Guida al COMMODORE 64

J. Heilborn
R. Talbott

Guida al COMMODORE 64

Guida al COMMODORE 64

J. Heilborn
R. Talbott

McGRAW-HILL Book Company GmbH

Amburgo · New York · St Louis · San Francisco · Auckland · Bogotá · Città del Guatemala · Città del Messico · Johannesburg · Lisbona · Londra · Madrid · Montreal · Nuova Delhi · Panama · Parigi · San Juan · San Paolo · Singapore · Sydney · Tokyo · Toronto

Titolo originale: *Your Commodore 64*

Realizzazione editoriale: EDIGEO snc, via Ozanam 10a, 20129 Milano
Traduzione: Paul Sala
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelox, Trento

ISBN 88-7700-001-5

1ª edizione Maggio 1984

Gli autori dedicano questo libro
a Heinz Max e Ingeborg Heilborn e
a Bill e Jo Talbott.

Ringraziamenti

Nessun libro è opera dei soli autori e questo non fa eccezione. Vorremmo esprimere la nostra gratitudine alle seguenti persone senza il cui aiuto quest'opera non sarebbe mai stata realizzata:
Michael Tomczyk, il cui entusiasmo durante il breve incontro a Santa Clara ci ha stimolato a iniziare il lavoro;
Bill Hindorff, la cui esperienza tecnica ci ha aiutato a chiarire molti degli aspetti meno conosciuti del C-64;
Larry Ercolino, che ci ha svelato segreti e misteri della trasmissione dati e ha costituito una fonte apparentemente inesauribile d'informazioni.
Molti altri ci hanno dato il loro aiuto quando ne avevamo più bisogno, e spesso alle ore meno convenienti. Alcuni di essi sono: Pat McAllister, Jeff Hand, Andy Finkel, Neil Harris, Steve Murri, Steven Moser e John Stockman.
Da ultimo vorremmo ringraziare Denise Penrose i cui suggerimenti editoriali ci sono stati d'inestimabile aiuto per completare quest'opera.

J.H. e R.T.

Indice

Introduzione

Il Commodore 64 fa parte della grande famiglia degli home computer e, come molti suoi simili, sta avendo una grande diffusione per il suo basso prezzo e le sue ottime prestazioni.
Solo pochi anni fa, un computer con analoghe caratteristiche sarebbe stato costoso, ingombrante e difficile da utilizzare; il terzo fattore che ha favorito la grande diffusione del C-64 infatti è proprio quello della facilità d'uso: è sufficiente conoscere i primi rudimenti del BASIC per poter immediatamente programmarlo e ottenere interessanti risultati.
Per poter sfruttare appieno le inesauribili risorse del C-64 però è necessario approfondire le proprie conoscenze andando a scoprirne gli innumerevoli segreti; con questo intento è stata concepita questa *Guida al Commodore 64*: essa infatti vi prende per mano fin dal primo contatto con il C-64, quando lo estraete dal suo imballo e, passo dopo passo, ve ne spiega puntualmente tutte le funzioni.
Il capitolo 1 descrive l'aspetto fisico del C-64 e delle sue più comuni periferiche. Il capitolo 2 spiega come familiarizzare con i fondamentali modi operativi e come scrivere o leggere programmi utilizzando le cassette o i dischetti.
Il capitolo 3 è un completo corso di programmazione in BASIC che viene completato dalle nozioni più avanzate descritte nel capitolo 4.
Il capitolo 5 spiega il funzionamento dei joystick, indispensabili per creare e usare programmi di giochi che in generale richiedono un ampio uso delle capacità grafiche del C-64, descritte in dettaglio nel capitolo 6, probabilmente il più importante dell'intera guida.
Nel capitolo 7 si parla del suono e di come creare melodie con l'aiuto del computer.

Le periferiche sono descritte in dettaglio nel capitolo 8 insieme a tutti i comandi necessari. Vengono trattati il Datassette, l'unità a dischetti 1541, la stampante MPS 801 e, in modo più semplice, il collegamento a distanza con l'uso del modem.
L'ultima parte del libro contiene numerose appendici sull'architettura del sistema, la mappa della memoria, le porte di I/O, tavole di conversione e tabelle dei caratteri ASCII e grafici, messaggi d'errore e un riepilogo di tutti i comandi e le istruzioni BASIC.

Nota all'edizione italiana

Talvolta si può creare confusione tra istruzioni BASIC e tasti; per ovviare a questo inconveniente è stata adottata la seguente convenzione:
maiuscoletto per i tasti - RETURN
maiuscolo per le istruzioni - GO TO

Capitolo 1

Descrizione del C-64

Appena tolto dall'imballo il vostro C-64, troverete il seguente corredo come mostra la figura 1.1:

— Il computer C-64
— L'alimentatore
— Il manuale d'uso
— Un cavo di collegamento video

Ponete il C-64 su di un piano sul quale ci sia spazio a sufficienza per un televisore.
La descrizione che segue identifica tutti gli elementi e ne spiega la funzione.

1.1 Interruttore, prese e porte di collegamento

L'interruttore, i collegamenti e le interfacce sono posti sul retro e sul fianco del C-64; essi sono indicati nella figura 1.2. È importante conoscere il funzionamento e l'ubicazione di ogni elemento mentre si effettuano i collegamenti, per evitare di danneggiare il computer.

Figura 1.1. La dotazione base del Commodore 64.

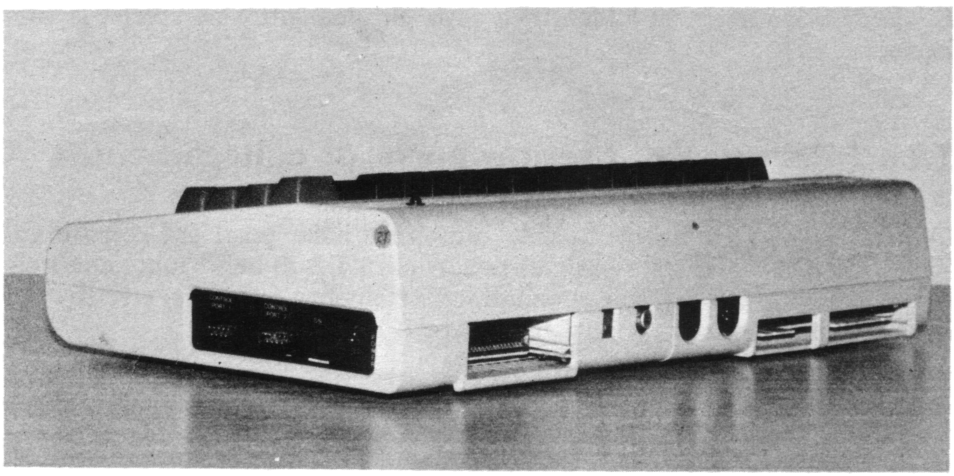


Figura 1.2. Alloggiamento delle prese e dell'interruttore sul C-64.

INTERRUTTORE GENERALE

Assicuratevi che il C-64 sia spento: l'interruttore è situato sul lato destro del computer.
Una volta acceso, lo schermo rimarrà scuro per qualche istante; durante questo tempo il computer si sta "inizializzando", sta controllando cioè i propri sistemi interni e la memoria.
Quando lo spegnete, vengono persi tutti i programmi ed i dati che non sono stati trasferiti sul dischetto o sul nastro magnetico.

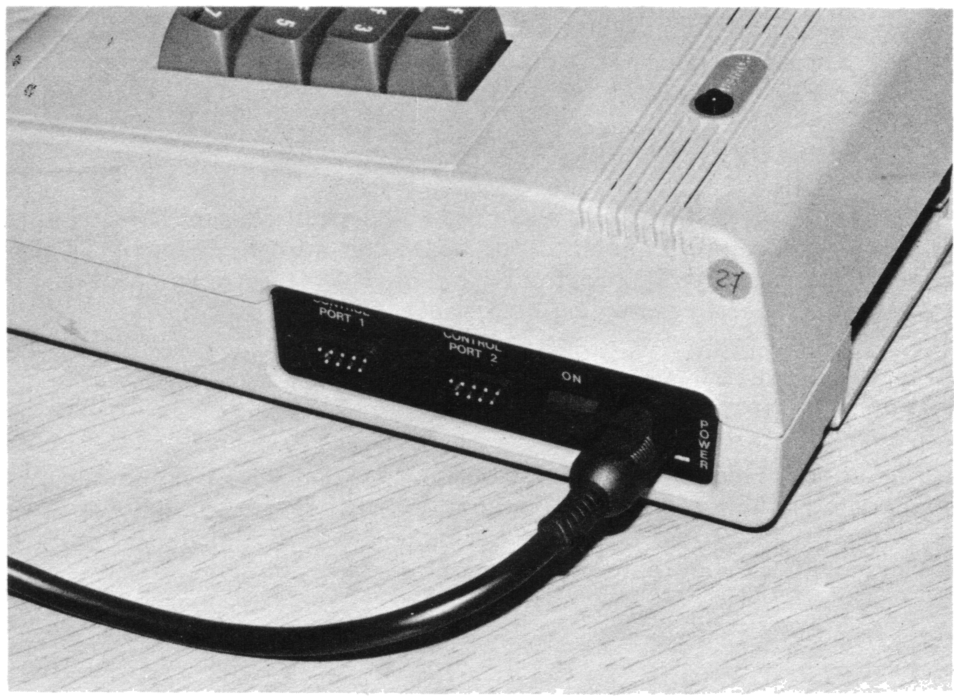


Figura 1.3. Collegamento del cavo di alimentazione al C-64 sulla destra dell'interruttore di accensione.

L'ALIMENTATORE

Il dispositivo d'alimentazione ha due cavi: uno si inserisce in una qualsiasi presa a 220 volt, l'altro si inserisce direttamente nell'apposita presa sulla destra del C-64, di fianco all'interruttore ON/OFF.

PRESE PER I GIOCHI

Queste prese sono usate per collegare i vari comandi dei giochi disponibili per il C-64, per la penna ottica e per altri dispositivi usati in applicazioni speciali. Con questa presa, funzionano i joystick e le manopole della Commodore così come quelli della Atari.

PORTA PARALLELA

La porta parallela permette di collegare dispositivi come il VIC modem (collegamento telefonico) al C-64. Gli utenti più esperti possono usare questa porta per collegarvi dispositivi per qualunque applicazione.

INTERFACCIA PER CASSETTE

L'interfaccia per le cassette è usata per collegare il *Datassette*, che è uno speciale registratore digitale. Si può usare con il C-64 per tenere permanentemente memorizzati programmi e dati. Il Datassette verrà descritto più avanti in questo stesso capitolo.

PRESA PER DISPOSITIVI SERIALI

La presa per dispositivi seriali viene usata per collegare il computer alla stampante modello MPS 801, all'unità a dischetti 1541 e ad altri dispositivi con porte seriali. Le istruzioni per collegare la stampante e l'unità a dischetti al C-64 sono date più avanti in questo capitolo.

PRESE PER IL VIDEO

Il C-64 fornisce il suono e le immagini abbinandoli in un solo segnale chiamato *video composto*. Questo segnale è emesso dalla presa video (connettore antenna TV).
Mediante il cavo in dotazione è possibile collegare questa presa video con la presa d'antenna del vostro TV e poi sintonizzarsi sul canale 36 UHF fino ad avere una immagine nitida.
I segnali audio e video non modulati sono emessi dall'altra porta dotata di presa DIN 41524. È perciò possibile collegare un monitor ed ottenere immagini migliori di quelle della TV, oppure l'amplificatore di una catena HI-FI per un suono di alta qualità.

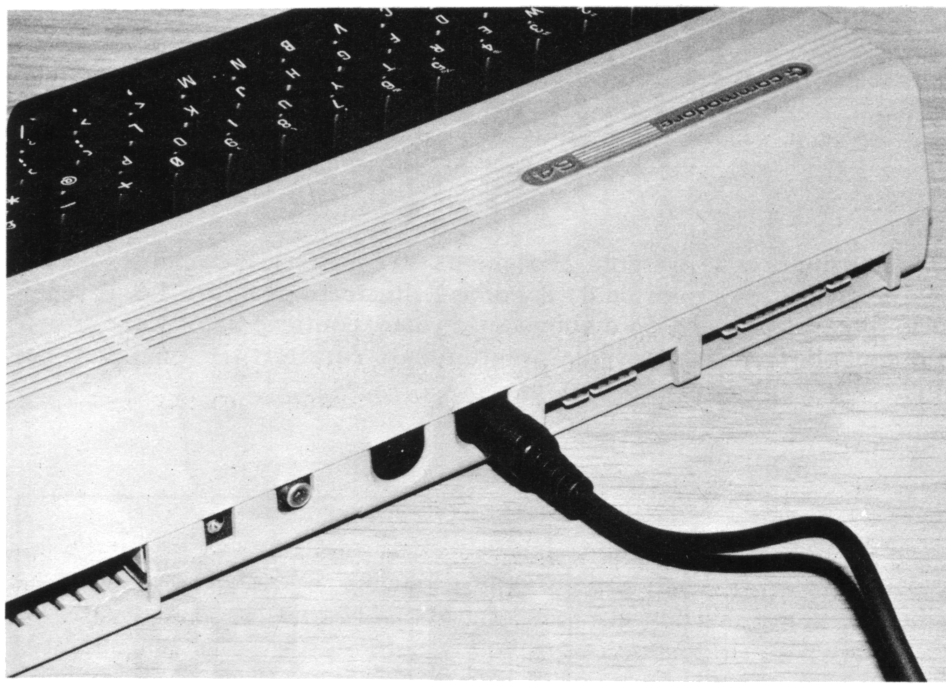


Figura 1.4. Cavo dell'unità a disco (o della stampante) inserito nella presa seriale.

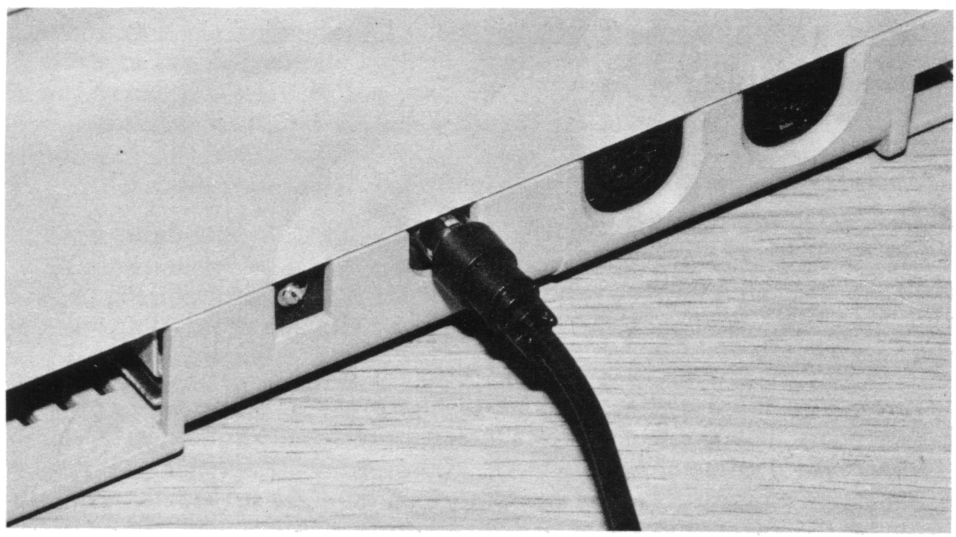


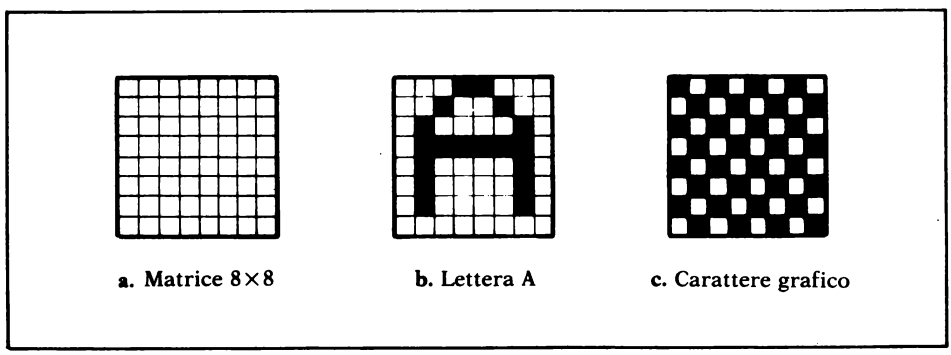
Figura 1.5. Collegamento del C-64 con il video.

CONNETTORE PER CARTUCCIA

Questa porta permette l'inserimento di cartucce contenenti programmi su ROM.

VIDEO

All'accensione, esso presenta 25 righe da 40 caratteri. Il computer genera i caratteri con una matrice 8×8, come è illustrato in figura 1.6. Il repertorio di caratteri del C-64 è abbastanza vasto, contiene 256 lettere, numeri e simboli. È anche possibile programmare caratteri personalizzati per applicazioni speciali (vedi Cap. 6).



a. Matrice 8×8 b. Lettera A c. Carattere grafico

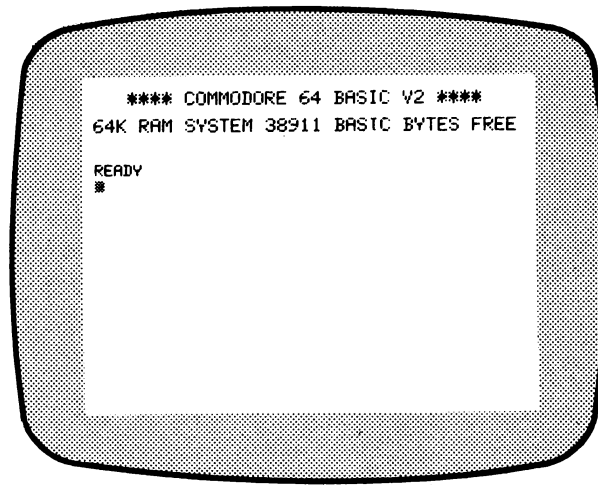
Figura 1.6. Matrice dei caratteri.

1.2. Installazione

Collegate il cavo TV alla presa video sul retro del C-64 (vedi Fig. 1.5). Inserite lo spinotto del cavo d'alimentazione nella sua presa, poi procedete come segue:

1. Inserire la spina d'alimentazione in una presa di corrente.
2. Accendere, posizionando su ON l'interruttore situato sulla destra vicino alla presa d'alimentazione.
3. Aspettare alcuni secondi la presentazione del C-64 sul video; in questo tempo il C-64 esegue i suoi controlli interni e la procedura d'inizializzazione.

Dovrebbe ora apparire la seguente presentazione:



Se questa non appare, spegnete, aspettate circa dieci secondi poi accendete di nuovo. Qualora non si ottenga ancora la presentazione, controllate i collegamenti. Se anche questi sono a posto , contattate il rivenditore Commodore.

1.3. La tastiera

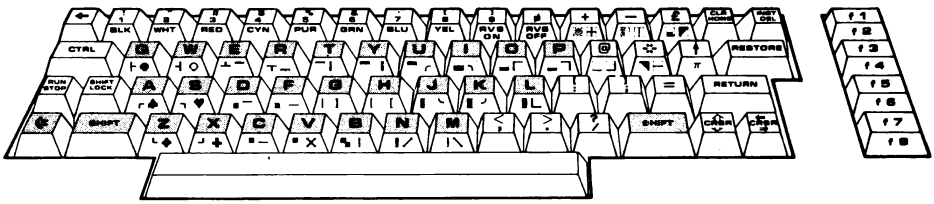
La tastiera è lo strumento più usato per comunicare col C-64. I tasti sono disposti in maniera simile a quelli di una normale macchina da scrivere. A differenza dei tasti di una macchina da scrivere però, quelli del C-64 possono essere usati per accedere a tre o anche quattro simboli, caratteri o funzioni differenti.
I tasti del C-64 si possono classificare per funzione come segue:

— Tasti alfabetici
— Tasti numerici
— Tasti dei simboli speciali
— Tasti grafici
— Tasti di funzione
— Tasti di controllo del cursore

TASTI ALFABETICI

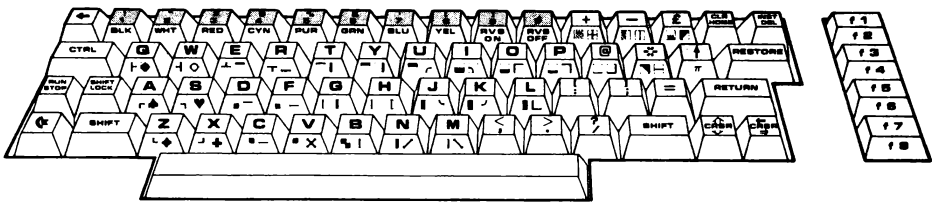
I tasti alfabetici comprendono le 26 lettere dell'alfabeto sia maiuscole che minuscole. Quando il C-64 viene acceso le lettere appaiono in maiu-

scolo. Per avere le minuscole premete simultaneamente i tasti COMMODORE e SHIFT. Qualora si stia scrivendo in minuscolo e si desideri un'occasionale maiuscola, usate il tasto SHIFT come con una normale macchina da scrivere. Premete di nuovo COMMODORE-SHIFT per ritornare alle maiuscole.



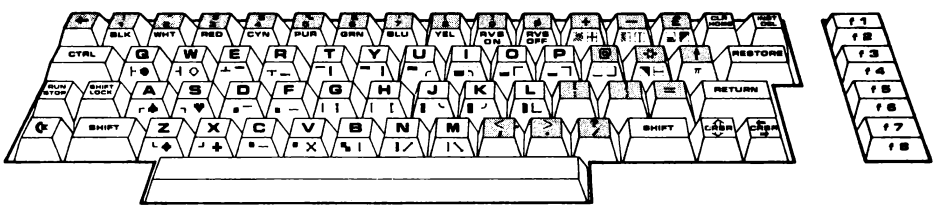
TASTI NUMERICI

I tasti numerici si usano per inserire le cifre dallo 0 al 9.



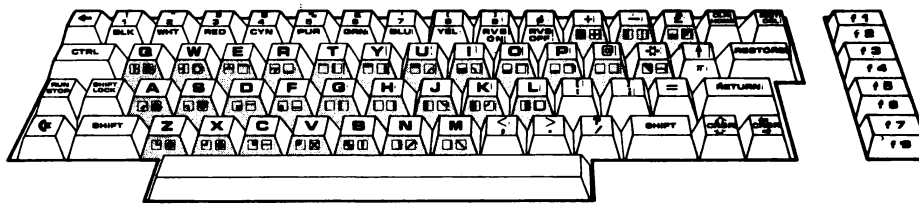
TASTI DEI SIMBOLI SPECIALI

I tasti dei simboli speciali comprendono la seguente punteggiatura standard: ! ' " . , ; : ?. Comprendono anche i seguenti simboli matematici: − + = / * ↑ (notare che la barra è usata per indicare la divisione, l'asterisco per la moltiplicazione e una freccia rivolta in alto per l'elevamento a potenza).
Altri simboli speciali disponibili sul C-64 sono: π # $ & @ % £ π < >[ ] ←.



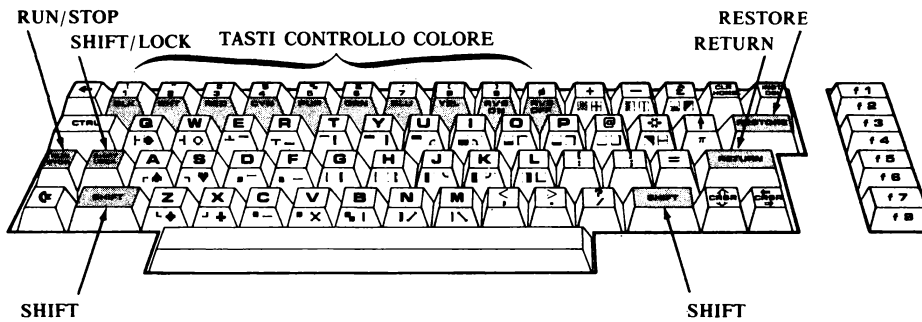
TASTI GRAFICI

Il Commodore C-64 dispone di 62 simboli grafici che sono accessibili attraverso l'uso dei tasti SHIFT o COMMODORE. Usando questi simboli grafici si possono realizzare disegni abbastanza sofisticati.
I simboli grafici ed i loro nomi sono elencati nella tabella 1.1. Simboli simili sono stati raggruppati in modo da rendere immediatamente ovvie le possibilità grafiche. Notate che il quadrato che racchiude ognuno dei simboli raffigurati nella tabella 1.1 non fa effettivamente parte del simbolo stesso ma è stato aggiunto al solo scopo di mostrare la posizione del simbolo all'interno della matrice 8×8.



TASTI DI FUNZIONE

Ogni tasto che "fa qualcosa" invece di "scrivere qualcosa" è un tasto di funzione. Per esempio, CRTL-RED (premere i tasti CTRL e RED simultaneamente) non stampa nulla, ma fa sì che tutti i caratteri seguenti vengano stampati in rosso sullo schermo.



SHIFT

La maggior parte dei tasti ha sia un carattere, o funzione, "maiuscolo" che uno "minuscolo". Il tasto SHIFT, usato insieme a qualunque altro fa

Tabella 1.1. Tasti dei caratteri grafici

Linea orizzontale


Alta


3/4 Alta


2/3 Alta


Quasi centrale


Centrale


2/3 Bassa


3/4 Bassa


Bassa

Linea verticale


Sinistra


3/4 Sinistra


2/3 Sinistra


Quasi centrale


Centrale


2/3 Destra


3/4 Destra


Destra

Linea sottile


Alta


Bassa


Sinistra


Destra

Linea spessa


Alta


Bassa


Sinistra


Destra

Mezzo carattere


Sinistra


Basso

Mezzo carattere diagonale


Alto sinistra


Alto destra

Quarto di carattere (pieno)


Alto sinistra
Alto destra


Basso sinistra
Basso destra


Diagonale

Quarto di carattere (vuoto)


Alto sinistra
Alto destra


Basso sinistra
Basso destra

Spigoli


Alto sinistra
Alto destra


Basso sinistra
Basso destra

Settore angolare


Alto sinistra
Alto destra


Basso sinistra
Basso destra

Semi


Picche, cuori


Quadri, fiori

T


Alto

Basso

Sinistra

Destra

Simboli

X

Croce

Diagonale

Diagonale

Griglia

Piena

Metà sinistra

Metà inferiore

Cerchio

Vuoto

Pieno

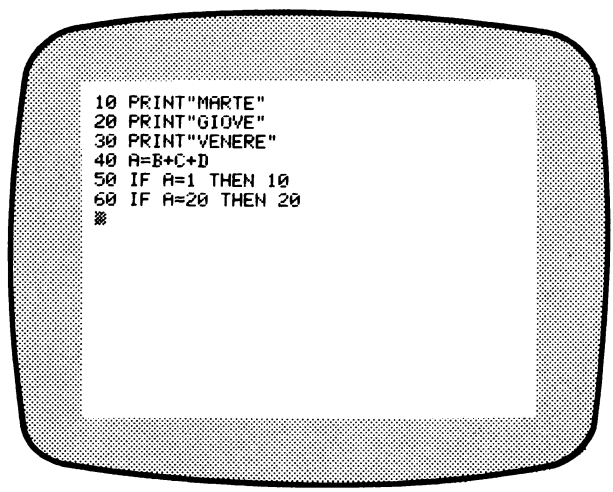
accedere alla funzione o carattere shiftato (maiuscolo) di quel tasto. Per esempio, lettere minuscole shiftate diventano maiuscole e il CRSR UP/DOWN shiftato muove il cursore verso l'alto. Ci sono due tasti SHIFT sulla tastiera del C-64: uno è posto nell'angolo in basso a sinistra della tastiera e l'altro in basso a destra.

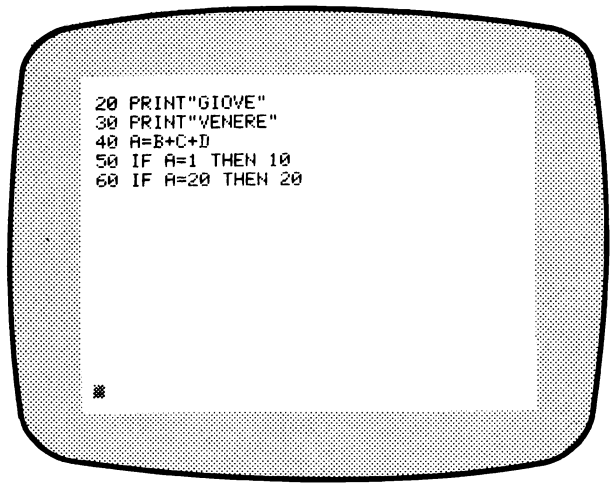
SHIFT LOCK

Occasionalmente si può aver bisogno di una sequenza di caratteri shiftati. Per facilitare questa operazione il C-64 dispone di uno SHIFT LOCK che è simile a quello di una normale macchina da scrivere; premendo lo SHIFT LOCK fino a che non si sente un "clic" si bloccherà la tastiera sul maiuscolo.

RETURN

Il tasto RETURN è molto simile al tasto di "a capo" di una macchina da scrivere. Fa tornare il cursore (il quadrato lampeggiante che indica la posizione in cui il prossimo carattere apparirà) al margine sinistro della riga successiva.
Il tasto RETURN è anche usato per caricare istruzioni in BASIC: dopo aver scritto una riga del programma, si preme RETURN per caricare quella riga in memoria.
Eseguendo un RETURN quando il cursore si trova sull'ultima riga, si fa scivolare l'intera pagina verso l'alto.





REVERSE ON/OFF

I tasti RVS ON e RVS OFF permettono d'invertire le parti chiare dei caratteri sullo schermo con quelle scure. Lo stato normale di questa funzione è RVS OFF; il RVS ON è come il negativo di una fotografia. Per invertire i caratteri premete i tasti CTRL e RVS ON insieme: tutti i caratteri seguenti saranno invertiti. Per ritornare al modo normale premete CTRL e RVS OFF.

```
SCRITTURA NORMALE  SCRITTURA INVERSA
```

RUN/STOP

Nell'uso normale, cioè non shiftato, il tasto RUN/STOP ferma il programma in esecuzione, facendo ritornare il controllo del computer alla tastiera. Esso presenterà anche il numero della riga del programma a cui era giunto il computer prima di ricevere l'istruzione d'arresto.
A dimostrazione di ciò caricate il seguente programmino:

```
10 X = 0
20 PRINT X
30 X = X + 1
40 GOTO 20
99 END
```

Ora battete RUN e premete RETURN. Dovreste vedere apparire una serie di numeri lungo il bordo sinistro dello schermo. Dopo aver premuto RUN/STOP lo schermo dovrebbe apparire come segue:

```
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18

BREAK IN 20
READY.
```

Se invece il C-64 non sta eseguendo un programma, RUN/STOP non fa nulla. Premendo SHIFT e RUN/STOP si carica ed esegue un programma dal Datassette.

RESTORE

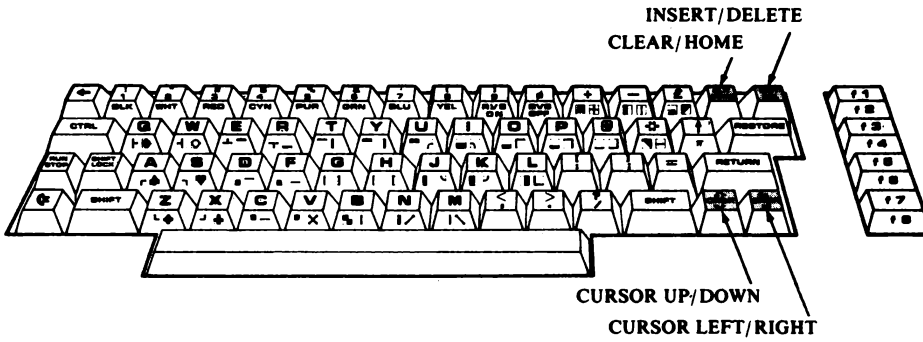
Se premendo il tasto RUN/STOP da solo il programma non si arresta, premete simultaneamente RUN/STOP e RESTORE. Se anche questo non dovesse funzionare spegnete il tutto e poi riaccendete.

TASTI PER IL CONTROLLO DEL COLORE

Lungo la parte alta della tastiera, appena sotto ai tasti numerici (1-8), sono ubicati gli otto tasti di controllo del colore: nero, bianco, rosso, azzurro, viola, verde, blu, giallo. Questi tasti cambiano il colore dei caratteri scritti sul video. Per usarli bisogna premere CTRL ed il colore desiderato.

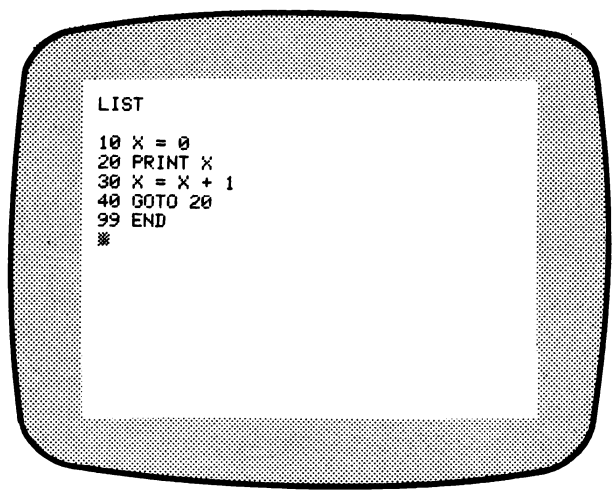
Per esempio, quando si accende il C-64, il video s'illumina con un bordo azzurro, un quadro interno in blu e lettere in azzurro chiaro. Premendo CTRL e PUR insieme si comincerà a scrivere con lettere viola (purple=viola). Premendo il tasto COMMODORE e quelli dei colori si potranno ottenere altri otto colori.

TASTI DI CONTROLLO DEL CURSORE

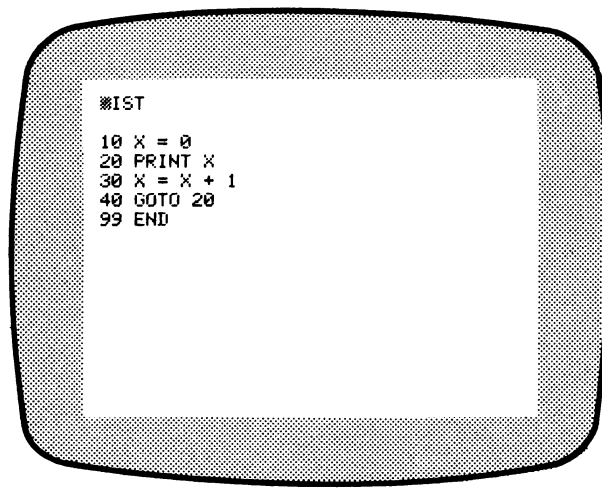


CLEAR/HOME

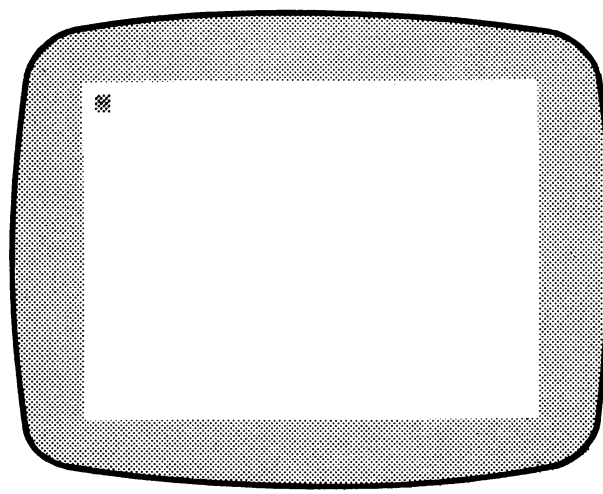
Usato senza premere SHIFT il tasto CLEAR/HOME fa saltare il cursore direttamente all'angolo in alto a sinistra del video (la posizione di riposo o "*home*"). Per dimostrarlo battete LIST e premete RETURN. Dovreste vede-



re il listato del programma caricato in precedenza. Qualora aveste nel frattempo spento il computer dovrete ricaricarlo. Dopo aver battuto LIST vedrete che il cursore si trova sotto al listato dopo la parola READY. Premete il tasto CLR/HOME. Il cursore dovrebbe saltare in alto a sinistra.

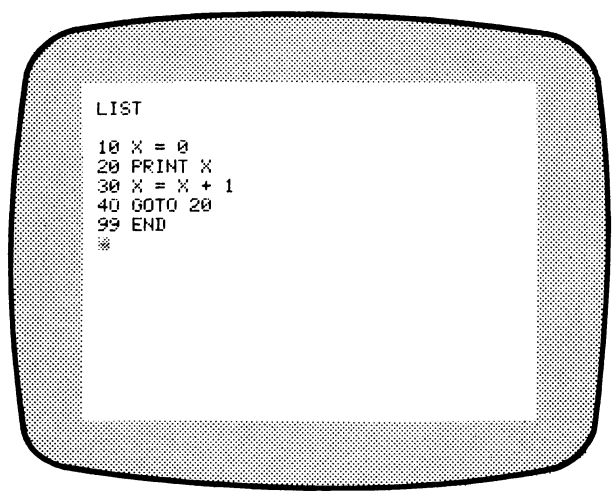


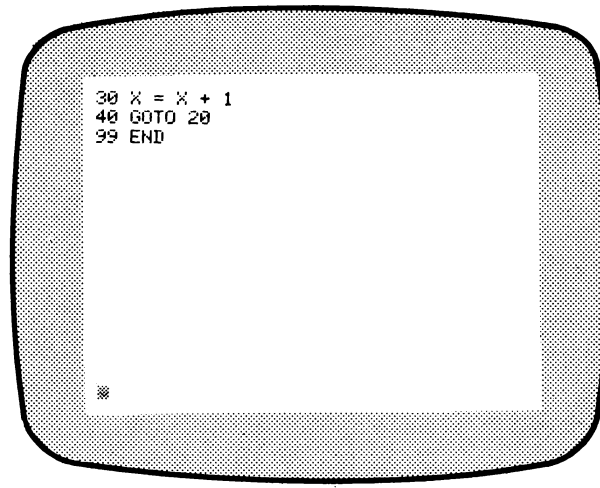
Nell'uso con SHIFT inserito, il tasto CLR/HOME non solo farà tornare il cursore alla posizione di riposo ma cancellerà qualsiasi cosa appaia sullo schermo. Battete di nuovo LIST e premete simultaneamente SHIFT e CLR/HOME.



CURSOR UP/DOWN

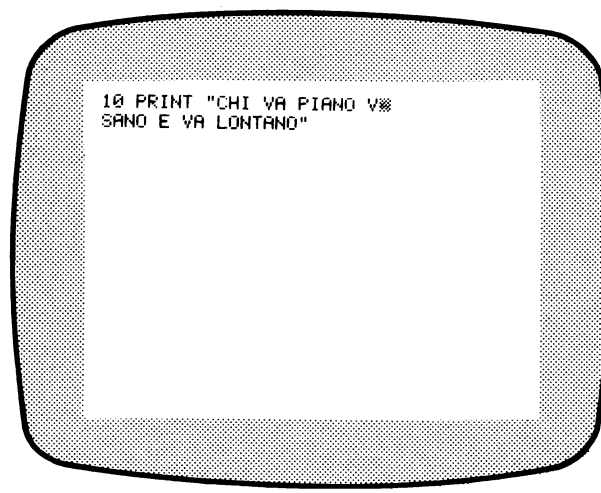
Senza SHIFT inserito, il tasto CRSR UP/DOWN muove il cursore in giù di una riga. Tenendo premuto il tasto CRSR UP/DOWN lo farà scendere fino ad arrivare in fondo allo schermo. Se il cursore si trova sull'ultima riga, tutto lo schermo viene fatto scivolare in su di una riga mentre il cursore resterà in fondo allo schermo.
LISTate di nuovo il programma. Premete il CRSR UP/DOWN diverse volte o mantenetelo premuto. Quando il cursore raggiungerà il fondo dello schermo tutte le righe si sposteranno in su. Nel caso di listati di programma, la pressione di CRSR UP/DOWN provoca lo scivolamento verso l'alto di una riga logica di 80 caratteri. Poiché il video contiene soltanto 40 caratteri può avvenire che nel caso di istruzioni particolarmente lunghe lo scivolamento sia di due righe.
La *lunghezza logica* di una riga è il numero di caratteri che il C-64 è in grado di trattare internamente in un'istruzione numerata.
Con lo SHIFT premuto il tasto CRSR UP/DOWN fa muovere il cursore in su di una riga alla volta e quando arriva in alto si ferma. Il video non scivolerà automaticamente verso il basso, cioè non sposterà tutte le righe in giù di una o più posizioni.

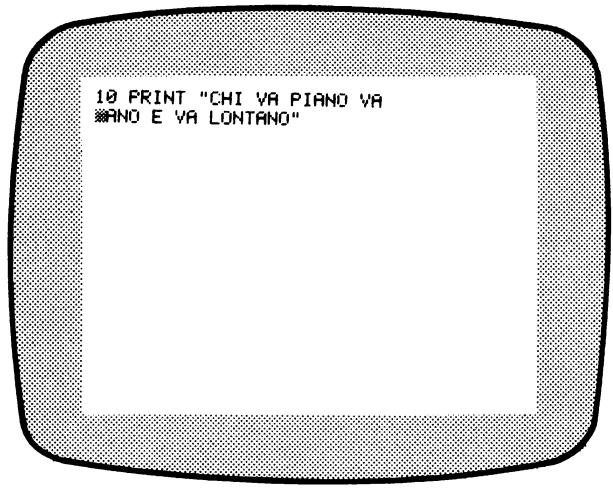




CURSOR LEFT/RIGHT

Il tasto CRSR LEFT/RIGHT normalmente muove il cursore a destra di un carattere. Premete il CRSR LEFT/RIGHT e mantenetelo schiacciato: osservate che, quando il cursore arriva in fondo ad una riga, salta direttamente alla posizione più a sinistra della riga successiva.
Premendo lo SHIFT, il CRSR LEFT/RIGHT farà muovere il cursore a sinistra di un carattere; quando il cursore arriva al margine sinistro del video salta alla posizione più a destra della riga superiore.





Quando il cursore si trova alla estrema destra del video, se si preme il CRSR RIGHT, il video scivola in su di una riga. L'avanzamento del cursore all'inizio della riga successiva produce in effetti un CRSR DOWN. I tasti CRSR UP/DOWN e CRSR LEFT/RIGHT si usano per muovere il cursore sopra il testo senza cambiarlo. Per cambiare il testo, bisogna spostare il cursore fino al carattere che si desidera correggere; scrivendoci sopra un altro carattere lo si sostituirà, correggendo così il testo facilmente sul video stesso.

INSERT/DELETE

Il tasto INST/DEL si usa per inserire o cancellare caratteri sul video. Senza lo SHIFT schiacciato, il tasto INST/DEL cancella il carattere immediatamente a sinistra del cursore. Ciò causa anche lo spostamento di uno spazio verso sinistra di tutti gli altri caratteri di quella riga.

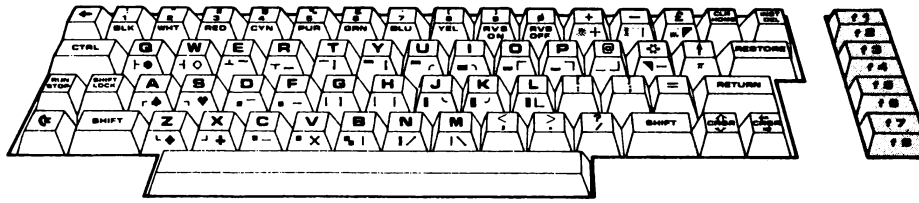
```
10 PRINT "MARMM LLATA"

10 PRINT "MARM LLATA"
```

Premendo SHIFT e INST/DEL insieme, viene inserito uno spazio sul video. Tutti i caratteri a destra del cursore verranno spostati di una posizione a destra.

```
10 A=B+C+D-(4 /F)

10 A=B+C+D-(4  /F)
```

TASTI FUNZIONE PROGRAMMABILI

Oltre ai tasti già esaminati il C-64 dispone di quattro tasti programmabili a doppia funzione. Sono numerati f1/f2, f3/f4, f5/f6 e f7/f8. La funzione di questi tasti è definibile dall'utente.



LA PROGRAMMAZIONE DEI TASTI FUNZIONE

La maggior parte dei tasti funzione può essere inserita nelle istruzioni PRINT nei programmi. Per esempio, potete inserire la funzione di comando colore in una istruzione PRINT tipo:

```
10 PRINT "<CTRL><CYAN>SAN VALENTINO<CTRL> <RED>
   <SHIFT>S"
```

Quando esegue questa riga il computer scriverà le parole SAN VALENTINO in azzurro (cyan), seguite da un cuore rosso.

Tabella 1.2. Tabella caratteri/funzione

CLEAR SCREEN	CYAN	f1
HOME	PURPLE	f2
CURSOR UP	GREEN	f3
CURSOR DOWN	BLUE	f4
CURSOR LEFT	YELLOW	f5
CURSOR RIGHT	REVERSE ON	f6
BLACK	REVERSE OFF	f7
WHITE	DELETE	f8
RED		

Ad eccezione dei tasti SHIFT, SHIFT/LOCK, RETURN, RUN/STOP e RESTORE qualsiasi tasto funzione può essere programmato in istruzioni di PRINT. Quando i tasti di funzione sono programmati in istruzioni di PRINT essi appaiono sullo schermo con caratteri inversi. Questi simboli sono elencati nella tabella 1.2.

1.4. Il Datassette

Vi accorgerete ben presto che caricare tutti i vostri programmi manualmente è noioso. Un registratore di dati risolverà questo problema. Ci sono due unità che funzionano con il C-64, praticamente uguali. Una è l'unità a cassette PET/CBM, l'altra è il VIC Datassette (vedi Fig. 1.7).
Un vantaggio del Datassette sul sistema precedente è la presenza di un contatore che facilita la ricerca dei programmi registrati su nastro. L'uso e l'installazione di queste due unità sono identici. Per usarle seguite le seguenti istruzioni.
Osservate l'interfaccia della cassetta sul retro del C-64 (Fig. 1.2). Come si vede in figura 1.8, il Datassette è fornito di una spina che si inserisce nella presa.

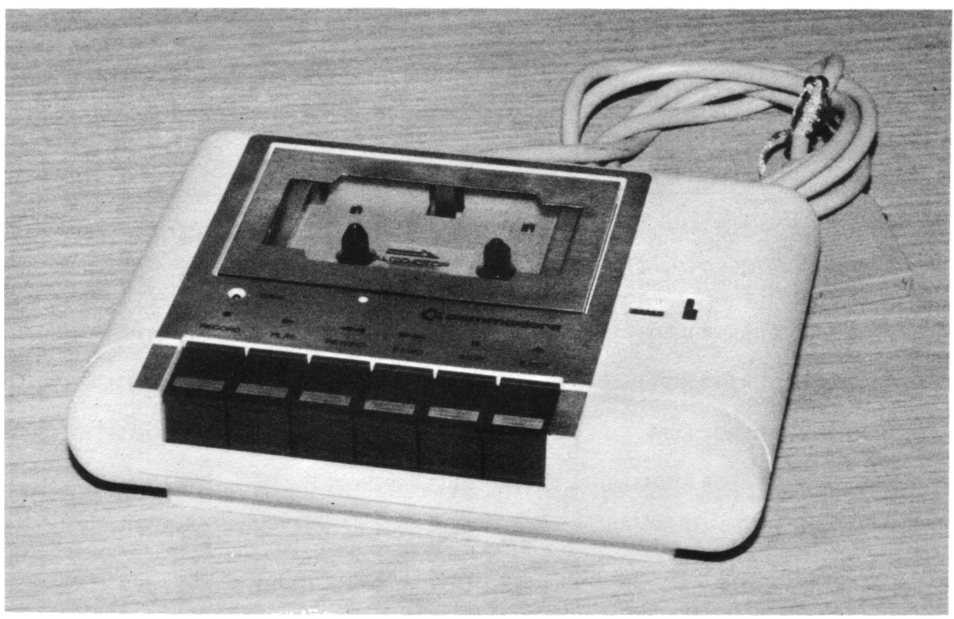


Figura 1.7. Il Datassette Commodore

Talvolta si può riscontrare qualche difficoltà nel rimuovere la spina quando è in posizione. Se la spina s'innesta saldamente nell'apertura, si può essere certi che è stato fatto un buon collegamento.
Per collegare il Datassette al C-64 seguite questa procedura:

1. Spegnere l'alimentazione (OFF).
2. Presentare la spina in modo che la barretta s'inserisca nell'apertura della presa.
3. Delicatamente spingere la spina nel connettore. Non forzare.
4. Assicurarsi che la spina sia perfettamente in posizione.
5. Accendere l'alimentazione.

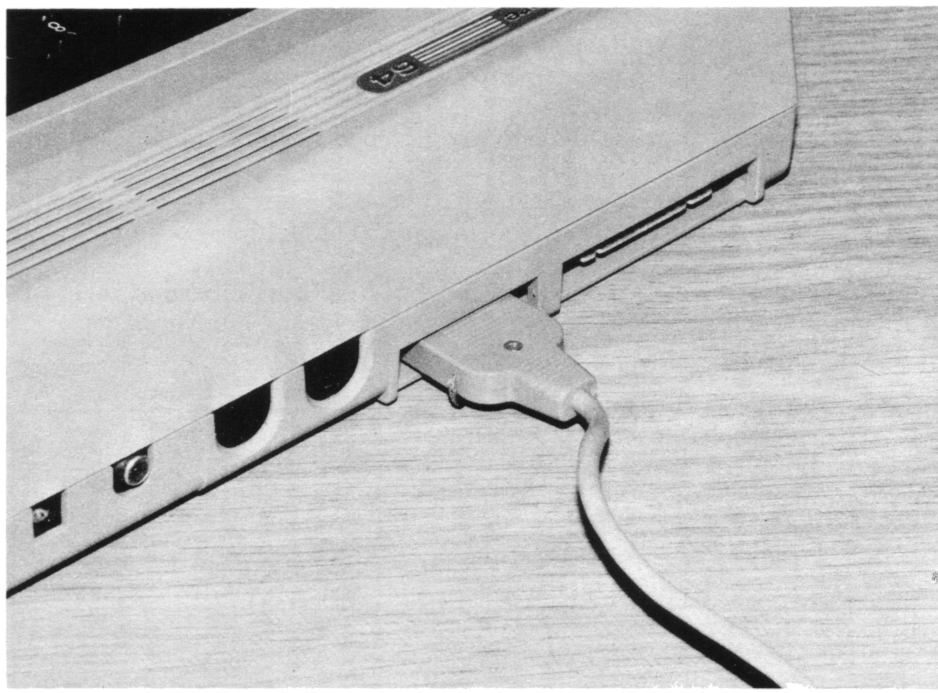


Figura 1.8. Cavo del Datassette inserito nel C-64.

COLLAUDO DEL DATASSETTE

Prima di procedere oltre, è bene controllare il buon funzionamento meccanico del Datassette. Ecco qui un semplice controllo per assicurarsi che tutte le funzioni di comando siano efficienti:

1. Accendere il C-64. Assicurarsi che nessun tasto del registratore sia premuto e che il motorino giranastri sia spento.
2. Aprire lo sportello delle cassette sopra al registratore premendo il tasto STOP/EJECT. Guardando dentro al Datassette premere il tasto PLAY. Si dovrebbe vedere fuoriuscire le testine magnetiche (Fig. 1.9). Contemporaneamente la ruota di trascinamento dovrebbe spostarsi in fuori, toccare il capstan e cominciare a ruotare in senso antiorario.
3. Premere di nuovo STOP/EJECT; le testine dovrebbero rientrare e gli alberini fermarsi.
4. Premere il tasto F.FWD (Fast Forward - Avanzamento veloce). Le testine devono rimanere nascoste e l'albero di avvolgimento, sulla destra, mettersi a girare molto velocemente in senso antiorario.
5. Premere il tasto STOP/EJECT.
6. Premere il tasto REW (Rewind - Riavvolgimento). L'albero di sinistra si deve mettere a girare velocemente in senso orario.
7. Premere il tasto STOP/EJECT.
8. Con delicatezza premere il tasto REC (Record-Registrazione). Dovrebbe essere bloccato.
9. Prendere un nastro nuovo. Osservare che le linguette di protezione poste sul retro della cassetta siano integre. (Fig. 1.10).

Premendo i tasti PLAY e RECORD insieme le testine dovrebbero avanzare fino a contattare il nastro che dovrebbe cominciare a muoversi.

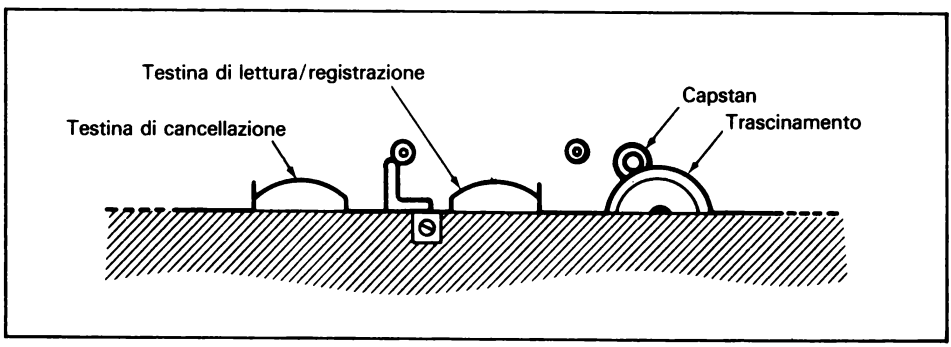


Figura 1.9. Testine del Datassette.

Se l'esito di tutte queste prove è positivo, il registratore è pronto per l'uso. Viceversa, se alcune o tutte le prove sono negative, procedere come segue:

— Assicurarsi che l'apparecchio sia acceso.

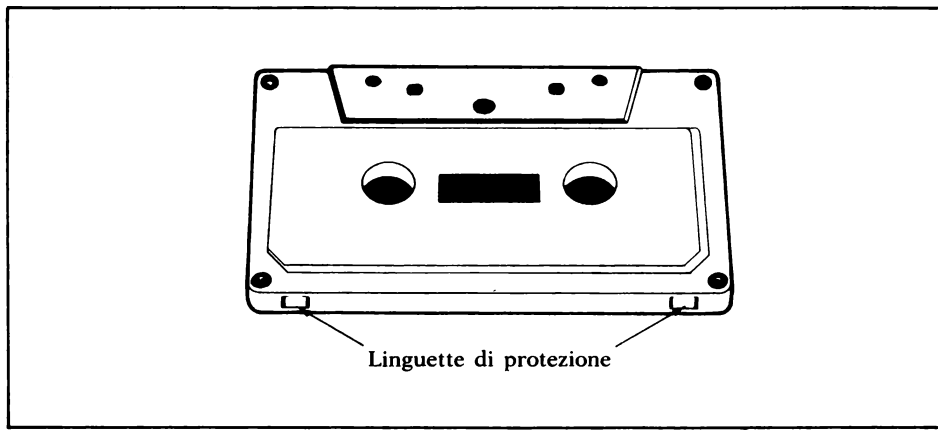


Figura 1.10. Protezione contro la sovrascrittura.

— Premere un solo tasto alla volta (eccetto quando si prova la registrazione).
— Premere a fondo fino a sentire un "clic".

Qualora non si abbia ancora successo, contattare il distributore Commodore.

PULIZIA E SMAGNETIZZAZIONE DELLE TESTINE

Le testine del Datassette possono essere verificate aprendo il coperchio delle cassette con l'apparecchio spento e il tasto PLAY premuto. Fate riferimento alla figura 1.9 per l'ubicazione delle componenti di cui si parla in questa parte.
Le testine sono i dispositivi che vanno a contatto del nastro magnetico e che leggono o scrivono i dati. Dato che il nastro è in contatto fisico con le testine, un po' dell'ossido di rivestimento viene depositato su di esse. Per assicurare un corretto funzionamento del Datassette è necessario effettuare una periodica rimozione di questa patina di ossido usando un batuffolo di cotone imbevuto di alcool denaturato. Pulite entrambe le testine, il capstan e il trascinamento. Fate asciugare completamente prima di richiudere il coperchio.
L'azione di lettura e scrittura di una superficie magnetica come quella di un nastro porta ad un accumulo di magnetismo residuo nelle testine del registratore. Per eliminarlo è buona norma smagnetizzare le testine ogni volta che si puliscono. Non fare questa operazione durante la manuten-

zione ordinaria porta, col tempo, ad una perdita di fedeltà che può dare errori di lettura e scrittura dei vostri programmi.
Per smagnetizzare le testine occorre uno smagnetizzatore: è un apparecchio di basso costo che si può acquistare da un qualunque rivenditore HI-FI.
Il Datassette deve essere spento per questa operazione. Aprite lo sportello della cassetta e premete il tasto PLAY. Assicuratevi che lo smagnetizzatore sia almeno a mezzo metro di distanza dal Datassette prima di inserirne la spina nella presa di corrente. Inserite la spina dell'apparecchio e portatelo *lentamente* verso il Datassette fino a toccare una delle testine; con delicatezza muovetelo intorno alla testina e poi all'altra. Toccate poi tutte le altre superfici metalliche vicino alle testine, quindi allontanate *lentamente* lo smagnetizzatore dal Datassette. Disinserite la spina dell'apparecchio quando lo si è riportato ad una distanza di almeno 50/60 cm dal Datassette.

CURA DELLE CASSETTE

Quando si usa un nastro nuovo è bene uniformarne la tensione facendolo avanzare fino alla fine con il F.FWD e poi riavvolgerlo usando il REW. Quest'operazione aiuterà ad evitare errori di scrittura. Usate nastri corti, da 15 a 30 minuti al massimo: questo ridurrà il tempo necessario per localizzare il programma desiderato e inoltre avrete a disposizione nastri più spessi e robusti e quindi meno soggetti a stiramenti e rotture con l'uso. Evitate i prodotti a basso prezzo perché tendono a causare errori più frequentemente di quelli di alta qualità e basso rumore di fondo (low-noise).
Tenete le cassette in un luogo fresco ed asciutto lontano da fonti di magnetismo.
NOTA BENE: Uno dei posti peggiori per tenere le cassette è sopra o vicino al televisore, che produce un campo magnetico sufficientemente forte da alterare i dati registrati. Non toccate mai direttamente il nastro con le dita perché si danneggia facilmente.

PROTEZIONE DELLA REGISTRAZIONE DELLE CASSETTE

Si può evitare di incidere sopra a programmi che si vogliono conservare proteggendone la registrazione. Osservate la figura 1.10; ogni cassetta ha due linguette di protezione della registrazione, una per facciata. Togliendo questa linguetta si blocca il tasto di registrazione REC del Datassette. Qualora si desideri, dopo aver rotto una linguetta, incidere nuovamente su quella facciata del nastro, occorre semplicemente applicare un pezzetto di nastro adesivo sull'apertura.

1.5. L'unità a dischetti 1541

Il C-64 può usare l'unità di gestione dei dischetti (drive) modello 1541 (Fig. 1.11) che si interfaccia direttamente con il C-64 mediante la presa seriale. La tabella 1.3 elenca le specifiche dell'unità 1541.

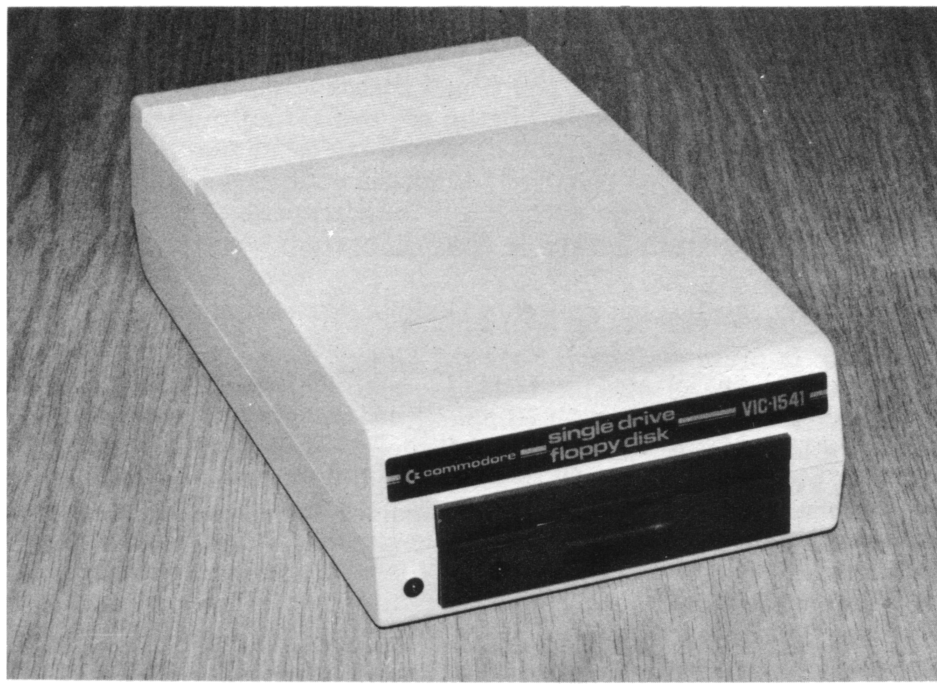


Figura 1.11. Unità a dischetti VIC-1541.

L'unità 1541 contiene dischi con una capacità di memoria di 174.848 byte ciascuno.

Tabella 1.3. Caratteristiche dell'unità a dischetti 1541

Memoria	
Capacità totale dei dischi	174.848 byte per dischetto
No. max programmi	144 per dischetto
Settori per traccia	17-21
Byte per settore	256
Numero tracce	35
Numero settori	683 (664 blocchi disponibili)

(continua)

Dimensioni:	
Altezza	97 mm
Larghezza	200 mm
Profondità	374 mm
Caratteristiche elettriche:	
Tensione	220 V CA
Frequenza	50 Hz
Assorbimento	25 Watt
Supporto:	
Dischetto	Standard 5" 1/4, singola faccia, singola densità

COLLEGAMENTO DELL'UNITÀ A DISCHETTI

Seguite le seguenti fasi per collegare l'unità a dischetti al computer C-64:

1. Staccare la presa d'alimentazione del C-64.
2. Collegare il cavo di interfaccia fornito con il drive alla presa per dispositivi seriali sul retro del computer.
3. Inserire la presa di corrente del drive in un punto d'alimentazione.
4. Reinserire la spina del C-64.
5. Controllare tutti i collegamenti; se appaiono buoni procedere al controllo operativo.

CONTROLLO OPERATIVO

Per fare il controllo operativo, seguite la seguente procedura:

1. Accendere il C-64. Attendere fino a che abbia completato la fase di autocontrollo.
2. Aprire lo sportello dell'unità e verificare che sia vuota, cioè che non contenga un dischetto.
3. Accendere l'unità a dischetti.

INDICATORI LUMINOSI

L'unità di gestione 1541 ha due indicatori luminosi sul pannello anteriore. Quello verde s'illumina quando arriva corrente al drive; quello rosso è l'indicatore d'attività del dischetto. Quest'ultimo s'illumina quando il drive è in moto e lampeggia quando si verifica una condizione d'errore.

IL SUPPORTO MAGNETICO

La figura 1.12 illustra le varie parti del dischetto (floppy disk). La superficie magnetica è composta di un materiale plastico sottile e flessibile, simile a quello usato per i nastri delle cassette. Questo fragile disco è rinchiuso in una busta protettiva. La busta è, a sua volta, inserita in una custodia che protegge l'apertura di lettura e trascrizione dati.
L'unità 1541 mette più 'blocchi' di dati (*settori*) sulle tracce più esterne del dischetto. Alcune unità di gestione trascrivono lo stesso numero di settori su ogni traccia. Tra queste ce ne sono alcune che usano dischetti a *settori rigidi*. Questi dischetti hanno una serie di fori equidistanti nelle vicinanze del centro. Il drive usa questi fori per posizionare i settori. Dato che il 1541 non ha settori disposti uniformemente, non fa uso di questi fori. Il 1541 usa solamente dischetti che hanno un solo foro e li formatta seguendo le proprie specifiche.
Per verificare quale tipo di dischetto si ha a disposizione, seguite la seguente procedura:

1. Togliere il dischetto dalla custodia (non dalla busta) e tenerlo per i lati.
2. Inserire con delicatezza due dita nel foro centrale.
3. Far ruotare il dischetto con le dita nel foro centrale fino a far coincidere il foro nel dischetto con il foro nella busta.
4. Continuare a ruotare il dischetto nella busta. Se si trova un solo foro, il dischetto è del tipo "soft-sectored". Qualora se ne trovassero di più, il dischetto è del tipo a settori rigidi e non può essere utilizzato con l'unità 1541.

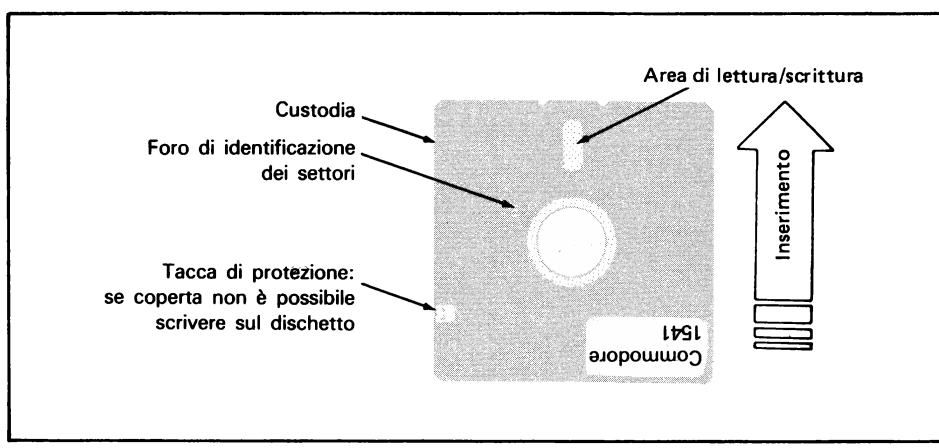


Figura 1.12. Dischetto.

CARICAMENTO DELL'UNITÀ A DISCHETTI

Eseguite le seguenti operazioni per caricare l'unità 1541:

1. Assicurarsi che l'unità sia inattiva (luce rossa spenta).
2. Tenere il dischetto per la busta. Non toccare le parti esposte del dischetto! L'etichetta deve essere rivolta verso l'alto e la tacchetta di protezione (Fig. 1.12) sulla sinistra.
3. Con delicatezza infilare il dischetto nella fessura dell'unità fino a sentire un 'clic'. Se non dovesse entrare con facilità, estraetelo e riprovate. Forzandolo si può danneggiare sia il dischetto stesso sia l'unità di gestione.
4. Usando due dita premere con fermezza sulla chiusura fino al blocco.

SCARICO DELL'UNITÀ A DISCHETTI

Per scaricare il drive procedete come segue:

1. Assicurarsi che il drive sia inattivo (luce rossa spenta).
2. Usando due dita premere e sollevare la chiusura; il dischetto dovrebbe uscire leggermente dalla fessura.
3. Prendere il dischetto con pollice e indice ed estrarlo dal drive. *Non piegare o forzare il dischetto!*
4. Rimettere il dischetto nella custodia.

1.6. I dischetti

Se usati correttamente i dischetti sono un sistema conveniente di memorizzazione di dati. Essi sono, però, assai fragili ed è facile scrivere sopra a dati importanti perdendoli irrimediabilmente.

CURA DEI DISCHETTI

I dischetti vanno trattati con cura. Tutte le informazioni che vi saranno memorizzate sono esposte a gravi rischi e qualora il disco venga danneggiato è quasi impossibile ricuperarle. Ecco qui qualche suggerimento che vi aiuterà a proteggere i vostri dischetti:

1. Quando il dischetto è fuori dall'unità di gestione rimetterlo sempre nella sua custodia.
2. Non togliere *mai* un dischetto dalla sua busta!

3. Per scrivere sull'etichetta posta sulla busta protettiva, usare solo pennarelli. Matite o penne biro potrebbero danneggiare il dischetto.
4. Non toccare o cercare di pulire la superficie del dischetto. Ciò lo danneggerebbe.
5. Non fumare mentre si usa il dischetto. La cenere e il fumo danneggiano la superficie del dischetto.
6. *Tenere i dischetti lontano da campi magnetici!* Anche solo appoggiandoli sul televisore si può provocare il danneggiamento dei dati memorizzati.
7. Non esporre il dischetto al calore o alla luce solare.

PROTEZIONE DALLA SOVRASCRITTURA

Si possono proteggere le informazioni contenute sul dischetto dalla sovrascrittura usando l'apposita protezione. Per effettuare la protezione basta coprire la tacca (vedi Fig. 1.12) con l'etichetta adesiva fornita con il dischetto, oppure con un pezzo di nastro adesivo opaco. Rimuovendo la copertura della tacca di protezione si potrà nuovamente scrivere sul dischetto.

1.7. La stampante grafica MPS 801

È possibile usare praticamente qualsiasi stampante con il C-64 collegandola tramite la presa parallela o a mezzo di un'interfaccia IEEE 488 o RS 232C. La stampante grafica MPS 801 è stata progettata per funzionare direttamente con il C-64 (vedi Fig. 1.13); è in grado di stampare ogni carattere o simbolo standard del C-64, anche inverso ed è possibile programmarla per stampare grafici per punti. La MPS 801 stampa i caratteri usando una matrice di punti 6×7. Nella tabella 1.4 sono raccolte le specifiche della stampante grafica.

COLLEGARE LA STAMPANTE

Per collegare la stampante al C-64 seguite la seguente procedura:

1. Togliere la spina del computer dalla presa.
2. Osservare il retro del C-64. Vi sono due prese circolari, simili tra loro, nel centro, una con cinque spinotti e una con sei. Quest'ultima è la presa per dispositivi seriali. Collegare la stampante a questa presa.
3. L'altro capo del cavo di collegamento va inserito nella presa sul retro della stampante (Fig. 1.14).

Figura 1.13. La stampante grafica MPS 801.

Tabella 1.4. Caratteristiche della stampante grafica MPS 801

Caratteristiche	
Metodo di stampa	a matrice di punti
Matrice del carattere	6X7
Caratteri	maiuscoli, minuscoli, numerici, simboli e caratteri grafici
Grafica	punti indirizzabili singolarmente (480 punti per riga)
Codifica caratteri	CBM ASCII
Formato carattere	altezza: 7 punti (2.82 mm) larghezza: 6 punti (2.53 mm)
Velocità di stampa	50 caratteri al secondo (da sinistra a destra, unidirezionale)
Numero massimo di colonne	80
Spaziatura dei caratteri	10 caratteri per pollice
Interlinea	modo carattere: 6 linee per pollice modo grafico: 9 linee per pollice
Trascinamento carta	trattori
Larghezza carta	da 4.5 a 10 pollici
Copie multiple	originale più una copia
Dimensioni	438×237×115 mm
Peso	4.8 kg circa
Alimentazione	CA 120V (USA), 220-240V (Europa) 50/60 Hz
Assorbimento	25W in stampa 8W in attesa

4. Accendere il C-64 ed aspettare che completi la fase iniziale (apparirà sul video READY).
5. Accendere la stampante. La luce verde sopra alla stampante si deve illuminare e la testina scrivente portarsi al centro della riga e ritornare alla sinistra.

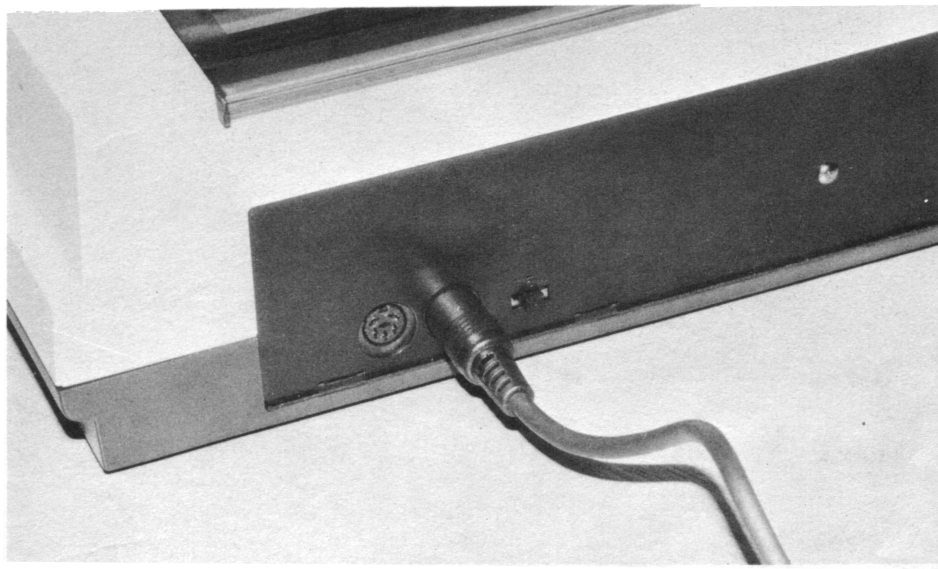


Figura 1.14. Interruttore T-5-4 alla destra della presa per il collegamento con il C-64.

Qualora la stampante non funzionasse, controllate tutti i collegamenti e ripetete la procedura di prova sopra descritta. Se l'esito fosse ancora negativo, consultate il rivenditore Commodore.

INSTALLAZIONE DEL NASTRO

Seguite i seguenti passi per installare il nastro:

1. Togliere la copertura trasparente insonorizzante (Fig. 1.15).
2. Togliere la cartuccia dall'imballo (Fig. 1.16).
3. Inserire la cartuccia curando che il nastro scorra liberamente tra la testina e la carta.
4. Rimettere il coperchio.

Per togliere il nastro fate semplicemente l'operazione inversa.

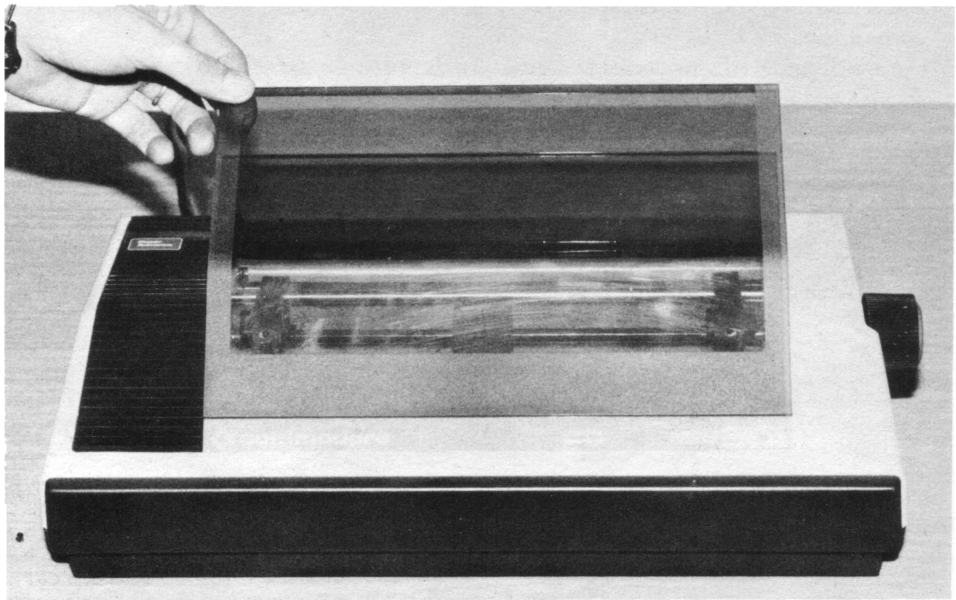


Figura 1.15. Rimozione del coperchio della stampante MPS 801.

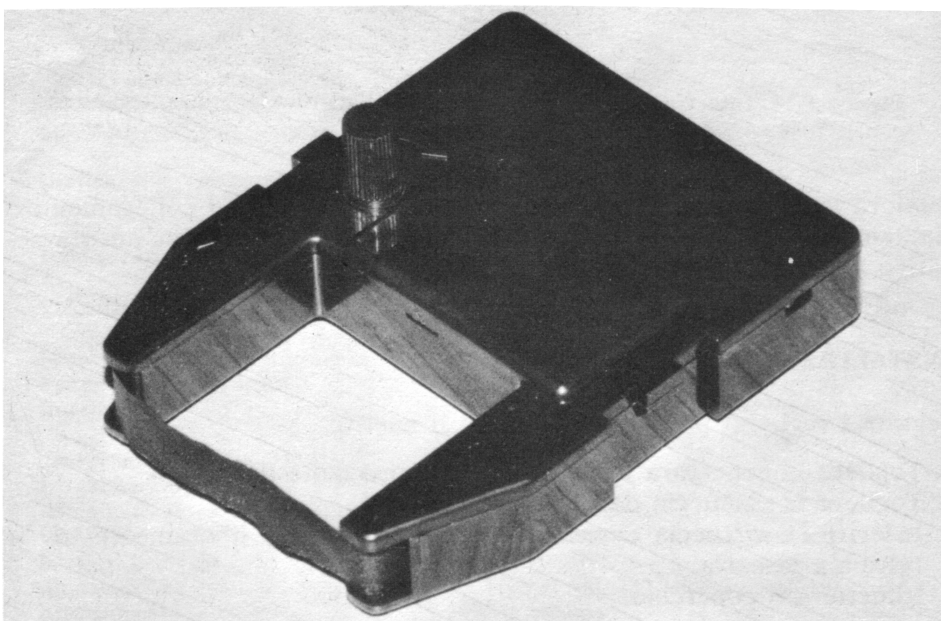


Figura 1.16. Cartuccia del nastro della stampante.

INSERIMENTO DELLA CARTA

La stampante utilizza moduli continui da 11 a 25 cm di larghezza. La carta ha una perforazione da ambo i lati e può avere due strati (un originale e una copia) purché lo spessore totale non superi 0,2 mm.

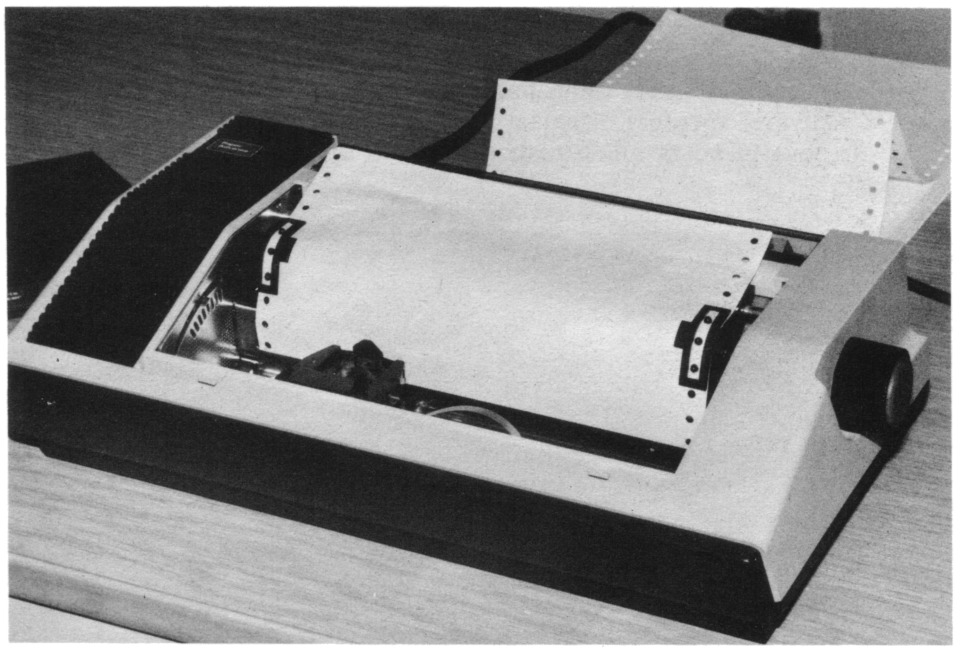


Figura 1.17. Scorrimento della carta.

I seguenti passaggi spiegano come inserire la carta nella stampante:

1. Spegnere la stampante e togliere la presa di corrente.
2. Togliere la copertura di plastica trasparente.
3. Aprire le guide della carta.
4. Inserire la carta dal retro.
5. Continuare a spingere la carta finché questa non emerge dalla parte anteriore della stampante.
6. Tirare la carta dal davanti ed allineare le guide laterali in modo da far combaciare i fori della carta con il meccanismo di trascinamento.
7. NOTA BENE: La carta non deve essere troppo tesa altrimenti si può strappare lungo i fori, ma se non è ben tirata si arriccia e si blocca durante l'avanzamento.

8. Chiudere le guide della carta e rimettere la copertura plastica insonorizzante.
9. Far avanzare la carta manualmente, facendo attenzione che scorra liberamente attraverso il meccanismo di trascinamento.
10. Inserire la presa di corrente ed accendere la stampante (ON).

TESTINA STAMPANTE

Si può regolare l'intensità della battuta della testina stampante per compensare i diversi spessori di carta o l'usura del nastro scrivente, usando l'apposita levetta posta alla sinistra del vano di stampa.

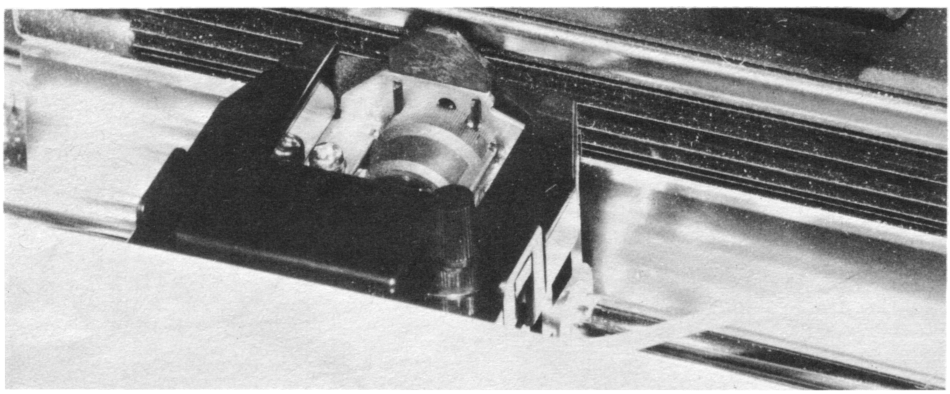


Figura 1.18. Particolare della testina stampante.

Capitolo 2

Uso del C-64

Questo capitolo vi introdurrà all'uso del C-64 e di alcuni dei suoi dispositivi periferici: il Datassette, l'unità a dischetti 1541 e la stampante MPS 801. È molto importante che prendiate confidenza con la tastiera e col video del C-64 e apprendiate i due modi d'uso del computer: il modo diretto e quello programmato.

2.1. Il modo diretto

Quando si accende, il C-64 funziona in modo diretto. Il cursore lampeggiante indica che il computer sta attendendo istruzioni e dove apparirà il primo carattere che verrà scritto sullo schermo.
Nel modo diretto si può usare il computer come se fosse una calcolatrice. Si inseriscono delle dichiarazioni, cioè delle istruzioni per far apparire informazioni, eseguire calcoli o qualsiasi altra funzione richiesta. Quando si preme il tasto RETURN, il C-64 esegue l'istruzione, non prima però di aver controllato la sintassi di ciò che è stato inserito, ovvero la corretta combinazione di caratteri in una dichiarazione. Se la sintassi è corretta la dichiarazione viene eseguita; altrimenti appare il seguente messaggio:

```
?SYNTAX ERROR

READY

▒
```

Quando si riceve un messaggio di errore di sintassi, bisogna controllare l'ortografia della dichiarazione fatta.
Nel modo diretto il computer si comporta esattamente come indica il nome: ogni istruzione viene eseguita immediatamente non appena viene premuto il RETURN.
L'istruzione PRINT è quella usata più frequentemente nel modo diretto. Essa ordina al computer di presentare sul video quel che segue l'istruzione stessa. Per esempio, PRINT scriverà la risposta di calcoli come:

```
PRINT 360+199+1000
```

che, in questo caso, è 1559.
L'istruzione PRINT scrive anche caratteri o sequenze di caratteri. Una stringa è una sequenza di caratteri che include lettere, numeri, spaziature e simboli. Per scrivere una stringa come:

```
QUEL RAMO DEL LAGO DI COMO
```

sul video del C-64, si scrive la seguente istruzione nel modo diretto per poi premere RETURN:

```
PRINT "QUEL RAMO DEL LAGO DI COMO"
```

La stringa è rinchiusa tra virgolette; qualsiasi cosa tra virgolette in un'istruzione PRINT verrà presentata letteralmente come una sequenza di caratteri. Per esempio, un'istruzione PRINT tipo:

```
PRINT "360+199+1000"
```

non calcolerebbe nulla. Il computer farebbe apparire una sequenza di caratteri, che in questo caso sarebbe casualmente composta da tre numeri collegati da +. Viceversa l'istruzione:

```
PRINT QUEL RAMO DEL LAGO DI COMO
```

non avrà alcun effetto se non un messaggio d'errore di sintassi. La sintassi di un'istruzione PRINT pretende sempre un seguito d'informazioni numeriche o in forma di stringa. La parola QUEL non è un numero e non è tra virgolette; quindi il C-64 rifiuta l'istruzione. È possibile abbreviare PRINT con un punto di domanda; le seguenti istruzioni producono lo stesso risultato:

```
PRINT"GIOVANNI MARIANI"
```

equivale a

```
?"GIOVANNI MARIANI"
```

EDITOR DEL VIDEO

Una delle caratteristiche più potenti del C-64 è il suo editor del video. La chiave dell'uso delle capacità dell'editor è il cursore. Potete spostare il cursore in quattro direzioni: verso l'alto, verso il basso, a sinistra e a destra. Potete anche inserire o cancellare caratteri in qualunque punto sullo schermo, o anche cancellare completamente il video con una sola battuta.

Correzione del testo sulla riga corrente

Se commettete un errore mentre state scrivendo, potete correggerlo facendo arretrare il cursore fino all'errore e poi modificarlo. Per esempio:

```
CONTO DELLA STESA
```

avrebbe dovuto essere: CONTO DELLA SPESA. Si può cambiare la T con una P facilmente. Scrivete la riga sopracitata ed usate poi i tasti SHIFT e CURSOR LEFT/RIGHT per posizionare il cursore sulla T.

```
CONTO DELLA S▒ESA

CONTO DELLA SPESA
```

Per sostituire la T con una P battete semplicemente una P. Ciò sostituisce la vecchia lettera con quella nuova e sposta il cursore di una posizione a destra.

Arretramento del cursore con il tasto DELETE

Se avete appena inserito un carattere che si vuol correggere, premete il tasto INST/DEL per rimuovere il carattere sbagliato e poi continuate a battere. Usando il tasto CRSR LEFT/RIGHT invece, il cursore si sposta semplicemente sul video senza cambiare nulla.

Spostamento e cancellazione del testo con DELETE

Nel seguente esempio la parola SPESA è stata erroneamente scritta con due E. È necessario quindi cancellare una delle E e spostare il testo a sinistra di uno spazio per chiudere il vuoto lasciato dalla E. Per far questo usate il tasto CRSR LEFT/RIGHT per posizionare il cursore sopra alla seconda S, poi premete INST/DEL una volta. Ciò cancellerà la lettera alla sinistra del cursore, depennando così la E e spostando il resto del testo per colmare lo spazio.

```
CONTO DELLA SPEE▒A
CONTO DELLA SPE▒A
```

Spostamento del testo per inserire nuovi caratteri

Nell'esempio che segue è necessario cambiare CONTO SPESA in CONTO DELLA SPESA. Per far ciò è necessario usare il CRSR LEFT/RIGHT e posizionare il cursore nello spazio tra CONTO e SPESA.
DELLA ha cinque lettere e servirà anche uno spazio alla fine della parola. Con SHIFT premuto battete sei volte INST/DEL. Questo produrrà sei spazi tra le due parole. Potete ora scrivere DELLA tra CONTO e SPESA.
Ricordatevi che è il carattere direttamente sotto il cursore che viene spostato verso destra. Ricordatevi inoltre d'inserire abbastanza spazi per ogni parola da aggiungere più uno per la separazione tra una parola e l'altra.

```
CONTO▒SPESA
CONTO▒      SPESA
CONTO DELLA▒SPESA
```

Correzione del testo tra virgolette

Qualora si dovesse correggere del testo tra virgolette, sarà necessario prendere alcune precauzioni perché qualunque cosa inserita in una stringa viene incorporata in essa (ad eccezione delle virgolette, di RETURN e RUN/STOP). Questo modo "tra virgolette" permette di inserire caratteri speciali, ma può essere frustrante quando si vuole semplicemente correggere un errore in un'istruzione. Per esempio, la seguente riga dovrebbe essere PRINT "FRUTTA E VERDURA". Inserite la riga esattamente come scritto sotto; non mettete le virgolette di chiusura o premete RETURN. Provate a fare la correzione a FRULTA.

```
PRINT"FRULTA E VERDURA
```

Quando cercate di far arretrare il cursore per correggere FRULTA, il computer scrive un carattere grafico (barra verticale invertita). Per arretrare il cursore quando siete nel modo "tra virgolette", è necessario prima uscirne. Un metodo per riuscirci è di mettere un'altra virgoletta (di chiusura). Mentre è necessario entrare nel modo "tra virgolette" per scrivere una stringa, bisogna uscirne per poterla correggere.
Un altro sistema d'uscita dal modo "tra virgolette" consiste nel tener premuto SHIFT e battere RETURN. Questo sposterà il cursore in giù di una riga permettendo di muoverlo successivamente in su e apportare tutte le correzioni desiderate.
Questa caratteristica che potrebbe sembrare sconveniente, permette di scrivere dei programmi usando i tasti di comando del cursore. Per esempio, l'istruzione in modo diretto

```
QUESTA RIGA SU
PRINT"■QUESTA RIGA SU"
```

farà apparire sul video il testo sopra all'istruzione PRINT al contrario di come avviene normalmente.

CALCOLI ARITMETICI

Il C-64 è in grado di eseguire le quattro operazioni aritmetiche standard: addizione, sottrazione, moltiplicazione e divisione. I simboli per l'addizione e la sottrazione sono quelli convenzionali di + e −, mentre per la moltiplicazione è usato un asterisco (*) e per la divisione una barra (/). Quindi per moltiplicare 4×4 si deve scrivere

```
PRINT 4*4
```

oppure

```
?4*4
```

Per dividere 8 per 2 si scrive

```
PRINT 8/2
```

oppure

```
?8/2
```

2.2. Il modo programmato

Sia in modo diretto che in modo programmato, si possono dare delle istruzioni ed il computer reagisce di conseguenza. Tuttavia le istruzioni effettuate nel modo diretto sono molto labili: qualora si premesse il tasto CLEAR SCREEN, le istruzioni andrebbero perse. Gli esempi eseguiti in precedenza erano costituiti da semplici istruzioni di una sola riga. Una volta presa un po' di confidenza con il vostro computer, vorrete sicuramente scrivere dei programmi più lunghi. I programmi in BASIC possono essere composti da centinaia di istruzioni, anche meno immediate di quelle in modo diretto. Il C-64 conserva le istruzioni in modo programmato nella memoria principale; esse sono più potenti di quelle in modo diretto perché vengono auto-eseguite, una dopo l'altra e non, come nel modo diretto, quando voi premete RETURN dopo ognuna di esse. Nel modo programmato le istruzioni sono eseguite automaticamente nell'ordine prescritto.

CARICAMENTO DEI PROGRAMMI

I programmi possono essere caricati in memoria direttamente dalla tastiera, tramite il Datassette o l'unità a dischetti. Ogni istruzione caricata dalla tastiera ha un suo numero di riga; premendo RETURN alla fine di ogni istruzione si carica in memoria centrale quella riga. Si possono utilizzare per la numerazione delle righe tutti i numeri da 0 a 63999. Le righe di programma saranno eseguite nell'ordine di numerazione e non in quello in cui sono state caricate. Per esempio, se si inseriscono le seguenti righe

```
70 PRINT "GIULIETTA"
10 PRINT "ROMEO"
30 PRINT "AMA"
```

esse vengono eseguite nel seguente ordine:

```
ROMEO
AMA
GIULIETTA
```

Non importa se ci sono intervalli tra i numeri usati; il BASIC del C-64 mette in ordine i numeri delle righe mentre vengono inserite. È buona norma lasciare dei numeri inutilizzati tra una riga e l'altra dei programmi in modo da poter aggiungere altre righe in un secondo tempo. Tutto ciò sarà discusso in maggior dettaglio nel capitolo 3.

ESECUZIONE DI UN PROGRAMMA

Il comando RUN fa eseguire al computer qualunque programma si trovi in memoria in quel momento. Caricate il seguente programma:

```
10 X=0
20 PRINT X
30 X=X+1
40 GOTO 20
```

L'istruzione GOTO a riga 40 ordina al computer di tornare alla riga 20 ed eseguire quell'istruzione. Battendo il comando RUN il programma comincerà l'esecuzione dalla riga con il numero inferiore.

```
RUN
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
```

GOTO seguito da un numero di riga farà cominciare il programma alla riga specificata nell'istruzione.

```
GOTO 30
 22
 23
 24
 25
 26
 27
```

```
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

2.3. Uso del Datassette

Il Datassette vi risparmierà il tempo e la noia di dover inserire i vostri programmi ogni volta con la tastiera. Il Datassette registra e carica nel computer programmi in BASIC quando ce n'è bisogno.
Le operazioni che esso è in grado di compiere vanno ben oltre i limiti di questo capitolo. Il capitolo 8 esplorerà ampiamente tutto il potenziale del Datassette per la memorizzazione e la lettura dei dati.

MEMORIZZAZIONE DI UN PROGRAMMA

Controllate che il piccolo programma caricato nell'ultimo paragrafo sia ancora in memoria scrivendo LIST. Se non lo fosse, ricaricatelo. Per trasferirlo permanentemente su cassetta battete:

```
SAVE"PIPPO"
```

Il C-64 risponderà con:

```
PRESS RECORD & PLAY ON TAPE
```

Premete PLAY e RECORD simultaneamente sul Datassette. Premendo soltanto PLAY il C-64 manderebbe dati al Datassette senza registrare niente. Dopo che avrete azionato i tasti PLAY e RECORD del registratore, il C-64 risponderà con:

```
OK
SAVING PIPPO
```

Mentre il C-64 trasferisce il programma al Datassette il cursore sparisce. Quando ha finito, il Datassette si ferma, il C-64 dà il messaggio di READY e il cursore ritorna a lampeggiare.

VERIFICA

Dopo aver registrato un programma sul Datassette è buona norma verificare che l'operazione sia avvenuta correttamente. Ogni tanto cassette di bassa qualità o piccole fluttuazioni meccaniche possono causare la scorretta registrazione di un programma.
Per proteggersi contro la perdita di programmi, verificateli sempre immediatamente dopo la registrazione. Per fare ciò seguite queste fasi:

1. Riavvolgere il nastro e premere il tasto STOP/EJECT.
2. Battere VERIFY sulla tastiera seguito dal nome del programma e premere RETURN.

Un tipico dialogo di verifica con il C-64 potrebbe configurarsi così:

```
VERIFY "PIPPO"

PRESS PLAY ON TAPE
OK

SEARCHING FOR PIPPO
FOUND PIPPO

VERIFYING
OK
READY
```

Qualora il programma non fosse stato correttamente registrato, si avrà un messaggio d'errore. Se ciò dovesse accadere si deve ripetere SAVE e verificare di nuovo.

CARICAMENTO DI UN PROGRAMMA

Per caricare un programma dal Datassette, occorre semplicemente battere LOAD"*nome del programma*". Il C-64 risponderà con:

```
PRESS PLAY ON TAPE
```

Dopo aver premuto il tasto PLAY del registratore, il C-64 risponderà con OK e poi

```
SEARCHING FOR ["nome del programma"]
```

Quando si carica un programma dal Datassette, non è necessario indicare il nome del programma: se si batte semplicemente LOAD, il computer caricherà il primo programma che trova sul nastro. L'uso del nome fa sì che il computer cerchi il file richiesto, saltando tutti gli altri che trova. Dopo aver caricato un programma da un nastro si può anche lasciar premuto il tasto PLAY. Tuttavia, non dovendo caricare altri dati a breve scadenza, è buona norma fermare il registratore in modo da non rischiare di fare un SAVE per sbaglio mentre il Datassette è in modo PLAY. Il C-64 è in grado di sapere quando viene premuto un tasto del Datassette, ma non quale. Quindi, se PLAY e RECORD sono abbassati (per memorizzare un programma) e si cerca di fare un LOAD, il Datassette cancellerà il programma invece di caricarlo.
Occasionalmente si può avere qualche difficoltà nel caricare un programma da un nastro. Invece di rispondere con

```
READY
```

il C-64 potrebbe mostrare il seguente messaggio:

```
?LOAD
ERROR
READY
```

Se questo dovesse succedere, riavvolgete il nastro e provate a ricaricarlo di nuovo; riprovate diverse volte se necessario. Qualora non riusciate, ci potrebbe essere stato un errore non scoperto al momento della registrazione. Verificate (VERIFY) sempre i programmi dopo averli registrati sul nastro e se sono importanti o hanno richiesto un tempo notevole per essere caricati, è bene farne delle copie di scorta.
Un modo per ridurre questi errori durante le registrazioni su nastro è quello di far avanzare il nastro velocemente (F.FWD) fino alla fine e poi riavvolgerlo (REW) prima di cominciare una registrazione.
Quest'azione d'avvolgimento e riavvolgimento tende a far sì che il nastro sia più uniforme nel suo scorrere attraverso il meccanismo del registratore. Quando i nastri sono prodotti infatti, sono avvolti sulle loro bobine a velocità elevata. Questo procedimento spesso dà luogo a tensioni nel nastro, e la prima volta che viene usato può fare dei piccoli balzi mentre passa attraverso il meccanismo; potrebbe anche tirare e forzare legger-

mente nella cassetta. Questi effetti non sono normalmente percettibili nelle applicazioni audio, ma possono causare errori o perdite di dati nell'uso con il computer.
Prima di leggere un programma cercato ricordatevi di riavvolgere il nastro fino ad un punto situato prima del programma da caricare, altrimenti il computer non lo troverà mai. Una buona regola è quella di annotare il numero del contatore all'inizio del programma e di scriverlo sull'etichetta identificatrice a fianco del nome del programma. Questo vi risparmierà del tempo quando caricate un programma, dato che il computer non avrà bisogno di cercare a lungo il programma desiderato.

LOAD e RUN

Premendo SHIFT e RUN/STOP insieme si caricherà (LOAD) ed eseguirà (RUN) automaticamente il successivo programma sul Datassette, ma solamente quando il computer è in modo diretto (cioè non sta eseguendo alcuna istruzione contenuta in un programma).

2.4. Uso dell'unità a dischetti 1541

Se avete già esperienza dell'unità a cassette, avrete sicuramente apprezzato quanto tempo viene risparmiato rispetto al caricamento manuale dei programmi tramite tastiera. L'unità 1541 può far risparmiare ancora più tempo, data la sua maggior flessibilità e la più elevata velocità d'accesso ai dati. Questa parte tratterà le operazioni fondamentali dell'unità, il catalogo (o indice) del dischetto, il caricamento e l'esecuzione dei programmi. Le operazioni e le istruzioni verranno trattate in maggior dettaglio nel capitolo 8.

CARICAMENTO DI UN PROGRAMMA

Per caricare un programma da un dischetto, battete LOAD "*nome del file*",8. Il numero "8" è il numero d'identificazione del drive. Questo numero è stabilito al momento della fabbricazione. Il C-64 presenterà SEARCHING FOR "*nome del file*". L'unità verrà attivata e si accenderà la luce rossa sul davanti. Se il programma è sul dischetto, sul video apparirà anche

```
LOADING
READY
```

Altrimenti, si vedrà

```
?FILE NOT FOUND
ERROR
READY
```

Se non siete sicuri del nome di un file particolare o di cosa sia memorizzato sul dischetto, allora battete

```
LOAD"$",8
```

e l'indice del dischetto verrà caricato in memoria centrale. Per vedere questo elenco scrivete

```
LIST
```

e l'indice apparirà sullo schermo. Attenzione! Questa operazione, caricando in memoria l'indice, cancella il programma precedentemente caricato.

```
0 [illegible]
13    "HOW TO USE"          PRG
5     "HOW PART TWO"        PRG
4     "VIC-20 WEDGE"        PRG
1     "C-64 WEDGE"          PRG
4     "DOS 5.1"             PRG
11    "COPY/ALL"            PRG
9     "PRINTER TEST"        PRG
4     "DISK ADDR CHANGE"    PRG
4     "DIR"                 PRG
6     "VIEW BAM"            PRG
4     "CHECK DISK"          PRG
14    "DISPLAY T&S"         PRG
9     "PERFORMANCE TEST"    PRG
5     "SEQUENTIAL FILE"     PRG
13    "RANDOM FILE"         PRG
558 BLOCKS FREE.
```

Per richiamare un programma da un dischetto, è assolutamente necessario scrivere anche il nome del file esattamente così come appare nell'indice: per questo è utile listare l'indice del dischetto prima di caricare un programma.
Osservate la videata precedente: la prima riga (in caratteri negativi) mostra il nome del dischetto ed il suo numero di identificazione (ID). Il drive memorizza sempre questo numero e ogni volta che si chiede accesso ad un dischetto fa una comparazione con il numero memorizzato dall'ultimo accesso al disco: se i numeri sono identici l'unità assume che sia lo stesso dischetto e compie le sue operazioni di SAVE o LOAD.

Se il numero non è uguale a quello nella memoria dell'unità, quest'ultima crea una mappa di tutti i file e aggiorna la propria memoria d'identificazione.
Questo procedimento si chiama inizializzazione. L'unità si inizializza automaticamente ogni volta che si cambia dischetto. Se si hanno due dischetti con lo stesso numero ID si dovrà inizializzare manualmente l'unità mediante il seguente comando:

```
OPEN 1,8,15,"I0"
```

Ricordatevi che è necessario inizializzare solo quando si cambia dischetto e solo qualora il numero ID e il nome del dischetto appena tolto fossero identici a quelli del nuovo disco.
Una volta che un programma è stato caricato nella memoria centrale, sia da un nastro che da un dischetto o manualmente, il computer lo tratterà nella medesima maniera, per cui le procedure di esecuzione e correzione del programma restano inalterate.

FORMATTAZIONE DI UN DISCHETTO

Prima di poter memorizzare i dati su un dischetto nuovo, bisogna prepararlo usando una procedura chiamata formattazione. Il computer registra e ritrova i dati accedendo a posizioni speciali sul dischetto chiamate settori. Queste posizioni devono essere preparate sul dischetto perché sia possibile memorizzarvi alcuni dati. Ogni settore è un segmento delle tracce sul dischetto. Le tracce sono simili a quelle di un normale disco musicale, ma sistemate in cerchi concentrici invece che a forma di spirale. La figura 2.1 mostra come si presenterebbero le tracce di un dischetto se si potessero vedere. Ogni traccia è divisa in parti più piccole, i settori, ognuna delle quali può contenere fino a 256 byte di dati. Quando si acquista un dischetto nuovo, esso non è diviso in settori, e non può memorizzare alcun dato. I dischetti non sono formattati quando si comprano perché ogni produttore di unità a dischi utilizza una formattazione leggermente diversa. I dischetti del C-64 devono essere formattati su un'unità Commodore. Ecco due metodi per formattare un dischetto.

Metodo 1:

```
OPEN 1,8,15
PRINT#1, "N:n° unità:nome dischetto, ID"
```

Esempio:

```
OPEN 1,8,15
PRINT#1,"N:DISCO PROVA,01"
```

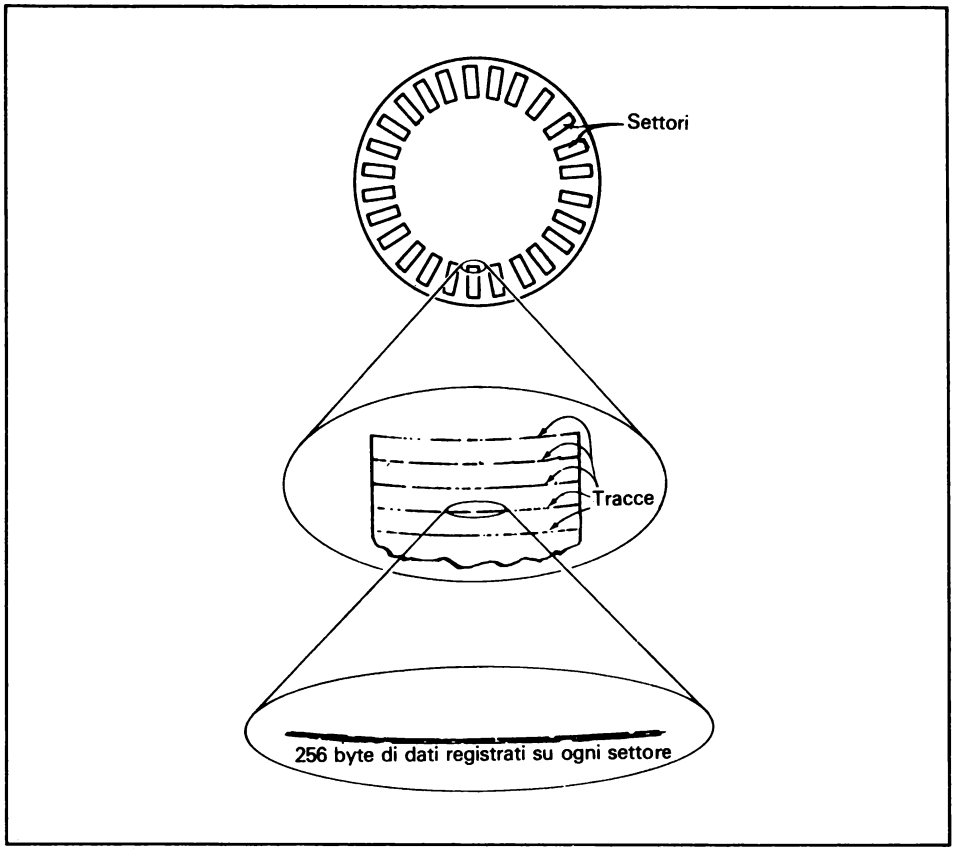


Fig. 2.1. Superficie del dischetto.

Il numero dell'unità può essere omesso quando ce n'è una sola collegata al C-64.

Metodo 2:

OPEN 1,8,15,"N *n° unità:nome dischetto*,Ø1"

Esempio:

```
OPEN 1,8,15,"N:DISCO PROVA,Ø1"
```

Ancora una volta il numero dell'unità può essere omesso in sistemi che ne hanno una sola. Avendo un dischetto da cancellare completamente e poi preparare come nuovo, si può semplicemente battere:

OPEN 1,8,15,"N *n° unità:nome dischetto*".

Esempio:

```
OPEN 1,8,15,"N:DISCO PROVA"
```

Come prima il numero dell'unità può essere omesso e in questo caso il numero ID è appositamente tralasciato. È inserito il nuovo nome e mantenuto il vecchio numero ID. Tutti i programmi ed i dati memorizzati sul dischetto saranno cancellati. Non confondete questa operazione con l'inizializzazione che prepara il dischetto e l'unità di gestione per l'uso abbinato. La formattazione cancella completamente il dischetto.

MEMORIZZAZIONE DI UN PROGRAMMA

Memorizzare un programma su dischetto è quasi uguale che sul Datassette. La differenza maggiore è nella sintassi del comando. Per farlo battete l'istruzione SAVE"*nome del programma*",8.
Per esempio, per memorizzare il programma "PIPPO" di prima, caricatelo e battete

```
SAVE "PIPPO",8
```

L'unità dovrebbe attivarsi, la spia rossa illuminarsi e il video dovrebbe mostrare

```
SAVING PIPPO
```

e il cursore dovrebbe sparire. Quando il programma è stato trascritto, il video mostrerà il messaggio READY e il cursore ritornerà. Dopo aver trascritto un programma è buona norma verificarlo, come per i programmi su nastro. Per verificare che PIPPO sia stato memorizzato sul dischetto, battete

```
VERIFY "PIPPO",8
```

Sul video apparirà

```
SEARCHING FOR PIPPO
VERIFYING
OK
READY
```

se il programma è stato memorizzato correttamente. Altrimenti si avrà un messaggio d'errore VERIFY ERROR. In questo caso provate a memo-

rizzare di nuovo e poi verificare. Ogni tanto può succedere che ci siano dei difetti sulla superficie dei dischetti. Qualora si cercasse di memorizzare un programma senza riuscirci, potrebbe esserci un difetto sulla traccia o sul settore a cui si cerca di accedere: in questo caso provate con un altro dischetto.

2.5. Uso della stampante grafica MPS 801

Si può usare la stampante del C-64 per scrivere i risultati dei programmi e per listare i programmi stessi. La stampante può essere usata sia nel modo diretto che in quello programmato.

L'ISTRUZIONE OPEN

Prima di poter inviare i dati alla stampante, si deve aprire ad essa un canale d'accesso, usando la seguente istruzione: OPEN *numero da 1 a 255* ,4.
Il primo numero (tra 1 e 255) è il numero che userete per indirizzare la stampante. Il secondo è il numero di identificazione dell'unità stampante: può essere 4 o 5.
Ecco un esempio di un'istruzione OPEN alla stampante:

```
OPEN 1,4
```

Si possono ora indirizzare tutti i dati da stampare al canale 1 scrivendo

```
PRINT#1, "ROSSO DI SERA BEL TEMPO SI SPERA"
```

La stampante dovrebbe ora stampare ROSSO DI SERA BEL TEMPO SI SPERA e tornare a capo facendo avanzare di una riga la carta.
In un programma si potrebbe scrivere alternativamente sulla stampante e sullo schermo semplicemente scrivendo PRINT quando si vuole visualizzare sul monitor e PRINT#1 quando si vuole scrivere sulla stampante. PRINT manda sempre l'output al dispositivo primario (il video, per esempio, o un altro dispositivo selezionato da un'istruzione CMD). PRINT# manda l'output ad un canale aperto specificamente a quel numero d'identificazione.

ISTRUZIONE CMD

Avendo un programma che presenta tutti i dati sul video e volendo invece stamparli su carta, basta aggiungere due righe all'inizio del programma stesso:

```
OPEN 1,4
CMD 1
```

L'istruzione OPEN apre un canale alla stampante e l'istruzione CMD manda tutto l'output alla stampante. Così facendo ogni istruzione PRINT viene automaticamente eseguita sulla stampante. L'istruzione CMD permette anche di listare l'indice di un dischetto. Per far ciò occorre scrivere

```
LOAD"$",8
```

Questo caricherà l'indice in memoria centrale. Ora scrivendo

```
OPEN 1,4
CMD 1
LIST
```

si otterrà sulla stampante un indice completo del dischetto. Questo sistema vi permette anche di produrre una copia dell'indice da applicare alla custodia di protezione del dischetto.
Per uscire dal modo CMD, scrivete PRINT# *n° del dispositivo.*

ISTRUZIONE CLOSE

Dopo aver terminato di stampare, bisogna chiudere il canale alla stampante. Inserite CLOSE *n° del canale.* Avendo aperto il canale n° 1 (come nell'esempio sopra) battete

```
CLOSE 1
```

Dopo aver usato un'istruzione CMD, sarà necessario far precedere all'istruzione CLOSE un PRINT#.
Ecco un esempio:

```
OPEN 1,4
CMD 1
.
.
.
PRINT#1:CLOSE 1
```

Queste istruzioni assicurano che tutti i file sono stati correttamente chiusi. Omettere questa operazione può generare errori di file.

Capitolo

Programmazione del C-64

Questo capitolo vi insegnerà a programmare il vostro C-64 usando il BASIC . Se siete già a conoscenza del BASIC, le appendici G e H serviranno da dettagliato manuale d'uso per ogni singola istruzione. Se invece siete principianti, cominciate con questo capitolo: esso vi darà le necessarie nozioni per continuare con il resto del libro.

3.1. Elementi del linguaggio di programmazione

Le istruzioni di un programma devono essere scritte seguendo delle regole ben precise. Queste regole, prese nell'insieme, formano la *sintassi*. Ci sono diversi gruppi di regole che definiscono il modo in cui devono essere scritte le istruzioni di un programma: ogni serie si applica a un ben definito linguaggio di programmazione. Tutte le regole di sintassi descritte in questo libro sono applicabili solamente al BASIC del C-64.
I linguaggi di programmazione sono diversi tra di loro quanto le lingue parlate. Oltre al BASIC, altri linguaggi comuni sono il PASCAL, FORTRAN, COBOL, APL, PL-1 e FORTH; i linguaggi meno comuni sono centinaia.
Sfortunatamente, come quelli parlati, i linguaggi di programmazione hanno forme dialettali. Un programma scritto in BASIC per il C-64 potrebbe non funzionare su di un altro computer anch'esso programmabile in BASIC. Queste variazioni nella sintassi dei linguaggi sono dovute alle limitazioni e alle caratteristiche speciali dei singoli computer. Tuttavia, una

volta imparato a programmare il C-64 in BASIC, si avranno pochi problemi ad apprendere il dialetto BASIC di un altro computer.
Alcune delle regole di sintassi sono evidenti. Gli esempi di addizione e sottrazione del capitolo 2 utilizzano una sintassi ovvia: non è necessario essere programmatori per capire queste semplici istruzioni di calcolo. Un gran numero di queste regole invece sembrano essere arbitrarie, e a volte lo sono; per esempio, perché usare "*" per rappresentare la moltiplicazione? Si usa comunemente una "X" per questo; ma il computer non potrebbe in alcun modo differenziare una "X" rappresentante una moltiplicazione da una rappresentante semplicemente la lettera dell'alfabeto, quindi quasi tutti i computer usano un asterisco (*) per la moltiplicazione. La divisione è universalmente rappresentata da una barra trasversale (/). Dato che il simbolo standard per la divisione (÷) non è presente sulle tastiere dei computer o delle macchine da scrivere, è stata scelta la barra perché fa apparire l'espressione come una frazione.
La sintassi delle istruzioni BASIC tratta separatamente la numerazione delle righe, i dati e le istruzioni al computer. Le descriveremo ora una per una.

NUMERAZIONE DELLE RIGHE

Nel modo programmato ogni riga di un programma in BASIC deve avere un numero d'identificazione. La prima riga di un programma deve avere il numero più basso, mentre l'ultima quello più alto. Tra queste due le altre righe devono avere numeri intermedi in ordine crescente; non importa in che ordine si inseriscono le righe sul video, il C-64 le metterà nel giusto ordine sequenziale. Considerate un programma con la seguente numerazione di righe:

120
130
140
150
160
170
180
190

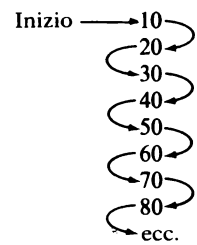
Se si inserisce una riga con il numero 165, la nuova istruzione apparirà inizialmente sotto alla riga 190, ma il computer la inserirà poi automaticamente tra le righe 160 e 170.

Numerazione delle istruzioni prima dell'inserimento della 165	Stesso elenco riorganizzato
120	120
130	130
140	140
150	150
160	160
170	165
180	170
190	180
	190
165	

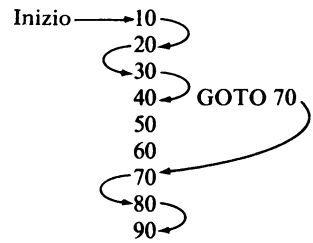
Qualora il numero di una nuova riga fosse uguale a quello di una già esistente, l'istruzione nuova rimpiazzerà quella vecchia.
Il BASIC del C-64 permette numeri di riga da 1 a 63999. Le cifre poste all'inizio di una riga vengono interpretate come il numero della riga. Se il numero è superiore a 63999, verrà trasmesso un messaggio d'errore sintattico. Tutti i dialetti BASIC richiedono la numerazione delle righe come descritto sopra: tuttavia il numero massimo permesso varia da un dialetto all'altro.
I numeri di riga vengono usati come *indirizzi* che identificano la posizione delle istruzioni all'interno di un programma. Questo è un concetto importante, dato che ogni programma contiene i seguenti due tipi di istruzione:

1. Istruzioni che creano o modificano dati.
2. Istruzioni che controllano la sequenza in cui le operazioni vengono eseguite.

L'idea che le operazioni di un programma debbano essere eseguite in una sequenza ben definita è semplice da capire. L'esecuzione di un programma normalmente comincia con la prima istruzione e prosegue in ordine di numerazione.



Tuttavia la maggior parte dei programmi contiene alcune sequenze di istruzioni che non sono in ordine sequenziale. È qui che i numeri di riga diventano importanti, perché sono usati per identificare un cambiamento nella sequenza di esecuzione.



DATI

L'istruzione che segue un numero di riga specifica le operazioni che il computer deve fare, ed inoltre fornisce i dati con cui eseguire queste operazioni. Si descriveranno ora i diversi tipi di dati che è possibile inserire in un programma BASIC sul C-64.
Ci sono due tipi di numeri che possono essere memorizzati dal C-64: numeri a virgola mobile (detti anche numeri reali) e numeri interi.

Numeri a virgola mobile

La virgola mobile è la rappresentazione standard dei numeri usati dal C-64. Tutte le operazioni aritmetiche sono eseguite usando numeri a virgola mobile. Il nome si riferisce alla possibilità della virgola di muoversi, permettendo così l'uso di numeri con diverse cifre decimali. Un numero a virgola mobile può consistere in tutto il numero o nella parte decimale preceduta da un punto. Il numero può essere negativo (−) o positivo (+). Se non ha segno è considerato positivo. Ecco alcuni numeri a virgola mobile che sono equivalenti a numeri interi:

```
5
-15
65000
161
0
```

Ed ecco invece alcuni esempi di numeri a virgola mobile che contengono un punto decimale:

```
0.5
0.0165432
-0.0000009
1.6
24.0055
-64.2
3.1416
```

Notate che mettendo una virgola (invece di un punto) nei numeri si otterrà un messaggio d'errore sintattico: scrivete quindi 0.5 e non 0,5.

Arrotondamento

I numeri hanno sempre almeno otto cifre di precisione; possono averne in qualche caso fino a nove. Il BASIC del C-64 arrotonda le cifre significative ulteriori. Normalmente arrotonda per eccesso quando la successiva cifra è 5 o più e arrotonda per difetto quando essa è 4 o meno, ma esistono degli arrotondamenti anomali.
Eccone alcuni esempi:

```
?.5555555556
 .555555555
```

```
?.5555555557
 .555555556
```

Sembra arrotondare per difetto se l'ultima cifra è 6 o meno, per eccesso se è 7 o più

```
?.1111111115
 .111111111
```

```
?.1111111116
 .111111112
```

Sembra arrotondare per difetto se l'ultima cifra è 5 o meno, per eccesso se è 6 o più

Notazione esponenziale

Numeri a virgola mobile molto grossi vengono presentati con la notazione esponenziale. Quando si inseriscono numeri con più di dieci cifre, il BASIC del C-64 li converte automaticamente in questa forma:

```
READY.
?1111111114
 1.11111111E+09
```

```
READY.
?1111111115
 1.11111112E+09
```

Un numero in notazione esponenziale ha la seguente composizione:

<numero>E ± <ee>

dove

numero è un intero o frazionario. Questa parte contiene le cifre significative del numero; è chiamata "coefficiente". Se non appare nessun punto decimale si presume che esso sia alla destra del coefficiente.
E sta per "esponente".
± è il segno positivo oppure quello negativo.
ee è l'esponente a una o due cifre. L'esponente specifica la grandezza del numero: il numero di posizioni a destra (per esponenti positivi) o a sinistra (per esponenti negativi) di cui deve essere spostato il punto decimale per ottenere il numero impostato in notazione standard.

Ecco alcuni esempi.

Notazione esponenziale	*Notazione standard*
2E1	20
10.5E+4	105000
66E+2	6600
66E−2	0.66
−66E−2	−0.66
1E−10	0.0000000001
94E20	9400000000000000000000

La notazione esponenziale è una forma conveniente per esprimere numeri molto grandi o molto piccoli. Il BASIC del C-64 accetta numeri tra 0.01 e 999 999 999 usando notazione standard; i numeri al di fuori di questi limiti sono scritti in notazione esponenziale.

```
?.009
 9E-03

READY.
?.01
 .01

READY.
?999999998.9
 9999999999
```

```
READY.
 ?999999999.6
  1E+09
```

Anche usando la notazione esponenziale c'è un limite alla grandezza dei numeri che il BASIC del C-64 è in grado di trattare. I limiti sono:

Numero a virgola mobile massimo: +1.70141183E+38
Numero a virgola mobile minimo: +2.93873588E−39

Qualsiasi numero di grandezza superiore darà un errore di overflow.

```
?1.70141183E+38
 1+70141183E+38

READY.
?-1.70141183E+38
-1.70141183E+38
```

Nessun errore di overflow

```
READY.
?1.70141184E+38

?OVERFLOW ERROR
READY.
?-1.70141184E+38

?OVERFLOW ERROR
```

Errore di overflow

Qualsiasi numero inferiore alla grandezza minima verrà espresso come zero.

```
?2.93873588E-39
 2.93873588E-39

READY.
?-2.93873588E-39
-2.93873588E-39

READY.
?2.93873587E-39
0

READY.
?-2.93873587E-39
0
```

Un numero intero è un numero che non ha punto decimale. Può essere negativo (−) o positivo (+). Un numero senza segno si presume positivo. Per essere memorizzati dal computer, i numeri interi devono essere compresi tra −32768 e +32767.

```
0
1
44
32699
-15
```

Qualsiasi numero intero può essere rappresentato come numero a virgola mobile, dato che gli interi sono un sottoinsieme dei numeri reali. Il BASIC del C-64 in ogni caso converte automaticamente i numeri interi in numeri a virgola mobile prima di usarli nei calcoli.

Stringhe

Il termine stringa è usato per descrivere dati formati da una sequenza di caratteri. Questi possono essere qualsiasi cosa che non debba venir interpretata come un numero. Abbiamo già usato delle stringhe per far apparire dei messaggi sul video del C-64. Una stringa è composta da uno o più caratteri scritti tra virgolette.

```
"SALVE!"
"PIPPO"
"12345"
"IL PREZZO È 12000"
"VIALE AOSTA, 32 - 20121 MILANO"
```

In una stringa si può includere qualunque carattere alfabetico o numerico, simboli speciali o grafici, caratteri di controllo del cursore (CLR/HOME, CRSR UP/DOWN, CRSR LEFT/RIGHT) e il tasto RVS ON/OFF. Gli unici tasti che non possono essere inclusi in una stringa sono RUN/STOP, RETURN e INST/DEL. Tutti i caratteri di una stringa appaiono così come sono inseriti. I tasti di controllo del cursore, di controllo del colore e RVS ON/OFF tuttavia non scrivono nulla di per sè. Per visualizzarli nella stringa sono usati alcuni simboli elencati nella tabella 3.1.

Tabella 3.1. Simboli speciali

Funzione	*Tasto*	*Simbolo*
Reverse On	CTRL RVS ON	(R inverso)
Reverse Off	CTRL RVS OFF	(R inverso shiftato)
Home Cursor	CLR HOME	(S inverso)
Clear Screen	Shift CLR HOME	(S inverso shiftato)
Cursor Down	CRSR	(Q inverso)
Cursor Up	Shift CRSR	(Q inverso shiftato)
Cursor Right	CRSR	(] inverso)
Cursor Left	Shift CRSR	(H inverso shiftato)

Variabili

Il concetto di variabile è molto facile da capire. Considerate le seguenti istruzioni:

```
100 A=B+C
200 ?A
```

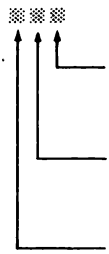
Queste due istruzioni fanno apparire sul video la somma di due numeri, ovvero di ciò che B e C rappresentano al momento in cui l'istruzione viene eseguita. Nel seguente esempio

```
90 B=4.65
95 C=3.72
100 A=B+C
200 ?A
```

a B è assegnato il valore 4.65, mentre a C è assegnato il valore 3.72. Quindi A è uguale a 8.37.

Nomi delle variabili

I nomi delle variabili possono essere usati per rappresentare dati numerici o stringhe. Se non avete mai studiato l'algebra elementare, pensate ad un nome di variabile come ad una casella postale. Qualunque cosa sia contenuta nella casella postale diviene il valore assegnato a quell'indirizzo. Un nome di variabile può avere uno, due o tre caratteri; sono permesse le seguenti opzioni:



Il terzo carattere deve essere $ per una stringa o % per una variabile intera. Una variabile a virgola mobile può avere solo due caratteri.

Il secondo carattere può essere qualunque lettera maiuscola (da A a Z) o qualsiasi cifra (1, 2, 3, 4, 5, 6, 7, 8, 9, 0) per qualunque tipo di variabile.

Il primo carattere deve essere una lettera maiuscola (da A a Z) per qualunque tipo di variabile.

Quindi l'ultimo carattere del nome di una variabile distingue, per il BASIC del C-64, il tipo di dato rappresentato dalla variabile.
Notate che per il primo e secondo carattere di un indirizzo, sono usate lettere senza SHIFT inserito; a seconda del modello di C-64, le lettere senza SHIFT inserito possono essere minuscole o maiuscole. Comunque sia sono le lettere che appaiono quando il tasto SHIFT non è premuto.
Le variabili a virgola mobile sono le più usate nel BASIC del C-64.
Ecco alcuni esempi di nomi di variabili a virgola mobile,

A
B
C
A1
AA
Z5

nomi di variabili intere,

A%
B%
C%
A1%
MN%
X4%

e nomi di variabili a stringa.

```
A$
M$
MN$
M1$
ZX$
F6$
```

Tutti questi nomi possono avere più di due caratteri alfanumerici, ma solo i primi due contano per l'identificazione. Quindi BANANA e BANDIERA vengono interpretati come lo stesso nome, dato che entrambe cominciano per BA. Il BASIC del C-64 permette di avere fino a 86 caratteri nei nomi di variabili, tuttavia nomi così lunghi sono poco pratici. Da quattro a otto caratteri è un limite più realistico; i nomi lunghi possono addirittura rendere la lettura del vostro programma più difficoltosa. L'esempio illustra come vengono interpretati i nomi lunghi dal BASIC C-64.

MAGIC$	*interpretata come*	MA$
N123456789	*interpretata come*	N1
MMM$	*interpretata come*	MM$
ABCDEF%	*interpretata come*	AB%
CALENDAR	*interpretata come*	CA

Usando nomi di variabili con più di due caratteri si deve tenere a mente quanto segue:

1. Solo i primi due caratteri più il simbolo identificatore ($ o %) sono significativi. Non usare nomi come LOOP1 e LOOP2; questi vengono interpretati come la stessa variabile: LO.
2. Il BASIC C-64 ha un certo numero di parole riservate che hanno un significato speciale nelle istruzioni. Esse comprendono istruzioni come PRINT ed altre che verranno trattate in seguito. Nessun nome di variabile può contenere, al suo interno, una parola riservata. Per esempio, non si può usare PRINTER come nome di variabile perché il BASIC lo interpreterebbe come "PRINT ER". Questo problema si rivela spesso come un errore di sintassi in una riga altrimenti corretta. La tabella 3.4 mostra una lista completa di parole riservate.
3. Caratteri addizionali occupano spazio in memoria che potrebbe essere necessario in programmi più lunghi. D'altra parte nomi di variabili più lunghi rendono più leggibili i programmi. NUMTEL ad esempio è più significativo di NT come nome di variabile che descrive il numero di telefono di un amico in un programma di archivio.

OPERATORI

L'istruzione BASIC

```
100 ?10.2+4.7
```

ordina al computer di sommare 10.2 e 4.7 e di far apparire il risultato. L'istruzione

```
250 C=A+B
```

ordina al C-64 di sommare i due numeri a virgola mobile rappresentati dalle variabili A e B, e di assegnare il risultato alla variabile a virgola mobile rappresentata da C.
Il segno + specifica l'addizione, e pertanto si chiama *operatore*. Esso è un operatore aritmetico, poiché la somma è un'operazione aritmetica. Vi sono altri due tipi di operatori: relazionali e Booleani. Questi ultimi necessitano di qualche ulteriore spiegazione dato che esprimono condizioni e decisioni piuttosto che operazioni aritmetiche.

Tabella 3.2. Operatori

	Priorità	*Operatore*	*Significato*
	9	( )	Parentesi (definiscono l'ordine di esecuzione)
Operatori Aritmetici	8	↑	Elevamento a potenza
	7	—	Segno negativo
	6	*	Moltiplicazione
	6	/	Divisione
	5	+	Addizione
	5	—	Sottrazione
Operatori Relazionali	4	=	Uguale
	4	<>	Diverso
	4	<	Minore
	4	>	Maggiore
	4	< = o ≤	Minore o uguale
	4	> = o ≥	Maggiore o uguale
Operatori Booleani	3	NOT	Complemento logico
	2	AND	AND logico
	1	OR	OR logico

Operatori aritmetici

Un operatore aritmetico specifica addizione, sottrazione, moltiplicazione, divisione o elevamento a potenza. Le operazioni aritmetiche sono eseguite con numeri a virgola mobile; i numeri interi sono convertiti automaticamente in virgola mobile prima di eseguire un'operazione aritmetica e poi riconvertiti ad interi se la variabile che rappresenta il risultato è di tipo intero. I dati su cui sono eseguite le operazioni si chiamano operandi. Gli operatori aritmetici richiedono due operandi ciascuno, che possono essere numeri, variabili numeriche o una combinazione di tutt'e due.

Addizione (+). Il segno + specifica che il dato (o operando) a sinistra del segno è da addizionare al dato (o operando) a destra dello stesso. Per quantità numeriche questa è una semplice addizione.
Il segno + è usato anche per "sommare" delle stringhe. In questo caso esso non addiziona i valori delle stringhe in senso proprio; le stringhe vengono unite o *concatenate* fra loro per formare una stringa più lunga. La differenza tra l'addizione numerica e il concatenamento di stringhe può essere visualizzato come segue:

Addizione di numeri: num1+num2 = num3
Addizione di stringhe: stringa1+stringa2 = stringa1stringa2

Usando il concatenamento si possono sviluppare stringhe lunghe fino a 255 caratteri.

"AUTO"+"MOBILE"	dà "AUTOMOBILE"
"BUONA"+" "+"SERA"	dà "BUONA SERA"
A$+B$	dà il concatenamento delle due stringhe rappresentate dalle due variabili A$ e B$
"1"+CH$+E$	dà il carattere "1" seguito dal concatenamento delle due stringhe rappresentate dalle variabili CH$ e E$

Se A$ è uguale a "AUTO" e B$ a "MOBILE", allora A$+B$ dà lo stesso risultato di "AUTO"+"MOBILE".
Provando a costruire una stringa di più di 255 caratteri si otterrebbe un messaggio d'errore: STRING TOO LONG.

Sottrazione (−). Il segno meno specifica che l'operando alla destra del segno va sottratto a quello alla sinistra dello stesso.

4−1	dà 3
100−64	dà 36

A−B	significa che la variabile B viene sottratta dalla variabile rappresentata da A
55−142	dà −87

Il segno meno da solo identifica un numero negativo. Per esempio,

−5	
−9E4	
−B	
4−−2	Notate che 4−−2 è uguale a 4+2

Moltiplicazione (*). Un asterisco specifica che l'operando alla sua destra deve essere moltiplicato per l'operando alla sinistra dello stesso.

100*2	dà 200
50*0	dà 0
A*X1	dà la moltiplicazione di due numeri a virgola mobile rappresentati dalle variabili A e X1
R%*14	dà la moltiplicazione di un numero intero rappresentato da R% per 14

Nell'esempio qui sopra, se alla variabile A si assegna il valore 4.2 e alla variabile X1 il valore 9.63, la risposta è 40.446. A e X1 potrebbero contenere i valori interi 100 e 2 per duplicare il primo esempio; tuttavia, i due numeri verrebbero memorizzati come 100.00 e 2.0, dato che A e X1 sono variabili a virgola mobile. Per moltiplicare 100 per 2, rappresentando questi due numeri come interi, l'esempio avrebbe dovuto essere A%*X1%.

Divisione (/). La barra specifica che l'operando alla sua sinistra deve essere diviso per il dato alla sua destra.

10/2	dà 5
6400/4	dà 1600
A/B	significa che il numero a virgola mobile rappresentato dalla variabile A viene diviso per quello rappresentato da B
4E2/XR	significa che 400 viene diviso per il numero a virgola mobile rappresentato da XR.

Il terzo esempio, A/B, può duplicare il primo o il secondo anche se A e B rappresentano numeri a virgola mobile; sarebbe più esatto comunque scrivere A%/B%.

Elevamento a potenza (↑). La freccia rivolta verso l'alto specifica che l'operando alla sinistra della freccia è elevato alla potenza specificata

dall'operando che si trova alla destra. Se il dato alla destra è 2, il numero alla sinistra viene elevato al quadrato, se è 3 quello a sinistra viene elevato al cubo, e così via. La potenza può essere qualsiasi numero, variabile o espressione, purché il risultato rimanga entro i limiti accettati per numeri a virgola mobile.

2 ↑ 2	dà 4
12 ↑ 2	dà 144
1 ↑ 3	dà 1
A ↑ 5	significa che il numero rappresentato da A è elevato alla 5ª potenza
2 ↑ 6.4	dà 84.4485064
NM ↑ −10	significa che il numero a virgola mobile assegnato alla variabile NM è elevato alla decima potenza negativa
14 ↑ F	significa che 14 è elevato alla potenza specificata dalla variabile a virgola mobile F

Ordine di calcolo

Un'espressione può contenere più operazioni aritmetiche, così come nella seguente istruzione:

```
A+C * 10/2 ↑ 2
```

In questi casi c'è un ordine fisso in cui vengono eseguite le operazioni. Prima viene l'elevamento a potenza (↑), seguito dalla valutazione del segno, poi da moltiplicazione e divisione (* /), e infine vengono l'addizione e la sottrazione (+ −). Operazioni di uguale priorità vengono eseguite da sinistra a destra. Quest'ordine può essere modificato dall'uso di parentesi: qualsiasi operazione tra parentesi viene eseguita per prima. Per esempio,

4+1 * 2	dà 6
(4+1) * 2	dà 10
100 * 4/2−1	dà 199
100 * (4/2−1)	dà 100
100 * (4/(2−1))	dà 400

Quando sono presenti delle parentesi, il BASIC C-64 calcola prima il risultato delle parentesi più interne, poi di quelle immediatamente più esterne e così via fino ad arrivare a quelle più esterne di tutte. Le parentesi si possono "annidare" fino a qualsiasi livello e possono essere usate liberamente per chiarire l'ordine di esecuzione delle operazioni di una espressione.

Operatori relazionali

Gli operatori relazionali rappresentano le seguenti condizioni: maggiore di (>), minore di (<), uguale (=), diverso (<>), maggiore o uguale (>=) e minore o uguale (<=).

1 = 5−4	dà un risultato vero (−1)
14>66	dà un risultato falso (0)
15>= 15	dà un risultato vero (−1)
A<>B	il risultato dipenderà dai valori assegnati alle variabili a virgola mobile A e B

Il BASIC C-64 assegna arbitrariamente un valore 0 ad una condizione "falsa" e un valore −1 ad una condizione "vera". Questi valori 0 e −1 possono essere usati in equazioni. Per esempio, nell'espressione (1=1) * 4, l'equazione (1=1) è vera. Vero ha un valore di −1, quindi l'espressione è uguale a (−1) * 4, che risulta perciò essere −4. Si può usare qualunque operatore relazionale in espressioni in BASIC C-64. Ecco alcuni ulteriori esempi.

25+(14>66)	è uguale a 25+0
(A+(1 = 5−4)) * (15 >= 15)	è uguale a (A−1) * (−1)

Gli operatori relazionali possono essere usati per confrontare le stringhe. A questo scopo va tenuto presente che le lettere dell'alfabeto hanno un ordine A<B, B<C, C<D e così via. Le stringhe sono confrontate un carattere alla volta, cominciando da quello più a sinistra.

"A"<"B"	è vero (−1)
"X" = "XX"	è falso (0)
C$ = A$+B$	il risultato dipende dai valori assegnati alle variabili C$, B$ e A$

("GIORGIO">"LUCA")+37	è uguale a −1+37
("AAA"<"AA") * (Z9−("OTTO">"AB"))	è uguale a 0 * (Z9−(−1))

Operatori Booleani

Gli operatori Booleani danno ai programmi la possibilità di prendere decisioni logiche. Ci sono tre operatori Booleani nel BASIC C-64: AND, OR e NOT. Una semplice analogia con la vita quotidiana può servire per illustrare la logica Booleana.
Si supponga di dover comperare caramelle con due bambini. L'operatore

Booleano AND permette di comperare un certo tipo di caramella solo nel caso che il bambino A *e* il bambino B scelgano quel tipo.
L'operatore OR permette di acquistare un tipo di caramella nel caso in cui *o* l'uno *o* l'altro scelgano quel tipo.
L'operatore NOT genera un opposto logico. Il bambino B insiste nel non accordarsi con A; allora la scelta di B è sempre il contrario della scelta di A.
La tabella 3.3 riassume il modo in cui gli operatori Booleani trattano i numeri. Essa viene chiamata tabella della verità. Gli operatori Booleani controllano principalmente la logica dei programmi.
Eccone alcuni esempi:

```
IF A = 1ØØ AND B = 1ØØ GOTO 1Ø
    Se sia A e B sono uguali a 100, saltare a riga 1Ø
IF X<Y AND B>= 44 THEN F = Ø
    Se X è minore di Y e B è maggiore o uguale a 44 allora assegnare
        il valore 0 ad F
IF A = 1ØØ OR B = 1ØØ GOTO 2Ø
    Se una delle due variabili (o anche tutt'e due) è uguale a 100,
        saltare a riga 2Ø
IF X<Y OR B> =44 THEN F = Ø
    F è fissata uguale a 0 se X è minore di Y oppure se B è maggiore di 43
IF A = 1 AND B = 2 OR C = 3 GOTO 3Ø
    Fare il salto a riga 3Ø qualora si verifichi che A e B sono rispettivamente
    1 e 2, oppure se C = 3. Infatti nel caso si trovino più operatori Boolea-
      ni di seguito, prima vengono eseguiti gli AND e poi gli OR.
```

Ogni operando viene controllato per vedere se è vero o falso. Un operando che appare da solo è implicitamente seguito da "< >0"; qualunque valore che non sia zero viene considerato vero, lo zero è considerato falso.

```
IF A THEN B = 2
IF A< > Ø THEN B = 2
    Le due istruzioni sopra sono equivalenti.
IF NOT B GOTO 1ØØ
    Saltare se B è falso, ossia uguale a zero.
    Potrebbe anche venir scritto come
IF B = Ø GOTO 1ØØ
```

Tutte le operazioni Booleane usano operandi interi. Se si cerca di fare operazioni Booleane con numeri a virgola mobile, questi ultimi sono trasformati in interi, per cui devono rimanere entro i limiti dei numeri interi. Se siete dei principianti è improbabile che vi venga automatico usare gli operatori Booleani nel modo che adesso descriveremo. Se non vi interes-

sa approfondire questo argomento, passate subito al paragrafo successivo.

Tabella 3.3. Tabella della verità

AND dà risultato 1 se entrambi i numeri sono uguali a 1

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

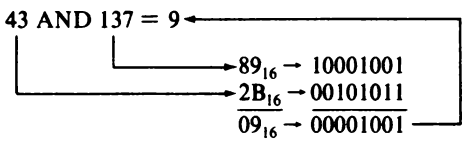
OR dà risultato 1 se almeno un numero è uguale a 1

1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0

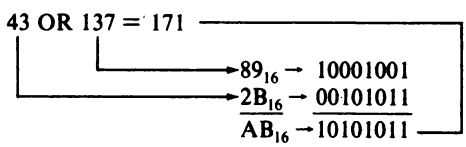
NOT dà il complemento logico

NOT 1 = 0
NOT 0 = 1

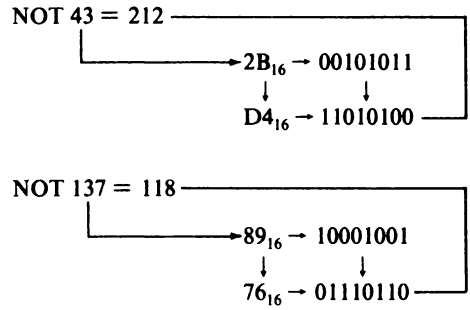
Gli operatori Booleani agiscono sugli operandi interi su una cifra binaria alla volta. Il BASIC C-64 memorizza tutti i numeri in forma binaria, utilizzando la notazione di complemento a due per rappresentare i numeri negativi. Passando per facilità di lettura attraverso la notazione esadecimale, si può illustrare una operazione AND come segue:



Ecco qui un'operazione OR.



Ed ecco due operazioni NOT.



Gli operandi sono sempre trasformati nel formato intero; l'operazione Booleana viene eseguita ed il risultato è fornito con uno 0 o un 1. Se un operatore Booleano ha operandi relazionali, questi ultimi sono calcolati come −1 o 0 prima di eseguire l'operazione Booleana. Per cui l'operazione

```
A = 1 OR C<2
```

è equivalente a

$$\left\{\begin{matrix}-1\\0\\\emptyset\end{matrix}\right\} \text{OR} \left\{\begin{matrix}-1\\0\\\emptyset\end{matrix}\right\}$$

Si consideri questa operazione più complessa:

```
IF A = B AND C<D GOTO 4Ø
```

Prima sono calcolate le espressioni relazionali; supponete che la prima espressione sia vera e la seconda falsa: l'istruzione allora diventa

```
IF −1 AND Ø GOTO 4Ø
```

Eseguendo l'operazione AND si ottiene un risultato 0.

```
IF Ø GOTO 4Ø
```

Ricordate che un singolo operando è implicitamente seguito da "< >0". L'espressione diventa quindi

```
IF Ø < > Ø GOTO 4Ø
```

Per cui il salto non è eseguito.
Per contro un'operazione Booleana eseguita su due variabili può dare come risultato qualunque numero intero.

```
IF A% AND B% GOTO 4Ø
```

Si supponga che A% = 255 e che B% = 240. L'operazione Booleana 255 AND 240 dà 240. L'istruzione diventa quindi

```
IF 24Ø GOTO 4Ø
```

oppure con "< > Ø",

```
IF 24Ø < > Ø GOTO 4Ø
```

Quindi il salto verrà eseguito.
Ora paragonate le seguenti istruzioni di assegnazione:

```
A = A AND 1Ø
A = A<1Ø
```

Nel primo esempio il valore in quel momento di A è logicamente "ANDato" con 10 ed il risultato diviene il nuovo valore di A. A deve trovarsi entro i limiti degli interi tra −32768 e +32767. Nel secondo esempio l'espressione relazionale A<10 è valutata a −1 o 0, quindi A finisce per avere un valore di −1 o 0.

GLI ARRAY

Gli array vengono largamente usati in numerosi programmi per computer. Le informazioni che seguono sono molto importanti per sviluppare le vostre capacità di programmazione.
Concettualmente gli array sono semplici: quando si hanno due o più elementi di dati connessi fra loro, invece di assegnare ad ogni elemento un nome diverso, si dà all'elenco di dati collegati un unico nome. Poi si selezionano i singoli elementi individuali usando un numero di posizione, che nel gergo informatico si chiama indice.
Una lista della spesa, per esempio, può essere composta da sei articoli di carne e pesce, quattro di frutta e verdura e tre di prodotti di latteria; questi tre gruppi di elementi potrebbero essere rappresentati ognuno da un solo nome di variabile come segue:

```
CP$(0) = "BISTECCHE"
CP$(1) = "POLLO NOVELLO"
CP$(2) = "CARNE TRITA"
CP$(3) = "FILETTI DI SOGLIOLA"
CP$(4) = "GAMBERETTI"
CP$(5) = "TRIPPA"

FV$(0) = "ARANCE"
FV$(1) = "CAVOLFIORI"
FV$(2) = "FAGIOLI"
FV$(3) = "FRAGOLE"

PL$(0) = "LATTE"
PL$(1) = "BURRO"
PL$(2) = "PARMIGIANO"
```

CP$ è un unico nome di variabile che identifica tutti i prodotti di carne e di pesce. FV$ identifica tutta la frutta e verdura, mentre PL$ identifica i prodotti di latteria.

Un indice segue ogni nome di variabile, per cui ogni dato viene identificato con un nome di variabile e un indice. Notate che il primo indice negli esempi è 0, non 1. Gli indici in BASIC cominciano da 0 perché ciò semplifica la programmazione di molti problemi scientifici e matematici. Molti si trovano a disagio con questa convenzione: se non vi trovate bene usando l'elemento 0 di un array, potete semplicemente ignorarlo e cominciare con 1. Si sprecherà un po' di spazio di memoria, ma se non si riesce ad "adattarsi" alla macchina si correranno meno rischi di errori di programmazione. Si può portare il concetto degli array avanti di un altro passo, chiamando l'intera lista con un unico nome, ma usando due indici. Il primo indice specifica il gruppo ed il secondo l'elemento di ogni gruppo. In questo modo un singolo array variabile con due indici potrebbe rimpiazzare gli array ad un solo indice usati nell'esempio precedente.

```
LS$(0,0) = CP$(0)    LS$(1,0) = FV$(0)    LS$(2,0) = PL$(0)
LS$(0,1) = CP$(1)    LS$(1,1) = FV$(1)    LS$(2,1) = PL$(1)
LS$(0,2) = CP$(2)    LS$(1,2) = FV$(2)    LS$(2,2) = PL$(2)
LS$(0,3) = CP$(3)    LS$(1,3) = FV$(3)
LS$(0,4) = CP$(4)
LS$(0,5) = CP$(5)
```

Gli array possono contenere variabili intere, variabili a virgola mobile o stringhe, ma ogni array può contenere un solo tipo di dati. In altre parole, una singola variabile non può mischiare numeri interi e a virgola mobile. Gli array sono un comodo sistema di descrivere un gran numero di variabili connesse. Si consideri una tabella contenente 10 righe di numeri, con 20 numeri in ogni riga: in tutto 200 numeri. Pensate ad assegnare nominativi individuali ad ognuno di quei 200 numeri! Sarebbe molto più semplice dare un nome all'intera tabella e identificare i singoli numeri dalla loro posizione, che è esattamente ciò che succede in un array.
Gli array possono avere una o più *dimensioni:* quelli ad una sola dimensione sono equivalenti ad una tabella con una sola riga di numeri. L'indice identifica un numero nell'unica riga.

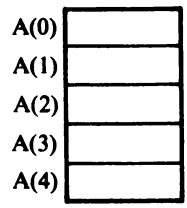
Un array a due dimensioni produce una normale tabella con righe e colonne: una dimensione identifica la riga, l'altra la colonna. Tre dimensioni producono un "cubo" di numeri, o se preferite, una pila di tabelle. Quattro o più dimensioni producono array sicuramente difficili da visualizzare, ma che, matematicamente, non sono più complessi dei loro fratelli minori.
Esaminiamo gli array più in dettaglio. Un singolo elemento di un array ha la seguente forma:

<nome>(i)

dove

nome è il nome della variabile array: può essere usato qualsiasi tipo di nome
i è l'indice dell'elemento dell'array: deve cominciare da 0.

Un array chiamato A e composto di cinque elementi, può essere visualizzato come segue:



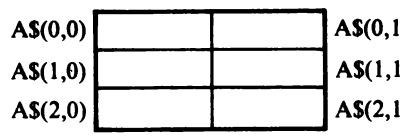
Il numero di elementi è uguale al numero indice più alto incrementato di uno; ciò perché tiene conto dell'elemento 0.
Un vettore a due dimensioni ha la seguente forma:

<nome>(i, j)

dove

nome è il nome della variabile array
i è l'indice della colonna
j è l'indice della riga.

Un array di stringhe a due dimensioni di nome A$, avente due colonne e tre righe, potrebbe essere visualizzato così:



La sua grandezza è il prodotto delle dimensioni più alte di riga e di colonna incrementate di 1. Nell'esempio è $3 \times 2 = 6$ elementi.
Ulteriori dimensioni possono essere aggiunte nella definizione:

<nome>(i, j, k...)

Gli array fino a 11 elementi (da 0 a 10 nel caso di una dimensione) possono essere usati comunemente nel BASIC del C-64. Quelli contenenti più di 11 elementi devono essere specificati in un'istruzione di dimensionamento che verrà descritta più avanti in questo capitolo.
Se non si inserisce l'indice dopo il nome di un array in un programma, questo sarà trattato dal BASIC del C-64 come una variabile diversa. Ciò può portare ad errori di programmazione difficili da rintracciare. Non è consigliabile sfruttare questa particolarità poiché altri linguaggi ed altri dialetti BASIC non funzionano nello stesso modo; questa tecnica potrebbe creare confusione ad altri programmatori che volessero leggere il vostro programma ed anche voi stessi potreste pensare, in un secondo tempo, di aver erroneamente omesso l'indice, e cercare di correggere l'"errore". Non chiamate mai quindi una variabile con lo stesso nome di un array.

3.2. I comandi BASIC

Nel secondo capitolo si è visto come certi comandi possono essere eseguiti sulla tastiera per controllare le operazioni del C-64; RUN è uno di questi. I comandi possono tutti essere eseguiti come istruzioni BASIC.

PAROLE RISERVATE

Tutte le combinazioni di caratteri che definiscono le operazioni di un'istruzione BASIC e tutte le funzioni sono parole riservate. La tabella 3.4 riporta queste parole per il BASIC C-64. Il C-64 quando esegue un programma in BASIC controlla ogni istruzione per verificare se contiene parole riservate, fatta eccezione per stringhe di caratteri racchiusi tra virgolette. Ciò può causare complicazioni qualora ci fossero parole riservate contenute nei nomi delle variabili: il C-64 non è in grado di identificare i nomi delle variabili dalla loro posizione nell'istruzione BASIC. Bisogna fare molta attenzione a tenere le variabili ben distinte dalle parole riservate, ricordandosi sempre che solo i primi due caratteri del nome di una variabile vengono utilizzati per l'identificazione.

Tabella 3.4. Tabella delle parole riservate e loro abbreviazioni

PAROLE	ABBREVIAZ. (1)	(2)	PAROLE	ABBREVIAZ. (1)	(2)
ABS	A I	aB	OPEN	O ¬	oP
AND	A /	aN	OR		
ASC	A ♥	aS	PEEK	P ¯	pE
ATN	A I	aT	POKE	P ¬	pO
CHR$	C I	cH	POS		
CLOSE	CL ¬	clO	PRINT	?	?
CLR	CL	cL	PRINT#	P _	pR
CMD	C \	cM	READ	R ¯	rE
CONT	C ¬	cO	REM		
COS			RESTORE	RE ♥	reS
DATA	D ♠	dA	RETURN	RE I	reT
DEF	D ¯	dE	RIGHT$	R \	rI
DIM	D \	dI	RND	R /	rN
END	E /	eN	RUN	R ?	rU
EXP	E ♠	eX	SAVE	S ♠	sA
FN			SGN	S I	sG
FOR	F ¬	fO	SIN	S \	sI
FRE	F _	fR	SPC(	S ¬	sP
GET	G ¯	gE	SQR	S ●	sQ
GET#			ST		
GOSUB	GO ♥	goS	STATUS		
GOTO	G ¬	gO	STEP	ST ¯	stE
IF			STOP	S I	sT
INPUT			STR$	ST _	stR
INPUT#	I /	iN	SYS	S I	sY
INT			TAB(	T ♠	tA
LEFT$	LE _	leF	TAN		
LEN			THEN	T I	tH
LET	L ¯	lE	TI		
LIST	L \	lI	TIME		
LOAD	L ¬	lO	TI$		
LOG			TIME$		
MID$	M \	mI	TO		
NEW			USR	U ♥	uS
NEXT	N ¯	nE	VAL	V ♠	vA
NOT	N ¬	nO	VERIFY	V ¯	vE
ON			WAIT	W ♠	wA

(1): SET STANDARD DI CARATTERI
(2): SET ALTERNATIVO (SHIFT/COMMODORE)

ABBREVIAZIONE DI PAROLE BASIC

Si è visto all'inizio di questo libro che l'istruzione PRINT poteva essere inserita dalla tastiera battendo "?". Il comando viene poi sviluppato dall'interprete del BASIC C-64 per formare l'intera parola PRINT.
La maggior parte dei comandi, delle istruzioni e delle funzioni in BASIC possono essere abbreviate usando le prime due lettere della parola, inserendo il secondo carattere con SHIFT premuto. Con il normale set di caratteri del C-64, il secondo carattere appare come un carattere grafico. Per esempio, l'abbreviazione di LIST appare come:

```
L↖
```

oppure come

```
lI
```

Il BASIC C-64 non fa distinzione tra le due abbreviazioni; entrambe sono interpretate come LIST. Se un'abbreviazione di due lettere è ambigua (ST vuol dire STEP o STOP?), l'abbreviazione è assegnata alla parola usata più frequentemente dalla tastiera e l'altra non viene abbreviata oppure vengono usati i primi tre caratteri, con il terzo inserito con SHIFT premuto. Ad esempio, STOP è abbreviato come

```
sT
SI
```

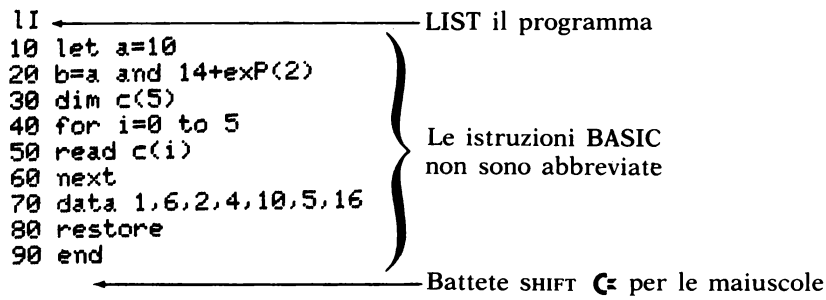
e STEP è abbreviato come

```
stE
ST▔
```

Per abbreviare STEP battete dunque una S maiuscola senza usare SHIFT, una T senza SHIFT e una E con SHIFT.
La seguente serie di semplici istruzioni usa abbreviazioni a due e tre lettere quando possibile. Tutte le abbreviazioni vengono poi espanse automaticamente quando si lista il programma.

←——— Battete SHIFT C= per le minuscole

```
10 lE a=10
20 b=a aN 14+eX(2)
30 dI c(5)
40 fO i=0 to 5
50 rE c(i)
60 nE
70 dA 1,6,2,4,10,5,16
80 reS
90 eN
```



Premendo SHIFT e i tasti simultaneamente si vedranno apparire le abbreviazioni con caratteri grafici al posto di quelli con SHIFT inserito. Il listato apparirà in caratteri maiuscoli.
Ricordatevi che le espansioni dalle abbreviazioni per le due funzioni SPC e TAB includono parentesi sinistre. Questo significa che se si usano le abbreviazioni per una di queste non si deve scrivere la parentesi sinistra dell'argomento. Per esempio,

```
10 ?sP(5)
```

viene espanso come

```
10 Print sPc((5)
```

Le due parentesi sinistre danno luogo ad un errore di sintassi

La corretta sequenza da inserire è

```
10 ?sP5)
```

Questa regola delle parentesi riguarda solo le funzioni SPC e TAB ed è una particolarità da tener presente. Per tutte le altre funzioni si usano entrambe le parentesi, per esempio:

```
10 ?rN(1)
```

3.3. Le istruzioni BASIC

L'operazione eseguita da un'istruzione è specificata usando una parola riservata (vedi tabella 3.4). Le istruzioni non sono descritte dettagliatamente in questo capitolo. Per una completa descrizione di tutte le istruzioni riconosciute dal BASIC C-64 si deve far riferimento alle appendici G e H. Questo capitolo è un'introduzione ai concetti di programmazione con particolare attenzione al modo in cui vengono utilizzate le istruzioni.

COMMENTI (REM)

Quando i primi tre caratteri di un'istruzione BASIC sono R E M (dall'inglese REMark=commento), il computer ignora completamente l'istruzione.
Può sembrare singolare cominciare una discussione sulle istruzioni BASIC con la descrizione dell'unica istruzione che il computer ignora. E allora perché includere un'istruzione di questo tipo? La risposta è che i commenti rendono più leggibile il programma.
Quando si scrive un programma breve di cinque o dieci righe non c'è nessuna difficoltà nel ricordarsi cosa esso faccia, a meno di non lasciarlo inutilizzato per sei mesi e poi cercare di usarlo di nuovo. Ma quando si scrivono programmi di 100 o 200 istruzioni è assai probabile dimenticare qualcosa d'importante anche il giorno dopo averlo scritto; dopo aver scritto dozzine di programmi poi, è impossibile ricordarseli tutti nei minimi particolari. La soluzione a questo problema è di documentare ogni programma inserendo dei commenti che descrivono quello che succede nelle sue varie parti. I bravi programmatori usano molte REM in tutti i loro lavori. Negli esempi in questo capitolo si fa uso di REM che descrivono le funzioni delle istruzioni semplicemente per iniziarvi a questa abitudine.

ISTRUZIONI DI ASSEGNAMENTO

Sono le istruzioni che permettono di assegnare i valori alle variabili; se ne incontreranno molto spesso in ogni tipo di programma in BASIC. Ecco alcuni esempi di istruzioni di assegnamento.

```
90 REM INIZIALIZZA LA VARIABILE X
100 LET X=3.24
```

Nell'istruzione 100, alla variabile a virgola mobile X è assegnato il valore 3.24

```
150 X=3.24
```

Istruzione equivalente alla 1ØØ precedente; il LET è opzionale in tutte le istruzioni di assegnamento

```
215 A$="EVVIVA"
```

Alla variabile a stringa A$ è assegnata una parola.

Notate che la prima istruzione di assegnamento (riga 1ØØ) comincia con la parola "LET", ma non le altre due. In origine tutte le istruzioni di questo tipo dovevano cominciare con LET: lo scopo era quello di far identificare il tipo di istruzione dalla prima parola. Oggigiorno praticamente tutti i dialetti BASIC hanno abbandonato questa prassi. Nonostante LET non sia richiesto dal BASIC C-64, rimane una parola riservata e pertanto non deve apparire in un nominativo di variabile. Vediamo tre istruzioni che assegnano valori all'array PL$(1), che si è incontrato in precedenza.

```
200 REM PL$(I) E' L'ARRAY "PRODOTTI DI LATTERIA"
210 PL$(Ø)="LATTE"
220 PL$(1)="BURRO"
230 PL$(2)="PARMIGIANO"
```

Ricordatevi che è possibile mettere più di una istruzione in ogni riga. Le tre istruzioni precedenti potrebbero essere messe su di una sola riga come segue:

```
200 REM PL$(I) E' L'ARRAY "PRODOTTI DI LATTERIA"
210 PL$(Ø)="LATTE":PL$(1)="BURRO":PL$(2)="PARMIGIANO"
```

I due punti (:) devono separare le istruzioni consecutive sulla stessa riga. Le istruzioni di assegnamento possono contenere tutti gli operatori aritmetici e relazionali descritti in questo capitolo.

```
90 REM QUESTO NON E' UN MODO INTELLIGENTE DI ASSEGNARE
UN VALORE A V
100 V=3.24+7.96/8.5
```

Questa istruzione assegna il valore 4,17647059 alla variabile a virgola mobile V. È equivalente alle tre istruzioni

```
100 X=7.96
110 Y=8.5
120 V=3.24+X/Y
```

che potrebbero essere scritte tutte su di una sola riga come segue:

```
100 X=7.96:Y=8.5:V=3.24+X/Y
```

Nel caso delle operazioni Booleane già descritte, avremo delle istruzioni di questo tipo:

```
90 REM QUESTI ESEMPI SONO STATI DESCRITTI
PRECEDENTEMENTE NEL CAPITOLO
100 A%=43 AND 137
200 B%=43 OR 137
```

L'esempio che segue mostra come a una variabile a stringa possa essere assegnato il valore usando il concatenamento di stringhe.

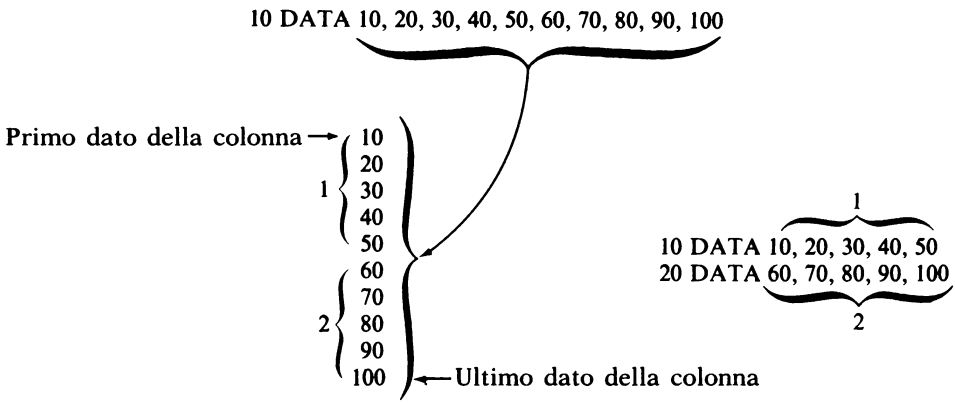
```
100 G$="GATTO"
200 S$="SORIANO"
300 X$=G$+" "+S$
400 REM A X$ E' ASSEGNATO IL VALORE "GATTO SORIANO"
```

Istruzioni DATA e READ

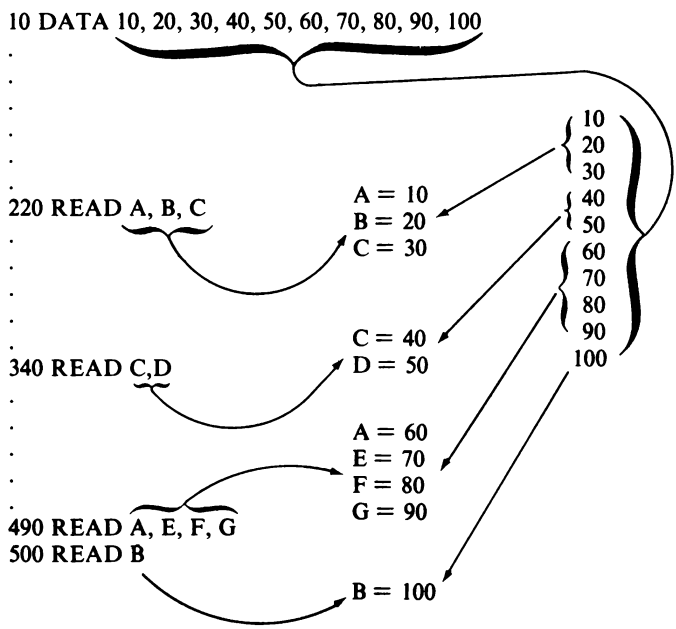
Quando il numero di variabili che hanno bisogno di assegnamento è alto, bisognerebbe usare, invece di LET, le istruzioni DATA e READ. Si consideri il seguente esempio:

```
 5 REM INIZIALIZZA TUTTE LE VARIABILI DEL PROGRAMMA
10 DATA 10,20,-4,16E6
20 READ A,B,C,D
```

L'istruzione a riga 1Ø riporta quattro dati numerici. Questi quattro valori sono assegnati alle quattro variabili a riga 2Ø. Dopo l'esecuzione delle istruzioni sulle righe 1Ø e 2Ø avremo $A = 10$, $B = 20$, $C = -4$, $D = 16 \times 10^6$. Una o più istruzioni DATA nel programma si possono visualizzare come una colonna di numeri. Ad esempio, un'istruzione DATA che contiene una lista di dieci numeri costituisce una colonna di dieci elementi. Due istruzioni DATA, ognuna costituita da cinque dati, costruirebbero esattamente la stessa colonna. Ciò si può illustrare come segue:

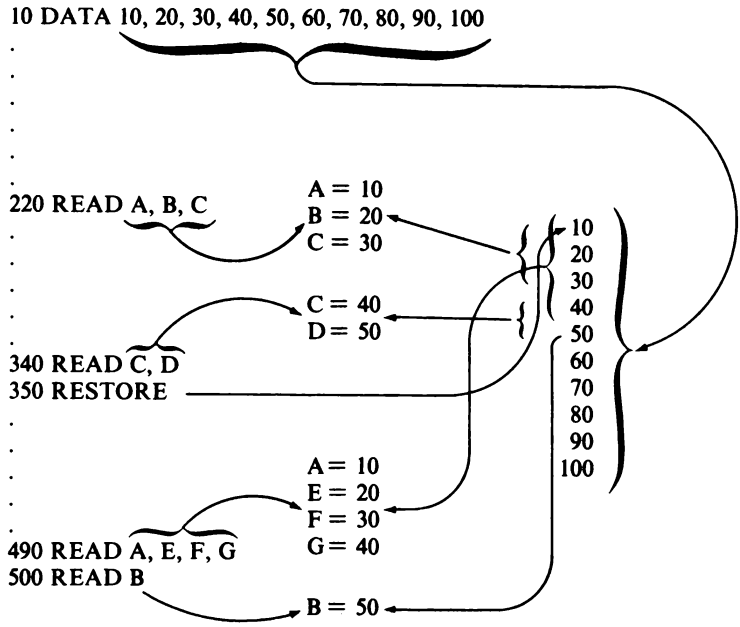


La prima istruzione READ eseguita in un programma comincia dal primo elemento della colonna, assegnando ogni valore alla corrispondente variabile scritta nell'istruzione READ. La seconda e le seguenti istruzioni READ prendono i valori dalla colonna cominciando dal punto in cui il precedente READ aveva smesso.



Istruzione RESTORE

Si può rimandare il puntatore all'inizio della colonna numerica in ogni momento eseguendo un'istruzione RESTORE. Ecco un esempio dell'uso di RESTORE.



ISTRUZIONI DI DIMENSIONAMENTO (DIM)

Il BASIC C-64 normalmente assume che una variabile abbia una sola dimensione, con valori dell'indice da 0 fino a 10; ciò genera un array a undici elementi. Volendo creare un array con più di undici elementi, si dovrà includere il suo nome in un'istruzione di dimensionamento (DIM). Ciò è necessario anche se questo è a due o più dimensioni, non importa quanti elementi abbia. Il seguente esempio definisce le dimensioni per gli array variabili CP$, FV$ e PL$. Si sono già usate queste variabili nella precedente discussione sugli array.

```
DIM CP$(5),FV$(3),PL$(2)
```

L'array a due dimensioni che rappresenta la lista della spesa, LS$, va dimensionato come segue:

```
DIM LS$(3,5)
```

Un'istruzione DIM può assegnare dimensioni a parecchie variabili, purché stiano su una riga di 80 caratteri. Il numero che segue un nome in un'istruzione DIM è uguale al valore più alto possibile dell'indice per quella particolare dimensione; ricordatevi che gli indici cominciano da 0. Per cui DIM CP$(5) dimensiona CP$ ad avere sei valori, non cinque, dato che saranno accettati gli indici 0, 1, 2, 3, 4 e 5. Altrettanto dicasi per DIM LS$(3,5), che specifica una variabile bidimensionale a 24 elementi, dato che la prima dimensione può avere i valori 0, 1, 2 e 3 e la seconda da 0 fino a 5. Una volta dichiarato un array in un'istruzione di dimensionamento, bisogna in seguito sempre riferirsi ad esso specificando il numero degli indici; ogni indice dovrà avere un valore da 0 al numero indicato nella DIM. Se una qualunque di queste regole viene infranta si ottiene un errore di sintassi.

ISTRUZIONI DI SALTO

Le istruzioni di un programma in BASIC vengono normalmente eseguite in ordine crescente di numero. La sequenza di esecuzione è stata spiegata in precedenza in questo capitolo con la descrizione dei numeri di riga. Le istruzioni di salto cambiano questa sequenza d'esecuzione.

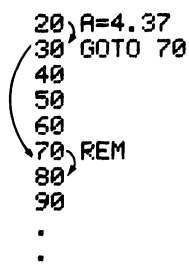
Istruzione GOTO

GOTO è l'istruzione di salto più semplice: permette di specificare l'istruzione che verrà eseguita immediatamente dopo. Si consideri il seguente esempio:

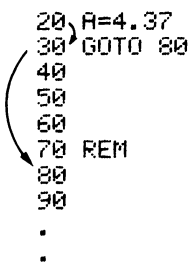
```
20 A=4.37
30 GOTO 100
40
50
60
70
80
90
100
110
.
.
```

L'istruzione alla riga 2Ø è un assegnamento: assegna un valore alla variabile a virgola mobile A. L'istruzione successiva è un GOTO: specifica che

l'esecuzione del programma deve saltare fino alla riga 1ØØ. Quindi la sequenza d'esecuzione delle istruzioni in questa parte del programma sarà: riga 2Ø, poi riga 3Ø e poi riga 1ØØ. Naturalmente un'altra istruzione dovrà far saltare il programma indietro a riga 4Ø, altrimenti le istruzioni da 4Ø a 9Ø non verrebbero mai eseguite.
Si può saltare a qualsiasi numero di riga, anche se questa è una REM: il computer ignora la REM e quindi l'effetto è quello di saltare alla riga successiva. Come esempio considerate il seguente salto:

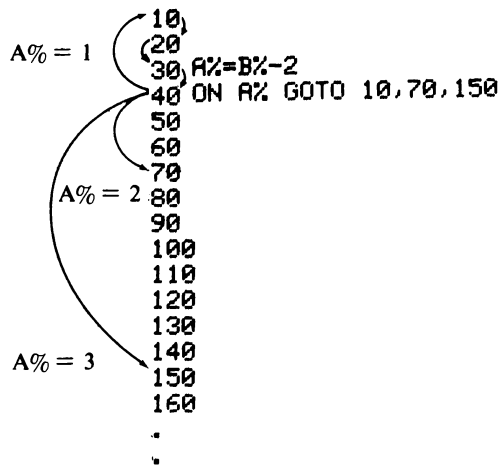


Il programma salta dalla 3Ø alla 7Ø: c'è solo una REM alla 7Ø, quindi il computer avanza alla riga 8Ø, eseguendo l'istruzione ivi contenuta. Anche se è permesso saltare ad una riga contenente una REM, tanto vale saltare alla riga successiva.



Istruzione GOTO calcolata

Esiste anche un'istruzione GOTO calcolata che permette alla logica del programma di saltare ad una fra due o più righe, a seconda del valore di una variabile in quel momento.



L'istruzione alla 4Ø è un GOTO calcolato. Quando viene eseguita questa istruzione il programma salta alla 1Ø se la variabile A% = 1, alla 7Ø se A% = 2, o alla 15Ø se A% = 3. Se A% assume un qualsiasi valore che non sia 1, 2 o 3, il programma non salta. Notate che alla variabile A% è assegnato un valore alla riga 3Ø. Il valore assegnato ad A% dipende dal valore corrente di B%. Il programma non mostra come sia calcolata B%, ma finché B% ha un valore di 3, 4 o 5 l'istruzione a riga 4Ø effettuerà il salto. Per controllare l'istruzione GOTO calcolata, caricate il seguente programma:

```
10 B%=4
20 PRINT B%
30 A%=B%-2
40 ON A%GOTO 10,70,150
70 PRINT B%
80 B%=5
90 GOTO 30
150 PRINT B%
160 B%=3
170 GOTO 20
```

Ora fate eseguire il programma inserendo RUN su una riga vuota. Non scrivete RUN su una riga che contenga già qualche cosa, perché otterrete un errore di sintassi ed il programma non partirà.
Siete in grado di capire il perché della sequenza di numeri visualizzata? Provate a modificare il programma in modo da far apparire le cifre nell'ordine 345345345...

ISTRUZIONI DI CONTROLLO

In ogni programma la sequenza in cui vengono eseguite le istruzioni è importante quanto le istruzioni stesse. Il BASIC C-64 dispone di varie istruzioni che controllano la maniera in cui il programma viene eseguito, da cui il nome istruzioni di controllo. Esse reindirizzano la sequenza d'esecuzione di un programma; alcune scelgono una tra le varie soluzioni possibili; altre eseguono una sequenza di istruzioni un numero specificato di volte.

IL COSTRUTTO FOR-NEXT

Le istruzioni GOTO semplice e GOTO calcolato permettono di svolgere il programma in qualunque sequenza sia richiesta dalla sua logica. Supponete di dover ripetere un'istruzione (o un gruppo d'istruzioni) molte volte. Per esempio, supponete di avere un array A(I) con 100 elementi e che a ognuno di essi si debba assegnare un valore da 0 a 99. Scrivere 100 istruzioni di assegnamento sarebbe laborioso; è molto più facile far eseguire una sola istruzione 100 volte, usando le istruzioni FOR e NEXT.

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 NEXT I
```

Le istruzioni comprese tra FOR e NEXT sono eseguite più volte. In questo caso tra di esse c'è una sola istruzione.
Per dimostrare come funzionano i loop FOR-NEXT faremo apparire i valori di A(I) creati all'interno del loop. Caricate il seguente programma:

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 REM SE AVETE UN'ISTRUZIONE GOTO CHE RIMANDA A SE
55 REM STESSA, IL COMPUTER ESEGUE UN LOOP
60 REM SENZA USCITA: INFATTI ASPETTA
90 GOTO 90
```

Ora battete RUN. Verranno presentati 100 numeri, da 0 a 99. Premete il tasto STOP per fermare l'esecuzione del programma. Le istruzioni comprese tra FOR e NEXT vengono eseguite il numero di volte specificato

dal valore dell'indice posto immediatamente dopo FOR. Nell'esempio sopra illustrato questo indice variabile è I, che aumenta di valore da 0 a 99 con incrementi di 1. La prima volta che l'istruzione di assegnamento (riga 3Ø) è eseguita, I sarà uguale a 0 e l'istruzione verrà eseguita come segue:

```
30 A(0)=0
```

I aumenta del passo (STEP) specificato, cioè di 1. La seconda volta che viene eseguita l'istruzione, I è uguale a 1 e l'istruzione stessa diventa

```
30 A(1)=1
```

I continua ad aumentare del valore dell'incremento specificato fino a che il valore massimo di 99 non è raggiunto o superato. Il valore dell'incremento non deve necessariamente essere 1; può avere qualunque valore. Cambiate il valore di incremento alla linea 2Ø da 1 a 5 ed eseguite di nuovo il programma; questa volta l'istruzione di assegnazione viene eseguita solo 20 volte, dato che incrementando I di 5, in diciannove volte raggiunge il valore di 95 (il 20° incremento lo porterebbe a 100, che è maggiore di 99).
L'incremento non deve essere necessariamente positivo, ma se è negativo, il valore iniziale di I deve essere maggiore del valore finale. Per esempio, se l'incremento è −1 e si vuole assegnare ai 100 elementi di A(I) valori da 99 a 0, allora si dovrà riscrivere la riga 2Ø come segue:

```
10 DIM A(99)
20 FOR I=99 TO 0 STEP -1
30 A(I)=I
35 PRINT A(I);
40 NEXT I
80 GOTO 80
```

Eseguite questo programma per verificare l'incremento negativo.
In questo esempio il valore iniziale e finale di I, e la grandezza dell'incremento, sono trattati come interi. Tuttavia si possono rappresentare questi tre valori usando variabili o espressioni a virgola mobile. Le espressioni saranno calcolate in modo da produrre risultati a virgola mobile; il risultato sarà convertito in intero usando le regole di arrotondamento descritte precedentemente. Poiché gli arrotondamenti possono causare dei problemi vi consigliamo vivamente di usare valori iniziali, finali e di incremento interi e di non usare espressioni, dato che queste complicano inutilmente il programma. Se si deve calcolare uno di questi valori, è più semplice e veloce farlo in un'istruzione separata.

Quando il valore dell'incremento è 1 (come spesso è), non è necessario includere la definizione dell'incremento. In assenza di un valore specificato il BASIC del C-64 assume un incremento uguale a 1. Quindi l'istruzione a riga 2Ø potrebbe essere riscritta come segue:

```
10 DIM A(99)
15 REM IL VALORE DELL'INCREMENTO E' UNO
20 FOR I=0 TO 99
30 A(I)=I
35 PRINT A(I);
40 NEXT I
80 GOTO 80
```

Non è neppure necessario specificare la variabile dell'indice nell'istruzione NEXT, anche se in questo modo si rende più leggibile il programma.

Loop annidati

La struttura FOR-NEXT viene chiamata un *loop di programma*, dato che l'esecuzione delle istruzioni cicla da FOR a NEXT e ritorna a FOR. Questa struttura è molto comune; quasi tutti i programmi che scriverete includeranno uno o più loop di questo tipo, spesso annidati l'uno dentro l'altro. La sequenza d'istruzioni tra un FOR e NEXT può essere di qualsiasi lunghezza e può contenere decine o anche centinaia di istruzioni, all'interno delle quali si possono avere altri loop. Il seguente listato mostra un esempio di annidamento:

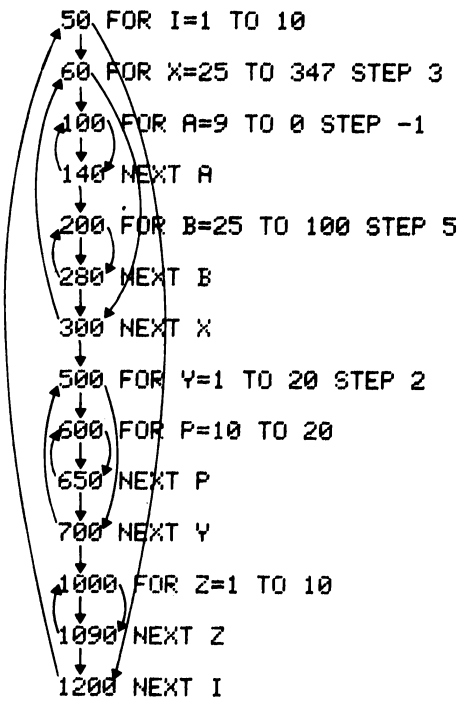
```
10 DIM A(99)
20 FOR I=0 TO 99
30 A(I)=I
40 REM MOSTRA TUTTI I VALORI DI A(I) FIN QUI ASSEGNATI
50 FOR J=0 TO I
60 PRINT A(J)
70 NEXT J
80 NEXT I
90 GOTO 90
```

Complesse strutture di loop capitano frequentemente, anche in programmi relativamente brevi. Ecco qui un esempio che mostra le istruzioni FOR e NEXT, ma non quelle intermedie.

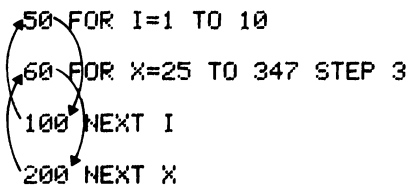
```
50 FOR I=1 TO 10
60 FOR X=25 TO 347 STEP 3
.
100 FOR A=9 TO 0 STEP -1
```

```
.
140 NEXT A
200 FOR B=25 TO 100 STEP 5
.
280 NEXT B
300 NEXT X
.
500 FOR V=1 TO 20 STEP 2
.
600 FOR P=10 TO 20
.
650 NEXT P
700 NEXT V
.
1000 FOR Z=1 TO 10
.
1090 NEXT Z
1200 NEXT I
```

Il loop più esterno usa l'indice I; esso contiene tre loop annidati che usano gli indici X, Y e Z. Il primo di essi contiene due altri loop che utilizzano gli indici A e B; il secondo contiene un loop che usa l'indice P; il terzo non ne contiene alcuno. Ogni loop annidato deve usare una variabile indice differente. Le sequenze d'esecuzione si possono illustrare come segue:



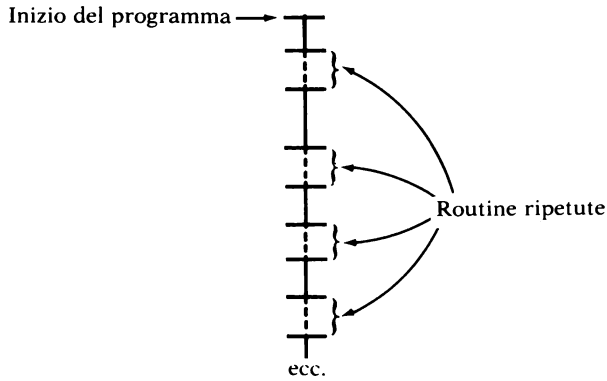
Le strutture dei loop sono facili da visualizzare ed usare ma si deve fare attenzione a non commettere un errore comune: chiudere un loop esterno prima di terminarne uno interno. Per esempio la seguente struttura non è permessa:



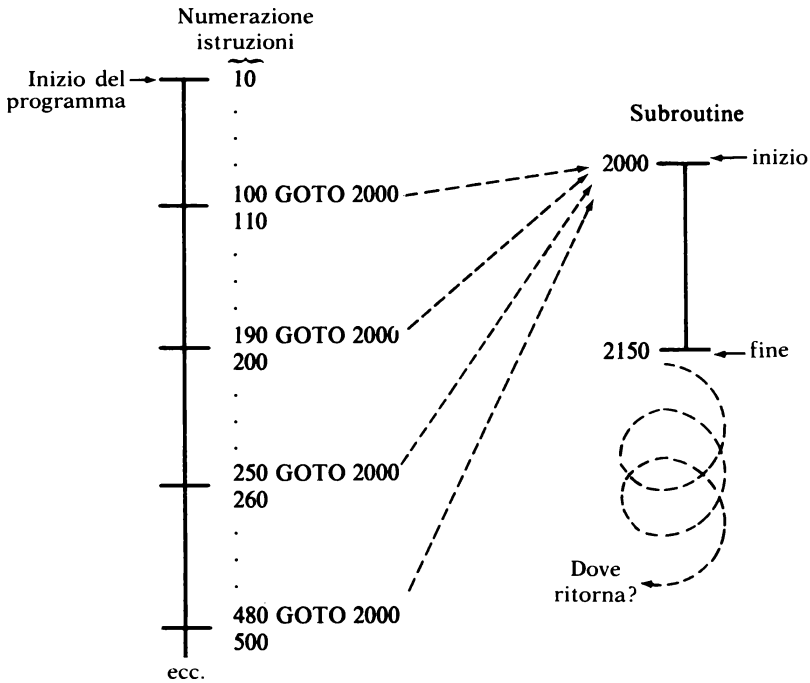
Se non si include la variabile dell'indice nell'istruzione NEXT, la logica del programma chiuderà automaticamente i loop in modo corretto, dato che esiste una sola possibile chiusura corretta ogni volta che si incontra un'istruzione NEXT. Ogni programma deve avere un ugual numero di istruzioni FOR e NEXT, poiché ogni loop deve cominciare con un FOR e terminare con un NEXT. Ad esempio, si supponga di avere due istruzioni FOR, ma un solo NEXT. Il secondo loop costituisce un loop interno e verrà eseguito correttamente mentre il loop esterno non ha un'istruzione NEXT per chiuderlo; il programma verrà eseguito perciò in modo scorretto. Analogamente anche troppi NEXT causerebbero un errore di sintassi.

LE SUBROUTINE

Quando si scrivono programmi che contengono un certo numero di righe, si trovano, fin dall'inizio, delle sequenze d'istruzioni o *routine* che vengono usate ripetutamente. Per esempio, supponete di avere un array A(I) che deve essere reinizializzato di frequente in diversi punti del programma. Potreste semplicemente ripetere le tre istruzioni che costituiscono il loop FOR NEXT descritto in precedenza; siccome impiega solo tre istruzioni tanto vale fare così.
Supponete invece di dover inizializzare questo array e poi eseguire una decina di istruzioni che lavorano in qualche modo sui dati del vettore. Se fosse necessario usare questo loop numerose volte nel programma, riscrivere dieci righe ogni volta sarebbe uno spreco di tempo, e ancor più grave sarebbe lo spreco di spazio di memoria. Ciò può essere illustrato come segue:

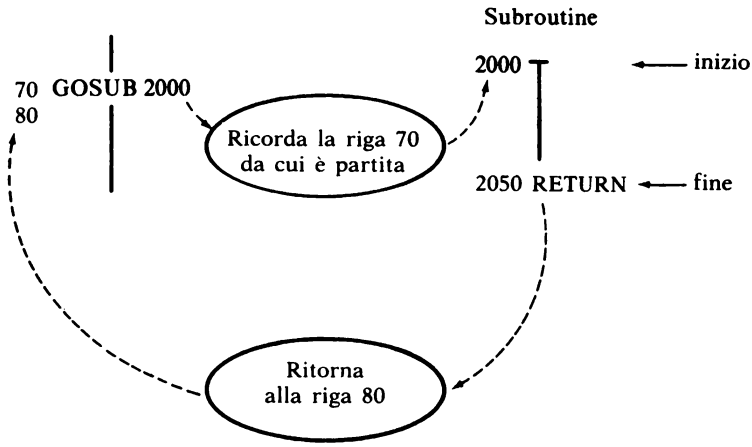


Per risolvere questo problema si potrebbero scrivere separatamente queste righe di programma e saltare ad esse ogni volta; questo gruppo di istruzioni si chiama *subroutine*. Ma si viene a creare un problema: saltare dal programma alla subroutine è semplice dato che essa comincia ad un numero di riga determinato mentre il punto di ritorno al programma principale varia di volta in volta.
Per risolvere il problema ci serve una nuova istruzione.



L'istruzione GOSUB

Questa istruzione salta nella stessa maniera di un GOTO ma in più ricorda anche il numero di riga a cui deve ritornare. Il suo funzionamento è illustrato nel seguente schema.



L'istruzione RETURN causa un salto indietro alla riga memorizzata dall'istruzione GOSUB. Il loop di tre righe che inizializza il vettore A(I), se trasformato in subroutine, si presenterebbe come segue:

```
10 REM PROGRAMMA PRINCIPALE
20 REM POTETE DIMENSIONARE LE VARIABILI DI SUBROUTINE
21 REM NEL PROGRAMMA PRINCIPALE
30 REM E' UNA BUONA ABITUDINE DIMENSIONARE
31 REM TUTTE LE VARIABILI ALL'INIZIO
50 REM DEL PROGRAMMA PRINCIPALE.
60 DIM A(99)
70 GOSUB 2000
80 REM VERIFICA DI RITORNO DALLA SUBROUTINE
90 PRINT "RITORNATO"
100 GOTO 100
2000 REM SUBROUTINE
2010 FOR I=0 TO 99
2020 A(I)=I
2030 PRINT A(I);
2040 NEXT I
2050 RETURN
```

Subroutine annidate

Le subroutine possono essere annidate, possono cioè, a loro volta, *chiamare* un'altra subroutine. Non è necessario fare nulla di particolare per poter usare subroutine annidate: basta semplicemente fare un salto ad un'altra subroutine usando l'istruzione GOSUB. Ogni subroutine termina con un'istruzione RETURN; il BASIC C-64 memorizzerà il numero di riga corretto per ogni RETURN. Il seguente programma illustra l'uso delle subroutine annidate:

```
10 REM PROGRAMMA PRINCIPALE
20 REM POTETE DIMENSIONARE VARIABILI DI SUBROUTINE
21 REM NEL PROGRAMMA PRINCIPALE
30 REM E' UNA BUONA ABITUDINE DIMENSIONARE
31 REM TUTTE LE VARIABILI ALL'INIZIO
50 REM DEL PROGRAMMA PRINCIPALE.
60 DIM A(99)
70 GOSUB 2000
80 REM VERIFICA DI RITORNO DALLA SUBROUTINE
90 PRINT "RITORNATO"
100 GOTO 100
2000 REM SUBROUTINE PRIMO LIVELLO
2010 FOR I=0 TO 99
2020 A(I)=I
2030 GOSUB 3000
2040 NEXT I
2050 RETURN
3000 REM SUBROUTINE ANNIDATA
3010 PRINT A(I)
3020 RETURN
```

Questo programma sposta solamente l'istruzione PRINT A(I) dalla prima subroutine e la pone in una subroutine annidata.

Istruzione GOSUB calcolata

Siccome la logica delle istruzioni GOTO e GOSUB è molto simile, non sarà una sorpresa scoprire che esiste un'istruzione GOSUB calcolata del tutto simile a quella di GOTO calcolata. Essa permette di saltare a subroutine diverse a seconda del valore di un indice.
Considerate la seguente istruzione:

```
90
100 ON A GOSUB 1000,500,5000,2300
110
```

Quando viene eseguita l'istruzione a riga 1ØØ, se A=1 viene chiamata la subroutine che inizia a riga 1ØØØ; se A=2 viene chiamata quella che comincia a riga 5ØØ; se A=3 si salta a quella con inizio a riga 5ØØØ e per A=4 a quella di riga 23ØØ. Qualora il valore di A non fosse nè 1, nè 2, 3 o 4 il programma continuerebbe semplicemente l'esecuzione progressiva con riga 11Ø. L'istruzione GOSUB calcolata memorizza il numero di riga successivo (in questo caso 11Ø). Non ha importanza quale subroutine viene chiamata: il RETURN della subroutine in questione ritornerà alla riga memorizzata. Si possono annidare le istruzioni GOSUB calcolate esattamente come le GOSUB normali.

IL COSTRUTTO IF-THEN

Gli operatori aritmetici e relazionali descritti prima in questo capitolo sono usati frequentemente insieme a IF-THEN. Questo dà ai programmi in BASIC la possibilità di prendere delle decisioni. Dopo un IF si può scrivere qualsiasi espressione: se l'espressione è vera, le istruzioni che seguono THEN vengono eseguite; se, invece, l'espressione è falsa le istruzioni dopo THEN non vengono eseguite. Esaminiamo questi tre semplici esempi di IF-THEN:

```
10 IF A=B+5 THEN PRINT MSG1
40 IF CC$<"M" THEN IN=0
50 IF Q<14 AND M<>M1 GOTO 66
```

L'istruzione a riga 1Ø farà sì che l'istruzione PRINT venga eseguita qualora il valore della variabile A sia uguale a B+5. L'istruzione a riga 4Ø assegna il valore 0 alla variabile IN se la stringa CC$ contiene una qualunque lettera dell'alfabeto tra A e L. L'istruzione a riga 5Ø esegue un salto a riga 66 se la variabile Q è minore di 14 e M è diversa da M1; altrimenti l'esecuzione del programma continuerà con la riga successiva (GOTO può sostituire THEN).
Qualora non comprendiate le espressioni che seguono gli IF, riguardatevi la spiegazione di questo argomento fatta all'inizio del capitolo.

ISTRUZIONI DI INPUT/OUTPUT

Ci sono una varietà di istruzioni BASIC che controllano il trasferimento di dati da e al computer. Collettivamente si chiamano istruzioni di input/output.
Le istruzioni di input/output più semplici controllano il caricamento di dati dalla tastiera e la presentazione sul video dei risultati. Istruzioni più

complesse controllano il trasferimento tra il computer e le periferiche come le unità a cassette, a dischetti e le stampanti. Queste ultime sono descritte nel capitolo 8. Iniziamo col trattare l'istruzione PRINT.

Istruzione PRINT

Si può usare la parola PRINT o un punto di domanda per creare una istruzione di stampa. Perché si usa PRINT invece di DISPLAY (che significa far apparire sul video) o di una abbreviazione della parola *display?* La risposta è che nei primi anni sessanta, quando veniva sviluppato il linguaggio BASIC, gli schermi video erano molto costosi e generalmente non disponibili con i sistemi di medio o basso costo. Il terminale dei computer aveva solo una tastiera e una stampante: le informazioni erano stampate (printed) invece di essere presentate su video; da qui l'uso della parola "print" per descrivere un'istruzione che visualizza i dati su di uno schermo.
L'istruzione PRINT fa apparire qualsiasi dato: il testo scritto deve essere rinchiuso tra virgolette. Per esempio, la seguente istruzione visualizza sullo schermo la parola "TESTO":

```
10 PRINT "TESTO"
```

o

```
10 ?"TESTO"
```

Per scrivere un numero, si mette il numero stesso o il nome di una variabile dopo PRINT. Quindi

```
10 A%=10
20 ?5,A%
```

presenta il numero 5 e poi il 10 sulla stessa riga. Si può far apparire un miscuglio di testo e numeri elencando le informazioni dopo PRINT. Usate le virgole per separare i singoli dati. La seguente istruzione PRINT presenta le parole "UNO", "DUE", "TRE", "QUATTRO" e "CINQUE", seguite dalla relativa cifra:

```
10 ?"UNO",1,"DUE",2,"TRE",3,"QUATTRO",4,"CINQUE",5
```

Separando le variabili con virgole, come sopra, il C-64 assegna automaticamente un campo di 11 caratteri ad ogni variabile richiamata sul video; in altre parole divide ogni riga a metà. Per convincervi, provate ad ese-

guire l'istruzione illustrata sopra in modo diretto; se volete una videata senza gli spazi vuoti, separate le variabili con punto e virgola, come segue:

```
10 PRINT "UNO";1;"DUE";2;"TRE";3;"QUATTRO";4;
"CINQUE";5
```

Caricate questa istruzione in modo diretto e fatela apparire sul video per vedere come funziona il punto e virgola.
Un'istruzione PRINT fa avanzare automaticamente il video alla riga successiva se non mettete una virgola o un punto e virgola dopo l'ultima variabile. Una virgola dopo l'ultima variabile farà continuare il video al successivo campo di 11 caratteri. Caricate il seguente programma:

```
10 PRINT "UNO";1;"DUE";2
20 PRINT "TRE";3
```

Ora aggiungete un punto e virgola alla fine dell'istruzione a riga 10 e di nuovo eseguite il programma scrivendo RUN. Sul video i dati appariranno ora sulla medesima riga.
Si è finora parlato di numeri inseriti direttamente nell'istruzione PRINT. È possibile, qualora lo si desideri, far apparire invece il contenuto di variabili. Provate ad inserire e a eseguire il seguente programma, che usa la variabile A%(I) per scrivere dei numeri:

```
10 FOR I=1 TO 5
20 A%(I)=I
30 NEXT
40 PRINT "UNO";A%(1);"DUE";A%(2);"TRE";A%(3);
   "QUATTRO";A%(4)
50 GOTO 50
```

Si potrebbero inserire le parole in un array a stringa e spostare l'istruzione PRINT nel loop FOR-NEXT cambiando il programma come segue:

```
10 DATA "UNO","DUE","TRE","QUATTRO","CINQUE"
20 FOR I=1 TO 5
30 A%(I)=I
40 READ N$(I)
50 PRINT N$(I);A%(I);
60 NEXT
70 GOTO 70
```

Il programma riportato sopra non è ben scritto. A%(I) può essere eliminato e non è necessario che N$ sia un array. Siete capaci di riscrivere il programma usando N$ ed eliminando A%(I)?

Stringhe contenenti virgolette

Sebbene la maggior parte dei programmi in BASIC non abbia bisogno di usare le virgolette, ve ne sono alcuni che devono poterlo fare, ad esempio quelli che trattano testi. Siccome le virgolette indicano l'inizio e la fine delle stringhe, non è possibile inserirle nel mezzo di una stringa, se non usando la funzione CHR$.
CHR$ si comporta come un array di tutti i possibili caratteri. Si fornisce un indice e CHR$ genera il carattere che corrisponde a quel numero. I valori di questo indice e i caratteri a cui corrisponde sono elencati nell'appendice E; il valore per le virgolette è 34. Usando CHR$ si possono scrivere le virgolette con un'istruzione del seguente tipo

```
100 PRINTCHR$(34);"QUESTA FRASE APPARE
TRA VIRGOLETTE";CHR$(34)
```

Se si esegue un PRINT di una stringa che contiene caratteri di controllo come CRSR UP o HOME, si deve fare un ulteriore passaggio. Nel secondo capitolo è stato descritto il *modo tra virgolette*; in questo modo di utilizzo i tasti di controllo del cursore vengono tradotti in caratteri speciali che possono essere inseriti in stringhe. Ciò permette al vostro programma di eseguire queste funzioni durante l'esecuzione.
Il modo tra virgolette funziona anche in output. Per permettere il listing di programmi che contengono questi caratteri di controllo, quella parte di BASIC preposta alla presentazione delle informazioni "cerca" le virgolette: quando ne trova si mette in modo tra virgolette e presenta i caratteri di controllo nella forma inversa a quella che si vede quando si inseriscono da tastiera. Attenzione però che questa caratteristica può causare dei brutti scherzi in una videata preparata minuziosamente.
Si può uscire dal modo tra virgolette mentre si sta stampando proprio come si fa quando si inserisce un programma dalla tastiera: battendo le virgolette o un RETURN. Visto che i programmi che stampano virgolette normalmente lo fanno a coppie, ci saranno raramente dei problemi. Se il vostro programma deve stamparne una sola, allora si potrà usare CHR$ per cancellare la prima e poi stamparne un'altra per uscire dal modo tra virgolette.

```
100 PRINTCHR$(34);CHR$(20);CHR$(34);"■FRASE
INVERTITA■"CHR$(34)
```

CHR$(20) cancella le prime virgolette. Solamente le seconde appariranno sullo schermo.

Funzioni di formattazione dei PRINT

Si usa l'espressione *formattazione* per descrivere il procedimento di organizzazione delle informazioni sul video (o sulla stampante) per renderle più facilmente comprensibili o più piacevoli da guardare. Se ci fosse solo l'istruzione PRINT e nient'altro, la formattazione diventerebbe un'operazione complessa e noiosa. Per esempio, supponete di dover stampare un titolo nel centro della prima riga in alto. Si potrebbero stampare degli spazi fino a raggiungere la posizione corretta del primo carattere ma ciò non solo provocherebbe una perdita di tempo e probabilmente causerebbe errori, ma sprecherebbe anche memoria, dato che ogni codice di spaziatura comporta un'istruzione al computer. Fortunatamente il BASIC del C-64 è provvisto di tre supporti alla formattazione dei PRINT: SPC, TAB e POS.

Funzione SPC

SPC è una funzione di spaziatura e si include nelle istruzioni PRINT come uno dei termini. Come argomento della funzione SPC si inserisce, tra parentesi, il numero di spazi che si desidera lasciare liberi; per esempio, si potrebbe scrivere un titolo che comincia dalla sinistra dello schermo così facendo:

```
10 PRINT"TITOLO"
```

Per centrare il titolo sul video bisogna prima muoversi di otto spazi, così:

```
10 PRINT SPC(8);"TITOLO"
```

Notate il punto e virgola dopo la funzione SPC. Una virgola dopo la funzione SPC farebbe iniziare la scrittura alla prima posizione del campo di 11 caratteri successivo al numero di spazi specificato da SPC.
Quando si include questa funzione in un'istruzione PRINT si fa apparire il carattere seguente spostato del numero di caratteri specificato da SPC; non va cambiata alcuna regola della sintassi dell'istruzione PRINT.

Funzione TAB

La funzione TAB è simile a quella di tabulazione di una normale macchina da scrivere. Supponendo di voler presentare delle informazioni in colonna, è necessario innanzitutto calcolare la posizione di ogni colonna.

Numero di colonna	
0	12
ROSSI G.	496340
GATTI A.	3742835
MILANI G.	230977
ecc.	ecc.

Nell'esempio sopra esposto, le colonne cominciano alla posizione 0 e 12. Invece di calcolare le spaziature di riga in riga, dopo ogni elemento di ogni colonna si mette una funzione TAB nell'istruzione PRINT. Si consideri una riga della videata: contando le posizioni dei caratteri si può costruire l'istruzione senza usare il TAB, così:

```
10 PRINT "ROSSI G.      496340"
```

Invece di inserire spazi direttamente, si potrebbe usare la funzione SPC, accorciando così l'istruzione:

```
10 PRINT "ROSSI G.";SPC(4);"496340"
```

Ma l'uso del TAB è più facile perché incolonna i dati senza bisogno di contare gli spazi. Notate che i dati della seconda colonna sono numeri che sono stati immessi come testo. Provate a riscrivere l'istruzione PRINT in modo da scriverli come "numeri"; essi non si allineano più come quando erano stati caricati come caratteri (nel capitolo 5 verranno discusse le eccezioni che si riscontrano con la formattazione del video). In questo caso i numeri lasciano uno spazio per il segno negativo e non scrivono gli zeri dopo il punto decimale: questo è il motivo del differente incolonnamento.

Funzione POS

POS indica la posizione attuale del cursore; viene indicata sotto forma di un numero equivalente al numero della colonna in cui si trova il cursore. Scrivete la funzione come POS(0); la seguente istruzione spiega la sintassi della funzione POS:

```
10 PRINT"LA POSIZIONE DEL CURSORE E' ";POS(0)
```

Fate eseguire quest'istruzione in modo diretto. Sul video si vedrà apparire:

```
?"LA POSIZIONE DEL CURSORE E' ";POS(0)
LA POSIZIONE DEL CURSORE E' 28
```

Il cursore si trovava alla 28ª posizione dopo aver scritto LA POSIZIONE DEL CURSORE È. Se si aggiungessero degli spazi dopo È prima di chiudere le virgolette, il valore di POS aumenterebbe di conseguenza.

Istruzione INPUT

Quando viene eseguita un'istruzione INPUT il computer attende un'immissione dalla tastiera; non succede nulla finché questa immissione (*input*) non è stata ricevuta. Un'istruzione di questo tipo comincia con la parola INPUT, che deve essere seguita da una lista di variabili. I dati immessi verranno assegnati alle variabili nominate il cui tipo determina la forma dei dati che dovranno essere immessi. Un nome di variabile a stringa (che termina con $) accetta solo un input di testo con qualsiasi numero di caratteri. A dimostrazione di ciò, caricate ed eseguite il seguente programma:

```
10 INPUT A$
20 PRINT A$
30 GOTO 10
```

Quando esegue un'istruzione INPUT, il computer scrive un punto interrogativo e poi attende i vostri dati. Il programma illustrato sopra fa apparire sul video qualunque testo mentre esso viene caricato. Il testo è presentato una seconda volta dall'istruzione PRINT sulla riga seguente. La prima presentazione avviene quando viene eseguita l'istruzione a riga 1Ø; la seconda è la conseguenza dell'istruzione PRINT a riga 2Ø.
Si possono inserire dati numerici interi o a virgola mobile a seconda del tipo di variabile scritto dopo INPUT. Bisogna separare i singoli dati con virgole che in un'istruzione INPUT non sono usate come punteggiatura. Il seguente esempio carica una parola di testo, un numero a virgola mobile ed uno intero, poi presenta sul video i tre dati.

```
10 INPUT A$,A,A%
20 PRINT A$,A,A%
30 GOTO 10
```

Si dovrà caricare del testo seguito da una virgola, poi un numero a virgola mobile seguito da una virgola, poi un numero intero seguito da RETURN. Qualsiasi variazione di questa sequenza d'immissione causerà un errore; a seguito di un errore il computer risponde con due punti interrogativi e si dovrà allora reinserire il dato nella forma giusta. Se ancora il computer risponde con REDO FROM START (ricomincia da capo), caricate i dati di nuovo. Ora riscrivete l'istruzione PRINT in modo che A$, A e A% siano in un ordine differente da quello dell'istruzione INPUT e rieseguite il programma.
Come già detto in precedenza, qualunque intero può essere rappresentato usando un numero a virgola mobile, per cui si può caricare un valore intero per una variabile a virgola mobile mentre non è possibile fare il contrario. Non è accettabile caricare del testo in variabili intere o a virgola mobile, ma è possibile caricare un numero in una variabile stringa; il numero verrebbe inteso come sequenza di caratteri piuttosto che come valore numerico. Provate queste varie combinazioni per assicurarvi di aver capito le opzioni di caricamento dei dati.
L'istruzione INPUT è molto schizzinosa; la sua sintassi è troppo complessa per un normale operatore: un utente che non sa nulla delle tecniche di programmazione, incontrando i messaggi d'errore che vengono trasmessi se una virgola è fuori posto, potrebbe anche abbandonarsi alla disperazione. È quindi probabile che passerete molto tempo a scrivere programmi per il caricamento dei dati che siano "a prova di stupido"; questi programmi devono essere studiati per controllare ogni tipo immaginabile d'errore che un operatore potrebbe commettere mentre carica dei dati. Il capitolo 4 descrive dettagliatamente la programmazione degli input. Un semplice stratagemma che vale la pena di ricordare è la possibilità dell'istruzione INPUT di funzionare come un PRINT e di visualizzare i dati: può far precedere ogni richiesta di dati da un breve messaggio che indichi all'operatore cosa fare. Il messaggio appare tra virgolette nell'istruzione INPUT davanti alla variabile da caricare e deve essere separato da quest'ultimo da un punto e virgola. Eccone un esempio:

```
10 INPUT "INSERISCI IL NUMERO 1";N
20 IF N<>1 THEN GOTO 50
30 PRINT "OK"
40 GOTO 40
50 PRINT "NO, FALSO,"
60 GOTO 10
```

Questo programma scrive un messaggio, poi aspetta un singolo dato.
La caratteristica dell'istruzione INPUT ha, però, uno svantaggio: se la stringa del suggerimento è troppo lunga, il BASIC cerca di leggere il sug-

gerimento insieme al dato caricato dalla tastiera: questo succederà solamente nel caso in cui il testo di suggerimento si estenda oltre la fine della riga sul video. Per evitare questo problema, assicuratevi che i suggerimenti non superino i 40 caratteri di lunghezza. L'inconveniente si può verificare anche se l'istruzione INPUT segue direttamente un PRINT che termina con un punto e virgola. Dato che il suggerimento comincerebbe nel mezzo della riga, deve essere abbastanza corto da non "straripare" sulla riga successiva del video.
Qualora inavvertitamente si scrivesse un suggerimento troppo lungo, ci si potrebbe trovare intrappolati. Il BASIC continuerebbe a darvi un REDO (rifare) per poi riemettere di nuovo il suggerimento. Per uscire da questa trappola, usate il tasto DEL per cancellare il suggerimento, poi caricate il dato come richiesto. L'unica altra maniera è di premere i tasti RUN/STOP e RESTORE simultaneamente, per arrestare il programma.

INPUT con risposta prestabilita

Dopo aver stampato il suggerimento, INPUT stampa un punto interrogativo e uno spazio. Qualsiasi cosa alla destra di quello spazio viene considerata come se fosse caricata dalla tastiera. Aggiungendo spazi al suggerimento si può "prefissare" la risposta in maniera tale che l'utente debba premere soltanto RETURN. Per poter usufruire di questa caratteristica, aggiungete due spazi alla stringa di suggerimento, seguita poi dalla risposta da caricare. Poi usate CRSR LEFT per spostare il cursore dietro al primo spazio aggiunto. Quando INPUT scrive il suo punto di domanda e lo spazio, questi rimpiazzeranno i due spazi aggiunti prima, posizionando così il cursore sulla risposta. Se l'utente preme semplicemente RETURN, l'INPUT leggerà, ovvero caricherà, ciò che si era stabilito nell'INPUT. Si può anche prestabilire la risposta assegnando in anticipo un valore alla variabile che verrà caricata. Nel caso che l'utente risponda solo con un RETURN, il valore già stabilito della variabile non verrà cambiato. Notate però che INPUT è un'istruzione del tipo "o tutto o niente"; perciò se desiderate caricare variabili multiple, una volta assegnato il valore della prima variabile, dovranno essere caricati anche i valori di tutte le altre.

Istruzione GET

L'istruzione GET acquisisce un singolo carattere. Non è necessario alcun ritorno a capo del cursore (RETURN). Il singolo input può essere qualunque carattere che il C-64 riconosca, oppure una cifra da 0 a 9. L'input sarà interpretato come un carattere se il nome della variabile che segue GET è a stringa. Caricate il seguente programma e fatelo girare:

```
10 GET A$
20 PRINT A$
30 GOTO 10
```

Quando questo programma viene eseguito, tutto ciò che è sullo schermo sparirà velocemente verso l'alto così come il carattere ogni volta che si preme un tasto. Ciò perchè GET non attende il carico di un carattere, ma presume che quest'ultimo ci sia già. Si può far attendere il GET con un'istruzione che controlli che venga scritto un carattere specifico nel seguente modo:

```
10 GET A$
20 IF A$<>"X" THEN GOTO 10
30 PRINT A$
40 GOTO 10
```

Questo programma aspetta che la lettera X venga caricata. Nessun altro carattere lo potrebbe soddisfare. Un GET può essere anche programmato per attendere una qualunque battuta sulla tastiera; ciò può essere realizzato poiché l'istruzione GET assegna ad una variabile a stringa un codice di carattere nullo fino a che qualcosa non venga immesso dalla tastiera. Il codice nullo, 00, non può essere caricato da tastiera, ma può essere specificato da programma usando due virgolette contigue (" "). Ecco qui la sequenza di programma necessaria.

```
10 GET A$
20 IF A$="" THEN GOTO 10
30 PRINT A$
40 GOTO 10
```

Se l'istruzione GET specifica una variabile intera o a virgola mobile, l'input è interpretato come una cifra numerica. La variabile intera o a virgola mobile che appare in un'istruzione GET ha valore 0 fino a che il dato di input viene ricevuto. Ma siccome si può caricare 0 dalla tastiera, il programma non riesce a distinguere se lo 0 rappresenta un valore lecito o significa l'assenza di input. Ciò può causare dei problemi ai programmi che contengono controlli di input, come esemplificato sopra. Per questo motivo, le istruzioni GET normalmente ricevono caratteri a stringa. Spesso i programmi utilizzano l'istruzione GET per dialogare con l'operatore; per esempio, un programma può attendere che l'operatore indichi la propria presenza caricando una lettera specifica, come la "Y" per "YES".

```
10 PRINT "CI SEI? BATTI Y SE CI SEI"
```

```
20 GET A$
30 IF A$<>"Y" THEN GOTO 20
40 PRINT "OK, ANDIAMO AVANTI"
```

Notate che questa sequenza non presenta mai sul video il carattere caricato dalla tastiera. Provate a riscrivere il programma in modo da far apparire sul video ogni carattere caricato.

ISTRUZIONI PEEK E POKE

PEEK e POKE sono due istruzioni che si incontreranno spesso nei capitoli successivi. Il C-64 ha un totale di 65536 locazioni di memoria, ognuna delle quali può contenere un numero tra 0 e 255 (questo strano limite superiore è uguale a $2^8 - 1$). Tutti i programmi e i dati sono convertiti in numeri e memorizzati in questo modo.
L'istruzione PEEK permette di leggere il numero memorizzato in ognuna delle locazioni di memoria del C-64. Si consideri la seguente istruzione PEEK:

```
10 A%=PEEK(200)
```

Essa assegna alla variabile A% il contenuto della locazione di memoria 200. L'argomento di PEEK può essere un numero, come nell'esempio, una variabile o un'intera espressione, ma comunque sia deve risultare come l'indirizzo di una locazione di memoria.
L'istruzione POKE trasferisce, o scrive, dei dati ad una locazione di memoria. Per esempio l'istruzione

```
20 POKE 8000,A%
```

trasferisce e memorizza nella cella 8000 il contenuto della variabile A%.Ogni argomento di POKE può essere un numero, una variabile, come nel caso sopra, o un'espressione con un valore tra 0 e 255. Una variabile a virgola mobile è convertita automaticamente in intero.
Si può fare un PEEK in una locazione di memoria a lettura e scrittura (read/write) o a sola lettura (read only) ma si può fare un POKE solo ad una locazione di memoria a lettura e scrittura. La memoria a sola lettura può essere, come indica il nome stesso, solamente letta, ma non si può trasferire in essa alcun dato.

ISTRUZIONI END E STOP

Le istruzioni END e STOP arrestano l'esecuzione del programma. Si può poi continuare l'esecuzione scrivendo CONT. Non è obbligatorio includere istruzioni END o STOP nei programmi, ma esse li rendono più facili da usare. In molti degli esempi di programma forniti in questo capitolo si è usato un GOTO che manda il programma a se stesso per fermare l'esecuzione. Per esempio l'istruzione

```
50 GOTO 50
```

verrà eseguita all'infinito, dato che l'istruzione GOTO continua a trasferire il controllo dell'esecuzione a se stessa: si potrebbe sostituire questa istruzione con uno STOP. Quando viene eseguita un'istruzione STOP appare il seguente messaggio:

```
BREAK IN XXXX
```

XXXX è il numero della riga dello STOP. Se esiste più di una istruzione STOP nel programma, si può identificare quale di esse ha causato l'arresto del programma grazie al numero indicato.

3.4. Le funzioni

Elementi importanti del BASIC C-64 sono le funzioni che eseguono operazioni matematiche su variabili numeriche e operazioni su variabili a stringa. Forse la maniera più semplice di illustrare una funzione è vederne un esempio in un'istruzione di assegnamento:

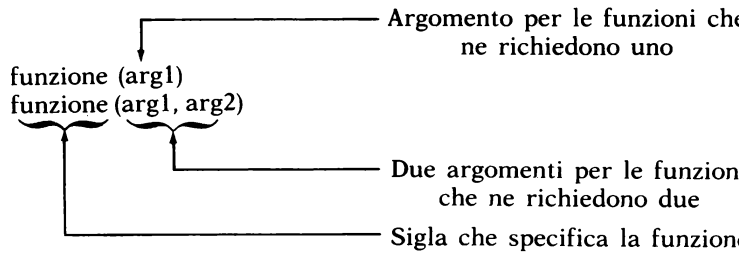
```
10 A=SQR(B)
```

La variabile A è fissata uguale alla radice quadrata della variabile B. SQR specifica la funzione di radice quadrata. Ecco qui una funzione a stringa:

```
20 C$=LEFT$(D$,2)
```

In questo esempio la stringa C$ è posta uguale ai primi due caratteri della variabile a stringa D$. In un'istruzione BASIC, le funzioni possono sostituire variabili o costanti ovunque, ma non a sinistra dell'uguale. In altre parole si può dire A=SQR(B), ma non SQR(B)=A. Noi abbiamo già usato quattro funzioni: SPC, TAB, e POS sono funzioni di sistema usate

con l'istruzione PRINT per formattare i dati; PEEK è la quarta.
Presentiamo qui un breve e parziale elenco delle funzioni BASIC C-64; una descrizione completa di tutte le funzioni si trova nelle appendici G e H.
Si specifica una funzione usando un'abbreviazione, come SQR per radice quadrata (square root), seguita da uno o più argomenti tra parentesi. Nel caso di A=SQR(B), SQR richiede un solo argomento, che in questo caso è la variabile B. Per C$=LEFT$(D$,2), LEFT$ specifica la funzione; i due argomenti D$ e 2 sono tra parentesi. In generale una funzione avrà una delle due forme seguenti:



Una funzione in un'istruzione BASIC è calcolata prima degli altri operatori. Ognuna viene ridotta ad un singolo valore numerico o di stringa prima che sia calcolata qualunque altra parte dell'istruzione; per esempio, nella seguente istruzione, le funzioni SQR e SIN sono calcolate per prime:

```
10 B=24.7*(SQR(C)+5)-SIN(0.2+D)
```

Si supponga SQR(C)=6.72 e che SIN(0.2+D)=0.625. L'istruzione a riga 1Ø verrà prima ridotta a

```
10 B=24.7*(6.72+5)-0.625
```

e poi calcolata

FUNZIONI ARITMETICHE

Segue una lista di funzioni aritmetiche che possono essere usate col BASIC del C-64.

INT Converte un argomento a virgola mobile nell'intero equivalente, per troncamento.

SGN	Dà il segno di un argomento: +1 per un argomento positivo, −1 per uno negativo e 0 se l'argomento è 0.
ABS	Dà il valore assoluto di un argomento. Se esso è positivo non cambia; se è negativo viene convertito nell'equivalente positivo.
SQR	Calcola la radice quadrata dell'argomento.
EXP	Eleva "e" alla potenza rappresentata dall'argomento (e^{arg}).
LOG	Dà il logaritmo naturale dell'argomento.
RND	Genera un numero a caso. Vi sono alcune regole a proposito dell'uso di RND; esse sono riportate nel capitolo 5.
SIN	Dà il seno trigonometrico dell'argomento in radianti.
COS	Dà il coseno trigonometrico dell'argomento in radianti.
TAN	Dà la tangente trigonometrica dell'argomento in radianti.
ATN	Dà l'arcotangente trigonometrica dell'argomento in radianti.

Il seguente esempio usa una funzione aritmetica:

```
10 A=2.743
20 B=INT(A)+7
30 PRINT B
40 STOP
```

Quando si esegue questo programma, il risultato ottenuto è 9, dato che il valore intero di A è 2. Per esercizio, cambiate l'istruzione a riga 1Ø in un'istruzione INPUT. Cambiate riga 4Ø con GOTO 1Ø. Ora potete caricare una varietà di valori per A ed osservare la funzione al lavoro. Usate questo programma per sperimentare le varie funzioni.
Ecco un esempio più complesso che usa anch'esso funzioni aritmetiche.

```
10 INPUT A,B
20 IF LOG(A)<0 THEN A=1/A
30 PRINT SQR(A)*EXP(B)
40 GOTO 10
```

Se avete dimestichezza con i logaritmi, per esercizio cambiate l'istruzione a riga 2Ø, rimpiazzando la funzione LOG con funzioni aritmetiche che svolgono la stessa operazione. L'argomento di una funzione può essere un'espressione la quale può contenere delle funzioni. Per esempio, cambiate la riga 3Ø con la seguente istruzione ed eseguite il programma:

```
30 PRINT SQR(A*EXP(B)+3)
```

Ora provate ad usare funzioni aritmetiche creando istruzioni PRINT che facciano un uso più complesso delle funzioni aritmetiche.

FUNZIONI DI STRINGA

Le funzioni di stringa permettono di manipolare i dati nelle stringhe in diversi modi. Mentre forse non vi sarà necessario far uso di tutte le funzioni aritmetiche, dovrete fare lo sforzo di conoscere tutte le funzioni di stringa perché sono estremamente utili in ogni programma. Ecco una lista di quelle utilizzabili con il BASIC C-64.

STR$	Trasforma un numero nell'equivalente stringa di caratteri di testo.
VAL	Trasforma una stringa di caratteri di testo nel loro numero equivalente (se tale trasformazione è possibile).
CHR$	Trasforma un codice binario a 8 bit nell'equivalente carattere ASCII.
ASC	Trasforma un carattere ASCII nell'equivalente binario a 8 bit.
LEN	Dà il numero di caratteri contenuto in una stringa di testo.
LEFT$	Estrae la parte sinistra di una stringa di testo. Gli argomenti della funzione identificano la stringa e il numero di caratteri da estrarre partendo da sinistra.
RIGHT$	Estrae la parte destra di una stringa di testo. Gli argomenti sono come per LEFT$.
MID$	Estrae la parte centrale di una stringa di testo. Gli argomenti della funzione identificano la stringa e la sezione centrale richiesta.

Le funzioni di stringa permettono di determinare la lunghezza di una stringa, estrarre porzioni di essa e convertire caratteri numerici, ASCII e di stringa: queste funzioni usano uno, due o tre argomenti. Eccone alcuni esempi.

```
STR$(14)

LEN("ABC")

LEN(A$+B$)

LEFT$(ST$,1)
```

FUNZIONI DI SISTEMA

Per completezza, elenchiamo di seguito le funzioni di sistema: esse effettuano operazioni di cui è improbabile che abbiate bisogno prima di diventare esperti programmatori. L'unica funzione che potreste usare abbastanza presto è quella che legge l'ora. Qualora si stampassero molte variazioni di un programma (o di qualunque altro materiale) nello stesso giorno, potrebbe essere utile stampare l'ora all'inizio del programma. In

questo modo sarebbe poi possibile conoscere la sequenza cronologica delle diverse versioni.

PEEK	Legge il contenuto di un byte di memoria.
TI$,TI	Legge l'ora da un clock di sistema.
FRE	Dà lo spazio di memoria disponibile — il numero di byte di RAM ancora inutilizzati.
SYS	Trasferisce il controllo a un sottosistema.
USR	Trasferisce il controllo a una routine in linguaggio *Assembler*.

FUNZIONI DEFINIBILI DALL'UTENTE

Oltre alle molte funzioni standard del BASIC C-64, è possibile definire proprie funzioni aritmetiche, purché non siano molto complicate; non sono permesse funzioni di stringa definibili dall'utente. Ecco l'esempio di un piccolo programma che utilizza una istruzione DEF FN (definizione di funzione):

```
10 DEFFNP(X)=100*X
20 INPUT A
30 PRINT A,FNP(A)
40 GOTO 20
```

Dopo la DEF FN si può avere qualsiasi nome. In questo caso è stato caricato P, quindi il nominativo della funzione diventa FNP. Se il nome fosse AB, la funzione si chiamerebbe FNAB.
Nell'istruzione DEF FN una singola variabile, tra parentesi, deve seguire FN. Questo è l'unico nome di variabile che può essere messo a destra dell'uguale ed è usato solamente all'interno dell'istruzione DEF FN; può essere usato nel resto del programma senza alcun effetto sulla funzione.

Capitolo

Programmazione avanzata

4

Nel capitolo precedente abbiamo esaminato la maggior parte del BASIC C-64, ma c'è ancora molto da scoprire sulla programmazione!
Questo capitolo ed i successivi sono infatti dedicati alle tecniche di programmazione e ai consigli per ottenere il massimo rendimento dal vostro C-64.
Trattandosi di programmazione avanzata, gli esempi e le spiegazioni dei programmi saranno più lunghi ed è consigliabile caricare ed eseguire ogni esempio in modo da capire meglio i concetti in discussione.
Molti degli esempi in questo capitolo sono stati concepiti per venir usati all'interno di programmi da voi scritti; alcuni sono presentati come subroutine, altri possono essere trasformati in subroutine con minimi cambiamenti.

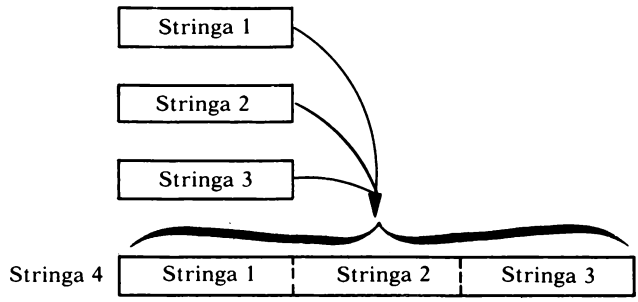
4.1. Programmazione con uso di stringhe

Una stringa può fare molto di più che non semplicemente contenere quei dati che non possono essere espressi in forma numerica. Le operazioni e le funzioni stringa danno la possibilità di cambiare e manipolare i dati.

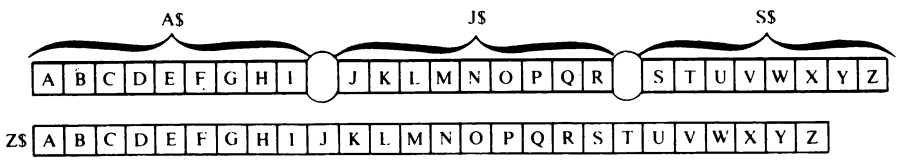
CONCATENAMENTO DI STRINGHE

Le stringhe possono contenere caratteri alfanumerici o numerici o una combinazione dei due. Nell'uso pratico può essere utile unire stringhe

corte una in fila all'altra come una catena per formarne una più lunga. Questo procedimento, come ricorderete, si chiama concatenamento.



Si supponga, per esempio, di voler creare una stringa lunga, Z$, contenente l'alfabeto dalla A alla Z. Per far questo si collega l'ultimo carattere di A$, come si vede sotto, al primo carattere di J$ e l'ultimo carattere di J$ al primo di S$.



Quando un segno + è posto tra due espressioni numeriche, somma i valori delle due espressioni; quando dalle due parti del segno compaiono due variabili a stringa il segno + le concatena. Ciò non vale per il segno meno: le stringhe non possono essere "deconcatenate" nello stesso modo in cui sono concatenate; non possono venir "sottratte" nel modo in cui sono "addizionate". Per esempio, per creare la stringa X$, composta dalle stringhe J$ e S$ viste prima, sarebbe errato scrivere

```
X$=Z$-A$ <---------- Errato
```

Provatelo voi stessi. Caricate i valori di A$, J$, S$ e X$=Z$−A$ nel C-64 come mostrato. Il computer risponderà con il messaggio ?TYPE MISMATCH ERROR IN LINE 5Ø.

```
10 A$="ABCDEFGHI"
20 J$="JKLMNOPQR"
30 S$="STUVWXYZ"
```

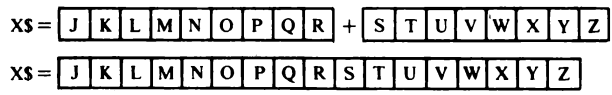
```
40 Z$=A$+J$+S$
50 X$=Z$-A$  <-------- Tentativo errato di ottenere J$+S$ da Z$
60 PRINT X$

RUN

 ?TYPE MISMATCH
  ERROR IN 50
```

Il modo corretto per estrarre una porzione di una stringa più lunga è di usare le funzioni di stringa: LEFT$, MID$ e RIGHT$. Nell'esempio in questione, le lettere da J a Z possono essere estratte come segue:

```
50 X$=RIGHT$(Z$,17)
```

Il numero 17 indica che il 17° carattere da destra (RIGHT$) diventa il primo carattere della stringa che contiene tutti i rimanenti caratteri alla propria destra. Questa stringa può essere costruita anche concatenando J$ e S$.

```
50 X$=J$+S$
```



STRINGHE NUMERICHE

Una stringa numerica è una stringa il cui contenuto può essere valutato come un numero. Le stringhe numeriche possono essere create in due modi diversi, ognuno dei quali dà risultati leggermente differenti. Quando delle variabili numeriche vengono assegnate a stringhe usando la funzione STR$, il valore del segno che precede il numero (spazio vuoto se positivo, – se negativo) viene trasferito insieme al numero. Ciò è dimostrato nel seguente breve programma:

```
10 AB=12345
20 PI= -1*3.14159265
```

```
30 T$=STR$(AB)
40 N$=STR$(PI)
50 PRINT "AB=";AB
60 PRINT "T$=";T$
70 PRINT "N$=";N$

RUN
AB= 12345
T$= 12345
N$=-3.14159265
```

Spazio per il segno

Se il numero è caricato tra virgolette oppure è caricato come stringa da una istruzione INPUT o READ, la stringa numerica viene trattata come una qualsiasi altra stringa alfabetica o grafica: non viene lasciato alcuno spazio vuoto per il segno positivo davanti al numero. Questo è evidenziato nel seguente programma:

```
10 AB=12345
20 T$="12345"
30 READ R$
40 DATA 12345
50 PRINT "AB=";AB
60 PRINT "T$=";T$
70 PRINT "R$=";R$

RUN
AB= 12345
T$=12345
R$=12345
```

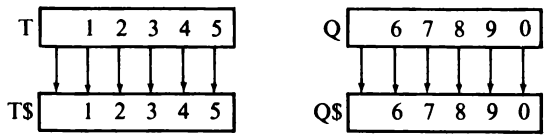
Spazio inserito
Nessuno spazio inserito
Nessuno spazio inserito

Concatenate le due stringhe numeriche T$ e Q$ per fare una nuova stringa W$ in modo che la stringa W$ contenga le dieci cifre 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. Ecco una possibilità:

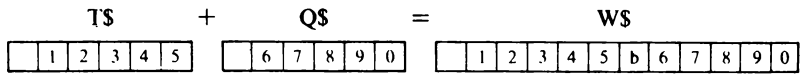
```
10 T=12345
20 Q=67890
30 T$=STR$(T)
40 Q$=STR$(Q)
50 W$=T$+Q$
60 PRINT "W$=";W$

RUN
W$=  12345 67890
```

Genera la nuova stringa W$

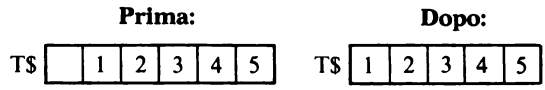
Come mai ci sono spazi prima dell'1 e del 6? Le stringhe T$ e Q$ erano in origine variabili numeriche positive T e Q. Quando T e Q sono state trasformate da numeri in stringhe, insieme ad essi sono stati trasferiti gli spazi vuoti del segno positivo.



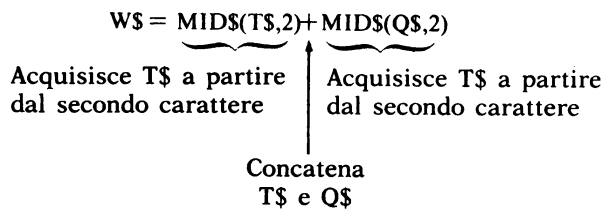
Quindi quando vengono concatenate T$ e Q$, la nuova stringa W$ contiene uno spazio nella prima posizione ed uno nella settima, prima della prima cifra di Q$.



Per eliminare gli spazi, ritornate alle stringhe separate T$ e Q$. Guardate di nuovo il loro contenuto: gli unici valori che si vogliono trasferire in W$ sono i numeri alla destra del segno sia in T$ che in Q$. Con i comandi LEFT$, MID$ e RIGHT$, si può selezionare qualsiasi carattere o gruppo di caratteri da una data stringa; nel nostro caso occorre T$=MID$(T$,2).



Dato che la prima cifra necessaria si trova nella seconda posizione della stringa, al C-64 viene ordinato di usare solamente i valori a cominciare dalla seconda posizione. Si possono concatenare T$ e Q$ e perdere gli spazi iniziali, tutto in un'unica istruzione.



Ecco come appare il programma-esempio usato in precedenza, modificato per eliminare le cifre del segno:

```
10 T=12345
```

T = 12345

```
20 Q=67890
```

Q = 67890

```
30 T$=STR$(T)
```

T$ = | | 1 | 2 | 3 | 4 | 5 |

```
40 Q$=STR$(Q)
```

Q$ = | | 6 | 7 | 8 | 9 | 0 |

```
50 W$=M$(T$,2)+MID$(Q$,2)
```

W$ = T$ | 1 | 2 | 3 | 4 | 5 | + Q$ | 6 | 7 | 8 | 9 | 0 |

W$ = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |

```
60 PRINT "W=";W$

RUN
W$=1234567890
```

4.2. Programmazione di input e output

Per i programmatori principianti è generalmente facile arrivare ad avere una certa dimestichezza con il modo in cui il BASIC calcola i numeri. La programmazione diventa però più difficile quando si scrivono programmi che ricevono un input dalla tastiera e presentano i dati sul video.
Quasi tutti i programmi richiedono un input dalla tastiera: se siete voi stessi a usare il programma, probabilmente sarete sempre in grado di rispondere correttamente alle istruzioni di input, ma se è qualcun altro ad operare sul computer, è facile che prima o poi batta un tasto sbagliato o esegua un'errata immissione. Dovreste, perciò, scrivere programmi che tengano conto dei prevedibili errori umani.
Lo stesso è vero anche per la programmazione dell'output. Se si presentano i risultati di un programma con una serie di istruzioni PRINT, essi devono essere leggibili e comprensibili all'utente. Non si può realizzare

un buon output scrivendo e modificando continuamente le istruzioni di un programma fino a che non assume un aspetto accettabile; bisogna avere ben chiara, fin dall'inizio, la struttura della videata. Si supponga di voler scrivere un programma che carichi nomi ed indirizzi; si potrebbe farlo senza difficoltà così:

```
10 REM PROGRAMMA NOME E INDIRIZZO
20 DIM NM$(20),IN$(20),CP$(20),CA$(5)
21 REM GLI ARRAY SONO:
22 REM NM$() PER IL NOME
23 REM IN$() PER L'INDIRIZZO
24 REM CP$() PER LA CITTA' E LA PROVINCIA
26 REM CA$() PER IL CAP
30 FOR I=1 TO 20
40 INPUT "NOME:";NM$(I)
50 INPUT "INDIRIZZO:";IN$(I)
60 INPUT "CITTA'PROVINCIA:";CP$(I)
70 INPUT "CAP:";CA$(I)
80 NEXT I
90 END
```

Ecco come si presenterebbe sullo schermo:

```
RUN
NOME:? MARCO VIVALDI
INDIRIZZO:? VIA PO 15
CITTA'PROVINCIA:? MONZA MILANO
CAP:? 20052
NOME:? LUISA ADRIANI
INDIRIZZO:? VIA TRENTO 27
CITTA'PROVINCIA:? RECCO GENOVA
CAP:? 16036
```

In questo programma il video del C-64 non è formattato. L'operatore che durante l'esecuzione di questo programma si accorge, dopo aver premuto RETURN, di aver commesso un errore in un nome, non può tornare indietro a correggerlo se il programma continua a richiedere il caricamento dei dati.
Altri difetti di questo programma: la presentazione sul video non è molto ben leggibile, ogni input di un nome ed indirizzo segue immediatamente il precedente per tutta la lunghezza dello schermo. Tutta questa massa di dati confusi può essere causa di ulteriori sbagli da parte dell'operatore. L'istruzione INPUT a riga 60 può causare una sequenza di errori, se

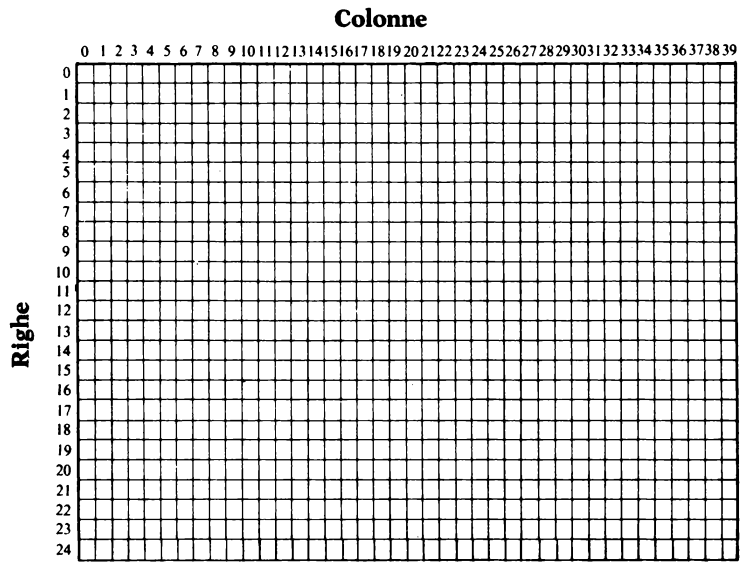
l'operatore mette una virgola tra la città e la provincia. Provate a caricare i nomi di una città e quello della provincia separati da una virgola (per esempio, MONZA, MILANO). Questo è quanto ne risulterebbe.

```
CITTA' PROVINCIA:? MONZA,MILANO
?EXTRA IGNORED
```

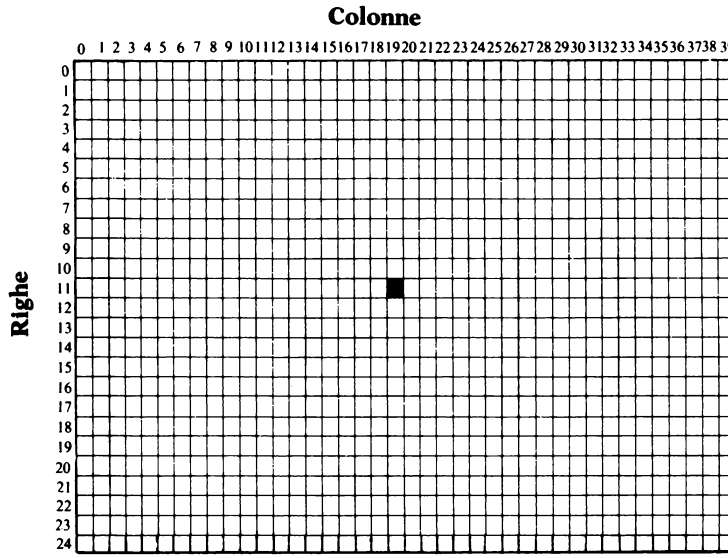
Nel capitolo 3 si è visto come l'istruzione INPUT permette di caricare più elementi purché siano separati da una virgola. Quindi quando si carica MONZA, MILANO il BASIC C-64 lo interpreta come due stringhe separate mentre soltanto una era richiesta ed attesa; da qui il messaggio ?EXTRA IGNORED. Il programma memorizza solo MONZA e scarta MILANO, in quanto considerato un input "extra".

RAPPRESENTAZIONE DEL VIDEO

Se iniziamo a numerare le posizioni sul video alla riga 0, colonna 0, (l'angolo in alto a sinistra), la colonna più a destra è la 39 e la riga più in basso è la 24. La rappresentazione del video in forma di griglia determina una serie di *coordinate*: ogni punto è individuato da una coppia di coordinate.



Una coppia di coordinate sul video del C-64 è espressa come (*riga, colonna*), quindi l'intersezione della 12ª riga con la 20ª colonna verrà espressa come (11,19). La prima colonna della quarta riga è (3,0) e così via. (Si ricordi che le righe e le colonne cominciano da 0 e non da 1.) Le coordinate (11,19) apparirebbero in questo punto dello schermo.



CREAZIONE DI UNA VIDEATA

Tenete sempre presente la massima lunghezza possibile di ogni dato da immettere. Per esempio, un nome con una professione come "DOTTOR DE FRANCESCHINI — DIRETTORE DEL LABORATORIO DI ANALISI CHIMICA", andrebbe a capo sulla riga successiva. Quando si formatta un video bisogna lasciare spazio sufficiente per queste immissioni di dati. Centrando le scritte sullo schermo, si otterrà anche l'effetto di far apparire più ordinata la presentazione; la formattazione del video è consigliabile in tutti i programmi che necessitano di numerosi dati in input.
Le tre caratteristiche di una buona tecnica di input di dati sono: una presentazione leggibile e non "intasata", istruzioni e suggerimenti all'operatore chiari e la possibilità per l'operatore di correggere eventuali errori.

PROGRAMMAZIONE DEI MOVIMENTI DEL CURSORE

Se avete già incontrato il modo tra virgolette del C-64 mentre caricavate istruzioni di programma, sapete già come programmare i movimenti del cursore. Quando si cerca di correggere un'istruzione che contiene una variabile a stringa, il BASIC C-64 interpreta anche i tasti di controllo del cursore come un carattere della stringa che si sta cercando di correggere. Per esempio, quando si preme il tasto CRSR UP/DOWN tra le virgolette di una stringa, esso apparirà come una Q inversa sul video. Invece di muovere il cursore su o giù come si vorrebbe, il BASIC C-64 inserisce il tasto CRSR UP/DOWN nella stringa. Se si scrive, con PRINT, una stringa che contiene questo carattere di controllo cursore, quest'ultima si sposta in giù. L'istruzione

```
1010 R$="■■■■■■■■■■■■■■■■■■■■■■"
```

crea una stringa di 22 caratteri CRSR DOWN che, quando stampata da un'istruzione PRINT, sposta il cursore in giù di 22 righe, così come un'istruzione contenente 22 caratteri CRSR RIGHT sposta il cursore a destra di 22 posizioni.

```
1020 C$="■■■■■■■■■■■■■■■■■■■■■■"
```

Queste stringhe contenenti i caratteri di controllo del cursore permettono di stampare un carattere sullo schermo in qualsiasi posizione.

Una subroutine di movimento del cursore

Con tre stringhe che contengono i caratteri di controllo del cursore (muovere alla posizione di home, muovere in giù e muovere a destra) è possibile spostarsi in ogni punto dello schermo del C-64. Come?
Le coordinate dell'angolo in alto a sinistra sono (0,0) (home). Si può posizionare il cursore ovunque, stampando una stringa contenente il tasto di controllo CLR/HOME, seguita da stringhe che muovono il cursore in giù e a destra. Ecco un esempio.

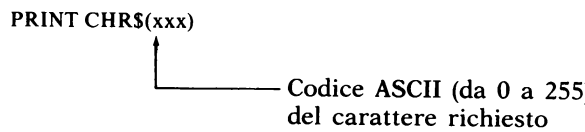
```
10 REM MOVIMENTO PROGRAMMATO DEL CURSORE
20 PRINT "■":REM PULISCE LO SCHERMO
30 R%=20:C%=4:GOSUB 1000
40 PRINT "ECCO FATTO"
50 GOTO 50
1000 REM SUBROUTINE DI POSIZIONAMENTO DEL CURSORE
1010 R$="■■■■■■■■■■■■■■■■■■■■■■"
1020 C$="■■■■■■■■■■■■■■■■■■■■■■"
```

```
1030 PRINT "■"; :REM MUOVE IL CURSORE IN (0,0)
1040 PRINT LEFT$(R$,R%);LEFT$(C$,C%);
1050 RETURN
```

La riga 2Ø pulisce lo schermo, rimuovendo tutto il testo preesistente. (Questo non ha nulla a che vedere con il posizionamento del cursore; è semplicemente una buona abitudine per ottenere una presentazione "pulita"). Le variabili intere R% e C% alla riga 3Ø rappresentano "riga" e "colonna". Le tre istruzioni abbinate a riga 3Ø fissano la riga (20) e la colonna (4), seguite da un GOSUB che muove il cursore alle coordinate (20,4). Per spostare il cursore alle coordinate giuste, il programma deve sapere quante righe e colonne separano il cursore dalle coordinate desiderate. Il modo più facile ed efficiente per far ciò è di muoverlo per prima cosa a (0,0).
Ora esamineremo ogni singola riga della subroutine di posizionamento del cursore a cominciare dalle righe 1Ø1Ø e 1Ø2Ø: esse sono due istruzioni di assegnamento che inizializzano le stringhe di movimento del cursore: R$ contiene 22 caratteri di controllo CRSR DOWN; C$ contiene 22 caratteri CRSR RIGHT.
La riga 1Ø3Ø stampa il carattere HOME CURSOR, posizionando così il cursore a (0,0). La riga 1Ø4Ø è quella che compie il passo cruciale: usa la funzione di stringa LEFT$ e stampa quindi i primi 20 caratteri di R$, poi i primi 4 di C$. Questa subroutine di posizionamento sarà una parte integrante della creazione di presentazioni formattate.

LA FUNZIONE CHR$: PROGRAMMAZIONE DI CARATTERI ASCII

Se non è possibile usare un tasto per includere un carattere in una stringa di testo, si può sempre selezionare il carattere usando il suo valore ASCII. La funzione CHR$ trasforma un numero di codice ASCII nel carattere corrispondente. Questo è il suo formato.



Per conoscere il codice ASCII di un carattere usate la tabella E.1. Esaminate la tabella fino a trovare il carattere desiderato e mettete questo numero tra le parentesi della funzione CHR$. Per esempio, per creare il simbolo $ dal suo codice ASCII, caricate 36 nella funzione come segue:

```
PRINT CHR$(36)
```

Provate a stampare questo carattere nel modo diretto:

```
PRINT CHR$(36)
$
```

Si può usare la funzione CHR$ in un'istruzione PRINT nel seguente modo:

```
10 PRINT CHR$(36);CHR$(42);CHR$(166)

RUN
$*▒
```

La funzione CHR$ permette di includere caratteri come RETURN, INST/DEL e le virgolette (") che non sarebbero altrimenti utilizzabili in un'istruzione PRINT. Si può anche utilizzare la funzione CHR$ per fare un test dell'esistenza di caratteri speciali come RETURN e INST/DEL.
Supponete che un programma abbia bisogno di controllare l'input dei caratteri dalla tastiera: si potrebbe cercare un RETURN (che ha un codice ASCII 13) nel seguente modo:

```
10 GET X$:IF X$<>CHR$(13) THEN 10
```

Questo sarebbe impossibile se si cercasse di mettere RETURN tra virgolette.

```
20 IF X$<>" RETURN " THEN 10
                  ↑
              Impossibile
```

Premendo RETURN dopo le prime virgolette infatti, il cursore andrebbe automaticamente a capo.

```
20 IF X$<>" ←——— RETURN
▒   ←——————┘
```

Se si cercasse di programmare il tasto INST/DEL o RETURN si avrebbero dei risultati sorprendenti. Il tasto INSERT è programmabile: all'interno di una coppia di virgolette in un'istruzione PRINT, si presenterebbe come ▮▮. Qualora si cercasse di programmare il tasto DELETE in un'istruzione PRINT si cancellerebbe soltanto il carattere precedente, a meno che il tasto DELETE non fosse collocato in una sequenza di caratteri "inseriti". Il tasto DELETE può essere caricato dopo un INSERT, ma ciò non è molto utile. La conseguenza positiva più evidente del fatto che il tasto DELETE non

sia programmabile è che, battendo DELETE anche in modo tra virgolette, si ottiene l'esecuzione del comando e non la visualizzazione di un carattere speciale. Il carattere RETURN in un'istruzione di PRINT rimuoverebbe immediatamente il cursore dall'istruzione e lo porterebbe alla riga successiva.

IMMISSIONE DI DATI (INPUT)

Il caricamento di dati dovrebbe essere programmato in unità funzionali: un programma di archivio, per esempio, richiede l'inserimento di nomi e indirizzi. Si deve trattare ogni nome e indirizzo completo come un'unica unità funzionale piuttosto che separarne gli elementi. In altre parole, il programma deve richiedere nome e indirizzo permettendo all'operatore di caricare tutte queste informazioni e poi di cambiarne una qualunque parte. Quando l'operatore è soddisfatto della correttezza del nome e indirizzo, il programma acquisisce l'intera informazione, per passare poi a richiedere il successivo nome e indirizzo completo. È una pessima abitudine quella di dividere i dati di input nelle loro componenti piú piccole. In un programma di indirizzi sarebbe senza senso richiedere solamente il nome, acquisire quest'informazione appena caricata e poi richiedere la via e la città, trattando ogni parte come un'unità funzionale separata. Questo tipo di approccio al problema tende a impedire la messa a punto e il cambiamento dei programmi e a renderli anche poco leggibili.
L'obiettivo dei programmi per immissione dati dovrebbe essere quello di permettere all'operatore di accorgersi degli errori e di poter correggerli facilmente.

Messaggi di suggerimento

Qualsiasi programma che richieda il caricamento di dati dovrebbe stimolare l'operatore facendo delle domande; esse sono di solito su una riga sola e richiedono semplicemente risposte tipo "si" o "no" (yes o no). Per esempio, un messaggio tipo CAMBIAMENTI? (Y o N) indicherebbe chiaramente la richiesta e le scelte possibili. Un operatore risponde a questo messaggio premendo i tasti Y o N. La pratica consiglia di introdurre un controllo che impedisca che qualsiasi altro tasto tranne Y o N venga accettato. Se l'operatore risponde alla domanda CAMBIAMENTI con una Y, un altro messaggio chiederà QUALE LINEA? (1–6). In questo caso potrebbe essere cambiata una delle sei linee; tutto quello che dovrebbe fare l'operatore sarebbe di battere il numero corrispondente alla linea caricata erroneamente. Naturalmente, con questa soluzione, ogni linea caricata

deve avere il suo numero d'identificazione.
Questo tipo di input dei dati andrebbe meglio scritto come subroutine. Inoltre, siccome è permesso solamente un numero limitato di scelte, dovrebbe contenere la logica necessaria a controllare che il dato caricato sia tra le scelte permesse. Ciò implica due cose:

1. La subroutine deve ricevere dei parametri dal programma principale. Per esempio, se il suggerimento richiede all'operatore di caricare un numero, alla subroutine devono venir forniti come parametro il numero massimo e minimo accettabile.
2. La subroutine deve riportare la risposta dell'operatore al programma principale. Questa variabile può essere un carattere (per esempio, Y o N), una parola (come sì o no) oppure un numero.

Una subroutine che suggerisce una risposta di Y per "yes" o N per "no" utilizza un'istruzione PRINT per fare la richiesta, seguita da un GET per ricevere la risposta di un carattere. Dato che si potrebbero avere dei programmi che richiedono molte risposte di "sì o no", la subroutine dovrebbe anche permettere che la domanda le venga passata dal programma principale sotto forma di variabile a stringa. Ecco qui le istruzioni necessarie:

```
3000 REM PONE UNA DOMANDA E ATTENDE UNA RISPOSTA
3005 REM 'Y' O 'N'
3010 PRINT "□"
3020 PRINT "VUOI FARE QUALCHE CAMBIAMENTO?";
3030 GET YN$: IF YN$<>"N" AND YN$<>"Y" THEN 3030
3040 PRINT YN$
3050 RETURN
```

La variabile a stringa QU$ deve essere assegnata dal programma che chiama la subroutine. Quest'ultima è generalizzata, ossia fa apparire qualsiasi suggerimento o domanda passati dal programma principale. La risposta viene ripassata a quest'ultimo nella variabile a stringa YN$.
Ora si consideri un dialogo che permetta ad un operatore di caricare un numero. Si presume che il programma principale passi alla subroutine il valore numerico minimo in LO% e quello massimo in HI%. Appena l'operatore carica un numero entro i limiti permessi, la subroutine passa il numero caricato in NM%. Qui sotto troverete la subroutine che preleva ciò che è caricato dalla tastiera, lo controlla per stabilire se è entro i limiti e lo passa poi al programma principale in NM%.

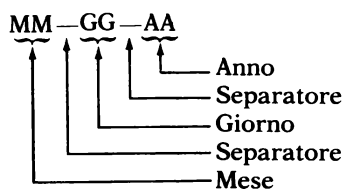
```
3500 REM CERCA UN NUMERO
3510 REM VISUALIZZA IL NUMERO IN NM%
3520 REM NM% DEVE ESSERE <= HI% AND >=LO%
```

```
3530 REM PRINT QU$;
3540 GET C$:IF C$=" " THEN 3540
3550 NM%=VAL(C$)
3560 IF NM%<LO% OR NM%>HI% THEN 3540
3570 PRINT C$;
3580 RETURN
```

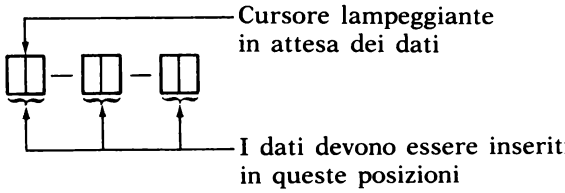
Scrivete un piccolo programma che assegni dei valori a HI% e LO% e vada poi alla subroutine 35ØØ. Aggiungete la subroutine precedente ed eseguitelo. Siete in grado di modificare la subroutine perché accetti un input di due cifre? Se non ci riuscite, aspettate fino alla sezione seguente di questo capitolo dove troverete la subroutine necessaria nel programma che controlla il caricamento della data.

Come convalidare l'input di una data

La maggior parte dei programmi richiede un input di dati relativamente semplice, ma più che un solo sì o no: si consideri ad esempio una data. Bisogna fare attenzione: con tutta probabilità la data sarà solo un elemento in una sequenza di dati. Progettando con cura l'immissione dati per ogni singola voce, si può evitare di dover ricominciare una lunga sequenza di caricamento ogni volta che l'operatore commette un errore in un singolo dato. La data è da caricare come segue:



I trattini che separano il mese, il giorno e l'anno potrebbero essere delle barre o qualunque altro carattere appropriato.
Bisogna progettare il caricamento dei dati in modo che sia accettabile all'operatore: egli dovrebbe poter vedere subito dove caricare l'informazione, che tipo di dato è richiesto e a che punto è arrivata la procedura d'immissione dati. Un buon modo di mostrare dove deve essere inserito il dato è di stampare l'intera riga sul video in caratteri inversi. Per esempio, il programma che richiede una data potrebbe generare la seguente presentazione:



Si può creare una simile presentazione con la seguente istruzione:

```
10 PRINT "■■■■  ■-■  ■-■  ■";CHR$(13);"■";
```

L'istruzione PRINT comprende comandi del cursore che posizionano la data da caricare alla riga 2. L'istruzione PRINT pulisce anche lo schermo in modo che non rimanga nulla attorno alla richiesta della data. Dopo aver presentato la riga per l'inserimento della data, l'istruzione PRINT fa ritornare il cursore alla prima posizione della data usando i caratteri RETURN e CRSR UP. Provate ad usare un'istruzione INPUT per acquisire il dato del mese. Questo si può fare come segue:

```
20 INPUT M$;
```

Caricate le istruzioni alle righe 1Ø e 2Ø, come illustrato sopra, e fate eseguire questo programma. L'istruzione INPUT non funzionerà. A parte il fatto che un punto interrogativo rimpiazza il primo carattere della riga, l'istruzione INPUT carica l'intera riga dopo il punto di domanda. A meno di non scrivere sopra a tutta la presentazione di caricamento della data — che richiederebbe l'immissione di un numero veramente grande — si otterrebbe un messaggio d'errore ogni volta che si premesse il tasto RETURN, perché il BASIC C-64 accetta tutto ciò che si trova sulla riga come se provenisse dalla tastiera. Questa è un'occasione in cui si deve usare l'istruzione GET.

```
10 PRINT "■■■■  ■-■  ■-■  ■";CHR$(13);"■";
20 GET C$:IF C$="" "THEN 20
30 PRINT C$;:MM$=C$
40 GET C$:IF C$="" "THEN 40
50 PRINT C$;:MM$=MM$+C$
60 STOP
```

Queste istruzioni accettano un input di due cifre, che fa apparire la pri-

ma parte della data. L'input di due cifre non richiede nessun RETURN o altro carattere di controllo. Il programma termina l'operazione automaticamente dopo che sono stati caricati due caratteri.
Sono necessarie due cifre per il mese, il giorno e l'anno. Invece di ripetere le istruzioni da riga 2Ø a 5Ø si potrebbe metterle in una subroutine e chiamarla tre volte, come segue:

```
10 PRINT "■■■■ ■■■■ ■■■■ ■";CHR$(13);"■";
20 GOSUB 1000:MM$=TC$:PRINTTAB(3)
30 GOSUB 1000:GG$=TC$:PRINTTAB(6)
40 GOSUB 1000:AA$=TC$
50 STOP
60 STOP
1000 REM SUBROUTINE DI INSERIMENTO DI DUE CARATTERI
1010 GET C$:IF C$=""THEN 1010
1020 PRINT C$;
1030 GET CC$:IF CC$=""THEN 1030
1040 PRINT CC$;
1050 TC$=C$+CC$
1060 RETURN
```

Le variabili MM$, GG$ e AA$ contengono il giorno, il mese e l'anno, rispettivamente. Ognuna è memorizzata come stringa di due caratteri. Si deve svuotare il buffer (memoria temporanea) di carico prima di accettare il primo dato, altrimenti qualsiasi carattere preesistente nel buffer sarebbe letto dalla prima istruzione GET della subroutine di acquisizione dei due caratteri. È necessario svuotare il buffer una sola volta prima del primo GET.
Ci sono due modi per consentire all'operatore di ricuperare eventuali errori commessi durante l'immissione di una data:

1. Il programma può controllare un'immissione di dati validi per mese giorno e anno.
2. L'operatore può ricominciare il caricamento premendo un tasto specifico.

Il programma può controllare che il mese sia entro i limiti di 01 e 12, e anche se non si curerà degli anni bisestili, verificherà il massimo numero di giorni accettabile nel mese specifico; è permesso qualsiasi anno da 00 a 99. Qualunque dato non valido farà ripartire l'intera sequenza di caricamento della data. L'intera sequenza ripartirà anche se l'operatore preme il tasto RETURN.
Il programma definitivo per l'immissione di una data appare in figura 4.1. Notate che la data è costruita pezzo per pezzo nella stringa di 8 caratteri DT$ quando giorno, mese e anno sono caricati.

```
5 REM PROGRAMMA DI ACQUISIZIONE E VERIFICA DELLA DATA
10 PRINT "■■■■ ■-■ ■-■ ■";CHR$(13);"■";
50 GOSUB 1000:REM ACQUISISCE IL MESE
60 IF TC$=CHR$(13) THEN 10
70 DT$=TC$:PRINTTAB(3)
80 REM CONTROLLA LA VALIDITA' DEL MESE
90 M%=VAL(TC$)
100 IF M%<1 OR M%>12 THEN 10
110 REM ACQUISISCE IL GIORNO
120 G%=31
130 IF M%=2 THEN G%=28
140 IF M%=4 OR M%=6 OR M%=9 OR M%=11 THEN G%=30
150 GOSUB 1000
160 IF TC$=CHR$(13) THEN 10
170 DT$=DT$+"-"+TC$:PRINTTAB(6)
190 REM CONTROLLA LA VALIDITA' DEL GIORNO
200 IF VAL(TC$)<1 OR VAL(TC$)>G% THEN 10
210 GOSUB 1000:REM ACQUISISCE L'ANNO
220 DT$=DT$+"-"+TC$
230 IF TC$=CHR$(13) THEN 10
270 STOP
1000 REM SUBROUTINE DI INSERIMENTO DI DUE CARATTERI
1010 GET C$:IF C$=""THEN 1010
1011 IF VAL(B$)>100 THEN PRINT"T"
1015 IF C$=CHR$(13) THEN 1050
1016 IF C$<"0" OR C$>"9" THEN 1010
1020 PRINT C$;
1030 GET CC$:IF CC$=""THEN 1030
1035 IF CC$=CHR$(13) THEN 1060
1036 IF CC$<"0" OR CC$>"9" THEN 1030
1040 PRINT CC$;
1050 TC$=C$+CC$
1060 RETURN
```

Figura 4.1.

Tre verifiche sono eseguite sui dati mentre vengono caricati:

1. Il carattere è un RETURN?
2. Se il carattere non è un RETURN, è una cifra valida?
3. La combinazione a doppio carattere è un mese valido per il primo dato e un giorno valido per il secondo?

Il RETURN è stato selezionato come carattere che fa ripartire l'esecuzione. Rimpiazzando CHR$(13) nelle righe 6Ø, 16Ø, 23Ø e 1Ø35, si può selezionare qualunque altro carattere per far abortire il programma. L'intera sequenza di carico ricomincia quando l'operatore preme il tasto selezionato. È necessario controllare la presenza del carattere CHR$(13) nella subroutine di input dei due caratteri (a riga 1Ø35), dato che si vuole avere la possibilità di ripartire anche dopo l'immissione della prima o seconda cifra. Il programma principale deve anch'esso controllare l'esistenza del carattere RETURN in modo da poter saltare indietro a riga 1Ø e ricominciare l'intera sequenza; si potrebbe saltare dalla subroutine alla riga 1Ø del programma principale, eliminando così la necessità di fare il controllo nel programma, ma non è un buon esempio di programmazione. Ogni subroutine dovrebbe essere trattata come un modulo a sè con uno o più punti d'entrata specificati e punti di ritorno standard.
L'uso di GOTO per saltare da subroutine a programma principale è una fonte certa di errori di programmazione. Saltando dalla subroutine al programma principale senza passare dal RETURN, ci si espone al rischio di errori subdoli, difficili da scoprire a meno di non essere già programmatori esperti.
La logica di programma che ricerca caratteri non in forma di cifre può essere contenuta interamente nella subroutine di input. Questo programma ignora qualsiasi carattere che non sia una cifra. Le istruzioni alle righe 1Ø16 e 1Ø36 controllano che i caratteri non siano cifre confrontando il valore ASCII del carattere immesso con i valori ASCII delle cifre.
La logica necessaria a controllare la validità del giorno, mese e anno deve risiedere nel programma principale dato che ognuno di questi valori a due caratteri ha dei limiti permessi diversi: l'istruzione a riga 1ØØ verifica la validità del mese; le istruzioni sulle righe 12Ø, 13Ø e 14Ø calcolano il numero massimo di giorni accettabile per il mese caricato; è per questo che si preferisce conservare nell'input della data l'uso americano (MMGGAA) invece di quello europeo (GGMMAA). L'istruzione a riga 2ØØ controlla la validità del giorno. L'equivalente numerico del mese è generato a riga 9Ø, ma non quelli del giorno o dell'anno; questo perché il giorno e l'anno non sono usati molto spesso, mentre il mese è usato da riga 9Ø fino a riga 14Ø. Si risparmierà sia memoria che tempo d'esecuzione usando una rappresentazione numerica del mese.
Certamente si impiega più tempo a scrivere un buon programma che controlli la validità dei dati immessi, permettendo all'operatore di ricomin-

ciare in ogni momento. Ne vale la pena? Senz'altro sì. Il programma si scrive una volta, mentre l'operatore potrebbe usarlo centinaia o anche migliaia di volte, per cui spendendo più tempo all'inizio nello scrivere il programma è possibile risparmiare agli operatori centinaia d'interruzioni ed errori.

INPUT STRUTTURATO DI DATI

Il modo migliore di trattare il caricamento di dati suddivisi in molteplici voci è di presentare un modulo e di far poi rispondere alle domande. Si consideri la visualizzazione di nomi ed indirizzi vista in precedenza in questo capitolo.

```
INSERIRE NOME E INDIRIZZO

1        NOME:

2         VIA:

3      CITTA':

4  PROVINCIA:

5         CAP:

6   TELEFONO:
```

Ogni riga ha un suo numero corrispondente. In questo caso presenta i numeri in modo invertito. L'operatore comincia con l'elemento 1 e finisce con l'elemento 6; egli può poi modificare qualunque linea. Le seguenti istruzioni puliscono lo schermo e fanno apparire il modulo iniziale:

```
10 REM INSERISCE NOME E INDIRIZZO
20 REM MOSTRA I DATI
30 PRINT "INSERIRE NOME E INDIRIZZO"
40 PRINT "1        NOME:"
50 PRINT "2         VIA:"
60 PRINT "3      CITTA':"
70 PRINT "4  PROVINCIA:"
80 PRINT "5         CAP:"
90 PRINT "6   TELEFONO:"
```

Il programma listato nella figura 4.2 è una versione più completa di quello proposto prima per la raccolta di nomi ed indirizzi: usa la formattazio-

```
10 REM INSERISCE NOME E INDIRIZZO
20 REM MOSTRA I DATI
30 PRINT "□■INSERIRE NOME E INDIRIZZO"
40 PRINT "■1■      NOME:"
50 PRINT "■2■       VIA:"
60 PRINT "■3■    CITTA':"
70 PRINT "■4■ PROVINCIA:"
80 PRINT "■5■       CAP:"
90 PRINT "■6■  TELEFONO:"
100 EDITING%=0
200 REM NOME DI 20 CARATTERI
210 R%=2:C%=15:LN%=20:GOSUB 8000
220 NA$=CC$
230 IF EDITING% THEN 500
250 REM VIA
260 R%=3:C%=15:LN%=20:GOSUB 8000
270 SR$=CC$
280 IF EDITING% THEN 500
300 REM CITTA' DI 12 CARATTERI
310 R%=4:C%=15:LN%=12:GOSUB 8000
320 CI$=CC$
330 IF EDITING% THEN 500
350 REM PROVINCIA DI 2 CARATTERI
360 R%=5:C%=15:LN%=2:GOSUB 8000
370 IF EDITING% THEN 500
400 REM CAP DI 5 CARATTERI
410 R%=6:C%=15:LN%=5:GOSUB 8000
420 ZI$=CC$
430 IF EDITING% THEN 500
450 REM NUMERO DI TELEFONO DI 12 CIFRE
460 R%=7:C%=15:LN%=12:GOSUB 8000
470 PH$=CC$
500 REM CHIEDE SE CI SONO CAMBIAMENTI DA FARE
510 EDITING%=-1
520 R%=10:C%=0:GOSUB 9000
530 QU$="CAMBIAMENTI?  ■"
540 GOSUB 3000:REM GET "Y" OR "N"
550 IF C$="N" THEN PRINT "□";:END
560 REM CHIEDE QUALE LINEA NECESSITA CAMBIAMENTI
570 QU$="QUALE LINEA (1-6)?:  ■"
580 R%=12:LO%=1:HI%=6
590 GOSUB 3500
600 ON NM% GOTO 200,250,300,350,400,450
610 GOTO 520
3000 REM FA UNA DOMANDA E ASPETTA UNO Y O N
3020 PRINT QU$;
3030 GOSUB 5000
3040 IF C$<>"Y" AND C$<>"N" THEN 3030
3050 PRINT C$;
3060 RETURN
3500 REM VERIFICA CHE IL NUMERO
```

(continua)

```
3510 REM SELEZIONATO SIA
3520 REM MINORE DI HI%
3530 REM E MAGGIORE DI LO%
3540 GOSUB 9000
3550 PRINT QU$;
3560 GOSUB 5000
3570 NM%=VAL(C$)
3580 IF NM%<LO% OR NM%>HI% THEN 3560
3590 PRINT C$;
3600 RETURN
5000 REM MOSTRA IL CURSORE LAMPEGGIANTE
5001 REM E ACQUISISCE IL CARATTERE
5010 FOR I=0 TO 60
5020 IF  I=0 THEN PRINT "▣ ▀▌";
5030 IF I=30 THEN PRINT " ▌";
5040 GET C$: IF C$<>"" THEN I=60
5050 NEXT I
5060 IF C$="" THEN 5000
5070 RETURN
8000 REM SUBROUTINE DI INSERIMENTO
8020 SP$="                             "
8040 GOSUB 9000
8060 PRINT "▣";LEFT$(SP$,LN%);"▀";
8070 GOSUB 9000: REM POSIZIONA IL CURSORE
8100 CC$=""
8110 GET C$:IF C$="" THEN 8110
8120 IF C$=CHR$(13) THEN 8200
8130 IF C$=CHR$(20) THEN 8160
8140 IF LEN(CC$)<LN% THEN CC$=CC$+C$:PRINT C$;
8150 GOTO 8110
8160 IF CC$="" THEN 8110
8170 PRINT"▌▣ ▀▌";
8175 REM CANCELLA IL CARATTERE DALLA STRINGA CC$
8180 CC$=LEFT$(CC$,LEN(CC$)-1)
8190 GOTO 8110
8200 IF LN%>LEN(CC$) THEN
8205 PRINT LEFT$(SP$,LN%-LEN(CC$));
8210 RETURN
9000 REM SUBROUTINE DI POSIZIONAMENTO DEL CURSORE
9010 R$="▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣▣"
9020 C$="▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶"
9030 PRINT "▣"; :REM SPOSTA IL CURSORE IN (0,0)
9040 PRINT LEFT$(R$,R%);LEFT$(C$,C%);
9050 RETURN
9500 GOSUB 5000:PRINT C$; :GOTO 9500
```

Figura 4.2.

ne video riportata qui sopra. Facendolo girare ne capirete meglio la struttura e il funzionamento che è spiegato riga per riga.
Per formattare la videata, le righe da 1Ø a 9Ø stampano ogni riga di immissione. Il carattere di controllo RVS ON precede ogni istruzione PRINT e RVS OFF la segue. Questi caratteri non stampano nulla e non occupano alcuno spazio sulla riga del video ma cambiano il modo di presentazione: qualunque carattere che segue un RVS ON sarà presentato in modo video inverso e qualsiasi carattere che segue un RVS OFF verrà stampato in maniera normale.

La sequenza di immissione dei dati

Il caricamento, una volta che appare la videata iniziale comincia alla linea del NOME. Il programma presenta una barra nera per indicare dove il dato deve cominciare, oltre alla lunghezza dello stesso. L'operatore può retrocedere lungo la linea per correggere eventuali errori di battitura usando il tasto INST/DEL. Quando il dato è completo l'operatore preme RETURN e il programma va alla linea successiva. Questa sequenza si traduce nelle seguenti istruzioni BASIC:

```
200 REM NOME DI 20 CARATTERI
210 R%=2:C%=15:LN%=20:GOSUB 8000
220 NA$=CC$
230 IF EDITING% THEN 500
250 REM VIA
260 R%=3:C%=15:LN%=20:GOSUB 8000
270 SR$=CC$
280 IF EDITING% THEN 500
300 REM CITTA' DI 12 CARATTERI
310 R%=4:C%=15:LN%=12:GOSUB 8000
320 CI$=CC$
330 IF EDITING% THEN 500
350 REM PROVINCIA DI 2 CARATTERI
360 R%=5:C%=15:LN%=2:GOSUB 8000
370 IF EDITING% THEN 500
400 REM CAP DI 5 CARATTERI
410 R%=6:C%=15:LN%=5:GOSUB 8000
420 ZI$=CC$
430 IF EDITING% THEN 500
450 REM NUMERO DI TELEFONO DI 12 CIFRE
460 R%=7:C%=15:LN%=12:GOSUB 8000
470 PH$=CC$
```

In queste istruzioni, divise in sei gruppi separati, c'è una certa uniformità. I gruppi cominciano alle righe 2ØØ, 25Ø, 3ØØ, 35Ø, 4ØØ e 45Ø, ognuno corrispondente ad un'immissione di dati che l'operatore eseguirà; ogni

gruppo comincia con una REM. Le righe da 200 fino a 230 hanno la stessa struttura degli altri cinque gruppi di istruzioni.

```
200 REM NOME DI 20 CARATTERI
210 R%=2:C%=15:LN%=20:GOSUB 8000
220 NA$=CC$
230 IF EDITING% THEN 500
```

La riga 210 assegna valori alle variabili ed esegue un GOSUB a riga 8000, che è una subroutine di immissione dati; essa usa le variabili R% e C% per specificare dove avverrà sullo schermo il caricamento dei dati. LN% contiene la lunghezza massima del dato. La riga 220 assegna il dato caricato in NA$, la variabile a stringa preposta a contenere il nome; la 230 è un controllo logico che si può ignorare per il momento, ma che sarà spiegato tra breve. Controllate ora il gruppo con inizio a riga 250: benché i valori assegnati a R%, C% e LN$ siano differenti e la riga 270 non sia proprio identica, la struttura delle righe 250-280 è molto simile a quella delle righe 200-230, e così anche per tutti gli altri gruppi.

Correzione dei dati caricati

Una volta caricate tutte le sei linee, il programma risponde con CAMBIAMENTI? e attende una risposta (Y o N). Se la risposta è N il programma ripulirà lo schermo e terminerà. Se la risposta è Y chiederà quale linea ha bisogno di essere cambiata. A questo punto l'operatore può cambiare qualunque linea in qualsiasi ordine fino a che tutte le correzioni non siano state eseguite. Le righe che eseguono questi passaggi sono le seguenti:

```
500 REM CHIEDE SE CI SONO CAMBIAMENTI DA FARE
510 EDITING%=-1
520 R%=10:C%=0:GOSUB 9000
530 QU$="CAMBIAMENTI?  "
540 GOSUB 3000:REM GET "Y" OR "N"
550 IF C$="N" THEN PRINT "";:END
560 REM CHIEDE QUALE LINEA NECESSITA CAMBIAMENTI
570 QU$="QUALE LINEA (1-6)?:  "
580 R%=12:LO%=1:HI%=6
590 GOSUB 3500
600 ON NM% GOTO 200,250,300,350,400,450
610 GOTO 520
```

La riga 510 è di particolare interesse perché abilita la procedura di correzione. A riga 100 la variabile intera EDITING% era stata azzerata. Alla fine di ogni gruppo d'istruzioni il test logico di EDITING% fa continuare

il programma al gruppo successivo. Ora che EDITING% non è più 0 (è uguale a −1) la logica del programma può accedere ad ogni gruppo in ordine casuale, permettendo così di correggere le righe caricate in qualsiasi ordine. Le righe da 52Ø a 54Ø chiamano la subroutine di YES or NO; se l'operatore carica N in risposta alla richiesta CAMBIAMENTI? il programma termina. Se carica Y il programma continua. Le righe 57Ø e 58Ø fissano le variabili per la subroutine di immissione numerica e la riga 59Ø la chiama. La subroutine riporta nella variabile intera NM% il numero della linea da cambiare (da 1 a 6) e il programma procede a riga 6ØØ. L'istruzione ON-GOTO a riga 6ØØ utilizza il numero caricato in NM% per cambiare una delle sei linee di nome ed indirizzo saltando indietro ad uno dei 6 gruppi di istruzioni. EDITING% è di particolare importanza qui perché il test logico alla fine di ogni gruppo d'istruzioni farà saltare il programma direttamente a riga 5ØØ, che è l'inizio della routine di cambiamento. Se EDITING% è 0 questo non succede: il programma avanza incondizionatamente al gruppo successivo. Provate a cambiare la riga 51Ø con EDITING%=0, e osservate la differenza di esecuzione.

Subroutine di caricamento dei dati

Ci sono sei subroutine in questo programma, ognuna delle quali ha una specifica funzione. Una delle subroutine non è chiamata dal programma principale ma è usata dalle altre subroutine. Studiate per prima la subroutine che comincia a riga 3ØØØ e finisce a riga 3Ø6Ø. Questa fa una domanda che richiede una risposta di Y o N. La subroutine espone sul video la domanda che è stata passata nella variabile a stringa QU$, chiama la subroutine a riga 5ØØØ, la quale a sua volta carica un carattere dalla tastiera. Se il carattere è Y o N, la subroutine termina e trasmette la risposta in C$.

```
3000 REM FA UNA DOMANDA E ASPETTA UN  Y O N
3020 PRINT QU$;
3030 GOSUB 5000
3040 IF C$<>"Y" AND C$<>"N" THEN 3030
3050 PRINT C$;
3060 RETURN
```

Ma per quale motivo usare una subroutine quando basterebbe un'istruzione GET per caricare un carattere?
Un'istruzione GET aspetta dalla tastiera la pressione di un tasto, ma non dà all'operatore nessun segnale sul fatto che è richiesto un carattere. La subroutine a riga 5ØØØ fa lampeggiare il cursore mentre aspetta che venga premuto un tasto, così da rendere più evidente il fatto che il C-64 sta

attendendo un qualche tipo di dato. Comunque la subroutine di acquisizione di un carattere è già utilizzata da un'altra subroutine in questo programma; è logico quindi affidare questa funzione di basso livello ad un'altra subroutine.
La subroutine che comincia a riga 35ØØ e finisce a riga 36ØØ richiede il caricamento di una singola cifra: nel programma dei nomi ed indirizzi l'unico dato numerico è il numero della riga da cambiare o modificare, che varia da 1 a 6.

```
3500 REM VERIFICA CHE IL NUMERO
3510 REM SELEZIONATO SIA
3520 REM MINORE DI HI%
3530 REM E MAGGIORE DI LO%
3540 GOSUB 9000
3550 PRINT QU$;
3560 GOSUB 5000
3570 NM%=VAL(C$)
3580 IF NM%<LO% OR NM%>HI% THEN 3560
3590 PRINT C$;
3600 RETURN
```

Questa subroutine ha bisogno di avere diverse variabili fissate dal programma principale prima di essere chiamata. Per prima cosa i valori massimi e minimi permessi del dato da caricare devono essere fissati nelle variabili intere HI% e LO%; poi, la subroutine presenta una domanda sul video contenuta in QU$ che deve pure essere stabilita prima di chiamare la subroutine.
Essa posiziona il cursore in una data posizione sullo schermo e le coordinate di questo punto devono essere trasferite alla subroutine come variabili R% e C%. La riga 354Ø chiama la subroutine di riga 9ØØØ, la quale posiziona il cursore. Riga 355Ø presenta la richiesta e riga 356Ø chiama la subroutine che carica (con GET) il dato dalla tastiera. La subroutine a riga 5ØØØ non ha bisogno di alcun parametro. Una istruzione FOR-NEXT a riga 5Ø1Ø fa cominciare un ciclo temporizzatore che presenta un'area in video inverso per le prime 30 esecuzioni del ciclo. Una volta che l'indice variabile I raggiunge il valore di 30, l'istruzione a riga 5Ø3Ø cancella il cursore. Durante questo periodo la subroutine cerca costantemente un dato caricato dalla tastiera. Quando viene premuto un tasto l'istruzione a riga 5Ø4Ø fa terminare il loop fissando I= a 60, e così finisce anche la subroutine e il carattere caricato viene passato in C$.

```
5000 REM MOSTRA IL CURSORE LAMPEGGIANTE
5001 REM E ACQUISISCE IL CARATTERE
```

```
5010 FOR I=0 TO 60
5020 IF  I=0 THEN PRINT "■ ■";
5030 IF I=30 THEN PRINT " ■";
5040 GET C$: IF C$<>"" THEN I=60
5050 NEXT I
5060 IF C$="" THEN 5000
5070 RETURN
```

La subroutine che inizia a riga 8ØØØ e finisce alla 821Ø è preposta a posizionare il cursore e ad accettare una stringa di una lunghezza specificata; quindi ha bisogno di conoscere le coordinate (riga e colonna) dell'inizio e la lunghezza della stringa da caricare prima di poter cominciare.
LN% contiene la lunghezza in caratteri della stringa da caricare, mentre R% e C% specificano la riga e la colonna del dato. Alla variabile SP$ vengono assegnati 22 spazi ed è usata in due punti nella subroutine. Poi viene chiamata dall'istruzione GOSUB a riga 8Ø4Ø la subroutine che posiziona il cursore. La riga 8Ø6Ø contiene il comando RVS ON, seguito da un blocco di caratteri di spaziatura inversi che mostrano all'operatore la lunghezza del dato da caricare, e poi RVS OFF.

```
8000 REM SUBROUTINE DI INSERIMENTO
8020 SP$="                      "
8040 GOSUB 9000
8060 PRINT "■";LEFT$(SP$,LN%);"■";
8070 GOSUB 9000: REM POSIZIONA IL CURSORE
8100 CC$=""
8110 GET C$:IF C$="" THEN 8110
8120 IF C$=CHR$(13) THEN 8200
8130 IF C$=CHR$(20) THEN 8160
8140 IF LEN(CC$)<LN% THEN CC$=CC$+C$:PRINT C$;
8150 GOTO 8110
8160 IF CC$="" THEN 8110
8170 PRINT"■■ ■";
8175 REM CANCELLA IL CARATTERE DALLA STRINGA CC$
8180 CC$=LEFT$(CC$,LEN(CC$)-1)
8190 GOTO 8110
8200 IF LN%>LEN(CC$) THEN
8205 PRINT LEFT$(SP$,LN%-LEN(CC$));
8210 RETURN
```

Le istruzioni da riga 81ØØ fino a riga 815Ø accettano un carattere dalla tastiera. Se il carattere caricato è un CHR$(13), ovvero RETURN, l'immissione è considerata completata. Viene così eseguito un salto a riga 82ØØ. La riga nera sparisce e la subroutine ritorna con il carico completo contenuto in CC$. La subroutine controlla anche l'esistenza del carattere corrispondente al tasto INST/DEL; quando esso viene premuto, la subroutine salta a riga 817Ø.

Se il carattere caricato non è nè RETURN nè INST/DEL, quello caricato in C$ è concatenato a CC$ che contiene tutti i caratteri caricati fino ad allora. Notate il test logico su questa riga:

```
8140 IF LEN(CC$)<LN% THEN CC$=CC$+C$:PRINT C$;
```

Se la lunghezza di CC$ non raggiunge ancora il massimo numero di caratteri permesso per quel dato (in LN%), allora C$ è attaccato alla fine di CC$, il carattere caricato è fatto apparire sul video e contemporaneamente avviene un salto indietro all'istruzione GET. Qualora fosse stato raggiunto il massimo numero di caratteri, l'ultimo verrebbe semplicemente ignorato.
Cosa fanno le istruzioni da riga 816Ø a 819Ø? Avviene un salto a queste istruzioni quando l'operatore preme il tasto INST/DEL. Se l'operatore preme INST/DEL ma la stringa di carico CC$ è vuota, nessun carattere deve venir cancellato. L'istruzione IF-THEN a riga 816Ø controlla questa condizione e salta a riga 811Ø se non deve essere cancellato nessun carattere. Altrimenti la logica di programmazione prosegue a riga 817Ø.
La riga 817Ø stampa i caratteri CRSR LEFT e RVS ON per posizionare il cursore sull'ultimo carattere caricato, uno spazio (in video inverso) che cancella quel carattere, poi un RVS OFF e un CRSR LEFT. Il risultato di tutto ciò è di muovere a sinistra, cancellare l'ultimo carattere e muovere di nuovo a sinistra nella posizione dello spazio.
La riga 818Ø cancella l'ultimo carattere di CC$ misurandone la lunghezza, utilizzando la funzione LEN, sottraendo 1 e riassegnando a CC$ tutti i suoi caratteri originali meno l'ultimo. Una volta che il carattere è stato cancellato dallo schermo e da CC$, la subroutine salta indietro fino all'istruzione GET di riga 811Ø.
Si dovrebbe studiare attentamente il programma e comprendere i seguenti artifici che sono stati usati per il caricamento dei dati:

— L'inversione del campo sulla linea per indicare in modo chiaro che sono richiesti dei dati e quanti caratteri sono disponibili.
— Un operatore non ha bisogno di riempire tutti i caratteri di una linea. Quando si preme il tasto RETURN, lo spazio rimanente viene riempito di spazi vuoti.
— In ogni momento l'operatore può ritornare indietro e correggere errori su di una linea usando il tasto INST/DEL.
— Quando vengono poste delle domande il programma riconosce soltanto risposte significative: Y o N per "sì" e "no", o un numero da 1 a 6 per scegliere una linea. Per esempio, riconoscere Y come "sì" ma qualsiasi altro carattere come "no" sarebbe disastroso, dato che premendo per sbaglio qualunque altro tasto si cancellerebbe il gruppo di dati visualizzati in quel momento. Riconoscere N come "no" ma gli altri caratte-

ri come "sì" porterebbe l'operatore a ricaricare di nuovo i dati nel campo qualora fosse premuto un tasto sbagliato.

Qui sono riportate alcune precauzioni addizionali per la protezione di dati da caricare:

— Controllare il codice postale CAP per evitare l'inclusione di caratteri non numerici: tuttavia, codici postali al di fuori dell'Italia usano anche caratteri alfanumerici.
— Molti programmatori "cauti" fanno la domanda: SEI SICURO? quando un operatore risponde con "no" alla domanda CAMBIAMENTI? Questo dà una seconda possibilità all'operatore che toccasse per sbaglio un tasto.
— Si potrebbe aggiungere un comando per fare rifiutare il dato appena caricato e ripristinare il valore precedente. Per esempio, se l'operatore carica il numero sbagliato nello scegliere il campo che deve essere modificato, il programma impone all'operatore di ricaricare tutta la riga anche se è corretta.

Provate a modificare il programma per includere anche le misure di sicurezza aggiuntive descritte sopra.

4.3. L'orologio

Un'altra caratteristica del C-64 è l'orologio collegato al clock del sistema, ovvero quel particolare circuito che scandisce il ritmo di esecuzione delle istruzioni da parte del microprocessore. Questo orologio segna l'ora in un ciclo di 24 ore, in ore, minuti e secondi. Le variabili a stringa riservate TIME$ o TI$ tengono conto dell'ora.

REGOLAZIONE DELL'ORA

Per regolare l'orologio usate il seguente formato:

TIME$ = *"hhmmss"*

dove:

hh è l'ora, fra 0 e 23
mm sono i minuti, fra 0 e 59
ss sono i secondi, fra 0 e 59

Per regolare TIME$ all'ora giusta, predisponete con anticipo l'assegnazione, poi appena l'ora esatta è raggiunta, premete RETURN e l'orologio inizia a contare.

```
TIME$="120150"
```

ACCESSO ALL'OROLOGIO

Per accedere all'ora, battete la seguente istruzione nel modo diretto:

```
?TIME$
```

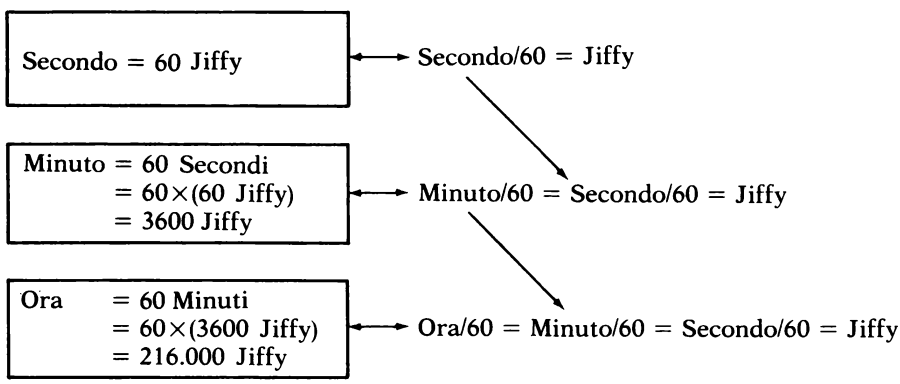
Il computer presenterà l'ora nella forma di *hhmmss*.

```
?TIME$
120200
```

L'orologio del C-64 funziona fino a che il computer stesso è acceso e deve venir regolato nuovamente se il computer viene spento.

FUNZIONAMENTO DELL'OROLOGIO

Il C-64 conta il passare del tempo in "*jiffy*"; un jiffy è 1/60 di secondo. TIME, o TI, è una variabile numerica riservata che si incrementa automaticamente ogni 1/60 di secondo; essa è fissata a zero all'inizio e a ogni 51.839.999 jiffy. TIME$ è una variabile a stringa che viene generata da TIME. Quando viene chiamata, il computer presenta l'ora in ore, minuti e secondi (hhmmss), ovvero converte il numero dei jiffy in tempo reale. Notate che TIME$ e TI$ non sono le rappresentazioni in stringa di TIME e TI; sono numeri che rappresentano il tempo reale, calcolato da quello in jiffy (TIME, TI). La conversione avviene come segue: ogni secondo è diviso in 60 jiffy. Un minuto è composto da 60 secondi e ogni ora da 60 minuti. Per cui un secondo è uguale a 60 jiffy, un minuto a 3600 jiffy e un'ora a 216.000 jiffy, come illustrato.



Il seguente programma converte il tempo in jiffy (J) in tempo reale, raffigurato come ore (H), minuti (M) e secondi (S).

```
10 J=TI
20 H=INT(J/216000)
```

Calcola le ore. La funzione INTeger preleva solo la parte intera del numero.

```
30 IF H<>0 THEN J=J-H*216000
```

Se ci sono delle ore, sottrae il numero di jiffy in un'ora moltiplicato per H per ottenere i jiffy rimanenti.

```
40 M=INT(J/3600)
```

Calcola i minuti. La funzione INT preleva solo i minuti interi.

```
50 IF M<>0 THEN J=J-M*3600
```

Se vi sono dei minuti, sottrae il numero di jiffy contenuti in un minuto moltiplicato per M per restare con i jiffy rimanenti.

```
60 S=INT(J/60)
```

Calcola i secondi. La funzione INT preleva solo i secondi interi.

```
5 PRINT "■REAL TIME":PRINT:PRINT
10 J=TI
15 T$=TIME$
20 H=INT(J/216000)
30 IF H<>0 THEN J=J-H*216000
40 M=INT(J/3600)
50 IF M<>0 THEN J=J-M*3600
60 S=INT(J/60)
70 H$=RIGHT$(STR$(H),2)
80 M$=RIGHT$(STR$(M),2)
90 S$=RIGHT$(STR$(S),2)
100 PRINT"H:M:S: ";H$;":";M$;":";S$;"   ";
105 PRINT"TIME$: ";T$;"■■■■■■■■■■■■■■■"
110 PRINT "■■■";:GOTO 10
```

In questo programma le istruzioni da 7Ø a 9Ø convertono i risultati numerici in una forma adatta ad una presentazione ordinata. L'istruzione numero 1ØØ stampa sia il tempo reale calcolato dal programma sia TIME$, il tempo reale calcolato automaticamente dal computer. Notate che il risultato è identico in entrambi i casi. Per avere un'idea della velocità dei jiffy e della conversione dall'orologio in jiffy a quello in ore normali, caricate il seguente programma che presenta il tempo di TIME$ e di TIME (TI) in continuo.

```
5 REM **OROLOGIO**
10 PRINT"■REAL TIME: ":PRINT:PRINT"JIFFY TIME:";"■"
```

```
15 T$=TIME$
20 FOR I=1 TO 235959
30 :PRINT"■";TAB(13);TIME$;"■"
40 :FOR J=1 TO 60 STEP 2
50 :PRINT"■";TAB(12);TI;"■"
60 :NEXT
65 PRINT"■■";
70 NEXT
```

Il loop FOR-NEXT per TIME a riga 40 incrementa di STEP 2 (ogni due jiffy) per le seguenti ragioni:

— La presentazione di 60 jiffy al secondo è troppo veloce per poter essere letta e inoltre impiega più tempo che a fare l'incremento. Questo ritarderebbe il loop, quindi la presentazione di TIME$ sarebbe più lenta di quanto e dovrebbe essere. Si può memorizzare questo problema di rallentamento incrementando e stampando ogni due jiffy. Fate eseguire questo programma e vedrete che i jiffy incrementano fino a 60 ogni secondo. Eseguite questo programma senza STEP 2 a riga 40 ed osservate il ritardo nella stampa di TIME$.

Calcolare il tempo in jiffy è utile per conoscere il tempo d'esecuzione dei programmi, e controllarne l'efficienza. Considerate il seguente breve programma:

```
10 PRINT"■**PROVA TASTIERA**":PRINT
20 FOR I=32 TO 127
30 PRINT CHR$(I);
40 NEXT I
50 FOR J=161 TO 255
60 PRINT CHR$(J);
70 NEXT J
80 PRINT:PRINT:PRINT"**FINE PROVA**"
```

Si può calcolare il tempo d'esecuzione come segue:

1. TI (o TIME) viene assegnato ad una variabile all'inizio del ciclo da controllare.
2. TI (o TIME) viene controllato alla fine del periodo controllato. Sottraendo il primo valore di TI dall'ultimo, la differenza sarà il tempo necessario, in jiffy, per eseguire il programma in questione.

Il seguente listato aggiunge questi ultimi passi:

```
10 PRINT"□**PROVA TASTIERA**":PRINT
15 A=TI
20 FOR I=32 TO 127
30 PRINT CHR$(I);
40 NEXT
50 FOR J=161 TO 255
60 PRINT CHR$(J);
70 NEXT
80 PRINT:PRINT:PRINT"**FINE PROVA**"
100 PRINT:PRINT"TI = ";TI-A
```

Mentre il programma procede, TI incrementa 60 volte ogni secondo. La riga 1ØØ sottrae il primo valore di TI (A) dall'ultimo valore.
C'è voluto un tempo di 41 jiffy per far apparire i caratteri della tastiera. Dividete il tempo in jiffy per 60 (il numero di jiffy in un secondo):

41/60 = 0,6833

per cui ci sono voluti 0,6833 secondi per l'esecuzione di questo programma.

4.4. Numeri casuali

I numeri casuali possono essere utilizzati nei programmi di giochi sul C-64; essi hanno usi pratici anche in statistica e in altre aree. Il C-64 genera numeri casuali utilizzando la funzione RND (random): essa fornisce un numero reale da 0 a 1. Questo numero è di fatto *pseudo* casuale, cioè non è generato strettamente a caso, ma questo è un argomento d'interesse solo per i matematici.
Il numero prodotto è una buona approssimazione di un numero casuale. Per determinare il grado di "casualità" che avrà la funzione RND, si fornisce un numero iniziale o "*seme*". Usando 0 come seme, ovvero RND(0), i valori generati si basano su tre clock interni separati: le probabilità che tutti e tre i clock segnino la stessa ora due volte di seguito sono molto basse, quindi qualsiasi numero generato si può considerare casuale.
La sequenza della maggior parte dei numeri generati sarà sempre la stessa; le uniche eccezioni sono RND(0) e RND(TI). Si possono ottenere numeri quasi completamente a caso usando un seme uguale a 0. Una sequenza di numeri prevedibili invece può essere ottenuta usando un seme negativo. Può sembrare poco utile avere un numero casuale con un'estensione di valori così limitata. Per ottenere numeri più grandi basta moltiplicare

il numero per il valore massimo desiderato. Per esempio, per ottenere un numero casuale tra 0 e 100, moltiplicate il numero ottenuto per 100. Caricate il seguente programma:

```
1 REM RANDOM
5 X=100
10 R1=RND(0):REM ACQUISISCE UN NUMERO CASUALE
20 R2=X*R1
50 PRINT" BASE       #";R1
55 PRINT"RICHIESTO   #";R2
```

Battete RUN e RETURN. Il computer sceglierà un numero casuale e lo moltiplicherà per 100, per poi visualizzarlo.

```
 BASE       # .246096194
RICHIESTO   # 24.6096194
```

Date ad X un valore positivo inferiore a 100, poi fate girare il programma per vedere un numero con limiti diversi. Provate a dare ad X un valore negativo intero e fate eseguire di nuovo il programma: questo vi darà un numero casuale negativo con un limite massimo di 0. Avendo bisogno di avere il risultato arrotondato al numero intero o decimale più vicino, aggiungete al programma le seguenti righe:

```
2 Y=3:REM NUMERI CASUALI CON DECIMALI
30 R3=INT(X*R1+.5):REM ARROTONDA ALL'INTERO
31 REM PIU' VICINO
40 R4=INT(X*R1*10↑Y)/10↑Y:REM ARROTONDA AI
41 REM DECIMALI RICHIESTI
50 PRINT" BASE       #";R1
55 PRINT"RICHIESTO   #";R2
60 PRINT"ARROTONDATO ";R3
65 PRINT"ARROTONDATO CON 3 DECIMALI";R4
```

La variabile Y controlla il numero di cifre decimali presenti nel numero arrotondato. I risultati saranno simili ai seguenti:

```
 BASE       # .246096194
RICHIESTO   # 24.6096194
ARROTONDATO  24
ARROTONDATO CON 3 DECIMALI 24.610
```

GENERAZIONE DI UN LANCIO CASUALE DI DADI

I numeri casuali vengono generati fra 0 e il limite tendente a 1. Si dovrà poi convertire il numero generato entro i limiti da 1 a 6 (come per il tiro di un dado) moltiplicando il numero casuale per 6.

```
6 * RND(1)
```

Ciò riporta un numero reale entro un limite appena superiore a 0 ma inferiore a 6. Aggiungete 1 per ottenere un numero tra 1 e 6.

```
6 * RND(1)+1
```

Convertite poi il numero in un intero, scartando la parte decimale del numero, e ottenendo così il numero tra 1 e 6 ma in forma di intero.

```
INT(6 * RND(0) + 1)
o:
A% = 6 * RND(0) + 1
```

Sono evidenziati di seguito i casi generali per convertire la frazione RND in un numero intero con diversi limiti.

INT (RND(0) * N)	Limite da 0 a N − 1
INT (RND(0) * N + 1)	Limite da 1 a N
INT (RND (0) * N + M)	Limite da M a N + M − 1

Ora sperimentate limiti di generazione differenti, modificando le istruzioni sopra illustrate. Il programma che segue mostra l'uso di −TI per ottenere un seme casuale. Questo programma calcola numeri entro i limiti di M e N. I valori di M e N, che possono essere negativi, sono fissati alla riga 1Ø per ogni esecuzione del programma. Nel seguente esempio il video presenta una sequenza infinita di numeri a caso tra −50 e +50. (Premete il tasto STOP per arrestare il programma). Verrà stampata una sequenza diversa ogni volta che il programma viene eseguito, dato che −TI fornisce un seme a caso. Notate che è il valore di X calcolato da RND(−TI) che è stampato sul video invece del valore di TI.

```
10 M=-50:N=50
20 X=RND(-TI):PRINT X
30 FOR I=1 TO 8
40 C%=(N-M+1)*RND(1)+M
50 PRINT C%
60 NEXT I
```

```
RUN
 8.27633085E-06
-14
 29
 7
 35
-32
-12
 48
-18
```

Per illustrare i diversi limiti dei numeri, cambiate i valori di M e N alla riga 1Ø del precedente programma. Per esempio, fissate M=1 e N=6; ciò genererà una sequenza infinita di numeri a caso tra 1 e 6.

SELEZIONE CASUALE DI CARTE DA GIOCO

Un rapido sguardo alla videata precedente rivela che alcuni numeri si ripetono tra i primi 100 generati. Quindi 101 numeri non includeranno ogni numero tra i limiti −50 e +50 senza alcuna duplicazione. Questo va benissimo in un gioco con i dadi ad esempio, ma per altre applicazioni si potrebbe avere bisogno di produrre numeri a caso entro certi limiti, senza ripetizioni. Una smazzata di carte è una di queste applicazioni: una volta che una carta è stata scelta non può più esserlo nella stessa mano. Il programma che segue mostra una maniera di mischiare un mazzo di carte sul C-64. Questo programma riempie una tabella a 52 elementi D% con i numeri da 1 a 52 in un ordine a caso (l'elemento D%(0) non è usato). Le carte possono essere accoppiate ai numeri casuali in qualunque modo, come

A = 1, 2 = 2, 3 = 3, ..., Q = 12, K = 13
Picche = 0; Cuori = 13; Quadri = 26; Fiori = 39

Con questo schema l'Asso di Picche è uguale a 1+0 = 1, la Donna di Picche è uguale a 12+0 = 12, il Tre di Cuori a 3+13 = 16 e così via.
Nel programma di mescolamento, una tabella-flag FL a 52 elementi tiene conto di quando una carta è stata scelta. Le istruzioni PRINT sono inserite per stampare i valori del seme della funzione, seguito dai numeri, in un formato di stampa su righe. Notate che sono stampati esattamente 52 numeri e che nessuno di essi è ripetuto. Ogni nuova esecuzione del programma produrrà una sequenza diversa, sempre a caso.

```
10 DIM A(53)
20 E=52
30 FOR R=0 TO 51
40 A(R)=R
50 NEXT R
60 B=INT(RND(0)*E)
70 PRINT A(B),
80 FOR R=B TO E
90 A(R)=A(R+1)
100 NEXT R
110 E=E-1
120 IF E>0 THEN 60
```

```
8        48        23        34
  0        22        41         9
    51         2         18         13
      38         25         24         33
        4         11         44         20
          14         6         28         50
            12         10         21         36
              17         47         45         39
                42         35         37         1
                  15         5         27         43
                    3         30         31         32
                      29         49         46         7
                        40         26         16         19
```

Nei prossimi programmi si scopriranno diversi usi per i numeri casuali, specialmente nei programmi di giochi. Nei prossimi due capitoli si vedrà come utilizzare appieno le capacità del C-64 come macchina da gioco.

Capitolo

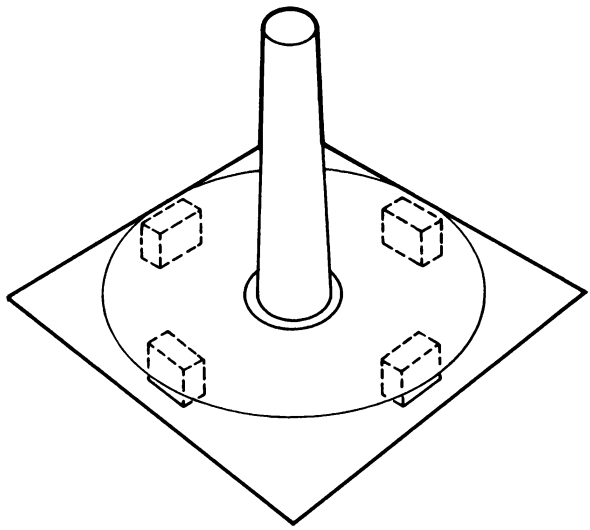
Dispositivi di comando per giochi

5

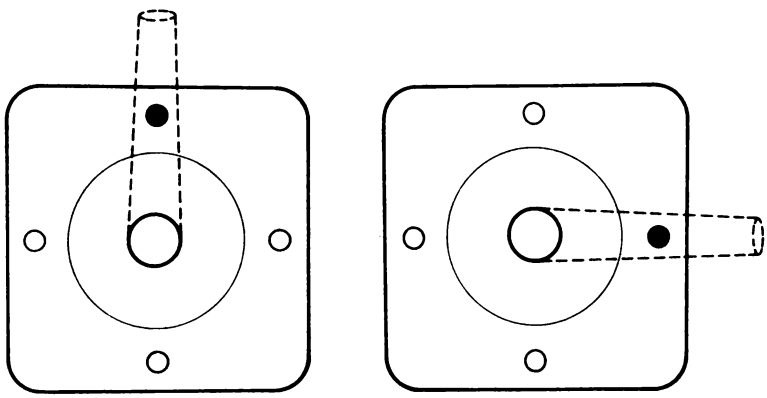
Nei programmi esaminati sino ad ora è stato usato un tono colloquiale con l'operatore; i programmi infatti si fermavano e rimanevano in attesa di risposta o immissione dalla tastiera per poi agire in base ad essa. Ciò funziona benissimo finché si tratta di far quadrare i conti in banca o scrivere delle lettere, ma molte applicazioni del C-64 richiedono un altro stile di comunicazione. Un programma che simula il pilotaggio di un aeroplano non è verosimile se l'aereo si ferma in aria mentre il "pilota" inserisce le istruzioni dalla tastiera!
Per rendere più realistico questo tipo di programma (e meno noioso da usare), il C-64 può ricevere gli ordini da un'altra fonte: il C-64 è in grado di usare dei "*controllori*", come il *joystick* o il *paddle* simili a quelli usati nei giochi elettronici dei bar.
In questo capitolo si vedrà come realizzare programmi che utilizzano questi dispositivi e si descriverà anche come usare la tastiera "in corsa", eliminando i passaggi d'arresto e attesa. Se non si possiede un joystick l'esempio con la tastiera spiega come simularne uno.

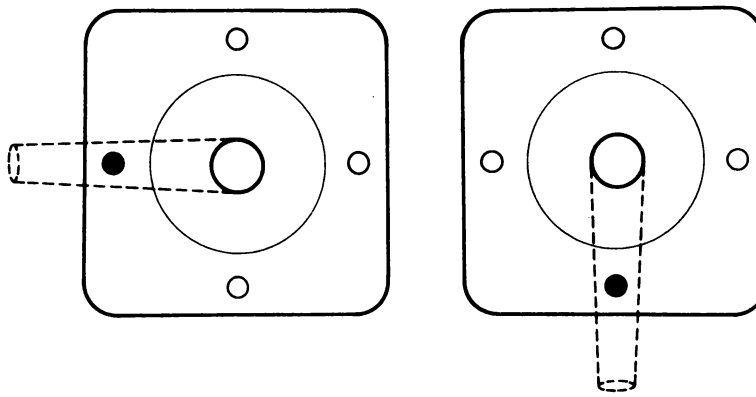
5.1. Il joystick

Il comando joystick, come la cloche dei vecchi aeroplani, controlla per mezzo di quattro interruttori (su, giù, sinistra e destra), il movimento in quattro direzioni degli elementi sul video. All'interno del joystick ci sono dei contatti che azionano questi interruttori quando esso viene mosso.

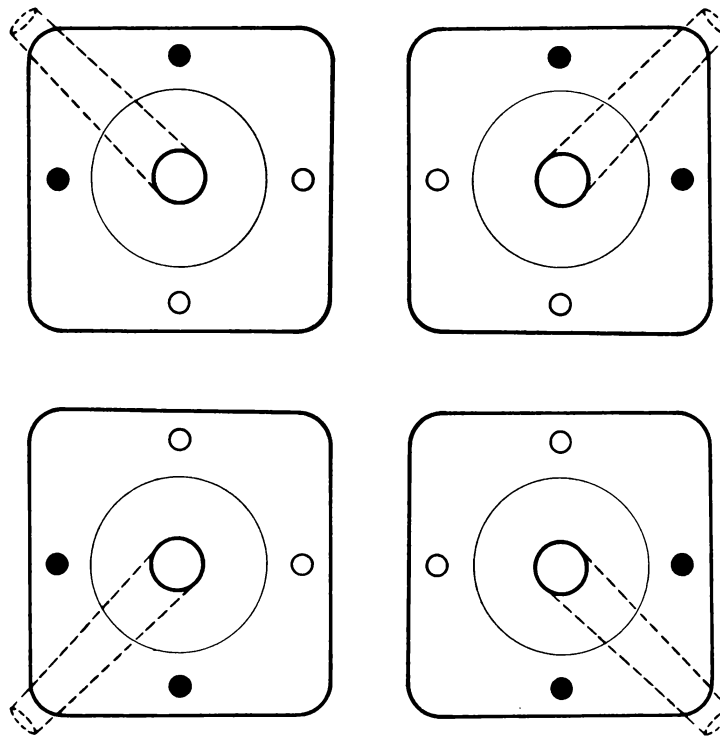


Se il comando è mosso in su, in giù, o da un lato, viene azionato un solo interruttore.





Se il comando è mosso diagonalmente, vengono interessati due interruttori.



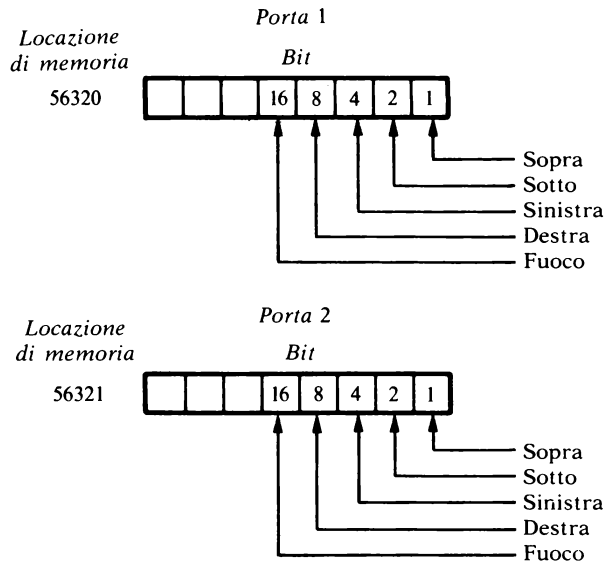
C'è anche un pulsante di "fuoco", fornito di un suo interruttore. Usando il metodo descritto nelle sezioni seguenti, il vostro programma può capire se questi interruttori sono aperti o chiusi e determinare la posizione del joystick.

LA SCHEDA CIA

Il joystick è collegato al C-64 tramite circuiti integrati chiamati *Complex Interface Adapters* (*CIA*). Alcuni *pin* della scheda CIA sono collegati al "mondo esterno"; essi ricevono i segnali loro inviati (input) o mandano segnali ad un altro dispositivo (output). I circuiti nella scheda CIA permettono al C-64 di assegnare o esaminare i segnali su questi pin utilizzando locazioni di memoria centrale. I segnali possono essere letti e controllati dalle istruzioni BASIC, PEEK e POKE.

PROVA DEGLI INTERRUTTORI DEL JOYSTICK

I pin input/output della CIA sono divisi in due gruppi di otto. Ciascun gruppo può essere assegnato o verificato tramite una singola locazione di memoria, con un bit corrispondente a ogni pin. Gli interruttori per la porta di controllo 1 sono collegati ad un gruppo di pin e quelli per la porta di controllo 2 sono collegati all'altro gruppo.



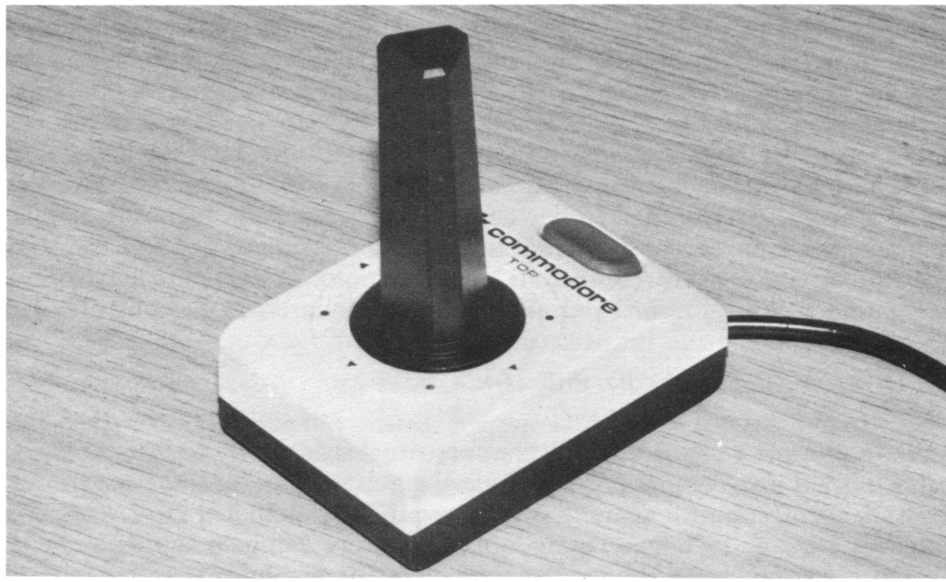


Figura 5.1. Il joystick Commodore.

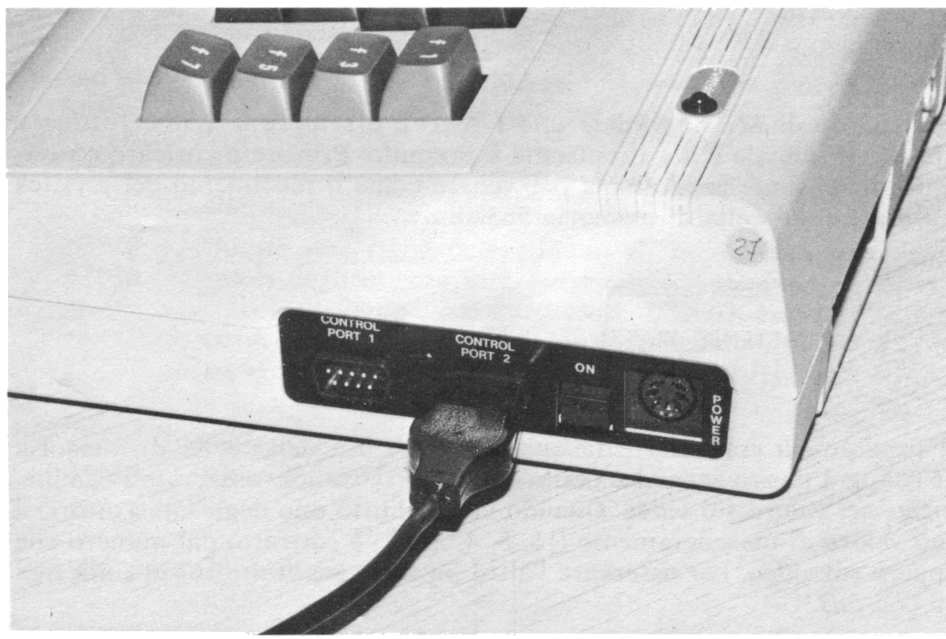


Figura 5.2. Collegamento del joystick al C-64.

Siccome i quattro interruttori direzionali e il pulsante di fuoco sono nella stessa cella di memoria, viene utilizzato l'operatore AND per isolare il bit di un particolare interruttore dagli altri bit richiamati dalla stessa locazione tramite l'istruzione PEEK. Questo uso di AND è stato esaminato nella sezione "Operatori Booleani" del capitolo 3. Come risultato dell'operazione AND verranno azzerrati tutti i bit eccetto quello che si vuole esaminare. (Questa operazione si chiama *mascheramento.* Proprio come si usa il nastro adesivo per coprire i serramenti in legno mentre si imbiancano le pareti, i programmi usano delle maschere per coprire i bit che non devono essere controllati.) Per esempio, la seguente istruzione determina se è premuto il pulsante di fuoco della porta di controllo 1:

```
10 FB = (PEEK(56320) AND 16) = 0
```

AND 16 elimina i valori degli altri interruttori mascherando tutti i bit eccetto il 5°. Il programma poi paragona il risultato a 0. Gli interruttori del joystick forniscono un segnale 0 ai loro pin CIA quando l'interruttore è chiuso e 1 quando è aperto; quando l'interruttore è chiuso significa che il pulsante è premuto. Studiate l'operazione AND in binario.

```
00000110
00010000
--------
00000000
```

Il risultato di AND è 0, dato che 1 AND 0 è sempre 0. L'interruttore è chiuso, il segnale è 0 e il pulsante è premuto. Provate a caricare ed eseguire il seguente programma per vedere come il movimento del joystick influisce sulla cella di memoria 56320:

```
10 PRINT PEEK(56320)
20 FOR I=1 TO 250 : NEXT I
25 REM ATTENDE MEZZO SECONDO
30 GOTO 10
```

Il programma controlla e fa apparire il valore della cella di memoria 56320 ogni mezzo secondo. Come muovete il joystick, osservate i cambiamenti nel valore sul video. Quando viene chiuso uno degli interruttori, il suo valore di mascheramento (16, 8, 4, 2 o 1) è sottratto dal numero che appare sul video. Per osservare l'altro joystick, sostituite 56320 sulla riga 1Ø con 56321.

UN CONTROLLORE COMPLETO DI JOYSTICK

Ora verrà esaminata la programmazione necessaria a convertire i bit di memoria in movimenti fisici. Per utilizzare i joystick in giochi diversi sarà comoda una subroutine che possa essere inserita in qualsiasi programma. Questa subroutine fissa tre variabili per il programma principale:

XI	è l'incremento X; controlla il movimento a destra e a sinistra.
YI	è l'incremento Y; controlla il movimento in su o in giù.
FB	segnala quando è premuto il pulsante di fuoco.

XI è fissato a −1 per la sinistra, +1 per la destra e a 0 per nessun movimento laterale. YI è fissato a −1 per il basso, +1 per l'alto e 0 per nessun movimento verticale. Se il pulsante di fuoco è premuto, FB sarà 1; altrimenti sarà 0.

```
63000 XT%=PEEK(56320)AND31
63020 XI=SGN(XT%AND4)-SGN(XT%AND8)
63030 YI=SGN(XT%AND1)-SGN(XT%AND2)
63040 FB=1-SGN(XT%AND16)
63050 RETURN
```

Questo programma usa alcuni trucchi per farlo eseguire più velocemente. Si esamini il suo funzionamento riga per riga.

63000	Controlla il joystick inserito nella porta e disattiva la tastiera.
63020 63030	Queste due righe determinano gli incrementi X e Y dai valori degli interruttori. Per ridurre il tempo necessario a calcolarli, viene usata la funzione SGN. Quest'ultima riporta un valore di 1 se l'interruttore è spento e di 0 se è acceso, cosa più veloce che non paragonare il risultato di AND a 0.
63040	Preleva il valore del pulsante di fuoco.
63050	Ritorna (RETURN) al programma principale.

Uso del controllore di joystick

Il seguente semplice programma illustra le capacità del controllore di joystick, muovendo un oggetto sullo schermo:

```
100 PRINT "■*";
200 GOSUB 63000
300 IF XI=0 AND YI=0 THEN 200
400 PRINT CHR$(20); : REM CANCELLA CARATTERE
500 IF XI=-1 THEN PRINT "■*";
600 IF XI= 1 THEN PRINT "■*";
700 IF YI=-1 THEN PRINT "■*";
800 IF YI= 1 THEN PRINT "■*";
900 GOTO 200
```

5.2. I paddle

I paddle (o "racchette") derivano il loro nome dall'uso nei video giochi del tipo ping-pong. Ogni comando consiste in una resistenza variabile chiamata *potenziometro*, comandata da una manopola e da un pulsante come il joystick. Come quello del joystick, l'interfaccia del paddle è compatibile con tutti i paddle disponibili sul mercato.
Il valore del potenziometro viene letto dal SID del C-64 e convertito in un numero tra 0 e 255. Gli interruttori sono letti dalla scheda CIA utilizzando due dei pin del joystick. (La tabella 5.1 elenca l'ubicazione e il contenuto della memoria di controllo del paddle.)

Tabella 5.1. Locazioni di memoria del controllore del paddle

Locazione	*Contenuto*
54297	movimento a sinistra
54298	movimento a destra
56320	pulsanti dei paddle A e B
56321	pulsanti dei paddle C e D

Prima di poter leggere i valori dei quattro potenziometri, si deve disattivare il controllore della tastiera, usando una semplice routine di due byte in linguaggio macchina. Poi, quando si devono leggere i paddle, basta fare un SYS alla vostra routine. NOTA: si deve fare il SYS e leggere il paddle sulla stessa riga altrimenti il controllore della tastiera verrà riattivato. La seguente routine legge i quattro valori dei paddle e li assegna alle variabili A, B, C e D:

```
10 PRINT "■"
20 POKE 8000, 120 : POKE 8001, 96
30 SYS 8000 : POKE 56320, 64 :
A=PEEK(54297) : B=PEEK(54298)
40 SYS 8000 : POKE 56320, 128 :
C=PEEK(54297) : D=PEEK(54298)
```

Il pulsante di fuoco può essere letto dalle posizioni di memoria 56320 e 56321. Aggiungendo la seguente routine, le variabili FA, FB, FC e FD indicheranno se i pulsanti di fuoco sono premuti. Uno 0 indica che il pulsante è premuto.

```
70 FA = SGN(PEEK(56320) AND 4)
80 FB = SGN(PEEK(56320) AND 8)
90 FC = SGN(PEEK(56321) AND 4)
100 FD = SGN(PEEK(56321) AND 8)
```

5.3. Input da tastiera con GET

Nel capitolo 3 si sono già introdotte le istruzioni GET e INPUT. Molti degli esempi esposti sino ad ora hanno fatto uso di INPUT, ma erano tutti programmi del tipo "stop-and-go", che quando hanno bisogno di un'immissione da tastiera si fermano ed aspettano il dato. Volendo scrivere programmi d'azione che ricevano istruzioni da tastiera, si può usare GET.
Come INPUT anche GET legge le informazioni, ma la somiglianza finisce qui. Ecco le principali differenze:

1. INPUT legge uno o più numeri completi o stringhe, mentre GET legge soltanto un singolo tasto.
2. Usando INPUT il programma attende che venga premuto RETURN. Se non viene immesso nulla il programma attende indefinitamente. GET invece non aspetta mai; il programma indica se non viene premuto nulla, ma continua ad eseguire.
3. Quando si inserisce qualcosa come risposta ad un INPUT, il "?" stimola la risposta che viene poi presentata sul video. GET non dà nessun segnale e non fa apparire la risposta sul video.

In altre parole con un'istruzione GET un programma è in grado di determinare se un tasto è stato premuto, ma non attende in caso contrario. Il computer può capire se la persona che sta usando il programma non fa nulla ed agire di conseguenza.

SINTASSI DELL'ISTRUZIONE GET

La sintassi dell'istruzione GET è semplice:

GET *nome di variabile*

Non esistono altre opzioni: deve esserci solamente una variabile. Al contrario di INPUT non è permessa alcuna stringa di suggerimento. Si può, tuttavia, presentare facilmente un suggerimento con un'istruzione PRINT che termini con un ";" in modo da impedire al C-64 di eseguire un ritorno a capo.

```
10 PRINT "QUESTO E' UN SUGGERIMENTO:";
20 GET A$
```

La variabile usata con GET può essere di qualsiasi tipo (intera, a virgola mobile o a stringa), ma quelle a stringa funzionano meglio, per due ragioni:

1. Se viene usata una variabile numerica, il BASIC cerca di interpretare ogni tasto premuto come un numero. Caricando qualsiasi cosa che non sia un numero, si provoca un errore di sintassi e il programma si arresta.
2. Se non viene premuto alcun tasto, alla variabile numerica viene assegnato un valore 0; il programma non ha modo di distinguere se non è stato premuto nulla o se è stato caricato apposta uno 0.

Con una variabile a stringa si può caricare con GET quasi qualunque tasto, inclusi quelli di comando del cursore e RETURN. (I tasti STOP e RESTORE non possono essere letti e i vari SHIFT e CTRL funzionano normalmente). Se non viene premuto nessun tasto, GET assegna una stringa vuota alla variabile ed il programma può riconoscerlo. Volendo aspettare fino a che non viene caricato qualcosa, usate una riga del tipo:

```
10 GET A$ : IF A$ = "" THEN 10
```

Si noti che non vi è alcuno spazio tra le virgolette, il che significa una stringa vuota. Se la variabile che è stata caricata da GET è uguale a questa stringa vuota, vuol dire che non è stato premuto alcun tasto.

VISUALIZZAZIONE DI TASTI PREMUTI

Come già detto, i caratteri caricati da GET non compaiono sullo schermo. A volte, però, è necessario vedere cosa si è caricato: e ciò è possibile aggiungendo un'istruzione PRINT al programma.

```
10 GET A$ : IF A$ = "" THEN 10
20 PRINT A$;
30 GOTO 10
```

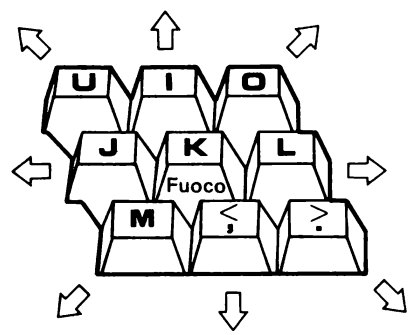
Il semplice programma riportato qui sopra farà apparire sullo schermo esattamente quello che è stato caricato dalla tastiera. Per interromperlo premete STOP.

LA TASTIERA COME JOYSTICK

Questa sezione viene presentata come un esempio; verranno mostrati per prima cosa alcuni dei passaggi adottati nel progettare il programma, poi verranno spiegati sia il listato che il procedimento del programma stesso. Il finto joystick si comporterà esattamente come quelli veri e gli verranno indicate le varie direzioni (sinistra, destra, su, giù, e in diagonale) premendo tasti diversi. Ci sarà inoltre un pulsante di fuoco, anche se non potrà essere premuto contemporaneamente a un altro tasto.

Scelta dei tasti

Per prima cosa si devono scegliere i nove tasti necessari a simulare un joystick, otto per le direzioni di movimento e uno per il pulsante di fuoco, che devono essere sistemati in modo da essere comodi da usare. La figura sottostante mostra una scelta di tasti facilmente controllabili dalla mano destra mentre si usa la tastiera.



Progetto dell'interfaccia con il programma principale

Studiamo ora la programmazione necessaria ad interpretare i tasti per simulare il joystick. La subroutine dovrebbe essere compatibile con quella scritta per il joystick vero, in modo che si possano scrivere programmi

in grado di utilizzare sia l'una che l'altro. Questa subroutine deve agire sulle stesse tre variabili del programma principale.

XI	è l'incremento X; controlla il movimento a destra e a sinistra.
YI	è l'incremento Y; controlla il movimento in su e in giù.
FB	segnala quando viene premuto il pulsante di fuoco.

XI è fissato a −1 per la sinistra, a +1 per la destra, e a 0 per nessun movimento. YI è fissato a −1 per il basso, a +1 per l'alto e a 0 per nessun movimento. Premendo il tasto K (fuoco) FB è uguale a 1, altrimenti resta 0.

Caricamento delle tabelle

Il programma interpreta i tasti di movimento cercandoli in una tabella e ne usa delle altre per i valori di XI e YI; tutte sono memorizzate come array. Il primo array (KT$) contiene i valori dei tasti che formano il joystick, gli altri due i valori di XI e YI che corrispondono a quei tasti. Per esempio, se KT$(5) contiene la lettera "U" che significa "in alto a sinistra", allora XT$(5) conterrà −1 e YT(5) conterrà +1. Per risparmiare tempo nell'esaminare la tastiera, queste tabelle vengono inizializzate in una subroutine separata che viene eseguita una volta sola all'inizio del programma.

```
62000 REM TABELLA VALORI TASTI
62100 DATA I,O,L,".",",",M,J,U
62200 REM VALORE DI X
62300 DATA 0,1,1,1,0,-1,-1,-1
62400 REM VALORE DI Y
62500 DATA 1,1,0,-1,-1,-1,0,1
62600 FOR I=0 TO 7: READ KT$(I): NEXT
62700 FOR I=0 TO 7: READ XT(I): NEXT
62800 FOR I=0 TO 7: READ YT(I): NEXT
62900 POKE 650,128
```

Il programma deve chiamare questa subroutine prima di usare il finto joystick, per poter tradurre i tasti correttamente. Nell'uso di questa subroutine fate attenzione a dove mettere le istruzioni DATA. Ricordatevi che l'istruzione READ inizia con il primo DATA del programma. Inserendo istruzioni DATA addizionali nel programma, potrebbe essere utile separare quelle qui sopra dalla subroutine e raggrupparle insieme a quelle del programma principale per mantenerle tutte nell'ordine giusto.

Subroutine di interpretazione della tastiera

Alla subroutine-interprete attribuiamo dei numeri di riga elevati per assicurarci che venga posta alla fine del programma.

```
63000 XI=0: YI=0: FB=0: LI=0
63010 GET KE$
63020 IF KE$ = "" THEN RETURN
63030 IF KE$ = KT$(LI) THEN YI = YT(LI):
      XI = XT(LI):RETURN
63040 LI = LI+1: IF LI < 8 THEN 63030
63050 IF KE$ = "K" THEN FB = 1
63060 RETURN
```

Questa subroutine ha qualche passaggio difficile e viene quindi spiegata riga per riga.

63000 XI, YI e FB sono azzerate. Siccome vi sono tre possibilità di RETURN, è necessario cancellare i valori preesistenti prima di cominciare.

63020 Controlla che un tasto sia stato premuto; in caso contrario esegue un RETURN, lasciando tutte le variabili uguali a 0.

63030
63040 Queste righe formano un loop per esaminare la tabella dei tasti e riconoscere quale tasto è stato premuto. Non viene utilizzato un loop FOR-NEXT, dato che si vuole eseguire il RETURN non appena è stato trovato un tasto. (Ricordatevi che si deve sempre terminare un loop FOR con un NEXT e che noi invece vogliamo fermare questo loop non appena possibile.)

63050 Controlla l'uso del pulsante di fuoco.

63060 RETURN al programma principale.

Uso del joystick in tastiera

Il vostro programma deve essere caricato a monte della riga 62000. Si può eseguire un SAVE per mantenere separatamente una copia di queste routine e farne uso come base di partenza per costruire altri programmi. Per rendervi conto di cosa si può ottenere con queste routine di "joystick in tastiera", provate ad usarle con il programma di dimostrazione che segue.

```
10 GOSUB 62000
20 A$(0)=" LEFT": A$(1)="      ": A$(2)="RIGHT"
30 B$(0)="DOWN ": B$(1)="      ": B$(2)="UP    "
40 C$(0)="FIRE": C$(1)="■FIRE■"
50 POKE 650, 128
100 PRINT "□"
110 PRINT "■";A$(XI+1);"/";B$(YI+1);SPC(5);C$(FB)
120 GOSUB 63000
130 GOTO 110
62000 REM TABELLA VALORI TASTI
62100 DATA I,O,L,".",",",M,J,U
62200 REM VALORE DI X
62300 DATA 0,1,1,1,0,-1,-1,-1
62400 REM VALORE DI Y
62500 DATA 1,1,0,-1,-1,-1,0,1
62600 FOR I=0 TO 7: READ KT$(I): NEXT
62700 FOR I=0 TO 7: READ XT(I): NEXT
62800 FOR I=0 TO 7: READ YT(I): NEXT
63000 XI=0: YI=0: FB=0: LI=0
63010 GET KE$
63020 IF KE$ = "" THEN RETURN
63030 IF KE$ = KT$(LI) THEN YI = YT(LI): XI = XT(LI):RETURN
63040 LI = LI+1: IF LI < 8 THEN 63030
63050 IF KE$ = "K" THEN FB =1
63060 RETURN
```

Capitolo

La grafica 6

In questo libro il termine "grafica" significa visualizzazione sullo schermo di immagini, testo, dati o programmi. L'"immagine" potrebbe essere un viso, un disegno d'architettura, una sagoma geometrica o semplicemente un assieme di caratteri.
Il C-64 ha notevoli capacità grafiche: alcune di esse, come i caratteri grafici standard e la possibilità di visualizzare a otto o sedici colori, sono già state menzionate nei capitoli precedenti. In questa parte si esamineranno in maggior dettaglio queste caratteristiche e si descriveranno le altre capacità grafiche del C-64, inclusa la possibilità di usare caratteri disegnati dall'utente. A mano a mano si spiegheranno le tecniche di programmazione per produrre videate colorate ed animate.

LA SCHEDA D'INTERFACCIA VIDEO

La scheda d'interfaccia video 6566 (VIC-II) è un circuito integrato complesso che invia al televisore l'immagine e i suoni prodotti dal C-64. (Si chiama VIC-II perché è una versione migliorata della scheda VIC originale usata dal Commodore VIC 20).
Essa funge da interfaccia con il video: traduce i segnali dal computer in segnali che producono un'immagine sul televisore. I programmi comunicano con la VIC-II per mezzo di istruzioni PEEK e POKE. L'abilità di produrre immagini grafiche sul C-64 consiste proprio nel sapere cosa fare col POKE e dove eseguire un PEEK.

LO SCHERMO DEL C-64

Esaminate ancora una volta lo schermo, che è la tela per l'esecuzione artistica dei disegni. Nella figura 6.1 è rappresentato uno schermo vuoto. Notate che è diviso in due aree, la cornice e lo sfondo. La cornice inquadra lo sfondo, riempiendo lo spazio tra esso e il bordo dello schermo; lo sfondo è l'area di lavoro, dove vengono presentati sia le immagini che il testo creati dai vostri programmi. Esso consiste di 25 righe di 40 caratteri ciascuna, come mostra la figura 6.2. Notate che le righe sono numerate da 0 a 24, e le colonne da 0 a 39. Le formule per manipolare i dati sullo schermo che verranno proposte in questo capitolo saranno molto più semplici da capire se si ha ben chiaro che il video inizia a riga 0 e colonna 0 invece che a riga 1 e colonna 1.

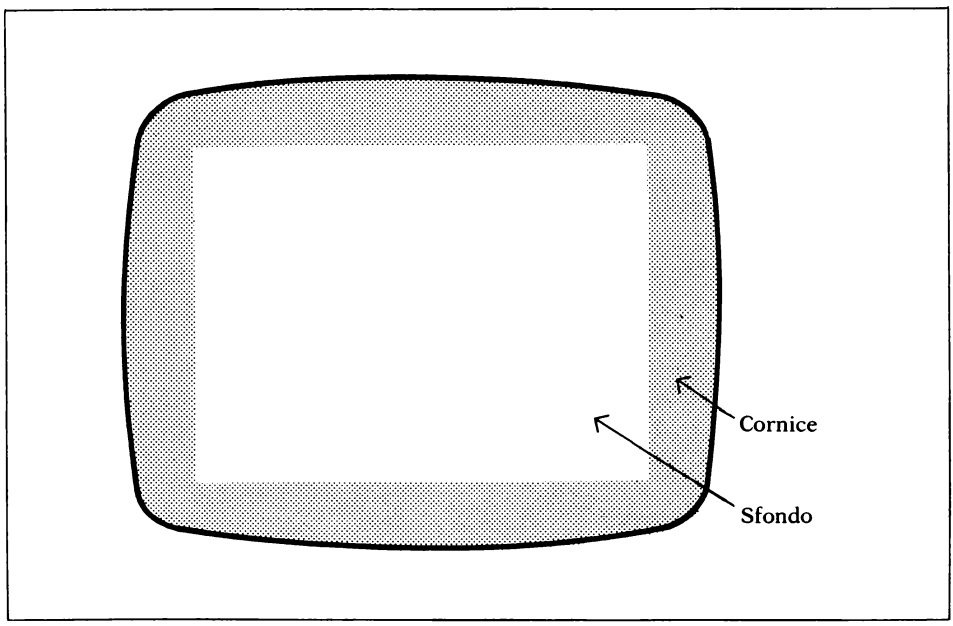


Figura 6.1. Lo schermo del C-64

COLORI DELLA CORNICE, DELLO SFONDO E DEI CARATTERI

I colori della cornice, dello sfondo e dei caratteri possono essere assegnati singolarmente. Quando il C-64 è messo in funzione (o riattivato con i tasti RUN/STOP e RESTORE), lo sfondo è blu scuro mentre la cornice e i ca-

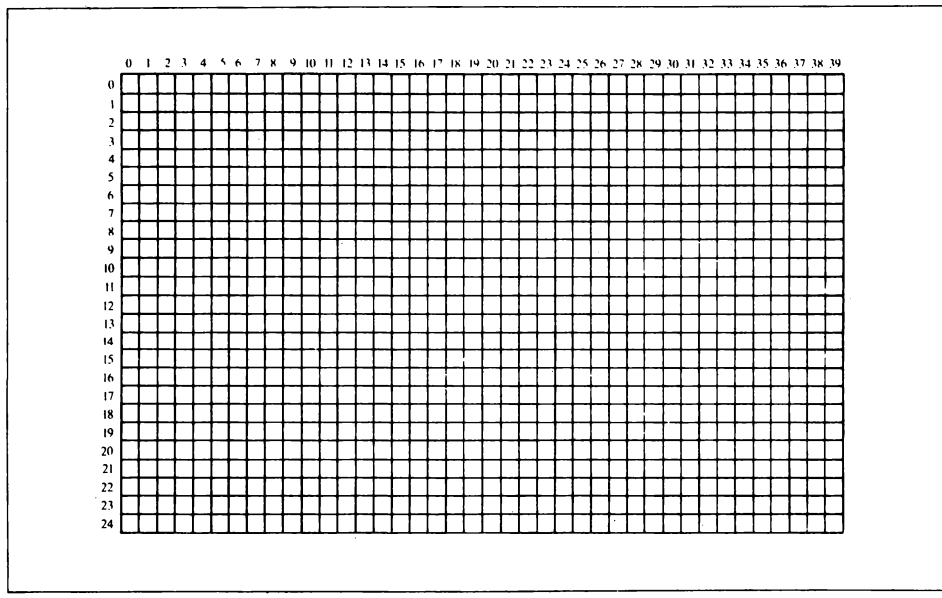


Figura 6.2. Lo schermo diviso in righe e colonne

ratteri sono azzurri. Questi colori si possono cambiare in ogni momento. I colori della cornice e dello sfondo sono memorizzati dalla scheda di interfaccia e possono essere cambiati semplicemente da un'istruzione POKE che carica un numero in una locazione di memoria: il colore della cornice è controllato dalla cella 53280, mentre lo sfondo dalla 53281. Nell'appendice E è fornita una tabella dei valori POKE per tutti i colori possibili. Ogni carattere ha la propria locazione in un'area di memoria chiamata *memoria del colore*. Più avanti in questo capitolo verrà spiegato come accedere a questa memoria del colore direttamente, ma è possibile far sì che se ne occupi il C-64. Sulla parte anteriore dei tasti numerici dall'1 all'8 sono riportati dei comandi per gli otto colori principali.

Tasto	*Abbreviazione*	*Colore*
1	BLK	Nero/Black
2	WHT	Bianco/White
3	RED	Rosso/Red
4	CYN	Cyan
5	PUR	Viola/Purple
6	GRN	Verde/Green
7	BLU	Blu/Blue
8	YEL	Giallo/Yellow

Per cambiare il colore dei caratteri, premete e mantenete premuto il tasto CTRL, poi il tasto del colore desiderato: tutti i caratteri generati dalla tastiera appariranno in quel colore. Saranno modificati solo quelli stampati dopo che è stato selezionato un nuovo colore, non quelli già sul video. Ci sono altri otto colori tra cui scegliere, non marcati sulla tastiera, selezionabili usando il tasto Commodore.

Tasto	*Colore*
1	Arancio/Orange
2	Marrone/Brown
3	Rosso chiaro/Light Red
4	Grigio scuro/Dark Gray
5	Grigio medio/Medium Gray
6	Verde chiaro/Light Green
7	Blu chiaro/Light Blue
8	Grigio chiaro/Light Gray

Per passare ad uno di questi colori, mantenete premuto il tasto Commodore, poi premete il tasto del colore desiderato. Esattamente come per i colori principali, saranno del nuovo colore solo i caratteri che appaiono dopo il cambiamento.

GIOCATORI E CAMPI-GIOCO

Molti programmi grafici, specialmente i giochi e le simulazioni, muovono uno o più oggetti su di uno sfondo immobile. Per evitare confusione con altri termini ci si riferirà agli oggetti come ai *giocatori* e allo sfondo come al *campo-gioco*. Il C-64 offre una varietà di modi per costruire giocatori e campi-gioco.
Non tutti i giocatori devono muoversi, e i campi possono cambiare; di fatto, in alcune applicazioni, i giocatori rimangono nello stesso posto e sono i campi a muoversi, proprio come nei vecchi film dove la diligenza rimaneva ferma e la scenografia dipinta veniva spostata dietro ad essa.

6.1. Grafica con i caratteri standard

Si possono creare giocatori semplici usando i caratteri grafici del C-64 stesso, ovvero i simboli e le sagome stampate sulla parte anteriore dei tasti. Per esempio, il seguente piccolo programma presenta sullo schermo il disegno di un'automobile:

```
100 PRINT "♥"
200 PRINT " /‾\_  "
300 PRINT "/    \ "
400 PRINT "L____J "
500 PRINT " O  O  "
```

Uno dei vantaggi delle presentazioni con istruzioni PRINT è che si possono fare delle prove sullo schermo in modo diretto fino a che si è soddisfatti del risultato, per poi costruirvi attorno il programma. Questo è il modo in cui si costruisce l'auto nel programma sopra descritto.
Studiate ora riga per riga la procedura di un semplice programma che traccia il disegno di un rombo:

Fase 1: Cancellate lo schermo usando i tasti SHIFT e CRL/HOME (Figura 6.3a).

Fase 2: Disegnate la parte superiore del rombo: battete N con SHIFT, poi M con SHIFT (Figura 6.3b). Non premete RETURN altrimenti il C-64 cercherà di eseguire ciò che è appena stato caricato e stamperà un messaggio READY (se si batte qualunque carattere non grafico, si potrebbe anche ottenere un messaggio ?SYNTAX ERROR). Per evitare ciò, battete RETURN con SHIFT. Il cursore si sposterà all'inizio della riga successiva, e il C-64 non cercherà di eseguire la figura.

Fase 3: Disegnate la parte inferiore del rombo. Usate RETURN con SHIFT per arrivare alla riga successiva. Battete M con SHIFT e N con SHIFT per completare il rombo (Figura 6.3c).

Fase 4: Mandate il cursore a home: premete il tasto CLR/HOME senza SHIFT (Figura 6.3d).

Fase 5: A questo punto siete soddisfatti del disegno e volete creargli attorno l'istruzione PRINT.
Caricate quattro spazi con il tasto INST/DEL. Questo lascia posto per il numero di riga, per "?" al posto di PRINT e le virgolette (") per iniziare una stringa (Figura 6.3e).

Fase 6: Caricate l'istruzione PRINT: il numero di riga (1∅), un punto interrogativo e le virgolette (Figura 6.3f).

Fase 7: Premete RETURN, senza SHIFT. Il BASIC memorizza la prima riga del rombo come riga di programma. Ripetete le fasi da 5 a 7 per la seconda riga del disegno, questa volta usando il numero di riga 2∅.

Si può usare questo metodo per riprodurre praticamente qualsiasi cosa si possa abbozzare sullo schermo, da un semplice quadrato ad un disegno complesso. L'unica restrizione è che non è possibile riempire completamente lo schermo per lasciare lo spazio necessario alle istruzioni PRINT.

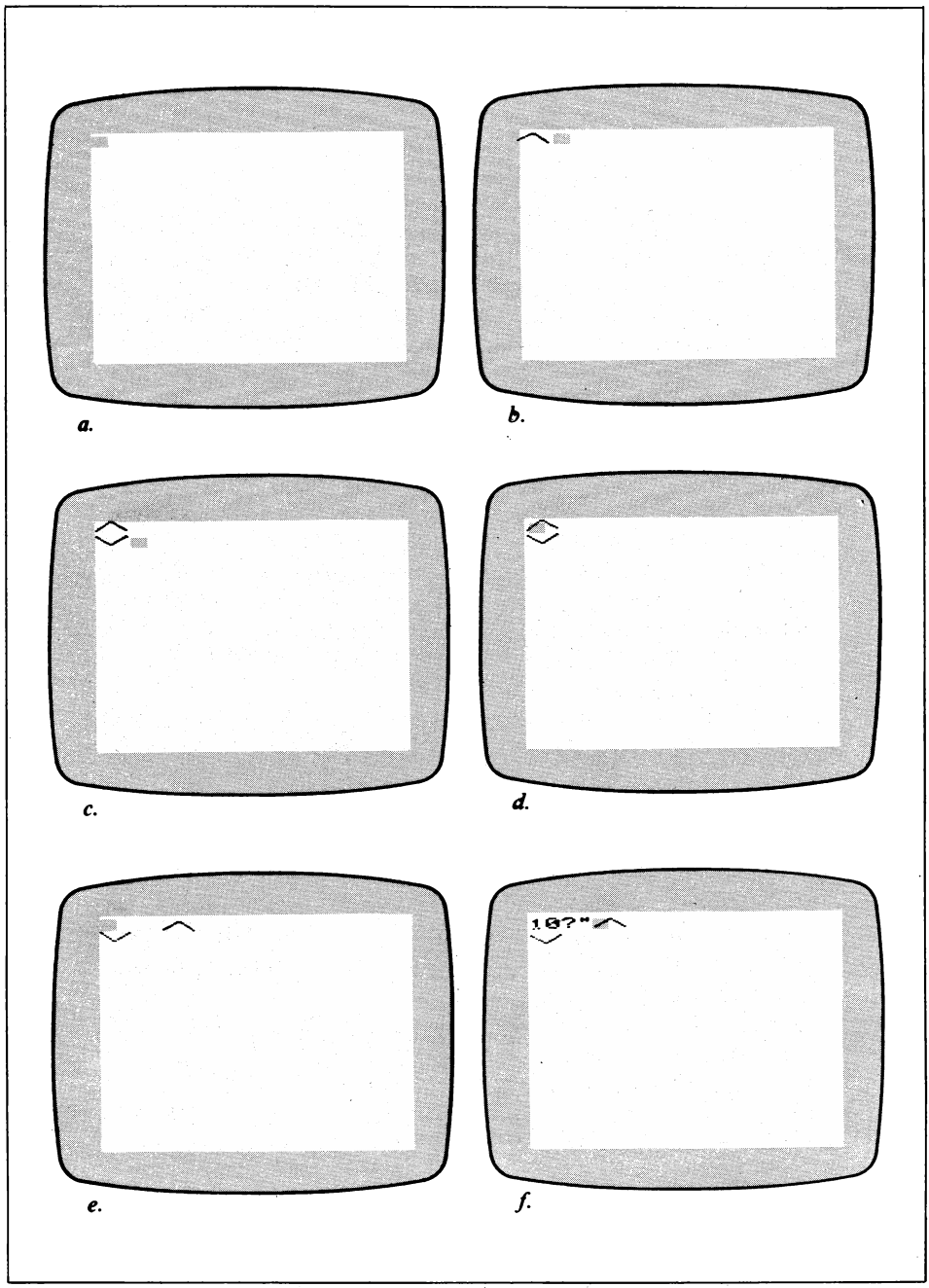


Figura 6.3. Tracciare un rombo

Ora sperimentate qualche disegno di vostra scelta. Per il momento, evitate i caratteri inversi, che richiedono alcune precauzioni speciali che verranno trattate nella prossima sezione.

USO DI CARATTERI INVERSI

Si sarà già notato che i caratteri inversi hanno un uso speciale nelle istruzioni PRINT: essi rappresentano i tasti, come HOME, i comandi del cursore e i tasti del colore che non generano caratteri "normali". Caricando un carattere inverso direttamente in un'istruzione PRINT, si verificherà una delle seguenti alternative:

1. Il carattere verrà interpretato come un tasto speciale per muovere il cursore, cambiare il colore e così via.
2. Il carattere non si accoppierà con nessuno dei tasti speciali; sarà memorizzato come un carattere non inverso quando si preme RETURN. Ciò si verificherà anche se il carattere appare nel programma tra virgolette.

Per stampare un singolo carattere inverso all'interno di una stringa, caricate un RVS ON (CTRL-9) prima del carattere inverso. Il carattere stesso deve comparire nell'istruzione PRINT come carattere "normale" (non inverso). Si può inserire il RVS ON usando il tasto INST se si costruiscono istruzioni PRINT da un abbozzo sullo schermo. Ricordatevi che premendo il tasto INST ci si immette nel modo tra virgolette per un carattere solamente, per cui il RVS ON verrà memorizzato come parte dell'istruzione PRINT invece di essere eseguito. Il RVS ON è annullato automaticamente alla fine della riga PRINT. Ricordatevi inoltre che un'istruzione PRINT può occupare più di una riga sul video: il RETURN ne indica la fine.
Se un'istruzione PRINT termina con un ";", il BASIC non aggiunge un ritorno a capo e i caratteri continueranno ad essere stampati in "inverso". Se i caratteri normali seguono quelli inversi sulla riga PRINT, inserite un RVS OFF (CTRL-0) davanti al primo carattere non inverso. Per illustrare il problema e la sua soluzione, modifichiamo il programma del rombo per produrre una figura piena invece di un contorno. Seguite le fasi rappresentate nella figura 6.3, ma usate caratteri diversi per costruire il giocatore.

— Nella fase 2, dopo aver pulito il video, battete RVS ON (CTRL-9), £ con SHIFT e * con C= .
— Nella fase 3, battete * con C= e £ con SHIFT.

Quando si lista il programma, si noterà che il primo £ con SHIFT è tornato normale, mentre * con C= no.

```
10 PRINT "◤◣"
20 PRINT "▼"
```

Ora fate eseguire il programma ed osservate l'effetto di * con C= . La seconda riga è diventata azzurra (cyan).
Per modificare il programma in modo da far apparire il progetto originario, seguite le seguenti fasi: listate il programma sul video; posizionate il cursore sopra a £ con SHIFT a riga 1Ø e premete INST con SHIFT. Questo inserisce uno spazio e mette il C-64 temporaneamente in modo tra virgolette. Battete ora CTRL-9 (RVS ON). Dato che è nel modo tra virgolette, il C-64 inserirà il RVS ON nell'istruzione PRINT (presentato come una "R" inversa), invece di passare ai caratteri inversi. Quando l'istruzione PRINT viene eseguita, la "R" inversa verrà riconosciuta e il C-64 comincerà a far apparire i caratteri inversi.
Cambiate *con C= inverso in uno normale, in modo che non farà diventare lo schermo di nuovo azzurro. Muovete il cursore e battete un * con C= sopra a quello preesistente. Ora il programma dovrebbe essere come segue:

```
10 PRINT "◤◣"
20 PRINT "▼"
```

Eseguite di nuovo il programma, e vedrete che il rombo è ora presentato correttamente.

AGGIUNGERE IL COLORE

Come i caratteri inversi, le presentazioni colorate necessitano di particolare attenzione. Il problema ora è differente: il C-64 dimentica il colore di un carattere ogni volta che non è più sullo schermo, perché il colore non fa parte del carattere. Abbiamo detto che vi è un'area appositamente riservata per memorizzare i colori dei caratteri sullo schermo: quando si lista il programma, il C-64 fissa il colore di ogni carattere in base a quello attivo al momento; quando viene acceso usa il blu. Abbiamo spiegato come cambiare il colore dalla tastiera usando il tasto CTRL. Come si vedrà ora, anche un programma è in grado di cambiarlo. Una volta che un carattere non è più sullo schermo, la memoria del colore è riutilizzata per segnalare il colore del carattere che lo sostituisce: solo il carattere stesso è memorizzato come parte integrante del programma. Come per i caratteri inversi, questo problema è risolto caricando nel programma dei caratteri di comando per attribuire i colori.
Per esempio, per far diventare il rombo rosso invece di blu, battete un

CTRL-3 nella riga 1Ø dopo il RVS ON (R inversa) in modo che la riga diventi così:

```
10 PRINT "■■■■"
20 PRINT "■■"
```

Ora fate eseguire il programma. Notate che l'intero rombo è rosso, non solamente la prima riga. I comandi dei colori, al contrario del comando inverso, non sono riassegnati quando si inizia una nuova riga; essi rimangono attivi fino a che non viene assegnato un nuovo colore, o fino a che si reinizializza il C-64 con i tasti RUN/STOP e RESTORE.

6.2. Creare immagini con POKE

A volte non è pratico usare PRINT per costruire un'immagine grafica sul video. Per esempio, se diversi giocatori si muovono sullo schermo, può essere necessario determinare quando entrano in collisione tra di loro o con un oggetto sul campo-gioco. Oppure potrebbe essere necessario avere sullo schermo il cursore quando si caricano dei caratteri con GET. Per applicazioni come queste si dovrà accedere al video direttamente. Il C-64 permette quest'operazione tramite PEEK e POKE.

LA MEMORIA DEL VIDEO

I caratteri presentati sullo schermo sono memorizzati nella memoria del C-64. La scheda VIC-II legge l'immagine di un carattere dalla memoria così come viene visualizzato. I caratteri sono memorizzati in un'area chiamata "*memoria del video*", e i colori dei caratteri sono memorizzati nella "*memoria del colore*". Si può accedere a queste aree usando PEEK e POKE per esaminare e cambiare la videata.

Ubicazione della memoria del video

Nella maggior parte dei computer la locazione della memoria del video è parte dell'*hardware* e non può essere cambiata. Nel C-64, benché il numero di righe e colonne sia fisso, non lo è il punto di partenza. La scheda VIC-II usa uno dei suoi registri interni per memorizzare l'inizio della memoria del video e questo puntatore può essere cambiato. Alcuni dei programmi che verranno illustrati più avanti in questo capitolo faranno uso di questa possibilità.

I programmi che eseguono dei PEEK e POKE direttamente nella memoria del video devono ovviamente conoscerne l'ubicazione: una locazione di memoria del BASIC indica al C-64 dove inizia la memoria del video; se si esegue un PEEK a questa locazione (cella 648), e si moltiplica il contenuto per 256, si ottiene l'indirizzo dell'inizio della memoria.

```
10 SB=256*PEEK(648)
```

I nostri programmi di esempio terranno aggiornata la cella 648 quando spostano la memoria del video. Se questa convenzione verrà seguita, si potranno scrivere dei programmi funzionanti indipendentemente da dove sia localizzata la memoria del video.

Disposizione della memoria del video

I caratteri che si vedono sullo schermo sono memorizzati in un array di 25 righe e 40 colonne. Questa non è una variabile BASIC, è semplicemente un'area di 1000 byte di memoria a cui si può accedere con PEEK e POKE, ma è più chiaro rappresentarla come un array. Ogni elemento contiene un carattere del video: l'elemento (0,0) contiene il carattere in alto a sinistra, e l'elemento (24,39) contiene quello in basso a destra. Una volta localizzato l'inizio della memoria del video, usando la formula della precedente sezione, si può rapidamente calcolare dove fare un POKE a un particolare carattere. La formula è:

locazione POKE = inizio del video+ colonna +40*riga

Per poter usare questa formula, si devono numerare le colonne video da 0 a 39, e le righe da 0 a 24. Come esempio di questa formula, caricate il seguente programma:

```
90 POKE 53281,6
100 SB=256*PEEK(648)
110 REM RIEMPIE LA MEMORIA DEL VIDEO CON "@"
120 FOR I=0 TO 999
124 POKE55296+I,3
125 IF PEEK(SB+I)=32THENPOKE55296+I,6
130 POKE SB+I,0
140 NEXT I
150 REM ASPETTA CHE VENGA PREMUTA LA BARRA
160 GET X$
170 IF X$<>" "THEN 160
180 REM CAMBIA IL FONDO IN NERO
190 POKE 53281,0
```

Prima di far eseguire il programma, svuotate lo schermo e listate il programma, lasciando il listato sullo schermo. Ora fate eseguire il programma e osservate il risultato; tutti i caratteri sullo schermo, eccetto gli spazi, vengono mutati in @. (Di fatto il programma ha mutato tutti i caratteri in @, ma quelli che compaiono sullo schermo come degli spazi sono invisibili). Quando il BASIC svuota lo schermo, muta il colore di tutti i caratteri presenti nello stesso colore dello sfondo; quando poi viene stampato ogni carattere il C-64 fissa la sua locazione nella memoria del colore. Dato che alcune delle @ sono presentate sullo schermo in blu su uno sfondo blu, sono impossibili da vedere. Per farle diventare visibili premete la barra della spaziatura. Il programma trasformerà il colore dello sfondo in nero, e compariranno le @ blu.
Si deve tener a mente questo problema dei caratteri che spariscono quando si usa POKE in una videata. Il programma deve assicurarsi che il colore della cella nella quale si sta eseguendo un POKE sia fissato correttamente. La sezione seguente descrive come fare.

LA MEMORIA DEL COLORE

Si è già accennato che i colori dei caratteri sul video sono memorizzati in una zona o area speciale chiamata memoria del colore. Come dimostra l'ultimo esempio, è essenziale capire come usare la memoria del colore quando si usa un POKE nelle visualizzazioni. Quindi, prima di procedere oltre con i POKE della memoria del video, studiamo brevemente la memoria del colore.

Localizzazione della memoria del colore

Al contrario della memoria del video, quella del colore non si muove mai e rimane sempre allocata nelle celle da 55296 fino a 56319. Come la memoria del video anche quella del colore ha una locazione per ogni carattere del video; l'ordine dei colori dei caratteri è lo stesso dei caratteri, quindi la formula usata per localizzare il colore di un carattere è come quella usata per localizzare il carattere stesso sul video.

Locazione di memoria del colore = 55296+colonna+40*riga.

Contenuto della memoria del colore

Il colore di ogni carattere è memorizzato in forma di un numero da 0 a 15:

0	Nero	8	Arancio
1	Bianco	9	Marrone
2	Rosso	10	Rosso chiaro
3	Cyan	11	Grigio scuro
4	Viola	12	Grigio medio
5	Verde	13	Verde chiaro
6	Blu	14	Blu chiaro
7	Giallo	15	Grigio chiaro

Contrariamente ad altre aree della memoria del C-64, quella del colore usa solo quattro bit per cella: ne servono infatti solo quattro per memorizzare i numeri da 0 a 15. Dato che i quattro bit superiori non sono utilizzati, la Commodore non ha previsto una memoria per contenerli: se si cerca di fare un POKE a un numero più grande di 15 nella memoria del colore, la parte contenuta nei bit "mancanti" verrà perduta.

10101010	Rappresenta il valore binario di 204:
	quando sono persi i quattro bit superiori
1010	viene memorizzato come 12.
11111111	Nello stesso modo 255
1111	diventa 15.
00000111	Il numero 7, invece,
0111	non viene cambiato.

Dato che non vi è nessuna scheda di memoria per fornire i quattro bit superiori quando si legge nella memoria del colore, essi assumeranno valori imprevedibili. Ricordatevi che un bit deve essere 0 o 1; anche se non vi è alcun segnale in input, la scheda deve assegnargli un valore; senza un segnale proveniente dai quattro bit superiori essa assegna arbitrariamente 1 o 0. Quando si fa un PEEK nella memoria del colore, bisogna sempre ignorare i bit inesistenti, usando AND per mascherarli come spiegato nel capitolo 5:

```
10 BY=PEEK(55400) AND 15
```

CODICI DI VISUALIZZAZIONE SULLO SCHERMO

Una volta che si è capito dove fare un POKE per trasformare una videata, si deve sapere quale valore metterci. I computer Commodore non usa-

no per la memoria del video gli stessi codici di carattere che usano nei programmi. La maggior parte dei computer memorizzano i caratteri in un codice standard chiamato ASCII (American Standard Code for Information Interchange) mentre i computer Commodore usano un ASCII "esteso" per tutti gli scopi ad eccezione della rappresentazione dei caratteri nella memoria del video.
Le "estensioni" consistono in alcuni caratteri di comando non usati da altri computer e nei caratteri grafici. A molti dei codici di comando non corrisponde un carattere presentabile sul video; per rendere disponibile il maggior numero possibile di codici di caratteri grafici, la Commodore ha ideato un codice diverso per la memoria del video che elimina alcuni caratteri ASCII e cambia il valore di altri. L'appendice E contiene una tabella dei codici video. Se il vostro programma sta usando un POKE con dei caratteri che sono stabiliti al momento in cui lo si scrive, si può semplicemente consultare la tabella. Se non si conoscono in anticipo i caratteri che verranno messi con POKE, il programma dovrà trasformarli dal codice ASCII in quello video. Ciò potrebbe essere necessario se il programma usa un GET per leggere le istruzioni dalla tastiera, e vuole presentarli in qualche posto particolare sullo schermo. La trasformazione in codice video è di facile applicazione perché i codici ASCII cambiati vengono spostati in blocchi di 32 caratteri. I cambiamenti sono elencati nella tabella 6.1.
Una semplice subroutine che preleva il valore di un tasto, KV$, e lo trasforma in codice video, SC, è esemplificata più avanti. Se il tasto non può essere tradotto (un tasto di comando del cursore, per esempio), la subroutine riporta un valore uguale a −1. Il programma principale può così distinguere la differenza tra un tasto visualizzabile e uno che non lo è.

Tabella 6.1. Conversione del codice ASCII in codice video

Codice ASCII	*Codice video*
0-31	Nessuno
32-63	32-63
64-95	0-31
96-127	64-95
128-129	Nessuno
160-191	96-127
192-254	64-126
255	94

```
60000 SC=ASC(KV$)
60010 IF SC<32 THEN SC=-1 :RETURN
60020 IF SC<64 THEN RETURN
60030 IF SC<96 THEN SC=SC-64:RETURN
60040 IF SC<128 THEN SC=SC-32:RETURN
60050 IF SC<160 THEN SC=128:RETURN
60060 IF SC<192 THEN SC=SC-64:RETURN
60070 IF SC<255 THEN SC=SC-128:RETURN
60080 SC=126:RETURN
```

IL PUNTO MOBILE

Nel capitolo 5 abbiamo presentato un programma che spostava un punto in giro per lo schermo a seguito di comandi forniti dal joystick. Qui invece si vede come potrebbe essere scritto quel programma usando POKE al posto di PRINT. Come prima questo programma non è completo: si deve aggiungere la subroutine adatta per usare la tastiera o il joystick come comando.

```
100 REM PULISCE LO SCHERMO
110 PRINT"■"
120 REM PREDISPONE LA MEMORIA DEL COLORE
130 FOR I=55296 TO 56295
140 POKE I,14
150 NEXT I
160 XP=0:YP=0
170 SB=PEEK(648)*256
180 REM LOOP PRINCIPALE
190 GOSUB 63000
200 IF (XI=0)AND(YI=0)AND(FB=0)THEN190
210 REM CANCELLA IL VECCHIO PUNTO MOBILE
220 POKE SB+XP+40*YP,32
230 REM CALCOLA LA NUOVA POSIZIONE DELLA X
240 XP=XP+XI
250 IF XP>39 THEN XP=0
260 IF XP<0 THEN XP=39
270 REM CALCOLA LA NUOVA POSIZIONE DELLA Y
280 YP=YP+YI
290 IF YP>24 THEN YP=0
300 IF YP<0 THEN YP=24
310 REM CREA IL NUOVO PUNTO MOBILE
320 POKE SB+XP+40*YP,81
330 GOTO 190
```

6.3. Animazione dei giocatori

Abbiamo già esaminato come spostare un giocatore sullo schermo. Vediamo ora come modificare il giocatore stesso, in modo che sembri muoversi. Per animare un giocatore vengono cambiati uno o più caratteri che lo compongono, dando l'illusione del movimento. Per esempio, questo programma produce un parabrezza con il tergicristallo in funzione:

```
10 DLY=150
20 REM DISEGNA UN PARABREZZA
30 PRINT"◻╭──────╮"
40 PRINT"│      │"
50 PRINT" ╰──────╯"
60 REM MOVIMENTO VERSO DESTRA
70 FOR I=1 TO 4
80 ON I GOSUB 180,240,300,360
90 FOR J=1 TO DLY : NEXT
100 NEXT
110 REM MOVIMENTO VERSO SINISTRA
120 FOR I=3 TO 2 STEP -1
130 ON I GOSUB 180,240,300,360
140 FOR J=1 TO DLY : NEXT
150 NEXT
160 GOTO 70
170 REM TERGICRISTALLI SULLE 10
180 POKE 1065,77
190 POKE 1066,32
200 POKE 1067,77
210 POKE 1068,32
220 RETURN
230 REM TERGICRISTALLI SULLE 12
240 POKE 1065,32
250 POKE 1066,101
260 POKE 1067,32
270 POKE 1068,101
280 RETURN
290 REM TERGICRISTALLI SULLE 2
300 POKE 1065,32
310 POKE 1066,78
320 POKE 1067,32
330 POKE 1068,78
340 RETURN
350 REM TERGICRISTALLI SULLE 3
360 POKE 1066,100
370 POKE 1068,100
380 RETURN
```

Il programma ha vari passaggi importanti, che esamineremo attentamente. Le righe da 3Ø a 5Ø cancellano lo schermo e fanno apparire il para-

brezza per mezzo di istruzioni PRINT. Quelle da 7Ø a 1ØØ fanno spazzolare da sinistra a destra il tergicristallo; esso poi torna da destra a sinistra con istruzioni 12Ø–15Ø. Questi loop cambiano la videata chiamando, a turno, subroutine che posizionano i caratteri in particolari punti, usando dei POKE. Ogni subroutine cancella il tergicristallo dal video e lo mette in una nuova posizione. Due delle righe più importanti sono la 9Ø e la 14Ø. I loop FOR-NEXT rallentano il programma; senza dei *loop di ritardo* infatti il cambiamento sarebbe troppo veloce e verrebbe ridotto ad una azione indistinta. La scelta dei loop di ritardo, specialmente per l'animazione che produce movimenti complessi, è molto delicata. Nello sviluppo di questi programmi ci si deve aspettare di passare molto tempo ad affinare i ritardi in modo da produrre un'animazione che si muova con scioltezza alla velocità desiderata. Nell'esempio la durata del ritardo è controllata dalla variabile DLY ed il suo valore è fissato a riga 1Ø. Provate con valori diversi: un valore più basso riduce il ritardo dei loop, facendo muovere il tergicristallo più rapidamente; aumentando i valori, aumenta la durata dei loop e il movimento rallenta.
È difficile fare animazioni complesse con la serie standard di caratteri a meno di non lavorare con un giocatore molto grosso. In molti casi si deve muovere parte del giocatore nella misura di un intero carattere per spostarlo effettivamente, rendendo il movimento piuttosto discontinuo, a meno che il giocatore non occupi gran parte dello schermo. Per produrre movimenti più sottili con giocatori piccoli, si devono disegnare caratteri propri: queste tecniche saranno discusse più avanti in questo capitolo.

L'USO COMBINATO DI PRINT E POKE NELLA GRAFICA

Come già visto, entrambi i metodi di PRINT e POKE hanno vantaggi e svantaggi nella realizzazione di immagini grafiche. Spesso la soluzione migliore per quei programmi con giocatori che si muovono su uno sfondo fisso è un abbinamento di PRINT e POKE. In generale, l'uso di istruzioni PRINT per far apparire simboli grafici sullo schermo in abbinamento con istruzioni POKE che saltano in qualsiasi punto del video, può facilitare notevolmente lo sviluppo di un programma.

CORSA SU PISTA

La figura 6.4 presenta il listato di un semplice gioco di corsa su pista che mostra alcune delle tecniche finora discusse in questo capitolo.
La versione originale con carta e matita di questo gioco si esegue su un

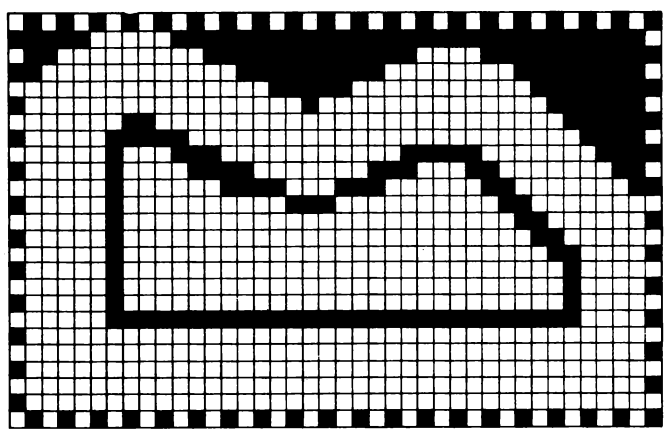
```
100 GOSUB 40000
200 CR=14
300 CC=2
356 POKE 53275,254
400 POKE SB+CC+CR*40,0
1000 IF XS<>0 AND TI>XT THEN CC=CC+SGN(XS):
     XT=TI+60/ABS(XS)
1100 IF YS<>0 AND TI>YT THEN CR=CR+SGN(YS):
     YT=TI+60/ABS(YS)
1200 IF CC<0 OR CC>39 THEN 4000
1300 IF CR<0 OR CR>24 THEN 4000
1400 POKE BA,32
1500 BA=SB+CC+CR*40
1600 TG=PEEK(BA)
1700 IF TG<>0 AND TG<>32 THEN 5500
1800 POKE BA,0
1900 IF TT>TI THEN 1000
2000 TT=TI+30
2100 POKE V2,129
2200 GOSUB 63000
2300 POKE V2,0
2400 XS=XS+XI : YS=YS+YI
2500 GOTO 1000
4000 IF CC<0 THEN CC=0
4100 IF CC>39 THEN CC=39
4200 IF CR<0 THEN CR=0
4300 IF CR>24 THEN CR=24
4400 BA=SB+CC+CR*40
5000 POKE BA,0
5100 FOR I=1 TO 350 : NEXT I
5200 POKE BA,32
5300 FOR I=1 TO 350 : NEXT I
5400 GET A$: IF A$<>"" THEN 100
5500 GOTO 5000
40000 PRINT"□";
40100 SB=256*PEEK(648)
40200 FOR I=55296 TO 56295:POKE I,14:NEXT I
40300 XS=0 : YS=0 : XT=0 : YT=0
40400 REM VARIABILI JOYSTICK
40500 JS=56321
40600 UM%=1 : DM%=2 : LM%=4 : RM%=8
40700 REM SUONA
40800 V2=54283
40900 POKE V2+1,80
41000 POKE 54295,0
41100 POKE 54296,15
```

(continua)

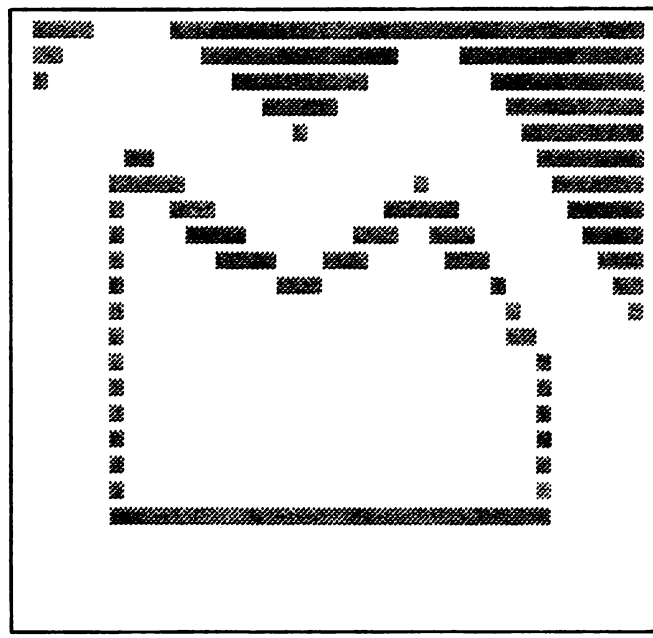
```
41200 PRINT"
41300 PRINT"
41400 PRINT"
41500 PRINT"
41600 PRINT"
41700 PRINT"
41800 PRINT"
41900 PRINT"
42000 PRINT"
42100 PRINT"
42200 PRINT"
42300 PRINT"
42400 PRINT"
42500 PRINT"
42600 PRINT"
42700 PRINT"
42800 PRINT"
42900 PRINT"
43000 PRINT"
43100 PRINT"                                                    ";
43200 RETURN
63000 SS%=PEEK(JS)
63010 XI=SGN(SS% AND LM%)-SGN(SS% AND RM%)
63020 YI=SGN(SS% AND UM%)-SGN(SS% AND DM%)
63030 RETURN
```

Figura 6.4.

foglio di carta a quadretti con il percorso tracciato. Ecco un tipico percorso:



Il gioco utilizza le coordinate che identificano un carattere sullo schermo per ogni quadratino sulla carta.



Le regole del gioco sono semplici.

1. L'auto si muove sia in orizzontale che in verticale ad una velocità di un certo numero di quadratini al secondo. Si può muovere lungo le due dimensioni contemporaneamente; ad esempio, per muoversi diagonalmente può spostarsi alla medesima velocità di un quadratino al secondo in verticale e in orizzontale.
2. Ad ogni turno (circa due volte al secondo) il giocatore può accelerare o rallentare di un quadratino al secondo in ogni dimensione. Per esempio, se un giocatore si sta muovendo ad una velocità di tre quadratini verso l'alto e uno a destra, può modificare la propria velocità in quattro verso l'alto, uno a destra; oppure due verso l'alto, due a destra; oppure ancora due verso l'alto, zero a destra e così via.
3. Se l'auto va fuori pista, si schianta e finisce il gioco.

Il programma di corsa su pista listato nella figura 6.4 è una versione piuttosto elementare del gioco, ma fornisce qualche esempio pratico delle tecniche di scrittura dei programmi di grafica e animazione. Esso usa PRINT per far apparire il campo-gioco che è stato sviluppato in origine

come abbozzo sul video. Contiene anche qualche esempio dell'uso di POKE per muovere il giocatore e di PEEK per rilevare le collisioni. Esaminiamo passo passo il programma iniziando dalle variabili principali.

Nome della variabile	*Descrizione*
CC	*Colonna auto* — numero di colonna del video dove si trova l'auto.
CR	*Riga auto* — riga del video dove si trova l'auto.
TT	*Controllo del tempo* — valore di TI usato per verificare se il joystick sta inviando dei comandi.
XS	*Velocità X* — velocità dell'auto nella dimensione X: negativa verso sinistra, positiva verso destra.
XT	*Tempo X* — valore di TI al momento di muoversi nella dimensione X.
YS	*Velocità Y* — velocità dell'auto nella dimensione Y: negativa verso l'alto, positiva verso il basso.
YT	*Tempo Y* — valore di TI al momento di muoversi nella dimensione Y.

Le righe da 1ØØØ a 25ØØ formano il loop principale di calcolo del programma, che ripete continuamente la serie di istruzioni aspettando comandi dall'utente. Ad ogni passaggio controlla il tempo per determinare se deve muovere l'auto (righe 1ØØØ e 11ØØ) oppure controllare il joystick (riga 16ØØ). Se qualcuna di queste azioni ha bisogno di essere eseguita, viene assegnato anche il valore del tempo per la successiva occasione. Alle righe 12ØØ e 13ØØ il programma controlla che l'auto non sia finita fuori dalla cornice del video; le righe 15ØØ e 16ØØ controllano che l'auto non stia per urtare il bordo della pista. Se si verifica una di queste condizioni il programma rimanda alla subroutine di "incidente" a riga 4ØØØ. Le righe 21ØØ e 23ØØ generano un "toc" ogni volta che il programma controlla il joystick. (L'uso del suono del C-64 verrà spiegato al capitolo 7). La routine di "incidente" (righe da 4ØØØ fino a 55ØØ) riporta l'auto sulla pista nell'eventualità che fosse uscita (righe da 4ØØØ a 44ØØ) e la fa lampeggiare (righe 5ØØØ-53ØØ). Appena l'utente preme un qualsiasi tasto, la riga 54ØØ fa ricominciare il gioco.
La corsa su pista è un esempio di tecniche grafiche e una base sulla quale potrete fare degli ulteriori esperimenti, tipo la modifica del tracciato o il controllo che l'auto abbia veramente fatto tutto il percorso invece di "tagliare" la strada, con l'aggiunta della visualizzazione dei secondi.

6.4. Segmenti di memoria

Le caratteristiche grafiche esaminate sino ad ora hanno utilizzato la memoria del video nelle locazioni 1024-2023. Le funzioni grafiche più avanzate fanno uso anche di altre posizioni; una delle caratteristiche della scheda VIC-II è quella di poter accedere soltanto a 16K (16384) byte di memoria e, dato che il C-64 ne ha 64K, sembrerebbe che la maggior parte di essa non possa venir utilizzata dall'interfaccia. Per superare questo problema la Commodore ha diviso la memoria del C-64 in quattro "segmenti", ognuno di 16K byte, e i programmi possono selezionare quale di questi segmenti debba essere usato dalla VIC-II. Dato che il primo segmento (selezionato dal C-64 appena acceso) è usato dal BASIC, appositi programmi faranno indirizzare la scheda ad altri segmenti. I metodi usati per scegliere il segmento giusto sono complessi e quindi sono stati lasciati al paragrafo "Grafica avanzata con la scheda VIC-II" alla fine del capitolo.
Vi sono due punti importanti che è necessario comprendere prima di procedere oltre. Per prima cosa, tutte le aree di memoria usate dall'interfaccia (la memoria del video e le altre di cui fra poco sapremo di più) devono essere nello stesso segmento di 16K; inoltre, ricordatevi che premere RUN/STOP-RESTORE non cancella completamente la memoria. La locazione 648 controlla il posizionamento della memoria dello schermo e RUN/STOP-RESTORE non la ripristina al valore che ha al momento dell'accensione.
È possibile che il BASIC, se qualcosa non dovesse funzionare in uno dei programmi di grafica, si confonda su dove trovare la memoria del video. In questo caso non basta premere RUN/STOP-RESTORE per ripartire, ma è necessario spegnere il C-64. Prima di far girare uno di questi programmi sarà quindi opportuno farne una copia. Questo tipo di problema non capita molto spesso, ma un programma che usa molti POKE può provocare un blocco, cambiando una locazione di memoria in maniera sbagliata.

6.5. I caratteri personali

Il C-64 è in grado di cambiare la forma dei caratteri con facilità. La maggior parte dei personal richiede cambiamenti di *hardware* per poter usare dei caratteri diversi, mentre il C-64 permette di creare una serie personalizzata di caratteri, da usare con un semplice POKE. Questa sezione mostrerà come si definisce e progetta una serie di caratteri personali.

COME APPAIONO I CARATTERI SULLO SCHERMO

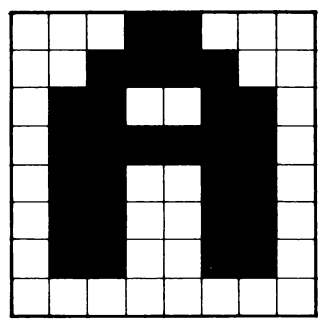
Per prima cosa parliamo brevemente del televisore. Osservando attentamente lo schermo noterete che ogni riga è formata da molti punti, circa 500. All'interno del tubo catodico un raggio di elettroni esplora avanti e indietro una riga dopo l'altra, illuminando un punto alla volta. Il segnale determina la luminosità e il colore di ogni punto luminoso. Sebbene il procedimento sia molto più complicato, basterà sapere che l'immagine è composta da righe di punti. La scheda VIC-II genera il segnale TV.

LA MEMORIA DEI CARATTERI

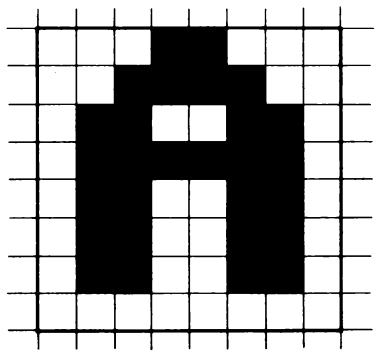
Esaminiamo ora come l'interfaccia controlla tutti i punti dello schermo per stampare un carattere. Ricorderete che i caratteri sono memorizzati nella memoria del video. Quando stampa, la VIC-II legge la memoria e acquisisce i caratteri uno alla volta; per determinare quali punti illuminare consulta una tabella che contiene queste informazioni e che è stata programmata e memorizzata, alla fabbricazione, in un apposito chip chiamato ROM (Read Only Memory: memoria di sola lettura), che contiene le caratteristiche di tutti i caratteri anche quando il computer è spento. L'area occupata da questa scheda si chiama memoria dei caratteri. Con un POKE un programma può ordinare all'interfaccia di usare un'area differente per la memoria dei caratteri, area che avrete caricato con una tabella di simboli da voi disegnati.
Esamineremo ora la memoria dei caratteri e le aree per la creazione di caratteri speciali.

Struttura della memoria dei caratteri

Ogni carattere è formato da una matrice di 8 punti per 8. Un ingrandimento della lettera A, per esempio, si presenta così:

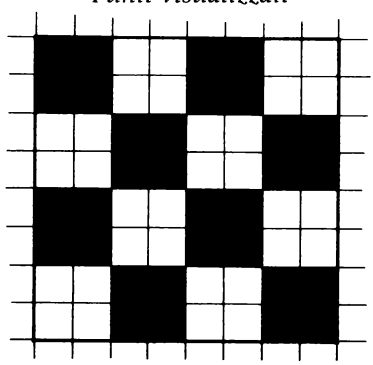


È stato scelto il formato 8×8 perché così ogni riga del carattere si carica convenientemente in un byte di memoria. Questo formato semplifica molto anche il disegno dei caratteri personali.
Disegnate il carattere desiderato su di un normale foglio di carta a quadretti, e convertite ogni riga nel corrispondente valore in binario. Gli otto byte che formano un carattere sono ubicati uno accanto all'altro nella memoria. Ogni punto su di una riga corrisponde ad un bit e il punto più a sinistra ha il valore binario più alto, 128. Un valore bit di 1 significa che il punto è "acceso" (appare nel colore del carattere). Un valore di 0 significa che è "spento" (appare col colore dello schermo). La lettera A si presenta così nella memoria dei caratteri



Punti visualizzati	*Binario*	*Decimale*
	00011000	24
	00111100	60
	01100110	102
	01111110	126
	01100110	102
	01100110	102
	01100110	102
	00000000	0

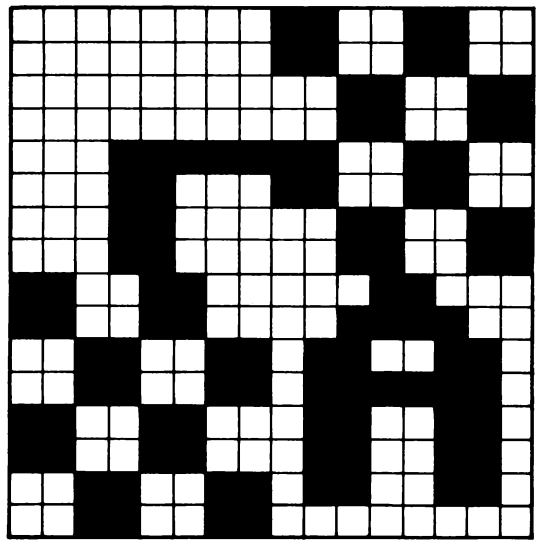
Qui, come paragone, c'è un carattere a scacchi:



Punti visualizzati	*Binario*	*Decimale*
	11001100	204
	00110011	51
	11001100	204
	00110011	51
	11001100	204
	00110011	51
	11001100	204
	00110011	51

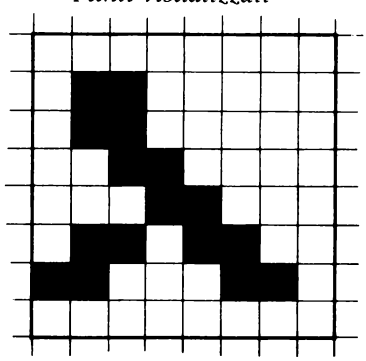
Notate che il carattere a scacchi ha dei punti che toccano il margine della matrice, mentre la A possiede il bordo libero su tre lati; questo spazio

è l'unico esistente tra caratteri adiacenti sul video. Le matrici dei caratteri presentati sullo schermo si toccano producendo un campo continuo di punti. Ecco un ingrandimento di una piccola parte dello schermo.



Il contatto diretto tra caratteri grafici è quello che permette di costruire i soggetti composti di caratteri multipli. Ciò significa che si devono includere degli spazi in alcuni caratteri per renderli distinguibili l'uno dall'altro. Se, per esempio, si vuole stampare la lettera greca "lambda" in una formula scientifica, si potrebbe definirla così:

Punti visualizzati



Binario	*Decimale*
00000000	0
01100000	96
01100000	96
00110000	48
00011000	24
01101100	108
11000110	198
00000000	0

Abbiamo lasciato degli spazi vuoti alla destra e sotto al carattere, per poterlo allineare con gli altri.

I punti mancanti

Si fa presto a calcolare che se vi sono 8 punti in ogni carattere e 40 caratteri per riga, ogni riga dello schermo è costituita da solo 320 punti. Cosa è successo agli altri visto che, come precedentemente detto, un televisore ne ha circa 500 per riga? Alcuni vanno perduti perché il fascio di elettroni esplora anche oltre i bordi dello schermo e altri sono occupati dalla cornice del video, per cui, sebbene ve ne siano più di 500, sono disponibili solo 320 punti.

Localizzazione di un carattere nella memoria

Le sagome di definizione dei caratteri sono memorizzate come codice video, che è in effetti un indice per la matrice della memoria dei caratteri. Come per la memoria del video e del colore, la memoria dei caratteri non è propriamente un array variabile in BASIC, ma può essere visualizzata così.
Come accennato prima, gli otto byte che definiscono un carattere sono memorizzati in locazioni adiacenti nella memoria del colore, con la riga più alta per prima e quella più bassa per ultima. Ecco come sono utilizzati i primi byte dei caratteri incorporati dalla Commodore (la memoria dei caratteri inizia alla locazione 53248):

Locazione	*Contenuto*
53248	Prima riga di "@"
53249	Seconda riga di "@"
53250	Terza riga di "@"
53251	Quarta riga di "@"
53252	Quinta riga di "@"
53253	Sesta riga di "@"
53254	Settima riga di "@"
53255	Ultima riga di "@"
53256	Prima riga di A
53257	Seconda riga di A
.	
.	
53263	Ultima riga di A
53264	Prima riga di B
53265	Seconda riga di B
.	
.	

Questo prospetto della memoria usa la seguente formula per trovare la definizione di una particolare riga di un dato carattere:

locazione = inizio della memoria dei caratteri + riga +
8 * codice video

Analisi della memoria dei caratteri

L'esame del contenuto della memoria dei caratteri richiede un po' di "trucchi". L'avere sempre a disposizione la ROM occupa spazio di memoria altrimenti disponibile. Dato che la maggior parte dei programmi non la utilizzano, il C-64 è stato progettato in modo che la memoria dei caratteri sia normalmente nascosta. Questo fa sì che 4096 byte di memoria in più sono disponibili per i programmi e per i dati, ma significa anche che è necessario un ulteriore sforzo di programmazione per quei programmi che hanno bisogno delle definizioni dei caratteri.
Il seguente programma permette di scegliere e ingrandire un carattere sullo schermo. Esso presenta anche altre informazioni (la sua locazione in memoria e il valore decimale delle righe) che vi aiuteranno ad acquistare familiarità con il modo in cui sono disegnati i caratteri.

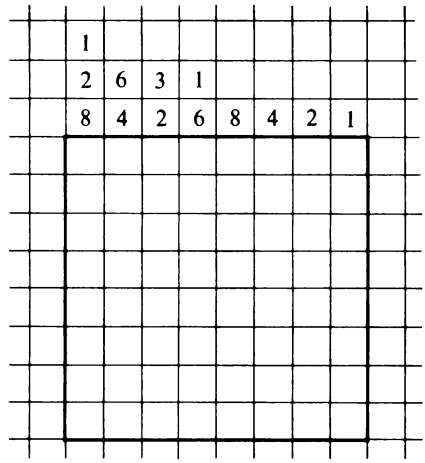
```
10 PRINT "■"
20 PRINT :PRINT"ATTENDERE PREGO"
100 POKE 52,128 : CLR
110 REM COPIA LA ROM DEI CARATTERI NELLA RAM
120 POKE 56334,PEEK(56334)AND 254
130 POKE 1,PEEK(1) AND 251
140 FOR I=0 TO 2047:POKE 32768+I,PEEK(53248+I):NEXT I
150 POKE 1,PEEK(1) OR 4
160 POKE 56334,PEEK(56334) OR 1
170 PIX$(0)=" ":PIX$(1)="■ ■"
180 PRINT"■INGRANDITORE DI CARATTERI"
190 REM ACQUISISCE UN CARATTERE
200 PRINT"SCEGLI UN CARATTERE";
210 GET KV$:IF KV$="" THEN 210
215 PIX$(0)=" ":PIX$(1)="■ ■"
220 GOSUB 490
230 REM IGNORA I CARATTERI NON STAMPABILI
240 IF SC=128 THEN 210
250 REM TROVA LA DEFINIZIONE DEL CARATTERE
260 CB=32768+8*SC
270 PRINT "■CARATTERE: ";KV$
280 PRINT "CODICE ASCII: ";ASC(KV$)
290 PRINT "CODICE VIDEO: ";SC
300 PRINT "IL CARATTERE E' POSTO ALLA: "; CB
310 PRINT
320 REM  STAMPA IL CARATTERE INGRANDITO
330 PRINT "■■          "
340 FOR I=CB TO CB+7
```

```
350 ROW=PEEK(I)
360 FR$=RIGHT$("        "+STR$(ROW)),6)
370 OS$="■■ ■■"
380 REM TRASFERISCE UNA RIGA DI BIT IN UNA STRINGA
390 FOR J=7 TO 0 STEP -1
400 BIT=SGN(ROW AND 2↑J)
410 OS$=OS$+PIX$(BIT)
420 NEXT J
430 OS$=OS$+"■■ ■■"
440 PRINT OS$+FR$
450 NEXT I
460 PRINT "■■          ■"
470 PRINT:PRINT:PRINT:GOTO 200
480 REM RICERCA IL CODICE VIDEO
490 SC=ASC(KV$)
500 IF SC < 32 THEN SC=128: RETURN
510 IF SC < 64 THEN RETURN
520 IF SC < 96 THEN SC=SC-64: RETURN
530 IF SC < 128 THEN SC=SC-32 : RETURN
540 IF SC < 160 THEN SC=128: RETURN
550 IF SC < 192 THEN SC=SC-64 : RETURN
560 IF FC < 255 THEN SC=SC-128: RETURN
570 SC=94: RETURN
```

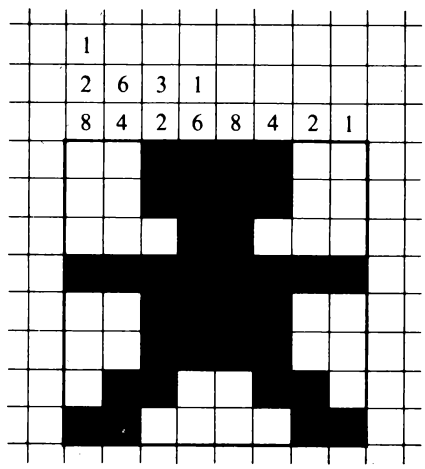
La parte difficile è nelle righe 12Ø-16Ø del programma che visualizzano il contenuto della ROM dei caratteri e lo copiano nelle celle da 20480 a 24575. Il funzionamento di questa parte del programma è descritto nella sezione "Grafica avanzata con la scheda VIC-II"; per il momento caricatelo e usatelo così com'è. Questo programma sarà anche di aiuto come fonte di esempi pratici nella prossima sezione, quindi sarebbe opportuno salvarlo con un SAVE.

DISEGNARE I CARATTERI

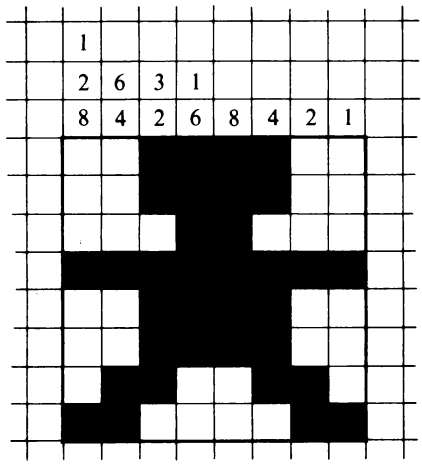
Si otterrà un buon risultato disegnando prima i caratteri su carta quadrettata (da 3 a 4 millimetri). Disegnate un quadrato con otto elementi per lato, in modo da riprodurre la matrice del carattere. Poi numerate le colonne così:



Si otterrà un'area che corrisponde ad un carattere, dove ogni elemento della matrice rappresenta un punto dello schermo. Ora bisogna riempire gli elementi dei punti che devono essere "accesi". Ecco un esempio di carattere semplice.



Una volta che il disegno vi soddisfa, calcolate i valori di ogni riga della memoria dei caratteri e per ogni riga sommate i numeri formati dai punti "accesi". Ecco i valori per l'omino della figura:



Binario	*Decimale*
00111100	60
00111100	60
00011000	24
11111111	255
00111100	60
00111100	60
01100110	102
11000011	195

Il nuovo carattere è ora pronto per una prova sullo schermo.

Uso dei caratteri personali

Dovete seguire queste fasi per usare i caratteri che avete disegnato:

1. Il programma deve riservare un'area di memoria per i caratteri.
2. I caratteri devono essere caricati in memoria.
3. Il programma deve ordinare alla VIC-II di usare i caratteri personali invece di quelli standard Commodore.

Per semplificare i primi esperimenti, si può cominciare facendo una copia dei caratteri Commodore, sostituendoli uno alla volta con quelli personali. Per cominciare, caricate ed eseguite il seguente programma:

```
100 REM RISERVA LA MEMORIA
110 POKE 52,128 : POKE 56,128 : CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 56576,(PEEK(56576) AND  252) OR 1
140 POKE 53272,32
150 REM E LO COMUNICA AL BASIC
160 POKE 648,136
170 PRINT "□■ABCDEFGHIJKLMNOPQRSTUVWXYZ[£]↑←!";
180 PRINT CHR$(34);CHR$(34);CHR$(20);
190 PRINT "#$%&'()*+,-.";
200 PRINT "/0123456789:;<=>?"
210 REM COPIA LA ROM DEI CARATTERI NELLA RAM
220 POKE 56334,PEEK(56334) AND 254
```

```
230 POKE 1,PEEK(1) AND 251
240 FOR I=0 TO 2047:POKE 32768+I,PEEK(53248+I)   NEXT
250 POKE 1,PEEK(1) OR 4
260 POKE 56334,PEEK(56334) OR 1
```

Ora esaminiamo questo programma riga per riga. La riga 11Ø cambia due dei puntatori interni del BASIC per far sembrare che vi sia meno spazio di memoria disponibile. (Il metodo per scegliere questo valore di POKE verrà spiegato in seguito.) Essa include anche un'istruzione CLR, che impone al BASIC di cancellare tutte le variabili che sono state definite e di assegnare gli altri puntatori al ridotto spazio di memoria. I POKE e CLR devono essere eseguiti prima che venga definita qualsiasi variabile, altrimenti i valori assegnati a quelle variabili verranno persi. È meglio mettere quest'istruzione all'inizio di tutti i programmi che definiscono dei caratteri personali.
Le righe 13Ø e 14Ø specificano alla scheda VIC-II che la memoria del video inizia alla cella 34816 e la memoria dei caratteri alla 32768. Dopo l'esecuzione di queste istruzioni, i caratteri sul video diventeranno incoerenti perché non sono ancora state caricate le definizioni personali. La riga 16Ø aggiorna la cella 648, per cui il BASIC può localizzare la nuova memoria dei caratteri.
Le righe da 17Ø a 2ØØ creano un'area nella memoria del video dove si può vedere il risultato della variazione stampando con PRINT la serie di caratteri personali.
Da 22Ø a 26Ø caricano la nuova serie di simboli con i caratteri maiuscoli e grafici standard Commodore: per questo occorreranno vari secondi. Quando il programma comincerà a copiare i caratteri inversi, si potranno osservare le righe in alto cambiare nel momento in cui vengono caricate le loro definizioni. Si è scelto di copiare i caratteri inversi e normali perché il C-64 genera il cursore accendendo e spegnendo il bit 7 (il bit "inverso") del carattere. Se si fossero copiati solamente i caratteri normali, il carattere sotto al cursore lampeggerebbe alternandosi fra se stesso e qualcosa di incoerente, perché il suo inverso non è stato caricato nella memoria dei caratteri.
Negli esempi che seguono si effettueranno dei cambiamenti nei caratteri inversi, lasciando intatti quelli normali. Questo renderà leggibili i listati dei programmi e le istruzioni in modo diretto. Come primo esperimento, sostituiremo la @ con il carattere dell'omino. Per cambiare la memoria del carattere @, caricate le seguenti istruzioni POKE; dovreste battere l'istruzione POKE una sola volta e continuare ad usare la stessa riga muovendo il cursore per evitare di far sparire i caratteri in alto sullo schermo.

```
POKE 33792, 60
POKE 33793, 60
```

```
POKE 33794, 24
POKE 33795,255
POKE 33796, 60
POKE 33797, 60
POKE 33798,102
POKE 33799,195
```

Quando viene eseguita ogni istruzione POKE, una riga del carattere @ cambia fino ad essere sostituita dall'omino. Ora battete CTRL-RVS ON, poi @; il C-64 stamperà un omino. Premete il CRSR LEFT fino a che il cursore non sia posizionato sull'omino. Invece di lampeggiare tra @ inversa e normale, il carattere lampeggia tra @ e omino. Ora potete provare con caratteri di vostra invenzione. Seguite semplicemente le fasi illustrate: fate un POKE con i valori calcolati sulla carta e usate CTRL-RVS ON per far apparire il carattere proveniente dalla parte riservata della memoria dei caratteri. Raccomandiamo di provare a lungo diversi caratteri prima di mettersi al lavoro per stendere un programma.

COME DISEGNARE CARATTERI PERSONALI

Un editor dei caratteri può essere di grande aiuto se si intende far largo uso di caratteri personali: esso presenta i caratteri sia in grandezza normale che ingranditi, permette di cambiare i punti individualmente e calcola automaticamente i valori di POKE. Si può sviluppare un programma del genere da sè o acquistarne uno già pronto. Molte parti del programma, se si vuole scrivere il proprio editor, possono essere ricavate dagli esempi nei capitoli 5 e 6 di questo libro; ad esempio, può essere usato il programma "Ingranditore di caratteri" per visualizzare il carattere, mentre il programma che sposta i punti può essere usato come base per la subroutine che cambia i punti. (Suggerimento: usate un loop di ritardo per far lampeggiare il punto ingrandito e usate il pulsante di fuoco del joystick per cambiare i punti). Può essere sufficiente, usando i caratteri personali solo occasionalmente, usare la carta a quadretti per disegnare i caratteri e calcolare i valori POKE a mano.

SVILUPPO DI PROGRAMMI CHE FANNO USO DI CARATTERI PERSONALI

I programmi che usano i caratteri personali non sono molto differenti da quelli che usano i caratteri standard. Vi è sempre la scelta dell'uso di PRINT o POKE (o entrambi) per costruire la presentazione. Tuttavia vi sono due differenze: il ridotto spazio di memoria e l'assenza della serie di caratteri standard.

Come destreggiarsi con la ridotta capacità di memoria

I caratteri personali usano la memoria in due modi. Una parte di memoria deve essere riservata per la memoria dei caratteri; nei nostri esempi si usa un'area di 8192 byte per semplificare, ma si può ridurla considerevolmente: in un paragrafo verso la fine di questo capitolo sarà spiegato come fare. Questa minor disponibilità di memoria non costituirà un problema se i caratteri personali non sono numerosi. Se invece volete definire un completo set di caratteri, potreste avere delle difficoltà per mancanza di spazio. Un'istruzione DATA per memorizzare un carattere occupa da 25 a 40 byte, a seconda del numero di cifre necessario. Ricordatevi che gli spazi in un'istruzione DATA sono memorizzati insieme al programma: si può quindi ridurre il fabbisogno di memoria eliminandoli. Generalmente si consiglia di usare spazi nei programmi per migliorarne la leggibilità, in questo caso le virgole che separano i valori di istruzioni DATA sono sufficienti.
Si può aumentare la disponibilità di memoria del programma principale dividendolo in due: una parte per costruire i caratteri, e una per svolgere la funzione principale. Se si fa eseguire per primo il programma che costruisce i caratteri questo può essere sostituito dal programma principale quando ha finito. Ciò può essere eseguito anche automaticamente, mettendo un'istruzione LOAD alla fine del programma che costruisce i caratteri. Caricando il programma da un nastro, registrate il programma che costruisce i caratteri per primo, seguito dal programma principale (l'ordine non ha invece importanza usando i dischetti). Le ultime due istruzioni del programma che costruisce i caratteri devono essere CLR e LOAD.

```
100 REM CARICA I CARATTERI PERSONALI
110 DATA 5,12,17,32
  .
  .
  .
340 REM RISERVA LA MEMORIA
350 POKE 52,128 : POKE 56,128
360 REM CANCELLA LE VARIABILI
370 CLR
380 REM CHIAMA IL PROGRAMMA PRINCIPALE
390 LOAD "MAIN"
```

Il C-64 caricherà ed eseguirà il programma chiamato MAIN. Il CLR è necessario per cancellare tutte le variabili definite nel programma che costruisce i caratteri. Il BASIC memorizza le variabili immediatamente dopo il testo del programma. Dato che il vostro programma principale probabilmente sarà più lungo di quello che costruisce i caratteri, una parte di questo verrà memorizzato nell'area occupata dalle variabili preceden-

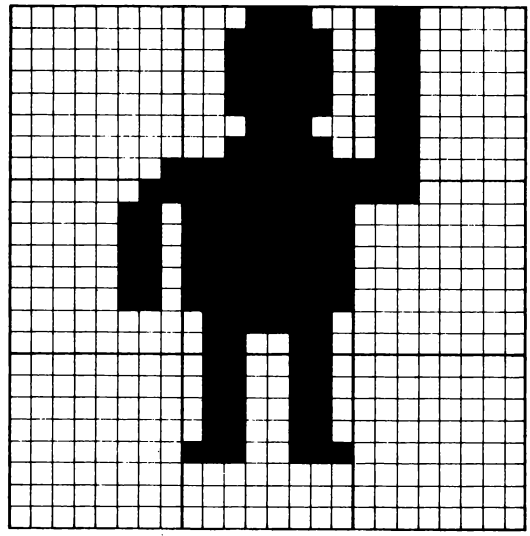
ti. Se non ci fosse l'istruzione CLR e ci fosse una variabile del programma principale con lo stesso nome di una variabile del programma che costruisce i caratteri, il BASIC potrebbe utilizzare dei valori assegnati a quest'ultima per tutt'altro scopo. Il danno così provocato ad un programma è grave e imprevedibile, ma si può facilmente evitare: includete sempre un'istruzione CLR prima di LOAD, ordinando così al BASIC di "dimenticare" le variabili preesistenti.

Accesso alla serie dei caratteri standard

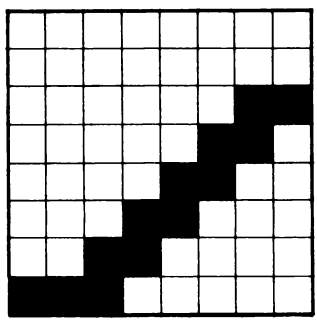
Il vostro programma potrebbe aver bisogno di servirsi anche dei caratteri standard. Se volete stampare dei messaggi, avete due possibilità: se non dovete ridefinire tutti i 256 caratteri, lasciate semplicemente intatti i caratteri alfabetici, se invece vi servono 256 caratteri, è possibile passare alternativamente dalla serie di caratteri personali alla serie di caratteri incorporati. La sezione "Grafica avanzata", più avanti in questo capitolo, spiega le modalità di questo passaggio.
I messaggi provenienti dal BASIC possono costituire un problema. Un semplice messaggio READY, se si ridefiniscono i caratteri alfabetici onde costruire dei giocatori, potrebbe apparire come un geroglifico egizio. Questo potrebbe essere causa di serie difficoltà, ad esempio durante la fase del *debugging* (correzione dei piccoli errori) del programma. I messaggi prodotti potrebbero essere difficili da interpretare se si interrompe l'esecuzione con il tasto STOP, oppure se il BASIC trova un errore.
Si noti inoltre che RUN/STOP-RESTORE fanno tornare al set dei caratteri standard, ma lasciano la memoria dei caratteri personali intatta e protetta dal BASIC. Sfortunatamente questo cancella anche il video e con esso i messaggi, per cui non è molto utile durante il debugging dei programmi.

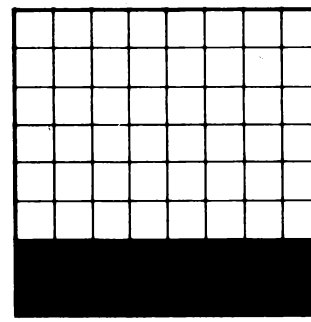
ANIMAZIONE DEI GIOCATORI

Un oggetto costituito da caratteri personali può essere animato modificandone la forma nella memoria dei caratteri; questa tecnica è generalmente troppo lenta se scritta in BASIC, mentre è sufficientemente veloce con programmi in linguaggio macchina. Un metodo più facile e veloce è quello di definire più di una versione del giocatore nella memoria dei caratteri. Il programma può poi farlo muovere cambiando la versione presentata con un POKE alla memoria del video.
Come esempio di questo metodo di animazione, modificate l'omino costruendolo di nove caratteri invece di uno.



Per animarlo un po', gli si farà muovere un braccio definendo due diverse versioni del carattere che contiene il braccio.





Per vederlo in azione, caricate il seguente programma:

```
100 REM RISERVA LA MEMORIA
105 PRINT "■■■■■■■■■ATTENDERE PREGO"
110 POKE 52,128 : POKE 56,128 : CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 56576,(PEEK(56576) AND 252) OR 1
140 POKE 53272,32
150 REM E LO COMUNICA AL BASIC
```

```
160 POKE 648,136
170 REM COPIA LA ROM DEI CARATTERI NELLA RAM
180 POKE 56334,PEEK(56334) AND 254
190 POKE 1,PEEK(1) AND 251
200 FOR I=0 TO 2047:POKE 32768+I,PEEK(53248+I)   NEXT
210 POKE 1,PEEK(1) OR 4
220 POKE 56334,PEEK(56334) OR 1
230 REM PULISCE LO SCHERMO E PREDISPONE L'AREA
240 DLY=250 : SB=34816
250 PRINT "■■■AB"
260 PRINT "■CDE"
265 PRINT "■FGH"
270 REM OMINO BASE
280 FOR I=33792 TO 33863 : READ X : POKE I,X   NEXT I
290 DATA 0,0,0,0,0,0,0,0
300 DATA 28,62,62,62,62,28,62,255
310 DATA 96,96,96,96,96,96,96,96
320 DATA 3,6,6,6,6,6,0,0
330 DATA 255,255,255,255,255,255,126,102
340 DATA 192,0,0,0,0,0,0,0
350 DATA 0,0,0,0,0,0,0,0
360 DATA 102,102,102,102,231,0,0,0
370 DATA 0,0,0,0,0,0,0,0
380 REM AGGIUNGE LE BRACCIA
390 FOR I=33864 TO 33879 : READ X : POKE I,X : NEXT I
400 DATA 0,0,3,6,12,24,48,224
410 DATA 0,0,0,0,0,0,255,255
420 POKE SB+2,130
430 FOR I=1 TO DLY   NEXT I
440 POKE SB+2,137
450 FOR I=1 TO DLY : NEXT I
460 POKE SB+2,138
470 FOR I=1 TO DLY   NEXT I
480 POKE SB+2,137
490 FOR I=1 TO DLY : NEXT I
500 GOTO 420
```

Quando si fa eseguire questo programma, l'omino viene posizionato nell'angolo in alto a sinistra dello schermo e il suo braccio comincia a muoversi.
Nota: vi è una lunga pausa prima che l'omino appaia sullo schermo.

Animazioni più complesse

Il C-64 è capace d'animazioni ben più complesse dell'esempio dell'omino, con l'utilizzo però delle stesse tecniche d'animazione. Il vostro programma potrebbe, per esempio, avere più di un oggetto in movimento contemporaneamente, oppure uno che compie più di un movimento allo stesso

tempo. Entrambe queste possibilità possono essere eseguite definendo in anticipo diverse versioni dei caratteri personali. Si studi ora un esempio di due oggetti in moto simultaneamente. Si disegna un altro omino che saluta il primo col braccio. Modificate il programma dell'omino come segue:

```
5 PRINT "■■■■■■■■■■■■ATTENDERE PREGO"
10 REM RISERVA LA MEMORIA
100 CLR:POKE52,128:POKE56,128:CLR
110 GOSUB 10020
120 REM L'OMINO DI SINISTRA SALUTA
130 FOR J=1 TO 2 : GOSUB 220 : NEXT J
140 REM ENTRAMBI GLI OMINI SALUTANO
150 FOR J=1 TO 5 : GOSUB 320 : NEXT J
160 REM L'ØMINO DI DESTRA SALUTA
170 FOR J=1 TO 2 : GOSUB 500 : NEXT J
180 REM ENTRAMBI GLI OMINI SI FERMANO
190 FOR J=1 TO 1000 : NEXT J
200 GOTO 130
210 REM L'OMINO DI SINISTRA MUOVE IL BRACCIO
220 POKE SB+2,130
230 FOR I=1 TO DLY : NEXT I
240 POKE SB+2,137
250 FOR I=1 TO DLY : NEXT I
260 POKE SB+2,138
270 FOR I=1 TO DLY : NEXT I
280 POKE SB+2,137
290 FOR I=1 TO DLY : NEXT I
300 RETURN
310 REM ENTRAMBI GLI OMINI MUOVONO LE BRACCIA
320 POKE SB+2,130
330 FOR I=1 TO DLY/2 : NEXT I
340 POKE SB+8,137
350 FOR I=1 TO DLY/2 : NEXT I
360 POKE SB+2,137
370 FOR I=1 TO DLY/2 : NEXT I
380 POKE SB+8,138
390 FOR I=1 TO DLY/2 : NEXT I
400 POKE SB+2,138
420 POKE SB+8,137
430 FOR I=1 TO DLY/2 : NEXT I
440 POKE SB+2,137
450 FOR I=1 TO DLY/2 : NEXT I
460 POKE SB+8,130
470 FOR I=1 TO DLY/2 : NEXT I
480 RETURN
490 REM L'OMINO DI DESTRA MUOVE IL BRACCIO
500 POKE SB+8,137
510 FOR I=1 TO DLY : NEXT I
520 POKE SB+8,138
530 FOR I=1 TO DLY   NEXT I
540 POKE SB+8,137
```

```
550 FOR I=1 TO DLY : NEXT I
560 POKE SB+8,130
570 FOR I=1 TO DLY   NEXT I
580 RETURN
590 REM INIZIALIZZAZIONE
600 REM STABILISCE LA LUNGHEZZA DEL LOOP DI RITARDO
610 DLY=100
10020 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
10030 POKE 56576,(PEEK(56576) AND 252) OR 1
10040 POKE 53272,32
10050 REM E LO COMUNICA AL BASIC
10060 POKE 648,136
10070 REM COPIA LA MEMORIA DEI CARATTERI
10080 POKE 56334,PEEK(56334) AND 254
10090 POKE 1,PEEK(1) AND 251
10100 FOR I=0 TO 2047
10105 POKE 32768+I,PEEK(53248+I)   NEXT
10110 POKE 1,PEEK(1) OR 4
10120 POKE 56334,PEEK(56334) OR 1
10130 REM PULISCE LO SCHERMO E PREDISPONE L'AREA
10140 DLY=250 : SB=34816
10150 PRINT "■ ■AB■     ■AJ■
10160 PRINT "■CDE■    ■CDE"
10170 PRINT "■FGH■    ■FGH"
10180 REM OMINO BASE
10190 FOR I=33792 TO 33863   READ X
10195 POKE I,X : NEXT I
10200 DATA 0,0,0,0,0,0,0,1
10210 DATA 28,62,62,62,62,28,62,255
10220 DATA 96,96,96,96,96,96,96,224
10230 DATA 3,6,6,6,6,6,0,0
10240 DATA 255,255,255,255,255,255,126,102
10250 DATA 192,0,0,0,0,0,0,0
10260 DATA 0,0,0,0,0,0,0,0
10270 DATA 102,102,102,102,231,0,0,0
10280 DATA 0,0,0,0,0,0,0,0
10290 REM AGGIUNGE LE BRACCIA
10300 FOR I=33864 TO 33879 : READ X
10305 POKE I,X : NEXT I
10310 DATA 0,0,3,6,12,24,48,224
10320 DATA 0,0,0,0,0,0,0,255
10330 RETURN
```

Quando si esegue il programma, l'omino alla sinistra del video inizia ad agitare il braccio. Dopo un attimo, quello a destra comincia a fare lo stesso fino a che l'omino a sinistra smette di salutare, seguito da quello a destra; quindi il ciclo si ripete.

Le tre subroutine sono la chiave del movimento (righe 21Ø, 31Ø e 49Ø). La prima, con inizio a riga 21Ø, fa solamente muovere il braccio all'omino di

sinistra. La seconda con inizio a riga 31Ø, è responsabile del movimento delle braccia di entrambi gli omini. Notare che l'omino a sinistra si muove, poi il programma si ferma brevemente prima di movere anche l'omino sulla destra. Dopo il movimento del braccio della figura di destra, il programma attende brevemente. Ciò mantiene la velocità dei due omini quasi uguale, anche se i due salutano insieme. Quello a destra inoltre agita il braccio in senso opposto all'altro, di modo che non sembri essere "legato" all'altro. Queste piccole differenze fanno sì che la presentazione sia più interessante.
L'ultima subroutine, con inizio a riga 49Ø, muove solamente la figura di destra, mentre quella a sinistra rimane ferma.

6.6. Grafica ad alta risoluzione

Con la grafica *bit-mapped*, il programma gestisce i punti dello schermo individualmente invece che a caratteri interi. Questa tecnica deriva il suo nome dal fatto che esiste un'area di memoria utilizzata appositamente per rappresentare il contenuto del video punto per punto. Proprio come città, parchi ed aeroporti sono rappresentati sulle carte geografiche da simboli, i punti sullo schermo sono rappresentati da bit in memoria. La topografia dei bit può essere usata per disegnare linee sottili sullo schermo o per rendere più fluido il movimento di un giocatore, dato che quest'ultimo si può muovere ad incrementi di un punto alla volta invece che di un intero carattere.
L'uso di questa tecnica sul C-64 è molto simile all'uso dei caratteri personali. La scheda VIC-II tratta entrambe esattamente nello stesso modo: in ambedue i casi, analizza il contenuto di una cella di memoria di 8 punti per 8, e presenta i punti una riga per volta. La differenza sta nel modo in cui la VIC-II decide dove cercare nella tabella; nella visualizzazione di un carattere usa il valore nella memoria del video (il "codice video" del carattere) per trovare l'informazione nella memoria dei caratteri. Per visualizzare una mappa di bit, la VIC-II consulta la tabella in modo sequenziale. È come se si avesse una tabella di 1000 caratteri personali e lo schermo contenesse codici video numerati da 0 a 999.
Si possono usare anche in questo caso molti dei concetti e dei metodi descritti nei paragrafi dedicati ai caratteri personali. La differenza principale è questione di tecnica di programmazione. Per usare i caratteri personali, si definisce la memoria dei caratteri e si modifica la presentazione cambiando la memoria del video. Nella visualizzazione di una mappa di bit, la memoria del video rimane invariata ma la memoria dei caratteri cambia.

Per capire questa tecnica, torniamo a quanto detto sulla memoria dei caratteri, ma con una prospettiva differente. Dal punto di vista della visualizzazione del testo, si concepisce la memoria del video come contenente caratteri da rappresentare, mentre la memoria dei caratteri contiene la posizione relativa dei punti che rappresentano i caratteri. In termini di grafica ad alta risoluzione, la memoria dei caratteri diventa una rappresentazione del video punto per punto e la memoria del video è sostituita da un puntatore che segnala alla scheda VIC-II la posizione del punto sullo schermo. In ambedue i casi, la scheda VIC-II compie esattamente la stessa azione: consulta la disposizione dei punti nella memoria dei caratteri e li presenta sul video. È differente solo il metodo di ricerca della disposizione dei punti nella tabella.
Pensate al metodo utilizzato per rappresentare un omino con molti caratteri: si può pensare alla grafica ad alta risoluzione come all'uso dell'intero schermo per disegnare un singolo "omino gigante".
La grafica ad alta risoluzione comporta anche qualche inconveniente. Primo, usa una gran quantità di memoria: per memorizzare completamente il video "topograficamente" occorrono 8000 byte di memoria. Secondo, è una tecnica univoca: non è possibile avere sullo schermo dei caratteri comuni e grafici ad alta risoluzione allo stesso tempo. È possibile tuttavia usare l'alta risoluzione senza perdere la possibilità di visualizzare dei caratteri. Dato che una presentazione in grafica ad alta risoluzione è molto simile a quella dei caratteri personali, si può creare una mini presentazione grafica usando solo una piccola porzione dello schermo. Le tecniche usate per definire una serie di caratteri personali possono essere impiegate anche per definire un'area di 64 punti×64 per la sperimentazione con la grafica ad alta risoluzione. Il programma di preparazione (*Setup*) listato qui di seguito potrebbe sembrare già visto: è stato realizzato infatti apportando solo alcune modifiche a quello che prepara la serie di caratteri personali.

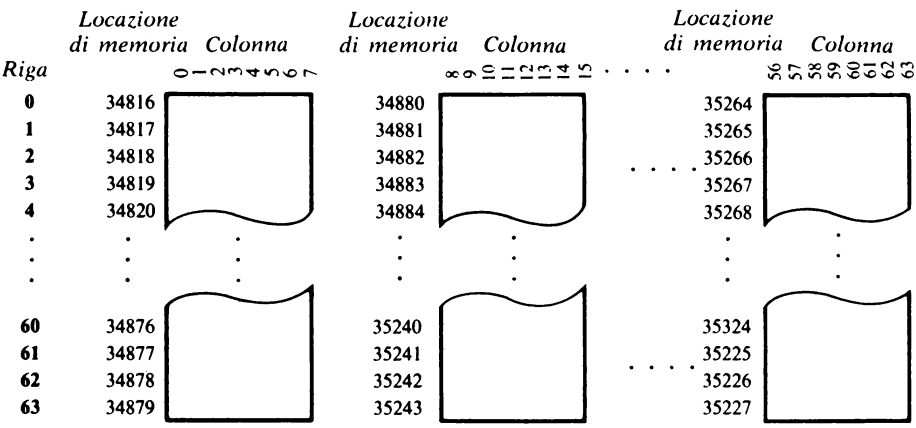
```
100 REM PROGRAMMA "SETUP"
110 REM RISERVA LA MEMORIA
120 POKE 52,128 : POKE 56,128 : CLR
130 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
140 POKE 56576,(PEEK(56576) AND 252) OR 1
150 POKE 53272,32
160 REM E LO COMUNICA AL BASIC
170 POKE 648,136 : SB=34816
180 REM COPIA LA ROM DEI CARATTERI NELLA RAM
190 POKE 56334,PEEK(56334) AND 254
200 POKE 1,PEEK(1) AND 251
210 FOR I=0 TO 1023:POKE 32768+I,PEEK(53248+I)   NEXT
220 POKE 1,PEEK(1) OR 4
```

```
220 POKE 1,PEEK(1) OR 4
230 POKE 56334,PEEK(56334) OR 1
240 REM AZZERA L'AREA AD ALTA RISOLUZIONE
250 FOR I=33792 TO 34303 : POKE I,0 : NEXT I
260 REM PULISCE LO SCHERMO E PREDISPONE L'AREA
270 PRINT "J■■HPX (08"
280 PRINT "■AI@Y!)19"
290 PRINT "■BJRZ";CHR$(34);CHR$(34);CHR$(20);"*2:"
300 PRINT "■CKS[#+3;"
310 PRINT "■DLT£$,4<"
320 PRINT "■EMU]%-5="
330 PRINT "■FNV↑&.6>"
340 PRINT "■GOW←'/7?"
```

Dopo l'esecuzione di questo programma, l'angolo superiore sinistro dello schermo diventa un'area di lavoro per grafica ad alta risoluzione nella quale condurre tutti gli esperimenti possibili.

CAMBIARE I PUNTI NELL'AREA DI LAVORO AD ALTA RISOLUZIONE

Dopo aver fatto eseguire il programma SETUP, la parte di memoria che definisce l'area di lavoro si presenta così:



Per cambiare un punto sul video, il vostro programma deve trovare la locazione giusta in cui fare un POKE e anche il valore da metterci. I 64 bit della dimensione X sono divisi in 8 byte da 8 bit ciascuno. Per trovare la colonna giusta, usate la seguente formula:

```
450 COL=INT(X/8)
```

Dato che ogni colonna di byte dell'area dedicata è alta 64 righe, il numero della colonna deve essere moltiplicato per 64. Il calcolo per trovare la locazione corretta per POKE è

```
460 PL=33792+Y+64*COL
```

Il bit all'interno di quel byte è il resto della divisione per 8 fatta per trovare la colonna. Per calcolare il bit giusto, usate

```
460 PL=33792+Y+64*COL
470 BIT=7-(X-COL*8)
```

Si è sottratto il resto da 7 perché le colonne ad alta risoluzione sono numerate da sinistra a destra, ma i bit in un byte sono numerati da destra a sinistra. Non è sufficiente conoscere semplicemente il numero del bit, perché il vostro programma farà uso di POKE per cambiare la visualizzazione e deve calcolare il numero che corrisponde a quel bit. Ricordatevi che ogni bit in un byte rappresenta una potenza di 2.

Bit	7	6	5	4	3	2	1	0
Valore di POKE	128	64	32	16	8	4	2	1

In questo caso si può facilmente operare la conversione dal numero bit al valore POKE usando l'operatore dell'elevamento a potenza.

```
480 PV=2↑BIT
```

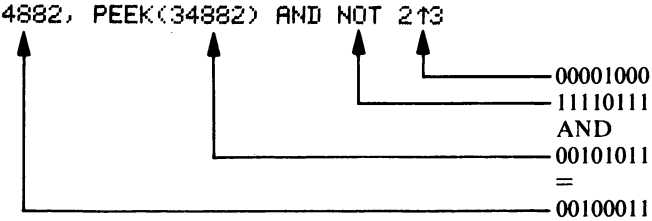
Il vostro programma non può semplicemente usare POKE a caso nel byte trovato perché vi sono altri 7 bit che non si devono disturbare. Per cambiare un solo bit, bisogna fare un PEEK nel byte che deve essere cambiato, modificare solamente il bit giusto, poi eseguire un POKE per rimetterlo al suo posto. Per cambiare solo un singolo bit si possono usare gli operatori AND e OR. Per esempio, per azzerare un bit (portare quel punto al colore dello sfondo), si può usare una variazione della tecnica di mascheramento usata per isolare i bit degli interruttori del joystick.

```
500 POKE PL,PEEK(PL) AND NOT PV
```

Notate l'uso dell'operatore NOT. Quando abbiamo isolato gli interruttori

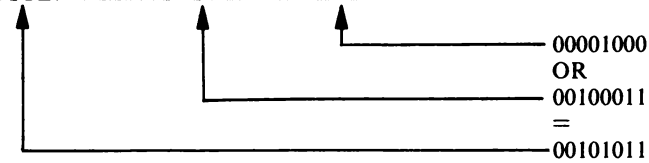
del joystick, veniva usato il valore del bit direttamente, con un mascheramento nel quale il bit desiderato era un 1 e tutti gli altri erano degli 0. Questa volta si usa una maschera nella quale il bit richiesto è uno 0 e tutti gli altri degli 1. L'istruzione AND farà diventare il bit 0 e lascerà gli altri indisturbati.
Si supponga di voler azzerare il bit 3 nella locazione 34882, che contiene al momento un valore di 43; si potrebbe usare un'istruzione BASIC come questa:

```
10 POKE 34882, PEEK(34882) AND NOT 2↑3
```



Quando con POKE si rimette il risultato nella locazione 34882, solo il bit 3 è cambiato. Gli altri bit mantengono i loro valori. Per far tornare il punto alla situazione precedente usate l'operatore OR.

```
10 POKE 34882, PEEK(34882) OR 2↑3
```



Come prima, è cambiato solo il valore del bit 3.
Per ottenere il cambiamento di un solo bit ci vuole molto lavoro. Tuttavia, dato che ci sono solo 8 bit in un byte e vi sono solo 8 valori possibili per la maschera AND e 8 per la maschera OR, è pratico (e molto più veloce) calcolarli in anticipo e memorizzarli in tabelle. Il frammento di programma che segue può essere usato come subroutine nei programmi che producono immagini ad alta risoluzione. Esso crea due matrici di maschere chiamate M1% (pone il bit=1) e M0% (azzera).

```
100 FOR I=0 TO 7
110 M1%(I)=2↑I
120 M0%(I)=NOT M1%(I)
130 NEXT I
```

Notate che gli indici delle matrici, come i numeri dei bit, iniziano con 0 perché si calcola il numero del bit usando il resto di una divisione che può essere 0. Le matrici sono specificate come variabili intere perché il BASIC esegue le operazioni Booleane con numeri interi. Usare queste maschere precalcolate non solo rende più veloci i programmi, ma li rende anche più facili da leggere. Paragonate i precedenti esempi per assegnare e riassegnare un bit con istruzioni che eseguono le stesse operazioni usando la tabella.

```
10 POKE 34882,PEEK(34882) AND M0%(3)
20 POKE 34882,PEEK(34882) OR M1%(3)
```

Anche se si usano tecniche come il precalcolare le maschere, il BASIC è generalmente troppo lento per l'animazione ad alta risoluzione: vi sono troppi bit da cambiare per muovere un oggetto sullo schermo. Si potrebbero muovere i bit più rapidamente in linguaggio macchina. Il BASIC è, tuttavia, abbastanza utile per visualizzazioni che non si muovono, come disegni e grafici. Per esempio, il seguente programma traccerà un triangolo sullo schermo:

```
100 REM PROGRAMMA "SETUP"
110 REM RISERVA LA MEMORIA
120 POKE 52,128 : POKE 56,128 : CLR
130 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
140 POKE 56576,(PEEK(56576) AND 252) OR 1
150 POKE 53272,32
160 REM E LO COMUNICA AL BASIC
170 POKE 648,136 : SB=34816
180 REM COPIA LA ROM DEI CARATTERI NELLA RAM
190 POKE 56334,PEEK(56334) AND 254
200 POKE 1,PEEK(1) AND 251
210 FOR I=0 TO 1023:POKE 32768+I,PEEK(53248+I) : NEXT
220 POKE 1,PEEK(1) OR 4
230 POKE 56334,PEEK(56334) OR 1
240 REM AZZERA L'AREA AD ALTA RISOLUZIONE
250 FOR I=33792 TO 34303 : POKE I,0 : NEXT I
260 REM PULISCE LO SCHERMO E PREDISPONE L'AREA
265 REM L'AREA DI LAVORO
270 PRINT "■■■HPX (08"
280 PRINT "■AI0Y!)19"
290 PRINT "■BJRZ";CHR$(34);CHR$(34);CHR$(20);"*2:"
300 PRINT "■CKS[#+3;"
310 PRINT "■DLT£$,4<"
320 PRINT "■EMU]%-5="
330 PRINT "■FNV↑&.6>"
```

```
340 PRINT "■GOW←/7?"
350 REM COSTRUISCE L'ARRAY-MASCHERA
360 FOR I=0 TO 7:M1%(I)=2↑I : NEXT I
370 REM TRACCIA LA BASE DEL TRIANGOLO
380 Y=63
390 FOR X=0 TO 63
400 GOSUB 540
410 NEXT
420 REM TRACCIA IL LATO SINISTRO DEL TRIANGOLO
430 FOR X=0 TO 30
440 Y = 63-X*2
450 GOSUB 540
460 NEXT
470 REM TRACCIA IL LATO DESTRO DEL TRIANGOLO
480 FOR X=31 TO 62
490 Y = 63-(62-X)*2
500 GOSUB 540
510 NEXT
520 END
530 REM SUBROUTINE DI TRACCIAMENTO
540 COL=INT(X/8)
550 PL=33792+Y+64*COL
560 BIT=7-(X-COL*8)
570 POKE PL,PEEK(PL) OR  M1%(BIT)
580 RETURN
```

Quando si esegue il programma si nota che il triangolo viene disegnato troppo lentamente per essere utile in un gioco d'azione; potrebbe essere utile invece in un programma didattico che presenti figure geometriche.

USO DELL'INTERO VIDEO

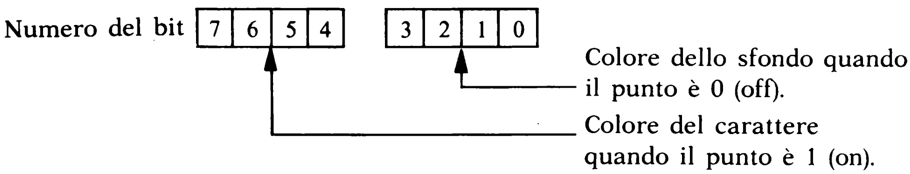
Le tecniche trattate nell'ultima sezione funzionano ugualmente bene per un intero schermo ad alta risoluzione. Tuttavia, è necessario cambiare le formule per trovare il byte giusto per operare il POKE. Una subroutine per eseguire il calcolo si potrebbe presentare così:

```
100 PL = BM+(40*INT(Y/8))+(Y AND 7)+INT(X/8)
```

È necessario ordinare alla VIC-II di passare al modo ad alta risoluzione. Ciò è controllato dal bit 5 nella locazione 53265. Quando questo bit è uguale a 1, la VIC-II passa alla grafica ad alta risoluzione. Gli altri bit in questa locazione dovrebbero essere preservati; usate quindi la stessa tecnica di mascheramento usata per i byte della memoria dei caratteri:

```
100 POKE 53265,PEEK(53265) OR 32
```

Rimane da considerare un aspetto della grafica ad alta risoluzione: i colori dei punti sono specificati differentemente dalla visualizzazione dei caratteri. Ci si chiederà cosa ne è stato della memoria del video nel modo ad alta risoluzione. Ricordatevi che la locazione nella memoria dei caratteri, nelle visualizzazioni ad alta risoluzione, non proviene dalla memoria del video. Questo rende la memoria del video disponibile per memorizzare i codici dei colori. Esattamente come per la visualizzazione dei caratteri, un punto 0 in alta risoluzione appare sullo schermo nel colore dello sfondo, un punto 1 appare nel colore del carattere. Ma la memoria del video è larga 8 bit, non 4, quindi ci stanno due codici di colore. Nel modo ad alta risoluzione ogni "cellula" da 8 punti×8 ha il proprio colore di sfondo e di carattere. Il colore dello sfondo è memorizzato nei bit 0-3 e il colore del carattere nei bit 4-7.



Ciò vi dà una grande libertà di mescolare i colori sul video. Per calcolare il valore da trasferire con POKE per una data posizione sul video, usate la formula:

```
100 POKE 36867,(PEEK(36867) AND 127)
OR ((SB/8) AND 128)
```

Ora considerate come funzionano queste tecniche in un programma vero. Modificate il programma del disegno del triangolo per usare uno schermo intero ad alta risoluzione. Se lo si paragona a quello che usava caratteri personali, si vedrà che non è cambiato molto.

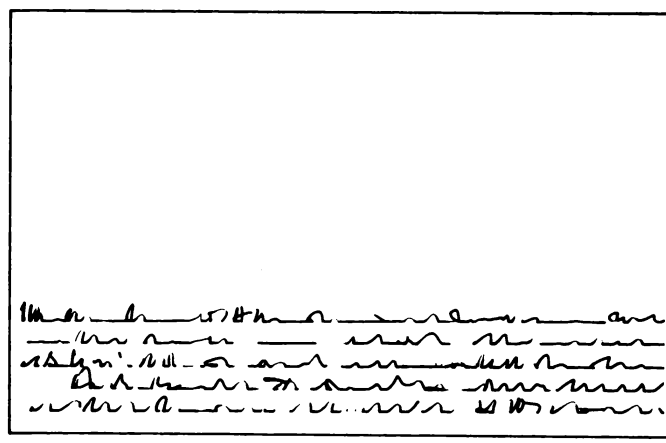
```
100 REM GRAFICA AD ALTA RISOLUZIONE
105 REM PROGRAMMA DIMOSTRATIVO
110 REM RISERVA LA MEMORIA
115 PRINT "▒▒▒▒▒▒▒▒▒▒▒▒▶▶▶▶▶▶▶▶▶▶▶▶▶▶ATTENDERE PREGO"
120 POKE 52,64 : POKE 56,64 : CLR
130 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
140 POKE 56576,(PEEK(56576) AND 252) OR 2
150 POKE 53272,8
160 REM PREDISPONE LA VIC-II AL MODO AD ALTA RISOLUZIONE
170 POKE 53265,PEEK(53265) OR 32
180 REM POSIZIONA IL PUNTATORE
```

```
190 BM=24576 : SB=16384
200 REM AZZERA L'AREA AD ALTA RISOLUZIONE
210 FOR I=BM TO BM+7999 : POKE I,0 : NEXT I
220 REM RIEMPIE LA MEMORIA DELLO SCHERMO
225 REM CON I CODICI DEI COLORI
230 FOR I=SB TO SB+999 : POKE I,230   NEXT I
240 REM COSTRUISCE L'ARRAY MASCHERA
250 FOR I=0 TO 7:M1%(I)=2↑(7-I) : NEXT I
260 REM TRACCIA LA BASE DEL TRIANGOLO
270 Y=63
280 FOR X=0 TO 63
290 GOSUB 530
300 NEXT
310 REM TRACCIA IL LATO SINISTRO DEL TRIANGOLO
320 FOR X=0 TO 30
330 Y = 63-X*2
340 GOSUB 530
350 NEXT
360 REM TRACCIA IL LATO DESTRO DEL TRIANGOLO
370 FOR X=31 TO 62
380 Y = 63-(62-X)*2
390 GOSUB 530
400 NEXT
410 REM ASPETTA CHE UN TASTO VENGA PREMUTO
420 GET A$ : IF A$="" THEN 420
430 REM RIPRISTINA IL SISTEMA
440 REM RESTITUISCE AL BASIC L'AREA AD ALTA RISOLUZIONE
450 POKE 52,128 : POKE 56,128 : CLR
460 REM RIPORTA LA VIC-II AL MODO STANDARD
470 POKE 56576,(PEEK(56576) AND 252) OR 3
480 REM RITORNA AL MODO CARATTERI
490 POKE 53265,PEEK(53265) AND 223
500 POKE 53272,21
510 END
520 REM SUBROUTINE DI TRACCIAMENTO
530 PL=BM+(40*(Y AND 248))+(Y AND 7)+(X AND 504)
540 POKE PL,PEEK(PL) OR M1%(X AND 7)
550 RETURN
```

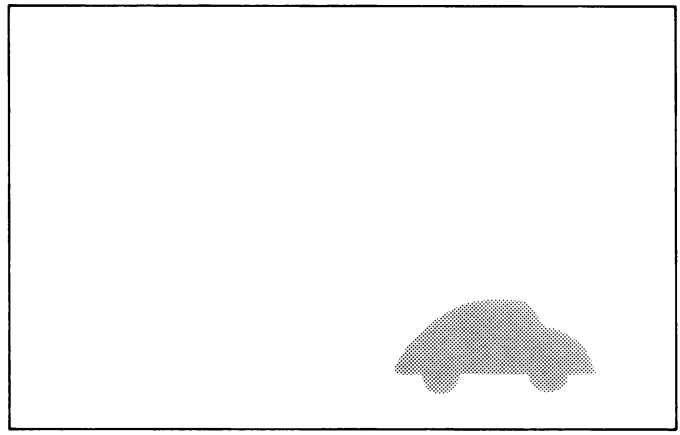
Una differenza importante è che il programma entra in un loop quando ha terminato. Quando si preme un tasto qualsiasi, esso ritornerà allo stato "normale" della VIC-II. Usando la grafica ad alta risoluzione, bisogna eseguire quest'operazione nel programma: non è possibile fare i POKE direttamente dalla tastiera, perché questa funziona solamente nel modo standard. Questo programma è anch'esso troppo lento per animazioni veloci; nella prossima sezione si parlerà di una tecnica per il movimento fluido degli oggetti che può essere impiegata con il BASIC.

6.7. Grafica sprite

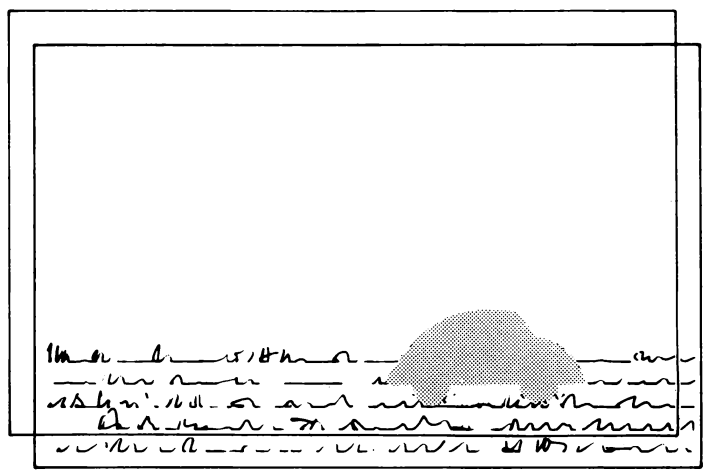
L'uso dei caratteri personali per animare i programmi presenta alcuni svantaggi: mentre le parti di un giocatore possono essere mosse singolarmente un po' come singoli punti, il giocatore nel suo complesso può essere mosso solamente di un intero carattere per volta. Si può aggirare questo ostacolo definendo diverse versioni del giocatore in differenti posizioni nelle dimensioni X e Y, ma questa tecnica consumerebbe i 256 caratteri personali possibili molto rapidamente. C'è un metodo più semplice per ottenere il movimento fluido di un giocatore: l'uso degli "*sprite*" o "folletti".
Gli sprite sono una forma speciale di giocatore, molto simile a quelli costruiti con i caratteri personali, dai quali si differenziano però in modo rilevante: sono completamente indipendenti dal resto del video, possono sovrapporsi a qualunque carattere già esistente e muoversi sul video senza influire sul contenuto. Per capire come funzionano gli sprite, immaginate uno sfondo dipinto su una lastra di vetro:



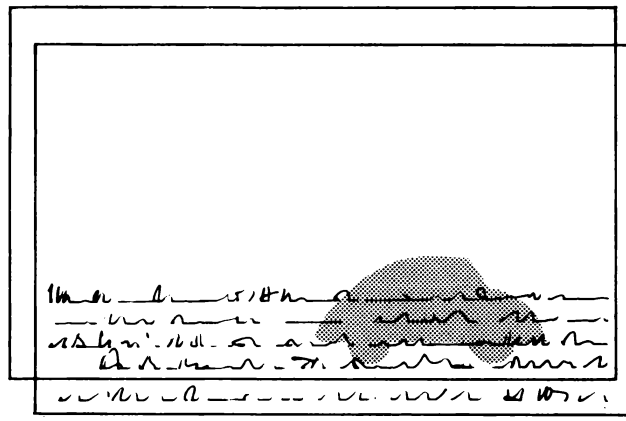
Benché su questo video si possano disporre dei giocatori costituiti da caratteri standard o personali, per ora ci si limiterà, per semplificare, ad assumere che si tratti interamente di sfondo. Su una seconda lastra di vetro si disegnerà lo sprite, un'automobile:



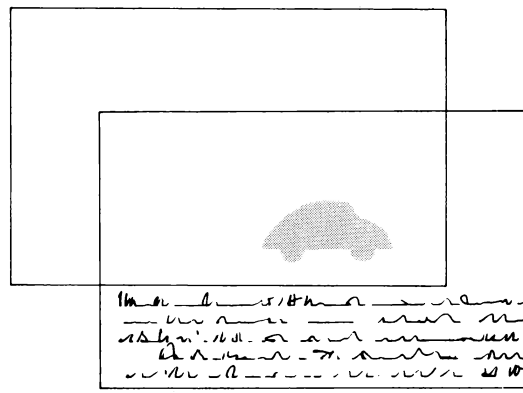
Dato che il vetro è trasparente, si può posizionare la macchina davanti allo sfondo:



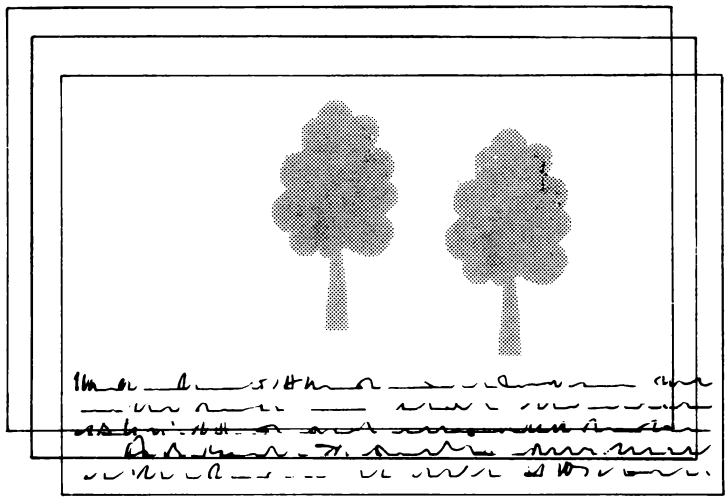
o dietro di esso:



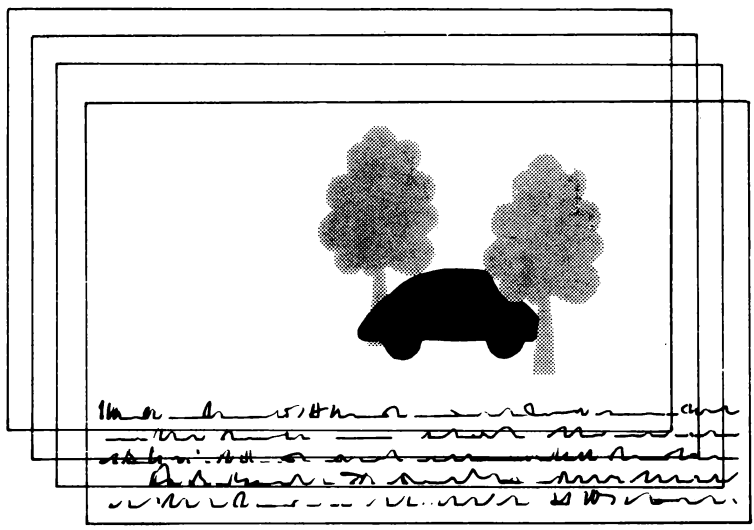
Si può anche muovere il vetro con lo sprite, lasciando lo sfondo immobile:



La differenza tra gli sprite e gli altri oggetti definiti fino ad ora consiste nel fatto che il programma non ha bisogno di spostare lo sprite all'interno della memoria per farlo muovere sul video. Il programma informa semplicemente la VIC-II della posizione sul video dove desidera lo sprite, usando POKE per assegnare questa posizione a certe locazioni di memoria. Questo fa sì che gli sprite siano molto facili da muovere. Per movimentare un po' il video, si possono aggiungere altri pannelli di vetro, ognuno con il proprio sprite. Per esempio, si possono aggiungere alla scena alcuni alberi:

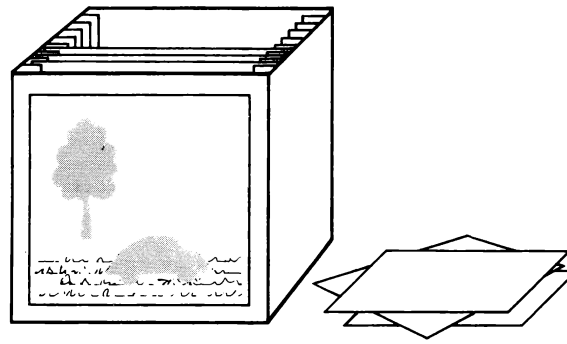


Si possono sistemare questi pannelli in modo che la macchina passi davanti ad uno degli alberi e dietro all'altro:



Durante tutti questi movimenti, il video di sfondo rimane immutato e non è richiesta particolare attenzione da parte del programmatore per te-

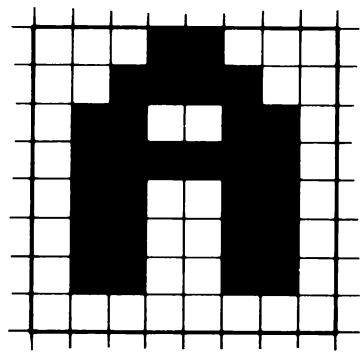
ner conto di ciò che appare in ogni particolare settore del video: la scheda VIC-II ne tiene conto automaticamente. Possono apparire sul video contemporaneamente fino ad 8 sprite e se ne possono avere molti altri già definiti in memoria "in attesa dietro le quinte". Può essere d'aiuto immaginare il video come una "cornice" che può tenere fino ad 8 pannelli di vetro alla volta, con altri pannelli accanto pronti ad essere sostituiti a quelli già posizionati.



Ora vediamo più approfonditamente come funzionano gli sprite e come si possono dipingere i pannelli per poi posizionarli nella "cornice".

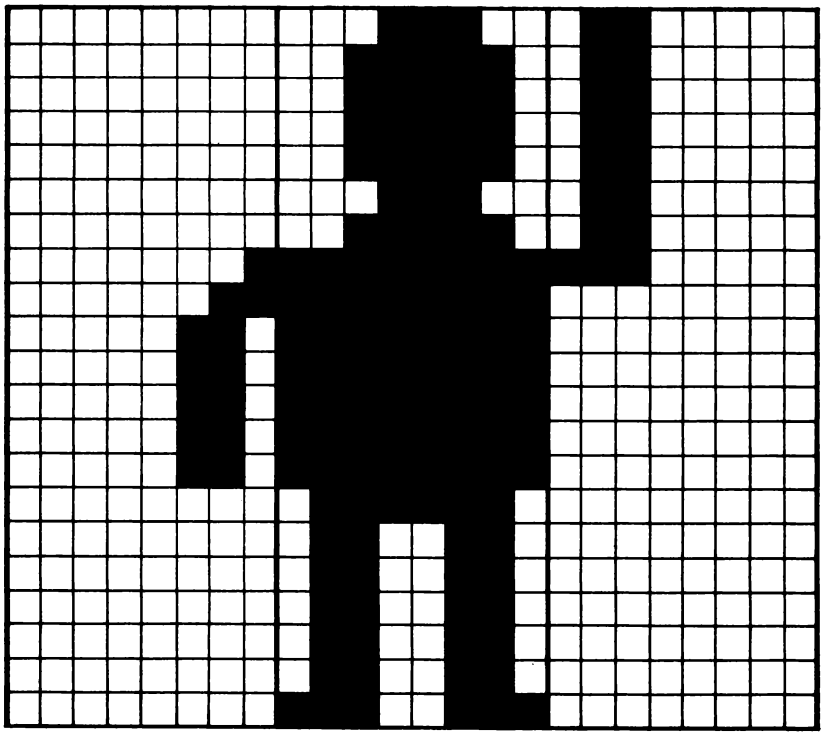
La visualizzazione degli sprite

Gli sprite vengono visualizzati in maniera molto simile ai normali caratteri. Come già spiegato, ogni carattere è definito da una tabella in memoria, con i bit che ordinano alla VIC-II quali punti dello schermo accendere o spegnere. Ecco, ancora una volta, la tabella che definisce la lettera "A".



Punti visualizzati	*Binario*	*Decimale*
	00011000	24
	00111100	60
	01100110	102
	01111110	126
	01100110	102
	01100110	102
	01100110	102
	00000000	0

La tabella per uno sprite è molto simile, ma più grande: gli sprite sono larghi 24 punti e alti 21. Una tabella sprite è larga perciò 3 byte e alta 21 bit. Riprendiamo ad esempio l'omino disegnato prima in questo capitolo e trasformiamolo in uno sprite.



Notate che i byte vuoti che circondano l'omino sono necessari. Le dimensioni di uno sprite sono fisse, proprio come quelle di un carattere, e gli spazi vuoti sono riempiti da degli zero. Ciò rende uno sprite simile a un oggetto costituito da diversi caratteri personali. Può essere d'aiuto immaginare uno sprite come un carattere sovradimensionato.

LA MEMORIA SPRITE

Le definizioni sprite sono tenute in memoria esattamente come le definizioni dei caratteri: la riga in alto per prima (nelle locazioni di memoria più basse), seguita dalla seconda riga e così via fino alla riga 21. I 3 byte

che memorizzano i 24 punti di ogni riga sono l'uno accanto all'altro in memoria. Se la definizione del nostro omino sprite avesse inizio alla 32768, la definizione apparirebbe così:

32768	0	prima riga
32769	28	
32770	96	
32771	0	seconda riga
32772	62	
32773	96	
32774	0	terza riga
32775	62	
32776	0	

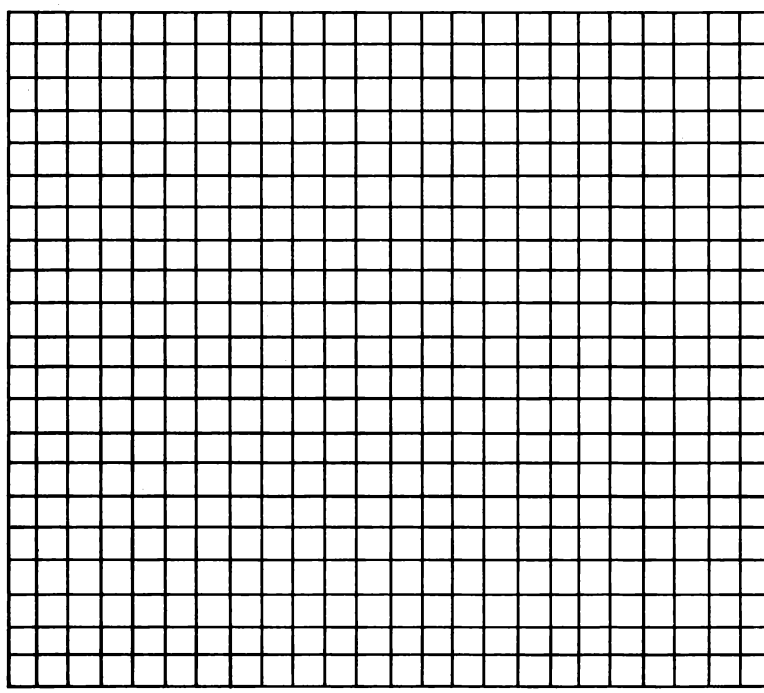
32825	0	20ª riga
32826	102	
32827	0	
32828	0	21ª riga
32829	231	
32830	0	

Questa disposizione in memoria si differenzia da quella di un oggetto simile composto da caratteri personali, tuttavia i valori nella tabella byte sono gli stessi, per cui un oggetto composto da caratteri personali può essere trasformato in uno sprite con poco lavoro. Gli strumenti e le tecniche adottate per il disegno sono molto simili e verranno esaminate più avanti in questo capitolo.
Una grossa differenza tra la memoria sprite ed altre aree già esaminate è che non vi è alcun blocco specifico di memoria riservata agli sprite. I caratteri presentati sul video e le tabelle di definizione dei caratteri sono raggruppati insieme nelle proprie aree di memoria. Si può cambiare la locazione di partenza della memoria del video e dei caratteri, ma queste aree non possono essere frazionate. Non è possibile, per esempio, mettere le definizioni dei primi 128 caratteri alle 32768-33791 e i secondi 128 alle 34816-35839.
Questa limitazione non si applica agli sprite. La scheda VIC-II ha un puntatore separato per ogni sprite alla locazione di memoria dove ha inizio

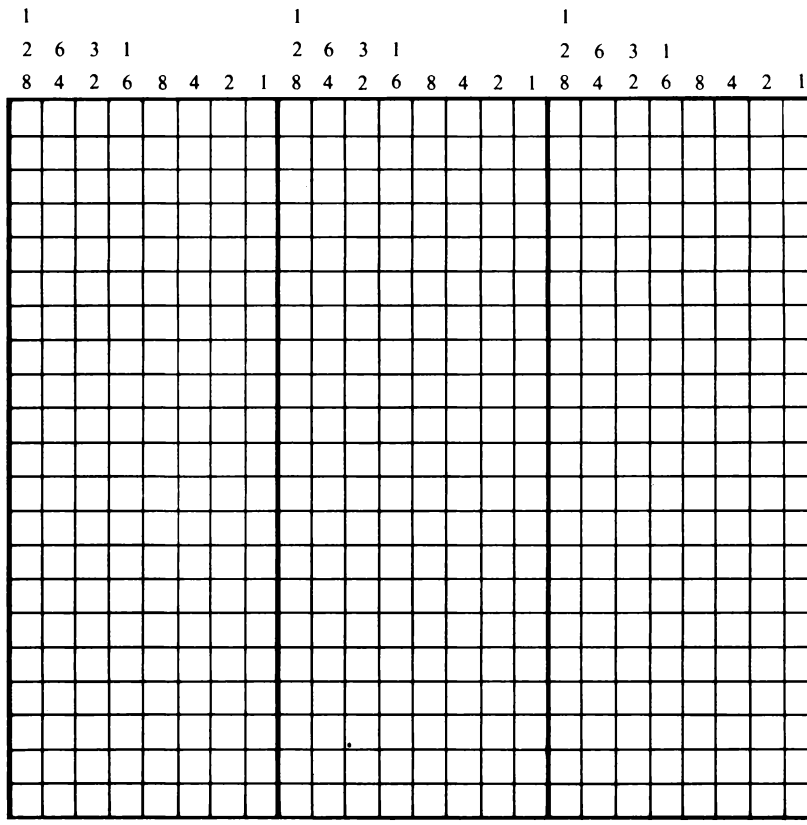
la sua definizione. Per cui, mentre tutte le definizioni per ogni sprite devono essere insieme, i vari sprite possono essere sistemati a piacere in tutto il segmento di 16K di memoria della VIC-II.
Il segreto della flessibilità degli sprite è proprio la possibilità di tenere le loro definizioni distribuite in tutto il segmento. Gli sprite possono essere usati non solamente per estendere la gamma di caratteri, ma anche per animarli. Esamineremo le tecniche di animazione un po' più avanti.

Il disegno degli sprite

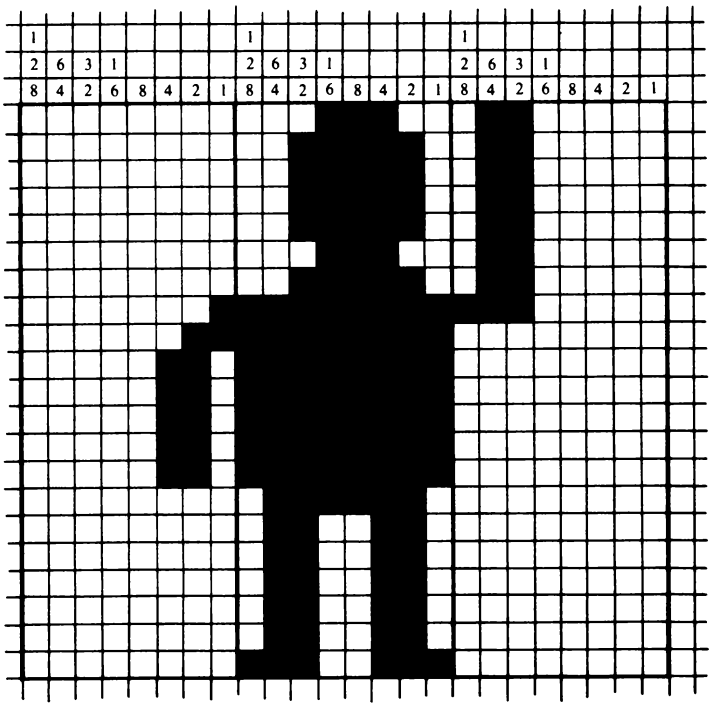
I procedimenti di disegno di uno sprite e di un giocatore costituito da caratteri personali sono molto simili. Incominciamo con un foglio da disegno in carta quadrettata, proprio come per i caratteri personali. Disponete il lato lungo della carta orizzontalmente, e disegnate un riquadro largo 24 quadretti e alto 21, in modo che si presenti così:



Ora dividete il riquadro in 3 colonne di 8 quadretti, e numeratele come segue:



Come per il modulo di progetto dei caratteri personali, ogni quadretto corrisponde ad un punto del video. Le tre colonne corrispondono ai byte della memoria degli sprite. In pratica, il modulo di disegno degli sprite funziona esattamente come quello dei caratteri personali. Ecco di nuovo il nostro omino, disegnato come sprite:



0	28	96
0	62	96
0	62	96
0	62	96
0	62	96
0	28	96
0	62	96
1	255	224
3	255	0
6	255	0
6	255	0
6	255	0
6	255	0
6	255	0
0	126	0
0	102	0
0	102	0
0	102	0
0	102	0
0	102	0
0	231	0

In questo esempio abbiamo omesso i valori in binario: come acquisterete più dimestichezza nella conversione dei punti in numeri, probabilmente farete altrettanto. Abbiamo anche scritto tutti gli 0 per i byte "vuoti": questa è una buona abitudine da sviluppare, altrimenti è facile dimenticarsene.
Se avete acquistato o sviluppato in proprio programmi per disegnare oggetti costituiti da caratteri personali, potrete utilizzarli anche per gli sprite. Ricordatevi soltanto che i byte sono memorizzati in ordine diverso. Ora vediamo le tecniche di programmazione per l'uso degli sprite.

L'uso degli sprite

Una volta disegnato lo sprite, bisogna seguire questa sequenza di operazioni necessarie a farlo funzionare:

1. Riservare la memoria per la definizione dello sprite.
2. Caricare la definizione in memoria.

3. Passare alla scheda VIC-II l'ubicazione in memoria della definizione dello sprite.
4. Ordinare alla VIC-II di iniziare a visualizzare lo sprite.
5. Muovere lo sprite sullo schermo.

Può sembrare una grande quantità di lavoro, ma ognuna di queste fasi è in effetti piuttosto semplice. Nella sezione seguente esamineremo ognuna di queste fasi singolarmente.

ALLOCAZIONE DELLA DEFINIZIONE DEGLI SPRITE IN MEMORIA

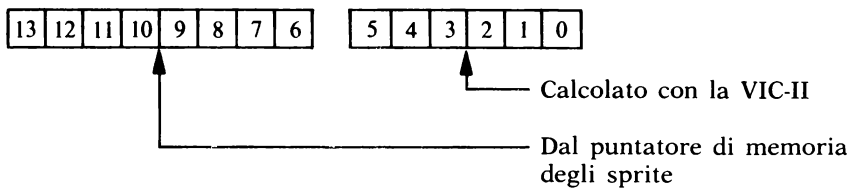
Proprio come per i caratteri personali, le prime fasi in un programma che fa uso di sprite sono dedicate a riservare lo spazio di memoria necessario alle definizioni degli sprite, poi a caricarle in quella memoria. Questo programma carica il nostro "omino" in memoria.

```
100 REM RISERVA LA MEMORIA
110 CLR:POKE 52,128:POKE 56,128:CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 648,132
140 POKE 56576,(PEEK(56576)AND 252) OR 1
150 PRINT "J"
160 REM CARICA LO SPRITE
170 FOR I=32832 TO 32894 :  READ X : POKE I,X : NEXT
180 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
190 DATA 0,28,96,     0,62,96
200 DATA 0,62,96,     0,62,96
210 DATA 0,62,96,     0,28,96
220 DATA 0,62,96,     0,255,96
230 DATA 3,255,192,   6,255,0
240 DATA 6,255,0,     6,255,0
250 DATA 6,255,0,     6,255,0
260 DATA 0,126,0,     0,102,0
270 DATA 0,102,0,     0,102,0
280 DATA 0,102,0,     0,102,0
290 DATA 0,231,0
```

È molto simile a quello usato per caricare un carattere personale. Come già visto, la prima riga del programma fissa i limiti di memoria e esegue un CLR per riassegnare i puntatori interni del BASIC. Il programma esegue poi due POKE nella scheda CIA che controlla il segmento di memoria della VIC-II e il puntatore alla memoria video del BASIC. Infine, il programma usa un loop READ per caricare la definizione dello sprite in memoria.

COME PASSARE ALLA VIC-II L'UBICAZIONE DELLO SPRITE

Ora che la definizione dello sprite è in memoria, la scheda VIC-II deve conoscerne l'ubicazione. Come per la memoria del video e dei caratteri, vi sono puntatori utilizzati per individuare all'interno del segmento di 16K le definizioni di sprite. I puntatori degli sprite sono anch'essi incompleti: alcuni dei bit provengono dalla memoria e altri sono calcolati dalla VIC-II. Invece di fornire solo tre o quattro bit della locazione di memoria, gli indicatori delle definizioni sprite ne forniscono otto.



Anche la formula per calcolare i valori POKE deve cambiare. La seguente istruzione esegue quest'operazione:

```
100 PV=SDA/64 AND 255
```

Il risultato del calcolo, PV, sarà il valore numerico degli 8 bit che va assegnato alla VIC-II. Si divide SDA (Sprite Definition Address — indirizzo di definizione sprite) per 64 perché 6 bit del numero della locazione sono calcolati dalla VIC-II e 64 è uguale a due elevato alla sesta. La divisione elimina quei bit calcolati, mentre l'operazione AND è usata per eliminare i bit alla sinistra. Ricordatevi che vi sono 16 bit nel numero che identifica una locazione di memoria (0-65535), ma solo 14 bit in un numero che ne identifica una nel segmento di 16K della VIC-II. Dato che un byte può contenere solo 8 bit, si devono eliminare gli altri. L'AND assicura che solo valori da 0 a 255 possano essere trasferiti con POKE. I due bit che selezionano i segmenti di 16K corretti sono quelli trasferiti alla scheda CIA con POKE, quindi non c'è pericolo di perderli.
Vediamo ora questa formula in pratica. Iniziamo la definizione per l'omino sprite alla locazione 32832 e osserviamo cosa succede ai bit mentre procede il calcolo.

1000000001000000	32832
	/64
	=
0000001000000001	513
	AND
0000000011111111	255
	=
0000000000000001	1

Il valore corretto per POKE è 1. Cosa sarebbe successo se si fosse deciso di iniziare la definizione dello sprite alla locazione 32833? Il valore binario di 32833 è 1000000001000001. L'ultimo bit verrebbe perso nel calcolo, per cui la VIC-II interpreterebbe l'inizio della definizione come 32832. Questo genererebbe un omino dallo strano aspetto, perché gli ultimi 8 punti di ogni riga risulterebbero spostati alla successiva. Il fatto che la scheda VIC-II calcola gli ultimi 6 bit dell'indirizzo, limita le locazioni dove possono iniziare le definizioni degli sprite in memoria.

0
64
128
192
256
320

.

16192
16256
16320

Ora che si conosce il valore per il POKE, è necessario sapere dove va eseguito. Come abbiamo visto prima, i computer sono macchine binarie; perciò, benché vi siano solo 1000 posizioni visibili nella memoria del video, la VIC-II interpreta la memoria del video come fosse composta da un'area di 1024 byte. I puntatori degli sprite sono memorizzati in alcuni dei 24 byte in più, per cui la mappa della memoria del video in effetti si presenta così:

Indirizzo	*Contenuto*
0	Riga 0, colonna 0
1	Riga 0, colonna 1
2	Riga 0, colonna 2
3	Riga 0, colonna 3
.	
.	
997	Riga 24, colonna 37
998	Riga 24, colonna 38
999	Riga 24, colonna 39
1000	Non usato
1001	Non usato
.	
.	
.	
1014	Non usato
1015	Non usato
1016	Puntatore sprite 0
1017	Puntatore sprite 1
1018	Puntatore sprite 2
1019	Puntatore sprite 3
1020	Puntatore sprite 4
1021	Puntatore sprite 5
1022	Puntatore sprite 6
1023	Puntatore sprite 7

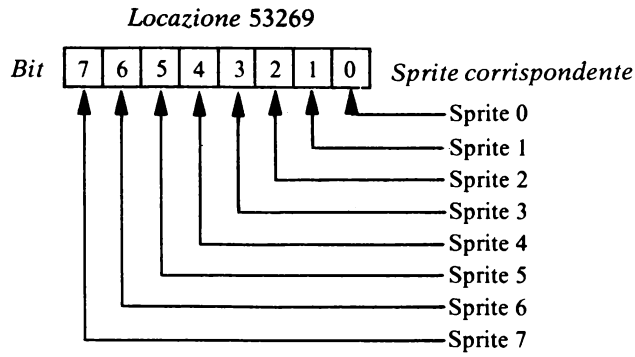
Negli esempi, si usa il segmento che inizia alla 32768 e la memoria del video con inizio alla 33792, quindi gli indicatori degli sprite iniziano alla locazione 34808. Per definire lo sprite 0 dell'omino, il punto corretto per il POKE è 34808. L'istruzione per indicare alla VIC-II la definizione dello sprite è

```
100 POKE 34808,1
```

Caricate ed eseguite ora queste istruzioni. Nulla cambierà sul video per il momento, perché va ancora eseguita qualche fase prima che lo sprite sia pronto all'uso.

ATTIVAZIONE E DISATTIVAZIONE DEGLI SPRITE

Gli sprite devono essere attivati (accesi) e disattivati (spenti) dal vostro programma. Ciò significa che non è necessario definire uno sprite vuoto nel caso in cui non vi sia bisogno di tutti e 8 gli sprite contemporaneamente e che, ancora più importante, è possibile costruire lo sprite senza che appaia sul video prima di essere finito. Gli sprite sono attivati e disattivati dai bit nella 53269, che è inclusa nella scheda VIC-II:



Se il bit che controlla uno sprite è acceso (ha un valore 1), lo sprite è attivato e appare sul video. Se il bit è spento lo sprite è disattivato. Per accendere o spegnere il bit di un particolare sprite senza disturbare gli altri, usate AND e OR per mascherare i bit come descritto prima. Per esempio, per attivare lo sprite 1 si può usare l'istruzione:

```
100 POKE 53269,PEEK(53269) OR 2
```

Ricordatevi che è necessario fare un PEEK al valore esistente nel byte, per ottenere i valori degli altri bit da ritrasferire con POKE. Per disattivare uno sprite usate AND per spegnere il suo bit. L'istruzione

```
100 POKE 53269,PEEK(53269) AND 254
```

disattiverà lo sprite 0, facendolo sparire dallo schermo. L'istruzione precedente esegue anch'essa prima un PEEK in modo da proteggere il valore degli altri bit. Seguono le maschere da usare per attivare e disattivare gli sprite:

Numero sprite	*Per attivarlo, usare OR con*	*Per disattivarlo, usare AND con*
0	1	254
1	2	253
2	4	251
3	8	247
4	16	239
5	32	223
6	64	191
7	128	127

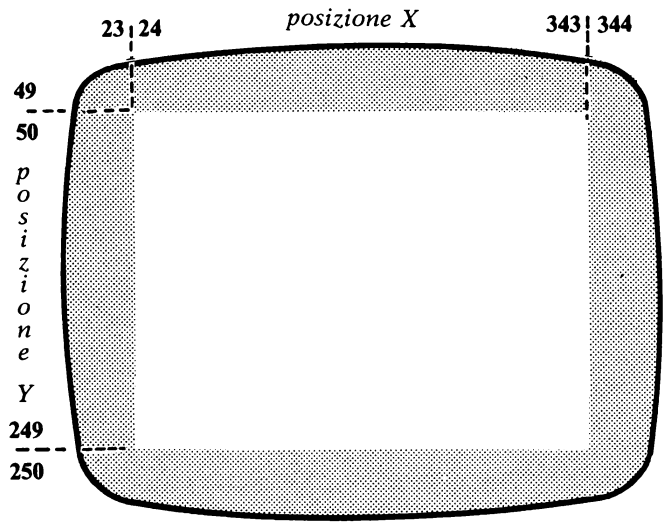
Ora attivate lo sprite caricando la seguente istruzione:

```
POKE 53269,PEEK(53269) OR 1
```

Ancora una volta non accade nulla sul video. Lo sprite è effettivamente attivato, ma dietro al margine; è ora necessario ordinare alla VIC-II di portarlo sul video.

MOVIMENTO DEGLI SPRITE

Da quanto descritto finora, avrete capito che le posizioni degli sprite sono espresse in punti e che i punti, come le righe e le colonne dei caratteri, sono numerati dall'angolo superiore sinistro a quello inferiore destro. Esiste una caratteristica del posizionamento degli sprite che può sorprendere: la numerazione non inizia nella parte visibile del video. Il punto con il numero più basso è in effetti situato nella zona della cornice! Esistono delle buone ragioni anche se a prima vista questo può sembrare strano. Per la maggior parte delle volte, non si vorrà che lo sprite appaia improvvisamente sul video. Normalmente lo sprite entrerà in scena da dietro e per far ciò bisogna poterlo posizionare in quella zona dello schermo.
La numerazione si presenta così:



Per evitare confusione con le posizioni dei caratteri sul video, andrà fatto riferimento al numero dei punti come la "posizione X" per la dimensione

orizzontale e la "posizione Y" per quella verticale, invece di riga e colonna. Il vostro programma controlla le posizioni X e Y degli sprite per mezzo di POKE nelle locazioni VIC-II. Ogni sprite ha la sua coppia di locazioni.

Numero sprite	*Locazione posizione X*	*Locazione posizione Y*
0	53248	53249
1	53250	53251
2	53252	53253
3	53254	53255
4	53256	53257
5	53258	53259
6	53260	53261
7	53262	53263

Per ottenere lo sprite 0 caricate queste istruzioni:

```
POKE 53248,100
POKE 53249,100
```

Provate alcuni ulteriori POKE e osservate cosa succede alla posizione dello sprite. In particolare provate a trasferire con POKE un valore di 255 alla 53248, la posizione X dello sprite. Come avrete osservato, non è possibile posizionarlo vicino al lato destro del video. Date un'occhiata al disegno raffigurante le posizioni X e Y sullo schermo e vedrete il motivo. Una posizione X del valore di 255 (il numero più alto che è possibile inserire in un byte) farà muovere uno sprite solo di circa tre quarti del video. Per arrivare al margine occorre un numero a 9 bit che può contenere valori da 0 a 511. Dato che un byte può contenere solo 8 bit, la VIC-II deve ottenere il nono bit da un'altra locazione. Sono necessari altri 8 bit (1 per ogni sprite), e la Commodore li ha raggruppati in un'unica cella della VIC-II. Questi bit sono nella stessa sequenza dei bit di attivazione, permettendo l'uso delle stesse maschere di bit usate per attivare e disattivare gli sprite. Per comprendere il meccanismo, studiate una subroutine completa che posiziona uno sprite sul video. A questa subroutine sono passate 3 variabili.

SN	il numero di sprite (da 0 a 7)
XP	la posizione X dello sprite
YP	la posizione Y dello sprite

Una subroutine del genere si presenterebbe così:

```
10000 XL = 53248 + 2 * SN
10010 YL = XL + 1
10020 IF XP > 255 THEN 10050
10030 B9 = PEEK(53264) AND NOT (2↑SN)
10040 GOTO 10060
10050 B9 = PEEK(53264) OR (2↑SN)
10060 XV = XP AND 255
10070 POKE XL,XV : POKE  YL,YP : POKE 53264,B9
```

Questa subroutine è probabilmente poco chiara. Esaminiamola perciò riga per riga:

10000 Calcola la locazione di memoria per la posizione X da usare con POKE.

10010 Calcola la locazione di memoria per la posizione Y da usare con POKE.

10020 Devono essere eseguite operazioni di mascheramento differenti, a seconda del valore del 9° bit nella posizione X. Se XP è minore di 256 il 9° bit è uno 0.

10030 B9 è posta uguale al nuovo valore per la locazione "bit 9" nella VIC-II. L'elevamento a potenza fornisce una maschera nella quale solo il bit dello sprite desiderato è un 1. Il PEEK preleva il valore attuale e l'OR accende il bit.

10040 Il GOTO salta le istruzioni necessarie per una posizione X inferiore a 256.

10050 Se il 9° bit è uno 0, si deve spegnere il bit nella locazione "bit 9". Usando un NOT si genera una maschera nella quale il bit dello sprite è uno 0 e tutti gli altri sono degli 1. L'AND spegne il bit.

10060 L'AND con 255 assicura che non si cerchi, con POKE, di trasferire nella posizione X un numero troppo grande.

10070 Questa riga trasferisce con POKE informazioni sulla posizione alla VIC-II.

L'ultima riga di questa subroutine è molto importante. Non è buona norma di programmazione abbinare istruzioni su una riga, perché ciò rende i programmi più difficili da leggere. Una eccezione già menzionata è un piccolo loop FOR-NEXT, come i loop di ritardo usati per l'animazione, dove mettere tutto su una riga rende il programma più facile da seguire. Nella subroutine esaminata sopra, si sono abbinati i POKE per una ragione diversa: ci vuole tempo per fare i calcoli dei valori di POKE. Calcolare i valori per ogni POKE separatamente, causerà un moto discontinuo dello sprite, specialmente quando la posizione X va da 255 a 256. Se si calcola il POKE della posizione X, poi si calcola il valore del bit 9, lo sprite sparisce dal video duran-

te il calcolo, per poi riapparire quando si trasferisce con POKE il valore del bit 9. Ciò può causare un notevole lampeggio, a seconda dell'ordine in cui i calcoli e i POKE vengono eseguiti. Non è necessario eseguire i calcoli nell'ordine sopra citato, ma si troverà che i risultati migliori sono ottenuti eseguendo prima i calcoli e poi i POKE.

ANIMAZIONE DEGLI SPRITE

Come i caratteri personali, gli sprite possono essere animati definendone diverse versioni da sostituire "al volo". Vi è tuttavia una differenza; mentre è possibile ridefinire parte di un oggetto a caratteri personali, è necessario definire uno sprite completamente nuovo per ogni posizione nella sequenza di animazione. Nella sezione precedente si è animato l'omino facendolo salutare. Segue un programma che compie la stessa azione con l'omino sprite.

```
900 REM RISERVA LA MEMORIA
1000 CLR:POKE 52,128:POKE 56,128:CLR
1010 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
1020 POKE 648,132
1030 POKE 56576,(PEEK(56576)AND 252) OR 1
1040 PRINT "□"
1100 REM  CARICA LO SPRITE 1
1200 FOR I=32832 TO 32894 : READ X : POKE I,X : NEXT
1300 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
1400 DATA 0,28,96,     0,62,96
1401 DATA 0,62,96,     0,62,96
1402 DATA 0,62,96,     0,28,96
1403 DATA 0,62,96,     0,255,96
1404 DATA 3,255,192,   6,255,0
1405 DATA 6,255,0,     6,255,0
1406 DATA 6,255,0,     6,255,0
1407 DATA 0,126,0,     0,102,0
1408 DATA 0,102,0,     0,102,0
1409 DATA 0,102,0,     0,102,0
1410 DATA 0,231,0
2100 REM CARICA LO SPRITE 2
2200 FOR I=32896 TO 32958 : READ X : POKE I,X : NEXT
2300 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
2400 DATA 0,28,0,      0,62,0
2401 DATA 0,62,3,      0,62,6
2402 DATA 0,62,12,     0,28,24
2403 DATA 0,62,48,     0,255,224
2404 DATA 3,255,192,   6,255,0
2405 DATA 6,255,0,     6,255,0
2406 DATA 6,255,0,     6,255,0
2407 DATA 0,126,0,     0,102,0
2408 DATA 0,102,0,     0,102,0
2409 DATA 0,102,0,     0,102,0
```

```
2410 DATA 0,231,0
3100 REM CARICA LO SPRITE 3
3200 FOR I=32960 TO 33022 : READ X : POKE I,X : NEXT
3300 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
3400 DATA 0,28,0,      0,62,0
3401 DATA 0,62,0,      0,62,0
3402 DATA 0,62,0,      0,28,0
3403 DATA 0,62,255,    0,255,255
3404 DATA 3,255,192,   6,255,0
3405 DATA 6,255,0,     6,255,0
3406 DATA 6,255,0,     6,255,0
3407 DATA 0,126,0,     0,102,0
3408 DATA 0,102,0,     0,102,0
3409 DATA 0,102,0,     0,102,0
3410 DATA 0,231,0
4100 REM ATTIVA LO SPRITE
4110 POKE 34808,1
4120 POKE 53269,PEEK(53269) OR 1
4200 REM LO METTE SULLO SCHERMO
4210 POKE 53248,100
4220 POKE 53249,100
4300 REM COMINCIA A FARLO MUOVERE
4310 POKE 34808,2
4320 FOR I = 1 TO 250   NEXT
4330 POKE 34808,3
4340 FOR I = 1 TO 250 : NEXT
4350 POKE 34808,2
4360 FOR I = 1 TO 250 : NEXT
4370 POKE 34808,1
4380 FOR I = 1 TO 250   NEXT
4390 GOTO 4300
```

Le righe da 43ØØ a 438Ø sono la chiave dell'animazione: si esegue un POKE alla locazione 34808, che è il puntatore per lo sprite 0. Non ha importanza di quanto si modifichi lo sprite; è sufficiente un solo POKE. È molto più facile animare uno sprite che un oggetto a caratteri personali se i movimenti devono essere complessi, come, ad esempio, una figura che cammina e saluta contemporaneamente. Vi è tuttavia un inconveniente: il metodo sprite normalmente richiede più memoria. Nella maggior parte dei casi, questo non è un problema, ma è un fattore da tenere a mente quando si scrivono i programmi. Aggiungendo al programma le seguenti istruzioni, si arricchirà la presentazione:

```
4222 XP=100:YP=100:SN=0
4224 XI=3:YI=3
4300 REM COMINCIA A FARLO MUOVERE
4310 POKE 34808,2
4320 GOSUB 4400
```

```
4330 POKE 34808,3
4340 GOSUB 4400
4350 POKE 34808,2
4360 GOSUB 4400
4370 POKE 34808,1
4380 GOSUB 4400
4390 GOTO 4300
4400 FOR I=1 TO 5
4410 XP=XP+XI
4420 YP=YP+YI
4430 GOSUB 10000
4440 IF XP<21 OR XP>320 THEN XI=-XI
4450 IF YP<51 OR YP>228 THEN YI=-YI
4460 FOR J=1 TO 25:NEXT
4470 NEXT I
4480 RETURN
10000 XL = 53248 + 2 * SN
10010 YL = XL + 1
10020 IF XP > 255 THEN 10050
10030 B9 = PEEK(53264) AND NOT (2↑SN)
10040 GOTO 10060
10050 B9 = PEEK(53264) OR (2↑SN)
10060 XV = XP AND 255
10070 POKE XL,XV : POKE YL,YP   POKE 53264,B9
10080 RETURN
```

Quando si esegue il programma, l'omino comincerà di nuovo a salutare, ma questa volta si muoverà anche sul video. Quando arriva al margine, rimbalza e si muove in una direzione diversa. Useremo poi versioni modificate di questo programma per illustrare altri concetti di sprite. Fate una copia con SAVE del programma per evitare di doverlo ribattere.

COLORAZIONE DEGLI SPRITE

Come ogni locazione di memoria del video ne ha una corrispondente nella memoria del colore, così ogni sprite ha una locazione che contiene il colore. I colori degli sprite sono memorizzati nella scheda VIC-II. Le locazioni sono:

Locazione	*Numero dello sprite*
53287	0
53288	1
53289	2
53290	3
53291	4
53292	5
53293	6
53294	7

Queste locazioni trasferiscono gli stessi valori POKE delle locazioni della memoria del colore. Per osservare i colori degli sprite in azione, cambiate il programma come segue:

```
100 REM RISERVA LA MEMORIA
110 CLR:POKE 52,128:POKE 56,128:CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 648,132
140 POKE 56576,(PEEK(56576)AND 252) OR 1
150 PRINT "□"
160 REM LOAD SPRITE 1
170 FOR I=32832 TO 32894 : READX : POKE I,X : NEXT
180 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
190 DATA 0,28,96,     0,62,96
200 DATA 0,62,96,     0,62,96
210 DATA 0,62,96,     0,28,96
220 DATA 0,62,96,     0,255,96
230 DATA 3,255,192,   6,255,0
240 DATA 6,255,0,     6,255,0
250 DATA 6,255,0,     6,255,0
260 DATA 0,126,0,     0,102,0
270 DATA 0,102,0,     0,102,0
280 DATA 0,102,0,     0,102,0
290 DATA 0,231,0
300 REM CARICA LO SPRITE 2
310 FOR I=32896 TO 32958 : READ X : POKE I,X : NEXT
320 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
330 DATA 0,28,0,      0,62,0
340 DATA 0,62,3,      0,62,6
350 DATA 0,62,12,     0,28,24
360 DATA 0,62,48,     0,255,224
370 DATA 3,255,192,   6,255,0
380 DATA 6,255,0,     6,255,0
390 DATA 6,255,0,     6,255,0
400 DATA 0,126,0,     0,102,0
410 DATA 0,102,0,     0,102,0
420 DATA 0,102,0,     0,102,0
430 DATA 0,231,0
440 REM CARICA LO SPRITE 3
450 FOR I=32960 TO 33022 : READ X : POKEI,X : NEXT
460 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
470 DATA 0,28,0,      0,62,0
480 DATA 0,62,0,      0,62,0
490 DATA 0,62,0,      0,28,0
500 DATA 0,62,255,    0,255,255
510 DATA 3,255,192,   6,255,0
520 DATA 6,255,0,     6,255,0
530 DATA 6,255,0,     6,255,0
540 DATA 0,126,0,     0,102,0
550 DATA 0,102,0,     0,102,0
560 DATA 0,102,0,     0,102,0
570 DATA 0,231,0
```

```
580 REM ATTIVA LO SPRITE
590 POKE 34808,1
600 POKE 53269,PEEK(53269) OR 1
610 REM LO METTE SULLO SCHERMO
620 SN=0
630 XP=100 : XI=1
640 YP=100 : YI=1
650 GOSUB 10000
660 REM COMINCIA A MUOVERLO
670 POKE 34808,2
680 GOSUB 760
690 POKE 34808,3
700 GOSUB 760
710 POKE 34808,2
720 GOSUB 760
730 POKE 34808,1
740 GOSUB 760
750 GOTO 660
760 FOR I = 1 TO 5
770 XP = XP+XI
780 YP = YP+YI
790 GOSUB 10000
800 IF XP<21 OR XP > 320 THEN XI = -XI : GOSUB 860
810 IF YP<51 OR YP > 228 THEN YI = -YI : GOSUB 860
830 NEXT I
840 RETURN
850 REM LO FA DIVENTARE ROSSO
860 POKE 53287,10
870 FOR J = 1 TO 250   NEXT
880 POKE 53287,1
890 RETURN
10000 XL = 53248 + 2 * SN
10010 YL = XL + 1
10020 IF XP > 255 THEN 10050
10030 B9 = PEEK(53264) AND NOT (2↑SN)
10040 GOTO 10060
10050 B9 = PEEK(53264) OR (2↑SN)
10060 XV = XP AND 255
10070 POKE XL,XV : POKE YL,YP   POKE 53264,B9
10080 RETURN
```

Ora eseguite il programma ed osservatelo. Ogni volta che l'omino urta la parete diventa rosso per un attimo prima di continuare.

INTERAZIONE DEGLI SPRITE

Quando gli sprite si muovono sullo schermo, a volte si sovrappongono fra di loro e ad oggetti sullo sfondo. Come nella realtà, due sprite non possono occupare lo stesso spazio allo stesso tempo. Un solo oggetto può essere pre-

sentato in un particolare punto allo stesso tempo. La VIC-II possiede una serie di regole per determinare le priorità tra gli oggetti che vengono presentati. Possiede anche un modo di indicare al programma quando due oggetti entrano in collisione. Nella prossima sezione tratteremo proprio di questo.

PRIORITÀ DI VISUALIZZAZIONE

All'inizio del paragrafo gli sprite sono stati presentati come figure su lastre di vetro. Ogni lastra può contenere un solo sprite, quindi quando due cercano di occupare la stessa posizione, uno deve passare "dietro" l'altro. La regola di priorità (passare sopra) è molto semplice: lo sprite 0 passa sempre sopra allo sprite 1, il quale passa sopra allo sprite 2, e così via. Per osservare questa priorità, caricate ed eseguite il seguente programma:

```
100 REM RISERVA LA MEMORIA
110 CLR:POKE 52,128:POKE 56,128:CLR
120 REM ASSSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 648,132
140 POKE 56576,(PEEK(56576) AND 252) OR 1
150 PRINT "□"
160 REM CARICA LO SPRITE 0 (ROMBO)
170 FOR I=32832 TO 32894 : READ X : POKE I,X : NEXT
180 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
190 DATA 0,0,0,       0,0,0
200 DATA 0,0,0,       0,0,0
210 DATA 0,24,0,      0,60,0
220 DATA 0,126,0,     0,255,0
230 DATA 1,255,128,   3,255,192
240 DATA 7,255,224,   7,255,224
250 DATA 3,255,192,   1,255,128
260 DATA 0,255,0,     0,126,0
270 DATA 0,60,0,      0,24,0
280 DATA 0,0,0,       0,0,0
290 DATA 0,0,0
300 REM CARICA LO SPRITE 1 (QUADRATO)
310 FOR I=32896 TO   32958 : POKE I,255   NEXT
320 REM ATTIVA GLI SPRITES
330 POKE 34808,1
340 POKE 34809,2
350 POKE 53269,PEEK(53269) OR 3
360 REM LI METTE SULLO SCHERMO
370 X(0)=50 : XI(0)=3
380 X(1)=50 : XI(1)=-3
390 YP=150
400 FOR SN=0 TO 1 : XP=X(SN) : GOSUB 10000   NEXT
410 REM COMINCIA A FARLI MUOVERE
420 FOR SN = 0 TO 1
```

```
430 X(SN) = X(SN)+XI(SN)
440 XP=X(SN)
450 IF X(SN)<21 OR X(SN)> 75 THEN XI(SN) = -XI(SN)
460 GOSUB 10000
480 NEXT
490 GOTO 410
10000 XL = 53248 + 2 * SN
10010 YL = XL + 1
10020 IF XP > 255 THEN 10050
10030 B9 = PEEK(53264) AND NOT (2↑SN)
10040 GOTO 10060
10050 B9 = PEEK(53264) OR (2↑SN)
10060 XV = XP AND 255
10070 POKE XL,XV : POKE YL,YP : POKE 53264,B9
10080 RETURN
```

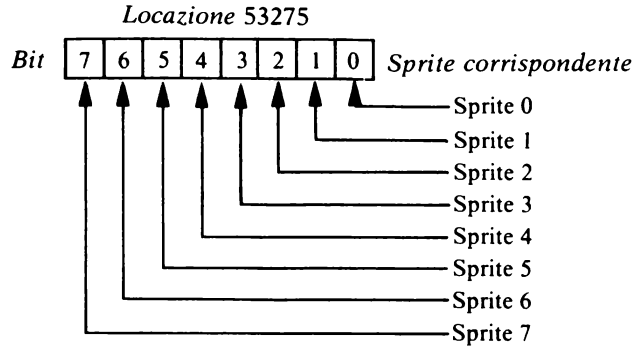
Quando il programma viene eseguito, esso crea due semplici sprite e li muove avanti e indietro sul video. Il rombo è lo sprite 0 e passa sempre sopra al quadrato. Notate che la parte "trasparente" del rombo non cancella il quadrato; solo la parte "visibile" si sovrappone ai punti che formano il quadrato. Interrompete il programma e modificate queste righe:

```
330 POKE 34808,2
340 POKE 34809,1
```

I POKE hanno invertito gli sprite, il quadrato è diventato sprite 0 e il rombo sprite 1. Ora il rombo passa sotto al quadrato. Si noti che non fa differenza dove è situato lo sprite in memoria; l'unica cosa importante è quale puntatore di sprite lo indica. Inoltre, gli sprite possono passare anche sopra o sotto lo sfondo. Per illustrare ciò, aggiungiamo al programma le seguenti righe:

```
355 REM METTE IL QUADRATO DIETRO LO SFONDO
356 POKE 53275,PEEK(53275) OR 2
```

Quando si esegue il programma, si vedrà che il quadrato (sprite 1) passa dietro allo sfondo, ma il rombo (sprite 0) passa sopra ad entrambi. Il trucco è il POKE a riga 356 che agisce su un bit, il quale indica che lo sprite 1 ha priorità sullo sfondo. Così come molti degli altri bit che controllano gli sprite, ve ne sono 8, raggruppati in una unica locazione numero 53275. Il bit più a destra controlla lo sprite 0 e quello più a sinistra controlla lo sprite 7:



Se il bit di uno sprite è 0, ha priorità sullo sfondo. Se il bit è 1 passa sotto allo sfondo. I bit sono, naturalmente, controllati con operazioni di mascheramento. Come sempre si deve fare un PEEK allla locazione 53275 per primo, per proteggere i valori degli altri bit. Per mettere lo sprite 3 dietro allo sfondo, si usa un'istruzione come

```
100 POKE 53275,PEEK(53275) OR 8
```

Per posizionare lo sprite 7 davanti allo sfondo, si usa

```
100 POKE 53275,PEEK(53275) AND 127
```

L'indipendenza delle priorità tra gli sprite stessi e tra gli sprite e lo sfondo procura un interessante effetto collaterale. Sostituite riga 356 con

```
356 POKE 53275,1
```

Eseguite il programma ed osservate quello che accade. Il quadrato (sprite 1) è sopra allo sfondo e il rombo (sprite 0) sotto. Tuttavia il rombo passa sopra al quadrato perché sprite 0 ha sempre priorità su sprite 1, quindi quando il rombo e il quadrato s'incontrano lo sfondo è sopra a quella parte del quadrato che è coperta dal rombo.

COLLISIONI

Nell'eventualità di "collisioni", la scheda VIC-II segnala chi ha colpito che cosa assegnando dei flag. La possibilità di poter identificare le collisioni è molto utile nei programmi di giochi e in qualsiasi programma che usa gli sprite per simulare oggetti in movimento. Fare altrettanto calcolando le posizioni d'incontro richiederebbe molto tempo e rallenterebbe l'azione. Vi sono due tipi di collisione: uno si verifica quando si incontrano due sprite e

l'altro quando si incontrano uno sprite e lo sfondo. Ogni tipo di collisione viene memorizzato in un'apposita locazione: la 53279 segna le collisioni tra gli sprite e lo sfondo, la 53278 segna le collisioni tra gli sprite. Come molte altre locazioni nella VIC-II, ognuna contiene un bit per ogni sprite, con il bit più basso corrispondente allo sprite 0 e quello più alto allo sprite 7. Ciò permette l'uso delle stesse maschere per esaminare i bit di "controllo" usati. Il seguente esempio usa gli sprite dell'esempio precedente, ma questa volta essi non si sovrappongono: quando si incontrano, rimbalzano nella direzione opposta. Sono aggiunti due muri sullo sfondo contro cui rimbalzano gli sprite:

```
100 REM RISERVA LA MEMORIA
110 CLR:POKE 52,128:POKE 56,128:CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 648,132
140 POKE 56576,(PEEK(56576) AND 252) OR 1
150 PRINT "□"
160 REM CARICA LO SPRITE 0 (ROMBO)
170 FOR I=32832 TO 32894 : READ X : POKE I,X : NEXT
180 REM DATI DELLO SPRITE, 2 RIGHE PER ISTRUZIONE
190 DATA 0,0,0,       0,0,0
200 DATA 0,0,0,       0,0,0
210 DATA 0,24,0,      0,60,0
220 DATA 0,126,0,     0,255,0
230 DATA 1,255,128,   3,255,192
240 DATA 7,255,224,   7,255,224
250 DATA 3,255,192,   1,255,128
260 DATA 0,255,0,     0,126,0
270 DATA 0,60,0,      0,24,0
280 DATA 0,0,0,       0,0,0
290 DATA 0,0,0
300 REM CARICA LO SPRITE 1 (QUADRATO)
310 FOR I=32896 TO   32958 : POKE I,255   NEXT
312 REM COSTRUISCE I MURI
313 FOR I=1 TO 8:PRINT:NEXT
314 FOR I=1 TO 10:PRINT"▒                    ▒":NEXT
320 REM ATTIVA GLI SPRITE
330 POKE 34808,1
340 POKE 34809,2
350 POKE 53269,PEEK(53269) OR 3
360 REM LI METTE SULLO SCHERMO
370 X(0)=60 : XI(0)=6
380 X(1)=35 : XI(1)=-3
390 YP=150
400 FOR SN=0 TO 1 : XP=X(SN) : GOSUB 10000 : NEXT
410 REM COMINCIA A FARLI MUOVERE
420 FOR SN = 0 TO 1
430 X(SN) = X(SN)+XI(SN)
440 XP=X(SN)
460 GOSUB 10000
465 REM CONTROLLA LA COLLISIONE CON L'ALTRO SPRITE
```

```
470 IF PEEK(53278)=0 THEN GOTO 600
475 REM COLLISIONE! INVERTE LA DIREZIONE
480 XI(0)=-XI(0):XI(1)=-XI(1)
485 REM FA TORNARE INDIETRO LO SPRITE APPENA MOSSO
490 X(SN)=X(SN)+XIX(SN)*2
500 XP=X(SN)
560 GOSUB 10000
570 REM AZZERA I FLAG DI COLLISIONE DEGLI SPRITE
580 DD=PEEK(53278)
590 REM CONTROLLA LA COLLISIONE CON LO SFONDO
600 IF(PEEK(53279)AND 2↑SN)=0 THEN GOTO 690
610 REM COLLISIONE COL MURO!INVERTE LA DIREZIONE
620 XI(SN)=-XI(SN)
630 REM FA TORNARE INDIETRO LO SPRITE APPENA MOSSO
640 X(SN)=X(SN)+XI(SN)
650 XP=X(SN)
660 GOSUB 10000
670 REM AZZERA IL FLAG DI COLLISIONE DELLO SFONDO
680 DD=PEEK(53279)
690 NEXT SN
700 GOTO 420
10000 XL = 53248 + 2 * SN
10010 YL = XL + 1
10020 IF XP > 255 THEN 10050
10030 B9 = PEEK(53264) AND NOT (2↑SN)
10040 GOTO 10060
10050 B9 = PEEK(53264) OR (2↑SN)
10060 XV = XP AND 255
10070 POKE XL,XV : POKE YL,YP   POKE 53264,B9
10080 RETURN
```

La maggior parte di questo programma vi sarà ormai familiare, per cui descriveremo solo le parti riguardanti le collisioni. La riga 47Ø esegue un PEEK alla 53278, che contiene il flag di collisione tra sprite. Nell'esempio vi sono solo due sprite, per cui qualsiasi valore diverso da 0 significa che i due sono entrati in collisione. Un programma più complesso, con molti sprite, userebbe il mascheramento di bit per determinare quali si sono scontrati. Per esaminare la collisione tra gli sprite 3 e 5 si può usare l'istruzione

```
100 IF PEEK(53278) AND 40 = 40 THEN GOTO 200
```

Da dove proviene il 40? Il valore di maschera per lo sprite 3 è 8 e quello dello sprite 5 è 32. Sommando 8 e 32 si ottiene una maschera che azzera tutti i bit ad eccezione di quelli per gli sprite 3 e 5. Il confronto con 40 è importante: verifica che entrambi i bit siano accesi. Se sprite 3 si è scontrato con sprite 2, il valore PEEK deve essere 12 (8+4). Se 12 e 40 sono entrambi esaminati da AND, il risultato è 8. Questo valore non è 0, per cui un semplice IF riporterebbe un risultato "vero". Paragonando il risultato dell'AND con 40

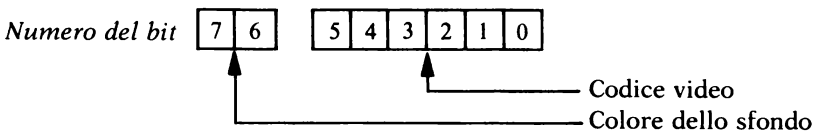
si vedrebbe se entrambi gli sprite, o solamente uno di essi, sono interessati alla collisione.
Il vostro programma dovrebbe controllare i flag di collisione dopo ogni movimento degli sprite. Se non lo fa, potrebbe essere difficile determinare chi ha urtato che cosa. Per esempio, se gli sprite 3 e 2 sono in collisione nell'angolo superiore sinistro dello schermo e gli sprite 0 e 4 sono contemporaneamente in collisione in basso a destra, un PEEK alla 53278 riporterebbe un valore di 15, perché tutti e quattro sono coinvolti in collisioni. Al fine di determinare quali sprite si toccano fra loro, il vostro programma deve affidarsi ai calcoli che il meccanismo d'esame delle collisioni è preposto ad eliminare. Si evita questo lavoro extra controllando dopo ogni movimento.
La riga 58Ø esemplifica un'altra importante caratteristica dei flag di collisione. L'istruzione REM a riga 57Ø informa che si azzerano i flag, e ci si aspetta che questa operazione sia fatta con un POKE e non con un PEEK. Il motivo è da trovarsi nel disegno della scheda VIC-II. Il flag è attivato ogniqualvolta lo sprite è coinvolto in una collisione. Il bit rimane attivo finché non viene esaminato con un PEEK. Se il flag è azzerato con PEEK, perché eseguire di nuovo PEEK? Appena i flag sono azzerati con il primo PEEK, la scheda VIC-II scopre che gli sprite sono ancora in collisione, per cui attiva di nuovo i flag. Per azzerare i flag si è dovuto spostare lo sprite e poi fare un altro PEEK. Se i flag non fossero azzerati si avrebbe una falsa indicazione: anche se i due sprite non sono più in contatto, la VIC-II comunicherebbe lo stato di collisione. Provate a cancellare dal programma la riga 58Ø e osservate il risultato. Il programma ritiene che i due sprite siano sempre in collisione; una volta che si sono toccati, rimangono sul posto e vibrano, continuando a collidere.
Benché non si sia ancora accennato ai flag di collisione tra sprite e sfondo, il loro schema è lo stesso di quelli tra sprite e sprite. Sarà necessario usare gli stessi metodi per azzerarli ed evitare false collisioni.

6.8. Più colore sullo schermo

Mentre nella maggior parte dei casi la combinazione dei caratteri grafici standard e personali è sufficiente, può non esserlo la gamma dei colori base. Avendo bisogno di sfumature di colore, non è sufficiente accendere e spegnere un punto.
Il C-64 offre due tecniche per rendere più colorato il video. La prima, chiamata "Extended Color Mode" permette di controllare il colore dello sfondo per ogni carattere sul video. La seconda, "Multicolor Mode", permette di usare più di due colori in un carattere. Il modo multicolore può essere usato anche per presentazioni video ad alta risoluzione e per gli sprite.

EXTENDED COLOR MODE (ECM)

Vi sono due modi di attirare l'attenzione su un messaggio usando i caratteri normali: presentarlo in un colore diverso o stamparlo in caratteri inversi. Essi sono efficaci se l'operatore è attivamente impegnato nel leggere il video, ma vi sono situazioni in cui è necessario un richiamo più evidente. Un caso del genere potrebbe essere l'uso del C-64 per controllare un esperimento in laboratorio. Potrebbe essere necessario per lo sperimentatore allontanarsi dal computer e il messaggio d'allarme potrebbe essere particolarmente importante. L'uso dell'Extended Color Mode assolve egregiamente a questa funzione.
Con l'Extended Color Mode sia lo sfondo che i caratteri sono colorati individualmente. Il colore del carattere è conservato nella memoria del colore, come al solito. Per assegnare il colore dello sfondo, la scheda VIC-II preleva due bit dalla memoria del video.



I due bit che controllano il colore dello sfondo sono usati per selezionare una delle quattro locazioni della scheda VIC-II.

Coppia di bit	*Locazione scelta*
00	53281 (Colore dello sfondo 0)
01	53282 (Colore dello sfondo 1)
10	53283 (Colore dello sfondo 2)
11	53284 (Colore dello sfondo 3)

Il colore dello sfondo 0 è il colore normale di sfondo per il video. Queste quattro locazioni, come la memoria del colore, memorizzano un codice di colore da 0 a 15. I bit da 4 a 7 vengono ignorati quando si esegue un POKE e si deve operare un AND tra il valore e 15 quando si esegue un PEEK. Questa flessibilità nella scelta del colore dello sfondo comporta uno svantaggio. Con i bit 6 e 7 impegnati per il colore dello sfondo, i rimanenti 6 bit permettono solo 64 caratteri diversi. È necessario "raggirare" il computer per usare i caratteri inversi. Infine l'uso di colori extra richiede un po' di lavoro di programmazione, compensato da un programma più facilmente utilizzabile.

Attivazione dell'Extended Color Mode

L'ECM, come molte delle caratteristiche grafiche già discusse, è controllato da un bit della scheda VIC-II. Il video è predisposto in ECM quando il bit 6 della locazione 53265 è acceso. Provate ad eseguire:

```
POKE 53265,PEEK(53265) OR 64
```

Si nota come prima cosa che il cursore, invece di cambiare da blu scuro a blu chiaro (da normale a inverso), cambia da blu scuro a rosso. Ricordatevi che il C-64 spegne il bit 7 e lo riaccende per far lampeggiare il cursore. Sul video standard ciò modifica il carattere da normale a inverso mentre in un video in ECM, cambiare il bit 7 modifica la locazione da cui è prelevato il colore. Quando il C-64 è riportato alla serie dei colori normali, le quattro locazioni dei colori di sfondo sono fissate in blu scuro, bianco, rosso e azzurro. Uno spazio "inverso" (codice schermo 160) diventa uno spazio rosso in ECM, perché i bit 7 e 6 contengono i valori 1 e 0 rispettivamente prelevando il colore dello sfondo dalla locazione 52383. Caricate ed eseguite questo programma per ottenere una chiara impressione degli effetti in ECM:

```
100 REM TEST EXTENDED COLOR MODE
110 PRINT"□"
120 REM RIEMPIE LO SCHERMO
130 SB=256*PEEK(648)
140 FOR I=0 TO 255 : POKE SB+I,I : NEXT
150 REM RIEMPIE LA MEMORIA DEL COLORE
160 FOR I=0 TO 255 : POKE 55296+I,1 : NEXT
170 REM DISPONE I COLORI
180 POKE 53282,4
190 POKE 53283,5
200 POKE 53284,9
210 REM ACCENDE E SPEGNE L'EXTENDED COLOR MODE
220 POKE 53265,PEEK(53265) OR 64
230 FOR I=1 TO 800 : NEXT
240 POKE 53265,PEEK(53265) AND 191
250 FOR I=1 TO 800 : NEXT
260 GOTO 220
```

Il programma riempie i primi 256 byte di memoria del video con tutti i 255 codici video possibili, poi accende e spegne l'ECM. Con ECM spento il video contiene tutti i 128 caratteri maiuscoli e grafici seguiti dalle loro forme inverse. Con ECM acceso, il video contiene 4 gruppi di 64 caratteri, ciascuno con un colore di sfondo differente. I 64 caratteri che appaiono sul video sono quelli con i codici video da 0 a 63: le lettere, i numeri e la punteggiatura. Questi sono gli unici caratteri che si possono usare per i messaggi che sono

presentati in ECM (l'appendice E contiene una tabella dei codici video di tutti i caratteri). I 6 bit più bassi di un byte possono contenere numeri fino a 63. Un POKE con un numero maggiore nella memoria del video influenzerà i bit 6 e 7, cambiando il colore dello sfondo.

Creare visualizzazioni in ECM con POKE

Il POKE in ECM funziona come per i caratteri standard o personali. L'unica differenza è che si deve aggiungere il valore appropriato del colore nei bit 6 e 7. Nel presente capitolo si è usata una subroutine per convertire codici CHR$ in codici video. Eccone una versione modificata che esegue la conversione e poi aggiunge i bit di codice del colore appropriato. Come la precedente, questa preleva un carattere dalla variabile KV$ e riporta il codice del video in SC. Per questa subroutine è necessaria una nuova variabile, BC, che contiene il colore di sfondo (da 0 a 3) per il carattere.

```
12000 REM CONVERTE I CARATTERI PER ECM
12010 SC=ASC(KV$)
12020 REM COPRE I CARATTERI NON APPROPRIATI
12030 IF (SC<32) OR (SC>95) THEN SC=32+64*BC   RETURN
12040 IF SC>63 THEN SC=SC-64+64*BC : RETURN
12050 SC=SC+64*BC : RETURN
```

Questa subroutine è molto più corta, perché devono essere tradotti solo due gruppi di 32 caratteri mentre gli altri sono trasformati in spazi vuoti.

Uso di PRINT con ECM

A prima vista può sembrare che si possano stampare con PRINT caratteri con colori di sfondo diversi. Non vi sono invece comandi BASIC per trattare l'ECM direttamente; aggiungere i bit di controllo del colore ai caratteri trasformerebbe semplici messaggi in lunghe stringhe di CHR$ illeggibili. Tuttavia, con qualcuno dei "trucchi" citati in precedenza, si può usare PRINT per visualizzare messaggi in ECM, ed essere sempre in grado di leggere il programma.
La chiave del controllo del colore dello sfondo sta nell'influire sui valori dei bit 6 e 7 nella memoria video. Controllare il bit 7 è facile: stampando con PRINT i caratteri di comando REVERSE ON e REVERSE OFF il bit diventa 1 o 0 per i caratteri seguenti. Più difficile è controllare il bit 6; se si stampano con PRINT direttamente tutti i 64 caratteri utilizzabili in ECM, il bit è sempre 0. Così si otterrebbe solamente il colore di sfondo normale più il colore 2 (per i

caratteri inversi). Questo può essere sufficiente per alcuni programmi, ma volendo una maggiore flessibilità è necessario il seguente "trucco": per usare i colori di sfondo 1 e 3 assegnate un valore di 1 al bit 6 per i caratteri da stampare in quei colori. A questo scopo si userà una tecnica di "traduzione" simile a quella usata per convertire i caratteri in codici video. Questa traduzione sarà necessaria in molti punti del programma (ovunque si voglia stampare un messaggio), per cui è stata scritta in forma di subroutine:

```
12000 REM CONVERTE LA STRINGA PER ECM
12010 REM REVERSE SE BC=2 O 3
12020 PS$="■" : IF BC >= 2 THEN PS$="■"
12030 FOR CP=1 TO LEN (MS$)
12040 PC=ASC(MID$(MS$,CP,1))
12050 REM CONVERTE UN CARATTERE
12060 REM ESEGUE IL CONTROLLO DEI CARATTERI
12070 IF (PC AND 127) <32 THEN PS$=PS$+CHR$(PC)
12075 RETURN
12080 REM COPRE I CARATTERI NON APPROPRIATI
12090 IF PC>95 THEN PC = 32
12100 IF BC=0 OR BC=2 THEN 12150
12110 REM FORZA IL BIT 6 A 1 SE BC=1 O 3
12120 IF PC>63 THEN PC=PC+32
12130 IF PC<64 THEN PC=PC+128
12140 REM AGGIUNGE BYTE ALLA STRINGA PRODOTTA
12150 PR$=PS$+CHR$(PC)
12160 NEXT
12170 RETURN
```

La subroutine preleva una *stringa di messaggio*, MS$ e il colore di sfondo, BC e li usa per costruire una *stringa di stampa*, PS$. La stringa di messaggio è il testo del messaggio che si vuole stampare, in semplici caratteri leggibili. La stringa di stampa

```
400 PRINT PS$
```

è una stringa di caratteri che sono stati tradotti, per stamparli sul corretto colore di sfondo. Come già accennato, queste istruzioni sono piuttosto complesse, le spiegheremo quindi dettagliatamente.
Il bit 7 nella memoria del video è controllato dai caratteri RVS ON e RVS OFF. La riga 12Ø2Ø inizia la stringa di stampa con un RVS OFF se il colore di sfondo è 0 o 1 (bit 7 spento), oppure con un RVS ON per i colori di sfondo 2 e 3 (bit 7 acceso). Le righe 12Ø3Ø e 1216Ø formano un loop FOR-NEXT che fa avanzare, un carattere alla volta, la variabile di indice CP sulla stringa di messaggio. Le righe comprese nel loop convertono un carattere della stringa di messaggio e lo aggiungono alla stringa di stampa. Siccome gli operatori Booleani ed aritmetici del BASIC funzionano solo con numeri, la riga 12Ø4Ø

converte i caratteri da tradurre in numeri per mezzo della funzione ASC. La riga 12Ø7Ø controlla il carattere per verificare se è uno dei caratteri di controllo (RETURN, comandi del colore ecc.). I caratteri di controllo hanno tutti i valori compresi tra 0 e 31 e tra 128 e 159. È necessario un solo confronto per identificare tutti i caratteri di controllo se si esegue un AND tra il valore del carattere e 127: trovando un carattere di controllo *bypassa* il resto dei caratteri della traduzione e li fa passare inalterati.
I caratteri che si possono visualizzare hanno un valore ASCII compreso tra 32 e 95. Sono eliminati tutti quelli con valore inferiore a 32 con il test per i caratteri di controllo e infine la riga 12Ø9Ø converte tutti quelli con valore superiore a 95 in spazi.
Se il colore di sfondo è 0 o 2 il bit 7 verrà fissato opportunamente dal codice inverso a riga 12Ø2Ø e il bit 6 a 0. Quindi la riga 121ØØ salta il resto del testo per questi colori.
Per i colori di sfondo 1 e 3, è necessario assicurarsi che il bit 6 sia acceso nella memoria del video cambiando il codice ASCII del carattere con uno che dia il giusto valore al bit 6. Per trovare la soluzione si devono consultare le tabelle dei codici ASCII e video nell'appendice E, con il risultato di costruire la seguente tabella:

MS$ Stringa messaggio	*Codici video*	*Stringa di stampa*	*Operazione*
32-63	96-127	160-191	+128
64-95	64-95	96-127	+ 32

La prima colonna è il valore ASCII del carattere nella stringa di messaggio. La seconda colonna è il codice video necessario per presentare quel carattere con il bit acceso. Confrontando le tabelle dei codici del video e dei caratteri, si sono trovati i valori ASCII necessari ad accedere a quei codici del video e l'operazione BASIC per compiere la traduzione. Con questi dati si sono scritte le righe 1212Ø e 1213Ø.
La variabile PC contiene il valore del codice video desiderato. La riga 1215Ø chiude il loop riconvertendo PC a un valore di stringa e concatenando quest'ultima alla stringa di stampa. Fatto ciò, si ripete il loop fino a che MS$ non è completamente tradotta, poi si esegue il RETURN. Per osservare la subroutine in azione, caricatela insieme al seguente programma ed eseguite:

```
100 REM TEST EXTENDED COLOR MODE
110 POKE 53265,PEEK(53265) OR 64
120 POKE 53281,14
130 POKE 53282,1
140 POKE 53283,5
150 POKE 53284,7
160 PRINT"■"
```

```
170 MS$="MESSAGGIO TEST BLU"
180 BC=0
190 GOSUB 340
200 PRINT PS$
210 MS$="MESSAGGIO TEST BIANCO"
220 BC=1
230 GOSUB 340
240 PRINT PS$
250 MS$="MESSAGGIO TEST VERDE"
260 BC=2
270 GOSUB 340
280 PRINT PS$
290 MS$="MESSAGGIO TEST GIALLO"
300 BC=3
310 GOSUB 340
320 PRINT PS$
330 END
340 REM CONVERTE LA STRINGA PER ECM
350 REM IN MODO INVERSO SE BC = 2 O 3
360 PS$="■" : IF BC >= 2 THEN PS$="■"
370 FOR CP=1 TO LEN(MS$)
380 PC=ASC(MID$(MS$,CP,1))
390 REM CONVERTE UN CARATTERE
400 REM ESEGUE IL CONTROLLO DEI CARATTERI
410 IF (PC AND 127) < 32 THEN PS$=PS$+CHR$(PC)
    RETURN
420 REM COPRE I CARATTERI NON APPROPRIATI
430 IF PC > 95 THEN PC = 32
440 IF BC=0 OR BC=2 THEN 490
450 REM FORZA IL BIT 6 A UNO SE BC=1 O 3
460 IF PC>63 THEN PC=PC+32
470 IF PC<64 THEN PC=PC+128
480 REM AGGIUNGE IL CARATTERE ALLA STRINGA
490 PS$=PS$+CHR$(PC)
500 NEXT
510 RETURN
```

Il programma fissa il colore dello sfondo nelle righe 12Ø-15Ø e presenta un messaggio in ogni colore. Per ogni colore il procedimento è lo stesso: assegna il messaggio a MS$, fissa il colore di sfondo in BC, chiama la subroutine e stampa PS$. Si sarebbe potuto mettere l'istruzione PRINT nella subroutine, ma è stata lasciata fuori per avere così maggior flessibilità. Dato che è il programma principale ad eseguire il PRINT, esso può utilizzare caratteristiche come TAB, o terminare l'istruzione con un ";" per continuare a stampare sulla stessa riga.
Potreste perfino mischiare i colori di sfondo sulla stessa riga aggiungendo un ";" alla fine della riga 24Ø; provate e vedrete.

Come far lampeggiare lo sfondo per richiamare l'attenzione

Una maniera veramente efficace per richiamare l'attenzione su un messaggio importante è di far lampeggiare lo sfondo. Ci si può rendere conto di quanto sia efficace questa soluzione osservando il cursore: per quanto lo schermo sia pieno, l'occhio è sempre attirato dal cursore lampeggiante.
Si possono far risaltare messaggi completi cambiando semplicemente il valore in una delle locazioni del colore di sfondo della VIC-II. Per vedere questo effetto, aggiungete le seguenti righe al programma dell'ultima sezione:

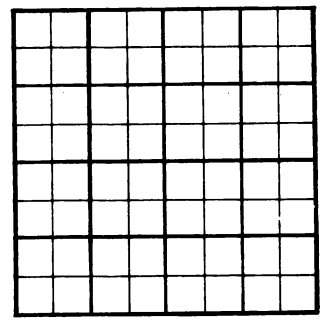
```
322 REM FA LAMPEGGIARE LO SFONDO
323 POKE 53284,14
324 FOR I=1 TO 500 : NEXT
325 POKE 53284,7
326 FOR I=1 TO 500   NEXT
327 GOTO 323
```

Quando si esegue il programma, esso presenta i messaggi come prima, ma lo sfondo del MESSAGGIO TEST GIALLO lampeggia. Il segreto sta nelle righe 323-326. Esse cambiano il valore del colore di sfondo 3 da 7 (giallo) a 14 (blu) e viceversa. Quando lo sfondo è blu, è uguale al resto dello schermo e sparisce. Quando invece diventa giallo, lo sfondo risulta "acceso". Anche il più distratto degli utenti non potrebbe non notare una cosa simile, specialmente se abbinata ad alcuni degli effetti sonori che verranno proposti nel capitolo 7.
Ad esempio, un programma che controlla tramite un'interfaccia un qualche dispositivo, potrebbe far cambiare il colore di un messaggio da rosso a giallo a verde per evidenziare che la regolazione del dispositivo, secondo i dati rilevati dal computer, si avvicina ai parametri desiderati.
Tutti gli esempi preposti usano i caratteri standard, ma si possono usare anche quelli personali.
La limitazione di 64 caratteri è però sempre in vigore, quindi è possibile rimanere a corto di caratteri se il vostro programma usa anche alcuni dei caratteri standard per i messaggi.

MODO MULTICOLORE

Il modo multicolore è adatto per quelle applicazioni che hanno bisogno di una presentazione ancor più colorata di quanto sia possibile con l'ECM. Nel modo ad alta risoluzione o ECM vi sono otto punti su ogni riga di un carattere, ma ognuno di essi è vincolato ad essere o del colore del carattere o del colore dello sfondo. Il modo multicolore sacrifica alcuni di quei punti per ottenere dei colori extra. In un carattere multicolore vi sono solo 4 punti

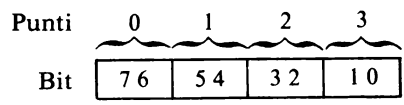
per riga, ma ogni punto può assumere uno a scelta tra quattro colori, invece di due solamente.



Un carattere multicolore ha le stesse dimensioni esterne di uno ad alta risoluzione e ogni punto è grande il doppio. Per ottenere i colori extra del modo multicolore si deve "dipingere con un pennello grosso". Vi sono sempre otto righe nel carattere e l'altezza del punto rimane la stessa. La definizione grafica del carattere è però inferiore come qualità.

Definizione in memoria dei caratteri multicolori

Le posizioni relative dei punti dei caratteri multicolori sono memorizzate nella memoria dei caratteri nello stesso ordine di quelle dei caratteri ad alta risoluzione, ma ogni riga (byte) dello schema si presenta così:



Ogni punto di un carattere multicolore è rappresentato in memoria da due bit invece di uno. Vi sono quattro possibili combinazioni di questi due bit (00, 01, 10 e 11), che forniscono così quattro colori. I colori selezionati da queste combinazioni sono:

Coppia di bit	*Colore scelto*
00	Colore dello sfondo 0
01	Colore dello sfondo 1
10	Colore dello sfondo 2
11	Colore del carattere

Le coppie di bit selezionano il colore del doppio punto nello stesso modo in cui i bit 6 e 7 determinano il colore di sfondo in ECM. Si noti, però, che il colore di sfondo 3 non è più disponibile. Invece, quando entrambi i bit sono accesi, il punto è presentato sul video nel colore memorizzato nella memoria dei caratteri nella locazione di quel carattere.

Attivazione del modo multicolore

Come per l'ECM, il modo multicolore è attivato da un bit in una locazione della VIC-II. Il bit interessato è il bit 4 della locazione 53270, quindi l'istruzione

```
100 POKE 53270,PEEK(53270) OR 16
```

attiverà il modo multicolore e

```
100 POKE 53270,PEEK(53270) AND 239
```

lo disattiverà, tornando al modo standard.

Costruzione di un carattere multicolore

Attivare il modo multicolore non significa automaticamente far diventare multicolore tutto lo schermo. Il modo multicolore può essere attivato e disattivato per ogni carattere. Il fatto che un carattere sia in modo colore standard o multicolore dipende dal bit 3 nella memoria del colore. Se questo bit è 0, il carattere è ad alta risoluzione, se è 1, il carattere è in modo multicolore. Questo fa sì che sia possibile alternare caratteri multicolori con caratteri colorati standard o personali. Poiché quel bit è stato prelevato proprio dalla memoria del colore, ne rimangono solo tre per il codice del colore. In una presentazione multicolore, solo gli otto colori primari (nero, bianco, rosso, cyan, viola, verde, blu e giallo) possono essere usati come colori dei caratteri.

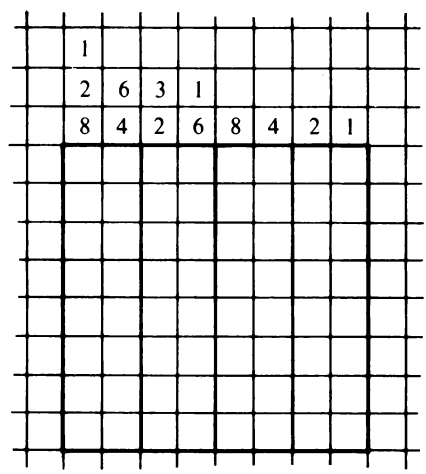
Strumenti per il disegno di caratteri multicolori

Utilizzeremo gli stessi strumenti dei caratteri personali per produrre i caratteri multicolori. I programmi e le tecniche descritte precedentemente, e quelli sviluppati da voi stessi, serviranno per il disegno di caratteri multicolori. Tuttavia saranno necessarie piccole modifiche. Ad esempio, per assi-

curarvi che i caratteri dei messaggi di prova siano in multicolore, modificate il programma principale "SETUP" come segue:

```
100 REM RISERVA LA MEMORIA
110 POKE 52,128 : POKE 56,128 : CLR
120 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
130 POKE 56576,(PEEK(56576) AND 252) OR 1
140 POKE 53272,32
150 REM E LO COMUNICA AL BASIC
160 POKE 658,136
170 PRINT"□■#ABCDEFGHIJKLMNOPQRSTUVWXYZ[£]↑←!";
180 PRINT CHR$(34);CHR$(34);CHR$(20);
190 PRINT "##$%&'()*+,-.";
200 PRINT "/0123456789:;<=>?"
210 REM COPIA LA ROM DEI CARATTERI NELLA RAM
220 POKE 56334,PEEK(56334) AND 254
230 POKE 1,PEEK(1) AND 251
240 FOR I=0 TO 2047:POKE 32768+I,PEEK(53248+I)   NEXT
250 POKE 1,PEEK(1) OR 4
260 POKE 56334,PEEK(56334) OR 1
270 REM MODO MULTICOLORE
280 POKE 53270,PEEK(53270) OR 16
290 REM RENDE MULTICOLORI I CARATTERI
300 FOR I=55296 TO 55359 : POKE I,PEEK(I) OR 8   NEXT
```

Lo schema di disegno deve anch'esso essere modificato, in modo da avere quattro doppi punti per riga.



Il calcolo dei valori POKE non è influenzato dal modo multicolore. Le colonne vanno sommate esattamente come per i caratteri personali. La modifica nel raggruppamento è fatta solamente per facilitare la visualizzazione del carattere come apparirà sullo schermo. Per disegnare un carattere multicolore sarà più facile usare due fogli di carta. Sul primo disegnate il carattere a colori, ed usate il secondo per convertirli nei loro valori di bit e per effettuare i calcoli. La figura 6.5 raffigura l'omino di prima, a colori.

Esperimenti con caratteri multicolori

Prima di cominciare a disegnare i propri caratteri multicolori, sarà bene farsi un po' d'esperienza con gli effetti dei colori.
Caricate ed eseguite il programma "SETUP" listato sopra. Come si può os-

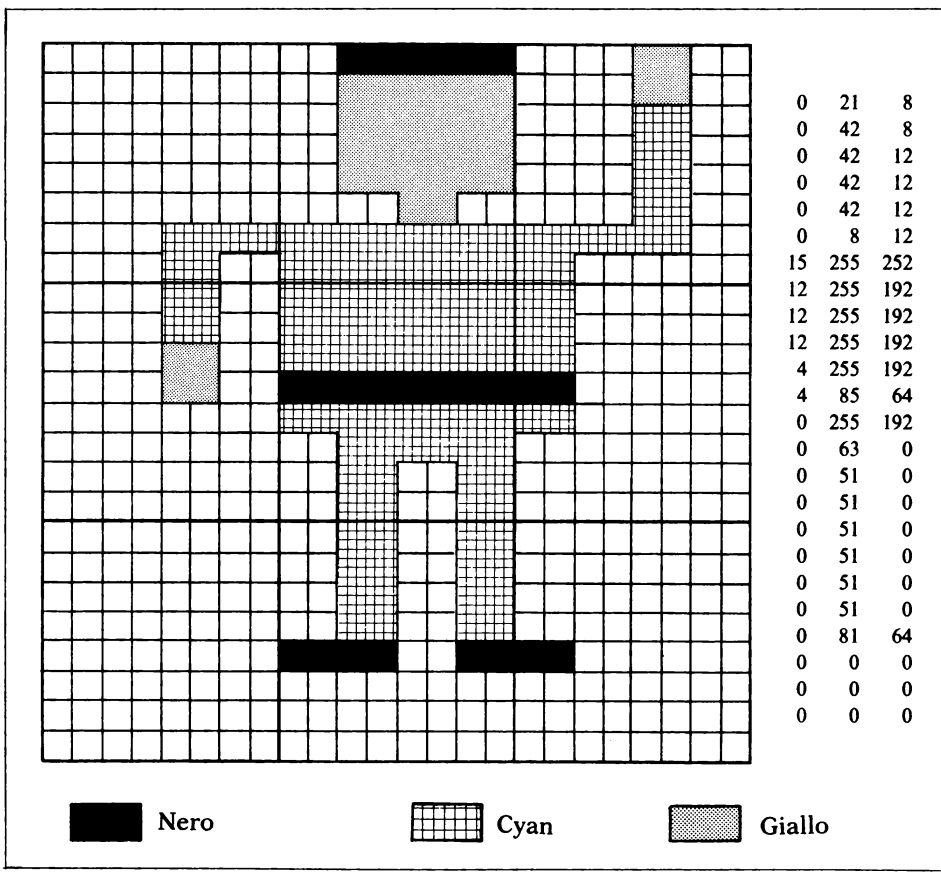


Figura 6.5. Omino a colori

servare, alcuni dei caratteri, specialmente quelli con componenti verticali ed orizzontali, rimangono più o meno riconoscibili. Gli altri sono solo un ammasso di colore. Come regola generale i caratteri ad alta risoluzione devono essere modificati per essere usati nel modo multicolore. Cambiando le definizioni dei punti nella memoria dei caratteri, il modo multicolore tende a distorcere lo schema dei caratteri ad alta risoluzione. Vi sono delle eccezioni; alcuni dei caratteri grafici, specialmente quelli squadrati, cambieranno semplicemente colore.
Si può semplificare l'esperimento caricando la seguente istruzione in modo immediato:

```
POKE 648,9
```

La locazione 648 contiene il valore che il BASIC pone nella memoria del colore per ogni carattere visualizzato. Cambiando questo valore si possono posizionare i caratteri nell'area di lavoro e il BASIC li lascerà nel modo multicolore. Questo significa che anche tutti i messaggi provenienti dal BASIC stesso saranno multicolori, ma non dovrebbe essere difficile riconoscerli.

Come mescolare caratteri multicolori con altri

Dato che il modo multicolore può essere attivato o disattivato per ogni singolo carattere sul video, è possibile mescolare caratteri multicolori, caratteri personali e testo normale sul video. Per esempio provate a mettere le tre versioni dell'omino sullo schermo contemporaneamente.
I caratteri multicolori sono molto utili per disegnare giocatori e campi di gioco con grande versatilità, ma non è tutto. Vedremo in seguito che anche immagini ad alta risoluzione e sprite possono essere multicolori.

VISUALIZZAZIONI MULTICOLORI AD ALTA RISOLUZIONE

Nella sezione riguardante i caratteri multicolori, si è visto che il modo multicolore non cambia la struttura del video, la memoria del colore e quella dei caratteri, ma solamente il modo in cui la scheda VIC-II "interpreta" il contenuto delle locazioni modificate. Lo stesso avviene per le visualizzazioni ad alta risoluzione: le locazioni e gli usi della memoria del video e della memoria dei caratteri sono identici. La grossa novità è che i bit nel modo multicolore vengono usati a coppie.

Come predisporre il video in modo multicolore ad alta risoluzione

Assegnando sia i bit del modo multicolore che quelli che controllano lo schermo ad alta risoluzione otterrete le due caratteristiche combinate. Dato che questi bit sono in due locazioni separate, sono necessari due POKE.

```
100 POKE 53265,PEEK(53265) OR 32
110 POKE 53270,PEEK(53270) OR 16
```

Dopo aver eseguito queste due istruzioni, il video sarà predisposto come previsto. Ora si vedrà cosa farne.

Programmazione di visualizzazioni multicolori ad alta risoluzione

In precedenza abbiamo esaminato una subroutine che calcolava la posizione in cui presentare un bit sul video in base alla sua posizione X-Y (la distanza dai margini superiore e sinistro). Questa tecnica è ritardante per i programmi che ne fanno uso, perché occorre tempo per eseguire il GOSUB e RETURN, ma rende più facile la scrittura e la lettura del programma principale. L'uso di una subroutine per presentazioni multicolori ad alta risoluzione è altrettanto sensata ora, per le stesse ragioni di semplificazione del programma principale, anche se ciò comporta una piccola perdita di velocità. La subroutine per il multicolore è molto simile a quella per l'alta risoluzione e utilizza anch'essa le coordinate X e Y per posizionare il punto sul video. Questa volta, però, il valore di X deve essere tra 0 e 159 perché il video multicolore è largo solo 160 punti. C'è bisogno anche di un'altra variabile, DC, per il colore del punto. Come la variabile BC usata per il colore di sfondo, anche DC avrà un valore da 0 a 3, identificando la fonte del codice del colore e non il codice del colore stesso. Le maschere dei bit per il modo multicolore hanno valori diversi perché si cambiano due bit alla volta. Le maschere per assegnare al punto il colore di sfondo (fissare i due bit a 0) sono:

Punto	0	1	2	3
Coppia di bit	7 6	5 4	3 2	1 0
Maschera	63	207	243	252

Si ricordi che i punti sono numerati da sinistra verso destra, mentre i bit, al contrario, da destra a sinistra. Dato che i bit devono essere azzerati, queste maschere sono usate con l'operatore AND:

```
100 REM STABILISCE I PUNTI PER IL COLORE DELLO SFONDO
110 POKE 27450,PEEK(27450) AND 63 : REM PUNTO 0
```

```
120 POKE 27451,PEEK(27451) AND 207 : REM PUNTO 1
130 POKE 27452,PEEK(27452) AND 243 :REM  PUNTO 2
140 POKE 27453,PEEK(27453) AND 252 : REM PUNTO 3
```

Per fissare il punto sono necessarie 16 maschere differenti (4 colori possibili per ognuno dei quattro punti controllati da un byte):

Punto	0	1	2	3
Sfondo	0	0	0	0
Codice video (4-7)	64	16	4	1
Codice video (0-3)	128	32	8	2
Memoria del colore	192	48	12	3

Le seguenti maschere sono usate per porre i punti uguale a 1, quindi funzionano con OR.

```
100 REM STABILISCE I PUNTI PER GLI ALTRI COLORI
105 REM PUNTO 0 DALLA MEMORIA DELLO SCHERMO BIT 4-7
110 POKE 27450,PEEK(27450) AND 63 OR 64
115 REM PUNTO 1 DALLA MEMORIA DELLO SCHERMO BIT 0-3
120 POKE 27451,PEEK(27451) AND 207 OR 32
125 REM PUNTO 2 DALLA MEMORIA DELLO SCHERMO
130 POKE 27452,PEEK(27452) AND 243 OR 12
```

La subroutine per assegnare i punti in modo multicolore

La prima delle due subroutine listate in seguito (righe 15000-15030) assegna i valori nell'area ad alta risoluzione, ed è quasi identica a quella usata in precedenza. Le differenze stanno nel calcolo delle locazioni (vi sono 4 bit per byte invece di 8) e delle maschere dei bit (si cambiano due bit alla volta invece di uno).
La seconda subroutine (righe 16000-16080) crea le maschere. Deve essere chiamata con un GOSUB posto all'inizio del programma. La riga 16040 è una formula che calcola i valori delle maschere OR. Si potrebbero caricare con istruzioni READ e DATA, ma la formula risparmia spazio in memoria e evita errori di battitura.

```
15000 REM SUBROUTINE CHE ASSEGNA I PUNTI
15010 PL = BM+(40*(Y AND 248))+(Y AND 7)+2*(X AND 252)
15020 POKE PL,PEEK(PL) AND M0%(X AND 3) OR CM%(X AND 3,DC)
15030 RETURN
16000 REM COSTRUISCE LA MASCHERA
16010 DIM CM%(3,3)
16020 FOR I=0 TO 3
```

```
16030 FOR J=0 TO 3
16040 CM%(I,J) = 2↑(2*(3-I))*J
16050 NEXT J
16060 M0%(I) = 255 AND NOT CM%(I,3)
16070 NEXT I
16080 RETURN
```

Come esempio del funzionamento di questa subroutine, ecco il programma del triangolo della sezione precedente, modificato per una presentazione multicolore:

```
100 REM GRAFICA AD ALTA RISOLUZIONE
105 REM PROGRAMMA DIMOSTRATIVO
110 REM RISERVA LA MEMORIA
120 POKE 52,64 : POKE 56,64 : CLR
130 REM ASSEGNA ALLA VIC-II UN SEGMENTO DI MEMORIA
140 POKE 56576,(PEEK(56576) AND 252) OR 2
150 POKE 53272,8
160 REM PREDISPONE LA VIC-II AL MODO MULTICOLORE
165 REM AD ALTA RISOLUZIONE
170 POKE 53265,PEEK(53265) OR 32
180 POKE 53270,PEEK(53270) OR 16
190 REM POSIZIONA IL PUNTATORE ALL'AREA
195 REM AD ALTA RISOLUZIONE
200 BM=24576 : SB=16384
210 REM AZZERA L'AREA AD ALTA RISOLUZIONE
220 FOR I=BM TO BM+7999 : POKE I,0 : NEXT I
230 REM RIEMPIE LA MEMORIA DELLO SCHERMO
235 REM CON I CODICI DEI COLORI
240 FOR I=SB TO SB+999 : POKE I,230 : NEXT I
250 REM COSTRUISCE L'ARRAY MASCHERA
260 GOSUB 16000
270 DC = 1
280 REM TRACCIA LA BASE DEL TRIANGOLO
290 Y=63
300 FOR X=0 TO 63
310 GOSUB 15000
320 NEXT
330 REM TRACCIA IL LATO SINISTRO DEL TRIANGOLO
340 FOR X=0 TO 30
350 Y = 63-X*2
360 GOSUB 15000
370 NEXT
380 REM TRACCIA IL LATO DESTRO DEL TRIANGOLO
390 FOR X=31 TO 62
400 Y = 63-(62-X)*2
410 GOSUB 15000
420 NEXT
430 REM ASPETTA CHE VENGA PREMUTO UN TASTO
440 GET A$ : IF A$="" THEN 440
450 REM RIPORTA IL SISTEMA ALLA NORMALITA'
```

```
460 REM RESTITUISCE AL BASIC L'AREA
465 REM AD ALTA RISOLUZIONE
470 POKE 52,128 : POKE 56,128 : CLR
480 REM RIPORTA LA VIC-II AL MODO STANDARD
485 REM E ALLA MEMORIA DEI CARATTERI
490 POKE 56576,(PEEK(56576) AND 252) OR 3
500 REM RITORNA AL MODO CARATTERE
510 POKE 53265,PEEK(53265) AND 223
520 POKE 53270,PEEK(53270) AND 239
530 POKE 53272,21
540 END
15000 REM SUBROUTINE DI TRACCIAMENTO
15010 PL = BM+(40*(Y AND 248))+(Y AND 7)
      +2*(X AND 252)
15020 POKE PL,PEEK(PL) AND M0%(X AND 3) OR
      CM%(X AND 3,DC)
15030 RETURN
16000 REM COSTRUISCE LA MASCHERA
16010 DIM CM%(3,3)
16020 FOR I=0 TO 3
16030 FOR J=0 TO 3
16040 CM%(I,J) = 2↑(2*(3-I))*J
16050 NEXT J
16060 M0%(I) = 255 AND NOT CM%(I,3)
16070 NEXT I
16080 RETURN
```

Paragonate questo programma alla versione ad alta risoluzione. L'unico cambiamento apportato è stato quello di assegnare un valore alla variabile DC, la variabile del colore del punto. Questo è un ottimo esempio dell'utilità di creare subroutine generalizzate. Naturalmente non tutti i programmi sono così facilmente adattabili dall'alta risoluzione al multicolore, nondimeno l'uso di subroutine generalizzate rende molto più facile e razionale lo sviluppo di nuovi programmi.

SPRITE MULTICOLORI

Gli sprite multicolori sono molto simili ai caratteri multicolori, così come gli sprite stessi somigliano ai caratteri personali. Come per i caratteri multicolori, le definizioni degli sprite multicolori sono composte di coppie di bit, ognuna rappresentante uno tra quattro colori. La differenza tra gli sprite e i modi multicolori è la provenienza dei codici dei colori.

Coppia di bit	*Colori visualizzati*
00	Trasparente
01	Locazione 53285
10	Locazioni 53287-53294
11	Locazione 53286

Le locazioni 53285 e 53286 sono usate solamente per contenere i codici per gli sprite multicolori. Come le altre per il colore, esse hanno solo 4 bit e devono essere mascherate quando il programma esegue un PEEK.

Costruzione di sprite multicolori

La locazione 53276 identifica quali sprite sono multicolori. Come accade nella maggior parte di queste locazioni, ogni sprite ha un bit di controllo in questo byte. Quando è 1, lo sprite è multicolore. Dato che il bit dello sprite nella 53276 determina se esso è multicolore o no, tutti i 4 bit della locazione del colore sono disponibili e quindi vi possono essere memorizzati tutti i 16 codici dei colori.

6.9. Sprite estesi

Dopo aver imparato a disegnare uno sprite, bisogna imparare ad ingrandirlo. Si potrebbero abbinare più sprite come per i caratteri personali, ma vi è una soluzione più facile. La VIC-II può raddoppiare le dimensioni degli sprite con semplici POKE. L'ingrandimento degli sprite è controllato nelle locazioni 53277 e 53271. Anche qui ogni sprite ha un suo bit. Quando il bit nella 53277 è 1, lo sprite si allarga del doppio; un valore di 1 per il bit nella 53271 genera uno sprite di doppia altezza.
Raddoppiare uno sprite non è esattamente lo stesso che metterne due affiancati. Ogni bit nella definizione dello sprite ingrandito è rappresentato da due punti e vi è, quindi, qualche perdita di definizione ottica; gli sprite ingranditi sono tuttavia più facili da controllare degli sprite multipli abbinati. Anche gli sprite multicolori possono essere ingranditi.

6.10. Uso avanzato della scheda VIC-II

Sono qui presentati alcuni aspetti della scheda VIC-II utili per scrivere programmi sofisticati. Se avete difficoltà a seguire questi argomenti, potrete riesaminarli una volta acquisita più dimestichezza con il C-64.

LA MEMORIA E LA SCHEDA VIC-II

La VIC-II ed il resto del computer interpretano la memoria in modo differente. Il microprocessore 6502 del C-64 può accedere a 65536 byte di memoria, mentre la VIC-II soltanto a 16384. Inoltre, la VIC-II usa byte a 12 bit per

la memoria del video (8 per il carattere e 4 per il colore), mentre il resto del computer usa i byte "normali" di 8 bit. Questo per rendere più veloce l'azione della VIC-II, permettendole di prelevare tutte le informazioni necessarie per stampare un carattere in una sola volta. Per permettere ai due chip di comunicare fra di loro tramite la stessa memoria, il C-64 è progettato in modo che la VIC-II abbia una "finestra" d'accesso alla memoria, attraverso la quale possa accedere solo a parte della memoria. Questa finestra permette alla VIC-II di leggere uno per volta i quattro segmenti della memoria del C-64, ciascuno di 16384 byte.
La sezione di memoria che la VIC-II usa, è controllata da due uscite della scheda CIA#2, lo stato delle quali è fissato dai bit 0 e 1 della locazione 56576. I segmenti di memoria scelti da questi bit sono:

Coppia di bit	*Locazioni utilizzate*
11	0-16383
10	16384-32767
01	32768-49151
00	49152-65535

Le altre uscite associate alla 56576 sono usate per altri scopi, quindi i valori degli altri bit devono essere protetti. Per scegliere una sezione di memoria, usate una delle seguenti istruzioni:

```
95 REM SELEZIONA 0-16383
100 POKE 56576,PEEK(56576) AND 252 OR 3
105 REM SELEZIONA 16384-32767
110 POKE 56576,PEEK(56576) AND 252 OR 2
115 REM SELEZIONA 32768-49151
120 POKE 56576,PEEK(56576) AND 252 OR 1
125 REM SELEZIONA 49152-65535
130 POKE 56576,PEEK(56576) AND 252
```

La ROM dei caratteri

Se è vero che la ROM dei caratteri è situata nella 53248 e la VIC-II può accedere solo a 16K di memoria alla volta, come fa a leggere le tabelle dei caratteri mentre usa, per esempio, le locazioni 0-16383? La Commodore ha progettato il C-64 in modo che la VIC-II acceda alla ROM dei caratteri alle locazioni 4096-8191 e 36864-40959. Di fatto la VIC-II non legge la ROM dei caratteri alla 53248 ma un'area di RAM che rimane normalmente invisibile al resto del computer.
Nelle due aree dove la VIC-II legge la ROM dei caratteri, non può accedere alla RAM. Quest'area RAM è accessibile solo al chip 6502. Quando si selezio-

na un'area di memoria per memorizzarvi la memoria del video, le tabelle dei caratteri personali, o gli sprite, si deve tenere a mente che la ROM dei caratteri è disponibile solo in certe sezioni.

Scrivere nelle ROM

La tabella dei caratteri non è l'unica ROM del C-64: il *software* incorporato (il BASIC e le routine di input/output per la cassetta, video ecc.) è pure memorizzato nelle ROM che occupano le locazioni 40960-49151 e 57344-65535. Cartucce di software aggiuntive si inseriscono nelle locazioni 32768-40959. Il microprocessore 6502 non è in grado di accedere a queste aree, ma la VIC-II sì.
Con alcune precauzioni è possibile usare queste aree per la grafica. Quando il 6502 legge queste locazioni, "vede" le ROM, ma i dati che possono venire scritti vanno alla RAM. Poiché non è necessario rileggere il valore così memorizzato, si può usare la RAM "nascosta" per il video, gli sprite o la grafica ad alta risoluzione. Un esempio può essere uno sfondo prefissato all'inizio del programma che non cambia per tutta la durata dello stesso. La mappa potrebbe essere memorizzata alle locazioni 57344-65343, il video e gli sprite a 49152-53247.
Con questa tecnica è necessario modificare la subroutine usata negli esempi per fissare i bit, perché essa si basa sulla possibilità di poter eseguire dei PEEK ai valori già presenti nella mappa. Il vostro programma avrebbe bisogno di calcolare i valori di tutti i bit in una particolare locazione e poi prelevarli tutti assieme con un POKE.

Come cambiare la locazione della memoria del video e dei caratteri

Le memorie del video e dei caratteri possono essere situate ovunque nella "finestra" della VIC-II. Le locazioni iniziali sono conservate dal chip e possono essere cambiate con un POKE. Esse sono abbinate in un unica locazione, la 53272. Ogni metà del byte contiene un numero da 0 a 15, che rappresenta in Kbyte (1K=1024) l'*offset* nella finestra (la "distanza" dall'inizio del segmento alla locazione desiderata). La formula per calcolare il valore da usare con POKE è

```
(locazione di memoria del carattere / 1024)+
    16*(locazione di memoria del video / 1024)
```

L'ultimo bit viene ignorato benché la locazione di memoria del carattere sia espressa in unità di 1024 byte. La memoria del carattere deve cominciare ad un offset di 0, 2048, 4096, 6144, 8192, 10240 o 14336 nella finestra. Quando

si esegue un PEEK alla locazione 53272, il bit 0 assumerà dei valori imprevedibili. Se il vostro programma ha bisogno di esaminare il contenuto di questa locazione, deve mascherare questo bit eseguendo un AND tra il suo valore e 254.

Memoria del colore

Il chip della memoria del colore ha due serie di collegamenti elettronici: una che permette alla scheda VIC-II di leggerlo come la parte superiore del byte a 12 bit di quest'ultima, e una che permette al resto del computer di leggerlo come la parte inferiore di un byte di 8 bit. Sono differenti anche i collegamenti per i numeri delle locazioni. Mentre il resto del computer percepisce la memoria del colore solo tra le 55296 e 56319, alla VIC-II appare come se fosse ovunque nella sua "finestra". Il byte 0 nella memoria del colore, che il vostro programma "vede" come locazione 55296, è letto dalla VIC-II alla 0, 1024, 2048, 3072, 4096 e così via fino a 15360 nella sua "finestra". Il byte 1 appare alle 1, 1025, 2049, ecc., per cui, dovunque si sposta la memoria del video, viene usato lo stesso chip della memoria del colore.

6.11. Come riservare la memoria

Negli esempi si sono usate istruzioni POKE per impedire al BASIC di usare la memoria riservata ai caratteri personali, alla grafica ad alta risoluzione, agli sprite ecc. Bisognerà approfondire le regole che sottostanno alle istruzioni se si scrivono programmi che utilizzano diverse aree di memoria per caratteri personali. Le locazioni 55 e 56 formano un puntatore che segna la fine della memoria BASIC: questo è un numero a 16 bit che rappresenta la locazione del byte dopo l'ultimo disponibile per il BASIC.
Le locazioni 51 e 52 sono un puntatore simile per l'area di memoria delle stringhe BASIC. Esse normalmente contengono uno 0 dopo un comando CLR o RUN e non hanno bisogno di essere cambiate.
Per calcolare il valore per il POKE alle locazioni 52 e 56, dividete la locazione di inizio della memoria del carattere per 256. È essenziale che i POKE, per limitare la memoria del BASIC, siano eseguiti prima che il vostro programma definisca le variabili, e che siano seguiti da un CLR, altrimenti il BASIC non riconoscerebbe i limiti che gli avete cercato di imporre.
Anche programmi acquistati possono sottrarre memoria al BASIC. È probabile che la maggior parte di questi, sia prodotti dalla Commodore sia da altri fornitori, usino i 4K di memoria da 49152 a 53247, ma ve ne saranno sicuramente alcuni che non lo fanno. Intendendo usare caratteri personali con uno di questi programmi, studiate la documentazione fornita per evitare possibili sovrapposizioni. In alcuni casi sarà necessario fare alcune prove.

Alcuni programmi non si attengono ai limiti di memoria del BASIC ma usano il valore nella 644, che non è controllato dal BASIC, per determinare l'ultimo byte di memoria del C-64. Questa locazione può essere caricata con un POKE dello stesso valore usato per le locazioni 52 e 56, ma è necessario fare attenzione, perché non è possibile prevedere il modo in cui altre cartucce potranno usarla. Spesso la causa del mancato funzionamento di un programma può essere una cartuccia inserita. Se è richiesto un POKE alle locazioni 644, deve essere eseguito contemporaneamente ai POKE alle 52 e 56, prima del CLR.

COME REGISTRARE E CARICARE DATI GRAFICI

L'uso d'istruzioni DATA per memorizzare il vostro set di caratteri personali, sprite e grafica ad alta risoluzione presenta diversi svantaggi. Le istruzioni DATA usano prezioso spazio di programmazione e il BASIC impiega molto tempo per trasferire informazioni in memoria con READ e POKE. Un Datassette può impiegare due o tre minuti per caricare con LOAD ed eseguire un programma che usa una grafica ad alta risoluzione o un set di caratteri personali.
Per accelerare i tempi, il vostro programma può usare le stesse subroutine SAVE e LOAD che sono impiegate dai comandi SAVE e LOAD del BASIC. In questo stesso capitolo si è visto un esempio di programma per costruire un set di caratteri personali e poi si è usato un comando LOAD per caricare il programma principale in memoria. Questo metodo risolve il problema delle dimensioni del programma, ma non del tempo. Introduce pure nuovi problemi; si deve dividere il programma in due e "dimenticare" le variabili usate nel programma di caricamento usando un CLR prima del LOAD. I programmi BASIC non possono usare direttamente le subroutine incorporate di SAVE e LOAD mentre lo possono fare i programmi in linguaggio macchina.
Per accelerare e compattare quei programmi che usano caratteri personali e sfondi ad alta risoluzione, sono stati scritti due programmi in linguaggio macchina che possono essere chiamati dai programmi in BASIC per registrare e prelevare grandi blocchi di dati da dischetti o cassette. Per usarli, costruite il vostro set di caratteri personali, o sprite o immagini grafiche, e usate il programma SAVE per memorizzarli sul dischetto. Il programma che impiega i dati dovrebbe includere il programma LOAD come subroutine.

Il programma SAVE

Prima di iniziare a lavorare sul set di caratteri personali, si dovrebbe caricare il programma SAVE e memorizzarlo su dischetto o nastro.

```
100 REM PROGRAMMA SAVE
110 PS=49152
120 DATA 160, 16,177,253,170,200,177,253
130 DATA 160,  0, 32,186,255,169, 16,166
140 DATA 253,164,254, 32,189,255,160, 18
150 DATA 177,253,133,251,200,177,253,133
160 DATA 252,160, 20,177,253,170,200,177
170 DATA 253,168,169,251, 32,216,255,176
180 DATA   2,169,  0,133,251, 96
190 FOR I=PS TO PS+53 : READ X : POKE I,X : NEXT
200 DCB=50000 : DA=20000 : SIZE=256 : DV=8
210 FI$="LOAD/SAVE TEST  "
220 REM MEMORIZZA IL NOME DEL FILE
230 FOR I=1 TO 16 : POKE DCB+I-1,ASC(MID$(FI$,I,1)):
    NEXT
240 POKE DCB+16,DV : REM NUMERO DEL DISPOSITIVO
250 POKE DCB+17,1 : REM NUMERO DEI FILE INATTIVI
260 POKE DCB+18,DA-256*INT(DA/256)
265 REM INDIRIZZO INFERIORE
270 POKE DCB+19,DA/256 : REM INDIRIZZO SUPERIORE
280 DE=DA+SIZE-1 : REM ULTIMA LOCAZIONE DA SALVARE
290 POKE DCB+20,DE-256*INT(DE/256)
295 REM INDIRIZZO INFERIORE
300 POKE DCB+21,DE/256 : REM INDIRIZZO SUPERIORE
310 REM FISSA IL PUNTATORE
320 POKE 253,DCB-256*INT(DCB/256)
330 POKE 254,DCB/256
340 REM PROVA IL SALVATAGGIO
350 SYS PS
360 IF PEEK(251) <>0 THEN 390
370 PRINT "FATTO!"
380 END
390 PRINT"ERRORE:";PEEK(251)
400 END
```

Il programma SAVE deve essere prima personalizzato: vi sono cinque variabili i cui valori dipendono dai dati che si vogliono memorizzare. Le prime quattro variabili sono assegnate a riga 2ØØ:

DCB è la locazione di un'area di 22 byte in cui sono memorizzati i valori per la subroutine in linguaggio macchina. Il programma dell'esempio utilizza le locazioni 50000-50021. Non è necessario cambiare questo valore, a meno di non usare quest'area di memoria per altri scopi.

DA è la locazione del primo byte dei dati da memorizzare. Questo valore probabilmente non corrisponde a quello richiesto.

SIZE è il numero dei byte da memorizzare.

DV è il numero del dispositivo dell'unità di gestione dei dischetti o delle cassette su cui verranno trascritti i dati. Se i dati sono da tra-

sferire su nastro, usate un valore 1. Nel caso di impiego dei dischetti, usate lo stesso numero adottato con i comandi LOAD e SAVE del BASIC (in genere 8).

L'ultima variabile da cambiare è FI$. Questa variabile è assegnata a riga 21Ø ed è il nome del file che verrà creato dal SAVE. Per realizzare un programma in linguaggio macchina semplice e breve, bisogna verificare la lunghezza del nome del file: deve essere esattamente di 16 caratteri. Se il nome desiderato è più corto, bisogna aggiungere spazi fino a raggiungere 16 caratteri. Se il nome è più lungo, qualsiasi carattere oltre il sedicesimo è ignorato. (Se non si vogliono contare i caratteri, aggiungete semplicemente spazi in più per assicurarvi che il nome sia abbastanza lungo).
La subroutine in linguaggio macchina è memorizzata nelle locazioni da 49152 fino a 49205. Nel caso in cui vi siano memorizzati dati grafici, si deve spostare la subroutine. Quest'ultima è progettata in modo da poter essere collocata ovunque, ma l'area in cui è memorizzata deve essere protetta dal BASIC. Dovendo spostare il programma, cambiate il valore della variabile PS a riga 11Ø con la locazione di memoria dove il programma deve cominciare. Quando i dati da memorizzare sono pronti, eseguite un LOAD sul programma SAVE. Inserite il nastro o il dischetto su cui volete memorizzare i dati ed eseguite il programma.
Qualora la routine SAVE incorporata indicasse un errore, la subroutine memorizzerebbe il codice d'errore nella locazione 251. Il capitolo 8 descrive questi codici (sono gli stessi di quelli riportati nella variabile ST). Vi è una condizione in cui il programma SAVE non funziona: cercando di sostituire un file preesistente sul dischetto, il programma presenterà il messaggio che comunica il buon funzionamento di SAVE, ma la luce rossa sull'unità a dischetti lampeggerà. Il file originale deve essere cancellato e il programma rieseguito (vedere le istruzioni al capitolo 8).

Il programma LOAD

La porzione BASIC del programma LOAD è molto simile a quella del programma SAVE.

```
100 REM PROGRAMMA LOAD
110 PS=49152
120 DATA 160, 16,177,253,170,200,177,253
130 DATA 160,  0, 32,186,255,169, 16,166
140 DATA 253,164,254, 32,189,255,160, 18
150 DATA 177,253,170,200,177,253,168,169
160 DATA   0, 32,213,255,144,  7,133,251
170 DATA 169,  0,133,252, 96,134,253,133
180 DATA 252, 96
190 FOR I=PS TO PS+49 : READ X   POKE I,X   NEXT
```

```
200 DCB=50000 : DA=20000 : DV=8
210 FI$="TEST LOAD/SAVE  "
220 REM MEMORIZZA IL NOME DEL FILE
230 FOR I=1 TO 16 : POKE DCB+I-1,ASC(MID$(FI$,I,1))
    NEXT
240 POKE DCB+16,DV : REM NUMERO DEL DISPOSITIVO
250 POKE DCB+17,8 : REM NUMERO DEI FILE INATTIVI
260 POKE DCB+18,DA-256*INT(DA/256)
265 REM BYTE INFERIORE
270 POKE DCB+19,DA/256 : REM BYTE SUPERIORE
280 REM FISSA IL PUNTATORE
290 POKE 253,DCB-256*INT(DCB/256)
300 POKE 254,DCB/256
310 REM ESEGUE IL CARICAMENTO
320 SYS PS
330 IF PEEK(252) = 0 THEN 360
340 PRINT "FATTO!"
350 END
360 PRINT"ERRORE:";PEEK(251)
370 END
```

Il programma LOAD è concepito per essere usato come parte del programma che impiega i dati da caricare. Può essere usato come subroutine, sostituendo le istruzioni END con RETURN e chiamandolo con un GOSUB.
Come il programma SAVE, quello LOAD deve essere adattato al vostro uso. Le variabili DA, DCB, PS e FI$ devono essere fissate in accordo con le direttive sopra esposte. La variabile SIZE non viene usata, perché la dimensione è memorizzata con i dati nel momento in cui sono registrati. Dato che il programma LOAD verrà abbinato al vostro, ci si deve assicurare che le istruzioni DATA che contengono il linguaggio macchina siano collocate correttamente. Ricordarsi che la prima istruzione READ preleva le sue informazioni dalla prima istruzione DATA. Se il vostro programma contiene dei READ eseguiti prima di quelli del programma LOAD, le istruzioni DATA devono precedere quelle contenenti la subroutine in linguaggio macchina. Se si usa più di una caratteristica grafica, come una combinazione di sprite e di caratteri personali, è conveniente assegnare i valori di DA e FI$ nel programma principale e chiamare la subroutine LOAD per caricare ogni blocco di dati.
Chiamando la subroutine più di una volta, si deve saltare l'istruzione READ dopo la prima volta, per mezzo di un GOSUB a riga 2ØØ invece di 1ØØ.

Le subroutine in linguaggio macchina

Nelle figure 6.6 e 6.7 sono presentati i listati delle subroutine in linguaggio macchina usate dai programmi SAVE e LOAD. Non avrete bisogno di studiarle a meno che non siate dei programmatori in linguaggio macchina o non desideriate cambiarle. I programmi si possono usare così come sono.

```
              SORCIM 650x Assembler ver 3.2  05/20/83 13:10 Page1
absolute saver subroutine for C-64 BASIC            A:C64SAVE .ASM

              ; Questa subroutine e' progettata per permettere al
              ; programmatore BASIC di conservare immagini video,
              ; definizioni di sprite,ecc su disco o nastro.
              ;
              ; Quando e' chiamata,richiede in locazione $FD un
              ; puntatore ad una struttura della forma seguente:
              ;
              ; Offset    Contenuto
              ;   0  Nomefile: 16 caratteri,riempito con spazi
              ;  16  Indirizzo dell'unita'
              ;  17  Numero di file logico (non deve venir usato)
              ;  18  Indirizzo di partenza (in formato lo-hi)
              ;  20  Indirizzo finale (in formato lo-hi)
              ;
              ; Al ritorno della subroutine,la locazione $FB
              ; contiene il codice di ritorno dalla SAVE
              ;
   = FFD8     SAVE      equ     $FFD8
   = FFBA     SETLFS    equ     $FFBA
   = FFBD     SETNAM    equ     $FFBD
   = 00FD     DCB       equ     $FD     ; puntatore al blocco ctl
   = 00FB     START     equ     $FB     ; inizio del puntatore
                                          save e codice di ritorno

              ; attiva l'indirizzo dell'unita'

0000 A010          ldy   #16
0002 B1FD          lda   (dcb),y  ; indirizzo dell'unita'
0004 AA            tax            ; pone in x
0005 C8            iny
0006 B1FD          lda   (dcb),y  ; numero del file logico
0008 A000          ldy   #0       ; indirizzo secondario di 0
000A 20BAFF        jsr   setlfs

              ; attiva il nome del file

000D A910          lda   #16      ; lunghezza del nome
000F A6FD          ldx   dcb      ; nome
0011 A4FE          ldy   dcb+1    ; locazione
0013 20BDFF        jsr   SETNAM
```

(continua)

```
                ; pone l'indirizzo di partenza in START

0016 A012           ldy   #18       ; punta al byte inferiore
0018 B1FD           lda   (dcb),y
001A 85FB           sta   start     ; lo memorizza
001C C8             iny             ; punta al byte superiore
001D B1FD           lda   (dcb),y
001F 85FC           sta   start+1   ; lo memorizza

                ; attiva l'indirizzo di fine memoria

0021 A014           ldy   #20       ; punta al byte inferiore;
0023 B1FD           lda   (dcb),y
0025 AA             tax             ; lo pone in x
0026 C8             iny             ; punta al byte superiore
0027 B1FD           lda   (dcb),y
0029 A8             tay             ; lo pone in y
002A A9FD           lda   #dcb      ; punta alla locazione iniziale
002C 20D8FF         jsr   SAVE

                SORCIM 650x Assembler ver 3.2  05/20/83 13:10 Page2
absolute saver subroutine for C-64 BASIC              A:C64SAVE .ASM

                ; verifica eventuali errori

002F B002  0033     bc$   error     ; si'
0031 A900           lda   #0        ; no,poni il codice a 0

                ; SAVE in esecuzione. Conserva il codice di errore
                  per una eventuale richiesta dell'operatore

0033 85FB          error: sta       start
0035 60            rts              ; ritorna
0036               end

no ERRORs,  6 Labels, 9D7Bh bytes not used. Program LWA = 0036h.
```

Figura 6.6. Routine SAVE

```
              SORCIM 650x Assembler ver 3.2  05/20/83 13:10 Page2
absolute loader subroutine for C-64 BASIC          A:C64LOAD .ASM

              ; Questa subroutine e' progettata per permettere al
              ; programmatore BASIC di caricare immagini video,
              ; definizioni di sprite,ecc da disco o nastro.
              ;
              ; Quando e' chiamata,richiede in locazione $FD un
              ; puntatore ad una struttura della forma seguente:
              ;
              ; Offset     Contenuto
              ;  0   Nomefile: 16 caratteri,riempito con spazi
              ; 16   Indirizzo dell'unita'
              ; 17   Numero di file logico (non deve venir usato)
              ; 18   Indirizzo di partenza (in formato lo-hi)
              ;
              ; Al ritorno della subroutine,la locazione $FB
              ; contiene l'ultimo byte caricato, o il codice
              ; di ritorno dalla LOAD
              ;
    = FFD5    LOAD          equ     $FFD5
    = FFBA    SETLFS        equ     $FFBA
    = FFBD    SETNAM        equ     $FFBD
    = 00FD    DCB           equ     $FD     ; puntatore al blocco ctl
    = 00FB    ENDLD         equ     $FB     ; codice di ritorno e
                                                indirizzo finale

              ; attiva l'indirizzo dell'unita'

0000 A010          ldy    #16
0002 B1FD          lda    dcb),y   ; indirizzo dell'unita'
0004 AA            tax             ; lo pone in x
0005 C8            iny
0006 b1FD          lda    (dcb),y  ; numero del file logico
0008 A000          ldy    #0       ; indirizzo secondario di 0
000A 20BAFF        jsr    setlfs

              ; attiva il nome del file

000D A910          lda    #16      ; lunghezza del nome
000F A6FD          ldx    dcb      ; nome
0011 A4FE          ldy    dcb+1    ; locazione
0013 20BAFF        jsr    SETNAM
```

(continua)

```
                ; attiva l'indirizzo di memoria

0016 A012         ldy   #18       ; punta al byte inferiore
0018 B1FD         lda   (dcb),y
001A AA           tax               ; lo pone in x
001B C8           iny               ; punta al byte superiore
001C B1FD         lda   (dcb),y
001E A8           tay               ; lo pone in y
001F A900         lda   #0          ; indica LOAD
0021 20D5FF       jsr   LOAD

                ; verifica se tutto e' andato bene

0024 9007  002D   bcc   ok          ; si

                ; errore in fase di LOAD. Conserva il codice per
                   una eventuale richiesta dell'operatore

0026 85FB          sta   endld
0028 A900          lda   #0         ; pone il byte superiore a 0 per
                                   mostrare che e'un codice di errore
002A 85FC         sta   endld+1    ;
002C 60           rts              ; ritorna

               SORCIM 650x Assembler ver 3.5  05/20/83 13:10 Page2
absolute loader subroutine for C-64 BASIC          A:C64LOAD .ASM

                ; tutto a posto,quindi restituisci
                   l'indirizzo di memoria

002D 86FB      ok: stx   endld     ; byte inferiore
002F 84FC          sty   endld+1   ; byte superiore
0031 60            rts             ; ritorna all'operatore
0032              end

no ERRORs,  6 Labels, 9D7Bh bytes not used. Program LWA = 0032h.
```

Figura 6.7. Routine LOAD

LIMITAZIONI DELLO SCHERMO TV

Il televisore è progettato per rappresentare immagini in movimento, e non per presentazioni di computer. Benché il C-64 fornisca gli stessi segnali usati nelle trasmissioni televisive, il *contenuto* del video è differente e ciò può causare dei problemi quando si disegnano le videate. L'immagine televisiva non è affatto così ben definita come appare, si presenta molto nitida se vista a distanza, sfocata da vicino; essa inoltre si rinnova 30 volte al secondo, quindi l'occhio non ne percepisce i difetti.
Usando la televisione con il computer, si sarà molto più vicini allo schermo del normale. Ciò evidenzierà la mancanza di qualità dell'immagine; le videate normalmente non cambiano così velocemente come un film, quindi c'è più tempo per notare i difetti. Vi è inoltre un altro fattore importante: i punti dell'immagine televisiva normalmente non sono isolati. È molto raro che l'immagine televisiva contenga una riga larga un solo punto: un punto sul video fa quasi sempre parte di un oggetto più grande. Si noterà altresì che non vi sono molti contrasti di colore.
Le videate sono molto differenti: è infatti abbastanza comune tracciare linee molto sottili con colori contrastanti affiancati. Sfortunatamente la televisione, anche in questo caso, ha una qualità scarsa. Terminali grafici a colori, capaci di presentare accuratamente due punti affiancati di qualsiasi colore senza distorsioni, costano facilmente alcuni milioni e non sono proponibili agli utenti del C-64. Nel disegnare le videate dovete tenere a mente le limitazioni dello schermo della vostra televisione.

Larghezza delle linee verticali

Il problema delle sottili linee verticali larghe un solo punto è in verità connesso al problema dei cambiamenti di colore. Se i colori sono incompatibili, è possibile che si renda necessario più di un punto per effettuare il cambio. Per esempio, ci vogliono quasi 3 punti per effettuare il cambio da blu scuro a rosso scuro. Un singolo punto rosso su uno sfondo blu è quasi invisibile e non sembra neppure rosso. Come regola generale si può affermare che è possibile tracciare linee sottili scure su un fondo chiaro ma non viceversa. Sfortunatamente questa non è una regola ferrea. Gli apparecchi televisivi variano moltissimo e ciò che funziona su uno può non funzionare su un altro. Anche agire sui controlli del televisore può influire sull'esito della rappresentazione. Si dovrebbe evitare l'uso di punti isolati e linee larghe un solo punto, a meno di non essere certi che il vostro programma verrà eseguito sempre con lo stesso tipo di televisore.

Capitolo 7

Il suono

Questo capitolo è dedicato a come far generare suoni al C-64 attraverso il controllo dei registri dedicati e all'uso di tecniche per produrre effetti particolari come il vibrato o il tremulo. Inoltre vedremo come memorizzare su dischetto o cassetta i suoni prodotti per poterli risentire in seguito.

7.1. I registri del suono

IL CHIP SID

Come le presentazioni su video, anche i suoni prodotti dal C-64 sono creati da uno speciale circuito integrato, chiamato in questo caso "*Sound Interface Device*" (Dispositivo d'interfaccia del suono) o "SID". Il dispositivo SID contiene tre generatori di suono diversi, o "voci", che sono combinati in maniera da formare suoni udibili tramite l'altoparlante della TV. Il SID ha 25 locazioni di memoria, chiamate registri del suono, che controllano il volume, il tono ed il tipo di suono emesso. La tabella 7.1 riporta i registri di memoria e la funzione di ognuno di essi.

Tabella 7.1. Locazioni di memoria dei registri del suono

Locazione di memoria	*Descrizione del registro del suono*
	54272 - 54278 VOCE #1
54272	Controllo frequenza voce (byte basso)
54273	Controllo frequenza voce (byte alto)
54274	Durata dell'impulso (byte basso)
54275	Durata dell'impulso (byte alto)
54276	Registro di controllo della forma d'onda
54277	Registro di controllo Attack e Decay
54278	Registro di controllo Sustain e Release
	54279 - 54285 VOCE #2
54279	Controllo frequenza voce (byte basso)
54280	Controllo frequenza voce (byte alto)
54281	Durata dell'impulso (byte basso)
54282	Durata dell'impulso (byte alto)
54283	Registro di controllo della forma d'onda
54284	Registro di controllo Attack e Decay
54285	Registro di controllo Sustain e Release
	54286 - 54292 VOCE #3
54286	Controllo frequenza voce (byte basso)
54287	Controllo frequenza voce (byte alto)
54288	Durata dell'impulso (byte basso)
54289	Durata dell'impulso (byte alto)
54290	Registro di controllo della forma d'onda
54291	Registro di controllo Attack e Decay
54292	Registro di controllo Sustain e Release
	54293 - 54296 FUNZIONI DI FILTRO
54293	Valore filtro cutoff (byte basso)
54294	Valore filtro cutoff (byte alto)
54295	Controllo risonanza filtro/ingresso voce
54296	Controllo del volume

La locazione 54296 controlla il volume del suono. Si possono ottenere 16 livelli di volume differenti. Questi variano da 0 (spento) a 15 (massimo volume). Per controllare il volume caricate un valore tra 0 e 15. Da solo, però, questo registro non produce alcun suono; per ottenerne si devono assegnare anche i registri delle voci.

REGISTRI DI CONTROLLO DELLE VOCI

Osservando la tabella 7.1 si noterà che ognuna delle tre "voci" ha una serie di 7 registri di memoria associati, ciascuno dei quali controlla il tipo di suono prodotto da quella voce. Le tre serie di registri funzionano nella stessa maniera, per cui, anche se gli esempi impiegheranno prevalentemente la voce #1, si possono usare gli stessi valori eseguendo i POKE ai registri delle voci 2 e 3.

Inizializzazione dei registri

Quando il C-64 viene acceso i registri assumono valori casuali. Se non viene eseguito un POKE con i valori corretti, il SID produce suoni imprevedibili, o addirittura nessun suono. Per inizializzare i registri correttamente, trasferite con POKE i seguenti valori:

```
POKE 54274,0
POKE 54275,8
POKE 54276,65
POKE 54277,0
POKE 54278,240
```

In questo capitolo esamineremo gli usi dei vari registri e la determinazione dei valori di POKE. I primi due che verranno presi in esame controllano la frequenza del suono.

Un suono con un POKE

La tabella 7.2 mostra le frequenze che il SID è in grado di produrre, le note musicali approssimativamente realizzate e i valori di POKE per generarle. La nota più bassa prodotta dal SID è un "Do" basso (circa 16 Hz, o 16 cicli al secondo). Caricate i valori per questa nota:

```
POKE 54272,12:POKE 54273,1
```

ed attivate il massimo volume.

```
POKE 54296, 15
```

Per spegnere il suono, usate uno dei seguenti metodi:

— POKE 0 nei registri del suono

— POKE 0 nel registro del volume
— Premere simultaneamente RUN/STOP e RESTORE.

Il primo metodo mette un tono inesistente nel registro del suono, il secondo fissa il volume al livello minimo, che è zero, il terzo metodo azzera tutte le variabili di sistema, e quindi oltre ai registri del suono, anche quelli del colore e così via. È meglio usare uno dei primi due metodi, il terzo potrebbe fare più danni di quanto si possa pensare.

Tabella 7.2. Tabella delle equivalenze del valore di byte alto/basso - note musicali

(Byte alto) Valore POKE	*(Byte basso) Valore POKE*	*Frequenza (Hz)*	*Note musicali*
	PRIMA OTTAVA		
1	12	16	DO
1	28	17	DO#
1	45	18	RE
1	62	19	RE#
1	81	21	MI
1	101	22	FA
1	123	23	FA#
1	145	24	SOL
1	169	25	SOL#
1	195	27	LA
1	221	29	LA#
1	250	31	SI
	SECONDA OTTAVA		
2	24	32	DO
2	56	34	DO#
2	90	37	RE
2	125	39	RE#
2	163	41	MI
2	203	44	FA
2	246	46	FA#
3	35	49	SOL
3	83	52	SOL#
3	134	55	LA
3	187	58	LA#
3	244	62	SI
	TERZA OTTAVA		
4	48	65	DO
4	112	69	DO#

Tabella 7.2. (continua)

(Byte alto) Valore POKE	*(Byte basso) Valore POKE*	*Frequenza (Hz)*	*Note musicali*
4	180	73	RE
4	251	76	RE #
5	71	82	MI
5	151	87	FA
5	237	92	FA #
6	71	98	SOL
6	167	104	SOL #
7	12	110	LA
7	119	117	LA #
7	233	123	SI
	QUARTA OTTAVA		
8	97	131	DO
8	225	139	DO #
9	104	147	RE
9	247	156	RE #
10	143	165	MI
11	47	175	FA
11	218	185	FA #
12	142	196	SOL
13	77	208	SOL #
14	24	220	LA
14	238	233	LA #
15	210	247	SI
	QUINTA OTTAVA		
16	195	262	DO
17	194	277	DO #
18	208	294	RE
19	238	311	RE #
21	30	330	MI
22	95	349	FA
23	180	370	FA #
25	29	392	SOL
26	155	415	SOL #
28	48	440	LA
29	221	466	LA #
31	164	494	SI

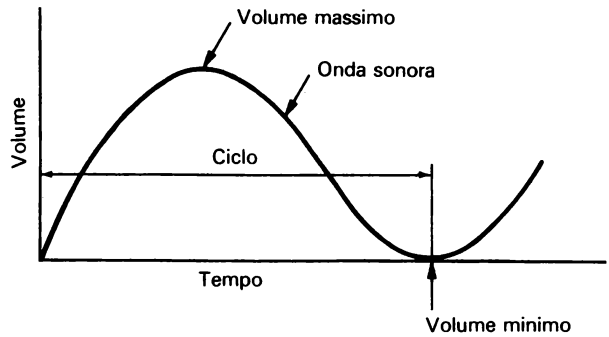
Tabella 7.2. (continua)

(Byte alto) Valore POKE	*(Byte basso) Valore POKE*	*Frequenza (Hz)*	*Note musicali*
	SESTA OTTAVA		
33	134	523	DO
35	132	554	DO#
37	161	587	RE
39	221	622	RE#
42	60	659	MI
44	191	698	FA
47	104	740	FA#
50	58	784	SOL
53	55	831	SOL#
56	97	880	LA
59	187	932	LA#
63	72	988	SI
	SETTIMA OTTAVA		
67	12	1046	DO
71	8	1109	DO#
75	66	1175	RE
79	187	1244	RE#
84	121	1319	MI
89	127	1397	FA
94	209	1480	FA#
100	117	1568	SOL
106	110	1661	SOL#
112	194	1760	LA
119	118	1865	LA#
126	145	1976	SI
	OTTAVA OTTAVA		
134	24	2093	DO
142	17	2217	DO#
150	132	2349	RE
159	119	2489	RE#
168	242	2637	MI
178	254	2794	FA
189	163	2960	FA#
200	234	3136	SOL
212	220	3322	SOL#
225	132	3520	LA
238	237	3729	LA#
253	34	3951	SI

7.2. Le componenti del suono

TONI PURI

Con i valori dei registri di controllo usati sin qui, il SID produce frequenze continue tra 1 Hz e 3995,669 Hz circa. La frequenza rappresenta il numero di cicli al secondo ed un ciclo corrisponde a una completa oscillazione da un valore minimo al massimo e di nuovo al minimo.



Il tono di un suono è una funzione diretta della sua frequenza; raddoppiando la frequenza, si otterrà un tono esattamente di un'ottava superiore all'originale. Provate questo programma:

```
10 REM AZZERA I REGISTRI DEL SUONO
20 FOR R=54272 TO 54296: POKE R,0: NEXT
30 REM FISSA IL REGISTRO #1
40 POKE 54274, 0: POKE 54275, 8
50 POKE 54278, 240: POKE 54296, 15: T=1
60 POKE 54277, 0: POKE 54276, 65
70 REM LEGGE I TASTI FUNZIONE
90 GET A$: IF A$="" THEN 90
100 IF A$=CHR$(133) THEN 160
110 IF A$=CHR$(134) THEN 170
120 IF A$=CHR$(135) THEN 180
130 IF A$=CHR$(136) THEN 200
140 GOTO 90
150 REM FA UN POKE DEI TONI NEI REGISTRI DEL SUONO #1
160 POKE 54272,48:POKE 54273, 4:GOTO 90
170 POKE 54272,97:POKE 54273, 8:GOTO 90
180 POKE 54272,195:POKE 54273,16:GOTO 90
190 REM CONTROLLO VOLUME
200 IF T=1 THEN POKE 54296,0:T=-1:GOTO 90
210 POKE 54296, 14: T=1: GOTO 90
```

La riga 2Ø azzera tutti i registri, in modo che non vengano prodotti suoni a caso prima di caricare quelli desiderati. Si sarà notato che ogni volta che si vuole caricare un valore di tono in un registro, si devono usare due valori. Questo è dovuto al fatto che il C-64 ha una gamma di 65535 toni differenti, ma ogni singola locazione di memoria può contenere solo 256 valori (255, non contando lo 0 che non produce alcun suono). Quindi per produrre l'intera gamma di 65535 suoni differenti, il C-64 impiega due locazioni di memoria per ogni tono. I valori in queste due locazioni si abbinano per formare un numero tra 0 e 65535.
Il numero così prodotto è rappresentato da un numero binario a 16 bit, ma non è necessario conoscere i numeri binari per usare i registri; consultate semplicemente la tabella 7.2 per trovare la nota desiderata e caricate con POKE i valori indicati. Le righe 4Ø-6Ø usano i registri per produrre toni continui esattamente come visto sopra. Le righe 9Ø-13Ø leggono i tasti di funzione e saltano alle routine che suonano la nota "Do" in una di tre ottave differenti (nelle righe 16Ø-18Ø). Premendo F7 si salta a riga 2ØØ che "accende" o "spegne" il suono. Il programma poi ritorna a riga 9Ø ed attende che venga premuto un altro tasto.

Le scale

Si può cambiare il valore di un tono mentre è attivo, passando così immediatamente da una nota ad un'altra.
Provate il seguente esempio.

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54272, 0: POKE 54273, 0
30 POKE 54274, 0: POKE 54275, 8
40 POKE 54278, 240: POKE 54296, 15
50 POKE 54277, 0: POKE 54276, 65
60 FOR F=0 TO 65535 STEP 256
70 H=INT(F/256): L=INT(F-(256*H))
80 POKE 54272, L: POKE 54273, H
90 NEXT
100 FOR R=54272 TO 54296:POKE R,0: NEXT
```

Questo programma esplora l'intera gamma di toni disponibili con incrementi di 256. L'incremento si può cambiare modificando il valore di STEP a riga 6Ø. La riga 7Ø contiene la routine per trasformare i numeri decimali da 0 a 65535 nei corretti valori di POKE per i registri. Si può usare questa routine per creare qualsiasi tono si desideri.

Toni multipli

È possibile aggiungere profondità agli effetti sonori, facendo suonare due o più toni alla volta. L'aggiunta delle seguenti righe al programma precedente, genera due toni contemporaneamente.

```
25 POKE 54281, 0: POKE 54282, 8
35 POKE 54285, 240
45 POKE 54284, 0:POKE 54283, 65
75 POKE 54279, L: POKE 54280, H
```

Si può anche armonizzare. Eseguite un NEW e caricate il seguente programma:

```
10 REM AZZERA I REGISTRI DEL SUONO
20 FOR R=54272 TO 54296: POKE R,0: NEXT
30 REM ASSEGNA I REGISTRI DEL TONO
40 POKE 54274, 0: POKE 54275, 8
50 POKE 54281, 0: POKE 54282, 8
60 POKE 54288, 0: POKE 54289, 8
70 POKE 54278, 240: POKE 54296, 15
80 POKE 54285, 240 : POKE 54292, 240
90 POKE 54276, 65: POKE 54283, 65: POKE 54290, 65
100 REM ATTIVA IL SUONO
110 POKE 54273, 16: POKE 54272, 195
120 FOR G=0 TO 500: NEXT
130 POKE 54280, 21: POKE 54279, 30
140 FOR G=0 TO 500: NEXT
150 POKE 54278, 25: POKE 54286, 29
160 FOR G=0 TO 500: NEXT
170 FOR G=0 TO 1500: NEXT
180 REM SPEGNE
190 FOR R=54272 TO 54296: POKE R,0: NEXT
```

Con una progettazione attenta, si può far credere agli ascoltatori di sentire più di tre registri. Provate ad aggiungere le seguenti istruzioni:

```
162 POKE 54273, 33: POKE 54272, 134
165 FOR G=0 TO 500 : NEXT
```

Toni scanditi

Un altro metodo per modificare i registri del suono è quello di accenderli e spegnerli rapidamente. Questo metodo può creare l'effetto di un ronzio. Per capire meglio il funzionamento di questo effetto, guardate la figura 7.1 con riferimento al seguente programma:

```
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54274, 0: POKE 54275, 8
30 POKE 54278, 240: POKE 54296, 15
40 POKE 54277, 0: POKE 54276, 65
50 FOR T=0 TO 50
60 POKE 54272, 0: POKE 54273, 0
70 FOR F=0 TO 40: NEXT
80 POKE 54272, 67: POKE 54273, 12
90 NEXT
100 POKE 54296, 0
```

Le righe da 1Ø a 4Ø assegnano i registri, ma non vi è alcun output di suono fino a che il registro non è acceso. La riga 5Ø inizia il loop che determina il numero di pulsazioni prodotte. La riga 6Ø azzera i registri della frequenza, come mostra la figura 7.1. La riga 7Ø è il loop di ritardo che determina il tempo di disattivazione dell'impulso. La riga 8Ø genera il suono. La riga 9Ø completa il loop, rimandando il programma alla riga 5Ø. Nella riga 6Ø, il suono è di nuovo spento. Dopo essersi ripetuto per 50 volte, il loop termina alla riga 1ØØ azzerando il volume. Incrementando il

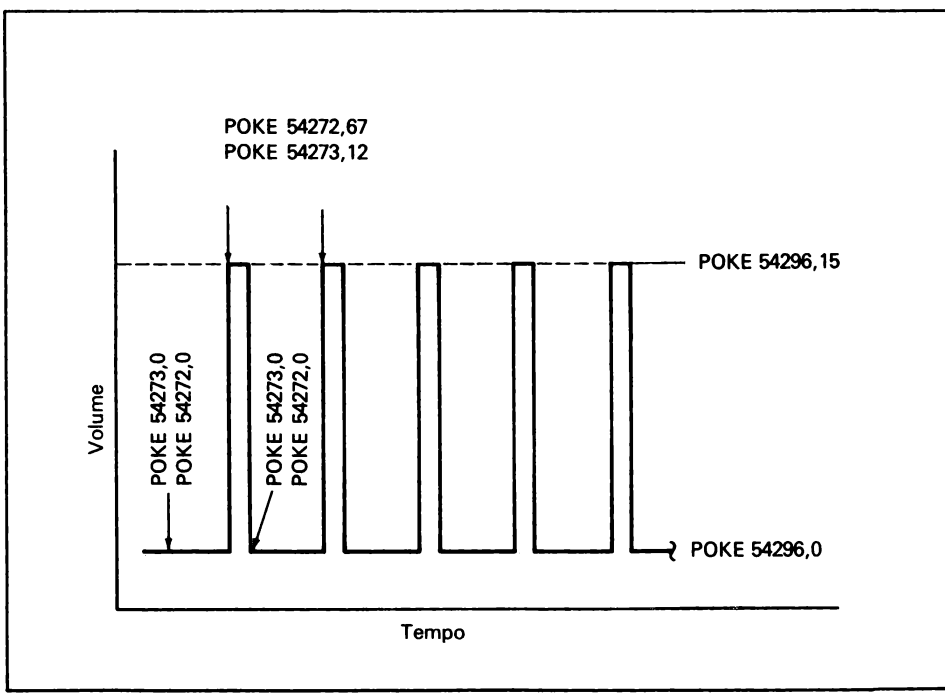


Figura 7.1. Forma d'onda ad impulso

tempo morto tra una pulsazione e l'altra, si può modificare questo programma fino a produrre l'effetto di una pallina da ping-pong che rimbalza. Cambiate la riga 7Ø così:

```
70 FOR F=0 TO 400: NEXT
```

REGOLAZIONE DEL VOLUME

Il volume è stato usato fin qui come interruttore; ovviamente lo si può impiegare anche per modificare la natura dei suoni prodotti. È possibile creare un buon numero di effetti semplicemente variando il volume.

Toni decrescenti

Riducendo lentamente il volume, può sembrare che la pallina stia rimbalzando sempre più lontano. Per ottenerlo cambiate la riga 5Ø nel seguente modo:

```
50 FOR T=0 TO 15 STEP.3
```

Il programma continua a produrre 50 ripetizioni del loop, ma in questo modo, T non supera mai 15 (il massimo volume permesso). Poi eseguite un POKE al registro del volume con il valore del loop così:

```
55 POKE 54296, 15-T
```

Ora il volume diminuisce ad ogni ciclo. Nello stesso modo si può anche far diminuire gli intervalli tra i rimbalzi. Provate a modificare la riga 7Ø

```
70 FOR F=0 TO 400-T*26: NEXT
```

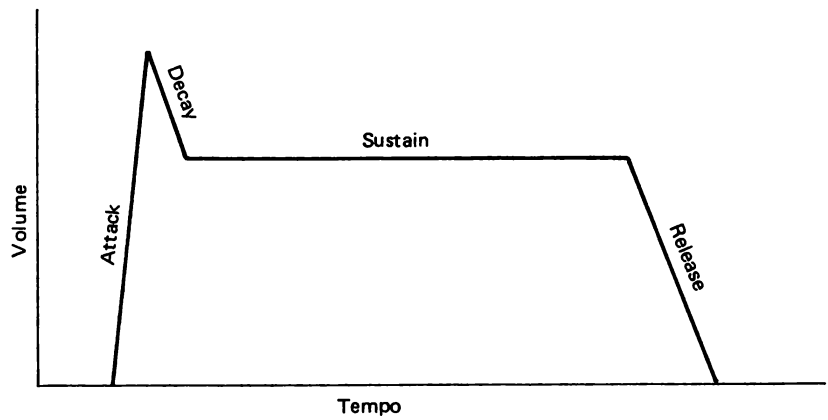
È stato scelto il numero 26 perché 400 diviso 15 fa circa 26. Il ritardo è così diviso in 50 decrementi uniformi. Sottraendo il valore del loop da 400 ad ogni passaggio si riduce l'intervallo tra una pulsazione e l'altra progressivamente. Si provi a modificare i toni del programma della pallina da ping-pong per simulare il ticchettio di un orologio usando due toni differenti nei cicli alternati del loop.

Attack/Decay/Sustain/Release

Quando si suona una nota su uno strumento musicale il volume del suono emesso segue un ciclo che possiamo così sintetizzare:

— Il volume sale rapidamente al valore massimo (Attack)
— Si attenua fino al valore medio (Decay)
— Si mantiene su questo valore fino a che la sorgente sonora è lasciata libera di vibrare (Sustain)
— Si azzera rapidamente (Release).

Queste quattro fasi sono controllate da una coppia di registri per ciascuna voce.



Tutti i suoni prodotti sino ad ora hanno avuto un Attack molto veloce e un altrettanto veloce Decay; la modifica di questi parametri fa cambiare notevolmente il suono. Ecco di nuovo il programma della palla che rimbalza.

```
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54274,0: POKE 54275,8
30 POKE 54278,240: POKE 54296,15
40 POKE 54277,0: POKE 54276,65
50 FOR T=0 TO 15 STEP .3
60 POKE 54272,0: POKE 54273,0
70 FOR F=0 TO 400: NEXT
80 POKE 54272,67: POKE 54273,12
90 NEXT
100 POKE 54296,0
```

Per produrre un Decay del suono si aggiunge un loop che causa la riduzione del volume ad ogni ciclo

```
85 FOR V=0 TO 15: POKE 54296,15-V:NEXT
```

Ecco come appare il programma con l'aggiunta di alcune righe che producono un Attack:

```
5 INPUT "ATTACK"; A
7 INPUT "DECAY"; D
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54274,0: POKE 54275,8
30 POKE 54278,240: POKE 54296,15
40 POKE 54277,0: POKE 54276,65
50 FOR T=0 TO 15 STEP .3
60 POKE 54272,0: POKE 54273,0
70 FOR F=0 TO 400: NEXT
80 POKE 54272,67: POKE 54273,12
85 FOR V=0 TO 15 STEP A: POKE 54296,V:NEXT
87 FOR V=0 TO 15 STEP D: POKE 54296,15-V:NEXT
90 NEXT
100 POKE 54296,0
```

Si ascolti la differenza introdotta da diversi parametri di Attack e Decay. Nell'istruzione input si può caricare qualsiasi numero positivo, comprese le frazioni. Attribuendo un valore 0 al Decay, la nota non terminerà mai. Il Sustain può essere introdotto ponendo un ritardo tra l'Attack e il Decay, nel modo seguente:

```
9 INPUT "SUSTAIN"; S
86 FOR SS=1 TO S: NEXT
```

Il Decay con le funzioni incorporate

L'unico svantaggio del produrre Attack e Decay in questa maniera è che qualunque variazione influenzerà tutti i registri dei toni. Il C-64 permette un'altra soluzione per ottenerli: il segreto sta nei registri della tabella 7.1. La figura 7.2 raffigura i registri di controllo dell'Attack/Decay per il registro #1, situati in 54277. Inserendo con POKE valori tra 0 e 15 nei 4 bit inferiori di questa locazione di memoria (il registro di Decay), si possono creare ritardi di varia durata.

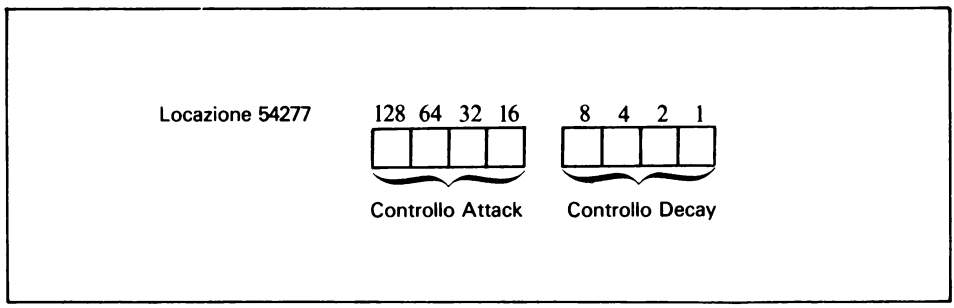


Figura 7.2. Locazione di memoria 54277: controllo Attack/Decay

```
5 INPUT"DECAY"; D
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54274,0: POKE 54275,8
30 POKE 54278,0: POKE 54296,15
40 POKE 54277,D: POKE 54276,65
50 FOR T=1 TO 1
60 POKE 54272,67: POKE 54273,12: POKE 54276,65
70 FOR F=0 TO 1000 : NEXT:POKE 54273,0:
75 POKE 54272,0: POKE 54276,64
80 NEXT
90 POKE 54296,0
```

Notate cosa accade usando Decay di diversa durata. I valori inferiori (ritardi minori) azzerano il volume prima della fine della nota. D'altro canto valori di Decay più lunghi lo riducono così lentamente che questo è ancora abbastanza alto quando la nota finisce. Bisogna tener conto del ritardo che si introduce usando questa funzione e far durare la nota quanto basta al Decay.

L'Attack con le funzioni incorporate

È analogo produrre Attack con le funzioni incorporate con l'unica differenza che bisogna caricare i valori di Attack nella porzione superiore (byte alto) del registro 54277. Il modo più semplice è quello di moltiplicare il valore di Attack per 16 ed eseguire un POKE al registro. Provate a cambiare le righe 5 e 40 come segue:

```
5 INPUT "ATTACK"; A
40 POKE 54277, A*16
```

Sustain

Eseguendo questo programma si scoprirà un'altra caratteristica interessante delle funzioni sonore incorporate. Le funzioni d'Attack più corte fanno iniziare il suono velocemente e lo fanno diminuire altrettanto bruscamente, mantenendo il volume basso per tutta la durata della nota. Questo perché il registro del suono abbassa il volume alla fine della funzione stessa. Per farla continuare dopo l'Attack, bisogna aggiungere anche un valore di Sustain, che mantiene il tono al suo valore massimo per la durata stabilita dalla funzione. Per ottenere ciò si carica con POKE un valore elevato nella parte superiore (byte alto) del registro Sustain/Release (54278 usando la voce #1) alla riga 3Ø.

```
30 POKE 54278, 240: POKE 54296, 15
```

È possibile che il suono venga spento prima d'aver raggiunto il massimo volume, se il tempo di rallentamento del loop di ritardo è troppo corto. Per compensare, si dovrà aumentare questo valore quando si impiegano valori d'Attack molto lunghi. Il Sustain può essere ottenuto anche mediante loop di ritardo. Ecco una routine che produce Attack, Decay e Sustain usando dei loop di ritardo.

```
10 INPUT"ATTACK"; A
20 INPUT"DECAY"; D
30 INPUT"SUSTAIN"; S
40 FOR R=54272 TO 54296: POKE R,0:NEXT
50 POKE 54274,0: POKE 54275,8
60 POKE 54278,240: POKE 54296,15
70 POKE 54277,0: POKE 54276,65
80 POKE 54272,0: POKE 54273,0
100 POKE 54272,67: POKE 54273,12
110 FOR V=0 TO 15 STEP A: POKE 54296,V:NEXT
120 FOR L=0 TO S
125 NEXT
130 FOR V=0 TO 15 STEP D: POKE 54296,15-V: NEXT
140 POKE 54296,0
```

Vibrato/tremulo

Il vibrato e il tremulo agiscono solo durante la fase sostenuta di un tono. Il vibrato è un passaggio in cui il volume è alzato ed abbassato velocemente. Caricate le seguenti righe per produrre un vibrato:

```
121 FOR V=0 TO S
122 FOR W=0 TO 7: POKE 54296,15-W: NEXT
```

```
123 FOR W=8 TO 15: POKE 54296,W: NEXT
125 NEXT
```

Il tremulo è un vibrato veloce. Modificando le righe 122 e 123 in questo modo

```
122 FOR W=15 TO 8 STEP -J: POKE 54296, W: NEXT
123 FOR W=8 TO 15 STEP J: POKE 54296, W: NEXT
```

ed aggiungendo una riga di input per caricare l'intervallo del vibrato

```
1 INPUT "INTERVALLO DEL VIBRATO"; J
```

si potranno variare questi parametri all'inizio di ogni tono. Caricando valori più alti aumenterà la velocità del vibrato, mentre valori più bassi la diminuiranno. Non caricate valori maggiori di 15, non avrebbero alcun effetto. Uno 0 genera un loop senza fine.

COME MESCOLARE I TONI

In questo capitolo abbiano già visto tre toni armonizzati per produrre un accordo. È anche possibile combinare toni in modo da creare suoni completamente nuovi.

Battimento

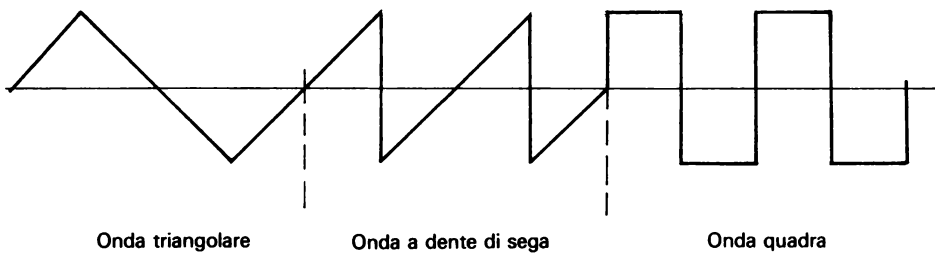
Un battimento è un fenomeno prodotto quando due toni vicini sono suonati contemporaneamente. Provate il seguente esempio:

```
5 PRINT"□"
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54275,8: POKE 54296,15
25 POKE 54282,8
30 POKE 54278,240: POKE 54276,65
35 POKE 54285,240: POKE 54283,65
40 INPUT " FREQUENZA DI PARTENZA";F
50 FF=F/.06097
60 H=INT(FF/256):L=INT((FF/256-H)*256)
80 FOR G=F-100 TO F+100
85 FO=G/.06097
87 HO=INT(FO/256):LO=INT((FO/256-HO)*256)
88 POKE 54279,LO: POKE 54280,HO: NEXT
90 FOR R=54272 TO 54296: POKE R,0: NEXT
```

Questo permetterà di sentire l'effetto di note (o frequenze) diverse su se stesse. Ascoltate il battimento che avviene con alcune frequenze: è causato dall'interazione tra di esse.

SELEZIONE DELLA FORMA D'ONDA

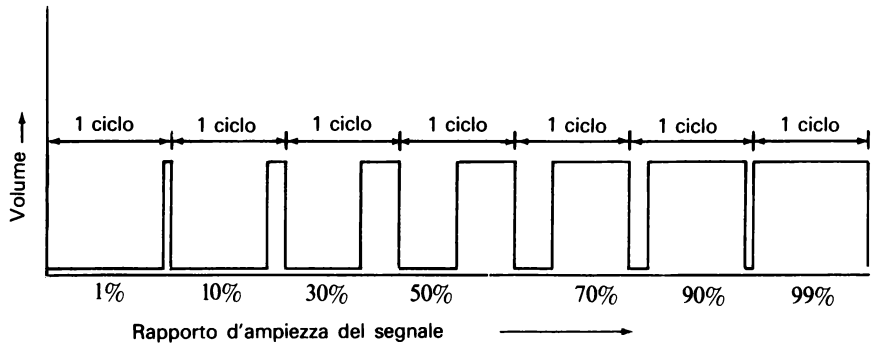
Fino ad ora si sono prodotti suoni impiegando solo una forma d'onda: l'onda quadra. Il C-64 è in grado di produrne altre tre: onde triangolari, dentellate (a dente di sega), e rumore bianco (fruscio). Osservando le sagome delle onde, è facile comprendere come esse influiscano sul suono. Per esempio, le onde quadre usate finora iniziano rapidamente, rimangono al livello massimo esattamente per metà del loro ciclo e poi decadono bruscamente, restando al minimo per l'altra metà.



Le onde quadre sono chiamate simmetriche quando i due livelli di emissione sono di eguale durata; questa simmetria può essere modificata cambiando i registri della lunghezza dell'impulso. Per la voce #1, i registri sono 54274 e 54275.
Ecco una routine che suona una sola nota variando la durata dell'impulso da un estremo all'altro con incrementi di 16.

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54272,15: POKE 54273,10
30 POKE 54296,15: POKE 54278,240
40 POKE 54276,65
50 FOR R=0 TO 4095 STEP 16
60 H=INT(R/256):L=R-256*H
70 POKE 54274,L: POKE 54275,H: NEXT
80 FOR G=F-100 TO F+100
90 FOR R=54272 TO 54296: POKE R,0:NEXT
```

In questo programma, il suono comincia come un ronzio, poi diviene più pieno; infine, all'estremo della variazione, il suono torna nuovamente ad essere un ronzio.



I registri delle onde triangolari, a dente di sega e del rumore bianco non sono influenzati dall'ampiezza dell'impulso. Per attivare l'onda triangolare, cambiate la riga 40 per agire sul registro di controllo della forma d'onda (54276).

```
40 POKE 54276, 17
```

Per la forma a dente di sega, POKE 54276 con 33 invece di 17.

```
40 POKE 54276, 33
```

Per ottenere il rumore bianco, caricate 129 invece di 33.

```
40 POKE 54276, 129
```

Uso del registro del rumore

Il registro del rumore funziona esattamente come quello dei toni. Ecco il programma della palla che rimbalza; questa volta impiega il registro del rumore.

```
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54274,0: POKE 54275,8
30 POKE 54278,240:POKE 54277,0
40 POKE 54296,15
50 POKE 54272,67: POKE 54273,12
60 FOR T=0 TO 15
70 POKE 54276,128
80 FOR F=0 TO 400: NEXT
90 POKE 54276,129
100 NEXT
110 FOR R=54272 TO 54296: POKE R,0:NEXT
```

Ora si aggiunga un po' di Decay al suono.

```
30 POKE 54278, 0: POKE 54277, 9
95 FOR N=0 TO 500: NEXT
```

Modificando anche la durata del ciclo, si può simulare il rumore di un treno come segue:

```
80 FOR F=0 TO 400-26*T: NEXT
```

Mischiando i suoni si ottiene anche il fischio del treno. Aggiungete le seguenti righe al programma:

```
11 POKE 54285,0: POKE 54284,13
12 POKE 54280,16: POKE 54279,195
110 FOR T=0 TO 150
120 POKE 54276,128
125 IF T=30 OR T=35 OR T=70 OR T=75 THEN 180
130 POKE 54276,129
140 POKE 54296,15-T/10
150 FOR N=0 TO 500-T*3: NEXT
160 NEXT
170 GOTO 200
180 POKE 54283,32: POKE 54283,33
190 GOTO 130
200 FOR R=54272 TO 54296: POKE R,0:NEXT
```

7.3. Programmare la musica con il C-64

Caricando valori e pause per mezzo di POKE nei registri del suono si possono scrivere dei programmi musicali.

```
5 REM FISSA I REGISTRI
6 PRINT"□"
7 PRINTSPC(10);"YANKEE DOODLE"
10 FOR R=54272 TO 54296: POKE R,0:NEXT
20 POKE 54278,240: POKE54277,0
30 POKE 54296,15
35 REM FA INIZIARE LA MUSICA
40 POKE 54272,134: POKE 54273,33
50 POKE 54276,17
60 FOR R=0 TO 200: NEXT
70 POKE 54276,16
80 FOR R=0 TO 100: NEXT
90 POKE 54276,17
100 FOR R=0 TO 200: NEXT
```

```
110 POKE 54276,16
120 FOR R=0 TO 100: NEXT
130 POKE 54272,161: POKE 54273,37
140 POKE 54276,17
150 FOR R=0 TO 200: NEXT
160 POKE 54276,16
170 FOR R=0 TO 100: NEXT
180 POKE 54272,60: POKE 54273,42
190 POKE 54276,17
200 FOR R=0 TO 200: NEXT
210 POKE 54276,16
220 FOR R=0 TO 100: NEXT
230 POKE 54272,134: POKE 54273,33
240 POKE 54276,17
250 FOR R=0 TO 200: NEXT
260 POKE 54276,16
270 FOR R=0 TO 100: NEXT
280 POKE 54272,60: POKE 54273,42
290 POKE 54276,17
300 FOR R=0 TO 200: NEXT
310 POKE 54276,16
320 FOR R=0 TO 100: NEXT
330 POKE 54272,161: POKE 54273,37
340 POKE 54276,17
350 FOR R=0 TO 600: NEXT
360 POKE 54276,16
370 FOR R=0 TO 100: NEXT
380 POKE 54272,134: POKE 54273,33
390 POKE 54276,17
400 FOR R=0 TO 200: NEXT
410 POKE 54276,16
420 FOR R=0 TO 100: NEXT
430 POKE 54276,17
440 FOR R=0 TO 200: NEXT
450 POKE 54276,16
460 FOR R=0 TO 100: NEXT
470 POKE 54272,161: POKE 54273,37
480 POKE 54276,17
490 FOR R=0 TO 200: NEXT
500 POKE 54276,16
510 FOR R=0 TO 100: NEXT
520 POKE 54272,60: POKE 54273,42
530 POKE 54276,17
540 FOR R=0 TO 200: NEXT
550 POKE 54276,16
560 FOR R=0 TO 100: NEXT
570 POKE 54272,134: POKE 54273,33
580 POKE 54276,17
590 FOR R=0 TO 600: NEXT
600 POKE 54276,16
610 FOR R=0 TO 100: NEXT
620 POKE 54272,164: POKE 54273,31
630 POKE 54276,17..
```

```
640 FOR R=0 TO 600: NEXT
650 POKE 54276,16
660 FOR R=0 TO 100: NEXT
670 POKE 54272,134: POKE 54273,33
680 POKE 54276,17
690 FOR R=0 TO 200: NEXT
700 POKE 54276,16
710 FOR R=0 TO 100: NEXT
720 POKE 54276,17
730 FOR R=0 TO 200: NEXT
740 POKE 54276,16
750 FOR R=0 TO 100: NEXT
760 POKE 54272,161: POKE 54273,37
770 POKE 54276,17
780 FOR R=0 TO 200: NEXT
790 POKE 54276,16
800 FOR R=0 TO 100: NEXT
810 POKE 54272,60: POKE 54273,42
820 POKE 54276,17
830 FOR R=0 TO 200: NEXT
840 POKE 54276,16
850 FOR R=0 TO 100: NEXT
860 POKE 54272,191: POKE 54273,44
870 POKE 54276,17
880 FOR R=0 TO 200: NEXT
890 POKE 54276,16
900 FOR R=0 TO 100: NEXT
910 POKE 54272,60: POKE 54273,42
920 POKE 54276,17
930 FOR R=0 TO 200: NEXT
940 POKE 54276,16
950 FOR R=0 TO 100: NEXT
960 POKE 54272,161: POKE 54273,37
970 POKE 54276,17
980 FOR R=0 TO 200: NEXT
990 POKE 54276,16
1000 FOR R=0 TO 100: NEXT
1010 POKE 54272,134: POKE 54273,33
1020 POKE 54276,17
1030 FOR R=0 TO 200: NEXT
1040 POKE 54276,16
1050 FOR R=0 TO 100: NEXT
1060 POKE 54272,164: POKE 54273,31
1070 POKE 54276,17
1080 FOR R=0 TO 200: NEXT
1090 POKE 54276,16
1100 FOR R=0 TO 100: NEXT
1120 POKE 54272,29: POKE 54273,25
1130 POKE 54276,17
1140 FOR R=0 TO 200: NEXT
1150 POKE 54276,16
1160 FOR R=0 TO 100: NEXT
1170 POKE 54272,48: POKE 54273,28
```

```
1180 POKE 54276,17
1190 FOR R=0 TO 200: NEXT
1200 POKE 54276,16
1210 FOR R=0 TO 100: NEXT
1220 POKE 54272,164: POKE 54273,31
1230 POKE 54276,17
1240 FOR R=0 TO 200: NEXT
1250 POKE 54276,16
1260 FOR R=0 TO 100: NEXT
1270 POKE 54272,134: POKE 54273,33
1280 POKE 54276,17
1290 FOR R=0 TO 600: NEXT
1300 POKE 54276,16
1310 FOR R=0 TO 100: NEXT
1320 POKE 54276,17
1330 FOR R=0 TO 600: NEXT
1340 POKE 54276,16
1350 FOR R=54272 TO 54296: POKE R,0:NEXT
```

Questo brano musicale non dura molto, ma il programma che lo genera è piuttosto lungo. Cercando di caricare tutte le note di una canzone in questa maniera, probabilmente si esaurirebbe la memoria. Un sistema migliore consiste nell'uso di istruzioni DATA. Un programma di questo tipo richiede solo tre brevi sezioni: una routine che legge le note, una per suonarle e una che contiene le note come dati. Provate questa versione del brano musicale.

```
5 PRINT"□"
6 PRINT SPC(10);"YANKEE DOODLE"
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54278,240: POKE 54296,15
30 DATA 134,33,200,134,33,200,161,37,200,60,42,
        200,134,33,200,60,42,200,161,37
40 DATA 600,134,33,200,134,33,200,161,37,200,60,42,
        200,134,33,600,164,31,600
50 DATA 134,33,200,134,33,200,161,37,200,60,42,
        200,191,44,200,60,42,200,161,37
60 DATA 200,134,33,200,164,31,200,29,25,200,48,28,
        200,164,31,200,134,33,600,134
70 DATA 33,600,999,999,999,999
80 READ A,B,C
90 IF A=999 THEN 160
100 POKE 54272,A:POKE 54273,B
110 POKE 54276,17
120 FOR R=0 TO C: NEXT
130 POKE 54276,16
140 FOR R=0 TO 100: NEXT
150 GOTO 80
160 FOR R=54272 TO 54296: POKE R,0: NEXT
```

Cambiando i valori nelle istruzioni DATA, si può suonare qualsiasi brano. Caricate il valore 999 per concludere il brano, come mostrato sopra.

PROGRAMMARE UN RITMO

La programmazione di un ritmo con il C-64 è molto simile a quanto abbiamo fatto prima per il rumore del treno. Cominciamo con un semplice rullo di tamburo.

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54277,6: POKE 54296,15
30 POKE 54272,15: POKE 54273,2
40 DATA 375,150,165,999
50 READ A
60 IF A=999 THEN 100
70 POKE 54276,129
80 FOR R=0 TO A: NEXT
90 POKE 54276,128 : GOTO 50
100 RESTORE: GOTO 50
```

Usando un altro tono nello stesso registro si può aggiungere una grancassa:

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54277,6: POKE 54296,15
30 POKE 54272,15: POKE 54273,2
40 DATA 33,375,129,150,129,165,999,999
50 READ A,B
60 IF A=999 THEN 100
70 POKE 54276,A
80 FOR R=0 TO B: NEXT
90 POKE 54276,0 : GOTO 50
100 RESTORE: GOTO 50
```

Cambiando la variabile del loop varierà il ritmo.

```
80 FOR R=0 TO B/2: NEXT
```

Aggiungendo istruzioni READ e DATA supplementari al programma del ritmo è possibile includere una melodia. Ecco il listato completo.

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT R
20 POKE 54277,40: POKE 54296,15
30 POKE 54272,15: POKE 54273,3
35 POKE 54285,240: POKE 54283,17
40 DATA 16,185,17,375,    18,208,129,150
41 DATA 18,208,129,150,    19,238,17,375
50 DATA 19,238,129,150,    19,238,129,150
51 DATA 18,208,17,375,    18,208,129,150
60 DATA 18,208,129,150,    16,195,17,375
61 DATA 16,195,129,150,    16,195,129,150
70 DATA 16,195,17,375,    18,208,129,150
71 DATA 18,208,129,150,    19,238,17,375
80 DATA 25,29,129,150,    25,29,129,150
81 DATA 18,208,17,375,    19,238,129,150
90 DATA 19,238,129,150,    16,195,17,375
100 DATA 999,999,999,999
160 READ H,L,R,N
170 IF H=999 THEN 230
180 POKE 54279,L: POKE 54280,H
190 POKE 54276,R
200 FOR M=0 TO N: NEXT
210 POKE 54276,0
220 GOTO 160
230 RESTORE: GOTO 160
```

Bisogna tener presente, lavorando con questo tipo di programma, che l'aggiunta di altre istruzioni all'interno dei loop comporta un tempo d'esecuzione maggiore. Istruzioni di test, come quella a riga 17Ø, impiegano anch'esse tempo supplementare. Verificate d'aver tenuto conto di ogni riga: se siete incerti sul ritmo, eseguite il programma ed ascoltate per verificare gli errori. Normalmente è possibile compensare gli errori variando i valori dei loop.

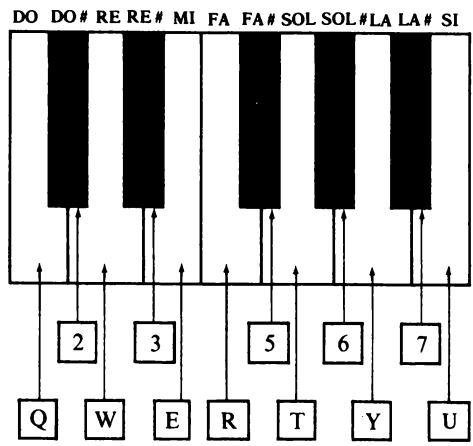
L'ORGANO ELETTRONICO DEL C-64

Ecco un programma che impiega GET per acquisire note dalla tastiera e suonarle. Si sono usati i valori delle note riportati nella tabella 7.2, inseriti nei registri del suono.

```
10 FOR R=54272 TO 54296: POKE R,0: NEXT
20 POKE 54278,240
30 GET A$: IF A$="" THEN 30
40 IF A$="Q" THEN U=8:L=97
50 IF A$="2" THEN U=8:L=225
60 IF A$="W" THEN U=9:L=104
70 IF A$="3" THEN U=9:L=247
80 IF A$="E" THEN U=10:L=143
90 IF A$="R" THEN U=11:L=47
```

```
100 IF A$="5" THEN U=11:L=218
110 IF A$="T" THEN U=12:L=142
120 IF A$="6" THEN U=13:L=77
130 IF A$="Y" THEN U=14:L=24
140 IF A$="7" THEN U=14:L=238
150 IF A$="U" THEN U=15:L=210
160 POKE 54272,L:POKE 54273,U
170 POKE 54296,15:POKE 54276,17
180 FOR G=0 TO 500 STEP .5: NEXT
190 POKE 54296,0: POKE 54276,16
200 GOTO 30
```

Questo programma è stato scritto per una sola ottava. Aumentando il numero di tasti letti, si possono aggiungere più ottave. Per far suonare più a lungo i tasti, si può allungare il loop di ritardo a riga 19Ø diminuendo lo STEP frazionario.



REGISTRAZIONE DELLA MUSICA

Un'alternativa allo scrivere un gran numero di istruzioni DATA è quella di usare l'organo elettronico per memorizzare le proprie composizioni musicali. Ora si spiegherà come memorizzarle su nastro o dischetto.

Come creare vettori musicali

Aggiungere le seguenti righe al vostro programma vi permetterà di memorizzare in un array unidimensionale o vettore le note suonate; qui è

impiegato un vettore di 100 note. Se si desidera, è possibile creare vettori contenenti un maggior numero di note, ma ricordate che ogni nota occupa due byte e che non è possibile creare un vettore maggiore dello spazio disponibile in memoria.

```
5 DIM U(100),L(100)
31 IF A$="■" THEN U(X)=U: L(X)=L: X=X+1: GOTO 30
32 IF A$<>"■" THEN 40
33 FOR Z=0 TO X
34 POKE 54272,L(Z): POKE 54273,U(Z)
35 POKE 54296,15: POKE 54276,17
36 FOR G=0 TO 500: NEXT
38 POKE 54296,0: POKE 54276,16: NEXT
39 GOTO 30
```

Ogni volta che si preme un tasto verrà ottenuto un suono. Premendo F1 si ripeteranno tutte le note presenti in memoria. Per aggiungere una nota, premete F3 dopo il tasto desiderato.

Come memorizzare la musica su nastro o dischetto

Quando si spegne il C-64, la musica memorizzata in macchina viene perduta. Se volete registrare i vostri brani e risentirli in seguito, sarà necessario memorizzarli su nastro o su dischetto. Aggiungendo le seguenti righe al vostro programma potrete caricare il brano musicale in un file di dati.

```
45 IF A$="■" THEN 300
300 PRINT"□": INPUT "NOME DEL FILE"; F$
310 INPUT "SALVA SU ■D■ISCO O ■C■ASSETTA";S$
320 IF S$="D" THEN 350
330 IF S$="C" THEN 360
340 PRINT "□";:GOTO 310
350 OPEN1,8,4,F$+",W": GOTO 370
360 OPEN1,1,1,F$
370 FOR Z=0 TO X
380 PRINT#1, U(Z): PRINT#1,L(Z): NEXT
390 CLOSE 1: GOTO 30
1000 OPEN15,8,15:INPUT#15,A,B$,C,D:
     PRINTB$:CLOSE15: END
```

Premendo il tasto di funzione F5, il programma richiederà il nome di un file e la sua ubicazione (Datassette o dischetto). Memorizzerà poi il contenuto del brano musicale nel file specificato.

Come leggere la musica da nastro o dischetto

Per leggere un brano da un file, sarà necessario trasferire i dati ad un vettore che può essere interpretato dal programma di organo elettronico. La seguente routine esegue quest'operazione:

```
55 IF A$="■"THEN 400
400 PRINT"□":   INPUT "NOME DEL FILE"; F$
410 INPUT "CARICA DA □D■ISCO O DA □C■ASSETTA";S$
420 IF S$="D" THEN 450
430 IF S$="C" THEN 460
440 PRINT "□";:GOTO 410
450 OPEN1,8,4,F$+",R": GOTO 470
460 OPEN1,1,0,F$ : J=0
470 INPUT#1, U(J): INPUT#1,L(J):BB=ST:J=J+1
480 IF BB=0 THEN 470
490 X=J:CLOSE 1: GOTO 30
1000 OPEN15,8,15:INPUT#15,A,B$,C,D:
     PRINTB$:CLOSE15: END
```

Una volta aggiunte queste istruzioni al vostro programma, potete utilizzare il tasto F7 per caricare in memoria brani salvati in precedenza su nastro o dischetto.

7.4. Abbinare il suono con l'animazione

Nei programmi di video giochi il suono è utilizzato molto frequentemente. I suoni possono essere impiegati per creare un'atmosfera o per dare al giocatore maggiori informazioni su ciò che accade sullo schermo.

SINCRONIZZAZIONE

Durante un gioco, l'operatore è più attento al suono che all'immagine. Provate questo gioco della palla che rimbalza senza suono.

```
5 PRINT"□"
10 A$="                                        "
20 FOR R=0 TO 2: PRINT A$;: NEXT
30 FOR T=1104 TO 1143
40 FOR Y=0 TO 10: NEXT
50 POKE T-1,32: POKE T,81: NEXT
60 FOR T=1143 TO 1104 STEP -1
70 FOR Y=0 TO 10: NEXT
80 POKE T+1,32: POKE T,81: NEXT
90 GOTO 30
```

Ora aggiungete il suono con le seguenti righe:

```
15 FOR I=54272 TO 54296: POKE I,0: NEXT
16 POKE 54272,12: POKE 54273,36
17 POKE 54296,15: POKE 54278,240
55 POKE 54276,17: POKE 54276,0
85 POKE 54276,17: POKE 54276,0
```

L'aggiunta di suono aiuta a creare l'effetto di una palla che rimbalza. Per sincronizzare il suono con un oggetto visualizzato sullo schermo, sarà necessario tener conto della sua posizione sul video. Nell'esempio precedente, la sincronizzazione è semplice perché il suono è realizzato alla fine di ogni loop. Per sincronizzare suoni semplici con movimenti, i suoni devono essere generati quando un oggetto entra in collisione con un altro. Il seguente programma determina il movimento di oggetti e il suono, basandosi sulla loro posizione sul video:

```
10 FOR I=54272 TO 54296: POKE I,0: NEXT
20 POKE 54272,12: POKE 54273,36
25 POKE 54279,12: POKE 54280,100
30 POKE 54296,15: POKE 54278,240
35 POKE 54285,240
40 A$="                                "
50 PRINT"□";:FOR R=0 TO 5: PRINTA$;: NEXT
60 B1=1104: B2=1144: R1=1: R2=.7
70 POKE B1,81: POKE B1-1,32: POKE B1+1,32
80 POKE B2,81: POKE B2-1,32: POKE B2+1,32
90 B1=B1+R1: B2=B2+R2
100 IF B1=1104 OR B1=1143 THEN R1=R1*-1:
    POKE 54276,17: POKE 54276,0
110 IF B2<1144.5 OR B2>1182.5 THEN R2=R2*-1:
    POKE 54283,17: POKE 54283,0
120 GOTO 70
```

Dato che sono stati usati valori differenti, è facile distinguere quale oggetto ha appena urtato la parete.

Capitolo

Le periferiche 8

Il C-64 si può collegare a vari dispositivi, come il Datassette, l'unità a dischetti e la stampante MPS 801, che ne aumentano le capacità fornendo la possibilità di fare copie permanenti, su carta o supporti magnetici, di programmi e dati. I supporti magnetici aumentano lo spazio di memoria disponibile. Con l'aggiunta di un *modem*, il C-64 è in grado di comunicare, per mezzo di linee telefoniche, con qualsiasi altro computer.

MEMORIZZAZIONE DEI DATI

I dispositivi più comuni per memorizzare permanentemente dei dati sono i nastri e i dischetti magnetici. I dischetti hanno il vantaggio di essere dispositivi ad *accesso casuale* (random); si può accedere cioè direttamente ai dati ovunque memorizzati sulla loro superficie. I nastri invece memorizzano i dati sequenzialmente e perciò si deve far scorrere il nastro fino al punto a cui si vuole accedere. I sistemi a nastro, meno flessibili, sono molto meno costosi dei dischetti.

8.1. I file

In un computer tutti i dati sono conservati e organizzati nei file esattamente come le informazioni su supporto cartaceo sono conservate negli archivi.
Il modo nel quale i dati sono organizzati nei file può essere paragonato a

quello tradizionale: un dato elementare (byte) viene scritto in una casella (campo) su una scheda (record) che insieme a tutte le altre della stessa specie viene conservata in un classificatore (file).
Questo è il metodo principale di gestione delle informazioni nei computer e anche il C-64 non sfugge a questa regola.
I file di dati del C-64 possono avere nomi lunghi fino a 16 caratteri. La lunghezza di un file è limitata solo dallo spazio disponibile sul dischetto o nastro. La directory (catalogo) del dischetto può contenere un massimo di 144 nomi diversi di file. Essi sono di due tipi: file di programma e file di dati.

FILE DI PROGRAMMA

Quando si ha un programma che si vuole trasferire dalla memoria centrale ad un'unità di memoria, si effettua un SAVE su nastro o dischetto. Per riportarlo in memoria lo si carica con LOAD. Ogni programma deve avere un nome distinto in modo che il computer possa identificarlo. Impiegando il Datassette non è necessario usare un nome perché al computer può essere ordinato di caricare il primo programma che trova, mentre con i dischetti ogni file deve avere un suo nome.
Generalmente la dimensione massima di un programma è limitata dalla memoria disponibile nel computer. Un modo di trattare programmi troppo grossi per la memoria centrale del C-64 è di dividerli in sezioni più piccole e caricarle ed eseguirle separatamente. Anche se non è semplice, in questo modo è possibile usare programmi che sono molto più grossi dello spazio disponibile. È necessario che la prima sezione del programma caricata sia più lunga delle sezioni che verranno caricate dopo, perché le variabili di un programma sono memorizzate alla fine dello stesso; se in seguito viene caricata una routine più lunga, le variabili o parte del programma caricato verranno perse. Per far caricare la sezione successiva di un programma dal programma stesso, basta terminare la prima sezione con un'istruzione LOAD:

LOAD "*nome del programma successivo*",8

Questa caricherà ed eseguirà la successiva.

FILE DI DATI

I file di dati non contengono programmi, quindi non possono essere caricati ed eseguiti; essi contengono solo dati che devono essere caricati in memoria da un programma o caricati dall'operatore con un'istruzione in modo immediato.

Campi e record

Le informazioni in un file di dati possono essere divise in record e campi; ciò è determinato dal programma che legge i dati del file. Immaginate un file di dati che contiene informazioni su un volo da Milano a New York e ritorno.

Mr. G. Rossi	MIL	NYK	Flight 600	12.00	6 giugno 1984
Mr. G. Rossi	NYK	MIL	Flight 601	19.30	21 giugno 1984

Il file contiene dati sul volo che il sig. G. Rossi farà da Milano a New York e ritorno. Per semplificare, immaginate che si tratti di un solo file che contiene due record: il volo d'andata e il volo di ritorno da New York. Ogni record contiene diversi campi, ovvero gruppi di caratteri che formano parole o numeri completi. Per esempio, la parola "GIUGNO" contiene 6 caratteri (o byte) che sono memorizzati sul nastro o dischetto come le lettere individuali G,I,U,G,N,O. Logicamente, però, le lettere devono essere prelevate come una parola sola "GIUGNO", come le cifre 6,0,0 devono essere lette come 600. Per distinguere i dati all'interno di un record, è importante che i campi siano separati.

TRASFERIMENTO DI DATI

La prima volta che si accede ad un file su nastro o dischetto, si può essere indotti a credere che il sistema non funzioni correttamente. Infatti ci aspettiamo che l'unità di memoria entri in funzione in perfetta sincronia con le letture o le registrazioni effettuate dal programma; ciò non sempre accade, anzi.
Molto spesso avvertiamo un sensibile ritardo tra il comando di registrazione e l'attivazione del drive del dischetto. Questo è provocato da una memoria "tampone" o buffer che ottimizza l'accesso del C-64 alle periferiche, minimizzandone l'uso.
I dati da registrare vengono accumulati nel buffer fino al suo completo riempimento; a questo punto l'intero contenuto del buffer viene trasferito con un'unica operazione alla periferica interessata.

FILE LOGICI E DISPOSITIVI PERIFERICI

Viene definita programmazione di input/output una qualsiasi forma di software che controlli il trasferimento di dati tra computer e dispositivi

periferici, come il Datassette, l'unità di gestione dei dischetti o la stampante, tutte unità fisiche esterne. Per trasferire dati da o ad una di queste, è necessario indicare a quale si vuole accedere, perché ciascuna unità possiede una interfaccia con particolari modalità di ricevimento messaggi. Per di più, se vi è collegata più di una periferica, il computer deve conoscere quale tra esse è interessata alla particolare operazione di trasferimento.
Volendo caricare dati dalla tastiera, impiegheremmo un'istruzione BASIC come

```
10 INPUT A
```

Questa riga arresta l'esecuzione del programma ed attende che l'operatore carichi qualche dato e prema RETURN. I dati sono assegnati alla variabile A. Se il C-64 non riceve istruzioni contrarie, tutte le istruzioni di questo tipo aspettano dati provenienti dalla tastiera. Desiderando eseguire un output al video, si potrebbe usare un'istruzione:

```
20 PRINT A
```

Il valore della variabile A apparirebbe così sul video. L'istruzione INPUT ordina al computer di ricevere dei dati e l'istruzione PRINT ordina di emettere dei dati.
Benché la maggior parte dei dati siano caricati dalla tastiera e i risultati mandati al video, queste due non sono le uniche unità disponibili. Prima di poter "comunicare con" un altro dispositivo, è necessario aprire con un'istruzione OPEN, un *canale logico* a quel dispositivo.
Quest'istruzione designa un canale ed il relativo dispositivo caratterizzato da un numero.
La struttura di un'istruzione OPEN si presenta come segue:

10 OPEN *fn, dn, sa, nome del file*

dove:

fn	è il numero del canale usato per accedere ai dati. Qualsiasi numero tra 0 e 255 è accettabile
dn	è il numero del dispositivo; ovvero il numero dell'unità periferica che si desidera attivare
sa	è l'indirizzo secondario utilizzato dal dispositivo
nome del file	il nome del file è utilizzato per specificare il file durante le operazioni di lettura e scrittura. Specificando il nome del file durante la lettura o la scrittura sul Datassette, tutti gli altri verranno saltati fino a che sarà trovato quello specificato.

La tabella 8.1 mostra il numero di dispositivo e gli indirizzi secondari che corrispondono ai dispositivi logici del C-64.

Tabella 8.1. Numeri e indirizzi secondari delle unità periferiche

Unità	*Numero del dispositivo*	*Indirizzo secondario*	*Funzione*
Tastiera	0	Nessuno	
Datassette	1	0	Aperto per lettura dati
		1	Aperto per scrittura dati
		2	Aperto per scrittura dati, ma con controllo di fine file
	2		
Schermo	3	Nessuno	
Stampante MPS 801	4 o 5	7	Set di caratteri alternativo
Unità a dischetti	8*	0	Istruzione LOAD
		1	Istruzione SAVE
		2-14	Non assegnati
		15	Canale di servizio
Altri dispositivi allacciati alla porta IEEE 488	5,6,7 e da 9 a 31		L'indirizzo dei dispositivi è assegnato dalle singole interfacce IEEE 488
	da 12 a 255 non (ancora) utilizzati		

*Normalmente 8, ma può essere utilizzato anche 9, 10 e 11 (v. sezione Sistemi a dischetti multipli)

FILE SUL DATASSETTE

Vediamo ora l'applicazione dei file al Datassette. Il Datassette è il dispositivo di memoria esterna di base: ovvero, le informazioni saranno trasferite ad esso, se non viene specificato alcun numero di dispositivo. Per esempio, per trasferire un programma chiamato 'NOME DEL FILE' al Datassette, si batte:

```
SAVE "NOME DEL FILE"
```

e RETURN. Il Datassette comincia ad operare dopo che sono stati premuti i tasti PLAY e RECORD. Finita la trascrizione, si fermerà e il cursore ricomincerà a lampeggiare mentre il computer presenterà READY sullo schermo.
Per caricare il programma dal nastro, battete:

```
LOAD "NOME DEL FILE"
```

Dopo aver premuto il tasto PLAY, il registratore si metterà in funzione e il computer presenterà:

```
SEARCHING FOR NOME DEL FILE
```

Mano a mano che trova i vari programmi che precedono quello cercato, presenterà i loro nomi.

```
FOUND PRIMO FILE
FOUND SECONDO FILE
FOUND TERZO FILE
```

Quando trova quello desiderato, sul video apparirà:

```
FOUND NOME DEL FILE
LOADING
```

Scrittura dei file di dati

Il trasferimento di file di programmi al Datassette è un'operazione semplice. Fare altrettanto con file di dati è quasi la stessa cosa, ma richiede una conoscenza del modo in cui le informazioni sono registrate sul nastro. Il metodo più comune è quello di trascrivere ogni dato individualmente, con le seguenti istruzioni. Come prima cosa, è necessario aprire un file.

```
10 OPEN 1,1,2,"DATA FILE"
```

Così verrà aperto un file al Datassette con il numero 1 e il nome DATA FILE. Quando un file viene chiuso, viene trascritta una segnalazione di fine file, che indica al C-64 che nessun ulteriore record è presente in quel file. Ora si può caricare direttamente dalla tastiera.

```
20 INPUT A$
```

Tabella 8.2. Contenuto del registro di stato ST

Dispositivi e operazioni	*Stato*							
	00000001 1	00000010 2	00000100 4	00001000 8	00010000 16	00100000 32	01000000 64	10000000 128
Lettura da cassetta	Operazione OK	Operazione OK	Blocco di dati corto. Il blocco letto contiene meno dati del previsto	Blocco di dati lungo. Il blocco letto contiene più dati del previsto	Errore di lettura	Errore di parità. Uno o più bit letti non correttamente	Fine del file	Fine del nastro
Verifica cassetta					Errore di verifica		—	
Unità a dischetti	Dispositivo ricevitore non disponibile	Dispositivo trasmettitore non disponibile	—	—	—	—	Fine del file	Unità a dischetti non collegata

Se state caricando dati numerici, potete usare una variabile numerica come A. Ad un certo punto vorrete terminare l'input. Ammettiamo che, caricando XXX, il computer esca dalla routine.

```
30 IF A$ = "XXX" THEN 60
```

Se l'input non è XXX, lo si vuole memorizzare sul nastro

```
40 PRINT#1, A$
```

e tornare a caricare altri dati.

```
50 GOTO 20
```

Quando avete finito, chiudete il file.

```
60 CLOSE 1
```

Per eseguire il programma, si deve per prima cosa svolgere interamente il nastro e riavvolgerlo fino a vedere la giunzione tra nastro magnetico e la sezione iniziale non magnetica. Ecco il programma completo.

```
10 OPEN 1,1,2, "DATA FILE"
20 INPUT A$
30 IF A$="XXX" THEN 60
40 PRINT#1, A$
50 GOTO 20
60 CLOSE 1
```

Ora chiudete il coperchio del Datassette e fate eseguire il programma. Questo farà una pausa (e il cursore scomparirà) per qualche istante, mentre il file verrà aperto e il suo nome trascritto sul nastro. Il cursore ritornerà insieme a un punto interrogativo.
Caricate alcuni caratteri e premete RETURN; dopo caricate XXX. Il cursore dovrebbe sparire di nuovo; questa volta il file viene chiuso e la segnalazione di fine file viene scritta sul nastro.

Lettura dei file di dati

È necessario, per prelevare i dati dal nastro, aprire un canale di lettura. Aggiungete le seguenti istruzioni al programma di trasferimento dati:

```
70 PRINT "RIAVVOLGI IL NASTRO"
80 PRINT "QUANDO IL NASTRO E'"
```

```
90 PRINT "RIAVVOLTO, PREMI STOP"
100 PRINT "SUL DATASSETTE"
110 PRINT "POI PREMI <RETURN>"
120 INPUT C
130 OPEN 1,1,0, "DATA FILE"
```

Le righe da 70 a 110 comunicano all'operatore di riavvolgere il nastro e di premere RETURN per segnalare al C-64 la fine di quest'operazione. La riga 120 fa attendere il computer fino al completo riavvolgimento del nastro e la riga 130 apre il file per la lettura. Ora può iniziare la lettura dei dati dal Datassette.

```
140 INPUT#1, A$
```

Questo carica il primo record memorizzato e lo assegna alla variabile A$. L'istruzione

```
150 PRINT A$
```

preleva i dati appena caricati in A$ e li visualizza.

IL REGISTRO DI STATO

Vi è un nome speciale di variabile (simile a TI e TI$) che indica lo *stato* dei dispositivi esterni collegati al C-64: è la variabile ST. La tabella 8.2 interpreta i valori di ST. Per determinare se si è arrivati al termine di un file, è possibile controllare il valore di ST dal programma; se il valore è 64, significa che i dati sono stati letti tutti. Aggiungete la seguente riga:

```
160 IF ST <> 64 THEN 150
```

Il vostro programma salterà indietro a leggere un altro record nel file; se ST è 64, tutti i dati sono stati letti e visualizzati. Chiudete il file,

```
170 CLOSE 1
```

seguite quindi le istruzioni sullo schermo. Il computer visualizzerà tutti i dati caricati nella precedente sezione del capitolo.

USO DI GET# PER LEGGERE I FILE

È pure possibile far uso di GET, esattamente come si farebbe con la tastiera. L'istruzione GET# legge un byte alla volta dal Datassette. Cambiate la riga 14Ø così:

```
140 GET#1, A$
```

e, in modo immediato, battete

```
GOTO 70
```

Questa volta tutti i dati sono stampati in colonna. Come mai? Si osservi la riga 14Ø. La funzione GET# carica solo un carattere alla volta che viene poi assegnato alla variabile A$ e stampato. Il computer esegue un ritorno a capo e il successivo carattere è caricato con il nuovo GET#. Alla fine di ogni stringa, il computer aggiunge automaticamente un ritorno a capo, che funge da delimitatore, separando una stringa dalla successiva; procedimento che gli permette di distinguere le variabili. In alcune applicazioni questo ritorno si può eliminare. Per esempio, si potrebbe aver bisogno di risparmiare spazio in un file strapieno in cui tutte le variabili o stringhe sono della stessa lunghezza. In questi casi, voi stessi potreste eliminare il ritorno a capo e separare le variabili senza dimenticare che, creando un file senza delimitatori, è necessario che il programma separi i dati. Per sopprimere i ritorni a capo si può cambiare la riga 4Ø in modo da farla apparire:

```
40 PRINT#1, A$;
```

Nota: Generalmente i migliori programmi sono quelli più semplici; quindi siate ben sicuri di aver bisogno di utilizzare questa possibilità. Inoltre tenete a mente il fatto che un file senza delimitatori non può essere letto e caricato da un'istruzione INPUT# se contiene più di 80 caratteri perché il buffer di INPUT non ne contiene di più.

FILE SU DISCHETTO

L'unità di gestione dei dischetti è in grado di registrare e memorizzare dati come il Datassette. La differenza più significativa è che l'unità può accedere direttamente a qualsiasi posizione di memoria, mentre il Datassette deve procedere sequenzialmente.

Modalità di memorizzazione su dischetto

I dati sono memorizzati su dischetto in anelli concentrici, detti *tracce* (tracks). L'unità 1541 possiede un totale di 35 tracce. Si può accedere direttamente, e quindi velocemente, ad ogni traccia. In generale l'unità non registra sull'intera traccia perché ciò significherebbe allungare inutilmente i file; per evitare questo spreco, ogni traccia è divisa in un certo numero di *settori*, ciascuno formato da 256 byte.
Si osservi la figura 8.2: le tracce non sono tutte della stessa lunghezza; quelle più vicine al bordo esterno sono più lunghe. Per far miglior uso dello spazio disponibile, l'unità di gestione colloca sulle tracce esterne più lunghe più settori; il numero dei settori varia da 17 per le tracce in-

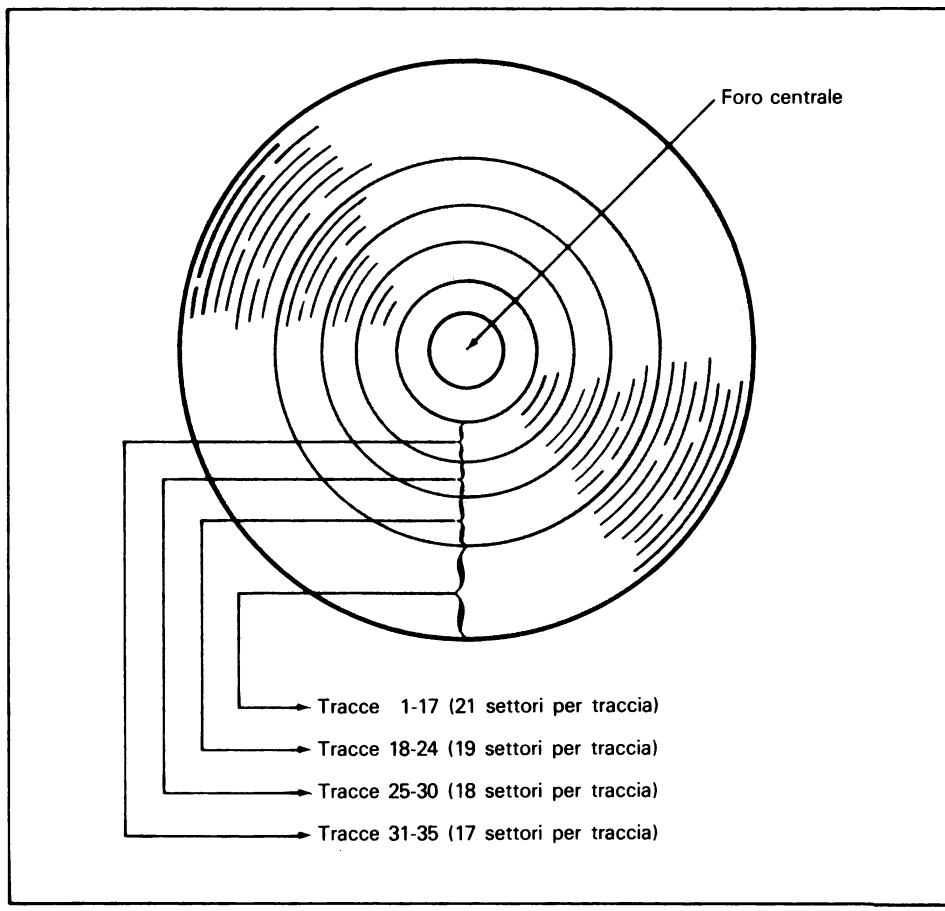


Figura 8.2. Schema del dischetto

terne a 21 per le esterne. Ciò significa che un dischetto è in grado di contenere 169984 byte nei suoi 664 settori.

I settori

Ruotando manualmente il dischetto nella sua busta, si osservano uno o più fori che si allineano con quello praticato nella busta. Se il foro è uno solo, il dischetto è *soft sectored*, se è più di uno, è *hard sectored*. Questi fori vengono usati dall'unità per posizionare le testine sul dischetto; se il foro è uno solo, esso serve solo per stabilire un inizio mentre i settori sono creati liberamente dall'unità per mezzo di un apposito software.
Nell'altro caso, ad ognuno dei numerosi fori è abbinato un settore e questa disposizione non può essere modificata dall'unità a dischetti.
Il C-64 adotta come standard l'unità 1541 che usa SOLO dischetti *soft sectored.*

Catalogo del dischetto

La traccia n° 18 sul dischetto 1541 è impiegata per il catalogo. Quest'ultimo memorizza i nomi, gli indirizzi d'inizio e il tipo di tutti i file presenti nel dischetto. Per listare il catalogo, lo si carica in memoria con

```
LOAD "$",8
```

Il nome del catalogo è $ (dollaro), che viene caricato dal dispositivo n° 8 (il drive dei dischetti). Per listarlo, battete semplicemente

```
LIST
```

e apparirà la lista di tutti i file.
Attenzione! Il catalogo viene visto dal C-64 come un programma quindi il fatto di caricarlo per consultazione determina la perdita del programma attualmente presente in memoria. È una caratteristica estremamente fastidiosa e può essere evitata solo utilizzando il programma di supporto del DOS (Disk Operating System) contenuto nel dischetto demo dell'unità 1541.

Schema della disponibilità dei blocchi - BAM (Block Availability Map)

La BAM risiede sulla traccia 18 del dischetto. Contiene informazioni sulla disponibilità dello spazio di memoria sul dischetto.

Inizializzazione

Ogni volta che si accede all'unità a dischetti, questa paragona il numero d'identificazione del dischetto con il numero di ID memorizzato nella sua memoria. Se questi due numeri sono uguali, l'unità procede senza altre verifiche, in caso contrario esegue automaticamente un'*inizializzazione* del dischetto. Quando viene inizializzato un dischetto, il contenuto della BAM viene copiato nella memoria dell'unità. Se i numeri di ID di due dischetti sono uguali, l'unità di gestione non inizializzerà il dischetto automaticamente, non aggiornerà la BAM con le informazioni di disposizione e potrebbe trascrivere sopra a settori di altri programmi o dati, contaminandoli. Per evitarlo, formattate i dischetti con numeri ID differenti, quando possibile. Per compiere un'inizializzazione manuale, eseguite le seguenti istruzioni:

```
OPEN 1,8,15
PRINT#1, "INITIALIZE"
```

Una versione abbreviata del comando sarebbe:

```
OPEN 1,8,15, "I"
```

Formattazione di un dischetto

Prima di poter usare un dischetto nuovo, lo si deve formattare. La formattazione trascrive il nome del dischetto, un numero ID e tutte le informazioni riguardanti le tracce e i settori sul dischetto in modo che possano esservi trascritti dei dati dal C-64. Per formattare un dischetto mai usato, procedete come segue:

```
OPEN 1,8,15
PRINT#1,"NEW:NOME DEL DISCO,ID"
```

Il nome del dischetto può essere una qualsiasi stringa di 16 caratteri, il numero ID può essere qualunque numero. Il DOS utilizza il numero ID per identificare il dischetto attualmente inserito nell'unità. Usate numeri ID sempre diversi, così il DOS potrà determinare se è necessario inizializzare o meno. Così facendo, quest'operazione sarà sempre automatica, altrimenti sarà necessario farlo manualmente ogni volta che si cambia dischetto. Un buon metodo consiste nel formattare un'intera scatola di dischetti nuovi con numeri ID in sequenza.
Quando necessario, si può eseguire una versione abbreviata della formattazione che cancella tutti i dati registrati e cambia il nome lasciando il

numero ID originale. In questo modo nessun dischetto nella scatola avrà mai lo stesso numero ID di un altro. Per compiere quest'operazione, usate la seguente istruzione.
Nota: Questo non funziona con dischetti nuovi mai formattati.

```
OPEN 1,8,15
PRINT#1,"N:NOME DEL DISCO"
```

Si noti che in questa istruzione la lettera N è stata usata come abbreviazione di NEW.

COME CAMBIARE IL NOME AL FILE

È possibile cambiare il nome di un file con l'istruzione RENAME, nel modo seguente:

```
OPEN 1,8,15
PRINT#1,"RENAME:NUOVO NOME = VECCHIO NOME"
```

La lettera R è permessa come abbreviazione di RENAME. Per cambiare il nome di un file da DOG.1 a CAT.1, si carica:

```
OPEN 1,8,15
PRINT#1,"R: CAT.1 = DOG.1"
```

CANCELLAZIONE DI FILE

Per cancellare un file da un dischetto, usate il comando SCRATCH. L'istruzione appropriata è

```
OPEN 1,8,15
PRINT#1,"SCRATCH:NOME DEL FILE"
```

È ammessa l'abbreviazione S per SCRATCH, per cui si può anche battere:

```
PRINT#1, "S: NOME DEL FILE"
```

per cancellare programmi o dati.

IL COMANDO VALIDATE

Si può impiegare il comando VALIDATE per riordinare un dischetto. Con l'uso il dischetto può contenere file non correttamente chiusi o altri usati temporaneamente, ma che non fanno più nemmeno parte di file esistenti. Per compiere le necessarie operazioni di riordino, usate il comando VALIDATE. Esso cancella i file non correttamente chiusi e blocchi di dati che non fanno parte di file ancora esistenti.

DUPLICAZIONE DI FILE

È possibile fare una copia di un file sul dischetto per mezzo del comando COPY. Lo stesso comando, con una sintassi leggermente diversa, può concatenare più file. Per eseguire una copia, battete la seguente sequenza d'istruzioni:

```
OPEN 15,8,15
PRINT#15, "COPY:NUOVO NOME = VECCHIO NOME"
```

Si può impiegare l'abbreviazione C al posto della parola COPY, come nel seguente esempio:

```
OPEN 15,8,15
PRINT#15, "C:DOG.2 = DOG.1"
```

CONCATENAMENTO DI FILE SUL DISCHETTO

Due o più file possono essere concatenati per formare un unico file più lungo. Per esempio, per creare un file nuovo chiamato NUOVO FILE, concatenando gli esistenti VECCHIO FILE 1 e VECCHIO FILE 2 sul dischetto, battete:

```
PRINT#1,"C:NUOVOFILE=VECCHIOFILE1,VECCHIOFILE2"
```

Nota: La lunghezza massima dell'istruzione è di 40 caratteri; impiegate, quindi nomi brevi per i file.

Sistemi a dischetti multipli

Possedendo un sistema con più di un'unità a dischetti, è possibile fare delle copie di file da un'unità all'altra. Per prima cosa, però, sarà neces-

sario poter differenziare le unità. Quando vengono accese, le unità 1541, sono contraddistinte dal numero 8. Collegando più di una unità al sistema, ciascuna di esse avrà bisogno di un proprio numero, che può essere 8, 9, 10 o 11. Per cambiare il numero di dispositivo di un'unità, procedete come segue:

1. Spegnere tutte le unità ad eccezione di quella della quale si vuole cambiare il numero.
2. Eseguire un comando del tipo:

```
OPEN 15,8,15
```

3. Caricare la seguente sequenza di comando.
   Nota: Non preoccupatevi di capire completamente il significato dell'istruzione: è spiegato nella sezione sui comandi di servizio.

```
PRINT#15, "M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(9+32)CHR$(9+64)
```

   Quest'istruzione cambia il numero di dispositivo a 9. Per impiegare un altro numero, caricate quel numero al posto di 9 nelle due parti terminali (+32 e +64) del comando.
4. Accendere la successiva unità e ripetere la sequenza sopra descritta con un numero differente.

Nota: Non spegnete un'unità il cui numero è stato cambiato: si cancellerebbe il nuovo numero assegnato.

FILE DI DATI SU DISCHETTO

Come per il Datassette, vi sono differenze nel modo in cui vengono trattati file di programma e dati dall'unità di gestione dischetti. Vi sono tre tipi di file possibili: file di programma (già descritti), file di dati sequenziali e file ad accesso casuale.

File sequenziali

Come per i file sequenziali sul Datassette, quelli su dischetto devono essere aperti prima di poter essere usati. Per aprirne uno usare la struttura:

OPEN *lfn, dn, sa, "drn:nome del file,* SEQ,W"

dove:

lfn	è il numero logico del file
dn	è il numero di dispositivo dell'unità
sa	è l'indirizzo secondario. Si può impiegare qualsiasi numero tra 2 e 14. Sia 0 che 1 sono riservati dal C-64 per le operazioni SAVE e LOAD e 15 è usato per aprire un canale di comando.
drn	è il numero dell'unità; può essere omesso se ne usate una sola
nome del file	è il nome del file al quale si vuole accedere
SEQ	indica che questo è un file sequenziale
W	indica la scrittura (WRITE); per leggere il file, il comando READ può essere abbreviato con R.

Ecco il risultato del trasferimento ad un file chiamato MUSICA.

```
OPEN 2,8,4,"0:MUSICA,SEQ,W"
```

Quando si apre un file su dischetto, la luce rossa sull'unità s'illumina fino alla chiusura del file. Se si cerca di aprire un file sequenziale già esistente, la luce rossa si mette a lampeggiare, indicando una condizione d'errore. Volendo scrivere sopra ad un file preesistente, modificate il comando OPEN come segue:

```
OPEN 1,8,10,"@0:MUSICA,SEQ,W"
```

Il simbolo @ informa l'unità che si vuole scrivere sopra al file specificata. Se il file non esiste, l'unità esegue la normale procedura.

Uso di variabili a stringa come nomi di file

È possibile utilizzare una stringa al posto di un nome di file volendo creare il file da programma. Ecco un programma che richiede il nome del file prima che venga aperto, così da permettere di usare lo stesso programma per aprire i file, come per esempio, in un programma "word processor" che usa diversi nomi per i file di testo.

```
10 INPUT "NOME DEL FILE"; FN$
20 OPEN 2,8,4, "0:"+FN$+ ",W"
```

La riga 1Ø richiede un nome di file. A riga 2Ø il nome è concatenato al comando OPEN. Questo è molto importante perché il comando OPEN deve essere un'unica stringa. È necessario l'uso di (+) per quest'operazione.

Chiusura di file su dischetto

Quando un programma trasferisce dati ad un file, questi vengono prima passati a un buffer e poi, quando questo è pieno, passati al dischetto. Al termine del trasferimento, gli ultimi dati rimasti nel buffer non verranno trascritti a meno di non chiudere il file. La chiusura trascrive tutto ciò che rimane nel buffer ed è per questo che è molto importante.
Nota: Il C-64 permette di tenere aperti fino a dieci canali, ma solo 5 di questi all'unità a dischetti. È quindi consigliabile chiudere i canali subito dopo ogni operazione di lettura e scrittura, anche se mantenere aperto un canale di lettura non sarà certo la causa di errori catastrofici. Ben più grave è cercare di scrivere attraverso un canale ancora chiuso.

IL COMANDO PRINT#

Il comando PRINT# si usa per mandare dati ad un'unità periferica. Il C-64 termina automaticamente ogni record con un ritorno a capo. In alcuni casi, come ad esempio usando la stampante MPS 801, sarà utile mandare un ritorno a capo e un avanzamento di riga per terminare i record di un file. Usate i numeri logici di file da 1 a 127 per il solo ritorno a capo e da 128 a 255 per un ritorno a capo più un avanzamento di riga.

LETTURA DI UN FILE DI DATI

Le istruzioni INPUT# e GET# operano sostanzialmente allo stesso modo sia con i nastri che con i dischetti. INPUT# può caricare stringhe fino a 80 caratteri; per caricarne di più è necessario usare GET# e leggere le stringhe un byte per volta.

FILE AD ACCESSO CASUALE (RANDOM) O DIRETTO

Si può creare file ad accesso casuale accedendo direttamente ai blocchi di dati e ai buffer di memoria. Ogni blocco di dati occupa un singolo settore. Vi sono 8 buffer disponibili sul C-64, ma quattro di essi sono utilizzati per la BAM, il canale di comando I/O e il controllore del dischetto. Questo lascia solo 4 buffer disponibili per i file ad accesso diretto. Non aprite mai più di quattro buffer contemporaneamente: ciò provocherebbe un errore di sistema.
Le informazioni sono trasmesse ai file ad accesso casuale con il comando PRINT e sono specificate tramite parametri dell'istruzione OPEN con la seguente struttura:

OPEN *lfn, dn, sa, "buf"*

dove:

lfn è il numero logico del file. Per trascrizioni di dati usate i numeri da 2 a 14. Per eseguire un comando di servizio, usate il numero 15. È generalmente una buona norma aprire il canale di servizio (15) e uno di dati per ogni operazione

dn è il numero di dispositivo

sa è l'indirizzo secondario (deve avere un valore tra 2 e 14)

buf è il numero del buffer assegnato all'indirizzo secondario specificato. Non è necessario usare questa specificazione; omettendolo il DOS selezionerà automaticamente un buffer.

8.2. Comandi di servizio del dischetto

In questa sezione sono descritti i comandi di servizio del dischetto; la tabella 8.3 ne riporta un sommario.

BLOCK-READ

IL comando BLOCK-READ legge un settore (*block*) in uno dei buffer di memoria. Prima, però, è necessario aprire il canale di comando

```
10 OPEN 15,8,15
```

e anche un canale ad accesso diretto.

```
20 OPEN 2,8,4,"#"
```

Scegliete il blocco da leggere (traccia e settore).

```
30 INPUT "TRACCIA"; A
40 INPUT "SETTORE"; B
```

La seguente istruzione trasferisce un blocco di dati al buffer:

```
50 PRINT#15, "B-R:"4;0;A;B
```

Tabella 8.3. Comandi di servizio del disco

Comando	*Abbreviazione*	*Formato*
BLOCK-READ	B-R	PRINT#15, "B-R:"sa;dr;t;s
BLOCK-ALLOCATE	B-A	PRINT#15, "B-A:"dr;t;s
BLOCK-WRITE	B-W	PRINT#15, "B-W:"sa;dr;t;s
BLOCK-EXECUTE	B-E	PRINT#15, "B-E:"sa;dr;t;s
BUFFER-POINTER	B-P	PRINT#15, "B-P:"sa;byte
BLOCK-FREE	B-F	PRINT#15, "B-F:"dr;t;s
MEMORY-WRITE*	M-W	PRINT#15, "M-W"CHR$(byte basso) CHR$ (byte alto) CHR$(n° byte)CHR$(dati)CHR$ (dati)...
MEMORY-READ*	M-R	PRINT#15, "M-R"CHR$(byte basso) CHR$(byte alto)
MEMORY-EXECUTE*	M-E	PRINT#15, "M-E"CHR$(byte basso) CHR$(byte alto)
U1	UA	In sostituzione di BLOCK-READ
U2	UB	In sostituzione di BLOCK-WRITE
U3**	UC	Locazione puntata: $0500
U4**	UD	Locazione puntata: $0503
U5**	UE	Locazione puntata: $0506
U6**	UF	Locazione puntata: $0509
U7**	UG	Locazione puntata: $050C
U8**	UH	Locazione puntata: $050F
U9**	UI	Locazione puntata: $FFFA
U:	UJ	Salta alla routine di accensione

*Per queste istruzioni si devono usare le abbreviazioni.
**Questi comandi attivano le routine del microprocessore del drive del dischetto poste alle locazioni indicate.

Le componenti di quest'istruzione sono:

PRINT#15 per tutti i comandi in questa sezione occorre il canale di servizio (15)

"B-R:" è l'abbreviazione di "BLOCK-READ". I ":" servono per posizionare i dati che seguono l'istruzione

4 è l'indirizzo secondario (v. riga 2Ø)

0 è il numero dell'unità: obbligatorio con le istruzioni ad accesso diretto

A è il numero della traccia caricato con input a riga 3Ø. Si possono usare sia variabili che costanti per designare tracce e settori

B è il numero del settore.

Tutti i parametri di quest'istruzione devono essere separati da ";" come mostrato nell'esempio. Ora che tutti i dati presi dal blocco designato sono nel buffer, è necessario usare GET# o INPUT# per estrarli. L'istruzione INPUT# acquisisce tutti i byte fino ad includere il primo ritorno a capo che trova. Dato che non sempre si conosce prima l'esatto contenuto di un settore, è possibile superare la capacità dell'INPUT#, che è di 80 caratteri (byte).
Data questa limitazione è preferibile usare GET# quando si è incerti sui dati da leggere. GET# leggerà un byte alla volta.

```
60 GET#2, A$
```

Vi sono 256 byte di dati in un blocco, ma non tutti i blocchi sono pieni. Per leggere tutti i dati di un blocco e fermarsi alla fine del file, verificate che la variabile di stato (ST) sia 0.

```
70 IF ST=0 THEN PRINT A$;:GOTO 60
```

Nota: Benché vi sia un punto e virgola nell'istruzione PRINT, il video salta alla riga successiva quando trova un ritorno a capo nei dati, separandoli così proprio come sono stati caricati.
Se ST=0 vi sono ancora dati ed è necessario ritornare a prelevare un altro byte; altrimenti, va alla riga 8Ø e chiude tutti i canali.

```
80 CLOSE2: CLOSE15
```

Ed ecco l'intero programma di BLOCK-READ.

```
10 OPEN 15,8,15
20 OPEN 2,8,4, "#"
30 INPUT "TRACCIA"; A
40 INPUT "SETTORE"; B
50 PRINT#15, "B-R:"4;0;A;B
60 GET#2, A$
70 IF ST=0 THEN PRINT A$;: GOTO 60
80 CLOSE 2: CLOSE 15
```

BLOCK-ALLOCATE

La BAM tiene conto di tutti i record allocati (che contengono cioè dati). Nelle istruzioni ad alto livello (come SAVE) il DOS usa queste informazioni per determinare dove posizionare dati sul dischetto. Il DOS non fa uso della BAM con le funzioni ad accesso diretto e rende perciò possibile trascrivere qualsiasi dato in qualsiasi blocco sul dischetto, anche se contiene già dei dati. È possibile trasferire dati anche nella traccia riservata al catalogo, ma è meglio evitare di farlo perché si rischia di perdere il normale accesso a tutto ciò che è memorizzato sul dischetto. Si consiglia di eseguire un'istruzione BLOCK-ALLOCATE prima di trasferire un blocco. L'istruzione BLOCK-ALLOCATE controlla il settore per determinare se contiene già dei dati. Se è disponibile (come indicato nella BAM), essa segna il settore indicando che ora è assegnato; se già assegnato, lascia immutata la BAM e segnala il successivo settore libero.
Ecco una routine per eseguire un BLOCK-ALLOCATE.

```
10 OPEN 15,8,15
20 INPUT "TRACCIA"; A
30 INPUT "SETTORE"; B
40 PRINT#15, "B-A:"0;A;B
```

Le componenti dell'istruzione sono le seguenti:

PRINT#15, attiva il canale di comando
"B-A:" è l'istruzione BLOCK-ALLOCATE
A è il numero della traccia
B è il numero del settore

```
60 INPUT#15,E,EM$,T,S
70 PRINT E,EM$
80 PRINT T,S
90 CLOSE 15
99 END
```

Al termine di ogni operazione, lo stato del dischetto può essere verificato leggendo 4 variabili i cui valori sono accessibili al canale di servizio (15). Nel caso di un'istruzione BLOCK-ALLOCATE, il canale 15 vi dirà se la traccia e il settore scelti sono disponibili; qualora non lo fossero, vi darà i numeri della successiva traccia e settore disponibili. La riga 60 verifica ciò trascrivendo i valori che trova nelle variabili E (codice d'errore), EM$ (messaggio d'errore), T (traccia) e S (settore). Le righe 70 e 80 stampano i dati e le righe 90 e 99 chiudono il canale e terminano il programma.

Quando si usa questo programma come subroutine chiamata da un altro programma, è possibile utilizzare i dati così ottenuti per allocare un altro blocco se quello prescelto è già occupato; se invece è disponibile, viene presentato il messaggio "OK".

BLOCK-WRITE

Prima di usare un BLOCK-WRITE, è consigliabile eseguire un BLOCK-ALLOCATE in modo da determinare se il settore che si desidera è disponibile o, se non lo fosse, conoscere l'ubicazione del primo disponibile. Il seguente programma di BLOCK-WRITE si serve di BLOCK-ALLOCATE per controllare il settore selezionato:

```
10 OPEN 15,8,15
20 INPUT "TRACCIA"; A
30 INPUT "SETTORE"; B
40 PRINT#15, "B-A:"0;A;B
50 INPUT#15,E,EM$,T,S
```

Ora è necessario esaminare EM$ per stabilire se il settore prescelto è disponibile.

```
60 IF EM$ = "OK" THEN 100
```

Se non lo è, usate i valori ottenuti come nuovi parametri di traccia e settore.

```
70 A=T
80 B=S
```

Nota: Se non vi sono più settori disponibili sul dischetto, l'istruzione B-A riporta valori di 0 per la traccia e il settore. Questi non rappresentano un settore e si otterrebbe un errore cercando di accedervi. Per evitare questo problema, ricontrollate i risultati; se nelle variabili T e S vi sono degli zeri bisogna esaminare i settori con numeri inferiori.

```
90 IF A=0 AND B=0 THEN PRINT "DISCO PIENO":
GOTO 160
```

Nel caso non si presenti questa condizione, procedete con BLOCK-WRITE.

```
100 PRINT "TRACCIA";A;"";"SETTORE";B
105 OPEN 2,8,4,"#"
110 INPUT A$
```

Caricando una X, il programma termina l'input dei dati.

```
120 IF A$="X" THEN 150
```

Se no, trascrive i dati nel file

```
130 PRINT#2, A$
```

e torna alla 11Ø per acquisire altri dati.

```
140 GOTO 110
```

Impiegate la seguente istruzione per caricare i dati nel settore prescelto:

```
150 PRINT#15, "B-W:"4;0;A;B
```

Nota: La struttura di BLOCK-WRITE è uguale a quella di BLOCK-READ. Ancora una volta si devono chiudere i canali e terminare il programma.

```
160 CLOSE 2: CLOSE 15
```

BLOCK-EXECUTE

Questo comando carica un settore del dischetto contenente una routine in linguaggio macchina e la esegue dalla locazione 0 nel buffer fino a che non incontra un'istruzione RTS.
La struttura è:

```
PRINT#15,"B-E:"15;8;1;4
```

PUNTATORE DI BUFFER

Come si è visto, i buffer di dati contengono le informazioni lette sul dischetto. *Il puntatore di buffer* tiene conto di quale byte si sta leggendo e avanza di uno ogni volta che un byte viene letto.
Si supponga di voler leggere i dati in un file specifico di 240 byte in forma di record separati. Il primo record si trova nei byte da 1 a 120 e il secondo tra 121 e 240. Leggere i file in ordine sarebbe facile perché il puntatore di buffer si troverebbe già al punto in cui inizia il secondo file dopo aver letto il primo. Ma volendo leggere solo il secondo?
Una maniera è quella di eseguire 120 istruzioni GET#, per mezzo di un breve loop, fino ad arrivare al byte 121. Tuttavia vi è un metodo più facile: l'istruzione BUFFER-POINTER permette di indicare qualsiasi byte nel buffer. La sua struttura è:

PRINT#15, "B-P:"*sa;byte*

dove:

PRINT#15, attiva il canale di comando
"B-P.:" è il comando BUFFER-POINTER
sa; è l'indirizzo secondario
byte è il byte al quale si vuole accedere.

Per esempio, volendo eseguire un GET al 15° byte di un blocco, si dovrebbe usare

```
PRINT#15,"B-P:"4;15
```

BLOCK-FREE

L'istruzione BLOCK-FREE libera qualsiasi blocco sul dischetto. Quest'istruzione ordina alla BAM di marcare il blocco specificato come disponibile, permettendo così di trascrivere dati a quel blocco. Per eseguire un BLOCK-FREE, usate la seguente struttura:

OPEN 15,8,15
PRINT#15, "B-F:" *dr;t;s*

dove:

PRINT#15, attiva il canale di servizio
"B-F:" è il comando BLOCK-FREE
dr è il numero dell'unità
t è il numero della traccia
s è il numero del settore.

Ecco una routine utilizzabile per liberare qualsiasi blocco sul dischetto:

```
10 OPEN 15,8,15
20 INPUT "TRACCIA"; A
30 INPUT "SETTORE"; B
40 PRINT#15, "B-F:"0;A;B
50 CLOSE 15
99 END
```

8.3. Operazioni sulla memoria del dischetto

Il controller del drive 1541 interpreta anche comandi esterni e fa sì che l'unità li esegua. Il drive contiene un microprocessore 6502, simile a quello del C-64, dispone di 2K di RAM e del DOS, che è contenuto su due ROM. Parte di questa memoria è utilizzata per i buffer dei quali si è parlato nelle ultime sezioni; una parte per scopi di ordine interno, come la manutenzione della BAM e informazioni speciali di file; un'altra è dispo-

nibile per essere usata per applicazioni speciali. La RAM che è disponibile all'utente per scriverci delle routine è la stessa usata dal DOS per i buffer. Se si decide di scrivere uno speciale programma in linguaggio macchina in quelle aree, si deve tener conto di quali aree utilizzare e quali riservare ai buffer. Vi sono 5 *pagine* di memoria, ognuna delle quali contiene 256 byte.

Buffer	*Locazione di memoria (esadecimale)*
#1	300—3FF
#2	400—4FF
#3	500—5FF
#4	600—6FF
#5	700—7FF

Non è consigliabile usare il buffer #5, perché è spesso utilizzato dal DOS per varie attività di ordinamento e dati lí posizionati possono alterare le funzioni del DOS o essere cancellati. Lo spazio di memoria nei buffer da 1 a 4 è impiegato solo da essi, pertanto, se non si è richiesto un buffer e si è sicuri che non ne è stato aperto uno dal sistema, si può utilizzare liberamente questo spazio di memoria. Un metodo sarebbe di specificare quali buffer usare quando si apre un canale all'unità dischetti e caricare le routine nei buffer non utilizzati.
Le informazioni che seguono non sono indirizzate a programmatori principianti. L'uso di MEMORY-READ, MEMORY-WRITE e MEMORY-EXECUTE richiede una conoscenza approfondita della programmazione in linguaggio macchina e del DOS.

MEMORY-WRITE

Per caricare dati nella memoria dell'unità di gestione a dischetti è necessario usare il comando MEMORY-WRITE. Come POKE in BASIC, quest'istruzione carica il dato specificato nella locazione di memoria indicata. Si prenda ad esempio l'istruzione POKE:

```
POKE 768, 255
```

Con un'istruzione POKE viene trascritto solo un byte alla volta mentre il comando MEMORY-WRITE permette di trascriverne fino a 34 per volta. Per eseguire la stessa operazione dell'esempio sopracitato è necessario trasformare il numero decimale della locazione in un numero esadecimale (a base 16); 300 è l'equivalente esadecimale di 768 decimale. Dal momento che POKE trascrive solo un byte per volta, il BASIC sa sempre quanti byte aspettarsi (uno). Con MEMORY-WRITE è necessario indicare

quanti byte verranno trascritti. Nel caso specifico, è da trascrivere un solo byte, quindi il comando sarà breve. Vi sono dalle limitazioni speciali da osservare con queste istruzioni, dato che sono solo estensioni del codice macchina:

1. L'indirizzo di memoria deve essere caricato in due byte, prima il byte basso, poi quello alto.
2. Tutti i dati devono essere passati in forma di stringhe di caratteri (CHR$).
3. Il 6502 interpreta solo dati in binario. L'istruzione permette di caricare il numero in esadecimale; ma il BASIC non usa questa notazione. Per esempio,

   65536 decimale=FFFF esadecimale

   Questi sono due byte esadecimali (FF e FF). Per rappresentarli in questo "modo", bisogna convertirli ai loro valori decimali equivalenti, 255. Il numero 65536 sarebbe così memorizzato nella memoria del 1541 come CHR$(255)CHR$(255). Qualsiasi numero che può essere espresso in un solo byte (da 0 a 255) deve essere caricato così.
4. Si deve indicare quanti byte verranno trasferiti. Anche questo deve essere espresso in numeri esadecimali convertiti in notazione decimale, come sopra.
5. L'istruzione MEMORY-WRITE deve essere abbreviata in "M-W". Non è permessa nessuna virgola o altra punteggiatura.

Perciò l'istruzione per caricare 255 alla locazione 768 ($0300) è la seguente:

```
OPEN 15,8,15
PRINT#15,"M-W"CHR$(00)CHR$(03)CHR$(1)CHR$(255)
CLOSE 15
```

MEMORY-READ

È possibile leggere una locazione di memoria del 1541 per mezzo dell'istruzione MEMORY-READ. Questa permette di leggere un byte alla volta dalla memoria dell'unità di gestione dei dischetti, in maniera simile all'uso di PEEK per leggere la memoria centrale. L'istruzione BASIC per leggere la locazione 768 sarebbe:

```
PRINT PEEK(768)
```

La stessa istruzione, facendo uso di MEMORY-READ, si presenterebbe così:

```
40 OPEN 15,8,15
50 PRINT#15,"M-R"CHR$(00)CHR$(03)
60 GET#15, A$
70 PRINT A$
80 CLOSE 15
```

La sequenza è la seguente: si apre il canale di servizio, poi viene richiesto il comando MEMORY-READ. Sulla stessa riga (senza aggiunta di punteggiatura) è specificato il byte da leggere (prima quello più basso poi quello più alto). Questa è la locazione 300 (768 in decimale).
Nota: "M-R" è l'*unico* modo valido di chiamare quest'istruzione. Scriverla per esteso causerà un errore. I dati dopo la lettura possono essere trasferiti in memoria centrale tramite il canale 15. Per leggerli dal canale si impiega l'istruzione GET#15, A$. I valori sono così disponibili nella variabile A$, che è stata stampata sullo schermo. Infine, si chiude il canale. Dato che i dati letti dal 1541 sono trasmessi tramite il canale servizio, non si deve tentare di leggere una condizione d'errore dal canale fino a che questo è stato chiuso e riaperto. Se non si esegue questo passo si otterranno i dati trasmessi al posto del messaggio d'errore cercato.

MEMORY-EXECUTE

L'istruzione MEMORY-EXECUTE è usata per far eseguire un programma in linguaggio macchina caricato nella memoria del 1541. Il programma *deve* terminare con un'istruzione RTS in modo da ripristinare il controllo al C-64; in caso contrario è probabile che la memoria del 1541 entri, nel migliore dei casi, in un loop senza fine.
La struttura della istruzione MEMORY-EXECUTE (M-E) è essenzialmente la stessa di MEMORY-READ.

```
40 OPEN 15,8,15
50 PRINT#15,"M-E"CHR$(00)CHR$(03)
60 CLOSE 15
```

Questa ordina al DOS di iniziare una routine con inizio alla locazione 300 (768 decimale) della memoria del 1541.

8.4. Comandi utente

Il 1541 fa uso di un certo numero di comandi utente (non confondeteli con le routine USR), che fanno parte del BASIC. Questi comandi eseguono delle funzioni di comando simili alle istruzioni viste in precedenza.

U1

Il comando U1 è simile a BLOCK-READ ed ha identica struttura. B-R è semplicemente sostituito da U1. Il comando B-R legge solo i dati in un particolare blocco (settore); il comando U1 legge tutte le informazioni nel blocco, inclusi i due byte che precedono i 254 byte di dati. Questi byte contengono il legame (link) al blocco seguente. Questa concatenazione è rappresentata dal numero della traccia e del settore dove proseguire.

U2

Il comando U2 è simile a BLOCK-WRITE. La sua struttura è identica a quella di B-W. La differenza sta nel fatto che chiamare B-W chiude il file; ovvero il legame della traccia e del settore (i due byte che precedono i 254 byte di dati nel blocco) è fissato in modo da indicare che questo blocco è il termine del file. Problemi possono sorgere dal fatto che traccia e settore successivi non sono indicati. Per esempio se si volessero trascrivere dei dati in mezzo ad altri preesistenti, B-W chiuderebbe il file, mentre U2 non lo farebbe.

U3-U9 e U:

Questi comandi utente sono simili a MEMORY-EXECUTE. Essi saltano ad una specifica locazione di memoria ed iniziano ad eseguire istruzioni. Le locazioni alle quali saltano sono elencate nella tabella 8.3. La sintassi per U3 per esempio è:

```
OPEN 15,8,15
PRINT#15, "U3:"
CLOSE 15
```

Il comando U: (o UJ) fa saltare il DOS alla sua routine di attivazione (accensione). Gli indirizzi di memoria ai quali i comandi utente saltano sono solo di 3 byte, perché sono concepiti per contenere un'istruzione in codice macchina di salto al programma definito.

8.5. Il modem

Il modem permette al C-64 di comunicare con altri computer per mezzo di linee telefoniche e di ricevere informazioni non solo da altri C-64, ma anche da reti locali. Ad esempio, permette di usare il C-64 come terminale di un grosso computer collegato alla rete. Le reti telematiche danno accesso ad informazioni come data-base di vario tipo, o messaggi da altri utenti (electronic mail).

INSTALLAZIONE DEL MODEM

Per installare un modem, procedete come segue:

1. Spegnere il C-64.
2. Inserire il collegamento al modem nella presa per utenze parallele del C-64 (vedi fig. 1.2).
3. Accendere il C-64.
4. Caricare un programma di I/O.
5. Formare il numero telefonico appropriato per accedere alla rete desiderata o al computer che si vuole chiamare. Chiamando una rete, si dovrà introdurre con il C-64 il proprio codice di identificazione e di accesso. Nel caso di chiamata diretta ad un altro computer, il vostro modem dovrà essere posizionato su "origine" e quello dell'altro computer su "risposta".
6. Attendere un tono alto continuo e spostare il deviatore del modem da "fonia" a "dati".
7. Attenzione alla cornetta. Non apprenderla perché interromperebbe la comunicazione.

TERMINOLOGIA

Segue una lista di termini comunemente usati nelle telecomunicazioni:

Modem Il dispositivo periferico che converte i segnali emessi dal computer in informazioni che possono essere trasmesse dalle linee telefoniche. (Modem sta per "Modulazione/Demodulazione")

Handshaking Processo di trasmissione dati ed attesa di un segnale di risposta dal computer ricevente.

Full/half duplex Due modi di *handshaking*. Nel full duplex il computer ricevente ripete tutti i dati ricevuti al computer emittente. Se i dati ripetuti sono uguali a quelli emessi, il computer trasmette il byte suc-

cessivo; in caso contrario, i dati sono ritrasmessi. In half duplex un computer trasmette dei dati e quello ricevente risponde semplicemente che sono stati ricevuti.

Velocità di trasmissione (Baud) Indica la massima velocità con la quale è possibile trasmettere informazioni. Il modem trasmette e riceve dati da 300 a 19200 baud. La velocità è espressa anche in BPS, o bit per secondo. Un sistema che trasmette e riceve a 300 baud, trasmette e riceve a 300 bit al secondo.

Risposta/origine Quando ci si collega ad una rete si chiamerà il sistema. Dato che siete voi ad "originare" la chiamata, il modem deve essere posizionato su "origine". Questo inizia il protocollo "d'introduzione" elettronico. Quando si comunica direttamente con un altro computer, uno dei due deve essere in modo "origine" e l'altro nel modo "risposta". Una volta stabilita la comunicazione, non ha importanza quale sia in "origine" e quale in "risposta".

Parity Alcuni computer per controllare la correttezza delle informazioni trasmesse, mandano un bit extra insieme agli altri chiamato parity bit. Se il numero di bit accesi, incluso il parity bit, è pari, allora la trasmissione è a parity pari. Se manca questo bit, la trasmissione non ha alcuna parity. Per comunicare correttamente, entrambi i computer devono avere la stessa parity.

Lunghezza della parola È il numero di bit contenuto in ogni byte. La maggior parte dei computer usa una parola di 7 o 8 bit.

Bit di inizio e di stop Alcuni computer richiedono un certo numero di bit nulli trasmessi al termine di ogni byte. I bit nulli sono chiamati bit di stop. Se il sistema usa questi bit, lo schema più comune è un bit di stop ed uno di inizio. In questo caso il numero totale di bit per ogni byte aumenta a dieci. Ciò rallenta l'effettiva velocità di trasmissione, ma migliora la affidabilità.

Avanzamento di riga Al termine di una riga di dati, il computer normalmente trasmette un ritorno a capo. Alcuni sistemi richiedono un avanzamento di riga dopo ogni riga di dati o file di dati.

ASCII (American Standard Code for Information Interchange-Codice standard americano per lo scambio di informazioni). È il codice standard per comunicare: comprende i numeri da 0 a 9 e le lettere maiuscole e minuscole. Include inoltre un certo numero di simboli e caratteri speciali. Il C-64 usa una estensione alla serie di caratteri ASCII per la propria serie di caratteri speciali, ma deve limitare l'uso di questi alle comunicazioni con altri computer Commodore.

8.6. La stampante MPS 801

La stampante MPS 801 è fornita di una serie di caratteri incorporati simile a quella del C-64. Include lettere maiuscole e minuscole, numeri e simboli grafici. Si può inoltre usare la stampante MPS 801 per stampare interamente i caratteri grafici personali di quasi tutte le misure.

ISTRUZIONE OPEN

Per avere accesso alla stampante, è necessario aprire ad essa un canale. L'istruzione OPEN per la stampante ha la seguente struttura:

OPEN *fn, dn, sa*

dove:

- *fn* Il numero del file è il numero prescelto di accesso al file. Sono accettabili tutti i numeri da 0 a 255
- *dn* Il numero di dispositivo: per la stampante può essere 4 o 5. Può essere selezionato da un interruttore sul retro della stampante.
- *sa* Nella maggior parte dei casi non si userà un numero di indirizzo secondario con la stampante.

Una volta aperto un canale alla stampante, tutto quello che resta da fare per stampare è inserire i dati in un'istruzione PRINT#, tipo:

```
PRINT#1,"INSERITE I DATI"
```

La stampante stamperà:

```
INSERITE I DATI
```

ISTRUZIONE CLOSE

Quando si termina l'accesso ad un file, che sia alla stampante o a qualunque altro dispositivo, è sempre consigliabile chiuderlo. Per chiudere un file usate la seguente istruzione:

CLOSE *numero del file*

Avendo aperto un canale con il numero 1 con

```
OPEN 1,4
```

lo si chiuderà con

```
CLOSE 1
```

ISTRUZIONE CMD

Normalmente tutto ciò che il C-64 vuole comunicare all'esterno viene visualizzato sullo schermo. Quest'ultimo è chiamato il *dispositivo di output primario*. Tuttavia è possibile cambiare questa condizione così da mandare il tutto automaticamente ad un altro dispositivo invece che allo schermo. Questo procedimento si effettua con un'istruzione CMD, la cui struttura è la seguente:

CMD *numero del dispositivo*

Ordinate al C-64 di impiegare la stampante come dispositivo primario. Per prima cosa, aprite ad essa un canale con:

```
OPEN 1,4
```

Poi battete l'istruzione CMD come segue:

```
CMD 1
```

Ora tutto ciò che è normalmente presentato sul video verrà stampato dalla stampante, incluso il messaggio READY e i messaggi d'errore. I dati caricati continueranno ad essere presentati sul video. Provate ad eseguire alcune istruzioni PRINT.

Come uscire dal modo CMD

Vi sono tre metodi per uscire dal modo CMD.

1. Premere RUN/STOP e RESTORE contemporaneamente. Questo inizializza il sistema di numero riportando il computer alle condizioni presenti all'accensione.
2. Indirizzare l'istruzione CMD ad un altro dispositivo, come il video, dispositivo #3.

```
CMD 3
```

3. Caricare un PRINT# per il dispositivo primario. Per esempio, avendo assegnato come primario il dispositivo #1 con CMD 1, si può uscire con:

```
PRINT#1
```

Dei tre metodi, è da preferire l'ultimo perché, oltre ad uscire dal modo CMD, svuota il buffer della stampante. Vi potrebbero essere dei caratteri lasciati da un'istruzione PRINT incompleta (ad es. conclusa da un ";").

I SET DI CARATTERI DELLA STAMPANTE MPS 801

Battete sulla tastiera il vostro nome, premete RETURN e caricate dei cuori ("S" con SHIFT) sulla riga sotto. Premete ora i tasti del simbolo COMMODORE e SHIFT simultaneamente. Le lettere del vostro nome diverranno minuscole e i cuori delle "S" maiuscole. Questi sono due set di caratteri disponibili oltre che sul C-64 anche sulla stampante MPS 801 che possono essere selezionati a volontà. Vi sono due metodi di sceglierli.

1. Specificare un indirizzo secondario 7 quando si apre il canale della stampante.

```
OPEN 1,4,7
```

   Questo selezionerà il set di caratteri alternativo. Per accedere al set standard aprite il canale alla stampante con

```
OPEN 1,4
```

   omettendo l'indirizzo secondario.

2. Stampare il comando CHR$(17) per il set alternativo o CHR$(145) per quello standard, come mostra l'esempio:

```
10 OPEN 1,4
20 PRINT#1,CHR$(17);"MINUSCOLO"
30 PRINT#1,CHR$(145);"MAIUSCOLO"
40 CLOSE 1
```

FORMATTAZIONE DELLA STAMPA

Le istruzioni TAB e SPC, la virgola e il punto e virgola aiutano a posizionare dati sullo schermo; anche la stampante usa queste istruzioni ma non esattamente alla stessa maniera.

Virgola

La virgola fa incolonnare i dati in quattro zone dividendo il video in quattro colonne da 10 caratteri l'una. Provate questo esempio:

```
10 FOR T=0 TO 30
20 PRINT T,
30 NEXT
```

Notate che le colonne sono presentate ordinatamente anche se i numeri contenuti non sono di uguale lunghezza. Sulla stampante, una virgola metterà 10 spazi tra i dati caricati, ma le colonne non saranno necessariamente ordinate. Provate lo stesso programma sulla stampante.

```
5 OPEN 1,4
10 FOR T=0 TO 30
20 PRINT#1, T,
30 NEXT
40 CLOSE 1
```

Punto e virgola

Il punto e virgola funziona nello stesso modo sia sul video che sulla stampante. Si usa per separare le variabili senza inserirvi degli spazi. Si ricordi, però, che i numeri verranno comunque preceduti da uno spazio. Per evidenziare questa caratteristica caricate:

```
10 OPEN 1,4
20 A=21 : B=300 : AB=57
30 PRINT#1,A;B;AB
```

TAB E SPC

Le istruzioni TAB e SPC sono così simili che spesso vengono confuse. La funzione TAB definisce una posizione assoluta e SPC indica una posizione relativa.
La seguente istruzione TAB stamperà un asterisco nella colonna 10 del video:

```
PRINT TAB(10);"*"
```

Ora provate la seguente istruzione. Stamperà un asterisco nella colonna 10 ed un altro nella 11.

```
PRINT TAB(10);"*";TAB(10);"*"
```

L'istruzione SPC inizia dalla posizione attuale del cursore e si sposta del numero di spazi indicati. Si provi la seguente istruzione:

```
PRINT SPC(10);"*";SPC(10);"*"
```

Questa istruzione pur somigliando molto alla TAB precedente, stampa un asterisco nella colonna 10 e un altro nella 21; questo perché SPC comincia a contare dallo spazio immediatamente dopo il primo asterisco (colonna 11) e stampa il secondo asterisco dieci posizioni più in là.

USO DI POS SULLA STAMPANTE

Usati in istruzioni PRINT#, sia TAB che SPC funzionano come SPC. Si noti, tuttavia, che TAB e SPC non possono comparire direttamente dopo PRINT# [per esempio, PRINT#1,TAB(20)]. Per osservare queste due funzioni, aprite un canale alla stampante.

```
OPEN 1,4
```

Ora caricate la seguente riga:

```
1 PRINT#1,"";SPC(10);"*";SPC(10);"*"
```

Questo stamperà un asterisco nella colonna 10 ed un altro nella 21. Sostituendo SPC con TAB, si otterrà esattamente lo stesso risultato. Provate:

```
PRINT#1,"";TAB(10);"*";TAB(10);"*"
```

Per ottenere la funzione TAB sulla stampante occorre usare l'istruzione POS, che è trasmessa alla stampante come CHR$(16). Ora si carichi:

```
PRINT#1,CHR$(16)"10*";
PRINT#1,CHR$(16)"21*"
```

Quando la stampante incontra questo comando usa i due caratteri immediatamente seguenti per indicare dove iniziare a stampare sulla riga. Si può anche impiegare il codice dei caratteri numerici per indicare la posizione.

```
PRINT#1,CHR$(16)CHR$(49)CHR$(48)"*"
```

GRAFICA DELLA STAMPANTE

La stampante possiede diversi modi di stampa, che sono elencati nella tabella 8.4. I caratteri che riceve verranno trattati differentemente a seconda del modo attivo in quel momento. Usando i vari modi si può stampare qualsiasi cosa, dai caratteri normali alla grafica.

Caratteri a doppia larghezza

Per stampare caratteri a doppia larghezza sulla stampante, usate il comando CHR$(14) prima della stringa da stampare.

```
PRINT#1,CHR$(14);"CARATTERE ELONGATO"
```

Poi tornate alla larghezza normale inserendo CHR$(15).

```
PRINT#1,CHR$(14);"ELONGATO";CHR$(15);"COMPRESSO"
```

Caratteri inversi

Caratteri inversi possono essere stampati per mezzo del comando CHR$(18), nel modo seguente

```
PRINT#1,CHR$(18);"CARATTERI INVERSI"
```

Tabella 8.4. Modi di stampa

Modo	*Comando di stampa*
Stampa caratteri a doppia larghezza	CHR$(14)
Stampa caratteri a larghezza standard	CHR$(15)
Stampa caratteri inversi	CHR$(18)
Stampa caratteri non-inversi	CHR$(146)
Attiva il modo grafico	CHR$(8)
Attiva il set di caratteri alternativo	CHR$(17)
Ritorna al set di caratteri standard	CHR$(145)
Ripete un carattere grafico	CHR$(26)

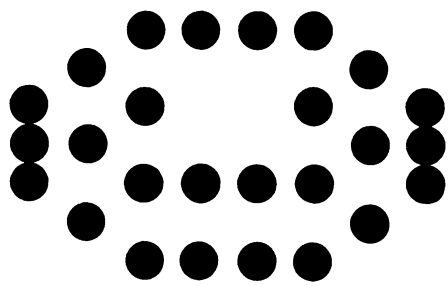
Notate che il comando CHR$(18) è lo stesso che stampa i caratteri inversi sul video. Sul video è possibile battere CTRL RVS ON per passare ai caratteri inversi. Ciò funziona anche sulla stampante mettendo l'istruzione in un comando PRINT come segue:

```
PRINT#1,"<CTRL><RVS ON>";"CARATTERI INVERSI"
```

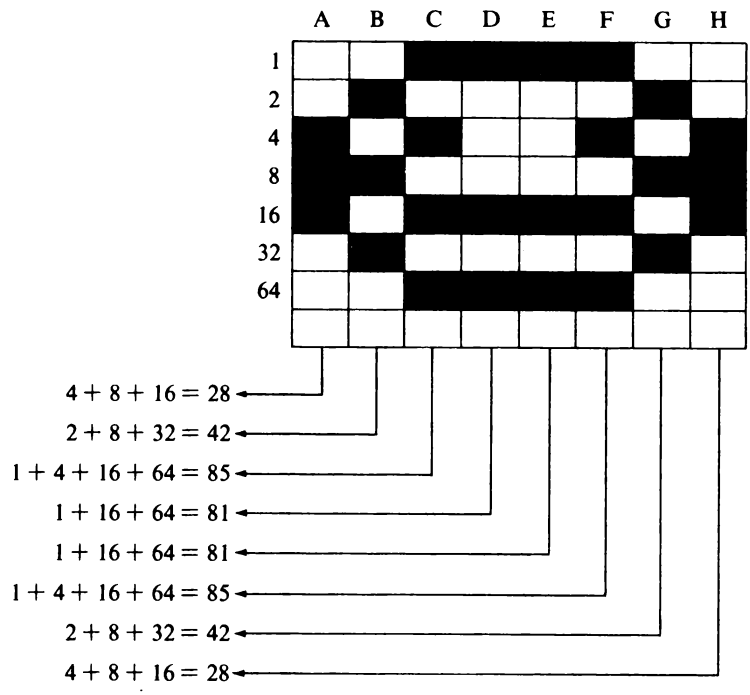
È possibile utilizzare sia CHR$(146) che CTRL RVS OFF per tornare ai caratteri normali.

Modo grafico

Il comando CHR$(8) ordina alla stampante di attivare il modo grafico. La grafica è creata stampando una serie di punti; per esempio, una faccia sorridente potrebbe essere composta da una serie di punti nel seguente schema:



Questo schema viene trasformato in una serie di numeri ed indirizzato alla stampante nel modo seguente: ad ogni riga di punti viene assegnato un valore numerico. La prima riga è 1, quella dopo è 2, e così via, con ogni riga successiva avente un valore doppio rispetto alla precedente. Il valore della settima riga è 64. Sommate i numeri delle posizioni in cui si vogliono stampare dei punti. A questo numero si aggiunga 128 e il risultato sarà il numero che trasmette alla stampante. Si esamini la faccia sorridente per determinare i valori da trasmettere alla stampante.



Nelle colonne i quadrati neri rappresentano le posizioni in cui la stampante deve stampare i punti. Sommando i valori nella colonna A si ottiene: 4+8+16=28. La colonna B è 2+8+32=42.
Continuando per tutte le colonne si ottengono i valori: 28, 42, 85, 81, 81, 85, 42 e 28. Per stampare questi punti come carattere grafico, inserite i valori in istruzioni CHR$, come segue:

```
10 DATA 28,42,85,81,81,85,42,28
20 OPEN 1,4
30 PRINT#1, CHR$(8);
40 FOR R=1 TO 8
50 READ A
60 PRINT#1,CHR$(A+128);
70 NEXT
80 PRINT#1
```

Questo stamperà una faccia sorridente. Se ne possono stampare di più ripetendo lo schema. Si possono anche usare i tasti delle funzioni per produrre caratteri speciali. Ecco un programma che stampa il testo caricato più una faccia sorridente. Quest'ultima verrà stampata ogni volta che si preme il tasto F1 (solo sulla stampante — sul video appare un asterisco).

```
10 OPEN 1,4
20 GET A$: IF A$="" THEN 20
30 IF A$ ="■" THEN 70
40 PRINT#1, A$;
50 PRINT A$;
60 GOTO 20
70 DATA 28,42,85,81,81,85,42,28
80 PRINT#1,CHR$(8);
90 FOR R=1 TO 8
100 READ A
110 PRINT#1, CHR$(A+128);
120 NEXT:PRINT "*"
130 PRINT#1, CHR$(15)
140 RESTORE: GOTO 20
```

Funzione di ripetizione grafica

La funzione di ripetizione grafica permette di stampare più volte qualsiasi schema di sette punti verticali (fino a 255 volte per comando). Ecco un esempio di funzione di ripetizione.

```
OPEN 1,4
PRINT#1, CHR$(26)CHR$(10)CHR$(255)
```

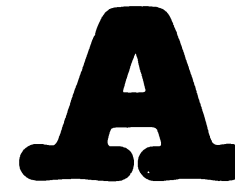
Questo stamperà una barra orizzontale piena alta sette punti e lunga dieci. La funzione di ripetizione è CHR$(26), CHR$(10) è il numero di volte da ripetere e CHR$(255) produce lo schema di punti contenente sette punti verticali. Questa funzione può facilmente essere incorporata nel programma della faccia sorridente ed utilizzata per ingrandirla.

```
10 DATA 28,42,85,81,81,85,42,28
20 OPEN 1,4
30 INPUT "LARGHEZZA";L
40 PRINT#1,CHR$(8);
50 FOR R=1 TO 8
60 READ A
70 PRINT#1,CHR$(26) CHR$(L) CHR$(A+128);
80 NEXT
90 PRINT#1
```

Inserite valori diversi nella variabile della larghezza e notate come cambia l'immagine. Naturalmente ad un certo punto, diventa talmente larga da non essere più riconoscibile.

Appendice

Architettura del sistema



La figura A.1 mostra un diagramma a blocchi del C-64 che illustra la connessione tra gli elementi del sistema descritti nel capitolo 1. Le frecce indicano il flusso dei dati tra questi elementi.

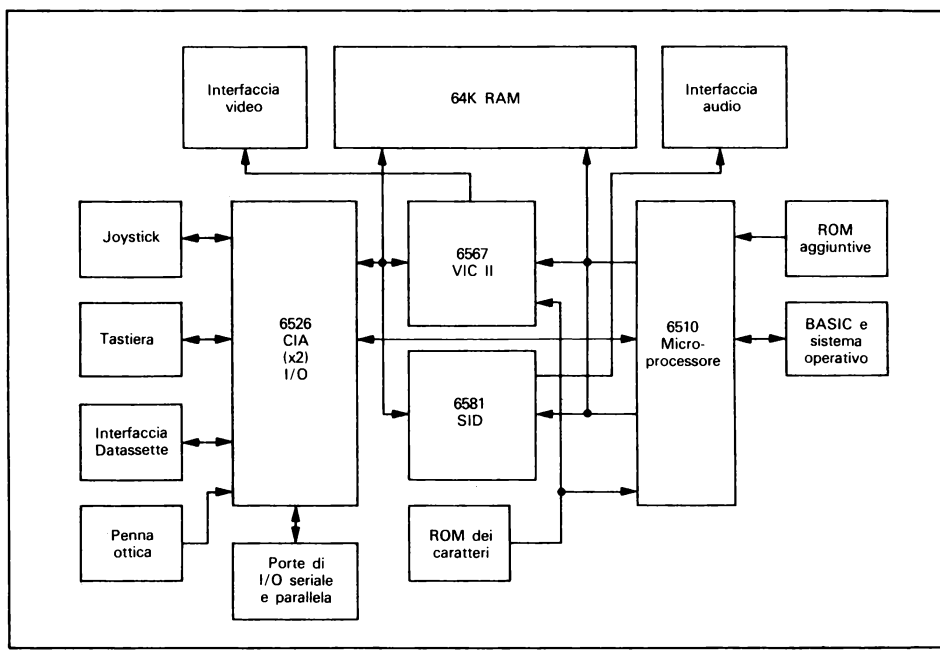
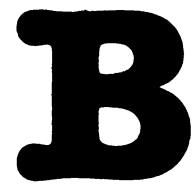


Figura A.1. Diagramma a blocchi del sistema

Appendice

Utilizzo della memoria



La funzione principale di una mappa della memoria è quella di essere d'aiuto nell'utilizzo di molte delle funzioni incorporate del computer. La seguente guida alla memoria non elenca la funzione di ogni singola locazione del C-64, ma solo le più utili; include inoltre una dettagliata spiegazione della funzione di ognuna di esse.
Molte locazioni incluse in questo elenco, invece di eseguire la funzione indicata, indicano la posizione della funzione elencata: ciò permette di spostare alcune funzioni in diverse zone della memoria. Per esempio, la locazione 648 contiene l'indirizzo della memoria del video. Cambiando questo numero, si possono usare altre parti della memoria per il video; è così possibile avere contemporaneamente differenti presentazioni e passare da una all'altra. Inoltre, alcune di queste locazioni tengono conto di funzioni interne del computer, come la 144 che segue lo stato del dispositivo di I/O inserito al momento nel sistema. Leggendo questa locazione, si può identificare una condizione OK o di errore. Le locazioni che sono usate come indirizzi indiretti contengono due numeri che rappresentano un singolo numero esadecimale. Per trasformarlo in valore decimale (per i PEEK e i POKE) impiegate la seguente formula:

valore decimale = byte nella locazione inferiore+
+(byte nella locazione superiore×256)

Ecco un breve programma che esegue questa funzione:

```
10 INPUT "VALORE NELLA LOCAZIONE
DI MEMORIA INFERIORE";L
20 INPUT "VALORE NELLA LOCAZIONE
```

```
DI MEMORIA SUPERIORE";H
30 PRINT "VALORE DECIMALE =";L+256*H
```

Locazione di memoria	*Funzione*
43 & 44	Contengono l'indirizzo del primo byte del vostro programma BASIC.
45 & 46	Contengono l'indirizzo del primo byte delle variabili del vostro programma.
47 & 48	Contengono l'indirizzo dell'inizio degli array BASIC.
49 & 50	Il numero contenuto indica la fine degli array BASIC aumentata di uno.
51 & 52	Il numero contenuto indica la locazione più bassa usata per memorizzare le stringhe del vostro programma BASIC.
53 & 54	Indicano l'ultima stringa memorizzata.
55 & 56	Indicano la locazione più alta disponibile per il BASIC.
144	Il byte di stato I/O. Indica la condizione del dispositivo I/O in uso. Uno 0 in questa locazione segnala una condizione OK. La tabella 8.2 contiene i codici d'errore.
160-162	L'orologio jiffy. Il numero in queste tre locazioni è un semplice numero binario. Per interpretarlo, usate il seguente programma:

```
10 A=PEEK(160)
20 B=PEEK(161)
30 C=PEEK(162)
40 D=C+((256)*B)+(65536*A)
50 PRINT D
```

192	Questo è l'interruttore del motorino del Datassette. Normalmente usato per accendere e spegnere questo motore, può essere usato per controllare altri dispositivi a bassa tensione. Vedere la figura C.6 per l'ubicazione della linea del motore sul connettore.
197	Il valore del tasto attualmente premuto può essere letto in questa locazione.
198	Contiene il numero di caratteri che al momento risiedono nel buffer della tastiera (un massimo di dieci).

631-640	Il buffer della tastiera, gestito in modo FIFO.
641-642	Indicano la locazione più bassa del sistema operativo.
643-644	Indicano la locazione più alta del sistema operativo.
648	Questa locazione controlla la posizione della memoria del video. La posizione effettiva può essere determinata moltiplicando il numero qui trovato per 256.
649	Questa locazione controlla la dimensione del buffer della tastiera. Cambiando questo valore (fissato a dieci ogni volta che il sistema è inizializzato) si può controllare il numero di tasti memorizzati mentre il computer è in funzione.
650	Funzione di ripetizione dei tasti. ON=128, OFF=0.
651	Controlla la velocità con cui opera la funzione di ripetizione.
652	Ritardo della ripetizione. Determina quanto tempo un tasto deve rimanere premuto prima di cominciare a ripetere.
657	Funzione che attiva/disattiva lo SHIFT. Controlla l'accesso al set alternativo di caratteri dalla tastiera ottenuto tramite COMMODORE/SHIFT.
770-771	Ripartenza a "caldo". Un SYS a questa locazione dà lo stesso risultato che premere RUN/STOP-RESTORE.
1024-2023	Locazione della memoria del video all'accensione.
2040-2047	Controllano la posizione degli sprite in memoria.
2048-40959	RAM disponibile all'utente.
32768-40959	Queste locazioni di memoria possono essere impegnate da cartucce esterne; in questo caso dati e programmi contenuti in quest'area potranno essere persi.
40960-49151	Interprete BASIC.
49152-53247	Buffer di 4K. Questa RAM è disponibile all'utente. Porre dei dati in quest'area non disturberà la RAM del BASIC nelle 2048-40959.

53248-54271	**Registri di controllo VIC-II**
53248	Sprite #0: posizione X
53249	Sprite #0: posizione Y
53250	Sprite #1: posizione X
53251	Sprite #1: posizione Y

53252 Sprite #2: posizione X
53253 Sprite #2: posizione Y
53254 Sprite #3: posizione X
53255 Sprite #3: posizione Y
53256 Sprite #4: posizione X
53257 Sprite #4: posizione Y
53258 Sprite #5: posizione X
53259 Sprite #5: posizione Y
53260 Sprite #6: posizione X
53261 Sprite #6: posizione Y
53262 Sprite #7: posizione X
53263 Sprite #7: posizione Y
53264 Sprite #0-7: nono bit di posizione X
53265 Registro di controllo VIC-II
Bit 0-2: Scorrimento verticale rallentato
Bit 3: Seleziona 24/25 righe di testo (24=0)
Bit 4: Pulisce il video (Blank=0)
Bit 5: Attiva il modo ad alta risoluzione (1=ON)
Bit 6: Extended Color Mode (1=ON)
Bit 7: Valore raster (nono bit)
53266 Valore raster (Bit 0-7)
53267 Penna ottica (Valore X)
53268 Penna ottica (Valore Y)
53269 Registro di controllo visualizzazione sprite
Bit 0: Sprite #0 (1=ON)
Bit 1: Sprite #1 (1=ON)
Bit 2: Sprite #2 (1=ON)
Bit 3: Sprite #3 (1=ON)
Bit 4: Sprite #4 (1=ON)
Bit 5: Sprite #5 (1=ON)
Bit 6: Sprite #6 (1=ON)
Bit 7: Sprite #7 (1=ON)
53270 Registro di controllo VIC-II #2
Bit 0-2: Scorrimento orizzontale rallentato
Bit 3: Seleziona 38/40 colonne (1=40)
Bit 4: Modo multicolore (1=ON)
Bit 5-7: non usati
53271 Registro espansione sprite (verticale)
Bit 0: Sprite #0 (1=2*Y)
Bit 1: Sprite #1 (1=2*Y)
Bit 2: Sprite #2 (1=2*Y)
Bit 3: Sprite #3 (1=2*Y)

Bit 4: Sprite #4 (1=2*Y)
Bit 5: Sprite #5 (1=2*Y)
Bit 6: Sprite #6 (1=2*Y)
Bit 7: Sprite #7 (1=2*Y)

53272 VIC-II Registro di controllo indirizzo
Bit 0: Non usato
Bit 1-3: Locazione set di caratteri
Bit 4-7: Locazione schermo

53273 Registro di controllo interruzioni VIC-II
Bit 0: Confronto raster
Bit 1: Collisione sprite-sfondo
Bit 2: Collisione sprite-sprite
Bit 3: Penna ottica
Bit 4-6: Non usati

53274 Registro di attivazione delle interruzioni VIC-II
Bit 0: Confronto raster (1=ON)
Bit 1: Collisione sprite-sfondo (1=ON)
Bit 2: Collisione sprite-sprite (1=ON)
Bit 3: Penna ottica (1=ON)
Bit 4-6: Non usati (1=ON)

53275 Registro di priorità sprite-sfondo
Bit 0: Sprite #0 (1=Sfondo)
Bit 1: Sprite #1 (1=Sfondo)
Bit 2: Sprite #2 (1=Sfondo)
Bit 3: Sprite #3 (1=Sfondo)
Bit 4: Sprite #4 (1=Sfondo)
Bit 5: Sprite #5 (1=Sfondo)
Bit 6: Sprite #6 (1=Sfondo)
Bit 7: Sprite #7 (1=Sfondo)

53276 Registro di controllo sprite multicolori
Bit 0: Sprite #0 (1=Multicolore)
Bit 1: Sprite #1 (1=Multicolore)
Bit 2: Sprite #2 (1=Multicolore)
Bit 3: Sprite #3 (1=Multicolore)
Bit 4: Sprite #4 (1=Multicolore)
Bit 5: Sprite #5 (1=Multicolore)
Bit 6: Sprite #6 (1=Multicolore)
Bit 7: Sprite #7 (1=Multicolore)

53277 Registro espansione sprite (orizzontale)
Bit 0: Sprite #0 (1=2*X)
Bit 1: Sprite #1 (1=2*X)
Bit 2: Sprite #2 (1=2*X)

Bit 3: Sprite #3 (1=2*X)
Bit 4: Sprite #4 (1=2*X)
Bit 5: Sprite #5 (1=2*X)
Bit 6: Sprite #6 (1=2*X)
Bit 7: Sprite #7 (1=2*X)

53278 Registro collisione sprite-sprite
Bit 0: Sprite #0 (1=collisione)
Bit 1: Sprite #1 (1=collisione)
Bit 2: Sprite #2 (1=collisione)
Bit 3: Sprite #3 (1=collisione)
Bit 4: Sprite #4 (1=collisione)
Bit 5: Sprite #5 (1=collisione)
Bit 6: Sprite #6 (1=collisione)
Bit 7: Sprite #7 (1=collisione)

53279 Registro collisione sprite-sfondo
Bit 0: Sprite #0 (1=collisione)
Bit 1: Sprite #1 (1=collisione)
Bit 2: Sprite #2 (1=collisione)
Bit 3: Sprite #3 (1=collisione)
Bit 4: Sprite #4 (1=collisione)
Bit 5: Sprite #5 (1=collisione)
Bit 6: Sprite #6 (1=collisione)
Bit 7: Sprite #7 (1=collisione)

53280 Colore cornice
0 = NERO
1 = BIANCO
2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO
11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

53281 Colore sfondo #0
0 = NERO
1 = BIANCO

2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO
11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

53282 Colore sfondo #1
0 = NERO
1 = BIANCO
2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO
11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

53283 Colore sfondo #2
0 = NERO
1 = BIANCO
2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO

11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

53284 Colore sfondo #3
0 = NERO
1 = BIANCO
2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO
11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

53285 Registro sprite multicolori #0

53286 Registro sprite multicolori #1

53287-53294 Registro colore sprite #0-7
0 = NERO
1 = BIANCO
2 = ROSSO
3 = CYAN
4 = VIOLA
5 = VERDE
6 = BLU
7 = GIALLO
8 = ARANCIO
9 = MARRONE
10 = ROSSO CHIARO
11 = GRIGIO SCURO
12 = GRIGIO MEDIO
13 = VERDE CHIARO
14 = BLU CHIARO
15 = GRIGIO CHIARO

54272-55295	**Registri di controllo SID**
54272-54278	Registri voce #1
54272	Controllo frequenza (byte basso)
54273	Controllo frequenza (byte alto)
54274	Durata dell'impulso (byte basso)
54275	Durata dell'impulso (byte alto)
54276	Registro di controllo forma d'onda Bit 0: Suono ON/OFF (1=ON) Bit 1: Bit di sincronizzazione (1=ON) Bit 2: Modulazione anulare (1=ON) Bit 3: Bit di test (normalmente non usato) Bit 4: Onda triangolare (1=ON) Bit 5: Onda a dente di sega (1=ON) Bit 6: Onda quadra (1=ON) Bit 7: Rumore bianco (1=ON)
54277	Registro di controllo attack/decay Bit 0-3: Valore di decay Bit 4-7: Valore di attack
54278	Registro di controllo sustain/release Bit 0-3: Valore di release Bit 4-7: Valore di sustain
54279-54285	Registri voce #2
54279	Controllo frequenza (byte basso)
54280	Controllo frequenza (byte alto)
54281	Durata dell'impulso (byte basso)
54282	Durata dell'impulso (byte alto)
54283	Registro di controllo forma d'onda Bit 0: Suono ON/OFF (1=ON) Bit 1: Bit di sincronizzazione (1=ON) Bit 2: Modulazione anulare (1=ON) Bit 3: Bit di test (normalmente non usato) Bit 4: Onda triangolare (1=ON) Bit 5: Onda a dente di sega (1=ON) Bit 6: Onda quadra (1=ON) Bit 7: Rumore bianco (1=ON)
54284	Registro di controllo attack/decay Bit 0-3: Valore di decay Bit 4-7: Valore di attack
54285	Registro di controllo sustain/release Bit 0-3: Valore di release

Bit 4-7: Valore di sustain

54286-54292 Registri voce #3

54286 Controllo frequenza (byte basso)

54287 Controllo frequenza (byte alto)

54288 Durata dell'impulso (byte basso)

54289 Durata dell'impulso (byte alto)

54290 Registro di controllo forma d'onda
Bit 0: Suono ON/OFF (1=ON)
Bit 1: Bit di sincronizzazione (1=ON)
Bit 2: Modulazione anulare (1=ON)
Bit 3: Bit di test (normalmente non usato)
Bit 4: Onda triangolare (1=ON)
Bit 5: Onda a dente di sega (1=ON)
Bit 6: Onda quadra (1=ON)
Bit 7: Rumore bianco (1=ON)

54291 Registro di controllo attack/decay
Bit 0-3: Valore di decay
Bit 4-7: Valore di attack

54292 Registro di controllo sustain/release
Bit 0-3: Valore di release
Bit 4-7: Valore di sustain

54293-54296 Funzioni di filtraggio del suono

54293 Valore del filtro cutoff (byte basso)

54294 Valore del filtro cutoff (byte alto)

54295 Registro di controllo filtro/risonanza
Bit 0: Filtro voce #1 (1=ON)
Bit 1: Filtro voce #2 (1=ON)
Bit 2: Filtro voce #3 (1=ON)
Bit 3: Filtro esterno (1=ON)
Bit 4-7: Valore di risonanza

54296 Controllo volume
Bit 0-3: Registro di controllo volume
Bit 4: Filtro passa-basso volume
Bit 5: Filtro passa-banda
Bit 6: Filtro passa-alto
Bit 7: Voce #3 ON/OFF (1=OFF)

54297 Valore X del paddle

54298 Valore Y del paddle

54299 Oscillatore #3. Generatore di numeri casuali

54300 Registro inviluppo

56320-56335	**Registri di controllo CIA #1**
56320	Porta A Bit 0-7: Valori delle colonne della tastiera Bit 0-3: Direzione (Joystick A) Bit 2&3: Pulsante di fuoco del paddle Bit 4: Pulsante di fuoco (Joystick A) Bit 6&7: Paddle, porta A
56321	Porta B Bit 0-7: Valori delle righe della tastiera Bit 0-3: Direzione (Joystick B) Bit 2&3: Pulsante di fuoco del paddle Bit 4: Pulsante di fuoco (Joystick B) Bit 6&7: Paddle, porta B
56322	Porta A Registro di direzione Input/Output
56323	Porta B Registro di direzione Input/Output
56324-56327	Timer
56324	Timer A (Byte basso)
56325	Timer A (Byte alto)
56326	Timer B (Byte basso)
56327	Timer B (Byte alto)
56328-56331	Orologio
56328	Decimi di secondo
56329	Secondi
56330	Minuti
56331	Bit 0-6: Ore Bit 7: Indicatore AM/PM
56332	Buffer I/O seriale (sincrono)
56333	Registro di controllo interruzioni Bit 0: Timer A Bit 1: Timer B Bit 2: Suoneria Bit 3: Porta seriale Bit 4: Lettura cassetta/input bus seriale Bit 7: Bit di rilevamento dell'interruzione
56334	Registro di controllo A Bit 0: Start/Stop Timer (1=Start) Bit 1: Uscita timer (1=ON) Bit 2: Modo Timer (1=Bistabile 0=a impulsi)

Bit 3: Modo funzionamento Timer (1=a scatti; 0=Continuo)
Bit 4: Preset Timer (1=Preset)
Bit 5: Contatore Timer
Bit 6: Porta I/O seriale (1=Output)
Bit 7: Frequenza orologio (0=60 Hz; 1=50 Hz)

56335 Registro di controllo B
Bit 0: Start/Stop Timer (1=Start)
Bit 1: Uscita Timer (1=ON)
Bit 2: Modo Timer (1=Bistabile 0=a impulsi)
Bit 3: Modo funzionamento Timer (1=a scatti 0=Continuo)
Bit 4: Preset Timer (1=Preset)
Bit 5&6: Selezione modo Timer B
0=Clock di sistema
1=Clock impulsi positivi
2=Riporto Timer A
3=Riporto Timer A: impulsi positivi
Bit 7: Imposta allarme (1=Set)
2=Riporto Timer A
3=Riporto Timer A: impulsi positivi
Bit 7: Imposta allarme (1=Set)

56576-56591 Registri di controllo CIA #2

56576 Porta Dati A
Bit 0&1: Seleziona segmento di memoria VIC-II
0=Segmento 3 (49152-65535)
1=Segmento 2 (32768-49151)
2=Segmento 1 (16384-32767)
3=Segmento 0 (00000-16383)
Bit 2: Uscita RS-232
Bit 3: Uscita ATN
Bit 4: Uscita clock bus seriale
Bit 5: Uscita bus seriale
Bit 6: Ingresso clock bus seriale
Bit 7: Ingresso bus seriale

56577 Porta dati B (RS-232)
Bit 0: Dati ricevuti
Bit 1: Richiesta dati
Bit 2: Terminale pronto
Bit 3: Indicatore
Bit 4: Ricevuto segnale di linea

Bit 5: Utente
Bit 6: Azzera per invio
Bit 7: Dati disponibili

56578 Registro di direzione I/O Porta A

56579 Registro di direzione I/O Porta B

56580 Timer A (Byte basso)

56581 Timer A (Byte alto)

56582 Timer B (Byte basso)

56583 Timer B (Byte alto)

56584-56587 Orologio

56584 Decimi di secondo

56585 Secondi

56586 Minuti

56587 Bit 0-6: Ore
Bit 7: Indicatore AM/PM

56588 Buffer I/O seriale (sincrono)

56589 Registro di controllo interruzioni
Bit 0: Timer A
Bit 1: Timer B
Bit 2: Suoneria
Bit 3: Porta seriale
Bit 4: Lettura cassetta/input bus seriale
Bit 7: Bit di rilevamento dell'interruzione

56590 Registro di controllo A
Bit 0: Start/Stop Timer (1=Start)
Bit 1: Uscita Timer (1=ON)
Bit 2: Modo timer (1=Bistabile 0=a impulsi)
Bit 3: Modo funzionamento Timer (1=a scatti 0=Continuo)
Bit 4: Preset Timer (1=Preset)
Bit 5: Contatore Timer
Bit 6: Porta I/O seriale (1=Output)
Bit 7: Frequenza orologio (0=60 Hz; 1=50 Hz)

56591 Registro di controllo B
Bit 0 Start/Stop Timer (1=Start)
Bit 1 Uscita Timer (1=ON)
Bit 2 Modo Timer (1=Bistabile 0=a impulsi)
Bit 3 Modo funzionamento Timer (1=a scatti 0=Continuo)

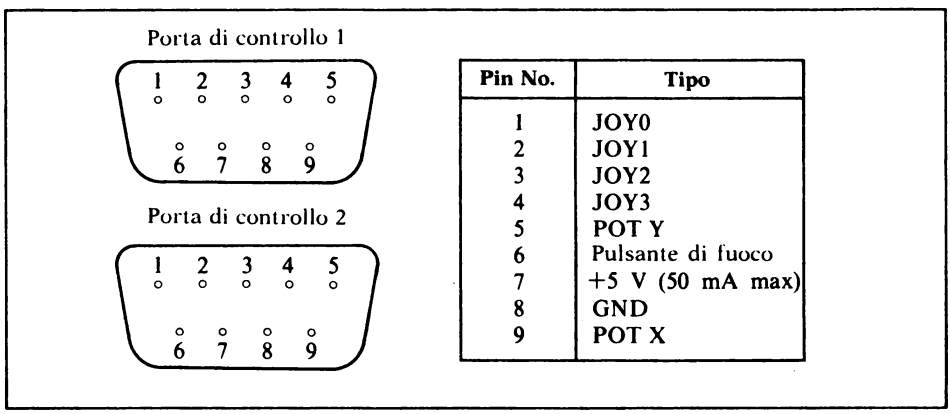
Bit 4: Preset Timer (1=Preset)
Bit 5&6: Selezione modo Timer B
0=Clock di sistema
1=Clock impulsi positivi
2=Riporto Timer A
3=Riporto Timer A: impulsi positivi
Bit 7: Imposta allarme (1=Set)

Appendice

Porte I/O del C-64

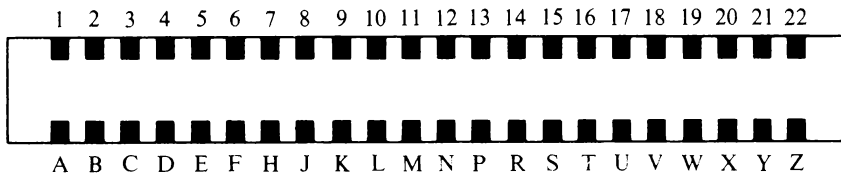
C

Questa sezione contiene gli schemi di tutte le porte per i collegamenti di I/O del C-64. Facendo uso di queste informazioni si possono progettare interfacce particolari per dispositivi che non si collegano direttamente al C-64.



Pin No.	Tipo
1	JOY0
2	JOY1
3	JOY2
4	JOY3
5	POT Y
6	Pulsante di fuoco
7	+5 V (50 mA max)
8	GND
9	POT X

Figura C.1. Pin della porta per i joystick



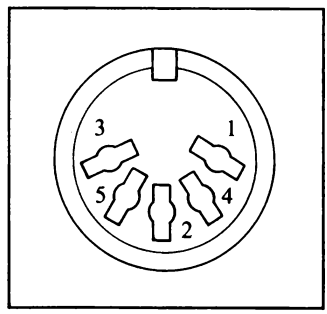
Pin No.	Tipo
1	GND
2	+5v
3	+5v
4	$\overline{\text{IRQ}}$
5	R/$\overline{\text{W}}$
6	DOT CLOCK
7	I/O 1
8	$\overline{\text{GAME}}$
9	$\overline{\text{EX ROM}}$
10	I/O 2
11	$\overline{\text{ROM L}}$

Pin No.	Tipo
12	$\overline{\text{BA}}$
13	$\overline{\text{DMA}}$
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

Pin No.	Tipo
A	GND
B	$\overline{\text{ROM H}}$
C	$\overline{\text{RESET}}$
D	$\overline{\text{NMI}}$
E	S 02
F	A 15
H	A 14
J	A 13
K	A 12
L	A 11
M	A 10

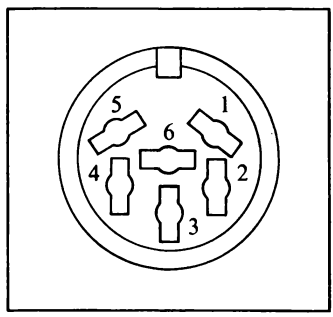
Pin No.	Tipo
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

Figura C.2. Contatti della porta di espansione



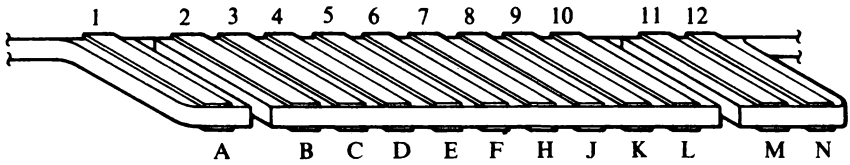
Pin No.	Tipo
1	LUMINOSITÀ
2	TERRA
3	USCITA AUDIO
4	USCITA VIDEO
5	ENTRATA AUDIO

Figura C.3. Pin della porta audio/video



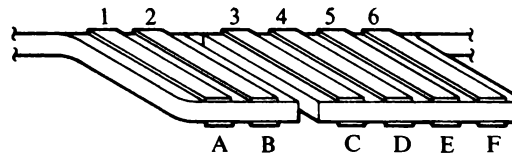
Pin No.	Tipo
1	SERIAL $\overline{\text{SRQ IN}}$
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	$\overline{\text{RESET}}$

Figura C.4. Pin della porta I/O seriale



Pin No.	Tipo
1	GND
2	+5 V
3	RESET
4	CONTROL 1
5	SP 1
6	CONTROL 2
7	SP 2
8	PC 2
9	SERIAL ATN IN
10	+9 V (100 mA max)
11	+9 V (100 mA max)
12	GND
A	GND
B	FLAG 2
C	PB0
D	PB1
E	PB2
F	PB3
H	PB4
J	PB5
K	PB6
L	PB7
M	PA2
N	GND

Figura C.5. Contatti della *user port*



Pin No.	Tipo
A e 1	TERRA
B e 2	+5 V
C e 3	MOTORE
D e 4	LETTURA
E e 5	SCRITTURA
F e 6	CASSETTA

Figura C.6. Contatti dell'interfaccia per il Datassette

Appendice

Tabelle di conversione e funzioni trigonometriche

D

CONVERSIONE ESADECIMALI/DECIMALI

La tabella D.1 fornisce conversioni dirette tra interi esadecimali compresi tra 0 e FFF e interi decimali compresi tra 0 e 4095. Per la conversione di numeri interi più elevati, i valori della tabella possono essere sommati a quelli della tabella D.2.
Frazioni esadecimali possono essere trasformate in frazioni decimali come segue:

1. Rappresentare la frazione esadecimale come un intero moltiplicato per 16^{-n}, dove n è il numero di cifre esadecimali significative poste alla destra del punto esadecimale.

$$0.\,CA9BF3_{16} = CA9\ BF3_{16} \times 16^{-6}$$

2. Trovare il decimale equivalente dell'intero esadecimale

$$CA9\ BF3_{16} = 13\ 278\ 195_{10}$$

3. Moltiplicare l'equivalente decimale per 16^{-n}

13	278	195
$\times$ 596	046	448×10^{16}
0.791	442	096_{10}

Tabella D.1. Conversione esadecimali/decimali

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

Tabella D.1. Conversione esadecimali/decimali (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0631
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

Tabella D.1. Conversione esadecimali/decimali (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

Tabella D.1. Conversione esadecimali/decimali (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

Tabella D.1. Conversione esadecimali/decimali (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

Tabella D.1. Conversione esadecimali/decimali (continua)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

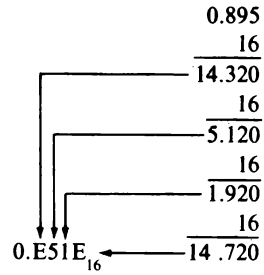
Tabella D.2. Valori di conversione

Esadecimale	Decimale	Esadecimale	Decimale	Esadecimale	Decimale	Esadecimale	Decimale
01 000	4 096	11 000	69 632	30 000	196 608	400 000	4 194 304
02 000	8 192	12 000	73 728	40 000	262 144	500 000	5 242 880
03 000	12 288	13 000	77 824	50 000	327 680	600 000	6 291 456
04 000	16 384	14 000	81 920	60 000	393 216	700 000	7 340 032
05 000	20 480	15 000	86 016	70 000	458 752	800 000	8 388 608
06 000	24 576	16 000	90 112	80 000	524 288	900 000	9 437 184
07 000	28 672	17 000	94 208	90 000	589 824	A00 000	10 485 760
08 000	32 768	18 000	98 304	A0 000	655 360	B00 000	11 534 336
09 000	36 864	19 000	102 400	B0 000	720 896	C00 000	12 582 912
0A 000	40 960	1A 000	106 496	C0 000	786 432	D00 000	13 631 488
0B 000	45 056	1B 000	110 592	D0 000	851 968	E00 000	14 680 064
0C 000	49 152	1C 000	114 688	E0 000	917 504	F00 000	15 728 640
0D 000	53 248	1D 000	118 784	F0 000	983 040	1 000 000	16 777 216
0E 000	57 344	1E 000	122 880	100 000	1 048 576	2 000 000	33 554 432
0F 000	61 440	1F 000	126 976	200 000	2 097 152		
10 000	65 536	20 000	131 072	300 000	3 145 728		

Frazioni decimali possono essere trasformate in frazioni esadecimali moltiplicando successivamente la frazione per 16_{10}. Dopo ogni moltiplicazione, la parte intera è rimossa per formare una frazione esadecimale alla destra del punto esadecimale. Tuttavia, siccome in questa trasformazione è utilizzata l'aritmetica decimale, la parte intera di ogni prodotto deve essere convertita in numeri esadecimali.

Esempio:

Trasformare 0.895_{10} nel suo equivalente esadecimale



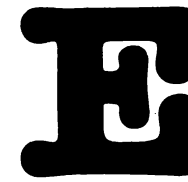
Le funzioni che non sono incorporate nel BASIC C-64 possono essere calcolate come mostra la tabella D.3.

Tabella D.3. Funzioni trigonometriche

Funzione	*Equivalente BASIC*
Secante	SEC(X) = 1/COS(X)
Cosecante	CSC(X) = 1/SIN(X)
Cotangente	COT(X) = 1/TAN(X)
Arcoseno	ARCSIN(X) = ATN(X/SQR(− X*X + 1))
Arcocoseno	ARCCOS(X) = −ATN(X/SQR (−X*X + 1)) + π/2
Arcosecante	ARCSEC(X) = ATN(X/SQR(X*X − 1))
Arcocosecante	ARCCSC(X) = ATN(X/SQR(X*X − 1)) + (SGN(X) − 1)* π/2
Arcocotangente	ARCOT(X) = ATN(X) + π/2
Seno iperbolico	SINE(X) = (EXP(X) − EXP(− X))/2
Coseno iperbolico	COSH(X) = (EXP(X) + EXP(− X))/2
Tangente iperbolica	TANH(X) = EXP(− X)/EXP(X) + EXP (−X))*2 + 1
Secante iperbolica	SECH(X) = 2/(EXP(X) + EXP(− X)
Cosecante iperbolica	CSCH(X) = 2/(EXP(X) − EXP(− X))
Cotangente iperbolica	COTH(X) = EXP(− X)/(EXP(X) − EXP(− X)*2 + 1
Arcoseno iperbolico	ARCSINH(X) = LOG(X + SQR(X*X + 1)
Arcocoseno iperbolico	ARCCOSH(X) = LOG(X + SQR(X*X − 1)
Arcotangente iperbolica	ARCTANH(X) = LOG((1 + X)/(1 − X))/2
Arcosecante iperbolica	ARCSECH(X) = LOG((SQR (− X*X + 1) + 1/X)
Arcocosecante iperbolica	ARCCSCH(X) = LOG((SGN(X)*SQR (X*X + 1)/X
Arcocotangente iperbolica	ARCCOTH(X) = LOG((X + 1)/(X − 1))/2

Appendice

Codici dei caratteri e abbreviazioni dei comandi



Le tabelle in questa appendice contengono tutti i codici dei caratteri del C-64.

La tabella E.1 contiene tutti i caratteri e le funzioni che possono essere visualizzati con l'istruzione CHR$. In molti casi l'uso di CHR$ è facoltativo; tuttavia alcune istruzioni, come RETURN e RUN/STOP, non sono programmabili con la funzione PRINT. Per programmare usando queste funzioni è necessario l'uso dell'istruzione CHR$ e dei codici di questa tabella.

I codici impiegati nella istruzione CHR$ non sono uguali a quelli utilizzati dai comandi POKE del video. I codici elencati nella tabella E.2 sono presentati nello stesso ordine dei caratteri in memoria. Si noti che mancano tutti i caratteri di controllo: questo perché non vi sono codici di visualizzazione specifici per essi; i codici di controllo fanno uso dei codici per i caratteri inversi standard (per esempio, un cuore in campo inverso per CLR/HOME).

Il C-64 è in grado di usare alcune abbreviazioni per i comandi che fanno risparmiare tempo nella battitura di righe d'istruzioni. In molti casi consistono della prima lettera della parola e della seconda lettera con SHIFT; in alcuni casi si devono caricare le prime due lettere e la terza con SHIFT. Vedere la tabella E.3 per ogni comando e notare che la presentazione non contiene la seconda (o terza) lettera, ma un carattere grafico.

Tabella E.1. Codici dei caratteri C-64

Simboli	CHR$	Simboli	CHR$	Simboli	CHR$	Simboli	CHR$
	0	CLR HOME	19		38		58
	1				39		59
	2	INST DEL	20		40		60
	3		21		41		61
	4		22		42		62
WHT	5		23		43		63
	6		24		44		64
	7		25		45		65
	8		26		46		66
	9		27		47		67
	10	RED	28		48		68
	11	CRSR →	29		49		69
	12	GRN	30		50		70
RETURN	13	BLU	31		51		71
SWITCH TO LOWER-CASE	14		32		52		72
			33		53		73
	15		34		54		74
	16		35		55		75
CRSR ↓	17		36		56		76
RVS ON	18		37		57		77

Tabella E.1. Codici dei caratteri C-64 (continua)

Simboli	CHR$	Simboli	CHR$	Simboli	CHR$	Simboli	CHR$
	78		98		118	f4	138
	79		99		119	f6	139
	80		100		120	f8	140
	81		101		121	SHIFT RETURN	141
	82		102		122		
	83		103		123	SWITCH TO UPPER-CASE	142
	84		104		124		
	85		105		125		143
	86		106		126	BLK	144
	87		107		127	↑ CRSR	145
	88		108		128	RVS OFF	146
	89		109	ORANGE	129	CLR HOME	147
	90		110		130		
	91		111		131	INST DEL	148
	92		112		132	BROWN	149
	93		113	f1	133	LT RED	150
	94		114	f3	134	DK GREY	151
	95		115	f5	135	MED GREY	152
	96		116	f7	136	LT GREEN	153
	97		117	f2	137	LT BLUE	154

Tabella E.1. Codici dei caratteri C-64 (continua)

Simboli	CHR$	Simboli	CHR$	Simboli	CHR$	Simboli	CHR$
LT GREY	155		175		195		215
PUR	156		176		196		216
CRSR ←	157		177		197		217
YEL	158		178		198		218
CYN	159		179		199		219
	160		180		200		220
	161		181		201		221
	162		182		202		222
	163		183		203		223
	164		184		204		224
	165		185		205		225
	166		186		206		226
	167		187		207		227
	168		188		208		228
	169		189		209		229
	170		190		210		230
	171		191		211		231
	172		192		212		232
	173		193		213		233
	174		194		214		234

Tabella E.1. Codici dei caratteri C-64 (continua)

Simboli	CHR$	Simboli	CHR$	Simboli	CHR$	Simboli	CHR$
	235		241		246		251
	236		242		247		252
	237		243		248		253
	238		244		249		254
	239		245		250		255
	240						

Tabella E.2. Codici video (Set di caratteri 1 e 2)

Set 1	Set 2	POKE	Set 1	Set 2	POKE	Set 1	Set 2	POKE
		0			10			20
		1			11			21
		2			12			22
		3			13			23
		4			14			24
		5			15			25
		6			16			26
		7			17			27
		8			18			28
		9			19			29

Tabella E.2. Codici video (continua)

Set 1	Set 2	POKE	Set 1	Set 2	POKE	Set 1	Set 2	POKE
		30			50			70
		31			51			71
		32			52			72
		33			53			73
		34			54			74
		35			55			75
		36			56			76
		37			57			77
		38			58			78
		39			59			79
		40			60			80
		41			61			81
		42			62			82
		43			63			83
		44			64			84
		45			65			85
		46			66			86
		47			67			87
		48			68			88
		49			69			89

Tabella E.2. Codici video (continua)

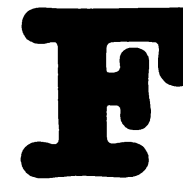
Set 1	Set 2	POKE	Set 1	Set 2	POKE	Set 1	Set 2	POKE
		90			103			116
		91			104			117
		92			105			118
		93			106			119
		94			107			120
		95			108			121
		96			109			122
		97			110			123
		98			111			124
		99			112			125
		100			113			126
		101			114			127
		102			115			

Tabella E.3. Abbreviazioni dei comandi

COMANDO	ABBREVIAZ.	VIS.	COMANDO	ABBREVIAZ.	VIS.
ABS	A SHIFT B	A	OPEN	O SHIFT P	O
AND	A SHIFT N	A	PEEK	P SHIFT E	P
ASC	A SHIFT S	A	POKE	P SHIFT O	P
ATN	A SHIFT T	A	PRINT	?	?
CHR$	C SHIFT H	C	PRINT#	P SHIFT R	P
CLOSE	CL SHIFT O	CL	READ	R SHIFT E	R
CLR	C SHIFT L	CL	RESTORE	RE SHIFT S	RE
CMD	C SHIFT M	C	RETURN	RE SHIFT T	RE
CONT	C SHIFT O	C	RIGHT$	R SHIFT I	R
DATA	D SHIFT A	D	RND	R SHIFT N	R
DEF	D SHIFT E	D	RUN	R SHIFT U	R
DIM	D SHIFT I	D	SAVE	S SHIFT A	S
END	E SHIFT N	E	SGN	S SHIFT G	S
EXP	E SHIFT X	E	SIN	S SHIFT I	S
FOR	F SHIFT O	F	SPC(	S SHIFT P	S
FRE	F SHIFT R	F	SQR	S SHIFT Q	S
GET	G SHIFT E	G	STEP	ST SHIFT E	ST
GOSUB	G0 SHIFT S	GO	STOP	S SHIFT T	S
GOTO	G SHIFT T	G	STR$	ST SHIFT R	ST
INPUT#	I SHIFT N	I	SYS	S SHIFT Y	S
LEFT$	LE SHIFT F	LE	TAB(	T SHIFT A	T
LET	L SHIFT E	L	THEN	T SHIFT H	T
LIST	L SHIFT I	L	USR	U SHIFT S	U
LOAD	L SHIFT O	L	VAL	V SHIFT A	V
MID$	M SHIFT I	M	VERIFY	V SHIFT E	V
NEXT	N SHIFT E	N	WAIT	W SHIFT A	W
NOT	N SHIFT O	N			

Appendice

Messaggi d'errore



I messaggi d'errore del C-64 possono essere emessi in risposta a qualunque istruzione diretta o durante l'esecuzione del programma. In quest'appendice sono elencati e spiegati i messaggi visualizzati dall'interprete del BASIC e dal sistema operativo. Quando l'interprete del BASIC individua un errore, presenta un messaggio diagnostico, preceduto da un "?", del tipo:

?*messaggio* ERROR IN *numero*

dove *messaggio* è il tipo di errore e *numero* è il numero della riga del programma in cui è avvenuto l'errore (non appare, ovviamente, in modo immediato). A seguito di un messaggio d'errore, il C-64 ritorna al modo immediato e visualizza READY.
Segue un elenco in ordine alfabetico dei messaggi d'errore accompagnato dalla descrizione della causa dell'errore e dei possibili rimedi.

BAD SUBSCRIPT

È stato richiamato un elemento di un array al di fuori delle sue dimensioni: può dipendere dall'aver indicato un numero di dimensioni differente dall'istruzione DIM, o dall'uso di un valore maggiore di 10 per un vettore non dimensionato.
Per ovviare all'errore, correggete il numero dell'elemento dell'array per rimanere entro i limiti originali o aumentatene le dimensioni perché possa accettare più elementi.

CAN'T CONTINUE

È stato battuto un comando CONT, ma l'esecuzione del programma non può continuare perché questo è stato modificato, ampliato o cancellato in modo immediato o perché l'arresto era stato causato da un errore. L'esecuzione di un programma non può continuare oltre un messaggio d'errore.

Correggete l'errore: la soluzione più prudente è di battere RUN e di cominciare di nuovo. Si può tentare anche di rilanciare l'esecuzione al punto d'interruzione con un GOTO diretto.

DEVICE NOT PRESENT

Nessun dispositivo è collegato al canale di I/O. La variabile di stato (ST) avrà un valore di 2, indicando un timeout. Questo messaggio può capitare per qualsiasi comando I/O.

Se l'identificazione del dispositivo è errata, correggete l'istruzione OPEN (o altra); se l'istruzione è corretta, specialmente se ha funzionato in precedenza, controllate il dispositivo interessato alla ricerca di un guasto, di un errato collegamento o di una mancata accensione.

DIVISION BY ZERO

È stato fatto un tentativo di divisione con un divisore uguale a zero. Controllate i valori delle variabili nella riga indicata. Modificate il programma in modo che il divisore non possa mai essere uguale a zero, oppure aggiungete un test di verifica prima di eseguire la divisione.

FILE ALREADY EXISTS

Il nome del file che si sta copiando con il comando COPY esiste già sul dischetto di destinazione.

FILE NOT FOUND

Il nome del file specificato nell'istruzione LOAD o OPEN non è stato trovato sul dispositivo utilizzato.

Controllate che nel dispositivo ci sia il dischetto o nastro giusto. Controllate i nomi dei file sul dischetto o nastro alla ricerca di un possibile errore di ortografia.

FILE NOT OPEN

È stato tentato un accesso ad un file logico non ancora aperto con un'istruzione OPEN. Aprite il file.

FILE OPEN

Si è tentato di aprire un file logico già aperto da una precedente istruzione OPEN.

Controllate il numero logico del file (il primo parametro nell'istruzione OPEN) per verificare che un numero diverso sia stato usato per ogni file. Inserite un'istruzione CLOSE se desiderate riaprire lo stesso file su un dispositivo diverso, o per un'operazione di I/O differente.

FORMULA TOO COMPLEX

Questo non è un errore di programma, ma indica che una espressione matematica nel programma è troppo complessa per il BASIC C-64.

Dividete l'espressione indicata in due o più parti (anche per una migliore leggibilità) e rieseguite il programma.

ILLEGAL DIRECT

È stato dato un comando nel modo immediato che è valido solamente nel corso di un programma. I comandi: DATA, DEF FN, GET, GET#, INPUT, INPUT#, non sono validi nel modo immediato.

Caricate l'operazione desiderata in forma di (breve) programma ed eseguitelo.

ILLEGAL QUANTITY

Una funzione il cui valore è al di fuori dei limiti permessi. Questo accade spesso con POKE quando si usano variabili maggiori di 255 o minori di 0.

Questo messaggio si verifica anche quando la funzione USR è usata prima di memorizzare l'indirizzo della subroutine nelle locazioni 1 e 2.

LOAD

Un numero inaccettabile (maggiore di 31) di errori si è accumulato durante il caricamento da un nastro. Essi non sono stati risolti leggendo la seconda traccia.

NEXT WITHOUT FOR

È stata incontrata un'istruzione NEXT non abbinata ad una precedente istruzione FOR, oppure manca un'istruzione FOR, oppure la variabile nell'istruzione NEXT non si trova nel corrispondente FOR.

Deve essere inserita la parte FOR di un loop FOR-NEXT, oppure rimossa l'istruzione NEXT interessata. Assicuratevi che le variabili indice siano uguali.

NOT INPUT FILE

Tentativo di leggere da un file su nastro, aperto solamente per output.

Controllate i parametri delle istruzioni READ# e OPEN. La lettura richiede uno 0 come terzo parametro di un OPEN.

NOT OUTPUT FILE

Si è tentato di scrivere su un file su nastro che è stato aperto solamente per input.
Controllate i parametri di PRINT# e di OPEN. Scrivere su un file richiede un 1 (o un 2 nel caso vi sia un EOT alla fine del file) come terzo parametro dell'istruzione OPEN.

OUT OF DATA

È stato eseguito un READ, ma tutti i DATA del programma sono stati già letti. Per ogni elemento di un READ, ve ne deve essere uno corrispondente in un DATA.
Aggiungete altri elementi DATA o diminuite il numero dei READ. Inserite un RESTORE per leggere di nuovo gli elementi di DATA, oppure aggiungete un flag alla fine dell'ultima istruzione DATA (può essere utilizzato qualsiasi valore che non compare come elemento di DATA) che ferma l'esecuzione dei READ quando si arriva a leggere il valore del flag.

OUT OF MEMORY

È stata riempita l'area di memoria a disposizione dell'utente e si richiede un'ulteriore riga di programma. Il messaggio può anche essere causato da annidamenti multipli di FOR-NEXT o GOSUB che riempiono l'area libera. Ne avrete la verifica se dopo ?FRE(0), è ancora disponibile un considerevole spazio di memoria di programma.
Semplificate il programma, con particolare attenzione alle dimensioni degli array. Potrebbe essere necessario ristrutturare il programma in diversi sottoprogrammi consecutivi.

OVERFLOW

Un calcolo ha dato un risultato che è al di fuori dei limiti massimi permessi. Il numero più alto permesso è 1,70141183E+38.
Controllate i calcoli. Potrebbe essere possibile eliminare l'errore cambiando l'ordine di esecuzione delle operazioni.

REDIM'D ARRAY

Un nome di array appare in più d'una istruzione DIM. Questo errore è presente anche se un nome di array viene usato e poi compare in un'istruzione DIM.
Posizionate tutte le istruzioni DIM vicino all'inizio di un programma. Controllate che ogni DIM venga eseguito una sola volta. DIM non deve

comparire all'interno di un loop FOR-NEXT o di una subroutine che potrebbe essere eseguita più di una volta.

REDO FROM START

Messaggio diagnostico durante un'operazione INPUT. Indica che sono stati caricati dati errati (a stringa invece di numerici o viceversa) in risposta ad una richiesta di input.
Ricaricate i dati correttamente. INPUT continuerà a richiederne fino a che non avrà ricevuto una risposta soddisfacente.

RETURN WITHOUT GOSUB

È stato incontrato un RETURN senza un precedente GOSUB corrispondente.
Inserite un'istruzione GOSUB o cancellate il RETURN. L'errore può essere causato se si entra per sbaglio nelle istruzioni di una subroutine. Un END o STOP posizionato immediatamente prima della subroutine aiuta ad identificare questo genere di errore.

STRING TOO LONG

Si è tentato di creare, per mezzo dell'operatore di concatenamento (+), una stringa più lunga di 255 caratteri.
Dividete la stringa in più stringhe brevi; fate uso di LEN per verificarne la lunghezza prima di concatenarle.

SYNTAX

Vi è un errore di sintassi nella riga appena caricata (modo immediato) o appena controllata per l'esecuzione (modo programmato). È il messaggio d'errore più comune, causato da errori ortografici, punteggiatura scorretta, parentesi non abbinate e caratteri estranei.
Esaminate la riga attentamente ed apportate le correzioni. Notate che gli errori sintattici sono diagnosticati al momento dell'esecuzione, non al momento dell'input dalla tastiera.

TYPE MISMATCH

Si è cercato di caricare una stringa in una variabile numerica o viceversa, oppure come parametro di una funzione.
Cambiate il dato errato e sostituitelo con uno corretto.

UNDEF'D FUNCTION

Si è fatto riferimento ad una funzione definita dall'utente non precedentemente definita da un'istruzione DEF FN. La definizione deve precedere il riferimento alla funzione.
Definite la funzione. Posizionate i DEF FN vicino all'inizio del programma.

UNDEF'D STATEMENT
Tentativo di saltare ad una riga inesistente.
Inserite un'istruzione con il numero di riga richiesto o saltate ad un altro numero di riga.

VERIFY ERROR
Il programma in memoria e il file specificato non sono uguali. Questo messaggio è usato congiuntamente al comando VERIFY.

Appendice

Istruzioni BASIC

In questa appendice è spiegata la sintassi di tutte le istruzioni BASIC. Sono presentate in ordine alfabetico e includono sia le funzioni interne che i comandi di I/O.

CLOSE

L'istruzione CLOSE chiude un file logico.

Struttura:

CLOSE *lf*

L'istruzione CLOSE chiude il file logico *lf*. Tutti i file dovrebbero essere chiusi dopo il loro utilizzo. Un file logico può essere chiuso una volta soltanto.

Esempio:

```
CLOSE 1      chiude il file logico 1
CLOSE 14     chiude il file logico 14
```

CLR

L'istruzione CLR azzera tutte le variabili numeriche e assegna valori nulli a tutte le variabili a stringa. Tutto lo spazio di memoria occupato dagli array viene reso di nuovo disponibile. È equivalente allo spegnere il computer, poi riaccenderlo e ricaricare il programma in memoria. CLR chiude tutti i file logici aperti dal programma in atto.

Struttura:

```
CLR
```

Dopo un'istruzione CLR il programma continuerà a girare purché l'esecuzione dell'istruzione non influisca sulla logica del programma, bloccandolo.

Esempio:

```
100 CLR
```

CMD

Il comando CMD trasferisce tutto l'output, che normalmente verrebbe presentato sul video, ad un altro dispositivo specificato. L'output viene mandato a quell'unità fino all'esecuzione di un'istruzione PRINT# seguita dal numero di file logico aperto. Almeno un'istruzione PRINT# deve sempre seguire un comando CMD.

Struttura:

```
CMD lf
```

L'istruzione CMD assegna un canale di output al file logico *lf*. Dopo l'esecuzione di un comando CMD al canale della stampante, sia PRINT che LIST stampano dati invece di presentarli sul video.

Esempio:

```
OPEN 5,4        Apre il file logico 5 selezionando la stampante
CMD 5           Indirizza l'output successivo alla stampante
LIST            Stampa il listato del programma
```

```
PRINT#5      Invia un ritorno a capo e disinserisce la stampante
CLOSE 5      Chiude il file logico 5
```

CONT

L'istruzione CONT, battuta sulla tastiera nel modo immediato, fa ricominciare l'esecuzione dopo un BREAK (interruzione).

Struttura:

```
CONT
```

L'interruzione può essere causata da un'istruzione STOP o END inserita nel programma o dalla pressione del tasto STOP durante l'esecuzione di un programma. Dopo un CONT, l'esecuzione continuerà esattamente dal punto in cui è avvenuta l'interruzione.
Il programma si blocca anche battendo RETURN dopo un INPUT. Battendo CONT dopo questa interruzione si farà eseguire l'INPUT di nuovo.

Esempio:

```
CONT
```

DATA

L'istruzione DATA contiene delle costanti che vengono assegnate a variabili da comandi READ.

Struttura:

DATA *costante*[,*costante*,*costante*,...,*costante*]

Istruzioni DATA possono comparire ovunque nel programma; esse specificano dati numerici o a stringa. Le stringhe sono generalmente rinchiuse tra virgolette; queste non sono necessarie se la stringa non contiene caratteri grafici, spazi vuoti, virgole o due punti. Questi ultimi dati vengono ignorati se non rinchiusi tra virgolette. Le virgolette stesse non possono essere incluse in una stringa di DATA; esse devono essere specificate per mezzo della funzione CHR$(34). L'istruzione DATA è valida solo nel modo programmato.

Esempio:

```
10 DATA NAME,"C.D."      Definisce due variabili a stringa
50 DATA 1E6,-10,XYZ      Definisce due variabili numeriche e una a stringa
```

Fate riferimento più avanti all'istruzione READ per vedere come le costanti di un DATA sono usate in un programma.

DEF FN

La funzione DEF FN permette di definire funzioni speciali e di usarle in un programma.

Struttura:

DEF FN*nvar*(*arg*)=*espressione*

La variabile a virgola mobile *nvar* identifica la funzione, che viene poi richiesta con il nome FN*nvar* (*data*). (Se *nvar* è lunga più di 5 lettere viene riportato un errore di sintassi, come pure nel caso in cui *nvar* sia una variabile a stringa o intera.) La funzione è specificata da *espressione*, che può essere qualunque espressione aritmetica contenente combinazioni di costanti, variabili o operatori numerici. La falsa variabile *arg* può (e normalmente lo è) essere inclusa nell'espressione; *arg* è l'unica variabile nell'espressione che può essere specificata quando viene fatto riferimento a FN*nvar*(*data*). Qualsiasi altra variabile deve essere specificata prima di chiamare la funzione la prima volta. FN*nvar*(*data*) calcola l'espressione usando *data* come valore di *arg*. L'intera istruzione DEF FN deve comparire su una singola riga di 80 caratteri; tuttavia una funzione precedentemente definita può essere inclusa in espressione, permettendo così di sviluppare funzioni speciali di qualsiasi complessità. Il nome della funzione *nvar* può essere riutilizzato e quindi ridefinito per un'altra funzione nello stesso programma. La definizione DEF FN non è valida nel modo immediato, ma una funzione speciale definita dal programma attualmente in memoria può essere usata da un'istruzione nel modo immediato.

Esempio:

```
10 DEF FNC(R)=π*R↑2      Definisce una funzione che calcola la superficie di un cerchio. Preleva un solo argomento R, il raggio del cerchio, e riporta un solo valore numerico, la superficie del cerchio.
?FNC(1)                  Stampa 3.14159265 (il valore di π)
```

```
55 IF FNC(X)>60 GOTO 150
```
Usa il valore della funzione speciale FNC come condizione di un salto. Il valore attualmente in X è usato per calcolare FNC.

DIM

L'istruzione di dimensionamento DIM assegna spazio di memoria agli array.

Struttura:

DIM *var*(*numel*[,*numel*,*numel*])

dove *var* è il nome dell'array e *numel* è il numero degli elementi contenuti.
DIM identifica array a una o più dimensioni come segue:

var(*numel*)	Array a una dimensione o vettore
var(*numel*,*numel*)	Array a due dimensioni o matrice
var(*numel*,*numel*,*numel*)	Array a tre dimensioni

I vettori con più di 11 elementi devono essere dimensionati con DIM; quelli con meno di 11 elementi (indici da 0 a 10) si possono sempre impiegare poiché sono dimensionati da un DIM; questi vettori sono dimensionati automaticamente quando il programma incontra il primo elemento del vettore. Un vettore con più di 11 elementi deve essere dimensionato in un'istruzione DIM prima che venga usato il primo elemento dello stesso. Se un array è dimensionato più di una volta, si verifica un errore e il programma si blocca.

Esempio:

```
10 DIM A(3)
```
Dimensiona un vettore a 3 elementi

```
45 DIM X$(44,2)
```
Dimensiona una matrice di 88 elementi

```
1000 DIM MU(X,3*B),N(12)
```
*Dimensiona una matrice bidimensionale di X volte 3*B elementi e un vettore a 12 elementi. X e B devono essere stati assegnati (avere dei valori) prima di eseguire DIM*

END

L'istruzione END conclude un programma e riporta il computer al modo diretto.

Struttura:

```
END
```

Il comando END offre la possibilità di bloccare il programma prima del termine delle istruzioni. Le istruzioni END possono essere usate quando vi è più d'un programma in memoria contemporaneamente. È facoltativo un END dopo l'ultima istruzione del programma.

Esempio:

```
20001 END
```

FOR-NEXT STEP

Tutte le istruzioni che si trovano tra un FOR e un NEXT sono eseguite lo stesso numero di volte.

Struttura:

```
FOR nvar=start TO end [STEP incremento]
[istruzioni nel loop]
NEXT [nvar]
```

dove

nvar	è l'indice del loop. Contiene il numero attuale dei cicli eseguiti. *nvar* viene spesso utilizzata da istruzioni all'interno del loop.
start	è una costante, variabile, espressione numerica che specifica il valore iniziale dell'indice.
end	è una costante, variabile, espressione numerica che specifica il valore finale dell'indice. Il loop è completo quando il valore dell'indice è uguale al valore di *end*, o quando lo supera.
incremento	quando presente, è una costante, variabile o espressione numerica che specifica l'aumento che subisce l'indice ad ogni passaggio. Questo STEP può essere positivo o negativo. Per difetto *incremento* è fissato =1.

Non è obbligatorio indicare *nvar* nell'istruzione NEXT. È ammesso un

solo NEXT per loop annidati che terminano allo stesso punto. L'istruzione NEXT in tal caso prende la forma di

NEXT $nvar_1$, $nvar_2$...

Il loop FOR-NEXT verrà eseguito sempre almeno una volta, anche se il valore iniziale di *nvar* è maggiore di quello finale. Se si omette il NEXT e non vengono incontrati NEXT in seguito, il loop verrà eseguito una volta. I valori di *start*, *end* e *incremento* sono letti una sola volta alla prima esecuzione di FOR. Non è possibile cambiare questi valori all'interno del loop, ma è possibile cambiare quello di *nvar*. Ciò rende possibile terminare il loop prima che sia raggiunto il valore finale fissando *nvar* al valore di *end*. Non uscite da un loop con un'istruzione GOTO e non iniziate un loop fuori da una subroutine per poi terminarlo dentro. I loop FOR-NEXT possono essere annidati; ognuno di essi, però, deve avere un nome di variabile *nvar* differente. Ogni loop annidato deve essere contenuto interamente all'interno del successivo esterno; al massimo, i loop possono terminare allo stesso punto.

GET

L'istruzione GET riceve caratteri singolarmente come input dalla tastiera.

Struttura:

GET *var*

Il GET può essere eseguito solamente nel modo programmato. Quando viene eseguita un'istruzione GET, *var* viene azzerata, perdendo ogni valore precedente. Poi GET preleva il primo carattere dal buffer della tastiera e lo assegna a *var*. Se il buffer della tastiera è vuoto, *var* conserva il suo valore di 0 o nullo.
GET è impiegata per acquisire un carattere dalla tastiera; accetta anche il tasto RETURN come input e, in questo caso, assegna a *var* un valore di CHR$(13). Se *var* è una variabile numerica e non viene premuto alcun tasto, si carica uno zero. Inoltre se *var* è numerica e viene caricato un carattere che non sia una cifra (0-9), viene generato un messaggio ?SYNTAX ERROR e il programma si ferma. L'istruzione GET accetta come parametro più di una variabile, ma diventa difficile da usare.

GET *var,var,...,var*

Esempio:

```
10 GET C$

10 GET D

10 GET A,B,C
```

GET#

L'istruzione GET# riceve come input singoli caratteri caricati da un dispositivo di memoria esterno identificato attraverso un numero di file logico.

Struttura:

GET#*lf,var*

Il comando GET#, che può essere usato solo nel modo programmato, preleva un solo carattere e lo assegna a *var*. Il dispositivo di memoria esterno è identificato dal numero di file *lf*. Questo deve essere stato precedentemente aperto da un'istruzione OPEN. Le istruzioni GET# e GET trattano variabili e dati in maniera identica.

Esempio:

```
10 GET#4,C$:IF C$="" GOTO 10
```
Preleva un carattere dal file logico 4 e ripete se non ha trovato nulla.

GOSUB

L'istruzione GOSUB fa saltare l'esecuzione del programma ad una riga specificata e predispone il ritorno dalla subroutine alla riga immediatamente successiva.

Struttura:

GOSUB *ln*

L'istruzione GOSUB chiama una subroutine con inizio alla riga *ln*. L'inizio di una subroutine è la riga che viene eseguita per prima; questa non

deve necessariamente essere la riga della subroutine con il numero inferiore. L'istruzione RETURN alla fine della subroutine rimanda il programma alla riga successiva al GOSUB che aveva causato il salto. Siccome un'istruzione GOSUB si può collocare ovunque nel programma, ne consegue che una subroutine può essere chiamata da un qualunque punto del programma.

Esempio:

```
100 GOSUB 2000      Salta alla subroutine a riga 2000
110 A=B*C
.
.
2000                Punto d'inizio della subroutine
.
.
2090 RETURN         Esegue il salto di ritorno a riga 110
```

GOTO

Il comando GOTO compie un salto incondizionato alla riga specificata.

Struttura:

GOTO *ln*

Esempio:

```
10 GOTO 100
```

Eseguita nel modo immediato, l'istruzione GOTO salta alla riga specificata facendo così eseguire il programma senza azzerare i valori delle variabili.

IF-THEN

Il costrutto IF-THEN provoca l'esecuzione di altre istruzioni secondo il risultato di un test condizionale.

Struttura:

IF *condizione* THEN *istruzione*
IF *condizione* THEN GOTO *riga*

Se la condizione proposta è vera, allora sono eseguite l'istruzione o le istruzioni che seguono il THEN, altrimenti l'esecuzione passa alla riga successiva, senza eseguire le istruzioni poste dopo il THEN.
Per ottenere un salto condizionato, l'indicazione della riga del salto segue il GOTO.
Sono accettabili anche le forme contratte: IF *condizione* THEN *ln* e IF *condizione* GOTO *ln*

```
IF A=1 THEN 50
IF A=1 GOTO 50
IF A=1 THEN GOTO 50
```

Equivalenti

Se GOTO è una delle istruzioni che seguono THEN sulla stessa riga, deve essere l'ultima ed avere la struttura normale del GOTO. Nel caso in cui il GOTO non fosse l'ultima istruzione della riga, le istruzioni successive non verrebbero mai eseguite.
Le seguenti istruzioni non sono ammesse per degli IF-THEN eseguiti nel modo immediato: DATA, GET, GET#, INPUT, INPUT#, REM, RETURN, END, STOP, WAIT.
La riga oggetto del GOTO deve esistere nel programma. I comandi CONT e DATA non sono permessi in un IF-THEN nel modo programmato. Se un loop FOR-NEXT segue un THEN, deve essere completamente contenuto sulla riga stessa. Istruzioni IF-THEN aggiuntive possono trovarsi sulla riga dopo THEN, ma devono, in tal caso, essere completamente contenute su di essa. Tuttavia è consigliabile usare operatori Booleani piuttosto che numerosi IF-THEN concatenati. Per esempio, le seguenti due righe sono equivalenti, ma la seconda è preferibile.

```
10 IF A$="X" THEN IF B=2 THEN IF C>D THEN 50
10 IF A$="X" AND B=2 AND C>D THEN 50
```

Esempio:

```
400 IF X>Y THEN A=1
500 IF M+1 THEN AG=4.5 GOSUB 1000
```

INITIALIZE

Si usa PRINT# per inizializzare un dischetto prima di compiere un'operazione su di esso.

Struttura:

PRINT# *file*,"I[dr]"

Il dischetto nell'unità *dr* è così inizializzato. Se il parametro *dr* non è presente, verrà inizializzato il dischetto nell'unità 0.

Esempio:

```
OPEN 1,8,15      Apre il canale di comando del dischetto
PRINT#1,"I"      Inizializza il dischetto nel drive 0
```

INPUT

L'istruzione INPUT riceve dati dalla tastiera.

Struttura:

INPUT { *(spazio vuoto)* / *"messaggio";* } *var[var,...,var]*

INPUT può essere usato soltanto nel modo programmato. Quando viene eseguito un INPUT il computer presenta sul video un "?" per richiedere i dati.
L'utente deve caricare esattamente il numero e tipo di dati che compaiono nella lista dei parametri dell'istruzione INPUT; se ve n'è più d'uno, i dati caricati dalla tastiera devono essere separati da virgole e l'ultimo dato seguito da un ritorno a capo (<RETURN>).

```
?1234<RETURN>                  Acquisisce un dato
?1234,567.89,NOW<RETURN>       Acquisisce diversi dati
```

L'eventuale "*messaggio*" viene presentato prima del "?": può essere lungo fino a un massimo di 80 caratteri. Nel caso in cui sia caricato un numero di dati insufficiente, il BASIC C-64 richiede ulteriori dati con "?" fino all'avvenuto carico del numero giusto di dati. Se se ne sono caricati troppi, compare il messaggio ?EXTRA IGNORED. I dati supplementari vengono così ignorati, ma l'esecuzione continua.

Esempio:

Istruzione	*Risposta dell'operatore*	*Risultato*
10 INPUT A,B,C$	? 123,456,NOW	A=123,B=456,C$="NOW"
10 INPUT A,B,C$	? 123 ?? 456 ?? NOW	A=123 B=456 C$="NOW"
10 INPUT A,B,C$	? NOW ?REDO FROM START ? 123 ?? 456 ?? 789	 A=123 B=456 C$="789"
10 INPUT "A=";A	A= ? 123	A=123

Notate che è necessario caricare dati numerici in risposta ad un INPUT per variabili numeriche, ma è possibile caricare sia dati numerici sia a stringa per variabili a stringa.

INPUT #

L'istruzione INPUT esterna (INPUT #) carica uno o più dati da un'unità esterna identificata dal numero di file logico.

Struttura:

INPUT # *lf,var*[,*var*,...,*var*]

Il comando INPUT # preleva i dati da un'unità esterna e li assegna alla variabile *var*. Questi devono essere uguali in numero e tipo ai parametri specificati dall'istruzione INPUT #. Se viene rilevata una segnalazione di fine record prima che tutti i parametri siano stati soddisfatti sarà generato uno stato d'errore OUT OF DATA, ma il programma continuerà a girare. INPUT e INPUT # funzionano in maniera identica con la differenza che INPUT # accetta dati da un file. Inoltre INPUT # non presenta messaggi d'errore, riporta uno *stato* d'errore che il programma deve poter interpretare. Le stringhe di input non devono superare gli 80 caratteri (79 caratteri e un ritorno a capo) dato che il buffer di INPUT dispone di un massimo di 80 byte di memoria. Le virgole e i ritorni a capo vengono trattati come separatori di dati; sono riconosciuti, ma non vengono forniti al computer come dati. INPUT # è valido solo nel modo programmato.

Esempio:

```
1000 INPUT#10,A
```
Preleva il dato successivo dal file logico 10. Assegna il valore alla variabile numerica A

```
946 INPUT#12,A$
```
Preleva il dato successivo dal file logico 12. Si aspetta una stringa che viene assegnata alla variabile A$

```
900 INPUT#5,B,C$
```
Preleva i due dati successivi dal file logico 5. Il primo dato è numerico e va assegnato a B; il secondo è uno a stringa che va alla variabile C$

LET

L'istruzione di assegnamento LET assegna un valore ad una variabile specifica.

Struttura:

[LET] *var*=*data*

Alla variabile *var* viene assegnato il valore di *data*. La parola LET è facoltativa, e generalmente omessa.

Esempio:

```
10 A=2
450 C$="▼"


300 M(1,3)=SGN(X)
310 XX$(I,J,K,L)="LUNGA STRINGA"
```

LIST

LIST visualizza una o più righe di programma; le istruzioni così visualizzate possono essere corrette dall'editor del video.

Struttura:

LIST { numero riga
numero riga$_1$ – numero riga$_2$
–numero riga
numero riga – }

In risposta ad un LIST viene listato l'intero programma. Per listare solo parte di programmi lunghi che non starebbero sullo schermo usate i parametri di limitazione.

Esempio:

```
LIST          Lista l'intero programma
LIST 50       Lista la riga 50
LIST 60-100   Lista le righe da 60 a 100 incluse
LIST -140     Lista tutte le righe fino a 140
LIST 20000-   Lista tutte le righe da 20000 in poi
```

Le righe listate sono ristrutturate come segue:

1. I "?" caricati come abbreviazione di PRINT vengono scritti per esteso. Esempio:

   ```
   ?A  diviene PRINT A
   ```

2. Gli spazi che precedono il numero di riga sono eliminati, così come quelli tra il numero e il contenuto.

   ```
      50      A=1                    50 A=1
                       diventano
   60     B=2                        60 B=2
   ```

3. È inserito uno spazio tra il numero di riga e il resto dell'istruzione. Esempio:

   ```
   70C=B+A              diventano     70 C=B+A
   ```

LIST è sempre usato nel modo diretto. Un LIST nel programma verrebbe eseguito, ma farebbe poi tornare il computer al modo diretto interrompendo l'esecuzione. Il tentativo di far continuare il programma con CONT ripeterebbe semplicemente il LIST.

Come stampare il listato di un programma

Per stampare il listato di un programma invece di presentarlo sul video, aprite un file alla stampante ed eseguite un CMD prima del LIST, come segue:

```
OPEN 4,4      Apre il canale 4 alla stampante
CMD 4         Indirizza l'output al canale aperto invece che al video
LIST          Stampa il listato del programma
PRINT#4       Indirizza l'output di nuovo al video
```

LOAD

L'istruzione LOAD carica un programma in memoria da un dispositivo di memoria esterna.

Lettura di programma su cassetta

LOAD["*nome del file*"],[*dev*]

L'istruzione LOAD carica in memoria il file del programma specificato dal *nome del file*, dall'unità di registrazione selezionata dal numero di dispositivo *dev*.
In assenza di questo numero, è selezionata l'unità numero 1. In assenza di un nome di file, verrà caricato il primo file trovato.

Esempio:

```
LOAD              Carica il primo programma trovato sull'unità #1. Richiedendo un LOAD mentre l'unità sta leggendo un programma si farà continuare la lettura oltre la fine del primo programma e si caricherà il seguente.
LOAD "",2         Carica il primo programma trovato sull'unità #2
LOAD "DELFINO"    Ricerca il programma chiamato DELFINO sull'unità #1 e lo carica
N$="GATTO"        Ricerca il programma chiamato GATTO sull'unità #1 e lo
LOAD N$           carica
```

Lettura di programma su dischetto

LOAD "[*dr:*]*nome del file*",*dev*

Il comando LOAD carica in memoria dall'unità a dischetti il programma chiamato *nome del file*. Il numero *dev* per l'unità del C-64 è 8. In assenza di *dev* il valore per difetto è 1, che è l'unità a nastro. Un asterisco al posto del nome del file farà caricare il primo programma trovato sull'unità.

Esempio:

```
LOAD "0:*",8          Carica il primo programma trovato sul drive numero 0

LOAD "0:TOPO",8       Ricerca il programma chiamato TOPO sul drive 0 e lo
                      carica
T$="0:TRICHECO"       Ricerca il programma chiamato TRICHECO sul drive 0
LOAD T$,8             e lo carica
```

Quando un LOAD è eseguito nel modo immediato, il BASIC C-64 esegue automaticamente un CLR prima di caricare il programma. Una volta caricato, un programma può essere listato, corretto o eseguito.
Un LOAD può essere eseguito anche nel modo programmato. per ottenere un *overlay*. Un LOAD eseguito da un programma farà fermare quel programma e ne caricherà un altro. In questo caso il C-64 non eseguirà un CLR, per cui il primo programma è in grado di passare tutti i valori delle proprie variabili all'altro. Quando viene eseguita nel modo programmato un'istruzione LOAD che accede ad un'unità delle cassette, tutti i messaggi vengono soppressi se non è premuto il tasto PLAY sul registratore; in questo caso compare il messaggio PRESS PLAY ON TAPE. Un LOAD programmato a un drive non fa comparire alcun messaggio.

NEW

L'istruzione NEW cancella dalla memoria il programma in essa contenuto.

Struttura:

```
NEW
```

Quando viene eseguita un'istruzione NEW tutte le variabili sono azzerate e viene reso disponibile lo spazio di memoria degli array. Sono azzerati anche i puntatori che tengono conto dell'esecuzione delle istruzioni del programma. Le operazioni di NEW sono eseguite automaticamente per ogni LOAD. Se vi è un programma in memoria, si deve eseguire un NEW prima di battere un programma sulla tastiera, altrimenti il nuovo programma si sovrappone al preesistente sostituendo le righe con lo stesso numero, ma lasciando le altre intatte, con il risultato di creare un miscuglio inutilizzabile dei due programmi.

Esempio:

```
NEW
```

NEW è sempre usato nel modo immediato. Se viene eseguito da un programma, quest'ultimo si "autodistrugge".

NEW (Comando DOS)

Viene impiegato con PRINT# per preparare e formattare un dischetto nuovo o per cancellare e riformattarne uno usato.

Struttura:

PRINT# *lf*,"N[EW]*dr:nome del disco,num*"

Il dischetto nell'unità *dr* viene formattato. Durante quest'operazione i settori sono disposti sul dischetto, il catalogo e la BAM sono inizializzati. Al dischetto sono assegnati il *nome del disco* e il numero *num*, i quali sono presentati in caratteri inversi in testa al catalogo.

Esempio:

```
OPEN 1,8,15                    Apre il canale di comando all'unità dischetti.
.
PRINT#1,"N8:GRAFICI,02"        Un dischetto è stato preparato all'uso nel
                               drive 8. Il dischetto è chiamato GRAFICI e
                               ha il numero 02
```

ON-GOSUB

L'istruzione ON-GOSUB chiama diverse subroutine a seconda del valore di una variabile.

Struttura:

ON *var* GOSUB $line_1$[, $line_2$, $line_3$...$line_n$]

ON-GOSUB ha la stessa struttura di ON-GOTO; fate riferimento a ON-GOTO per le regole di salto. Se necessario, *var* è calcolato e troncato ad un valore intero.
Per *var*=1, l'esecuzione passa alla subroutine alla riga $line_1$. Questa subroutine termina con un RETURN che fa saltare l'esecuzione alla riga immediatamente successiva a quella di ON-GOSUB. Se *var*=2, viene chiamata la subroutine con inizio alla riga $line_2$, e così via. ON-GOSUB è nor-

malmente eseguito nel modo programmato, ma può essere eseguito anche nel modo immediato purché vi siano dei corrispondenti numeri di riga nel programma al momento in memoria.

Esempio:

```
10 ON A GOSUB 100,200,300
```

ON-GOTO

L'istruzione ON-GOTO fa saltare l'esecuzione del programma ad una delle righe specificate secondo il valore di una variabile.

Struttura:

ON *var* GOTO $line_1$[,$line_2$,$line_2$,...$line_n$]

var viene calcolato e, se necessario, troncato a valore intero. Se *var*=1, avviene un salto alla riga $line_1$. Se *var*=2, avviene un salto alla riga $line_2$, e così via. Se *var*=0 non avviene nessun salto, così pure se *var* è entro i limiti permessi, ma non vi è nessuna riga corrispondente nel programma. Quando non avviene nessun salto l'esecuzione procede all'istruzione successiva, che può anche essere sulla stessa riga dell'ON-GOTO, separata da ":" o sulla riga seguente. Se l'indice *var* ha un valore reale al di fuori dei limiti permessi, il programma si blocca con un messaggio d'errore. Si possono specificare tanti numeri di riga quanti ce ne stanno su una riga di 80 caratteri. ON-GOTO viene normalmente eseguito nel modo programmato, ma può essere eseguito anche nel modo immediato purché vi siano righe corrispondenti nel programma in memoria.

Esempio:

```
40 A=B<10
50 ON A+2 GOTO 100,200
```

Salta all'istruzione 100 se la condizione è verificata (−1) oppure salta alla 2000.

```
50 X=X+1
60 ON X GOTO 500,600,700
```

Salta all'istruzione 500 se X=1, alla 600 se X=2, alla 700 se X=3. Non esegue alcun salto se X>3

OPEN

L'istruzione OPEN apre un file logico al dispositivo prescelto.

File di dati su cassetta

OPEN *lf*[,*dev*][,*sa*][,"*nome del file*"]

Viene aperto il file chiamato *nome del file* all'unità a cassette identificata dal numero *dev*, per il tipo di accesso specificato dall'indirizzo secondario *sa*; il tutto tramite il file logico o canale numero *lf*. Se non è specificato nessun nome, si accederà al primo file trovato sull'unità prescelta.
Non specificando *dev*, verrà assegnato il numero 1. Il numero scelto per l'indirizzo secondario *sa*, se non altrimenti specificato, sarà 0, e il file aperto per la sola lettura. Un indirizzo secondario di 1 apre il file per la scrittura, mentre un *sa* 2 apre il file per la scrittura con una segnalazione di fine file apposta alla sua chiusura.

Esempio:

`OPEN 1`	*Apre il file logico 1 all'unità a cassette #1 per sola lettura (sa=0) del primo file trovato sul nastro (nessun nome specificato)*
`OPEN 1,1`	*Come sopra*
`OPEN 1,1,0`	*Come sopra*
`OPEN 1,1,0,"DATI"`	*Come sopra, ma accede al file di nome DATI*
`OPEN 3,1,2`	*Apre il file logico #3 della cassetta #1 per una scrittura con segnalazione di fine file. Il nuovo file è senza nome e verrà trascritto alla attuale posizione fisica del nastro.*
`OPEN 3,1,2,"PENTAGRAMMA"`	*Come sopra, ma accede al file chiamato PENTAGRAMMA*

File di dati su dischetto

OPEN *lf,dev,sa,"dr:nome del file,tipo*[,*accesso*]"

Si apre il file chiamato *nome del file* sul dischetto nell'unità *dr* con un numero di file logico *lf*. *tipo* identifica il file come sequenziale (SEQ), di programma (PRG) o accesso diretto (USR). Se il file è sequenziale, l'*accesso* deve essere WRITE per specificare una scrittura, oppure READ per specificare una lettura. *Accesso* non va specificato per i file di programma ad accesso diretto.
Un file sequenziale preesistente può essere aperto per un accesso di scrittura se *dr* è preceduto dal simbolo @. Il contenuto precedente è sostituito dai dati nuovi. Il numero *dev* deve essere presente: è 8 per tutte le unità a dischetti standard. Se *dev* manca, viene assegnato un valore =1 e scelta così l'unità a cassette. Per un file di dati l'indirizzo secondario *sa*

può avere un valore tra 2 e 14, ma ogni file di dati aperto dovrebbe avere il proprio numero d'indirizzo secondario. Un numero *sa*=15 seleziona il canale di comando dell'unità a dischetti. Indirizzi secondari di 0 e 1 sono usati per accedere ai file di programma: un indirizzo 0 per leggere un file di programma e 1 per memorizzarlo.

Esempio:

```
OPEN 1,8,2,"0:DAT,SEQ,READ"
```
Apre il file logico 1 sul dischetto nell'unità 0. Legge dal file sequenziale DAT

```
OPEN 5,8,3,"1:EDIGEO,SEQ,WRITE"
```
Apre il file logico 5 sul dischetto nell'unità 1. Scrive sul file sequenziale EDIGEO

```
OPEN 4,8,4,"@1:GIOCO,SEQ,WRITE"
```
Apre il file logico 4 sul dischetto nell'unità 1. Scrive sul file sequenziale GIOCO sostituendo il contenuto precedente.

POKE

L'istruzione POKE memorizza un byte di dati in una specifica locazione di memoria.

Struttura:

POKE *indirizzo,byte*

Un valore tra 0 e 255, determinato da *byte*, è trascritto nella locazione di memoria all'*indirizzo* specificato.

Esempio:

```
10 POKE 1,A
```
Carica il valore della variabile A nella memoria all'indirizzo 1

```
POKE 32768,ASC("A")-64
```
Carica 1 (il valore di ASC("A")-64) nella memoria all'indirizzo 32768

PRINT

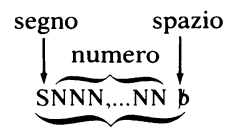
L'istruzione PRINT visualizza i dati; viene usata anche per stampare sulla stampante.

Struttura:

$$\left\{\begin{matrix}\text{PRINT}\\ ?\end{matrix}\right\}\ dati\ \left\{\begin{matrix},\\ ;\end{matrix}\right\}\ dati...\ \left\{\begin{matrix},\\ ;\end{matrix}\right\}\ dati$$

Struttura dei campi dell'istruzione PRINT

I campi numerici sono presentati usando la notazione standard per numeri compresi tra i valori di 0,01 e 999999999. La notazione esponenziale è impiegata per numeri al di fuori di questi limiti. I numeri sono preceduti dal carattere del segno e seguiti da uno spazio.



Visualizzazione di un campo numerico

I numeri positivi non hanno segno, quelli negativi hanno segno meno (−). Le stringhe sono presentate senza modifiche.

Struttura di PRINT

Primo dato. Il primo dato è presentato nella posizione del cursore. Il carattere di formattazione di PRINT (virgola o punto e virgola) dopo il primo dato specifica la posizione in cui sarà stampato il secondo. I dati possono essere nella stessa istruzione PRINT o in altre separate.
Nuova riga. Quando non vi è una virgola o un punto e virgola dopo l'ultimo dato di un PRINT, viene eseguito automaticamente un ritorno a capo dopo che è stato stampato l'ultimo dato dell'istruzione.
Tabulazione. Una virgola dopo un dato, fa sì che quello successivo venga presentato nella successiva posizione di tabulazione. Le tabulazioni standard sono alle colonne 1, 11, 21 e 31. Se una virgola precede il primo dato, questo sarà visualizzato alla seconda tabulazione.
Stampa contigua. Se un punto e virgola segue un dato, il successivo viene presentato nella successiva posizione disponibile. I dati numerici sono seguiti da un carattere nullo. Le stringhe sono presentate contiguamente senza alcun carattere inserito tra di esse.

Esempio:

```
40 PRINT A
40 PRINT A,B,C
40 PRINT A;B;C
40 PRINT, A;B;C
40 PRINT "NUMERI",A;B;C
40 PRINT "NU";"MERI";
41 PRINT "S",A;B;C
```

PRINT #

L'istruzione PRINT esterna (PRINT #) trasmette uno o più dati ad un dispositivo esterno (unità a cassette, unità a dischetti o stampante) identificato da un numero di file logico.

Struttura:

PRINT # *lf,dati separatori dati separatori dati*

I *dati* elencati nell'istruzione PRINT # come parametri della stessa sono trasmessi all'unità identificata dal file logico *lf*, intercalati da *separatori* che possono essere RETURN o CHR$(13), la virgola o CHR$(44), il punto e virgola o CHR$(59). Bisogna rispettare regole di punteggiatura molto rigide usando unità esterne; un breve riassunto di queste è fornito qui di seguito.

L'output ai file su cassetta

Ogni variabile numerica o a stringa trasferita ad un file su cassetta deve essere seguita da un carattere di ritorno a capo. Quest'ultimo è generato automaticamente da un PRINT # con un unico dato, ma un PRINT # che abbia più dati da trasmettere deve includere anche caratteri di ritorno a capo. Usate CHR$(13) per trasmettere un ritorno a capo, oppure una variabile a stringa alla quale sia stato assegnato quel valore come ad es. C$=CHR$(13).

L'output ai file su dischetti

Le regole di output sopra descritte per i file su cassetta sono applicabili anche a quelli su dischetti, con una eccezione: gruppi di variabili a stringa possono essere separati da virgole (CHR$(44)). Le virgole usate come caratteri separatori, come il ritorno a capo, devono essere inserite con CHR$. Le variabili a stringa scritte in un file su dischetto con virgole come separatori devono poi essere ricaricate in memoria usando un solo INPUT#.
L'istruzione INPUT# legge tutto il testo da un carattere di ritorno a capo all'altro.

L'output alla stampante

Quando l'istruzione PRINT# trasmette l'output ad una stampante, CHR$ deve essere CHR$(59). Nessuna punteggiatura deve separare i dati da CHR$, come illustrato nella definizione della struttura di PRINT#.
Attenzione: Non è ammessa la forma contratta ?# in sostituzione di PRINT#.

READ

L'istruzione READ assegna valori contenuti in un'istruzione DATA alle variabili elencate nella propria lista di parametri.

Struttura:

READ *var*[,*var*,*var*]

READ, usato per assegnare valori a variabili, può sostituire numerose istruzioni di assegnazione (vedi LET).
Le istruzioni READ con elenchi di variabili richiedono corrispondenti istruzioni DATA con elenchi di costanti da assegnare. Gli elenchi di READ e DATA devono coincidere come tipo di dati.
Una variabile a stringa accetta qualsiasi tipo di valore, ma variabili numeriche accettano esclusivamente valori numerici. Il numero delle istruzioni READ e DATA può essere differente, ma vi deve essere un valore DATA disponibile per ogni variabile di READ. Vi possono essere più valori DATA che variabili READ, ma se ve ne sono di meno il programma si blocca con un messaggio d'errore ?OUT OF DATA.
READ è normalmente eseguito nel modo programmato, ma può esserlo anche nel modo immediato purché vi siano corrispondenti costanti DATA nel programma in memoria al momento.

Esempio:

```
10 DATA 1,2,3          Quando eseguito, A=1, B=2, C=3
20 READ A,B,C


150 READ C$,D,F$       Quando eseguito, C$="STR", D=14.5, F$="TM"
160 DATA STR
170 DATA 14.5,"TM"
```

REM

Il comando REM permette di inserire commenti nel programma a scopo di documentazione, senza disturbare l'esecuzione.

Struttura:

REM *commento*

dove:

commento è qualsiasi sequenza di caratteri contenuta in due righe.

Le istruzioni REM vengono riportate sui listati dei programmi, ma vengono ignorate. Una REM può occupare una riga o essere posta come ultima istruzione su una riga di istruzioni multiple. Non può essere posizionata davanti a una istruzione, dal momento che tutto ciò che segue la REM sulla stessa riga viene trattato come commento e quindi ignorato dal computer durante l'esecuzione. Le REM possono trovarsi nel corpo del programma e anche essere il punto d'arrivo di un salto.

Esempio:

```
10 REM *** * * * * ***
20 REM ***BIANCANEVE***
30 GOTO 55:REM SALTA
```

RESTORE

L'istruzione RESTORE fa ritornare il puntatore dell'istruzione DATA all'inizio dei dati.

Struttura:

RESTORE

RESTORE si può eseguire sia nel modo immediato che programmato.

Esempio:

```
10 DATA 1,2,N44
20 READ A,B,B$          A=1, B=2, B$="N44"
30 RESTORE
40 READ X,Y,Z$          X=1, Y=2, Z$="N44"
```

RETURN

L'istruzione RETURN fa saltare l'esecuzione del programma alla riga immediatamente successiva a quella del GOSUB eseguito per ultimo. Ogni subroutine deve terminare con un RETURN.

Struttura:

RETURN

Esempio:

```
100 RETURN
```

Notate che l'istruzione RETURN riporta l'esecuzione al programma principale, mentre il tasto RETURN fa saltare il cursore all'inizio della riga successiva. Le due azioni non sono minimamente collegate.

RUN

RUN fa iniziare l'esecuzione del programma in memoria, chiude ogni file aperto e azzera tutte le variabili.

Struttura:

RUN[*line*]

Quando RUN è eseguito nel modo immediato, il computer compie le operazioni di CLR e RESTORE delle variabili e dei DATA prima di iniziare

l'esecuzione del programma. Se RUN specifica un numero di riga, il computer compie sempre le operazioni CLR e RESTORE, ma inizia l'esecuzione alla riga specificata. Un RUN che specifica un numero di riga non deve essere usato per continuare l'esecuzione dopo un'interruzione; per questo bisogna eseguire un CONT o un GOTO. RUN può essere impiegato anche nel modo programmato: fa ricominciare l'esecuzione del programma da capo compiendo tutte le operazioni CLR e RESTORE di cui sopra.

Esempio:

RUN	*Inizializza le variabili e lancia l'esecuzione del programma*
RUN 1000	*Inizializza le variabili e lancia l'esecuzione del programma dalla riga 1000*

SAVE

L'istruzione SAVE copia un programma residente in memoria su un'unità di memoria esterna.

L'unità a cassette

SAVE["*nome del file*"][,*dev*][,*sa*]

L'istruzione SAVE trascrive il programma in memoria sul nastro dell'unità specificata dal numero *dev*. Se *dev* non è specificato, viene assunto, per difetto, il numero 1 che specifica l'unità a cassette principale. Il nome del file, se presente, è trascritto all'inizio del programma. Se è specificato un indirizzo secondario (*sa*) diverso da zero viene trascritta una segnalazione di fine file alla fine del programma.
Benché non sia necessario quando si trascrive su nastro, è sempre meglio dare un nome a tutti i programmi in modo da poterli poi ricaricare per mezzo del nome. Un programma senza nome può essere caricato solo conoscendo la sua posizione sulla cassetta. L'istruzione SAVE è usata più frequentemente nel modo immediato, ma può essere impartita anche dall'interno di un programma.

Esempio:

SAVE	*Trascrive il programma in memoria sul nastro nell'unità 1 lasciandolo senza nome*
SAVE "COLORE"	*Trascrive il programma in memoria sul nastro nell'unità 1, chiamandolo COLORE*

```
A$="COLORE"          Come il precedente
SAVE A$
SAVE "RAMINO",2,1    Trascrive il programma in memoria sul nastro
                     nell'unità 2, chiamandolo RAMINO.
```

L'unità a dischetti

SAVE '[*dr:*]*nome del file*",*dev*

L'istruzione SAVE trascrive una copia del programma residente in memoria sul dischetto dell'unità *dr*. Il programma è chiamato *nome del file* e *dev* deve essere presente; in genere ha un valore di 8. Nel caso in cui manchi *dev*, è assunto un valore = 1 e la copia mandata all'unità a cassette. Il nome assegnato al programma non deve essere già presente nel catalogo del disco altrimenti verrà generato un errore di sintassi.
Un file può essere sostituito usando il simbolo @ davanti al numero *dr* nell'istruzione SAVE; il programma residente in memoria verrà trascritto al posto della versione originale dello stesso nome. L'istruzione SAVE è anch'essa utilizzata principalmente nel modo immediato, ma può essere anche inserita nel programma.

STOP

L'istruzione STOP causa l'interruzione dell'esecuzione del programma in atto e riporta il controllo al BASIC C-64; fa comparire sul video un messaggio d'interruzione.

Struttura:

STOP

Esempio:

```
655 STOP          Farà apparire il messaggio BREAK IN 655
```

VALIDATE

Struttura:

PRINT # *lf*,"V[ALIDATE]*dr*"

Il dischetto nel drive *dr* è convalidato. Quando non è presente il parametro *dr*, viene convalidato il dischetto nel drive usato per ultimo. Durante una convalida, viene creata una BAM nuova per tutti i file di dati presenti sul dischetto.
Qualsiasi file, impropriamente chiuso o non chiuso del tutto, viene cancellato con conseguente recupero dello spazio sul dischetto. Non convalidate un dischetto contenente file ad accesso diretto: quest'operazione li cancellerebbe. Se avviene un errore di lettura durante una convalida, l'operazione è interrotta e il dischetto lasciato nel suo stato originale. Un dischetto deve essere inizializzato dopo una convalida.

Esempio:

```
OPEN 1,8,15          Apre il canale di comando del dischetto
PRINT#1,"V0"         Convalida il dischetto nel drive 0
PRINT#1,"I0"         Inizializza il dischetto nel drive 0
```

VERIFY

L'istruzione VERIFY confronta un file contenuto in un'unità di memoria con il programma al momento in memoria.

L'unità a cassette

VERIFY["*nome del file*"][,*dev*]

Il programma attualmente in memoria è paragonato a quello nel file chiamato *nome del file* sul nastro dell'unità a cassette identificata dal numero *dev*. Se *dev* manca, viene assunto il valore 1. Se manca il *nome del file* viene verificato il primo file incontrato sul nastro.
È buona norma verificare un programma immediatamente dopo averlo trascritto. L'istruzione VERIFY è quasi sempre eseguita nel modo immediato.

Esempio:

```
VERIFY               Verifica il primo programma trovato sul nastro
VERIFY "LIBRO"       Ricerca il programma chiamato LIBRO sull'unità #1
                       e lo verifica
A$="LIBRO"           Come sopra
VERIFY A$
```

L'unità a dischetti

VERIFY"[*dr:*]*nome del file*",*dev*

Il programma residente in memoria è paragonato a quello nel file chiamato *nome del file* sul dischetto nel drive numero *dr*. Il parametro *dev* deve essere presente e, se non specificato altrimenti, avere un valore di 8. Se manca quest'ultimo parametro è assunto un valore di 1 e quindi selezionata l'unità a cassette.
Per verificare il programma trascritto per ultimo agite come segue:

```
VERIFY "*",8
```

Si dovrebbero verificare i programmi immediatamente dopo averli copiati. Quest'istruzione è eseguita quasi sempre nel modo immediato.

Esempio:

```
VERIFY "*",8          Verifica il programma appena trascritto

VERIFY "0:ZOO",8      Ricerca il programma chiamato ZOO nel drive 0 e
                        lo verifica
C$="0:ZOO"            Come sopra
VERIFY C$
```

WAIT

L'istruzione WAIT arresta l'esecuzione del programma fino a che una locazione di memoria acquisisce un valore definito.

Struttura:

WAIT *indirizzo,mask*[,*xor*]

dove:

mask è il valore di maschera di un byte
xor è il valore di maschera di un byte

L'istruzione WAIT opera come segue:

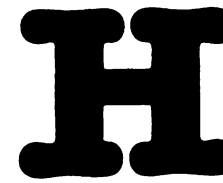
1. Legge il contenuto della locazione di memoria specificata.
2. Il valore ottenuto dalla fase 1 è sottoposto ad un'operazione OR esclusivo con *xor*, se presente. Questa fase non ha alcun effetto se *xor* non è specificato.

3. Il valore ottenuto al punto 2 è sottoposto a AND con il valore di maschera specificato.
4. Se il risultato è 0, WAIT ritorna al punto 1, rimanendo in un loop che trattiene il programma all'istruzione WAIT.
5. Se il risultato non è 0, l'esecuzione del programma continua con l'istruzione che segue WAIT.

Il tasto STOP non interrompe l'esecuzione dell'istruzione WAIT.

Appendice

Funzioni BASIC



Il C-64 può definire un gran numero di funzioni direttamente dal BASIC. Queste funzioni includono molte funzioni matematiche, istruzioni di formattazione del video e di manipolazione di stringhe. Sono elencate in ordine alfabetico.

ABS

ABS riporta il valore assoluto di un numero.

Struttura:

ABS(*arg*)

Esempio:

```
A=ABS(10)                          Fornisce il risultato A=10
A=ABS(-10)                         Fornisce il risultato A=10
PRINT ABS(X),ABS(Y),ABS(Z)
```

ASC

ASC riporta il numero di codice ASCII di un carattere specificato.

Struttura:

ASC(*carattere*)

Se la stringa è più lunga di un carattere, ASC riporta solamente il valore del primo carattere della stringa. L'argomento riportato è un numero e può essere impiegato in operazioni aritmetiche. I codici ASCII sono elencati nell'appendice E.

Esempio:

```
?ASC("A")     Stampa il valore ASCII di "A" che è 65
X=ASC("S")
?X            Stampa il valore ASCII di "S" che è 83
```

ATN

ATN riporta l'arcotangente dell'argomento.

Struttura:

ATN(*arg*)

ATN riporta il valore in radianti entro i limiti ±17.

Esempio:

```
A=ATN(AG)
?180π*ATN(A)
```

CHR$

CHR$ riporta il carattere corrispondente al codice ASCII specificato.

Struttura:

CHR$(*ASCII*)

CHR$ può essere usato per specificare caratteri che non è possibile rappresentare nelle stringhe come le virgolette o il ritorno a capo.

Esempio:

```
IF C$=CHR$(13) GOTO 10          Salta se C$ è un ritorno a capo (CHR$(13))

?CHR$(34):"HOHOHO":CHR$(34)     Stampa gli otto caratteri "HOHOHO" (dove i due CHR$(34) rappresentano le virgolette)
```

COS

COS riporta il coseno dell'argomento.

Struttura:

COS(*arg*)

EXP

EXP riporta il valore di e^{arg}. Il valore di e usato è 2,71828183.

Struttura:

EXP(*arg*)

arg deve avere un valore entro i limiti ±88,029691. Un numero più elevato del limite superiore darà un errore di OVERFLOW, mentre uno più basso del limite inferiore darà un risultato 0.

Esempio:

```
?EXP(0)              Stampa 1
?EXP(1)              Stampa 2,71828183
EV=EXP(2)            Fornisce il risultato EV=7,3890561
EB=EXP(50.24)        Fornisce il risultato EB=6,59105247E+21
?EXP(88.0296919)     Numero più elevato permesso, 1,70141183E+38
?EXP(-88.0296919)    Numero più basso permesso, 5,87747176E-39
?EXP(88.029692)      Fuori limite, messaggio di OVERFLOW
?EXP(-88.029692)     Fuori limite, riporta 0
```

FRE

FRE è una funzione del sistema che raccoglie tutti i byte di memoria liberi in un unico blocco (chiamato "raccolta dei rifiuti") e riporta il numero di byte disponibili.

Struttura:

FRE(*arg*)

arg è un argomento falso; può essere a stringa o numerico.
FRE può essere impiegato ovunque, ma è normalmente usato nell'istruzione PRINT nel modo immediato.

Esempio:

```
?FRE(1)
```
Esegue la "raccolta dei rifiuti" e stampa il numero di byte liberi

INT

INT riporta la parte intera di un numero, arrotondando per difetto.

Struttura:

INT(*arg*)

Per numeri positivi, INT equivale a troncare la parte decimale senza arrotondare, per quelli negativi a troncare la parte decimale ed aggiungere 1. Si noti che INT *non* trasforma un numero a virgola mobile (5 byte) in uno intero (2 byte).

Esempio:

```
A=INT(1.5)
```
Dà A=1

```
A=INT(-1.5)
```
Dà A=-2

```
X=INT(-0.1)
```
Dà X=-1

Attenzione: dato che i numeri a virgola mobile sono solo buone approssimazioni di numeri reali, un argomento potrebbe non dare il valore aspettato. Per esempio, si consideri il numero 3,89999999. La funzione INT(3,89999999) dà come risultato 3.

```
?INT(3.89999999)
3
```

LEFT$

LEFT$ riporta i caratteri più a sinistra di una stringa.

Struttura:

LEFT$(*arg*$,*byte*)

byte specifica il numero di caratteri da estrarre a sinistra dalla stringa di caratteri *arg*$.

Esempio:

```
?LEFT$("ARG",2)
```
Stampa AR

```
A$=LEFT$(B$,10)
```
Stampa i dieci caratteri più a sinistra della stringa B$

LEN

LEN riporta la lunghezza della stringa in argomento.

Struttura:

LEN(*arg*$)

LEN riporta un numero che rappresenta il numero di caratteri presenti nella stringa specificata da *arg*$.

Esempio:

```
?LEN("ABCDEF")
```
Riporta 6

```
N=LEN(C$+D$)
```
Riporta la somma dei caratteri nelle stringhe C$ e D$

LOG

LOG calcola il logaritmo naturale, o in base e. Il valore di e è 2,71828183.

Struttura:

LOG(*arg*)

Viene fornito il messaggio d'errore ILLEGAL QUANTITY ERROR se l'argomento è zero o negativo.

Esempio:

```
?LOG(1)              Stampa 0
A=LOG(10)            Fornisce il risultato A=2,30258509
A=LOG(1E6)           Fornisce il risultato A=13,8155106
A=LOG(X)/LOG(10)     Calcola il logaritmo in base 10
```

MID$

MID$ riporta qualsiasi porzione specificata di una stringa.

Struttura:

MID$(*data*$,*byte*$_1$,[,*byte*$_2$])

I due parametri *byte*$_1$ e *byte*$_2$ determinano la porzione della stringa da estrarre. I caratteri di una stringa sono numerati dalla sinistra verso destra, il primo carattere ha il numero 1. Il valore di *byte*$_1$ asporta tutti i caratteri fino alla posizione determinata da *byte*$_2$. Se *byte*$_2$ non è presente, verranno estratti tutti i caratteri fino alla fine della stringa. È presentato un messaggio ILLEGAL QUANTITY ERROR se un parametro è fuori portata.

Esempio:

```
?MID$("ABCDE",2,1)   Stampa B
?MID$("ABCDE",3,2)   Stampa CD
?MID$("ABCDE",3)     Stampa CDE
```

PEEK

PEEK riporta il contenuto di una locazione di memoria specificata. PEEK è l'inverso di POKE.

Struttura:

PEEK(*indirizzo*)

Esempio:

```
?PEEK(1)              Stampa il contenuto della locazione di memoria 1
A=PEEK(20000)
```

POS

POS riporta il numero della colonna in cui si trova il cursore.

Struttura:

POS(*arg*)

arg è un dato inutile, non viene adoperato e quindi può avere qualunque valore. POS riporta la posizione attuale del cursore. Le posizioni del cursore sono numerate da sinistra partendo da 0. Ricordate che la logica di programma tratta righe di 80 caratteri mentre il video del C-64 è di soli 40 caratteri. Se la logica del programma nel C-64 sta elaborando un carattere nella seconda metà della riga virtuale, POS riporta un valore da 41 a 80. Per mezzo del concatenamento possono essere generate variabili a stringa fino a 255 caratteri, per cui se il programma sta elaborando una stringa molto lunga, la funzione POS riporta un valore tra 0 e 255.

Esempio:

```
?POS(1)               All'inizio di una riga, riporta 0
?"ABCABC";POS(1)      Dà come risultato 6
```

RIGHT$

RIGHT$ riporta i caratteri più a destra di una stringa.

Struttura:

RIGHT$(*arg*$,*byte*)

byte identifica il numero di caratteri più a destra da estrarre dalla stringa specificata da *arg*$.

Esempio:

```
RIGHT$("ARG",2)          Riporta RG

MM$=RIGHT$(X$+"#",5)  A MM$ vengono assegnati gli ultimi 4 caratteri di
                         X$, più il carattere #
```

RND

RND genera numeri casuali tra 0 e 1.

Struttura:

RND(*arg n*)	Riporta un numero casuale
RND(−*arg n*)	Memorizza un nuovo valore di seme

Esempio:

```
A=RND(-1)        Memorizza un nuovo seme basato su un valore -1

A=RND(1)         Riporta il numero casuale successivo nella sequenza
```

Un argomento zero è trattato come caso speciale; non memorizza un nuovo seme, nè riporta un numero casuale. RND(0) usa il valore attualmente presente in TI per introdurre un ulteriore elemento di casualità. Un seme pseudo-casuale è memorizzato dalla seguente funzione:

```
RND(-TI)         Memorizza un seme pseudo-casuale.
```

RND(0) può essere usato per memorizzare un seme più propriamente a caso con:

```
RND(-RND(0))     Memorizza un seme casuale
```

Per una trattazione completa sui numeri casuali, vedere il capitolo 4.

SGN

SGN determina se un numero è positivo o negativo o zero.

Struttura:

SGN(*arg*)

La funzione SGN riporta +1 se il numero è positivo, 0 se zero e −1 se negativo.

Esempio:

```
?SGN(-6)        Riporta -1
?SGN(0)         Riporta 0
?SGN(44)        Riporta 1

IF A>C THEN SA=SGN(X)
IF SGN(M)>=0 THEN PRINT "NUMERO POSITIVO"
```

SIN

SIN calcola il seno dell'argomento.

Struttura:

SIN(*arg n*)

Esempio:

```
A=SIN(AG)
?SIN(45*π/180)    Calcola il seno di 45°
```

SPC

SPC muove il cursore a destra del numero di posizioni specificate.

Struttura:

SPC(*byte*)

La funzione SPC è usata in istruzioni PRINT per muovere il cursore di un certo numero di posizioni a destra, senza modificare i caratteri eventualmente incontrati. La funzione muove il cursore a destra a partire dalla posizione in cui si trova quando viene eseguito il comando, al contrario di TAB che lo muove fino ad una posizione fissa misurata dall'estrema sinistra del video (vedi TAB per gli esempi).

SQR

SQR calcola la radice quadrata di un numero positivo. Un numero negativo darebbe luogo ad un messaggio d'errore.

Struttura:

SQR(*arg*)

Esempio:

```
A=SQR(4)          Riporta A=2
A=SQR(4.84)       Riporta A=2,2
?SQR(144E30)      Riporta 1,2E+16
```

ST

ST riporta il valore attuale della variabile di stato I/O. Questo stato è fissato a diversi valori a seconda dei risultati dell'ultima operazione di input/output.

Struttura:

ST

I valori di ST sono elencati nella tabella 8.2.
Lo stato dovrebbe essere controllato dopo l'esecuzione di qualsiasi istruzione che accede a un dispositivo esterno.

Esempio:

```
10 IF ST<>0 GOTO 500          Salta se è presente un qualsiasi errore

50 IF ST=4 THEN ?"BLOCCO CORTO"
```

STR$

STR$ riporta l'equivalente di stringa di un argomento numerico.

Struttura:

STR$(*arg n*)

STR$ riporta la stringa di caratteri equivalenti al numero generato risolvendo *arg n.*

Esempio:

```
A$=STR$(14.6)          Riporta 14,6
?A$

?STR$(1E2)             Riporta 100
?STR$(1E10)            Riporta 1E+10
```

SYS

SYS è una funzione del sistema che trasferisce il controllo di un programma ad un sottosistema indipendente.

Struttura:

SYS(*mem adr*)

mem adr è l'indirizzo del registro dove deve iniziare l'esecuzione del sottosistema. Il valore deve essere compreso tra 0 e 65535.

TAB

TAB muove il cursore a destra ad una posizione specificata di colonna.

Struttura:

TAB(*arg*)

TAB muove il cursore alla posizione n+1, dove n è il numero ottenuto risolvendo *arg.*

Esempio:

```
?"QUARK";SPC(10);"W"
QUARK          W
?"QUARK";TAB(10);"W"
QUARK      W
```

Questi due esempi mostrano la differenza tra TAB e SPC. SPC salta 10 posizioni dall'ultima posizione del cursore, mentre TAB salta alla 10+1 posizione sulla riga

TAN

TAN calcola la tangente dell'argomento.

Struttura:

TAN(*arg*)

Esempio:

```
?TAN(3.2)                    Riporta 0,0584738547
XY(1)=TAN(180*π/180)
```

TI,TI$

TI e TI$ rappresentano due variabili di sistema legate al clock interno.

Struttura:

TI	Numero di jiffy (1 jiffy=1/60 di secondo) dall'accensione
TI$	Stringa dell'ora del giorno

Esempio:

```
?TI
23454
?TI$
081523
```

USR

USR è una funzione del sistema che passa un parametro ad una subroutine scritta dall'utente in linguaggio macchina il cui indirizzo è contenuto nelle locazioni di memoria 1 e 2. USR preleva poi un parametro di ritorno da questa subroutine.

Struttura:

USR(*arg*)

VAL

VAL riporta l'equivalente numerico di una stringa.

Struttura:

VAL(*arg*$)

Il numero riportato da VAL può essere usato nei calcoli aritmetici. VAL trasforma l'argomento a stringa scartando per prima cosa gli eventuali caratteri nulli all'inizio della stringa. Se il primo carattere non nullo non è una cifra (0-9), l'argomento è considerato come 0; nel caso in cui il primo carattere sia una cifra, VAL trasforma la stringa in un numero reale. Se successivamente trova un carattere non numerico, sospende il procedimento in modo che il valore riportato sia l'equivalente numerico della stringa fino al primo carattere non numerico trovato.

Esempio:

```
A=VAL("123")

NN=VAL(B$)
```

Indice analitico

Nella stessa serie:

— C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*

— T. Woods, *L'Assembler per lo ZX Spectrum*

— R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*

Di prossima pubblicazione:

— G. Bishop, *Progetti hardware con lo ZX Spectrum*

— H. Mullish e D. Kruger, *Il BASIC Applesoft*

— N. Williams, *Inventa i tuoi giochi con lo ZX Spectrum*

— H. Peckham, *Il BASIC pratico per l'IBM-PC*

— H. Peckham, *Il BASIC pratico per il Commodore 64*

— S. Nichols, *Tecniche avanzate in Assembler per giochi veloci con lo ZX Spectrum*

— K. Skier, *L'Assembler per il VIC 20 e il Commodore 64*

— S. Kamins e M. Waite, *Programmate meglio il vostro Apple*

Il Commodore 64 è uno dei più diffusi home computer, caratterizzato da ottime prestazioni e da una grande praticità d'uso.
Per poterne però sfruttare appieno le possibilità è importante approfondire le proprie conoscenze, in particolare sul BASIC, sulla grafica e sulla gestione delle periferiche.
È questo il preciso compito della **Guida al Commodore 64**: partendo dal primo approccio con la macchina ancora imballata, il manuale aiuta a risolvere, per gradi, tutti i problemi che si possono presentare, portando l'utente del C-64 ad una completa conoscenza del suo sistema.
Gli argomenti sono così suddivisi negli otto capitoli e nelle appendici del volume:

- introduzione generale sulle apparecchiature Commodore
- modi operativi del C-64, diretti e programmati
- introduzione alla programmazione BASIC
- programmazione BASIC avanzata
- uso del joystick e degli altri comandi per i giochi
- grafica
- suono
- unità periferiche
- architettura dei sistemi
- uso della memoria
- descrizione dettagliata delle porte di I/O
- tavole di conversione e tabelle di codifica ASCII e C-64
- riepilogo delle istruzioni e delle funzioni BASIC

Lire 36 000
(IVA 2% inclusa)

ISBN 88 7700 001 5

Heilborn - Talbott **Guida al COMMODORE 64**

McGraw-Hill