Corso di Calcolatori Elettronici I
A.A. 2010-2011

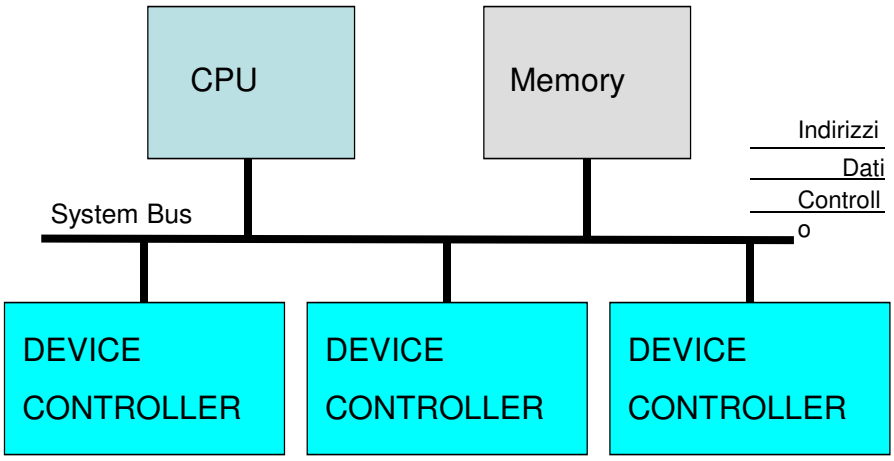
Il sistema di Input/Output

Lezione 35

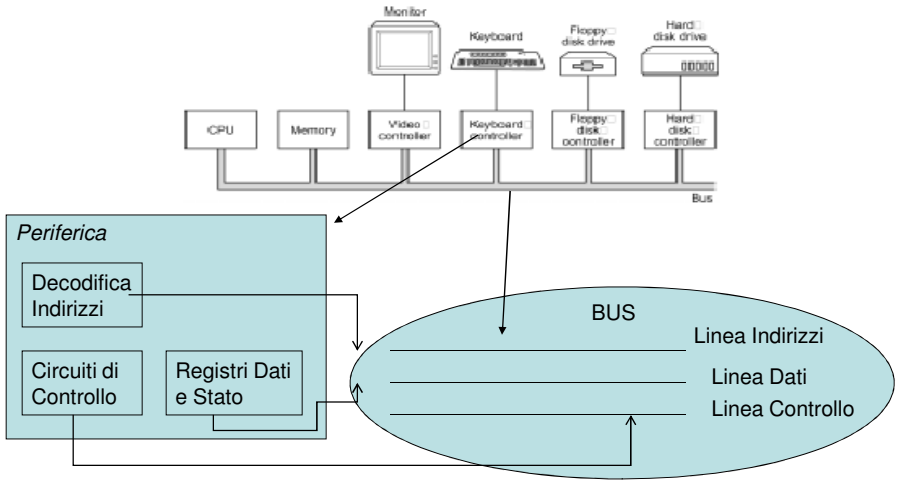
Prof. Roberto Canonico

Università degli Studi di Napoli Federico II
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica (allievi A-DE+Q-Z)
Corso di Laurea in Ingegneria dell'Automazione

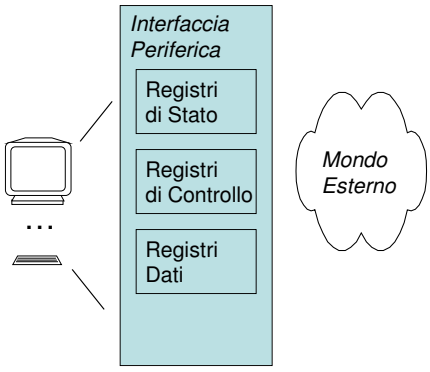
Modello architetturale: bus singolo



Il Calcolatore e le periferiche



Interfaccia



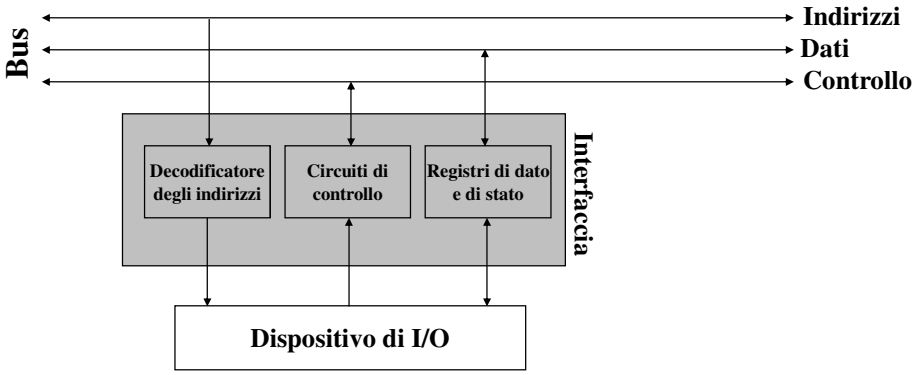
E' il modo in cui la periferica si presenta all'esterno.

In genere è una collezione di registri che permettono di

- Osservare lo Stato
- Modificare lo Stato
- Fornire Dati

Indirizzamento dei Dispositivi di I/O

- A ciascun dispositivo di I/O è di solito associata un'interfaccia, che lo collega al bus del sistema



Elementi Fondamentali della comunicazione CPU-periferica

- Lato Periferica
  - Interfaccia della periferica
  - Protocollo di Comunicazione
- Lato Processore
  - Meccanismo di Controllo
  - Tipo di istruzioni di I/O
  - Driver

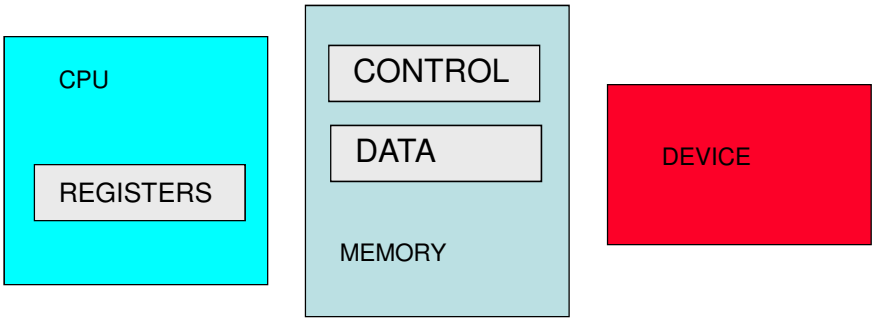
Porte

- La comunicazione tra il processore (o la memoria) ed il dispositivo avviene attraverso i registri delle interfacce (o *porte*)
- Questi sono accessibili tramite il bus di sistema, al pari delle celle di memoria, in quanto a ciascuno di essi è associato un indirizzo
- La loro connessione al bus può avvenire secondo 2 modalità:
  - *memory-mapped* I/O
  - *isolated* I/O

Modelli di programmazione

- Memory Mapped:
  - Le periferiche e la memoria condividono il medesimo spazio di indirizzamento
  - Le operazioni di I/O si eseguono mediante istruzioni normali, su locazioni di memoria particolari
  - E' la soluzione adottata dal 68000
- Isolated I/O con istruzioni speciali:
  - Le periferiche e la memoria hanno spazi di indirizzamento distinti
  - Le operazioni di I/O si eseguono mediante istruzioni dedicate (es. IN ed OUT dei processori Intel x86)

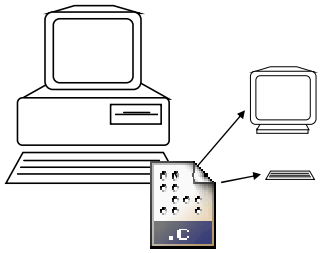
Memory Mapped I/O: schema concettuale



Meccanismi di Gestione dell'I/O

- Si differenziano per il diverso livello di coinvolgimento della CPU nella gestione dei dispositivi di I/O.
- Possibili soluzioni sono:
  - I/O programmato
  - Interrupt
  - Direct Memory Access (DMA)
  - Processori di I/O.

Il Driver



E' il programma che gestisce il dialogo tra il processore e la periferica.

Implementa il protocollo di comunicazione e gestisce la comunicazione

Astrae e semplifica la comunicazione tra processore e periferica

I/O Controllato da programma:

Il Driver è un programma come gli altri

Interrupt:

Il Driver è tipicamente un procedura particolare (ISR)

Meccanismi di I/O

Controllato da programma

Il Processore controlla lo stato della periferica in continuazione aspettando un cambiamento dello stato

Interruzioni

Il Processore procede nel suo lavoro, quando la periferica cambia stato manda un segnale al processore che interrompe il lavoro corrente e procede a gestire l'evento.

Accesso Diretto in Memoria

La periferica è in grado di accedere alla memoria senza l'intervento del Processore

I/O Programmato

- In questo caso la gestione dei dispositivi di I/O è totalmente demandata alla CPU.
- Ogni dato viene prima trasferito dal buffer associato alla periferica ad un registro interno della CPU, e poi immagazzinato in memoria (o viceversa).
- Lo spostamento di ciascun dato implica l'esecuzione di almeno un'istruzione da parte della CPU.
- Quando l'I/O programmato è basato sulla ripetizione di un test sul registro di stato per verificare quando il programma può procedere oltre, si parla di *polling*.

Limiti dell'I/O programmato

- Ogni dispositivo deve dipendere dalla CPU per essere servito. Ne consegue che:
  - l'efficienza (in termini di uso del dispositivo) dipende dalla frequenza con cui il test viene ripetuto
  - tutti i dati devono passare attraverso la CPU, e non esiste connessione diretta tra dispositivo e memoria
  - la CPU dedica una parte del suo tempo ad eseguire banali operazioni di test e trasferimento dati.

Protocolli di Comunicazione

Un Protocollo è quell'insieme di regole che gestiscono la comunicazione tra due entità.

Protocollo Sincrono

E' previsto un segnale di sincronizzazione (clock) che permette di gestire la temporizzazione delle comunicazioni

Protocollo Asincrono

Tutta la temporizzazione della comunicazione è gestita dal protocollo stesso attraverso lo scambio dei messaggi.

HandShake *(Protocollo Asincrono)*

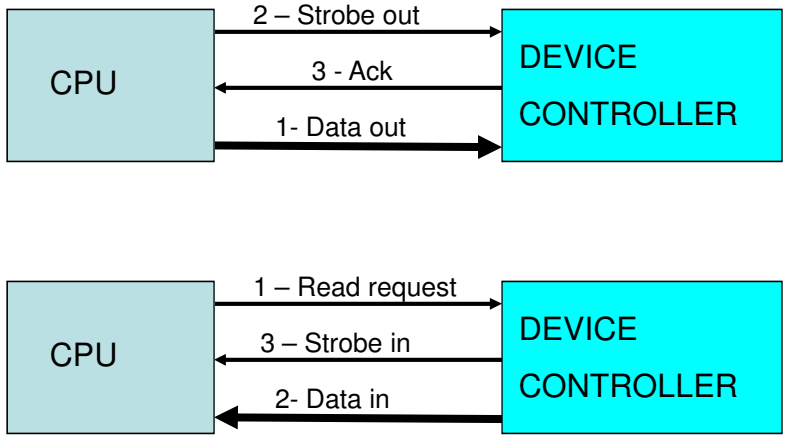
Protocollo di Ingresso

- Manda la Richiesta
- Aspetta un *Ack* (Strobe-In)
- Leggi il dato

Protocollo di Uscita

- Scrivi i dati
- Manda la Richiesta (Strobe-Out)
- Aspetta un *Ack*

Handshake: sequenza dei segnali



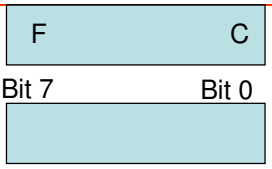
Esempio di driver per I/O programmato

```
STRT  Inizializza
POLL  Testa lo stato
      BEQ POLL
      Leggi Il Dato
POLL2 Testa lo stato
      BEQ POLL
      Scrivi il Dato
      (Deve continuare?)
      BEQ POLL
```

Lo schema riprodotto mostra un esempio di comunicazione con un terminale che permette di leggere un carattere e rimetterlo a video con un meccanismo di controllo dello I/O da programma.

I/O controllato da programma: output

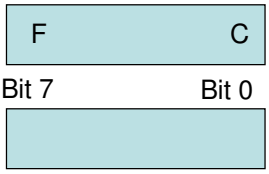
```
DataRegister   db;
ControlBits    F,C;

db.write ();
C.set ();

do {

}
while ( F.isReset () )

C.reset ();
F.reset ();
```




```
                ORG     $8000
START           MOVE.B  D0,IF_DATA
                ORI.B   #$01,IF_CTRL
LOOP            BTST.B  #7,IF_CTRL
                BEQ     LOOP
                MOVE.B  #0,IF_CTRL

                ORG     $8020
IF_CTRL         DS.B    1
                ORG     $8022
IF_DATA         DS.B    1
```

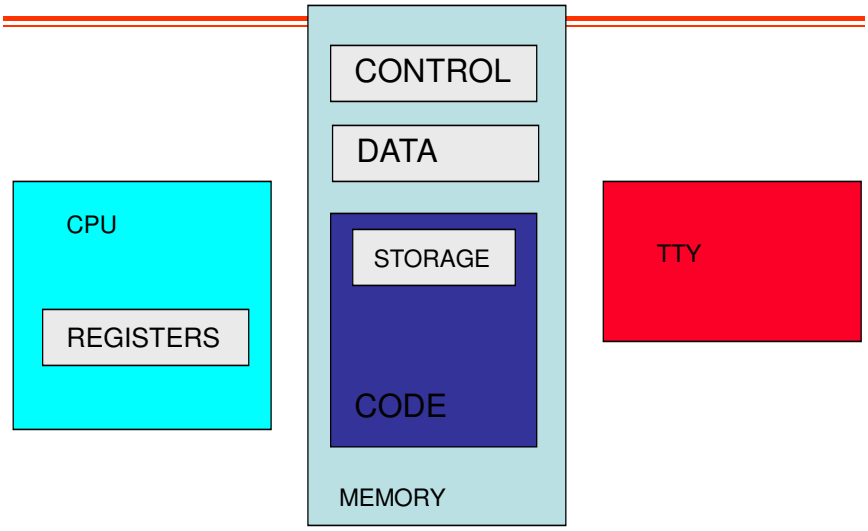
I/O controllato da programma: input

```
DataRegister   db;
ControlBits    F,C;

C.set ();
F.reset ();

do {

}
while ( F.isReset () )

db.read ();
C.reset ();
```




```
                ORG     $8000
START           MOVE.B  #1,IF_CTRL
LOOP            BTST.B  #7,IF_CTRL
                BEQ     LOOP
                MOVE.B  IF_DATA,D0
                AND.B #$FE,IF_CTRL

                ORG     $8020
IF_CTRL         DS.B    1
                ORG     $8022
IF_DATA         DS.B    1
```

Esempio di I/O programmato



Terminal.a68 – 1/3

```
* terminal.a68
* Demonstrates programmed I/O

MEM    EQU $8000

* Leave half K free for later use (data storage)
        ORG $8200

TER       EQU $2000     * Address of tty data register
EOS       EQU 0
ENTER     EQU $D
ENTERHIT  EQU $80 * 10000000
TTYCFG    EQU $30 * 00110000 Abilita tastiera, Abilita eco

QUEST     DC.B     'Come ti chiami ? ',0
ANSW      DC.B     'Ciao, ',0
```

Terminal.a68 – 2/3

```
* Initialize A0 to tty data register
* Initialize A2 to tty control register
* Initialize control register to 00110000
BEGIN   MOVEA.L #TER,A2
        MOVEA.L A2,A0
        ADD.L   #1,A2
        MOVE.B  #TTYCFG,(A2)
* Initialize A1 to start of output message
* Output characters until EOS found
        LEA.L   QUEST,A1
OUTPUT  MOVE.B  (A1)+,D0
        CMP.B   #EOS,D0
        BEQ     CONT
        MOVE.B  D0,(A0)
        BCC     OUTPUT
* Keep checking tty control register until ENTER hit
CONT    MOVE.B  (A2),D1
        AND.B   #ENTERHIT,D1
        BEQ     CONT
```

Terminal.a68 – 3/3

```
* Initialize A1 to start of storage area
        MOVEA.L   #MEM,A1
* Copy characters from tty data register to storage area
* Stop when ENTER is found in data
IN1     MOVE.B  (A0),D0
        MOVE.B  D0,(A1)+
        CMP.B   #ENTER,D0
        BNE     IN1
* Initialize A1 to start of output message
* Output characters until EOS found
        LEA.L   ANSW,A1
OUT1    MOVE.B  (A1)+,D0
        CMP.B   #EOS,D0
        BEQ     CONT1
        MOVE.B  D0,(A0)
        BCC     OUT1
* Initialize A1 to start of storage area
* Output characters until ENTER found
     …
```

I/O sincronizzato con interrupt

- È basato su un segnale asincrono che il dispositivo invia alla CPU quando ha bisogno di un servizio.
- In questo modo:
  - il tempo per ottenere l'attenzione della CPU si riduce
  - la CPU non perde tempo a scandire in polling il dispositivo per testarne lo stato.

I/O sincronizzato con interrupt (2)

- La gestione del servizio dei dispositivi di I/O tramite *interrupt* permette un migliore efficienza dell'esecuzione della elaborazione della CPU.
- Un input esterno *asincrono* informa il microprocessore che un dispositivo esterno richiede di essere servito. La CPU interrompe l'esecuzione del programma corrente e salta all'esecuzione di una *procedura di servizio* dell'interruzione (*Interrupt Service Routine - ISR*).
- Il dispositivo di controllo dell'interrupt funziona come un gestore delle richieste di interruzione tra i dispositivi periferici e la CPU.

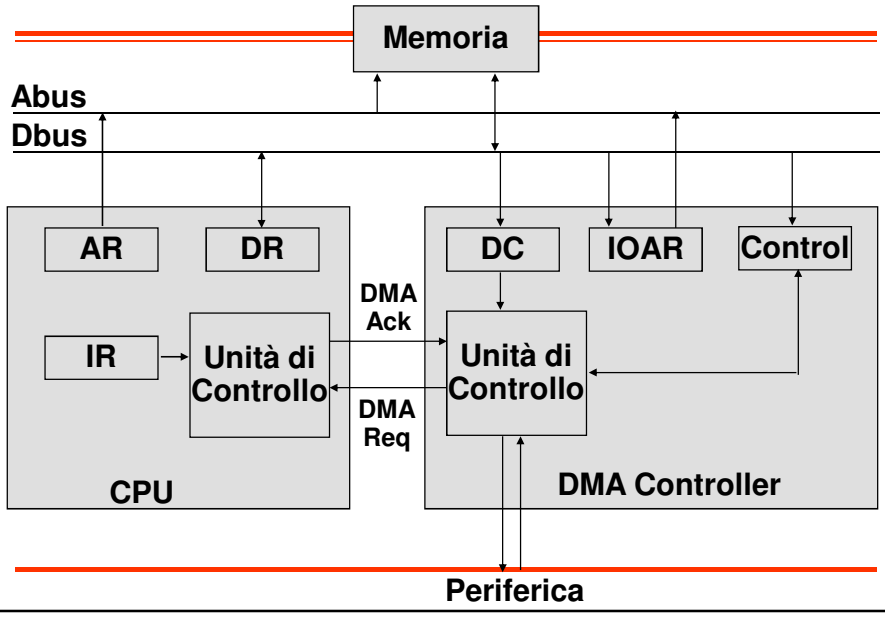
Driver con interruzioni

- Si sviluppa la procedura ISR
- La si carica in memoria all'indirizzo predefinito
- Si associa la procedura al suo codice e lo si fornisce al dispositivo.

DMA

- È il metodo preferito quando si devono trasferire grosse moli di dati.
- Una circuiteria apposita (*DMA Controller*) provvede ad eseguire il trasferimento di dati da una periferica alla memoria (o viceversa).
- La circuiteria deve essere in grado di fungere da *bus master*, ossia deve generare gli indirizzi ed i segnali di controllo secondo la tempistica opportuna.
- Il DMA Controller deve inoltre essere in grado di negoziare con la CPU l'acquisizione del controllo del bus ed il suo rilascio.

Circuiteria per il DMA



Canali di DMA

- Ciascun DMA controller può normalmente gestire più periferiche (*canali*). Per ciascuna periferica il DMA controller dispone di
  - Linee di richiesta e di acknowledge
  - Registri DC e IOAR
  - Circuiteria di controllo.
- In tal caso il DMA controller è anche in grado di arbitrare tra richieste di DMA contemporanee.

Trasferimento in DMA

- Il trasferimento di un blocco in DMA si articola in varie fasi:
  - la CPU carica nei registri IOAR e DC l'indirizzo dell'area di memoria ed il numero di parole da trasferire; la CPU informa inoltre il DMA controller della direzione del trasferimento (memoria → periferica o viceversa);
  - il DMA Controller riceve una richiesta di trasferimento da parte di una periferica;
  - Il DMA Controller invia un segnale di DMA Request alla CPU;
  - quando la CPU giunge ad un punto di rilevamento del segnale di DMA Request, rilascia il bus e attiva il segnale di DMA Acknowledge;
  - il DMA Controller inizia il trasferimento; dopo il trasferimento di ciascuna parola, IOAR e DC vengono aggiornati;

DMA: trasferimento a blocchi

- Prevede che il DMA Controller, una volta acquisito il controllo del bus, lo mantenga per tutto il tempo richiesto per trasferire un blocco di dati.
- In tal modo il trasferimento avviene alla massima velocità mala CPU è bloccata per tutta la durata del trasferimento.
- Il trasferimento a blocchi è importante per periferiche quali i dischi magnetici, dove il trasferimento del blocco non può essere interrotto).

Trasferimento con *Cycle Stealing*

- Il DMA Controller trasferisce i dati in piccoli blocchi, occupando il bus per periodi limitati di tempo.
- In tal modo:
  - la velocità di trasferimento è minore
- ma
  - la CPU non è bloccata per periodi troppo lunghi.

Trasferimento in *Transparent DMA*

- Il DMA Controller è in grado di rilevare quando la CPU non utilizza il bus, e solo in quei periodi esegue il trasferimento dei dati.
- In tal modo la CPU non è praticamente rallentata dal DMA Controller.