Abbiamo visto:

- sistema monolitico (I + E + O)
- dividiamo I e O da E, e affidiamo loro una CPU
- replichiamo gli I e gli O per parallelizzare singolarmente gli I e O



## Indice lezione:

- Parallelizzazione tra Input e Output
  - Controllo e busy wait
- Parallelizzazione tra I/O e E
  - Interruzioni (da dispositivo e polling)
- Veri vantaggi solo con la multiprogrammazione
  - Interruzioni in multiprogrammazione



## Richiami e notazioni:

- Un sistema di calcolo è costituito da una o più CPU, memoria principale, memoria secondaria, dispositivi di I/O

- Memoria: un vettore M[0:n-1] (es. n=32MB) a cui la CPU accede tramite le istruzioni "LOAD N1" e "STORE N1"

- CPU: la CPU contiene una serie di registri che referenzia per nome o in maniera implicita.
  Due registri speciali sono: IR (instruction register) e PC (program counter)



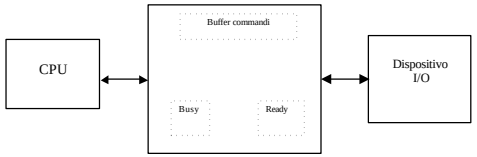
- Ciclo di Fetch-Execute:
  La CPU ripete continuamente, *in hardware*, il ciclo:

```
repeat
    IR := M[PC]
    PC := PC + 1
    execute(istruz. in IR)
until  CPU halt
```



## Un modello di interazione tra CPU e dispositivi di I/O

Descrizione del problema e della necessità di un sistema operativo efficiente




*Processore del dispositivo*



- La CPU inserisce un comando nel buffer ed attiva il processore del dispositivo mettendo ad 1 il flag BUSY

- Il processore del dispositivo, se non sta eseguendo un comando, ispeziona continuamente il flag BUSY

- Non appena lo trova con il valore ad 1, lo azzera ed inizia l'esecuzione del comando contenuto nel buffer

- Al termine dell'esecuzione il flag READY viene messo a 1 per avvertire la CPU che il comando è stato eseguito

- La CPU azzera il flag READY e inserisce un nuovo comando nel buffer

## Interazione tra CPU e dispositivi di I/O

- Nel sistema operativo considerato la CPU non esegue *nessun altro lavoro* durante l'attesa per il completamento di un comando.



- Il modello di comunicazione può essere espresso più precisamente come segue:

| **CPU** | **I/O Processor** |
|---|---|
| *repeat* | *repeat* |
| invio comando; | *repeat* esamina BUSY |
| BUSY := 1; | *until* BUSY = 1 |
| *repeat* esamina READY | BUSY := 0; |
| *until* READY = 1 | <esegui comando> |
| READY := 0; | READY := 1; |
| *until* transmission_completed | *until* device_halt |



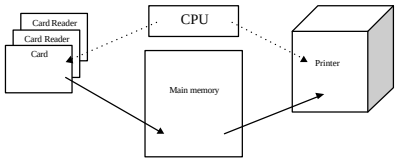
## Interazione tra CPU e dispositivi di I/O

- Generalmente il tempo dedicato all'I/O costituisce una parte fondamentale del *tempo complessivo di esecuzione di un programma* (cioè del tempo che intercorre tra la lettura del programma e la stampa degli ultimi risultati).

- Le prestazioni di un sistema di calcolo possono essere notevolmente migliorate riducendo il tempo dedicato alle attività di I/O.



## Esempio di un semplice S.O. (1)




Si analizza ora un esempio di sistema operativo che presenta una maggiore efficienza



## Esempio di un semplice S.O. (2)

- Sequenza di operazioni:

*repeat*
leggi il pacco di schede
compila
carica
esegui
stampa i risultati
*until* il calcolatore si ferma



## Esempio di un semplice S.O. (3)

- Compito del S.O. e` controllare la sequenza di passi e *gestire i dispositivi di I/O* (lettore di schede e stampante).

- Trasforma il calcolatore in una *macchina virtuale* capace di compilare ed eseguire una sequenza di programmi.

CR e LP possono essere in ogni istante in uno dei tre *stati*:
LIBERO, PRONTO, OCCUPATO

- Lo stato di CR dipende da:
  interruttore, comando, pacco di schede (deck)
- Lo stato di LP dipende da:
  interruttore, comando

| LP | start | com | stato |
|---|---|---|---|
| 0 | 0 | 0 | LIBERO |
| 1 | 0 | 1 | LIBERO |
| 2 | 1 | 0 | PRONTO |
| 3 | 1 | 1 | OCCUP. |

| CR | start | deck | com | stato |
|---|---|---|---|---|
| 0...3 | 0 | 0,1 | 0,1 | LIBERO |
| 4 | 1 | 0 | 0 | LIBERO |
| 5 | 1 | 0 | 1 | LIBERO |
| 6 | 1 | 1 | 0 | PRONTO |
| 7 | 1 | 1 | 1 | OCCUP. |



## Programmi di controllo I/O

a) Lettura di un pacco di schede

INDCR = indirizzo dell'area di memoria riservata, destinata a contenere le informazioni sulle schede
SL = contatore del numero di schede lette
IC2 = indirizzo corrente dell'area di memoria



programma di controllo CR:

```
fissa INDCR
SL := 0
IC2 := INDCR
attendi fino a che CR diventa PRONTO
repeat
   invia comando (IC2)   /* leggi scheda */
   SL := SL + 1
   calcola il nuovo indirizzo IC2
   attendi mentre CR e` OCCUPATO
until  CR diventa LIBERO
```



## Programmi di controllo I/O

b) Stampa di linee

INDLP = indirizzo di inizio dell'area di memoria contenente le linee da stampare
LS = contatore linee da stampare
IC1 = indirizzo corrente area di memoria



programma di controllo LP:

```
IC1 := INDLP
repeat
   attendi fino a che LP PRONTA
   invia comando(IC1)    /* stampa una linea */
   LS := LS - 1
   calcola nuovo indirizzo IC1
until  LS = 0
```



## Programmi di controllo I/O

Osservazioni:

- In entrambi i programmi compaiono delle *fasi di attesa* che dipendono dalla *diversa velocità della CPU* che esegue il programma di controllo *nei confronti dei dispositivi*.

- Si è introdotta una *forma di sincronizzazione* tra due dispositivi di per sé *asincroni*.

## Programmi di controllo I/O
## Prestazioni del S.O. (1)

- La notevole differenza di velocità tra CPU e dispositivi periferici fa sì che i programmi di controllo trascorrano la maggior parte del loro tempo in *attesa*.

- I tempi legati all'accesso alla memoria principale per depositare i caratteri della scheda (80) o prelevare i caratteri della linea (120) (a cura dei processori dei dispositivi) sono trascurabili



## Programmi di controllo I/O
## Prestazioni del S.O. (2)

- E' l'attesa dell'*esecuzione fisica* del comando da parte del dispositivo che rallenta i programmi di controllo di I/O del S.O.



- Esempio: LP

| | istruzioni macchina |
|---|---|
| inizializzazione | 2 |
| attesa fino a che LP pronta | ? |
| invia comando | 5 |
| decrementa LS | 2 |
| calcola nuovo indirizzo | 5 |
| test su LS | 2 |

N.B.: tecnologia 1976 (oggi: processor clock > 1 GHZ)



- Il tempo di esecuzione medio di una istruzione macchina, assumendo un tempo di ciclo di 1 usec, e` circa 2,5 – 3 usec
- Il controllo della stampa di una linea richiede circa 45 usec
- Velocita` di stampa: 20 linee / secondo
- Tempo di stampa di una linea: 1/20 sec = 50000 usec



## Prestazioni

- Il tempo necessario per eseguire n programmi è:

$$t = I + CLE + O$$

dove:

- I = somma dei tempi di input
- CLE = somma dei tempi per: compile, load, execute
- O = somma dei tempi di output



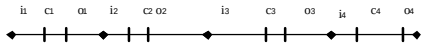
- Il throughput è:

$$\text{throughput} = \frac{n}{t} = \frac{n}{I + CLE + O}$$

- Per migliorare il throughput occorre *sovrapporre* le tre fasi I, CLE, O nel tempo

- Tale sovrapposizione richiede che i dispositivi periferici e la CPU operino in modo *il più possibile indipendente*.

## Sovrapposizione tra ingresso e uscita

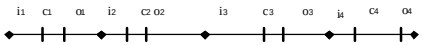


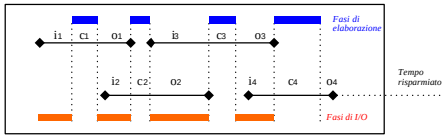

- throughput di un sistema seriale:

$$\frac{n}{t} = \frac{n}{CLE + I + O} = \frac{n}{CLE + \sum_{k=1}^{n} (i_k + o_k)}$$



## Sovrapposizione tra ingresso e uscita






- throughput di un sistema con sovrapposizione di I/O:

mk = max(Ik+1, Ok) k = 1,2,...,n-1

tempo di I/O:

$$M = i_1 + \sum_{k=1}^{n-1} m_k + o_n$$

throughput: $\frac{n}{t} = \frac{n}{CLE + M}$



## Programma di controllo I/O (1)

- Per raggiungere l'obiettivo della sovrapposizione dell'ingresso di un programma con l'uscita di un altro occorre *un unico programma di controllo di I/O*, IOC (input-output control).

- IOC invia comandi ad ogni dispositivo PRONTO ed attende solo quando tutti i dispositivi sono OCCUPATI.



## Programma di controllo I/O (2)

- Il programma riceve come dati di ingresso il numero di linee da stampare e l'indirizzo di memoria in cui sono contenute.

- Il programma termina quando tutte le linee sono stampate e CR è vuoto (LIBERO).

- A differenza del caso precedente, il programma non attende se uno dei dispositivi è PRONTO, ma lo utilizza.



## Programma IOC

INDLP = indirizzo di inizio area di memoria contenente le linee da stampare

INDCR = indirizzo di inizio area di memoria in cui inserire le schede dati

IC1 = indirizzo corrente relativo a INDLP

IC2 = indirizzo corrente relativo a INDCR

programma IOC:

```
fissa INDCR
IC1 := INDLP, IC2 := INDCR
SL := 0
repeat
    attendi mentre CR and LP OCCUPATI
    if CR PRONTO then
        invia comando (IC2)
        SL := SL + 1
        calcola il nuovo indirizzo IC2
    fi
    if LP PRONTO then
        invia comando(IC1)
        LS := LS - 1
        calcola nuovo indirizzo IC1
    fi
until LS = 0 and CR LIBERO
```



## Funzionamento con sovrapposizione delle attività di I/0

- Programma di controllo del sistema di calcolo da parte del S.O.:

```
LS := 0
repeat
    fase di I/O
    fase di elaborazione
until halt
```

```
if SL > 0 then
    compile; load; execute
fi
```



Osservazioni:

- Le due fasi, I/O ed elaborazione, sono sequenzializzate
- La fase di attesa della CPU per il completamento delle operazioni di I/O si è ridotta ma è comunque presente (si ha attesa quando entrambi i dispositivi sono occupati)
- Si tratta inoltre di un'attesa attiva (*busy waiting*) che disturba l'accesso dei processori dei dispositivi di I/O alla memoria
- Per migliorare ulteriormente le prestazioni del sistema è necessario sovrapporre le fasi di I/O e di elaborazione



## Sovrapposizione attività CPU e I/O (1)

- Il coinvolgimento della CPU nelle operazioni di I/O è modesto.
- Per migliorare le prestazioni occorre che durante le fasi di attesa dei programmi di controllo (in cui viene completata l'esecuzione degli specifici comandi di I/O), la CPU possa eseguire altri programmi.
- Occorre cioè *sovrapporre anche la attività della CPU con le operazioni di I/O*.



## Sovrapposizione attività CPU e I/O (2)

- Nello schema proposto (programma IOC) la sovrapposizione non è possibile perchè la CPU *si dedica* alla esecuzione dei programmi di controllo I/O, rimanendo comunque impegnata (in *busy waiting*) per l'intera durata della più lunga tra le operazioni di I/O.

- La gestione delle operazioni di I/O richiede peraltro *il contributo* della CPU per l'invio dei comandi ai dispositivi.



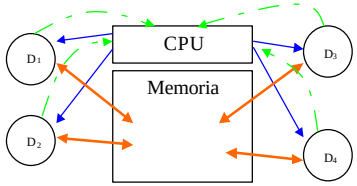
## Sovrapposizione attività CPU e I/O (3)

- La sovrapposizione potrebbe essere ottenuta inserendo nei programmi, ad intervalli regolari, la richiesta di esecuzione dei programmi di controllo. Questa soluzione è *inaccettabile* perchè fa ricadere sul programmatore un compito del S.O.

- La soluzione si ha attraverso il concetto di *interruzione*, realizzato tramite hardware.

## Interruzione da dispositivo (1)

- E' un segnale hardware inviato da un dispositivo di I/O alla CPU per indicare che un comando è stato eseguito




dati
comandi
interruzioni



## Interruzione da dispositivo (2)

Ciclo base di Fetch-Execute *in assenza di interruzioni* :

```
repeat
    IR := M[PC]
    PC := PC + 1
    execute(istruz. in IR)
until  CPU halt
```



## Modello di comportamento dell'interruzione da dispositivo

- Si può supporre che nella CPU sia contenuto un *vettore di bit* (registro di richieste di interruzione - IRR) in cui ogni bit *memorizza* l'interruzione *di un particolare dispositivo*.



- Il ciclo di Fetch-Execute diventa:

```
repeat
    if  IRR = 0 then
        IR := M[PC]
        PC := PC + 1
    else  IR := M[0] fi
    execute(istruz. in IR)
until  CPU halt
```

- M[0] è l'indirizzo in memoria di una procedura, chiamata *interrupt routine*



L'azione della *interrupt routine* è la seguente:

interrupt procedure:

```
begin
    salva lo stato del programma interrotto;
    x := indice del bit che ha causato l'interruzione;
    IRR[x] := 0;   /* azzera il bit di richiesta x-esimo */
    chiama il programma di controllo x-esimo;
    ripristina lo stato; continua il programma interrotto;
end
```



## Interrupt da timer e gestione a polling dell'I/O (1)

- In alternativa alla gestione ad interrupt dei singoli dispositivi, è possibile utilizzare un unico *interrupt periodico* generato da un timer hardware (*real-time clock*) con frequenza programmabile dal S.O.

- La routine di servizio dell'interrupt andrà ad esaminare le *condizioni di attivazione* dei programmi di controllo I/O, provvedendo eventualmente ad effettuarne la chiamata.

## Interrupt da timer e gestione a polling dell'I/O (2)

- Le variabili dei programmi di controllo I/O *non sono inizializzate ad ogni attivazione* del programma, ma vengono trattate come variabili globali. Ad esempio il programma di controllo del CR quando attivato usa il valore corrente di SL.
- Ogni programma di controllo I/O può essere scritto ed eseguito indipendentemente dagli altri, pur mantenendo la sovrapposizione tra elaborazione e gestione dell'I/O.



timer interrupt procedure:

*begin*
salva lo stato del programma interrotto;
*if* CR READY *then* call CR_control *fi* ;
*if* LP READY *and* LS > 0 *then* call LP_control *fi* ;
ripristina lo stato; continua il programma interrotto;
*end*



## Programmi di controllo I/O (con timer interrupt)

LP_control:

```
begin
    invia comando(IC1)
    LS := LS - 1
    calcola nuovo indirizzo IC1
end
```

CR_control:

```
begin
    invia comando (IC2)
    SL := SL + 1
    calcola il nuovo indirizzo IC2
end
```

- Lo schema che fa uso di un unico interrupt da timer è particolarmente *semplice*: è sufficiente un unico livello di interruzione.



E' semplice, ma è meno *efficiente* rispetto all'impiego di interrupt da dispositivi:

- Può accadere che nessuna delle condizioni di attivazione sia verificata. Si ha pertanto un *overhead* dovuto (più che alla valutazione delle condizioni) al salvataggio e ripristino del programma interrotto.
- Un dispositivo può rimanere in attesa del comando successivo (al più per un periodo del real-time clock).