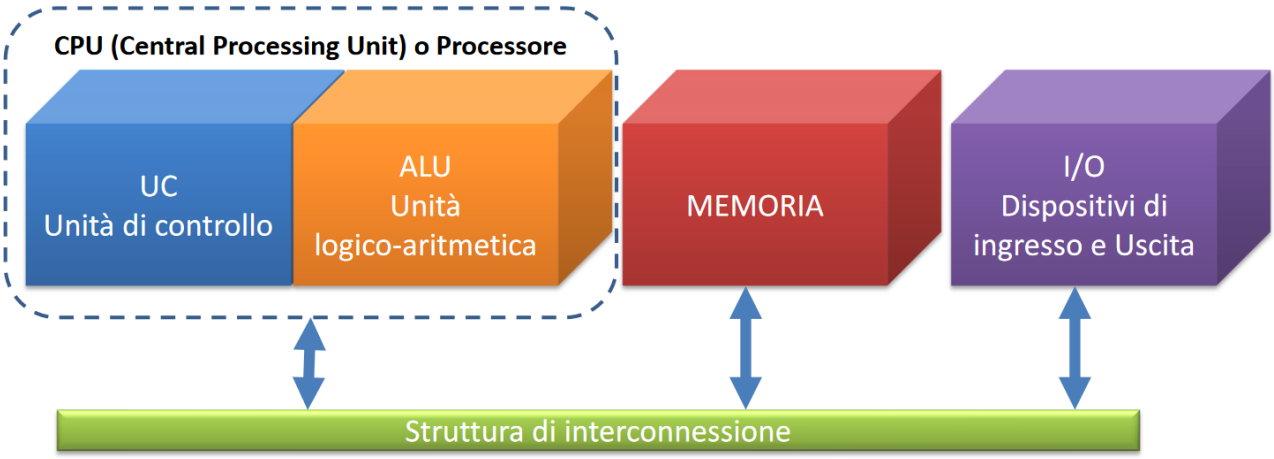
ARCHITETTURA DEGLI ELABORATORI

SOMMARIO

MACCHINA DI VON NEUMANN

La macchina di Von Neumann è la prima macchina programmabile, ancora oggi è il punto di riferimento per la progettazione di un qualsiasi elaboratore elettronico.

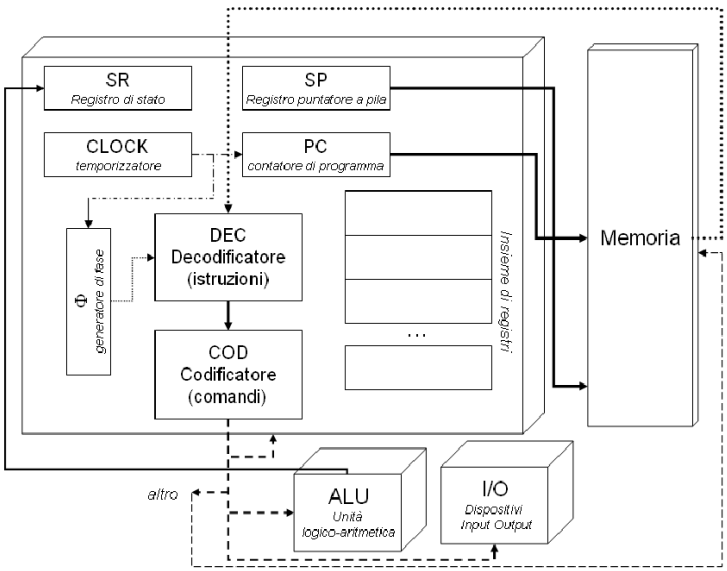
Prevede la presenza di quattro componenti diverse:



UNITÀ DI CONTROLLO (UC)

L'unità di Controllo (Control Unit) è predisposta a scandire le sequenze di operazioni elementari necessarie per eseguire le singole istruzioni.

Le istruzioni vengono prelevate dalla memoria centrale, e trasferite alla circuiteria interna all'Unità di Controllo che riconosce il tipo di dato e genera comandi specifici per l'esecuzione delle istruzioni



REGISTRI UNITÀ DI CONTROLLO

Nell'Unità di Controllo sono presenti dei registri che possono essere classificati ad uso **generale** e ad uso **speciale**.

I registri ad **uso speciale** sono:

- **Contatore di Programma (*Program counter*)**: Contiene un valore numerico corrispondente ad un indirizzo della cella di memoria dove è memorizzata l'istruzione da eseguire
- **Registro di Stato (*Status Register*)**: Contiene le informazioni relative allo stato della Control Unit dopo l'esecuzione di un'istruzione
- **Puntatore alla pila (Stack Pointer)**: Contiene l'indirizzo della cima della pila, cioè la zona di memoria usata per il passaggio di parametri tra funzioni

L'insieme dei registri ad **uso generale** è anche denominato **File Register (FR)**, tali registri vengono usati per memorizzare i risultati temporanei provenienti dall'ALU e le informazioni di controllo, allo scopo di diminuire il numero di accessi alla memoria centrale e velocizzare l'elaborazione

GENERATORE DI FASE

Un generatore di fase è tipicamente realizzato con un Program Counter, ha il compito di scandire le fasi delle operazioni elementari eseguite dalla Control Unit, ci sono tre fasi:

- **Caricamento (Fetch)**: lettura dalla memoria della parola puntata dal Program Counter. In questa fase la Memoria Centrale e il Codificatore presente nella CU si connettono per trasferire l'istruzione
- **Decodifica (Decode)**: riconoscimento del tipo di istruzione e del modo di riferimento degli operandi. In questa fase non vi è una connessione con componenti esterni perché il decodificatore è interno alla CU
- **Esecuzione (Execute)**: esecuzione dei comandi definiti dal codice operativo dell'istruzione. In questa fase vi è una connessione con tutte le unità richieste. Mentre le prime due fasi sono uguali per ogni istruzione, la fase di esecuzione varia in relazione dell'operazione coinvolta (Es. una moltiplicazione impiega più tempo di una addizione)

TEMPI DI ESECUZIONE DELLE ISTRUZIONI

Un'istruzione è sempre eseguita in queste fasi ma comprende un numero di cicli macchina variabile dipendenti, ad esempio dal tipo di operazione, dal numero di accessi in memoria o alle unità di I/O.

Ogni ciclo macchina è implementato mediante una successione di un piccolo numero di operazioni elementari eseguite in circuiti diversi sotto il controllo del *Transcodificatore*.

Ogni operazione elementare occupa un periodo di clock e pertanto la durata di un'istruzione dipende dal numero di accessi alla memoria, all'esterno della CPU e dal numero di operazioni elementari richieste.

Istruzione	**Significato**	**Tempo di esecuzione**
BNE	Salto condizionato	3Δt
LOAD <imm>	Caricamento di un valore	2Δt
DEX	Decremento di un'unità	3Δt
NOP	Nessuna operazione	2Δt

Una volta che l'istruzione è stata caricata viene passata al **Transcodificatore** (ovvero il decodificatore delle istruzioni connesso al codificatore dei comandi) che riconosce l'istruzione e genera opportuni comandi per eseguire l'istruzione stessa.

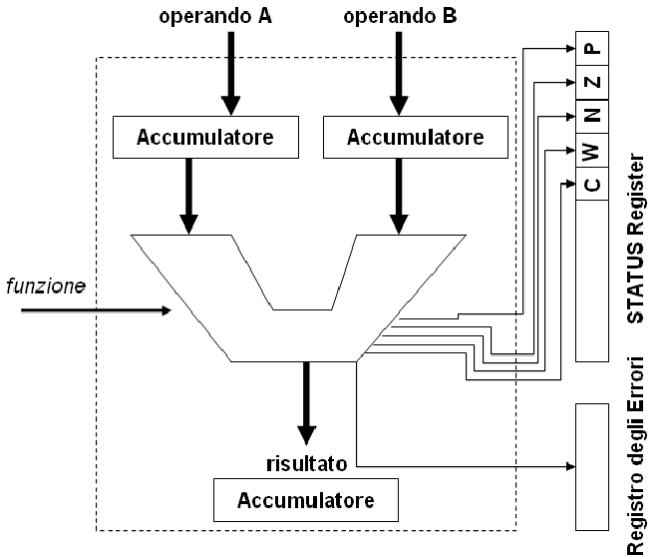
UNITÀ LOGICO-ARITMETICA (ALU)

Per il funzionamento di questa componente, di solito sono impiegati un insieme di registri ad uso speciale, contenenti l'insieme di operandi e il risultato delle varie operazioni, denominati **accumulatori**, i quali non possono essere modificati dal programmatore tramite istruzioni di alcun tipo. Gli accumulatori hanno un ruolo fondamentale per *l'indirizzamento implicito*.

Sono presenti, inoltre, delle linee di ingresso che individuano la funzione/operazione che deve essere attivata e delle linee di uscita su cui è ricondotto il risultato.

Le informazioni relative all'ultima operazione eseguita vengono riportate dai **flags** o **condition code** (linee di uscita).

Oltre agli accumulatori la ALU è provvista anche di un **registro degli errori** contenente le operazioni non risolvibili, grazie ad essa possiamo attivare un'interruzione interna

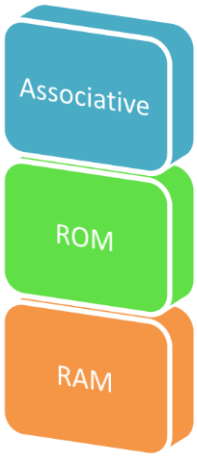


Col tempo si è provveduto a realizzare ALU specifiche, denominate **co-processori matematici**, che eseguono funzioni complesse come i calcoli in virgola mobile con un set di istruzioni dedicato estraneo al set di istruzioni della macchina.

Inoltre possiamo utilizzare un ALU attaccata per eseguire operazioni diverse dall'ALU nativa, in questo caso l'ALU attaccata è vista come un dispositivo I/O.

MEMORIA

La memoria può essere classificata per la sua tipologia in tre macrocategorie:



- **MA** → Associative Memory
- **ROM** → Read Only Memory
- **RAM** → Random Access Memory

RAM (RANDOM ACCESS MEMORY)

Tipo di memorie **Volatili**, cioè che perdono l'informazione a seguito di uno spegnimento dell'alimentazione

L'accesso ad ogni locazione impiega tempo costante $\Theta(1)$

- **SRAM** (Static RAM) → Memoria statica dove l'informazione viene memorizzata in un equivalente di un D latch.
  Sono solitamente usate per le memorie *cache*, dove elevate velocità e ridotti consumi sono caratteristiche fondamentali
- **DRAM** (Dinamic RAM) → Memoria dinamica nelle quali l'informazione è memorizzata in un condensatore, il quale viene ricaricato grazie a dei *refresh* periodici effettuati dal sistema.
- **SDRAM** (Synchronous DRAM) → consente una maggiore flessibilità di impiego, permettendo di modificare il dato contenuto nella cella utilizzando il vecchio dato memorizzato.
  Si differenziano dalle DRAM normali per il fatto che l'accesso è sincrono, ovvero governato dal clock.

ROM (READ ONLY MEMORY)

Memoria di tipo **non Volatile** che mantengono l'informazione indipendentemente dalla presenza di alimentazione. Sono memorie programmate dal costruttore ma non modificabili dall'utilizzatore.

Anche in questo caso l'accesso ad una qualunque locazione di memoria avviene in tempo costante

- **PROM** (Programmable ROM) → Memorie dove è possibile scrivere i dati una sola volta, utilizzando un apposito dispositivo che brucia fusibili.
- **EPROM** (Erasable Programmable ROM) → Memorie dove è possibile scrivere i dati più volte, utilizzando dispositivi di cancellazione (a raggi ultravioletti).
- **EEPROM** (Electrically Erasable PROM) → Memorie EPROM dove è possibile eseguire la cancellazione per via elettrica, ma a livello globale.

MEMORIA CENTRALE

La **memoria centrale** è una memoria volatile di tipo **DRAM** (tipologia DDR), le quali consentono di operare *non solo* quando il segnale del clock è alto, ma anche quando è basso *raddoppiando la velocità di trasferimento* costituita da tante locazioni (celle) ciascuna delle quali può immagazzinare una stringa di n bit finiti, che rappresentano istruzioni, operandi o indirizzi.

La Memoria Centrale presenta delle aree riservate, in cui risiedono delle informazioni basilari utili al funzionamento della macchina (kernel del Sistema Operativo), ed altre in cui, per comodità sono riservate per operazioni particolari (Stack, zona per trasferimento I/O…).

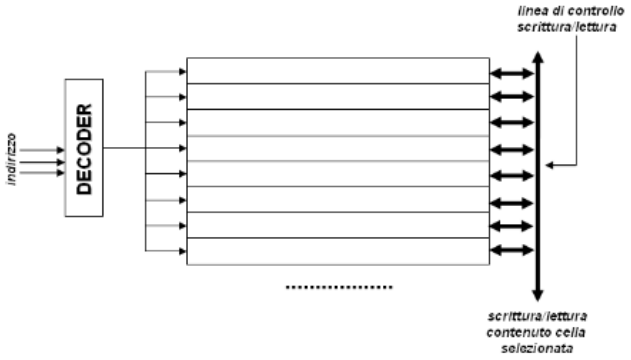
Per motivi progettuali la Memoria Centrale ha locazioni di memoria lunghe 8 bit, questo tipo di progettualità permette inoltre il prelievo di dati di tipo Byte, Halfword, Word.

Per poter interagire con la memoria centrale, è necessario che ci siano:

- **Linee di ingresso**: specificano un indirizzo
- **Linee di uscita**: inviano o trasferiscono il dato
- **Segnale di Controllo**: viene generato dalla CU, serve per la lettura o la scrittura del dato

ORGANIZZAZIONE FISICA

C'è il bisogno di costituire un'architettura basata su un **decoder** che riceve in ingresso l'indirizzo di locazione di memoria alla quale si vuole accedere ed una linea che abilita questa cella a porre il suo contenuto in uscita dalla memoria.

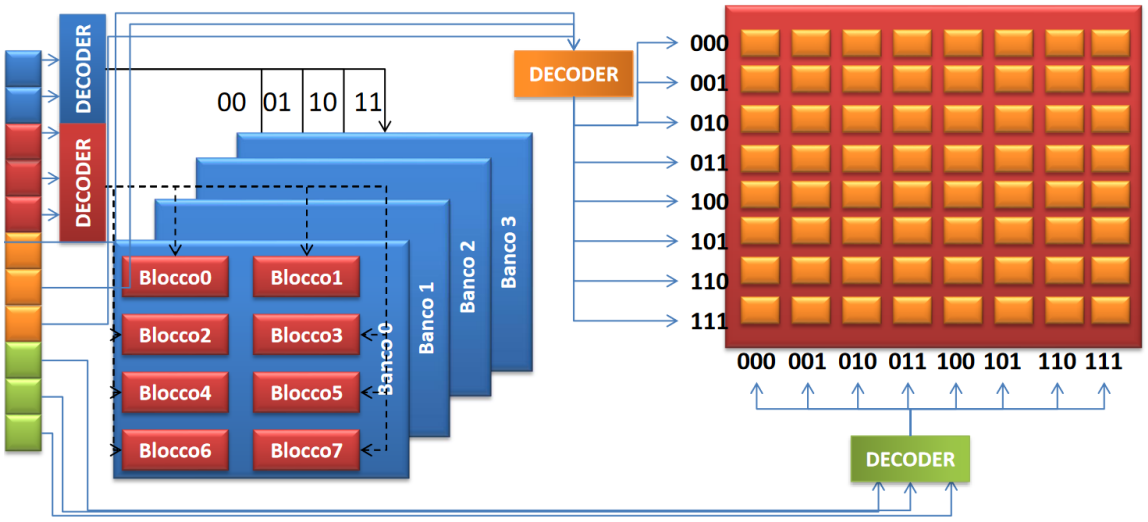


Un'organizzazione di questo tipo però diventa **impraticabile** nel caso in cui la memoria abbia una grande dimensione, infatti con indirizzi di lunghezza *m*, è possibile indirizzare 2^m celle di memoria

Dal 2010 si usano sistemi con parole lunghe 32 bit, avremmo quindi 2^{32} celle di memoria, un decoder con 32 linee di ingresso e circa 4 miliardi di linee di uscita è impensabile

Per ovviare a questo limite, si ricorre ad una **suddivisione logica** della Memoria Centrale, quest'ultima viene suddivisa in **banchi** e **blocchi**, in questo modo l'indirizzo è suddiviso in campi ognuno con

significato associato a banchi, blocchi e locazioni presenti, per ogni campo inoltre è presente un decoder, in questo modo si risparmia dal 90% al 99% in termini di linee di connessione.



I dati in memoria possono avere due tipi di ordinamento Endian:

- Big Endian: la numerazione comincia a partire dal bit più significato (MSB)
- Little Endian: la numerazione comincia dal bit meno significativo (LSB)

DISPOSITIVI I/O

I dispositivi I/O consentono di collegare l'elaboratore, ed in particolare la Memoria Centrale con dispositivi esterni.

Le velocità di trasferimento sono inferiori a quelle possibili all'interno delle altre componenti a causa della natura tecnologica di fabbricazione, comportando problemi di sincronizzazione e adattamento velocità.

Dispositivo	Comportamento	Dati scambiati (Kb/s)
Tastiera	Input	0,01
Mouse	Input	0,02
Stampante ad aghi	Output	1
Floppy	Storage	50
Stampante Laser	Output	100
Disco	Storage	10.000
LAN	Input/Output	10.000
Display	Output	30.000

Quando vi è presente un **trasferimento dati** è importante che il tutto avvenga seguendo un **protocollo**, consentendo l'interazione sotto il controllo del processore.

→ **Protocollo di input**: la periferica vuole inviare dati all'elaboratore in Memoria Centrale

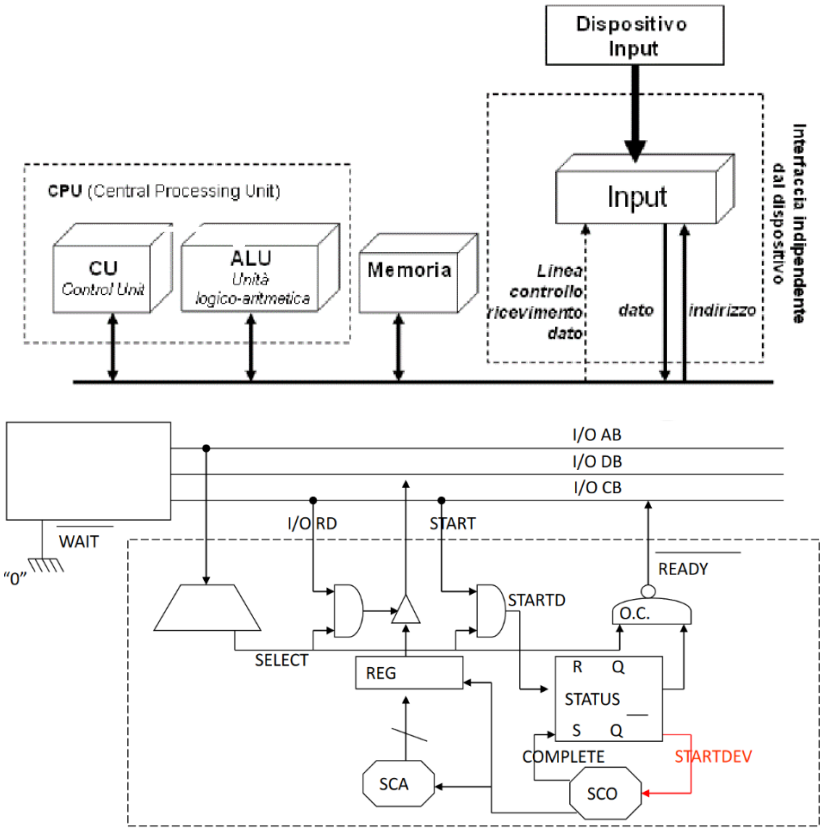
1. Individuazione del dispositivo di input che vuole inviare dati
2. Ricerca di un'area libera di memoria
3. Prelievo del dato

→ **Protocollo di output**: la periferica ospita i dati prodotti dall'elaboratore e li ri-elabora in relazione alla propria funzione

1. Individuazione del dispositivo che deve ricevere i dati
2. Controllo sul dispositivo
3. Invio dei dati

Per interagire con il Processore, ogni dispositivo deve essere interconnesso ad un **modulo di I/O**, cioè una rete sequenziale che comunica con il processore inviando e ricevendo i segnali di trasferimento.

Di seguito vediamo uno schema di un dispositivo di **input**, si evidenzia la sottorete indipendente dal dispositivo, la quale è incaricata di comunicare col processore.

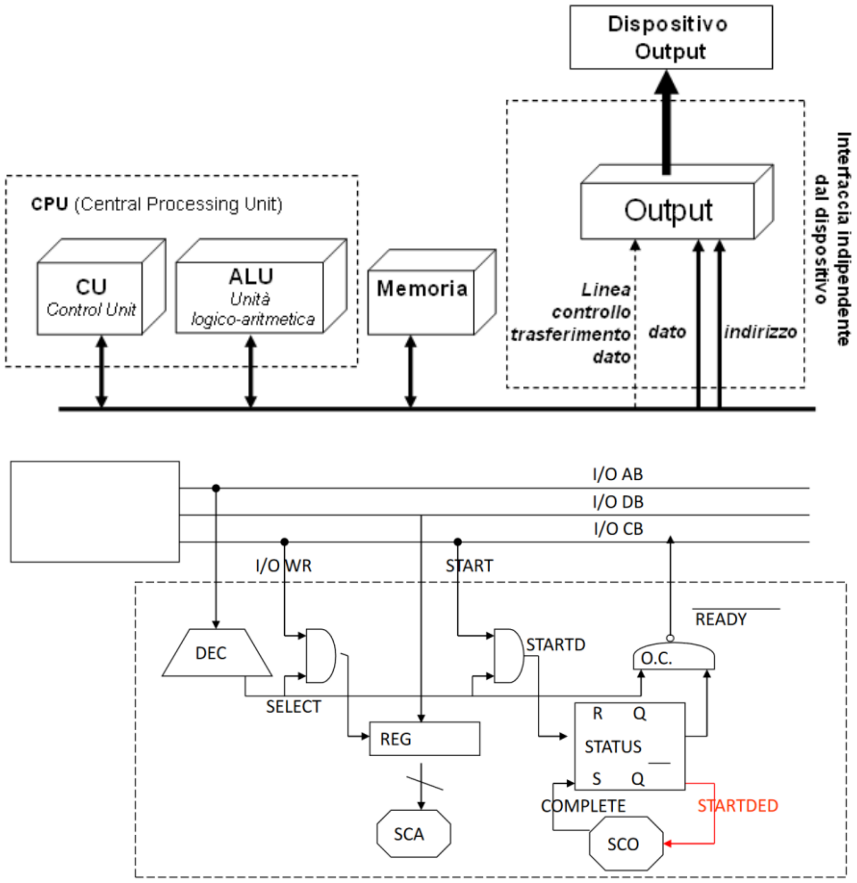


Protocollo Input:

1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato.
2. Se il dispositivo non è pronto il processore deve attendere e tornare al punto 1.
3. Il processore avverte il dispositivo che può prendere un dato (seleziona il dispositivo tramite le linee indirizzi e invia il segnale START). Questo segnale resetta il flip-flop STATUS e rimane in tale stato per tutta la durata delle operazioni.
4. Quando il dato è stato prodotto ed è disponibile in REG, il dispositivo genera il segnale COMPLETE, settando STATUS a 0.
5. Nel frattempo, il processore, in attesa del dato, esamina il flip-flop STATUS campionando il segnale READY.
6. Se READY=1 il processore deve attendere e tornare al punto 5, altrimenti il processore invia il segnale IO/RD per trasferire il dato presente in REG all'interno della locazione di memoria libera.

Protocollo Output:

1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato tramite la linea di controllo READY.
2. Se il dispositivo non è pronto il processore deve attendere e tornare al punto 1, o svolgere un ulteriore istruzione. Se è pronto va al prossimo punto.
3. Se READY=0 il processore trasferisce il contenuto di una locazione di memoria nel registro di interfaccia del dispositivo
4. Il processore avverte il dispositivo che gli ha trasferito un dato inviando il segnale START. Anche qui il segnale START resetta il flip-flop STATUS e rimane in questo stato fino al termine delle operazioni. Quando il dato è stato letto da REG, il dispositivo genera segnale COMPLETE, settando STATUS a 0.
5. Nel frattempo, il processore esamina lo stato di STATUS campionando il segnale READY.
6. Se READY=1, il processore deve attendere e tornare al punto 5, altrimenti può eseguire un'altra istruzione.



Per indirizzare le unità di I/O e consentire l'accesso al dato da trasferire/recuperare, il processore ricorre a due tecniche:

- **I/O Canonico**: Riservare all'I/O uno spazio di indirizzamento indipendente
- **I/O Programmato**: Riservare una porzione dello spazio di indirizzamento in Memoria Centrale, in modo che ogni volta che il processore utilizza un indirizzo di questa porzione, in realtà capisce di far riferimento ad un dispositivo I/O.

ELEMENTI DI INTERCONNESSIONE

Le informazioni elaborate da un calcolatore elettronico prendono in considerazione delle stringhe binarie (*word*) che hanno il significato di operando, indirizzo o istruzione, le quali hanno una dimensione prefissata che deve essere considerata come un'unità indivisibile.

Le singole cifre costituenti una parola sono memorizzate in **latch** e l'insieme risultante è un registro.

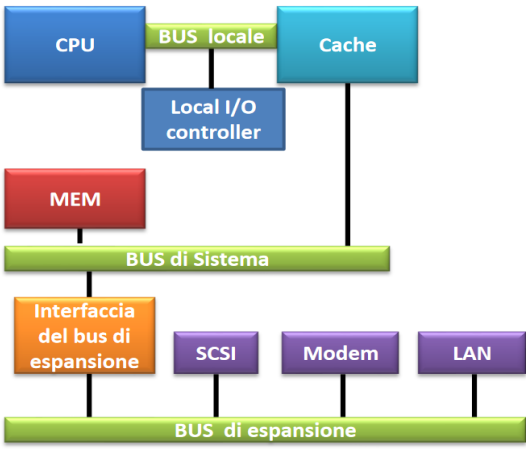
	Sorgente Prefissata	Sorgente Variabile
Destinazione Prefissata	Punto a punto	Multiplexer
Destinazione Variabile	Demultiplexer	Mesh, Bus

Ci sono diverse tipologie di Interconnessione:

- L'interconnessione da **Punto a Punto** effettua il trasferimento della parola contenuta in un registro sorgente (Rs) a un registro destinazione (Rd).
- Il **multiplexer** è la rete d'interconnessione che consente il trasferimento tra *m* registri sorgenti e un registro destinazione prefissato.
- Il **Demultiplexer** è la rete di interconnessione atta a favorire il trasferimento tra un registro sorgente e uno degli m registri destinatari
- Le **reti mesh** sono reti in grado di interconnettere tra loro *m* registri, o più in generale *m* componenti
- Il **bus** è un fascio di k linee. Per il trasferimento è sufficiente attivare la linea di ingresso di selezione (s) del registro sorgente e quella del registro destinazione.

Il bus sfrutta un **buffer tri-state**, un dispositivo usato per permettere a più porte logiche di pilotare la stessa uscita, generalmente un bus. Si utilizza un segnale di attivazione del buffer tristate, se acceso consente il passaggio dei dati da IN a OUT, se spento non consente il passaggio dei dati.

I primi elaboratori avevano un unico bus chiamato anche **bus di sistema**. Esso era composto dai 50 ai 100 fili paralleli di rame che si inserivano nella *scheda madre* e i cui connettori erano distanziati a intervalli regolari per permettere l'inserimento di memorie e schede di I/O.



Attualmente gli elaboratori utilizzano più bus (**multi-bus**): uno specifico tra la CPU e la Memoria Centrale e almeno un altro per le periferiche.

Alcune periferiche che si collegano al bus sono attive (**master**) e possono iniziare un trasferimento dati, mentre altre sono passive (**slave**) e restano in attesa di una richiesta. Inoltre, sappiamo che la Memoria Centrale non può mai svolgere la funzione di Master.

Master	Slave	Esempio
CPU	Memoria	Prelievo delle istruzioni e dei dati
CPU	Dispositivo I/O	Inizio del trasferimento dei dati
Dispositivo I/O	Memoria	Scambio di dati
Coprocessore Matematico	CPU	Prelievo

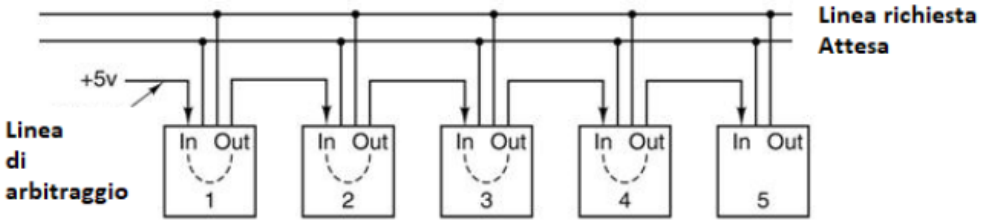
Il numero di linee che costituisce il bus influenza la progettazione della macchina (costi, organizzazione topologica…)

Per aggirare il problema di bus multipli si può usare un **bus multiplexato**. In questa architettura invece di tenere separate le linee d'indirizzo e quelle dei dati, si utilizza un certo numero di linee per entrambi: all'inizio di un'operazione sul bus le linee sono utilizzate per gli indirizzi, mentre in seguito vengono impiegate per i dati.

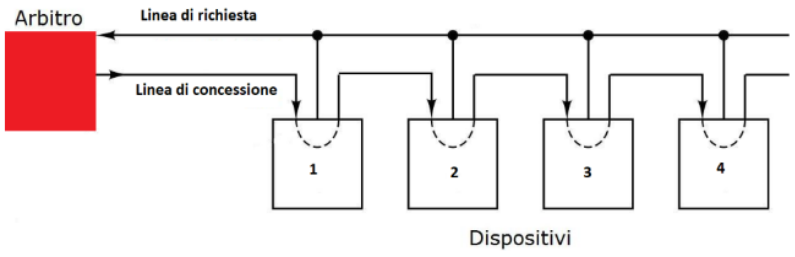
I bus separati sono molto *rapidi*, ma molto *costosi*. Il bus multiplexato è *lento* ma *economico*.

Nel caso di un solo bus e di due o più dispositivi che richiedono contemporaneamente l'uso del bus, si ricorre ad un arbitraggio del bus. Questo arbitraggio può essere:

- **Decentralizzato**: Si usano *più linee di richiesta*, ciascuna con la sua priorità. Rispetto al metodo centralizzato questo schema di arbitraggio richiede un maggior numero di linee di bus, ma evita il potenziale costo dell'arbitratore. Un altro limite è che il numero di dispositivi non può superare il numero delle linee di richiesta



- **Centralizzato**: Un arbitro del bus determina chi è il prossimo dispositivo, sfrutta il concetto del ***Daisy chaining***, ovvero controlla solo se è presente una richiesta e, se presente, abilita la linea del bus e mano mano che arriva un dispositivo richiedente, utilizza la linea.
  Per evitare una Starvation possono essere usate delle *linee di priorità*.

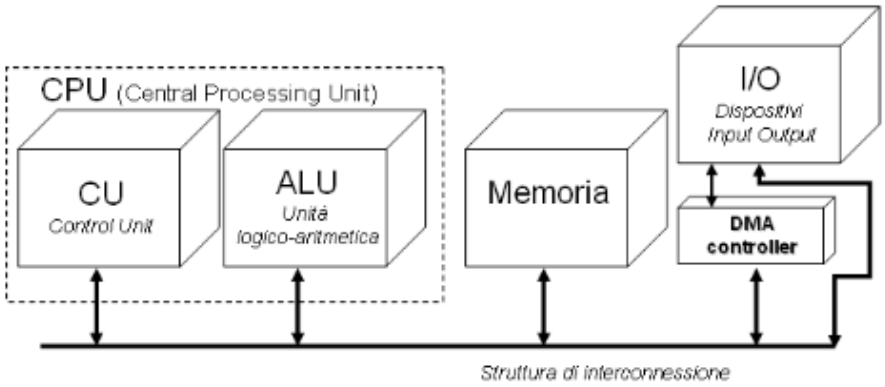


Con il *daising chain* si riscontra una perdita significativa del tempo in cui la CPU potrebbe essere coinvolta in elaborazioni più rilevanti come, ad esempio, l'esecuzione di una istruzione di un programma.

Un *sistema a interruzione* risulta essere efficiente se il trasferimento è per pochi dati, ma diventa effimera se il trasferimento dati coinvolge un gran numero di parole.

Per superare questi limiti, si utilizza un **DMA** (Direct access memory), il quale permette il trasferimento autonomo dei dati tra un dispositivo I/O e la memoria centrale con l'obiettivo di ridurre il carico sulla CPU e migliorare quindi l'efficienza del sistema.

Viene implementato grazie all'utilizzo di un supporto hardware dedicato, chiamato **DMA controller**.



Le componenti fisiche di base del DMA Controller sono:

• **Current Address Register** (CAR): Registro che indica la locazione di memoria destinata a ricevere o fornire il prossimo dato da trasferire. A volte diviso in:

- *Source Address Register* (SAR)
- *Destination Address Register* (DAR)

• **Word Counter** (WC): Contatore dei dati da trasferire.

• **Registro di Stato**: Contiene informazioni sullo stato del trasferimento.

• **Linee di Comando**: Gestiscono la richiesta e l'assegnazione del bus per evitare conflitti di accesso.

I passi elementari che devono essere svolti per eseguire un trasferimento possono essere sintetizzati come segue:

1. **Accesso alla Posizione di Memoria**: Utilizzando il registro CAR, si accede alla posizione di memoria interessata.
2. **Trasferimento dei Dati**: I dati vengono trasferiti lungo il bus.
3. **Incremento/Decremento del CAR**: Il registro CAR viene aggiornato per puntare alla prossima locazione di memoria.
4. **Decremento del WC**: Il contatore WC viene decrementato.

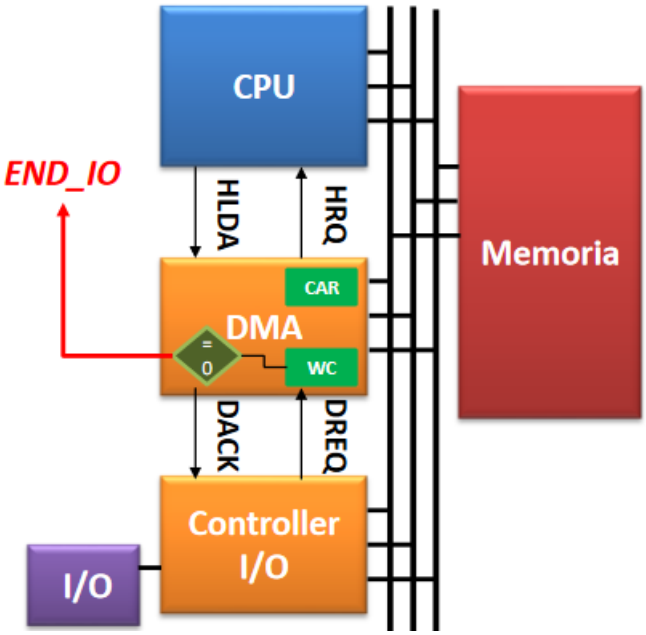
Vi è infine un *Interruzione di Fine Trasferimento*, ovvero quando il WC raggiunge lo zero, viene richiesta un'interruzione per segnalare il completamento del trasferimento

Protocollo di Trasferimento

1. **Richiesta di Salvataggio dell'Utente**: L'utente richiede il salvataggio di un file, avviando il processo.
2. **Routine di Interruzione**: La routine di salvataggio individua il dispositivo di output, inizializza i registri CAR e WC, e invia un segnale START_IO al dispositivo di output.
3. **Operazioni tra Memoria, DMA Controller e Dispositivo di Output**:
   - *DREQ* (DMA Request): Il dispositivo di output richiede l'uso del DMA settando i parametri necessari.
   - *HRQ* (Hold Request): Il DMA controller richiede alla CPU l'uso del bus.

- *HLDA* (Hold Acknowledgement): La CPU rilascia il bus al DMA.
- Indirizzo sul Bus degli Indirizzi: Il DMA invia l'indirizzo CAR lungo il bus degli indirizzi.
- *DACK* (DMA Acknowledgement): Il DMA invia un segnale di conferma al dispositivo di output, consentendo il trasferimento dei dati.
- *Aggiornamento dei Registri*: Il CAR viene aggiornato e il WC decrementato dopo ogni trasferimento.

4. **Completamento del Trasferimento**:
   - Quando il WC arriva a zero, il DMA controller disattiva HRQ e la CPU riprende il controllo del bus.
   - Viene generata una richiesta di interruzione END_IO (o DMA_INT).
5. **La routine di servizio aggiorna lo stato del sistema**, liberando le risorse utilizzate



Le operazioni di accesso al bus di memoria avvengono in *intervalli temporali* di durata corrispondente al tempo di accesso in memoria stessa. Ciascuno di questi intervalli può essere utilizzato sia dal processore (ad esempio nella fase di LOAD) sia dai dispositivi.

La logica di assegnazione del bus è conglobata nel processore e realizzata con la modalità **master slave**, tale cioè che il processore ne abbia il controllo (master) e ne rilascia l'uso ai dispositivi (slave) quando questo ultimo ne fa richiesta.

Nel caso in cui ci siano **conflittualità**, il processore rimanda ad un ciclo successivo la propria operazione di accesso alla memoria *privilegiando* le richieste di DMA (cycle stealing, furto di un ciclo).

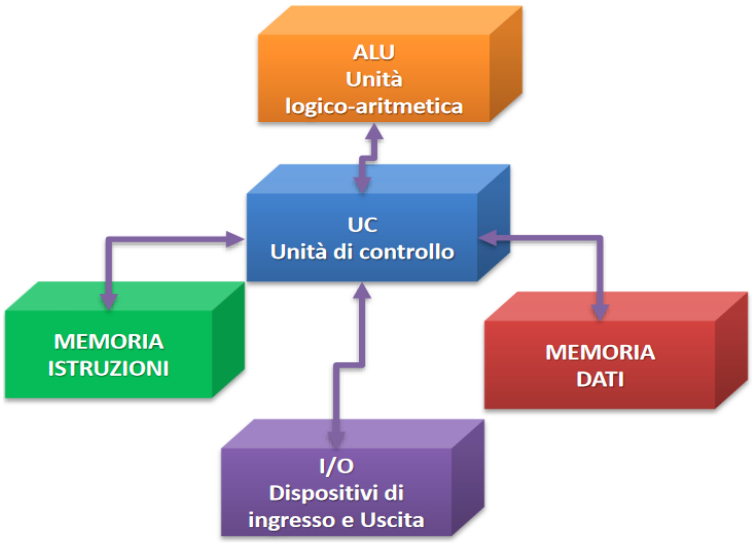
Nelle attuali architetture il furto di ciclo è un *caso superato* perché è possibile accedere direttamente alle celle di memoria e in modalità parallela grazie all'uso di **bus dedicati** e all'introduzione della **cache**.

Un caso reale è il MOTOROLA MCF5307.

MACCHINA DI HARVARD

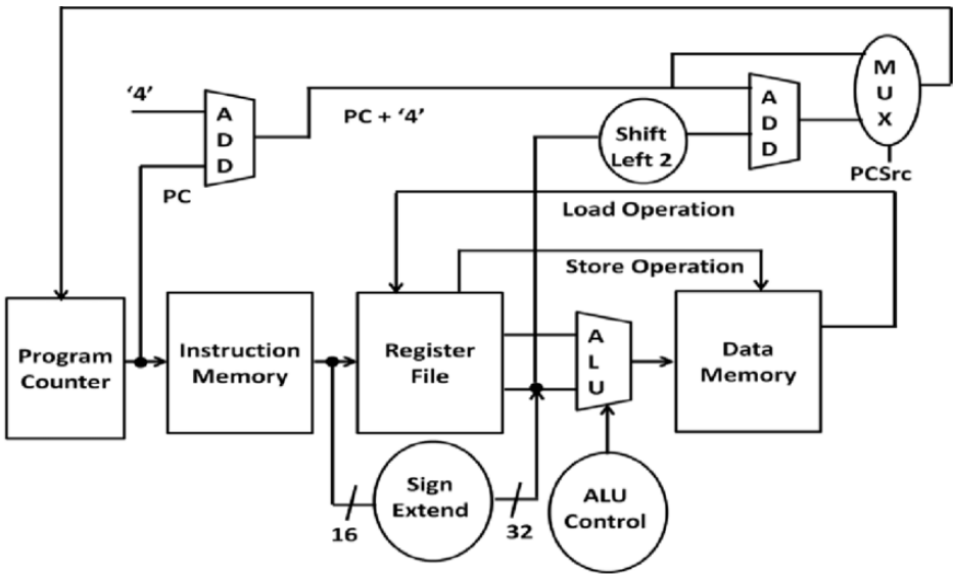
La prima data ufficiale in cui si realizzò un modello di elaboratore elettronico fu il 1944 con la "**Macchina di Harvard**", dal nome del college in cui si trovava il gruppo di lavoro che l'aveva ideata.

La macchina di Harvard era costituita da una Unità di Calcolo, una Unità di Controllo, una Memoria delle Istruzioni, una Memoria Dati ed un modulo per i Dispositivi di input e output, opportunamente collegati per consentire un flusso di comandi e di controlli che ne permettevano il funzionamento e colloquio reciproco e lo svolgimento di una istruzione ad un colpo di clock.



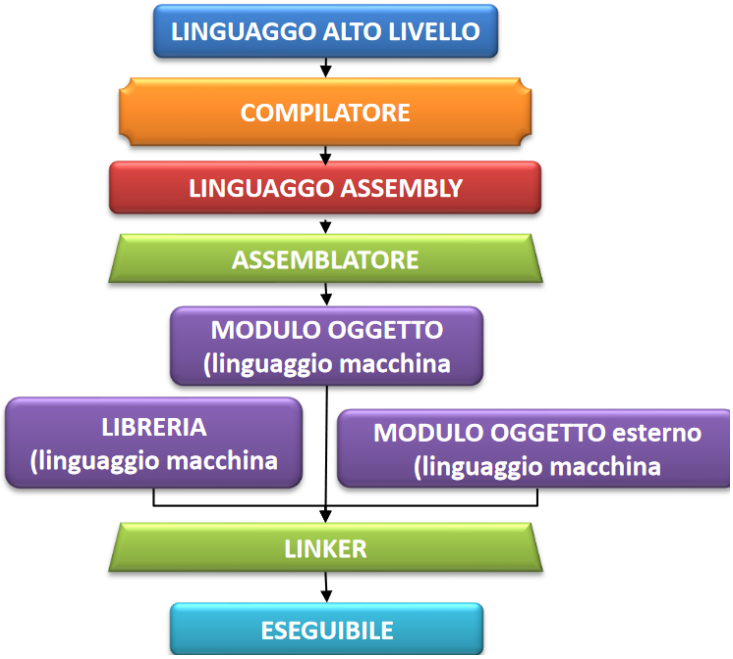
La "**macchina di Harvard**" fu migliorata nel 1982 con un set appropriato, multibus e una serie di registri ad uso generale che consentivano di eseguire ciascuna istruzione di lunghezza fissa a 32bit in un solo colpo di clock.

La macchina di Harvard avrà quindi un codice più lungo ma sarà molto più rapida nell'eseguire l'istruzione poiché le singole operazioni vengono svolte in minor tempo e, nel caso di più istruzioni in coda, le operazioni svolte in poco tempo permettono l'esecuzione dell'operazione in coda avendo liberato la priorità ad esse.



PROGETTAZIONE DI UN PROGRAMMA

L'esecuzione di un programma è il punto di arrivo di una sequenza di azioni che nella maggior parte dei casi iniziano con la scrittura di un programma in un linguaggio simbolico di alto livello. Le azioni principali che compongono tale sequenza nel caso si parta da un linguaggio ad alto livello sono quelle che vedono in gioco il compilatore, l'assemblatore e il collegatore.



COMPILATORE

Il compilatore **trasforma**, dopo un controllo sintattico, il programma scritto in un linguaggio ad alto livello in uno in linguaggio assembly, cioè in una forma simbolica che il calcolatore è in grado di capire ma, ancora, non eseguire.

Durante la generazione del codice, il compilatore **riordina** le istruzioni, cioè, quali istruzioni sono trasmesse al processore e in quale ordine (utile nella canalizzazione o per il calcolo parallelo).

Infine, il compilatore **ottimizza** il codice, ovvero toglie le istruzioni inutili o variabili non utilizzate.

ASSEMBLATORE

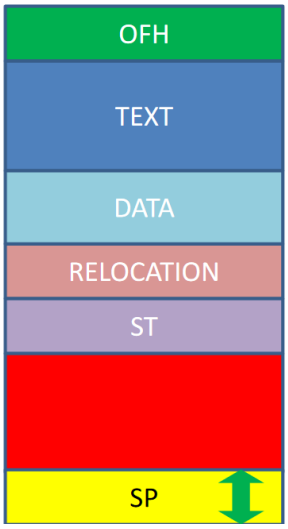
L'assemblatore **converte** un programma *assembly* in un file oggetto, che è una combinazione di istruzioni in linguaggio macchina, di dati e di informazioni necessarie a collocare le istruzioni in memoria nella posizione opportuna

Un **programma assembly** è tradotto in una sequenza di istruzioni (opcode, indirizzi, costanti, ecc.) attraverso il processo di assemblaggio (assembler) costituito da due passi logici successivi ed in parte indipendenti:

1. Il programma assembly è letto *sequenzialmente*, si identificano le istruzioni e i loro operandi, si calcola la lunghezza e si assegna un indirizzo (relativo) a ciascuna istruzione; inoltre, quando è letto un simbolo, nome e indirizzo sono inseriti in una tabella dei simboli (**symbol table**): nome e indirizzo di un simbolo possono essere inseriti nella symbol table in momenti diversi se un simbolo è usato prima di essere definito

2. il programma assembly è letto *sequenzialmente*, a tutti i simboli è sostituito il valore numerico corrispondente presente nella **symbol table**, a tutte le istruzioni e ai relativi operandi ancora in forma simbolica è sostituito il valore numerico corrispondente

I file oggetto sono suddivisi e disposti in memoria di solito in sei sezioni distinte:



1. *Object file header*: descrive la dimensione e la posizione delle altre sezioni del file oggetto
2. *Text segment*: contiene le istruzioni in linguaggio macchina
3. *Data segment*: contiene tutti i dati che fanno parte del programma
4. *Relocation information*: identifica le istruzioni e i dati che dipendono da indirizzi assoluti e che dovranno essere rilocati dal linker
5. *Symbol table*: contiene i simboli che non sono ancora definiti, ad esempio le etichette che fanno riferimento a moduli esterni
6. *Debugging information*: contiene informazioni per il debugger

Inoltre, ci si riserva dello spazio nel quale può avvenire uno scambio di dati, chiamato **Stack Pointer**.

COLLEGATORE (LINKER)

Quando un programma simbolico è costituito da più moduli contenuti in diversi file sorgenti, il processo di traduzione (compilazione e assemblaggio) è ripetuto per ciascun modulo

I **file oggetto** (object) risultanti devono essere **collegati** (linked) opportunamente tra di loro all'interno di unico file eseguibile, che solo allora può essere caricato in memoria

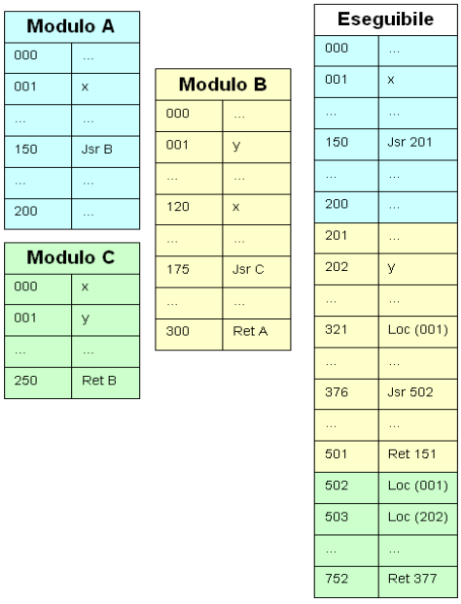
Per ogni modulo tradotto separatamente l'indirizzo iniziale è lo stesso: è compito del linker modificare gli indirizzi di ciascun modulo in modo che non ci siano sovrapposizioni

La **traslazione** dell'indirizzo di ogni istruzione in ciascun modulo permette di unire tutti i moduli ma non è sufficiente; infatti, è necessario traslare in maniera consistente anche tutti gli indirizzi (assoluti) che compaiono come operandi

Per ogni *riferimento* da parte di un modulo a un indirizzo di un altro modulo è necessario calcolare coerentemente l'**indirizzo esterno** (riferimento esterno o external reference). Esempi in questo senso sono le variabili globali (che devono essere viste da tutti i moduli)

Per questo, il linker costruisce una **tabella dei moduli** grazie alla quale è possibile procedere alla **rilocazione** e al **calcolo** dei riferimenti esterni a ciascun modulo

Infine, il linker produce un **file eseguibile** che di norma ha la stessa struttura di un file oggetto, ma non contiene riferimenti non risolti



CARICATORE (LOADER)

Una volta che il file eseguibile è memorizzato sul rapporto di massa (disco magnetico), il caricatore può caricarlo in memoria per l'esecuzione e quindi effettuare diverse azioni:

- Lettura dell'intestazione del file eseguibile per determinare la dimensione dei segmenti testo e dati.
- Creazione di un nuovo spazio di indirizzamento, grande a sufficienza per contenere istruzioni, dati e stack.
- Copia delle istruzioni e dei dati dal file oggetto al nuovo spazio di indirizzamento.
- Copia sullo stack degli eventuali argomenti del programma.
- Inizializzazione dei registri della CPU
- Inizio dell'esecuzione a partire da una direttiva di inizio che copia gli argomenti del programma dallo stack agli opportuni registri e che chiama la **direttiva di inizio** (BEGIN) fino ad una **direttiva di terminazione** (END).

ISTRUZIONI

Un'**istruzione** è una stringa binaria che indica all'elaboratore dei compiti da svolgere, è suddivisa in sottostringhe denominate **campi**, la suddivisione in campi individua il formato dell'istruzione.

I campi principali sono:

1. **Codice operativo** (OPCODE): specifica il tipo di operazione da eseguire
2. **Operando**: indica il dato su cui devono essere effettuate le operazioni indicate dal codice operativo
3. **Riferimento**: indirizzo di memoria in cui è immagazzinato un operando (indirizzamento diretto) o una etichetta che specifica un registro

Generalmente possiamo avere due tipi di formato per le istruzioni, a *lunghezza fissa* e a *lunghezza variabile*.

I processori Intel X86 hanno un formato a lunghezza variabile, dopo l'OPCODE ci sono dei campi che specificano quanti bit appartengono al campo MODE.

Il MIPS ha un formato **a lunghezza fissa a 32 bit**, nel caso in cui si utilizzi un indirizzamento assoluto in cui il riferimento richieda più di 16 bit, l'istruzione viene divisa in due istruzioni elementari.

Le istruzioni sono eseguite quando sono scritte in **linguaggio macchina**.

LINGUAGGIO ASSEMBLATIVO

Il programmatore ricorre ad una rappresentazione simbolica delle istruzioni (codici mnemonici), le **istruzioni assembly**.

La sintassi di un'istruzione assembly è costituita da:

1. **Indirizzo**: dove risiede l'istruzione in memoria
2. **Etichetta** (opzionale)
3. **Istruzione**: formata a sua volta da
    - **Codice mnemonico**: descrivente l'istruzione con pochi, ma significativi caratteri
    - **Modo di indirizzamento**: i dati su cui operare
4. **Commenti**: indispensabile per la comprensione del codice

Indirizzo	Etichetta	Istruzione	Commento
100	Ciclo:	Add $t0, $t1, $t2	#Somma i valori dei registri

L'insieme delle istruzioni assembly definiscono un **linguaggio assemblativo**.

Il legame tra istruzione macchina e istruzione assembly è di **uno a uno**, nel senso che ad ogni istruzione macchina corrisponde una ed una sola istruzione assembly.

Inoltre, per comodità, molti linguaggi utilizzano delle **pseudoistruzioni**, ovvero delle istruzioni che sono composte da una o più istruzioni elementari.

Un linguaggio assemblativo consente di definire delle **macro**, la quale sostituisce una serie di istruzioni. Ogni volta che si richiama una macro, l'assemblatore riscrive le istruzioni definite nella macro.

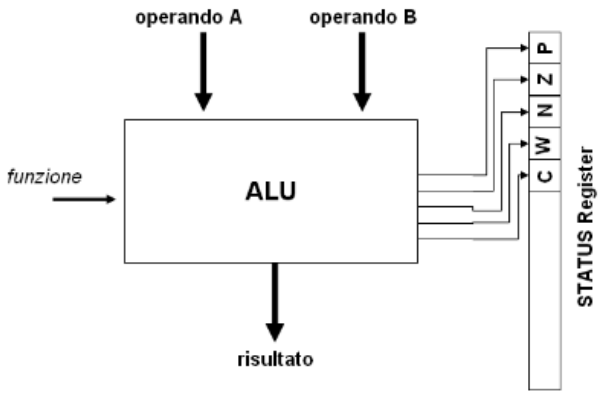
CODICI DI CONDIZIONI

Ad ogni tempo, dettato dal clock, l'elaboratore esegue una istruzione. Ogni istruzione logico-matematica produce dei bit, definiti **flags**, che saranno implicitamente memorizzati nel **registro di stato**.

I Condition Codes svolgono un ruolo fondamentale per le istruzioni di salto condizionato.

I principali flags sono:

- **C – Carry**: individua il trabocco ed è impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un riporto (addizione) o un prestito (sottrazione) a sinistra del bit più significato del risultato, 0 altrimenti
- **N – Negative**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha prodotto un risultato negativo, 0 altrimenti. Quindi Negative è una copia del bit più significativo del risultato
- **Z - Zero**: impostato ad 1 se l'ultima operazione effettuata dall'ALU è nulla, 0 altrimenti.
- **W – Overflow**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha superato la capacità di rappresentazione data dalla lunghezza della parola
- **P – Parity**: impostato ad 1 se l'ultima operazione effettuata dall'ALU ha dato un risultato con un numero pari di "1", 0 altrimenti



Ci sono delle particolari istruzioni che consentono di modificare il contenuto dei registri di stato:

Codice	Commento	Codice	Commento
CLRC	Imposta a 0 il flag C	**SETC**	Imposta a 1 il flag C
CLRN	Imposta a 0 il flag N	**SETN**	Imposta a 1 il flag N
CLRZ	Imposta a 0 il flag Z	**SETZ**	Imposta a 1 il flag Z
CLRW	Imposta a 0 il flag W	**SETW**	Imposta a 1 il flag W

ISTRUZIONI DI SALTO

Le **istruzioni di salto** individuano una classe particolare in quanto non agiscono direttamente sui dati, ma sono utilizzate per modificare l'ordine sequenziale delle istruzioni del programma stesso o uno esterno.
Le istruzioni di salto si dividono in:

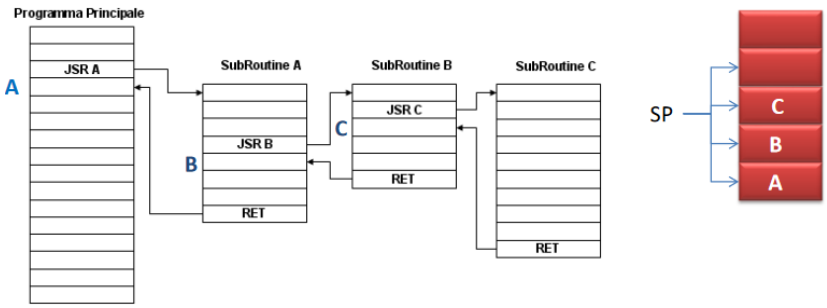
- Salto all'interno dello stesso programma:
  - **Condizionato**: il salto viene eseguito in base ad una certa condizione fissata dal programmatore (Branch)
  - **Incondizionato**: il salto viene sempre eseguito (Jump)
- Salto ad un altro programma: **salto a subroutine**
- **Trap**: interruzione del software

Molto comoda l'istruzione del salto a subroutine poiché permette di saltare da un programma principale ad un sottoprogramma.

In MIPS un salto a subroutine è ottenuto salvando il valore del Program Counter in un registro speciale **$ra** e viene chiamata con "JAL + etichetta".

La decodifica ed esecuzione di una istruzione di ritorno da subroutine è così descritta "JR $ra".
Sono ammesse anche delle subroutine che a loro volta richiamano delle subroutine, denominate **nested subroutine**.

La gestione di funzioni ricorsive o l'annidamenti di funzioni avviene grazie all'uso di una **pila** (stack), una zona di memoria riservata per il passaggio di parametri gestiti tramite una politica **LIFO**



MODI DI INDIRIZZAMENTO

Una istruzione macchina contiene generalmente una suddivisione in campi in cui una parte è **operazionale** (OPCODE) e specifica la classe di istruzione da eseguire ed un secondo **campo indirizzo** (ADDRESS MODE) che indica quale è l'operando, cioè quale è il registro/i o la locazione di memoria che contiene l'operando sul quale si deve applicare l'istruzione.

INDIRIZZAMENTO IMPLICITO

In tale modo non sono esplicitati gli indirizzi dove risiedono gli operandi, ma questi ultimi si trovano in una posizione predeterminata, grazie al **referenziamento implicito** è possibile utilizzare delle istruzioni con **lunghezza fissa e minima**.

L'indirizzamento implicito non può essere utilizzato per le operazioni di trasferimento.

INDIRIZZAMENTO ESPLICITO

Il modo di indirizzamento esplicito consente:

- **Accesso** ad una locazione di memoria il cui indirizzo non è noto nel momento in cui il programma è scritto, ma è calcolato nel momento in cui il programma è eseguito
- **Manipolare** gli indirizzi, cioè, permettere delle operazioni su di essi
- **Calcolo** degli indirizzi relativamente alla posizione dell'istruzione, in modo tale che il programma possa essere caricato in memoria in qualsiasi parte della memoria

Definiamo come **indirizzo effettivo**, l'indirizzo degli operandi interessati. In un'istruzione di salto, per esempio, l'indirizzo effettivo è quello dell'istruzione a cui si vuole saltare.

Esistono diversi modi per indirizzare un certo tipo di dato:

- **Indirizzamento immediato**: così definito perché l'operando si trova nella posizione di memoria immediatamente successiva all'istruzione e pertanto, è nel corpo del programma e non in un'area dati. Un uso eccessivo può incrementare la lunghezza del programma in memoria rendendolo inefficiente. Si usa perlopiù per inizializzazioni
- **Indirizzamento diretto**: specifica l'indirizzo effettivo di una parola di memoria, è pertanto allocato nella parola immediatamente successiva a quella contente l'istruzione che deve operarlo. L'indirizzo è fissato nel momento della in cui si scrive il programma. Utile per operare con un dato contenuto in una posizione di memoria fissata o quando si fa un salto, anch'esso ad una istruzione contenuta in una posizione fissata della memoria
- **Indirizzamento a registro**: specifica l'etichetta del registro in cui è presente l'indirizzo effettivo. Ogni CU è dotata di un certo numero di registri interni detti registri a uso generale. Grazie a questa particolarità, le istruzioni sono eseguite più velocemente per un motivo principalmente: non occorrono accessi alla memoria principale perché i registri sono già all'interno della CU
- **Indirizzamento indiretto**: fa accedere ad un indirizzo effettivo attraverso un indirizzo presente nell'istruzione che punta in memoria ad un altro indirizzo. Utile per condividere delle variabili tra il programma principale e una funzione
- **Indirizzamento indiretto a registro**: prevede che il registro specificato nell'istruzione non contiene l'indirizzo effettivo ma un indirizzo all'indirizzo effettivo. Utile per fare riferimento ad un operando in memoria principale con un'istruzione breve e per modificare l'indirizzo contenuto nel registro per puntare altri dati (scorrimento liste e array)
- **Indirizzamento differito indiretto**: individua l'indirizzo effettivo (un operando) mediante un indirizzo memorizzato in registro presente nell'istruzione che punta in memoria ad un altro indirizzo. Utile per gestire strutture dati
- **Indirizzamento con spiazzamento**: consente di raggiungere l'indirizzo effettivo dopo aver sommato al contenuto di un registro, uno spiazzamento (offset) contenuto nella parola successiva all'istruzione. Utile quando si hanno delle strutture dati con informazioni disposte in maniera sequenziale (stringhe, array, matrici)
- **Indirizzamento relativo**: si fa riferimento al fatto che l'indirizzo è relativo al Program Counter. L'indirizzo effettivo, in questo caso, è dato dalla somma tra lo spiazzamento (offset) presente nell'istruzione ed il contenuto del PC. Utile per realizzare programmi *indipendenti* dalla posizione del codice

- **Indirizzamento pre/post incremento**: è simile all'indiretto a registro solo che il contenuto nel registro è automaticamente decrementato (incrementato) prima o dopo l'esecuzione dell'istruzione stessa. Utile per poter accedere facilmente ad insiemi di dati allocati in memoria sequenzialmente ed in particolare per le operazioni di estrazioni ed inserimenti della pila

MICROPROGRAMMAZIONE

La microprogrammazione fu proposta da Maurice Vincent Wilkes negli anni '50. Consiste nell'uso di una memoria di controllo **ROM** in cui sono archiviate le sequenze di comandi trasformati in segnali di controllo dall'unità di controllo **microprogrammata (UC-M)**. Questa tecnica è alternativa al transcodificatore combinatorio, che richiede molti componenti e interconnessioni, risultando costoso e poco flessibile.

Tra le componenti essenziali della micro-macchina ci sono:

- **Generatore di indirizzo**: cioè, il circuito predisposto al calcolo dell'indirizzo della successiva microistruzione
- **Registro RAR** (ROM address register): un registro al cui all'interno è presente l' indirizzo alla micro-istruzione successiva è una sorta di Program counter per la macchina microprogrammata

Ad ogni istruzione macchina corrisponde un **microprogramma** memorizzato in ROM, il cui contenuto è fisso e non modificabile. Questo permette di decodificare le istruzioni complesse in sequenze di microistruzioni eseguibili in hardware

MACCHINE CISC

Le **macchine CISC**, sviluppate negli anni '70, hanno un set di istruzioni numeroso e spesso a **lunghezza variabile**, con molti modi di indirizzamento. Queste caratteristiche permettono un codice compatto ma richiedono transcodificatori complessi

Alcune delle caratteristiche principali:

- Istruzioni di dimensione variabile.
- Decodifica complessa.
- Molti accessi alla memoria per istruzione.
- Pochi registri interni

MACCHINE RISC

Le **macchine RISC** furono sviluppate negli anni '80 per ottimizzare le prestazioni utilizzando un set di istruzioni ridotto. Queste CPU non avevano bisogno di essere compatibili con i prodotti esistenti, permettendo ai progettisti di scegliere istruzioni ottimizzate per la velocità e l'efficienza

Alcune delle caratteristiche principali:

- Istruzioni di dimensione fissa.

- Operazioni ALU solo tra registri.
- Molti registri interni.
- Modi di indirizzamento semplici.
- Durata fissa delle istruzioni

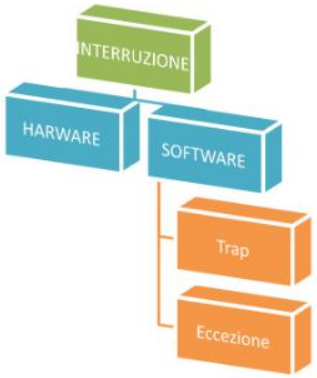
INTERRUZIONI

Una **interruzione** (interrupt) è un evento, in genere determinato da un dispositivo di input o di output, che cambia la normale sequenza di un programma in esecuzione, concepite per migliorare l'efficienza dell'elaborazione

Ci sono diverse possibili cause tra cui: *I/O, malfunzionamento hardware, programma errato, timer*.

Le interruzioni possono essere classificate come:

- **Interruzione hardware**: il segnale di interruzione è scaturito da un componente esterno alla CPU
- **Interruzione software**: il segnale di interruzione è scaturito da un programma
    - **Trap**: interruzione richiesta dal programmatore (*syscall*)
    - **Eccezioni**: interruzione determinata durante l'elaborazione di un programma



Oltre a questa classificazione ce ne è una seconda in relazione al *clock* della macchina:

- **Asincrone**: il loro accadimento non è noto, sono indipendenti dal clock
    - **Interruzioni esterne**: generate da un dispositivo I/O
    - **Interruzioni interne**: generate da un programma, note come eccezioni
        - Divisione per zero (*zero divide*)
        - Mancanza di tensione di alimentazione (*power failure*)
        - Indirizzo errato
- **Sincrone**: il loro accadimento è noto, perché richiesto dal programmatore
    - **Trap**

Un elaboratore dotato di un sistema di interruzione può avere un'architettura:

- **Mascherabile**: l'interruzione può essere interrotta a sua volta per svolgerne un'altra
- **Non mascherabile**: una volta richiesta l'interruzione questa deve essere espletata senza la possibilità di essere interrotta da altre

Un **interrupt** interrompe il programma in esecuzione e trasferisce il controllo a un **gestore di interruzione** deputato a svolgere le azioni appropriate. Al loro compimento, il gestore di interruzione restituisce il controllo al programma interrotto. È suo compito far riprendere il processo interrotto esattamente dallo stesso stato e posizione in cui si trovava al momento dell'interruzione.

Un **sistema di interruzione** deve:

- Garantire che non vi siano interferenze indesiderate sul processo interrotto (**commutazione del contesto**)
    - Il contesto di un programma è costituito dalle informazioni conservate nel Program Counter, Registro di Stato.
    - Vi è una "*fase protetta*" nella quale sono in corso le operazioni di commutazione del contesto, in questa fase è opportuno che non si verifichino altre interruzioni
    - Vi è, inoltre, un *campo di interruzione* che consente di definire se un programma può essere interrotto, tale campo è contenuto nel Registro di Stato

- Identificare il dispositivo che ha generato l'interruzione al fine di selezionare ed attivare la routine ad esso associata (**riconoscimento delle interruzioni**)
    - L'individuazione può avvenire tramite una scansione delle interruzioni (*Polling*)
    - Prevedendo che il dispositivo, oltre a generare la richiesta di interruzione invii una informazione più completa come, ad esempio, un codice di identificazione univocamente associato al dispositivo chiamante (*interruzione vettorizzata*)
- Risolvere le diverse interruzioni generate dai svariati dispositivi tramite la definizione di un ordine di gestione nel caso di un sistema a mascheramento (**gerarchia di priorità**)

POLLING

Tecnica in cui la CPU verifica periodicamente lo stato dei dispositivi per vedere se necessitano attenzione.

Procedura:

1. Alla fine di ogni istruzione, la CPU interroga i registri di stato dei dispositivi.
2. Se una periferica richiede un'interruzione, la CPU lo rileva.
3. La CPU può bloccarsi in un ciclo di attesa (*busy-waiting*) durante il polling.

POLLING E INTERRUZIONI

Per quanto riguarda le interruzioni vediamo come viene segnalata una **richiesta di interruzione**

- **Segnalazione di un dispositivo**:
    - Il modulo di interfaccia del dispositivo include un bit I in un registro di stato che segnala una richiesta di interruzione.
    - Questo bit è collegato al bit di interrompibilità della CPU (nel registro di stato) per abilitare o disabilitare le interruzioni.
    - Dopo ogni istruzione, la CPU verifica il bit I: se è 1, l'interruzione viene accettata.

- **Segnalazione di più dispositivi**:
    - Più dispositivi possono essere collegati tramite *un'architettura logica OR* (o wired OR) usando una singola linea per le richieste alla CPU

POLLING: COMMUTAZIONE DEL CONTESTO

Per quanto riguarda la **commutazione del contesto**, Il completamento di essa può avvenire in maniera più o meno radicale.

La commutazione può salvare tutte le informazioni dei registri e della ALU (codice semplice ma inefficiente) o solo quelle alterate (codice complesso ma efficiente).

Operazioni essenziali includono il salvataggio e il ripristino del PC (Program Counter) e del SR (Status Register).

Procedure di salvataggio	Procedure di ripristino
SR ← Push: salva lo SR nello stack	PC ← Pop: ripristina il valore del PC
PC ← Push: salva il PC nello stack	SR ← Pop: ripristina il valore del SR

Set del flag I a 0 per evitare altre interruzioni non mascherabili.

Inserimento dell'indirizzo della prima istruzione della routine di servizio nel PC.

POLLING: IDENTIFICAZIONE DELL'INTERRUZIONE

La CPU interroga i dispositivi uno per uno fino a trovare quello che ha richiesto il servizio.

Sequenza di polling:

- JR DISP1, SERV1: salta alla routine di servizio del dispositivo 1.
- JR DISP2, SERV2: salta alla routine di servizio del dispositivo 2.
- Continua per ogni dispositivo.

POLLING: GERARCHIA DI PRIORITÀ

L'ordine di analisi dei dispositivi nella sequenza di polling determina la gerarchia di priorità.

Se si devono gestire interruzioni multiple, un'istruzione SETI può riabilitare le interruzioni durante l'esecuzione di una routine di servizio, rendendo la CPU nuovamente interrompibile.

Per migliorare *l'efficienza* di un sistema di interruzione ci sono due strategie principali:

Ottimizzazione del Tempo di Risposta:

- Tempo di risposta: intervallo tra la richiesta di intervento da parte di un dispositivo e l'inizio dell'esecuzione della prima istruzione utile da parte del processore.
- Obiettivo: ridurre questo tempo, che può variare da 1-2 microsecondi a 20-30 microsecondi in sistemi meno sofisticati

Commutazione del Contesto:

- Salvataggio del contesto in registri specifici per ogni livello di priorità.
- Vantaggio: aumento della velocità poiché non c'è bisogno di salvare il contesto nello stack.
- Svantaggio: aumento dei costi dovuto alla necessità di avere più registri.

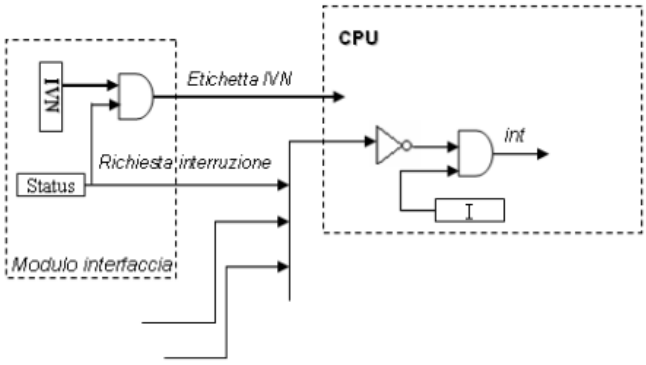
INTERRUZIONE VETTORIZZATA

Sistema in cui ogni dispositivo ha un registro con un codice di identificazione o **numero vettorizzato di interruzione** (IVN). Il codice viene inviato al processore quando si verifica un'interruzione.

Funzionamento:

- **Identificazione Diretta**: Il codice IVN può puntare direttamente alla routine di servizio del dispositivo.
- **Tabella delle Interruzioni**:
  - Memoria riservata per memorizzare gli indirizzi delle routine di servizio.
  - Solitamente posizionata nella parte alta della memoria per l'uso di codici con pochi bit.

- *Indirizzamento indiretto*: il processore utilizza il codice IVN per trovare l'indirizzo della routine di servizio.



Salvataggio e Ripristino del Contesto:

Salvataggio:

- SR ← Push: salva lo SR nello stack.
- PC ← Push: salva il PC nello stack.
- 0 ← I: imposta il flag I a 0 per evitare altre interruzioni.
- PC ← (IVN): imposta il PC all'indirizzo del vettore di interruzione puntato dall'IVN.

Ripristino:

- PC ← Pop: ripristina il PC.
- SR ← Pop: ripristina lo SR.

Vantaggi dell'Interruzione Vettorizzata:

Efficienza: Il processore non solo riceve la richiesta di interruzione, ma anche l'indirizzo indiretto della routine di servizio tramite l'IVN.

Ottimizzazione della Memoria: Le routine di servizio possono essere allocate in punti diversi della memoria, ottimizzando l'uso della memoria centrale.

Gerarchia di Priorità: Determinata dal valore del IVN

La presenza di interruzioni è spesso legata alla **gestione delle funzionalità di basso livello** dell'elaboratore come il disco, le periferiche e i timer. I programmi per queste funzionalità devono avere accesso a tutte le caratteristiche dell'elaboratore.

Modalità di accesso:

- **Supervisore**: Accesso pieno alle funzionalità del sistema, usato per la gestione delle interruzioni e altre operazioni critiche.
- **Utente**: Accesso limitato alle istruzioni standard, usato per i programmi utente.

Quando viene accettata un'interruzione, la CPU cambia dalla modalità Utente alla modalità Supervisore.

DRIVER

La modalità Supervisore è destinata all'esecuzione di routine gestite dal Sistema Operativo. Le routine che gestiscono l'interazione con una periferica specifica sono chiamate **driver**.

Installazione dei Driver:

- Ogni periferica ha il proprio set di driver necessari per il suo funzionamento.

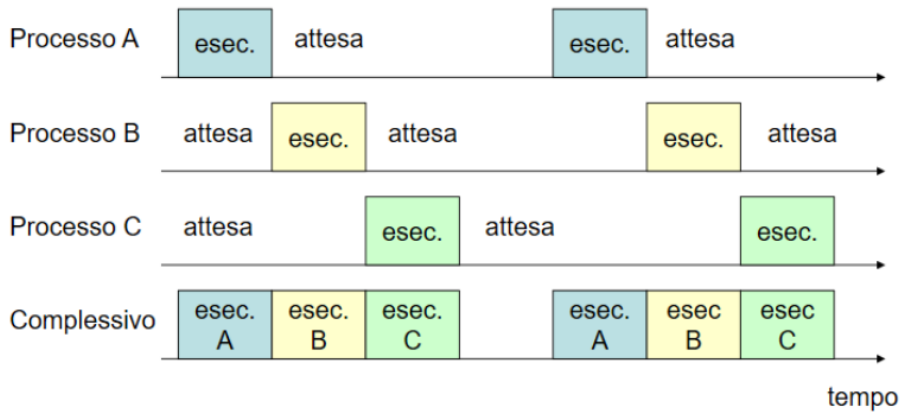
- L'installazione di una nuova periferica richiede l'installazione dei relativi driver per permettere al Sistema Operativo di interagire correttamente con essa.

Nel caso in cui subentra una richiesta di interruzione a più alta priorità, ci sono alcune fasi del servizio di un'interruzione che non possono essere interrotte (**fase protetta**)

1. Sospensione esecuzione del programma corrente
2. Salvataggio del contesto
3. Inizializzazione del PC all'indirizzo di ingresso della routine per la gestione dell'interrupt
4. Esecuzione della routine di interruzione ISR
5. Ripristino del contesto
6. Continuazione esecuzione del programma corrente

In questo caso, solo la procedura numero "4" non rientra nella fase protetta

Grazie alle interruzioni generate nel Sistema Operativo a tempi prestabiliti è possibile la **multiprogrammazione**: cioè, più programmi eseguiti in memoria ad intervalli regolari e separati che danno un effetto di pseudo parallelismo.



INTERRUZIONI NEL MIPS

Prima di saltare all'handler, la CPU del MIPS salva un identificatore numerico del tipo di eccezione/interruzione in un registro interno.
L'handler accede a questo registro per determinare la causa dell'eccezione/interruzione.

Registro Cause: Utilizzato per memorizzare il motivo dell'eccezione/interruzione.

Registro EPC (Exception Program Counter): Salva il valore del Program Counter corrente quando si verifica l'interruzione.

Tutto il resto dello stato del programma deve essere salvato dall'handler.

Progettazione del Sistema:

- Individuare l'Evento: Il sistema deve essere progettato per rilevare l'evento inatteso.
- Interrompere l'Istruzione Corrente: Fermare l'istruzione in esecuzione.
- Salvare il PC Corrente: Salvare il Program Counter nel registro *EPC*.
- Salvare la Causa dell'Interruzione: Salvare la causa dell'interruzione nel registro *Cause*.
- Saltare alla Routine del SO: Eseguire la routine dell'handler all'indirizzo fisso 0xC0000000.

Il MIPS non salva alcun altro registro oltre il valore del Program Counter, è compito dell'handler salvare altre porzioni dello stato corrente del programma, se necessario → **Approccio RISC** (MIPS): semplice e minimale

Alcune CPU salvano automaticamente un'ampia porzione dello stato del programma (inclusi tutti i registri generali) prima di saltare all'interrupt handler. Questo salvataggio esteso è garantito dal microcodice, rendendo il processo più complesso → **Approccio CISC**

CANALIZZAZIONE (PIPELINE)

La **pipeline**, o **canalizzazione**, è una tecnica che consiste nella scomporre una rete logica (combinatoria) in una serie di reti logiche più semplici mediante l'inserimento di opportuni **registri di disaccoppiamento** (interlock). Questo accorgimento permette di aumentare la frequenza di clock e svolgere *più fasi in parallelo.*

L'esecuzione di una istruzione, in una **macchina RISC** (MIPS), è di solito suddivisa in cinque fasi (o sezioni o stage):

- **F – Fetch**: prelievo istruzione (con incremento del Program Counter)
- **D – Decode**: decodifica/riconoscimento istruzione
- **E – Execution**: esecuzione istruzione
- **M – Memory**: accesso in memoria per scrittura o lettura (load o store)
- **WB – Write Back**: quando il risultato dell'ALU (o durante una operazione di load) viene messo nel registro destinazione

Senza la pipeline, solo un'unità funzionale è attiva, mentre le altre non sono impegnate.

CANALIZZAZIONE MACCHINE RISC

In una macchina canalizzata ogni unità funzionale elabora la fase che gli corrisponde e passa all'istruzione successiva

Esecuzione sequenziale																				
Δt	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Istruzione 1	F	D	E	M	WB																
Istruzione 2		F	D	E	M	WB															
Istruzione 3			F	D	E	M	WB														
Istruzione 4				F	D	E	M	WB													
Istruzione 5					F	D	E	M	WB												

Ogni fase della pipeline deve poter essere svolta **contemporaneamente** alle altre, imponendo una **durata uniforme**, stabilita sulla base della fase più lenta. Questo impone un clock più lungo per tutte le classi di istruzione, ma migliora il tempo complessivo di esecuzione del programma una volta che la pipeline è a regime.

Il **guadagno** offerto da una pipeline a regime è dato dal numero di fasi elementari della pipeline (in una pipeline con 5 fasi, il guadagno è di 5 volte)

In una macchina con pipeline, il **compilatore** assume un ruolo fondamentale, deve infatti presiedere alla suddivisione delle istruzioni complesse in gruppi di istruzioni semplici ed ottimizzare la sequenza di esecuzione, sfruttando appieno i vantaggi della pipeline.

La canalizzazione è un accorgimento **trasparente** al programmatore.

CANALIZZAZIONE NELLE MACCHINE CISC

L'implementazione di pipeline su **elaboratori CISC** risulta difficoltosa a causa della intrinseca complessità delle istruzioni

L'esecuzione di una istruzione, nelle macchine CISC, è di solito suddivisa in sei fasi la cui durata è molto variabile:

- **F – Fetch**: prelievo istruzione (con incremento del Program Counter)
- **D – Decode**: decodifica/riconoscimento istruzione
- **L – Load**: prelievo operandi dalla memoria (tipico delle istruzioni con modi di indirizzamento complesso)
- **E – Execution**: esecuzione istruzione
- **M – Memory**: accesso in memoria per scrittura o lettura (load o store)
- **WB – Write Back**: quando il risultato dell'ALU (o durante una operazione di load) viene messo nel registro destinazione

La fase Fetch può sovrapporsi alla fase Decode nelle macchine CISC solamente se l'istruzione ha **dimensione fissa** e quindi è possibile calcolare il prossimo valore del Program Counter senza sapere la classe di istruzione

Altrimenti nelle architetture CISC con **istruzioni a dimensione variabile**, oltre alla presenza di una eventuale fase di Load, la fase di **decodifica** è variabile così come quelle di esecuzione e di scrittura dei dati.

L'alternativa è commisurare il tempo **più lungo** di ciascuna fase delle diverse classi di istruzioni ed uniformare ogni fase di tutte le classi di istruzioni a quelle più lunghe (esattamente come avviene per le macchine RISC); con un dispendio temporale più lungo per le istruzioni "meno complesse".

Esecuzione RISC

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	D	E	M	WB																
Istruzione 2		F	D	E	M	WB															
Istruzione 3			F	D	E	M	WB														
Istruzione 4				F	D	E	M	WB													
Istruzione 5					F	D	E	M	WB												

Esecuzione CISC

Δt	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Istruzione 1	F	D	D	L	L	L	E	E	M	WB											
Istruzione 2				F	D	D	L	L	L	E	M	WB									
Istruzione 3						F	D	D	D	D	L	L	E	M	M	WB					
Istruzione 4										F	F	D	L	L	L	E	M	WB			
Istruzione 5												F	F	D	D	L	L	L	E	M	WB

Seppur fattibile, la canalizzazione nelle Macchine CISC è molto più complicata da realizzare

Allo stesso tempo una istruzione CISC compie funzioni più "articolate" che in una macchina RISC richiederebbero più istruzioni

Malgrado questi due punti, uno sfavorevole ma fattibile e l'altro estremamente vantaggioso, si predilige l'uso di Macchine RISC perché, da analisi statistiche, il 90% dei programmi utilizza istruzioni semplici mentre l'impiego di istruzioni complesse, è riservato a pochi applicativi specifici.

Come in ogni aspetto, ci sono anche delle **condizioni svantaggiose** per la pipeline, **hazard**.

Le istruzioni possono essere eseguite in modo ottimale nella pipeline fino a quando non si verifica una criticità (hazard). Ad esempio, un salto (branch) richiede di svuotare la pipeline e caricare le nuove istruzioni a partire dal punto di arrivo del salto.

Ci sono diversi tipi di hazard:

- **Control Hazard**: Cambiamenti nel flusso di esecuzione delle istruzioni, come salti o interruzioni.
- **Data Hazard**: Un dato necessario non è ancora disponibile perché in corso di calcolo in un'istruzione precedente.
- **Hazard Strutturali**: Le risorse hardware necessarie non sono disponibili (es. ALU occupata)

Per risolvere il degrado delle prestazioni, ci sono diverse soluzioni:

- **Propagazione (Forwarding)**:
    - Se il dato necessario è già disponibile nella pipeline, si creano scorciatoie nel datapath per consegnare il dato all'unità funzionale che ne ha bisogno senza aspettare il WB.
    - Evita i problemi di pipeline interlocked block.
- **Stallo**: quando il forwarding non è possibile, è necessario inserire un tempo di attesa.
- **Eliminazione degli stalli** (soluzione ai dati hazard): si ottiene riordinando le istruzioni, con la condizione da rispettare di mantenere la stessa semantica.

Delle strategie per mitigare i **control hazard** possono essere:

- **Anticipare la Decisione**:
    - Descrizione: Calcolare prima possibile se un salto condizionato o incondizionato deve essere eseguito.
        - Salto Condizionato: Se il salto è calcolato dall'ALU nella fase E (Execution), ci saranno due istruzioni da scartare perché si deve aspettare la fine della terza fase.
        - Salto Incondizionato: Se riconosciuto nella fase D (Decode), consente di scartare solo una istruzione.
    - Vantaggio: Riduce il numero di istruzioni da scartare, minimizzando i cicli di clock persi.

- **Branch Prediction** (Previsione del Salto)
    - Descrizione: La CPU osserva i salti eseguiti in precedenza e tenta di precaricare l'istruzione corretta (seguente o di destinazione) che deve essere eseguita.
        - *Static Branch Prediction*: Previsione basata su una regola fissa, come sempre predire il salto.
        - *Dynamic Branch Prediction*: Previsione basata su un algoritmo che tiene conto della storia dei salti recenti.
    - Vantaggio: Migliora l'efficienza della pipeline riducendo i cicli di clock sprecati per i salti non previsti correttamente.

- **Delayed Branch** (Ritardare il Salto):

- Descrizione: Se possibile, eseguire l'istruzione che segue il salto condizionato prima di eseguire effettivamente il salto.
- Condizione: L'istruzione che segue il salto deve poter essere eseguita senza influenzare la logica e il significato del programma.
- Vantaggio: Elimina la necessità di uno stallo, mantenendo la pipeline in esecuzione continua.

CPU MIPS CON PIPELINE

La canalizzazione, per funzionare correttamente, richiede che le varie unità siano sincronizzate e che i dati delle varie istruzioni non si sovrappongano; quindi, all'interno delle pipeline sono posti dei blocchi (**interlock**) che sorvegliano il completamento delle varie istruzioni e fanno procedere la pipeline solamente quando tutti gli stadi sono pronti. Tuttavia, introduce stalli che possono deprimere le prestazioni.

La **caratteristica distintiva** del progetto MIPS è che tutte le istruzioni sono completate dagli stadi della pipeline in un solo ciclo di clock in modo da non introdurre ritardi e stalli nella pipeline

In una CPU MIPS con pipeline (senza gestione hazard) avremo:

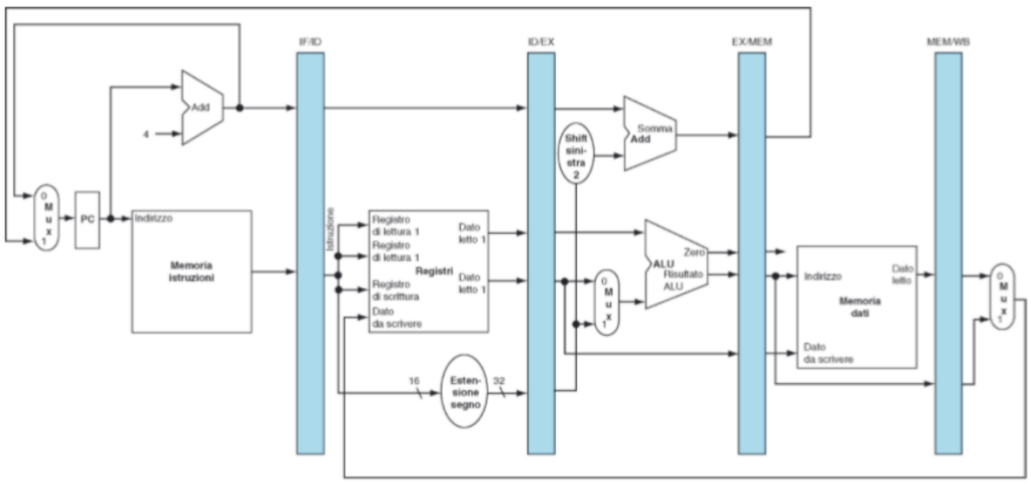
Obiettivo:

- **Fasi Veloci**: Il periodo di clock è determinato dalla durata della fase più lenta.
- **Compiti Specifici per Fase**: Ciascuna fase esegue un solo compito (o più compiti in parallelo).
- **Passaggio di Informazioni e Segnali di Controllo**: Ogni fase riceve informazioni e segnali di controllo e li passa alla fase successiva.
- **Stabilità dei Segnali**: I segnali devono rimanere stabili durante tutta la fase. Ciò viene ottenuto utilizzando latch o registri che cambiano solo alla transizione del clock.

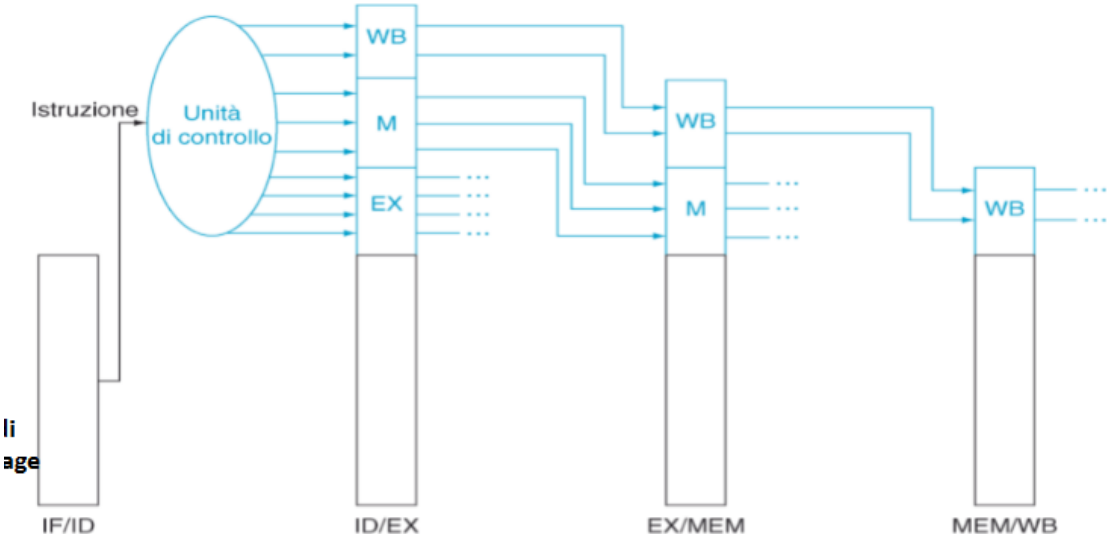
Soluzione:

- **Separazione delle Fasi**: Ogni fase è separata dalla successiva tramite un registro (interblock) che riceve informazioni e segnali di controllo dalla fase precedente e li mette a disposizione per la fase successiva.

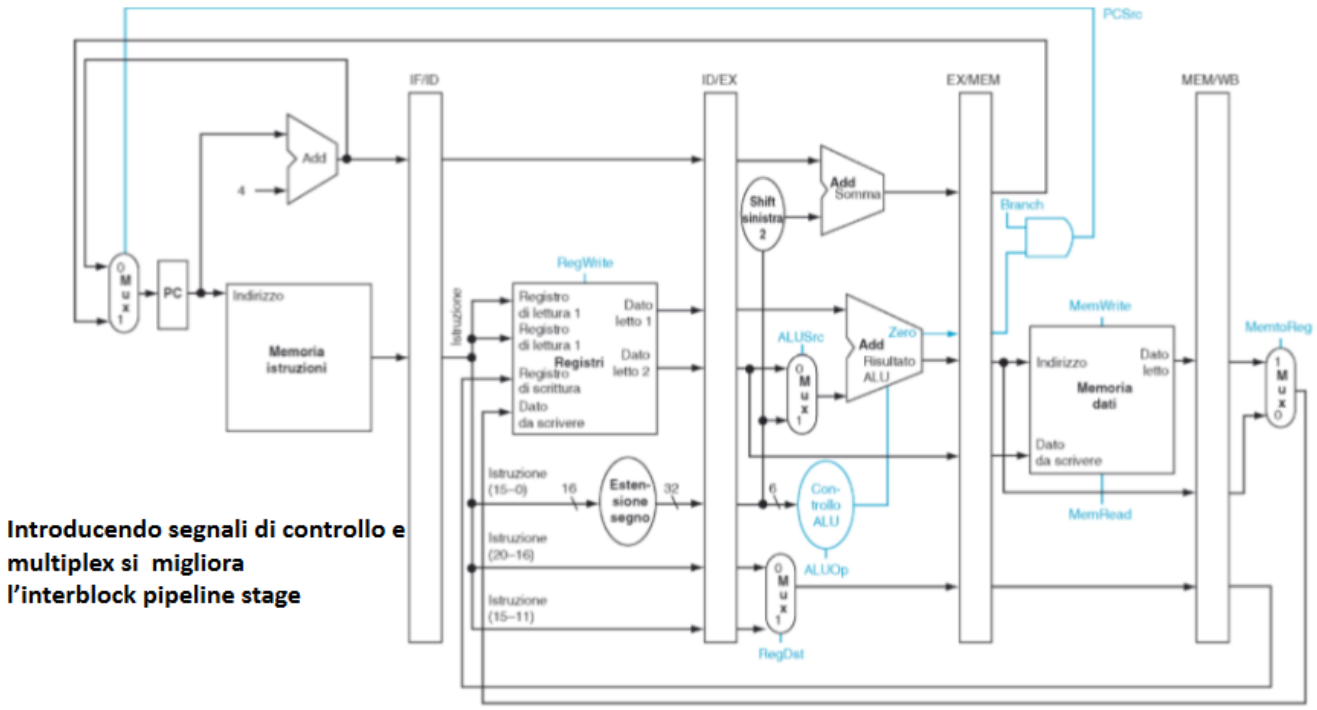
MIPS microprocessor without interlocked pipeline stages (microprocessore senza fasi bloccanti nella pipeline). Questo design permette di mantenere la pipeline sempre attiva, senza necessità di interblocchi complessi che potrebbero produrre stalli (molto efficiente).



Ad ogni fase nei **registri** della pipeline sono presenti i dati (letti dai registri o dalla parte immediata) e i comandi (generati dalla CU).

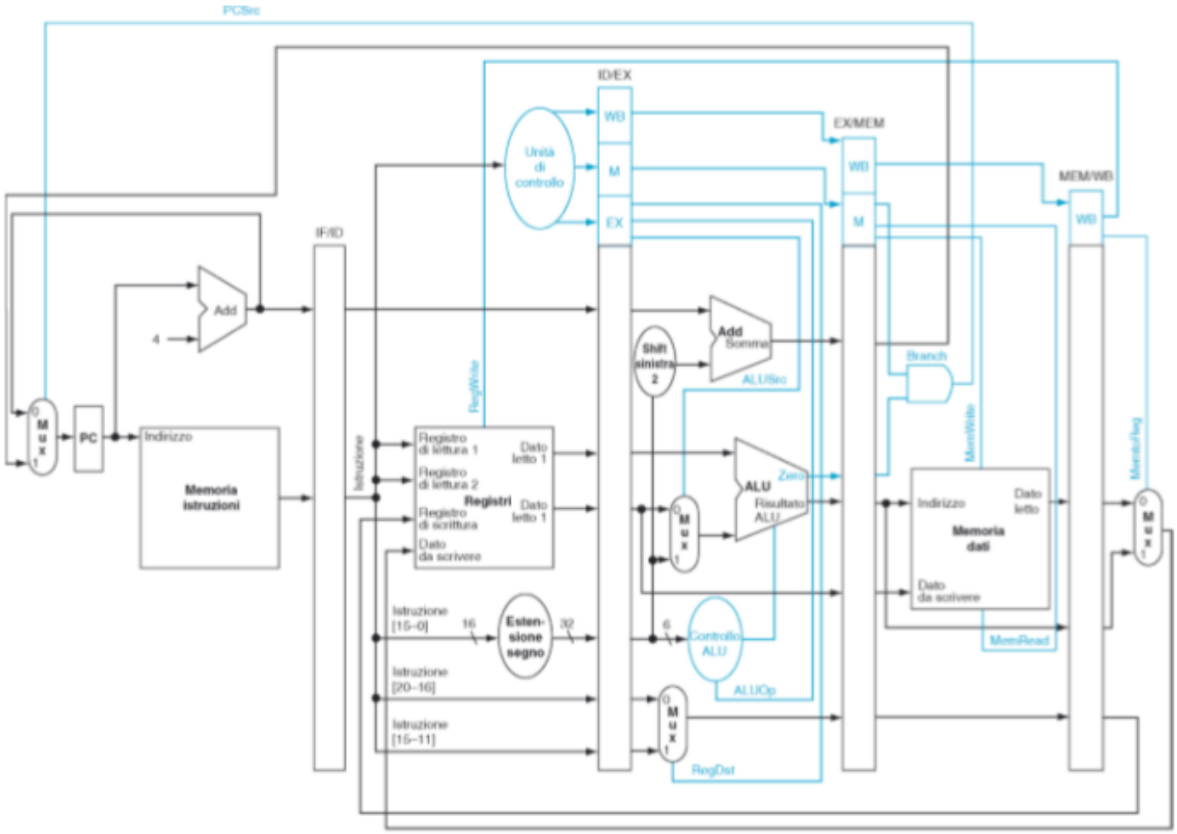


Pipeline con WB corretto e Salti:



Introducendo segnali di controllo e multiplex si migliora l'interblock pipeline stage

CPU MIPS:



CONSIDERAZIONI SUGLI HAZARD

DATA HAZARD

Un **data hazard** si verifica quando un'istruzione dipende dal risultato di una precedente che non è ancora stata scritta nel registro (Write-back). Specificamente, nel MIPS, un data hazard può sorgere se:

- Il registro destinazione di un'istruzione è uguale a uno dei registri sorgenti di una delle due istruzioni successive.
- L'istruzione precedente ha il segnale di controllo "**RegWrite**" attivo.

Problema: Stallo

Quando si verifica un data hazard, può essere necessario inserire uno stallo nella pipeline per attendere che l'istruzione precedente completi il write-back del valore richiesto.

Soluzione: Forwarding o Eliminazione dello Stallo

Il MIPS adotta una tecnica chiamata **forwarding** per risolvere i data hazard senza introdurre stalli. Quando il forwarding non è possibile, si inserisce uno stallo.

Nel **MIPS**, per fermare l'istruzione con uno stallo, si deve (nella fase di decodifica):

- *Annullare l'istruzione* che deve attendere (una bolla che continuerà senza fare nulla) ovvero azzerare i segnali di controllo MemWrite e RegWrite e bloccare il **registro F/D** (stadio intermedio nella pipeline tra la fase di Fetch e Decode).
- *Ripetere l'istruzione* che è stata letta e deve entrare nella fase Decodifica ovvero impedire l'aggiornamento del registro F/D della pipeline
- *Rileggere la stessa istruzione* di nuovo perché possa essere rieseguita un clock dopo, ovvero impedire che il PC si aggiorni

CONTROL HAZARD

Un **control hazard** si verifica quando il flusso sequenziale di esecuzione delle istruzioni viene interrotto da istruzioni che alterano il Program Counter, come i salti condizionati, i salti incondizionati e le chiamate a subroutine. Questo accade perché, nel tempo necessario a determinare se il salto deve essere eseguito o meno, la **pipeline** potrebbe aver già fetchato, decodificato o addirittura iniziato l'esecuzione delle istruzioni che non dovrebbero essere eseguite in caso di salto.

Salto Incondizionato:

- **Circuiteria di Anticipo**: Per gestire i salti incondizionati, viene utilizzata una circuiteria che analizza l'opcode dell'istruzione. Se l'istruzione è un salto incondizionato, il Program Counter (PC) viene aggiornato immediatamente dopo la fase di fetch, senza necessità di decodificare l'istruzione.
- **Beneficio**: Questa tecnica consente di evitare di scartare istruzioni, poiché la successiva istruzione fetchata sarà quella corretta, migliorando così l'efficienza della pipeline

Salto condizionato:

- **Predizione del salto**:
  - **Bit di predizione**: A ogni istruzione di salto possono essere associati dei bit di predizione che, basandosi sulla storia dei salti precedenti, indicano se è più probabile che il salto venga eseguito o meno
  - **Predizione con un bit**: Un singolo bit può indicare se l'ultimo salto è stato eseguito. Tuttavia, questa soluzione può essere inefficace durante i cicli, poiché sbaglierà la predizione sia entrando che uscendo dal ciclo.
  - **Predizione con due bit**: Utilizzando due bit, si può implementare una macchina a stati finiti che richiede due errori consecutivi per cambiare la predizione, riducendo così il numero di predizioni errate, soprattutto in cicli con una forte prevalenza di un tipo di scelta.

- **Ritardo del salto** (Delay slot): Per recuperare il tempo perso a causa degli stalli dovuti ai salti condizionati, si può ritardare il salto eseguendo in ogni caso l'istruzione successiva al salto.
  - **Istruzione precedente** (Caso 1): Se possibile, una istruzione precedente che non abbia dipendenze con il salto condizionato viene copiata nel *delay slot*.
  - **Istruzione di Destinazione del Salto** (Caso2): Se non esistono istruzioni precedenti senza dipendenze, un'istruzione dalla destinazione del salto può essere copiata nel *delay slot*.

Queste istruzioni verranno sempre eseguite e non devono creare problemi se il salto non viene effettuato.

La corretta gestione dei delay slot è **cruciale** per mantenere l'integrità del flusso di controllo e per evitare errori nell'esecuzione del programma.

HAZARD STRUTTURALE

Un **hazard strutturale** MIPS si può verificare quando l'architettura del processore non è in grado di gestire correttamente le richieste simultanee di risorse hardware condivise da più istruzioni nella pipeline.

Alcuni esempi di hazard strutturali possono essere:

- **Accesso Simultaneo all'ALU**: Se due istruzioni richiedono l'uso dell'ALU nello stesso ciclo di clock, ma il processore ha solo una ALU, si verifica un hazard strutturale.
  Se il processore ha solo una ALU, queste due istruzioni non possono essere eseguite contemporaneamente.
  Esempio pratico: add $t0, $s0, $zero # Istruzione 1: Usa l'ALU
  add $t1, $s1, $zero # Istruzione 2: Usa l'ALU
- **Accesso simultaneo alla memoria**: Se due istruzioni richiedono l'accesso alla memoria nello stesso ciclo di clock, ma il processore ha solo un modulo di memoria, si verifica un hazard strutturale.

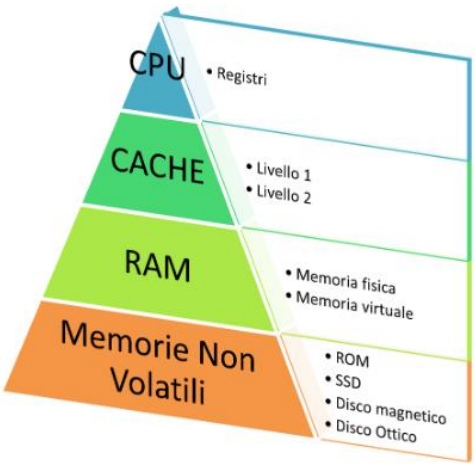
Per **mitigare gli hazard strutturali MIPS**, sono state sviluppate varie tecniche di progettazione dei processori, come l'implementazione di *buffer* e *pipeline più complesse*, l'uso di tecniche di predizione delle dipendenze, *l'allocazione dei registri* e *l'implementazione di tecniche di scheduling* delle istruzioni.

Un'altra strategia è quella di aggiungere dei cicli di attesa, la ristrutturazione del codice in modo che gli accessi alla ALU non siano dipendenti oppure si deve ricorrere al forwarding.

MEMORIA CACHE

A causa delle differenze di costo e della velocità, si definisce una **gerarchia di livelli di memoria**, con una memoria più veloce (e quindi più costosa) posta vicino al processore e una più lenta (e meno costosa) sita a maggiore distanza.

Nella gerarchia di solito si comprende anche le **memorie non volatili**, cioè quei dispositivi che consento l'archiviazione permanente dei dati (es.: disco e nastro magnetico, supporti ottici, MSS) che sono memorie estremamente lente (a causa dei componenti elettromeccanici utilizzati per archiviare e reperire i dati).



L'introduzione della memoria cache si basa sul **principio di località**:

- **Località temporale**: quando si fa riferimento a un elemento di memoria, c'è la tendenza a far riferimento allo stesso elemento entro breve.
- **Località spaziale**: quando si accede a un elemento di memoria, c'è la tendenza a far riferimento entro breve tempo ad altri elementi che hanno indirizzo vicino (per esempio quando si accede a dati organizzati in vettori o matrici).

L'obiettivo della gerarchia di memoria è quello di dare al programmatore *l'illusione* di poter usufruire di una memoria al tempo stesso veloce (idealmente, quanto la memoria al livello più basso) e grande (quanto quella al livello più alto).

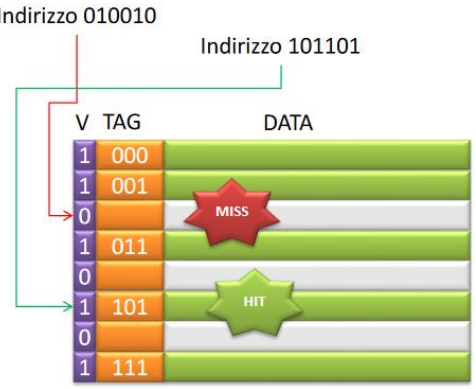
A causa della differente dimensione, diventa necessario, durante l'esecuzione dei programmi, **trasferire informazione** fra memorie di livelli diversi (solo tra livelli adiacenti).

La memoria **Cache** è organizzata fisicamente in linee composte da:

- 1 bit di **VALIDITÁ** che indica se la linea contiene dei dati
- Il campo **TAG** (o Etichetta) che distingue quale parola (o blocco) della Memoria Centrale è nella linea
- Il campo **DATA** (parola o blocco), cioè la parola (o il blocco) memorizzato nella linea
- **Altri bit utili**: *Dirty, Used, Writable*

Quando la CPU *richiede un indirizzo*, se il dato richiesto fa parte di uno dei blocchi presenti nella cache di livello inferiore, si dice che la richiesta ha avuto successo (**HIT**).

Se invece il dato non si trova nel livello inferiore, si dice che la richiesta fallisce (**MISS**). In questo caso bisogna accedere al livello superiore della gerarchia e trasferire il blocco corrispondente al livello inferiore.



PROGETTAZIONE DI UNA CACHE

Il **progetto** di una gerarchia di memoria richiede la risoluzione di quattro problemi:

1. **Problema del piazzamento di un blocco/parola**: dove mettere una parola (o un blocco) che è portato dal livello superiore al livello inferiore
2. **Problema della ricerca di un blocco/parola**: dove si trova la parola (o il blocco) che contiene il dato richiesto
3. **Problema della sostituzione di un blocco/parola**: quale parola (o blocco) presente al livello inferiore deve essere sostituito da uno del livello superiore nel caso di fallimento
4. **Problema della strategia di scrittura**: cosa succede nel caso di scrittura sulla parola (o sul blocco) presente nella cache

PIAZZAMENTO DI UN BLOCCO/PAROLA

Quando la CPU tenta di accedere a una parola in memoria, è importante determinare come e dove questa parola sarà posizionata nella cache per ottimizzare le prestazioni del sistema. Ci sono tre principali schemi di piazzamento della parola nella cache:

- **Cache a Indirizzamento Diretto** (Direct Mapped Cache)
  - Ogni parola o blocco di memoria ha una corrispondenza esatta con una specifica linea della cache.
  - La posizione nella cache è determinata dall'indirizzo della parola modulo il numero di linee nella cache.
  - *Pro*: hardware semplice ed economico, facile determinare la linea del dato cercato
  - *Contro*: possibilità di molte miss se diversi blocchi mappano sulla stessa linea (*trashing*).

- **Cache Set-Associativa a n Vie** (n-Way Set Associative Cache):
    - La cache è suddivisa in insiemi (**set**), e ogni insieme contiene un numero fisso di vie (**linee**).
    - Ogni parola può essere posizionata in una qualsiasi delle linee del set determinato dall'indirizzo della parola modulo il numero di insiemi nella cache.
    - *Pro*: combina la flessibilità della cache completamente associativa con l'efficienza dell'indirizzamento diretto.
    - *Contro*: hardware più complesso rispetto alla cache a indirizzamento diretto
- **Cache Completamente Associativa** (Fully Associative Cache)
    - Ogni blocco di memoria può essere posizionato in una qualsiasi linea della cache.
    - Durante la ricerca, tutti i blocchi della cache devono essere esaminati.
    - *Pro*: massima flessibilità nel piazzamento dei blocchi.
    - *Contro*: hardware molto complesso e costoso a causa della necessità di esaminare tutte le linee della cache.

RICERCA DI UNA PAROLA

All'inizio una cache è **vuota** e i bit di validità sono negativi (N). A seguito del riempimento di una linea di cache, si aggiorna il relativo bit di validità.

Consideriamo di andare a ricercare una parola in una **cache direct mapped** vuota:

Indirizzo di memoria: 10110; *Numero di linea*: 110; *Stato iniziale*: bit di validità N e tag senza significato

Quando si ricerca l'indirizzo 10110 avremo:

Linea di cache: 110; *Bit di validità*: N (miss); *Azione*: Trasferimento del dato dalla RAM alla cache

Dopo un **miss**, la cache si aggiorna:

Linea di cache: 110; *Bit di validità*: S (positivo); *Tag*: 10; *Dato*: parola corrispondente a 10110

SOSTITUZIONE DI UN BLOCCO/PAROLA

Per **ottimizzare** le prestazioni di accesso alla memoria, la cache è organizzata in modo da consentire il prelievo di blocchi di parole adiacenti. La **struttura** per **prelevare** più parole adiacenti (un blocco) è costituita da:

- **Etichetta** (Tag): utilizzata per verificare se il blocco richiesto è presente nella cache
    - *Cache a indirizzamento diretto*: Confrontata con il tag del blocco selezionato dall'indice.
    - *Cache set-associativa*: Confrontata con i tag di tutti i blocchi nell'insieme selezionato dall'indice.
    - *Cache completamente associativa*: Confrontata con i tag di tutti i blocchi nella cache.
- **Indice**: serve a identificare l'insieme o il blocco specifico nella cache
    - *Cache a indirizzamento diretto*: Identifica direttamente il blocco nella cache.
    - *Cache set-associativa*: Identifica l'insieme nella cache.
    - *Cache completamente associativa*: Non utilizzato, poiché c'è un solo insieme.
- **Spiazzamento** (Offset): indica l'indirizzo specifico della parola all'interno del blocco. Utilizzato per accedere alla parola esatta all'interno del blocco di dati.

Per quanto riguarda il **rimpiazzo** della parola/blocco, ci sono diverse politiche.

Quando si verifica un *fallimento* nell'accesso alla cache (**cache miss**), il blocco da sostituire deve essere deciso in base alla politica di rimpiazzo adottata. Le politiche di rimpiazzo sono *fondamentali* per mantenere l'efficienza della cache, soprattutto nelle cache set-associative e completamente associative, dove ci sono più blocchi candidati alla sostituzione.

Cache a indirizzamento diretto: vi è una sostituzione immediata, poiché ogni blocco di memoria ha una sola posizione possibile nella cache, non c'è bisogno di decidere quale blocco sostituire. La sostituzione è automatica e semplice.

Cache set-associative e completamente associative: in queste tipologie di cache, occorre stabilire quale blocco all'interno dell'insieme (nel caso di set-associativa) o della cache (nel caso di completamente associativa) deve essere sostituito.

Le principali strategie di sostituzione sono:

- **Least Recently Used** (LRU): Sostituisce il blocco che è stato utilizzato meno di recente.
    - *Implementazione*: Ogni blocco ha un contatore che viene aggiornato ad ogni accesso. Il blocco con il contatore più basso (non aggiornato da più tempo) viene sostituito.
    - *Pro*: Buona approssimazione della località temporale.
    - *Contro*: Può essere complesso e costoso da implementare per insiemi di grandi dimensioni
- **Least Frequently Used** (LFU): Sostituisce il blocco meno frequentemente utilizzato.
    - *Implementazione*: Ogni blocco ha un contatore che viene incrementato ad ogni accesso. Il blocco con il contatore più basso viene sostituito.
    - *Pro*: Favorisce i blocchi che vengono acceduti spesso.
    - *Contro*: Non considera la località temporale, un blocco che è stato molto utilizzato in passato potrebbe non essere più rilevante.
- **Sostituzione Casuale** (Random): Sostituisce un blocco scelto casualmente tra quelli disponibili.
    - *Implementazione*: La scelta del blocco da sostituire è fatta in modo casuale, eventualmente usando componenti hardware.
    - *Pro*: Semplice da implementare e non richiede contatori.
    - *Contro*: Meno efficiente rispetto ad altre politiche in termini di riduzione dei cache miss.

STRATEGIA DI SCRITTURA E AGGIORNAMENTO DELLA MEMORIA

Quando un dato è modificato nella cache, è **necessario aggiornare la memoria di livello inferiore**.

Le principali politiche di aggiornamento della memoria sono:

- **Write Through**: Ogni volta che un dato viene scritto nella cache, lo stesso dato viene immediatamente scritto anche nella memoria principale.
    - *Pro*: Mantiene la coerenza dei dati tra cache e memoria principale, particolarmente utile in sistemi multiprocessore.
    - *Contro*: Può rallentare le prestazioni a causa delle frequenti scritture nella memoria principale. L'uso di un buffer di scrittura può mitigare questo problema.
- **Write Back**: Il dato viene scritto nella memoria principale solo quando il blocco viene sostituito nella cache.
    - *Pro*: Riduce il numero di scritture nella memoria principale, migliorando le prestazioni della cache.

- *Contro*: La cache può contenere dati non aggiornati rispetto alla memoria principale, complicando la gestione della coerenza in sistemi multiprocessore. L'uso di un bit "Dirty" può aiutare a identificare i blocchi modificati che necessitano di essere scritti in memoria.

Il **miglioramento della cache è essenziale** per ottimizzare le prestazioni dei sistemi di calcolo moderni. La cache è una memoria veloce e di piccole dimensioni che immagazzina temporaneamente i dati più frequentemente utilizzati dal processore, riducendo i tempi di accesso rispetto alla memoria principale (RAM).

Miglioramenti nella gestione della cache:

- **Cache Multilivello**:
  - *Cache di 1° livello* (L1): integrata nel chip del processore, ad accesso rapidissimo
  - *Cache di 2° livello* (L2): Spesso esterna al chip del processore, con accesso rapido.
  - *Cache di 3° livello* (L3): Talvolta presente, con accesso meno rapido ma ancora più veloce della RAM.
  - *Scopo*: Minimizzare i tempi di accesso ai dati e ridurre l'impatto dei cache miss.
- **Aumento delle dimensioni del blocco di Cache**:
  - *Pro*: Riduce il numero di cache miss fino a un certo punto, sfruttando la località spaziale.
  - *Contro*: Oltre una certa dimensione, le prestazioni possono peggiorare a causa delle istruzioni di salto che riducono l'efficienza della località spaziale.
- **Cache separate per istruzioni e dati** (Split Cache):
  - *Descrizione*: Cache separate per istruzioni e dati, raddoppiando di fatto la larghezza di banda della memoria.
  - *Pro*: Operazioni di lettura e scrittura possono essere eseguite indipendentemente in ognuna delle due cache, migliorando l'efficienza.
  - *Contro*: Maggior complessità nella gestione delle due cache separate

MEMORIA VIRTUALE

Iniziamo il paragrafo andando a definire il concetto di processo. Un programma in Memoria Centrale in esecuzione è detto processo.

ALLOCAZIONE DELLA MEMORIA

La memoria principale è limitata e deve essere allocata efficientemente tra i processi. È divisa in due parti: una per i processi utente e una per il Sistema Operativo.

Allocazione della memoria può essere:

- **A partizione singola**: un solo processo alla volta può essere allocato, con SO in posizione fissa.
- **A partizioni multiple**: la memoria è divisa in partizioni logiche di dimensioni fisse o variabili.

Memory management unit (MMU): La rilocazione dinamica avviene tramite un registro di rilocazione. La MMU usa un addizionatore e un registro base per sommare l'indirizzo logico e ottenere l'indirizzo fisico.

ALLOCAZIONE MEMORIA A PARTIZIONI MULTIPLE

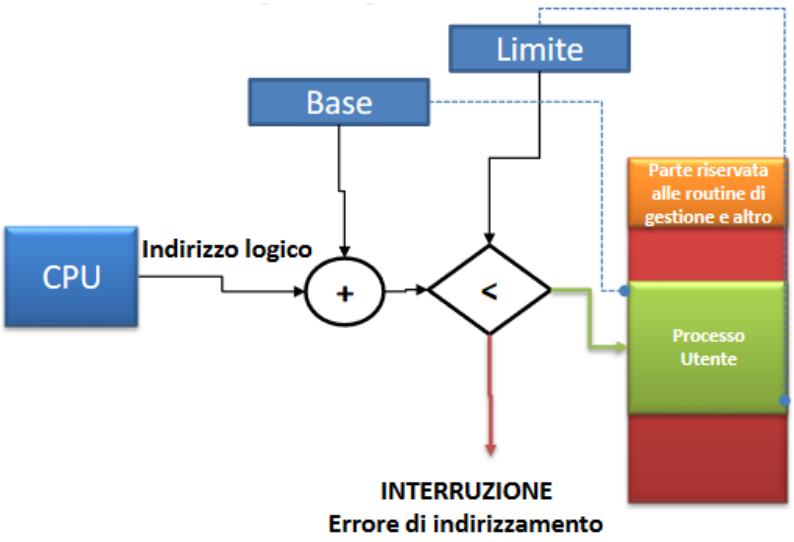
Dimensioni Fisse (MFT):

- Ogni partizione ha dimensioni fisse.
- *Problemi*: frammentazione interna, multiprogrammazione limitata, dimensione massima dei processi limitata.

Dimensioni Variabili (MVT):

- Partizioni allocate dinamicamente.
- *Vantaggi*: eliminazione frammentazione interna, multiprogrammazione variabile, dimensione massima processi più grande.
- *Problemi*: scelta area da allocare, frammentazione esterna (risolvibile con compattazione, a svantaggio delle prestazioni).

Poiché sono presenti più processi è opportuno disporre di un sistema di protezione, vengono infatti usati un **registro base** (prima istruzione) e un **registro limite** (ultima istruzione) per proteggere i processi in esecuzione, con un'interruzione in caso di errore di indirizzamento.



La **frammentazione** della memoria è un *problema significativo* che deve essere risolto per garantire l'efficienza e l'efficacia della gestione della memoria nei sistemi operativi.

Possiamo avere una frammentazione:

- **Interna**: memoria allocata può essere maggiore di quella richiesta, causando sottoutilizzo.
- **Esterna**: spazio disponibile non contiguo
    - Risolvibile con compattazione, che richiede rilocazione dinamica
    - I processi con I/O pendenti non possono essere spostati

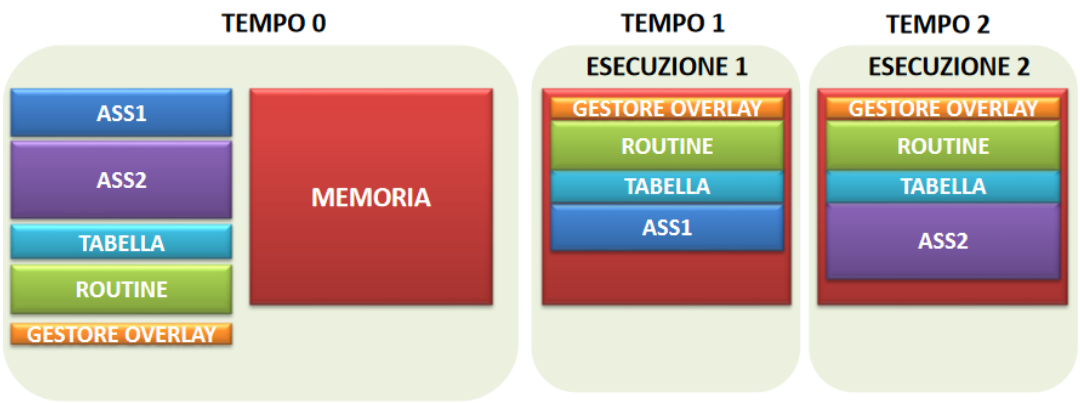
Nel caso in cui si voglia eseguire un programma che *richiede una area di memoria più grande* della partizione ammissibile o dello spazio fisico disponibile si può decidere di ricorrere ad una tecnica chiamata **Overlay**.

Concetto: Si basa sulla possibilità di mantenere in memoria solo le istruzioni e i dati utilizzati in un determinato momento o con maggior frequenza.

Operazioni di I/O: Quando sono necessarie altre istruzioni, queste vengono caricate nello spazio precedentemente occupato da istruzioni non più utilizzate, comportando operazioni di **swapping** che possono influenzare negativamente le prestazioni.

L'overlay è a discrezione e a completa **implementazione del programmatore**.

- **Gestione dell'Overlay**: Consiste nel caricare e scaricare dalla memoria solo le parti del programma necessarie, sostituendole ai dati/moduli caricati precedentemente nella zona dedicata.
- **Gestore di Overlay**: È una porzione di software che controlla il caricamento delle porzioni di memoria necessarie, mappando i riferimenti alle routine e ai piazzamenti.
- **Dati Comuni**: Solo i dati comuni e usati globalmente per l'applicazione devono rimanere in memoria.
- **Complessità**: L'uso dell'overlay è complicato e può causare comportamenti imprevedibili del processo se i caricamenti e scaricamenti non sono gestiti correttamente.
- **Limitazioni**: Confinato a sistemi con memoria limitata (primi elaboratori) e senza hardware sofisticato per supportare altre tecniche di gestione della memoria.



MEMORIA VIRTUALE (VM)

La **memoria virtuale** è una tecnica che permette di eseguire programmi senza che siano completamente contenuti nella memoria centrale (RAM). Il *vantaggio principale* che offre questa tecnica è quello di permettere che i programmi siano più grandi della Memoria Centrale.

Sebbene esista la tecnica di **overlay**, si *privilegia* una implementazione dell'uso della Memoria Virtuale perché solleva il programmatore dal gestire quale porzione di programma deve entrare e quale uscire dalla RAM di volta in volta.

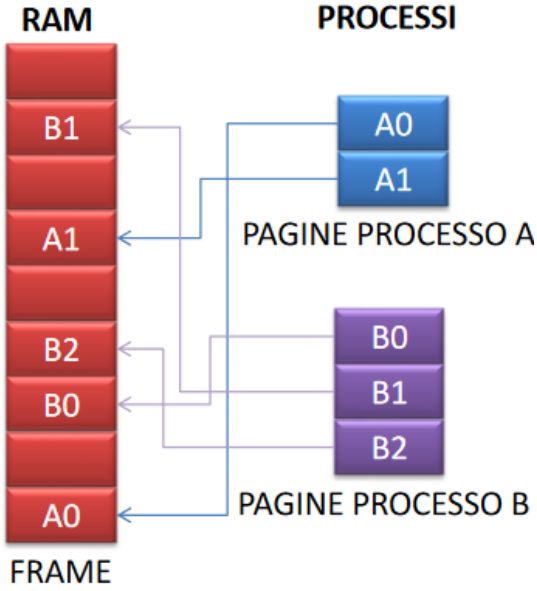
Inoltre, permette di aumentare il numero di processi caricati in memoria, migliorando la **multiprogrammazione** e riducendo i tempi di trasferimento dei processi dalla memoria di massa alla memoria centrale.

La Memoria Virtuale richiede la definizione e l'implementazione della tecnica della **paginazione**.

La **paginazione** è una tecnica di gestione della memoria che elimina la necessità di indirizzi contigui per lo spazio degli indirizzi fisici di un programma, per questa caratteristica, *risolve il problema della frammentazione esterna* e la necessità di compattazione.

Struttura della Paginazione:

- **Frame e Pagine**: La memoria fisica è suddivisa in porzioni di dimensione fissa chiamate frame. I programmi sono suddivisi in parti della stessa dimensione dei frame, chiamate pagine.
- **Caricamento Flessibile**: Ogni pagina può essere caricata in un frame della memoria fisica, utilizzando la RAM in modo più flessibile e senza necessità di contiguità.



Dimensione delle Pagine:

- **Dimensione delle Pagine**: Stabilita dall'architettura del calcolatore, solitamente compresa tra 512 byte e 16 MB.

Traduzione degli indirizzi:

- **Indirizzi Logici e Fisici**: La paginazione crea uno spazio di indirizzi logici separato dagli indirizzi fisici.
- **Tabella delle Pagine**: Utilizzata per trasporre gli indirizzi logici negli indirizzi fisici.
- **Schema di Traduzione degli Indirizzi**:
  - *Numero di Pagina* (p): Indice della tabella delle pagine contenente l'indirizzo iniziale di ciascuna pagina nella memoria fisica.
  - *Offset nella Pagina* (d): Combinato con l'indirizzo base per definire l'indirizzo fisico da raggiungere.

Vantaggi della Memoria Virtuale:

- Maggiore utilizzo della RAM disponibile.
- Riduzione delle operazioni di gestione manuale della memoria per i programmatori.
- Aumento del numero di processi eseguibili in parallelo, migliorando la multiprogrammazione.

Tabella dei frame: per allocare le pagine nei frame è necessario sapere quali frame sono assegnati e a quale pagina e sapere quali e quanti frame sono effettivamente liberi.

Queste informazioni sono contenute nella **lista di frame liberi**, una lista dove ogni elemento è associato ad un frame, indicando se libero o assegnato.

La **Memoria Virtuale** rappresenta la separazione della memoria logica, vista dall'utente, dalla memoria fisica presente nell'elaboratore. Questa separazione permette al programmatore una **memoria molto ampia**.

Residuo in Memoria Virtuale: I processi risiedono in una memoria virtuale, spesso costituita da un disco magnetico o memoria a stato solido.

Gestione delle Pagine: Il programma non è caricato interamente in memoria; le pagine necessarie sono trasferite da disco a RAM (PAGE IN) e da RAM a disco (PAGE OUT).

Paging: Tecnica che manipola singole pagine dei processi senza avvicendamento dei processi interi.

Paginatore: Il modulo che si occupa dello scambio delle pagine

- **Page Fault**: Si verifica quando la CPU richiede un indirizzo logico non presente in memoria centrale, causando un'interruzione (Page Fault).
- **Bit di Validità**: Nella tabella delle pagine, indica se una pagina è in memoria centrale, in memoria virtuale o se l'indirizzo è illecito.

Funzionamento della paginazione

- **Traduzione degli indirizzi**:
  - La CPU richiede un indirizzo logico.
  - Se il bit di validità è impostato a True, l'indirizzo fisico è ricostruito dalla tabella delle pagine.
  - Se il bit di validità è False, viene generata una Page Fault.
- **Gestione dell'Eccezione**:
  - *Controllo* della PCB (Process Control Block) per verificare la validità dell'indirizzo.
  - Se non valido, il processo è terminato; se valido, viene caricato dalla memoria virtuale.
  - *Individuazione* di un frame libero.
  - *Caricamento* della pagina tramite operazione di I/O.
  - *Aggiornamento* della tabella interna.
  - *Riavvio* dell'istruzione che ha causato il Page Fault.

La paginazione pura, dove un processo è caricato interamente in memoria virtuale, può causare una perdita di efficienza.

Politiche di caricamento: Quando una pagina richiesta non è presente in memoria centrale, si attivano i meccanismi di **prelievo** (fetch from virtual memory) o di **rimpiazzo** (replacement).

Ci sono due tipologie di politiche di **prelievo**:

- **On-Demand Paging**: Carica la pagina solo quando richiesta, inizialmente causando molti page fault.
- **Prepaging**: Carica più pagine contigue in anticipo, basandosi sul principio di località. Spesso, le pagine sono contigue anche nella memoria virtuale (ad esempio, tracce contigue su disco magnetico), migliorando i tempi di recupero e trasferimento.

Per quanto riguarda il **rimpiazzo** bisogna utilizzare un algoritmo di rimpiazzo, la qualità di un algoritmo dipende dal numero di page fault che genera, meno page fault significano una strategia migliore.

Algoritmi di rimpiazzo:

- **OPT** (Sostituzione Ottimale):
  - Utopistica, sostituisce la pagina che verrà usata più tardi. Non realizzabile in pratica poiché non è possibile determinare in anticipo quale pagina sarà usata per ultima.
- **LRU** (Least Recently Used):
  - Sostituisce la pagina meno usata di recente.
  - Ogni pagina ha un contatore che registra l'ultima volta che è stata usata. Durante un page fault, si sostituisce la pagina non utilizzata da più tempo.
- **FIFO** (First In – First Out):
  - Sostituisce la pagina presente in memoria da più tempo.
  - Ogni pagina ha un time-stamp che indica quando è stata caricata in memoria. Durante un page fault, si sostituisce la pagina più vecchia.
  - Semplice ma non sfrutta il principio di località.
- **CLOCK**:
  - Ogni pagina è organizzata in una coda circolare e ha un bit di riferimento settato a 1 quando viene acceduta.
  - Durante un page fault, un puntatore esamina il bit di riferimento di ogni pagina: se è 1, viene resettato a 0 e si passa alla successiva; se è 0, la pagina viene sostituita.

SISTEMA OPERATIVO

Ci sono diversi tipi di linguaggi di programmazione che si differenziano tra linguaggi di basso livello (orientati alla macchina) e linguaggi di alto livello (orientati all'uomo).

Un **Firmware** è un programma eseguibile che si avvia automaticamente dopo l'accensione della macchina (fase di bootstrap), risiede su una memoria preservante di tipo EEPROM ed è raramente aggiornato.

Il **Sistema Operativo**: insieme di programmi che gestiscono e coordinano le componenti fisiche dell'elaboratore, mettendo a disposizione un ambiente per installare, gestire ed eseguire i programmi applicativi, e permettono l'interazione utente-elaboratore senza conoscere tutti i dettagli dell'architettura.

Un **Sistema distribuito** è un insieme di processori che non condividono lo stesso clock e memoria ma possono scambiarsi dati attraverso una struttura di interconnessione.

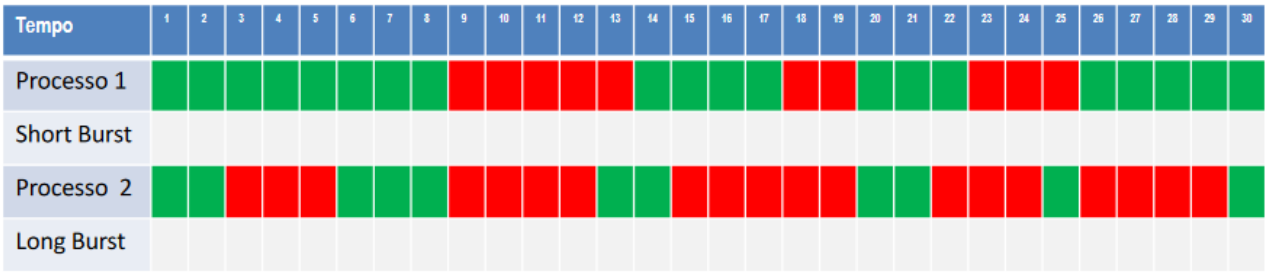
GESTIONE DEI PROCESSI

Ogni **processo** ha una tabella delle informazioni associate (PCB - Process Control Block), che include:

- *Stato del processo* (new, ready, running, waiting, halted)
- *Program Counter*: l'indirizzo della prossima istruzione da eseguire.
- *Registri CPU*: informazioni salvate durante un'interruzione.
- Informazioni sullo scheduling della CPU.

- Informazioni sulla gestione della memoria: registri base e limite, tabella delle pagine.
- Informazioni di contabilità: tempo utilizzato, slot temporale assegnato.
- *Informazioni I/O*: elenco file aperti, risorse I/O richieste.

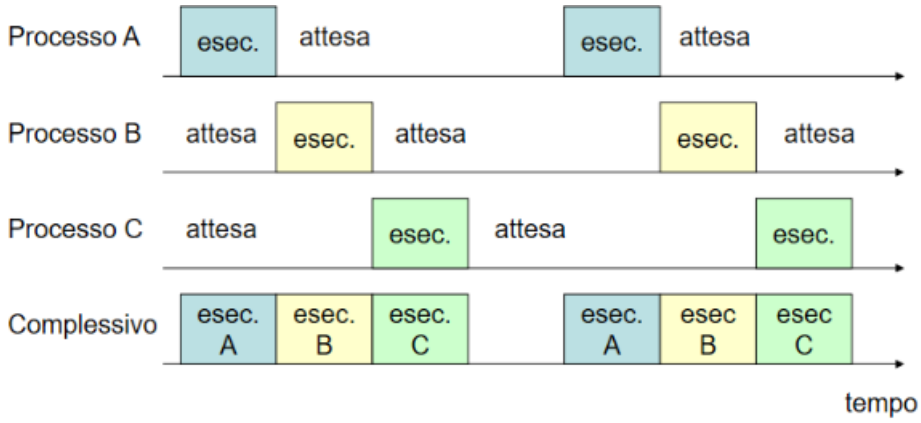
Durante lo svolgimento di un processo si assiste a periodi di tempo, brevi, impiegati dalla CPU per elaborare istruzioni (**CPU burst**) e intervalli, molto più lunghi e a volte assai frequenti, riservati per l'interazione con le periferiche o con l'utente (**I/O burst**).



Introduciamo il concetto di **multiprogrammazione**: la multiprogrammazione è una tecnica basata su interruzioni che consente l'alternanza di processi in modo da massimizzare l'uso della CPU.

L'idea alla base della multiprogrammazione è la seguente:

- I processi sono *contemporaneamente* nella RAM ed in modo concorrenziale ordinato condividono le risorse del sistema (tra cui la CPU)
- Finché un processo è in attesa di "qualcosa", un altro è in esecuzione nella CPU
- **Obiettivo**: la CPU non deve mai essere inattiva



Time-sharing:

È possibile condividere la CPU tra più processi suddividendo il tempo di esecuzione del processore. Ogni processo utilizza periodicamente un intervallo di tempo prestabilito (*time slice*; di solito 10-100ms) scandito da un **Timer**. Effetto di uno pseudo-parallelismo.

File System:

Parte del Sistema Operativo che consente di memorizzare e gestire i documenti digitali su un supporto di massa permanente.

Permette una visione astratta dei documenti digitali su memoria di massa permanente e permette all'utente di:

- **Identificare** ogni file mediante il suo nome
- **Operare** sui file mediante opportune operazioni (apertura, lettura, scrittura, chiusura, esecuzione)

- **Effettuare** l'accesso alle informazioni grazie ad operazioni ad alto livello (es.: fopen/fread/fwrite) che non richiedono la conoscenza del tipo di memorizzazione (si accede allo stesso modo ad un file memorizzato su HDD oppure su un CD)

Sfruttando il file system un utente può **strutturare** i file e **proteggerli** andando ad impedire a utenti non autorizzati di leggerli e modificarli.

Il **tipo di file** specifica il documento (testuale, binario, immagine, video, suono). Ci sono diverse operazioni che possono essere applicate: creazione, cancellazione, copia, visualizzazione, stampa, modifica, ridenominazione, visualizzazione delle proprietà.

I file sono contenuti in una **directory** (cartella) la quale aiuta l'organizzazione generale. Le cartelle sono organizzate in modo **gerarchico** con una cartella "root" e delle sottocartelle.

Il file è suddiviso in parti, ciascuna occupante un **blocco** del disco magnetico (HDD) o della memoria a stato solido (SSD).

Per questo motivo c'è bisogno di una struttura che ne descrive la posizione e che allo stesso tempo minimizza i tempi di accesso e massimizza l'utilizzo dello spazio:

- **Allocazione contigua**: In questa tecnica, ogni file occupa un insieme di blocchi contigui su memoria di massa. Il file descriptor riporta l'indirizzo del blocco di partenza e la lunghezza (numero di blocchi).
    - **Vantaggi**:
        - Accesso semplice e veloce (blocchi contigui).
    - **Svantaggi**:
        - Tempo sprecato nel trovare lo spazio adeguato.
        - Spreco di spazio dovuto alla frammentazione.
- **Allocazione a lista concatenata** (linked): Ogni file è una lista di blocchi, che possono essere sparsi ovunque. Il file descriptor contiene puntatori al primo e all'ultimo blocco. Ogni blocco contiene un puntatore al blocco successivo, con l'ultimo blocco che ha un puntatore NULL.
    - **Vantaggi**:
        - Semplice creazione e incremento di file.
        - Nessuno spreco, tranne per lo spazio del puntatore.
    - **Svantaggi**:
        - Accesso casuale non possibile.
        - Scarsa efficienza e affidabilità (perdita di un puntatore = perdita del file)
- **Allocazione indicizzata**: Ogni file ha un blocco indice (index block) contenente una tabella degli indirizzi (index table) dei blocchi fisici. Il file descriptor contiene l'indirizzo del blocco indice.
    - **Vantaggi:**
        - Accesso casuale efficiente.
        - Accesso dinamico senza frammentazione esterna.
    - **Svantaggi:**
        - La dimensione del blocco limita la dimensione del file.
        - Soluzione: uso di schemi a più livelli (es. indici multilivello o schema concatenato).

MULTIPROCESSORI

Nonostante i progressi fatti nel campo tecnologico si è arrivati a dover affrontare dei **limiti tecnici** ed economici come la velocità di trasmissione, impossibilità di miniaturizzare i componenti.

Per questo motivo si è aumentato il numero di **processori** ed è stato introdotto il **calcolo parallelo**, che può essere svolto con una tecnologia **multiprocessore** (più processori sulla scheda madre) e **multicore** (più "core", nucleo, montati sullo stesso chip).

Vi è una classificazione basata sulla nozione di **flusso di informazioni**:

- Architettura **SISD**: costituita da un solo processore, algoritmi sequenziali, nessun parallelismo
- Architettura **MISD**: n processori, ad ogni clock viene elaborato il dato su tutti i processori, (parallelismo a livello di istruzioni), perlopiù teorica
- Architettura **SIMD**: un processore *master* n-1 processori *slave* (dotati di propria memoria), ad ogni clock, tutti i processori eseguono la stessa istruzione su dati diversi (parallelismo sui dati)
- Architettura **MIMD**: Potente, ogni processore ha la sua memoria e una propria unità di controllo (parallelismo a livello di thread)

Livelli di parallelismo:

1. **Data Level Parallelism** (DLP):
    - I dati vengono suddivisi in segmenti che vengono processati simultaneamente
2. **Istruction Level Parallelism** (ILP):
    - Istruzioni distribuite ed eseguite da più processori contemporaneamente
3. **Thread Level Parallelism** (TLP):
    - Le applicazioni usano thread/processi concorrenti, ovvero che sono eseguiti in parallelo su più processori

MEMORIA CONDIVISA

I sistemi multiprocessore a **memoria condivisa** si caratterizzano per l'accesso comune alla memoria da parte di più processori. Questo approccio comporta diversi **problemi di coerenza** della cache e sincronizzazione, ma offre vantaggi in termini di velocità di accesso ai dati condivisi.

Questo problema *sorge* quando un processore modifica un dato nella memoria principale che è contemporaneamente utilizzato da altri processori.

Il nuovo valore deve essere *propagato* dalla cache del processore che l'ha modificato alla memoria condivisa e quindi a tutte le altre cache dei processori, evitando che essi lavorino con valori obsoleti.

La *risoluzione* richiede implementazioni hardware che gestiscono problemi di concorrenza e sincronizzazione, simili ai controlli sui thread a livello di programmazione.

Caratteristiche:

- **Memoria logica comune**: Tutti i processori lavorano con gli stessi indirizzi logici, accedendo alle stesse locazioni di memoria.
- **Sincronizzazione**: La memoria condivisa viene concessa a turno ai vari processori, assicurando che una locazione non venga modificata mentre un altro processore vi accede.
- **Velocità di condivisione dati**: Rapida comunicazione tra processori, poiché il tempo di accesso alla memoria è quello richiesto per leggere una singola locazione.
- **Scalabilità limitata**: Limitata dal numero di vie d'accesso alla memoria, che può causare attese e aumentare i tempi di latenza.

Organizzazione:

- **Uniform Memory Access** (UMA): Tempo di accesso alla memoria costante per ogni processore e locazione di memoria (es. Symmetric Multiprocessor, SMP).
- **Non-Uniform Memory Access** (NUMA): Memoria suddivisa in zone ad alta velocità assegnate a ciascun processore e una zona comune a accesso più lento (Distributed Shared Memory Systems, DSM).

MEMORIA DISTRIBUITA

In un sistema a **memoria distribuita**, la memoria è associata ai singoli processori, che possono indirizzare solo la propria memoria locale. Questa configurazione *elimina conflitti di bus* e *problemi di coerenza* della cache, ma complica la comunicazione tra processori.

Caratteristiche:

- **Memoria fisicamente distribuita**: Ogni processore accede direttamente solo alla propria memoria locale.
- **Sincronizzazione tramite Message Passing**: Comunicazione tra processori tramite scambio di pacchetti dati discreti.
- **Suddivisione accurata dei dati**: Minimizzare la comunicazione tra processori per ottimizzare le prestazioni.

Organizzazione:

- **NO-Remote Memory Access** (NORMA): Memoria distribuita fisicamente tra i processori, con memorie locali private e comunicazione tramite protocollo di scambio di messaggi.
- **Cache Only Memory Access** (COMA): Sistemi dotati solo di memorie cache, eliminando duplicazioni dei dati nella memoria principale

Classificazione dei sistemi multiprocessori a memoria distribuita:

- **Massively Parallel Processing** (MPP):
  - Composto da centinaia o migliaia di processori collegati da una rete di comunicazione.
  - Utilizzato nelle macchine più veloci del pianeta.
- **Cluster of Workstations** (CoW):
  - Basato su calcolatori collegati da reti di comunicazione.
  - Include cluster di calcolo che rientrano in questa classificazione.

Un **Thread** è una suddivisione di un processo in due o più sottoprocessi.

Per **Multithread** si intende la capacità di un processore di passare da un thread all'altro

Il multithreading permette a diversi thread di condividere le risorse hardware di un unico processore in modo tale che esso possa eseguirli in parallelo.

I tre tipi principali di hardware multithreading sono:

- **Fine-Grained Multithreading (FMT)**
  - consente l'alternanza di thread ad ogni ciclo di clock
  - Alternanza creata con l'algoritmo Round Robin con politica FIFO
  - *Vantaggio*: sia per stalli brevi che per stalli lunghi, mantiene un certo rendimento
  - *Svantaggio*: rallenta l'esecuzione di un singolo thread

- **Coars Grained Multithreading (CMT)**
    - Rende più efficiente l'utilizzo delle unità funzionali mediante l'esecuzione di un solo thread alla volta, per un certo numero di cicli di clock. *L'efficienza* è dovuta appunto all'eliminazione di questi cicli di clock inattivi
    - Il processore, appena rileva che un thread è in stato di stallo prolungato, passa immediatamente all'esecuzione di un altro thread.
    - *Svantaggio*: risulta totalmente inefficace nel caso in cui gli stalli siano molto piccoli.
- **Simultaneous Multithreading (SMT)**
    - esegue istruzioni provenienti da diversi thread, in qualsiasi momento, in una qualsiasi unità funzionale
    - *Vantaggio*: In questo multithreading il processore è in grado di gestire un parallelismo sia a livello di thread sia a livello di istruzioni
    - Incrementare o decrementare le dimensioni della cache nei vari livelli, può portare a vantaggi o a svantaggi

MEMORIE DI MASSA

Le **memorie di massa** sono nate per l'archiviazione di grandi quantità di dati, quando consentono la memorizzazione di documenti digitali in maniera permanente, sono denominate **memorie persistenti**.

Vi è una classificazione in accordo al metodo di accesso ai dati:

- **Accesso sequenziale**: per acquisire un dato è necessario scorrere tutti quelli che lo precedono
- **Accesso diretto**: è possibile reperire un dato all'indirizzo in cui risiede
- **Accesso indicizzato**: in cui si raggiunge il dato spostandosi ad una posizione prossima a dove esso risiede e reperirlo attraverso una scansione sequenziale

Le memorie di massa possono essere classificate anche in base alla tipologia di materiale di composizione: **Magnetico** (nastro, disco), **Ottico** (CD, DVD), a **stato solido**

Un **documento digitale** è una collezione di dati organizzata secondo un determinato formato, ed è archiviato su un **supporto digitale**.

La digital preservation si occupa di garantire accesso ai dati nel tempo e conservazione dei dati.

NASTRO MAGNETICO

Il **nastro magnetico** è costituito da uno strato di particelle magnetiche depositate su un supporto flessibile, le principali componenti sono: Rivestimento dorsale, Base, Strato Magnetico, Legante.

Funzionamento:

- Ogni particella magnetica ha un verso di magnetizzazione dovuto alla posizione dei propri poli, un insieme di particelle definiscono un **Dominio**, il quale può cambiare verso di magnetizzazione.

Scrittura delle Informazioni:

- La scrittura avviene modificando l'orientamento dei domini tramite un elettromagnete (testina induttiva).
- La corrente inviata lungo la spira crea un campo magnetico che influenza l'orientamento dei domini sottostanti.

Lettura delle Informazioni (Prima Generazione):

- La lettura si realizza leggendo le variazioni del campo magnetico.
- Un esempio di segnale binario: una coppia di flussi inversi (domini con orientamento opposto ai confini) rappresenta un '1', mentre il passaggio da un flusso non inverso a uno inverso rappresenta uno '0'.

Tecnologie Moderne di Scrittura-Lettura:

- MR (Magneto-Resistenza): Incisione dei dati in orizzontale con elettromagneti di ridotte dimensioni utilizzando ossidi di ferro come particelle magnetiche.
- GMR (Magneto-Resistenza Gigante): Incisione dei dati in verticale con elettromagneti monopolo e lettura con testina GMR, utilizzando un composto di Ferro e Bario (BaFe) come materiale magnetico.

Organizzazione Logica

Prima Generazione: Lo strato magnetico è suddiviso in tracce (o piste) e blocchi. I file erano scritti e letti in maniera sequenziale.

- Marcatori di limitazione:
  - BOT: Inizio del nastro
  - EOT: Fine del nastro
  - EOF: Fine del file
  - EOD: Fine dei dati

Codice per il Rilevamento di Errori: Si utilizzano ECC (Codici di Correzione degli Errori) per rilevare eventuali errori nei blocchi. Nei vecchi nastri si usava il doppio bit di parità. Gli errori possono essere corretti immediatamente anche nei nuovi metodi.

Nuova Generazione (LTO): Il blocco nel formato LTO è strutturato in:

- *Preambolo di Sincronizzazione*: Segnala al controller del disco che si leggeranno dati.
- *Marcatore di Inizio*: Indica al controller il punto del nastro.
- *Dati*: Contenuto del blocco (512 byte).
- *Indirizzo del Blocco*: Per sapere dove risiedono i dati.
- *Informazioni Ridondanti*: Per il rilevamento e la correzione di errori (CRC di 2 byte).

DISCO MAGNETICO

Il **disco magnetico** è costituito da uno strato di particelle magnetiche depositato su un supporto fisso, le cui componenti principali sono *Base* e *Strato Magnetico*.

Funzionamento:

I *dati* sono rappresentati da porzioni magnetiche (dominio) in cui le particelle hanno un determinato orientamento

La *scrittura e la lettura* delle informazioni avvengono modificando o leggendo l'orientamento dei domini

Nei nuovi dischi si usano testine GMR che consentono la registrazione perpendicolare (offre maggiore capacità) per cui si analizza il dominio considerando il suo magnetismo in verticale.

Organizzazione Logica:

Tracce e Settori:

- Lo strato magnetico è suddiviso in tracce concentriche.
- Ogni traccia è organizzata in settori, che corrispondono a segmenti di arco della traccia.
- Più settori contigui formano un cluster

Densità dei Dati:

- *Densità Costante*: I settori periferici contengono più dati rispetto a quelli prossimi al centro.
- *Densità Variabile*: Ogni settore contiene la stessa quantità di dati, con domini più grandi nei settori esterni e più compatti in quelli interni.

Dischi e Cilindri:

- Ogni disco (o piatto) ha due facce o superfici.
- Le stesse tracce su dischi differenti formano un cilindro.
- Ogni faccia del disco ha una testina di lettura e scrittura.
- Il movimento delle testine sullo stesso piatto è coordinato, e può essere radiale o trasversale

DISCO OTTICO

I supporti ottici, la cui lettura e scrittura dei dati avvengono tramite un raggio laser, hanno una struttura a strati di materiali eterogene, le cui componenti principali sono Substrato, Strato dati, Strato riflettente, Strato protettivo.

Funzionamento

I supporti ottici, la cui lettura e scrittura dei dati avviene tramite un raggio laser, hanno una struttura a strati di materiali eterogene

Rappresentazione dei Dati:

- I dati sono rappresentati da pit (depressioni) e land (spianate).
- I dati sono disposti lungo una traccia a spirale, che parte dal centro e termina alla periferia.
- La traccia è organizzata in settori, e più settori contigui formano un blocco.

Lettura dei Dati:

- Un raggio laser segue la traccia. La luce polarizzata riflessa dallo strato riflettente varia in intensità quando incontra pit e land.
- Un fotodiodo converte queste variazioni in segnali digitali.
- La transizione tra pit e land individua un '1' logico, mentre la lunghezza delle aree (pit o land) rappresenta gli '0' logici.
- I pit possono avere lunghezza variabile da 3T (3 zeri consecutivi) a 11T (11 zeri consecutivi).

Scrittura dei Dati:

- Supporti di Sola Lettura (CD-ROM, DVD-ROM, BR-ROM): I pit sono impressi su un disco master e poi stampati su dischi in policarbonato.
- Supporti Registrabili una Volta (CD-R, DVD-R, BR-R): Una pellicola organica viene bruciata da un laser per creare aree riflettenti e non riflettenti.
- Supporti Riscrivibili (CD-RW, DVD-RW, BR-RW): Un materiale cristallino viene alterato dal laser per passare da stato cristallino (riflettente) a stato amorfo (non riflettente).

MEMORIA A STATO SOLIDO

L'architettura delle **memorie a stato solido** (o non volatile RAM, nvRAM) è una matrice in cui ad ogni intersezione di riga e colonna è presente una cella di memoria

Struttura del Transistore a Griglia Fluttuante

- Utilizza il floating-gate MOSFET (FGMOS), noto anche come transistor flash o cella FLOTOX.
- È un semiconduttore a base di silicio con una source (entrata) e un drain (uscita).
- Gli elettroni si muovono dal source al drain attraverso il substrato.
- La griglia fluttuante (floating gate) immagazzina le informazioni, intrappolata tra strati isolanti.
- Tra la griglia fluttuante e il substrato si trova la zona di tunnel, soggetta a degrado.

Funzionamento della Cella di Memoria

Scrittura:

- Applicazione di 7V al drain induce il movimento degli elettroni dalla source.
- Applicazione di 12V al control gate permette agli elettroni di rimanere intrappolati nella griglia fluttuante.

Lettura:

- Applicazione di 1V al drain induce il movimento degli elettroni dalla source.
- Il numero di elettroni intrappolati nella griglia fluttuante determina la fluidità del passaggio degli elettroni.
- Maggiore intrappolamento degli elettroni ostacola il passaggio (stato logico 0), mentre un passaggio fluido indica uno stato logico 1.
- La tensione di uscita varia in base al numero di elettroni, consentendo la memorizzazione di 2, 4 o 8 stati differenti (multi-data cell).

Cancellazione:

- Applicazione di -9V al control gate e 6V alla source induce il ritorno degli elettroni intrappolati nella source.
- La griglia fluttuante viene liberata, riportando la cella allo stato iniziale.
- La cancellazione avviene su più celle adiacenti (un blocco) per motivi progettuali

Organizzazione Fisica

Struttura:

- Le celle sono organizzate in pagine.
- Ciascuna pagina è formata da segmenti (chunck) con il relativo codice di correzione.
- Più pagine formano blocchi.
- Una memoria a stato solido (MSS) contiene più blocchi.
- I dati sono recuperati con accesso casuale, simile alla RAM.

Accesso e Modifica: Nelle memorie NAND Flash, la lettura e scrittura avvengono a livello di pagina, mentre la cancellazione avviene a livello di blocco.

Cancellazione di un Blocco

Problema della Frammentazione:

- Le pagine invalidate non possono memorizzare nuovi dati finché l'intero blocco non è cancellato.
- La frammentazione può esaurire lo spazio di archiviazione se i blocchi non vengono cancellati

Garbage Collection (GC):

- Il controller della MSS esegue periodicamente il GC per ridurre l'impatto della frammentazione.
- Il GC identifica blocchi frammentati e migra le pagine valide su blocchi liberi, liberando spazio.
- L'indirizzo fisico viene mappato con quello logico creando un indirizzo virtuale