


Gestione dell'Input / Output

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento Patterson: 6.9 e 5.2

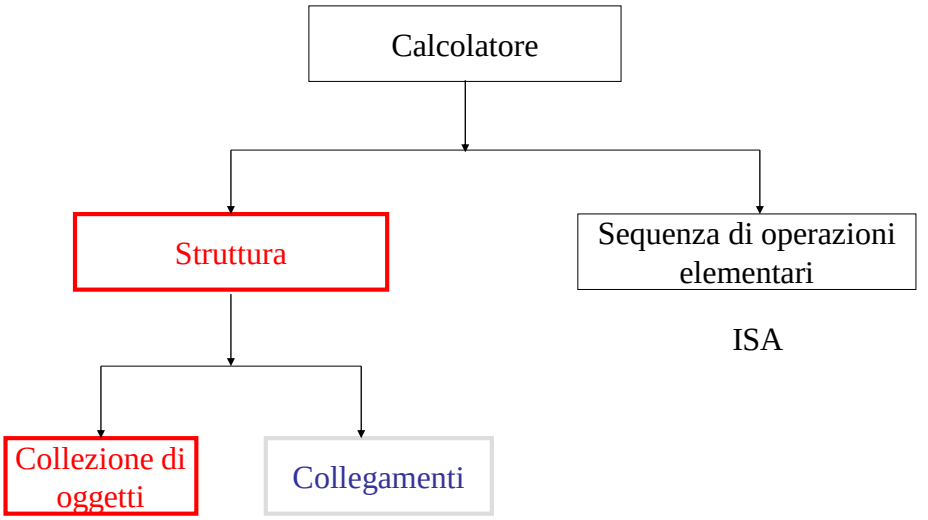

Sommario

I collegamenti tra CPU e gli altri componenti

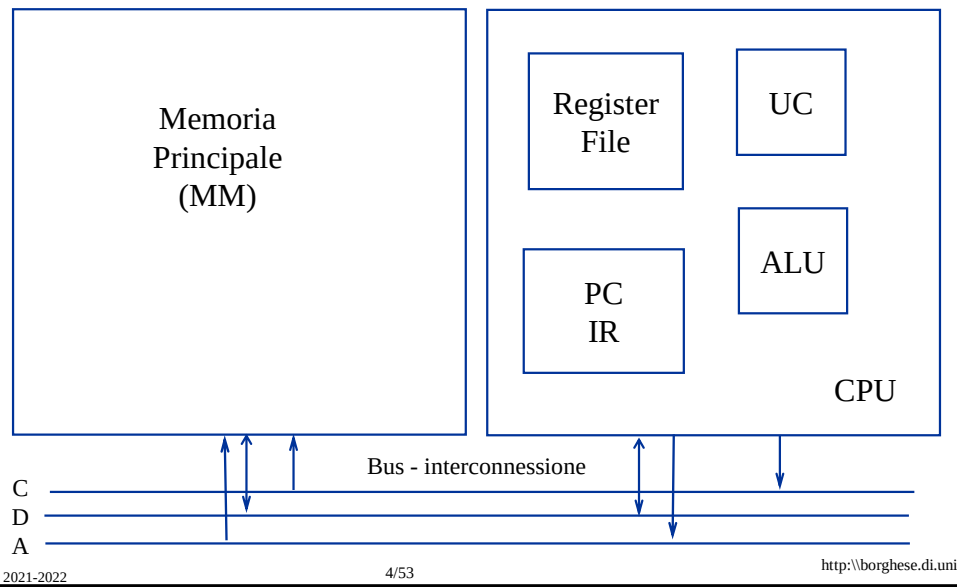
Arbitraggio

Modalità di trasferimento tra I/O e CPU

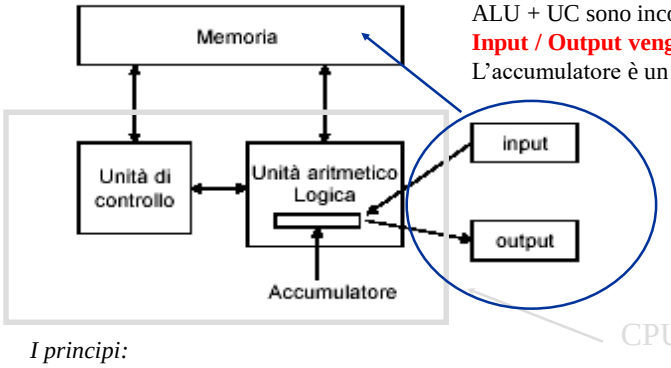
Descrizione di un elaboratore



Gli attori principali di un'architettura



Architettura di Von Neumann

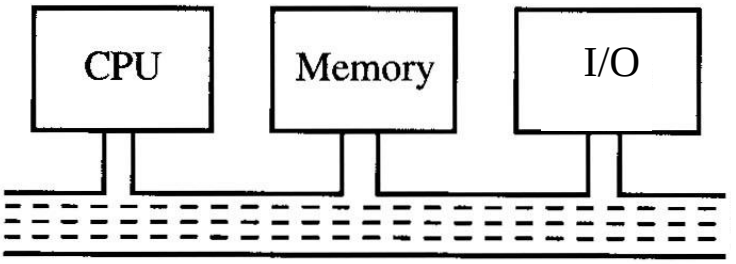


ALU + UC sono incorporate nella CPU.
Input / Output vengono in realtà trasferiti via bus.
L'accumulatore è un possibile tipo di architettura.

I principi:
• I dati e le istruzioni sono memorizzate in una memoria read/write.
• Il contenuto della memoria può essere recuperato in base alla sua posizione, e non è funzione del tipo di dato.
• L'esecuzione procede sequenzialmente da un'istruzione alla seguente.

•Gia' vista e modificata (evoluzione nel tempo).

Il bus



Bus: dispositivo che college questi tre sotto-sistemi di un'architettura.

Le architetture contengono uno o più bus che collegano questi tre componenti.

I/O

E' la parte più complessa di un'architettura per la sua variabilità

Dispositivi eterogenei per:

velocità di trasferimento.

quantità di dati trasferiti.

latenze.

sincronizzazione.

modalità di interazione (con l'uomo o con una macchina).

Dispositivi di I/O - classificazione

Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	Input	Human	0.0001
Mouse	Input	Human	0.0038
Voice input	Input	Human	0.2640
Sound input	Input	Machine	3.0000
Scanner	Input	Human	3.2000
Voice output	Output	Human	0.2640
Sound output	Output	Human	8.0000
Laser printer	Output	Human	3.2000
Graphics display	Output	Human	800.0000–8000.0000
Cable modem	Input or output	Machine	0.1280–6.0000
Network/LAN	Input or output	Machine	100.0000–10000.0000
Network/wireless LAN	Input or output	Machine	11.0000–54.0000
Optical disk	Storage	Machine	80.0000–220.0000
Magnetic tape	Storage	Machine	5.0000–120.0000
Flash memory	Storage	Machine	32.0000–200.0000
Magnetic disk	Storage	Machine	800.0000–3000.0000

+ latenza

La comunicazione tra i componenti e la CPU

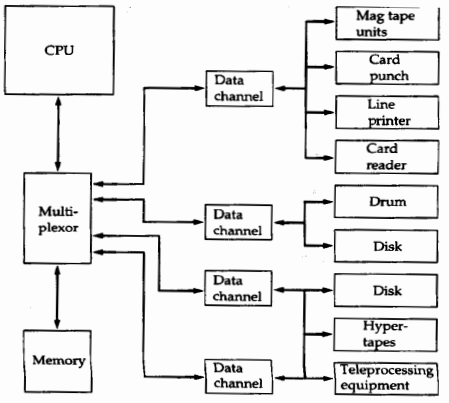


Figure 2.5 An IBM 7094 Configuration

Switch centralizzato - 1959 (multiplexor -> bridge)

Programma di "canale" per la gestione del trasferimento

Architettura a bus condiviso

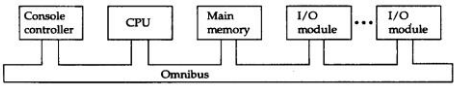
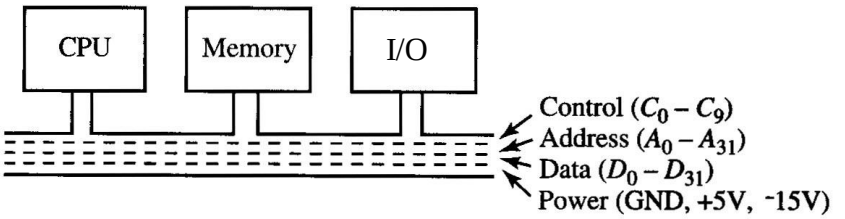


Figure 2.9 PDP-8 Bus Structure

Il bus (connessione a cammino comune)



Pathway che connette tutti i dispositivi in modo bidirezionale (a partire dal PDP-8, omnibus, 1965). Trasferimento **parallelo** dei dati. Connessioni bidirezionali.

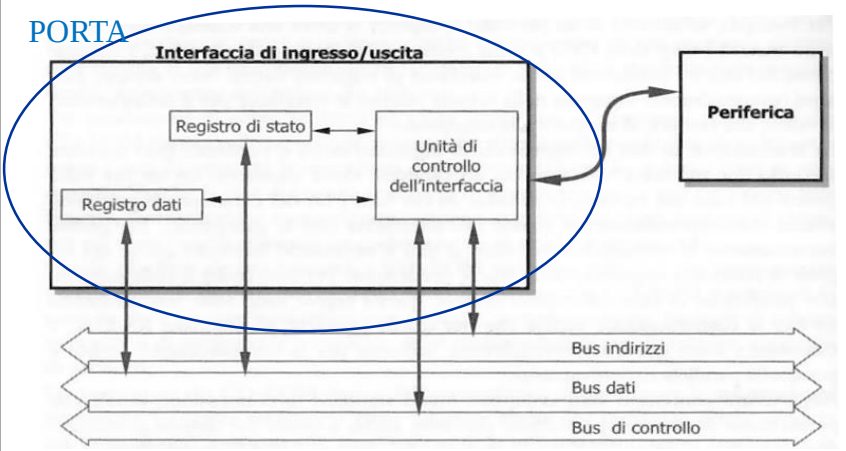
Principali vantaggi della struttura a bus singolo:
- elevata flessibilità (si possono facilmente aggiungere / togliere dispositivi)
- bassi costi.

Problemi:
- I dispositivi non possono trasmettere contemporaneamente.
- I dispositivi devono venire sincronizzati.

Bus & buffer

- I dispositivi sono collegati al bus tramite porte (HW/SW).
- I dispositivi collegati al bus variano in termini di velocità dell'esecuzione delle operazioni e quindi di frequenza con cui i dati vengono resi disponibili per l'I/O ⇒ è necessario un meccanismo di sincronizzazione per garantire il trasferimento efficiente delle informazioni sul bus.
- Tipicamente all'interno delle unità che utilizzano il bus sono presenti dei **registri di buffer** per mantenere l'informazione durante i trasferimenti e non vincolarsi alla velocità del dispositivo più lento connesso al bus **(nasconde la latenza, cf. Dischi, RAM…).**
- All'interno dell'ampiezza di banda massima si può:
  - **Aumentare la velocità di trasferimento.** Buffer grossi.
  - **Ridurre i tempi di risposta (latenza).** Buffer piccoli.

Interfaccia di una periferica



Segnali di controllo: Busy, Ack, Interrupt...
Registri:
•Dati (registro o buffer di memoria)
•Stato: situazione della periferica (idle, busy, down….) e comando in esecuzione.
Micro-controllore dell'interfaccia.

Livelli di definizione

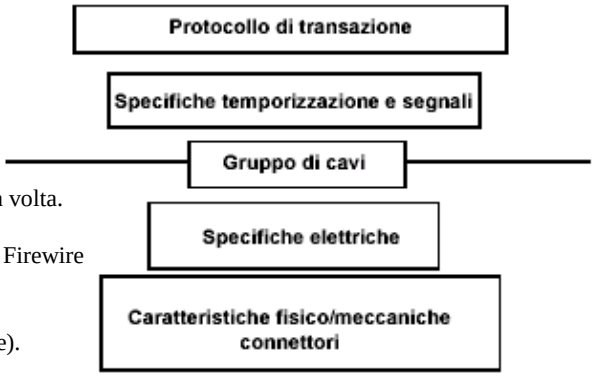
Tipi di bus:

Bus parallelo (bus PCI-32 / 512)

Bus seriale (RS232, RS432). 1 bit alla volta.

Bus I/O seriale ad alta velocità: USB, Firewire (IEEE 1394),

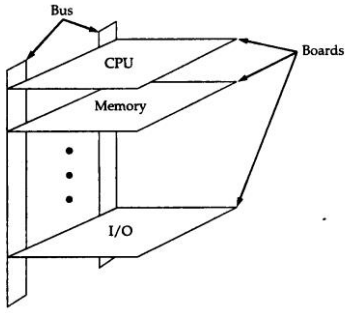
Bus seriale ad altissima velocità (PCIe).



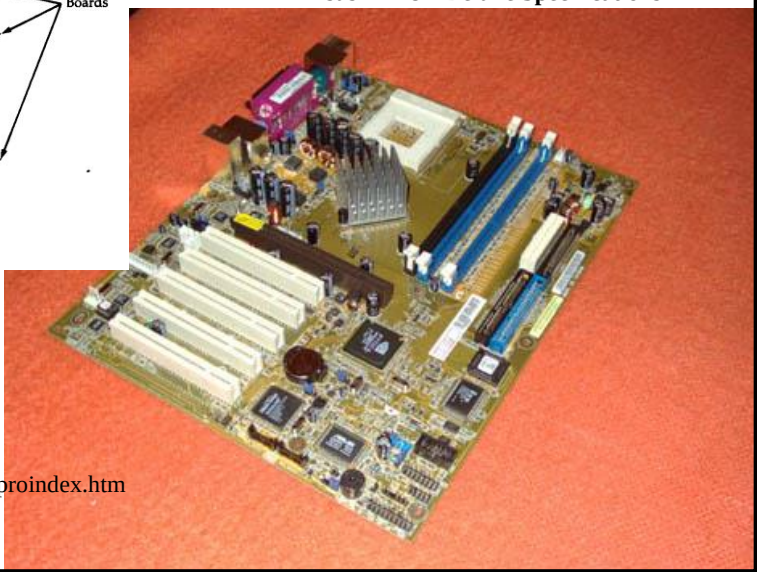
Transazione su bus: sequenza di operazioni che partono da una richiesta, e si concludono con il trasferimento di dati.

I livelli di definizioni riguardano anche la rete Ethernet, e in generale i bus wireless, e.g. Bluetooth, WiFi,…

Esempio di mother board con bus di backplane



Asus A7N8X Deluxe Specifications



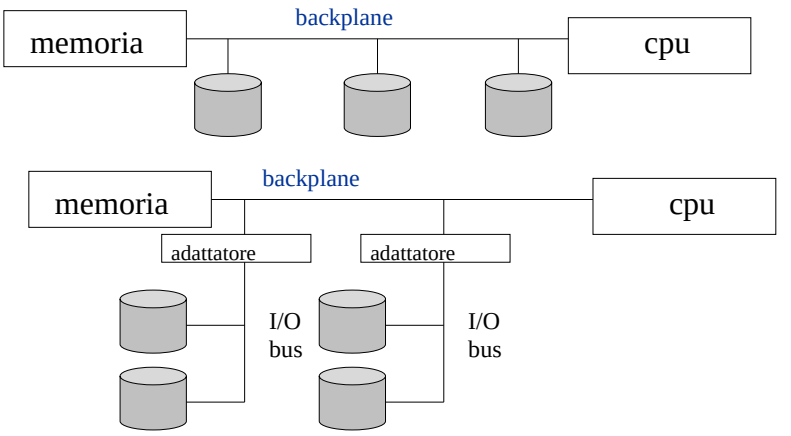
http://www.asus.it/products/proindex.htm

I bus PCI

Caratteristica	PCI	PCI Express
Tipo di bus	Back-plane	Versatile (back-plane, I/O)
Ampiezza di base del bus (numero di segnali per i dati)	32-64 (collegamento bidirezionale)	4 + 4 (In/Out) (collegamento monodirezionale)
Numero di dispositivi master	molti	1
Temporizzazione	Sincrono 33-66Mhz	Sincronizzato: 2.5GHz
Modalità di funzionamento	Sincrona parallela (1,024Mbit/s – 4,096Mbit/s)	Sincrona seriale (2,5Gbit/s)
Ampiezza di banda di picco teorica	133-512MB/s (PCI64)	300MB/s per direzione
Ampiezza di banda stimata raggiungibile per bus di base	80MB/s	1,200MB/s (x 4 linee)
Massimo numero di dispositivi	1024 (32 dispositivi per segmento)	1
Massima lunghezza del bus	0,5 metri	0,5 metri
#Mbyte / s / linea	4-16	75 + 75
Nome dello standard	PCI	PCI - Express

bottleneck

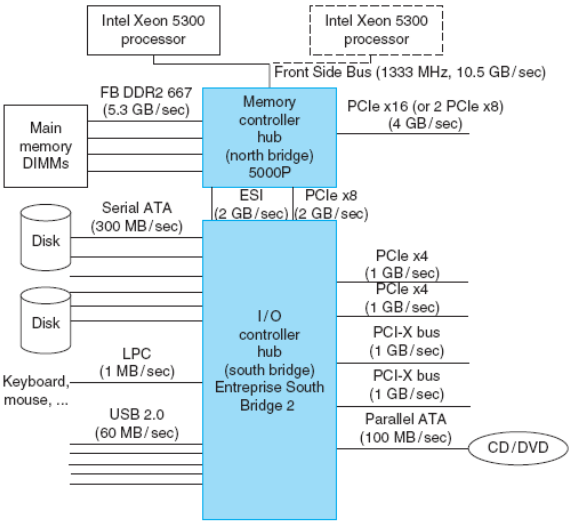
Tipologie di bus – vecchio stile



Struttura gerarchica dei bus topologia a stella degli adattatori -> hub

Gli adattatori sono dispositivi attivi – bridges.

CPU Xeon biprocessore



I bridge (controllo dei bus)

	Intel 5000P chip set	Intel 975X chip set	AMD 580X CrossFiret
Target segment	Server	Performance PC	Server/Performance PC
Front Side Bus (64 bit)	1066/1333 MHz	800/1066 MHz	—
Memory controller hub ("north bridge")			
Product name	Blackbird 5000P MCH	975X MCH	
Pins	1432	1202	
Memory type, speed	DDR2 FBDIMM 667/533	DDR2 800/667/533	
Memory buses, widths	4 × 72	1 × 72	
Number of DIMMs, DRAM/DIMM	16, 1 GB/2 GB/4 GB	4, 1 GB/2 GB	
Maximum memory capacity	64 GB	8 GB	
Memory error correction available?	Yes	No	
PCIe/External Graphics Interface	1 PCIe x16 or 2 PCIe x	1 PCIe x16 or 2 PCIe x8	
South bridge interface	PCIe x8, ESI	PCIe x8	
I/O controller hub ("south bridge")			
Product name	6321 ESB	ICH7	580X CrossFire
Package size, pins	1284	652	549
PCI-bus: width, speed	Two 64-bit, 133 MHz	32-bit, 33 MHz, 6 masters	—
PCI Express ports	Three PCIe x4		Two PCIe x16, Four PCI x1
Ethernet MAC controller, interface	—	1000/100/10 Mbit	—
USB 2.0 ports, controllers	6	8	10
ATA ports, speed	One 100	Two 100	One 133
Serial ATA ports	6	2	4
AC-97 audio controller, interface	—	Yes	Yes
I/O management	SMbus 2.0, GPIO	SMbus 2.0, GPIO	ASF 2.0, GPIO

SATA - Serial Advanced Technology Attachment (evoluzione dell'interfaccia IDE verso i dischi)

Sommario

I collegamenti tra CPU e gli altri componenti

Arbitraggio

Modalità di trasferimento tra I/O e CPU

La transazione su bus

Invio dell'indirizzo
Lettura / scrittura nel dispositivo di I/O

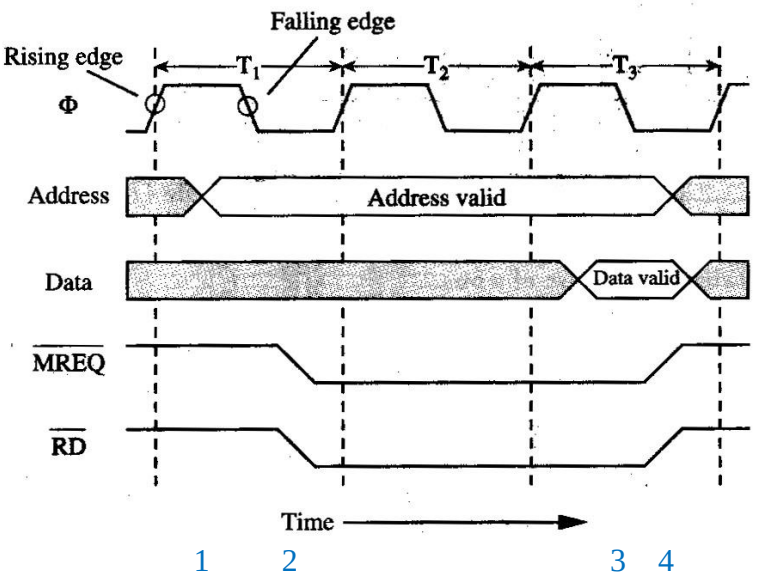
Esistono due schemi principali di comunicazione su di un bus (di operare una transazione):
Sincrono
Asincrono

Bus sincroni

- The le linee di controllo è presente la linea che porta il segnale di clock **(bus clock).** Esiste un protocollo di comunicazione scandito dai cicli di clock, in generale diverso (ma sicnronizzato) da quello della CPU.
- Questo tipo di protocollo permette di ottenere bus molto veloci.
- *Svantaggi:*
  - Ogni device deve essere sincronizzato.
  - Lunghezza limitata (per evitare che i ritardi nei fronti dovuti alla propagazione producano disallineamenti, clock skew).
  - Tutti i dispositivi devono potere lavorare alla frequenza imposta dal bus clock.
- I bus processor-memory sono spesso sincroni in quanto:
  - hanno dimensioni ridotte.
  - hanno pochi elementi connessi.

Ciclo di bus (***bus cycle***): numero di cicli per effettuare una transazione: tipicamente da 2 a 5 cicli di bus clock.

Bus sincroni: esempio



Controllo mediante FSM

Bus asincroni

- Un bus asincrono **non** è dotato di clock.
- La comunicazione tra due parti avviene mediante un protocollo di **handshaking**.

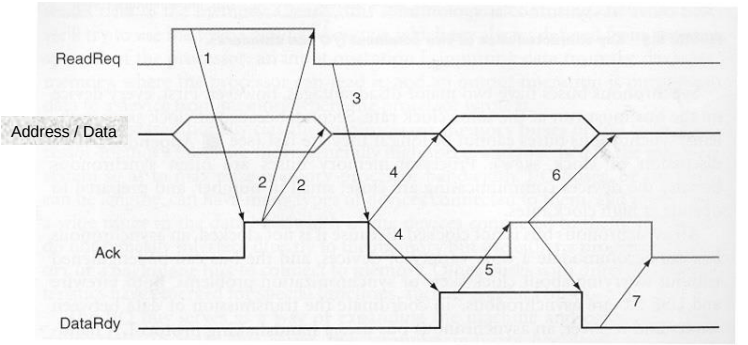
  (!MSYN) -> Job -> (!SSYN) -> (MSYN) -> (SSYN)
- I bus asincroni possono avere lunghezza elevata per connettere molti dispositivi.
- Sono efficienti quando i tempi di esecuzione delle varie periferiche variano molto tra loro.
- Spesso i bus di I/O sono asincroni.

Esempio di handshaking

ReadReq: segnale di controllo di MemoryRead (MREQ+RD). In corrispondenza di questo segnale l'**indirizzo** della parola di memoria viene inviato sul bus dati.

Data Rdy: viene utilizzato per indicare che la parola è pronta sulle linee di dato. Questo segnale viene inviato dalla memoria quando il dato è disponibile in uscita dalla memoria.

Ack: viene utilizzato come risposta ai 2 segnali precedenti. Feed-back (letto indirizzo, letto / scritto il dato).



I segnali di ReadReq e DataReady rimangono alti fino a quando il ricevente (la memoria) non ha visto il comando e letto / scritto i dati corrispondenti.

Arbitraggio del bus

Protocollo La comunicazione su bus deve essere regolata attraverso un **protocollo di comunicazione**.

Viene introdotto il concetto di **bus master (padrone del bus)**, il cui scopo è quello di controllare l'accesso al bus.

L'architettura più semplice è quella che prevede un unico bus master (il processore) in cui tutte le comunicazioni vengono mediate dal processore stesso.

Questo può creare un collo di bottiglia. Ad esempio nel caso di trasferimento di dati da I/O a memoria.

In questo caso occorre definire e rispettare una policy che coordini i vari dispositivi bus master. Questa policy si chiama di **arbitraggio** del bus. Un solo dispositivo alla volta può essere master, tutti gli altri ascoltano.

Questo è il principale inconveniente dei bus a nodo comune.

La gestione dell'arbitraggio è più semplice nelle architetture di bus con bridge, i quali controllano direttamente anche l'arbitraggio.

Protocollo di arbitraggio

Occorre stabilire quale master autorizzare all'utilizzo del bus.

Ad ogni dispositivo viene assegnata una *priorità*.

Il dispositivo a priorità maggiore può accedere prima al bus.

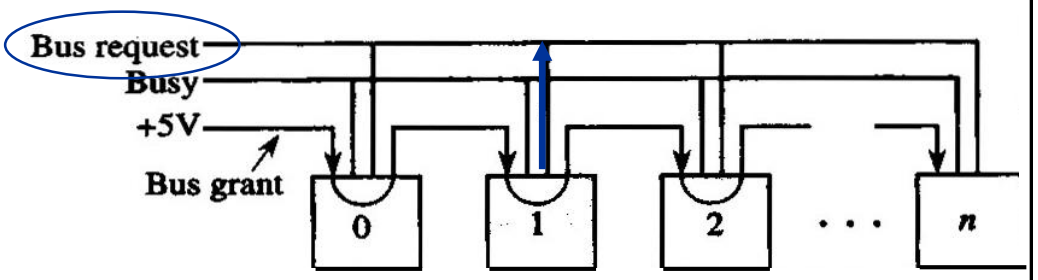
Meccanismo di accesso al bus diventa:
1. Richiesta del bus (*bus request*)
2. Assegnamento del bus (*bus grant*)

Problema è assicurare una *fairness*.

Compromesso tra fairness e priorità.

Un arbitro si preoccupa quindi di gestire *bus request* e *bus grant.*

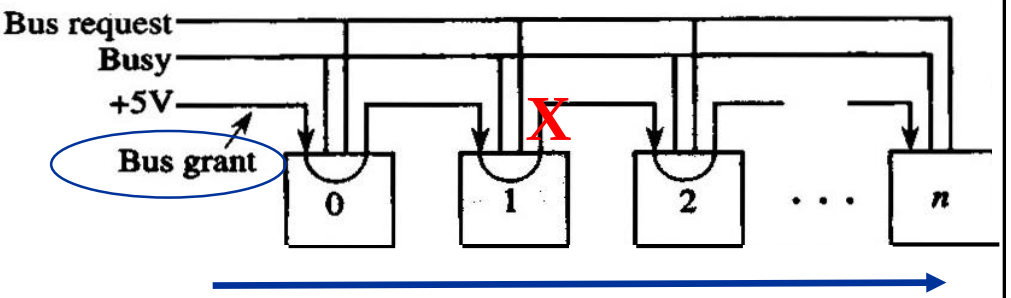
Arbitraggio decentralizzato: daisy chain



Busy definisce se il bus è occupato o meno.

Il trasferimento può iniziare solo quando il bus non è busy.

Arbitraggio decentralizzato: daisy chain



Busy = 0

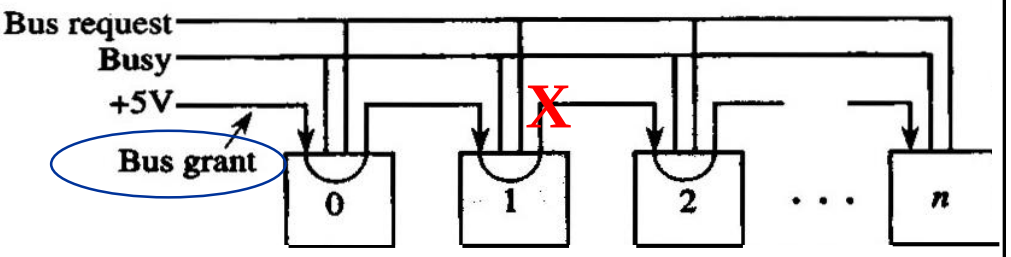
Bus request -> bus grant (a meno di inibizioni)

Grant viene bloccato dal primo dispositivo incontrato che ha un segnale di request alto.

Viene abbassato il segnale di request a alzato il segnale di busy -> trasferimento dati.

.

Arbitraggio decentralizzato: daisy chain



Busy = 1

Bus request -> bus grant (a meno di inibizioni)

Grant viene bloccato dal primo dispositivo incontrato che ha un segnale di request alto.

a) Se il dispositivo incontrato per primo è quello che sta trasferendo, dopo un time out il dispositivo che ha fatto la richiesta (che si troverà dopo) può reiterare.
b) Se il dispositivo incontrato per primo è il dispositivo che ha fatto la richiesta, il dispositivo che sta trasferendo si vedere ritirare il segnale di grant e termina la transazione mettendo busy = 0.

Viene abbassato il segnale di request a alzato il segnale di busy -> trasferimento dati.

Arbitraggio distribuito con autoselezione

In questo schema un dispositivo che vuole prendere il controllo del bus, deve:

- 1) Inviare il segnale di richiesta del bus.
- 2) Scrivere sul bus il codice che lo identifica.
- 3) Controllare se il bus è libero.
- 4) Se il bus è libero ma ci sono richieste contemporanee di bus, controllare il codice dei dispositivi che hanno fatto richiesta.
- 5) Occupare il bus se è libero o i dispositivi hanno priorità minore: inviare 0 sulla linea di bus grant ai dispositivi successivi, asserisce la linea di busy (il bus è occupato), altrimenti non fare nulla. *Ciascun dispositivo è arbitro.*
- 6) Deasserire la richiesta del bus.

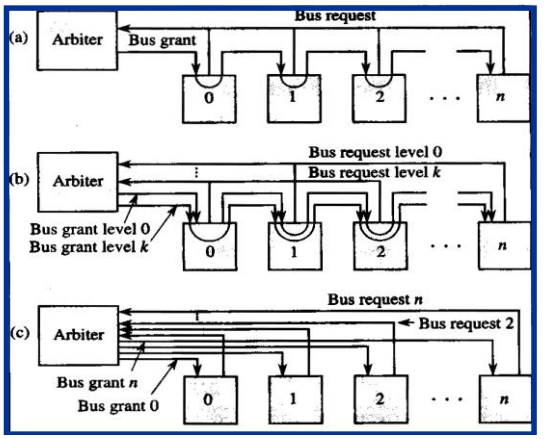
Problema: rilevamento delle collisioni. Occorre prevedere un segnale di ricevuto.

Schemi di arbitraggio centralizzati

Arbitraggio
In Daisy Chain

Arbitraggio
Centralizzato
Con priorità

Arbitraggio
Centralizzato
Parallelo



- Arbitraggio centralizzato parallelo non scala con il numero di dispositivi (e.g. arbitraggio nei bridge)
- Nell'arbitraggio centralizzato con priorità, esistono k segnali di grant per k gruppi di dispositivi, gestiti in daisy chain (e.g. Bus firmware).

Sommario

I collegamenti tra CPU e gli altri componenti

Arbitraggio

Modalità di trasferimento tra I/O e CPU

Gestione dell'I/O

2 modalità di gestione dell'input / output da parte della CPU:

- Memory-mapped
- Istruzioni speciali di I/O

Istruzioni speciali di I/O

Istruzioni appartenente alla ISA che indirizzano direttamente il dispositivo di I/O (i registri del dispositivo):
- Numero del dispositivo
- Parola di comando (o indirizzo della parola che contiene il comando)

La CPU può inviare:
- il numero del dispositivo (numero della porta) -> indirizzo
- i dati da scrivere nel dispositivo (o aspettare i dati da leggere)
- Inviare dei comandi al dispositivo.

Rendendo le istruzioni illegali al di fuori del kernel mode del processore, i programmi utenti non accedono direttamente ai device controller.

Esempio di architetture di questo tipo: Intel IA-32, IBM370.

```
out 70h, ax    # Trasferisci alla porta di I/O 70, il contenuto di AX
```


Indirizzamento memory-mapped

•I registri del device controller sono considerati come celle di memoria situate nello spazio Kernel di I/O.
• Non fanno parte della memoria RAM anche se condividono lo spazio di indirizzamento.
•Il processore esegue operazioni di I/O come se fossero operazioni di lettura/scrittura in memoria.

Esempio:

sw $s0, indirizzo
lw $s0, indirizzo

dove l'*indirizzo* è al di fuori dallo spazio fisico della memoria RAM.

• I controller ascoltano tutti i segnali in transito sul bus (*bus snooping*) e si attivano solamente quando riconoscono sul bus indirizzi, l'indirizzo corrispondente alla propria locazione di memoria (o porta).

• Gli indirizzi riservati ai registri del controller fanno di solito riferimento alla porzione di memoria riservata al SO e non accessibile quindi al programma utente.

• I programmi utente devono quindi passare dal SO per accedere a questi indirizzi riservati (**modalità kernel**) e quindi effettuare operazioni di I/O. Questo è quanto viene fatto ricorrendo alle istruzioni dell'ISA di *System Call (syscall)*.

Funzionamento di un driver

Funzione dei driver (sono implementati nel **controllore della porta/interfaccia di I/O**.

- Controllano l'operato dei device controller.
- Gestiscono lo scambio dei dati dal controller (registro dati) e la memoria.

1. CPU richiede alla periferica (controller della periferica) l'esecuzione di un'operazione di read o di write.
2. I dati coinvolti nell'operazione devono essere trasferiti da e verso la memoria centrale.

Per potere eseguire un'operazione di read / write, occorre spesso una serie di operazioni sul dispositivo, che vengono eseguite attraverso il controller.

Esempio: una stampante ha 1 registro dati ed 1 registro di stato. Il registro di stato contiene il bit done, che viene impostato a 1 quando il carattere è stato stampato; ed il bit error che, indica se ci sono problemi. Il processore deve controllare che non ci siano errori e che il bit di done sia stato settato ad 1 prima di inviare un altro dato.

Per potere inviare i comandi al controller, occorre prima avere individuato il controller giusto!!
A ciascun dispositivo viene dato uno o più numeri -> indirizzo personalizzato.

Modalità di trasferimento dati

Controllo da programma:
con polling.

Mediante interrupt:
con DMA.

I/O a controllo di programma

E.g. chiamata syscall per la stampa di una stringa.

La periferica ha un ruolo passivo. Il processore esegue tutto il lavoro.

```
begin
        1. Predisponi i registri del controller ad effettuare una
        operazione di lettura.
        2. While (ready-bit = = 0) do;          // spin lock (o busy waiting)
        3. Carica il dato acquisito;
end;
```

Svantaggio: La CPU dopo avere predisposto il controller all'esecuzione dell'I/O si ferma e si mette ad interrogare il registro di stato della periferica in attesa che il ready bit assuma un determinato valore. Stato *busy waiting* o *spin lock*.

Esempio: Receiver (tastiera)

```
# NB i dispositivi vengono indirizzati tramite gli indirizzi "alti".

.text
.globl main
main:
        li $t0, 0xC000 0000         # indirizzo del receiver status register (3 Gbyte)
        li $t2, 0xC000 0004          # indirizzo del receiver data register

# Ciclo di spin-lock associato alla lettura di un carattere
ciclo:  lw $t1, 0($t0)              # Contenuto del registro di stato
        beq $t1, $zero, ciclo       # if ($t1 != 0) esci

        lw $a0, 0($t2)              # Caricamento del dato in a0

        li $v0, 10                  # exit
        syscall
```

I/O a gestione di programma - costo

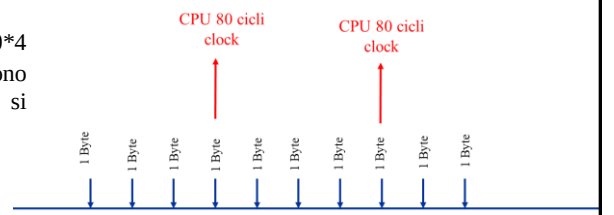
Ipotesi:
1) Tastiera gestita a controllo di programma che opera a 0,01Kbyte/s (10 tasti / secondo)
2) Frequenza di clock: 50Mhz.

Determinare il tempo in cui verrebbe effettivamente utilizzata la CPU per trasferire 1 word (su 4 byte), tenendo conto che ci vogliono 20 cicli di clock per trasferire ogni byte.

Tempo per trasferire una word = 4 [byte] / 10 [byte] / [s] = 0,4 s

#cicli_clock = $50*10^6$ [#cicli] / [s] * 0,4 [s] = 20 * 10^6 [#cicli]

Per il trasferimento vengono richiesti 20*4 cicli di clock (cicli utili), ma questi sono «sparsi» tra 20 milioni di cicli che si verificano in 0,4 s.



%Sfruttamento della CPU è:

$80/(20,0 * 10^6) *100 = 0,0004\%$

Se la CPU fosse un micro-controllore potrebbe anche andare bene, ma…

Polling

Interrogazione del registro di stato della periferica.

Ciclo di polling: durante un ciclo di busy-waiting su un dispositivo si esegue il **polling** sugli altri dispositivi di I/O.

Quando una periferica necessita di un qualche intervento, si soddisfa la richiesta e si prosegue il ciclo di polling sugli altri I/O.

```
// Leggi dato da perif_x
begin
a.   Predisponi i registri dei controller ad eseguire una read;
b.   if(ready_bit (perif_1) = = 1) servi perif_1;    #Esempio: Mouse
     if (ready_bit (perif_2) = = 1) servi perif_2;   #Esempio: Hard disk1
     if (ready_bit (perif_3) = = 1) servi perif_3;   #Esempio: Hard disk2

          ….
     if (ready_bit (perif_n) = = 1) servi perif_n;
     UpdateFunctions;                                #Programma di gestione in funzione della
                                                     #situazione delle periferiche (sistemi di controllo)
     goto b;
end;
```

I/O a gestione di programma – costo polling

Ipotesi:

1) Costo del polling (# cicli di clock per un'operazione di polling, costituita da trasferimento del controllo alla procedura di polling, accesso al dispositivo di I/O, trasferimento dati e ritorno al programma utente): 400 cicli di clock. Se non si trasferiscono dati il costo è considerato trascurabile.
2) Frequenza di clock: 500Mhz
3) Parola di 4 byte.

Determinare l'impatto del polling per 3 dispositivi diversi:

A) Mouse. Deve essere interrogato almeno 30 volte al secondo per non perdere alcun movimento dell'utente.
B) Hard disk1. Trasferisce dati al processore in parole da 16 bit ad una velocità di 50 Kbyte/s.
C) Hard disk2. Trasferisce dati al processore in blocchi di 4 parole e può trasferire 4 Mbyte/s.

Supponiamo che il costo del trasferimento per la CPU sia dovuto principalmente alle operazioni di preparazione della memoria (latenza), mentre il costo per la CPU dovuto al trasferimento tramite il bus sia trascurabile (ad esempio perchè viene utilizzata la modalità *burst* delle DRAM).

I/O a gestione di programma – costo mouse

Frequenza di clock della CPU: 500Mhz
Costo polling: 400 cicli di clock

Mouse:

(Per ogni accesso trasferisco **2 Byte**: x, y.

Occorre una granularità temporale di almeno 30 accessi / s -> transfer rate = 60 Byte/s

30 accessi al secondo in polling -> 30 accessi / sec x 400 cicli_clock = 12,000 cicli_clock/s

Percentuale di utilizzo della CPU: 12,000 cicli /sec / 500,000,000 cicli /sec =
0,000024 => 0,0024%

Fortunatamente la CPU grazie al polling puo' fare altro tra un accesso e l'altro.

La CPU spende 12,000 cicli di clock per una lettura, poi fa altro, poi fa una seconda lettura e così via.

I/O a gestione di programma – costo dischi

Frequenza di clock della CPU: 500 MHz.
Costo polling: 400 cicli di clock

Hard disk1: Trasferisce dati al processore in parole da 16 bit ad una velocità di 50 KByte/s.

Per ogni accesso possiamo trasferire 1half word.
Occorrono quindi 25k accessi/s.
In termini di cicli di clock: 25 k accessi/s x 400 cicli = 10 Mcicli_clock/s
Percentuale di utilizzo della CPU: 10 x 10^6 cicli / sec / 500 x 10^6 cicli /sec =
0,02 => 2%

Hard disk2: Trasferisce dati al processore in blocchi di 4 parole e può trasferire 4 MByte/s.
Per ogni accesso possiamo trasferire 16 byte.
Occorrono quindi: 4 Mbyte/s / 16 Byte /accesso = 250 k accessi/s
In termini di cicli di clock: 250k accessi/s x 400 cicli = 100 Mcicli_clock/s
Percentuale di utilizzo della CPU: 100 x 10^6 cicli / sec / 500 x 10^6 cicli /sec =
0,2 => 20%

Ciclo di polling complessivo: 12 x 10^3 + 10 x 10^6 + 100 x 10^6 = 110,012 x 10^6 cicli /sec
Percentuale di utilizzo della CPU: 110,012 x 10^6 cicli / sec / 500 x 10^6 cicli /sec = 0.220024 = 22,0024%

Nascondo la latenza di mouse e di Hard disk1. Ma alla fine utilizzo la CPU per meno del 75% del tempo.

I/O a controllo di programma

- I miglioramenti del polling rispetto al controllo di programma sono molto limitati.
- I problemi principali del polling (e dell'I/O a controllo di programma) sono:
  - Con periferiche lente, un eccessivo spreco del tempo di CPU, che per la maggior parte del tempo rimane occupata nel ciclo di busy waiting.
  - Con periferiche veloci, il lavoro svolto dalla CPU è quasi interamente dovuto all'effettivo trasferimento dei dati.

Il polling funziona bene per i sistemi embedded. E per i micro-controllori (e.g. bridge)

Nei dischi, si potrebbe attivare il polling quando ne è richiesto l'utilizzo: la latenza per il tempo per il posizionamento delle testine (=> spin lock) viene nascosto da altri input/output.

Occorre disaccoppiare il trasferimento tra I/O e memoria e il funzionamento della CPU, per trasferimenti di grandi quantità di dati.

Interrupt

E' la periferica a segnalare al processore (su una linea del bus dedicata) di avere bisogno di attenzione.

La segnalazione viene chiamata ***interrupt*** perché interrompe il normale funzionamento del processore (***interrupt request***).

Quando il processore "se ne accorge" (fase di fetch), riceve un segnale di ***interrupt acknowledge***.

Viene eseguita una procedura speciale, chiamata ***procedura di risposta all'interrupt***.

Problema: Il programma utente deve potere procedere dal punto in cui è stato interrotto ➔ *Salvataggio del contesto*.

Interrupt – esempio – comando print

1. Invio del comando print.
2. Se la periferica è in stato busy, CPU torna alla sua attività, scaricando sul registro di controllo la richiesta di output.
3. Quando la periferica diventa ready, viene inviato un interrupt.
4. Il programma di risposta all'interruzione, provvederà a trasferire alla periferica il dato che su vuole stampare.

I/O ad interrupt - costo

Frequenza di clock è 500Mhz
Il costo di ogni interruzione è 500 cicli di clock (comprensivi dello switch dello stato)

Hard disk2:

Trasferimento di blocchi di 4 parole (16 Byte)
Trasferimento a 4MByte/s
Il disco sta trasferendo dati solamente per il 5% del tempo.

Per ogni accesso trasferiamo 16 Byte
ccorrono quindi 4 Mbyte/s / 16 Byte = 250k accessi/s (250k interrupt/s).
In termini di cicli di clock: 250 k accessi/s x 500 cicli = 125 Mcicli_clock/s
Percentuale di utilizzo della CPU: 125 x 10^6 cicli / sec / 500 x 10^6 cicli /sec = 0,25 => 25%
(con polling era il 20%, ma nel resto del tempo la CPU fa altro!)

Trasferimento per il 5% del tempo

- Con polling avremmo spin-lock nel 95% del tempo
- Con interrupt avremo:

  Frazione di utilizzo del processore, tenendo conto che il disco trasferisce solo per il 5% del tempo: 125 Mcicli *0.05 / 500M cicli = 0.0125 => = 1,25%

NB L'interrupt è più costoso del polling dal punto di vista dell'esecuzione in senso stretto, ma il costo si recupera perchè l'esecuzione della risposta all'interrupt è attiva solamente in concomitanza dell'interrupt.

DMA

Tra il momento in cui termina l'invio del comando al controller ed il momento in cui il dato è disponibile sul controller, la CPU può fare altro (tipicamente l'esecuzione di un altro programma).

Il meccanismo interrupt driven non svincola la CPU dal dovere eseguire le operazioni di trasferimento dati.

Per periferiche veloci, le operazioni di trasferimento dati occupano un tempo preponderante rispetto al tempo speso in spin lock.

Per evitare l'intervento della CPU nella fase di trasferimento dati, è stato introdotto il protocollo di trasferimento in Direct Memory Access (DMA).

Viene disaccoppiato il colloquio processore-Memoria dal colloquio IO-Memoria. Questo è reso più facile dalla struttura a bus gerarchici.

Il device controller che gestisce il trasferimento diventa **bus master**.

Caratteristiche della DMA

Il DMA controller è un processore specializzato nel trasferimento dati tra dispositivo di I/O e memoria centrale.

Per attivare il trasferimento viene richiesto alla CPU:

1. Spedire al DMA controller il tipo di operazione richiesta
2. Spedire al DMA controller l'indirizzo da cui iniziare a leggere/scrivere i dati.
3. Spedire al DMA controller il numero di byte riservati in memoria.

Per attivare il trasferimento al controller viene richiesta la corretta lettura dello stato della memoria e l'aggiornamento dell'indirizzo a cui trasferire il dato. *E' il controller che gestisce il trasferimento del singolo dato.*

La CPU si svincola completamente dall'esecuzione dell'operazione di I/O.

Il controller avvia l'operazione richiesta e trasferisce i dati da/verso memoria mentre la CPU sta facendo altro.

Dopo avere trasferito tutti i dati, il DMA invia un interrupt alla CPU per segnalare il completamento del trasferimento.

La CPU perciò controlla il (device) controller.

I/O DMA - costo

Frequenza di clock è 500Mhz
Il costo dell'inizializzazione del DMA è di 1000 cicli di clock.
Il costo dell'interruzione al termine del DMA è di 500 cicli di clock.

Hard disk2:
Trasferimento di blocchi di 8kbyte per ogni DMA.
Trasferimento a 4Mbyte/s (come per il disco precente)

Per ciascun trasferimento DMA occorre:

1000 + 500 cicli di clock = tempo di inizio + tempo di fine
Numero di DMA: 4(Mbyte/s) / 8kByte / accesso = 500 accessi / s (500 DMA /s)
Numero di cicli di clock richiesti: 1500 cicli * 500 accessi /s = 750,000 cicli /s
Frazione del processore utilizzata: 750k / 500M = 0,15% (rispetto al 20% in precedenza)
NB durante il trasferimento DMA la Memoria Principale è occupata.

E' sottointeso che il tempo di trasferimento sia << al periodo di attivazione della DMA.
La DMA viene attivata ogni 2ms (500 DMA / s).
Il tempo richiesto per l'attivazione è di 1500 cicli di clock (=2 ns *1500 = 3 μs)
Il tempo richiesto per trasferire da I/O a memoria tramite bus 8Kbyte deve essere << (2ms-3μs)
Trasferimento burst a 4 Gbyte / s (0,25 ns / Byte) => 8kByte => 2μs
Totale costo trasferimento dati: 3μs (controller DMA) + 2μs = 5μs ∀DMA = ∀2ms

Sommario

I collegamenti tra CPU e gli altri componenti

Arbitraggio

Modalità di trasferimento tra I/O e CPU