

MEZZI DI COMUNICAZIONE CON IL CALCOLATORE

GENERALITA'

Passiamo ora ad esaminare le caratteristiche delle unità di ingresso e di uscita (input-output), attraverso le quali il calcolatore comunica con il mondo esterno. In un certo senso queste unità rappresentano la parte più importante e delicata nel lavoro di un calcolatore di processo. Sia l'ingresso al calcolatore che l'uscita da esso, sono caratterizzate dal fatto che in generale non adoperano lo stesso linguaggio del calcolatore e non funzionano alla sua stessa velocità.

I tempi di esecuzione delle varie operazioni da parte di un calcolatore sono dell'ordine dei microsecondi, mentre una scrivente elettrica non stampare circa 15 caratteri al secondo. Impiega cioè circa 65000 microsecondo per un carattere. Durante tale tempo un calcolatore non eseguire ben 20000 operazioni. Anche nel caso di processi industriali avremo in ingresso al calcolatore segnali elettrici in funzione delle grandezze fisiche controllate che possono in genere variare con tempi dell'ordine del minuto.

Ad evitare quindi che il calcolatore rimanga inoperoso per lunghi intervalli di tempo, provvedono speciali circuiti elettronici e tecniche di programmazione. Attraverso queste unità di collegamento, il calcolatore potrà comunicare con:

- gli operatori
- le unità periferiche
- il processo

Ciascuno dei tre canali di comunicazione elencati, possiede caratteristiche differenti e particolari.

Con gli operatori, il calcolatore comunica normalmente attraverso visualizzatori numerici, lampade di segnalazione, pulsanti di comando, raggruppati nella cosiddetta console operativa.

Con le unità periferiche (telescriventi, perforatori e lettori di nastro, dischi ecc) il calcolatore invece comunicherà in ingresso ed uscita attraverso segnali elettrici che dovranno di volta in volta essere adattati alle caratteristiche specifiche di ciascun tipo di periferica.

Questo adattamento si ottiene con l'impiego, come interfaccia fra periferiche e calcolatore, di registri tampone chiamati *buffer*. In tali registri le informazioni da trasferire sono memorizzate temporaneamente finchè l'unità centrale non è pronta per riceverle, oppure le unità periferiche non sono pronte ad utilizzarle.

Le informazioni fluiscono carattere per carattere tra l'unità centrale ed i registri tampone, facendo uso di segnali di interruzione per indicare che il dato è pronto per essere scritto o che la scrittura del dato è stata ultimata.

Grazie a questo tipo di organizzazione, l'unità centrale può così continuare la sua elaborazione a velocità elettronica, mentre le unità periferiche lavorano alla loro normale velocità di funzionamento.

Infine il calcolatore comunicherà con il processo, ancora attraverso segnali elettrici che principalmente sono in una forma differente dal linguaggio del calcolatore ed inoltre vengono generati in posti distanti rispetto al luogo di utilizzazione. Infatti questi segnali di ingresso (come anche eventuali segnali di uscita) possono essere analogici o digitali.

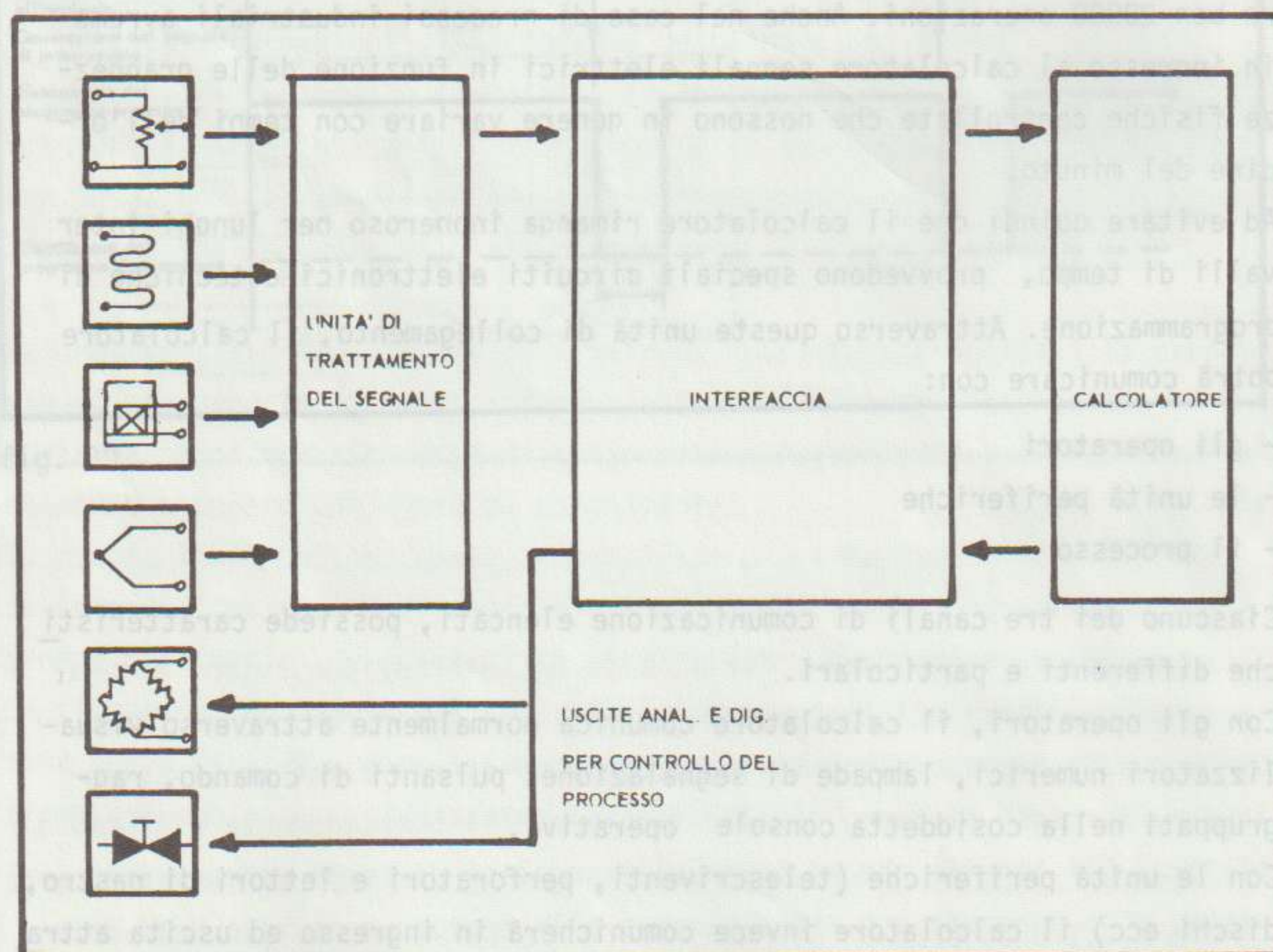


fig. 31

Quelli digitali non presentano problemi di compatibilità con il calcolatore che usa logica digitale.

Quelli analogici sono invece più complessi da trattare e richiedono non solo un convertitore analogico/digitale, ma una serie di unità dalle quali dipendono in larga misura le prestazioni di tutto il sistema.

La fig. 31 riassume quanto abbiamo esposto.

Riassumendo quindi, per poter "comunicare" con il processo, il calcolatore necessiterà di una unità intermedia, più o meno complessa a seconda del numero e del tipo di segnali, che svolga un certo numero di funzioni come per esempio:

- acquisizione di segnali e razionalizzazione dei loro livelli d'ingresso.
- esplorazione sequenziale dei vari ingressi.
- conversione analogica/digitale dei segnali acquisiti per renderli accettabili come informazioni al calcolatore.
- invio di uscite per scopi quali: comandi e controllo.

Accennando solo di sfuggita alle caratteristiche dei segnali elettrici, diremo che i vari tipi di trasduttori utilizzati nei processi industriali (per misure di pressione relativa, pressione assoluta, livello, portata, ecc), danno in generale grandezze analogiche in corrente e talvolta in valori di tensione piccoli o medi.

E' bene sottolineare infine come il calcolatore, o meglio l'interfaccia che lo collega al processo, accetti solo segnali di tensione.

UNITA' PERIFERICHE

Possiamo dire che il calcolatore, inteso come unità a sè stante, avrebbe un limitato motivo di esistere dal punto di vista funzionale, se non fosse collegato a delle apparecchiature periferiche tramite le quali egli può comunicare con l'esterno. Tali periferiche infatti lavorano, rispetto al calcolatore, con funzioni di ingresso e/o di uscita.

La fig. 32 mostra, con uno schema a blocchi, alcune delle unità periferiche che possono essere collegate ad un calcolatore: le frecce indicano il senso operativo di tali unità.

La fig. 33 successiva sintetizza invece, secondo un criterio logico, la importanza delle varie periferiche nei sistemi.

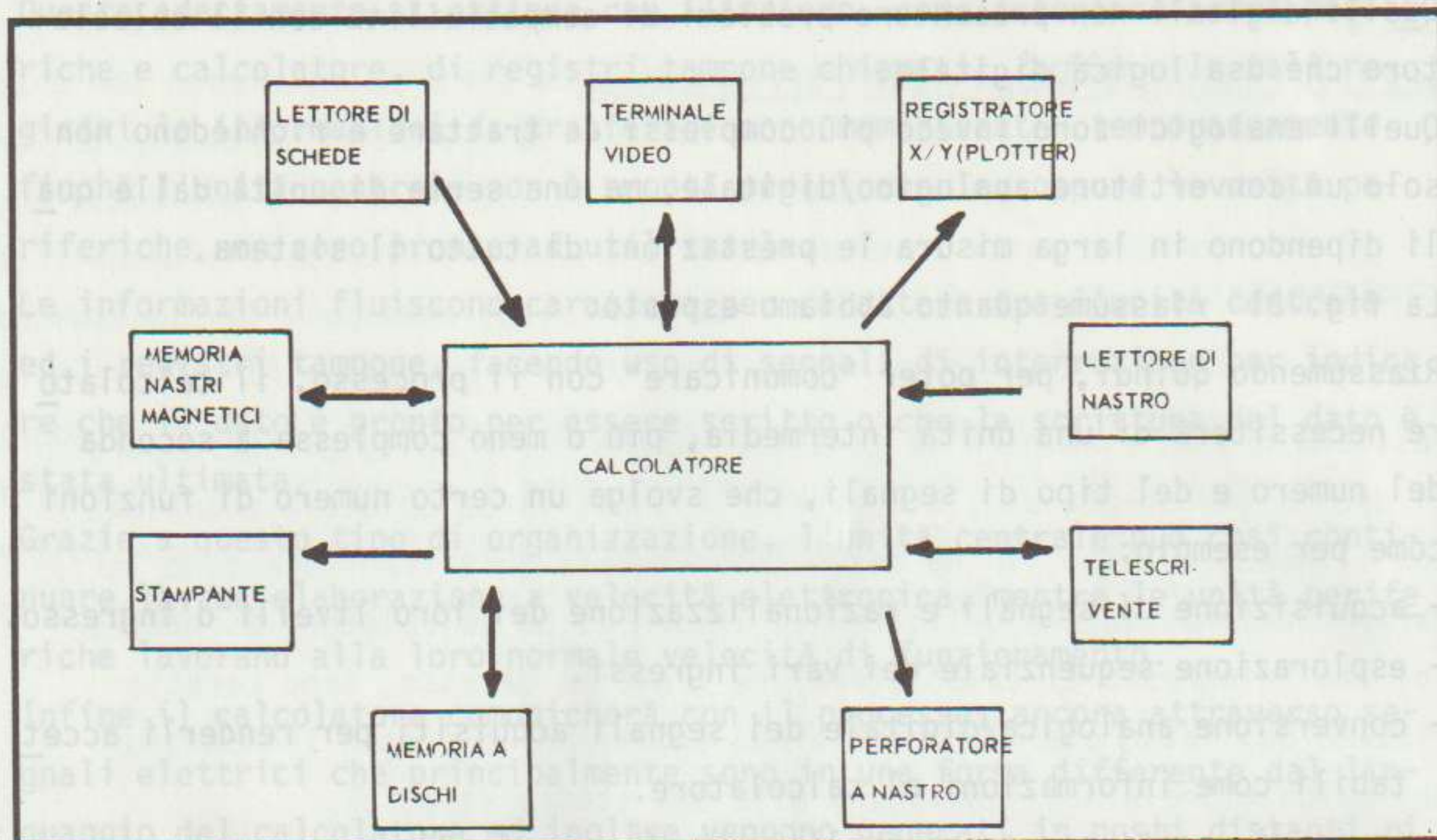


Fig. 32

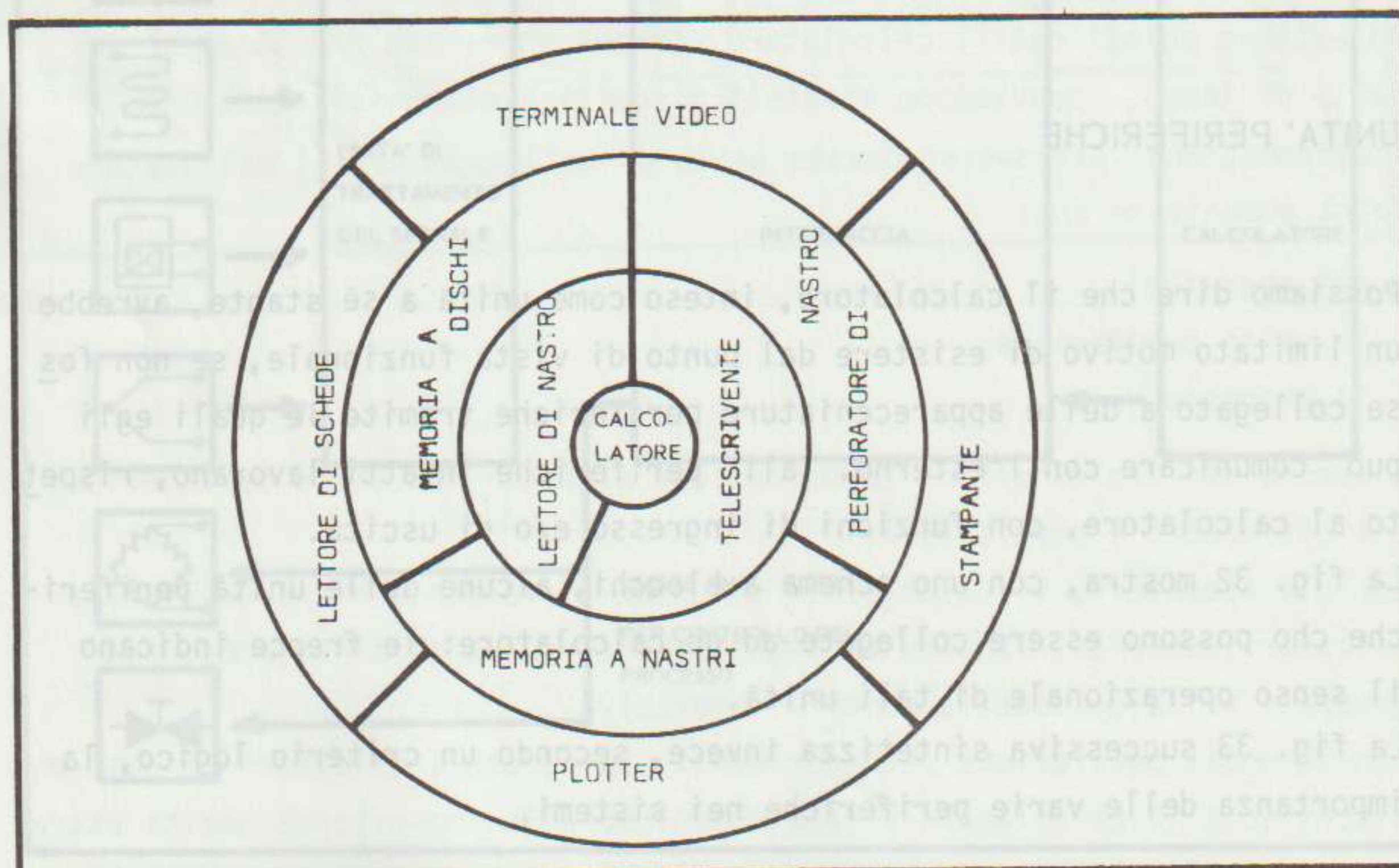


fig. 33

TELESCRIVENTE

Questa periferica, la quale è una particolare scrivente elettrica a carrello fisso, è senz'altro quella che troveremo più comunemente accoppiata ad un calcolatore di processo, per motivi sia economici che di alta versatilità di impiego. Ve ne sono oggi sul mercato una vasta gamma; alcune di queste sono atte a ricevere solo dati dal calcolatore, altre sono invece predisposte anche per l'invio di dati al calcolatore stesso.

In genere la telescrivente trasmette i dati utilizzando una linea a due fili; uno usato come linea di segnale, l'altro come filo di ritorno.

Quando un carattere viene trasferito dalla telescrivente al calcolatore, esso viene raccolto da un registro tampone (buffer) di 9 bits. Il conseguente trasferimento del contenuto di tale registro al calcolatore avviene in parallelo (input mode). Cioè tutti i bits vengono inviati contemporaneamente. Quando il carattere viene trasferito dal calcolatore alla telescrivente il procedimento è inverso, ma analogo. In questo caso infatti i caratteri componenti l'informazione vengono trasferiti sempre in parallelo attraverso appositi canali di collegamento al medesimo registro tampone e quindi mandati alla telescrivente (output mode).

Si è già accennato nei paragrafi precedenti che molto spesso il codice del linguaggio usato dalla periferica non è uguale a quello del calcolatore; la telescrivente infatti usa ad esempio il codice ASCII (American Standard Code for Information Interchange). Che cosa significa e come si utilizza, vedremo in seguito.

Vediamo ora quali siano le parti principali componenti questa apparecchiatura.

L'unità di stampa è il componente di ricezione. Un segnale inviato a tale unità è convertito in un movimento meccanico. Questo movimento meccanico determina il posizionamento della testina di scrittura sul carattere selezionato e la esecuzione di funzioni come ritorno carrello o cambio di linea. Un motore fornisce la forza elettromeccanica per effettuare la stampa e le diverse funzioni complementari.

L'unità di controllo, è l'elemento di collegamento tra il calcolatore e l'unità meccanica della scrivente. In alcuni casi questa unità presiede a ciascuna o tutte le funzioni di invio, controllo, accettazione dei dati.

La tastiera è il componente attraverso il quale è possibile materialmente intervenire dall'esterno sul sistema. Ciascuno dei tasti che la compongono fa capo ad un insieme di leve, le quali posizionano dei contatti elettrici per tradurre il carattere in impulsi elettrici.

La telescrivente potrà essere eventualmente corredata da un lettore/perforatore di nastro. Il perforatore, a differenza di altri, funziona basandosi su un principio meccanico. Infatti le stesse barrette che comandano l'unità di stampa, muovono i punzoni del perforatore.

Il lettore invece, funziona in quanto i punzoni sono sensibili alla presenza o meno di fori sul nastro, andando così in concomitanza a chiudere o meno una serie di contatti elettrici.

La fig. 34 mostra una configurazione a blocchi delle unità citate.

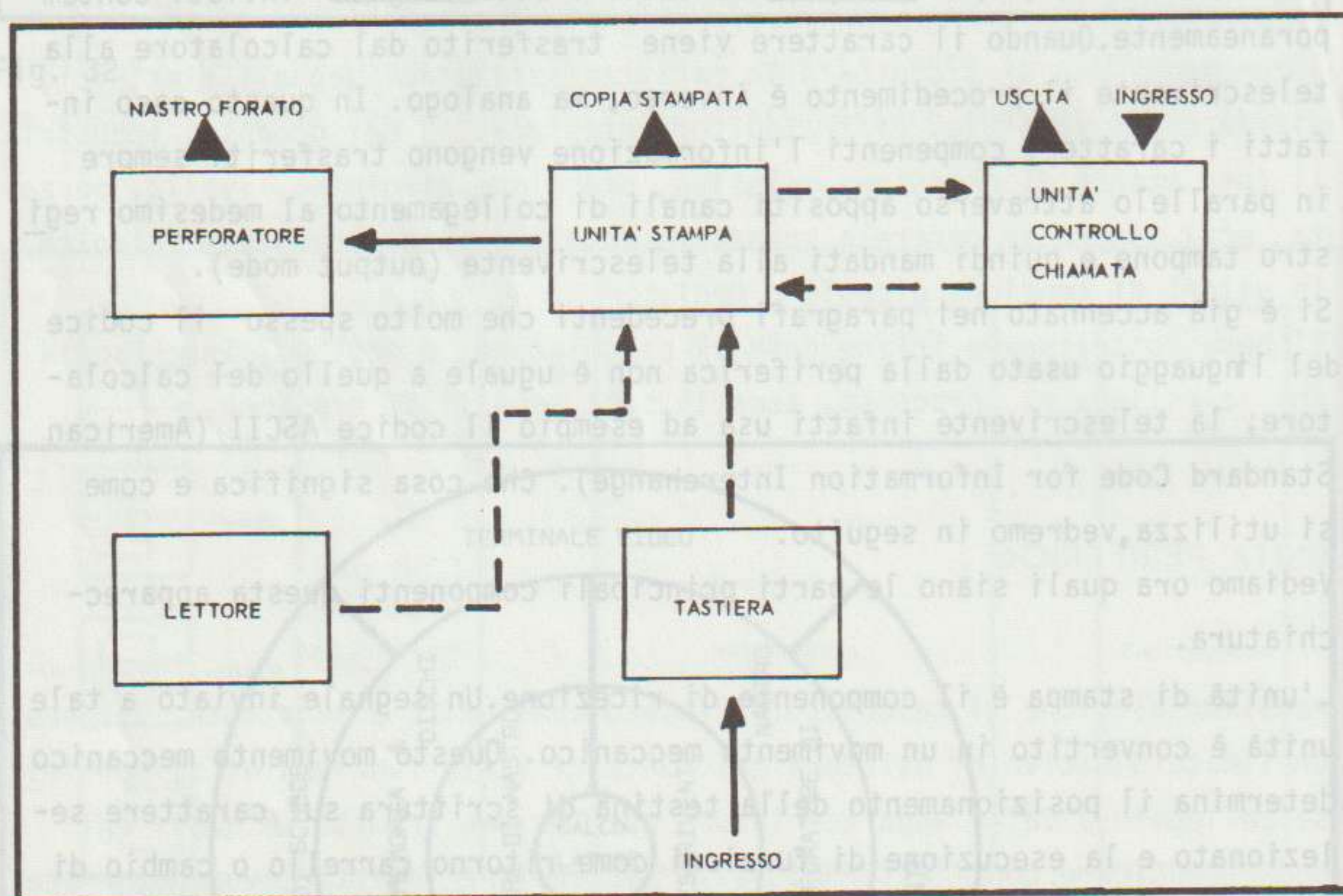


FIG. 34 - SCHEMA A BLOCCHI DI UNA TELESKRIVENTE CON RELATIVO LETTORE MECCANICO DI NASTRO

LETTORE VELOCE DI NASTRO

Questa unità è indispensabile nei sistemi con calcolatore. Infatti attraverso di esso vengono messi nella memoria del calcolatore i programmi applicativi per essere successivamente utilizzati. E' quindi una apparec-

chiatura che invia informazioni solo verso il calcolatore, ricevendo da questo segnali di controllo quali l'avvio o l'arresto della lettura. Generalmente la lettura del nastro avviene tramite diodi fotoelettrici, i quali se eccitati emettono impulsi elettrici che opportunamente amplificati vengono poi inviati al registro tampone e successivamente al calcolatore. L'eccitazione di questi diodi dipenderà dalla presenza o meno di un buco sul nastro in corrispondenza di ciascuno di questi diodi. Il nastro potrà avere per esempio una configurazione del tipo in figura. Ciascuna serie orizzontale di fori presenterà una canale, mentre la serie verticale è, in codice, un messaggio o una parte di esso.

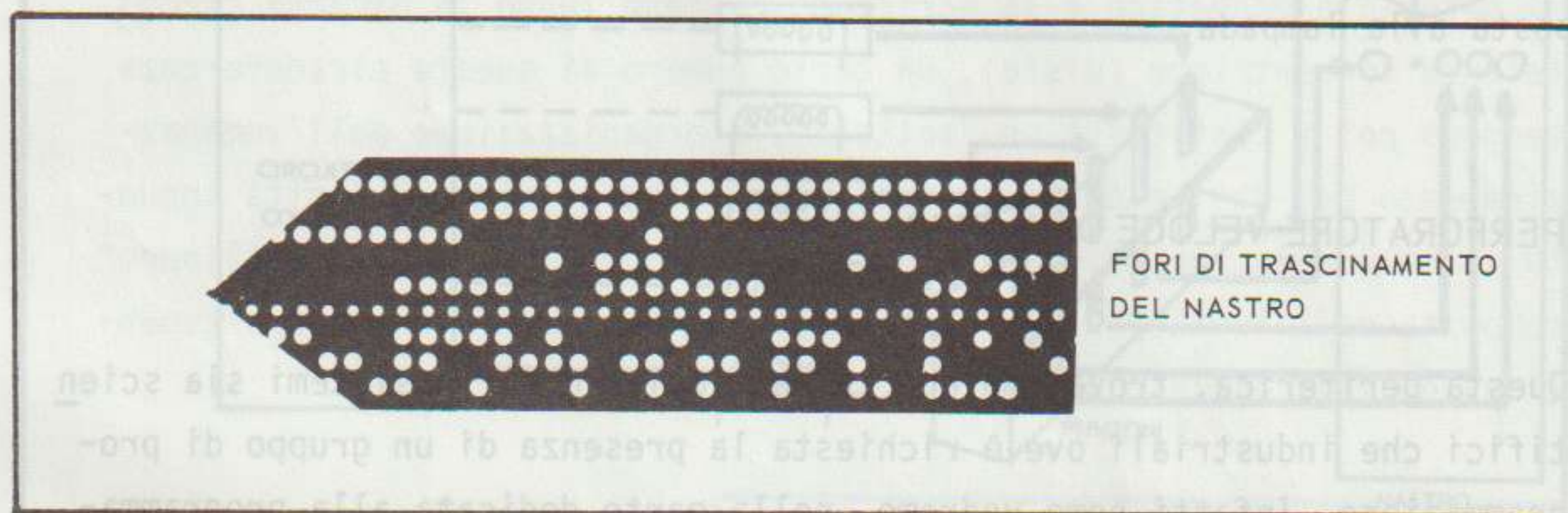


fig. 35

Se ora ci riferiamo alla fig. 36, vediamo come il nastro di carta perforato, dopo essere stato introdotto nel lettore, venga fatto passare tra due rulli.

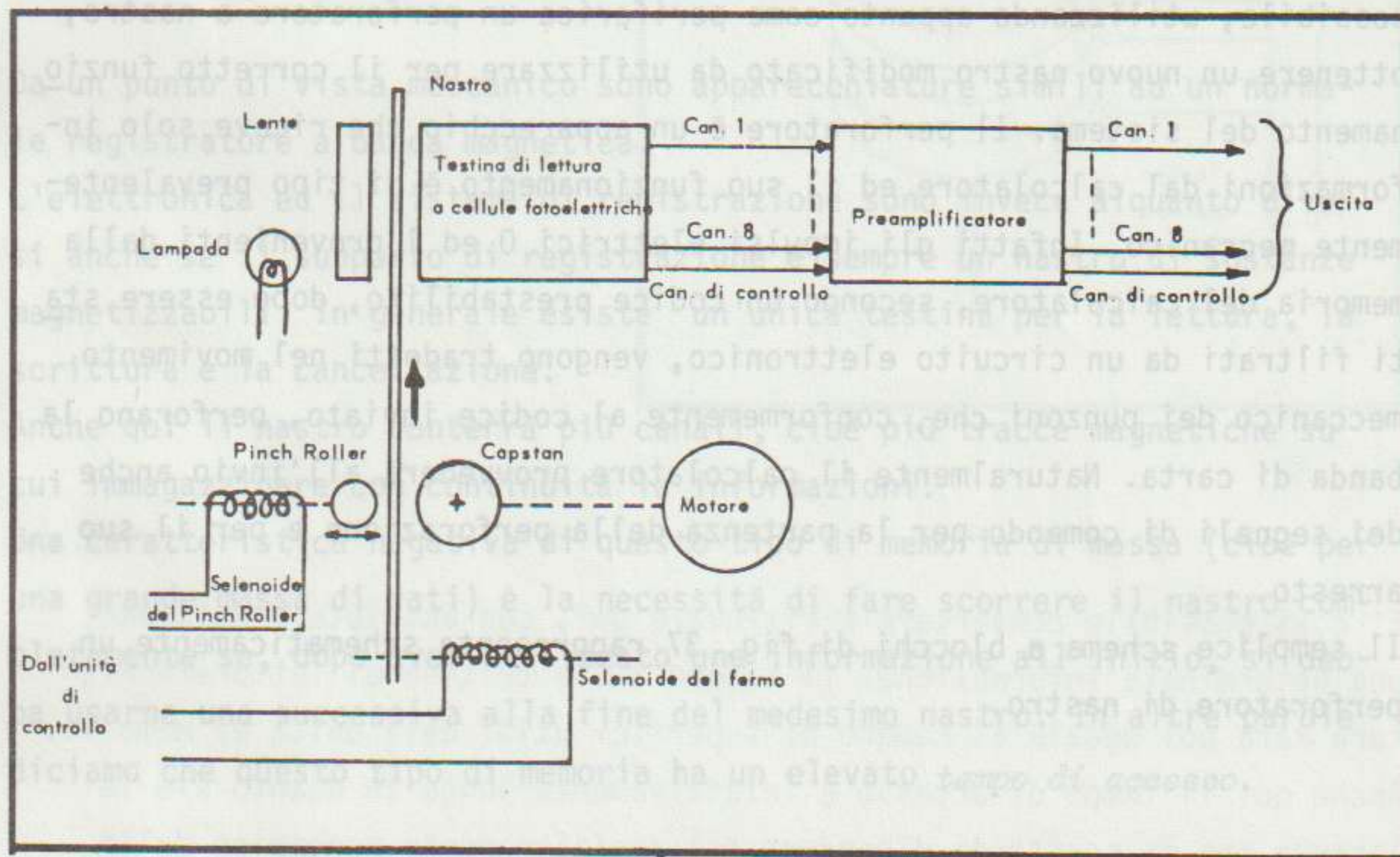


fig. 36

Il primo di essi (capstan) gira intorno ad un asse fisso mosso da un motore elettrico; il secondo (pinch roller), avendo l'asse mobile, assumerà due posizioni a seconda che il solenoide che ne comanda il movimento sia eccitato o meno. Nel caso il solenoide sia eccitato, questo rullo premerà contro il primo il nastro di carta, che verrà così trascinato. Per bloccare il movimento secondo l'esigenza del programma dovrà essere eccitato un altro solenoide che bloccherà tra le sue espansioni il nastro. Come abbiamo già accennato una lampada, il cui fascio di luce viene concentrato da una apposita lente, ecciterà in corrispondenza dei fori nel nastro, i relativi diodi che si trovano in posizione diametralmente opposta alla lampada.

PERFORATORE VELOCE DI NASTRO

Questa periferica, trova impiego in particolar modo in sistemi sia scientifici che industriali ove è richiesta la presenza di un gruppo di programmazione. Infatti come vedremo nella parte dedicata alla programmazione, un programma nella sua stesura finale viene perforato su nastro o su schede.

Può capitare che per un errore di perforazione o per l'esigenza di introdurre modifiche nel programma originale, il nastro che lo contiene debba essere cambiato. Attraverso complesse operazioni di programmazione sarà possibile, utilizzando appunto come periferica un perforatore a nastro, ottenere un nuovo nastro modificato da utilizzare per il corretto funzionamento del sistema. Il perforatore è un apparecchio che riceve solo informazioni dal calcolatore ed il suo funzionamento è di tipo prevalentemente meccanico. Infatti gli impulsi elettrici 0 ed 1 provenienti dalla memoria del calcolatore, secondo un codice prestabilito, dopo essere stati filtrati da un circuito elettronico, vengono tradotti nel movimento meccanico dei punzoni che, conformemente al codice inviato, perforano la banda di carta. Naturalmente il calcolatore provvederà all'invio anche dei segnali di comando per la partenza della perforazione e per il suo arresto.

Il semplice schema a blocchi di fig. 37 rappresenta schematicamente un perforatore di nastro.

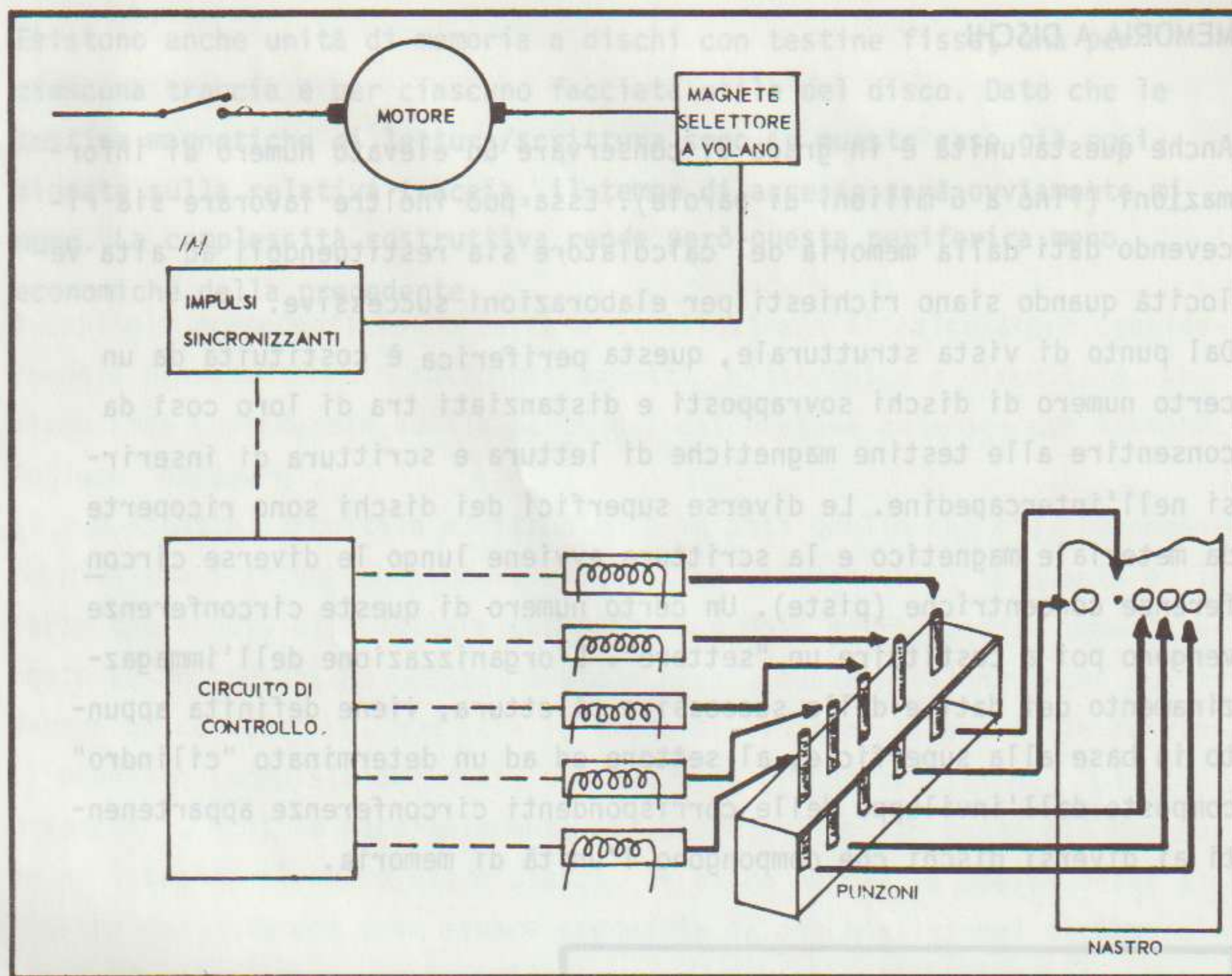


FIG. 37 SCHEMA A BLOCCHI DI UN PERFORATORE DI NASTRO

MEMORIA A NASTRO MAGNETICO

Da un punto di vista meccanico sono apparecchiature simili ad un normale registratore a banda magnetica.

L'elettronica ed il sistema di registrazione sono invece alquanto diversi anche se il supporto di registrazione è sempre un nastro di sostanze magnetizzabili. In generale esiste un'unica testina per la lettura, la scrittura e la cancellazione.

Anche qui il nastro conterrà più canali, cioè più tracce magnetiche su cui immagazzinare con continuità le informazioni.

Una caratteristica negativa di questo tipo di memoria di massa (cioè per una grande massa di dati) è la necessità di fare scorrere il nastro completamente se, dopo aver utilizzato una informazione all'inizio, si debba usarne una successiva alla fine del medesimo nastro. In altre parole diciamo che questo tipo di memoria ha un elevato *tempo di accesso*.

MEMORIA A DISCHI

Anche questa unità è in grado di conservare un elevato numero di informazioni (fino a 6 milioni di parole). Essa può inoltre lavorare sia ricevendo dati dalla memoria del calcolatore sia restituendoli ad alta velocità quando siano richiesti per elaborazioni successive.

Dal punto di vista strutturale, questa periferica è costituita da un certo numero di dischi sovrapposti e distanziati tra di loro così da consentire alle testine magnetiche di lettura e scrittura di inserirsi nell'intercapedine. Le diverse superfici dei dischi sono ricoperte da materiale magnetico e la scrittura avviene lungo le diverse circonferenze concentriche (piste). Un certo numero di queste circonferenze vengono poi a costituire un "settore". L'organizzazione dell'immagazzinamento dei dati e delle successive riletture, viene definita appunto in base alla superficie, al settore ed ad un determinato "cilindro" composto dall'involuppo delle corrispondenti circonferenze appartenenti ai diversi dischi che compongono l'unità di memoria.

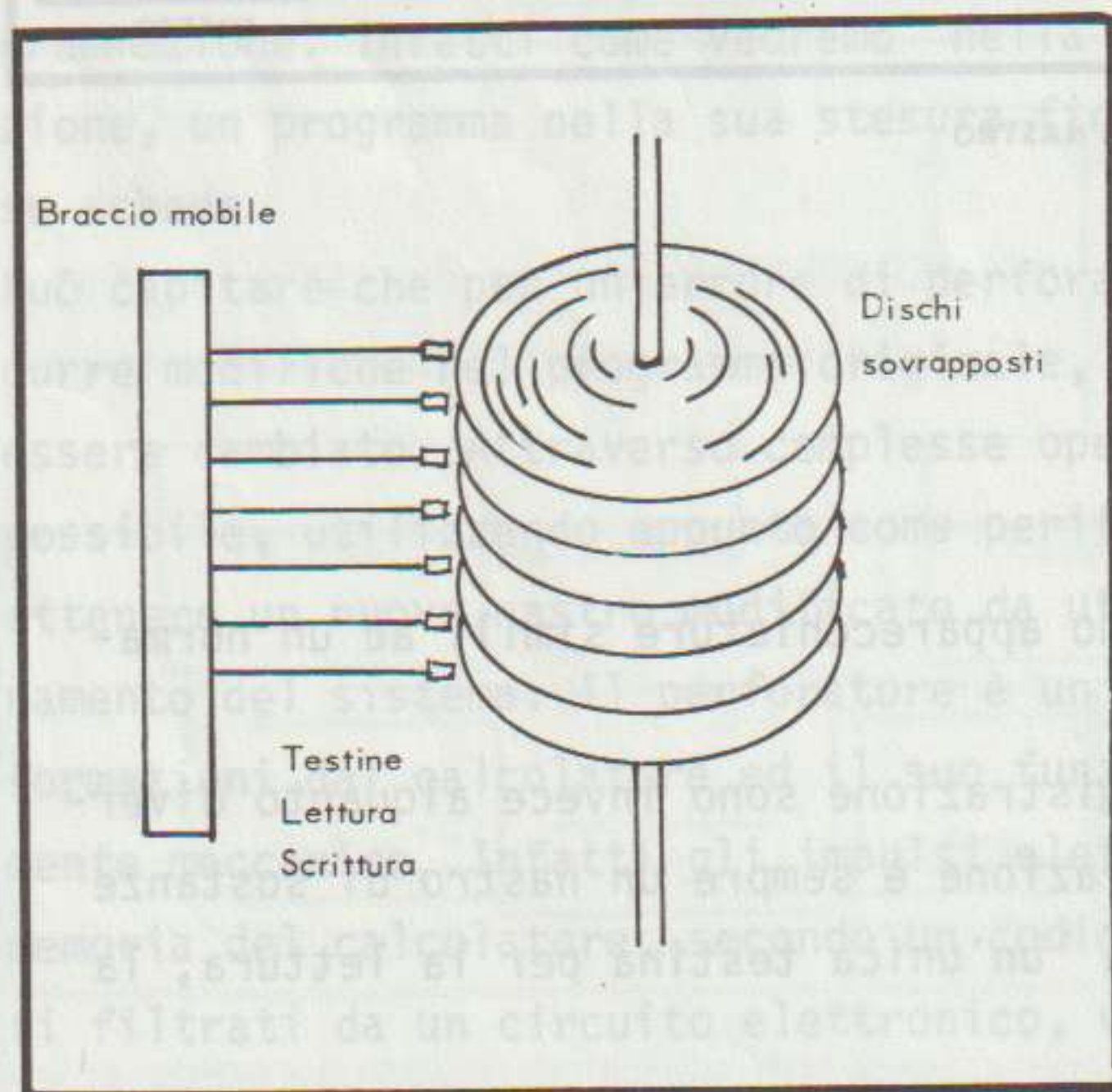


fig. 38

E' così possibile conservare e ritrovare poi, con assoluta precisione, una determinata informazione. Il numero delle testine di lettura/scrittura sarà poi uguale al numero di superfici utili dell'unità di memoria. Anche qui il tempo di accesso è intuitivamente lungo in quanto sia la lettura che la scrittura dipendono dal posizionamento meccanico delle testine relative.

Esistono anche unità di memoria a dischi con testine fisse, una per ciascuna traccia e per ciascuno fasciata utile del disco. Dato che le testine magnetiche di lettura/scrittura sono in questo caso già posizionate sulla relativa traccia, il tempo di accesso sarà ovviamente minore. La complessità costruttiva rende però questa periferica meno economiche della precedente.

I capitoli precedenti hanno cercato di illustrare il calcolatore considerando prevalentemente sotto gli aspetti elettronico e circuitale. Abbiamo cioè considerato quella parte del calcolatore chiamata con termine inglese "hardware".

Il suo aspetto circuitale elettronico è infatti quello che più comunemente richiama l'attenzione superficiale di un utilizzatore, in quanto è la parte che è possibile toccare con mano facendola corrispondere immediatamente ad un prezzo. Ma questa "scatola nera", quale possiamo anche considerare un calcolatore, non sarà in grado di svolgere alcuna delle funzioni previste (acquisizione dati, controllo di grandezze fisiche, calcoli matematici, ecc.) se parallelamente non verrà sviluppato anche ciò che serve ad istruire il calcolatore stesso. In altre parole la scelta della elettronica del sistema deve essere preceduta da una analisi del sistema al quale il calcolatore verrà applicato e da una concretizzazione di questa analisi in un linguaggio assimilabile dal calcolatore, cioè la fase di programmazione.

Questi due aspetti di analisi e di programmazione che dobbiamo considerare come qualcosa di strettamente legato al calcolatore per assicurarne la funzionalità operativa, sono appunto quelli meno visibili o addirittura invisibili per il profano. Ecco quindi la ragione delle incomprensioni che nascono sull'interpretazione e quindi sull'accettazione di un determinato prezzo per un sistema con calcolatore ed anche sulla possibile differenza talvolta notevole tra prezzi di sistemi apparentemente poco dissimili dal punto di vista "hardware".

L'analisi del sistema e la programmazione sono due fasi di un lavoro di gruppo che può impegnare tecnici qualificati per periodi di tempo anche rilevanti: da qui l'elevato costo spesso superiore a quello corrispondente alla parte "hardware", che andrà ad incidere sul prezzo finale del sistema.

Analizzare un sistema infatti vuol dire ridurre in modo logico una funzione operativa più o meno complessa, in una serie di funzioni elementari che potranno essere eseguite dal calcolatore sia in modo sequenziale che in parallelo (multiprogrammazione).

LA PROGRAMMAZIONE

GENERALITA'

I capitoli precedenti hanno cercato di illustrare il calcolatore considerando prevalentemente sotto gli aspetti elettronico e circuitale. Abbiamo cioè considerato quella parte del calcolatore chiamata con termine inglese "hardware".

Il suo aspetto circuitale elettronico è infatti quello che più comunemente richiama l'attenzione superficiale di un utilizzatore, in quanto è la parte che è possibile toccare con mano facendola corrispondere immediatamente ad un prezzo. Ma questa "scatola nera", quale possiamo anche considerare un calcolatore, non sarà in grado di svolgere alcuna delle funzioni previste (acquisizione dati, controllo di grandezze fisiche, calcoli matematici, ecc) se parallelamente non verrà sviluppato anche ciò che serve ad istruire il calcolatore stesso. In altre parole la scelta della elettronica del sistema deve essere preceduta da una analisi del sistema al quale il calcolatore verrà applicato e da una concretizzazione di questa analisi in un linguaggio assimilabile dal calcolatore, cioè la fase di programmazione.

Questi due aspetti di analisi e di programmazione che dobbiamo considerare come qualcosa di strettamente legato al calcolatore per assicurarne la funzionalità operativa, sono appunto quelli meno visibili o addirittura invisibili per il profano. Ecco quindi la ragione delle incomprensioni che nascono sull'interpretazione e quindi sull'accettazione di un determinato prezzo per un sistema con calcolatore ed anche sulla possibile differenza talvolta notevole tra prezzi di sistemi apparentemente poco dissimili dal punto di vista "hardware".

L'analisi del sistema e la programmazione sono due fasi di un lavoro di gruppo che può impegnare tecnici qualificati per periodi di tempo anche rilevanti: da qui l'elevato costo spesso superiore a quello corrispondente alla parte "hardware", che andrà ad incidere sul prezzo finale del sistema.

Analizzare un sistema infatti vuol dire ridurre in modo logico una funzione operativa più o meno complessa, in una serie di funzioni elementari che potranno essere eseguite dal calcolatore sia in modo sequenziale che in parallelo (multiprogrammazione).

Un utile supporto pratico durante questa fase di indagine è lo schema di flusso (flow-chart). In questo modo, attraverso simboli con significati ben definiti, è possibile costruire un modello logico di approccio al problema. La fig. 39 mostra in chiave umoristica un esempio di questi schemi di flusso.

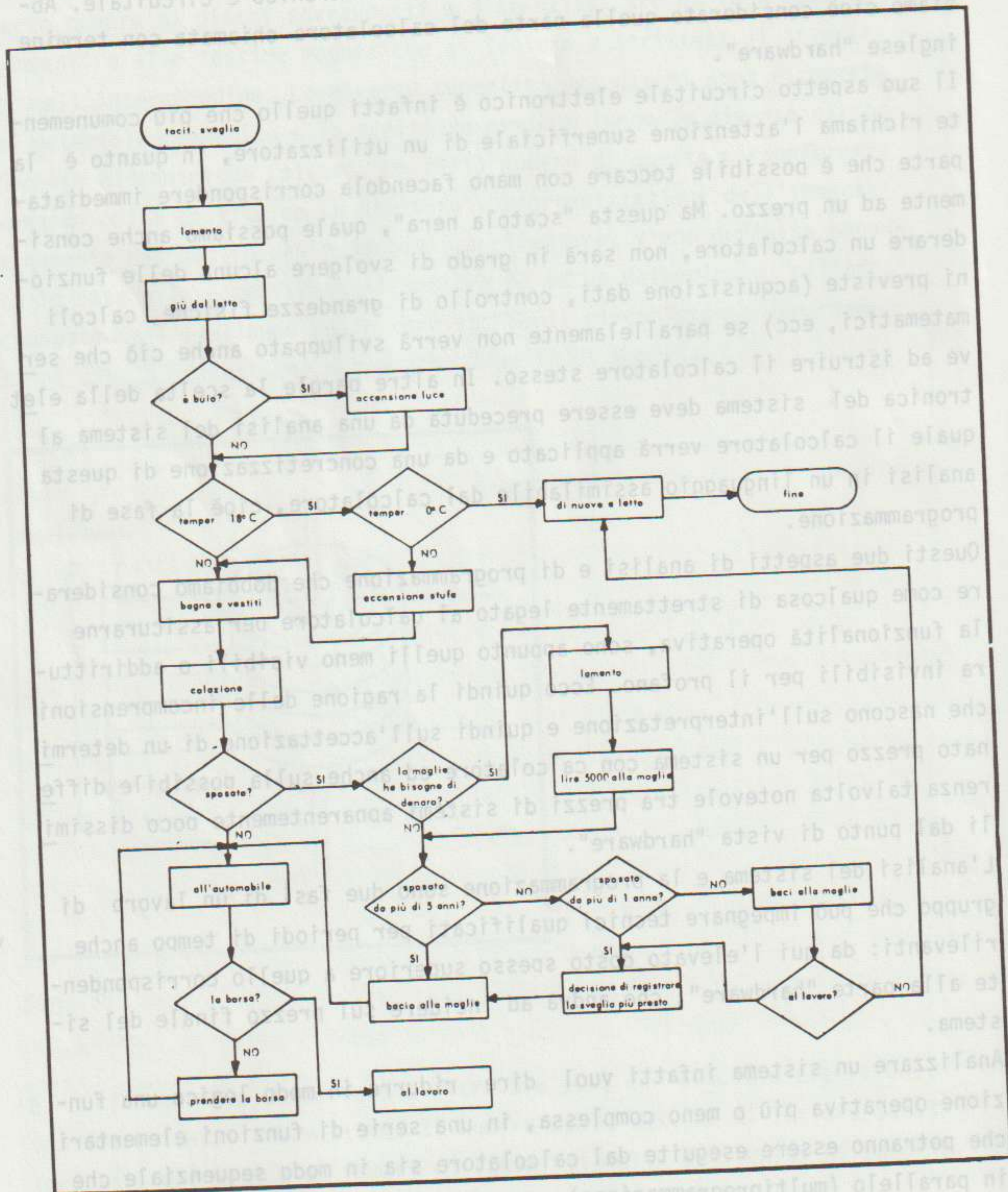


FIG. 39 - Diagramma di flusso: "sveglia al mattino"

Nella figura i rettangoli indicano le funzioni operative, i rettangoli con gli spigoli arrotondati le funzioni di inizio e fine del programma, i rombi le funzioni di interrogazione e confronto. Terminata questa prima fase, nella quale è rilevante l'apporto del tecnico di processo, sopravviene la successiva di programmazione. Il programmatore o meglio un "gruppo" di programmatori tradurrà poi le funzioni elementari ottenute dall'analisi del sistema, in componenti ancora più essenziali cioè istruzioni e dati numerici. Questi che possiamo considerare come immagine binaria del problema, messi ciascuno in una parola della memoria corrisponderanno ad operazioni base del calcolatore. Avremo così ad esempio la somma del contenuto di due parole oppure lo spostamento del contenuto di una parola in un registro e così via. Le funzioni così espresse, disposte in successione secondo un criterio logico, rappresenteranno in definitiva il programma sulla base del quale il calcolatore, cioè l'hardware, opererà.

L'organizzazione del programma base si articolerà principalmente su liste di istruzioni componenti i diversi sottoprogrammi (un indirizzo in memoria per ciascuna istruzione con una sequenza che rispetti lo scorrere logico delle operazioni da svolgere) e su aree di memoria riservate ai dati su cui operare ed ai risultati delle operazioni svolte dal programma stesso. Dobbiamo inoltre tener presente come sia possibile, non solo scrivere in memoria in una locazione vergine, ma anche sovrascrivere su una posizione di memoria già occupata, cancellando quanto precedentemente memorizzato.

In senso generale questo ci porta a concludere che il calcolatore è riprogrammabile utilizzando la stessa memoria già impiegata in un precedente programma non più di attualità. Possiamo quindi utilizzare lo stesso calcolatore, cioè lo stesso hardware, con diversi programmi (diverso software). Mentre questa possibilità è sfruttata nel caso dei calcolatori per calcoli matematici e gestionali, essa lo è solo eccezionalmente nel caso dei calcolatori applicati ai processi industriali o scientifici. Qui il calcolatore è "dedicato" ad un certo numero di funzioni, senza necessità di sostituirle integralmente nel tempo. Nel passato per queste applicazioni si sono usate talvolta le logiche cablate a programmi fissi ed oggi si usano in parte le memorie di sola lettura (Read only memory = ROM) affiancate alle normali memorie di lavoro.

Dipenderà dai criteri tecnici ed economici con i quali si vuole impostare l'automazione di un processo, la scelta tra queste diverse soluzioni pos-

sibili. Nei paragrafi seguenti cercheremo di introdurre in modo semplice i concetti fondamentali della programmazione, ma mettendo comunque in risalto i problemi e le difficoltà.

PROGRAMMA = PIU' ISTRUZIONI

Abbiamo detto che affinché sia comprensibile per un calcolatore, la struttura di una istruzione deve essere composta da un insieme di zero ed uno. Questo insieme, che chiamiamo forma binaria, non è però di facile uso da parte di un programmatore, specie nel caso di programmi di una certa complessità. Sorge quindi la necessità di disporre di un mezzo di scrittura più immediata delle istruzioni, quale è appunto il linguaggio assembler (Assembler). Con questo linguaggio simbolico, una istruzione scritta in "assembler" corrisponderà ad una combinazione binaria unica e caratteristica nella memoria del calcolatore. Anche se i linguaggi assembler variano con il tipo di calcolatore, vi sono delle istruzioni le cui funzioni sono caratteristiche e sono quindi sempre presenti nei linguaggi assembler più comuni. Una istruzione fondamentale che come risultato provvede a trasferire il contenuto di una certa locazione nel registro A è rappresentata dalla espressione mnemonica LDA '107 (load A register = carica il registro A). Il valore ottale '107 significa che ogni parola in memoria è individuata da un suo "indirizzo" specifico. Gli indirizzi o locazioni sono numerati da 0 a 4095 e così via sino alla massima capacità della memoria (generalmente 16 K o 32 K).

L'espressione simbolica LDA, detta *mnemonica* e l'indirizzo ottale relativo ('107) detto *indirizzo dell'operando* compongono una istruzione che per il suo svolgimento fa riferimento alla memoria (*MEMORY REFERENCE INSTRUCTION*). A questa istruzione corrisponderà poi, come abbiamo già detto, una unica combinazione di simboli binari in memoria. Analogamente le istruzioni ADD '701 e STA '507 (ADD '701 = somma il contenuto della locazione '701 con il contenuto del registro A e STA '507 = immagazzina il contenuto del registro A nella locazione di memoria '507) permetteranno rispettivamente di aggiungere aritmeticamente il contenuto della locazione indicata al contenuto di A, con il risultato di nuovo in A (cancellando così la precedente scrittura) e di trasferire il contenuto del registro A, qualora lo si voglia conservare, nella locazione indicata.

Ora che conosciamo il significato ed il modo di operare di queste tre istruzioni, possiamo scrivere in "assembler" uno dei più elementari "pro-

FIG. 39 - Diagramma di flusso: "sveglia al mattino"

grammi":

LDA '107

ADD '701

STA '507

Come risultato avremo nella locazione '507, la somma dei contenuti delle locazioni '107 e '701. Oltre a queste semplici istruzioni di base ve ne sono altre un poco più complesse in rapporto alle diverse funzioni che singolarmente possono svolgere. Un esempio interessante è l'istruzione CAS (compare and skip = paragona e salta) la quale confronta il contenuto del registro A con il contenuto di una certa locazione di memoria e quindi determina differenti azioni a seconda che $(A) \geq (\text{locazione } X)$. Supponiamo che, in un sistema di acquisizione dati, una certa variabile non debba scostarsi da un valore predeterminato e che, se ciò avviene, debba essere segnalata una situazione di allarme. L'istruzione ora citata organizzerà tutto questo. Infatti se il dato in ingresso, che sappiamo di trovare nel registro A è minore del nostro valore di soglia (supposto nella locazione '412) il programma potrà proseguire con l'istruzione successiva. Se $(A) > ('412)$ allora l'istruzione CAS provvede ad un "salto" delle due istruzioni successive, facendo sì che il programma continui in una terza locazione ove il programmatore avrà previsto una determinata azione conseguente alla situazione verificatasi.

Se infine $(A) = ('412)$ il programma salterà la istruzione immediatamente successiva ed ancora sarà compito del programmatore prevedere, nella locazione di arrivo dopo il "salto", l'azione successiva da prendere (ad esempio la stampa di un allarme).

Da questo si vede come sia possibile, attraverso la programmazione, predeterminare spostamenti nelle varie zone di un programma così da soddisfare in momenti diversi funzioni eventualmente comuni. Quando abbiamo parlato di acquisizione dati, implicitamente abbiamo chiamato in causa l'istruzione INA (input to A = ingresso nel registro A) la quale svolge appunto la funzione di porre ad esempio un segnale analogico proveniente da un processo, opportunamente trattato e trasformato in valore digitale, nel registro A. Analogamente per inviare un dato verso l'esterno avremo una istruzione OTA (output from A = uscita dal registro A). Entrambe le istruzioni saranno seguite da un codice che identificherà o la periferica da dove il dato proviene o quella verso cui il dato deve dirigersi attraverso le vie di ingresso od uscita (input/output bus).

Prima di illustrare un esempio riassuntivo delle istruzioni analizzate, riteniamo necessario introdurre anche il concetto legato all'istruzione JMP (jump = salto), la quale effettua, quando inserita in una sequenza di istruzioni, il salto del programma alla locazione indicata (es. JMP '571).

Supponiamo ora di voler raddoppiare ripetutamente il contenuto della locazione '770 fino a che il risultato non sia divenuto maggiore di un certo valore, posto nella locazione '771 ed introdotto da una unità periferica (una telescrivente) con codice di identificazione '0004. Avremo così il seguente programma:

```
INA '0004 ingresso del valore dalla periferica nel registro A
STA '771 immagazzinamento del contenuto di A nella locazione '771
POS LDA '770 contenuto di '770 nel registro A
ADD '770 somma del contenuto di '770 a quello di A
        (equivale a moltiplicare per due)
CAS '771 confronto
JMP SUR salto ad un sottoprogramma in locazione SUR
JMP POS salto in POS per ripetere l'operazione
JMP POS salto in POS per ripetere l'operazione
SUR ... .. prima istruzione del sottoprogramma SUR
```

Questo esempio è istruttivo anche per due aspetti che vogliamo evidenziare. Infatti per ripetere una certa sequenza di istruzioni un determinato numero di volte non è necessario riscrivere continuamente le relative istruzioni. Basterà alla fine della sequenza di istruzioni predisporre una istruzione JMP, che ci riporti all'inizio della stessa, per poter ripetere anche indefinitivamente l'operazione. Questa sequenza in inglese si chiama "Loop" (= anello). Le indicazioni SUR, POS, che in fase di stesura risultano comode per facilitare la scrittura del programma, verranno poi dal programma assembler tradotte in reali posizioni di memoria all'atto della introduzione del programma nel calcolatore attraverso corrispondenti rappresentazioni binarie.

LE FUNZIONI DEL LINGUAGGIO ASSEMBLATORE

I programmi sono scritti in linguaggio simbolico ed abbiamo visto alcuni esempi nei paragrafi precedenti. Questo modo di codificare la scrittura dei programmi è chiamato appunto "linguaggio assembler".

Il *programma assembler*, cioè una specie di "traduttore" che fa parte

dei programmi standard di molti calcolatori, provvederà poi a trasformare ciascuna istruzione simbolica nella corrispondente sequenza di uno e zeri ed a fissarla in una determinata posizione della memoria del calcolatore. Vedremo ora come questo avviene, tenendo presente che la maggior parte dei minicalcolatori di processo utilizzano come supporto per la introduzione dei dati il nastro di carta perforato.

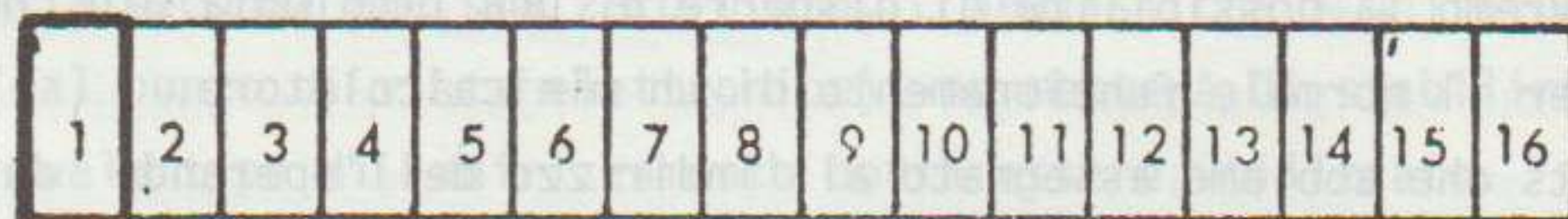
Su questa striscia di carta viene appunto impresso, attraverso la presenza o l'assenza di determinati fori, il programma (che dovrà essere eseguito dal calcolatore) secondo un'appropriata convenzione espressa per esempio dal codice ASCII (vedi appendice) ove ogni carattere, lettera, numero o segno convenzionale, è rappresentato da otto bits. Questo però non è che il primo passo verso la traduzione del nostro programma in forma effettivamente utilizzata dal calcolatore.

Infatti la parola del calcolatore è composta da 16 bits che rappresenteranno sia il codice dell'istruzione, sia l'indirizzo dell'operando; nel codice ASCII invece, essendo ogni carattere espresso da otto bits, già l'espressione mnemonica LDA occuperà 24 bits. Se poi teniamo presente che altre conformazioni saranno necessarie ai fini operativi, vedremo facilmente come ciò che in memoria occuperà 16 bits, in ASCII è rappresentato da più parole (cioè da più gruppi di 16 bits).

Dovremo quindi prevedere, soprattutto dal punto di vista del software, la possibilità di rendere "digeribile" da parte del calcolatore quanto contenuto nel nastro di partenza (source tape).

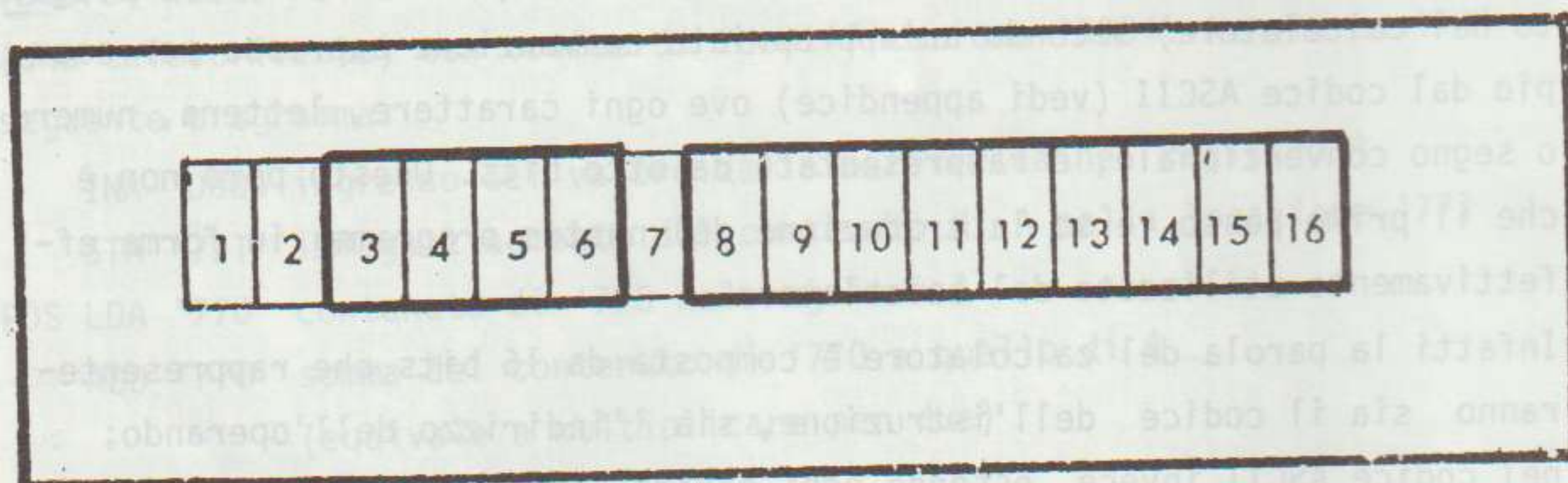
Prima di vedere quale è il successivo passo di questo complesso processo, rivediamo come vengono organizzati nella memoria di un calcolatore digitale i dati numerici o le istruzioni operative.

Sempre limitandoci a parole di 16 bits, un dato numerico verrà rappresentato in memoria con questa configurazione:



Il bit 1 servirà per dare un segno algebrico al valore numerico (uno: valore negativo, zero: valore positivo) mentre i rimanenti quindici bits serviranno per rappresentare l'entità del numero stesso. Quindi, tenendo

conto di quanto già visto sulla rappresentazione numerica in binario, potremo rappresentare il numero $+32.767_{10}$. Questo comunque non è il numero massimo rappresentabile, giacché utilizzando l'aritmetica "floating point" sarà possibile considerare numeri comunque grandi. Una istruzione che faccia riferimento alla memoria per la sua esecuzione (LDA, STA, per esempio) avrà invece, in generale e sempre con parole di 16 bits, questa configurazione:



Il bit 1 (se zero o uno rispettivamente) indica se ci troviamo nel caso di indirizzamento diretto o indiretto, mentre il bit 2 identifica se uguale ad uno, la presenza di un particolare tipo di indirizzamento attraverso uno o più registri indice. I bits da 3 a 6 indicano il tipo di istruzione che dovrà essere eseguita (quindi corrisponderanno al mnemonico appropriato), mentre il bit 7 darà un'indicazione sulla effettiva posizione di memoria ove ci troviamo ad operare. Infine i bits da 8 a 16 contengono l'indirizzo dell'operando, cioè l'indirizzo della locazione di memoria sul cui contenuto operare. In particolare i quattro bits destinati a rappresentare in binario il codice di una istruzione, possono dare luogo ad una combinazione di $2^4 = 16$ differenti gruppi di zero ed uno. Cioè altrettante istruzioni. Se consideriamo che altre istruzioni sono dotate di un diverso numero di bits per la parte che compete alla loro identificazione, avremo la possibilità di disporre di una tipologia più che sufficiente per il normale funzionamento di un minicalcolatore.

I nove bits che abbiamo assegnato all'indirizzo dell'operando daranno $2^9 = 512_{10} = 777_8$ diverse combinazioni. Cioè 512 diversi indirizzi. Ma questo limiterebbe in modo inaccettabile la possibilità di indirizzamento dell'intera memoria. Attraverso il già citato bit 7 sarà possibile appunto risolvere in parte questa difficoltà. Infatti immaginando di dividere come in figura una unità di 4K (4096 parole) in otto settori di 512 parole avremo la possibilità ponendo ad uno il bit 7, di indirizzare nello

stesso settore operativo, oppure con lo stesso bit uguale a zero, di indirizzare nel settore "zero":

0	000 - '777	} 4K
1	000 - '777	
2	000 - '777	
3		
4		
5		
6		
7	000 - '777	

In questa impostazione, che è comune alla maggior parte dei minicalcolatori oggi in commercio, il settore 0 assume un aspetto particolare. Attraverso questo settore è quindi possibile effettuare l'indirizzamento diretto di 1024 parole. E' comunque evidente anche con questo artificio, la limitazione di indirizzamento della memoria. Quindi la possibilità di indirizzamento di tutte le locazioni di memoria verrà soddisfatta attraverso la funzione dell'"indirizzamento indiretto". In questo caso bisogna porre molta attenzione al fatto che ora ciò che è rappresentato in binario nei bits da 8 a 16 dell'istruzione non è più l'indirizzo del contenuto su cui operare, ma un indirizzo dove è contenuto l'indirizzo dell'operando effettivo (EA).

In questo caso il settore zero (o settore base) viene spesso usato per memorizzare gli indirizzi indiretti (address links). Teniamo inoltre presente che la parola utilizzata per l'indirizzo effettivo avrà disponibili per questo scopo i bits dal 2 al 16 e non più solo da 8 a 16 come nel primo indirizzamento.

Ripetiamo che la condizione di indirizzamento diretto od indiretto è dato dal bit 1 a seconda che sia zero o uno. L'analisi che l'unità centrale farà poi (dal punto di vista elettronico), permetterà di stabilire automaticamente quale tipo di indirizzamento dovrà essere utilizzato.

Ovviamente potranno essere utilizzati congiuntamente i singoli effetti dei bits 1 e 7. Potremo così indirizzare senza problemi tutta la memoria ed anche utilizzare successivi indirizzamenti indiretti.

Nella fase di scrittura dei programmi, il carattere mnemonico che identifica un indirizzamento indiretto è un'asterisco (*) posto fra l'espressione mnemonica che rappresenta l'istruzione e l'indirizzo in codice otta-

le (LDA * '34). Abbiamo anche accennato all'indirizzamento con registro indice e anche se non vogliamo addentrarci in questo argomento diremo che in questo caso l'indirizzo effettivo del contenuto sul quale operare, sarà ottenuto sommando al contenuto dei bits $8 \div 16$, il contenuto del registro X (registro indice). Ricordiamo che questo indirizzamento viene caratterizzato dal contenuto del bit 2. Il nuovo indirizzo sarà quello effettivo cercato. Una istruzione tipica di questo modo di operare è: LDX.

Vediamo ora di riprendere il nostro discorso interrotto a proposito del come tradurre un certo programma in una forma binaria finale "digeribile" da parte del calcolatore. Il nastro perforato sul quale è stato trasferito il programma scritto in linguaggio simbolico (preparato senza l'intervento del calcolatore), secondo un determinato codice (ASCII) e che chiamiamo in inglese *source tape* dovrà essere ora tradotto non solo in forma compatibile al calcolatore ma anche ciascuna sua istruzione o dato dovranno essere associati ad un reale indirizzo di memoria, al fine di essere inequivocabilmente individuati dal programma.

A questo punto provvederà un programma detto *assembler* che introdotto preventivamente nel calcolatore "digerirà" il nastro source, a sua volta introdotto, per fornire all'uscita su un perforatore collegato al calcolatore un nuovo nastro detto *object*. Questo programma *assembler* esplicherà anche la funzione di verifica degli eventuali errori presenti nel programma (funzione diagnostica). Come risultato quindi, avremo un nuovo nastro detto "object", sul quale avremo rappresentato le istruzioni ed i dati nella loro forma finale binaria (e quindi non più in codice ASCII) ma dove ancora l'indirizzo sarà espresso in 14 bits. Questo in pratica comporterà di occupare tre unità base di formazione (frame) sul nastro perforato "object": cioè in totale 24 bits per informazione. A questo punto utilizzando un'altro programma standard (loader program) ed attraverso un lettore veloce di nastro, sarà possibile caricare in memoria del calcolatore il programma originale, ormai "digeribile" attraverso la trasformazione contenuto nel nastro "object", e renderlo così operativo. Una delle funzioni del programma "loader" sarà appunto quella di tradurre gli indirizzi di 14 bits, assegnando a ciascuno di essi un determinato indirizzo indiretto, così da ottenere in definitiva una espressione contenibile in 16 bits totali. Questa funzione del "loader" è chiamata in inglese "desectorizing", il che significa appunto definire gli indirizzi reali tenendo conto dei diversi settori di memoria interessati.

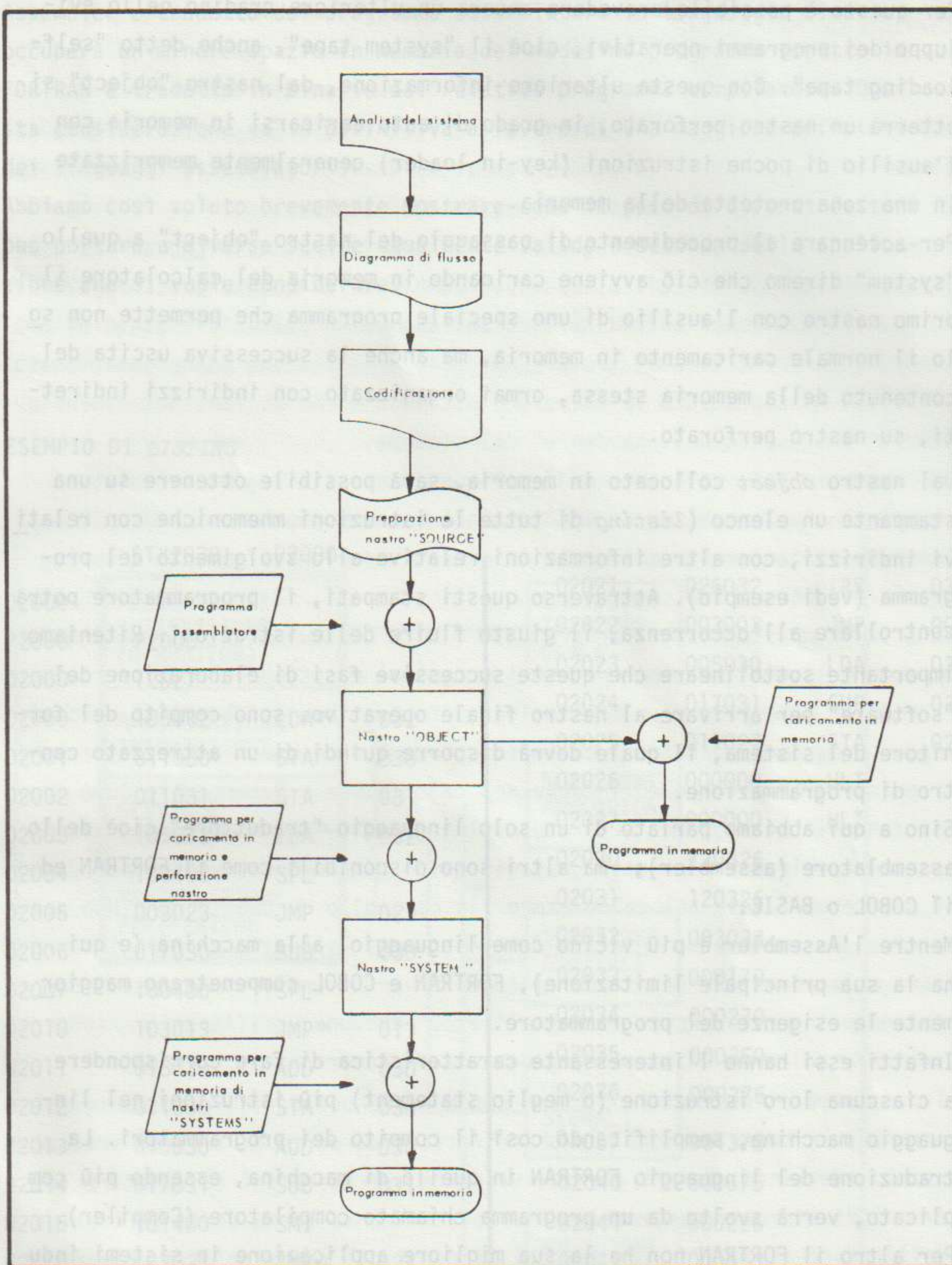


Fig. 40

Questo metodo di porre in memoria i programmi affinché essi possano operare sul sistema, richiede però una estensione della capacità della memoria stessa, superiore a quella realmente necessaria per lo svolgimento di un determinato programma.

Per questo è possibile prevedere ancora un ulteriore gradino nello sviluppo dei programmi operativi, cioè il "system tape", anche detto "self-loading tape". Con questa ulteriore informazione, dal nastro "object" si otterrà un nastro perforato, in grado di auto-caricarsi in memoria con l'ausilio di poche istruzioni (key-in-loader) generalmente memorizzate in una zona protetta della memoria.

Per accennare al procedimento di passaggio del nastro "object" a quello "system" diremo che ciò avviene caricando in memoria del calcolatore il primo nastro con l'ausilio di uno speciale programma che permette non solo il normale caricamento in memoria, ma anche la successiva uscita del contenuto della memoria stessa, ormai organizzato con indirizzi indiretti, su nastro perforato.

Dal nastro *object* collocato in memoria, sarà possibile ottenere su una stampante un elenco (*listing* di tutte le istruzioni mnemoniche con relativi indirizzi, con altre informazioni relative allo svolgimento del programma (vedi esempio). Attraverso questi stampati, il programmatore potrà controllare all'occorrenza, il giusto fluire delle istruzioni. Riteniamo importante sottolineare che queste successive fasi di elaborazione del "software" per arrivare al nastro finale operativo, sono compito del fornitore del sistema, il quale dovrà disporre quindi di un attrezzato centro di programmazione.

Sino a qui abbiamo parlato di un solo linguaggio "traduttore" cioè dello assemblatore (assembler); ma altri sono disponibili come il FORTRAN ed il COBOL o BASIC.

Mentre l'Assembler è più vicino come linguaggio alla macchina (e qui ha la sua principale limitazione), FORTRAN e COBOL compenetrano maggiormente le esigenze del programmatore.

Infatti essi hanno l'interessante caratteristica di fare corrispondere a ciascuna loro istruzione (o meglio statement) più istruzioni nel linguaggio macchina, semplificando così il compito dei programmatori. La traduzione del linguaggio FORTRAN in quello di macchina, essendo più complicato, verrà svolto da un programma chiamato compilatore (Compiler).

Per altro il FORTRAN non ha la sua migliore applicazione in sistemi industriali o scientifici, ove i programmi introdotti nel calcolatore sono legati in modo definitivo alla applicazione e non richiedono modifiche sostanziali. In altre parole, sistemi ove l'utilizzatore non ha motivo di intervenire sui programmi se non con modifiche marginali. Come considerazione quantitativa possiamo anche dire che un programma scritto in

Assembler e tradotto dal programma assemblatore in linguaggio binario, occuperà un minore spazio in memoria del medesimo programma scritto in FORTRAN e tradotto in binario dal relativo programma compilatore. Questa considerazione va in definitiva a favore di una maggiore efficienza dei linguaggi assembler.

Abbiamo così voluto brevemente mostrare come il peso di diversi fattori può portare a diverse scelte ugualmente valide a seconda della applicazione che si vuole considerare.

ESEMPIO DI LISTING

	1TX2030	P2000			
02000			02021	025032	IRS 032
P2000	P2000		02022	003003	JMP 003
02000	T2027		02023	005030	LDA 030
02000	105032	LDA* 032	02024	017031	SUB 031
02001	011030	STA 030	02025	011027	STA 027
02002	011031	STA 031	02026	000000	HLT
02003	105032	LDA* 032	02027	000000	HLT
02004	100400	SPL	02030	120326	
02005	003023	JMP 023	02031	120326	
02006	017030	SUB 030	02032	003033	
02007	100400	SPL	02033	000170	
02010	103013	JMP* 013	02034	000270	
02011	015030	ADD 030	02035	000350	
02012	011030	STA 030	02036	000376	
02013	015030	ADD 030	02037	001370	
02014	017031	SUB 031	02040	001575	
02015	101400	SMI	02041	000226	
02016	103021	JMP* 021	02042	000653	
02017	015031	ADD 031	02043	000765	
02020	011031	STA 031	02044	100000	

CONSIDERAZIONI FINALI

Nei precedenti paragrafi abbiamo considerato i vari componenti di un programma e come si arrivi, dal programma scritto con espressioni mnemoniche, alla forma finale "digeribile" da parte del calcolatore.

In realtà la stesura dei programmi e la loro interconnessione per renderli operativi e funzionali è assai più complessa di quanto possa apparire. Solo una vasta esperienza di programmazione ed un nucleo affiatato di programmatori, permetterà di trovare non solo la soluzione ad un determinato problema, ma soprattutto la soluzione ottimale anche ai fini del dimensionamento successivo dell'hardware" del sistema.

Tutto questo poi, vale ancora di più quando si utilizza un calcolatore ed il relativo programma in applicazioni in tempo reale, cioè con il calcolatore costantemente collegato in entrata ed eventualmente in uscita con il processo sotto controllo.

Infatti in questo caso il programmatore nel "costruire" il software dovrà prevedere la utilizzazione dei vari programmi e sottoprogrammi (o routines) in modo da determinare un flusso logico in funzione dei tempi di esecuzione dei programmi, dei tempi di intervento sul processo e delle interruzioni prioritarie esterne ed interne del calcolatore. Determinato come questi fattori influiscono sul flusso logico della programmazione, viene spontaneo chiedersi che cosa gestirà queste sequenze una volta introdotto il programma finale nel calcolatore. Il controllo di questo continuo passare da una sezione ad un'altra del programma, la gestione della interruzione e la scansione del tempo al quale i vari eventi (rappresentati da routine o ingresso/uscita di segnali ecc) debbono verificarsi, sono funzioni svolte da un apposito programma di supervisione o "monitor".

Il programma di supervisione tiene conto anche delle richieste contemporanee per la medesima funzione o periferica, da parte di diversi sottoprogrammi, adeguando così la elevata velocità operativa del calcolatore (in termini di microsecondi) con quella relativamente lenta della periferica (dell'ordine dei millisecondi o dei secondi). Il concetto di programma "monitor" può essere poi messo in relazione con quello di "multiprogrammazione" e "time-sharing" (letteralmente: divisione dei tempi).

Queste due tecniche di lavoro del calcolatore hanno in comune la possibilità di fare eseguire le numerose funzioni, esplicate dai diversi sottoprogrammi o routines di sistema, in modo da ottimizzare il tempo e fare

apparire contemporaneamente ciò che viene svolto sequenzialmente, grazie alla velocità ed alla funzionalità operativa. Ad esempio il trasferimento di un carattere della unità centrale alla corrispondente scrivente richiede circa 100 msec: per trasferire un secondo carattere il calcolatore dovrebbe attendere quindi un tempo grandissimo. Immaginiamo invece di porre il carattere da trasferire in un registro tampone che mantenga l'informazione, lasciando così il calcolatore libero di trasferire in ingresso un carattere da un lettore di nastro all'unità centrale. Le funzioni scrittura e lettura, in realtà sequenziali, appariranno simultanee, dati i tempi in giuoco.

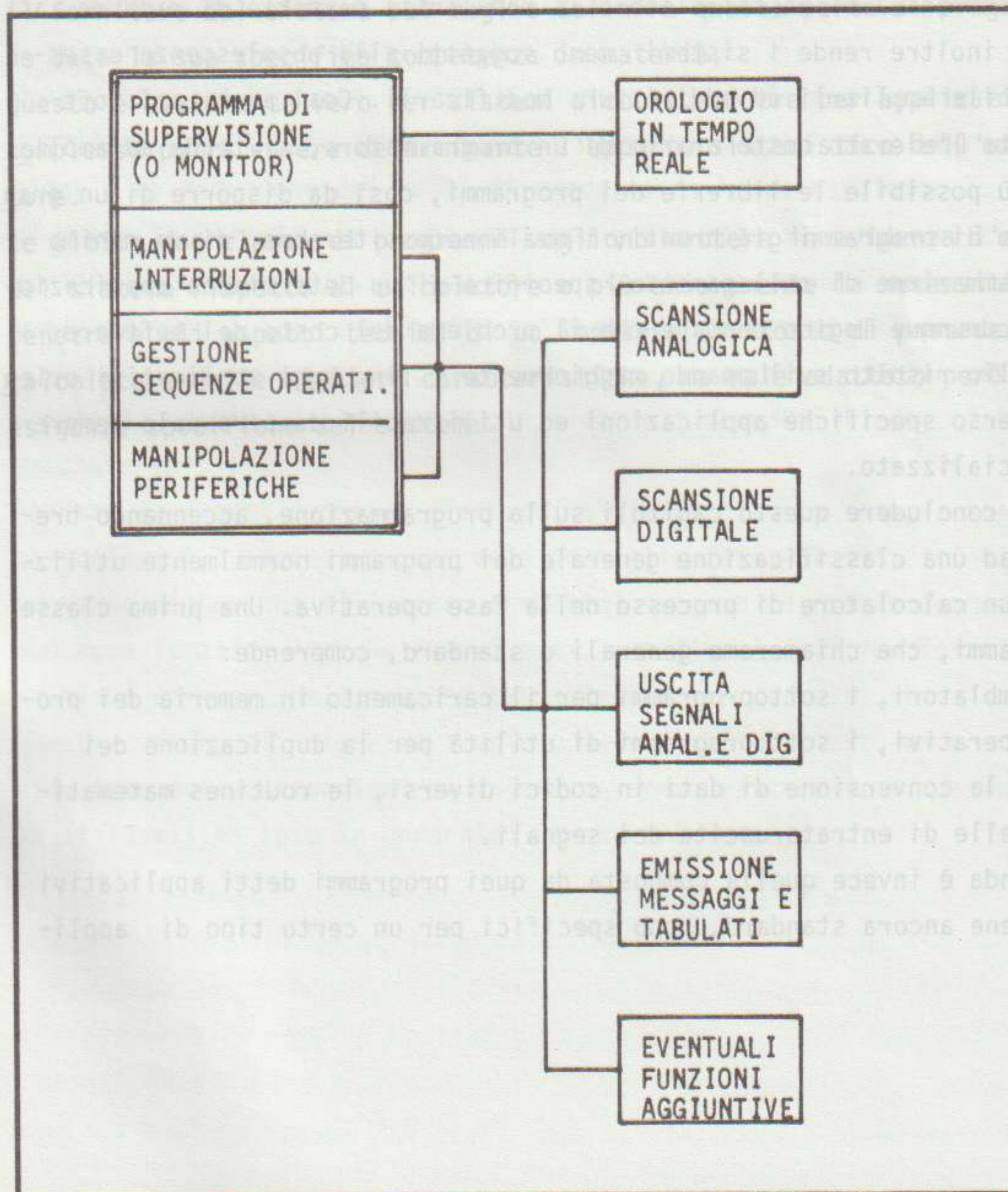


Fig. 41

Una differenza però sostanziale tra multiprogrammazione e "time-sharing" sta nel fatto che la prima mette a disposizione dei diversi programmi intervalli di tempo strettamente legati alle necessità operative, mentre il time-sharing dispone per ciascun utilizzatore esterno, un intervallo di tempo stabilito e sempre uguale che sia utilizzato o no. Un esempio di "time-sharing" nelle applicazioni scientifiche è la gestione di più gascromatografi da parte di un calcolatore. Per effetto di interruzioni automatiche date dall'orologio principale del calcolatore (per esempio: ogni 5 msec) il calcolatore si dedicherà per quel tempo ad uno strumento, per collegarsi al successivo nei seguenti cinque millisecondi e così via. Durante l'analisi, o subito dopo, sarà disponibile il risultato. Da tutte le considerazioni precedenti si può anche trarre una considerazione generale di modularità anche per la programmazione. Nella parte elettronica di un calcolatore la modularità permette di ridurre i costi ed inoltre rende i sistemi meno soggetti alla obsolescenza essendo possibile ampliarli od addirittura modificarli. Così anche nel software, dato l'elevato costo orario di un programmatore, si tende ad ampliare il più possibile le librerie dei programmi, così da disporre di un grande numero di programmi già pronti, i quali necessiteranno di un minimo di programmazione di collegamento, specifica di un determinata applicazione. E' comunque logico pensare che il problema del costo del software sarà meglio risolto sviluppando maggiormente i linguaggi applicativi orientati verso specifiche applicazioni ed utilizzabili da personale sempre meno specializzato.

Vogliamo concludere questo capitolo sulla programmazione, accennando brevemente ad una classificazione generale dei programmi normalmente utilizzati da un calcolatore di processo nella fase operativa. Una prima classe di programmi, che chiameremo generali o standard, comprende: gli assembler, i sottoprogrammi per il caricamento in memoria dei programmi operativi, i sottoprogrammi di utilità per la duplicazione dei nastri e la conversione di dati in codici diversi, le routines matematiche e quelle di entrata/uscita dei segnali.

Una seconda è invece quella composta da quei programmi detti applicativi che sebbene ancora standard, sono specifici per un certo tipo di appli-

na e "time-sharing" (letteralmente: divisione dei tempi). Queste due tecniche di lavoro del calcolatore hanno in comune la possibilità di fare eseguire i programmi, applicati dai diversi sottoprogrammi o routines, in modo da ottimizzare il tempo e fare

cazione industriale o scientifica. Preparare questi programmi richiederà competenza anche sistemistica e strumentale e quindi la loro fornitura potrà essere fatta soprattutto da chi è familiare con entrambi gli aspetti: calcolatore ed applicazione.

In questo modo è anche possibile ridurre l'aggravio di costo sull'utilizzatore in quanto la caratteristica di plurivalenza di questi programmi permetterà la loro utilizzazione su sistemi diversi ma con funzioni identiche.

Infine possiamo anche prevedere una classe ove includere i programmi applicativi specificatamente chiesti da un utilizzatore.

Questi programmi hanno la sostanziale caratteristica di essere estremamente particolari ed inutilizzabili per altre applicazioni. In generale sono la parte aggiuntiva ai precedenti due tipi di software. Anche qui il fornitore del sistema può meglio definire questa parte di programmazione data la sua specifica competenza in materia.

Questo è ovviamente vero per sistemi piccoli, mentre per applicazioni più complesse potrà essere determinante l'apporto sistemistico dell'utilizzatore.

Le ultime due classi di programmi vogliono mettere in evidenza l'enfasi del sistema rispetto al calcolatore o più chiaramente la necessità di penetrare gli aspetti tecnici di un impianto piuttosto di possedere un calcolatore dalle migliori caratteristiche, ma male adattato però alle esigenze specifiche del sistema.

Abbiamo così quello che in elettronica viene chiamata integrazione su larga scala (LSI). Quindi su una sola pastiglia (cip) di silicio (avente dimensioni di qualche millimetro quadrato) vengono ad essere concentrate numerose funzioni logiche equivalenti a configurazioni funzionali assai critiche per una realizzazione tradizionale con circuiti ibridi.

Questi componenti, oltre al pregio come abbiamo già detto di una notevole riduzione delle dimensioni di ingombro, sono caratterizzati da una elevata affidabilità (più in generale: durata nel tempo).

APPENDICE

CENNI SUI CIRCUITI INTEGRATI

Nei capitoli precedenti abbiamo parlato dell'aspetto elettronico di un calcolatore digitale, trascurando volutamente le eventuali implicazioni circuitali ed i componenti relativi.

Vogliamo ora prendere brevemente in considerazione uno dei componenti fondamentali di un calcolatore moderno: il circuito integrato monolitico. Sappiamo che in linea generale un circuito elettronico di tipo logico, è composto da condensatori, resistori, semiconduttori. Un insieme opportunamente cablato di questi componenti, costituirà un circuito a componenti discreti RTL (Logica a transistori e resistori), DTL (Logica a diodi e transistori) o TTL (Logica a tutti transistori).

Questo tipo di circuito ha però dimensioni notevoli e nel caso di schemi complessi, richiede cablaggi assai laboriosi. Esigenze di miniaturizzazione e di riduzione dei costi di produzione, hanno richiesto lo sviluppo di nuove tecnologie che hanno portato, prima allo sviluppo dei circuiti ibridi (componenti passivi su piastrine ove era anche trasferito l'elemento attivo cioè il semiconduttore) e successivamente al processo inverso di localizzazione degli stessi elementi passivi sull'elemento di cristallo semiconduttore mediante elaborati processi di deposizione a maschere multiple.

Abbiamo così quello che in elettronica viene chiamata integrazione su larga scala (LSI). Quindi su una sola pastiglia (cip) di silicio (avente dimensione di qualche millimetro quadrato) vengono ad essere concentrate numerose funzioni logiche equivalenti a configurazioni funzionali assai critiche per una realizzazione tradizionale con circuiti ibridi.

Questi componenti, oltre al pregio come abbiamo già detto di una notevole riduzione delle dimensioni di ingombro, sono caratterizzati da una elevata affidabilità (più in generale: durata nel tempo).

CIRCUITO FLIP/FLOP

Nel corso della descrizione della struttura di un calcolatore digitale, abbiamo più volte parlato dei registri ed abbiamo visto come essi possano operare associati agli organi principali cui sono legati. Riteniamo ora interessante descrivere uno di questi registri, tenendo presente che sono strutturalmente tutti uguali.

Cominciamo quindi con il dire che un registro è formato da 16 parti elementari, uguali tra di loro, denominate *FLIP/FLOP* (F/F).

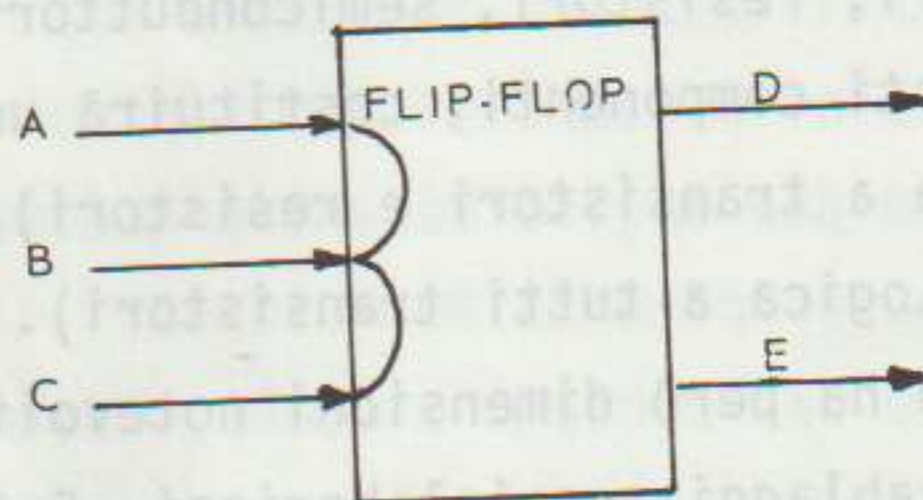


FIG 42 - IL FLIP - FLOP

Con riferimento alla fig. 42, vediamo la configurazione funzionale di un circuito F/F:

A - ingresso set

B - abilitazione ingresso

C - ingresso reset

D - uscita set

E - uscita reset

Un F/F può assumere due stati stabili: *SET* e *RESET*.

Diremo che è *SET* quando l'uscita D assume il valore logico 1 corrispondente al livello di tensione + 6 V; mentre sarà *RESET* quando ancora l'uscita D assume il valore logico 0 corrispondente al livello di tensione 0 V.

Le uscite *SET* e *RESET* sono sempre complementari tra di loro. Un F/F può cambiare di stato in relazione ai segnali presenti agli ingressi, purché questi segnali siano applicati parallelamente ad un segnale di abilitazione applicato in B. La fig. 43 rappresenta il circuito F/F, relativo al bit 5 dei registri P ed Y. Il diagramma temporale di fig. 43 illustra il trasferimento del contenuto di P in Y, (sempre relativamente al solo bit 5). Questo trasferimento è abilitato ad ogni ciclo dal temporizzatore per mezzo di un segnale applicato in B.

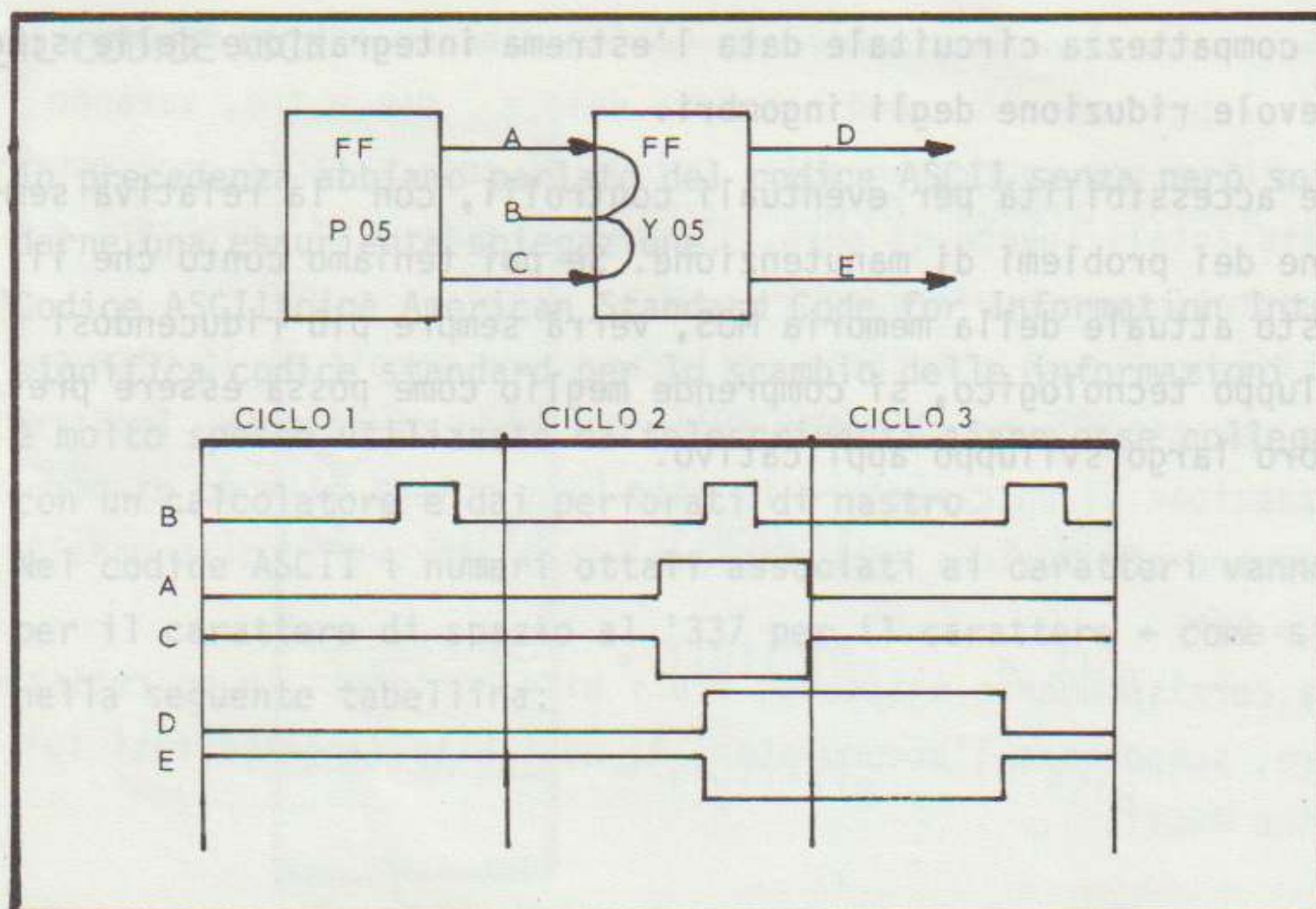


FIG. 43 - FLUSSO TRA I FLIP - FLOP

MEMORIE DI TIPO MOS (metal-oxide-semiconductor)

Lo sviluppo di applicazioni sempre più complesse che implicano tempi operativi sempre più elevati, richiede ora e sempre più nel futuro, la disponibilità di calcolatori a più alta affidabilità, capacità di memoria e velocità di esecuzione.

Questo è possibile sviluppando memorie di nuova concezione tecnologica basata sull'uso dei semiconduttori.

Queste memorie risultano così composte da elementi attivi ad alta velocità e guadagno. Inoltre maggiore è la densità di immagazzinamento delle informazioni rispetto alle attuali memorie a nucleo magnetico. Una differenza sostanziale tra i due tipi di memoria sta nel fatto che mentre in quella a nuclei magnetici un bit corrisponde ad un nucleo, nell'altra un gruppo di elementi attivi rappresenterà un bit.

Tra i vantaggi delle memorie a semiconduttore potremmo elencare:

- la possibilità di conoscere il contenuto della memoria senza cambiare il suo stato. Infatti gli elementi costituenti sono sostanzialmente dei circuiti "flip-flop" e basterà "leggere" lo stato nel quale si trovano per rilevare l'informazione.

82

IL CODICE ASCII

In precedenza abbiamo parlato del codice ASCII senza però soffermarci a darne una esauriente spiegazione.

Codice ASCII (cioè American Standard Code for Information Interchange) significa codice standard per lo scambio delle informazioni. Tale codice è molto spesso utilizzato da telescriventi siano esse collegate o non con un calcolatore e dai perforati di nastro. Nel codice ASCII i numeri ottali associati ai caratteri vanno dal '240 per il carattere di spazio al '337 per il carattere ← come si può vedere nella seguente tabellina:

CARATTERE	COD. NUMERICO	CARATTERE	COD. NUMERICO
SPAZIO	240	@	360
!	241	A	301
"	242	B	302
£	243	C	303
\$	244	D	304
%	245	E	305
&	246	F	306
•	247	G	307
(	250	H	310
)	251	I	311
*	252	J	312
+	253	K	313
,	254	L	314
-	255	M	315
.	256	N	316
/	257	O	317
0	260	P	320
1	261	Q	321
2	262	R	322
3	263	S	323
4	264	T	324
5	265	U	325
6	266	V	326
7	267	W	327
8	270	X	330
9	271	Y	331
:	272	Z	332
;	273	[	333
<	274	\	334
=	275	]	335
>	276	^	336
?	277	←	337

Riportando come esempio in binario il primo e l'ultimo codice, otterremo:
spazio '240 = 010.100.000

← '337 = 011.011.111

Poichè tutti i caratteri sono espressi in configurazione ottale di tre cifre e quella di rango superiore può assumere valore due o tre, saranno sufficienti due bits per rappresentarla. Quindi il codice ASCII sarà espresso con otto bits totali invece di nove. Il massimo numero di caratteri e codici rappresentati sono ovviamente $2^8 = 254$.

Con questo artificio è così possibile utilizzare una parola di 16 bits per trasferire due caratteri economizzando tempo e spazio di memoria. Inoltre questa organizzazione di un carattere in otto bits spiega il modo di perforazione su nastro che avviene come abbiamo visto attraverso otto canali uno per ciascun bit.

Vediamo ora la corrispondenza esistente tra i bits contenuti in un registro del calcolatore, supponiamo l'accumulatore A, ed i bits rappresentati sul nastro in codice ASCII.

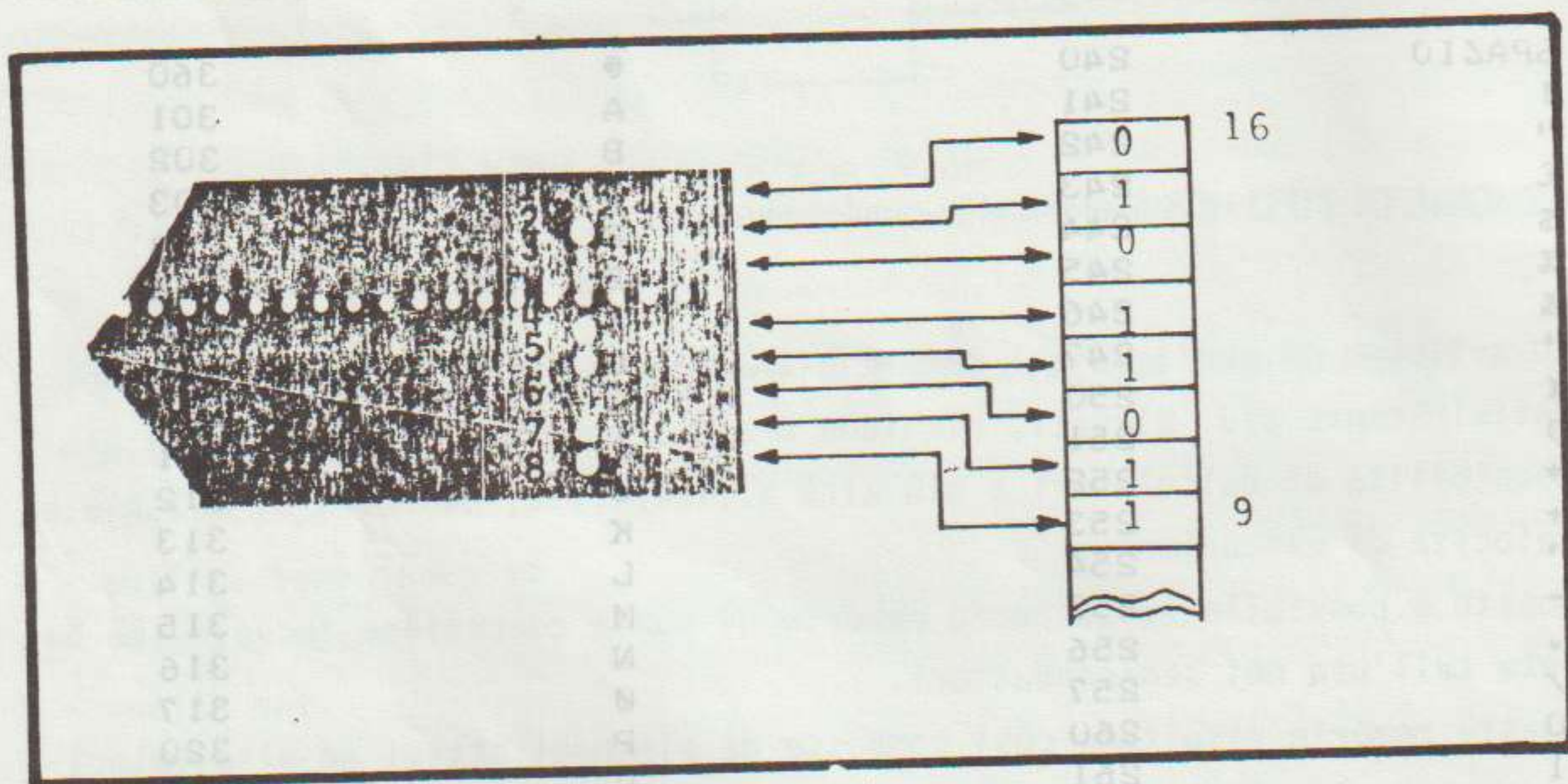
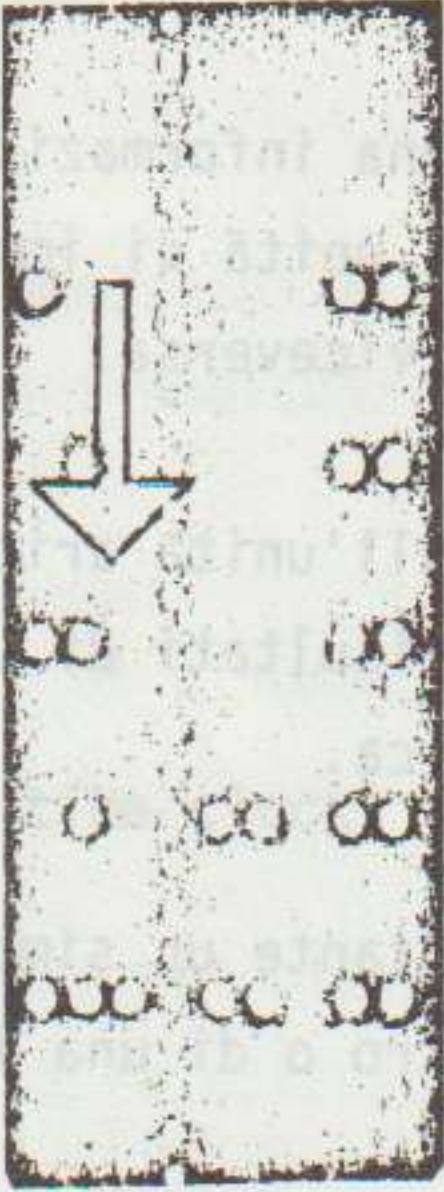
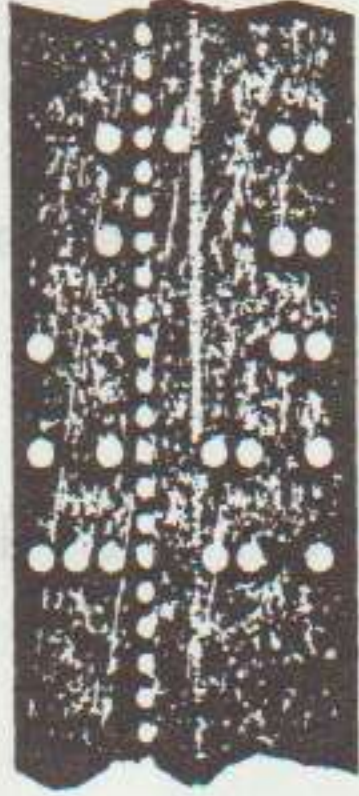
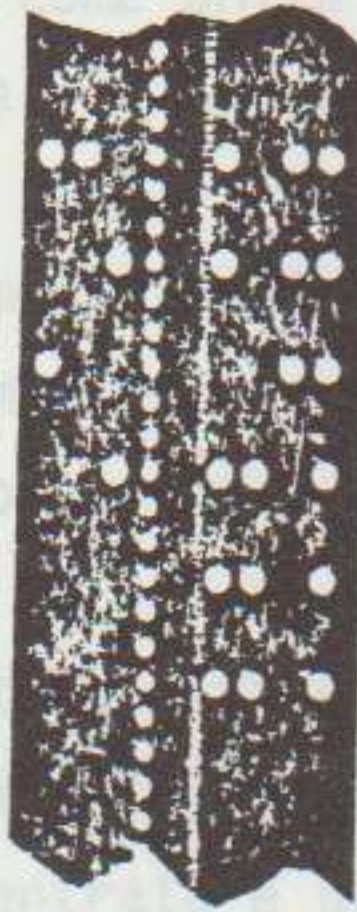


FIG. 44

E' interessante osservare come quello che nel codice ASCII è considerato il primo canale, corrisponde nel linguaggio del calcolatore al bit 16, il secondo canale al bit 15 e così via. Il motivo fondamentale di questa convenzione è la semplificazione del trasferimento dei dati dal registro al nastro di carta. Considerato infatti che l'uscita dei dati dal registro avviene da sinistra a destra, spostandosi ad ogni trasferimento di un posto, sarebbe estremamente più complesso trasferire nel primo canale il bit nove e così via. Tenendo ora presente che ad un buco corrisponde un impulso elettrico (cioè un uno) ed all'assenza di tale foro uno zero (non c'è infatti alcun impulso elettrico) vediamo questo esempio riassuntivo:

	numero binario	numero ottale	carattere
	= 11000001 ₂	= 301 ₈	= A
	= 11000010 ₂	= 302 ₈	= B
	= 11000011 ₂	= 303 ₈	= C
	= 11011010 ₂	= 332 ₈	= Z
	= 11011111 ₂	= 337 ₈	= +

Può anche essere interessante vedere come vengono rappresentate in codice ASCII due tra le più importanti istruzioni, LDA e STA.

	LDA '57		STA '400
---	---------	---	----------

GLOSSARIO DELLA TERMINOLOGIA INGLESE

Access time	Tempo impiegato da una informazione per essere trasferita da una unità di immagazzinamento al calcolatore o viceversa.
Accumulator	Registro associato all'unità aritmetica, dove vengono messi i risultati di una operazione aritmetica o logica.
Address	Identificazione, mediante un simbolo od un numero, di un registro o di una locazione di memoria.
A/D converter	Convertitore analogico/digitale.
ASCII code	Codice di trasmissione di informazioni, particolare per telescriventi.
Basic cycle	Tempo richiesto per completare le operazioni necessarie allo svolgimento di una istruzione del programma.
BCD	Codice binario per la trasmissione di informazioni digitali in generale.
Binary code	Sistema numerico avente due soli simboli uno o zero (vero o falso).
Bit	Abbreviazione di "binary digit". E' l'unità elementare del sistema binario.
Buffer	Registro di transito posto tra due unità aventi differenti tempi di esecuzione.

Byte	Unità generalmente di 8 bits.
Bulk storage	Unità (dischi, nastri magnetici, tamburi) usate come memorie di immagazzinamento per grandi quantità di informazioni.
Channel bus	Canale di collegamento tra diverse unità, usato per trasmettere dati in forma digitale.
Clock (real time clock)	Oscillatore che genera impulsi periodici per sincronizzare le operazioni interne di un calcolatore. Fornisce anche il tempo in giorni, ore e minuti.
Computer	Macchina designata per ricevere dati o segnali, eseguire operazioni programmate su di essi e fornire in uscita risultati o comandi.
Core memory	Memoria generalmente a nuclei magnetici, con accesso casuale.
Cycle time	Intervallo di tempo fra la richiesta e la ricezione delle informazioni dalla memoria. È usato in genere come unità di misura della velocità base dell'elaboratore.
Data	Termine generalmente usato per indicare una qualsiasi informazione in arrivo od in partenza dal calcolatore.
D.D.C.	Controllo diretto digitale. Metodo con cui l'elaboratore calcola le azioni di controllo, per mezzo dell'entrata e degli algoritmi ad essa connessi, per poi generare l'uscita che andrà ad attuare le decisioni prese.

Dedicated computer	Calcolatori programmati per assolvere ad una specifica applicazione industriale o scientifica.
Direct addressing	Indirizzo che indica la locazione di memoria sul cui contenuto (istruzione o dato) operare.
DMA	Accesso diretto alla memoria ad altissima velocità.
DMC	Data multiplexer control. Per introdurre in memoria una grande quantità di dati ad alta velocità
Debugging routines	Sono sotto-programmi che aiutano il programmatore a verificare e correggere i programmi operativi.
Fixed point arithmetic	Sistema di calcolo aritmetico in cui la posizione della virgola viene definita a programma.
Floating point arithmetic	Sistema di calcolo aritmetico in cui la posizione della virgola è fissa in quanto si ragiona con esponente e mantissa.
Hardware	Parte elettronica e meccanica di un progetto computerizzato.
High speed arithmetic unit	Unità elettronica del calcolatore nella quale vengono eseguite le operazioni aritmetiche e logiche.
I/O bus	Unità di ingresso/uscita (I = Input = ingresso; (O = Output = uscita)
Indirect addressing	Indirizzo il quale indica una locazione nella quale si trova un nuovo indirizzo sul contenuto del quale operare.

Instruction repertoire Insieme delle istruzioni caratteristiche di un calcolatore.

Interrupt Possibilità di interruzione sul fluire di un programma in corso di esecuzione, per considerare eventi di emergenza.

Machine language Codici numerici, quasi sempre binari, usati dall'elaboratore per la manipolazione interna dei dati.

Microprogramming Istruzioni per programmare con diversi comandi in linguaggio di macchina. Generalmente essi si trovano in una unità di memoria detta R.O.M. per aumentare la velocità delle operazioni comuni.

Modem Modulatore/demodulatore usato per modulare o demodulare i segnali in un canale di comunicazione.

Monitor Programma base di un calcolatore, che permette la esecuzione organica dei programmi applicativi.

Multiplexer Unità di controllo per canali di comunicazione che permette, attraverso una scansione successiva, il collegamento contemporaneo degli ingressi all'unità centrale.

Multiprogramming Esecuzione contemporanea di due o più programmi.

On-line Con questo tipo di collegamento il calcolatore svolge funzioni direttamente connesse con una contemporanea acquisizione dati.

Off-line	Il calcolatore svolge funzioni o manipola dati senza collegamento diretto ad un processo attivo.
Parity bit	Bit aggiunto ad una certa serie di bits, per renderli (in numero) pari o dispari ed effettuare così un controllo sulla validità del messaggio da essi rappresentato.
Peripheral	Unità periferica (lettore di nastro, telescrivente, ecc) collegata al calcolatore ed atta a ricevere od inviare messaggi ed informazioni.
R.O.M. (read only memory)	Unità di memoria le cui informazioni contenute possono essere solo lette e sulla quale è inibita la scrittura.
Real time	Cioè che viene svolto in tempo reale ed è pertinente ad un processo fisico che si sta sviluppando.
Register	Unità capace di memorizzare temporaneamente informazioni usate per elaborazioni aritmetiche o manipolazione dati.
Sector	Suddivisione della memoria per motivi pratici operativi.
Serial processing	Esecuzione sequenziale di una operazione complessa.
Shift	Indica il decalaggio nella posizione dei bits di una informazione.
Software	Programmi, routines, codici ed altre informazioni scritte, usate con calcolatori digitali al fine del loro funzionamento.
Word	Sequenza di bits che compongono una informazione od un dato. E' composta comunemente da 16, 24 o 32 bits.