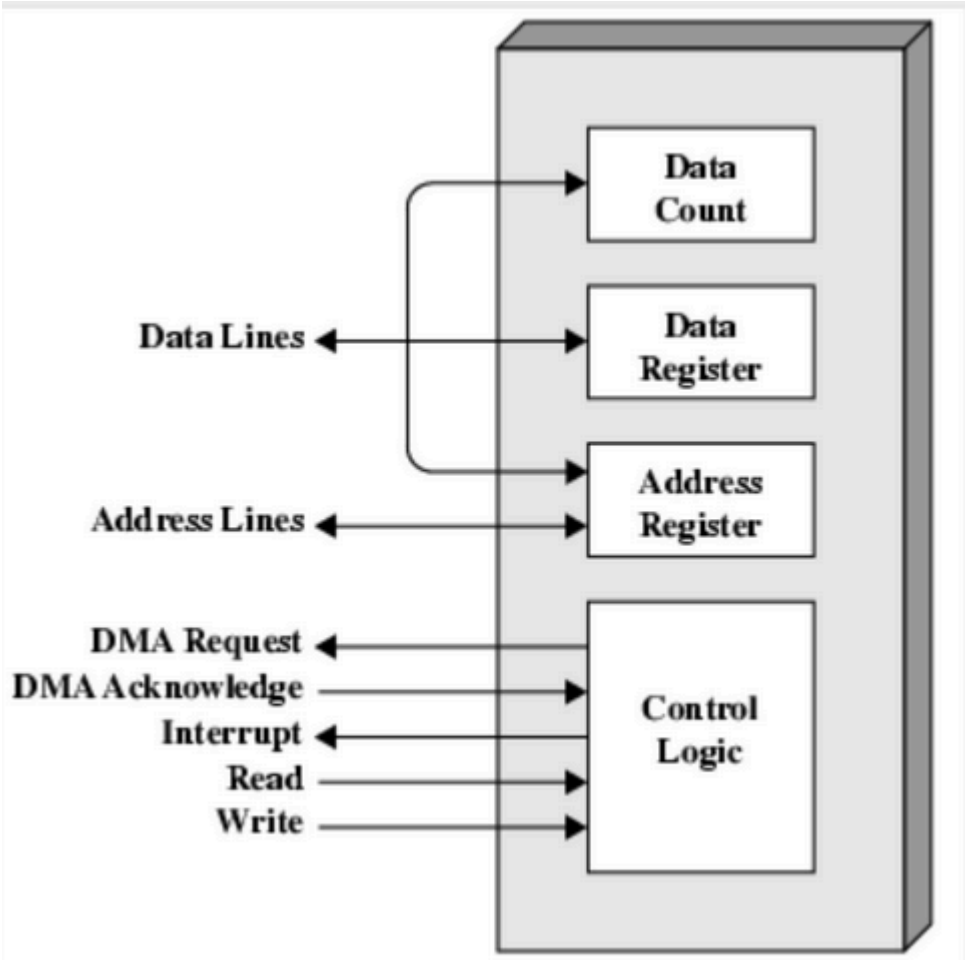
Questo metodo risolve il problema di tutti e due i metodi, implementando un dispositivo fisico sul bus

- La CPU comunica con il controller DMA, ma continua ad eseguire altre istruzioni
- La CPU comunica l'operazione da svolgere (Lettura/Scrittura, Indirizzo del dispositivo, Indirizzo di partenza del blocco di memoria per i dati ,Quantità di dati da trasferire)
- La CPU intanto svolge altri lavori mentre il controller DMA si occupa del trasferimento
- Il controller DMA invia un interrupt quando ha concluso il trasferimento

Il DMA al suo interno si collega al bus dati e presenta al suo interno:

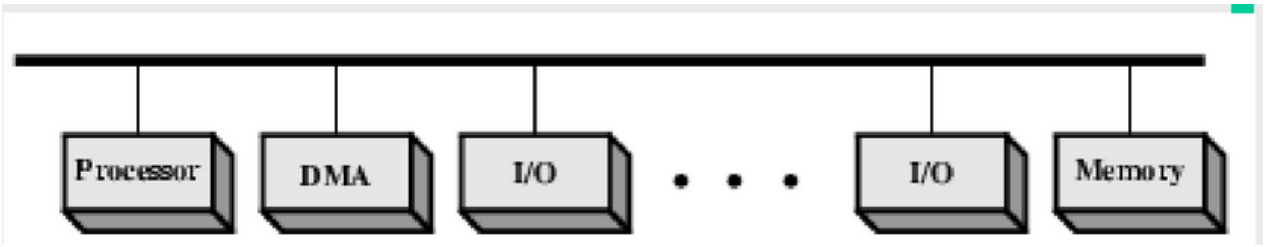
- Il registro degli indirizzo
- Una Control Unit che contiene diverse informazioni
  - Richiesta di DMA
  - DMA acknowledge, mandato dalla CPU
  - Interrupt
  - Lettura

- Scrittura

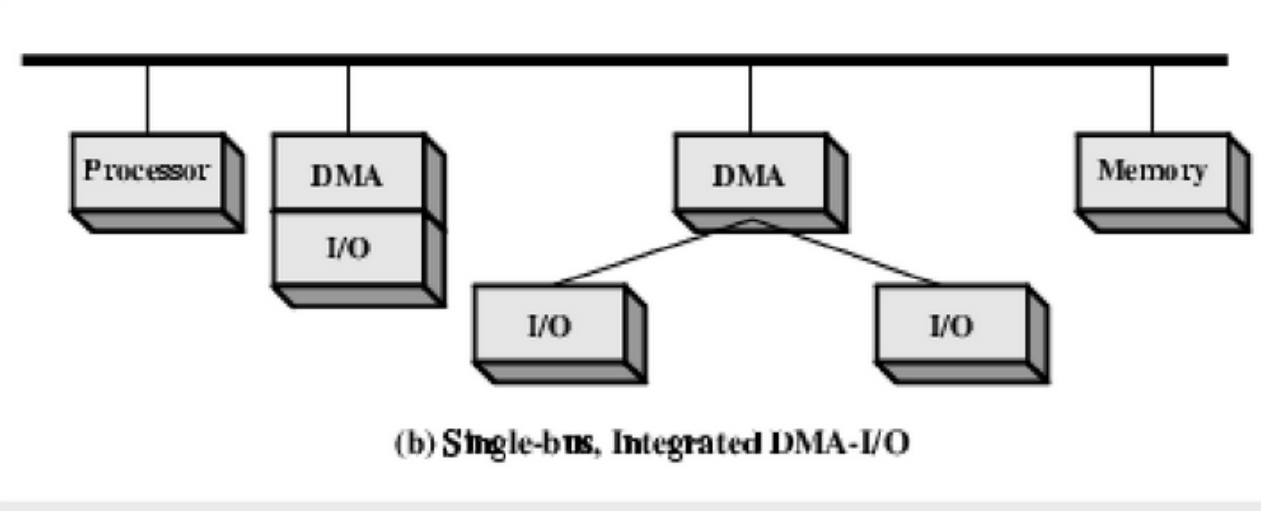


Il DMA presenta diverse configurazioni

1. Configurazione a unico DMA, che viene utilizzato due volte per ogni trasferimento (da I/O a DMA e poi DMA a memoria), la più economica

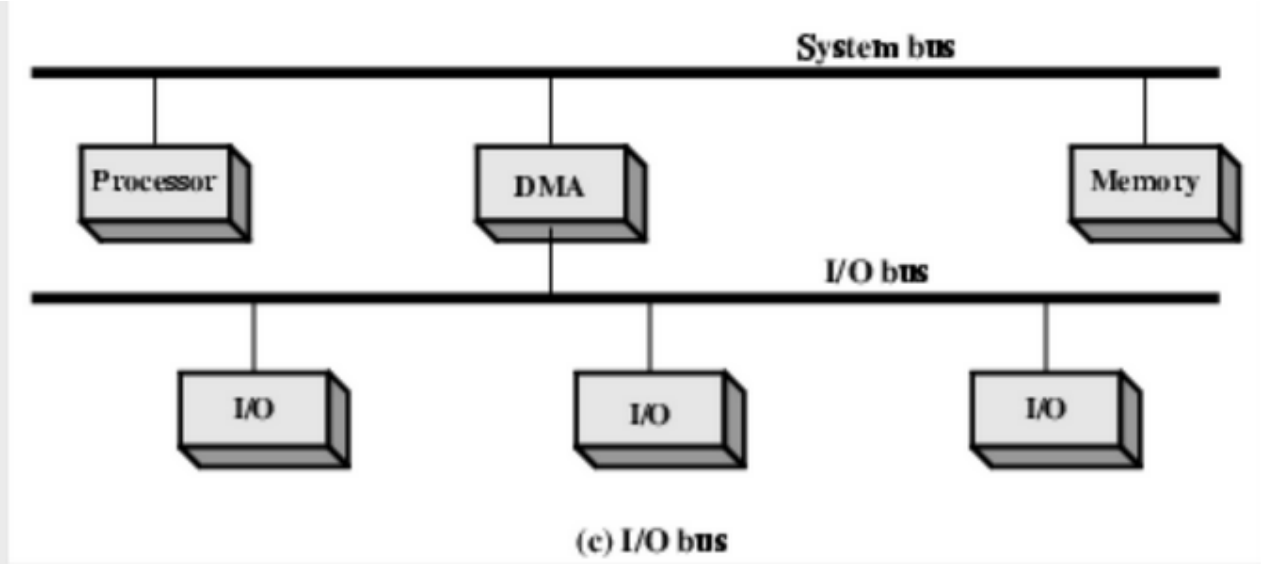


2. Configurazione a due dispositivi DMA, viene associato a due dispositivo oppure associato direttamente a un dispositivo, il bus viene utilizzato una volta



(b) Single-bus, integrated DMA-I/O

3. Configurazione a doppio bus, con bus dedicato all' I/O, ogni trasferimento impegna il bus una sola volta



(c) I/O bus

Hard Disk

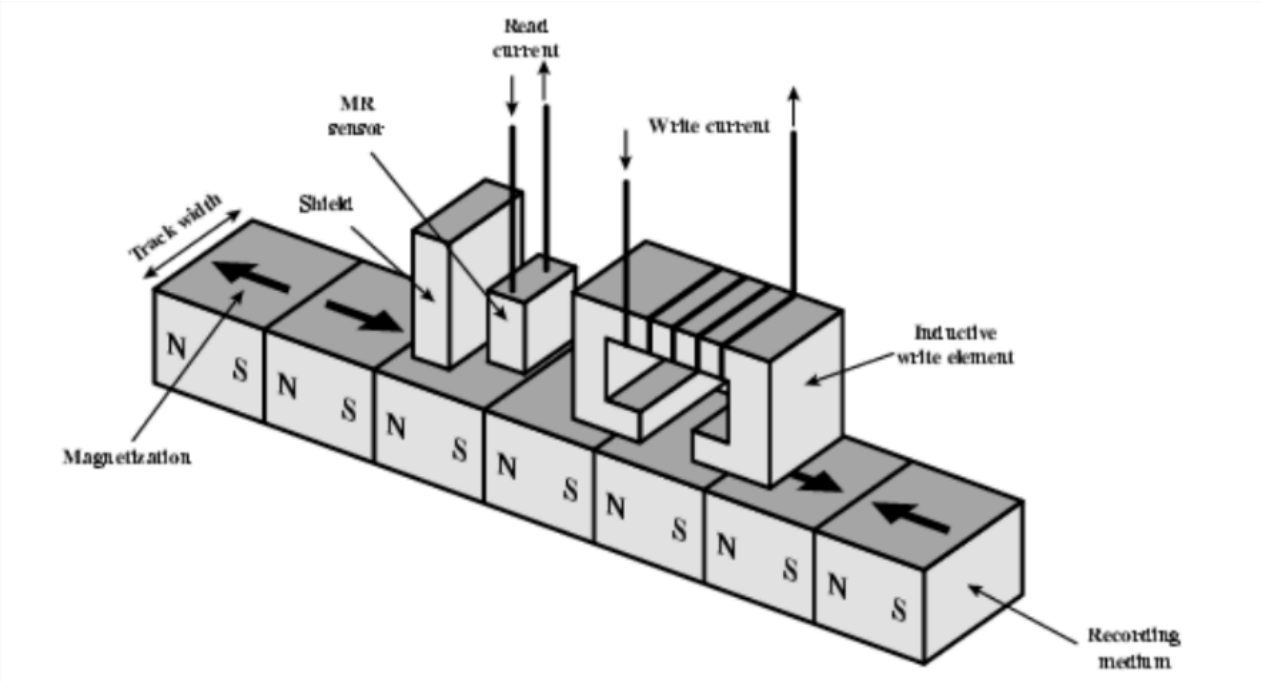
Gli hard disk sono un tipo di memoria permanente, costituito da uno o più dischi a rapida rotazione, realizzati in alluminio o vetro, rivestiti di materiali ferromagnetico e da due testine per ogni disco, una per lato.
Hanno dimensioni da 2.5 o 3.5 pollici e presentano diverse di velocità di rotazione, che si traducono velocità maggiore = MB/s maggiori

Meccanismi di scrittura e lettura

Durante le operazioni di scrittura e lettura la testina è fissa e i piatti ruotano, ci sono modi diversi per la lettura e la scrittura:

- Scrittura: la corrente attraverso i fili produce un campo magnetico, che viene memorizzata nella superficie sottostante

- Lettura: il campo magnetico in movimento produce nel filo una corrente elettrica



Organizzazione di dati

I dati sono organizzati in anelli concentrici (o tracce) con dei gaps, le tracce sono divise in settori e ogni unità minima scrivibile e leggibile è il settore.

Piatti multipli

Gli hard disk hanno una testina per lato (due testine per piatto), tutte allineate.
Le tracce su più piatti individuano un cilindro, dove vengono distribuiti i dati in più tra loro per ridurre il movimento delle testine e aumentare la velocità complessiva

Velocità di accesso

La velocità d'accesso si misura con diversi parametri:

- Seek time: tempo necessario alla testina per posizionarsi sulla traccia di interesse
- Rotational delay: tempo necessario affinché la testina si posizioni sul settore di interesse
- Transfer time: tempo necessario al trasferimento dei dati
- Access time; tempo che incorre tra a richiesta di accesso ai dati e l'istante in cui sono disponibili

Su quale parametro influisce la disposizione dei dati secondo cilindri?
La disposizione dei dati secondo cilindri influisce sul **seek time** (tempo di ricerca) dell'hard disk, poiché le testine di lettura/scrittura sono allineate su tutti i piatti, organizzare i dati in cilindri riduce il movimento necessario delle testine per leggere dati correlati, in questo modo, le testine non devono spostarsi continuamente tra tracce lontane, ma possono accedere rapidamente ai dati nel medesimo cilindro, migliorando l'efficienza di lettura/scrittura.

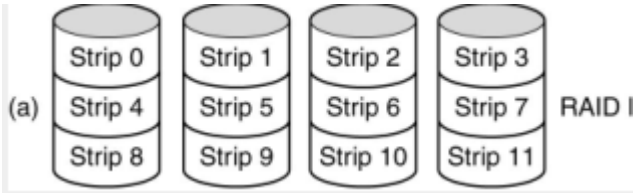
Su quale parametro influisce la velocità di rotazione del disco?
La velocità di rotazione del disco influisce sul **rotational delay** (ritardo di rotazione), una velocità di rotazione più alta riduce il tempo necessario affinché la testina raggiunga il settore desiderato sul piatto.
In pratica, più è veloce la rotazione, minore sarà il tempo di attesa per posizionarsi sul settore corretto, e quindi maggiore sarà il transfer rate (velocità di trasferimento dei dati), permettendo all'hard disk di leggere e scrivere dati più velocemente.

RAID (Redundant Array of Inexpensive Disks)

Il RAID è una tecnica che ha lo scopo di migliorare prestazioni ed affidabilità di un sistema di memorizzazione di massa utilizzando una batteria di dischi piuttosto che un unico disco.
Tutti i RAID, oltre ad apparire al sistema operativo come unico disco, permette di distribuire dati su diverse unità consentendo elaborazione parallele.
Per implementarlo si usa la tecnologia **SCSI (SCASI)** che consente con un controller e dischi annessi, l'utilizzo di più dischi con buone prestazioni
I RAID vanno da 0 a 5 livelli

RAID 0

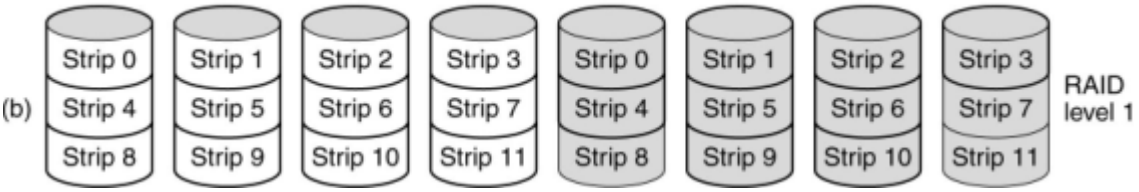
Questa è una tecnica di striping, il disco simulato RAID è visto come se ognuno dei suoi k settori fosse suddiviso in strisce con settori da 0 a $k-1$ per la strip 0, k a $2k-1$ per la strip 1 e così via per ogni strip. Il suo tipo di organizzazione scrive sulle strio consecutive in modalità consecutiva in modo ciclico (round robin);
Per la lettura/scrittura dati di 4 settori successivi il controllore RAID spezzerà questo comando in 4 comandi separati (uno per ciascun disco) e li farà eseguire in parallelo.
Il RAID 0 lavora meglio quando le richieste sono di grandi dimensioni e peggio in caso di sistemi operativi che tendono a richiedere i dati un settore alla volta.
Il RAID 0 non è ridondante (per questo non viene considerato un vero RAID)



RAID 1

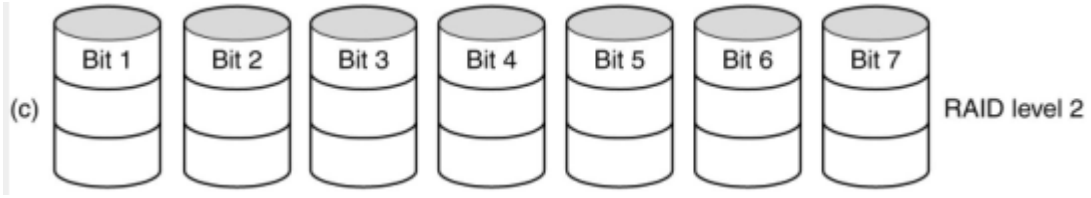
In questo caso si usa una tecnica di striping e mirroring, come nel RAID 0 ogni strip viene mappata su due diversi dischi, è composto da dischi pari di striping e di mirroring.
Nel caso di una scrittura lo strip viene scritta due volte, mentre nella lettura si possono usare entrambe le copie, distribuendo il carico di lavoro su più unità di lettura, ma non migliorando quindi le prestazioni di scrittura,che risulterebbero uguali a quella di una singola unità, ma solo quelle di scrittura che sono il doppio in questo caso. Il malfunzionamento di un disco è

facilmente recuperabile ma ha costi elevati



RAID 2

Usa la tecnica di accesso parallelo, tutti i dischi del RAID partecipano all'esecuzione di I/O in maniera sincrona (tutte le testine di tutti i dischi assumono posizioni analoghe in ogni momento).
Il RAID 2 adotta lo striping dei dati, le strisce assumono dimensioni di mezzo o un byte o di una parola, lavorando con mezzo byte (più tre bit di controllo dati) si ottengono i sette bit memorizzati tutti su dischi diversi.
La lettura e la scrittura avvengono in parallelo (di mezzo byte più tre bit di controllo) ma è comunque dispendioso, i dischi devono ruotare in maniera sincronizzata e sovraccarica il controllore che deve controllare rapidamente i bit di controllo



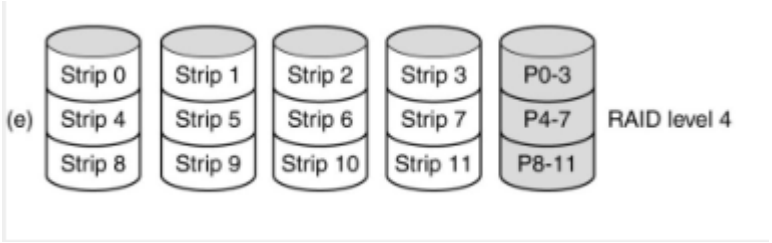
RAID 3

Usa la tecnica di accesso parallelo, una versione semplificata del RAID 2.
Il bit di di parità viene calcolato per ogni parola di dati e poi scritto su un apposito disco. Dato che le parole di dati sono distribuite su più unità, anche in questo caso i dischi devono essere sincronizzati, ma nel caso di rottura di un disco, attraverso il bit di parità si riesce a recuperare i dati, la lettura e la scrittura avviene quindi in parallelo (mezzo byte più un bit di parità) ma i dischi devono comunque essere sincronizzati nella rotazione

RAID 4

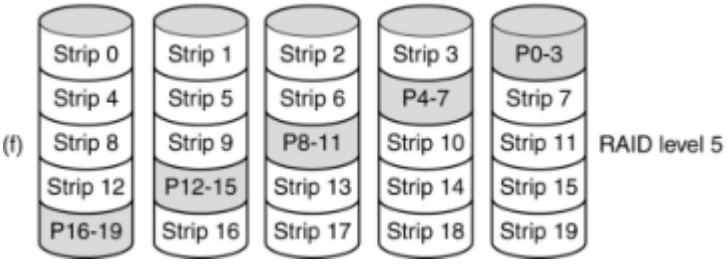
Usa una tecnica basata sullo striping ed è praticamente come il RAID 0 con la differenza di avere una parità calcolata strip per strip su un disco aggiuntivo, essendo quindi più protetta di un RAID 0 dalla rottura di un disco ma con prestazioni scarse se si cambiano piccole moli di dati (bisogna ricalcolare tutto lo strip di parità) e il disco di parità diventa un collo di

bottiglia per il sistema



RAID 5

Usa una tecnica basata sullo striping, il RAID 5 è come il RAID 4 con strip di parità distribuiti su tutti i dischi in modalità round robin, una parità calcolata strip per strip scritta su un disco aggiuntivo.
Protegge dalla rottura di un disco ma ha prestazioni scarse se si cambiano piccole moli di dati



Gestione dell'Hard Disk

Deframmentazione

La deframmentazione è un processo che riorganizza i dati sul disco rigido per migliorarne le prestazioni. Quando un file viene salvato o modificato, potrebbe essere suddiviso in frammenti e distribuito in diverse aree del disco. Con il tempo, questo processo può causare la frammentazione, in cui i dati di un singolo file si trovano sparsi su più settori fisici. Questo rende la lettura dei file più lenta, perché le testine devono spostarsi in vari punti per accedere ai dati.
La deframmentazione risolve questo problema riunendo i frammenti dei file, posizionandoli in settori contigui e liberando spazio. In questo modo, il disco può accedere ai file più velocemente, migliorando la velocità complessiva del sistema.

Formattazione

La formattazione di un disco consiste nella preparazione di una struttura di memorizzazione che permetta di scrivere, leggere e organizzare i dati sul disco. Esistono due tipi di formattazione:

- **Formattazione fisica o a basso livello:** Suddivide il disco in tracce e settori tramite un processo che ne organizza fisicamente la superficie. Viene eseguita solo una volta in fabbrica o durante una completa re inizializzazione. Durante questo processo, viene preparata la superficie magnetica per poter contenere i dati.

- **Formattazione logica o ad alto livello:** Organizza il disco in base al file system scelto (ad esempio FAT32, NTFS, etc.). Questa suddivisione logica crea una struttura che il sistema operativo può gestire per catalogare i file e permettere un rapido accesso ai dati.

Partizionamento

Il partizionamento divide un disco fisico in più sezioni logiche (partizioni), ognuna delle quali può essere formattata e utilizzata indipendentemente. Le partizioni possono avere sistemi di file diversi e vengono gestite dal sistema operativo come unità separate. Questa suddivisione permette, per esempio, di avere una partizione per il sistema operativo e un'altra per i dati, facilitando l'organizzazione e la gestione dei file.

Cosa significa cancellare un file?
Cancellare un file normalmente significa **segnare lo spazio che occupa come disponibile**, senza rimuovere immediatamente i dati dal disco. Il file system elimina il riferimento al file, rendendolo "invisibile" per l'utente e permettendo di riutilizzare quello spazio per nuovi dati. Tuttavia, il contenuto originale del file rimane fisicamente presente sul disco finché non viene sovrascritto da nuovi dati.
Esistono software di recupero dati, come *drive rescue*, che possono ripristinare file cancellati finché non sono stati sovrascritti. Per eliminare completamente un file, occorre sovrascrivere più volte i dati, rendendoli irrecuperabili.

SSD - Memorie a stato solido

Gli SSD, o dischi a stato solido, sono dispositivi di memorizzazione che usano una tecnologia diversa dagli hard disk tradizionali: anziché basarsi su dischi magnetici rotanti e testine di lettura, utilizzano **memorie flash**. Questo cambiamento di struttura e tecnologia comporta vantaggi significativi in termini di velocità e affidabilità, ma anche alcune limitazioni:

Vantaggi principali degli SSD

Uno dei principali vantaggi degli SSD è la **velocità**. A differenza degli hard disk, che devono far ruotare un disco e posizionare fisicamente la testina sulla traccia giusta per accedere ai dati, negli SSD l'accesso ai dati è immediato e diretto. Questa caratteristica rende le operazioni di lettura e scrittura molto più rapide, migliorando nettamente le prestazioni del sistema e riducendo i tempi di attesa durante l'apertura di file o l'avvio di applicazioni.

Inoltre, poiché gli SSD non hanno parti in movimento, risultano anche più **resistenti e affidabili** in caso di urti o vibrazioni. Questo li rende particolarmente adatti per i dispositivi portatili come laptop e tablet, che spesso sono soggetti a spostamenti e rischi di caduta.

Limiti e svantaggi degli SSD

Tuttavia, gli SSD hanno alcuni svantaggi. **Il costo** per unità di memorizzazione, ad esempio, è ancora più alto rispetto agli hard disk tradizionali, anche se questa differenza si sta

riducendo nel tempo. Quindi, per chi ha bisogno di una grande capacità di memorizzazione a un costo ridotto, gli hard disk magnetici restano un'opzione interessante.

Un altro aspetto da considerare è che le celle di memoria di un SSD hanno una durata limitata in termini di cicli di scrittura. Ogni cella può essere scritta solo un certo numero di volte, generalmente intorno a **100.000 cicli**. Una volta superato questo limite, le celle possono iniziare a degradarsi e a perdere la capacità di memorizzare informazioni. Per mitigare questo problema, gli SSD moderni utilizzano una tecnica chiamata *wear leveling*, che distribuisce uniformemente le scritture su tutte le celle per prolungare la vita complessiva del dispositivo.

Infine, anche se gli SSD sono generalmente più affidabili degli hard disk tradizionali, questa durata limitata delle celle può rappresentare una **sfida di affidabilità** nel lungo termine. Se un SSD viene usato intensivamente per scritture frequenti e pesanti, potrebbe deteriorarsi prima di un hard disk che non soffre dello stesso tipo di limite.

GPU (Graphic Processing Unit)

La GPU è un unità di elaborazione grafica, è un particolare coprocessore all'interno di un computer specializzato nella resa grafica
Presenta:

- Processore grafico
- RAM (in versione DMA, non la condivide con nessuno)
- BIOS
- BUS I/O

Rispetto ad una CPU presenta migliaia di core ed è unicamente utile per problemi parallelamente parallelizzabili

Processore GPU Fermi

È la base delle schede grafiche moderne.
È un architettura SIDM, ci sono calcoli simultanei ma un singolo processo in un dato momento, è particolarmente applicabile ad attività comuni come la regolazione del contrasto di un immagine digitale o la regolazione del volume dell'audio digitale, aumentando la prestazioni all'uso multimediale

Parallelismo

Il parallelismo è la capacità di eseguire più azioni nello stesso istante.
Esistono due tipi di parallelismo

- **A livello di processore**: più processori lavorano congiuntamente sullo stesso problema
- **A livello di istruzione**: il parallelismo viene sfruttato all'interno delle singole istruzioni per fare in modo che l'elaboratore esegue più istruzioni contemporaneamente

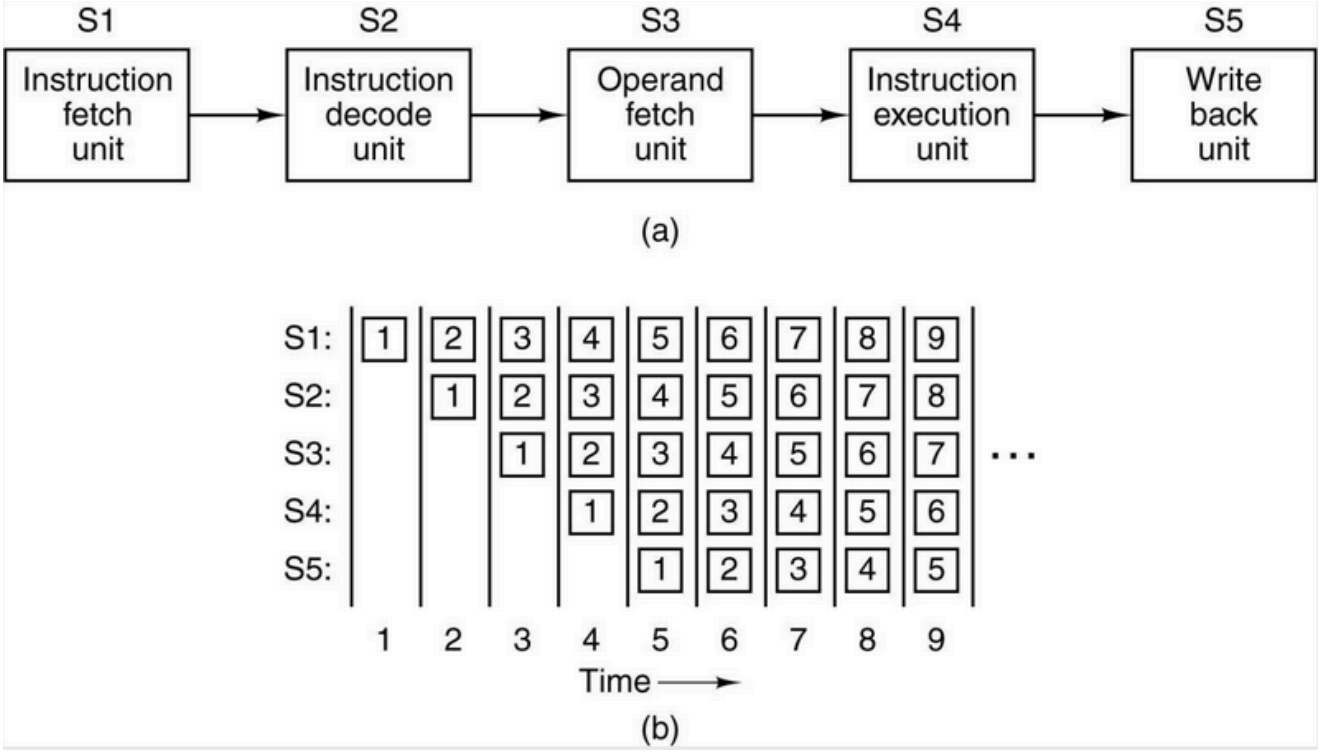
Categorie di Calcolatori con Tassonomia di Flynn

Si classificano poi con la classificazione della Tassonomia di Flynn:
Singolo processore

- Single Instruction Single Data(SISD): in un istante la macchina con un unico processore è in grado di eseguire una singola istruzione su un singolo set di dati, praticamente la macchina di Von Neumann pura
  Processori paralleli
- Single Instruction Multiple Data (SIMD): In un istante la macchina con processori paralleli è in grado di eseguire una singola istruzione contemporaneamente su set diversi di dati (ALU Vettoriali)
- Multiple Instruction Multiple Data (MIMD): In un istante la macchina con un set di processori esegue simultaneamente diverse sequenze di istruzioni su set diversi di dati

Pipeline (Parallelismo a livello di istruzione)

Uno dei concetti applicati è quello della pipeline, dove in una CPU si divide il ciclo di esecuzione di più parti ciascuna parte è affidata ad un componente hardware



Un processore integra una pipeline a cui sono associate più unità funzionali, come l'ALU (unità logica-aritmetica) e le unità di accesso alla memoria, per eseguire più istruzioni simultaneamente, questa struttura permette di aumentare ulteriormente il livello di parallelismo: ogni unità funzionale può elaborare istruzioni diverse nello stesso ciclo.

Come si vede da questo schema, c'è una differenza tra un clock su una CPU senza pipeline (macchina di Von Neumann pura) e un clock su una CPU con pipeline

N.Istruzione	Clock senza pipeline	Clock con pipeline

i	5	5
i+1	10	6
i+2	15	7
i+3	20	8
i+4	25	9
i+5	30	10

Come impatta la pipeline nell'architettura?
In un'architettura senza pipeline, ogni istruzione viene completata interamente prima di iniziare quella successiva, come avviene in una macchina di Von Neumann "pura". Questo significa che ogni ciclo di clock è dedicato interamente a una singola istruzione fino alla sua conclusione, risultando in un elevato numero di cicli per completare una serie di istruzioni.

Con la pipeline, invece, il ciclo di esecuzione viene suddiviso in fasi separate (come fetch, decode, execute, memory access e write-back), ciascuna gestita da unità funzionali dedicate. Questo consente di eseguire contemporaneamente porzioni di diverse istruzioni: mentre una fase della pipeline completa la parte successiva di un'istruzione, un'altra fase può iniziare con l'istruzione successiva. Di conseguenza, una nuova istruzione può avanzare in ogni ciclo di clock, riducendo notevolmente il tempo totale di esecuzione.

Calcolo del parallelismo

Il vantaggio della pipeline è misurabile in termini di **speed-up**, o velocizzazione, questa è la formula per il calcolo normale e quello per il calcolo del suo vantaggio rispetto a quello normale

Sia
- k - numero di stadi della pipeline
- n - numero di istruzioni da eseguire
- t - tempo di uno stadio (tempo massimo).

(si considerino inoltre trascurabili i tempi di commutazione da uno stadio al successivo)

In una macchina sequenziale senza parallelismo il tempo per eseguire le istruzioni è

$$T_{convenzionale} = nkt$$

In una macchina dotata di pipe (senza salti):

$$T_{pipe} = [k + (n-1)] \cdot t$$

Il fattore di velocizzazione (speed-up) della pipeline rispetto ad una architettura tradizionale è:

$$\frac{T_{convenzionale}}{T_{pipe}} = \frac{nkt}{[k+(n-1)]t} = \frac{nk}{[k+(n-1)]}$$

Trattamento dei salti

Ma come si fa a gestire un salto (condizionato) all'interno di una pipeline per non perdere prestazioni?
Esistono delle soluzioni che un produttore di processore implementa:

Flussi multipli

Si replicano le parti iniziali della pipeline e si prelevano entrambe le istruzioni coinvolte nel salto, facendo usi di due flussi. Potrebbe generare dei ritardi nel caso di contesa tra i due flussi. Soluzione utilizzata comunque poco, poichè le istruzioni potrebbero essere uguali

Prelievo anticipato della destinazione

Quando viene riconosciuto un salto nella fase di decode viene prelevata anticipatamente (pre-fetching) la sua destinazione, l'indirizzo è mantenuto fino a quando si dovrà eseguire l'istruzione di salto.

Buffer circolare (loop buffer)

(Un buffer è come un vettore che contiene indirizzi) Piccola e veloce memoria che contiene le ultime n istruzioni prelevate con il fetch, se occorre effettuare un salto l'hardware prima controlla se la destinazione si trova nel buffer, in caso affermativo la successiva istruzione viene prelevata dal buffer. Anticipando il fetch è possibile avere già nella memoria circolare alcune istruzioni evitando di dover fare riferimento in memoria alcune volte;
Nei cicli, se le istruzioni del suo intero riescono a stare nel buffer circolare si ha un massimo vantaggio

Predizione di salto (branch prediction):

Soluzione più utilizzata nel lato pratico, si può prevedere di:

- Saltare sempre (approccio statico)
- Saltare mai (approccio statico)
- Saltare in base al codice operativo (approccio statico): il codice operativo guida la decisione se eseguire o meno un salto; il processore può anticipare tramite predizione e poi confermare il salto effettivo solo in fase di esecuzione, applicando la logica necessaria alla condizione specifica dell'istruzione di salto
- Bit taken/not taken (approccio dinamico): un istruzione di salto condizionato in cache viene associato un bit, che memorizza il comportamento recente di quella istruzione e questa informazione viene utilizzata per anticipare il comportamento dell'istruzione di salto
- Tabella di predizione dei salti o branch prediction (approccio dinamico): Viene usata una tabella con la storia dei salti e ciascuna riga contiene:
    1. L'indirizzo dell'istruzione di salto

2. Un certo numero di bit di storia
3. Informazioni circa l'istruzione destinazione (di solito il suo indirizzo)

Multi processori (Parallelismo a livello di processore)

Un multi processore è una serie di processori che condividono una RAM, serve che tutte i processori siano sincronizzati (devono svolgere compiti che non vanno in conflitto) per evitare conflitti nelle operazioni di memoria condivisa.

1. Le CPU si contendono il bus
2. Le CPU non possono eseguire due istruzioni contemporaneamente sullo stesso dato poichè ogni CPU ha i suoi registri temporanei delle azioni che esegue, le due o più CPU quindi devono essere sincronizzate per evitare conflitti sulla memoria durante le 3 fasi di ciclo in cui la memoria viene utilizzata.

Multi computer

Più CPU dotate di una memoria privata nel quale vengono contenuti dati che non sono condivisi ma utili all'elaborazione, il minor scambio sul bus rende più veloce l'esecuzione e l'area condivisa è utilizzata per contenere, ad esempio, il codice del programma da eseguire. Le CPU sono **debolmente connesse**, la loro comunicazione avviene attraverso messaggi che si scambiano sul bus

Symmetric Multi Processing (SMP)

Un calcolatore con molti processori, tutti i processori sono identici e sono tutti in grado di eseguire le stesse istruzioni (o ISA), ognuno di loro ha la loro L1,L2 (destinata allo specifico core ma non al suo interno) e la L3 che è condivisa. Ogni processore può eseguire la stessa porzione di sistema operativo.
Presenta diversi vantaggi:

- Parallelismo: i thread possono essere schedulati su tutti i processori
- Disponibilità: se i core sono tutti uguali e tutti che possono eseguire la stessa istruzione, per un processo è sufficiente trovare un core libero
- Crescita incrementale: Siccome sono tutti uguali tra loro basta aggiungerne più CPU (NB: i vantaggi sono potenziali ma non garantiti, dipende dalla capacità del sistema operativo di sfruttare tutti i core)
- Presenza di più processori trasparenti all'utente (ottimizzazione per più CPU)

Vantaggio numerico di un Multi Processore

Sia P un processo e supponiamo che sia costituito da una frazione seriale f_s (realizzabile in un tempo T_s) e da una frazione f_p realizzabile in parallelo (realizzabile in un tempo T_p) dove ovviamente $f_s + f_p = 1$.

Sia T_1 e T_n rispettivamente il tempo di esecuzione del processo su 1 ed n processori:

$T_1 = T_s + T_p$ (può essere considerato normalizzato a 1)

$T_n = T_s + T_p/n$

Lo speed-up S(n) con n processori è pari a T_1/T_n (Legge di Andahl)

$$S(n) = \frac{T_1}{T_n} = \frac{1}{f_s + \frac{f_p}{n}} = \frac{1}{f_s + \frac{1 - f_s}{n}} = \frac{n}{nf_s + (1 - f_s)}$$