

# MEMORIE

Le memorie possono essere classificate secondo diversi criteri.

**1° Criterio : tipo di supporto fisico.** Possiamo indicare tre diversi tipi :

## 1. Supporto magnetico.

Si tratta di un nastro sottile oppure di un dischetto di materiale plastico, ricoperti di uno strato sottilissimo di ossidi di materiali ferromagnetici ( **Fe** , **Ni** , **Co** ), sensibili perciò all'azione di campi magnetici.

Esempi commerciali sono le audio e le video cassette , i floppy disk , i mini disk , gli hard disk.

L'informazione viene registrata in forma analogica nelle cassette , in forma digitale nei dischetti.

( Sono esistite per alcuni anni anche le cassette digitali, poi soppiantate dai CD )

In fase di **registrazione o scrittura**, nella testina di scrittura una corrente elettrica proporzionale al segnale sonoro o visivo ,corrente creata da opportuni **trasduttori ( microfoni o telecamere o fotocamere )** , crea un campo magnetico che magnetizza, cioè **orienta** in un determinato modo le micro particelle ferrose deposte sul nastro.

In fase di **lettura**, il nastro registrato, passando sotto la testina di lettura, vi **induce**, cioè vi fa nascere, una corrente assai simile a quella di scrittura e tale corrente, opportunamente amplificata ed elaborata , inviata a degli **attuatori ( altoparlanti o schermi LCD o televisori )** , ricreerà il suono o il video originali.

## 2. Supporto plastico ( per memorie ottiche )

Un disco di materiale plastico di una decina di cm di diametro e spesso circa un mm, il CD o il DVD, viene inciso in fase di scrittura da un raggio **laser** abbastanza intenso da fondere il supporto per alcune decine di micron, realizzando così piccolissimi pozzi, fino a raggiungere lo strato sottostante altamente riflettente.

I pozzi rappresentano gli **uni**, le zone del disco non incise rappresentano gli **zeri** ; **0 e 1** sono incisi lungo una spirale, dall'esterno verso l'interno del disco. Ovviamente l'informazione originale **analogica** (testi,suoni,immagini,video) deve essere **digitalizzata** , prima di essere registrata .

In fase di lettura, un diodo laser alloggiato sotto il cd emette un raggio luminoso che va a colpirne la superficie e, se entra in un pozzo, viene riflesso totalmente, a differenza dell'impulso luminoso che incontrando la superficie non incisa viene disperso lungo di essa.

I raggi riflessi vengono captati da un **fotodiodo** che reagisce alla maggiore o minore illuminazione fornendo una corrente rispettivamente grande ( **1** ) o piccola ( **0** ) .

## 3. Supporto semiconduttore .

E' il caso delle memorie integrate , cioè realizzate con le tecnologie di miniaturizzazione , drogaggio , etc, nei sottili chip di Silicio . **La singola cella di memoria è un latch o un flip-flop** , la locazione standard è costituita invece da **8 FF** , per memorizzare **1 byte** .Ovviamente l'informazione è esclusivamente digitale.

## 2° Criterio : mantenimento dei dati .

### • **Memorie non volatili** :

L'informazione permane per " sempre " sul supporto .

E' il caso delle memorie magnetiche, ottiche e , nel campo di quelle a semiconduttore, delle **ROM** , **PROM** , **EPROM** , **EEPROM** .

### • **Memorie volatili** :

L'informazione viene mantenuta solo finchè c'è l'alimentazione. E' il caso delle **RAM** . Queste si suddividono a loro volta in **Statiche e Dinamiche**.

Le **statiche**, finchè c'è l' alimentazione , mantengono perfettamente i dati , mentre le **dinamiche** , anche se alimentate, li perdono rapidamente .

Affinchè ciò non accada devono essere ciclicamente "**rinfrescate**", cioè i dati devono essere riconfermati ( ogni **2 [ms]** , tipicamente ) .

Altri criteri di classificazione possono essere: la **velocità** di accesso ai dati , il tipo di accesso (**casuale o sequenziale**), il tipo di segnale usato per la scrittura e la lettura ( **elettrico o ottico** ), le **dimensioni** fisiche, la **riscrivibilità**, il **costo**...

## ORGANIZZAZIONE INTERNA DELLE MEMORIE A SEMICONDUETTORE

Una RAM ( o ROM, PROM, EPROM, etc. ) contiene al suo interno un certo numero di locazioni di memoria, ciascuna costituita, in genere, da 8 flip-flop. Il numero di locazioni è sempre una potenza di 2 (  $2^N$  ), in tal modo ciascuna locazione è univocamente identificata da una combinazione binaria di un corrispondente numero **N** di bit, cioè ha un **indirizzo**.

**1° esempio** : una Memoria da **1 Kbyte** , cioè da **1024** byte, necessita di **10** bit di indirizzo (  $2^{10} = 1024$  ) e la prima locazione avrà indirizzo **000000000**, l'ultima **111111111** . In **HEX** si avrà rispettivamente **000 H** e **3FF H** .

**2° esempio** : se invece la RAM è da **32 KB** , ci vorranno **15** bit , essendo  $2^{15} = 2^5 * 2^{10} = 32 \text{ KB}$  e gli indirizzi andranno da **000000000000000** a **111111111111111** , cioè da **0000 H** a **7FFF H** .

All'interno della Memoria le locazioni sono disposte secondo una **matrice** , **quadrata o rettangolare**, a seconda del **N°** di bit .

Nel 1° caso , con **1024** locazioni e **10** bit di indirizzo , la matrice avrà **32** colonne e **32** righe ( infatti  $32*32 = 1024$  ) . Queste 32 righe e 32 colonne sono connesse alle uscite di **2 decoder 5 x 32 ( 5 IN e 32 OUT )**.

Gli ingressi sono i bit di indirizzo. Le uscite sono attive basse . All'incrocio di ciascuna riga e di ciascuna colonna si trova una locazione : per potervi entrare bisogna che i bit presenti sulla riga e sulla colonna siano **0**.

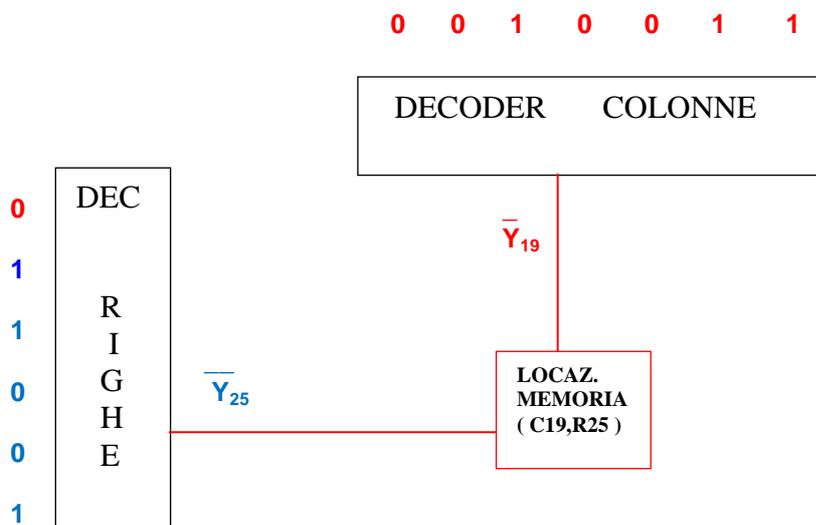
**Esempio : RAM da 8 Kbyte** :

vediamo a quale locazione corrisponde il seguente indirizzo :  $A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$   
0 0 1 0 0 1 1 0 1 1 0 0 1

Supponiamo di usare i bit più significativi per le colonne e i meno significativi per le righe :

Il decoder per le colonne attiverà l'uscita  $\overline{Y}_{19}$ , essendo  $(0010011)_2 = (19)_{10}$

mentre il decoder per le righe attiverà  $\overline{Y}_{25}$ , essendo  $(011001)_2 = (25)_{10}$



Le 2 uscite attivate ( sulle 2 linee c'è il bit 0 ) permettono di accedere alla locazione avente indirizzo (C19,R25) e di effettuare l'operazione indicata dall'istruzione ( lettura o scrittura ) .

In una scheda di memoria per un microprocessore , però , ci sono più RAM e/o ROM , per cui l'indirizzo presente nell'istruzione deve indicare , oltre alla locazione prescelta **dentro** la Ram, anche di quale Ram, fra tante, si tratta.

**Quante ce ne possono stare , sulla scheda ?**

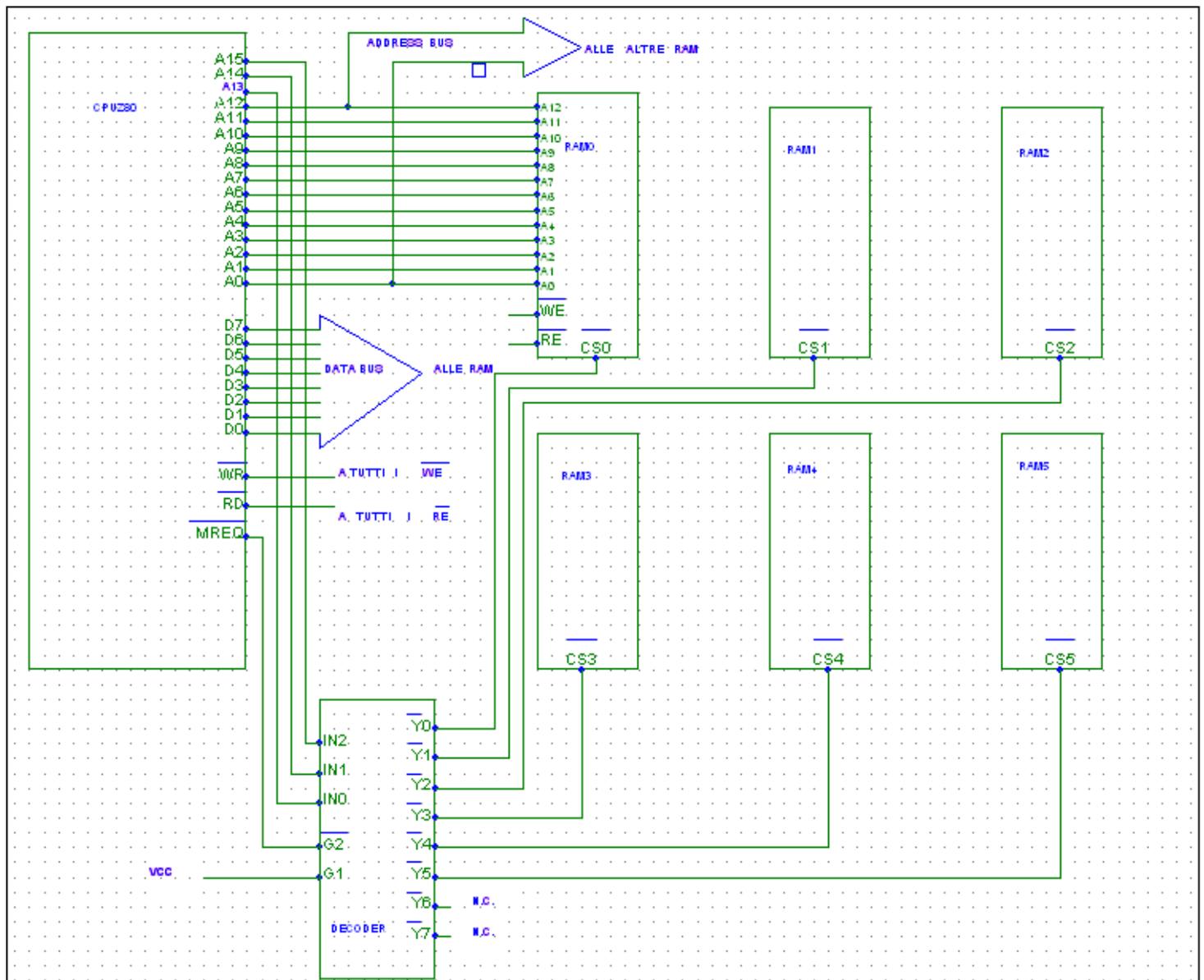
Dato che la capacità **max** di memoria di un  $\mu P$  a **8 bit ( di dati )** e **16 ( di indirizzo )** è di  $2^{16}$  locazioni, non ci potranno essere più di **8 RAM da 8 KBYTE** ciascuna , per esempio, oppure **4 RAM da 16 KB** o altre combinazioni il cui prodotto dia **64 KB** .

La capacità della scheda può essere inferiore alla max, mai ovviamente superiore.

Chi permette di effettuare la scelta fra le varie RAM ?

Un decoder esterno, 3X8 nel primo esempio ( 8 RAM da 8 KB ) , 2x4 nel secondo ( 4 RAM da 16 KB ) .

Vediamo un altro esempio : SCHEDA DI MEMORIA per  $\mu P$  Z80 , composta da 6 RAM da 8 Kbyte e un decoder 3 x 8 con 2 enable ,  $G_1$  attivo alto e  $G_2$  attivo basso.



**N.B.** Si sono indicati i principali segnali e collegamenti solo sulla RAM 0 .

**Analizziamo lo schema :** dato che le RAM contengono ciascuna 8 KB , dovranno ricevere 13 bit dall' address bus dello Z80. I 3 bit + significativi dell'indirizzo (  $A_{15}$ ,  $A_{14}$  e  $A_{13}$  ), invece , sono inviati ai 3 IN del decoder. In base al codice binario presente su tali IN, verrà attivata la corrispondente uscita del decoder e tale uscita attiverà la RAM ad essa collegata, tramite il pin CS ( attivo basso ). Anche il decoder deve essere attivato : su  $G_1$  viene fornito il livello alto, fisso,

mentre  $G_2$  è collegato al segnale  $\overline{MREQ}$ , che vale 0 solo quando il  $\mu P$  esegue un'istruzione di lavoro in memoria.

Quando il  $\mu P$  esegue altre istruzioni, non in memoria,  $\overline{MREQ}$  è disattivato, cioè vale 1 , per cui il decoder è disattivato e così pure tutto il banco di memoria.

Tutte le RAM sono fisicamente collegate ai vari BUS ( ADDRESS , DATA , SUPPLY BUS ) e ricevono anche 2 segnali del CONTROL BUS : le abilitazioni alla lettura e alla scrittura ( RE e WE )

Soltanto la RAM abilitata dal decoder , però , risulta elettricamente collegata . Tutte le altre sono in ALTA IMPEDENZA, cioè irraggiungibili dai segnali .

INDIRIZZI DELLE 6 RAM :		$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$		
RAM 0	INIZIALE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000	H
	FINALE	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF	H
RAM 1 ( 2000 ÷ 3FFF )		RAM 2 (4000 ÷ 5FFF)			RAM 3 ( 6000 ÷ 7FFF )														
RAM 4 ( 8000 ÷ 9FFF )		RAM 5 ( A000 ÷ BFFF )																	

## SISTEMA A MICROPROCESSORE

I blocchi **fondamentali** di un sistema a microprocessore comprendono :

- la memoria
- il microprocessore
- le unità input / output ( **I / O** )

I collegamenti tra i blocchi avvengono grazie a collegamenti chiamati **bus**, realizzati generalmente come piste di rame sulle quali viaggiano segnali di natura **elettrica**.

Sui bus viaggiano **4 tipi di informazioni** il cui controllo è sempre a carico del microprocessore:

- dati
- istruzioni di programmi
- indirizzi
- comandi e segnalazioni varie.

I **programmi** risiedono nella **memoria** della macchina e servono a dare indicazioni sui compiti da svolgere e sulle operazioni da eseguire sui **dati**, con i dati anch'essi residenti in memoria.

Gli **indirizzi** servono a identificare e recuperare in modo univoco le **informazioni** richieste dal lavoro in corso, oppure per attivare e richiamare un'unità di **ingresso** o di **uscita**.

Il microprocessore ha **sempre** e **comunque** una posizione di **predominio** e di **controllo** su tutte le altre parti del sistema.

Il microprocessore, infatti, si fa carico anche di gestire **tutti gli altri blocchi**, ovvero richiamare, **attivare** e fermare le **unità periferiche**, sincronizzare le **operazioni** tra i vari blocchi e controllare le situazioni di **emergenza ( interrupt non mascherabili )**.

Per fare tutto questo ci sono delle **linee speciali** che mettono in comunicazione i singoli **blocchi** con il **microprocessore**, in modo che il microprocessore stesso possa fornire **comandi** e ricevere **segnalazioni** in modo **veloce** e **chiaro**.

La distinzione di cui sopra comporta la successiva suddivisione delle linee che compongono il bus.

Si distinguono cioè i **4** successivi bus :

- |                        |   |         |     |
|------------------------|---|---------|-----|
| • bus dati             | → | DATA    | BUS |
| • bus indirizzi        | → | ADDRESS | “   |
| • bus di controllo     | → | CONTROL | “   |
| • bus di alimentazione | → | SUPPLY  | “   |

Il **bus dati** è composto da **8 linee bidirezionali**, con ognuna delle 8 linee che può essere usata sia in **entrata** che in **uscita** dal microprocessore, a seconda dell'istruzione in esecuzione.

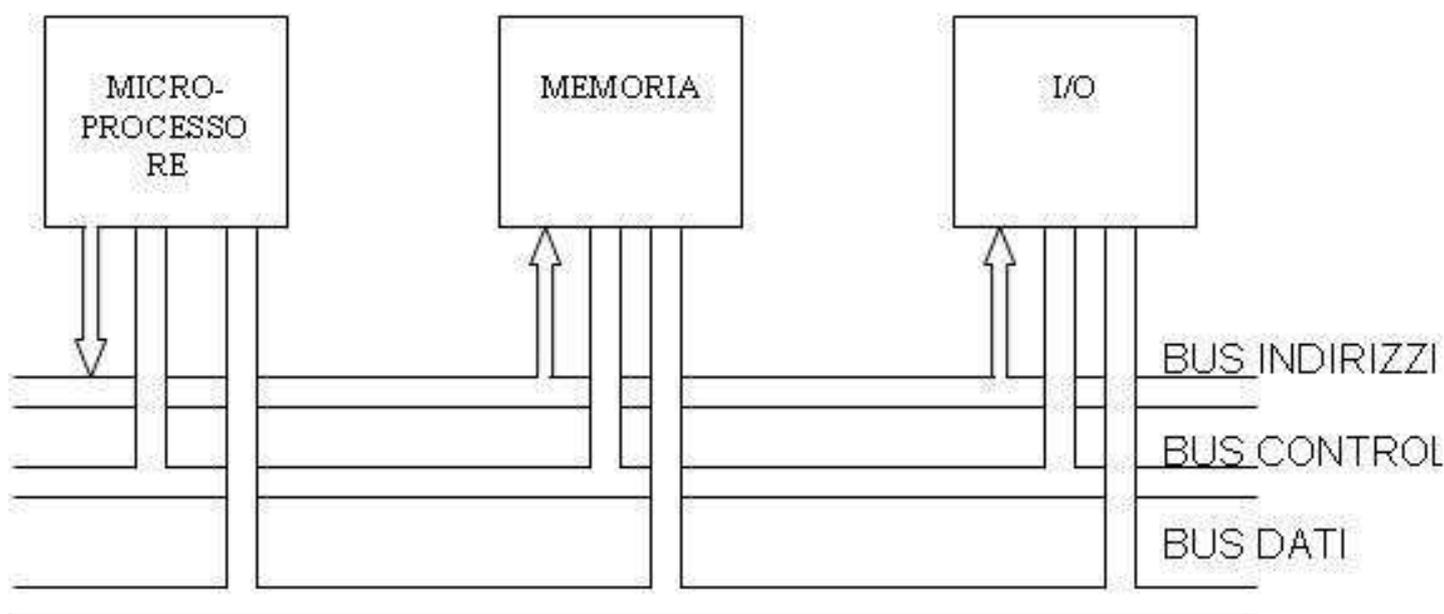
Il **bus indirizzi** è composto da **16 linee unidirezionali** solo in **uscita** dal microprocessore.

Il **bus controllo** è composto da un numero variabile di linee unidirezionali in **ingresso** e da un numero variabile di linee unidirezionali in **uscita**.

Il **bus di alimentazione** è composto da 3 linee : **GND**, ovvero il collegamento di massa, **VCC** , ovvero l'alimentazione, e il **CLOCK**.

Il **CLOCK** è un segnale di sincronizzazione e di temporizzazione che **scandisce** le operazioni del microprocessore.

Nella figura successiva sono illustrati i **blocchi fondamentali** di un **sistema a microprocessore** con i relativi **bus**.



( **negli schemi viene solitamente sottinteso il bus di alimentazione** )

Il **microprocessore** si incarica di **gestire il programma** e i suoi **dati** e di **eseguire i calcoli** richiesti.

Le azioni appena elencate rendono necessario che il microprocessore abbia da qualche parte, al suo interno, qualcosa che gli consenta di **prendere nota** di ciò che sta **facendo** e di trascrivere i **risultati parziali** dei suoi **calcoli**.

Ed infatti all'interno del microprocessore c'è una serie di **registri**, appunto impiegati per tutta quella serie di operazioni che devono essere svolte con **velocità**, dati i **frequenti accessi** richiesti, oppure che servono alla definizione dello **stato** del microprocessore stesso.

Tali registri sono collocati all'interno del microprocessore, e ciò **per risparmiare il tempo** che sarebbe richiesto se si impiegassero i 3 bus già esaminati.

I registri si dividono in due categorie :

- i registri generali
- i registri speciali

I registri generali non hanno uno scopo specifico, ma servono per mantenere traccia del lavoro in corso sui dati / indirizzi e vengono usati solo in determinate istruzioni.

I registri speciali, invece, servono a prendere nota di un particolare aspetto o evento relativo allo stato complessivo del microprocessore ; alcuni di essi sono utilizzati in ogni singola istruzione.

Nella figura successiva è riportato un esempio di blocchi interni di un microprocessore.

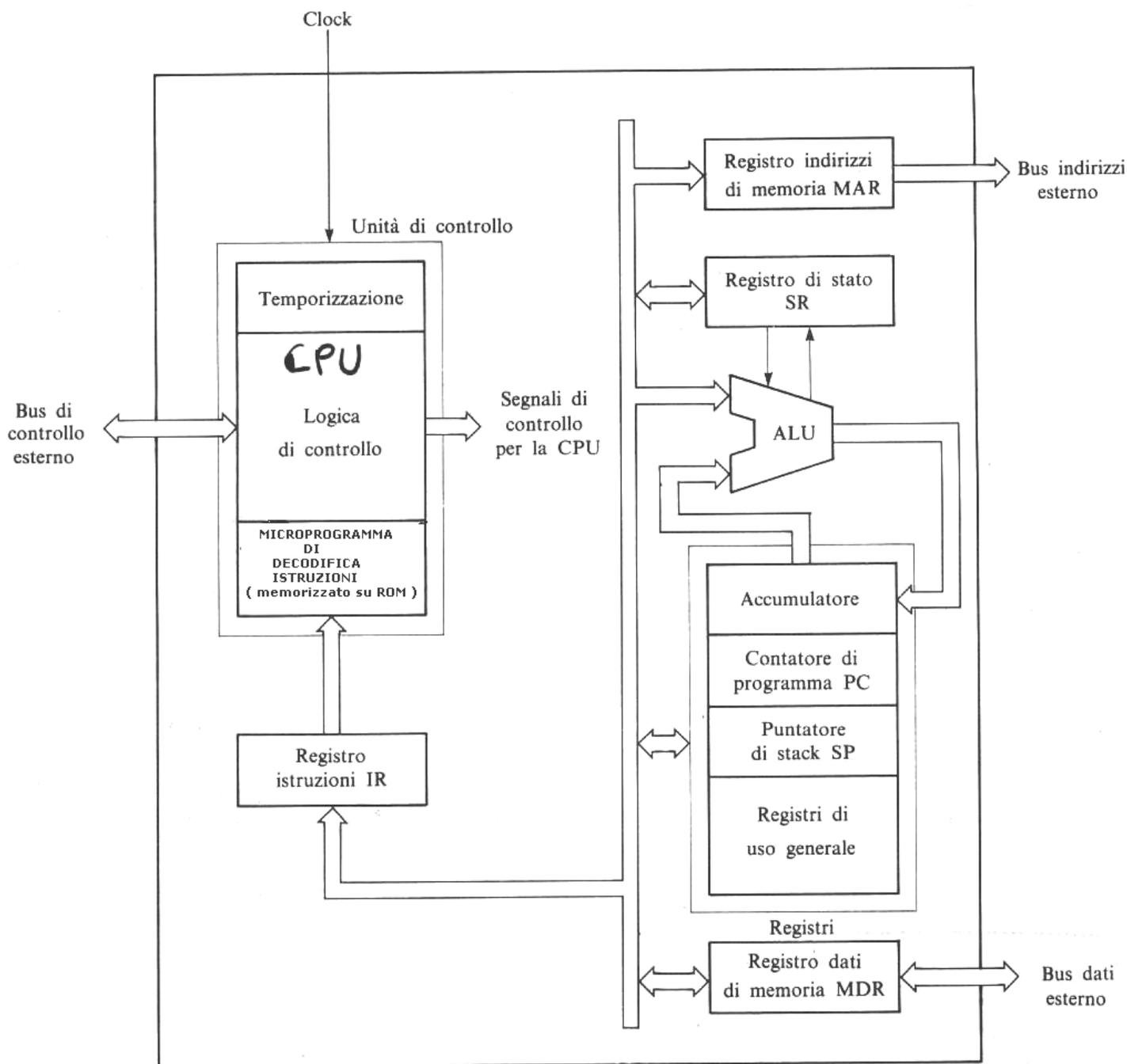


FIG. 1 Struttura di un µP.

- registro **PC** ( **program counter** ) è il registro contatore di programma, esso serve per conservare l'indirizzo della locazione successiva di memoria da analizzare, in cui è scritta la prossima istruzione da eseguire .
- registro **MAR** ( **memory address register** ) è il registro indirizzi, esso è un registro di parcheggio contenente l'indirizzo che è stato inviato sul bus indirizzi. E' un registro 3-state (buffer) che serve a **interfacciare l'address bus interno del  $\mu$ P con quello esterno , di sistema.**
- **MDR** ( **Memory data register** ) è il registro di lettura e scrittura dei dati, il suo scopo è di fungere da “area di parcheggio dei dati” che :
  - devono essere posti sul bus dati verso l'esterno del microprocessore ( istruzione di scrittura in MEMORIA o in PERIFERICA )
  - sono appena arrivati all'interno del microprocessore per essere manipolati secondo quanto indicato dal programma. ( istruzione di lettura in MEMORIA o in PERIFERICA )

E' un registro 3-state (buffer) che serve a interfacciare il data bus interno del  $\mu$ P con quello esterno, di sistema.

- **IR** ( **instruction register** ) è il registro di istruzione, la sua funzione è di conservare il codice operativo dell'istruzione in corso di esecuzione, per tutto il tempo necessario alla sua decodifica ed esecuzione.

Si interfaccia con una ROM in cui risiede il programma di decodifica di tutti i codici operativi .

- **ACC** ( **accumulator** ) è il registro accumulatore, usato nelle operazioni logiche aritmetiche perché alternativamente svolge il ruolo di :
  - sorgente di uno degli operandi dell' istruzione da svolgere all'interno dell' ALU.
  - destinazione del risultato dell' istruzione svolta all'interno dell'ALU.
- **SP** ( **stack pointer** ) è il puntatore all'area di stack, questo registro contiene l'indirizzo dell'ultima locazione occupata dall'area di stack ( particolare area di memoria usata in alcune , ad esempio nello svolgimento delle subroutine )
- **SR** ( registro di **stato** ), in questo registro ogni bit ha un significato specifico visto che ognuno indica il verificarsi o meno di un evento specifico, la sua consultazione è fondamentale ogni volta che il microprocessore deve prendere una decisione sul flusso di azioni da eseguire, con i flag che sono modificati solo da operazioni di tipo logico o aritmetico
- L'**unità di controllo UC** è il referente unico del bus di controllo; in base, infatti, ai segnali di controllo eventualmente attivati, o alle istruzioni date da programma, l'unità di controllo deve prendere le sue decisioni e attivare le parti coinvolte del processore. L'unità di controllo dunque si preoccupa di identificare e decodificare l'istruzione e di conseguenza eseguire l'azione relativa. Per permettere il riconoscimento e la decodifica, in testa ad ogni istruzione, è presente il cosiddetto **codice operativo**.

- L'**ALU** lavora con la maggior parte dei registri indicati nella figura sopra riportata, tuttavia il suo lavoro ha effetto immediato sicuramente sull'accumulatore ACC (che le fornirà sicuramente uno degli operandi), e sul flag F (che conterrà indicazioni sul risultato delle operazioni).

## Unità di controllo

L'unità di controllo (vedi fig. 1) comprende il *decodificatore di istruzioni* e i circuiti di *temporizzazione* che generano segnali di comando e sincronizzazione sia per il  $\mu\text{P}$  che per la memoria e gli altri organi esterni. Tutte le operazioni dell'unità di controllo sono a loro volta regolate e sincronizzate da un *clock*, che può essere fornito dall'esterno o generato all'interno del  $\mu\text{P}$ . In linea di massima l'unità di controllo

- comanda il prelievo (fetch) dalla memoria dell'istruzione indirizzata dal contatore di programma (PC) e il suo trasferimento al registro istruzioni (IR), incrementando infine il PC stesso;
- decodifica l'istruzione e precisamente il suo codice operativo contenuto nell'IR;
- esegue (execute) l'istruzione attivando con i segnali opportuni gli altri componenti della CPU e generando i segnali di controllo per la memoria o per i dispositivi di I/O.

Di solito il decodificatore di istruzioni è costituito da una memoria ROM che contiene il **microprogramma**. A ciascuna delle istruzioni che il  $\mu\text{P}$  è in grado di eseguire (*set di istruzioni*), ossia a ciascun codice operativo, corrisponde una sequenza di microistruzioni memorizzate in codice binario nella ROM. Pertanto, in corrispondenza di ciascun codice operativo, sulle uscite del decodificatore si presenta una sequenza di configurazioni binarie, che costituiscono i segnali di attivazione per i vari elementi del  $\mu\text{P}$  e dei dispositivi esterni. Si noti che i segnali prodotti dalla sequenza di codici binari sono sincronizzati con il clock o con un segnale da esso derivato.

L'attività dell'unità di controllo dipende anche da segnali forniti alla CPU da sorgenti esterne. Tali segnali, che verranno esaminati in dettaglio nel prossimo paragrafo, possono ad esempio essere la richiesta di interrompere il programma o la richiesta di liberare i bus esterni dei dati e degli indirizzi.

## Temporizzazioni

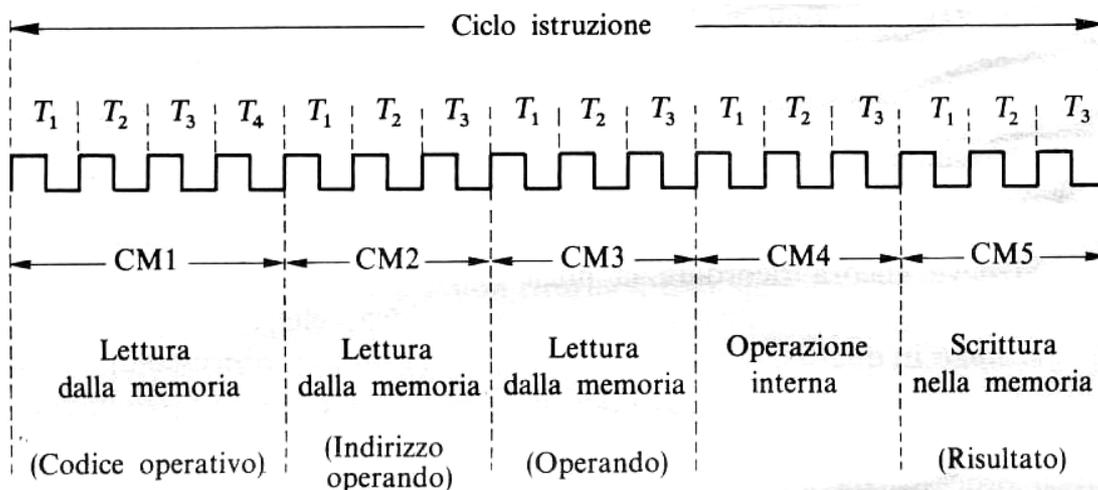
L'esecuzione di ciascuna istruzione di cui è composto un programma costituisce il *ciclo istruzione*; esso prevede due fasi distinte chiamate rispettivamente fase di *fetch*, o di prelievo dell'istruzione dalla memoria, e fase di *execute* o di esecuzione vera e propria.

Mentre la fase di fetch è sostanzialmente uguale per tutte le istruzioni, lo sviluppo della fase di execute dipende dal tipo di istruzione, e quindi dal suo codice operativo, e dal tipo e dalla posizione (memoria, registro interno, ecc.) degli operandi. In ogni caso, il ciclo istruzione è sempre costituito da una sequenza di passi, chiamati *cicli macchina (CM)*, del tipo illustrato in fig. 7. Durante ciascun ciclo macchina la CPU esegue una delle seguenti operazioni fondamentali:

- fetch dell'istruzione;
- lettura o scrittura di dati in memoria;
- lettura o scrittura di dati in un dispositivo periferico;
- operazione interna alla CPU stessa.

A queste si aggiungono le operazioni che la CPU deve avviare in risposta a segnalazioni provenienti dall'esterno, siano esse interruzioni o richieste di controllo dei bus.

Poiché i  $\mu P$  sono circuiti, o macchine, *sequenziali sincroni*, tutte le azioni che vengono intraprese durante un ciclo macchina sono sincronizzate da un segnale di cadenza; in altri termini la CPU attiva i vari segnali, interni ed esterni, in corrispondenza di determinate transizioni e livelli del clock. Ciascun periodo o *ciclo di clock* (detto anche *stato macchina*) rappresenta l'intervallo di tempo durante il quale viene eseguito il più piccolo passo di elaborazione ossia una microoperazione. Si può aggiungere che ciascuna microoperazione



**Fig. 7.9** - Il ciclo istruzione comprende più cicli macchina ciascuno dei quali è costituito da più cicli di clock.

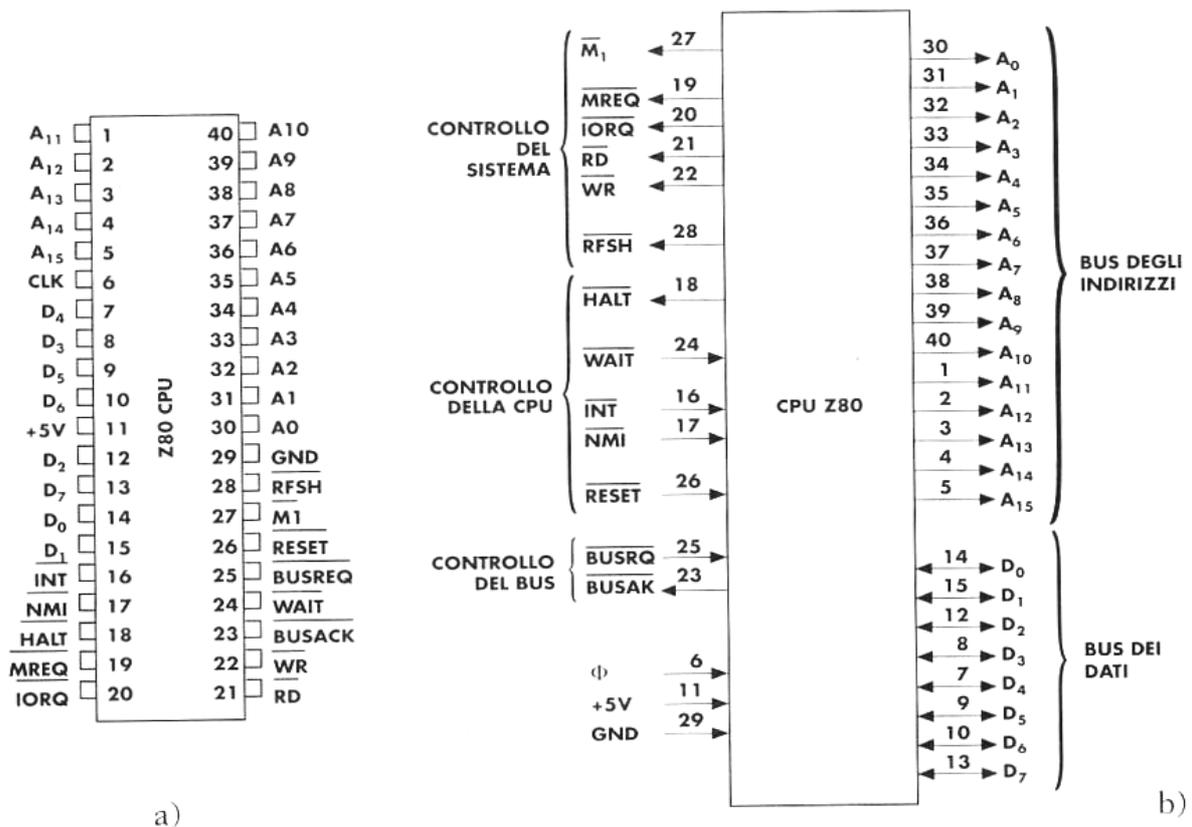
è determinata da un insieme di microcomandi: tale insieme corrisponde ad una *microistruzione* memorizzata nella ROM del decodificatore di istruzioni (vedi unità di controllo). In fig. 7 si vede ad esempio che il ciclo macchina CM1, o di fetch, dura quattro periodi di clock mentre i cicli successivi solo tre.

Di solito il segnale di cadenza per la CPU è proprio il clock fornito dall'esterno attraverso la linea *CLOCK* o generato internamente. In alcuni  $\mu P$  invece, ciascun ciclo di clock è suddiviso in *sottocicli* che temporizzano la CPU per l'esecuzione di passi di livello ancora inferiore. I sottocicli possono essere generati combinando più fasi di clock, applicate esternamente o ricavate da un clock singolo.

## PIEDINATURA DI UN MICROPROCESSORE A 8 BIT

Piedinatura dello Z80.

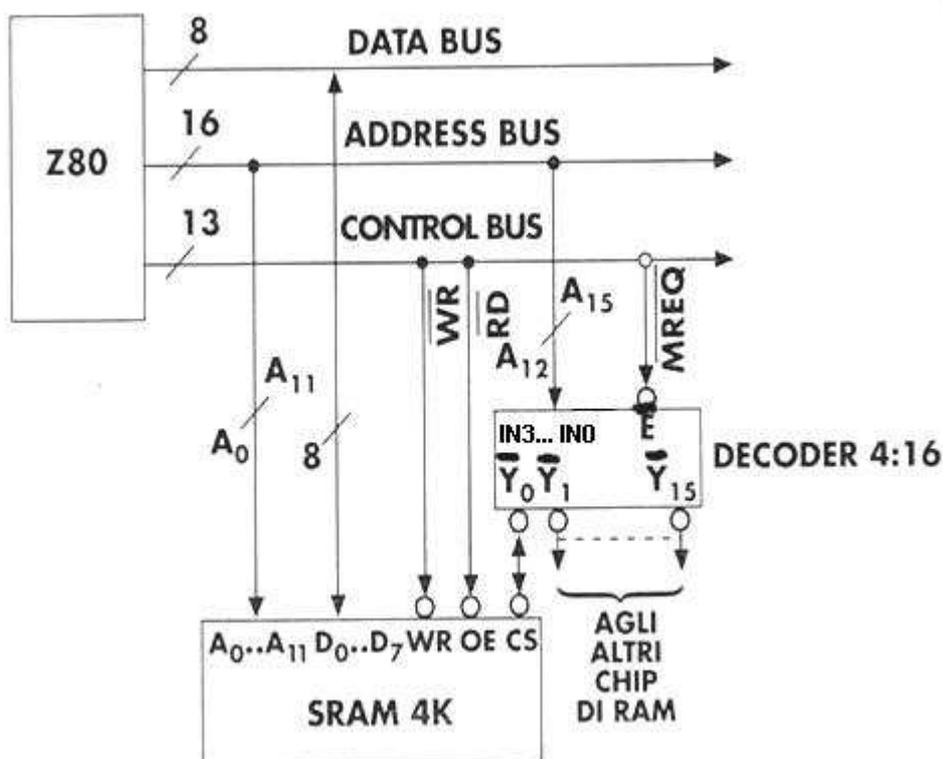
**FIG.8**



## CICLI MACCHINA DI UN μP ( es: Z80 )

- CICLO M1 ( fetch+decodifica+refresh )
- CICLI LETTURA / SCRITTURA IN MEMORIA
- CICLI LETTURA / SCRITTURA IN PERIFERICA
- CICLI DI INTERRUPT ( MASCHERABILE E NON )
- CICLO DI RICHIESTA DEI BUS
- CICLO DI HALT
- CICLO DI RESET

## Esempio di interfacciamento HW con 16 chip di RAM



SW : esempio di istruzioni di lavoro in Memoria

- LETTURA LDA, ( nn )H
- SCRITTURA LD( nn )H, A

Durante il ciclo di Lettura in Memoria, che dura come minimo 3 Tck ma può essere prolungato su richiesta della Memoria stessa, se questa è più lenta rispetto alla CPU, tramite il segnale di WAIT, vengono attivati determinati segnali di controllo, in ben precisi istanti di tempo e con riferimento ai fronti di salita o di discesa del CK.

Viene dapprima messo sull' Address Bus l' indirizzo della locazione in cui si vuole leggere e tale indirizzo vien decodificato su 2 livelli :

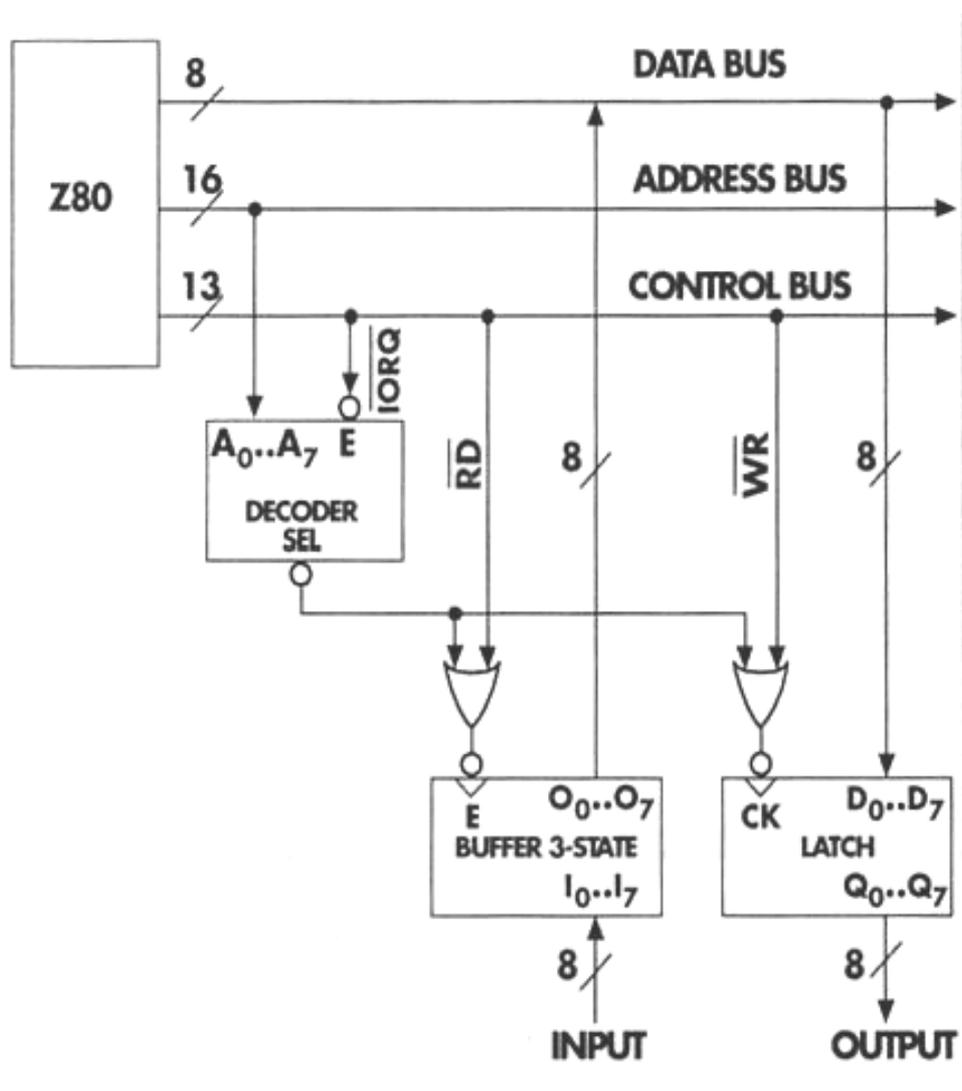
- I bit + significativi, **A15 ÷ A12**, sono connessi ai 4 IN ( IN3 ÷ IN0 ) di un DECODER 4x16, abilitato dal segnale MREQ generato dall'istruzione e provocano l'attivazione ( al livello BASSO ) dell'uscita corrispondente al codice binario presente in IN, es : 0000 >>> **Y0**, la quale attiverà il chip RAM 0 ( **un solo chip tra i 16 disponibili** )
- I bit rimanenti, **A11 ÷ A0**, sono collegati agli IN INDIRIZZI di tutti i 16 chip di RAM e vengono decodificati, come già visto, dai decoder interni di Riga e di Colonna. Come risultato della decodifica, verrà attivata **una sola uscita tra le 64 possibili**, per ciascuno dei 2 decoder di colonna e di riga : questi 2 segnali, insieme, sbloccheranno ( toglieranno dallo stato di HZ ) la corrispondente locazione di memoria ( **una sola tra le 4096, contenute in ogni chip di RAM** )

Ciascun chip di RAM riceve anche i segnali di controllo di lettura e scrittura.

Un volta individuata la locazione, il dato in essa contenuto viene posto sul DATA BUS di sistema e trasferito, durante l'ultimo Tck del ciclo macchina di lettura in Memoria, nel registro specificato dall'istruzione in corso, in questo esempio l' Accumulatore, previo transito attraverso il MDR, buffer di interfaccia tra il DATA BUS di sistema e quello interno al microProcessore.

In modo analogo avviene il trasferimento dati durante il ciclo di SCRITTURA IN MEMORIA.

**Ciclo di lettura/scrittura con le periferiche.** Per quanto riguarda il dialogo con i dispositivi di input/output (fig. 7a), il gioco dei segnali è sostanzialmente lo stesso, tranne che è  $\overline{IORQ}$  ad andare attivo in luogo di  $\overline{MEMQ}$ , e solo le 8 linee più basse dell'address bus sono coinvolte nell'indirizzamento. Anche qui un dispositivo lento può chiedere tramite  $\overline{WAIT}$  l'inserimento di cicli di attesa. A differenza della scrittura/lettura in memoria, un ciclo d'attesa  $T_w$  è comunque inserito "d'ufficio". In figura è riportata una semplice porta per l'input e l'output di un byte. L'input è servito da un buffer 3-state, l'output da un latch, entrambi naturalmente a 8 bit. Il segnale di abilitazione del 3-state e il clock del latch, entrambi supposti attivi bassi, sono dedotti rispettivamente dal  $\overline{RD}$  e dal  $\overline{WR}$  dello Z80, condizionati da un riconoscimento di indirizzo e dall'attivazione di  $\overline{IORQ}$ . Qui il decoder è una semplice rete combinatoria che attiva la propria uscita  $SEL$  quando l'indirizzo su  $A_7..A_0$  è quello impostato e  $\overline{IORQ}$  è basso.



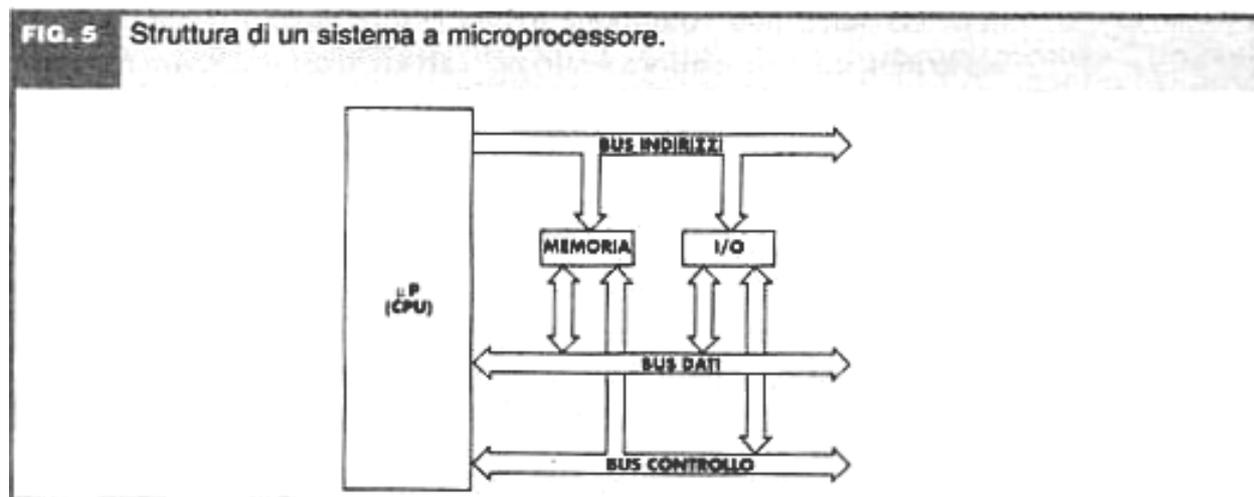
## Il colloquio con le periferiche nello Z80

Un microprocessore ha bisogno, per lavorare, di poter scambiare dati con diversi tipi di dispositivi (genericamente, periferiche). Anzitutto, deve poter accedere a una memoria che contenga il programma da eseguire, codificato come successioni di combinazioni di bit (byte, nel nostro caso) corrispondenti alle singole istruzioni. La zona di memoria programma potrà essere in RAM, come avviene nei calcolatori di uso generale, o in ROM, come di solito si verifica nei sistemi di controllo dedicati, in cui non c'è necessità di cambiare frequentemente il programma, oppure in una combinazione delle due. In ogni caso, sarà necessaria una sezione di memoria RAM più o meno estesa, per allocare lo stack e salvare i risultati intermedi dell'elaborazione. In secondo luogo, il  $\mu P$  deve poter interfacciarsi con il mondo esterno, il che avviene tramite i dispositivi di Input/Output (sinteticamente I/O).

In termini del tutto generali, il  $\mu P$  colloquia con le altre componenti del sistema scambiando con loro informazioni attraverso il bus dati (che è bidirezionale). Poiché tutti gli interlocutori fanno capo a questo bus, il  $\mu P$  deve arbitrare lo scambio ed evitare conflitti, decidendo di volta in volta che dispositivo debba "parlare" mettendo i suoi dati sul bus, o "ascoltare", cioè leggere i dati dal bus stesso. Ci si riferisce a questa funzione di coordinamento e direzione svolta dal  $\mu P$ , dicendo che esso costituisce il master (padrone) del sistema, mentre i suoi interlocutori sono gli slave (schiavi).

La comunicazione del  $\mu P$  con uno slave presuppone innanzitutto la selezione dello slave stesso. La selezione viene fatta attraverso il bus degli indirizzi: ciascun sottosistema slave (e qui si intende la singola cella di memoria o la singola locazione di I/O) è associato a un indirizzo; per essere attivato, lo slave deve essere indirizzato, cioè deve riconoscere sull'address bus la combinazione di bit che costituisce il proprio indirizzo. La decisione sul tipo di operazione di colloquio da eseguire (lettura o scrittura, accesso alla memoria o allo I/O) è comunicata dal  $\mu P$  agli slave attraverso opportuni segnali del bus di controllo. L'organizzazione generale del sistema è così quella riportata in figura 5 (in questa figura il bus di controllo è indicato come bidirezionale per tenere conto anche dei casi in cui i segnali di controllo vanno al  $\mu P$  (ad esempio, in una richiesta di interruzione).

FIG. 5 Struttura di un sistema a microprocessore.



Gli slave che inviano dati sul bus (genericamente, le sorgenti o talker, parlatori) devono essere dotati in uscita di un'interfaccia di tipo 3-state, per potersi "staccare" elettricamente dal bus quando la sorgente attiva è un'altra, evitando così conflitti di livello. Gli slave che leggono dati dal bus (destinazioni o listeners, ascoltatori) sono invece solitamente dotati di qualche sistema per memorizzare il dato letto. Abbastanza lapalissianamente, e per definizione, ciò avviene nelle memorie, ma anche nei port (porti o, un po' impropriamente, porte) di output, perché in genere il mondo esterno è più lento del  $\mu P$  ed è quindi necessario "congelare" il dato in uscita fino a che esso non sia stato utilizzato; si dice allora che le porte di uscita esplicano una funzione di latch (to latch = mettere sotto chiave).

Come abbiamo già accennato, lo Z80 è un microprocessore a I/O separato (I/O mapped), che cioè distingue tra le operazioni di scrittura/lettura in memoria e quelle in I/O. La distinzione avviene mediante la coppia di segnali di qualificazione  $\overline{MEMQ}$  e  $\overline{IORQ}$ , il primo attivato dal  $\mu P$  durante il colloquio con la memoria, il secondo durante quello con i dispositivi di input/output. Un'ulteriore differenza fra gestione di memoria e gestione di I/O è che in quest'ultima sono utilizzate per l'indirizzamento dei dispositivi solo le 8 linee basse  $A_0..A_7$  dell'address bus; il numero di dispositivi di I/O che possono scambiare un byte con il  $\mu P$  è così limitato a 256.

Il trasferimento di informazioni fra la CPU, corredata della relativa memoria, e il mondo esterno avviene attraverso i dispositivi periferici ossia gli organi di ingresso-uscita (I/O); esso riguarda essenzialmente il caricamento di programmi in memoria e il loro eventuale salvataggio su memorie di massa, l'acquisizione e/o la memorizzazione dei dati da elaborare, il trasferimento all'esterno dei risultati delle elaborazioni.

Gli innumerevoli tipi di dispositivi periferici, si differenziano profondamente oltre che per il ruolo che svolgono, per le loro caratteristiche funzionali ed elettriche. Per questo motivo vengono connessi alla CPU attraverso specifici circuiti di interfaccia (*interfacce di I/O*) che provvedono alla sincronizzazione e all'adattamento funzionale ed elettrico dei segnali che devono essere scambiati.

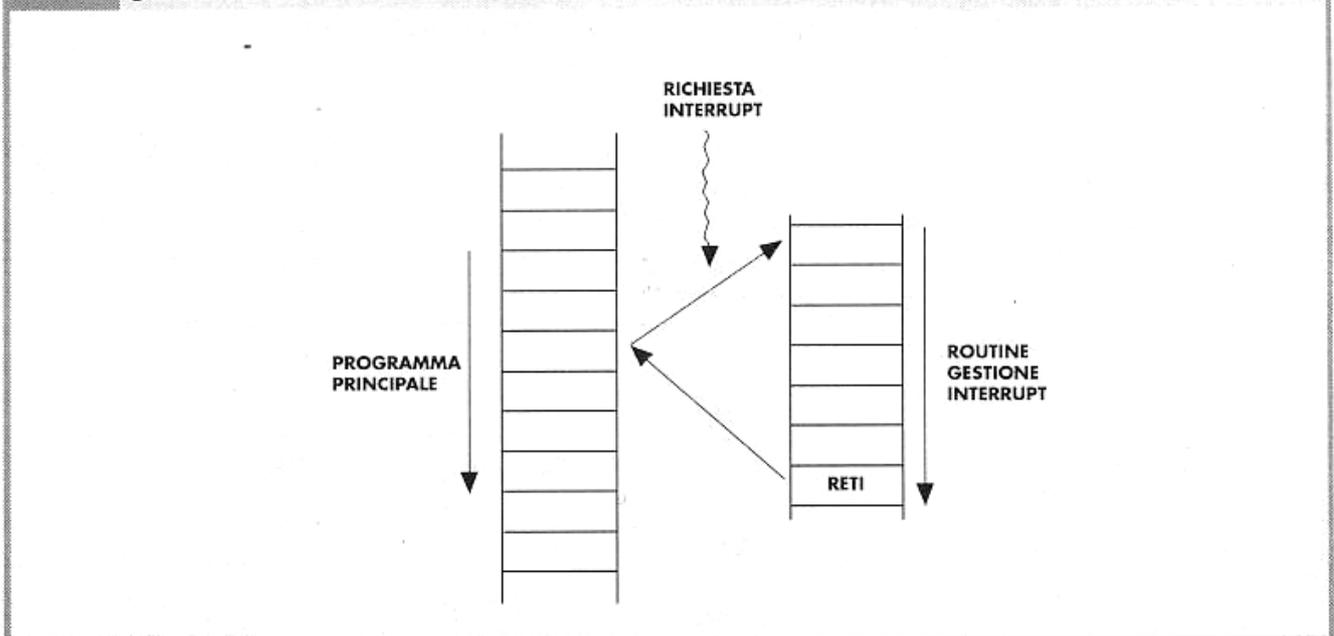
In linea generale un circuito di interfaccia è connesso da un lato al dispositivo periferico di I/O che deve pilotare e dall'altro alle linee esterne della CPU, il bus dati, il bus indirizzi ed eventuali linee di controllo. Ogni dispositivo di interfaccia comprende uno o più *canali di comunicazione*, in genere bidirezionali, che consentono lo scambio di dati in un senso e nell'altro fra la CPU e l'unità periferica; tali canali sono denominati *porte di I/O* (*I/O port*). La larghezza massima di una porta è pari al numero di linee del bus dati anche se i bit del dato in trasferimento possono essere in numero inferiore. Le informazioni che vengono scambiate possono infatti essere dati veri e propri o parole di controllo, ossia sequenze di bit che condizionano determinate funzioni dell'interfaccia o dell'unità periferica.

Esistono **interfacce** dedicate e programmabili, seriali o parallele, per i vari processori. ( SIO Z80, PIO Z80 ) Esse permettono il colloquio con **periferiche** semplici di OUT, come LED, interruttori, relais, per cui basta una sola linea di uscita oppure con periferiche più complesse come un DAC, per cui si usano tutte le uscite, in parallelo. Analogamente, per quanto riguarda le periferiche di IN, potranno essere molto semplici ( sensori digitali a un solo bit, per segnalare il superamento di un valore di soglia da parte di una grandezza fisica) o più complesse ( ADC ).  
Va anche detto che le periferiche più complesse, come monitor (OUT), scanner (IN), dispongono di un CONTROLLER (HW) e di un DRIVER (SW) per dialogare con un microP.

## La gestione degli interrupt nello Z80

Tutti i microprocessori, in forme più o meno complesse, hanno la possibilità di essere interrotti nell'esecuzione del programma e chiamati a gestire la situazione che ha determinato l'interruzione (interrupt), per poi ritornare a riprendere l'esecuzione del programma originale dall'istruzione successiva all'ultima eseguita. Lo schema generale di questo procedimento è illustrato in figura 10 dove, al solito, sezioni di programma sono schematicamente rappresentate dalle locazioni in memoria contenenti le istruzioni, ordinate dall'alto verso il basso nel senso degli indirizzi crescenti, cioè nell'ordine di esecuzione. RETI è la particolare istruzione che termina (nello Z80 e per una particolare categoria di interrupt) la sezione di programma eseguita in conseguenza dell'interruzione, cioè, come si dice, la *routine di gestione dell'interruzione*.

**FIG. 10** Una richiesta di interruzione determina il passaggio dal programma in esecuzione a quello di gestione dell'interruzione; al termine di questa routine si ritorna al programma precedente.



La tecnica di interruzione sostanzialmente *serve a instaurare un dialogo asincrono tra  $\mu P$  e dispositivi di I/O*: quando il dispositivo necessita di servizio da parte del  $\mu P$  (di solito lo scambio di dati in un senso o nell'altro), lo chiede tramite interrupt.

In questo modo il  $\mu$ P stabilisce la comunicazione *solo quando è necessario*, mentre per il resto del suo tempo-macchina può continuare a eseguire altri compiti. In ogni caso, per poter gestire le interruzioni, un  $\mu$ P deve assolvere ad alcune funzioni irrinunciabili:

- 1) deve essere, ovviamente, in grado di ricevere le richieste di interruzione;
- 2) deve segnalare l'avvenuta accettazione all'esterno, in particolare al dispositivo interrompente per avvertirlo che è pronto a un'eventuale comunicazione;
- 3) deve salvare (di solito sullo stack) l'indirizzo dell'istruzione a cui deve ritornare al termine dell'interruzione (indirizzo di rientro), in genere il proprio stato interno (i flag), ed eventualmente il contenuto di alcuni registri (in pratica, deve "ricordarsi" dove era arrivato e che cosa stava facendo prima di essere interrotto);
- 4) se, come succede spesso, i dispositivi che possono chiedere interruzione sono più d'uno e richiedono servizi diversi, deve prevedere qualche logica con cui individuare il dispositivo interrompente e selezionare la relativa routine di servizio;
- 5) deve, infine, dirimere il conflitto fra diverse richieste di interruzione che si sovrappongono.

Analizziamo come lo Z80, peraltro in modo abbastanza tipico, assolve a queste funzioni; inizialmente trattiamo gli **interrupt mascherabili**, cioè di *quelli a cui il  $\mu$ P può essere abilitato a rispondere oppure no via software* (a questi ci si riferiva nei 5 punti precedenti). Dell'altro tipo di interrupt, quello **non mascherabile**, anch'esso previsto in tutti i modelli di microprocessore, diremo in seguito.

**Lo Z80 controlla il proprio ingresso di INT sul fronte di salita dell'ultimo Tck dell'istruzione in corso, onde evitare di lasciarla incompiuta, e se trova il livello attivo (basso) fa partire il CICLO Macchina di riconoscimento Interrupt, durante il quale capirà quale periferica lo sta interrompendo, per inviargli dati.**

**Ovviamente solo le periferiche di IN possono lanciare Interrupt e dato che l' interrupt è un **evento asincrono**, e nel caso peggiore può arrivare al microP all' inizio del 1° Tck dell'istruzione più lunga, la periferica interrompente deve mantenere attiva la richiesta almeno per la durata di esecuzione dell' istruzione più lunga, per l'appunto.**

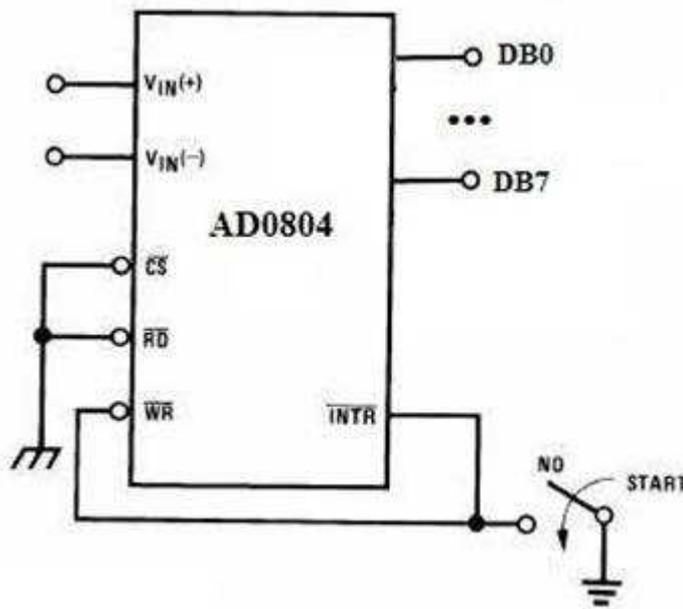
## INTERFACCIAMENTO TRA ADC E MICROPROCESSORE

### Free running mode

Il modo più semplice per usare un convertitore A/D è il cosiddetto free running mode: consiste nel collegare direttamente l'uscita di fine conversione ( **EOC** ) con l'ingresso di inizio conversione ( **SOC** ) del convertitore.

In questo modo al termine di ogni conversione ne viene avviata una nuova automaticamente, senza nessun intervento dall'esterno e il periodo di campionamento coincide esattamente col tempo impiegato dal convertitore per effettuare una conversione.

Si consideri, a titolo di esempio, lo schema seguente, basato sul convertitore AD0804. Nello schema sono stati indicati solo i pin fondamentali per la comprensione del meccanismo di free running.



Si tratta di un convertitore a 8 bit, con uscite DB0, DB1,... DB7.

Il segnale analogico di ingresso viene applicato in modo differenziale ( senza riferimento a massa ) fra i piedini **Vin(+)** e **Vin(-)**.

Gli ingressi CS e RD, **attivi bassi**, sono collegati a massa in modo da attivare sempre l'integrato e abilitarne le uscite.

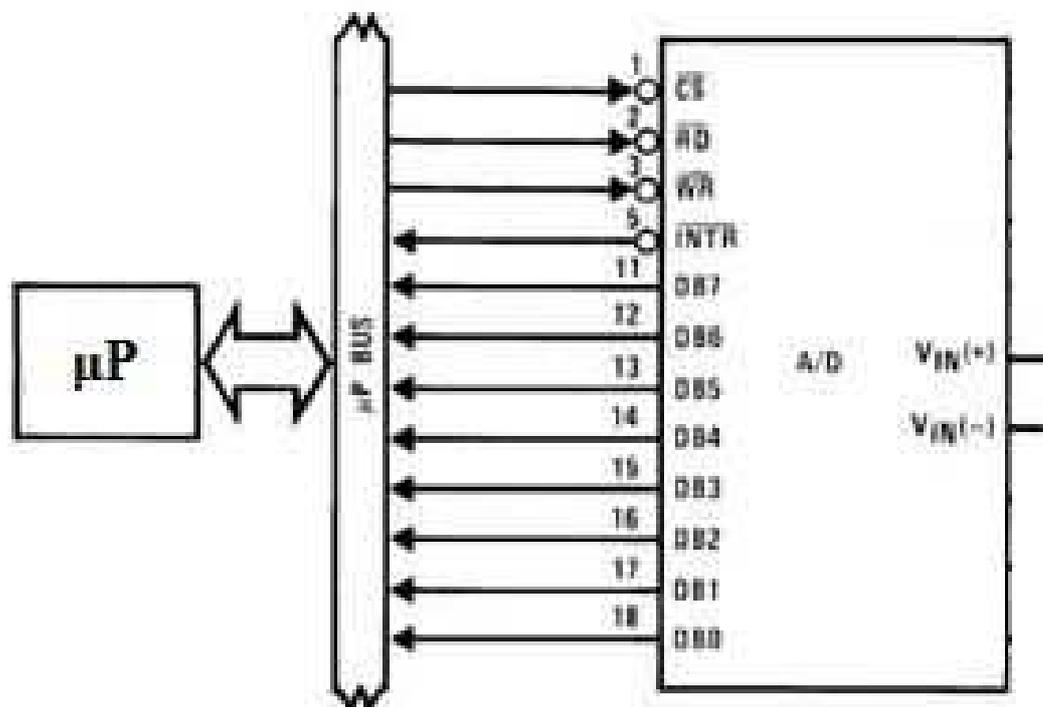
L'ingresso di inizio conversione ( denominato WR ) è collegato direttamente con l'uscita di fine conversione ( INTR ), di modo che ogni conversione terminata ne avvil un'altra.

L'interruttore in ingresso serve per avviare la prima conversione ( è necessario, altrimenti la sequenza di conversioni non inizierebbe mai ).

Questa soluzione è semplice, ma presenta l'inconveniente di **non poter modificare il periodo di campionamento**. Inoltre il periodo di campionamento non viene temporizzato in modo preciso, poichè dipende essenzialmente dai ritardi interni all'ADC.

## Interfacciamento con un microprocessore

Una possibilità più evoluta consiste nell'interfacciare il convertitore ADC con un microprocessore (  $\mu\text{P}$  ), come mostrato in modo sintetico nella figura seguente:



In questo caso il  $\mu\text{P}$  si occupa di avviare ciascuna conversione e di leggere i valori presenti sui pin di uscita al termine di ogni conversione. Il periodo di campionamento viene in questo caso gestito dal  $\mu\text{P}$  stesso ( **temporizzazione software** ) oppure tramite hardware esterno ( **temporizzazione hardware** ).

Per temporizzazione software si intende l'esecuzione da parte del  $\mu\text{P}$ , all'inizio di ogni ciclo di regolazione, di un ciclo di ritardo di durata pari al periodo di campionamento.

Questo tipo di temporizzazione è poco costosa, in quanto non richiede hardware aggiuntivo, ma presenta lo svantaggio di essere piuttosto imprecisa, soprattutto per periodi di campionamento molto lunghi.

Inoltre **durante il ciclo di ritardo il  $\mu\text{P}$  risulta impegnato inutilmente** e non può essere usato per altri scopi.

La soluzione alternativa è costituita dalla cosiddetta temporizzazione hardware.

In questo caso è necessario usare un timer esterno, cioè un dispositivo in grado di inviare al  $\mu\text{P}$  un segnale di interruzione ( *interrupt* ) allo scadere di ogni periodo di campionamento.

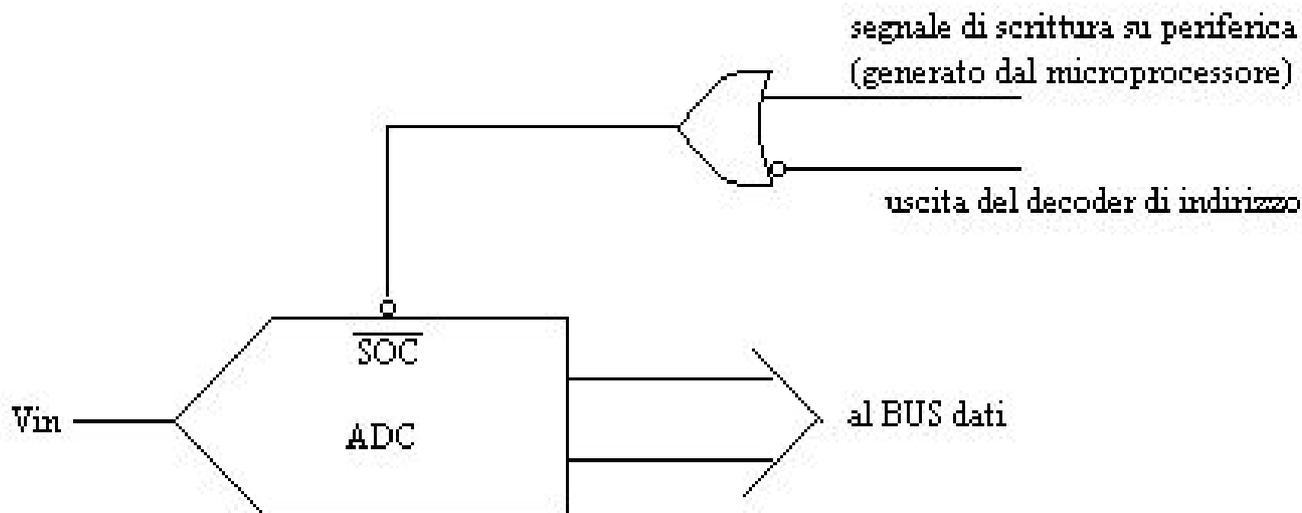
Ogni volta che il  $\mu\text{P}$  riceve un segnale di interrupt dal timer, esso esegue una opportuna **routine di servizio dell'interrupt**, all'interno della quale viene acquisito un campione digitale.

## Gestione dello start of conversion

Allo scadere di ogni periodo di campionamento, il  $\mu\text{P}$  avvia la conversione inviando all'ADC un segnale impulsivo di start of conversion ( SOC ).

Ciò avviene eseguendo sul microprocessore una istruzione di invio dati a periferica ( OUT ) all'indirizzo del convertitore ( il valore inviato con tale istruzione non ha nessuna importanza, dal momento che l'istruzione serve solo per attivare l'indirizzo dell'ADC ).

Lo schema di riferimento è mostrato in figura:



Nello schema in figura il segnale di start of conversion è attivo a **livello basso**.

Per avviare la conversione, il microprocessore genera un segnale di scrittura su periferica ( attivo anch'esso a livello basso ) ; supponiamo invece che il decoder di indirizzo produca un segnale a livello alto, il quale viene inviato all'ADC.

La porta OR pertanto produce un impulso di SOC a livello basso quando il  $\mu\text{P}$  effettua un'operazione di scrittura all'indirizzo del convertitore.

## Gestione dell' end of conversion

Una volta avviata la conversione, l'ADC segnala la fine della conversione, portando a livello basso un' opportuna linea di uscita detta di end of conversion ( EOC ).

La lettura del dato convertito non può avvenire dunque immediatamente dopo l'impulso di SOC, ma occorre prima attendere che l'ADC abbia completato la conversione.

Il modo più semplice per risolvere il problema consiste nel non testare il valore della linea di end of conversion.

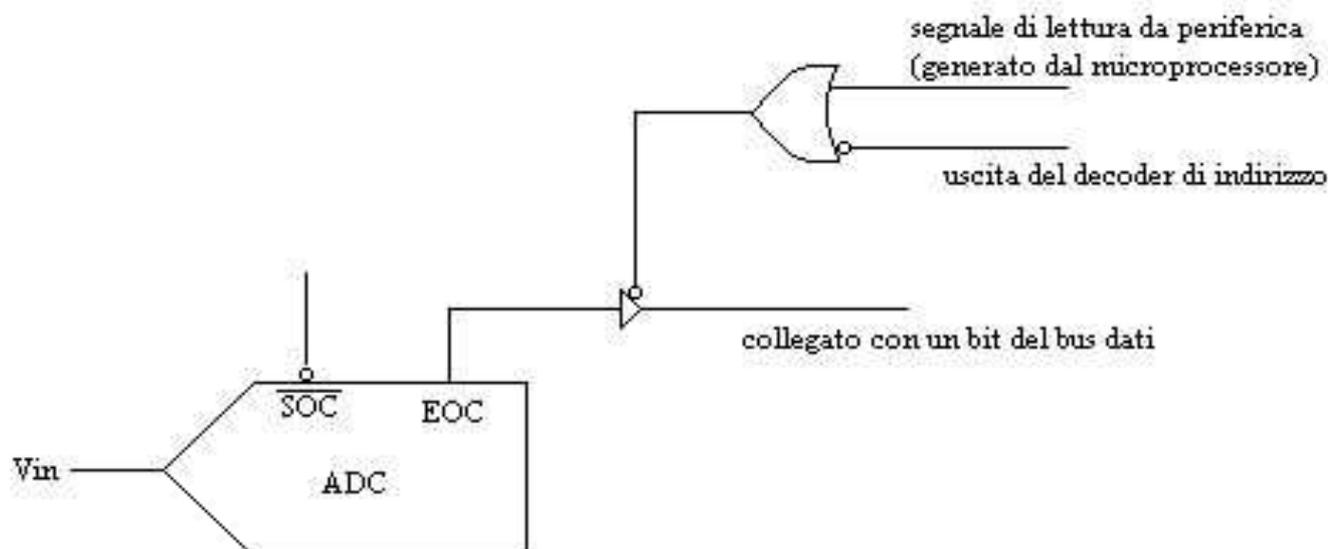
In pratica basta far eseguire al  $\mu$ P un ciclo di ritardo di durata opportuna fra l'avvio della conversione e la lettura del dato convertito.

Lo svantaggio è che, per assicurare una corretta conversione in tutti i casi, occorre sovradimensionare la durata del ciclo di ritardo ed in tale modo la gestione del tempo non viene ottimizzata.

Un'altra possibilità consiste nell'interrogare ciclicamente in **polling** la linea EOC del convertitore.

L'interrogazione ciclica avviene eseguendo sul microprocessore una istruzione di lettura da periferica ( IN ) all'indirizzo assegnato alla linea EOC.

Lo schema di riferimento è mostrato in figura:



L'alternativa più complicata per la gestione dell'EOC ( ma anche quella che ottimizza la gestione dei tempi ) consiste nel generare un segnale di interrupt al  $\mu$ P ogni volta che è terminata una acquisizione.

La routine di servizio dell'interrupt si occupa quindi di leggere il valore digitale convertito.

Su alcuni ADC la gestione ad interrupt dell'end of conversion è suggerita dal nome stesso del segnale usato per segnalare la fine della conversione ( INTR ). In pratica basta collegare direttamente l'uscita INTR dell'ADC con l'ingresso di interrupt del  $\mu$ P ( se non vi sono altri segnali di interrupt da gestire ).