# Corso
# Sistemi Operativi

**Prof. Pierfrancesco Bellini**
pierfrancesco.bellini@unifi.it
Laboratorio DISIT
Dip. Ingegneria dell'Informazione
Via S. Marta, 3

## Programma A.A. 2019/20

- **Introduzione**
  - Struttura di un calcolatore, gestione I/O ed interruzioni
  - Introduzione ai sistemi operativi
  - Struttura dei sistemi operativi
- **Le basi del linguaggio Java**
- **Gestione dei processi**
  - I processi
  - I thread
  - Scheduling della CPU
  - Sincronizzazione tra processi
  - Gestione dello stallo
- **Gestione della memoria**
  - Gestione della memoria centrale
  - Memoria virtuale

# Libri di testo principali

*A. Silberschaz, P. Galvin, G. Gagne,*
***Sistemi Operativi: Concetti ed esempi*,**
decima edizione, Pearson Addison-Wesley.

*G. Bucci,*
**Calcolatori elettronici, Architettura e organizzazione**
2017, McGraw-Hill

*K. Arnold, J. Gosling, D. Holmes*
***Java manuale ufficiale*,**
Pearson Addison-Wesley

Altro materiale su:

http://www.disit.org/sistemi-operativi



# Esame

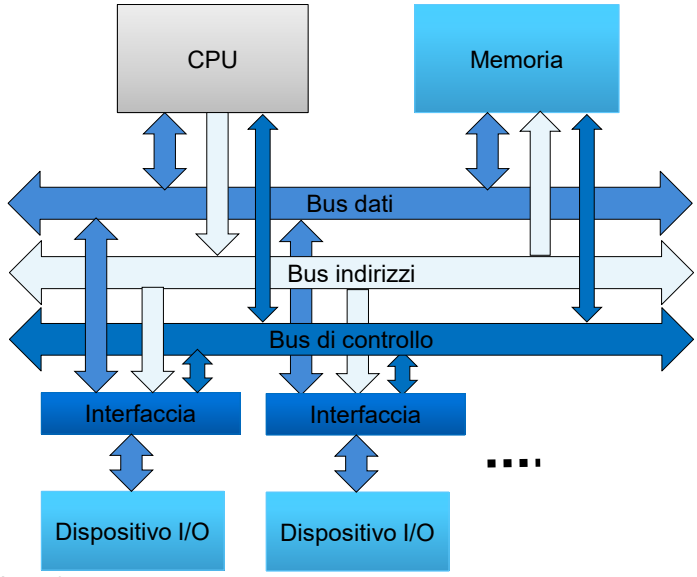
- Corso da 6 crediti
- Corso integrato con Calcolatori elettronici, per poter verbalizzare l'esame il modulo di calcolatori deve essere stato sostenuto
- Esame **scritto** e **orale**
- Due prove in itinere sostituiscono lo scritto (voto>=16 e media >=18)
- Non è previsto il salto appello
- Uno scritto sufficiente è valido per i due appelli successivi
- All'orale lo studente deve portare implementazione funzionante degli esercizi della prova scritta.

# Gestione Input/Output e Interruzioni

## Architettura del calcolatore

- Componenti del calcolatore
  - **CPU**
  - **Memoria**
  - **Periferiche I/O**
    - Tastiera, schermo, dischi rigidi, USB, etc
  - **Bus** (Dati, Indirizzi, Controllo)

# Architettura calcolatore






# Interfaccia dispositivo

- Intermediario tra dispositivo e bus
- Dispositivi e CPU procedono in modo asincrono ognuno alla propria velocità.
- L'interfaccia deve fornire:
  - registri di appoggio per i dati da inviare/ricevere
  - registri per i comandi alla periferica
  - tenere traccia dello stato dela periferica ed eventuali errori
- L'interfaccia espone dei registri (o pseudo-registri)
  - **DREG** per lo scambio dei dati
  - **CREG** per i comandi alla periferica
  - **SREG** per leggere lo stato della periferica

# Input/output

- La CPU per accedere ai registri delle interfacce può usare
  - I/O mappato in memoria
  - I/O isolato
  - entrambi
- **I/O mappato in memoria:**
  - una parte della memoria riservata per la comunicazione con i dispositivi
  - es. memoria video
- **I/O isolato**
  - istruzioni specifiche usate per interazione con dispositivi tramite porte di input/output
  - porte identificate da un numero (indirizzo)
  - **spazio di indirizzamento distinto da quello della memoria**

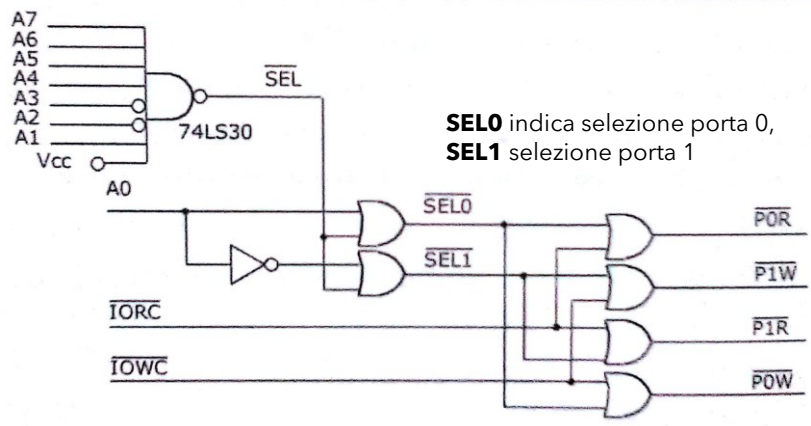


# I/O isolato

- **Istruzioni dedicate a input/output**
- la cui esecuzione hanno cicli di bus del tutto analoghi a quelli di accesso alla memoria solo che vengono usate due linee di comando specifiche
  - IORC (I/O Read Command)
  - IOWC (I/O Write Command)
- Processore x86
  - **IN AL, *PORT***
    - presenta ***PORT*** sul bus indirizzi
    - asserisce **IORC**
    - legge dal bus dati e mette in **AL**
  - **OUT *PORT*, AL**
    - presenta ***PORT*** sul bus indirizzi
    - presenta sul bus dati il contenuto di **AL**
    - asserisce il comando **IOWC**

# Decodifica indirizzi

Fa parte dell'interfaccia, identifica se viene richiesto uso di una porta dell'interfaccia, e se richiesta in lettura o scrittura




**SEL0** indica selezione porta 0,
**SEL1** selezione porta 1

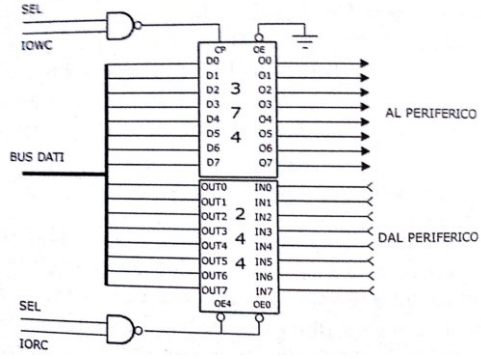
IN AL, F2H
OUT F3H,AL

queste istruzioni quale porta attivano?



# Porte dell'interfaccia

- **porta di ingresso (dalla periferica)**
  - buffer con uscita in terzo stato, uscita abilitata quando asserito IORC (istruzione IN)
- **porta di uscita (verso la periferica)**
  - latch per memorizzare il dato, acquisizione effettuata quando viene asserito IOWC (istruzione OUT)



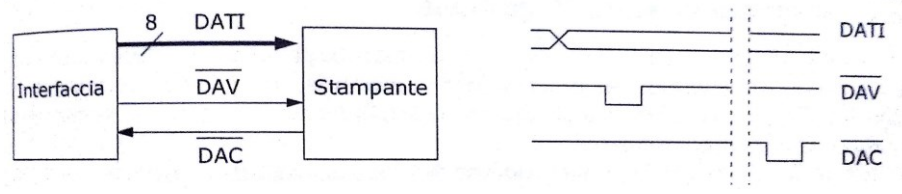
## Modalità esecuzione I/O

- nessuna sincronizzazione tra programma in esecuzione e periferiche (molto più lente)
- la CPU deve aspettare…
- le tecniche fondamentali per la gestione delle periferiche sono:
  I. gestione a controllo di programma
  II. sotto controllo di interruzione
  III. tramite accesso diretto alla memoria o con processori I/O

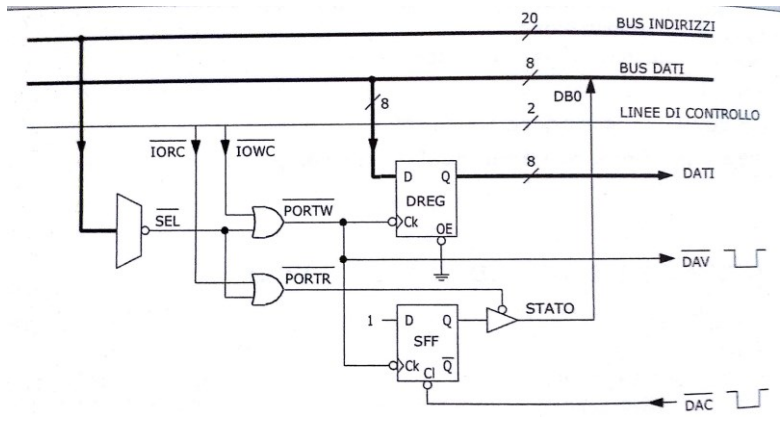


## Controllo di programma

- Consideriamo una **interfaccia di uscita** verso una stampante
- è necessario un protocollo di *hand-shacking* con la stampante
  - DAV (Data AVailable) indica che il dato è disponibile
  - DAC (Data ACkowledge) indica che il dato è stato acquisito ed un nuovo dato può essere inviato.



# interfaccia di uscita




**SEL** attivo quando sul bus indirizzi c'è E1



# Sottoprogramma di gestione

- sottoprogramma STAMPA che si aspetta:
  - nel registro SI l'offset del buffer da stampare
  - nel registro CX il numero di byte da stampare
- uso:

```
        MOV     SI, <offset BUFFER>
        MOV     CX, <n>
        CALL    STAMPA
```

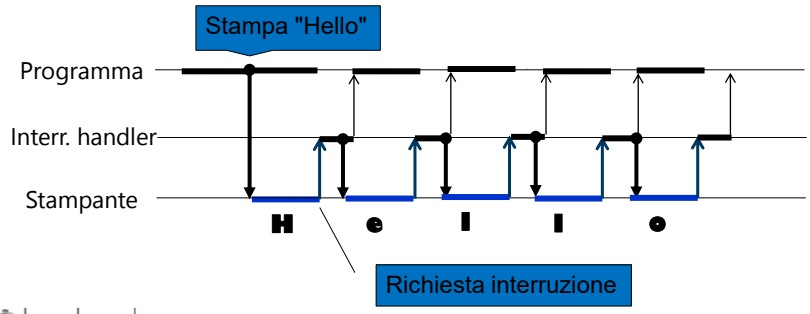
- implementazione:

```
STAMPA: MOV     AL, [SI]
        OUT     E1h, AL
ATTESA: IN      AL, E1h
        AND     AL, 1
        JNZ     ATTESA
        INC     SI
        LOOP    STAMPA
        RET
```

CPU attende il DAC=0 dalla stampante
**Tempo CPU sprecato!**

# Gestione sotto controllo di interruzione

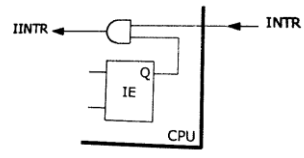
- Durante l'attesa del DAC la CPU è bloccata, in realtà potrebbe fare altro ed essere interrotta quando il DAC viene asserito dalla stampante.
- All'interruzione viene eseguita una routine di servizio (interrupt handler) che invierà l'eventuale nuovo dato alla stampante e la CPU riprenderà esecuzione da dove era stata interrotta.
- L'esecuzione della routine avviene tra l'esecuzione di due istruzioni



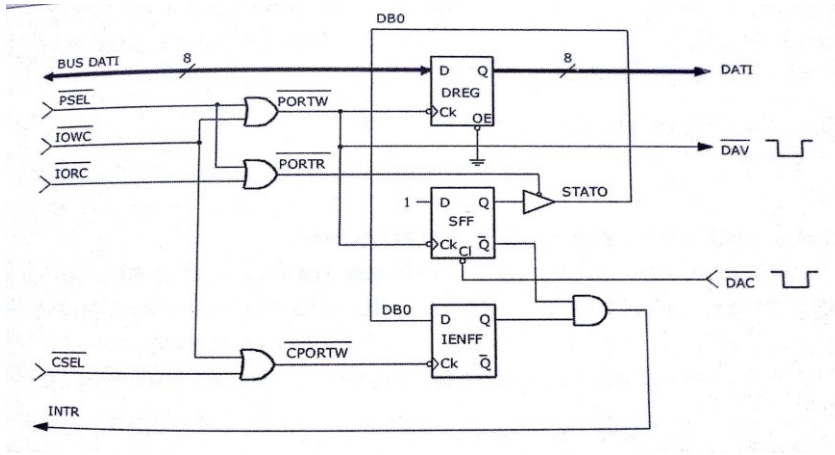



# Modello semplificato sistema interruzione

- Si ipotizza che il sistema piloti una sola periferica
- la linea INTR in ingresso alla CPU indica una richiesta di interruzione
- esiste un flag nella CPU che indica abilitazione delle interruzioni IE (Interrupt Enable)
- Quando viene ricevuta una interruzione IINTR la CPU deve azzerare IE (in modo che non possa essere interrotta) e fare fetch dell'istruzione all'indirizzo 0 (senza modificare PC)
- L'istruzione all'indirizzo 0 dovrà essere una CALL alla procedura di gestione interrupt
- La procedura terminarà con istruzione IRET che abiliterà anche interruzioni



# Interfaccia modificata




- aggiunto IENFF tipo D, registro accessibile in scrittura per attivare/disattivare generazione interrupt (usando una porta specifica)
- DAV asserito ➔ INTR=0
- DAC asserito ➔ INTR=1 (se IENFF=1)



# Routine di servizio

- L'esecuzione della routine di servizio dell'interrupt deve essere trasparente rispetto al programma interrotto
- alla posizione 0 viene inserita istruzione CALL alla routine gestione interruzione
- la routine deve:
  1. salvare sullo stack PSW (parola di stato) e tutti i registri usati
  2. trasferire il prossimo dato
  3. disasserire la richiesta di interruzione
  4. ripristinare registri
  5. ritornare al programma interrotto
- per il punto 5 c'è istruzione IRET che è come RET solo che abilita le interruzioni

# Routine di servizio

- Sezione di inizializzazione:

```
STAMPA:   MOV    BUSY,1
          MOV    IND, SI
          MOV    COUNT, CX
          MOV    AL, [SI]
          OUT    PORT, AL
          MOV    AL, 1
          OUT    CPORT, AL
          RET
```



# Routine di servizio

- Routine di servizio:

```
INTSTAMP: PUSH   PSW
          PUSH   AX
          PUSH   CX
          PUSH   SI
          MOV    SI, IND
          INC    SI
          MOV    CX, COUNT
          DEC    CX
          JZ     FINE
          MOV    AL, [SI]
          OUT    PORT, AL
          MOV    IND, SI
          MOV    COUNT, CX
```

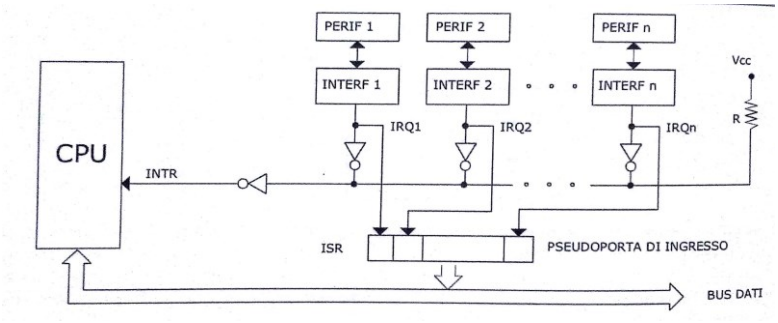
```
ESCI:     POP    SI
          POP    CX
          POP    AX
          POP    PSW
          IRET
FINE:     MOV    BUSY,0
          MOV    AL,0
          OUT    CPORT,AL
          JMP    ESCI
```

# Interruzione da parte di più periferiche

- Si hanno i seguenti problemi:
  - riconoscere la periferica dal quale arriva l'interruzione
  - scegliere quale è la routine di servizio da eseguire
  - gestione priorità, si possono avere richieste contemporanee, si deve stabilire quale ha priorità maggiore
  - gestire l'interrompibilità della routine di servizio da parte di periferica a priorità maggiore



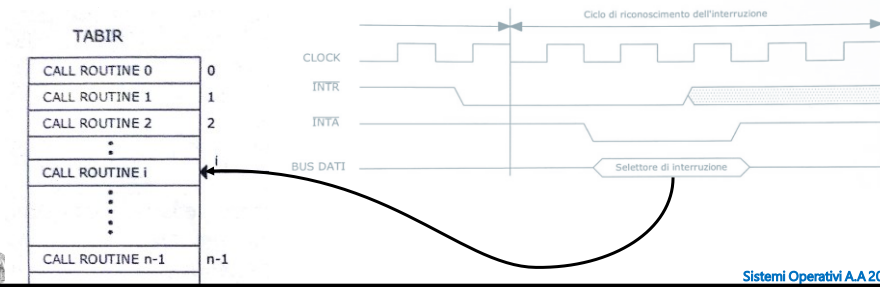
# Discriminazione da programma




- Ogni periferica indica la sua eventuale richiesta in un bit del registro ISR (Interruprt Service Request)
- Leggendo la porta ISR la routine di servizio stabilisce quale periferica ha generato interruzione
- L'ordine di esame dei bit indica la priorità
- è necessario che la richiesta venga disasserita prima della fine della gestione altrimenti genera una nuova interruzione
- metodo inefficiente per le istruzioni in più per determinare periferica

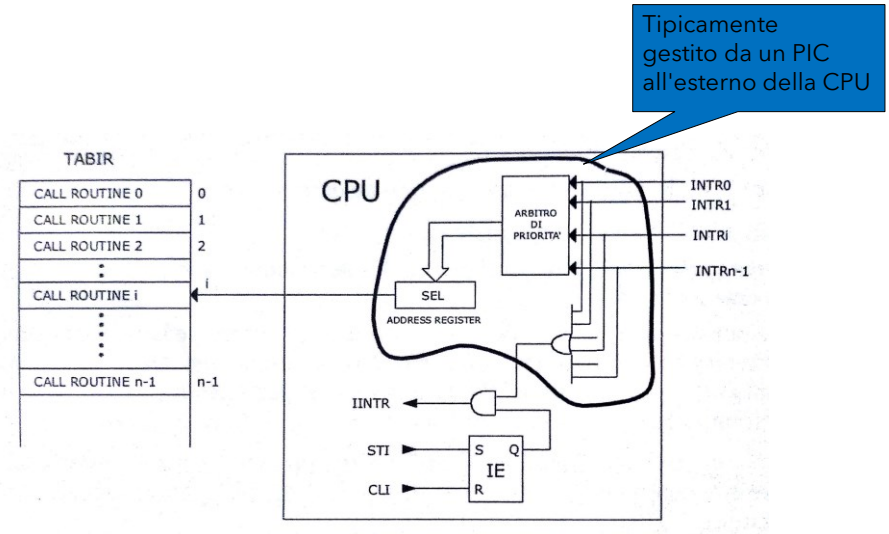
# Interruzioni vettorizzate

- inefficienza superata:
  - il dispositivo indica il numero di interrupt (IRQ i)
  - la CPU esegue direttamente la routine associata usando tabella TABIR (contenente le procedure di gestione)
  - Il numero della interrupt viene acquisito tramite un ciclo di bus
    - il segnale INTA indica che la CPU ha ricevuto interruzione e attende su bus dati il numero IRQ

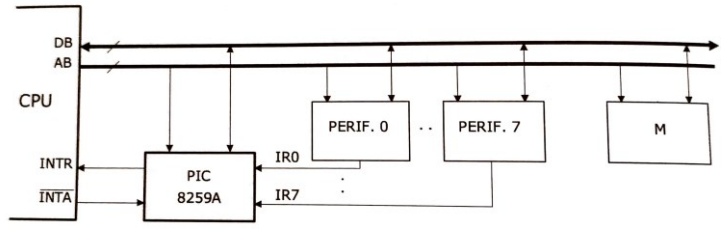





# Interruzioni vettorizzate



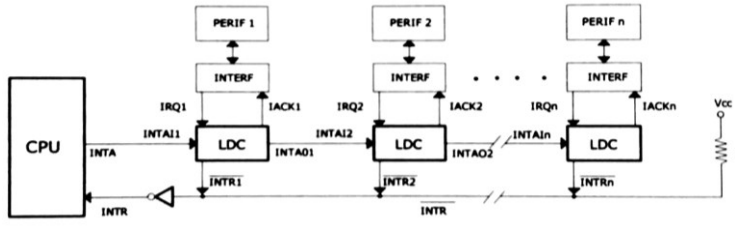
# Interruzioni vettorizzate con PIC

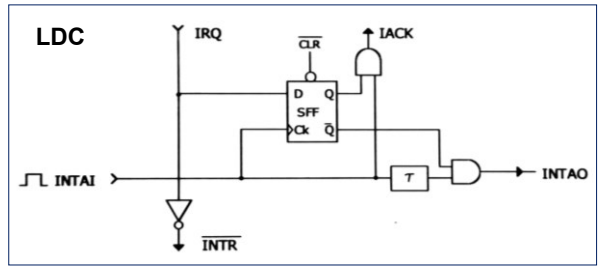



- PIC = Programmable Interrupt Controller
- Priorità programmabile
- Usa due porte di I/O (nei PC 20h/21h e A0h/A1h), una per la programmazione e una per il mascheramento delle interruzioni
- E se ci si vogliono gestire più di 8 periferiche?
- Si usano due PIC in cascata per gestire 15 periferiche (master + slave)



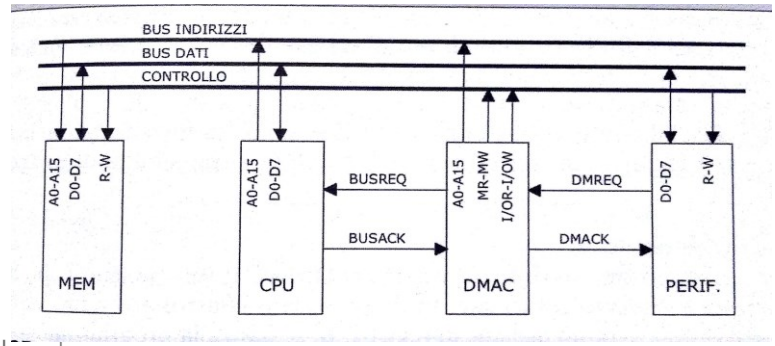
# Interruzioni con daisy-chain



# Accesso diretto alla memoria

- Per dispositivi che leggono/scrivono tanti dati velocemente (es. dischi rigidi, schede di rete) trasferire tutti i singoli byte usando le interruzioni sarebbe molto inefficiente
- La CPU può delegare questa attività al **DMAC** (Direct Memory Access Controller)

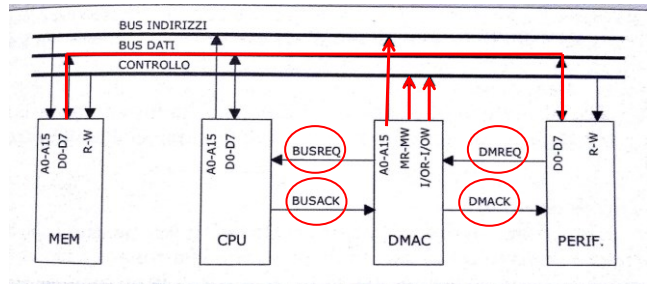





# DMA

**Operazioni**

1. Interfaccia richiede servizio DMA (DMREQ)
2. DMAC richiede alla CPU uso del bus (BUSREQ)
3. CPU concede bus al DMAC (BUSACK) fintanto che BUSREQ è asserito
4. DMAC mette indirizzo su bus indirizzi, attiva MR e IOW o MW e IOR
5. La periferica scrive/legge su bus dati il dato da trasferire
6. Finito il trasferimento DMAC disattiva BUSREQ e CPU acquisisce nuovamente BUS e disattiva BUSACK



## Struttura DMA

- L'architettura del DMA prevede:
  - un contatore del numero di caratteri/parole da trasferire
  - un puntatore alla posizione dove andrà scritto/letto il dato in memoria
  - un registro di comando con il tipo di trasferimento
  - un eventuale registro di stato
- Fase di programmazione
  - visto come dispositivo dotato di un insieme di registri
- Trasferimento dati con due modalità
  - **trasferimento singolo**, trasferisce un singolo carattere
  - **trasferimento a blocchi**, trasferisce tutte le parole indicate dal contatore, incrementando o decrementando la posizione, occupa il BUS fino alla terminazione del trasferimento, **al termine del trasferimento viene generata interruzione**



## Interruzioni

- **asincrone**: generate dai dispositivi I/O
- **sincrone**: generate dall'esecuzione delle istruzioni
  - **dovute ad errori** che avvengono nell'esecuzione di una istruzione (dette in questo caso anche eccezioni)
  - **dovute ad una chiamata esplicita** con una istruzione specifica es. "INT 21h", simile a chiamata a procedura, chiama la procedura associata a una posizione della tabella delle interruzioni