Il Livello Logico-Digitale

Il BUS del calcolatore

Lezione tenuta da:
Alessandro A. Nacci - alessandro.nacci@polimi.it

Capitolo 8 Hamacher

Il BUS del calcolatore

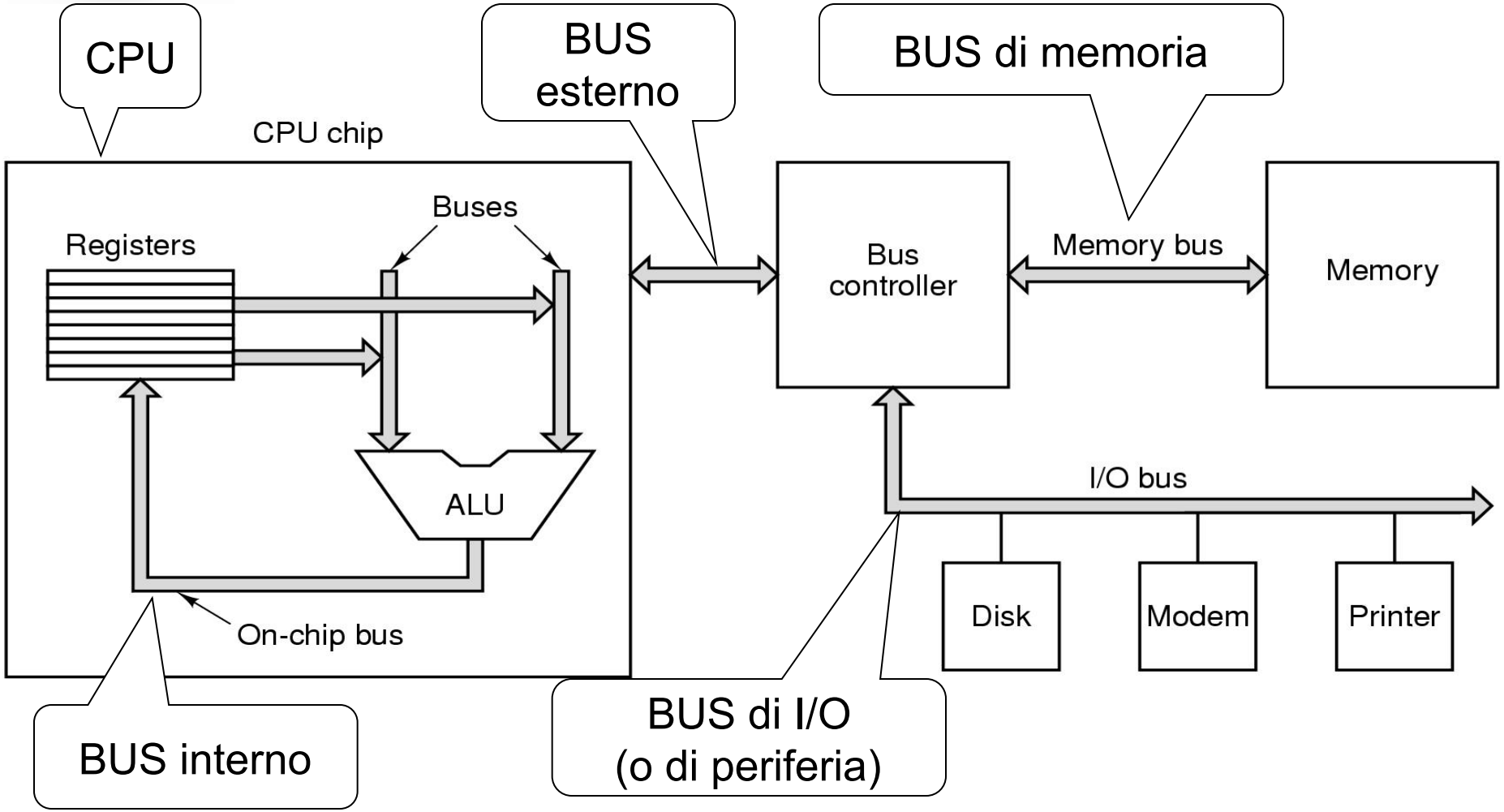
- Il calcolatore elettronico è un insieme di unità funzionali:
  - unità centrale di elaborazione (CPU), o processore
  - unità funzionali di memoria, o banchi di memoria
  - unità funzionali di interfacciamento alle periferiche
- Le unità sono interconnesse tramite un organo di collegamento: il BUS
- Lo scopo del BUS è quello di effettuare tutti i **trasferimenti di informazioni** tra le unità funzionali del calcolatore
  - Le informazioni da trasferire sono costituite da **istruzioni** e **dati**
  - Ogni trasferimento costituisce un'operazione sul bus

BUS interni ed esterni

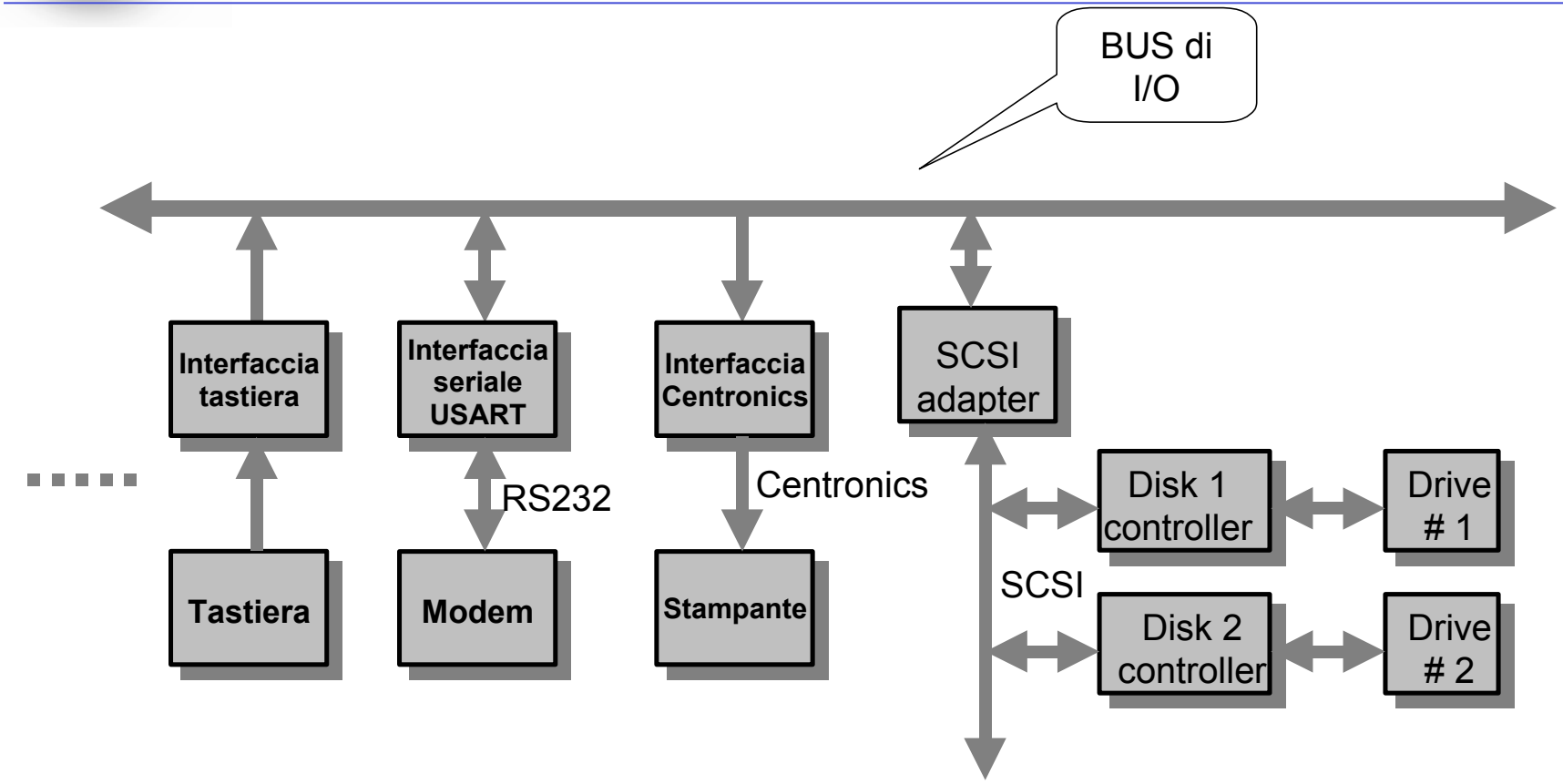
- I BUS del calcolatore si distinguono in:
  - BUS interni, confinati all'interno di una singola unità funzionale, e che collegano i blocchi funzionali contenuti nell'unità
  - BUS esterni, che si estendono all'esterno dell'unità funzionale, e che la collegano alle altre unità funzionali. Sono solitamente standardizzati

- I primi calcolatori erano dotati di un unico BUS esterno (BUS di sistema), collegante CPU, memoria e unità di I/O
- La maggior parte dei calcolatori odierni è dotata di due bus esterni:
  - **BUS di memoria**, che collega CPU e unità funzionali di memoria (banchi di memoria)
  - **BUS di I/O**, che collega CPU e unità funzionali di I/O

Sistema con diversi BUS



Dettaglio BUS di I/O



Controllo del BUS

- In ogni istante, una sola unità funzionale possiede il controllo del BUS, cioè decide quali operazioni di trasferimento 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 funzionali
- L'unità funzionale che detiene il controllo del BUS si chiama MASTER:
  - decide quale operazione eseguire, lettura oppure scrittura, e in quale istante di tempo
  - decide qual è l'unità funzionale da cui leggere oppure su cui scrivere
- Le rimanenti unità funzionali, che non detengono il controllo del BUS, si chiamano SLAVE

Ruoli e modalità di connessione al bus

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

- Per potersi collegare al BUS, le unità MASTER e SLAVE sono dotate di un componente (o blocco funzionale) di interfaccia, chiamato BUS DRIVER
- Il BUS driver è in grado di:
  - collegarsi e scollegarsi elettricamente al o dal BUS (p. es. tramite la tecnica di alta impedenza, o anche tramite altre tecniche)
  - amplificare opportunamente i segnali da trasmettere / ricevere sul / dal BUS

Problematiche del BUS

- Il progetto di BUS esterni per calcolatori comprende quattro aspetti principali:
  - struttura e dimensione, cioè numero di collegamenti presenti nel BUS
  - funzionamento base, operazioni e cicli di bus
  - gestione della temporizzazione del BUS
  - arbitraggio del controllo del BUS

- Per progettare o descrivere un BUS occorre considerare tutti gli aspetti

1 - Struttura del BUS

- Un generico BUS esterno di calcolatore ha una struttura simile alla piedinatura della CPU, cui va collegato
- È diviso in tre componenti:
  - BUS degli indirizzi (address BUS)
  - BUS dei dati (data BUS)
  - BUS di controllo (control BUS)
- La corrispondenza con i piedini della CPU non è però sempre uno-a-uno
- Versioni del bus: molti tipi di BUS sono stati progressivamente potenziati
  - aumento della larghezza del BUS indirizzi
  - aumento della larghezza del BUS dati
  - e anche qualche aggiunta di segnali al BUS di controllo

Dimensione del BUS

- Dimensione (o larghezza) del BUS indirizzi: con m ≥ 1 linee si ottengono 2^m indirizzi, numerati da 0 a 2^m - 1
- Dimensione (o larghezza) del BUS dati: con n ≥ 1 linee si possono scambiare (leggere o scrivere) parole da n bit
- Per installare nel calcolatore grandi quantità di memoria occorrono BUS indirizzi (e anche dati) molto larghi
- Si possono riunire le funzioni del BUS di indirizzo e del BUS dati in un unico gruppo di linee: il BUS multiplato
- Quando il processore deve leggere:
  - nel primo ciclo invia l'indirizzo sul BUS multiplato
  - nel secondo ciclo riceve il dato, sempre sul BUS multiplato
  - La scrittura funziona in modo simile
- Meno collegamenti, ma velocità inferiore

2 - Funzionamento del BUS

- Le attività del calcolatore si sviluppano per **cicli di BUS**: in ogni ciclo avviene un'operazione
- Le operazioni sono governate dal MASTER che detiene il controllo del BUS. Gli SLAVE non possono dare inizio a un'operazione in modo autonomo
- Le operazioni sono trasferimenti di informazioni tra MASTER e SLAVE e può essere necessario specificare se il trasferimento è relativo a memoria o a interfaccia di I/O
  - Operazione di lettura: un dato viene trasferito da uno SLAVE al MASTER. Lo SLAVE è la sorgente dell'informazione
  - Operazione di scrittura: un dato viene trasferito dal MASTER a uno SLAVE. Il MASTER è la sorgente dell'informazione
  - La direzione del trasferimento (lettura o scrittura) è relativa al MASTER che detiene (in quel momento) il controllo del BUS

Unità funzionali e collegamenti al BUS

- Poiché è il MASTER a dare inizio a un'operazione di lettura o scrittura, solo il MASTER può immettere un indirizzo sul BUS degli indirizzi
  - Gli SLAVE non possono generare indirizzi e corrispondono ai diversi indirizzi, secondo la mappa di indirizzamento del calcolatore
- I collegamenti delle unità funzionali al BUS dati variano, a seconda che l'unità funzionale (MASTER o SLAVE) sia in lettura e scrittura, sola lettura o sola scrittura
- I collegamenti delle unità funzionali al BUS di controllo sono quelli meno regolari e classificabili e dipendono dalle funzioni dell'unità considerata

Cicli di BUS

- In generale, ogni operazione sul bus corrisponde a un ciclo di BUS. I cicli di bus sono classificabili come segue:
  - lettura di una parola di memoria
  - scrittura di una parola di memoria
  - lettura di un registro di I/O
  - scrittura di un registro di I/O
  - riposo: il BUS non viene usato

- Una singola operazione di lettura o scrittura può anche svilupparsi su più cicli di BUS
  - L'uso di più cicli di BUS può rendersi necessario quando un MASTER veloce deve trasferire dati con uno SLAVE lento

Esempio di divisione in cicli di BUS

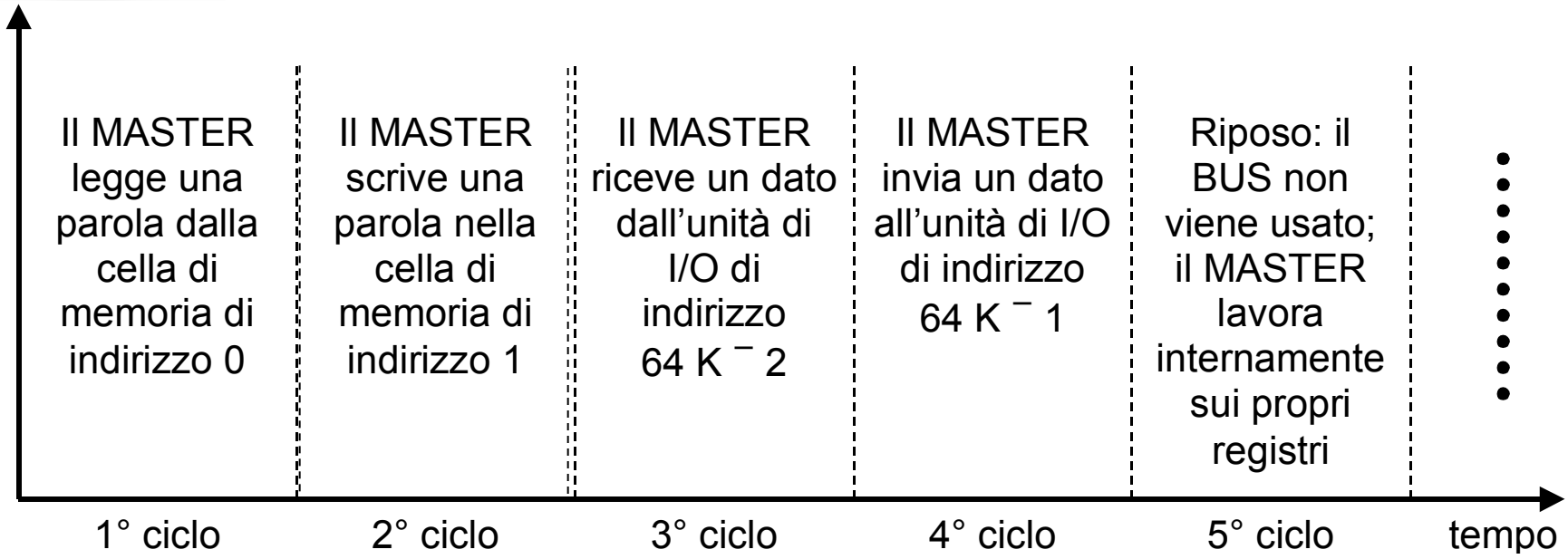


Diagramma temporale (schematico)
dei cicli di BUS

Fasi di un ciclo di BUS

- Un ciclo di trasferimento si può generalmente scomporre nelle seguenti fasi
  - Selezione: in questa fase il master seleziona l'elemento slave coinvolto dal trasferimento, precisando l'indirizzo, il tipo di elemento (memoria o I/O) e la direzione (lettura o scrittura)
  - Eventuale attesa: questa fase viene eseguita solo se l'elemento slave coinvolto è relativamente lento e quindi richiede, per il corretto trasferimento, che venga concesso un tempo di accesso maggiore rispetto ai normali cicli di bus (tramite l'inserimento di "stati" di wait) o l'attivazione di un segnale di pronto
  - Trasferimento dei dati: in questa fase, l'unità destinazione del trasferimento "cattura" (cioè memorizza localmente) l'informazione emessa dalla sorgente
  - Conclusione: in questa fase tutti i segnali sono ordinatamente riportati nello stato di riposo

3 - Temporizzazione del BUS

- Per realizzare la scansione dei **cicli di BUS** esistono due metodi fondamentali:
  - BUS sincrono: funzionamento governato da un segnale di clock
  - BUS asincrono: funzionamento governato solo dall'interazione tra master e slave

Convenzioni sui termini e sui segnali

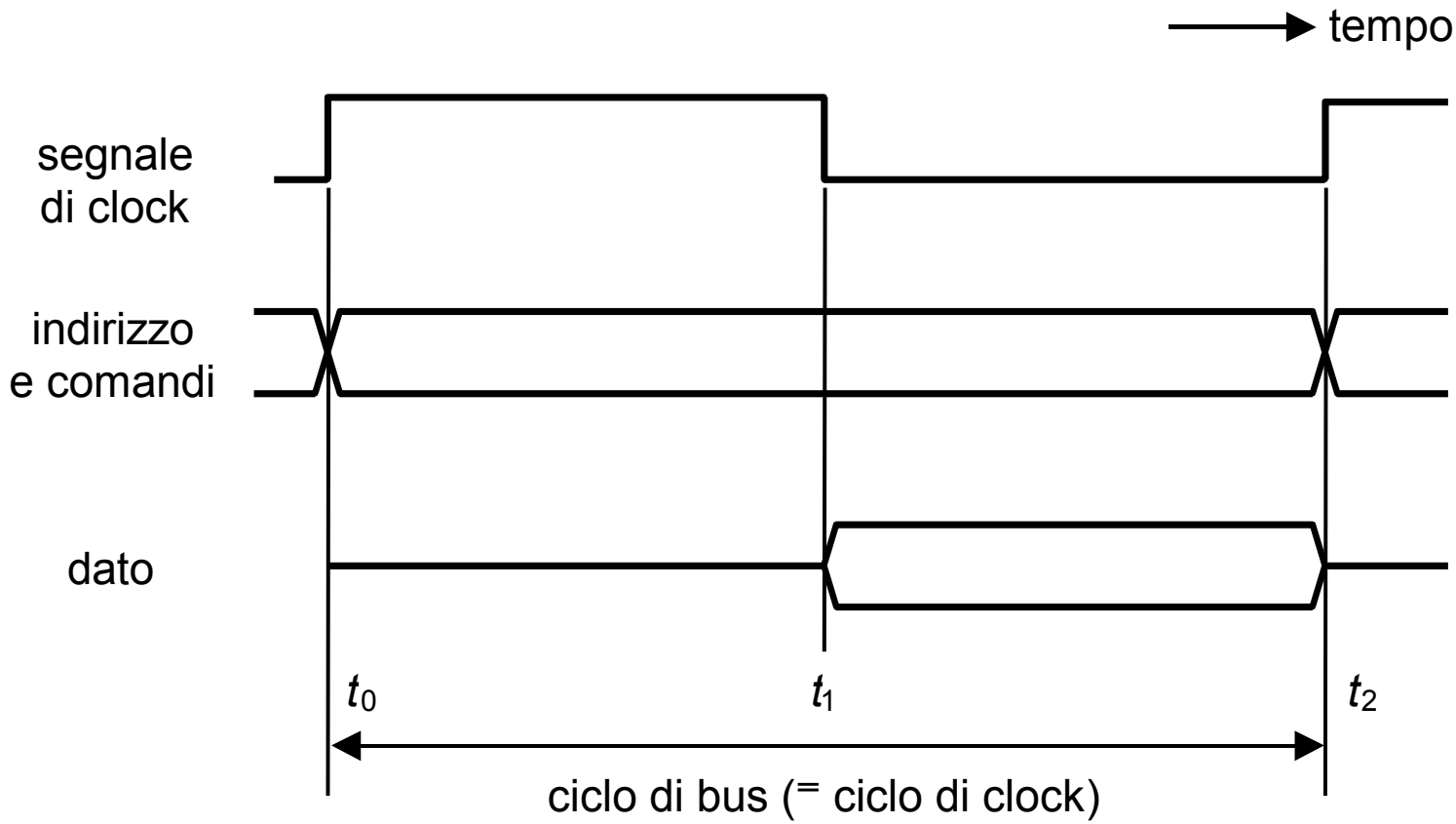
- Mostreremo sequenze degli eventi di meccanismi di I/O, con alcune precisazioni in più rispetto all'Hamacher, orientate allo svolgimento dei temi d'esame;
- Per convenzione il ciclo di clock inizia col fronte di **salita;**
- Convenzione di denominazione dei segnali (vedi Hamacher):
  - DATA = dato
  - ADDRESS = indirizzo
  - REQ = comando
  - SRDY = unità slave pronta

  Indipendentemente dal fatto che un segnale sia attivo alto/basso, usiamo i termini **set / reset per indicare attivazione/disattivazione**

BUS sincrono

- Il BUS di controllo contiene una linea per il segnale di clock, a frequenza prestabilita
- Il clock viene distribuito a tutte le unità funzionali collegate al BUS sulle cui transizioni avvengono gli eventi
- Il segnale di clock scandisce le varie transizioni di segnale e il passaggio da un ciclo di BUS al ciclo successivo
- Tutte le unità sanno sempre quando si passa al ciclo successivo
- La durata del ciclo di bus dipende dai dispositivi e dai tempi di latenza/propagazione

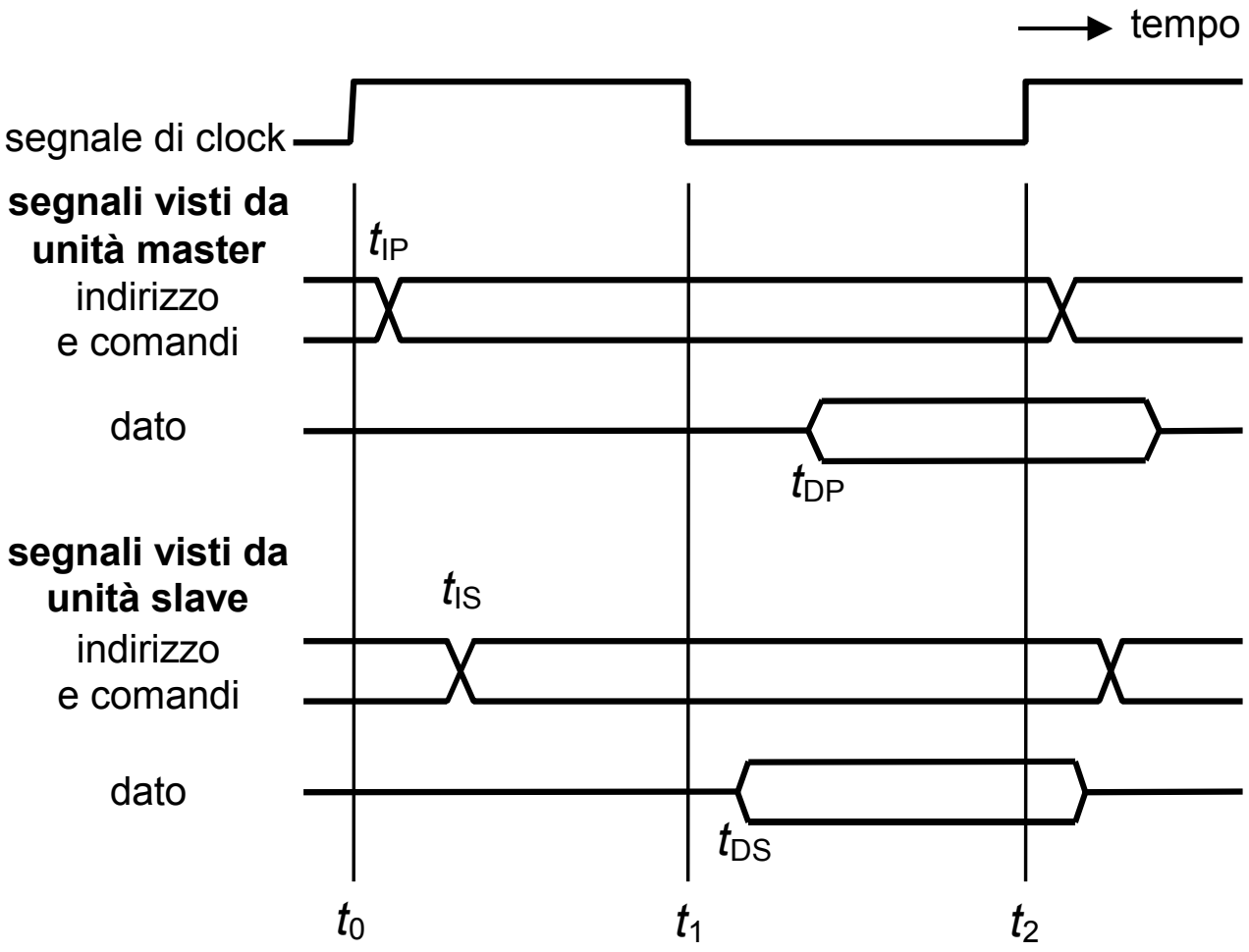
Bus sincrono: funzionamento base per la lettura



Durata del ciclo di clock del bus

- t1 – t0 non deve essere inferiore al massimo ritardo di propagazione tra due unità collegate al bus incluso il tempo necessario per decodificare l'indirizzo e interpretare la richiesta sul bus di controllo in modo che lo slave sia in grado di rispondere a partire da t1
- t2 – t1 non deve essere inferiore alla somma del massimo tempo di propagazione dei dati sul bus dati e del tempo di setup del registro
- t2 il dato sul bus viene campionato dalla CPU e memorizzato in un registro

Ciclo di lettura del bus sincrono: dettagli



Effetto dei tempi di propagazione nella lettura

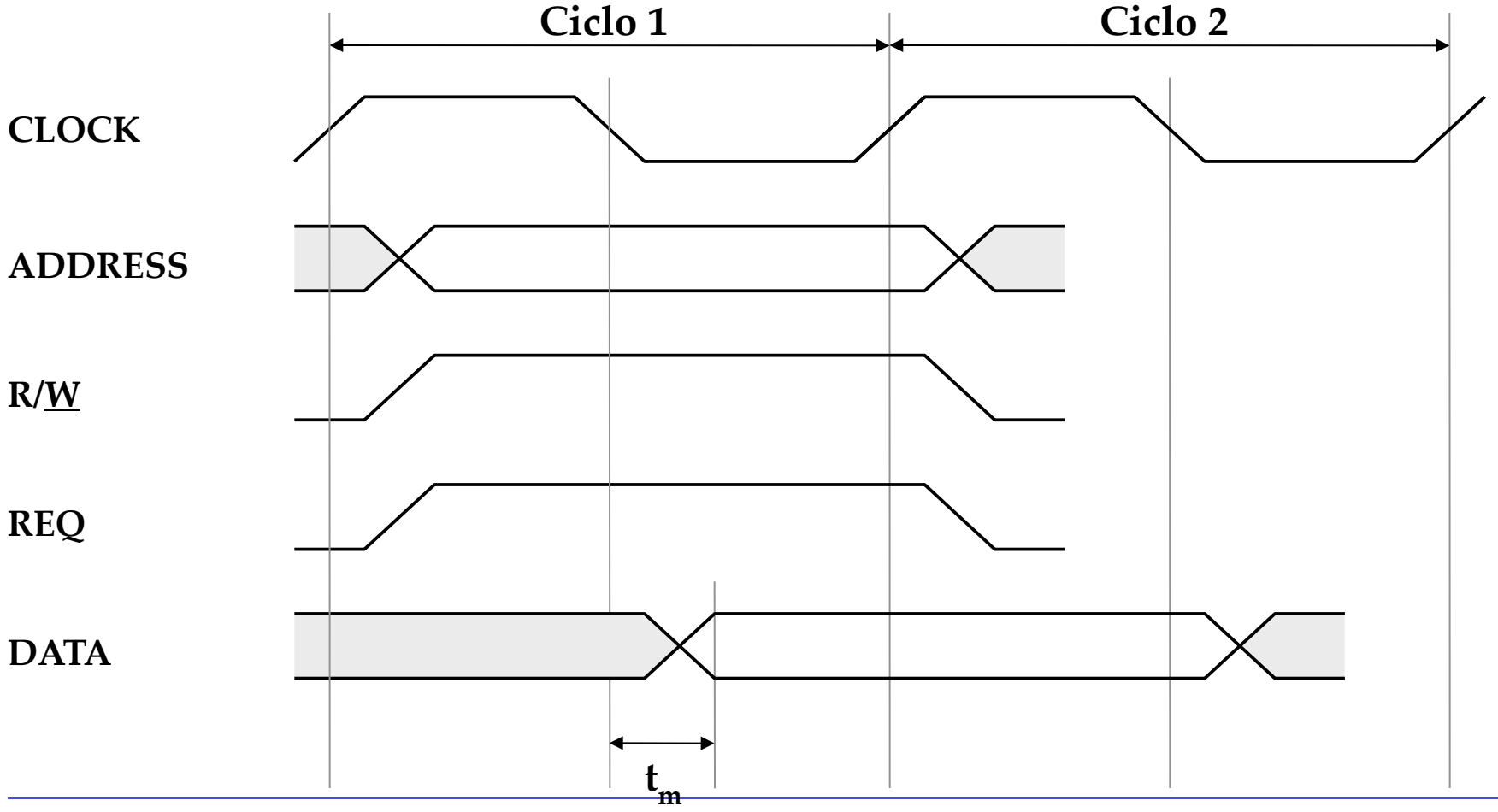
- t0: master emette
  - indirizzo a cui leggere sul bus indirizzo
  - segnale di lettura
- tIP: segnali sul bus (ritardo)
- tIS: ritardo con cui l'indirizzo raggiunge lo slave
- t1: unità slave decodifica l'indirizzo ed emette il dato richiesto
- tDS: dato sul bus
- tDP: ritardo con cui il dato arriva al master
- t2: unità master legge il dato e lo scrive in un registro interno
- t2 – tDP >= tempo di setup del registro
- Dato deve rimanere stabile sul bus per un intervallo di tempo dopo t2 > del tempo di hold del registro

Bus sincrono monociclo, lettura

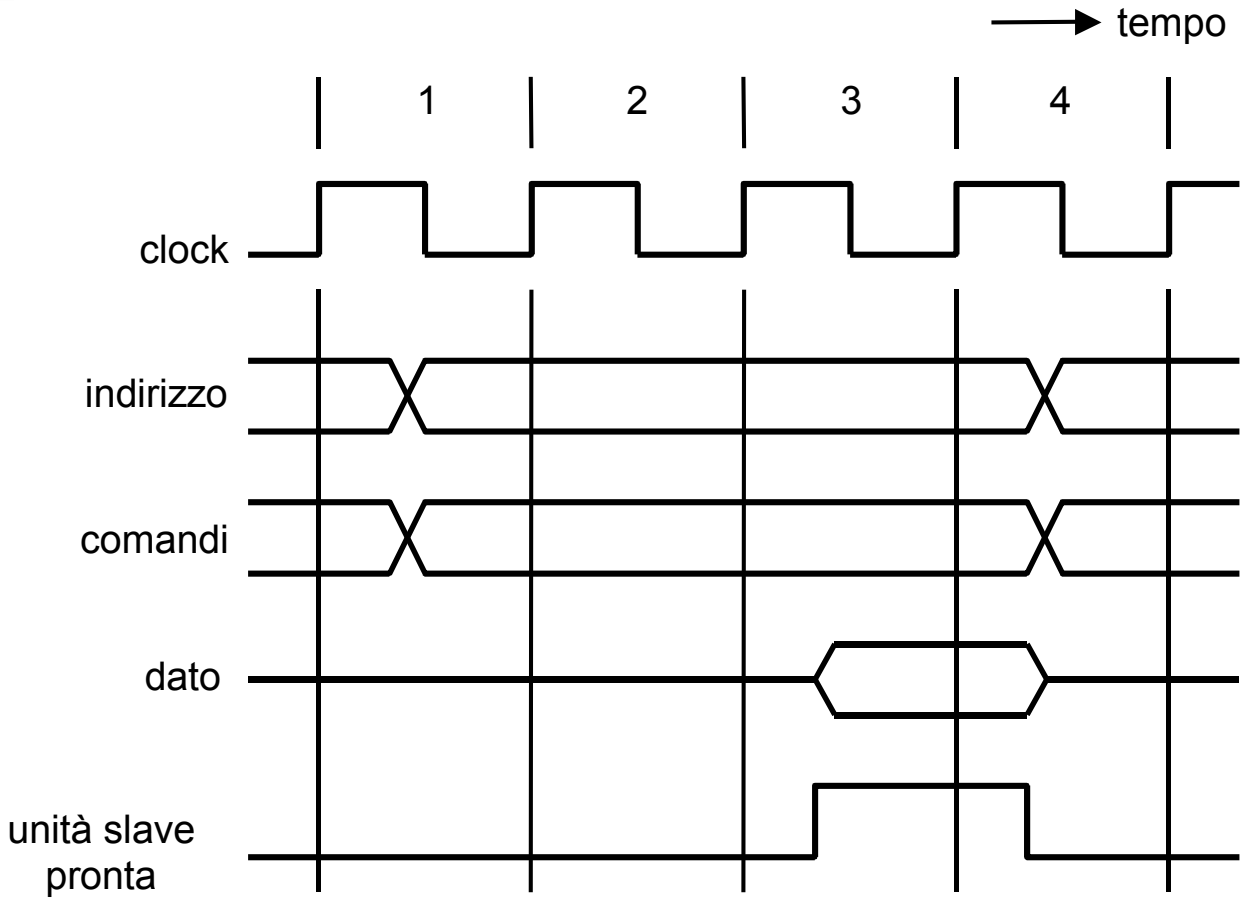
- Sequenza degli eventi:
  - @ *clock* ↗ *master* *set* *ADDRESS, R/W*
    master *set* *REQ*
  - @ *clock* ↘ *slave* *sample* *ADDRESS, R/W*
    ...
    slave *set* *DATA*
  - @ *clock* ↗ *master* *sample* *DATA*
    master *unset* *REQ, ADDRESS*
    (master può iniziare lettura successiva)
  - @ *clock* ↘ *slave* *unset* *DATA*
  - L'operazione successiva può essere una lettura
    (in tal caso si completa una lettura in ciascun ciclo);
    ma non può essere una scrittura: master deve attendere unset DATA;

Esempio: lettura monociclo

Clock = 10 MHz (T=100 ns); latenza della memoria t_m = 20ns; lettura in un ciclo;



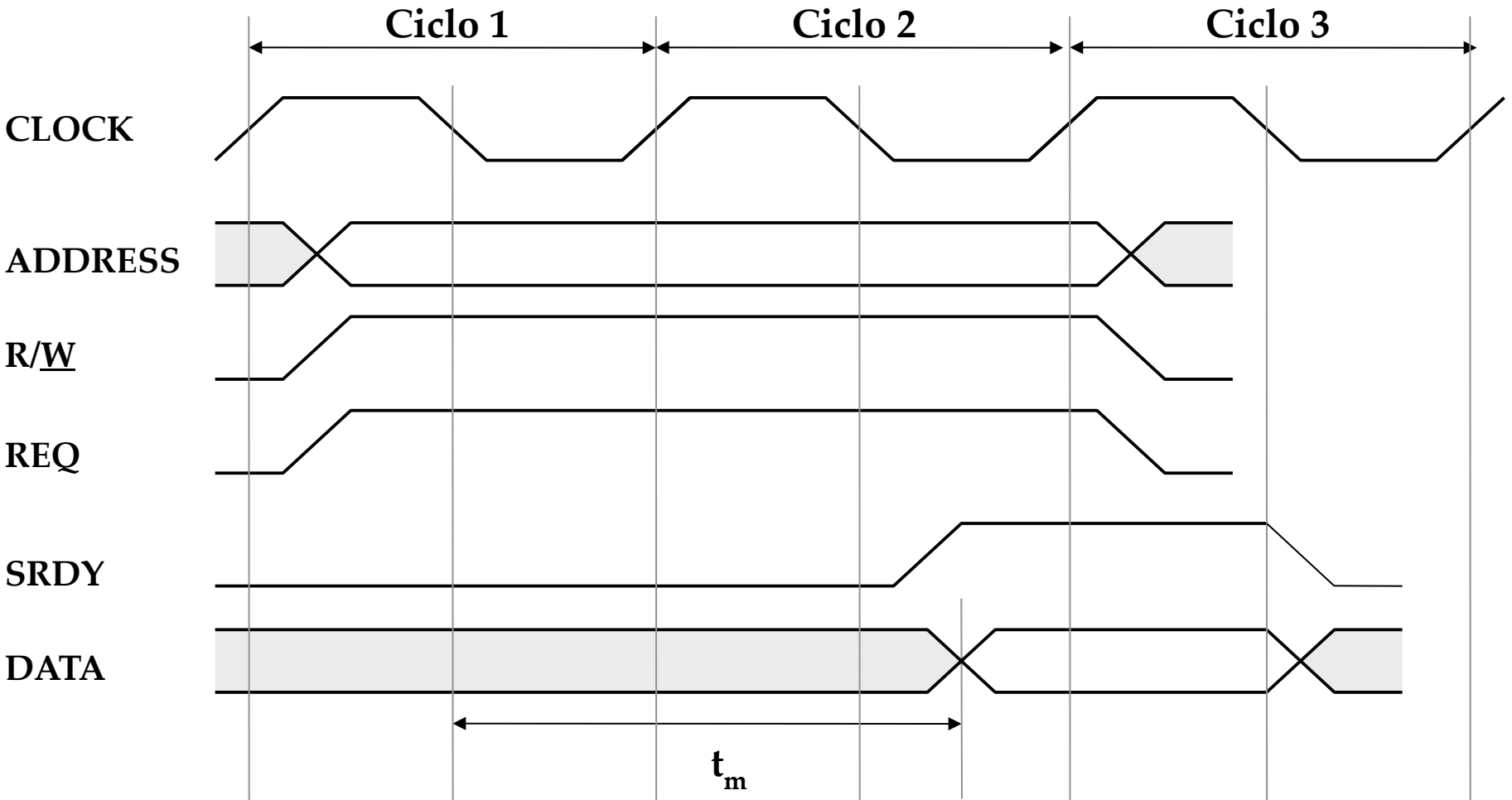
Trasferimento multiciclo



Bus sincrono multiciclo, lettura

- *Sequenza degli eventi:*
  - *@ clock ↗* *master* *set* *ADDRESS*
    master *set* *R/W*
    master *set* *REQ*
  - *@ clock ↘* *slave* *sample* *ADDRESS, R/W*
  - *ritardo di 0 o più cicli di clock; a lettura completata:*
    slave *set* *SRDY*
    slave *set* *DATA*
  - *@ clock ↗* *master* *sample* *DATA*
    master *unset* *REQ, ADDRESS*
    (master può iniziare lettura successiva)
  - *@ clock ↘* *slave* *unset* *DATA, SRDY*
- *Il master campiona i dati al termine del ciclo in cui vede SRDY attivato;*

Esempio: lettura multiciclo



Clock = 10 MHz (T=100 ns); latenza della memoria t_m = 120ns; lettura in 2 cicli;

BUS asincrono

- Non esiste alcun segnale di clock comune alle unità funzionali collegate al BUS del calcolatore
- Le transizioni di segnale e il passaggio da un ciclo di BUS al ciclo successivo non sono sincronizzati

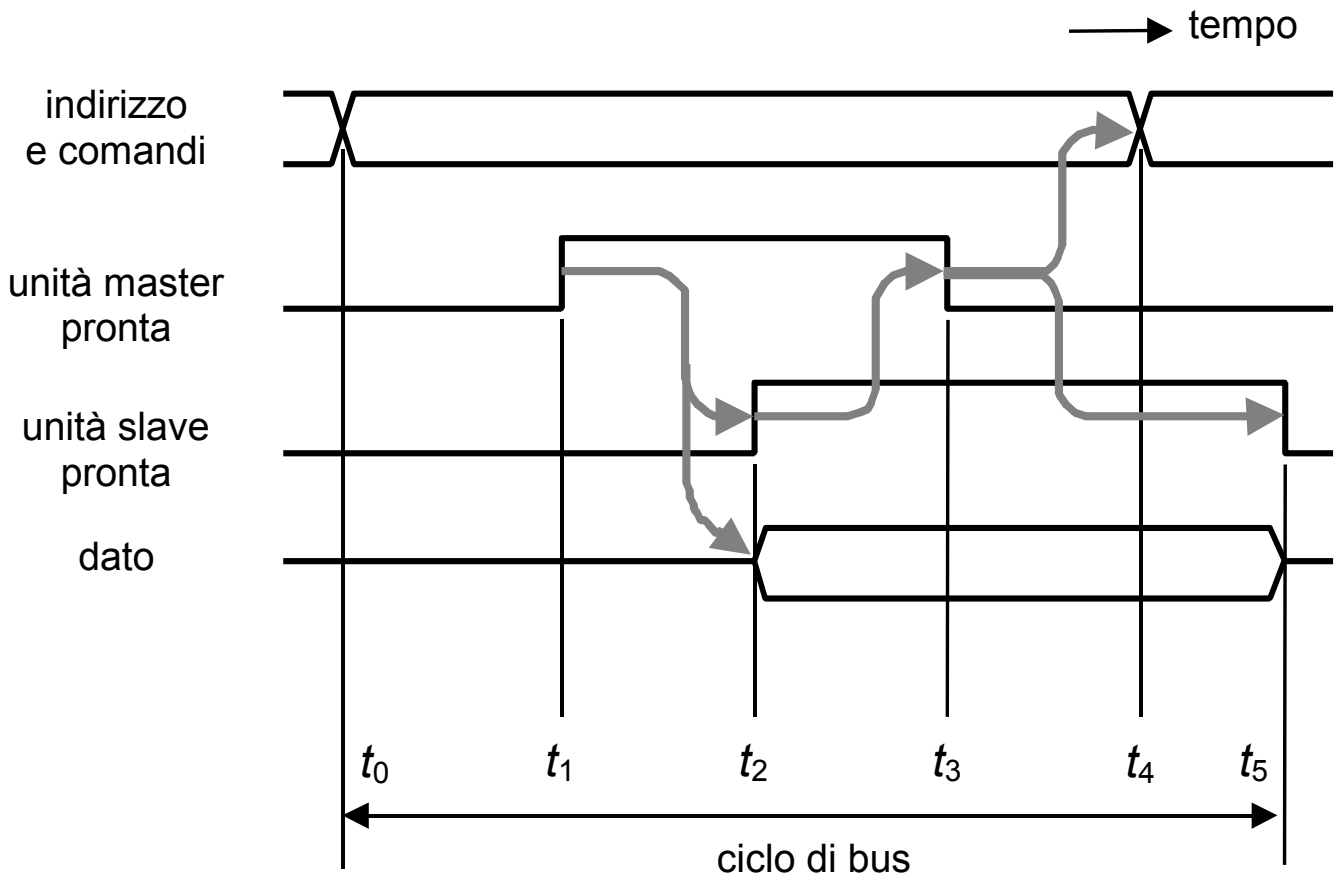
- Le unità funzionali osservano il BUS di controllo: quando avviene una transizione di segnale significa che si verifica un avanzamento dell'operazione
  - Il bus è dotato di opportuni segnali di controllo (di cui vengono osservate le transizioni) che realizzano un protocollo di sincronizzazione a segnale (ad es. full-handshake)

Segnali di controllo per l'handshake

- MSYN (Master Synchronisation): unità master pronta
  - il MASTER segnala di avere impostato indirizzo (e MREQ) e RD;
- SSYN (Slave Synchronisation): unità slave pronta
  - lo SLAVE segnala di avere completato l'operazione

  - Poiché non esiste un segnale di clock che marchi gli istanti di tempo in cui i vari segnali di controllo si possono attivare o disattivare, nel diagramma temporale vanno indicati i rapporti causa-effetto esistenti tra i vari segnali di controllo
  - I segnali di controllo transitano di valore in reazione a transizioni precedenti

Operazione di lettura sul bus asincrono



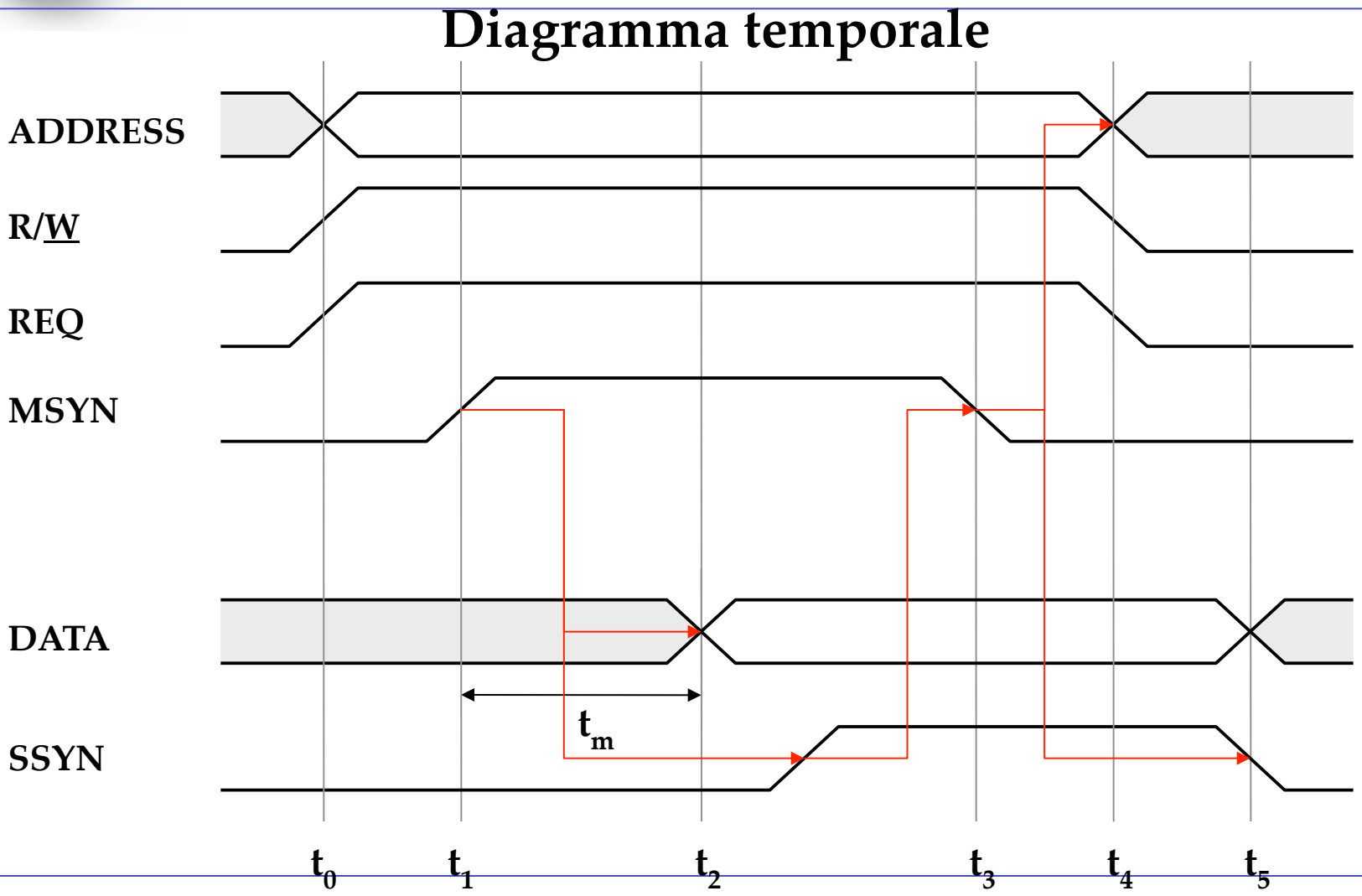
Fasi dell'operazione di lettura

- t0: Il MASTER (CPU) manda l'indirizzo sul BUS indirizzi, attiva MREQ e RD,
- t1: dopo averli tutti stabilizzati, il master attiva MSYN
- Il tempo t1-t0 compensa lo sfasamento temporale dovuto al fatto che i segnali sul bus si propagano con ritardi variabili in funzione della distanza
- t2: Lo SLAVE (memoria) esegue l'operazione, nel minor tempo possibile, e fornisce la parola sul BUS dati, poi attiva SSYN
- t2-t1 varia in base alle unità coinvolte e deve tenere conto del ritardo con cui le unità slave emettono il dato sul bus e il ritardo di propagazione sul bus
- t3: segnale SSYN arriva al master e rende noto che il dato è presente sul bus
  - Unità master prima di leggere il dato dal bus DATI deve attendere eventuali ritardi di propagazione del dato + tempo di setup del registro
- t3: Il MASTER disattiva MSYN
- t4: Il **MASTER** toglie l'indirizzo e comandi dal bus
- t5: lo **SLAVE** disattiva **SSYN**

Bus asincrono, lettura

- *Sequenza degli eventi:*
  - *t0:* *master set* *ADDRESS, REQ, R/W*
  - *t1:* *master set* *MSYN (dopo t)*
  - *t2:* *slave set* *DATA (ASAP)*
    slave set *SSYN (dopo.)*
  - *t3:* *master vede* *SSYN*
    master sample *DATA*
  - *t4:* *master reset* *MSYN, REQ, R/W*
  - *t5:* *slave reset* *SSYN*
    slave unset *DATA*

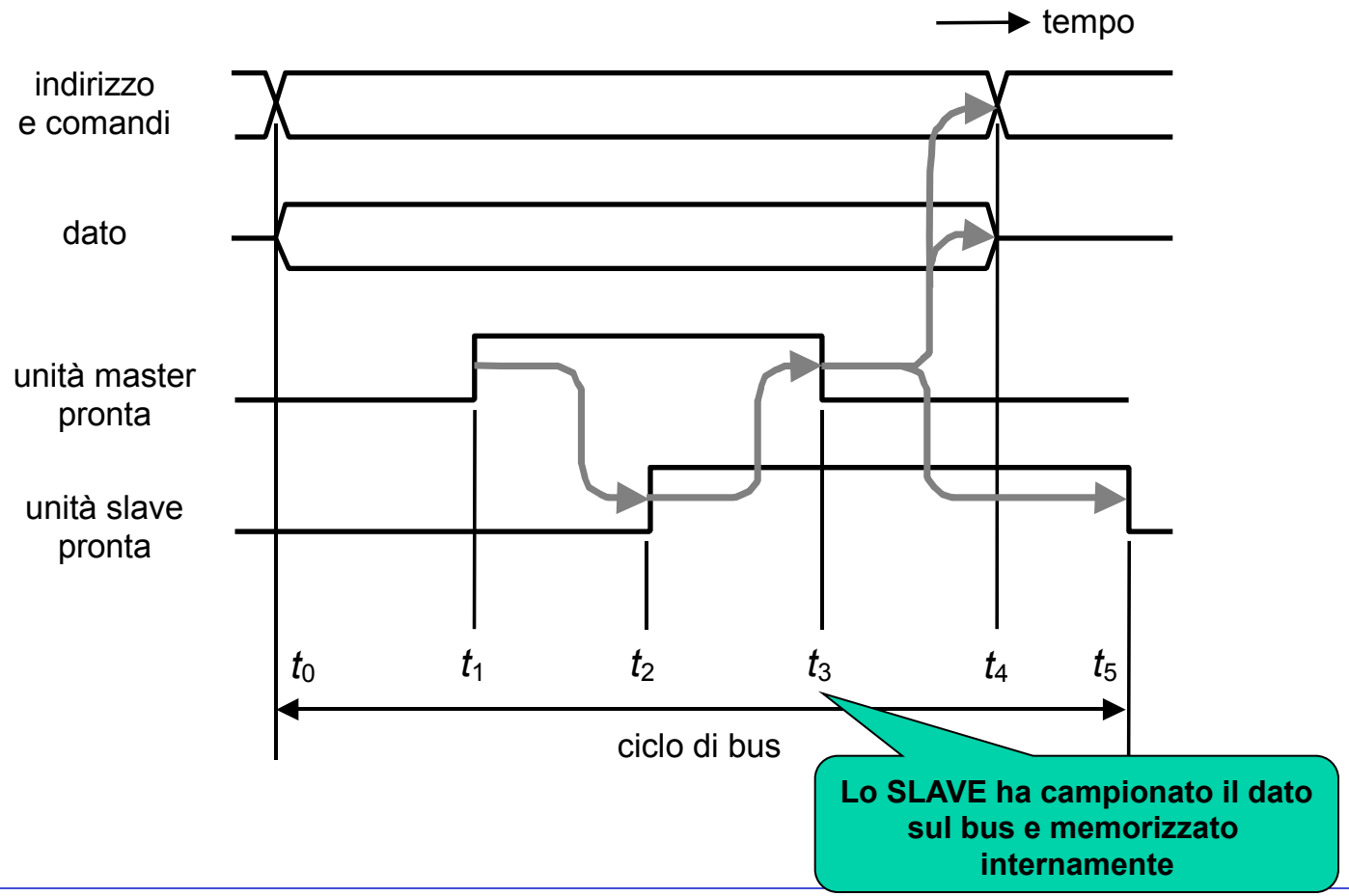
Bus asincrono, lettura



Procedura full-handshake

- Lo schema asincrono appena visto è incentrato sulla seguente procedura:
  - MSYN viene attivato
  - SSYN viene attivato in risposta a MSYN
  - MSYN viene disattivato in risposta a SSYN
  - SSYN viene disattivato in risposta a MSYN
- La procedura ha nome full-handshake: è indipendente dal clock ed è insensibile alla presenza di SLAVE lenti
  - Fully interlocked: ogniqualvolta uno dei due segnali di sincronismo varia, anche l'altro varia in risposta

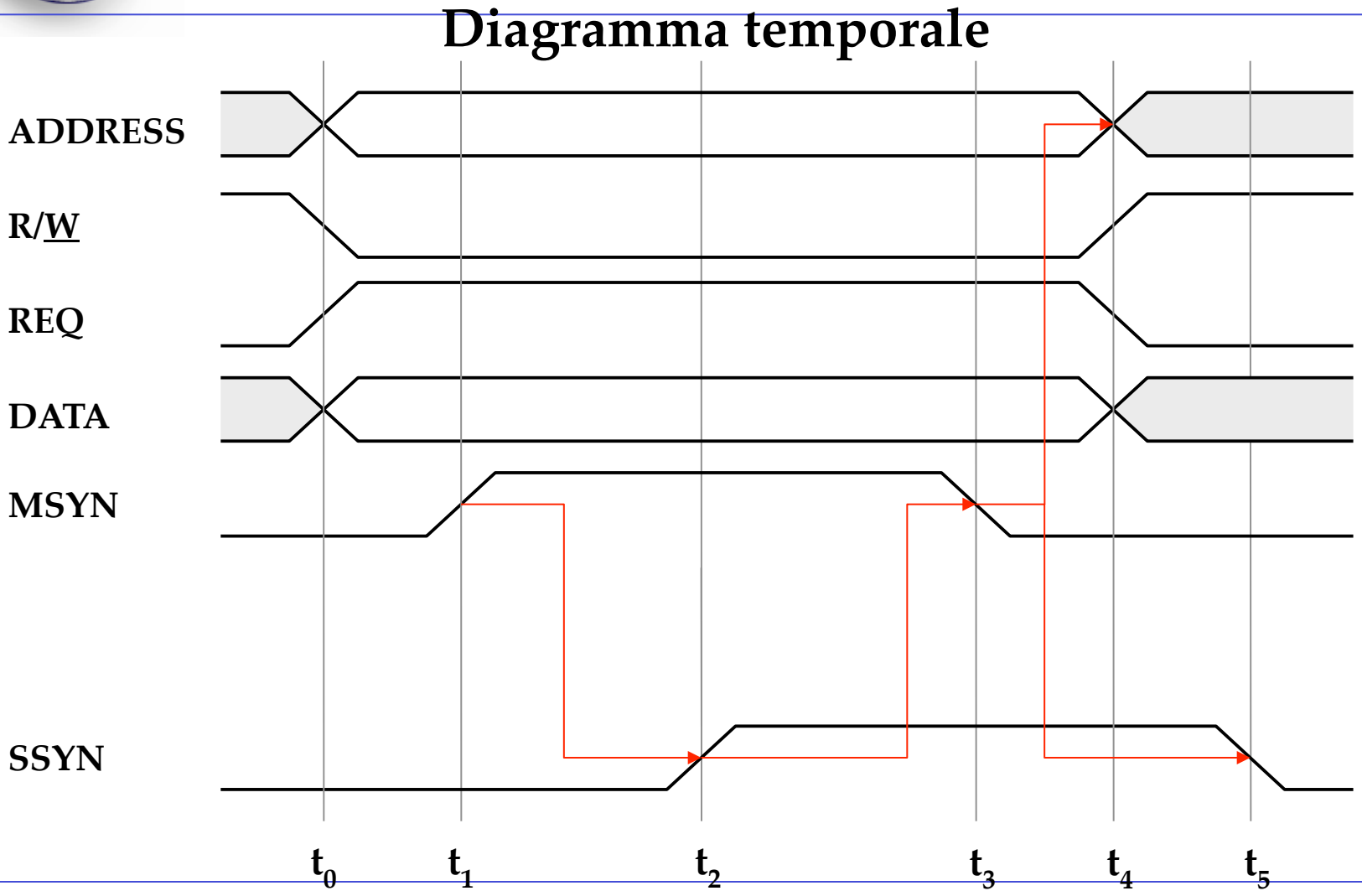
Operazione di scrittura sul bus asincrono



Bus asincrono, scrittura

- *Sequenza degli eventi:*
  - *t0:* *master* *set* *ADDRESS, REQ, DATA*
    master *reset* *R/W (scrittura)*
  - *t1:* *master* *set* *MSYN (dopo t prop.)*
  - *slave* *sample* *DATA*
    slave *(scrittura)*
  - *t2:* *slave* *set* *SSYN*
  - *t3:* *master* *vede* *SSYN*
  - *t4:* *master* *unset* *MSYN, REQ, R/W, DATA*
  - *t5:* *slave* *reset* *SSYN*

Bus asincrono, scrittura



Confronti

Il BUS sincrono

- vantaggi: semplicità di controllo e maggiori frequenze di trasferimento
- svantaggio: "sprechi" di tempo (ogni operazione si deve svolgere in un numero intero di cicli di clock):
  - quando un'operazione si potrebbe completare in meno di un ciclo di clock
  - quando un'operazione richiede un numero frazionario di cicli di clock

  Necessità di progettare il circuito di distribuzione del clock in modo accurato

Il BUS asincrono

- vantaggi:
  - maggior efficienza nell'uso dei cicli (l'operazione si completa esattamente nel tempo necessario)
  - Semplicità nella realizzazione delle interfacce
- svantaggi: frequenza di trasferimento limitata dalla necessità per ogni dato trasferito di uno scambio di due segnali di controllo tra master e slave

I BUS di calcolatore sono in massima parte di tipo sincrono

4- Arbitraggio del bus: cambio di MASTER

- 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, per scopi particolari:
  - Unità di I/O: possono diventare MASTER per trasferire dati direttamente con la memoria, senza bisogno della CPU (DMA)
  - Coprocessore: può diventare MASTER per prelevare operandi dalla memoria
- Il BUS del calcolatore può avere, in ogni istante, un unico MASTER
- In caso di cessione del ruolo di MASTER da un'unità funzionale a un'altra, occorre dunque 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

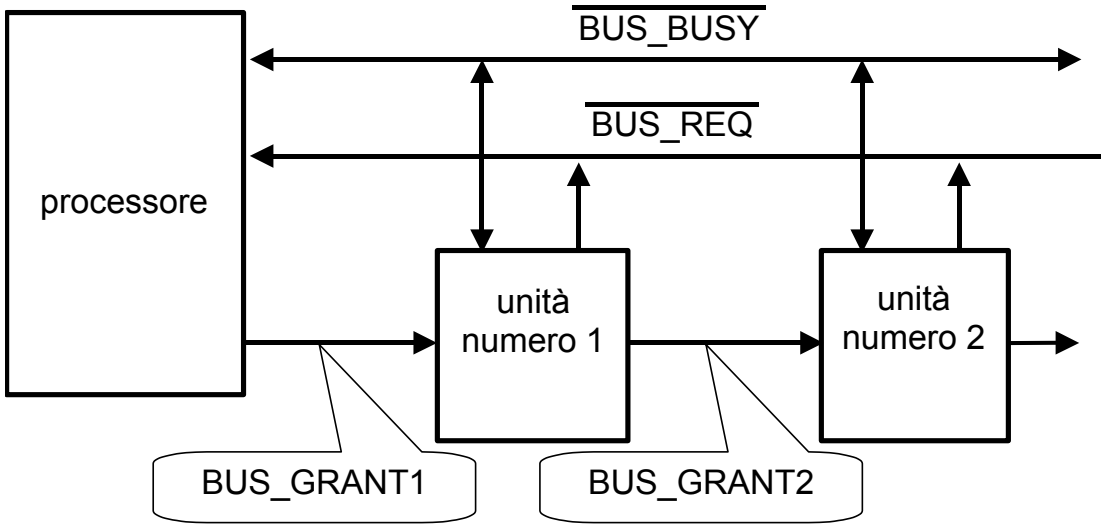
Arbitraggio centralizzato

Il meccanismo di arbitraggio centralizzato prevede:

- un'apposita unità funzionale, che svolge la funzione di arbitro del BUS (il processore o una unità indipendente collegata al bus)
- alcune linee (appartenenti al BUS di controllo) che collegano l'arbitro alle unità funzionali potenziali richiedenti il controllo del BUS:
  - **Bus Request** (richiesta di cessione del controllo)
  - **Bus Grant** (conferma di cessione del controllo)
  - **Bus Busy** (stato del bus: esistenza di un master che detiene il bus)

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

Arbitraggio centralizzato con collegamento a festone semplice

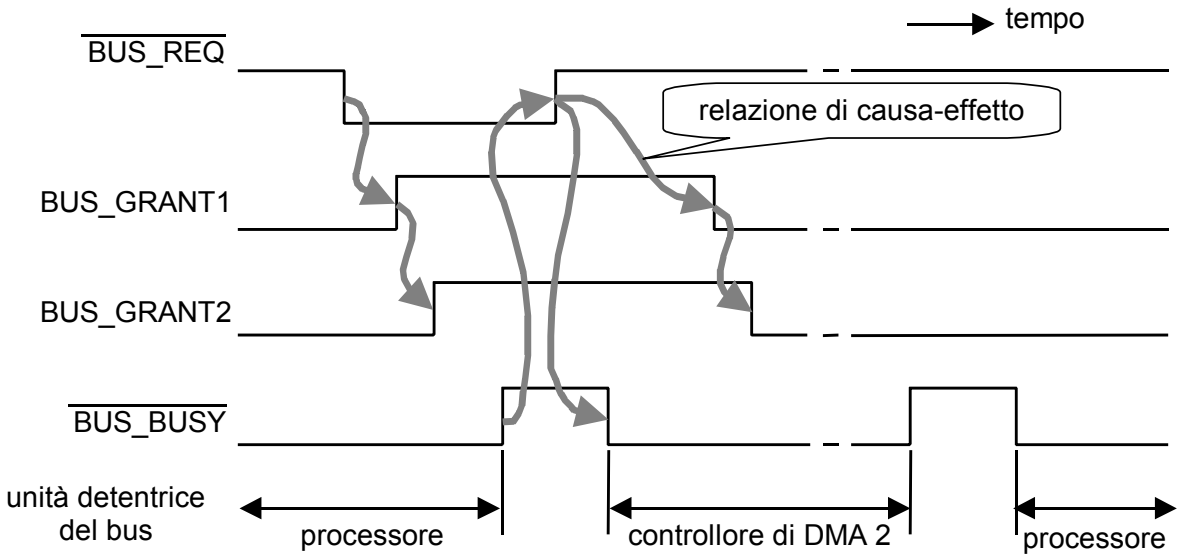


- Quando l'unità di arbitraggio riceve una richiesta di BUS (BUS_REQ), attiva la linea di conferma (BUS_GRANT)
- La conferma viene passata in cascata alle unità potenziali richiedenti:
  - se un'unità non ha una richiesta pendente, passa la conferma all'unità successiva
  - altrimenti, trattiene per sé la conferma, senza passarla all'unità successiva, e prende il controllo del BUS, comportandosi come MASTER
- L'unità che in un dato istante detiene il controllo del bus lo comunica attivando BUS_BUSY

Arbitraggio del bus centralizzato

- *Segnali:*
  - *Bus Request* BUS_REQ *richiesta del bus*
  - *Bus Granted* BUS_GRANTx *autorizzazione all'uso del bus*
  - *Bus BuSY* BUS_BUSY *bus occupato*
- *Sequenza degli eventi:*
  - *PERIF:* *set* BUS_REQ
  - *ARB:* *set* BUS_GRANT1
  - *PERIF:* *attende reset* BUS_BUSY
    reset BUS_REQ
  - *ARB:* *reset* BUS_GRANT1
  - *PERIF:* *set* BUS_BUSY
    usa *il bus*
    reset BUS_BUSY
- *Attenzione: numerose varianti possibili*

Arbitraggio centralizzato

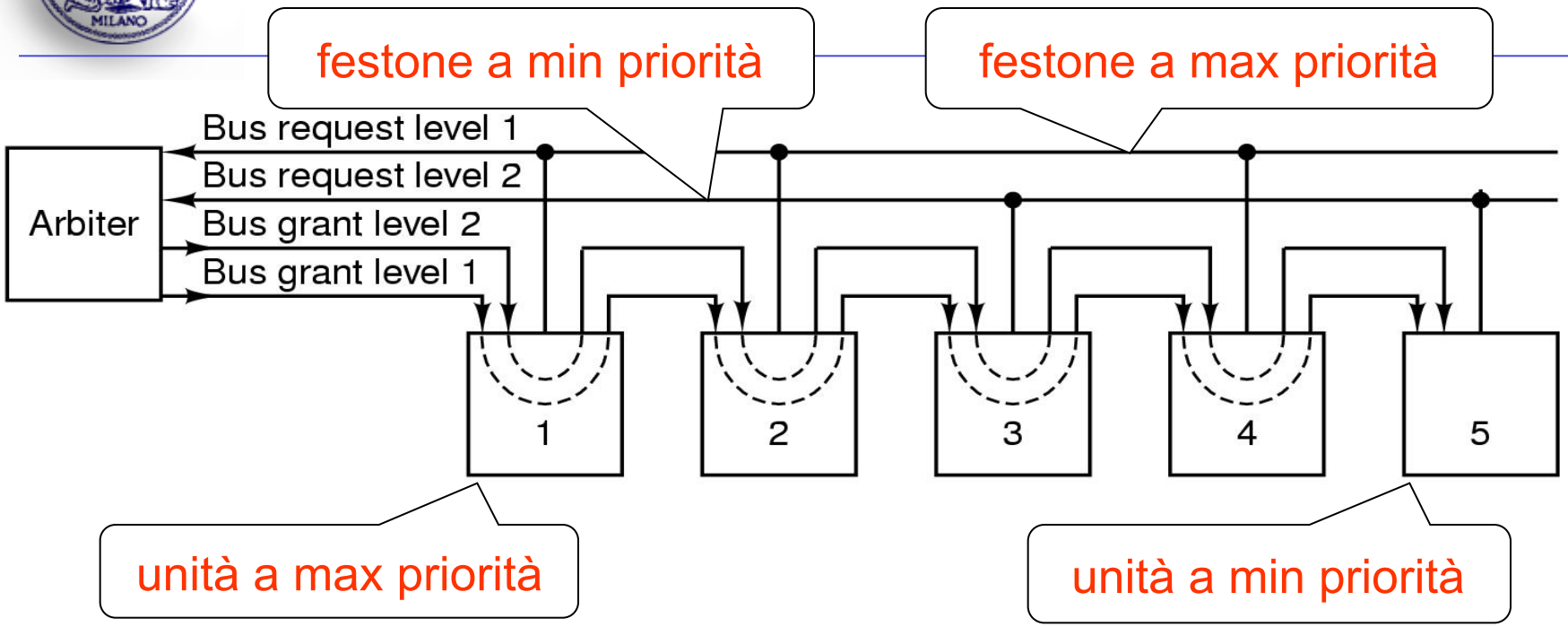


- L'unità 2 richiede il bus
- Segnali di BUS_REQ e BUS_BUSY attivi bassi
- BUS_BUSY tenuto a 0 dal master per tutto l'intervallo di tempo in cui detiene il bus
- BUS_REQ può essere attivato anche se BUS_BUSY attivo e viene disattivato dall'unità richiedente non appena ottiene il bus

Richiesta con priorità

- Il meccanismo di arbitraggio centralizzato viene chiamato COLLEGAMENTO A FESTONE (Daisy Chaining)
- È un modo cablato (hardware) per assegnare una priorità alle unità
  - L'unità a contatto con l'arbitro è quella a priorità max, mentre quella all'estremo del festone è quella a priorità min e ogni unità è a un livello di priorità diverso
- Per introdurre una certa flessibilità nel meccanismo di arbitraggio, si può:
  - dotare il BUS di più festoni di richiesta / conferma
  - stabilire una scala di priorità tra i festoni
  - e collegare ogni unità a uno solo dei festoni
- L'unità usa il festone cui è collegata, e dunque opera al livello di priorità relativo

Arbitraggio flessibile

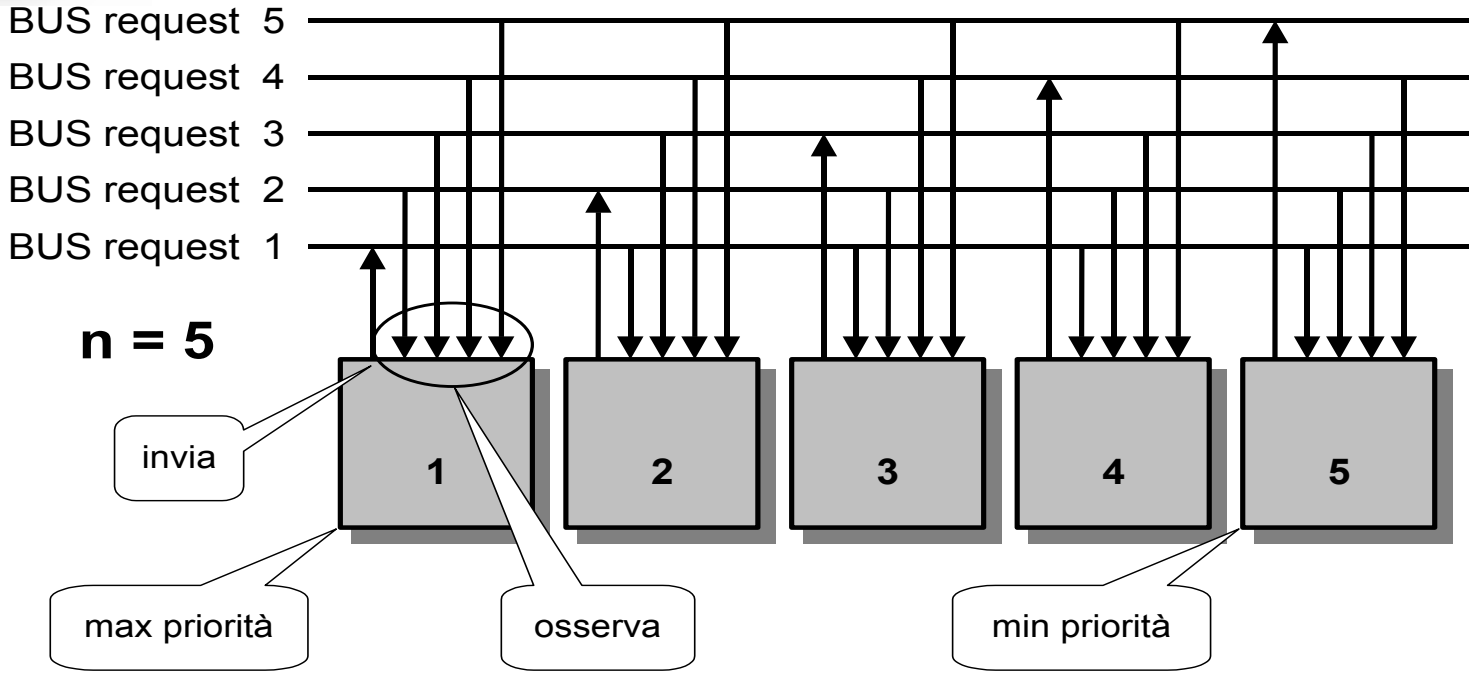


- contiene più festoni di richiesta/conferma con livelli di priorità differenti;
- ogni unità usa il festone cui è collegata;
- in caso di richieste simultanee su due o più festoni, l'arbitro invia il grant a quello con priorità più alta

Arbitraggio distribuito

- Tutte le unità partecipano con uguale responsabilità al processo di cessione del ruolo di master
- Necessario disporre di circuiti logici di arbitraggio su ogni interfaccia di I/O
- Si utilizzano in genere linee del bus di controllo su cui i dispositivi pongono il proprio codice identificativo per indicare che richiedono l'utilizzo del bus
- Esistono tecniche diverse di arbitraggio distribuito
- L'interfaccia deve essere in grado di leggere e scrivere contemporaneamente sulle linee di arbitraggio
  - I valori scritti vengono modificati in funzione di quanto letto per realizzare la funzione di arbitraggio

Soluzione semplice: 1 linea per dispositivo

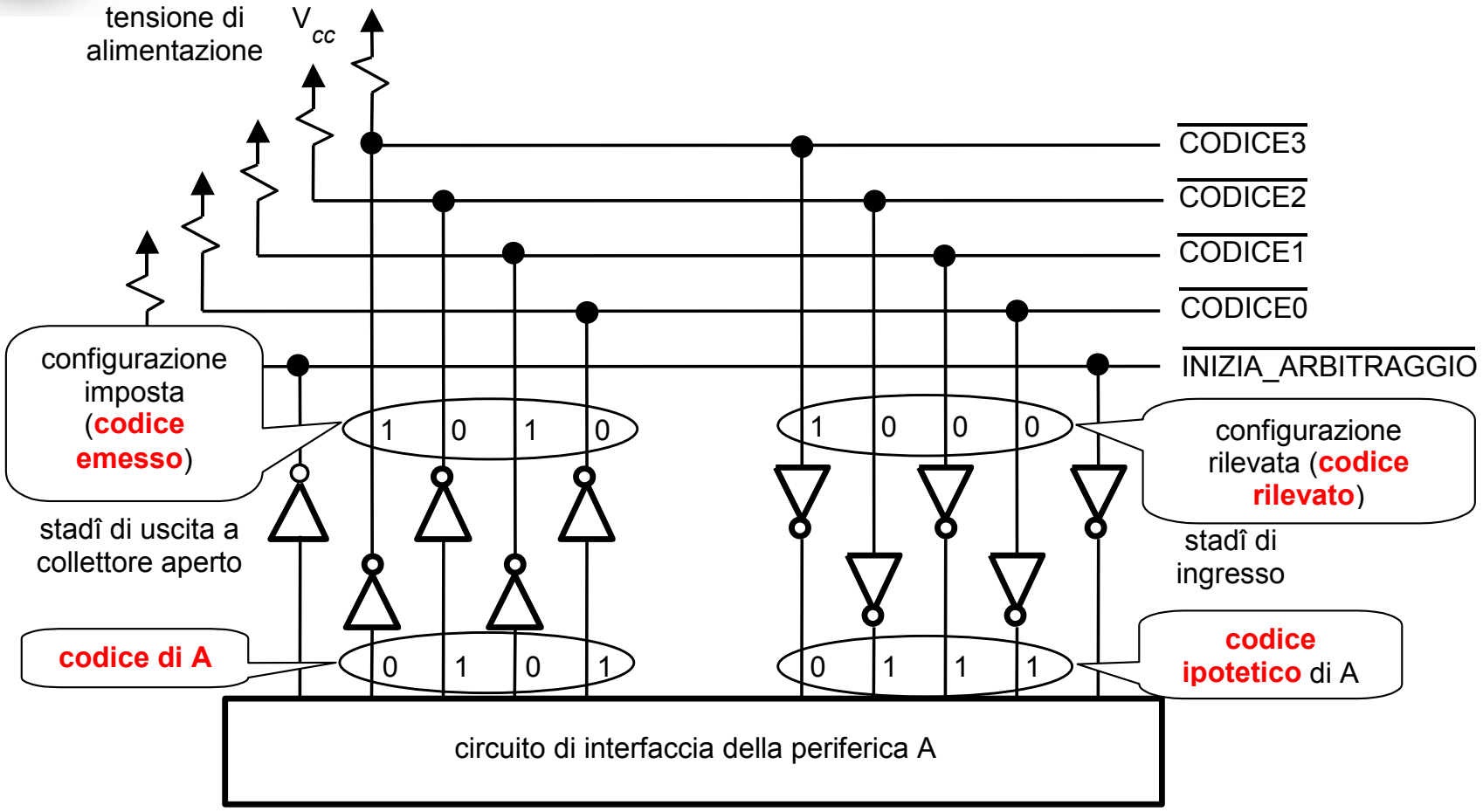


•Se l'unità vuole diventare MASTER, si accerta che nessun'altra unità a priorità superiore abbia attivato la propria linea di richiesta BUS
•Se così è, l'unità attiva la sua linea di richiesta BUS e ottiene il controllo del BUS per il ciclo successivo, diventando MASTER
•Si risparmia il costo dell'arbitro, ma il numero di linee di controllo cresce con quello delle unità

Arbitraggio distribuito con codice identificativo per le interfacce

- Il codice determina la priorità prefissata dell'unità
- Ogni unità possiede un codice identificativo a 4 bit
  - Nell'esempio sono previste 16 unità
- L'unità che ottiene il controllo del bus è quella con priorità più alta (valore del codice identificativo più alto)
- 5 linee di controllo del bus:
  - 1 linea per il controllo dell'inizio della procedura di arbitraggio (INIZIA_ARBITRAGGIO)
  - 4 linee per il codice identificativo (CODICE0 – CODICE4)
  - Le linee CODICEi sono collegate in modo open collector che funziona da negatore (se l'unità pone 0101 sulle linee CODICE si vedrà 1010)
  - Se due unità tentano di porre il proprio codice sulle linee di codice nello stesso istante, per esempio 0101 e 0110 dalla sovrapposizione si ottiene 1000 perché in open collector vince chi cerca di portare a 0 la linea

Arbitraggio distribuito: schema di collegamento di una unità



Procedura

- Tutti scrivono il proprio indirizzo
- Partendo dal bit più significativo se riscontrano una differenza con il proprio indirizzo mettono a 0 il bit corrispondente e tutti quelli a priorità più bassa
- Il valore delle linee può oscillare per un numero di cicli che è al massimo pari al numero di linee di codice
- Poi si stabilizza al valore uguale all'indirizzo di chi ha vinto, che riconosce il proprio indirizzo e diventa master del bus

Esempio

- Due unità:
  - A codice identificativo 5 = 0101 (**codice emesso** = 1010)
  - B codice identificativo 6 = 0110 (**codice emesso** = 1001)
- Richiesta del bus scrivendo il proprio codice su CODICE0-CODICE4
  - Risultato della sovrapposizione 1000 (**codice rilevato**)
- Le due unità leggono il valore negato che corrisponde al codice 0111(= codice ipotetico della periferica)
- A e B confrontano tale codice con il proprio a partire dal bit più significativo
- La prima unità che trova una differenza di bit tra il proprio codice e quello letto impone a 0 tutti i bit a partire da quello da cui si differenzia fino a quello meno significativo
  - A identifica nel terzo bit a partire da quello più significativo una differenza (0 nel codice e 1 nel valore ipotetico), quindi a partire da questo bit impone 0 e cioè 0100 che negato diventa sul bus 1011 (nuovo codice emesso)
  - sovrapposto a 1001 diventa 1001, che negato è uguale al codice di B
- B non rileva differenza di bit tra il suo codice e quello sulle linee e si considera vincitore e detentore del bus nel ruolo di master
- Il risultato rimane invariato indipendentemente dall'ordine temporale di intervento delle unità coinvolte

Conclusioni

- Arbitraggio distribuito più affidabile di quello centralizzato
  - Meno soggetto a errori e guasti perchè la procedura non dipende da un solo dispositivo
- Esempio di arbitraggio distribuito: bus SCSI