# HOME COMPUTER AMICO 2000

*Questo articolo è totalmente dedicato al software, saranno dati cioè gli strumenti per poter lavorare maggiormente con la scheda AMICO 2000A con l'apprendimento di nuove importanti istruzioni del microprocessore 6502. Cominceremo con l'esaminare un'altra importante operazione matematica: la sottrazione. Saranno poi visti nel dettaglio gli altri bit del registro di Status e i salti condizionati determinati proprio dal valore di questi bit. Sarà esaminato poi in dettaglio il concetto di flow chart, i suoi simboli, la sua costruzione. In questo articolo viene presentata e spiegata la tabella delle istruzioni del microprocessore che costituisce una guida fondamentale per la creazione di programmi da far girare sul sistema AMICO 2000A.*

a cura della A.S.E.I. s.r.l. - parte sesta

## LA SOTTRAZIONE

### I numeri negativi

Tutte le operazioni fatte fino a questo momento hanno preso in considerazione solo numeri positivi; le nostre operazioni infatti hanno utilizzazo cifre dalla 00 alla FF (numeri di 8 bit) senza che ci fosse ombra di segno. Una domanda che viene spontanea è: come si possono rappresentare i numeri con il segno?

Il sistema che esamineremo per rispondere a questa domanda si chiama COMPLEMENTO A DUE ed è universalmente usato.

I numeri che prendiamo in considerazione sono sempre di otto bit, ma nell'ambito di questi otto bit introdurremo anche il segno. Come si fa?

È molto semplice, basta prendere atto di questa convenzione:
i numeri che hanno il 1° bit (quello più significativo) uguale a 1 sono NEGATIVI; i numeri che hanno il 1° bit uguale a 0 sono POSITIVI.

Per chiarire le idee si osservi la fig. 1.

Bisogna prestare attenzione a questo particolare: in un numero formato da otto bit, sette di questi rappresentano il numero vero e proprio in valore assoluto, mentre l'ottavo - il più significativo - rappresenterà il segno.

Abbiamo allora che con 7 bit possiamo rappresentare $2^7 = 128$ numeri diversi; considerando ora sia quelli positivi che quelli negativi essi raddoppiano e diventano 256, ritroviamo cioè il valore $2^8 = 256$ che già conosciamo per i numeri da 8 bit.

Secondo la convenzione suddetta diciamo dunque che i numeri FF (1111 1111), 82 (1000 0010), A0 (1010 0000), e 95 (1001 0101) sono numeri **negativi** (il primo bit è a 1) mentre i numeri 00 (0000 0000), 14 (0001 0100), 7E (0111 1110) e 39 (0011 1001) sono numeri **positivi** (il primo bit è a 0).

Facciamo notare che in questa convenzione il numero 0 è considerato come numero positivo.

Il problema immediatamente successivo da risolvere è il cambiamento di segno di un numero positivo e viceversa.

Per esempio il nostro numero 1 sappiamo che in esadecimale si scrive 01; il nostro -1 allora come si scrive in esadecimale con numeri da 8 bit?

La regola generale dice che bisogna prendere il numero positivo, negarlo bit per bit, quindi sommare 1. Cioè in pratica riferendoci alla fig. 2 abbiamo che secondo questa regola:

+1 = 01
−1 = FF

Attenzione, per evitare di farvi confondere le idee vi rammentiamo che se consideriamo il numero FF in valore assoluto (come abbiamo fatto fino ad ora), esso è il numero più grande da 8 bit e vale $255_{10}$; nella convenzione di complemento a due però, nella quale il primo bit è quello di segno, FF è il numero negativo −1.

Sembrerebbe ora che col nostro microcomputer si possano sommare e/o sottrarre numeri non superiori a +127 e non inferiori a −128. Questo avviene solo se lavoriamo con numeri contenuti in una sola locazione di memoria (8 bit); naturalmente nella matematica che ci costruiremo i numeri potranno essere contenuti in più locazioni di memoria:

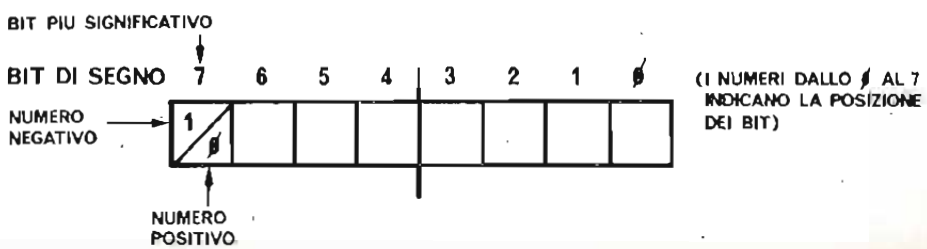
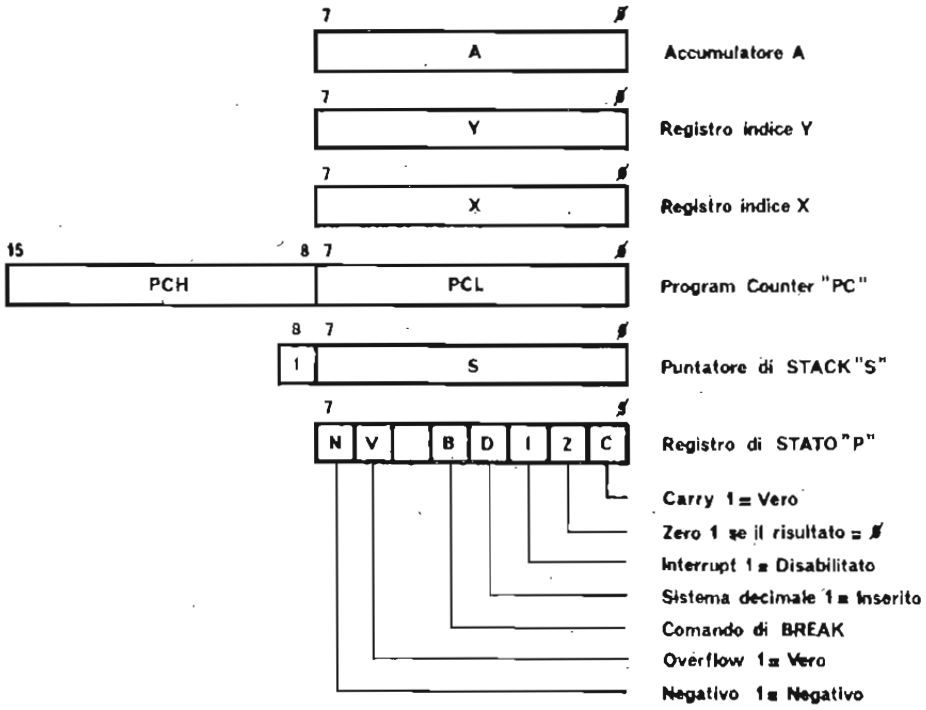



*Fig. 1 - Segno di un numero di 8 bit (di cui sette costituiscono il valore assoluto del numero e uno, il più significativo, il segno.*

## SCHEMATIZZAZIONE DELLA CPU 6502




per esempio tre successive.

Riprendiamo ora il discorso del numero positivo e negativo e proviamo ad eseguire la somma:

12 + (−1).

Eseguiamola prima sulla carta:

```
  0001 0010  +     (12)
  1111 1111  =     (FF)
1 0001 0001        (11)
↑
Carry
```

Come risultato della operazione otteniamo 11. Il Carry esce dalla operazione a 1, ma in questo caso non se ne tiene conto.

In questo modo abbiamo sottratto 1 al numero 12.

Eseguiamo ora la stessa operazione sull'AMICO 2000A con questo semplice programma:

```
0200  18   CLC
   1  D8   CLD
   2  A9   LDA #$12      Carico 12 in accumulatore in maniera immediata
   3  12
   4  69   ADC #$FF      Sommo FF in maniera immediata
   5  FF
   6  85   STA 00        Porto il risultato nella locazione di memoria 00
   7  00
   8  4C   JMP MONITOR   Torno al monitor. Stop.
   9  22
   A  FE
```

Eseguendo il programma verrà immediatamente visualizzato il risultato.

Modifichiamo il programma per rendere più agevole il cambiamento dei dati:

```
0200  18   CLC
   1  D8   CLD
   2  A5   LDA 00        Carico il dato della locazione 00 in accumulatore
   3  00
   4  65   ADC 01        Sommo il contenuto della locazione 01
   5  01
   6  85   STA 00        Riporto il risultato nella locazione 00
   7  00
   8  4C   JMP MONITOR
   9  22
   A  FE
```

I dati vanno introdotti nelle locazioni 00 e 01. Scriviamo allora il programma, cari-

chiamo il Program Counter per utilizzare il tasto REG (alle locazioni 00F6 e 00F7 in modo che richiami l'indirizzo 0200 poi premiamo i tasti nella sequenza

RES | DA | 12 | ↑ | FF

REG | RUN |.

Il risultato (11) compare immediatamente nel display dati.

Per cambiare i dati di partenza basta introdurli alle locazioni 00 e 01, quindi far partire il programma.

A questo punto vi insegnamo un piccolo programma che permette di calcolare il numero negativo una volta dato il numero positivo. Attenzione! Se volete non guardate ciò che segue e provate a farlo voi; con un po' di pazienza dovreste riuscire poichè avete tutte le cognizioni richieste.

Ad ogni modo il programma è il seguente:

```
0300   18    CLC
   1   D8    CLD
   2   A5    LDA 00          Prelevo il dato da trasformare dalla locazione 00
   3   00
   4   49    EOR #$FF        Nego il numero contenuto in accumulatore
   5   FF
   6   69    ADC#1           Sommo 1
   7   01
   8   85    STA 00          Riporto il dato nella locazione 00
   9   00
   A   4C    JMP MONITOR. Stop.
   B   22
   C   FE
```

In sostanza questo programma esegue la procedura di calcolo prima indicata: negazione del numero e somma con 1 trascurando il Carry.

Per usare il programma procediamo come al solito introducendo nella locazione di memoria 00 il numero del quale si vuole calcolare il negativo per ottenere nella stessa locazione 00 il risultato cercato.

Vi diamo alcuni numeri con il loro corrispondente negativo:

| (p) | (n) | |
|---|---|---|
| 00 | 00 | (il negativo di 0 è ancora 0) |
| 01 | FF | |
| 0E | F2 | |
| 47 | B9 | |
| 7F | 81 | |

Notate che nell'ambito dei numeri da 8 bit non esiste un numero positivo che abbia come numero negativo 80.

Questo perchè i numeri positivi vanno, in decimale, da $0_{10}$ a $+127_{10}$, quelli negativi da $-1_{10}$ a $-128_{10}$, in tutto 256 numeri (compreso lo zero).

le particolarità da tener presenti sono dunque:
con 8 bit si possono rappresentare
128 numeri positivi 0 ÷ 127
128 numeri negativi −1 ÷ −128.

Il numero negativo corrispondente a 0 è ancora 0.

Il programma appena illustrato è ovviamente in grado di trasformare un numero negativo nel suo corrispondente positivo. Verificate i seguenti risultati:

| (n) | (p) |
|---|---|
| B9 | 47 |
| C4 | 3C |
| 98 | 68 |
| 8F | 71 |

## Sottrazione tramite i numeri negativi

A questo punto è evidente che la differenza fra due numeri si può sempre riportare alla somma fra due numeri.

Infatti 13 − 7A = 13 + (−7A) = 13 + + 86 = 99 = -67.

Tenete sempre presente che si tratta di numeri esadecimali.

Verificate queste operazioni con i programmi appena scritti.

Può essere utile spesso poter disporre di un mezzo che ci permetta, senza fare fatica, di trasformare un numero negativo o positivo del sistema decimale in quello corrispondente nel sistema complemento a due.

Per far questo abbiamo scritto un programma di cui troverete a fine articolo la lista delle istruzioni (codice oggetto).

Il programma parte dalla locazione 0200 e arriva alla 02D3. Vi consigliamo di registrarlo su cassetta in modo da poterlo utilizzare ogni volta che ne avete bisogno. Attenzione però, questo programma non è rilocabile, cioè deve essere sempre scritto a partire dalla locazione di memoria 0200.

Per utilizzarlo, una volta scritto, è molto semplice: premete i tasti AD 0200 RUN.

Premete il tasto C e sul display indirizzi compariranno tre zeri. Scrivete ora il numero che intendete trasformare, premete il tasto F se volete farlo diventare negativo (e viceversa): contemporaneamente sul display dati compare il risultato. Da notare che questo programma è stato scritto per evitare che commettiate errori: tutti i numeri fuori della portata -128÷+127 non vengono accettati. Oltre a questo programma riportiamo nella tabella 1 una tavola di conversioni da numeri negativi (sopra) e positivi (sotto) in sistema decimale a numeri negativi e positivi in sistema complemento a due (e viceversa).

Usare queste tavole è molto semplice, facciamo un esempio.

Vogliamo sapere a cosa corrisponde il numero −77 (in decimale) nel sistema complemento a due.

Cerchiamo 77 all'interno della tavola in alto (numeri negativi), sulla riga del 77 a sinistra troviamo il numero B sulla colonna "1ª cifra"; sulla colonna del 77 troviamo in alto il numero 3 sulla riga "2ª cifra": la traduzione sarà quindi B3. Procedete all'inverso per trasformare un numero da complemento a due a decimale.

## L'Overflow

Analizziamo ora un'altro bit dello status, quello di *overflow*. Il concetto di overflow nasce dall'introduzione dei numeri con segno appena fatta. Infatti abbiamo sempre detto che la somma di due numeri può generare un riporto (Carry) se il risultato eccede la dimensione degli otto bit che abbiamo a disosizione.

Nel caso di numeri col segno però abbiamo solo 7 bit per il numero, dato che il bit più significativo è stato utilizzato per il segno. Può succedere che sommando due numeri negativi si ottenga un numero positivo o sommando due numeri positivi si ottenga un numero negativo; questo perchè il risultato eccede

```
0 0 0 0   0 0 0 1      +1
1 1 1 1   1 1 1 0      (+1)   Il numero precedente viene negato bit per bit, cioè dove c'era uno
                              "zero", si scrive un "uno" e dove
                +             c'era un "uno", si scrive uno
                              "zero".
0 0 0 0   0 0 0 1  =   +1     Si somma 1 al risultato della
1 1 1 1   1 1 1 1              negazione.
                               Risultato FF
```

*Fig. 2 - Trasformazione del numero 01 positivo in negativo (−1) nel sistema complemento a due: il isultato è FF.*

i 7 bit andando ad influenzare l'ottavo bit, cioè quello di segno.

Chiariamo subito le idee con qualche esempio:

```
1111 0111 +   (F7 = -09)
1111 1110 =   (FE = -02)
-------------------------
1  1111 0101     (F5 = -0B)
Carry
```

Nel risultato il bit di disegno (il primo a sinistra) è 1 e il numero è quindi negativo, infatti esso non eccede il limite dei 7 bit.

Vediamo ora la somma di due numeri negativi più grandi:

```
1001 0110 +   (96 = -6A)
1000 1111 =   (8F = -71)
-------------------------
1  0010 0101   (25 = +25)
Carry
```

Il risultato è un numero che ha *cambiato segno*. Il bit di segno infatti è 0, si è ottenuto cioè un numero positivo sommando due numeri negativi!

Operando in matematica con complemento a due bisogna tener conto di questi cambiamenti di segno per evitare errori. Si può dimostrare che, sviluppando una funzione di questo genere:

$$V = A_7 \cdot B_7 \cdot \overline{R_7} + \overline{A_7} \cdot \overline{B_7} \cdot R_7$$

in cui:

$A_7$ è il bit più significativo del primo addendo; $B_7$ è il bit più significativo del secondo addendo; $R_7$ è il bit più significativo del risultato. La barretta sopra il segno del bit sta per negato; Il · è il simbolo della operazione di AND; Il + è il simbolo della operazione di OR.

- se V = 0 non ci sono stati errori nella operazione.
- se V = 1 c'è stato un cambiamento di segno nell'operazione.

Verifichiamo V per i due esempi appena fatti.

1° esempio: $A_7 = 1. B_7 = 1. R_7 = 1$

$$V = 1 \cdot 1 \cdot 0 + 0 \cdot 0 \cdot 1 = 0.$$

Il risultato è dunque esatto.

2° esempio: $A_7 = 1, B_7 = 1, R_7 = 0$

$$V = 1 \cdot 1 \cdot 1 + 0 \cdot 0 \cdot 0 = 1$$

C'è un cambiamento di segno nel risultato. V si chiama "bit di overflow" e viene calcolato automaticamente dalla CPU ad ogni operazione che lo condiziona.

In generale, in forma molto semplice possiamo dire che la CPU pone V = 1 se il risultato dell'operazione eseguita esce dai limiti permessi per un numero con segno da 8 bit (7 di valore assoluto e 1 di segno) cioè se è al di fuori del campo -128 ÷ +127.

## L'istruzione SBC

Questa istruzione esegue la differenza (sempre operando nella convenzione dei numeri negativi ovvero del complemento a due) fra il contenuto dell'accumulatore, il contenuto di una locazione di memoria e il negato del bit di Carry. L'operazione in forma simbolica si scrive:

$$A - M - \overline{C} \rightarrow A$$

Cioè la differenza fra l'accumulatore, il dato in memoria, il negato del Carry, va a finire nell'accumulatore. Questa istruzione può operare sia in sistema decimale che esadecimale, a seconda se la CPU sta lavorando in maniera decimale o meno. $\overline{C}$ (negato del bit di Carry) si chiama in inglese *borrow* ed è il riporto dell'operazione. La presenza del $\overline{C}$ è un artificio del microprocessore per poter calcolare in codice binario complemento a due.

Tenete presente una regola molto importante: quando eseguite delle somme dovete sempre mettere, prima dell'esecuzione della operazione di somma, il C = 0.

Quando fate delle sottrazioni tramite l'istruzione SBC dovete sempre mettere, prima dell'esecuzione della differenza, il C = 1.

Ci sono ancora delle particolarità molto importanti da tener presenti: l'istruzione SBC mette a 1 il bit di Carry se il risultato dell'operazione è maggiore di 0; mette a 0 il bit di Carry se il risultato dell'operazione è minore di 0.

Il bit di overflow viene messo a 1 se il risultato dell'operazione non è compreso fra + 127 e -128, altrimenti viene messo a 0.

Eseguiamo ora l'operazione di calcolo di numero negativo già descritta in precedenza utilizzando questa volta l'istruzione SBC. (Posizionatevi su una locazione di memoria a piacere per cominciare a scrivere il programma).

```
D8  CLD
38  SEC (notate che mettiamo il Carry
                                  a 1)
A9  LDA # 00 Carico il numero 0 nel-
00                  l'accumulatore
E5  SBC $00 Sottraggo il dato da com-
00                        plementare
85  STA $00 Metto il risultato in loca-
00                          zione 00
4C  JMP MONITOR, Stop
22
FE
```

Abbiamo fatto un programma che calcola

00 - NUMERO = - NUMERO

Il numro (dato di ingresso) viene prelevato all'inizio dell'operazione dalla locazione dalla locazione di memoria 00 e alla fine viene ancora depositato nella locazione 00 con il segno cambiato.

Verificate con questo semplice programma le operazioni di calcolo precedentemente eseguite verificandone la corrispodenza.

## STATUS

I bit che fanno parte del registro di status sono riportati tutti nella tabella delle istruzioni del 6502; abbiamo già

visto alcuni di questi bit, comunque di seguito li riesaminiamo tutti insieme.

C = Carry
– Bit di Carry. È il riporto in varie operazioni, per esempio in quella di somma.

Z = Zero
– Questo bit dello stato viene messo a 1 se il risultato dell'ultima operazione eseguita è uno zero. Molte istruzioni modificano questo bit. Per esempio quella di Somma, Load, quelle logiche di AND, OR, EOR.

I = Richiesta di interrupt
– Se questo bit è a zero risulta abilitato l'interrupt in ingresso alla CPU. In un capitolo dedicato esamineremo questo aspetto dell'hardware.

D = Modo di funzionamento decimale
– Se questo bit è a uno la CPU sta operando in matematica decimale. Questo bit può essere posizionato a zero o a uno tramite due apposite istruzioni (CLD - SED).

B = Comando di BRK
– Questo comando sarà esaminato in seguito.

V = Bit di overflow
– È il bit esaminato nelle operazioni differenza. Molte operazioni lo condizionano.

N = Bit di negativo
– Viene messo a uno se il risultato dell'ultima operazione eseguita è negativo. In pratica è il bit più significativo del risultato dell'ultima operazione eseguita. Viene condizionato da molte operazioni sia aritmetiche che logiche.

I bit di status I e B sono legati essenzialmente alla circuiteria esterna al microelaboratore e non li esamineremo finché non avremo a che fare con questi dispositivi.

In questo momento i bit di status che maggiormente ci interessano sono quattro, e cioè: C - Z - V - N.

Questi bit in generale vengono condizionati automaticamente dalla istruzione che stiamo eseguendo e ci permettono, eseguita la istruzione, di prendere delle decisioni in base al risultato della operazione eseguita.

Sono proprio queste decisioni che rendono l'elaboratore "intelligente".

Le istruzioni che prendono una decisione in base ai risultati delle operazioni, sono le istruzioni di BRANCH (salto condizionato).

## Le istruzioni di BRANCH

È una categoria di istruzioni molto importante, che permette di eseguire o non eseguire un salto da un punto all'altro del programma in base al risultato di una operazione precedente.

Queste istruzioni sono:

| | | |
|---|---|---|
| BCC il salto viene eseguito se il bit di Carry | è = 0: codice op. | 90 |
| BCS il salto viene eseguito se il bit di Carry | è = 1: codice op. | B0 |
| BEQ il salto viene eseguito se il bit di Zero | è = 1: codice op. | F0 |
| BMI il salto viene eseguito se il bit di Negativo | è = 1: codice op. | 30 |
| BNE il salto viene eseguito se il bit di Zero | è = 0: codice op. | D0 |
| BPL il salto viene eseguito se il bit di Negativo | è = 0: codice op. | 10 |
| BVC il salto viene eseguito se il bit di Overflow | è = 0: codice op. | 50 |
| BVS il salto viene eseguito se il bit di Overflow | è = 1: codice op. | 70 |

Vediamo ora quale è il significato di alcune di queste istruzioni indipendentemente dai bit di status, ovvero come se non dovessimo tener conto del valore di essi.

Ci spieghiamo meglio.

Spesso le istruzioni di Branch vengono utilizzate subito dopo le istruzioni di *comparazione* ("compare" in inglese) come CMP, CPX (che vedremo fra poco), CPY; di *decremento* come DEC, DEX (che vedremo fra poco), DEY o di incremento come INC, INX (che vedremo fra poco) e INY.

Tutte queste istruzioni eseguono una certa operazione: nella comparazione, ad esempio, la CPU esegue una sottrazione fra i due dati da comparare dando un certo risultato che influenza i bit di Status interessati (si veda la tabella riassuntiva delle istruzioni del 6502). In pratica quando i due dati che vengono comparati sono uguali il risultato di questa sottrazione è 0, ma noi non lo vediamo. Questo risultato però mette a 1 il bit Z dello Status.

Premesso ciò vediamo il significato per esteso delle seguenti istruzioni:

| | |
|---|---|
| BEQ | Salta se il risultato (della operazione precedente) è = 0 |
| BMI | Salta se il risultato (della operazione precedente) è < 0 (negativo) |
| BNE | Salta se il risultato (della operazione precedente) è ≠ 0 |
| BPL | Salta se il risultato (della operazione precedente) è ≧ 0 (positivo) |

Ricordiamo che 0 nel sistema di numerazione complemento a due è un numero positivo.

Visto questo, se a una comparazione facciamo seguire la istruzione BEQ otteniamo che verrà eseguito un salto quando i due dati messi a confronto saranno uguali (la differenza fa 0, quindi va a 1 il bit di Status Z, quindi BEQ esegue il salto).

Per riassumere diremo allora che se qualsiasi operazione eseguita immediatamente prima di BEQ dà come risultato 0 (potrà essere una somma, una differenza, una operazione logica tipo AND, OR, ecc.), verrà eseguito il salto.

Lo stesso ragionamento vale per le altre istruzioni: BMI ad esempio eseguirà un salto se il risultato di una operazione immediatamente precedente è negativo (cioè se il bit più significativo del risultato è = 1, cioè quando il bit di Status N = 1).

Una ultima nota sulla istruzione BVC: questa può essere usata per corregere un errore in un operazione di sottrazione.

Facciamo notare che tutte le istruzioni di salto (Branch) sono formate sempre da due byte di cui il primo è il codice operativo (es. 90), il secondo rappresenta l'entità del salto, ovvero il numero di byte che si devono saltare nell'esecuzione del programma. Quest'ultimo numero è espresso in complemento a due, cioè il salto rispetto alla posizione in cui il Program Counter è in quel momento, può essere negativo (numero di byte indietro) o positivo (numero di byte in avanti). A ben comprendere e saper utilizzare le diverse istruzioni di Branch si imparerà man mano che verranno presentati esempi di programma. L'importante per ora è che impariate a memoria le condizioni che provocano il salto per ogni istruzione di Branch.

Vediamo subito di utilizzare alcune di queste istruzioni per costruire un programma che analizza un numero n.

Se n ≧ 0 il microelaboratore dovrà scrivere 00 nella locazione di memoria 0000;

se n < 0 si scrive FF nella locazione di memoria 0000.

Questo programma può essere scritto sotto forma di svolgimento logico o diagramma di flusso come rappresentato in Fig. 3.

Vediamo in questa figura due simboli nuovi, il rettangolo e il rombo, usati rispettivamente per indicare una operazione da eseguire e una domanda che ci si pone. Più avanti si analizzerà in dettaglio questo tipo di rappresentazione grafica.

Scriviamo ora il programma proposto. Il numero di cui vogliamo analizzare il segno sia contenuto nella locazione di memoria 0001.

```
0320    A5   LDA 01
   1    01
   2    10   BPL AVA1
   3    07
   4    A9   LDA #$FF
   5    FF
```

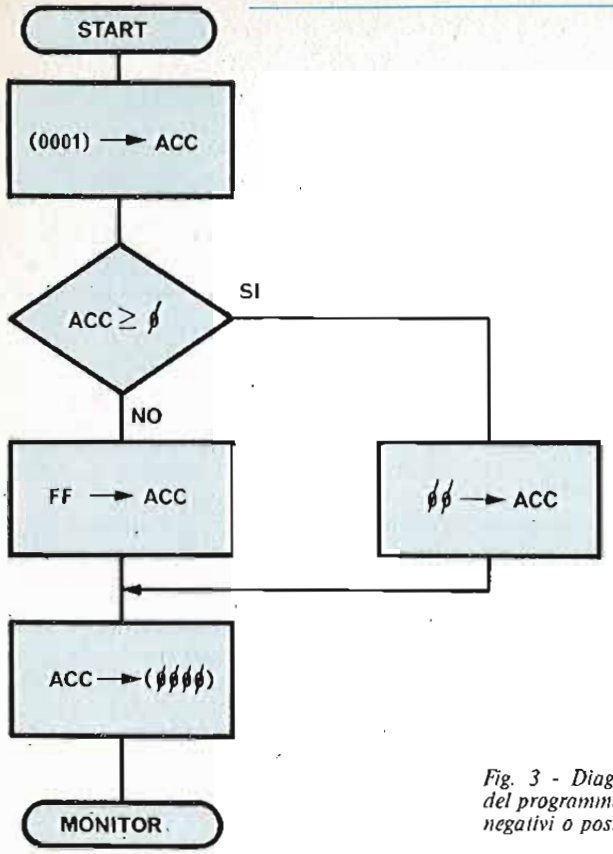



*Fig. 3 - Diagramma di flusso (o flow-chart) del programma per il riconoscimento di numeri negativi o positivi.*

```
6  IND1   85   STA ØØ
7         ØØ
8         4C   JMP Monitor
9         22
A         FE
B  AVA1   A9   LDA #ØØ
C         ØØ
D         F0   BEQ IND1
E         F7
```

Commentiamo di seguito per esteso il programma che è stato appena scritto.

*1ª istruzione: LDA Ø1* - Si carica il contenuto della locazione di memoria ØØØ1 (pagina zero) nell'accumulatore. Automaticamente la CPU influenza i bit di stato Z e N, per cui nella istruzione successiva possiamo andarne a controllare uno (in particolare N) per decidere se eseguire il salto condizionato.

*2ª istruzione: BPL AVA1* - Abbiamo "etichettato" questa istruzione con un nome di fantasia (AVA1). Questa procedura è molto comoda nella stesura dei programmi perché serve ad indicare dove dobbiamo eseguire il salto. Con l'istruzione BPL la CPU va a vedere se il bit di negativo (N) è = Ø. Notiamo che è l'istruzione che precede (la prima) che ha condizionato il bit N. Se N = Ø (contenuto dell'accumulatore ≧ Ø) il salto viene eseguito, diversamente si prosegue nella esecuzione del programma con la istruzione immediatamente successiva.

*3ª - 4ª - 5ª - 6ª istruzione* - Nessun commento salvo che alla 4ª è stato attribuito il nome IND 1.

*7ª istruzione: BEQ IND1* - Notiamo subito che la 6ª istruzione ha caricato in maniera immediata zero nell'accumulatore. Quindi il risultato della 6ª istruzione è sempre e comunque uno zero; allora dopo questa istruzione avrò sempre e comunque Z = 1. È chiaro allora che l'istruzione BEQ in questa posizione equivale ad un salto incondizionato, cioè il salto viene sempre e comunque eseguito. (LDA #ØØ infatti genera sempre Z = = 1, BEQ va a controllare che, se è uguale a uno, salta: quindi esegue sempre il salto in questo caso).

Analizziamo ora l'entità di questi salti.

Nel microprocessore 6502 il Branch è sempre relativo alla posizione del Program Counter. Ciò significa che quando si esegue il Branch, il numero che segue il codice operativo del Branch viene sommato al valore del Program Counter in quel momento, quindi l'esecuzione del programma continua dal nuovo Program Counter generato.

Nel nostro esempio la CPU preleva dalla memoria alle locazioni Ø322 - Ø323 puntate dal Program Counter l'istruzione 1Ø, Ø7 (BPL AVA1) posizionandosi subito dopo alla locazione successiva Ø324 e preparandosi, se non succede niente, ad eseguire la istruzione contenuta in questa locazione. Prima della esecuzione della BPL ci troviamo quindi nella seguente condizione:

PC = Ø324.

Se il salto deve essere eseguito bisogna posizionarsi sulla locazione Ø32B, ovvero il PC deve essere incremento di Ø32B − Ø324 = Ø7. Notiamo subito che Ø7 è il 2° byte dell'istruzione di branch. Esso rappresenta, come si suol dire, il "displacement" del Program Counter ed è relativo al P.C. medesimo, è cioè indipendente dalla locazione di memoria nella quale abbiamo cominciato a caricare il programma (per esempio il programma può girare anche se caricato alla locazione di memoria Ø21Ø invece che alla Ø32Ø). Lo stesso conto per spostare il Program Counter può essere fatto nel salto negativo (ultima istruzione). In questo caso bisogna saltare dalla locazione Ø32F, dove si trova posizionato il P.C., alla Ø326. Vale allora la relazione "displacement" = Ø326 − Ø32F = −(Ø32F − Ø326) = −Ø9 = F7, che è appunto il 2° byte della istruzione BEQ IND1. In pratica quando scriviamo un programma lasciamo uno spazio bianco dopo le istruzioni di Branch: il valore del salto viene definito a conclusione del programma.

Per calcolare i numeri positivi o negativi che rappresentano le entità del salto si può utilizzare con vantaggio un comodo programma (presente del Monitor) che fino ad ora non abbiamo adoperato, ma che è stato previsto proprio per questo scopo.

Questo programma parte dalla locazione di memoria FF97. Premiamo allora i tasti [AD] FF97 [RUN].

A questo punto riprendiamo per esempio il programma che abbiamo precedentemente analizzato: il primo salto (BPL AVA 1) doveva essere fatto dalla locazione Ø322 (con questo metodo non dobbiamo tener conto della posizione del P.C. poiché ci pensa il programma) alla Ø32B. Ora basterà semplicemente battere sulla tastiera nella successione:

1) la parte bassa (2° byte) dell'indirizzo di partenza: 22.

2) la parte bassa dell'indirizzo a cui si vuole saltare: 2B.

Sul display indirizzi comparirà 222B e sul display dati il numero Ø7, che è il valore relativo al salto nel sistema complemento a due.

Stessa procedura per il salto all'indietro: battete sulla tastiera 2D (l'istruzione BEQ IND1 è infatti all'indirizzo Ø32B poi 26 (infatti bisogna saltare alla locazione Ø326) e otterrete il risultato cercato F7.

Il tutto, come si è visto, è molto semplice: ogni volta che serve basta richiamare questo programma che è sempre lì, registrato permanentemente nella PROM del Monitor.

Si fa notare infine che i salti in avanti e indietro possono essere fatti nel campo dei numeri positivi e negativi del sistema complemento a due; per salti più lunghi si associa alla istruzione di Branch quella di JMP (salto incondizionato) che consente di andare in qualsiasi punto dell'intera memoria.

In particolare osserviamo che il pro-

gramma per il calcolo del salto che abbiamo descritto permette di eseguire salti in avanti fino a +129 posizioni e indietro fino a -126 posizioni (127+2=129 e -128+2= V = - 126 poiché il programma tiene conto delle due posizioni più avanti del P.C.).

## Index X

In questo paragrafo introduciamo un nuovo registro interno alla CPU: *il registro indice X.*

Fino ad ora della CPU abbiamo visto tre soli registri: l'Accumulatore A, il Program Counter P.C., e la Status S. Vedremo che c'è ne sono ancora tre; cominciamo ad esaminare il primo di questi.

Il registro indice X è un registro di 8 bit sul quale è possibile eseguire dei calcoli, o, cosa più importante, indirizzare la memoria con metodi nuovi rispetto a quelli visti fino ad ora.

Analizziamo prima questo ultimo aspetto considerando il metodo di indirizzamento tramite il registro X.

Premettiamo che le istruzioni che usano questo tipo di indirizzamento sono sempre formate da due byte, di cui il primo è come sempre, il codice operativo, il secondo è un byte così detto di spostamento (in inglese *displacement).*

## Indirizzamento in pagina zero indicizzato

Per i codici operativi di questa istruzione si faccia riferimento anche alla tabella riassuntiva di tutte le istruzioni del 6502 pubblicata in questo articolo.

Usando il sistema di indirizzamento indicizzato, l'indirizzo della locazione di memoria sulla quale si vuole operare, viene calcolato dalla CPU sommando il contenuto del secondo byte dell'istruzione con il contenuto del registro indice, senza tener conto del Carry. Il risultato di questo calcolo (che avviene sempre in codice esadecimale) è un numero da 8 bit che rappresenta la parte bassa dell'indirizzo essendo la parte alta 00 perchè si opera in pagina zero.

Per chiarire meglio le idee vediamo subito un esempio pratico.

Ammettiamo che sia X = 3B (registro indice X = 3B). Se si esegue la istruzione

```
B5   LDA 02, X
02
```

la CPU calcola l'indirizzo della locazione di memoria il cui contenuto viene portato in accumulatore eseguendo la operazione

```
3B +   (contenuto dell'indice X)
02 =   (2° byte dell'istruzione)
3D     (indirizzo della locazione di memoria interessata)
```

N.B. B5 è il codice operativo della istruzione LDA con il sistema di indirizzamento in pagina zero indicizzato.

La domanda che sorge ora è l'uso che si può fare di questo sistema.

Innanzitutto premettiamo che esiste un certo numero di istruzioni che operano su X che sono:

CPX: Confronto di X con un numero.
DEX: Decremento di uno il valore di X (X = X − 1).

**Tabella 1 - Tavole di conversione da numeri in codice decimale a numeri in complemento a due e viceversa.**

NUMERI NEGATIVI

| 1ª cifra \ 2ª cifra | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 128 | 127 | 126 | 125 | 124 | 123 | 122 | 121 | 120 | 119 | 118 | 117 | 116 | 115 | 114 | 113 |
| 9 | 112 | 111 | 110 | 109 | 108 | 107 | 106 | 105 | 104 | 103 | 102 | 101 | 100 | 99 | 98 | 97 |
| A | 96 | 95 | 94 | 93 | 92 | 91 | 90 | 89 | 88 | 87 | 86 | 85 | 84 | 83 | 82 | 81 |
| B | 80 | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 | 70 | 69 | 68 | 67 | 66 | 65 |
| C | 64 | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 |
| D | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 |
| E | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 |
| F | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

NUMERI POSITIVI

| 1ª cifra \ 2ª cifra | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 2 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 3 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 4 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 5 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 6 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 7 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |

INX: Incremento di uno il valore di X ( X = X + 1).
LDX: Carico X con un certo valore immediato o no.
STX: porto il valore di X in una certa locazione di memoria.
TAX, TXA: porto il contenuto dell'accumulatore in X e viceversa.

Detto questo vediamo di risolvere un semplice problema che utilizzi queste istruzioni: scrivere un programma che porti a zero tutte le locazioni di memoria comprese fra 0020 e 002F. Si noti che questo tipo di operazione viene normalmente fatto nei programmi di inizializzazione di un elaboratore all'atto della accensione.

Il diagramma di flusso del programma può essere quello rappresentato in fig. 4.

Scriviamo allora il seguente programma:

| | | | |
|---|---|---|---|
| 0200 | A2 | LDX #00 | Carico X con 0 in modo immediato |
| 1 | 00 | | |
| 2 | 8A | TXA | Copio il contenuto di X in Accumulatore |
| 3 Loop | 95 | STA 20,X | Memorizzo il contenuto dell'Acc. nella locazione di memoria X + 20 |
| 4 | 20 | | |
| 5 | E8 | INX | Incremento X (X = X + 1) |
| 6 | E0 | CPX #$10 | Confronto X con 10, cioè X = 10? |
| 7 | 10 | | |
| 8 | D0 | BNE LOOP | Se X non è uguale a 10 torno ad eseguire la istruzione STA |
| 9 | F9 | | |
| A | 4C | JMP Monitor | Stop |
| B | 22 | | |
| C | FE | | |

Scrivete il programma nell'AMICO 2000A, fatelo girare e controllate che tutte le locazioni di memoria dalla 0020 alla 002F contengano ora 00.

Il programma può sembrare un tantino complicato, vediamo allora di descriverlo in modo discorsivo in modo da comprendere ogni passaggio.

Per prima cosa si è caricato nell'index X il numero 00 (dato immediato), quindi, siccome ci interessa che anche nell'accumulatore ci sia 00 si copia il contenuto di X in accumulatore.

Questo è stato fatto per risparmiare un byte di programmazione (l'alternativa era LDA# 00, cioè A900 = 2 byte) . A questo punto inizia un "loop", che viene identificato con una etichetta sulla prima istruzione, che è STA e cioè il trasferimento del contenuto dell'accumulatore nella locazione di memoria il cui indirizzo è 20 + X. Al primo passaggio di questo giro (loop) questa locazione sarà 0020 + 00 = 0020 e cioè la prima locazione di memoria che deve essere portata a zero. A questo si incrementa il contenuto del registro indice X di uno. Si noti che con questa istruzione (INX) il contenuto di X viene incrementato nel sistema di numerazione esadecimale anche se la macchina è in funzionamento decimale. Dato che devo portare a zero 16 locazioni di memoria mi chiedo se $X = 10_{16}$ (questa uguaglianza verrà verificata al 17° passaggio, infatti $10_{16} = 17_{10}$) e cioè se ho finito il mio lavoro (ad ogni loop X si incrementa di uno perchè si passa attraverso l'istruzione INX). La istruzione seguente è quella di Branch e viene eseguita se X non è uguale a 10. Ricordiamo che F9 significa −7, la CPU cioè riparte da 7 byte indietro cioè dalla prima istruzione del loop.

Notiamo che con questo programma si è eseguita una operazione di STA per 16 volte successive, sempre in posizioni diverse e con pochissime istruzioni. Abbiamo fatto un loop, come si dice in inglese, incrementando ad ogni giro l'indirizzo di memorizzazione del dato, cosa resa possibile dal sietma di indirizzamento indicizzato. Ripetiamo ancora una volta che stiamo operando con locazioni di memoria site in pagina zero.

Lo stesso programma può essere scritto almeno in altre due maniere diverse risparmiando qualche istruzione, provateci e mandateci la soluzione.

## Il Flow-Chart

Le operazioni logiche che si eseguono nella stesura di un programma possono venir rappresentate tramite un sistema grafico detto dei diagrammi di flusso o *flow-chart*. Questo sistema è molto utile per rappresentazioni visivamente immediate e comprensibili di un programma e utilizza dei simboli grafici come quelli rappresentati in fig. 5.

Alcuni di questi simboli sono già stati visti; di essi sono particolrmante interessanti i blocchi 2 e 3. Il 2 indica una operazione che bisogna eseguire che in generale può essere indicata come aritmetica o di movimento dati; questo blocco ha un ingresso e una uscita.

Il blocco 3 è quello decisionale; sta ad indicare una domanda che ci si pone, domanda che ha sempre due risposte, SI e NO. Questo blocco ha dunque un ingresso e due uscite, una uscita per il SI e una per il NO.

Il blocco 1 caratterizza le operazioni di ingresso e di uscita dall'elaboratore. Ad esempio lo si usa per indicare il prelevamento di un dato da una tastiera. I blocchi 4 e 5 verranno descritti in una altra sede. L'incrocio indicato al punto 6 è chiaro: esso indica un punto di rientro

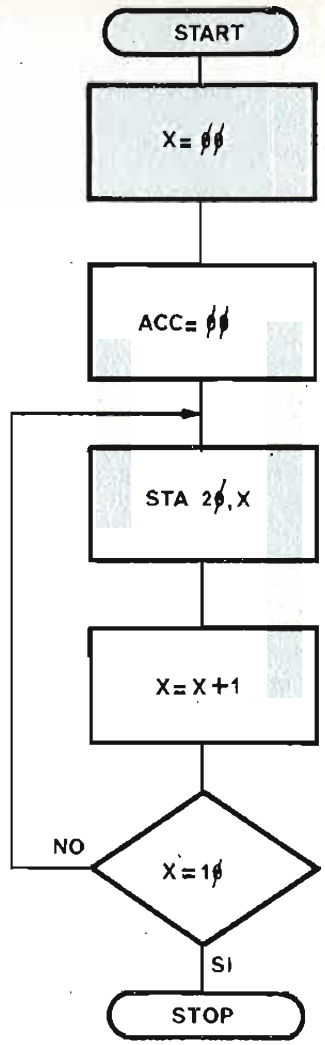



*Fig. 4 - Diagramma di flusso per l'azzeramento di una zona di memoria.*

su un altro flusso, quando, per esempio si effettua un rientro da un test (blocco 3). Il simbolo al blocco 7 indica l'inizio o la fine di un programma.

Generalmente quando si affronta un problema di programmazione per prima cosa si stende una sequenza logica di operazioni che risolvono il problema usando i diagrammi di flusso. Questi diagrammi sono immediatamente comprensibili e di facile correzione. Il vantaggio che offrono è che la stesura del diagramma è indipendente da come verrà scritto il programma effettivo; per esempio è indipendente dal particolare set di istruzioni della CPU che poi eseguirà il programma.

Una volta verificata la correttezza delle operazioni logiche descritte nel diagramma di flusso si può cominciare a scrivere il programma stesso nel linguaggio della macchina che si vuole utilizzare. Il primo passo è sempre il più importante perchè è quello che risolve effettivamente il problema. . secondo è solo la fase realizzativa del tutto. Se infatti supponiamo

di avere un programma scritto in linguaggio Assembler per un certo microprocessore diverso dal nostro, ci accorgeremmo subito che è estremamente difficoltoso riscriverlo nel linguaggio del nostro computer. Se però abbiamo il flow-chart relativo a quel programma riscriverlo sarà una cosa molto semplice.

## Le istruzioni del 6502

In questo numero pubblichiamo una tabella che riassume tutte le istruzioni del microprocessore 6502 utilizzato nel nostro AMICO 2000. Di tutte le istruzioni riportate molte vi sono ormai note. Oltre che per ricordare i vari codici operativi, la tabella fornisce altre importanti indicazioni. Vediamo come si legge.

Nella prima riga in alto sono riportati i vari sistemi di indirizzamento, alcuni dei quali sono stati già analizzati.

Nella seconda riga troviamo, dopo la rappresentazione mnemonica della istruzione e l'operazione ad essa associata, tre simboli di cui diamo il significato:
OP sta per Codice Operativo
N è il numero di byte che formano la istruzione:
# rappresenta il numero dei cicli macchina (Nel caso dell'AMICO 2000 il numero dei microsecondi impiegati per eseguire l'istruzione).

Nell'ultima colonna di questa stessa riga sono riportati i nomi dei bit dello Status che abbiamo già analizzato. Se l'istruzione influenza qualcuno di questi bit, in corrispondenza si vede un segno v, diversamente appare un trattino -.

Per meglio comprendere facciamo un esempio pratico.

Si voglia tradurre l'istruzione LDA #$05.

Dalla tabella vediamo che si tratta di una istruzione di 2 byte, che viene eseguita in 2 μs, il cui codice (per lo indirizzamento immediato) è A9 e che influenza i bit N e Z.

La traduzione di questa istruzione in linguaggio macchina è quindi semplice: A9 05.

I simboli che compaiono nella colonna OPERAZIONE verranno spiegati via via che verrà analizzata l'istruzione.

## Esercizi

Questa volta vi faremo fare un po' di esercizi per abituarvi a prendere confidenza con le istruzioni fino ad ora imparate.

Per questi esercizi non vi daremo qui la soluzione, ma la aspettiamo da voi: potrete spedircela se volete. In ogni caso nel prossimo articolo vi presenteremo tutte le soluzioni che poi confronterete con le vostre.

*1° esercizio* - Scrivere un programma che esegue la somma di due numeri ciascuno da due byte (le somme che abbiamo fatto fino ad ora sono sempre state fra due numeri di 1 byte ciascuno) e positivi. Estendere poi il programma a numeri di più byte.

Per aiutarvi ricordiamo che se la somma di due numeri ha un riporto questo va automaticamente nel Carry. Se ad esempio scriviamo 93 + 75, il risultato è di 1 68 cioè 68 e 1 nel Carry. Se ora sommiamo due numeri formati da quattro cifre come 1393 + 7275 potremmo fare come segue:

13 93+
72 75=

scomponiamo

```
   93+
   75=
  ----
 [1] 68
     e
    13+
    72+
  ↳  1=
  ----
    86
```

il risultato finale è 8668.

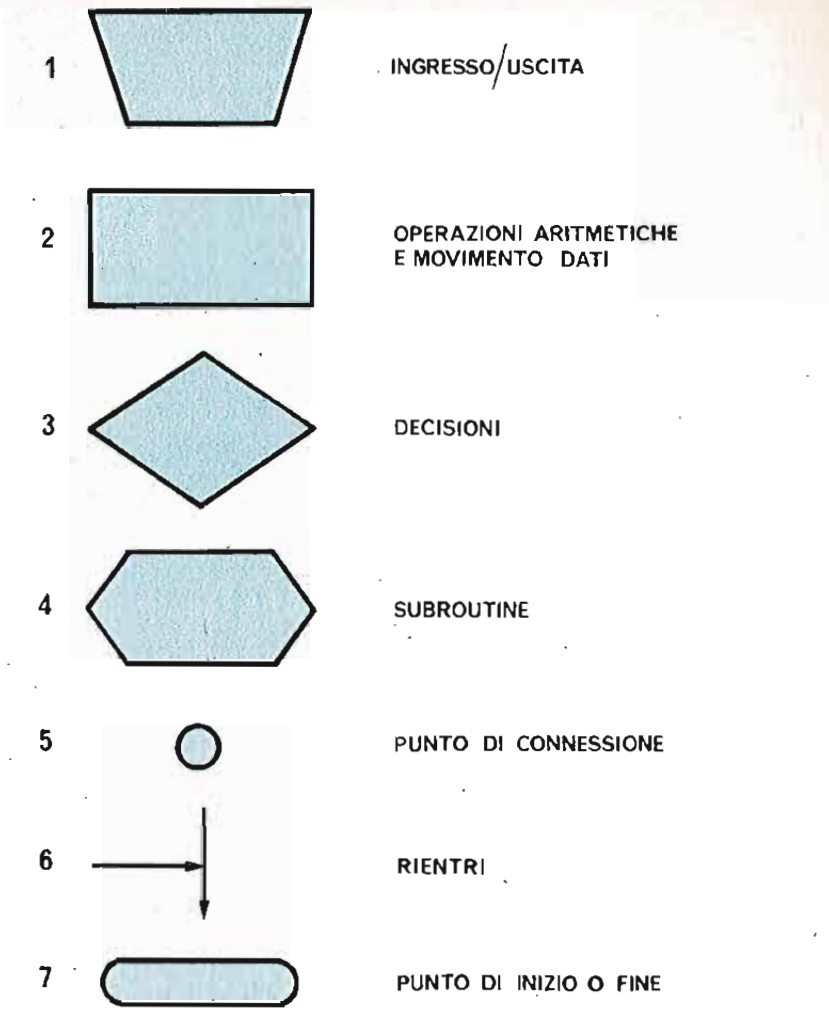



*Fig. 5 - Simboli grafici usati nella costruzione di un programma per mezzo della flow-chart.*

Utilizzare lo stesso metodo per scrivere il programma tenendo presente ancora un volta che il Carry è automatico e che in quattro successive locazioni di memoria potreste mettere rispettivamente il 1° e il 2° byte del primo numero e il 1° e il 2° byte del secondo numero; in altre due locazioni di memoria successive potreste mettere il 1° e il 2° byte del risultato.

Sviluppando il programma per la somma di numeri di più byte vi facciamo notare che un numero di 4 byte arriva a 4 miliardi!

*2° esercizio* - Scrivere un programma che esegua la moltiplicazione di due numeri da 8 bit (1 byte).

Il metodo più immediato, per tutto ciò che fino ad ora avete appreso, è quello di caricare il 1° numero nell'index X e sommare l'altro tante volte quanto è il contenuto di X; questo è possibile se ad ogni loop si decrementa X. Attenzione però che X si decrementa (e incrementa) sempre in esadecimale.

*3° esercizio* - Scrivere un programma che carichi automaticamente dalla locazione 0000 alla 0050 numeri crescenti

| ISTRUZIONE | | IMMEDIATE | | | ABSOLUTE | | | ZERO PAGE | | | ACCUM. | | | IMPLIED | | | (IND.X) | | | (IND), Y | | | Z. PAGE, X | | | ABS, X | | | ABS, Y | | | RELATIVE | | | INDIRECT | | | Z. PAGE, Y | | | CONDITION CODES | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MNEMONICO | OPERAZIONE | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | OP | N | # | N | Z | C | I | D | V |
| A D C | A+M+C → A (4) (1) | 69 | 2 | 2 | 6D | 4 | 3 | 65 | 3 | 2 | | | | | | | 61 | 6 | 2 | 71 | 5 | 2 | 75 | 4 | 2 | 7D | 4 | 3 | 79 | 4 | 3 | | | | | | | | | | √ | √ | √ | – | – | √ |
| A N D | A ∧ M → A (1) | 29 | 2 | 2 | 2D | 4 | 3 | 25 | 3 | 2 | | | | | | | 21 | 6 | 2 | 31 | 5 | 2 | 35 | 4 | 2 | 3D | 4 | 3 | 39 | 4 | 3 | | | | | | | | | | √ | √ | – | – | – | – |
| A S L |  | | | | 0E | 6 | 3 | 06 | 5 | 2 | 0A | 2 | 1 | | | | | | | | | | 16 | 6 | 2 | 1E | 7 | 3 | | | | | | | | | | | | | √ | √ | √ | – | – | – |
| B C C | BRANCH ON C=0 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 90 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B C S | BRANCH ON C=1 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | B0 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B E Q | BRANCH ON Z=1 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | F0 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B I T | A ∧ M | | | | 2C | 4 | 3 | 24 | 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | M7 | √ | – | – | – | M6 |
| B M I | BRANCH ON N=1 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 30 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B N E | BRANCH ON Z=0 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | D0 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B P L | BRANCH ON N=0 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 10 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B R K | | | | | | | | | | | | | | 00 | 7 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | 1 | – | – |
| B V C | BRANCH ON V=0 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 50 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| B V S | BRANCH ON V=1 (2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 70 | 2 | 2 | | | | | | | – | – | – | – | – | – |
| C L C | 0 → C | | | | | | | | | | | | | 18 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | 0 | – | – | – |
| C L D | 0 → D | | | | | | | | | | | | | D8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | 0 | – |
| C L I | 0 → I | | | | | | | | | | | | | 58 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | 0 | – | – |
| C L V | 0 → V | | | | | | | | | | | | | B8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | 0 |
| C M P | A-M (1) | C9 | 2 | 2 | CD | 4 | 3 | C5 | 3 | 2 | | | | | | | C1 | 6 | 2 | D1 | 5 | 2 | D5 | 4 | 2 | DD | 4 | 3 | D9 | 4 | 3 | | | | | | | | | | √ | √ | √ | – | – | – |
| C P X | X-M | E0 | 2 | 2 | EC | 4 | 3 | E4 | 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | √ | – | – | – |
| C P Y | Y-M | C0 | 2 | 2 | CC | 4 | 3 | C4 | 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | √ | – | – | – |
| D E C | M – 1 → M | | | | CE | 6 | 3 | C6 | 5 | 2 | | | | | | | | | | | | | D6 | 6 | 2 | DE | 7 | 3 | | | | | | | | | | | | | √ | √ | – | – | – | – |
| D E X | X – 1 → X | | | | | | | | | | | | | CA | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| D E Y | Y – 1 → Y | | | | | | | | | | | | | 88 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| E O R | A ∀ M → A (1) | 49 | 2 | 2 | 4D | 4 | 3 | 45 | 3 | 2 | | | | | | | 41 | 6 | 2 | 51 | 5 | 2 | 55 | 4 | 2 | 5D | 4 | 3 | 59 | 4 | 3 | | | | | | | | | | √ | √ | – | – | – | – |
| I N C | M + 1 → M | | | | EE | 6 | 3 | E6 | 5 | 2 | | | | | | | | | | | | | F6 | 6 | 2 | FE | 7 | 3 | | | | | | | | | | | | | √ | √ | – | – | – | – |
| I N X | X + 1 → X | | | | | | | | | | | | | E8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| I N Y | Y + 1 → Y | | | | | | | | | | | | | C8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| J M P | JUMP TO NEW LOC | | | | 4C | 3 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 6C | 5 | 3 | | | | – | – | – | – | – | – |
| J S R | JUMP SUB | | | | 20 | 6 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| L D A | M → A (1) | A9 | 2 | 2 | AD | 4 | 3 | A5 | 3 | 2 | | | | | | | A1 | 6 | 2 | B1 | 5 | 2 | B5 | 4 | 2 | BD | 4 | 3 | B9 | 4 | 3 | | | | | | | | | | √ | √ | – | – | – | – |
| L D X | M → X (1) | A2 | 2 | 2 | AE | 4 | 3 | A6 | 3 | 2 | | | | | | | | | | | | | | | | | | | BE | 4 | 3 | | | | | | | B6 | 4 | 2 | √ | √ | – | – | – | – |
| L D Y | M → Y (1) | A0 | 2 | 2 | AC | 4 | 3 | A4 | 3 | 2 | | | | | | | | | | | | | B4 | 4 | 2 | BC | 4 | 3 | | | | | | | | | | | | | √ | √ | – | – | – | – |
| L S R |  | | | | 4E | 6 | 3 | 46 | 5 | 2 | 4A | 2 | 1 | | | | | | | | | | 56 | 6 | 2 | 5E | 7 | 3 | | | | | | | | | | | | | 0 | √ | √ | – | – | – |
| N O P | NO OPERATION | | | | | | | | | | | | | EA | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| O R A | A V M → A | 09 | 2 | 2 | 0D | 4 | 3 | 05 | 3 | 2 | | | | | | | 01 | 6 | 2 | 11 | 5 | 2 | 15 | 4 | 2 | 1D | 4 | 3 | 19 | 4 | 3 | | | | | | | | | | √ | √ | – | – | – | – |
| P H A | A → Ms S-1 → S | | | | | | | | | | | | | 48 | 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| P H P | P → Ms S-1 → S | | | | | | | | | | | | | 08 | 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| P L A | S+1→ S Ms → A | | | | | | | | | | | | | 68 | 4 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| P L P | S+1 → S Ms → P | | | | | | | | | | | | | 28 | 4 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | Ripristinato | | | | | |
| R O L |  | | | | 2E | 6 | 3 | 26 | 5 | 2 | 2A | 2 | 1 | | | | | | | | | | 36 | 6 | 2 | 3E | 7 | 3 | | | | | | | | | | | | | √ | √ | √ | – | – | – |
| R O R |  | | | | 6E | 6 | 3 | 66 | 5 | 2 | 6A | 2 | 1 | | | | | | | | | | 76 | 6 | 2 | 7E | 7 | 3 | | | | | | | | | | | | | √ | √ | √ | – | – | – |
| R T I | RTRN. INT. | | | | | | | | | | | | | 40 | 6 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | Ripristinato | | | | | |
| R T S | RTRN SUB | | | | | | | | | | | | | 60 | 6 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| S B C | A-M-C̄ → A (1) | E9 | 2 | 2 | ED | 4 | 3 | E5 | 3 | 2 | | | | | | | E1 | 6 | 2 | F1 | 5 | 2 | F5 | 4 | 2 | FD | 4 | 3 | F9 | 4 | 3 | | | | | | | | | | √ | √ | (3) | – | – | √ |
| S E C | 1 → C | | | | | | | | | | | | | 38 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | 1 | – | – | – |
| S E D | 1 → D | | | | | | | | | | | | | F8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | 1 | – |
| S E I | 1 → I | | | | | | | | | | | | | 78 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | 1 | – | – |
| S T A | A → M | | | | 8D | 4 | 3 | 85 | 3 | 2 | | | | | | | 81 | 6 | 2 | 91 | 6 | 2 | 95 | 4 | 2 | 9D | 5 | 3 | 99 | 5 | 3 | | | | | | | | | | – | – | – | – | – | – |
| S T X | X → M | | | | 8E | 4 | 3 | 86 | 3 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 96 | 4 | 2 | – | – | – | – | – | – |
| S T Y | Y → M | | | | 8C | 4 | 3 | 84 | 3 | 2 | | | | | | | | | | | | | 94 | 4 | 2 | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| T A X | A → X | | | | | | | | | | | | | AA | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| T A Y | A → Y | | | | | | | | | | | | | A8 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| T S X | S → X | | | | | | | | | | | | | BA | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| T X A | X → A | | | | | | | | | | | | | 8A | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |
| T X S | X → S | | | | | | | | | | | | | 9A | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | – | – | – | – | – | – |
| T Y A | Y → A | | | | | | | | | | | | | 98 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | √ | √ | – | – | – | – |

(1) Aggiungere 1 a "N" se si passano i limiti di pagina.
(2) Aggiungere 1 a "N" se il BRANCH rimane nei limiti della pagina. Aggiungere 2 a "N" se il BRANCH avviene su un'altra pagina.
(3) Carry Negato = BORROW.
(4) Se si è in funzionamento "DECIMALE" il Flag Z non è valido. Bisogna eseguire un test dell'accumulatore per rivedere il risultato zero.

X Registro Indice X
Y Registro Indice Y
A Accumulatore
M Locazione di memoria dell'indirizzo effettivo
Ms Locaz. di memoria puntata dallo STACK POINTER.

\+ Somma.
– Differenza
∧ AND.
∨ OR
∀ OR esclusivo.
√ Modificato.
– Non modificato.

M7 Bit 7 della loc. di memoria (MSB).
M6 Bit 6 della loc. di memoria.
N N. di cicli.
# N. di Bytes

## METODI DI INDIRIZZAMENTO

**IMM** - Indirizzamento Immediato. - L'operando è contenuto nel 2° byte dell'istruzione.

**ABS** - Indirizzamento assoluto. - Il secondo byte dell'istruzione contiene gli 8 bit più bassi dell'indirizzo effettivo. Il terzo byte contiene gli 8 bit più alti dell'indirizzo effettivo.

**Z Page** - Indirizzamento in pagina zero. - Il secondo byte dell'istruzione contiene gli 8 bit più bassi dell'indirizzo effettivo. Gli 8 bit più alti sono zero.

**A** - Accumulatore. - Istruzioni da 1 byte che agiscono sull'accumulatore

**Z. Page, X - Z. Page, Y.** Indirizzamento indicizzato in pagina zero. - Il secondo byte della istruzione viene sommato al registro indice (il Carry viene ignorato) per formare il byte basso dell'indirizzo effettivo. Il Byte alto dell'indirizzo effettivo è zero.

**ABS, X - ABS, Y** indirizzamento indicizzato assoluto. - L'indirizzo effettivo viene calcolato sommando il registro indice al secondo e terzo byte dell'istruzione.

**(IND, X)** Indirizzamento indicizzato indiretto. - Il secondo byte dell'istruzione viene aggiunto al registro indice X (senza tenere conto del carry). Il risultato punta su una locazione in pagina zero, che contiene gli 8 bit più bassi dell'indirizzo effettivo. Il byte successivo della pagina zero, contiene gli 8 bit più alti dell'indirizzo effettivo.

**(IND), Y** Indirizzamento indiretto indicizzato. - Il secondo byte dell'istruzione punta su una locazione in pagina zero. Il contenuto di questa locazione di memoria viene aggiunto al registro indice Y. Il risultato è gli 8 bit più bassi dell'indirizzo effettivo. Il carry della prima somma viene aggiunto al contenuto della successiva locazione di memoria di pagina zero e il risultato è gli 8 bit più alti dell'indirizzo effettivo.

**Programma 1 - Trasformazione di numeri decimali negativi e positivi in corrispondenti nel sistema complemento a due.**

```
0200 A5 11 F0 0C A2 00 F8 18 E8 69 99 C9 00 D0 F8 8A
0210 18 D8 A6 10 F0 02 69 64 60 EA A8 29 0F AA BD EA
0220 FF 85 0F 98 4A 4A 4A 4A AA BD EA FF 60 EA EA EA
0230 A9 00 AA 95 00 E8 E0 9F D0 F9 A5 11 20 1A 02 85
0240 91 A5 0F 85 92 A5 12 20 1A 02 85 93 A5 0F 85 94
0250 A5 10 D0 04 A9 3F D0 02 A9 06 85 90 20 7E FF EA
0260 F8 20 2D FF D0 04 85 0A F0 D0 A5 0A D0 CC E6 0A
0270 A9 99 8D 03 FD 20 57 FF C9 15 D0 09 A9 40 45 8F
0280 85 8F 4C 99 02 C9 10 B0 44 48 A5 11 0A 0A 0A 0A
0290 26 10 85 11 68 05 11 85 11 A9 01 25 10 85 10 F0
02A0 1B A5 3F D0 0D A5 11 C9 27 90 05 A9 27 85 11 18
02B0 90 0A A5 11 C9 28 90 F8 A9 27 D0 F1 20 00 02 A6
02C0 8F F0 05 49 FF 38 69 00 85 12 4C 3A 02 C9 12 D0
02D0 F9 4C 30 02
```

**Programma 2 - Gioco del "master mind". Il programma deve essere fatto partire (RUN) una volta caricato dalla locazione 02A0.**

```
0200 20 0C FF 20 2D FF D0 04 85 0D F0 F4 A5 0D D0 F0
0210 E6 0D F8 A9 99 8D 03 FD 20 57 FF C9 15 F0 28 EA
0220 EA EA EA C9 10 F0 D9 A2 04 06 FA 26 FB CA D0 F9
0230 05 FA 85 FA 4C 00 02 EA EA EA 48 29 0F 95 01 68
0240 4A 4A 4A 4A 95 00 60 A5 FA A2 03 20 3A 02 A5 FB
0250 A2 01 20 3A 02 A9 00 85 09 85 0A A0 04 A2 01 B5
0260 00 D5 04 D0 02 E6 09 D5 04 D0 02 E6 0A E8 E0 05
0270 D0 F5 20 85 02 88 D0 E5 A5 09 0A 0A 0A 0A 05 0A
0280 85 F9 4C 00 02 A2 01 B5 00 95 FF E8 E0 09 D0 F7
0290 B5 FB 95 FF B5 F7 95 FE 60 EA EA EA EA EA EA EA
02A0 A9 00 85 F9 85 FA 85 FB A2 05 18 F8 A5 0C 29 07
02B0 75 00 29 0F 95 00 C6 0C E8 E0 09 D0 EF A9 00 85
02C0 0B A2 05 B5 00 D5 01 D0 0E 48 18 69 01 29 0F 95
02D0 01 68 E6 0B 4C C1 02 E8 E0 08 D0 E9 C6 0E F0 06
02E0 20 85 02 4C C1 02 20 0C FF A9 99 8D 03 FD 20 57
02F0 FF C9 12 D0 03 4C 00 02 4C A8 02
```

in codice esadecimale dallo 00 al 50 (in pratica sul display dati si dovranno vedere numeri sempre uguali alla parte bassa dell'indirizzo fino alla locazione 0050).

## Un gioco: il "master mind"

Ed ora, dopo tanto lavorare con software, istruzioni, esercizi etc., ci sembra opportuno un attimo di relax con un giochino molto simpatico e molto conosciuto sotto il nome di "master mind".

Si tratta in sostanza di un programma che fa generare casualmente all'AMICO 2000A 4 numeri diversi (o meglio un numero di quattro cifre) che dobbiamo indovinare. Si tratta di un gioco interattivo cioè di un gioco in cui il microcalcolatore dialoga con noi dando delle risposte. Come? Spieghiamo subito il gioco. Il programma è riportato a fine articolo (viene dato il cosiddetto codice oggetto) e si carica a partire dalla locazione di memoria 0200.

Per farlo partire, una volta caricato, bisogna posizionarsi all'indirizzo 02A0 poi premere RUN.

A questo punto tutte le cifre del display lampeggiano con una frequenza abbastanza elevata: in questa situazione il microelaboratore sta scegliendo le cifre che compongono il numero.

Per arrestare questa operazione di ricerca dei numeri e per cominciare quindi il gioco si preme il tasto C, a questo punto tutte le cifre sono zeri.

Cominciamo il gioco battendo sulla tastiera una successione di quattro numeri qualunque (all'inizio tutti possono essere quelli buoni) che appariranno sul display indirizzi.

Premiamo ora il tasto F, sul display dati apparirà qualcosa (può rimanere anche 00):

– La cifra a destra del display dati indica quante cifre su quattro sono state indovinate;

– la cifra a sinistra del display dati indica quante cifre di quelle indovinate sono al loro giusto posto.

In base a queste indicazioni batteremo sulla tastiera altri numeri o li disporremo in maniera differente fino a che il display dati non indica 44: questo significherà che abbiamo trovato il numero esatto scelto dal microelaboratore (quattro cifre esatte tutte al loro posto).

Ricordate di battere sempre il tasto F dopo aver introdotto i nuovi numeri per avere la risposta del microelaboratore.

Per aiutarvi nei ragionamenti è meglio scrivere su un foglio di carta i numeri introdotti e la relativa risposta.

La bravura del giocatore sta nell'indovinare con il minor numero di tentativi (qualche volta si tratterà anche di fortuna, quella c'entra sempre!).

Il programma è piuttosto lungo da inserire a mano, vi consigliamo quindi di registrarlo subito, dopo averlo caricato la prima volta e averne verificato il buon funzionamento.

Attenzione, un ultimo avvertimento: il numero generato dal micro ha sempre tutte le cifre diverse; nel tentare di indovinarlo inserite sempre NUMERI CON CIFRE UNA DIVERSA DALLA ALTRA

**ATTENZIONE.** Molti lettori hanno scritto lamentandosi che le PROM scaldano troppo. La loro temperatura di funzionamento normale è intorno ai 60-70°C, tanto da non poterci tenere il dito sopra. Non bisogna perciò preoccuparsi di questo apparente surriscaldamento.

**ERRATA CORRIGE.** Nel programma della tombola pubblicato a pag. 524 del numero 6 Giugno 1979 di Sperimentare ci sono due errori di stampa nelle istruzioni: il codice operativo della istruzione 0294 da 4F (errato) diventa 4B (corretto), mentre quello della istruzione 02F1 da 0F (errato) diventa 0B.