

## Introduzione ai “computer”

Un *computer* è composto da tre sottosistemi, ciascuno dei quali ha un compito specifico

1. Il “*processore centrale*” (Central Processor Unit”, CPU) esegue l’elaborazione vera e propria sui dati.
2. La “*memoria*” che contiene programmi e i dati su cui operare
3. Il “*processore di ingresso/uscita*” (I/O processor), che costituisce il collegamento con l’utente. Consente di registrare in memoria programmi e dati e permette di conoscere i risultati

## Quale computer?

Per introdurre delle persone inesperte ai meccanismi interni di funzionamento di una struttura di elaborazione completa è **opportuno scegliere un esempio semplice** e per trovarlo, di solito, si torna indietro negli anni, quando le tecnologie di integrazione dei circuiti elettronici non consentivano di realizzare le attuali complesse strutture logiche.

Questo approccio, se offre il vantaggio di disporre della abbondante documentazione tecnica che esiste per tutti i dispositivi effettivamente realizzati, ha come svantaggio che **nessuna delle strutture, anche quelle che hanno avuto grande successo, presenta tutte le soluzioni tecniche che si sono poi affermate come le migliori nel risolvere i vari problemi**. Ciò è naturale, perché anche in Informatica, come in tutte le altre scienze, il progresso si basa su un continuo processo di brillanti invenzioni teoriche che, se sperimentalmente efficaci, passano nell’uso normale, altrimenti vengono accantonate, salvo a tornare d’attualità quando i progressi nella tecnologia d’integrazione consentiranno di realizzarle meglio.

## La “macchina di Mano”

Abbiamo scelto di presentare una **architettura di CPU astratta, messa insieme, appositamente per scopi didattici**, riducendo ai minimi elementi logici le strutture che oggi rappresentano la migliore soluzione trovata per ciascuno dei problemi che si incontrano nel sintetizzare un processore elettronico di dati.

Questo processore che il suo autore definisce *minimale* (basic, in inglese) è la “*macchina di Mano*”.

Pur non avendo documentazione tecnica vera e propria è descritta dall'autore con molto dettaglio ed è ormai adottato come sistema introduttivo in moltissimi corsi universitari di Informatica in tutto il mondo

## Compiti del processore

**L'interazione tra utente e processore avviene attraverso il “programma”.**

Il *programma* è una lista di istruzioni che specificano:

- le operazioni
- gli operandi
- l'ordine in cui le operazioni stesse vanno eseguite.

**Una istruzione è un codice numerico binario che specifica la sequenza di  $\mu$ istruzioni che bisogna eseguire su un dato.**

Istruzioni e dati sono registrati in memoria. La CPU legge ciascuna *istruzione* dalla memoria e la trasferisce nel registro di controllo.

La parte di controllo del processore traduce il codice binario in una successione di  $\mu$ operazioni che la parte operativa provvederà ad eseguire.

## Codice operativo

Ogni CPU ha un suo esclusivo “*repertorio di istruzioni*” (**instruction set**) che rappresenta per l’utente il risultato delle scelte elettroniche del progettista della struttura.

La CPU di un computer di uso generale (*general purpose computer*) è in grado, per definizione, di eseguire sui dati qualsiasi operazione logico-matematica che sia stata correttamente tradotta in una lista ordinata di istruzioni prese dal suo repertorio.

Il codice numerico di una istruzione è, di solito, diviso in due parti, ciascuna con un preciso compito.

La parte fondamentale di un codice di istruzione è quella “*operativa*”. Il *codice operativo* deve essere composto da un numero di bit  $n$ , sufficiente a codificare le diverse distinte operazioni che si vogliono eseguire ( $\leq 2^n$ ).

## Linguaggio macchina

Se supponiamo, ad esempio, che si voglia **un repertorio di 64 distinte operazioni**, allora il codice operativo deve essere di almeno **sei bit**.

Ciascuna operazione, di qualunque tipo essa sia, viene, di solito, associata ad una etichetta simbolica, a puro scopo mnemonico.

Se prendiamo in considerazione una istruzione che non può mancare, come **l’addizione aritmetica**, per essa quasi tutti i repertori d’istruzione usano il codice mnemonico “**ADD**”.

Nel sistema ipotizzato possiamo stabilire che, all’operazione ADD corrisponde, ad esempio, la combinazione binaria “110010”. Quando questo codice viene presentato alla logica di controllo, che come abbiamo visto può essere costituita da una ROM, esso viene “decodificato” per poter essere trasformato in una successione di comandi.

## Formato d'istruzione

Come abbiamo visto nel capitolo sulla sintesi dei sistemi di elaborazione il **codice di operazione** va semplicemente a costituire la parte più significativa dell'indirizzo della ROM che contiene il **microprogramma**. La parte meno significativa dell'indirizzo è, come è noto, costituita dalle **"condizioni"** e dalla combinazione di **"variabili interne"** che identifica il **"prossimo stato"** della rete.

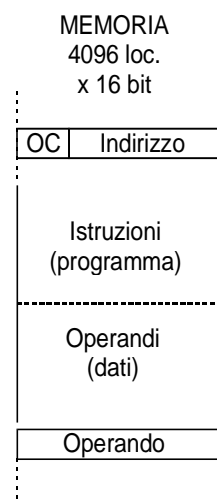
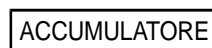
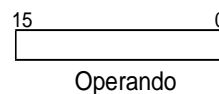
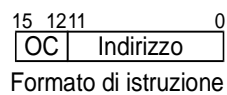
**Il codice dell'istruzione da eseguire costituisce solo una parte dell'informazione necessaria per effettuare una operazione**, esso va accompagnato dagli elementi necessari ad individuare l'**operando** o gli operandi su cui lavorare.

Gli operandi possono essere in **registri interni alla CPU** ed allora bisogna che il **"formato"** dell'istruzione preveda un **"campo"** dove si indica il registro dove si trova l'operando. Se l'operando è in memoria, allora il **formato** dell'istruzione deve prevedere un **campo indirizzo** con un numero di bit sufficiente a **"coprire"** la **"pagina di memoria"** dove si trovano gli operandi.

## Formato d'istruzione (2)

Processore con un **campo di indirizzamento** di 4096 locazioni (**12 bit d'indirizzo**). Il **"formato d'istruzione"** è a **16 bit**. Per il "codice d'operazione" (**Operative Code, OC**) rimangono 4 bit. L'**Accumulatore** è il registro principale della CPU e spesso si indica con la sigla **AC**.

In questo caso la parte di controllo della CPU, usando l'indirizzo a 12 bit, leggerebbe, nella parte della memoria riservata ai dati, un operando a 16 bit.



## Istruzioni “immediate”

Di solito in una CPU le operazioni aritmetiche e logiche avvengono tra un **operando che è contenuto nell’Accumulatore ed un altro specificato dall’istruzione.**

Nel caso di un codice operativo di **ADD**, per esempio, il numero letto in memoria all’indirizzo contenuto nell’istruzione verrebbe sommato al contenuto dell’**AC** ed il risultato rimarrebbe in **AC**.

Nel caso che il codice operativo si riferisca ad una operazione che non richieda un operando, **il campo indirizzo nel formato d’istruzione può essere usato per altri scopi.**

Non sono poche le istruzioni che non richiedono un secondo operando, basti pensare a tutte quelle che lavorano sul contenuto dell’Accumulatore, come: “**clear AC**” (azzera l’AC), “**complement AC**”, “**increment AC**”.

In alcune CPU il formato di istruzione è strutturato in modo che l’istruzione contenga, invece che l’indirizzo dell’operando, direttamente l’operando. In questo caso **l’istruzione viene definita “immediata”.**

## Indirizzo “diretto” ed “indiretto”

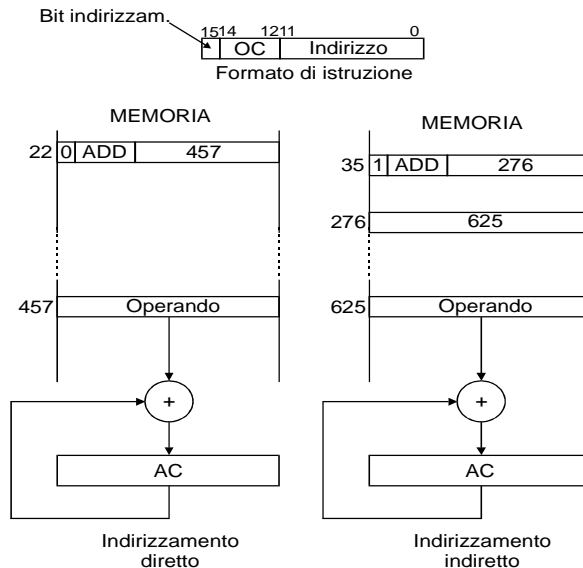
Quando l’istruzione contiene l’indirizzo dell’operando, si dice che l’indirizzamento è “**diretto**”.

Quando, invece, l’indirizzo contenuto nel formato d’istruzione non è quello dell’operando ma l’indirizzo della locazione di memoria che, a sua volta, contiene l’operando, l’indirizzamento si dice “**indiretto**”.

Viene naturale chiedersi il perché di questa complicazione che può sembrare, a prima vista, inutile. In realtà, a parte i grandi vantaggi che può dare in certe strutture di programmazione, l’*indirizzamento indiretto* può consentire un allargamento del campo di indirizzamento dell’operando. Nel caso visto prima l’indirizzo potrebbe passare da 12 a 16 bit.

**In generale una CPU evoluta deve poter usare entrambi i tipi di indirizzamento. Ciò significa che bisogna inserire nel formato d’istruzione un bit il cui valore booleano consenta di distinguere le istruzioni che usano l’indirizzamento *diretto* da quelle che prevedono quello *indiretto*.**

## Bit di indirizzamento



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

11

## L'indirizzo "vero"

Nel caso di sinistra, nella figura, il bit 15 è "0" ed il contenuto della memoria 457 viene sommato a quello dell'accumulatore. Nell'altro caso il bit 15 è "1" ed alla memoria 276 si trova l'indirizzo 625, dove si trova l'operando.

La locazione 625 si chiama *"indirizzo vero"* (*Effective Address, EA*).

**L'indirizzamento indiretto richiede due successivi accessi alla memoria.**

Nell'indirizzamento indiretto si usa spesso dire che la locazione indirizzata nell'istruzione, *"punta"* all'*indirizzo vero*.

Si può avere un diverso tipo di indirizzamento indiretto in cui nell'istruzione viene indicato un *"registro interno"* della CPU che *punta all'indirizzo vero*. Il vantaggio di questo tipo di *"indirizzamento tramite registro"* è che il tempo di accesso ad un registro interno è molto più breve che quello alla memoria.

**La CPU che presenteremo in questo ciclo di lezioni può utilizzare sia l'indirizzamento diretto che quello indiretto.**

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

12

## *La macchina di Mano*

Quelle fatte finora sono considerazioni a carattere generale. La descrizione della *macchina di Mano* comincia adesso, ma sarà evidente fra poco che alcuni dei numeri che sono stati usati negli esempi di indirizzamento, di codici operativi ecc.. sono proprio quelli della macchina di Mano.

La macchina di Mano si chiama così dal nome del suo ideatore M. Morris Mano, professore della Università dello stato della California a Los Angeles ed è descritta in tutti i suoi aspetti hardware e software nel volume, "Computer System Architecture", edizioni Prentice Hall International al quale è opportuno fare riferimento.

Il volume è disponibile presso la biblioteca della nostra Facoltà

## **Registri della CPU**

Il processore centrale deve avere un registro per contenere l'istruzione (codice operativo + indirizzo). Questo registro si chiama "**registro istruzione**" (**Instruction Register, IR**)

**Il programma è fatto di una successione di istruzioni che sono immagazzinate in memoria, di solito, in locazioni successive in una zona riservata a questo scopo**, il processore legge la prima, la scompone nella corrispondente  $\mu$ sequenza ed esegue le varie  $\mu$ istruzioni che la compongono.

Tutto ciò richiede un apposito registro, in grado di elaborare l'indirizzo della prossima istruzione. In pratica, il registro quasi sempre deve semplicemente "**puntare**" alla locazione di memoria successiva a quella dell'istruzione che si sta eseguendo, e, pertanto, viene detto "**Program Counter, PC**".

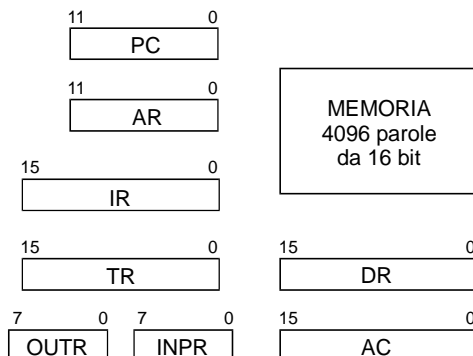
**E' chiaro che è utile che il processore abbia almeno un registro dove poter registrare un dato numerico, un altro dove poter conservare un indirizzo ....e così via.**

## I registri del processore

Simbolo Registro	Numero bits	Nome del registro	Funzione
DR	16	Registro dati	Contiene l'operando letto in memoria
AR	12	Registro indirizzo	Contiene l'indirizzo cui accedere
AC	16	Accumulatore	Registro operativo
IR	16	Registro istruzione	Contiene il codice dell'istruzione
PC	12	Contatore di programma	Conserva l'indirizzo dell'istruzione
TR	16	Registro temporaneo	Contiene un dato temporaneo
INPR	8	Registro d'ingresso	Contiene un carattere in ingresso
OUTR	8	Registro d'uscita	Contiene un carattere in uscita

La memoria ha 4096 locazioni da 16 bit. Il *formato delle istruzioni* è quello visto in precedenza con un **campo di indirizzamento diretto di 12 bit**, quanti ne servono per indirizzare le locazioni a disposizione. Il **codice operativo utilizza tre bit** ed **1 bit è usato per indicare il tipo di indirizzamento**. Questo spiega perché il **Registro Indirizzo AR** ed il **Program Counter PC** hanno **12 bit**.

## Il programmer's model del sistema



I registri DR e AR servono per i collegamenti con la memoria, il registro TR serve per un eventuale risultato intermedio, mentre i registri INPR ed OUTR servono, rispettivamente, per ricevere o mandare un carattere di 8 bit ad un periferico.

Il *processore (CPU)* è rappresentato dai suoi registri. C'è poi la *memoria*.

La struttura, che nell'introduzione abbiamo chiamato *processore di ingresso/uscita (I/O processor)*, è rappresentato dai due registri INPR ed OUTR.

## Collegamento tra i registri

La parte operativa del sistema di elaborazione che stiamo componendo è formata da otto registri, oltre alla memoria e quasi tutti questi elementi debbono poter trasferire il proprio contenuto verso tutti gli altri.

Una rete che possa permettere questi trasferimenti in completa libertà sarebbe oltremodo onerosa in termini di componenti logici

**Lo schema più efficiente di collegamento è sicuramente quello di un bus dati che colleghi tutti o quasi i registri e la memoria.**

**Le uscite di sette degli otto registri sono connesse al bus costituito da 16 linee dati bidirezionali e da tre linee di selezione  $S_2, S_1, S_0$  che individuano il dispositivo abilitato ad immettere sulle linee le uscite.** Porte AND decodificano l'indirizzo sulle tre linee S ed abilitano i driver "tristate" di uscita del registro selezionato. Ovviamente i registri che hanno 12 bit sono collegati solo alle linee dati da 0 ad 11. Quando questi registri sono abilitati a porre i loro bit sul bus, **le linee da 12 a 15 sono tenute a "0"**.

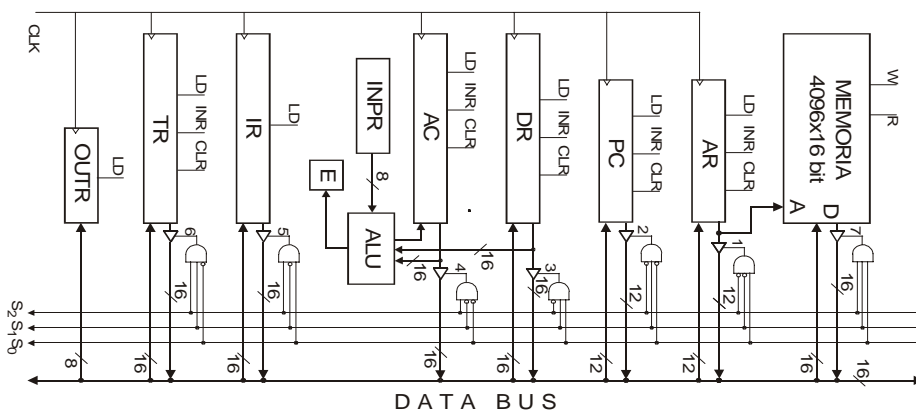
Se ad esempio,  $S_2S_1S_0$  sono 0 1 1 i valori booleani dei 16 bit del registro DR si propagano sulle linee dati del bus.

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

17

## Il bus del processore



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

18

## Caratteristiche dei registri

Cinque registri hanno tre ingressi di controllo: **LD (load)**, **INR (increment)**, e **CL (clear)**. Questo tipo di registro non è altro che un **contatore binario a caricamento parallelo con possibilità di azzeramento sincrono**. L'incremento si ottiene abilitando per un ciclo l'ingresso di conteggio sul quale è collegato il clock.

Due dei registri hanno solo l'**abilitazione al caricamento parallelo LD**. **Gli ingressi e le uscite dati della memoria sono connessi al bus ma gli ingressi per l'indirizzo della memoria sono collegati al registro AR**. L'unico modo per indirizzare la memoria è, quindi, quello di trasferire l'indirizzo desiderato in AR. **Questa tecnica serve per evitare che il bus debba avere delle altre linee appositamente previste per l'indirizzo**. La tecnica di indirizzamento del processore che stiamo idealmente costruendo prevede che l'indirizzo al quale la memoria si deve posizionare debba essere trasferito in AR prima dell'operazione di lettura o scrittura.

**Per sintetizzare il tipo di approccio si dice, di solito, che si effettua il "multiplexing" degli indirizzi sulle linee dati.**

## Caricamento dell'accumulatore

**Il dato da scrivere in memoria può essere fornito da qualsiasi registro, così come il dato letto in memoria può essere trasferito in qualsiasi registro, salvo l'AC.**

Il caricamento dei 16 bit dell'accumulatore avviene attraverso l'unità aritmetica e logica, ALU che ha tre diversi gruppi di ingressi. Un gruppo di 16 ingressi viene dalle uscite dello stesso AC ed è usato per ri-introdurre nell'accumulatore un dato sottoposto nella ALU ad una operazione logica od aritmetica ad un solo operando, come la complementazione od uno scorrimento (divisione o moltiplicazione per 2). Il secondo gruppo di ingressi, sempre da 16 bit, viene dal registro dati DR.

Questi due gruppi di ingressi da 16 bit vengono usati in simultanea durante le operazioni aritmetiche e logiche a due operandi. **Il risultato di una operazione aritmetica o logica (ad esempio una addizione aritmetica, codice mnemonico ADD) viene sempre trasferito in AC ed un eventuale riporto complessivo inviato al flip-flop E (estensione dell'AC).**

Il terzo gruppo di ingressi è ad 8 bit e viene dall'INPR.

## Operazioni contemporanee

L'uso dell'INPR e dell'OUTR sarà illustrato in seguito.

Si tenga presente che il contenuto di un qualunque registro può essere messo sul bus e su di esso può essere effettuata una delle operazioni aritmetiche o logiche del repertorio della ALU in un solo periodo di clock. La transizione finale del ciclo di clock registrerà il risultato dell'operazione nell'AC.

Per esempio, **le due operazioni,**

**$DR \leftarrow AC$  ed  $AC \leftarrow DR$**

**possono essere eseguite contemporaneamente.**

Selezionando il codice  $S_2 S_1 S_0 = 1 0 0$ , il contenuto dell'AC viene messo sul bus. L'abilitazione dell'ingresso LD del registro DR permette il caricamento in DR del dato. Contemporaneamente l'abilitazione all'ingresso nella ALU del contenuto di DR consente di trasferirlo nell'AC nello stesso periodo di clock.

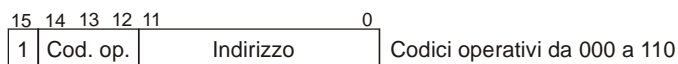
**E' del tutto ovvio che tutti i registri sono costituiti da flip-flop del tipo master/slave.**

## I formati di istruzione

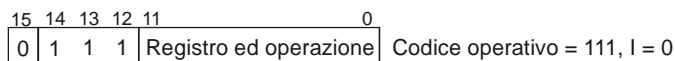
Il computer essenziale che stiamo ipotizzando, ha tre formati di istruzione. Ciascun formato ha 16 bit. **Il campo per il codice operativo è formato dai tre bit dalla posizione 12 a quella 14.** Il significato degli altri 13 bit dipende dal codice operativo.

- Il formato delle istruzioni per l'accesso alla memoria (*memory-reference instructions*) come abbiamo già visto in precedenza usa i 12 bit da 0 ad 11 per specificare un indirizzo ed il bit all'estrema sinistra (posizione 15) per specificare la *modalità di indirizzamento*, **I**. Quando **I = 0** significa che l'indirizzo va inteso "*diretto*". Quando, invece **I = 1**, l'indirizzo è da considerare "*indiretto*".
- Il formato delle istruzioni "*register-reference*" è contraddistinto dal codice operativo "1 1 1" e dal bit 15 posto a "0". Le istruzioni "*relative ai registri*" specificano una operazione od un test sul contenuto del **registro AC e del flip-flop E**. In queste istruzioni non è richiesto un secondo operando da prelevare in memoria e, pertanto **il campo indirizzo (gli altri 12 bit) sono a disposizione per specificare dettagliatamente l'operazione od il test da effettuare.**

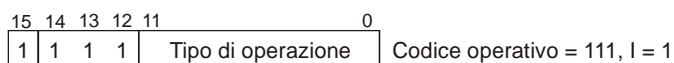
## I formati di istruzione (2)



Formato delle istruzioni "relative alla memoria"



Formato delle istruzioni "relative ai registri"



Formato delle istruzioni di ingresso ed uscita

- Il terzo formato è quello delle **"istruzioni di ingresso/uscita" (I/O instructions)**. E' caratterizzato anch'esso dall'**opcode 1 1 1** ma ha il **bit in posizione 15 (campo I) posto ad 1**. Anche in questo tipo d'istruzioni i **rimanenti 12 bit non servono per l'accesso alla memoria e vengono usati per specificare in dettaglio il tipo di operazione di I/O o di test da effettuare**.

## Riconoscimento istruzioni

**Il tipo d'istruzione è riconosciuto dalla rete sequenziale di controllo del processore esaminando i quattro bit dalla posizione 12 alla 15.** Se i tre bit da 12 a 14 non sono tutti ad 1 l'istruzione è del tipo "relativo alla memoria" ed allora si passa ad esaminare il bit in posizione 15 per stabilire il "modo" in cui bisogna usare i 12 bit di indirizzo. Se l'opcode è invece 1 1 1, allora il valore del bit 15 deciderà se si tratta di una istruzione "relativa al registro" (bit 15 a "0") od una istruzione di ingresso/uscita (bit 15 a 1). Il bit 15 si indica con I anche in questi casi in cui non indica il "modo" di indirizzamento.

**Le sigle mnemoniche per facilitare il compito dei programmatori sono di tre lettere.** Per ciascuna istruzione è dato il codice esadecimale, che per le 7 istruzioni il cui "opcode" va da 0 a E si riferisce ai quattro bit da 12 a 15. I codici sono suddivisi in base al valore di I.

Gli altri dodici bit nelle operazioni "register reference" ed nelle istruzioni di Input/Output, in pratica sono parte integrante del codice operativo e consentono di ottenere ben 18 operazioni distinte da una sola combinazione di opcode.

## Repertorio istruzioni (1) ( *relative alla memoria* )

Simbolo	Codice esadecimale		Descrizione
	I = 0	I = 1	
AND	0xxx	8xxx	AND tra parola della memoria e AC
ADD	1xxx	9xxx	Addizione parola della memoria ad AC
LDA	2xxx	Axxx	Caricamento parola dalla memoria in AC
STA	3xxx	Bxxx	Deposito del contenuto di AC in memoria
BUN	4xxx	Cxxx	Trasferimento incondizionato
BSA	5xxx	Dxxx	Trasfer. e salvataggio indirizzo di ritorno
ISZ	6xxx	Exxx	Incremento e salto se il valore è zero

## Repertorio istruzioni (2) ( *relative ai registri* )

Simbolo	Codice esadecimale		Descrizione
	I = 0	I = 1	
CLA	7800		Azzeramento di AC
CLE	7400		Azzeramento di E
CMA	7200		Complementazione di AC
CME	7100		Complementazione di E
CIR	7080		Circolazione verso destra di AC ed E
CIL	7040		Circolazione verso sinistra di AC ed E
INC	7020		Incremento di AC di una unità
SPA	7010		Salto prossima istruzione se AC è positivo
SNA	7008		Salto prossima istruzione se AC è negativo
SZA	7004		Salto prossima istruzione se AC è zero
SZE	7002		Salto prossima istruzione se E è zero
HLT	7001		Blocco della elaborazione

## Repertorio istruzioni (3) ( *ingresso / uscita* )

Simbolo	Codice esadecimale	Descrizione
INP	F800	Trasferimento carattere in ingresso ad AC
OUT	F400	Trasferimento carattere da AC in uscita
SKI	F200	Salto se la "flag" d'ingresso è ON
SKO	F100	Salto se la "flag" d'uscita è ON
ION	F080	Attivazione dell'interruzione
IOF	F040	Disattivazione dell'interruzione

## Completezza di un set d'istruzioni

Il set di istruzioni di un computer deve consentire all'utente di costruire programmi in "*linguaggio macchina*" per calcolare qualsiasi funzione che si sa essere calcolabile.

La completezza di un set d'istruzioni richiede un numero sufficiente di istruzioni dei seguenti tipi:

- 1. Istruzioni aritmetiche, logiche e di scorrimento.**
- 2. Istruzioni per spostare informazioni dai registri e la memoria e viceversa.**
- 3. Istruzioni per dirigere l'esecuzione del calcolo e verificarne lo stato.**
- 4. Istruzioni per l'ingresso e l'uscita delle informazioni.**

## Condiderazioni sul set di istruzioni

**Il set di istruzioni è minimale.** Basti considerare che comprende una sola istruzione aritmetica, quella di addizione (**ADD**) e due altre correlate, quella di complementazione dell'AC (**CMA**) e quella di incremento dello stesso (**INC**). con queste istruzioni è comunque possibile eseguire somme e sottrazioni di numeri interi con segno, nella rappresentazione in "complemento a 2".

Le istruzioni di *circolazione*, **CIR** e **CIL** servono nei programmi per effettuare operazioni aritmetiche più complesse come moltiplicazioni e divisioni e per altre operazioni di tipo logico.

Le istruzioni propriamente **logiche** sono **AND**, **CMA**, **CLA**. L'operazione di **AND** e quella di complementazione compongono l'operazione **NAND**.

L'istruzione di caricamento dell'accumulatore (**LDA**) è fondamentale per prelevare in memoria un dato, così come quella che consente di registrare in memoria il contenuto dell'AC (**STA**).

## Condiderazioni sul set di istruzioni (2)

Le istruzioni di "*branch*", **BUN**, **BSA**, **ISZ**, con le istruzioni di "*salto*", riassumono tutte le possibilità che l'utente ha di dirigere l'elaborazione e di verificarne lo stato.

Le istruzioni **INP** ed **OUT** consentono l'ingresso e l'uscita delle informazioni tra il sistema ed i dispositivi esterni.

**L'insieme delle istruzioni del nostro sistema è completo, ma certamente non consente una elaborazione efficiente**, perché mancano istruzioni di uso molto frequente quali quelle per eseguire direttamente, sottrazioni, moltiplicazioni, l'OR e l'OR-esclusivo, che nel nostro sistema debbono essere realizzate con programmi basati sulle istruzioni elementari.

**D'altra parte un set d'istruzioni più ricco avrebbe notevolmente complicato la struttura del processore.**

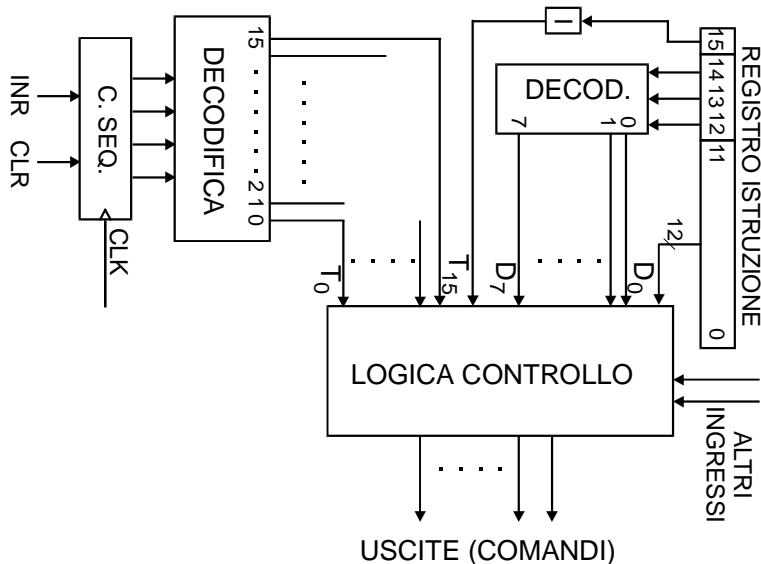
## Sincronizzazione e controllo

La sincronizzazione di tutti i registri ed i flip-flop, sia della parte operativa, che di quella di controllo è effettuata dal segnale di orologio generato da un unico oscillatore, di solito controllato da un quarzo. Il segnale viene distribuito a tutti gli elementi del sistema ma non influenza lo stato logico dei flip-flop, se non è attivo l'ingresso  $e$ , per i registri, se non è attivo almeno uno degli ingressi di controllo, oltre agli ingressi dati.

I segnali di gestione della struttura operativa della CPU sono prodotti dall'*unità di controllo* che, come è noto, può essere costituita da una rete sequenziale sincrona tradizionale, formata da una rete combinatoria e da un certo numero di flip-flop di tipo D, oppure da una rete sequenziale microprogrammata basata su una memoria a sola lettura. Quest'ultima soluzione presenta il vantaggio di consentire, in caso di necessità in rapido aggiornamento del sistema, semplicemente sostituendo la ROM di microprogrammazione.

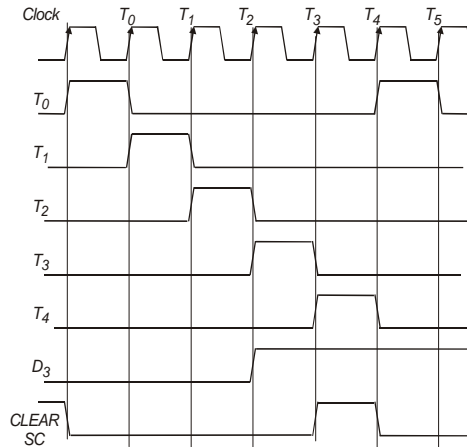
Illustreremo, però, una parte di controllo basata su funzioni combinatorie (in inglese si dice "hardwired") perché è più intuitiva.

## Logica di controllo "hardwired"



La **prima transizione** attiva del clock, dopo che è stato reso inattivo il segnale di “clear” del contatore di sequenza, trova il contenuto del contatore a 0 0 0 0 e, pertanto, per l'intero periodo del segnale di cadenza risulterà **attiva l'uscita  $T_0$**  del decoder. Alla **successiva transizione** il contatore andrà a 0 0 0 1 e sarà **attiva l'uscita  $T_1$**  e così successivamente quella  $T_2$  e poi quella  $T_3$  e poi, ancora,  $T_4$ , ma, se durante quest'ultimo periodo di clock viene attivato l'ingresso di controllo di “clear” del contatore, **la successiva uscita attiva non sarà,  $T_5$ , ma di nuovo  $T_0$** , ed il ciclo ricomincia per arrestarsi al nuovo segnale di CLR del contatore.

## Il “timing” del sistema



## Gestione “logica” del tempo

Il circuito costituito dal contatore SC e dalla decodifica è in grado di generare una serie di **segnali temporalmente disgiunti** che può arrivare fino ad un massimo di 16 impulsi.

Un dettaglio importante da notare nella figura 7 è che il segnale di “clear” del contatore (SC) è determinato dalla **coincidenza (AND) del segnale  $D_3$  e del segnale  $T_4$** . Questa possibilità logica suggerisce due considerazioni fondamentali.

- La prima è che è **possibile utilizzare una informazione logica** ( $D_3$  è attivato da un codice operativo) **per generare una segnale di temporizzazione**.
- La seconda è che **si possono generare cicli temporali la cui durata in termini di periodi di clock è funzione del codice operativo**.

Quest'ultima è una possibilità intuitivamente fondamentale perché il sistema possa eseguire microsequenze più o meno complesse o sincronizzare operazioni elementari di durata diversa

## Sincronizzazione di sistemi eterogenei. *Discretizzazione del tempo*

**Un esempio semplice e concreto:**

**Un ciclo di lettura o scrittura in memoria**, può essere iniziato sulla transizione positiva del clock ma, se esso durasse più di un periodo di clock, l'operazione successiva, per esempio, la registrazione del dato letto in un registro, non potrebbe essere sincronizzata dalla successiva transizione attiva del clock, per cui, **o si assume che i cicli di memoria sono di durata inferiore al periodo di clock, o bisogna, come avviene in molti sistemi, prevedere la possibilità di lasciare trascorrere un ciclo ulteriore di clock del processore prima di proseguire l'operazione.**

**Nel caso del nostro computer essenziale, per semplicità, supporremo che questo non sia necessario.**

## Il "timing" nel linguaggio RTL

La constatazione che informazioni logiche possono tradursi in segnali temporali, da un preciso significato logico alla notazione

$$T_0: AR \leftarrow PC$$

**Essa significa che il trasferimento del contenuto del PC nel registro indirizzo avverrà al tempo  $T_0$** , inteso come primo periodo di clock di un ciclo di esecuzione di un'istruzione che durerà un numero di periodi diverso, a seconda dell'istruzione.

**Naturalmente perché l'operazione abbia luogo, durante il periodo  $T_0$ , dovrà essere selezionato l'indirizzo dei driver tristate d'uscita del PC (0 1 0) per mettere il suo contenuto sul bus ed essere abilitato l'ingresso di controllo LD del registro AR.**

## Cicli istruzione

Un programma residente nella memoria di un computer è costituito da una sequenza di istruzioni. La sua esecuzione fa sì che la struttura **esegua un ciclo per ogni istruzione** che a sua volta è **costituito in una successione di fasi**. Nel computer elementare che stiamo analizzando, **ciascun ciclo è composto dalle seguenti fasi**:

1. **Prelievo di una istruzione dalla memoria.**
2. **Decodifica l'istruzione.**
3. **Lettura l'indirizzo vero dalla memoria, se l'istruzione usa un indirizzo indiretto.**
4. **Esecuzione dell'istruzione.**

**Completato il passo 4, il controllo ritorna al punto 1 per prelevare, decodificare ed eseguire la prossima istruzione. Questo processo continua indefinitamente fino a quando nel programma non viene incontrata una istruzione di HALT.**

## Prelievo e decodifica

Quando il sistema comincia a funzionare, dopo la procedura di inizializzazione, la prima operazione che viene eseguita è quella di caricare nel PC l'indirizzo della prima istruzione del programma.

Il *contatore di sequenza* viene azzerato e, questo, sappiamo, che genererà un segnale di "*timing*"  $T_0$ . Dopo ciascun impulso di clock SC è incrementato di una unità e questo produce una sequenza ordinata di segnali temporali  $T_0, T_1, T_2, ..$  e così via.

Le  $\mu$ operazioni per le fasi di *prelievo* e *decodifica* (*fetch* and *decode*) possono essere specificate dalla seguente sequenza di  $\mu$ operazioni espresse in linguaggio RTL.

$T_0: AR \leftarrow PC$

$T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$

$T_2: D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$



## I dettagli della sequenza

Per predisporre il percorso dei dati in modo che il contenuto del PC si trasferisca nell'AR, bisogna effettuare le seguenti operazioni logiche:

1. **Porre il contenuto del PC sul bus impostando  $S_2S_1S_0 = 0\ 1\ 0$**
2. **Trasferire il contenuto del bus all'AR abilitando il suo ingresso di LD.**

La transizione attiva del clock che produce il segnale  $T_0$  dà il via al trasferimento.

Il secondo passo della sequenza prevede l'operazione:

$$T_1: \text{IR} \leftarrow M[\text{AR}], \text{PC} \leftarrow \text{PC} + 1$$

Bisogna utilizzare il segnale di temporizzazione  $T_1$  per effettuare le seguenti connessioni sul bus:

1. **Abilitare l'ingresso di lettura della memoria.**
2. **Porre il contenuto della memoria sul bus ponendo  $S_2S_1S_0 = 1\ 1\ 1$ .**
3. **Trasferire il contenuto del bus all'IR abilitando l'ingresso di LD dello stesso.**
4. **Incrementare il PC abilitando l'ingresso di INR del PC.**

La transizione del clock che porta  $T_1 = 1$  avvia l'operazione di lettura ed incremento

## Determinazione dell'indirizzo vero

L'operazione di **decodifica** dell'istruzione impegna il tempo  $T_2$ .

**Durante il periodo  $T_3$  avviene il riconoscimento del tipo di istruzione.**

- $D_7 = 0$ : l'istruzione è di "*accesso in memoria*".
- $D_7 = 1, I = 0, I = 0$ : istruzione "*relativa ai registri*",
- $D_7 = 1, I = 1$ : istruzione "*ingresso/uscita*".

**Se  $D_7 = 1$ , l'istruzione viene eseguita e si esaurisce nel tempo  $T_3$ .**

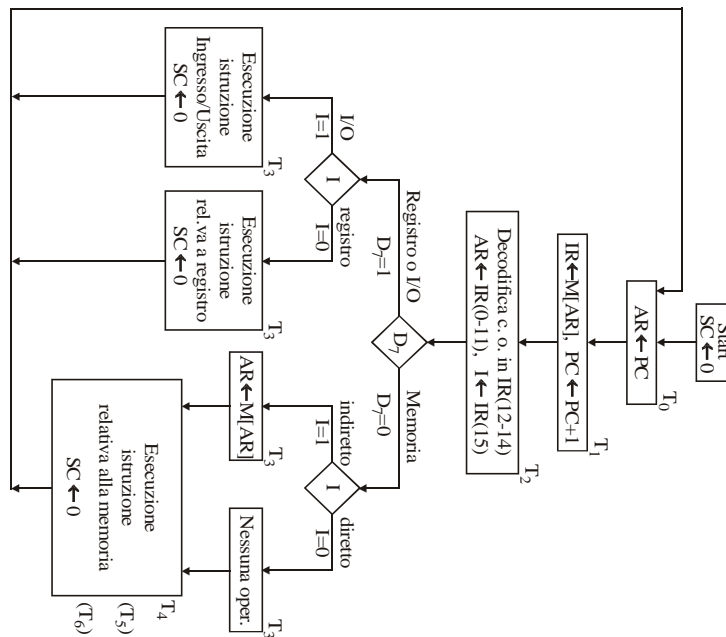
- $D_7 = 0, I = 0$ : **indirizzamento "diretto"**. L'indirizzo dell'operando è già disponibile in AR e **durante  $T_3$  non si fa niente**, in attesa di accedere alla memoria nel periodo successivo.
- $D_7 = 0, I = 1$ : **indirizzamento "indiretto"**. **Durante  $T_3$  si esegue, secondo le solite modalità, l'operazione:**

$$\text{AR} \leftarrow M[\text{AR}]$$

Si abilita la memoria alla lettura della locazione il cui indirizzo è in AR e, selezionando l'indirizzo sul bus della memoria, si pone il contenuto della locazione sulle linee dati.

L'abilitazione dell'ingresso di LD del registro, consentirà la registrazione in AR dello stato logico delle 12 linee meno significative.

## Flusso dei cicli istruzione



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

43

## Il periodo di clock $T_3$

In sintesi le quattro possibili operazioni da effettuare, in alternativa fra loro, al tempo  $T_3$  si possono indicare simbolicamente così:

$D_7^* I T_3$ :  $AR \leftarrow M[AR]$

$D_7^* I^* T_3$ : Niente

$D_7 I^* T_3$ : Esecuzione di una istruzione “register-reference”

$D_7 I T_3$ : Esecuzione di una istruzione “ingresso/uscita”

E' chiaro che la condizione  $D_7^* T_3$  comporta la necessità di almeno un ulteriore periodo di clock,  $T_4$  per l'esecuzione dell'istruzione, per cui l'operazione  $SC \leftarrow 0$  che determina la ripartenza della sequenza temporale da  $T_0$  è posticipata almeno di un ciclo di clock.

Si osservi che è stata adottata la tecnica di non dichiarare esplicitamente l'incremento di SC ad ogni periodo di clock:

$$SC \leftarrow SC + 1$$

considerandolo implicito.

Questa scelta impone, ovviamente, di dichiarare l'operazione di azzeramento del SC.

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

44

## Le istruzioni *relative ai registri*

Sono riconosciute dal controllo dalle condizioni  $D_7 = 1$  e  $I = 0$ . Questo tipo di istruzione usa i bit da 0 ad 11 del formato per specificare compiutamente l'operazione da effettuare (il repertorio prevede 12 istruzioni di questo tipo).

I 12 bit sono quelli che costituiscono il "*campo indirizzo*" nelle istruzioni *relative alla memoria*. Essi sono disponibili in **IR(0-11)** e vengono, anche, **trasferiti in AR al tempo  $T_2$** .

Le **funzioni di controllo e le microoperazioni** per le istruzioni relative ai registri sono elencate nella **tabella che segue**.

**Le istruzioni vengono eseguite al tempo  $T_3$ . Alla successiva transizione attiva del clock il registro SC viene azzerato e comincia un altro ciclo istruzione.**

Le **funzioni di controllo** per queste istruzioni **sono legate alla condizione booleana  $D_7 I^* T_3$**  che per sintetizzare possiamo indicare col simbolo  $r$ .

**Essendo i bit in IR(0-11) dodici, ciascuno di essi può essere associato ad una delle dodici istruzioni relative ai registri.**

## Condizioni logiche delle istruzioni sui registri

$D_7 I^* T_3 = r$  (condizione comune a tutte le istruzioni relative ad un registro)

$IR(i) = B_i$  [bit in IR(0-11) che specifica l'operazione]

	$r$ : $SC \leftarrow 0$	Azzeramento di SC
CLA	$r B_{11}$ : $AC \leftarrow 0$	Azzeramento di AC
CLE	$r B_{10}$ : $E \leftarrow 0$	Azzeramento di E
CMA	$r B_9$ : $AC \leftarrow AC^*$	Complementazione di AC
CME	$r B_8$ : $E \leftarrow E^*$	Complementazione di E
CIR	$r B_7$ : $AC \leftarrow shr AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circolazione a destra
CIL	$r B_6$ : $AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circolazione a sinistra
INC	$r B_5$ : $AC \leftarrow AC + 1$	Incremento di AC
SPA	$r B_4$ : if $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Salto se segno AC è posit.
SNA	$r B_3$ : if $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Salto se segno AC è neg.
SZA	$r B_2$ : if $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Salto se AC è zero
SZE	$r B_1$ : if $(E = 0)$ then $(PC \leftarrow PC + 1)$	Salto se E è zero
HLT	$r B_0$ : $S \leftarrow 0$ (S è il F.F. Start/Stop)	Arresto del computer

## Condiderazioni sulla tabella

Se indichiamo il generico bit di IR con  $B_i$  (con  $i$  che va da 0 ad 11) le funzioni di controllo delle istruzioni relative ai registri possono essere sintetizzate con:  $r B_i$ .

Per esempio, l'istruzione **CLA** ha un codice esadecimale pari a: **7800**, come si può facilmente verificare inserendo nella griglia di formato le condizioni  $I = 0$ , Opcode = 1 1 1 e  $B_{11} = 1$ :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
					7		8		0				0		

### Le istruzioni operano sui registri AC ed E.

Le prime sette effettuano operazioni di azzeramento, complementazione, circolazione od incremento.

Le successive quattro sono di salto condizionato. Le condizioni di segno operano sul bit AC(15), quella di AC = 0, sulla funzione OR del contenuto di tutti i flip-flop.

**La istruzione di HLT azzerava un flip-flop, S che blocca il registro SC.**

## Istruzioni relative alla memoria

Il repertorio di istruzioni relative alla memoria è già stato descritto ma è praticamente impossibile descrivere efficacemente a parole una istruzione.

**Il modo più semplice e, l'unico efficace, per descrivere un repertorio di istruzioni è quello di usare il linguaggio RTL.** La tabella elenca le sette istruzioni "memory-reference" del repertorio della macchina e la relativa descrizione in RTL.

Symbol	Operation decoder	Symbolic description
AND	$D_0$	$AC \leftarrow AC \wedge M[AR]$
ADD	$D_1$	$AC \leftarrow AC + M[AR]$
LDA	$D_2$	$AC \leftarrow M[AR]$
STA	$D_3$	$M[AR] \leftarrow AC$
BUN	$D_4$	$PC \leftarrow AR$
BSA	$D_5$	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	$D_6$	$M[AR] \leftarrow M[AR] + 1$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

## Fase “*execute*” delle istruzioni di memoria

Queste istruzioni, come abbiamo già visto, **non si esauriscono col tempo  $T_3$**  ma, anzi, cominciano l’esecuzione al tempo  $T_4$  e, talvolta, utilizzano anche il periodo di clock successivo  $T_5$  ed, **in un caso, perfino  $T_6$** .

Ciò è dovuto al fatto che per queste istruzioni è necessario prelevare in memoria uno degli operandi per trasferirlo in un registro prima di eseguire l’operazione richiesta.

Nella tabella sono riportate le operazioni relative alla fase successiva a quelle di “fetch”, “decodifica” e “prelievo in memoria dell’indirizzo vero”. Infatti, non c’è nessun riferimento al tipo di indirizzamento utilizzato per l’istruzione.

Naturalmente le istruzioni vanno sviluppate in microsequenze.

La microsequenza, ad esempio, corrispondente alla fase esecutiva dell’istruzione **AND**, che opera tra i bit corrispondenti dell’AC e quelli della parola letta in memoria all’*indirizzo vero* che si trova in AR, è:

**$D_0 T_4: DR \leftarrow M[AR]$**

**$D_0 T_5: AC \leftarrow AC DR, SC \leftarrow 0$**

## Le istruzioni di “*branch*”

La parola inglese “branch” si può tradurre “trasferimento” quando l’istruzione è incondizionata, “bivio” quando, invece, è condizionata.

L’istruzione di “*trasferimento*” tipica è **BUN** che significa: **Branch UNconditionally**. Mandava in esecuzione l’istruzione indicata dal suo “indirizzo vero”. La corrispondente microoperazione è:

**$D_4 T_4: PC \leftarrow AR, SC \leftarrow 0$**

**BSA** è la sigla mnemonica dell’istruzione **Branch and Save Return Address**, che consente di inserire nel programma un “sottoprogramma” (subroutine) che esegue una specifica operazione complessa.

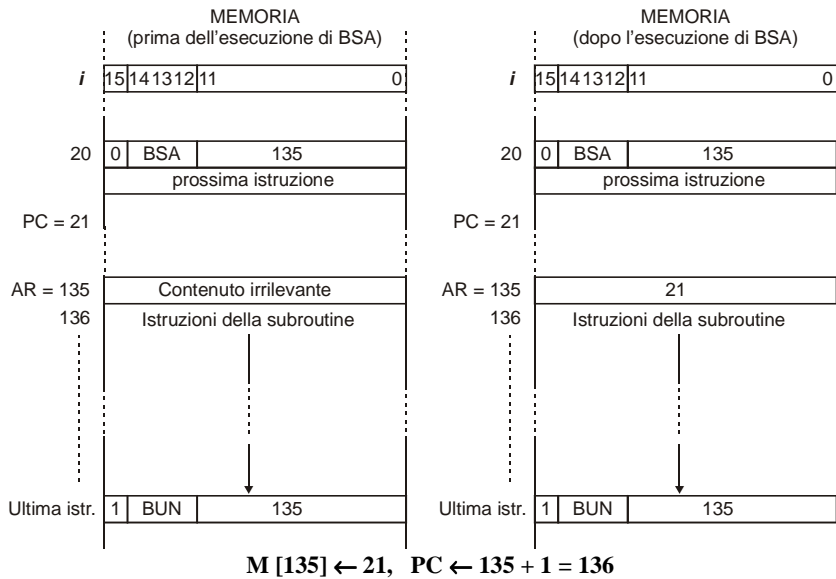
L’esecuzione di questa istruzione deposita in memoria l’indirizzo della prossima istruzione, che è disponibile nel PC, in una locazione di memoria specificata dall’indirizzo vero dell’istruzione.

Nel PC viene, poi, inserito lo stesso indirizzo vero, incrementato di una unità. In sintesi:

**$D_5 T_4: M[AR] \leftarrow PC, PC \leftarrow AR + 1$**

**$D_5 T_5: PC \leftarrow AR, SC \leftarrow 0$**

## Trasferimento a subroutine

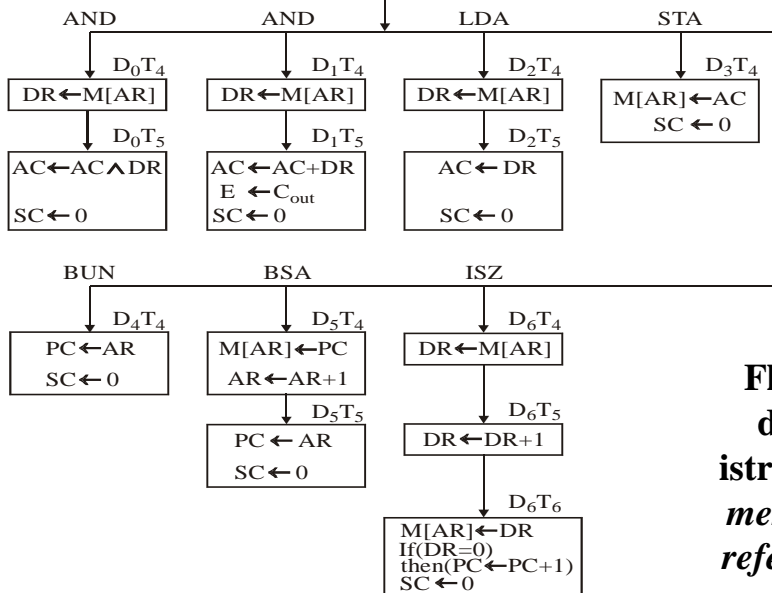


Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

51

### Istruzioni "relative alla memoria"



**Flusso  
delle  
istruzioni  
memory-  
reference**

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

52

## L'istruzione ISZ (Increment and Skip if Zero)

Si adopera per effettuare una operazione un determinato numero di volte. Si incrementa di una unità la parola specificata dall'indirizzo vero e, se dopo l'incremento essa risulta 0, non viene eseguita la prossima istruzione.

Si usa memorizzando un numero negativo (in complemento a 2) in una locazione di memoria ed ogni volta che si esegue il ciclo esso viene incrementato di una unità positiva. Dopo il prescritto numero di cicli, l'istruzione seguente, che rimandava il processore ad eseguire il ciclo viene saltata e l'elaborazione riprende da quella successiva.

La sequenza di microoperazioni corrispondente è:

$D_6 T_4: DR \leftarrow M[AR]$

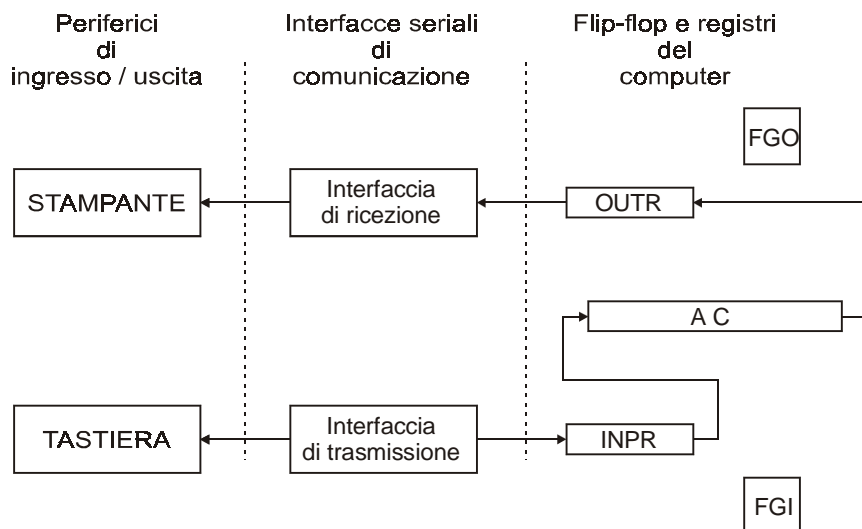
$D_6 T_5: DR \leftarrow DR + 1$

$D_6 T_6: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$

Si osservi che non essendo possibile incrementare direttamente il contenuto di una parola nella memoria è necessaria estrarla, incrementarla e, quindi, riscriverla allo stesso indirizzo.

**Questa è l'istruzione che richiede più periodi di clock di lavoro all'unità di controllo e ne utilizza solo sette. Questo significa che aver scelto il SC a 4 bit (16 periodi di clock) consente ampi margini di manovra sui tempi.**

## La configurazione di ingresso uscita



## Il concetto d'*interfaccia*

Il sistema comunica con l'esterno attraverso i due registri **INPR** ed **OUTR** che comunicano attraverso due "*interfacce*" **seriali** rispettivamente con una tastiera ed una stampante.

I due registri comunicano invece in **parallelo** con l'Accumulatore all'interno del sistema.

L'AC ha 16 bit, i registri sono collegati agli 8 bit meno significativi

Si definisce genericamente "*interfaccia*" **un circuito che assicura la compatibilità logica tra due sistemi.**

L'*interfaccia* si fa carico di tutti i problemi di interconnessione, da quelli di filosofia strutturale a quelli di compatibilità elettrica. In tempi ormai remoti, quando i livelli di alimentazione elettrica dei circuiti erano spesso diversi **l'interfaccia doveva perfino adattare le codifiche elettriche dei due simboli booleani.**

## L'*interfaccia seriale*

Nel caso specifico, l'interfaccia della tastiera non è altro che un **registro a scorrimento a caricamento parallelo** per ricevere il codice ASCII del tasto ed **uscita seriale** per trasmetterlo.

Il flip-flop **FGI** svolge la funzione di "*flag*" (bandiera).

Il suo passaggio ad "1" significa che il dato è pronto e può cominciare il trasferimento seriale.

Dopo che FGI è stato "attivato" il contenuto del registro non può più essere cambiato fino a quando non verrà "azzerato" dalla esecuzione della istruzione INP.

Il reset di FGI è il segno che può partire il successivo trasferimento.

**L'interfaccia tra OUSR e la stampante funziona in maniera perfettamente speculare.**

## Le istruzioni di I/O

Le istruzioni di I/O sono caratterizzate dalla prima cifra esadecimale del codice F, visto che hanno ad 1 sia il bit I IR(15) che i 3 bit IR(12-14), che vengono decodificati in D<sub>7</sub>.

La condizione logico temporale comune a tutte le istruzioni è:

$$D_7 I T_3 = p$$

Le sei istruzioni utilizzano, per la specificazione della funzione, i bit di IR(0-11) che vanno da B<sub>6</sub> (IOF) ad B<sub>11</sub> (INP).

---

D<sub>7</sub> I T<sub>3</sub> = p (comune a tutte le istruzioni di ingresso (uscita)

I R(i) = B<sub>i</sub> [bit in IR(6-11) che specifica l'istruzione]

	p: SC ← 0	Azzera SC
INP	p B <sub>11</sub> : AC(0-7) ← INPR, FGI ← 0	Entrata carattere
OUT	p B <sub>10</sub> : OUTR ← AC(0-7), FGO ← 0	Uscita carattere
SKI	p B <sub>9</sub> : If (FGI = 1) then (PC ← PC + 1)	Salta sulla flag d'ingresso
SKO	p B <sub>8</sub> : If (FGO = 1) then (PC ← PC + 1)	Salta sulla flag d'uscita
ION	p B <sub>7</sub> : IEN ← 1	Abilita l'interruzione
IOF	p B <sub>6</sub> : IEN ← 0	Disabilita l'interruzione

---

## Il trasferimento *programmato*

Il meccanismo di trasferimento è molto semplice.

**Le istruzioni, nella fase di esecuzione, vanno a controllare lo stato logico delle due "flag".**

**Se una di esse è ad 1, il dato in ingresso od in uscita è pronto ed allora, si salta la prossima istruzione e si va ad eseguire la successiva** che, per esempio, potrebbe essere una operazione di registrazione in memoria del byte. L'operazione saltata è, di solito, una operazione di "*trasferimento*" che rimanda il processore a controllare di nuovo lo stato della "flag".

Il difetto di questi *trasferimenti programmati* è che, finché il dato atteso non arriva, il processore è bloccato ad eseguire "*un ciclo di attesa*"(loop).

Esiste una diversa modalità di *ingresso/uscita* dei dati dal sistema che non costringe il processore ad una continua improduttiva attesa.

Questa procedura utilizza le istruzioni ION ed IOF.

## L'interruzione del programma

La tecnica di *“interruzione del programma”* è basata su un concetto diametralmente opposto rispetto alla precedente.

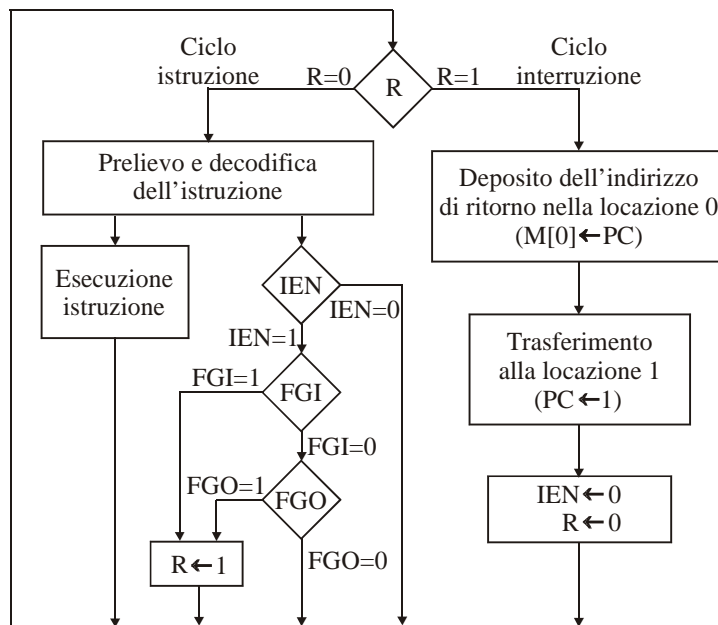
In questo caso il sistema è normalmente impegnato ad eseguire un programma nel quale non è inserita nessuna istruzione di controllo sulle *“flag”*.

Il sistema è dotato, però, di un circuito che avverte automaticamente il processore quando una *flag* è *“attivata”* il che significa che un dato è pronto per essere acquisito o trasmesso.

In questo caso il processore interrompe l'esecuzione del programma per effettuare l'operazione di I/O, per poi riprendere l'elaborazione esattamente dal punto di interruzione.

La possibilità di interruzione dell'elaborazione è associata allo stato logico del flip-flop **IEN (Interrupt ENable)** che può essere *“attivato”* od *“azzerato”* dalle due istruzioni **ION** ed **IOF**.

Queste due istruzioni, in pratica **consentono al programmatore di inserire o disinserire la possibilità di interruzioni di programma.**



**Ciclo di  
“program  
interrupt”**

## Il flip-flop R

Nella CPU esiste un flip-flop **R**.

Si chiama R da “*Request*” (richiesta), per cui quando **R = 0**, significa che **non ci sono richieste di interruzione** ed il controllo esegue il prossimo “*ciclo istruzione*”.

Durante la fase di “*esecuzione*” del ciclo istruzione, che segue quella di “*fetch*” e quella di “*decode*”, il controllo va a leggere lo stato del flip-flop **IEN**.

Se **IEN = 0** questo significa che il programmatore non vuole interruzioni di programma e si procederà al prossimo ciclo istruzione.

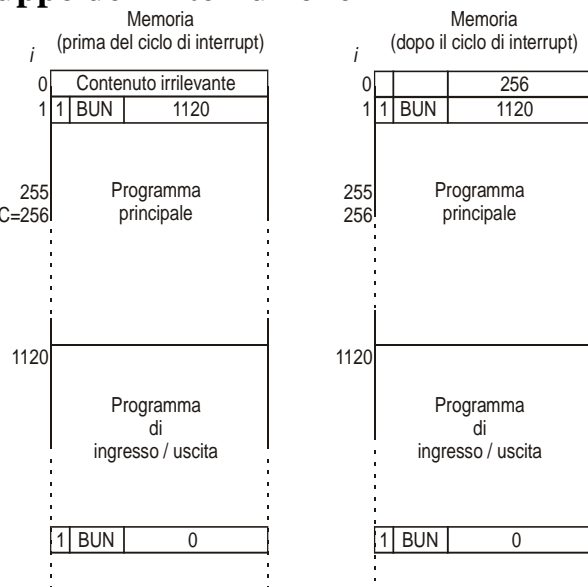
Ma se **IEN = 1** allora si vanno a leggere in successione le *flag* **FGI** ed **FGO**.

Se **entrambe sono a 0**, si procede al prossimo ciclo; ma se almeno una è ad 1, **si porta ad 1 il flip-flop R**, in modo da dare il via ad un *ciclo di interruzione* che si sviluppa secondo il ramo di destra del “*flowchart*”.

## Lo sviluppo dell'interruzione

L'azione mostrata comincia con l'attivazione ad 1 di R, mentre il processore sta eseguendo l'istruzione che si trova alla locazione 255 ed il PC punta alla successiva istruzione all'indirizzo 256 della memoria.

Il programmatore ha preventivamente registrato in memoria, a partire dalla locazione 1120, un programma per gestire il collegamento con i periferici, ed ha registrato alla locazione di indirizzo 1 una istruzione di “*Branch Unconditionally*” (**BUN**) alla locazione 1120.



## Chiusura del ciclo

**Quando il controllo, durante l'intervallo di tempo  $T_0$ , trova,  $R = 1$ , avvia il ciclo di interruzione.**

- Il contenuto del PC (256) viene memorizzato alla locazione 0.
- Nel PC viene scritto "1" ed R viene azzerato.

L'esecuzione dell'istruzione di BUN porta in esecuzione il programma di I/O che andrà per prima cosa a leggere le "**flag**" d'ingresso (**FGI**) e d'uscita (**FGO**) ed a trasferire i dati da **INPR** ad **AC** e, poi, in memoria o, viceversa, dalla memoria all'**AC** e da questo all'**OUTR**.

Il programma di I/O terminerà con la scrittura di "1" nell'**IEN**, per riabilitare l'**interrupt**, ed una istruzione di **BUN indiretto** ( $IR(15) = 1$ ), alla locazione 0, che riporterà nel PC l'indirizzo 256.

## Alternativa tra ciclo *istruzione* e ciclo *interruzione*

Il ciclo di interruzione comincia se **IEN = 1** ed il flip-flop **R** è stato attivato durante la fase di esecuzione di una istruzione, perché una, od entrambe, le **flag** sono ad 1. Questo significa che si vuole che R non possa essere attivato durante le fasi di "**fetch**" ( $T_0, T_1$ ) e di "**decode**" ( $T_2$ ) dell'istruzione.

**La condizione logica di attivazione di R si può scrivere, in linguaggio RTL:**

$$T_0 * T_1 * T_2 * (IEN) (FGI + FGO): R \leftarrow 1$$

In realtà si vuole anche il viceversa e cioè, che se **R = 1**, non possa iniziare un normale ciclo istruzione, che si sviluppa negli intervalli di tempo  $T_0$ ,  $T_1$ , e  $T_2$ .

Per ottenere di impedire l'avvio di un ciclo istruzione, quando è stata richiesta una interruzione, si possono condizionare i segnali di temporizzazione col valore booleano negato di R.

**Le temporizzazioni, delle fasi di "fetch" e "decode", quindi, saranno effettuate con le condizioni:**

$$R * T_0, R * T_1, \text{ e } R * T_2.$$

## Sequenza di *interrupt*

Lo sviluppo del *ciclo di interrupt* avverrà attraverso la seguente sequenza di microoperazioni:

**R T<sub>0</sub>:** AR ← 0, TR ← PC  
**R T<sub>1</sub>:** M[AR] ← TR, PC ← 0  
**R T<sub>2</sub>:** PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

Durante l'intervallo temporale T<sub>0</sub>, l'AR viene posizionato a 0 ed il contenuto del PC è trasferito nel *registro temporaneo*.

Al tempo T<sub>1</sub> il precedente contenuto del PC, che costituisce l'indirizzo di ritorno dopo l'interruzione, viene memorizzato alla locazione 0 ed il PC azzerato, in modo che al tempo T<sub>2</sub>, dopo l'incremento, punti all'indirizzo 1.

L'azzeramento di IEN impedisce ulteriori interruzioni, quello di R, che può essere fatto tranquillamente perché il ciclo di interruzione è ormai in corso, è indispensabile **perché possa essere eseguito il ciclo dell'istruzione di "branch"** alla prima istruzione della *routine* di I/O che è all'indirizzo 1. L'azzeramento del "contatore di sequenze" prepara l'avvio del nuovo ciclo temporale

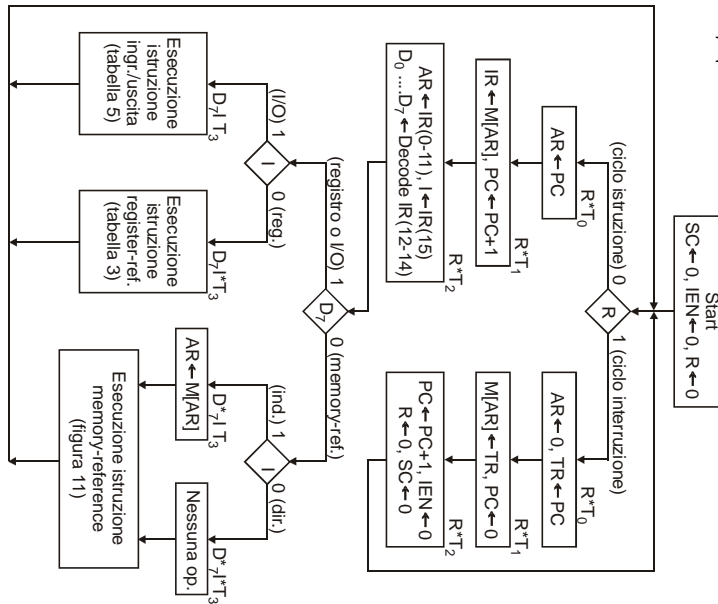
## Diagramma di flusso globale del sistema e tabella delle funzioni di controllo

Tutto quanto detto finora sulla struttura del sistema e sui suoi cicli di funzionamento può essere riassunto in un solo **diagramma di flusso complessivo** che non fa che raccogliere in un unico diagramma i pezzi di operazioni viste man mano che la descrizione procedeva.

Analogamente in una sola tabella possono riassunte tutte **le funzioni di controllo e le microoperazioni per i quattro stati indifferenziati della logica di controllo e per la fase di "esecuzione" di tutte le istruzioni del repertorio.**

**Le informazioni fornite globalmente dal diagramma di flusso e dalla tabella, non solo riassumono tutte le descrizioni della macchina fatte finora, ma sono tutto quello che serve per una completa progettazione hardware della "Macchina di Mano".**

## Diagramma di flusso globale



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

67

## Funzioni di controllo e microoperazioni (1)

Fetch	R* T <sub>0</sub> : AR ← PC
	R* T <sub>1</sub> : IR ← M[AR], PC ← PC + 1
Decode	R* T <sub>2</sub> : D <sub>0</sub> , ..., D <sub>7</sub> ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)
Indirect	D <sub>7</sub> * I T <sub>3</sub> : AR ← M[AR]
Interrupt	T <sub>0</sub> *T <sub>1</sub> *T <sub>2</sub> * (IEN)(FGI+FGO): R ← 1
	R T <sub>0</sub> : AR ← 0, TR ← PC
	R T <sub>1</sub> : M[AR] ← TR, PC ← 0
	R T <sub>2</sub> : PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

68

## Funzioni di controllo e microoperazioni (2)

---

### Memory-reference:

AND	D <sub>0</sub> T <sub>4</sub> : DR ← M[AR] D <sub>0</sub> T <sub>5</sub> : AC ← AC ∧ DR, SC ← 0
ADD	D <sub>1</sub> T <sub>4</sub> : DR ← M[AR] D <sub>1</sub> T <sub>5</sub> : AC ← AC + DR, E ← C <sub>out</sub> , SC ← 0
LDA	D <sub>2</sub> T <sub>4</sub> : DR ← M[AR] D <sub>2</sub> T <sub>5</sub> : AC ← DR, SC ← 0
STA	D <sub>3</sub> T <sub>4</sub> : M[AR] ← AC, SC ← 0
BUN	D <sub>4</sub> T <sub>4</sub> : PC ← AR, SC ← 0
BSA	D <sub>5</sub> T <sub>4</sub> : M[AR] ← PC, AR ← AR + 1 D <sub>5</sub> T <sub>5</sub> : PC ← AR, SC ← 0
ISZ	D <sub>6</sub> T <sub>4</sub> : DR ← M[AR] D <sub>6</sub> T <sub>5</sub> : DR ← DR + 1 D <sub>6</sub> T <sub>6</sub> : M[AR] ← DR, if(DR=0) then(PC ← PC + 1), SC ← 0

## Funzioni di controllo e microoperazioni (3)

---

### Register-reference:

	D <sub>7</sub> I* T <sub>3</sub> = r (comune a tutte le istruzioni di questo tipo) IR(i) = B <sub>i</sub> (i = 0, 1, 2, ..., 11) r: SC ← 0
CLA	r B <sub>11</sub> : AC ← 0
CLE	r B <sub>10</sub> : E ← 0
CMA	r B <sub>9</sub> : AC ← AC*
CME	r B <sub>8</sub> : E ← E*
CIR	r B <sub>7</sub> : AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL	r B <sub>6</sub> : AC ← shl AC, AC(0) ← E, E ← AC(15)
INC	r B <sub>5</sub> : AC ← AC + 1
SPA	r B <sub>4</sub> : If (AC(15) = 0) then (PC ← PC + 1)
SNA	r B <sub>3</sub> : If (AC(15) = 1) then (PC ← PC + 1)
SZA	r B <sub>2</sub> : If (AC = 0) then (PC ← PC + 1)
SZE	r B <sub>1</sub> : If (E = 0) then (PC ← PC + 1)
HLT	r B <sub>0</sub> : S ← 0

## Funzioni di controllo e microoperazioni (4)

---

### Input-output:

	$D_7 I^* T_3 = p$ (comune a tutte le istruzioni di input-output)
	$IR(i) = B_i$ ( $i = 6, 7, 8, 9, 10, 11$ )
	$p$ : $SC \leftarrow 0$
INP	$p B_{11}$ : $AC(0-7) \leftarrow INPR, FGI \leftarrow 0$
OUT	$p B_{10}$ : $OUTR \leftarrow AC(0-7), FGO \leftarrow 0$
SKI	$p B_9$ : If ( $FGI = 1$ ) then ( $PC \leftarrow PC + 1$ )
SKO	$p B_8$ : If ( $FGO = 0$ ) then ( $PC \leftarrow PC + 1$ )
ION	$p B_7$ : $IEN \leftarrow 1$
IOF	$p B_6$ : $IEN \leftarrow 0$

---

## Le strutture hardware della macchina.

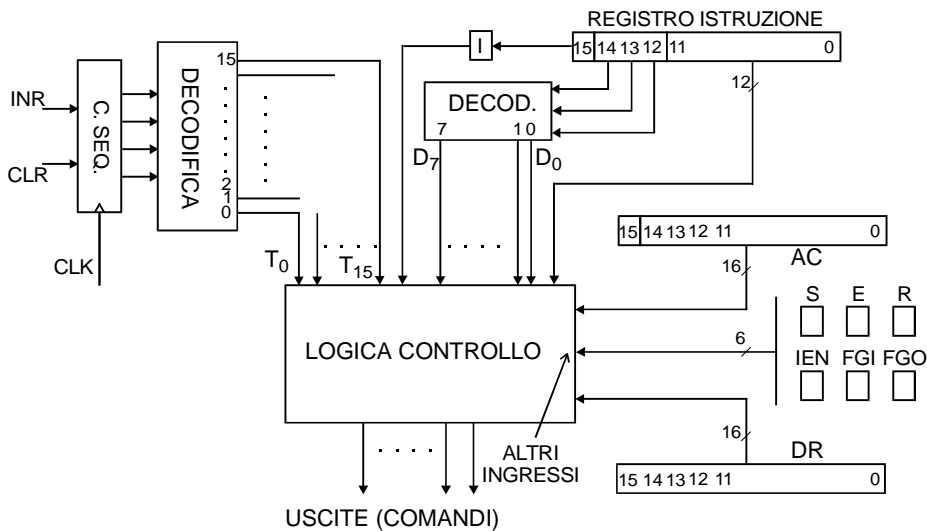
I componenti hardware del sistema descritto sono:

1. Una unità di memoria da 4096 parole da 16 bit.
2. Nove registri: AR, PC, DR, AC, IR, TR, OUTR, INPR, ed SC.
3. Sette flip-flop: I, S, E, R, IEN, FGI ed FGO.
4. Due decodifiche, una 3x8 per la decodifica del codice operativo ed una 4x16 per il timing.
5. Un bus a 16 linee dati.
6. La logica combinatoria di controllo
7. Un sommatore ed i circuiti logici per gestire gli ingressi nell'AC.

Su alcuni di questi elementi c'è poco altro da dire.

La memoria, ad esempio, ha una struttura standard per una RAM statica. Volendo realizzare il sistema non si avrebbe nessuna difficoltà ad usare un componente commerciale. Probabilmente l'unica difficoltà sarebbe il numero di locazioni molto basso che, quasi certamente costringerebbe ad una utilizzazione parziale di chip di memoria con un numero di locazioni molto maggiore.

## La logica di controllo



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

73

## Logica di controllo: ingressi ed uscite

Gli *“altri ingressi”* alla logica di controllo sono:

- il contenuto dei 16 bit dell'accumulatore per stabilire se  $AC = 0$  ed il bit segno dello stesso,  $AC(15)$ ,
- i 16 bit del registro DR per stabilire se  $DR = 0$ ,
- lo stato degli altri 6 flip-flop del sistema, visto che il settimo, I, era già in ingresso alla logica di controllo in figura 6.

Le uscite che la rete combinatoria di controllo deve produrre sono:

1. I segnali di controllo dei nove registri.
2. I segnali di Read e Write della memoria.
3. I segnali per attivare, azzerare o complementare i sette flip-flop.
4. I segnali S2 S1 S0 per la selezione dei registri sul bus.
5. I segnali necessari per gestire la ALU ed i circuiti annessi

Le condizioni logiche e temporali per la generazione di ciascuno di questi segnali possono essere estratti dalla lista delle assegnazioni (statements) in linguaggio RTL, nella tabella globale delle istruzioni

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

74

## I comandi sui registri

Gli ingressi di controllo di 5 dei registri sul bus sono LD, INR, e CLR. **Supponiamo di voler sapere in dettaglio come vengono gestiti i controlli del registro AR.**

Basta cercare nella tabella le **condizioni che cambiano il contenuto di AR.** Scorrendola con attenzione si ricava che tutte le assegnazioni che cambiano il contenuto di AR sono:

$$\begin{aligned} R^* T_0: & \text{ AR} \leftarrow \text{PC} \\ R^* T_2: & \text{ AR} \leftarrow \text{IR}(0-11) \\ D_7^* I T_3: & \text{ AR} \leftarrow \text{M}[\text{AR}] \\ R T_0: & \text{ AR} \leftarrow 0 \\ D_5 T_4: & \text{ AR} \leftarrow \text{AR} + 1 \end{aligned}$$

Le prime tre sono operazioni che portano in AR i contenuti di altri elementi connessi al bus. Questo significa che da una parte bisogna che la logica di controllo deve selezionare la sorgente perché metta il suo dato sulle linee del bus, dall'altra deve abilitare l'ingresso LD di AR per riceverlo.

La quarta operazione si ottiene attivando l'ingresso CLR e la quinta, attivando quello di incremento, INR.

Aprile 2002

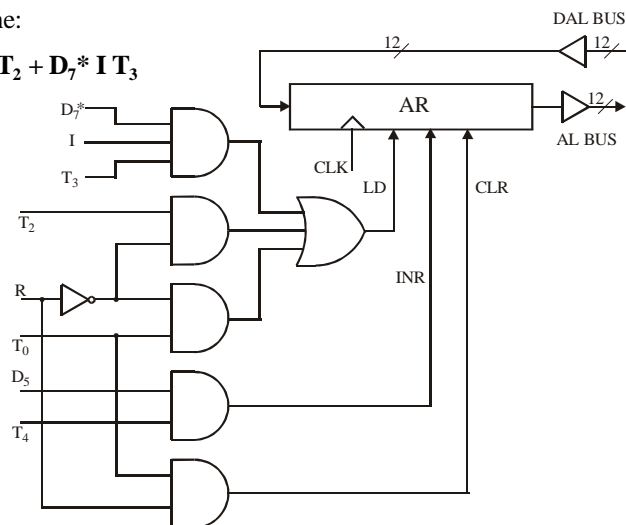
Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

75

## La logica combinatoria per il controllo di AR

Le microistruzioni si traducono nelle seguenti equazioni booleane:

$$\begin{aligned} \text{LD}(\text{AR}) &= R^* T_0 + R^* T_2 + D_7^* I T_3 \\ \text{CLR}(\text{AR}) &= R T_0 \\ \text{INR}(\text{AR}) &= D_5 T_4 \end{aligned}$$



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

76

## I segnali di controllo della memoria

In maniera del tutto analoga si può ricavare lo schema logico dei controlli associati agli ingressi di ciascuno degli altri registri. Così come quelli degli ingressi di “*scrittura*” e “*lettura*” della memoria. Per l’operazione di “*lettura*”, ad esempio, dalla tabella delle microoperazioni, cercando il simbolo  $\leftarrow M[AR]$  che la specifica, si ricava la seguente funzione booleana:

$$\text{Read} = R * T_1 + D_7 * I T_3 + (D_0 + D_1 + D_2 + D_6) T_4$$

L’uscita della corrispondente rete combinatoria deve essere collegata all’*ingresso di controllo*, R, dell’unità di memoria.

## I controlli dei flip-flop, esempio: IEN

La logica per il controllo dei sette flip-flop può essere estratta con la stessa tecnica. Per esempio, dalla solita tabella si ricava che IEN può cambiare per effetto delle due istruzioni ION e IOF.

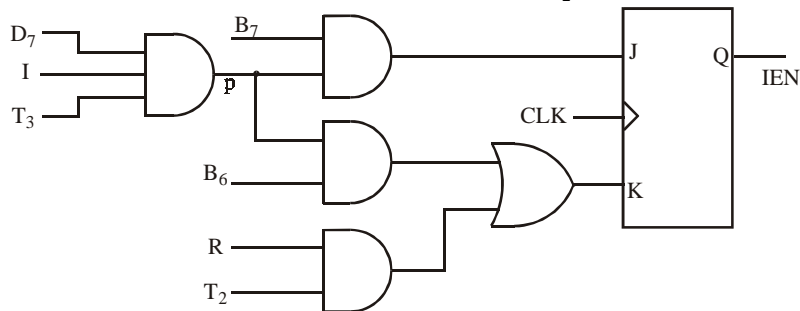
**p B<sub>7</sub>: IEN ← 1**

**p B<sub>6</sub>: IEN ← 0**

Dove **p** =  $D_7 I T_3$  e **B<sub>7</sub>** e **B<sub>6</sub>** sono i bit 7 e 6 dell’IR rispettivamente.

Alla fine del ciclo di “*interrupt*” IEN viene azzerato:

**R T<sub>2</sub>: IEN ← 0**



## I controlli del bus

L'accesso alle linee dati del bus è controllato dai tre bit di selezione  $S_2 S_1 S_0$  che considerando i gate AND di decodifica ed i driver tristate associati a ciascun registro ed alla memoria costituiscono un unico multiplexer da 8 (7) a 1 a 16 bit. Se assegniamo a ciascuna elemento del bus un bit di selezione esplicita,  $x_1, x_2, x_3, \dots, x_7$ , otteniamo la tabella di codifica dell'indirizzo

Ingressi							Uscite			Registro del bus selezionato
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$S_2$	$S_1$	$S_0$	
0	0	0	0	0	0	0	0	0	0	Nessuno
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memoria

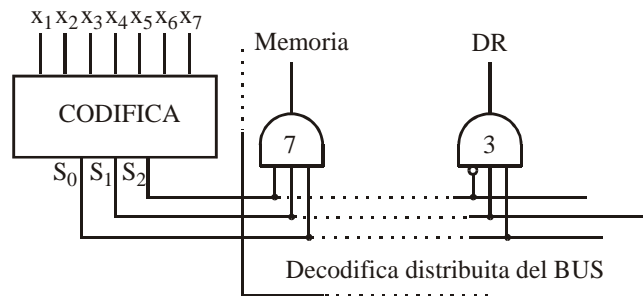
Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

79

## I controlli del bus (2)

La tabella tradotta in uno schema a blocchi porta alla configurazione di figura.



**La logica di generazione di ciascun bit di selezione si può ricavare con la solita procedura dalla tabella delle microoperazioni e delle corrispondenti condizioni logico-temporali, tenendo presente le associazioni dei bit.**

$x_1$ , per esempio, essendo associato ad AR, sarà dato dall'OR di tutte le condizioni relative ai trasferimenti in cui AR è la sorgente dell'informazione.

Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

80

### I controlli del bus (3)

Esistono solo due trasferimenti che coinvolgono AR:

$$D_4 T_4: PC \leftarrow AR$$

$$D_5 T_5: PC \leftarrow AR$$

Di conseguenza la funzione booleana di  $x_1$  è data da:

$$x_1 = D_4 T_4 + D_5 T_5$$

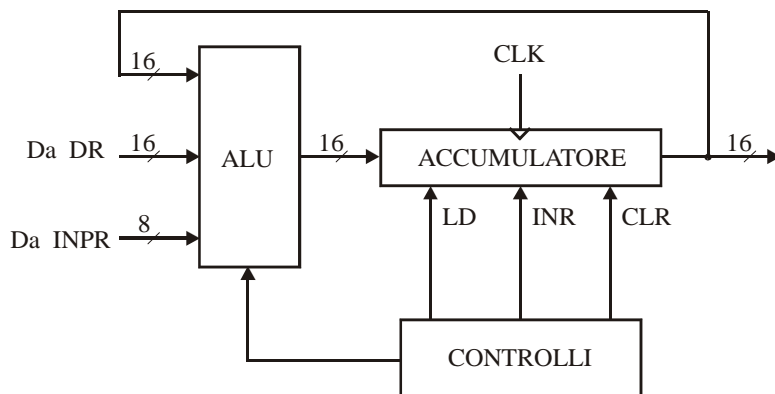
Lo stesso procedimento applicato, ad esempio, a  $x_7$ , che è il bit di selezione associato alla memoria, porterebbe alla relazione:

$$x_7 = R^* T_1 + D_7^* I T_3 + (D_0 + D_1 + D_2 + D_6) T_4$$

**che coincide con relazione che abbiamo ricavato per l'operazione di "lettura in memoria", come deve essere, perché quello è l'unico significato logico di  $x_7$ .**

**Lo stesso discorso si potrebbe fare per tutte le altre sorgenti del bus dati.**

### Controlli di AC ed ALU



## Controlli di AC ed ALU (2)

Gli elementi della progettazione debbono, come sempre, essere estratte dalla tabella delle microoperazioni associate alle istruzioni.

Le operazioni che, cambiano il contenuto dell'accumulatore e coinvolgono la ALU, risultano essere:

$D_0 T_5:$	$AC \leftarrow AC \text{ DR}$	AND con DR
$D_1 T_5:$	$AC \leftarrow AC + DR$	Addizione con DR
$D_2 T_5:$	$AC \leftarrow DR$	Trasferimento da DR
$p B_{11}:$	$AC(0-7) \leftarrow INPR$	Trasferimento da INPR
$r B_9:$	$AC \leftarrow AC^*$	Complemento
$r B_7:$	$AC \leftarrow shr AC, AC(15) \leftarrow E$	Shift a destra
$r B_6:$	$AC \leftarrow shl AC, AC(0) \leftarrow E$	Shift a sinistra
$r B_{11}:$	$AC \leftarrow 0$	Clear
$r B_5:$	$AC \leftarrow AC + 1$	Incremento

Da questa lista si possono dedurre tutte le strutture logiche di controllo di accumulatore ed ALU.

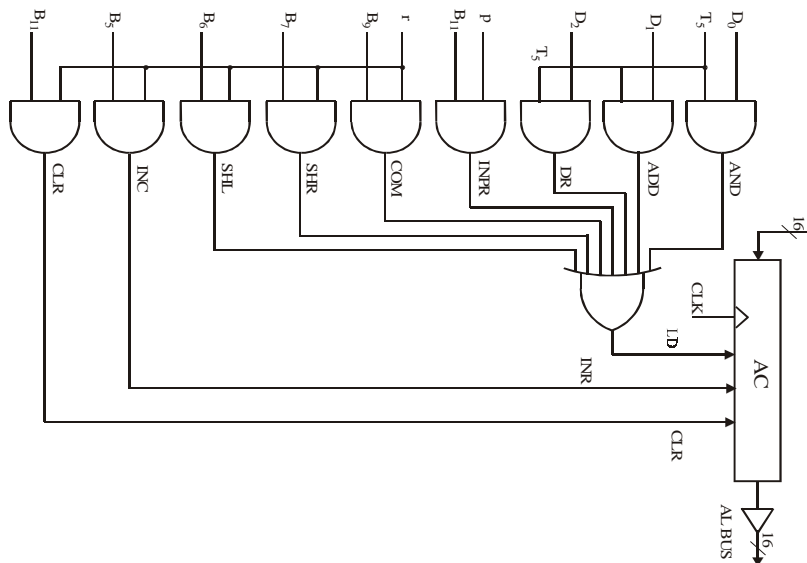
Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

83

## Controlli dell'accumulatore

La condizione logica per il "clear" di AC è:  
 $r B_{11}, \text{ dove, } r = D_7 I^* D_3 \text{ e } B_{11} = IR(11)$



Aprile 2002

Architettura degli Elaboratori - Mod. B - 2. Macchina di Mano

84

