A. Grasselli

INTRODUZIONE AI CALCOLATORI ELETTRONICI

Cap. III - Introduzione alla programmazione ed alla architettura del calcolatore

CAPITOLO III

INTRODUZIONE ALLA PROGRAMMAZIONE ED ALLA ARCHITETTURA DEL CALCOLATORE

"Perchè incomincio proprio dal lato più sgradevole della nostra convivenza con gli animali? Perchè il nostro amore per loro si misura proprio dai sacrifici cui siamo disposti a sobbarcarci".

Konrad Lorenz, "L'anello di re Salomone" (Adelphi, Milano 1967)

Lo straniero che visita un paese sconosciuto può, nei suoi colloqui con gli indigeni, comportarsi in due modi radicalmente diversi: cercare di imparare la lingua locale, o servirsi di un interprete. Nel primo caso, la comunicazione sarà all'inizio alquanto difficile, ed egli non potrà certamente impostare conversazioni interessanti: tuttavia, col tempo, l'acquisita padronanza della lingua gli darà modo di conoscere intimamente la civiltà, struttura, usi e costumi del paese e gli fornirà i mezzi per avvicinarsi alla psicologia degli abitanti.

Il colloquio attraverso un interprete, invece, è all'inizio strumentalmente molto più efficace, e quindi si adatta al turista frettoloso o all'uomo d'affari: questo tipo di comunicazione non potrà tuttavia mai permettere una conoscenza meno che superficiale del paese.

Analogamente, vi sono due modi per discorrere con il calcolatore: parlargli nella sua lingua (nel cosiddetto linguaggio macchina), oppure in una lingua molto più vicina alla nostra che a quella del calcolatore (in uno, cioè dei cosidetti linguaggi simbolici: FORTRAN, ALGOL, COBOL, PL/1, ecc.): nel secondo caso, il calcolatore non può direttamente comprendere, ma necessita di un programma traduttore che trasformi l'algoritmo codificato nel linguaggio simbolico in un programma in linguaggio macchina.

Mentre l'utilizzatore frettoloso può limitarsi ad apprendere quel linguaggio simbolico che più si adatta ai suoi scopi, colui che voglia studiare meno superficialmente il calcolatore deve, a parer nostro, assoggettarsi dapprima allo studio del linguaggio macchina: solamente così, infatti, gli sarà possibile avvicinare il calcolatore ed esaminare in maggiore o minore dettaglio l'architettura interna. Inoltre, la conoscenza del linguaggio macchina gli permetterà di valutare le caratteristiche recondite dei linguaggi simbolici, che altrimenti gli rimarrebbero assolutamente impenetrabili, e la maggiore o minore efficenza con cui le frasi del linguaggio simbolico verranno tradotte in istruzioni del linguaggio macchina. Si noti che anche chi abbia appreso il linguaggio macchina comunicherà spesso (o quasi sempre) con il calcolatore in uno o l'altro dei linguaggi simbolici (e qui cessa la nostra analogia con il turista di cui sopra): lo farà, però, con conoscenza di causa, sapendo ormai prevedere le reazioni dell'interlocutore ai suoi discorsi. Nell'uso comune, i linguaggi simbolici sono estremamente più compatti ed espressivi del linguaggio macchina, e quindi di esso più agevoli. Possiamo fin d'ora fare un esempio: la frase (in linguaggio ALGOL):

per i ← 1 passo 1 fino 100 esegui s ← s + a [i] ;

calcola la quantità:

$$s = a[1] + a[2] + \ldots\ldots\ldots + a[100]$$

Nella figura 3.18 , è riportato un programma per lo stesso calcolo nel linguaggio macchina di un calcolatore che verrà descritto nei paragrafi seguenti: un esame necessariamente superficiale farà apprezzare l'estrema compattezza della precedente frase ALGOL. Il linguaggio macchina è estremamente più minuzioso: dobbiamo però sobbarcarci alla fatica di apprenderlo se desideriamo acquisire una conoscenza intima del calcolatore.

In questo capitolo esamineremo un calcolatore un calcolatore che fisicamente non esiste, il CANE (Calcolatore Automatico Numerico Educativo): esso ci servirà per introdurre i concetti fondamentali della programmazione in linguaggio macchina e per prendere un primo contatto con l'architettura dei calcolatori.

Perchè esaminare un calcolatore inesistente, invece che uno dei tanti modelli disponibili sul mercato? Le ditte costruttrici di calcolatori sono di, solito (moderatamente) felici di fornire i manuali di programmazione delle loro macchine al pubblico interessato, e certamente ogni lettore potrebbe facilmente procu-

rarsene alcuni. Riteniamo tuttavia preferibile che il principiante compia i primi passi in modo diverso; rispetto ai calcolatori reali, il CANE è stato drasticamente semplificato in alcune parti, in particolare per quanto riguarda la unità di entrata ed uscita ed il loro funzionamento: questo allo scopo di illustrare i concetti essenziali della programmazione e della architettura appongiandoli ad una struttura semplice, senza dover dipanare ad ogni passo una matassa di inessenziali dettagli. Nello stesso tempo, la struttura del CANE è ancora abbastanza ricca da permettere di impostare un discorso ricco di particolari.

Il lettore noti che, nel multiforme zoo dei calcolatori reali, il CANE rappresenta, per certi suoi aspetti, una creatura mostruosa: certi suoi organi sono quasi atrofici, mentre altri sono in confronto innaturalmente sviluppati. Alcuni di questi scompensi sono stati assolutamente necessari, alla luce delle considerazioni precedenti; sull'essenzialità di altri il critico agguerrito troverà sicuramente materia di discussione. Tutte queste "mostruosità" verranno comunque esaminate a fondo nel seguito.

Abbiamo detto che il calcolatore CANE fisicamente non esiste; ciò è solo parzialmente vero. Infatti è relativamente semplice fornire un qualsiasi calcolatore (purchè esso superi certe dimensioni minime) di un programma, detto programma simulatore, che lo faccia "comportare" come il CANE (fig. 3.1): un programma simulatore è stato realizzato per il calcolatore IBM 7090 del Centro Nazionale Universitario di Calcolo Elettronico dell'Università di Pisa, ed è disponibile per chi voglia farne uso.(*)

(*) Vedi: A. Grasselli, G. Pacini, "CANE, SIMULCANE e SYSTEMCANE", Nota Tecnica C70/1 , Istituto di Elaborazione della Informazione del CNR, Pisa (gennaio 1970)

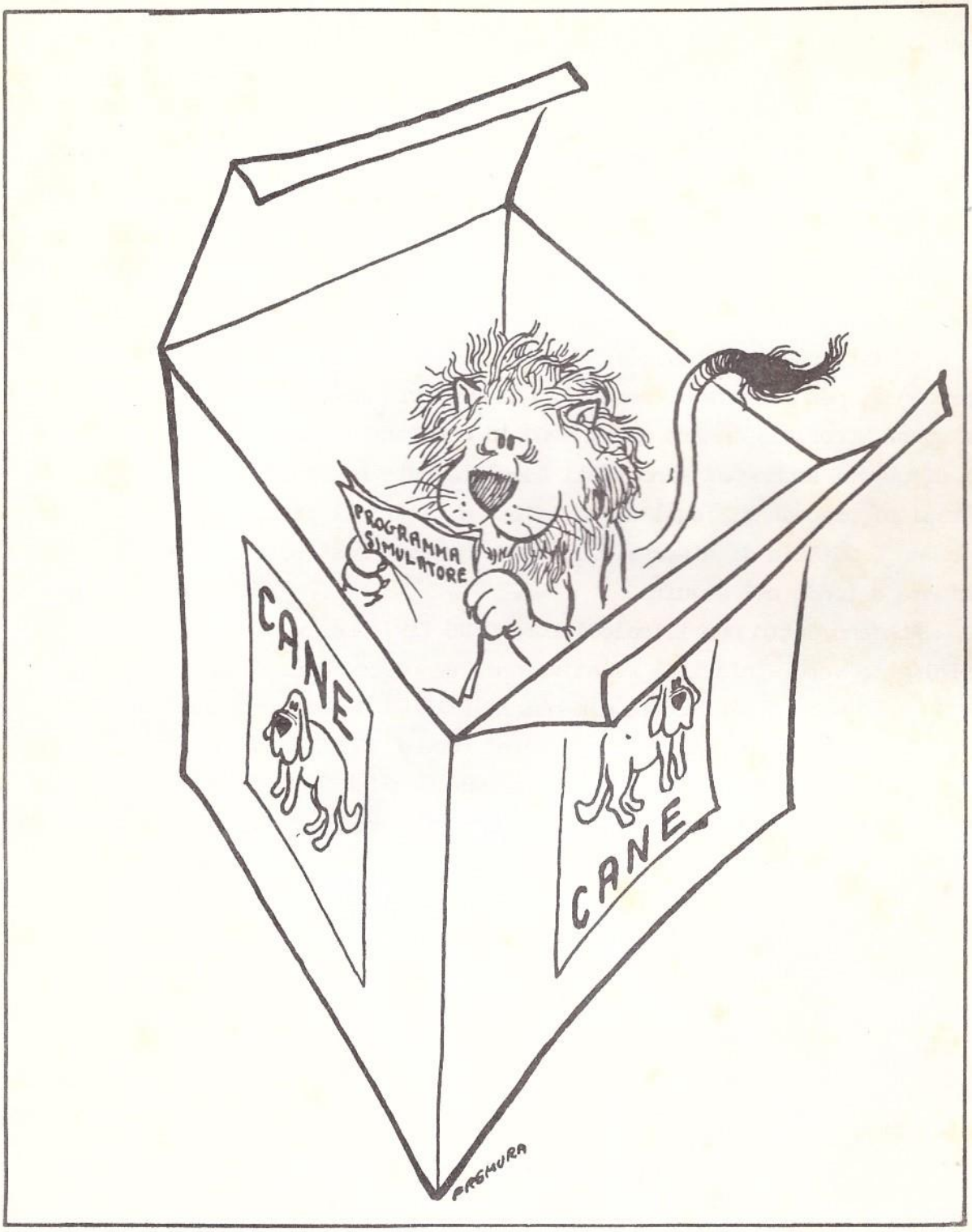


Fig. 3.1 - Il programma simulatore.

3.1 - Un primo sguardo al funzionamento del calcolatore.

Abbiamo visto nell'introduzione che un calcolatore digitale si compone di diverse "unità": la memoria, l'unità di elaborazione, il controllo, le apparecchiature di entrata ed uscita. Tralasciando per ora queste ultime, ricordiamo che la memoria serve a registrare il programma, i dati ed i risultati, l'unità di elaborazione ad eseguire le operazioni fondamentali, mentre il controllo presiede alla attività di tutti gli organi del calcolatore coordinandone le azioni.

Vediamo ora in modo grossolano che cosa avviene nel calcolatore nel corso dello svolgimento di un programma. Per semplificare il discorso, faremo riferimento ad alcuni personaggi addetti al funzionamento delle varie parti:"l'archivista" che opera nell'unità di memoria, "l'operaio" nell'unità di elaborazione, il "capufficio" nell'unità di controllo ed il "fattorino" che serve per i collegamenti (vedi fig. 3.2).

Come mostra la fig.3.2, l'archivista ha a sua disposizione una lavagna, suddivisa in un certo numero di caselle (nel seguito, queste verranno chiamate le celle della memoria); in ogni casella può essere scritta una istruzione, oppure un dato od un risultato. Prima di poter iniziare l'esecuzione di un programma qualsiasi, l'archivista deve ricevere dal fattorino un dispaccio (che proviene dalla unità di entrata) contenente il programma stesso, e deve ricopiare tale dispaccio sulla lavagna.

Nella fig. 3.3(a) lo stiamo osservando nel corso della esecuzione del programma scritto sulla lavagna: in particolare, deve venire eseguita l'istruzione nella casella 4, che dice "sommare casella 11 ad A". L'archivista ha un compito molto semplice: egli deve ricopiare l'istruzione su un foglietto, che il fattorino porterà al capufficio, unico dei nostri personaggi che è abilitato a decidere sul da farsi (fig.3.3(b)).

Il capufficio, presa visione dell'istruzione, impartisce gli ordini necessari: nella fig. 3.4(a) lo vediamo telefonare sia all'archivista che all'operaio. L'archivista eseguendo gli ordini(fig.3.4.(b)) ricopia il numero scritto nella casella 11 su un foglietto che consegna al fattorino; questi lo porterà all'operaio che,come mostra la fig. 3.5 (a),dispone di diverse "macchine" oltrechè di una piccola lavagna con alcune caselle denominate A,B,X1,X2, ecc (nel seguito queste verranno chiamate i registri della unità di elaborazione).

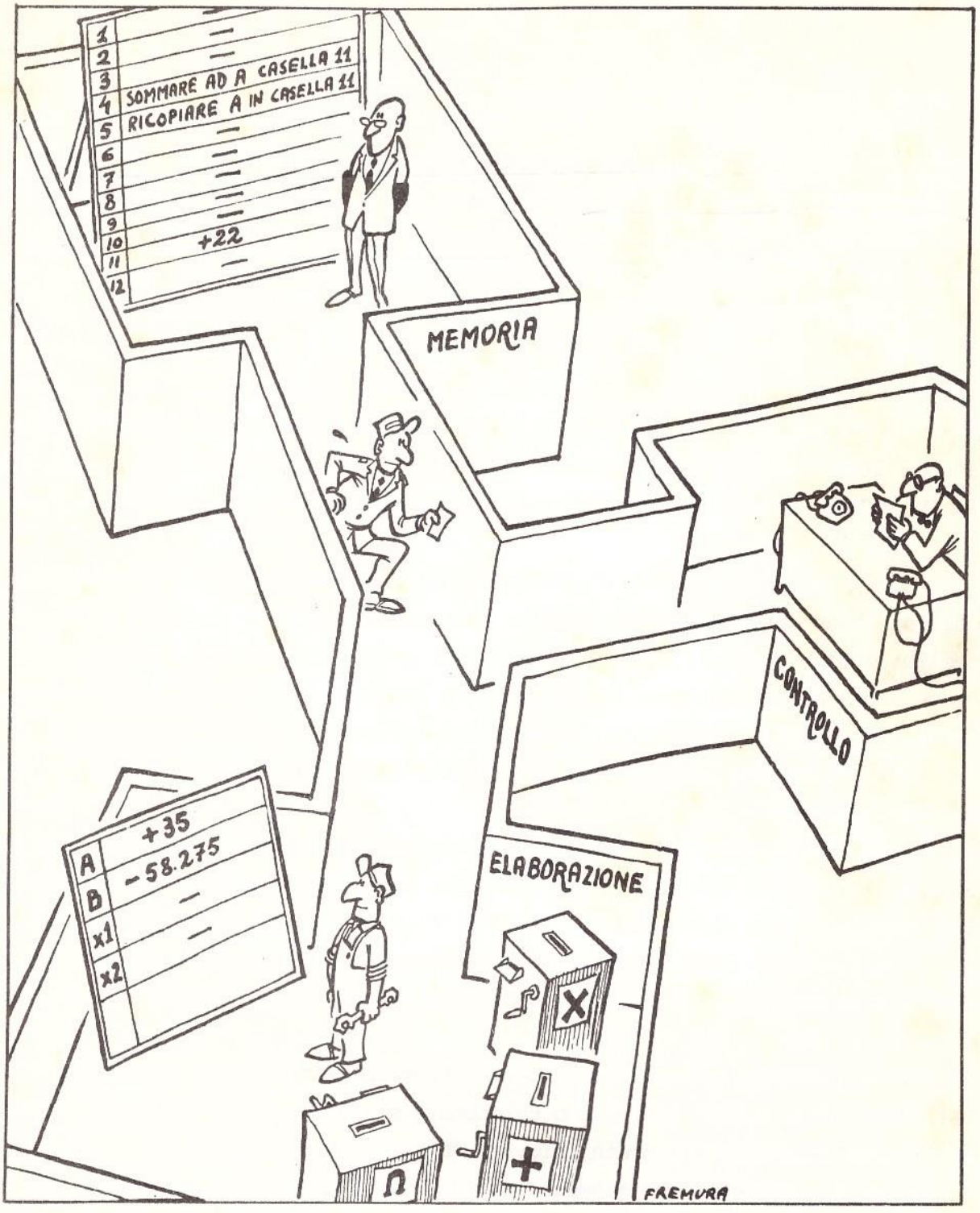


Fig. 3.2 - Le unità di memoria, controllo
ed elaborazione del calcolatore.

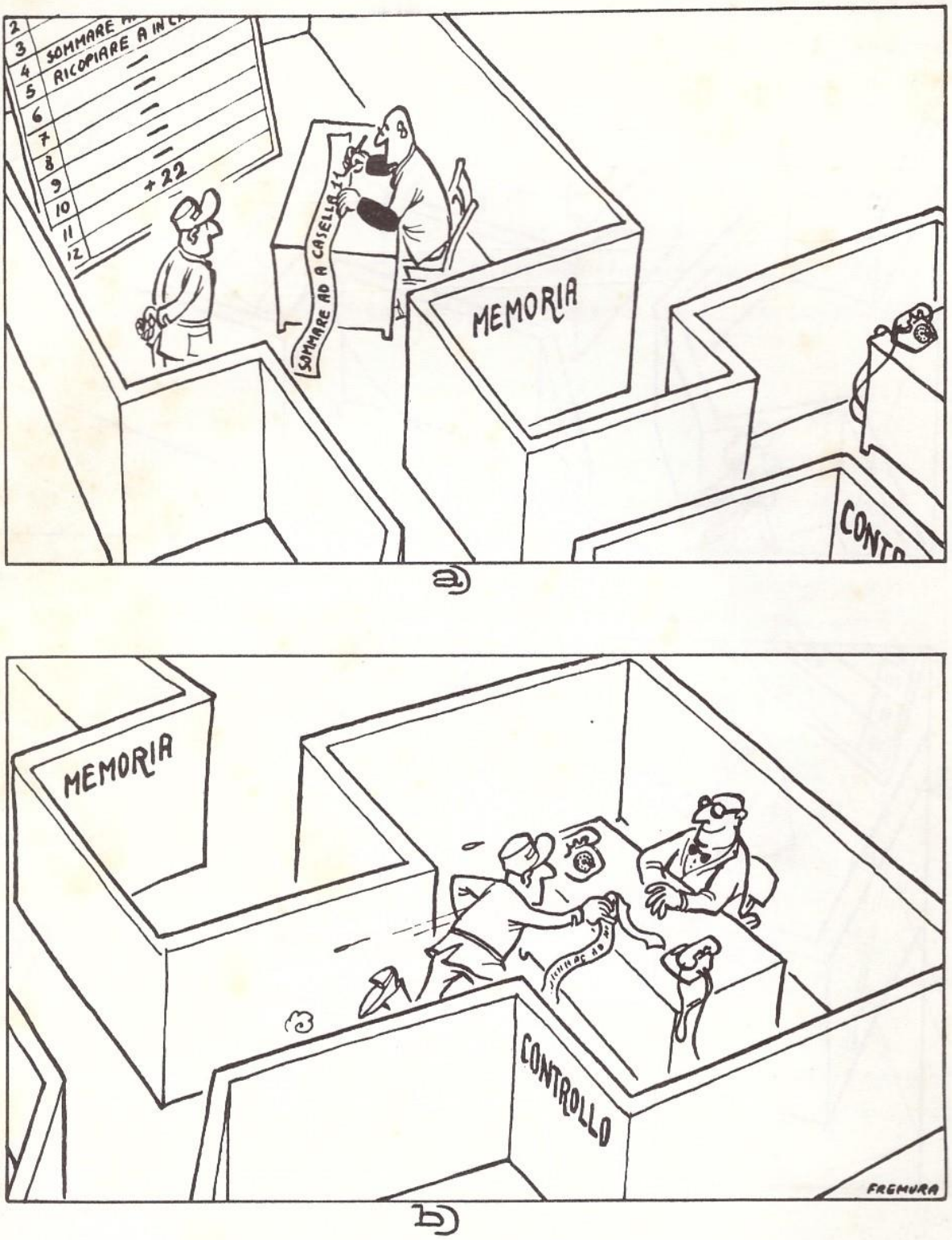
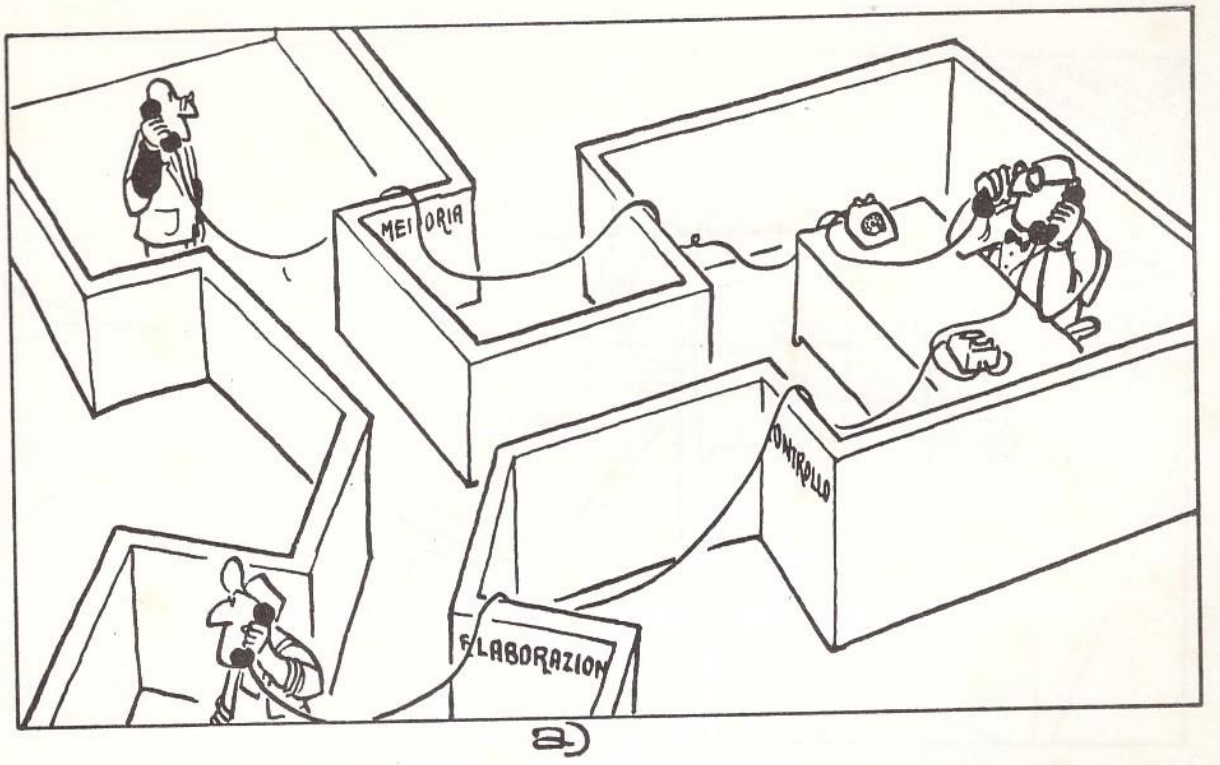


Fig. 3.3 - Fase di interpretazione dell'istruzione: (a) lettura dell'istruzione dalla memoria, (b) trasferimento dell'istruzione all'unità di controllo.



a)

b)

Fig. 3.4 - Prima parte della fase esecutiva.

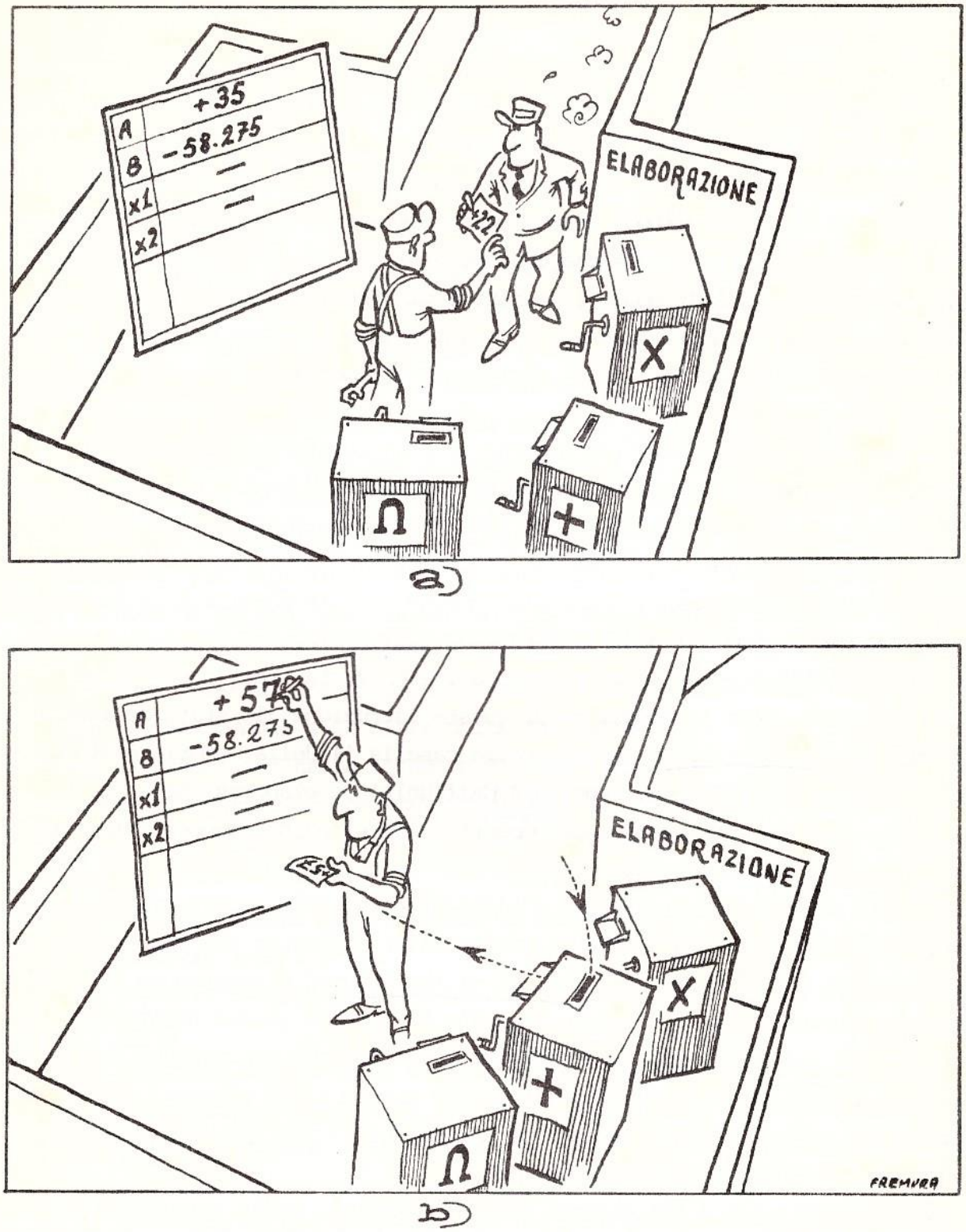


Fig. 3.5 - Seconda parte della fase esecutiva.

L'operaio deve ricopiare su un foglietto il numero scritto nella casella A della sua lavagna, introdurre questo foglietto e quello che gli ha portato il fattorino nella macchina SOMMA, e scrivere il risultato che esce dalla macchina al posto del vecchio numero nella casella A (vedi la fig. 3.5(b)). A questo punto, l'esecuzione della istruzione scritta nella casella 4 è terminata. Si noti che l'esecuzione della istruzione che abbiamo esaminato ha lasciato invariato lo stato della lavagna nella unità di memoria, mentre è cambiato lo stato della lavagna nella unità di elaborazione: la casella A ora contiene la somma del numero che vi era precedentemente scritto, e del numero scritto nella casella 11 della memoria.

In generale, l'istruzione successiva si trova nella casella successiva: nel caso che abbiamo esaminato, l'istruzione successiva è quella scritta nella casella 5; essa dice:"ricopiare A in casella 11". Gli eventi nel corso della esecuzione di questa istruzione si susseguono, come abbiamo visto per la istruzione precedente: l'archivista ricopia l'istruzione dalla casella 5, e la invia al capufficio che dirama gli ordini; eseguendo tali ordini,l'operaio ricopia il numero scritto nella casella A, e lo invia all'archivista, che lo scrive nella casella 11. Perciò, dopo l'esecuzione di questa istruzione, lo stato della unità di elaborazione non è cambiato, mentre nella casella 11 della memoria è stato ricopiato il numero scritto nella casella A dell'unità di elaborazione.

L'esecuzione di una istruzione comporta diversi eventi successivi: alcuni di questi (lettura della istruzione della memoria e suo trasferimento al controllo) sono i medesimi qualunque sia l'istruzione eseguita, e compongono la cosiddetta fase interpretativa dell'esecuzione; altri dipendono dal particolare tipo di istruzione (il CANE possiede 63 tipi di istruzione) e compongono la fase esecutiva propriamente detta. Proseguendo nella lettura di questo capitolo, il lettore noterà che la descrizione in questo paragrafo è stata volutamente semplificata: infatti, alcuni degli eventi descritti sono a loro volta composti da più eventi elementari; inoltre abbiamo taciuto su alcuni eventi non essenziali per una comprensione preliminare del funzionamento.

Si noti che tutti i calcolatori digitali operano, nella esecuzione di una istruzione, più o meno come abbiamo indicato: naturalmente, la complessità degli eventi che compongono l'esecuzione aumenta con la complessità del calcolatore.

In particolare, nei "grossi" calcolatori dell'attuale generazione (cui si accennerà nel seguito col nome di calcolatori giganti) tali eventi sono estremamente complessi.

3.2 - Celle e registri

La memoria del calcolatore CANE consta di 512 celle, come mostra la fig. 3.6(a); ogni cella (vedi la fig. 3.6(b)) contiene una configurazione di 18 informazioni binarie: si usa anche dire che le celle contengono o sono lunghe 18 bit. Le celle di memoria, come le porte delle abitazioni in una strada di città, sono contraddistinte da un numero d'ordine chiamato indirizzo: nella fig.3.1, sono indicate le celle di indirizzo 0 (prima della memoria), 1,2,226,510 e 511 (ultima cella). Il contenuto di una cella può venire interpretato dal calcolatore come una istruzione, oppure come un dato (numerico o di altro tipo); tale contenuto e, per estensione, ogni configurazione di 18 informazioni binarie, prende il nome di parola. Come mostra la fig. 3.6(b), le 18 posizioni nella parola sono numerate da 0 a 17 partendo dalla destra.

Durante l'esecuzione di un programma, l'informazione viene continuamente scambiata fra la memoria e le altre unità del calcolatore. Tutte le unità sono provviste di organi che hanno lo scopo di memorizzare piccole quantità di informazione: questi organi sono chiamati registri. Alcuni registri sono indicati nelle fig. 3.2 - 3.5 con i simboli A,B,X1,ecc. I registri del CANE sono di tre dimensioni:

a) registri lunghi come le celle di memoria cioè 18 bit,

b) registri lunghi 9 bit ,

c) registri lunghi 1 bit: questi ultimi vengono di solito chiamati indicatori.

La struttura fisica della memoria e dei registri verrà esaminata nel capitolo V; funzionalmente, il lettore può immaginare celle e registri come li mostrano le fig. 3.2-3.5, oppure, se preferisce può pensarli come scatoline che contengono informazioni binarie (per esempio, palline bianche e nere).

Il contenuto di una cella o di un registro può essere letto allo scopo di essere elaborato o trasferito altrove: la lettura non disturba il contenuto della

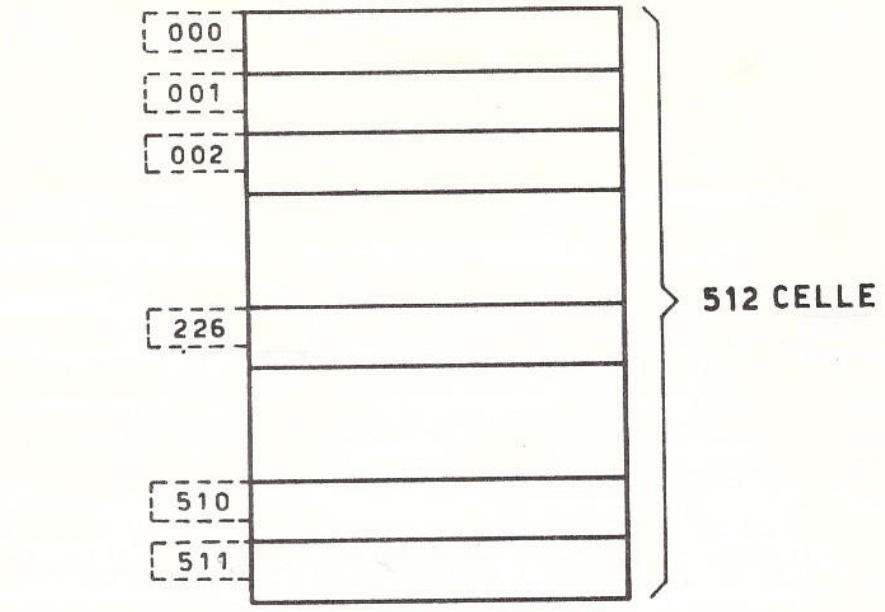


Fig. 3.6 - La memoria del CANE contiene 512 celle, lunghe 18 bit.

cella o del registro. In una cella o in un registro può venire scritta una configurazione di informazioni binarie; la scrittura distrugge il precedente contenuto della cella o del registro. Così se il contenuto di un registro che chiameremo sorgente viene ricopiato in un registro destinazione, dopo l'operazione il contenuto del registro sorgente è rimasto invariato, mentre il precedente contenuto del registro destinazione è definitivamente perduto.

3.3 - La memoria

La fig. 3.7 mostra tutti gli organi architettonici che compongono l'unità di memoria. Oltre alla memoria propriamente detta, che, come abbiamo detto, comprende 512 celle, esistono due registri: il registro di lettura e scrittura rLS, lungo 18bit, ed il registro di indirizzamento r I, lungo 9 bit.

Le celle di memoria vengono poste in comunicazione con il resto del calcolatore attraverso il registro di lettura e scrittura rLS. Istante per istante il registro rLS può comunicare con una sola cella di memoria: più precisamente, con quella cella il cui indirizzo è scritto nel registro di indirizzamento. Infatti, i 9 bit del registro rI rappresentano, nel sistema di numerazione binario, un numero compreso fra 0 e 511, e tale numero ha sempre il significato di indirizzo della cella interessata alla comunicazione.

La fig. 3.8 mostra come si svolge la lettura del contenuto di una cella di memoria; nel caso specifico, la cella interessata ha indirizzo 226; perciò durante la lettura il registro rI contiene la configurazione:

0	1	1	1	0	0	0	1	0

Infatti: $11100010_{(2)} = 226$.

Nella lettura del contenuto di una cella di memoria (fig. 3.8) il suo indirizzo, proveniente dalla unità di controllo, viene dapprima scritto nel registro rI (fig. 3.8(a)). Poi il contenuto della cella viene ricopiato nel registro rLS (fig. 3.8(b)), ed infine viene letto dal registro rLS (fig. 3.8 (c)) (per esempio per essere ricpiato nel registro rA dell'unità di elaborazione). Nella scrittura di una parola in una cella di memoria (fig. 3.9) l'indirizzo della cella viene scritto in rI (fig. 3.9(a)), poi la parola viene scritta dall'e -

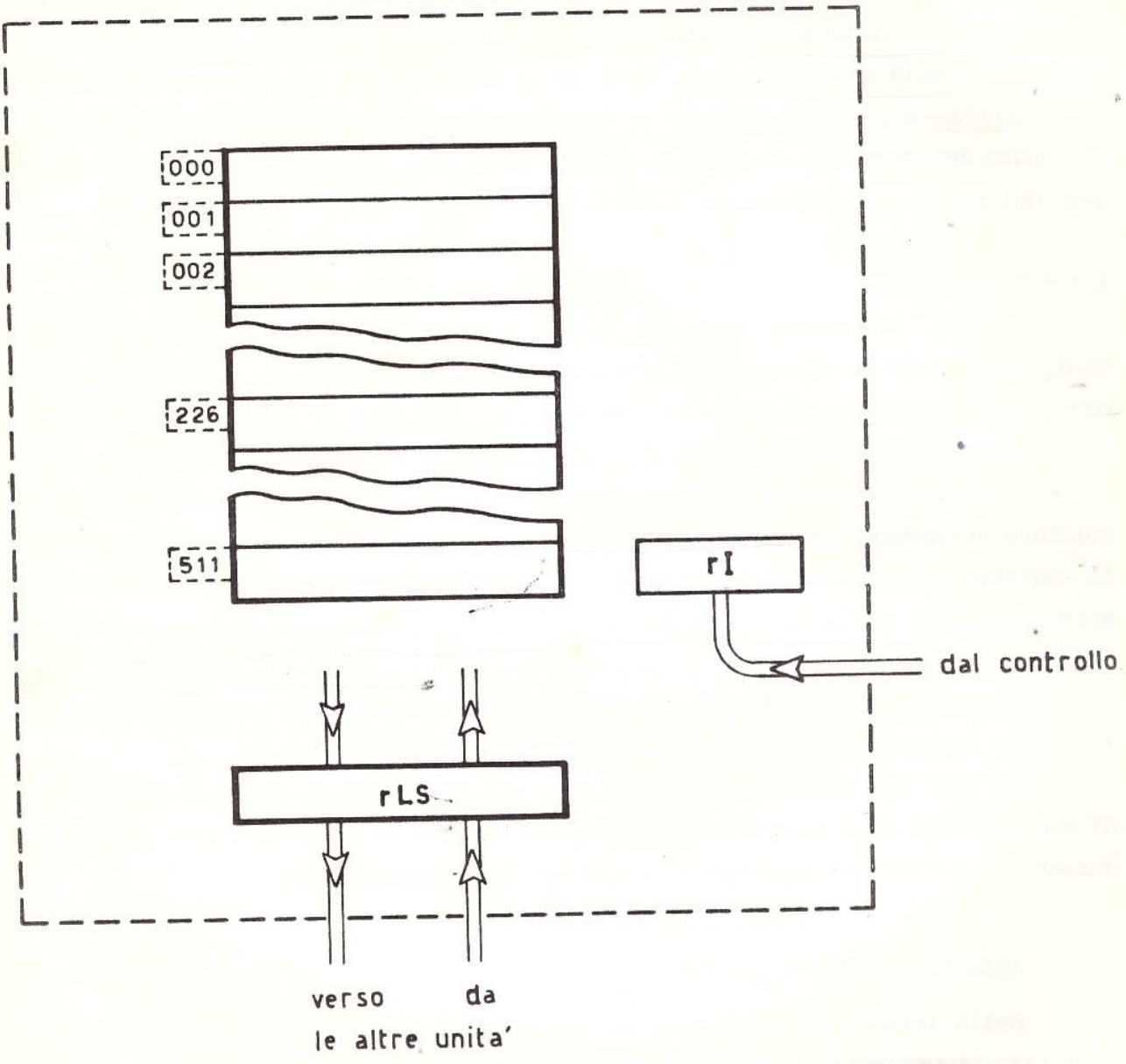


Fig. 3.7 - Organi architettonici dell'unità di memoria.

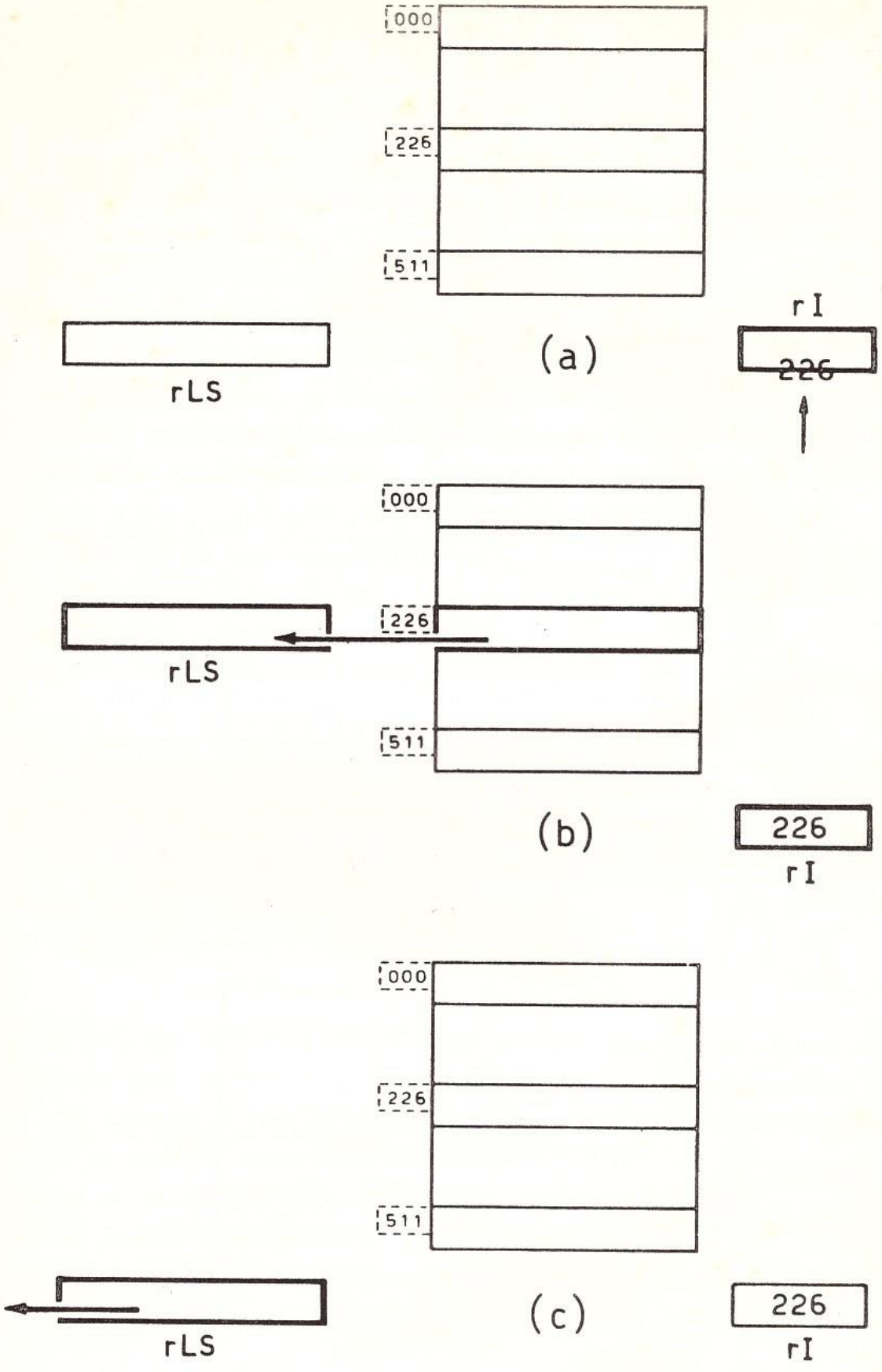


Fig. 3.8 - Lettura del contenuto di una cella.

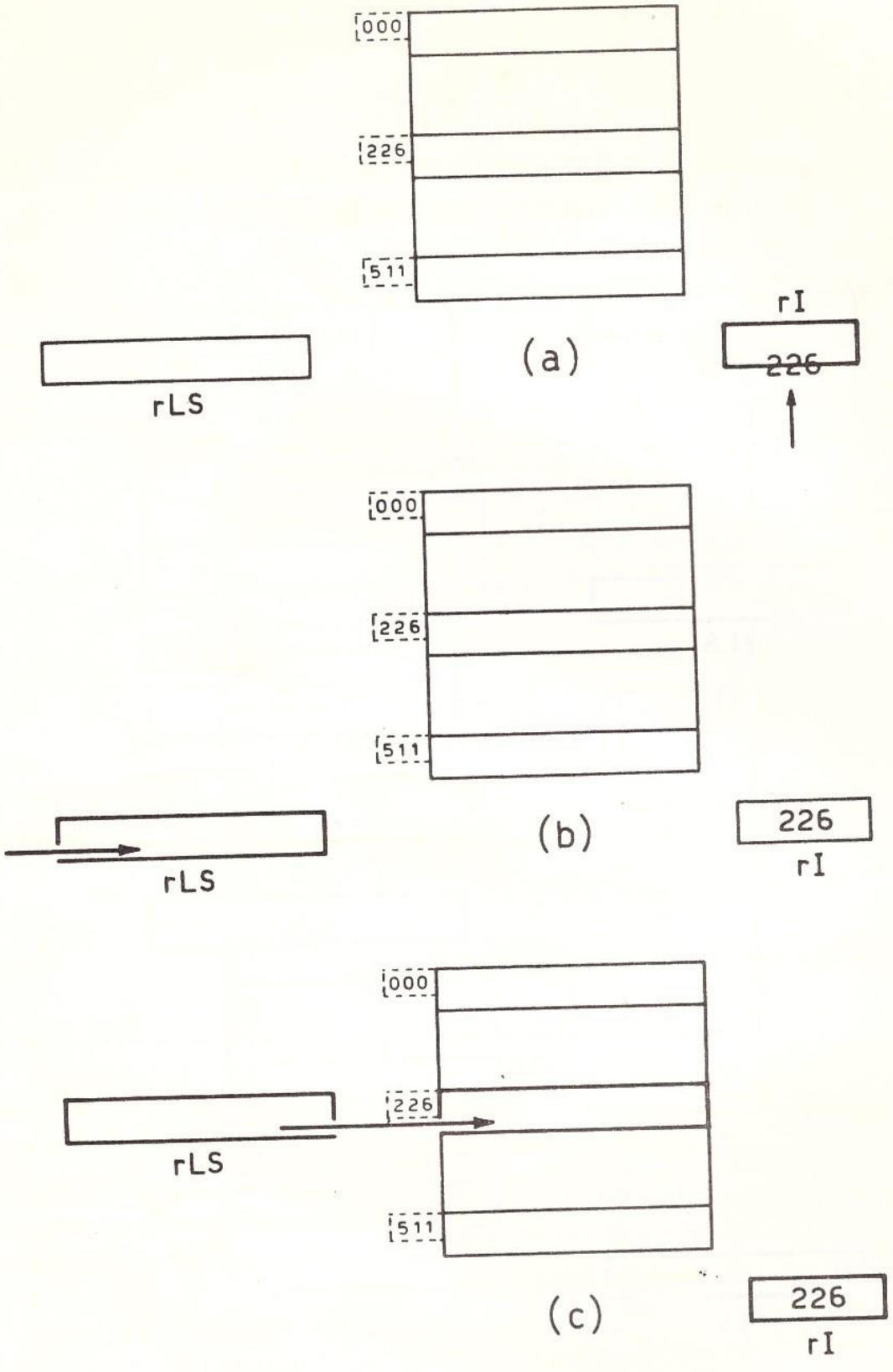


Fig. 3.9 - Scrittura di una parola in una cella.

sterno (fig. 3.9 (b)) in rLS (per esempio ricopiandola dal registro rA), ed infine il registro rLS viene ricopiato nella cella (fig. 3.9 (c)).

3.4 - Interpretazione del contenuto di una cella.

Il contenuto di una qualsiasi cella può venire interpretato dal calcolatore come una istruzione, oppure come un dato.

Facendo riferimento alla parola rappresentata nella fig. 3.6(b), se questa parola viene interpretata come istruzione, il suo significato è:

```
0 1 0 0 0 1 , 0 0 0 , 0 1 0 1 0 1 0 0 1
sottrarre ad          di indirizzo
rA il conte-          251(8)
nuto della
cella
```

Questa stessa parola può venire interpretata come numero binario:

```
0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1
+ 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1 (2) = 210251 (8)
```

oppure, secondo il codice della fig. 2. , come una sequenza di 3 caratteri alfanumerici:

```
0 1 0 0 0 1 | 0 0 0 0 1 0 | 1 0 1 0 0 1
      A           2             R
```

Si noti che la parola non contiene nessuna speciale etichetta che la contraddistingue come istruzione, o come numero binario, o come sequenza di 3 caratteri (*). Se durante la fase di interpretazione, la parola viene estratta dalla cella 226 ed inviata nella unità di controllo, essa viene interpretata come una istruzione; se invece, per esempio, il calcolatore obbedisce ad una istruzione che dice di sommare al registro rA il contenuto della cella 226, la parola che vi è contenuta viene interpretata coma numero binario; infine, se il calcolatore

(*) Tuttavia (vedi fig.2) non tutte le configurazioni di 6 bit corrispondono ad un carattere alfanumerico.

obbedisce ad una istruzione che dice di stampare il contenuto della cella 226, sulla unità esterna compaiono i 3 caratteri A2R.

Quindi, l'interpretazione delle parole scritte nella memoria dipende esclusivamente dal programma. Le istruzioni del programma vengono obbedite una per una; la successione delle istruzioni eseguite rispetta regole precise: dopo l'esecuzione della istruzione nella cella di indirizzo i, viene eseguita l'istruzione nella cella di indirizzo i + 1. Questa rigida sequenzialità viene, in certe condizioni, rotta dalle cosiddette istruzioni di salto, che designano l'indirizzo della istruzione successiva.

Concludiamo questo paragrafo osservando che, nel CANE, i numeri negativi sono rappresentati in complemento a 2; perciò, la configurazione:

1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 0

rappresenta il numero:

$-\,1\,0\,1\,0\,0\,1\,0\,1\,1\,1\,1\,0\,0\,1\,1\,1\,0_{(2)}$

Quindi, in una parola del CANE è possibile rappresentare un intero I appartenente all'intervallo:

$$-(2^{17}-1) \leq I \leq 2^{17}-1$$

3.5 - L'unità di elaborazione

L'unità di elaborazione, come abbiamo visto nel paragrafo 3.1, contiene alcuni registri, e gli organi aritmetico-logici, cioè quelle "macchine" che vengono impiegate per eseguire le operazioni fondamentali (somma, sottrazione,.....).

La fig. 3.10 mostra una schematizzazione della unità di elaborazione: vi sono due registri rA ed rB lunghi 18 bit (rispettivamente primo e secondo accumulatore), sette registri rX1,rX2,.....rX7 lunghi 9 bit (registri indice) e tre indicatori (registri lunghi 1 bit) iSP, iCF1 e iCF2 (rispettivamente: indicatore di supero, e primo e secondo indicatore di confronto).

La funzione degli accumulatori è quella di fornire un operando nell'esecuzione delle istruzioni di elaborazione (per esempio, il contenuto di rA, o di rB viene sommato al contenuto di una cella di memoria) e di registrare il risultato dell'elaborazione (per esempio, nelle istruzioni di somma, il con

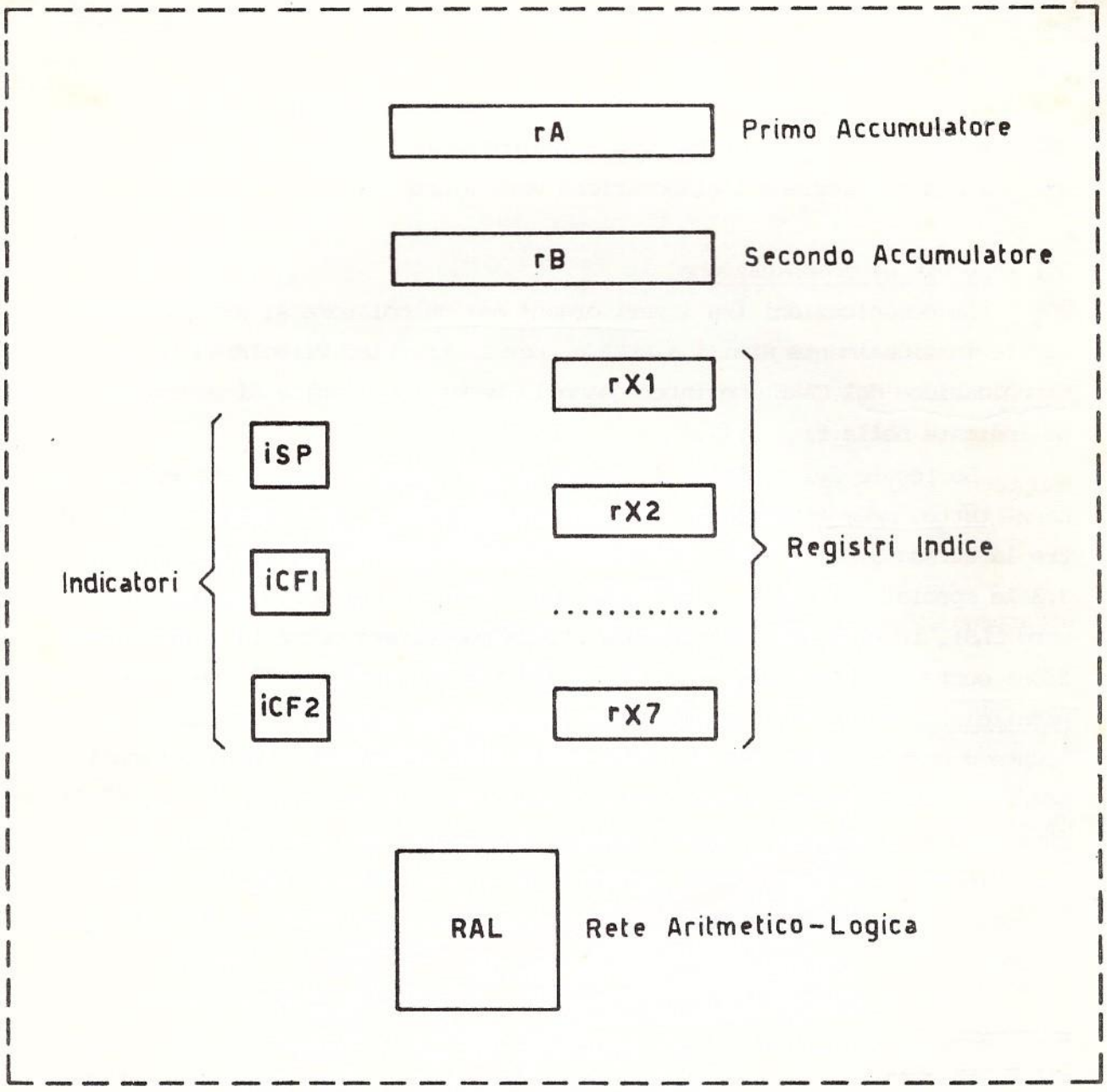


Fig. 3.10 - Registri ed indicatori dell'unità di elaborazione.

tenuto di rA, o di rB, viene sommato al contenuto di una cella di memoria, e memorizzato in rA, o in rB). Esamineremo nel seguito le funzioni dei registri indice e degli indicatori.

Le "macchine" che esguono le operazioni fondamentali, sono contenute nella scatola RAL (rete aritmetico - logica) della fig. 3.10: è appunto in questa scatola che avviene l'elaborazione vera e propria.

3.6 - Le vie di comunicazione

Le comunicazioni fra i vari organi del calcolatore si svolgono attraverso vie funzionalmente simili a strade per il traffico veicolare; le vie di comunicazione del CANE che interessano la memoria e l'unità di elaborazione sono indicate nella fig. 3.11.

Le regole del traffico su questo sistema viario sono molto semplici. Innanzi tutto, premettiamo che le strade 1,2,3,4,6 e 7 sono larghe 18 bit, mentre la strada 5 è larga 9 bit soltanto (abbiamo già considerato nel paragrafo 3.3 le speciali vie di comunicazione L ed S fra le celle di memoria ed il registro rLS). In un certo istante, una strada può essere occupata dalle informazioni contenute in un registro, che viaggiano affiancate sulla sede stradale (in parallelo) verso la loro destinazione; fintantochè queste informazioni non sono giunte a destinazione, sulla strada non possono viaggiare altri contenuti. Il traffico, naturalmente, viene regolato dall'unità di controllo; la fig. 3.12 mostra la situazione dei blocchi stradali durante l'esecuzione di una situazione che ricopia il contenuto del registro rA in una cella di memoria. Si noti che in questo caso l'informazione transita invariata attraverso la rete aritmetico-logica.

La rete aritmetico-logica è l'unico nodo del sistema viario nel quale sono permesse collisioni di due correnti di traffico; la parola che emerge alla uscita della RAL è una semplice funzione delle parole entranti: per esempio, la somma delle due parole entranti, considerate come rappresentazioni di numeri binari. La fig. 3.13 mostra appunto le vie di comunicazione interessate durante l'esecuzione di una istruzione di somma del contenuto di una cella di memoria al contenuto di rA; il contenuto della cella viene dapprima letto in rLS: successivamente, i contenuti di rA ed rLS vengono inviati rispettivamente

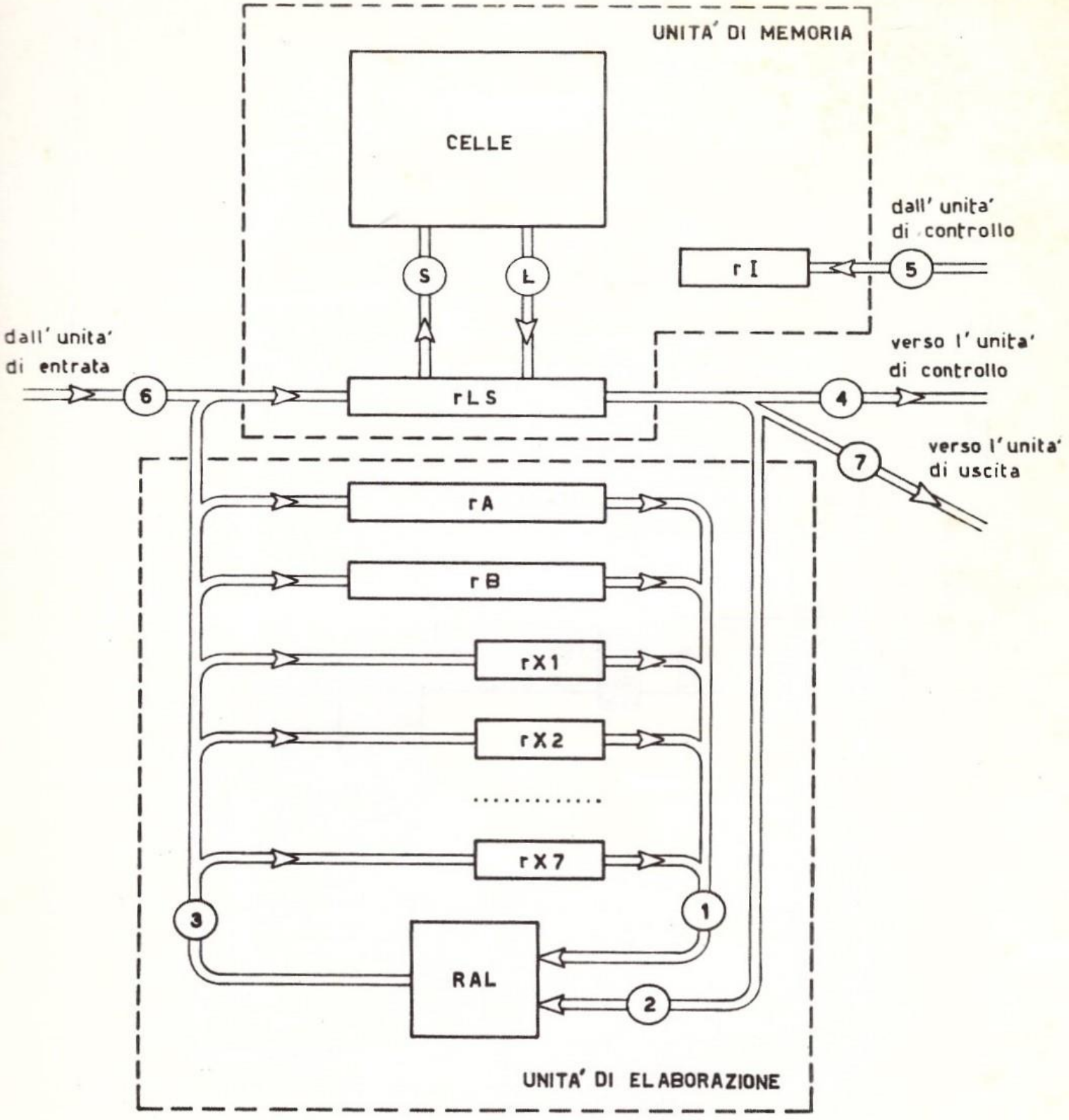


Fig. 3.11 - Alcune delle vie di comunicazione.

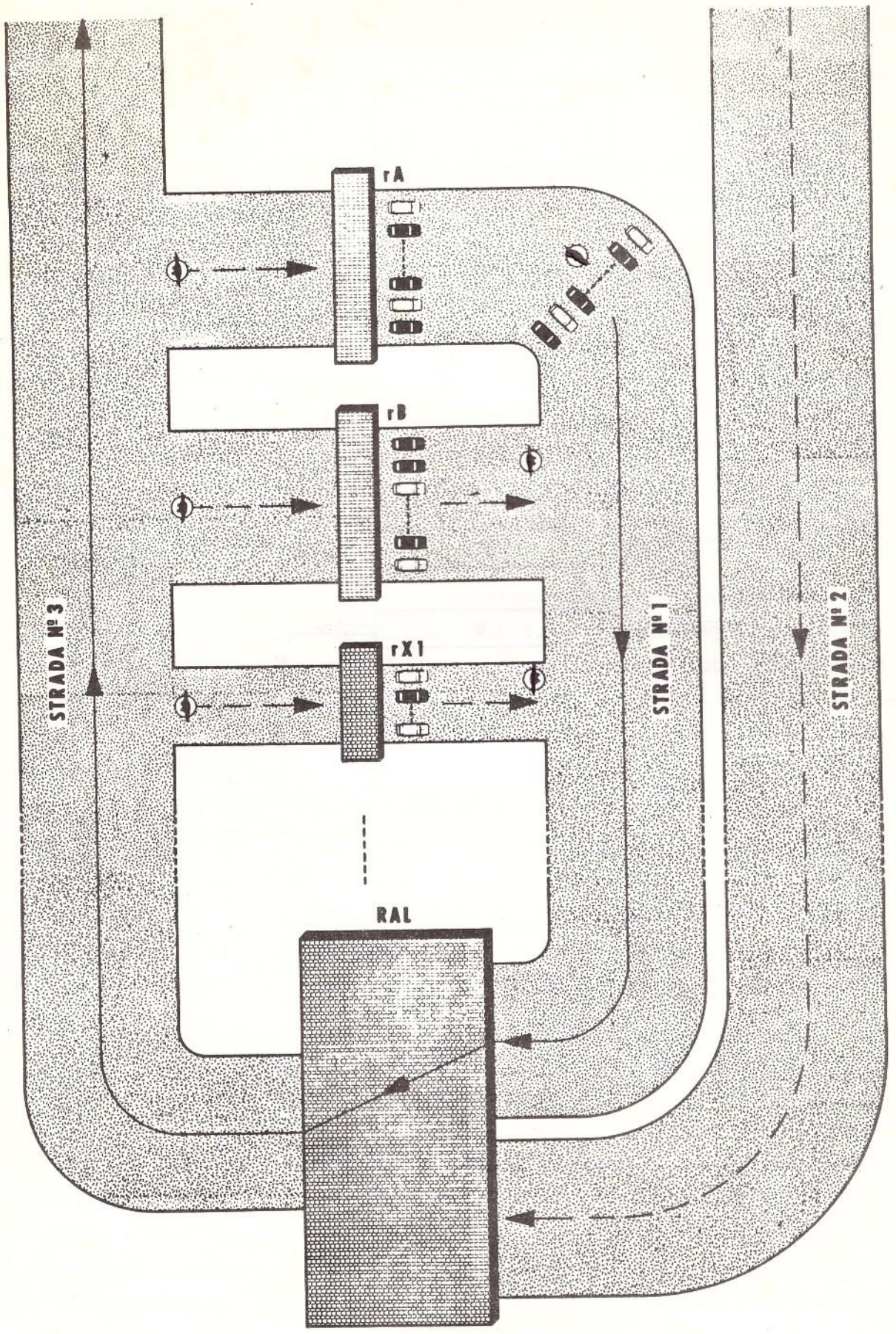


Fig. 3.12 - Il contenuto del registro rA viene ricopiato in una cella di memoria.

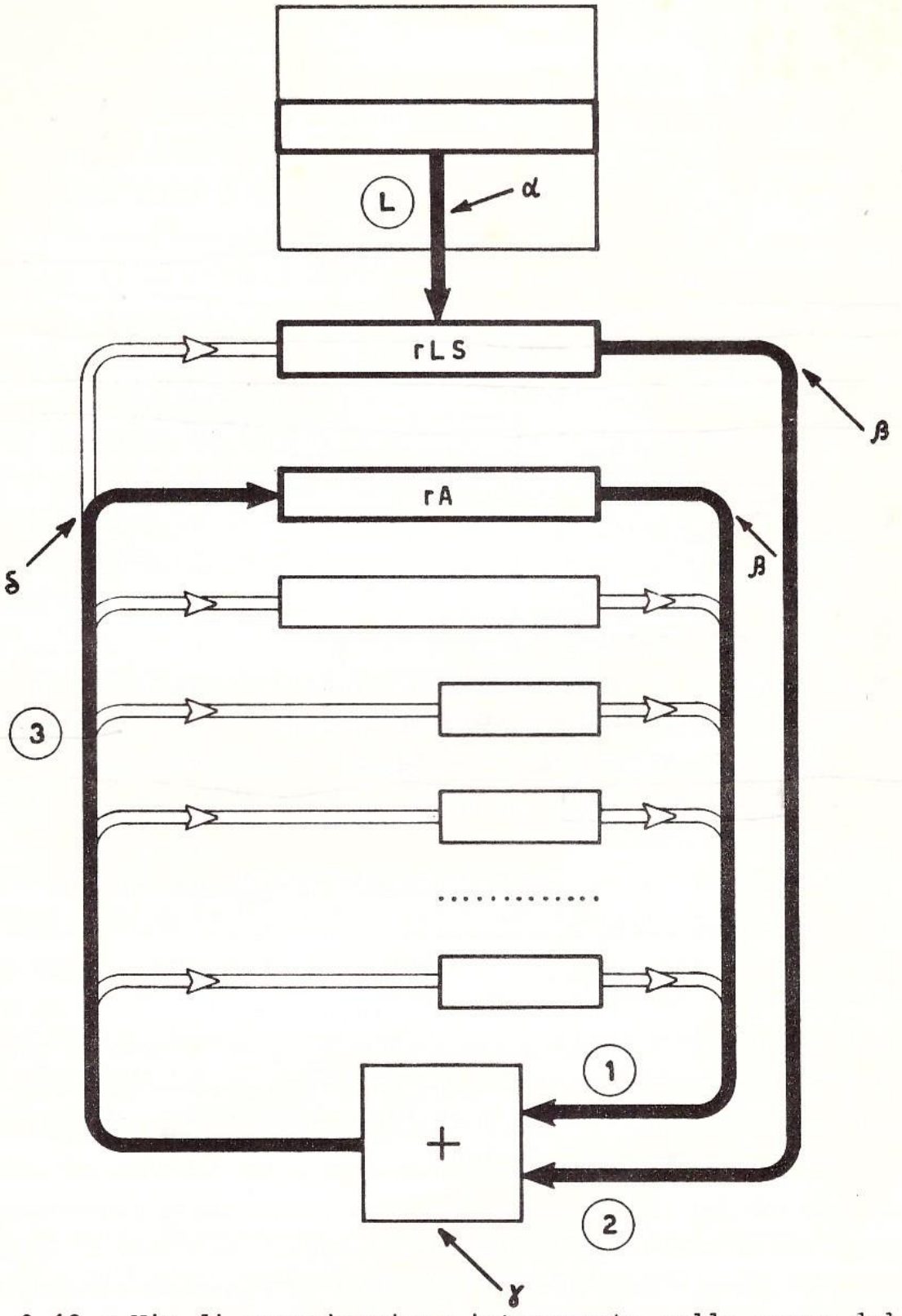


Fig. 3.13 - Vie di comunicazione interessate nella somma del contenuto di una cella al contenuto di rA.

sulle strade 1 e 2, alla rete aritmetico-logica; la parola emergente dalla rete, somma delle parole entranti, viene inviata per la strada 3 nel registro rA, cancellandovi la parola che vi era contenuta. Nella fig. 3.13, le lettere $\alpha, \beta, \gamma, \delta$ indicano i segnali inviati dalla unità di controllo durante l'esecuzione della istruzione: il segnale α provoca la lettura della cella di memoria il cui indirizzo è contenuto in rI, β abilita i contenuti di rLS ed rA al transito sulle strade che portano alla rete aritmetico-logico, γ condiziona la rete aritmetico-logica ad eseguire la somma; ed infine δ mette in comunicazione la strada 3 con il registro rA.

E' molto comodo indicare gli eventi elementari che avvengono nel calcolatore mediante una notazione simbolica, perchè ciò permette una certa economia di espressione. Il contenuto di un registro o di una cella di memoria verrà indicato ponendo fra parentesi tonde il nome del registro o l'indirizzo della cella di memoria: così, (rA) ed (h) indicano rispettivamente il contenuto di rA, ed il contenuto della cella di memoria di indirizzo h. Due coppie di parentesi verranno invece usate per indicare il "contenuto della cella di memoria il cui indirizzo è contenuto nel registro rI". Il trasferimento di informazione fra due registri rP ed rQ verrà indicato con la notazione:

(1) $(rQ) \leftarrow (rP)$

di interpretazione ovvia.

La notazione (1) può essere estesa: per esempio, il trasferimento nel registro rR della somma dei contenuti dei registri rP ed rQ verrà rappresentato da:

(2) $(rR) \leftarrow (rP) + (rQ)$

Notazioni analoghe alla (2) verranno impiegate qualunque sia l'operazione che avviene fra i contenuti dei due registri; se l'operazione fra i contenuti dei due registri viene indicata con il simbolo $*$, scriveremo:

(3) $(rR) \leftarrow (rP) * (rQ)$

Gli eventi durante l'esecuzione di una istruzione di somma ad rA (fase esecutiva della istruzione) descritti graficamente nella fig. 3.13, possono essere scritti come segue:

(4)
$$\begin{aligned} (rLS) &\leftarrow ((rI)) \\ (rA) &\leftarrow (rA) + (rLS) \end{aligned}$$

E' talvolta interessante (o addirittura necessario, se la sequenza di eventi è molto complessa) notare vicino ad ogni evento i segnali di controllo che ne regolano l'esecuzione. Nel caso esaminato, le (4) vengono completate così:

$$(5)\quad \begin{array}{ll} \alpha: & (rLS) \leftarrow ((rI)) \\ \beta, \gamma, \delta: & (rA) \leftarrow (rA) + (rLS) \end{array}$$

3.7 - Struttura delle istruzioni

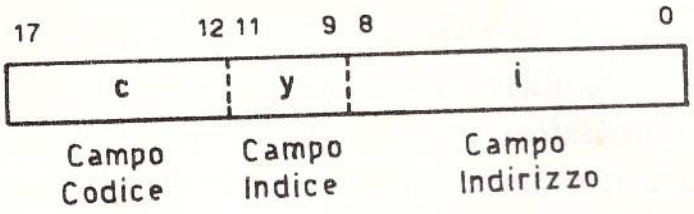
Le istruzioni del CANE hanno la struttura indicata nella fig. 3.14(a); i primi 6 bit da destra (posizioni 17,16,.....12) indicano il tipo di istruzione; i 3 bit successivi (posizioni 11,10,9) rappresentano un intero compreso fra 0 e 7 di cui vedremo nel seguito il significato; infine gli ultimi 9 bit (posizioni 8,7,....0) rappresentano un intero compreso fra 0 e 511 che nella maggior parte delle istruzioni sta ad indicare l'indirizzo di una cella. Le tre parti dell'istruzione si chiamano rispettivamente campo codice, campo indice e campo indirizzo: i contenuti di queste tre parti verranno nel seguito indicati rispettivamente con le lettere c, j, ed i, come mostra la fig. 3.14. In fig. 3.14(b) è rappresentata una parola; se questa parola viene interpretata come una istruzione:

c = 010001, j = 000, i = 010101001

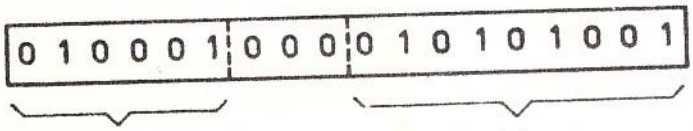
Abbiamo visto nel par.3.4 il significato di questa istruzione: possiamo ora dire che la configurazione c = 010001 è il codice della istruzione di "sottrazione ad rA".

Per il momento, non ci interesseremo del contenuto del campo indice; più precisamente, considereremo solo l'esecuzione di istruzioni per le quali il campo indice contiene la configurazione j = 000. Per maggiore semplicità e chiarezza, impiegheremo nel seguito molto frequentemente la notazione ottale per rappresentare le parole, gli indirizzi, ed i contenuti dei tre campi delle istruzioni: la base 8 verrà tuttavia indicata solamente dove la sua omissione possa ingenerare malintesi. In notazione ottale la parola in fig. 3.14(b) è quindi rappresentata come segue:

2 1 0 2 5 1



(a)



Codice della istruzione:
SOTTRAZIONE AD rA

Indirizzo :
$010101001_{(2)} = 251_{(8)}$

(b)

Fig. 3.14 - Struttura delle istruzioni.

Quindi:

$$c = 21, \quad j = 0, \quad i = 251$$

3.8 - Alcune istruzioni

Consideriamo la fig. 3.15 (a): essa mostra i contenuti delle celle di indirizzo 112, 113, 114 e 325, 326, 327. Stiamo osservando la memoria del CANE mentre il calcolatore si appresta ad eseguire la sequenza di 3 istruzioni contenute nelle celle 112, 113 e 114; l'effetto dell'esecuzione di queste tre istruzioni è quello di memorizzare nella cella 327 la somma dei numeri contenuti nelle celle 325 e 326:

$$(327) \leftarrow (325) + (326)$$

Ricordando che il nostro calcolatore impiega la rappresentazione in complemento a 2:

$$(325) = 000322 = +\ 322_{(8)}$$

$$(326) = 777256 = -\ 522_{(8)}$$

Dopo aver eseguito le tre istruzioni, perciò, la cella 327 conterrà:

$$(327) = 777600 = -\ 200_{(8)}$$

Ma analizziamo in dettaglio le tre istruzioni. La prima (quella contenuta nella cella 112) è scritta:

01 0 325

Le istruzioni che hanno codice c =01, come quella contenuta nella cella 112 del nostro esempio, sono istruzioni di trasferimento nel registro rA del contenuto della cella individuata dal campo indirizzo. Nel nostro caso, l'effetto dell'esecuzione dell'istruzione è:

$$(rA) \leftarrow (325),$$

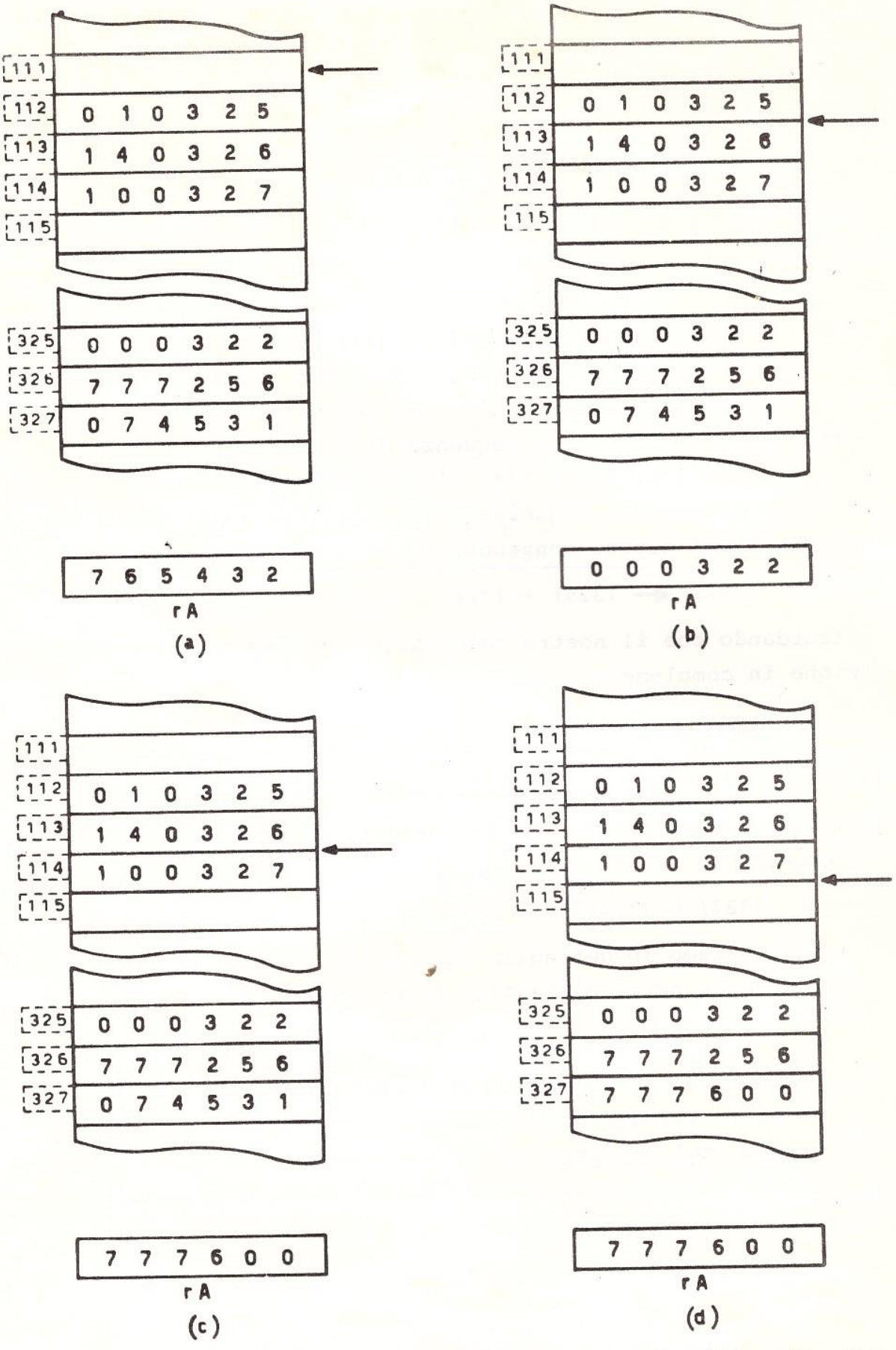


Fig. 3.15 - La sequenza di tre istruzioni nelle celle 112, 113, 114 esegue la somma dei contenuti delle celle 325 e 326, e memorizza il risultato nella cella 327.

ed in generale:

(rA) ← (i).

Nella fig. 3.15(b), vediamo la situazione dopo l'esecuzione della istruzione nella cella 112. L'istruzione che viene eseguita successivamente è contenuta nella cella 113; il contenuto del campo codice, c = 14, indica una istruzione di somma la contenuto di rA della cella individuata dal campo indirizzo. Come mostra la fig. 3.15(c), l'effetto dell'esecuzione è in questo caso:

(rA) ← (rA) + (326),

e perciò il primo accumulatore conterrà:

$(rA) = 777600 = -200_{(8)}$

In generale, l'operazione eseguita dall'istruzione di somma ad rA è:

(rA) ← (rA) + (i)

Infine, l'istruzione nella cella 114 (c = 10) è una istruzione di memorizzazione del contenuto di rA nella cella specificata dal campo indirizzo:

(i) ← (rA)

e nel nostro caso:

(327) ← (rA)

Perciò, dopo l'esecuzione di questa istruzione, il contenuto di rA è stato ricopiato nella cella 327 (vedi la fig. 3.15(d)). Un esame della fig. 3.15 mostra che sia l'istruzione di trasferimento, che quelle di somma e di memorizzazione hanno lasciato invariati i contenuti dei registri sorgente. Che cosa accade dopo che il calcolatore ha terminato l'esecuzione dell'istruzione nella cella 114? Il calcolatore esegue l'istru-

nella cella 115....., e poi? Per rispondere alla domanda, dovremmo sapere qual'è l'istruzione contenuta in 115: infatti, non tutte le istruzioni si comportano come i 3 tipi finora esaminati per quanto riguarda la scelta dell'istruzione successiva. Abbiamo visto che dopo avere eseguito una istruzione di trasferimento, somma o memorizzazione, il calcolatore esegue l'istruzione nella cella seguente (si usa anche dire che "il controllo passa all'istruzione nella cella seguente"). L'istruzione di salto incondizionato, invece, indica al controllo l'indirizzo dell'istruzione successiva; il codice di questa istruzione è c = 40. Per esempio, l'effetto dell'esecuzione dell'istruzione:

```
40  0  222
```

è quello di far saltare il calcolatore all'esecuzione dell'istruzione nella cella 222, e ciò indipendentemente dall'indirizzo della cella nella quale l'istruzione di salto è collocata. L'effetto di altre istruzioni di salto, nel CANE, dipende dal verificarsi di determinate condizioni. Per esempio, l'istruzione di salto se(rA) zero (che ha codice c =50) fa passare il controllo all'istruzione specificata nel campo indirizzo se il primo accumulatore contiene lo zero (se, cioè, i 18 bit di rA sono tutti 0); se, invece, (rA) $\neq$ 000000, viene eseguita l'istruzione nella cella seguente. Così, se l'istruzione nella cella 115 dell'esempio di fig. 3.15 fosse un "salto se (rA) zero":

```
111   ........
112   01 0 325
113   14 0 326
114   10 0 327
115   50 0 444
116   ........
```

il controllo, dopo l'istruzione in 114, passerebbe all'istruzione in 116, perchè, come mostra la fig. 3.15, il registro rA non contiene zero.

Nel CANE, vi sono numerose altre istruzioni di salto condizionato, oltre a quella che abbiamo appena esaminato.

Si noti che le istruzioni descritte funzionano allo stesso modo indipendentemente dall'indirizzo a cui sono collocate: così, se le tre istruzioni della fig. 3.15 fossero memorizzate nelle tre celle di indirizzo 501, 502 e 503, l'effetto della loro esecuzione sarebbe il medesimo.

Per brevità, si usa indicare il tipo di istruzione mediante una sigla mnemonica: per esempio, TRA per l'istruzione di "trasferimento in rA", ADA per l'istruzione di "somma al contenuto di rA", e così via. Le sigle impiegate sono riportate nella tabella della fig. 3.16. La tabella elenca alcune istruzioni che non abbiamo ancora discusso. L'istruzione di sottrazione al contenuto di rA sottrae il contenuto della cella specificata dall'indirizzo al contenuto del primo accumulatore:

(rA) ← (rA) - (i)

Spesso, nel corso di un programma, occorre servirsi di una costante; molto frequentemente, per esempio, si vuole incrementare o decrementare di una unità il contenuto di una cella di memoria: per eseguire questa operazione, se la costante 1 è memorizzata in una delle celle di memoria, ci si può servire delle istruzioni ADA e SOA che abbiamo già commentato. Naturalmente, è necessario in questo caso introdurre la costante nella cella di memoria che la ospita, per esempio leggendola in memoria assieme ai dati iniziali. Le tre istruzioni di trasferimento diretto in rA, somma diretta al contenuto di rA, sottrazione diretta al contenuto di rA permettono invece di manipolare costanti il cui valore non ecceda

istruzione	c	sigla	effetto
trasferimento in rA	01	TRA	(rA) ← (i)
somma ad rA	14	ADA	(rA) ← (rA) + (i)
sottrazione ad rA	15	SOA	(rA) ← (rA) - (i)
memorizzazione di rA	10	MEA	(i) ← (rA)
trasferimento diretto in rA	03	TDA	(rA) ← i
somma diretta ad rA	16	ADDA	(rA) ← (rA) + i
sottrazione diretta ad rA	17	SODA	(rA) ← (rA) - i
salto incondizionato	40	SLT	istruzione seguente nella cella i
salto se (rA) zero	50	SAZ	se (rA) zero, istruzione seguente nella cella i

Fig. 3.16 - Alcune istruzione del CANE

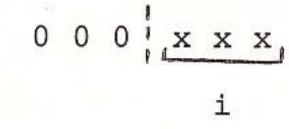
$777_{(8)}$: il campo indirizzo di queste istruzioni, infatti, non ospita l'indirizzo dell'operando, ma l'operando stesso. Per esempio, l'esecuzione delle tre istruzioni:

```
........
01 0 555
16 0 002
10 0 555
........
```

ha come effetto quello di incrementare di 2 il contenuto della

cella 555.

L'operando diretto è un numero binario positivo di 9 bit; la istruzione TDA (trasferimento diretto in rA) azzera i 9 bit più significativi del primo accumulatore (posizioni 17, 16,....., 9) e trasferisce il contenuto del suo campo indirizzo nei 9 bit meno significativi (posizioni 8, 7,....., 0). Quindi, è come se il contenuto del campo indirizzo venisse completato a sinistra con 9 cifre binarie, tutte uguali a 0:



Analogamente operano le due istruzioni ADDA (somma diretta al contenuto di rA) e SODA (sottrazione diretta al contenuto di rA): l'operando diretto viene completato a sinistra con nove 0, e sommato, o sottratto, al primo accumulatore. Come esempio, consideriamo il seguente segmento di programma:

```
........
03 0 001        TDA
10 0 500        MEA
16 0 002        ADDA
10 0 501        MEA
16 0 002        ADDA
10 0 502        MEA
16 0 002        ADDA
10 0 503        MEA
........
```

Le istruzioni precedenti memorizzano nelle celle 500, 501, 502 e 503 i primi quattro interi dispari.

3.9 - Un programma per calcolare la somma di 100 numeri

Supponiamo di dover redigere un programma per calcolare la somma di cento numeri a_1, a_2,....a_{100}:

$$s = \sum_{j=1}^{100} a_j$$

Si vedrà più oltre che il CANE possiede, come unità di entrata, un lettore di schede, e, come unità di uscita, una stampante. La prima operazione che il programma deve compiere è la lettura delle cento quantità a_1,....., a_{100}, perforate su schede, in cento celle di memoria; l'ultima operazione è la scrittura del risultato s sulla stampante del calcolatore. Non ci occuperemo qui delle operazioni di lettura e scrittura, perchè non abbiamo ancora preso in esame le istruzioni di entrata - uscita; supporremo che i nostri cento numeri siano già stati memorizzati nelle cento celle di indirizzi 300, 301,....., 442 e 443 e che si voglia il risultato s nella cella 444.

Senza grandi sforzi di immaginazione, possiamo scrivere il programma per il calcolo di s nel modo seguente:

```
........    (termina qui la lettura)
01 0 300    TRA di a1
14 0 301
14 0 302
14 0 303    99 istruzioni ADA
........
14 0 443
10 0 444    MEA di s
........    (inizia qui la scrittura)
```

ed utilizzare una zona qualsiasi di memoria per memorizzarlo. Il nostro CANE può però essere istruito in un modo assai più intelligente; consideriamo infatti l'algoritmo della fig.3.17, che ogni lettore dovrebbe a questo punto essere in grado di interpretare come una ricetta adeguata per il calcolo di s. Il programma che traduce l'algoritmo della fig. 3.17 è mostrato nella fig. 3.18, a destra.Questa figura mostra anche come sono state utilizzate le celle di memoria: il segmento di programma che calcola s è memorizzato nelle 15 celle 100, 101,...., 117, preceduto e seguito dai segmenti di programma che eseguono rispettivamente l'operazione di lettura e l'operazione di scrittura. I dati a_1, a_2, ..., a_{100} sono nelle celle 300, 301,..., 443, il risultato s nella cella 444, che, nel corso dell'esecuzione, conterrà la somma parziale; infine, la funzione della cella 277, usata dal programma come contatore, diverrà chiara fra breve.

Il programma è stato ripetuto nella fig. 3.19, sostituendo, al posto delle due cifre ottali del campo codice, le sigle della fig. 3.17; inoltre, è stata omessa la cifra ottale del campo indice, che abbiamo visto, per ora, essere sempre 0. La sequenza di istruzioni risulterà così più facilmente leggibile: il lettore ricordi, però, che il programma è contenuto nella memoria nelle configurazioni binarie riassunte dalle cifre ottali della fig. 3.18.

Vediamo ora di leggere il nostro programma. Le prime due istruzioni azzerano la cella 444, che conterrà via via la somma parziale, e finalmente il risultato; le due istruzioni seguenti memorizzano il valore 1 nella cella contatore 277. Queste quattro istruzioni costituiscono la fase di inizializzazione (vedi Capitolo I), e verranno eseguite una volta sola nel corso del programma.

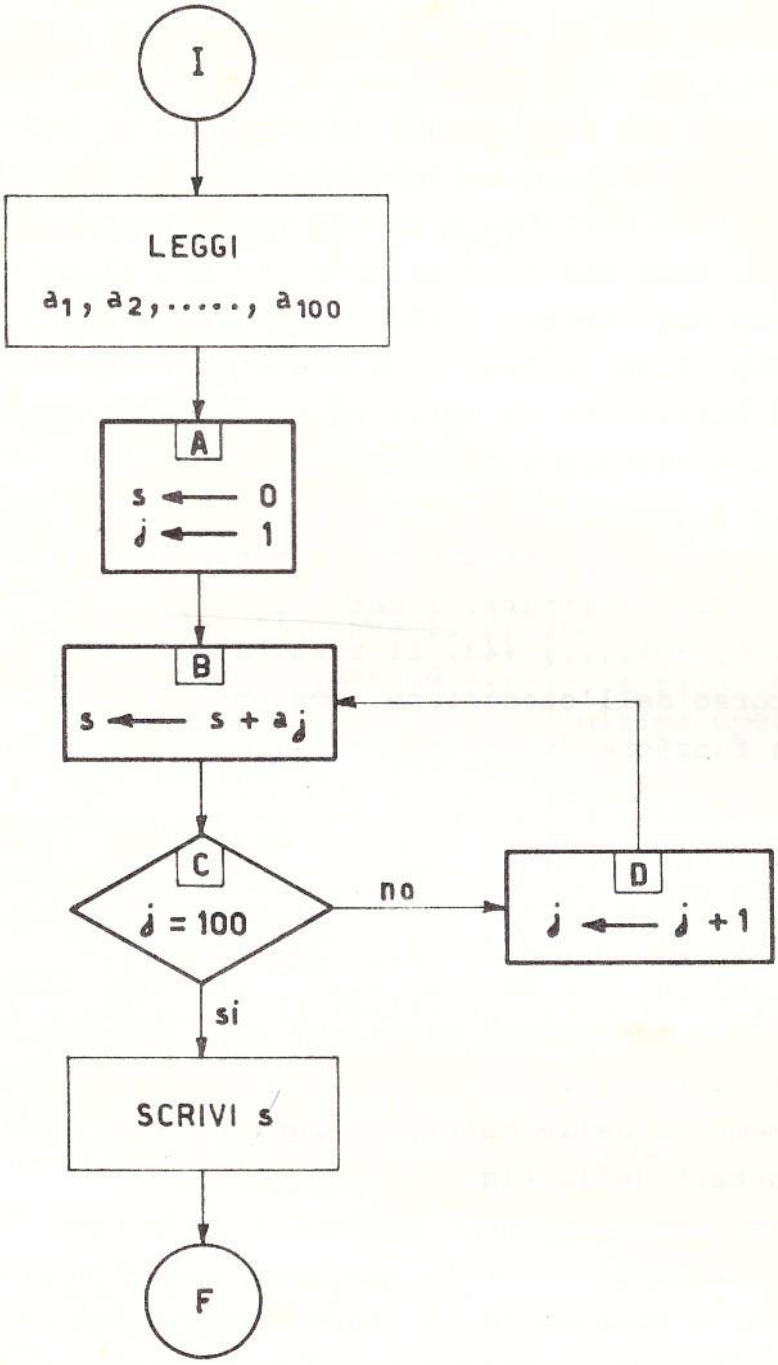


Fig. 3.17 - Schema a blocchi per la somma di 100 numeri.

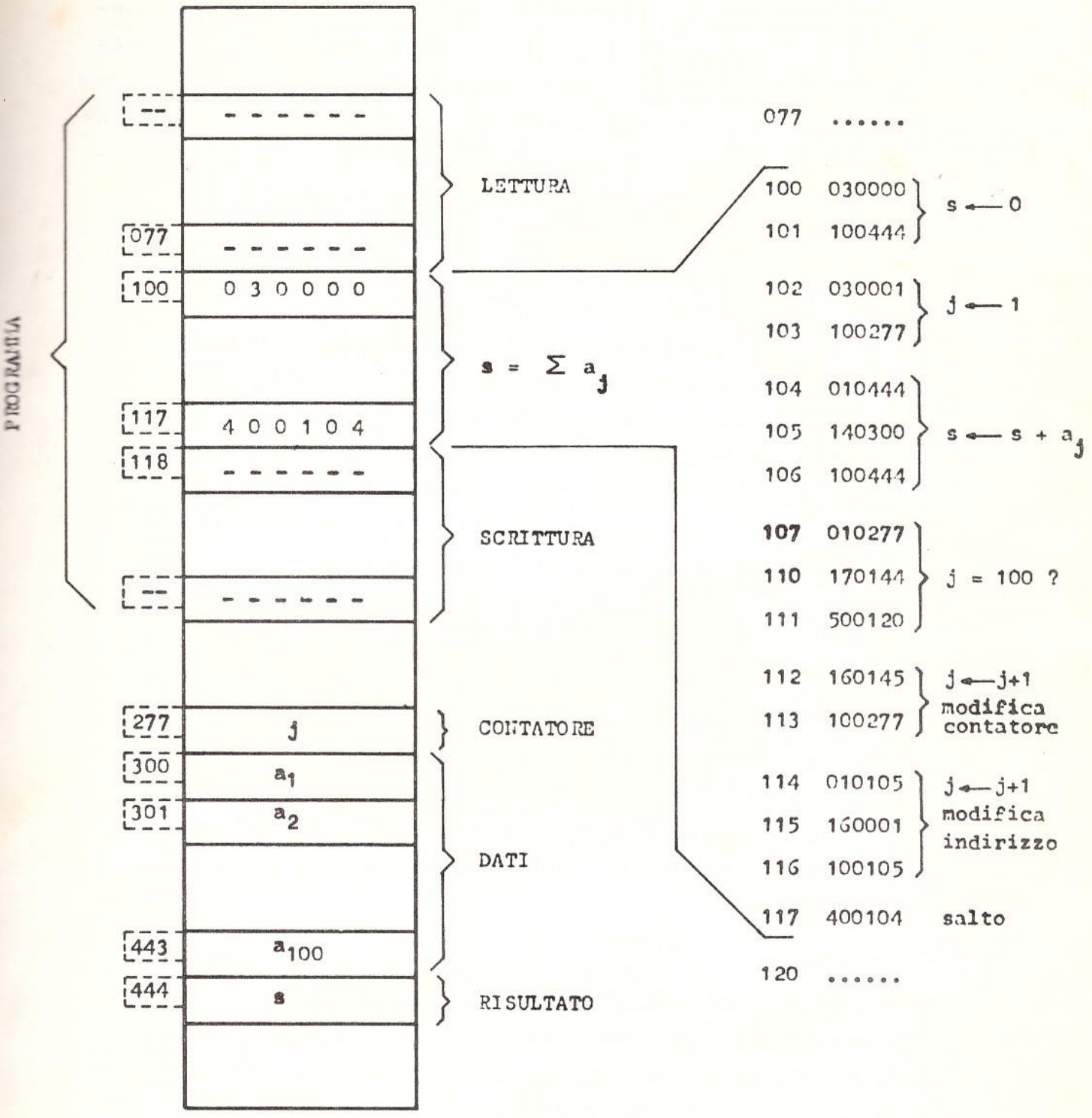


Fig. 3.18 - Somma di 100 numeri: utilizzazione della memoria e programma.

Con l'istruzione in 104, inizia la fase di elaborazione (blocco B dello schema in fig. 3.17): l'effetto della prima esecuzione delle istruzioni in 104, 105 e 106 è di memorizzare nella cella 444 la quantità a_1.

Il blocco C dello schema a blocchi (fase di test) corrisponde alle tre istruzioni in 107, 110 e 111: infatti, la cella contatore contiene il valore dell'indice j, che viene trasferito nel primo accumulatore, ed esaminato dall'istruzione SAZ in 111 dopo che l'istruzione SODA in 110 ha sottratto ad rA la costante $100_{(10)}$.
Quindi il salto SAZ viene eseguito (ed il controllo passa alla prima istruzione della fase di scrittura, nella cella 120) se j ha raggiunto il valore $100_{(10)}$. In caso contrario, si eseguono le istruzioni in 112 e seguenti, che corrispondono al blocco D (fase di modifica). Le prime due (in 112 e 113) incrementano l'indice j di una unità; lo scopo delle tre istruzioni successive dovrebbe risultare chiaro da un attento esame....che però mette in luce una manipolazione sospetta del contenuto della cella 105. Abbiamo visto come questa cella contenga l'istruzione:

14 0 300

che somma al contenuto di rA il contenuto della cella 300. Poichè vogliamo tornare ad eseguire le istruzioni corrispondenti al blocco B, è necessario modificare l'indirizzo di questa istruzione incrementandolo di una unità, acciocchè la seconda esecuzione abbia l'effetto:

$$s \leftarrow s + a_2$$

Non vi è nulla di sconveniente nel modo con cui abbiamo impiegato le tre istruzioni 114, 115 e 116: infatti, la prima di queste ricopia la parola contenuta in 105 nel registro rA, la seconda incrementa di una unità tale registro, e la terza ricopia il

indirizzo istruzione	istruzione	effetto
076	.....	
077	.....	(termina qui la lettura)
100	TDA 000	(rA) ← 000000
101	MEA 444	s ← (rA)
102	TDA 001	(rA) ← 000001
103	MEA 277	j ← (rA)
104	TRA 444	(rA) ← s
105	ADA 300	(rA) ← s + aj
106	MEA 444	s ← (rA)
107	TRA 277	(rA) ← j
110	SODA 144	(rA) ← (rA) - 100_{10}
111	SAZ 120	se (rA) = 000000, salto a 120
112	ADDA 145	(rA) ← (rA) + 101_{10}
113	MEA 277	j ← (rA)
114	TRA 105	(rA) ← (105)
115	ADDA 001	(rA) ← (rA) + 000001
116	MEA 105	(105) ← (rA)
117	SLT 104	salto a 105
120	.....	(inizia qui la scrittura)

Fig. 3.19 - Programma per la somma di 100 numeri.

contenuto del registro in 105. Quando si esegue il salto incondizionato in 117, che fa passare il controllo alla istruzione 104, l'indirizzo della istruzione in 105 è quindi divenuto 301; dopo la seconda esecuzione delle istruzioni che fanno parte del ciclo tale indirizzo è divenuto 302,...., dopo la 99esima esecuzione ha raggiunto il valore 443.

Si noti che il funzionamento corretto del segmento di programma che abbiamo esaminato non è più indipendente dalla collocazione in memoria; non è più possibile, infatti, scegliere un'altra zona di memoria per collocare il programma senza eseguire una piccola modifica: infatti, il programma fa riferimento (istruzioni in 114 e 116) ad una cella che contiene una istruzione del programma stesso: l'indirizzo di tale cella deve venire opportunamente modificato se il programma viene collocato in una diversa zona di memoria.

3.10 - La struttura dei cicli.

La tecnica che abbiamo impiegato nell'esempio del paragrafo precedente per modificare l'indirizzo di una istruzione è assolutamente generale, e nei primi calcolatori elettronici era l'unica disponibile per programmare cicli come quello esaminato.

Il CANE, come tutti i moderni calcolatori, è invece equipaggiato di registri indice che, come vedremo sul capitolo IV, possono venire impiegati nella duplice funzione di contatori e di modificatori d'indirizzo.

Non è necessario disporre le tre fasi di elaborazione,test e modifica nell'ordine esemplificato nel programma della fig. 3.19; le tre fasi possono infatti susseguirsi in un ordine qualsiasi. E'possibile, per esempio, disporre le tre fasi nell'ordine: test, elaborazione, modifica; è naturalmente essen-

ziale accertare che, comunque, la fase di elaborazione venga attraversata il numero di volte desiderato. Nel nostro esempio, se le 5 istruzioni che esaminano e modificano il contenuto della cella contatore (istruzioni C e D1 nella fig. 3.19) vengono premesse alle istruzioni che costituiscono la fase di elaborazione (istruzioni B nella fig. 3.19), occorre cambiare il campo indirizzo della istruzione SODA: mentre tale campo contiene $144_{(8)} = 100_{(10)}$ nel programma della fig. 3.19, esso dovrà contenere $145_{(8)} = 101_{(10)}$, altrimenti la fase di elaborazione verrebbe eseguita solamente 99 volte.

Si noti che la fase di test richiede sempre un attento esame: essa costituisce infatti, non appena il ciclo sia di complessità modesta, una inesauribile sorgente di errori. Se il ciclo deve essere eseguito n volte, è molto facile che un errore di programmazione comporti che il ciclo venga eseguito: a) n - 1 volte, b) n + 1 volte, c) mai, d) infinite volte (cioè, la fase di test è tale per cui il programma non esce mai dal ciclo).

Consideriamo ora l'impiego del contatore nel programma precedente: il suo contenuto viene incrementato di una unità ad ogni esecuzione. E' possibile, invece, inizializzare il contatore al numero di volte che si vuole eseguire il ciclo, e decrementarlo ad ogni esecuzione. Questa variante è presentata nella fig. 3.20(a): si noti che la fase di inizializzazione carica il numero $144_{(8)} = 100_{(10)}$ nel contatore, e che le prime due istruzioni dal ciclo (nelle celle 104 e 105) esaminano il contenuto del contatore, mentre l'istruzione seguente (in 106) lo decrementa di una unità. E' possibile far precedere l'istruzione SODA al salto SAZ: lo scambio di queste due istruzioni comporta però che nel contatore deve venire inizialmente caricato il numero $145_{(8)} = 101_{(10)}$. Si noti che abbiamo accorciato

il programma di una istruzione, e che questa istruzione viene "risparmiata", come tempo di esecuzione, ogni volta che si attraversa il ciclo.
Questa variante non corrisponde più, se vogliamo essere precisi, allo schema a blocchi della fig. 3.17: abbiamo infatti introdotto un nuovo indice, che viene inizializzato al valore $100_{(10)}$; tuttavia, è opportuno evitare una eccessiva pignoleria, ed accontentarci nel seguito di una corrispondenza in senso lato fra schema a blocchi e programma.

Una seconda variante è mostrata nella fig. 3.20(b): questa fa uso di una nuova istruzione, che compare nella cella 114: l'istruzione si chiama salto se (rA) positivo, il suo codice è c= , e la sigla che la identifica SAP. Il suo funzionamento dovrebbe essere, a questo punto, del tutto ovvio: il salto viene obbedito (e viene quindi eseguita l'istruzione specificata dal campo indirizzo) se il contenuto di rA è in numero positivo. Si noti che il campo indirizzo della istruzione in 102 (inizializzazione del contatore) è vuoto: abbiamo lasciato la sua determinazione corretta come esercizio per il lettore.

Concluderemo consigliando una semplice (e simpatica) regoletta per determinare il corretto valore cui inizializzare i contatori nei cicli. (il lettore potrà immediatamente applicarla all'esempio della fig. 3.20(b)). Si supponga che il ciclo debba venire attraversato una volta sola, e si analizzi il programma: che cosa deve contenere inizialmente il contatore? E' facile poi determinare il valore iniziale corretto nel caso che il ciclo debba venire eseguito n volte.

```
077     ........
100     TDA  000
101     MEA  444
102     TDA  144
103     MEA  277
104     TRA  277
105     SAZ  117
106     SODA 001
107     MEA  277
110     TRA  444
111     ADA  300
112     MEA  444
113     TRA  111
114     ADDA 001
115     MEA  111
116     SLT  104
117     ........
```

(a)

```
077     ........
100     TDA  000
101     MEA  444
102     TDA
103     MEA  277
104     TRA  444
105     ADA  300
106     MEA  444
107     TRA  105
110     ADDA 001
111     MEA  105
112     TRA  277
113     SODA 001
114     MEA  277
115     SAP  104
116     ........
```

(b)

Fig. 3.20 - Due varianti del programma per la somma di 100 numeri.

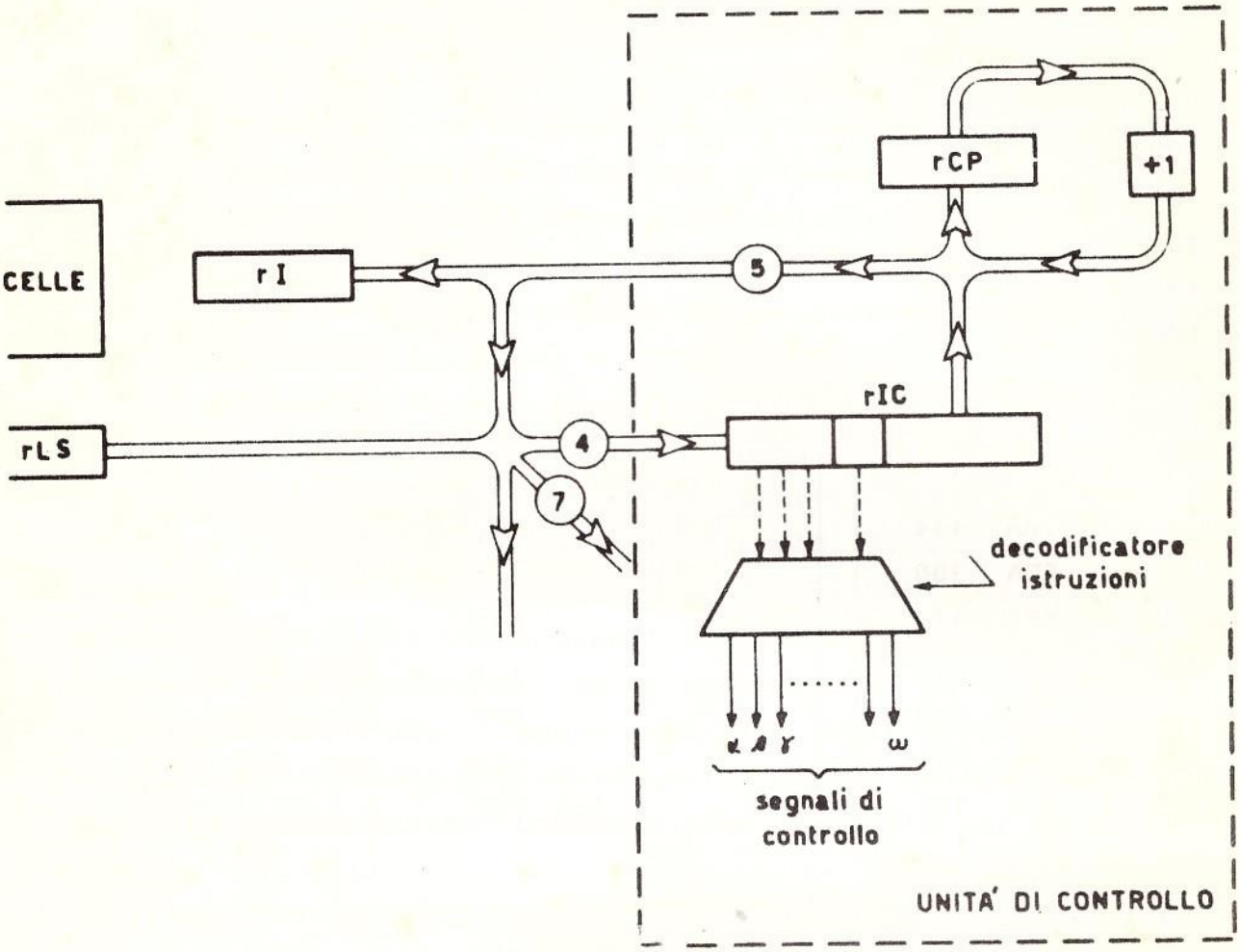


Fig. 3.21 - Registri e vie di comunicazione dell'unità di controllo.

3.11 - Fase interpretativa e fase esecutiva dell'istruzione.

Abbiamo già parlato nel paragrafo 3.1 della fase interpretativa e della fase esecutiva dell'istruzione, dando una descrizione molto grossolana degli eventi che le compongono. Vedremo ora questi eventi con maggior precisione; dobbiamo però innanzitutto esaminare la struttura funzionale della unità di controllo.

Nell'unità di controllo troviamo due registri: il registro istruzione corrente rIC, lungo 18 bit, ed il registro contatore programma rCP, lungo 9 bit. Questi due registri, e le vie di comunicazione che interessano l'unità di controllo, sono mostrati nella fig. 3.21. Il registro rCP contiene l'indirizzo dell'istruzione in esecuzione, mentre il registro rIC contiene l'istruzione stessa. Le tre suddivisioni indicate nel registro rIC corrispondono al campo codice, al campo indice, ed al campo indirizzo: le frecce tratteggiate che puntano dal campo codice ed indice verso la scatola trapezoidale denominata decodificatore istruzioni indicano che i segnali di controllo α, β, γ, ...che escono da questa scatola dipendono dal contenuto del campo codice ed indice.

Poichè non abbiamo ancora illustrato la funzione dei registri indice, semplificheremo la descrizione seguente supponendo ancora che essi non vengano mai adoperati. Siamo ora in grado di illustrare gli eventi che costituiscono le fasi interpretativa ed esecutiva.

All'inizio della fase interpretativa, l'indirizzo della istruzione da eseguire è contenuto nei registri rCP ed rI. La fase interpretativa consta dei due eventi:

(i1) (rLS)$\leftarrow$((rI))

(i2) (rIC)$\leftarrow$(rLS)

cioè, l'istruzione viene letta dalla cella di memoria in rLS, e trasferita poi nel registro istruzione corrente rIC.

Gli eventi successivi (fase esecutiva) dipendono dal tipo di istruzione, mentre i precedenti (i1) ed (i2) sono sempre gli stessi qualunque sia l'istruzione che sta per essere eseguita. Gli eventi della fase esecutiva sono regolati dai diversi segnali di controllo prodotti a seconda del contenuto del campo codice di rIC. Se si tratta di una istruzione ADA (somma ad rA), per esempio, gli eventi sono i seguenti:

(e1) (rI)$\leftarrow$(rIC)$_i$

(e2) (rLS)$\leftarrow$((rI))

(e3) (rA)$\leftarrow$(rA) + (rLS)

(e4) (rCP),(rI)$\leftarrow$(rCP) + 1

L'evento (e1) provoca il trasferimento dell'indirizzo dell'operando dal registro rIC al registro rI. L'operando viene poi letto dalla memoria in rLS, (e2), e sommato al contenuto di rA, (e3). L'ultimo evento, (e4), prepara nel registro di indirizzamento rI e nel registro contatore programma rCP, l'indirizzo dell'istruzione seguente; a questo punto, può iniziare la fase interpretativa dell'istruzione seguente.

Si noti che in (e4) abbiamo introdotto a sinistra della freccia una nuova notazione, che ha il significato seguente: il contenuto del registro rCP incrementato di una unità viene trasferito in entrambe i registri rCP ed rI.

Per una istruzione SLT (salto incondizionato) la fase esecutiva diviene:

(e) $(rCP), (rI) \leftarrow (rIC)_i$

e per una istruzione SAZ (salto se (rA) zero):

(e) $\{(rA) = 000000\}\ (rCP), (rI) \leftarrow (rIC)_i$
$\{(rA) \neq 000000\}\ (rCP), (rI) \leftarrow (rCP) + 1$

Ancora una notazione nuova nell'evento precedente:

{A} B;

l'interpretazione è ovvia: A è una condizione che deve essere verificata perchè si realizzi B.

Ritorneremo più oltre sulla struttura dell'unità di controllo, e sul susseguirsi degli eventi che costituiscono le due fasi.