# Struttura interna del sistema operativo Linux

## 1. Caratteristiche dell'hardware

A cura di:
Anna Antola  Giuseppe Pozzi
DEI, PoliMI, Milano
anna.antola/giuseppe.pozzi@polimi.it
- versione del 10 marzo 2003 -

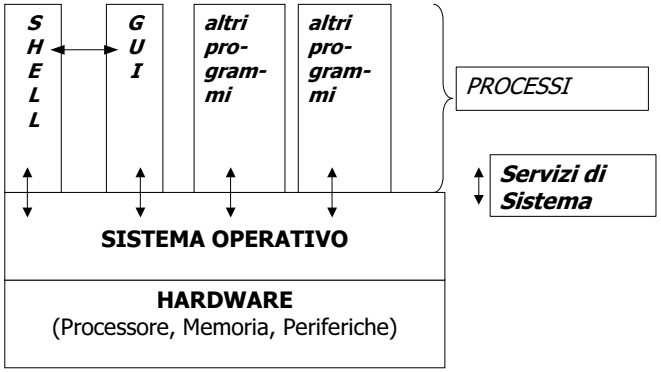


# Funzionalità del Sistema Operativo

- Realizzazione di macchine virtuali (processi) che eseguono i programmi in parallelo: astrazione rispetto allo hardware.
- Disponibilità di servizi di sistema (invocabili tramite chiamate di sistema) per l'utente e per i programmi applicativi.
- Gestione delle risorse fisiche del calcolatore (processore, memoria, periferiche).



# Hardware, Sistema Operativo e Processi






# Caratteristiche generali di Linux (1)

**Realizzazione dei processi paralleli**

- Linux è un SO multiprogrammato **time-sharing**. La virtualizzazione del parallelismo è ottenuta facendo eseguire in alternanza i diversi programmi dall'unico processore.
- Ad ogni programma è associato un **quanto di tempo**. Allo scadere del quanto di tempo il programma viene sospeso dall'esecuzione (*preemption*) e un nuovo programma può utilizzare il processore. In questo modo si garantisce un utilizzo equo del processore a tutti i programmi.
- Un processo può essere sospeso dall'esecuzione:
  - allo scadere del quanto di tempo;
  - per sospensione volontaria, tipicamente per l'esecuzione di una operazione di ingresso/uscita.

# Caratteristiche generali di Linux (2)

## Funzionalità esportate ai programmi applicativi (servizi di sistema):

- Servizi per la gestione e realizzazione dei processi;
- Servizi di accesso ai file;
- Servizi di accessi ai file speciali che rappresentano le periferiche (terminale, stampante).



# Caratteristiche generali di Linux (3)

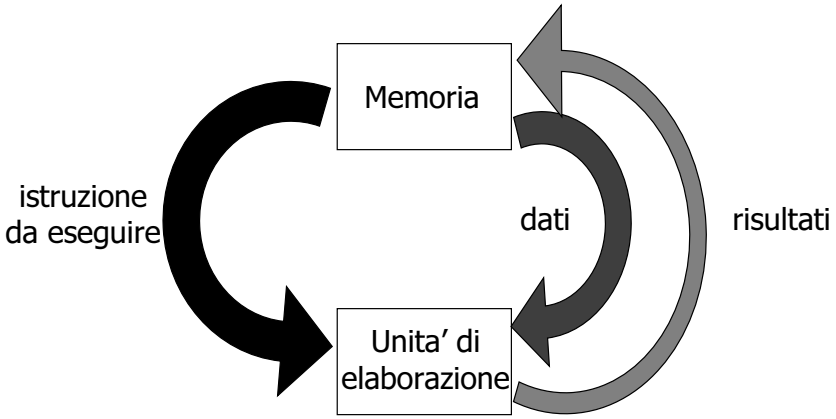
**Gestione delle risorse fisiche del sistema**

Il sistema operativo deve gestire risorse fisiche (scarse e condivise tra i processi) in modo efficiente:

- **gestione del processore (CPU)** che deve essere assegnato ai diversi processi in esecuzione e al SO stesso (*nucleo o kernel del SO*);
- **gestione della memoria centrale** che deve essere partizionata per contenere il codice e i dati dei processi applicativi in esecuzione e del SO stesso (*gestore della memoria*);
- **gestione delle periferiche** che vengono viste come file speciali, cui è possibile accedere astraendo dai dettagli di funzionamento e che vengono aggiornate frequentemente (*device driver*);
- **gestione della memoria di massa** che consente di accedere alla memoria di massa tramite "unità di informazioni elementari" rappresentate da file e direttori (*file system*).



# Architettura di von Neumann






# Architettura di un calcolatore

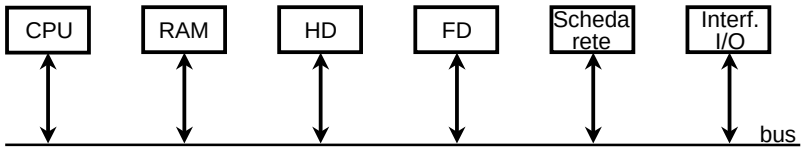
Insieme dei componenti collegati da un bus:

- CPU (central processing unit);
- memoria;
- disco fisso;
- disco flessibile;
- scheda di rete;
- interfacce per input/output.

Dispositivi di input/output:

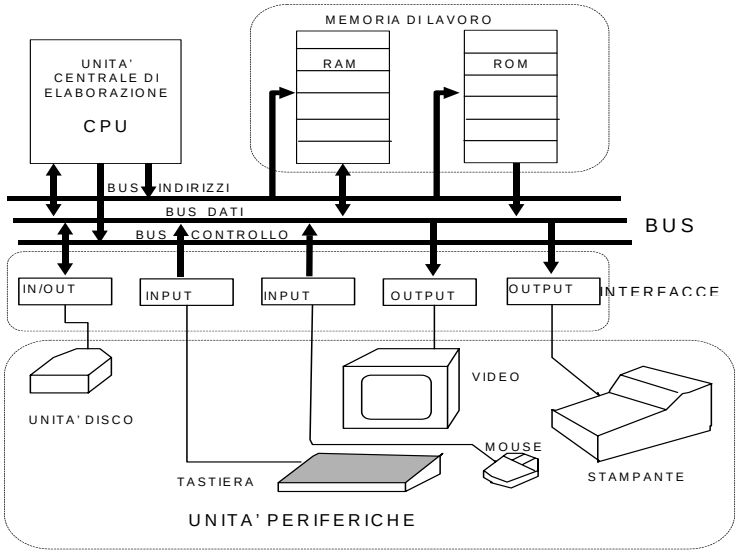
- monitor;
- tastiera;
- porte di comunicazione.

# Architettura di un calcolatore






# Caratteristiche dell'hardware






# Caratteristiche dell'hardware

- Per poter eseguire un programma è necessario che questo sia caricato (in formato eseguibile) nella **memoria di lavoro**.
- La CPU **legge da memoria, interpreta ed esegue le istruzioni** del programma operando sulle variabili (dati). Può essere considerata l'unità attiva del calcolatore
- La CPU per leggere ed eseguire le istruzioni **gestisce, controlla e temporizza** il funzionamento delle altre unità (memorie e interfacce) tramite i segnali del bus
- Le **istruzioni** devono essere espresse come **istruzioni macchina**, cioè nel formato direttamente interpretabile dalla CPU.
- Le **variabili** devono essere "espresse" in modo da essere accessibili alla CPU, cioè tramite riferimenti costituiti da un **indirizzo** della memoria di lavoro.



# Memoria di lavoro

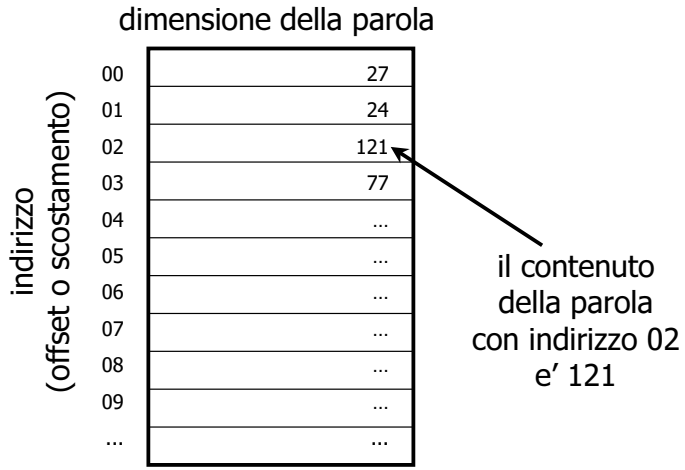
E' un insieme ordinato di parole (celle) che possono contenere (memorizzare) informazioni, e cioè **istruzioni** e **dati**.

Una **parola** di memoria è costituita da h **elementi di memoria binari** (ad es. h = 8, **16**, **32**, 64 bit).

La posizione di ogni parola di memoria è identificata in modo univoco da un numero intero positivo, detto **indirizzo** (o offset o scostamento) della parola di memoria.

Per **accedere** ad una generica parola di memoria è necessario fornire il suo indirizzo (che la identifica in modo univoco).

# Memoria centrale - RAM






# Processore (1)

- Esecuzione **sequenziale:** viene prelevata da memoria un'istruzione alla volta. Il registro **Program Counter** - **PC** viene man mano incrementato in modo da contenere l'indirizzo della parola di memoria in cui è presente la prossima istruzione da eseguire.
- L'istruzione da eseguire, una volta prelevata dalla memoria con una operazione di lettura, viene decodifcata (interpretata) dall'**unità di controllo**.

  L'esecuzione di una istruzione comporta la generazione di segnali del bus di controllo per l'accesso a memoria o a interfaccia di I/O, e la generazione di segnali di controllo interni alla CPU per l'esecuzione di operazioni aritmetiche o logiche.



# **Processore** (2)

L'**unità di controllo** ha il compito di generare i segnali di controllo per l'accesso a memoria o periferica e quelli che comandano i vari elementi della CPU.

L'**unità aritmetico-logica** (**ALU** - *Arithmetic Logic Unit*) è l'elemento della CPU in grado di eseguire le operazioni.

La CPU può essere dotata di **registri di lavoro** che mantengono i valori degli operandi che devono essere utilizzati dall'ALU, e i risultati prodotti.



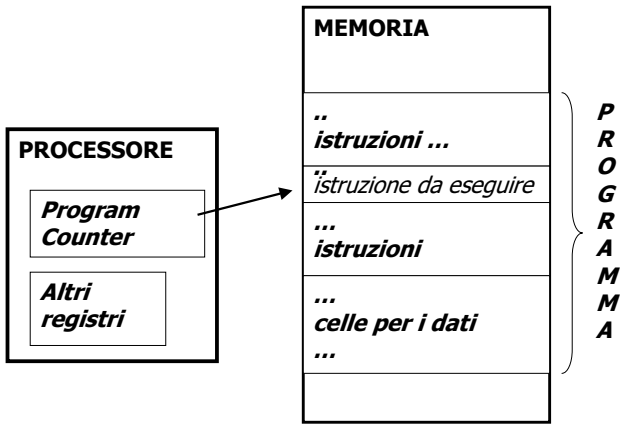
# Esecuzione delle istruzioni

Ciclo **Fetch–Decode–Execute**

- Legge da memoria l'istruzione il cui indirizzo è contenuto in **Program Counter**.
- Determina il tipo dell'istruzione corrente (**decodifica**) e **esegue** l'istruzione eventualmente accedendo a memoria per il recupero degli operandi.
- **Incrementa** il **program counter** (**PC**) in modo che contenga l'indirizzo dell'istruzione successiva.
- Torna al punto 1 e inizia a eseguire l'istruzione successiva.

All'avviamento dell'esecuzione, il Program Counter deve contenere l'indirizzo della prima istruzione eseguibile

# Processore e memoria





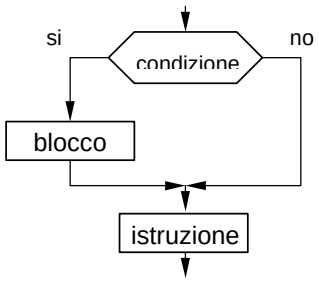

# Istruzioni

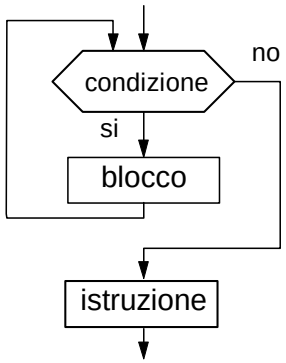
Classi tipiche di istruzioni:

- trasferimento dati tra registri e memoria, e viceversa;
- aritmetico-logiche;
- modifica del flusso di esecuzione: salti incondizionati e salti condizionati;
  - le istruzioni di salto consentono di realizzare a livello di linguaggio macchina i costrutti di controllo dei linguaggi di alto livello.



# Istruzioni di salto



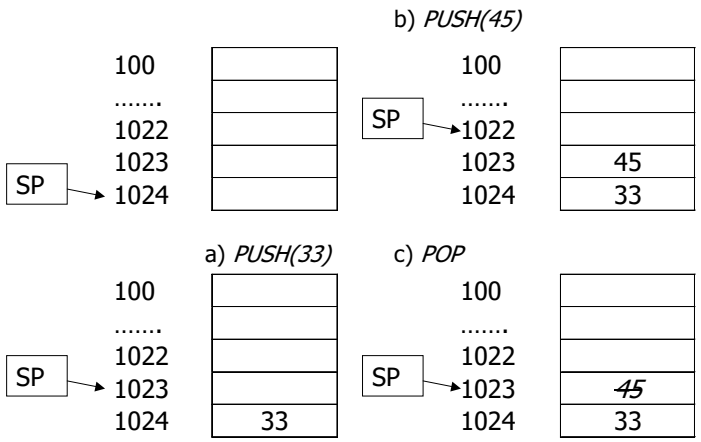






# Stack e Stack Pointer

- **Pila (stack):** zona della memoria centrale in cui la scrittura e la lettura delle informazioni avviene in modalità LIFO
- **Puntatore alla pila (Stack Pointer - SP):** registro del processore che consente di gestire lo stack
- Funzionamento:
  - la pila cresce da indirizzi alti della memoria verso indirizzi bassi della memoria;
  - SP punta alla prima parola libera della pila;
  - scrittura in pila (PUSH(dato)): il valore di *dato* viene scritto nella parola di memoria indirizzata da SP e SP viene decrementato;
  - lettura dalla pila (POP): SP viene incrementato e la parola di memoria indirizzata da SP viene letta.

# Stack e Stack Pointer






# Stack e sottoprogrammi (1)

- Lo stack viene utilizzato nel linguaggio C (come in molti altri linguaggi) come meccanismo di supporto alla gestione delle **chiamate dei sottoprogrammi**.
- Il C prevede che, ad ogni attivazione di una funzione, venga creato sullo stack il corrispondente *record di attivazione* che contiene:
  - l'**indirizzo di ritorno al chiamante**, e cioè l'indirizzo dell'istruzione successiva - nel chiamante - a quella di invocazione della funzione;
  - le **variabili locali della funzione** e i valori dei parametri all'atto dell'invocazione della funzione.
- In questo modo è possibile avere chiamate di funzioni **annidate e ricorsive.**

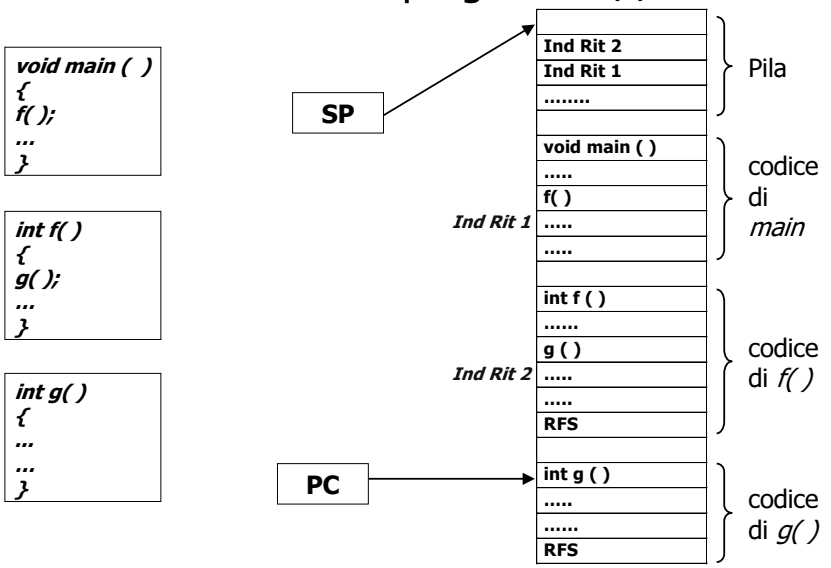


# Stack e sottoprogrammi (2)

- I record di attivazione vengono impilati sullo stack secondo l'ordine di chiamata.
- Quando una funzione termina, il corrispondente record di attivazione viene "deallocato":
  - viene rilasciata la zona di memoria dello stack che contiene le variabili locali (e i parametri) della funzione terminata;
  - viene eseguita un'istruzione macchina (**RFS** - Return From Subroutine) che preleva l'indirizzo di ritorno dallo stack e lo forza nel registro Program Counter. In questo modo il chiamante può proseguire nell'esecuzione a partire dall'istruzione successiva a quella di chiamata.
- Il record di attivazione in cima allo stack rappresenta quindi quello della funzione attualmente in esecuzione.



# Stack e sottoprogrammi (3)



# Accesso delle periferiche (1)

- Le unità periferiche interagiscono con il calcolatore, e cioè con il processore e la memoria centrale, attraverso **interfacce di ingresso/uscita** (dette anche adattatori) collegate al bus di sistema.
- Le interfacce sono quindi **dispositivi circuitali** che consentono al calcolatore di scambiare informazioni con le **unità periferiche**.
- La **struttura delle interfacce di ingresso/uscita** è costituita, in modo molto schematico, da alcuni **registri** leggibili e/o scrivibili da parte del processore, ma comunicanti anche con il mondo esterno (periferiche), e da circuiti ausiliari.

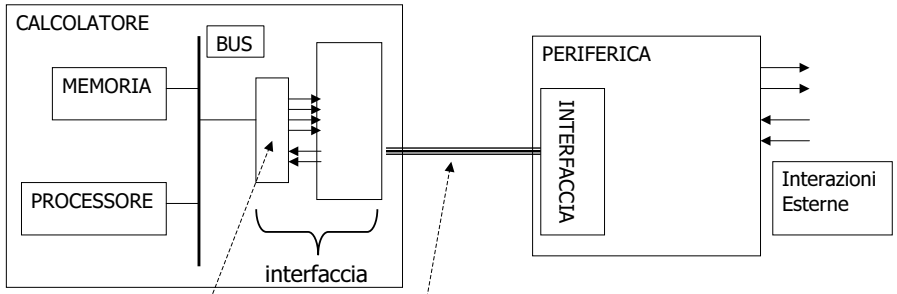


# Accesso delle periferiche (2)

- Considerando uno schema molto semplificato, una interfaccia è costituita dai seguenti registri:
  - registro **dati** periferica, che contiene i dati che la periferica deve leggere (periferica di uscita) o scrivere (periferica di ingresso);
  - registro **comandi** (o **controllo**) periferica, che contiene indicazioni sulle operazioni che la periferica deve compiere e sul suo modo di funzionamento;
  - registro di **stato** periferica, che contiene informazioni sullo stato di funzionamento della periferica (ad esempio "pronta" a gestire un nuovo dato, o "occupata").
- Molti processori utilizzano **istruzioni macchina specializzate** per eseguire le operazioni di ingresso e uscita sulle periferiche. Tali istruzioni (ad es. Nelle architetture Intel sono `IN` e `OUT`) fanno riferimento ai registri dell'interfaccia.



# Collegamento di una periferica al calcolatore (a livello di sistema)






# Tecniche di gestione dell'ingresso e uscita

- La gestione dell'I/O può avvenire tramite:
  - **controllo di programma;**
  - **interruzione;**
  - **accesso diretto alla memoria (DMA).**
- Le tecniche di gestione devono consentire *interazioni* tra calcolatore e periferica per:
  - la ***sincronizzazione*** tra periferica e calcolatore;
  - il ***trasferimento*** del dato da periferica a calcolatore o viceversa.

## Sincronizzazione e trasferimento

- Stampa di un carattere su stampante: *interazioni*
  - la stampante deve segnalare al calcolatore che è disponibile ad accettare un nuovo dato per la stampa (sincronizzazione);
  - il calcolatore esegue **un'istruzione di uscita** `OUT` sul registro dati della stampante, che potrà quindi eseguirne la stampa (trasferimento).
- Acquisizione di un carattere da tastiera: *interazioni*
  - la tastiera deve segnalare al calcolatore che è stato premuto un tasto e quindi esiste un carattere disponibile (sincronizzazione);
  - il calcolatore esegue **un'istruzione di ingresso** `IN` dal registro dati della tastiera per acquisire il valore del carattere (trasferimento).



## Controllo di programma (1)

La sincronizzazione e il trasferimento vengono eseguiti dal programma di gestione della periferica.

Il programma di gestione della periferica:

- legge il registro di stato dell'interfaccia della periferica (**`IN registro di stato`**) ;
- se lo stato è "periferica non pronta" il programma torna a leggere il registro di stato; altrimenti
- esegue il trasferimento del dato tramite:
  - **`IN registro dati`** se la periferica è di ingresso (ad es. tastiera);
  - **`OUT registro dati`** se la periferica è di uscita (ad es. stampante).



## Controllo di programma (2)

- Ovviamente:
  - se la periferica è d'ingresso, dopo l'istruzione IN il dato acquisito deve essere copiato nella variabile relativa;
  - se la periferica è di uscita, il dato che deve essere trasferito tramite l'istruzione OUT deve prima essere prelevato dalla variabile relativa.



## Controllo di programma (3)

- La gestione delle periferiche a controllo di programma implica un ciclo, da parte del processore, di attesa di periferica pronta.
- La tecnica è ideale se:
  - il ciclo di istruzioni del programma di gestione della periferica è inserito nel programma utente;
  - quando il programma utente è in attesa, non ha null'altro da fare;
  - non c'è nessun altro programma utente da eseguire.
- Se la periferica è pronta, l'esecuzione del programma di gestione della periferica è molto veloce.

# Interruzione (1)

Il meccanismo di interrupt si basa su:

- un insieme di *eventi* rilevati a **livello hardware** dal processore (*interruzioni*). Ad es. un segnale opportuno di una periferica, una condizione di errore …
- un insieme di funzioni, ognuna associata in generale ad un evento. Ogni funzione viene chiamata **gestore dell'interrupt** o **routine di interrupt**.
- Quando il processore rileva un evento:
  - interrompe il programma in esecuzione;
  - salta automaticamente ad eseguire la routine di interrupt corrispondente, ed esegue il servizio richiesto;
  - al termine dell'esecuzione della routine, torna ad eseguire il programma che era stato interrotto (in modo **trasparente** rispetto al programma stesso).

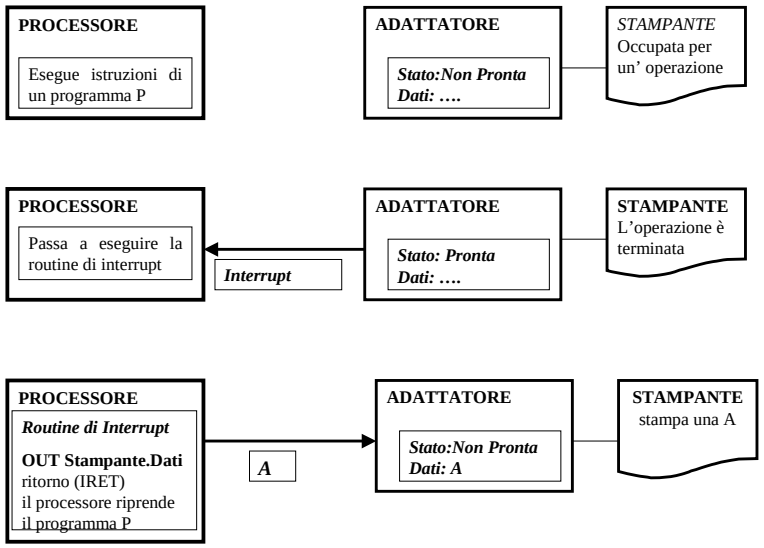


# Interruzione (2)

Acquisizione di un carattere da tastiera con gestione a interrupt:

- la parte di programma che esegue il trasferimento del dato dal registro dati della tastiera (`IN Registro Dati`) **NON** è nel programma ma è "silente" in memoria in una locazione convenuta;
- quando l'interfaccia della periferica scrive il dato nel registro dati, con un **segnale allerta il processore** (*sincronizzazione*);
- il processore **interrompe** l'esecuzione del programma in corso e salta automaticamente a eseguire la parte di programma che legge il registro dell'interfaccia (*trasferimento*);
- al termine di questo, il processore riprende il programma interrotto;
- in pratica, la periferica ha deciso **quando** l'istruzione di lettura dal registro dati dell'interfaccia della periferica deve essere eseguita.








# Interruzione (3)

Il meccanismo di interruzione è *simile* a quello di invocazione di un sottoprogramma e infatti prevede:

- la cessione del controllo dell'esecuzione tra il programma interrotto e la routine di risposta all'interrupt. E' quindi necessario **salvare sullo stack l'indirizzo di ritorno** di tale programma (e cioè l'indirizzo dell'istruzione successiva a quella in cui il programma è stato interrotto);
- il ritorno all'esecuzione del programma interrotto. Le routine di risposta all'interrupt utilizzano, per il ritorno, una istruzione macchina **IRET** speciale e diversa da RFS.

La differenza fondamentale rispetto alle chiamate a sottoprogramma è che queste sono attivata dal programma in esecuzione, mentre le risposte ad interrupt sono "scatenate" da eventi rilevati dal processore e asincroni rispetto all'esecuzione del programma che viene interrotto.

# Interruzioni e annidamento(4)

- Il salvataggio dell'indirizzo di ritorno sullo stack consente di gestire **interruzioni nidificate**, al pari di quanto accade nella gestione dei sottoprogrammi.
- Un interrupt può quindi interrompere, in linea di principio, la routine di risposta di un altro interrupt.



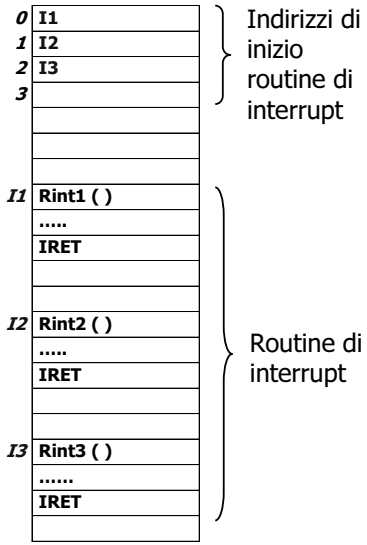
# Riconoscimento delle interruzioni (1)

Per gestire correttamente le risposte all'interruzione è necessario che a livello di sistema siano definite delle convenzioni per:

- l'identificazione della periferica interrompente. Ciò avviene tramite appositi circuiti collegati al bus di sistema (che approfondiremo più avanti);
- l'identificazione della **posizione in memoria** di lavoro **delle routine di risposta**. In generale, in corrispondenza di ogni interrupt, viene anche fornito al processore il **vettore di interrupt** e cioè un'*indicazione* della cella di memoria in cui si trova la prima istruzione della routine di interrupt.



# Riconoscimento delle interruzioni (2)






# Priorità degli interrupt e interrompibilità dei programmi

Un interrupt:

- può interrompere un programma in esecuzione (sia esso il Sistema Operativo o un programma utente);
- può essere a sua volta interrotto da un altro interrupt (interrupt nidificati).

Per gestire correttamente l'interrompibilità, devono essere disponibili a livello di processore dei meccanismi che consentono di attribuire **priorità agli interrupt** e cioè definire se un interrupt con una certa priorità può interrompere l'esecuzione del processore stesso.

# Priorità degli interrupt (1)

Un meccanismo generalmento adottato per gestire le priorità degli interrupt è il seguente:

- al **processore è associato un livello di priorità**, scritto in un particolare registro del processore stesso (registro di stato – **PSR** – Processor Status Register, o anche PSW – Processor Status Word);
- il livello di priorità del processore può essere modificato, a livello software, tramite opportune istruzioni macchina privilegiate che scrivono nel registro di stato;
- ad **ogni interrupt è associato un livello di priorità**, in generale fissato a livello hardware dalla configurazione della macchina;
- un interrupt viene **accettato**, e cioè sentito dal processore, solo se il suo livello di priorità è superiore a quello del processore stesso in quell'istante;
- se l'interrupt ha un livello di priorità inferiore o uguale a quello attuale del processore, esso viene tenuto "**pendente**" fino a quando la priorità del processore non consente che venga accettato.



# Priorità degli interrupt (2)

Esempio di funzionamento:

- all'inizio al processore viene associato il livello di priorità più basso. Il programma in esecuzione è quindi interrompibile da qualsiasi interrupt;
- si verifica un'interruzione che viene accettata e alla quale è associato un livello di priorità *X*. Al momento della cessione del controllo alla routine di risposta all'interrupt, viene salvato non solo l'indirizzo di ritorno del programma interrotto ma anche il livello di priorità del processore. Il livello di priorità *X* viene quindi scritto nel registro di stato PSR e quindi l'interrupt accettato può essere interrotto solo da interrupt di livello superiore (>=X+1). La scrittura nel registro PSR viene effettuata dal Sistema Operativo;
- ….. si itera il procedimento descritto per ogni eventuale altro interrupt accettato;
- quando una routine di risposta all'interrupt termina, oltre a ripristinare nel Program Counter l'indirizzo di ritorno, deve prevedere di ricaricare nel PSR il livello di priorità salvato.



# Gestione degli errori

Durante l'esecuzione di un programma possono verificarsi degli errori che ne impediscono la prosecuzione.

Esempi tipici di questi errori sono quelli di tipo "matematico" legati a operazioni impossibili (divisione per 0), di accesso a zone di memoria non permesse (indirizzi non validi), di esecuzione di istruzioni previlegiate …

Queste segnalazioni vengono spesso trattate con interruzioni e quindi, al loro verificarsi, il programma si interrompe e passa ad eseguire una routine del Sistema Operativo che prevede la gestione dell'errore stesso.



# Accesso Diretto alla Memoria - DMA

Il meccanismo di accesso diretto alla memoria (DMA) prevede che:

- la **periferica trasferisca** in modo **autonomo**, cioè senza l'intervento del processore che esegue istruzioni di ingresso o uscita del singolo dato, **un certo numero di dati in memoria centrale o dalla memoria centrale;**
- il meccanismo di DMA è utilizzato da periferiche quali i dischi magnetici dotate di interfacce "intelligenti" (DMA e controllori del disco) che sono in grado di trasferire velocemente uno o più settori di disco da o in memoria centrale.

# DMA (1)

La tecnica del DMA prevede le seguenti fasi:

1. ***Predisposizione.*** Una procedura (del Sistema Operativo) scrive nei registri dell'interfaccia:
   - l'indirizzo della memoria dal quale iniziare il trasferimento;
   - l'indirizzo sulla periferica dal quale iniziare il trasferimento;
   - il numero di dati (parole di memoria) da trasferire;
   - la direzione del trasferimento (lettura da periferica o scrittura su periferica);
2. ***Attivazione.*** La procedura del Sistema Operativo scrive nel registro di controllo dell'interfaccia il comando di avviamento ("start") dell'operazione e quindi il processore passa ad eseguire altri programmi **lasciando libero il bus**. La periferica può a questo punto procedere in modo autonomo nel trasferimento.

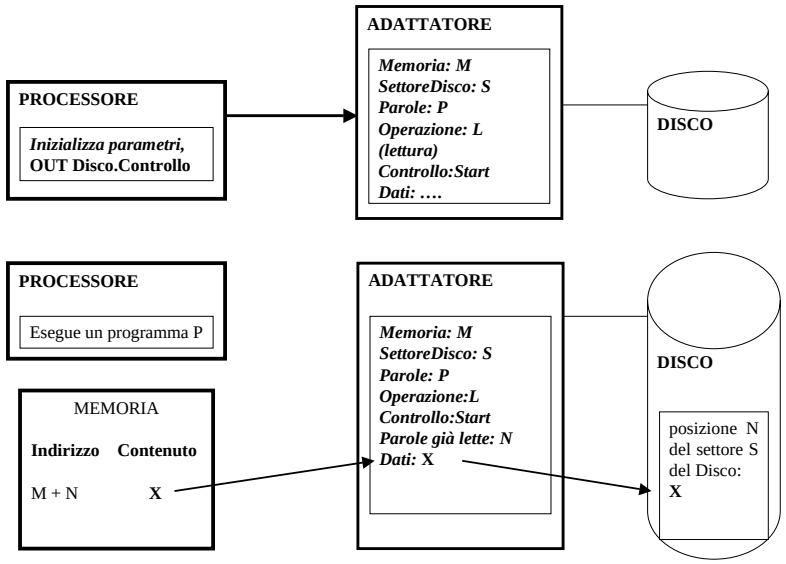


# DMA (2)

3. ***Trasferimento di un blocco di dati.***
   - Nel caso di lettura da periferica, ogni volta che dalla periferica arriva un dato all'interfaccia, l'interfaccia scrive il dato in memoria, incrementa l'indirizzo in memoria, decrementa il numero di dati ancora attesi e, se non sono arrivati tutti, aspetta il prossimo dato;
   - Nel caso di scrittura su periferica, ogni volta che le periferica è disponibile ad accettare nuovi dati, l'interfaccia preleva il dato da memoria, incrementa l'indirizzo in memoria, decrementa il numero dei dati ancora da trasferire e, se non sono trasferiti tutti, aspetta che la periferica sia disponibile per il prossimo dato.
4. ***Fine (tipicamente a interruzione).*** Se l'ultimo dato è stato trasferito, l'interfaccia segnala al processore che l'operazione in DMA si è conclusa.

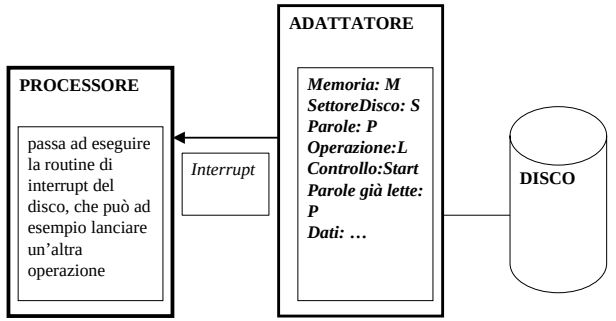


# DMA - esempio






# DMA - esempio



## DMA e Sistema Operativo

- Ai fini della realizzazione del Sistema Operativo, le periferiche gestite in DMA richiedono che il Sistema Operativo gestisca opportunamente aree di memoria (dette *buffer*) per il trasferimento dati.
- Inoltre, Linux nella definizione dei *device driver* distingue tra *block devices* (periferiche in DMA) e *character devices* (le altre periferiche).



## Meccanismi specifici per SO multiprogrammati

- Modo di funzionamento del processore e istruzioni privilegiate
- Rilocazione degli indirizzi di memoria
- Stack utente e stack di sistema



## Modalità di funzionamento del processore (1)

- Modo utente (**U**=User): è quello tipico di esecuzione dei processi applicativi;
- Modo Kernel (**S**=Supervisor o privilegiato o sistema): è quello tipico del Sistema Operativo.
- Durante l'esecuzione di un processo in modalità *user*, il processore può accedere al codice e dati del processo ma non ai dati e al codice del sistema operativo.
- Durante l'esecuzione in modalità *kernel*, il processore può accedere sia allo spazio di indirizzamento del sistema operativo sia a quello utente.
- Il modo di funzionamento del processore è memorizzato in uno dei bit del registro **PSW**.



## Modalità di funzionamento del processore (2)

- In modo U il processore può eseguire una parte delle istruzioni;
- in modo S il processore può eseguire tutte le istruzioni macchina.

- Le istruzioni eseguibili solo in modo S si chiamano ***istruzioni privilegiate***:
  - le istruzioni di accesso alle periferiche (`IN` e `OUT`) sono istruzioni privilegiate. Tutte le operazioni di ingresso e uscita sono gestite dal Sistema Operativo. Un processo eseguito in modo U non può accedere direttamente alle periferiche ma deve invocare i servizi del Sistema Operativo;
  - Se le istruzioni privilegiate vengono eseguite in modalità user generano un errore.

# Modifica del modo di funzionamento (1)

- Un programma in esecuzione in modo U può passare in funzionamento in modo S tramite una **chiamata di sistema** (system call) che invoca il servizio relativo.
- E' disponibile un'istruzione **non privilegiata SVC** (SuperVisor Call) che consente a un processo in esecuzione in modo U di invocare un servizio del SO.
- L'istruzione SVC esegue un salto ad una particolare funzione del SO (gestore di SVC) che è in grado di invocare il servizio richiesto:
  - la SVC non specifica l'indirizzo della funzione richiesta: tale indirizzo è specificato al processore tramite uno speciale vettore di interrupt associato alla SVC stessa. L'indirizzo d'inizio della funzione richiesta verrà quindi forzato nel Program Counter;
  - la SVC è un'istruzione non privilegiata e, dopo la sua esecuzione, il processore passa in modo S, cioè privilegiato. Quindi la SVC modifica il modo del processore (PSW);
  - poiche la SVC realizza una chiamata di funzione, deve essere previsto il ritorno al programma che l'ha invocata. E' quindi necessario salvare l'indirizzo di ritorno del programma che ha invocato la SVC. Il ritorno avviene tramite l'istruzione **IRET** che quindi è un'istruzione **privilegiata**.



# Modifica del modo di funzionamento (2)

**Interrupt**:

- le routine di risposta all'interrupt fanno parte del Sistema Operativo e quindi vengono eseguite in modo S;
- quando si verifica un'interruzione il processore passa ad eseguire la routine di risposta relativa all'evento;
- il modo del processore passa ad S (modifica di PSW), indipendentemente dal modo di esecuzione al momento dell'accettazione dell'interrupt stesso (U se processo applicativo in esecuzione, S se altra routine di risposta, o SO in esecuzione).

Sia nel caso di SVC che nel caso di interrupt il ***modo di partenza*** del processore deve essere **salvato sullo stack** per consentire un ritorno corretto al termine dell'esecuzione del servizio relativo alla SVC o della routine di interrupt.

Questo equivale a salvare sullo stack il valore del registro PSW (che nel caso degli interrupt contiene anche il livello di priorità del programma interrotto).



# L'istruzione privilegiata IRET

- Al termine di una routine di risposta all'interrupt o dell'esecuzione di un servizio di SO è prevista l'istruzione IRET che esegue il ritorno
  - al programma interrotto, oppure
  - al programma che ha eseguito la SVC.
- L'esecuzione dell'istruzione IRET implica che vengano prelevate dallo stack le informazioni per:
  - il caricamento dell'indirizzo di ritorno nel registro PC;
  - il ripristino del modo di funzionamento al momento dell'interrupt o della SVC (modo di ritorno).



| *Meccanismo di salto* | *Modo di partenza* | *Modo di arrivo* | *Indirizzo di salto* | *Meccanismo di ritorno* | *Modo dopo il ritorno* |
|---|---|---|---|---|---|
| Salto a funzione | U<br>S | U<br>S | Specificato nell'istruzio-ne di chiamata | Istruzione di ritorno (RFS) | U<br>S |
| SVC | U | S | Vettore di interrupt | IRET | U |
| Interrupt | U<br>S | S<br>S | Vettore di interrupt | IRET | U<br>S |

# Rilocazione degli indirizzi (1)

- La **memoria centrale** deve essere partizionata per contenere il codice e i dati dei processi applicativi in esecuzione e del Sistema Operativo stesso. Le tecniche complesse di gestione della memoria verranno approfondite più avanti.
- Il linker costruisce programmi che sono eseguiti correttamente se caricati in memoria a partire dalla cella con indirizzo 0.
- Per poter garantire che un processo applicativo, eseguito quindi in modo U, possa essere caricato in una qualsiasi posizione di memoria (indipendentemente dagli altri processi in esecuzione in un certo istante) è necessario che il suo **codice eseguibile** sia **generato in formato rilocabile** (e cioè caricato a partire da un certo indirizzo di riferimento).
- Si dice **rilocazione dinamica** il meccanismo che traduce, durante l'esecuzione del programma, un indirizzo in formato rilocabile in un indirizzo fisico (cioè effettivo di memoria).

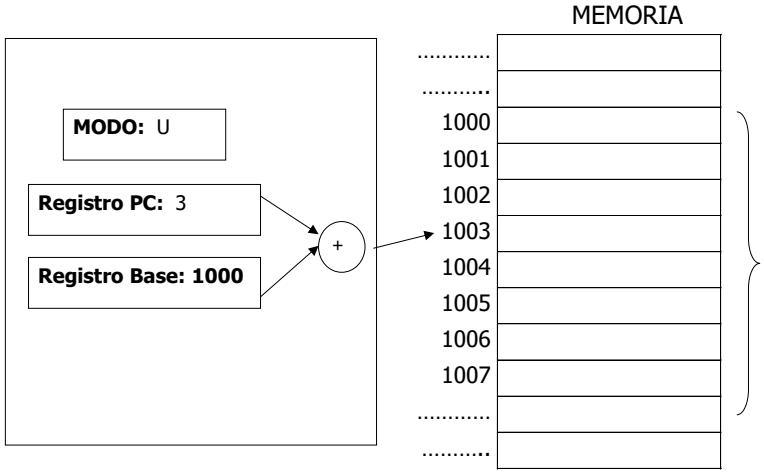


# Rilocazione degli indirizzi (2)

- Il meccanismo minimale che qui si considera per la rilocazione dinamica prevede:
  - la presenza di un **registro base** che contiene l'indirizzo d'inizio della zona di memoria in cui è effettivamente caricato il programma;
  - che il processore, in modo U, prima di accedere a memoria per la lettura di una istruzione o per la lettura/scrittura di un dato "costruisca" l'indirizzo effettivo sommando all'indirizzo rilocabile il valore del registro base;
  - quando un processo va in esecuzione, il Sistema Operativo provvede a caricare il registro base con l'opportuno valore relativo a quel processo.
- Quando il processore è in modo S la rilocazione non viene eseguita e gli indirizzi sono già indirizzi fisici.



# Rilocazione degli indirizzi: esempio






# Stack utente e stack di sistema (1)

- Per proseguire nella descrizione delle funzionalità del Sistema Operativo si considera un particolare meccanismo di *partizione dello stack*. Tale meccanismo non è necessariamente implementato in tutti i processori, anche se le funzionalità descritte devono, in qualche modo, essere offerte.
- Sono previsti 2 stack:
  - uno stack user che contiene le informazioni memorizzate durante l'esecuzione in modalità user. A livello hardware la gestione di tale stack richiede la presenza di un registro *UserStackPointer* (**uSP**).
  - uno stack kernel che contiene le informazioni memorizzate durante l'esecuzione in modalità kernel. A livello hardware la gestione di tale stack richiede la presenza di un registro *SystemStackPointer* (**sSP**).

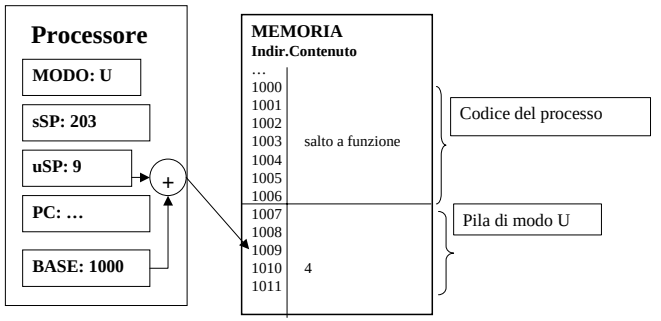
# Stack utente e stack di sistema (2)

Quando avviene una modifica del modo di funzionamento del processore (da U a S, oppure da S a U) deve cambiare anche lo stack utilizzato, e cioè lo Stack Pointer di riferimento.

| | *Modo di partenza* | *Modo di arrivo* | *Stack utilizzato per salvataggio/ripristino* | *Stack Pointer utilizzato per salv./ripristino* | *Stack e Stack Pointer attivi nel modo di arrivo* |
|---|---|---|---|---|---|
| Salto a funzione | U<br>S | U<br>S | User (salvataggio)<br>Sistema (salvataggio) | uSP<br>sSP | user<br>sistema |
| RFS | U<br>S | U<br>S | User (ripristino)<br>Sistema (ripristino) | uSP<br>sSP | user<br>sistema |
| SVC | U | S | Sistema (salvataggio indirizzo di ritorno e PSW) | sSP | sistema |
| Interrupt | U<br>S | S<br>S | Sistema (salvataggio indirizzo di ritorno e PSW) | sSP | sistema |
| IRET | S | U<br>S | Sistema (ripristino indirizzo di ritorno e PSW) | sSP | user<br>sistema |

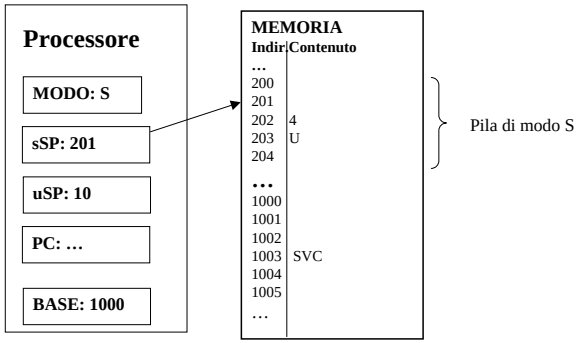


# Dopo salto a funzione






# Dopo esecuzione SVC






# Bibliografia


- Pelagatti G., Sistema Operativo Linux e TCP/IP, Progetto Leonardo, Bologna, 2002 - capitoli 7 e 8.