La gestione dei dispositivi di Input/Output

Matteo Sonza Reorda

Politecnico di Torino
Dip. di Automatica e Informatica

Introduzione

Una delle principali funzioni svolte dai sistemi di elaborazione è quella di interagire e di scambiare informazioni con il mondo esterno.

Tale funzione viene svolta attraverso il sotto-sistema di I/O.

Sotto-sistema di I/O

Il sotto-sistema di I/O di un calcolatore è composto da:

- **dispositivi di Input/Output (periferiche)**
- **unità per il controllo di questi dispositivi**
- **software per la loro gestione.**

Esempio

Si consideri la connessione di una *stampante*, realizzata normalmente attraverso un'interfaccia parallela.

La stampante sarà vista dalla CPU come 3 registri corrispondenti ad altrettanti indirizzi:

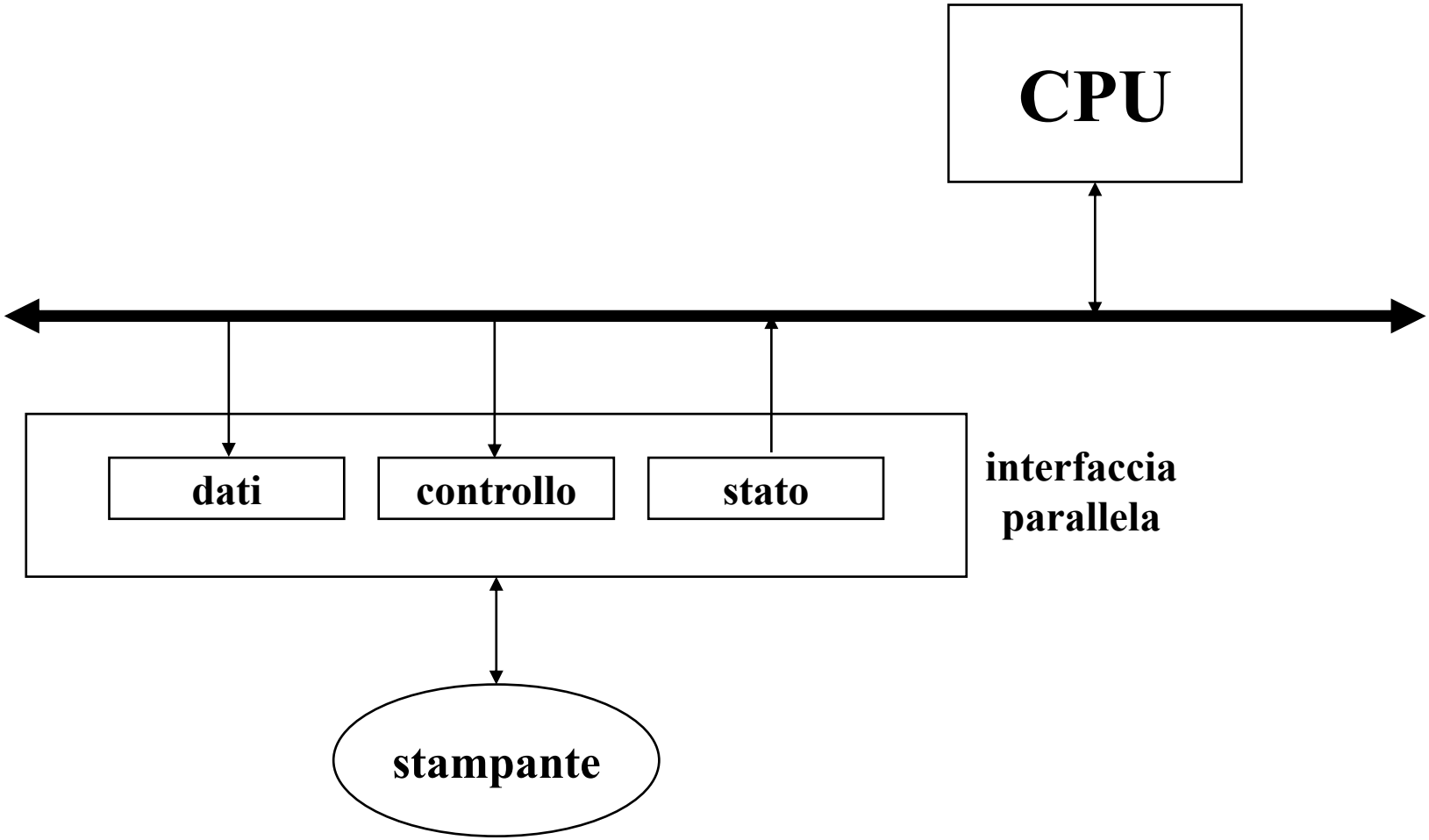
- **un registro per i *dati* (output)**
- **un registro per le informazioni di *controllo* (output)**
- **un registro per le informazioni di *stato* (input).**

Per far eseguire delle operazioni alla stampante la CPU eseguirà delle scritture o letture sui 3 registri.

Indirizzamento dei dispositivi di I/O

A ciascun dispositivo di I/O è di solito associata un'interfaccia, che lo collega al bus del sistema.

Esempio


CPU
dati
controllo
stato
interfaccia
parallela
stampante

Porte

La comunicazione tra il processore (o la memoria) e il dispositivo avviene attraverso i registri (o *porte*).

Questi sono accessibili tramite il bus di sistema, al pari delle celle di memoria, in quanto a ciascuno di essi è associato un indirizzo.

La loro connessione al bus può avvenire secondo 2 modalità:

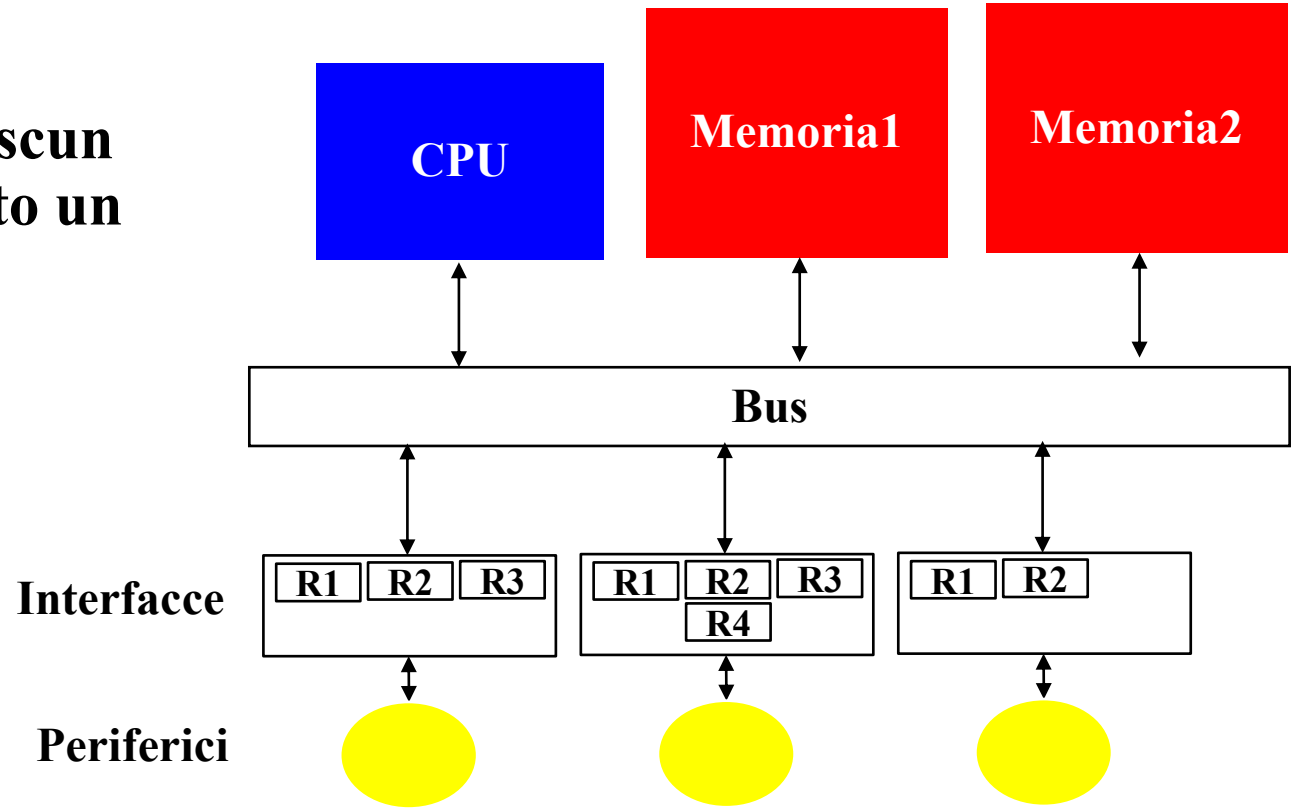
- ***memory-mapped* I/O**
- ***isolated* I/O (o *I/O-mapped* I/O).**

Porte

La comunicazione tra il processore e il dispositivo avviene attraverso i registri (o *porte*) presenti nelle relative interfacce.

Queste sono accessibili tramite il bus di sistema.

Come per le celle di memoria, a ciascun registro è associato un indirizzo.



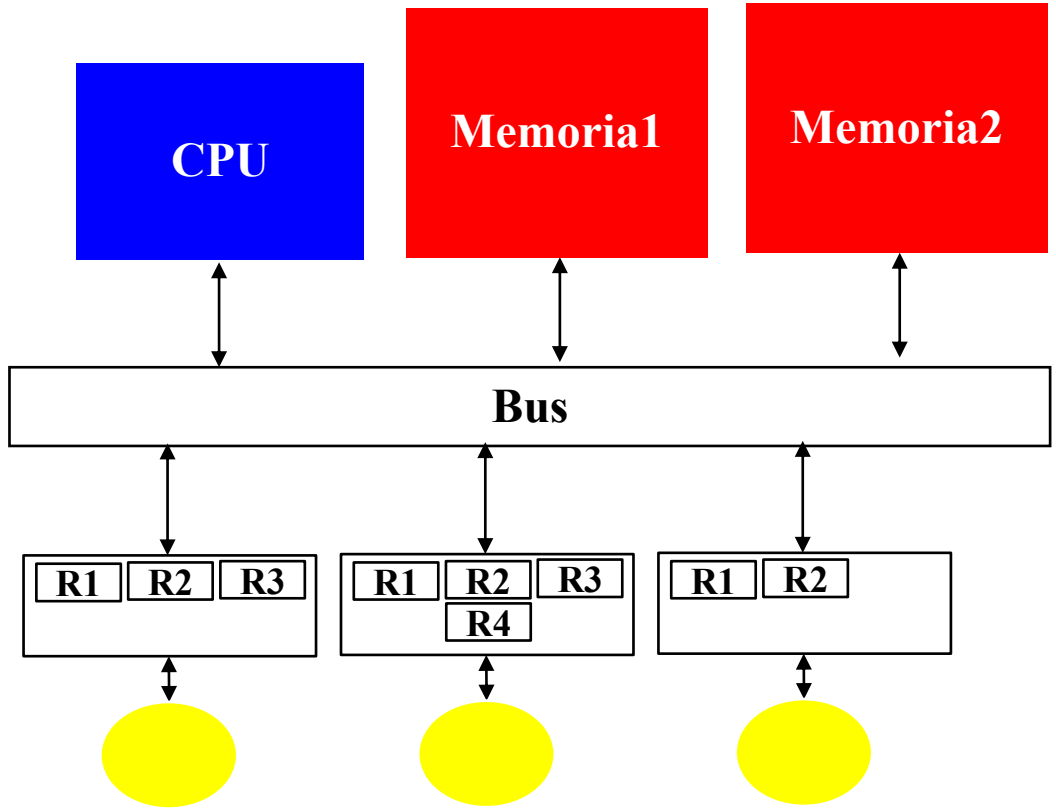
Porte

La comunicazione tra il processore e il dispositivo avviene attraverso i registri (o *porte*) presenti nelle relative interfacce.

Queste sono accessibili tramite il bus di sistema.

Come per le celle di memoria, a ciascun registro è associato un indirizzo.
La connessione dei periferici al bus può avvenire secondo 2 modalità:

- ***memory-mapped* I/O**
- ***isolated* I/O.**

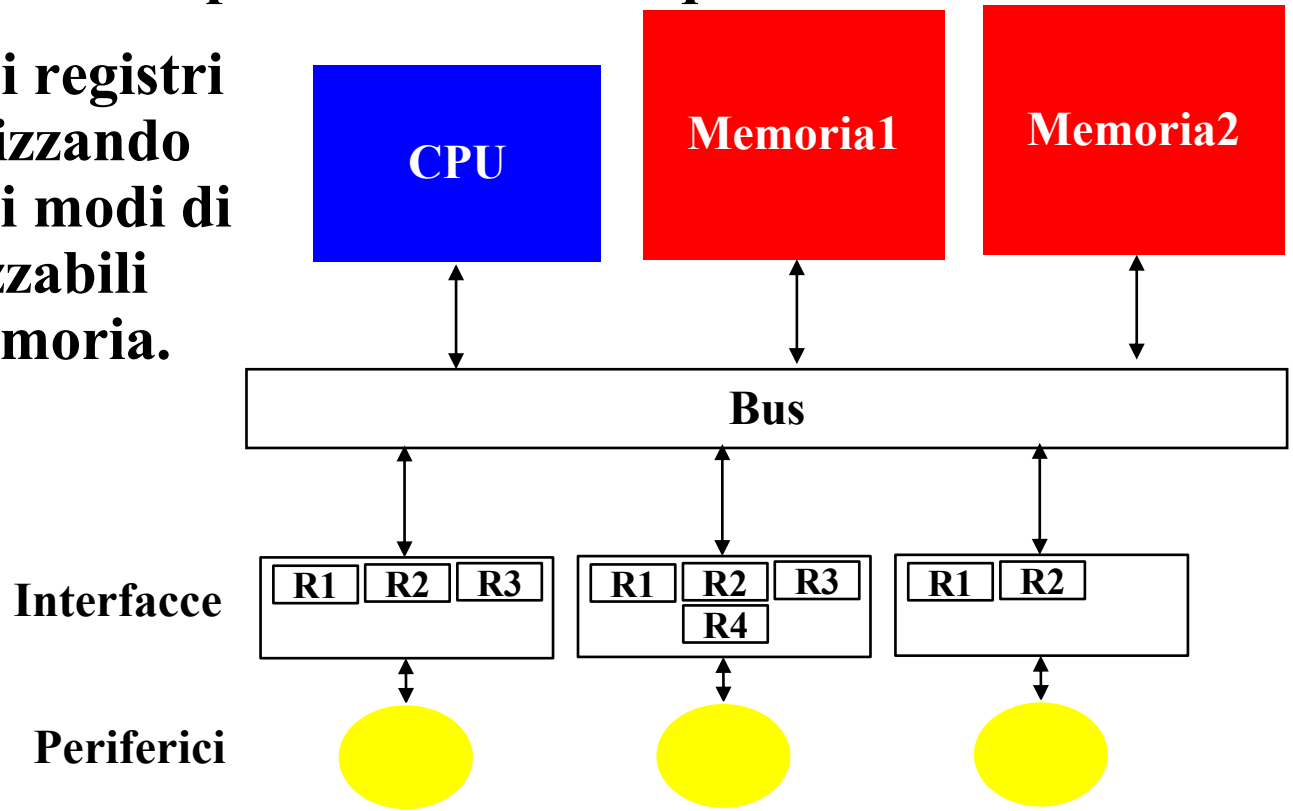


Memory-Mapped I/O

I registri dei dispositivi di I/O sono connessi come le normali celle di memoria.

Lo spazio di indirizzamento per la memoria è quindi ridotto.

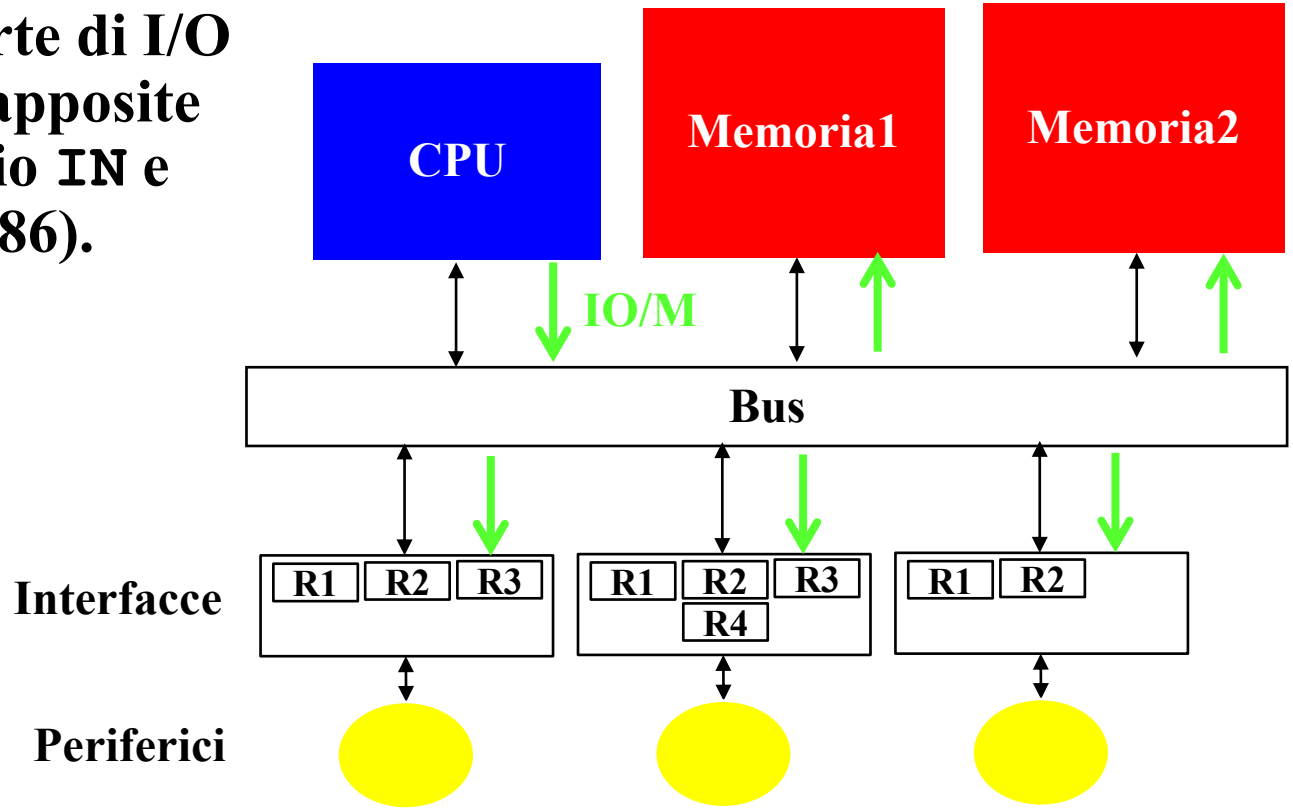
Si può fare accesso ai registri delle periferiche utilizzando tutte le istruzioni ed i modi di indirizzamento utilizzabili per accedere alla memoria.



Isolated I/O

Gli spazi di indirizzamento per la memoria e per le porte di I/O sono separati, e sono attivati alternativamente da appositi segnali (ad esempio IO/M nei processori x86).

Per accedere alle porte di I/O si devono utilizzare apposite istruzioni (ad esempio `IN` e `OUT` nei processori x86).



Sincronizzazione

Le velocità di operazione della CPU e dei vari dispositivi periferici possono essere molto diverse.

Il processore deve quindi sincronizzarsi con i dispositivi di I/O per eseguire le operazioni che li interessano.

La sincronizzazione può essere svolta in vari modi.

Meccanismi di gestione dell'I/O

Si differenziano per il diverso livello di coinvolgimento della CPU nella gestione dei dispositivi di I/O.

Possibili soluzioni sono:

- **I/O programmato**
- **Interrupt**
- **Direct Memory Access (DMA).**

I/O programmato

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 a 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.

La CPU deve spostare un nuovo dato al/dal periferico non appena questo è pronto. La disponibilità del dispositivo a procedere è segnalata da un bit del registro di stato.

Poiché l'I/O programmato è basato sulla ripetizione di un test sul registro di stato per verificare quando il programma può procedere oltre, è anche noto come *polling*.

Esempio

Per eseguire il trasferimento ad una porta parallela di un blocco di dati utilizzando l'I/O programmato, la CPU deve eseguire un frammento di codice quale il seguente:

```
Per ogni parola del blocco da trasferire
   1. Verifica se la porta è pronta
      leggendo il registro di stato
   2. Se è pronta, scrivi una parola sul
      registro di dato
   3. Se non è pronta, torna al punto 1
```

Caratteristiche

L'I/O programmato viene normalmente utilizzato nei sistemi più piccoli e meno complessi, in quanto ha le seguenti caratteristiche:

- \+ **poco costoso in termini di hardware**
- \- **poco efficiente.**

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 a eseguire banali operazioni di test e trasferimento dati.**

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.**

Interrupt

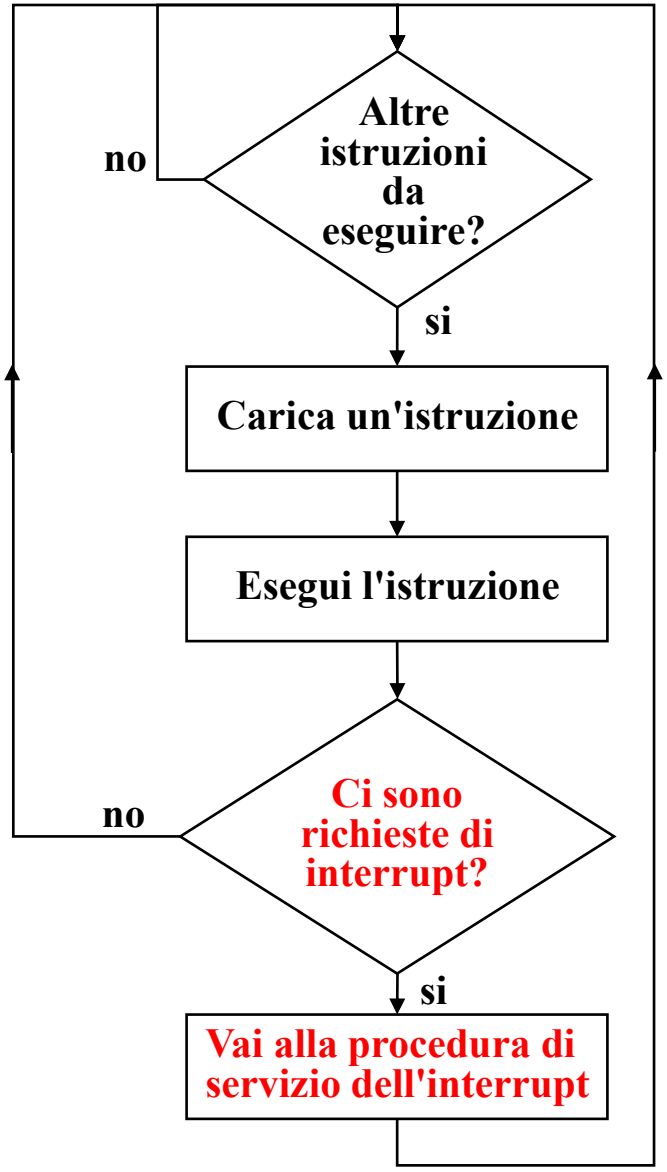
La gestione dei dispositivi di I/O tramite *interrupt* permette di aumentare l'efficienza della CPU.

Un segnale di input esterno *asincrono* informa il microprocessore che un dispositivo richiede di essere servito.

All'arrivo di una richiesta di interrupt la CPU

- **interrompe l'esecuzione del programma corrente**
- **salta all'esecuzione di una *procedura di servizio* dell'interruzione (*Interrupt Service Routine - ISR*).**

Sincronizzazione



Controllore dell'interrupt

Il processore ha spesso un unico segnale su cui attivare richieste di interrupt.

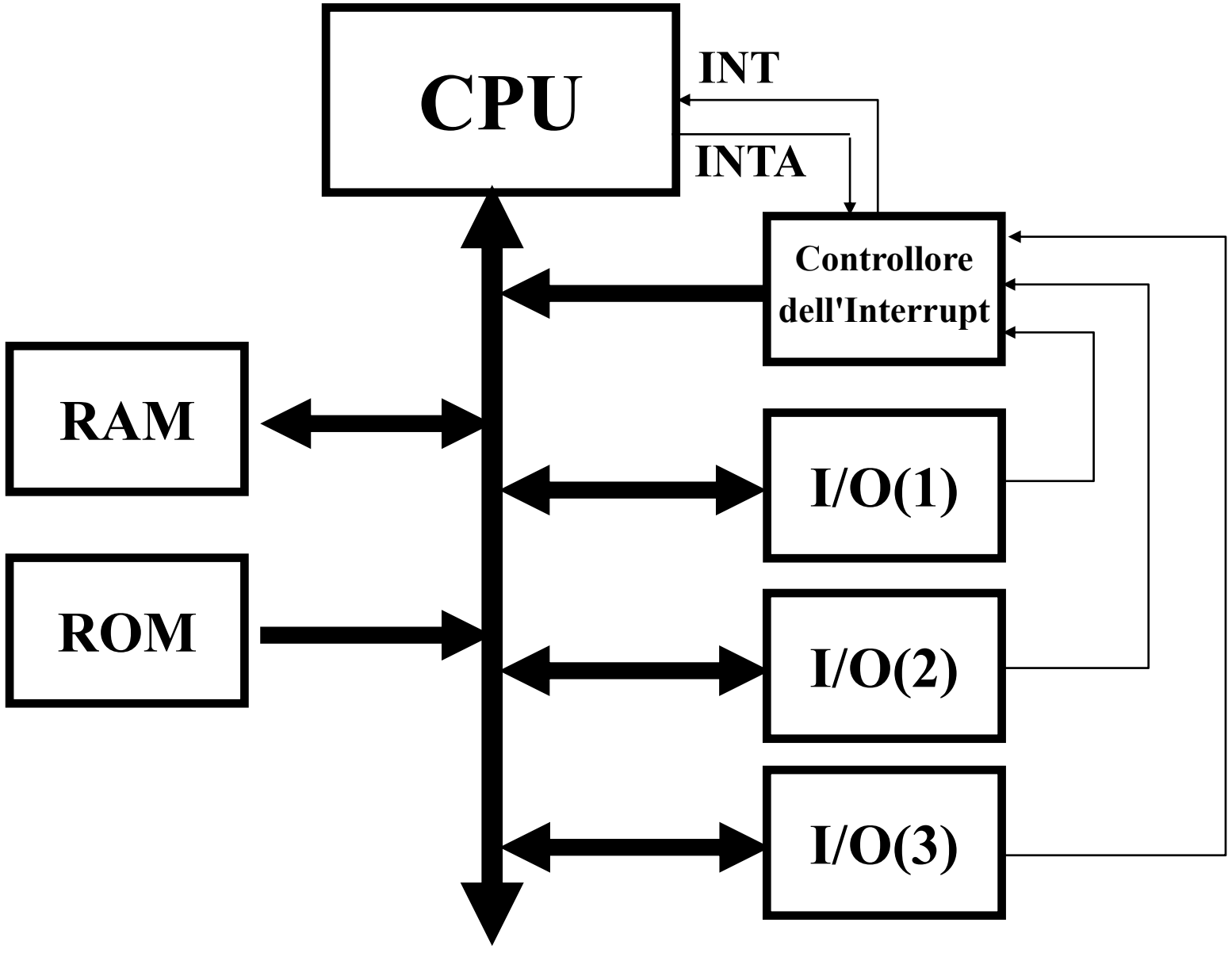
Le periferiche che possono richiedere l'interrupt sono di solito più di una.

È quindi necessario un modulo (denominato *controllore dell'interrupt* o Interrupt Controller o IC) che

- **riceva tutte le richieste**
- **piloti il segnale che va al processore.**

Il controllore dell'interrupt funziona come un gestore delle richieste di interruzione e si interpone tra i dispositivi periferici e la CPU.

Interrupt


CPU
INT
INTA
Controllore dell'Interrupt
RAM
ROM
I/O(1)
I/O(2)
I/O(3)
Richieste di interrupt

Procedure di servizio dell'interrupt

Differiscono dalle normali procedure in quanto la loro attivazione interrompe un programma a priori non noto.

Di conseguenza, al loro termine tale programma deve poter continuare la propria attività come se non fosse stato interrotto.

Per questo all'attivazione di una procedura di servizio dell'interrupt vengono salvati (a volte nello stack, a volte in appositi registri):

- **il PC e la parola di stato (via hardware)**
- **eventuali registri o parole di memoria utilizzate (via software).**

Al termine della procedura si deve

- **ripristinare il valore di registri e parola di stato**
- **ritornare al programma interrotto.**

Latenza di interrupt

È il tempo tra la richiesta di interrupt e la partenza della relativa procedura di servizio.

In talune applicazioni (ad esempio quelle per le elaborazioni in tempo reale) deve essere estremamente ridotta.

Disabilitazione degli interrupt

In alcuni casi è necessario evitare che una richiesta di interrupt interrompa il processore, ad esempio perchè questo sta eseguendo un'operazione essenziale (spesso corrispondente ad una procedura di servizio dell'interrupt).

La soluzione di solito prevede che

- **vi sia un apposito bit di abilitazione degli interrupt nel registro di stato del processore**
- **il bit venga automaticamente settato quando il processore entra nella procedura di servizio**
- **il bit venga nuovamente resettato al termine della procedura di servizio.**

Identificazione del dispositivo

Poiché più di un periferico può attivare una richiesta di interrupt, è necessario un meccanismo per l'individuazione del periferico, in modo da poter poi attivare la relativa procedura di servizio.

Esistono varie soluzioni:

- **Linee di interrupt multiple**
- **Polling**
- **Interrupt vettorizzato.**

Linee di interrupt multiple

La CPU possiede diversi segnali per le richieste di interrupt. Ogni periferica è collegata ad un diverso segnale.

È una soluzione che da sola non risolve il problema, poiché il numero di periferiche è quasi sempre superiore a quello dei segnali disponibili.

Polling

Esiste un solo segnale per le richieste di interrupt.

Quando la CPU percepisce la richiesta, inizia a scandire le parole di stato di tutte le periferiche (o dell'IC) per individuare quella che ha fatto la richiesta.

La latenza di interrupt in questo modo risulta normalmente troppo elevata.

Interrupt vettorizzato

È il meccanismo più utilizzato.

Si compone dei seguenti passi:

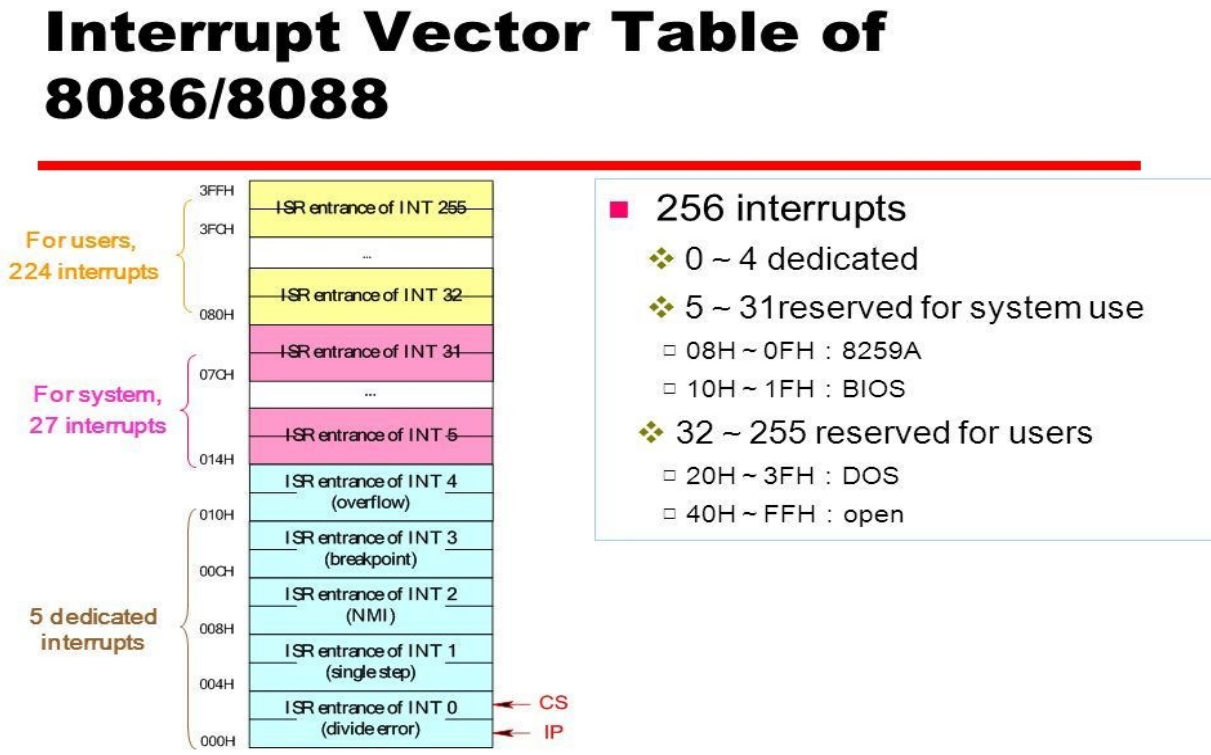
1 **Quando il processore è pronto a servire la richiesta di interrupt, invia un segnale di *Interrupt Acknowledge***

2 **Il periferico che ha fatto la richiesta (o l'Interrupt Controller) pone sul bus dati un codice di identificazione**

3 **Il codice viene utilizzato dal processore per determinare l'indirizzo della procedura di servizio, usandolo come indice per accedere ad un vettore contenente gli indirizzi delle procedure di servizio (*Interrupt Vector Table* o *IVT*).**

Interrupt Vector Table

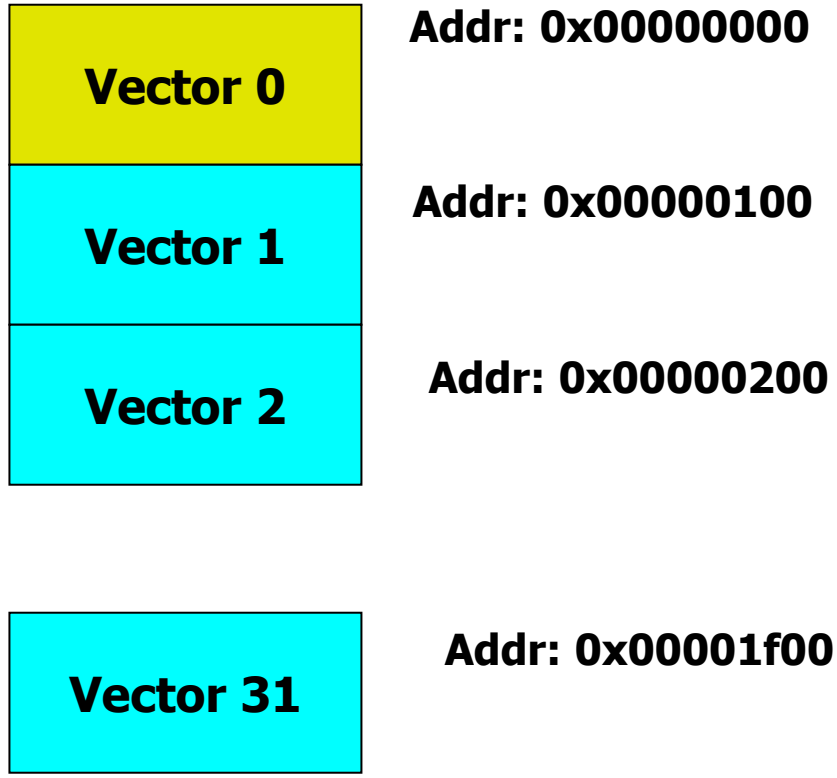
È una tabella esistente nella memoria principale (spesso a partire dall'indirizzo più basso) che contiene, per ogni tipo di interrupt, l'indirizzo della relativa Interrupt Service Routine.



Interrupt Vector Table (II)

In alcuni casi (ad es. PowerPC), l'IVT contiene direttamente le ISR:

- **32 elementi**
- **Per ogni elemento è assegnato uno spazio di 0x100 parole contenente l'ISR.**



Priorità dei periferici

Nei sistemi complessi è spesso necessario assegnare a ciascun periferico una priorità, e fare in modo che la procedura di servizio di un interrupt sia interrompibile solo da un periferico con priorità maggiore.

In alcuni casi le priorità assegnate ai periferici cambiano nel tempo.

La priorità della procedura in corso di esecuzione può essere scritta nella parola di stato.

In altri casi è il controllore degli interrupt che gestisce le priorità.

Richieste simultanee

Nel caso di richieste di interrupt simultanee, è necessario che venga servito prima il periferico con priorità maggiore.

Se si usa il meccanismo delle linee di interrupt multiple, è il processore che risolve il problema.

Nel caso del polling, il processore serve prima il periferico che compare prima nella sequenza con cui i periferici vengono interrogati.

Nel caso dell'interrupt vettorizzato esiste quasi sempre un *Interrupt Controller* che decide quale sia il primo periferico da servire, e fornisce alla CPU il relativo codice identificativo.

Procedura di interrupt (sistemi x86)

1. **Un periferico attiva la richiesta di interrupt verso l'IC**
2. **L'IC attiva la richiesta di interrupt verso la CPU**
3. **La CPU completa l'esecuzione dell'istruzione corrente**
4. **La CPU attiva il segnale Interrupt Acknowledge**
5. **L'IC manda alla CPU il codice del periferico che ha fatto richiesta (attraverso il bus dati)**
6. **La CPU salva nello stack il PC e il registro di stato**
7. **La CPU accede alla IVT e ne estrae l'indirizzo della ISR**
8. **Parte l'esecuzione della ISR**
9. **Alla fine della ISR**
   - **il registro di stato viene ripristinato**
   - **il programma interrotto riprende l'esecuzione.**

Eccezioni

Le eccezioni sono eventi (interni o esterni) che vengono gestiti dalla CPU come le richieste di interrupt provenienti dalle periferiche.

Sono quindi eventi in grado di interrompere l'esecuzione del programma corrente.

Esistono vari tipi di eccezioni:

- **interrupt di I/O (o esterni)**
- **eccezioni di errore**
- **eccezioni di debug**
- **eccezioni di privilegio.**

Ciascuno di questi segnali attiva opportune procedure di servizio, il cui indirizzo è contenuto nella IVT.

Eccezioni di errore

Possono essere generate (tra l'altro)

- **dai circuiti di controllo delle memorie, ad esempio quando si verifica un errore di parità**
- **dal circuito di decodifica delle istruzioni, quando il codice operativo dell'istruzione di cui si è fatto il fetch non corrisponde a nessuna delle istruzioni lecite**
- **dai circuiti aritmetici, quando si verifica una condizione di errore (ad esempio una divisione per 0).**

Eccezioni di debug

Sono utilizzate dai *debugger*:

- **Molti processori prevedono una modalità *trace*; quando viene attivata il processore scatena un'eccezione dopo l'esecuzione di ciascuna istruzione. Tale eccezione viene utilizzata per gestire il modo di esecuzione *single step*.**
- **Inoltre, i processori dispongono spesso di apposite istruzioni che scatenano un'eccezione voluta. Queste possono venire utilizzate per implementare i *breakpoint*.**

Eccezioni di privilegio

Se il processore prevede più modi di funzionamento in base alla priorità del programma in esecuzione (ad esempio *utente* o *supervisore*), viene attivata un'eccezione quando il programma corrente tenta di

- **eseguire un'istruzione non permessa dal livello di priorità corrente**
- **fare accesso a un'area di memoria senza averne i diritti.**

Uso dell'interrupt

I Sistemi Operativi sfruttano pesantemente il meccanismo dell'interrupt:

- **le procedure che compongono il Sistema Operativo (ad esempio per la gestione delle periferiche) sono attivate**
  - **tramite interrupt software dai programmi**
  - **tramite interrupt hardware dalle periferiche stesse**
- **il passaggio all'esecuzione delle procedure di interrupt corrisponde spesso al passaggio ad una modalità speciale (*supervisore*).**

DMA

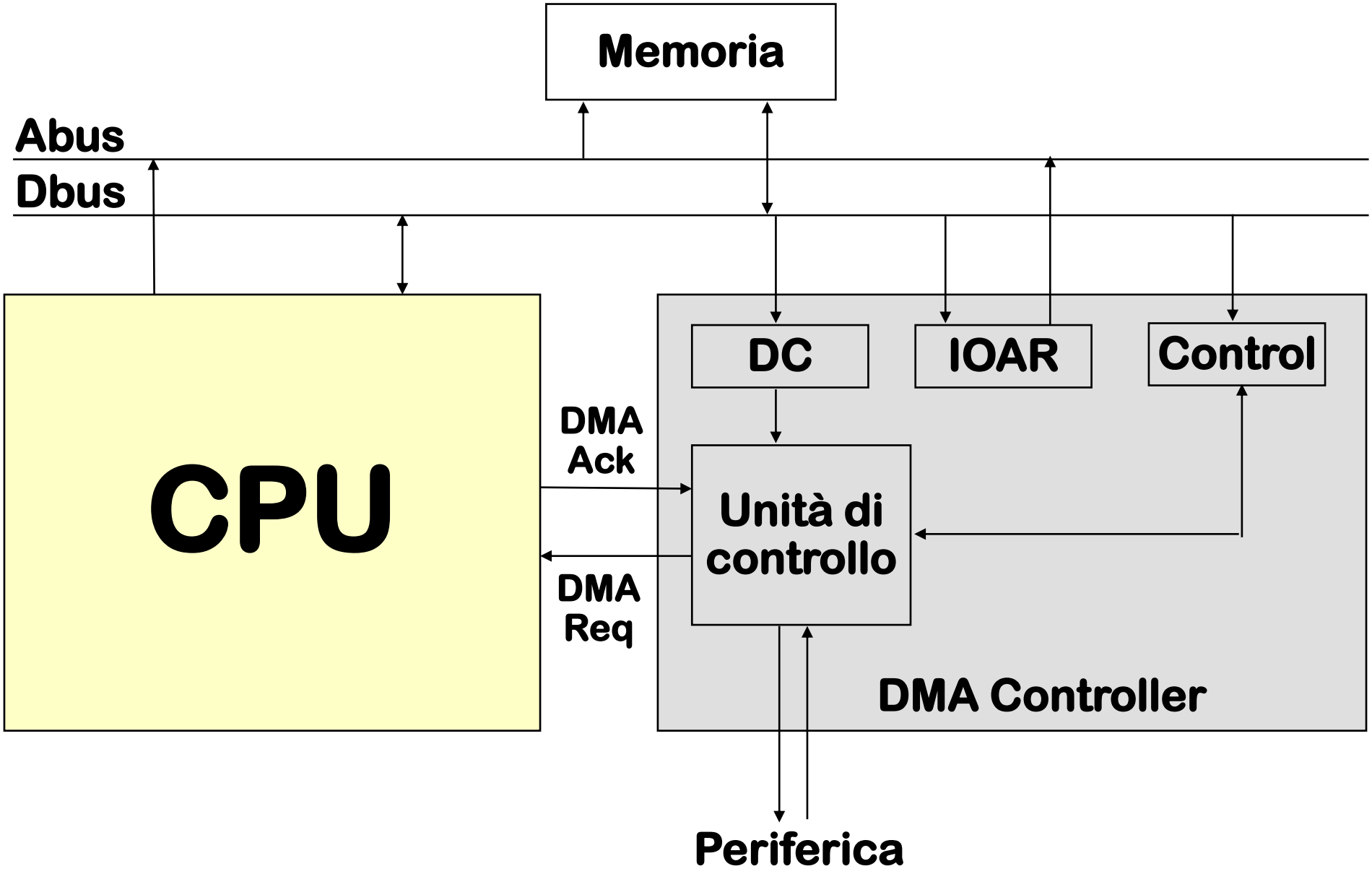
È il metodo preferito quando si devono trasferire grosse moli di dati da una periferica alla memoria (o viceversa).

Il trasferimento viene eseguito da un modulo apposito (*DMA Controller*).

Il modulo deve essere in grado di fungere da *bus master*, ossia deve generare gli indirizzi e i segnali di controllo secondo il protocollo del bus.

Il DMA Controller deve inoltre essere in grado di negoziare con la CPU l'acquisizione del controllo del bus ed il suo rilascio.

Circuiteria per il DMA



Canali di DMA

Ciascun DMA controller può normalmente gestire più periferiche (*canali*).

Per ciascuna periferica il DMA controller dispone di

- **linee di richiesta e di acknowledge**
- **registri DC e IOAR**
- **circuiteria di controllo.**

Il DMA controller è anche in grado di arbitrare tra richieste di DMA contemporanee.

Uso del DMAC

L'uso del DMAC si articola in 2 fasi:

- ***Programmazione*:**
  la CPU carica nei registri IOAR e DC l'indirizzo dell'area di memoria e il numero di parole da trasferire; la CPU informa inoltre il DMA controller della direzione del trasferimento (memoria → periferica o viceversa) e definisce le modalità di trasferimento
- ***Uso*:**
  il DMAC esegue il trasferimento.

Trasferimento in DMA (I)

Il trasferimento di un blocco in DMA (a valle della programmazione) si articola in vari passi:

- **il DMA Controller riceve una richiesta di trasferimento da parte di una periferica**
- **il DMA Controller invia un segnale di DMA Request alla CPU**
- **quando la CPU giunge a un punto di rilevamento del segnale di DMA Request, rilascia il bus e attiva il segnale di DMA Acknowledge**

Trasferimento in DMA (II)

- **il DMA Controller inizia il trasferimento; dopo il trasferimento di ciascuna parola, IOAR e DC vengono aggiornati**
- **il DMA Controller può sospendere temporaneamente il trasferimento (ad esempio perché la periferica non ha più dati da trasferire) disattivando DMA Request; la CPU disattiva DMA Acknowledge, e riprende il controllo del bus**
- **quando DC giunge a zero, il trasferimento termina**
- **il DMA Controller invia un Interrupt alla CPU.**

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

Il DMA Controller, una volta acquisito il controllo del bus, lo mantiene 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.

Alla fine di ogni blocco il DMAC rilascia il bus e subito dopo attiva nuovamente la richiesta di bus.

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 (ad esempio attraverso appositi segnali di stato prodotti dalla CPU), e solo in quei periodi esegue il trasferimento dei dati.

In tal modo la CPU non è praticamente rallentata dal DMA Controller.

Arbitraggio del bus

Il DMA Controller è uno dei dispositivi che possono divenire master del bus.

Un dispositivo può essere master del bus se è in grado di decidere l'operazione che deve essere svolta sul bus (lettura, scrittura, ecc.).

Se esistono diversi DMA Controller, è necessario aggiungere la circuiteria per la gestione dei conflitti.