

Don Inman e Kurt Inman

Imparate il linguaggio dell'Apple



Il piacere del computer



Il piacere del computer

Il piacere del computer serie diretta da Mauro Boscarol

- 1 *Tom Rugg e Phil Feldman* 32 programmi con il PET
- 2 *Rich Didday* Intervista sul personal computer, hardware
- 3 *Tom Rugg e Phil Feldman* 32 programmi con l'Apple
- 4 *Ken Knecht* Microsoft Basic
- 5 *Paul M. Chirlian* Pascal
- 6 *Tom Rugg e Phil Feldman* 32 programmi con il TRS-80
- 7 *Rich Didday* Intervista sul personal computer, software
- 8 *Herbert D. Peckham* Imparate il Basic con il PET
- 9 *Karl Townsend e Merl Miller* Il personal computer come professione
- 10 *Karen Billings e David Moursund* Te ne intendi di computer?
- 11 *Thomas Dwyer e Margot Critchfield* Il Basic e il personal computer, uno: introduzione
- 12 *Don Inman e Kurt Inman* Imparate il linguaggio dell'Apple

Don Inman e Kurt Inman

Imparate il linguaggio dell'Apple



franco muzzio & c. editore

Titolo originale *Apple Machine Language*
Traduzione di Antonio Filz
Copertina di Tomaso Boniolo
Redazione: Maddalena Redolfi

Prima edizione: giugno 1982
ISBN 88-7021-191-6

© 1982 franco muzzio & c. editore
Via Bonporti 36, 35100 Padova, tel. 049/661147-661873
© 1981 Reston Publishing Company, Reston, Virginia 22090
Tutti i diritti sono riservati

Indice generale

7 Prefazione	
9 1 Il Basic Applesoft II	BASIC
Comandi 11 Istruzioni di assegnazione 15 Istruzioni di visualizzazione 18 Istruzioni di ciclo e subroutine 21 Istruzioni per la grafica 23 Istruzioni di relazione 25 Precedenze nelle relazioni matematiche e logiche 25 Stringhe e funzioni 26 Istruzioni Basic particolarmente importanti 27 Esercizi 30 Risposte agli esercizi 31	
32 2 Attraversare il ponte	SOB
Uso della memoria 36 Un semplice sistema operativo in Basic 40 Il sistema operativo completo 54 Esercizi 56 Risposte agli esercizi 57	
58 3 Formato dei codici d'istruzione	SOB
Sistemi numerici 60 Accumulatore 66 Istruzioni di memoria 70 Uso del sistema operativo 71 Sommario 82 Esercizi 84 Risposte agli esercizi 85	
86 4 Semplice grafica	SOB
Disegno di un punto sul video 87 Disegno nei quattro angoli 94 Disegno di una linea orizzontale 97 Disegno di linee verticali 101 Disegno di un rettangolo 104 Sommario 107 Esercizi 109 Risposte agli esercizi 110	
111 5 Visualizzare testi	SOB
Visualizzazione di un carattere 112 Discussione delle nuove istruzioni 116 Un'occhiata all'alfabeto 123 Visualizzazione di codici	

- ASCII 126 Esecuzione del programma 129 Visualizzazione di più linee 131 Sommario 138 Esercizi 141 Risposte agli esercizi 142
- SOB** 143 **6 Il suono dell'Apple**
 Modifiche al sistema operativo in Basic 145 Descrizione del programma 146 Esecuzione del programma 149 Come lavorano le parti 2 e 3 156 Sommario 158 Esercizi 161 Risposte agli esercizi 162
- SOB** 164 **7 Ancora suoni e grafica**
 Combinazione dell'altoparlante e del video 164 Inserimento ed esecuzione del programma 168 Uso della tastiera per suonare le note 170 Descrizione del programma 173 Inserimento ed esecuzione del programma 177 Sommario 178 Esercizi 181 Risposte agli esercizi 182
- MS** 183 **8 Il monitor di sistema Apple**
 Il registro dis tato del microprocessore 191 Sottrazione 196 Sommario 201 Esercizi 202 Risposte agli esercizi 203
- MS** 204 **9 Precisione multipla e numeri negativi**
 Addizione a due byte 205 Sottrazione a due byte 209 Numeri negativi 212 Il gioco "Indovina il numero" 215 Sommario 22) Esercizi 228 Risposte agli esercizi 230
- MS** 231 **10 Ancora sul monitor**
 Addizione esadecimale - Modo immediato 231 Sottrazione esadecimale 235 Aritmetica decimale 237 Esaminare e modificare i registri 247 Sommario 253 Esercizi 255 Risposte agli esercizi 256
- MS** 258 **11 Mini-assembler e modi di indirizzamento**
 Uso del mini-assembler 260 Indirizzamento indicizzato 266 Indicizzazione in pagina zero 267 Indirizzamento assoluto indicizzato 271 Indirizzamento indicizzato diretto 276 Indirizzamento indiretto indicizzato 284 Sommario 288 Esercizi 290 Risposte agli esercizi 292
- 293 **12 Mettendo tutto assieme**
 Moltiplicazione a 8 bit 293 Moltiplicazione direttamente dal Basic 299 Moltiplicazione usando il sistema operativo in Basic 280 Moltiplicazione usando il monitor di sistema 304 Moltiplicazione usando il mini-assembler 305 Divisione a 8 bit 307 Sommario 312 Esercizi 314 Risposte agli esercizi 315
- 317 **Appendici**
 Istruzioni del Basic 317 Istruzioni del linguaggio macchina 318 Subroutine interne 319 Simboli 319 Programmi 320 Equivalenti esadecimali dei numeri decimali negativi 321 Memoria video 322 odici ASCII di schermo 324 Codici dei colori per grafici in bassa risoluzione 325 Codici delle istruzioni del 6502.

Prefazione

Lo scopo di questo libro è di introdurre gli utilizzatori del computer Apple, che conoscono il linguaggio Basic, alla programmazione in linguaggio macchina. Il passaggio dal Basic al linguaggio macchina è effettuato con piccoli e semplici passi. Per rendere i programmi dimostrativi interessanti e vivi, nel libro vengono usati molto presto i colori, la grafica ed il suono. Viene spiegata ogni nuova istruzione ed i programmi sono discussi passo per passo nelle loro parti funzionali.

Il lettore prima userà le istruzioni del Basic POKE, PEEK e CALL per introdurre ed eseguire programmi in linguaggio macchina dall'interno di un programma in Basic. Poi verrà sviluppato un sistema operativo scritto in Basic, con il quale si possono inserire ed eseguire questi programmi. Partire dal Basic, un linguaggio che il lettore già conosce, fornisce un approccio naturale che porta all'uso del monitor di sistema dell'Apple. Il monitor di sistema permette al lettore di inserire, esaminare ed eseguire direttamente programmi in linguaggio macchina. Il tempo impiegato dal computer per interpretare le istruzioni in Basic viene quindi eliminato.

Il passo finale consiste nell'utilizzo del mini-assembler dell'Apple, che libera il programmatore da molti dei noiosi dettagli concernenti la programmazione diretta in linguaggio macchina.

L'approccio al linguaggio macchina per mezzo del Basic dà modo al lettore di usare le sue precedenti conoscenze come base per l'esplorazione di una nuova area.

Si procederà nella lettura di questo libro in quattro fasi ben definite. I programmi in linguaggio macchina saranno introdotti ed eseguiti con quattro differenti metodi. Ad ogni stadio del libro verrà introdotto un metodo.

- | | |
|-------------------------------|--|
| 1. Basic | Introdotta nel Cap. 1. I programmi in linguaggio macchina sono sotto completo controllo del Basic, mediante le istruzioni POKE, CALL e PEEK. |
| 2. Sistema operativo in Basic | Introdotta nel Cap. 2. I programmi sono controllati da un sistema operativo scritto in Basic. |
| 3. Monitor di sistema | Introdotta nel Cap. 8. I programmi sono assemblati manualmente ed inseriti direttamente dal monitor di sistema dell'Apple. |
| 4. Mini-assembler | Introdotta nel Cap. 11. I programmi vengono assemblati dal mini-assembler dell'Apple. |

Le scritte **BASIC**, **SOB**, **MS**, **MA** appaiono nell'indice generale e all'inizio dei capitoli, per indicare che metodi vengono usati in un certo capitolo.

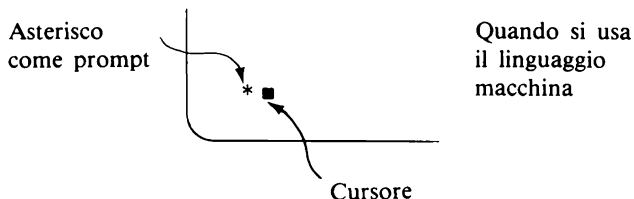
Il Basic Applesoft II

BASIC

Nella scrittura di questo libro sono state date per scontate parecchie cose. Gli autori ritengono che questo sia necessario a causa delle numerose versioni di computer Apple attualmente utilizzate.

1. Sono stati fatti i necessari collegamenti dell'hardware. Se non è così, consultate i manuali d'uso forniti con il calcolatore Apple.
2. Gli autori hanno usato una versione dell'Apple dotata di:
 - a. Basic Applesoft II su ROM.
 - b. Un interruttore sulla scheda per scegliere l'Applesoft II oppure l'Integer Basic.
3. Il lettore leggerà ed utilizzerà i manuali dell'Apple relativi alla sua particolare macchina.
4. Si conosce il modo di passare da un linguaggio di programmazione all'altro tra quelli a disposizione.

Il calcolatore Apple può usare parecchi linguaggi. Il carattere di *prompt* (pronto) indica che linguaggio può essere attualmente compreso dall'Apple. L'asterisco (*) avverte che si sta utilizzando il linguaggio macchina. Questo linguaggio risiede permanentemente nel computer e non richiede di essere "caricato" (inserito da un dispositivo esterno) da una cassetta o da un disco. Il monitor di linguaggio macchina, che controlla l'uso di questo linguaggio, è discusso nella seconda parte di questo libro (dal Cap. 8 in poi).



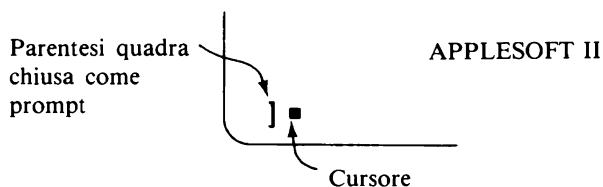
Se il vostro Applesoft è su ROM, l'Apple contiene anche un linguaggio ad alto livello orientato all'inglese chiamato Integer Basic e memorizzato permanentemente nella memoria ROM del calcolatore. La memoria ROM, che sta per "*Read Only Memory*" (memoria a sola lettura), può essere "letta" (usata dai vostri programmi), ma non è possibile "scriverci" (cioè cambiarla). Il carattere di prompt per l'Integer Basic è la freccia che punta a destra (\rightarrow). L'Integer Basic *non* è discusso in questo libro. Per maggiori informazioni, consultare l'*Apple II Basic Programming Manual* (prodotto Apple n. 2L005X).

L'Applesoft II è il linguaggio Basic esteso dell'Apple. Il carattere di prompt dell'Applesoft II è una parentesi quadra chiusa (]). Questo Basic esteso è ora disponibile in tre forme:

1. il sistema Apple II Plus con la ROM di monitor a caricamento automatico;
2. la scheda di interfaccia Applesoft;
3. l'Apple Language System.

Il sistema Apple II Plus ha il Basic Applesoft II su ROM. Perciò il mini-assembler, il *Floating Point Package* e l'interprete SWEET-16 (che sono memorizzati nelle ROM dell'Integer Basic) non sono disponibili sul sistema Apple II Plus.

Dato che, successivamente nel libro, sarà utilizzato il mini-assembler, ci focalizzeremo sul sistema contenente la scheda con le ROM dell'Applesoft II piuttosto che sul sistema Apple II Plus.



Questo libro serve come ponte per il lettore, nel passaggio dalla programmazione in Basic a quella nel linguaggio macchina originale del calcolatore. Il libro presuppone una conoscenza del Basic Applesoft II, ma

in questo capitolo verranno ripassate brevemente le istruzioni del Basic usate dal computer Apple. Se vi sentite sicuri della vostra conoscenza del Basic Applesoft II, passate tranquillamente al Cap. 2. Tuttavia, se le vostre basi non sono buone, rivedete ora i punti fondamentali.

Sebbene la parte che segue non rappresenti una trattazione completa delle possibilità dell'Applesoft, troverete qui tutte le istruzioni ed i comandi necessari per la comprensione dei rimanenti capitoli del libro. Un calcolatore Apple con 16 K di RAM, un registratore o un floppy disk, ed il Basic Applesoft costituiscono tutto il necessario per eseguire gli esempi dimostrativi e gli esercizi presentati.

COMANDI

Certi comandi fondamentali sono necessari nella preparazione, correzione ed esecuzione di un programma. Quelli qui discussi sono NEW, LIST, RUN, TEXT, GRaphics, LOAD, SAVE, CONTinue, TRACE e NOTRACE.



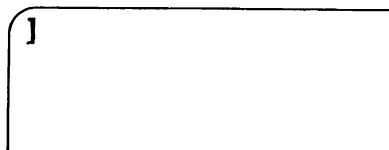
NEW Cancella ogni vecchio programma che si trova nella memoria del calcolatore. Non solo elimina il programma corrente, ma azzera anche tutte le variabili a cui questo programma ha assegnato dei valori. Viene usato prima di inserire un nuovo programma.

Esempio:

```
10 LET M = 50
20 PRINT M
30 LET M = M + 1
40 IF M < 60 THEN GOTO 20
50 END
```

] NEW ← Quando scrivete questo e
premete il tasto RETURN

PRESTO! È SCOMPARSO TUTTO!



LIST Fa sì che il programma corrente sia visualizzato sullo schermo. Negli esempi sono mostrate parecchie versioni di questo comando. Tutte presuppongono che ci sia un programma nel computer.

Esempi

1. Scrivete **LIST** e premete il tasto **RETURN**.
Verrà visualizzato l'intero programma. Se questo è molto lungo, quando lo schermo sarà pieno il testo verrà fatto scorrere verso l'alto.
2. Scrivete **LIST 20,100**
oppure
LIST 20-100 e premete il tasto **RETURN**.
Saranno visualizzate le linee di programma comprese tra la 20 e la 100.
3. Scrivete **LIST -150** e premete **RETURN**.
Veranno visualizzate tutte le linee dall'inizio del programma fino alla linea 150.
4. Scrivete **LIST 150-** e premete **RETURN**.
Saranno listate tutte le linee a partire dalla linea 150 fino alla fine del programma.
5. Scrivete **LIST 150** e premete **RETURN**.
Questo causa la visualizzazione della sola linea 150.

Per fermare momentaneamente la lista di un programma ad un certo punto, tenete premuto il tasto **CTRL** (control) e battete la lettera **S**. Usate ancora **CTRL S** per far ripartire la stampa. Questo vi permetterà di esaminare delle parti di programma. Un comando **LIST** può essere annullato mediante un **CTRL C**, ma in questo caso non è più possibile far riprendere la visualizzazione, a meno che non si utilizzi un comando **LIST** che faccia ripartire la visualizzazione dal punto in cui era stata fermata.

RUN Permette l'esecuzione del programma che risiede attualmente nella memoria. Tutte le variabili vengono azzerate e l'esecuzione inizia dalla linea con il numero più basso (a meno che, dopo la parola **RUN**, non venga specificato il numero di linea di partenza, come nell'Esempio 2).

Esempi

1. Scrivete RUN e premete RETURN.

Il programma sarà eseguito a partire dalla linea con il numero più basso.

2. Scrivete RUN 200 e premete RETURN.

Il programma verrà eseguito partendo dalla linea 200.

TEXT Questo comando predispone lo schermo alla visualizzazione di testi con un massimo di 40 caratteri per riga e con 25 righe. Questo è il formato normale usato quando si sta utilizzando il Basic Applesoft II. Il comando TEXT viene usato quando si vuole ritornare da un modo grafico al modo *text*. Può essere anche usato come istruzione in un programma.

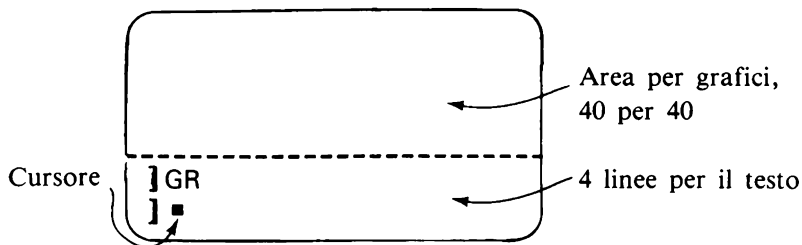
Esempio

Scrivete TEXT e premete RETURN.

GR Predispone il video per i grafici a bassa risoluzione. Con questo comando si ha a disposizione una quadrettatura 40×40 dello schermo per fare grafici. Il video viene cancellato ed appare a sfondo nero. Il cursore è portato all'inizio della finestra per 4 linee di testo posta in fondo allo schermo. Il colore da usare per i grafici è posto automaticamente sul nero (COLOR = 0). Per realizzare dei grafici, bisogna dare a COLOR qualche altro valore (nero su nero non è molto visibile).

Esempio

Scrivete GR e premete RETURN



LOAD Questo comando causa la lettura di un programma da una cassetta nella memoria del calcolatore. Il registratore deve essere pronto (posto all'inizio del programma desiderato e in posizione PLAY) *prima* che il comando LOAD venga dato. Un "bip" indica che l'Apple ha trovato l'informazione sul nastro. Un secondo bip annuncia che il programma è stato letto correttamente. A questo

punto apparirà sul video il prompt dell'Applesoft. La lettura di un programma può essere interrotta solamente premendo il tasto RESET, oppure togliendo la corrente.

Esempio

Preparate il registratore, poi scrivete LOAD e premete RETURN. Finita la lettura, il video mostrerà:



```
] LOAD
] ■
```

Indica la corretta lettura del programma

SAVE Scrive il programma attualmente in memoria sulla cassetta. Si devono premere i tasti RECORD e PLAY del registratore *prima* dell'esecuzione del comando SAVE. Due bip segnalano l'inizio e la fine del procedimento SAVE.

Esempio

Preparate il registratore e poi scrivete SAVE e premete RETURN.

CONT Se l'esecuzione di un programma è stata fermata mediante STOP, END o CTRL C, questo comando la fa ripartire dall'istruzione successiva a quella alla quale era stata interrotta. Non viene cancellato niente. CONT non può essere usato se avete (1) modificato, aggiunto o cancellato una qualsiasi linea di programma, o (2) ricevuto un messaggio di errore, che ha fermato l'esecuzione.

Esempio

Scrivete CONT e premete RETURN.

TRACE Serve nella correzione dei programmi. Fa apparire sul video i numeri di linea man mano che le linee vengono eseguite. Potete quindi vedere se il programma sta eseguendo la successione di operazioni desiderate. La scelta TRACE viene disabilitata con il comando NOTRACE.

Esempio

Scrivete TRACE e premete RETURN. Poi scrivete RUN per vedere l'ordine di esecuzione delle linee.

NOTRACE Disabilita il comando TRACE discusso sopra.

Esempio

Scrivete NOTRACE e premete RETURN.

Quando il programma sarà eseguito nuovamente, non verrà stampato nessun numero di linea.

ISTRUZIONI DI ASSEGNAZIONE

Ci sono parecchi modi di assegnare dati (sia numerici che stringhe) alle variabili. Le istruzioni che svolgono questo compito, discusse in questo paragrafo, sono LET, INPUT, READ e GET. Sono anche discusse le istruzioni DATA e RESTORE; esse vengono usate assieme all'istruzione READ.

LET Questa istruzione può essere usata per assegnare dei valori alle variabili. La parola LET è facoltativa, come si può vedere negli esempi alle linee 50 e 60.

Esempi

10 LET M = 50	← Assegna un valore numerico a M
20 FOR X = 1 TO 9	
30 LET A\$ = "MELE"	← Assegna la stringa MELE. ad A\$
40 LET M = M + 1	← Modifica il valore di una variabile
50 B = 1	← Non occorre usare la parola LET.
60 B\$ = "CASSETTA"	← È facoltativa.
70 PRINT M;A\$,B;B\$	
80 NEXT X	

INPUT Serve per assegnare un valore ad una variabile durante l'esecuzione di un programma. Quando il calcolatore incontra questa istruzione, si ferma ed aspetta che l'utente scriva il valore da assegnare alla variabile.

Esempi

50 INPUT A

? ■

L'esecuzione di questa istruzione visualizza un punto interrogativo ed aspetta che l'utente scriva il valore e prema RETURN.

70 INPUT A,B,C,

Si può assegnare all'istruzione INPUT più di una variabile. I valori saranno inseriti separati da una virgola.

80 INPUT"SCRIVI IL TUO NOME, PER FAVORE";C\$

Può essere stampato un messaggio per dirti che dato è richiesto. Il messaggio va racchiuso tra virgolette. La variabile C\$ richiede l'input di una stringa. In questa forma, l'istruzione INPUT non stampa il punto interrogativo.

SCRIVI IL TUO NOME, PER FAVORE ■

90 INPUT A\$

?■

Nessun messaggio questa volta. Quando l'istruzione sarà eseguita, sul video apparirà un punto interrogativo.

100 INPUT"QUAL E' IL VALORE DI A?"; A

Se volete il punto interrogativo quando fate stampare un messaggio, inseritelo all'interno delle virgolette.

QUAL E' IL VALORE DI A?■

READ Dice al computer di leggere un valore contenuto in un'istruzione DATA e di assegnarlo ad una variabile. La prima volta che viene eseguita un'istruzione READ, sarà usato il primo dato contenuto nella prima istruzione DATA. La seconda volta, verrà preso il secondo dato fra quelli contenuti nelle istruzioni DATA, ecc. (Per degli esempi vedere DATA.)

DATA Permette di memorizzare dei dati all'interno del programma. I dati saranno letti (con un'istruzione READ) sequenzialmente. In un programma si possono usare più istruzioni DATA. I dati saranno letti dalla prima istruzione DATA finché questa non ne conterrà più. Poi verranno letti quelli della seconda istruzione DATA, e così via.

Esempio

```

110 FOR X = 1 TO 10
120   READ Y           ← Legge i dati in questo ordine:
130 NEXT X              10
140 DATA 10,30,20,40,50 30
150 DATA 60,80,90,70,100 20
                        40
                        50
                        60
                        80
                        90
                        70
                        100

```

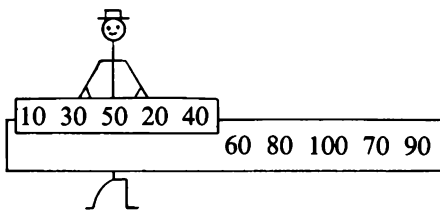
RESTORE Fa sì che la prossima istruzione READ eseguita legga il primo dato nella prima istruzione DATA.

Esempio

```

100 FOR X = 1 TO 5
110   READ Y: PRINT Y   ← Legge e stampa
                        10,30,50,20,40
120 NEXT X
130 RESTORE             ← Ritorna all'inizio della lista dei
                        dati.
140 FOR Z = 1 TO 10
150   READ W: PRINT W   ← Legge e stampa
                        10,30,50,20,40,60,80,100,70,90
160 NEXT Z
170 DATA 10,30,50,20,40
180 DATA 60,80,100,70,90

```



RESTORE

GET Questa istruzione prende (o legge) un singolo carattere dalla tastiera. Il calcolatore aspetta la pressione di un tasto, come in un'istruzione INPUT. Il carattere *non viene visualizzato e non occorre* premere il tasto RETURN.

Esempio

```

200 GET H$
210 IF H$ = "S" THEN GOTO 500
220 GOTO 100

```

La linea 200 aspetta che venga battuto un tasto. Il carattere scritto sarà memorizzato come variabile H\$. Se fosse una S, dopo la linea 210 si passerebbe alla 500. Altrimenti, l'esecuzione proseguirà alla linea 100.

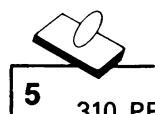
ISTRUZIONI DI VISUALIZZAZIONE

L'istruzione **PRINT** è usata in molte forme per visualizzare dati sul video. La stampa può essere anche cambiata da bianco su nero a nero su bianco mediante l'istruzione **INVERSE**. È anche possibile alternare questi due formati usando l'istruzione **FLASH**. L'istruzione **NORMAL** riporta il video al formato normale bianco su nero. **HOME** serve per cancellare il video. **SPC** viene usata per la spaziatura nelle stampe.

Esempi

320 PRINT

La parola **PRINT** usata da sola fa sì che sullo schermo siano eseguite le istruzioni di "nuova linea" (*line feed*) e "ritorno carrello" (*return*) (vedere la linea 320 dell'esempio).



310 PRINT A

Stampa il valore di A e causa un line feed ed un return (vedere la linea 310 dell'esempio).

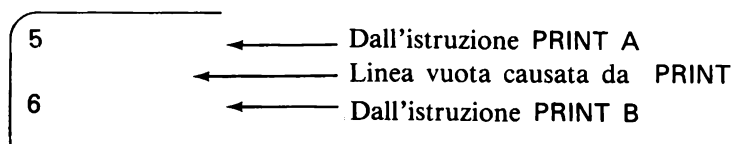
300 A = 5: B = 6

310 PRINT A

320 PRINT

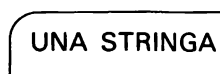
330 PRINT B

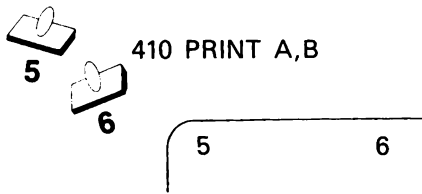
Risultato dell'esecuzione di queste linee:



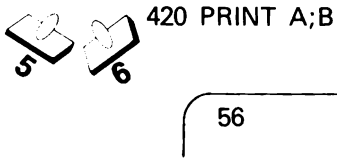
400 PRINT "UNA STRINGA"

Stampa le parole UNA STRINGA e passa alla prossima linea.



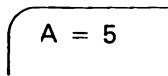


Se $A = 5$ e $B = 6$, causa la stampa dei due valori sulla stessa linea ben separati.



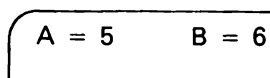
Questa volta i valori saranno stampati sulla stessa linea ma molto vicini.

430 PRINT A\$;A



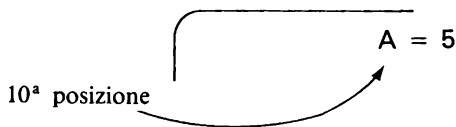
Se $A\$ = "A ="$ e $A = 5$, questa istruzione stamperà la stringa $A =$ ed il valore di A .

440 PRINT A\$;A,
450 PRINT B\$;B



Se $A\$ = "A ="$ e $B\$ = "B ="$ con $A = 5$ e $B = 6$, il risultato di queste due linee sarà quello mostrato. La virgola mantiene la stampa sulla stessa linea.

500 PRINT TAB(10)A\$;A



La funzione TAB sposta la posizione attuale di stampa alla colonna specificata. (Le posizioni di stampa su una data linea vanno da 1 a 40.)

FLASH Questa istruzione abilita il video al modo lampeggiante. L'output è mostrato alternativamente come bianco su nero e nero su bianco. Si usi l'istruzione **NORMAL** per ritornare al modo non lampeggiante.

Esempio

70 FLASH

80 PRINT "LAMPEGGIO" ← La parola LAMPEGGIO apparirà lampeggiante.

INVERSE Fa sì che l'output sia visualizzato a caratteri neri su sfondo bianco.

Esempio

100 INVERSE

110 PRINT"INVERSIONE" ← La parola INVERSIONE apparirà a
lettere nere su sfondo bianco.

NORMAL Fa tornare il video al modo normale con caratteri bianchi su sfondo nero.

Esempio

70 FLASH

80 PRINT "LAMPEGGIO" ← La parola LAMPEGGIO lampeggerà e rimarrà tale.

100 NORMAL

110 PRINT" NORMALE" ← La parola NORMALE, e tutto ciò che sarà stampato dopo, apparirà nel modo normale (a meno che non si usino ancora le istruzioni FLASH o INVERSE).

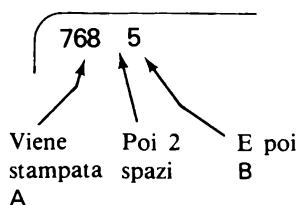
HOME Questa istruzione sposta il cursore nella posizione in alto a sinistra all'interno della finestra per il testo. Cancella anche tutto il testo contenuto nella finestra. **HOME** può essere usata nei modi **text** o **grafico**.

SPC(X) Inserisce X spazi tra l'ultimo dato stampato ed il prossimo da stampare se l'istruzione **SPC** è preceduta e seguita da un punto e virgola. È usata *soltanto* all'interno di istruzioni **PRINT**.

Esempio

```
400 PRINT A;SPC(2);B
```

Se $A = 768$ e $B = 5$, la linea 400 (quando eseguita) visualizzerà:



ISTRUZIONI DI CICLO E SUBROUTINE

Alcune parti di un programma possono essere ripetute utilizzando certe istruzioni del Basic, come GOTO, ON...GOTO, IF...THEN e FOR...NEXT. Le subroutine possono essere eseguite mediante GOSUB e RETURN.

GOTO Fa saltare l'esecuzione dalla linea in cui si trova il GOTO a quella specificata dopo la parola GOTO.

Esempi

```

30 GOTO 200
.
.
.
70 A = 73
80 PRINT A
90 END
.
.
.
200 PRINT "FINE"
210 GOTO 70

```

← Tutte le linee tra la 30 e la 200 vengono saltate.

← Il programma salta dalla linea 210 alla linea 70

ON... GOTO 100,200,300... Questa istruzione valuta l'espressione aritmetica che segue la parola ON. Poi salta al numero di linea (100, 200, 300,...) in base al risultato della valutazione. 100,200,300, ecc., devono essere numeri di linea presenti nel programma.

Esempio

```

.
.
.
150 ON INT(B/100) GOTO 200,300,400
160 PRINT "INT(B/100) E' 0 O >3"
.
.

```

Viene calcolata l'espressione $\text{INT}(B/100)$. Poi:

- se il risultato è = 1, dopo la linea 150, sarà eseguita la 200;
- se il risultato è = 2, dopo la linea 150, sarà eseguita la 300;
- se il risultato è = 3, dopo la linea 150, sarà eseguita la 400;
- se il risultato è = 0 o è >3, dopo la linea 150, sarà eseguita la 160.

RETURN Questa istruzione viene usata alla fine di una subroutine, per ritornare all'istruzione che segue immediatamente la più recente istruzione GOSUB eseguita. (Vedere l'esempio sopra.)

ISTRUZIONI PER LA GRAFICA

Le istruzioni per la grafica qui illustrate sono GGraphics, COLOR, PLOT, HLIN, VLIN, PDL e TEXT. Il modo grafico presenta un video che è piuttosto differente da quello del modo text. Si deve essere in grado di passare da un modo all'altro.

GR Abilita il modo grafico in bassa risoluzione. (Vedere GR nel paragrafo "Comandi".)

COLOR Sceglie il colore per grafici in bassa risoluzione. L'istruzione GR sceglie automaticamente il colore nero (0). I valori dei colori sono:

0 nero	4 verde scuro	8 marrone	12 verde
1 magenta	5 grigio	9 arancio	13 giallo
2 blu scuro	6 blu	10 grigio	14 acquamarina
3 viola	7 azzurro	11 rosa	15 bianco

PLOT Illumina uno dei 40 per 40 punti in bassa risoluzione dell'area grafica alla colonna e riga specificate. Il punto sarà del colore scelto con l'istruzione COLOR. Le tre istruzioni sono usate assieme.

Esempio

10 GR ← Abilita il modo grafico

20 COLOR = 9 ← Usa l'arancio

30 PLOT 20,30 ← Stampa un punto alla colonna 20, riga 30

HLIN Serve per disegnare una linea orizzontale. Vengono specificate le colonne iniziale e finale e la riga dove la linea va tracciata.

Esempio:

10 GR

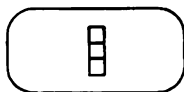
20 COLOR = 4 ← Colore verde scuro

30 HLIN 10,20 AT 30 ← Disegna alla riga 30

← Finisce alla colonna 20

← Inizia alla colonna 10

VLIN Disegna una linea verticale da una riga ad un'altra alla colonna specificata.



Esempio

```
10 GR
20 COLOR = 11
30 VLIN 6,14 AT 12
```

← Colore rosa
← Disegna alla colonna 12
← Finisce alla riga 14
← Inizia alla riga 6

PDL(0) o PDL(1) Legge il valore corrente di uno dei controlli per giochi (un numero da 0 a 255). Le *"paddle"* (controlli per giochi) possono essere usate per stampare punti nel modo grafico a bassa risoluzione, come mostrato sotto. (Possono anche essere utilizzate per grafici ad alta risoluzione.)

Esempio

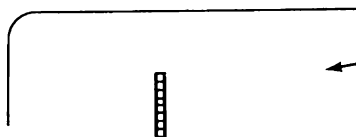
```
10 GR
20 COLOR = 14
30 PLOT PDL(0)/7, PDL(1)/7
```

← Riga
← Colonna

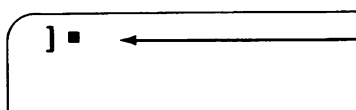
TEXT Questa istruzione è usata per tornare al modo TEXT dopo aver usato il modo grafico ad alta o bassa risoluzione.

Esempio

```
10 GR
20 COLOR = 13
30 VLIN 6,14 AT 12
40 FOR X = 1 TO 3000 }
50 NEXT X              ← Ritardo per vedere la barra colorata
60 TEXT                ← Ritorno al modo text
70 HOME                ← Cursore in alto a sinistra
```



← Per alcuni secondi vedrete la barra color giallo



← Poi lo schermo tornerà al modo text ed il cursore comparirà nell'angolo in alto a sinistra.

ISTRUZIONI DI RELAZIONE

Due valori possono essere confrontati usando una delle possibili istruzioni di relazione. Il risultato di questo confronto può essere usato dal calcolatore per "prendere una decisione", come ad esempio che cosa fare dopo.

RELAZIONI MATEMATICHE

= è uguale a
> è maggiore di
< è minore di
>= è maggiore o uguale a
<= è minore o uguale a
non è uguale a

RELAZIONI LOGICHE

AND Vera se entrambe le condizioni sono vere, altrimenti falsa.

OR Vera se una o entrambe le condizioni sono vere; altrimenti falsa.

NOT Negazione dell'espressione.

Esempi

200 IF A>5 THEN GOTO 340

300 IF A#B THEN PRINT "A NON E' UGUALE A B"

400 IF A = 5 AND B >= 6 THEN C = A + B

PRECEDENZE NELLE RELAZIONI MATEMATICHE E LOGICHE

Il computer valuta delle espressioni eseguendo le operazioni in un ordine specifico. L'ordine in cui esegue queste operazioni è in accordo con la seguente lista. Il loro ordine di precedenza è dall'alto verso il basso.

<i>Ordine</i>	<i>Operazione</i>	<i>Funzione</i>
1.	()	Valuta l'espressione tra parentesi
2.	NOT	Nega
3.	^	Eleva a potenza (esponenzia)
4.	*, /	Moltiplica o divide (da sinistra a destra)

- | | | |
|----|---------------------|--|
| 5. | +, - | Somma o sottrae (da sinistra a destra) |
| 6. | =, >, <, >=, <=, '# | Confronta |
| 7. | AND | Fa l'AND tra due espressioni |
| 8. | OR | Fa l'OR tra due espressioni |

STRINGHE E FUNZIONI

Si usano parecchie istruzioni per manipolare stringhe. Spiegheremo solo quelle che saranno usate in questo libro, e cioè ASC, CHR\$ e LEFT\$. Sono anche disponibili parecchie funzioni interne, ma noi useremo soltanto la funzione INTeger.

ASC Questo comando ritorna (fornisce) il codice decimale ASCII del primo carattere della stringa chiusa tra parentesi dopo le lettere ASC.

Esempio

```

100 PRINT ASC("S")
.
.
.
190 GET H$
200 IF ASC(H$)<60 THEN GOTO 100
210 ...

```

Stampa 83 (il codice ASCII della lettera S)

← Prende un singolo carattere

Se il codice ASCII del carattere scritto è < 60, va alla linea 100, altrimenti alla 210

CHR\$ Ritorna (fornisce) il carattere che corrisponde al valore contenuto fra parentesi. Questo valore deve essere compreso tra 0 e 255 inclusi.

Esempio

```

300 H = 14
310 PRINT CHR$(H + 55)

```

Se H = al numero decimale 14, allora H + 55 = 69. Il carattere il cui codice ASCII è 69 è la lettera E. Verrà stampata questa lettera.

LEFT\$ Ritorna (fornisce) il numero specificato di caratteri più a sinistra nella stringa racchiusa tra parentesi. Se non viene specificato nessun numero, ritorna il primo carattere della stringa.

Esempi

```
200 PRINT LEFT$( "YESTERDAY", 3)
```

```
.
.
.
.
```

Stamperà: YES (i 3 caratteri più a sinistra di YESTERDAY)

```
250 INPUT H$
```

```
260 IF LEFT$(H$) = "Y" THEN GOTO 100
```

```
270 ...
```

Se il primo carattere della stringa in input per H\$ è una Y, allora passerà alla linea 100. Altrimenti andrà alla linea 270.

INT La funzione **INT**eger ritorna il più grande intero minore o uguale dell'espressione che segue, tra parentesi, la parola **INT**.

Esempi

```
100 INT (A/3)
```

Se $A = 5$, $A/3 = 1.66667$ e $\text{INT}(A/3) = 1$

Se $A = 1$, $A/3 = 0.333333$ e $\text{INT}(A/3) = 0$

Se $A = 15$, $A/3 = 5$ e $\text{INT}(A/3) = 5$

Se $A = -5$, $A/3 = -1.66667$ e $\text{INT}(A/3) = -2$

ISTRUZIONI BASIC PARTICOLARMENTE IMPORTANT

Ci sono tre istruzioni che userete sempre più spesso nel costruire il ponte tra il Basic ed i programmi in linguaggio macchina che creerete. Queste istruzioni sono **POKE**, **PEEK**, e **CALL**.

In Basic, i numeri di linea servono come riferimento per il calcolatore. Le singole istruzioni vengono trovate ed eseguite secondo il numero di linea associato ad ogni istruzione.

Le istruzioni in linguaggio macchina sono eseguite in base alla loro posizione in memoria. Non ci sono numeri di linea. L'esecuzione inizia ad una locazione di memoria che deve essere specificata. Poi le istruzioni sono normalmente eseguite nell'ordine in cui appaiono in memoria. Il sistema operativo Basic (descritto nel Cap. 2) è usato per mettere le istruzioni in linguaggio macchina ed i dati nelle corrette locazioni di memoria che saranno usate dal programma in linguaggio macchina. Questo viene fatto principalmente dalla seguente istruzione Basic.

POKE indirizzo, dato

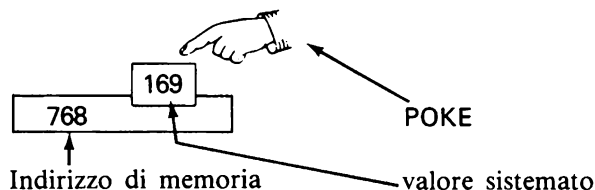
Dove l'indirizzo è quello decimale della locazione di memoria dove va posto il dato.

Poiché POKE è un'istruzione del Basic, i valori dell'indirizzo e del dato devono essere specificati come valori decimali.

Esempi

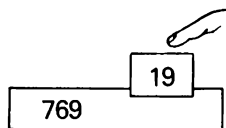
100 POKE 768, 169

Pone il valore 169 nella locazione di memoria 768



110 POKE 769, 19

Pone il valore 19 nella locazione di memoria 769



Ora abbiamo:

768	169
769	19

Indirizzi di memoria Dati posti in memoria

Ogni istruzione in linguaggio macchina ed ogni dato usato nel programma in linguaggio macchina sarà inserito dal sistema operativo del Basic con un'istruzione POKE.

Quando il programma in linguaggio macchina ed i dati sono stati inseriti dalle istruzioni POKE del Basic, il controllo deve essere passato dal Basic al programma in linguaggio macchina. Questo viene fatto con l'istruzione:

CALL indirizzo

Chiama l'esecuzione di un programma in linguaggio macchina (o una subroutine)

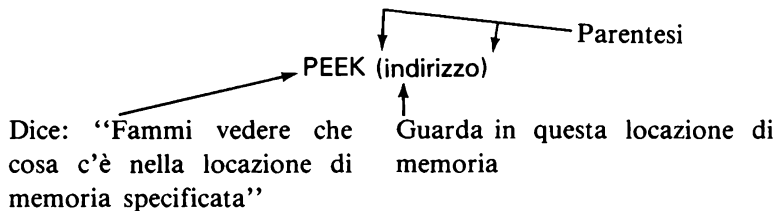
L'indirizzo decimale dove inizia il programma in linguaggio macchina



Esempio**CALL 768**

← Questo farà sì che il computer esegua il programma in linguaggio macchina che inizia alla locazione di memoria 768

Una terza istruzione del Basic, che userete spesso, vi permette di esaminare il contenuto di una specifica locazione di memoria. Potete esaminare una locazione di memoria con l'istruzione:



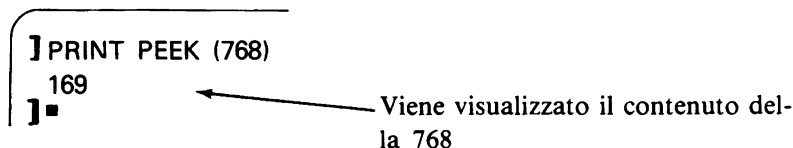
Se volete vedere sul video il contenuto della locazione, usate l'istruzione **PRINT**.

Esempio**900 PRINT PEEK (768)**

Questa istruzione farà apparire sul video il valore contenuto nella locazione di memoria 768.

768	169
-----	-----

Se 169 è nella memoria 768 ed eseguiamo l'istruzione **PRINT PEEK** nel modo immediato, vedremo questo sul video.



L'istruzione **PEEK** può essere usata per esaminare il programma in linguaggio macchina stesso, oppure per visualizzare il risultato della sua esecuzione, che è stato posto in una locazione di memoria.

Queste tre istruzioni (**POKE**, **CALL** e **PEEK**) saranno usate ripetutamente per stabilire un legame tra Basic e linguaggio macchina. Il programma in

linguaggio macchina viene posto in memoria per mezzo del Basic. Viene poi eseguito, dal Basic, mediante l'istruzione CALL. L'istruzione PEEK può essere usata per vedere i risultati o il programma stesso.

Farete spesso assegnamento su queste tre istruzioni del Basic. Se si vuole stabilire una solida relazione tra Basic e linguaggio macchina, si devono capire a fondo queste istruzioni fondamentali. Saranno usate ancora nel Cap. 2, dove è discusso il sistema operativo del Basic.

ESERCIZI

1. Dite che funzione ha ognuno dei seguenti comandi.
 - a. NEW _____

 - b. LIST _____

 - c. GR _____

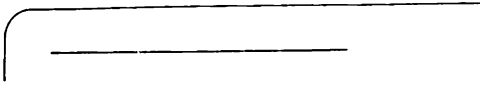
2. Il comando TRACE causa la visualizzazione dei numeri di linea mentre un programma viene eseguito. Quale comando lo disabilita? _____
3. Scrivete tre istruzioni del Basic usate per assegnare valori alle variabili.
 - a. _____
 - b. _____
 - c. _____
4. Se state attualmente usando la grafica a bassa risoluzione, e volete tornare al modo normale per i testi con il cursore in alto a sinistra, quali due comandi dovete dare?
 - a. _____
 - b. _____
5. Alcune delle operazioni mostrate sotto non sono nel corretto ordine di precedenza. Disponetele nell'ordine giusto.
() _____
*, / _____
> = _____
AND _____
NOT _____

6. Mettete i valori nelle corrette posizioni di memoria come indicato dalle seguenti istruzioni.

100 POKE 768,19	768	
110 POKE 770,14	769	
120 POKE 772,18	770	
130 POKE 771,15	771	
140 POKE 769,16	772	

7. Che cosa verrà visualizzato sullo schermo quando saranno eseguite le istruzioni dell'esercizio 6 e quella riportata qui sotto?

150 PRINT PEEK(772)



RISPOSTE AGLI ESERCIZI

1. a. NEW Elimina ogni precedente programma e cancella tutte le variabili.
- b. LIST Visualizza il programma corrente sul video.
- c. GR Abilita il modo grafico in bassa risoluzione.
2. NOTRACE
3. a. LET A = 5 (o soltanto A = 5)
- b. INPUT A
- c. READ A
4. a. TEXT
- b. HOME
5. ()
- NOT
- *, /
- >=
- AND

6.

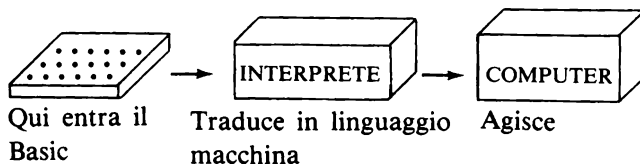
768	19
769	16
770	14
771	15
772	18

7. 18

Attraversare il ponte

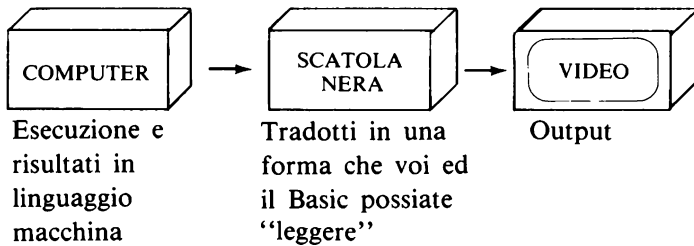
SOB

Quando comunicate con il computer in Basic, state parlando attraverso un interprete. Ogni linea di programma deve essere esaminata in dettaglio dall'interprete e tradotta in un codice che il calcolatore possa comprendere. Per voi è facile scrivere programmi in Basic, che però è una lingua "straniera" per il computer. Il calcolatore non può capire nemmeno la più semplice delle istruzioni Basic. Le parole e le istruzioni del Basic devono essere tradotte in codici numerici binari che hanno un preciso significato per il computer. Questi codici numerici sono "parole" che il calcolatore può capire. Essi sono il linguaggio del computer, chiamato *linguaggio macchina*. Le istruzioni devono essere in codice di linguaggio macchina prima che il calcolatore le possa comprendere.



Quando le istruzioni in Basic sono state interpretate, il computer le esegue. I risultati intermedi e finali devono essere tradotti ancora una volta

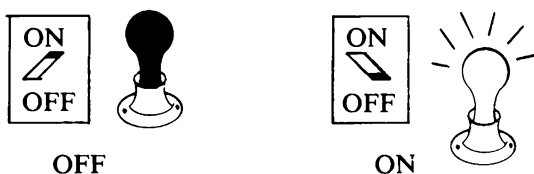
in una forma che il Basic possa usare e che voi siate in grado di capire facilmente.



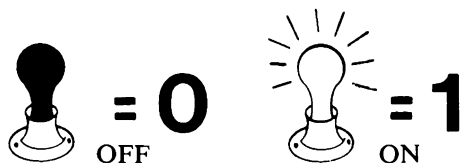
La traduzione del Basic è un lavoro che richiede del tempo all'interprete, ed è uno spreco di tempo per il computer. Inoltre, in Basic, non è sempre possibile specificare quello che si vuole che il calcolatore faccia.

Sebbene la programmazione in linguaggio macchina sia un compito più lungo e dettagliato rispetto a quella in Basic, vi porterà ad un contatto più stretto con il computer. Quando usate il linguaggio macchina, state comunicando con il computer direttamente. Otterrete risposte rapide ed avrete una miglior comprensione della "personalità" del calcolatore, delle sue reali possibilità ed anche dei suoi difetti. Scoprirete che il computer parla e comprende un linguaggio formale molto limitato. Ogni parola è della stessa lunghezza ed ha un formato rigido. Ma le regole di forma e sintassi del linguaggio macchina sono molto più semplici di quelle della lingua italiana.

Le parole del linguaggio macchina possono essere suddivise in otto bit (*binary digit* = cifra binaria) che hanno solo due possibili stati (o condizioni). Questi piccoli bit sono molto simili ad una lampadina, che può essere accesa o spenta.



Il computer interpreta questi bit come se fossero uno dei due simboli numerici 0 o 1. Una disposizione di 1 e 0 è per il computer una parola con significato, o un'idea completa. Perciò, se vogliamo comunicare direttamente con il calcolatore, dobbiamo imparare queste parole.



Un esempio di disposizione di 8 bit (cioè uno schema con grandezza e forma che il computer può capire) è questo.

0 **1** **0** **1** **1** **0** **1** **1**
 OFF ON OFF ON ON OFF ON ON

Il computer riconoscerà questa disposizione come un unico codice numerico e risponderà svolgendo una particolare azione o usando il numero come uno specifico dato.

Poiché le istruzioni del linguaggio macchina sono puramente dei numeri posti nella memoria del calcolatore, possiamo usare il Basic per effettuare questa operazione. L'istruzione del Basic:

POKE indirizzo, dato

memorizzerà il dato nella locazione di memoria stabilita dall'indirizzo. (Potete rivedere questa istruzione al Cap. 1.)

Esempio

POKE 768, 173

Il dato è posto nella
locazione di memoria 768

Il dato 173 è posto in memoria

Il dato contenuto nell'istruzione POKE deve stare tra 0 e 255, a causa della natura delle locazioni di memoria del computer. Numeri più grandi richiedono più di una locazione. Se tentate di inserire in memoria un numero maggiore di 255, il computer non accetterà l'istruzione e risponderà semplicemente con **ILLEGAL QUANTITY ERROR**.

Esempio

POKE 769,256

?ILLEGAL QUANTITY ERROR

]■

Troppo grande

È importante stare attenti quando si inseriscono valori in memoria. Se si utilizza una locazione sbagliata è possibile eliminare valori o istruzioni essenziali.

L'Applesoft II possiede un'istruzione che permette di vedere il valore decimale del contenuto di una certa locazione di memoria. Essa è:

PEEK (indirizzo)


Per visualizzare il contenuto di un dato indirizzo, potete usare l'istruzione 200 PRINT PEEK (768). Il computer stamperà il valore decimale (fra 0 e 255) contenuto nella locazione il cui indirizzo è 768.

L'Apple, quindi, vi permette di mettere dei valori nella sua memoria e di vedere quelli che già contiene. Cercate di familiarizzare con queste due istruzioni nel modo esecuzione immediata. Provate i seguenti esempi ed altri che potete inventare.

Esempi

Prima POKE (inserire) 

```
] POKE 768,173
] POKE 769,25
] POKE 770,3
] ■
```

Poi PEEK (guardare) 

```
] POKE 768,173
] POKE 769,25
] POKE 770,3
] PRINT PEEK(768)  Vedete, il valore 173 è nella loca-
173 ←              zione di memoria 768
] PRINT PEEK (769)  Il valore 25 è nella locazione 769
25 ←
] PRINT PEEK (770)  E il valore 3 è nella locazione 770
3 ←
] ■
```

I valori decimali che possono essere usati come indirizzo nelle istruzioni POKE e PEEK dipendono dalla capacità di memoria del calcolatore.

USO DELLA MEMORIA

Molti indirizzi di memoria non possono essere usati per programmi in linguaggio macchina, poiché contengono informazioni necessarie al sistema operativo dell'Apple. Il sistema operativo può essere paragonato alla torre di controllo di un aeroporto che dirige il flusso di traffico che viene e parte dall'aeroporto. Il sistema operativo dirige il flusso di operazioni fatte dal computer. Una POKE nelle locazioni del sistema operativo può alterare la funzionalità del sistema o del vostro programma in Basic.

Oltre al sistema operativo dell'Apple, userete il sistema operativo Basic, che sarà discusso in questo capitolo. È scritto in Basic e sarà memorizzato nella parte di memoria riservata ai programmi in Basic (vedere la mappa della memoria che segue). Esso controllerà l'input, le correzioni, e le operazioni dei programmi in linguaggio macchina che userete.

Non si possono nemmeno memorizzare dati agli indirizzi che contengono le ROM del monitor, le ROM dell'Applesoft o porte di input/output non utilizzate. Le ROM (*Read Only Memory*) sono memorie a sola lettura, cioè memorie dalle quali le informazioni possono essere lette, ma nelle quali *non* è possibile mettere delle informazioni. Esse contengono già delle informazioni che sono protette (non possono essere cambiate da un programma o con un input in modo immediato).

Ecco una mappa della memoria per il firmware Applesoft.

MAPPA DELLA MEMORIA APPLESOFT IN FIRMWARE (ROM)

<i>Indirizzo</i>	<i>Funzione</i>
00000-00511	Spazio di lavoro del programma, non a disposizione dell'utente
00512-00767	Buffer dei caratteri della tastiera
*00768-01023	A disposizione dell'utente per brevi programmi in linguaggio macchina
01024-02047	Area riservata al video
02048-XXXXX	Area a disposizione dell'utente per programmi in Basic e variabili. XXXXX è determinato dalla massima quantità di memoria RAM installata sulla tua macchina con 16 K, XXXXX = 16383 con 32 K, XXXXX = 32767 con 64 K, XXXXX = 49151
08192-16383	Usata dai grafici ad alta risoluzione (pagina 1)
16384-24575	Usata dai grafici ad alta risoluzione (pagina 2)

49152-53247	Indirizzi di I/O dell'hardware	
53248-63487	Interprete Applesoft (se l'interruttore è posizionato per il Basic Applesoft)	
53248-57343 } 57344-63487 }	{ Area per le ROM Integer Basic e mini-assembler }	(Se l'interruttore è posizionato per l'Integer Basic)
64488-65535	Monitor di sistema dell'Apple	

*Questa è l'area che si utilizzerà per i programmi in linguaggio macchina.

NOTA Se avete l'Applesoft su disco, potete trovare la mappa della memoria sul *Disk Operating System Instructional and Reference Manual*.

L'area che sarà usata per i programmi in linguaggio macchina è stata marcata con un asterisco su ogni mappa di memoria. Questa area, dall'indirizzo 768 all'indirizzo 1023, è la stessa per entrambe le versioni Applesoft. Essa sarà sufficiente per la maggior parte, se non per tutti, i programmi in linguaggio macchina. Se vi servirà più spazio, quando i vostri programmi si allungheranno, si potranno mettere nell'area normalmente riservata ai programmi in Basic.

Useremo il programma che segue per mostrare come si usa il Basic per sistemare in memoria un programma in linguaggio macchina. I dati nelle istruzioni POKE sono gli elementi del programma in linguaggio macchina. Controlleremo anche il programma, usando l'istruzione PEEK, per verificare che sia stato memorizzato agli indirizzi giusti.

USO DI POKE E PEEK

```

100 REM *CANCELLA LO SCHERMO*
110 HOME

200 REM *MEMORIZZA IL PROGRAMMA*
210 POKE 768,169
220 POKE 769,19
230 POKE 770,141
240 POKE 771,37
250 POKE 772,3
260 POKE 773,96

```

```
300 REM*VISUALIZZA IL PROGRAMMA*
310 FOR X = 768 TO 773
320 PRINT PEEK(X)
330 NEXT X
```

Ora se eseguite questo programma in Basic, i valori posti in memoria verranno visualizzati.

```
169
19
141
37
3
96
]■
```

Sì, il programma in linguaggio macchina è stato sistemato in memoria correttamente. Usando soltanto due istruzioni del Basic, avete gli strumenti necessari per inserire programmi in linguaggio macchina (con POKE) e controllarli (con PEEK). Tuttavia, ci serve anche un modo per eseguirli dopo che sono stati inseriti.

L'istruzione CALL causa l'esecuzione di un programma in linguaggio macchina che inizia ad uno specifico indirizzo. L'indirizzo da usare per il programma che avete appena inserito è 768. Si può eseguire il programma usando l'istruzione:

```
410 CALL 768
```

L'istruzione CALL è usata per eseguire un programma in linguaggio macchina che è una subroutine di un programma in Basic. Un'istruzione GOSUB esegue una subroutine in Basic da un programma in Basic; un'istruzione CALL esegue una subroutine in linguaggio macchina da un programma in Basic.

Aggiungeremo l'istruzione CALL al nostro programma cosicché potremo eseguire la subroutine in linguaggio macchina. Metteremo anche un'istruzione PEEK per vedere il risultato ed accertarci che il programma sia stato eseguito correttamente.

L'ultima istruzione eseguita nel programma in linguaggio macchina deve essere un "ritorno dalla subroutine" (RTS). Questa è un'istruzione che ha, per una subroutine in linguaggio macchina, la stessa funzione dell'istruzione RETURN del Basic per una subroutine in Basic. Essa riporta il controllo al programma in Basic dal quale la subroutine era stata chiamata.

Per ora non preoccupatevi delle istruzioni in linguaggio macchina che saranno usate nel programma. I codici di questo linguaggio saranno introdotti gradualmente a partire dal Cap. 3. Per il momento, una spiegazione di ogni parte del programma è data a destra dei codici di macchina. Il nostro programma completo è questo:

POKE, PEEK E POI ANCORA POKE



100 REM*CANCELLA LO SCHERMO*
110 HOME



200 REM *MEMORIZZA IL PROGRAMMA*
210 POKE 768,169 Prende il valore 19
220 POKE 769,19
230 POKE 770,141 Lo sistema in memoria
240 POKE 771,37
250 POKE 772,3
260 POKE 773,96 Ritorno dalla subroutine



300 REM *VISUALIZZA IL PROGRAMMA*
310 FOR X = 768 TO 773
320 PRINT PEEK(X)
330 NEXT X



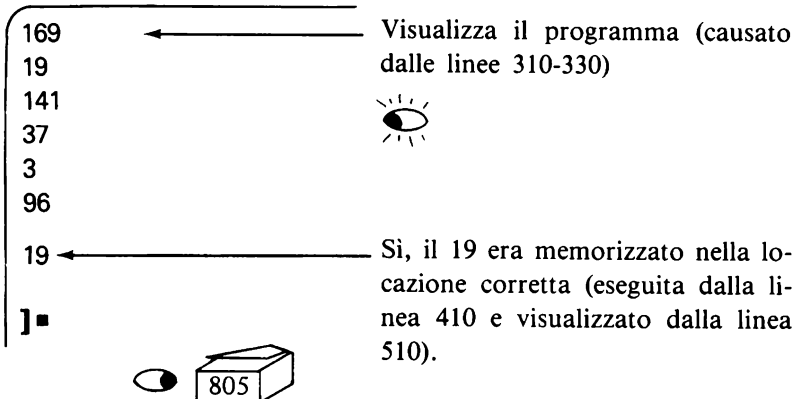
400 REM*ESEGUE IL PROGRAMMA*
410 CALL 768



500 REM *GUARDA IL RISULTATO IN MEMORIA*
510 PRINT: PRINT PEEK(805)

Abbiamo memorizzato 19 qui.

Quando il programma girerà, questo è quello che vedrete:



Quando il programma sarà eseguito, non vedrete le istruzioni in linguaggio macchina mentre vengono messe in memoria dalle istruzioni 210-260. Sullo schermo verranno visualizzati i risultati dell'esecuzione delle istruzioni del ciclo FOR...NEXT (linee 310-330):

169	Questi sono codici di linguaggio
19	macchina e valori di dati che sono
141	stati memorizzati dalle linee 210-
37	260 e stampati dall'istruzione alla
3	linea 320 nel ciclo FOR-NEXT.
96	

Dopo la visualizzazione dei dati, la linea 410 farà partire l'esecuzione del programma in linguaggio macchina. Quando il calcolatore ritornerà al programma in Basic (ritorno causato dall'ultima istruzione del programma in linguaggio macchina), la linea 510 causerà la stampa del valore 19, assicurandosi così che il dato è stato posto nella locazione di memoria che volevamo.

UN SEMPLICE SISTEMA OPERATIVO IN BASIC

Basandoci sulle istruzioni del Basic appena usate, costruiremo ora un semplice programma in Basic che possa accettare ed eseguire un programma in linguaggio macchina, e leggere i risultati forniti. Mancherà di molte caratteristiche che sono desiderabili per una programmazione più sofisticata, ma per i nostri scopi sarà sufficiente.

È un programma in linguaggio Basic, ma lo potrete usare per inserire ogni programma in linguaggio macchina che incontrerete nel resto del libro. Lo potrete anche usare per esaminare i programmi già inseriti nella ricerca di eventuali errori. Lo utilizzerete per eseguire i programmi usando l'istruzione CALL al momento appropriato. Questo programma in Basic permette anche di esaminare i risultati del programma in linguaggio macchina che esso ha creato.

Il programma in Basic sarà usato per così tante cose che abbiamo deciso di chiamarlo sistema operativo. Esso è l'operatore ed ha il controllo di tutte le operazioni che saranno effettuate dal computer una volta che sia stato inserito ed eseguito. Poiché è scritto in Basic, gli abbiamo dato il nome completo: sistema operativo in Basic (in breve sistema operativo). Ricordate che il sistema operativo è scritto in Basic, cosicché esso comprende (usa) istruzioni del Basic e numeri *decimali*. Tuttavia, come abbiamo detto prima, il computer può capire solo istruzioni in linguaggio macchina che sono codificate come numeri binari. In effetti, molti riferi-

menti standard al linguaggio macchina listano questi codici come numeri esadecimali. Questi strani numeri saranno spiegati nel Cap. 3, dove introdurremo le istruzioni in linguaggio macchina.

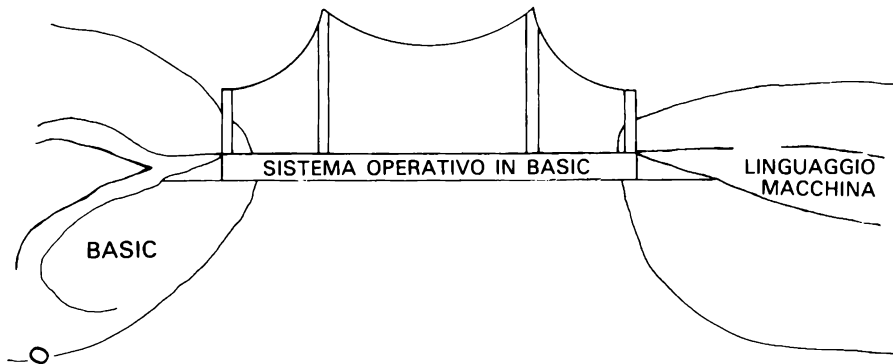
Ecco il nostro dilemma:



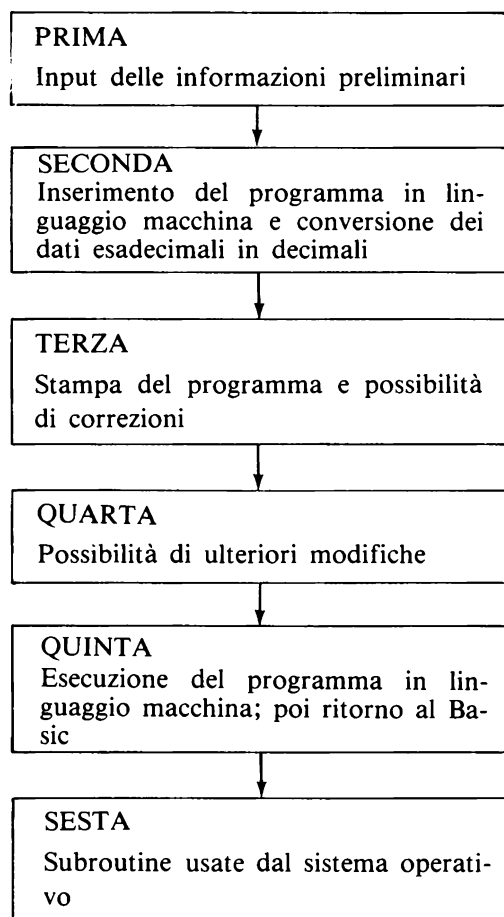
1. I riferimenti listano i codici di linguaggio macchina come numeri *esadecimali*.
2. Il computer comprende i codici solo come numeri *binari*.
3. L'interprete Basic deve ricevere numeri *decimali*, che poi converte in numeri *binari* che il computer usa.

Si potrebbe convertire ogni codice di linguaggio macchina esadecimale in un valore decimale che l'interprete Basic possa usare. Questa conversione è un compito lungo e noioso. Il computer potrebbe fare questa conversione più velocemente di un uomo, se fosse provvisto di un programma adatto allo scopo. Per questo motivo includeremo questa possibilità come parte del sistema operativo.

Seguiteci mentre progetteremo questo sistema operativo: esso servirà da ponte per portarvi dalla programmazione in Basic a quella in linguaggio macchina.



Inizieremo ora la costruzione del nostro sistema operativo, spiegandone ogni sua parte funzionale. Il breve programma in linguaggio macchina introdotto precedentemente sarà usato per mostrare come lavora ogni parte del sistema operativo. Ecco una breve descrizione del sistema operativo suddiviso in parti funzionali.



Prima



Dobbiamo avere qualche informazione fondamentale sul programma in linguaggio macchina che sarà usato. Abbiamo bisogno di sapere il suo indirizzo iniziale in memoria e quanto sarà lungo il programma (quante locazioni userà). Nella terminologia dei computer si dice che ogni locazione di memoria contiene un byte di informazioni. Un byte è costituito da 8 bit. I bit sono stati discussi brevemente in questo capitolo, e sia byte che bit saranno discussi più a fondo nel Cap. 3. Ecco come si ottengono le informazioni per ogni locazione di memoria.

PARTE 1

```

100 REM *RICHIESTA DI INFORMAZIONI SUL PROGRAMMA*
110 HOME
120 INPUT "INDIRIZZO INIZIALE DEL PROGRAMMA=?";S
130 INPUT "QUANTI BYTE?"; B
140 INPUT "BATTI RETURN PER INSERIRE IL PROGRAMMA";A$

150 A = S

```

entrambi gli input in
decimale

La linea 110 cancella lo schermo. Alla linea 120 inserite l'indirizzo iniziale del programma. (Noi, come indirizzo iniziale, useremo 768.) La linea 130 richiede il numero di byte. (Il programma che useremo è di 6 byte.) L'indirizzo iniziale è assegnato alla variabile S, mentre il numero di byte è assegnato alla variabile B. Il computer aspetta alla linea 140 che si incominci ad inserire il programma. Dopo aver premuto il tasto RETURN, alla variabile A viene assegnato lo stesso valore di S. Questo viene fatto per salvare in S l'indirizzo iniziale, mentre A viene usata come indirizzo di lavoro, che cambia per indicare al computer a quale indirizzo sistemare i successivi dati.

Esempio

```

┌ INDIRIZZO INIZIALE DEL PROGRAMMA=?768 ← S = 768
├ QUANTI BYTE?6 ← R = 6
└ BATTI RETURN PER INSERIRE IL PROGRAMMA ■
    ← Dopo aver premuto RETURN, A = 768

```

Seconda**2**

Poi il sistema operativo stamperà in ordine ogni indirizzo ed aspetterà che si inserisca un codice esadecimale a due cifre, sia come codice di un'istruzione del programma che come byte di dati. Ricordate, il sistema operativo si prenderà cura di convertire i valori esadecimali in valori decimali. Questo viene fatto in una subroutine localizzata alla linea 1000. Dopo la conversione, il risultato decimale è sistemato all'indirizzo specifico. L'indirizzo viene quindi incrementato di uno alla linea 290 e viene richiesto il prossimo input. Questo procedimento continuerà finché non sarà stato inserito l'intero programma.

Indirizzo + 1 = Prossimo indirizzo

PARTE 2

```

200 REM *SI INTRODUCE IL PROGRAMMA IN ESA SI
    CONVERTE IN DECIMALE*
210 FOR E = 1 TO B
220 PRINT A; SPC(2); ← Stampa dell'indirizzo
230 GET H$: PRINT H$; } ← Dati esadecimali
240 GET U$: PRINT U$ }
250 IF ASC(H$)<48 OR ASC(H$)>70 OR (ASC(H$)>57 AND
    ASC(H$)<65) THEN PRINT "PRIMA CIFRA NON ESA — RI
    PROVA": GOTO 220
260 IF ASC(U$)<48 OR ASC(U$)>70 OR (ASC(U$)>57 AND
    ASC(U$)<65) THEN PRINT "SECONDA CIFRA NON ESA —
    RIPROVA": GOTO 220
270 GOSUB 1000 ← Converte in decimale
280 POKE A,D ← Mette i dati in memoria
290 A = A + 1 ← Prossimo indirizzo
300 NEXT E

```

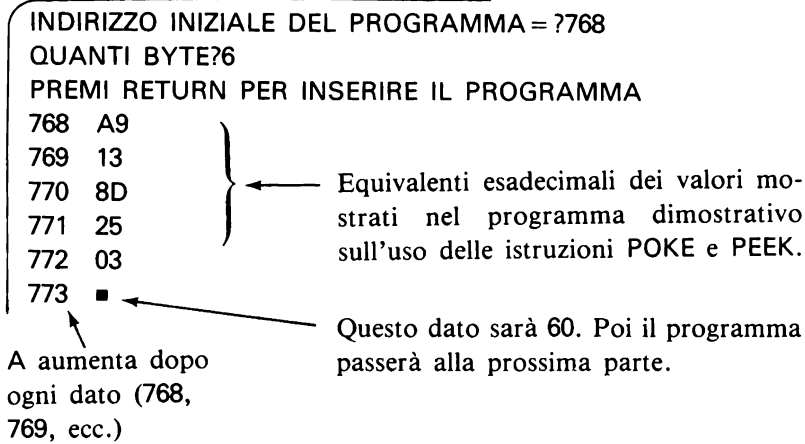
Le linee 250 e 260 assicurano che gli input siano scritti in esadecimale. Esse *non* controllano che sia stata usata un'istruzione valida del linguaggio macchina. Questo sta a voi, come programmatori.

Esempio

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
 QUANTI BYTE?6
 PREMI RETURN PER INSERIRE IL PROGRAMMA
 768 ■

← Il computer aspetta l'inserimento dei dati

Dopo che è stato inserito tutto, tranne l'ultima istruzione:



Terza



Ora dovremo stampare il programma, cosicché si possa controllare se ci sono errori. Per fare questo si scriverà una subroutine alla linea 2000. Per ora supporremo che sia stato fatto, ma torneremo su questo più avanti. Vogliamo anche che sia possibile modificare il programma nel caso vengano trovati degli errori. Il nostro programma dimostrativo è molto corto e può essere visualizzato per intero sullo schermo con la routine della parte 2. Perciò, questa parte vi può sembrare inutile. Tuttavia i vostri programmi saranno più lunghi in futuro, e qualcuno probabilmente non ci starà tutto sul video. La subroutine di stampa visualizzerà i programmi a blocchi di 20 linee, in modo che li possiate controllare.

Quando state usando questa parte, guardate l'intero programma e localizzate ogni cambiamento che volete fare. Le modifiche sono possibili *dopo* che tutto il programma è stato visualizzato dalla subroutine ed è stato eseguito un ritorno alla linea 420.

PARTE 3

```

400 REM *STAMPA DEL PROGRAMMA E CORREZIONI*
410 GOSUB 2000          ← Stampa il programma
420 PRINT "SE CI SONO MODIFICHE BATTI L'INDIRIZZO"
430 PRINT "ALTRIMENTI BATTI 99"
440 INPUT AD
450 IF AD = 99 GOTO 700 ← Esegue il programma
460 PRINT AD;
    
```

```

470 PRINT " DATO=?";      ← Riceve un dato corretto
480 GET H$: PRINT H$;
490 GET U$: PRINT U$
500 IF ASC (H$) < 48 OR ASC(H$) > 70 OR (ASC(H$) > 57 AND
    ASC(H$) < 65) THEN PRINT "PRIMA CIFRA NON ESA —
    RIPROVA": GOTO 460
510 IF ASC(U$) < 48 OR ASC(U$) > 70 OR (ASC(U$) > 57 AND
    ASC(U$) < 65) THEN PRINT "SECONDA CIFRA NON ESA —
    RIPROVA": GOTO 460
520 GOSUB 1000
530 POKE AD,D             ← Effettua una modifica

```

Se ci sono delle modifiche, dovrete scrivere l'indirizzo dove volete effettuare il cambiamento (linea 440). Quell'indirizzo verrà quindi stampato seguito dalla domanda DATO=?. Allora si inserirà il dato corretto in forma esadecimale. Questo sarà convertito in forma decimale dalla subroutine alla linea 1000 e sistemato in memoria dalla linea 530.

Esempio

Non occorrono modifiche dopo la stampa del programma:

ECCO IL TUO PROGRAMMA

```

768 A9
769 13
770 8D
771 25
772 03
773 60

```

```

PREMI UN TASTO QUALSIASI PER CONTINUARE ← Abbiamo premuto
SE CI SONO MODIFICHE BATTI L'INDIRIZZO    un tasto
ALTRIMENTI BATTI 99

```

```

?99

```

← Abbiamo scritto 99,
nessuna modifica.
Il programma verrà
allora eseguito.

Se dopo la stampa del programma servono modifiche:

ECCO IL TUO PROGRAMMA

```

768 A0      ← Qui c'è un errore  Oh, Oh!
769 13
770 8D
771 25
772 03
773 60
PREMI UN TASTO QUALSIASI PER CONTINUARE ← Abbiamo premuto un ta-
SE CI SONO MODIFICHE BATTI L'INDIRIZZO      sto dopo aver scoperto un
ALTRIMENTI BATTI 99                          errore
?768      ← Abbiamo scritto l'indirizzo
768 DATO=?A9 ← Poi il dato corretto
    
```

Per vedere cosa succede ora, dobbiamo aspettare la prossima parte.



Quarta

4

Può essere necessario più di un cambiamento. Così abbiamo previsto la possibilità di effettuare più modifiche. Questa parte viene eseguita soltanto nel caso in cui sia stata effettuata una correzione al programma.

PARTE 4

```

600 REM *ULTERIORI MODIFICHE*
610 INPUT "ALTRE MODIFICHE (SI O NO)?"; C$
620 IF LEFT$(C$,1) = "S" THEN GOTO 420 ← Se sì fa la modifica
630 GOSUB 2000      ← Poi stampa ancora il programma
640 INPUT "ALTRE MODIFICHE (SI O NO)?"; C$
650 IF LEFT$(C$,1) = "S" THEN GOTO 420
    
```

La linea 610 chiede se ci sono altre modifiche. Se la risposta è NO, il programma viene ristampato affinché lo si possa esaminare ancora una volta. Vi è un'ultima possibilità di cambiarlo alla linea 640. Se la risposta è ancora NO, il programma passa alla fase esecutiva. Se la risposta è SI (o perlomeno inizia con una S, a causa delle istruzioni LEFT\$ alle linee 620 e 650), il computer ritorna alla linea 420 per altre modifiche. Resterà in questo ciclo finché non si modificherà il programma come si desidera.

Supponiamo che durante l'inserimento del programma si siano commessi due errori.

Esempio

```

ECCO IL TUO PROGRAMMA

768 B9
769 13
770 8C          Vedete gli errori?
771 25
772 03
773 60
PREMI UN TASTO QUALSIASI PER CONTINUARE ■
  
```

Avete trovato gli errori e volete correggerli. Premete un tasto qualsiasi.

```

ECCO IL TUO PROGRAMMA
768 B9
769 13
770 8C
771 25
772 03
773 60
PREMI UN TASTO QUALSIASI PER CONTINUARE ← Avete premuto
SE CI SONO MODIFICHE BATTI L'INDIRIZZO      un tasto
ALTRIMENTI BATTI 99
prima
modifica → ?768          ← Avete scritto l'indirizzo
768 DATO = ?A9      ← Poi il dato
ALTRE MODIFICHE (SI O NO)?SI ← Avete risposto SI
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
seconda
modifica → ?770          ← Avete scritto l'indirizzo
770 DATO = ?8D      ← Poi il dato
ALTRE MODIFICHE (SI O NO)?NO ← Avete scritto NO
  
```

Lo schermo viene cancellato e viene visualizzato il programma corretto.

ECCO IL TUO PROGRAMMA

```
768 A9
769 13
770 8D
771 25
772 03
773 60
```

← Sembra corretto, così premete un tasto.

PREMI UN TASTO QUALSIASI PER CONTINUARE

ALTRE MODIFICHE (SI O NO)?NO ← A questa ultima
possibilità rispondete NO

Si va allora ad eseguire il programma nella prossima parte.

Quinta

5

Questa parte esegue il programma in linguaggio macchina. Si ferma alla linea 710 per farvi prendere ancora una volta coraggio. Funzionerà correttamente o no? Premete un tasto e tutto è finito, veloce come un battito di ciglia. Ragazzi, questa è velocità!

PARTE 5

```
700 REM *ESEGUE IL PROGRAMMA IN LINGUAGGIO MACCHINA*
710 PRINT "PREMI UN TASTO QUALSIASI PER PARTIRE": GET A$
```

```
.
.
.
800 CALL S
.
.
.
900 END
```

← Spazio lasciato per permettere all'utente di aggiungere input

← S è la variabile dell'indirizzo di inizio

← Spazio per le istruzioni per dare i risultati del programma in linguaggio macchina

Il programma in linguaggio macchina viene chiamato alla linea 800. Notate lo spazio lasciato tra le linee 710 e 800. Qui, se volete, potete inserire altre istruzioni in Basic per dare degli input speciali ai vostri programmi in linguaggio macchina, cioè, POKE indirizzo, dato. Tra le linee 800 e 900 si possono inserire delle istruzioni in Basic per leggere i risultati dei programmi, cioè, PEEK (indirizzo).

Per esempio, il nostro programma dimostrativo prende il valore esadeci-

male 13 e lo mette nella locazione di memoria 0325 (sempre in esadecimale). Se il programma lavorasse correttamente, il valore decimale 19 (13 esadecimale) dovrebbe essere stato posto nella locazione il cui indirizzo è 0325 (linee 771 e 772 del programma in linguaggio macchina). Questo valore in forma decimale è 805 ($3 \times 16^2 + 2 \times 16 + 5$). Possiamo scoprire se è stato fatto effettivamente questo, scrivendo:

PRINT PEEK(805)

dopo che il programma in linguaggio macchina è stato eseguito.

Esempio

ECCO IL TUO PROGRAMMA

```
768 A9
769 13
770 8D
771 25
772 03
773 60
```

PREMI UN TASTO QUALSIASI PER CONTINUARE

Avete premuto

un tasto

ALTRE MODIFICHE (SI O NO)?NO

← Avete battuto NO

PREMI UN TASTO QUALSIASI PER PARTIRE

← Avete premuto
un tasto

? [G] ?
[A] [C] ?
? [*] ?

]PRINT PEEK (805)

19

]■

Il valore trovato è 19. Si tratta dell'equivalente decimale di 13 esadecimale che è stato introdotto, ed è stato spostato nella corretta locazione di memoria.

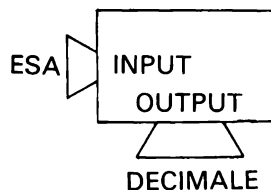
Sesta

6

Come ultima cosa vengono le subroutine del sistema operativo. La prima converte i dati esadecimali in valori decimali per l'interprete Basic.

PARTE 6A

```
1000 REM *CONVERTE DA ESA A DECIMALE*
1010 M = ASC(H$): N = ASC(U$)
1020 IF M>57 THEN M = M-55: GOTO 1040
1030 M = M-48
1040 IF N>57 THEN N = N-55: GOTO 1060
```




```

1050 N = N-48
1060 D = 16*M+N
1070 RETURN

```

La linea 1020 controlla il codice ASCII della prima cifra esadecimale. Sarà uno dei valori mostrati nella tabella che segue, nella colonna codice ASCII. Le linee 1020 e 1030 convertono il valore esadecimale nel suo equivalente decimale.

Esempi

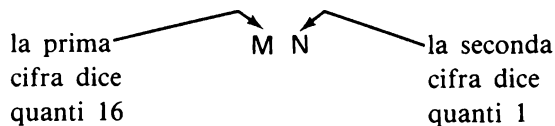
1. Il valore esadecimale 8 ha 56 come codice ASCII. Perciò, M non è > 57 e viene eseguita la linea 1030. Il nuovo M vale $56-48 = 8$ (l'equivalente decimale di 8).
2. Il valore esadecimale B ha codice ASCII 66. Quindi, $M > 57$. Il nuovo M vale $55-55 = 11$ (l'equivalente decimale di B).

Alle linee 1040 e 1050 viene seguito lo stesso procedimento per la seconda cifra. Ecco una tabella di conversione tra valori esadecimali, ASCII e decimali.

TABELLA DI CONVERSIONE

<i>Cifra esadecimale</i>	<i>Codice ASCII</i>	<i>Valore decimale</i>
0	48	0
1	49	1
2	50	2
3	51	3
4	52	4
5	53	5
6	54	6
7	55	7
8	56	8
9	57	9
A	65	10
B	66	11
C	67	12
D	68	13
E	69	14
F	70	15

Un numero esadecimale a due cifre ha i seguenti valori di posizione:



In forma esadecimale, M ed N possono essere una qualsiasi cifra da 0 a F. Le linee da 1020 a 1050 convertono queste cifre esadecimali (da 0 a F) nei loro equivalenti decimali (da 0 a 15).

M è il numero decimale di 16 (da 0 a 15)

N è il numero decimale di 1 (da 0 a 15)

Gli equivalenti decimali sono poi combinati in un numero decimale moltiplicando l'equivalente decimale di M per 16 e sommando l'equivalente decimale di N. Questa operazione viene effettuata alla linea 1060.

Esempio

Numero esadecimale originale = C7

ASCII 67

ASCII 55

dalla linea 1020, nuovo M = 67—55 = 12

linea 1050, nuovo N = 55—48 = 7

linea 1060 D = $16 \times 12 + 7 = 199$ (l'equivalente decimale di C7 esadecimale)

La subroutine di stampa visualizza fino a 20 linee alla volta del programma in linguaggio macchina. Il calcolatore aspetta che si esaminino queste linee e si preme un tasto prima di visualizzare le successive 20 linee di programma. Questa subroutine può essere richiamata dalle parti 3 o 4 del programma a seconda di dove sono state fatte le modifiche. Essa ritorna il controllo alla stessa parte dalla quale è stata richiamata.

La linea 2030 inizializza i contatori J ed I rispettivamente a 0 e 19, cosicché il ciclo FOR...NEXT compreso tra le linee 2200 e 2220 visualizzi sullo schermo 20 indirizzi e relativi dati.

PARTE 6B

```
2000 REM *SUBROUTINE PER VISUALIZZARE IL PROGRAMMA*
2010 HOME: PRINT "ECCO IL TUO PROGRAMMA"
2020 PRINT
2030 J = 0: I = 19
2040 ON INT((B-1)/20) + 1 GOTO 2090, 2080, 2070, 2060, 2050
2050 GOSUB 2200
```

```

2060 GOSUB 2200
2070 GOSUB 2200
2080 GOSUB 2200
2090 I = B - 1: GOSUB 2200
2100 RETURN

2200 REM
2210 FOR E = J TO I
2220 PRINT S + E; SPC(2);: GOSUB 3000
2230 NEXT E
2240 PRINT "PREMI UN TASTO QUALSIASI PER CONTINUARE":
      GET A$
2250 J = I + 1: I = I + 20
2260 RETURN

```

La linea 2040 usa il numero di byte del programma per calcolare quanti blocchi di 20 linee devono essere visualizzati. Essa usa l'istruzione ON...GOTO per decidere il numero di volte che la subroutine di stampa sarà usata (alla linea 2200). La linea 2090 visualizza l'ultimo blocco di linee del programma. Questo blocco può non essere di 20 linee. Perciò, il limite superiore del ciclo FOR...NEXT viene cambiato in modo che siano stampate soltanto le righe rimanenti del programma. Lo schermo viene cancellato alla linea 2200 ogni volta che deve essere visualizzato un nuovo blocco di linee. Il programma si ferma alla linea 2240 cosicché si possano esaminare attentamente le linee per vedere se ci sono errori. Quando si preme un qualsiasi tasto, viene stampato il prossimo blocco di linee.

Per ultima c'è la subroutine che converte i dati da valori decimali ottenuti con PEEK (alla linea 3010) in valori esadecimali usati come dati.

PARTE 6C

```

3000 REM *CONVERTE IN ASCII E VISUALIZZA*
3010 Y = PEEK(S + E)
3020 H = INT(Y/16)
3030 U = Y - 16*H
3040 IF H < 10 THEN PRINT H;: GOTO 3060
3050 PRINT CHR$(H + 55);
3060 IF U < 10 THEN PRINT U:GOTO 3080
3070 PRINT CHR$(U + 55)
3080 RETURN

```

La linea 3010 preleva il contenuto dell'indirizzo che verrà visualizzato. Le linee 3020 e 3030 separano il valore decimale nel numero di 16 (H) e

nel numero di 1 (U). Le linee da 3040 a 3070 convertono H e U nei loro equivalenti ASCII per la stampa.

Poiché useremo questo sistema operativo molto spesso nei prossimi capitoli, sarà meglio inserirlo nel computer e poi salvarlo su una cassetta o su un disco. Poi potrà essere letto velocemente quando necessario.

Fondamentalmente il sistema operativo permette di:

1. Inserire un programma in linguaggio macchina
2. Modificare le istruzioni e i dati
3. Eseguire il programma

Alcune istruzioni sull'uso del sistema operativo saranno date nei prossimi capitoli quando necessario.

IL SISTEMA OPERATIVO COMPLETO

Ecco il sistema operativo completo. Dovrete inserirlo nel vostro Apple. Quando l'avrete provato per essere sicuri che lavora correttamente, salvatelo su una cassetta o su un disco. Sarebbe troppo noioso doverlo riscrivere ogni volta che lo si deve usare.

```

100 REM *RICHIESTA DI INFORMAZIONI SUL PROGRAMMA*
110 HOME
120 INPUT "INDIRIZZO INIZIALE DEL PROGRAMMA=?";S
130 INPUT "QUANTI BYTE?"; B
140 INPUT "BATTI RETURN PER INSERIRE IL PROGRAMMA"; A$
150 A = S

200 REM *SI INTRODUCE IL PROGRAMMA IN ESA SI
    CONVERTE IN DECIMALE*
210 FOR E = 1 TO B
220 PRINT A; SPC(2);
230 GET H$: PRINT H$;
240 GET U$: PRINT U$
250 IF ASC(H$)<48 OR ASC(H$)>70 OR (ASC(H$)>57 AND
    ASC(H$)<65) THEN PRINT "PRIMA CIFRA NON ESA — RI
    PROVA": GOTO 220
260 IF ASC(U$)<48 OR ASC(U$)>70 OR (ASC(U$)>57 AND
    ASC(U$)<65) THEN PRINT "SECONDA CIFRA NON ESA —
    RIPROVA": GOTO 220
270 GOSUB 1000
280 POKE A,D

```

```
290 A = A + 1
300 NEXT E

400 REM *STAMPA DEL PROGRAMMA E CORREZIONI*
410 GOSUB 2000
420 PRINT "SE CI SONO MODIFICHE BATTI L'INDIRIZZO"
430 PRINT "ALTRIMENTI BATTI 99"
440 INPUT AD
450 IF AD = 99 GOTO 700
460 PRINT AD;
470 PRINT " DATO = ?";
480 GET H$: PRINT H$;
490 GET U$: PRINT U$
500 IF ASC(H$) < 48 OR ASC(H$) > 70 OR (ASC(H$) > 57 AND
    ASC(H$) < 65) THEN PRINT "PRIMA CIFRA NON ESA —
    RIPROVA": GOTO 460
510 IF ASC(U$) < 48 OR ASC(U$) > 70 OR (ASC(U$) > 57 AND
    ASC(U$) < 65) THEN PRINT "SECONDA CIFRA NON ESA —
    RIPROVA": GOTO 460
520 GOSUB 1000
530 POKE AD,D

600 REM *ULTERIORI MODIFICHE*
610 INPUT "ALTRE MODIFICHE (SI O NO)?"; C$
620 IF LEFT$(C$,1) = "S" THEN GOTO 420
630 GOSUB 2000
640 INPUT "ALTRE MODIFICHE (SI O NO)?"; C$
650 IF LEFT$(C$,1) = "S" THEN GOTO 420

700 REM *ESEGUI IL PROGRAMMA IN LINGUAGGIO MACCHINA*
710 PRINT "PREMI UN TASTO QUALSIASI PER PARTIRE": GET A$
.
.
.
800 CALL S
.
.
.
900 END

1000 REM *CONVERTE DA ESA A DECIMALE*
1010 M = ASC(H$): N = ASC(U$)
1020 IF M > 57 THEN M = M - 55: GOTO 1040
```

```
1030 M = M-48
1040 IF N>57 THEN N = N-55: GOTO 1060
1050 N = N-48
1060 D = 16*M+N
1070 RETURN

2000 REM *SUBROUTINE PER VISUALIZZARE IL PROGRAMMA*
2010 HOME: PRINT "ECCO IL TUO PROGRAMMA"
2020 PRINT
2030 J = 0: I = 19
2040 ON INT((B-1)/20)+1 GOTO 2090, 2080, 2070, 2060, 2050
2050 GOSUB 2200
2060 GOSUB 2200
2070 GOSUB 2200
2080 GOSUB 2200
2090 I = B-1: GOSUB 2200
2100 RETURN

2200 REM
2210 FOR E = J TO I
2220 PRINT S+E; SPC(2);: GOSUB 3000
2230 NEXT E
2240 PRINT "PREMI UN TASTO QUALSIASI PER CONTINUARE":
      GET A$
2250 J = I+1:I = I+20
2260 RETURN

3000 REM *CONVERTE IN ASCII E VISUALIZZA*
3010 Y = PEEK(S+E)
3020 H = INT(Y/16)
3030 U = Y-16*H
3040 IF H<10 THEN PRINT H;: GOTO 3060
3050 PRINT CHR$(H+55);
3060 IF U<10 THEN PRINT U:GOTO 3080
3070 PRINT CHR$(U+55)
3080 RETURN
```

ESERCIZI

1. Il linguaggio Basic usa numeri

(esadecimale, binario, decimale)

2. Quando si usa l'istruzione:

POKE indirizzo, dato
sia l'indirizzo che il dato devono essere numeri _____

(esadecimali, binari, decimali)

3. Supponete di eseguire le seguenti quattro istruzioni in modo immediato. Scrivete che cosa sarà stampato sulla linea che segue l'istruzione PRINT PEEK.

POKE 768, 169
POKE 769, 19
POKE 770, 141
PRINT PEEK(769)

4. Spiegate la funzione dell'istruzione CALL. _____

5. Quando state usando il sistema operativo in Basic, gli indirizzi per l'inserimento del programma in linguaggio macchina saranno stampati sul video usando numeri _____

(decimali, esadecimali, binari)

6. I dati e le istruzioni che inserite (con il sistema operativo) per il programma in linguaggio macchina devono essere forniti usando numeri _____

(decimali, esadecimali, binari)

RISPOSTE AGLI ESERCIZI

1. Decimali

2. Decimali

3. 19

4. L'istruzione CALL causa l'esecuzione di una subroutine in linguaggio macchina. L'indirizzo d'inizio della subroutine deve essere dato. (Esempio: CALL 768 richiamerà una subroutine in linguaggio macchina che inizia alla locazione di memoria il cui indirizzo è 768).

5. Decimali

6. Esadecimali

Formato dei codici d'istruzione

SOB

L'unità centrale di processo (CPU, *Central Processing Unit*) del computer Apple è stata costruita inizialmente dalla mos Technology, Inc. Attualmente, altre due compagnie (Synertek e Rockwell) la costruiscono. Questa unità si chiama microprocessore 6502. È detta unità centrale di processo perché tutte le istruzioni ed i valori numerici vengono mandati qui per essere processati.

Il microprocessore 6502 (e quindi il computer Apple), come molti altri microprocessori, comprende soltanto istruzioni che sono codificate in blocchi di otto cifre binarie, chiamati byte. Perciò, il più grande ostacolo alla programmazione in linguaggio macchina è imparare a lavorare con informazioni in forma binaria.

0	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

 ← Blocco di 8 "bit"
o 1 "byte"

1

 ← Un bit

10

 ← Due bit (non molto importanti oggi giorno)

1101

 ← Quattro bit (talvolta detti un *nybble*)

01111101

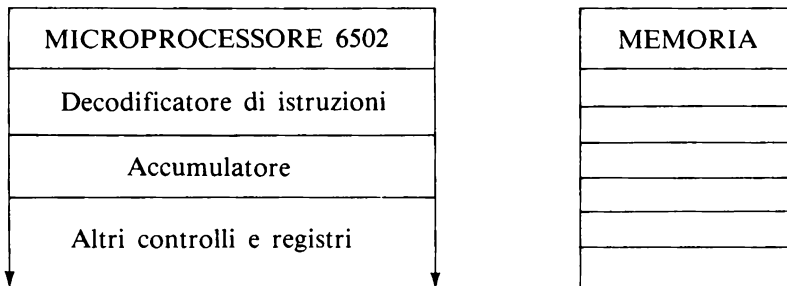
← Otto bit (comunemente chiamati un *byte*)

L'Apple usa parole lunghe otto bit; cioè, può comprendere parole la cui dimensione è un byte. Tutte le istruzioni ed i valori numerici devono essere spediti alla sua *unità centrale di processo* con questa grandezza. Una tipica istruzione, mostrata sotto, usata in un programma in linguaggio macchina avrebbe l'effetto di portare nell'*accumulatore* del computer il dato di un byte che segue l'istruzione.

CARICA L'ACCUMULATORE IN MODO IMMEDIATO	
CODICE MNEMONICO (abbreviazione)	CODICE BINARIO
LDA	10101001

Non lasciatevi spaventare dalla terminologia dei computer. L'accumulatore è simile ad una locazione di memoria che è usata in modo speciale e che discuteremo più avanti. L'abbiamo introdotto qui soltanto per mostrare il formato di un'istruzione.

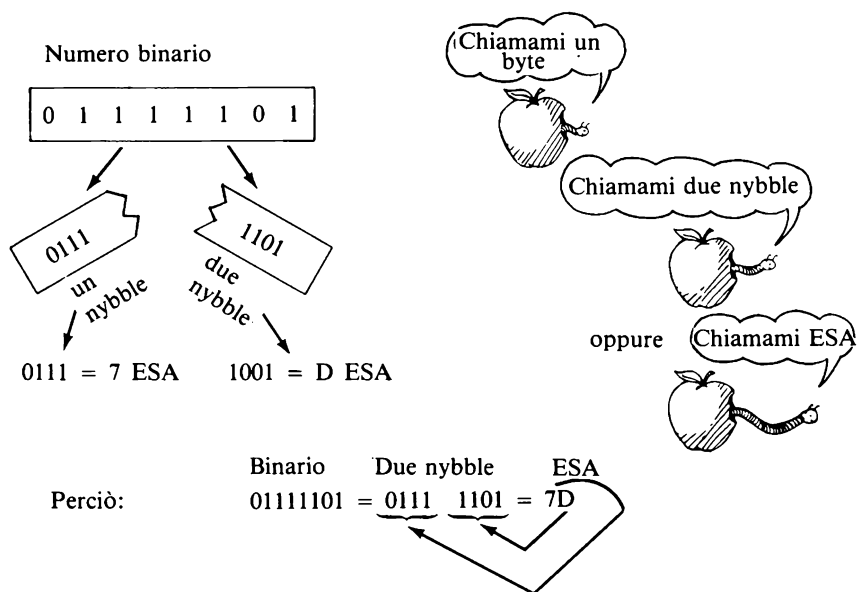
Il computer è composto di molte parti funzionali che introdurremo quando necessario per spiegare le operazioni che verranno svolte. L'unità centrale di processo dell'Apple è il microprocessore 6502. Il seguente schema mostra le "parti" con cui abbiamo attualmente a che fare.



Il decodificatore di istruzioni del 6502 "legge" l'istruzione e la decodifica. La maggior parte delle istruzioni utilizzano l'accumulatore (saranno discusse più avanti, in questo capitolo) come un centro per il movimento e la manipolazione dei dati. La memoria ed il microprocessore 6502 sono separati.

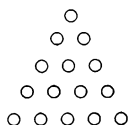
SISTEMI NUMERICI

Vi rendete conto che l'inserimento di molte istruzioni in codice binario sarebbe noioso. Poiché ci sono soltanto due simboli (0 e 1), la rappresentazione binaria dei numeri è piuttosto lunga. Molti computer, compreso l'Apple, hanno la capacità di accettare una rappresentazione più complessa. Questa facilitazione è il sistema numerico esadecimale (a cui, a volte, ci riferiremo con ESA). Quattro cifre binarie possono essere rappresentate da una cifra esadecimale. Così, la nostra istruzione di 8 bit, può essere rappresentata da un numero esadecimale di 2 cifre spezzando il byte (8 bit) in due parti (*nybble*).



Il sistema numerico esadecimale ha 16 simboli (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). La relazione tra valori decimali, binari ed esadecimali è mostrata nella seguente tabella.

<i>Decimale</i>	<i>Binario</i>	<i>ESA</i>
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



◀ Chiamami 15, 1111 oppure F, siamo sempre la stessa cosa.

Per dare significato alla tabella, diamo un'occhiata al sistema binario. Ogni posizione, nel sistema binario è una potenza di due, proprio come nel sistema decimale è una potenza di dieci. Due è la base del sistema binario e dieci è la base del sistema decimale. Se guardiamo i valori di posizione dei numeri binari da 0000 a 1111, possiamo comprendere meglio il loro significato.

<i>Posizioni binarie</i>				<i>Equivalente decimale</i>
2^3	2^2	2^1	2^0	
0	0	0	1	$0+0+0+1 = 1$
0	0	1	0	$0+0+2+0 = 2$
0	1	0	0	$0+4+0+0 = 4$
1	0	0	0	$8+0+0+0 = 8$

Usando delle combinazioni di questi valori di posizione, possiamo otte-

nere un qualsiasi valore decimale da 0 a 16 o un qualsiasi valore esadecimale da 0 a F.

Esempi

$0101 = 2^2 + 2^0 = 4 + 1 = 5$ decimale ed anche 5 ESA
 $1010 = 2^3 + 2^1 = 8 + 2 = 10$ decimale che è A ESA
 $1100 = 2^3 + 2^2 = 8 + 4 = 12$ decimale che è ESA
 $1101 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$ decimale che è D ESA

Vediamo ora più esattamente come è possibile esprimere un qualsiasi numero binario di 8 bit con due cifre esadecimali. Abbiamo visto prima che la più alta cifra esadecimale (F) corrisponde a 1111 binario. Il successivo valore binario è 10000. L'uno è nella posizione 2^4 , che vale 16. Perciò, abbiamo un 16 e niente altro. Questo può essere espresso con il valore esadecimale 10, che significa un 16 e nessun 1. C'è una relazione diretta tra i quattro bit più alti di un numero binario a otto bit e la cifra nella posizione dei sedici in un numero esadecimale.

Posizioni binarie				Valore ESA 16^1	
2^7	2^6	2^5	2^4		
0	0	0	1	1	$2^4 = 16$
0	0	1	0	2	$2^5 = 2*16 = 32$
0	1	0	0	4	$2^6 = 4*16 = 64$
1	0	0	0	8	$2^7 = 8*16 = 128$

Guardiamo ora i valori delle posizioni binarie del numero completo di 8 bit.

Posizione binaria								Equivalente decimale	Equiva- lente ESA
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
0	0	0	0	0	0	0	1	$0+0+0+0+0+0+0+1 = 1$	1
0	0	0	0	0	0	1	0	$0+0+0+0+0+0+2+0 = 2$	2
0	0	0	0	0	1	0	0	$0+0+0+0+0+4+0+0 = 4$	4
0	0	0	0	1	0	0	0	$0+0+0+0+8+0+0+0 = 8$	8
0	0	0	1	0	0	0	0	$0+0+0+16+0+0+0+0 = 16$	10
0	0	1	0	0	0	0	0	$0+0+32+0+0+0+0+0 = 32$	20
0	1	0	0	0	0	0	0	$0+64+0+0+0+0+0+0 = 64$	40
1	0	0	0	0	0	0	0	$128+0+0+0+0+0+0+0 = 128$	80

Usando combinazioni di tutti gli otto bit, si può ottenere un qualsiasi valore decimale da 0 a 255, oppure un qualsiasi valore esadecimale da 0 a FF. Se spezziamo un numero binario di 8 bit in due parti di 4 bit, ogni parte può essere rappresentata da una cifra esadecimale.

Esempi

	BINARIO	01111101	$64 + 32 + 16 + 8 + 4 + 1 = 125$ in decimale
Spezzato (separato)	BINARIO SEP.	0111 1101	
	ESA	7 D	$7 * 16 + 13 = 125$ in decimale
in due parti	BINARIO	11000011	$128 + 64 + 2 + 1 = 195$ in decimale
	BINARIO SEP.	1100 0011	
	ESA	C 3	$12 * 16 + 3 = 195$ in decimale
	BINARIO	10101010	$128 + 32 + 8 + 1 = 170$ in decimale
	BINARIO SEP.	1010 1010	
	ESA	A A	$10 * 16 + 10 = 170$ in decimale

Molto spesso i manuali relativi al linguaggio macchina riportano i codici delle istruzioni sia in forma binaria che in forma esadecimale. Il nostro sistema operativo in Basic usa il formato esadecimale per l'inserimento delle istruzioni dei programmi in linguaggio macchina. Poiché il Basic non comprende i numeri esadecimali, il sistema operativo li converte in numeri decimali per il Basic e in numeri binari per il computer. Anche se il sistema operativo in Basic è stato discusso nel Cap. 2, riteniamo utile ripetere qui la parte riguardante l'inserimento dei dati.

Le istruzioni sono inserite alle linee 200-300 del sistema operativo come numeri esadecimali.

PARTE 2

```

200 REM *SI INTRODUCE IL PROGRAMMA IN ESA SI
    CONVERTE IN DECIMALE*
210 FOR E = 1 TO B
220 PRINT A; SPC(2);
230 GET H$: PRINT H$;
240 GET U$: PRINT U$
250 IF ASC(H$) < 48 OR ASC(H$) > 70 OR (ASC(H$) > 57 AND
    ASC(H$) < 65) THEN PRINT "PRIMA CIFRA NON ESA - RI
    PROVA": GOTO 220
260 IF ASC(U$) < 48 OR ASC(U$) > 70 OR (ASC(U$) > 57 AND
    ASC(U$) < 65) THEN PRINT "SECONDA CIFRA NON ESA -

```

← Stampa dell'indirizzo

← Seguono dati in ESA

```

RIPROVA": GOTO 220
270 GOSUB 1000      ← Converti in decimale
280 POKE A,D        ← Mette il dato in memoria
290 A = A + 1       ← Prossimo indirizzo
300 NEXT E

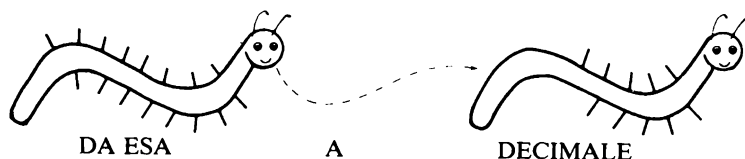
```

La conversione da esadecimale a decimale avviene in una subroutine alle linee 1000-1070. Poi l'interprete Basic cambia i numeri decimali negli equivalenti binari per il computer.

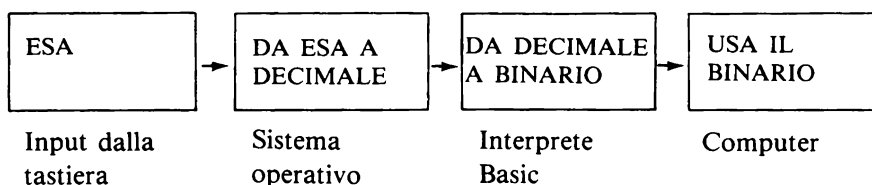
```

1000 REM *CONVERTE DA ESA A DECIMALE*
Codice ASCII → 1010 M = ASC(H$): N = ASC(U$)
in decimale   → 1020 IF M > 57 THEN M = M - 55: GOTO 1040
(M,N)         → 1030 M = M - 48
Conversione → { 1040 IF N > 57 THEN N = N - 55: GOTO 1060
                1050 N = N - 48
                1060 D = 16*M + N
                1070 RETURN

```



SCHEMA A BLOCCHI DELL'INPUT



Il nostro è alle pagine 54, 55 e 56

Il computer manipola i dati in blocchi di 8 bit, chiamati *byte*. Perciò, un dato di un byte è limitato dal valore binario 11111111 (FF esadecimale o 255 decimale). Tuttavia, combinando due byte, si possono trattare valori più grandi. Il computer usa questo metodo per accedere alle locazioni della sua memoria.

Quando viene usato un numero di due byte, ci si riferisce ad un byte come il byte meno significativo (BMS). L'altro è detto byte più significativo (BPS).

Esempio**BPS (byte più significativo)**

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	1	0	0	1

BMS (byte meno significativo)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	0	1	1	1

Non confondere i byte più e meno significativi con i bit più e meno significativi. Ogni byte ha un bps (bit più significativo) e un bms (bit meno significativo).

BYTE PIÙ SIGNIFICATIVO

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	1	0	0	1

↑
bit più
significativo

↑
bit meno
significativo

BYTE MENO SIGNIFICATIVO

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	0	1	1	1

↑
bit più
significativo

↑
bit meno
significativo

Per usare un numero di due byte, considerate il byte più significativo come un'estensione del byte meno significativo. Ai valori di posizione del byte meno significativo sono state assegnate le potenze di due da 0 a 7.

BMS

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	0	1	1	0

$$= 63 + 4 + 2 = 70 \text{ (decimale)}$$

Ai valori di posizione del byte più significativo sono assegnate le successive potenze di due (da 8 a 15).

BPS

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
1	0	0	0	1	0	0	1

$$= 32768 + 2048 + 256 = 35072 \text{ (decimale)}$$

Il valore decimale risultante dall'unione dei byte (considerati come un unico numero) è:

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

In decimale: $32768 + 2048 + 256 + 64 + 4 + 2 = 35142$

Diviso in parti di 4 bit:

<u>1000</u>	<u>1001</u>	<u>0100</u>	<u>0111</u>	← Questo valore binario è equivalente a questo
8	9	4	7	← valore ESA

Formato
ESA:

BPS	BMS
89	46

16^3	16^2	16^1	16^0
8	9	4	6

→ $8 \cdot 4096 = 32768$
+ $9 \cdot 256 = 2304$
+ $4 \cdot 16 = 64$
+ $6 \cdot 1 = 6$

35142
(decimale)

ACCUMULATORE

L'accumulatore è un registro (un luogo per la memorizzazione, simile ad una locazione di memoria) in cui vengono posti dei dati. Viene usato come un'area di memorizzazione temporanea quando si muovono dei dati da una locazione di memoria ad un'altra. Nell'accumulatore avvengono anche le operazioni aritmetiche e logiche sui dati. Quindi è usato frequentemente, e molte delle istruzioni del 6502 lo riguardano. Ricordate, il computer Apple usa l'unità centrale di processo 6502. Le istruzioni sono fissate nel 6502; cioè, ogni istruzione ha un unico e non modificabile codice di linguaggio macchina.

Anche l'accumulatore contiene un byte di dati.

1	0	0	1	1	1	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

La necessità di valori a due byte risulta evidente quando si usa un'istru-

zione per acquisire un dato da una locazione di memoria. Se volete caricare un dato nell'accumulatore dalla memoria, potreste usare l'istruzione che segue.

Precedentemente è stata vista un'istruzione per mettere un numero nell'accumulatore. Essa mette nell'accumulatore il numero che segue immediatamente l'istruzione.

Esempio

Valore binario	Valore ESA	
10101001	A9	Carica l'accumulatore
00001101	13	col valore ESA 13

I due byte (ognuno dei quali occupa una locazione di memoria diversa) indicano:

Primo: l'istruzione carica l'accumulatore (A9)

Secondo: con il valore esadecimale 13

La prossima istruzione che introdurremo mostra un secondo modo di mettere un numero nell'accumulatore. Essa prende il numero da caricare da una specifica locazione di memoria. Anche se il codice mnemonico per questa istruzione è LDA (lo stesso di quello dell'istruzione menzionata all'inizio del capitolo), il valore esadecimale che la rappresenta è differente.

Quando state caricando l'accumulatore da una specifica locazione di memoria, l'istruzione LDA ha per codice esadecimale AD. Anche se questo per voi non significa niente, essa è un'istruzione dell'Apple. La locazione di memoria è specificata dopo l'istruzione. Notate che i *due* byte necessari per specificare l'indirizzo della locazione sono dati in ordine *inverso*. Questo può sembrare ridicolo, ma per il computer è completamente logico. Il byte meno significativo viene posto nell'indirizzo di memoria più basso, e il byte più significativo è memorizzato nell'indirizzo più alto.

Indirizzo di memoria	Valore ESA	Codice mnemonico	Operando	Commenti
768	AD	LDA	0325	Carica l'accumulatore dalla locazione di memoria 0325 (ESA)
769	25			
770	03			

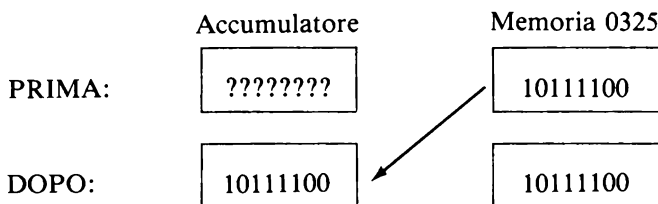
Istruzione

Ordine inverso per l'indirizzo

Questo è un esempio di istruzione a tre byte.

1. L'istruzione è data nel primo byte.
2. Il BMS della memoria compare nel successivo byte.
3. Il BPS della memoria è nell'ultimo byte.

Quando il computer eseguirà questa istruzione, il contenuto della locazione 0325 verrà copiato nell'accumulatore. Si avrà questo cambiamento:



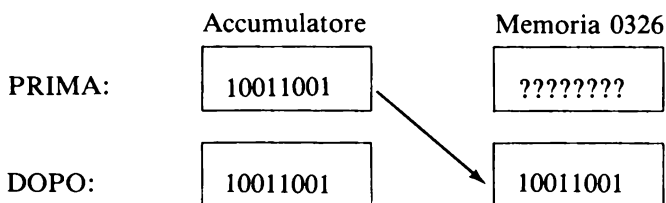
Il valore nella locazione di memoria rimane immutato ed è copiato nell'accumulatore. Sul valore contenuto nell'accumulatore si possono eseguire delle operazioni. Se lo si desidera, il risultato può poi essere trasferito in un'altra locazione. La cosa importante da ricordare è che la maggior parte delle operazioni del calcolatore avvengono nell'accumulatore. Perciò, molte delle istruzioni del 6502 riguardano questo utile registro.

Il dato nell'accumulatore può essere copiato in qualche locazione di memoria con un'istruzione di memorizzazione, come ad esempio:

	<i>Valore ESA</i>	<i>Codice mnemonico</i>	<i>Operando</i>	<i>Commenti</i>
Istruzione →	8D	STA	0326	Memorizza il valore contenuto nell'accumulatore nella locazione 0326 ESA
Locazione di memoria {	26			
	03			

Questa è un'altra istruzione a tre byte. In generale, la maggior parte delle istruzioni che si riferiscono ad una locazione di memoria richiedono tre byte. Troveremo delle eccezioni più avanti.

Quando il computer esegue questa istruzione, il dato nell'accumulatore viene copiato in una specifica locazione di memoria. Si ha questo cambiamento:



Il contenuto dell'accumulatore rimane invariato, ma il suo contenuto viene copiato (scritto) anche nella locazione 0326. Vedete che, con istruzioni come LDA e STA, l'accumulatore sta diventando un luogo molto attivo. Sarà anche importante tener conto delle locazioni di memoria usate per le varie operazioni. Ogniquale volta un dato viene memorizzato in una locazione, il dato contenuto prima viene perso.

Esempio

Supponiamo che la situazione sia questa:

Memoria	Contenuto
0325	19
0326	24
0327	00

Viene poi eseguito questo programma in linguaggio macchina:

Queste istruzioni in linguaggio macchina lavorano in modo simile alle due istruzioni del Basic consecutive:

200 LET A = 19
210 LET A = 24

768 AD Carica l'accumulatore dalla locazione di memoria 0325
769 25
770 03

19 ora nell'accumulatore

771 8D Memorizza l'accumulatore nella locazione di memoria 0327
772 27
773 03

Locazioni di memoria ora

0325 19

0326 24

0327 19 ← Questo è stato cambiato

774 AD Carica l'accumulatore dalla locazione di memoria 0326.
775 26
776 03

24 ora nell'accumulatore

777 8D Memorizza l'accumulatore nella locazione di memoria 0327
778 27
779 03

Locazioni di memoria ora

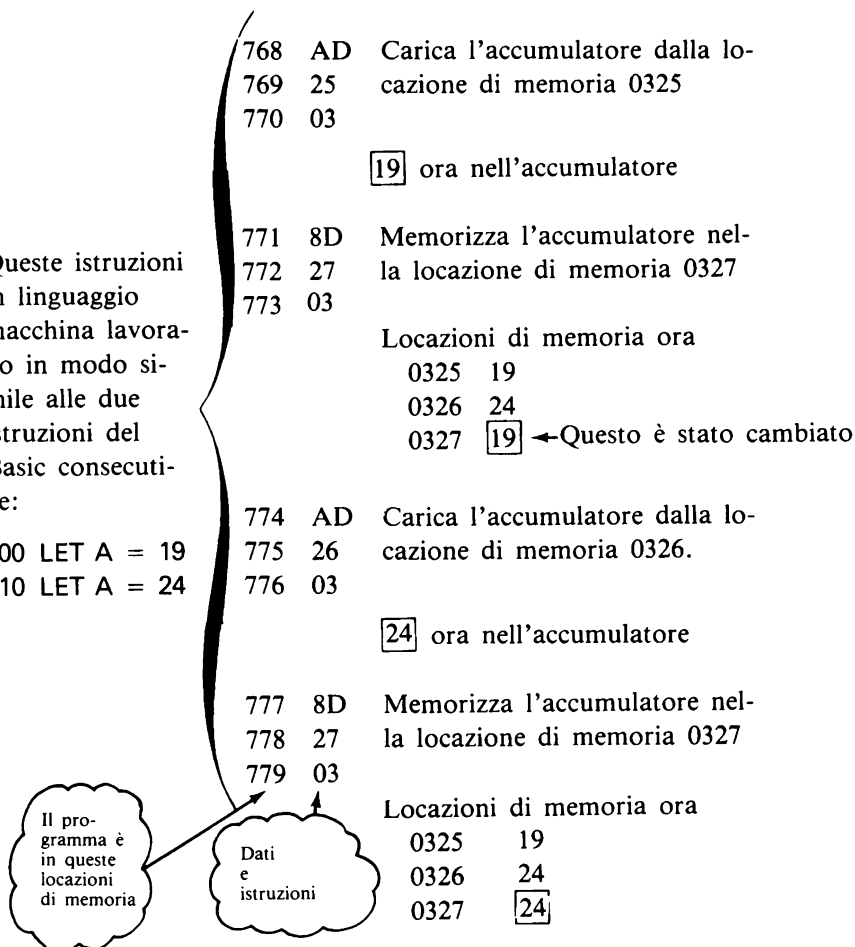
0325 19

0326 24

0327 24

Il programma è in queste locazioni di memoria

Dati e istruzioni



ISTRUZIONI DI MEMORIA

I programmi devono essere messi in memoria prima di poter essere eseguiti. Anche i vostri programmi in Basic occupano spazio in memoria. L'interprete del Basic si occupa per voi dell'assegnazione della memoria al programma in Basic, e voi non conoscete l'esatta posizione delle istruzioni. Tuttavia, dovete assegnare le locazioni di memoria ai vostri programmi in linguaggio macchina. Ad ogni byte di istruzione e ad ogni byte di dati va assegnata una locazione specifica. Ogni byte occupa una locazione di memoria.

Esempio

<i>Locazione di memoria</i>	<i>Byte di dati o di istruzione</i>	<i>Commenti</i>
0300	AD	Carica l'accumulatore dalla locazione di memoria 0325
0301	25	
0302	03	
.	.	.
.	.	.
.	.	.
0325	2B	Dati da caricare

Alcune istruzioni hanno più forme. Questa istruzione di caricamento differisce da quella usata in "Uso della memoria" nel Cap. 2, in quanto un certo valore viene caricato da una locazione di memoria piuttosto che dal byte che segue l'istruzione.

Per i nostri programmi in linguaggio macchina useremo le locazioni da 0300 a 03FF (esadecimale). Questa area della memoria non è usata dal sistema operativo dell'Apple, né dall'interprete Basic. Perciò è libera per i nostri programmi.

Useremo il nostro sistema operativo per mettere in questa area di memoria istruzioni in linguaggio macchina e dati.

*Notate ancora che la locazione di memoria 0325 viene inserita nel programma con il BMS prima ed il BPS poi. Questo può sembrare strano, ma per il calcolatore è del tutto normale. Il byte più significativo (BPS) è ora memorizzato in una locazione più alta del byte meno significativo (BMS).

codice esadecimale dell'istruzione da eseguire o del dato da usare. In questo caso è l'istruzione "carica l'accumulatore" con dato immediato.

Una lista delle istruzioni di linguaggio macchina usate in questo libro è data nell'Appendice A. Una lista completa delle istruzioni del 6502 è data nell'Appendice D.

L'istruzione che stiamo considerando consiste di due byte:

A9	(l'istruzione)
XX	(il valore ESA da caricare)

Nella nostra dimostrazione, caricheremo il valore 19 (13 in esadecimale). Perciò i due byte dell'istruzione saranno:

A9
13

Nei differenti sistemi numerici che abbiamo discusso:

Binario	Binario separato	ESA
10101001	1010 1001	A9
00010011	0001 0011	13

↑ Questi valori sono inseriti usando il sistema operativo in Basic.

- 2 Poi memorizzeremo il dato nella locazione di memoria il cui indirizzo è 0325 (esadecimale).

L'istruzione del 6502 necessaria per fare questo è:

MEMORIZZAZIONE DELL'ACCUMULATORE ASSOLUTA

CODICE OPERATIVO	MNEMONICO	OPERANDO
↓	↓	↓
8D	STA	memoria

Questa istruzione richiede tre byte:

8D	(L'istruzione)
25	(Il byte della memoria meno significativo)
03	(Il byte della memoria più significativo)

8D	}	L'istruzione è posta in tre locazioni di memoria successive
25		
03		

3 Ritorneremo quindi dal programma in linguaggio macchina al sistema operativo.

L'istruzione del 6502 usata è:

RITORNO DALLA SUBROUTINE

CODICE OPERATIVO	MNEMONICO	OPERANDO
↓	↓	↓
60	RTS	nessuno


Questa è un'istruzione di un byte.

60 (L'istruzione)

Ha la stessa funzione dell'istruzione del Basic:

RETURN

Abbiamo notato sulla mappa della memoria dell'Apple (Cap. 2) che le locazioni di memoria 768-1023 (decimale) sono libere per programmi in linguaggio macchina. Questa parte della mappa è qui riportata come riferimento.



<i>Indirizzo decimale</i>	<i>Equivalente ESA</i>
768	0300
769	0301
770	0302
771	0303
772	0304
	
1022	03FE
1023	03FF

Il nostro primo programma inizierà alla locazione di memoria 0300 esadecimale. Dobbiamo dare al nostro sistema operativo questa locazione come valore decimale. Gli indirizzi vengono inseriti nel computer usando numeri decimali che il Basic comprende. Ma le istruzioni e i dati devono essere forniti come numeri esadecimali, che il computer userà nella loro forma binaria.

16^3	16^2	16^1	16^0
0	3	0	0

$$= 3 \times 16^2 = 3 \times 256 = 768 \text{ decimale}$$

Questo è l'indirizzo dove sarà inserita la prima istruzione del programma. Dobbiamo anche dare al sistema operativo i codici operativi di ogni indirizzo.

<i>Indirizzo</i>	<i>Codice op.</i>	<i>Commenti</i>
768	A9	LDA con
769	13	dato 
770	8D	STA 
771	25	Memoria (BMS)
772	03	Memoria (BPS)
773	60	RTS

Notate: Il programma ha 6 byte (questo deve essere detto al sistema operativo).

Ecco una descrizione passo per passo di come usare il sistema operativo per inserire ed eseguire questo programma.

- 1 Inserite il sistema operativo (vedere Cap. 2). Questo può essere fatto da tastiera, oppure da cassetta o disco se avete precedentemente salvato il programma.
- 2 Scrivete: RUN (e premete RETURN)

INDIRIZZO INIZIALE DEL PROGRAMMA = ? ■

- 3 Scrivete: 768 (e premete RETURN)

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
QUANTI BYTE? ■

- 4 Scrivete: 6 (e premete RETURN)


```

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA ■

```

5 Premete il tasto RETURN

```

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA
768 ■

```

Il computer indica
il primo indirizzo

Ora inserite il programma. Non occorre che premiate RETURN dopo ogni dato. Il computer stamperà automaticamente l'indirizzo successivo. Se fate un errore, passate pure al prossimo dato. Potrete correggere qualsiasi errore quando il programma sarà stato inserito completamente.

6 Scrivete: A

```

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA
768 A ■

```

1^a cifra del
1° indirizzo

7 Scrivete: 9

```

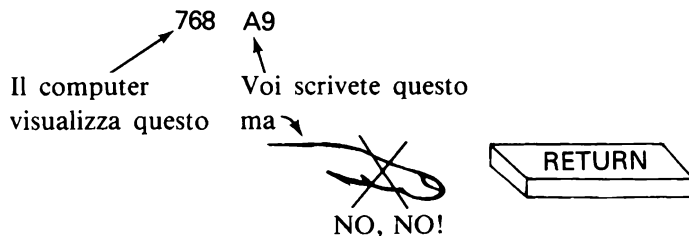
INDIRIZZO INIZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA
768 A9 ■
769 ■

```

1° dato completo

Il computer stampa il prossimo
indirizzo

Poiché non dovete premere RETURN dopo ogni tasto, d'ora in poi includeremo entrambi i tasti per ogni indirizzo in ogni passo.



L'istruzione GET non richiede che si prema RETURN. Se lo fate, l'istruzione GET lo interpreterà come uno dei vostri caratteri ed il sistema operativo lo visualizzerà:

```
768
PRIMA CIFRA NON ESA — RIPROVA
768 ■
```

Ritornando al programma,

8 Scrivete: 13

```
INDIRIZZO INZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA
768 A9
769 13
770 ■
```

9 Scrivete: 8D

```
INDIRIZZO INZIALE DEL PROGRAMMA = ?768
QUANTI BYTE?6
BATTI RETURN PER INSERIRE IL PROGRAMMA
768 A9
769 13
770 8D
771 ■
```

10 Scrivete: 25

QUANTI BYTE?6

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768

BATTI RETURN PER INSERIRE IL PROGRAMMA

768 A9

769 13

770 8D

771 25

772 ■

11 Scrivete: 03

INDIRIZZO INIZIALE DEL PROGRAMMA = ?768

QUANTI BYTE?6

BATTI RETURN PER INSERIRE IL PROGRAMMA

768 A9

769 13

770 8D

771 25

772 03

773 ■

12 Scrivete: 60

ECCO IL TUO PROGRAMMA

768 A9

769 13

770 8D

771 25

772 03

773 60

SE CI SONO MODIFICHE BATTI L'INDIRIZZO

ALTRIMENTI BATTI 99

?■

Prima di proseguire, confrontiamo questo programma in linguaggio macchina con il suo equivalente in Basic. Queste sei istruzioni in linguaggio macchina hanno una funzione molto simile alle due istruzioni del Basic:

BASIC		LINGUAGGIO MACCHINA
100 LET A = 19	↙	768 A9 LDA 19
110 RETURN	↘	769 13
	↘	770 8D STA 0325
	↘	771 25
	↘	772 03
	↘	773 60 RTS

Il programma in linguaggio macchina carica l'accumulatore con il numero esadecimale 13 (19 decimale). Poi lo memorizza in una locazione di memoria. L'istruzione del Basic LET A = 19 fa la stessa cosa. Essa mette il valore chiamato con la lettera A in memoria. L'istruzione dell'inguaaggio macchina RTS (*ReTurn from Subroutine*, ritorno dalla subroutine) esegue la stessa funzione dell'istruzione RETURN del Basic.

Studiate il programma per essere sicuri che tutti i dati siano corretti. Se avete fatto un errore, scrivete l'indirizzo corrispondente. Il calcolatore allora visualizzerà:

```

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99          Supponiamo:
?771 DATO = ? ← errore all'indirizzo
                                     771

```

Scrivete poi il dato corretto (due cifre).

```

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?771 DATO = ?25 ← Dato corretto inserito
ALTRE MODIFICHE (SI O NO)?

```

Poi battete SI o NO. Se avete scritto SI, il computer chiederà ancora l'indirizzo e il dato. Il procedimento sarà ripetuto finché non darete una risposta negativa (nessun'altra modifica). Il calcolatore allora stamperà il programma modificato e chiederà se ci sono altre modifiche. Se quest'ultima risposta è negativa, il computer visualizzerà:

```

PREMI UN TASTO QUALSIASI PER PARTIRE ■

```

Premete un tasto qualsiasi ed il programma verrà eseguito. Vedrete immediatamente il prompt ed il cursore lampeggiante. Però, questo linguaggio macchina è veloce!

PREMI UN TASTO QUALSIASI PER PARTIRE
]■

Ma il programma in linguaggio macchina è stato realmente eseguito? Come possiamo verificarlo? Il Basic ha un'istruzione chiamata PEEK che permette di vedere che cosa c'è in una specifica locazione di memoria. Se si usa:

PRINT PEEK (805) 325 ESA (la locazione dove il programma ha memorizzato il valore 19)

verrà stampato il valore memorizzato nella locazione di memoria 805 (decimale).

Poiché si voleva che il programma mettesse il numero esadecimale 13 (19 decimale) nella locazione di memoria 0325 esadecimale, possiamo usare l'istruzione PEEK per vedere che cosa c'è in questa locazione.

$$0325 \text{ ESA} = (3 \times 256) + (2 \times 16) + (5 \times 1) = 768 + 32 + 5 = 805 \text{ decimale}$$

Scrivete: PRINT PEEK(805) e premete RETURN

Presto! Eccolo!

```
]PRINT PEEK(805)
19  ← Funziona davvero!
]■
```

Potete aggiungere una linea al sistema operativo tra le linee 800 e 900 per stampare il valore in memoria usando l'istruzione PEEK.

810 PRINT PEEK(805)

Questa parte del sistema operativo (linee 800-900) può essere modificata per adattarsi allo specifico programma da eseguire. Aggiungete la linea 810 al sistema operativo per il nostro prossimo programma.

Il primo programma caricava un dato nell'accumulatore e poi lo metteva in memoria. Praticamente, tutti i programmi usano questo tipo di operazione per spostare i dati da un posto all'altro all'interno del computer. L'accumulatore viene usato anche per eseguire delle operazioni sui numeri. Una di queste operazioni sposta i bit dell'accumulatore di un posto verso sinistra. L'abbreviazione di questa operazione è ASL (*Arithmetic Shift Left*, spostamento aritmetico a sinistra). Il suo codice operativo è 0A.

Esempio

Contenuto dell'accumulatore prima dell'esecuzione dell'istruzione di spostamento

00010011

$16 + 2 + 1 = 19$ (decimale)

Questo bit è perso

Contenuto dell'accumulatore dopo l'esecuzione dell'istruzione di spostamento

00100110

$32 + 4 + 2 = 38$ (decimale)

Nell'ultimo bit viene posto automaticamente zero

Potete vedere che un'istruzione ASL è un modo di moltiplicare un numero per due, poiché il valore di ogni bit è raddoppiato.

Ora potete caricare un numero nell'accumulatore e spostare ogni bit di un posto a sinistra (moltiplicarlo per due) e poi memorizzare il risultato in una locazione di memoria. L'istruzione PEEK, che avete aggiunto alla linea 810 nel programma del sistema operativo, stamperà il risultato messo in memoria.

Usate il sistema operativo per inserire questo programma. Quando tutti i 7 byte saranno stati inseriti, lo schermo mostrerà:

ECCO IL TUO PROGRAMMA

```

768 A9      ← Carica l'accumulatore con il valore
769 13      ESA 13 (19 decimale)
770 0A      ← Sposta i bit a sinistra di 1 posto
771 8D      ← Mette il risultato in memoria (0325)
772 25
773 03
774 60      ← Ritorno al sistema operativo
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
? ■

```

Quando inserite 99, il programma mostrerà immediatamente il valore che è stato memorizzato nella locazione 0325 esadecimale (805 decimale). Ricordate che il valore sarà visualizzato come un numero decimale. Questo numero sarà 38. Ecco come apparirà lo schermo quando il programma sarà stato eseguito:

ECCO IL TUO PROGRAMMA

```

768 A9
769 13 ← (13 ESA = 19 decimale)
770 0A
771 8D
772 25
773 03
774 60
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?99
PREMI UN TASTO QUALSIASI PER PARTIRE
38 ← La risposta è data in forma decimale
                                dall'istruzione del Basic:
]■                                PRINT PEEK(805)

```

Poiché il valore nell'accumulatore viene raddoppiato ogni volta che viene eseguita l'istruzione ASL, potete immaginare che cosa succederebbe se la eseguiamo due volte. Il valore sarebbe raddoppiato e poi ancora raddoppiato (cioè moltiplicato per quattro). Verificalo aggiungendo una seconda istruzione ASL al programma sopra. Dopo aver inserito il programma modificato ed averlo eseguito, il video apparirà così (il programma ora è lungo 8 byte).

ECCO IL TUO PROGRAMMA

```

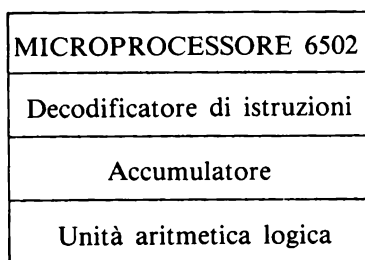
768 A9
769 13
770 0A } ← Due istruzioni ASL
771 0A }
772 8D
773 25
774 03
775 60
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?99
PREMI UN TASTO QUALSIASI PER PARTIRE
76 ← (19 per 4 = 76)
]■

```

Ecco che cosa è successo:

00010011 ← Valore iniziale nell'accumulatore
 ASL eseguita:
 00100110 ← Spostato a sinistra una volta
 ASL eseguita ancora:
 01001100 ← Spostato a sinistra due volte che è 4 per 19
 Risultato: $64 + 8 + 4 = 76$

Discuteremo dettagliatamente le capacità aritmetiche più avanti. Ricordate che l'accumulatore può contenere solo otto bit di dati. Anche le locazioni di memoria contengono otto bit. Se spostiamo a sinistra troppe volte, il numero verrà fatto uscire dall'accumulatore. Dobbiamo trovare nuove tecniche per poter trattare anche numeri grandi. Le funzioni aritmetiche e logiche sono eseguite dall'unità logica aritmetica del microprocessore 6502.



Avete scoperto che è possibile caricare dei numeri nel computer, cambiare i valori inseriti, spostare i valori all'interno del computer e stampare i risultati. Nel prossimo capitolo scoprirete qualcosa di più affascinante. Vedrete come visualizzare grafici sullo schermo.

SOMMARIO

Siete partiti molto bene. In questo capitolo, avete imparato:

1. Che il computer comprende solo istruzioni e dati binari.
2. Come convertire valori decimali, binari ed esadecimali tra di loro.
3. Che le locazioni di memoria contengono dati di un byte, cioè otto bit.
4. Che l'accumulatore (uno speciale registro simile ad una locazione di memoria) è usato per effettuare la maggior parte delle operazioni del computer.

5. Ad usare istruzioni del linguaggio macchina:

- a. LDA (*LoaD Accumulator*, carica l'accumulatore) usato per caricare l'accumulatore con un valore immediato. Il suo codice op. (A9) è seguito da un byte di dati che viene caricato nell'accumulatore.

Esempio: A9 ← Codice op.
 13 ← Dato da usare

- b. STA (*STore Accumulator*, memorizza l'accumulatore) usata per memorizzare il valore dell'accumulatore in una locazione di memoria assoluta. Il suo codice op. (8D) è seguito da un indirizzo di due byte.

Esempio 8D ← Codice op.
 25 ← byte meno significativo dell'indirizzo
 03 ← byte più significativo dell'indirizzo

- c. RTS (*ReTurn from Subroutine*, ritorno dalla subroutine) usata per causare un ritorno da un programma in linguaggio macchina (o da una subroutine).

Esempio 60 ← Codice op.

- d. ASL (*Arithmetic Shift Left*, spostamento aritmetico a sinistra) usata per spostare ogni bit dell'accumulatore di un posto a sinistra. Raddoppia il valore nell'accumulatore.

Esempio 0A ← Codice op.

6. A mettere tutte queste istruzioni in un programma in linguaggio macchina, inserito e controllato dal sistema operativo in Basic.

Anche se avete visto soltanto poche istruzioni del linguaggio macchina, siete stati in grado di capire ed usare un programma in questo linguaggio.

Buona parte delle istruzioni che userete hanno parecchie forme (o modi). Essi sono listati nell'Appendice A.

Quelli usati più di frequente sono: immediato, assoluto, implicito e pagina zero.

Voi avete usato LDA nel modo immediato. In questa forma, il dato da usare segue *immediatamente* il codice op.

Avete anche utilizzato l'istruzione STA nel modo assoluto. In questo modo, l'indirizzo completo (o *assoluto*) segue il codice op. dell'istruzione. Questo indirizzo viene dato con il byte meno significativo prima, se-

guito dal byte più significativo. Il modo implicito è stato usato per l'istruzione RTS. La sua funzione è *implicita* dell'istruzione. Perciò, non servono né l'indirizzo né il dato.

Queste forme vi diventeranno più familiari utilizzandole nei prossimi capitoli.

ESERCIZI

1. Il Basic usa numeri _____ , ma il computer
(decimali, binari)
comprende solo numeri _____.
(decimali, binari)

2. Quante cifre binarie rappresenta una cifra esadecimale? _____

3. Date gli equivalenti esadecimali e decimali di questi numeri binari.

Binario	ESA	Decimale
1001	_____	_____
1101	_____	_____
01010111	_____	_____

4. I dati vengono copiati nell'accumulatore dalla memoria con un'istruzione di _____
(caricamento, memorizzazione)

5. I dati vengono copiati nella memoria dall'accumulatore con un'istruzione di _____
(caricamento, memorizzazione)

6. Spiegate che cosa causerebbe l'esecuzione delle seguenti istruzioni.

Indirizzo	Codice op.	Commenti
771	8D	STA
772	40	memoria
773	03	

7. Il sistema operativo in Basic visualizza valori _____
(decimali, esadecimali)
per gli indirizzi del programma e valori _____
(decimali, esadecimali)
per i codici op. e i dati.

8. Scrivete il risultato che sarebbe posto nella locazione 0333 da questo programma.

```
768 A9 LDA 2C
769 2C
770 0ASL
```

771 BD STA 0333

772 33

773 03

774 60 RTS

_____ esa (in memoria) _____ equivalente decimale

9. Dite che cosa farebbero, se eseguite, le seguenti istruzioni.

a. A9 LDA 15

15

b. 8D STA 0310

10

03

c. 0A ASL

d. 60 RTS

RISPOSTE AGLI ESERCIZI

1. Decimali; binari.

2. 4

3.	Binario	ESA	Decimale
	1001	9	9
	1101	D	13
	01010111	57	87

4. Caricamento.

5. Memorizzazione.

6. Memorizza il valore contenuto nell'accumulatore nella locazione di memoria 0340.

7. Decimali, esadecimali.

8. 58 esadecimale (in memoria); equivalente decimale 88

(0010 1100 spostato a sinistra = 0101 1000 = 58 ESA

58 ESA = $5 \times 16 + 8 = 88$ decimale)

9. a. LDA 15 carica l'accumulatore con il valore 15 (esadecimale).

b. STA 0310 memorizza il contenuto dell'accumulatore nella locazione di memoria 0310.

c. ASL sposta ogni bit dell'accumulatore di un posto a sinistra.

d. RTS causa un ritorno dalla subroutine dove è usato.

Semplice grafica

SOB

Nel Cap. 3 avete visto come caricare un numero nell'accumulatore, eseguire un'operazione su di esso (spostare i bit a sinistra) e metterlo in memoria. In questo capitolo, imparerete a disegnare punti e linee sullo schermo. Userete alcune delle capacità interne del monitor di linguaggio macchina dell'Apple. Sarete aiutati da alcune subroutine che sono memorizzate permanentemente su ROM (*Read Only Memory*, memoria a sola lettura). Questo vi risparmierà parecchio lavoro, poiché queste routine saranno usate spessissimo nei prossimi programmi.

Senza dubbio, avrete usato molte volte delle subroutine in Basic. Una subroutine in linguaggio macchina lavora nello stesso modo, ma le istruzioni naturalmente sono differenti. In Basic, si usa GOSUB 2000 per dire al computer di andare alla subroutine che inizia alla linea 2000. Poi l'ultima linea della subroutine Basic fa ritornare il computer al programma principale.

In linguaggio macchina, l'istruzione da usare è:

JSR XXXX

Salta alla _____
subroutine

↑

↑

Indirizzo ESA della
la memoria dove
inizia la subroutine

Il codice operativo di linguaggio macchina per JSR è 20. Esempio di come si usa JSR in un programma:

779 20	← JSR F800	Salta alla subroutine
780 00	←	alla locazione di
781 F8		memoria F800 (ESA)

L'ultima istruzione usata nella subroutine deve essere un ritorno al programma principale. In linguaggio macchina, questa sarà:

RTS	Ritorno dalla subroutine
-----	--------------------------

Se state usando una delle subroutine interne dell'Apple, non dovete preoccuparvi dell'istruzione RTS, perché c'è già.

Disegno di un punto sul video

Nel nostro programma per disegnare un punto, il primo passo nell'imparare ad usare i grafici, useremo tre subroutine. Se avete usato il Basic Applesoft, sapete che per fare questo sono necessari parecchi passi di programma. Nel progettare il nostro programma in linguaggio macchina, sappiamo di dovere:

Istruzioni Basic equivalenti

- | | |
|-------------------------------------|------------|
| 1. Cancellare lo schermo | HOME |
| 2. Attivare il modo grafico | GR |
| 3. Scegliere il colore da usare | COLOR = 15 |
| 4. Scegliere la posizione del video | C = 5 |
| dove disegnare il punto | R = 32 |
| 5. Disegnare il punto | PLOT C,R |
| 6. Ritornare al sistema operativo | |
| in Basic | RETURN |

Ecco un semplice programma per disegnare un punto. Ogni funzione del programma è presentata in un blocco in accordo con quanto scritto sopra. Il computer stampa la prima colonna. Voi dovete scrivere la seconda.

- | | |
|-----------------------------------|---|
| 1. COMMENTO **CANCELLA IL VIDEO** | |
| 768 20 | ← JSR FC58 |
| 769 58 | ← |
| 770 FC | |
| | Salta alla subroutine a FC58. Una subroutine interna. |

2. COMMENTO ****ABILITA IL MODO GRAFICO****

771 20 ←JSR FB40

772 40 ←

773 FB

Salta alla subroutine a FB40. Un'altra subroutine interna.

3. COMMENTO ****SCEGLIE IL COLORE****

774 A9 ←LDA FF

775 FF ←

Carica l'accumulatore con il valore del colore 15 o F in ESA (entrambi i byte).

776 85 ←STA 0030

777 30 ←

Memorizzalo alla locazione 0030.

4. COMMENTO ****POSIZIONE DELLO SCHERMO****

778 A0 ←LDY 05

779 05 ←

Carica il registro Y con il numero di colonna, 5.

780 A9 ←LDA 20

781 20 ←

Carica l'accumulatore con la riga, 32 (20 ESA).

5. COMMENTO ****DISEGNA IL PUNTO****

782 20 ←JSR F800

783 00 ←

784 F8

Salta alla subroutine a F800. Un'altra subroutine interna.

6. COMMENTO ****RITORNA AL BASIC****

785 60 ←RTS

Ritorna al sistema operativo in Basic.

Si noti l'istruzione di memorizzazione (85 esadecimale) alla locazione di memoria 776. Ordinariamente, un caricamento da, o una memorizzazione in una locazione richiede due byte addizionali per dare l'indirizzo di memoria completo. L'unità centrale di processo del 6502 riconosce il codice 85 come un'istruzione speciale che fornisce soltanto il byte meno significativo dell'indirizzo. Il computer "comprende" che il byte più significativo di questa istruzione speciale di pagina zero è zero. Quindi, queste sono chiamate istruzioni di pagina zero e possono essere eseguite più velocemente di quelle dove è richiesto un indirizzo di due byte.


Il funzionamento di questo programma dipende dalla corretta esecuzione della subroutine alla locazione F800 che disegna il punto. I passi 3 e 4 del programma forniscono dei valori che devono essere usati dalla subroutine che traccia i punti.

PASSO 3 Mette il valore del colore (0-15) nella locazione 0030. Si noti che questa istruzione di memorizzazione, STA, usa soltanto l'ultima parte dell'indirizzo (la prima parte, 00, non occorre). Il valore del colore è dato alla locazione di programma 775 ed è inserito come numero esadecimale da 09 a F (da 0 a 15 con numeri decimali). La corretta cifra esadecimale va inserita in entrambi i byte.

Esempio

Arancio: A9 LDA 99
 99
 85 STA 30
 30

9 in entrambe le
cifre ESA



I valori dei colori sono dati nella seguente tabella.

TABELLA DEI COLORI

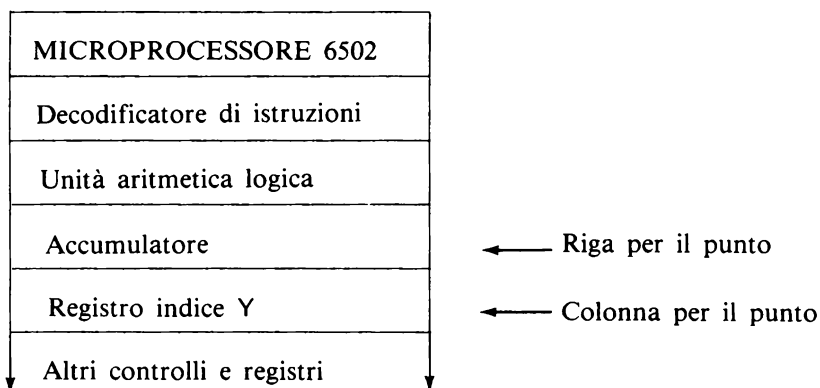
<i>Valore del colore</i>	<i>Valore ESA</i>	<i>Colore</i>
0	0	Nero
1	1	Magenta
2	2	Blu scuro
3	3	Viola
4	4	Verde scuro
5	5	Grigio
6	6	Blu
7	7	Azzurro
8	8	Marrone
9	9	Arancio
10	A	Grigio
11	B	Rosa
12	C	Verde
13	D	Giallo
14	E	Acquamarina
15	F	Bianco

Quando la subroutine di tracciamento verrà eseguita, preleverà il valore del colore nella locazione di memoria 0030. Perciò il programma in linguaggio macchina deve memorizzare questo valore proprio lì (viene fatto alle locazioni 776 e 777 del programma). Si deve anche dire alla subroutine dove disegnare il punto sullo schermo. Questa informazione è fornita nel passo 4 del programma.

PASSO 4 Fornisce la colonna e la riga dove si vuole disegnare il punto. Nei grafici a bassa risoluzione, questi valori possono variare da 0 a 39 compresi. La colonna viene caricata nel registro Y (una speciale locazione, usata da parecchie istruzioni del linguaggio macchina, che contiene 8 bit). La riga è caricata nell'accumulatore. Ri-

cordate che queste sono istruzioni del linguaggio macchina, quindi i valori devono essere in formato esadecimale.

PASSO 5 La subroutine disegna-punti guarda nel registro Y e nell'accumulatore per trovare la riga e la colonna che indicano dove disegnare il punto, e poi lo disegna.



PASSO 6 Dopo che il punto è stato disegnato, l'istruzione di ritorno dalla subroutine riporta il programma al sistema operativo in Basic alla posizione che segue l'istruzione CALL S della linea 800 (vedere Sistema operativo, Cap. 2).

Le subroutine ai passi 1 e 2 cancellano semplicemente lo schermo ed abilitano il modo graphics in bassa risoluzione, cosicché i punti possano essere disegnati.

Per questo programma, sarebbe conveniente essere in grado di cambiare il valore del colore, la colonna e la riga del punto. Per fare questo facilmente, aggiungete queste linee al programma del sistema operativo.

```
810 INPUT "VUOI CAMBIARE DATI (SI O NO)?";A$
820 IF A$ = "SI" GOTO 420
```

Poi, quando volete cambiare dei valori dopo l'esecuzione, potete cambiare il valore del codice a 775, la colonna a 779 e la riga a 781. Questa modifica vi risparmia la seccatura di dover riscrivere tutto il programma.

Ecco una semplice visualizzazione subito dopo l'inserimento.

18
byte {

```

ECCO IL TUO PROGRAMMA
768 20  ← Porta il cursore in alto a sinistra e
769 58      cancella lo schermo
770 FC
771 20  ← Abilita il modo grafico
772 40
773 FB
774 A9  ← Sceglie il colore
775 FF
776 85  ← Memorizza il valore del colore
777 30
778 A0  ← Colonna
779 05
780 A9  ← Riga
781 20
782 20  ← Disegna il punto
783 00
784 F8
785 60  ← Ritorno
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■

```

Ecco il video dopo l'esecuzione.

Punto bianco disegnato alla
colonna 5, riga 32

□

VUOI CAMBIARE DATI (SI O NO)? ■

Ora cambiate il colore a 3 (viola), la colonna a 14 (esadecimale) e la riga a 14 (esadecimale). Questo metterà un punto color viola al centro dello schermo. Dopo aver risposto SI alla domanda, il video mostrerà:

4 linee di
testo {

□

VUOI CAMBIARE DATI (SI O NO)?SI

SE CI SONO MODIFICHE BATTI L'INDIRIZZO

ALTRIMENTI BATTI 99

?■

Scrivete 775 e premete RETURN. Le 4 linee di testo in fondo allo schermo ora saranno:

```
SE CI SONO MODIFICHE BATTI L'INDIRIZZO  
ALTRIMENTI BATTI 99  
?775  
775 DATO=?■
```

Scrivete 33. Le 4 linee in fondo ora risulteranno:

```
ALTRIMENTI BATTI 99  
?775  
775 DATO=?33  
ALTRE MODIFICHE (SI O NO)?■
```

Scrivete SI e premete RETURN. Le 4 linee ora saranno:

```
ALTRE MODIFICHE (SI O NO)? SI  
SE CI SONO MODIFICHE BATTI L'INDIRIZZO  
ALTRIMENTI BATTI 99  
?■
```

Scrivete 779 e premete RETURN. Le 4 linee mostreranno:

```
SE CI SONO MODIFICHE BATTI L'INDIRIZZO  
ALTRIMENTI BATTI 99  
?779  
779 DATO=?■
```

Scrivete 14. Le 4 linee saranno:

```
ALTRIMENTI BATTI 99  
?779  
779 DATO=?14  
ALTRE MODIFICHE (SI O NO)?■
```

Scrivete SI e premete RETURN. Le 4 linee mostreranno:

```
ALTRE MODIFICHE (SI O NO)? SI
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■
```

Scrivete 781 e premete RETURN. Le 4 linee risulteranno:

```
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?781
781 DATO = ?■
```

Scrivete 14. Le 4 linee saranno:

```
ALTRIMENTI BATTI 99
?781
781 DATO = ?14
ALTRE MODIFICHE (SI O NO)? ■
```

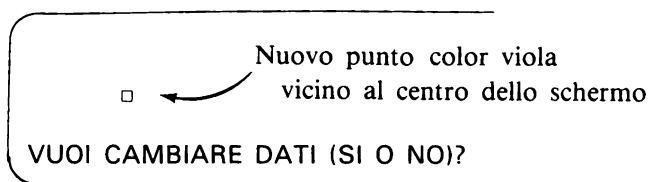
Scrivete NO e premete RETURN

```
783 00
784 F8
785 60
ALTRE MODIFICHE (SI O NO)? ■
```

Scrivete NO e premete RETURN. Le 4 linee mostreranno:

```
784 F8
785 60
PEMI UN TASTO QUALSIASI PER CONTINUARE
ALTRE MODIFICHE? NO
```

Premete un tasto qualsiasi per vedere il disegno del nuovo punto



Ora dipende da voi. Se volete disegnare dei punti con altri colori in altre posizioni, scrivete SI e ripetete il procedimento. Altrimenti scrivete NO, ed il computer ritornerà al sistema operativo. Sarete allora pronti per inserire il prossimo programma.

DISEGNO NEI QUATTRO ANGOLI

Prima di lasciare la tecnica di disegno per punti, scriviamo un programma per mettere un punto in ogni angolo dell'area grafica. I punti saranno:

<i>Colonna</i>		<i>Riga</i>		
Dec.	ESA	Dec.	ESA	
0	0	0	0	In alto a sinistra
0	0	39	27	In basso a sinistra
39	27	0	0	In alto a destra
39	27	39	27	In basso a destra

Il programma è simile al precedente. La differenza è che le parti 4 e 5 verranno ripetute, una volta per ogni ulteriore punto.

PROGRAMMA PER DISEGNARE NEI QUATTRO ANGOLI

```

1 { 768 20  Cancella lo schermo
   769 58
   770 FC

2 { 771 20  Abilita il modo grafico
   772 40
   773 FB
  
```

3	{	774 A9	Sceglie il colore bianco
		775 FF	
		776 85	Lo memorizza in 0030
		777 30	
4	{	778 A0	Colonna 0
		779 00	
		780 A9	Riga 0
		781 00	
5	{	782 20	Disegna nell'angolo in alto a sinistra
		783 00	
		784 F8	
4	{	785 A9	Cambia riga, lascia la stessa colonna
		786 27	
5	{	787 20	Disegna nell'angolo in basso a sinistra
		788 00	
		789 F8	
4	{	790 A0	Cambia colonna, lascia la stessa riga
		791 27	
		792 A9	Ricarica la riga nell'accumulatore
		793 27	
5	{	794 20	Disegna nell'angolo in basso a destra
		795 00	
		796 F8	
4	{	797 A9	Cambia riga, lascia la stessa colonna
		798 00	
5	{	799 20	Disegna nell'angolo in alto a destra
		800 00	
		801 F8	
		802 60	Ritorna al sistema operativo in Basic

Questo è il più lungo programma in linguaggio macchina che avete fatto finora. Ci sono 35 byte. Quando il programma sarà stato inserito completamente, il computer visualizzerà le prime 20 linee come segue:

ECCO IL TUO PROGRAMMA

768 20
769 58
770 FC
771 20
772 40
773 FB
774 A9
775 FF
776 85
777 30
778 A0
779 00
780 A9
781 00
782 20
783 00
784 F8
785 A9
786 27
787 20

PREMI UN TASTO QUALSIASI PER CONTINUARE



Per vedere il resto del programma, premete un tasto qualsiasi. Il computer cancellerà lo schermo e mostrerà le prossime 20 linee, se ce ne sono 20. Ecco quello che vedrete questa volta:

788 00
789 F8
790 A0
791 27
792 A9
793 27
794 20
795 00
796 F8
797 A9
798 00
799 20
800 00
801 F8
802 60

PREMI UN TASTO QUALSIASI PER CONTINUARE



Vedete che adesso è stato listato tutto il programma. Premendo un qualsiasi tasto, avrete la possibilità di effettuare correzioni. Le linee che seguiranno sono:

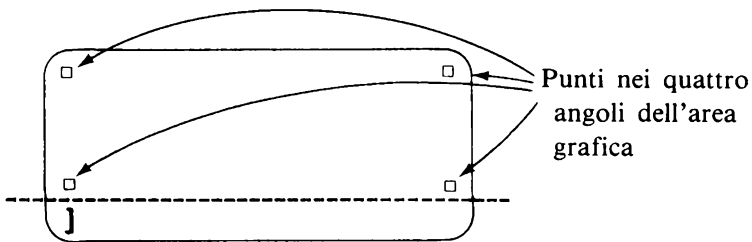
```
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■
```

Scrivete 99 se non ci sono modifiche.

Il programma visualizza

PREMI UN TASTO QUALSIASI PER PARTIRE

Premete un tasto qualsiasi e vedrete:



Ancora una volta potete aggiungere momentaneamente delle linee al programma del sistema operativo in Basic per poter cambiare dei valori.

DISEGNO DI UNA LINEA ORIZZONTALE

Se è possibile disegnare un punto, si può anche tracciare una linea disegnando una serie di punti. L'ultimo programma, per il disegno di soli quattro punti, era piuttosto lungo. Fortunatamente il monitor di linguaggio macchina dell'Apple contiene una subroutine interna che disegna una linea orizzontale. Tutto quello che dobbiamo fare è memorizzare la colonna dell'ultimo punto della linea nella locazione di memoria 002C dove la subroutine che traccia la linea la possa trovare.

La subroutine usata al passo 6 è equivalente all'istruzione del Basic Applesoft:

```
HLIN 16,32 AT 20
```

Ecco il programma completo organizzato in parti funzionali.

PROGRAMMA LINEA ORIZZONTALE

- 1 COMMENTO ****CANCELLA LO SCHERMO****
768 20 JSR FC58 Subroutine interna
769 58
779 FC
- 2 COMMENTO ****ABILITA IL MODO GRAFICO****

771 20 JSF FB40 Subroutine interna
772 40
773 FB
- 3 COMMENTO ****SCEGLIE IL COLORE****
774 A9 LDA FF Carica il colore bianco
775 FF

776 85 STA 30 Memorizza in 0030
777 30
- 4 COMMENTO ****DA' L'ULTIMO PUNTO****

778 A9 LDA 20 Ultimo punto a 20 (32 decimale)
779 20

780 85 STA 2C Memorizza in 002C
781 2C
- 5 COMMENTO ****DA' IL PRIMO PUNTO****

782 A0 LDY 10 Primo punto a 10 (16 decimale)
783 10

784 A9 LDA 14 Carica l'accumulatore con la riga
785 14 14 (20 decimale)
- 6 COMMENTO ****DISEGNA LA LINEA****

786 20 JSR F819 Subroutine interna
787 19
788 F8
- 7 COMMENTO ****RITORNO AL BASIC****

789 60 RTS Ritorno al sistema operativo

Se confrontate questo programma con il programma per tracciare punti, spiegato precedentemente in questo capitolo, scoprirete che sono molto simili. Nel nostro nuovo programma, dobbiamo dare il valore finale (colonna) prima di disegnare la linea. La subroutine che disegna la linea è situata ad una locazione di memoria differente da quella per la subroutine che disegnava un punto nel vecchio programma.

Dopo aver inserito il programma per mezzo del sistema operativo in Basic, il programma verrà visualizzato come al solito. Ci sono 22 byte che iniziano alla locazione 768.

ECCO IL TUO PROGRAMMA

```
768 20
769 58
770 FC
771 20
772 40
773 FB
774 A9
775 FF
776 85
777 30
778 A9
779 20
780 85
781 2C
782 A0
783 10
784 A9
785 14
786 20
787 19
PREMI UN TASTO QUALSIASI PER CONTINUARE
■
```

Controllate se è corretto. Poi premete un tasto qualsiasi.

```
788 F8
789 60
PREMI UN TASTO QUALSIASI PER CONTINUARE
■
```

Dopo aver premuto un tasto:

```

788 F8
789 60
PREMI UN TASTO QUALSIASI PER CONTINUARE
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■

```

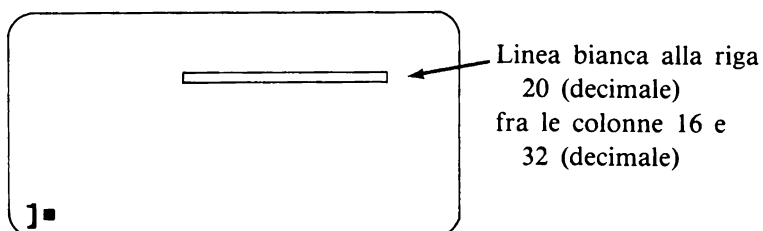
Se è tutto giusto, scrivete 99.

```

788 F8
789 60
PREMI UN TASTO QUALSIASI PER CONTINUARE
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?99
PREMI UN TASTO QUALSIASI PER PARTIRE
■

```

Ora quando premete un tasto, vedrete:



Ancora una volta potete voler fare delle modifiche. Le linee che avete usato prima lavorerebbero bene nel sistema operativo.

```

810 INPUT "VUOI CAMBIARE DATI (SI O NO)?";A$
820 IF A$ = "SI" GOTO 300

```

Se vengono usate queste linee, il programma visualizzerà la linea e poi vi permetterà di modificare i dati ad ogni indirizzo che volete. Se le linee 810 e 820 fanno parte del sistema operativo, sarà facile passare dal programma Linea Orizzontale al prossimo programma che disegna linee verticali.

DISEGNO DI LINEE VERTICALI

Le parti “Cancella lo schermo”, “Abilita il modo grafico” e “Scegli il colore”, del programma Linea Orizzontale, andranno bene anche per il programma Linea Verticale, perciò le lasceremo come sono.

La parte 4, la routine “Dà l’ultimo punto”, richiede soltanto la modifica della locazione per la memorizzazione dell’ultimo punto della verticale. Questo sarà ora memorizzato nella locazione 002D, invece della 002C come nel programma Linea Orizzontale. La subroutine che disegna linee verticali cerca il suo punto finale in 002D. Questo richiede che sia cambiato il valore contenuto nella locazione 781.

La parte 5 del programma Linea Orizzontale dava il punto iniziale della linea (la colonna dove iniziava la linea) ed anche la linea dove la linea orizzontale doveva essere disegnata. Dovete cambiare il punto iniziale della linea orizzontale nella colonna alla quale la linea verticale sarà disegnata. Per fare questo, cambiate il valore memorizzato nella locazione 783. Dovete anche modificare la locazione 785 del programma, che contiene la riga alla quale la linea orizzontale veniva disegnata. Questa deve ora essere cambiata nella riga iniziale alla quale la linea verticale sarà disegnata.

Deve essere modificata anche la parte 6. Invece di saltare alla subroutine che disegna una linea orizzontale, si deve passare alla subroutine che disegna una linea verticale. Dovete cambiare soltanto la locazione 787 del programma.

Ecco un sommario dei quattro cambiamenti:

Cambiare 781 da 2C a 2D	Locazione dell’ultimo punto
Cambiare 783 da 10 a 14	Colonna della linea
Cambiare 785 da 14 a 10	Primo punto della linea
Cambiare 787 da 19 a 28	Indirizzo della subroutine F828 invece di F819

La subroutine alla locazione F828 è equivalente all’istruzione del Basic:

VLIN 16,32 AT 20

Se al sistema operativo sono state aggiunte le linee 810 e 820, ed è stato eseguito il programma Linea Orizzontale, il computer finirà con la domanda che chiede se ci sono modifiche.

VUOI CAMBIARE DATI (SI O NO)? ■

Scrivete SI e premete RETURN. Le quattro linee in fondo mostreranno:

VUOI CAMBIARE DATI (SI O NO)?SI
 SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?■

Scrivete 781 e premete RETURN

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?781
 781 DATO = ?■

Scrivete: 2D

← Locazione di memoria dove è
 memorizzato il punto finale

ALTRIMENTI BATTI 99
 ?781
 781 DATO = ?2D
 ALTRE MODIFICHE (SI O NO)?■

Scrivete SI e premete RETURN.

ALTRE MODIFICHE (SI O NO)?SI
 SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?■

Scrivete 783 e premete RETURN

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?783
 783 DATO = ?■

Scrivete: 14

← Colonna della linea

ALTRIMENTI BATTI 99
 ?783
 783 DATO = ?14
 ALTRE MODIFICHE (SI O NO)?■

Scrivete SI e premete RETURN

ALTRE MODIFICHE (SI O NO)?SI
 SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?■

Scrivete 785 e premete RETURN

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?785
 785 DATO = ?■

Scrivete 10

← Punto d'inizio della linea

ALTRIMENTI BATTI 99
 ?785
 785 DATO = ?10
 ALTRE MODIFICHE (SI O NO)?■

Scrivete SI e premete RETURN

ALTRE MODIFICHE (SI O NO)?SI
 SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?■

Scrivete 787 e premete RETURN

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
 ALTRIMENTI BATTI 99
 ?787
 787 DATO = ?■

Scrivete 28

← Indirizzo di una nuova
 subroutine (byte meno
 significativo di F828)

ALTRIMENTI BATTI 99

?787

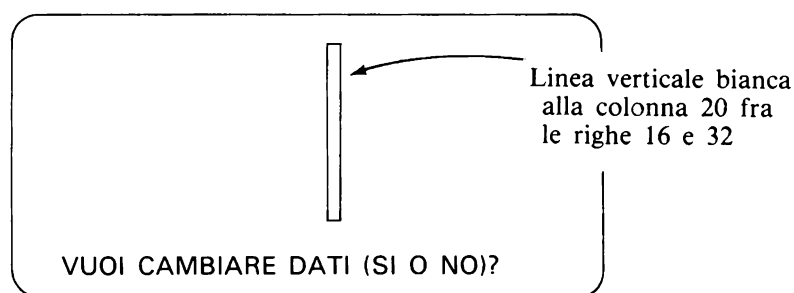
787 DATO = ?28

ALTRE MODIFICHE (SI O NO)?

Scrivete NO

Il vostro programma viene poi listato in due parti, come per il programma linea orizzontale.

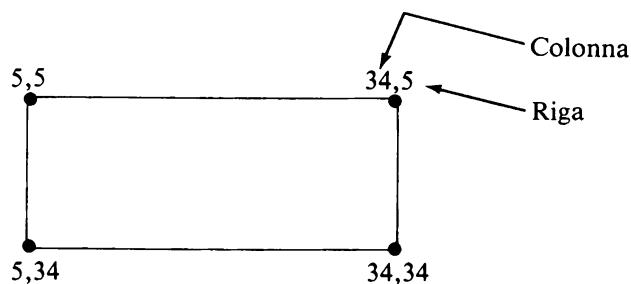
Quando eseguite il programma, vedrete:



Se volete provare ancora il programma Linea Verticale, fate le vostre modifiche a questo punto ed eseguitelo ancora. Diverrete presto esperti nel disegno di punti e linee in qualsiasi posizione dello schermo. Sarete quindi pronti per un programma che unisce delle linee per formare un rettangolo.

DISEGNO DI UN RETTANGOLO

Dato che sapete come disegnare linee orizzontali e verticali, sarete in grado di disegnare un rettangolo unendo coppie di linee. Il programma che segue disegnerà il rettangolo mostrato.



Non vengono usate istruzioni nuove, ma il programma è più lungo (45 byte questa volta). Mostriamo ancora una volta il programma suddiviso in varie parti, sia con i codici mnemonici che con i codici operativi.

PROGRAMMA RETTANGOLO

1 COMMENTO **CANCELLA LO SCHERMO**

768 20	JSR FC58	Salta alla subroutine FC58
769 58		
779 FC		

2 COMMENTO **ABILITA IL MODO GRAPHICS**

771 20	JSF FB40	Salta alla subroutine FB40
772 40		
773 FB		

3 COMMENTO **SCEGLIE IL COLORE**

774 A9	LDA FF	Carica l'accumulatore con il colore
775 FF		
776 85	STA 0030	Memorizza il valore nella memoria 0030
777 30		

4 COMMENTO **PUNTI FINALI DI ENTRAMBE LE LINEE**

778 A9	LDA 20	Fine alla colonna e alla riga 34 (decimale)
779 22		
780 85	STA 2C	Colonna finale memorizzata a 002C
781 2C		
782 85	STA 2D	Riga finale memorizzata a 002D
783 2D		

5 COMMENTO **INIZIO ORIZZONTALE E RIGA**

784 A0	LDY 05	Inizio delle linee orizzontali
785 05		
786 A9	LDA 05	Riga 5
785 05		

6 COMMENTO **DISEGNA IL LATO IN ALTO DEL RETTANGOLO**

788 20	JSR F819	Salta alla subroutine F819
787 19		
790 F8		

7 COMMENTO **RISTABILISCE L'INIZIO E LA RIGA**

791 A0	LDY 05	Ristabilisce il punto iniziale
792 05		
793 A9	LDA 22	Si sposta alla riga 34 (dec.)
794 22		

8 COMMENTO **DISEGNA LA BASE DEL RETTANGOLO**

795 20	JSR F819	Salta alla subroutine F819
796 19		
797 F8		

9 COMMENTO **INIZIO VERTICALE E COLONNA**

798 A0	LDY 05	Colonna 5
799 05		
800 A9	LDA 05	Inizio delle linee verticali
801 05		

10 COMMENTO **DISEGNA IL LATO SINISTRO**

802 20	JSR F828	Salta alla subroutine F828
803 28		
804 F8		

11 COMMENTO **RISTABILISCE L'INIZIO E LA COLONNA**

805 A0	LDY 22	Si sposta alla colonna 34 (dec.)
806 22		
807 A9	LDA 05	Ristabilisce il punto iniziale
808 05		

12 COMMENTO **DISEGNA IL LATO DESTRO**


```

809 20      JSR F828      Si sposta alla colonna 34 (dec.)
810 28
811 F8

```

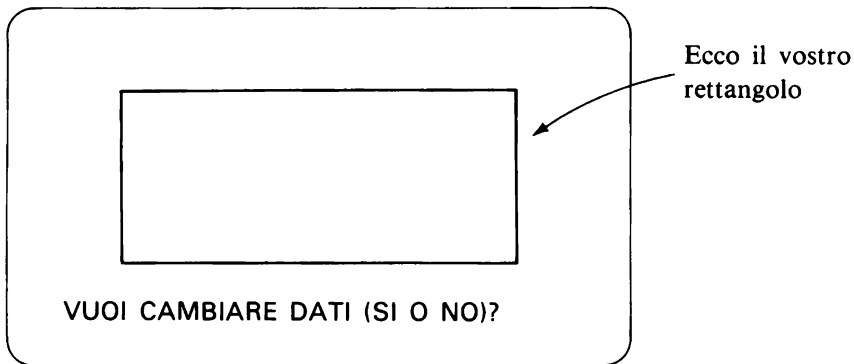
13 COMMENTO **RITORNO AL BASIC**

```

812 60      RTS

```

Questo programma ha 45 byte. Dopo averlo inserito, il computer lo visualizzerà in 3 parti (20 linee alla volta). Se ci sono errori, correggeteli. Altrimenti eseguitelo. Ecco quello che vedrete:



SOMMARIO

Le subroutine in linguaggio macchina sono state usate moltissimo in questo capitolo. Il monitor di linguaggio macchina dell'Apple ha parecchi programmi interni per questi scopi. Avete usato le subroutine per cancellare lo schermo, per abilitare il modo graphics, per disegnare punti, linee orizzontali e verticali.

Avete anche usato le istruzioni di caricamento e memorizzazione per muovere i dati alle locazioni dove possano essere trovati dalle subroutine appropriate.

Avete imparato ad usare queste istruzioni:

1. JSR (*Jump to Subroutine*, salta alla subroutine) Usata in modo assoluto per utilizzare le subroutine interne nei vostri programmi.

Esempio 20 codice op.
 19 byte meno significativo dell'indirizzo
 FB byte più significativo dell'indirizzo

2. LDY (*LoaD Y register*, carica il registro Y) Usata in modo immediato. Questo valore è stato usato da una subroutine.

Esempio A0 codice op.
 22 dato caricato nel registro Y

3. STA (*STore Accumulator*, memorizza l'accumulatore) Usata, questa volta, nel modo pagina zero. Questo modo viene usato soltanto quando i dati devono essere trasferiti a o da locazioni di memoria basse (quelle dove il byte più significativo dell'indirizzo è zero). La memoria compresa tra 0000 e 00FF è chiamata *memoria di pagina zero*.

Esempio 85 codice op.
 30 byte meno significativo dell'indirizzo

Le istruzioni del linguaggio macchina che avete usato finora in questo libro sono:

<i>Codice mnemonico</i>	<i>Modi di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
LDA	Immediato	A9	2	Carica l'accumulatore
LDY	Immediato	A0	2	Carica il registro Y
STA	Assoluto	8D	3	Mette l'accumulatore
STA	Pagina zero	85	2	in memoria
JSR	Assoluto	20	3	Salto alla subroutine
RTS	Implicito	60	1	Ritorno dalla subroutine
ASL	Accumulatore	0A	1	Sposta i bit dell'accumulatore a sinistra

L'accumulatore, il registro Y e certe locazioni di memoria sono state usate per memorizzare valori utilizzabili dalle subroutine interne. Ecco un sommario dei loro usi.

<i>Subroutine</i>	<i>Reg. Y</i>	<i>Accum.</i>	<i>0030</i>	<i>002C</i>	<i>002D</i>
Cancella lo schermo	—	—	—	—	—
Abilita il modo grafico	—	—	—	—	—
Disegna un punto	Colonna	Riga	Colore	—	—
Linea orizzontale	Colonna iniziale	Riga	Colore	Colonna finale	—
Linea verticale	Colonna	Riga iniziale	Colore	—	Riga finale

Siete ben avviati alla programmazione in linguaggio macchina. Nel prossimo capitolo vedremo come visualizzare caratteri alfanumerici sullo schermo.

ESERCIZI

Completare gli spazi vuoti.

- Il codice mnemonico JSR è l'abbreviazione di _____

- I valori dei colori usati nei programmi in linguaggio macchina sono numeri esadecimali a due cifre. Quali, tra quelli che seguono, sono corretti?
a. CC b. 0H c. FG d. 33

- Per disegnare un punto sullo schermo usando le subroutine interne, prima si devono fare queste cose.
a. abilitare il modo grafico
b. scegliere il valore del colore
c. caricare il registro Y con _____
d. caricare l'accumulatore con _____
- Che range di valori può essere usato per disegnare un punto, nel modo grafico a bassa risoluzione che abbiamo usato nel Cap. 3?
a. Dalla colonna _____ alla _____ incluse
b. Dalla riga _____ alla _____ incluse

5. Il programma interno che disegna una linea verticale e quello che disegna una linea orizzontale usano _____
(le medesime, differenti)
subroutine.
6. Spiegate quando si può usare l'istruzione STA (codice op. 85) nel modo pagina zero.

RISPOSTE AGLI ESERCIZI

1. *Jump to SubRoutine* salta alla subroutine
2. a e b
3. c. numero di colonna (0-27 esadecimale)
d. numero di riga (0-27 esadecimale)
4. a. colonne da 0 a 39 decimale (o da 0 a 27 esadecimale)
b. righe da 0 a 39 decimale (o da 0 a 27 esadecimale)
5. Differenti (linea orizzontale a F819, linea verticale a F828)
6. Quando i dati devono essere trasferiti dall'accumulatore nella memoria di pagina zero (da 0000 a 00FF), il byte più significativo è 00, e, per un'istruzione nel modo pagina zero, non è necessario.

Visualizzare testi

SOB

Il video è una finestra attraverso cui si può vedere che cosa sta facendo il computer. Lo schermo può visualizzare il contenuto di certe locazioni di memoria. Nel Cap. 4, avete fatto dei disegni sullo schermo usando le subroutine del computer. In questo capitolo, imparerete a mettere dei testi direttamente nell'area di memoria del video, cosicché si possano vedere dei messaggi.

Sarete introdotti ad alcune nuove istruzioni, che verranno usate per realizzare dei cicli simili a quelli di tipo IF...THEN e FOR...NEXT che avete usato in Basic.

Il calcolatore ha due registri che possono essere usati come "contatori". Questi contatori (i registri X e Y) possono essere utilizzati per *indicare*, o *contare*, il numero di volte che il computer esegue un ciclo. Possono anche essere usati per indicare le locazioni di memoria da cui caricare, o dove memorizzare, dei dati. Questa operazione porta all'uso di un nuovo modo di indirizzamento, chiamato indirizzamento assoluto indicizzato.

 VISUALIZZAZIONE DI UN CARATTERE

Inizieremo con un semplice programma che visualizza la lettera A nell'angolo in alto a sinistra dello schermo. Tutte le istruzioni che appaiono in questo programma sono già state usate. L'unica cosa nuova è l'uso del codice ASCII. Poiché il computer può capire solo istruzioni numeriche e dati, tutti i caratteri alfabetici e di punteggiatura (così come certi altri caratteri speciali) devono essere dati in forma numerica. A questo scopo vengono usati i codici ASCII. (Una lista completa di questi codici è data più avanti, in questo capitolo.)

PROGRAMMA VISUALIZZA UNA LETTERA


1. COMMENTO **CANCELLA LO SCHERMO**

```
768 20 JSR FC58    Salto alla subroutine
769 58
779 FC
```

2. COMMENTO **SCEGLIE LA LETTERA 'A'**

```
771 A9 LDA C1      Carica l'accumulatore con il codice
772 C1              ASCII di A
```

C1 è il
codice ESA per 'A'



3. COMMENTO **LA VISUALIZZA**

```
773 8D STA 0400    Memorizzalo nell'area di memoria
774 00              del video
775 04
```

4. COMMENTO **RITORNO AL BASIC**

```
776 60 RTS        Ritorno al sistema operativo
```

Avete visto la parte 1 precedentemente. E la subroutine interna che cancella lo schermo.

Nella parte 2, viene caricato nell'accumulatore il codice ASCII della lettera A (valore esadecimale C1) Altri valori darebbero altri caratteri.

La parte 3 usa il modo di indirizzamento assoluto per l'istruzione STA. La locazione di questo elemento di memoria (0400 esadecimale) è l'angolo in alto a sinistra del video.

La parte 4 ritorna il controllo al sistema operativo Basic.

Inserite il programma usando il sistema operativo e poi eseguitelo. Vedrete la lettera A comparire nell'angolo in alto a sinistra dello schermo. Il cursore lampeggiante indica che il programma è ritornato al sistema operativo.

Il video:



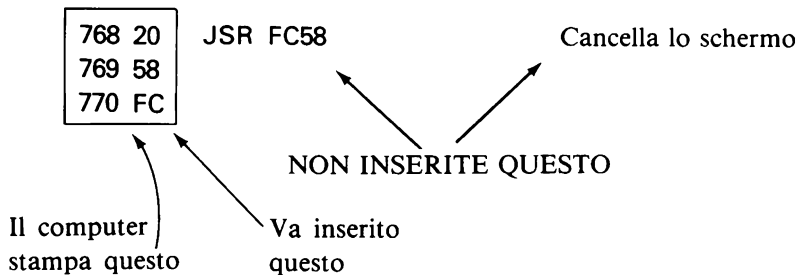
Il prossimo è un programma che visualizza più volte un carattere alfabetico nella parte alta dello schermo. Se volete provare più volte questo programma, aggiungete queste linee al sistema operativo:

```
810 PRINT: PRINT
820 INPUT "VUOI CAMBIARE DATI (SI O NO)?";A$
830 IF A$ = "SI" THEN 420
```

Poi usate il sistema operativo per inserire questo programma. Non inserite i commenti, i codici mnemonici o gli operandi.

Esempio

1 COMMENTO **CANCELLA LO SCHERMO**



Dovete inserire solo gli indirizzi e i dati. Il resto delle note e dei commenti serve soltanto per aiutarvi a capire la funzione di ogni parte del programma. Questo programma contiene 16 byte.

PROGRAMMA VISUALIZZA CARATTERI

```
1 COMMENTO **CANCELLA LO SCHERMO**
  768 20 JSR FC58    Cancella ancora lo schermo
  769 58
  779 FC
```

2 COMMENTO **PONE L'INDICE A ZERO**

771 A0	LDY 0	Usato per indicare la memoria
772 00		del video e per uscire dal ciclo

3 COMMENTO **SCEGLIE IL CARATTERE**

773 A9	LDA C1	Carica la A proprio come nel
774 C1		programma precedente

4 COMMENTO **ECCO IL CICLO**

Ciclo	→ 775 99	STA 0400, Y	Memoria nella locazione 0400 +
	776 00		il contenuto del registro Y
	777 04		
	778 C8	INY	Incrementa il registro Y per la
			prossima locazione
	779 C0	CPY 28	Confronta Y con 401 (decimale)
	780 28		
	781 D0	BNE F8	Salta (se i due valori non sono
	782 F8		uguali) alla locazione 0775

5 COMMENTO **RITORNO AL BASIC**

783 60	RTS
--------	-----

Seguiremo ora il programma passo per passo e discuteremo le istruzioni usate, e l'azione di ognuna di esse quando sarà eseguito il programma. Nella parte 1 viene cancellato lo schermo. Questa subroutine sarà usata nella maggior parte (se non in tutti) dei nostri programmi.

Nella parte 2, il registro Y viene caricato con il valore zero. L'istruzione LDY è stata discussa nel capitolo 4. In questo programma, il registro Y è usato per tener conto di quante volte viene eseguito il ciclo della parte 4. Lo poniamo a zero in preparazione del conteggio.

Registro Y

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 (in binario)

Nella parte 3, l'istruzione LDA è usata in modo immediato per caricare il valore esadecimale C1 nell'accumulatore. Questo valore sarà usato per visualizzare la lettera A nella parte 4 del programma.

Accumulatore

1	1	0	0	0	0	1
---	---	---	---	---	---	---

 (in binario)

Una lista completa dei codici ASCII è data nell'Appendice. C1, il codice della lettera A, è il solo codice ASCII usato in questo programma.

La parte 4 forma il ciclo che stampa la lettera A in ogni posizione lungo la linea più alta del video. Il ciclo lavora in questo modo:

La prima volta che si percorre il ciclo, registro Y = 0

- 1 L'istruzione STA memorizza il codice di A nella locazione di memoria 0400+0 (prima posizione, linea più alta). Notate che il valore memorizzato in Y (0 a questo punto) viene aggiunto al valore 0400 per determinare la locazione da usare per la memorizzazione.
- 2 L'istruzione INY (codice operativo C8) incrementa di uno il valore del registro Y. Perciò il registro Y ora contiene 1.
- 3 L'istruzione CPY (codice operativo C0) confronta il valore del registro Y (ora 1) con il valore esadecimale 28 (il secondo byte dell'istruzione).
- 4 L'istruzione BNE (codice operativo D0) causa un salto all'indietro all'inizio del ciclo se i valori confrontati nel passo 3 *non* sono uguali. Poiché 1 e 28 non sono uguali, questa volta il computer ritorna all'inizio del ciclo. Se i valori fossero uguali, il calcolatore andrebbe alla parte 5 del programma.

La seconda volta che si percorre il ciclo, registro Y = 1

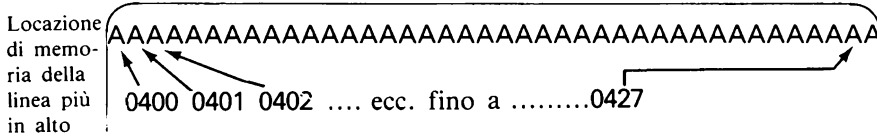
- 1 L'istruzione STA memorizza A nella locazione 0400+1 (0401 è la seconda posizione della linea in alto).
- 2 Il registro Y è incrementato dall'istruzione INY, ed ora ha valore 2.
- 3 Il valore 2 del registro Y viene confrontato con 28.
- 4 Poiché 2 non è uguale a 28, l'istruzione BNE manda ancora una volta il computer all'inizio del ciclo.

La terza volta che si percorre il ciclo, registro Y = 2

.
.
.
ecc.
.
.
.
.

Ogni volta che si percorre il ciclo, il carattere A viene stampato un posto a destra rispetto alla posizione precedente.

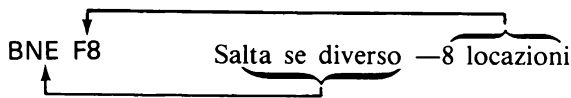
La ventottesima volta che si percorre il ciclo, Y = 27 esadecimale



L'istruzione INY alla locazione 778 del programma è un'istruzione ad indirizzamento implicito che incrementa il valore del registro Y. È simile all'istruzione $Y = Y + 1$ del Basic, ed è usata per incrementare la locazione in cui memorizzare il carattere A ogni volta che si percorre il ciclo. L'istruzione CPY 28 è qui usata nel modo di indirizzamento immediato. Il valore corrente del registro Y è confrontato con il valore esadecimale 28. Questo confronto abilita il computer a decidere, nel successivo passo, se tornare o no all'inizio del ciclo. Assieme all'istruzione BNE, CPY ha una funzione simile all'istruzione del Basic:

IF $N \neq 28$ THEN GOTO XX (dove XX è la linea a cui bisogna saltare)

L'ultima nuova istruzione, BNE F8, richiede alcune spiegazioni. È un'istruzione con indirizzamento relativo che completa il ciclo.

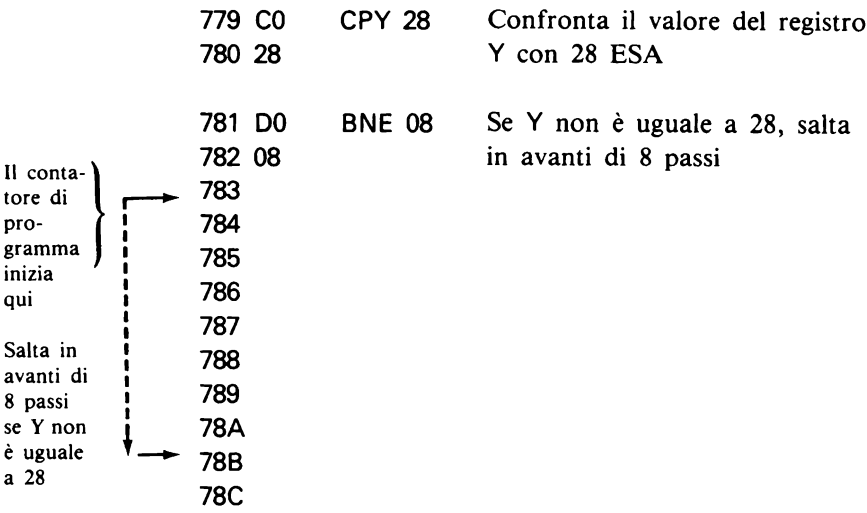


Se il risultato della precedente istruzione (CPY 28) *non* è zero, viene effettuato un salto all'inizio del ciclo. Il salto è fatto ad una locazione di memoria relativa alla posizione del contatore di programma. Il valore esadecimale F8 è equivalente al numero negativo -8 , e causerà un salto *all'indietro* di 8 passi dalla posizione corrente del contatore di programma.

Quando vengono usati come operandi in un'istruzione di salto come BNE, tutti i valori esadecimali da 01 a 7F inclusi causano un salto *in avanti* dalla posizione corrente del contatore di programma. La seguente istruzione causa un salto *in avanti* dalla locazione 783 (dove punta il contatore di programma quando viene eseguita l'istruzione BNE) alla locazione di memoria 78B ($783 + 8$).

781 D0 BNE 08
782 08

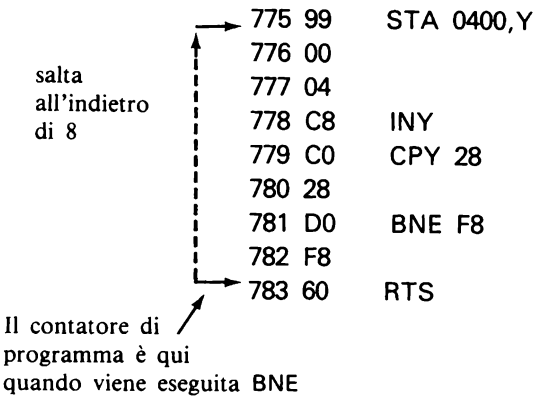
Ecco un esempio dell'uso di questa istruzione in una parte di programma.



Tutti i valori esadecimali da 80 a FF sono usati da istruzioni di salto per spostamenti all'indietro (o negativi). Nel programma Visualizza Caratteri, l'istruzione usata è:

781 D0	BNE F8	Se Y non è uguale a 28, salta all'indietro di 8 passi
782 F8		

Il salto è fatto all'indietro (o in direzione negativa) poiché F8 è compreso tra 80 e FF. Contando all'indietro 8 passi dalla locazione 783, si arriva a 775, l'inizio del ciclo.



Non analizzeremo il metodo usato dal computer per determinare i valori dei numeri negativi. Forniremo invece delle tavole per determinare l'operando usato nelle istruzioni di salto.

TAVOLA PER DETERMINARE I SALTI IN AVANTI

<i>Passi in avanti (decimale)</i>	<i>Operando di salto (ESA)</i>	<i>Passi in avanti (Decimale)</i>	<i>Operando di salto (ESA)</i>	<i>Passi in avanti (decimale)</i>	<i>Operando di salto (ESA)</i>
1	01	49	31	97	61
2	02	50	32	98	62
3	03	51	33	99	63
4	04	52	34	100	64
5	05	53	35	101	65
6	06	54	36	102	66
7	07	55	37	103	67
8	08	56	38	104	68
9	09	57	39	105	69
10	0A	58	3A	106	6A
11	0B	59	3B	107	6B
12	0C	60	3C	108	6C
13	0D	61	3D	109	6D
14	0E	62	3E	110	6E
15	0F	63	3F	111	6F
16	10	64	40	112	70
17	11	65	41	113	71
18	12	66	42	114	72
19	13	67	43	115	73
20	14	68	44	116	74
21	15	69	45	117	75
22	16	70	46	118	76
23	17	71	47	119	77
24	18	72	48	120	78
25	19	73	49	121	79
26	1A	74	4A	122	7A
27	1B	75	4B	123	7B
28	1C	76	4C	124	7C
29	1D	77	4D	125	7D
30	1E	78	4E	126	7E
31	1F	79	4F	127	7F
32	20	80	50		
33	21	81	51		
34	22	82	52		
35	23	83	53		
36	24	84	54		
37	25	85	55		
38	26	86	56		
39	27	87	57		
40	28	88	58		
41	29	89	59		
42	2A	90	5A		
43	2B	91	5B		
44	2C	92	5C		
45	2D	93	5D		
46	2E	94	5E		
47	2F	95	5F		
48	30	96	60		

Esempi che usano salti in avanti:

1. 792 D0 BNE 07

793 07

→ 794 .

. .

. .

. .

. .

→ 801 .

Il contatore di programma parte da 794

Salto desiderato a 801 (7 passi)

Cercate sulla tavola:

Passi in avanti (decimale)	Operando di salto (ESA)
7	07 ← Operando

Se la condizione controllata non è uguale a zero, salta in avanti a 801 (794+7 passi).

2. 798 D0 BNE 1F

799 1F

→ 800 .

. .

. .

. .

. .

→ 831

Il contatore di programma parte da 800

Salto desiderato a 831 (31 passi)

Cercate sulla tavola:

Passi in avanti (decimale)	Operando di salto (ESA)
31	1F ← Operando

Se la condizione controllata non è uguale a zero, salta in avanti a 831 (800+31 passi).

3. 814 D0 BNE 77

815 77

→ 816 .

. .

. .

. .

. .

→ 935 .

Il contatore di programma parte da 816

Salto desiderato a 935 (119 passi)

Cercate sulla tavola:

Passi in avanti (decimale)	Operando di salto (ESA)
119	77 ← Operando

Se la condizione controllata non è uguale a zero, salta in avanti a 935 (816+119 passi).

4. 922 D0 BNE 5B

923 5B

→ 924 .

. .

. .

. .

. .

→ 1015.

Il contatore di programma parte da 924

Salto desiderato a 1015 (91 passi)

Cercate sulla tavola:

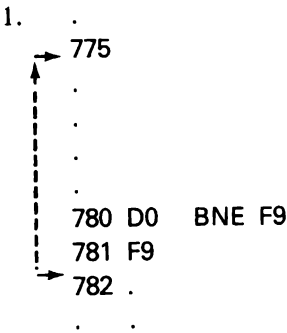
Passi in avanti (Decimale)	Operando di salto (ESA)
91	5B ← Operando

Se la condizione controllata non è uguale a zero, salta in avanti a 1015 (924+91 passi).

TAVOLA PER DETERMINARE I SALTI ALL'INDIETRO

<i>Passi all'indietro (decimale)</i>	<i>Operando di salto (ESA)</i>	<i>Passi all'indietro (decimale)</i>	<i>Operando di salto (ESA)</i>	<i>Passi all'indietro (decimale)</i>	<i>Operando di salto (ESA)</i>
1	FF	49	CF	97	9F
2	FE	50	CE	98	9E
3	FD	51	CD	99	9D
4	FC	52	CC	100	9C
5	FB	53	CB	101	9B
6	FA	54	CA	102	9A
7	F9	55	C9	103	99
8	F8	56	C8	104	98
9	F7	57	C7	105	97
10	F6	58	C6	106	96
11	F5	59	C5	107	95
12	F4	60	C4	108	94
13	F3	61	C3	109	93
14	F2	62	C2	110	92
15	F1	63	C1	111	91
16	F0	64	C0	112	90
17	EF	65	BF	113	8F
18	EE	66	BE	114	8E
19	ED	67	BD	115	8D
20	EC	68	BC	116	8C
21	EB	69	BB	117	8B
22	EA	70	BA	118	8A
23	E9	71	B9	119	89
24	E8	72	B8	120	88
25	E7	73	B7	121	87
26	E6	74	B6	122	86
27	E5	75	B5	123	85
28	E4	76	B4	124	84
29	E3	77	B3	125	83
30	E2	78	B2	126	82
31	E1	79	B1	127	81
32	E0	80	B0	128	80
33	DF	81	AF		
34	DE	82	AE		
35	DD	83	AD		
36	DC	84	AC		
37	DB	85	AB		
38	DA	86	AA		
39	D9	87	A9		
40	D8	88	A8		
41	D7	89	A7		
42	D6	90	A6		
43	D5	91	A5		
44	D4	92	A4		
45	D3	93	A3		
46	D2	94	A2		
47	D1	95	A1		
48	D0	96	A0		

Esempio di salti all'indietro:

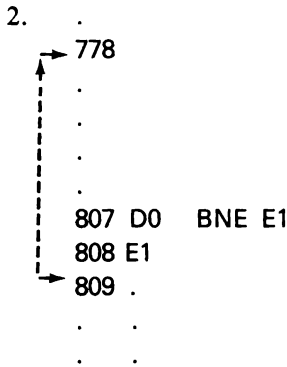


Il contatore di programma parte da 782
Salto desiderato a 775 (-7 passi)

Cercate sulla tavola:

Passi in all'indietro (decimale)	Operando di salto (ESA)
7	F9 ← Operando

Se la condizione controllata non è uguale a zero, salta indietro a 778 (809-31 passi).

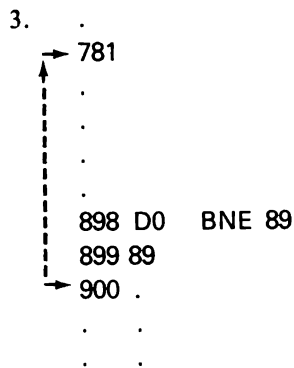


Il contatore di programma parte da 809
Salto desiderato a 778 (-31 passi)

Cercate sulla tavola:

Passi all'indietro (decimale)	Operando di salto (ESA)
31	E1 ← Operando

Se la condizione controllata non è uguale a zero, salta indietro a 778 (809-31 passi).

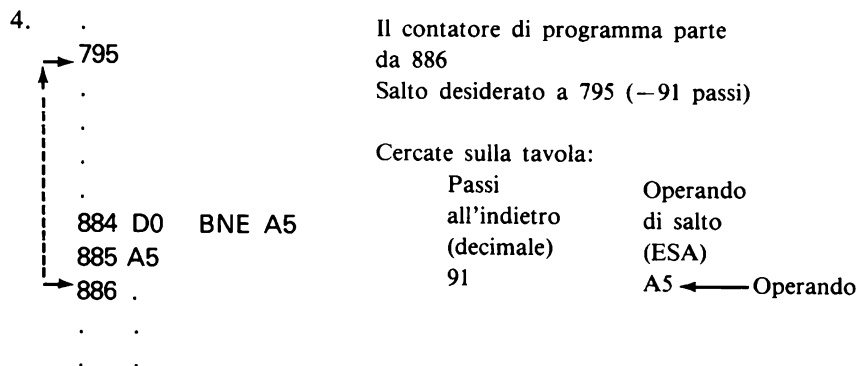


Il contatore di programma parte da 900
Salto desiderato a 781 (-119 passi)

Cercate sulla tavola:

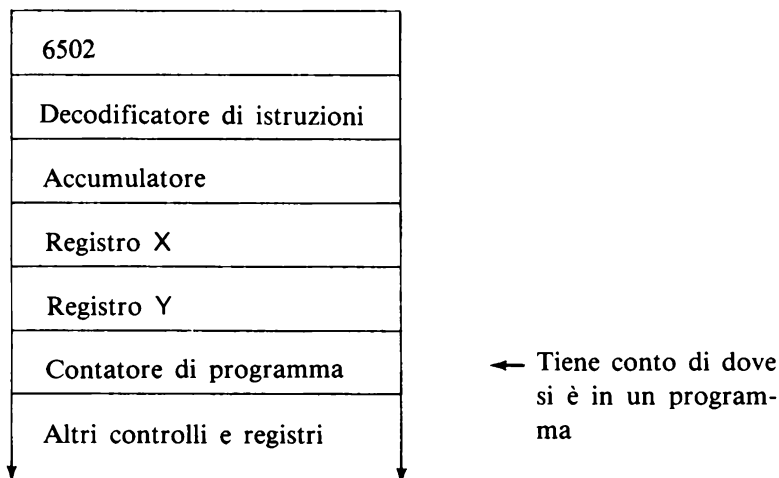
Passi all'indietro (decimale)	Operando di salto (ESA)
119	89 ← Operando

Se la condizione controllata non è uguale a zero, salta indietro a 781 (900-119 passi).



Mentre il programma è nel computer, provatelo cambiando il valore ASCII usato all'indirizzo 774. Usate i valori da C1 a DA per vedere differenti lettere dell'alfabeto. Quando avrete finito gli esperimenti, passate al prossimo programma che visualizza tutti i 26 caratteri alfabetici su una linea.

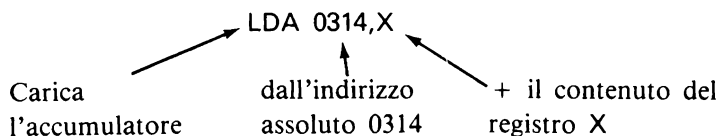
Abbiamo ora parlato di queste parti del microprocessore 6502.



UN'OCCHIATA ALL'ALFABETO

Nel precedente programma STA 0400, Y era usata per mettere una lettera in posizioni consecutive della linea più in alto del video. Il registro Y è stato usato per *indicare* la posizione.

Il registro X può essere usato nello stesso modo. Se i codici ASCII delle lettere dell'alfabeto saranno memorizzati in locazioni consecutive, potranno essere caricati nell'accumulatore con l'istruzione:



Ognuno dei due registri (X o Y) può essere usato con le istruzioni STA e LDA. Essi possono essere usati anche con altre istruzioni, come vedremo più avanti.

Dopo che un codice ASCII è stato utilizzato dal programma, il registro X può essere incrementato per fornire il codice corretto per il prossimo passaggio nel ciclo. Aggiungendo una nuova parte al precedente programma, potete prendere ogni carattere alfabetico in ordine e visualizzarlo in una nuova posizione dello schermo.

PROGRAMMA ALFABETO

1 COMMENTO **INIZIALIZZAZIONE**

```

768 20    JSR FC58    Cancella lo schermo
769 58
779 FC
  
```

```

771 A0    LDY 0
772 00
  
```

```

773 A2    LDX 0      Nuova istruzione — carica il regi-
774 00                                stro X con zero
  
```

2 COMMENTO **CICLO DI CARICAMENTO E VISUALIZZAZIONE**

Ciclo	→ 775 BD	LDA 0314,X	Nuova istruzione — carica l'accumulatore da 0314 + X
	776 14		
	777 03		
	778 99	STA 0400,Y	
	779 00		
	780 04		
	781 E8	INX	Nuova istruzione — incrementa il registro X
	782 C8	INY	
	783 C0	CPY 1A	1A ESA = 26 (numero di lettere dell'alfabeto)
	784 1A		
	785 D0	BNE F4	Salta se non uguale, indietro di 12 locazioni (dalla tavola, F4)
	786 F4		

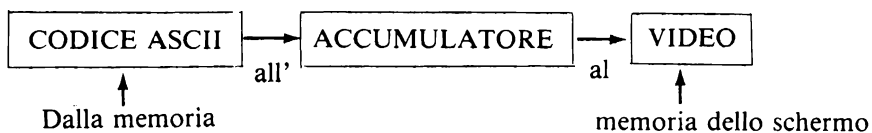
3 COMMENTO **RITORNO AL BASIC**
 787 60 RTS

4 COMMENTO **LISTA DEI DATI**

788 C1	Lettera A
789 C2	Lettera B
790 C3	Lettera C
791 C4	.
792 C5	.
793 C6	.
794 C7	.
795 C8	.
796 C9	.
797 CA	.
798 CB	.
799 CC	.
800 CD	.
801 CE	.
802 CF	.
803 D0	.
804 D1	.
805 D2	.
806 D3	.
807 D4	.
808 D5	.
809 D6	.
810 D7	.
811 D8	.
812 D9	Lettera Y
813 DA	Lettera Z

La parte 1 azzera i registri X e Y.

La parte 2 carica un carattere ASCII dalla memoria e lo visualizza sulla linea in alto.



L'istruzione LDA 0311,X è simile all'istruzione READ del Basic. Il codice ASCII è letto dai dati che stanno in memoria. La parte 2 è un ciclo che viene eseguito 26 volte (una per ogni lettera dell'alfabeto).

La parte 3 ritorna il controllo al sistema operativo dopo aver completato l'esecuzione.

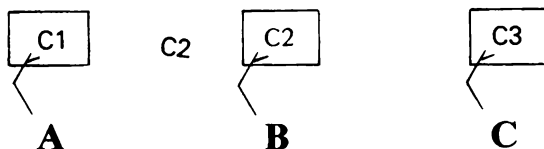
La parte 4 fornisce i dati che devono essere letti dall'istruzione LDA 0314,X. Questa parte è usata come l'istruzione DATA del Basic.

Usate il vostro sistema operativo per caricare ed eseguire il programma, che inizia alla locazione 768 ed è lungo 46 byte. Ecco come apparirà il video dopo l'esecuzione del programma.



ABCDEFGHIJKLMNOPQRSTUVWXYZ

Avete visto come visualizzare informazioni sulla linea più alta dello schermo. Nel prossimo programma, userete una subroutine interna che vi permetterà di inserire caratteri dalla tastiera. Questo successivo programma visualizzerà i codici ASCII usati per rappresentare le lettere dell'alfabeto.



VISUALIZZAZIONE DI CODICI ASCII

Questo programma usa una subroutine interna (JSR FD35) per leggere un carattere che avete premuto sulla tastiera. Mette il codice ASCII di quel carattere nell'accumulatore. Un'altra subroutine (JSR FDDA) viene usata per stampare il codice ASCII del carattere che avete inserito. Il programma vi permette di inserire da tastiera tutti i 26 caratteri dell'alfabeto, prima di ritornare al sistema operativo. Nel programma ci sono spazi e ritorni del carrello, cosicché i caratteri inseriti ed i loro codici ASCII appaiano chiaramente in colonne sulla parte sinistra dello schermo. Sono 41 byte di programma.

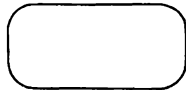
PROGRAMMA DI VISUALIZZAZIONE DI CODICI ASCII

1 COMMENTO **CANCELLA LO SCHERMO ED INIZIALIZZA**

768 A2 LDX 0 Inizializza il contatore
769 00

770 20 JSR FC58 Cancella lo schermo
771 58
772 FC

registro X = 00000000



Cancella

2 COMMENTO **RICEVE IL CARATTERE E STAMPA**

Inizio

→ 773 20 JSR FD35 Riceve il carattere
774 35
775 FD

776 8D STA 0340 Lo salva in memoria
777 40
778 03

779 20 JSR FDED Lo stampa
780 ED
781 FD

3 COMMENTO **VISUALIZZA UNO SPAZIO**

782 A9 LDA E0 Carica il codice ASCII dello spazio
783 E0

784 20 JSR FDED Visualizza lo spazio. La subroutine usa l'accumulatore per altre cose, così, se vogliamo un altro spazio, dobbiamo:
785 ED
786 FD

787 A9 LDA E0 Caricare un'altro spazio
788 E0

789 20 JSR FDED Lo visualizza
790 ED
791 FD

4 COMMENTO **STAMPA IL CODICE ASCII**

792 AD	LDA 0340	Carica l'accumulatore dalla me-
793 40		moria dove è stato salvato il ca-
794 03		rattere

795 20	JSR FDDA	Stampa il codice come due cifre
796 DA		ESA
797 FD		

5 COMMENTO **PASSA ALLA PROSSIMA LINEA**

798 A9	LDA 8D	Carica il codice ASCII per il ritor-
799 8D		no del carrello

800 20	JSR FDED	Effettua il ritorno del cursore
801 ED		
802 FD		

6 COMMENTO **RICOMINCIA SE NON HA FINITO**

803 E8	INX	Incrementa il contatore (registro X)
--------	-----	--------------------------------------

Ciclo da 773 per un nuovo tasto	804 E0	CPX 1A	Confronta il registro X con 26
	805 1A		(dec.)

806 D0	BNE DD	Se non uguale, salta all'indietro
807 DD		di 35 passi. (DD dalla tavola per
		determinare i passi all'indietro.)

7 COMMENTO **RITORNO AL BASIC**

808 60	RTS	Ritorno da questa subroutine al
		sistema operativo

La parte 1 pone il registro X (contatore del numero di tasti) uguale a zero e cancella lo schermo.

La parte 2 usa la subroutine a FD35 per leggere il carattere del tasto che viene premuto. Il carattere viene quindi salvato nella memoria 0340 per un uso successivo nella parte 4. Poi la subroutine a FDED stampa il carattere sullo schermo.

La parte 3 mette due spazi tra il carattere che è stato inserito ed il codice ASCII che sarà stampato nella parte 4. Il codice ASCII per lo spazio è E0. La parte 4 carica l'accumulatore con il carattere che era stato salvato nella memoria 0340 nella parte 2. Poi usa la subroutine a FDDA per

stampare il carattere come due cifre esadecimali (il codice ASCII) del carattere).

La parte 5 dà un ritorno del cursore (codice ASCII 8D) cosicché il prossimo carattere e codice appaiono su una nuova linea.

La parte 6 incrementa il contatore (registro X), confronta il suo valore con 26 (poiché ci sono 26 lettere nell'alfabeto), e, se questo non è uguale a 26, salta indietro ad aspettare la pressione di un nuovo tasto. Se invece è uguale a 26, passa alla parte 7.

La parte 7 ritorna il controllo al sistema operativo.

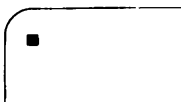
Per questo programma, suggeriamo di aggiungere ciò che segue al sistema operativo in Basic.

```
810 PRINT
820 INPUT "VUOI ESEGUIRLO ANCORA (SI O NO)?";A$
830 IF A$ = "SI" THEN 700
```

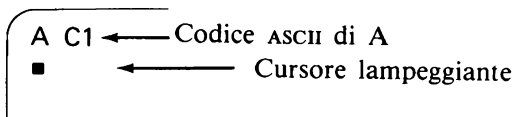
Questo vi permetterà di ripetere il programma con nuovi input dalla tastiera.

ESECUZIONE DEL PROGRAMMA

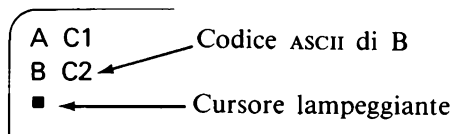
Quando il programma verrà eseguito, vedrete il cursore lampeggiante nell'angolo in alto a sinistra dello schermo. Questo significa che il computer è pronto per il vostro primo tasto.



Scrivete A



Scrivete B



Scrivete C

```

A C1
B C2
C C3
■

```

Codice ASCII di C

Cursore lampeggiante

Continuate in questa maniera finché avete visto tutti i 26 codici dei caratteri.

Y D9

Z DA

VUOI ESEGUIRLO ANCORA (SI O NO)? ■

Se siete curiosi, potete scrivere SI e tentare alcuni caratteri non alfabetici. Ricordate che l'unico limite del programma è di lasciarvi inserire 26 caratteri prima di ritornare al sistema operativo.

Potete provare i numeri da 0 a 9, i segni d'interpunzione, lo spazio, ecc. Ecco il risultato dall'inserimento delle cifre decimali.

```

0 B0
1 B1
2 B2
3 B3
4 B4
5 B5
6 B6
7 B7
8 B8
9 B9
■

```

Potete fare una tavola dei codici ASCII di tutti i tasti che avete provato. Confrontatela con la tavola dei codici ASCII nell'Appendice A.

Vi piacerebbe vedere le lettere stampate su sfondo invertito (nero su bianco)? O anche lampeggianti, da nero su bianco a bianco su nero? Il prossimo programma vi spiegherà come farlo. Proseguite quando siete pronti.

VISUALIZZAZIONE DI PIÙ LINEE

L'ultimo programma discusso, ed il prossimo, fanno uso di una delle subroutine interne del monitor dell'Apple, che visualizza un carattere (il cui codice ASCII è contenuto nell'accumulatore) sullo schermo. Poi si sposta automaticamente alla successiva posizione di stampa di quella linea. Il codice ASCII del ritorno del cursore (8D) viene usato per passare all'inizio della prossima linea, quando desiderato.

Se volete provare più volte questo programma, ricordate di includere nel sistema operativo le linee che vi permettono di cambiare i dati. Useremo:

```
810 PRINT
820 INPUT "VUOI CAMBIARE DATI (SI O NO)?";A$
830 IF A$ = "SI" THEN 420
```

Sarà utile poter cambiare i dati perché vi mostreremo come stampare bianco su nero, nero su bianco, e lettere che lampeggiano.

PROGRAMMA DI VISUALIZZAZIONE SU PIÙ LINEE

1. COMMENTO **CANCELLA LO SCHERMO ED INIZIALIZZA**

```
768 20 JSR FC58
769 58
770 FC

771 A2 LDX 0
772 00
```

2. COMMENTO **RICEVE IL CARATTERE E STAMPA**

```
773 BD LDA 0311,X
774 11
775 03

776 20 JSR FDED
777 ED Mette il carattere dell'accumulato-
778 FD re sullo schermo

779 E8 INX
```

780 E0	CPX 16	Nuova istruzione — confronta il
781 16		valore del registro X con 16

782 D0	BNE F5	Salta indietro se non uguale
783 F5		

3. COMMENTO **RITORNO AL BASIC**

784 60	RTS	RITORNO AL BASIC
--------	-----	------------------

4. COMMENTO ** LISTA DEI CODICI ASCII**

785 C1	Lettera A
786 D0	Lettera P
787 D0	Lettera P
788 CC	Lettera L
789 C5	Lettera E
790 8D	Ritorno del cursore (nuova linea)
791 C3	Lettera C
792 CF	Lettera O
793 CD	Lettera M
794 D0	Lettera P
795 D5	Lettera U
796 D4	Lettera T
797 C5	Lettera E
798 D2	Lettera R
799 8D	Ritorno del cursore (nuova linea)
800 C4	Lettera D
801 C9	Lettera I
802 D3	Lettera S
803 D0	Lettera P
804 CC	Lettera L
805 C1	Lettera A
806 D9	Lettera Y

La parte 1 azzerava il registro X e cancella lo schermo.

La parte 2 carica un codice ASCII dalla memoria (indicizzata da X) ed usa la subroutine interna per visualizzarlo sullo schermo. Questa parte è un ciclo. Ogni volta che viene percorso, il registro X è incrementato, cosicché può essere caricato dalla memoria il prossimo codice ASCII. Il valore del registro X viene confrontato con 16 esadecimale (22 decimale) poiché ci sono 22 caratteri nella lista dei dati. Se X non è uguale a 22, viene effettuato un salto all'inizio del ciclo (locazione 773). La parte 4 è la lista dei dati usati dal ciclo.

Ancora una volta, usate il sistema operativo per caricare ed eseguire il programma, che inizia alla locazione 768 ed è lungo 39 byte. Abbiamo risparmiato alcune linee usando la subroutine interna per visualizzare i dati. Quando l'esecuzione sarà terminata, il video mostrerà:

```
APPLE
COMPUTER
DISPLAY
```

```
VUOI CAMBIARE DATI (SI O NO)? ■
```

Questo programma è simile al programma Alfabeto. Tuttavia, i ritorni del carrello nella lista dei dati permettono di usare più di una linea del video.

Finora, abbiamo sempre utilizzato lettere bianche su sfondo nero. In realtà, il vostro computer visualizza lettere colorate debolmente su sfondo scuro. I libri però sono solitamente stampati con lettere nere su pagine bianche. Beh, non possiamo essere tutti perfetti. La versatilità di un computer è sorprendente. Proseguite la lettura e vedrete. Il calcolatore Apple ha la possibilità di invertire il colore di un testo, cosicché lo sfondo di una singola lettera sia bianco, e quest'ultima nera. Questo è detto *schermo invertito*. Per ottenere l'effetto d'inversione, cambiate il codice ASCII normale come nei seguenti esempi:

a. Codice ASCII della lettera A = C1

In binario: $\underbrace{1\ 1\ 0\ 0}_C\ \underbrace{0\ 0\ 0\ 1}_1$

Codice inverso della lettera A = 01

In binario: $\underbrace{0\ 0\ 0\ 0}_0\ \underbrace{0\ 0\ 0\ 1}_1$

b. Codice ASCII della lettera P = D0

In binario: $\underbrace{1\ 1\ 0\ 1}_D\ \underbrace{0\ 0\ 0\ 0}_0$

Codice inverso della lettera P = 10

In binario: $\underbrace{0\ 0\ 0\ 1}_1\ \underbrace{0\ 0\ 0\ 0}_0$

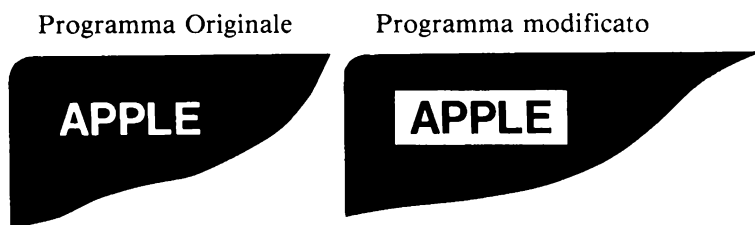
Per ottenere l'inversione del carattere, si devono togliere i due bit più significativi (quelli nelle due posizioni più a sinistra).

	da		da
1 1	0 0 0 0 0 1	C1	1 1
	a		D0
0 0 0 0 0 0 0 1	01	0 0 0 1 0 0 0 0	10

Come esempio di questa caratteristica, fate le seguenti modifiche al Programma di visualizzazione su più linee.

785 01	A invertita
786 10	P invertita
787 10	P invertita
788 0C	L invertita
789 05	E invertita

Ora, quando eseguite il programma modificato, vedrete questa differenza sul video.



Un'altra modifica al codice ASCII del carattere da visualizzare, lo fa lampeggiare. Come esempio, cambiate i codici ASCII della parola DISPLAY

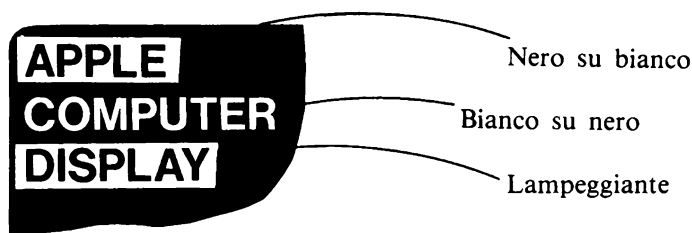
800 44	D lampeggiante
801 49	I lampeggiante
802 53	S lampeggiante
803 50	P lampeggiante
804 4C	L lampeggiante
805 41	A lampeggiante
806 59	Y lampeggiante

Osservate ancora una volta le tre possibili versioni della lettera A.

A normale	= C1 1100 0001 in binario
A invertita	= 01 0000 0001 in binario (le 2 cifre a sinistra = 0)
A lamp.	= 41 0100 0001 in binario (cifra a sinistra = 0)

Usate il sistema operativo per fare le modifiche in memoria alle locazioni da 800 a 806. Ora, quando eseguirete il programma, la parola DI-

SPLAY lampeggerà — prima invertita, poi normale, poi invertita, poi normale, ecc.

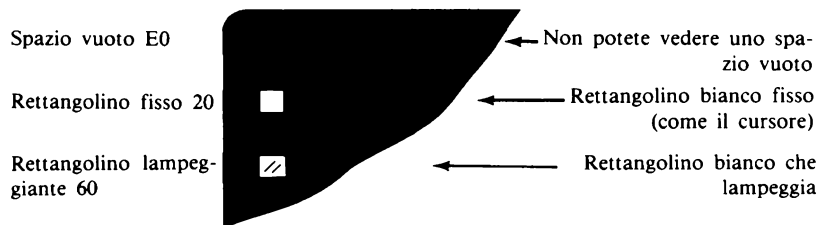


Potete cambiare il messaggio visualizzato da questo programma usando una qualsiasi lettera della seguente tavola, alle locazioni che partono da 800. Potete allungare o accorciare il messaggio cambiando il valore nella locazione 781 in accordo con il numero di caratteri usati.

TAVOLA DEI CODICI DELL'ALFABETO

<i>NORMALE</i>	<i>INVERTITA</i>	<i>LAMPEGGIANTE</i>	<i>LETTERA</i>
C1	01	41	A
C2	02	42	B
C3	03	43	C
C4	04	44	D
C5	05	45	E
C6	06	46	F
C7	07	47	G
C8	08	48	H
C9	09	49	I
CA	0A	4A	J
CB	0B	4B	K
CC	0C	4C	L
CD	0D	4D	M
CE	0E	4E	N
CF	0F	4F	O
D0	10	50	P
D1	11	51	Q
D2	12	52	R
D3	13	53	S
D4	14	54	T
D5	15	55	U
D6	16	56	V
D7	17	57	W
D8	18	58	X
D9	19	59	Y
DA	1A	5A	Z

Altri codici che potete provare sono:



Divertitevi un po' a provare questo programma prima di passare al prossimo che vi permetterà di scrivere informazioni sullo schermo man mano che le inserite dalla tastiera. Scommetto che avreste sempre voluto vedere il vostro nome lampeggiante. Questa è la vostra occasione!

L'ultimo programma di questo capitolo è uno dei più facili visti finora, ma è uno dei più potenti. Vi permette di riempire lo schermo delle frasi inserite dalla tastiera. Potete usare spazi, la punteggiatura, il tasto return, numeri, lettere, ecc. Sono state usate tre nuove istruzioni.

Dopo aver letto un tasto premuto sulla tastiera, viene usata l'istruzione *CMP* (*CoMPare*, confronta) per confrontare il valore preso dalla tastiera (il cui codice ASCII, dopo la pressione del tasto, si trova nell'accumulatore) con il valore *AF* (il codice per la barra, /). Se il tasto è stato la barra, la prossima istruzione, *BEQ* (*Branch if EQual*, salta se uguale) causerà un salto alla fine della subroutine. Perciò, quando siete stufo, inserite la barra. Questo è simile all'istruzione del Basic:

```

776 IF X = 175 THEN GOTO 784
              ↖ (AF ESA = 175 decimale)
784 RETURN
              oppure
776 IF X$ = "/" THEN GOTO 784
784 RETURN

```

Un'istruzione *JPM* (salto incondizionato) segue la subroutine di stampa. *JMP* fa tornare indietro il calcolatore per un altro carattere, ed è simile all'istruzione Basic:

```
781 GOTO 771
```

PROGRAMMA PER LA VISUALIZZAZIONE DI UN MESSAGGIO QUALSIASI

1 COMMENTO **CANCELLA LO SCHERMO**

```

768 20 JSR FC58
769 58
770 FC

```

2 COMMENTO **CICLO CHE RICEVE E STAMPA IL CARATTERE**

<pre> 771 20 JSR FD35 772 35 773 FD 774 C9 CMP AF 775 AF 776 F0 BEQ 06 777 06 778 20 JSR FDED 779 ED 780 FD 781 4C JMP 0303 782 03 783 03 </pre>	<pre> Riceve un carattere Confronta l'accumulatore con AF Se uguale, salta 6 passi; uscita dal ciclo Stampa il carattere Torna indietro per un altro carat- tere (0303 ESA = 771 decimale) </pre>
--	---

3 COMMENTO **RITORNO AL BASIC**

```

784 60 RTS          Ritorno al sistema operativo

```

Nella parte 2 del programma abbiamo usato tre nuove istruzioni. L'istruzione **CMP** è simile a **CPX** e **CPY**. Viene usato il modo immediato per confrontare il valore dell'accumulatore con quello che segue l'istruzione. L'istruzione **BEQ** che segue **CMP** causa un salto se i due valori sono uguali. Se il salto è effettuato, si passa all'istruzione **RTS**, sei passi più avanti (rispetto all'attuale posizione del contatore, 778). Se non viene fatto il salto, l'istruzione **JSR FDED** stampa il carattere e sposta il cursore in avanti di un posto.

Alla fine della parte 2 abbiamo la terza istruzione nuova **JMP**. Questo è un salto incondizionato. Il programma, se questa istruzione è eseguita, effettua sempre il salto. Essa lavora come l'istruzione **GOTO** del Basic. Il programma consta di soli 17 byte, ma vi permette di scrivere tutto il giorno sullo schermo. Quando questo è pieno, si sposta semplicemente in su per far posto per altri caratteri. Quando volete fermare il programma, inserite il simbolo della barra. Questo vi farà uscire dal ciclo e ritornare al sistema operativo. Naturalmente, se volete usare la barra mentre

state scrivendo, siete sfortunato. Questo programma usa la barra per terminare le sue operazioni.

Non possiamo mostrarvi come appare lo schermo mentre state dando dei comandi. Ecco visualizzerà ogni cosa voi scriviate.

SOMMARIO

In questo capitolo avete trovato delle altre utili subroutine interne per visualizzare caratteri e per leggerli dalla tastiera. Quindi siete ora in grado di usare programmi in linguaggio macchina per visualizzare sia grafici che testi. Avete imparato quali sono le locazioni assegnate al video, e ad usare i codici ASCII.

Avete utilizzato tutte le istruzioni introdotte nei precedenti capitoli, come pure parecchie altre nuove. Ecco un sommario delle istruzioni introdotte in questo capitolo:

1. **LDX** (*LoaD X register*, carica il registro X), usata nel modo immediato come un contatore, esattamente come LDY è stata usata con il registro Y.
Esempio A2 codice operativo
 00 dato caricato nel registro X
2. **INY** (*INcrement register Y*, incrementa il registro Y), usata per incrementare di uno il valore contenuto nel registro Y. Modo implicito.
Esempio C8 codice operativo
3. **INX** (*INcrement register X*, incrementa il registro X), usata per incrementare di uno il contenuto del registro X. Modo implicito.
Esempio E8 codice operativo
4. **CPY** (*ComPare Y register*, confronta il registro Y), usata per confrontare il contenuto del registro Y con il numero dato. È stata usata in modo immediato.
Esempio C0 codice operativo
 1A dato con cui viene confrontato il valore contenuto nel registro Y.
5. **CPX** (*ComPare X register*, confronta il registro X), usata per confrontare il valore contenuto nel registro X con il numero dato. È stata usata in modo immediato.
Esempio E0 codice operativo
 06 dato con cui viene confrontato il valore contenuto nel registro X
6. **BNE** (*Branch if Not Equal*, salta se non uguale), usata dopo un'istruzione di confronto per dire dove si trova la prossima istru-

zione da eseguire se il risultato del confronto non è uguale a zero. È un'istruzione in modo relativo includente un dato che dice al computer dove posizionare il contatore di programma relativamente alla presente posizione.

Esempio D0 codice operativo

F5 dato che dice di quanto spostare il contatore di programma (−11 in questo esempio)

7. STA (*STore Accumulator*, memorizza l'accumulatore), usata nel modo assoluto indicizzato. Memorizza il valore dell'accumulatore nella memoria assegnata dalla locazione assoluta data più il valore contenuto nel registro Y.

Esempio 99 codice operativo

00 byte meno significativo della locazione assoluta

04 byte più significativo della locazione assoluta

8. LDA (*LoaD Accumulator*, carica l'accumulatore), usata nel modo assoluto indicizzato questa volta. Carica l'accumulatore con il dato contenuto nella locazione di memoria data dall'indirizzo assoluto più il valore contenuto nel registro X.

Esempio BD codice operativo

14 byte meno significativo della locazione assoluta

03 byte più significativo della locazione assoluta

9. CMP (*CoMPare accumulator*, confronta l'accumulatore), usata nel modo immediato per confrontare il valore contenuto nell'accumulatore con un dato numero.

Esempio C0 codice operativo

AF dato con cui viene confrontato il valore contenuto nell'accumulatore

10. BEQ (*Branch if EQual*, salta se uguale), usata in modo relativo dopo un'istruzione di confronto. Causa un salto se i valori confrontati sono uguali. Il dato che segue l'istruzione dice di quanto, e in che direzione, saltare.

Esempio F0 codice operativo

06 dato che dice dove saltare (in questo esempio, 6 locazioni in avanti)

11. JMP (*JuMP*, salta), usata in modo assoluto per causare un salto incondizionato alla locazione di memoria specificata.

Esempio 4C codice operativo

03 byte meno significativo dell'indirizzo della destinazione

03 byte più significativo dell'indirizzo della destinazione

TAVOLA DELLE SUBROUTINE FINORA USATE

Funzione	Locazione di memoria
Cancella lo schermo	FC58
Abilita il modo grafico	FB40
Disegna un punto	F800
Disegna una linea orizzontale	F819
Disegna una linea verticale	F828
Riceve un carattere	FD35
Stampa il carattere nell'accumulatore	FD35
Stampa il valore nell'accumulatore come due cifre esa	FDDA

TAVOLA DELLE ISTRUZIONI DEL LINGUAGGIO
MACCHINA FINORA USATE

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
ASL	Accumulatore	0A	1	Sposta i bit a sinistra
BEQ	Relativo	F0	2	Salta se uguale
BNE	Relativo	D0	2	Salta se non = 0
CMP	Immediato	C9	2	Confronta l'accumulatore
CPX	Immediato	E0	2	Confronta il reg. X con il valore
CPY	Immediato	C0	2	Confronta il reg. Y con il valore
INX	Implicito	E8	1	Incrementa il registro X
INY	Implicito	C8	1	Incrementa il registro Y
JMP	Assoluto	4C	3	Salta
JSR	Assoluto	20	3	Salta alla subroutine
LDA	Immediato	A9	2	Carica l'accumulatore
LDA	Assoluto indicizzato	BD	3	Carica l'accumulatore
LDX	Immediato	A2	2	Carica il registro X
LDY	Immediato	A0	2	Carica il registro Y
RTS	Implicito	60	1	Ritorno alla subroutine
STA	Pagina zero	85	2	Memorizza l'accumulatore
STA	Assoluto	8D	3	Memorizza l'accumulatore
STA	Assoluto indicizzato	99	3	Memorizza l'accumulatore

Avete fatto molte cose in questo capitolo. Ora potete visualizzare sia grafici che testi sullo schermo. Nel prossimo capitolo vedremo se è possibile animare l'Apple con dei suoni.

ESERCIZI

1. L'istruzione in modo assoluto STA 0400 memorizza il valore contenuto nell'accumulatore nella locazione di memoria 0400. Descrivete che cosa fa l'istruzione in modo assoluto indicizzato STA 0400,Y. ____

2. Quanti caratteri si possono visualizzare su una linea del video dell'Apple? _____ (valore decimale)

3. Se il valore del registro Y è attualmente 26 (esadecimale), e viene eseguita la seguente parte di un programma in linguaggio macchina, che istruzione sarà eseguita dopo la BNE?

778 C8	INY
779 C0	CPY 28
780 28	
781 D0	BNE FB
782 FB	
783 60	RTS

4. Se il valore in Y è 27 (esadecimale) e viene eseguita ancora la parte di programma mostrata nell'Esercizio 3, che istruzione sarà eseguita dopo la BNE? _____

5. Se il codice ASCII della lettera A è C1, qual è il codice ASCII della lettera G? _____

6. Supponiamo che l'accumulatore contenga il codice ASCII della lettera A. Se venisse eseguita la subroutine interna a FDED, sarebbe visualizzata una A sullo schermo. Se invece fosse eseguita la subroutine interna a FDDA, con lo stesso valore nell'accumulatore, cosa apparirebbe sullo schermo?

7. Il codice ASCII per la visualizzazione normale, bianco su nero, della lettera C è C3 esadecimale (11000011 binario). Il codice per la lettera C invertita (nero su bianco) sarà:
 - a. _____ ESA oppure _____ binario.

Il codice per una C lampeggiante sarà:

b. _____ ESA oppure _____ binario.

8. Spiegare che cosa succederà quando saranno eseguite le seguenti istruzioni.

a. E0 CPX 06
06

b. C0 CPY 1A
1A

c. C9 CMP AF
AF

RISPOSTE AGLI ESERCIZI

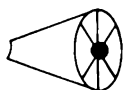
1. Memorizza il valore contenuto nell'accumulatore nella locazione 0400+il contenuto del registro Y. (Se $Y = 5$, la locazione sarà la $0400 + 5 = 0405$).
2. 40
3. 778 C8 INY (FB = -5 contando da 783)
4. 783 60 RTS ($Y = 27 + 1 = 28$. Perciò la BNE non viene eseguita)
5. C7
6. C1 (il codice ASCII di A)
7. a. 03 ESA = 00000011 binario
b. 43 ESA o 01000011 binario
8. a. Il valore del registro X viene confrontato con 06.
b. Il valore del registro Y viene confrontato con 1A ESA.
c. Il valore contenuto nell'accumulatore viene confrontato con AF ESA.

Il suono dell'Apple

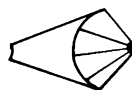
SOB

Nel Cap. 4 avete imparato come disegnare punti e linee. In questo capitolo esploreremo una caratteristica dell'Apple che stimola un altro dei vostri sensi.

All'interno dell'Apple c'è un altoparlante che potete “pizzicare” o “strimpellare” in accordo col vostro stato d'animo. Un programma può controllare l'altoparlante, cosicché possa emettere vari suoni. Il cono di carta dell'altoparlante può essere in due posizioni, in dentro o in fuori.



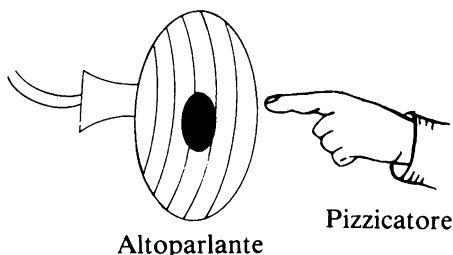
Cono in dentro



Cono in fuori

Vista esagerata del cono dell'altoparlante

Ogni volta che un programma fa riferimento alla locazione di memoria C030 (esadecimale), l'altoparlante cambia da dentro a fuori o viceversa. Questo cambiamento gli fa emettere un clic udibile. Facendo riferimento frequentemente all'indirizzo C030 (chiameremo questa operazione “pizzicare” l'altoparlante) viene prodotta una nota. Se pizzicate a differenti frequenze, saranno prodotti toni differenti. Potete anche controllare la durata della pizzicata per produrre note di varia lunghezza.



Con un po' di pazienza, sarete in grado di produrre una scala musicale che potrete usare per suonare una melodia. Se lo vorrete, potrete disegnare dei grafici sullo schermo ed accompagnarli con musica di vostra creazione.

Ma prima, diamo un'occhiata alle capacità acustiche dell'Apple con un semplice programma che produce una nota. Viene fatta un'aggiunta al sistema operativo per permettervi di scegliere la tonalità del suono che sarà emesso. Dopo la scelta del tono, viene richiamato il programma in linguaggio macchina che suona la nota. Poi il controllo è ritornato al sistema operativo, cosicché possiate scegliere una nuova nota. Questo processo continuerà finché non deciderete di smettere.

PROGRAMMA SPERIMENTALE CON LE NOTE

1 COMMENTO **INIZIALIZZA E CANCELLA LO SCHERMO**

768 A9	LDA C8	Fissa la durata della nota
769 C8		
770 85	STA 0001	La memorizza in una speciale loca-
771 01		zione
772 20	JSR FC58	Cancella lo schermo
773 58		
774 FC		

2 COMMENTO **PIZZICA L'ALTOPARLANTE**

* 775 AD	LDA C030	Carica l'accumulatore dalla memoria
776 30		C030; questo riferimento a C030
777 C0		pizzica l'altoparlante
* 778 88	DEY	Decrementa il registro Y

779 D0	BNE 04	Salta di +4 (se non uguale a zero a 785)
780 04		
* 781 C6	DEC 0001	Decrementa la locazione di memoria 00001
782 01		
783 F0	BEQ 08	Salta di +8 (se uguale a zero) a 793
784 08		
* 785 CA	DEX C8	Decrementa il registro X
786 D0	BNE F6	Salta di -10 (se non uguale a zero) a 778
787 F6		
* 788 A6	LDX 0000	Carica il registro X dalla memoria 0000
789 00		
790 4C	JMP 0307	Salta indietro a 0307 ESA (775 decimale)
791 07		
792 03		

3 COMMENTO **RITORNO AL BASIC**

793 60 RTS Ritorna al sistema operativo

*Nuove istruzioni o subroutine.

MODIFICHE AL SISTEMA OPERATIVO IN BASIC

Il programma sperimentale con le note richiede l'aggiunta di alcune linee al sistema operativo (Cap. 2) per scegliere la tonalità delle note da suonare. Includeremo anche un'istruzione che ci permetterà di decidere quando fermare il programma. Le linee da aggiungere sono:

```

720 INPUT "NOTA(1-255 O 0 PER FINIRE)?";P
730 IF P = 0 THEN 900
740 POKE 0,P
810 GOTO 720

```

↗
Inserite 1-255 per l'intensità,
oppure uno zero (0) per smettere

Queste linee devono essere inserite nella parte del sistema operativo che esegue i programmi in linguaggio macchina, come mostrato. Notate che le linee 720, 730 e 740 sono inserite *prima* della chiamata alla subroutine S, alla linea 800 del sistema operativo. La linea 810 verrà eseguita dopo il ritorno dalla subroutine cosicché possa essere scelta un'altra tonalità.

	700 REM * ESEGUE IL PROGRAMMA IN LINGUAGGIO MACCHINA*
	710 PRINT "PREMI UN TASTO QUALSIASI PER PARTIRE": GET A\$
aggiunte	720 INPUT "NOTA (1-255 O 0 PER FINIRE)?" ; P
	730 IF P = 0 THEN 900
	740 POKE 0,P
	.
	.
	.
	800 CALL S
aggiunte	810 GOTO 720
	.
	.
	.
	900 END

La linea 720 permette di inserire la nota desiderata. Il valore che inserite deve essere compreso tra 1 e 255, come indicato dall'avviso nell'istruzione di input 720. Se inserirete uno zero (0), la linea 730 causerà un salto a 900, dove il programma si fermerà.

DESCRIZIONE DEL PROGRAMMA

La parte 1 carica il valore esadecimale C8 (200 decimale) nella locazione 0001 usando l'istruzione di memorizzazione in pagina zero. Questo valore è usato per controllare la durata delle note. In questa parte viene anche cancellato lo schermo, per evitare distrazioni visive.

La parte 2 è un ciclo che pizzica l'altoparlante. Esso usa i registri X e Y per controllare il numero di volte che verrà pizzicato l'altoparlante nel periodo in cui dura la nota.

In questa parte appaiono parecchie nuove istruzioni.

a 775	AD LDA C030	Carica l'accumulatore dalla loca-
	30	zione di memoria C030 causando
	C0	una pizzicata dell'altoparlante.
		Un'istruzione in modo assoluto.



- a 778 88 DEY Un'istruzione in modo implicito che decrementa il valore nel registro Y di 1.

$Y = Y - 1$ in Basic

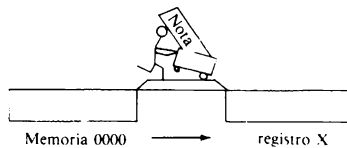
- a 781 C6 DEC 0001 Un'istruzione di pagina zero che decrementa il valore della locazione di memoria 0001 di 1.

$MEM = MEM - 1$ in Basic

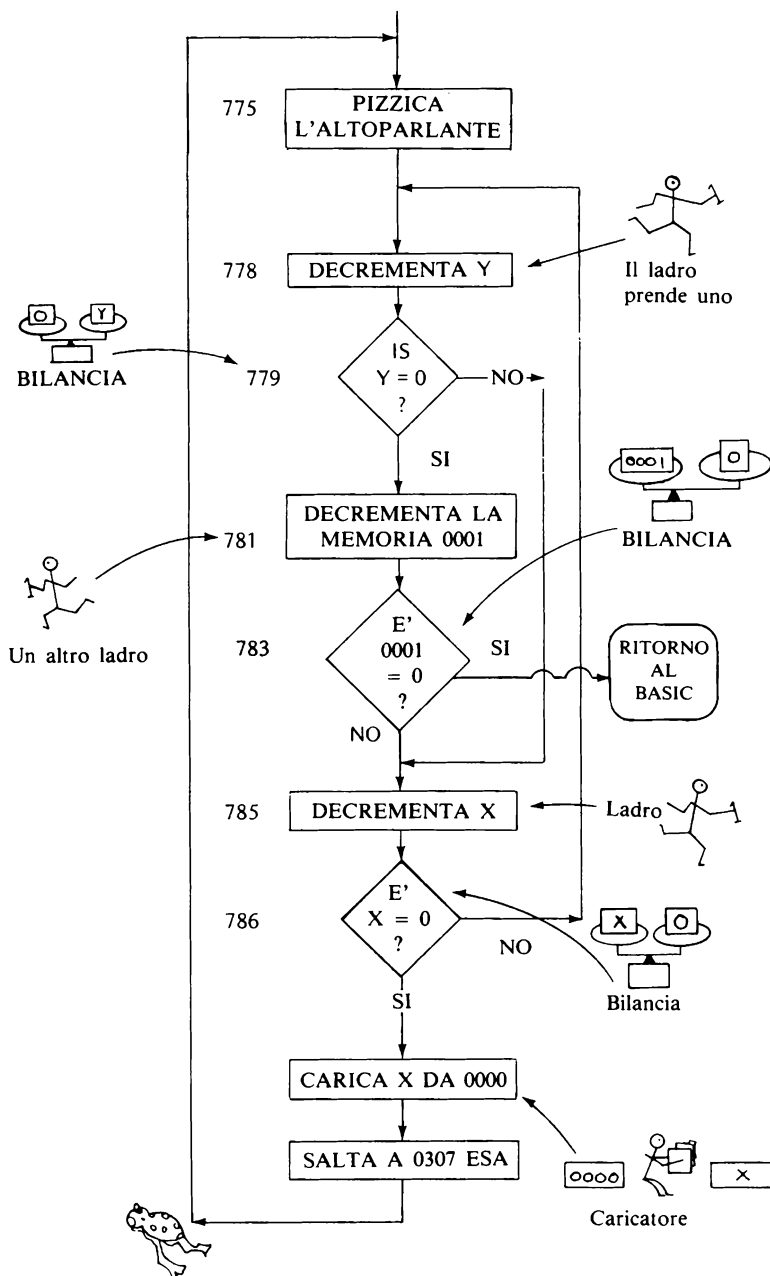
- 785 CA DEX Un'istruzione in modo implicito che decrementa di 1 il contenuto del registro X.

$X = X - 1$ in Basic

- a 788 A6 LDX 0000 Un'istruzione di pagina zero che carica il registro X dalla locazione di memoria 0000 (intensità).

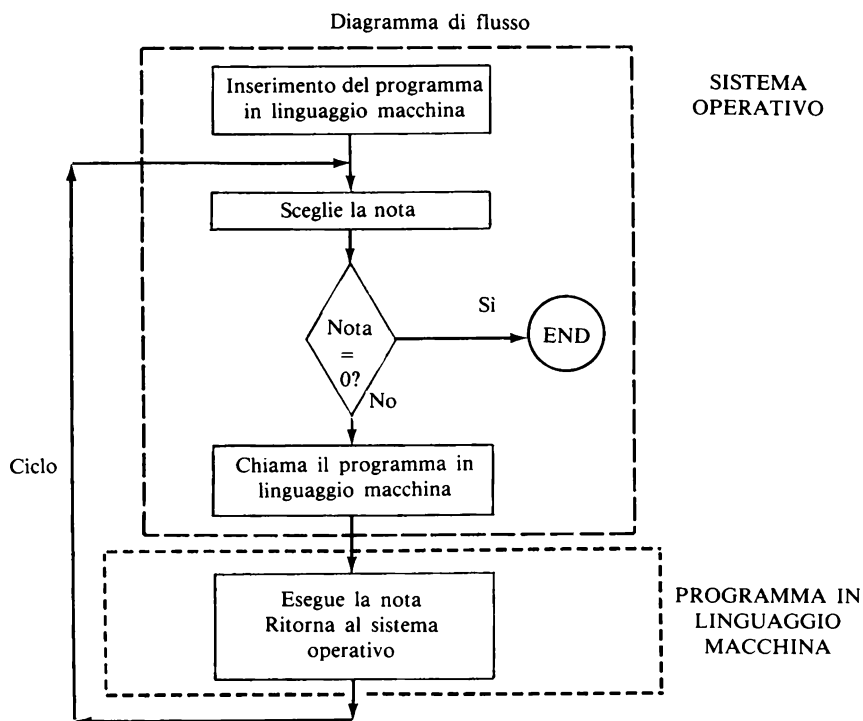


La parte 2 consiste di parecchi valori usati come contatori per determinare con quale frequenza, e quanto a lungo, deve essere pizzicato l'altoparlante. Se questa parte non vi è del tutto chiara, non preoccupatevi — funziona. Ecco un diagramma di flusso che mostra le operazioni della parte 2.



La parte 3 ritorna il controllo al sistema operativo permettendovi di scegliere una nuova nota.

Così abbiamo creato un ciclo dal sistema operativo al programma in linguaggio macchina e di nuovo al sistema operativo.



Studiando il diagramma di flusso, potete rendervi conto dell'utilità di un programma in linguaggio macchina usato come subroutine di un programma in Basic. Sebbene per voi possa essere più facile programmare in Basic, può capitare che certe parti di un programma richiedano un'esecuzione molto veloce. Il computer può eseguire un programma in linguaggio macchina molto più in fretta che in Basic (poiché il Basic deve essere interpretato). Perciò una subroutine in linguaggio macchina può essere usata per eseguire quella parte di programma che deve essere fatta velocemente. Poi si ritorna al Basic per eseguire il resto del programma, che non richiede una tale velocità.

ESECUZIONE DEL PROGRAMMA

Dopo aver modificato il sistema operativo aggiungendo le linee 720, 730, 740 e 810 come suggerito, inserite il programma in linguaggio macchina. Esso inizia alla locazione 768 ed è lungo 26 byte. Quando sarà stato inserito e controllato, vedrete il solito messaggio:

```

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■

```

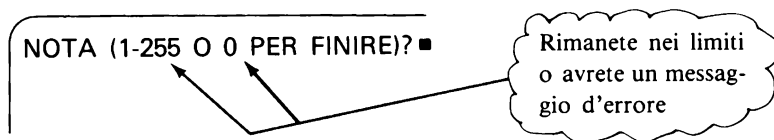
Dopo aver inserito 99 richiederà la nota

```

SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?99
NOTA (1-255 O 0 PER FINIRE)?■

```

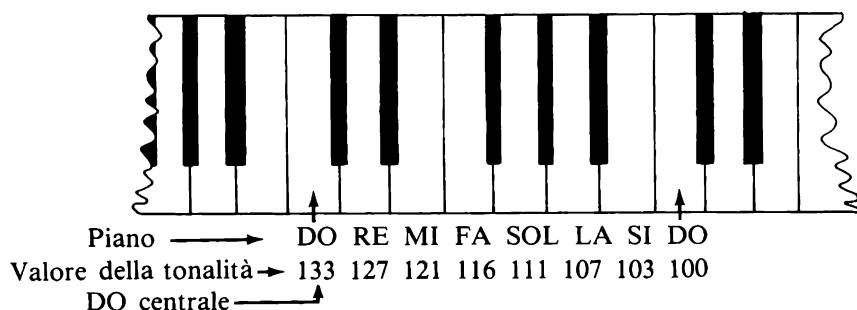
Inserite un valore che stia nell'intervallo richiesto (1-255) e premete il tasto RETURN. Lo schermo si azzererà, verrà suonata la nota, e vi sarà richiesta un'altra nota.



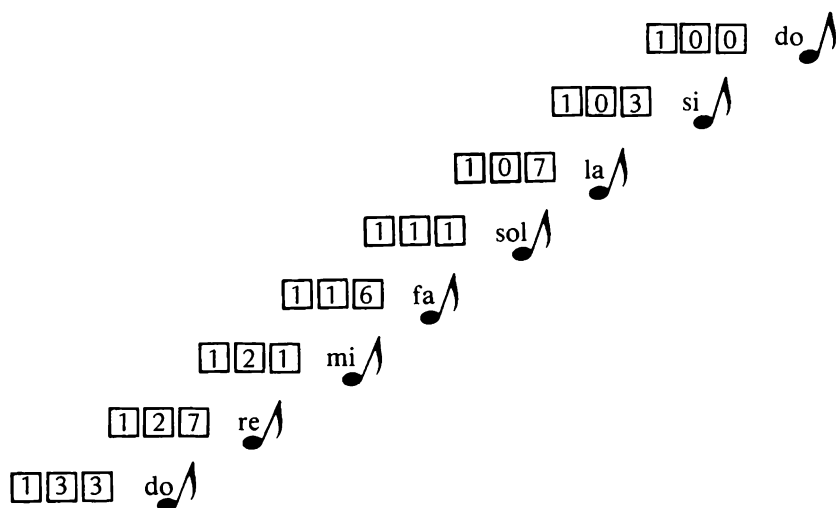
Provate l'intervallo completo della note, da molto basso (255) a molto alto (1). Non usate 0 a meno che non vogliate smettere l'esperimento. Probabilmente vorrete vedere se riuscite a costruire una scala musicale. Questo è molto facile se avete un pianoforte o un altro strumento a portata di mano. Se non avete molto orecchio per la musica, non preoccupatevi. Provate comunque. Non diciamo di essere degli esperti in campo musicale, ma non possiamo resistere alla tentazione di costruire una scala musicale.

Il nostro piano è in soggiorno ed il nostro Apple nello studio. Perciò abbiamo usato un registratore per avere le note di un'ottava del piano. Poi abbiamo portato il registratore nello studio ed abbiamo riascoltato il pezzo per costruire la scala.

CONVERSIONE DA PIANO A COMPUTER



Provate ad inserire questi valori in veloce successione per sentire come vi sembrano. Useremo questi valori (ed altri) nei futuri programmi. Se alcune note non vi suonano bene, fate le modifiche necessarie ai valori interessati. Poi suonate la scala in su e in giù. Fate parecchia pratica in modo da essere pronti per la musica che ora faremo.

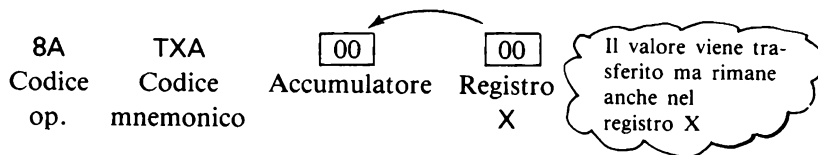


Dopo aver fatto pratica sulla scala, potete rilassarvi con il prossimo programma, che suona la scala per voi. Viene usato un ciclo per suonare tutte le otto note di un'ottava. Poi il controllo viene restituito al sistema operativo.

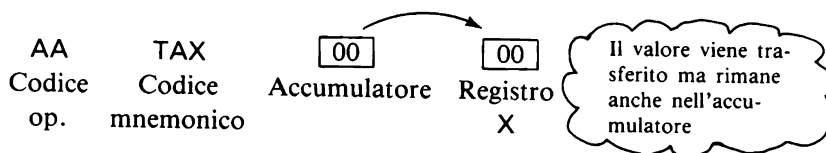
Vengono introdotte parecchie nuove istruzioni utili.

Due di queste nuove istruzioni sono usate per trasferire i dati avanti e indietro tra il registro X e l'accumulatore.

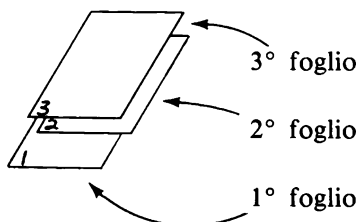
TXA è usata per trasferire il valore contenuto nel registro X nell'accumulatore. È un'istruzione di un byte in modo implicito. Il suo codice operativo è 8A.



TAX è usata per trasferire dati in direzione opposta, dall'accumulatore al registro X. Anche essa è un'istruzione di un byte. Il suo codice operativo è AA.



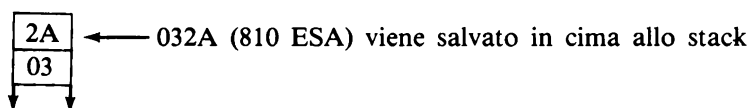
Le altre due istruzioni implicano l'uso di una parte speciale della memoria chiamata *stack*. Potete pensare allo stack come una pila di fogli di carta. Voi aumentate la pila aggiungendo un foglio in cima, e, allo stesso tempo, la diminuite togliendone uno, sempre dalla cima.



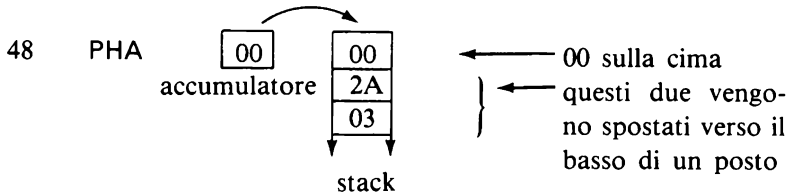
Nel disegno qui sopra, il terzo foglio deve essere rimosso prima del secondo. Il secondo va rimosso prima del primo.

Per mettere un valore sullo stack, viene usata l'istruzione PHA (*Push Accumulator on stack*, mette l'accumulatore sullo stack). Il suo codice operativo è 48.

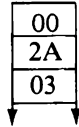
L'istruzione CALL del sistema operativo fa sì che, automaticamente, l'indirizzo corretto per il ritorno dalla subroutine in linguaggio macchina venga messo sullo stack.



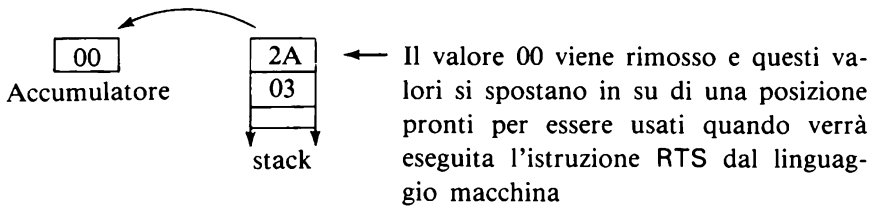
Se l'accumulatore contiene il valore zero e viene eseguita l'istruzione PHA, il valore zero sarà messo in cima allo stack, mentre gli altri due valori saranno spostati di un posto verso il basso.



Per togliere un valore dallo stack, viene usata l'istruzione PLA (*PuLL off stack to Accumulator*, toglie dallo stack e mette nell'accumulatore). Il suo codice operativo è 68. Supponiamo che lo stack contenga i valori:



Poi viene eseguita l'istruzione PLA:



PROGRAMMA SCALA AUTOMATICA

1 COMMENTO **CANCELLA LO SCHERMO ED INIZIALIZZA**

```

768 20 JSR FC58      Cancella lo schermo
769 58
770 FC

771 A9 LDA C8        Fissa la durata
772 C8

773 85 STA 0001
774 01

```

775 A2 LDX 00 Azzera il contatore
776 00

La prima parte del programma cancella lo schermo.



Schermo cancellato

La durata di C8 (190 decimale) viene caricata nell'accumulatore e da qui memorizzata nella locazione 0001. Sarà usata nel ciclo suona-note della parte 3.

0001

C8

Il registro X, che viene usato come indice nell'istruzione LDA nella seconda parte, è azzerato.

X

00

Tutte le istruzioni usate in questa parte vi sono molto familiari.

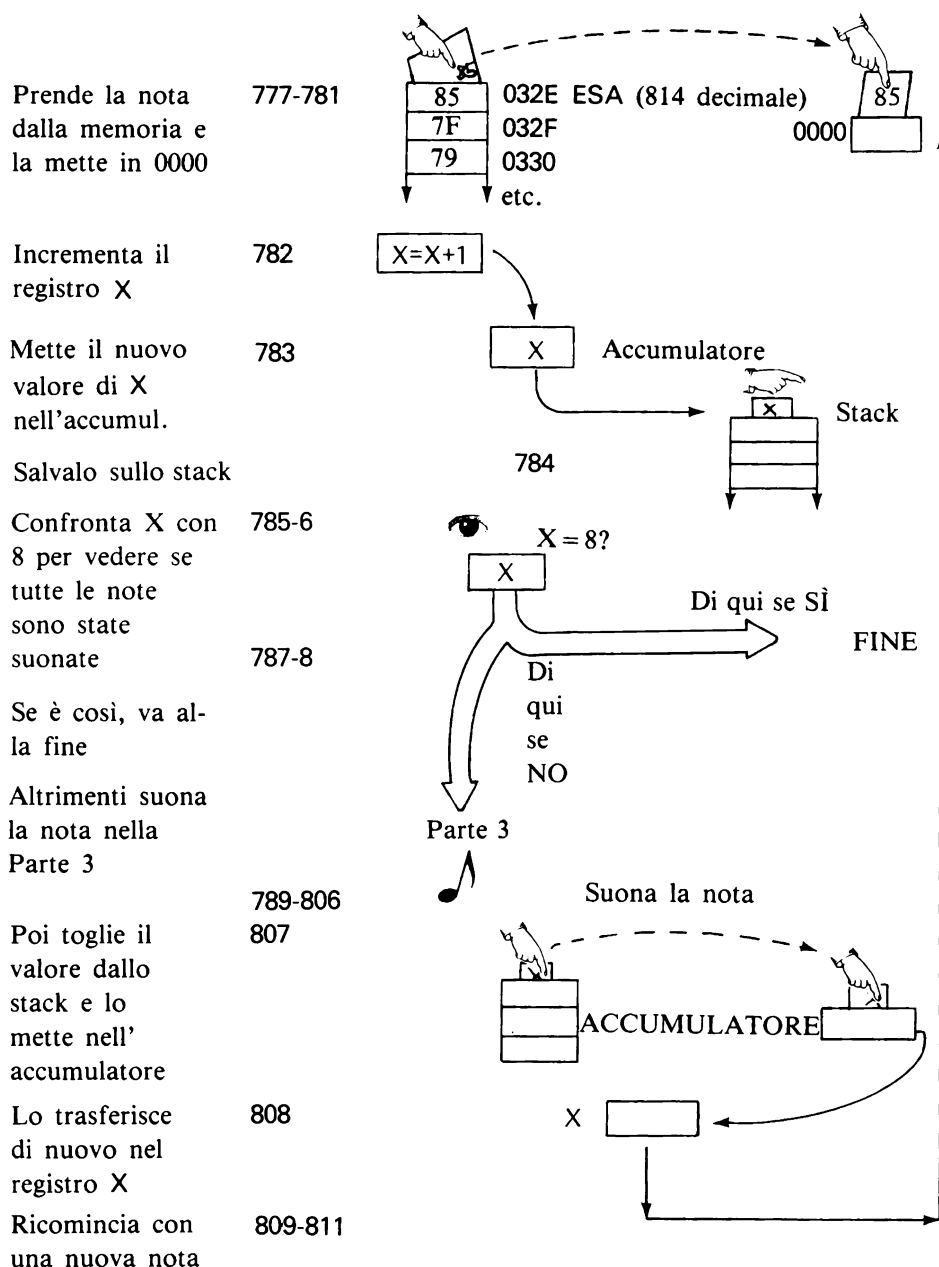
2 COMMENTO **CARICA LA NOTA**

777 BD	LDA 032E,X	Caricamento assoluto indicizzato, il registro X è usato come indice
778 2E		
779 03		
780 85	STA 0000	Mette nella memoria 0000
781 00		
782 E8	INX	Si prepara per la prossima volta
*783 8A	TXA	Trasferisce il contenuto del registro X nell'accumulatore.)
*784 48	PHA	La salva sullo stack
785 E0	CPX 08	Controlla se tutte le note sono state suonate
786 08		
787 F0	BEQ 17	(se è così) salta alla fine
788 17		

3 COMMENTO **SUONA LA NOTA**

Ciclo nuova nota	Ciclo nota	789 AD	LDA C030	Pizzica l'altoparlante
		790 30		
		791 C0		Lo stesso ciclo dell'ultimo programma. Questo ciclo è nidificato all'interno del ciclo che prende una nuova nota.
		792 88	DEY	
		793 D0	BNE 04	
		794 04		
		795 C6	DEC 0001	
		796 01		
		797 F0	BEQ 08	
		798 08		
		799 CA	DEX	
		800 D0	BNE F6	
		801 F6		
		802 A6	LDX 0000	
		803 00		
		804 4C	JMP 0315	
		805 15		
		806 03		
		*807 68	PLA	Toglie il valore salvato dallo stack
		*808 AA	TAX	Lo trasferisce nel registro X
		809 4C	JMP 0309	Ricomincia con una nuova nota
		810 09		
		811 03		

COME LAVORANO LE PARTI 2 E 3



Quando sarà stata suonata l'ultima nota, verrà effettuato un salto alla parte 4. Il vecchio valore di X verrà tolto dallo stack cosicché il corretto

indirizzo di ritorno sia disponibile in cima allo stack. Se non togliessimo il vecchio valore del registro X dallo stack, il computer penserebbe che *quel numero* fa parte dell'indirizzo a cui deve ritornare.

L'istruzione RTS a 813 riporta il calcolatore al sistema operativo.

La parte 5 contiene i dati usati per le note.

4. COMMENTO **RITORNO AL BASIC**

812 68 PLA Toglie il valore dallo stack

813 60 RTS

5. COMMENTO **LISTA DEI DATI**

814 85

815 7F

Scala musicale in ESA

816 79

817 74

818 6F

819 6B

820 67

821 64

Proprio come in Basic, una lista di dati non viene eseguita. La lista fornisce i valori utilizzati nella parte 2 dall'istruzione LDA alla locazione 777.

Inserite il programma come al solito, con il sistema operativo. Il programma inizia alla locazione di memoria 768 ed è lungo 54 byte. Controllate se ci sono errori. Quando tutto è pronto, eseguitelo. Sebbene non vediate nulla sullo schermo, verrà suonata un'ottava di note, dalla nota più bassa alla più alta. Poi il computer ritornerà ubbidiente al sistema operativo. Se volete ripetere la scala scrivete:

GOTO 700 e premete il tasto RETURN.

Il video mostrerà:

]**GOTO 700**
PREMI UN TASTO QUALSIASI PER PARTIRE■

La linea 700 del sistema operativo esegue il programma in linguaggio macchina

Quando premete un tasto, l'ottava sarà ripetuta.

Potrete aggiungere altri dati alla parte 5 in modo che le note siano suonate prima in su e poi in giù lungo la scala. Dovreste anche cambiare il valore a 786 nel programma per dare il numero totale di note nella nuova lista di dati. Questo chiude il capitolo. Nel prossimo, assieme ai suoni faremo dei grafici, ed impareremo a suonare alcune note direttamente dalla tastiera.

SOMMARIO

In questo capitolo, alle nostre capacità di programmazione, sono state aggiunte sette nuove istruzioni del linguaggio macchina ed una nuova subroutine interna. L'uso dei suoni ha aggiunto una nuova dimensione alle vostre possibilità di programmatori. Ora potete usare l'altoparlante assieme alla tastiera e al video.

Avete imparato a trasferire dati tra l'accumulatore ed il registro X e viceversa. Avete anche appreso ad usare lo stack per salvare informazioni da usare successivamente.

Ecco un sommario della nuova subroutine e delle nuove istruzioni introdotte in questo capitolo.

Nuova subroutine

1. LDA C030 è stata usata per pizzicare l'altoparlante. Quando questo viene fatto rapidamente, l'altoparlante produce un tono. L'intensità del tono, così come la sua durata, possono essere controllati dal programma.

Esempio	AD	codice op. per <i>Load Accumulator</i> (carica l'accumulatore)
	30	byte meno significativo dell'indirizzo
	C0	byte più significativo dell'indirizzo

Nuove istruzioni

1. DEC (*DECrement memory*, decrementa la memoria), usata nel modo pagina zero per decrementare (diminuire di uno) il valore contenuto nella locazione di memoria specificata della pagina zero.

Esempio	C6	codice op. per DEC
	01	byte meno significativo della locazione di memoria 0001 (è implicito che il byte più significativo è 00).

2. DEX (*DEcrement X register*, decrementa il registro X), usata in modo implicito per diminuire di uno il valore contenuto nel registro X.

Esempio CA codice op. per DEX.

3. LDX (*LoaD the X register*, carica il registro X), usata nel modo pagina zero per caricare il registro X con il valore contenuto nella locazione specificata di pagina zero.

Esempio A6 codice op. per LDX
00 byte meno significativo della locazione 0000
(è implicito che il byte più significativo è 00)

4. PHA (*PusH Accumulator on stack*, metti l'accumulatore sullo stack), usata per mettere il valore contenuto nell'accumulatore in cima allo stack. Questo valore verrà successivamente ripreso per essere riutilizzato. Modo implicito.

Esempio 48 codice op. per PHA

5. PLA (*PuLL from stack to Accumulator*, sposta dallo stack all'accumulatore), usata per recuperare un valore messo precedentemente sullo stack. Il valore viene posto nell'accumulatore per essere usato. Modo implicito.

Esempio 68 codice op. per PLA

6. TAX (*Transfer data from Accumulator to X register*, trasferisci il dato dall'accumulatore al registro X), usata per trasferire (copiare) il dato contenuto nell'accumulatore nel registro X. Modo implicito.

Esempio AA codice op. per TAX

7. TXA (*Transfer from the X register to the Accumulator*, trasferisci dal registro X all'accumulatore), usata per trasferire (copiare) il contenuto del registro X nell'accumulatore. Modo implicito.

Esempio 8A codice op. per TXA

TAVOLA DELLE SUBROUTINE FINORA USATE

<i>Funzione</i>	<i>Locazione di memoria</i>
Pizzica l'altoparlante	C030
Cancella lo schermo	FC58
Abilita il modo grafico	FB40
Disegna un punto	F800
Disegna una linea orizzontale	F819
Disegna una linea verticale	F828
Riceve un carattere	FD35
Stampa il carattere nell'accumulatore	FDED
Stampa il contenuto dell'accumulatore come cifre esadecimale	FDDA

TAVOLA DELLE ISTRUZIONI DEL
LINGUAGGIO MACCHINA

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
ASL	Accumulatore	0A	1	Sposta i bit a sinistra
BEQ	Relativo	F0	2	Salta se uguale
BNE	Relativo	D0	2	Salta se non uguale
CMP	Immediato	C9	2	Confronta l'accumulatore
CPX	Immediato	E0	2	Confronta il reg. X
CPY	Immediato	C0	2	Confronta il reg. Y
*DEC	Pagina zero	C6	2	Decrementa la memoria
*DEX	Implicito	CA	1	Decrementa il registro
INX	Implicito	E8	1	Incrementa il registro X
INY	Implicito	C8	1	Incrementa il registro Y
JMP	Assoluto	4C	3	Salta alla memoria
JSR	Assoluto	20	3	Salta alla subroutine
LDA	Immediato	A9	2	Carica l'accumulatore
LDA	Assoluto indicizzato	BD	3	Carica l'accumulatore
LDX	Immediato	A2	2	Carica il registro X
*LDX	Pagina zero	A6	2	Carica il registro X
LDY	Immediato	A0	2	Carica il registro Y
NOP	Implicito	EA	1	Nessuna operazione
*PHA	Implicito	48	1	Mette l'accumulatore sullo stack

<i>Codice mnemonico indirizzamento</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
*PLA	Implicito	68	1	Sposta dallo stack all'accumulatore
RST	Implicito	60	1	Ritorno dalla subroroutine
STA	Pagina zero	85	2	Memorizza l'accumulatore
STA	Absolute	8D	3	Memorizza l'accumulatore
STA	Ass. indicizzato	99	3	Memorizza l'accumulatore
*TAX	Implicito	AA	1	Trasferisce l'acc. nel registro X
*TXA	Implicito	8A	1	Trasferisce l'acc. nel registro X

*Istruzioni introdotte in questo capitolo

ESERCIZI

1. Spiegare il risultato dell'esecuzione di ognuna delle seguenti istruzioni in linguaggio macchina.

a. CA DEX

b. C6 DEC 0001
01

c. A6 LDX 0000
00


2. Che cosa succede se inserite uno 0 (zero) per l'intensità del Programma sui toni?


3. Un valore basso per l'intensità nel Programma sui toni produce un tono _____ Un valore alto produce un tono _____
(basso, alto) (basso, alto)

4. Descrivere le operazioni causate da queste istruzioni

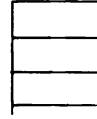
a. 8A TAX

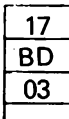
b. AA TXA

5. Lo stack ha questo contenuto 

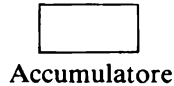
L'accumulatore contiene il valore .

Poi viene eseguita l'istruzione PHA. Mostrare che contenuto ha lo stack dopo questa esecuzione.



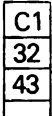
6. Se lo stack ha questo contenuto 

e viene eseguita l'istruzione PLA, mostrate il valore contenuto nell'accumulatore ed il valore in cima allo stack.





Stack

RISPOSTE AGLI ESERCIZI

1. a. Decrementa di uno il valore contenuto nel registro X.
b. Decrementa di uno il contenuto della locazione 0001.
c. Carica il registro X dalla locazione 0000.
2. L'esecuzione del programma in linguaggio macchina viene fermata. Il controllo è ritornato al sistema operativo.
3. Alto, basso.
4. a. TXA copia i dati del registro X nell'accumulatore.
b. TAX copia i dati dall'accumulatore nel registro X.
5.  ← C1 in cima
Tutto il resto è spostato in giù di una posizione

Stack

6.  Accumulatore 
Stack
17 è stato tolto
Tutto il resto si sposta in su di una posizione.

Ancora suoni e grafica

SOB

Nel Cap. 6 avete imparato come pizzicare l'altoparlante dell'Apple per produrre dei suoni. Il suono è bello, ma non è molto soddisfacente da solo. In questo capitolo, combineremo il suono con la grafica per stimolare contemporaneamente due dei vostri sensi.

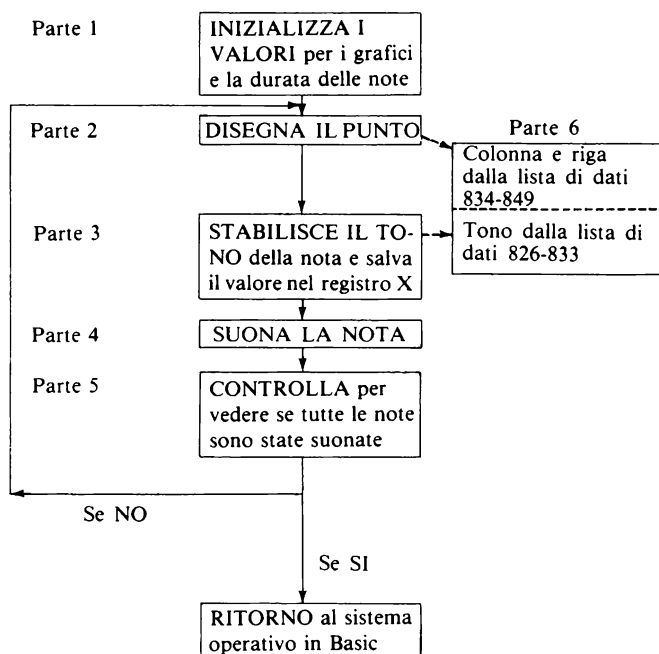
COMBINAZIONE DELL'ALTOPARLANTE E DEL VIDEO

Costruiremo un nuovo programma usando le tecniche grafiche apprese nel Cap. 4 ed il programma Scala automatica usato nel Cap. 6. La lunghezza del programma aumenterà e verranno fatte alcune modifiche per far sì che i suoni ed i grafici si adattino l'un l'altro in modo coordinato. Sarà necessaria soltanto una nuova istruzione. Questa istruzione, LDY, carica il registro Y con il contenuto della memoria indicizzata dal registro X. Ricordate, il programma disegna-punti usa il registro Y per contenere la colonna dove disegnare il punto. Questi valori sono contenuti in una lista di dati che inizia alla locazione 0834.

Prima di guardare il programma, studiate il diagramma di flusso che segue. Il diagramma è organizzato in blocchi che corrispondono a parti individuali del programma. Avete già visto ogni istruzione della parte 1. La durata delle note viene caricata nell'accumulatore (dall'istruzione

LDA a 768). Lo schermo è cancellato (dalla subroutine a FC58). Il registro X, che è usato come contatore, viene azzerato (dall'istruzione LDX a 775). Viene abilitato il modo grafico in bassa risoluzione (dalla subroutine a FB40). Per il disegno delle note è fissato il colore bianco (dall'istruzione LDA a 780 e dall'istruzione STA a 782).

Diagramma di flusso per il programma Scala con Note



PROGRAMMA SCALA CON NOTE

1 COMMENTO **INIZIALIZZAZIONE**

768 A9	LDA C8	Fissa la durata
769 C8		
770 20	JSR FC58	Cancella lo schermo
771 58		
772 FC		
775 A2	LDX 00	Azzera il contatore
776 00		

777 20	JSR FB40	Abilita il modo grafico
778 40		
779 FB		
780 A9	LDA 0F	Sceglie il colore bianco
781 0F		
782 85	STA 0030	Lo mette in memoria
783 30		

La parte 2 carica la colonna sulla quale dovrà essere disegnato il punto nel registro Y e la rispettiva riga nell'accumulatore. Quindi viene richiamata la subroutine che disegna il punto. I valori della colonna e della riga sono memorizzati nelle tavole di dati alla fine del programma. Il registro X è usato come puntatore per accedere a questi dati.

2 COMMENTO **DISEGNA IL PUNTO**

784 BC	LDY 0342,X	Carica il registro Y da 0342 indicizzata dal valore contenuto nel registro X
785 42		
786 03		
787 BD	LDA 034A,X	Carica l'accumulatore da 034A indicizzata dal valore nel registro X
788 4A		
789 03		
790 20	JSR F800	Chiama la subroutine interna disegna-punti
791 00		
792 F8		

Nella parte 3, il valore del tono della nota viene caricato nell'accumulatore dalla tavola di dati ed è poi posto nella locazione 0000, dove sarà necessario quando verrà suonata la nota. Il valore contenuto nel registro X è messo nell'accumulatore e da qui salvato sullo stack.

Questo deve essere fatto, dato che il registro X è usato nella parte 4 per altri scopi. Il valore salvato sullo stack sarà recuperato più avanti quando necessario.

3 COMMENTO **SI PREPARA PER LA NOTA**

793 BD	LDA 033A,X	Carica il valore del tono da 033A indicizzata dal valore nel registro X
794 3A		
795 03		
796 85	STA 0000	Memorizza in 0000
797 00		

798	8A	TXA	Mette il valore del registro X nell'accumulatore
799	48	PHA	Lo mette sullo stack per salvarlo

La quarta parte suona la nota. È la stessa di quella usata nella terza parte del programma Scala automatica nel Cap. 6.

4 COMMENTO **SUONA LA NOTA**

800	AD	LDA C030	Pizzica l'altoparlante
801	30		
802	C0		
803	88	DEY	Come nel precedente programma sui suoni
804	D0	BNE 04	
805	04		
806	C6	DEC 0001	
807	01		
808	F0	BEQ 08	
809	08		
810	CA	DEX	
811	D0	BNE F6(— 10)	
812	F6		
813	A6	LDX 0000	
814	00		
815	4C	JUMP 0320	
816	20		
817	03		

Nella parte 5, il computer controlla se tutte le note della scala sono state suonate. Se non è così, il programma ritorna alla parte 2 dove verrà disegnata e suonata una nuova nota. Se tutte le note sono state suonate, il controllo ritorna al sistema operativo. Il vecchio valore del registro X è tolto dallo stack e rimesso nel registro X. Questo valore è quindi incrementato di uno. Poi il nuovo valore viene confrontato con 8 (il numero di note desiderato).

5 COMMENTO **CONTROLLA SE TUTTE LE NOTE SONO STATE SUONATE**

818 68	PLA	Toglie il vecchio valore di X dallo stack
819 AA	TAX	Lo trasferisce nel registro X
820 E8	INX	Incrementa il contenuto di uno
821 E0	CPX 08	Confronta la nota suonata con 8
822 08		
823 D0	BNE D7	Se no, torna indietro a prendere il valore di una nuova nota da disegnare e tracciare a 784
824 D7		
825 60	RTS	Ritorna al monitor

Le tavole di dati per la colonna e le righe usate per disegnare le note ed i valori del tono formano la sesta parte.

6 COMMENTO **LISTA DI DATI**

826 85	← Valori dell'intensità
827 7F	
828 79	
829 74	
830 6F	
831 6B	
832 67	
833 64	
834 07	← Valori delle colonne per disegnare le note
835 09	
836 0B	
837 0D	
838 0F	
839 11	
840 13	
841 15	
842 16	← Valori delle righe per disegnare le note
843 14	
844 12	

845 10
846 0E
847 0C
848 0A
849 08

Combinando i grafici con il programma Scala automatica, siete in grado di visualizzare ogni nota mentre viene suonata.

La parte 1 fissa la durata della nota, cancella lo schermo ed azzerà il registro X cosicché si possa accedere ai dati nelle varie locazioni di memoria. Viene poi abilitato il modo graphics e scelto il colore (se volete, lo potete cambiare alla locazione 781).

La parte 2 carica il registro Y con la colonna e l'accumulatore con la riga dove dovrà essere disegnata la nota. Entrambe le istruzioni usano il registro X come un indice per scegliere i valori desiderati dalla tavola, posizionata in memoria ad una specifica locazione. Viene quindi "chiamata" la routine disegna-punti per disegnare il punto. La parte 3 sceglie il tono mediante le istruzioni LDA e STA a 793 e 796. Essa, a 798-799, salva anche il valore dell'indice sullo stack.

La parte 4 è un vecchio amico, suona la nota.

La parte 5 riprende il valore dell'indice e lo confronta con 8 per vedere se tutte le note sono state suonate. Se non è così, l'istruzione di salto a 823 causa un ritorno a 784 per disegnare e suonare una nuova nota. Se le otto note sono state suonate, si ritorna al sistema operativo. La parte 6 fornisce i dati in tre blocchi. Il primo blocco (826-833) contiene i valori del tono per suonare le 8 note. Il secondo blocco (834-841) riporta i valori della colonna in cui ognuna delle 8 note sarà disegnata. Il terzo (842-849) fornisce i valori delle righe in cui verranno disegnate le note.

INSERIMENTO ED ESECUZIONE DEL PROGRAMMA

È il momento di inserire il programma, che, ancora una volta, inizia alla locazione 768. È un programma lungo, 82 byte.

Dopo aver inserito il programma, controllatelo per vedere se ci sono errori. Quando ne sarà privo, eseguitelo.

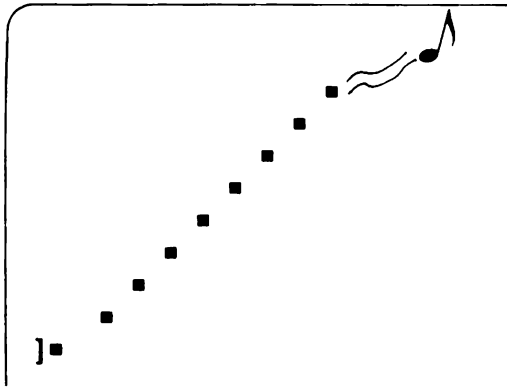
Prima apparirà una nota e udirete il do centrale.



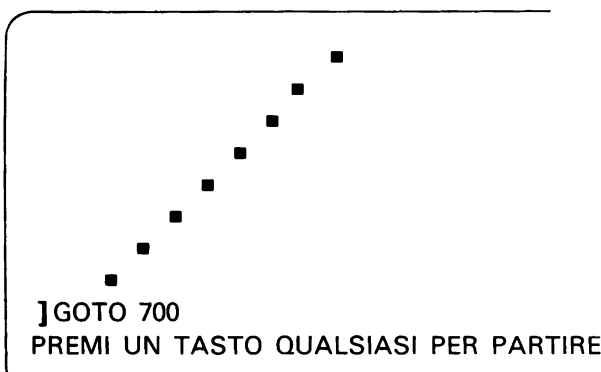
Poi apparirà una seconda nota con un secondo tono.



Questo continua finché tutte le 8 note sono state visualizzate e suonate.



Il programma poi ritorna al sistema operativo in Basic. Se volete vedere e riascoltare il risultato, scrivete GOTO 700 e premete RETURN. Il video mostrerà.



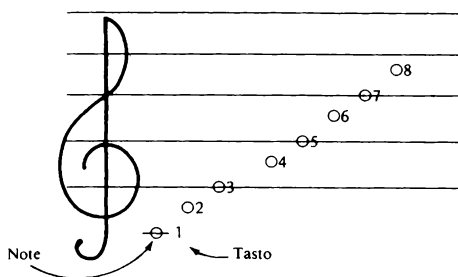
Premete un tasto qualsiasi ed il processo si ripeterà.

Ma ora pensate alle molte variazioni che potrebbero essere fatte. Potreste disegnare ogni nota in un colore differente. Oppure inserire una corta melodia al posto della scala. Potreste far scomparire ogni nota dopo essere stata suonata. Ci sono molte variazioni che potete tentare. Vi in-

coraggiamo a tentarle tutte e ogni altra che vi venga in mente. Ora avete le basi per realizzare un organo a colori che possa suonare musica e visualizzare colori sullo schermo. Non dovete fissarvi sul disegno di un singolo punto. Potete riempire lo schermo di colori, oppure progettare schemi di colori casuali mentre vengono suonate le note. Quando avete provato a sufficienza il programma, potete passare a qualcosa di più interessante. Non sarebbe bello riuscire a suonare le note che vengono scelte dalla tastiera? Allora sarete in grado di comporre e suonare musica vostra. Proseguite la lettura e vi mostreremo un programma che vi permetterà di "strimpellare" sulla tastiera la vostra musica.

USO DELLA TASTIERA PER SUONARE LE NOTE

Avete usato un programma che richiede in input la nota che si vuole suonare. Ne avete visto anche uno dove il computer suonava la scala musicale per voi. Il prossimo programma combina le due tecniche cosicché possiate inserire delle note con continuità. È limitato ad un'ottava per semplificare le cose. I valori per ogni nota sono memorizzati in una lista di dati e premendo uno dei tasti numerici 1, 2, 3, 4, 5, 6, 7, 8, il computer sceglierà la nota corrispondente. Non occorre che premiate il tasto RETURN. Ricordate, si è supposto che la nostra scala contenga l'ottava che parte con il do centrale. I tasti, e le note corrispondenti, sono mostrati qui sotto.



PROGRAMMA SUONA LA TUA MELODIA

1 COMMENTO **CANCELLA LO SCHERMO**

768 20 JSR FC58
 769 58
 770 FC

Routine interna per cancellare lo schermo

2 COMMENTO **RICEVE UN TASTO**

771 20	JSR FD35	Routine interna per leggere la tastiera
772 35		
773 FD		
774 C9	CMP B0	Il tasto è troppo basso?
775 B0		
*776 30	BMI F9	Se si torna indietro e guarda ancora (—7
777 F9		passi)
778 F0	BEQ 25	Se è uno zero, ritorna al Basic
779 25		
780 C9	CMP B8	Il tasto è troppo alto?
781 B9		
*782 10	BPL F3	Se sì, torna indietro e guarda ancora (—13
783 F3		passi)

3 COMMENTO **CONVERTE IL TASTO ASCII USATO COME INDICE**

*784 29	AND OF	Rimuove i 4 bit più in alto del codice ASCII
785 0F		
786 AA	TAX	Trasferisce il risultato nel registro X da usare come indice

4 COMMENTO **RICEVE LA DURATA E LA NOTA **

787 A9	LDA 60	Carica la durata
788 60		
789 85	STA 0001	La memorizza
790 01		
791 BD	LDA 0332,X	Carica l'accumulatore dalla lista di dati
792 32		
793 03		
794 85	STA 0000	Lo memorizza
795 00		

5 COMMENTO **SUONA LA NOTA**

796 AD	LDA C030	Pizzica l'altoparlante
797 30		
798 C0		
799 88	DEY	
800 D0	BNE 04	
801 04		
802 C6	DEC 0001	
803 01		
804 F0	BEQ 08	
805 08		

```
806 CA    DEX
807 D0    BNE F6
808 F6
809 A6    LDX 0000
810 00
811 4C    JMP 031C
812 1C
813 03
```

6 COMMENTO **RICEVE UNA NUOVA NOTA O FINISCE**

```
814 4C    JMP 0303    Riceve una nuova nota
815 03
816 03
817 60    RTS          Ritorno al sistema operativo in Basic
```

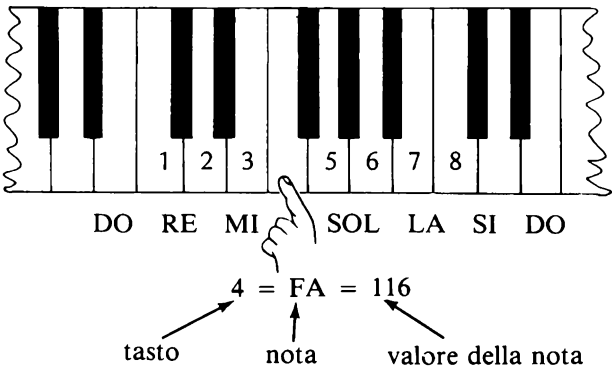
7 COMMENTO **LISTA DI DATI**

```
*818 EA    NOP
819 85
820 7F
821 79
822 74
823 6F
824 6B
825 67
826 64
```

← Nota più bassa dell'ottava

← Nota più alta dell'ottava

*Nuova istruzione



DESCRIZIONE DEL PROGRAMMA

La parte 1 cancella lo schermo. La 2 usa la subroutine interna per vedere se è stato premuto un tasto. Se è stato premuto, il codice ASCII del carattere di quel tasto sarà posto nell'accumulatore. I codici ASCII dei nove simboli usati sono:

Cifre	Codici ASCII (in ESA)
0	B0
1	B1
2	B2
3	B3
4	B4
5	B5
6	B6
7	B7
8	B8

Dopo la pressione di un tasto, il valore nell'accumulatore viene confrontato con B0 e B8 (il campo di variazione delle cifre). Se il codice ASCII è minore di B0 o maggiore di B8, il tasto non sarà accettato. Il programma non suonerà la nota, ma ritornerà a leggere un altro tasto. Questo è effettuato dalle istruzioni BMI (*Branch if MINus*, salta se meno) e BPL (*Branch if PLus*, salta se più). Queste due nuove istruzioni sono simili alle istruzioni BNE (*Branch if Not Equal*, salta se non uguale) e BEQ (*Branch if EQual*, salta se uguale) che avete già usato.

BMI (SALTA SE IL RISULTATO È MENO)

<i>Modo di indirizzamento</i>	<i>Codice mnemonico</i>	<i>Codice op.</i>
Relativo	BMI	30

Come è usato nel programma:

774 C9	Confronta il codice ASCII del tasto con B0
775 B0	
776 30	Se il risultato è negativo salta indietro di 7 passi
777 F9	(F9 = -7)

Se il codice ASCII del tasto è minore di B0, l'istruzione di salto sul meno manda l'esecuzione del programma indietro a 771 per leggere ancora un tasto. Così, ogni tasto il cui codice ASCII è minore di B0 *non sarà accettato*.

BPL (SALTA SE IL RISULTATO È PIÙ)

<i>Modo di indirizzamento</i>	<i>Codice mnemonico</i>	<i>Codice op.</i>
Relativo	BPL	10

Come è usato nel programma:

780 C9	Confronta il codice ASCII del tasto con B9
781 B8	
782 10	Se il risultato è più salta indietro di 13 passi
783 F3	(F3 = -13)

Se il codice ASCII del tasto è maggiore di B8 (zero è considerato positivo da questa istruzione), l'istruzione di salto sul più manda l'esecuzione del programma indietro a 771 per leggere un altro tasto. Così, ogni tasto il cui codice ASCII è maggiore di B8 *non sarà accettato*. Se si preme lo zero, viene effettuato un salto all'istruzione RTS che riporta al sistema operativo. È usato per fermare il programma quando avrete deciso di smettere di suonare note.

La parte 3 converte il valore ASCII in una cifra decimale cosicché possa essere usata nel registro X come indice per accedere alla nota corretta nella lista dei dati. Guardando la tabella dei codici ASCII delle cifre decimali data sopra, potete notare una somiglianza tra la cifra ed il suo codice ASCII. Viene usata una nuova istruzione, AND, usata per eliminare i quattro bit più in alto del codice ASCII. In altre parole, la B viene rimossa dai valori compresi tra B0 e B8, lasciando solo la cifra decimale sul lato destro del valore esadecimale a due cifre.

AND (AND CON L'ACCUMULATORE)

<i>Modo di indirizzamento</i>	<i>Codice mnemonico</i>	<i>Codice op.</i>
Immediato	AND	29

Come è usato in questo programma:

Il codice ASCII del tasto è nell'accumulatore.

784 29 Fa l'AND tra il valore contenuto nell'accumu-
785 0F latore ed il valore ESA 0F

Come esempio, vedremo il codice ASCII nella sua forma binaria. L'istruzione AND opera sul codice ASCII nell'accumulatore ed il valore che segue immediatamente l'istruzione AND. Essa confronta i bit corrispondenti dei due valori. Se un particolare bit è uno (1) per entrambi i valori, il corrispondente bit del risultato sarà uno. Se il bit di uno dei due valori (o di entrambi) è zero, il corrispondente bit del risultato sarà zero.

Esempi

Esegue l'AND fra l'accumulatore ed il valore 0F

Il tasto è 5.

Il codice ASCII è B5 = 1 0 1 1 0 1 0 1 in binario

AND con il valore 0F = 0 0 0 0 1 1 1 1 in binario

	<div style="border-top: 1px solid black; display: inline-block; width: 100%;"></div>
	<div style="display: flex; justify-content: space-around; width: 100%;"> ↓↓ </div>
Il risultato resta nell'accumulatore	0 0 0 0 0 1 0 1

Risultato = 05 (il tasto)

Esegue l'AND fra l'accumulatore ed il valore 0F.

Il tasto è 8.

Il codice ASCII è B8 = 1 0 1 1 1 0 0 0 in binario

AND con il valore 0F = 0 0 0 0 1 1 1 1 in binario

	<div style="border-top: 1px solid black; display: inline-block; width: 100%;"></div>
	<div style="display: flex; justify-content: space-around; width: 100%;"> ↓↓ </div>
Il risultato resta nell'accumulatore	0 0 0 0 1 0 0 0

Risultato = 08 (il tasto)

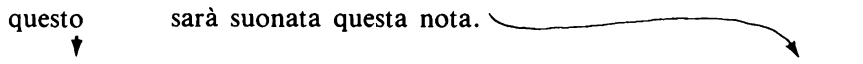
Il modo in cui viene usato l'AND in questo programma (AND 0F), fa sì che il risultato scarti i quattro bit a sinistra (bit più significativi) del

valore nell'accumulatore e tenga i quattro bit a destra (bit meno significativi). Il programma poi mette questo risultato nel registro X (istruzione TAX a 786) cosicché possa essere usato per indicizzare il comando LDA nella parte 4 che carica dalla memoria la tonalità corretta per un dato tasto.


La sequenza risultante da un tasto è mostrata qui sotto:

Se scrivete

questo sarà suonata questa nota.



Tasto premuto	Codice ASCII	Dopo l' AND	Accesso alla memoria con LDA 0332,X	Memoria ESA	Indirizzo decimale	Contenuto della memoria (Valore della nota)
1	B1	01	0332 + 1 ➡	0333	819	85
2	B2	02	0332 + 2 ➡	0334	820	7F
3	B3	03	0332 + 3 ➡	0335	821	79
4	B4	04	0332 + 4 ➡	0336	822	74
5	B5	05	0332 + 5 ➡	0337	823	6F
6	B6	06	0332 + 6 ➡	0338	824	6B
7	B7	07	0332 + 7 ➡	0339	825	67
8	B8	08	0332 + 8 ➡	033A	826	64



Questo valore viene messo nel registro X

Ancora una volta, vediamo che l'azione ha luogo nell'accumulatore (causata dall'istruzione AND) ed il risultato viene poi trasferito nella locazione (il registro X in questo caso) dove sarà usato.

La parte 4 carica la durata della nota (60 esadecimale) nella locazione 0001. Quindi il valore di tonalità viene caricato dalla locazione 0332 + il valore nel registro X. Questo è memorizzato nella locazione 0000. A causa dell'istruzione AND usata nella parte 3, la nota sarà caricata da 0332 + qualsiasi tasto sia stato battuto (1, 2, 3, 4, 5, 6, 7, 8).

La parte 5 vi dovrebbe essere ormai familiare. Essa suona la nota usando i valori di tonalità e di durata che sono memorizzati nella parte 4.

La parte 6 fornisce un'istruzione di salto per tornare ad un'altra nota. Contiene anche un'istruzione RTS che segue quella di salto. L'istruzione RTS sarà raggiunta soltanto dalla parte 2 (778 BEQ) se il tasto premuto è zero (0). Naturalmente, RTS vi riporta al sistema operativo come nei precedenti programmi.

La parte 7 è la lista dei dati da cui vengono selezionate le note. Una nuova istruzione (NOP) che non fa nulla è usata all'inizio della lista di dati. Questa istruzione (*No Operation*, nessuna istruzione) è usata come buffer per separare il programma dalla lista di dati. L'istruzione che carica i dati dalla lista (BD alla locazione 791) punta il valore EA (*No Operation*), ma è indicizzata dal registro X che è sempre maggiore o uguale a uno. Perciò, l'istruzione NOP (EA) non verrà mai eseguita. Spesso, si usano alcune istruzioni NOP per "salvare un posto" per altre istruzioni che potranno essere aggiunte successivamente per modificare il programma. Se volessimo inserire una nota minore del do centrale, potremmo usare la memoria occupata dall'istruzione NOP. Il programma potrebbe allora essere modificato in modo da usare il tasto zero per accedere a questa nota.

Il puntatore alla lista dei dati funziona così:

Dalla quarta parte	{	791 BD LDA 0332,X (0332 = 818 decimale)	
		792 32	
		793 03	X contiene il valore inserito (da 1 a 8) e determina a che valore accedere qui
			Il puntatore di dati parte qui
			Se X = 1, 0332 + 1 punta qui
			Se X = 2, 0332 + 2 punta qui
			Se X = 3, 0332 + 3 punta qui
			Se X = 4, 0332 + 4 punta qui
			Se X = 5, 0332 + 5 punta qui
			Se X = 6, 0332 + 6 punta qui
			Se X = 7, 0332 + 7 punta qui
			Se X = 8, 0332 + 8 punta qui
Dalla settima parte	{		818 EA
			819 85
			820 7F
			821 79
			822 74
			823 6F
			824 6B
			825 67
			826 64

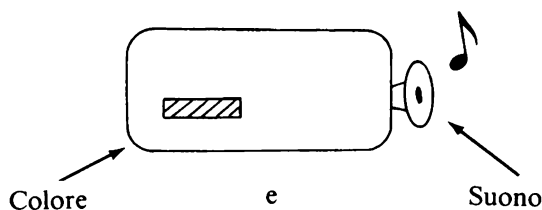
INSERIMENTO ED ESECUZIONE DEL PROGRAMMA

Inserite il programma che inizia alla locazione 768 decimale. È lungo 59 byte. Controllate se avete fatto errori, eventualmente correggeteli ed eseguite il programma.

Lo schermo verrà cancellato. Poi il cursore lampeggiante apparirà nell'angolo in alto a sinistra.



È ora di inserire le note. Premendo un qualsiasi tasto da 1 a 8 verrà suonata una nota. Dopo ogni nota il cursore tornerà nell'angolo in alto a sinistra ed il computer aspetterà il prossimo tasto. Suonate per un po' finché non avrete una certa confidenza con il calcolatore-organo. Poi chiamate i vostri amici e mostrate loro il vostro talento musicale. Sono possibili molte modifiche di questo programma. Aumentando la lunghezza della lista dei dati potete aumentare il numero di note disponibili. Potete anche aggiungere dei grafici colorati per avere un organo a colori.



SOMMARIO

In questo capitolo sono stati usati due lunghi programmi per dimostrare l'uso dei colori e dei suoni. Avete imparato a coordinare l'uso dell'altoparlante, del video e della tastiera. Sono state introdotte cinque nuove istruzioni. La lista di istruzioni che avete usato è cresciuta abbastanza per costruire programmi più complessi ed utili. Avete ricevuto ulteriori informazioni sulle possibilità del registro Y con due nuove istruzioni. Sono state introdotte due nuove istruzioni di salto, ed avete usato l'istruzione logica AND per convertire un codice ASCII nella sua cifra decimale equivalente.

Ecco il sommario delle nuove istruzioni introdotte in questo capitolo.

1. AND (*AND with accumulator*, AND con l'accumulatore), usata in modo immediato per eseguire l'AND logico tra il valore che segue l'istruzione ed il contenuto dell'accumulatore. Il risultato contiene 1 nei bit dove entrambi i valori hanno uno. Altrimenti zero.

Esempio 29 Codice op. per AND
 0F Valore ESA con cui eseguire AND

2. BMI (*Branch on MINus*, salta se meno), usata nel modo immediato per effettuare un salto quando una data condizione è negativa. Il dato che segue l'istruzione dice in che direzione, e di quanto, saltare dalla posizione corrente del contatore di programma.

Esempio 30 Codice op. per BMI
 F9 il dato dice di tornare indietro di 7 passi
 (F9 = -7)

3. BPL (*Branch on PLus*, salta se più), usata nel modo immediato per effettuare un salto quando una data condizione è positiva o nulla. Il dato che segue l'istruzione dice in che direzione, e di quanto, saltare dalla posizione corrente del contatore di programma.

Esempio 10 Codice op. per BPL
 F3 Il dato dice di tornare indietro di 13 passi
 (F3 = -13)

4. DEY (*DEcrement Y register*, decrementa il registro Y), usata nel modo implicito per diminuire di uno il valore contenuto nel registro Y.

Esempio 88 Codice op. per DEY

5. LDY (*LoaD Y register*, carica il registro Y), usata nel modo assoluto indicizzato per caricare il registro Y dalla locazione di memoria indicizzata dal valore contenuto nel registro X.

Esempio BC codice op. per LDY (assoluto indicizzato)
 42 byte meno significativo della locazione
 03 byte più significativo della locazione

Il valore nel registro X è aggiunto a questo indirizzo "di base" per ottenere il vero indirizzo.

TAVOLA DELLE ISTRUZIONI DEL LINGUAGGIO MACCHINA USATE

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
*AND	Immediato	29	2	Fa l'AND dei bit con l'accumulatore
ASL	Accumulatore	0A	1	Sposta i bit a sinistra
BEQ	Relativo	F0	2	Salta se uguale
*BMI	Relativo	30	2	Salta se meno
BNE	Relativo	D0	2	Salta se non uguale
*BPL	Relativo	10	2	Salta se più

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
CMP	Immediato	C9	2	Confronta l'accumulatore
CPX	Immediato	E0	2	Confronta il reg. X
CPY	Immediato	C0	2	Confronta il reg. X
DEC	Pagina zero	C6	2	Decrementa la memoria
DEX	Implicito	CA	1	Decrementa il reg. X
*DEY	Implicito	88	1	Decrementa il reg. X
INX	Implicito	E8	1	Incrementa il registro X
INY	Implicito	C8	1	Incrementa il registro Y
JMP	Assoluto	4C	3	Salta alla memoria
JSR	Assoluto	20	3	Salta alla subroutine
LDA	Immediato	A9	2	Carica l'accumulatore
LDA	Assoluto indicizzato	BD	3	Carica l'accumulatore
LDX	Immediato	A2	2	Carica il registro X
LDX	Pagina zero	A6	2	Carica il registro X
LDY	Immediato	A0	2	Carica il registro Y
*LDY	Assoluto indicizzato	BC	3	Carica il registro Y
NOP	Implicito	EA	1	Nessuna operazione
PHA	Implicito	48	1	Toglie dallo stack e mette nell'accumulatore
PLA	Implicito	68	1	Toglie dallo stack e mette nell'accumulatore
RTS	Implicito	60	1	Ritorno dalla subroutine
STA	Pagina zero	85	2	Memorizza l'accumulatore
STA	Assoluto	8D	3	Memorizza l'accumulatore
STA	Ass. indicizzato	99	3	Memorizza l'accumulatore
TAX	Implicito	AA	1	Trasferisce l'acc. nel registro X
TXA	Implicito	8A	1	Trasferisce l'acc. nel registro X

*Istruzioni introdotte in questo capitolo

TAVOLA DELLE SUBROUTINE FINORA USATE

<i>Funzione</i>	<i>Locazione di memoria</i>
Pizzica l'altoparlante	C030
Cancella lo schermo	FC58
Abilita il modo grafico	FB40
Disegna un punto	F800
Disegna una linea orizzontale	F819
Disegna una linea verticale	F828
Riceve un tasto	FD35
Stampa il carattere nell'accumulatore	FD35
Stampa il contenuto dell'accumulatore come cifre esadecimali	FDDA

ESERCIZI

1. Spiegate il risultato dell'esecuzione delle seguenti istruzioni del linguaggio macchina

- a. BC LDY 0342,X
 42
 03

b. 88 DEY

2. Nel programma Scala con note, alle linee 821 e 822, viene usata un'istruzione di confronto tra X e 08 (CPX 08). Quale delle seguenti operazioni esegue?

- a. 08 — valore nel registro X
 b. valore nel registro X — 08

3. Nel programma Suona la tua melodia viene usata una subroutine, che si trova all'indirizzo FD35, per leggere la tastiera. Quando, in questo programma, si preme un tasto per suonare una nota, è necessario poi che premete il tasto RETURN? _____

4. Se viene eseguito l'AND tra i seguenti valori esadecimali ed 0F, quali saranno i risultati?

- a. B7 _____ b. B4 _____ c. E5 _____

5. Quale sarà il risultato dell'AND tra 3F (esadecimale) e 63 (esadecimale)? _____

6. Nel programma Suona la tua melodia, quali dei seguenti valori ASCII farà suonare una nota?
- a. A9 b. B3 c. BA d. B0
-

RISPOSTE AGLI ESERCIZI

1. a. Il registro Y è caricato dalla locazione di memoria il cui indirizzo è 0342 + il valore nel registro X.
b. Il contenuto del registro Y viene diminuito (decrementato) di uno.
2. b. Valore nel registro X -8
3. No
4. a. 07 (o 7)
b. 04 (o 4)
c. 05 (o 5)
5. 23 3F = 0011 1111
 63 = 0110 0011

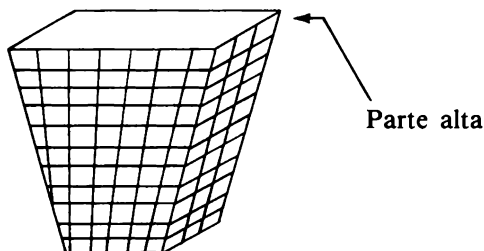
 AND = 0010 0011
 $\underbrace{\hspace{1.5cm}}_{2} \quad \underbrace{\hspace{1.5cm}}_{3}$
6. b. Solo B3 (A9 è troppo basso-non accettato, BA è troppo alto-non accettato, B0 causa un ritorno al sistema operativo in Basic).

Il monitor di sistema Apple

MS

Nei precedenti capitoli abbiamo usato il Basic per scrivere ed inserire programmi in linguaggio macchina. Questo metodo ci ha permesso di utilizzare le nostre conoscenze di Basic per introdurci al linguaggio del computer. È giunto il momento di lasciar perdere il sistema operativo in Basic. Non ne abbiamo più bisogno.

Il calcolatore Apple ha un monitor di sistema situato nella parte alta della sua memoria (vedere la mappa della memoria nel Cap. 2).



Il monitor controlla *tutti* i programmi, e *tutti* i programmi lo usano. Può darsi che non ve ne rendiate conto, ma lo avete usato in tutto questo libro. Da qui in poi, comunicherete con esso direttamente. Potete usare il monitor per guardare direttamente nelle locazioni di me-

moria, cambiarne i contenuti ed anche scrivere dei programmi in linguaggio macchina eseguibili direttamente dal microprocessore 6502.

Ricordate, il calcolatore comprende soltanto istruzioni codificate in binario. Il monitor di sistema accetta queste istruzioni in forma esadecimale dalla tastiera. Potete ritornare al Cap. 3 per rivedere brevemente il formato del codice delle istruzioni.

Ritorniamo ad uno dei primi programmi in linguaggio macchina, descritto nel Cap. 3. Esso carica l'accumulatore con il valore esadecimale 13 e memorizza il valore nella locazione 0325 (esadecimale). Il programma era stato memorizzato così:

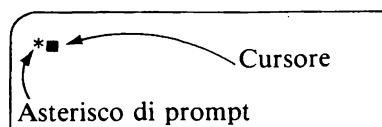
<i>Indirizzo</i>		<i>Codice</i>	<i>Commenti</i>
<i>Decimale</i>	<i>ESA</i>	<i>op.</i>	
768	0300	A9	LDA con il dato
769	0301	13	
770	0302	8D	STA in memoria
771	0303	25	
772	0304	03	0325
773	0305	60	RTS

Questo programma era stato chiamato dal nostro sistema operativo, cosicché l'ultima istruzione è RTS (*ReTurn from Subroutine*). Il monitor di sistema dell'Apple tratta i programmi in linguaggio macchina come subroutine. Perciò, RTS può essere anche usato come ultima istruzione nei programmi in linguaggio macchina. Il programma sopra può essere inserito direttamente usando il monitor di sistema. Vi mostreremo ora come usare il monitor, cosicché possiate vedere com'è facile caricare, esaminare ed eseguire direttamente un programma in linguaggio macchina.

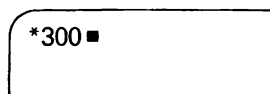
Guardate sul manuale d'uso per vedere come inserire il monitor di sistema per la vostra versione dell'Apple. Per la nostra, tutto quello che dobbiamo fare è premere il tasto RESET. Un altro modo è di spegnere e riaccendere il computer. Nella nostra versione, l'accensione ci fa accedere sempre al monitor. Se avete un sistema Apple II Plus con ROM Auto-start, entrate nel monitor di sistema eseguendo l'istruzione del Basic:

CALL 65385
oppure
CALL -151

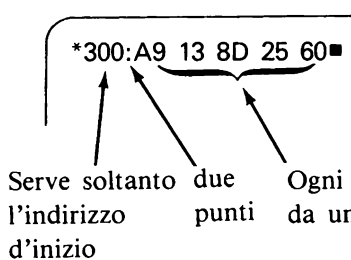
Quando siete nel monitor di sistema, lo schermo mostrerà l'asterisco come segnale di prompt.



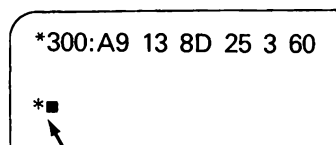
Per inserire il programma prima inserite l'indirizzo di partenza in esadecimale 300



Poi battete due punti, che dicono al monitor che volete alterare il contenuto della memoria. Dopo i due punti scrivete i codici esadecimali a due cifre per le istruzioni o i dati (se la prima cifra è zero, basta una sola cifra). Separate ogni codice con uno spazio. Il programma può essere scritto su una riga come segue:



Proprio come in un programma in Basic, si preme il tasto RETURN alla fine di ogni inserimento completo. Quando premete RETURN alla fine della linea, vedrete:



Pronto per nuovi comandi, ma questo è tutto

Questo è tutto quello che c'è da fare. Se non siete fiduciosi che il programma sia stato inserito correttamente, potete esaminare la memoria per verificare quello che avete appena scritto. Per mostrare che cosa c'è in un certo blocco di locazioni di memoria successive, inserite l'indirizzo di partenza e quello finale del blocco separati da un punto.

```

*300:A9 13 8D 25 3 60 ← Avete inserito
*300.305 ■ ← Scrivete questo e premete RETURN
Indirizzo di partenza      Indirizzo finale
                             punto

```

Allora vedrete:

```

*300:A9 13 8D 25 3 60 ← Avete inserito
*300.305 ← Volete esaminare
0300— A9 13 8D 25 03 60 ← Il computer vi mostra
*■ ← Pronto per ulteriori comandi

```

Sì, il programma è lì. Ora, sarà facile eseguirlo! Molto facile! Scrivete:
300 G

```

*300:A9 13 8D 25 3 60 ← Pronto
*300.305
0300— A9 13 8D 25 03 60 ← Set
*300G■ ← VIA!

```

Scrivendo l'indirizzo di partenza seguito dalla lettera G (per GO = VAI), il programma viene eseguito.

```

*300:A9 13 8D 25 3 60
*300.305
0300— A9 13 8D 25 03 60
*300G
*■

```

Bene, che cosa è successo? Oh, non abbiamo chiesto al computer di visualizzare qualcosa, così ha fatto il suo lavoro ed ha memorizzato il valore 43 nella locazione 0325. Come possiamo controllarlo?

Per vedere se 13 è stato effettivamente posto nella locazione 0325, dobbiamo esaminare quella memoria. Per guardare una sola locazione, scrivete semplicemente l'indirizzo della locazione e premete il tasto RETURN.

```

*300:A9 13 8D 25 3 60

*300.305

0300— A9 13 8D 25 03 60
*300G

*325
0325— 13
*■
  
```

Per vedere la locazione di memoria 0325

Sì, eccolo.

Bene, sapete come mettere dei dati in memoria. Questo non è molto utile di per se stesso, ma la maggior parte dei problemi da risolvere con il calcolatore implicano lo spostamento di dati avanti e indietro tra la memoria e l'accumulatore. Vediamo come si lavora su alcuni programmi che servono per degli scopi utili.

L'abbiamo fatto in tutto il libro fino al Cap. 8, senza fare un po' di aritmetica. Chi ha mai sentito parlare di un computer che non è capace di sommare o sottrarre?

Il microprocessore 6502 contiene delle istruzioni che sono in grado di addizionare e sottrarre. Le istruzioni per la moltiplicazione e la divisione *non* sono disponibili, ma il programmatore può creare delle routine per eseguire queste operazioni. Queste routine saranno discusse nel Cap. 12. L'addizione e la sottrazione sono eseguite da due istruzioni che hanno parecchi differenti modi di indirizzamento (immediato, pagina zero, assoluto, ecc.). Useremo per primo il modo immediato.

I codici mnemonici per le istruzioni di addizione e sottrazione sono:

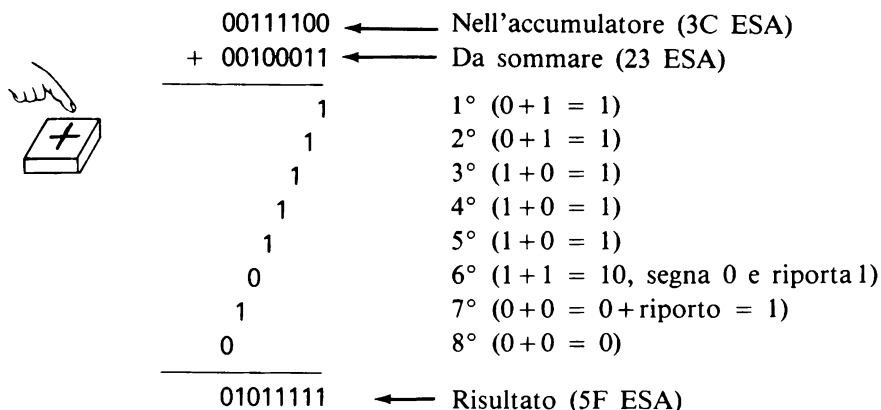
ADC (somma all'accumulatore con riporto)

e

SBC (sottrae dall'accumulatore con prestito)

Prestito è l'inverso di riporto

Ricordate che il 6502 è un microprocessore a 8 bit e può trattare soltanto dati di 8 bit (un byte) alla volta. L'addizione e la sottrazione sono eseguite in maniera binaria, su un bit alla volta.

Esempio

Proviamo il nostro esempio in un breve programma in linguaggio macchina. Queste sono le istruzioni di cui avrete bisogno.

1. 18 CLC (cancella il flag di riporto — o bit di riporto, modo implicito)
 cod. op. cod. mnemonico

Questa è una nuova istruzione che è necessaria per essere sicuri che venga posto zero nel bit di riporto del *registro di stato*. Se capitasse che il bit di riporto vale uno, si otterrebbe una risposta non corretta (troppo grande).

2. A9 LDA (Carica l'accumulatore, modo immediato)
 cod. op. cod. mnemonico

Avete già usato questa istruzione. Essa mette il primo valore da sommare (3C esadecimale = 00111100 binario) nell'accumulatore.

3. 69 ADC (somma all'accumulatore con riporto, modo immediato)
 cod. op. cod. mnemonico

Un'altra istruzione nuova. Essa somma il valore che segue immediatamente l'istruzione (23 esadecimale = 00100011 binario), il contenuto dell'accumulatore (3C esadecimale = 00111100 binario) ed il valore nel bit di riporto nel registro di stato (0 o 1).

Alla fine del programma useremo anche la subroutine del monitor che visualizza il contenuto dell'accumulatore cosicché si possa vedere il risultato dell'addizione.

Questo è il programma.

300 18	CLC	Cancella il flag di riporto
301 A9	LDA 3C	Carica 3C nell'accumulatore

302 3C		
303 69	ADC 23	Somma 22
304 23		
305 20	JSR FDDA	Visualizza l'accumulatore di for-
306 DA		ma esadecimale
307 FD		
308 60	RTS	Ritorno al monitor

Ora siamo pronti a partire.

1 Inserite il programma.

```
*300:18 A9 3C 69 23 20 DA FD 60
```

```
*■
```

2 Eseguite il programma iniziando a 300.

```
*300:18 A9 3C 69 23 20 DA FD 60
```

```
*300G
```

```
5F
```

```
*■
```

← $3C + 23 = 5F$

Provate a sommare qualche altra coppia di numeri esadecimali con il programma per addizioni. I valori esadecimali devono essere sostituiti nel programma originale alle locazioni 302 e 304.

Per cambiare una certa locazione, inserite l'indirizzo seguito da due punti e dal nuovo valore.

```
:
```

```
*300G
```

```
5F
```

```
*302:28 ← Prima modifica
```

```
*304:1E ← Seconda modifica
```

```
*■
```

Ecco alcuni esempi che potete provare.

Esempi e risultati che dovrete ottenere:

- Inserite questi
1. 40 decimale → **28** ESA → 00101000 binario
 30 decimale → **1E** ESA → 00011110 binario
 01000110 binario =
- 46 ESA ← Sarà visualizzato questo
- Inserite questi
2. 120 decimale → **78** ESA → 01111000 binario
 105 decimale → **69** ESA → 01101001 binario
 11100001 binario =
- E1** ESA ← Visualizzato
- Inserite questi
3. 165 decimale → **A5** ESA → 10100101 binario
 47 decimale → **2F** ESA → 00101111 binario
 11010100 binario =
- D4** ESA ← Visualizzato

Una cosa va osservata quando si usa questo programma. La somma dei due valori *deve* essere minore di 256 decimale, altrimenti non ci starà nell'accumulatore. Ricordate, state usando un computer a 8 bit. L'accumulatore e tutte le locazioni di memoria possono contenere soltanto 8 bit di dati. Il più grande numero binario a 8 bit è 11111111, che è FF esadecimale o 255 decimale.

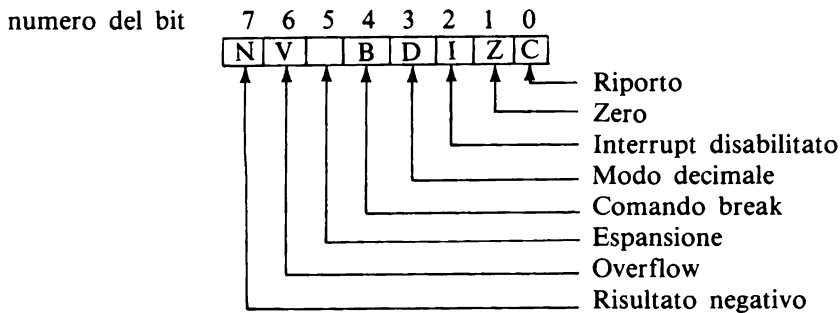
Esempio

Nell'accumulatore	→	11100000 = E0 ESA	
Somma	→	+ 10000001 = 81 ESA	
		<hr/>	
		1 01100001 = 61 ESA	
Bit in più che non ci sta	↖		
		Nell'accumulatore ci stanno solo questi 8 bit (61 ESA)	
			↖ Vero risultato

Per tener conto dei risultati più grandi di FF esadecimale quando si sommano due numeri di 8 bit, il 6502 ha un modo di riportare il bit in più che non entra nell'accumulatore. Usa un *registro* speciale.

IL REGISTRO DI STATO DEL PROCESSORE

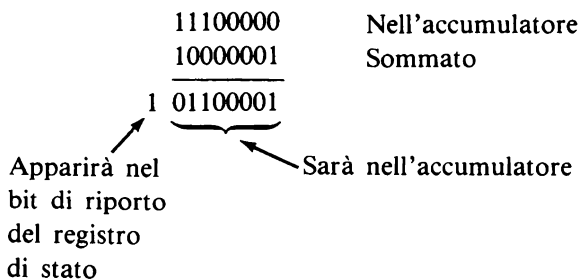
Il microprocessore 6502 ha uno speciale registro chiamato *registro di stato del processore* (brevemente registro di stato) che tiene conto di cose come overflow, riporto, risultato negativo, risultato nullo, ecc. Ad ogni bit di questo registro a 8 bit è assegnata una condizione speciale come mostrato.



È questo registro di stato che determina se i salti vanno effettuati o no quando si usano le seguenti istruzioni:

- BMI (Salta se meno) — il bit N = 1
- BPL (Salta se più) — il bit N = 0
- BEQ (Salta se eguale a zero) — il bit Z = 1
- BNE (Salta se non uguale a zero) — il bit Z = 0

Si noti il bit zero (etichettato C) del registro di stato. Se c'è un riporto quando viene eseguita un'istruzione, questo bit del registro di stato viene posto a uno (1). Nel nostro precedente esempio:



Anche se il bit in più non appare nell'accumulatore, non è stato perso. È nel bit di riporto del registro di stato. Il programmatore deve control-

lare questo bit per vedere se c'è stato un riporto. Il set di istruzioni del 6502 dà un modo per farlo. Infatti, si possono usare due istruzioni.

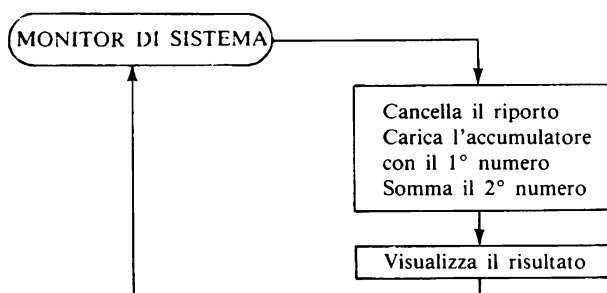
B0 BCS (salta se il riporto è uno)
 ↗
 Codice op.

Quando viene eseguita questa istruzione, *se il bit di riporto è uno*, viene effettuato un salto in avanti e indietro del numero di passi specificato.

90 BCC (salta se il riporto è zero)
 ↗
 Codice op.

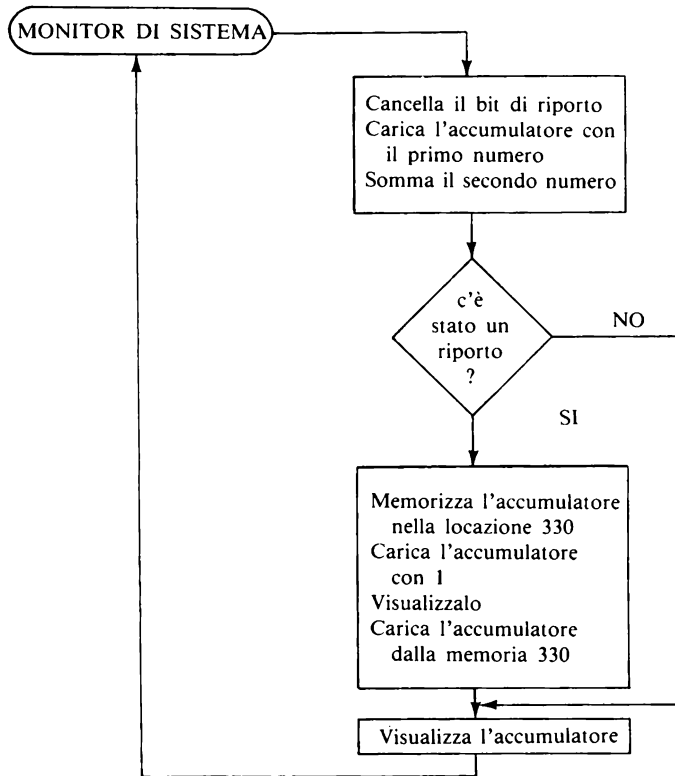
Quando viene eseguita questa istruzione, *se il bit di riporto è zero* (è stato cancellato), viene effettuato un salto in avanti o indietro del numero di passi specificato.

Il nostro primo programma di somma lavorava così:



Per tener conto del riporto creato da una somma maggiore di FF, dobbiamo modificare il diagramma di flusso.

Il programma di somma modificata lavorerà così:



Esistono due possibilità quando si è eseguita l'addizione. L'azione presa dipende dalla presenza o meno del riporto.

1. SENZA RIPORTO

$$\begin{array}{r}
 32 \\
 + 28 \\
 \hline
 5A
 \end{array}$$

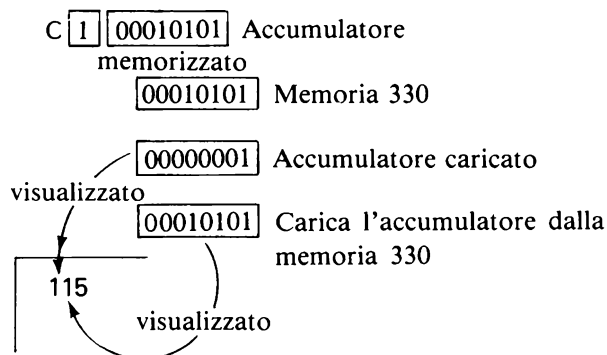
C 0 01011010 Accumulatore

RISPOSTA VISUALIZZATA IMMEDIATAMENTE

5A

2. CON RIPORTO

$$\begin{array}{r}
 32 \\
 + E3 \\
 \hline
 115
 \end{array}$$



Useremo l'istruzione di salto sul riporto uguale a zero per modificare il nostro programma di somma per tener conto di somme maggiori di FF.

PROGRAMMA DI SOMMA MODIFICATO

1. Cancella il bit di riporto

300 18 CLC Cancella il riporto

2. Carica e somma due numeri, poi salta se non c'è riporto

301 A9 LDA E0 Carica il 1° numero
302 E0
303 69 ADC 81 Somma il 2° numero
304 81
305 90 BCC 0B Se non c'è riporto, salta di 11
306 0B passi in avanti

3. Se c'è riporto, memorizza l'accumulatore; carica e visualizza il riporto

307 8D STA 0330 Salva la parte bassa del risultato
308 30
309 03
30A A9 LDA 01 Carica il riporto
30B 01
30C 20 JSR FDDA Visualizza
30D DA
30E FD
30F AD LDA 0330 Ricarica la parte bassa del risultato
310 30
311 03

4. Visualizza l'accumulatore e ritorna al monitor

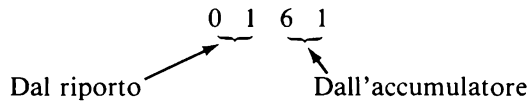
```

312 20 JSR FDDA Visualizza l'accumulatore
313 DA
314 FD
315 60 RTS Ritorna al monitor

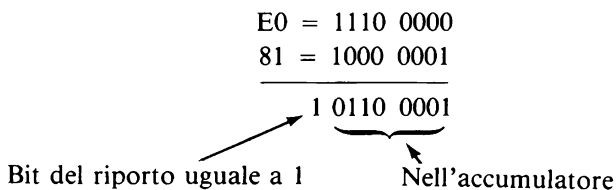
```

Se la somma dei numeri è FF o meno, non si avrà riporto. Perciò, l'accumulatore conterrà il vero risultato. A 305 sarà effettuato un salto, ed il valore nell'accumulatore sarà visualizzato dall'istruzione JSR a 312. (Le linee 307-311 vengono saltate.)

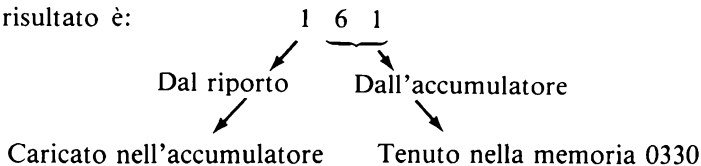
Se la somma è maggiore di FF, il bit di riporto sarà posto uguale a uno. L'accumulatore non contiene il vero risultato. Contiene solo gli 8 bit più bassi. Poiché la subroutine di visualizzazione stampa il contenuto dell'accumulatore, dobbiamo salvare gli 8 bit più bassi cosicché il video mostri prima il bit di riporto. Dopo aver messo la parte bassa in memoria, l'accumulatore viene caricato con uno. Quando l'uno sarà stato visualizzato, l'accumulatore sarà caricato con gli 8 bit bassi e stampato. Il risultato apparirà come un numero completo:



Nel nostro esempio, sappiamo che il risultato sarà maggiore di FF (dal nostro calcolo con carta e matita).



Il vero risultato è:



Ora, poiché sapete il risultato, inserite il programma.

```
*300:18 A9 E0 69 81 90 0B 8D 30 03 A9 01
20 DA FD AD 30 03 20 DA FD 60
```

*■ Premete RETURN dopo aver inserito tutti i codici

Ora eseguite il programma

```
*300:18 A9 E0 69 81 90 0B 8D 30 03 A9 01
20 DA FD AD 30 03 20 DA FD 60
```

```
*300G
```

```
0161
```

```
*■
```

Viene visualizzata la risposta corretta.
Prima 1 dalla JSR a 30C, poi 61 dalla
JSR a 312.

Nel programma è stata usata una nuova istruzione.

a 30F: LDA 0330 Questa indica di caricare l'accumulatore dalla memoria usando il modo di indirizzamento assoluto. Il dato viene caricato dalla locazione di memoria specificata.

SOTTRAZIONE

Diamo ora un'occhiata alla sottrazione. Consideriamo prima come verrebbe eseguita una sottrazione binaria con carta e matita. Ricordate che l'aritmetica binaria è fatta in base due. Così, quando si ha un prestito, si presta una potenza di due invece di una potenza di dieci, come si farebbe se usassimo il sistema decimale. Se un bit è 0, il prestito lo rende 10.

Esempio

Supponiamo di avere il numero binario 110. Guardate i valori di posizione di ogni bit.

```
22 21 20
1   1   0
```

← Questo bit è 0

Se viene fatto un prestito per aumentare il bit 2^0 ,

$$\begin{array}{r} 2^2 \quad 2^1 \quad 2^0 \\ 1 \quad 1 \quad 0 \end{array}$$

Il prestito è fatto dal bit 2^1

Pensate al risultato del prestito così:

$$\begin{array}{r} 2^2 \quad 2^1 \quad 2^0 \\ 1 \quad 1 \quad 10 \end{array}$$

Il prestito è
stato fatto
da qui

Abbiamo prestato un 2^1 , che è uguale a
due (10 binario) 2^0

Esempio di sottrazione:

6 decimale $-$ 3 decimale in binario è:

$$\begin{array}{r} 1 \quad 1 \quad 0 \\ - 0 \quad 1 \quad 1 \\ \hline \end{array}$$

Prestito di un 2^2

$$\begin{array}{r} 1 \quad 1 \quad 10 \\ - 0 \quad 1 \quad 1 \\ \hline 1 \end{array}$$

Due—uno è uno, ma ora dobbiamo presta-
re ancora per la prossima posizione

$$\begin{array}{r} 0 \quad 10 \quad 10 \\ - 0 \quad 1 \quad 1 \\ \hline 0 \quad 1 \quad 1 \end{array}$$

Risultato di $6 - 3 = 3$

È effettivamente proprio come la sottrazione nel sistema decimale, eccet-
to che si prestano potenze di due anziché potenze di dieci. Ecco alcuni
esempi di sottrazioni di numeri binari di 8 bit.

Senza prestito

$$\begin{array}{r} 10101101 = \text{AD ESA} = 173 \text{ decimale} \\ - 00100101 = 24 \text{ ESA} = 37 \text{ decimale} \\ \hline 10001000 = 88 \text{ ESA} = 136 \text{ decimale} \end{array}$$

Con prestito

$$\begin{array}{r}
 00111100 \\
 - 00100011 \\
 \hline
 \end{array}$$

Non possiamo sottrarre 1 da 0, così prestiamo un 4 = (10) due in binario

$$\begin{array}{r}
 001110 \ 10 \ 0 \\
 - 001000 \ 1 \ 1 \\
 \hline
 \end{array}$$

Ora si riporta un 2 = 10 uno
Questo lascia un 2 = 1 due

$$\begin{array}{r}
 001110 \ 1 \ 10 \\
 - 001000 \ 1 \ 1 \\
 \hline
 000110 \ 0 \ 1
 \end{array}$$

$$4C - 23 = 19 \text{ ESA}$$

Dopo tutto questo, dobbiamo ammettere che il calcolatore non fa le sottrazioni in quel modo. Non ha carta e matita. In effetti non sa per niente come sottrarre. Il computer usa un metodo chiamato *addizione in complemento a due*.

Può sembrare strano usare l'addizione per eseguire un'operazione di sottrazione. Tuttavia, facendo così, l'unità aritmetica del microprocessore 6502 necessita soltanto di un sommatore per eseguire tutta la sua aritmetica. Perciò, non deve essere così complicato come se dovesse avere un sommatore ed un sottrattore.

Per comprendere che cosa si intende per *complemento a due*, guardate i seguenti esempi di complemento a uno binario.

1. Un numero binario $\longrightarrow 10011100$
 Il suo complemento a uno $\longrightarrow 01100011 \longleftarrow$ Tutti gli 1 diventano 0.
 Tutti gli 0 diventano 1.
2. Un numero binario $\longrightarrow 01010101$
 Il suo complemento a uno $\longrightarrow 10101010$

Il complemento a uno è ottenuto cambiando ogni bit del numero (un uno o uno zero) nel suo opposto (o complemento). Tutti gli 1 sono cambiati in 0, e tutti gli 0 in 1.

Per ottenere il complemento a due di un numero binario, dobbiamo soltanto sommare uno (1) al complemento a uno del numero, come mostrato nei seguenti esempi.

1. Il numero	→	10011101
Complemento a uno	→	01100010
Somma uno		+ 1
Complemento a due	→	<u>01100011</u>
2. Il numero	→	10011100
Complemento a uno	→	01100011
Somma uno		+ 1
Complemento a due	→	<u>01100100</u>
3. Il numero	→	01010101
Complemento a uno	→	10101010
Somma uno		+ 1
Complemento a due	→	<u>10101011</u>

Vediamo ora come il calcolatore usa il complemento a due in un problema di sottrazione. Confrontate con la nostra sottrazione con carta e matita di prima.

Esempio

$$\begin{array}{r}
 1. \quad \quad \quad 10101101 \\
 - 00100101 \leftarrow \text{Questo numero viene cambiato nel} \\
 \hline
 11011011 \quad \leftarrow \text{suo complemento a due}
 \end{array}$$

Poi il complemento a due viene sommato al numero originale.

$$\begin{array}{r}
 10101101 \\
 + 11011011 \\
 \hline
 1 \quad 10001000
 \end{array}$$

Ignora il bit in più 88 ESA o 136 decimale

$$\begin{array}{r}
 2. \quad \quad \quad 00111100 \\
 - 00100011 \\
 \hline
 \end{array}$$

Complemento a due $11011100 + 1 \rightarrow$ $\begin{array}{r} = 00111100 \\ + 11011101 \\ \hline 00011001 \end{array}$ ESA o 25 decimale

Ignora il bit in più

Se il metodo del computer per la sottrazione vi sconcerta, rilassatevi! Non dovete preoccuparvi sul come fa la sottrazione. Potete controllare il risultato con il vecchio metodo carta e matita.

Tentiamo una sottrazione in un programma in linguaggio macchina. Cambiando due istruzioni nel nostro programma iniziale per l'addizione, avremo:

SOTTRAZIONE DI DUE NUMERI

1 COMMENTO **BIT DI RIPORTO UGUALE A UNO*

300 38 SEC Mette a uno il bit del riporto

2 COMMENTO *CARICA E SOTTRAE DUE NUMERI*

301 A9 LDA 3C (Carica il primo numero)
302 3C

304 23

3 COMMENTO *VISUALIZZA IL RISULTATO E RITORNA AL MONITOR*

305 20 JSR FDDA (Visualizza il risultato)
306 DA
307 FD

308 60 RTS (Ritorna al monitor)

Nella parte 1, l'istruzione SEC è usata per porre uno (1) nel bit di riporto del registro di stato. Nella sottrazione, il bit di riporto viene usato per ottenere il complemento a due del numero da sottrarre. Perciò, SEC (*SEt Carry flag*, metti uno nel flag di riporto) è data *prima* dell'esecuzione della sottrazione. Il codice op. di SEC è 38. L'istruzione è usata soltanto nel modo implicito.

Nella parte 2, il primo numero è caricato nell'accumulatore. Poi viene usata l'istruzione SBC (*SuBtraCt with borrow*, sottrazione con prestito) nel modo immediato (codice op. E9). Il complemento a due del numero che segue immediatamente l'istruzione è sommato al numero nell'accumulatore. Il risultato della sottrazione (addizione in complemento a due) è lasciato nell'accumulatore.

La parte 3, come prima, visualizza il risultato e ritorna al monitor di sistema.

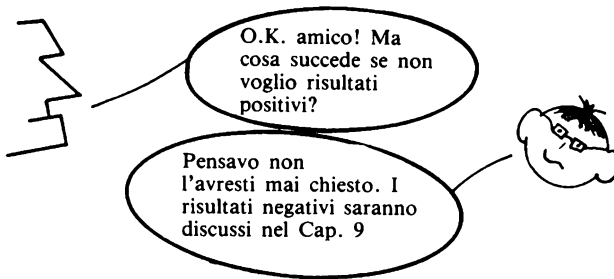
Inserite il programma con il monitor.

```
*300:38 A9 3C E9 23 20 DA FD 60
*■
```

Poi eseguite il programma

```
*300:38 A9 3C E9 23 20 DA FD 60
*300G
19 ← Risulta 19 (ESA), proprio come con carta e matita
*■
```

Provate altri numeri alle locazioni 302 e 304. Assicuratevi che il numero usato a 302 sia maggiore di quello che volete sottrarre (a 304), se volete risultati positivi.



Abbiamo ristretto la nostra discussione ai numeri positivi che possono essere espressi in 8 bit (un byte). Questi numeri sono minori di 256 (100 esadecimale). Nel prossimo capitolo prenderemo in considerazione i numeri negativi ed i numeri positivi che possono essere più grandi di 255.

SOMMARIO

È stato introdotto il monitor di sistema assieme ai seguenti simboli e comandi.

*	Prompt del monitor di sistema
*300:A9	Modifica una singola locazione di memoria

*300:A9 13 8D 25 3 60	Modifica locazioni successive. Usato nell'inserimento di nuovi programmi.
*300	Esamina una singola locazione
*300.305	Esamina le locazioni da 300 a 305 (esadecimale)
*300G	Esegue il programma in linguaggio macchina che inizia alla locazione 300 (esadecimale)

Avete anche imparato ad inserire un programma che somma 2 numeri esadecimali di un byte e visualizza il risultato. Il bit di riporto è stato usato per individuare i risultati maggiori di quelli che possono essere contenuti in un singolo byte.

È stata spiegata la sottrazione per mezzo dell'addizione in complemento a due.

Sono stati discussi i singoli bit del registro di stato del processore.

N	V		B	D	I	Z	C
---	---	--	---	---	---	---	---

Sono state introdotte parecchie nuove istruzioni.

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
CLC	Implicito	18	1	Cancella il flag di riporto
ADC	Immediato	69	2	Somma il dato immediato all'accumulatore
BCS	Relativo	B0	2	Salta se il riporto è uno
BCC	Relativo	90	2	Salta se il riporto è zero
LDA	Absolute	AD	3	Carica l'accumulatore dalla memoria specificata
SEC	Implicito	38	1	Poni il flag di riporto uguale a uno

ESERCIZI

1. Scrivete i codici mnemonici delle istruzioni di somma con riporto e di sottrazione con prestito.
_____ e _____
2. L'istruzione di addizione con riporto somma un numero al contenuto dell'accumulatore. Quale altro valore somma?

3. Se i valori esadecimali 35 e 6A vengono sommati dal programma Somma due numeri, che valore sarà visualizzato?

4. Se i valori esadecimali 85 e 9A vengono sommati dal programma Somma due numeri, che valore sarà visualizzato?

5. Se il programma di Somma modificato viene usato per sommare i numeri esadecimali 85 e 9A, che valore verrà memorizzato in: 0330_____
6. Nell'Esercizio 5, che cosa sarà visualizzato dopo l'esecuzione del programma?

7. Qual è il complemento a uno di 10110011? _____
8. Qual è il complemento a due di 01101101? _____
9. Se il programma Sottrazione di due numeri viene usato per sottrarre 23 (esadecimale) da AF (esadecimale), che cosa sarà visualizzato dopo l'esecuzione del programma?

10. Qual è il risultato di A3-2F (valori esadecimali)?

RISPOSTE AGLI ESERCIZI

1. ADC e SBC
2. Il valore nel bit di riporto
3. 9F (ESA)
4. 1F (ESA) (Il risultato effettivo è 11F, ma il bit di riporto non è visualizzato).
5. 0330 = 1F
6. 011F (ESA)
7. 01001100
8. 10010011
9. 8C (ESA)
10. 74 (ESA)

Precisione multipla e numeri negativi

MS

Per poter trattare numeri grandi, è necessario lavorare con valori contenuti in più byte. In questo paragrafo prenderemo in considerazione numeri di due byte. Se ci limitiamo ai numeri positivi, due byte possono rappresentare numeri grandi quanto:

$$\begin{array}{rcl}
 \underbrace{11111111}_{\text{Byte più significativo}} & \underbrace{11111111}_{\text{Byte meno significativo}} & = \text{FF FF ESA} \\
 \\
 \text{FF FF ESA} & = & (15 \times 4096) + (15 \times 256) + (15 \times 16) + 15 \\
 & = & \begin{array}{r} 61440 \\ 3840 \\ 240 \\ + 15 \\ \hline 65,535 \end{array}
 \end{array}$$

decimale

Massimo numero decimale per due byte

Si possono ottenere valori maggiori estendendo il numero di byte come desiderato.

ADDIZIONE A DUE BYTE



Due byte sono meglio di uno

Un'addizione con carta e matita di numeri di due byte ci aiuterà a decidere come scrivere un programma per eseguire l'operazione sul computer. Supponiamo di voler sommare questi due numeri esadecimali di due byte.

Byte più significativo

Byte meno significativo

$$\begin{array}{r} 55A4 = 01010101 \ 01000100 \\ + \ 3CB3 = 00111100 \ 10110011 \\ \hline \end{array}$$

L'addizione binaria per byte \longrightarrow

$$\begin{array}{r} 01010101 \ 01000100 \\ + \ 00111100 \ 10110011 \\ \hline \end{array} \longleftarrow \text{Prima il BMS}$$

Nei bit di riporto

1

Byte meno significativo (BMS) del risultato

Poi il BPS

$$\begin{array}{r} 01010101 \ 01000100 \\ + \ 00111100 \ 10110011 \\ \hline + \quad \quad 1 \ 01010111 \\ \hline \end{array} \longleftarrow \text{Dal BMS}$$

10010010

Byte più significativo (BPS) del risultato

Il risultato finale = $\begin{matrix} & \text{BPS} & & \text{BMS} & \\ 10010010 & 01010111 & & & \\ \uparrow & \nearrow & \uparrow & \nearrow & \\ 9 & 2 & 5 & 7 & \text{ESA} \end{matrix}$

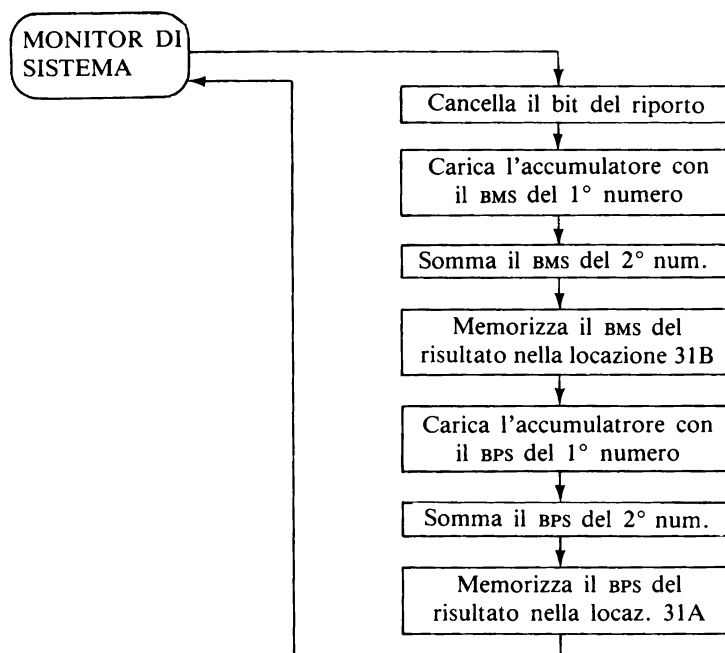
Notate, in questo esempio si ha un riporto dall'addizione dei byte meno significativi. L'istruzione ADC (*ADd with Carry*, somma con riporto)

sommerà automaticamente questo bit di riporto alla somma dei byte più significativi. Perciò, sembra che i numeri di due byte vengano sommati con una addizione usuale.

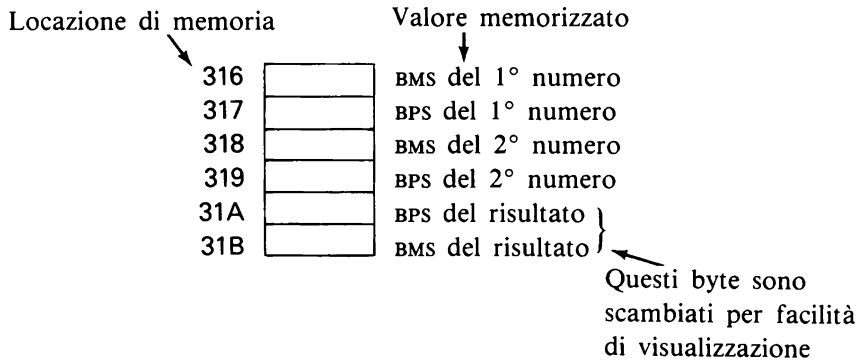
Prima i byte meno significativi e poi i byte più significativi.

Se disegniamo un diagramma di flusso delle operazioni che devono essere eseguite, ci aiuterà a scrivere il programma passo per passo.

DIAGRAMMA DI FLUSSO



Sembra un programma molto lineare. Fissiamo un blocco di locazioni per memorizzare i byte che devono essere sommati e quelli del risultato dell'addizione. Poiché ci sono 2 byte per ogni numero, abbiamo bisogno di 6 byte.



Seguendo il diagramma di flusso si può scrivere il programma.

ADDIZIONE A DUE BYTE

1 COMMENTO *CANCELLA IL BIT DI RIPORTO*

```
300 18  CLC
```

2 COMMENTO *CARICA, SOMMA E MEMORIZZA I BMS*

```
301 AD  LDA,0316 ← Carica il BMS del 1° numero
302 16
303 03
```

```
304 6D  ADC,0318 ← Somma il BMS del 2° numero
305 18
306 03
```

```
307 8D  STA,031B ← Memorizza il risultato del BMS
308 1B
309 03
```

3 COMMENTO *CARICA, SOMMA E MEMORIZZA I BPS E POI RETURN*

```
30A AD  LDA,0317 ← Carica il BPS del 1° numero
30B 17
30C 03
```

```
30D 6D  ADC,0319 ← Somma il BPS del 2° numero
30E 19
30F 03
```

```

310 8D  STA,031A ← Memorizza il risultato del bps
311 1A
312 03

```

```

313 60  RTS      ← Ritorna al monitor

```

4 COMMENTO *DATI*

```

314 00  BRK      Riempito
315 00  BRK      Riempito
316 A4                ← BMS del 1° numero
317 55                ← BPS del 1° numero
318 B3                ← BMS del 2° numero
319 3C                ← BPS del 2° numero

```

Notate, questa volta abbiamo usato le istruzioni LDA e ADC nel modo assoluto. Il codice op. per LDA, quando viene usata in questo modo, è AD. Il codice op. è seguito dal byte meno significativo dell'indirizzo che contiene il dato da caricare. Il byte più significativo di questo indirizzo segue come terzo byte dell'istruzione.

```

AD  ← Codice op. per LDA (modo assoluto)
16  ← Byte meno significativo dell'indirizzo
03  ← Byte, più significativo dell'indirizzo

```

Il codice per ADC nel modo assoluto è 6D. Anche qui, il codice op è seguito dall'indirizzo che contiene il numero da sommare. Quindi viene dato il byte meno significativo dell'indirizzo, e poi il byte più significativo.

```

6D  ← Codice op. per ADC (modo assoluto)
18  ← Byte meno significativo dell'indirizzo
03  ← Byte più significativo dell'indirizzo

```

Nella nostra addizione con carta e matita abbiamo trovato il risultato 9257 esadecimale. Ora facciamolo fare al computer. Prima inserite i programma.

```

*300:18 AD 16 03 6D 18 03 8D 1B 03 AD 17
03 6D 19 03 8D 1A 03 60 00 00 A4 55 B3
3C

```

*■

← Tutto inserito

Poi eseguitelo.

```

*300:18 AD 16 03 6D 18 03 8D 1B 03 AD 17
03 6D 19 03 8D 1A 03 60 00 00 A4 55 B3
3C

*300G ←————— RUN

*■

```

Adesso esaminiamo le locazioni 31A e 31B per vedere i risultati.

```

*300:18 AD 16 03 6D 18 03 8D 1B 03 AD 17
03 6D 19 03 8D 1A 03 60 00 00 A4 55 B3
3C

*300G

*31A.31B

031A—92 57 ←———— Eccolo, lo stesso che con carta e matita
*■

```

SOTTRAZIONE A DUE BYTE

La sottrazione di numeri di due byte è eseguita in modo simile. Il programma Somma due numeri può essere modificato con tre semplici cambiamenti.

a 300 cambiate 18 (CLC) con 38 (SEC) ← Bit del riporto uguale a uno

a 304 } cambiate 6D (ADC) con ED (SBC) ← Sottrazione con prestito
 e }
a 30 D }

La modifica a 300 pone a uno il bit di riporto in preparazione della sottrazione proprio come è stato fatto nel programma per la sottrazione ad un byte. L'istruzione di somma con riporto (ADC) viene sostituita da quella di sottrazione con prestito (SBC). L'istruzione di sottrazione è

usata nel modo assoluto con l'indirizzo che contiene il numero da sottrarre subito dopo il codice op.

ED ← Codice op. SBC (modo assoluto)
 18 ← Byte meno significativo dell'indirizzo
 03 ← Byte più significativo dell'indirizzo

Così, il byte meno significativo del valore da sottrarre è contenuto nell'indirizzo 0318 esadecimale. Il byte più significativo è invece contenuto nell'indirizzo 0319 esadecimale. Esso è sottratto dal byte più significativo del primo valore dall'istruzione:

ED ← Codice op. SBC (modo assoluto)
 19 ← Byte meno significativo dell'indirizzo
 03 ← Byte più significativo dell'indirizzo

Avete due scelte.

(a) Se avete ancora in memoria il programma di addizione a due byte, fate i tre cambiamenti.

```
*300:38
*304:ED
*30D:ED
*■
```

(b) Se non avete il programma in memoria, inserite il nuovo programma con le modifiche.

```
*300:38 AD 16 03 ED 18 03 8D 1B 03 AD 17
03 ED 19 03 8D 1A 03 60 00 00 A4 55 B3
3C
*■
```

Poi eseguite il programma.

```
*300G
*■
```


Ed esaminate le locazioni che contengono il risultato.

```

.
.
.
.
*300G

*31A.31B

0314— 18 F1 ← Risposta
*■

```

$$\begin{array}{r}
 55A4 = 0101\ 0101\ 1010\ 0100 \\
 - 3CB3 = 0011\ 1100\ 1011\ 0011 \\
 \hline
 0001\ 1000\ 1111\ 0001 \text{ binario}
 \end{array}$$

$$= 1\ 8\ F\ 1\ \text{ESA} \leftarrow \text{Sì, corrisponde!}$$

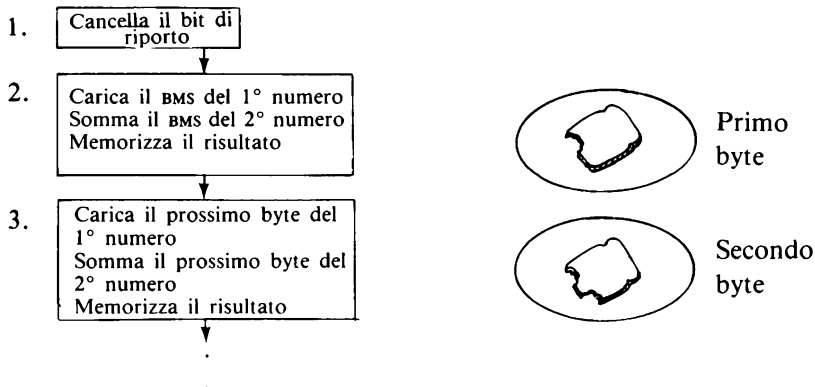
$$= (1 \times 16^3) + (8 \times 16^2) + (15 \times 16) + 1$$

$$= 4096 + 2048 + 240 + 1$$

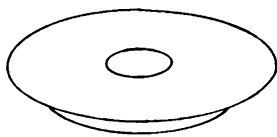
$$= 6385 \text{ decimale}$$

Per sommare o sottrarre numeri che richiedono più di due byte, si può estendere la procedura relativa ai due byte. L'operazione viene sempre eseguita dal byte meno significativo in avanti (da destra a sinistra).

DIAGRAMMA DI FLUSSO PER L'ADDIZIONE A PIÙ BYTE



4. Ripeti il Passo 3 finché tutti i byte sono stati memorizzati.



5.

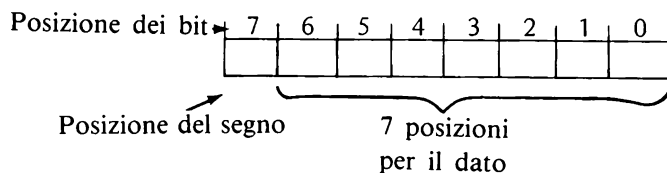
Carica il BPS del 1° numero
Somma il BPS del 2° numero
Memorizza il risultato
6.

Ritorna al monitor

NUMERI NEGATIVI

È possibile considerare il modo in cui i dati sono rappresentati nel computer in maniera differente. Se si devono rappresentare numeri con segno (quelli che sono positivi, negativi o nulli), il computer deve avere un modo per distinguerli.

Consideriamo un dato di 8 bit come composto da un bit del segno e sette bit di dato



(a) Se la posizione del segno contiene zero (0) il dato è considerato un numero positivo.

Esempio

$$\begin{aligned}
 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1 &= +127\ (64+32+16+8+4+2+1) \\
 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0 &= +126\ (64+32+16+8+4+2\) \\
 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1 &= +125\ (64+32+16+8+4\ +1) \\
 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 &= +124\ (64+32+16+8+4\)
 \end{aligned}$$

. . .
.
.

.	.	.
.	.	.
.	.	.
0 0 0 0 0 0 1 1	= +3 (+2 +1)
0 0 0 0 0 0 1 0	= +2 (+2)
0 0 0 0 0 0 0 1	= +1 (+1)
0 0 0 0 0 0 0 0	= +0 ()

Lo zero è considerato un numero
positivo dalle istruzioni di salto

(b) Se la posizione del segno contiene un uno (1), il dato è considerato un numero negativo.

Esempio

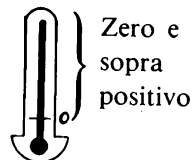
1 0 0 0 0 0 0 0	= -128
1 0 0 0 0 0 0 1	= -127
1 0 0 0 0 0 1 0	= -126
1 0 0 0 0 0 1 1	= -125
1 0 0 0 0 1 0 0	= -124

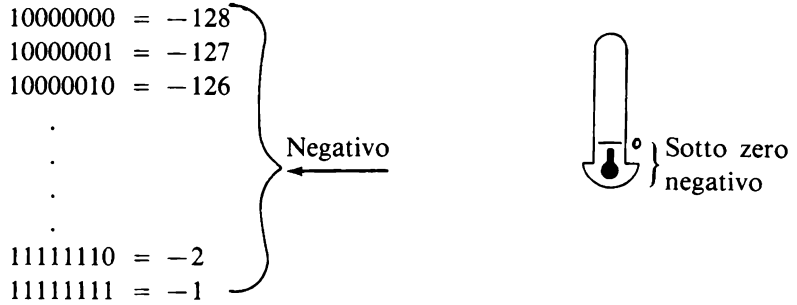
1 1 1 1 1 0 1 1	= -5
1 1 1 1 1 1 0 0	= -4
1 1 1 1 1 1 0 1	= -3
1 1 1 1 1 1 1 0	= -2
1 1 1 1 1 1 1 1	= -1

Abbiamo imparato ad interpretare numeri positivi sia binari che decimali, ma cosa si può dire dei numeri negativi? Essi non ci appaiono per niente familiari. Tuttavia, vogliamo vedere che ogni codice a 8 bit può rappresentare tutti gli interi da -128 a +127.

00000000	= +0
00000001	= +1
.	
.	
.	
.	
01111110	= +126
01111111	= +127

Positivo





Date un'occhiata ai negativi e guardate se c'è una relazione significativa con le loro controparti positive.

Considerate il numero positivo 126 = 01111110.

Il suo complemento a uno = 10000001

Il suo complemento a due = 10000010

Confrontate quest'ultimo con -126 = 10000010

La rappresentazione binaria di un numero negativo (da -1 a -127) è uguale al complemento a due della sua controparte positiva.

-127 = il complemento a due di +127

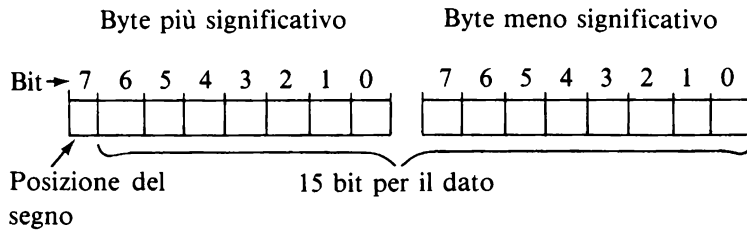
-126 = il complemento a due di +126

-125 = il complemento a due di +125

-2 = il complemento a due di +2

-1 = il complemento a due di +1

Per numeri di due byte, si considera come posizione del segno il *bit più significativo* del *byte più significativo*.



Una discussione completa dell'aritmetica dei numeri con segno va oltre gli scopi di questo libro. Per approfondire questo argomento consultate il *MOS Technology Programming Manual*.

Per i nostri scopi, dobbiamo renderci conto che certe istruzioni di salto controllano i numeri per vedere se sono positivi o negativi. Questa determinazione dipende dal flag negativo del registro di stato del processore, se vale 0 o 1. Il flag negativo viene posto a 1 quando il computer interpreta i risultati di certe istruzioni come numeri negativi (un 1 nel bit 7). Metteremo in risalto questo fatto nelle attività ricreative che seguono.

IL GIOCO "INDOVINA IL NUMERO"

Probabilmente avete visto questo gioco molte volte su libri e riviste. Solitamente è pubblicato in Basic. Noi mostreremo una versione in linguaggio macchina che utilizza le istruzioni di salto basate sull'interpretazione dei numeri con segno. Questa interpretazione dipende dal *flag negativo* (o bit) del registro di stato del processore.

BMI	(Salta sul risultato meno)	Salta se il flag negativo è uguale a 1 (risultato negativo)
	Codice op. = 30	
	Modo di indirizzamento relativo	
BPL	(Salta sul risultato più)	Salta se il flag negativo è uguale a 0 (risultato positivo)
	Codice op. = 10	
	Modo di indirizzamento relativo	

Useremo anche l'istruzione di salto sul risultato uguale, che fa uso del flag zero nel registro di stato del processore.

BEQ	(Salta sul risultato uguale)	Salta se il flag zero è uguale a 1 (risultato uguale a zero)
	Codice op. = F0	
	Modo di indirizzamento relativo	

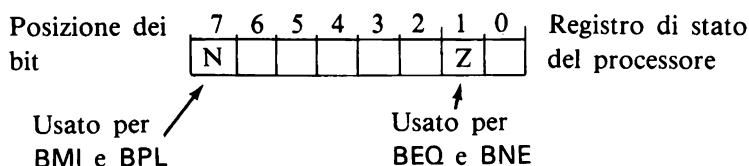
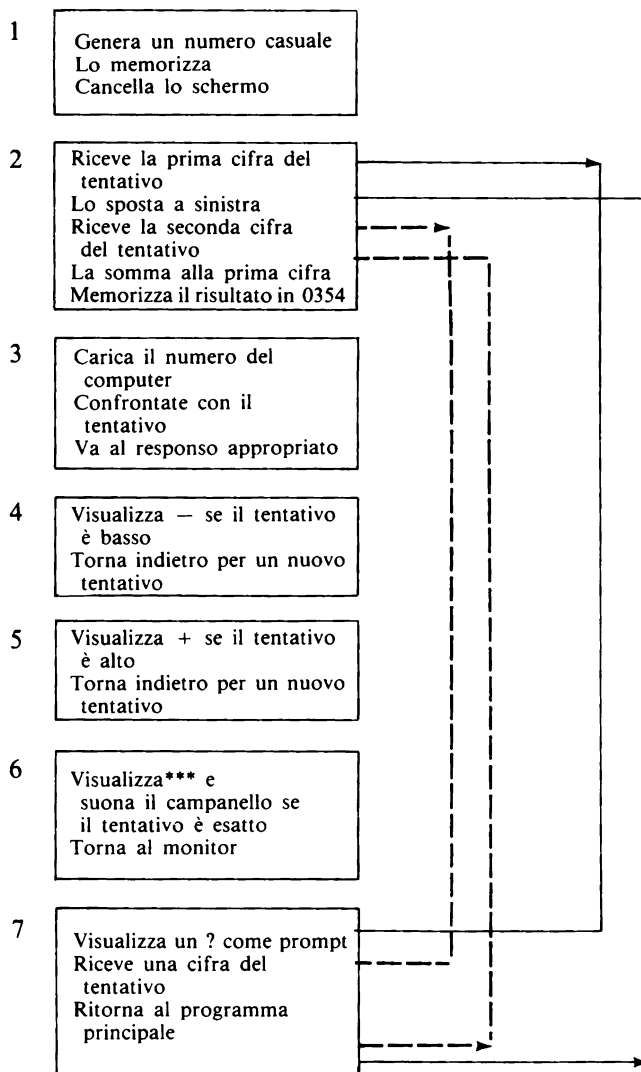


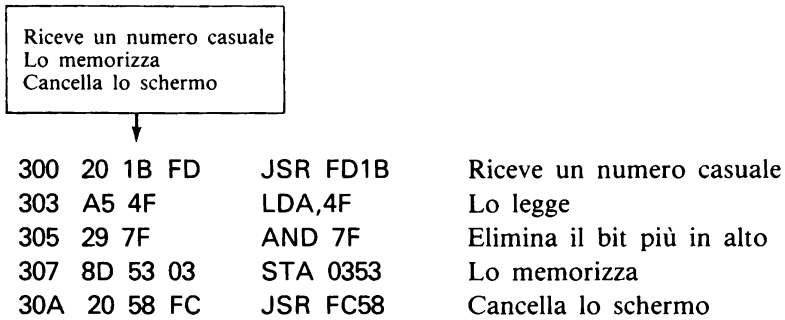
DIAGRAMMA DI FLUSSO DEL PROGRAMMA INDOVINA IL NUMERO



Il programma usa molte utili subroutine del monitor di sistema. Senza queste subroutine, il programma avrebbe richiesto parecchie pagine per essere listato e sarebbe diventato piuttosto complesso. Dovreste studiare nei particolari l'*Apple II Reference Manual* per poter sfruttare queste subroutine ogniqualvolta sia possibile.

Esaminare dettagliatamente il programma. Se capite come lavora, sarete in grado di modificarlo a vostro piacere. Abbiamo dato il programma fondamentale, ma voi potete fare numerose aggiunte.

Prima parte



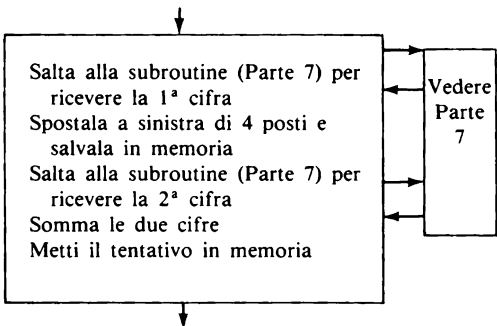
Prima si fa uso della subroutine KEYIN a FD1B. Essa legge la tastiera ed aspetta la pressione di un tasto. Quando viene premuto un tasto, nella locazione 004F viene posto un numero casuale. Il codice del carattere inserito è messo nell'accumulatore, ma il nostro programma lo ignora. Siamo interessati soltanto al numero casuale che è stato generato. Questo numero viene caricato nell'accumulatore da 004F. Poi viene eseguito l'AND con 7F. Questo restringe il valore casuale ad un numero esadecimale positivo tra 0 e 7F. Ricordate, il computer interpreta i numeri da 80 a FF come negativi. Non considereremo questi numeri. Se includessimo i numeri negativi, il programma diventerebbe molto più complicato. Vi ricordate l'istruzione AND del Cap. 7? Questa volta la stiamo usando per "togliere" il bit più alto del numero casuale per essere sicuri che il computer non dia un numero negativo.

Esempio: Supponiamo che il numero casuale sia F3 (negativo).

$$\begin{array}{rcl}
 \text{F3} & = & \begin{array}{r} 1111 \ 0011 \\ 0111 \ 1111 \\ \hline 0111 \ 0011 \end{array} \quad \begin{array}{l} \text{AND con } 7\text{F} \\ \\ \text{73 un numero positivo} \end{array}
 \end{array}$$

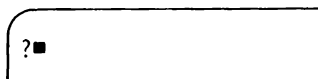
Il numero casuale viene quindi memorizzato nella locazione 0353. Lo richiameremo più avanti per confrontarlo con il nostro tentativo. Poi viene cancellato lo schermo, con la routine del monitor a FC58, in preparazione della parte di programma relativa al tentativo.

Seconda parte



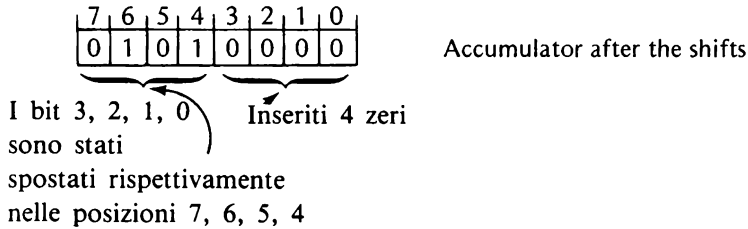
30D 20 55 03	JSR 0355	Riceve la prima cifra del tentativo
310 0A 0A 0A	ASL	Sposta a sinistra di quattro posizioni
313 0A		
314 8D 54 03	STA 0354	Memorizza la prima cifra
317 20 5A 03	JSR 035A	Riceve la seconda cifra del tentativo
31A 18	CLC	
31B 6D 54 03	ADC 0354	Combina le 2 cifre
31E 8D 54 03	STA 0354	Memorizzale nuovamente

A 30D viene chiamata una subroutine (vedere parte 7) che visualizza un punto interrogativo come prompt per farvi sapere che il computer sta aspettando il vostro tentativo (un numero esadecimale a due cifre minore di 20).



La subroutine permette l'inserimento di una cifra del vostro tentativo di due cifre, modifica il suo codice ASCII, e poi ritorna a 310, dove il valore è spostato a sinistra di 4 posizioni.
Supponiamo che abbiate inserito un 5:

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	1	Accumulatore prima dello spostamento.
					5			



Questo risultato viene memorizzato nella locazione 0354 come 50 (esadecimale). A 317, la subroutine (parte 7) è chiamata ancora per ricevere la seconda cifra del tentativo. La subroutine modifica il codice ASCII di quella cifra e ritorna al programma principale a 31A.

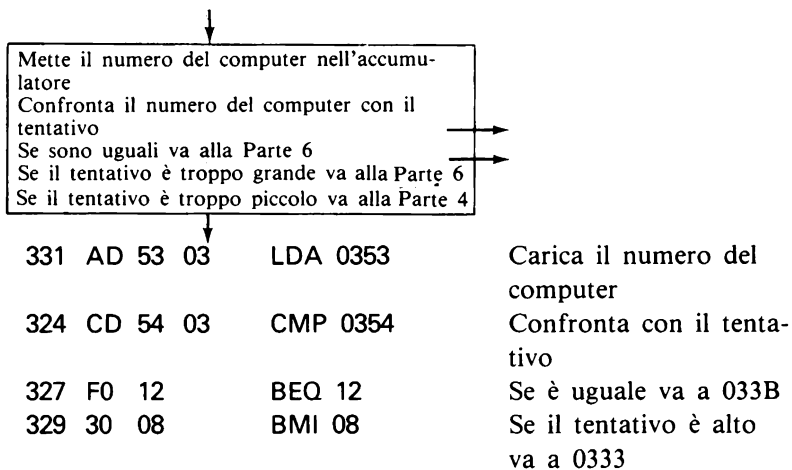
Ora la prima cifra è sommata alla seconda.

Supponiamo che la vostra seconda cifra sia 7.

0 0 0 0 0 1 1 1	Accumulatore (seconda cifra)
	+
0 1 0 1 0 0 0 0	Dalla memoria 0354
	=
0 1 0 1 0 1 1 1	Il tentativo a due cifre

Il tentativo è poi memorizzato nella locazione 0354.

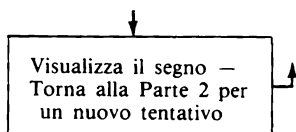
Terza parte



Il numero del calcolatore ed il vostro vengono confrontati. Se sono uguali, l'istruzione BEQ a 327 manda il programma a 338 (parte 6) per visualizzare il messaggio di vittoria. Se il tentativo è troppo alto, il con-

fronto (computer-tentativo) è negativo, e l'istruzione BMI a 329 manda il programma a 333 (parte 5) per visualizzare un segno + e ritornare per un nuovo tentativo. Se il tentativo è troppo basso, il programma procede alla parte 4.

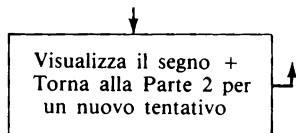
Quarta parte



32B A9 AD	LDA #AD	Carica (—)
32D 20 ED FD	JSR FDED	Visualizza
330 4C 0D 03	JMP 030D	Torna indietro per un nuovo tentativo

Viene visualizzato il segno meno per indicare che il tentativo è troppo basso. Il programma poi torna indietro per ricevere la prima cifra del nuovo tentativo.

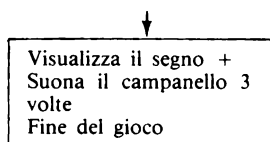
Quinta parte



333 A9 AB	LDA #AB	Carica (+)
335 20 ED FD	JSR FDED	Visualizza
338 4C 0D 03	JMP 030D	Torna indietro per un nuovo tentativo

Viene visualizzato il segno più per indicare che il tentativo è troppo alto. Il programma poi torna indietro per ricevere la prima cifra del nuovo tentativo.

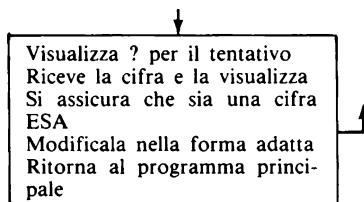
Sesta parte



33B 20 8E FD	JSR FD8E	Ritorno del carrello
33E A9 AA	LDA AA	Carica (*)
340 20 ED FD	JSR FDED	Visualizzalo 3 volte
343 20 ED FD	JSR FDED	
346 20 ED FD	JSR FDED	
349 20 3A FF	JSR FF3A	Suona il campanello 3 volte
34C 20 3A FF	JSR FF3A	
34F 20 3A FF	JSR FF3A	
352 60	RTS	Ritorna al monitor
353 00 00		Posto per i dati

Questo è il messaggio di vittoria. Vengono visualizzati tre asterischi seguiti da tre suoni del campanello (subroutine del monitor a FF3A). Il controllo è ritornato al monitor. Le locazioni 353 e 354 sono usate per memorizzare i numeri (casuale e del tentativo).

Settima parte




Questa subroutine viene usata dopo che è stato scelto il numero casuale nella parte 1. È visualizzato un punto interrogativo per avvisarvi di fare il tentativo. Dopo essere stata inserita, viene visualizzata la prima cifra. Poi la subroutine esegue una serie di test per assicurarsi che la cifra ha un codice ASCII compreso tra B0 e B9 per le cifre decimali 0,1,2,3,4,5,6,7,8,9, o tra C1 e C6 per le lettere A,B,C,D,E,F. Questo permette l'entrata di ognuna delle cifre esadecimali.

355 A9 BF	LDA #BF	Carica (?) come prompt per il tentativo
357 20 ED FD	JSR FDED	Lo visualizza
35A 20 35 FD	JSR FD35	Riceve una cifra
35D 20 ED FD	JSR FDED	La visualizza
360 C9 B0	CMP #B0	Controlla se è una cifra ESA valida


362	30	15	BMI 15	Se è bassa va a 379
364	C9	BA	CMP #BA	
366	30	0E	BMI 0E	Se è fra B0 e B9 va a 376
368	C9	C1	CMP #C1	
36A	30	0D	BMI 0D	Se è fra B9 e C1 va a 379
36C	C9	C7	CMP #C7	
36E	10	09	BPL 09	Se è troppo alta va a 379
370	29	0F	AND #0F	Elimina i bit in alto (C)
372	18		CLC	
373	69	09	ADC #09	Cambia ad (A-F)
375	60		RTS	Ritorna al programma principale
376	29	0F	AND #0F	Elimina i bit in alto (B)
378	60		RTS	Ritorna al programma principale
379	20	3A FF	JSR FF3A	Suona il campanello di input errato
37C	20	8E FD	JSR FD8E	Ritorno del carrello
37F	4C	55 03	JMP 0355	Da un altro prompt

Se viene battuto un tasto fuori dal range accettabile, suona il campanello ed il computer visualizza un nuovo punto interrogativo come prompt. Se il tasto ha un codice ASCII compreso tra B0 e B9, l'istruzione BMI a 366 manda il calcolatore a 376, dove la B del codice ASCII viene eliminata facendo un AND con 0F. Poi il computer ritorna al programma principale con il valore dell'accumulatore sistemato. Se il tasto ha codice ASCII compreso tra C1 e C6, l'istruzione AND a 370 elimina la C. Per ottenere una cifra esadecimale compresa tra A e F viene sommato nove al valore rimasto. Poi il calcolatore ritorna al programma principale.

Ora siamo pronti per inserire il programma. State molto attenti! Questo è un programma lungo. Non abbiate fretta. Inserirlo una parte alla volta e dopo aver inserito una parte controllatela prima di proseguire.

*300:20 1B FD A5 4F 29 7F 8D 53 03 02 58
 FC 20 55 03 0A 0A 0A 0A 8D 54 03 20 5A
 03 18 6D 54 03 8D 54 03 AD 53 03 CD 54 0
 3 F0  Inserite

*300.327

0300— 20 1B FD A5 4F 29 7F 8D  Controllate
 0308— 53 03 20 58 FC 20 55 03
 0310— 0A 0A 0A 0A 8D 54 03 20
 0318— 5A 03 18 6D 54 03 8D 54
 0320— 03 AD 53 03 CD 54 03 F0

*■

Poi

*328:12 30 08 A9 AD 20 ED FD 4C 0D 03 A9
 AB 20 ED FD 4C 0D 03 20 8E FD A9 AA 20
 ED FD 20 ED FD 20 ED

*328.347

0328— 12 30 08 A9 AD 20 ED FD
 0330— 4C 0D 03 A9 AB 20 ED FD
 0338— 4C 0D 03 20 8E FD A9 AA
 0340— 20 ED FD 20 ED FD 20 ED

*■

Poi

*348:FD 20 3A FF 20 3A FF 20 3A FF 60 00
 00 A9 BF 20 ED FD 20 35 FD 20 ED FD C9
 B0 30 15 C9 BA 30 0E

*348.367

0348— FD 20 3A FF 20 3A FF 20
 0350— 3A FF 60 00 00 A9 BF 20
 0358— ED FD 20 35 FD 20 ED FD
 0360— C9 B0 30 15 C9 BA 30 0E

*■

Ed infine

```
*368:C9 C1 30 0D C9 C7 10 09 29 0F 18 69
    09 60 29 0F 60 20 3A FF 20 8E FD 4C 55
03

*368.381

0368— C9 C1 30 0D C9 C7 10 09
0370— 29 0F 18 69 09 60 29 0F
0378— 60 20 3A FF 20 8E FD 4C
0380— 55 03

*■
```

Controllate quello che avete inserito. Assicuratevi che alcuni di quegli zeri non siano lettere O maiuscole. Prendete coraggio e provate ad eseguirlo. Ecco una delle nostre tipiche esecuzioni (dopo aver scoperto e corretto tutti i nostri errori di battuta).

```
.
.
9 0F 60 20 3A FF 20 8E FD 4C 55 03

*300G
```

Quando premete il tasto return, il cursore scompare e sembra che non succeda nulla. Il computer sta generando un numero casuale. Quando premete un tasto qualsiasi, il processo terminerà ed il calcolatore avrà un numero casuale. Quindi... premete un tasto.

1. PREMERE UN TASTO QUALSIASI.

```
?■
```

← Ora ha il numero casuale e vuole un tentativo

Abbiamo scritto 40

```
?40+?■
```

← Pronto per un nuovo tentativo
← Il tentativo è troppo alto

Abbiamo scritto 20

$$\overbrace{?40 + ?20 - ?\blacksquare} \quad \swarrow \text{Il tentativo è troppo basso}$$

Abbiamo dato 30

$$\overbrace{?40 + ?20 - ?30 + ?\blacksquare} \quad \swarrow \text{Il tentativo è troppo alto}$$

Abbiamo scritto 28

$$\overbrace{?40 + ?20 - ?30 + ?28 + ?\blacksquare} \quad \swarrow \text{Ancora troppo alto}$$

Abbiamo dato 24

$$\overbrace{?30 + ?20 - ?30 + ?28 + ?24} \\ \begin{array}{l} *** \leftarrow \\ * \blacksquare \end{array} \quad \begin{array}{l} \text{Tre squilli di campanello e tre} \\ \text{asterischi sullo schermo. Il numero} \\ \text{era 24!} \end{array}$$



Ding
Ding
Ding

In 5 tentativi!

Riproviamo per vedere che cosa succede se inseriamo dei caratteri non esadecimali.

Scrivete 300G per riprovare con un nuovo numero.

$$\overbrace{?40 + ?20 - ?30 + ?28 + ?24} \\ \begin{array}{l} *** \\ *300G \end{array}$$

Premete un tasto qualsiasi e verrà scelto un altro numero casuale. Lo schermo è cancellato e appare ?

$$\overbrace{?\blacksquare} \quad \swarrow \text{Pronto per un nuovo tentativo}$$

Ecco i risultati di input non esadecimali.

?N
?■ ← La N non è stata accettata. Appare un altro?.

?N
?1V
?■ ← L'1 è stato accettato, ma non la V

?N
?1V
?4-?■ ← Il 4 è stato accettato. Il segno - indica che il valore 14 è troppo basso.

?N
?1V
?4-?40--?■ ← Anche 40 è troppo basso

?N
?1V
?4-?40-?60-?■ ← Anche 60 è troppo basso

?N
?1V
?4-?40-?60-?70+?■ ← 70 è troppo alto

?N
?1V
?4-?40-?60-?70+?68
*** ← Ancora il campanello e gli asterischi.
*■ Il numero è 68.

Vediamo in memoria se i due numeri sono proprio uguali.


```

.
.
.
***
*353.354

0353— 68 68 ← Si. sono uguali.
*■

```

SOMMARIO

In questo capitolo avete imparato che si può eseguire un'addizione a due byte formattando i numeri grandi in due byte separati. Lo stesso vale per la sottrazione di numeri a due byte. Questa procedura può essere estesa all'aritmetica a più byte.

Sono stati visti programmi dimostrativi per addizioni e sottrazioni a due byte. Avete avuto la possibilità di esercitarvi esaminando e modificando il contenuto della memoria per mezzo del monitor di sistema.

Avete imparato che il formato per numeri con segno è:

Numeri a due byte:

posizione dei bit

7	6	5	4	3	2	1	0

Bit del segno

Bit del dato

Numero ad un byte:

posizione dei bit

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Bit del segno

Bit del dato

Siete anche in grado di trasformare i numeri binari negativi negli equivalenti decimali usando il sistema del complemento a due.

È stato spiegato il gioco indovina il numero, che utilizza delle decisioni basate sui numeri con segno.

Nuove istruzioni usate

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Byte usati</i>	<i>Funzione</i>
ADC	Assoluto	6D	3	Somma il contenuto della memoria all'accumulatore
SBC	Assoluto	ED	3	Sottrae il contenuto della memoria dall'accumulatore
LDA	Pagina zero	A5	2	Carica l'accumulatore dalla memoria di pagina zero (00XX)
CMP	Assoluto	CD	3	Confronta il contenuto dell'accumulatore con il contenuto della memoria

Nuove subroutine usate

<i>Funzione</i>	<i>Localione in memoria</i>
KEYIN per ricevere un numero casuale	FD1B
CROUT per eseguire un ritorno di cursore	FD8E
BELL1 per suonare il camapanello	FF3A

ESERCIZI

1. Qual è la rappresentazione esadecimale a quattro cifre del seguente valore binario senza segno a due byte?

$$\begin{array}{|c|c|} \hline 01101111 & 10100011 \\ \hline \end{array} = \text{_____ (ESA)}$$

byte più significativo
byte meno significativo

2. Il programma Addizione a due byte somma 55A4 e 3CB3 esadecimali. Mostrate come modificare il programma per cambiare questi due valori in 5A45 e 3B3C esadecimali.

* _____

3. Se il programma Addizione a due byte venisse eseguito con le modifiche dell'Esercizio 2, che risultato sarebbe visualizzato con il seguente comando?

*300G

*31A.31B

4. Nella sottrazione a due byte, su che byte si opera prima? Sul byte _____ significativo.
(più, meno)

5. Sottraete 03C5 da 25A2 (esadecimali). Usate un programma, l'addizione in complemento a due, oppure il metodo con carta e matita.

6. Qual è il bit usato per distinguere i numeri con segno?

7. Qual è l'interpretazione decimale del numero con segno 10111010?

8. Se il computer sceglie come numero esadecimale casuale 6D, e voi, alla richiesta del tentativo, inserite 40, che cosa sarà visualizzato?

?40 _____

9. Se nel gioco Indovina il numero viene generato il numero casuale 9E nella locazione 300, che numero sarà memorizzato nella locazione 353? _____

10. Se la prima cifra di un tentativo nel gioco Indovina il numero è 7, e la seconda cifra non è stata scelta, che valore verrà memorizzato nella locazione 354? (Suggerimento: vedere le parti 2 e 7 del programma.) _____

RISPOSTE AGLI ESERCIZI

1. 6F A3

2.

*316:5A 45 3B 3C

o

*316:5A

*317:45

*318:3B

*319:3C

3.

031A—95 81

4. Meno

5. 21DD

6. Il bit 7 del byte più significativo

7. -70

8.

?40—?

9. 1E

(l'AND di 9E con 7F è 1E

1001 1110

0111 1111

0001 1110 = 1E)

10. 70 (il valore è stato spostato a sinistra di 4 posizioni)

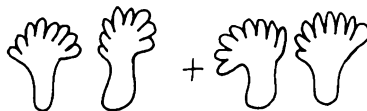
Ancora sul monitor

MS

Abbiamo usato carta e matita per controllare i risultati di addizioni e sottrazioni in esadecimale. I computer sono fatti per eseguire le parti ripetitive e noiose di un lavoro, e l'aritmetica esadecimale è sicuramente un compito noioso. È anche facilmente soggetta ad errori.

Il monitor esegue delle semplici addizioni e sottrazioni esadecimali di numeri di due cifre. Dovete soltanto inserire i valori separati dal simbolo dell'operazione. Non è necessario scrivere un programma per fare l'operazione.

ADDIZIONE ESADECIMALE - MODO IMMEDIATO



Ecco il metodo usato. È quasi come usare una calcolatrice.

```

*3C + 2F  ← Voi scrivete questo
= 6B      ← Il computer risponde
*■

```

```

*59 + C    ← Voi scrivete questo
= 65       ← Il computer risponde
*■

```

Potreste osservare che i nostri esempi forniscono un risultato minore di FF. Che cosa pensate succederà se il risultato è maggiore di FF? Provatelo.

```

*FF + 13
= 12 ← Il risultato è minore degli addendi. Perché?
*■

```

Guardate l'addizione binaria di FF e 13 e capirete perché.

```

FF =  1 1 1 1  1 1 1 1
13 =  0 0 0 1  0 0 1 1
-----
      1 0 0 0 1  0 0 1 0

```

Un bit di troppo per l'accumulatore

Questo è il risultato visualizzato

La vera risposta è 112

Questa caratteristica del monitor è ristretta alla visualizzazione di risultati di valore non maggiore di FF. Tuttavia, potete calcolare somme migliori di FF se tenete conto del bit in più che deve essere riportato alla prossima posizione.

Potete anche eseguire somme di numeri a più byte, sempre tenendo conto del bit di riporto.

Esempio Sommare 35D e 2F5

Prima separare i byte: 3 5D
 2 F5

Secondo sommare i byte più bassi:

$$\begin{array}{l} *5D + F5 \\ = 52 \end{array} \leftarrow \begin{array}{l} \text{Parte bassa del risultato, ma c'è un riporto} \\ \text{*■ non dimenticatelo.} \end{array}$$

Terzo, sommare i byte più alti ed il riporto.

$$\begin{array}{l} *3 + 1 \\ = 05 \leftarrow \text{Parte alta parziale del risultato} \\ *5 + 1 \leftarrow \text{Sommare il riporto} \\ = 06 \leftarrow \text{Parte alta finale del risultato} \\ *■ \end{array}$$

Perciò il risultato finale è: $35D + 1F5 = 652$

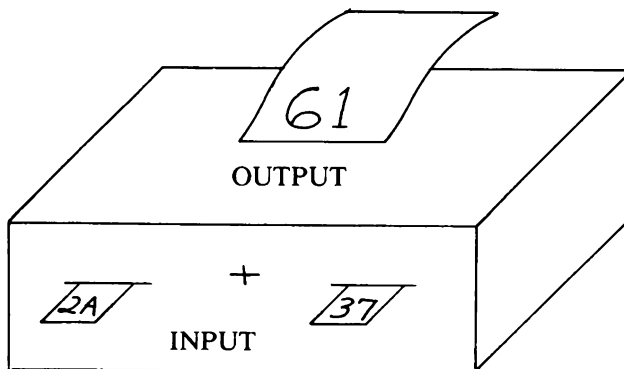
Il terzo passo deve essere fatto in due parti, poiché l'addizione e la sottrazione immediate operano solo su due valori. Se tentate di sommare tre valori, il secondo sarà ignorato. Verranno sommati soltanto il primo ed il terzo.

Esempi

$$\begin{array}{l} *3 + 2 + 1 \\ = 4 \end{array} \leftarrow \text{Dato da } 3 + 1$$

$$\begin{array}{l} *1F + 2F + 1 \\ = 20 \end{array} \leftarrow \text{Dato da } 1F + 1$$

$$\begin{array}{l} *2A + 11 + 37 \\ = 61 \end{array} \leftarrow \text{Dato da } 2A + 37$$



Ecco alcuni altri esempi:

$$7F30 + 25C8$$

*30 + C8	
= F8	← Parte bassa, nessun riporto
*1F + 25	
= 44	← Parte alta, nessun riporto
*■	

Perciò, $1F30 + 25C8 = 44F8$

$$1F33 + 2FF5$$

*33 + F5	
= 28	← Parte bassa, riporto 1
*1F + 2F	
= 4E	← Parte alta parziale
*4E + 1	← Somma del riporto
= 4F	← Parte alta finale
*■	Il risultato è 4F28

È possibile eccedere un risultato di due byte, come si vede in quest'ultimo esempio.

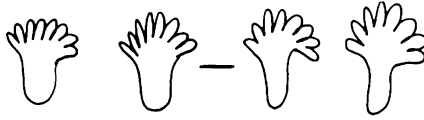
$$D14F + E213$$

*4F + 13	
= 62	← Parte bassa, nessun riporto
*D1 + E2	
= B3	← Parte alta, ma c'è un riporto.
*■	

La vera risposta è: 1B362

Se pensate di utilizzare questa possibilità del monitor con risultati a più di un byte (massimo FF), dovete essere in grado di stimare i risultati per sapere quando c'è riporto.

SOTTRAZIONE ESADECIMALE



Si può eseguire la sottrazione con precauzioni simili. Ecco alcuni esempi.

$$3D - 28$$

```

*3D-28 ← Voi scrivete
= 15 ← Il computer risponde
*
    
```

$$A4 - 2D$$

```

*A4-2D ← Voi scrivete
= 77 ← Il computer risponde
*
    
```

Sembra facile, ma che cosa succede se tentate di sottrarre un numero da uno più piccolo?

$$22 - 24 = ??$$

```

*22-24
= FE ← Ecco che cosa dice.
*
    
```

Se avete ancora qualche ricordo di algebra, saprete che si ottiene un valore negativo quando si sottraggono numeri positivi da altri più piccoli. FE può essere negativo? Sì. Riguardate la tavola dei numeri negativi nell'Appendice B. FE è equivalente al numero con segno -2 .

Potete usare la sottrazione per creare una tavola di valori negativi per futuri riferimenti. Per esempio,

$$*0-1$$

e

$$*0-4$$

Se volete, potete omettere lo zero. Provate ad inserire:

$$*-1$$

$$= FF$$

e

$$*-4$$

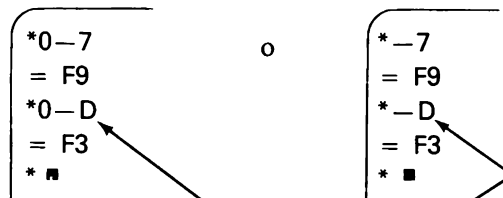
$$= FC$$

Ora create una tavola di numeri negativi.

*-0 = 00 *-1 = FF *-2 = FE *-3 = FD *-4 = FC *-5 = FB . . . *-7C = 84 *-7D = 83 *-7E = 82 *-7F = 81 *	<i>Tavola dei numeri negativi</i> FF = -1 FE = -2 FD = -3 FC = -4 FB = -5 . . . 84 = -7C 83 = -7D 82 = -7E 81 = -7F
--	---

La sottrazione può risultare molto utile nel calcolare l'operando usato nelle istruzioni di salto. Ritornate a "Descrizione del programma" nel Cap. 7, e controllate le istruzioni di salto ivi utilizzate.

A 776 e 777 del programma Suona la tua melodia, abbiamo usato F9 per un salto all'indietro di -7. A 782 e 783, per saltare indietro di -13, è stato usato F3. Controlliamo questi valori.



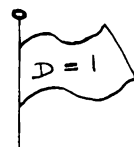
Ricordate, D è l'equivalente
esadecimale di -13

<i>BCD</i>	<i>DECIMALE</i>
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

L'istruzione che richiede al computer di eseguire addizioni o sottrazioni decimali è:

SED (*SEt Decimal mode*, abilita il modo decimale)
 Codice op. = F8
 Modo di indirizzamento implicito
 Lunga un byte
 Flag di stato interessati - D

Questa istruzione pone a 1 il flag decimale del registro di stato del processore. Una volta usata questa istruzione, tutte le istruzioni di somma e sottrazione saranno eseguite come operazioni decimali a causa dello stato del flag decimale. Nessun'altra operazione di altre istruzioni viene influenzata. Se l'istruzione SED è stata eseguita in un programma e si vuole eseguire un'addizione o una sottrazione binaria, il computer deve eseguire l'istruzione di disabilitazione del modo decimale.



CLD (*CLear Decimal mode*, disabilita il modo decimale)
 Codice op. = D8
 Modo di indirizzamento implicito
 Lunga un byte
 Flag di stato interessati - D

Questa istruzione azzerà il flag decimale del registro di stato del processore.

Supponiamo di voler sommare i numeri decimali 18 e 23. Dobbiamo usare questo programma.

SOMMA DI DUE NUMERI DECIMALI

300 F8	SED	Abilita il modo decimale
301 18	CLC	Azzera il bit di riporto
302 A9	LDA23	Carica 23 nell'accumulatore
303 23		
304 69	ADC 18	Somma 18
305 18		
306 20	JSR FDDA	Visualizza il risultato come due cifre ESA
307 DA		
308 FD		
309 60	RT	Ritorna al monitor

Per inserire il programma usando il monitor, scrivete:

300:F8 18 A9 23 69 18 20 DA FD 60

dopo l'asterisco di prompt. Poi eseguite il programma scrivendo 300G.

```
*300:F8 18 A9 23 69 18 20 DA FD 60
*300G
41 ←
*
```

Ecco la nostra risposta

Il risultato decimale di $18 + 23$ è 41. Se avessimo sommato i valori esadecimali di 18 e 23, il risultato sarebbe stato 3B. Ricordate, 18 e 23 in notazione esadecimale sono valori differenti dai corrispondenti in notazione decimale. Perciò, 41 decimale e 3B esadecimale *non* sono valori equivalenti.

Ora eseguiamo il programma partendo dalla locazione 301 (omettendo l'istruzione SED) ed osserviamo il risultato.

```

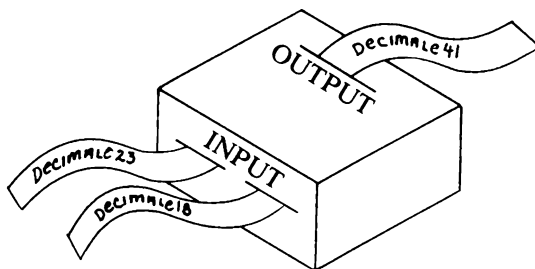
*300:F8 18 A9 23 69 18 20 DA FD 60

*300G
41 ← Prima risposta
*301G
3B ← Seconda risposta
*■

```

Una volta eseguita l'istruzione SED, tutte le addizioni nel programma sono eseguite in modo decimale. Per tornare all'addizione binaria durante l'esecuzione del programma, bisogna usare l'istruzione CLD (disabilita il modo decimale)

Quando il computer è ritornato al monitor dopo la prima esecuzione del programma sopra, il monitor ha eseguito automaticamente la propria istruzione CLD. Perciò, quando nella seconda esecuzione abbiamo saltato l'istruzione SED, il computer ha trattato i due numeri (18 e 23) come valori eadecimali. Il risultato è stato 3B. L'istruzione SED deve essere eseguita in ogni programma dove volete usare l'aritmetica decimale. Il monitor, quando viene utilizzato, pone automaticamente a 0 (il modo aritmetico binario) il bit decimale del registro di stato del processore. Il vantaggio dell'addizione decimale è che vi evita di dover convertire numeri da una base all'altra per interpretarli. Potete inserire valori decimali (come 23 e 18) ed ottenere risultati decimali.



Provate il programma Somma di due numeri decimali con altre coppie di valori decimali. Sostituiteli al posto dei valori contenuti nelle locazioni 303 e 305.

Esempio

```

*303:47 ← Modifica del primo numero
*305:35 ← Modifica del secondo numero
*300G ← Esecuzione
82 ← Risultato
*
```

Ancora una precauzione: assicuratevi che la somma dei due numeri sia minore di 100. Ogni risultato il cui valore è maggiore di 99 non potrà stare nell'accumulatore, né in un calcolatore a 8 bit. Ecco cosa succederà se insistete negli esperimenti (noi segretamente vi incoraggiamo).

```

*303:84
*305:29
*300G
13 ← Se i miei conti sono giusti, questo dovrebbe
    * essere 113
```

Ancora

```

*303:99
*305:1
*300G
00 ← Tutti sanno che 99 + 1 fa 100. È andata
    * persa la posizione delle centinaia
```


Riprendiamo carta e matita per vedere che cosa è successo al nostro risultato.

$$\begin{array}{rcl}
 84 & = & 1000\ 0100 \text{ in DCB} \\
 + 19 & = & 0010\ 1001 \text{ in DCB} \\
 \hline
 113 & = & 1\ 0001\ 0011 \text{ in DCB}
 \end{array}$$

Bit in più → Le ultime due cifre nell'accumulatore sono corrette
 → La somma decimale dà 13

Poiché il microprocessore 6502 tratta dati della grandezza di 8 bit, si devono prendere dei provvedimenti per i casi che possono dare risultati più grandi. Questa tecnica è stata spiegata nel Cap. 8. Mostreremo un'addizione decimale a due byte più avanti in questo capitolo. Diamo prima un'occhiata alla sottrazione decimale.

Useremo un'interessante capacità del monitor per spostare il nostro programma per l'addizione in una nuova zona della memoria, cosicché sia ancora disponibile se ne avremo bisogno più avanti.

<i>Indirizzo di memoria</i>	<i>Dato</i>	
300	F8	
301	18	
302	A9	
303	23	
304	69	
305	18	
306	20	
307	DA	
308	FD	
309	60	
.	.	
330	??	
339	??	

Copieremo i dati da 300-309 nelle locazioni 330-339

Poi modificheremo il programma originale per l'addizione per trasformarlo in uno per la sottrazione.

1. Spostamento del programma.

Si deve indicare al monitor qual è la parte di memoria da spostare (nel nostro caso, da 300 a 309). Si deve anche specificare a che locazione si vuole spostare il programma (330 nel nostro caso). Il formato di questo comando è:

330, ← destinazione
 per noi di minore
 300, ← inizio
 per noi decimale
 309, ← fine
 per noi
 M, per MOVE (sposta)

Sul video appare così:


```

*330<300.309M
  ↑      ↑      ↑      ↑
Destinazione Inizio Fine  sposta
    
```

Per vedere come lavora, prima esaminiamo il programma originale.

```

*300.309
0300— F8 18 A9 23 69 18 20 DA
0308— FD 60
*■
    
```




Poi facciamo lo spostamento

```

*300.309
0300— F8 18 A9 23 69 18 20 DA
0308— FD 60
*330<300.309M
*■
    
```

← Spostalo



Poi per essere sicuri che lo spostamento è stato eseguito, esaminiamo entrambe le aree della memoria.

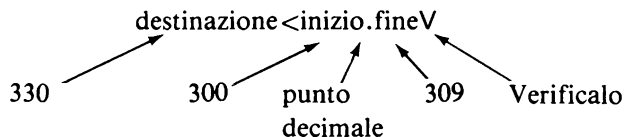
```

*300.309
0300— F8 18 A9 23 69 18 20 DA
0308— FD 60
*300<300.309M

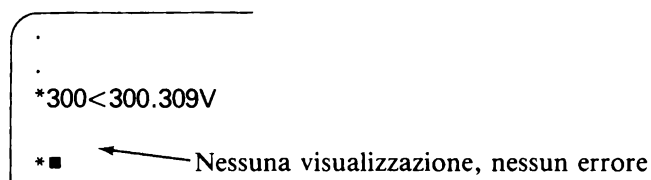
*330.339
0330— F8 18 A9 23 69 18 20 DA   Esaminiamo 330-339
0338— FD 60
*300.309
0300— F8 18 A9 23 69 18 20 DA   Esaminiamo 300-309
0308— FD 60
*■
    
```

} Entrambi uguali

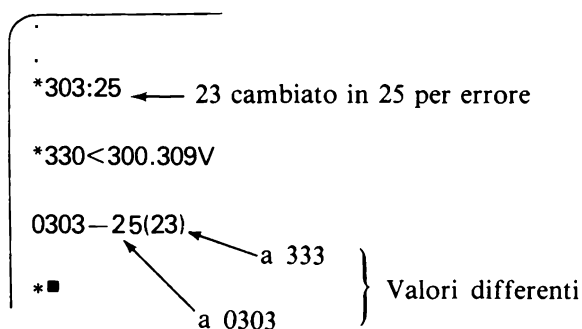
Un modo più semplice per farlo è di utilizzare un altro comando del monitor. Possiamo confrontare due aree della memoria e verificare che sono la stessa con il comando **VERIFY** (verifica). Per usare il comando di verifica, bisogna indicare al monitor la parte interessata e la destinazione. Il formato usato è:



Il monitor confronta la zona specificata con quella che inizia all'indirizzo della destinazione. Se ci sono delle discrepanze, viene visualizzato l'indirizzo dove è stata trovata la differenza assieme ai due valori corrispondenti. Se non ci sono differenze, non viene visualizzato niente.



Modifichiamo una locazione per vedere come vengono segnalate le differenze trovate.

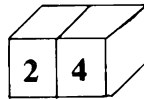
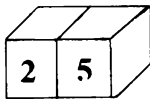


L'errore deve poi essere corretto (nella versione errata) e riverificato.

```

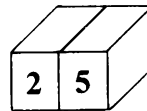
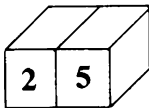
.
.
*303:23 ← Modifica
*330<300.309V ← Verifica
*■ ← Nessuna differenza, tutto O.K.

```



UGUALI?

NO



ED ORA?

SI

2. Modifica del programma originale.

Ora modificheremo il programma originale in uno per la sottrazione. Dobbiamo cambiare CLC (18) alla locazione 301 in SEC (38) e ADC (69) alla locazione 304 in SBC (E9).

```

.
.
*301:38 ← Cambiate 18 in 38
*304:E9 ← Cambiate 69 in E9
*■

```

Ora esaminate il programma modificato.

```

.
.
*301:38

*304:E9

*300.309
0300— F8 38 A9 23 E9 18 20 DA
0308— FD 60
*■

```

Le modifiche sono state fatte

Eseguite il programma per la sottrazione

```

.
.
*300G
05
*■

```

$23 - 18 = 5$ quando si lavora con numeri decimali

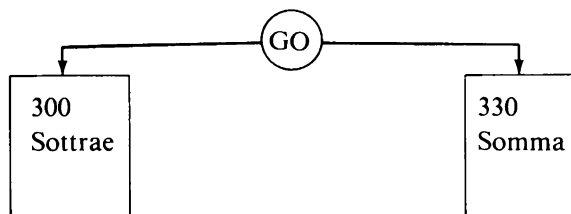
Se volete eseguire il programma per l'addizione,

```

.
.
*300G
05 ←  $23 - 18 = 5$ 
*330G
41 ←  $23 + 18 = 41$ 
*■

```

Entrambi i programmi sono in memoria. Potete scegliere quale eseguire.



 ESAMINARE E MODIFICARE I REGISTRI

Avete visto come esaminare e modificare la memoria. E i registri? Si possono esaminare e cambiare? Sì, potete fare anche questo.

Per esaminare un registro, tenete premuto il tasto CTRL. Contemporaneamente, premete il tasto E. Poi rilasciateli entrambi. Premete il tasto RETURN ed i registri verranno visualizzati sullo schermo. I contenuti dei registri A, X, Y, P, S saranno visualizzati in questo ordine da sinistra a destra. Vi ricordate che cosa significano queste lettere?

A	Accumulatore
X	Registro indice X
Y	Registro indice X
P	Registro di stato del processore
S	Registro puntatore di stack

I valori che vediamo subito dopo aver acceso l'Apple ed aver premuto CTRL E assieme seguiti da RETURN sono:

```

*          ← CTRL E non è visualizzato
A = FF X = FF Y = FF P = 00 S = FF
*■
  
```

Dopo aver esaminato i registri, i loro contenuti possono essere cambiati (in ordine da sinistra a destra) scrivendo due punti (:) seguiti dai nuovi valori. Provate questi:

```

*
A = FF X = FF Y = FF P = 00 S = FF
*:88 FF 80 33 40
*■
  A X Y P S
          ← Cambiamenti
  
```

Per vedere se sono stati effettivamente modificati, premete CTRL E per esaminarli ancora.

```

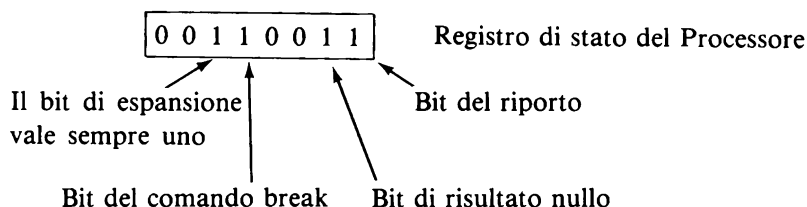
A = FF X = FF Y = FF P = 00 S = FF ← Inizialmente
*:88 FF 80 33 40 ← Le modifiche

* ← Viene premuto CTRL E

A = 88 X = FF Y = 80 P = 33 S = 40
*■ ← Le modifiche sono state fatte

```

Una delle modifiche riguardava il registro di stato del processore. Il suo nuovo valore è 33. Spezzandolo nei singoli bit di stato, abbiamo:



Notate che abbiamo posto a uno il bit del riporto. Se ora venisse esaminata un'addizione, il bit del riporto sarebbe incluso a causa dell'istruzione ADC. Proviamo.

```

*
A = 88 X = FF Y = 80 P = 33 S = 40
*300:A9 33 69 22 20 DA FD 60 ← Programma per sommare
                                33 e 22

*300G ← Esecuzione del programma
56 ← Il riporto è stato incluso
    (33 + 22 + riporto = 56)
*■

```

Ora esaminiamo di nuovo i registri ed azzeriamo il bit del riporto modificando il registro P (registro di stato del processore). Poi sommeremo ancora due numeri.

```

*300G
56
*
      ← Esaminiamo i registri
A = 88 X = FF Y = 80 P = 33 S = 40
*:88 FF 80 32 40
      ← Il registro P contiene 32
*300G      ← Nuova esecuzione
55      ← Il risultato è 55 poiché il
*■      bit di riporto valeva 0
          (33 + 22 + 0 = 55)

```

Questa volta caricheremo 33 nell'accumulatore, trasferiremo il risultato nel registro X, sommeremo 22 all'accumulatore e visualizzeremo il risultato. Poi daremo un'altra occhiata ai registri per vedere come sono cambiati.

```

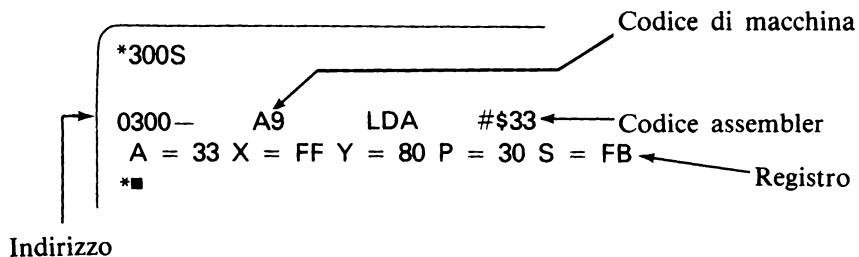
Il programma:  300 A9    LDA 33
                301 33
                302 AA    TAX          (trasferisce l'accumulato-
                                         re in X)
                303 69    ADC 22
                304 22
                305 20    JSR FDDA      Visualizza il risultato
                306 DA
                307 FD
                308 60    RTS          Ritorna al monitor

```

Uno dei migliori metodi per scoprire errori nei programmi è la caratteristica *single-step* (un passo alla volta) del monitor. Il comando *single-step* decodifica, visualizza ed esegue un'istruzione alla volta. L'istruzione viene visualizzata sia in codice macchina che in assembler (il mini-assembler sarà discusso nel Cap. 11).

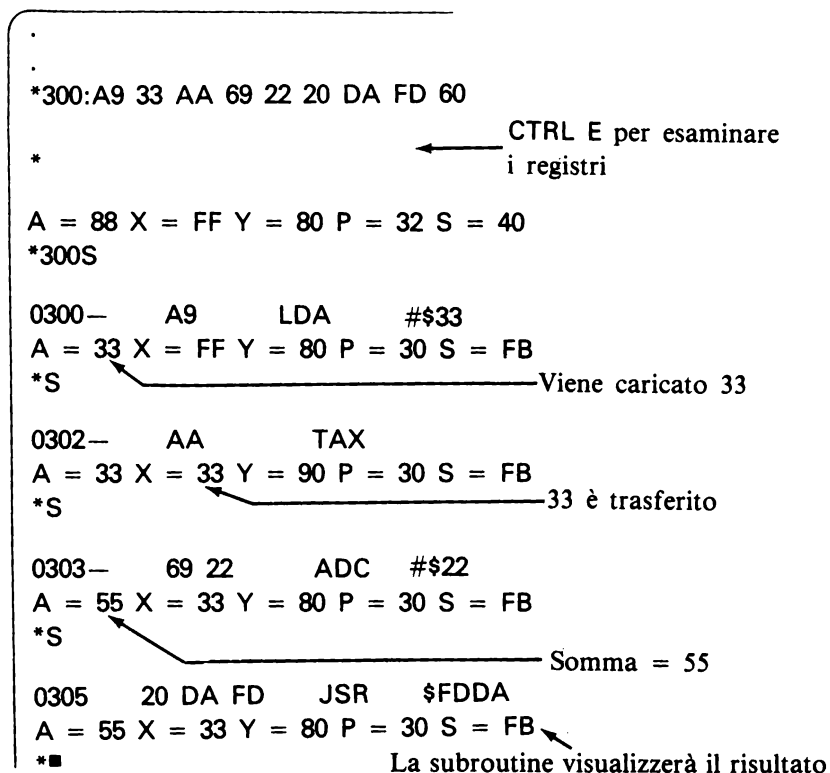
Quando l'istruzione è eseguita, i contenuti dei registri vengono visualizzati. Useremo il comando *single-step* per vedere i cambiamenti dei registri mentre viene eseguita ogni istruzione.

Il comando *single-step* è una S. Per usarlo, si scrive l'indirizzo di partenza del programma seguito dalla S.

Esempio

Per ogni successiva istruzione, scrivete S e premete RETURN, come nel seguente esempio.

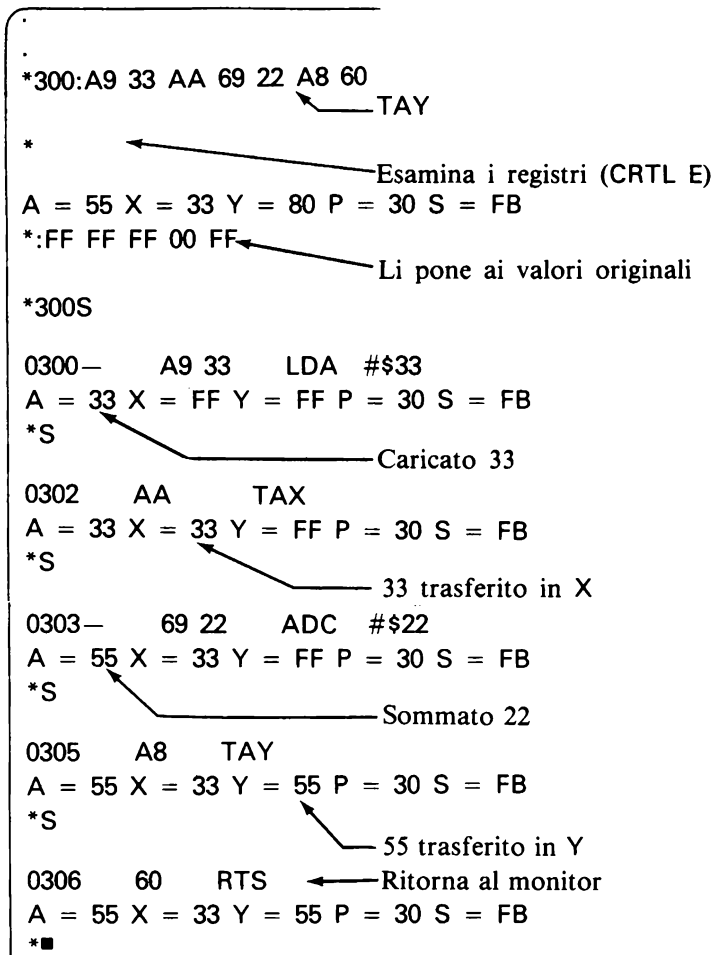
Inserite il programma e poi percorretelo un passo alla volta.



Percorrendo il programma un passo alla volta potete vedere i risultati dell'esecuzione delle istruzioni. Si possono anche vedere i cambiamenti che le singole istruzioni provocano nei registri. In questo esempio, l'accumulatore ed il registro X sono stati modificati durante l'esecuzione del programma.

La subroutine a FDDA contiene moltissimi passi, così ci siamo fermati quando siamo arrivati a quell'istruzione. Avrete anche notato che il valore nel registro di stato del processore era 32 prima che fosse eseguita la prima istruzione (LDA). Dopo l'esecuzione dell'istruzione LDA, il registro di stato è diventato 30. Il bit dello zero è stato azzerato perché 33 non è uguale a zero.

Aggiungiamo ora un'istruzione, TAY (A8), per trasferire il risultato dell'addizione nel registro Y. Questa volta non visualizzeremo il risultato.



Così avete visto che i registri X e Y possono essere usati per una memorizzazione temporanea. Nel Cap. 11, mostreremo che possono essere usati come registri indice per le istruzioni nei modi di indirizzamento indicizzati.

Proviamo ora un ultimo programma per concludere il capitolo. Questa volta incrementeremo il registro X, lo confronteremo con zero, e se il contenuto del registro X non è uguale a zero, lo incrementeremo ancora.

Il programma

300 E8

310 E0

302 00

303 D0

304 FB

305 60

INX

CPX 0

BNE FB

RTS

5

4

3 ciclo

2

1

←

←

←

←

←

FB = -5, se volete
controllare sull'Apple
(0-5 = FB)

Inserite il programma

*300:E8 E0 00 D0 FB 60

*■

Prima di eseguire il programma mettiamo il valore FD nel registro X. Quanto ci vorrà prima che il contenuto del registro X raggiunga zero? Per scoprirlo percorreremo il programma un passo alla volta.

*300:E8 E0 00 D0 FB 60

*

A = 55 X = 33 Y = 55 P = 30 S = EF

*:00 FD

*■

A

X

← Non ci interessano gli altri registri,
così cambiamo soltanto i primi due,
A ed X

Poi, percorriamo il programma un passo alla volta.

*300:E8 E0 00 D0 FB 60

*

A = 55 X = 33 Y = 55 P = 30 S = EF

*:00 FD

* 3 0 0 S

← Richiede il primo passo

0300—	E8	INX	
A=00 X=FE Y=55 P=B0 S=F1		←	X = FD + 1 = FE
*S		←	Richiede il prossimo passo
0301—	E0 00	CPX	#\$00
A=00 X=FE Y=55 P=B1 S=F1			
*S		←	Richiede il prossimo passo
0303—	D0FB	BNE	\$0300
A=00 X=FE Y=55 P=B1 S=F1			
*S		←	Prossimo passo
0300—	E8	INX	
A=00 X=FF Y=55 P=B1 S=F1		←	X = FE + 1 = FF
*S		←	Prossimo passo
0301—	E0 00	CPX	#\$00
A=00 X=FF Y=55 P=B1 S=F1			
*S		←	Prossimo passo
0303—	D0 FB	BNE	\$0300
A=00 X=FF Y=55 P=B1 S=F1			
*S		←	Prossimo passo
0300—	+8	INX	#\$00
A=00 X=00 Y=55 P=31 S=F1		←	X = FF + 1 = 00
*S			
			Bit del riporto uguale a uno perché
0301—	E0 00	CPX	#\$00
A=00 X=00 Y=55 P=33 S=F1			
*S		←	Prossimo passo
0303—	D0 FB	BNE	\$0300
A=00 X=00 Y=55 P=33 S=F1			
*S		←	Prossimo passo
0305—	60	RTS	
A=00 X=00 Y=55 P=33 S=F1		←	Il salto non è stato effettuato poiché il registro X = valore che segue CPX (00)
*■			

SOMMARIO

In questo capitolo avete conosciuto il monitor di sistema dell'Apple. Avete esplorato alcune sue caratteristiche che rendono più facile la programmazione in linguaggio macchina. Si spera che continuerete queste esplorazioni con altri esperimenti, finché non avrete completamente familiarizzato con le capacità del monitor.

Avete:

1. usato l'addizione e la sottrazione esadecimali nel modo immediato;
2. scoperto come sono interpretati i numeri negativi usando valori esadecimali;

3. imparato come eseguire operazioni di addizione e sottrazione con numeri decimali codificati in binario;
4. visto come spostare blocchi di dati da una zona della memoria in un'altra;
5. imparato come verificare che due blocchi di memoria sono lo stesso;
6. appreso come esaminare e modificare i registri.

Istruzioni nuove

1. SED (*SEt Decimal mode*, abilita il modo decimale) usata per porre a 1 il flag decimale del registro di stato del processore. Tutte le successive istruzioni di somma o sottrazione sono eseguite come se i valori su cui operano fossero numeri decimali codificati in binario.
2. CLD (*CLear Decimal mode*, disabilita il modo decimale), azzerava il flag decimale del registro di stato del processore. Questo riporta le operazioni di addizione e sottrazione al formato binario.
3. TAY (*Transfer Accumulator to Y register*, trasferisci l'accumulatore nel registro Y), copia il contenuto dell'accumulatore nel registro Y.

Nuovi comandi ed usi del monitor

1. *Addizione e sottrazione esadecimale* Inserite due numeri esadecimali di 2 cifre separati dal simbolo dell'operazione e premete RETURN. Verrà visualizzato il risultato esadecimale a 2 cifre.
2. *Spostamento di un blocco di dati in memoria* Indicate al monitor gli indirizzi che delimitano il blocco da spostare e l'indirizzo a cui spostarlo. Allora il monitor copierà i dati nella zona prefissata.

Formato: destinazione<inizio.fine M

Esempio: *330<300.309M

— copierà il blocco di dati che sta tra le locazioni 300 e 309 nelle locazioni da 330 a 339.

3. *Verifica di due blocchi di dati* Indica al monitor gli indirizzi che delimitano un blocco di dati e l'indirizzo iniziale del secondo blocco. Il monitor confronterà i due blocchi e vi dirà se ci sono differenze.

Formato: destinazione<inizio.fine M

Esempio: *330<300.309V

— confronterà i dati che stanno tra le locazioni 300 e 309 con quelli contenuti nelle locazioni da 330 a 339.

4. *Esamina i registri* I registri A, X, Y, P, S possono essere esaminati premendo i tasti CTRL ed E assieme, seguiti da un RETURN.
5. *Modifica dei registri* I registri A, X, Y, P, S possono essere modificati esaminandoli prima come al punto 4 e poi scrivendo due punti (:) seguiti dai nuovi valori, in ordine, separati da uno spazio.

ESERCIZI

1. Usando l'addizione esadecimale nel modo immediato, rispondete a ciò che segue:

a. $*2E + 35$	c. $*A4 - 31$
=	=
_____	_____
b. $*E5 + F$	d. $*5A - 2E$
=	=
_____	_____
2. Che cosa sarebbe visualizzato se eseguiamo ciò che segue in modo immediato?

a. $*D7 + 4F$	b. $*24 - 27$
=	=
_____	_____
3. Esprimete i seguenti numeri decimali in forma decimale codificata in binario.

a. 28 = _____
b. 37 = _____
c. 91 = _____
4. Se il programma Somma di due numeri decimali è modifico come segue, scrivete che cosa sarà visualizzato.

*303:42

*305:53

*300G

_____ ← segnate qui la risposta

5. Mostrate come si usa il comando del monitor MOVE per spostare il

programma ma Somma di due numeri decimali nella zona tra le locazioni 350 e 359.

```

* _____

```

6. Mostrate come si può verificare che lo spostamento dell'Esercizio E è stato fatto correttamente.

```

.
.
.
* _____

```

7. Descrivete come si esaminano i registri.

```

_____
_____

```

Supponiamo che voi abbiate esaminato i registri e che il video mostri i risultati indicati sotto. Riportate che cosa si dovrebbe scrivere per cambiare l'accumulatore a 00, il registro X a 80 ed il registro Y a 40.

```

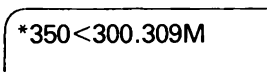
.
.
A = FF X = FF Y = FF P = 00 S = FF
* _____

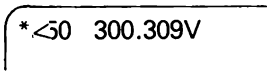
```

RISPOSTE AGLI ESERCIZI

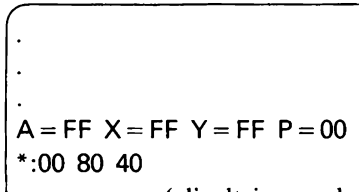
1. a. = 63 c. = 73
 b. = F4 d. = 2C
2. a. = 26 (D7 + 4F = 126, il riporto non viene visualizzato)
 b. = FD (FD = -3)
3. a. 28 = 0010 1000
 b. 37 = 0011 0111
 c. 91 = 1001 0001

4. 95 (42 + 53 = 95 in decimale)

5.  *350<300.309M

6.  *<50 300.309V

7. Premere i tasti CTRL ed E assieme.
Poi premere RETURN.

8.  .
.
.
.
A = FF X = FF Y = FF P = 00 S = FF
*:00 80 40
(gli altri non devono essere cambiati)

Mini-assembler e modi di indirizzamento

MA

MS

C'è un altro programma che fa parte della ROM dell'Integer Basic dell'Apple, ma che non è disponibile nel sistema Apple II Plus. Questo programma è chiamato Apple mini-assembler. È detto "mini" perché non è in grado di interpretare etichette simboliche come nell'assembler completo. Esso è molto utile nel creare programmi in linguaggio macchina. In precedenti capitoli, abbiamo mostrato i codici di linguaggio macchina assieme ai loro codici mnemonici. Il codice mnemonico è un'abbreviazione del nome di ogni istruzione. Abbiamo tradotto questi codici mnemonici in istruzioni di linguaggio macchina esadecimali o binarie (codici op.). Questo procedimento è detto *assemblaggio manuale*.

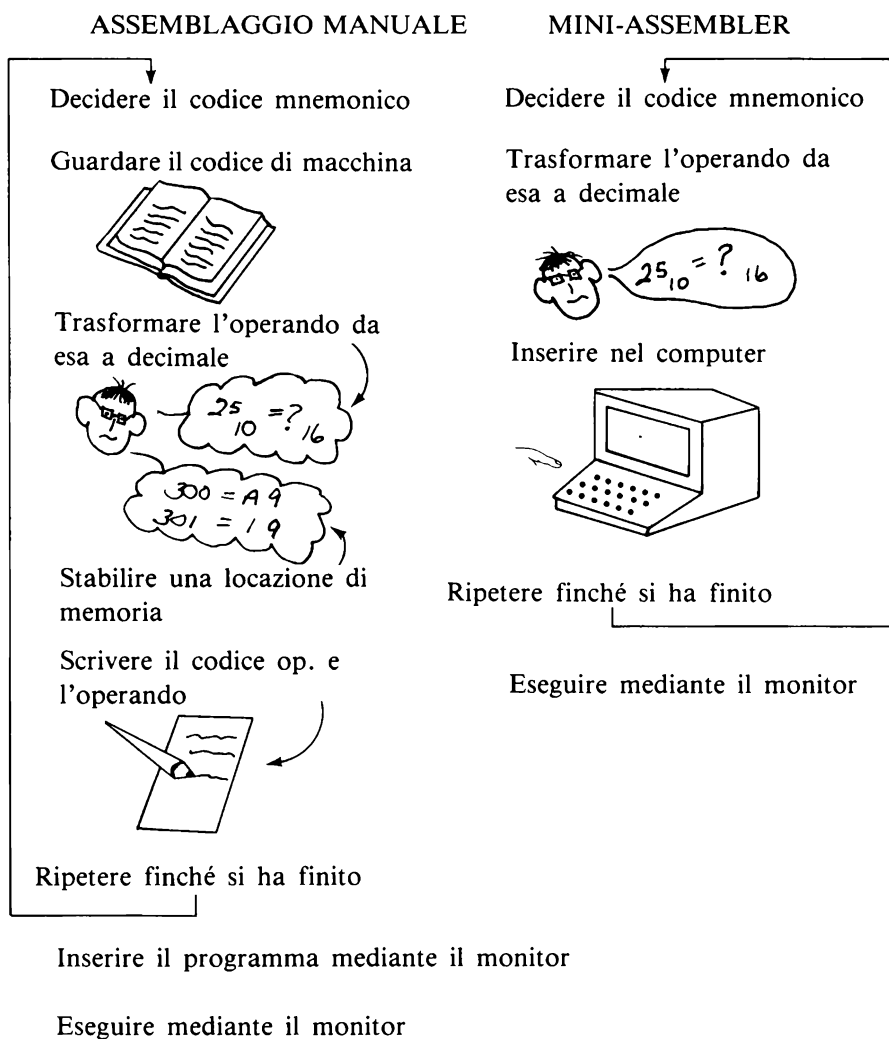
Esempi

<i>Codice mnemonico</i>	<i>Modo di indirizzamento</i>	<i>Codice di macchina</i>
LDA	Immediato	A9
ADC	Immediato	69
CLC	Implicito	18

L'assemblaggio manuale è un lavoro noioso, non interessante, dove è fa-

cile commettere piccoli ma disastrosi errori. La lunghezza delle istruzioni varia, e le destinazioni dei salti devono essere calcolate. Alcune istruzioni richiedono, come operandi, dati, mentre altre richiedono indirizzi di memoria o registri. È facile scegliere codici op. o indirizzi sbagliati. Ed è anche facile spostare o sbagliare cifre, ecc. Per noi sarebbe molto più facile assegnare il compito di assemblare un programma al computer. Il mini-assembler dell'Apple può assemblare facilmente i programmi se noi li scriviamo usando istruzioni del *linguaggio assembler*. Questo capitolo sarà dedicato all'apprendimento delle regole per usare il mini-assembler e la forma del linguaggio assembler per i vari modi di indirizzamento usati.

Confronto dei metodi di assemblaggio



USO DEL MINI-ASSEMBLER

Per usare il mini-assembler, dovete sapere come accedervi e come uscirne quando avete finito di assemblare il vostro programma. Se nel vostro Apple avete la scheda Applesoft II ROM posizionate l'interruttore che sta sul retro del calcolatore sull'Integer Basic. Il programma mini-assembler è su questa ROM. Se la vostra macchina non ha l'Integer Basic, non ha nemmeno il mini-assembler, e in tal caso questo capitolo non servirà per il vostro sistema. Sono disponibili altri tipi di assembler che possono essere caricati da cassetta o disco.

Per mandare in esecuzione il mini-assembler, battete F666G

```

  *F666G ← Voi scrivete
  !■ ← L'assembler risponde con il suo prompt,
        il punto esclamativo (!)

```

Alla fine potrete lasciare il mini-assembler e rientrare nel monitor per eseguire il programma che avete assemblato. Questo si può fare in due modi:

1. Premete il tasto RESET
2. Scrivete il comando del monitor (preceduto dal segno di dollaro):

```

  ! $FF69G ← Voi scrivete
  * ← Appare il prompt del monitor

```

Dopo l'accesso al mini-assembler e prima di ritornare al monitor, è stato inserito un programma in linguaggio assembler. Useremo il programma Somma di due numeri decimali del Cap. 10 per una breve dimostrazione.

1

```

  *F666G ← Per partire battete questo
  !■ ← Appare il prompt del mini-assembler

```

- 2 Inserite l'indirizzo della prima istruzione ed il suo codice mnemonico

separati da due punti. Il mini-assembler visualizza ogni istruzione al momento dell'assemblaggio.

```
*F666G
!300:SED ← Inserire l'indirizzo: istruzione
```

Quando premete RETURN, la vostra istruzione in assembler scomparirà ed apparirà il codice assemblato del linguaggio macchina.

```
*F666G
0300— F8 SED ← Eccolo
! ■ ← Mnemonico
      ← Codice op.
      ← Indirizzo
Cursore, pronto
per un'altra
istruzione
```

- 3 Scrivete soltanto il codice mnemonico dell'istruzione di azzeramento del riporto.

```
*F666G
0300— F8 SED
!CLC ■ ← Azzerà il riporto
      ← 1 spazio
```

Premete RETURN

```
*F666G
0300— F8 SED
0301— 18 CLC
! ■ ← Prossima?
```

Notate lo spazio che segue il prompt del mini-assembler. Se questo spa-

zio non viene messo, l'istruzione non sarà accettata. Se effettuate un input non valido, vi sarà indicato.

Esempio

```
*F666G
0300— F8      SED
!CLC
^
!■
```

La freccia in su indica che c'è un errore. Riprovate.

- 4 Finora, entrambe le istruzioni usate erano nel modo implicito, e non richiedevano nessun operando. La prossima istruzione, LDA, è nel modo immediato, e deve essere seguita dal segno # prima dell'operando 23 per dire al computer che questo è un operando immediato.

```
*F666G
0300— F8      SED
0301— 18      CLC
! LDA #23■
```

← Scrivete

↑ Spazio

Premete RETURN

```
*F666G
0300— F8      SED
0301— 18      CLC
0302— A9 23  LDA #$23
!■
```

- 5 La prossima è l'istruzione ADC seguita da 18.

```
.
.
.
! ADC #18■
```

← Scrivete

Premete RETURN

```
*F666G
0300— F8      SED
0301— 18      CLC
0302— A9 23   LDA #$23
0304— 69 18   ADC #18
!■
```

- 6 La successiva è l'istruzione JSR seguita dall'indirizzo assoluto FDDA.

```
.
.
.
!JSR FDDA■ ← Scrivete
```

Premete RETURN

```
*F666G
0300— F8      SED
0301— 18      CLC
0302— A9 23   LDA #$23
0304— 69 18   ADC #$18
0306— 20 DA FD JSR $FDDA
!■
```

- 7 Ed ultima, l'istruzione RTS (modo implicito)

```
.
.
.
!RTS■ ← Scrivete
```

Premete RETURN

```

*F666G
0300— F8          SED
0301— 18          CLC
0302— A9 23       LDA #$23
0304— 69 18       ADC #$18
0306— 20 DA FD    JSR $FDDA
0309— 60          RTS
!■

```

8 Ora lasciamo il mini-assembler. Il programma è stato assemblato

```

.
.
.
0306— 20 DA FD    JSR $FDDA
0309— 60          RTS
!$FF69G ←
*

```

Scrivete queste per tornare
al monitor

Ora il programma assemblato è in memoria. Potete listarlo con il comando del monitor.

Questo comando del monitor lavora come il comando LIST del Basic. Se scrivete:

300L e premete RETURN

il monitor listerà i contenuti di 20 locazioni consecutive ad iniziare dalla 300.

```

.
.
.
*300L ← Comando di lista
0300— F8          SED
0301— 18          CLC
0302— A9 23       LDA #$23
0304— 69 18       ADC #$18
0306— 20 DA FD    JSR $FDDA
0309— 60          RTS

```

030A—	FF	???
030B—	FF	???
030C—	FF	???
030D—	FF	???
030E—	FF	???
030F—	FF	???
0310—	FF	???
0311—	FF	???
0312—	FF	???
0313—	FF	???
0314—	FF	???
0315—	FF	???
0316—	FF	???
0317—	FF	???

*

Tutti questi dati
sono privi di signifi-
cato per noi e per
il mini-assembler

Le prime linee sono il vostro programma assemblato. Il resto delle linee non è usato dal programma. Ogni volta che si dà il comando di lista vengono visualizzate 20 locazioni. Per programmi più lunghi, verranno visualizzate altre 20 locazioni ogni volta che premete L e poi RETURN. È il momento di eseguire il programma.

```

.
.
.
*300G
41 ← Stesso risultato di prima
*
```

Notate che, quando si usa il mini-assembler, non occorre guardare i codici di linguaggio macchina per le istruzioni. I codici mnemonici sono più facili da ricordare di quelli numerici. Ricordate che, come già menzionato, alcune istruzioni possono essere usate in parecchi modi di indirizzamento. Come fa il mini-assembler a sapere quale modo vogliamo? Nel programma Somma di due numeri decimali, le scelte che doveva fare il mini-assembler erano facili. Osservate le istruzioni che sono state usate.

1. SED (abilita il modo decimale) Questa istruzione è usata soltanto nel modo implicito. Non occorre una scelta.
2. CLD (azzerà il riporto) Viene usata in modo implicito anche questa. Nessuna scelta.

3. LDA (carica l'accumulatore) Notate il segno '#' davanti al numero 23. Questo dice all'assembler che ci vuole il codice per il modo di indirizzamento immediato.
4. ADC (somma con riporto) Ancora una volta, il segno # serve a specificare il modo di indirizzamento immediato.
5. JSR (salta alla subroutine) Questa istruzione viene usata soltanto nel modo assoluto. Non occorre una scelta.

Avete usato frequentemente i modi di indirizzamento implicito, immediato, pagina zero e assoluto. Ora rivolgeremo la nostra attenzione all'indirizzamento indicizzato, che è molto utile sebbene un po' più complesso.

INDIRIZZAMENTO INDICIZZATO

Due istruzioni fra le più frequentemente usate sono quelle che caricano l'accumulatore e mettono il suo contenuto in memoria. Useremo queste due istruzioni per mostrare i modi di indirizzamento indicizzati. Ecco una lista dei modi indicizzati.

LDA

<i>Espressione in assembler</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Numero di byte usati</i>
LDA oper, X	Pagina zero, X	B5	2
LDA oper, X	Assoluto, X	BD	3
LDA oper, Y	Assoluto, Y	B9	3
LDA (oper, X)	(Indiretto, X)	A1	2
LDA (oper), Y	(Indiretto), X	B1	2

STA

<i>Espressione in assembler</i>	<i>Modo di indirizzamento</i>	<i>Codice op.</i>	<i>Numero di byte usati</i>
STA oper, X	Pagina Zero, X	95	2
STA oper, X	Assoluto, X	9D	3
STA oper, X	Assoluto, Y	99	3
STA (oper, X)	(Indiretto, X)	81	2
STA (oper), Y	(Indiretto), Y	91	2

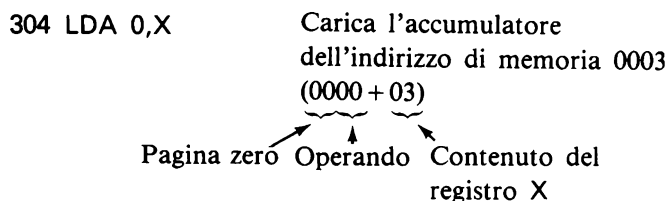
INDICIZZAZIONE IN PAGINA ZERO

Dato che le istruzioni di pagina zero sono eseguite più velocemente di quelle in altri modi di indirizzamento, il computer Apple utilizza già gran parte delle locazioni necessarie usate da queste istruzioni. Le locazioni di pagina zero sono quelle comprese tra 0000 e 00FF. Per riconoscere una di queste locazioni il computer necessita soltanto della parte bassa dell'indirizzo. Guardando la mappa della memoria della pagina zero (*Apple II Reference Manual*), potete notare che le locazioni da 0000 a 001F *non sono usate* dal monitor o dall'Integer Basic su ROM. *Sono usate* dal Basic Applesoft II. Se non utilizziamo il Basic Applesoft II, possiamo usare quest'area della memoria per dimostrare l'indirizzamento in pagina zero. Tutte le istruzioni che usano questo modo di indirizzamento, *eccetto* LDA (carica il registro X) e STX (memorizza il registro X), usano il registro X per modificare l'operando dell'istruzione (l'indirizzo usato).

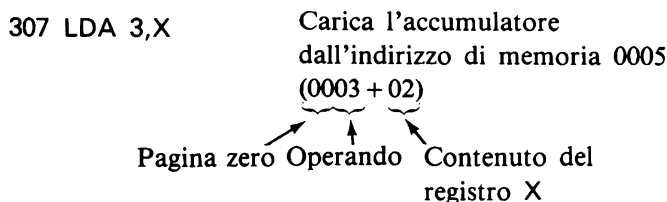
LDA oper, X	Carica l'accumulatore
Codice op. B5 (Pagina zero)	dall'indirizzo di pagina zero
2° byte, operando (parte	+ il contenuto del
bassa dell'indirizzo)	registro X

Esempio

1. Carica l'accumulatore dell'indirizzo di memoria = 3



2. Contenuto del registro X



STA oper,X	Carica l'accumulatore dall'indirizzo
Codice op. 95 (pagina zero)	zero + il contenuto del
2° byte, operando (parte	registro X
bassa dell'indirizzo)	

Questa istruzione lavora come la LDA oper,X (modo pagina zero) tranne che il contenuto dell'accumulatore viene memorizzato.

Entrambi i tipi sono mostrati nel seguente programma che sposta i valori dalle locazioni 0005 e 0006 alle locazioni 0015 e 0016. Useremo il mini-assembler per assemblare il programma.

```
*F666G
```

```
!■
```

Scrivete: 300:LDX#5

```
*F666G
```

Scrivete ➤	0300 — A205	LDX#\$05	← Prima istruzione assemblata
	! LDA #22		

```
*F666G
```

Scrivete ➤	0300 — A2 05	LDX#05	← Prossima istruzione
	0302 — A9 22	LDA#\$22	
	! STA 0,X		

```
*F666G
```

Scrivete ➤	0300 — A2 05	LDX#\$05	← Prossima
	0302 — A9 22	LDA#\$22	
	0304 — 95 00	STA \$00,X	
	! LDA #33		

```

*F666G
0300— A2 05      LDX # $05
0302— A9 22      LDA # $22
0304— 95 00      STA $00,X
0306— A9 33      LDA # $33 ← Prossima
Scrivete ► ! STA 1,X

```

```

*F666G
0300— A2 05      LDX # $05
0302— A9 22      LDA # $22
0304— 95 00      STA $00,X
0306— A9 33      LDA # $33
0308— 95 01      STA $01,X ← Prossima
Scrivete ► ! LDA 0,X

```

Ogni volta viene aggiunta una nuova istruzione

```

*F666G
0300— A2 05      LDX # $05
0302— A9 22      LDA # $22
0304— 95 00      STA $00,X
0306— A9 33      LDA # $33
0308— 95 01      STA $01,X
030A— 85 00      LDA $00,X
! STA 10,X

```

```

*F666G
0300— A2 05      LDX # $05
0302— A9 22      LDA # $22
0304— 95 00      STA $00,X
0306— A9 33      LDA # $33
0308— 95 01      STA $01,X
030A— 85 00      LDA $00,X
030C— 95 10      STA $10,X
! INX

```

*F666G

0300—	A2 05	LDX #\$05
0302—	A9 22	LDA #\$22
0304—	95 00	STA \$00,X
0306—	A9 33	LDA #\$33
0308—	95 01	STA \$01,X
030A—	85 00	LDA \$00,X
030C—	95 10	STA \$10,X
030E—	E8	INX
! LDA 0,X		

F666G

0300—	A2 05	LDX #\$05
0302—	A9 22	LDA #\$22
0304—	95 00	STA \$00,X
0306—	A9 33	LDA #\$33
0308—	95 01	STA \$01,X
030A—	85 00	LDA \$00,X
030C—	95 10	STA \$10,X
030E—	E8	INX
030F—	85 00	LDA \$00,X
! STA 10,X		

F666G

0300—	A2 05	LDX #\$05
0302—	A9 22	LDA #\$22
0304—	95 00	STA \$00,X
0306—	A9 33	LDA #\$33
0308—	95 01	STA \$01,X
030A—	85 00	LDA \$00,X
030C—	95 10	STA \$10,X
030E—	E8	INX
030F—	85 00	LDA \$00,X
0311—	95 10	STA \$10,X
! RTS		

F666G

```

0300— A2 05      LDX # $05
0302— A9 22      LDA # $22
0304— 95 00      STA $00,X
0306— A9 33      LDA # $33
0308— 95 01      STA $01,X
030A— 85 00      LDA $00,X
030C— 95 10      STA $10,X
030E— E8         INX
030F— 85 00      LDA $00,X
0311— 95 10      STA $10,X
0313— 60         RTS

```

! \$FF69G

Lasciamo il mini-assembler

*

Istruzioni di pagina zero indicizzate sono state usate a 0304, 0308, 030A, 030C, 030F e 0311. Ora eseguiamo il programma. Poi esaminiamo le locazioni 0005, 0006, 0015 e 0016.

*300G

*0005.0006

0005— 22 33 ← Originale

*0015.0016

0015— 22 33 ← Copia

*

INDIRIZZAMENTO ASSOLUTO INDICIZZATO

Entrambi i registri X e Y possono essere usati per indicizzare un indirizzo assoluto in questo modo. Non occorre nemmeno limitarsi alla memoria di pagina zero.

Esempi

LDA 301,X

Codice op. BD (assoluto indicizzato)

2° byte, parte bassa dell'indirizzo (10)

3° byte, parte alta dell'indirizzo (03)

Questa istruzione carica l'accumulatore dall'indirizzo di memoria 0310 + il contenuto del registro X. (Si poteva anche usare Y.)

STA 320,Y

Codice op. 99 (assoluto indicizzato)

2° byte, parte bassa dell'indirizzo (20)

3° byte, parte alta dell'indirizzo (03)

Questa istruzione memorizza il contenuto dell'accumulatore nell'indirizzo di memoria 0320 + il contenuto del registro Y. (Poteva essere usato anche X.)

Utilizzeremo queste istruzioni per memorizzare alcuni caratteri inseriti dalla tastiera, e per riprenderli e visualizzarli sullo schermo. Useremo il mini-assembler per scrivere il programma.

Prima memorizzeremo 17 caratteri inseriti dalla tastiera in locazioni consecutive usando l'istruzione STA nel modo di indirizzamento indicizzato, con X come registro indice.

*F666G

!300:LDX #0

← Carica Z con zero

*F666G

0300 — A2 00

LDX #\$00

! JSR FD35

← Riceve un carattere

*F666G

0300 — A2 00

LDX #\$00

0302 — 20 35 FD

JSR \$FD35

! STA 1000,X

← Lo memorizza

*F666G

```
0300— A2 00      LDX #00
0302— 20 35 FD    JSR $FD35
0305— 9D 00 10    STA $1000,X
! INX              ← Incremento X di uno
```

*F666G

```
0300— A2 00      LDX #00
0302— 20 35 FD    JSR $FD35
0305— 9D 00 10    STA $1000,X
0308— E8          INX
! CPX #11          ← Già finito?
```

*F666G

```
0300— A2 00      LDX #00
0302— 20 35 FD    JSR $FD35
0305— 9D 00 10    STA $1000,X
0308— E8          INX
0309— E0 11      CPX #11
! BNE 302         ← Se no, ritorna a 302
```

*F666G

```
0300— A2 00      LDX #00
0302— 20 35 FD    JSR $FD35
0305— 9D 00 10    STA $1000,X
0308— E8          INX
0309— E0 11      CPX #11
030B— D0 F5      BNE $0302
! JSR FC58        ← Cancella lo schermo
```

*F666G

```

0300— A2 00      LDX #$00
0302— 20 35 FD    JSR $FD35
0305— 9D 00 10    STA $1000,X
0308— E8          INX
0309— E011        CPX #$11
030B— D0 F5       BNE $0302
030D— 20 58 FC    JSR $FC58
!■

```

Fermiamoci per un po' per vedere come lavora questa parte del programma. Notate che questa parte contiene un ciclo molto simile ad un ciclo FOR...NEXT del Basic. La subroutine a FD35 aspetta che sia inserito un carattere dalla tastiera. Il carattere viene poi memorizzato dall'istruzione assoluta indicizzata (STA 1000,X). Il registro viene incrementato di uno e controllato per vedere se è uguale a 11 (esadecimale). Se non lo è, si ritorna alla subroutine per un altro carattere. Quando è stato inserito il diciassettesimo carattere, X avrà raggiunto 11. Lo schermo viene cancellato dalla subroutine a FC58. Il valore di X non controlla soltanto l'uscita dal ciclo; esso indicizza anche l'indirizzo della locazione dove verrà memorizzato il carattere. Così, i caratteri verranno memorizzati in queste locazioni consecutive:

<i>Esecuzione del ciclo</i>	<i>Reg. X (ESA)</i>	<i>Locazione usata per memorizzare il carattere</i>
1	0	1000
2	1	1001
3	2	1002
4	3	1003
.	.	.
.	.	.
15	E	100E
16	F	100F
17	10	1010

Proseguendo con la seconda parte del programma, inserite le seguenti istruzioni ogni volta che vedete il prompt del mini-assembler:

! LDA #0	Azzera X
! LDA 1000,X	Carica un carattere dalla memoria
! JSR FDED	Lo visualizza
! INX	Si prepara per il prossimo
! CPX #11	Già finito
! BNE 312	Se no, salta a 312
! RTS	Se sì, torna al monitor

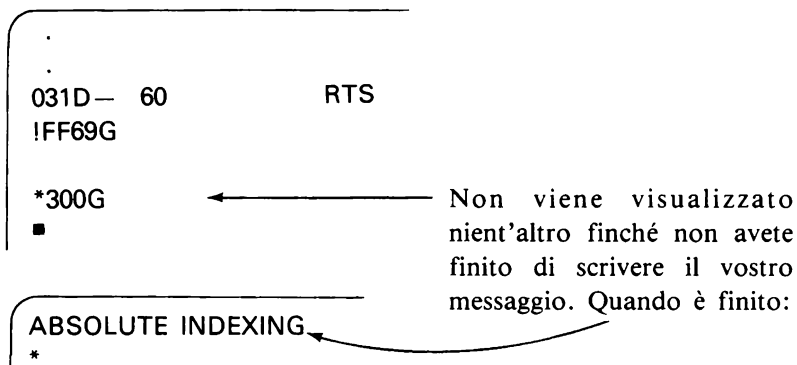
Dopo questo, il video visualizzerà:

*F666G		
0300—	A2 00	LDX #\$00
0302—	20 35 FD	JSR \$FD35
0305—	9D 00 10	STA \$1000,X
0308—	E8	INX
0309—	E0 11	CPX #\$11
030B—	D0 F5	BNE \$0302
030D—	20 58 FC	JSR \$FC58
0310—	A2 00	LDX #\$00
0312—	BD 00 10	LDA \$1000,X
0315—	20 ED FD	ISR \$FDED
0318—	E8	INX
0319—	E0 11	CPX #\$11
031B—	D0 F5	BNE \$0312
031D—	60	RTS
! \$FF69G	← Concludiamo lasciando il mini-assembler	
*		

Anche la seconda parte del programma contiene un ciclo che carica un carattere dalla memoria e lo visualizza. Il ciclo viene ripercorso finché tutti i 17 caratteri sono stati visualizzati. Poi si ritorna al monitor. L'istruzione con indirizzamento indicizzato (LDA 1000,X) è usata per caricare l'accumulatore ogni volta che si percorre il ciclo. Poiché il registro X è stato azzerato prima del ciclo, l'istruzione riprende i caratteri nello stesso ordine in cui sono stati memorizzati (vedere la tavola all'inizio di questo capitolo). L'indirizzamento indicizzato è molto pratico per caricare e memorizzare valori in blocchi di locazioni consecutive.

Quando eseguirete il programma, inserite 17 caratteri dalla tastiera (anche lo spazio è un carattere). Non vedrete i caratteri mentre li inserite. Saranno visualizzati *dopo* essere stati inseriti tutti 17. Ecco una tipica esecuzione con il messaggio "ABSOLUTE INDEXING". Potete usare un

qualsiasi messaggio di 17 caratteri, oppure fare alcune modifiche al programma e riempire lo schermo.



Come indice, invece di X, si poteva anche usare il registro Y. Si possono usare entrambi indifferentemente.

STA 1000,X o STA 1000,Y
 LDA 1000,X o LDA 1000,Y

Si poteva anche usare una combinazione dei due:

STA 1000,X
 LDA 1000,Y

Naturalmente, al registro scelto si devono dare valore iniziale ed incremento correttamente.

INDIRIZZAMENTO INDICIZZATO INDIRETTO

L'uso principale di questo modo di indirizzamento consiste nel prelevare dati da una lista di indirizzi per eseguire un'operazione.

Poiché non c'è molto spazio inutilizzato nella nostra memoria di pagina zero, possiamo presentare soltanto una banale applicazione di questo modo di indirizzamento. Ancora una volta, l'istruzione LDA servirà ad esempio.

Il formato è:

LDA (oper,X)
 Codice op. A1
 2° byte, offset

Il secondo byte dell'istruzione (offset) è aggiunto al contenuto del registro X (il riporto non viene considerato). Il risultato punta ad una *locazione di pagina zero* che contiene la parte bassa dell'indirizzo effettivo da cui sarà caricato il dato. La prossima locazione di pagina zero contiene la parte alta dell'indirizzo effettivo.

Esempio

La locazione di memoria 0019 contiene il valore 45

La locazione di memoria 001A contiene il valore 10

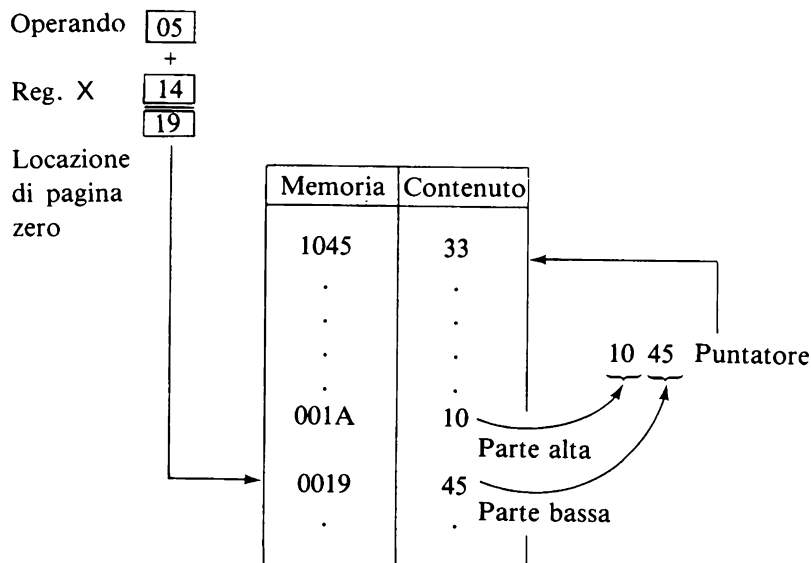
Il registro X contiene il valore 14

L'istruzione da eseguire è:

LDA (05,X)

Primo, il contenuto del registro X è sommato all'operando (offset) $14 + 5 = 19$. Il risultato è l'indirizzo di pagina zero che contiene la parte bassa dell'indirizzo della locazione da cui si deve caricare l'accumulatore. Secondo, la parte alta dell'indirizzo della locazione da cui sarà caricato il dato è trovata nel prossimo indirizzo di pagina zero ($19 + 1 = 1A$)

Nell'esempio:



Dopo l'esecuzione di LDA(05,X): accumulatore 33

Tentiamo ora una breve dimostrazione del modo di indirizzamento indicizzato indiretto per l'istruzione LDA, usando il mini-assembler per as-

sembrare il programma. Primo, inserite le istruzioni dopo il prompt (!). Ogni linea è assemblata appena inserita, ma non la mostreremo finché non sono state inserite tutte.

!300:LDA #14	Mette 14 nel registro X
! LDA #10	Mette 10 nell'accumulatore
! STA 1A	Lo memorizza in 001A
! LDA #45	Mette 45 nell'accumulatore
! STA 19	Lo memorizza in 0019
! LDA #33	Mette 33 nell'accumulatore
! STA 1045	Memorizza 33 in 1045
! LDA #0	Carica zero
! LDA (05,X)	Modo di indirizzamento indicizzato indiretto
! JSR FDDA	Visualizza l'accumulatore
! RTS	

Il video:

```

*F666G
0300— A2 14      LDX #$14
0302— A9 10      LDA #$10
0304— 85 1A      STA $1A
0306— A9 45      LDA #$45
0308— 85 19      STA $19
030A— A9 33      LDA #$33
030C— 8D 45 10   STA $1045
030F— A9 00      LDA #00
0311— A1 05      LDA ($05,X
0313— 20 DA FD   JSR $FDA
0316— 60         RTS
!$FF69G ←———— Lasciamo il mini-assembler
*

```

Eseguite il programma. L'accumulatore viene caricato dall'istruzione in modo indicizzato indiretto a 0311. Poi viene visualizzato il valore 33.

```

.
.
.
0313— 20 DA FD      JSR $FDDA
0316— 60            RTS
!$FF69G

*300G
33 ←
*

```

Eccolo!

Possiamo vedere che, come ci aspettavamo, 33 è stato visualizzato. Come facciamo ad essere sicuri che è stato effettivamente visualizzato a causa dell'istruzione di indirizzamento indicizzato indiretto alla locazione 0311? Se volete controllare percorrete il programma un passo alla volta finché non raggiungete il salto all'istruzione di subroutine a 0313. A quel punto 33 sarà posto nell'accumulatore dall'istruzione di indirizzamento indicizzato indiretto, LDA(05,X).

```

*300S

0300— A2 14          LDX #$14
      A = 00 X = 14 Y = FF P = 30 S = 0C
*S

0302— A9 10          LDA #$10
      A = 10 X = 14 Y = FF P = 30 S = 0C
*S

0304— 85 1A          STA $1A
      A = 10 X = 14 Y = FF P = 30 S = 0C
*S

0306— A9 45          LDA #$45
      A = 45 X = 14 Y = FF P = 30 S = 0C
*S

0308— 85 19          STA $19
      A = 45 X = 14 Y = FF P = 30 S = 0C
*S

```

<pre> 030A— A9 33 LDA #\$33 A = 33 X = 14 Y = FF P = 30 S = 0C *S 030C— 8D 45 10 STA \$1045 A = 33 X = 14 Y = FF P = 30 S = 0C *S 030F— A9 00 LDA #\$00 A = 00 X = 14 Y = FF P = 30 S = 0C *S 0311— A1 05 LDA (\$05,X) A = 33 X = 14 Y = FF P = 30 S = 0C *S 0313— 20 DA FD JSR \$FD DA A = 33 X = 14 Y = FF P = 30 S = 0C * </pre>	<p>Questo è stato fatto per essere sicuri che l'accumulatore sia azzerato</p> <p>L'accumulatore è caricato mediante indirizzamento indicizzato indiretto</p> <p>Stop qui</p>
--	--

Quest'ultimo programma non era molto interessante, vero? Questa volta useremo l'indirizzamento indicizzato indiretto per accedere a due differenti liste di dati. La prima lista consisterà di valori per il colore, la seconda conterrà righe a cui saranno visualizzate delle linee.

Useremo il mini-assembler per inserire il programma ed il monitor per inserire le liste di dati.

Iniziate con il mini-assembler.

*F666G

!300—:JSR FC58 ■ ← Indirizzo iniziale, due punti e prima istruzione

Premete RETURN

*F666G

0300— 20 58 FC JSR \$FC58
! JSR FB40 ■ ← Seconda istruzione

↑
Non dimenticate lo spazio

*F666G

0300— 20 58 FC JSR \$FC58

0303— 20 40 FB JSR \$FB40

! LDA #40 ← Terza istruzione

Continuate allo stesso modo finché tutte le istruzioni che seguono sono state inserite.

La lista completa delle istruzioni.

*F666G

!300:JSR FC58

! JSR FB40

! LDA #40

! STA 10

! LDA #50

! STA 12

! LDA #10

! STA 11

! STA 13

! LDY #5

! LDA #20

! STA 2C

! LDX #0

! LDA (10,X)

! STA 30

! LDX #2

! LDA (10,X)

! JSR F819

! INC 10

! INC 12

! LDA 10

! CMP #47

! BNE 314

! RTS

Valori iniziali dei
puntatori indicizzati

Punto iniziale della linea

Punto finale della linea

Stabilisce il colore

Stabilisce la riga

Disegna la linea

Incrementa i valori dei puntatori

Guarda se ha finito

CICLO

Quando avete inserito tutte le istruzioni, lo schermo apparirà così:

*F666G

```

0300— 20 58 FC      JSR $FC58
0303— 20 40 FB      JSR $FB40
0306— A9 40         LDA # $40
0308— 85 10         STA $10
030A— A9 50         LDA # $50
030C— 85 12         STA $12
030E— A9 10         LDA # $10
0310— 85 11         STA $11
0312— 85 13         STA $13
0314— A0 05         LDY # $05
0316— A9 20         LDA # $20
0318— 85 2C         STA $20
031A— A2 00         LDX # $00
031C— A1 10         LDA ($10,X)
031E— 85 30         STA $30
0320— A2 02         LDX # $02
0322— A1 10         LDA ($10,X)
0324— 20 19 F8      JSR $F819
0327— E6 10         INC $10
0329— E6 12         INC $12
032B— A5 10         LDA $10
032D— C9 47         CMP # $47
032F— D0 E3         BNE $0314
0331— 60           RTS

```

!■

Per inserire i dati, ritornate al monitor. Se tentate di inserire le liste di dati dall'assembler, verranno interpretate come istruzioni. E questo creerebbe una certa confusione.

```

.
.
.
0301— 60           RTS

```

!\$FF69G

*1040:FF 11 99 66 CC 33 DD ← colori

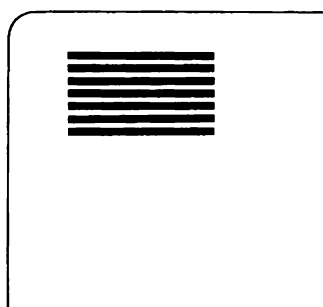
*1050:05 07 09 0B 0D 0F 11 ← righe

*■

Ora potete eseguire il programma



Lo schermo viene cancellato e le 8 linee colorate sono estratte dalla lista dei dati e visualizzate sullo schermo.

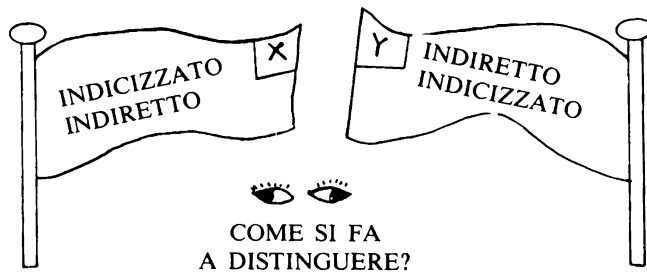


Ecco come vengono estratti i valori per disegnare le linee

	Memoria	Puntatore	
Se S = 0 →	0010	40	← Questi valori sono incrementati di 1 ogni volta che viene percorso il ciclo per indicare le locazioni per il colore e la riga
	0011	10	
Se X = 2 →	0012	50	←
	0013	10	
Colore	1040	FF	↓
	1041	11	
	1042	99	
	ecc.		
Riga	1050	05	↓
	1051	07	
	1052	09	
	ecc.		

 INDIRIZZAMENTO INDIRETTO INDICIZZATO

Questo modo differisce dall'indirizzamento indicizzato indiretto, anche se i loro nomi sono simili e possono essere confusi.



Questo tipo di indirizzamento usa il registro Y come indice. Useremo l'istruzione STA per illustrare come opera. Il formato è:

STA (oper), Y
 Codice op. 91
 2° byte, indirizzo di pagina zero

Il secondo byte dell'istruzione è un indirizzo di pagina zero che contiene un indirizzo *di base*. Il valore del registro Y viene sommato a questo indirizzo base per ottenere l'effettiva parte bassa dell'indirizzo dove deve essere memorizzato l'accumulatore. La locazione di pagina zero che segue quella data nel secondo byte contiene la parte alta dell'indirizzo.

Esempio

L'accumulatore contiene il valore 55
 La locazione di memoria 01 contiene il valore 32
 La locazione di memoria 02 contiene il valore 11
 Il registro Y contiene il valore 7
 L'istruzione da eseguire è:
 STA (01),Y

Primo, si prende il valore 32 dalla locazione 0001.

Secondo, viene sommato il contenuto del registro Y (7). Questo valore ($32 + 7 = 39$) è la parte bassa dell'indirizzo.

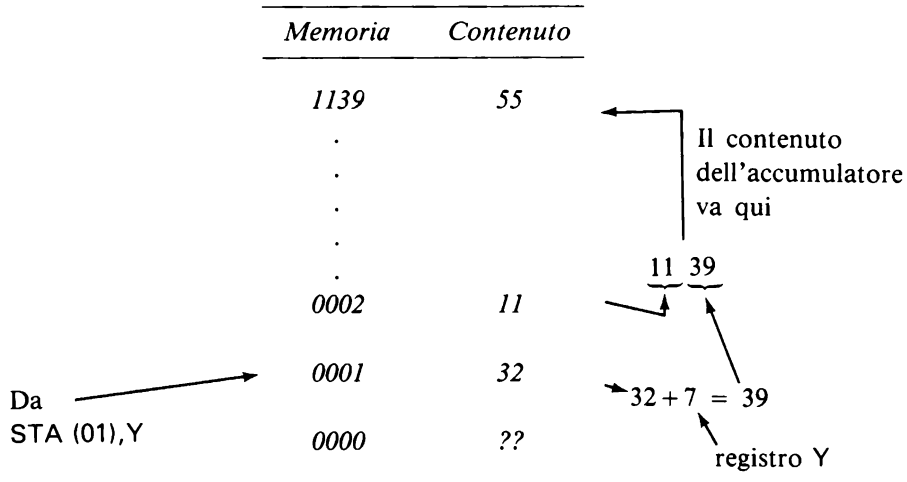
Terzo, il valore 11 contenuto nella locazione 0002 è usato come parte alta dell'indirizzo.

Così, il contenuto dell'accumulatore sarà memorizzato nella locazione 1139.

Nell'esempio:

Accumulatore 55

Viene eseguita l'istruzione STA(01),Y:



Ecco un breve programma dimostrativo che utilizza l'istruzione STA(oper),Y. Ancora una volta, mostreremo prima gli input.

!300:LDY #7	Carica 7 nel registro X
! LDA #32	Carica l'accumulatore 32
! STA 1	Lo memorizza in 0001
! LDA #11	Carica 11
! STA 2	Lo memorizza in 0002
! LDA #55	Mette 33 nell'accumulatore
! STA (01),Y	Carica 55
! RTS	Modo di indirizzamento indiretto indicizzato

Il video dopo l'inserimento delle istruzioni:

```

*F666G
0300— A0 07      LDY #07
0302— A9 32      LDA #32
0304— 85 01      STA $01
0306— A9 11      LDA #11
0308— 85 02      STA $02
030A— A9 55      LDA #55
030C— 91 01      STA ($01),Y
030E— 60         RTS
!$FF69G ←
*

```

Ancora una volta
scriviamo questo per
lasciare il mini-as-
sembler

Eseguite il programma. Non apparirà nulla. Quando appare il prompt alla fine del programma, esaminate la locazione 1139.

```

.
.
!$FF69G

*300G

*1139 ← Esaminate la memoria

1139— 55 ← Tutto bene, eccolo
*

```

Riguardate il programma Scala automatica nel Cap. 6. Modificheremo quel programma per ottenere una durata variabile delle note. Le durate ed i valori del tono saranno ottenuti da due liste di dati. Si accederà ad essi mediante istruzioni indirette indicizzate.

PROGRAMMA SCALA AUTOMATICA MODIFICATO

*F666G

0300—	20 58 FC	JSR \$FC58	
0303—	A0 00	LDY #0	
0305—	B1 03	LDA (03),Y	Carica la durata della lista
0307—	85 01	STA \$01	La memorizza
0309—	B1 05	LDA (05),Y	Carica l'intensità dalla lista
030B—	85 00	STA \$00	La memorizza
030D—	C8	INY	
030E—	98	TYA	Salva il contenuto di Y
030F—	48	PHA	
0310—	C0 09	CPY #\$09	Guarda se ha finito
0312—	FO 17ZBE-		
	Q \$032B	Se sì, va alla fine	
0314—	AD 30 C0	LDA \$C030	
0317—	88	DEY	
0318—	D0 04	BNE \$031E	Parte del programma che
031A—	C6 01	DEC \$01	pizzica l'altoparlante
031C—	F0 08	BEQ \$0326	
031E—	CA	DEX	
031F—	D0 F6 F8	BNE \$0317	
0321—	A6 00 03	LDX \$00	
0323—	4C 14 03	JMP \$0314	
0326—	68	PLA	Riprende il valore di X salvato
0327—	A8 03	TAY	
0328—	4C 05 03	JMP \$0305	Prende un'altra nota
032B—	68	PLA	
032C—	60	RTS	Ritorna al monitor

!\$FF69G

*1040:FF C8 80 40 C8 40 80 FF	Durata
*1050:85 7F 79 74 6F 6B 67 64	Intensità
*03:40 10 50 10	Puntatori per l'indirizzamento
*	indiretto indicizzato

All'inizio del programma i valori in memoria usati dall'indirizzamento indiretto indicizzato:

Memoria	03 = 40
	04 = 10

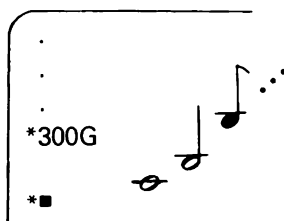
Memoria	05 = 50
	06 = 10

Il valore di Y viene sommato ad ognuna delle parti basse degli indirizzi (40 a 0003 e 50 a 0005) per trovare i valori corretti della durata e del tono per le note da suonare. Poiché il valore di Y viene incrementato ogni volta che si percorre il ciclo, ad ogni passaggio sarà suonato un nuovo tono con una nuova durata.

<i>Memoria</i>	<i>Durata</i>	<i>Memoria</i>	<i>Intensità</i>
1040	FF	1050	85
1041	C8	1051	7F
1042	80	1052	79
1043	40	1053	74
1044	C8	1054	6F
1045	40	1055	6B
1046	80	1056	67
1047	FF	1057	64

Cambiando una qualsiasi di queste locazioni, potete variare la tonalità e la durata. Con un po' d'immaginazione è possibile estendere queste liste di dati e suonare un intero pezzo con note di varia lunghezza. Se lo fate, non dimenticate di sostituire il valore contenuto nella locazione 0341, che deve rispecchiare il numero di note da suonare.

Quando eseguirete il programma, sarà suonata automaticamente la scala. In questo caso però, la lunghezza delle note varierà.



SOMMARIO

Congratulazioni! Ora siete programmatori in linguaggio assembler. Siete stati introdotti al mini-assembler dell'Apple ed avete esplorato alcuni suoi usi. Ora potete programmare in Basic, nel linguaggio macchina del

6502 ed in assembler. Noi speriamo che trascorrerete ancora un po' di tempo usando il mini-assembler. Riguardate alcuni dei programmi in linguaggio macchina apparsi all'inizio del libro ed inseriteli per mezzo dell'assembler. Tentate con qualche programma vostro. Più userete l'assembler, più abile diventerete, e più facile sarà utilizzarlo.

In questo capitolo avete imparato a:

1. passare dal mini-assembler al monitor e viceversa;
2. usare i codici mnemonici delle istruzioni del linguaggio macchina con il mini-assembler;
3. inserire istruzioni in assembler;
3. usare 4 modi di indirizzamento indicizzato:
 - a. indicizzato in pagina zero
 - b. assoluto indicizzato
 - c. indicizzato indiretto
 - d. indiretto indicizzato

Nuove istruzioni

1. LDA oper,X Caricamento dell'accumulatore con indirizzamento indicizzato in pagina zero. Carica l'accumulatore dall'indirizzo di pagina zero dell'operando, più il contenuto del registro X.
2. STA oper X Memorizzazione dell'accumulatore con indirizzamento indicizzato in pagina zero. Memorizza il contenuto dell'accumulatore nell'indirizzo di pagina zero dell'operando, più il contenuto del registro X.
3. LDA oper,X Caricamento dell'accumulatore assoluto indicizzato. Carica l'accumulatore dall'indirizzo completo dell'operando, più il contenuto del registro X (si può usare anche Y).
4. STA oper,X Memorizzazione dell'accumulatore assoluto indicizzato. Memorizza il contenuto dell'accumulatore nell'indirizzo completo dell'operando, più il contenuto del registro X (si può usare anche Y).
5. LDA (oper,X) Caricamento dell'accumulatore indicizzato indiretto. Carica l'accumulatore da una locazione di memoria che è indirizzata indirettamente dall'offset dell'operando, più il contenuto del registro X. Il risultato "punta" alle locazioni di pagina zero che contengono l'indirizzo effettivo.
6. STA (oper),Y Memorizzazione dell'accumulatore indiretta indicizzata. Memorizza il contenuto dell'accumulatore nella locazione che è ottenuta dalla memoria di pagina zero dell'operando (e pagina zero + 1). Il contenuto del registro Y viene sommato per ottenere la parte bassa dell'indirizzo effettivo. Pagina zero + 1 contiene la parte alta dell'indirizzo effettivo.
7. LDA (oper),Y Caricamento dell'accumulatore indiretto indicizzato.

Memorizza il valore dell'accumulatore nella locazione che è ottenuta dalla memoria di pagina zero dell'operando (e pagina zero + 1). Il contenuto del registro Y viene sommato per ottenere la parte bassa dell'indirizzo effettivo. Pagina zero + 1 contiene la parte alta dell'indirizzo effettivo.

Simboli e comandi del mini-assembler

1. F666G, per accedere al mini-assembler dal monitor
2. \$FF69G, per ritornare al monitor dal mini-assembler
3. !, prompt del mini-assembler

ESERCIZI

1. Qual è il comando esadecimale a quattro cifre per entrare nel mini-assembler?
* _____ G
2. Qual è il simbolo di prompt del mini-assembler?

3. Qual è il comando per lasciare il mini-assembler e ritornare al monitor di sistema:
!_____ G
4. Il seguente programma è stato inserito con il mini-assembler. Immaginatevi un'esecuzione e riportate il risultato.

*F666G

0300—	F8	SED
0301—	18	CLC
0302—	A9 17	LDA#\$17
0304—	69 78	ADC#\$78
0306—	20 DA FD	JSR \$FDDA
0309—	60	RTS
!\$FF69G		

* _____

_____ Mettete qui la vostra risposta

5. Un programma contiene le due seguenti istruzioni. Spiegare il risultato della loro esecuzione. Supponete che il registro X contenga 08.

Memoria	Codice op.	Codice dell'Assembler
0302	A9 22	LDA #22
0304	95 03	STA \$03,X

6. Dire che modo di indirizzamento è usato nelle seguenti istruzioni:
- STA\$03,X _____
 - STA\$1033,Y _____
 - LDA(05,X) _____
 - STA(07),Y _____
7. Quanti byte sono usati da ognuno dei seguenti tipi di istruzioni di indirizzamento?
- Assoluto indicizzato _____
 - Indicizzato in pagina zero _____
 - Indicizzato indiretto _____
8. Il programma alle pagine 240-244 vi permette di inserire 17 caratteri dalla tastiera. Indicate quali locazioni devono essere cambiate e quali modifiche si devono fare affinché il programma accetti 64 caratteri.

9. Supponiamo che il registro X contenga 9, il registro Y 8, e che i seguenti valori stiano nelle locazioni specificate:

Memoria	Contenuto
0012	10
0013	11
0014	12

viene eseguita l'istruzione: LDA (04,X).

Da quale locazione di memoria sarà caricato l'accumulatore?

10. Supponiamo di avere le stesse condizioni dell'Esercizio 9.
Viene eseguita questa istruzione: STA (12),Y
In che locazione verrà memorizzato l'accumulatore?

RISPOSTE AGLI ESERCIZI

1. F 6 6 6 G
 2. !
 3. ! \$ F F 6 9 G
 4. * 3 0 0 G
9 5
*
 5. Il valore 22 sarà caricato nell'accumulatore e poi memorizzato nella locazione 000B. ($0003 + 0008 = 000B$)
 6. a. Indicizzato in pagina zero c. indicizzato indiretto
b. Assoluto indicizzato d. Indiretto indicizzato
 7. a. 3 b. 2 c. 2
 8. Memoria Cambiare in
a. 030A 40 (30 ESA = 64 decimale)
b. 031A 40
 9. 1211 registro X = 9 più operando = 4 dà 13
la parte alta dell'indirizzo si ottiene da 0014 = 12
 10. 1118 10 dalla locazione 0012 (operando)
+ 8 dal registro Y
-
- 18 = parte bassa dell'indirizzo
11 dalla locazione (0012 + 1) = parte alta dell'indirizzo

Mettendo tutto assieme

BASIC

SOB

MS

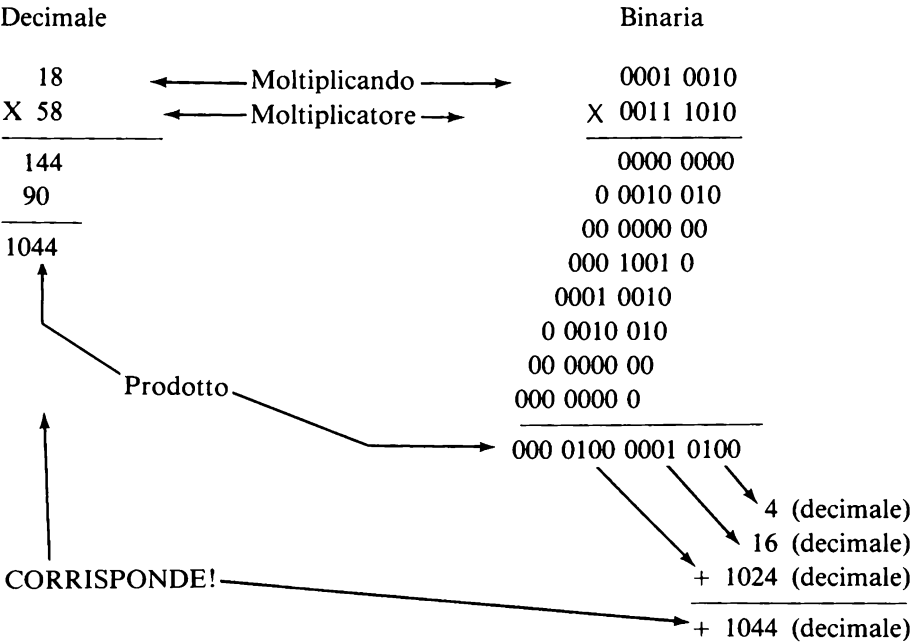
MA

In questo libro avete imparato quattro metodi per inserire ed eseguire programmi in linguaggio macchina.

1. Direttamente dal Basic, usando POKE, CALL e PEEK.
2. Mediante il sistema operativo in Basic
3. Direttamente in linguaggio macchina usando il monitor di sistema
4. Per mezzo del mini-assembler

In quest'ultimo capitolo, useremo ognuno di questi quattro metodi per inserire ed eseguire una moltiplicazione ad 8 bit. Il set di istruzioni del 6502 *non contiene* un'istruzione per la moltiplicazione o per la divisione. Ad ogni modo, ci sono molti modi per programmare queste due operazioni. Ne spiegheremo uno che è semplice e diretto.

Ricordate, il computer esegue la sua aritmetica usando numeri binari. Diamo prima un'occhiata ad un esempio di moltiplicazione binaria con carta e matita. Moltiplicheremo 18 (decimale) per 58 (decimale) usando sia la moltiplicazione decimale che quella binaria.



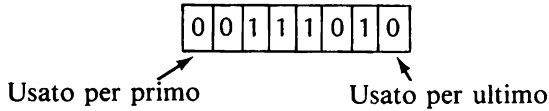
Notate che la nostra moltiplicazione implica la somma del moltiplicando ogni volta che compare un 1 nel moltiplicatore. Naturalmente, c'è uno *spostamento a sinistra* ogni volta che viene usato un bit del moltiplicatore, proprio come nella moltiplicazione decimale. Anche nella moltiplicazione binaria si procede da destra a sinistra man mano che si moltiplicano i bit del secondo numero.

Il programma che useremo fa praticamente la stessa cosa. La prima parte del programma inizializza le memorie con i valori appropriati. Metteremo queste quantità in memoria come segue:

<i>Indirizzo di memoria</i>	<i>Contenuto</i>
1000	Byte più significativo del prodotto
1001	Byte meno significativo del prodotto
1002	Moltiplicando (12)
1003	Moltiplicatore (3A)

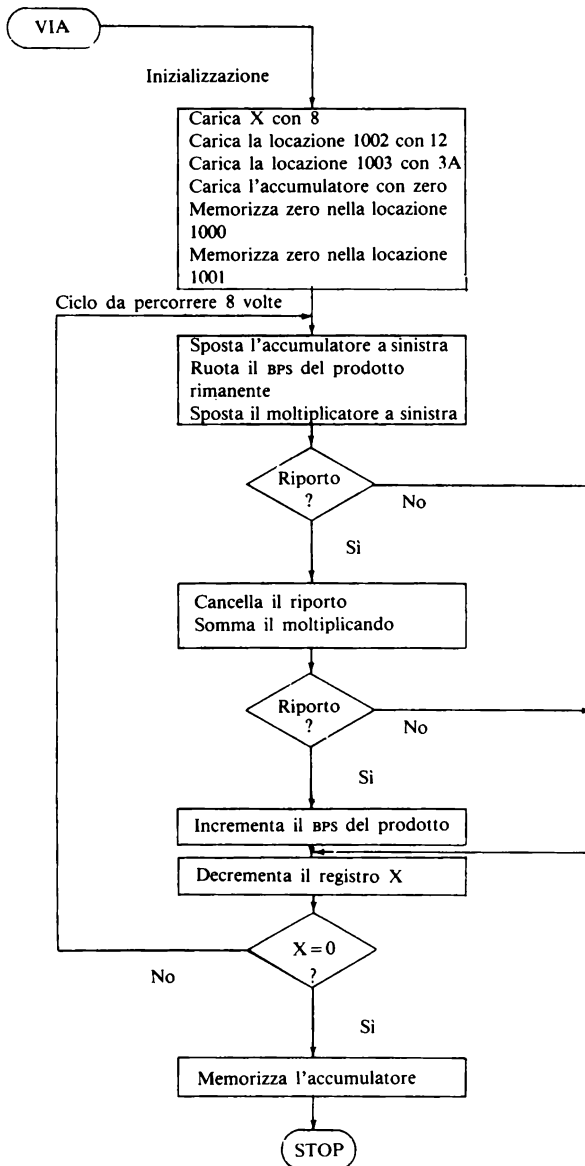
Durante l'esecuzione del programma, l'accumulatore conterrà tempora-

neamente il byte meno significativo del prodotto. Avrete notato che il moltiplicatore viene usato da sinistra a destra (l'opposto del metodo con carta e matita) per semplificare il procedimento.



Il moltiplicatore viene spostato a sinistra man mano che moltiplica.

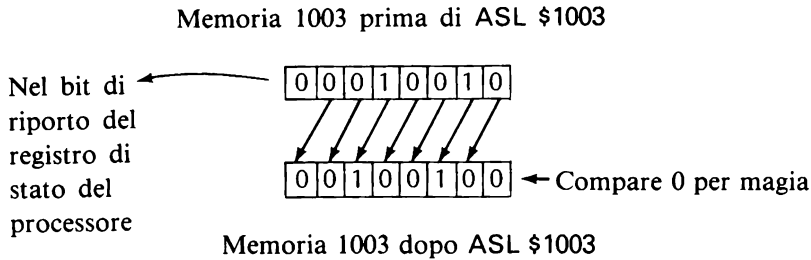
Ecco un diagramma di flusso per chiarire il programma.



La parte di inizializzazione è ora facile. Avete già usato ogni istruzione di questa parte del programma. Sono tutte istruzioni di caricamento o memorizzazione in vari modi di indirizzamento. La tavola che segue mostra i valori da inserire sia in forma decimale che esadecimale.

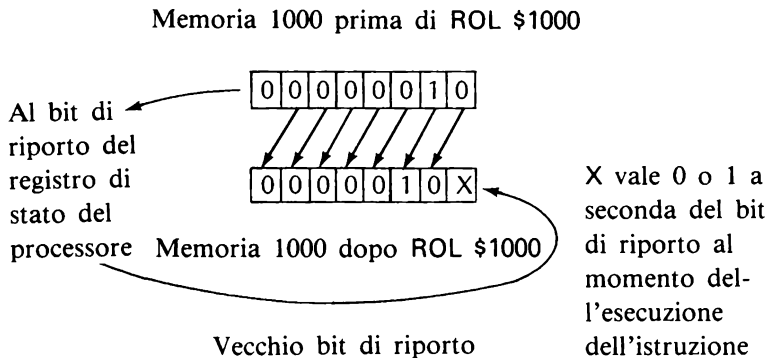
Indirizzo di memoria		Linguaggio macchina		Mnemonico assembler
Dec.	Esa	Dec.	Esa	
768	0300	162	A2	LDX #08
769	0301	8	08	
770	0302	169	A9	LDA #12
771	0303	18	12	
772	0304	141	8D	STA \$1002
773	0305	2	02	
774	0306	16	10	
775	0307	169	A9	LDA #3A
776	0308	58	3A	
777	0309	141	8D	STA \$1003
778	030A	3	03	
779	030B	16	10	
780	030C	169	A9	LDA #00
781	030D	0	00	
782	030E	141	8D	STA \$1000
783	030F	0	00	
784	0310	16	10	
785	0311	141	8D	STA \$1001
786	0312	1	01	
787	0313	16	10	

Il ciclo esegue la moltiplicazione. Compaiono due nuove istruzioni. Una di queste, *ASL (Arithmetic Shift Left*, spostamento aritmetico a sinistra), è stata discussa nel Cap. 3. Questa volta la usiamo, a 788 (decimale), per spostare il contenuto dell'accumulatore. La utilizziamo anche a 792 (decimale) nel modo di indirizzamento assoluto. Quando viene usata in questo modo, i bit della memoria specificata sono spostati a sinistra.

Esempio

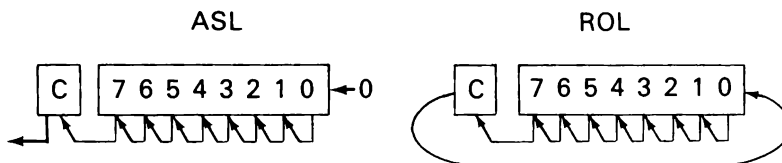
Sfrutteremo il fatto che il bit più significativo, in un'operazione di spostamento, viene portato nel bit di riporto del registro di stato del processore.

Un'altra nuova istruzione, ROL (*ROtate one bit Left*, ruota di un bit a sinistra), viene usata a 789 (decimale) nel modo di indirizzamento assoluto. Questa istruzione è simile ad ASL, ma differisce da questa per un particolare importante.

Esempio

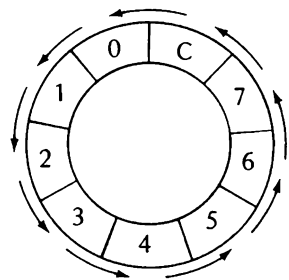
Durante l'esecuzione di ogni istruzione, il bit più significativo viene posto nel bit di riporto del registro di stato del processore. Nell'istruzione ASD il bit meno significativo viene azzerato. Nell'istruzione ROL invece, il bit meno significativo assume il valore del bit di riporto del registro di stato (0 o 1).

Il nome delle istruzioni deriva dall'azione che svolgono.



L'istruzione di spostamento aritmetico a sinistra non fa altro che spostare ogni bit a sinistra di una posizione. Nel bit più in basso viene messo zero, mentre il bit più in alto è messo nel bit di riporto. Il vecchio bit di riporto viene perso.

L'istruzione di rotazione a sinistra è come un cerchio. Ogni bit si muove di una posizione lungo il cerchio. Quindi, i bit vengono ruotati di un posto a sinistra.



Se ruotate a sinistra nove volte, tutto ritorna nella situazione iniziale. Se spostate a sinistra nove volte, avrete zero in ogni bit (incluso quello di riporto). La tavola che segue mostra i valori sia decimali che esadecimali da inserire per il ciclo e per uscire dal programma.

Indirizzo di memoria		Macchina		Assembler mnemonico
Dec.	Esa	Dec.	Esa	
788	0314	10	0A	ASL A
789	0315	46	2E	ROL \$1000
790	0316	0	00	
791	0317	16	10	
792	0318	14	0E	ASL \$1003
793	0319	3	03	
794	031A	16	10	
795	031B	144	90	BCC \$0326
796	031C	9	09	
797	031D	24	18	CLC

798	031E	109	6D	ADC \$1002
799	931F	2	02	
800	0320	16	10	
<hr/>				
801	0321	144	90	BCC \$0326
802	0322	3	03	
<hr/>				
803	0323	238	EE	INC \$1000
804	0324	0	00	
805	0325	16	10	
<hr/>				
806	0326	202	CA	DEX
<hr/>				
807	0327	208	D0	BNE \$0314
808	0328	235	EB	
<hr/>				
809	0329	141	8D	STA \$1001
810	032A	1	01	
811	032B	16	10	
<hr/>				
812	032C	96	60	RTS

 MOLTIPLICAZIONE DIRETTAMENTE DAL BASIC

Il programma in linguaggio macchina può essere messo in memoria direttamente dal Basic. Questo si otterrà con un ciclo FOR...NEXT contenente delle istruzioni READ e POKE. Il programma verrà poi eseguito con un'istruzione CALL, ed il risultato sarà visualizzato con un'istruzione. PRINT PEEK. Assicuratevi di essere in Basic Applesoft II.

MOLTIPLICAZIONE DAL BASIC

```

10 FOR M = 768 TO 812
20 READ D
30 POKE M,D
30 NEXT M
50 CALL 768
60 PRINT PEEK(4096)*256 + PEEK(4097)
70 END
80 DATA 162,8,169,18,141,2,16,169,58,141

```

```

90 DATA 3,16,169,0,141,0,16,141,1,16
100 DATA 10,46,0,16,14,3,16,144,9,24,109
110 DATA 2,16,144,3,238,0,16,202,208
120 DATA 235,141,1,16,96

```

Dopo che il programma è stato inserito, scrivete RUN per eseguirlo. Sarà visualizzato ciò che segue.

```

.
.
] RUN
1044 ← Il risultato è stato visualizzato
] ■

```

L'inserimento del programma direttamente dal Basic ha il vantaggio della velocità di inserimento. Un semplice ciclo FOR...NEXT mette tutti i codici di linguaggio macchina nelle loro rispettive locazioni. Il programma viene poi eseguito mediante un'istruzione CALL, ed il risultato è visualizzato con PRINT PEEK.

Tuttavia, ci sono degli svantaggi. Ogni istruzione ed indirizzo a cui si fa riferimento nel programma deve essere convertito dalla notazione esadecimale nel suo equivalente decimale prima che il programma possa essere inserito. Ricordate, i riferimenti al linguaggio macchina sono in codice esadecimale, ed il Basic usa numeri decimali. La conversione può richiedere del tempo se il programma è lungo. Il procedimento è soggetto a molti errori da parte del programmatore. Sebbene apparentemente piccoli, questi errori possono rivelarsi disastrosi nell'esecuzione del programma. Se vengono fatti degli errori, non c'è modo di correggere il programma direttamente, poiché, quando siamo in Basic, non stiamo usando il monitor di sistema dell'Apple.

MOLTIPLICAZIONE USANDO IL SISTEMA OPERATIVO IN BASIC

Quando viene utilizzato il sistema operativo in Basic, gli indirizzi vengono inseriti dal sistema (tranne l'indirizzo iniziale). I codici op. di linguaggio macchina sono inseriti in forma esadecimale direttamente dal manuale di riferimento.

Il sistema operativo deve essere caricato da una cassetta o inserito dalla

tastiera (vedere Cap. 27). Una volta eseguito, il programma richiede l'indirizzo iniziale ed il numero di byte. Il nostro indirizzo iniziale è 768, e ci sono 45 byte.

```
INDIRIZZO INIZIALE DEL PROGRAMMA = ?768  
QUANTI BYTE?45  
BATTI RETURN PER INSERIRE IL PROGRAMMA  
768 ■
```

Ora inserite il programma. Il computer terrà conto di ogni locazione di memoria e la stamperà in ordine. Tutto quello che dovete fare è inserire il codice esadecimale corretto (per i codici op. esadecimali da inserire, vedere le tavole mostrate precedentemente in questo capitolo). Dopo che il programma è stato inserito completamente, il sistema operativo visualizzerà i vostri input in blocchi di 20 byte, come segue.

ECCO IL TUO PROGRAMMA

```
768 A2  
769 08  
770 A9  
771 12  
772 8D  
773 02  
774 10  
775 A9  
776 3A  
777 8D  
778 03  
779 10  
780 A9  
781 00  
782 8D  
783 00  
784 10  
785 8D  
786 01  
787 10  
PREMI UN TASTO QUALSIASI PER CONTINUARE  
■
```

Dovete quindi controllare questa lista parziale per vedere se ci sono errori. Prendete nota delle correzioni da effettuare, ma non fatele finché non vi verrà chiesto alla fine dell'ultimo blocco. Ora siete pronti per passare al prossimo blocco di 20 (o meno) byte di dati.

```
788 0A
789 2E
790 00
791 10
792 0E
793 03
794 10
795 90
796 09
797 18
798 6D
799 02
800 10
801 90
802 03
803 EE
804 00
805 10
806 CA
807 D0
PREMI UN TASTO QUALSIASI PER CONTINUARE
■
```

Ora l'ultimo blocco di dati

```
808 EB
809 8D
810 01
811 10
812 60
PREMI UN TASTO QUALSIASI PER CONTINUARE
■
```

Ora, quando premete un tasto qualunque, avrete la possibilità di effettuare le vostre modifiche.

```

.
.
.
.
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?■

```

Se non ci sono modifiche, scrivete 99 e premete un tasto qualsiasi per l'esecuzione.

```

.
.
.
SE CI SONO MODIFICHE BATTI L'INDIRIZZO
ALTRIMENTI BATTI 99
?99
PREMI UN TASTO QUALSIASI PER PARTIRE ■

```

Quando il programma si sarà fermato, scrivete:

```
PRINT PEEK(4096)*256 + PEEK(4097)
```

```

]PRINT PEEK(4096)*256 + PEEK(4097)
1044
]■

```

Questo metodo elimina lo svantaggio di dover convertire i codici esadecimali in valori decimali per il Basic. Il sistema operativo lo fa per noi. L'utente deve soltanto inserire l'indirizzo iniziale, il numero di byte, ed ogni byte esadecimale di dati.

Naturalmente, dobbiamo prenderci la briga di caricare il sistema operativo ogni volta che lo vogliamo usare. Se il vostro sistema è fornito di dischi, questo non è un grosso problema. Il caricamento da registratore è più lento, e quello da tastiera è il più lento di tutti.

Il sistema operativo in Basic non è molto maneggevole, ma è servito allo scopo di introdurvi alla programmazione in linguaggio macchina con passi facili e divertenti per mezzo di comandi ed istruzioni del Basic.

 MOLTIPLICAZIONE USANDO IL MONITOR DI SISTEMA

Questo è il modo più diretto di inserire un programma in linguaggio macchina. Tutto quello che dovete fare è inserire l'indirizzo iniziale ed ogni valore esadecimale separato dagli altri da uno spazio. Sarà bene spezzare i programmi lunghi in parti di circa 3 linee fisiche di lunghezza. Questa volta useremo dati esadecimali.

```
*300:A2 08 A9 12 8D 02 10 A9 3A 8D 03 10
A9 00 8D 00 10 8D 01 10 0A 2E 00 10 0E
03 10 90 09 18 6D 02 10 90 03 EE 00 10 C
A D0 EB 8D 01 10 60
*
```

Per eseguire il programma, dovete semplicemente scrivere:

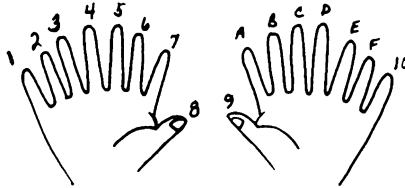
```
.
.
.
.
*300G
*
```

Per vedere il risultato, scrivete:

```
*300G
*1000.1001 ←
1000 — 04 14 ← 0414 ESA = 4 × 256 = 1024 decimale
*                                     1 × 16 = 16 decimale
                                     4 × 1 = 4 decimale
                                     -----
                                     1044
```

Questo metodo permette l'inserimento diretto di codici di macchina, un comando diretto per l'esecuzione, e un modo di esaminare le locazioni appropriate per vedere i risultati. La preparazione necessaria prima dell'inserimento è uno svantaggio. I codici mnemonici devono essere tra-

dotti. Tutte le destinazioni dei salti devono essere calcolate e convertite in valori esadecimali. Anche tutti i dati vanno inseriti in notazione esadecimale. I risultati sono in esadecimale, e devono essere convertiti in valori decimali affinché abbiano significato (a meno che non abbiate otto dita per mano).



A poche persone piace fare tutti questi calcoli matematici, cosicché, per creare programmi in linguaggio macchina, viene utilizzato un assembler.

MOLTIPLICAZIONE USANDO IL MINI-ASSEMBLER

Quando usa il mini-assembler, il programmatore ha soltanto a che fare con l'indirizzo iniziale ed i codici mnemonici delle istruzioni. I codici mnemonici sono molto più facili da ricordare di quelli esadecimali. Dopo averli usati un po', vedrete che non dovrete più cercarli. Poiché sono abbreviazioni degli effettivi nomi delle istruzioni, hanno un certo significato.

Per accedere al mini-assembler, scrivete: F666G

```
*F666G
```

```
!■
```

L'assembler risponde con il suo prompt

Inserite l'indirizzo iniziale, due punti ed il primo codice mnemonico.

```
*F666G
```

```
!300:LDX #8■
```

Il mini-assembler assembla questa istruzione e sostituisce la vostra linea

con l'indirizzo, il codice op. ed il codice mnemonico. Il cursore si sposta alla prossima linea, pronto per il successivo codice mnemonico ed un qualsiasi operando.

```
*F666G
0300—  A2 08      LDX #$08
!■
```

Continuate ad inserire i codici mnemonici con i loro operandi finché il programma sarà completo.

```
*F666G
0300—  A2 8        LDX #$08
0302—  A9 12       LDA #$12
0304—  8D 02 10    STA $1002
0307—  A9 3A       LDA #$3A
0309—  8D 03 10    STA $1003
030C—  A9 00       LDA #$00
030E—  8D 00 10    STA $1000
0311—  0D 01 10    STA $1001
0314—  0A          ASL8
0315—  2E 00 10    ROL $1000
0318—  0E 03 10    ASL $1003
031B—  90 09       BCC $0326
031D—  18          CLC
031E—  6D 02 10    ADC $1002
0321—  90 03       BCC $0326
0323—  EE 00 10    INC $1000
0326—  CA          DEX
0327—  D0 EB       BNE $0314
0329—  8D 01 10    STA $1001
032C—  60          RTS
!$FF69G ← Per tornare al monitor
*
```

Per eseguire il programma, usate il comando del monitor 300G.

```
*300G
```

```
*■
```


Per vedere il risultato, scrivete 1000.1001

.
 .
 .
 *300G
 *1000.1001
 1000— 04 14 ← Lo stesso risultato esadecimale
 * ■

La maggior parte delle persone che lavorano con il linguaggio macchina sono d'accordo che un assembler rimuove buona parte dei fastidi della programmazione in questo linguaggio. Non è possibile eseguire programmi dall'assembler, ma esso si prende il compito di assegnare i codici op. e gli operandi che li seguono.

Il mini-assembler si basa sul monitor di sistema per l'esecuzione del programma assemblato e nell'esame della memoria per vedere i risultati (sebbene una visualizzazione del risultato potesse essere inclusa nel programma). Avete visto ognuno dei quattro metodi in azione sullo stesso programma. Ricordatevi che abbiamo fornito i codici delle istruzioni necessari ed i dati per il programma. Provate questi metodi su un programma di vostra creazione e poi decidete che metodo preferite.

La combinazione del mini-assembler e del monitor di sistema è difficilmente battibile. Il mini-assembler fa tutto il lavoro di dettaglio, ed il monitor permette di correggere ed eseguire il programma.

DIVISIONE A 8 BIT

Anche questa volta, daremo un'occhiata alla divisione con carta e matita prima di vedere il metodo del calcolatore. Poiché la divisione è l'operazione inversa della moltiplicazione, useremo gli stessi numeri che abbiamo utilizzato nell'esempio relativo alla moltiplicazione, con una eccezione. Sceglieremo cioè il dividendo in modo che la divisione non sia esatta e si abbia un resto.

Esempio

1046 (decimale)

÷

58 (decimale)

Decimale

Binario

18 quoziente

18) 1046

58

466

464

2 resto

1 0010 quoziente

0011 1010) 0100 0001 0110

0011 1010

111 011

111 010

10 resto

Corrisponde

1 0010 = 12 ESA = 18 dec.

10 = 2 ESA = 2 dec.

Proprio come nella moltiplicazione, ci saranno degli spostamenti di bit mentre viene eseguita la divisione. In questo procedimento il valore di posizione è molto importante. Notate che la sottrazione viene fatta solo se il divisore è minore della parte di dividendo attualmente in esame. In questo caso, nel quoziente viene posto un 1. Se il divisore è maggiore, viene messo uno 0 nel quoziente. Il nostro programma in linguaggio macchina agir  praticamente nello stesso modo. La prima parte del programma mette i valori appropriati nelle loro rispettive locazioni di memoria. I valori sono memorizzati come segue:

<i>Indirizzo di memoria</i>	<i>Contenuto</i>
1000	Byte pi� significativo del dividendo
1001	Byte meno significativo del dividendo
1002	Resto

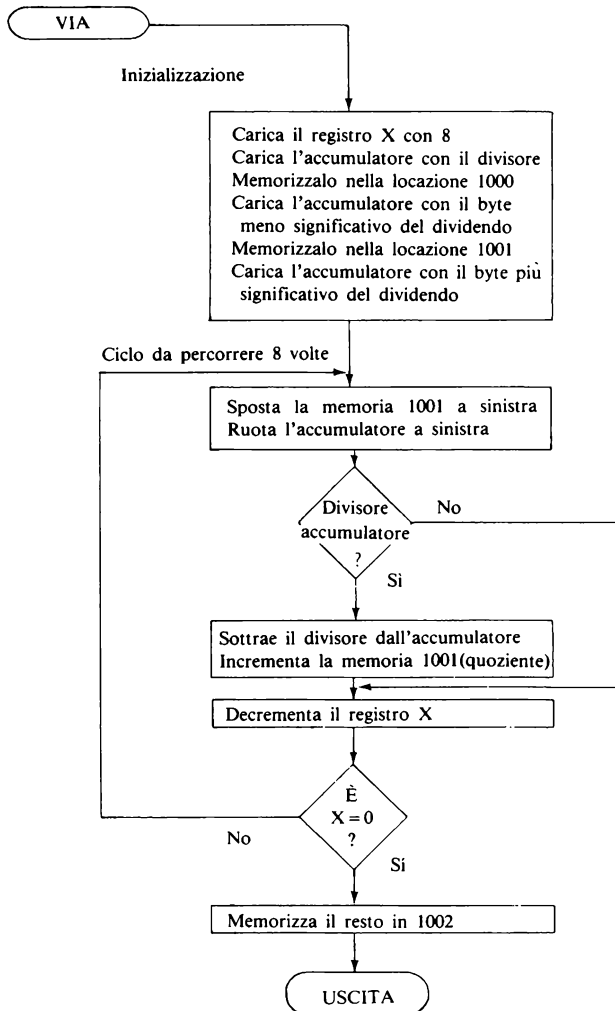
L'accumulatore e la locazione 1001 contengono il dividendo originale. Ogni volta che viene percorso il ciclo, il contenuto della locazione 1001 viene spostato a sinistra e l'accumulatore   ruotato a sinistra. Se, da questa azione sulla memoria 1001 si ottiene un riporto, esso apparir  come un 1 nel bit meno significativo dell'accumulatore quando questo sar  ruotato. Se invece non si ha riporto, nel bit meno significativo dell'accumulatore comparir  0. Cos , ogni volta che si percorre il ciclo, il dividendo viene spostato di una posizione dalla locazione 1001 nell'accumu-

latore. Questo permette al computer di confrontare il divisore con la parte più significativa del dividendo per una prova di divisione.

La divisione viene eseguita confrontando il divisore con l'accumulatore. Ogni volta che il divisore è minore o uguale dell'accumulatore, il divisore viene sottratto dall'accumulatore e la memoria 1001 è incrementata di uno.

Questo significa che comparirà 1 nel quoziente.

Se il divisore è maggiore dell'accumulatore, non viene effettuata la sottrazione, e la locazione 1001 non è incrementata. Questo significa che apparirà 0 nel quoziente. Quando i bit dell'accumulatore e della memoria sono spostati a sinistra, il quoziente appare nella memoria 1001 da destra (0 se il divisore è maggiore del dividendo, 1 altrimenti).



Dopo aver completato il ciclo (che è stato percorso 8 volte), il resto viene posto nella locazione 1002. Alla fine dell'esecuzione, troverete il quoziente di 8 bit nella locazione 1001 ed il resto nella memoria 1002. L'inizializzazione è simile a quella per il programma per la moltiplicazione.

Indirizzo di memoria		Macchina		Assembler mnemonico
Dec.	Esa	Dec.	Esa	
768	0300	162	A2	LDX 08
769	0301	8	08	
770	0302	169	A9	LDA 3A
771	0303	58	3A	
772	0304	141	8D	STA \$1000
773	0305	0	00	
774	0306	16	10	
775	0307	169	A9	LDA 16
776	0308	29	16	
777	0309	141	8D	STA \$ 1001
778	030A	1	01	
779	030B	16	10	
780	030C	169	A9	LDA 4
781	030D	4	04	

L'istruzione di rotazione appare, nel ciclo di questo programma, in una nuova forma. Abbiamo usato ROL A per ruotare l'accumulatore. Questa istruzione lavora nello stesso modo che nel caso della rotazione di una memoria, ma questa volta è il contenuto dell'accumulatore che viene ruotato. Il ciclo è percorso ancora 8 volte usando il registro X come contatore.

Ora la parte relativa alla divisione.

Indirizzo di memoria		Macchina		Assembler mnemonico
Dec.	Esa	Dec.	Esa	
782	030E	14	0E	ASL \$1001
783	030F	1	01	
784	0310	16	10	
785	0311	42	2A	ROL A
786	0312	199	CD	CMP \$1000
787	0313	0	00	
788	0314	16	10	
789	0315	144	90	BCC \$031D
790	0316	6	06	
791	0317	237	ED	SBC \$1000
792	0318	0	00	
793	0319	16	10	
794	031A	238	EE	INC \$1001
795	031B	1	01	
796	031C	16	10	
797	031D	196	CA	DEX
798	031E	208	D0	BNE \$030E
799	031F	239	EF	
800	0320	141	8D	STA \$1002
801	0321	2	02	
802	0322	16	10	
803	0323	96	60	RTS

Scegliete quale metodo volete usare per inserire ed eseguire il programma:

1. Basic
2. Sistema operativo in Basic
3. Monitor di sistema
4. Assembler

Noi useremo l'assembler.

*F666G		
0300—	A2 08	LDX #\$08
0302—	A9 3A	LDA #\$3A
0304—	8D 00 10	STA \$1000
0307—	A9 16	LDA #\$16
0309—	8D 01 10	STA \$1001
030C—	A9 04	LDA #\$04
030E—	0E 01 10	ASL \$1001
0311—	2A	ROL
0312—	CD 00 10	CMP \$1000
0315—	90 06	BCC \$031D
0317—	ED 00 10	SBC \$1000
031A—	EE 01 10	INC \$ 1001
031D—	CA	DEX
031E—	D0 EF	BNE \$030E
0320—	8D 02 10	STA \$1002
0323—	60	RTS
!\$FF69G		
*300G		
*1001.1002		
1001—	12 02	Resto
*		Quoziente

Questo conclude il capitolo, ed anche l'esplorazione del linguaggio macchina in questo libro. Ci sono ancora molte cose da imparare. Alcune istruzioni non sono state discusse; ma con le basi che ora avete, sarete in grado di scoprire tutto il resto da soli. La lista completa delle istruzioni è data nell'Appendice D. Queste sono anche riportate, assieme ad una descrizione del loro uso, nell'*MCS6500 Microcomputer Family Programming Manual*.

SOMMARIO

In questo capitolo avete avuto la possibilità di confrontare quattro modi di produrre un programma in linguaggio macchina:

1. Direttamente dal Basic

2. Dal sistema operativo in basic
3. Dal monitor di sistema dell'Apple
4. Dal mini-assembler

Ogni metodo ha vantaggi e svantaggi. Per spiegare questi metodi abbiamo usato degli esempi relativi alla moltiplicazione e alla divisione.

Avete imparato che

1. La moltiplicazione di due numeri di 8 bit dà un risultato di 16 bit.
2. Il computer può moltiplicare spostando e ruotando i dati e sommando il moltiplicatore in un modo predeterminato.
3. Ognuno dei quattro metodi fornisce risultati equivalenti.
4. Tutti i metodi, tranne l'assembler, richiedono la codifica delle istruzioni e/o la trasformazione di numeri da una base in un'altra.
5. Tra i quattro metodi, l'assembler è quello che richiede la fase preparatoria più breve — esso assembla un programma in linguaggio macchina facendo tutti i calcoli necessari e scegliendo i codici op. in base a quelli mnemonici da voi forniti.
6. Il computer esegue la divisione in modo simile al metodo usuale con carta e matita.
7. La divisione di un dividendo di 16 bit per un divisore di 8 bit produce un quoziente di 8 bit con un resto.

Nuove istruzioni

- ROL oper** Un'istruzione ad indirizzamento assoluto che ruota ogni bit della locazione specificata di un posto a sinistra. Il bit più significativo si sposta nel bit di riporto, mentre quest'ultimo finisce nel bit meno significativo della memoria specificata.
- ASL oper** Un'istruzione ad indirizzamento assoluto che sposta ogni bit nella locazione specificata di un posto a sinistra. Il bit più significativo si sposta nel bit di riporto, e nel bit meno significativo della memoria specificata viene posto zero.
- ROL A** Ruota i bit dell'accumulatore di una posizione a sinistra. Il bit più significativo si sposta nel bit di riporto, e quest'ultimo finisce nel bit meno significativo dell'accumulatore.

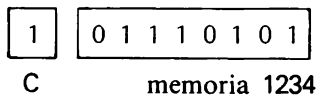
 ESERCIZI

1. Scrivete i quattro metodi spiegati per l'inserimento di programmi in linguaggio macchina.
- _____
 - _____
 - _____
 - _____

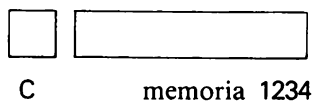
2. Spiegare che cosa succede nel bit di riporto dopo l'esecuzione dell'istruzione ASL A
Il bit di riporto _____

3. Spiegare il registro X è stato posto uguale a 8 all'inizio del programma di moltiplicazione.
- _____
- _____
- _____

4. Supposto che la memoria 1234 contenga ciò che segue e che il bit di riporto sia posto a uno,



mostrare che cosa conterranno dopo l'esecuzione dell'istruzione ROL \$1234



5. In che locazioni di memoria si troverà il prodotto nella moltiplicazione dal programma in Basic?
_____ (decimale) e _____ (decimale)
6. Se un programma in linguaggio macchina è lungo 50 byte ed è inserito per mezzo del sistema operativo, i dati verranno visualizzati in blocchi di lunghezza specificata. Quanto blocchi saranno necessari per visualizzare tutto il programma?
- _____

7. Se, per inserire un programma in linguaggio macchina, viene usato il monitor di sistema, che sistema numerico (decimale, esadecimale o binario) deve essere usato?
- _____

8. Che effetto ha il comando *F666G?
- _____
- _____

9. Che cosa sarà visualizzato dopo l'inserimento di *F666G?

*F666G

10. Se volete lasciare il mini-assembler e ritornare al monitor di sistema, che comando dovete dare?

.

.

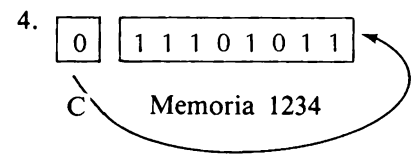
.

! _____

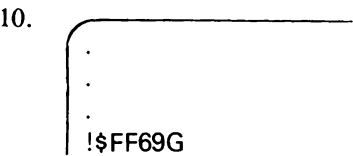
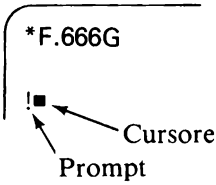
11. Che locazione di memoria viene usata per contenere il quoziente nel programma di Divisione?
- _____ (ESA)

RISPOSTE AGLI ESERCIZI

1. a. Direttamente dal Basic usando POKE, CALL e PEEK
- b. Il sistema operativo in Basic
- c. Il monitor di sistema
- d. Il mini-assembler
2. Il bit di riporto (originale) è perso. Viene sostituito dal bit più significativo dell'accumulatore.
3. Il registro X è usato per contare il numero di volte che viene percorso il ciclo. Poiché stiamo moltiplicando per un numero di 8 bit, il ciclo è percorso 8 volte.



- 5. 4097 e 4096 (I byte più e meno significativi, rispettivamente)
- 6. 3 (Il video mostra 20 linee alla volta)
- 7. Esadecimale
- 8. Si accede al mini-assembler dal monitor
- 9.



- 11. 1001 (ESA)

Appendici

L'Appendice A contiene informazioni che sono state usate in questo libro con riferimento alla pagina dove appaiono per la prima volta o dove vengono spiegate.

L'Appendice B contiene una tavola di conversione dei numeri decimali negativi nei loro equivalenti esadecimali.

L'Appendice C è una spiegazione della mappa di memoria usata per il video. Viene anche data una tavola di codici usati per visualizzare caratteri sullo schermo e una tavola dei colori utilizzati per la visualizzazione di grafici. L'Appendice D è una lista dei codici delle istruzioni del 6502.

A-1 ISTRUZIONI DEL BASIC

<i>Istruzione</i>	<i>Pagina</i>	<i>Istruzione</i>	<i>Pagina</i>
ASC	26	GOSUB	22
CALL	28, 38	GOTO	21
CHR\$	26	GR	13, 23
COLOR	23	HLIN	23
CONT	14	HOME	20
DATA	15	IF...THEN	22
FLASH	19	INPUT	15
FOR-NEXT	22	INT	27
GET	17	INVERSE	20

LEFT\$	26	POKE	27,37
LET	15	PRINT	18
LIST	12	READ	15
LOAD	13	RESTORE	17
NEW	11	RETURN	23
NORMAL	20	RUN	12
NOTRACE	14	SAVE	14
ON...GOTO	21	SPC	20
PDL	24	TEXT	13, 24
PEEK	29, 38	TRACE	14
PLOT	23	VLIN	24

A-2 ISTRUZIONI DEL LINGUAGGIO MACCHINA

<i>Codice mnemonico</i>	<i>Modo di indirizzam.</i>	<i>Pagina</i>	<i>Codice mnemonico</i>	<i>Modo di indirizzam.</i>	<i>Pagina</i>
ADC	Assoluto	187, 202	LDA	Indic. Ind.	276
ADC	Immediato	188	LDA	Ind. Indic.	266
AND	Immediato	171, 174	LDA	Pagina zero	228
ASL	Assoluto	297	LDA	Ind. p. zero	267
ASL	Accumul.	80	LDX	Immediato	124, 138
BCC	Relativo	194, 202	LDX	Pagina zero	147, 159
BCS	Relativo	192, 202	LDY	Ass. indic.	179
BEQ	Relativo	136, 139	LDY	Immediato	88, 108
BMI	Relativo	173	NOP	Implicito	172
BNE	Relativo	115, 117, 138	PHA	Implicito	155, 159
BPL	Relativo	171, 174	PLA	Implicito	155, 159
CLC	Implicito	188, 202	ROL	Assoluto	297
CLD	Implicito	238	ROL	Accumulatore	310
CMP	Assoluto	136, 139	RTS	Implicito	73
CPX	Immediato	132, 138	SBC	Assoluto	228
CPY	Immediato	115, 117, 138	SBC	Immediato	187
DEC	Implicito	147, 158	SEC	Implicito	200, 202
DEX	Implicito	147, 159	SED	Implicito	238
DEY	Implicito	147	STA	Assoluto	68, 72
INX	Implicito	124, 138	STA	Ass. Indic.	113, 139
INY	Implicito	115, 117, 138	STA	Ind. Indic.	284
JMP	Assoluto	137, 139	STA	Pagina zero	88, 108
JSR	Assoluto	86, 107	STA	Ind. p. zero	268
LDA	Assoluto	196, 202	TAX	Implicito	152, 159
LDA	Ass. ind.	124, 139	TAY	Implicito	254
LDA	Immediato	71	TXA	Implicito	142, 159

A-3 SUBROUTINE INTERNE

<i>Indirizzo di memoria</i>	<i>Funzione eseguita</i>	<i>Pagina</i>
F800	Disegna un punto	88
F819	Disegna una linea orizzontale	98
F828	Disegna una linea verticale	101
FB40	Abiliga il modo grafico	88
FC58	Cancella lo schermo	87
FD1B	Numero casuale	217
FD35	Riceve un carattere	127
FD8E	Ritorno del cursore	228
FDDA	Visualizza l'accumulatore (ESA)	128
FDED	Visualizza l'accumulatore	127
FF3A	Suona il campanello	221

LDA C030 non è realmente una subroutine, ma si comporta similmente. Essa pizzica l'altoparlante.

A-4 SIMBOLI

<i>Simboli</i>	<i>Funzione</i>
*	Prompt del monitor di sistema — usato per il linguaggio macchina
]	Prompt dell'Applesoft II — usato per il Basic Applesoft II
>	Prompt dell'Integer Basic — usato per l'Integer Basic
!	Prompt del mini-assembler — usato per assemblare programmi in linguaggio macchina
■	Cursore — indica la successiva posizione di stampa da usare (lampeggia

A-5 PROGRAMMI

<i>Titolo</i>	<i>Linguaggio</i>	<i>Pagina</i>
Addizione a due byte	L/M	207
Alfabeto	L/M	124
Carica l'accumulatore e memorizza	L/M	67
Disegna nei quattro angoli	L/M	94,96
Disegna un punto	L/M	87,91
Disegna un rettangolo	L/M	105
Divisione	L/A	312
Esperimenti con le note	L/M	144
Indirizzamento indiretto indicizzato	L/A	285
Indirizzamento indicizzato indiretto	L/A	278
Indovina il numero	L/M	216
Linea verticale	L/M	101
Linea orizzontale	L/M	98,99
Linee colorate	L/A	282
Memorizza dati dalla tastiera	L/M	282
Moltiplicazione dal monitor di sistema	L/M	304
Moltiplicazione dal mini-assembler	L/A	306
Moltiplicazione dal sistema operativo in Basic	L/M	301
Moltiplicazione dal Basic	L/M	299
POKE, PEEK e poi ancora POKE	Basic	39
Scala con note	L/M	164
Scala automatica	L/A	287
Scala automatica	L/M	153
Sistema operativo in Basic	Basic	54
Somma due numeri decimali	L/A	264
Somma due numeri decimali	L/M	239
Somma due numeri	L/M	194
Sottrazione di due numeri	L/M	200
Sottrazione a due byte	L/M	209
Suona la tua melodia	L/M	170
Uso di POKE e PEEK	Basic	37
Visualizzazione su più linee	L/M	131
Visualizza un messaggio	L/M	136
Visualizzazione di codici ASCII	L/M	126
Visualizza un carattere	L/M	113
Visualizza una lettera	L/M	112

L/M = linguaggio macchina

L/A = linguaggio assembler

Basic = Basic Applesoft II

B EQUIVALENTI ESADECIMALI DEI NUMERI DECIMALI NEGATIVI

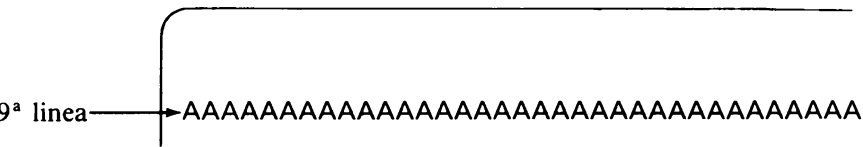
<i>Dec.</i>	<i>ESA</i>	<i>Dec.</i>	<i>ESA</i>	<i>Dec.</i>	<i>Esa</i>
-1	FF	-44	D4	-87	A9
-2	FE	-45	D3	-88	A8
-3	FD	-46	D2	-89	A7
-4	FC	-47	D1	-90	A6
-5	FB	-48	D0	-91	A5
-6	FA	-49	CF	-92	A4
-7	F9	-50	CE	-93	A3
-8	F8	-51	CD	-94	A2
-9	F7	-52	CC	-95	A1
-10	F6	-53	CB	-96	A0
-11	F5	-54	CA	-97	9F
-12	F4	-55	C9	-98	9E
-13	F3	-56	C8	-99	9D
-14	F2	-57	C7	-100	9C
-15	F1	-58	C6	-101	9B
-16	F0	-59	C5	-102	9A
-17	EF	-60	C4	-103	99
-18	EE	-61	C3	-104	98
-19	ED	-62	C2	-105	97
-20	EC	-63	C1	-106	96
-21	EB	-64	C0	-107	95
-22	EA	-65	BF	-108	94
-23	E9	-66	BE	-109	93
-24	E8	-67	BD	-110	92
-25	E7	-68	BC	-111	91
-26	E6	-69	BB	-112	90
-27	E5	-70	BA	-113	8F
-28	E4	-71	B9	-114	8E
-29	E3	-72	B8	-115	8D
-30	E2	-73	B7	-116	8C
-31	E1	-74	B6	-117	8B
-32	E0	-75	B5	-118	8A
-33	DF	-76	B4	-119	89
-34	DE	-77	B3	-120	88
-35	DD	-78	B2	-121	87
-36	DC	-79	B1	-122	86
-37	DB	-80	B0	-123	85
-38	DA	-81	AF	-124	84
-39	D9	-82	AE	-125	83
-40	D8	-83	AD	-126	82
-41	D7	-84	AC	-127	81
-42	D6	-85	AB	-128	80
-43	D5	-86	AA		

C-1 MEMORIA VIDEO

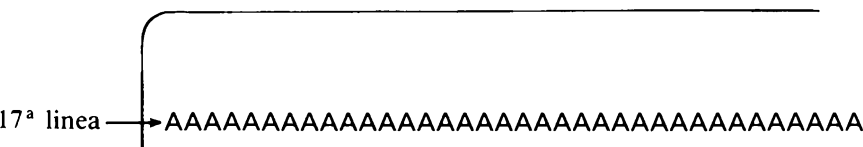
Nel Cap. 5 avete scoperto che la linea più in alto del video (linea 1) è associata alle locazioni di memoria comprese fra 0400 e 0427 esadecimali. Potreste pensare che la prossima linea sia associata alle locazioni comprese tra 0428 e 044F, ma non è così. Se ritornate al programma Visualizza caratteri nel Cap. 4 e cambiate l'istruzione alle locazioni 775, 776 e 777 con: STA 0428,Y il programma memorizzerà una stringa di A in quelle locazioni.

```
775 99      STA 0428,Y
776 28
777 04
```

Ecco dove saranno stampate le A se il programma verrà eseguito con questi valori.



Se il valore memorizzato nella locazione 776 venisse cambiato a 50, dove pensate apparirebbe la stringa di A? Se avete detto la diciassettesima linea, è esatto.



Finora abbiamo trovato le linee 1, 9 e 17.

linea 1	0400	0427
linea 9	0428	044F
linea 17	0450	0477

A questo punto, potreste aver indovinato quali locazioni di memoria sono associate alle 24 linee del video.

Linea 1	0400	→	0427
Linea 2	0480	→	04A7
Linea 3	0500	→	0527
Linea 4	0580	→	05A7
Linea 5	0600	→	0627
Linea 6	0680	→	06A7
Linea 7	0700	→	0727
Linea 8	0780	→	07A7
Linea 9	0428	→	044F
Linea 10	04A8	→	04CF
Linea 11	0528	→	054F
Linea 12	05A8	→	05CF
Linea 13	0628	→	064F
Linea 14	06A8	→	06CF
Linea 15	0728	→	074F
Linea 16	07A8	→	07CF
Linea 17	0450	→	0477
Linea 18	04D0	→	04F7
Linea 19	0550	→	0577
Linea 20	05D0	→	05F7
Linea 21	0650	→	0677
Linea 22	06D0	→	06F7
Linea 23	0750	→	0777
Linea 24	07D0	→	07F7

C-2 CODICI ASCII DI SCHERMO

<i>Car.</i>	<i>Nor.</i>	<i>Lamp.</i>	<i>Inv.</i>	<i>Car.</i>	<i>Nor.</i>	<i>Lamp.</i>	<i>Inv.</i>
	A0	60	20	@	C0	40	00
!	A1	61	21	A	C1	41	01
"	A2	62	22	B	C2	42	02
#	A3	63	23	C	C3	43	03
\$	A4	64	24	D	C4	44	04
%	A5	65	25	E	C5	45	05
&	A6	66	26	F	C6	46	06
'	A7	67	27	G	C7	47	07
(A8	68	28	H	C8	48	08
)	A9	69	29	I	C9	49	09
*	AA	6A	2A	J	CA	4A	0A
+	AB	6B	2B	K	CB	4B	0B
,	AC	6C	2C	L	CC	4C	0C
-	AD	6D	2D	M	CD	4D	0D
.	AE	6E	2E	N	CE	4E	0E
/	AF	6F	2F	O	CF	4F	0F
0	B0	70	30	P	D0	50	10
1	B1	71	31	Q	D1	51	11
2	B2	72	32	R	D2	52	12
3	B3	73	33	S	D3	53	13
4	B4	74	34	T	D4	54	14
5	B5	75	35	U	D5	55	15
6	B6	76	36	V	D6	56	16
7	B7	77	37	W	D7	57	17
8	B8	78	38	X	D8	58	18
9	B9	79	39	Y	D9	59	19
:	BA	7A	3A	Z	DA	5A	1A
;	BB	7B	3B	[DB	5B	1B
<	BC	7C	3C	\	DC	5C	1C
=	BD	7D	3D]	DD	5D	1D
>	BE	7E	3E	^	DE	5E	1E
?	BF	7F	3F	_	DF	5F	1F

Car. = carattere

Nor. = modo normale

Lamp. = modo lampeggiante

Inv. = modo invertito

C-3 CODICI DEI COLORI PER GRAFICI IN BASSA RISOLUZIONE

<i>Decimale</i>	<i>ESA</i>	<i>Colore</i>
0	0	Nero
1	1	Magenta
2	2	Blu scuro
3	3	Viola
4	4	Verde scuro
5	5	Grigio
6	6	Blu
7	7	Azzurro
8	8	Marrone
9	9	Arancio
10	A	Grigio
11	B	Rosa
12	C	Verde
13	D	Giallo
14	E	Acquamarina
15	F	Bianco

I colori possono variare leggermente a seconda dell'apparecchio televisivo.

D CODICI DELLE ISTRUZIONI DEL 6502

<i>Codice mnemon.</i>	<i>Modo di indirizzam.</i>	<i>In assembler</i>	<i>Cod. op</i>	<i>N. di byte</i>	<i>Flag interessati</i>
ADC	Immediato	ADC oper	69	2	NZCV
	Pagina zero	ADC oper	65	2	
	Pag. zero,X	ADC oper,X	75	2	
	Assoluto	ADC oper	6D	3	
	Assoluto,X	ADC oper,X	7D	3	
	Assoluto,Y	ADC oper,Y	79	3	
	(Indiretto,X)	ADC (oper,X)	61	2	
	(Indiretto),Y	ADC (oper),Y	71	2	
AND	Immediato	AND oper	29	2	NZ
	Pagina zero	AND oper	25	2	
	Pag. zero,X	AND oper,X	35	2	
	Assoluto	AND oper	2D	3	
	Assoluto,X	AND oper,X	3D	3	
	Assoluto,Y	AND oper,Y	39	3	
	(Indiretto,X)	AND (oper,X)	21	2	
	(Indiretto),Y	AND (oper),Y	31	2	
ASL	Accumulatore	ASL A	0A	1	NZC
	Pagina zero	ASL oper	06	2	
	Pag. zero,X	ASL oper,X	16	2	
	Assoluto	ASL oper	0E	3	
	Assoluto,X	ASL oper,X	1E	3	
BCC	Relativo	BCC oper	90	2	nessuno
BCS	Relativo	BCS oper	B0	2	nessuno
BEQ	Relativo	BEQ oper	F0	2	nessuno
BIT	Pagina zero	BIT oper	24	2	NZV
	Assoluto	BIT oper	2C	3	
BMI	Relativo	BMI oper	30	2	nessuno
BNE	Relativo	BNE oper	D0	2	nessuno
BPL	Relativo	BPL oper	10	2	nessuno

<i>Codice mnemon.</i>	<i>Modo di indirizzam.</i>	<i>In assembler</i>	<i>Cod. op</i>	<i>N. di byte</i>	<i>Flag interessati</i>
BRK	Implicito	BRK	00	1	I
BVC	Relativo	BVC oper	50	2	nessuno
BVS	Relativo	BVS oper	70	2	nessuno
CLC	Implicito	CLC	18	1	C
CLD	Implicito	CLD	D8	1	D
CLI	Implicito	CLI	58	1	I
CLV	Implicito	CLV	B8	1	V
CMP	Immediato	CMP #oper	C9	2	NCZ
	Pagina zero	CMP oper	C5	2	
	Pag. zero,X	CMP oper,X	D5	2	
	Assoluto	CMP oper	CD	3	
	Assoluto,X	CMP oper,X	DD	3	
	Assoluto,Y	CMP oper,Y	D9	3	
	(Indiretto,X)	CMP (oper,X)	C1	2	
	(Indiretto),Y	CMP (oper),Y	D1	2	
CPX	Immediato	CPX #oper	E0	2	NZC
	Pagina zero	CPX oper	E4	2	
	Assoluto	CPX oper	EC	3	
CPY	Immediato	CPY #oper	C0	2	NZC
	Pagina zero	CPY oper	C4	2	
	Assoluto	CPY oper	CC	3	
DEC	Pagina zero	DEC #oper	C6	2	NZ
	Pag. zero,X	DEC oper,X	D6	2	
	Assoluto	DEC oper	CE	3	
	Assoluto,X	DEC oper,X	DE	3	
DEX	Implicito	DEX	CA	1	NZ
DEY	Implicito	DEY	88	1	NZ

<i>Codice mnemon.</i>	<i>Modo di indirizzam.</i>	<i>In assembler</i>	<i>Cod. op</i>	<i>N. di byte</i>	<i>Flag interessati</i>
EOR	Immediato	EOR oper	49	2	NZ
	Pagina zero	EOR oper	45	2	
	Pag. zero,X	EOR oper,X	55	2	
	Assoluto	EOR oper	4D	3	
	Assoluto,X	EOR oper,X	5D	3	
	Assoluto,Y	EOR oper,Y	59	3	
	(Indiretto,X)	EOR (oper,X)	41	2	
	(Indiretto),Y	EOR (oper),Y	51	2	
INC	Pagina zero	INC oper	E6	2	NZ
	Pag. zero,X	INC oper,X	F6	2	
	Assoluto	INC oper	EE	3	
	Assoluto,X	INC oper,X	E9	3	
INX	Implicito	INX oper	E8	1	NZ
INY	Implicito	INY oper	C8	1	NZ
JMP	Assoluto	JMP oper	4C	3	nessuno
	Indiretto	JMP (oper)	6C	3	
JSR	Assoluto	JSR oper	20	3	nessuno
LDA	Immediato	LDA oper	A9	2	NZ
	Pagina zero	LDA oper	A5	2	
	Pag. zero,X	LDA oper,X	B5	2	
	Assoluto	LDA oper	AD	3	
	Assoluto,X	LDA oper,X	BD	3	
	Assoluto,Y	LDA oper,Y	B9	3	
	(Indiretto,X)	LDA (oper,X)	A1	2	
	(Indiretto),Y	LDA (oper),Y	B1	2	
LDX	Immediato	LDX oper	A2	2	NZ
	Pagina zero	LDX oper	A4	2	
	Pag. zero,Y	LDX oper,Y	B6	2	
	Assoluto	LDX oper	AE	3	
	Assoluto,Y	LDX oper,Y	BE	3	
LDY	Immediato	LDY A	A0	2	NZ
	Pagina zero	LDY oper	A4	2	

<i>Codice mnemon.</i>	<i>Modo di indirizzam.</i>	<i>In assembler</i>	<i>Cod. op</i>	<i>N. di byte</i>	<i>Flag interessati</i>
	Pag. zero,X	LDY oper,X	B4	2	
	Assoluto	LDY oper	AC	3	
	Assoluto,X	LDY oper,X	BC	3	
LSR	Accumulatore	LSR A	4A	2	NZC
	Pagina zero	LSR oper	46	2	
	Pag. zero,X	LSR oper,X	56	2	
	Assoluto	LSR oper	4E	3	
	Assoluto,X	LSR oper,X	5E	3	
NOP	Implicito	NOP	EA	1	nessuno
ORA	Immediato	ORA oper	09	2	NZ
	Pagina zero	ORA oper	05	2	
	Pag. zero,X	ORA oper,X	15	2	
	Assoluto	ORA oper	0D	3	
	Assoluto,X	ORA oper,X	1D	3	
	Assoluto,Y	ORA oper,Y	19	3	
	(Indiretto,X)	ORA (oper,X)	01	2	
	(Indiretto),Y	ORA (oper),Y	11	2	
PHA	Implicito	PHA oper	48	1	nessuno
PHP	Implicito	PHP	08	1	nessuno
PLA	Implicito	PLA	68	1	NZ
PLP	Implicito	PLP	28	1	tutti
ROL	Accumulatore	ROL A	2A	1	NZC
	Pagina zero	ROL oper	26	2	
	Pag. zero,X	ROL oper,X	36	2	
	Assoluto	ROL oper	2E	3	
	Assoluto,X	ROL oper,X	3E	3	
RTI	Implicito	RTI	40	1	tutti
RTS	Implicito	RTS	60	1	nessuno
SBC	Immediato	SBC oper	E9	2	NZCV

<i>Codice mnemon.</i>	<i>Modo di indirizzam.</i>	<i>In assembler</i>	<i>Cod. op</i>	<i>N. di byte</i>	<i>Flag interessati</i>
	Pagina zero	SBC oper	E5	2	
	Pagina zero,X	SBC oper,X	F5	2	
	Assoluto	SBC oper	ED	3	
	Assoluto,X	SBC oper,X	FD	3	
	Assoluto,Y	SBC (oper,Y)	F9	3	
	(Indiretto,X)	SBC (oper,X)	E1	2	
	(Indiretto),Y	SBC (oper),Y	F1	2	
SEC	Implicito	SEC	38	1	C
SED	Implicito	SED	F8	1	D
SEI	Implicito	SEI	78	1	I
STA	Pagina zero	STA oper	85	2	nessuno
	Pag. zero,X	STA oper,X	95	2	
	Assoluto	STA oper	8D	3	
	Assoluto,X	STA oper,X	9D	3	
	Assoluto,Y	STA oper,Y	99	3	
	(Indiretto,X)	STA (oper,X)	81	2	
	(Indiretto),Y	STA (oper),Y	91	2	
STX	Pagina zero	STX oper	86	2	nessuno
	Pag. zero,X	STX oper,Y	96	2	
	Assoluto	STX oper	8E	3	
STY	Pagina zero	STY oper	84	2	nessuno
	Pagina zero,X	STY oper,X	94	2	
	Assoluto	STY oper	8C	3	
TAX	Implicito	TAX	AA	1	NZ
TAY	Implicito	TAY	A8	1	NZ
TYA	Implicito	TYA	98	1	NZ
TSX	Implicito	TSX	BA	1	NZ
TXA	Implicito	TXA	8A	1	NZ
TXS	Implicito	TXS	9A	1	nessuno



Finito di stampare il 15 giugno 1982 presso Lito Velox — Trento
Fotocomposto e impaginato da Lito Velox - Trento
Printed in Italy

Ho trovato questa cartolina nel libro
acquistato in

In questo spazio potete scrivere quello che pensate di questo libro.

☐ Desidero ricevere il vostro catalogo.

Desidero ricevere:

- | | |
|---|-----------|
| <input type="checkbox"/> 32 programmi con il PET | L. 9.500 |
| <input type="checkbox"/> Due cassette con i 32 programmi per il PET | L. 30.000 |
| <input type="checkbox"/> Un disco con i 32 programmi per il PET (serie 3000 e 4000) | L. 30.000 |
| <input type="checkbox"/> Intervista sul personal computer, hardware | L. 9.500 |
| <input type="checkbox"/> 32 programmi con l'Apple | L. 9.500 |
| <input type="checkbox"/> Un disco con i 32 programmi per l'Apple | L. 30.000 |
| <input type="checkbox"/> Microsoft Basic | L. 6.500 |
| <input type="checkbox"/> Pascal | L. 8.500 |

- | | |
|--|-----------|
| <input type="checkbox"/> 32 programmi con il TRS-80 | L. 9.500 |
| <input type="checkbox"/> Due cassette con i 32 programmi per il TRS-80 | L. 30.000 |
| <input type="checkbox"/> Due dischi con i 32 programmi con il TRS-80 | L. 30.000 |
| <input type="checkbox"/> Intervista sul personal computer, software | L. 9.500 |
| <input type="checkbox"/> Imparate il Basic con il PET | L. 9.500 |
| <input type="checkbox"/> Il personal computer come professione | L. 7.500 |
| <input type="checkbox"/> Te ne intendi di computer? | L. 8.500 |
| <input type="checkbox"/> Il Basic e il personal computer 1 | L. 9.500 |

Pagherò al postino il prezzo indicato + 1000 lire di spese di spedizione.

Ho trovato questa cartolina nel libro
acquistato in

In questo spazio potete scrivere quello che pensate di questo libro.

☐ Desidero ricevere il vostro catalogo.

Desidero ricevere:

- | | |
|---|-----------|
| <input type="checkbox"/> 32 programmi con il PET | L. 9.500 |
| <input type="checkbox"/> Due cassette con i 32 programmi per il PET | L. 30.000 |
| <input type="checkbox"/> Un disco con i 32 programmi per il PET (serie 3000 e 4000) | L. 30.000 |
| <input type="checkbox"/> Intervista sul personal computer, hardware | L. 9.500 |
| <input type="checkbox"/> 32 programmi con l'Apple | L. 9.500 |
| <input type="checkbox"/> Un disco con i 32 programmi per l'Apple | L. 30.000 |
| <input type="checkbox"/> Microsoft Basic | L. 6.500 |
| <input type="checkbox"/> Pascal | L. 8.500 |

- | | |
|--|-----------|
| <input type="checkbox"/> 32 programmi con il TRS-80 | L. 9.500 |
| <input type="checkbox"/> Due cassette con i 32 programmi per il TRS-80 | L. 30.000 |
| <input type="checkbox"/> Due dischi con i 32 programmi con il TRS-80 | L. 30.000 |
| <input type="checkbox"/> Intervista sul personal computer, software | L. 9.500 |
| <input type="checkbox"/> Imparate il Basic con il PET | L. 9.500 |
| <input type="checkbox"/> Il personal computer come professione | L. 7.500 |
| <input type="checkbox"/> Te ne intendi di computer? | L. 8.500 |
| <input type="checkbox"/> Il Basic e il personal computer 1 | L. 9.500 |

Pagherò al postino il prezzo indicato + 1000 lire di spese di spedizione.

cedola di commissione libraria

L. 120

franco muzzio & c. editore

via bonporti, 36

35100 padova

il formato della cartolina è conforme alle vigenti norme postali

COGNOME E NOME

LOCALITÀ

CAP

cedola di commissione libraria

L. 120

franco muzzio & c. editore

via bonporti, 36

35100 padova

il formato della cartolina è conforme alle vigenti norme postali

COGNOME E NOME

LOCALITÀ

CAP

*Se conoscete già il Basic e possedete un computer Apple,
siete sulla strada giusta per diventare un esperto in materia.*

*Questo libro, partendo dalle istruzioni Basic POKE, PEEK, e CALL,
vi introduce al linguaggio macchina. Quindi potrete sviluppare
un sistema operativo in Basic che vi permetterà di scrivere ed
eseguire programmi in linguaggio macchina. Infine imparerete ad
usare il mini-assembler.*

Tutto ciò rapidamente ed in modo sorprendentemente facile.

*I colori, la grafica ed il suono dell'Apple usati direttamente dal
linguaggio macchina vi permetteranno di ottenere il massimo dal
vostro computer.*



franco muzzio & c. editore

