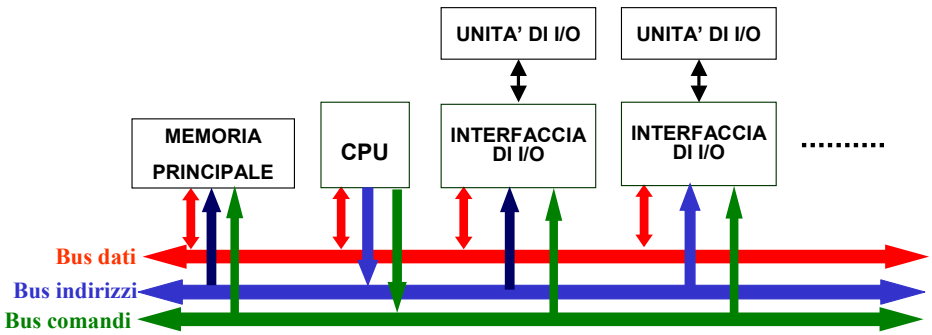
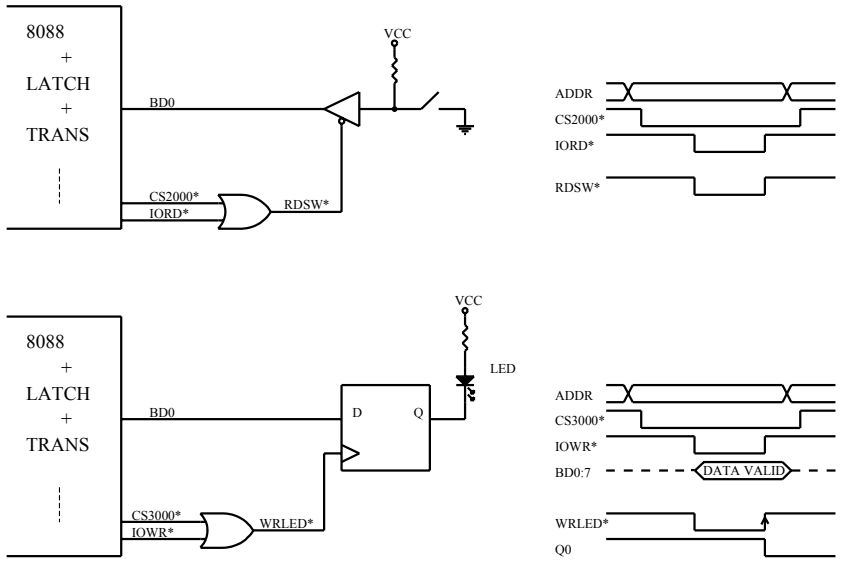
Il sottosistema di I/O

• Il sottosistema di I/O consente la comunicazione fra il calcolatore ed il mondo esterno. Fanno parte del sottosistema i dispositivi (*Unità di I/O*) per la comunicazione uomo/macchina (video, terminali, stampanti), quelli utilizzati per la memorizzazione permanente delle informazioni (unità a disco, nastri magnetici), la rete.

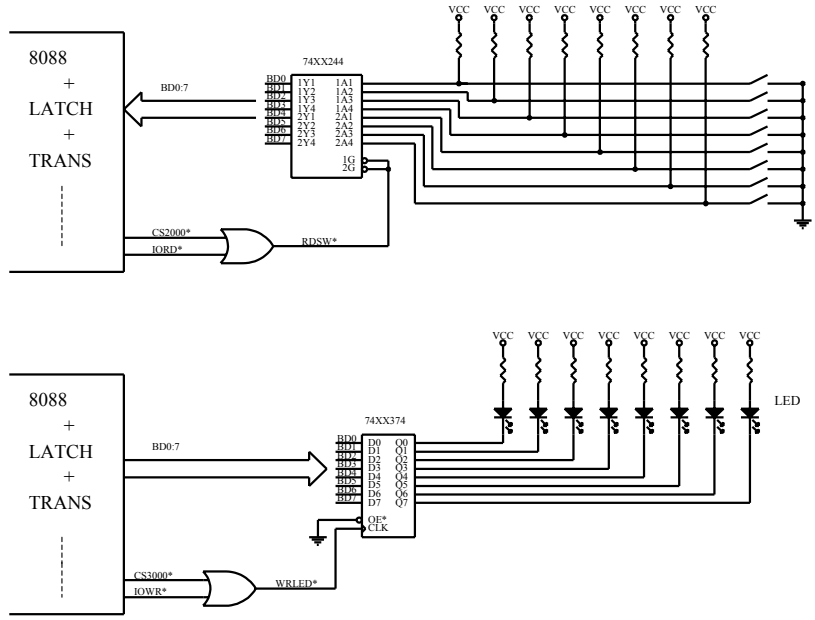
• Tutte le Unità di I/O sono collegate al bus di sistema mediante dispositivi detti ***Interfacce di I/O*** (o anche *Periferiche*, *Controllori di I/O*, *Porte di I/O*)



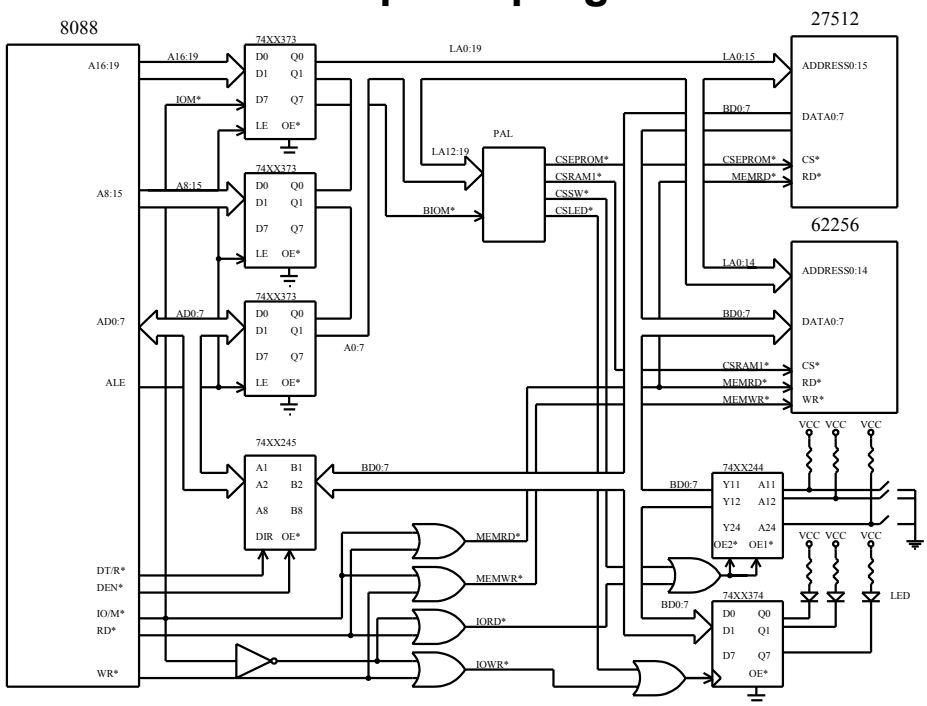
Input Output digitale



Input Output digitale

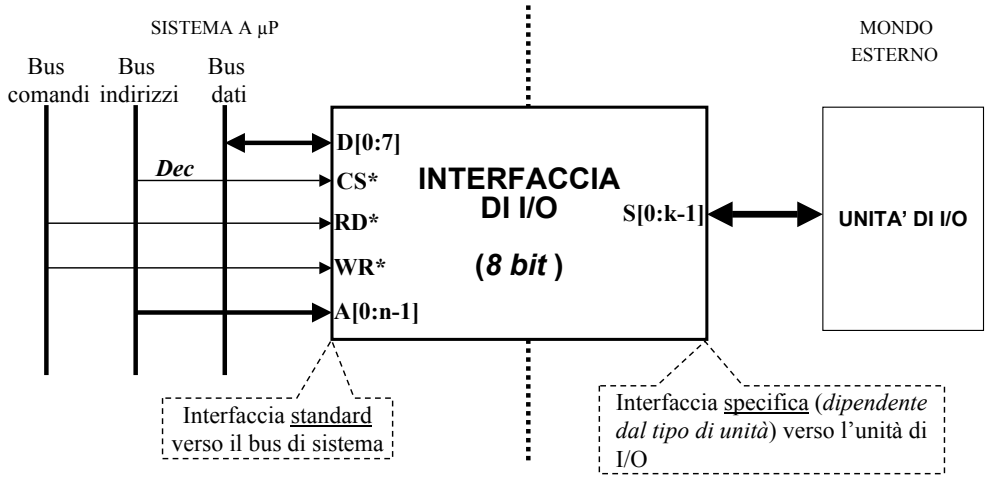


Esempio di progetto



Intro I/O 4

Interfacce di I/O (1)



• Grazie a questa strutturazione l'interfaccia svolge una funzione di *adattamento* fra la modalità di trasferimento dei dati utilizzata all'interno del sistema (cicli di bus) e quella utilizzata dall'Unità di I/O.

Interfacce di I/O (2)

• Buffer Dati

Strettamente associata alla funzionalità di *adattamento* fra calcolatore ed unità di I/O è la presenza all'interno dell'interfaccia di *registri di appoggio* ("*buffer*") utilizzati nei trasferimenti dei dati da CPU ad Unità di I/O e viceversa.

Per inviare un dato all'Unità di I/O la CPU effettua una scrittura del dato su un buffer dell'interfaccia, da cui poi quest'ultima si occupa di trasferire il dato all'Unità di I/O.

Per prelevare un dato dall'Unità di I/O la CPU effettua una lettura da un buffer dell'interfaccia, su cui quest'ultima ha precedentemente appoggiato il dato proveniente dall'Unità di I/O.

• Programmazione e Registri di Controllo

Le interfacce di I/O devono poter funzionare secondo un certo insieme di modalità differenti (es.: un'interfaccia per comunicazioni seriali asincrone RS232 deve poter scambiare dati con un modem impiegando trame e frequenze differenti).
Conseguentemente le interfacce di I/O sono tipicamente *programmabili*: sono presenti cioè al loro interno un certo numero di *registri di controllo* che vengono scritti dalla CPU all'atto dell'inizializzazione del dispositivo per impostare la modalità di funzionamento desiderata.

Interfacce di I/O (3)

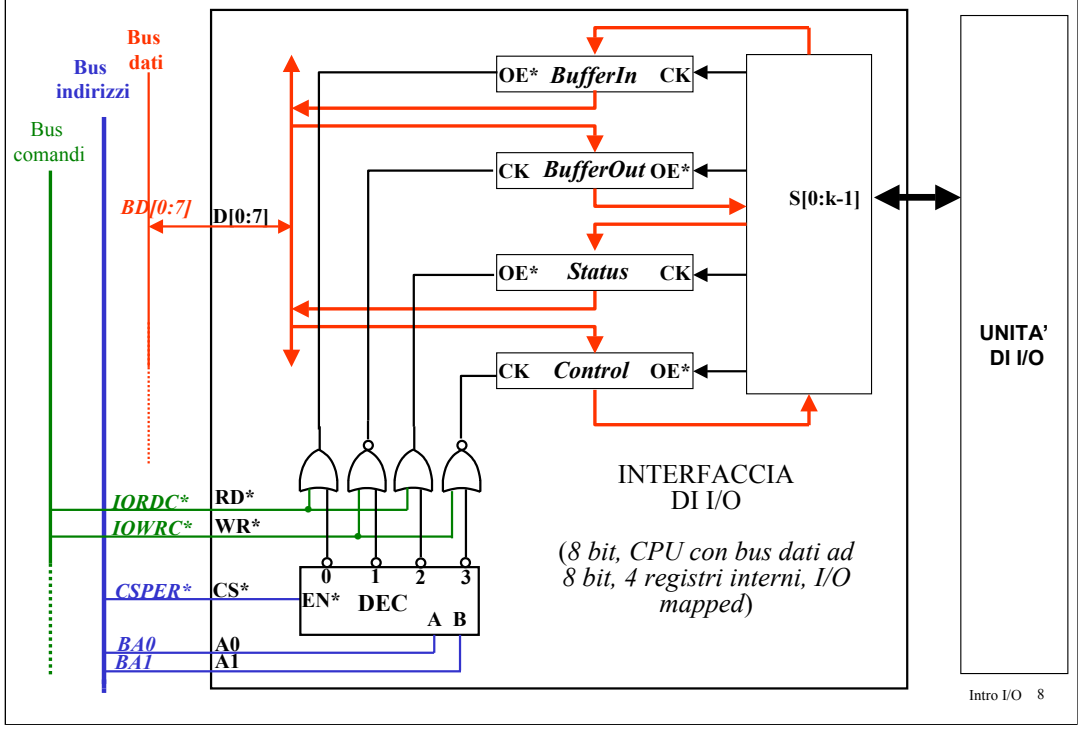
• **Sincronizzazione e Registri di Stato**

Di norma le Unità di I/O lavorano in modo asincrono rispetto al funzionamento della CPU e sono molto più lente di quest'ultima .
Si rende quindi necessario introdurre all'interno dell'interfaccia di I/O un qualche meccanismo di ***sincronizzazione*** fra l'attività svolta dalla CPU e quella svolta dalla Unità di I/O.

Tipicamente le interfacce di I/O dispongono di ***registri di stato*** tramite cui vengono rese disponibili alla CPU tutte le informazioni necessarie per la sincronizzazione con l'Unità di I/O.

In aggiunta a ciò, i registri di stato situati all'interno di una interfaccia di I/O sono spesso utilizzati per segnalare alla CPU il verificarsi di eventuali condizioni di malfunzionamento o errore (es.: errore di parità nelle comunicazioni seriali asincrone RS232).

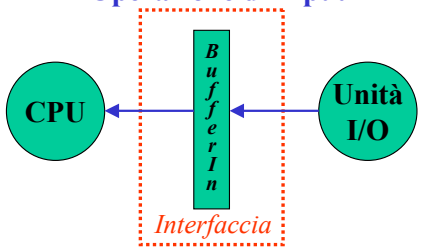
Struttura di una generica interfaccia di I/O



Le modalità di gestione dell'I/O

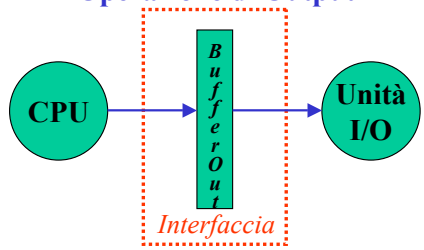
• Affrontiamo ora in modo più approfondito la problematica della sincronizzazione fra CPU (*programma in esecuzione sulla CPU*) ed Unità di I/O. Come già osservato, **la funzionalità di sincronizzazione è a carico dell'interfaccia di I/O**.

Operazione di Input



L'interfaccia deve comunicare alla CPU che l'Unità di I/O ha reso disponibile un nuovo dato su *BufferIn* (condizione di "BufferIn Full")

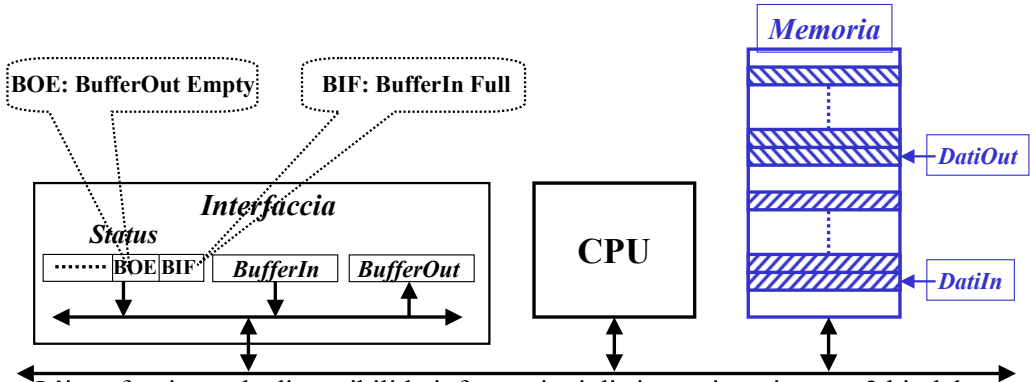
Operazione di Output



L'interfaccia deve comunicare alla CPU che l'Unità di I/O ha prelevato da *BufferOut* il dato precedentemente inviato dalla CPU e quindi la CPU può inviare un nuovo dato (condizione di "BufferOut Empty").

• Le interfacce possono comunicare alla CPU le informazioni di sincronizzazione in due modi diversi, cui corrispondono due differenti modalità di gestione dell'I/O dette **Gestione a Polling** (Controllo di Programma) e **Gestione ad Interrupt** (Interruzione).

Gestione a Polling



L'interfaccia rende disponibili le informazioni di sincronizzazione su 2 bit del registro di stato. La CPU, ogni qual volta deve eseguire un trasferimento, va prima a leggere il registro di stato (BIF oppure BOE) per verificare se l'interfaccia è pronta per il trasferimento. Se l'interfaccia è pronta la CPU esegue il trasferimento, viceversa legge in continuazione il registro di stato fino a che questo non segnala che l'interfaccia è pronta, quindi esegue il trasferimento.

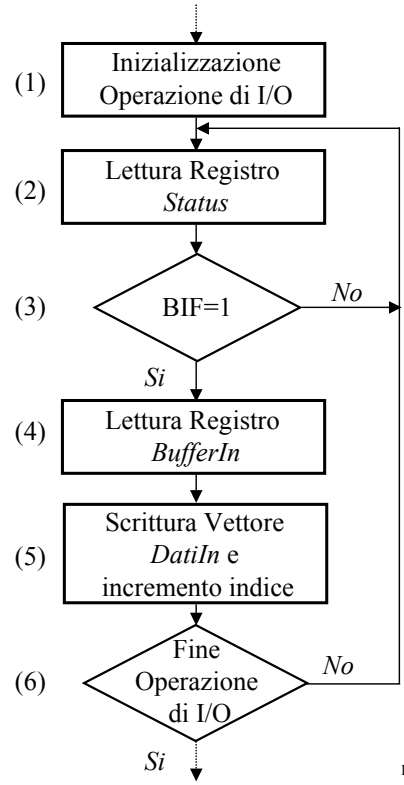
Il trasferimento del dato da parte della CPU (lettura di *BufferIn* o scrittura di (*BufferOut*) provoca il "CLEAR" del bit di sincronizzazione (BIF o BOE)

Intro I/O 10

Flusso di un programma che esegue una *Operazione di Input* a *Polling*.

Codifica in Assembler

```
(1)                MOV SI, 0
(2)  WaitDato:     IN AL, Status
(3)                TEST AL, 1
                   JZ WaitDato
(4)                IN AL, BufferIn
(5)                MOV DatiIn[SI], AL
                   INC SI
(6)                CMP SI, N
                   JNE WaitDato
```



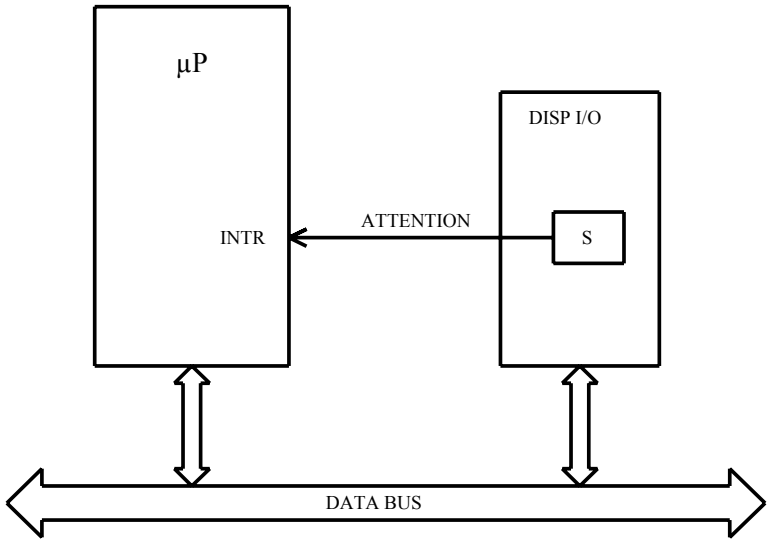
Intro I/O 11

Considerazioni sulla Gestione a Polling

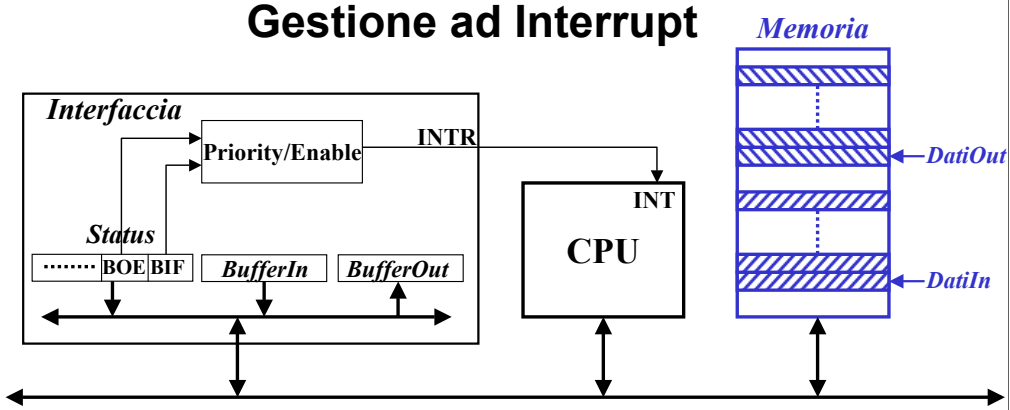
• Nella Gestione a polling la CPU è periodicamente impegnata ad osservare lo stato della periferica al fine di eseguire un trasferimento non appena quest'ultima risulta pronta. Conseguentemente, la CPU ha una scarsa efficianza se si verificano molti tentativi "a vuoto" di accedere alle periferiche che non abbiano ancora dati né da fornire né da chiedere. Più la periferica risulta "lenta" rispetto alla frequenza con la quale la CPU effettua il polling su di essa, più le prestazioni della CPU peggiorano. D'altra parte, abbassare la frequenza di polling significa degradare le prestazioni delle periferiche "lente" che sono costrette a lunghe attese prima di poter scambiare un dato dal momento in cui sono pronte, fintanto che la CPU non effetua il polling. Il polling risulta quindi adatto a periferiche in grado di sostenere accessi con una frequenza costante, elevata e simile a quella di polling.

• La Gestione ad Interrupt, che vedremo successivamente, differisce dal polling perché l'iniziativa per l'accesso spetta alla periferica che, quando è pronta, chiede alla CPU di effettuare il trasferimento. Per contro, rispetto al polling, un trasferimento dati fatto ad interrupt comporta un maggior numero di operazioni iniziali e finali (quindi tempi più lunghi) e va quindi usato solo se la frequenza delle richieste di interrupt è bassa rispetto a quella di CPU, altrimenti conviene il polling.

Gestione ad Interrupt

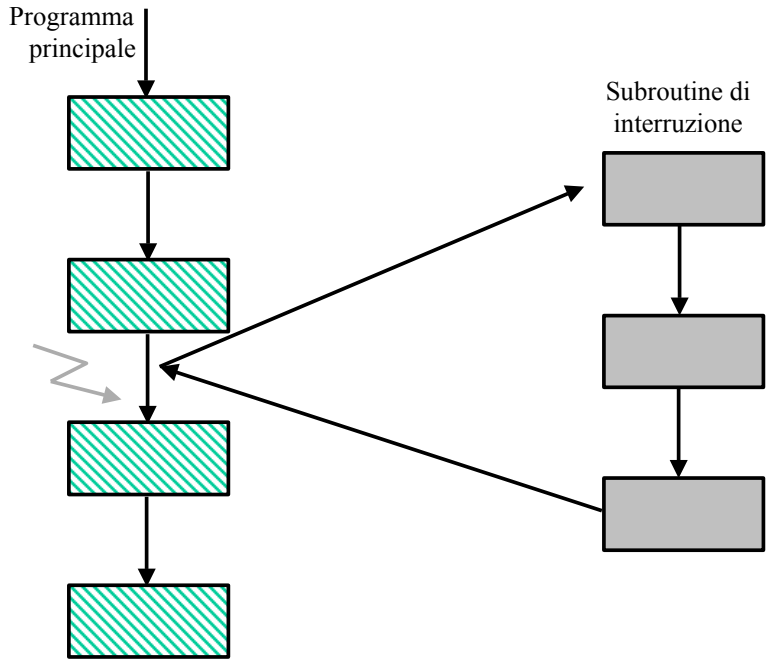


Gestione ad Interrupt

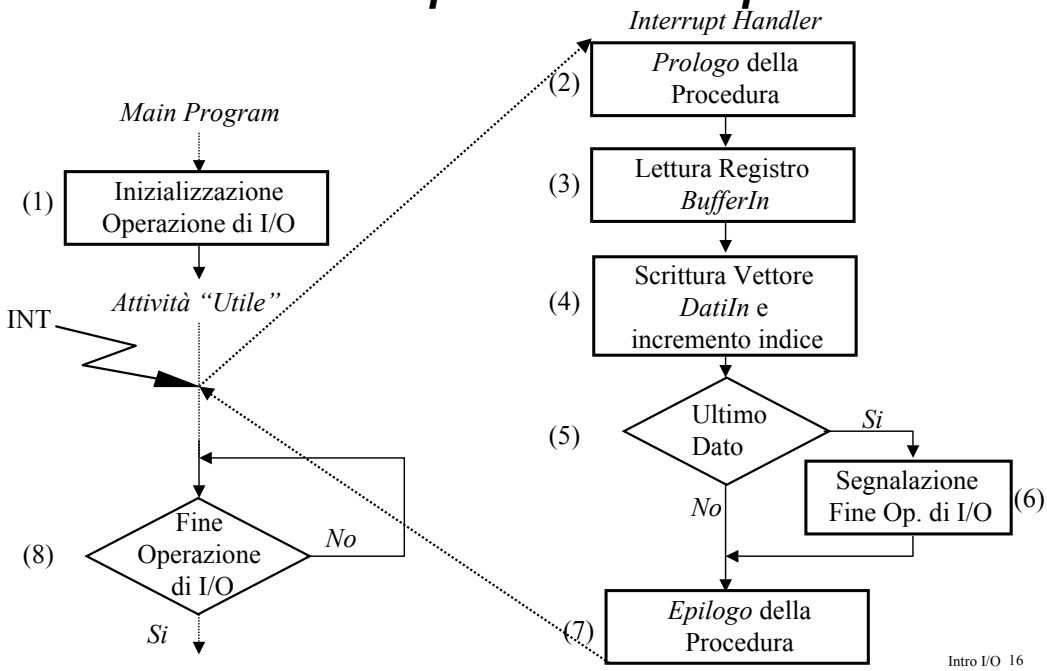


Quando l'interfaccia è pronta ad eseguire un nuovo trasferimento invia alla CPU una *richiesta di servizio* (detta *Interrupt*) mediante un apposito segnale (INTR, che per ora possiamo considerare connesso all'ingresso INT della CPU). All'atto della ricezione della *richiesta di servizio* la CPU *interrompe* il programma in esecuzione e trasferisce il controllo (secondo un meccanismo che studieremo successivamente) ad una apposita *procedura di servizio* (detta *Interrupt Handler*) che si occupa di effettuare il trasferimento del dato.
Una volta effettuato il trasferimento la procedura di servizio restituisce il controllo al programma che era in esecuzione all'atto della ricezione dell'Interrupt.
Nella Gestione ad Interrupt la CPU può eseguire qualsiasi attività durante gli intervalli di tempo che intercorrono fra due situazioni successive di "interfaccia pronta" (cioè fra due interrupt).

Flusso della gestione ad Interrupt

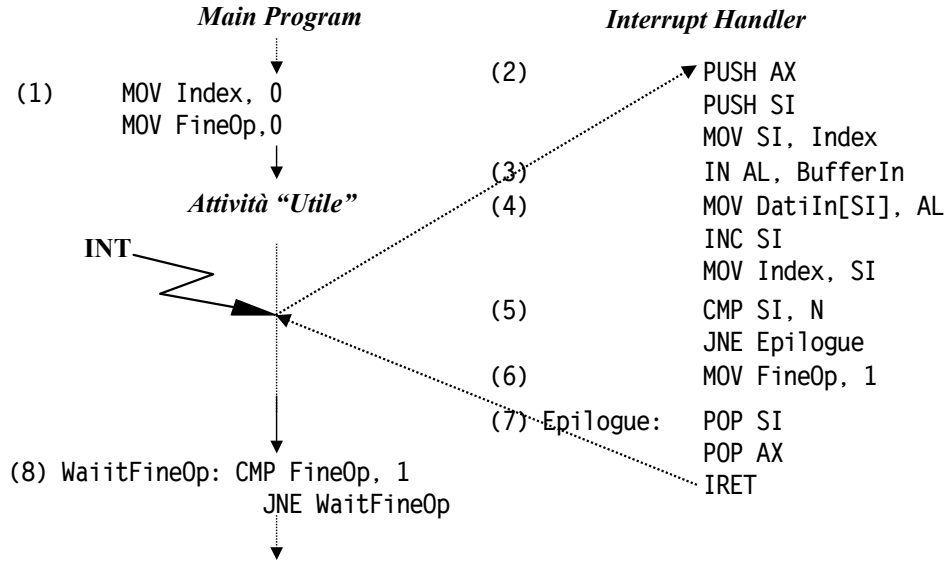


Flusso della gestione ad Interrupt di una *Operazione di Input*

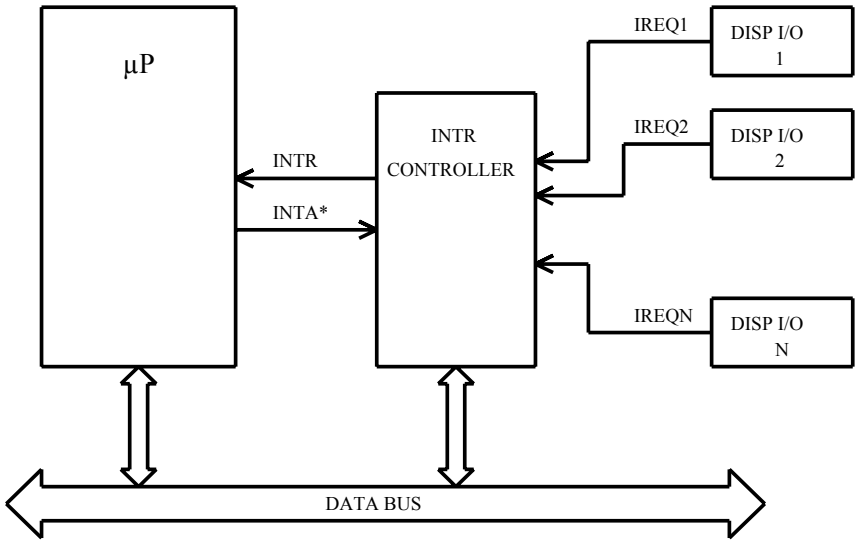


Intro I/O 16

Intr: codifica in Assembler



Interrupt Controller



Considerazioni sulla Gestione a Interrupt

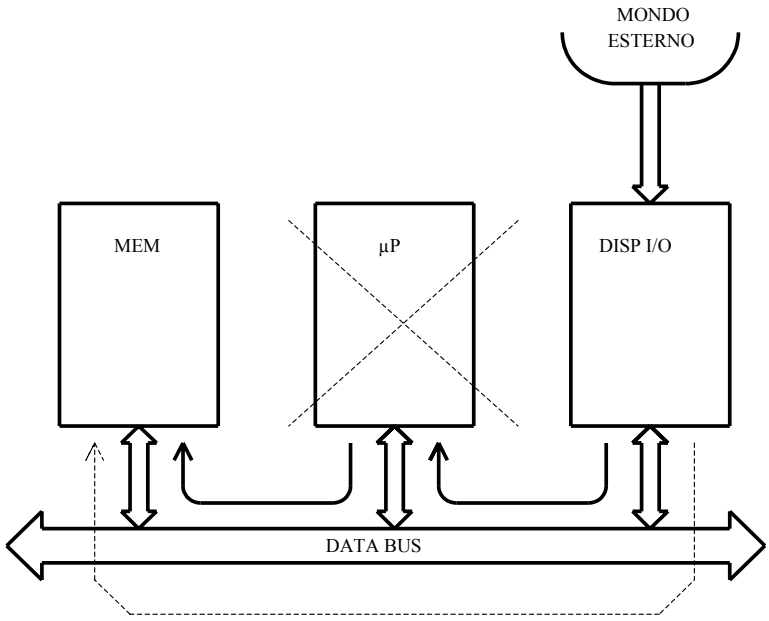
• La Gestione ad Interrupt consente un utilizzo efficiente della CPU: la CPU può essere impegnata in altre attività durante i "tempi morti" che intercorrono fra gli istanti in cui l'interfaccia è pronta ad eseguire un nuovo trasferimento.

• E' opportuno sottolineare che se l'Operazione di I/O richiede il trasferimento di *N* dati l'interfaccia genererà *N richieste di servizio* (cioè *N Interrupt*) e conseguentemente l'*Interrupt Handler* verrà eseguito *N* volte. Ad ogni esecuzione l'*Interrupt Handler* trasferisce un solo dato.

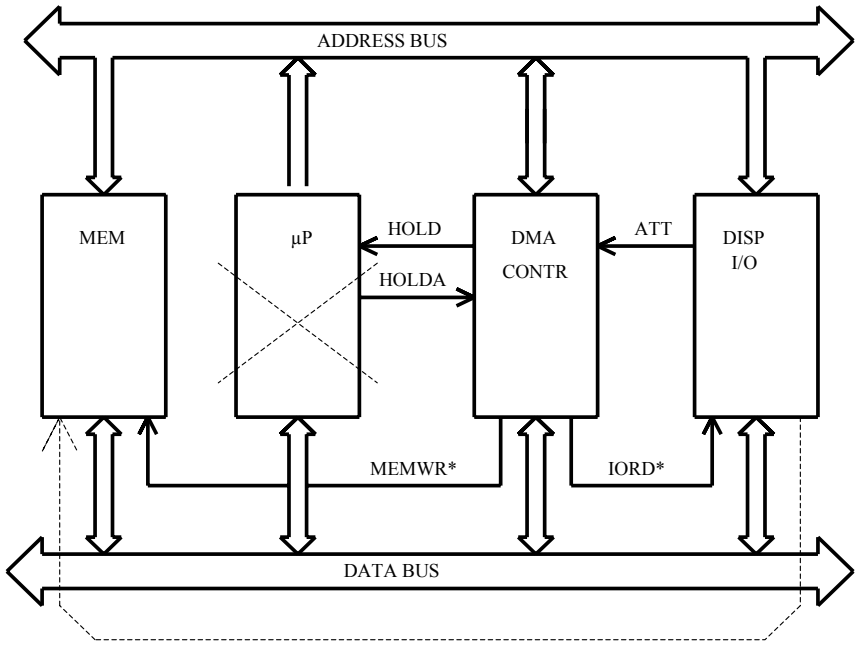
• Poiché i trasferimenti sono eseguiti dall'*Interrupt Handler*, è necessario prevedere un meccanismo di sincronizzazione fra quest'ultimo ed il *Main Program*. Nell'esempio mostrato la sincronizzazione è effettuata tramite la variabile *FineOp*, azzerata dal Main Program nella fase di inizializzazione e messa a 1 dall'Interrupt Handler quando ha completato l'Operazione di I/O.

•La Gestione ad Interrupt è particolarmente adatta ai *Sistemi Multitasking*: un *task* (programma), dopo aver "lanciato" l'Operazione di I/O viene "sospeso" dal *Sistema Operativo* e la CPU viene assegnata ad un altro *task*. Il *task* sospeso viene poi rimesso in esecuzione quando l'Operazione di I/O è stata completata dall'*Interrupt Handler* (che fa parte del *kernel* del *Sistema Operativo*).

Gestione in DMA

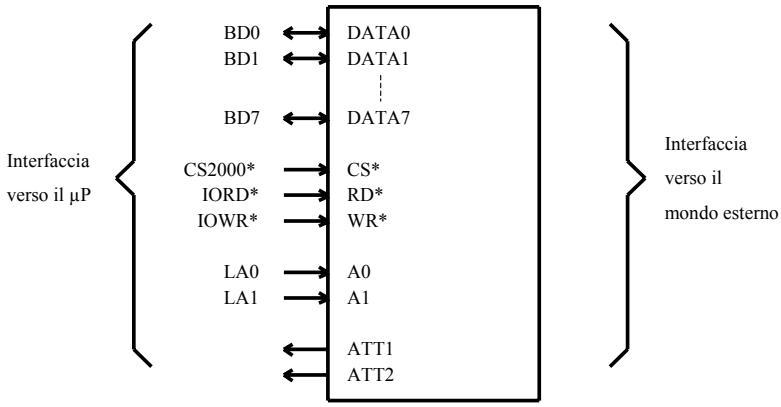


DMA Controller

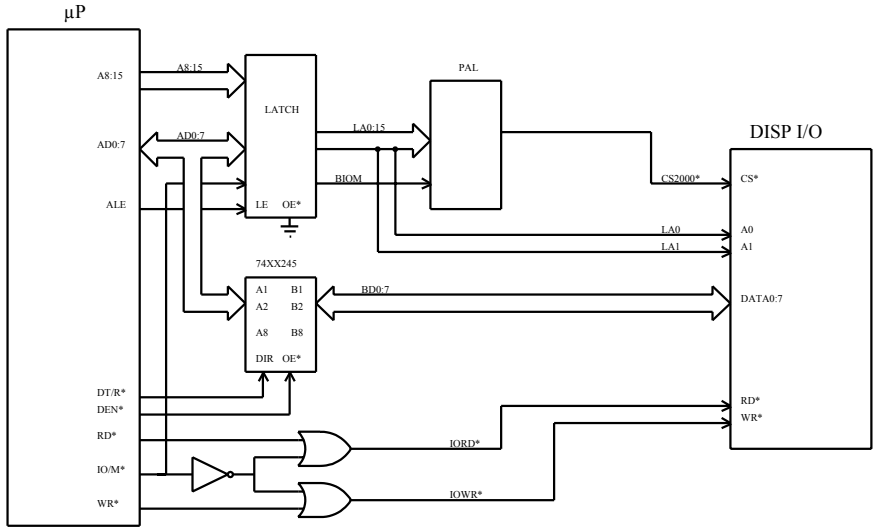


Intro I/O 21

Generico dispositivo di IO

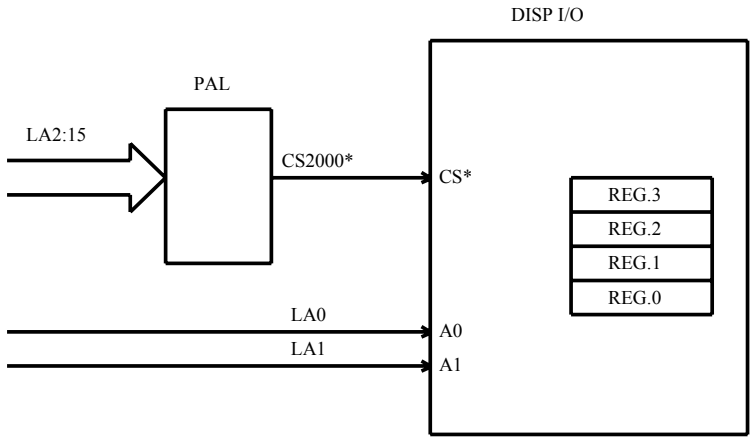


Disp I/O - Collegamento al µP



Intro I/O 23

Disp I/O - Registri interni selezionati con le sole linee di indirizzo (1)



Accesso ai registri interni dell'interfaccia con selezione dell'indirizzo (2)

Hyp. (1) : CSPER = $\overline{A15}\cdot\overline{A14}\cdot\overline{A13}\cdot\overline{A12}\cdot\overline{A11}\cdot\overline{A10}\cdot\overline{A9}\cdot\overline{A8}\cdot$**A7**$\cdot\overline{A6}\cdot\overline{A5}\cdot\overline{A4}\cdot\overline{A3}\cdot\overline{A2}$

La CPU vede i 4 registri interni dell'interfaccia agli indirizzi:
BufferIn: 80H (128), *BufferOut*: 81H (129), *Status*: 82H (130), *Control*: 83H (131)

Hyp. (2) : La configurazione binaria (*Control Word*) che è necessario scrivere nel registro di controllo per far si che l'interfaccia funzioni nel modo desiderato è *11010100 (D4H)*.

Un programmatore che deve scrivere un programma Assembler (IA16) che ha la necessità di accedere ai registri dell'interfaccia utilizzerà (tipicamente) la direttiva EQU per definire degli identificatori associati alle costanti che rappresentano gli indirizzi dei registri e la Control Word:


```
BufferIn          EQU 80H
BufferOut         EQU 81H
StatusRegister    EQU 82H
ControlRegister   EQU 83H
ControlWord       EQU D4H
```


Intro I/O 25

Accesso ai registri interni dell'interfaccia con selezione dell'indirizzo (3)

• Programmazione dell'interfaccia

Nella porzione di programma che si occupa di inizializzare tutte le periferiche programmabili presenti nel sistema il programmatore inserirà le seguenti istruzioni:

```
                ⋮
MOV AL, ControlWord      ; caricamento in AL della costante "ControlWord"
OUT ControlRegister, AL  ; trasferimento del contenuto di AL nel registro
                         ; mappato all'indirizzo "ControlRegister" dello spazio di I/O
                ⋮
```

• Lettura dello stato

Ogni qual volta il programma ha la necessità di leggere lo stato della periferica (ad esempio per controllare se si sono verificati degli errori oppure, *come vedremo meglio in seguito*, per sincronizzare il programma con l'Unità di I/O connessa al sistema tramite l'interfaccia) il programmatore inserirà la seguente istruzione:

```
                ⋮
IN AL, StatusRegister    ; caricamento in AL del contenuto del registro mappato
                         ; all'indirizzo "StatusRegister"dello spazio di I/O
                ⋮
```

Accesso ai registri interni dell'interfaccia con selezione dell'indirizzo (4)

• Trasferimento di dati (*Input*)
Supponiamo che il programma debba ricevere una sequenza di dati dall'Unità di I/O e memorizzarli nel vettore *DatiIn* (*Operazione di Input*). Supponiamo inoltre che il programma sia opportunamente sincronizzato con l'Unità di I/O (il programma va a leggere un dato dall'interfaccia quando questo è stato effettivamente reso disponibile in *BufferIn* dall'Unità di I/O). In queste ipotesi il programmatore inserirà le seguenti istruzioni al fine di prelevare l' *i*-esimo dato della sequenza:

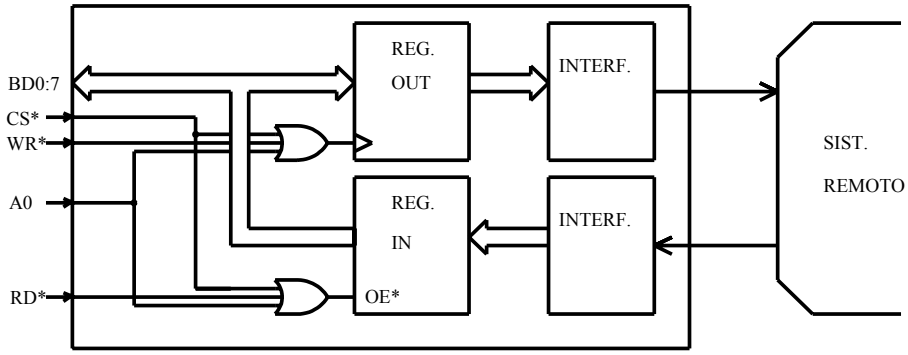
```
                .
                .
                .
IN AL, BufferIn         ; trasferimento in AL del contenuto del registro
                        ; mappato all'indirizzo "BufferIn dello spazio di I/O
MOV DatiIn[SI], AL      ; trasferimento del contenuto di AL nella cella di indice SI del vettore DatiIn. Se il programma
                        ; sta prelevando l'i-esimo dato SI, che è stato opportunamente inizializzato, contiene il valore i-1
INC SI                  ; incremento dell'indice del vettore, che così punta alla cella di memoria in cui deve essere
                .       ; inserito il prossimo dato prelevato dall'interfaccia
                .
                .
```

Accesso ai registri interni dell'interfaccia con selezione dell'indirizzo (5)

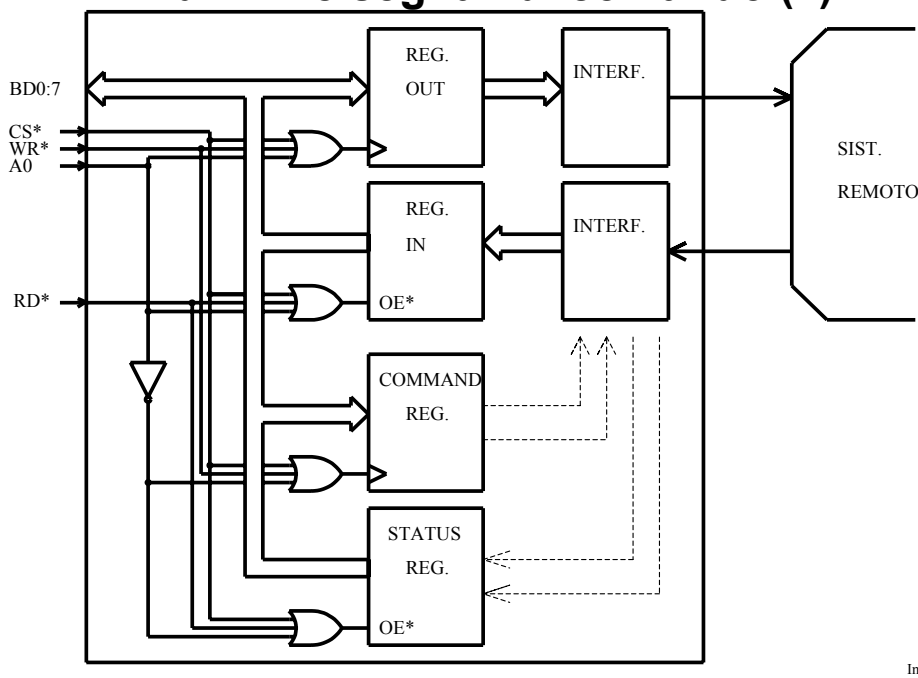
• Trasferimento di dati (*Output*)
Supponiamo che il programma debba inviare una sequenza di dati memorizzati nel vettore *DatiOut* all'Unità di I/O (*Operazione di Output*). Supponiamo inoltre che il programma sia opportunamente sincronizzato con l'Unità di I/O (il programma va a scrivere un dato sul registro *BufferOut* dall'interfaccia quando questo registro è "vuoto", cioè quando l'Unità di I/O ha ricevuto il dato inviato precedentemente). In queste ipotesi il programmatore inserirà le seguenti istruzioni al fine di inviare l' *i*-esimo dato della sequenza:

```
        ⋮
MOV AL, DatiOut[SI]   ; trasferimento in AL del contenuto della cella di indice SI del vettore
                      ; DatiOut. Se il programma sta inviando l'i-esimo dato SI, che è stato
                      ; opportunamente inizializzato, contiene il valore i-1 OUT BufferOut, AL
                      ; trasferimento del contenuto di AL nel registro mappato all'indirizzo
                      ; "BufferOut" dello spazio di I/O.
INC SI                ; incremento dell'indice del vettore, che così punta alla cella di memoria
        ⋮             ; da cui deve essere prelevato il prossimo dato da inviare all'interfaccia
```

Disp I/O - Registri interni selezionati con indirizzi e segnali di comando (1)

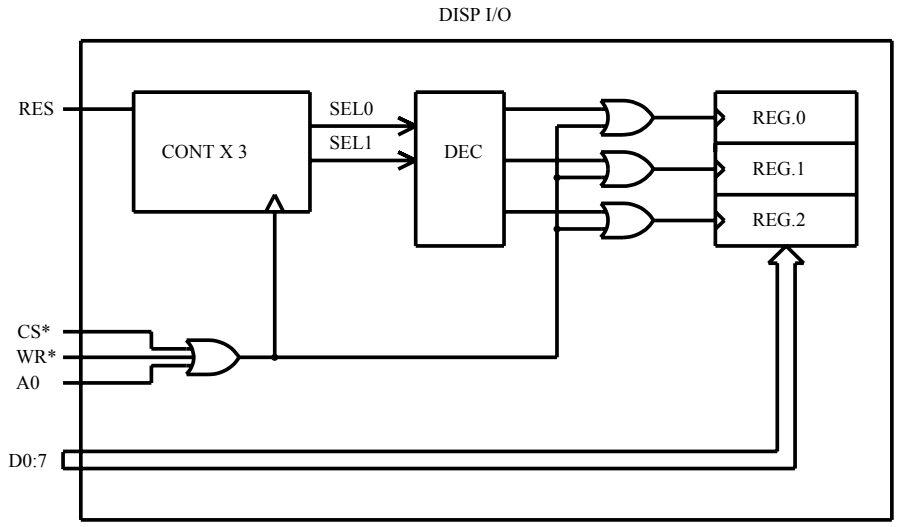


Disp I/O - Registri interni selezionati con indirizzi e segnali di comando (2)



Intro I/O 30

Disp I/O - Registri interni selezionati con decodifica sequenziale



Intro I/O 31

Disp I/O - Registri interni selezionati con decodifica indiretta