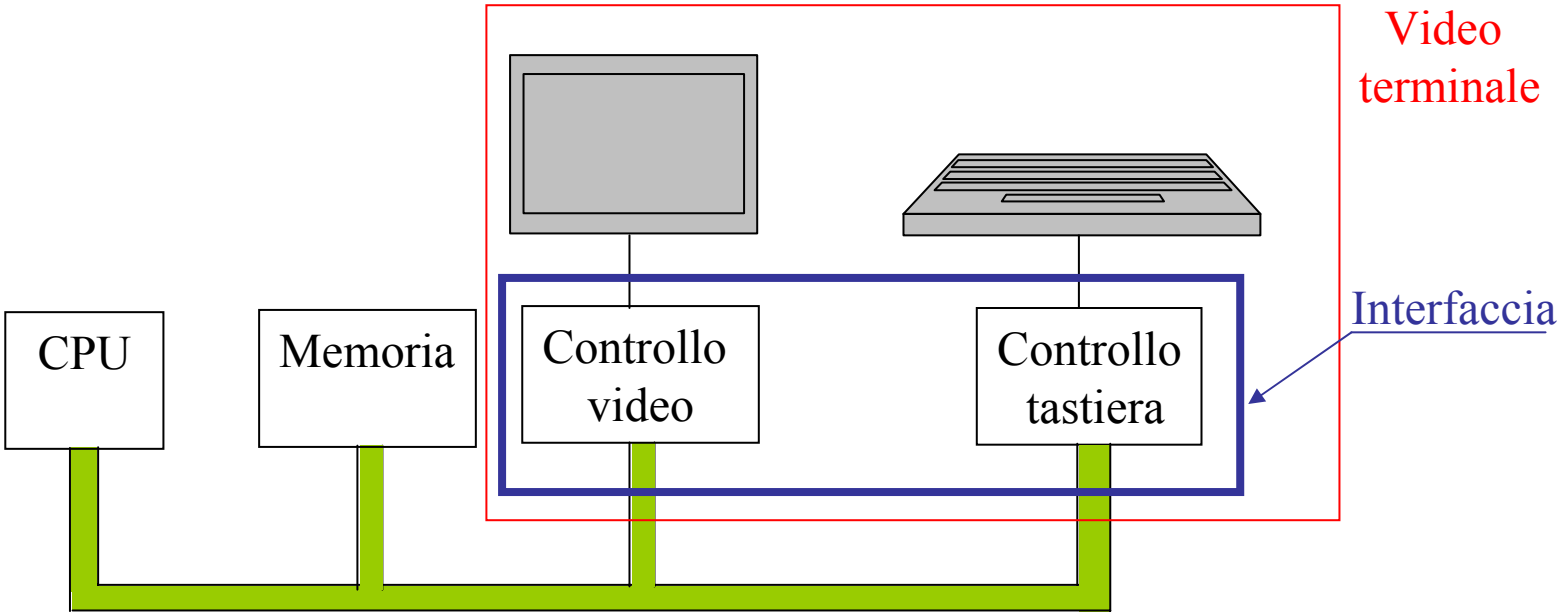
Calcolatori Elettronici B
a.a. 2005/2006

INPUT/OUTPUT: BUS

Massimiliano Giacomin

Schema semplice con bus unico (utilizzato nei primi calcolatori)



- Unico bus cui sono connessi tutti i dispositivi: CPU + Memoria + dispositivi I/O
- Fisicamente, nel bus è presente un certo numero di "slot" cui sono connessi i controller dei dispositivi (ovvero le interfacce ingresso-uscita); alcuni sono già predisposti (es. mouse e tastiera), altri sono a disposizione dell'utente per inserire i controller voluti (es. scheda video, scheda audio, ecc.)

PROBLEMI DELL'ORGANIZZAZIONE A BUS SINGOLO

- Coesistenza di dispositivi che lavorano a velocità diverse:
  - P.es. supponendo la presenza di un clock cui è dedicata una linea del bus, alcuni dispositivi lenti (lavorano con un clock di bassa frequenza) ed altri dispositivi veloci
    $\Rightarrow$ il clock deve adattarsi al dispositivo più lento:
    tutti i dispositivi lavorano alla frequenza del dispositivo più lento!
- Aumento lunghezza bus e aumento dispositivi fisici collegati
  $\Rightarrow$ (per ragioni fisiche) diminuzione della velocità

$\Rightarrow$ necessari diversi bus a seconda delle caratteristiche fisiche e dispositivi collegati
(p.es. bus processore-memoria diverso da bus di I/O)

• Nei primi calcolatori, unico bus per dispositivi “bilanciati”.
Con l’aumento di velocità di periferiche, processori, memorie:
inadeguatezza del bus

⇒ Impossibile abbandonare il vecchio bus per garantire la compatibilità con le periferiche già prodotte (ragione commerciale)

⇒ Soluzione: avere bus multipli!

P.es. al giorno d’oggi è comune avere – tra gli altri – i bus:

- bus processore-memoria proprietario per supportare alta velocità
- bus PCI (Peripheral Component Interconnect): bus standard veloce per periferiche più recenti
- bus ISA (Industry Standard Architecture) nella sua versione estesa (EISA) per supportare le periferiche meno recenti

QUINDI:

- Tutte le attuali organizzazioni del calcolatore prevedono più tipologie di bus
- Di questi, i bus I/O sono <u>standard</u> – adeguamento produttori di periferiche e produttori di calcolatori per garantire che le periferiche prodotte siano compatibili con i calcolatori e viceversa

Una categorizzazione dei bus nelle architetture a più bus

NB: ci riferiamo sempre a bus esterni alla CPU, ovvero bus che connettono CPU, Memorie e Controller I/O
Alcune CPU hanno bus interni… ma questi non riguardano l'argomento che stiamo trattando

- Bus processore-memoria:
  - corti
  - alta velocità
  - ottimizzati rispetto alle caratteristiche del sistema di memoria
  - tipicamente non standard
- Bus di I/O:
  - maggiore lunghezza
  - permettono di collegare dispositivi di tipo diverso
  - spesso consentono ampio intervallo della banda dei dispositivi connessi
  - standard
- Bus di backplane:
  - sono un compromesso tra i due (possono essere utilizzati come bus unico o come bus di I/O cui sono collegati altri bus di I/O)
  - tipicamente standard

Proprietà caratteristiche del bus (talvolta fissate da uno standard)

- Le linee del bus possono essere suddivise in:
  - indirizzi (individuano il dispositivo)
  - dati (contengono i dati da trasferire)

  } Talvolta riuniti in un unico gruppo di linee

  - controllo: stabiliscono il tipo di operazione in corso, cosa contengono le linee dati/indirizzi in quel momento, segnalano richieste e conferme, ecc.
    In generale: utilizzate per implementare il protocollo del bus

- Protocollo: l'insieme di regole che definiscono come il bus lavora
  (p.es. il significato dei segnali di controllo, la successione delle operazioni, ecc.) che devono essere rispettate dai dispositivi
  + specifiche elettriche (voltaggi, tempistica segnali, ecc.)
  + specifiche meccaniche (disposizione fisica connettori slot)

Tipicamente, nella specifica del protocollo si fa riferimento a due "ruoli" per i dispositivi che accedono al bus…

Master & Slave: Modo comune di lavoro

- Alcuni dispositivi possono “conquistare” l’accesso al bus e cominciare il trasferimento dei dati [master]
- Altri dispositivi rimangono in attesa di essere “chiamati” dai master [slave]
- Esistono dispositivi che possono essere sia master che slave

Esempi:

memoria: slave

processore: master

dispositivo-DMA (es. controller del disco):

può essere sia slave (quando riceve comandi dalla CPU)

sia master (quando comanda memoria e disco per leggere o scrivere i dati nel trasferimento da/a disco)

Principali scelte progettuali

NB: queste scelte possono corrispondere a bus proprietario oppure aderire ad uno standard!

Ampiezza bus
- Indirizzo
- Dati

Temporizzazione
- Sincrona
- Asincrona

Metodo di arbitraggio
- Centralizzato
- Distribuito

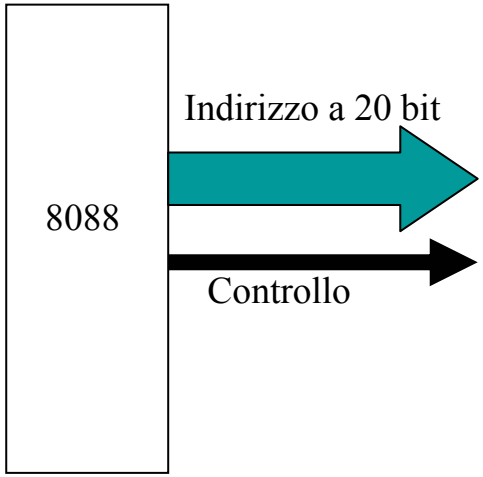
Tipi di transazioni sul bus previste dal protocollo
- Scrittura
- Lettura
- Lettura, modifica e scrittura
- …
- Blocco

VEDIAMO IN DETTAGLIO…

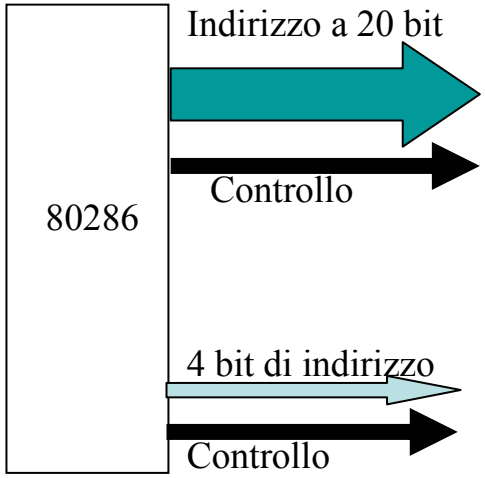
Ampiezza del bus

1) LINEE INDIRIZZI: il numero determina quante locazioni di memoria (e dispositivi di I/O) sono indirizzabili: n linee, 2^n spazio degli indirizzi

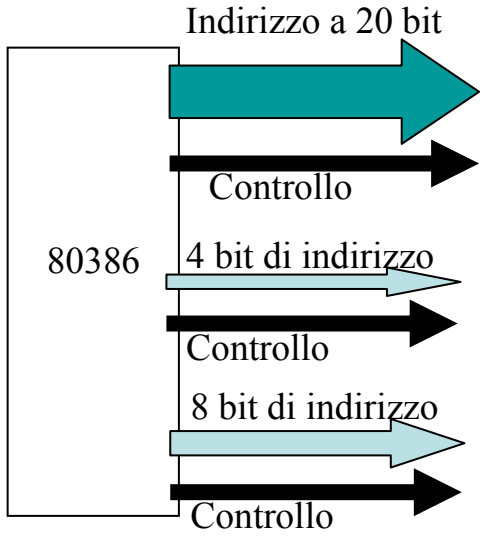
Aumento linee ⇒ aumento spazio fisico occupato + dimensioni connettori + costo



Il PC originario indirizzava 1MB di memoria



La seconda generazione indirizzava 16 MB di memoria. Le nuove linee sono aggiuntive senza disturbare il bus originale per ragioni di compatibilità con le generazioni precedenti. (backward compatibility)



Finalmente il bus EISA arriva a 32 bit ma con un progetto più confuso di quanto si sarebbe potuto fare partendo subito con 32 bit

2) LINEE DATI

Aumentare la banda: due possibilità

- Aumentare la velocità di trasmissione (più trasferimenti/sec)
  Possibile ma difficile: problemi di **skew** (allineamento segnali su linee diverse) +
  difficile mantenere compatibilità con schede più vecchie
  (backward compatibility) che possono non supportare le velocità più elevate
- Aumentare le linee dati: a parità di trasferimenti/sec, si hanno più bit/trasferimento
  e quindi più bit/secondo!

Anche in questo caso, le linee sono state aggiunte incrementalmente…

A volte, per risparmiare i costi dovuti all'ampiezza del bus:

Multiplexare le linee: usare le stesse linee per scopi diversi in tempi diversi
Bus con meno linee (meno costi di costruzione)
ma più lento (più costi di gestione)
ES: dati e indirizzi sulle stesse linee; per scrivere una parola in memoria, bisogna inviare l'indirizzo e, successivamente, il dato: trasferimento più lento!

Temporizzazione del BUS

Esistono due categorie di bus:

• BUS SINCRONO:

- una apposita linea trasporta un segnale di clock, che arriva ai dispositivi connessi
- le operazioni di trasferimento sono “sincronizzate” dal clock e, quindi, impiegano un numero intero di cicli di clock
- i dispositivi devono essere in grado di lavorare alla frequenza del clock [eventualmente utilizzando anche “cicli di attesa”]

• BUS ASINCRONO:

- non è presente un segnale di clock, ma i dispositivi che di volta in volta comunicano si scambiano segnali di sincronizzazione
- i “cicli di clock” risultanti possono avere qualsiasi lunghezza a seconda della velocità dei dispositivi e, quindi, variano a seconda della coppia di dispositivi che di volta in volta si scambiano dati

BUS SINCRONO

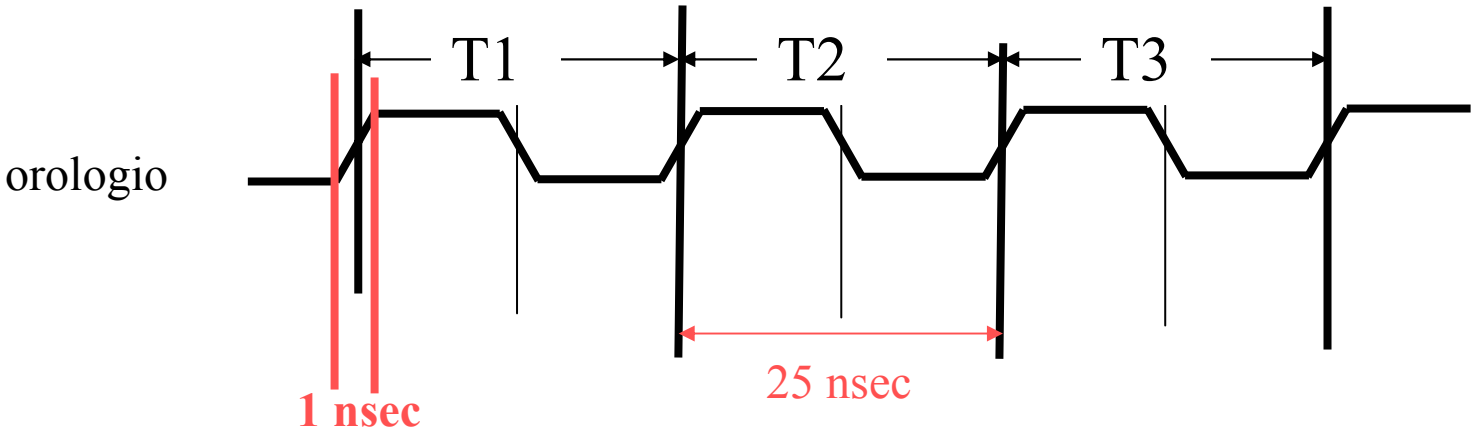
- Un bus sincrono ha una linea guidata da un clock (frequenza di clock).
  Il segnale su questa linea è un'onda quadra (ordine di grandezza: 5-n*100 Mhz).
  NB: notare che la frequenza del bus è molto inferiore rispetto a quella del processore!

- C'è clock: ogni attività parte sul fronte di salita
- L'attività parte con un master che indica l'indirizzo dello slave e cosa vuol fare
- Nel corso dell'attività, se necessario,lo slave indica quando è pronto e può indicare mediante un segnale al master di "aspettare" dei cicli di attesa
- In ogni caso, master e slave devono rispettare dei vincoli temporali (p.es. rispetto ai fronti di salita e di discesa del clock)
- In generale, una transazione sul bus comprende due fasi:
  - invio indirizzo
  - ricezione/invio dato

Vediamo un esempio: lettura di una parola dalla memoria alla CPU

Ipotesi assunte:

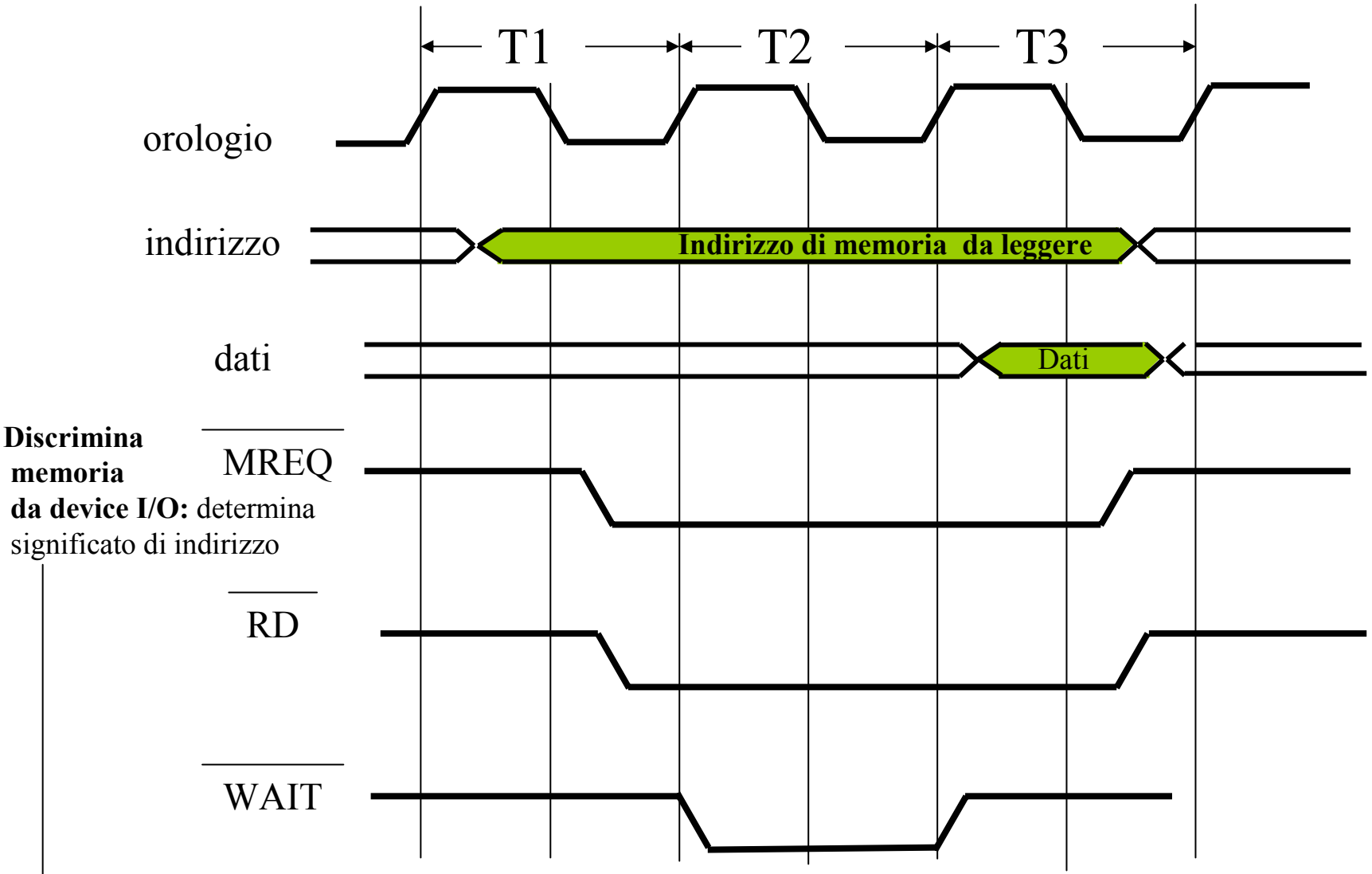
- clock 40 MHz (ciclo di 25 nsec)
- Memoria: la lettura del dato dal momento in cui indirizzo è stabile impiega 40 ns



Descrizione di massima:

- nel primo ciclo la CPU invia alla memoria l'indirizzo
- nei cicli successivi, la memoria effettua l'operazione e pone il dato nel bus: necessaria una linea con cui possa segnalare alla CPU di aspettare il dato

ORA VEDIAMO IL DIAGRAMMA TEMPORALE…



Cfr. memory mapped I/O vs. istruzioni speciali [è questo il caso]

Note sul diagramma precedente

- Il diagramma temporale presenta una ***lettura sincrona*** da memoria (slave) da parte della CPU (master)
- Linee di indirizzo e linee di dato separate
- Linee di controllo: $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WAIT}}$ (attive a livello basso!)
- Nell'intervallo temporale T_1 la CPU mette l'indirizzo della locazione che vuole leggere nelle linee di indirizzo del bus
- Dopo il fronte di discesa del clock in T_1 (dopo che il segnale sulle linee di indirizzo è divenuto stabile) la CPU asserisce le linee $\overline{\text{MREQ}}$ e $\overline{\text{RD}}$
- Il primo segnale indica una richiesta di accesso in memoria
- Il secondo segnale indica una richiesta di lettura da memoria (è asserito per lettura o de-asserito per scrittura)
- La memoria non può fornire i dati durante T_2 (impiega 40 ns dopo che l'indirizzo è presente nel bus) per cui asserisce il segnale $\overline{\text{WAIT}}$ all'inizio di T_2 per indicare alla CPU di non aspettarsi i dati sulla linea dei dati
- All'inizio di T_3, quando è sicura che fornirà i dati durante il ciclo corrente, la memoria nega il segnale $\overline{\text{WAIT}}$

Note sul diagramma precedente (continua)

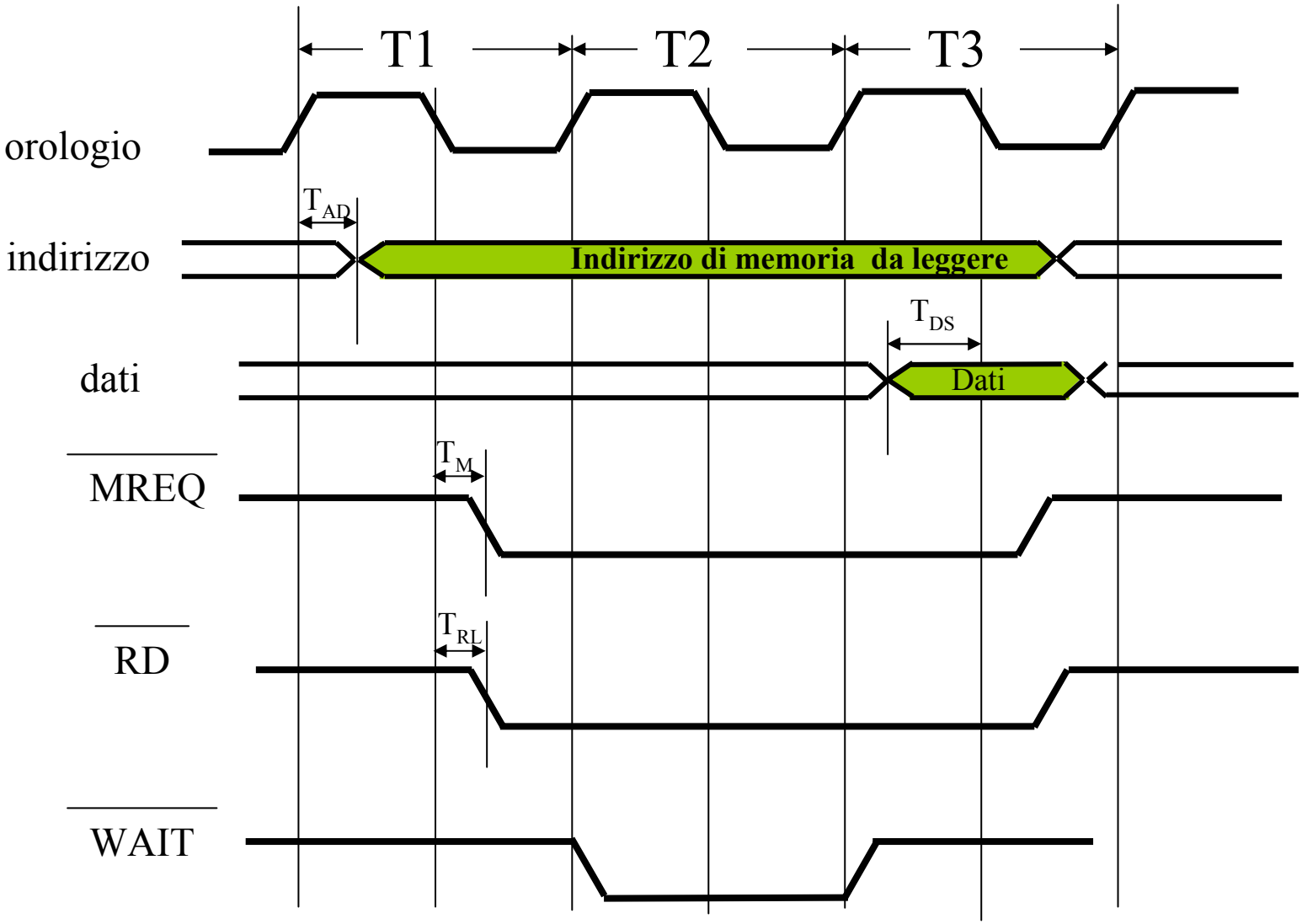
- Nella prima metà di T_3 la memoria inserisce i dati sulle linee di dato, mentre sul fronte di discesa di T_3 i dati vengono prelevati dal bus da parte della CPU
- A questo punto i segnali $\overline{MREQ}$ e $\overline{RD}$ possono essere disasseriti
- Con un bus sincrono si ha un protocollo fisso per la comunicazione basato sulla tempistica del clock
- In particolare, occorre fare attenzione ai vincoli imposti dalla CPU, che devono essere rispettati dalla memoria!

Esempi di vincoli specificati dalla CPU:

- ritardo T_{AD} ($\leq$ 11 nsec) tra la salita del fronte del clock e l'istante in cui un indirizzo è stabilito sulle linee;
- T_{DS} ($\geq$ 5 nsec) il tempo di setup dei dati: necessario che il dato sia posto T_{DS} ns prima della discesa del clock in cui CPU lo acquisisce

⇒ Dopo che l'indirizzo è presente sulle linee, la memoria deve fornire il dato in al più (25*2.5 – 11 – 5) = 46.5 ns

⇒ La memoria rispetta il vincolo (40 ns) ma se avessimo usato una memoria da 50 ns, necessario un altro ciclo di attesa!



T_M, $T_{RL} \leq 8$ ns

Altro vincolo per la memoria: da $\overline{\text{MREQ}}$ e $\overline{\text{MD}}$, dato disponibile entro

$25*2 - 8 - 5 = 37$ ns

NB: in realtà esistono molti altri vincoli di questo genere…

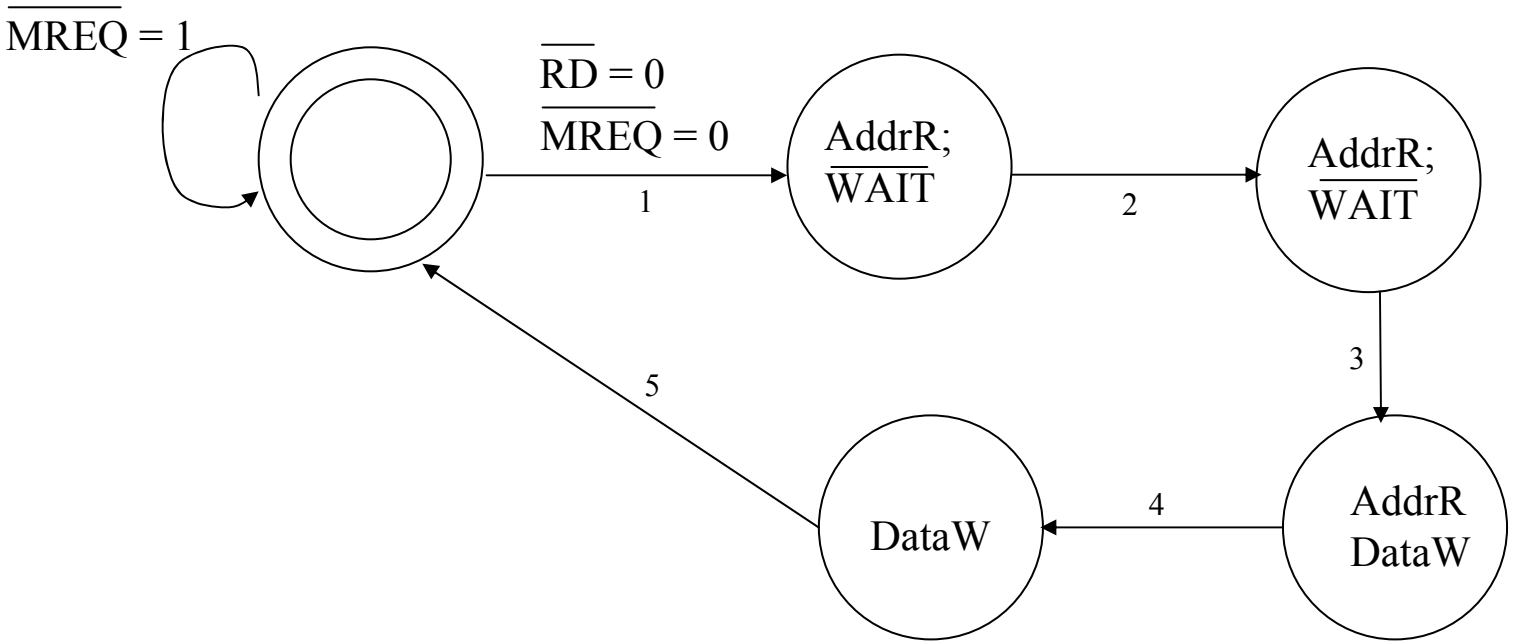
NB: il protocollo di comunicazione adottato da parte del master e dello slave può essere specificato con un automa a stati finiti le cui transizioni sono regolate dai ***fronti di salita e di discesa*** del clock.
Come si vedrà, ciò è diverso dal caso degli ***automi di handshaking*** utilizzati per specificare il protocollo di comunicazione su bus asincrono dove le transizioni sono specificate in corrispondenza ***dell'arrivo di un segnale di input*** (es. richiesta, ack, etc.)

La specifica del protocollo mediante automi a partire dal diagramma temporale rappresenta un **esercizio tipico**: vediamo il caso precedente per master e slave…

Partiamo dallo slave, che è più semplice…

Macchina a stati finiti (per lo slave)

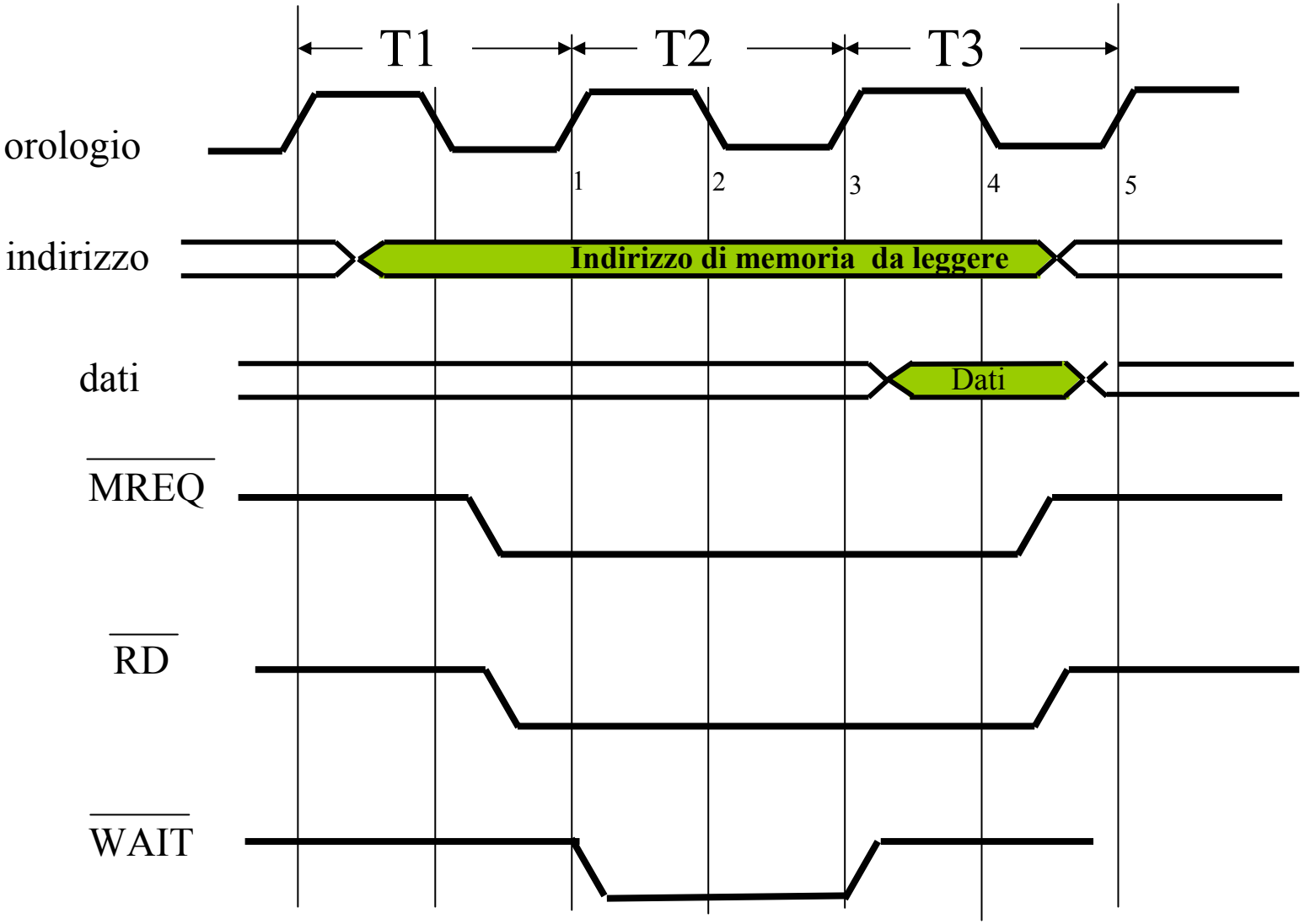
che implementa il protocollo di comunicazione per la lettura su bus sincrono



- AddR: segnale di controllo che abilita la lettura
  [da parte della memoria] dell'indirizzo presente sul bus
- DataW: segnale di controllo che abilita la scrittura sul bus [da parte della memoria]
  del dato prelevato dalla memoria stessa

Nota: MREQ è un segnale proveniente dal master, che costituisce per lo slave un ingresso

Spiegazione diagramma precedente



1: memoria rileva (durante la seconda metà di T1) il segnale $\overline{\text{MREQ}}$;
attiva AddR e $\overline{\text{WAIT}}$, che rimangono attivi durante la prima metà di T2

Spiegazione diagramma precedente (continua)

2: memoria continua ad attivare AddR e $\overline{\text{WAIT}}$ per la seconda metà di T2

3: memoria pone il dato nel bus (disponibile per tutta la prima metà di T3)

4: memoria continua a porre il dato nel bus e disattiva AddR

[verso la fine di T3 il dato non è più stabile]

5: alla fine di T3, transizione verso lo stato iniziale

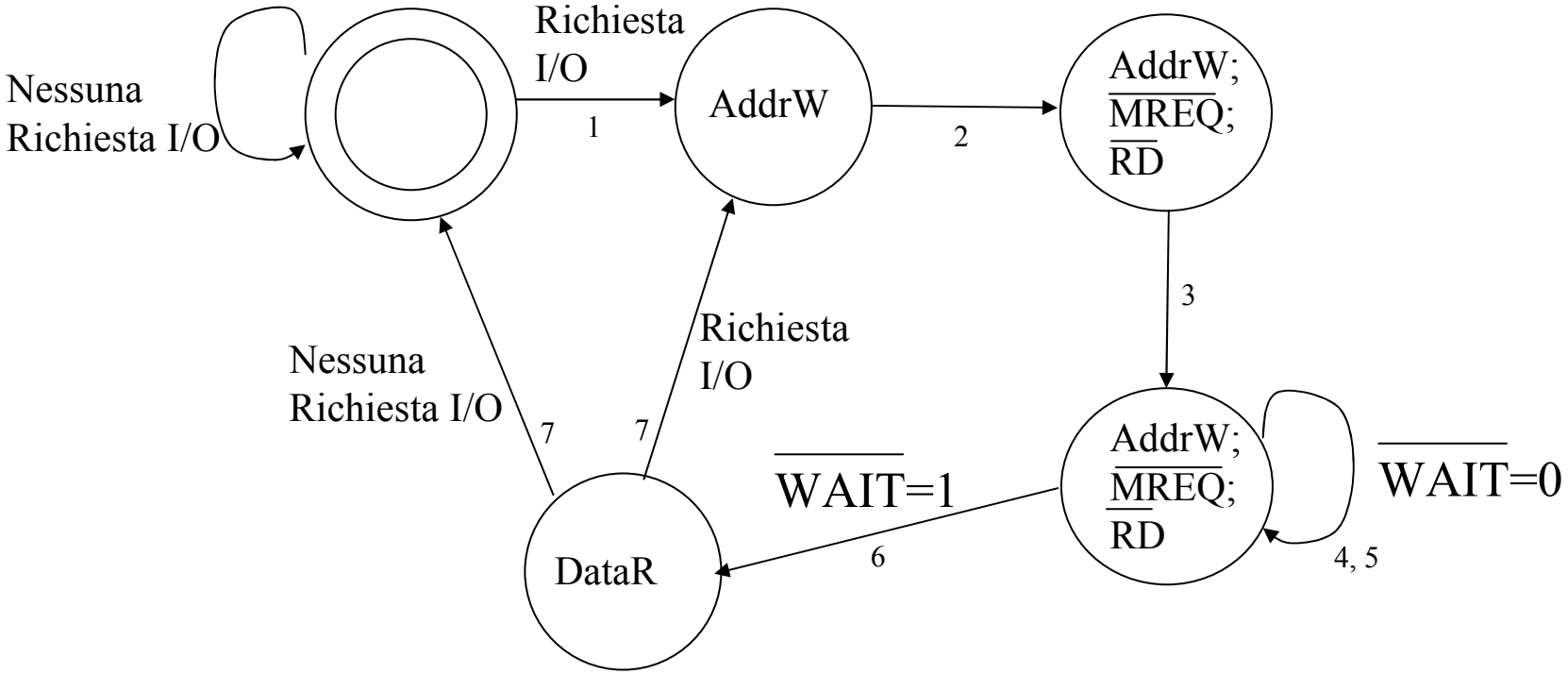
(si ricomincia dall'inizio nella prima metà di T1)

NB: si tratta di una descrizione di alto livello

(dettagli AddR, DataW ecc. inseriti per rendere l'idea, nella risoluzione dell'esercizio c'è un certo grado di arbitrarietà…)

Macchina a stati finiti (per il master)

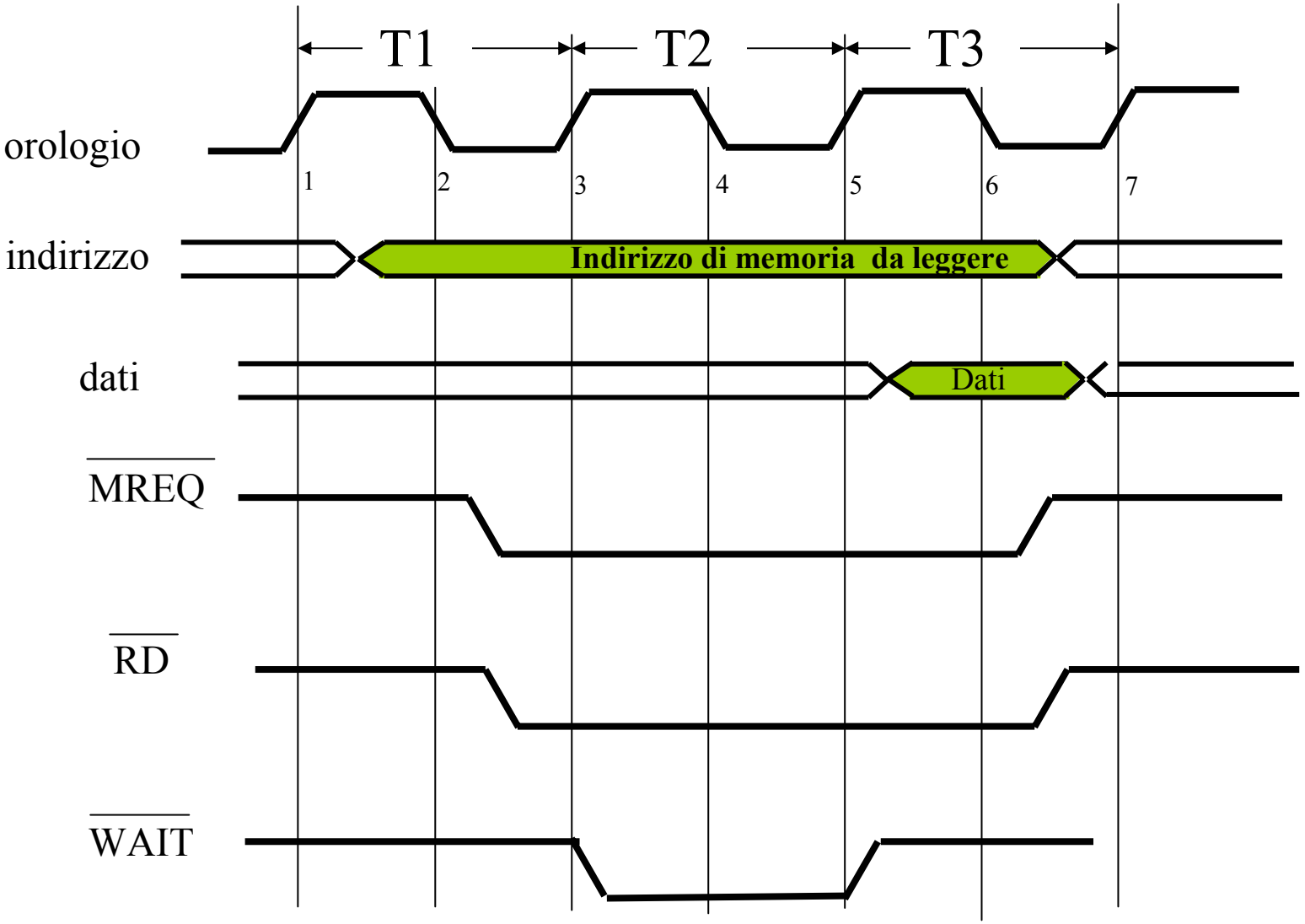
che implementa il protocollo di comunicazione per la lettura su bus sincrono



Nota: "AddrW" e "DataR" sono segnali di controllo che abilitano rispettivamente l'invio indirizzo [da parte del master] e l'acquisizione del dato [da parte del master] reso disponibile sul bus dallo slave.

Il segnale $\overline{\text{WAIT}}$ proviene dallo slave ed è un ingresso al master

Spiegazione diagramma precedente



Spiegazione diagramma precedente (continua)

1: La richiesta di I/O, arrivata nel semiciclo precedente a T1, viene rilevata e comincia la transazione: nella prima metà di T1 il master pone l'indirizzo sul bus mediante il segnale AddrW

2: Il master, nella seconda metà di T1, fa la richiesta di lettura alla memoria continuando a porre l'indirizzo sul bus.
Notare che il segnale $\overline{\text{WAIT}}$ non viene rilevato in questo semiciclo

3: Il master entra nella prima metà di T2, in cui la memoria eventualmente segnala il ciclo di attesa

4,5: Il master rimane in attesa finché $\overline{\text{WAIT}}$ non viene disattivato. Si noti che la rilevazione avviene nella prima metà di T3, quindi si esce da questo stato (con la transizione 6) alla fine della prima metà di T3

6: dopo aver rilevato la cessazione di $\overline{\text{WAIT}}$ nella prima metà di T3, il master (nella seconda metà di T3) acquisisce il dato e disabilita la richiesta alla memoria

7: alla fine di T3 si ritorna allo stato iniziale; se però si vuole poter ricominciare una nuova transazione già al ciclo successivo, è necessario già ora rilevare l'eventuale richiesta di I/O [in alternativa: si può supporre che vi sia un ciclo "idle" tra una transazione e la successiva]

NB: i più attenti possono aver notato che, se la richiesta di I/O arriva nella prima metà del ciclo precedente a T1 (o successivo a T3) si potrebbe entrare "sfasati di mezzo ciclo" nello stato che abilita AddrW.

⇒ si può supporre che ciò non avvenga [richiesta fatta solo nella seconda metà del ciclo]

oppure si può modificare il diagramma "sdoppiando" lo stato di attesa [da fare per esercizio]

NB2: come si è notato, alcuni segnali sono attivi quando hanno valore logico 1, altri (contrassegnati dal simbolo di negazione) sono attivi con valore logico 0: la scelta è progettuale, dipendente dalle convenzioni utilizzate e quindi può variare a seconda dei casi

Commenti sul bus sincrono

Aspetti positivi:

- Semplici da gestire e da realizzare; logica delle interfacce ridotta
- Hardware di controllo semplice ⇒ può raggiungere un'elevata velocità

Aspetti negativi:

- Problema del disallineamento del clock (vedi lucidi prima lezione):
  per conseguire velocità elevate, necessario mantenere lunghezza bus ridotta
  [tutti i bus processore-memoria sono sincroni]
- Tutti i dispositivi devono lavorare alla stessa frequenza di clock:
  per unità di diverse velocità, il ciclo di clock è legato alla più lenta mentre le unità più veloci non sono pienamente sfruttate
- E' impossibile completare la transazione in un numero non intero di cicli
  [se transazione può concludersi in 3.1 cicli, occorre aspettare la fine del quarto]
- Difficile adattamento ai miglioramenti tecnologici: una volta che si è scelto il periodo di clock, aumento velocità dispositivi può non offrire vantaggi.
  - Nell'esempio precedente, memoria da 20ns evita il ciclo di WAIT, riducendo tempo di transazione a 25*2=50ns, però una memoria p.es. da 10ns richiede comunque due cicli e non diminuisce il tempo di transazione

BUS ASINCRONO

- Per superare limiti del bus sincrono si abolisce il clock:
  ciascun dispositivo risponde ai segnali dell'altro "il più velocemente possibile"
  senza sincronizzazione con un segnale di clock fisso.
- Il "ciclo di bus" diviene variabile anche per coppie di dispositivi: velocità
  diversa a seconda delle capacità dei dispositivi.
- La mancanza di problemi di disallineamento del clock permette una lunghezza
  maggiore del bus.

Le transazioni in assenza della sincronizzazione mediante il segnale di clock si basano sul protocollo di **handshaking** (*stretta di mano*)

PROTOCOLLO DI HANDSHAKING: IDEA DI BASE

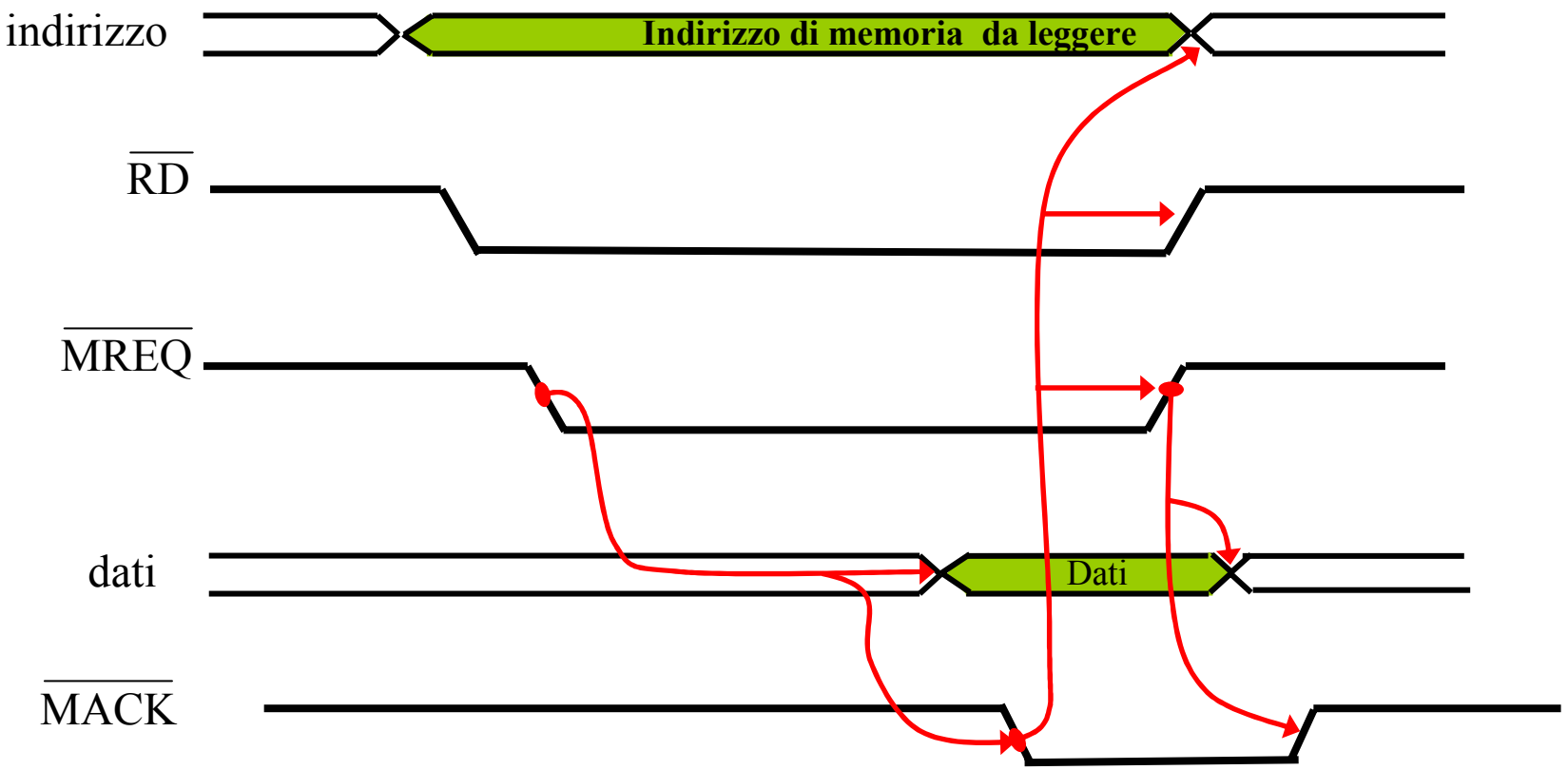
Il protocollo si basa su due segnali (i nomi sono solo indicativi!):

- REQUEST, inviato dal master allo slave
- ACK, inviato dallo slave al master

La transazione si svolge in quattro passi:

1. Il master richiede un servizio allo slave mediante un opportuno segnale (REQUEST)
2. Lo slave, ricevuto il segnale, comincia ad eseguire il servizio secondo i suoi tempi. Poiché il master non sa quando finisce, è necessario che lo slave, una volta eseguito il servizio [e prodotto il dato] mandi un segnale di accettazione (ACK)
3. Il master riceve il segnale di ACK e può allora utilizzare il risultato del servizio richiesto (p.es. accedendo ai dati nel bus).
   Poiché lo slave non sa quando il master finisce di utilizzare i dati, il master segnala ciò riabbassando REQUEST
4. A questo punto, lo slave rileva l'abbassamento di REQUEST e può quindi terminare la transazione (p.es. ritirando il dato dal bus) abbassando ACK

UN ESEMPIO: Lettura di una parola dalla memoria su bus asincrono



$\overline{\text{MREQ}}$: segnale di richiesta del master di accedere alla memoria ($\overline{\text{RD}}$ indica lettura)

$\overline{\text{MACK}}$: segnale di acknowledgement dello slave (memoria)

CHIARIMENTI SUL LUCIDO PRECEDENTE:

- Le linee rosse con frecce indicano relazioni di causa-effetto tra segnali
- Secondo il protocollo di handshaking, le fasi sono:
  - Master pone indirizzo sul bus e richiede lettura con $\overline{\text{MREQ}}$
  - Memoria effettua l'operazione di lettura e, quando pronta, pone il dato sul bus e segnala ciò al master con $\overline{\text{MACK}}$
  - Il master acquisisce il dato e, una volta fatto, lo segnala con $\overline{\text{MREQ}}$ (contestualmente ritira l'indirizzo dal bus e alza $\overline{\text{RD}}$)
  - La memoria allora termina di porre il dato sul bus e alza $\overline{\text{MACK}}$

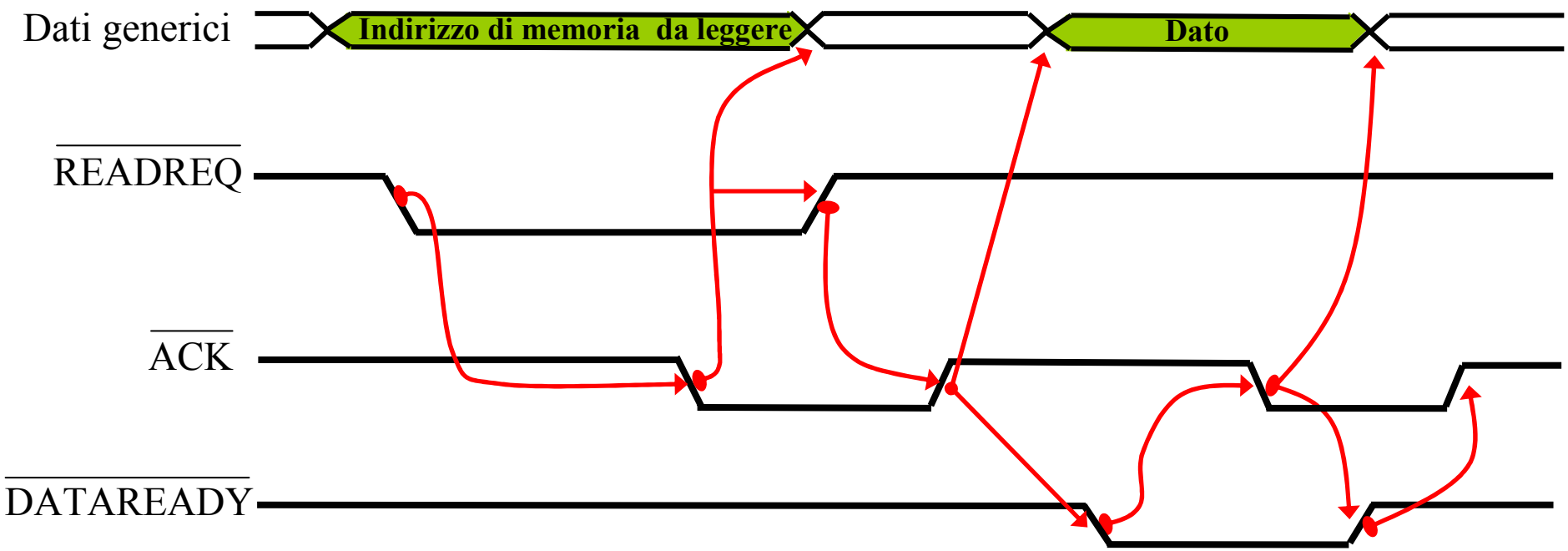
NB: sono possibili varianti di questo schema base.

Ad esempio il significato dei segnali può cambiare

P.es., nel caso precedente avevamo linee diverse per dati e indirizzi;

vediamo come cambiano le cose se nel bus si hanno le stesse linee per dati e indirizzi…

UN ESEMPIO: Lettura di una parola da memoria con unica linea per indirizzi e dati



In questo caso supponiamo di avere i segnali (cfr. Patterson, esempio a pag. 584):

READREQ: segnale di richiesta del master di leggere la memoria (prima se ne usavano due)

DATAREADY: segnale con cui si segnala che il dato è pronto (funge da REQ per il servizio di accesso al dato)

ACK: segnale di acknowledgement usato dalla memoria e dal master!

CHIARIMENTI SUL LUCIDO PRECEDENTE:

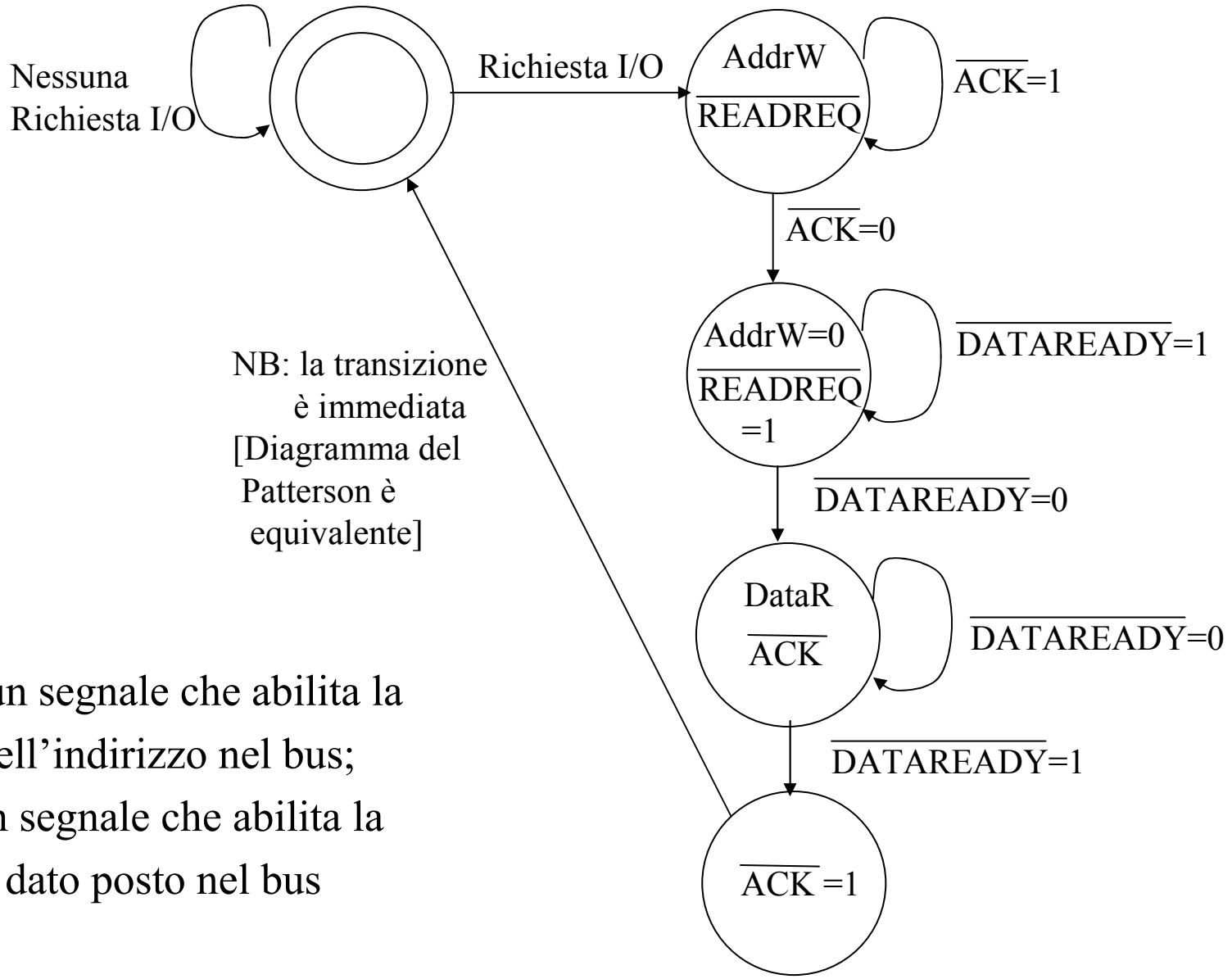
- In pratica il protocollo di handshaking è usato due volte; le fasi sono:
  - Master pone indirizzo sul bus e richiede lettura con $\overline{\text{READREQ}}$
  - Memoria acquisisce l'indirizzo e, quando questa operazione è terminata, lo segnala al master con $\overline{\text{ACK}}$.

    NB: la memoria non può porre il dato sul bus perché occupato dall'indirizzo: è necessario avvisare il master (appunto con $\overline{\text{ACK}}$) affinché liberi il bus.
  - Il master libera il bus e alza $\overline{\text{READREQ}}$
  - A questo punto la memoria alza $\overline{\text{ACK}}$ e comincia l'operazione di lettura
  - Quando il dato è pronto, memoria lo pone sul bus e segnala al master con $\overline{\text{DATAREADY}}$
  - Il master acquisisce il dato e, appena terminato, lo segnala alla memoria ($\overline{\text{ACK}}$)
  - La memoria ritira il dato dal bus e alza $\overline{\text{DATAREADY}}$
  - Il master alza $\overline{\text{ACK}}$, indicando il completamento della trasmissione

Esercizio tipico

Dato il diagramma temporale, ricavare gli automi per il master e per lo slave

Master



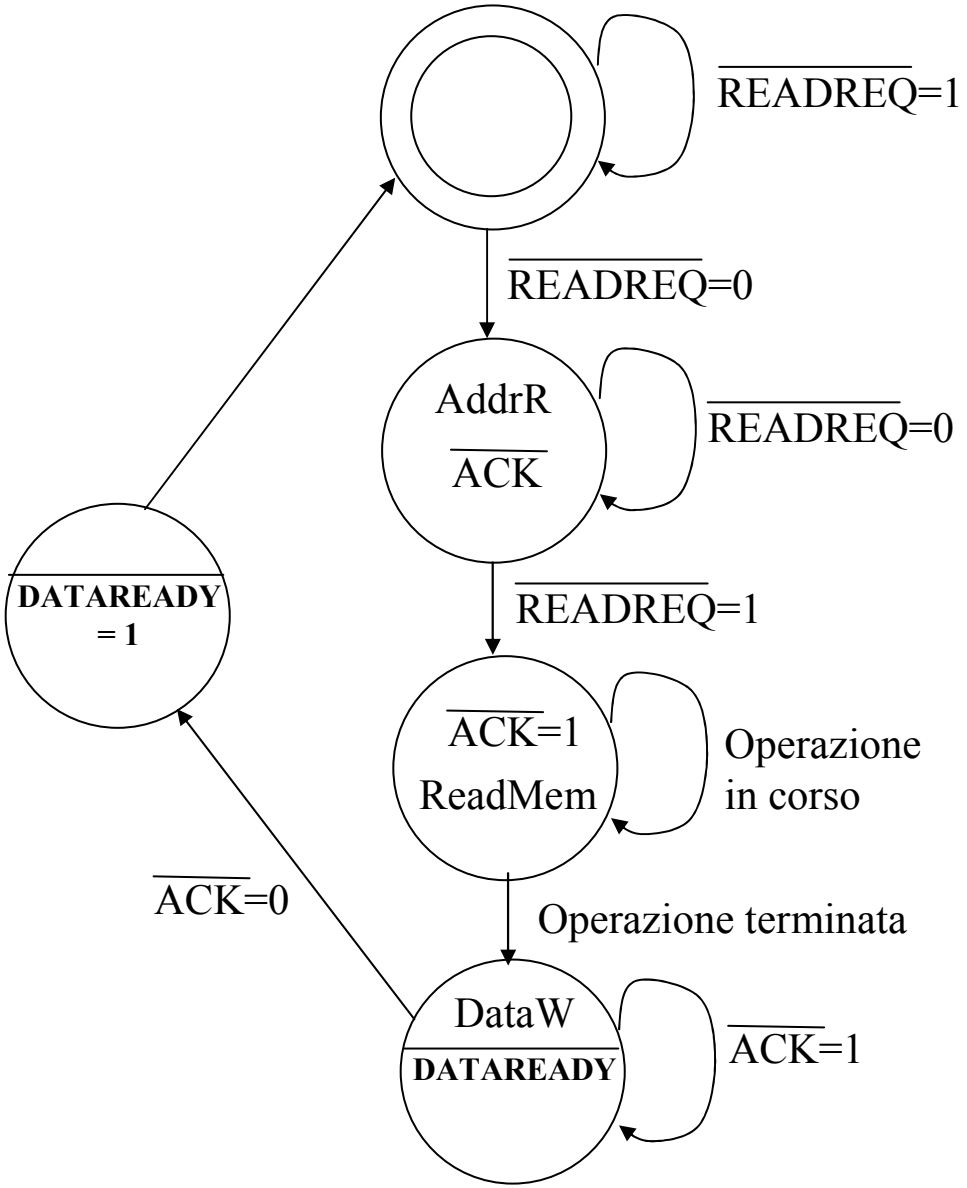
- AddrW è un segnale che abilita la scrittura dell'indirizzo nel bus;
- DataR è un segnale che abilita la lettura del dato posto nel bus

NB: negli automi che implementano il protocollo handshaking,

la transizione di stato avviene quando un ingresso [uscita dell'altro dispositivo]
assume un certo valore;

le transizioni sono asincrone, non avvengono in corrispondenza
ad un fronte di clock!

Slave (Memoria)



- AddrR è un segnale che abilita la memorizzazione dell'indirizzo presente nelle linee del bus;
- ReadMem è un segnale che abilita l'operazione di lettura in memoria
- DataW è un segnale che abilita la scrittura del dato sul bus

NB: questi due stati sono riuniti nel Patterson: entrambe le soluzioni sono ok (comportamento equivalente ai fini del protocollo)

NB: esercizi tipici sono

- Dato un diagramma temporale, sviluppare gli automi a stati finiti
- Proporre varianti al diagramma temporale
  (altro tipo di operazioni, per esempio la scrittura di una parola di memoria)

Esercizio proposto:

- sviluppare gli automi per il primo caso considerato
  (linee distinte per dati e indirizzi)

Commenti finali sul bus asincrono

Abbiamo visto che il bus asincrono presenta il vantaggio di adattarsi alle caratteristiche dei dispositivi: nessun dispositivo deve aspettare un clock di periodo fissato a priori, ma soltanto un segnale dal dispositivo con cui sta facendo la transazione.

⇨ Se una coppia è lenta, non influisce sulle coppie più veloci!

Tuttavia, la maggior parte dei bus sono sincroni! Ciò si deve principalmente a:

- facilità costruttiva (assenza di feedback – relazioni causa effetto tra segnali)
- il protocollo di handshaking comporta comunque un costo aggiuntivo in termini di tempo
- investimenti nella tecnologia basata su bus sincroni

Metodo di arbitraggio

- Se un sistema ha solo un master (il processore) allora non c'è necessità di arbitraggio: l'accesso al bus è sempre garantito al processore che "pilota" lo slave con cui correntemente effettua la transazione, che ovviamente avviene senza conflitto (secondo il protocollo specificato dal bus)
- Problema di questo approccio: le comunicazioni passano tutte per il processore.

  P.es. lettura di un blocco da disco:

  - il processore (master) legge una parola dal disco (slave)
  - il processore (master) scrive la parola in memoria (slave)
  - e così per ogni parola del blocco

⇒ Introduzione di più dispositivi attivi in grado di agire da master (DMA)

⇒ Possibile conflitto per la risorsa bus: più dispositivi possono in uno stesso momento fare richiesta di accesso al bus

⇒ Necessità di un meccanismo di arbitraggio: risolve i conflitti decidendo chi ha accesso al bus, evitando accessi contemporanei

Parametri per giudicare un meccanismo di arbitraggio

Trade-off

- Priorità:
  possibilità di suddividere i dispositivi in categorie con priorità diversa, accordando il bus ai dispositivi con priorità più alta
- Equità (fairness):
  garantire a tutti i dispositivi la possibilità di accedere al bus
- Latenza:
  tempo dedicato all'arbitraggio più basso possibile (e possibilmente sovrapposto con il tempo di trasferimento sul bus)
- Complessità di realizzazione e costo

Due categorie di meccanismi di arbitraggio

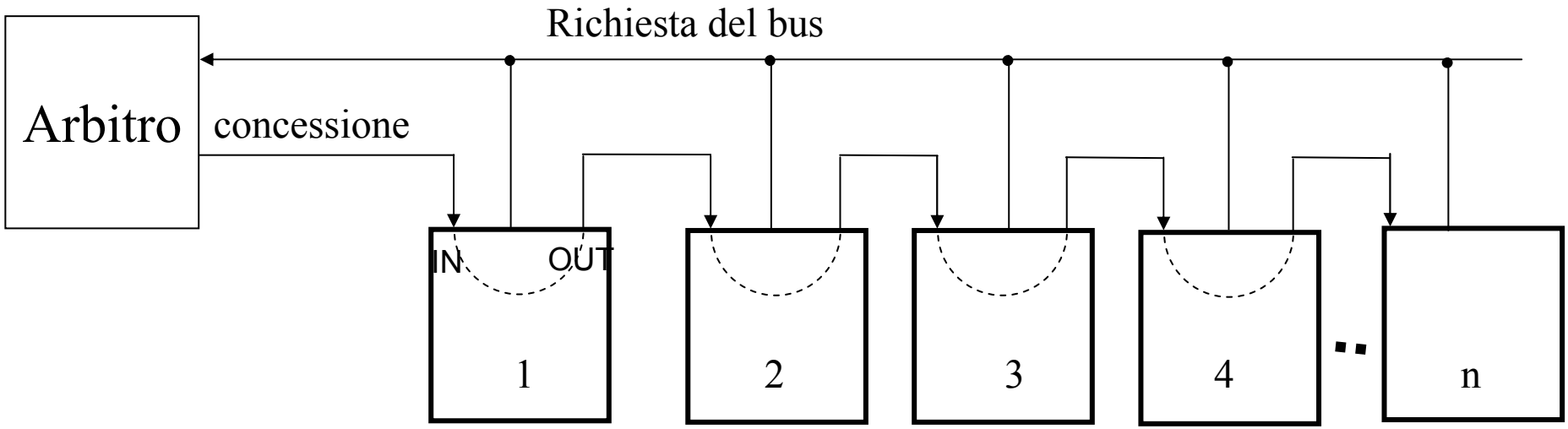
- Centralizzato:
  dispositivo hardware che riceve tutte le richieste e decide chi ha l'accesso
- Distribuito:
  dispositivi collegati al bus, esiste un protocollo per cui si scambiano informazioni e ciascuno decide se è il suo turno

Vediamo:

- Meccanismo centralizzato usando daisy chain
- Meccanismo centralizzato usando livelli + daisy chain
- Meccanismo completamente centralizzato parallelo
- Meccanismo decentralizzato

(e cenni a possibili varianti)

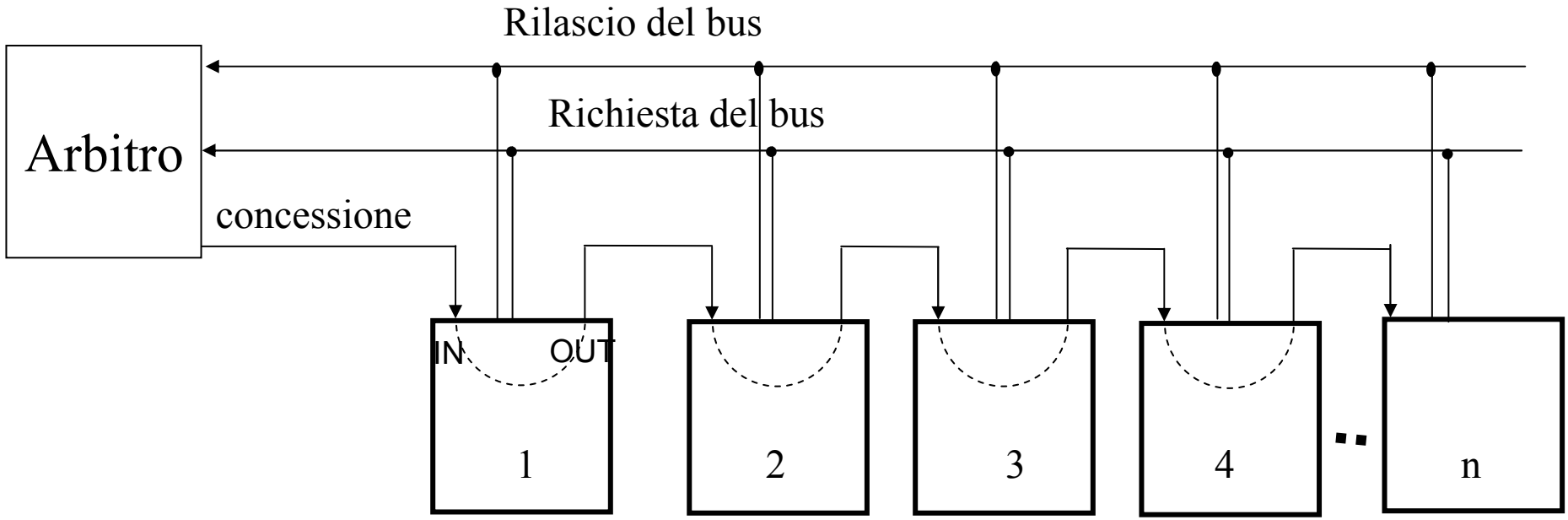
Meccanismo centralizzato usando daisy chain (già visto a proposito di interrupt)



Idea: concessione bloccata da un dispositivo più vicino all'arbitro

Se si vuole garantire che dispositivo a più alta priorità non interrompa l'accesso, occorre rendere dispositivi sensibili ai fronti…

Per effettuare arbitraggio durante il trasferimento per risparmiare tempo, si può usare una linea aggiuntiva che permette di “tenere occupato” il bus



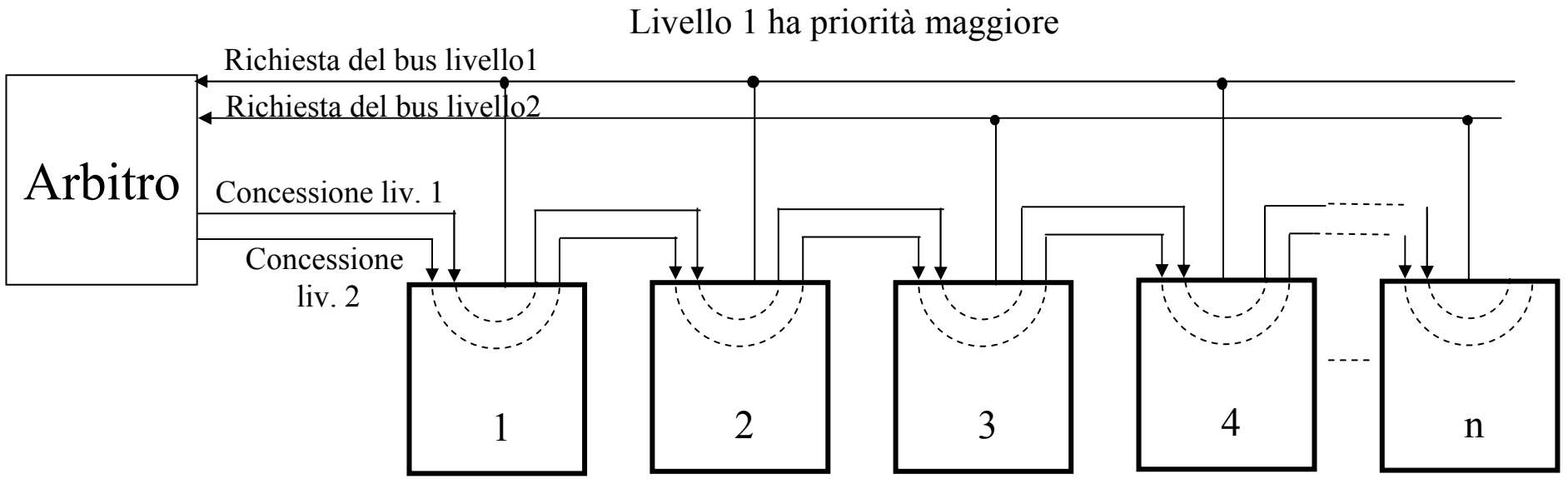
- L’arbitraggio si svolge già mentre un dispositivo ha accesso al bus, ma il dispositivo che ha la concessione del bus (e vi sta accedendo) non rilascia il bus finchè non ha finito.
- Appena rilasciato il bus, l’accesso avviene immediatamente da parte del dispositivo che lo ha conquistato

Valutazione meccanismo con daisy chain

- priorità: legata alla posizione fisica (vince il più vicino)
- equità: i dispositivi a bassa priorità possono non essere mai serviti se dispositivi a più alta priorità continuano a richiedere il bus
- latenza: il tempo dedicato all'arbitraggio (request-grant) dipende dalla posizione del dispositivo (tempo di propagazione segnali)
  $\Rightarrow$ limitazione velocità in funzione della lunghezza
- complessità & costo: semplice da realizzare, di basso costo

Meccanismo centralizzato usando daisy chain + livelli

(già visto a proposito di interrupt)

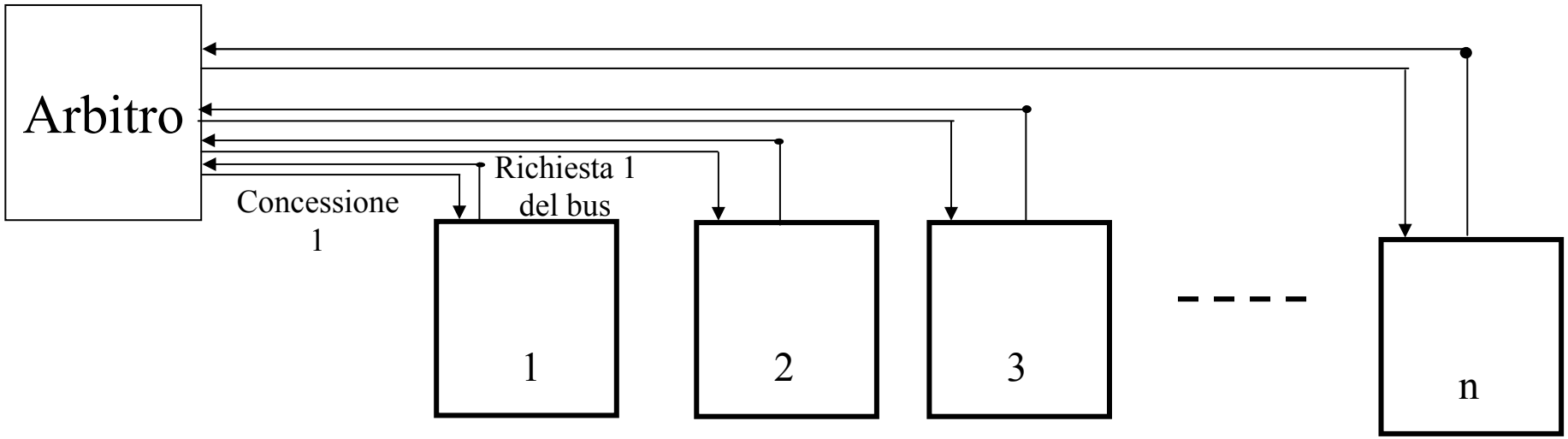


In questo caso la priorità è, nell'ordine: 1, 2, 4, 3, n

NB: una linea di concessione potrebbe attraversare solo i dispositivi collegati alla relativa linea di richiesta, ma è più semplice realizzare il collegamento come in figura [dal punto di vista logico, è come avere linee separate]

Esercizio possibile (banale): dato lo schema ordinare i dispositivi secondo la priorità

Meccanismo centralizzato parallelo



- I dispositivi richiedono il bus, ciascuno con la propria linea
- L'arbitro centralizzato sceglie uno dei richiedenti, concedendogli il bus mediante la relativa linea di concessione

VALUTAZIONE

Vantaggio:
- può utilizzare diversi algoritmi per la concessione del bus (es. priorità, round robin, ecc.) garantendo un buon compromesso tra priorità ed equità

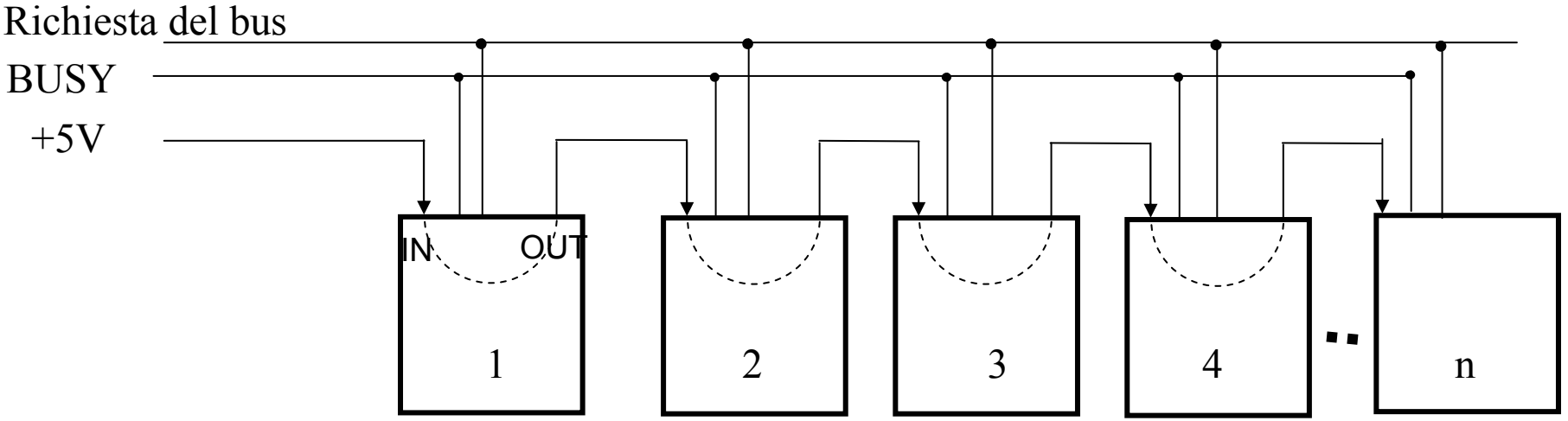
Svantaggio:
- più costoso rispetto a daisy chain
- può costituire il collo di bottiglia

NB: Il bus PCI utilizza un arbitraggio centralizzato:

ogni dispositivo ha una linea di richiesta e di concessione.

Il protocollo non specifica l'algoritmo secondo cui l'arbitro decide!

Un Meccanismo decentralizzato



Il pretendente master controlla IN e BUSY (Occupato):

se IN non asserito: bus occupato da dispositivo a maggior priorità:
il dispositivo non può diventare master, disasserisce OUT [cfr. daisy chain]

se IN asserito e BUSY asserito: bus occupato da unità con minor priorità.
Disasserisce OUT e questo provocherà a valle il disasserimento di IN-OUT

se IN asserito e BUSY non asserito (bus libero):
asserisce BUSY e OUT , e inizia il trasferimento.

E' possibile vedere che alla fine si arriva ad uno stato stabile in cui un dispositivo accede al bus (quello che ha IN asserito e OUT non asserito)

Il meccanismo precedente è molto simile a daisy chain:

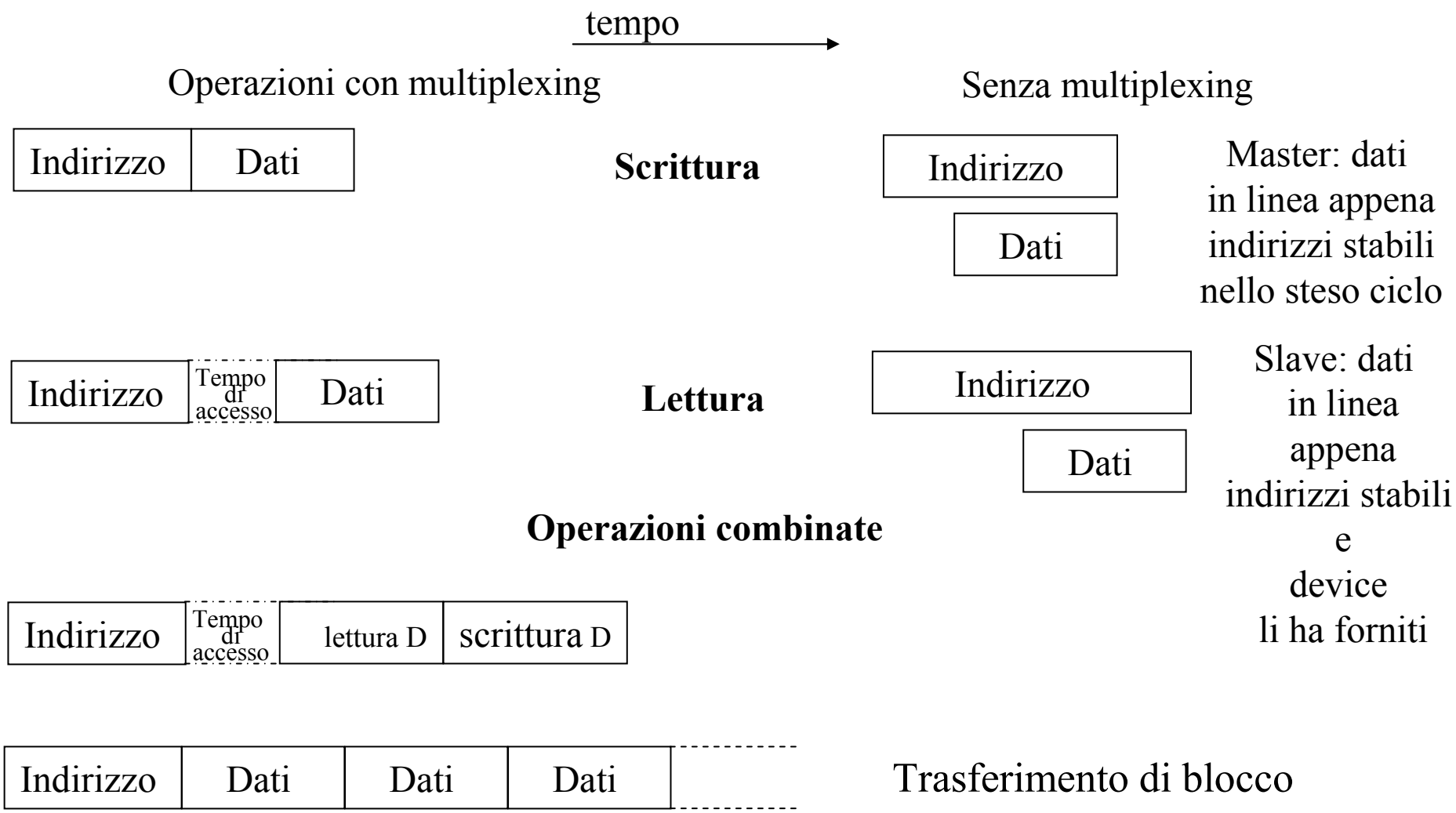
- non equo (il più vicino vince sempre)
- ma:
  senza arbitro: economico, più veloce, evita fallimenti dell'arbitro

Esistono anche altri meccanismi decentralizzati:

- Es: ciascun dispositivo scrive sul bus un codice; dalla combinazione risultante, letta da ciascun dispositivo, ciascuno è in grado di sapere se è il richiedente con priorità più elevata
- Es: arbitraggio distribuito con rilevamento delle collisioni (vedi rete Ethernet)

Tipi di operazioni dati

Oltre ai cicli di bus "ordinari" (trasferimento di parola) il protocollo può prevedere diverse altre possibilità



Ulteriori ritardi per arbitraggi

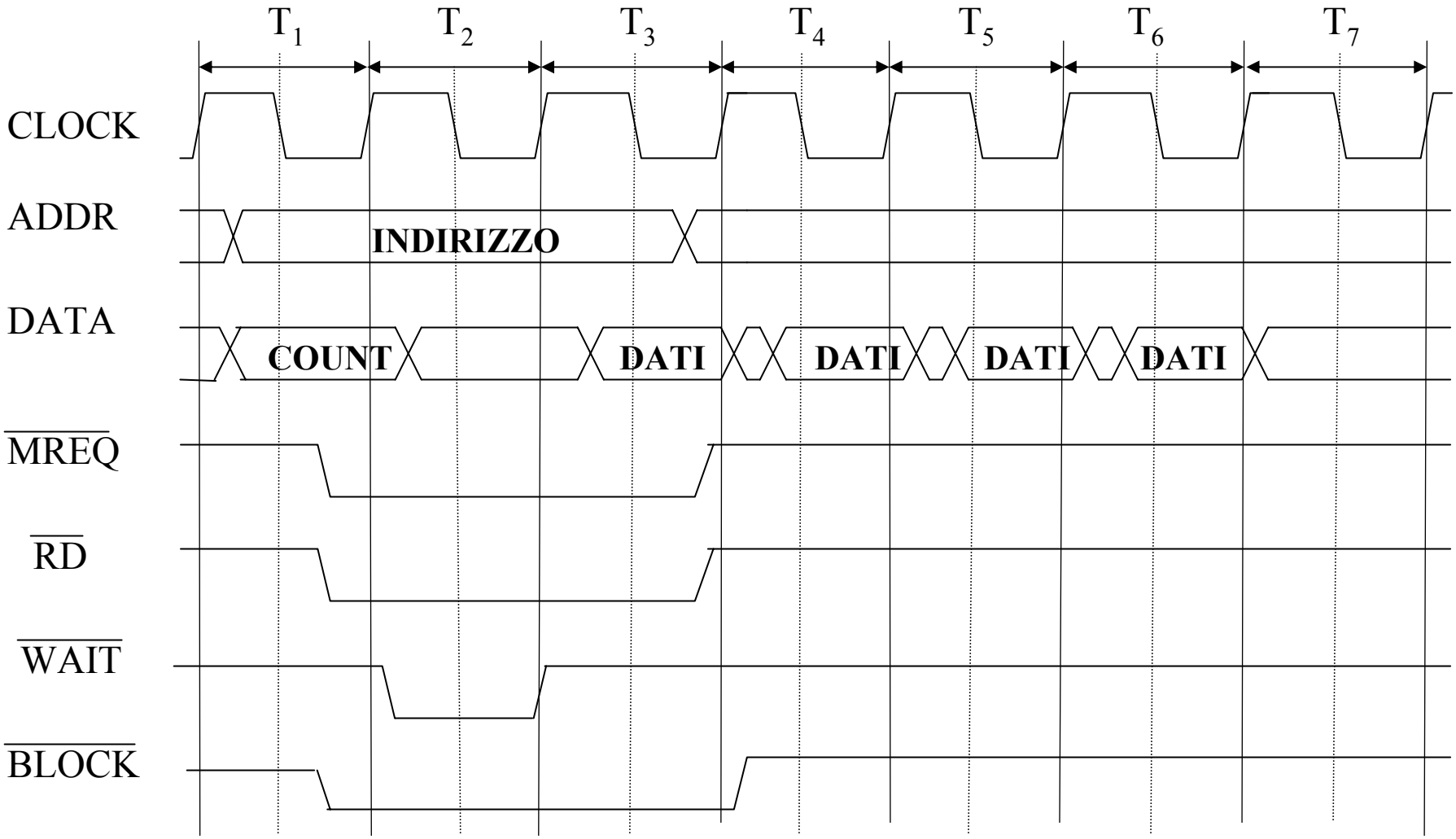
Vediamo un tipo di operazione

Trasferimento di un blocco di parole su bus sincrono

- Esempio: esiste la cache e c'è la necessità di prelevare un intero blocco di parole
- Il trasferimento riguarda 1 blocco piuttosto che una singola parola
- Quando inizia una lettura (o scrittura) del blocco il master del bus comunica allo slave del bus il numero di parole che devono essere trasferite
- Questo numero può essere ad esempio posto sulla linea dei dati
- Nel caso sincrono, lo slave trasferirà (o riceverà) una parola durante ogni ciclo di clock

Bus sincrono: lettura di un blocco di 4 parole

[impiega 6 cicli invece di 12]



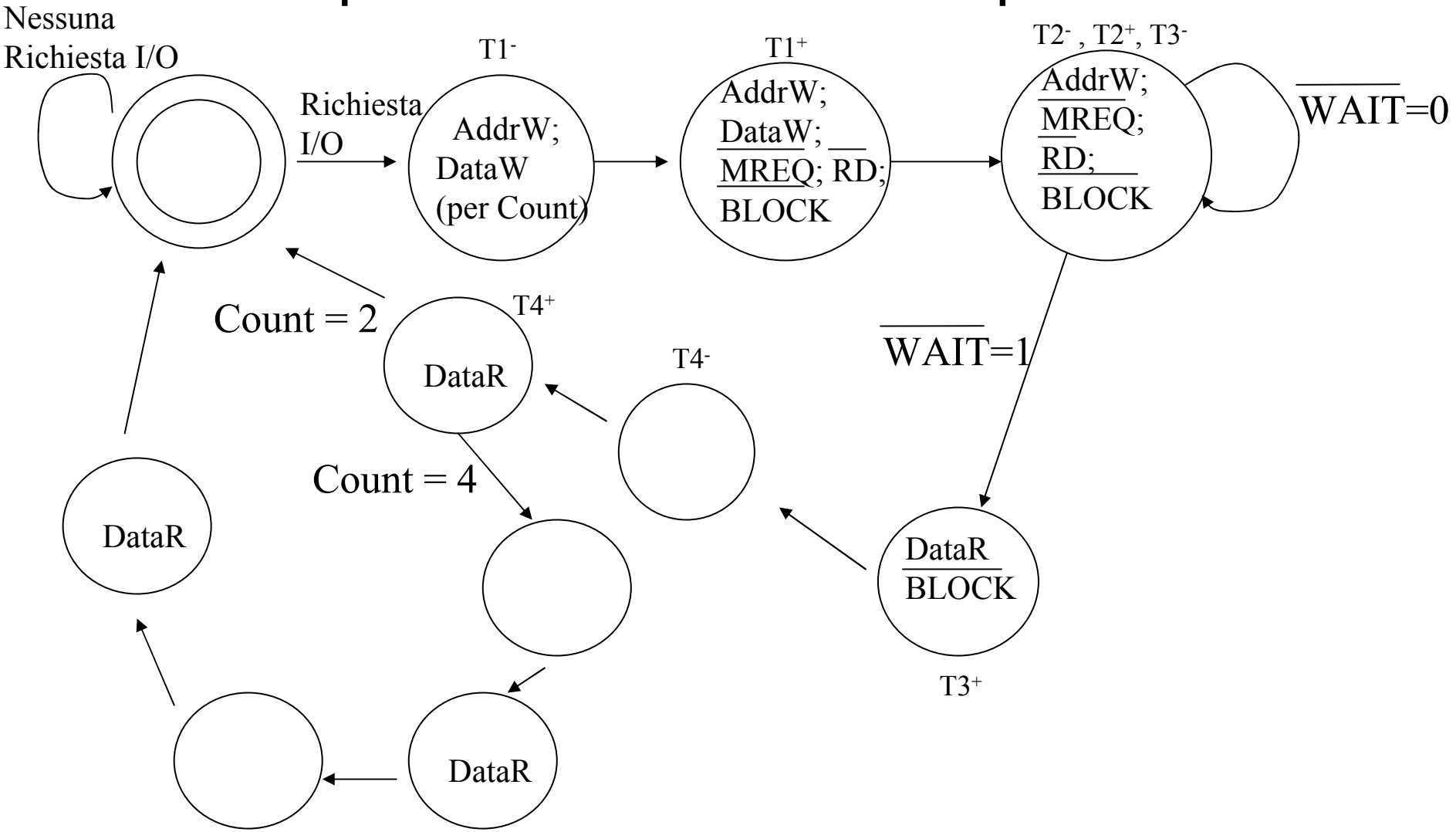
Note sul trasferimento di blocchi

- Contemporaneamente all'indirizzo viene inserito il dato "count" (numero di parole) sulle linee di dati
- Così lo slave, invece di ritornare una parola, ritorna una parola ad ogni ciclo di clock finchè non viene raggiunto il valore di count
- Il segnale BLOCK viene asserito per indicare che viene richiesto un trasferimento di blocco

ESERCIZIO TIPICO: DIAGRAMMA A STATI (P.ES. DEL MASTER)

Esempio di macchina a stati finiti (master)

per trasferimento blocchi di 2 o 4 parole



TEMA D'ESAME DEL 29 GIUGNO 2005

E' dato un bus asincrono che collega un processore P e diversi dispositivi, tra cui la memoria. Il bus è costituito da *n* linee dati, utilizzate per la trasmissione sia dei dati che degli indirizzi (bus multiplexato) e dalle linee di controllo seguenti:

$\overline{\text{READREQ}}$: utilizzato dal processore per segnalare una richiesta di lettura.
$\overline{\text{WRITEREQ}}$: utilizzato dal processore per segnalare una richiesta di scrittura.
$\overline{\text{DATAREADY}}$: utilizzato dal processore o dal dispositivo per segnalare di aver posto un dato sulle linee dato.
$\overline{\text{ACK}}$: segnale di acknowledgement usato dal processore o dal dispositivo.

La figura riporta l'evoluzione di una operazione di trasferimento da dispositivo a P (lettura)

Si chiede di:

a) Illustrare le relazioni tra i segnali del diagramma precedente,
con riferimento al protocollo di handshaking ed alle relazioni di causa-effetto
che intercorrono tra i segnali stessi.

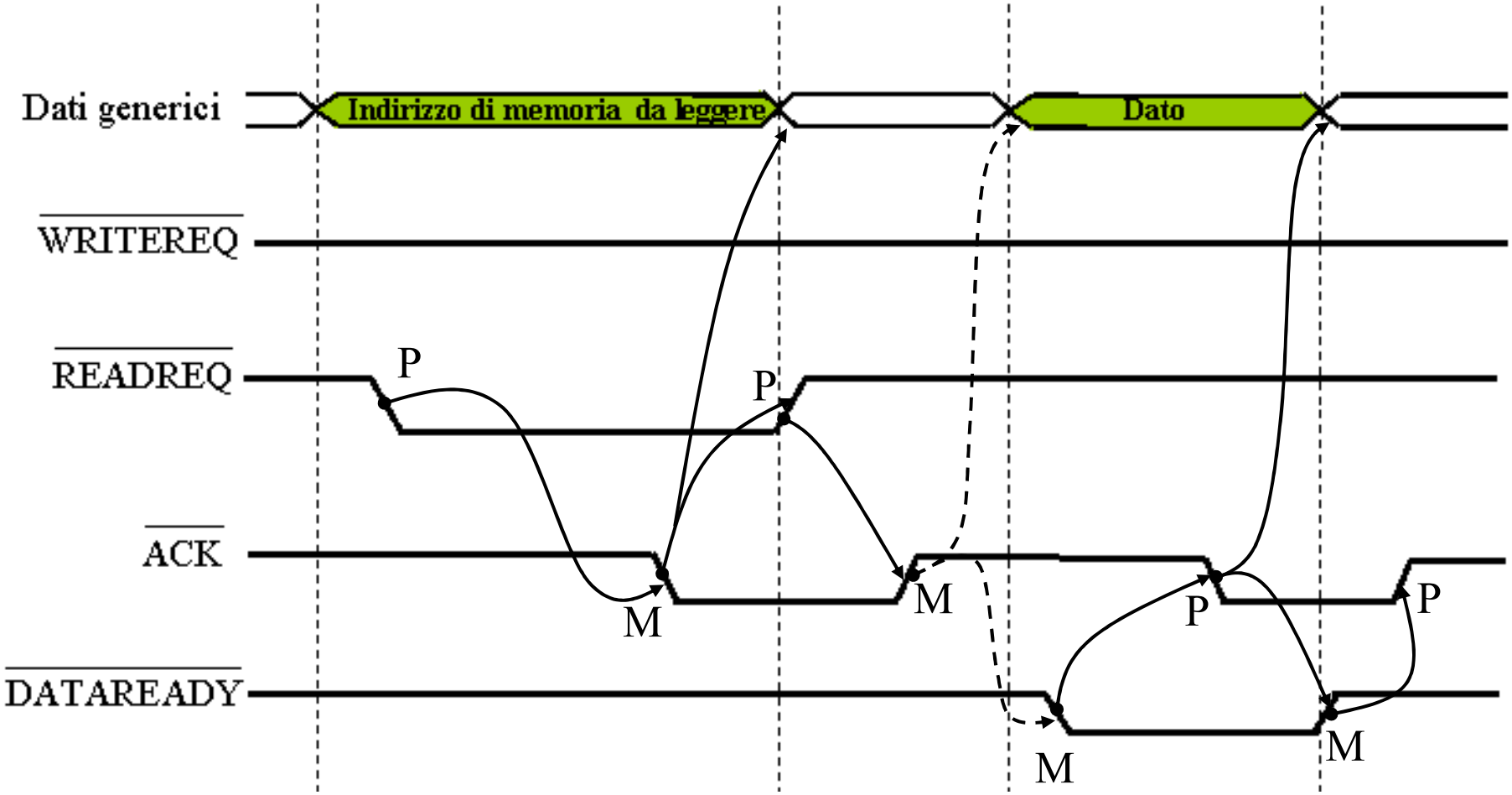
b) Mostrare in un diagramma temporale come può avvenire un'operazione di scrittura
(dal processore P al dispositivo),
illustrando brevemente il significato dei segnali.

c) Specificare la macchina a stati finiti che controlla l'esecuzione,
nel dispositivo (slave), del protocollo di handshaking in scrittura
di cui al punto b).

[8]

Risoluzione punto a

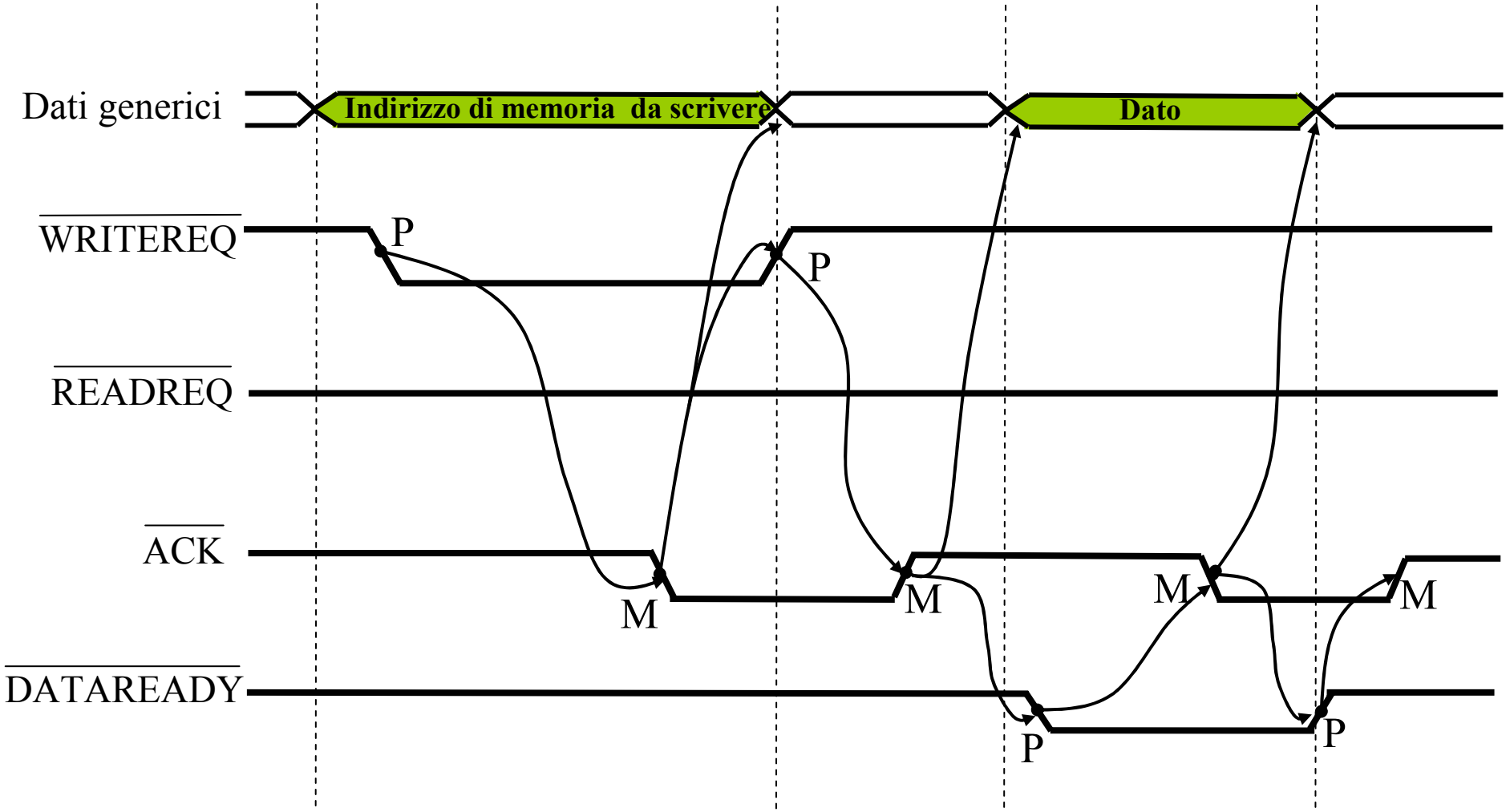
P: processore
M: slave (memoria)



+ spiegazione testuale…

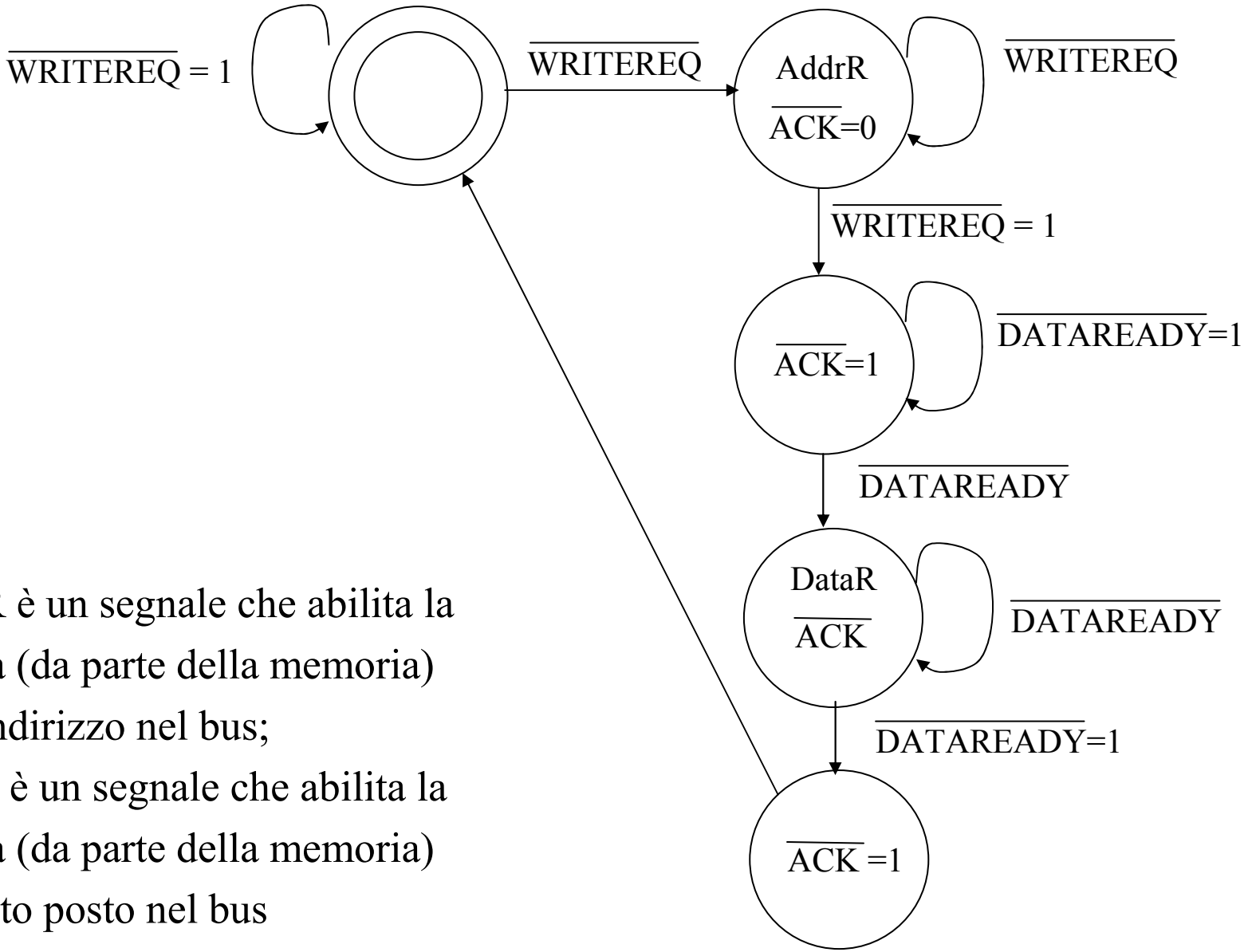
Risoluzione punto b

OPERAZIONE DI SCRITTURA



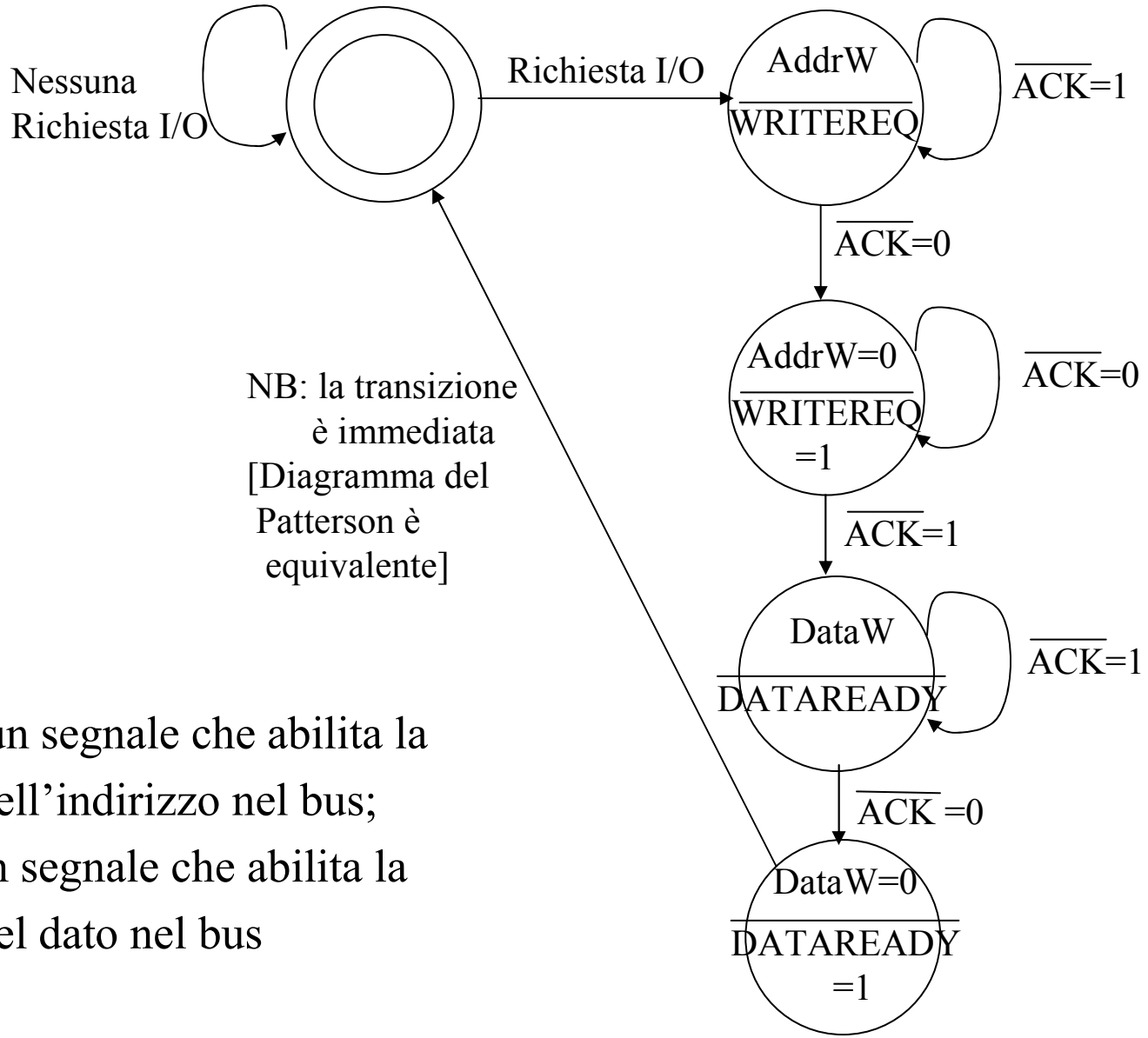
Risoluzione punto c

DIAGRAMMA SLAVE (operaz. di scrittura)



- AddrR è un segnale che abilita la lettura (da parte della memoria) dell'indirizzo nel bus;
- DataR è un segnale che abilita la lettura (da parte della memoria) del dato posto nel bus

VEDIAMO PER ESERCIZIO IL DIAGRAMMA DEL MASTER (P)



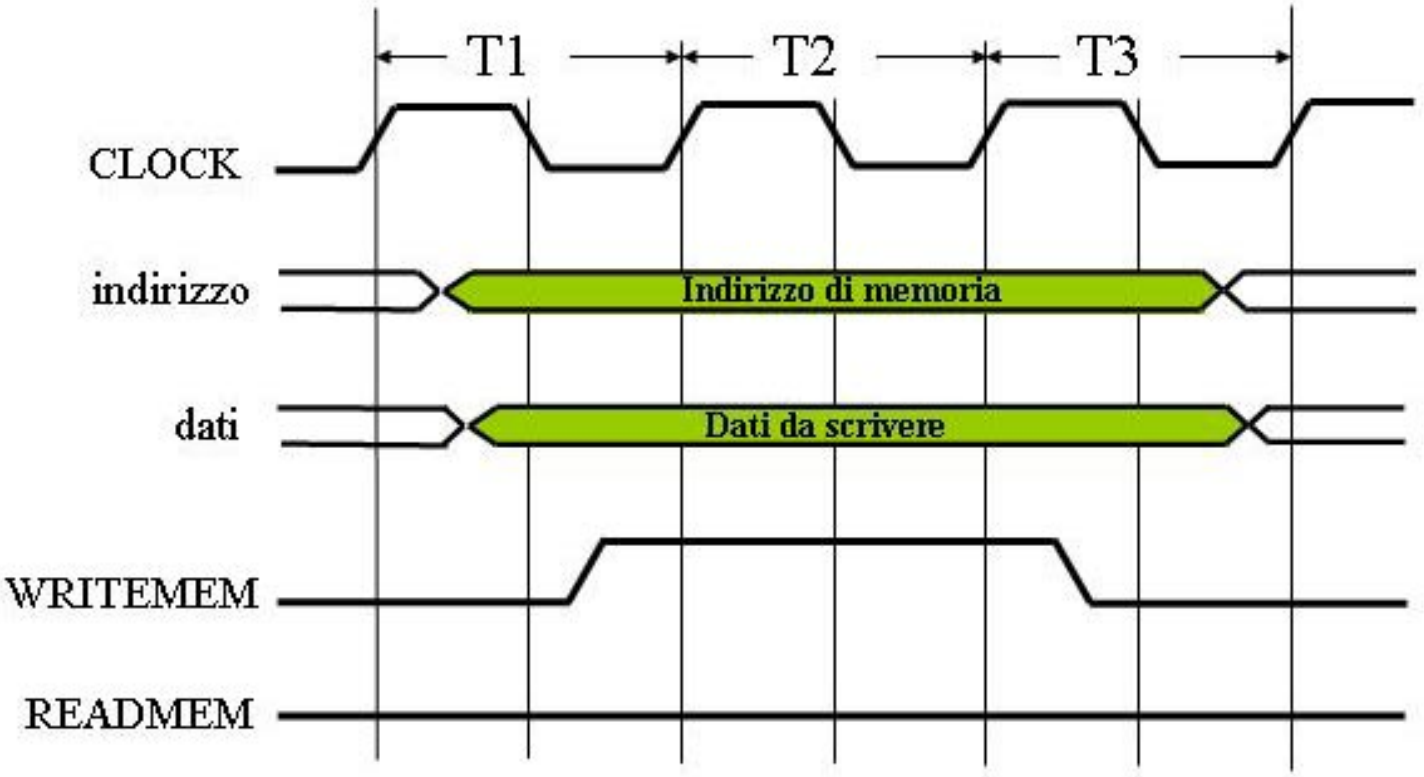
- AddrW è un segnale che abilita la scrittura dell'indirizzo nel bus;
- DataR è un segnale che abilita la scrittura del dato nel bus

ESERCIZIO 8 del TEMA D'ESAME 22 settembre 2005

E' dato un bus sincrono che collega un processore alla memoria.
Oltre alla linea per il segnale di clock, il bus è costituito da linee distinte per dati e indirizzi, nonché dalle linee di controllo seguenti
(entrambi i segnali di controllo sono attivi a livello alto):

- WRITEMEM: utilizzato da processore per segnalare richiesta di scrittura in memoria.
- READMEM: utilizzato al processore per segnalare richiesta di lettura dalla memoria.

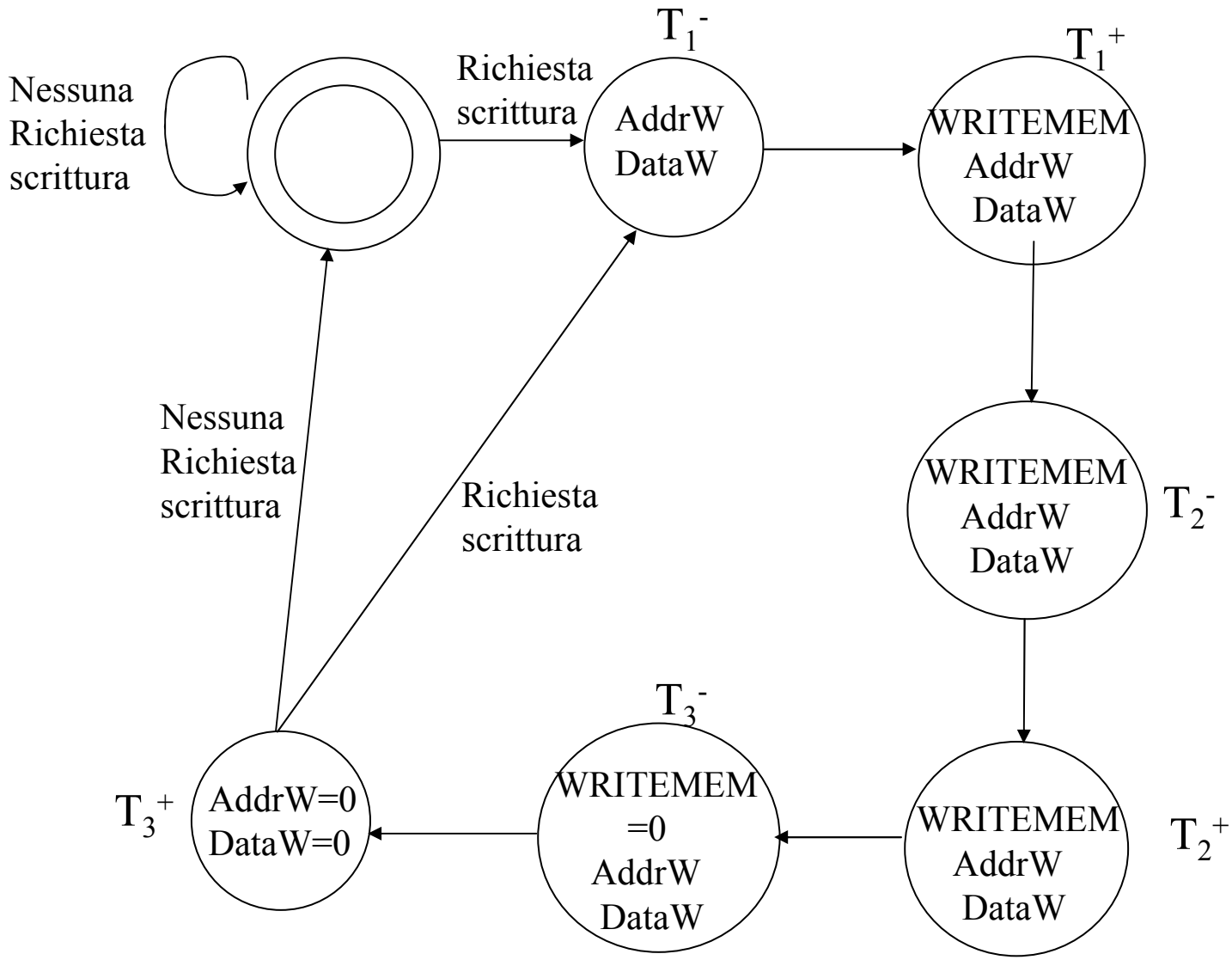
Evoluzione temporale di un'operazione di scrittura in memoria



Si noti che non c'è alcun segnale di ritorno dalla memoria al processore:
si assume che la memoria richieda un determinato numero di cicli di clock
per completare l'operazione di scrittura, esattamente come avviene nel diagramma.

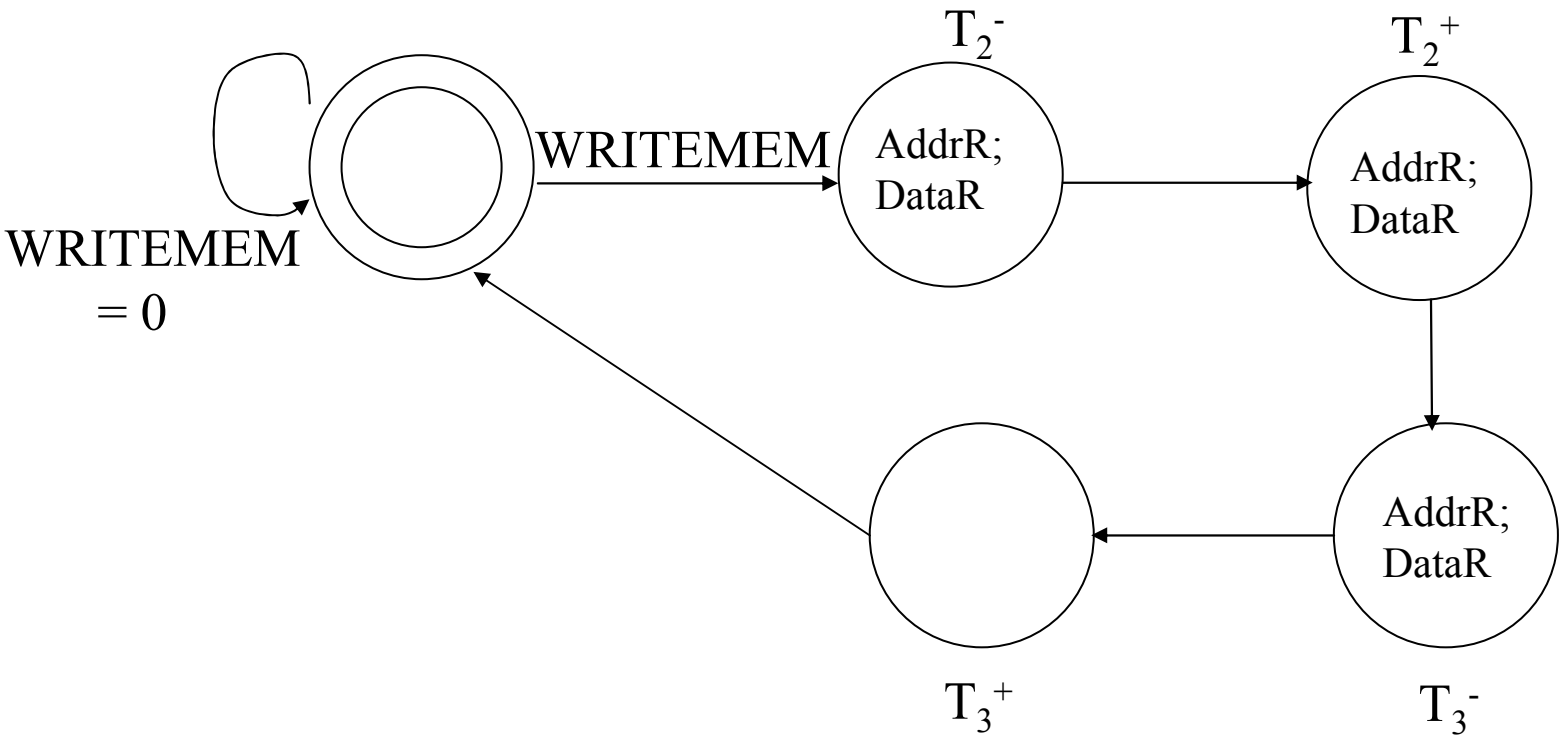
Si specifichino le due macchine a stati finiti che controllano l'operazione
di scrittura indicata nel diagramma, rispettivamente
per il processore e per la memoria. [6]

PROCESSORE (SCRITTURA)



Nota: "AddrW" e "DataW" sono segnali di controllo che abilitano rispettivamente l'invio indirizzo e del dato [da parte del master]

MEMORIA (SCRITTURA)



Nota: "AddrR" e "DataR" sono segnali di controllo interni alla memoria che abilitano l'acquisizione dal bus rispettivamente rispettivamente dell'indirizzo e del dato

Prestazioni dei sistemi di I/O

- Le prestazioni dei sistemi di I/O dipendono dalla velocità di trasferimento dei dati
- Questa velocità è espressa in

  MB/sec (***ampiezza di banda***)
- Nei sistemi di I/O i MB sono misurati usando la base 10 (1 MB = 10^6 byte)
- Ricordare che quando si parla di memoria 1 MB = 2^{20} byte
- Ad esempio, avendo un bus a 100 MB/s, il tempo necessario per trasferire $100\ MB_2$ (100 MB di memoria) è:

  (100*1.048.576 bytes) / (100 *10^6 bytes/s) = 1,048576 s

  [differenza spesso trascurabile]

Promemoria

Prefissi

10^{12}	T	Tera	2^{40}	1.099,521.422.776
10^{9}	G	Giga	2^{30}	1.073.741.824
10^{6}	M	Mega	2^{20}	1.048.576
10^{3}	K	Kilo	2^{10}	1024
10^{2}	h	Etto		
10^{1}	da	Deca		
10^{0}		unità	2^{0}	1
10^{-1}	d	Deci		
10^{-2}	c	Centi		
10^{-3}	m	Milli		
10^{-6}	μ	Micro		
10^{-9}	n	Nano		
10^{-12}	p	Pico		

Varie

1 byte=8bit

$2^{11} = 2048$

16 K= 2^{14}

Esercizio

- Si consideri il bus sincrono illustrato in precedenza (cfr. lucido 51), assumendo di disporre di 32 bit per le linee dati e di un clock di 200 MHz
- Si assuma di poter effettuare (secondo le modalità viste) tre tipi di trasferimento:
  - singola parola
  - blocchi di 2 parole
  - blocchi di 4 parole

  Tra un trasferimento e il successivo, è necessario 1 ciclo di clock di attesa
- Si consideri il trasferimento di 256 parole (di 32 bit):

  Si calcoli, per ciascuna delle tre modalità di trasferimento, il tempo di trasferimento e l'ampiezza di banda (in byte/s) raggiunta dal sistema memoria-bus
- Si assuma di poter disporre di una memoria arbitrariamente veloce.

  Trovare – nello stesso caso considerato – le massime prestazioni raggiungibili [in termini di tempo di trasferimento e ampiezza di banda]

Soluzione

- F_{bus} = 200 MHz $\Rightarrow$ T_{bus} = 5 ns [$1/(2*10^8) = 10/2 * 10^{-9} = 5 * 10^{-9}$]
- In ogni caso, vengono trasferiti 256*4 bytes = 1024 bytes

A questo punto, dal diagramma temporale si ricavano i cicli necessari per ciascun tipo di trasferimento, consentendo di ricavare i dati richiesti…

Trasferimento a parole:

1 parola (4 bytes) richiede 3 cicli di clock + 1 di attesa

Totale: 256* 4 = 1024 cicli

$\Rightarrow$ T_{trasf} = 1024*5 ns = 5120 ns

$\Rightarrow$ Larghezza di banda = $(1024/5120*10^{-9})$ = 200 MB/s

Trasferimento a blocchi di due parole:

E' necessario trasferire 256/2 = 128 blocchi

1 blocco (2 parole) richiede 4 cicli di clock + 1 di attesa

Totale: 128* 5 = 640 cicli

$\Rightarrow T_{trasf} = 640*5$ ns = 3200 ns

$\Rightarrow$ Larghezza di banda = $(1024/3200*10^{-9})$ = 320 MB/s

Trasferimento a blocchi di quattro parole:

E' necessario trasferire 256/4 = 64 blocchi

1 blocco (4 parole) richiede 6 cicli di clock + 1 di attesa

Totale: 64* 7 = 448 cicli

$\Rightarrow T_{trasf} = 448*5$ ns = 2240 ns

$\Rightarrow$ Larghezza di banda = $(1024/2240*10^{-9})$ = 457 MB/s

Se si dispone di una memoria arbitrariamente veloce, scompare nel diagramma temporale il ciclo di WAIT, quindi ciascun tipo di trasferimento richiede un ciclo in meno…

Si possono rifare tutti i calcoli con i nuovi numeri di cicli richiesti