# Arbitraggio e operazioni del BUS

Aniello Minutolo

# Arbitraggio del bus

# Sommario

1. Arbitraggio del bus
2. Arbitraggio decentralizzato del bus
3. Operazioni del bus

# Necessità dell'arbitraggio del bus

**La CPU non è l'unico master del bus**

- anche i chip di I/O devono diventare master per poter leggere e scrivere in memoria e provocare interrupt, e anche i coprocessori potrebbero avere la necessità di fungere da maste.

L'**arbitraggio del bus** è necessario quando più dispositivi richiedono contemporaneamente di diventare **master** del bus

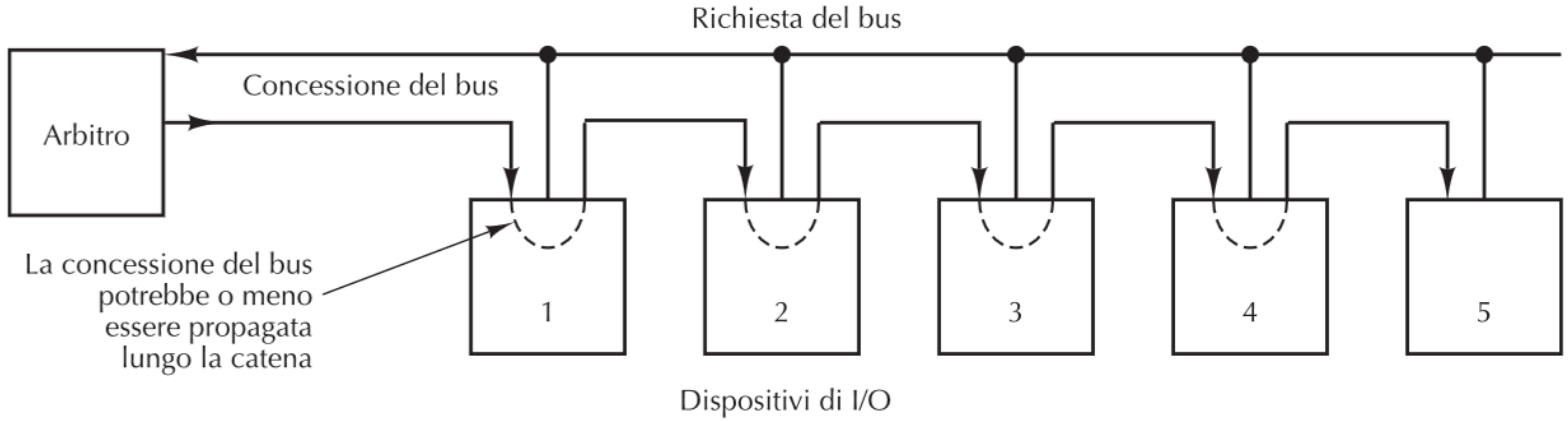
- senza un meccanismo di arbitraggio, si creerebbe confusione tra i dispositivi che competono per l'accesso al bus;
- l'arbitraggio garantisce che solo un dispositivo alla volta possa controllare il bus, evitando conflitti.

# Arbitraggio centralizzato

L'**arbitraggio centralizzato** utilizza una singola entità che funge da arbitro del bus e determina chi sarà il prossimo master tra i dispositivi che lo hanno richiesto

- l'arbitro del bus può essere integrato nella CPU o essere un chip separato, a seconda dell'architettura del sistema.

Il bus include una linea di richiesta **OR-cablata** che può essere attivata da uno o più dispositivi



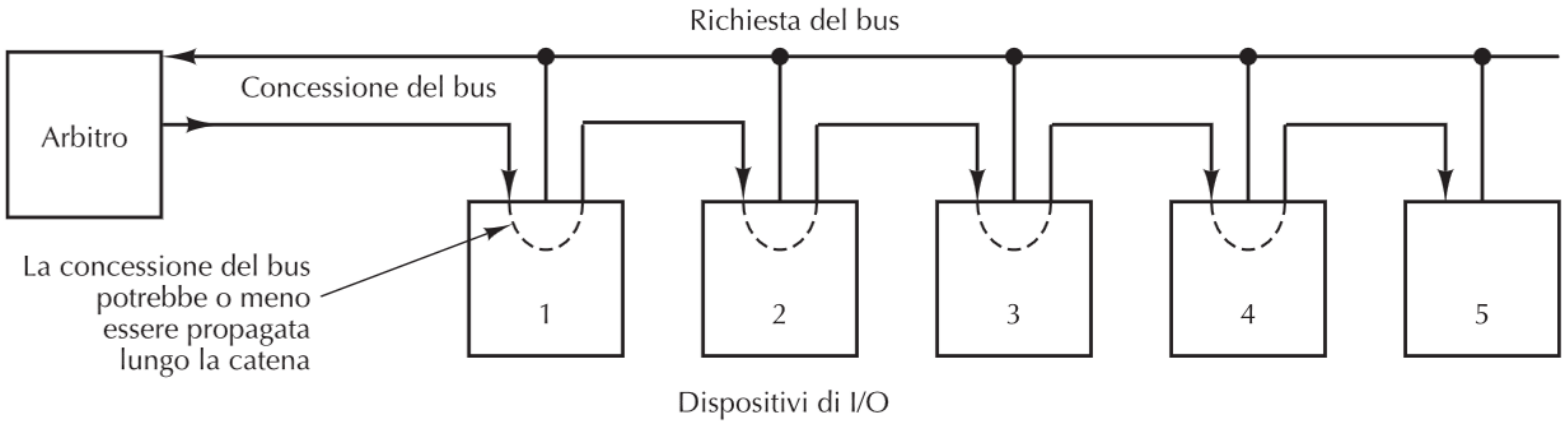
# Arbitraggio centralizzato

L'arbitro del bus non ha alcun modo per sapere quanti dispositivi hanno richiesto di diventare master del bus

- l'arbitro può solo distinguere tra "qualche richiesta" e "nessuna richiesta".

Quando l'arbitro vede una richiesta di utilizzo del bus, lo concede asserendo la linea per la concessione del bus

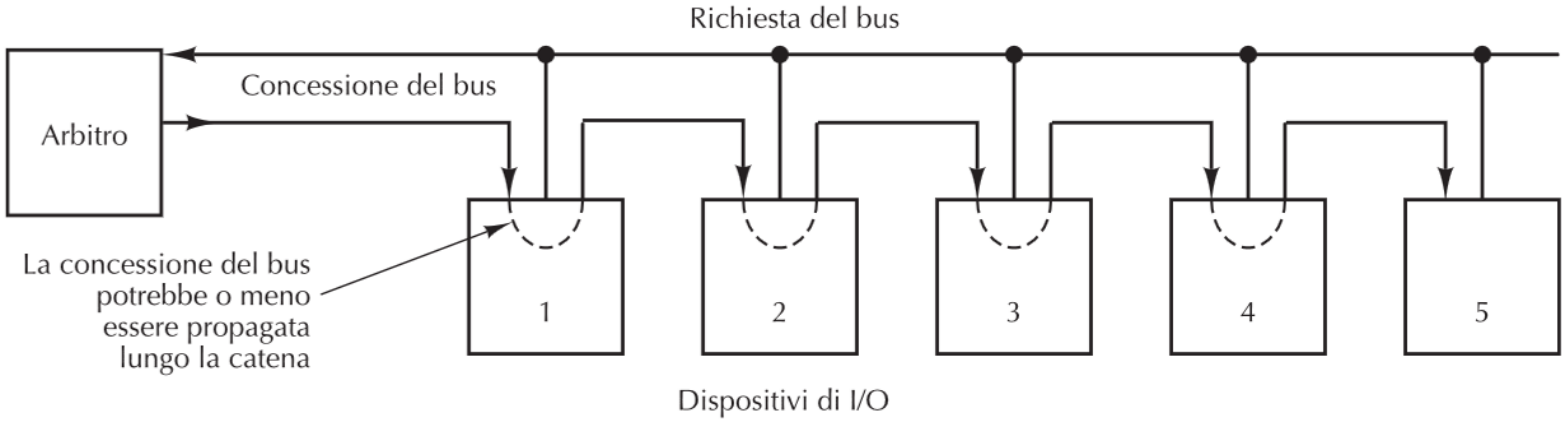
- a questa linea sono collegati in serie tutti i dispositivi di I/O, come un semplice filo di lucine natalizie.



# Arbitraggio centralizzato

Quando il dispositivo fisicamente più vicino all'arbitro vede la concessione, verifica se ne ha fatto richiesta

- in caso affermativo si impossessa del bus, senza però propagare la concessione lungo il resto della linea;
- se invece non ha fatto richiesta, propaga la concessione sulla linea in direzione del prossimo dispositivo che si comporterà allo stesso modo, e così pure i successivi, finché un dispositivo accetterà la concessione e si impossesserà del bus.



# Collegamento a festone (daisy chaining)

Questo schema si chiama **daisy chaining** (collegamento a festone)

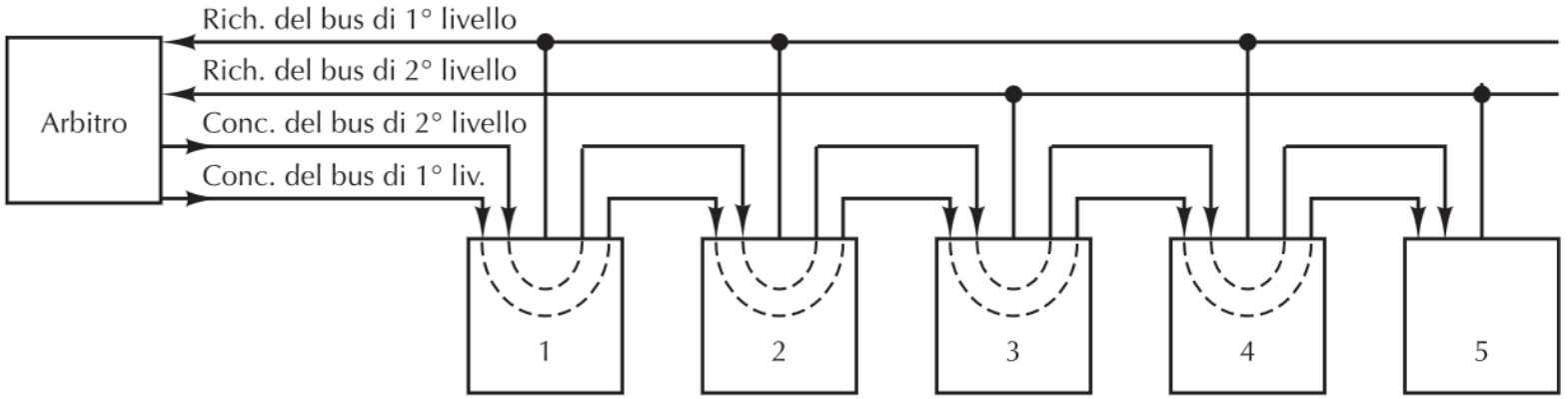
- assegna ai dispositivi diverse priorità in base alla loro vicinanza all’arbitro;
- il dispositivo più vicino all'arbitro ha la priorità più alta e può bloccare la propagazione della concessione del bus.

Lo schema **daisy chaining** è molto semplice ma limita la flessibilità nella gestione delle priorità.

# Livelli di priorità

Per superare i limiti del **daisy chaining**, molti bus definiscono esplicitamente i livelli di priorità dei dispositivi

- ogni livello di priorità ha una linea di richiesta e una linea di concessione dedicate;
- i dispositivi con temporizzazioni critiche, come quelli di I/O, ricevono tipicamente priorità più elevate.



# Concessione del bus con priorità

- Se in uno stesso momento ci sono richieste da livelli di priorità diversi l’arbitro concede il bus al dispositivo di priorità più alta.
- Tra dispositivi con lo stesso livello di priorità, si applica il **daisy chaining** per determinare l'ordine di accesso al bus.

# Linea di conferma

Alcuni arbitri hanno una terza **linea di conferma** che viene asserita da un dispositivo nel momento in cui accetta una concessione e si impadronisce del bus

- dopo aver asserito questa linea di conferma, il dispositivo può negare le linee di richiesta e di concessione. Il risultato è che altri dispositivi possono richiedere il bus mentre il primo lo sta ancora utilizzando.

L’utilizzo della linea di conferma permette di migliorare l'efficienza del bus ma richiede hardware aggiuntivo.

# Priorità della CPU

- Nei sistemi con memoria sul bus principale, la CPU compete con i dispositivi di I/O per l'accesso al bus.
- Spesso alla CPU viene assegnata la priorità più bassa, poiché può attendere, a differenza dei dispositivi di I/O.
- Una soluzione alternativa è separare il bus della memoria da quello dei dispositivi di I/O.

# Arbitraggio decentralizzato del bus

# Arbitraggio decentralizzato

- L'arbitraggio decentralizzato elimina la necessità di un arbitro centrale distribuendo la logica di decisione tra i dispositivi.
- Ogni dispositivo monitora direttamente le linee di richiesta del bus per determinare se ha diritto o meno a utilizzare il bus.
- Questo approccio richiede più linee di bus ma evita il potenziale costo dell'arbitro centrale.

# Schema a linee di richiesta multiple

Un calcolatore con arbitraggio del bus decentralizzato potrebbe per esempio avere 16 linee di richiesta, ciascuna con la propria priorità

- quando un dispositivo vuole utilizzare il bus asserisce la linea di richiesta;
- tutti i dispositivi monitorano tutte le linee di richiesta;
- alla fine di ciascun ciclo del bus ogni dispositivo che ha richiesto il bus può sapere se era il richiedente con priorità più elevata e se quindi ha diritto a utilizzare il bus durante il ciclo successivo.

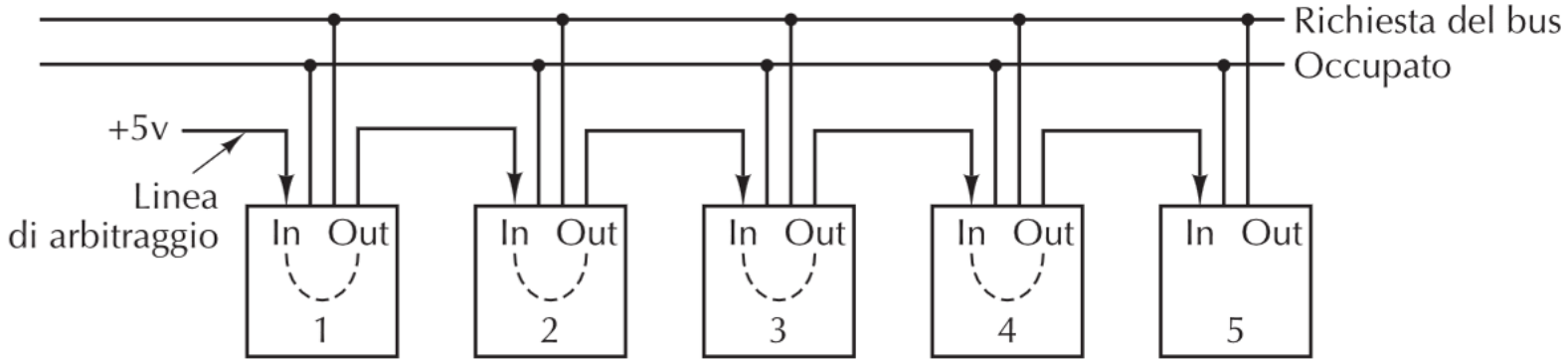
# Limitazioni dell'arbitraggio decentralizzato

- L'arbitraggio decentralizzato richiede più linee rispetto a quello centralizzato, aumentando la complessità del bus e i costi di realizzazione.
- Un altro limite dell’arbitraggio del bus decentralizzato è che il numero di dispositivi non può superare il numero delle linee di richiesta.
- Nonostante i costi hardware, questo metodo offre tempi di risposta più prevedibili.

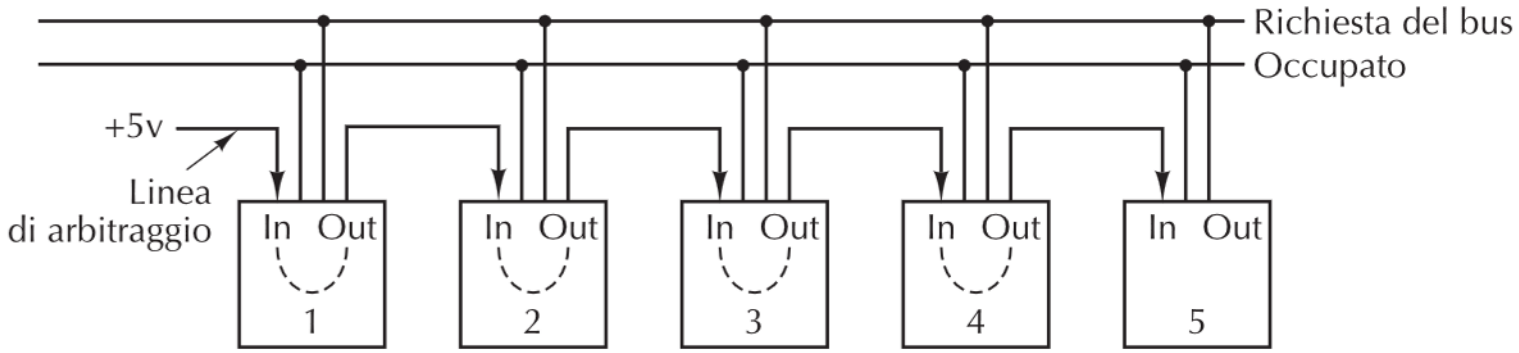
# Schema a tre linee

Un altro tipo di arbitraggio decentralizzato del bus utilizza solamente **tre linee**, indipendentemente da quanti dispositivi sono presenti

- la prima linea è una linea OR-cablata per effettuare le richieste al bus;
- la seconda linea è chiamata **BUSY** ed è asserita dal master corrente;
- la terza linea è utilizzata per arbitrare il bus ed è un collegamento a festone fra tutti i dispositivi;
  - viene mantenuta asserita collegandola a un'alimentazione di 5 volt.



# Schema a tre linee



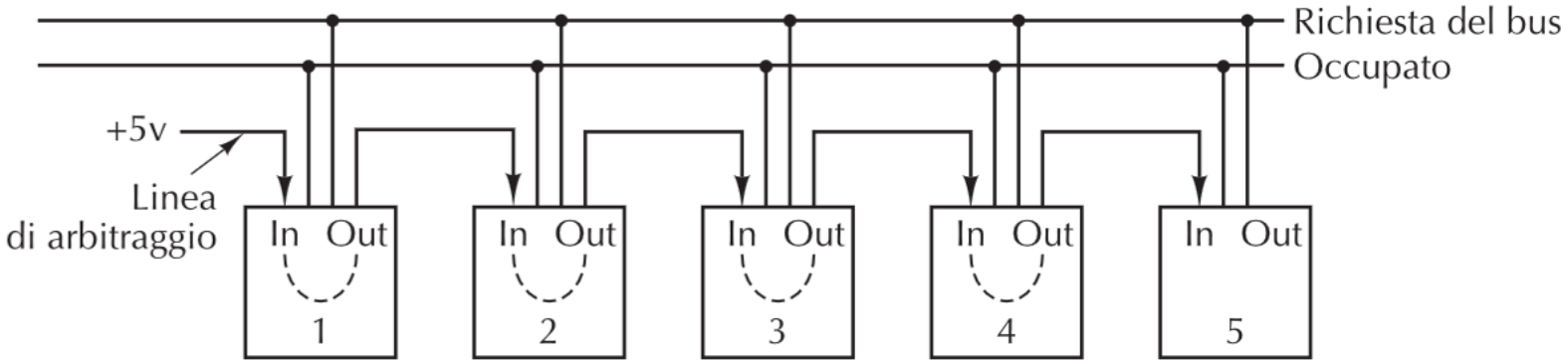

Quando nessun dispositivo richiede il bus la linea di arbitraggio, che è asserita, viene propagata attraverso tutti i dispositivi

- ogni dispositivo rileva che il proprio **IN** è asserito e, non essendo interessato a richiedere il bus, asserisce il proprio **OUT** contribuendo così a mantenere asserita la linea di arbitraggio lungo la daisy chaining.

# Schema a tre linee

Per acquisire il bus, un dispositivo deve prima controllare se il bus è inattivo e se il proprio segnale **IN** dell'arbitraggio è asserito

- se il suo **IN** è negato, <u>non può</u> diventare il master e nega il suo **OUT;**
- se il suo **IN** è asserito**,** <u>può</u> diventare il master e nega il suo **OUT;**
- la negazione del suo **OUT** comporta che tutti i dispositivi successivi vedranno il proprio **IN** negato e negheranno a loro volta il proprio **OUT.**

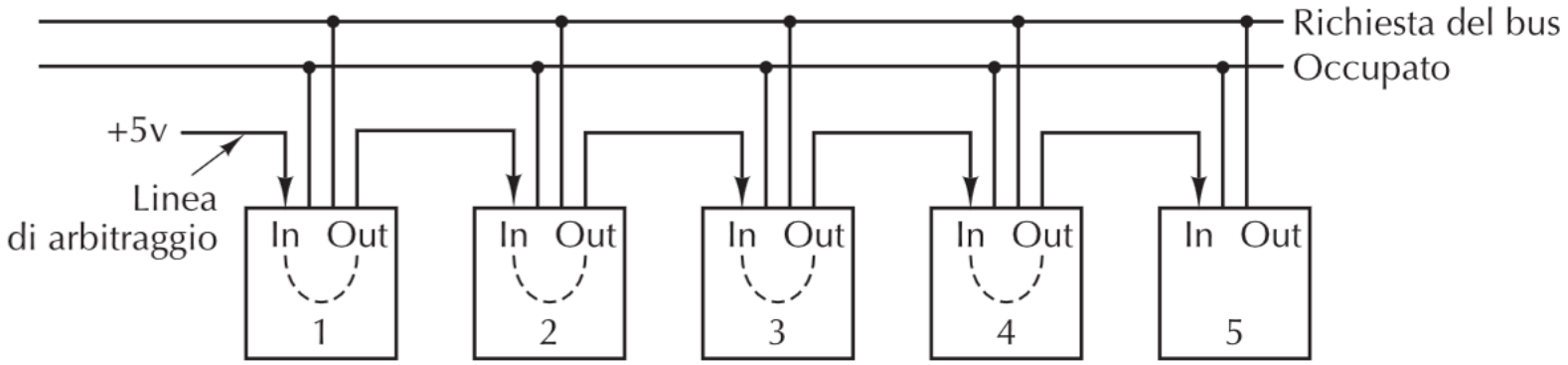


# Schema a tre linee

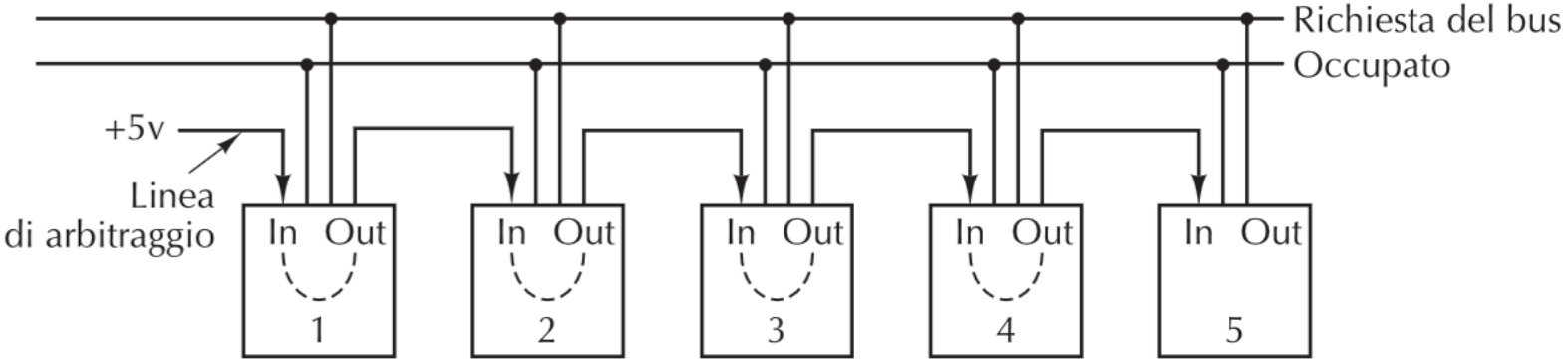
Una volta tornata la calma solo un dispositivo rimarrà con il proprio segnale **IN** asserito e con il proprio **OUT** negato

- esso diventerà il master del bus, asserirà **BUSY** e **OUT** e inizierà il proprio trasferimento.

Si osservi che fra i dispositivi che richiedono contemporaneamente il bus lo otterrà quello che si trova più a sinistra.



# Schema a tre linee




- Questo schema è simile all'originario arbitraggio con collegamento a festone, tranne il fatto che non c'è l'arbitro.
- Questa differenza lo rende più economico, più veloce e non soggetto ai potenziali guasti dell'arbitro.

# Operazioni del bus

# Introduzione

Finora abbiamo trattato solamente bus ordinari

- un master (generalmente la CPU) che legge da uno slave (di solito la memoria) o scrive su di esso.

Nei casi reali sono possibili anche altri tipi di bus

- di solito viene trasferita una parola alla volta;
- se si utilizza la cache è preferibile prelevare un’intera linea di cache (per esempio 8 parole consecutive di 64 bit).

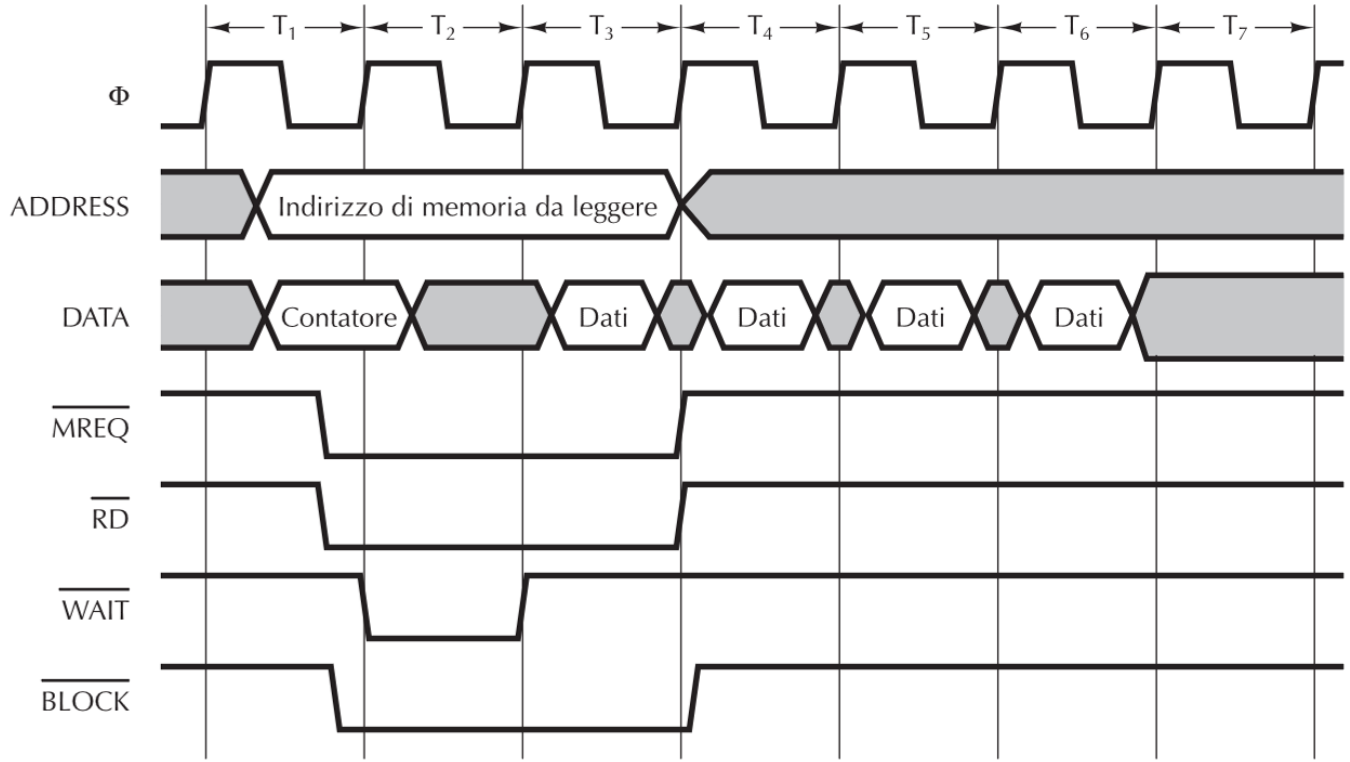
# Trasferimenti di blocchi

- I trasferimenti di blocchi permettono di leggere o scrivere più **parole consecutive** in un'unica operazione, migliorando l'efficienza.
- Durante una lettura a blocchi, il **master comunica** allo slave il **numero di parole da trasferire** tramite le linee dei dati.
- Lo **slave**, invece di restituire un’unica parola, **genera in output a ogni ciclo una nuova parola** finché il contatore non si sia esaurito.

# Trasferimenti di blocchi

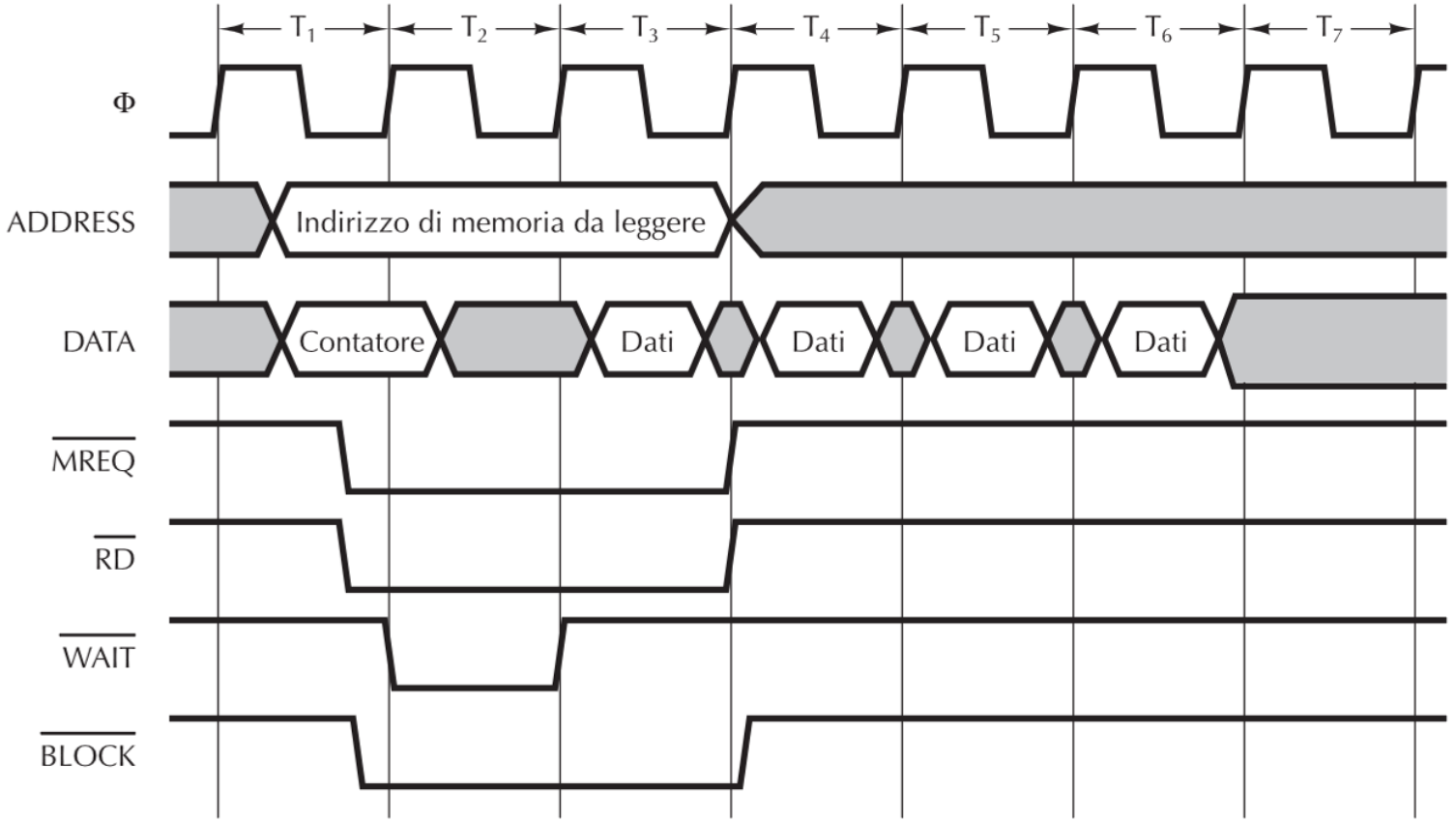
Ad esempio, in figura è mostrato l'utilizzo di un segnale **BLOCK** che viene asserito per indicare la richiesta di trasferimento di un blocco

- in questo esempio una lettura di un blocco di 4 parole richiede 6 cicli invece che 12



# Trasferimenti di blocchi

- Ad esempio, in figura è mostrato l'utilizzo di un segnale **BLOCK** che viene asserito per indicare la richiesta di trasferimento di un blocco;
- in questo esempio una lettura di un blocco di 4 parole richiede 6 cicli invece che 12.



# Bus nei sistemi multiprocessore

Nei sistemi multiprocessore con due o più CPU sullo stesso bus è spesso necessario assicurarsi che, in un dato momento, soltanto una CPU utilizzi delle strutture dati critiche della memoria.

Una tipica soluzione del problema consiste nell'avere in memoria una variabile che vale 0 quando nessuna CPU sta utilizzando la struttura dati e 1 quando invece è in uso.

- Il problema è che, con una buona dose di sfortuna, due CPU potrebbero leggere la variabile durante cicli di bus consecutivi, rilevando entrambe la disponibilità della struttura dati e pensando entrambe di poterla usare;
  - race condition tra CPU concorrenti che accedono alla stessa risorsa.

# Cicli leggi-modifica-scrivi

Per evitare questa situazione i sistemi multiprocessore hanno spesso uno speciale ciclo di bus **leggi-modifica-scrivi**

- consente a una qualsiasi CPU di leggere una parola dalla memoria, analizzarla e riscriverla in memoria, tutto senza rilasciare il bus;
- evita che altre CPU che intendono utilizzare il bus riescano a impadronirsene interferendo con l'operazione della prima CPU che ha in uso il bus.

# Gestione degli interrupt

Un importante tipo di ciclo di bus serve a gestire gli interrupt

Quando la CPU ordina a un dispositivo di I/O di effettuare un'operazione si aspetta di solito un interrupt alla fine del lavoro

- la segnalazione di questi interrupt richiede l'utilizzo del bus.

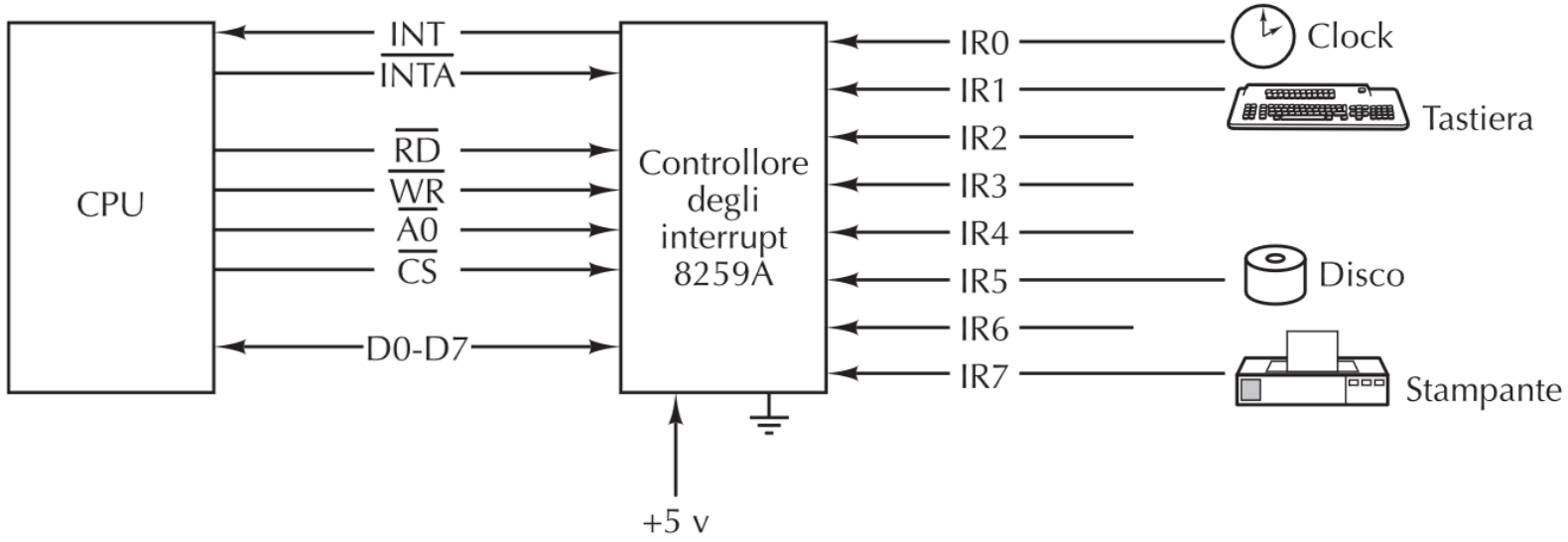
Dato che più dispositivi potrebbero voler generare un interrupt simultaneamente, si verificano gli stessi tipi di problemi di arbitraggio visti nel caso dei cicli di bus ordinari

- la soluzione più usuale prevede di assegnare priorità ai dispositivi e di utilizzare un arbitro centralizzato per dare priorità al dispositivo per cui la temporizzazione è più critica.

# Chip 8259A

Esistono chip controllori di interrupt di tipo standard che sono utilizzati in modo molto diffuso

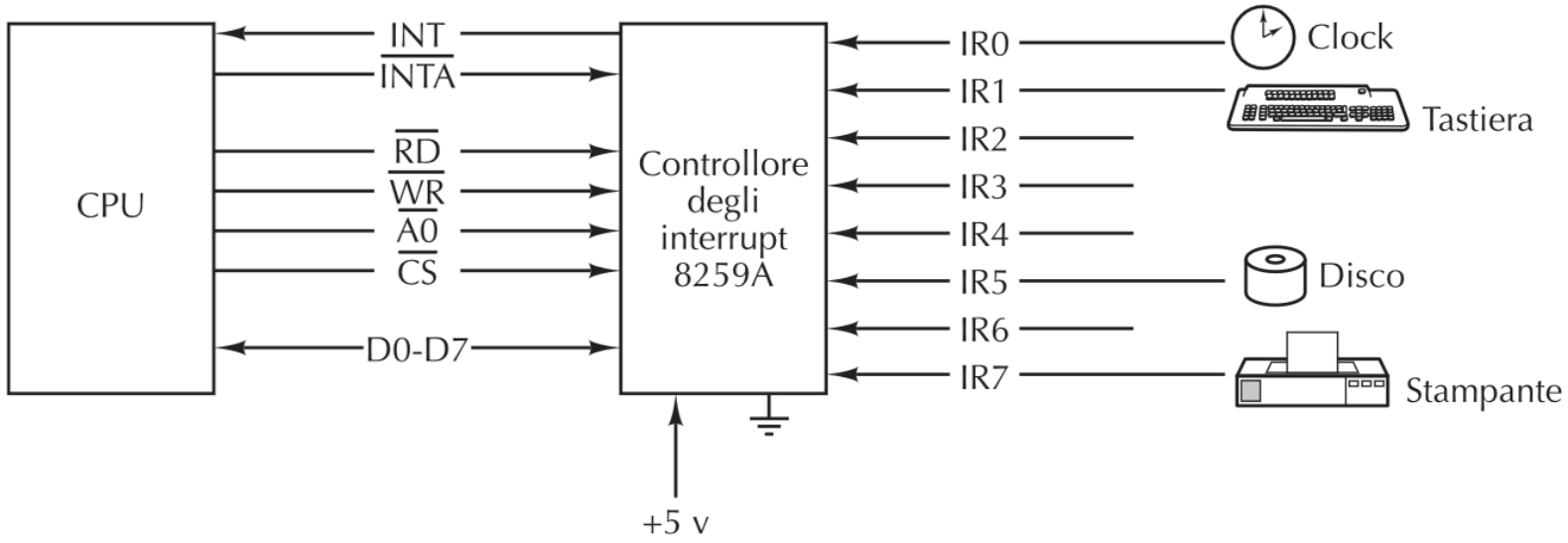
- nei PC basati su processori Intel il chipset incorpora il controllore di interrupt 8259A.



# Chip 8259A

L'**8259A** asserisce il segnale **INT** alla CPU quando un dispositivo collegato richiede un interrupt

- dopo la conferma della CPU, l'**8259A** invia il numero dell'interrupt sul bus dei dati per identificare la routine da eseguire;
- il software può configurare l'**8259A** tramite registri per abilitare, disabilitare o mascherare interrupt specifici.



# Bibliografia

**Libro di testo**

- Andrew S. Tanenbaum e Todd Austin. Architettura dei calcolatori. Un approccio strutturale. 6/ED. Anno 2013. Pearson Italia spa. (Disponibile nella sezione “Biblioteca”)

**Fonte argomenti e immagini**

- Capitolo 3, Paragrafi da 3.4.5 a 3.4.6, pp. 202-208