

**Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base**  
**Corso di Laurea in Ingegneria Informatica**



# **Corso di Calcolatori Elettronici I**

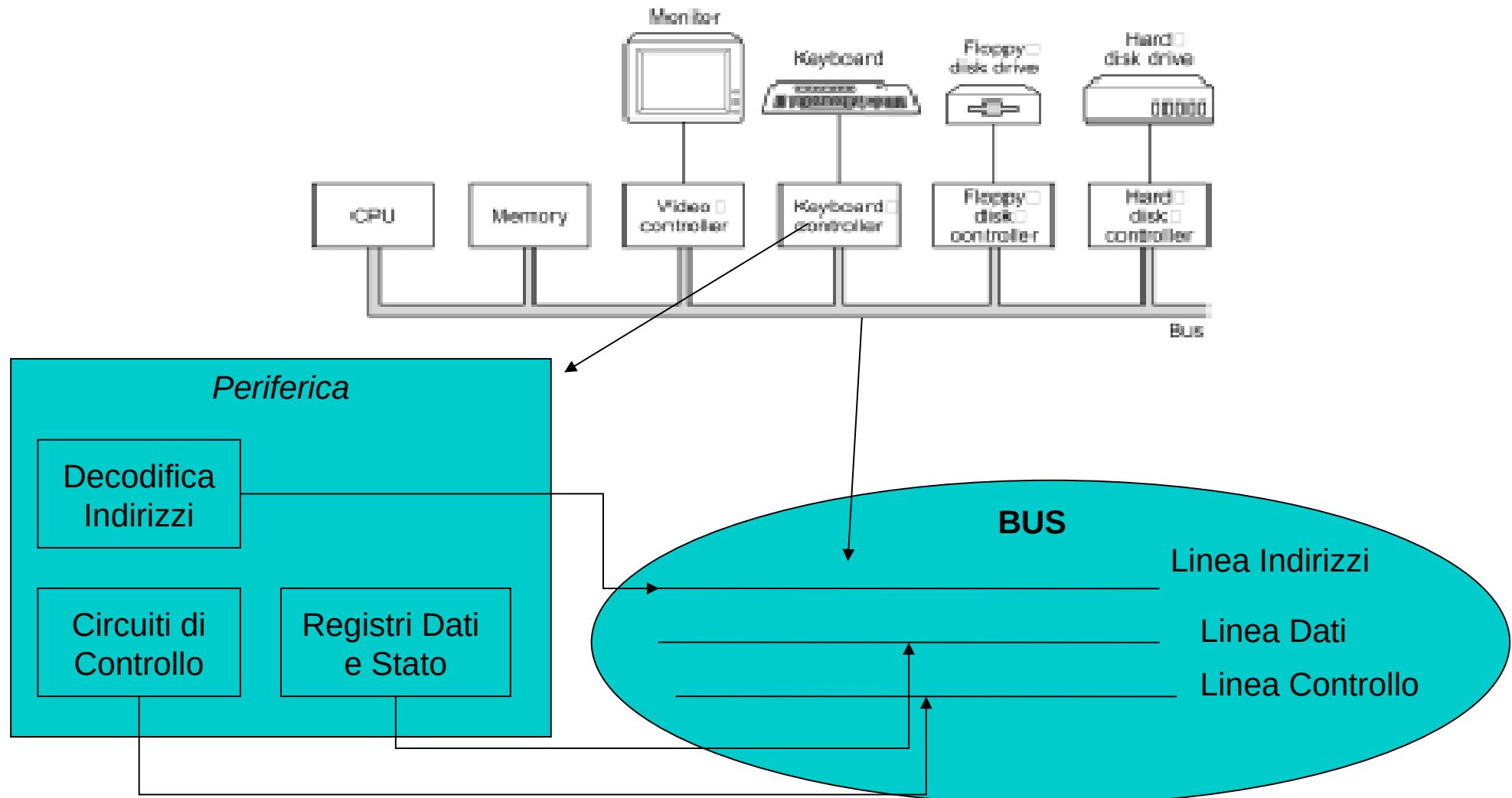
Architettura del sottosistema di Input/Output



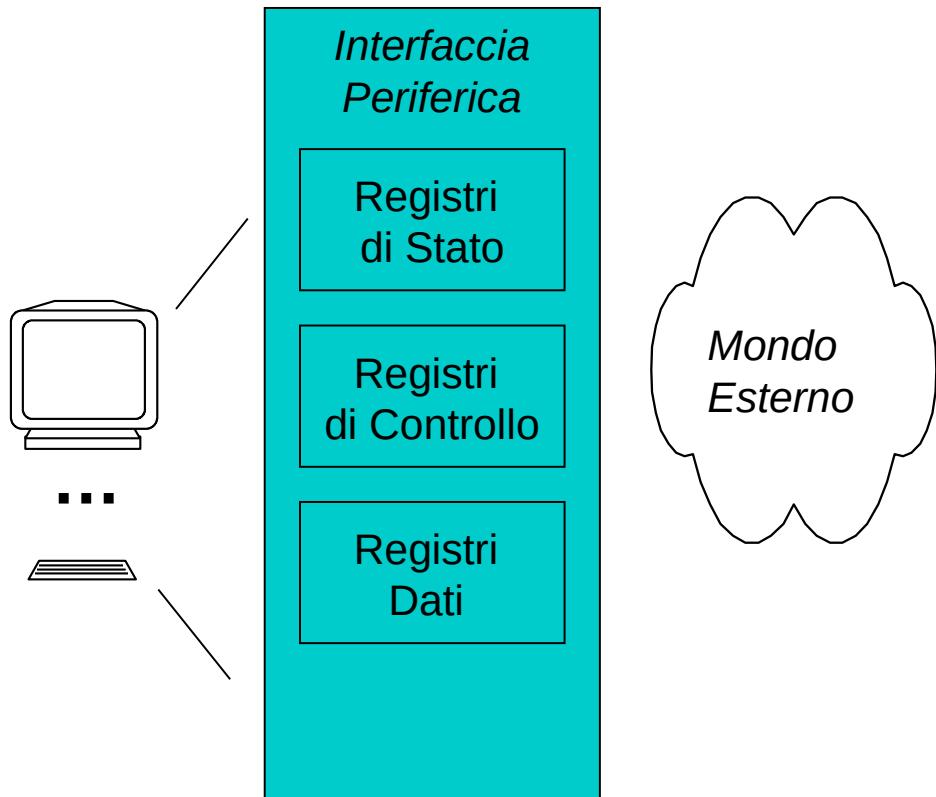
# Il Calcolatore e le periferiche



DIE  
TI.  
UNI  
NA



# Interfaccia



- E' il modo in cui la periferica si presenta all'esterno.
- In genere è una collezione di registri che permettono di
  - Osservare lo Stato
  - Modificare lo Stato
  - Fornire Dati

# Protocolli di Comunicazione



- Un Protocollo è quell'insieme di regole che gestiscono la comunicazione tra due entità

## *Protocollo Sincrono*

E' previsto un segnale di sincronizzazione (clock) che permette di gestire la temporizzazione delle comunicazioni

## *Protocollo Asincrono*

Tutta la temporizzazione della comunicazione è gestita dal protocollo stesso attraverso lo scambio dei messaggi

## *Protocollo di Ingresso*

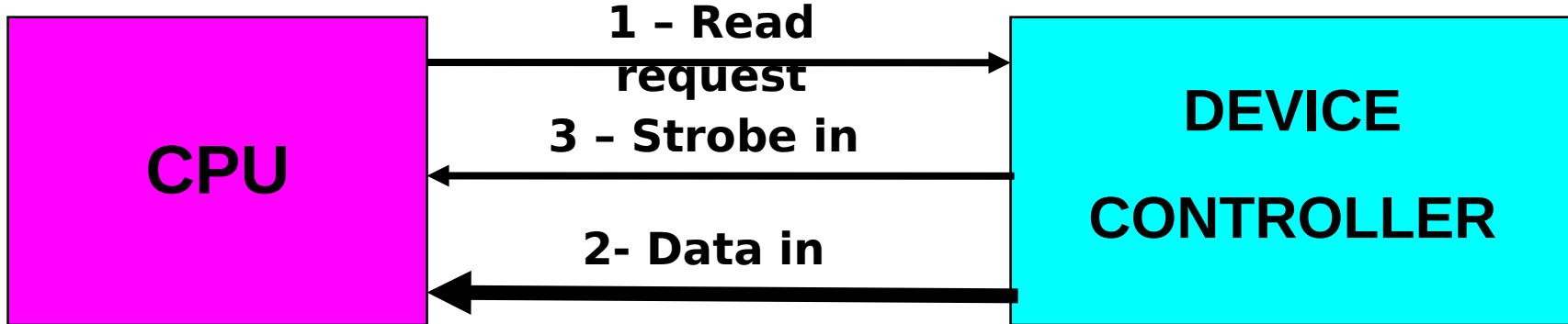
- Manda la Richiesta
- Aspetta un Ack (Strobe-In)
- Leggi il dato

## *Protocollo di Uscita*

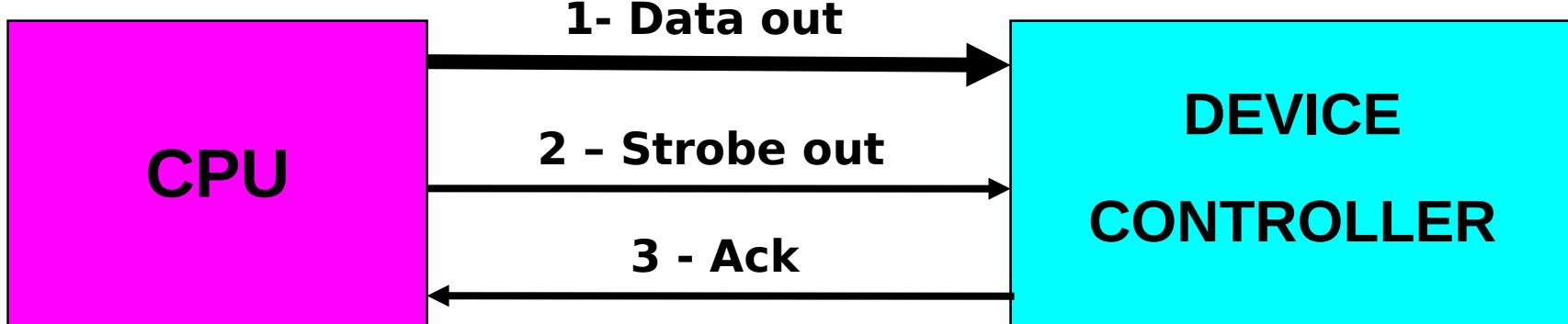
- Scrivi i dati
- Manda la Richiesta (Strobe-Out)
- Aspetta un Ack

# Handshake - Sequenza dei segnali

**Il processore legge un dato dalla periferica**



**Il processore scrive un dato verso la periferica**



## Comunicazione Parallelia

- La comunicazione avviene direttamente con una parola di un byte

## Comunicazione seriale

- La comunicazione avviene un bit per volta

# I problemi della comunicazione seriale (I)

- La periferica comunica con il processore attraverso una interfaccia.
- La comunicazione tra interfaccia e processore è sempre parallela
- La comunicazione tra interfaccia e periferica avviene mediante la porta seriale.
- E' necessario convertire la comunicazione da parallela a seriale

# I problemi della comunicazione seriale (II)

- Il processore comunica all'interfaccia il dato ma questo non viene immediatamente comunicato
- Per comunicare un nuovo dato è necessario aspettare che termini la comunicazione precedente
  - Ampio utilizzo dei buffer

# Tipi di comunicazione seriale



- Trasmissione Sincrona
  - Viene mandata una sequenza di inizio comunicazione
  - Vengono mandati i bit
  - Viene mandata una sequenza di fine comunicazione
  - La sincronizzazione avviene grazie ad un segnale di clock
- Trasmissione Asincrona
  - I bit vengono mandati uno per volta
  - Prima di ciascun byte viene mandata una particolare sequenza di sincronizzazione
  - Non viene utilizzato nessun clock esterno

# Elementi Fondamentali della Comunicazione



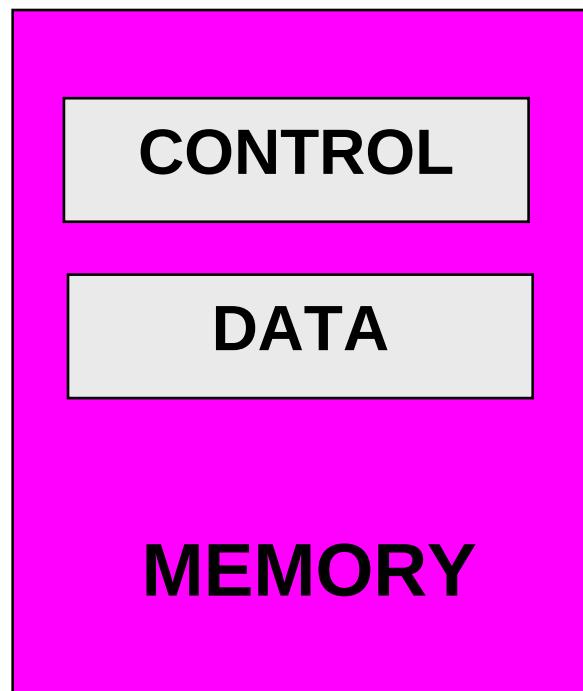
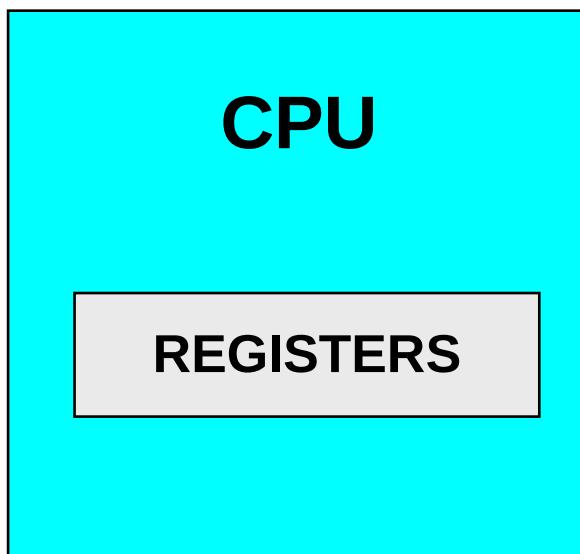
DIE  
TI.  
UNI  
NA

- Lato Periferica
  - Interfaccia della periferica
  - Protocollo di Comunicazione
- Lato Processore
  - Meccanismo di Controllo
  - Tipo di istruzioni di I/O
  - Driver

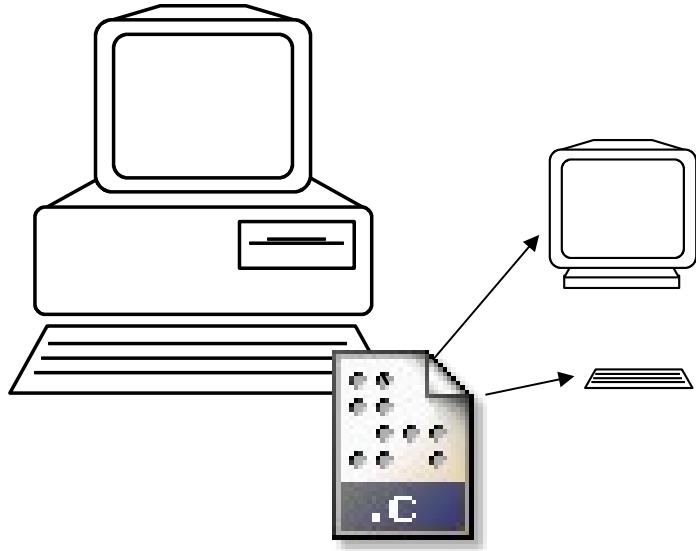
- Controllato da programma
  - Il Processore controlla lo stato della periferica in continuazione aspettando un cambiamento dello stato
- Interruzioni
  - Il Processore procede nel suo lavoro, quando la periferica cambia stato manda un segnale al processore che interrompe il lavoro corrente e procede a gestire l'evento
- Accesso Diretto in Memoria
  - La periferica è in grado di accedere alla memoria senza l'intervento del Processore

- Istruzioni di I/O
  - Istruzioni Specifiche (ad es. IN ed OUT dei processori Intel x86)
  - Indirizzamento separato per le periferiche
- Memory Mapped
  - Lo spazio di memoria visto dal processore è condiviso tra la memoria vera e propria e le periferiche
  - Le operazioni di I/O si eseguono mediante istruzioni normali, su locazioni di memoria particolari
  - È la soluzione adottata dal 68000

# Memory Mapped I/O - Schema concettuale



# Il Driver



- E' il programma che gestisce il dialogo tra il processore e la periferica
- Implementa il protocollo di comunicazione e gestisce la comunicazione
- Astrae e semplifica la comunicazione tra processore e periferica
- I/O Controllato da programma:
  - Il Driver è un programma come gli altri
- Interrupt:
  - Il Driver è tipicamente un procedura particolare (ISR)

# Esempio di Driver

```
STRT Inizializza
POLL Testa lo stato
      BEQ POLL
      Leggi Il Dato
POLL2 Testa lo stato
      BEQ POLL2
      Scrivi il Dato
      (Deve continuare?)
      BEQ POLL
```

Lo schema riprodotto mostra un esempio di comunicazione con un terminale che permette di leggere un carattere e rimetterlo a video con un meccanismo di controllo dello I/O da programma.

# I/O controllato da programma

- In questo caso la gestione dei dispositivi di I/O è totalmente demandata alla CPU
- Ogni dato viene prima trasferito dal buffer associato alla periferica ad un registro interno della CPU, e poi immagazzinato in memoria (o viceversa)
- Lo spostamento di ciascun dato implica l'esecuzione di almeno un'istruzione da parte della CPU
- Quando l'I/O programmato è basato sulla ripetizione di un test sul registro di stato per verificare quando il programma può procedere oltre, si parla di polling

# I/O controllato da programma - Output



```
DataRegister db;  
ControlBits F,C;
```

```
db.write();
```

```
C.set();
```

```
do {
```

```
}
```

```
while ( F.isReset() )
```

```
C.reset();  
F.reset();
```

IF\_CTRL



IF\_DATA



START

ORG	\$8000
MOVE.B	D0, IF_DATA
ORI.B	#\$01, IF_CTRL
BTST.B	#7, IF_CTRL
BEQ	LOOP
MOVE.B	#0, IF_CTRL

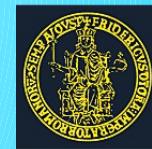
LOOP

IF\_CTRL

ORG	\$8020
DS.B	1
ORG	\$8022
DS.B	1

IF\_DATA

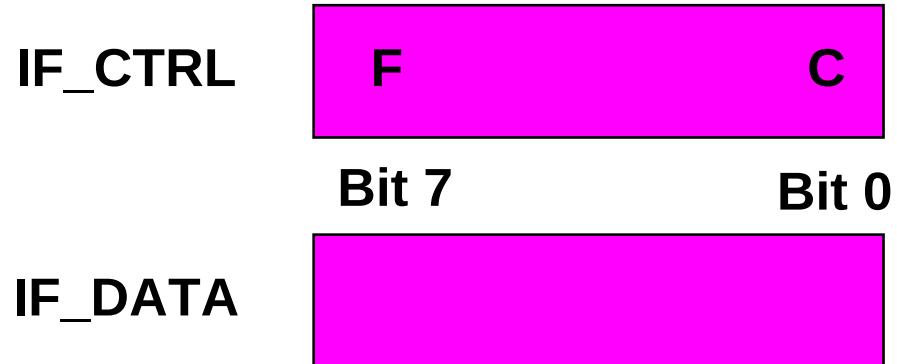
# I/O controllato da programma - Input



```
DataRegister db;  
ControlBits F,C;
```

```
C.set();  
F.reset();  
  
do {  
  
}  
  
while ( F.isReset() )
```

```
db.read();  
C.reset();
```



```
ORG        $8000  
MOVE.B    #1, IF_CTRL  
BTST.B    #7, IF_CTRL  
BEQ        LOOP  
MOVE.B    IF_DATA, D0  
AND.B    #$FE, IF_CTRL  
  
ORG        $8020  
IF_CTRL   DS.B   1  
ORG        $8022  
IF_DATA   DS.B   1
```

# Limiti dell'I/O programmato

- Ogni dispositivo deve dipendere dalla CPU per essere servito. Ne consegue che:
  - l'efficienza (in termini di uso del dispositivo) dipende dalla frequenza con cui il test viene ripetuto
  - tutti i dati devono passare attraverso la CPU, e non esiste connessione diretta tra dispositivo e memoria
  - la CPU dedica una parte del suo tempo ad eseguire banali operazioni di test e trasferimento dati.

# I/O sincronizzato con interrupt

- È basato su un segnale asincrono che il dispositivo invia alla CPU quando ha bisogno di un servizio
- In questo modo:
  - il tempo per ottenere l'attenzione della CPU si riduce
  - la CPU non perde tempo a scandire in polling il dispositivo per testarne lo stato

- La gestione del servizio dei dispositivi di I/O tramite *interrupt* permette una migliore efficienza dell'esecuzione della elaborazione della CPU
- Un input esterno *asincrono* informa il microprocessore che un dispositivo esterno richiede di essere servito. La CPU interrompe l'esecuzione del programma corrente e salta all'esecuzione di una *procedura di servizio* dell'interruzione (*Interrupt Service Routine - ISR*)
- Il dispositivo di controllo dell'interrupt funziona come un gestore delle richieste di interruzione tra i dispositivi periferici e la CPU

# Direct Memory Access (DMA)



- È il metodo preferito quando si devono trasferire grosse moli di dati.
- Una circuiteria apposita (DMA Controller) provvede ad eseguire il trasferimento di dati da una periferica alla memoria (o viceversa).
- Il DMA Controller deve essere in grado di negoziare con la CPU l'acquisizione del controllo del bus ed il suo rilascio.

# DMA: modi di funzionamento



- Il trasferimento dei dati in DMA può avvenire in vari modi:
  - trasferimento a blocchi (*burst transfer*)
  - trasferimento con *cycle stealing*
  - trasferimento in *transparent DMA*

# Trasferimento a blocchi

- Prevede che il DMA Controller, una volta acquisito il controllo del bus, lo mantenga per tutto il tempo richiesto per trasferire un blocco di dati
- In tal modo il trasferimento avviene alla massima velocità ma la CPU è bloccata per tutta la durata del trasferimento
- Il trasferimento a blocchi è importante per periferiche quali i dischi magnetici, dove il trasferimento del blocco non può essere interrotto



# Trasferimento con *Cycle Stealing*

- Il DMA Controller trasferisce i dati in piccoli blocchi, occupando il bus per periodi limitati di tempo
- In tal modo:
  - la velocità di trasferimento è minore
- ma
  - la CPU non è bloccata per periodi troppo lunghi

# Trasferimento in *Transparent DMA*

- Il DMA Controller è in grado di rilevare quando la CPU non utilizza il bus, e solo in quei periodi esegue il trasferimento dei dati
- In tal modo la CPU non è praticamente rallentata dal DMA Controller