Input/Output
a livello hardware

Registri delle periferiche

- ciascuna periferica possiede un certo numero di registri che servono per gestirla
- i ***registri dati*** contengono i dati che la periferica deve leggere o scrivere
- i ***registri di controllo e stato*** contengono indicazioni sulle operazioni che la periferica deve svolgere oppure sullo stato della periferica
- nel registro di stato si trova un bit detto ***Ready*** che indica se la periferica è pronta, se *Ready* = 1, oppure che è occupata, se *Ready* = 0

  il bit *Ready* indica la possibilità per il processore di svolgere un'operazione di lettura o scrittura di un dato

  una periferica di ingresso, come una tastiera, è in stato di pronto quando un tasto è stato premuto e dunque c'è un carattere nel suo registro dati che il processore può leggere

  una periferica di uscita, come una stampante, è pronta se può ricevere un dato dal processore per stamparlo

Esempio di funzionamento

- una stampante molto rudimentale può funzionare nel modo seguente
  - appena accesa ha il bit *Ready* = 1, e il registro dati è vuoto
  - quando la *CPU* scrive un dato nel registro dati, il bit *Ready* va a 0
  - quando la stampante ha finito di stampare il dato e il registro dati è nuovamente vuoto, il bit *Ready* torna a 1
- invece una tastiera funziona nel modo seguente
  - appena accesa ha il bit *Ready* = 0, e il registro dati è vuoto
  - quando viene premuto un tasto, il bit *Ready* va a 1
  - quando la *CPU* legge il dato, il bit *Ready* torna a 0

Istruzioni di Input/Output

- il processore interagisce con le periferiche utilizzando certe istruzioni macchina specializzate, dette istruzioni di Input/Output (*I/O*)
- tali istruzioni macchina operano sui ***registri delle periferiche***
- gli indirizzi dei registri delle periferiche sono detti ***Port*** (o ***indirizzi di I/O***)
- ecco le due istruzioni di I/O fondamentali

IN	*port_di_input*	legge dal registro associato all'ind. *port_di_input*
OUT	*port_di_output*	scrive nel registro associato all'ind. *port_di_output*

- i nomi *IN* e *OUT* di tali istruzioni sono in parte convenzionali, e in alcuni processori possono differire

Funzionamento a *controllo di programma*

- il modo più semplice per gestire una periferica è detto a ***controllo di programma***
- tale modalità di gestione è usata principalmente per periferiche semplici e piuttosto lente
- generalmente essa è realizzata direttamente dal programma applicativo, in sistemi con *SO* semplice o del tutto assente
- pertanto essa è inadeguata per essere applicata in un *SO* multiprogrammato
- tuttavia è comunque istruttivo vedere come il controllo di programma funziona:
  1. leggi il registro di stato della periferica (*IN* sul *Port* del registro di stato)
  2. se il bit *Ready* = 1, va' al passo 3, altrimenti torna al passo 1 (ciclo di monitoraggio)
  3. esegui l'operazione voluta, per esempio:

     la lettura di un dato dalla tastiera, ossia esegui *IN* sul *Port* del registro dati

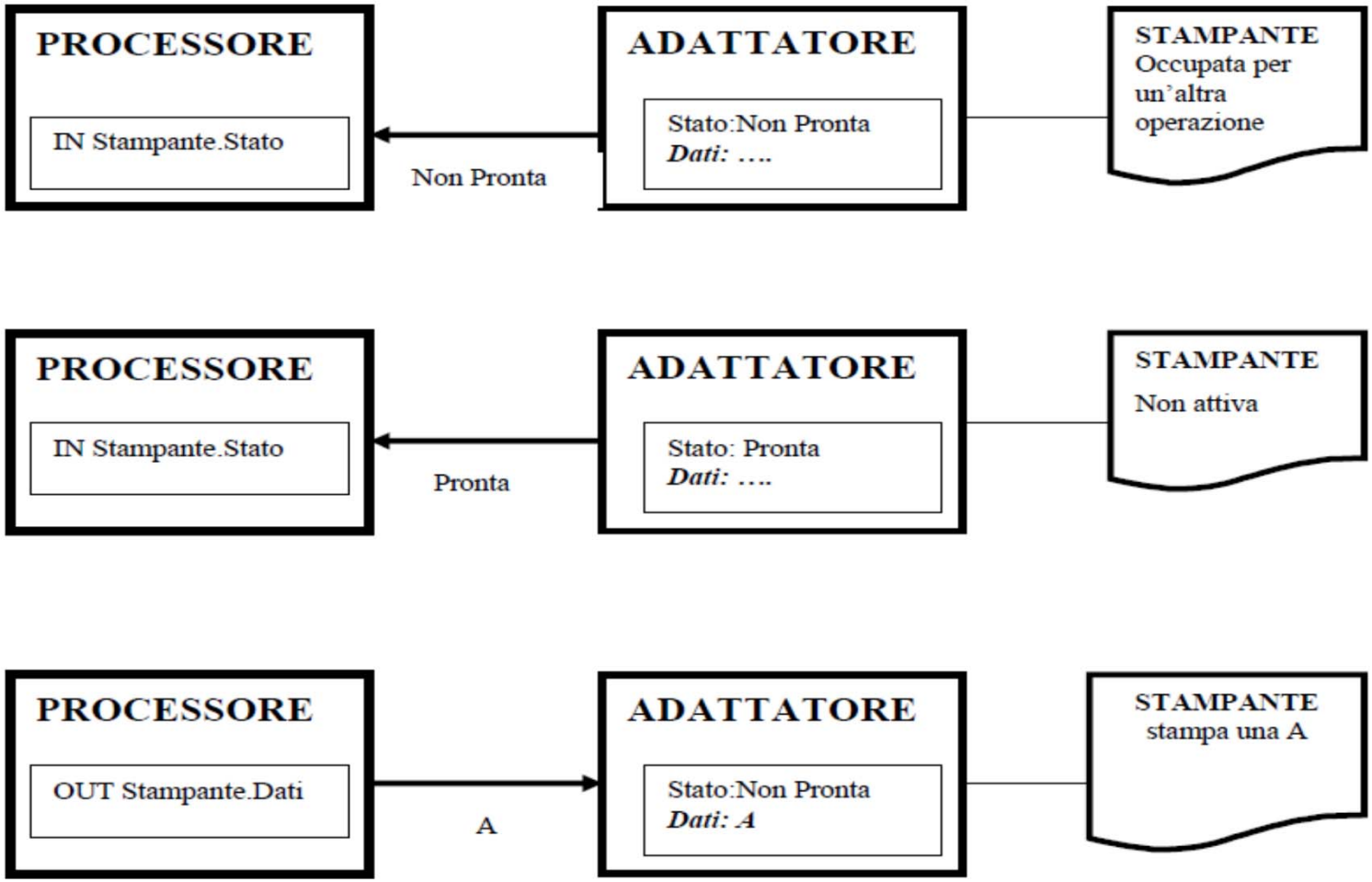
     l'invio di un dato alla stampante, ossia esegui *OUT* sul *Port* del registro dati

     e poi torna al passo 1 (riprendi il ciclo di monitoraggio della periferica)

Funzionamento a controllo di programma – Adattatore

in realtà non è la periferica bensì il suo ***adattatore***, detto anche interfaccia o controllore, che gestisce i registri e l'interazione con la *CPU*

gli adattatori accoppiano i diversi tipi di *CPU* con i diversi tipi di periferica



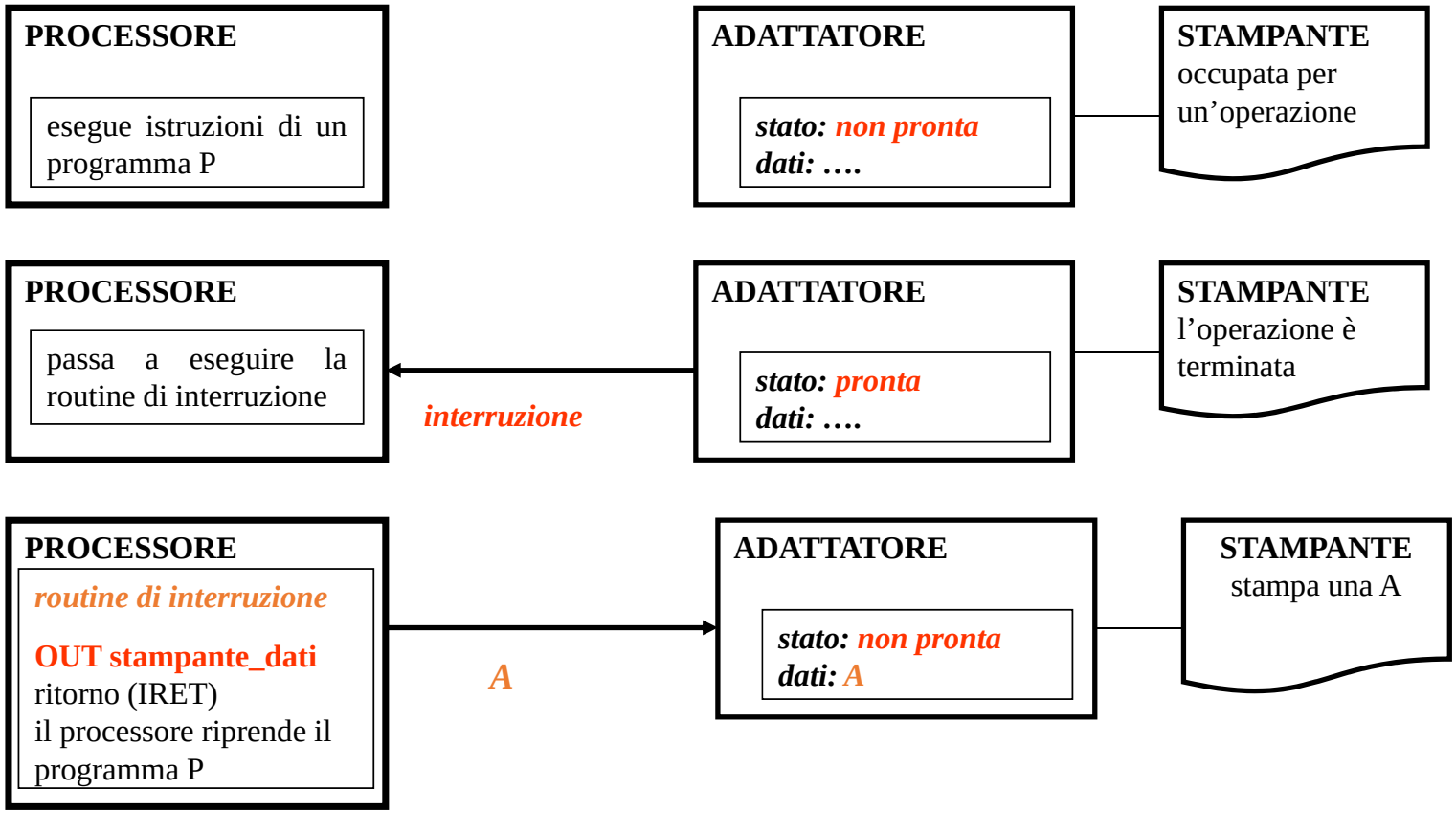
Gestione delle Periferiche tramite meccanismo di Interrupt

- gestire una a controllo di programma ha il difetto di obbligare il processore a restare in un ciclo di attesa (monitoraggio) che la periferica diventi pronta
- invece nel *SO* si vuole che il processore metta il processo in stato di attesa e passi a eseguire altre attività fino a quando la periferica non diventa pronta
- questo obiettivo viene ottenuto utilizzando il meccanismo di *interrupt*, così:

  richiede che la periferica segnali il passaggio da occupata a pronta tramite un apposito *interrupt*

  ossia richiede che la periferica segnali via *interrupt* la transizione del bit *Ready* da 0 a 1

Interruzione – esempio – stampante



Dispositivi di memorizzazione non-volatile

- la memoria di lavoro (*RAM*) di un calcolatore è di tipo *volatile*, cioè perde il suo contenuto quando viene tolta l'alimentazione
- ogni calcolatore deve possedere almeno una memoria di tipo *non-volatile*, dove memorizzare il sistema operativo, i programmi e i dati
- ecco i due tipi principali di memoria non-volatile
  - la memoria non-volatile principale è costituita dai dischi magnetici (*HDD – Hard Disk Drive*)
  - a partire dai primi anni 2000 si sono diffuse sempre più le memorie a stato solido (*SSD – Solid State Drive* o *Solid State Disk*), prima nel settore dei dispositivi mobili e poi nei *PC*
- potrebbero affermarsi a breve anche certe tecnologie alternative, per esempio
  - memoria *RAM* non-volatile (veloce come la *RAM* dinamica volatile)

Volume – definizione logica

- il volume è un modo logico di organizzare la memoria di massa, indipendente dalle diverse strutture geometriche e modalità di accesso fisico all'informazione delle diverse tecnologie di disco *HDD* o *SSD*
- l'indirizzamento dei dati si basa sul concetto di *Logical Block Address* o ***LBA***
- *LBA* è uno schema di indirizzamento secondo cui l'intero disco è rappresentato come un vettore lineare di blocchi, ciascun blocco costituito da un certo numero di byte (generalmente 512 byte o un multiplo)
- si userà il termine ***volume*** per indicare qualsiasi tipo di dispositivo di memorizzazione non-volatile di massa, sia esso *HDD* oppure *SSD*, dotato di uno schema di indirizzamento *LBA*
- il blocco costituisce l'unità fondamentale di informazione da trasferire con una sola operazione tra il disco e la memoria centrale (*RAM*)

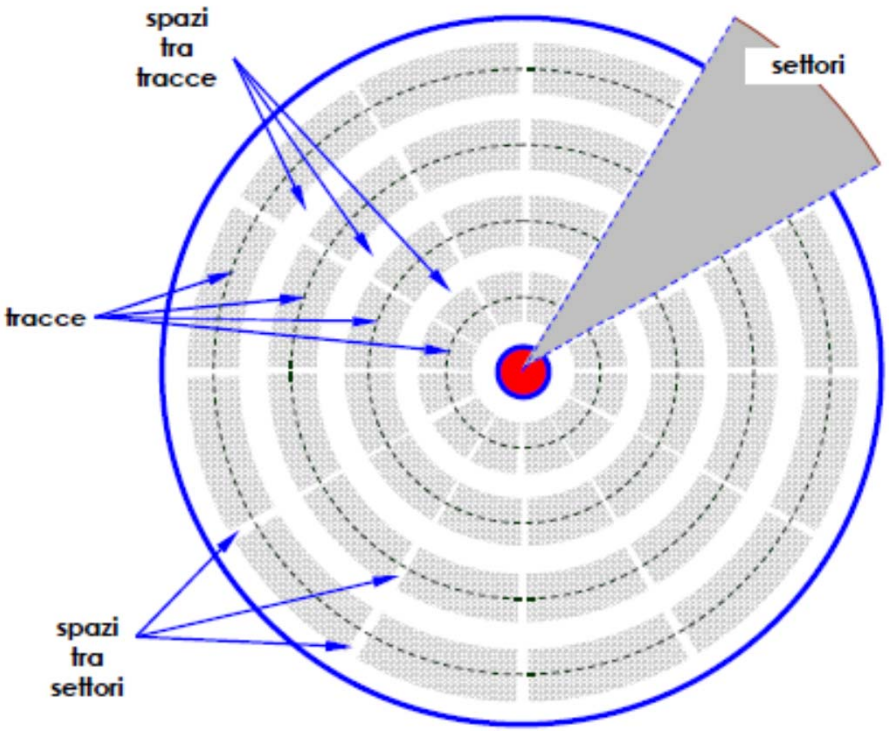
Volume – prestazioni (latenza e trasferimento)

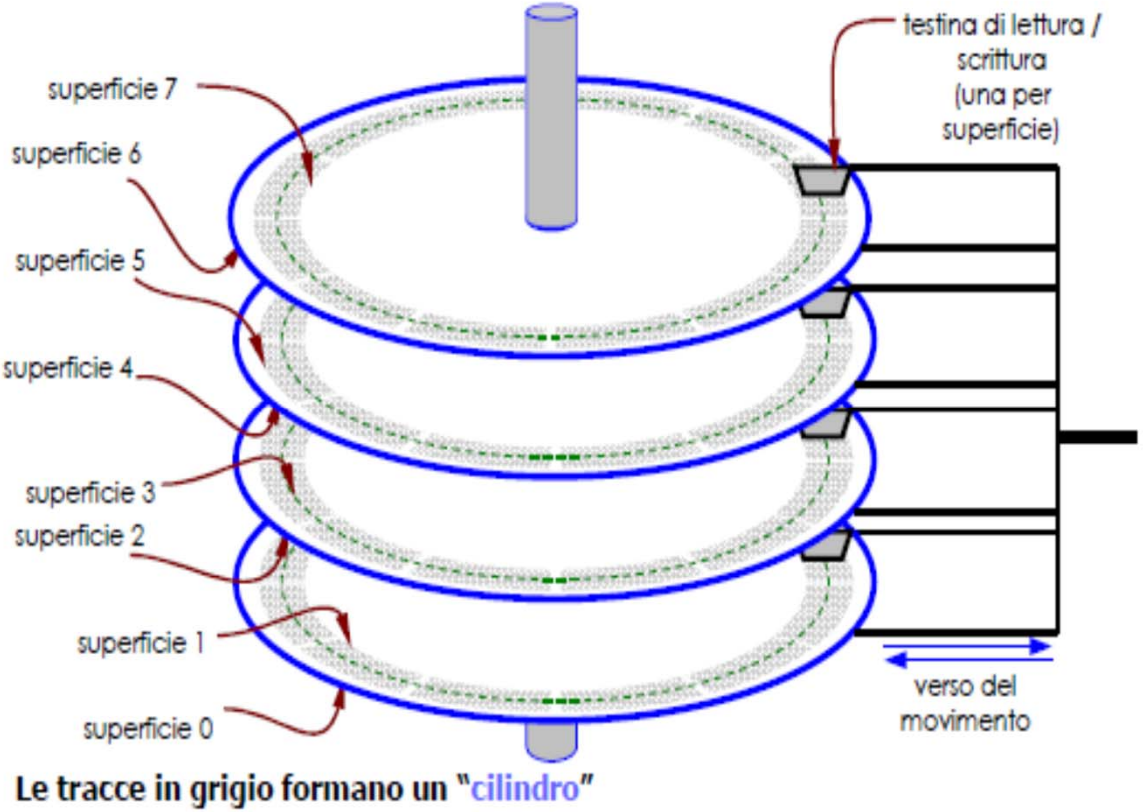
tempi di funzionamento fisico di un disco *HDD*

- il posizionamento delle testina di lettura/scrittura all'inizio di un blocco del volume richiede un tempo molto lungo (il cosiddetto tempo di ***latenza***), nell'ordine delle **decine di *millisecondi* (*ms*)**
- una volta posizionata la testina all'inizio del blocco, il tasso (o velocità) di ***trasferimento*** dei blocchi (in lettura o scrittura) è molto elevato, dell'ordine di **parecchi *Mega byte* al *secondo***

considerazioni analoghe valgono anche per una memoria *SDD*: qui non c'è una testina di lettura/scrittura e nessuna parte del dispositivo è in movimento, ma localizzare un blocco sul circuito integrato (chip) richiede comunque una certa latenza e trasferirlo richiede un certo tempo

Struttura del disco magnetico – *HDD*





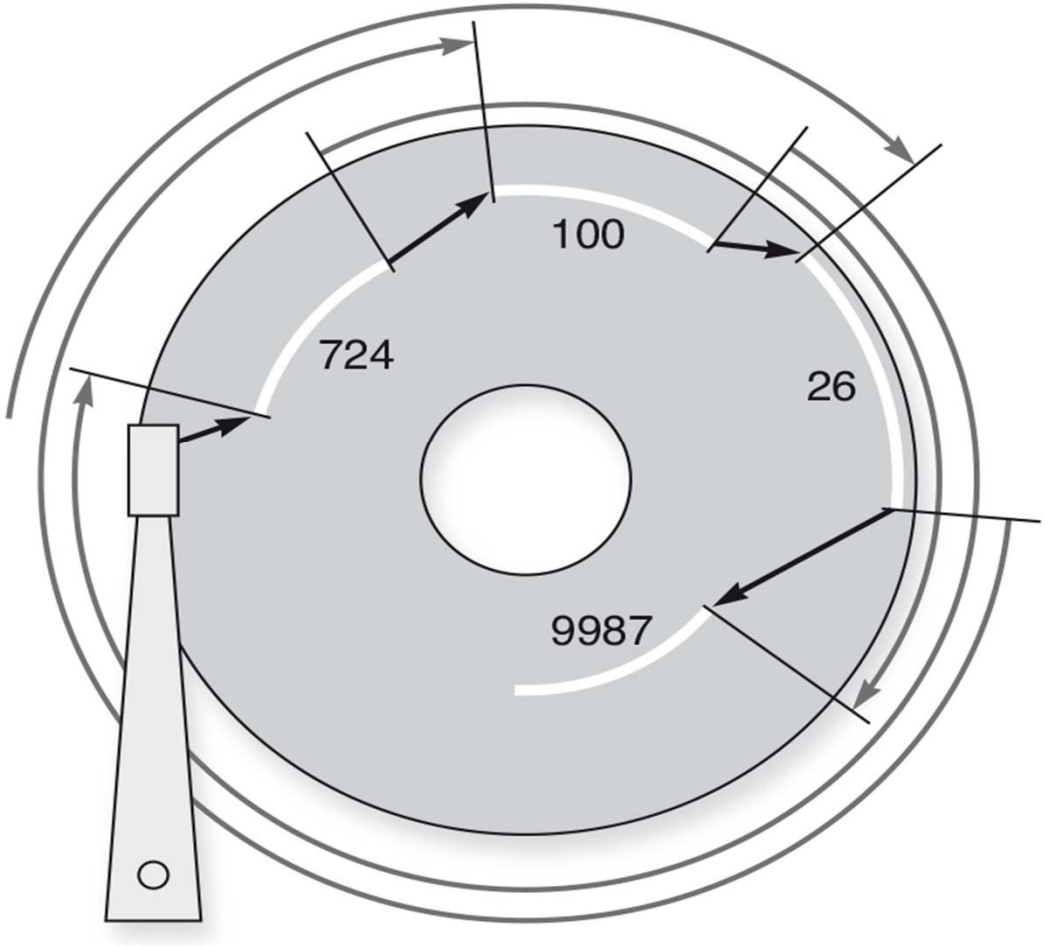
Le tracce in grigio formano un "cilindro"

Importanza dell'ordine di accesso al settore

accesso ai settori 26, 100, 724 e 9987
→ occorre un certo numero di rotazioni del disco (frecce disegnate esternamente)

accesso ai settori 724, 100, 26, 9987
→ basta una sola rotazione del disco, a condizione che gli spostamenti radiali della testina si completino durante gli intervalli di rotazione disponibili

→ quando bisogna accedere a numerosi blocchi di un volume, è possibile e anzi è preferibile scegliere un ordine di accesso ottimale, sulla base del tempo che ciascun accesso richiede (strategia di accesso)



Accesso Diretto a Memoria – *DMA* (*Direct Memory Access*)

- l'idea di fondo della tecnica di *DMA* è che la ***periferica trasferisca*** in modo ***autonomo*** un certo numero di dati in memoria centrale o dalla memoria centrale
- la tecnica di *DMA* è utilizzata da periferiche quali dischi magnetici con interfaccia intelligente (*DMA controller*) capace di trasferire velocemente uno o più settori di disco da o in memoria centrale, senza intervento da parte del processore
- un adattatore (interfaccia) funzionante in *DMA* possiede più registri per contenere i parametri dell'operazione

  l'indirizzo della memoria da dove *iniziare il trasferimento*

  l'indirizzo della periferica da dove *iniziare il trasferimento*

  il *numero di dati* (parole di memoria) *da trasferire*

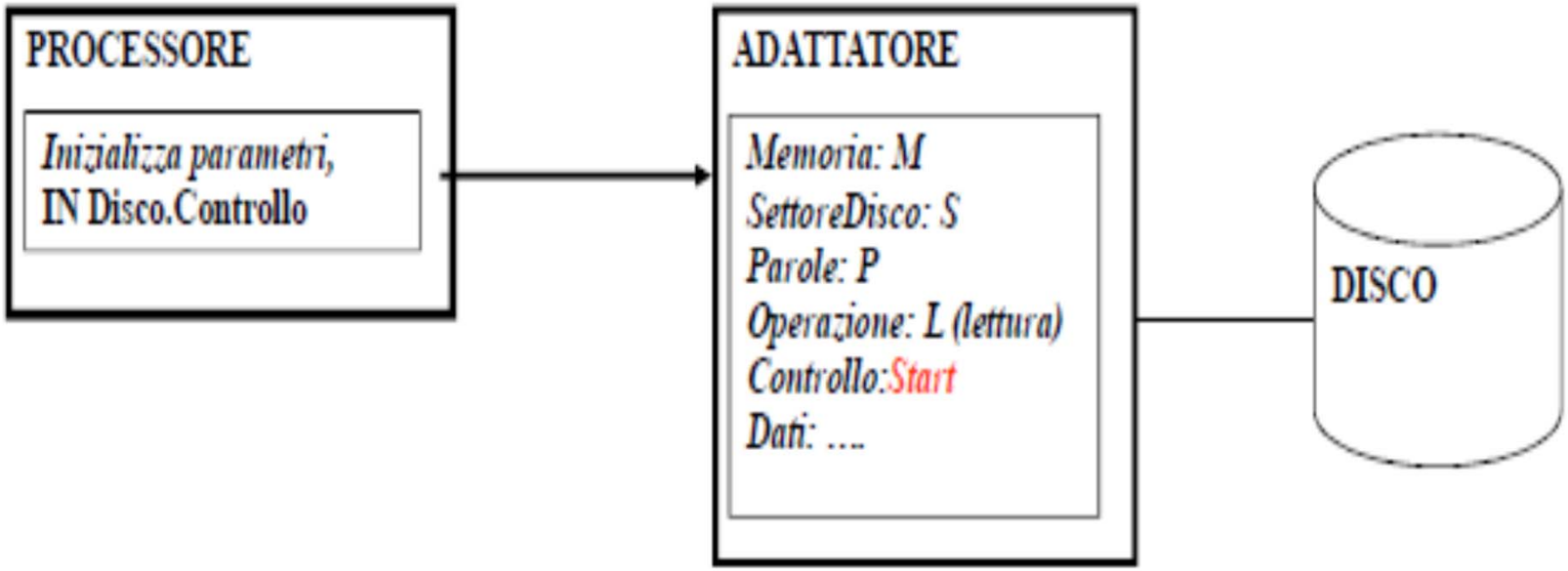
  il *verso del trasferimento* (lettura da periferica o scrittura su periferica)

la tecnica di *DMA* prevede queste due fasi

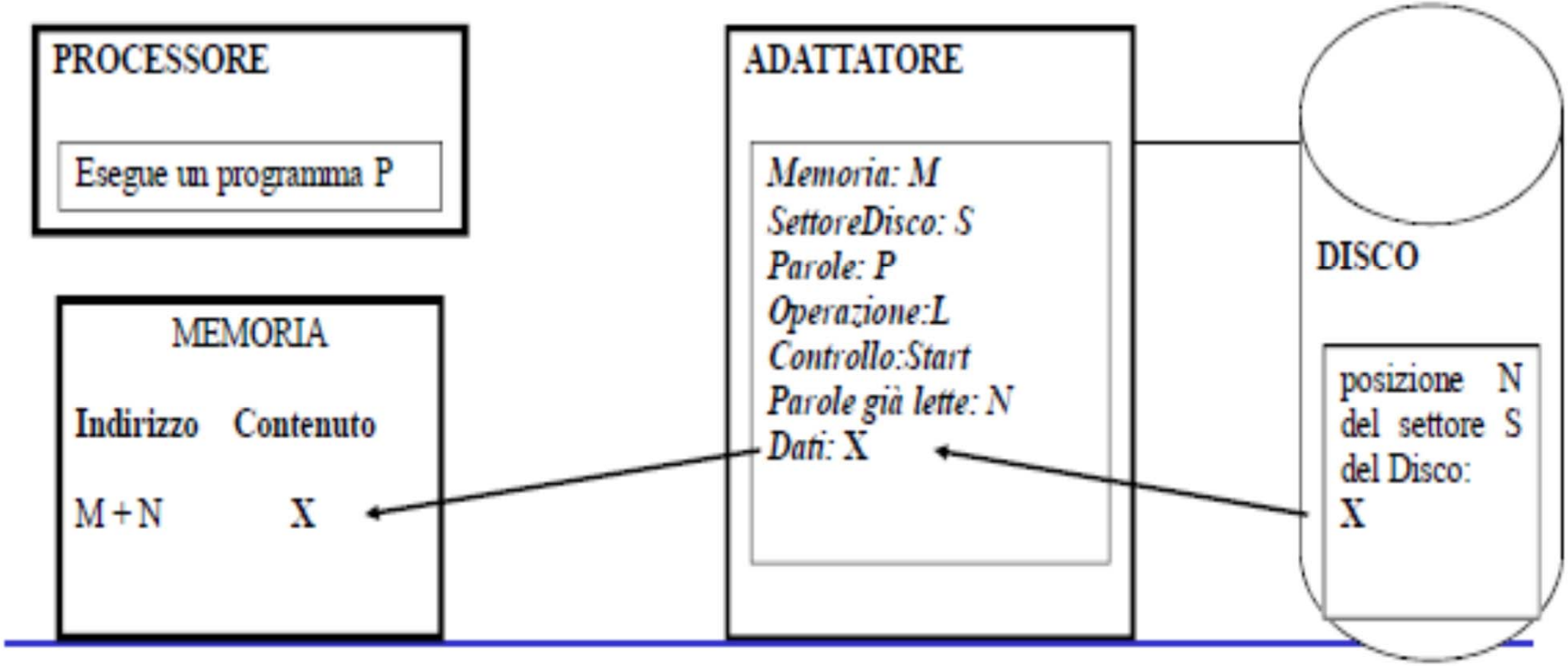
predisposizione: una procedura del *SO* scrive nei registri dell'interfaccia della periferica

attivazione: una procedura di *SO* scrive il comando di avviamento (*start*) nel registro di controllo dell'interfaccia della periferica

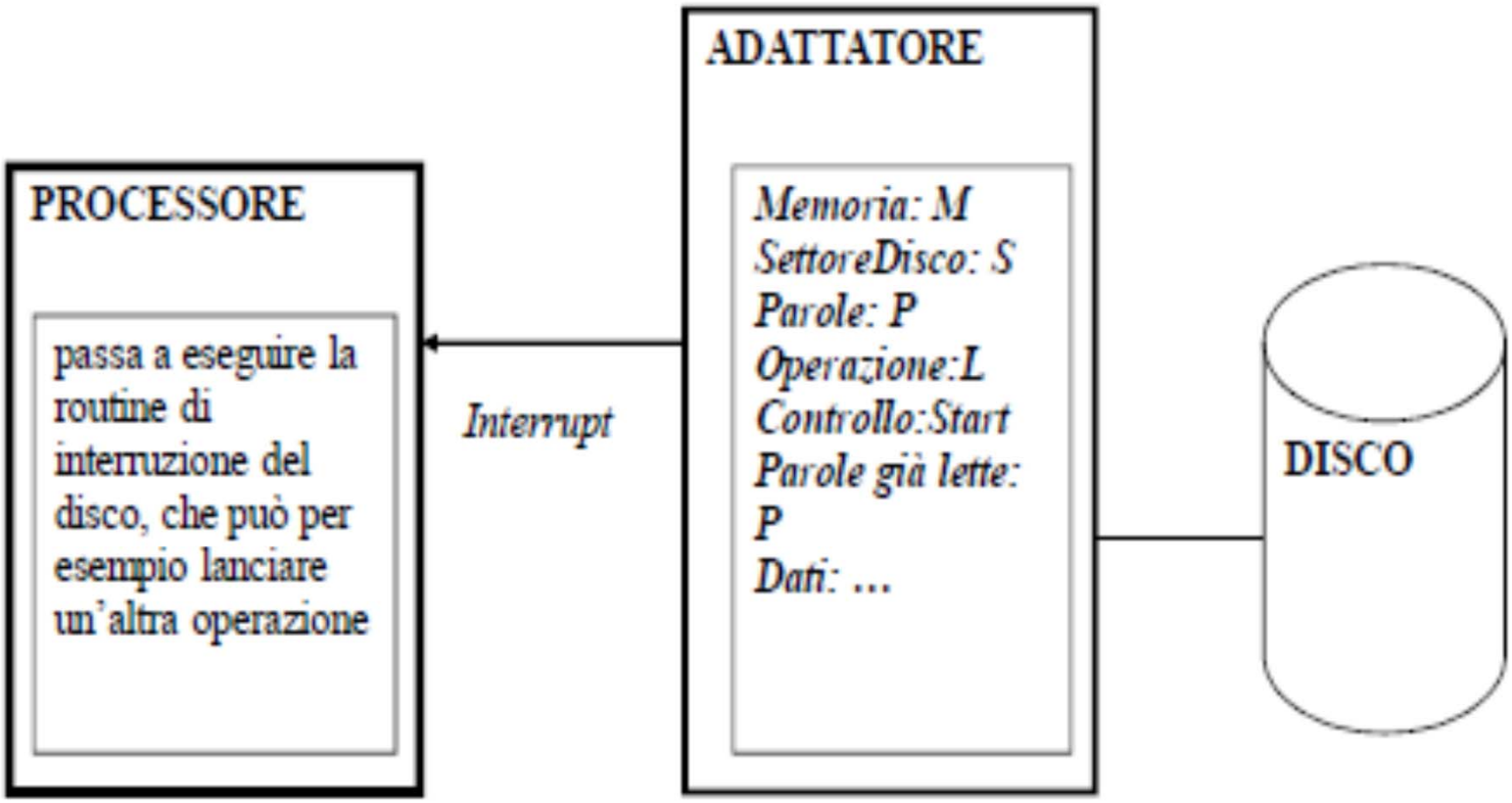
DMA – esempio – inizializzazione



DMA – esempio – fase di lavoro del *DMA*



DMA – esempio – *interrupt* a conclusione del *DMA*



BUS del calcolatore

- il calcolatore elettronico è un insieme di unità funzionali:
  - l'unità centrale di elaborazione (*CPU*), o processore
  - le unità funzionali di memoria, o banchi di memoria
  - le unità funzionali di interfacciamento alle periferiche (adattatori)
- le unità sono interconnesse tramite un organo di collegamento: il ***BUS***
- una ***transazione*** (di *bus*) è un insieme di operazioni sul *bus*, la quale permette di raggiungere un obiettivo, in particolare
  - ***transazione di trasferimento***: trasferisce una certa quantità di informazione tra due unità funzionali
  - ***transazione di interrupt***: permette a una periferica di segnalare un interrupt alla *CPU*
- in genere la quantità di informazione trasferita da una singola transazione di *bus* varia da 8 a 64 bit

BUS interni ed esterni

- i *BUS* del calcolatore si distinguono in due tipologie

  BUS interni, confinati all'interno di una singola unità funzionale, i quali collegano i blocchi logici (combinatori e sequenziali) contenuti nell'unità, e che solitamente non sono standardizzati

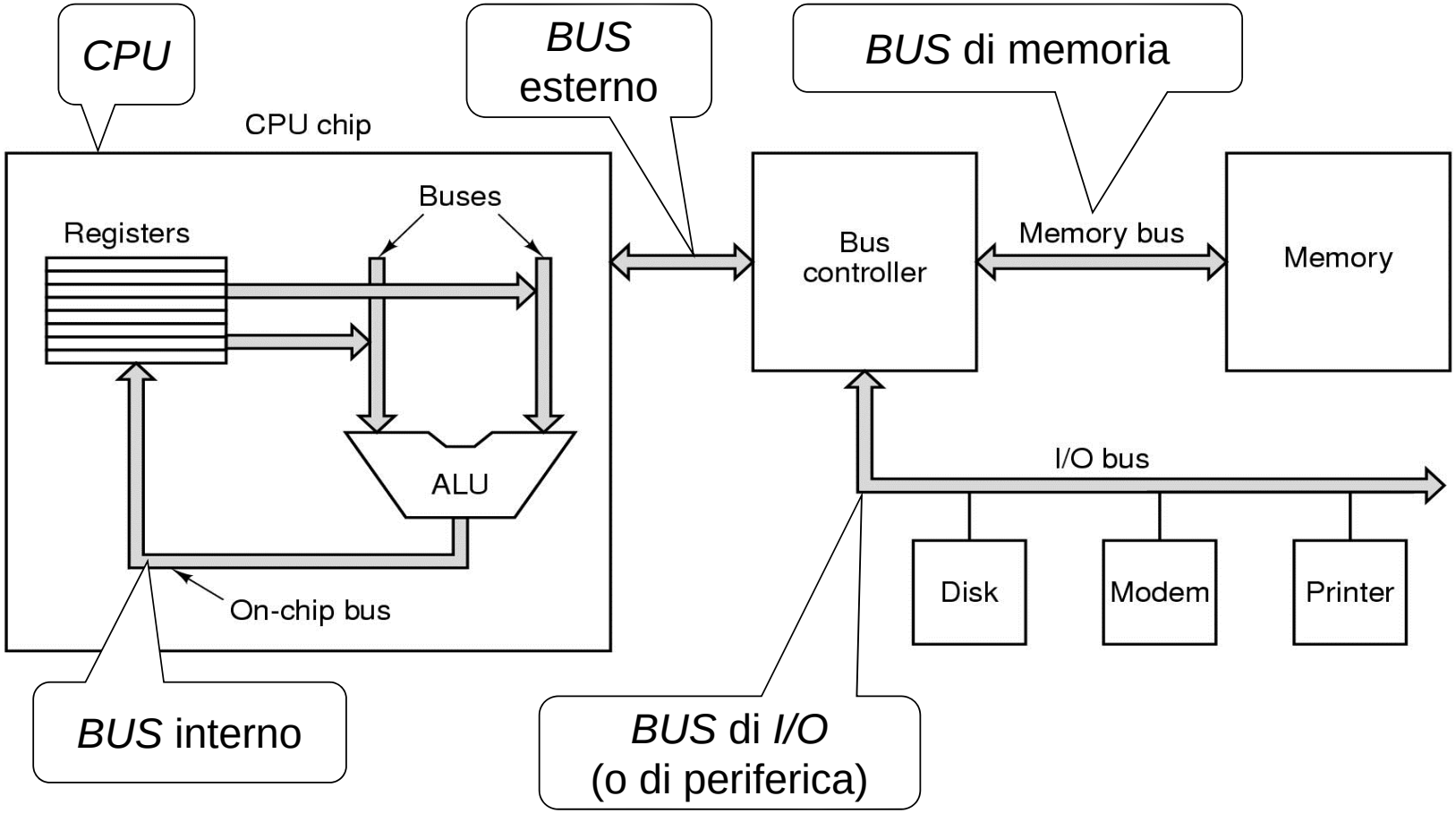
  BUS esterni, che collegano le unità funzionali, e che solitamente sono standardizzati

- i calcolatori più semplici sono dotati di un solo *BUS* esterno (generalmente chiamato *BUS* di sistema), che collega *CPU*, memoria e unità di *I/O*
- la maggior parte dei calcolatori odierni è dotata di numerosi *BUS* esterni, in particolare di questi due

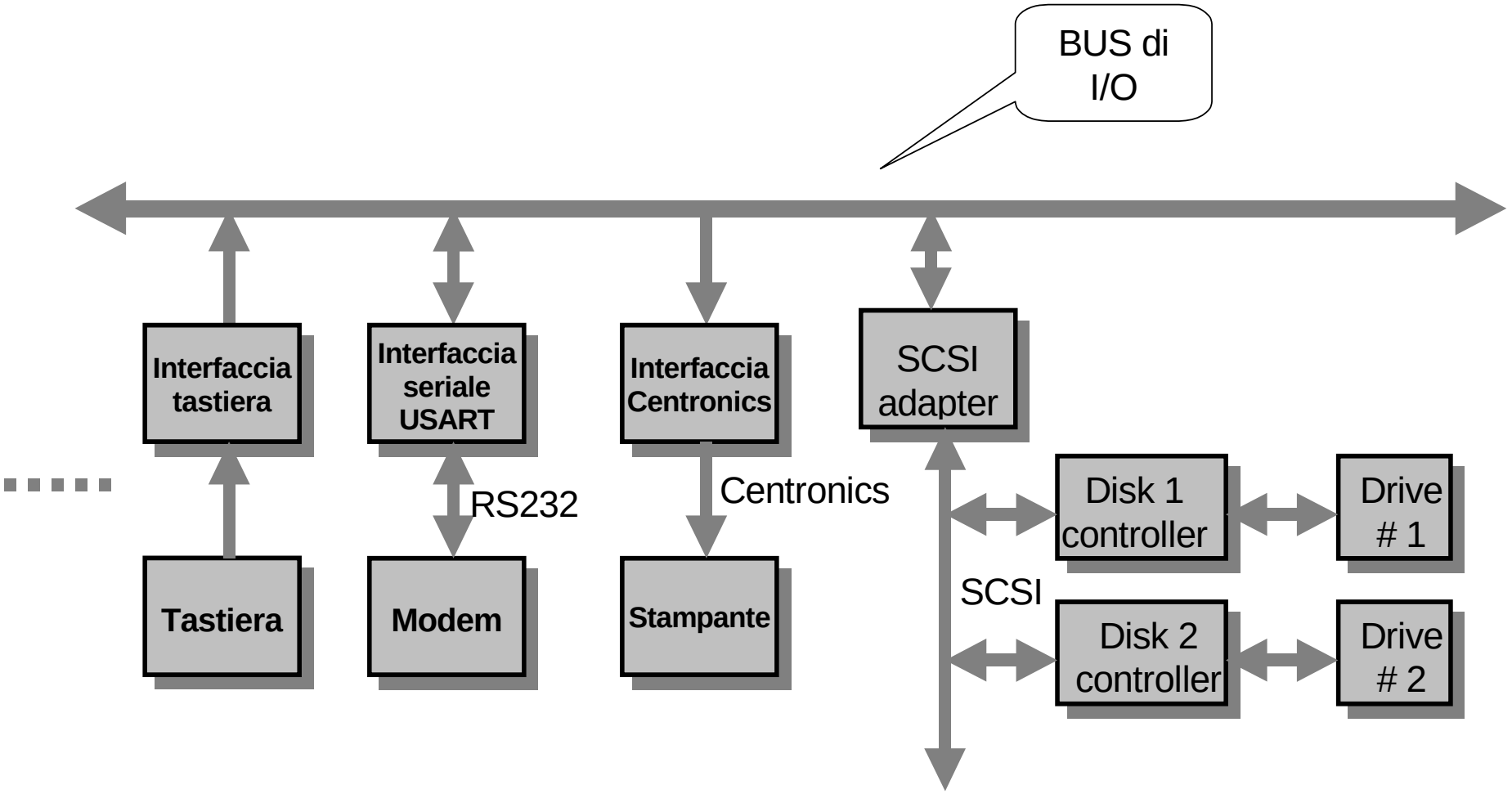
  ***BUS* di memoria**, che collega la *CPU* e le unità funzionali di memoria (banchi di memoria)

  BUS* di *I/O, che collega la *CPU* e le unità funzionali di *I/O*

Sistema con diversi *BUS*



Esempi di adattatori (interfacce) del *BUS* di *I/O*

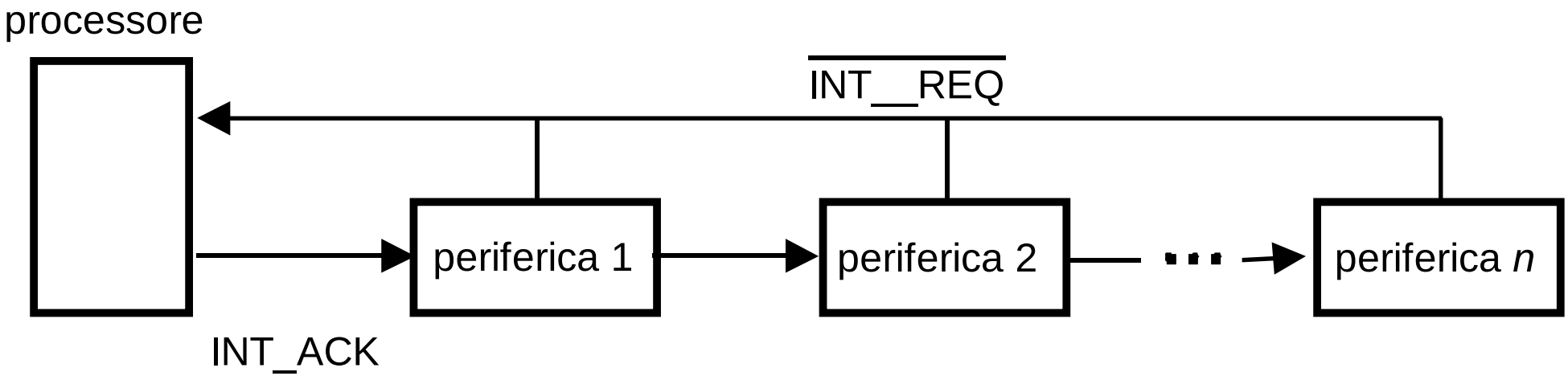


Transazione di *interrupt*

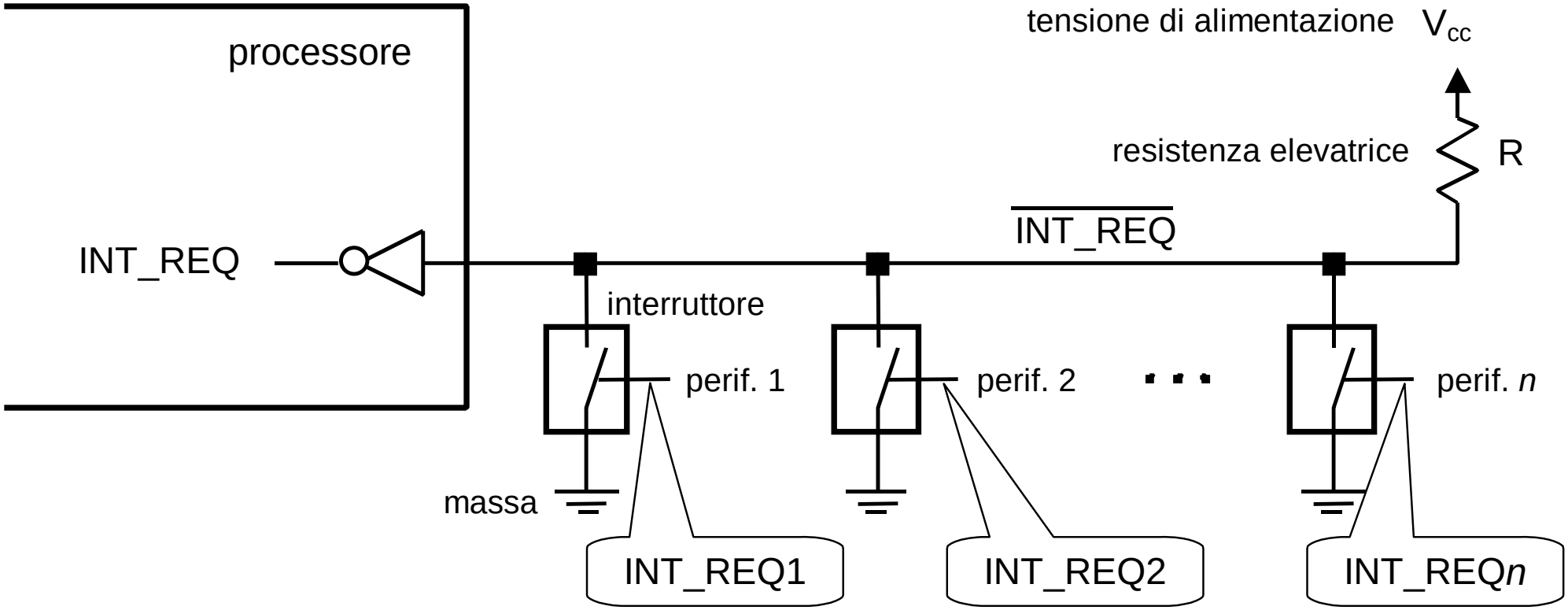
una ***transazione di interrupt*** è l'insieme di operazioni che porta da una segnalazione di *interrupt* da parte di una periferica alla presa in carico del *servizio di interrupt* da parte della *CPU*

ci sono vari modi di collegare le linee di interrupt delle periferiche alla *CPU*, come

INT_REQ richiesta di interrupt in ***wired_or***
INT_ACK accettazione dell'interrupt in ***daisy* chain** (*festone*)



Collegamento *WIRED NOR* di una linea di *INT_REQ*



Transazione di trasferimento

- generalmente una ***transazione di trasferimento*** si può scomporre in due fasi principali, ciascuna costituita da diverse operazioni

  fase di ***arbitraggio***: serve a selezionare un'unità, detta ***MASTER***, che controlla il bus durante l'operazione

  fase di ***trasferimento*** vero e proprio, durante la quale avvengono le operazioni seguenti

  il *MASTER* seleziona un'altra unità, detta ***SLAVE***, con cui operare

  il *MASTER* indica la direzione del trasferimento

  lettura: dallo *SLAVE* verso il *MASTER*

  scrittura: dal *MASTER* verso lo *SLAVE*

  ed effettua il vero e proprio trasferimento di unità di informazione tra *MASTER* e *SLAVE*

- nota bene: le unità che svolgono il ruolo di *MASTER* e di *SLAVE* sono fissate per ogni singola operazione di trasferimento del *bus*, ma possono variare tra operazioni diverse (cessione del ruolo di *MASTER*)

Unità funzionali che possono diventare *MASTER*

- nei sistemi multiprocessore le varie *CPU* possono tutte diventare *MASTER*
- tuttavia in determinate circostanze anche altre unità funzionali possono assumere il ruolo di *MASTER*, più o meno temporaneamente per scopi particolari

  adattatori in *DMA*: possono diventare *MASTER* per trasferire dati direttamente con la memoria, senza bisogno della *CPU* (*DMA*)

  co-processore: può diventare *MASTER* per prelevare operandi dalla memoria

Master	Slave	Esempio
CPU	Memoria	Prelievo istruzioni e lettura/scrittura dati
CPU	Unità di I/O	Ricezione/invio dati da/a un'unità di I/O
CPU	Coprocessore	La CPU dà istruzioni al coprocessore
I/O	Memoria	Accesso diretto alla memoria (DMA)*
Coprocessore	CPU	Il coprocessore legge operandi dalla CPU

Esempio: arbitraggio centralizzato in *Daisy Chain*

- fase di arbitraggio per il ***prossimo MASTER*** sovrapposta alla fase di trasferimento controllata dal ***MASTER corrente***
- questo meccanismo di arbitraggio prevede

  un'unità funzionale apposita, che svolge la funzione di ***arbitro del BUS***

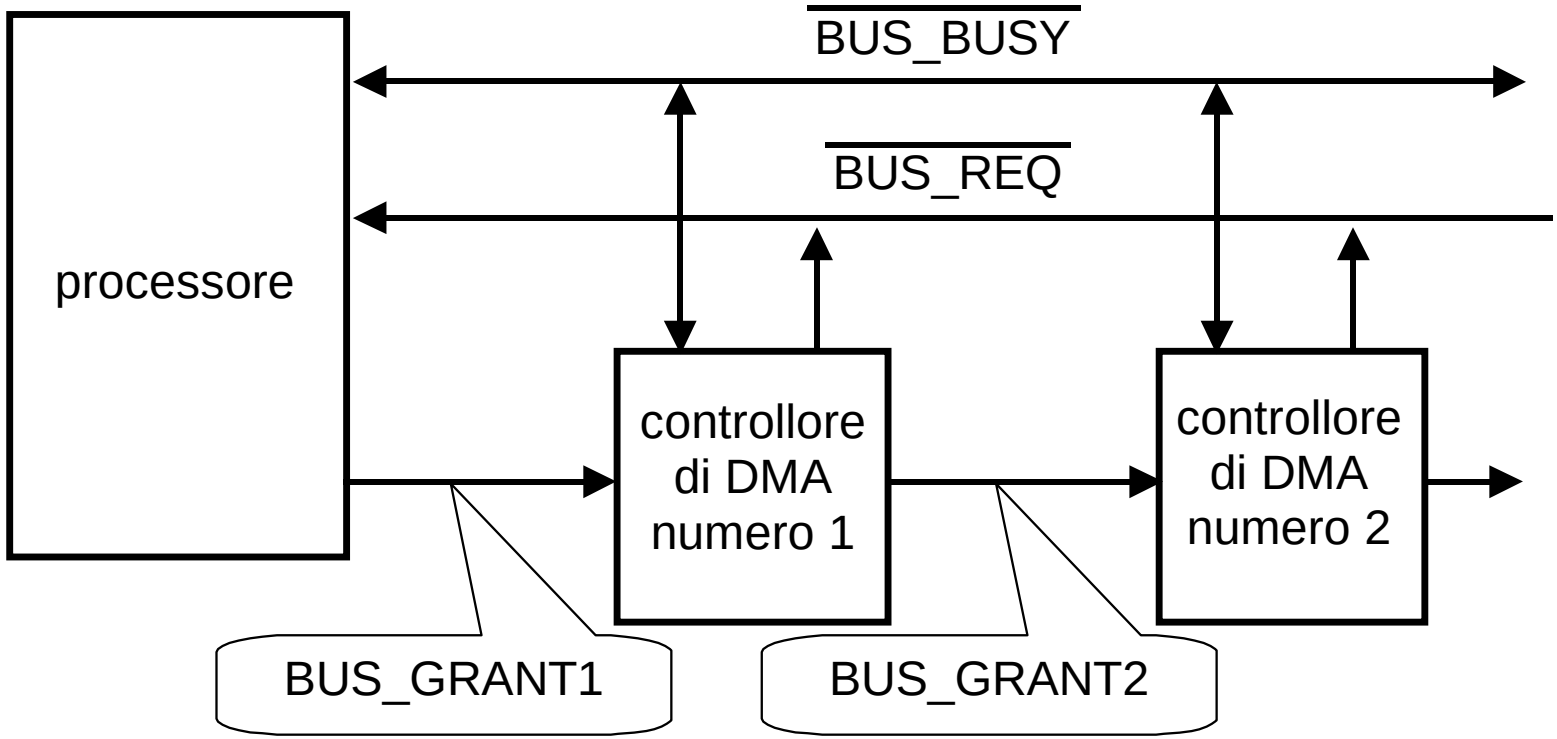
  talvolta la funzione di arbitro è svolta dalla *CPU*

  linee che collegano l'arbitro alle unità funzionali potenziali richiedenti il controllo del *BUS*

Bus Request	richiesta di cessione del controllo in *wired or*
Bus Grant	conferma di cessione del controllo in *daisy chain*
Bus Busy	indicazione che esiste un *MASTER* corrente che sta eseguendo un trasferimento in *wired or*

- quando l'arbitraggio termina determinando il prossimo *MASTER*, questo deve attendere la fine del trasferimento corrente prima di iniziare la propria fase di trasferimento

Esempio: arbitraggio tra *CPU* (arbitro) e diversi *DMA*



Sincronizzazione delle operazioni

- tutti i tipi di operazione visti richiedono di adottare certi schemi di ***sincronizzazione***
- tali schemi sono resi complicati da numerosi problemi
  - le velocità di risposta delle varie unità differiscono
  - le complessità delle operazioni che le varie unità sono in grado di svolgere differiscono
  - i segnali di controllo del *bus* richiedono tempo per propagarsi da un'unità a un'altra
  - inoltre questo tempo può variare in base alla distanza (fisica e strutturale) tra le unità coinvolte
  - anche i segnali che partono nello stesso istante di tempo da una medesima sorgente possono arrivare a destinazione sfasati tra loro
- per tutti questi motivi ci sono numerosi e svariati metodi di sincronizzazione, che si differenziano in base a questi elementi
  - i requisiti di velocità e di complessità logica che impongono alle unità
  - il numero di linee di *BUS* che richiedono
  - la velocità raggiungibile nelle diverse transazioni
- qui questo argomento non verrà trattato