UNIVERSITA' DEGLI STUDI DI PISA
ISTITUTO DI SCIENZE DELL' INFORMAZIONE

A. Grasselli

DESCRIZIONE DI UN CALCOLATORE DIDATTICO

Parte II

Note per il Corso di
Teoria ed Applicazioni delle Macchine Calcolatrici

ANNO ACCADEMICO 1972 - 73

A. Grasselli

TEORIA ED APPLICAZIONI DELLE MACCHINE CALCOLATRICI

Capitolo IV

Mancano in queste dispense le figg. 4.3, 4.4, 4.8, che non sono comunque indispensabili nella comprensione del testo.

4.1 - I registri indice

Abbiamo finora ignorato il significato del campo indice, e supposto che nelle istruzioni esaminate esso contenesse la configurazione 000. Ognuna delle altre 7 possibili configurazioni di questo campo (001,010,...,111) specifica uno dei 7 registri indice:(*) quando il campo indice contiene il numero j, il registro indice specificato è rX[j](per esempio, j=110 specifica rX[6]).

In tutte le istruzioni che abbiamo esaminato nel capitolo precedente, e nelle altre degli stessi tipi, il contenuto del registro indice specificato nella istruzione viene sommato al contenuto del campo indirizzo per costituire un indirizzo effettivo (dell'operando, o di salto), o un operando effettivo (nelle istruzioni dirette). Nella fase esecutiva di una istruzione ADA (somma ad rA), che è stata esaminata come esempio nel par. 3.11, l'evento (e1), nel quale la parte indirizzo di rIC viene trasferita in rI, è preceduto da altri due eventi, che hanno lo scopo di predisporre l'indirizzo effettivo:(*) (**)

(e1a) rLSI ⟵ rLSI+rX[J]

(e1b) I ⟵ rLSI

Supponiamo, per esempio, che:

rX[3] = 234

e consideriamo l'istruzione:

ADA 3 111

L'effetto di questa istruzione, perciò, è quello di sommare al registro rA il contenuto della cella di indirizzo:

rX[J] + I = 234 + 111 = 345

Si noti che né il campo indirizzo della cella che contiene l'istruzione, né il registro indice specificato vengono modificati dall'esecuzione.

Se supponiamo che J=000 specifichi un ottavo registro indice rX0, il cui contenuto è sempre zero, possiamo descrivere così l'effetto dell'istruzione ADA:

rA ⟵ rA + Mw[I+rX[J]]

(*)- L'insieme dei registri indice verrà indicato con rX[1:7] ⟨8:0⟩.

(**)- Definiamo rLSI:=rLS ⟨8:0⟩ .

Per tutte le istruzioni esaminate, l'indirizzo od operando effettivo vengono calcolati come per ADA. Perciò, la tabella della fig.3.16 può essere modificata come nella tabella 4.1 tenendo conto dei registri indice.

istruzione	c	sigla	effetto
trasferimento in rA	01	TRA	rA ⟵ Mw[I+rX[J]]
somma ad rA	14	ADA	rA ⟵ rA + Mw[I+rX[J]]
sottrazione da rA	15	SOA	rA ⟵ rA - Mw[I+rX[J]]
memorizzazione di rA	10	MEA	Mw[I+rX[J]] ⟵ rA
trasf. diretto in rA	03	TDA	rA ⟵ I+rX[J]
somma diretta ad rA	16	ADDA	rA ⟵ rA + I+rX[J]
sottraz. diretta da rA	17	SODA	rA ⟵ rA - I-rX[J]
salto incondizionato	40	SLT	istruz.seguente nella cella I+rX[J]
salto se rA zero	50	SAZ	se rA zero, istruz.successiva nella cella I+rX[J]

Tab. 4.1 - Effetto delle istruzioni del capitolo precedente tenendo conto dei registri indice.

Se nella somma del contenuto del campo indirizzo al contenuto del registro indice (evento (e1a) precedentemente esaminato) i due addendi sono tali che la loro somma è maggiore di $777_{(8)}$, il riporto che si origina nella cifra binaria più significativa viene ignorato, cioè l'operazione di somma avviene modulo 2^9. Perciò, se:

rX[5] = 765

e viene eseguita l'istruzione:

TDA 5 456

dopo la sua esecuzione:

rA = 000443

poichè:

modulo 2^9 $(765_{(8)} + 456_{(8)}) = 443_{(8)}$

Come abbiamo già preannunciato nel capitolo precedente, i registri indice servono come contatori e come modificatori di indirizzo. Prima di illustrarne l'impiego con alcuni esempi, è tuttavia necessario esaminare 6 nuove istruzioni, che manipolano il contenuto dei registri indice. La prima di queste è il trasferimento nel registro indice (TRX): questa istruzione trasferisce nel registro indice specificato dal campo indice, gli ultimi 9 bit della cella di memoria specificata dal campo indirizzo. Per esempio, supponiamo che:

Mw [765] = 123456

Dopo l'esecuzione dell'istruzione:

TRX 4 765

il contenuto di rX [4] sarà:

rX[4] = 456

E' possibile anche trasferire direttamente il contenuto del campo indirizzo nei registri indice, mediante l'istruzione di trasferimento diretto nel registro indice (TDX); per esempio, dopo l'esecuzione dell'istruzione:

TDX 4 765

il contenuto di rX[4] sarà:

rX[4] = 765

Le altre quattro istruzioni che modificano il contenuto dei registri indice sono la somma al registro indice (ADX), la somma diretta al registro indice (ADDX), la sottrazione dal registro indice (SOX), e la sottrazione diretta dal registro indice (SODX). Supponiamo, per esempio, che:

rX[5] = 444 , Mw[333] = 521320

L'effetto dell'esecuzione di una delle precedenti istruzioni, con J =5 ed I=333, sul contenuto di rX5, è rispettivamente:

ADX 5 333 : rX[5] = 764
ADDX 5 333 : rX[5] = 777
SOX 5 333 : rX[5] = 124
SODX 5 333 : rX[5] = 111

Le quattro istruzioni precedenti funzionano mod 2^9: perciò, per esempio le due istruzioni:

ADDX J 773 SODX J 005

sono del tutto e per tutto equivalenti.

Poichè le configurazioni nei registri indice rappresentano numeri di 9 bit senza segno, è necessario precisare come funzionano le 2 istruzioni SOX e SODX: esse sommano al contenuto del registro indice specificato i complementi a 2^9 (rispettivamente, del contenuto del campo indirizzo della cella i, e del contenuto del campo indirizzo della istruzione stessa). Quindi, se:

rX[4] = 543 , Mw [722] = 111766

le 4 istruzioni:

ADX 4 722 , ADDX 4 722 , SOX 4 722 , SODX 4 722

hanno rispettivamente l'effetto:

rX[4] = 531 , rX[4]= 465 , rX[4] = 555 , rX[4] = 621

Quando, dopo l'esecuzione di una delle 4 istruzioni ADX, ADDX, SOX e SODX, il registro indice interessato contiene la configurazione 000, si verifica una condizione eccezionale: viene eseguito uno skip, viene saltata, cioè, l'istruzione seguente (si dice anche che l'istruzione seguente viene skippata). Consideriamo come esempio il segmento di programma:

```
        .......
200     TDX   1   010
201     ---   -   ---
202     ---   -   ---
203     SODX  1   001
204     SLT   0   201
205     ---   -   ---
        ............
```

In questo segmento, il ciclo (costituito dalle 4 istruzioni in 201, 202, 203 e 204) viene eseguito 7 volte; dopo l'ottava esecuzione dell SODX in 203, rX[1] =0: viene quindi eseguito lo skip, cioè viene saltat l'istruzione in 204 ed obbedita l'istruzione in 205.

Come secondo esempio, si consideri il segmento di programma:

```
     ........
300  TDX  2  777
301  ADA  2  501 <-┐
302  SODX 2  002   |
303  SLT  0  301 -┘
304
     ........
```

777 è dispari

L'esecuzione della SODX in 302 non provoca mai uno skip (perchè?), e quindi il programma non esce mai dal ciclo delle 3 istruzioni in 301, 302 e 303.

L'esecuzione di una istruzione:

TRX, TDX, ADX, ADDX, SOX e SODX

con campo indice J=0, non ha alcun effetto sul calcolatore. Oltre alle precedenti 6 istruzioni, esiste anche una istruzione di memorizzazione del registro indice (MEX), che trasferisce negli ultimi 9 bit della cella specificata dal campo indirizzo il contenuto del registro indice specificato dal campo indice; l'esecuzione di questa istruzione lascia invariati i primi 9 bit della cella interessata. L'esecuzione di una istruzione:

MEX 0 I

trasferisce nove 0 nella cella I.

Le caratteristiche delle 7 istruzioni che manipolano il contenuto dei registri indice sono riassunte nella Tabella 4.2.

istruzione	c	sigla	effetto
trasferimento in rXj	06	TRX	rX[J] ← Mw[I]<8:0>
trasf. diretto in rXj	07	TDX	rX[J] ← I
somma ad rXj	22	ADX	rX[J] ← rX[J] + Mw[I]<8:0> skip se rX[J] = 0
somma diretta ad rXj	24	ADDX	rX[J] ← rX[J] + I skip se rX[J] = 0
sottrazione da rXj	23	SOX	rX[J] ← rX[J] - Mw[I]<8:0> skip se rX[J] = 0
sottraz. diretta da rXj	25	SODX	rX[J] ← rX[J] - I skip se rX[J] = 0
memorizzazione di rXj	13	MEX	Mw[I]<8:0> ← rX[J]

Tab. 4.2 - Le istruzioni che modificano i registri indice

4.2 - Una seconda visita al programma per la somma di 100 numeri

Possiamo ora riprogrammare il problema della somma di 100 numeri impiegando i registri indice. Una prima versione di tale programma è la seguente (versione A):

```
077  ............
100  TDX  1  144    j ← 144(8)
101  TDA  0  000    rA ← 0
102  ADA  1  277    rA ← rA + a_j
103  SODX 1  001    j ← j-1
104  SLT  0  102    rCP ← 102
105  MEA  0  444    s ← rA
106  ............
```

La interpretazione di questo programma non dovrebbe creare difficoltà ai nostri lettori; si noti che i dati vengono sommati nell'ordine:

$a_{100}, a_{99}, \ldots, a_1$

cioè per indici decrescenti. Ecco invece un programma che li somma per indici crescenti (versione B):

```
077  ............
100  TDX  1  634    j ← 1000(8) - 144(8) = 634(8)
101  TDA  0  000    rA ← 0
102  ADA  1  444    rA ← rA + a_j
103  ADDX 1  001    j ← j+1
104  SLT  0  102    rCP ← 102
105  MEA  0  444    s ← rA
106  ............
```

Si noti come la lunghezza del programma si sia ridotta da 14, 15 o 16 istruzioni (vedi le 3 versioni del capitolo precedente) a sole 6 istruzioni; un esame più approfondito mostra inoltre che il miglioramento ottenuto è ancora più sostanziale: infatti, il ciclo, che viene eseguito 100 volte, consta di 12 istruzioni per il programma della fig. 3.19, e di 11 e 10 rispettivamente per le 2 versioni della fig. 3.20, mentre è di sole 3 istruzioni per i due programmi di questo paragrafo.

Il tempo di esecuzione di un programma è naturalmente proporzionale al numero di istruzioni eseguite; un confronto fra le varie versioni del nostro programma è riportato nella tabella 4.3.

versione	numero istruzioni
fig. 3.19	4 + 12.99 + 6 = 1198
fig. 3.20(a)	4 + 11.99 + 2 = 1095
fig. 3.20(b)	4 + 10.99 = 994
con i registri indice (versioni A e B)	2 + 3.99 + 3 = 302

Tab. 4.3 - Confronto fra le varie versioni per il programma di somma di 100 numeri.

Supponiamo ora di voler generalizzare il nostro problema: vogliamo un programma che somma N numeri $a_1, a_2, \ldots, a_N$, dove N è uno dei dati del nostro problema. Supponiamo che il dato N sia stato memorizzato dal segmento di lettura nella cella 277, e che $a_1, a_2, \ldots, a_N$ siano stati memorizzati nelle celle 300,301,...,300+N-1. E' opportuno memorizzare il risultato in una cella diversa dalla 444 se si prevede di usare il programma per N > 100, e se i dati $a_1, a_2, \ldots, a_N$ servono anche nel seguito del programma: la memorizzazione di s in 444 cancellerebbe infatti il dato a_{101}.

Ecco come si modifica la versione A (chiameremo questo nuovo programma "versione A_1"):

```
077  .............
100  TRX  1  277      j ← N   (questa è l'unica istruzione
101  TDA  0  000              modificata)
102  ADA  1  277
103  SODX 1  001
104  SLT  0  102
105  MEA  0  777      s ← rA  (qui è stato modificato solamente
106  .............             l'indirizzo)
```


La versione B non è altrettanto facilmente modificabile: perchè?

Una ulteriore generalizzazione: vogliamo un programma che sommi N numeri memorizzati in N celle consecutive di una zona qualsiasi del memoria; supponiamo che il segmento di lettura abbia memorizzato nella cella 276 l'indirizzo di a_1 (indichiamolo con $i(a_1)$), e nella cella 277 il numero N. La versione A_1 può essere modificata così (versione A_2):

```
077  .............
100  TRX  1  277      rX[1] ← N
101  ADX  1  276      rX[1] ← rX[1] + i(a1)
102  TRX  2  277      rX[2] ← N
103  TDA  0  000
104  ADA  1  000 ←    (l'indirizzo è ora 000)
105  SODX 1  001
106  SODX 2  001
107  SLT  0  104
110  MEA  0  777 ←
111  .............
```

In questo programma, tutto l'indirizzo di $a_N, a_{N-1}, \ldots, a_1$ è ora contenuto in rX[1] (che serve come modificatore di indirizzo); rX[2] viene usato come contatore.

Alternativamente, l'indirizzo $i(a_1)$ può essere "impiantato" nella istruzione ADA nel modo seguente (versione A_3):

```
077  .............
100  TRA  0  276
101  ADA  0  105
102  MEA  0  105
103  TRX  1  277
104  TDA  0  000
105  ADA  1  000
106  SODX 1  001
107  SLT  0  105
110  MEA  0  777
111  .............
```

4.3 - Gli indicatori di confronto

Parlando della struttura della unità di elaborazione, abbiamo detto che in essa si trovano: l'indicatore di supero iSP (lungo 1 bit), e l'indicatore di confronto (lungo 2 bit) iCF <1:2>.

In questo paragrafo parleremo dell'indicatore di confronto, e nel paragrafo seguente esamineremo la funzione dell'indicatore di supero.

L'indicatore di confronto viene modificato dalle istruzioni di confronto. L'istruzione di confronto con il registro rA (CFA) ha il seguente effetto:

se rA < Mw[I+rX[J]] : iCF ← 01
se rA = Mw[I+rX[J]]: iCF ← 11
se rA > Mw[I+rW[J]] : iCF ← 10

Del tutto analogamente funzionano le istruzioni di confronto con il registro indice rX[J] (CFX), e di confronto diretto con il registro indice rX[J] (CFXD).

L'indicatore di confronto conserva il suo stato fino a quando questo viene mutato attraverso una nuova istruzione di confronto. Si noti che dopo l'esecuzione di una qualsiasi delle 3 istruzioni non si verifica mai la condizione:

iCF = 00.

Una istruzione CFX con campo indice J=0 si comporta nel modo seguente: il campo indirizzo della cella i viene confrontato con il registro indice (ideale) rX0 che, come abbiamo detto, contiene nove 0. Analogamente, CFXD con campo indice J=0: il campo indirizzo della istruzione viene confrontato con 0.

Le caratteristiche delle 3 istruzioni di confronto sono riassunte nella tabella 4.4.

Lo stato dell'indicatore di confronto può essere interrogato mediante 3 istruzioni di salto: salto se il contenuto del registro è minore (SMIN), salto se il contenuto del registro è uguale (SUG), e salto se il contenuto del registro è maggiore (SMAG).

Nel caso dell'istruzione SMIN, il salto viene eseguito (ed il controllo passa quindi all'istruzione nella cella I+rX[J]) se:

iCF = 01.

istruzione	c	sigla	effetto
confronto con rA	34	CFA	iCF ← ((rA < Mw[I+rX[J]]) ⟹ 01 (rA > Mw[I+rX[J]]) ⟹ 10 (rA = Mw[I+rX[J]]) ⟹ 11)
confronto con rX[J]	36	CFX	iCF ← ((rX[J] < Mw[I]<8:0>) ⟹ 01 (rX[J] > Mw[I]<8:0>) ⟹ 10 (rX[J] = Mw[I]<8:0>) ⟹ 11)
confronto diretto con rX[J]	37	CFXD	iCF ← ((rX[J] < I) ⟹ 01 (rX[J] > I) ⟹ 10 (rX[J] = I) ⟹ 11)

Tab.4.4 - Tre istruzioni di confronto

Perciò il salto viene eseguito se, in una precedente istruzione il confronto, il contenuto del registro confrontato (rA o rX[J]) era minore del termine di confronto.

Per l'istruzione SUG, il salto viene eseguito se:

iCF = 11

e per SMAG, infine, se:

iCF = 10.

La tabella 4.5 riassume le caratteristiche delle istruzioni SMIN, SUG e SMAG:

istruzione	c	sigla	effetto
salto se minore	43	SMIN	se (iCF=01), prossima istruzione in I+rX[J]
salto se ~~uguale~~ maggiore (10)	44	SMAG	se (iCF=10) prossima istruzione in I+rX[J]
salto se ~~maggiore~~ uguale (11)	45	SUG	se (iCF=11) prossima istruzione in I+rX[J]

Tab.4.5 - Le istruzioni di salto che interrogano gli indicatori di confronto.

Come esempio di impiego delle istruzioni descritte in questo paragrafo, consideriamo il seguente segmento, che mostra l'uso di CFXD ed SMIN: esso è una ennesima versione del programma che somma 100 numeri (la chiameremo "versione C"):

```
077   ..............
100   TDX   1   000      j ← 0
101   TDA   0   000      rA ← 0
102   ADA   1   300←     rA ← rA+a_j
103   ADDX  1   001      j ← j+1
104   CFXD  1   144      se rX[1]<144 allora iCF ← 01
105   SMIN  0   102─     se iCF = 01 salta a 102
106   MEA   0   444      s ← rA
107   ..............
```


La versione C calcola la sommatoria per indici crescenti; essa richiede la esecuzione di 403 istruzioni, ed è quindi meno efficiente delle versioni A e B del par. 4.2.

Se vogliamo sommare N numeri, memorizzati in una zona qualsiasi della memoria (supponendo ancora che N ed $i(a_1)$ siano nelle celle 276 e 277 rispettivamente, e che si voglia il risultato s nella cella 777), possiamo impiegare il programma seguente (versione C_1):

```
077   ..............
100   TRX   1   277
101   TDX   2   000
102   TDA   0   000
103   ADA   1   000←
104   ADDX  1   001
105   ADDX  2   001
106   CFX   2   276
107   SMIN  0   103─
110   MEA   0   777
111   ..............
```

Questo programma impiega due registri indice, rX[1] ed rX[2]: il primo per modificare l'indirizzo dell'istruzione in 103, il secondo come contatore (che viene confrontato con il contenuto della cella 276). E' possibile risparmiare una istruzione nel ciclo (ed impiegare un solo registro indice) calcolando nella fase di inizializzazione la somma: $N+i(a_1)$, e memorizzando questa quantità in una cella disponibile, per esempio nella cella 275 (versione C_2):

```
077   ..............
100   TRA   0   276
101   ADA   0   277
102   MEA   0   275
103   TRX   1   277
104   TDA   0   000
105   ADA   1   000←
106   ADDX  1   001
107   CFX   1   275
110   SMIN  0   105─
111   MEA   0   777
112   ..............
```

Invece delle 4 + 5N istruzioni eseguite con la versione precedente, vengono eseguite con questa versione 6 + 4N istruzioni.

Un'ultima precisazione sull'impiego dell' indicatore di confronto: non è naturalmente necessario che l'istruzione di salto (SMIN, SUG, SMAG) segua direttamente l'istruzione di confronto; infatti, come è stato detto, l'indicatore conserva il suo stato, modificabile solamente attraverso un'altra istruzione di confronto, ed è quindi interrogabile, anche ripetutamente, in un punto qualsiasi del programma. Per esempio:

```
350   .............
351   CFA   4   555
352   ADA   0   646
353   ADA   0   464
354   SUG   0   357 ─┐
355   ──    ──  ── │
356   ──    ──  ── │
357   MEA   0   647 ←┘
360   SUG   6   421 ─┐
361   ............. ↓
```

Se nel programma precedente:

$$rA = Mw[555 + rX[4]]$$

il salto SUG in 354 viene eseguito, e così pure il salto SUG in 360.

4.4 - Il supero di capacità e l'indicatore di supero

Fino ad ora, abbiamo ignorato una questione importante: che cosa succede quando, nella esecuzione di una istruzione di somma (ADA, ADDA) o di sottrazione (SOA, SODA) viene superata la capacità del registro rA? Come sappiamo, è possibile rappresentare in una parola del CANE un intero I appartenente all'intervallo:

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

e quindi anche il risultato di una operazione di somma o di sottrazione

deve essere contenuto in questo intervallo per essere rappresentabile nel registro rA o in una cella di memoria.

Teniamo presente che:

$$2^{17} - 1 = 377777_{(8)}$$

e supponiamo che:

$$Mw[222] = +346571_{(8)}$$

$$rA = +140000_{(8)}$$

La somma supera la capacità delle parole del CANE:

$$Mw[222] + rA = +506571_{(8)}$$

Nessuno, tuttavia, ci impedisce di sommare questi due addendi; se, con i precedenti contenuti, viene eseguita l'istruzione:

```
.............
ADA   0   222
.............
```

si dice che l'istruzione ha provocato un supero di capacità del primo accumulatore. Quando questo avviene viene settato(*) l'indicatore di supero: iSP ⟵ 1. Poichè il calcolatore procede in ogni caso all'esecuzione della istruzione successiva, è necessaria una indicazione esplicita sul verificarsi della condizione (anomala) di supero di capacità: questa è contenuta nello stato dell'indicatore di supero, iSP, che, come abbiamo detto, è stato settato nel corso dell'operazione. Lo stato dell'indicatore di supero è interrogabile, nel programma, mediante l'istruzione di salto se l'indicatore di supero è settato o, più brevemente, salto se supero (SSP). Questa istruzione, che ha campo codice c=42, ha l'effetto seguente:

se (iSP=1) allora {iSP ⟵ 0 e prossima istruzione in I+rX[J]}.

Cioè, se l'indicatore di supero è settato, viene eseguito un salto, e l'indicatore viene risettato. L'istruzione di salto se supero permette quindi di individuare il verificarsi di una condizione di supero di capacità di un registro durante l'esecuzione di un programma.

(*) Si usa dire settare e risettare un indicatore per indicare rispettivamente il trasferimento di un 1 e uno Ø nell'indicatore.

4.5 - La moltiplicazione e la divisione

Oltre al registro rA, esiste nell'unità di elaborazione del CANE un altro registro che può contenere una parola di 18 bit: il registro rB. In questo registro, si possono eseguire molte delle operazioni tipiche del registro rA, ed in particolare:

- trasferimento in rB (TRB)
- somma ad rB (ADB)
- sottrazione da rB (SOB)
- memorizzazione di rB (MEB)
- confronto con rB (CFB)
- salto se (rB) negativo (SBN)
- salto se (rB) zero (SBZ)
- salto se (rB) positivio (SBP)

Il registro rB è però da considerarsi il parente povero di rA, perchè non possiede le istruzioni con operando diretto (TDA, ADDA,...). Oltre alle istruzioni precedenti, esistono due istruzioni che riguardano insieme i registri rA ed rB: le istruzioni di moltiplicazione e divisione.

L'istruzione di moltiplicazione (MOL) esegue il prodotto del numero contenuto in rB per il numero nella cella di indirizzo $I+rX[J]$; poichè il moltiplicando ed il moltiplicatore sono numeri di 18 bit, il prodotto è un numero di 36 bit: ad istruzione eseguita, la metà più significativa del prodotto si trova (con il suo segno) nel registro rA, e la metà meno significativa (con il suo segno) nel registro rB. Si noti che la metà più significativa del prodotto (cioè il contenuto di rA) va pensata moltiplicata per 2^{17}:

$$Mw[I + rX[J]] \times rB \longrightarrow rA \times 2^{17} + rB$$

prima dell'operazione — dopo l'operazione

E' certamente utile che il lettore consideri attentamente gli esempi seguenti (la lunghezza della parola è stata anche qui limitata a 5 bit):

Primo esempio

$Mw[I+rX[J]] = +15$ rB = +13 $+15 \times (+3) = +195$

dopo l'operazione:

rA = $+12 \times (2^4)$ rB = +3

I+rX[J]		rB		rA		rB
0 1 1 1 1	x	0 1 1 0 1	→	0 1 1 0 0	+	0 0 0 1 1
+15		+13		$+12.2^4$		3

Secondo esempio

$Mw[I+rX[J]] = +15$ rB = -13 $+15 \times (-13) = -195$

dopo l'operazione:

rA = $-12 \times (2^4)$ rB = -3

I+rX[J]		rB		rA		rB
0 1 1 1 1	x	1 0 0 1 1	→	1 0 1 0 0	+	1 1 1 0 1
+15		-13		-12.2^4		-3

Terzo esempio

$Mw[I+rX[J]] = +3$ rB = -1 $+3 \times (-1) = -3$

dopo l'operazione:

rA = 0 rB = -3

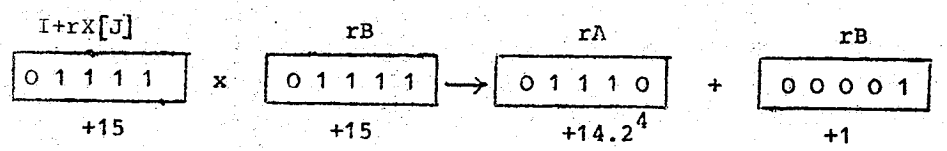
I+rX[J]		rB		rA		rB
0 0 0 1 1	x	1 1 1 1 1	→	0 0 0 0 0	+	1 1 1 0 1
+3		-1		0		-3

Quarto esempio

Mw[I+rX[J]]= +15 rB = +15 +15 x (+15) = +225

dopo l'operazione:

rA = +14 x (2^4) rB = +1

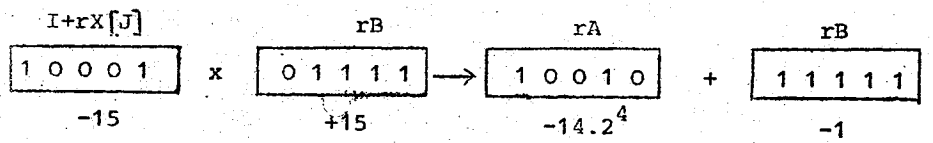


Quinto esempio

Mw[I+rX[J]] = -15 (rB) = +15 +15 x (+15) = -225

dopo l'operazione:

(rA) = -14 x (2^4) (rB) = -1



L'istruzione di divisione (DIV) esegue la divisione del numero contenuto in rA ed rB (la metà più significativa, con il suo segno, in rA, e la metà meno significativa, con il suo segno, in rB) per il numero nella cella di indirizzo I+rX[J] : ad istruzione eseguita, il quoziente si trova in rB, ed il resto in rA; il resto ha sempre lo stesso segno del dividendo.

Perchè la divisione dia risultato corretto, debbono essere rispettate alcune condizioni: innanzitutto, non può essere:

$$|\,rA\,| \geq |\,Mw[I+rX[J]]\,|$$

In questo caso, la divisione non viene eseguita e viene settato l'indicatore di supero iSP: infatti, il quoziente supererebbe la capacità del registro rB. Inoltre, le due metà del dividendo, in rA ed rB, debbono essere dello stesso segno, salvo nel caso che rA oppure rB

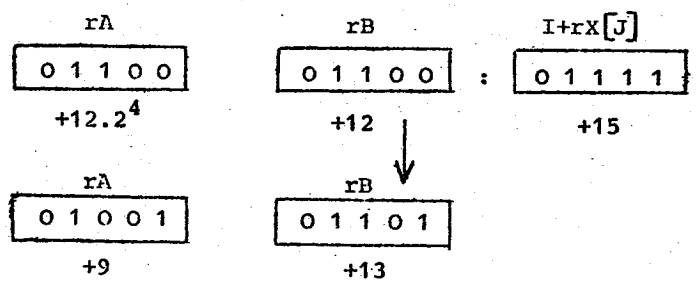
siano zero (si ricordi che nella rappresentazione in complemento a 2 lo zero ha segno positivo); se questa condizione non è rispettata, la divisione viene ugualmente eseguita, ma il risultato (quoziente e resto) è errato.

Primo esempio (si considerano solo 5 bit)

rA x 2^4+rB = +204 Mw[I+rX[J]] = +15

dopo l'operazione:

rB = +13 rA = +9

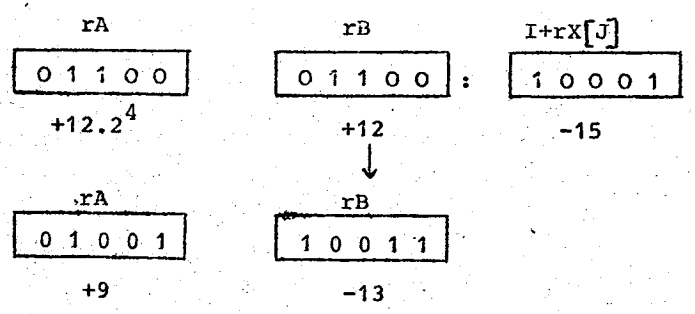


Secondo esempio

rA x 2^4+rB = +204 Mw[I+rX[J]] = -15

dopo l'operazione:

rB = -14 rA = -6



Terzo esempio

rA x 2^4+rB = -204 Mw[I+rX[J]] = +15

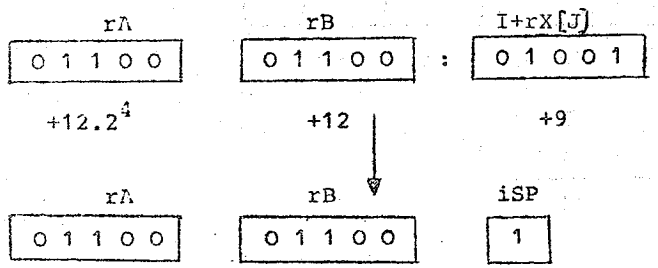
dopo l'operazione:

rA = -9 rB = -13

Quarto esempio

rA x 2^4+rB = +204 Mw[I+rX[J]] = +9

la divisione non viene eseguita, e viene settato iSP:



4.6 - Il lettore di schede e le istruzioni di lettura

Il CANE è equipaggiato di un solo organo di entrata (un lettore di schede), e di un solo organo di uscita (una stampante); di solito, i calcolatori reali dispongono invece di parecchie unità di entrata-uscita: nel seguito prenderemo in esame alcuni casi tipici.

La scheda perforata rappresenta, accanto al nastro perforato, uno dei due mezzi più comuni per introdurre l'informazione in un sistema di elaborazione. Esistono diversi tipi di scheda, ma di gran lunga la più diffusa è la cosiddetta scheda IBM (questa società ne deteneva il brevetto, ora scaduto da qualche anno). Tale scheda (vedi la fig. 4.2) ha 12 righe e 80 colonne; le posizioni di perforazione sono, perciò, 12 x 80 = 960. Come mostra la fig. 4.2, le prime due righe si chiamano riga Y, e riga X, e le altre 10 righe sono numerate dall'alto verso

il basso:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

mentre le colonne sono numerate da 1 a 80 da sinistra verso destra.

Ogni colonna della scheda può portare una, due o tre perforazioni; il gruppo di perforazioni di una colonna rappresenta un carattere alfanumerico, secondo un codice che si chiama codice Hollerith, e che può essere desunto da un esame della scheda nella fig. 4.2: per esempio, il carattere "(" viene codificato da perforazioni nelle righe 0,4 e 8.

La fig. 4.2 mostra i 47 caratteri che costituiscono l'alfabeto alfanumerico del CANE, così suddivisi:

- 26 lettere alfabetiche
- 10 cifre decimali
- 11 caratteri speciali

Abbiamo già esaminato come questi 47 caratteri vengano rappresentati nella memoria del CANE (vedi la Tab. I). Si noti che nell'insieme dei 47 caratteri abbiamo contato anche il carattere "spazio", che sulla scheda viene codificato dall'assenza di perforazioni in una colonna. Per semplicità, si usa di solito indicare lo spazio (sia per quanto riguarda la scheda, che per quanto riguarda la rappresentazione interna al calcolatore) con il simbolo "b" (iniziale del termine inglese "blank", che significa appunto "spazio" o "posizione non occupata").

L'alfabeto alfanumerico dei moderni calcolatori è più articolato di quello del CANE (che ha invece imparato ad abbaiare insieme a quelli della passata generazione), e comprende 64 caratteri; tuttavia, l'insieme di 47 caratteri è più che adeguato alle nostre esigenze, ed il CANE non soffre di alcun complesso di inferiorità sulle sue possibilità di comunicazione con il mondo esterno.

Le schede perforate da introdurre nel calcolatore vengono preparate manualmente mediante una macchina chiamata perforatrice (vedi la fig. 4.3) che non è nostra intenzione descrivere in questa sede; le abilità manuali necessarie per servirsene possono essere acquisite in un'ora al più da chiunque sappia usare (anche male!) una macchina da scrivere.

La fig. 4.4 mostra il lettore di schede del CANE; la fig. 4.5a) illustra molto schematicamente la struttura di un lettore di schede:

Carattere	Codice	Carattere	Codice
A	010001	0	000000
B	010010	1	000001
C	010011	2	000010
D	010100	3	000011
E	010101	4	000100
F	010110	5	000101
G	010111	6	000110
H	011000	7	000111
I	011001	8	001000
J	100001	9	001001
K	100010		
L	100011	b(spazio)	110000
M	100100	.	011011
N	100101	(	111100
O	100110	+	010000
P	100111	$	101011
Q	101000	*	101100
R	101001	)	011100
S	110010	-	100000
T	110011	/	110001
U	110100	,	111011
V	110101	=	001011
W	110110		
X	110111		
Y	111000		
Z	111001		

Tabella I - Codice caratteri

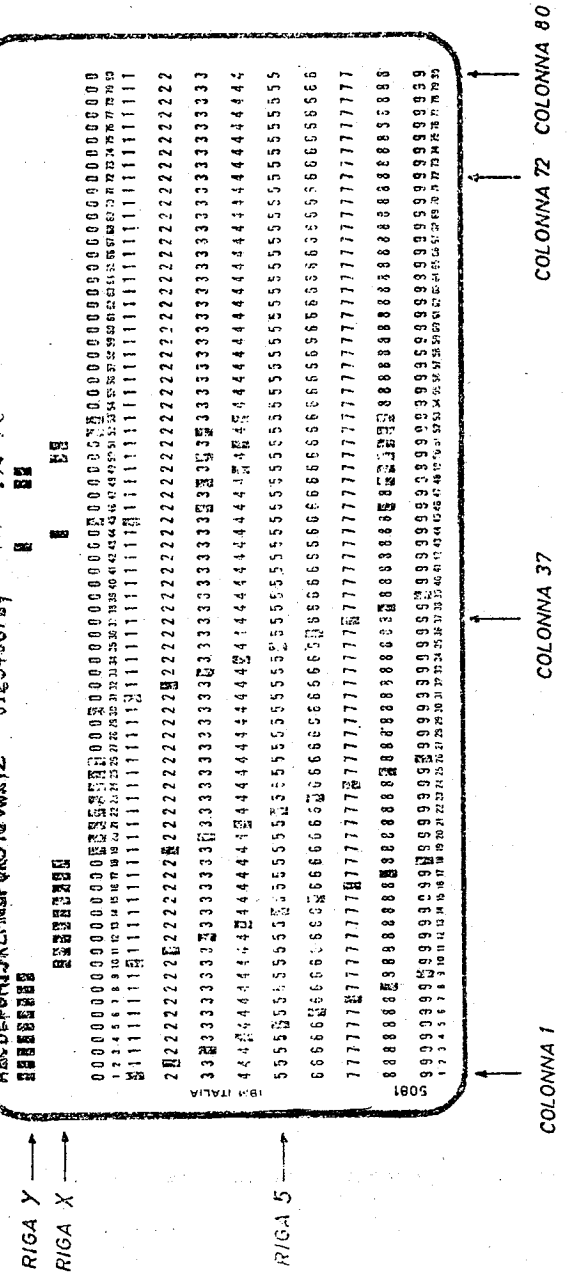


Fig. 4.2 - Una scheda perforata.

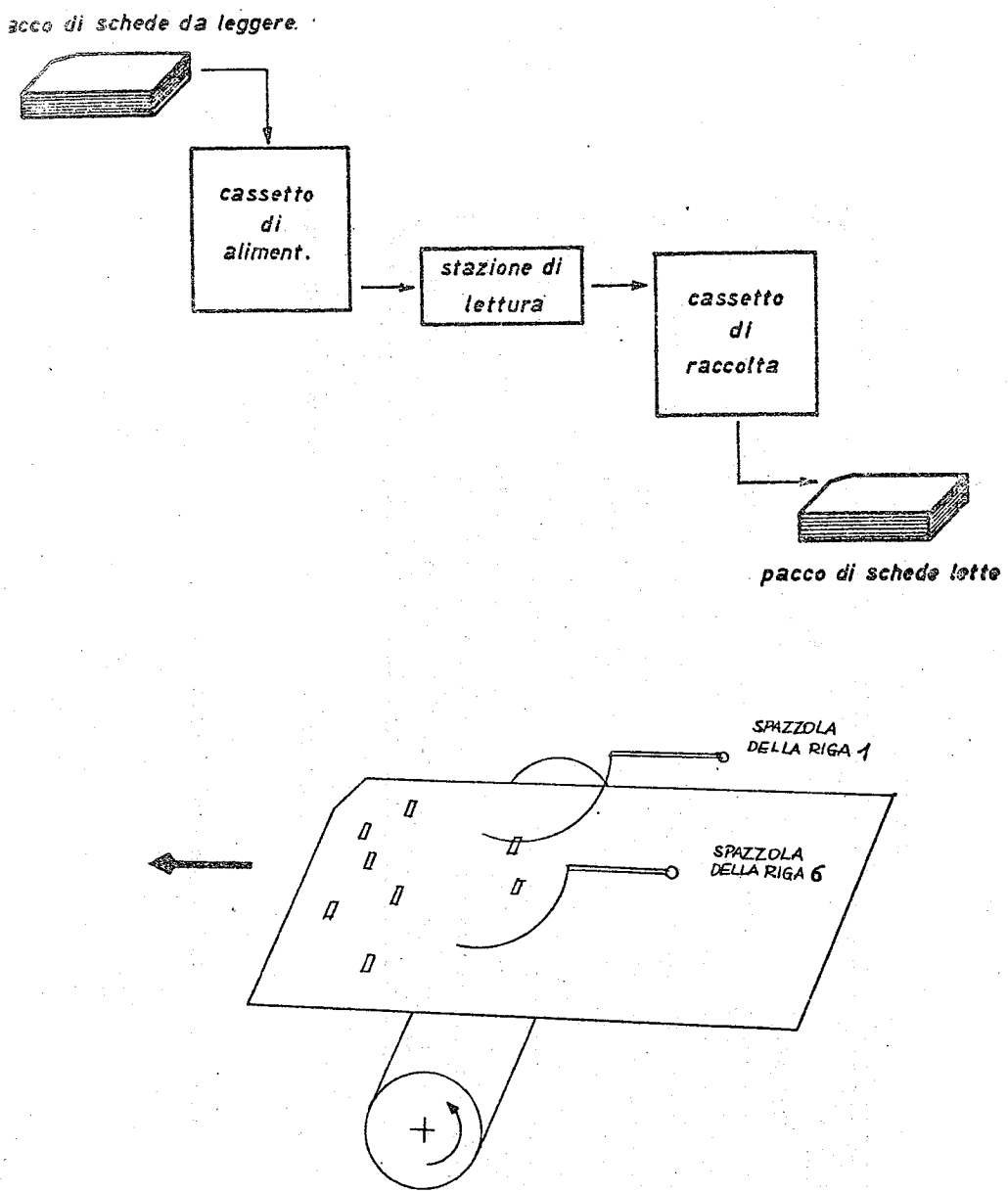


Fig. 4.5 - Principio operativo del lettore di schede.

le schede da leggere vengono deposte in un cassetto di alimentazione, passano una per una per una stazione di lettura, e finiscono in un cassetto di raccolta.

La lettura della scheda può essere meccanica o fotoelettrica, per righe o per colonne. La fig. 4.5 b) mostra il principio della lettura meccanica per righe : la scheda viene trascinata sotto una fila di spazzole, una per riga (per semplicità, la figura mostra solo due spazzole): la presenza di una perforazione in una determinata riga stabilisce in un determinato istante un contatto fra la spazzola ed il cilindro, contatto che può essere rivelato elettricamente. I lettori fotoelettrici impiegano invece elementi fotosensibili (fotocellule), che vengono attivati da luce trasmessa attraverso la perforazione (oppure, in altri lettori, dalla luce riflessa in assenza di perforazione).

Nei lettori che leggono per colonne, la scheda viene trascinata nell'altra direzione, ed esiste una spazzola (o una fotocellula) per ogni colonna della scheda.

I lettori meccanici sono più semplici e più lenti dei lettori fotoelettrici; i lettori per righe sono più economici e più lenti dei lettori per colonne. Alcune velocità tipiche:

meccanico per righe: 150 schede al minuto

" per colonne: 600 " "

fotoelettrico per colonne: 1500 "

Il lettore di schede del CANE non legge tutte le 80 colonne della scheda, ma solo le prime 72 (da sinistra); nelle colonne 73-80 il programmatore può perforare caratteri a piacere, perchè quell'informazione non entra nel calcolatore. Poichè i pacchi di schede, maligni come tutte le cose inanimate, hanno il brutto vizio di cadere ogni tanto per terra, sparpagliandosi sul pavimento, è saggio impiegare le ultime 8 colonne per numerare in modo progressivo le schede del pacco, così da poterle facilmente ordinare dopo l'inevitabile incidente.

Il CANE legge le schede in due diverse modalità: nella modalità binaria le schede vengono lette mediante l'istruzione di entrata binaria (ENB), e nella modalità alfanumerica mediante l'istruzione di entrata alfanumerica (ENA).

Nella modalità di entrata binaria si possono leggere solamente schede che contengono i caratteri: 0, 1 e b, e che verranno chiamate schede binarie (*); in questa modalità, tutti i b (spazi) vengono interpretati come 0. L'istruzione di entrata binaria:

```
ENB   J   I
```

legge all'interno del calcolatore una scheda; per quanto riguarda ENB, la scheda è suddivisa in 4 campi:

1^campo : le colonne da 1 a 18
2^campo : le colonne da 19 a 36
3^campo : le colonne da 37 a 54
4^campo : le colonne da 55 a 72

L'informazione perforata nei 4 campi costituisce 4 parole di 18 bit; il contenuto del primo campo viene memorizzato dall'istruzione ENB nella cella di indirizzo I+rX[J], il contenuto del secondo nella cella I+rX[J]+1 il contenuto del terzo nella cella I+rX[J]+2 ed infine il contenuto del quarto nella cella I+rX[J]+3. Per esempio, supponiamo che venga eseguita l'istruzione:

```
ENB   0   333
```

e che mediante essa venga letta la scheda binaria della fig. 4.6. Dopo l'esecuzione:

Mw[333] = 471000
Mw[334] = 050541
Mw[335] = 030000
Mw[336] = 270542

Nella modalità di entrata alfanumerica il calcolatore legge schede contenenti caratteri qualsiasi (fra i 47 ammissibili), e che verranno chiamate schede alfanumeriche. L'istruzione di entrata alfanumerica:

```
ENA   J   I
```

legge all'interno del calcolatore una scheda; per quanto riguarda ENA,

(*) Se si tenta di leggere, mediante ENB, una scheda che contiene altri caratteri oltre ad 1, 0 e b, il calcolatore si ferma.

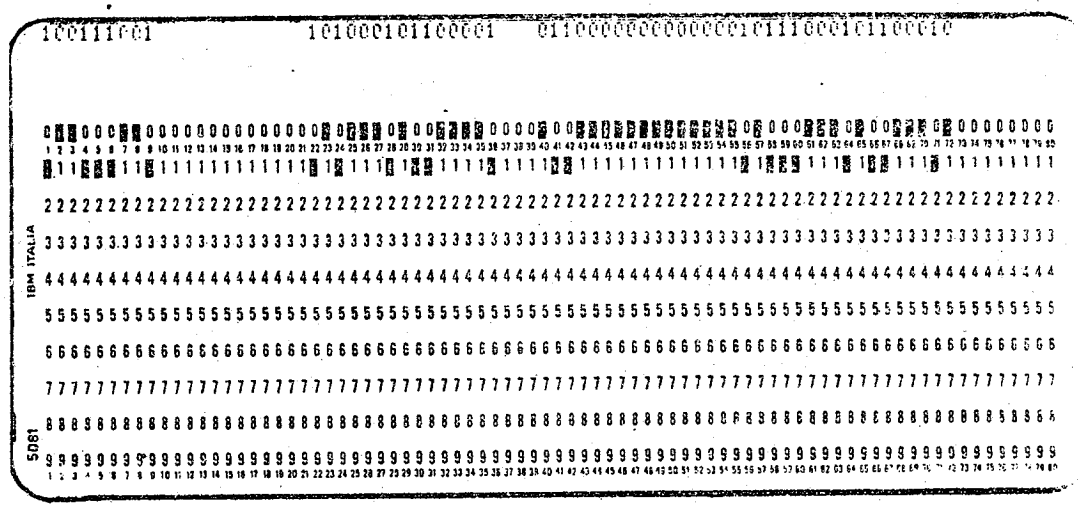


Fig. 4.6 - Scheda binaria.

la scheda è divisa in 24 campi:

1^ campo : le colonne 1,2,3
2^ campo : le colonne 4,5,6
.....
24^campo : le colonne 70,71,72

Nel corso della esecuzione dell'istruzione ENA, il contenuto di ogni campo (3 caratteri) viene convertito nella rappresentazione interna (secondo la Tab. I) e memorizzato in una cella di memoria: il primo campo nella cella I+rX[J] , il secondo nella cella I+rX[J]+1 ,...., il 24^ nella cella I+rX[J]+23.
Per esempio, supponiamo che venga eseguita l'istruzione:

```
ENA   0   440
```

e che mediante essa venga letta la scheda alfanumerica della fig. 4.7.
Dopo l'esecuzione:

```
Mw[440] = 224645 = 'BON'
Mw[441] = 214721 = 'APA'
Mw[442] = 516325 = 'RTE'
  ......................
Mw[467] = 446060 = 'Mbb'
```

dove con la notazione 'XXX' abbiamo indicato la rappresentazione interna dei caratteri XXX.

4.7 - Il controllo delle istruzioni di entrata-uscita.

Anche se non si è ancora parlato dei tempi di esecuzione delle istruzioni, possiamo premettere che le istruzioni "interne" (tutte quelle che abbiamo esaminato finora, salvo ENB ed ENA) hanno tempi di esecuzione molto più piccoli dei tempi di esecuzione delle istruzioni di entrata-uscita. Tipicamente, i moderni calcolatori eseguono una istruzione interna in un tempo variabile fra 1 e 10 μs (1 μs = 1 microsecondo = 10^{-6} secondi), a seconda del tipo di istruzione, mentre, per esempio, una scheda viene letta (da un lettore che operi a 600 schede al minuto) in 1/10 di secondo: il rapporto fra i tempi è perciò enorme,

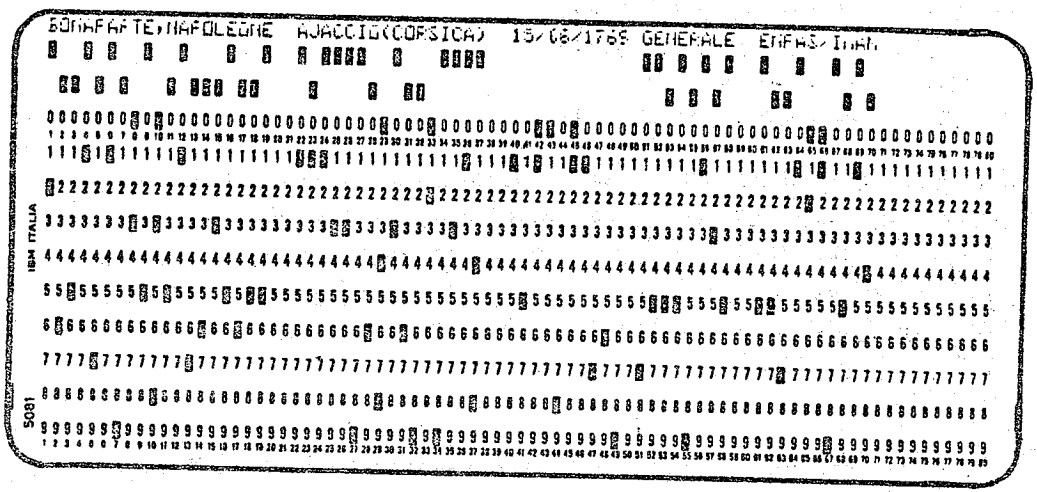


Fig. 4.7 - Scheda alfanumerica.

addirittura di 4 ordini di grandezza! Infatti, le apparecchiature di entrata-uscita hanno componenti meccanici, mentre gli organi interni della macchina sono elettronici.

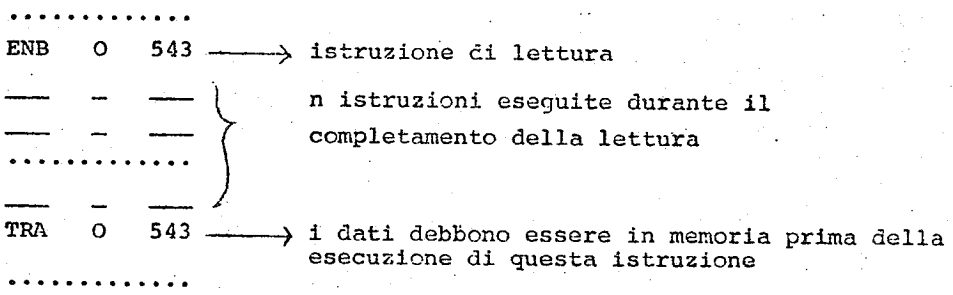
Se non venissero presi adeguati e complessi provvedimenti, che influiscono tanto sulla architettura delle macchine che sulle loro modalità di impiego, l'entrata ed uscita dell'informazione nel sistema costituirebbero un inaccettabile collo di bottiglia dell'elaborazione automatica, al punto da vanificare i considerevoli progressi raggiunti nelle tecnologie elettroniche, che hanno permesso di raggiungere velocità elevatissime nel funzionamento degli organi interni.

Nei sistemi semplici si cerca di sovrapporre il più possibile la esecuzione delle operazioni di entrata-uscita e la esecuzione del programma di elaborazione, come vedremo fra breve a proposito del CANE; nei sistemi complessi, le operazioni di lettura di schede e di stampa di tabulati vengono addirittura effettuate da un calcolatore periferico, che comunica con il calcolatore centrale (quello che elabora) attraverso unità a nastri magnetici, che possono essere letti e scritti dal calcolatore centrale a velocità paragonabili a quelle dell'esecuzione delle istruzioni interne.

Nel caso del CANE, il calcolatore inizia l'esecuzione dell'istruzione di entrata (ENB o ENA), e delega poi il completamento di queste istruzioni ad un organo di controllo (controllore periferico) che risiede nell'unità di entrata; liberatosi del compito di eseguire l'istruzione di entrata, che ha affidato al suo subordinato, l'unità di controllo procede nell'esecuzione delle istruzioni che seguono. Ad un certo punto, dopo i tempi propri delle apparecchiature elettromeccaniche proprie dei componenti del lettore di schede, quest'ultimo è in grado di fornire al calcolatore l'informazione letta sulla scheda: per esempio, 4 parole nel caso dell'istruzione ENB; le 4 parole debbono transitare, una per una, dalla via di comunicazione 6 della fig. 3.11, attraverso il registro rLS, per essere memorizzate nelle celle opportune. Di tutto questo si occupa il controllore periferico, senza disturbare l'unità di controllo: si noti che il controllore periferico non si può permettere di invadere il registro rLS (ed il registro rI, nel quale deve inviare l'indirizzo della cella nella quale la parola verrà memorizzata) in un momento qualsiasi, per esempio nel bel mezzo

dell'esecuzione di una istruzione, ma deve scegliere il momento opportuno: per semplicità, supporremo che tale trasferimento venga effettuato dopo la fine della fase esecutiva di una istruzione, e che durante il trasferimento stesso il controllore periferico inibisca temporaneamente la fase interpretativa della istruzione successiva.

Naturalmente, sorge a questo punto un grosso problema: come si fa a sapere quando, rispetto all'inizio dell'esecuzione dell'istruzione di entrata, tale istruzione è stata completata, e quindi i dati sono disponibili nella memoria? Lo scopo di sovrapporre nel tempo l'esecuzione di altre istruzioni del programma all'esecuzione dell'istruzione di entrata è il tempo considerevole impiegato per il completamento di quest'ultima: tale sovrapposizione permette di "lanciare" l'operazione di lettura molto tempo prima che i dati servono nell'elaborazione, continuando nel frattempo ad eseguire il programma. La situazione può essere schematizzata così:



Conoscendo il tempo necessario a leggere una scheda, ed il tempo di esecuzione delle istruzioni interne (*), è possibile calcolare (**) quante istruzioni si possono eseguire fra inizio e completamento dell'istruzione di lettura. Si noti che il calcolo precedente ci dice che

---

(*) Supponendo, per semplicità, che questo sia uguale per tutte le istruzioni.

(**) Il calcolo è in tutti i casi approssimato, perchè il tempo di lettura di una scheda è noto semplicemente come valor medio: è tuttavia possibile maggiorare opportunamente il risultato, per garantire in tutte le circostanze l'avvenuto completamento.

debbono essere eseguite n istruzioni fra inizio e completamento, è necessario nel programma prevedere l'effettiva esecuzione di almeno n istruzioni per avere la garanzia che i dati letti siano effettivamente in memoria quando essi sono necessari. A parte l'ovvia difficoltà di calcolare quante saranno le istruzioni effettivamente eseguite in un programma (caso dei cicli che terminano quando è soddisfatta una certa condizione, per esempio!), sarà molto spesso necessario aggiungere delle istruzioni "inutili" che hanno il solo scopo di "far perdere tempo" al calcolatore.

Una soluzione migliore è quella di prevedere nel calcolatore un meccanismo esplicito, che permetta al programma di interrogare l'unità di lettura sull'avvenuto completamento dell'operazione: nel CANE, l'istruzione di controllo entrata (CEN) assolve questo scopo. L'istruzione:

```
CEN   J  I
```

funziona come un salto: se l'operazione di lettura non è ancora finita, viene eseguito un salto alla istruzione nella cella I+rX[J], e viene invece eseguita l'istruzione successiva se la lettura è finita. Un modo tipico di impiegare CEN:

```
      .............
      ENB   0   543   ------>  istruzione di lettura
      ____  _   ____ }
      ____  _   ____ }  un numero qualsiasi
      ............. }  di istruzioni
      ____  _   ____ }
231   CEN   0   231   ------>  il programma continua ad eseguire questa
                               istruzione fino al completamento della
      TRA   0   543   ------>  lettura
      .............            i dati sono sicuramente in memoria quando
                               questa istruzione viene eseguita
```

Se due istruzioni di entrata (ENB o ENA) si susseguono nel programma, anche senza che fra di esse vi sia alcuna istruzione CEN, il calcolatore attende, per iniziare la seconda istruzione, che la prima sia stata completata. Nel caso seguente:


```
      .............
074   ENA   0   543
075   ____  _   ____
076   ____  _   ____
077   ENB   1   111
      .............
```

il calcolatore, dopo aver eseguito l'istruzione in 076, attende per iniziare l'esecuzione dell'istruzione ENB in 077 che sia stata completata la precedente ENA.

Quindi, una istruzione di entrata che segue un'altra istruzione di entrata esegue automaticamente su quest'ultima il controllo di completamento.

4.8 - Il pulsante di lettura ed il caricatore minimo

In questo paragrafo illustreremo un dispositivo che permette al CANE di leggere un programma in memoria, e di iniziarne l'esecuzione. La consolle del nostro calcolatore è provvista di un pulsante, chiamato pulsante di lettura, o pulsante L: quando esso viene premuto a calcolatore fermo, viene letta una scheda binaria nelle 4 celle 000, 001, 002 e 003; completata la lettura, il calcolatore esegue l'istruzione nella cella 000 (*).

Supponiamo di voler leggere un programma di 12 istruzioni nelle celle:

004, 005,....., 017

e di voler iniziare l'esecuzione di questo programma dalla istruzione in 004. La prima operazione è quella di perforare 3 schede binarie con le 12 istruzioni, ed un'altra scheda binaria che contenga le 4 istruzioni seguenti:

```
TDX   1   764
ENB   1   020
ADDX  1   004
SLT   0   001
```

(*) Il pulsante L non ha alcun effetto se esso viene premuto mentre il calcolatore sta eseguendo un programma.

Quest'ultima scheda (ed il segmento di programma che vi è stato perforato) prende il nome di caricatore minimo. Il pacco formato dal caricatore minimo, seguito dalle 3 schede del programma da eseguire, viene messo nel lettore di schede, e viene premuto il pulsante L. Perciò, nelle prime 4 celle della memoria vengono memorizzate le 4 istruzioni del caricatore minimo:

```
000   TDX   1   764
001   ENB   1   020
002   ADDX  1   004
003   SLT   0   001
```

ed il calcolatore esegue l'istruzione nella cella 000. Un esame delle 4 istruzioni mostra che il loro scopo è quello di leggere 3 schede nelle celle:

004, 005,....., 017

Iniziata la lettura dell'ultima scheda, viene eseguita l'istruzione ADDX in 002, e dopo di essa: (rX1)=0; perciò, viene skippata l'istruzione in 003 ed eseguita l'istruzione in 004, che è appunto la prima del piccolo programma di 12 istruzioni:

```
000   TDX   1   764
001   ENB   1   020
002   ADDX  1   004 ---+
003   SLT   0   001    |
004   ---   -   ---    |
005   ---   -   ---    |
006   ---   -   ---    |
.....                  |
013   CEN   0   013    |
014   ---   -   ---    |
015   ---   -   ---    |
016   ---   -   ---    |
017   ---   -   ---    |
                       |
      programma di     |
      12 istruzioni    |
```

E' necessario che una delle prime 8 istruzioni del programma (per esempio, quella in 013) controlli il completamento della lettura dell'ultima

scheda, altrimenti vengono eseguite le istruzioni in 014 e seguenti senza che le istruzioni stesse siano ancora state lette dalla scheda. Si noti che è necessario controllare solamente la lettura dell'ultima scheda (perchè?).

L'operazione di leggere un programma in memoria prende il nome di caricamento del programma. Mediante il caricatore minimo (con una scelta opportuna dei campi indirizzo delle prime due istruzioni) è possibile caricare un programma qualsiasi di n istruzioni in n celle consecutive a partire dalla cella 004, iniziandone l'esecuzione (dopo il caricamento) dalla istruzione in 004. Infatti, se n è multiplo di 4, basterà impiegare il seguente caricatore minimo:

```
TDX   1   777-(n-1)
ENB   1   n+4
ADDX  1   004
SLT   0   001
```

Se, invece, n non è multiplo di 4, indichiamo con m l'intero più piccolo maggiore di n, che sia multiplo di 4. Il caricatore minimo:

```
TDX   1   777-(m-1)
ENB   1   m+4
ADDX  1   004
SLT   0   001
```

carica m/4 schede; l'ultima di queste porterà m-n=1,2 o 3 istruzioni, e rispettivamente 3, 2 o 1 campi "bianchi". L'effetto dell'esecuzione del caricatore minimo è quello di caricare n istruzioni nelle celle 004, 005,...,n+3, e di azzerare le m-n celle seguenti.

Nel programma da caricare è necessario prevedere esplicitamente il controllo della lettura dell'ultima scheda nel punto opportuno, e cioè prima di eseguire una delle istruzioni dell'ultima scheda, oppure una istruzione il cui corretto funzionamento sia subordinato in un modo qualsiasi alla presenza in memoria di una delle parole contenuta nell'ultima scheda.

Il caricamento di un programma mediante il caricatore minimo presenta alcuni inconvenienti:

a) non è possibile caricare un programma in una zona diversa da quella indicata;

b) il numero n di istruzioni da caricare deve essere noto; infatti, esso compare come una delle componenti degli indirizzi delle prime due istruzioni del caricatore minimo.

Vedremo nel seguito come sia opportuno, per garantire la necessaria elasticità nel caricamento dei programmi, impiegare dei programmi caricatori più flessibili. A loro volta, questi programmi saranno caricati in memoria dal caricatore minimo, che riveste quindi insieme al pulsante L una importanza fondamentale nella gestione del CANE; la struttura di un pacco per l'esecuzione di un programma P sarà quindi in generale la seguente:

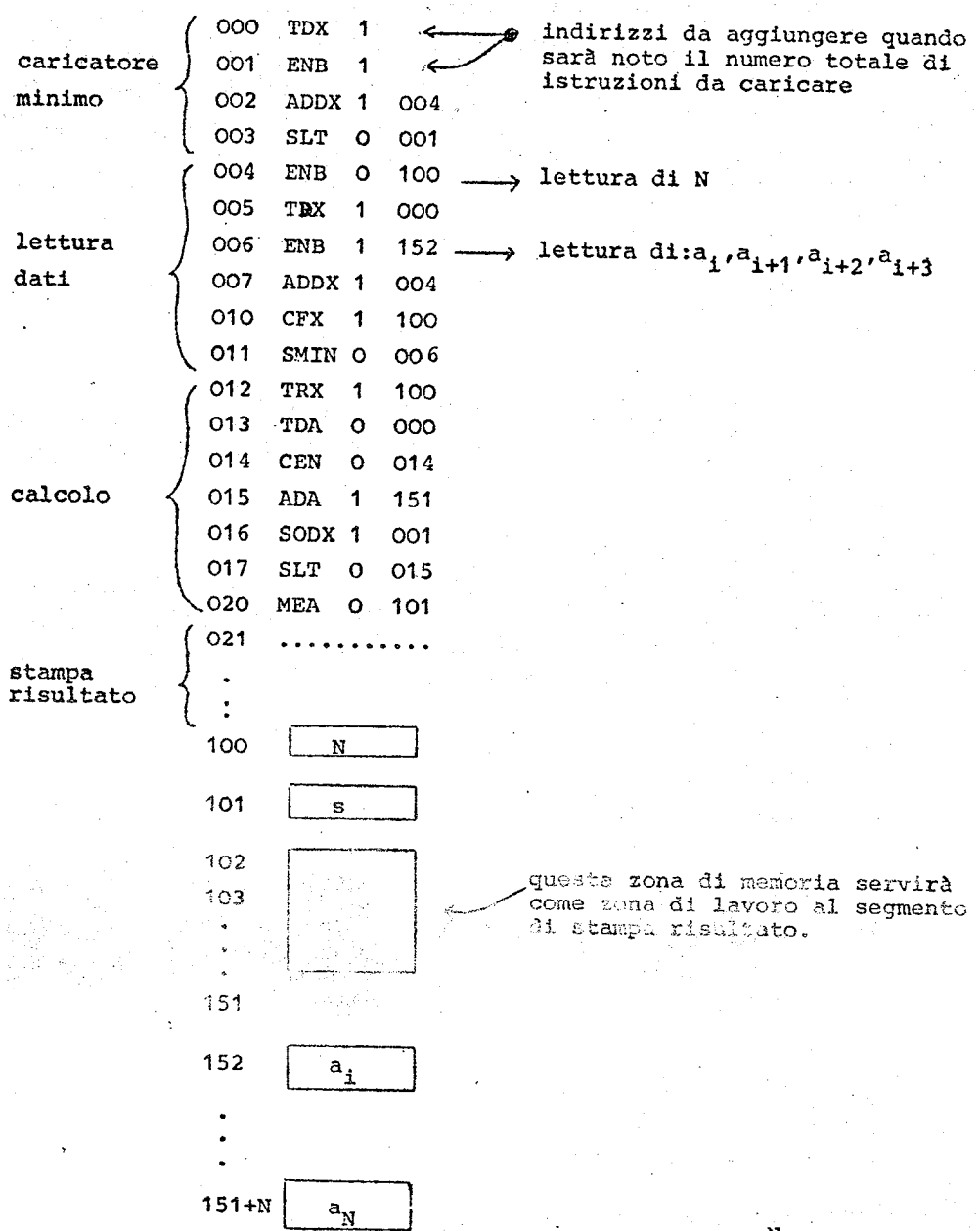
- 1 scheda contenente il caricatore minimo
- schede del programma caricatore
- schede del programma P
- schede contenenti i dati.

Il caricatore minimo e gli altri caricatori più generali operano nella modalità cosiddetta "load and go" (cioè, "carica e vai"), perchè l'esecuzione del programma inizia non appena il programma stesso è stato caricato.

Come esempio di impiego del caricatore minimo, consideriamo il solito programma per il calcolo di:

$$s = \sum_{i=1}^{N} a_i$$

Possiamo a questo punto scrivere anche il segmento di lettura dei dati ($N, a_1, a_2, \ldots, a_N$). La Tab. 4.6 riporta il programma (escluso il segmento di stampa del risultato), e il contenuto della zona di memoria destinata ai dati: si è supposto che i dati siano perforati in binario, N sulla prima scheda, e gli a_i quattro a quattro sulle schede seguenti.



Tab. 4.6 - Il solito programma per il calcolo di $s = \sum_{i=1}^{N} a_i$ cui è stato aggiunto il segmento di lettura; il programma è in una versione da caricare mediante il caricatore minimo.

4.9 - La stampante e l'istruzione di stampa

L'unità stampante del calcolatore CANE produce stampati (chiamati anche tabulati) come quello della fig. 1.4; ogni riga di stampa ha 120 posizioni.

Vi è una sola modalità di stampa, quella alfanumerica, e corrispondentemente una sola istruzione di stampa: l'istruzione di uscita (USC), che provoca la stampa di una riga di 120 caratteri. Prima di eseguire questa istruzione, occorre preparare l'informazione che costituisce la riga di stampa in una zona di 40 celle consecutive: la prima di queste celle conterrà le 3 configurazioni corrispondenti ai primi tre caratteri, la seconda le configurazioni corrispondenti al 4°, 5° e 6° carattere,..., la quarantesima quelle corrispondenti agli ultimi tre caratteri della riga.

L'esecuzione dell'istruzione:

```
USC  J  I
```

ha come effetto la stampa di una riga; i 120 caratteri da stampare sono ordinatamente memorizzati nelle celle di indirizzo I+rX[J], I+rX[J]+1,..., I+rX[J]+47.

Supponiamo che

```
Mw[555] = 606713 = 'bX='
Mw[556] = 604003 = 'b-3'
Mw[557] = 020760 = '27b'
Mw[560] = 736070 = ',bY'
Mw[561] = 136020 = '=b+'
Mw[562] = 041160 = '49b'
```

Mw[563] ... Mw[634] = 606060 = 'bbb'

L'esecuzione dell'istruzione

```
USC  0  555
```

provoca la stampa dell'ultima riga del tabulato nella fig. 4.9.

Le considerazioni svolte a proposito del controllo delle istruzioni

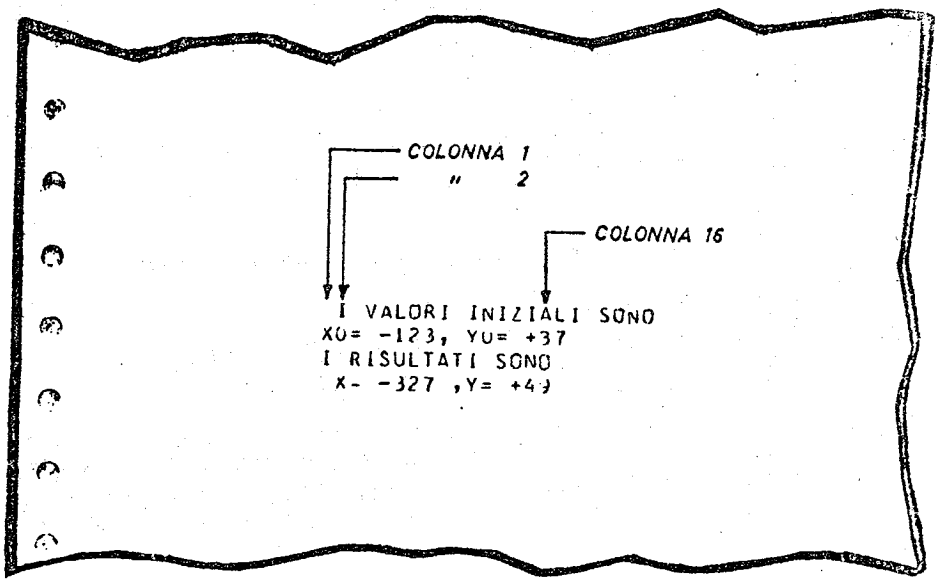
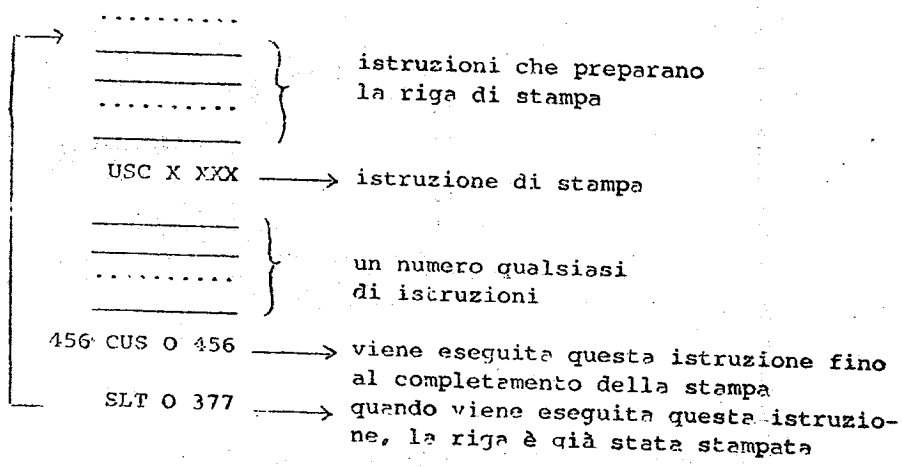
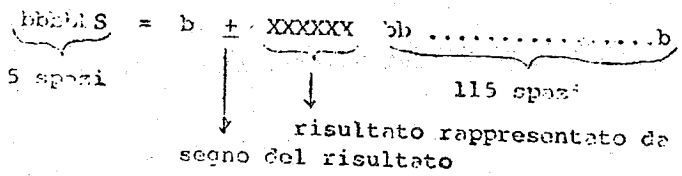


Fig. 4.9 - Esempio di impiego della istruzione USC.

di entrata valgono anche a proposito della istruzione di uscita; l'esecuzione di questa istruzione viene iniziata dal calcolatore, che prosegue poi nell'esecuzione delle istruzioni successive, affidando la responsabilità del completamento dell'istruzione al controllore periferico che risiede nell'unità di uscita. Il completamento della istruzione di uscita viene rivelato attraverso una istruzione di <u>controllo uscita</u> (CUS): se l'operazione di stampa non è ancora finita viene eseguito il salto, mentre se l'operazione di stampa è finita viene eseguita l'istruzione successiva. È necessario accertarsi dell'avvenuto completamento prima di riutilizzare la zona di 40 celle nella quale è stata memorizzata la riga di stampa (per memorizzare la riga successiva, o per qualsiasi altro scopo). Un modo tipico di impiegare CUS:



Come esempio di preparazione di una riga di stampa, ritorniamo al programma della Tab. 4.6, ricordando che avevamo riservato le 40 celle di indirizzo 102.103,...,151 per il segmento di stampa. Supponiamo di voler stampare il risultato nel formato seguente:



	020	.....			
imbiancatura zona di stampa	021	TDX	1	000	Mw[102], Mw[103],...,Mw[151] ← 'b
	022	TDX	2	050	
	023	MCA	2	101	
	024	SODX	2	001	
	025	SLT	0	023	
preparazione della riga di stampa	026	TDX	1	003	Mw[103] ← 'bbS'
	027	TDA	0	062	
	030	MCA	0	103	
	031	TDA	0	000	Mw[104] ← 'bb±'
	032	CFA	0	101	
	033	SMIN	0	037	
	034	TDA	0	040	
	035	SLT	0	037	
	036	TDA	0	020	
	037	MCA	0	104	
	040	TDX	1	001	Mw[104] ← '=b±'
	041	TDA	0	013	
	042	MCA	0	104	
	043	CAR	0	100	risultato (in cifre decimali) in 105 e 106
	044	MEA	0	105	
	045	MEB	0	106	
	046	USC	0	102	
	047	ALT	0	000	
	⋮	⋮			
	101	s			
	102	zona di stampa			
	151				
	152	.....			

Tab. 4.7 - Segmento di stampa del programma della Tab. 4.6

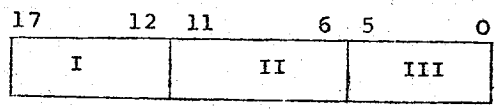
Il segmento di stampa è riportato nella Tab. 4.7. notiamo nel segmento due istruzioni (MCA e CAR) che non sono ancora state illustrate. Prima di interpretare la Tab. 4.7, è quindi opportuno che il lettore studi il significato delle nuove istruzioni nel paragrafo seguente.

4.10 - Le istruzioni per la manipolazione di caratteri

Queste istruzioni riguardano il movimento di caratteri fra il registro rA e le celle di memoria (MCA e TCA), e la trasformazione di rappresentazione numeriche in caratteri e viceversa (CAR e NUM).

L'effetto delle istruzioni di memorizzazione carattere (MCA) e di trasferimento carattere (TCA) dipende dal contenuto delle due posizioni binarie meno significative del registro indice rX[1]; rX[1]<1:0>.

Inoltre, indichiamo con gli indici I,II e III rispettivamente le prime 6 posizioni dalla sinistra (bit 12-17), le seconde 6 (bit 6-11), e le ultime 6 (bit 0-5) di una cella di memoria o del registro rA.



rA o cella di memoria

Per esempio, rA_{III} indica rA<5:0>,cioè il campo 0-5 del registro rA (terzo carattere di rA), e $Mw[222]_{II}$ indica Mw[222]<11:6>,cioè il campo 6-11 della cella 222 (secondo carattere di 222).

L'istruzione di memorizzazione carattere (MCA) memorizza il terzo carattere di rA nel primo carattere della cella I+rX[J] se rX[1]<1:0> =01, nel secondo carattere se rX[1]<1:0>=10, e nel terzo carattere se rX[1]<1:0> =11: vedi la fig. 4.10 a). Conversamente (vedi la fig. 4.10 b)), l'istruzione di trasferimento carattere (TCA), dopo aver azzerato il registro rA, trasferisce nel terzo carattere di questo registro rispettivamente il primo, secondo o terzo carattere della cella I+rX[J], a seconda che rX[1]<1:0> = 01,10,11.

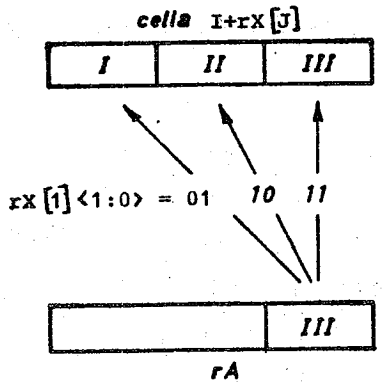
Se viene eseguita MCA con rX[1]<1:0> =00, la cella I+rX[J] viene riempita di(tre) spazi; invece, l'esecuzione di TCA con rX[1]<1:0> =00 non ha alcun effetto.

Per illustrare queste due istruzioni, consideriamo il seguente segmento:

```
            .............
            TDX   1   000
            TDX   2   100
            TDX   2   100      Mw[100], Mw[101],...,Mw[177] ← 'bbb'
      550   MCA   2   000      (imbiancatura delle celle
            ADDX  2   001      100,101,...,177)
            CFDX  2   200
            SMIN  0   550
            TDX   1   002
            TDX   2   100      Mw[100+h]II ← Mw[200+h]II
      556   TCA   2   100      per 0 ≤ h ≤ 77
            MCA   2   000
            ADDX  2   001
            CFXD  2   200
            SMIN  0   556
            .............
```

Lo scopo delle 6 istruzioni è quello di riempire di spazi la zona di memoria dalla cella 100 alla cella 177; per brevità, questa operazione verrà chiamata imbiancatura (vedi nella Tab.4.7, l'imbiancatura della zona di stampà).Nelle rimanenti istruzioni, il secondo carattere della cella 200+h viene ricopiato nel secondo carattere della cella 100+h.

Tipicamente, l'istruzione MCA viene impiegata nella "composizione" di una riga di stampa: il carattere opportuno viene generato nel registro rA mediante una TDA, e memorizzato poi nella posizione opportuna mediante una MCA.



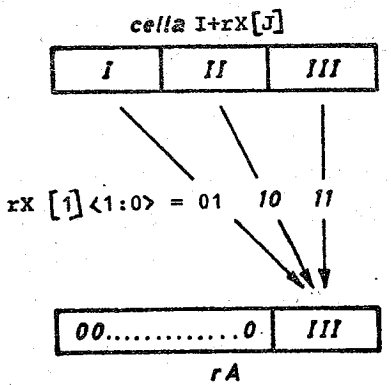


Fig. 4.10 - Funzionamento delle istruzioni: a) MCA, b) TCA.

Per esempio:

```
.............
TDX   1   001
TDA   0   021
MCA   0   300
ADDX  1   001
TDA   0   022   } Mw[300] ←— 'ABC'
MCA   0   300
ADDX  1   001
TDA   0   023
MCA   0   300
.............
```

ha l'effetto di memorizzare la sequenza di caratteri 'ABC' nella cella 300.

Per esaminare un impiego tipico della TCA, dobbiamo introdurre una nuova istruzione: lo spostamento logico di rA verso sinistra (ALS), che appartiene alla famiglia delle istruzioni di spostamento, su cui torneremo nel seguito. Nell'istruzione ALS, il contenuto del registro rA viene spostato di I+rX[J] posti verso sinistra: i primi I+rX[J] bit del contenuto di rA (da destra) vengono definitivamente perduti, e le ultime I+rX[J] posizioni di rA vengono riempite con 0.

Se, per esempio:

rA = 0101000111111001111

rX[6] = 004

e viene eseguita l'istruzione:

ALS 6 001

dopo l'esecuzione:

rA = 0011111100111100000

Supponiamo che la cella 200 contenga la rappresentazione, in caratteri alfanumerici, di un numero ottale di 3 cifre (per esempio, Mw[200]='456'); questa informazione può essere stata memorizzata in 200 da una istruzione ENA. Vogliamo convertire questo numero nella sua rappresentazione numerica CANE, e memorizzarlo nella parte meno significativa (ultime 9 posizioni) della cella 300. Il segmento di programma si presenta così:

```
.............
TDX   1   001
TCA   0   200
ALS   0   006
MEA   0   300
ADDX  1   001
TCA   0   200
ALS   0   003
ADA   0   300
MEA   0   300
ADDX  1   001
TCA   0   200
ADA   0   300
MEA   0   300
.............
```

Per interpretare il precedente segmento, si consideri come sono rappresentate nel codice caratteri le cifre 0,1,...,7.

Le ultime due istruzioni che illustreremo in questo paragrafo sono la conversione in caratteri (CAR), e la conversione in rappresentazione numerica (NUM).

Queste istruzioni vengono impiegate nella conversione dal codice caratteri alla rappresentazione numerica che segue l'operazione di lettura (NUM), o nella conversione inversa che precede l'operazione di stampa (CAR).

La conversione in caratteri (CAR) converte il valore assoluto del numero nella cella I+rX[J] nella sua rappresentazione decimale in caratteri (6 cifre): le tre cifre più significative si trovano, ad istruzione eseguita, in rA, e le tre cifre meno significative in rB.
Se,per esempio:

$$Mw[246] = 640721 = -137057_{(8)} = -48687_{(10)}$$

dopo l'esecuzione dell'istruzione

```
CAR  0  246
```

avremo (vedi fig. 4.11 a)):

rA = 000408 , rB = 060807

Conversamente, l'istruzione di conversione in rappresentazione numerica (NUM) converte un numero (positivo) di sei cifre decimali, rappresentate in codice caratteri nel registro rA (le tre più significative) e nel registro rB (le tre meno significative) nella corrispondente rappresentazione binaria, che viene memorizzata nella cella I+rX[J] Se nella conversione si origina un supero di capacità (si ricordi, infatti, che il numero decimale in rA ed rB è compreso nell'intervallo: $0 \leq D \leq 999999$) viene settato l'indicatore di supero iSP, e dopo l'esecuzione dell'istruzione la cella I+rX[J] contiene i 17 bit meno significativi della rappresentazione binaria (il primo bit è sicuramente 0, poichè il numero convertito è positivo).
Se, per esempio:

rA = 000004 , rB = 000208

dopo l'esecuzione dell'istruzione

```
NUM  0  725
```

avremo (vedi fig. 4.11 b)):

$$Mw[725] = 007674 = +7674_{(8)} = +4028_{(10)}$$

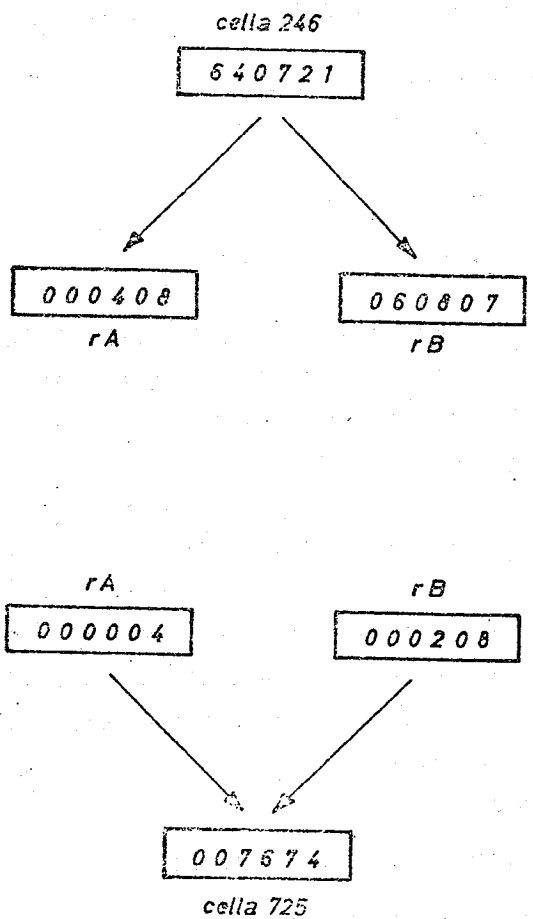


Fig. 4.11 - Funzionamento delle istruzioni: a) CAR, b) NUM.

Se invece:

rA = 020606 , rB = 010702

l'esecuzione della stessa istruzione lascia ancora

Mw[725] = 007674, ed: iSP = 1.

Il lettore, a questo punto, è in grado di esaminare il segmento di stampa della Tab. 4.7; le prime 5 istruzioni di questo programma imbiancano la zona nella quale le istruzioni successive comporranno la riga di stampa.
L'ultima istruzione del programma (ALT O 000) ferma il calcolatore.