

Modulo Calcolatori Elettronici

Proff. Gian Luca Marcialis, Giulia Orrù, Lorenzo Putzu, Fabio Roli

Corsi di Laurea in Ingegneria Biomedica Ingegneria Elettrica, Elettronica ed Informatica

Contatti:

marcialis@unica.it, giulia.orrù@unica.it, lorenzo.putzu@unica.it

Capitolo 6

Unità di Ingresso e Uscita

Fonti Principali: Patterson, A.D., Hennessy, J., "Struttura e progetto dei calcolatori elettronici", Zanichelli editore, 2019

Sommario

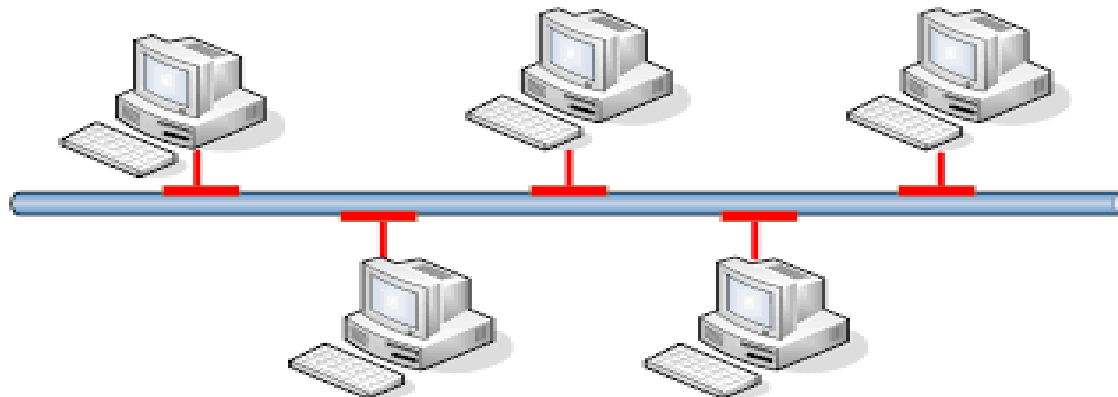
- Introduzione
- Il Bus e la sua gestione
- Modulo I/O e Periferiche
 - I/O da programma
 - I/O con interruzioni
 - Direct Memory Access (DMA)
- Cenni sulle interfacce

Strutture di Interconnessione

- L'insieme delle "strutture" di collegamento fra i vari moduli di un calcolatore (Memoria, CPU, dispositivi esterni, ecc.) viene detta Struttura di Interconnessione
- La Struttura di Interconnessione dipende dai tipi di "dati" (istruzioni, indirizzi, ecc.) che devono essere scambiati fra i principali moduli di un calcolatore.
- La struttura di interconnessione deve garantire trasferimenti di dati da:
 - Memoria a CPU e CPU a memoria
 - Da/a dispositivi esterni (che chiameremo anche periferiche) a/da CPU
 - Da/a periferiche a/da memoria

Struttura di Interconnessione a BUS

- Un Bus è una struttura di interconnessione che collega due o più moduli/dispositivi
- La sua caratteristica principale è quella di essere una struttura di interconnessione “condivisa” (“shared” Bus)
 - ✓ Più dispositivi sono collegati ad un unico bus
 - ✓ Solo un dispositivo alla volta può usare il Bus

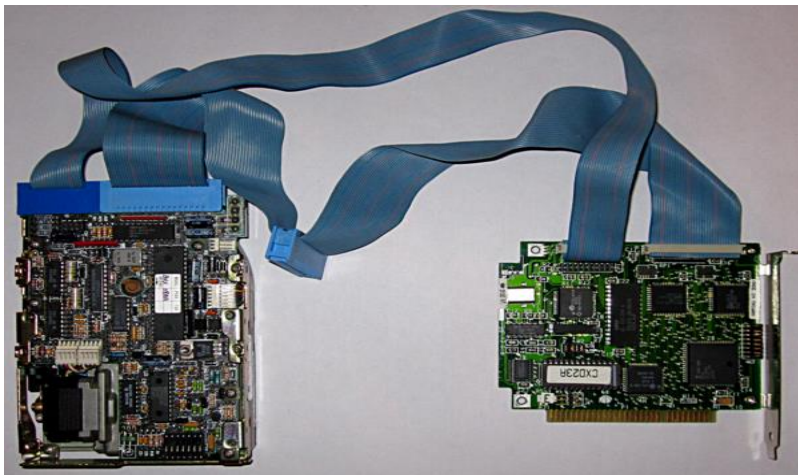


Struttura di Interconnessione a BUS

- Un Bus è tipicamente costituito da un insieme di linee di trasmissione per segnali digitali
 - Le trasmissioni sul Bus possono essere “**parallele**” o “**seriali**”. Oggi prevalgono i bus seriali come *USB* (Universal Serial Bus) che occupano meno spazio, hanno costi minori e altri vantaggi che saranno illustrati in altri corsi.
 - Storicamente il Bus che collega i principali moduli (CPU, Memoria, I/O) del calcolatore viene detto **Bus di Sistema** (**System Bus**). In un moderno calcolatore si hanno **molteplici** Bus.
 - Esistono diverse tecnologie per realizzare un bus (vedi dopo).
- In questo corso ci limiteremo a studiare il bus a **livello logico**

Cenni sulla fabbricazione del Bus

- I BUS sono realizzati tramite collegamenti elettrici
- Le connessioni elettriche del bus possono essere realizzate direttamente sul circuito stampato oppure tramite appositi cavi.
 - ✓ schede di BUS, con piste di collegamento e connettori montati sulla scheda
 - ✓ cavi elettrici flessibili connettorizzati



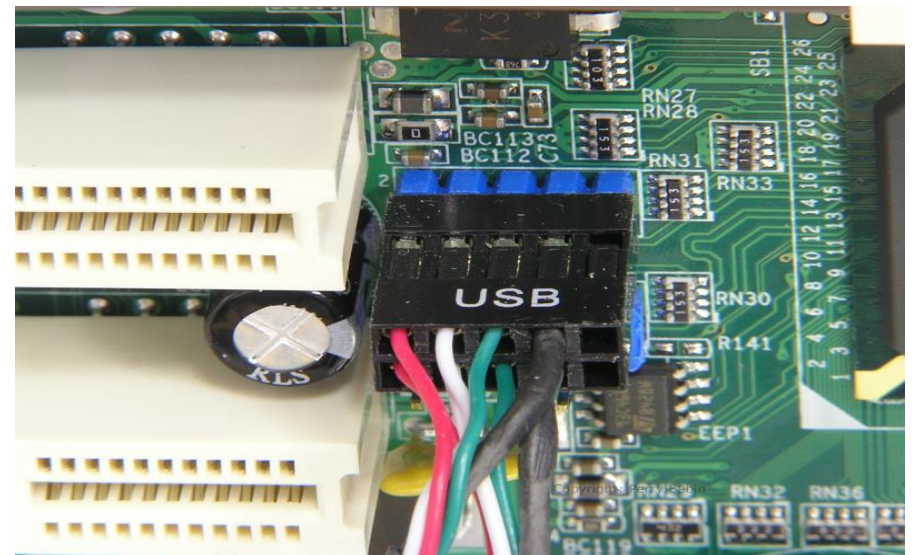
Calcolatori Elettronici



Unità di I/O

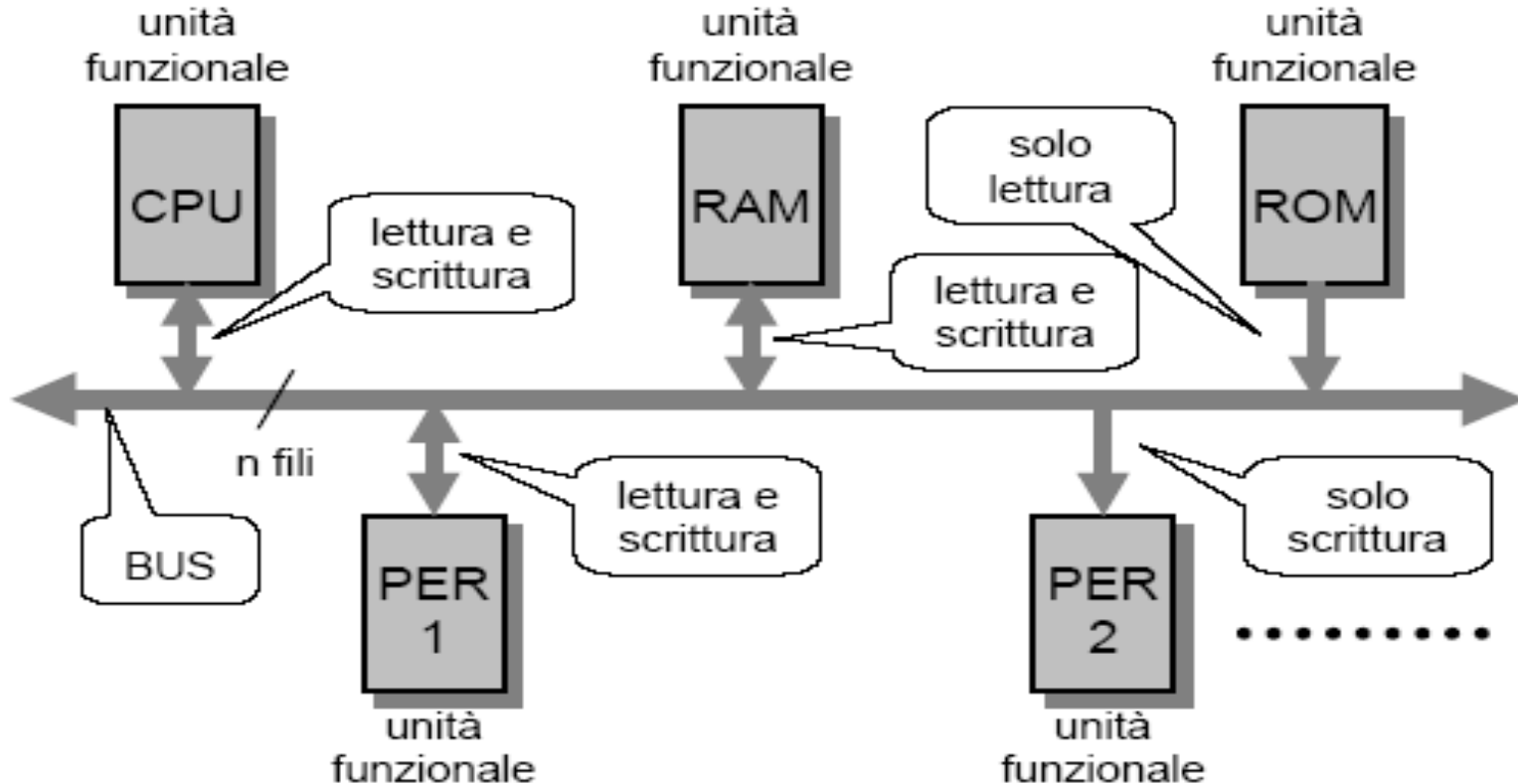
Cenni sulla fabbricazione del Bus

- Alcuni BUS, ad altissime prestazioni, sono realizzati in fibra ottica
- Alcuni BUS si basano su trasmissione via etere (onde radio, Bluetooth)
- Oggi prevalgono bus seriali come il bus **USB**



Struttura di interconnessione a bus

Un calcolatore può essere visto come un insieme di unità “funzionali”, CPU, Unità di Memoria, Periferiche esterne per I/O (PER), interconnesse tramite una struttura detta **Bus**



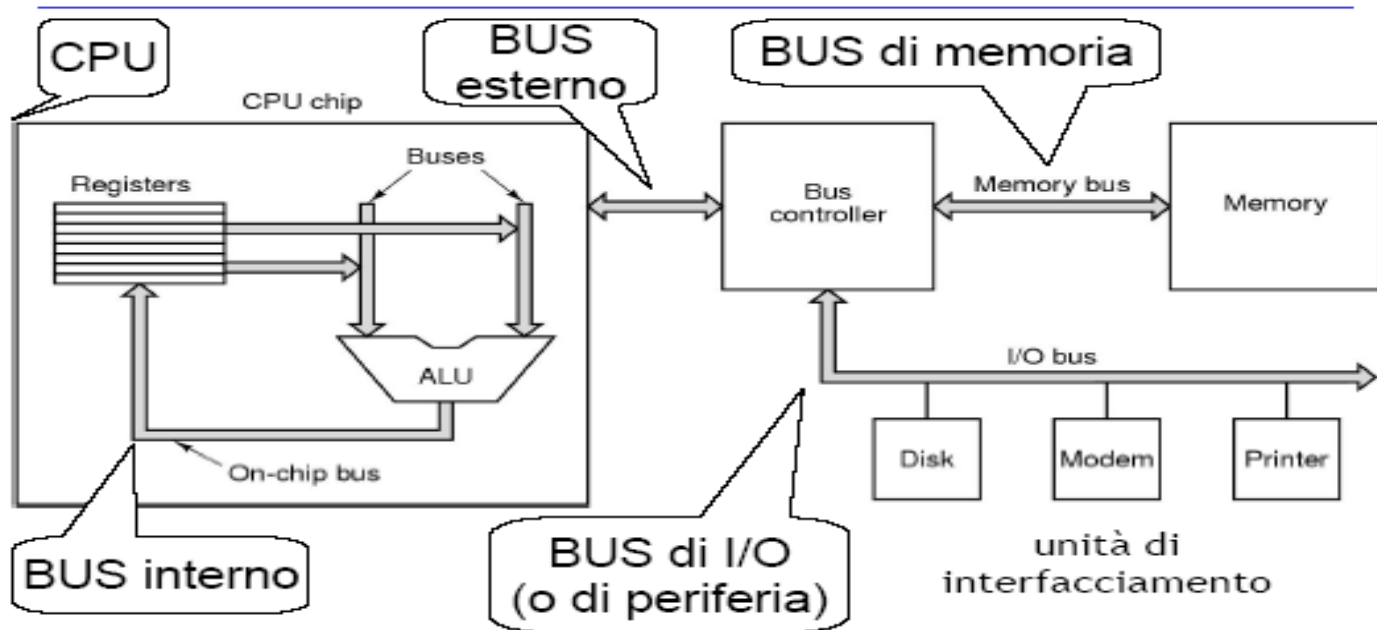
Bus interni ed esterni

In un moderno calcolatore si hanno:

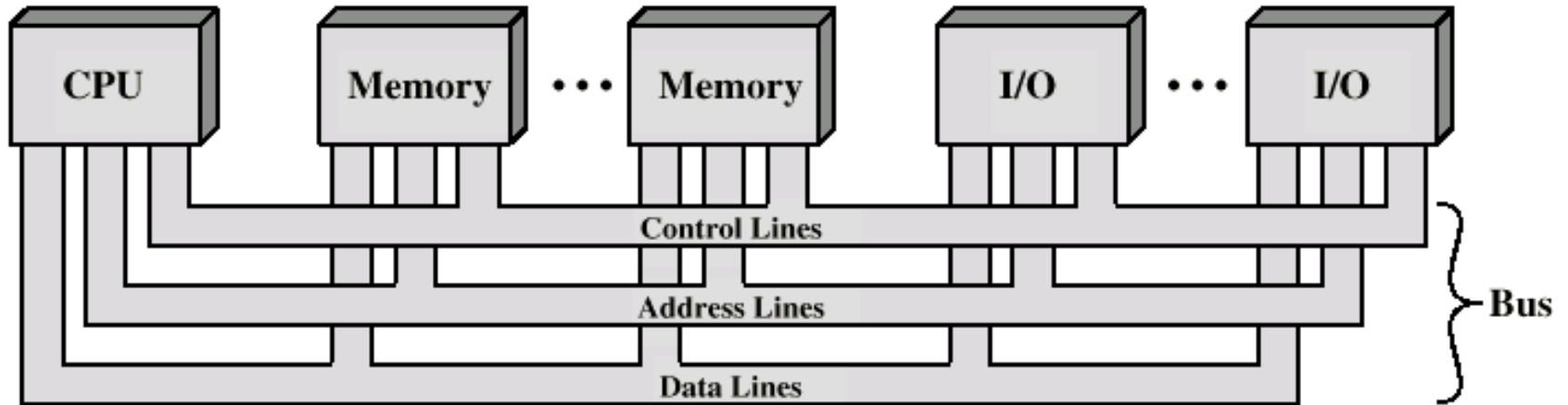
➤ **BUS interni**, confinati all'interno di una singola unità funzionale, e che collegano i blocchi funzionali contenuti nell'unità

➤ **BUS esterni**, che si estendono all'esterno dell'unità funzionale, e che la collegano alle altre unità funzionali. I BUS esterni del calcolatore sono solitamente standardizzati.

I Bus interni non sono normalmente «standardizzati»



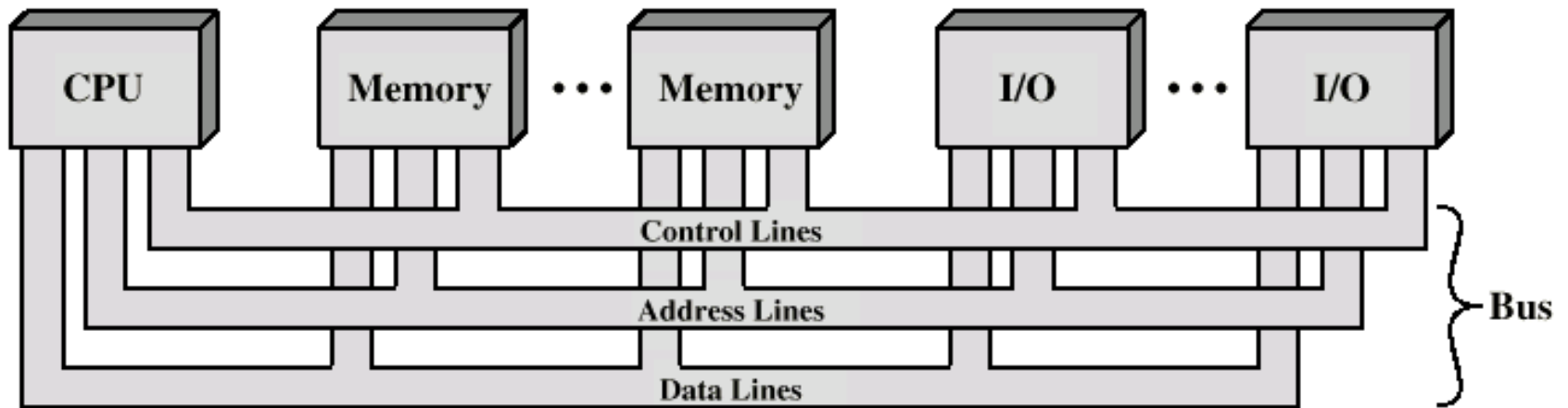
Schema logico di un BUS



Data Bus:

- ✓ Trasporta dati ed istruzioni
- ✓ Può avere diverse “ampiezze” nel caso parallelo (32, 64 bit)
- ✓ Oggi prevalgono i bus seriali

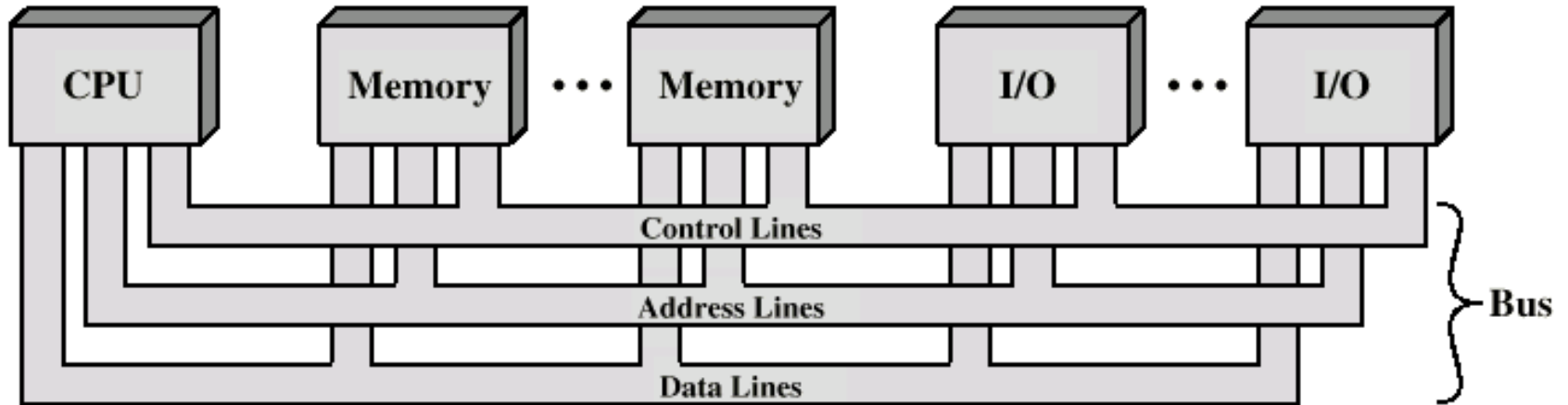
Schema di un BUS



Address Bus:

- ✓ Identifica la sorgente o la destinazione dei dati
 - ✓ La sua ampiezza determina lo spazio di memoria indirizzabile esplicitamente
- Esempio storico: Intel 8080 aveva un address bus da 16 bit, quindi poteva indirizzare uno spazio di memoria di 64K

Schema di un BUS



Control Bus:

- Trasmette informazioni di controllo e temporizzazione
 - Segnali di read/write
 - Interrupt request
 - Clock signal
 - Altri segnali di controllo

Concetti chiave sull' I/O

Tutte le operazioni di I/O fra le unità avvengono sostanzialmente attraverso i Bus

Tali operazioni coinvolgono i seguenti problemi principali:

- **Controllo** del Bus: quali unità usano in un certo intervallo di tempo il Bus per le loro operazioni di I/O; come vengono gestite richieste in “conflitto” da parte di diverse unità
- **Funzionamento** del Bus: come vengono eseguite (e “temporizzate”) le operazioni di I/O sul Bus
- **Modalità di I/O**: chi “esegue” le operazioni di I/O fatte sul BUS

Controllo del Bus

- In ogni istante, una sola unità possiede il controllo del BUS, cioè decide quali operazioni di I/O eseguire
- Generalmente la CPU possiede il controllo del BUS (o dei BUS, se ve n'è più d'uno), ma può anche cedere temporaneamente questo ruolo ad altre unità

Concetto di **Master** e **Slave**

- L'unità che detiene il controllo del BUS si chiama MASTER: decide quale operazione eseguire, lettura oppure scrittura, e in quale istante di tempo; decide ovviamente quale è l'unità funzionale da leggere oppure da scrivere
- Le rimanenti unità funzionali, che non detengono il controllo del BUS, vengono dette SLAVE
- Ogni BUS è controllato da un unico MASTER; ogni MASTER può controllare più BUS. Il calcolatore deve possedere almeno un MASTER (ad es. la CPU)
- Ogni BUS può collegare più SLAVE

Arbitraggio del Bus

- Normalmente la CPU ha il ruolo di MASTER tra le unità funzionali
- Tuttavia in determinate circostanze anche altre unità funzionali possono assumere temporaneamente il ruolo di MASTER. Unità di I/O: possono diventare MASTER per trasferire dati direttamente con la memoria, senza bisogno della CPU (vedremo il caso del DMA). (Co)processore di I/O: può diventare MASTER e prelevare operandi dalla memoria

Arbitraggio del Bus

- In caso di cessione del ruolo di MASTER da un'unità funzionale a un'altra, o di più richieste contemporanee del Bus, occorre un meccanismo di ARBITRAGGIO DEL BUS, che ne regoli l'utilizzo da parte delle unità funzionali
- Esistono due meccanismi principali di arbitraggio: **centralizzato** e **distribuito**

Arbitraggio Centralizzato

Il meccanismo di arbitraggio centralizzato prevede:

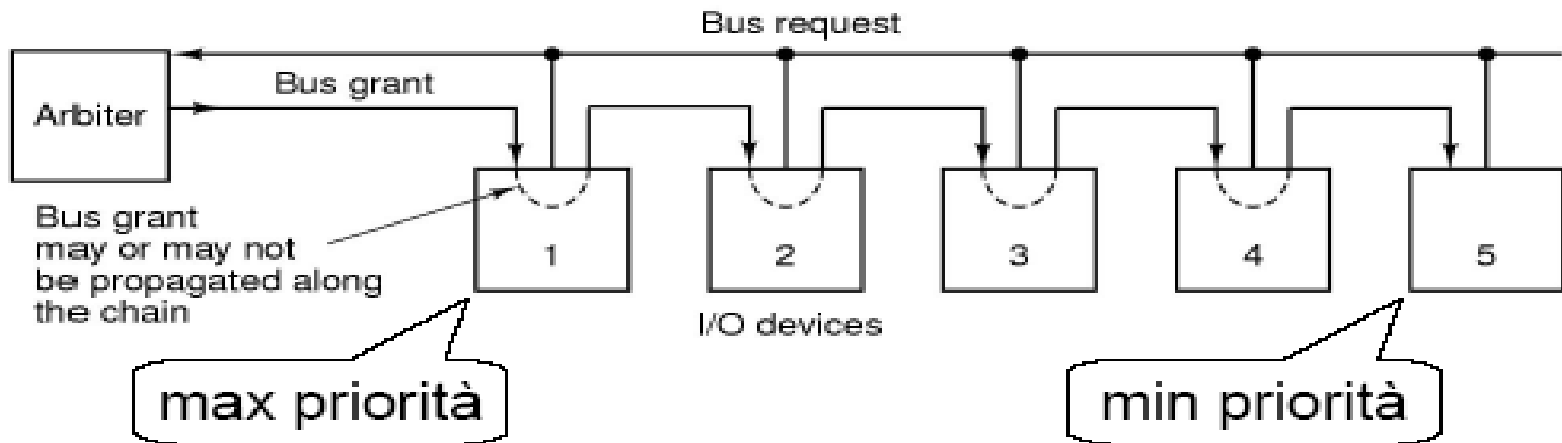
- un'apposita unità funzionale, che svolge la funzione di «arbitro» del BUS
- alcune linee (appartenenti alle linee di controllo del Bus) che collegano l'arbitro del BUS alle unità funzionali che potrebbero richiedere il controllo del BUS

➤ L'arbitro realizza il meccanismo di cessione del controllo del BUS, vale a dire la cessione del ruolo di MASTER

Arbitraggio Centralizzato in “Daisy Chain”

Quando l'unità di arbitraggio riceve una richiesta di BUS (BUS request), attiva la linea di conferma (BUS grant). La conferma viene passata in cascata alle unità potenziali richiedenti.

Se un'unità non ha una richiesta pendente, passa la conferma all'unità successiva. Altrimenti trattiene per sé la conferma, senza passarla all'unità successiva, e prende il controllo del BUS, comportandosi come MASTER

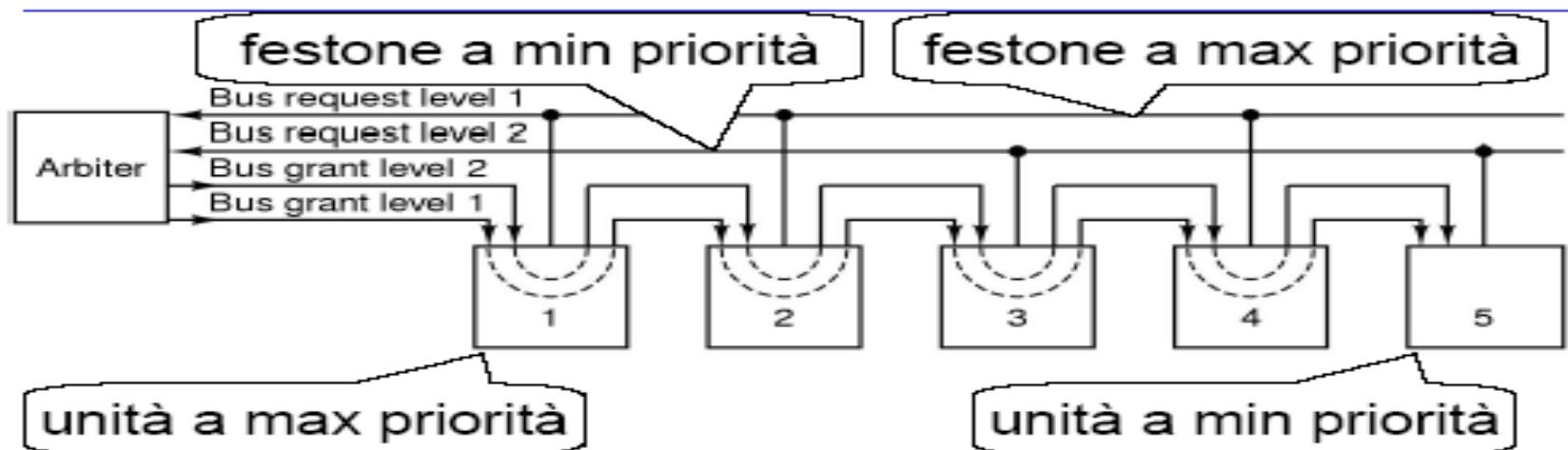


Meccanismo di arbitraggio centralizzato di tipo semplice, basato sullo scambio di un segnale di conferma (o BUS grant), per regolare i turni di utilizzo del BUS

Daisy chain con più “festoni”

Il meccanismo di arbitraggio centralizzato con collegamento a “festone” (Daisy Chain) è un modo cablato (cioè hardware) per assegnare le “priorità” alle unità. L’unità più vicina all’arbitro è sempre quella a priorità massima.

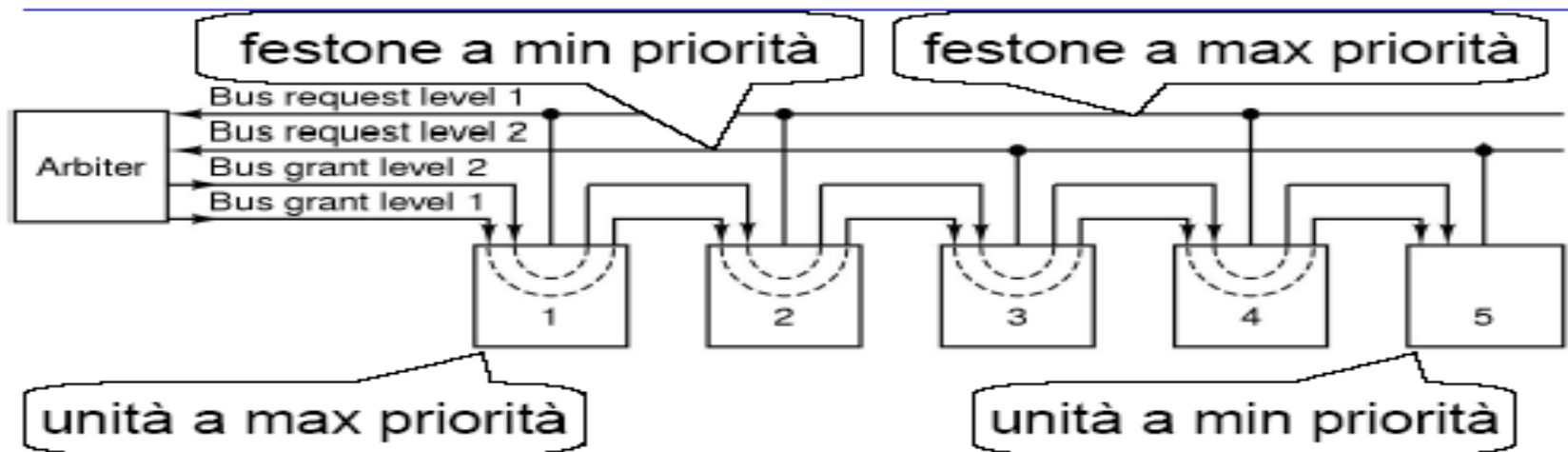
Per rendere più flessibile l’assegnazione delle priorità si possono usare più “festoni”, ciascuno con la possibilità di inviare una sua “richiesta” indipendente



Meccanismo di arbitraggio centralizzato flessibile: contiene più festoni di richiesta / conferma, posti a livelli di priorità differenti; ogni unità usa il festone cui è collegata

Daisy chain con più “festoni”

- Festone 1 che collega le unità 1, 2 e 4
- Festone 2 che collega le unità 3 e 5
- 2 linee di Bus Request, una per ciascun festone
- L'arbitro può cambiare dinamicamente la priorità dei 2 festoni. All'interno di ciascun festone la priorità è invece gestita in «daisy chain».



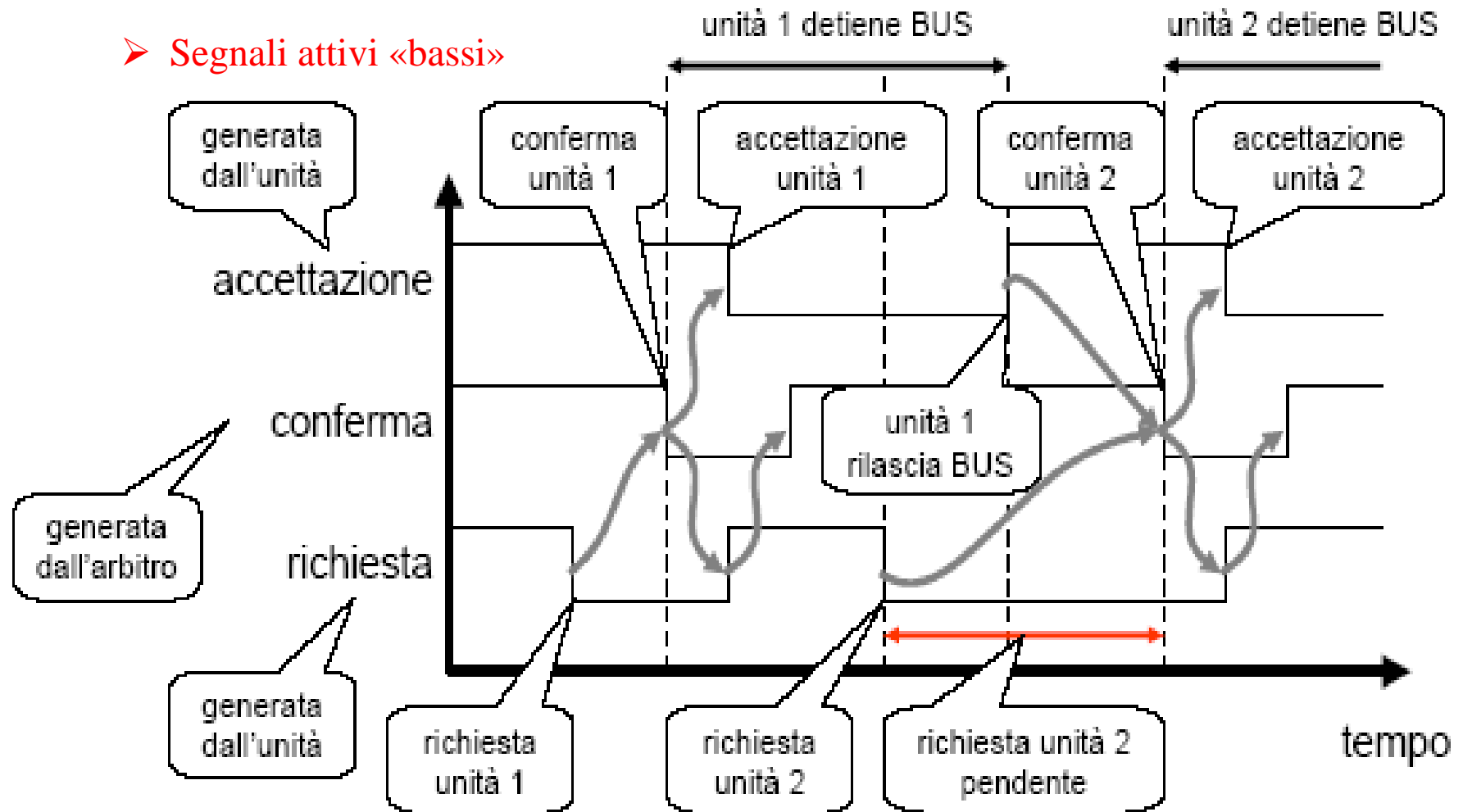
Meccanismo di arbitraggio centralizzato flessibile: contiene più festoni di richiesta / conferma, posti a livelli di priorità differenti; ogni unità usa il festone cui è collegata

Bus Acknowledge

- Nei meccanismi di arbitraggio descritti, le unità non sono in grado di sapere quando un'unità che aveva chiesto e ottenuto il controllo del BUS lo “rilascia”
- Si migliora l'efficienza dell'arbitraggio aggiungendo una terza linea di controllo: il segnale di “accettazione”: **BUS acknowledge (Bus Ack)**
- La linea di accettazione viene attivata dalle unità, non dall'arbitro:
 - ✓ un'unità richiede e ottiene dall'arbitro il controllo del BUS
 - ✓ non appena l'arbitro le invia la conferma, l'unità che prende il controllo del BUS attiva la linea di accettazione (**Bus Ack**) e disattiva la linea di richiesta
 - ✓ non appena l'arbitro vede attivarsi la linea di accettazione, disattiva la linea di conferma
- Così, subito dopo che l'unità ha preso il controllo del BUS, le linee di richiesta e di conferma sono già libere per un'altra unità può dunque inviare subito una nuova richiesta, che resterà «**pendente**»;
- Questa seconda unità riceverà la conferma non appena la prima unità avrà «**rilasciato**» **il controllo del BUS, disattivando la linea di accettazione (Bus Ack)**
- Il meccanismo di accettazione è più complesso, ma è più efficiente

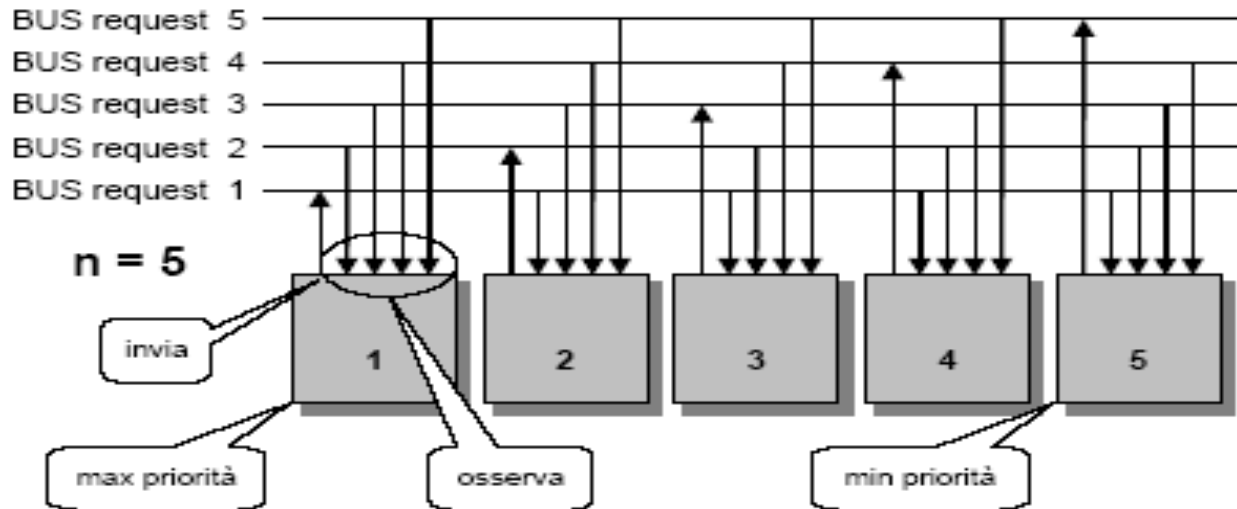
Bus Acknowledge: Esempio con 2 unità

➤ Segnali attivi «bassi»



Arbitraggio Distribuito

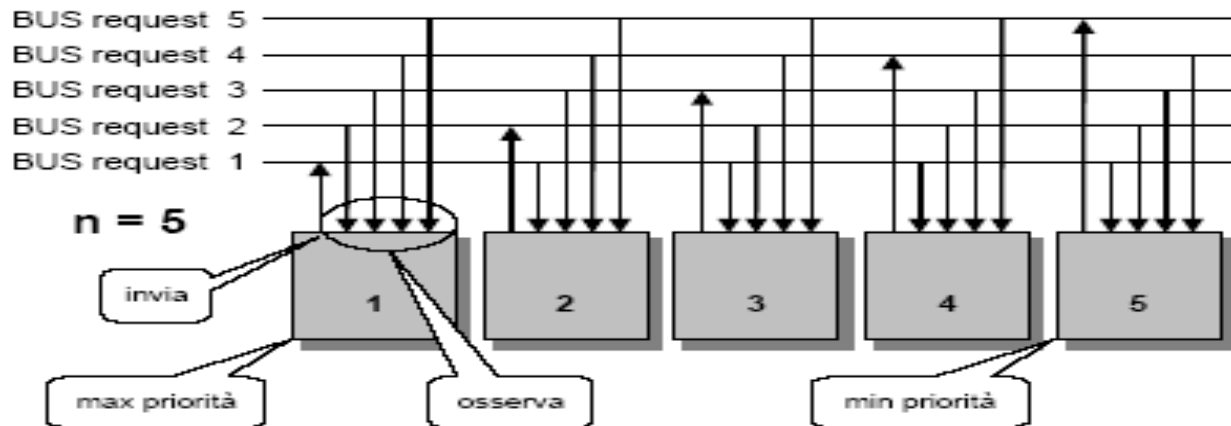
- Il meccanismo di arbitraggio distribuito a “n” linee prevede una linea di richiesta BUS per ogni unità. Non esiste alcun arbitro del BUS
- Le unità hanno priorità diverse e fissate. Le unità osservano tutte le linee di richiesta BUS
- Ogni unità può attivare solo la propria linea di richiesta BUS



Ogni unità è dotata di una propria linea di richiesta BUS e osserva lo stato delle linee di richiesta BUS di tutte le altre unità

Arbitraggio Distribuito

- Quando l'unità deve diventare MASTER, si accerta che nessun'altra unità a priorità superiore alla sua abbia attivato la propria linea di richiesta BUS
- Se così è, l'unità attiva la sua linea di richiesta BUS e ottiene il controllo del BUS, diventando MASTER
- Confronto: si risparmia il costo dell'arbitro, ma il numero di linee di controllo cresce con quello delle unità

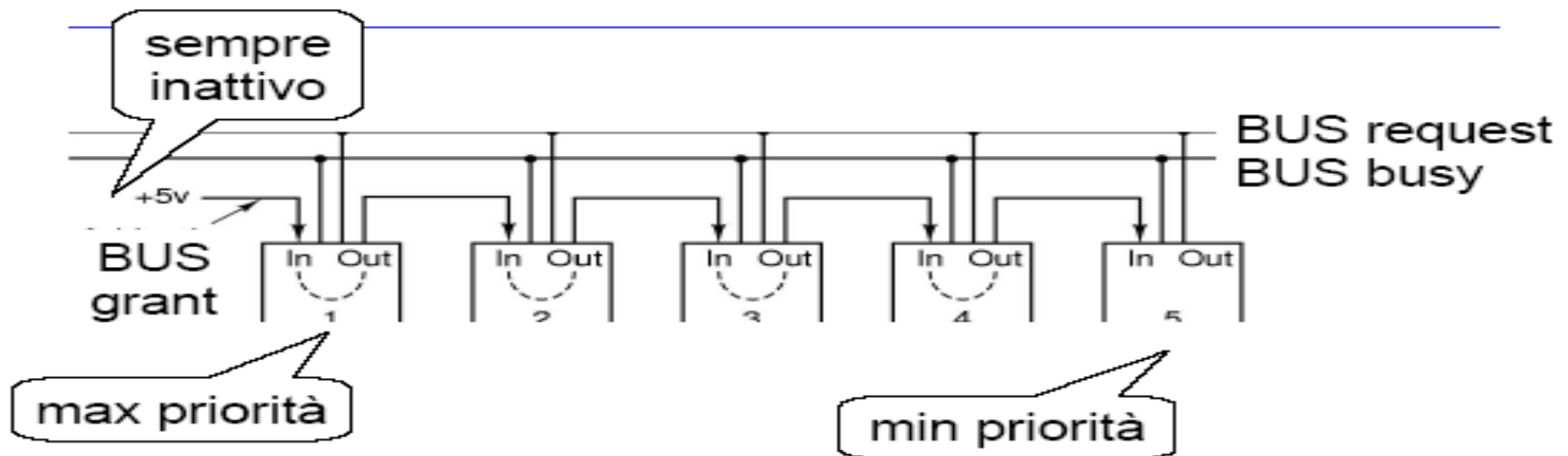


Ogni unità è dotata di una propria linea di richiesta BUS e osserva lo stato delle linee di richiesta BUS di tutte le altre unità

Arbitraggio Distribuito

➤ Si può migliorare l'arbitraggio distribuito a n linee usando solo 3 linee di controllo:

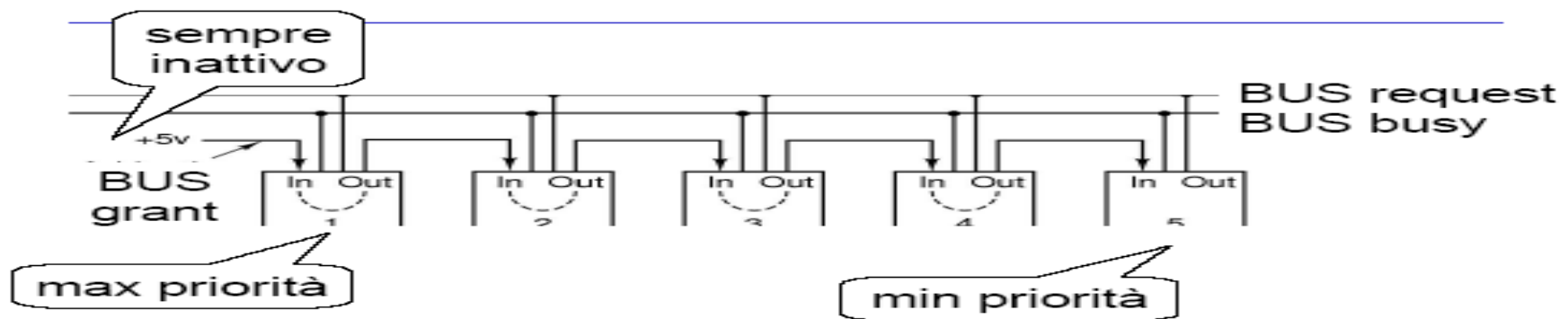
- BUS occupato (BUS busy): questa linea viene mantenuta attiva dall'unità che detiene correntemente il controllo del BUS
- conferma BUS (BUS grant): è la linea di conferma, collegata a festone
- richiesta BUS (BUS request): l'unità attiva questa linea per richiedere il BUS



Arbitraggio distribuito con schema di collegamento a 3 sole linee di controllo
(tutti i segnali sono attivi bassi)

Arbitraggio Distribuito con 3 linee

- Presupposto: un'unità che desideri diventare MASTER del BUS attende sempre di vedere: il segnale di impegno BUS inattivo e il proprio ingresso inattivo (NB: segnali attivi "bassi")
 - prima di avanzare la propria richiesta di controllo del BUS
- In altri termini: l'unità si accerta che un'altra unità a priorità superiore non sia correntemente il MASTER del BUS
- Notare che il Bus Grant iniziale è sempre inattivo (+5 volt, segnali attivi bassi) e quindi la prima unità (a massima priorità) ha sempre la possibilità di attivare la propria richiesta (Bus request) per prima



Arbitraggio distribuito con schema di collegamento a 3 sole linee di controllo
(tutti i segnali sono attivi bassi)

Arbitraggio Distribuito con 3 linee

- Inoltre tutte le unità osservano la regola seguente: se l'ingresso IN è attivo, anche l'uscita OUT va tenuta attiva, indipendentemente dal resto
- In altri termini: se un'unità vede che un'altra unità a priorità superiore detiene il controllo del BUS, passa questa informazione a tutte le unità con priorità inferiore alla propria
- Quando si verificano le condizioni che consentono di avanzare una richiesta di controllo del BUS: l'unità attiva la linea di richiesta BUS, attiva la linea di impegno BUS e l'unità attiva la propria uscita OUT che sarà propagata alle altre unità con priorità inferiore alla propria
- Così l'unità ottiene il controllo del BUS, diventando MASTER, e notifica il fatto a tutte le unità di priorità inferiore alla propria (poi disattiva la richiesta BUS)
- Il meccanismo di arbitraggio distribuito a 3 linee di controllo è ovviamente più semplice di quello a «n» linee di controllo

Funzionamento e TempORIZZAZIONE del BUS

- Le attività del calcolatore si sviluppano per **cicli di BUS**: in ogni ciclo avviene un'operazione (o transazione)
- Le operazioni sono governate dal **MASTER** che detiene il controllo del BUS
- Gli **SLAVE** non possono dare inizio a un'operazione in modo autonomo
- **Operazioni**: trasferimento di dati tra MASTER e SLAVE
- Operazione di lettura: un dato viene trasferito da uno SLAVE al MASTER
- Operazione di scrittura: un dato viene trasferito dal MASTER a uno SLAVE
- Nota bene: il punto di vista per stabilire la direzione del trasferimento (lettura o scrittura) è sempre e solo quello del MASTER che detiene (in quel momento) il controllo del BUS

Temporizzazione del Bus

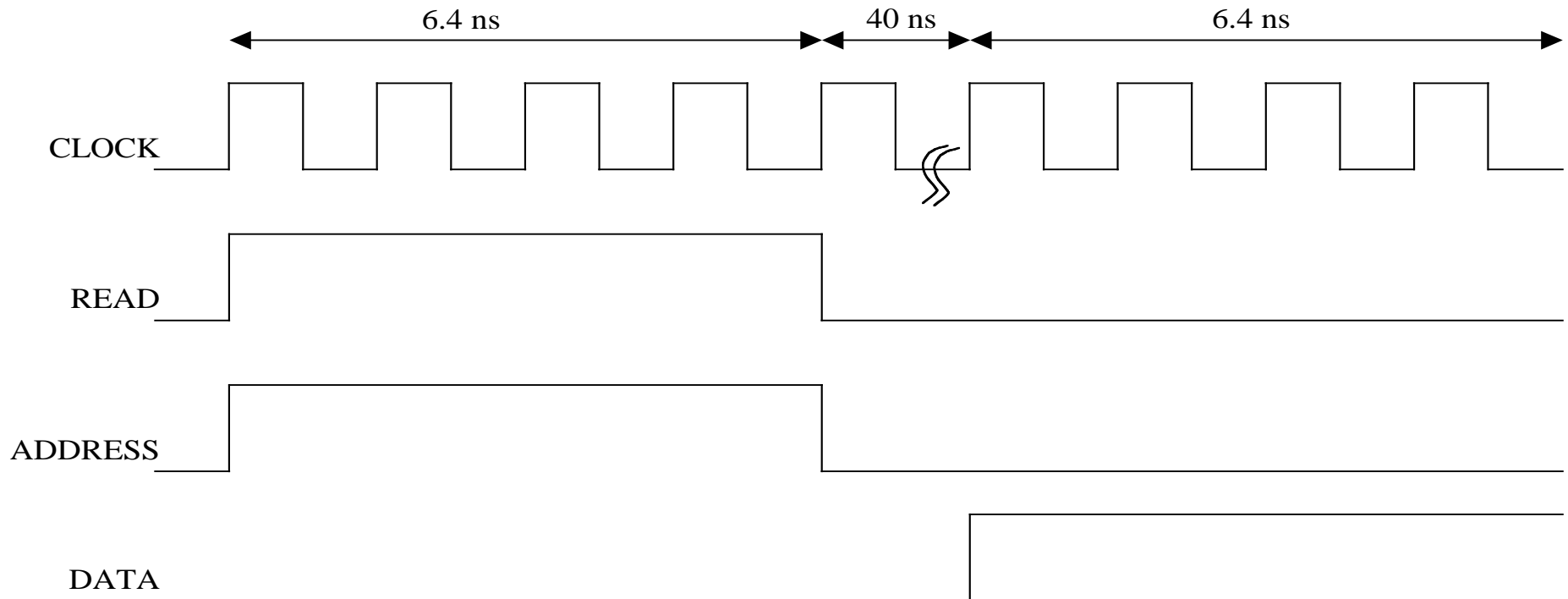
- La temporizzazione è il modo con cui gli eventi (lettura, scrittura, ecc.) sono coordinati sul Bus
- Temporizzazione **Sincrona**
 - Eventi determinati da un “clock”. Bus cycle=Clock Cycle
 - Tutto sincronizzato sui “fronti” del clock
 - Molti “eventi” occupano uno o più cicli di clock
- Temporizzazione **Asincrona**
 - L’occorrenza di un evento dipende da quella dell’evento precedente
- Modalità sincrona è più semplice ma meno flessibile. E comunque la più usata.
- Modalità asincrona consente a dispositivi di diversa “velocità” di condividere efficacemente il Bus

Esempio di “Lettura” Sincrona Semplificata

Clock a 625 MHz. Ogni trasmissione sul bus impiega 4 cicli di clock = 6.4 ns

Tempo di ciclo della memoria (tempo che ipotizziamo serva alla memoria per posizionare sul bus i dati richiesti dalla lettura): 40 ns

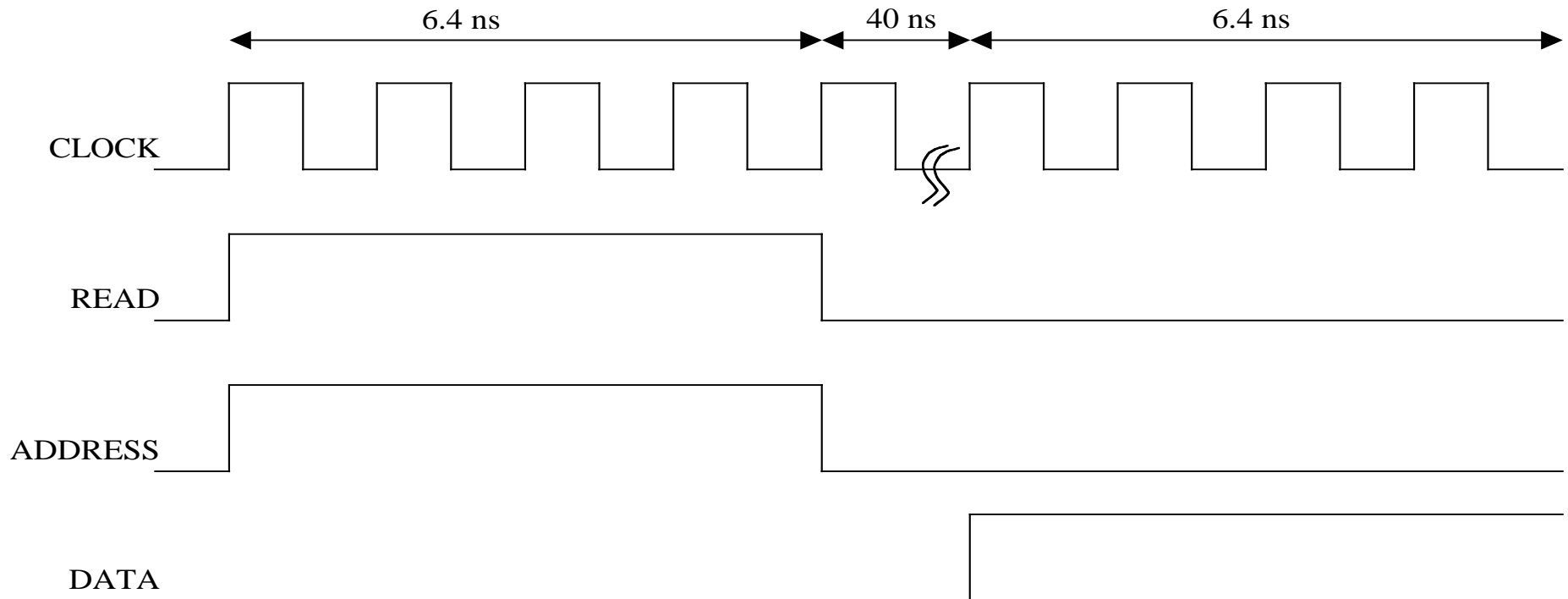
Tempo di lettura di una parola da 32 bit su bus da 32 bit: 52.8 ns



Esempio di “Lettura” Sincrona Semplificata

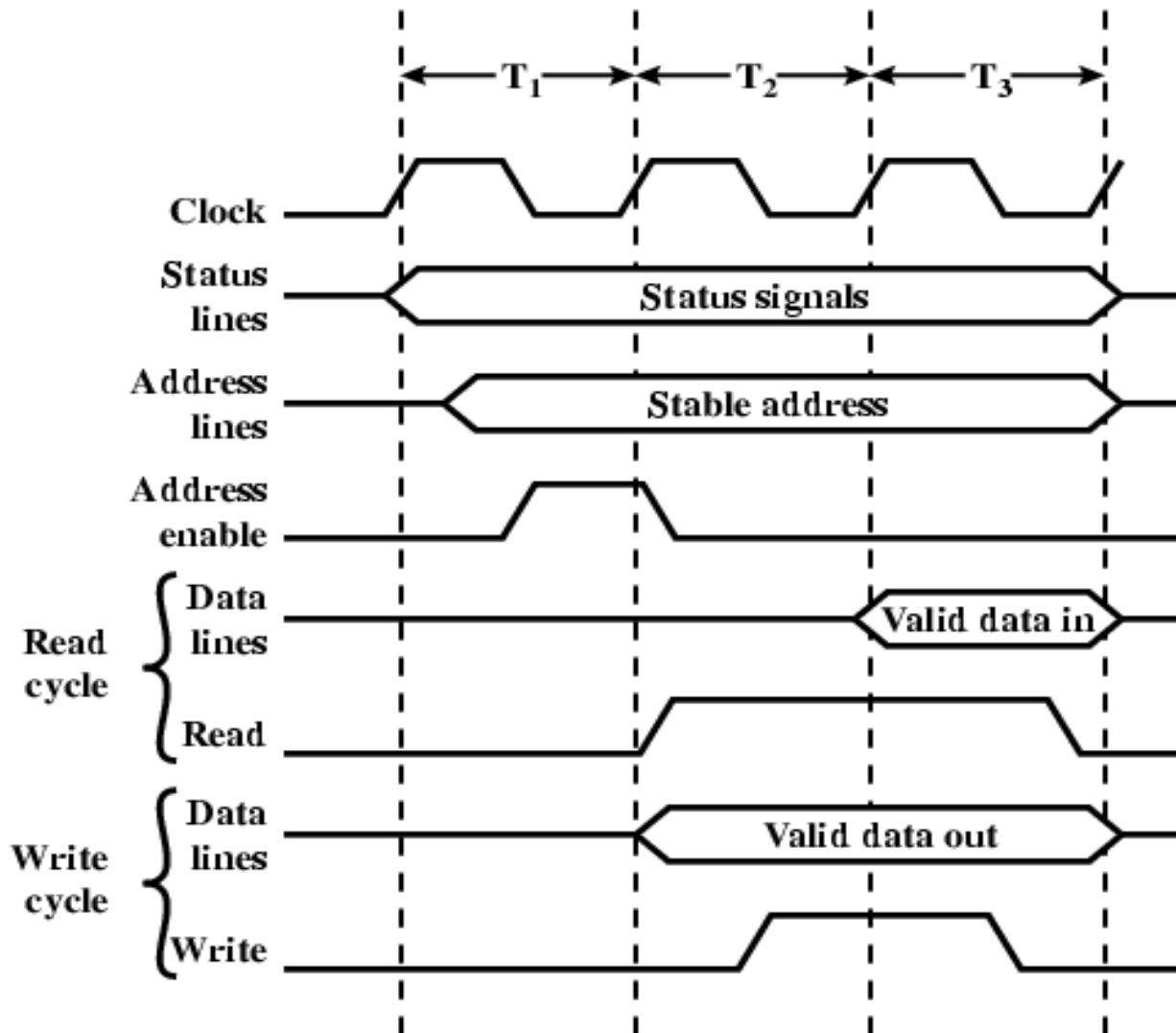
Tempo di lettura di una parola da 32 bit su bus da 32 bit: 52.8 ns

- Stiamo assumendo che ogni trasmissione di segnali di controllo (Read), indirizzi, e dati impieghi 4 cicli di clock per essere trasmessa sul bus.



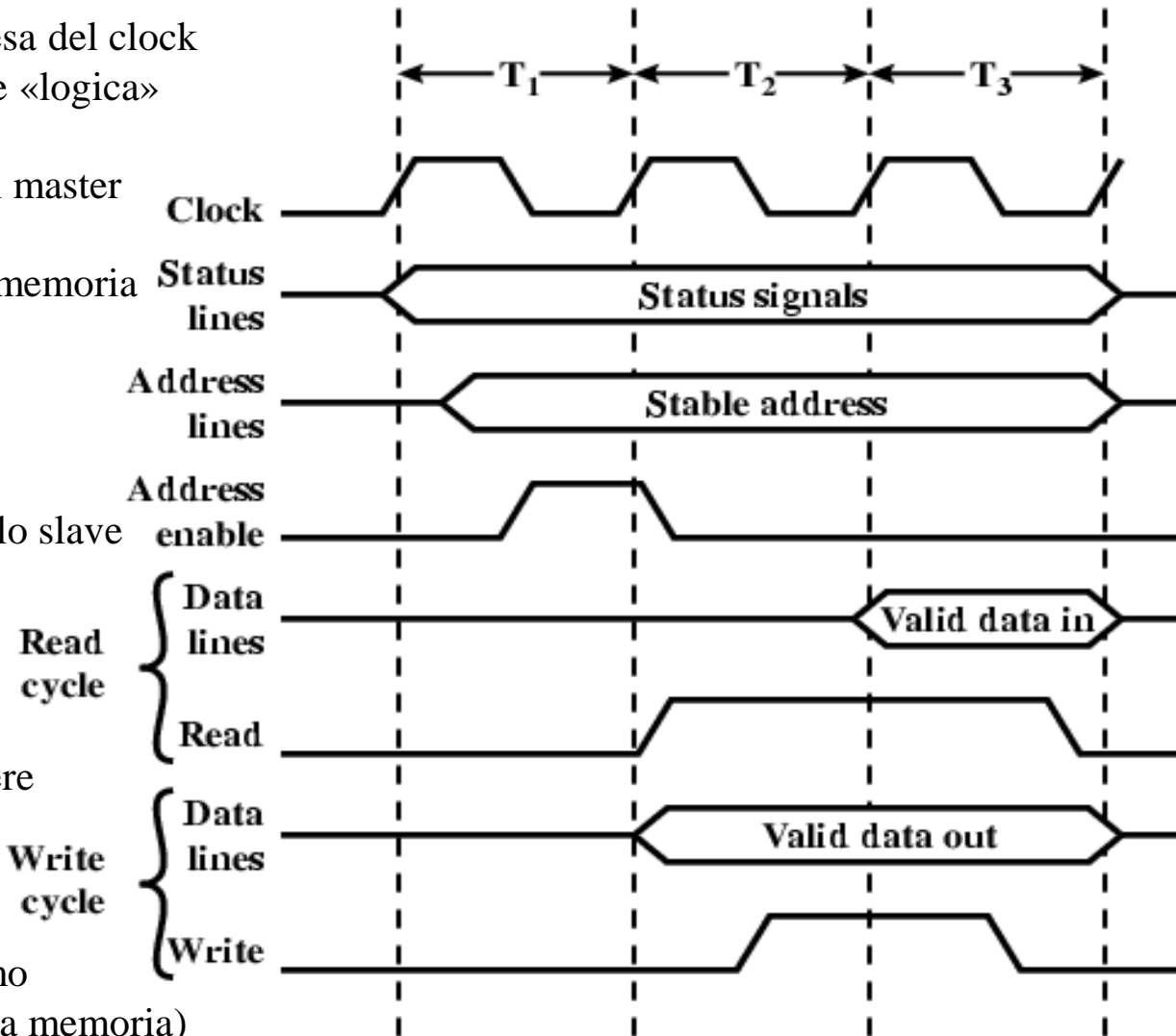
Esempio di Temporizzazione Sincrona

- Questo esempio serve ad illustrare dei possibili segnali che vengono scambiati durante delle operazioni di lettura/scrittura in modalità sincrona
- E' ovviamente solo un esempio per scopi didattici
- Diversi bus impiegano diversi segnali, con diversi nomi e possono avere modalità di scambio segnali diverse
- Ma resta sempre vero che tutto è «temporizzato» dal bus nella modalità sincrona



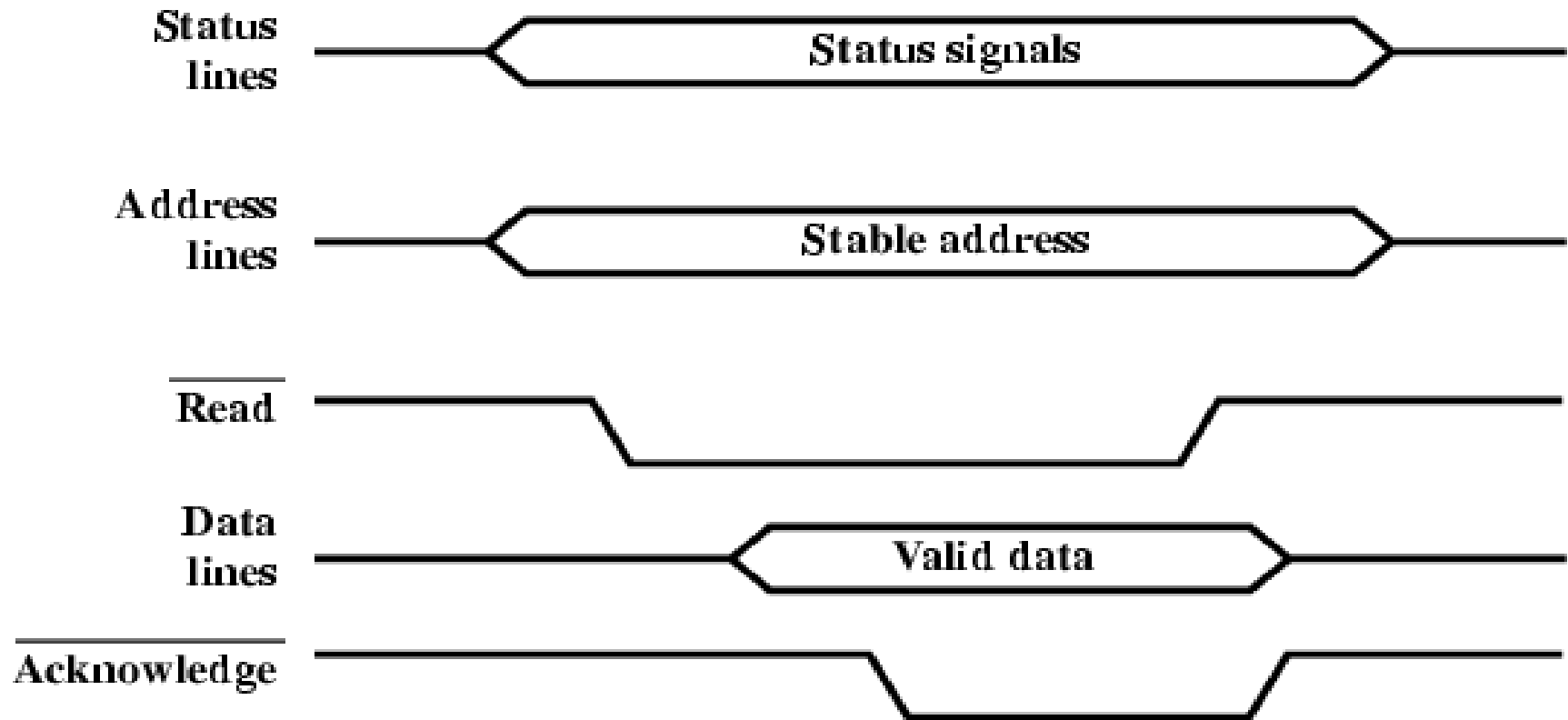
Esempio di Temporizzazione Sincrona

- «pendenza» fronte di salita/discesa del clock non ha importanza per la trattazione «logica» di questo corso
- Durante il primo ciclo di clock il master (ad es. la CPU) pone sul bus gli indirizzi (ad es. di una locazione di memoria da leggere). Al tempo stesso potrebbe inviare dei segnali di stato (non di interesse in questo esempio)
- In questo esempio il Segnale di «address enable» serve a indicare allo slave che gli indirizzi sono stabili e li può leggere
- Sul secondo ciclo di clock può avvenire READ o WRITE
- Per WRITE prima bisogna mettere sul bus i dati da scrivere
- Nel caso del READ otteniamo i dati sul bus dopo un ritardo di un ciclo (il tempo che prima avevamo ipotizzato pari al tempo di ciclo della memoria)



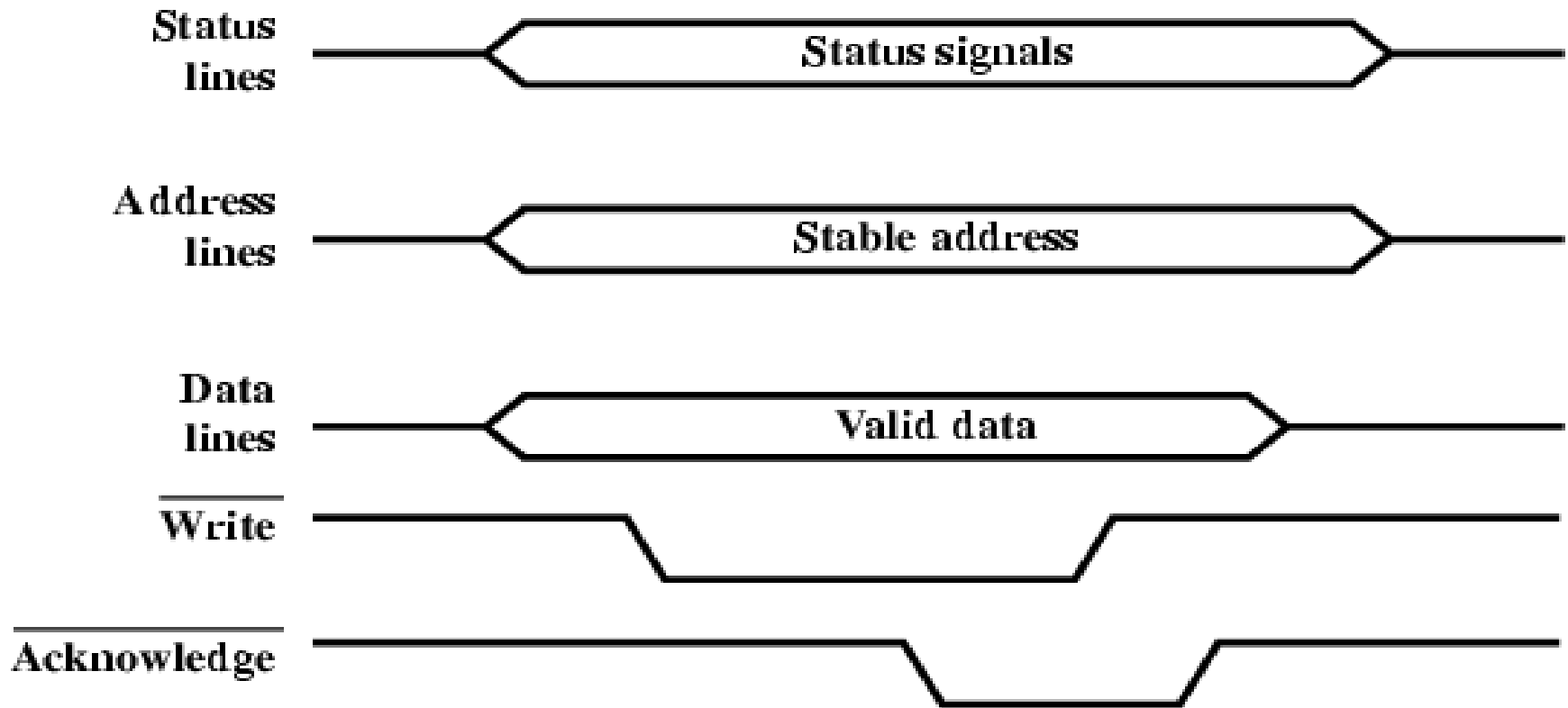
Esempio di “Lettura” Asincrona

- Segnali di Read e Acknowledge attivi «bassi»
- Lo Slave (ad es. memoria) segnala con «Acknowledge» al master che i dati che voleva leggere sono disponibili sul bus



Esempio di “Scrittura” Asincrona

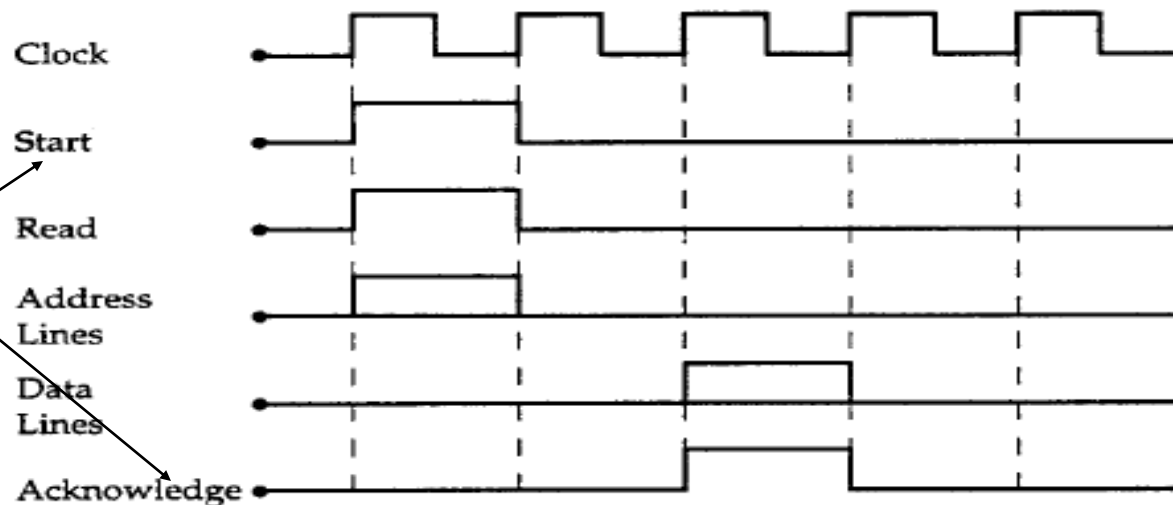
- Segnali di Read e Acknowledge attivi «bassi»
- Slave (ad es. memoria) segnala con «Acknowledge» al master di avere «copiato» i dati che master voleva «scrivere»



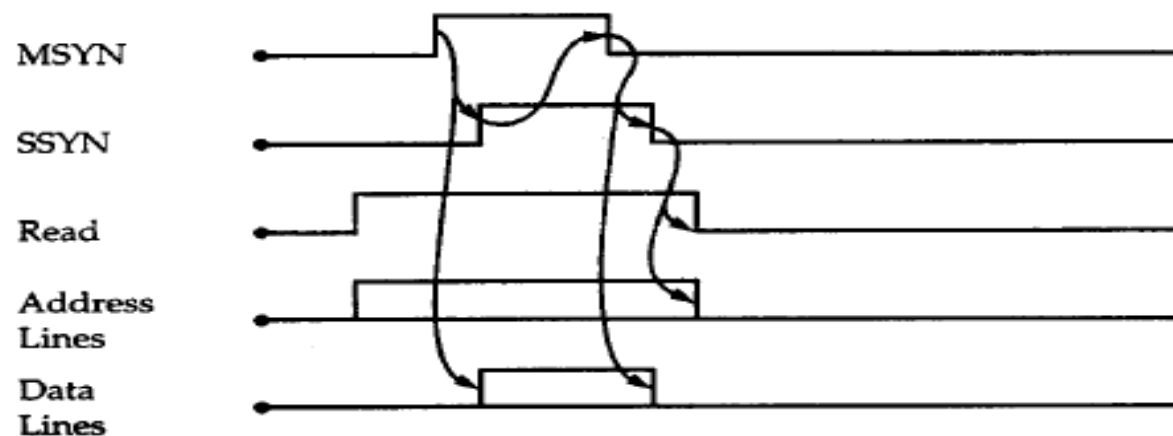
Esempio di Temporizzazione Sincrona e Asincrona

N.B. anche nel caso sincrono possiamo avere dei segnali di controllo di “sincronismo”.

Serve a rendere maggiormente sicura la comunicazione, anche in presenza di «errori» di sincronismo sul clock



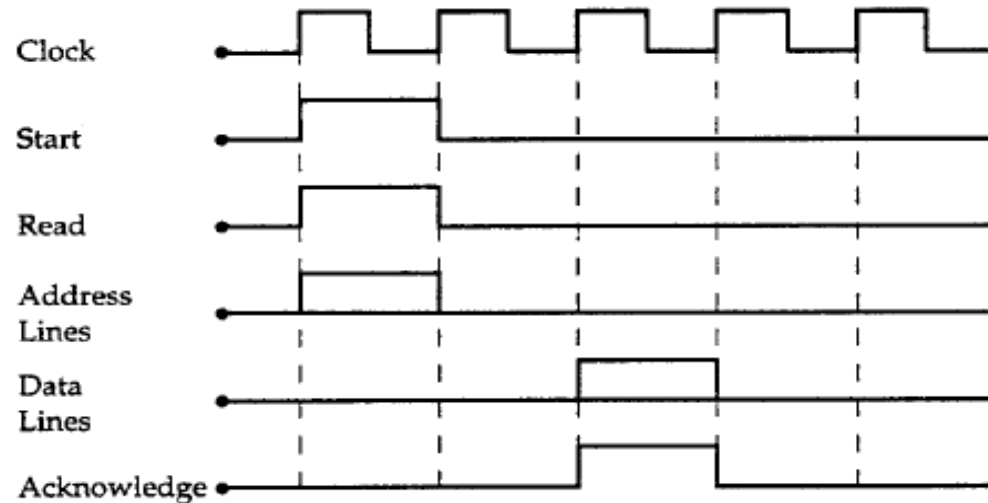
(a) Synchronous Timing



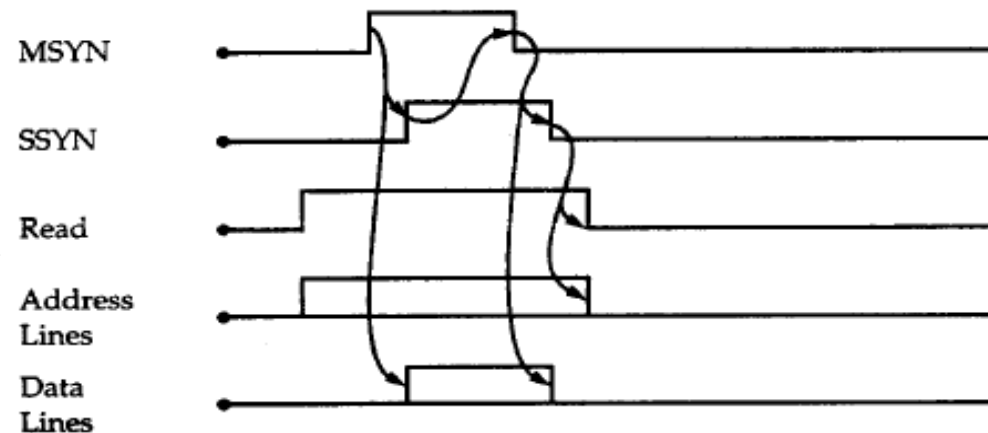
(b) Asynchronous Timing

Esempio di Temporizzazione Sincrona e Asincrona

- MSYN: segnale di sincronismo del Master
- SSYN: segnale di sincronismo dello Slave
- Master manda READ e Indirizzi e segnala operazione a slave con segnale controllo MSYN
- Slave mette dati richiesti in lettura sul bus e fornisce «acknowledge» con segnale SSYN



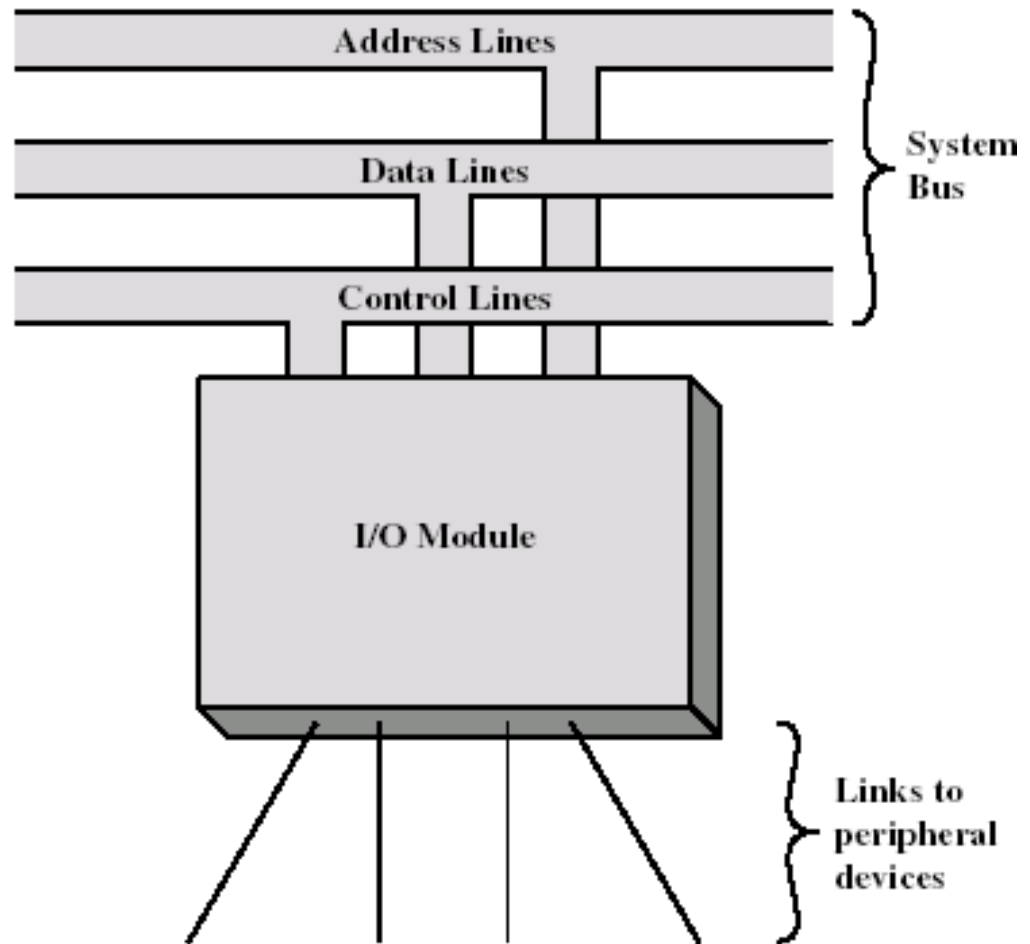
(a) Synchronous Timing



(b) Asynchronous Timing

Introduzione alle Modalità di I/O

- Unità di I/O = Insieme di Moduli di I/O
- Ciascuna modulo di I/O copre due funzioni fondamentali:
 - controlla una o più periferiche connesse al sistema
 - comunica dati da/alla periferica a/dagli altri moduli dell' architettura



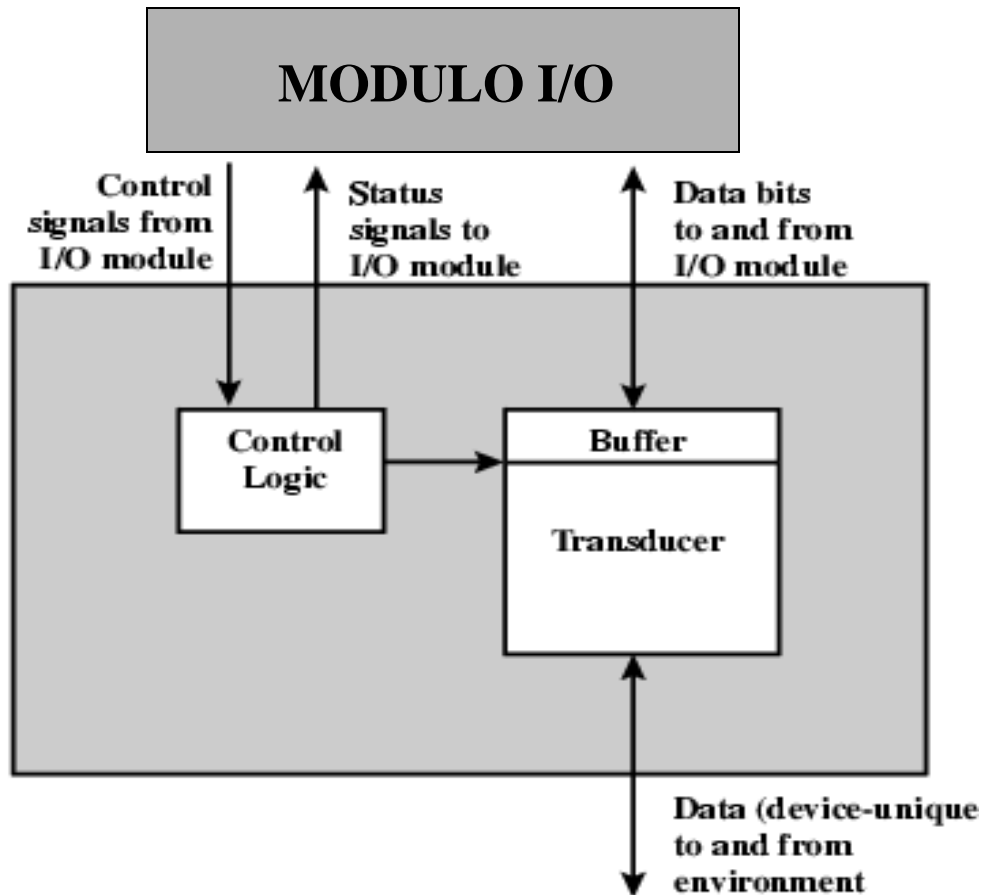
Moduli di I/O: necessità

- Perché le periferiche non vengono connesse direttamente al bus di sistema?
- Esiste una grande varietà di periferiche
 - in termini di funzionalità
 - stampante, modem, etc.
 - in termini di velocità di trasferimento
 - Da decine di bit/sec (tastiera) a Gigabit connessione rete
 - Periferiche sempre più lente di CPU o memoria
 - in termini di formato e lunghezza dei dati
 - bisognerebbe adottare una logica diversa per ogni formato
- E' dunque necessario un modulo che faccia da "interfaccia" tra CPU, memoria e periferiche

Periferiche molte diverse

- Tipo “human readable”
 - video, stampanti, tastiera
 - Tipo “machine readable”
 - dischi, smart phone, ecc.
 - Tipo “communication”
 - Modem, ecc.
- Possono quindi servire diverse “interfacce” (moduli I/O) per i diversi tipi di periferiche

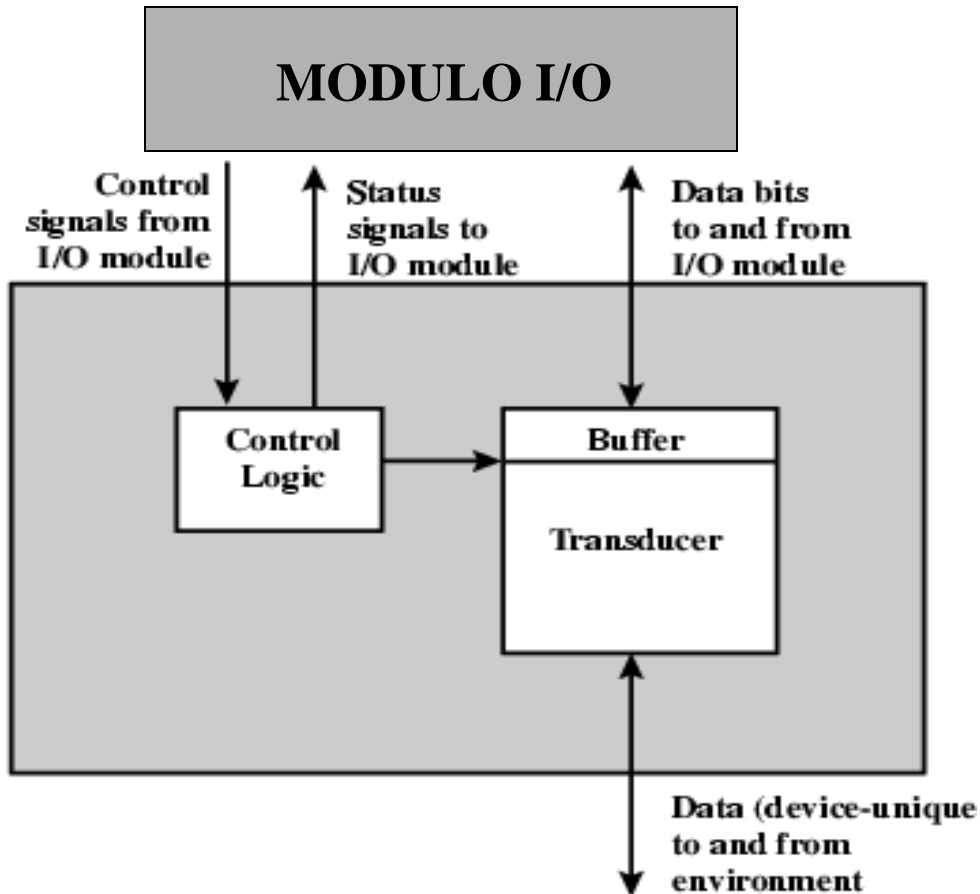
Schema funzionale di una periferica



- Control signals:
 - READ/WRITE
- Data bits:
 - informazioni da/alla periferica
- Status signals
 - Es. READY/NOT READY
- Control Logic
 - pilota le operazioni del dispositivo (es. Leggi/scrivi, invia segnali di stato come "carta inceppata", etc.)
- Transducer
- Buffer

➤ Ai diversi tipi di interfacce fra modulo I/O e periferiche faremo un cenno più avanti

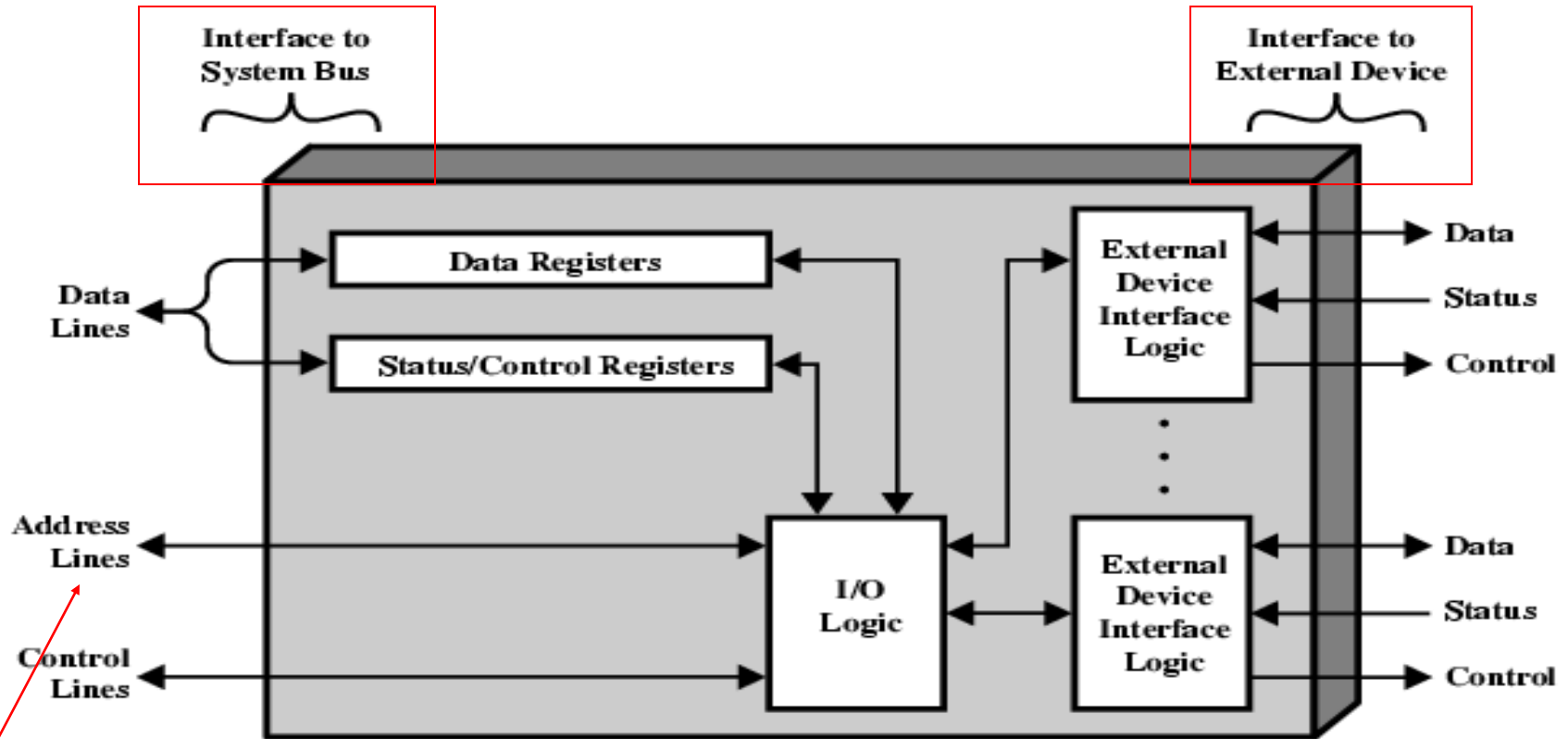
Schema funzionale di una periferica



➤ Vedere «slide» successiva per schema funzionale Modulo I/O

- Transducer=trasduttore, tipicamente memorizza in un buffer (memoria "tampone") i dati in corso di trasferimento
- In una normale tastiera la pressione di un tasto genera un segnale elettrico che viene trasdotto (codificato) in una sequenza di 8 bit secondo il codice ASCII

Struttura funzionale del modulo I/O



Le periferiche “pilotate” da un modulo di I/O rispondono ad un certo insieme di indirizzi

Circuiti logici di controllo per ciascuna delle periferiche. N.B. più periferiche sono associate ad un unico modulo di I/O

Modalità di I/O

- **I/O programmato**

- controllo totale delle operazioni di I/O da parte del processore, mediante l'esecuzione diretta di opportune istruzioni di I/O

- **Pilotato da interruzioni**

- un segnale interrompe l'esecuzione di un programma per avviare un'opportuna "procedura" di esecuzione delle operazioni di I/O

- **Accesso Diretto alla Memoria** ("Direct Memory Access", **DMA**)

- controllo ed esecuzione delle operazioni di I/O da parte di un modulo dedicato, diverso dal processore, che può accedere direttamente alla memoria

Sommario delle Tecniche di I/O

Table 7.3 I/O Techniques

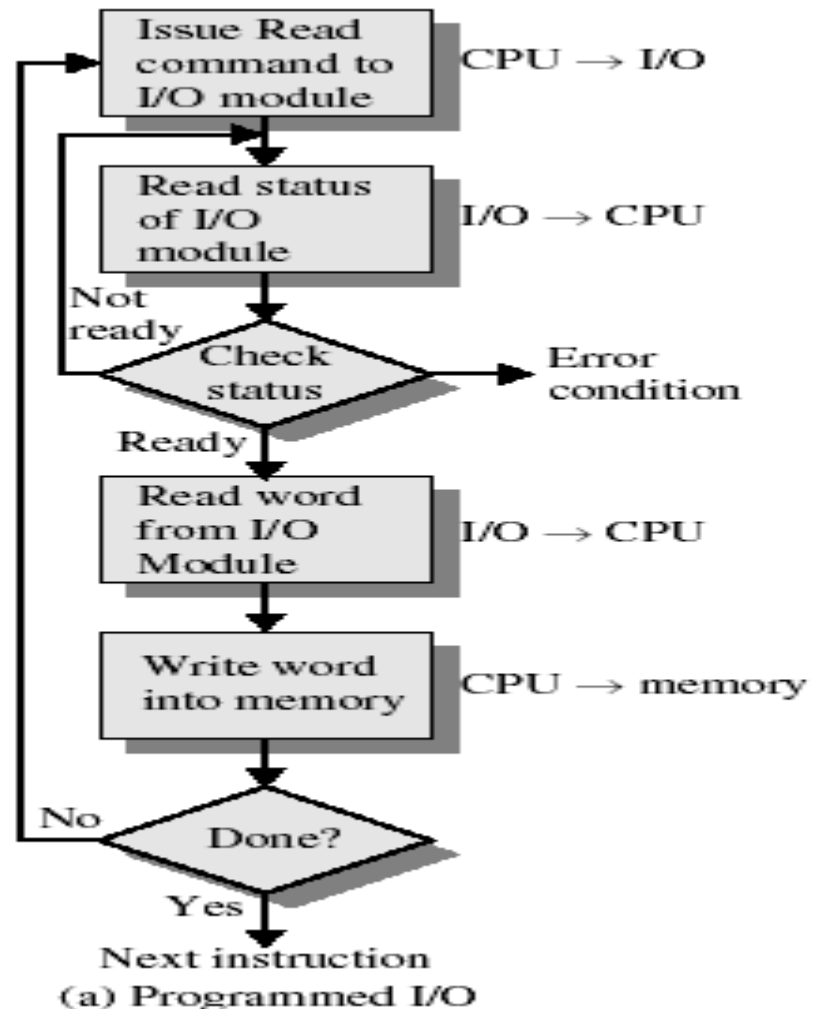
| | No Interrupts | Use of Interrupts |
|---|----------------------|----------------------------|
| I/O-to-memory transfer through processor | Programmed I/O | Interrupt-driven I/O |
| Direct I/O-to-memory transfer | | Direct memory access (DMA) |

I/O programmato

- La responsabilità delle comunicazioni I/O e del trasferimento dati è interamente del “programmatore”, ovvero della CPU
- Il processore invia/riceve dati direttamente dal modulo I/O, e controlla periodicamente lo stato della periferica
- La velocità di trasferimento dipende dalla periferica
- Il processore attende che la periferica completi il suo compito prima di proseguire l’esecuzione del programma

Esempio di flusso di I/O programmato (“read”)

- Esempio di lettura di una parola
- Tutte le istruzioni nel diagramma di flusso sono eseguite dalla CPU, e devono essere previste dal “programmatore”



I/O programmato: le istruzioni alla periferica

- Le istruzioni abitualmente usate per il trasferimento dati dai registri alla memoria vengono facilmente utilizzate come istruzioni I/O
- Per identificare in modo univoco la periferica, le si associa un indirizzo
- In tal modo è possibile “mapparla” in memoria associandole uno dei possibili indirizzi della linea indirizzi del bus (“memory-mapped” I/O)
- Altrimenti, si può modificare il formato dell’istruzione in modo da distinguere indirizzi di memoria e indirizzi di periferica (“isolated” I/O)

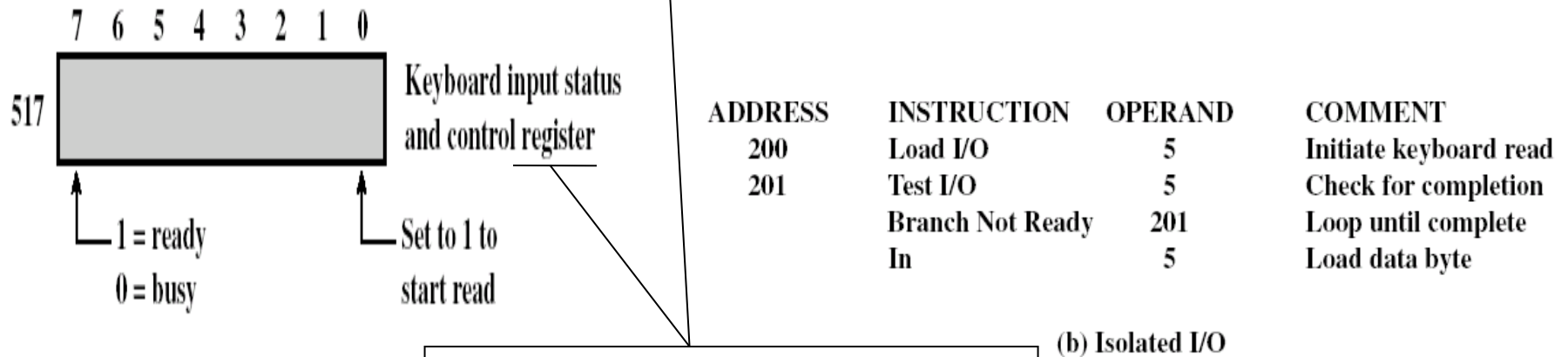
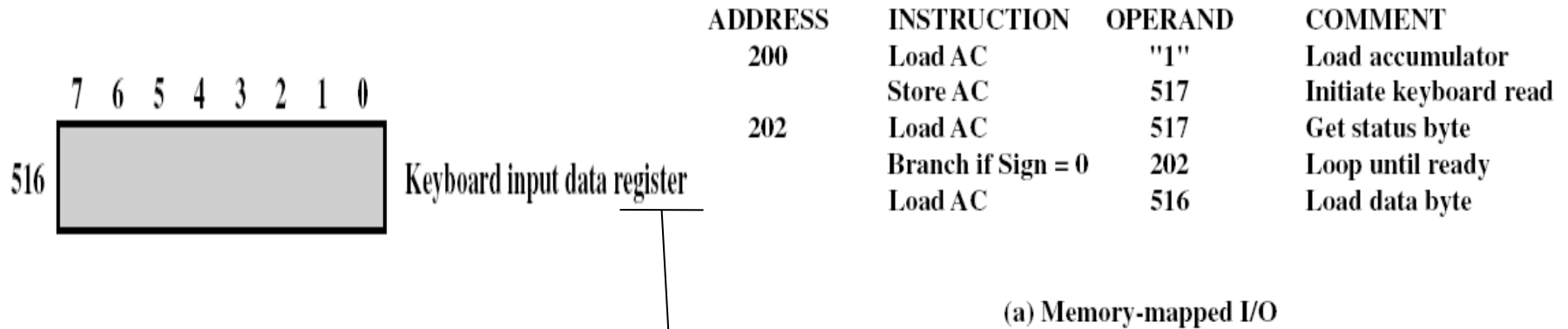
“Memory-mapped” Un esempio

I/O

e

“Isolated”

I/O:



Memory-mapped I/O e Isolated I/O: vantaggi e svantaggi

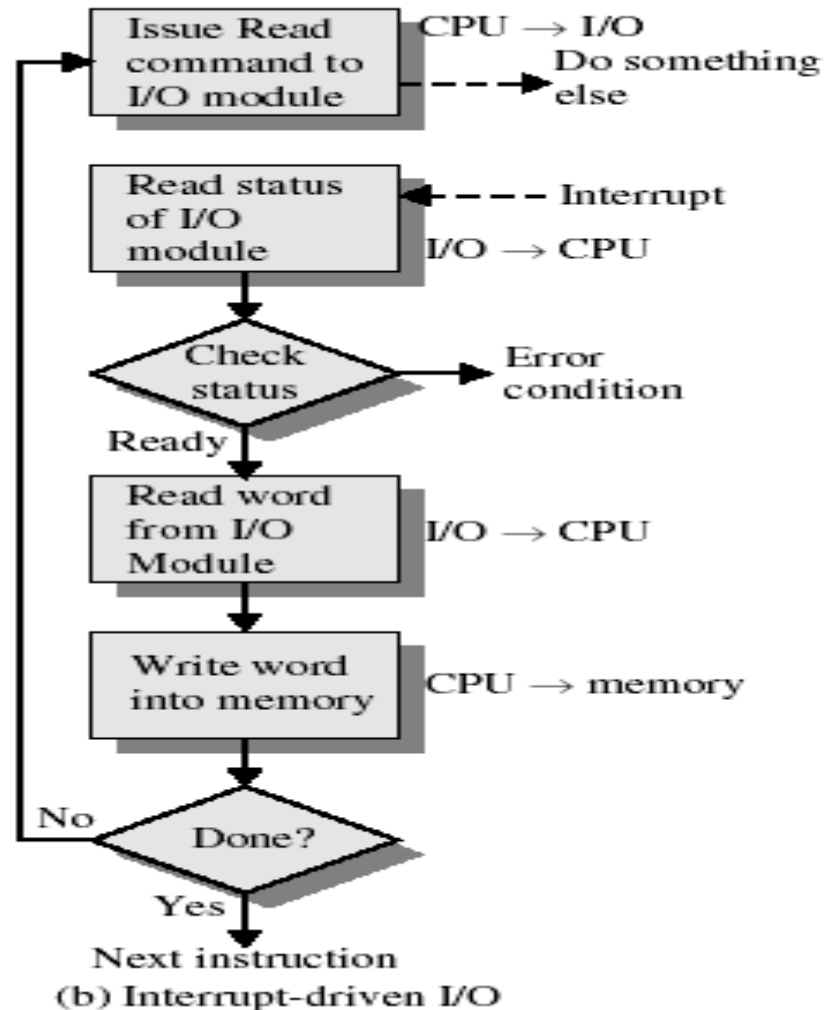
- Memory-mapped I/O
 - sfrutta appieno le potenzialità delle istruzioni di macchina
 - riduce la memoria disponibile
- Isolated I/O
 - istruzioni “dedicate”
 - garantisce la piena disponibilità della memoria

I/O pilotato con interruzioni

- Nell'attesa che la periferica risponda, il processore può fare altro lavoro utile
- Quando la periferica è pronta, avverte il processore con un segnale sul bus di controllo (segnale di "interruzione")
- Il processore interrompe quanto sta facendo, ed esegue una opportuna procedura per la gestione dell'interruzione, ovvero dell'operazione di I/O richiesta
- Dopo aver compiuto l'operazione di I/O, la CPU riprende l'esecuzione del programma interrotto

Esempio di I/O a mezzo interruzioni (“read”)

- Esempio di lettura di una parola
- La CPU deve verificare se il modulo di I/O ha inviato un segnale di “interruzione”

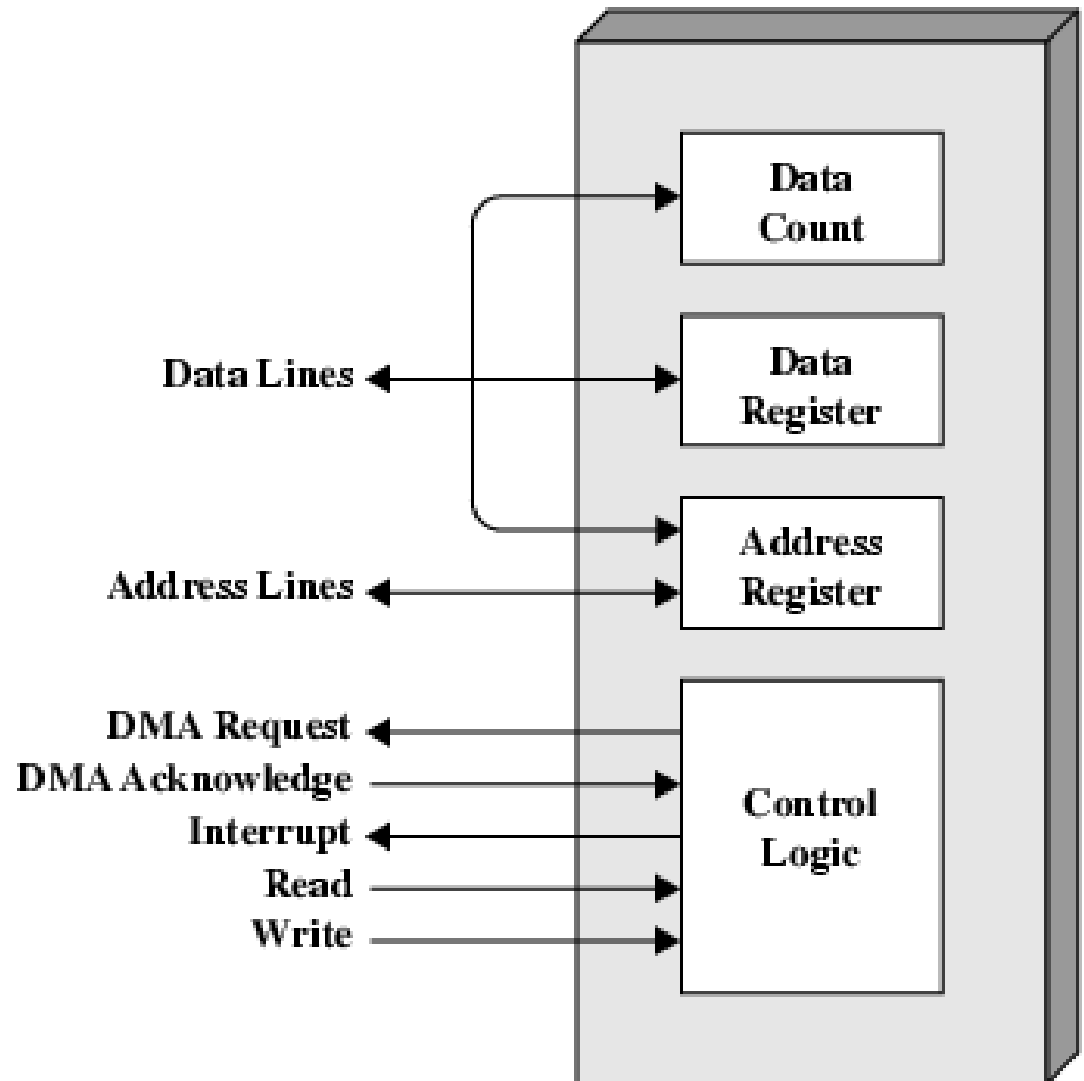


Direct Memory Access (DMA)

- Limiti del I/O programmato e del I/O pilotato con interruzioni
 - la velocità di trasferimento è limitata dalla velocità con cui il processore può “verificare” e “servire” una periferica
 - le istruzioni di I/O sono comunque eseguite dal processore
- Il modulo per l'Accesso Diretto alla Memoria (DMA) supera tali limiti, soprattutto quando il numero di trasferimenti è elevato

Il modulo DMA

- E' un modulo hardware aggiuntivo al modulo I/O
- Richiede in Input:
 - la locazione iniziale su/da cui trasferire i dati
 - il numero di parole da trasferire
 - l'indirizzo del relativo modulo di I/O



Algoritmo base con il DMA

- Il DMA riceve dalla CPU le informazioni necessarie per il trasferimento
- Il processore prosegue nell'esecuzione del programma in corso
- Nel frattempo il DMA effettua il trasferimento richiesto
- Al termine del trasferimento il DMA emette un'interruzione per segnalare al processore che l'operazione di I/O è conclusa

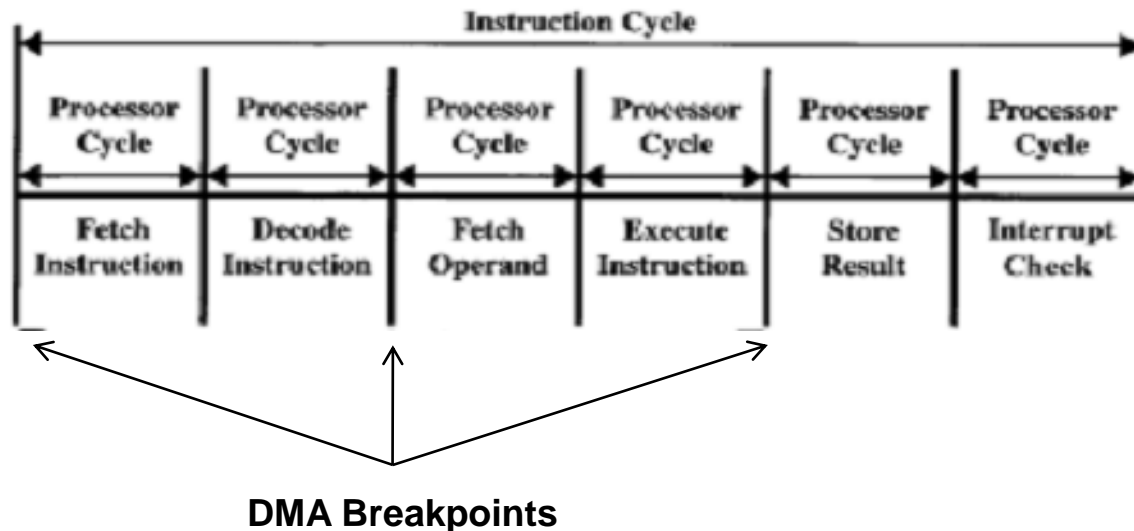
Modalità di trasferimento con DMA

- Block transfer
 - la modalità più veloce
 - il DMA diventa “master” del bus per il trasferimento dei dati
 - il processore ne riprende il controllo solo a trasferimento avvenuto
 - utile per unità I/O del tipo disco o stampanti
- Transparent
 - la modalità più lenta
 - il DMA effettua un trasferimento solo quando il bus non serve al processore

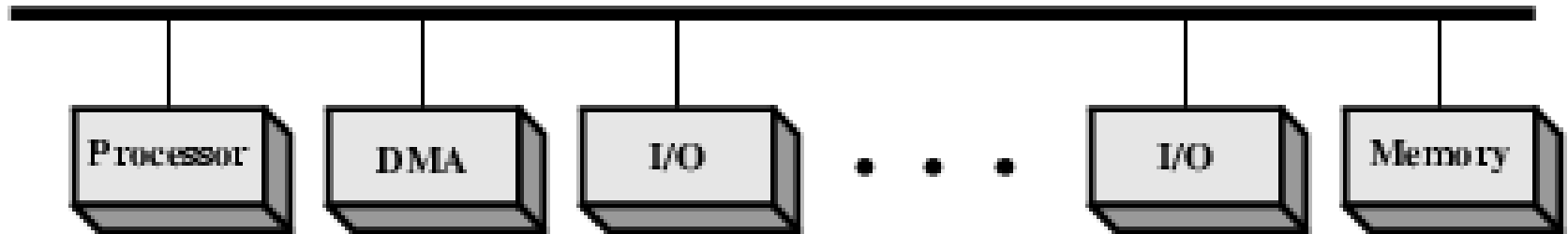
Modalità di trasferimento con DMA

- **Furto di ciclo**

- il trasferimento avviene in determinati punti del ciclo di esecuzione delle istruzioni
- il processore viene interrotto prima che acceda al bus, del quale il DMA assume il controllo per un ciclo

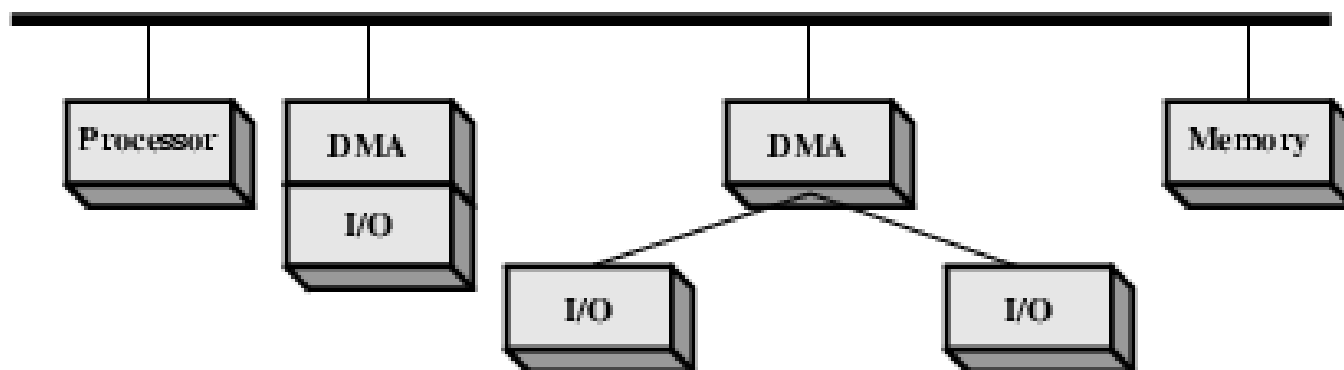


Configurazioni del modulo DMA (1)

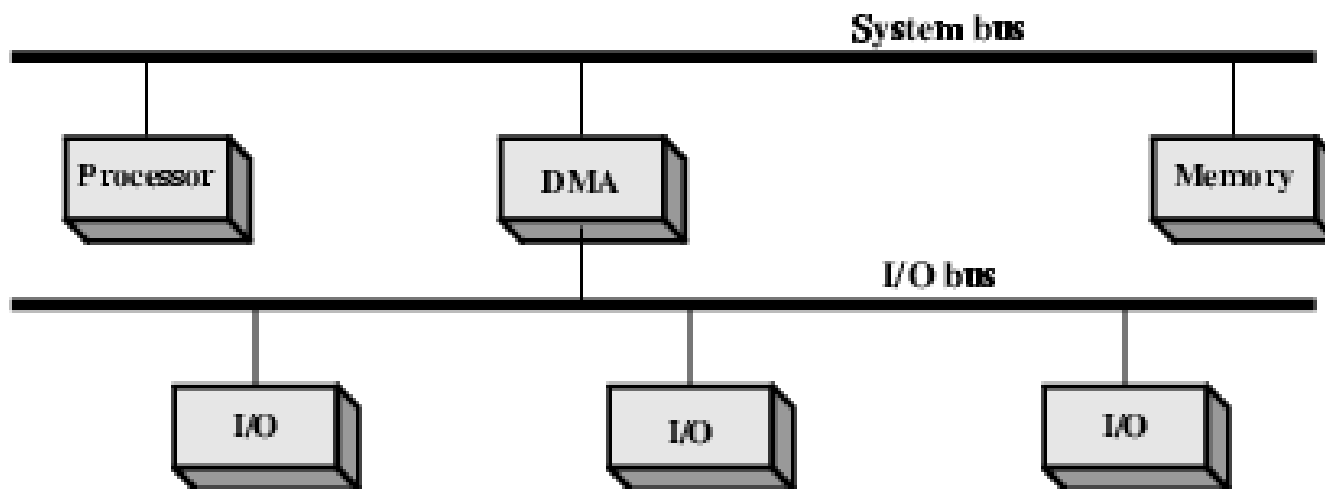


- Il DMA funziona da “secondo processore”
- Ogni trasferimento richiede due cicli di clock

Configurazioni più efficienti del modulo DMA (2)

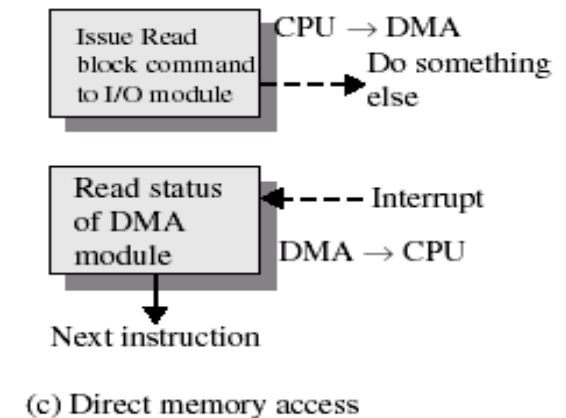
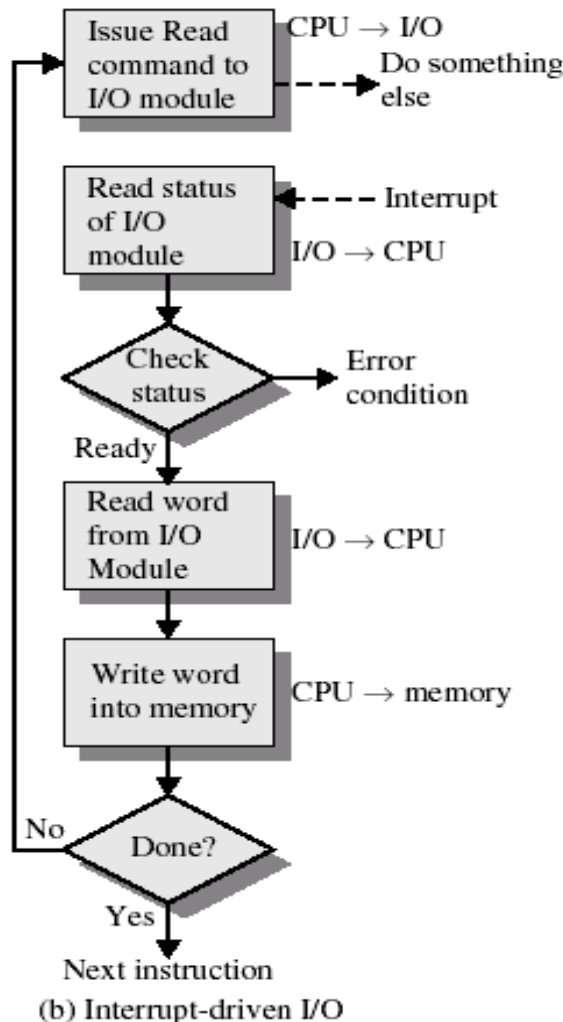
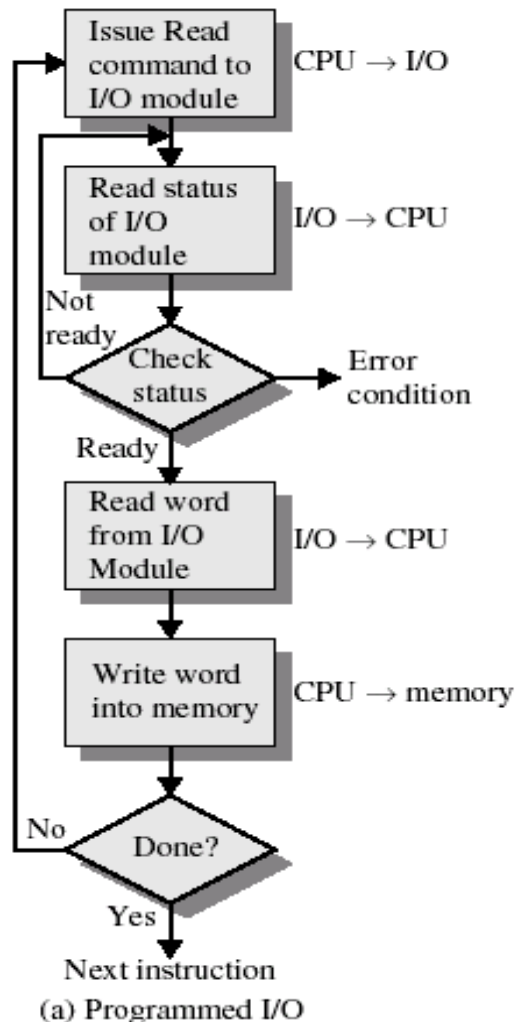


(b) **Single-bus, Integrated DMA-I/O**



(c) **I/O bus**

I/O programmato, I/O con interruzioni e DMA a confronto



Evoluzione della gestione del I/O

- Controllo totale da parte della CPU
- I/O programmato
- I/O con interruzioni
- I/O con DMA
- Canali I/O: il modulo di I/O viene dotato di un processore in grado di eseguire un proprio set di istruzioni I/O senza l'intervento della CPU
- Processori I/O: il canale I/O viene dotato di memoria locale
 - molto usati nel controllo di terminali interattivi

Interfacciamento delle periferiche

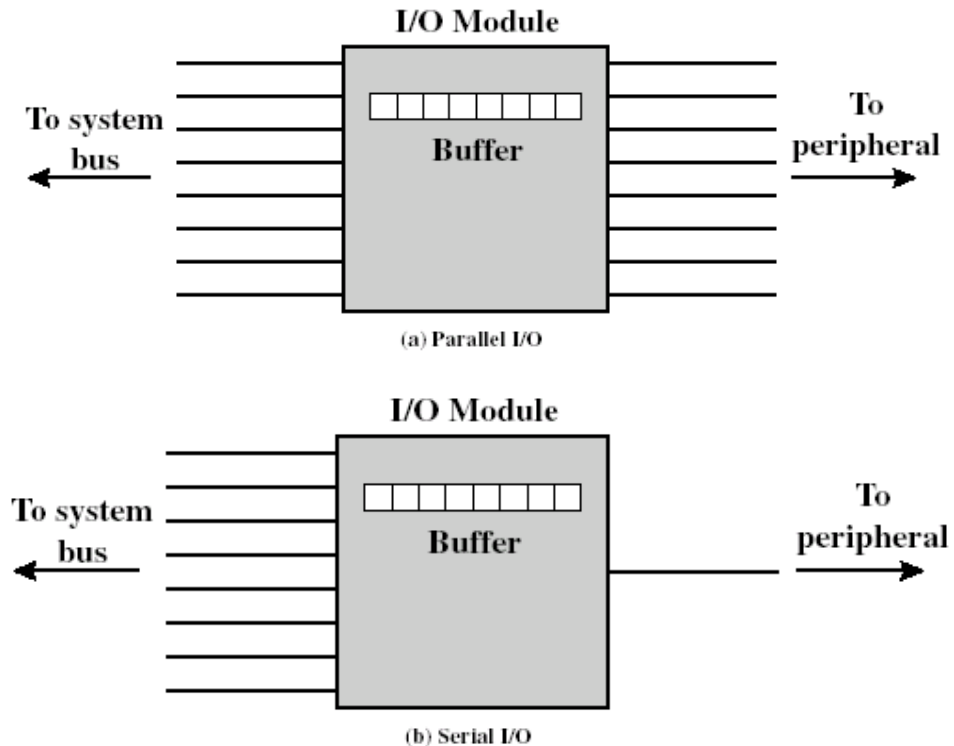
- E' la connessione tra modulo I/O e periferica
- Cambia in funzione del tipo di periferica
- Due tipi di interfacciamento

- Parallelo: linee multiple di connessione, tutti i bit di una parola vengono trasmessi simultaneamente

- > dischi, nastri, periferiche molto veloci

- Seriale: un'unica linea di connessione, i bit vengono trasmessi uno alla volta

- > stampanti, terminali



Bus USB - Universal Serial Bus

Ai calcolatori sono spesso collegate numerose periferiche a bassa/media velocità: tastiera, puntatore (mouse), lettori floppy e CD, casse acustiche, cuffia, microfono, macchine fotografiche, ecc.

In origine ognuna di queste periferiche aveva un proprio BUS di I/O indipendente

Il BUS USB è stato ideato allo scopo di fornire un BUS di I/O universale per le periferiche a bassa/media velocità; oggi si usa nei calcolatori di classe PC

Per saperne di più

- Vedere il sito www.t10.org, sito del Technical Committee of the National Committee on Information Technology Standards, dove si possano trovare molte informazioni su periferiche, interfacce, etc.
- Per sapere qualcosa di più sull'interfaccia USB (Universal Serial Bus): <http://www.usb.org/>