

# PRINCIPI DELL'HARDWARE DI I/O

Danilo Croce

Gennaio 2025



# GESTIONE DELL'I/O NEL SISTEMA OPERATIVO

- **Funzione Principale:**

- Controllo dei dispositivi di I/O: invio di comandi, intercettazione di interrupt, gestione degli errori.
- Fornire un'interfaccia semplice e uniforme tra dispositivi e il sistema (indipendenza dai dispositivi).

- **Importanza nel Sistema Operativo:** Il codice per l'I/O costituisce una parte significativa del sistema operativo.

- **Argomenti:**

- **Principi dell'Hardware per l'I/O:** Fondamenti dell'hardware specifico per l'I/O.
- **Software per l'I/O:** Strutturato in livelli, ciascuno con funzioni specifiche.



# OVERVIEW

- Principi dell'hardware di I/O
- Livelli del software di I/O



# OVERVIEW

- **Principi dell'hardware di I/O**
- Livelli del software di I/O



# PRINCIPI DELL'HARDWARE DI I/O

- **Diverse Prospettive:**

- **Ingegneri Elettronici:** Vedono l'hardware di I/O in termini di componenti fisici (chip, cavi, alimentatori, motori, etc.).
- **Programmatori:** Interessati all'interfaccia software dell'hardware, inclusi i comandi accettati, le funzioni eseguibili e i possibili errori.

- **Focus del Corso:**

- **Concentrazione sulla programmazione dei dispositivi di I/O**, non sulla loro progettazione, costruzione o manutenzione.
- Interesse specifico **nella interazione con l'hardware**, piuttosto che nel suo funzionamento interno.

- **Connessione tra Programmazione e Operatività Interna:**

- La programmazione di molti dispositivi di I/O è spesso legata all'operatività interna dei dispositivi stessi.



# DISPOSITIVI DI I/O

- **Categorie dei Dispositivi di I/O:**

- **Dispositivi a Blocchi:** Archiviazione di informazioni in blocchi di dimensioni fisse (512 a 32.768 byte).
  - **Esempi:** dischi fissi magnetici, unità SSD, unità a nastro magnetico.
  - **Caratteristica chiave:** ogni blocco può essere letto o scritto indipendentemente.
- **Dispositivi a Caratteri:** Flusso di caratteri senza struttura a blocchi.
  - Non indirizzabili, senza operazioni di ricerca.
  - **Esempi:** stampanti, interfacce di rete, mouse.

- **Limitazioni della Classificazione:**

- Alcuni dispositivi non si adattano perfettamente a questa classificazione (es. clock, schermi mappati in memoria, touch screen).
- Modello comunque utile per astrarre alcuni software di I/O nel sistema operativo.

- **Applicazioni nel Sistema Operativo:**

- File system gestisce dispositivi a blocchi astratti.
- Software di livello inferiore gestisce le specificità dei dispositivi.



# DISPOSITIVI DI I/O (2)

- Dispositivi di I/O possono variare notevolmente in termini di velocità di trasferimento, creando sfide per il software di gestione.

Dispositivo	Velocità di trasferimento
<b>Tastiera</b>	10 byte/s
<b>Mouse</b>	100 byte/s
<b>Modem a 56 K</b>	7 KB/s
<b>Bluetooth 5 BLE</b>	256 KB/s
<b>Scanner a 300 dpi</b>	1 MB/s
<b>Videocamera digitale</b>	3,5 MB/s
<b>Wireless 802.11n</b>	37,5 MB/s
<b>USB 2.0</b>	60 MB/s
<b>Disco Blu-ray 16x</b>	72 MB/s
<b>Gigabit Ethernet</b>	125 MB/s
<b>Disco fisso SATA 3</b>	600 MB/s
<b>USB 3.0</b>	625 MB/s
<b>Bus PCIe 3.0 single lane</b>	985 MB/s
<b>Wireless 802.11ax</b>	1,25 GB/s
<b>SSD NVME PCIe Gen 3.0 (lettura)</b>	3,5 GB/s
<b>USB 4.0</b>	5 GB/s
<b>PCI Express 6.0</b>	126 GB/s





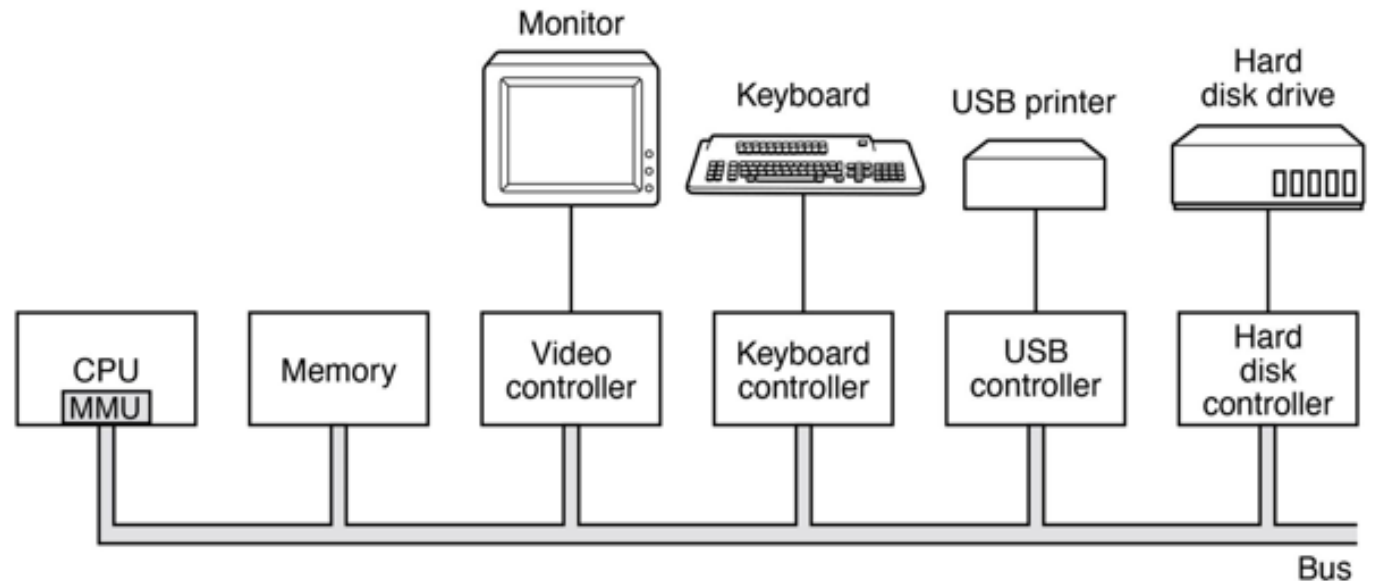
# CONTROLLER DEI DISPOSITIVI - PANORAMICA GENERALE

- **Componenti dei Dispositivi di I/O:**

- Composti da una parte meccanica (il dispositivo stesso) e una elettronica (controller del dispositivo o adattatore).
- Controller spesso integrato nella scheda madre o come scheda aggiuntiva su slot PCIe.

- **Connettività e Gestione Multi-dispositivo:**

- Controller con connettori per il collegamento a dispositivi.
- Capacità di gestire più dispositivi identici.





# CONTROLLER DEI DISPOSITIVI - FUNZIONAMENTO E EVOLUZIONE

- **Standardizzazione dell'Interfaccia:**

- Interfacce conformi a standard (ANSI, IEEE, ISO) o de facto (SATA, SCSI, USB, ThunderBolt).
- Permette la produzione di dispositivi e controller compatibili da diverse aziende.

- **Interfaccia di Basso Livello:**

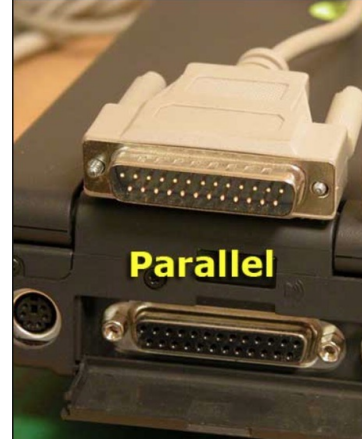
- **Esempio:** flusso seriale di bit in dischi, con preambolo, dati e ECC.
- **Compito del controller:** convertire il flusso seriale in blocchi di byte, correggere errori, trasferire in memoria principale.

- **Impatto sulla Programmazione del Sistema Operativo:**

- **Senza il controller, il programmatore dovrebbe gestire dettagli complessi** come la modulazione di ciascun pixel.
- Controller **inizializzato con parametri essenziali, gestisce autonomamente** dettagli complessi.



# DALLA “VECCHIA” PORTA PARALLELA...



## ■ Definizione e Utilizzo:

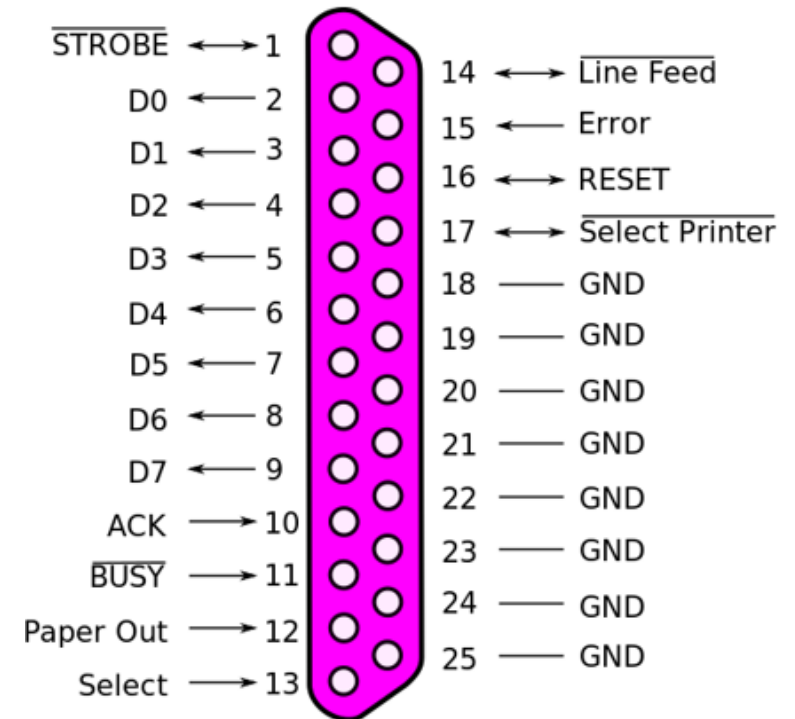
- Tipo di interfaccia di comunicazione tra computer e dispositivi (es. stampanti).
- Trasmette dati multi-bit simultaneamente su più canali (pin).

- Comunemente dotata di una connessione a 25 pin (standard DB-25) o 36 pin (Centronics).

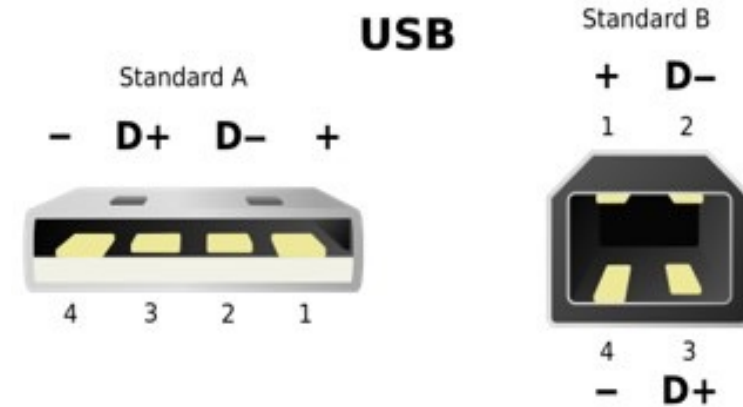
## ■ Distribuzione e Funzione dei Pin (Standard DB-25):

- **Pin 1-8:** Dati (D0-D7) - Trasmettono i dati effettivi.
- **Pin 9-16:** Controlli e Status - Includono pin di selezione della stampante, inizializzazione, linefeed, errori, occupato, ack, etc.
- **Pin 17-25:** Masse e alimentazione - Forniscono il ritorno elettrico e la connessione di terra.

- **Caratteristiche Tecniche:** Velocità variabile a seconda della modalità e del dispositivo collegato.



# ... ALLA PORTA USB



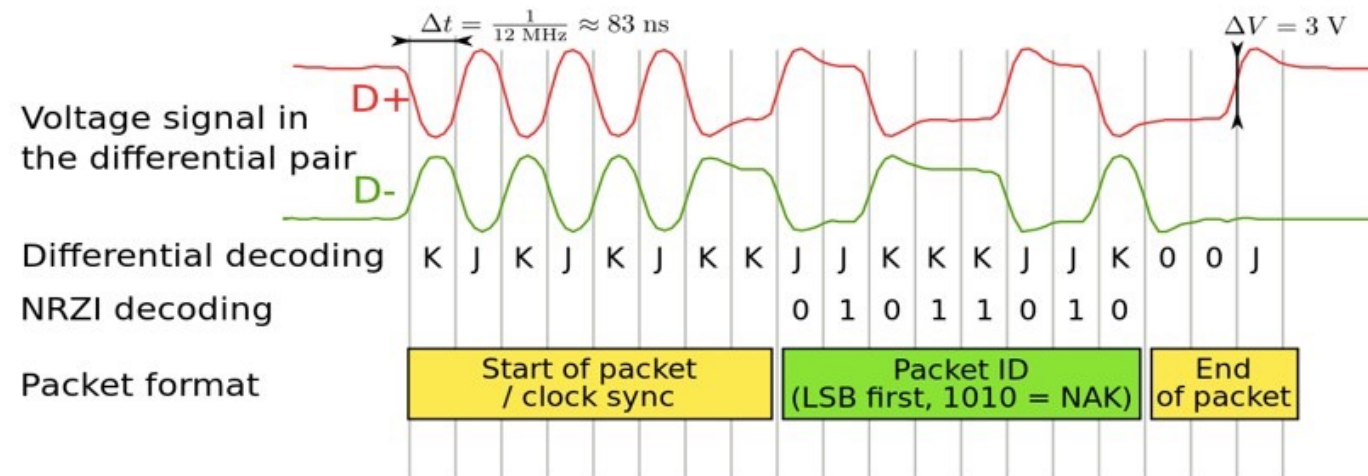
## Definizione e Utilizzo:

- L'USB (Universal Serial Bus) è un'interfaccia standardizzata per la connessione e comunicazione tra dispositivi e computer.
- Utilizzata per trasferire dati e fornire alimentazione elettrica.

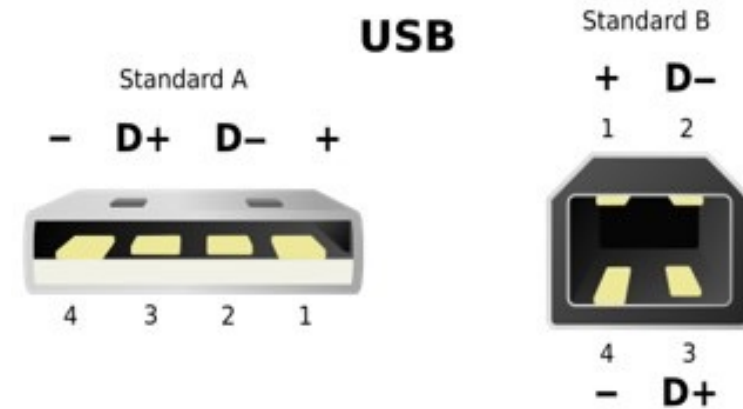
## Struttura dei Pin (Tipico USB 2.0):

- Pin 1 (Vcc):** Alimentazione (+5V).
- Pin 2 (D-):** Dati negativi.
- Pin 3 (D+):** Dati positivi.
- Pin 4 (GND):** Terra.

Pin	Name	Cable color	Description
1	VCC	Red	+5 VDC
2	D-	White	Data -
3	D+	Green	Data +
4	GND	Black	Ground



# ... ALLA PORTA USB (2)



## ■ **Caratteristiche Tecniche:**

- Supporta versioni USB 1.x, 2.0, 3.x, con velocità di trasferimento da 1,5 Mbps a 10 Gbps.
- USB 2.0 comunemente utilizzato per periferiche come tastiere, mouse, e dispositivi di archiviazione.

## ■ **Applicazioni e Vantaggi:**

























- Ampia applicazione in vari dispositivi come smartphone, periferiche di computer, e dispositivi di archiviazione.
- Facilità d'uso, connessione plug-and-play, supporto per trasmissione dati e alimentazione.

## ■ **Compatibilità:**

- Retrocompatibile con versioni precedenti, assicurando ampio supporto per vari dispositivi.



# MOLTE PORTE USB

USB 1.0	USB 2.0	USB 3.0 USB 3.1 Gen 1 USB 3.2 Gen 1	USB 3.1 Gen 2 USB 3.2 Gen 2	USB 3.2 Gen 2x2
12 Mbit/s	480 Mbit/s	5 Gbit/s	10 Gbit/s	20 Gbit/s
  Type A  Type B   Mini-A    Mini-B   Micro-A    Micro-B	  Type A  Type B   Mini-A    Mini-B   Micro-A    Micro-B	  Type A  Type B  Mini-B  Micro-B	  Type A  Type-C	  Type-C





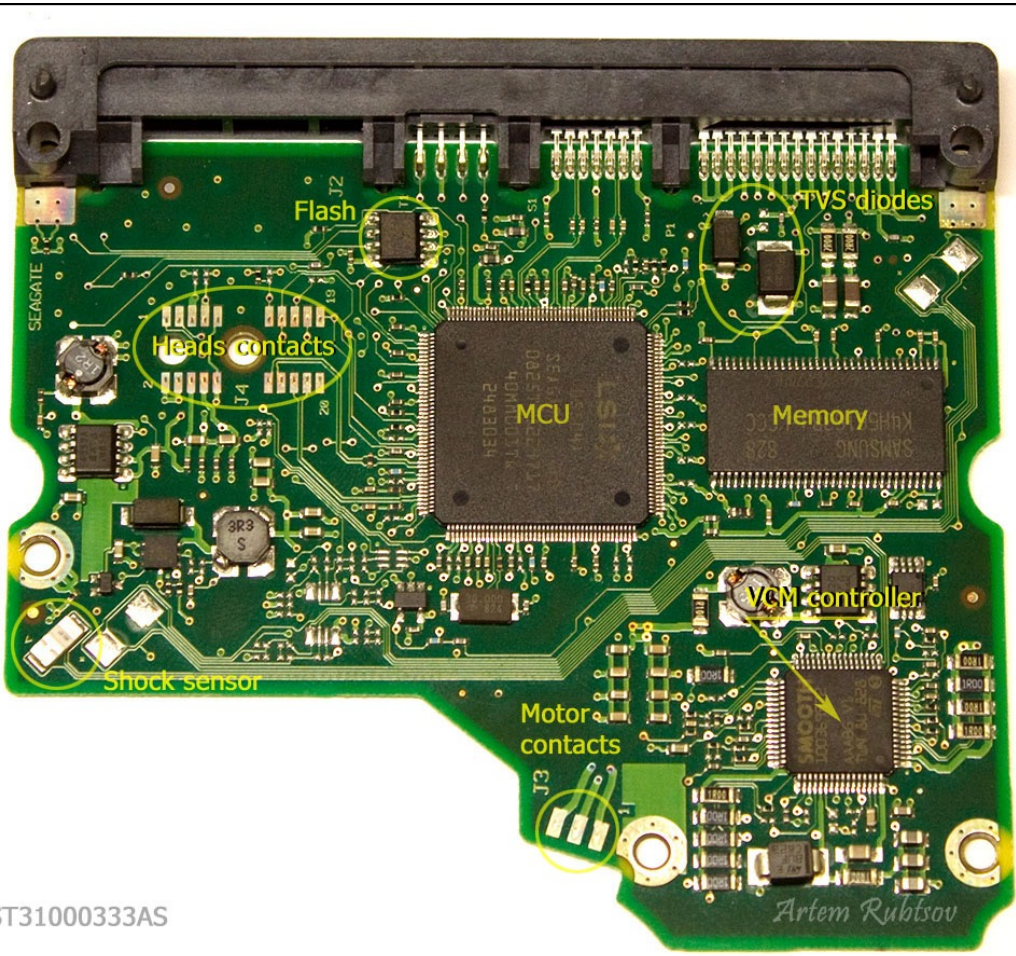
# E IL DISCO?



da: [https://hddscan.com/doc/HDD\\_from\\_inside.html](https://hddscan.com/doc/HDD_from_inside.html)



# UNO SGUARDO AI CIRCUITI

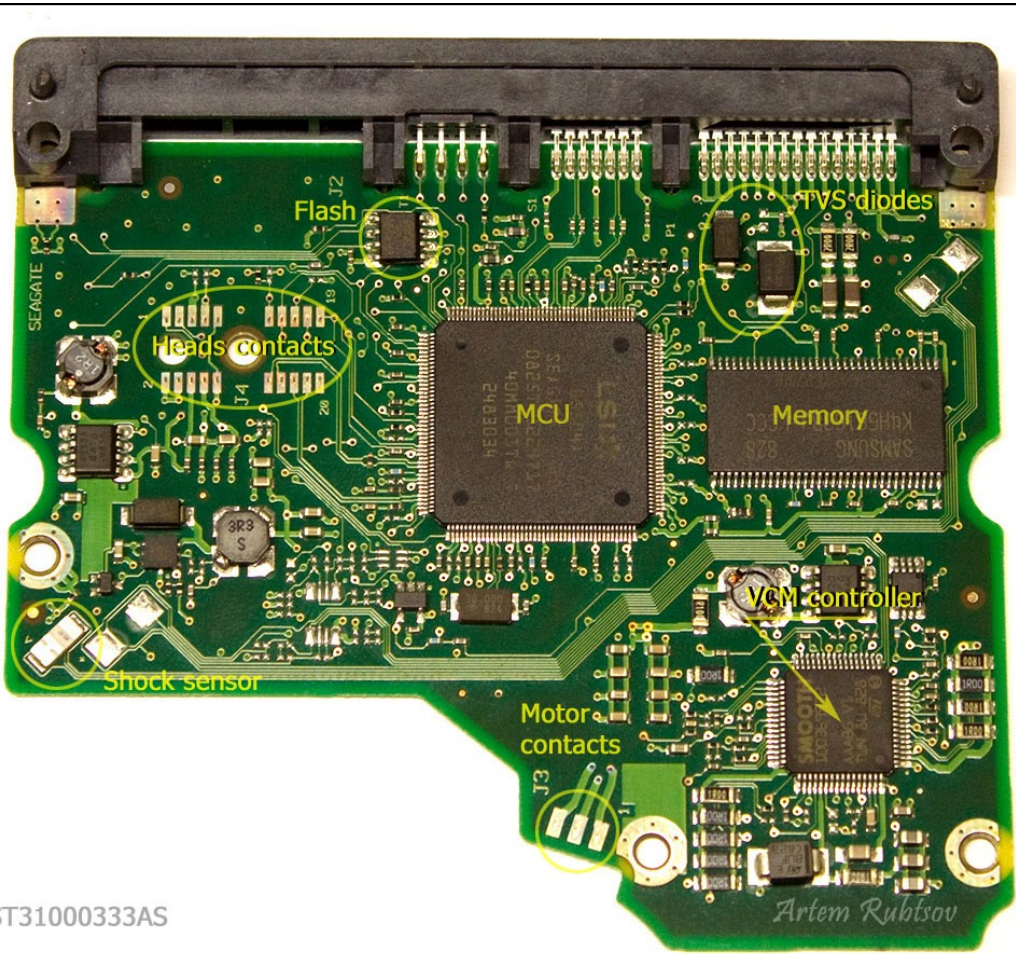


- Il **PCB** (Printed Circuit Board) è realizzato in vetro intrecciato verde e rame, dotato di connettori SATA e di alimentazione.
  - Supporta e collega i componenti elettronici dell'HDD.
  - Comunica con il controller SATA e il BUS
- Al centro del PCB si trova il MCU (**Micro Controller Unit**), il chip più grande
  - essenziale per le operazioni dell'HDD.
  - Il MCU include
    - una CPU (Central Processor Unit)
    - un canale di lettura/scrittura per convertire i segnali analogici in digitali.
- **Cache:** La memoria è un chip di tipo DDR SDRAM.
  - La sua capacità definisce la cache dell'HDD.
  - **Esempio:** un chip da 32MB indica una cache teorica di 32MB.





# UNO SGUARDO AI CIRCUITI (2)

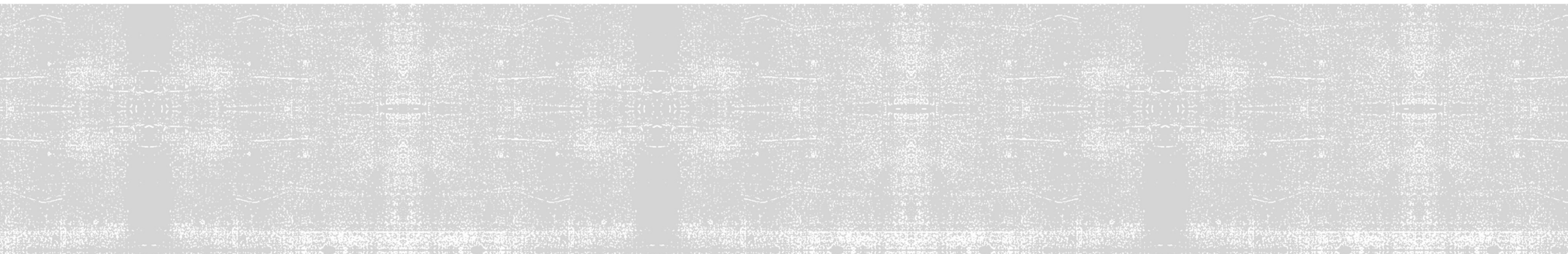


- Il controller VCM (**Voice Coil Motor**) controlla la rotazione del motore del disco e i movimenti delle testine
  - consumando la maggior parte dell'energia elettrica sul PCB.
- Il chip flash memorizza parte del firmware dell'HDD, essenziale per l'avvio.
- Sensori di shock e diodi TVS (**Transient Voltage Suppression**) proteggono l'HDD da urti e sovratensioni.
- Il sensore di shock rileva urti eccessivi, inviando segnali al controller VCM per proteggere l'HDD.
- I diodi TVS proteggono da sovratensioni, sacrificandosi in caso di picchi di tensione.





**MA COME FANNO CPU E  
DISPOSITIVO A  
COMUNICARE?**



# INPUT/OUTPUT MAPPATO IN MEMORIA

- **Registri dei Controller:**

- Ogni controller di dispositivo ha registri per comunicare con la CPU.
- **Scrittura nei registri:** invia comandi al dispositivo (trasferimento dati, accensione/spegnimento, altre azioni).
- **Lettura dai registri:** verifica lo stato del dispositivo e la prontezza per nuovi comandi.

- **Buffer di Dati:**

- Molti dispositivi includono un buffer di dati per scrittura/lettura del sistema operativo.
- **Esempio:** RAM video utilizzata per visualizzare punti sullo schermo.

- **Comunicazione CPU-Dispositivo:** Come la CPU comunica con i registri di controllo e i buffer dei dati? Due approcci principali:

1. Port-mapped I/O
  2. Memory-mapped I/O
- + (bonus) Approccio Ibrido (Port-mapped I/O + Memory-mapped I/O )

- **Importanza nel Sistema Operativo:**

- Cruciale per gestire efficientemente il trasferimento di dati e il controllo dei dispositivi.
- Incide sulla progettazione e sulle prestazioni del sistema.

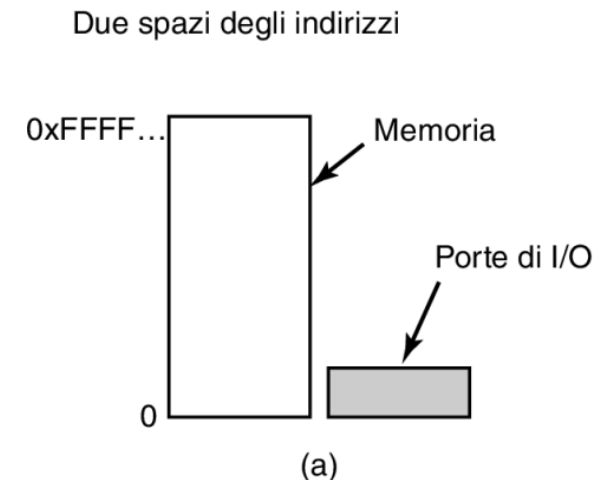




# ALTERNATIVA 1

## PORT-MAPPED I/O (PMIO)

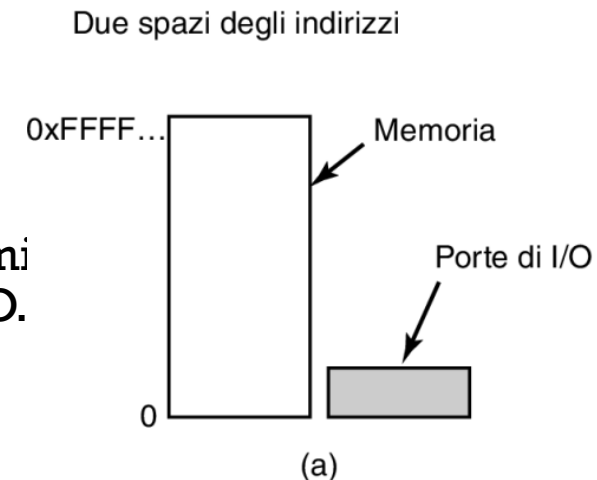
- **Assegnazione delle Porte di I/O:**
  - Ogni registro di controllo ha un numero di porta di I/O associato (intero di 8 o 16 bit).
  - Formano lo spazio delle porte di I/O, accessibile solo dal sistema operativo per motivi di protezione.
- **Istruzioni di I/O Speciali:**
  - **Lettura:** `IN REG, PORT`
    - La CPU legge dal registro di controllo `PORT` e salva il risultato in `REG`.
  - **Scrittura:** `OUT PORT, REG`
    - La CPU scrive il contenuto di `REG` in un registro di controllo.
- Ampio uso in vecchi computer e mainframe (es. IBM 360).



# ALTERNATIVA 1

## PORT-MAPPED I/O (PMIO)

- **Separazione:** Spazi di indirizzi della memoria e dell'I/O sono distinti e non correlati (a).
  - Esempi di istruzioni:
    - `IN R0, 4` Legge dalla porta di I/O 4 e salva in R0.
    - `MOV R0, 4` Legge dalla parola di memoria 4 e salva in R0.
  - Numero identico (es. 4) riferisce a spazi di indirizzi diversi (primo I/O, secondo di memoria)
- **Impatto sul Design di Sistema:**
  - Questo approccio influisce sul design e sulla programmazione dei sistemi operativi, differenziando nettamente la gestione della memoria e dell'I/O.



# ALTERNATIVA 2

## MEMORY-MAPPED I/O (MMIO)

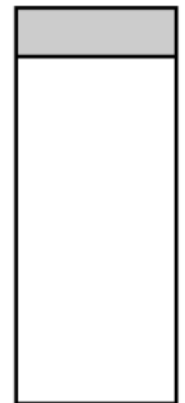
- **Approccio I/O Mappato in Memoria:**

- Introdotta con il PDP-11, questa metodologia assegna a ogni registro di controllo un indirizzo di memoria univoco.
- Registri di controllo mappati nello spazio della memoria (b).

- **Vantaggi dell'I/O Mappato in Memoria:**

- **Elimina la necessità di istruzioni speciali** di I/O come IN o OUT.
- I **registri** di controllo possono essere trattati **come variabili in C**, consentendo la scrittura di driver completamente in C.
- **Protezione semplificata:** i processi utente non accedono direttamente ai registri di controllo.
  - Il sistema operativo utilizza la gestione della memoria per proteggere gli indirizzi dei registri hardware, rendendoli accessibili solo al kernel.

Spazio degli indirizzi singolo



(b)

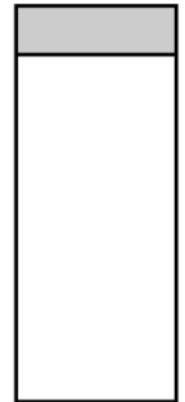


# ALTERNATIVA 2

## MEMORY-MAPPED I/O (MMIO)

- **Controllo Selettivo dei Dispositivi:** Attraverso la gestione delle pagine di memoria, è possibile dare controllo selettivo su dispositivi specifici.
  - Consente l'esecuzione di driver di dispositivi in spazi di indirizzo separati, aumentando la sicurezza e riducendo le dimensioni del kernel.
- **Attenzione a combinare MMIO e cache:**
  - **Rischio di caching dei registri di controllo:** Un registro finisce in cache, la CPU controlla solo la cache e non controlla se è stato modificato dal device
  - Caso limite: se un ciclo while è in attesa che il contenuto del registro cambi: **ciclo infinito**
  - Necessità di disabilitare selettivamente la cache per alcuni indirizzi (complessità aggiuntiva).
    - Bisogna identificare con precisione quali pagine sono dedicate ai dispositivi hardware.
    - Gli accessi a queste aree non saranno ottimizzati dalla cache, quindi possono essere più lenti.

Spazio degli indirizzi singolo



(b)

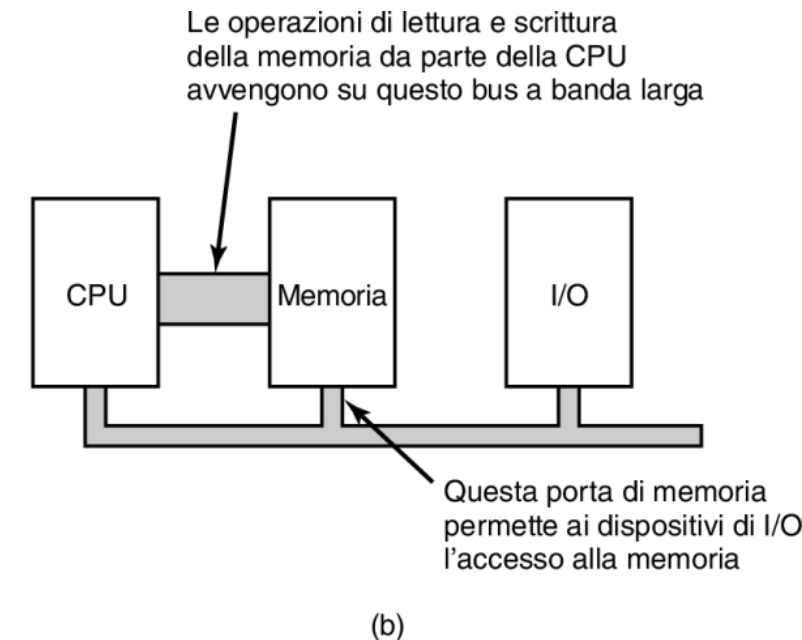
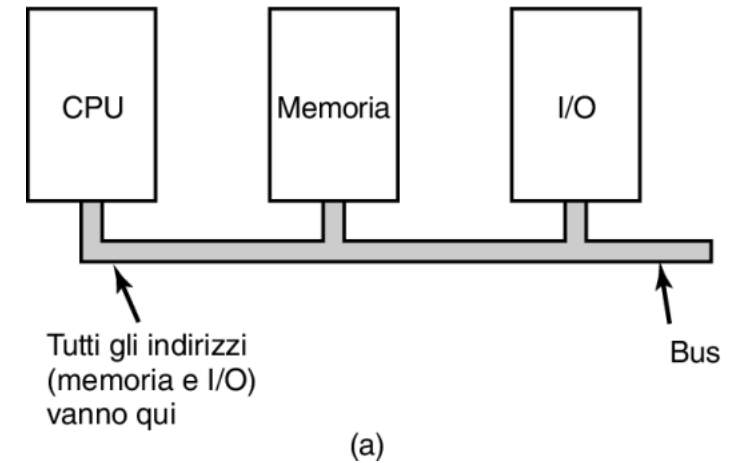




# ALTERNATIVA 2

## MEMORY-MAPPED I/O (MMIO)

- **Gestione degli Indirizzi e Architetture del Bus:**
  - **Necessità** per tutti i moduli di memoria e dispositivi di I/O di **esaminare ogni riferimento alla memoria**.
  - **Problemi con bus della memoria separati in architetture** come quelle x86 con bus multipli (memoria, PCIe, SCSI, USB).
- **Soluzioni:**
  1. **Tentativi sequenziali (Memory-first):** La richiesta è indirizzata prima alla memoria principale. Se fallisce, viene inoltrata ad altri bus.
    - **Pro:** Semplice da implementare.
    - **Contro:** Maggiore latenza per gli accessi I/O.
  2. **Bus Snooping ("Spia"):** Un dispositivo sul bus monitora gli indirizzi e reindirizza quelli destinati ai dispositivi I/O.
    - **Pro:** Accessi I/O più rapidi.
    - **Contro:** Maggiore complessità hardware.



# ALTERNATIVA 2

## MEMORY-MAPPED I/O (MMIO)

### ■ Prestazioni della Memoria:

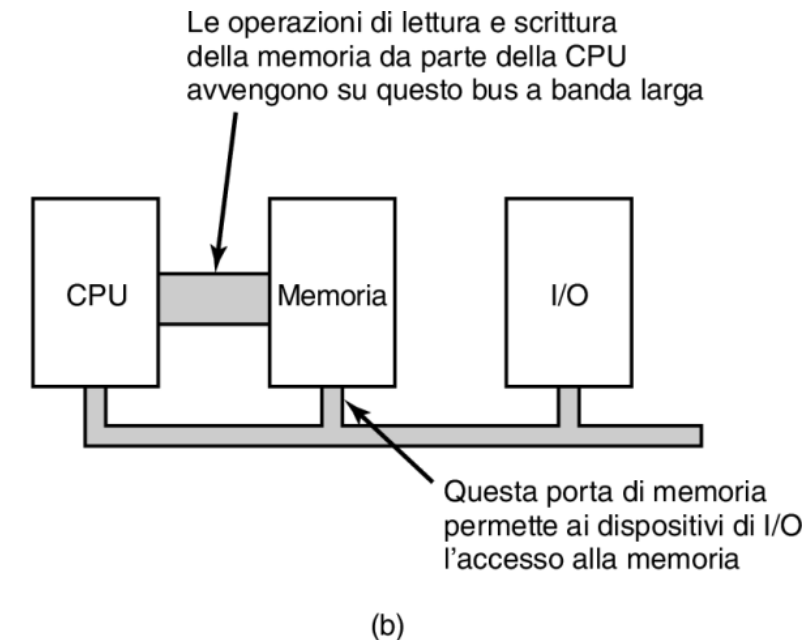
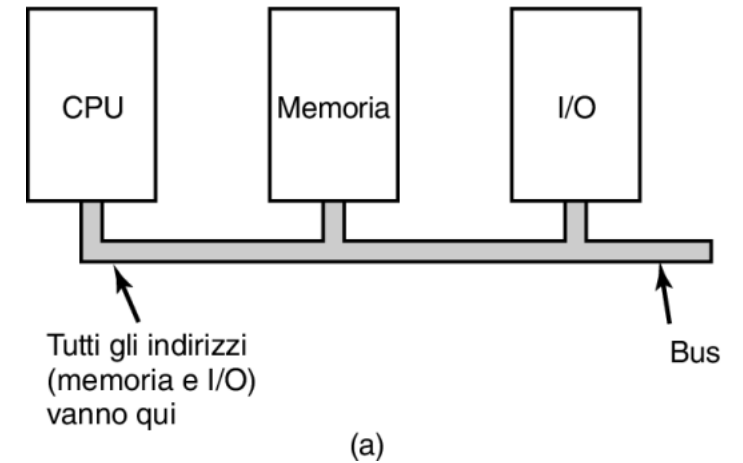
- Il bus della memoria è ottimizzato per velocità, ma deve gestire sia gli accessi alla memoria principale che ai dispositivi.
- Tentativi sequenziali preservano le prestazioni della memoria (dando priorità agli accessi alla memoria principale).

### ■ Complessità nella Gestione dell'I/O:

- Dispositivi come “bus snooping” riducono la latenza I/O, ma introducono complessità hardware e software.
- La gestione avanzata richiede coordinamento tra memoria e dispositivi su bus separati.

### ■ Conclusione: L'I/O mappato in memoria richiede un bilanciamento tra:

- **Prestazioni:** Minimizzare i ritardi negli accessi a memoria e dispositivi.
- **Complessità:** Mantenere un design efficiente senza overhead eccessivo.



# ALTERNATIVA 3

## IBRIDO (PMIO + MMIO)

- **Cos'è un approccio ibrido?**

- Combina **Port-Mapped I/O (PMIO)** e **Memory-Mapped I/O (MMIO)**:

- **PMIO**: Utilizza un indirizzamento separato per i dispositivi I/O con istruzioni dedicate (IN e OUT).
    - **MMIO**: I registri dei dispositivi sono mappati nello spazio della memoria, accessibili con normali istruzioni di memoria (LOAD, STORE).

- **Come funziona?**

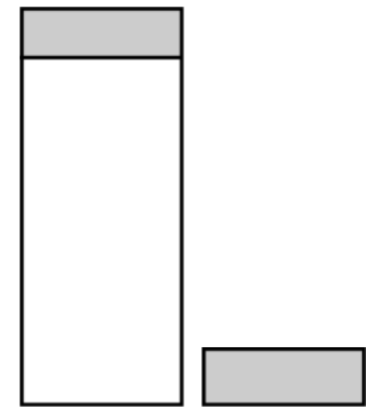
- 1. **Configurazione tramite PMIO:**

- Configurazione iniziale di dispositivi hardware (es. attivazione o setup di registri di base).
    - Usa lo spazio di indirizzi delle porte e istruzioni IN/OUT.

- 2. **Accesso ai dati tramite MMIO:**

- Operazioni ad alta velocità come trasferimenti di dati (es. schede grafiche, controller di rete).
    - I dispositivi sono mappati nello spazio di memoria principale.

Due spazi degli indirizzi



(c)



# ALTERNATIVA 3

## IBRIDO (PMIO + MMIO)

### ■ Vantaggi

1. **Flessibilità:** PMIO per configurazioni semplici, MMIO per operazioni rapide.
  - Accesso diretto tramite istruzioni di memoria, Nessun overhead dovuto al cambio di modalità, Cache e parallelismo
2. **Ottimizzazione delle prestazioni:** MMIO è più veloce per trasferimenti dati continui.
3. **Compatibilità legacy:** PMIO consente di supportare dispositivi più vecchi.
4. **Separazione logica:** Configurazione e utilizzo sono gestiti in modi distinti.

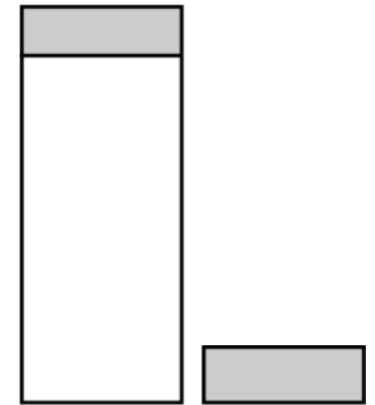
### ■ Svantaggi

1. **Aumento della complessità:** Due modalità richiedono logiche hardware e software più complesse.
2. **Overhead iniziale:** PMIO può rallentare la configurazione rispetto a un approccio puramente MMIO.
3. **Limitazioni architetturali:** Alcune CPU moderne (es. ARM) non supportano PMIO.

### ■ Esempio pratico: x86

- PMIO per configurare i dispositivi PCIe (es. usando porte 0xCF8/0xCFC).
- MMIO per accedere ai registri del dispositivo PCIe dopo la configurazione.

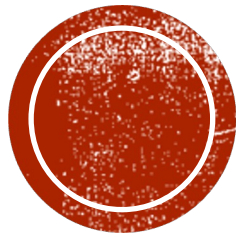
Due spazi degli indirizzi



(c)



# IN ATTESA DI RICHIESTE DI I/O...



Ora che possiamo inviare comandi ai dispositivi, ma cosa succede se l'operazione richiesta richiede tempo?

- La maggior parte dei dispositivi offre bit di stato nei propri registri per segnalare che una richiesta è stata completata (e il codice di errore).
- Il sistema operativo può interrogare questo bit di stato (**polling**).
- È una buona soluzione?



# DIRECT MEMORY ACCESS (DMA) - CONCETTI BASE E CONFIGURAZIONE

- **Definizione e Necessità del DMA:**

- DMA permette alla CPU di scambiare dati con i controller dei dispositivi bypassando il trasferimento manuale byte per byte.
- Riduce lo spreco di tempo della CPU, migliorando l'efficienza del trasferimento dati.

- **Configurazione Hardware:**

- Presenza di un controller DMA in molti sistemi, a volte integrato nei controller di dispositivi.
- Il controller DMA può gestire trasferimenti a più dispositivi, spesso situato sulla scheda madre.

- **Registri e Funzionamento del Controller DMA:**

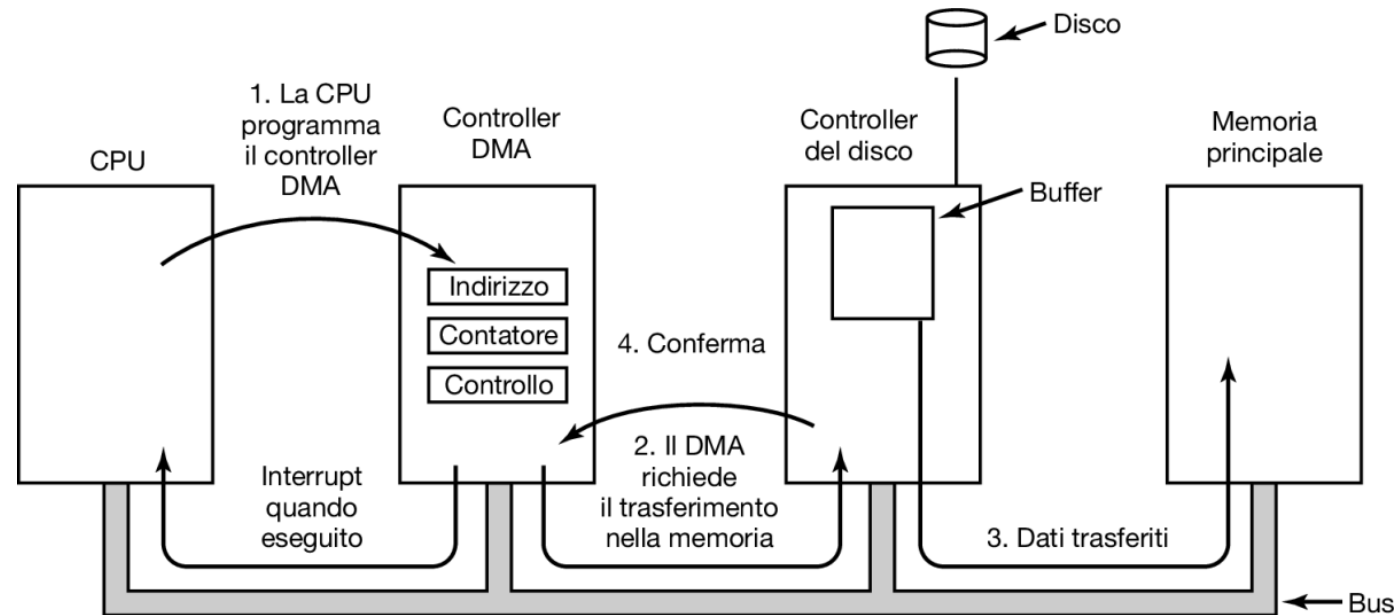
- Contiene registri per indirizzi di memoria, conteggi di byte e controlli (direzione di trasferimento, unità di trasferimento, etc.).



# DIRECT MEMORY ACCESS (DMA) - PRINCIPI BASE E FUNZIONAMENTO

## ■ Processo Tradizionale vs DMA:

- **Senza DMA:** Il controller del disco legge i dati e li memorizza nel suo buffer. Dopo aver controllato gli errori, provoca un interrupt e il SO copia i dati in memoria.
- **Con DMA:** La CPU imposta il controller DMA e invia un comando al controller del disco (**Passo 1**).

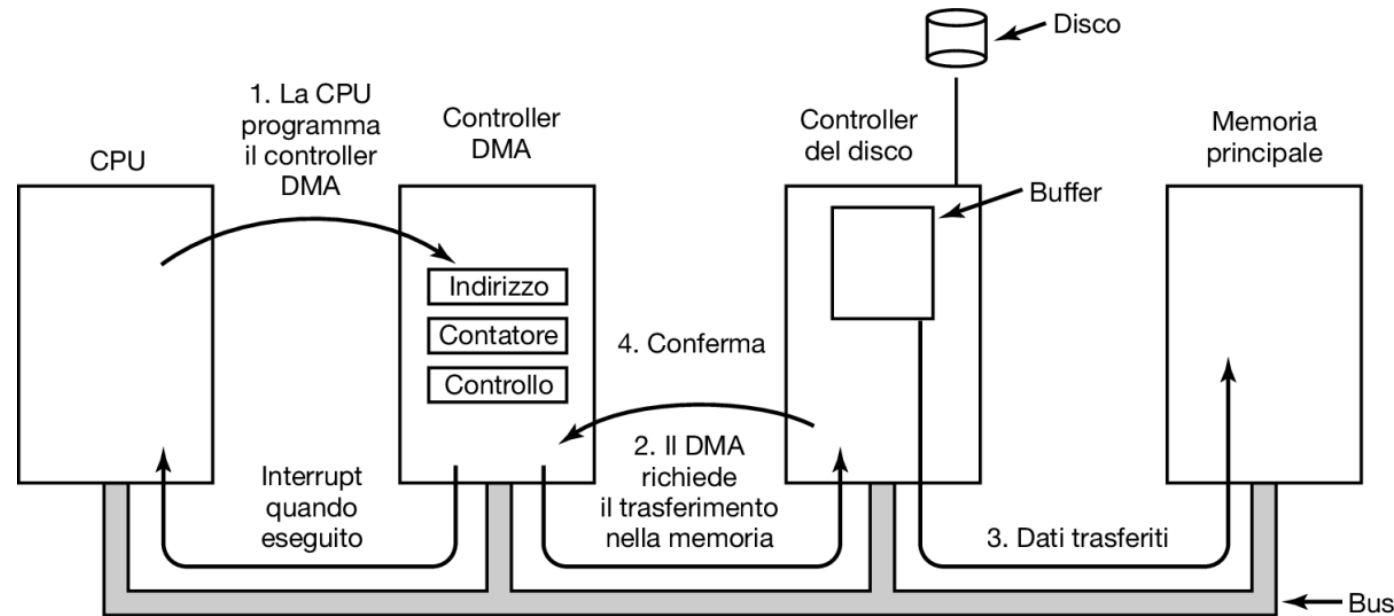




# DIRECT MEMORY ACCESS (DMA) - PRINCIPI BASE E FUNZIONAMENTO

## ■ Funzionamento Dettagliato del DMA:

- **Passo 2:** Il controller DMA richiede la lettura al controller del disco.
- **Passo 3:** Scrittura in memoria da parte del controller del disco.
- **Passo 4:** Conferma dal controller del disco al controller DMA.
- Ripetizione dei passi 2-4 fino al completamento del trasferimento.
- DMA invia un interrupt alla CPU al termine del trasferimento.



# DIRECT MEMORY ACCESS (DMA) - PANORAMICA E COMPLESSITÀ

- **Variabilità nei Controller DMA:**
  - **Range:** Da semplici (un trasferimento alla volta) a complessi (gestione di trasferimenti multipli simultanei).
- **Controller DMA Complessi:**
  - Multipli set di registri per diversi canali.
  - Ogni canale programmabile per trasferimenti specifici.
  - Capacità di gestire contemporaneamente diversi controller di dispositivi.
- **Gestione dei Trasferimenti Multipli:**
  - Selezione del dispositivo successivo post-trasferimento tramite algoritmi come round-robin o prioritari.
  - Linee di conferma separate per ogni canale DMA sul bus.
- **Considerazioni sul Non-Uso del DMA:**
  - In alcune situazioni, la CPU può essere più veloce del DMA nel gestire trasferimenti.
  - Nei computer embedded, l'eliminazione del controller DMA può essere una scelta per ridurre i costi.



# DIRECT MEMORY ACCESS (DMA) - PANORAMICA E COMPLESSITÀ (2)

- **Modalità DMA e Interazioni con il Bus:**
  - **Cycle Stealing:** DMA trasferisce una parola per volta, "rubando" cicli alla CPU.
    - Questo rallenta leggermente la CPU ma permette la condivisione del bus.
  - **Modalità Burst:** DMA ottiene controllo completo del bus, eseguendo trasferimenti multipli in una volta.
    - Questa modalità è efficiente ma può bloccare la CPU per periodi prolungati.
  - **Fly-by Mode:** DMA trasferisce dati direttamente alla memoria principale senza intermediari
    - Riducendo l'uso del bus ma richiedendo un ciclo extra per ogni trasferimento.
      - Richiede un ciclo aggiuntivo per ogni trasferimento diretto, poiché il DMA deve gestire sia il dispositivo sorgente sia la memoria di destinazione in tempo reale.
- **Gestione degli Indirizzi:** DMA utilizza tipicamente indirizzi fisici, richiedendo conversione da parte del sistema operativo.
- **Alcune periferiche (come il disco) hanno un Buffer Interno; perché?**
  - Utilizzato per verificare i dati (checksum) e gestire l'afflusso costante di bit dal disco.
  - Questo approccio semplifica il design del controller evitando problemi di temporizzazione e rischi di buffer overrun.



# INTERRUPT NEL SISTEMA OPERATIVO - CONCETTI BASE

## ■ Tipi di Interrupt:

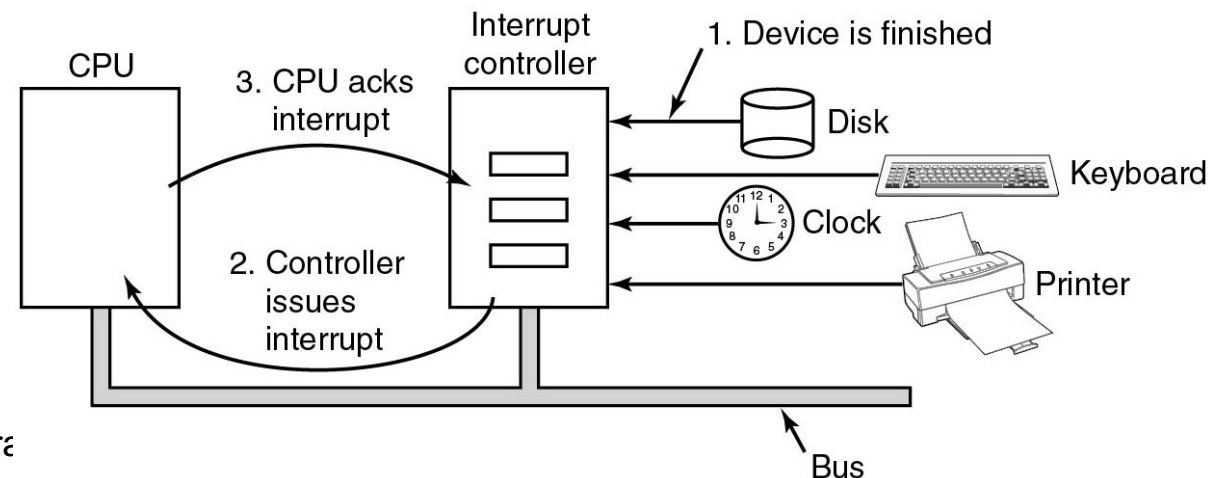
- **Trap:** Azione deliberata del codice del programma, come una chiamata di sistema.
- **Fault o Eccezione:** Azioni non deliberate, come errori di segmentazione o divisione per zero.
- **Interrupt Hardware:** Segnali inviati da dispositivi come stampanti o reti alla CPU.

## ■ Funzionamento degli Interrupt Hardware:

- Un dispositivo di I/O invia un segnale di interrupt alla CPU tramite una linea del bus assegnata.
- Gestito dal chip del controller degli interrupt sulla scheda madre.

## ■ Gestione degli Interrupt da Parte del Controller:

- Se non ci sono altri interrupt in corso, il controller gestisce immediatamente l'interrupt.
- In caso di interrupt simultanei o prioritari, il dispositivo è temporaneamente ignorato.



# PROCESSO DI GESTIONE DEGLI INTERRUPT

- **Segnalazione dell'Interrupt alla CPU:** Il controller assegna un numero alle linee degli indirizzi per specificare il dispositivo che richiede attenzione e invia un segnale di interruzione alla CPU.
- **Interruzione e Gestione da Parte della CPU:** La CPU interrompe il suo attuale compito.
  - Utilizza il numero sulle linee degli indirizzi come indice nella tabella del vettore degli interrupt per ottenere un nuovo contatore di programma.
- **Vettore degli Interrupt:** Punta all'inizio della procedura di servizio degli interrupt corrispondente.
  - Posizione fissa o variabile in memoria, con un registro della CPU che indica l'origine.
- **Conferma e Gestione degli Interrupt:** La procedura di servizio conferma l'interrupt scrivendo su una porta del controller degli interrupt.
  - Questo evita race condition tra interrupt quasi simultanei.



# SALVATAGGIO DELLO STATO E PROBLEMI RELATIVI

- **Salvataggio dello Stato:** Al minimo, il contatore di programma deve essere salvato per riavviare i processi interrotti.
  - Alcune CPU salvano tutti i registri visibili e interni.
- **Dove Salvare le Informazioni prelevate dai dispositivi di I/O:** Salvataggio nei registri interni o sullo stack.
  - **Problemi con i registri interni:** tempi morti lunghi e rischio di sovrascrittura.
- **Uso dello Stack:** Uso dello stack corrente (processo utente) o dello stack del kernel.
  - **Problemi con lo stack corrente:** puntatori non leciti, rischio di errori di pagina.
  - **Uso dello stack del kernel:** cambiamento di contesto della MMU, invalidazione della cache e del TLB - overhead





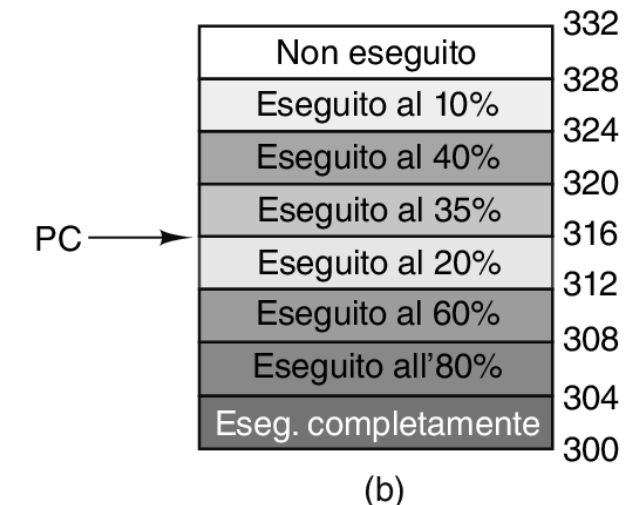
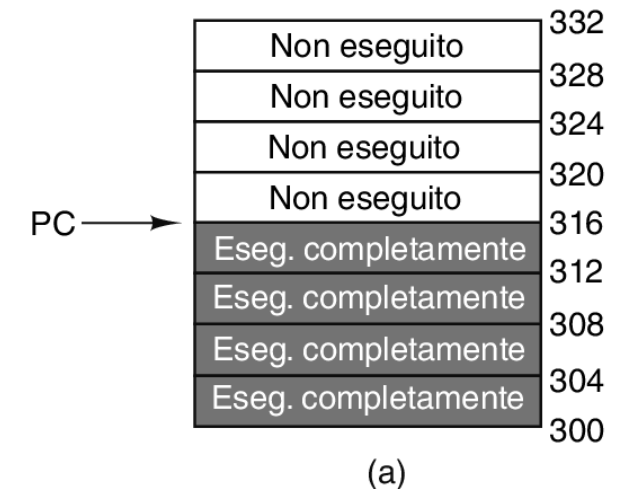
# INTERRUPT PRECISI VS IMPRECISI - CONTESTO E DIFFERENZE FONDAMENTALI

- **Il problema:**

- Le moderne CPU utilizzano architetture pipeline e superscalari, complicando la gestione degli interrupt.
- Questa complessità nasce dalla capacità delle CPU di iniziare l'esecuzione di istruzioni multiple prima del completamento delle precedenti.

- **Definizione di Interrupt Precisi e Imprecisi:**

- a) **Interrupt Precisi:** Situazione in cui il sistema può determinare con esattezza quali istruzioni sono state completate al momento dell'interrupt e quali no.
- b) **Interrupt Imprecisi:** Condizione in cui diverse istruzioni vicino al contatore di programma (PC) si trovano in vari stati di completamento al momento dell'interrupt, rendendo incerto lo stato esatto del programma.





# INTERRUPT PRECISI - DEFINIZIONE E CARATTERISTICHE

- **Caratteristiche di un Interrupt Preciso:**
  - Il Program Counter (PC) è salvato in un luogo noto.
  - Tutte le istruzioni eseguite prima del PC sono completate.
  - Nessuna istruzione dopo il PC è stata eseguita.
  - Lo stato dell'istruzione puntata dal PC è noto.
- **Gestione degli Interrupt Precisi:**
  - La CPU cancella gli effetti di eventuali istruzioni transitorie eseguite dopo il PC.
  - Questo approccio è utilizzato da architetture come x86 per garantire compatibilità e prevedibilità.



# INTERRUPT IMPRECISI - COMPLESSITÀ E IMPLICAZIONI

- **Interrupt Imprecisi:**

- Occorre quando diverse istruzioni vicino al PC sono in vari stati di completamento.
- Richiede che la CPU "vomiti" una grande quantità di stato interno sullo stack.

- **Problemi con gli Interrupt Imprecisi:**

- Rende il sistema operativo più complesso e lento.
- Il salvataggio di molte informazioni rallenta il processo di interrupt e ripristino.

- **Impatto sull'Architettura della CPU e Sicurezza:**

- Gli interrupt precisi richiedono una logica interna complessa, ma semplificano la gestione del sistema operativo.
- Gli interrupt imprecisi possono avere implicazioni di sicurezza poiché non tutti gli effetti sono completamente annullati.

