CALCOLATORI
I/O

Giovanni Iacca
giovanni.iacca@unitn.it

Lezione basata su materiale preparato con i Prof. Luigi Palopoli e Marco Roveri

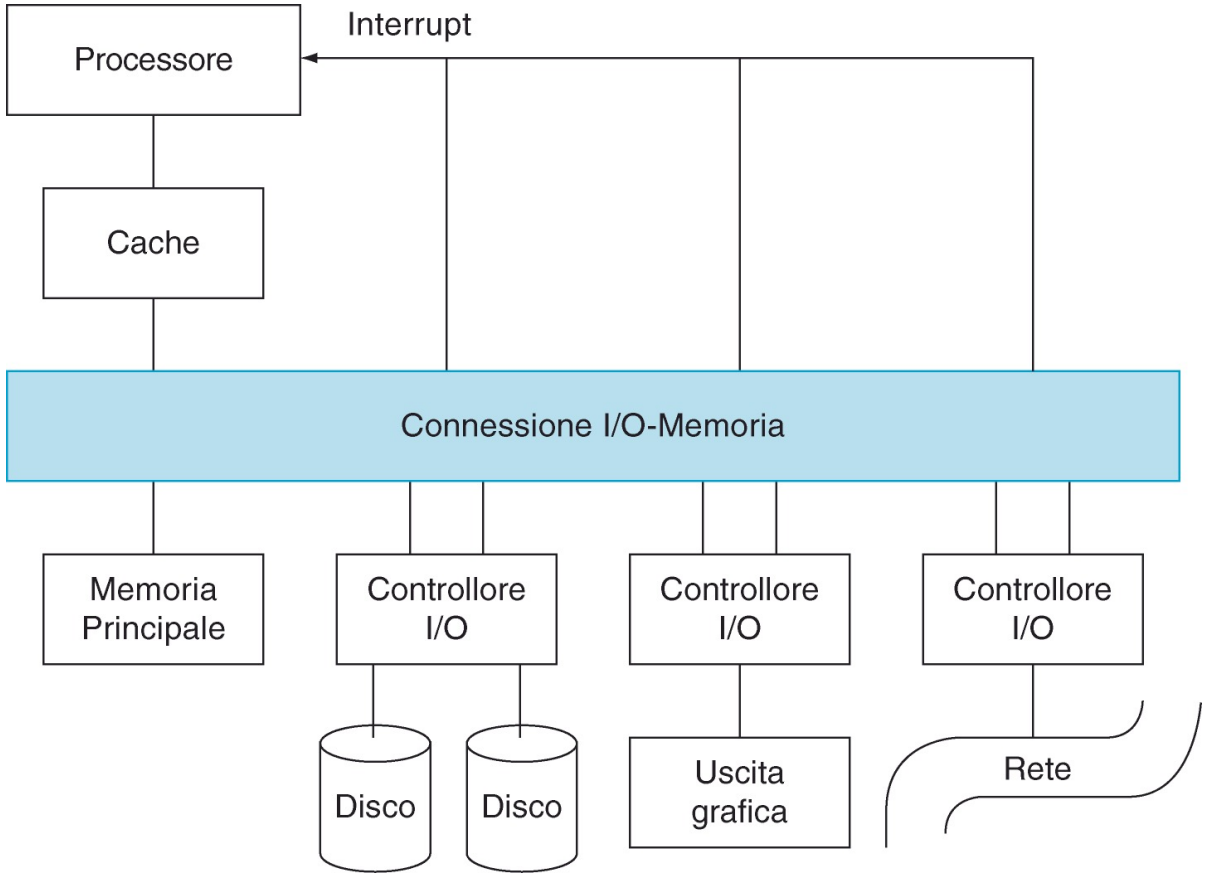
UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

La necessità di comunicare

- Un calcolatore è completamente inutile senza la possibilità di caricare/salvare dati e di comunicare con l'esterno
- I dispositivi di I/O devono essere
  - espandibili
  - eterogenei
- I dispositivi di I/O sono molto vari e la tipologia di prestazione è diversa
  - In alcuni casi interessa il tempo di accesso (la latenza) e il tempo di risposta
    - ✓ Es. dispositivi interattivi come tastiere o mouse
  - In altri casi siamo interessati al throughput
    - ✓ Caso di dischi o interfacce di rete

Un semplice schema

- I dispositivi sono collegati al processore da un dispositivo di comunicazione chiamato *bus*



Classificazione

- I dispositivi di I/O sono di vario tipo e possono essere classificati in vari modi
  - Comportamento: che operazioni posso effettuare con il dispositivo (R/W)
  - Partner: può essere un uomo o una macchina
  - Velocità di trasferimento

Esempi

Dispositivo	Comportamento	Partner	Frequenza dati (Mbit/s)
Tastiera	Input (ingresso)	Uomo	0,0001
Mouse	Input (ingresso)	Uomo	0,0038
Input vocale	Input (ingresso)	Uomo	0,2640
Input audio	Input (ingresso)	Macchina	3,0000
Scanner	Input (ingresso)	Uomo	3,2000
Output vocale	Output (uscita)	Uomo	0,2640
Output audio	Output (uscita)	Uomo	8,0000
Stampante laser	Output (uscita)	Uomo	3,2000
Display grafico	Output (uscita)	Uomo	800,0000-8000,0000
Modem via cavo	Input o output	Macchina	0,1280-6,0000
Rete/LAN	Input o output	Macchina	100,000-10000,0000
Rete/LAN wireless	Input o output	Macchina	11,0000-54,0000
Disco ottico	Memoria	Macchina	80,0000-220,0000
Nastro magnetico	Memoria	Macchina	5,0000-120,0000
Memoria flash	Memoria	Macchina	32,0000-200,0000
Disco magnetico	Memoria	Macchina	800,0000-3000,0000

Prestazioni

- A seconda del tipo di applicazione, posso essere interessato a diverse prestazioni
  - Ad esempio per un sistema di streaming mi interessa il throughput
  - Per un sistema bancario, mi può servire massimizzare il numero di file di piccole dimensioni su cui opero contemporaneamente

Connessione tra processori e periferiche

- Le connessioni avvengono tramite delle strutture di comunicazione chiamate *bus*
- Esistono due tipi di bus
  - Bus processore/memoria:
    - ✓ specializzati, corti e veloci
  - Bus I/O
    - ✓ possono essere lunghi e permettono il collegamento con periferiche eterogenee
    - ✓ tipicamente, non sono collegati alla memoria in maniera diretta ma richiedono un bus processore/memoria o un bus di sistema
- Nelle prime architetture avevamo un unico grosso bus parallelo che collegava tutto
- Per problemi di clock e frequenze ora si usano architetture di comunicazione più complesse fatte di più bus paralleli condivisi e di bus seriali punto/punto

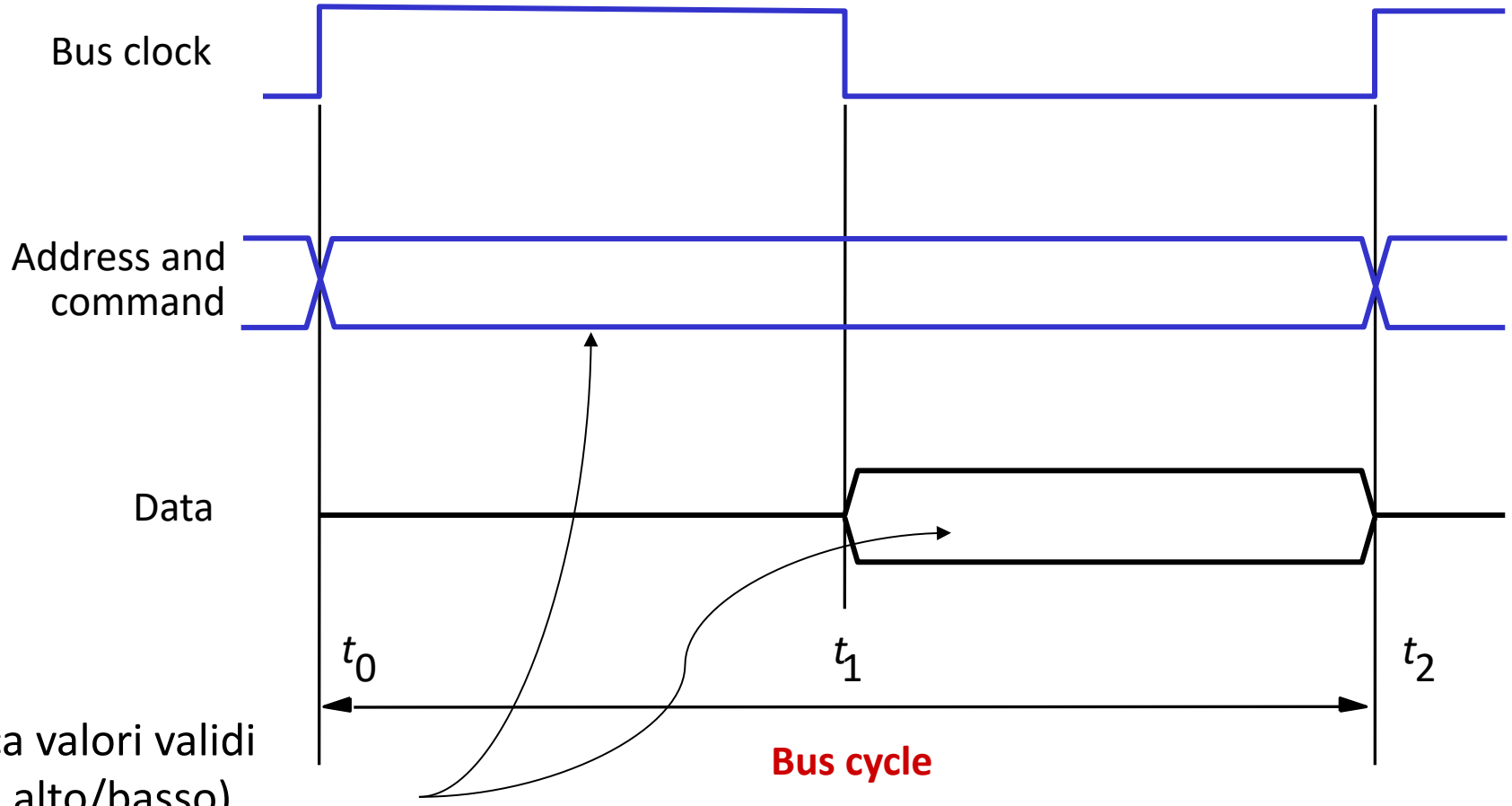
Terminologia

- Transazione di I/O
  - Invio indirizzo e spedizione o ricezione dei dati
- Input
  - Trasferimento di dati da una periferica verso la memoria dove il processore può leggerla
- Output
  - Trasferimento dalla memoria al dispositivo

Bus sincrono

- Tra le linee di controllo deve avere clock
- Le comunicazioni avvengono con un protocollo collegato al ciclo di clock
- Esempio: dato richiesto al clock n viene messo sul bus al clock n+5

Bus sincrono: funzionamento base



Indica valori validi
(1/0, alto/basso)
su una struttura parallela

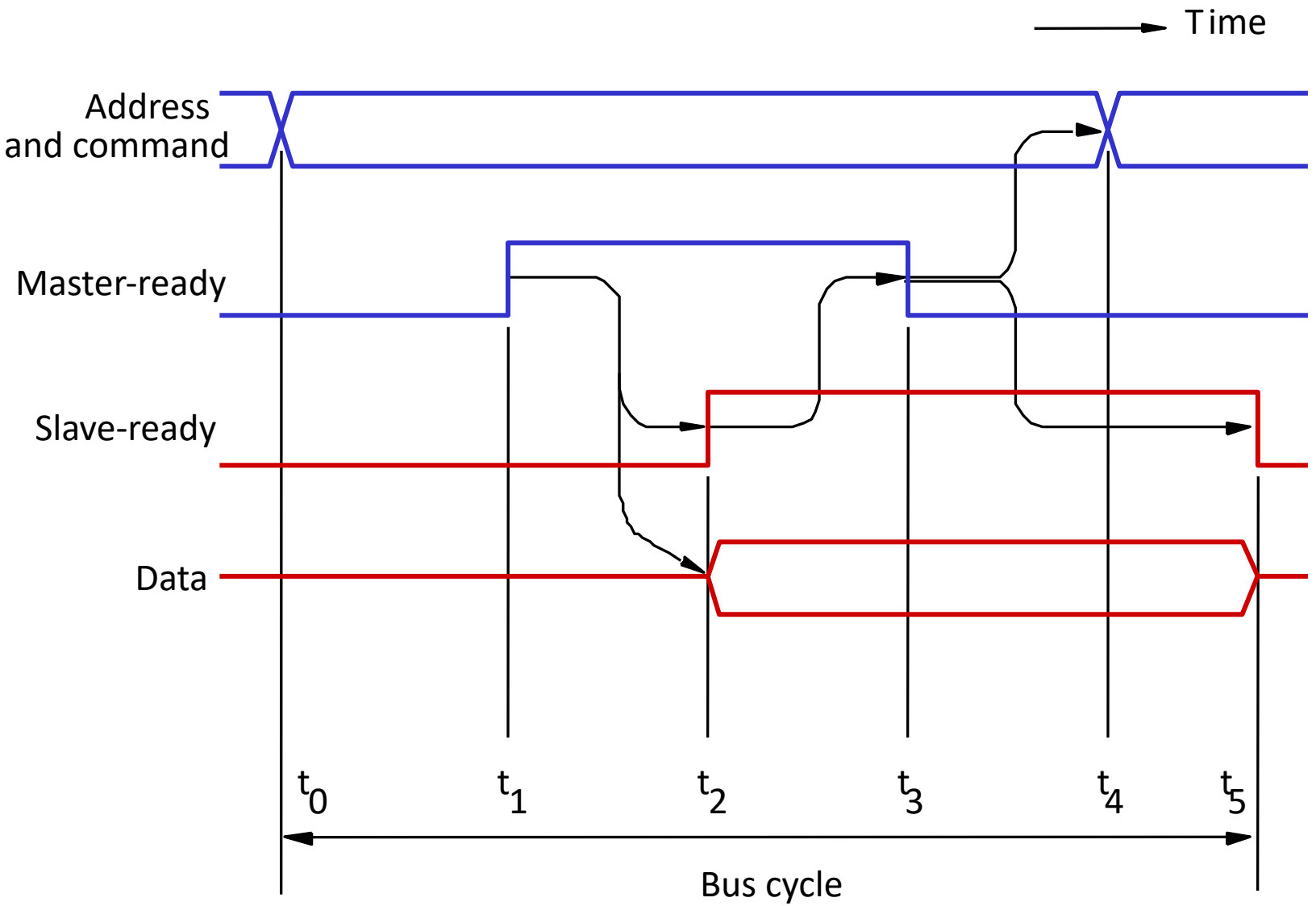
Bus sincrono

- Pro
  - Molto semplice da implementare (piccola macchina a stati finti)
  - Molto veloce (pochi segnali di controllo)
- Contro
  - Poca robustezza al *drift* del clock
  - Tutte le periferiche devono andare alla velocità del clock

Bus asincrono

- Per ovviare agli inconvenienti discussi si tende a usare interconnessioni asincrone
- In sostanza non abbiamo più un clock e tutte le transazioni sono governate da una serie di segnali di *handshake*
- Questo richiede l'introduzione di apposite linee di controllo per segnalare inizio e fine di transazioni, ma permette di collegare periferiche a velocità diversa

Ciclo di un bus asincrono



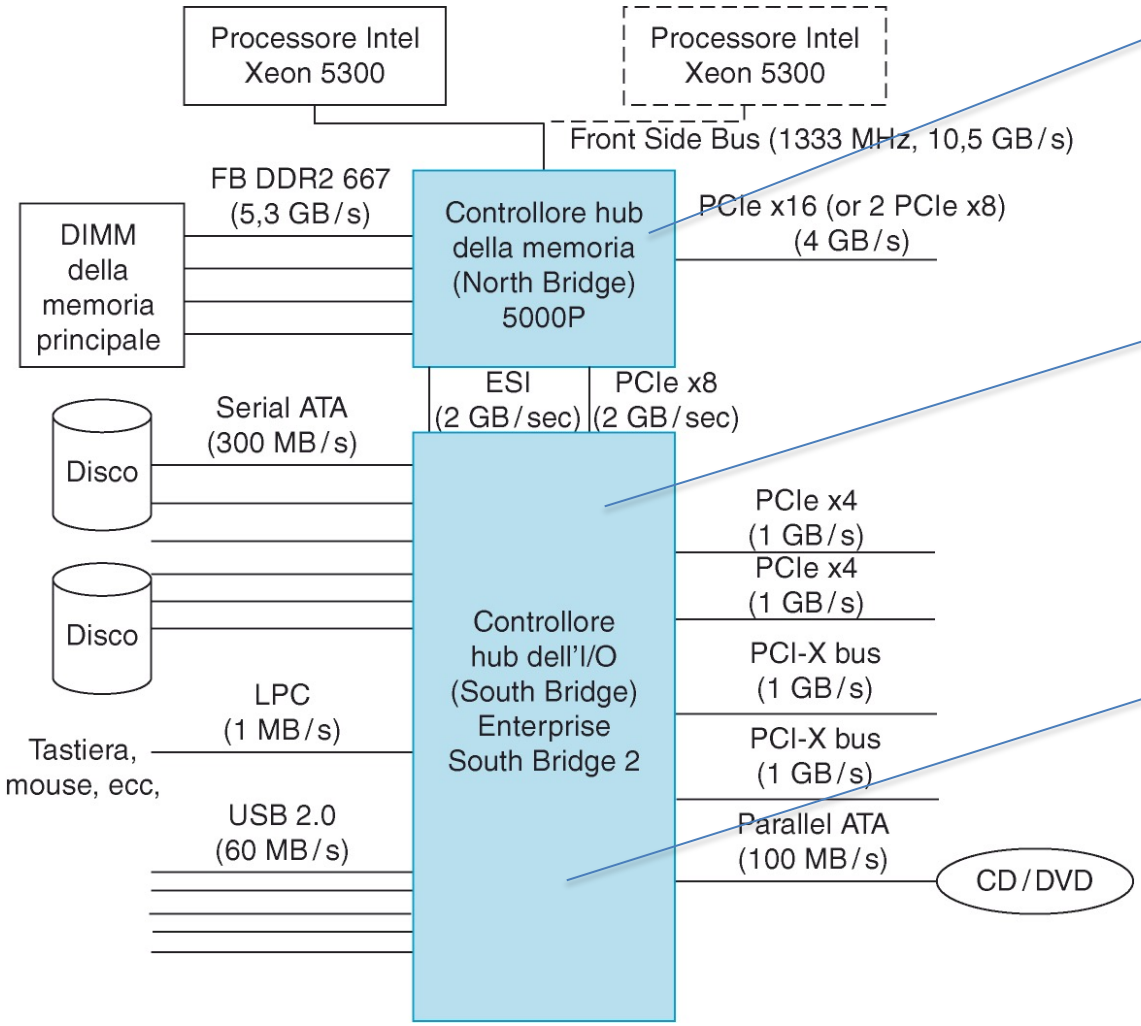
Bus asincrono

- Pro
  - Consente di essere robusto rispetto a ritardi
  - Consente di comunicare con periferiche di tipo diverso
- Contro
  - Lento nelle interazioni (diversi segnali di controllo devono circolare per riuscire a comunicare)
  - Circuiteria di gestione del protocollo complessa
- Spesso si usano tecnologie ibride (in cui c'è un segnale di clock) ma prevalentemente asincrone

Tecnologie (asincrone) attuali

Caratteristica	Firewire (1394)	USB 2.0	PCI Express	Serial ATA	Serial Attached SCSI
Utilizzo previsto	Esterno	Esterno	Interno	Interno	Esterno
Numero dispositivi per canale	63	127	1	1	4
Larghezza base dei dati (numero di segnali)	4	2	2 per linea	4	4
Larghezza di banda di picco teorica	50 MB/s (Firewire 400) o 100 MB/s (Firewire 800)	0,2 MB/s (low speed), 1,5 MB/s (full speed), o 60 MB/s (high speed)	250 MB/s per linea (1x); le schede PCIe sono disponibili in versione 1x, 2x, 4x, 8x, 16x o 32x	300 MB/s	300 MB/s
Collegamento a caldo	Sì	Sì	Dipende dalle dimensioni	Sì	Sì
Lunghezza massima del bus (piste in rame)	4,5 metri	5 metri	0,5 metri	1 metro	8 metri
Nome dello standard	IEEE 1394, 1394b	Forum degli implementatori USB	SIG PCI	SATA-IO	Comitato T10

Esempio x86



HUB per il controllo memoria

HUB per connessione ad altri bus

Trend più recente: incorporare il tutto all'interno del processore

Prospettiva del programmatore

- Rimane da capire
  - Come trasformare una richiesta di I/O in un comando per la periferica
  - Come trasferire i dati
  - Qual è il ruolo del sistema operativo?
- Riguardo al SO occorre osservare
  - Programmi che condividono il processore condividono anche il sistema di I/O
  - I trasferimenti dati vengono spesso effettuati usando interrupt, che hanno un impatto sulle funzionalità del SO
    - ✓ Quindi devono essere eseguiti in una particolare modalità del processore (supervisor) cui solo il codice del *kernel* può accedere
  - Il controllo di operazioni I/O spesso si interseca con problematiche di concorrenza

Funzionalità richieste al SO

- Garantire che un dato utente abbia accesso ai dispositivi di I/O cui ha diritto (permessi) di accedere
- Fornire comandi di alto livello per gestire le operazioni di basso livello
- Gestire le interruzioni generate dai dispositivi di I/O (in maniera simile a quanto avviene con le eccezioni generate nei programmi)
- Ripartire l'accesso a ciascun dispositivo in maniera equa tra i vari programmi che lo richiedono

Requisiti

- Per implementare le funzionalità appena discusse occorre
  - Rendere possibile al SO di inviare comandi alle periferiche
  - Rendere possibile ai dispositivi notificare la corretta esecuzione di un'operazione
  - Consentire trasferimenti diretti di dati tra dispositivo e memoria

Come impartire i comandi ai dispositivi

- Questo si fa fornendo sulle relative linee di bus alcune «parole» di controllo
- Può essere fatto in due modi:
  - Scrivendo/leggendo in particolari locazioni di memoria (*memory mapped I/O*)
  - Tramite alcune istruzioni speciali (dedicate all'I/O)

Esempio

- Scrivendo una particolare parola in una locazione di memoria associata al dispositivo
  - Il sistema di memoria ignora la scrittura
  - Il controllore di I/O intercetta l'indirizzo particolare e trasmette il dato al dispositivo sotto forma di comando
- Queste particolari locazioni di memoria NON sono accessibili ai programmi utente ma solo al sistema operativo (quindi occorre una chiamata di sistema che faccia commutare il processore in modalità supervisore)
- Il dispositivo stesso può usare queste locazioni per trasmettere dati o pre-segnalare il suo stato
- Ad esempio posso chiedere la stampa di un carattere a terminale, e a stampa finita un particolare bit di un registro di stato mappato in memoria verrà commutato (a 1 per indicare la corretta stampa)

Come trasmettere/ricevere i dati

- La modalità più semplice per trasferire i dati è la cosiddetta attesa attiva (polling)
- In sostanza si manda un comando di lettura/scrittura alla periferica e poi si fa un ciclo di attesa testando il bit di stato per vedere quando il dato è pronto

Esempio

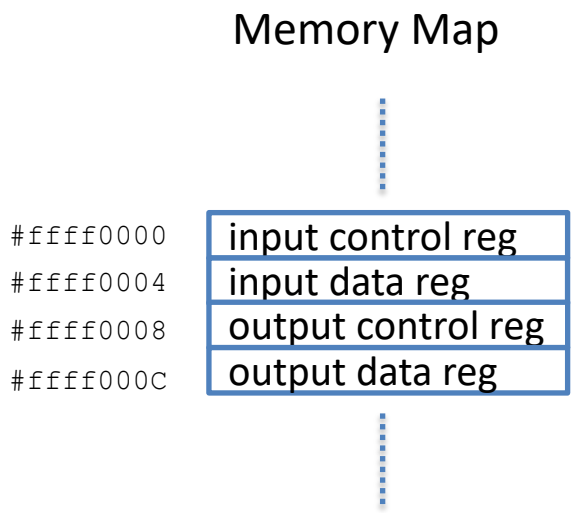
- Input: lettura dalla tastiera in x7

```
            lui  x5, 0xffff #ffff0000
Waitloop:   lw   x6, 0(x5) #control
            andi x6, x6,0x0001
            beq  x6, x0, Waitloop
            lw   x7, 4(x5) #data
```

- Output: stampa del dato da x7

```
            lui  x5, 0xffff #ffff0000
Waitloop:   lw   x6, 8(x5) #control
            andi x6,x6,0x0001
            beq  x6,x0, Waitloop
            sw   x7, 12(x5) #data
```

- Questo ciclo di attesa è chiamato *polling*



Costo del polling

- Consideriamo un processore a 500Mhz e supponiamo che occorrano 400 cicli di clock per un'operazione di polling. Qual è il costo percentuale?
  - Esempio 1: Mouse. Per non perdere movimenti da parte dell'utente occorre acquisire il dato 30 volte al secondo.
  - Esempio 2: Hard disk. I dati vengono trasferiti in blocchi di 16 byte a 8MB/s senza la possibilità di perdite.

Esempio 1 (Mouse)

- Cicli di clock al secondo spesi per il polling

  = 30 * 400 = 12000 clocks/sec

  - % Processor for polling:

    $12*10^3/500*10^6 = 0.002\%$

    $\Rightarrow$ Fare polling sul mouse «ruba» un utilizzo di processore trascurabile

Questo overhead viene pagato sempre, sia che ci sia il trasferimento, sia che non ci sia

Esempio 2 (Hard disk)

- Numero di volte/sec che occorre fare cicli di attesa per non perdere dati:

  = 8 MB/s /16B = 500K polls/sec

- Spesa in cicli di clock/sec

  = 500K * 400 = 200,000,000 clocks/sec

- % processore

  $200*10^6/500*10^6 = 40\%$

  $\Rightarrow$ Inaccettabile perché pagata sempre