COMMODORE 64

Guida di riferimento per il programmatore

Edizione 1983

APPENDICE

INTRODUZIONE

La GUIDA AL COMMODORE 64 e' stata sviluppata come strumento di lavoro e come guida di riferimento per coloro che vogliano sfruttare al massimo le notevoli possibilita' offerte dal COMMODORE 64. Questo Manuale contiene le informazioni necessarie alla programmazione, dall'esempio piu' semplice a quello piu' complesso. La GUIDA e' stata realizzata in modo tale che chiunque, dal programmatore dilettante in BASIC, al professionista esperto nel Linguaggio Macchina del 6502, possa trarre delle informazioni utili per sviluppare i propri programmi. Allo stesso tempo, questo libro mostra quanto realmente sia versatile il COMMODORE 64.

Lo scopo di questa GUIDA non e' l'insegnamento del linguaggio BASIC o del Linguaggio Macchina del 6502. Vi si trova comunque un ricco Glossario di termini ed una introduzione "quasi-guidata" a molte parti del libro. Se non si e' gia' in possesso di una discreta conoscenza del BASIC, e' consigliabile esaminare attentamente la Guida Utente del COMMODORE 64 di cui e' corredato il Computer. Tale Guida fornisce una facile introduzione al linguaggio BASIC. Se cio' nonostante si incontrassero ancora delle difficolta' nell'uso del BASIC, si rimanda all'esame della Bibliografia posta al termine di questo Volume (o all'Appendice N della Guida Utente).

La GUIDA AL COMMODORE 64 non e' altro che un riferimento; come per gran parte dei Manuali di Riferimento, la capacita' di applicare in modo creativo le sue informazioni dipende in effetti dal grado di conoscenza che si ha dell'argomento. In altre parole, un programmatore alle prime armi non e' in grado di usare tutti i dati e le informazioni contenute in questa GUIDA, finche' non abbia aumentato la propria esperienza di programmatore.

Questo libro puo' servire per ricavare un notevole bagaglio di preziose informazioni riguardanti la programmazione, scritte in un Italiano scorrevole e comprensibile, con la spiegazione dei termini tipici della programmazione. Il programmatore professionista vi potra' trovare, invece, tutte quelle informazioni necessarie per sfruttare al meglio le capacita' del COMMODORE 64.

CHE COSA CONTIENE

* Il "dizionario BASIC" include i comandi, le istruzioni e le funzioni del CBM 64 BASIC elencati in ordine alfabetico. E' stato realizzato un prontuario contenente tutti i termini e le relative abbreviazioni. Questo e' seguito da una sezione dedicata alla descrizione piu' dettagliata di ciascun termine, seguita a sua volta da esempi di programmi in BASIC che ne illustrano il modo di operare.
* Se si ha bisogno di un'Introduzione all'uso del Linguaggio Macchina con i programmi in BASIC, un aiuto iniziale viene fornito dalle illustrazioni dedicate ai principianti.
* Il KERNAL e' una potente caratteristica di tutti i computer prodotti dalla Commodore. Esso assicura che i programmi scritti oggi possano essere utilizzati anche sui Commodore di domani.
* La sezione riguardante la programmazione di I/O offre l'opportunita' di usare il computer al limite. Questa sezione descrive come collegare ed usare Penna Ottica, "Joystick", Unita' Disco, Stampanti e Dispositivi di Telecomunicazione chiamati Modem.
* Si puo' entrare nel mondo delle ANIMAZIONI, dei caratteri programmabili, della grafica ad alta risoluzione e delle piu' perfezionate e dettagliate immagini animate dell'industria dei microcomputer.
* Si puo' entrare anche nel mondo della sintesi musicale, creando canzoni ed effetti sonori con il miglior sintetizzatore disponibile su qualsiasi altro Personal Computer.
* Ai programmatori esperti, la sezione riguardante i linguaggi non residenti mette a disposizione le informazioni sulle capacita' del COMMODORE 64 di far "girare" linguaggi ad alto livello sotto CP/M (*). Questo in aggiunta al BASIC.

La Guida al COMMODORE 64 deve essere quindi considerata come un utile strumento d'aiuto, in grado di allietare le ore spese per la programmazione.

(*) CP/M e' un marchio registrato della Digital Research, Inc.)

COME USARE QUESTA GUIDA

In questo manuale vengono usate alcune notazioni convenzionali per descrivere la sintassi (la struttura delle frasi) dei comandi o delle istruzioni BASIC, e per illustrare, di ciascuna parola chiave del BASIC, sia la parte obbligatoria che quella facoltativa. Le regole per interpretare la sintassi delle istruzioni sono le seguenti:

1. Le parole chiave del BASIC sono visualizzate in lettere maiuscole. Esse devono comparire seguendo l'ordine dato nell'istruzione, inserendole esattamente come sono illustrate.
2. Le voci comprese fra i doppi apici ("") indicano dati variabili da introdurre. Sia i doppi apici che i dati al loro interno devono comparire come mostrato in ciascuna istruzione.

3. Le voci comprese tra parentesi quadrate ([]) indicano un parametro facoltativo delle istruzioni. Un parametro e' una limitazione o un aggettivo addizionale per le istruzioni. Ad ogni parametro facoltativo e' associato un dato. Inoltre, l'ellissi (...) indica che una voce facoltativa puo' essere ripetuta tante volte quante lo consente la linea di programma.
4. Se una voce fra parentesi quadrate e' SOTTOLINEATA, significa che DEVONO ESSERE USATI quei determinati caratteri nei parametri facoltativi inseriti esattamente come sono illustrati.
5. Le voci comprese fra parentesi acute (<>) indicano dati variabili che devono essere forniti dall'Utente. Mentre la barra (/)indica che si deve compiere una scelta fra due opzioni mutuamente esclusive.

ESEMPIO DI FORMATO DI SINTASSI:

```
OPEN<numero-file>,<dispositivo>[,<indirizzo>],
    "[<drive>:<nome-file>],[<modo>]"
```

ESEMPI DI ISTRUZIONI EFFETTIVE:

```
10 OPEN 2,8,6,"0:STOCK FOLIO,S,W"
20 OPEN 1,1,2,"CHECKBOOK"
30 OPEN 3,4
```

Quando si applicano le regole sintattiche ad una situazione pratica, la sequenza dei parametri nelle istruzioni puo' non essere quella mostrata negli esempi sintattici. Gli esempi non pretendono di mostrare tutte le possibili sequenze, ma soltanto i parametri necessari e quelli facoltativi.

Gli esempi di programmi di questo libro sono illustrati con le parole e gli operatori separati da spazi bianchi per una maggiore leggibilita'. Di solito il BASIC non richiede tale separazione, a meno che la loro omissione non possa creare ambiguita' o scorrettezze nella sintassi.

Di seguito sono illustrati alcuni esempi dei simboli usati per diversi parametri delle istruzioni nei capitoli seguenti. Tale lista non ha la pretesa di fornire ogni possibilita', ma di dare una miglior comprensione di come sono presentati gli esempi sintattici.

SIMBOLO	ESEMPIO	DESCRIZIONE
<numero-file>	50	Numero logico di un file
<dispositivo>	4	Numero di dispositivo hardware
<indirizzo>	15	Numero di indirizzo secondario di un dispositivo sul bus seriale
<drive>	0	Numero di disk drive fisico
<nome-file>	"TEST.DATA"	Nome di un file dati o programma
<costante>	"ABCDEFG"	Dato letterale fornito dal programmatore
<variabile>	X145	Qualunque nome o costante dei dati variabili del BASIC
<stringa>	AB$	Uso della variabile di tipo stringa richiesta
<numero>	12345	Uso della variabile di tipo numerico richiesta
<numero-linea>	1000	Numero di linea del programma attuale
<numerico>	1.5E4	Variabile intera o reale

GUIDA ALLE APPLICAZIONI DEL COMMODORE 64

Quando per la prima volta si e' pensato all'acquisto di un computer, ci si e' probabilmente chiesto: "Ora che mi posso permettere di comprare un computer, che cosa ci posso fare?". Il bello del COMMODORE 64 e' che puo' realizzare tutto cio' che si vuole! Puo' essere utilizzato per calcolare e registrare sia il bilancio di casa che quello del lavoro, come elaboratore di testi, come compagno in tutta una serie di giochi d'azione, si puo' farlo cantare, si possono creare addirittura cartoni animati personali, ed altro ancora. Il bello dell'avere un COMMODORE 64 e' che, anche se facesse soltanto una delle cose elencate piu' avanti, i soldi sarebbero stati comunque spesi bene. Ma il 64 e' un computer completo, che fa proprio TUTTO quanto elencato, e quindi abbastanza!

Oltre a quanto detto si possono trovare numerose altre idee creative e pratiche iscrivendosi ad un Club locale per utenti Commodore ed abbonandosi alle riviste Commodore.

APPLICAZIONE	COMMENTI/REQUISITI
GIOCHI D'AZIONE	Si possono trovare i veri videogiochi Bally Midway come Omega Race, Gorf e Wizard of War, oppure i giochi "gioca ed impara" come Math Teacher I, Home Babysitter e Commodore Artist.
PUBBLICITA' COMMERCIALE	Collegando il COMMODORE 64 alla TV ed esponendolo in vetrina con un messaggio musicale animato, si puo' ottenere un forte richiamo per il negozio.
ANIMAZIONE	L'animazione grafica del COMMODORE 64 consente di creare veri cartoni animati ad otto differenti livelli, permettendo alle figure di muoversi liberamente.
BABYSITTER	La cartuccia HOME BABYSITTER del COMMODORE 64 tiene occupati per ore i bambini, insegnando nello stesso tempo l'associazione alfabeto/tastiera. Essa insegna anche particolari rapporti e concetti istruttivi.
PROGRAMMAZIONE IN BASIC	La GUIDA PER L'UTENTE DEL COMMODORE 64 e la serie di manuali e nastri IMPARA A PROGRAMMARE DA SOLO offrono un valido punto di partenza.
BUSINESS SPREADSHEET	Il COMMODORE 64 offre la Serie "Easy" di supporto al lavoro professionale che comprende il piu' potente WORD PROCESSOR ed il piu' vasto "spreadsheet" (foglio elettronico) disponibile per qualunque altro Personal Computer.
COMUNICAZIONI	Il COMMODORE 64 consente di entrare nell'affascinante mondo del collegamento in rete dei calcolatori. Se si collega un VICMODEM al

COMMODORE 64, si puo' comunicare con altri proprietari di computer in tutto il mondo.

COMPOSIZIONE DI CANZONI

Il COMMODORE 64 e' dotato del piu' sofisticato sintetizzatore musicale disponibile su qualsiasi altro computer. Possiede tre voci completamente programmabili, nove ottave e quattro forme d'onda controllabili. Sono a disposizione le Commodore Music Cartridges ed i Manuali Commodore Music, che aiutano a creare o riprodurre tutti i tipi di musica ed effetti sonori.

CP/M(*)

La Commodore mette a disposizione una cartuccia di facile montaggio per l'utilizzo del CP/M (*) e del software relativo.

(*) CP/M e' un marchio registrato della Digital Research, Inc.

ESERCIZI DI ABILITA'

Il coordinamento mano/occhio, la destrezza manuale sono sollecitati dai numerosi giochi Commodore... tra cui "Jupiter Lander" e quelli di simulazione della guida notturna.

EDUCATIVO

Poiche', di per se' stesso, lavorare al computer e' educativo, il COMMODORE Educational Resource Book contiene le informazioni generali sugli usi didattici dei computer. Sono disponibili inoltre diverse cartucce didattiche realizzate allo scopo di insegnare un po' di tutto, dalla musica alla matematica, dalla astronomia all'arte.

LINGUA STRANIERA

Il set di caratteri programmabili del COMMODORE 64 consente di sostituire il set di caratteri standard con dei caratteri di una lingua straniera definiti dall'Utente.

GRAFICA E ARTE

Oltre alla Grafica Animata menzionata piu' sopra, il COMMODORE 64 permette di tracciare disegni ad alta risoluzione e a piu' colori, caratteri programmabili e le combinazioni di tutti i differenti modi di visualizzazione della grafica e dei caratteri.

STRUMENTO DI CONTROLLO

Il COMMODORE 64 ha una Porta Seriale, una Porta RS-232 ed una Porta Utente, per essere utilizzato in tutta una serie di speciali applicazioni industriali. Come optional, e' inoltre disponibile una cartuccia IEEE/488.

DIARI E SCRITTURA CREATIVA

Il COMMODORE 64 offrira' presto un eccezionale sistema di scrittura che uguaglia o supera la qualita' e la flessibilita' dei piu' costosi sistemi di scrittura disponibili. Ovviamente, si possono memorizzare le informazioni sia sul Disk Drive 1541 sia su registratore Datassette (TM), stampandole mediante una VICPRINTER o un PLOTTER.

CONTROLLO DELLA PENNA OTTICA	Le applicazioni che richiedono l'uso di una Penna Ottica si possono realizzare con una qualsiasi Penna Ottica che sia compatibile con il connettore della Porta Giochi (Game-Port) del COMMODORE 64.
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA	La GUIDA AL COMMODORE 64 include una sezione sul Linguaggio Macchina, ed una sezione riguardante l'interfacciamento fra BASIC e codice macchina. Per uno studio piu' approfondito e' disponibile una bibliografia.
STAMPA DI MODULI E DI STIPENDI	Il COMMODORE 64 puo' essere programmato per trattare una quantita' di problemi commerciali basati sulle registrazioni a giornale. Lettere maiuscole/minuscole in unione con la grafica "Business Form" del COMMODORE 64 facilitano il disegno di moduli che successivamente possono essere stampati.
STAMPA	Il COMMODORE 64 puo' interfacciarsi con un grande numero sia di stampanti ad aghi e "letter quality", sia di plotter.
RICETTE	Tramite il COMMODORE 64 si possono registrare le ricette preferite su unita' di memoria a disco o a cassetta risolvendo cosi' il problema dei fogli disordinati, che spesso vengono persi quando se ne avrebbe piu' bisogno.
SIMULAZIONE	La simulazione per mezzo del computer consente di condurre esperimenti costosi o pericolosi ad un costo e rischio minimi.
SPORT	Source (TM) e Compuserve (TM) offrono entrambi informazioni sportive ottenibili sul COMMODORE 64 impiegando un VICMODEM.
QUOTAZIONI DI BORSA	Con un VICMODEM, ed iscrivendosi ad una rete di servizi adatta, il COMMODORE 64 diventa una telescrivente privata.

Queste sono solo alcune applicazioni per il COMMODORE 64. Come si puo' comprendere, per lavoro o per divertimento, a casa, a scuola o in ufficio, il COMMODORE 64 offre una soluzione pratica per qualsiasi tipo di necessita'. La Commodore tiene a far sapere che la sua assistenza agli Utenti INIZIA con l'acquisto di un computer Commodore. Ecco perche' si sono realizzate due pubblicazioni contenenti informazioni Commodore provenienti da tutto il mondo (in Canada e negli Stati Uniti esiste anche la possibilita' di collegamento da costa a costa ad una rete "bidirezionale" di "Computer Information")

Inoltre, incoraggiamo vivamente e sosteniamo in tutto il mondo la crescita dei Club di Utenti Commodore. Essi sono un'ottima sorgente di informazioni per ogni proprietario di computer Commodore, dal principiante fino al piu' esperto. Le riviste descritte piu' avanti contengono le informazioni piu' aggiornate per entrare in contatto con il Club di Utenti piu' vicino.

Infine, il Rivenditore Commodore locale e' un'utile sorgente di assistenza ed informazione Commodore.

POWER/PLAY
The Home Computer Magazine

Quando si tratta di divertimento, di istruzione a domicilio, di applicazioni pratiche casalinghe, **POWER/PLAY** e' la prima sorgente di informazioni per gli utenti "casalinghi" Commodore. Scoprire dove si trovano i Club di Utenti piu' vicini e cosa stanno facendo, venire a conoscenza di software, giochi, tecniche di programmazione, telecomunicazioni, e novita'. **POWER/PLAY** e' il filo diretto personale con altri Utenti Commodore, centri di sviluppo esterni di software e hardware e la Commodore stessa. Pubblicazione trimestrale. Solo $10.00 per un anno di eccitante programmazione fra le mura di casa vostra.

COMMODORE
The Microcomputer Magazine

Ampiamente letto da insegnanti, uomini d'affari e studenti, cosi' come dai "computeristi", Commodore Magazine e' il veicolo principale per la diffusione di informazioni esclusive sull'uso piu' tecnico dei sistemi Commodore. Articoli pubblicati regolarmente su ogni numero si occupano di affari, scienza e didattica, suggerimenti per la programmazione, "riassunti da un taccuino tecnico", e molte altre caratteristiche di particolare interesse per chiunque usi o sia in procinto di acquistare un sistema Commodore per applicazioni commerciali, scientifiche o didattiche. COMMODORE e' il complemento ideale di **POWER/PLAY** .

Pubblicazione bimestrale. Abbonamento annuale: $15.00.

E PER ULTERIORI INFORMAZIONI...
...RIVOLGERSI ALLA NOSTRA RIVISTA SENZA CARTA

COMMODORE INFORMATION NETWORK

Ecco la rivista del futuro. Come supplemento ed ampliamento alle riviste **POWER/ PLAY**. e COMMODORE, la COMMODORE INFORMATION NETWORK - la nostra rivista "senza carta" - e' ora disponibile via telefono (negli Stati Uniti) per i computer Commodore usando un modem.

Entrando in uno dei nostri Computer Club, si puo' essere aiutati a risolvere un problema di computer, si puo' "parlare" con altri amici Commodore, oppure ottenere informazioni "fresche" sui nuovi prodotti e sulle risorse di software ed educative. In poco tempo si puo' essere addirittura in grado di risparmiarsi la fatica di digitare i programmi trovati su **POWER/PLAY** o COMMODORE, prelevandoli direttamente dalla Rete Informativa (un nuovo servizio per l'Utenza progettato per l'inizio del 1983). La cosa migliore e' che le risposte sono presenti anche prima che vengano formulate le domande (niente male, vero ?).

L'allacciamento alla nostra rivista elettronica richiede solamente un modem e l'abbonamento a Compuserve(TM), una delle piu' vaste reti di telecomunicazione degli Stati Uniti [in ogni pacchetto VICMODEM e' incluso un abbonamento annuale GRATUITO a Compuserve(TM)].

Per entrare in contatto con la Banca Dati Compuserve(TM) e' sufficiente comporre il numero telefonico e predisporre il modem. Al comparire sullo schermo del testo video Compuserve(TM), battere da tastiera G CBM. Quando compare l'indice del COMMODORE INFORMATION NETWORK (menu'), si puo' scegliere uno dei 16 argomenti, mettersi a proprio agio e godersi la rivista senza carta.

Ulteriori informazioni sono disponibili presso i Rivenditori Commodore, oppure contattando il Servizio Clienti Compuserve(TM) (800-848-8990; per l'Ohio 614-457-8600).

COMMODORE INFORMATION NETWORK

Descrizione Menu' Principale	Rivenditori Commodore
Codici Accesso Diretto	Risorse Educative
Comandi Speciali	Associazioni Utenti
Domande Utenti	Descrizioni
Public Bulletin Board	Domande e Risposte
Riviste e Aggiornamenti	Suggerimenti Software
Prodotti Annunciati	Suggerimenti Tecnici
Commodore News Direct	Directory Descriptions

CAPITOLO 1

regole della programmazione in BASIC

- Introduzione
- Codici dello Schermo Video (Insieme dei Caratteri BASIC)
- Numeri e variabili della Programmazione
- Espressioni ed Operatori
- Tecniche di Programmazione

INTRODUZIONE

Questo Capitolo tratta di come il BASIC memorizza e manipola i dati. Gli argomenti comprendono:

1) Un breve accenno alle componenti ed alle funzioni del Sistema Operativo e all'insieme di caratteri utilizzato dal COMMODORE 64.
2) La formazione di costanti e variabili. Quali sono i tipi di variabili. E come sono memorizzate le variabili e le costanti.
3) Le regole di calcolo aritmetico, test relazionali, trattamento di stringhe ed operazioni logiche. Sono comprese anche le regole per la composizione delle espressioni e le conversioni dei dati necessarie quando si usa il BASIC con diversi tipi di dati.

CODICI DELLO SCHERMO VIDEO
(SIEME DEI CARATTERI BASIC

IL SISTEMA OPERATIVO

Il Sistema Operativo risiede nei Circuiti di Memoria a Sola Lettura (ROM) ed e' una combinazione di tre moduli programma separati ma interdipendenti:

1) L'Interprete BASIC
2) Il KERNAL
3) L'Editor Video

1) All' Interprete Basic e' demandata la funzione di analizzare la sintassi delle istruzioni BASIC, e quella di realizzare i calcoli e/o il trattamento dei dati che gli vengono richiesti. L'Interprete BASIC ha un vocabolario di 65 "parole chiave" che assumono un particolare significato. I caratteri alfabetici maiuscoli e minuscoli e le cifre da 0 a 9 vengono usati per comporre sia parole chiave che nomi di variabili. Per l'Interprete hanno significato anche alcuni caratteri della punteggiatura e simboli speciali. La tabella 1.1 illustra i caratteri speciali ed il loro uso.
) Il KERNAL tratta la maggior parte dei processi di controllo dei livelli di interruzione del sistema (per una trattazione dettagliata dei livelli di interruzione si rimanda al Capitolo 5). Il KERNAL controlla inoltre l'ingresso e l'uscita effettivi dei dati.
3) L'Editor Video controlla l'output diretto allo schermo video (televisore), e l'editazione del testo di un programma BASIC. Inoltre, l'Editor Video esamina l'input proveniente da tastiera cosi' da poter stabilire se i caratteri introdotti debbano essere eseguiti immediatamente, oppure inoltrati all'Interprete BASIC.

CARATTERE	NOME E DESCRIZIONE
	BLANK - Separa le parole chiave e i nomi delle variabili
;	PUNTO E VIRGOLA - Usato nelle liste di variabili per formattare l' output
=	UGUALE - Assegnazione di un valore ed impostazione di relazioni
+	PIU' - Addizione aritmetica o concatenazione di stringhe
-	MENO - Sottrazione aritmetica, operatore unario (-1)
*	ASTERISCO - Moltiplicazione aritmetica
/	BARRA - Divisione aritmetica
↑	FRECCIA IN ALTO - Elevamento aritmetico a potenza
(	PARENTESI APERTA - Valutazioni e funzioni di una espressione
)	PARENTESI CHIUSA - Valutazioni e funzioni di una espressione
%	PERCENTUALE - Dichiara un nome di variabile come intero
#	POUND - Precede il numero logico di un file in istruzioni di input/output
$	DOLLARO - Dichiara un nome di variabile come stringa
,	VIRGOLA - Usato nelle liste di variabili per formattare l'output; separa anche i parametri di comando
.	PUNTO - Punto decimale nelle costanti in virgola mobile
"	VIRGOLETTE - Racchiudono costanti stringa
:	DUE PUNTI - Separano piu' istruzioni BASIC su una linea
?	PUNTO INTERROGATIVO - Abbreviazione per la parola chiave PRINT
<	MINORE - Usato nei test relazionali
>	MAGGIORE - >> >> >>
π	PI GRECO - Costante numerica pari a 3.141592654

Tabella 1.1 - Insieme dei Caratteri CBM BASIC

Il Sistema Operativo fornisce due modi di operare in BASIC:

1) Modo DIRETTO
2) Modo PROGRAMMA

1) Usando il Modo DIRETTO le istruzioni BASIC non hanno il numero di linea all'inizio dell'istruzione. Esse vengono eseguite non appena viene premuto il tasto RETURN.
2) Il Modo PROGRAMMA e' quello usato per far girare i programmi. Usando questo modo, tutte le istruzioni BASIC devono avere all'inizio i numeri di linea. Si possono avere piu' istruzioni BASIC su una stessa linea, ma il loro numero e' limitato dal fatto che una linea logica di video non puo' avere piu' di 80 caratteri. Cio' vuol dire che se si supera il limite di 80 caratteri, occorre riportare l'istruzione BASIC che eccede tale lunghezza su una nuova linea, con un nuovo numero di linea.

Il COMMODORE 64 possiede due insiemi completi di caratteri che si possono usare sia da tastiera che da programma.
L'insieme i comprende le lettere alfabetiche maiuscole ed i numeri da 0 a 9, che possono essere battuti senza ricorrere al tasto SHIFT Tenendo premuto questo tasto durante la battitura, si utilizzano i caratteri grafici indicati sulla DESTRA della parte verticale dei

tasti. Tenendo premuto C= (tasto Commodore) durante la battitura, si utilizzano invece i caratteri grafici indicati sulla SINISTRA della parte verticale dei tasti. Tenendo premuto il tasto SHIFT ed un qualunque tasto sprovvisto di simboli grafici, si otterra' il simbolo indicato nella parte superiore del tasto.

L'insieme 2 comprende le lettere alfabetiche minuscole ed i numeri da 0 a 9, che possono essere battuti senza ricorrere al tasto SHIFT Tenendo premuto questo tasto durante la battitura, si utilizzano le lettere alfabetiche maiuscole. Anche in questo caso, tenendo premuto il tasto C= (tasto Commodore), si utilizzano i simboli grafici indicati sulla SINISTRA della parte verticale dei tasti, mentre tenendo premuto il tasto SHIFT si utilizzano i simboli riportati nella parte superiore dei tasti sprovvisti di caratteri grafici.

Per passare da un insieme di caratteri all'altro, basta premere C= (tasto Commodore) e SHIFT contemporaneamente.

NUMERI E VARIABILI DELLA PROGRAMMAZIONE
COSTANTI INTERE, REALI E STRINGA

Le COSTANTI sono i valori che si inseriscono nelle istruzioni BASIC. Il BASIC usa questi valori per descrivere i dati durante l'esecuzione dell'istruzione. Il BASIC CBM puo' riconoscere ed elaborare tre tipi di dati:

1) NUMERI INTERI
2) NUMERI REALI
3) STRINGHE

Le COSTANTI INTERE sono l'insieme dei numeri (senza punti decimali); devono essere comprese fra -32768 e +32767. LE COSTANTI INTERE NON DEVONO AVERE PUNTI DECIMALI O VIRGOLE FRA LE CIFRE. Se il segno piu' (+) e' tralasciato, la costante e' assunta come numero positivo. Gli zeri che precedono una costante vengono ignorati e non dovrebbero essere usati poiche' sprecano della memoria e rallentano il programma. Comunque, non provocano un errore. Gli interi sono memorizzati come numeri binari di due byte. Alcuni esempi di costanti intere sono:

```
-12
8765
-32768
+44
0
-32767
```

NOTA: NON inserire virgole all'interno del numero. Ad esempio, battere sempre 32000 anziche' 32,000. Se si inserisce una virgola all'interno di un numero si ottiene il messaggio di errore ?SYNTAX ERROR.

Le COSTANTI REALI sono numeri positivi o negativi, compresi i frazionari. Le parti decimali di un numero si possono visualizzare adoperando il punto decimale. Ancora una volta si ricordi che le virgole NON devono essere usate tra i numeri. Se il segno piu' (+) e' omesso, il COMMODORE 64 assume il numero come positivo. Se si tralascia il punto decimale, il computer considera il punto come se si trovasse dopo l'ultima cifra del numero. E come per gli interi, gli

zeri che precedono una costante vengono ignorati. Le costanti reali si possono utilizzare in due modi:

1) NOTAZIONE SEMPLICE
2) NOTAZIONE SCIENTIFICA

Le costanti reali vengono visualizzate sullo schermo fino alla nona cifra. Queste cifre possono descrivere valori compresi fra -999999999 e +999999999. Se si inseriscono piu' di nove cifre, il numero viene arrotondato alla decima cifra. Se questa cifra e' un numero maggiore o uguale a 5, avviene un arrotondamento per eccesso, altrimenti avviene un arrotondamento per difetto. Cio' puo' essere importante per i totali finali di alcuni numeri con cui ci si puo' trovare a lavorare. I numeri reali (memorizzati usando 5 bytes di memoria) vengono trattati nei calcoli usando una precisione di dieci cifre. Tuttavia, al momento della stampa dei risultati, i numeri sono arrotondati a nove cifre. Alcuni esempi di alcuni numeri reali semplici sono:

```
1.23
-.998877
+3.1459
.7777777
-333.
.01
```

I numeri piu' piccoli di .01 o piu' grandi di 999999999. sono stampati in NOTAZIONE SCIENTIFICA. Una costante reale in notazione scientifica e' composta da tre parti:

1) MANTISSA
2) LETTERA
3) ESPONENTE

La MANTISSA e' un numero reale semplice. La LETTERA E viene usata per indicare che il numero viene visualizzato in forma esponenziale. In altri termini, E rappresenta *10 (ad esempio, 3E3=3*10 3=3000). L'ESPONENTE indica per quale potenza di 10 il numero viene moltiplicato.

Sia la mantissa che l'eponente sono numeri con segno (+ o -). Il campo di definizione dell'esponente e' compreso fra -39 e +38 ed indica il numero di posizioni di cui il punto decimale nella mantissa verrebbe spostato a sinistra (-) o a desta (+) se il valore della costante fosse rappresentato come un numero semplice.

C'e' un limite per la grandezza dei numeri reali che il BASIC e' in grado di trattare, questo anche per la notazione scientifica: il numero piu' grande e' +1.70141183E+38, ed i calcoli che diano come risultato un numero maggiore di questo comportano il messaggio di errore ?OVERFLOW ERROR. ; il numero reale piu' piccolo e' +2.93873588E-39 e, se i calcoli danno un risultato inferiore, questo viene assunto come zero e NON SI HA ALCUN MESSAGGIO DI ERRORE. Alcuni esempi di numeri reali in notazione scientifica ed i loro valori decimali sono:

```
235.988E-3     (.235988)
2359E6         (2359000000.)
-7.09E-12      (-.00000000000709)
-3.14159E+5    (-314159.)
```

Le COSTANTI STRINGA sono gruppi di informazioni alfanumeriche come lettere, numeri e simboli. Quando si digita una stringa da tastiera, la sua lunghezza non puo' superare lo spazio disponibile sulla linea di programma di 80 caratteri (cioe' tutto lo spazio NON occupato dal numero di linea e dal resto dell'istruzione).

Una costante stringa puo' contenere spaziature, lettere, numeri, punteggiature e caratteri di controllo del cursore o del colore in qualsiasi combinazione. Si possono addirittura inserire delle virgole fra i numeri. L'unico carattere che non puo' essere inserito in una stringa e' il doppio apice ("), in quanto questo carattere viene usato per definire l'inizio e la fine della stringa. Una stringa puo' assumere anche un valore nullo - cio' significa che non contiene alcun carattere. Si puo' trascurare il doppio apice al termine di una stringa se questa e' l'ultima parte di una linea o se e' seguita da due punti (:). Alcuni esempi di costanti stringa sono:

```
""    (Stringa nulla)
"L. 250.000"
"NUMERO DI IMPIEGATI"
```

NOTA: Per includere nelle stringhe il doppio apice si usa CHR$(34).

VARIABILI INTERE REALI E STRINGA

Le VARIABILI sono nomi che rappresentano i valori usati nelle istruzioni BASIC. Il valore rappresentato da una variabile puo' essere assegnato ponendo questa uguale ad una costante, oppure puo' essere il risultato dei calcoli del programma. I dati della variabile, come le costanti, possono essere numeri interi, reali o stringhe. Se si fa riferimento al nome di una variabile di un programma, prima che le sia stato assegnato un valore, l'interprete BASIC crea automaticamente la variabile assegnandole il valore zero, se questa e' un numero intero o reale, oppure il valore nullo, se e' una stringa.

I nomi delle variabili possono avere una lunghezza qualsiasi, ma solamente i primi due caratteri hanno significato per il CBM BASIC. Questo significa che tutti i nomi usati per le variabili NON devono avere i primi due caratteri uguali. I NOMI DELLE VARIABILI NON POSSONO ESSERE NE' CONTENERE PAROLE RISERVATE DEL BASIC. Le parole riservate includono tutti i comandi BASIC, le istruzioni, i nomi di funzione ed i nomi degli operatori logici. Se accidentalmente viene usata una parola riservata all'interno del nome di una variabile, sullo schermo viene visualizzato il messaggio **?SYNTAX ERROR**

I caratteri usati per formare i nomi delle variabili sono quelli dell'alfabeto ed i numeri da 0 a 9. Il primo carattere del nome deve essere una lettera. I caratteri di dichiarazione del tipo di dato (% e)) possono essere usati come ultimo carattere del nome. Il segno di percentuale (%) indica che la variabile e' un intero, ed il simbolo di dollaro ($) dichiara una variabile stringa. Se non si usa un carattere di dichiarazione di tipo, l'interprete assume la variabile come reale. Alcuni esempi di nomi di variabili, assegnazioni di valori e tipi di dati sono:

```
A$="VENDITA     "    (Variabile stringa)
MTH$="GEN"+A$        (Variabile stringa)
```

```
K%=5                    (Variabile intera)
CNT%=CNT%+1             (Variabile intera)
FP=12.5                 (Variabile reale)
SOM=FP*CNT%             (Variabile reale)
```

SCHIERE INTERE REALI E STRINGA

Una SCHIERA e' una tabella (o lista) di dati associati a cui si puo' fare riferimento mediante un unico nome di variabile. In altre parole, una schiera e' una sequenza di variabili correlate. Per esempio, una tabella di numeri puo' essere vista come una schiera. I singoli numeri all'interno della tabella diventano gli "elementi" della schiera.

Le schiere sono un modo pratico e compatto di descrivere un grande numero di variabili correlate. Prendiamo ad esempio una tabellina di numeri, e supponiamo che abbia 10 righe di 20 numeri ciascuna, per un totale di 200 numeri. Se non si avesse la possibilita' di usare un unico nome per tutta la schiera, si dovrebbe assegnare un nome diverso a ciascun valore della tabellina. Invece, grazie alle schiere, occorre adoperare soltanto un nome per la schiera, mentre tutti i suoi elementi vengono identificati attraverso le posizioni che occupano al suo interno.

Le schiere possono essere di tipo intero, reale o stringa, e tutti gli elementi che le compongono sono del suo stesso tipo. Le schiere possono essere ad una dimensione (come una lista semplice) o a piu' dimensioni (come ad esempio una griglia contrassegnata con righe e colonne, o un Cubo di Rubik[R]). Si puo' identificare e fare riferimento univocamente ad ogni elemento di una schiera attraverso una variabile indice (o sottoscritto), racchiusa fra parentesi, che segue il nome della schiera.

Il massimo numero di dimensioni che, in teoria, puo' avere una schiera e' 255, ed il numero di elementi per ogni dimensione e' limitato a 32767. In pratica pero' le dimensioni delle schiere vengono limitate dallo spazio di memoria disponibile per registrare i loro dati e/o una linea logica di 80 caratteri dello schermo. Se una schiera ha soltanto una dimensione ed il suo indice non supera mai 10 (11 elementi: da 0 a 10), la schiera viene creata automaticamente dall'interprete e riempita di zeri (o di stringhe vuote se di tipo stringa) non appena si fa riferimento per la prima volta ad un qualsiasi elemento, altrimenti occorre usare l'istruzione BASIC DIM per definire struttura e dimensioni della schiera. Si puo' determinare la quantita' di memoria necessaria per memorizzare una schiera in questo modo:

```
          5 byte  per il nome della schiera
         +2 byte  per ogni dimensione della schiera
         +2 byte  per ogni elemento di tipo intero
OPPURE   +5 byte  per ogni elemento di tipo reale
OPPURE   +3 byte  per ogni elemento di tipo stringa
E        +1 byte  per ogni carattere presente
           in ciascun elemento di tipo stringa
```

Gli indici possono essere variabili e/o costanti intere, oppure un'espressione aritmetica che dia un risultato intero. Gli indici delle schiere a piu' dimensioni devono essere separati fra loro da virgole. Gli indici possono assumere dei valori che vanno da zero al

numero di elementi appartenenti ad ognuna delle dimensioni della schiera. Valori che superino tale intervallo provocano il messaggio d'errore ?BAD SUBSCRIPT. Alcuni esempi di nomi di schiere, assegnazioni e tipi sono:

```
A$(0)="VENDITE"            (schiera stringa)
MTH$(K%)="GEN"             (schiera stringa)
G2%(X)=5                   (schiera intera)
CNT%(G2%(X))=CNT%(1)-2     (schiera intera)
FP(12*K%)=24.8             (schiera reale)
SOM(CNT%(1))=FP K%         (schiera reale)
```

`A(5)=0` (imposta a 0 l'elemento 5 della schiera unidimensionale di nome "A")

`B(5,6)=0` (pone uguale a 0 l'elemento situato nella posizione di riga 5 e colonna 6 della schiera bidimensionale di nome "B")

`C(1,2,3)=0` (pone uguale a zero l'elemento situato nella posizione di riga 1, colonna 2 e profondita' 3 della schiera tridimensionale di nome "C")

ESPRESSIONI ED OPERATORI

Le espressioni sono formate usando costanti, variabili e/o schiere. Un'espressione puo' essere una singola costante, una variabile semplice o una variabile schiera di ogni tipo. Puo' anche essere una combinazione di costanti e variabili con operatori aritmetici, relazionali o logici, designate per produrre un singolo valore. Il funzionamento degli operatori verra' spiegato in seguito. Le espressioni possono essere distinte in due classi:

1) ARITMETICHE
2) STRINGA

Normalmente, le espressioni si pensano composte da due o piu' voci chiamate operandi. Al fine di produrre il risultato desiderato, ciascun operando e' separato da un singolo operatore. Cio' di solito viene fatto assegnando il valore di un'espressione ad un nome di variabile. Tutti gli esempi di costanti e variabili visti rapidamente erano anche esempi di espressioni.

Un operatore e' un simbolo speciale che l'interprete BASIC del COMMODORE 64 riconosce come rappresentante di un'operazione che deve essere eseguita sulle variabili o sui dati costanti. Uno o piu' operatori combinati con una o piu' variabili e/o costanti formano un'espressione. Il BASIC del COMMODORE 64 riconosce operatori logici, relazionali ed aritmetici.

ESPRESSIONI ARITMETICHE

Quando vengono risolte, le espressioni aritmetiche danno un valore intero o reale. Gli operatori aritmetici (+,-,*,/,)sono usati per eseguire addizioni, sottrazioni, moltiplicazioni, divisioni ed elevamenti a potenza, rispettivamente.

OPERAZIONI ARITMETICHE

Un operatore aritmetico definisce un'operazione aritmetica eseguita sui due operandi che si trovano ai lati dell'operatore. Le operazioni aritmetiche vengono eseguite usando numeri reali. Prima dell'esecuzione di un'operazione aritmetica, i numeri interi vengono convertiti in numeri reali. Il risultato viene di nuovo riconvertito in un intero se e' assegnato ad un nome di variabile intera.

ADDIZIONE (+): Il segno piu' (+) indica che l'operando sulla destra e' addizionato all'operando sulla sinistra.

ESEMPI:
```
2+2
A+B+C
X%+1
BR+10E-2
```

SOTTRAZIONE (-): Il segno meno (-) indica che l'operando di destra viene sottratto da quello di sinistra.

ESEMPI:
```
4-1
100-64
A-B
55-142
```

Il segno meno puo' essere usato anche come operatore unario. Cio' significa che, posto davanti ad numero, fa interpretare quest'ultimo come numero negativo. In altre parole, e' come se il numero venisse sottratto da zero.

ESEMPI:
```
-5
-9E4
-B
4-(-2)    equivalente a 4+2
```

MOLTIPLICAZIONE(*): Un asterisco indica che l'operando di sinistra viene moltiplicato per l'operando di destra.

ESEMPI:
```
100*2
50*0
A*X1
R%*14
```

DIVISIONE (/): La sbarra specifica che l'operando di sinistra viene diviso per quello di destra.

ESEMPI:
```
10/2
6400/4
A/B
4E2/XR
```

ELEVAMENTO A POTENZA (↑): La freccia verso l'alto indica che

l'operando sulla sinistra e' elevato alla potenza specificata dall'operando di destra (esponente). Se l'operando sulla destra e' 2, allora l'operando sulla sinistra e' elevato al quadrato, se e' 3 al cubo, ecc. Affinche' il risultato dell'operazione dia un numero reale valido, l'esponente puo' essere un numero qualsiasi.

ESEMPI:

2↑2	equivalente a 2*2
3↑3	equivalente a 3*3*3
4↑4	equivalente a 4*4*4*4
AB↑CD	
3↑-2	equivalente a 1/3*1/3

OPERATORI RELAZIONALI

Gli operatori relazionali (<,=,>,<=,=>,<>) sono usati principalmente per confrontare i valori di due operandi, ma producono anche un risultato aritmetico. Quando si usano in operazioni di confronto, gli operatori relazionali e gli operatori logici (AND, OR, NOT) producono la valutazione aritmetica vero/falso di un'espressione. In un'espressione, se il rapporto e' vero al risultato viene assegnato il valore -1, mentre se e' falso il risultato e' 0. Gli operatori relazionali sono i seguenti:

<	MINORE
=	UGUALE
>	MAGGIORE
<=	MINORE O UGUALE
>=	MAGGIORE O UGUALE
<>	DIVERSO

ESEMPI:

1=5-4	vero (-1)
14>66	falso (0)
15>=15	vero (-1)

Gli operatori relazionali possono essere utilizzati per il confronto di stringhe. In questo caso, l'ordinamento delle lettere dell'alfabeto e' A<B<C<D ecc. Le stringhe sono confrontate valutando la relazione tra caratteri corrispondenti, muovendo da sinistra a destra (vd. Operazioni su Stringhe).

ESEMPI:

"A" < "B"	vero (-1)
"X" = "YY"	falso (0)
BB$ <> CC$	

I dati numerici possono essere confrontati solamente con altri dati numerici, cosi' come le stringhe posono essere confrontate solamente con altre stringhe; in ogni altro caso viene generato il messaggio di errore ?TYPE MISMATCH . Gli operandi numerici che devono essere confrontati vengono innanzitutto convertiti dalla forma intera a quella reale. Dopodiche' si confrontano i valori reali per attribuire loro il risultato di vero o falso.

Al termine di ogni confronto si ottiene un numero intero,

indipendentemente dal tipo di dato dell'operando (anche nel caso che siano entrambi stringhe). Di conseguenza, il confronto fra due operandi puo' essere usato come operando durante l'esecuzione dei calcoli. Il risultato e' 0 oppure -1, e puo' essere usato in qualunque modo eccetto che come divisore, dato che la divisione per zero non ha significato.

OPERATORI LOGICI

Gli OPERATORI LOGICI (AND, OR, NOT) possono essere usati per modificare i significati degli operatori relazionali, o per produrre un risultato aritmetico. Gli operatori logici possono dare risultati diversi da 0 e -1, anche se qualsiasi risultato diverso da zero viene assunto come vero, quando si testi una condizione vera/falsa.

Gli operatori logici (chiamati talvolta Operatori Booleani) possono anche essere usati per eseguire operazioni logiche su due operandi, di cui viene considerata una sola cifra binaria (bit). Quando si usa l'operatore NOT, l'operazione viene eseguita sul solo operando di destra. Gli operandi devono essere compresi fra -32768 e +32767 (i numeri reali vengono convertiti in interi), e le operazioni logiche devono dare un risultato intero.

Le operazioni logiche sono eseguite bit per bit su due operandi. La AND logica da' come risultato 1 solo se entrambi i bit rispettivi degli operandi sono a 1. La OR logica da' come risultato 1 se uno dei bit degli operandi e' uguale a 1. La NOT logica da' come risultato il valore opposto di ciascun bit di un singolo operando. In altre parole, e' come dire "Se e' NOT 1 allora e' 0, se e' NOT 0 allora e' 1".

La OR esclusiva (XOR) non ha operatore logico, ma viene eseguita come parte dell'istruzione WAIT. La OR esclusiva significa che se i bit dei due operandi sono uguali allora il risultato e' 0, altrimenti e' 1.

Le operazioni logiche sono definite da gruppi di istruzioni, che insieme costituiscono la "Tavola della Verita'" booleana, come mostrato nella tabella 1.2.

Il risultato dell'operazione AND e' 1 solo se entrambi i bit sono uguali a 1:

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

Il risultato della OR e' 1 solo se almeno un bit e' 1:

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

NOTA: segue il complemento logico di ciascun bit:

NOT 1 = 0
NOT 0 = 1

La OR esclusiva (XOR) fa parte dell'istruzione WAIT:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Tabella 1.2 - Tavola booleana della Verità

Gli operatori logici AND, OR, NOT specificano che, nelle espressioni a due operandi, un'operazione di aritmetica booleana deve essere eseguita da tutte e due le parti dell'operatore. Solamente nel caso di NOT viene preso in considerazione il solo operando di destra. Le operazioni logiche (o di aritmetica booleana) non vengono eseguite finche' non siano state completate tutte le operazioni aritmetiche e logiche di un'espressione.

ESEMPI:

```
IF A=100 AND B=100 THEN 10 (Se sia A che B hanno valore 100 allora il
                            risultato e' vero)

A=96 AND 32:PRINT A (A=32)

IF A=100 OR B=100 THEN 20 (Se A o B sono uguali a 100 allora il
                           risultato e' vero)

A=64 OR 32:PRINT A          (A=96)

IF NOT X<Y THEN 30          (Se X=>Y allora il risultato e' vero)

X=NOT 96                    (Il risultato e' -97 [complemento a 2])
```

GERARCHIA DELLE OPERAZIONI

Tutte le espressioni eseguono tipi diversi di operazioni, in base alla gerarchia fissata. In altri termini, alcune operazioni vengono eseguite prima di altre. Il normale ordine delle operazioni puo'

essere modificato racchiudendo due o piu' operandi fra parentesi (), creando cosi' una "sottoespressione". In questo modo, prima viene calcolato il valore dentro le parentesi, poi quello al di fuori.

Usando le parentesi, si deve fare attenzione che il numero delle parentesi di destra sia uguale a quello delle parentesi di sinistra. Se alcune parentesi rimangono aperte, compare il messaggio BASIC **?SYNTAX ERROR**

Inserendo dentro le parentesi gruppi di operandi racchiusi essi stessi fra parentesi, si possono formare delle espressioni a piu' livelli. Questo procedimento prende il nome di nidificazione. Le parentesi possono essere nidificate fino ad un massimo di 10 livelli. L'espressione che si trova al livello piu' basso viene eseguita per prima. Alcuni esempi di espressioni sono:

```
A+B
C↑(D+E)/2
((X-C↑(D+E)/2)*10)+1
GG$>HH$
JJ$+"MORE"
K%=1 AND M<>X
K%=2 OR (A=B AND M<X)
NOT (D=E)
```

Normalmente, l'interprete BASIC realizza le operazioni su espressioni eseguendo prima le operazioni aritmetiche, poi le operazioni relazionali ed infine le operazioni logiche. Sia gli operatori logici che quelli aritmetici hanno un proprio ordine di precedenza (o gerarchia di operazione). D'altra parte, gli operatori relazionali non hanno un ordine di precedenza, e vengono eseguiti da sinistra a destra durante la valutazione dell'espressione.

Se tutti gli operatori rimanenti hanno lo stesso livello di precedenza, le operazioni vengono eseguite da sinistra a destra. Nelle espressioni con le parentesi, viene ugualmente mantenuto l'ordine di precedenza. La gerarchia delle operazioni logiche ed aritmetiche dalla prima all'ultima, in ordine di precedenza, e' mostrata nella Tabella 1.3.

OPERATORE	DESCRIZIONE	ESEMPIO
↑	Elevamento a potenza	BASE.EXP
-	Negazione (Meno Unario)	-A
*/	Moltiplicazione/Divisione	AB*CD, EF/GH
+-	Addizione/Sottrazione	CNT+2, JK-PQ
>=<	Operazioni Relazionali	A <= B
NOT	NOT logico (complemento intero a 2)	NOT K%
AND	AND logico	JK AND 128
OR	OR logico	PQ OR 15

Tabella 1.3 - Gerarchia delle operazioni su espressioni

OPERAZIONI SU STRINGHE

Le stringhe vengono confrontate utilizzando gli stessi operatori relazionali (=, <>, <=, =>, <, >) utilizzati per confrontare i numeri. I confronti di stringhe vengono fatti prendendo un carattere alla volta (da sinistra a destra) da ciascuna stringa, e valutando ciascuna posizione del codice del carattere prelevato dal set di caratteri

PET/CBM. Se i codici carattere sono uguali, allora anche i caratteri sono considerati uguali. Se i codici carattere sono diversi, il carattere con il numero di codice piu' basso e' il piu' basso nell'insieme dei caratteri. I confronti terminano quando viene raggiunta la fine dell'una o dell'altra stringa. Se tutti i codici sono uguali, viene considerata minore la stringa piu' corta. GLI SPAZI BIANCHI CHE UNA STRINGA SI PORTA DIETRO SONO SIGNIFICATIVI.

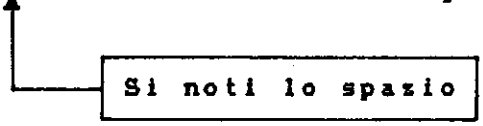
Alla fine di tutti i confronti, indipendentemente dal tipo dei dati, si ottiene un risultato intero. Cio' e' vero anche se entrambi gli operandi sono stringhe. Inoltre, il confronto di due operandi stringa puo' essere usato come operando nell'esecuzione di calcoli. Il risultato e' -1 o 0 (vero o falso, rispettivamente), e puo' essere usato in qualunque modo eccetto che come divisore, poiche' la divisione per zero non ha significato.

ESPRESSIONI STRINGA

Le espressioni sono trattate come se fossero seguite da un implicito "<>0". Cio' significa che se un'espressione risulta vera, allora vengono eseguite le istruzioni BASIC che la seguono, sulla stessa linea di programma. Se invece l'espressione e' falsa, il resto della linea e' ignorato, e viene eseguita la linea di programma successiva.

Come per i numeri, le operazioni possono essere eseguite anche su variabili stringa. L'unico operatore aritmetico stringa riconosciuto dal BASIC CBM e' il segno piu' (+), usato per eseguire le concatenazioni delle stringhe. Quando si concatenano le stringhe, la stringa alla destra del segno piu' viene aggiunta alla stringa di sinistra, ottenendo una terza stringa, che puo' essere stampata subito, usata in un confronto oppure assegnata ad nome di variabile. Se si confronta una stringa con un numero, o viceversa, compare il messaggio BASIC ?TYPE MISMATCH. I seguenti sono esempi di concatenazioni ed espressioni stringa:

```
10 A$="NOME":B$="FILE"
20 NAM$=A$+B$              (da' la stringa NOMEFILE)
30 RES$="NUOVO "+A$+B$ (da' la stringa NUOVO NOMEFILE)
```



TECNICHE DI PROGRAMMAZIONE

CONVERSIONE DEI DATI

Quando e' necessario, l'interprete CBM BASIC converte un dato numerico da intero a reale o viceversa, rispettando le seguenti regole:

* Tutte le operazioni aritmetiche e relazionali sono eseguite in virgola mobile. Per valutare l'espressione, gli interi vengono convertiti in reali, ed il risultato viene riconvertito in intero.
* Se il nome di una variabile numerica di un certo tipo e' posto uguale ad un dato numerico di tipo diverso, il numero viene convertito e memorizzato nello stesso modo del tipo di dato

dichiarato nel nome della variabile.

* Quando un valore reale viene convertito ad intero, la parte frazionaria viene troncata, ed il risultato intero risulta minore o uguale al valore in virgola mobile. Se il risultato esce dall'intervallo che va da -32768 a +32767, compare il messaggio BASIC: ?ILLEGAL QUANTITY

USO DELL'ISTRUZIONE INPUT

Una volta conosciute le variabili, il passo successivo consiste nel loro abbinamento all'istruzione INPUT per applicazioni pratiche della programmazione.

Come primo esempio, si puo' pensare ad una variabile come ad una "cella di memoria" dove il COMMODORE 64 memorizza la risposta ad una domanda dell'Utente. Se si vuole che un programma chieda all'Utente di digitare un nome, tale nome puo' essere assegnato alla variabile N$. In questo modo, ogni volta che nel programma si scrive PRINT N$, il COMMODORE 64 stampa automaticamente il nome digitato dall'Utente.

Per prima cosa, digitare sul COMMODORE 64 la parola NEW, premere il tasto RETURN e battere questo esempio:

```
10 PRINT"COME TI CHIAMI ?":INPUT N$
20 PRINT"CIAO, "N$"!"
```

In questo esempio si usa N per ricordare che in questa variabile e' contenuto il nome. Il segno del dollaro ($) e' usato per indicare al computer che si sta usando una variabile stringa. Risulta molto importante la differenza fra questi due tipi di variabile:

1) NUMERICA
2) STRINGA

Si ricordera' dai primi paragrafi che le variabili numeriche sono usate per memorizzare valori numerici come 1, 100, 4000, ecc. Una variabile numerica puo' essere rappresentata da una lettera (A), da due lettere (AB), da una lettera ed un numero (A1), oppure da due lettere ed un numero (AB1). L'uso di nomi corti consente di risparmiare memoria. Un altro aiuto si ha utilizzando nello stesso programma lettere e numeri di categorie diverse (A1, A2, A3). Inoltre, se da una risposta si desiderano ricevere numeri interi anziche' decimali, basta ggiungere il segno di percentuale alla fine del nome della variabile (AB%, A1%, ecc.).

Consideriamo ora alcuni esempi riportanti diversi tipi di variabili ed espressioni unite all'istruzione INPUT:

```
10 PRINT "BATTI UN NUMERO":INPUT A
20 PRINT A

10 PRINT "BATTI UNA PAROLA":INPUT A$
20 PRINT A$

10 PRINT "BATTI UN NUMERO":INPUT A
20 PRINT A"MOLTIPLICATO 5 UGUALE"A*5
```

NOTA: Il terzo esempio mostra che i MESSAGGI vengono inseriti fra i doppi apici (""), mentre le variabili sono al di fuori. Si puo' inoltre notare, nella riga 20, che la variabile A viene stampata

prima del messaggio "MOLTIPLICATO 5 UGUALE", e che per ultimo viene eseguito il calcolo A*5 (moltiplica per 5 la variabile A)

Nella maggior parte dei programmi, i calcoli sono molto importanti. Si puo' scegliere se usare "numeri effettivi" o variabili, ma se si sta lavorando con numeri forniti dall'Utente ci si deve ricordare di sostituirli con variabili numeriche. Iniziamo chiedendo all'Utente di battere due numeri; un modo e' il seguente:

```
10 PRINT"BATTI DUE NUMERI":INPUT A:INPUT B
```

ESEMPIO DI BILANCIO DI ENTRATA/USCITA

```
5 PRINT "□"—SHIFT CLR/HOME
10 PRINT"MONTHLY INCOME":INPUT IN
20 PRINT
30 PRINT"EXPENSE CATEGORY 1":INPUT E1$
40 PRINT"EXPENSE AMOUNT":INPUT E1
50 PRINT
60 PRINT"EXPENSE CATEGORY 2":INPUT E2$
70 PRINT"EXPENSE AMOUNT":INPUT E2
80 PRINT
90 PRINT"EXPENSE CATEGORY 3":INPUT E3$
100 PRINT"EXPENSE AMOUNT":INPUT E3
110 PRINT "□"—SHIFT CLR/HOME
120 E=E1+E2+E3
130 EP=E/IN
140 PRINT"MONTHLY INCOME: $"IN
150 PRINT"TOTAL EXPENSES: $"E
160 PRINT"BALANCE EQUALS: $"IN-E
170 PRINT
180 PRINT E1$"="(E1/E)*100"% OF TOTAL EXPENSES"
190 PRINT E2$"="(E2/E)*100"% OF TOTAL EXPENSES"
200 PRINTE3$"="(E3/E)*100"% OF TOTAL EXPENSES"
210 PRINT
220 PRINT"YOUR EXPENSES="EP*100"% OF YOUR TOTAL
INCOME"
230 FOR X=1TO5000:NEXT:PRINT
240 PRINT"REPEAT? (Y OR N)":INPUT Y$:IF Y$="Y"THEN5
250 PRINT "□":END
          SHIFT CLR/HOME
```

NOTA: **IN NON puo' essere uguale a zero; E1, E2, E3 NON possono essere tutti uguali a zero contemporaneamente.**

SPIEGAZIONE LINEA PER LINEA DEL BILANCIO ENTRATA/USCITA

Linea	Descrizione
5	Azzera lo schermo
10	Istruzione PRINT/INPUT
20	Inserisce una linea
30	Categoria di Spesa 1 = E1$
40	Ammontare della spesa 1 = E1
50	Inserisce una linea
60	Categoria di Spesa 2 = E2$
70	Ammontare della spesa 2 = E2
80	Inserisce una linea
90	Categoria di Spesa 3 = E3$
100	Ammontare della spesa 3 = E3
110	Azzera lo schermo
120	Somma l'ammontare delle spese = E
130	Calcola la percentuale Uscita/Entrata
140	Visualizza l'Entrata
150	Visualizza il Totale Uscite
160	Visualizza le Entrate e le Uscite
170	Inserisce una linea
180-200	Calcolano qual e' la percentuale di ogni Uscita rispetto al Totale Uscite
210	Inserisce una linea
220	Visualizza la percentuale Entrata/Uscita
230	Ciclo di ritardo

Moltiplichiamo ora quei due numeri per generare una nuova variabile C, come e' riportato nella seguente linea 20:

```
20 C=A*B
```

e stampiamo il risultato sotto forma di un messaggio:

```
30 PRINT    A"MOLTIPLICATO"B"UGUALE"C
```

Digitiamo queste tre linee e "lanciamo" (RUN) il programma. Si puo' notare che i messaggi sono racchiusi fra i doppi apici, mentre le variabili no.

Poniamo ora il segno del dollaro ($) davanti al numero rappresentato dalla variabile C. Tale segno deve essere stampato dentro gli apici e davanti alla variabile C. Per aggiungere il $ nel programma premere i tasti RUN/STOP e RESTORE , quindi digitare la linea 40 come segue:

```
40 PRINT "$" C
```

Premere ora RETURN , e poi RUN seguito di nuovo da RETURN Il segno del dollaro deve essere posto tra apici, in quanto la variabile C rappresenta solo un numero, e non puo' contenere un $. Se C contiene il valore 100, il COMMODORE 64 visualizza $100. Ma se si prova a scrivere PRINT $C viene emesso il messaggio **?SYNTAX ERROR** .

Un ultimo accenno a $: si puo' anche creare una variabile che rappresenta il segno del dollaro, e che puo' essere sostituita al simbolo $ che si vuole usare. Per esempio:

```
10 Z$="$"
```

A questo punto, quando si ha bisogno di inserire un $, si puo' usare la variabile stringa Z$:

```
10 Z$="$":INPUT A    T A
20 PRINT Z$A
```

La riga 10 definisce il simbolo $ come una variabile stringa di nome Z$; inoltre, richiede un numero chiamato A. La linea 20 stampa Z$ ($) accanto ad A (numero).

Probabilmente, risulta piu' facile assegnare a variabili stringa (rti caratteri, come $, piuttosto che digitare "$" ogni volta che si voglia calcolare una cifra in dollari o qualunque altra segno richiedente i doppi apici, come %.

USO DELL'ISTRUZIONE GET

La maggior parte dei programmi semplici usa l'istruzione INPUT per prendere i dati inviati dall'Utente che opera sul computer. Quando si manifestano esigenze piu' complesse, come la protezione dagli errori di battitura, l'istruzione GET permette di operare in modo piu' flessibile, rendendo i programmi piu' "intelligenti". In questo paragrafo si illustra l'uso dell'istruzione GET per permettere ai vari programmi di sfruttare alcune caratteristiche particolari dell'editing di schermo.

Il COMMODORE 64 ha un buffer di tastiera che puo' contenere fino a 10 caratteri. Cio' significa che se il computer e' occupato in alcune operazioni e non e' disponibile per la tastiera, si puo' lo stesso

continuare ad introdurre fino a 10 caratteri, usati non appena il computer ha terminato il suo lavoro. Quanto esposto e' dimostrabile provando a digitare:

```
10 TI$="000000"
20 IF TI$ < "000015" THEN 20
```

Digitare ora RUN e premere RETURN ; quindi, mentre il programma sta "girando", proviamo a digitare la parola HELLO.

Si puo' notare che non accade nulla per circa 15 secondi dall'inizio del programma. Trascorso questo tempo, sullo schermo compare il messaggio HELLO.

Supponiamo di essere in coda davanti ad un cinema. La prima persona in coda e' la prima a prendere il biglietto ed a lasciare la coda, mentre l'ultima persona in coda e' l'ultima a prendere il biglietto. L'istruzione GET si comporta come il bigliettaio: per prima cosa si accerta che ci siano dei caratteri "in coda", cioe' che sia stato battuto qualche tasto. Se la risposta e' affermativa, il carattere incontrato prende posto nella variabile appropriata; se la risposta e' negativa, alla variabile viene assegnato il valore vuoto.

A questo punto, e' importante notare che se si prova ad inserire nel buffer piu' di 10 caratteri alla volta, tutti quelli dopo il decimo sono persi.

Poiche' l'istruzione GET e' attiva anche quando non viene inserito alcun carattere, risulta necessario inserire tale istruzione in un ciclo, in modo da farle attendere la prima battitura di un tasto o il primo carattere inviato da programma.

La forma consigliata per l'istruzione GET e' la seguente (digitare NEW per cancellare il programma precedente):

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

Si noti che non ci sono spazi fra gli apici: cio' indica un valore vuoto, che rimanda il programma indietro all'istruzione GET, con un ciclo ("loop") continuo, finche' non viene premuto un tasto del computer. Una volta battuto il tasto, il programma riprende dalla linea successiva. Aggiungiamo al programma precedente la seguente linea:

```
100 PRINT A$;: GOTO 10
```

Se ora digitiamo RUN, si puo' notare che sullo schermo non compare nessun cursore (■), ma qualsiasi carattere digitato viene visualizzato. Queste due linee di programma possono essere riscritte in un programma di editor di schermo, come mostrato in seguito.

Con l'editor di schermo si possono fare molte altre cose. Si puo' realizzare un cursore lampeggiante. Si possono gestire numerosi tasti, quali CLR/HOME , che provvedono ad azzerare lo schermo. Si possono usare tasti personali di funzioni per rappresentare intere parole o frasi. A proposito di tasti funzionali, le linee del seguente programma attribuiscono a ciascun tasto funzionale uno scopo speciale. Questo e' solo l'inizio di un programma che si puo' adattare alle nostre necessita'.

```
20 IF A$ = CHR$(133) THEN POKE 53280,8:GOTO 10
30 IF A$ = CHR$(134) THEN POKE 53281,4:GOTO 10
40 IF A$ = CHR$(135) THEN A$= "GENTILE SIGNORE : " +CHR$(13)
```

```
50 IF A$ = CHR$(136) THEN A$="CORDIALI SALUTI," +CHR$(13)
```

I numeri tra parentesi dei CHR$ provengono dalla tabella dei codici di CHR$ riportati nell'Appendice C; tale tabella riporta un numero differente per ciascun carattere. I quattro tasti funzionali sono impostati per eseguire i compiti rappresentati dalle istruzioni che seguono la parola THEN di ciascuna linea. Cambiando i numeri dentro le parentesi di CHR$, si possono designare caratteri differenti. Cambiando le informazioni dopo l'istruzione THEN, si possono eseguire istruzioni diverse.

COMPATTAZIONE DEI PROGRAMMI BASIC

Nei programmi BASIC possono essere compattate piu' istruzioni, rendendo il programma stesso piu' corto e piu' veloce possibile. Questo processo di riduzione dei programmi si chiama "compattazione".

La compattazione dei programmi permette di ridurre il numero massimo delle istruzioni di un programma. Con questo metodo si riduce molto anche la misura del programma, tanto da riuscire a farlo "girare" in uno spazio di memoria diversamente non sufficiente; senza contare che l'aumento di spazio che ne deriva consente la memorizzazione di una maggiore quantita' di dati.

ABBREVIAZIONE DELLE PAROLE CHIAVE

L'Appendice A riporta una lista delle abbreviazioni delle parole chiave. Cio' permette di inserire in una linea una maggiore quantita' di istruzioni. L'abbreviazione piu' comunemente usata e' il punto interrogativo (?), che sostituisce il comando PRINT. Tuttavia, richiedendo la lista di un programma con l'istruzione LIST, il COMMODORE 64 non ripete le abbreviazioni, ma visualizza le parole chiave per esteso. Se una linea di programma eccede gli 80 caratteri (2 linee dello schermo) con le parole chiave non abbreviate, e si desidera modificarla, e' necessario riscrivere tutta la linea con le parole chiave abbreviate, dopodiche' si puo' salvare il programma. Il salvataggio di un programma include le parole-chiave senza dilatare le linee, in quanto il COMMODORE 64 codifica con un singolo carattere le parole chiave del BASIC. In genere, le abbreviazioni vengono introdotte dopo che un programma e' stato scritto e non c'e' piu' bisogno di LISTarlo prima di salvarlo.

RIDUZIONE DEI NUMERI DI LINEA DI UN PROGRAMMA

Gran parte dei programmatori inizia i programmi dalla linea 100, assegnando ad ogni linea successiva un intervallo di 10 (ad esempio, 100, 110, 120). Cio' consente di inserire altre linee (111, 112, ecc.) man mano che il programma viene sviluppato. Un metodo di compattazione dei programmi una volta che siano stati completati e' PORTARE I NUMERI DI LINEA IL PIU' IN BASSO POSSIBILE (ad esempio, 1, 2, 3), in quanto i numeri di linea piu' alti occupano piu' memoria: il numero 100, ad esempio, occupa tre bytes (uno per cifra), mentre il numero 1 ne occupa solo 1.

ISTRUZIONI MULTIPLE SU UNA LINEA

Su ciascuna linea numerata del programma puo' trovare posto piu' di un'istruzione, ognuna separata dalle altre da due punti (:). L'unica

limitazione riguarda la lunghezza della linea, che non deve superare gli 80 caratteri standard, compresi i due punti. Il seguente e' un esempio di programma, prima e dopo la compattazione:

PRIMA DELLA COMPATTAZIONE

```
10 PRINT "HELLO. . .";
20 FOR T=1 TO 500:NEXT
30 PRINT "HELLO, AGAIN . . ."
40 GOTO 10
```

DOPO LA COMPATTAZIONE

```
10 PRINT "HELLO . . .";:FORT=1TO
   500:NEXT:PRINT"HELLO,
   AGAIN . . .":GOTO10
```

RIMOZIONE DELLE ISTRUZIONI REM

Le istruzioni REM sono utili per ricordarsi - o illustrare ad altri programmatori - quello che fa una certa parte del programma. Tuttavia, quando il programma e' completato nella sua versione definitiva, non c'e' piu' bisogno di tali istruzioni, per cui si puo' risparmiare un po' di spazio di memoria rimuovendole. Se in seguito si pensa di rivedere o studiare la struttura del programma, e' consigliabile tenerne una copia, completa di tutte le istruzioni REM.

VARIABILI

Se in un programma si usano ripetutamente un numero, una parola o una frase, e' utile, in genere, definire al loro posto una variabile che li contenga. Parole e frasi possono essere definite come variabili stringa, mediante l'uso di una lettera e del simbolo "$". Un esempio puo' essere il seguente:

PRIMA DELLA COMPATTAZIONE

```
10 POKE 54296,15
20 POKE 54276,33
30 POKE 54273,10
40 POKE 54273,40
50 POKE 54273,70
60 POKE 54296,0
```

DOPO LA COMPATTAZIONE

```
10 V=54296:F=54273
20 POKEV,15:POKE54276,33
30 POKEF,10:POKEF,40:POKEF,70
40 POKEV,0
```

ISTRUZIONI READ E DATA

Grosse quantita' di dati possono essere digitate una sola volta come unico blocco di dati, ricominciando dall'inizio tutte le volte...oppure, si puo' scrivere UNA SOLA VOLTA la parte di programma relativa alle istruzioni, e riversare tutti i dati che devono essere trattati in una lista che prende il nome di istruzione DATA. Questo procedimento e' particolarmente valido per riempire lunghe liste di numeri di un programma.

SCHIERE E MATRICI

Schiere e matrici sono simili alle istruzioni DATA, poiche' consentono il trattamento di grosse quantita' di dati sotto forma di lista, dal quale la parte di programma relativa al trattamento dei dati preleva seguenzialmente questi ultimi. Le schiere differiscono per la possibilita' di avere tale lista a piu dimensioni.

ELIMINAZIONE DEGLI SPAZI

Uno dei modi piu' facili per ridurre la misura di un programma e' l'eliminazione degli spazi. Anche se spesso si introducono spazi nei programmi di prova per una maggiore chiarezza, in effetti non ce n'e' alcun bisogno, e la loro eliminazione comporta un risparmio di memoria.

PROCEDURE GOSUB

Se si fa un uso ripetuto di una particolare linea o istruzione, e' consigliabile saltare a quella linea da diversi punti del programma facendo ricorso all'istruzione GOSUB, anziche' scrivere tutta la linea ogni volta che se ne ha bisogno.

TAB E SPC

Piuttosto che stampare diversi comandi di cursore per posizionare un carattere sullo schermo, spesso e' piu' economico usare le istruzioni TAB e SPC.

CAPITOLO 2

vocabolario del linguaggio BASIC

- Introduzione
- Parole Chiave. Abbreviazioni e Tipi di Funzione del BASIC
- Descrizione delle Parole Chiave del BASIC
- Tastiera e Caratteristiche del COMMODORE 64
- Editor di schermo

INTRODUZIONE

Questo capitolo spiega ampiamente le parole chiave del linguaggio BASIC CBM. Per prima cosa viene riportato un prontuario delle parole chiave, che contiene anche le relative abbreviazioni e la rappresentazione sullo schermo di ciascuna lettera. Successivamente, vengono spiegati la sintassi ed il funzionamento di ciascuna parola chiave, riportando esempi esplicativi, del loro uso nei programmi.

Per convenienza, il BASIC del COMMODORE 64 permette di abbreviare la maggior parte delle parole chiave. Le abbreviazioni sono introdotte nella macchina digitando il numero di lettere della parola chiave sufficiente per distinguerla dalle altre parole chiave; l'ultima lettera o carattere grafico viene immesso premendo il tasto SHIFT .

Quando vengono usate nel programma, le abbreviazioni NON salvano memoria, perche' l'interprete BASIC riduce tutte le parole chiave ad un "token" di un singolo carattere. Un programma contenente delle abbreviazioni viene listato con le parole chiave nella loro forma intera. Le abbreviazioni possono essere usate per inserire piu' istruzioni in una linea di programma, anche se superano la lunghezza della linea logica di 80 caratteri dello schermo. L'editor di schermo lavora su una linea di 80 caratteri. Cio' significa che, se si usano abbreviazioni su qualsiasi linea che supera gli 80 caratteri, non siamo in grado di editare questa linea al momento di listarla. Cio' che invece si deve fare e' (1) digitare nuovamente l'intera linea, includendo tutte le abbreviazioni, oppure (2) dividere la singola linea di codice in due linee, ciascuna delle quali ha il proprio numero di linea, ecc.

Una lista completa di parole chiave, abbreviazioni, e della loro visualizzazione sullo schermo e' presentata nella tabella 2.1. La lista e' seguita da una descrizione di tutte le istruzioni, i comandi e le funzioni disponibili sul COMMODORE 64.

Questo capitolo mostra inoltre le funzioni interne dell'interprete del linguaggio BASIC. Le funzioni di sistema possono essere usate direttamente nelle istruzioni o in qualsiasi programma,senza dover definire prima la funzione stessa. Le stesse regole non valgono per le funzioni definite dall'utente. I risultati delle funzioni di sistema del BASIC possono essere usati come output immediati o possono essere assegnati a nomi di variabili di tipo appropriato.

Ci sono due tipi di funzioni BASIC:

1) NUMERICHE

2) STRINGA

Gli argomenti delle funzioni di sistema sono spesso racchiusi tra parentesi (). Le parentesi vengono poste dopo la parola chiave della funzione e NON SONO AMMESSI spazi tra l'ultima lettera della parola chiave e la parentesi di sinistra.

Il tipo di argomento necessario e'generalmente deciso dal tipo di dato del risultato. Le funzioni che hanno come risultato un valore stringa sono identificate dalla presenza del segno dollaro ($) come ultimo carattere della parola chiave. In certi casi, le funzioni stringa contengono uno o piu' argomenti numerici.

Le funzioni numeriche convertono, come necessario, tra formato intero e reale. Nella descrizione che segue, vengono mostrati i nomi delle funzioni con i relativi tipi di dato come valore di ritorno. I tipi degli argomenti vengono dati con il formato dell'istruzione.

PAROLE CHIAVE, ABBREVIAZIONI, TIPI DI FUNZIONE DEL BASIC

Tabella 2.1 - DESCRIZIONE DELLE PAROLE CHIAVE DEL BASIC

COMANDO	ABBR.	SCHERMO	FUNZIONE	COMANDO	ABBR.	SCHERMO	FUNZIONE
ABS	A SHIFT B	A	numerica	LET	L SHIFT E	L	
AND	A SHIFT N	A		LIST	L SHIFT I	L	
ASC	A SHIFT S	A	numerica	LOAD	L SHIFT O	L	
ATN	A SHIFT T	A	numerica	LOG		LOG	numerica
CHR$	C SHIFT H	C	stringa	MID$	M SHIFT I	M	stringa
CLOSE	CL SHIFT O	CL		NEW		NEW	
CLR	C SHIFT L	C		NEXT	N SHIFT E	N	
CMD	C SHIFT M	C		NOT	N SHIFT O	N	
CONT	C SHIFT O	C		ON		ON	
COS		COS	numerica	OPEN	O SHIFT P	O	
DATA	D SHIFT A	D		OR		OR	
DEF	D SHIFT E	D		PEEK	P SHIFT E	P	numerica
DIM	D SHIFT I	D		POKE	P SHIFT O	P	
END	E SHIFT N	E		POS		POS	numerica
EXP	E SHIFT X	E	numerica	PRINT	?	?	
FN		FN		PRINT#	P SHIFT R	P	
FOR	F SHIFT O	F		READ	R SHIFT E	R	
FRE	F SHIFT R	F	numerica	REM		REM	
GET	G SHIFT E	G		RESTORE	RE SHIFT S	RE	
GET#		GET#		RETURN	RE SHIFT T	RE	
GOSUB	GO SHIFT S	GO		RIGHT$	R SHIFT I	R	stringa
GOTO	G SHIFT O	G		RND	R SHIFT N	R	numerica
IF		IF		RUN	R SHIFT U	R	
INPUT		INPUT		SAVE	S SHIFT A	S	
INPUT#	I SHIFT N	I		SGN	S SHIFT G	S	numerica
INT		INT	numerica	SIN	S SHIFT I	S	numerica
LEFT$	LE SHIFT F	LE	stringa	SPC(	S SHIFT P	S	
LEN		LEN	numerica	SQR	S SHIFT Q	S	numerica

COMANDO	ABBR.	SCHERMO	FUNZIONE
STATUS	ST	ST	numerica
STEP	ST SHIFT E	ST	
STOP	S SHIFT T	S	
STR$	ST SHIFT R	ST	stringa
SYS	S SHIFT Y	S	
TAB(	T SHIFT A	T	stringa
TAN		TAN	numerica

COMANDO	ABBR.	SCHERMO	FUNZIONE
THEN	T SHIFT H	T	
TIME	TI	TI	numerica
TIME$	TI$	TI$	stringa
TO		TO	
USR	U SHIFT S	U	numerica
VAL	V SHIFT A	V	numerica
VERIFY	V SHIFT E	V	
WAIT	W SHIFT A	W	

DESCRIZIONE DELLE PAROLE CHIAVE DEL BASIC

ABS

TIPO: Funzione Numerica
FORMATO: ABS(<espressione>)

Azione: Ritorna il valore assoluto del numero,che rappresenta il valore del numero senza il segno. Il valore assoluto di un numero negativo e' quel numero moltiplicato per -1.

ESEMPI di Funzione ABS:

```
10 X = ABS ( Y )
10 PRINT ABS ( X * J )
10 IF X = ABS (X) THEN PRINT "POSITIVE"
```

AND

TIPO: Operatore
FORMATO: <espressione> AND <espressione>

Azione: La AND viene usata nelle operazioni Booleane per testare i Bit. Puo' essere usata anche per testare se entrambi gli operatori sono veri. Nell'algebra Booleana, il risultato della AND e' 1 solamente se entrambi gli operatori confrontati sono uguali a 1. Il risultato e'0 se uno solo o entrambi gli operatori sono uguale a 0 (falso).

ESEMPI dell'operazione AND su 1 Bit:

```
    0         1         0         1
AND 0     AND 0     AND 1     AND 1
-----     -----     -----     -----
    0         0         0         1
```

Il COMMODORE 64 esegue l'operazione AND su numeri compresi fra -32768 e +32767. Non si possono usare valori frazionari; i numeri al di fuori dell'intervallo causano un messaggio d'errore ?ILLEGAL QUANTITY
Quando viene convertito nel formato binario, l'intervallo ammesso mette a disposizione 16 bit per ogni numero. Bit corrispondenti vengono confrontati insieme, formando un risultato a 16 bit ancora compreso nell'intervallo di cui sopra.

ESEMPI di operazione AND su 16 Bit:

```
                          17
                     AND 194
                -------------------
                0000000000010001
      AND       0000000011000010
-----------------------------------
(BINARIO)       0000000000000000
-----------------------------------
(DECIMALE)                     0
```

```
                    32007
                AND 28761
            ----------------
            0111110100000111
      AND   0111000001011001
------------------------------
(BINARIO)   0111000000000001
------------------------------
(DECIMALE)            28673


                    -241
                AND 15359
            ----------------
            1111111100001111
      AND   0011101111111111
------------------------------
(BINARIO)   0011101100001111
(DECIMALE)            15119
```

Quando valuta se un numero e' vero o falso, il computer assume che tale numero sia vero purche' il suo valore non sia 0. Quando valuta un confronto, assegna il valore -1 se il risultato e' vero, 0 se il risultato e' falso. Quando valuta vero/falso con AND, ottiene vero come risultato solamente se ogni bit del risultato e'vero.

ESEMPI dell'uso della AND nelle valutazioni VERO/FALSO:

```
50 IF X=7 AND W=3 THEN GOTO 10: REM VERO SOLO SE SONO VERI X=7 E W=3

60 IF A AND Q=7 THEN GOTO 10: REM VERO SE A <> 0 E Q=7
```

ASC

TIPO: Funzione Numerica
FORMATO: ASC (<stringa>)

Azione: ASC ritorna un numero da 0 a 255 corrispondente al valore ASCII Commodore del primo carattere della stringa. La tabella dei valori ASCII Commodore e' riportata nell'appendice C.

ESEMPI della funzione ASC:

```
10 PRINT ASC("Z")
20 X = ASC("ZEBRA")
30 J = ASC(J$)
```

Se nella stringa non e' presente alcun carattere, viene visualizzato il messaggio d'errore ?ILLEGAL QUANTITY Nel terzo esempio sopra, se J$="" la funzione ASC non ha effetto. Le istruzioni GET e GET# leggono un CHR$0 come stringa nulla.

Per eliminare questo inconveniente bisogna aggiungere CHR$(0) alla fine della stringa, come mostra l'esempio seguente:

ESEMPIO di funzione ASC che elimina l'errore ?ILLEGAL QUANTITY:

```
30 J = ASC(J$ + CHR$(0))
```

ATN

TIPO: Funzione Stringa
FORMATO: ATN (<numero>)

Azione: Questa funzione restituisce il valore dell'arcotangente di (numero). Il risultato rappresenta l'angolo (in radianti) la cui tangente e' il numero dato. Il risultato e' sempre compreso fra /2 e +/2.

ESEMPI di funzione ATN:

```
10 PRINT ATN ( 0 )
20 X = ATN ( J ) * 180 / π        :REM CONVERSIONE IN GRADI
```

CHR$

TIPO: Funzione Stringa
FORMATO: CHR$ (<numero>)

Azione: Questa istruzione converte un codice ASCII Commodore nel suo carattere equivalente. Per la lista dei codici e dei caratteri equivalenti si veda l'Appendice C . Il (numero) deve essere compreso tra 0 e 255, altrimenti compare il messaggio **?ILLEGAL QUANTITY**

ESEMPI di funzione CHR$:

```
10 PRINT CHR$(65) : REM 65 =          A MAIUSCOLA

20 A$ = CHR$(13) : REM 13 =          TASTO RETURN
50 A = ASC(A$) : A$ = CHR$(A): REM        CONVERTE IN C64 ASCII, E VICEVERSA
```

CLOSE

TIPO: Istruzione di I/O
FORMATO: CLOSE (<numero di file>)

Azione: Questa istruzione elimina ogni collegamento tra qualsiasi file di dati o canale e un dispositivo. Il numero del file e' lo stesso di quello riportato nell'istruzione di apertura del file o del dispositivo (OPEN) (vedere l'istruzione OPEN e la sezione sulla programmazione dell'input e dell'output). Quando si lavora con dispositivi di memorizzazione come registratori a cassetta ed unita' disco, l'operazione CLOSE memorizza sul dispositivo qualsiasi buffer

non completo. Quando cio' non avviene, il file risulta incompleto su nastro e non leggibile su disco.
Per altri dispositivi, CLOSE non e' altrettanto necessaria, a meno che non liberi memoria per altri file. Per ulteriori informazioni si veda il manuale dei dispositivi esterni.

ESEMPI di istruzioni CLOSE:

```
10 CLOSE 1
20 CLOSE X
30 CLOSE 9 * ( 1 + J )
```

CLR

TIPO: Istruzione
FORMATO: CLR

Azione: Questa istruzione rende disponibile la memoria RAM usata, ma non e' di gran lunga necessaria. Questa istruzione non influenza i programmi BASIC attualmente in memoria, bensi' tutte le variabili, le tabelle, gli indirizzi dei GOSUB, i cicli FOR....NEXT, le funzioni definite dall'utente ed i file, che vengono cancellati dalla memoria, rendendo cosi' disponibile questo spazio per l'inserimento di nuove variabili, ecc.
Nel caso di file su nastro e su disco, l'istruzione CLR non esegue correttamente una CLOSE per quei file. Le informazioni riguardanti sia i file che i buffer non completi sono perse, in quanto il drive del disco "pensa" che il file sia ancora aperto. Per ulteriori informazioni si veda l'istruzione CLOSE.

ESEMPI di istruzione CLR:

```
10 X=25
20 CLR
30 PRINT X

RUN
0

READY
```

CMD

TIPO: Istruzione di I/O
FORMATO: CMD <numero di file> [, <stringa>]

Azione: Questa istruzione trasferisce il dispositivo primario di output dallo schermo TV al file specificato su disco,su nastro, su stampante o su un dispositivo di I/O come il modem. Il numero del file deve essere specificato nella precedente istruzione OPEN. La stringa, se riportata, viene inviata al file. Questa e' utile per intitolare stampati, ecc.

Quando viene eseguito questo comando, qualsiasi istruzione PRINT e comando LIST non visualizza il testo sullo schermo, ma lo invia al file con lo stesso formato. Per riportare di nuovo l'output sullo schermo, il comando PRINT# invia una linea vuota al dispositivo CMD prima di chiuderlo, in modo da disabilitare questo dispositivo dall'attesa di altri dati (dispositivo di "NON RICEZIONE").
Qualsiasi errore di sistema (come ?SYNTAX ERROR) viene visualizzato sullo schermo. Questo non pone i dispositivi nella condizione di non-ricezione, per cui si puo' inviare una linea vuota dopo una condizione d'errore (per ulteriori dettagli si veda il manuale della stampante o dell'unita' disco).

ESEMPI dell'istruzione CMD:

```
OPEN 4, 4: CMD 4, "TITLE" :          LIST:REM LISTA IL PROGRAMMA SU STAMPANTE
PRINT# 4: CLOSE 4: REM              PONE LA STAMPANTE NELLA CONDIZIONE DI NON-
                                    RICEZIONE, CHIUDENDO IL FILE AD ESSA DIRETTO
```

```
10 OPEN 1, 1, 1, "TEST":        REM CREA UN FILE SEQUENZIALE
20 CMD 1:     REM OUTPUT SU NASTRO ANZICHE' SU VIDEO
30 FOR L = 1 TO 100    100
40 PRINT L:      REM INSERISCE IL NUMERO NEL BUFFER DEL NASTRO
50 NEXT
60 PRINT# 1:       REM NON-RICEZIONE
70 CLOSE 1:        REM SCRIVE IL BUFFER NON COMPLETO, TERMINA CORRETTAMENTE
```

CONT

TIPO: Comando
FORMATO: CONT

Azione: Questo comando pone di nuovo in esecuzione un programma interrotto da STOP, da END o dalla pressione del tasto RUN/STOP. Il programma riparte dal punto esatto dal quale era stato interrotto.
Mentre il programma e' fermo, l'utente puo' controllare e cambiare qualsiasi variabile, o anche l'intero programma. Durante l'esame o la correzione del programma, l'istruzione STOP puo' essere sistemata in punti strategici per permettere di esaminare le variabili e di testare il flusso del programma. Dall'editazione del programma risulta il messaggio d'errore: **CAN'T CONTINUE** (anche se si e' appena premuto RETURN con il cursore posizionato su una linea non modificata), se il programma si e' fermato a causa di un errore interno o dell'Utente avvenuto prima di digitare CONT per far ripartire il programma.

ESEMPI dell'istruzione CONT:

```
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT PI
40 C=C+4:GOTO 20
```

Questo programma calcola il valore di PI. Se si prova a lanciare (RUN) il programma e poco dopo si preme il tasto RUN/STOP, viene visualizzato:

BREAK IN 20 NOTA: potrebbe comparire un qualsiasi altro numero)

Per vedere che cosa e' successo si puo' usare il comando PRINT C. Quindi si puo' usare CONT per ripartire da dove il COMMODORE 64 si era fermato.

COS

TIPO: Funzione
FORMATO: COS (<espressione numerica>)

Azione: - Questa funzione matematica calcola il coseno del numero, dove il numero rappresenta un angolo in radianti.

ESEMPIO di funzione COS:

```
10 PRINT COS ( 0 )
20 X = COS (Y * π / 180)     :REM CONVERTE I GRADI IN RADIANTI
```

DATA

TIPO: Istruzione
FORMATO: DATA (<lista di costanti>)

Azione: - L'istruzione DATA memorizza informazioni all'interno di un programma. Il programma usa le informazioni per mezzo dell'istruzione READ, che preleva le costanti successive dalle istruzioni DATA.
Le istruzioni DATA non devono essere eseguite dal programma ma devono essere solamente presenti. Di solito, percio', sono situate alla fine del programma.
Tutte le istruzioni DATA di un programma vengono trattate come liste continue. Il dato e' letto da sinistra a destra e dalla linea con numero inferiore fino alla linea con numero maggiore. Se l'istruzione READ incontra un dato che non risponde al tipo richiesto (se ad esempio vuole un numero ed incontra una stringa) si causa un messaggio d'errore.
Qualsiasi carattere puo' essere trattato come dato, ma talvolta la voce del dato deve essere racchiusa tra i doppi apici ("") - caratteri come la virgola (,),i due punti (:), gli spazi vuoti, le lettere ottenute tenendo premuto il tasto SHIFT, la grafica e i caratteri di controllo cursore devono essere racchiusi tra i doppi apici.

ESEMPI di istruzione DATA:

```
10 DATA 1, 10, 5, 8
20 DATA JOHN, PAUL, GEORGE, RINGO
30 DATA  . "CARA MARY, COME STAI, AMORE, BILLY"
40 DATA -1.7E-9, 3.33
```

DEF FN

TIPO: Istruzione
FORMATO: DEF FN <nome> (<variabile>) = <espressione>

Azione: - Questa istruzione imposta una funzione definita dall'utente utilizzabile successivamente nel programma. La funzione puo' essere costituita da qualsiasi formula matematica. Le funzioni definite dall'utente consentono di risparmiare spazio in quei programmi dove una lunga formula viene usata numerose volte. La formula necessita di essere specificata una sola volta, nell'istruzione di definizione; successivamente, viene abbreviata come nome di funzione. DEF FN deve essere eseguita una sola volta; tutte le esecuzioni successive sono ignorate.
Il nome della funzione e' costituito dalle due lettere FN seguite dal nome di una variabile, lungo uno o due caratteri, di cui il primo deve essere una lettera ed il secondo una lettera o un numero.

ESEMPI di istruzione DEF FN:

```
10 DEF FN A (X) = X + 7
20 DEF FN AA (X) = Y * Z
30 DEF FNA9 (Q) = INT( RND( 1)* Q+ 1)
```

La funzione viene richiamata piu' avanti nel programma usando il nome della funzione. Quest'ultimo viene usato come qualsiasi altra variabile; il suo valore viene calcolato automaticamente.

ESEMPI dell'uso di FN:

```
40 PRINT FN A (9)
50 R = FNAA (9)
60 G = G + FN A9 (10)
```

Nella linea 50, il numero 9 dentro le parentesi non influenza il risultato della funzione, perche' la funzione definita nella linea 20 non usa la variabile fra parentesi. Il risultato e' Y volte Z, senza tener conto del valore di Z. Nelle altre funzioni fra parentesi, invece, il numero 9 influenza il risultato.

DIM

TIPO: Istruzione
FORMATO: DIM <variabile> (<sottoscritti>),
[<variabile>(<sottoscritti>).....]

Azione: - Questa istruzione definisce un vettore o una matrice di variabili. Cio' permette di usare il nome della variabile con un sottoscritto; quest'ultimo punta all'elemento che sta per essere usato. Il numero piu' basso in un vettore e' zero, il piu' alto e' il numero riportato nell'istruzione DIM, che ha un massimo di 32767.
L'istruzione DIM deve essere eseguita una sola volta per ciascun

vettore, altrimenti si verifica l'errore **REDIM'D ARRAY** . Percio', la maggior parte dei programmi esegue tutte le istruzioni DIM all'inizio. Il numero delle dimensioni puo' essere qualunque; i sottoscritti del vettore sono al massimo 255. Tutto e' limitato solamente dalla RAM disponibile per le variabili. I vettori possono essere costituiti da normali variabili numeriche, da stringhe o da numeri interi. Se le variabili sono di tipo stringa si usa il segno "$" dopo il nome della variabile, se sono numeri interi si usa il segno "%". Se un vettore a cui si fa riferimento in un programma non e' mai stato dimensionato, viene dimensionato automaticamente a 11.

ESEMPIO di istruzione DIM:

```
10 DIM A ( 100 )
20 DIM Z ( 5, 7), Y ( 3, 4, 5)
30 DIM Y7% ( Q )
40 DIM PH$ (1000)
50 F (4) =9: REM ESEGUE AUTOMATICAMENTE DIM F(10)
```

ESEMPIO di SEGNAPUNTI DEL CALCIO ottenuto usando DIM:

```
10 DIM S(1,5), T$(1)
20 INPUT "NOMI DELLE SQUADRE";T$(0),T$(1)
30 FOR Q=1 TO 5: FOR T=0 TO 1
40 PRINT T$(T),     "PUNTEGGIO PARZIALE"Q
50 INPUT S(T,Q): S(T,0)= S(T,0)+ S(T,Q)
60 NEXT T,Q
70 PRINT CHR$(147)      "TABELLONE"
80 PRINT "TEMPO"
90 FOR Q=1 TO 5
100 PRINT TAB( Q*2 +9) Q;
110 NEXT: PRINT TAB(15)          "TOTALE"
120 FOR T=0 TO 1: PRINT T$(T);
130 FOR Q=1 TO 5
140 PRINT TAB(Q*2 +9) S(T,Q);
150 NEXT: PRINT TAB(15) S(T,0)
160 NEXT
```

COME CALCOLARE LA MEMORIA USUFRUITA DA DIM:

5 bytes per il nome del vettore
2 bytes per ciascuna dimensione
2 bytes/elemento per le variabili intere
5 bytes/elemento per le variabili normali numeriche
3 bytes/elemento per le variabili stringa
1 byte per ogni carattere contenuto in ogni elemento stringa

END

TIPO: Istruzione
FORMATO: END

Azione: - Questa istruzione termina l'esecuzione di un programma e fa visualizzare il messaggio READY rimandando il controllo all'Utente. All'interno di un programma ci possono essere diverse istruzioni END.

Mentre non e' affatto necessario includere piu' istruzioni END, e' raccomandabile invece che un programma termini con una di esse piuttosto che continuare l'elaborazione oltre l'ultima linea.
L'istruzione END e' simile allo STOP. L'unica differenza e' STOP visualizza il messaggio **BREAK IN LINE XX** , mentre END visualizza solo READY. Entrambi le istruzioni permettono al Computer di riprendere l'esecuzione dopo che si e' digitato il comando CONT.

ESEMPIO di istruzione END:

```
10 PRINT  "VUOI ESEGUIRE QUESTO PROGRAMMA"
20 INPUT A$
30 IF A$ = "NO"THEN END
40 ARRESTO DEL PROGRAMMA
999 END
```

EXP

TIPO: Funzione Numerica
FORMATO: EXP (<numero>)

Azione: - Questa funzione matematica calcola il valore ottenuto dall'elevamento a potenza della costante e (=2,71828183) per il numero dato Un valore maggiore di 88,0296919 causa il messaggio d'errore: ?OVERFLOW

ESEMPIO di funzione EXP:

```
10 PRINT EXP (1)
20 X = Y * EXP (Z * Q)
```

FN

TIPO: Funzione Numerica
FORMATO: FN <nome> (<numero>)

Azione: - Questa funzione si riferisce alla formula specificata dal nome vista in precedenza. Il numero (se riportato) viene inserito al suo posto e la formula calcolata. Il risultato e' un valore numerico.
Questa funzione puo' essere usata in modo diretto, basta che sia stata eseguita l'istruzione di DEFinizione.
Se si esegue una FN prima dell'istruzione DEF che la definisce, si verifica il messaggio d'errore UNDEF'D FUNCTION

ESEMPIO di funzione FN (definita dall'Utente):

```
PRINT FN A ( Q )
1100 J = FN J (7) + FN J (9)
9990 IF FN B7 (I+1)= 6 THEN END
```

FOR...TO...[STEP...]

TIPO: Istruzione
FORMATO: FOR <variabile> = <inizio> TO <fine> [STEP <incremento>]

Azione: - Questa e' un'istruzione speciale del BASIC che permette di usare facilmente una variabile come contatore. Si devono specificare alcuni parametri: il nome della variabile reale, il suo valore d'inizio, il limite del conteggio, e l'incremento da usare durante ciascun ciclo.
Quello seguente e' un semplice programma BASIC che conta da 1 a 10, stampando ciascun numero e terminando quando l'elaborazione e' completa. Non usa alcuna istruzione FOR:

```
100 L = 1
110 PRINT L
120 L = L + 1
130 IF L <= 10 THEN 110
140 END
```

LO STESSO PROGRAMMA, USANDO L'ISTRUZIONE FOR, DIVENTA:

```
100 FOR L = 1 TO 10
110 PRINT L
120 NEXT L
130 END
```

Come si puo' vedere, il programma e' piu' corto e permette di capire piu' facilmente l'uso dell'istruzione FOR.
Quando viene eseguita l'istruzione FOR, hanno luogo numerose operazioni. Nella ‹variabile› che viene usata come contatore viene sistemato il valore di ‹partenza›. Nell'esempio precedente la variabile L assume il valore 1.
Quando viene raggiunta l'istruzione NEXT, la ‹variabile› viene aumentata del valore dell'‹incremento›. Se non e' incluso lo STEP, si assume l'‹incremento› uguale a +1. La prima volta, il programma precedente, in riferimento alla linea 120, aggiunge 1 a L; il nuovo valore di L e' quindi 2.
A questo punto, il valore della ‹variabile› e' confrontato con il ‹limite›. Se il ‹limite› non e' stato ancora raggiunto, il programma passa alla linea successiva a quella dell'istruzione FOR. Nel nostro caso, il valore contenuto in L (2) e' minore del limite (10), percio' il programma passa alla linea 110.
Nel caso in cui il valore della ‹variabile› superi il ‹limite›, il ciclo viene interrotto e il programma riprende dall'istruzione successiva a NEXT. Nel nostro caso, il valore di L raggiungera' 11 e allora, essendo superato il limite (10), il programma passa alla linea 130.
Quando il valore dell'‹incremento› e' positivo, la ‹variabile› deve diventare maggiore del ‹limite›, quando e' negativo deve diventare minore del ‹limite›.

NOTA: Un Loop viene sempre eseguito almeno una volta

ESEMPI dell'istruzione FOR...TO...STEP:

```
100 FOR L = 100 TO 0 STEP -1
100 FOR L = PI TO 6*π STEP .01
100 FOR AA = 3 TO 3
```

FRE

TIPO: Funzione
FORMATO: FRE (<variabile>)

Azione: - Questa funzione informa sulla quantita' di memoria RAM disponibile per programmi e variabili. Se un programma tenta di usare piu' spazio di quello disponibile, si verifica l'errore **OUT OF MEMORY**
Il numero tra parentesi puo' essere qualsiasi valore, e non viene usato nel calcolo.

NOTA: Se il risultato di FRE e' negativo aggiungere 65536 al risultato per avere il numero di byte disponibili in memoria

ESEMPIO di funzione FRE:

```
PRINT FRE ( 0 )
10 X = ( FRE ( K ) - 1000 ) / 7
950 IF FRE ( 0 ) < 100 THEN PRINT    "NON E' SUFFICENTE"
```

NOTA: La seguente espressione riporta sempre la quantita' attuale di memoria RAM disponibile: PRINT FRE(0)-(FRE(0)<0)*65536

GET

TIPO: Istruzione
FORMATO: GET <lista di variabili>

Azione: - Questa istruzione legge tutti i tasti digitati dall'Utente. Se l'utente sta digitando, i caratteri vengono memorizzati nel buffer della tastiera del COMMODORE 64. Il Buffer puo' contenere fino a 10 caratteri; qualsiasi altro carattere digitato oltre il decimo viene perso. Via via che GET legge un carattere, si libera il posto per un altro carattere.
Se l'istruzione GET specifica dati numerici, e l'Utente digita tasti diversi da numeri, compare il messaggio d'errore **?SYNTAX ERROR** . Per essere sicuri, conviene leggere i tasti come stringhe e convertirli poi in numeri.
L'istruzione GET puo' essere usata per imporre alcuni limiti all'istruzione INPUT. Per ulteriori informazioni, vedere la sezione sull'uso dell'istruzione GET nella parte relativa alle Tecniche di Programmazione.

ESEMPI di istruzione GET:

```
10 GET A$: IF A$ = "" THEN 10: REM      RESTA FERMO A QUESTA ISTRUZIONE FINCHE'
   HIT                                  NON SI BATTE UN TASTO
20 GET A$, B$, C$, D$, E$:    REM LEGGE 5 TASTI
30 GET A, A$
```

GET

TIPO: Istruzione
FORMATO: GET# <numero file>, <lista variabili>

Azione: - Questa istruzione legge caratteri, uno alla volta, da un dispositivo o file specifico. Funziona come l'istruzione GET, solo che i dati provengono da dispositivi diversi dalla tastiera. Se non viene ricevuto alcun carattere, la variabile e' impostata a stringa vuota (uguale a"") o a 0 per variabili numeriche. I caratteri usati per separare i dati nei file, come la virgola (,) o il codice del tasto RETURN (codice ACII=13), vengono ricevuti come qualsiasi altro carattere.

Quando GET # viene usata con il dispositivo #3 (schermo TV) essa legge un carattere alla volta dallo schermo. Ciascuna istruzione GET # sposta verso destra il cursore di 1 posizione. Il carattere posto alla fine della linea logica e' cambiato in CHR$(13), cioe' al codice del tasto RETURN.

ESEMPI dell'istruzione GET#

```
5 GET# 1, A$
10 OPEN 1, 3: GET# 1, Z7$
20 GET# 1, A, B, C$, D$
```

GOSUB

TIPO: Istruzione
FORMATO: GOSUB <numero di linea>

Azione: - Questa e' una forma particolare dell'istruzione GOTO, che implica una differenza importante: GOSUB si ricorda dove deve ritornare il controllo del programma. Quando nel programma viene raggiunta l'istruzione RETURN (differente dal tasto RETURN della tastiera), il controllo ritorna alla linea immediatamente successiva all'istruzione GOSUB di origine.

Il maggior uso di una sotto-procedura (GOSUB in realta' significa vai alla sotto-procedura) si ha quando un piccolo segmento di programma viene usata da diversi altri segmenti del programma. Usando le sotto-procedure, piuttosto che ripetere le stesse linee in diversi posti del programma, si risparmia molto spazio di memoria. In questo caso GOSUB e' simile a DEF FN. DEF FN permette di salvare lo spazio usando una formula, mentre GOSUB salva spazio usando una routine di molte linee. Questo e' un programma inefficiente che non usa la GOSUB:

```
100 PRINT"QUESTO PROGRAMMA SCRIVE"
110 FOR L = 1 TO 500 : NEXT
120 PRINT "LENTAMENTE SULLO SCHERMO"
130 FOR L = 1 TO 500 : NEXT
140 PRINT "USANDO UN SEMPLICE LOOP"
150 FOR L = 1 TO 500 : NEXT
160 PRINT"COME RITARDO DI TEMPO"
170 FOR L = 1 TO 500 : NEXT
```

Lo stesso programma che usa la GOSUB diventa:

```
100 PRINT'"QUESTO PROGRAMMA SCRIVE"
110 GOSUB 200
120 PRINT "LENTAMENTE SUL VIDEO"
130 GOSUB 200
140 PRINT "USANDO UN SEMPLICE LOOP"
150 GOSUB 200
160 PRINT'"COME RITARDO DI TEMPO"
170 GOSUB 200
180 END
200 FOR L = 1 TO 500 : NEXT
210 RETURN
```

Ogni volta che il programma esegue la GOSUB, sia il numero di linea che la posizione nel programma vengono salvati in un'area speciale detta "STACK" (pila) che occupa 256 bytes della memoria. Questo limita l'ammontare dei dati che possono essere memorizzati sullo STACK. Percio', il numero di indirizzi di ritorno della sottoprocedura che possono essere memorizzarti e' limitato, e si deve fare attenzione che ciascuna GOSUB abbia la relativa RETURN, altrimenti l'elaborazione prosegue fuori della memoria, anche se si hanno ancora bytes liberi di memoria.

GOTO

TIPO: Istruzione
FORMATO: GOTO <numero di linea> o GO TO <numero di linea>

Azione: - Questa istruzione permette al programma BASIC di eseguire le linee fuori dall'ordine numerico. La parola GOTO, seguita da un numero, fa saltare il programma alla linea indicata da quel numero. Una GOTO senza numero di linea equivale ad GOTO 0. Dopo l'istruzione GOTO ci deve sempre essere un numero di linea. Con GOTO e' possibile creare dei cicli che non hanno mai fine. Il piu' semplice esempio di quanto detto e' una linea che rimanda a se stessa, come 10 GOTO 10. Questi tipi di cicli possono essere fermarti usando il tasto RUN/STOP da tastiera.

ESEMPI d'istruzione GOTO

```
GOTO 100
10 GO TO 50
20 GOTO 999
```

IF...THEN...

TIPO: Istruzione
FORMATO: IF <espressione> THEN <numero di linea>
IF <espressione> THEN <istruzione>
IF <esprezzione> GOTO <numero di linea>

Azione: - Questa istruzione rende il BASIC "intelligente", dandogli l'abilita' di valutare condizioni e di compiere azioni diverse in base al risultato.
La parola IF e' seguita da un'espressione che puo' includere variabili, stringhe, numeri, confronti e operatori logici. La parola THEN compare nella stessa linea ed e' seguita dal numero di linea oppure da una o piu' istruzioni BASIC. Quando l'espressione risulta falsa, tutto cio' che si trova dopo la parola THEN su quella linea viene ignorato e l'esecuzione continua con la successiva linea di programma. Se il risultato e' vero, il programma salta al numero di linea dopo la parola THEN, oppure esegue qualsiasi altra istruzione BASIC che trova su quella linea.

ESEMPI d'istruzione IF...GOTO...

```
100 INPUT"BATTI UN NUMERO";N
110 IF N <= 0 GOTO 200
120 PRINT'"RADICE QUADRATA=":SQR(N)
130 GOTO 100
200 PRINT'"IL NUMERO DEVE ESSERE>0"
210 GOTO 100.
```

Questo programma stampa la radice quadrata di qualsiasi numero positivo. L'istruzione IF e' qui usata per riscontrare la validita' del risultato di INPUT. Quando il risultato di N 0 e' vero, il programma salta alla linea 200, quando e' falso sara' eseguita la linea 120. Si noti che THEN..GOTO.. non e' necessaria come IF.. THEN, come si vede nella linea 110 dove GOTO 200 significa THEN..GOTO 200.

ESEMPIO dell'istruzione IF...THEN...

```
100 FOR L = 1 TO 100
110 IF RND(1) < .5 THEN X = X+ 1 : GOTO 130
120 Y = Y+ 1
130 NEXT L
140 PRINT"IN TESTA E'"X
150 PRINT"IN CODA E'"Y
```

IF nella linea 110 controlla un numero casuale per vedere se e' < di .5. Quando il risultato e' vero, viene eseguita l'intera serie di istruzioni che seguono la parola THEN: per prima cosa X viene incrementata di 1, quindi il programma salta alla linea 130. Se il risultato e' falso, il programma passa all'istruzione successiva sulla linea 120.

INPUT

TIPO: Istruzione
FORMATO: INPUT [«<descrizione>] <lista di variabili>

Azione: - Questa istruzione permette al programmatore di "alimentare" l'informazione nel computer. Quando viene eseguita, questa istruzione stampa un punto interrogativo (?) sullo schermo, e posiziona il cursore uno spazio a destra del punto interrogativo. A questo punto il Computer aspetta, Facendo lampeggiare il cursore che l'operatore

digiti la risposta e prema il tasto RETURN .
La parola INPUT puo' essere seguita da qualsiasi testo compreso fra i doppi apici (""). Questo testo viene stampato sullo schermo e alla sua destra compare un punto interrogativo.
Dopo il testo, viene inserito un punto e virgola (;) e il nome di una o piu' variabili separate da virgola (,). La variabile indica dove il Computer memorizza l'informazione digitata dall'operatore. La variabile puo' essere qualsiasi nome legale di variabile, per cui si possono avere numerosi nomi di variabili, uno per ciascun INPUT diverso.

ESEMPI dell'istruzione INPUT:

```
100 INPUT A
110 INPUT B, C, D
120 INPUT'"DESCRIZIONE";E
```

Quando questo programma "gira", appar" un punto interrogativo per avvertire l'operatore che il COMMODORE 64 sta aspettando un INPUT alla linea 100. Qualsiasi numero digitato va a finire in A, per poter essere riutilizzato nel programma. Se la risposta digitata non e' un numero, appare il messaggio **?REDO FROM START** , significando che una stringa e' stata ricevuta quando invece ci si aspettava un numero. Se l'operatore batte RETURN senza aver digitato nulla, il valore della variabile non cambia
A questo punto compare un ? alla linea 110 . Se digitiamo solamente un numero e battiamo RETURN il COMMODORE 64 visualizza due ??, il cui significato e' la richiesta di piu' INPUT. Si possono digitare tanti INPUT quanti necessari, separati da virgole, che prevengono l'apparizione del doppio ??. Se vengono digitati piu' dati di quanti ne richiede l'istruzione INPUT, compare il messaggio **?EXTRA IGNORED** , che significa che le voci digitate in piu' non vengono messe in alcuna variabile.
La linea 120 visualizza la parola DESCRIZIONE prima che compaia il ? . Il punto e virgola e' richiesto tra la descrizione e la lista di variabili. L'istruzione INPUT non puo' mai essere usata fuori da un programm. Il COMMODORE 64 richiede spazio per un buffer per le variabili di INPUT, lo stesso spazio usato per i comandi.

INPUT#

TIPO: Istruzione di I/O
FORMATO: INPUT# <numero di file>, <lista di variabili>

Azione: - Questo e' il metodo piu' veloce e piu' facile per recuperare i dati memorizzati su un file su disco o su nastro. Il dato e' nella forma di variabile intera lunga massimo 80 caratteri; questa forma e' opposta al metodo di un carattere alla volta della GET#. Per prima cosa, il file deve essere aperto, poi INPUT# puo' riempire le variabili.
Il comando INPUT# assume che una variabile e' terminata quando legge un codice RETURN (CHR$(13)), una virgola(,), un punto e virgola(;) o due punti(:). Se necessario, durante la scrittura, si possono usare le virgolette ("") per racchiudere questi caratteri (si veda l'istruzione PRINT #). Se la variabile e' di tipo numerico e viene ricevuto un dato

non numerico, si verifica l'errore **BAD DATA** . Con la INPUT# si possono leggere stringhe lunghe fino a 80 caratteri oltre i quali compare l'errore **STRING TOO LONG** .
Quando questa istruzione viene usata con il dispositivo #3 (lo schermo), viene letta un'intera linea logica e spostato il cursore sulla linea successiva.

ESEMPI d'istruzione INPUT#:

```
10 INPUT# 1, A
20 INPUT# 2, A$, B$
```

INT

TIPO: Funzione Intera
FORMATO: INT (<espressione numerica>)

Azione: - Ritorna il valore intero di un'espressione. Se l'espressione e' positiva la parte frazionaria, viene perduta, se l'espressione e'negativa, qualsiasi frazione provoca il ritorno numero intero minore piu' vicino.

ESEMPI di funzione INT:

```
120 PRINT INT(99.4343), INT(-12.34)
 99        -13
```

LEFT$

TIPO: Funzione Stringa
FORMATO: LEFT$ (<stringa>, <intero>)

Azione: - Ritorna una stringa comprendente i caratteri ‹interi› piu' a sinistra della ‹stringa›. Il valore dell'argomento intero deve essere compreso fra range 0 e 255. Se l'intero e' maggiore della lunghezza della stringa, il risultato di ritorno e' l'intera stringa. Se si usa il valore intero 0, ritorna una stringa nulla (cioe' di lunghezza zero).

ESEMPI della funzione LEFT$:

```
10 A$ = "COMMODORE COMPUTERS"
20 B$ = LEFT$(A$,9): PRINT B$
RUN
COMMODORE
```

LEN

TIPO: Funzione Intera
FORMATO: LEN (<stringa>)

Azione: - Ritorna il numero dei caratteri dell'espressione stringa. Vengono contati anche i caratteri non stampati e i blank.

ESEMPI della funzione LEN:

```
CC$ = "COMMODORE COMPUTER": PRINT LEN(CC$)
 18
```

LET

TIPO: Istruzione
FORMATO: [LET] <variabile> = <espressione>

Azione: - L'istruzione LET puo' essere usata per assegnare un valore ad una variabile. Ma la parola LET e' opzionale, percio' i programmatori piu' esperti lasciano fuori LET perche' e' sottointesa e occupa memoria. Il segno di uguale (=) e' sufficiente quando si assegna il valore di un espressione al nome di una variabile.

ESEMPI d'istruzione LET:

```
10 LET D= 12        (e' lo stesso di D=12)
20 LET E$ = "ABC"
30 F$ = "PAROLE"
40 SUM$ = E$ + F$ (SUM$ e' uguale a ABCPAROLE)
```

LIST

TIPO: Comando
FORMATO: LIST [(<prima linea>) - (<ultima linea>)]

Azione: - Il comando LIST permette di visualizzare le linee del programma BASIC che si trova attualmente nella memoria del COMMODORE 64. Cio' consente di sfruttare la potenza dell'editor di schermo del computer per editare, velocemente e facilmente i programmi listati. Il comando di sistema LIST visualizza tutto o parte del programma che si trova attualmente in memoria sul dispositivo standard di OUTPUT. La LIST e' normalmente diretta allo schermo, si puo' usare CMD per riportare l'OUTPUT su un dispositivo esterno quale la stampante o il disco. Il Comando LIST puo' essere inserito in un programma, ma il BASIC ritorna al messaggio di sistema READY, dopo che una LIST e' stata eseguita. Quando si lista un programma sullo schermo, lo "scrolling" dal basso in alto puo' essere rallentato premendo il tasto di controllo CTRL. Digitando il tasto RUN/STOP si provoca l'interruzione dell'effetto di LIST.

Se nel comando di LIST non viene riportato alcun numero, sul video viene listato l'intero programma. Se viene specificato il numero di linea, seguito dal segno meno (-), vengono listate tutte le linee esistenti da quel numero di linea in poi. Se, invece, viene

specificato il numero, preceduto dal (-), allora vengono listate tutte le linee dall'inizio del programma fino a quella indicata dal numero stesso. Se sono indicati entrambi i numeri vengono listate le linee comprese nell'intervallo specificato dai due numeri di linea.

ESEMPI di comandi LIST:

```
LIST           lista il programma in memoria
LIST 500       lista la linea 500
LIST 150-      lista tutte le linee dalla 150 fino alla fine
LIST-1000      lista tutte le linee fino alla 1000
LIST150-1000   lista dalla linea 150 alla 1000 inclusa
```

```
10 PRINT ' "QUESTA E' LA LINEA 10"
20 LIST     LIST usata in MODO PROGRAMMA
30 PRINT  "QUESTA E' LA LINEA 30
```

LOAD

TIPO: Comando
FORMATO: LOAD [«<nome del file>»] [, <canale>] [, <indirizzo>]

Azione: - L'istruzione LOAD legge il contenuto su nastro o su disco e lo trasferisce in memoria. Si puo' cosi' usare l'informazione caricata o cambiarla. Il numero del canale e' opzionale, ma se esso viene omesso il computer assume per default il numero 1, cioe' il registratore. L'unita' disco porta normalmente il numero 8. LOAD chiude tutti i file aperti e, se usata in modo diretto, esegue un CLR (azzeramento) prima di leggere il programma. Se la LOAD viene eseguita da programma, il programma viene subito fatto "girare". Cio' significa che si puo' usare la LOAD per "concatenare" piu' programmi assieme. Nessuna delle variabili viene azzerata durante l'operazione di concatenamento.

Se si usa un nome di file come mezzo di riconoscimento, il primo file corrispondente al nome usato viene caricato in memoria. L'asterisco tra gli apici ("*") fa si' che venga caricato il primo programma della directory. Se il nome del file usato non e' presente sul disco oppure non e' il nome di un file di programma, compare il messaggio d'errore: **?FILE NOT FOUND**

Quando si caricano programmi da nastro, il (nome del file) puo' essere omesso; in tal caso, viene caricato il prossimo file programma. Quando viene premuto il tasto PLAY, il COMMODORE 64 azzera lo schermo portandolo al colore di contorno. Quando il file programma viene trovato, lo schermo viene azzerato e viene visualizzato il messaggio "FOUND".

Quando viene premuto il tasto  , oppure dopo una pausa di 15 secondi, il file viene caricato. Se si e' premuto il tasto SPACE BAR, il file attualmente in osservazione viene saltato e si tenta di caricare il file successivo. I programmi sono caricati in memoria a partire dalla locazione 2048, a meno che non venga usato l'(indirizzo) indiretto secondario 1, nel qual caso il programma viene caricato nelle locazioni di memoria dalle quali e' stato salvato.

ESEMPI del comando LOAD:

```
LOAD                    Legge il prossimo programma su nastro
LOAD A$                 Usa il nome contenuto in AA  per caricarlo
LOAD"*",8               Carica il primo programma da disco
LOAD"",1,1              Osserva il primo programma su nastro e lo  carica
                        nella stesso segmento di memoria di provenienza.

LOAD"STAR TREK"         Carica un file da nastro
PRESS PLAY ON TYPE
FOUND STAR TREK
LOADING
READY

LOAD"FUN",8             Carica un file da disco
SEARCHING FOR FUN
LOADING
READY

LOAD"GIOCO 1",8,1       Carica  un file  nella specifica  locazione di
SEARCHING FOR GIOCO 1   memoria da cui il  programma e' stato  salvato
                        su disco.
LOADING
READY
```

LOG

TIPO: Funzione Reale
FORMATO: LOG (<numerica>)

Azione: - Ritorna il logaritmo naturale (logaritmo in base e) dell'argomento. Se il valore dell'argomento e' 0 o negativo il BASIC emette il messaggio d'errore: **?ILLEGAL QUANTITY**

ESEMPIO di funzione LOG:

```
25 PRINT LOG(45/7)
 1.86075234

10 NUM=LOG(ARG)/LOG(10) (calcola il LOGARITMO in base 10 di ALG)
```

MID$

TIPO: Funzione Stringa
FORMATO: MID$ (<stringa>, <espressione numerica-1>
[, <espressione numerica-2>)

Azione: - La funzione MID$ ritorna una sottostringa prelevata dalla ‹stringa› argomento piu' lunga. La posizione di partenza della sottostringa e'definita dall'argomento ‹espressione numerica 1›, la lunghezza della sottostringa dell'argomento ‹espressione numerica 2›.
Entrambi gli argomenti numerici possono avere valori compresi fra 0 e 255. Se l'‹espressione 1› e' maggiore della lunghezza della ‹stringa› o se l'‹espressione 2› e' zero, allora MID$ riporta una stringa nulla. Se l'‹espressione 1› viene omessa, allora il computer assume come

lunghezza della sottostringa la differenza tra la lunghezza della stringa ed il valore dell'⟨espressione numerica 2⟩. Se la stringa sorgente e' piu' corta dell'⟨espressione 2⟩, allora viene usata l'intera stringa, dalla posizione di inizio fino alla fine.

ESEMPI di funzione MID$:

```
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$ + MID$(B$, 8, 8)
```

NEW

TIPO: Comando
FORMATO: NEW

Azione: - Il comando NEW viene usato per cancellare il programma che si trova in memoria ed azzerare tutte le variabili. NEW deve essere usato in modo diretto prima di digitare un nuovo programma, per azzerare la memoria. NEW puo' essere usata anche da programma, ma occorre tener presente che cancella tutto cio' che e' stato fatto prima e che si trova ancora nella memoria del computer. Cio' puo' creare molti problemi durante la correzione di un programma.

ATTENZIONE: Non cancellare un vecchio programma prima di digitarne uno nuovo puo' causare confusione fra i due programmi.

ESEMPIO di comando NEW:

```
NEW      (Azzera tutte le variabili del programma)

10 NEW   (Esegue una NUOVA Operazione e ARRESTA il programma)
```

NEXT

TIPO: Istruzione
FORMATO: NEXT [<contatore>] [, <contatore>].....

Azione: - L'istruzione NEXT viene usata con FOR per stabilire la fine di un ciclo FOR...NEXT. NEXT non deve essere posta necessariamente alla fine del gruppo di istruzioni appartenenti al ciclo, ma e' sempre l'ultima istruzione eseguita dal ciclo. Il ⟨contatore⟩ e' il nome della variabile a indice del ciclo usata con FOR per dare inizio al ciclo. Una singola NEXT puo' arrestare numerosi cicli nidificati. quando sia seguita dai relativi nomi delle variabili ⟨contatore⟩ di ciascuna FOR. Per realizzare cio', i nomi delle variabili devono comparire in ordine dal ciclo piu' interno al ciclo piu' esterno. Quando usia una singola NEXT per incrementare e fermare numerose variabili contatore,ciascun nome della variabile deve essere separato da una virgola. I cicli possono essere nidificati fino a 9 livelli.Se si omette il contatore della variabile (variabili), viene incrementeto il contatore associato alla FOR del corrente livello di nidificazione. Quando viene raggiunta la NEXT, il valore del contatore viene

incrementato di 1 o del valore specificato dallo STEP opzionale. Viene quindi testato di nuovo il valore finale per vedere se il ciclo deve essere interrotto o meno. Un ciclo viene arrestato quando il valore del contatore, aggiornato con l'ultima NEXT, risulta maggiore del valore finale.

ESEMPIO di istruzione NEXT:

```
10 FOR J=1 TO 5: FOR K = 10 TO 20: FOR N = 5 TO -5 STEP -1
20 NEXT N, K, J     :REM       ARRESTA I CICLI NIDIFICATI

10 FOR L = 1 TO 100
20 FOR M = 1 TO 10
30 NEXT M
400 NEXT L                     SI NOTI COME I CICLI NON SI INCROCIANO

10 FOR A = 1 TO 10
20 FOR B. = 1 TO 20
30 NEXT
40 NEXT                        NOTARE CHE NON E' NECESSARIO ALCUN NOME DI
VARIABILE
```

NOT

TIPO: Operatore Logico
FORMATO: NOT <espressione>

Azione: - L'operatore logico NOT fa il "complemento" del valore di ciascun bit nel suo singolo operando, producendo come risultato intero il "complemento a 2". In altre parole, NOT significa "Se non e'....". Quando si ha a che fare con un numero reale, gli operandi vengono convertiti in interi e qualsiasi frazione viene perduta. L'operatore NOT puo' essere usato anche in un confronto per trasformare il valore vero/falso, ottenuto come risultato di un test relazionale; quindi cambia il significato del confronto. Nel primo esempio seguente, se il "complemento a 2 "di"AA"e' uguale a "BB", e se "BB" NON e' uguale a "CC" allora l'espressione e' vera.

ESEMPI dell'operatore NOT:

```
10 IF NOT AA = BB AND NOT(BB = CC) THEN . . . .

NN% = NOT 96: PRINT NN%
-97
```

NOTA: Per trovare il valore di NOT usare l'espressione x=(-(x+1)) (il complemento a 2 di qualsiasi intero si ottiene eseguendo il complemento al bit e sommando 1).

ON

TIPO: Istruzione
FORMATO: ON <variabile> GOTO/GOSUB <numero di linea>
(, <numero di linea>).....

Azione: - L'istruzione ON viene usata per effettuare un GOTO ad uno dei molti numeri di linea, in base al valore di una variabile compreso tra 0 e il numero delle linee date. Se il valore non e' intero, la parte frazionaria viene persa. Per esempio, se la variabile ha valore 3, la ON esegue il GOTO al terzo numero di linea della lista. Se il valore della variabile e' negativo compare il messaggio BASIC **?ILLEGAL QUANTITY** . Se il numero e' zero o maggiore dei numeri delle linee riportate nella lista,il progrsamma "ignora" l'istruzione e continua con l'istruzione che segue la ON.
La ON e' in realta' un sottouso dell'istruzione IF...THEN. Invece di usare numerose istruzioni IF, ciascuna delle quali manda al programma specificato dalla linea, si usa la ON, che prende in considerazione una lista di numeri di linea a cui saltare. Osservando il primo esempio, si puo' notare che l'istruzione ON prende il posto di 4 istruzioni IF...THEN.

ESEMPI di istruzione ON:

```
ON -(A=7)-2*(A=3)- 3*(A<3)-4*(A>7)GOTO 400,900,1000,100
ON X GOTO 100,130,180,220
ON X+3 GOSUB 9000,20,9000
100 ON NUM GOTO 150, 300, 320, 390
500 ON SUM / 2 + 1 GOSUB 50, 80, 20
```

OPEN

TIPO: Istruzione di I/O
FORMATO: OPEN <numero del file>, (<dispositivo>) (, <indirizzo>)
(, «<nome del file> (, <tipo>) (, <modo>)»)

Azione: - Questa istruzione apre un canale di INPUT/OUTPUT per un dispositivo periferico. Tuttavia non e' sempre necessario specificare tutte le parti dell'istruzione OPEN. Alcune istruzioni OPEN richiedono solo due codici:

1) Numero del file logico
2) Numero del dispositivo

Il (nome del file) e' il numero del file logico, che pone in relazione le istruzioni OPEN, CLOSE, CMD, GET#, INPUT#, PRINT# le une con le altre , associandole al nome del file e al dispositivo che deve essere usato. Il numero del file logico deve essere compreso tra 1 e 255; si puo' usare liberamente uno di questi numeri.

NOTA: I numeri di file sopra 128 sono in realta'usati per altri scopi, per cui e' buona regola usare solamente i numeri fino a 127.

Ciascun dispositivo del sistema ha un proprio numero identificazione. Il numero di (dispositivo) e' usato nella OPEN per specificare su

quale dispositivo e' presente il file. Per le periferiche quali cassette, unita' disco o stampanti, si richedono anche i diversi indirizzi secondari. Questi ultimi possono essere immaginati come codici che dicono quale operazione deve eseguire ciascun dispositivo. Il numero del dispositivo del file logico viene usato con ogni GET#, INPUT#,PRINT#. Se il numero del ⟨dispositivo⟩ viene omesso, il computer assume automaticamente le informazioni vengono inviate o ricevute dal Datasette(TM) che porta il numero 1. Anche il nome del file puo' essere omesso,ma nel programma non e' poi possibile richiamare quel file se non gli e' stato assegnato un nome nella OPEN. Quando si registrano file su registratori a cassetta, e si omette l'indirizzo secondario, allora il computer assume 0 come ⟨indirizzo⟩ secondario (operazione di lettura).

I come ⟨indirizzo⟩ secondario apre il file su cassetta per l'operazione di scrittura. 2 come ⟨indirizzo⟩ secondario provoca la scrittura di un segnale di fine nastro alla successiva chiusura del file. L'indicatore di fine nastro protegge da una lettura accidentale oltre la fine dei dati, che provocherebbe il messaggio d'errore del BASIC **?DEVICE NOT PRESENT.**

Per i file su disco, sono disponibili gli indirizzi secondari da 2 a 14 per i file dati, mentre gli altri numeri hanno un significato speciale per i comandi DOS (Per ulteriori dettagli si veda il manuale del drive del disco con i comandi DOS).

Il ⟨nome del file⟩e' una stringa da 1 a 16 caratteri ed e' opzionale per i file su cassetta e su stampante. Se viene omesso il ⟨tipo⟩ di file, si assume per default un file programma, a meno che non sia specificato il ⟨modo⟩.

I file sequenziali vengono aperti per la lettura con ⟨modo⟩=R, a meno che non sia stato specifi ato il ⟨modo⟩=W, implicando cosi' che il file deve essere aperto per la scrittura. Il ⟨tipo⟩ di file puo' essere usato per aprire un file relativo esistente. Usare REL per ⟨tipo⟩ con i file relativi, che, insieme a quelli sequenziali, si trovano solamente su disco.

Se si tenta di accedere ad un file prima di averlo aperto, si verifica il messaggio d'errore **?FILE NOT OPEN** . Se si apre un file su disco per la scrittura e questo file esiste gia', compare il messaggio d'errore **FILE EXISTS** . Non ci sono test di questo tipo per i file su nastro, per cui si deve essere sicuri che il nastro sia posizionato esattamente, altrimenti si puo' scrivere sopra dati precedentemente salvati. Se si apre un file gia' aperto compare il messaggio BASIC: **FILE OPEN** (per ulteriori dettagli vedere il manuale per la stampante).

ESEMPI di istruzione OPEN:

```
10 OPEN 2,8,4"DISK-OUTPUT,SEQ,W" (Apre file sequenziali su disco)

10 OPEN 1,1,2"TAPE-WRITE" (Scrive la fine file alla chiusura)

10 OPEN 50,0 (Input da tastiera)

10 OPEN 12,3 (Output da video)

10 OPEN 1,1,0,"NOME" (Legge da cassetta)

10 OPEN 1,1,1,"NOME" (Scrive su cassetta)

10 OPEN 1,2,0,CHR$(10) (Apre il canale al dispositivo RS-232)
```

```
10 OPEN 1,4,0,"STRINGA" (Predispone la stampa a maiuscolo/grafica)

10 OPEN 1,4,7,"STRINGA" (Predispone la stampa a maiuscolo/minuscolo

10 OPEN 1,5,0,"STRINGA" (Predispone la stampa a maiuscolo/grafica sul
                         dispositivo 5)

10 OPEN 1,8,15,"COMANDO" (Invia un comando al disco)
```

OR

TIPO: Operatore Logico
FORMATO: <operando> OR <operando>

Azione: - Come gli operatori relazionali possono essere usati per prendere delle decisioni riguardanti il flusso del programma, cosi' gli operatori logici possono collegare due o piu' relazioni e dare come risultato il valore vero/falso, che puo' poi essere usato in una decisione. Quando viene usato in un calcolo, l'operatore OR da' come risultato 1 se i corrispondenti bit di uno o di tutti e due gli operandi sono uguali a 1. Il risultato di questo confronto e' un numero intero, il cui valore dipende da quello degli operandi. Quando invece viene usato in un confronto, l'operatore OR ha la capacita' di unire due espressioni in una singola espressione composta. Se una delle due espresioni, o entrambi, risulta vera, il risultato dell'espressione composta e' vero (-1). Nel primo degli esempi che seguono, se AA e' uguale a BB OPPURE XX e' 20 allora l'espressione e' vera
Gli operatori logici funzionano convertendo gli operandi in numeri interi a 16 bit, con segno, rappresentati in complemento a 2 e compresi nell'intervallo -32768...+32767. Se gli operandi non sono compresi in questo intervallo, si genera un messaggio di errore. Ciascun bit del risultato viene determinato a partire dal valore dei corrispondenti bit dei due operandi.

ESEMPI dell'operatore OR:

```
100 IF (AA = BB) OR (XX = 20) THEN . . . . !

230 KK% = 64 OR 32: PRINT KK%
(Quest'istruzione e' stata battuta assumendo un valore di 1000000
 per 64, ed un valore di 100000 per 32)

 96   (Il computer ha risposto con il valore binario 1100000 = 96)
```

PEEK

TIPO: Funzione Intera
FORMATO: PEEK (<espressione numerica>)

Azione: - Ritorna un intero compreso nell'intervallo 0...255, letto da una locazione di memoria. L'<espressione numerica> e' una locazione di memoria che deve essere compresa fra 0 e 65535, altrimenti il BASIC

emette il messaggio di errore **?ILLEGAL QUANTITY**

ESEMPI di funzione PEEK:

```
10 PRINT PEEK(53280) AND 15          (Ritorna il valore del colore di bordo
                                      dello schermo)
5 A%=PEEK(45)+PEEK(46)*256           (Ritorna un indirizzo riportato nella
                                      tabella delle variabili del BASIC)
```

POKE

TIPO: Istruzione
FORMATO: POKE <locazione>, <valore>

Azione: - L'istruzione POKE viene usata per scrivere un valore binario lungo un byte (8 bit) in una locazione di memoria o in un registro di input/output. La (locazione) e' un'espresione aritmetica il cui risultato deve essere compreso fra 0 e 65535; il (valore) e' un'espressione che puo' essere ricondotta ad un numero intero compreso fra 0 e 255. Se un valore supera i limiti di questi due intervalli, compare il messaggio BASIC **?ILLEGAL QUANTITY**

Le istruzioni PEEK e POKE (funzioni interne che considerano una locazione di memoria) sono utili per memorizzare dati, per controllare la grafica o per generare suoni, per caricare sottoprocedure in linguaggio assembly e per passare argomenti e risultati a/da una sottoprocedura in linguaggio assembly. Inoltre, usando l'istruzione PEEK si possono esaminare i parametri del Sistema Operativo, mentre l'uso dell'istruzione POKE consente il loro trattamento e la loro modifica. Una mappa completa delle locazioni utili e' riportata in Appendice G.

ESEMPIO di istruzione POKE:

```
POKE 1024, 1        (Scrive una "A" nella posizione 1 dello schermo)
POKE 2040, PTR      (Aggiorna il puntatore dati dell'animazione 0)
10 POKE RED, 32
20 POKE 36879, 8
2050 POKE A, B
```

POS

TIPO: Funzione Intera
FORMATO: POS (<dummy>)

Azione: - Riporta la posizione attuale del cursore, che e' compresa fra 0 (carattere piu' a sinistra) e 79 per una linea logica di schermo lunga 80 caratteri. Dato che il COMMODORE 64 ha un video con 40 colonne, qualsiasi posizione da 40 a 79 si riferisce alla seconda linea dello schermo. L'argomento dummy viene ignorato.

ESEMPIO di funzione POS:

```
1000 IF POS(0) > 38 THEN PRINT CHR$(13)
```

PRINT

TIPO: Istruzione
FORMATO: PRINT [<variabile>] [<, /;> <variabile>].....

Azione: - L'istruzione PRINT viene comunemente usata per scrivere dati sullo schermo. Tuttavia, si puo' ricorrere all'istruzione CMD per riindirizzare l'output a qualsiasi altro dispositivo del sistema. La <variabile> nella lista di output puo' essere un'espressione di qualunque tipo. Se non viene riportata alcuna lista di output, viene stampata una linea bianca. La posizione di ciascuna voce stampata e' determinata dalla punteggiatura usata per separare le voci riportate sulla lista di output.

I caratteri di punteggiatura utilizzabili sono gli spazi, le virgole o i punti e virgola. La linea logica di schermo di 80 caratteri e' divisa in otto zone di stampa di 10 spazi ciascuna. Se si introduce una virgola nella lista delle espressioni, la stampa del prossimo carattere inizia nella zona di stampa successiva. Il punto e virgola fa si' che il prossimo valore venga stampato immediatamente dopo il valore precedente. Tuttavia, ci sono due eccezioni a questa regola:

1) Gli elementi numerici sono seguiti da uno spazio aggiuntivo.
2) I numeri positivi sono preceduti da uno spazio.

Lo stesso effetto del punto e virgola puo' essere ottenuto usando, come separatori fra le costanti stringa o i nomi delle variabili, uno spazio o nessun tipo di punteggiatura. Va tuttavia notato che gli spazi tra una stringa ed un numero, o tra due numeri, interrompono l'output senza che il secondo elemento venga stampato.

Se alla fine della lista di output si pone una virgola o un punto e virgola, la prossima istruzione PRINT comincia la stampa sulla stessa linea; se invece alla fine della lista non compare alcuna puntaggiatura, dopo l'ultimo dato viene stampato un ritorno carrello ed un incremento di linea, per cui la prossima istruzione PRINT inizia la stampa sulla linea successiva. Se l'output destinato allo schermo e' piu' lungo di 40 colonne, viene continuato sulla linea successiva dello schermo.

Nessuna istruzione BASIC e' piu' varia dell'istruzione PRINT. Associata a questa istruzione, infatti, si trova una tale varieta' di simboli, funzioni e parametri, da far considerare questa istruzione come un linguaggio a se' stante interno al BASIC; un linguaggio particolarmente designato per scrivere sullo schermo.

ESEMPI di istruzione PRINT:

1)
```
 5 X = 5
10 PRINT -5*X, X-5, X+5, X↑5

-25          0          10         3125
```

2)
```
 5 X=9
10 PRINT X;     "AL QUADRATO =";X*X;"E";
```

```
20 PRINT X ; "AL CUBO =" X 3

9 AL QUADRATO = 81 E 9 AL CUBO = 729
```

3)
```
90 AA$="ALPHA":BB$="BAKER": CC$="CHARLIE":DD$="DOG": EE$="ECHO"
100 PRINT AA$BB$;CC$ DD$,EE$

ALPHABAKERCHARLIEDOG      ECHO
```

MODO VIRGOLETTE

Una volta digitato il segno delle virgolette (SHIFT 2), il cursore controlla il termine dell'operazione e comincia a visualizzare i caratteri "reverse" che si stanno digitando. Cio' permette di programmare i controlli del cursore, poiche' una volta che il testo dentro le virgolette viene stampato esse eseguono le loro funzioni. L'unico controllo cursore non interessato dal "modo virgolette" e' INST/DEL

1. MOVIMENTO DEL CURSORE

I controlli cursore programmabili con il "modo virgolette" sono:

TASTO	VISUALIZZAZIONE
CLR/HOME	S
SHIFT CLR/HOME	♥
↑↓ CRSR	0
SHIFT ↑↓ CRSR	○
←→ CRSR	↑
SHIFT ←→ CRSR	←

Se ad esempio si vuole stampare in diagonale la parola HELLO, partendo dall'angolo in alto a sinistra, si deve digitare:

PRINT " CLR/HOME H ↑↓ CRSR E ↑↓ CRSR L ↑↓ CRSR L ↑↓ CRSR O"

che viene visualizzato:

PRINT " S H Q E Q L Q L Q O" "

2. CARATTERI «REVERSE»

Premendo i tasti CTRL e 9 dentro le virgolette appare il carattere R , che fa si' che tutti i caratteri vengano stampati in VIDEO REVERSE (come il negativo di un disegno). Per terminare la stampa "reverse", premere CTRL e Ø , dopodiche' compare il carattere ▭, oppure viene stampato un RETURN [CHR$(13)] (lo stesso risultato si ottiene chiudendo l'istruzione PRINT con una virgola o un punto e virgola).

3. CONTROLLI DEL COLORE

Premendo il tasto CTRL o il tasto C=, insieme ad uno degli otto tasti colore, compare dentro le virgolette un carattere "reverse"

speciale: quando questo carattere viene stampato, si verifica il cambiamento di colore.

TASTO	COLORE	VISUALIZZAZIONE
CTRL 1	Nero	
CTRL 2	Bianco	E
CTRL 3	Rosso	
CTRL 4	Azzurro	
CTRL 5	Porpora	
CTRL 6	Verde	
CTRL 7	Blu	
CTRL 8	Giallo	
C= 1	Arancio	
C= 2	Marrone	
C= 3	Rosso chiaro	
C= 4	Grigio 1	
C= 5	Grigio 2	
C= 6	Verde chiaro	
C= 7	Blu chiaro	
C= 8	Grigio 3	

Se si vuole stampare la parola HELLO in azzurro e la parola THERE in bianco, occorre digitare:

PRINT " CTRL 4 HELLO CTRL 2 THERE"

che viene visualizzato:

PRINT " ▚ HELLO E THERE"

4. MODO INSERIMENTO

Gli spazi creati con il tasto INST/DEL hanno alcune caratteristiche comuni al modo virgolette. I controlli cursore ed i controlli colore appaiono come caratteri "reverse". L'unica differenza sono INST e DEL nel modo virgolette DEL esegue la sua normale funzione, mentre in questo caso visualizza la T Ed INST , che nel modo virgolette crea un carattere speciale, ora inserisce normalmente gli spazi.

Cio' perche' e' possibile creare un'istruzione PRINT contenente DEL , che nel modo virgolette non puo' essere stampato. Il modo di operare per ottenere quanto detto e' illustrato nel seguente esempio:

10 PRINT"HELLO" INST/DEL SHIFT INST/DEL SHIFT INST/DEL INST/DEL INST/DEL P"

che viene visualizzato:

10 PRINT"HELLO T T P"

Quando la precedente linea viene eseguita, la parola visualizzata e' HELP, poiche' le ultime due lettere sono cancellate e sostituite da P.

AVVERTENZA: DEL e' attiva sia con LIST che con PRINT, per cui e' difficile editare una linea con questi caratteri.

La condizione "modo inserimento" termina quando viene premuto il

tasto RETURN (o SHIFT RETURN), oppure quando vengono digitati tanti caratteri quanti sono gli spazi inseriti.

5. ALTRI CARATTERI SPECIALI

Ci sono altri caratteri che possono essere stampati per funzioni speciali, anche se non sono facilmente disponibili da tastiera. Per inserire questi caratteri fra virgolette, si possono lasciare sulla linea spazi a loro riservati, premendo RETURN o SHIFT RETURN , e ritornare agli spazi con il controllo cursore. Per iniziare a digitare i caratteri in "reverse", bisogna premere CTRL RVS/ON , e quindi i seguenti tasti:

FUNZIONE	TASTO
SHIFT RETURN	SHIFT M
Imposta le minuscole	N
Imposta le maiuscole	SHIFT N
Disabilita i tasti di impostazione	H
Abilita i tasti di impostazione	I

SHIFT RETURN funziona sia con PRINT che con LIST, per cui l'uso di questi tasti rende quasi impossibile l'editazione; anche la LISTa di un programma appare in un modo molto strano.

PRINT

TIPO: Istruzione di I/O
FORMATO: PRINT# <numero-file> [<variabile] [<, /;> <variabile].....

Azione: - L'istruzione PRINT# viene usata per scrivere dati su un file logico, per il quale deve essere adoperato lo stesso numero usato nell'istruzione OPEN relativa a quel file. L'output va al numero di dispositivo usato nell'istruzione OPEN. L'espressione ‹variabile› della lista di output puo' essere di qualunque tipo. La punteggiatura tra gli elementi e' la stessa dell'istruzione PRINT e puo' essere usata nella stessa maniera. Gli effetti della punteggiatura sono invece diversi per due importanti motivi.

Quando si usa PRINT# con un file su nastro, la virgola, anziche' spaziare le zone di stampa, ha lo stesso effetto del punto e virgola; percio', se non si usano spazi, virgole, punti e virgola o altri tipi di punteggiatura per separare i dati, l'effetto sulla spaziatura e' lo stesso. I dati vengono scritti come un flusso continuo di caratteri. Gli elementi numerici sono seguiti da uno spazio e, se positivi, sono preceduti da uno spazio.

Se la lista termina senza alcun carattere di punteggiatura, alla fine dei dati viene scritto un ritorno carrello e si avanza di una riga; se invece la lista termina con una virgola o un punto e virgola, il ritorno carrello e l'avanzamento della linea vengono soppressi. Senza badare alla punteggiatura, la successiva istruzione PRINT# inizia a stampare nella posizione del primo carattere disponibile. Quando viene usata l'istruzione INPUT#, l'avanzamento linea agisce come uno stop, lasciando la variabile vuota al momento dell'esecuzione della successiva INPUT#. L'avanzamento linea puo' essere soppresso o compensato come viene mostrato nel'esempio seguente.

Il modo piu' facile per scrivere piu' di una variabile su un file su

nastro o su disco e' quello di impostare la variabile stringa a CHR$(13), ed usare questa stringa come separatore delle variabili, al momento della scrittura del file.

ESEMPI di istruzione PRINT :

1)
```
10 OPEN 1,1,1,,"TAPE FILE"
20 R$ = CHR$(13)
30 PRINT# 1,1;R$;2;R$;3;R$;4;R$;5
40 PRINT# 1,6
50 PRINT# 1,7
```

(Cambiando CHR$(13) in CHR$(44) si usa come separatore una virgola; se invece si usa CHR$(59), si ottiene un punto e virgola).

2)
```
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1, "AAA"CO$"BBB", "CCC";"DDD";"EEE"CR$ "FFF"CR$;
30 INPUT#1, A$,BCDE$,F$
```

```
AAA,BBB       CCCDDDEEE
(ritorno carrello)
PPP (ritorno carrello)
```

3)
```
 5 CR$=CHR$(13)
10 PRINT#2, "AAA";CR$;"BBB"
20 PRINT#2, "CCC";
30 INPUT#2, A$,B$,DUMMY$,C$
```

```
(10 spazi) AAA
BBB
(10 spazi) CCC
```

READ

TIPO: Istruzione
FORMATO: READ <variabile> [,<variabile>].....

Azione: - L'istruzione READ e' usata per riempire i nomi delle variabili con il contenuto delle costanti provenienti dalle istruzioni DATA. Il tipo di dato letto deve essere conforme al tipo di variabile specificata, altrimenti compare il messaggio BASIC **?SYNTAX ERROR** (*). Le variabili della lista di input dell'istruzione DATA devono essere separate da una virgola.
Una singola istruzione READ puo' accedere ad una o piu' istruzioni DATA, l'accesso alle quali avviene in ordine (vd. DATA), oppure diverse istruzioni READ possono accedere alla stessa istruzione DATA. Se vengono eseguite piu' READ di quanti siano gli elementi dell'istruzione DATA, allora compare il messaggio BASIC **?OUT OF DATA** . Se il numero delle variabili specificate e' minore del numero di elementi dell'istruzione DATA, allora la successiva READ inizia a leggere dal successivo elemento della lista (vd. RESTORE).

NOTA: Il messaggio **?SYNTAX ERROR** riporta il numero di linea della istruzione DATA, e NON il quello dell'istruzione READ.

ESEMPI di istruzione READ:

```
110 READ A,B,C$
120 DATA 1,2,HELLO

100 FOR X=1 TO 10: READ A(X):NEXT

200 DATA 3.08, 5.19, 3.12, 3.98, 4.24
210 DATA 5.08, 5.55, 4.00, 3.16, 3.37

1 READ CITY$,STATE$,ZIP
5 DATA DENVER,COLORADO,80211
```

(Riempie la lista degli elementi secondo l'ordine delle costanti (linea 5)).

REM

TIPO: Istruzione
FORMATO: REM [<commento>]

Azione: - L'istruzione REM permette di commentare un programma in modo tale da renderlo piu' comprensibile al momento di LISTarlo. Il commento rappresenta cio' che si vuole fare con un'istruzione o ungruppo di istruzioni in un particolare contesto del programma; ad esempio, puo' servire per ricordare che una variabile viene usata per un determinato scopo. Il commento puo' essere costituito da qualsiasi testo, parola o carattere, compresi anche i due punti (:) o le parole chiave del BASIC.

Tutto cio' che viene scritto dopo REM, e che sta sullo stesso numero di linea, viene ignorato dal BASIC, ma i commenti vengono regolarmente stampati nello stesso modo in cui sono stati immessi. Un'istruzione REM puo' essere riferita ad una GOTO o ad una GOSUB; l'esecuzione del programma continua con la prossima linea eseguibile del programma (tale linea deve avere numero maggiore della precedente, e contenere un'istruzione eseguibile).

ESEMPI di istruzione REM:

```
10 REM CALCOLO DELLA VELOCITA' MEDIA
20 FOR X=1 TO 20 : REM CICLO DI VENTI VALORI
30 SUM=SUM + VEL(X): NEXT
40 AVG=SUM/20
```

RESTORE

TIPO: Istruzione
FORMATO: RESTORE

Azione: - Il BASIC mantiene un puntatore interno ai prossimi dati costanti (DATA) che devono essere letti (READ). In un programma, il puntatore puo' essere impostato daccapo al primo dato costante utilizzando l'istruzione RESTORE. Quest'ultima puo' essere usata ovunque nel programma di rilettura delle istruzioni DATA.

ESEMPIO di istruzione RESTORE:

```
100 FOR X=1 TO 10: READ A(X): NEXT
200 RESTORE
300 FOR Y=1 TO 10: READ B(Y): NEXT

4000 DATA 3.08, 5.19, 3.12, 3.98, 4.24
4100 DATA 5.08, 5.55, 4.00, 3.16, 3.37

(Riempie le due schiere con dati identici)

 10 DATA 1,2,3,4
 20 DATA 5,6,7,8
 30 FOR L=1 TO 8
 40 READ A: PRINT A
 50 NEXT
 60 RESTORE
 70 FOR L=1 TO 8
 80 READ A: PRINT A
 90 NEXT
```

RETURN

TIPO: Istruzione
FORMATO: RETURN

Azione: - L'istruzione RETURN viene usata per uscire da una sottoprocedura chiamata per mezzo dell'istruzione GOSUB. Permette di continuare l'esecuzione delle istruzioni che si trovano dopo la GOSUB. Se si opera con sottoprocedure nidificate, ciascuna GOSUB deve essere accoppiata ad un'istruzione RETURN finale. Una sottoprocedura puo' contenere un numero qualsiasi di istruzioni RETURN, ma la prima di tali istruzioni che viene incontrata fa uscire dalla sottoprocedura.

ESEMPIO di istruzione RETURN:

```
10 PRINT"QUESTO E' IL PROGRAMMA"
20 GOSUB 1000
30 PRINT"IL PROGRAMMA CONTINUA"
40 GOSUB 1000
50 PRINT"ANCORA PROGRAMMA"
60 END
1000 PRINT:"QUESTO E' IL GOSUB":RETURN
```

RIGHT$

TIPO: Funzione Stringa
FORMATO: RIGHT$ (<stringa>, <espressione numerica>)

Azione: - La funzione RIGHT$ ritorna una sottostringa presa dalla parte piu' a destra fino alla fine della (stringa) argomento. La lunghezza della sottostringa e' definita dall'argomento (espressione numerica), che puo' essere un intero compreso fra 0 e 255. Se il valore dell'espressione numerica e' 0, viene ritornata una stringa

nulla (""); se tale valore e' maggiore della lunghezza della <stringa>, viene ritornata l'intera stringa.

ESEMPIO di funzione RIGHT$:

```
10 MSG$ = "COMMODORE COMPUTERS"
20 PRINT RIGHT$(MSG$,9)
RUN

COMPUTERS
```

RND

TIPO: Funzione Reale
FORMATO: RND (<espressione numerica>)

Azione: - La funzione RND crea un numero casuale (random) compreso fra 0.0 e 1.0. Il computer genera una sequenza di numeri casuali eseguendo dei calcoli su un numero di partenza chiamato SEME. Questo viene generato all'accensione del sistema. L'argomento <espressione numerica> e' privo di significato, ad eccezione del segno (positivo, zero o negativo).

Se l'argomento e' negativo, viene ritornata la stessa sequenza di numeri pseudocasuali, a partire dal valore del seme dato. Semi diversi ritornano sequenze numeriche diverse, ma ciascuna sequenza e' ripetitiva, partendo infatti dallo stesso numero di seme. Per eseguire dei test sui programmi, fa comodo avere a disposizione una sequenza di numeri "casuali" conosciuta.

Se come argomento dell'<espressione numerica> si sceglie 0, RND genera un numero direttamente dal clock di sistema. Un argomento negativo implica che per ogni chiamata alla funzione RND viene dato un nuovo seme.

ESEMPI di funzione RND:

```
220 PRINT INT(RND(0)*50)              (Ritorna un numero casuale compreso fra 0
                                       e 49)

100 X=INT(RND(1)*6)+INT(RND(1)*6)+2          (Simula due dadi)
100 X=INT(RND(1)*1000)+1
100 X=INT(RND(1)*150)+100             (Numeri casuali comresi fra 100 e 249)

100 X=RND(1)*(U-L)+L                  (Numeri casuali compresi fra U, estremo
                                           superiore, e L, estremo inferiore)
```

RUN

TIPO: Comando
FORMATO: RUN [<numero di linea>]

Azione: - Il comando di sistema RUN e' usato per lanciare l'elaborazione del programma attualmente in memoria. Il comando RUN causa, prima della partenza del programma, l'esecuzione

dell'istruzione CLR; tuttavia, tale operazione di azzeramento puo' essere evitata usando CONT o GOTO, al posto di RUN, per rilanciare l'elaborazione. Se si specifica il (numero di linea), il programma inizia da quella linea, altrimenti ha inizio dalla prima linea di programma. Il comando RUN puo' essere usato anche all'interno di un programma. Se il (numero di linea) specificato non esiste, compare il messaggio BASIC **UNDEF'D STATEMENT**

Quando si raggiunge una END, una STOP o l'ultima linea di programma, oppure se si verifica un errore durante l'esecuzione, il programma in corso di elaborazione si ferma, restituendo il controllo al BASIC, che si posiziona sul modo diretto.

ESEMPI di comando RUN:

```
RUN      (Inizia dalla prima linea del programma)
RUN 500  (Inizia dalla linea 500)
RUN X    (Inizia dalla linea X oppure causa UNDEF'D STATEMENT ERROR se
          la linea X non esiste)
```

SAVE

TIPO: Comando
FORMATO: SAVE [«<nome del file>»] [,<numero del dispositivo>] [,<indirizzo>]

Azione: - Il comando SAVE viene usato per memorizzare su nastro o disco il programma attualmente in memoria. Durante il funzionamento di SAVE, il programma che deve essere salvato non puo' essere interessato da altri comandi. Il programma salvato rimane ancora in memoria, anche dopo il termine dell'operazione, finche' non venga coperto da un altro programma o non si usi qualche comando. Il tipo di file deve essere "prg" (programma). Se si omette il (numero del dispositivo), allora il COMMODORE 64 assume come dispositivo su cui salvare il programma il dispositivo numero 1, cioe' il nastro. Se il (numero del dispositivo) e' 8, allora il programma viene salvato su disco. L'istruzione SAVE puo' essere usata anche all'interno di un programma: al termine dell'operazione di salvataggio, l'esecuzione riprende dalla linea successiva alla SAVE.

I programmi su nastro sono automaticamente memorizzati due volte, in modo che il COMMODORE 64 possa eseguire dei controlli per scoprire eventuali errori, quando il programma viene nuovamente caricato in memoria. Quando un programma viene salvato su nastro, il (nome del file) e l'(indirizzo) secondario sono facoltativi, anche se il nome del programma tra gli apici ("") o in una variabile stringa (---$) facilita al COMMODORE 64 il compito della ricerca. Se un file viene salvato senza un nome, la successiva ricerca usando un nome non ha esito.

Un indirizzo secondario 1 ordina al KERNAL di caricare il programma su nastro a partire da una locazione di memoria diversa da quella standard, cioe' la locazione 2048. Un indirizzo secondario 2 fa seguire al programma un segnale di fine nastro. Un indirizzo secondario 3 combina entrambe queste funzioni.

Quando si salva un file su disco, il nome del file deve essere sempre presente.

ESEMPI di comando SAVE:

```
SAVE                    (Memorizza su nastro senza assegnare un nome al file)
SAVE "ALPHA", 1         (Memorizza su nastro con il nome "ALPHA")
SAVE "ALPHA", 1, 2      (Memorizza "ALPHA" seguito dall'indicatore di fine
                         nastro)
SAVE "FUN.DISK",8       (Salva su disco [dispositivo 8 = disco])
SAVE A$                 (Memorizza su nastro con il nome contenuto in A$)
10 SAVE "HI"            (Salva il programma epassa alla linea successiva)
SAVE"ME",1,3            (Memorizza nella stessa locazione di memoria, ponendo
                         al termine un indicatore di fine nastro)
```

SGN

TIPO: Funzione Intera
FORMATO: SGN (<espressione numerica>)

Azione: - SGN ritorna un valore intero in base al segno dell'argomento <espressione numerica>: se tale argomento e' positivo il risultato e' 1, se e' negativo il risultato e' -1, se e' nullo il risultato e' 0.

ESEMPIO di funzione SGN:

```
90 ON SGN(DV)+2 GOTO 100, 200, 300
(Salta a 100 se DV=negativo, a 200 se DV=nullo, a 300 se DV=positivo)
```

SIN

TIPO: Funzione Reale
FORMATO: SIN (<espressione numerica>)

Azione: - SIN ritorna il valore del seno, espresso in radianti, dell'<espressione numerica>. Il valore di COS(X) e' uguale a SIN(X+3.14159265/2).

ESEMPIO di funzione SIN:

```
235 AA = SIN(1.5): PRINT AA
 .997494987
```

SPC

TIPO: Funzione Stringa
FORMATO: SPC (<espressione numerica>)

Azione: - La funzione SPC viene usata per controllare la formattazione dei dati sia sullo schermo che in un file logico. Il numero di spazi dati dall'argomento <espressione numerica> viene stampato a partire dalla prima posizione disponibile. Per lo schermo e per il nastro tale valore deve essere compreso tra 0 e 255, per il disco fino a 254. Per

i file su stampante, quest'ultima esegue un ritorno carrello ed un avanzamento linea nel caso che venga stampato uno spazio nell'ultima posizione carattere di una linea. Sulla linea seguente non viene stampato alcuno spazio.

ESEMPIO di funzione SPC:

```
10 PRINT "RIGHT "; "HERE &";
20 PRINT SPC(5) "OVER" SPC(14) "THERE"
RUN

RIGHT HERE &    OVER                THERE
```

SQR

TIPO: Funzione Reale
FORMATO: SQR (<espressione numerica>)

Azione: - SQR ritorna il valore della radice quadrata dell'argomento <espressione numerica>. Questo valore non deve essere negativo, altrimenti compare il messaggio BASIC **?IL-LEGAL QUANTITY** .

ESEMPIO di funzione SQR:

```
FOR J = 2 TO 5: PRINT J*5, SQR(J * 5): NEXT

 10    3.16227766
 15    3.87298335
 20    4.47213595
 25    5
       READY
```

STATUS

TIPO: Funzione Intera
FORMATO: STATUS

Azione: - Ritorna lo stato di completamento dell'ultima operazione di input/output eseguita su un file aperto. Lo stato puo' essere letto da qualsiasi dispositivo periferico. La parola chiave dello stato (ST) e' il nome di una variabile definita dal sistema in cui il KERNAL inserisce lo stato delle operazioni di I/O. La seguente e' la tabella dei valori dei codici di stato per operazioni su file su nastro, stampante, disco ed RS-232.

POSIZIONE DEL BIT ST	VALORE NUMERICO DI ST	VERIFICA NASTRO + CARICO	LETTURA REGISTRATORE	R/W DEL BUS SERIALE
0	1		Supero tempo scrittura	
1	2		Supero tempo lettura	
2	4	Blocco corto		Blocco corto
3	8	Blocco lungo		Blocco lungo
4	16	Errore di lettura non recuperabile		Qualunque errore
5	32	Errore di "checksum"		Errore di "checksum"
6	64	Fine file	EOI	
7	-128	Fine nastro	Dispositivo non presente	Fine nastro

ESEMPI di funzione STATUS:

```
10 OPEN1,4:OPEN2,8,4,"MASTER FILE,SEQ,W"
20 GOSUB 100 : REM CONTROLLA LO STATO
30 INPUT#2,A$,B,C
40 IF STATUS AND 64 THEN 80 : REM TRATTAMENTO DELLA FINE DEL FILE
50 GOSUB 100 : REM CONTROLLA LO STATO
60 PRINT#1,A$,B;C
70 GOTO 20
80 CLOSE1:CLOSE2
90 GOSUB 100:END
100 IF ST > 0 THEN 9000 : REM TRATTAMENTO ERRORE DI I/O DEL FILE
110 RETURN
```

STEP

TIPO: Istruzione
FORMATO: [STEP <espressione>]

Azione: - La parola chiave opzionale STEP segue il (valore finale) che compare nell'istruzione FOR, definendo l'incremento della variabile contatore del ciclo. STEP consente di usare come incremento qualsiasi valore diverso da zero. Se viene omessa, il valore dell'incremento e' +1. Quando in un ciclo FOR si raggiunge l'istruzione NEXT, viene presa in considerazione STEP, quindi viene nuovamente testato il valore finale per controllaare se il ciclo e' terminato (per maggiori informazioni si veda l'istruzione FOR).

NOTA: Una volta inserito in un ciclo il valore di STEP non puo' essere modificato

ESEMPI di istruzione STEP:

```
25 FOR XX = 2 TO 20 STEP 2          (Ripete il ciclo 10 volte)
35 FOR ZZ = 0 TO -20 STEP -2        (Ripete il ciclo 11 volte)
```

STOP

TIPO: Istruzione
FORMATO: STOP

Azione: - L'istruzione STOP viene usata per arrestare l'esecuzione del programma attuale e per ritornare al modo diretto. Digitando da tastiera RUN/STOP si ottiene lo stesso effetto dell'istruzione STOP, per cui il BASIC visualizza il messaggio d'errore **?BREAK IN LINE nnnnn**, seguito da READY (nnnnn = numero di linea dove si e' verificata l'interruzione). Qualsiasi file aperto viene mantenuto tale; tutte le variabili mantengono i rispettivi valori, permettendo cosi' di essere esaminate. Il programma puo' essere fatto ripartire usando le istruzioni CONT o GOTO.

ESEMPI di istruzione STOP:

```
10 INPUT#1, AA, BB, CC
20 IF AA = BB AND BB = CC THEN STOP
30 STOP

BREAK IN LINE 20 (Se AA vale -1 e BB=CC)
BREAK IN LINE 30 (In tutti gli altri casi)
```

STR$

TIPO: Funzione Stringa
FORMATO: STR$ (<espressione numerica>)

Azione: - Ritorna la rappresentazione stringa del valore numerico dato dall'argomento. Al momento della conversione del valore dell'(espressione numerica), tutti i numeri sono seguiti da uno spazio e, se positivi, preceduti da uno spazio.

ESEMPIO di funzione STR$:

```
100 FLT = 1.5E4: ALPHA$ = STR$(FLT)
110 PRINT FLT, ALPHA$

 15000     15000
```

SYS

TIPO: Istruzione
FORMATO: SYS <locazione di memoria>

Azione: - Questo e' il modo piu' comune per fondere programmi BASIC con programmi in linguaggio macchina. Questi ultimi cominciano dalla locazione di memoria indicata dalla SYS. Il comando di sistema SYS viene usato sia in modo diretto che da programma per trasferire il controllo dal microprocessore ad un programma in linguaggio macchina esistente in memoria. La locazione di memoria data e' un'espressione numerica e puo' essere qualunque locazione di memoria ROM o RAM.
Quando si usa l'istruzione SYS, la sezione di codice in linguaggio macchina deve terminare con l'istruzione RTS, in modo che, al termine del programma in linguaggio macchina, l'elaborazione venga ripresa dal BASIC con l'esecuzione dell'istruzione successiva alla SYS.

ESEMPI di istruzione SYS:

```
SYS 64738      (Salta alla ROM riservata alla Partenza a Freddo)
10 POKE 4400,96: SYS 4400          (Salta alla locazione 4400 del codice
                                    macchina, e ritorna immediatamente)
```

TAB

TIPO: Funzione Stringa
FORMATO: TAB (<espressione numerica>)

Azione: - La funzione TAB sposta il cursore di un numero di spazi dato dall'argomento (espressione numerica), a partire dalla posizione piu' a sinistra della linea attuale. Il valore dell'argomento e' compreso fra 0 e 255. L'istruzione TAB puo' essere usata solo con l'istruzione PRINT, dato che non ha alcun effetto se usata con PRINT# per un file logico.

ESEMPIO di funzione TAB:

```
100 PRINT "NOME" TAB(25)、 "QUANTITA'": PRINT
110 INPUT#1, NAM$, AMT$
120 PRINT NAM$ TAB(25) AMT$  AMT$

NAME                              QUANTITA'

G.T. JONES                  25.
```

TAN

TIPO: Funzione Reale
FORMATO: TAN (<espressione numerica>)

Azione: - Ritorna il valore, in radianti, della tangente espressa dall'argomento (espressione numerica). Se per la funzione TAN si verifica un overflow, il BASIC visualizza il messaggio ?DIVISION BY ZERO.

ESEMPIO di funzione TAN:

```
10 XX = .785398163: YY = TAN(XX): PRINT YY
 1
```

TIME

TIPO: Funzione Numerica
FORMATO: TI

Azione: - La funzione TI legge l'intervallo dell'orologio. Questo tipo di "clock" e' chiamato "jiffy clock". All'accensione del sistema il "jiffy clock" viene impostato a zero (inizializzazione). Questo orologio ad intervalli di 1/60 di secondo viene spento durante un I/O su nastro.

ESEMPIO di funzione TI:

```
10 PRINT TI/60   "Secondi dall'accensione"
```

TI$

TIPO: Funzione Stringa
FORMATO: TI$

Azione: - Il timer TI$ funziona come un orologio reale per tutta la durata dell'accensione del sistema. L'intervallo dell'orologio (o "jiffy clock") letto viene usato per aggiornare il valore di TI$, usato successivamente come "stringa di tempo", formata da sei caratteri rappresentanti le ore, i minuti ed i secondi. Il timer TI$ puo' anche assumere un punto arbitrario di partenza, funzionando in questo caso come un cronometro. Dopo un I/O su nastro, il valore di TI$ non e' esatto.

ESEMPIO di funzione TI$:

```
1 TI$ = "000000": FOR J=1 TO 10000: NEXT: PRINT TI$

000011
```

USR

TIPO: Funzione Reale
FORMATO: USR (<espressione numerica>)

Azione: - La funzione USR salta ad una sottoprocedura scritta in linguaggio macchina dall'Utente, il cui indirizzo di partenza e' puntato dal contenuto delle locazioni 785-786. L'indirizzo di partenza viene stabilito, prima di chiamare la funzione USR, usando l'istruzione POKE per impostare le locazioni suddette; in caso contrario, si ha il messaggio di errore ?ILLEGAL QUANTITY.
Il valore dell'<espressione numerica> viene memorizzato nell'accumulatore reale a partire dalla locazione 97, per permettere l'accesso al codice assembler; il risultato della funzione USR e' il valore contenuto in quella locazione quando la sottoprocedura fa ritorno al BASIC.

ESEMPI di istruzione USR:

```
10 B = T * SIN(Y)
20 C = USR (B/2)
30 D = USR (B/3)
```

VAL

TIPO: Funzione Numerica
FORMATO: VAL (<stringa>)

Azione: - Ritorna un valore numerico rappresentante il dato contenuto nell'argomento <stringa>. Se il primo carattere della stringa diverso dallo spazio (" ") non e' il piu' (+), il meno (-) o un numero, il valore ritornato da VAL e' 0. La conversione della stringa termina quando viene incontrata la fine della stringa stessa, oppure al primo carattere diverso da una cifra (ad eccezione del punto decimale e della E esponenziale).

ESEMPIO di funzione VAL:

```
10 INPUT#1, NAM$, ZIP$
20 IF VAL(ZIP$) < 19400 OR VAL(ZIP$) > 96699
   THEN PRINT NAM$ TAB(25) "GREATER PHILADELPHIA"
```

VERIFY

TIPO: Comando
FORMATO: VERIFY [«<nome del file>»], [, <dispositivo>]

Azione: - Il comando VERIFY viene usato in modo diretto per confrontare il contenuto di un programma BASIC, registrato su nastro o su disco, con il programma attualmente in memoria. In genere viene usato subito dopo una SAVE, per essere sicuri della corretta memorizzazione del programma.

Se si omette il numero del ‹dispositivo›, al programma viene assegnato il numero 1, per cui la registrazione avviene sul registratore Datassette [TM]. Per i file su nastro, se si omette il ‹nome del file›, il programma attualmente in memoria viene confrontato con il primo programma trovato sul nastro. Per i file su disco, il ‹nome del file› deve essere presente. Una qualsiasi differenza riscontrata nel testo del programma causa la visualizzazione del messaggio BASIC ?VERIFY ERROR.

Il nome del programma puo' essere passato sia direttamente, all'interno degli apici (""), sia attraverso una variabile stringa. VERIFY e' anche usato per posizionare il nastro esattamente dopo la fine dell'ultimo programma, allo scopo di evitare errori accidentali di sovrascrittura.

ESEMPI di comando VERIFY:

```
VERIFY                          (Controlla il primo programma su nastro)
PRESS PLAY ON TAPE
OK
SEARCHING
FOUND <FILENAME>
VERIFYING

9000 SAVE "ME",8:
9010 VERIFY "ME",8              [Cerca il programma sul disopsitivo 8 (disco)]
```

WAIT

TIPO: Istruzione
FORMATO: WAIT <locazione>, <maschera 1> [, <maschera 2>]

Azione: - L'istruzione WAIT fa si' che l'esecuzione del programma venga sospesa fino al riconoscimento, da parte di un indirizzo di memoria, di una specificata configurazione di bit. In altre parole, WAIT puo' essere usata per arrestare un programma finche' non si verifichino alcuni eventi esterni. Cio' viene svolto controllando lo stato dei bit dei registri di I/O. Il dato usato con WAIT puo' essere

rappresentato da una qualunque espressione numerica, ma deve essere convertito in un valore intero.

Per la maggioranza dei programmatori, questa istruzione non dovrebbe essere mai usata: essa causa infatti l'arresto del programma finche' i bit di una locazione di memoria specificata non cambino in modo specifico; questo fatto puo' tornare utile solamente per determinate operazioni di input/output.

WAIT estrae il valore dalla locazione di memoria ed esegue un'operazione di AND logico con i valori della ⟨maschera 1⟩. Se nell'istruzione viene riportata anche la ⟨maschera 2⟩, allora viene eseguita una OR esclusiva tra il risultato della prima operazione e la ⟨maschera 2⟩. In altri termini, la maschera 1 "esclude" qualsiasi bit non si desideri testare. Per ogni bit uguale a 0 nella maschera 1, il corrispondente bit del risultato e' 0. Il valore della maschera 2 interessa tutti i bit, in modo che il controllo possa avvenire sia per condizione attivata che per condizione disattivata. Tutti i bit testati per zero devono avere un 1 nella corrispondente posizione della maschera 2.

Se i bit corrispondenti degli operandi della ⟨maschera 1⟩ e della ⟨maschera 2⟩ differiscono, la OR esclusiva ha come risultato un bit a 1, altrimenti tale bit e' 0. Tramite l'istruzione WAIT si puo' creare una pausa di lunghezza infinita, nel qual caso il sistema viene ripristinato premendo i tasti RUN/STOP e RESTORE (premere RESTORE mantenendo premuto RUN/STOP). Il primo esempio attende che si prema un tasto del registratore, prima di proseguire l'elaborazione del programma; il secondo esempio attende che un'animazione entri in contatto con lo sfondo dello schermo.

ESEMPI di istruzione WAIT:

```
WAIT 1,32,32
WAIT 53273,6,6
WAIT 36868,144,16    (144 e 16 sono maschere (144=10010000, 16=10000).
                      WAIT ferma il programma finche' il bit 128 non
                      sia in ON, o il bit 16 non sia in OFF)
```

TASTIERA DEL COMMODORE 64 E SUE CARATTERISTICHE

Il Sistema Operativo ha un buffer di tastiera di 10 caratteri, usato per contenere i dati digitati prima che vengano elaborati. Questo buffer, o coda, mantiene i caratteri dei tasti premuti nell'ordine di entrata (il primo tasto ad essere stato premuto genera il carattere elaborato per primo). Se ad esempio viene immesso un secondo carattere prima che il precedente sia stato elaborato, il secondo carattere viene memorizzato nel buffer mentre continua l'elaborazione del primo carattere. Al termine di tale elaborazione, il buffer della tastiera viene controllato per vedere se contiene altri caratteri, dando cosi' inizio all'elaborazione del carattere generato dal secondo tasto premuto. Senza questo buffer, un'immissione rapida di caratteri da tastiera potrebbe provocare perdite accidentali di caratteri.

In altre parole, il buffer della tastiera permette di "digitare in anticipo" sul sistema, abbreviando i tempi di risposta del sistema ai messaggi di INPUT o alle istruzioni GET. Durante la battitura, i valori dei rispettivi caratteri vengono allineati su fila singola (disposti in coda) nel buffer, dove attendono l'elaborazione

nell'ordine di ingresso. Questa caratteristica di "battitura anticipata" genera il problema occasionale delle battiture accidentali di un carattere, per cui il programma puo' prelevare dal buffer un carattere non corretto.

Normalmente, i dati non corretti immessi nel sistema non causano alcun problema, poiche' possono essere corretti con il tasto di spostamento a sinistra del cursore (CRSR), oppure cancellati (tasto INST/DEL e ribattuti correttamente. Se invece e' gia' stato eseguito un ritorno carrello (e' gia' stato premuto il tasto RETURN), non e' piu' possibile alcuna correzione, poiche' tutti i caratteri contenuti nel buffer vengono elaborati prima di qualsiasi correzione. Questa situazione puo' essere evitata usando un ciclo di caricamento del buffer della tastiera prima di leggere una risposta designata:

```
10 GET JUNK$:IF JUNK <> "" THEN 10:REM VUOTA IL BUFFER DELLA TASTIERA
```

Oltre che con GET e INPUT, la tastiera puo' leggere anche con PEEK, andando a prendere da una locazione di memoria (ad esempio, la 197 [$00C5 HEX]) il valore intero del tasto appena battuto. Se al momento dell'esecuzione della PEEK non si e' battutto alcun tasto, viene ritornato il valore 64. I valori numerici ed i simboli della tastiera, insieme ai corrispondenti codici carattere (CHR$), sono riportati nell'Appendice C. Il seguente esempio mostra un ciclo mantenuto attivo fino alla prima battuta di un tasto, dopodiche' esegue la conversione dell'intero al valore di un carattere.

```
10 AA=PEEK(197):IF AA=64 THEN 10
20 BB$=CHR$(AA)
```

La tastiera puo' essere vista come un insieme di interruttori organizzato in una matrice di 8 righe per 8 colonne. Questa matrice viene scandita dal KERNAL alla ricerca degli interruttori chiusi (e quindi dei tasti premuti) per mezzo del circuito di I/O CIA#1 (Adattatore Interfaccia Complessa MOS 6526). Per eseguire questa scansione si utilizzano due registri del CIA: il registro 0, allocato in 56320 ($DC00 HEX), per le colonne, ed il registro 1, allocato in 56321 ($DC01 HEX), per le righe.

I bit 0-7 della locazione di memoria 56320 corrispondono alle colonne 0-7, i bit 0-7 della locazione di memoria 56321 corrispondono alle righe 0-7. Il KERNAL decodifica gli interruttori impostati nei corrispondenti valori CHR$(n) dei tasti premuti, prima scrivendo in sequenza i valori delle colonne, poi leggendo i valori delle righe.

Il prodotto delle otto righe per le otto colonne da' 64 valori possibili; tuttavia, se il primo tasto tasto ad essere battuto e' RVS CTRL o C=, oppure SHIFT seguito da un carattere, allora si generano dei valori addizionali. Cio' perche' il KERNAL decodifica questi tasti separatamente, "ricordandosi" quando uno dei tasti di controllo e' stato premuto. Il risultato della scansione della tastiera e' poi memorizzato nella locazione 197.

I caratteri possono anche essere scritti direttamente nel buffer della tastiera, dalla locazione 631 alla 640, ricorrendo all'istruzione POKE. In questo caso, tali caratteri sono elaborati quando si usa l'istruzione POKE per impostare un contatore carattere nella locazione 198. Questi fatti possono essere usati per far si' che una serie di comandi in modo diretto vengano eseguiti automaticamente stampando le istruzioni sullo schermo, inserendo nel buffer un ritorno

carrello, e quindi impostando il contatore carattere. Nell'esempio che segue, il programma lista se' stesso sulla stampante, riattivando successivamente l'esecuzione:

```
10 PRINT CHR$(147)"PRINT#1:CLOSE1:GOTO 50"
20 POKE 631,19:POKE 632,13:POKE 633,13: POKE 198,3
30 OPEN 1,4:CMD1:LIST
40 END
50 REM IL PROGRAMMA RICOMINCIA DA QUI
```

EDITOR DI SCHERMO

L'Editor di Schermo fornisce una serie di agevolazioni potenti e convenienti per l'editazione del testo di un programma. Una volta listato sullo schermo un segmento di programma, si possono usare il tasto cursore ed altri tasti speciali per muovere il cursore sullo schermo per apportare le modifiche desiderate. Dopo aver terminato, per una linea di testo, tutte le correzioni, battendo il tasto RETURN in qualsiasi posizione della linea si ottiene una lettura da parte dell'Editor di Schermo dell'intera linea logica dello schermo (80 caratteri)

Quindi il testo viene passato all'interprete per essere ridotto in simboli ed essere memorizzato nel programma. La linea editata prende in memoria il posto della vecchia versione di quella linea. Si puo' creare una copia addizionale di qualsiasi linea di testo, semplicemente cambiando il numero di linea e premendo RETURN .

Se si usano abbreviazioni delle parole chiave che comportano per quella linea il superamento degli 80 caratteri, al momento di editazione della linea i caratteri in eccesso vengono persi, poiche' l'Editor puo' leggere solo due linee fisiche di schermo. Cio' spiega anche perche' non e' possibile usare l'istruzione INPUT per piu' di 80 caratteri complessivi. Percio', la lunghezza della linea di testo del BASIC, per tutti gli usi pratici, viene limitata a 80 caratteri, come visualizzato sullo schermo.

In particolari condizioni l'Editor di Schermo tratta i tasti di controllo cursore in maniera diversa da quella normale. Se il cursore e' posizionato alla destra di un numero dispari di doppi apici ("), l'Editor opera in MODO VIRGOLETTE.

Questo modo consente ai dati carattere di essere immessi normalmente, ma il controllo cursore, anziche' muovere il cursore, visualizza caratteri "reverse"; lo stesso accade per i tasti di controllo del colore. Pertanto, all'interno di un dato stringa compreso in un programma si possono includere sia i controlli di cursore che i controlli di colore. Infatti, quando viene stampato sullo schermo il testo compreso fra gli apici, il posizionamento del cursore e le funzioni di controllo del colore vengono eseguite automaticamente, facendo parte della stringa. Il controllo cursore puo' essere usato nel modo seguente:

```
Caratteri digitati      10 PRINT"A(R)(R)B(L)(L)(L)C(R)(R)D"
                        20 REM (R)=DESTRA, (L)=SINISTRA

Stampa                  AC BD
```

Il tasto DEL e' il solo controllo cursore non interessato dal modo

virgolette. Se percio' si commette un errore mentre si e' in modo virgolette, non si puo' usare il tasto ⇐CRSR per salvare il resto della frase e riposizionarsi sul punto in cui si e' verificato l'errore - anche il tasto INST produce sul video un carattere reverse. In questo caso, conviene terminare la linea, e dopo aver premuto il tasto RETURN editare la linea normalmente. Come alternativa, se nella stringa non e' richiesto alcun controllo cursore, si possono premere i tasti RUN/STOP e RESTORE che disattivano il modo virgolette. I tasti di controllo cursore utilizzabili in una stringa sono illustrati nella tabella 2.2

TASTO DI CONTROLLO		VISUALIZZAZIONE
CRSR up	⇑CRSR	○
CRSR down	CRSR⇓	0
CRSR left	⇐CRSR	■
CRSR right	CRSR⇒	J
CLR	CLR/	♥
HOME	/HOME	S
INST	INST/DEL	▌

TABELLA 2.2 – CARATTERI DI CONTROLLO CURSORE NEL MODO VIRGOLETTE

Se NON si e' nel modo virgolette, premendo i tasti SHIFT e INST (INSerT - Inserimento) si verifica uno spostamento dei dati sulla destra del cursore, in modo da aprire uno spazio fra due caratteri per l'inserimento di altri dati. Quindi l'Editor comincia ad operare nel Modo Inserimento, fino al riempimento di tutto lo spazio aperto.

Come nel modo virgolette, anche nel modo inserimento il controllo cursore ed i controlli del colore visualizzano caratteri "reverse". L'unica differenza riguarda il tasto INST/DEL DEL ete INSTer I - Cancellazione/Inserimento): DEL, anziche' operare normalmente come nel modo virgolette, visualizza una T reverse, mentre INST, anziche' visualizzare un carattere reverse, inserisce degli spazi come nel modo normale.

Cio' significa che si puo' creare un'istruzione PRINT contenente caratteri DEL, cosa non possibile nel modo virgolette. Il modo inserimento viene disattivato premendo i tasti RETURN SHIFT e RETURN oppure RUN/STOP e RESTORE ; altrimenti, viene disattivato riempiendo tutti gli spazi inseriti. Un esempio di come usare i caratteri DEL in una stringa e' il seguente:

```
10 PRINT"HELLO"  DEL INST INST DEL DEL P"        (Sequenza di battuta)

10 PRINT"HELP"     (Sequenza listata)
```

Quando viene eseguito questo esempio, viene visualizzata la parola HELP, in quanto le lettere LO sono state cancellate prima di stampare la P. Il carattere DEL all'interno di una stringa funziona sia con LIST che con PRINT: si puo' cosi' "nascondere" una parte o tutta una linea di testo; tuttavia, l'editazione di una linea con questi caratteri e' difficile, se non impossibile.

Per funzioni speciali si possono usare alcuni altri caratteri, anche

se non sono facilmente disponibili da tastiera. Per inserire questi caratteri fra virgolette, occorre lasciare degli spazi vuoti nella linea, premere RETURN ritornare ad editare la linea. Per iniziare a digitare caratteri reverse, premere CTRL e RVS/ON Battere i seguenti tasti:

FUNZIONE DEL TASTO	TASTO PREMUTO	VISUALIZZAZIONE
RETURN shiftato	SHIFT RETURN	
Imposta maiuscolo/minuscolo	SHIFT M	
Imposta maiuscolo/grafica	SHIFT N	

Premere i tasti SHIFT e RETURN comporta un ritorno carrello ed un avanzamento linea sullo schermo, ma non termina la stringa. Cio' funziona sia con LIST che con PRINT, per cui l'editazione e' quasi impossibile se si usa questo carattere. Quando si imposta l'output su stampante attraverso l'istruzione CMD, il carattere N "reverse" abilita l'insieme carattere maiuscolo/minuscolo, e , SHIFT N abilita l'insieme carattere maiuscolo/grafica.

I caratteri "reverse" possono essere inclusi in una stringa premendo i tasti CTRL e RVS, provocando la visualizzazione di una R reverse fra virgolette. Cio' fa si' che tutti i caratteri vengano visualizzati in modo reverse (come il negativo di una fotografia). Per terminare la stampa reverse, premere CTRL e RVS/OFF, visualizzando cosi' una R "reverse". I dati numerici possono essere stampati in modo reverse stampando prima CHR$(18); la disabilitazione avviene, in questo caso, stampando CHR$(146) oppure inviando un ritorno carrello.

CAPITOLO 3

programmazione grafica del commodore 64

- Panoramica sulla Grafica
- Locazioni sulla Grafica
- Modo Carattere Standard
- Caratteri Programmabili
- Grafica nel Modo Multicolore
- Modo Colore di Fondo Esteso
- Grafica Bit Map
- Modo Bit Map Multicolore
- Scrolling Rallentato
- Animazioni
- Altre Caratteristiche della Grafica
- Programmazione delle animazioni – Un Ulteriore Sguardo

PANORAMICA SULLA GRAFICA

Tutte le capacita' grafiche del COMMODORE 64 derivano dal Circuito Interfaccia-Video 6567 (conosciuto anche come circuito VIC-II). Questo circuito fornisce una gran quantita' di modi grafici, compresi un video di testo di 40 colonne per 25 righe, un video ad alta risoluzione di 320 per 200 punti, e le animazioni, piccoli oggetti mobili che semplificano la scrittura dei giochi. E se questo non fosse ancora abbastanza, buona parte dei modi grafici possono essere miscelati sullo stesso schermo. E' possibile, ad esempio, definire la parte alta dello schermo nel modo ad alta risoluzione e quella bassa nel modo testo. E le animazioni si combinano con tutto! Parleremo delle animazioni piu' tardi. Vediamo prima gli altri modi grafici

Il circuito VIC-II possiede i seguenti modi grafici di visualizzazione:

A) MODO CARATTERE VIDEO

1) Modo Carattere Standard
   a) caratteri Rom
   b) caratteri programmabili della Ram

2) Modo Carattere Multicolore
   a) caratteri Rom
   b) caratteri programmabili della Ram

3) Modo Colore di Fondo esteso
   a) caratteri Rom
   b) caratteri programmabili della Ram

B) MODO BIT MAP

1) Modo Bit Map Standard
2) Modo Bit Map Multicolore

C) ANIMAZIONI

1) Animazioni Standard
2) Animazioni Multicolore

LOCAZIONI DELLA GRAFICA

Innanzi tutto qualche informazione di carattere generale. Sullo schermo del COMMODORE 64 ci sono mille possibili locazioni. Normalmente, lo schermo parte dalla locazione 1024 (\$0400 in notazione esadecimale-indicata con HEX) ed arriva alla locazione 2023. Ciascuna di queste locazioni comprende 8 bit. Cio' significa che puo' contenere qualsiasi numero intero compreso fra 0 e 255. Connesso alla Memoria Video c'e' un gruppo di 1000 locazioni chiamate MEMORIA DEL COLORE o RAM DEL COLORE. Queste iniziano alla locazione 55296 (\$D800 HEX) e

terminano a 56295. Ciascuna RAM del colore occupa 4 bit, cioe' puo' contenere numeri interi da 0 a 15. Di conseguenza ci sono 16 possibili colori a disposizione del COMMODORE 64.

Inoltre, in qualunque istante si possono visualizzare 256 caratteri diversi. Per uno schermo video normale, ognuna delle 1000 locazioni della memoria di schermo contiene un numero di codice che informa il circuito VIC-II su quale carattere visualizzare su quella locazione di schermo.

I vari modi grafici sono selezionati dai 47 registri di CONTROLLO del circuito VIC-II. Gran parte delle funzioni della grafica possono essere controllate introducendo (con l'istruzione POKE) il valore appropriato in uno dei registri. Il circuito VIC-II e' locato a partire dalla posizione 53248($D000 HEX) fino alla posizione 53294 ($D02E HEX).

SELEZIONE DEL BANCO VIDEO

Il circuito VIC-II e' in grado di accedere ("vedere") 16K di memoria ad ogni istante. Poiche' ci sono 64 K nella memoria del COMMODORE 64, si puo' desiderare che il circuito VIC-II la scorra per intero. Quello descritto di seguito puo' essere un modo. Ci sono quattro possibili banchi (o segmenti) di 16K di memoria. Tutto cio' che e' necessario e' un mezzo che controlli a quale banco di 16K il circuito VIC-II sta accedendo. I bit di SELEZIONE BANCO, che permettono di accedere a tutti i differenti segmenti di memoria, sono locati nel CIRCUITO #2 ADATTATORE DELL'INTERFACCIA COMPLESSA 6526 (CIA#2). Le istruzioni BASIC PEEK e POKE (o la loro versione in linguaggio macchina) sono usate per scegliere un banco controllando i bit 0 e 1 della PORTA A del CIA#2 [locazione 56576($DD00 HEX)]. Questi due bit devono essere impostati ad OUTPUT per cambiare Banchi. Tutto questo e' illustrato nel seguente esempio:

```
POKE56578,PEEK(56578)OR3:REM ASSICURA CHE I BIT 0 E 1 SIANO IMPOSTATI
                             AD OUTPUT

POKE 56576,(PEEK(56576)AND252)ORA:REM CAMBIA I BANCHI
```

"A" assume uno dei seguenti valori:

VALORE DI A	BIT	BANCO	LOCAZIONE DI PARTENZA	ESTENSIONE DEL CIRCUITO VIC-II
0	00	3	49152	($C000-$FFFF)*
1	01	2	32768	($8000-$BFFF)
2	10	1	16384	($4000-$7FFF)*
3	11	0	0	($0000-$3FFF) (valore di default)

Questo concetto del Banco di 16K e' solo una parte di tutto cio' che fa il circuito VIC-II. Si dovrebbe essere sempre a conoscenza di quale banco il CIRCUITO VIC II sta puntando, in quanto questo influenza la provenienza delle configurazioni dei dati carattere e delle animazioni, l'origine dello schermo, ecc. All'accensione del COMMODORE 64, i bit 0 e 1 della locazione 56576 sono impostati automaticamente sul BANCO 0 ($0000-$3FFF) per ricevere tutte le informazioni del Video.

* NOTA: L'insieme di caratteri del COMMODRE 64 nei BANCHI 1 e 3 non e'

accessibile al circuito VIC-II (ved. il Capitolo della memoria carattere)

MEMORIA DI SCHERMO

La locazione della memoria di schermo puo' essere modificata semplicemente da una POKE rivolta al registro di controllo 53272 ($D018 HEX). Tuttavia, questo registro e' usato anche per controllare quale insieme di caratteri si e' usato, per cui occorre fare attenzione a non interferire in quella parte del registro di controllo. La locazione della memoria di schermo e' controllata dai bit MAGGIORI di 4. Per rimuovere lo schermo, si deve usare la seguente istruzione:

```
POKE 53272,(PEEK(53272)AND15)ORA
```

dove "A" assume uno dei seguenti valori:

A	BIT	LOCAZIONE (*)	
		DECIMALE	HEX
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (DEFAULT)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

* Ricordarsi di sommare l'INDIRIZZO DEL BANCO del Circuito VIC-II

MEMORIA COLORE

La memoria del colore NON puo' essere rimossa. E' sempre locata da 52296($3800 HEX) a 56295($0BE7). La memoria di schermo (1000 locazioni a partire da 1024) e la memoria del colore sono usate in maniera differente nei diversi modi della grafica. Un disegno creato in un modo grafico sembra spesso differente quando viene visualizzato in un altro modo grafico.

MEMORIA CARATTERE

Per la programmazione della grafica e' importante conoscere il punto esatto da cui il VIC-II ottiene le informazioni carattere. Normalmente, il circuito ottiene i caratteri che si voglioni visualizzare dalla ROM GENERATORE DI CARATTERE. In questo circuito sono memorizzate le configurazioni che compongono le varie lettere, numeri, simboli della punteggiatura e tutto quello che e' visibile

sulla tastiera. Una delle caratteristiche del COMMODIORE 64 e' la capacita' di usare le configurazioni locate nella memoria RAM. Queste configurazioni RAM possono essere create dall'utente, per cui si ha un insieme quasi infinito di simboli per giochi, applicazioni commerciali, ecc.

Un normale insieme di caratteri e' composto da 256 caratteri, dove ogni carattere e' definito da 8 bytes di dati. Cio' significa che, occupando ogni carattere 8 bytes, un intero insieme di caratteri occupa 256 X 8 bytes = 2K bytes di memoria. Poiche' il circuito VIC-II osserva ad un istante 16K di memoria, per un insieme di caratteri completo ci sono 8 possibili locazioni. Naturalmente, non si e' obbligati ad usare un intero insieme di caratteri. In ogni caso, questo deve iniziare ad una delle 8 possibili locazioni di partenza.

La locazione della memoria carattere e' controllata da 3 bit del registro di controllo del VIC-II, locati alla posizione 53272($D018 HEX). I bit 3, 2 e 1 controllano in quale blocco di 2K e' locato l'insieme dei caratteri. Il bit 0 e' ignorato. Va ricordato che questo registro e' lo stesso che determina la locazione della memoria di schermo, per cui e' consigliabile evitare di interferire nei bit della memoria di schermo. Per cambiare la locazione della memoria carattere, si puo' usare la seguente istruzione BASIC:

```
POKE53272,(PEEK(53272)AND240)ORA
```

Dove "A" assume uno dei seguenti valori:

VALORE DI A	BIT	LOCAZIONE DELLA MEMORIA CARATTERE (*)	
		DECIMALE	HEX
0	XXXX000X	0	$0000-$07FF
2	XXXX001X	2048	$0800-$0FFF
4	XXXX010X	4096	$1000-$17FF IMMAGINE ROM NEL BANCO 0 & 2 (default)
6	XXXX011X	6144	$1800-$1FFF IMMAGINE ROM NEL BANCO 0 & 2
8	XXXX100X	8192	$2000-$27FF
10	XXXX101X	10240	$2800-$2FFF
12	XXXX110X	12288	$3000-$37FF
14	XXXX111X	14336	$3800-$3FFF

* Ricordarsi di aggiungere l'indirizzo del banco

L'IMMAGINE ROM della tabella precedente si riferisce alla ROM generatore di carattere. Questa compare in sostituzione della RAM nelle precedenti locazioni del banco 0, come pure nella RAM nel banco 2, dalla locazione 36864 ($9000 HEX) alla 40959 ($9FFF HEX). Poiche' il circuito VIC-II, ad un determinato istante, puo' accedere solamente a 16K di memoria, le configurazioni carattere ROM compaiono nel blocco di memoria di 16K osservato dal circuito VIC-II. Percio' il sistema e' stato progettato in modo che il circuito VIC II consideri i caratteri ROM nelle locazioni da 4096 a 8191 ($1000-$1FFF) HEX) per il banco 0. e da 36864 a 40959 ($9000-$9FFF) per il banco 2, anche se la ROM carattere e' attualmente nelle locazioni da 53243 a 57343 ($0000-$DFFF HEX).

Questa rappresentazione e' applicabile solamente al mod di vedere i dati carattere del circuito VIC II. Puo' essere usata per programmi,

altri dati, ecc., in maniera del tutto analoga ad ogni altra memoria RAM.

NOTA: Se tali rappresentazioni ROM fanno parte della grafica impostare ancora i BIT DI SELEZIONE BANCO ad uno dei BANCHI sprovvisti di rappresentazioni ROM (BANCHI 1 O 3).

La locazione ed i contenuti dell'insieme carattere nella ROM sono i seguenti:

BLOCCO	INDIRIZZO		IMMAGINE DEL VIC-II	CONTENUTO
	DECIMALE	HEX		
0	53248	D000-D1FF	1000-11FF	Caratteri Maiuscoli
	53760	D200-D3FF	1200-13FF	Caratteri Grafici
	54272	D400-D5FF	1400-15FF	Caratteri Maiuscoli "reverse"
	54784	D600-D7FF	1600-17FF	Caratteri Grafici "reverse"
1	55296	D800-D9FF	1800-19FF	Caratteri Minuscoli
	55808	DA00-DBFF	1A00-1BFF	Caratteri Maiuscoli e grafici
	56320	DC00-DDFF	1C00-1DFF	Caratteri Minuscoli "reverse"
	56832	DE00-DFFF	1E00-1FFF	Caratteri Maiuscoli e Grafici "reverse"

I lettori piu' attenti avranno gia' osservato qualcosa. Le locazioni occupate dalla ROM carattere sono le stesse di quelle occupate dai registri di controllo del circuito VIC-II. Cio' e' possibile in quanto le stesse locazioni non sono occupate nello stesso tempo. Quando il circuito VIC-II ha bisogno di accedere ai dati carattere, la ROM viene posizionata di conseguenza. Essa diviene un'immagine nel banco di memoria di 16K puntato dal circuito VIC-II. Altrimenti, quest'area viene occupata dai registri di controllo dell'I/O, e la ROM carattere rimane a disposizione del circuito VIC-II.

Tuttavia, si puo' aver bisogno di accedere alla ROM carattere nel caso in cui si vogliano usare i caratteri programmabili, e si desideri copiare una parte della ROM carattere per delle definizioni carattere personalizzate. In questo caso si deve escludere il registro di I/O, posizionare la ROM carattere ed eseguire la copia. Al termine, occorre posizionare di nuovo i registri di I/O. Durante la ricopiatura (quando l'I/O e' escluso), non deve avvenire alcuna interruzione, in quanto ai registri di I/O e' demandata la funzione di gestire tale interruzione. Nel caso venga eseguita accidentalmente un'interruzione, la macchina reagira' in maniera anomala. Per escludere la tastiera e le altre normali interruzioni che si verificano sul COMMODORE 64, usare la seguente POKE:

```
POKE56334,PEEK(56334)AND254 (ESCLUDE LE INTERRUZIONI).
```

Al termine del prelievo dei caratteri dalla ROM carattere, ed al momento di proseguire con il programma, si deve riattivare l'esame della tastiera con la seguente POKE:

```
POKE56334,PEEK(56334)OR1 (ATTIVA LE INTERRUZIONI)
```

La seguente POKE esclude i I/O e posiziona la ROM CARATTERE:

```
POKE1,PEEK(1)AND251
```

Dopo l'esecuzione di questa istruzione, la ROM carattere si trova nelle locazioni da 53248 a 57343 ($D000-$DFFF).
Per riposizionare l'I/O in $0000 per l'uso in una normale operazione, usare la seguente POKE:

```
POKE1,PEEK(1)OR4
```

MODO CARATTERE STANDARD

Il modo carattere standard e' il modo in cui si trova il COMMODORE 64 al momento dell'accensione per la prima volta. E' il modo in cui di solito si programma.
I caratteri possono essere prelevati dalla ROM o dalla RAM, ma di solito sono prelevati dalla ROM. Quando per un programma si desiderano particolari caratteri grafici, tutto cio' che e' necessario fare e' definire nella RAM le nuove forme carattere, e comunicare al circuito VIC-II di prelevare le informazioni carattere da li' anziche' dalla ROM carattere. Questo argomento e' trattato piu' dettagliatamente nel prossimo paragrafo.
Per visualizzare sullo schermo i caratteri a colori, il circuito VIC-II accede alla memoria schermo allo scopo di determinare il codice carattere per quella locazione sullo schermo. Nello stesso tempo, accede alla memoria del colore per determinare quale colore si desidera per il carattere visualizzato. Il codice carattere viene tradotto dal VIC-II nell'indirizzo di partenza del blocco di 8 byte contenente la configurazione del carattere. Questo blocco di 8 byte e' locato nella memoria carattere. La traduzione non e' molto complicata, anche se per generare l'indirizzo desiderato vengono combinate diverse voci. Innanzitutto, il codice carattere usato per modificare (POKE) la memoria schermo viene moltiplicato per 8. Al risultato viene aggiunto l'indirizzo di partenza della memoria carattere (vd. il paragrafo MEMORIA CARATTERE). Successivamente vengono presi in considerazione i bit di SELEZIONE BANCO, che vengono sommati all'indirizzo base. (vd. il paragrafo SELEZIONE DEL BANCO VIDEO). La formula seguente illustra quanto sopra esposto:

```
INDIRIZZO CARATTERE=CODICE SCHERMO * 8 + (INSIEME CARATTERE*2048) +
                   (BANCO * 16384)
```

DEFINIZIONE DEL CARATTERE

Ogni carattere e' formato da una griglia (matrice) di 8 punti X 8, ogni punto della quale puo' essere acceso (ON) o spento (OFF). Le immagini carattere del COMMODORE 64 sono registrate nel circuito ROM GENERATORE DI CARATTERI. I caratteri sono registrati come insiemi di 8 byte per ogni carattere, ogni bit rappresentando un punto. Un bit a 0 indica che il punto e' spento, e un bit a 1 indica che il punto e' acceso. Nella Rom, la memoria carattere inizia alla locazione 53248 (quanto l' I/O e' escluso). I primi 8 byte, dalla locazione 53248 ($D000 HEX) alla 53255 ($D007 HEX) contengono la configurazione del simbolo @, il cui codice carattere ha valore 0 nella memoria di schermo. Gli 8 bytes successivi, dalla locazione 53256 ($D008) alla 53263 ($D00F), contengono le informazione necessarie alla costruzione della lettera "A".

IMMAGINE	BINARIO	PEEK
**	00011000	24
****	00111100	60
** **	01100110	102
******	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Ogni insieme carattere completo occupa 2k di memoria (2048 bit), pari a 256 caratteri di 8 byte ciascuno. Poiche' ci sono due insiemi carattere, uno per le lettere maiuscole e la grafica, l'altro per le lettere maiuscole e minuscole, la ROM generatore di caratteri occupa in totale 4K locazioni.

CARATTERI PROGRAMMABILI

Poiche' i caratteri sono memorizzati nella ROM, sembrerebbe a prima vista non esserci modo di cambiarli a favore di caratteri definiti dall'Utente. Tuttavia, la locazione di memoria che indica al circuito VIC-II dove trovare i caratteri e' un registro programmabile che puo' essere modificato per puntare a numerosi segmenti della memoria. Modificando il puntatore alla memoria carattere in modo che esso punti alla RAM, l'insieme carattere puo' essere programmato per ogni necessita'.

Se si vuole locare in RAM un'insieme carattere, ci sono pochi punti MOLTO IMPORTANTI da prendere in considerazione all'atto della programmazione di un insieme carattere personalizzato. Inoltre, ci sono altri due punti importanti di cui si deve essere a conoscenza per creare caratteri speciali personalizzati:

1) E' un operazione "tutto o niente". Di solito, se si usa un insieme carattere personalizzato per comunicare al circuito VIC-II che l'informazione carattere e' disponibile nell'apposita area RAM, i 64 caratteri standard Commodore non sono accessibili. Questo e' risolvibile copiando nell'apposita memoria carattere RAM tutte le lettere, i numeri o la grafica standard del COMMODORE 64 che si intende usare. Si possono scegliere e prendere solo i caratteri che si desiderano, e senza neanche metterli in ordine!

2) L'iniseme carattere toglie spazio di memoria al programma BASIC. Naturalmente, con 38K disponibili per un programma BASIC, quasi tutte le applicazioni non danno problemi.

ATTENZIONE: Occorre fare attenzione nel proteggere l'insieme carattere dalla sovrapposizione del programma BASIC, visto che anch'esso usa la RAM.

Nel COMMODORE 64 ci sono due locazioni da cui far partire l'insieme carattere personalizzato che NON DEVONO ESSERE USATE CON IL BASIC: locazione 0 e locazione 2048. La prima non deve essere usata in quanto il sistema memorizza dati importanti sulla pagina 0 la seconda non puo' essere usata poiche' da tale locazione ha origine il programma

BASIC! Ci sono pero' sei altre posizioni per l'insieme carattere personalizzato. Il punto di partenza migliore in cui porre l'insieme carattere personalizzato per l'uso con il BASIC durante le prove e' la locazione 12288($3000 HEX). Questo viene fatto modificando (tramite l'istruzione POKE) i quattro bit piu' bassi della locazione 53272 con 12. Un esempio puo' essere:

```
POKE53272,(PEEK(53272)AND240)+12
```

Subito, tutte le lettere sullo schermo si trasformano in caratteri indecifrabili in quanto finora non c'e' alcun insieme di caratteri alla locazione 12288... solo bytes in ordine casuale. Impostare da capo il COMMODORE 64 al normale premenddo il tasto RUN/STOP e quindi RESTORE

Iniziamo ora la creazione della grafica. Per proteggere l'insieme carattere dal BASIC e' opportuno ridurre la quantita' di memoria che il BASIC ritiene di avere a disposizione. La quantita' di memoria del Computer rimane la stessa.....Soltanto che al BASIC viene impedito l'utilizzo di una parte di essa. Ad esempio:

```
PRINT FRE(0)-(SGN(FRE(0))(0)*65535
```

Il numero visualizzato e' la quantita' di memoria inutilizzata. Battere ora:

```
POKE52,48:POKE56,48:CLR
```

e quindi:

```
PRINT FRE(0)-(SGN(FRE(0))(0)*65535
```

Che cosa e' cambiato? Ora il BASIC ha a disposizione meno memoria. La memoria appena tolta al BASIC e' quella dove va a risiede l'insieme carattere, al riparo dalle azioni del BASIC. Il passo successivo e' introdurre i caratteri nella Ram. All'inizio, partendo dalla locazione 12288 ($3000 HEX) si devono introdurre nella RAM le configurazioni carattere (analoghe a quelle residenti in ROM) a disposizione del circuito VIC-II. Il programma seguente trasporta 64 carattteri dalla ROM alla RAM riservata all'insieme carattere.

```
5 PRINTCHR$(142) : REM IMPOSTA LE MAIUSCOLE
10 POKE52,48:POKE56,48:CLR : REM RISERVA MEMORIA PER CARATTERI
20 POKE56334,PEEK(56334 )AND254 : REM DISATTIVA IL TIMER DI
21 REM INTERRUZIONE DELLA TASTIERA.
30 POKE1,PEEK(1)AND251 : REM ATTIVA UN CARATTERE
40 FORI=0TO511:POKEI+12288,PEEK(I+53248 ):NEXT
50 POKE(1),PEEK(1)OR4 : REM ATTIVA L'I/O
60 POKE56334,PEEK(56334 )OR1 : REM RIATTIVA IL TIMER DI
61 REM INTERRUZIONE DELLA TASTIERA
70 END
```

Modifichiamo ora la locazione 53272 con (PEEK(53272)AND240)+12. Che cosa succede? Beh, quasi nulla. Ora il COMMODORE 64 sta traendo l'informazione carattere dalla RAM anziche' dalla ROM. Ma finche' i

caratteri non sono stati copiati esattamente dalla ROM, non si puo' osservare alcuna differenza.

A questo punto si possono finalmente modificare i caratteri. Azzerare lo schermo e battere @. Abbassare il cursore di alcune linee, e battere:

```
FORI=12288TO12288+7:POKE1,255-PEEK(I).NEXT
```

Si e' appena creato una @ in campo "reverse"!

INFORMAZIONE: I caratteri in campo inverso sono solamente caratteri le cui configurazioni di bit nella memoria carattere sono state invertite.

Spostare ora il cursore di nuovo sul programmma e premere di nuovo RETURN per invertire un'altra volta il carattere (riportandolo a normale). Osservando la tavola dei codici dello schermo video, ci si puo' rendere conto di quale posizione occupi ogni carattere nella RAM. Da notare solamente che ogni carattere sottrae alla memoria 8 locazioni. Qui di seguito sono dati alcuni esempi:

CARATTERI	CODICE VIDEO	LOCAZIONE D'INIZIO DELLA RAM
@	0	12288
A	1	12296
?	33	12552
>	62	12784

Va ricordato che abbiamo usato solo i primi 64 caratteri. Se si desiderano altri caratteri, occorre fare qualcos'altro. Ad esempio, cosa andrebbe fatto se si desiderasse il carattere 154, una Z "reverse"?

Si potrebbe semplicamente battere una Z "reverse", oppure copiare dalla ROM l'insieme dei caratteri in campo inverso, o anche prelevare dalla ROM il carattere desiderato e sostituirlo nella RAM ad uno dei caratteri che non vengono usati.

Supponiamo di non usare il segno ">", e sostituiamolo con Z "reverse". Digitare:

```
FORI=0TO7:POKE12784+I,255-PEEK(I+12496):NEXT
```

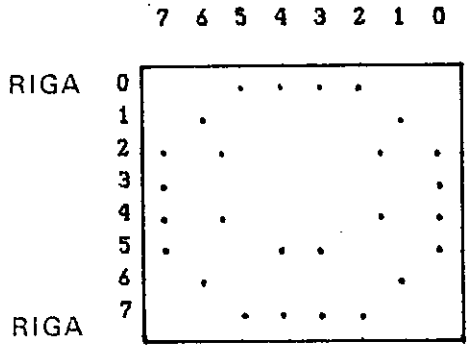
Digitare ora il segno >: comparira' una Z "reverse". Tutte le volte che verra' battuto >, comparira' una Z "reverse" (in realta' questo scambio non avviene. Anche se il segno > fa comparire una Z in campo inverso, in un programma agisce ancora come >. Trovando qualcosa che richieda il segno >, si osservera' che funziona ancora bene, solo che sembrera' strano).

RAPIDO RIASSUNTO: a questo punto si e' in grado di copiare caratteri dalla ROM alla RAM, perfino di sceglere quelli che vogliamo. Rimane soltanto un passo da compiere nei caratteri programmabili (cioe' il migliore!).....costruire caratteri personalizzati. Ogni carattere e' memorizzato nella ROM come un gruppo di 8 byte. Le configurazioni di bit dei byte controllano direttamente il carattere. Se si incolonnano 8 byte, e si scrive ogni byte come una seguenza di 8 numeri binari, si costruisce una matrice 8 X 8, che somiglia ai caratteri. Quando un bit

e' a 1, in quella locazione c'e' un punto, quando un bit e' a 0 in quella locazione c'e' uno spazio. Quando si creano caratteri personalizzati, in memoria viene impostato lo stesso tipo di tabella. Battere NEW e poi il seguente programma:

```
10FORI=12448TO12455:READA:POKEI,A:NEXT
20DATA60,66,165,129,165,153,66,60
```

Ora battere RUN. Il programma sostituisce la lettera T con una faccia sorridente. Ogni numero dell'istruzione DATA della linea 20 corrisponde ad una riga dalla faccia sorridente La matrice della faccia e' simile a questa:



BINARIO	DECIMALE
00111100	60
01000010	66
10100101	165
10000001	129
10100101	165
10011001	153
01000001	66
00111100	60

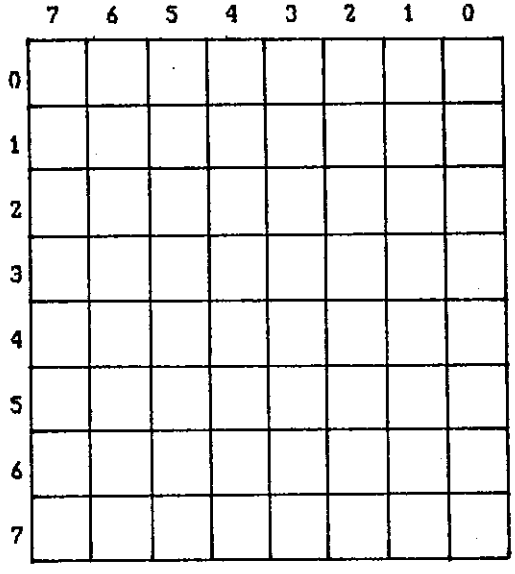


FIGURA 3.1. Tabella per la costruzione di caratteri programmabili

La tabella per la costruzione di caratteri programmabili (figura 3.1) puo' aiutare a disegnare i caratteri. Sul foglio c'e' una matrice 8 X 8, con i numeri di riga e di colonna (se si considera ogni riga come una parola binaria, i numeri sono il valore di quella posizione del bit. Ogni numero e' una potenza di due. Il bit piu' a sinistra e' uguale a 128 (=2 elevato alla potenza 7), il successivo e' uguale a 64 (=2 elevato alla potenza 6) e cosi' via, finche' non si raggiunge il bit piu' a destra che e' uguale ad 1 (=2 elevato alla potenza 0).

Tracciare sulla matrice una X in ogni locazione dove si vuole un punto nel carattere. Quando il carattere e' pronto si puo' creare l'istruzione DATA per quel carattere.

A partire dalla prima riga, scrivere il numero in cima ad ogni

colonna dove sia stata piazzata una X (tale numero, come spiegato sopra, e' una potenza di 2). Completata la prima riga, sommare tali numeri e scrivere il risultato vicino alla riga. Questo e' il numero da inserire nell'istruzione DATA per disegnare tale riga.

Operando allo stesso modo con le altre 7 righe, si ottengono 8 numeri compresi fra 0 e 255. Per essere corretti, tali numeri devono essere compresi in questo intervallo! Se si hanno meno di 8 numeri si e' saltata una riga. Se alcuni sono 0 va bene. La riga 0 ha la stessa importanza delle altre. Sostituire i numeri dell'istruzione DATA alla linea 20 con quelli appena calcolati, e lanciare (RUN) il programma. Poi battere una T. Ogni volta che questo tasto viene battuto, compare il carattere personalizzato.

Se il carattere risultante non e' riuscito bene, basta cambiare i numeri nell'istruzione DATA e rilanciare (RUN) il programma finche' non si e' soddisfatti. Questo e' tutto!

SUGGERIMENTO: per ottenere migliori risultati si consiglia di fare le linee verticali ampie almeno due punti (bit). Cio' previene il disturbo CHROMA (distorsione del colore) quando i caratteri vengono visualizzati su uno schermo TV.

Il seguente esempio illustra un programma che usa i caratteri programmabili standard:

```
10 REM * ESEMPIO 1 *
20 CREAZIONE DI CARATTERI PROGRAMMABILI
30 REM DISATTIVA KB E I/O
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM INTERVALLO CARATTERI DA COPIARE DALLA ROM
36 FORJ=0TO7:REM COPIA TUTTI GLI 8 BYTE DI UN CARATTERE
37 POKE12288+I*8+J,PEEK(12288+I*8+J):REM COPIA UN BYTE
38 NEXTJ:NEXTI:REM PASSA AL PROSSIMO BYTE O CARATTERE
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM ATTIVA I/O E KB
40 POKE53272,(PEEK(53272)AND240)+12:REM IMPOSTA IL PUNTATORE
41 REM CARATTERE ALLA LOCAZIONE DI MEMORIA 12288
60 FORCHAR=60TO63:REM PROGRAMMA I CARATTERI DA 60 A 63
80 FORBYTE=0TO7:REM COSTRUISCE TUTTI GLI 8 BYTE DI UN CARATTERE
100 READ NUMBER:REM LEGGE 1/8 DEI DATI DI UN CARATTERE
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM MEMORIZZA I DATI
140 NEXTBYTE:NEXTCHAR:REM PASSA AL PROSSIMO BYTE O CARATTERE
150 PRINTCHR$(147)TAB(255)CHR$(60);
155 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 150 VISUALIZZA I CARATTERI APPENA DEFINITI
170 GETA$:REM ATTENDE CHE L'UTENTE PREMA UN TASTO
180 IFA$=""THENGOTO170:REM SE NON SI PREMONO TASTI, ATTENDE DI NUOVO!
190 POKE53272,21:REM RITORNA AI CARATTERI NORMALI
200 DATA4,6,7,5,7,7,3,3:REM DATI DEL CARATTERE 60
210 DATA32,96,224,160,224,224,192,192:REM DATI DEL CARATTERE 61
220 DATA7,7,7,31,31,95,143,127:REM DATI DEL CARATTERE 62
230 DATA224,224,224,248,248,248,240,224:REM DATI DEL CARATTERE 63
240 END
```

GRAFICA DEL MODO MULTICOLORE

La grafica standard ad alta risoluzione da' il controllo su punti molto piccoli dello schermo. Ogni punto della memoria carattere puo' assumere due valori, 1 per acceso e 0 per spento. Quando un punto e' spento, il colore dello schermo viene usato sullo spazio riservato a quel punto. Se il punto e' acceso, viene colorato con il colore del carattere che si e' scelto per quella posizione dello schermo. Quando si usa la grafica standard ad alta risoluzione,tutti i punti interni ad un carattere 8 X 8 possono avere sia il colore di fondo che il colore principale. Questo limita in qualche modo la risoluzione del colore all'interno dello spazio. Ad esempio, possono insorgere dei problemi quando s'incrociano due linee di differenti colori.

Il modo multicolore risolve questo problema. Ogni punto nel MODO MULTICOLORE puo' essere di quattro colori: colore di schermo (registro #0 del colore di fondo), colore contenuto nel registro #1 di fondo, colore contenuto nel registro #2 di fondo, o colore carattere. L'unico sacrificio e' a carico della risoluzione orizzontale, in quanto ogni punto del modo multicolore e' largo il doppio di un punto ad alta risoluzione. Questa minima perdita di risoluzione e' largamente compensata dalle alte capacita' del modo multicolore.

BIT DEL MODO MULTICOLORE

Per attivare il Modo Carattere Multicolore, impostare a 1 il BIT 4 del registro di controllo del VIC-II situato nella locazione 53270 (\$D016) usando la seguente POKE:

```
POKE 53270,PEEK(53270)OR16
```

Per disattivare il modo carattere multicolore, impostare a 0 il bit 4 della locazione 53270 per mezzo della seguente POKE:

```
POKE53270,PEEK(53270)AND239
```

Il modo multicolore e' impostato a ON o OFF oer ogni spazio dello schermo, in modo che la grafica multicolore possa essere usata insieme alla grafica ad alta risoluzione (hi-res). Cio' viene controllato dal BIT 3 della memoria del colore, che inizia alla locazione 55296 (\$D800 HEX). Se il numero della memoria del colore e' minore di 8 (0-7), lo spazio corrispondente sullo schermo video viene considerato standard ad alta risoluzione, nel colore (0-7) scelto. Se il numero locato nella memoria del colore e' maggiore o uguale a 8 (8-15), allora lo spazio viene visualizzato nel modo multicolore. Introducendo con una POKE un numero nella memoria del colore, si puo' cambiare il colore del carattere in una data posizione dello schermo. Un numero compreso fra 0 e 7 da' colori carattere normale. Un numero compreso fra 8 e 15 converte lo spazio nel modo multicolore. In altri termini, mettendo a ON il bit 3 della memoria colore, si imposta il MODO MULTICOLORE, mentre mettendo lo stesso bit a OFF viene impostato il modo normale ad ALTA RISOLUZIONE. Una volta che uno spazio e' impostato al modo multicolore, i bit del carattere determinano con quali colori devono essere visualizzati i punti. L'esempio seguente illustra la

costruzione della lettera A e la sua configurazione di bit:

IMMAGINE	CONFIGURAZIONE BIT
`  * *  `	00011000
` ***** `	00111100
`**   **`	01100110
`*******`	01111110
`**   **`	01100110
`**   **`	01100110
`**   **`	01100110
	00000000

Nel modo normale o ad alta risoluzione,il colore dello schermo e' visualizzato per tutti i bit impostati a 0, mentre il colore carattere viene visualizzato per tutti i bit impostati a 1. Il modo multicolore usa coppie di bit, come nell'esempio seguente:

IMMAGINE	CONFIGURAZIONE BIT
AABB	00 01 10 00
CCCC	00 11 11 00
AABBAABB	01 10 01 10
AACCCCBB	01 11 11 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
AABBAABB	01 10 01 10
	00 00 00 00

Nella precedente area immagine, gli spazi indicati con AA sono tracciati con il colore di fondo #1, quelli indicati con BB usano il colore di fondo #2, e quelli indicati con CC usano il colore carattere. Tutto cio' viene determinato dalle coppie di bit, secondo quanto illustrato nella seguente tabella:

COPPIA DI BIT	REGISTRO COLORE	LOCAZIONE
00	Colore #0 di fondo (colore schermo)	53281 ($D021)
01	Colore #1 di fondo	53282 ($D022)
10	Colore #2 di fondo	53283 ($D023)
11	Colore specificato dai 3 bit bassi della memoria del colore	RAM colore

Battere NEW e poi il seguente programma dimostrativo:

```
100 POKE53281,1:REM IMPOSTA A BIANCO ILCOLORE #0 DI FONDO
110 POKE53282,3:REM IMPOSTA AD AZZURRO IL COLORE #1 DI FONDO
120 POKE53283,8:REM IMPOSTA AD ARANCIO IL COLORE #2 DI FONDO
130 POKE53270,PEEK(53270)OR16:REM ATTIVA IL MODO MULTICOLORE
140 C=13*4096+8*256:REM IMPOSTA C PER PUNTARE ALLA MEMORIA COLORE
150 PRINTCHR$(147)"AAAAAAAAAA"
160 FORL=0TO9
170 POKEC+L,8
180 NEXT
```

Il colore di schermo e' bianco, il colore carattere e' nero, un registro colore e' azzurro, l'altro e' arancione.
In realta', nello spazio considerato non vengono introdotti i codici di colore al posto del colore carattere, piuttosto si fanno dei riferimenti ai registri associati a quei colori. Si ha cosi' un risparmio di memoria, in quanto si usano due soli bit per scegliere 16 colori di fondo oppure 8 colori carattere. Si possono inoltre realizzare alcuni trucchetti: semplicemente modificando uno dei registri indiretti si puo' cambiare ogni punto tracciato con quel colore. Percio', tutto quanto e' stato tracciato con i colori di schermo e di fondo puo' essere istantaneamente modificato sull'intero schermo. Quello seguente e' un esempio di come viene modificato il registro colore di fondo #1:

```
100 POKE53270,PEEK(53270)OR16:REM ATTIVA IL MODO MULTICOLORE
110 PRINTCHR$(147)CHR$(18);
120 PRINT"[C= 1]" ::REM BATTE C= & 1 PER ARANCIO O PER FONDO MULTICOLORE
125 REM                NERO
130 FOR=1TO22:PRINTCHR$(65);:NEXT
135 FORT=1TO500:NEXT
140 PRINT"[CTRL 7]" ::REM BATTE CTRL & 7 PER CAMBIARE IL COLORE IN BLU
145 FORT=1TO500:NEXT
150 PRINT"[CTRL 1] BATTI UN TASTO"
160 GETA$:IFA$=""THEN160
170 X=INT(RND(1)*16)
180 POKE53282,X
190 GOTO160
```

Usando il tasto [C=] ed i tasti colore, i caratteri assumono qualunque colore, compresi i caratteri multicolore. Battere ad esempio il seguente comando:

```
POKE53270,PEEK(53270)OR16:PRINT"[CTRL 3][£]";(rosso acceso/rosso multicolore)
```

La parola READY e qualunque altra parola battuta viene visualizzata nel modo multicolore. Un altro controllo di colore riporta al testo regolare. Il seguente programma illustra l'uso dei caratteri multicolore programmabili:

```
10 REM *ESEMPIO 2*
20 REM CREAZIONE DI CARATTERI PROGRAMMABILI MULTICOLORE
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM INTERVALLO CARATTERI DA COPIARE DALLA ROM
36 FORJ=0TO7:REM COPIA TUTTI GLI 8 BYTE DI UN CARATTERE
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPIA UN BYTE
38 NEXTJ,I:REM PASSA AL PROSSIMO BYTE O AL PROSSIMO CARATTERE
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM ATTIVA I/O E KB
40 POKE53272,(PEEK(53272)AND240)+12:REM IMPOSTA PUNTATORE CARATTERE
45 REM                                  ALLA LOCAZIONE 12288
50 POKE53270,(PEEK(53270)OR16
51 POKE53281,0:REM IMPOSTA A NERO IL COLORE DI FONDO #0
52 POKE53282,2:REM IMPOSTA A ROSSO IL COLORE DI FONDO #1
53 POKE53283,7:REM IMPOSTA A GIALLO IL COLORE DI FONDO #2
60 FORCHAR=60TO63:REM PROGRAMMA I CARATTERI DA 60 A 63
```

```
80 FORBYTE=0TO7:REM PREPARA TUTTI GLI 8 BYTE DI UN CARATTERE
100 READNUMBER:REM LEGGE 1/8 DEI DATI DI UN CARATTERE
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM MEMORIZZA I DATI
140 NEXTBYTE,CHAR
150 PRINT"[SHIFT CLR/HOME]"TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 50 VISUALIZZA I NUOVI CARATTERI APPENA DEFINITI
170 GETA$:REM ATTENDE LA PRESSIONE DI UN TASTO
180 IFA$=""THEN170:REM SE NESSUN TASTO VIENE BATTUTO, RICOMINCIA
190 POKE53272,21:POKE53270,PEEK(53270)AND239
191 REM RITORNA AI CARATTERI NORMALI
200 DATA129,37,21,29,93,85,85,85:REM DATI CARATTERE 60
210 DATA66,72,84,116,117,85,85,85:REM DATI CARATTERE 61
220 DATA87,87,85,21,8,8,40,0:REM DATI CARATTERE 62
230 DATA213,213,85,84,32,32,40,0:REM DATI.CARATTERE 63
240 END
```

MODO COLORE DI FONDO ESTESO

Il modo colore di fondo esteso controlla oltre al colore principale, il colore di fondo di ogni singolo carattere. In questo modo, ad esempio, si riesce a visualizzare su uno schermo bianco un carattere blu su fondo giallo.

Il modo colore di fondo esteso ha a disposizione 4 registri, ognuno dei quali puo' essere impostato con uno dei 16 colori.

La memoria del colore viene usata per contenere il colore di fondo nel modo di fondo esteso. Viene usata analogamente a quanto visto per il modo carattere standard.

Il modo carattere esteso, tuttavia, pone un limite al numero di differenti caratteri visualizzabili. Quando si e' nel modo colore esteso, si possono usare solamente i primi 64 caratteri della ROM carattere (oppure i primi 64 caratteri dell'insieme caratteri programmabili). Questo perche' per la selezione del colore di fondo si usano 2 bit del codice carattere. Questo concetto puo' essere espresso anche come segue:

Il codice carattere (il numero da inviare allo schermo con una POKE) della lettera "A" e' 1. Quando si e' nel modo colore esteso, se si invia allo schermo un 1, appare una "A". Se si inviasse normalmente allo schermo un 65, ci si potrebbe aspettare la comparsa di un carattere il cui codice carattere e' 129, cioe' una A "reverse". Nel modo colore esteso questo NON avviene: compare infatti la stessa A "reverse" di prima, ma visualizzata su un diverso colore di fondo. Tutti i codici sono riassunti nella seguente tabella:

CODICE CARATTERE INTERVALLO	BIT 6	BIT 7	REGISTRO DEL COLORE DI FONDO NUMERO	INDIRIZZO
0-63	0	0	0	53281($D021)
64-127	0	1	1	53282($D022)
128-191	1	0	2	53283($D023)
192-255	1	1	3	53284($D024)

Il modo colore esteso viene attivato (ON) impostando a 1 il BIT 6 del registro del VIC-II contenuto nella locazione 53265 ($D011 HEX), tramite la seguente POKE:

```
POKE53265,PEEK(53265)OR64
```

GRAFICA «BIT MAP»

Quando si scrivono giochi, grafici commerciali o altri tipi di programmi, presto o tardi ci si imbatte nella necessita' di una visualizzazione ad alta risoluzione.

Il COMMODORE 64 e' stato progettato proprio per questo: l'alta risoluzione e' disponibile per mezzo della scansione bit map dello schermo. Con questo metodo e' possibile assegnare ad ogni possibile punto di risoluzione (pixel) dello schermo un bit (locazione) in memoria. Se questo bit di memoria e' a 1 , il punto a cui viene assegnato e' acceso, se invece il bit e' a 0 il punto e' spento.

Il modello della grafica ad alta risoluzione presenta un paio di inconvenienti, che spiegano perche' non viene usata appieno. Innanzi tutto, scandire punto a punto l'intero schermo richiede una notevole quantita' di memoria in quanto un simile controllo richiede un bit di memoria per ogni pixel. Poiche' ogni carattere e' una matrice di 8 bit X 8 e ci sono 25 linee di 40 caratteri l'una, la risoluzione per l'intero video e' data da 320 pixel (punti) per 200 pixel, per un totale di 64000 punti distinti, ognuno dei quali richiede un bit di memoria. In altri termini, per tracciare l'intero video sono necessari 8000 byte.

Generalmente, le operazioni ad alta risoluzione sono composte da molte procedure brevi, semplici e ripetitive che purtroppo rallentano la stesura di un programma in BASIC. Tuttavia, questo tipo di procedure viene gestito al meglio dal linguaggio macchina. La soluzione e' quindi scrivere i programmmi interamente in linguaggio macchina, oppure richiamare il linguaggio macchina, usando da BASIC il comando SYS per le sottoprocedure ad alta risoluzione. Si ottine cosi', per la grafica, sia la semplicita' della stesura in BASIC, sia la velocita' del linguaggio macchina. Per raggiungere i comandi ad alta risoluzione al BASIC del COMMODORE 64 e' disponibile anche la CARTUCCIA VSP.

Tutti gli esempi di questo paragrafo vengono dati in Basic per chiarire questi concetti. Passiamo ora ai dettagli tecnici.

IL BIT MAPPING e' una delle tecniche di grafica piu' diffuse; viene usata per creare figure molto dettagliate. Fondamentalmente, quando il COMMODORE 64 entra nel modo bit map, automaticamente visualizza sullo schermo TV una sezione di memoria di 8 K; nel modo bit map si puo' controllare direttamente se un singolo punto del video e' acceso o spento. A disposizione del COMMODORE 64 ci sono due tipi di "bit mapping":

1)MODO BIT-MAP STANDARD(alta risoluzione da 320 X 200 punti)

2)MODO BIT-MAP MULTICOLORE(risoluzione da 160 X 200 punti).

Entrambi sono molto simili al tipo carattere con cui sono stati chiamati: il modo bit map standard ha maggiore risoluzione ma meno possibilita' di colore, il modo bit map multicolore compensa la risoluzione orizzontale con una grande possibilita' di colori per un quadrato di 8 punti X 8.

MODO BIT MAP STANDARD AD ALTA RISOLUZIONE

Il modo bit map standard ha una risoluzione di 320 punti orizzontali per 200 verticali, per una scelta di 2 colori in ogni segmento di 8 X 8 punti. Il modo bit map viene selezionato (ON) impostando a 1 il bit 5 del registro di controllo del VIC-II locato alla posizione 53265 ($D011 HEX), tramite la seguente POKE:

```
POKE 53265,PEEK(53265)OR32
```

Analogamente, il modo bit map viene disattivato (OFF) impostando a 0 il bit 5 dello stesso registro tramite la seguente POKE:

```
POKE53265,PEEK(53265)AND223
```

Prima di entrare nel dettaglio del modo bit map c'e' ancora un punto da apprendere: dove locare l'area per il bit map.

FUNZIONAMENTO

Nel paragrafo riguardante i caratteri programmabili si e' detto che e' possibile impostare a piacere la configurazione di bit di un carattere memorizzato nella RAM. Se dunque si riesce a cambiare il carattere visualizzato sullo schermo, si e' anche in grado di modificare un singolo punto. Questo e'il concetto fondamentale del "bit mapping". L'intero schermo puo'essere riempito di caratteri programmabili,e le modifiche avvengono direttamente nella memoria da cui i caratteri programmabili traggono le configurazioni.

Ciascuna locazione della memoria di schermo, precedentememte usata per controllare il carattere visualizzato, viene ora utilizzata per l'informazione del colore. La locazione 1, ad esempio, anziche' contenere i per far apparire una "A" nell'angolo in alto a sinistra dello schermo, controlla il colore del bit di quello stesso spazio.

Al contrario di quanto avviene nei modi carattere, nel modo bit map i colori dei quadrati non provengono dalla memoria del colore, bensi'dalla memoria dello schermo. I 4 bit piu' alti della memoria di schermo diventano il colore di tutti i bit impostati a 1 nell'area 8x8 controllata da quella locazione della memoria di schermo. I 4 bit piu' bassi diventano il colore di tutti i Bit impostati a 0.

ESEMPIO – Battere:

```
5 BASE =2*4096:POKE53272,PEEK(53272)OR8:REM PONE IL BIT MAP A 8192
10 POKE53265,PEEK(53265)OR:32:REM ENTRA NEL MODO BIT MAP
```

Lanciare (RUN) il programma: sullo schermo compaiono caratteri indecifrabili. Come nel modo schermo normale, si e' azzerato lo schermo ad ALTA RISOLUZIONE (HI-RES) prima di usarlo. In questo caso, pero',non serve a niente battere CLR: si deve infatti azzerare la sezione di memoria usata per i caratteri programmabili. Battere i tasti RUN/STOP e RESTORE , e poi aggiungere alle precedenti le seguenti linee per azzerare lo schermo HI-RES:

```
20 FORI=BASETOBASE+7999:POKE1,0:NEXT:REM AZZERA IL BIT MAP
30 FORI=1024TO2023:POKEI,3:NEXT:REM IMPOSTA I COLORI AZZURRO E NERO
```

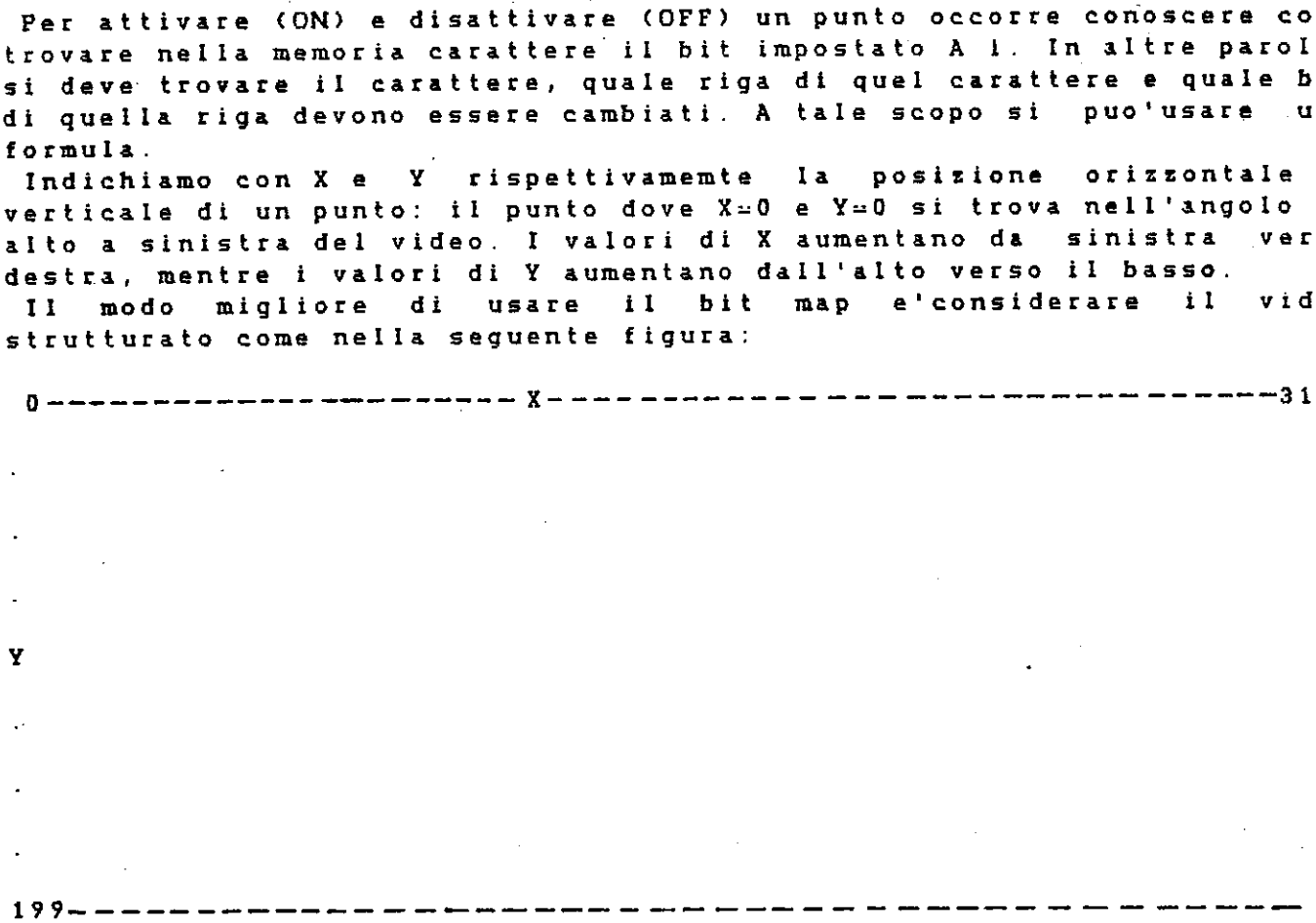
Una volta rilanciato il programma, si puo' osservare l'azzeramento

dello schermo, e successivamente l'azzurro che ricopre l'intero schermo. Vediamo a questo punto come attivare e disattivare punti sul video HI-RES.

Per attivare (ON) e disattivare (OFF) un punto occorre conoscere come trovare nella memoria carattere il bit impostato A 1. In altre parole, si deve trovare il carattere, quale riga di quel carattere e quale bit di quella riga devono essere cambiati. A tale scopo si puo'usare una formula.

Indichiamo con X e Y rispettivamemte la posizione orizzontale e verticale di un punto: il punto dove X=0 e Y=0 si trova nell'angolo in alto a sinistra del video. I valori di X aumentano da sinistra verso destra, mentre i valori di Y aumentano dall'alto verso il basso.

Il modo migliore di usare il bit map e'considerare il video strutturato come nella seguente figura:



Anche se la rappresentazione attuale e'la seguente:

```
------BYTE 0   BYTE 8   BYTE 16   BYTE 24  ..........BYTE 312
L   R  BYTE 1   BYTE 9     ..        ..              BYTE 313
I   I  BYTE 2   BYTE 10    ..        ..              BYTE 314
N   G  BYTE 3   BYTE 11    ..        ..              BYTE 315
E   A  BYTE 4   BYTE 12    ..        ..              BYTE 316
A   0  BYTE 5   BYTE 13    ..        ..              BYTE 317
1      BYTE 6   BYTE 14    ..        ..              BYTE 318
------BYTE 7   BYTE 15    ..        ..              BYTE 319

------BYTE 320 BYTE 328 BYTE 336 BYTE 344 ..... BYTE 632
L   R  BYTE 321 BYTE 329 ..        ..              BYTE 633
I   I  BYTE 322 BYTE 330 ..        ..              BYTE 634
N   G  BYTE 323 BYTE 331 ..        ..              BYTE 635
E   A  BYTE 324 BYTE 332 ..        ..              BYTE 636
A   1  BYTE 325 BYTE 333 ..        ..              BYTE 637
2      BYTE 326 BYTE 334 ..        ..              BYTE 638
------BYTE 327 BYTE 335 ..        ..              BYTE 639
```

I caratteri programmabili di cui e' composta la bit map sono disposti in 25 righe di 40 colonne ciascuna: questa rappresentazione, che e' un buon metodo di organizzazione per il testo, presenta invece alcune difficolta' per quanto concerne il bit map (vedere il paragrafo Modi

Misti).
La seguente formula semplifica il controllo di un punto sullo schermo del bit map:
L'inizio dell'area di memoria del video si chiama BASE. La riga (da 0 24) su cui si trova un carattere e' data da:

```
ROW = INT(Y/8) (320 BYTE per riga)
```

La colonna (da 0 a 39) su cui si trova un carattere e' data da:

```
CHAR = INT(Y/8) (8 BYTE  per colonna)
```

La linea della posizione (da 0 a 7) in cui si trova quel carattere e'data da:

```
LINEA = Y AND 7
```

Il bit (punto) di quel BYTE (linea) e' dato da:

```
BIT = 7-(X and 7)
```

Riunendo queste formule, si ottiene il byte in cui e' locato il punto (X,Y) della memoria carattere:

```
BYTE = BASE+ROW*320+CHAR*8+LINEA
```

Mentre ogni bit della matrice di coordinate (X,Y) sara' attivato da:

```
POKE BYTE,PEEK(BYTE)OR 2 bit
```

Il seguente programma esemplificativo traccia una curva sinusoidale:

```
 50 FORx=0TO319STEP.5:REM ONDA SINUSOIDALE
 60 V=INT(90+80*SIN(X/10)
 70 CH=INT(X/8)
 80 RO=INT(V/8)
 85 LN=VAND7
 90 BY=BASE+RO*320+8*CH+LN
100=BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2 BI)
120 NEXTX
125 POKE1024,16
130 GOTO130
```

Il calcolo di cui alla linea 60 modifica i valori della funzione seno dall'intervallo (-1,1) all'intervallo (10,170). Le istruzioni da 70 100 calcolano il carattere, la riga, il byte ed il bit interessati usando le formule precedentememte illustrate. L'istruzione 125 segnala il termine del programma cambiando il colore dell'angolo in alto a sinistra dello schermo. L'istruzione 130 fa entrare il programma in un LOOP infinito. Al termine della visualizzazione, premere RUN/STOP e quindi RESTORE.
Come ulteriore esempio, si puo' modificare il programma della curva sinusoidale appena illustrato visualizzando un semicerchio. I cambiamemti necessari sono dati nelle istruzioni che seguono:

```
 50 FORX=0TO160:REM DIVIDE A META' LO SCHERMO
```

```
55 Y1=100+SQR(160*X-X*X)
56 Y2=100-SQR(160*X-X*X)
60 FORY=Y1TOY2STEPY1-Y2
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2 BI)
114 NEXT
```

In questo modo viene disegnato un semicerchio nell'area HI-RES dello schermo.

AVVERTENZA: Le variabili BASIC possono sovrapporrsi allo schermo ad alta risoluzione. Se c'e' bisogno di altro spazio di memoria occorre spostare la parte bassa del BASIC oltre l'area dello schermo ad alta risoluzione. Questa situazione NON si verifica in linguaggio macchina, ma SOLAMENTE in BASIC.

MODO BIT MAP MULTICOLORE

Come i caratteri del modo multicolore, il modo bit map multicolore consente di visualizzare fino a 4 differenti colori in ogni sezione 8x8 della bit map. E come nel modo multi-carattere, cio' comporta una riduzione della risoluzione orizzontale (da 320 a 160 punti).

Il modo bit map multicolore usa per la bit map una sezione di 8K di memoria. La selezione dei colori per il modo bit map multicolore avviene:

1) Dal registro 0 del colore di fondo
2) Dalla matrice video (i 4 bit piu' alti danno un colore, i 4 piu' bassi un altro)
3) Dalla memoria del colore

Il modo bit map multicolore viene attivato impostando a 1 il bit 4 della locazione 53270 ($D016 HEX), tramite la seguente POKE:

```
POKE 53265,PEEK(53265)OR32:POKE53270,PEEK(53270)OR 16
```

Analogamemte, il modo Bit Map multicolore viene disattivato impostando a 0 i due registri di cui sopra, tramite la seguente POKE:

```
POKE 53265,PEEK(53265)AND 223:POKE 53270,PEEK(53270)AND 239
```

Come nel modo bit map standard (HI-RES),si crea una corrispondenza univoca tra il segmento di 8K di memoria usato per la visualizzazione e quanto viene visualizzato sullo schermo. Tuttavia, i punti orizzontali sono larghi 2 Bit, e formano nell'area di memoria video un punto per il quale si ha a disposizione uno fra quattro colori.

BIT	PROVENIENZA DELL'INFORMAZIONE SUL COLORE
00	Colore #0 di fondo (colore schermo)
01	4 bit piu' alti della memoria schermo
10	4 bit bassi della memoria schermo
11	Semibyte colore (1 nybble=1/2byte=4bits)

SCROLLING RALLENTATO

Il circuito VIC-II consente lo scrolling rallentato sia in orizzontale che in verticale. Lo scrolling rallentato consiste nello spostamento di un pixel di tutto lo schermo in una direzione. Questo spostamento puo' avvenire verso l'alto, il basso, la destra o la sinistra. Si usa per fare apparire lentamente sullo schermo nuove informazioni mentre quelle vecchie escono lentamente dalla parte opposta.

Anche se il circuito VIC-II svolge gran parte di questo lavoro, lo scrolling effettivo deve essere effettuato da un programma in linguaggio macchina. Il circuito VIC-II possiede la capacita' di porre lo schermo video in una qualunque di 8 posizioni orizzontali ed 8 verticali. Questo posizionamento e' controllato dai registri di scrolling del VIC-II. Questo circuito possiede anche un modo a 24 righe e 38 colonne; queste misure ridotte dello schermo vengono utilizzate per fare posto ai nuovi dati dai quali deve partire lo scrolling.

Lo scrolling rallentato può essere riassunto nei seguenti punti:

1) Stringere lo schermo (il bordo, di conseguenza, si espande);
2) Impostare il registro di scrolling al valore massimo o minimo (secondo la direzione in cui si vuole eseguire lo scrolling);
3) Introdurre i nuovi dati nella posizione dello schermo piu'adatta (i dati non vengono visualizzati);
4) Aumentare (diminuire) il registro di scroll finche' non raggiunga il valore massimo (minimo)
5) Usare la procedura in linguaggio macchina per muovere lo schermo di un carattere nella direzione dello scroll.
6) Ritornare al punto 2).

Per portarsi nel modo a 38 colonne, occorre impostare a 0 il bit 3 della locazione 53270 ($D016 HEX) con la seguente POKE:

```
POKE 53270,PEEK(53270)AND 247
```

Per tornare nel modo a 40 colonne, impostare a 1 lo stesso bit di cui sopra:

```
POKE 53270,PEEK(53270)OR 8
```

Il modo a 24 righe viene attivato impostando a 0 il bit 3 della locazione 53265 ($D011 HEX):

```
POKE 53265,PEEK(53265)AND 247
```

Mentre il ritorno nel modo a 25 righe avverra' impostando a 1 il bit

precedente:

```
POKE 53265,PEEK(53265)OR 8
```

Eseguendo uno scrolling nella direzione X (righe), occorre porre il circuito VIC-II nel modo a 38 colonne, in modo di fare posto ai nuovi dati da cui iniziare lo scroll. Se lo scrolling avviene verso SINISTRA, i nuovi dati vanno piazzati sulla DESTRA, e viceversa. Da notare che la memoria dello schermo ha ancora 40 colonne, ma solamente 38 sono visibili.
Eseguendo uno scrolling nella direzioneY (colonne), occorre porre il circuito VIC-II nel modo a 24 righe. Se lo scrolling avviene verso l'ALTO, i dati vanno piazzati nell'ULTIMA riga,e viceversa. Diversamente dallo scrolling lungo X, dove c'e' un'area non visualizzata da ogni lato dello schermo, nello scrolling lungo Y c'e' solamente un'area non visualizzata. Quando il registro di scrolling nella direzione Y viene impostato a 0, non viene visualizzata la prima linea, che in questo modo viene messa a disposizione dei nuovi dati. Quando invece il registro di scrolling nella direzione Y viene impostato a 7, non viene visualizzata l'ultima riga.
Per eseguire lo scrolling nella direzione X, il registro di scroll e'locato nei bit da 2 a 0 della locazione 53270 ($D016 HEX), in cui si trova il registro di controllo del VIC-II. Come sempre,e'importante modificare solamente questi bit; cio' si ottiene con la seguente POKE:

```
POKE 53270,(PEEK(53270)AND248)+X
```

Dove X e' un numero compreso tra 0 e 7 che indica la posizione X dello schermo. Per eseguire lo scrolling nella direzione Y, il registro di scroll e' locato nei bit da 2 a 0 della locazione 53265 ($D011 HEX),in cui si trova il registro di controllo del VIC-II. Per modificare solamente questi bit si usa la seguente POKE:

```
POKE 53265,(PEEK(53265)AND 248)+Y
```

Dove Y e' un numero compreso fra 0 e 7 che indica la posizione Y dello schermo. Per eseguire lo scroll di un testo partendo dal basso, impostare da 0 a 7 i tre bit piu' bassi della locazione 53265, introdurre altri dati nella linea non visualizzata nella parte bassa dello schermo, e poi ripetere il passaggio. Per eseguire lo scroll da sinistra a destra, impostare da 0 a 7 i tre bit piu' bassi della locazione 53270, riportare una nuova colonna di dati sulla colonna 0 dello schermo, e quindi ripetere il passaggio.
Impostando i bit di scroll a partire da 1, il testo si muove nella direzione opposta.

ESEMPIO – Scrolling eseguito partendo dal basso dello schermo

```
10 POKE53265,PEEK(53265)AND247      :REM PASSA AL MODO 24 RIGHE
20 PRINTCHR$(147)                   :REM AZZERA LO SCHERMO
30 FORX=1TO24:PRINTCHR$(17);:NEXT   :REM POSIZIONA IL CURSORE IN BASSO
40 POKE53265,(PEEK(53265)AND248)+7  :REM SI POSIZIONA PER IL PRIMO
41 REM                                   "SCROLL"
50 PRINT"      HELLO";
60 FORP=60TO0STEP-1
70 POKE53265,(PEEK(53265)AND248)+P
80 FORX=1TO50:NEXT                  :REM CICLO DI RITARDO
90 NEXT:GOTO40
```

ANIMAZIONI

Un'animazione e' un tipo di carattere speciale, definibile dall'utente, che puo' essere visualizzato dovunque sullo schermo Tutto cio' che si deve fare e' comunicare ad un'animazione "a cosa somigliare", "di che colore essere", e "dove comparire", il resto lo fa il circuito VIC-II! Le animazioni possono essere in uno qualunque dei 16 colori disponibili.

Le animazioni possono essere usate con QUALUNQUE modo della grafica, sia bit map, carattere, multicolore, ecc., mantenendo la propria forma in ognuno. L'animazione possiede un colore, un modo (HI-RES o multicolore) ed una forma propri.

Il circuito VIC-II puo' conservare automaticamente, ad un certo istante, fino a 8 animazioni. Un numero maggiore puo' essere visualizzato usando la tecnica di INTERRUZIONE DEL QUADRO.

Caratteristiche delle animazioni:

1) Dimensioni: 24 punti orizzontali X 21 verticali.
2) Controllo di colore indivuduale per ogni animazione.
3) Modo animazione multicolore.
4) Ingrandimento (2X) orizzontale, verticale o entrambi.
5) Priorita' selezionabile tra animazione e fondo.
6) Priorita' fisse tra animazioni.
7) Individuazione dei punti di contatto tra animazioni.
8) Individuazione dei punti di contatto tra animazionie e fondo.

Queste particolari capacita' di un'animazione semplificano la stesura di molti giochi. Poiche' le animazioni sono conservate dall'hardware, e' persino possibile scrivere giochi in BASIC di buona qualita'. Il circuito VIC-II sostiene direttamente fino a 8 animazioni, numerate da 0 a 7. Ogni animazione ha la propria locazione di definizione, i propri registri di posizione e colore ed i propri bit per l'abilitazione e la scoperta dei punti di contatto.

DEFINIZIONE DI UN'ANIMAZIONE

Le animazioni vengono definite come i caratteri programmabili, solo che, essendo di dimensioni maggiori, richiedono una maggiore quantita' di byte. Come gia' detto, un'animazione e' composta da 24 punti X 21, per un totale di 504 punti, pari a 63 bytes (504/8 bit), questi ultimi diposti su 21 righe di 3 byte ciascuna. Una definizione di animazione puo' essere data come segue:

BYTE 0	BYTE 1	BYTE 2
BYTE 3	BYTE 4	BYTE 5
BYTE 6	BYTE 7	BYTE 8
..	..	..
..	..	..
...	..	..
BYTE 60	BYTE 61	BYTE 62

Un altro modo per la creazione di un'animazione e' considerare il blocco di definizione di una'animazione a livello di bit, come illustrato nella figura 3.2.

COLONNA	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
BIT	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
VALORE (ON = 1 x VAL)	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
RIGA 0																								
RIGA 1																								
RIGA 2																								
RIGA 3																								
RIGA 4																								
RIGA 5																								
RIGA 6																								
RIGA 7																								
RIGA 8																								
RIGA 9																								
RIGA 10																								
RIGA 11																								
RIGA 12																								
RIGA 13																								
RIGA 14																								
RIGA 15																								
RIGA 16																								
RIGA 17																								
RIGA 18																								
RIGA 19																								

Figura 3.2 - Blocco per la definizione di un'animazione

In un'animazione standard (HI-RES), ogni bit impostato a 1 viene visualizzato nel colore principale di quell'animazione, mentre ogni bit impostato a 0 e' trasparente e visualizza qualunque dato ci sia dietro di esso, analogamente ad un carattere standard.

Le animazioni multicolore sono simili ai caratteri multicolore. La perdita di risoluzione orizzontale e' bilanciata dalla particolare risoluzione del colore. La risoluzione dell'animazione comprende 12 punti orizzontali X 21 verticali. Ogni punto dell'animazione ha una grandezza doppia, ma il numero di colori visualizzati nell'animazione sale a 4.

PUNTATORI DELL'ANIMAZIONE

Oltre ai 63 byte necessari per definire ogni animazione, e' necessario un altro byte contenente uno spazio alla fine di ogni animazione. Quindi ognuna di esse occupa 64 byte, il che semplifica la ricerca in memoria della definizione dell'animazione, poiche' 64 e' un numero pari e, in binario, una potenza pari.

Ciascuna delle 8 animazioni ha un byte associato chiamato PUNTATORE ALL'ANIMAZIONE. Questi puntatori controllano la locazione in memoria di ogni definizione di animazione. Sono sempre locati negli ultimi 8 byte del segmento di 1K della memoria dello schermo; cio' significa che, sul COMMODORE 64, la loro locazione inizia di solito a 2040 ($07F8 HEX). Tuttavia, questa locazione segue gli spostamenti assegnati alla memoria video.

Ogni puntatore all'animazione contiene un numero compreso fra 0 e 255, che punta alla definizione di quell'animazione. Poiche' ogni definizione di animazione occupa 64 byte, il puntatore puo' "vedere" dovunque nel blocco di memoria di 16 K a cui puo' accedere il circuito VIC-II (256*64 = 16K).

Se il puntatore #0 dell'animazione, situato alla locazione 2040, contiene ad esempio il numero 14, cio' vuol dire che l'animazione 0 sara' visualizzata usando i 64 bytes che iniziano alla locazione 14 X 64 = 896, situata nel buffer della cassetta. Quindi:

```
LOCAZIONE=(BANCO * 16384)+(VALORE DEL PUNTATORE DELL'ANIMAZIONE * 64)
```

Essendo BANCO il segmento di memoria di 16K accessibile dal circuito VIC-II a quell'istante; il suo valore e' compreso fra 0 e 3. Questa formula fornisce il byte di partenza del blocco di 64 byte di definizione dell'animazione.

Quando il circuito VIC-II sta accedendo al BANCO 0 o al BANCO 2, come ricordato in precedenza, in alcune locazioni ci possono essere configurazioni ROM dell'insieme carattere. In queste locazioni NON ci devono essere definizioni di animazioni. Se per qualche motivo si ha bisogno di piu' di 128 differenti definizioni di animazioni, si deve ricorrere ai banchi 1 o 3 che sono sprovvisti di CONFIGURAZIONI ROM.

ATTIVAZIONE DELLE ANIMAZIONI

Il registro di controllo del VIC-II, locato a 53269 ($D015 HEX), e' conosciuto come registro di ABILITAZIONE ANIMAZIONE. Questo registro contiene un bit di ciascuna animazione, in modo da poter controllare se una data animazione sia stata attivata oppure no. La forma del registro e' la seguente:

```
$D015   76543210
```

Ad esempio, per attivare l'animazione 1 e' necessario impostare a 1 il corrispondente bit:

```
POKE53269,PEEK(53269)OR 2
```

Quindi, generalizzando:

```
POKE 53269,PEEK(53269)OR (2↑SN)
```

Essendo SN il numero dell'animazione; quest'ultimo deve essere compreso fra 0 e 7.

NOTA: Un'animazione deve essere attivata (ON) prima di diventare visibile.

DISATTIVAZIONE DELLE ANIMAZIONI

Un'animazione viene disattivata impostando a 0 il corrispondente bit del registro di controllo del VIC-II locato a 53269 ($D015 HEX):

```
POKE 53269, PEEK(53269)AND (255−2↑SN)
```

Essendo SN il numero di attivazione; quest'ultimo deve essere compreso fra 0 e 7.

COLORI

Un'animazione puo' assumere uno qualunque dei 16 colori generati dal circuito VIC-II. Ciascuna animazione ha il proprio registro colore, le cui locazioni di memoria sono:

INDIRIZZO	DESCRIZIONE
53287 ($D027)	REGISTRO COLORE DELL'ANIMAZIONE 0
53288 ($D028)	REGISTRO COLORE DELL'ANIMAZIONE 1
53289 ($D029)	REGISTRO COLORE DELL'ANIMAZIONE 2
53290 ($D02A)	REGISTRO COLORE DELL'ANIMAZIONE 3
53291 ($D02B)	REGISTRO COLORE DELL'ANIMAZIONE 4
53292 ($D02C)	REGISTRO COLORE DELL'ANIMAZIONE 5
53293 ($D02D)	REGISTRO COLORE DELL'ANIMAZIONE 6
53294 ($D02E)	REGISTRO COLORE DELL'ANIMAZIONE 7

Tutti i punti accesi dell'animazione sono visualizzati con il colore contenuto nel registro colore dell'animazione. La rimanente parte dell'animazione e' trasparente, cioe' nei punti "spenti" viene visualizzato qualunque cosa si trovi dietro l'animazione (generalmente lo spazio, locato nel REGISTRO COLORE DELL'ANIMAZIONE 0).

MODO MULTICOLORE

Il modo multicolore mette a disposizione di ogni animazione fino a 4 differenti colori. Tuttavia, come negli altri modi multicolore, la risoluzione orizzontale e' dimezzata: in altri termini, quando si lavora nel modo multicolore dell'animazione, come nel modo multicolore carattere, anziche' avere a disposizione per tutta l'animazione 24 punti, si hanno 12 coppie di punti, ognuna delle quali viene chiamata

coppia di bit (coppia di punti) come un singolo punto dell'intera animazione. La seguente tabella fornisce i valori della coppia di bit necessari all'attivazione di ciascuno dei quattro colori scelti per quell'animazione.

COPPIE DI BIT	DESCRIZIONE
00	COLORE VIDEO TRASPARENTE
01	REGISTRO ANIMAZIONE MULTICOLORE #0 (53285) ($D025)
10	REGISTRO COLORE DELL'ANIMAZIONE
11	REGISTRO ANIMAZIONE MULTICOLORE #1 (53286) ($D026)

IMPOSTAZIONE DI UN'ANIMAZIONE NEL MODO MULTICOLORE

Per attivare un'animazione nel modo multicolore, occorre impostare a ON il registro di controllo del VIC-II locato a 53276 ($D01C HEX):

```
POKE 53276,PEEK(53276) OR (2↑SN)
```

dove SN e' il numero di animazione, che deve essere compreso fra 0 e 7. Per disattivare un'animazione dal modo multicolore, occorre impostare a OFF lo stesso registro precedente:

```
POKE 53276,PEEK(53276) AND (255−2↑SN)
```

dove SN e' il numero di animazione (compreso fra 0 e 7).

ANIMAZIONI INGRANDITE

Il circuito VIC-II ha la capacita' di ingrandire un'animazione in verticale, in orizzontale o in entrambe le direzioni. Ovviamente, ogni punto ingrandito ha le sue dimensioni raddoppiate, senza alcun aumento della risoluzione...semplicemente, l'animazione diventa piu' grande.

L'ingrandimento orizzontale di un'animazione si ottine impostando a ON (mettendo a 1) il bit corrispondente del registro di controllo del VIC-II locato a 53277 ($D01D HEX):

```
POKE 53277,PEEK(53277)OR (2↑SN)
```

dove SN e' il numero di animazione (compreso tra 0 e 7).

Per riportare in grandezza normale un'animazione ingrandita in orizzontale, occorre impostare a OFF (mettere a 1) il corrispondente bit del registro di controllo del VIC-II locato a 53277 ($01D HEX):

```
POKE 53277,PEEK(53277)AND (255−2↑SN)
```

dove SN e' il numero di animazione, che deve essere compreso fra 0 e 7.

Per espandere un'animazione nella direzione verticale, occorre impostare a ON (mettere a 1) il corrispondente bit del registro di controllo del VIC-II, locato a 53271 ($D017 HEX):

```
POKE 53271,PEEK(53271)OR (2↑SN)
```

dove SN e' il numero di animazione (compreso fra 0 e 7).

Per riportare in grandezza normale un'animazione ingrandita in verticale, occorre impostare ad OFF (mettere a 0) il corrispondente

bit del registro di controllo del VIC-II locato a 53271 ($D017 HEX):

```
POKE 53271,PEEK(53271)AND (255-2↑SN)
```

dove SN e' il numero di animazione (compreso fra 0 e 7).

POSIZIONAMENTO DELLE ANIMAZIONI

Una volta creata un'animazione, il COMMODORE 64 usa tre registri di posizionamento per muovere l'animazione sullo schermo:

1) REGISTRO POSIZIONE X DELL'ANIMAZIONE (ORIZZONTALE)
2) REGISTRO POSIZIONE Y DELL'ANIMAZIONE (VERTICALE)
3) BIT PIU' SIGNIFICATIVO DEL REGISTRO POSIZIONE X

Ogni animazione ha 512 posizioni possibili lungo l'asse X e 256 lungo l'asse Y.

I registri di posizione X e Y lavorano in coppia. Le locazioni di questi registri appaiono in memoria come segue: prima il registro X per l'animazione 0, poi il corrispondente registro Y; segue poi la coppia di registri dell'animazione 1 e cosi' via. I 16 registri X e Y sono seguiti dal bit piu' significativo della posizione X (MSB X), locato nel proprio registro.

La tabella seguente illustra la locazione del registro di posizione di ogni animazione; il loro uso avviene per mezzo di una POKE.

LOCAZIONE		DESCRIZIONE
DECIMALI	HEX	
53248	($D000)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 0
53249	($D001)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 0
53250	($D002)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 1
53251	($D003)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 1
53252	($D004)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 2
53253	($D005)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 2
53254	($D006)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 3
53255	($D007)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 3
53256	($D008)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 4
53257	($D009)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 4
53258	($D010)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 5
53259	($D011)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 5
53260	($D012)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 6
53261	($D013)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 6
53262	($D014)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 7
53263	($D015)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 7
53264	($D016)	REGISTRO POSIZIONE X DELL'ANIMAZIONE X MSB

La posizione di un'animazione viene calcolata a partire dall'angolo IN ALTO A SINISTRA della zona di 24 punti X 21 in cui e' stata disegnata l'animazione. NON importa da quanti punti e' formata un'animazione. Anche se, come animazione, si usasse un solo punto, e lo si volesse visualizzare al centro dello schermo, si dovrebbe lo stesso calcolare l'esatto posizionamento a partire dalla locazione dell'angolo in alto a sinistra.

Posizionamento Verticale

L'allestimento delle posizioni orizzontali e' leggermente piu' complesso del posizionamento verticale, per cui comincermo l'esposizione da quest'ultimo.

Nella direzione verticale dello schermo TV si possono programmare indipendentemente 200 posizioni di punti; i registri di posizione Y dell'animazione possono trattare numeri fino a 255. Si ha quindi un'eccedenza nelle locazioni del registro disponibili per il movimento di un'animazione. E' anche possibile il movimento lento di un'animazione, per il quale sono a disposizione piu' di 200 valori.

Partendo dall'alto del video, il primo valore che fa comparire sul video una animazione non ingrandita e', per la direzione verticale, 30, mentre se l'animazione e' ingrandita tale valore e' 9 (diversamente, si avrebbe una perdita di risoluzione, poiche' ogni punto e' alto il doppio, e la posizione iniziale e' ancora calcolata a partire dall'angolo in alto a sinistra dell'animazione).

Il primo valore di Y per il quale un'animazione (ingrandita o no) compare interamente sullo schermo (vengono visualizzate tutte le 21 linee disponibili) e' 50.L'ultimo valore di Y per il quale un'animazione non ingrandita compare interamente sullo schermo e' 229; se l'animazione e' ingrandita, tale valore si riduce a 208. Il primo valore di Y per il quale un'animazione e' completamente fuori schermo e' 250.

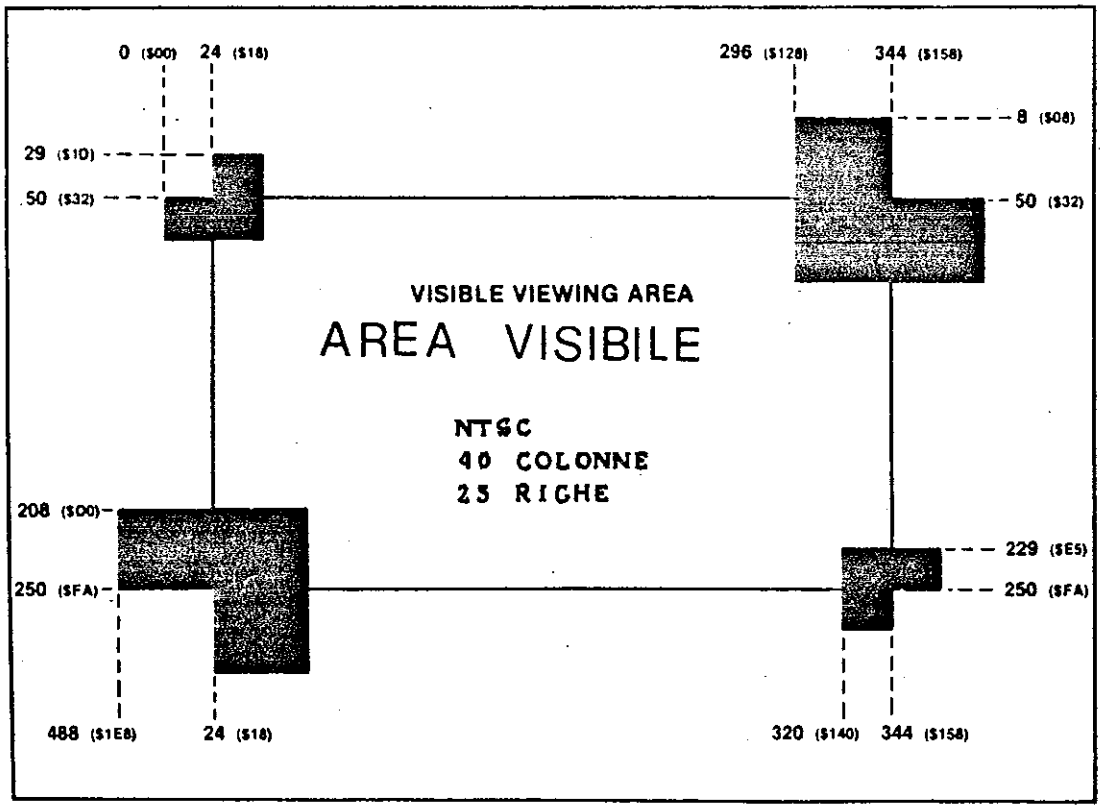
ESEMPIO:

```
10 PRINT"□"  : REM AZZERA LO SCHERMO
20 POKE2040,13 : REM LEGGE DAL BLOCCO 13 I DATI DELL'ANIMAZIONE 0
30 FORI=0T062:POKE832+I,129:NEXT : REM INSERISCE NEL BLOCCO 13
31 REM I DATI DELL'ANIMAZIONE (13*64=832)
40 V=53248! : REM SI DISPONE ALL'INIZIO DEL CIRCUITO VIDEO
50 POKEV+21,1 : REM ABILITA L'ANIMAZIONE 1
60 POKEV+39,1 : REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
70 POKEV+1,100 : REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
80 POKEV+16,0:POKEV,100 : REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
```

(SHIFT CLR/HOME)

POSIZIONE ORIZZONTALE

Il posizionamento nella direzione orizzontale e' piu' complesso, poiche' ci sono 256 posizioni, il che comporta l'introduzione di un bit speciale, il nono, usato per controllare la posizione lungo l'asse X. Aggiungendo, quando necessario, questo bit, un'animazione viene ad avere a disposizione 512 possibili posizioni per il movimento destra/sinistra nella direzione X. Si ha cosi' un aumento delle possibili combinazioni che possono essere visualizzate sulla parte visibile dello schermo. Anche se ogni animazione puo' assumere una posizione compresa fra 0 e 511, lo schermo consente la visualizzazione dei valori compresi tra 24 e 343. Se la posizione X di un'animazione e' maggiore di 255 (sulla destra dello schermo), il bit del registro POSIZIONE DEL BIT PIU' SIGNIFICATIVO nella posizione X deve essere impostato a 1. Se la posizione X di un'animazione e' minore di 256 (sulla sinistra dello schermo), allora l'MSB nella direzione X di quell'animazione deve essere 0. I bit da 0 a 7 del registro MSB nella direzione X corrispondono rispettivamente alle animazioni 0 a 7.



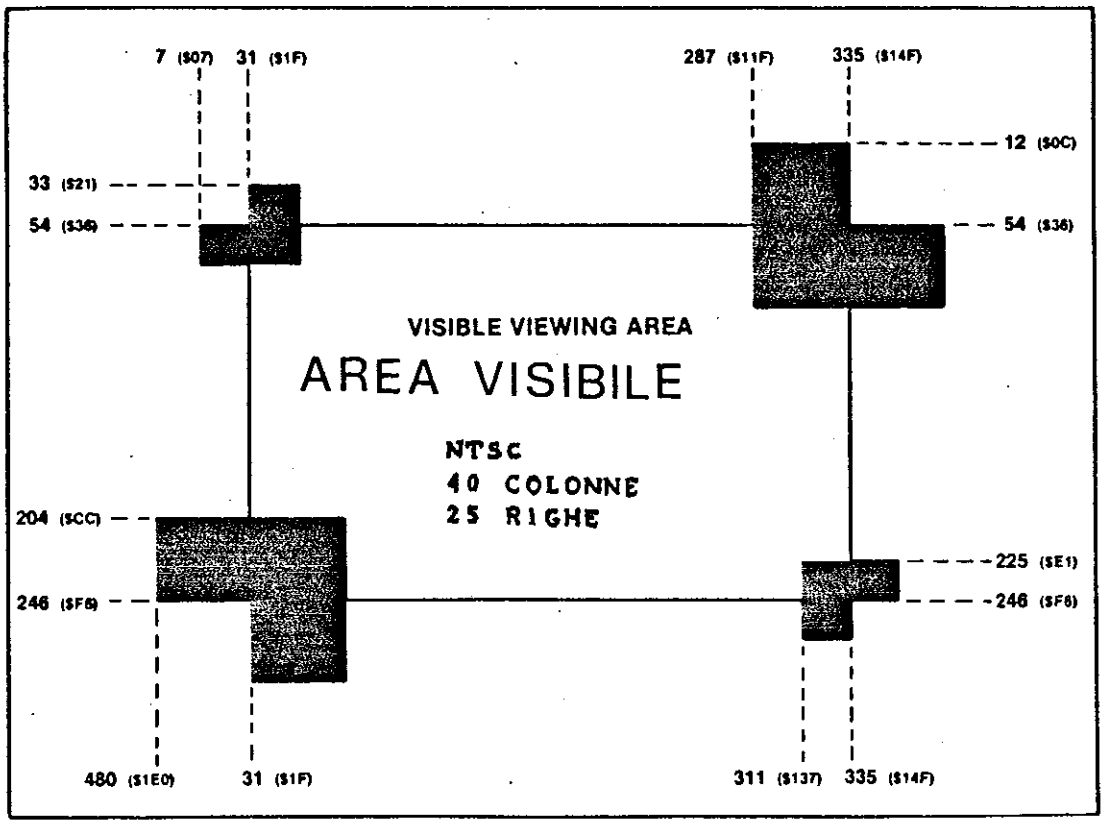


Figura 3.3 - Tabelle per il posizionamento delle animazioni

ESEMPIO:

```
10 PRINT"[SHIFT CLR HOME]"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV+1,100
80 FORJ=0TO347
90 HX=INT(J*256):LX=J-256*HX
100 POKE,LX:POKEV,LX:POKEV+16,HX:NEXT
```

Quando si muovono animazioni ingrandite verso la sinistra dello schermo nella direzione X, occorre far partire l'animazione DA FUORI SCHERMO sulla DESTRA, in quanto un'animazione ingrandita e' piu' grande della quantita' di spazio disponibile sulla sinistra dello schermo.

ESEMPIO:

```
10 PRINT"[SHIFT CLR HOME]"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1:POKEV+23,1:POKEV+29,1
70 POKEV+1,100
80 J=488
90 HX=INT(J*256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX
110 J=J+1:IFJ<511THENJ=0
120 IFJ>488ORJ<348GOTO90
```

Per il posizionamento di una animazione si veda la figura 3.3.
L'uso di questi valori consente di posizionare un'animazione dovunque. E' facile realizzare un movimento molto lento spostando l'animazione di un solo punto alla volta.

RIASSUNTO DEL POSIZIONAMENTO DELLE ANIMAZIONI

Animazioni non ingrandite sono visibili, almeno parzialmente, nel modo a 40 colonne X 25 riche, impostando:

$$1 <= X <= 343$$

$$30 <= Y <= 249$$

Nel modo a 38 colonne, il parametro X assume

$$8 <= X <= 334$$

Nel modo a 24 righe, il parametro Y assume

$$34 <= Y <= 245$$

Animazioni ingrandite sono visibili, almeno parzialmente, nel modo a

40 colonne X 25 righe, impostando

489 > = X < = 343

9 > = Y < = 249

Nel modo a 38 colonne, il parametro X assume

496 > = X < = 334

Nel modo a 24 righe, il parametro Y assume

13 < = Y < = 245

PRIORITÀ DI VISUALIZZAZIONE DELLE ANIMAZIONI

Le animazioni possono incrociarsi, come pure passare davanti o dietro ad altri oggetti presenti sullo schermo. Si ha cosi' un vero effetto tridimensionale per i giochi.
La priorita' tra animazioni e' fissa: la piu' alta e' quella dell'animazione 0, la piu' bassa e' quella dell'animazione 7. Se ad esempio si incrociano le animazioni 1 e 6, la 1 passera' davanti alla 6.
Percio', quando si stabilisce quali figure far apparire in primo piano, occorre assegnare loro numeri di animazione piu' bassi di quelli delle animazioni in secondo piano.

NOTA: E' possibile un effetto "finestra": se un'animazione ha una priorita' maggiore di un'altra, ed ha al suo interno dei "buchi" (zone di punti non impostati a 1 e quindi spenti), allora l'animazione di priorita' piu' bassa sara' visibile attraverso tali "buchi". Cio' avviene anche fra un'animazione e un dato che compare sullo sfondo.

La priorita' tra le animazioni e lo sfondo e' controllabile per mezzo del registro di priorita' ANIMAZIONE-SFONDO posto nella locazione 53275 ($D01B HEX). Questo registro contiene un bit per ogni animazione: se questo bit vale 0, l'animazione ha una priorita' maggiore dello sfondo, cioe' compare davanti ai dati presenti sullo sfondo; se invece il bit vale 1, l'animazione ha una priorita' minore dello sfondo, per cui l'animazione compare dietro ai dati presenti sullo sfondo.

DETERMINAZIONE DEI PUNTI DI CONTATTO

Una delle caratteristiche piu' interessanti del circuito del VIC-II e' la capacita' di scoprire i punti di contatto tra le animazioni o tra le animazioni e i dati che compaiono sullo sfondo. Un contatto avviene quando una parte "non zero" (i cui bit sono impostati a 1, visualizzando quindi quei particolari punti) di un'animazione si sovrappone ad una parte "non zero" di un'altra animazione o di un carattere.

PUNTI DI CONTATTO TRA ANIMAZIONI

I punti di contatto fra animazioni vengono riconosciuti dal computer e segnalati nel registro contatto tra animazioni locato a 53278 ($D01E

HEX) del registro di controllo del circuito VIC-II.
Il registro contatto fra animazioni contiene un bit per ogni animazione. Se tale bit e' a 1, cio' vuol dire che la corrispondente animazione e' interessata in un contatto. I bit di questo registro rimangono impostati fino alla successiva lettura (che avviene tramite PEEK). Una volta letto, il registro viene automaticamente azzerato, per cui e' consigliabile salvare il contenuto in una variabile per tutto il tempo che se ne ha bisogno.

NOTA: I contatti possono avvenire anche fra animazioni fuori schermo

Questo punto di contatto viene riconosciuto dal registro di contatto ANIMAZIONI-DATI locato a 53279 ($D01F HEX) del registro di controllo del circuito VIC-II.
Il registro di contatto ANIMAZIONE-DATI contiene un bit per ogni animazione: se questo e' a 1, allora la corrispondente animazione e' interessata in un contatto. I bit di questo registro rimangono impostati fino alla successiva lettura (che avviene tramite PEEK). Una volta letto, il registro viene azzerato automaticamente, per cui e' di nuovo consigliabile il salvataggio del contenuto in una variabile.

NOTA: Il dato multicolore 01 e' considerato trasparente al contatto, anche se viene visualizzato. Quando si imposta uno schermo di fondo e' consigliabile procedere in modo da evitare un contatto con 01 nel modo multicolore.

```
10 REM ANIMAZIONI - ESEMPIO 1
20 REM LA MONGOLFIERA
30 VIC=13*4096:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,1:REM ABILITA L'ANIMAZIONE 0
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,1:REM INGRANDISCE L'ANIMAZIONE 0 LUNGO LA DIREZIONE Y
38 POKEVIC+29,1:REM INGRANDISCE L'ANIMAZIONE 0 LUNGO LA DIREZIONE X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
220 POKEVIC+39,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
250 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
251 REM              UN'ANIMAZIONE
300 READA:REM :LEGGE UN BYTE
310 POKE192*64+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXTY:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIC):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO DELLO...
370 REM SCHERMO, ALLORA INVERTE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM SOMMA DELTA X A X
470 X=XAND255 REM CONTROLLA CHE X SIA COMPRESO NEI LIMITI CONSENTITI
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INSERISCE IL NUOVO VALORE DI X NELLA POSIZIONE X
491 REM              DELL'ANIMAZIONE 0
510 POKEVIC+1,Y:REM INSERISCE IL NUOVO VALORE DI Y NELLA POSIZIONE Y
511 REM                DELL'ANIMAZIONE 0
530 GOTO40
600 REM ***** DATI ANIMAZIONE *****
610 DATA0,127,0,1,255,192,3,255,224,3,231,224
620 DATA7,217,240,7,220,240,7,217,240,3,231,224
630 DATA3,255,224,3,255,224,2,255,160,1,127,64
640 DATA1,62,64,0,156,128,0,156,128,0,73,0,0,73,0,
650 DATA0,62,0,0,62,0,0,62,0,0,28,0,0
```

```
10 REM ANIMAZIONI - ESEMPIO 2
20 REM ANCORA LA MONGOLFIERA
30 VIC=13*4096:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,63:REM ABILITA LE ANIMAZIONI 0-5
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,3:REM INGRANDISCE LE ANIMAZIONI 0 E 1 LUNGO Y
38 POKEVIC+29,3:REM INGRANDISCE LE ANIMAZIONI 0 E 1 LUNGO X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
50 POKE2041,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 1
60 POKE2042,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 2
70 POKE2043,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 3
80 POKE2044,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 4
90 POKE2045,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 5
100 POKEVIC+4,30:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 2
110 POKEVIC+5,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 2
120 POKEVIC+6,65:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 3
130 POKEVIC+7,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 3
140 POKEVIC+8,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 4
150 POKEVIC+9,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 4
160 POKEVIC+10,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 5
170 POKEVIC+11,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 5

175 PRINT"[CTRL 2]"TAB(15)"QUESTE SONO DUE ANIMAZIONI HI-RES"
    [SHIFT CLR HOME]
176 PRINTTAB(55)"UNA SOPRA L'ALTRA"
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
200 POKEVIC+2,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 1
210 POKEVIC+3,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 1
220 POKEVIC+39,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
230 POKEVIC+41,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 2
240 POKEVIC+43,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 4
250 POKEVIC+40,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 1
260 POKEVIC+42,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 3
270 POKEVIC+44,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 5
280 FORX=192TO193:REM INIZIO DEL CICLO DI DEFINIZIONE
281 REM                  DELLE ANIMAZIONI
290 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
291 REM              UN'ANIMAZIONE
300 READA:REM LEGGE UN BYTE
310 POKEX*64+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXTY,X:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIV):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO...
370 REM DELLO SCHERMO, ALLORA INVERTE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=3
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM ADD DELTA X TO X
470 X=XAND255:REM CONTROLLA CHE X SIA NEI LIMITI CONSENTITI
```

```
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE X
491 REM            DELL'ANIMAZIONE 0
500 POKEVIC+2,X:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE X
501 REM              DELL'ANIMAZIONE 1
510 POKEVIC+1,Y:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE Y
511 REM              DELL'ANIMAZIONE 0
520 POKEVIC+3,Y:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE Y
521 REM              DELL'ANIMAZIONE 1
530 GOTO340
600 REM ***** DATI ANIMAZIONI *****
610 DATA0,255,0,3,153,192,7,24,224,7,56,224,14,126,112,14,126,112,14,
    126,112
620 DATA6,126,96,7,56,224,7,56,224,1,56,128,0,153,0,0,90,0,0,56,0
630 DATA0,56,0,0,0,0,0,0,0,0,126,0,0,42,0,0,840,0,40,0,0
640 DATA0,0,0,0,102,0,0,231,0,0,195,0,1,129,128,1,129,128,1,129,128
650 DATA1,129,128,0,195,0,0,195,0,4,195,32,2,102,64,2,36,64,1,0,128
660 DATA1,0,128,0,153,0,0,153,0,0,0,0,0,84,0,0,42,0,0,20,0,0
```

```
10 REM ANIMAZIONI - ESEMPIO 3
20 REM THE HOT AIR GORF
30 VIC=53248:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,1:REM ABILITA L'ANIMAZIONE 0
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,1:REM INGRANDISCE L'ANIMAZIONE 0 NELLA DIREZIONE Y
38 POKEVIC+29,1:REM INGRANDISCE L'ANIMAZIONE 0 NELLA DIREZIONE X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
50 POKEVIC+28,1:REM ATTIVA IL MODO MULTICOLORE
60 POKEVIC+37,7:REM IMPOSTA IL MULTICOLORE 0
70 POKEVIC+38,4:REM IMPOSTA IL MULTICOLORE 1
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
220 POKEVIC+39,2:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
290 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
291 REM              UN'ANIMAZIONE
300 READA:REM LEGGE UN BYTE
310 POKE12288+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXT Y:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIC):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO...
370 REM DELLO SCHERMO, ALLORA INVERTE DELTA Y
380 IF X=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDX=1THENX=-1:SIDE=1
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM SOMMA DELTA X A X
470 X=XAND255:REM SI ASSICURA CHE X SIA COMPRESO NELL'INTERVALLO
471 REM              CONSENTITO
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INTRODUCE IL NUOVO VALORE DI X NELLA POSIZIONE X
491 REM              DELL'ANIMAZIONE 0
510 POKEVIC+1,Y:REM INTRODUCE IL NUOVO VALORE DI Y NELLA POSIZIONE Y
511 REM                DELL'ANIMAZIONE 0
520 GETA$:REM LEGGE UN CARATTERE PROVENIENTE DALLA TASTIERA
521 IFA$="M"THENPOKEVIC+28,1:REM MULTICOLORE SELEZIONATO DALL'UTENTE
522 IFA$="H"THENPOKEVIC+28,0:REM ALTA RISOLUZIONE SELEZIONATA
523 REM                              DALL'UTENTE
530 GOTO340
600 REM ***** DATI ANIMAZIONE *****
610 DATA64,0,1,16,170,4,6,170,144,10,170,160,42,170,168,41,105,104,
169,235,106
620 DATA169,235,106,169,235,106,170,170,170,170,170,170,170,170,
170,170,170,170
630 166,170,154,169,85,106,170,185,170,42,170,168,10,170,160,1,0,
64,1,0,64
640 DATA5,0,80,0
```

ALTRE CARATTERISTICHE DELLA GRAFICA

AZZERAMENTO DELLO SCHERMO

Il bit 4 del registro di controllo del Vic-II presiede la funzione di azzeramento dello schermo. Tale bit si trova nel registro di controllo alla locazione 53265 ($D011 HEX). Quando viene impostato a 1, lo schermo e' nello stato normale, quando e' impostato a 0 (disattivato) l'intero schermo assume il colore del bordo.
L'azzeramento dello schermo non comporta la perdita dei dati.semplicemente questi ultimi non vengono piu' visualizzati. Per azzerare lo schermo si usi la seguente POKE:

```
POKE 53265,PEEK(53265)AND 239
```

Mentre il ritorno dello schermo alla posizione iniziale e'dato da:

```
POKE 53265,PEEK(53265)OR 16
```

NOTA: Disabilitare lo schermo rende il processore leggermente piu' veloce. Di conseguenza, anche il programma attualmente residente in memoria acquista velocita'.

REGISTRO DI QUADRO (Televisivo)

Il registro di quadro si trova nel circuito VIC-II alla locazione 53266 ($D012 HEX), ed ha un doppio scopo. La lettura di questo registro ritorna gli 8 bit piu' bassi della attuale posizione del quadro.
La posizione di quadro del bit piu' significativo e' contenuta nel registro locato in 53265 ($D011 Hex). Il registro di quadro si usa per impostare temporanei cambiamenti del video, in modo da liberarsi dallo sfarfallio (tremolio) delle immagini sullo schermo. Questi cambiamenti devono essere eseguiti quando il quadro non si trova nella zona visibile del video, in cui le posizioni dei punti sono comprese fra 51 e 251.
Quando si scrive nel registro di quadro (compreso l'MSB), il numero scritto viene salvato per poter essere usato con la funzione di paragone del quadro. Quando il valore attuale di quest'ultimo uguaglia il numero scritto nel registro di quadro, viene impostato a 1 il bit del registro interruzione del circuito VIC-II la cui locazione e' 53273 ($D019 HEX).

NOTA: Attivando il bit appropriato, si genera un'interruzione(IRQ)

REGISTRO DI STATO DELL'INTERRUZIONE

Il registro di stato dell'interruzione contiene lo stato attuale di ogni sorgente di interruzione. Lo stato attuale del bit 2 del registro interruzione viene impostato a 1 quando si scontrano due animazioni; un analogo discorso vale anche peri bit 0...3 illustrati di seguito. Il bit 7 viene impostato a 1 qualunque interruzione avvenga. Il registro di stato dell'interruzione si trova nella locazione 53273

($D019 HEX), ed ha la seguente configurazione:

CIRCUITO	BIT	DESCRIZIONE
IRST	0	Impostato quando il contatore di quadro corrente e' uguale al contatore di quadro registrato
IMDC	1	Impostato da un contatto ANIMAZIONE-DATI (solo per il primo fino al ripristino successivo)
IMCC	2	Impostato da un contatto ANIMAZIONE-ANIMAZIONE (solo per il primo fino al prossimo ripristino)
ILP	3	Impostato dalla transizione negativa della penna ottica(uno per inquadratura)
IRQ	7	Impostato dall'attivazione di un circuito latch e abilitato

Una volta impostato, un bit di interruzione viene trascritto nell'apposito circuito "latch"; al momento del suo utilizzo, occorre azzerare tale bit impostandolo a 1. Cio' consente un trattamento selettivo dell'interruzione, senza il bisogno di registrare gli altri bit di interruzione.

II REGISTRO ABILITATORE DELL'INTERRUZIONE e' locato a 53274 ($D01A HEX), ed ha la stessa forma del registro di stato dell'interruzione. Per tutto il tempo in cui il corrispondente bit del registro abilitatore dell'interruzione rimane impostato a 1, da questa sorgente non puo' derivare alcuna interruzione. Il registro di stato dell'interruzione puo' essere ancora consultato per ottenere delle informazioni, ma non genera piu' altre interruzioni.

Per abilitare una richiesta di interruzione il corrispondente bit di abilitazione dell'interruzione (come illustrato nella precedente tabella) deve essere impostato a 1.

Questa struttura dell'interruzione consente di usare modi di schermo distinti. Ad esempio, si puo' avere meta' schermo nel modo bit map, meta' testo, piu' di otto animazioni contemporanee, ecc. Tutto sta nell'uso corretto delle interruzioni. Per esempio, se si vuole che la parte alta dello schermo sia nel modo bit map e la parte bassa nel modo testo, occorre impostare il registro di confronto del quadro in modo che divida a meta' lo schermo (come esposto in precedenza). Al verificarsi dell'interruzione, impostare il circuito VIC-II in modo che prelievi caratteri dalla ROM, poi impostare il registro di confronto del quadro per generare un'interruzione quando si trova nella parte alta dello schermo. Al verificarsi di questa nuova interruzione, impostare il circuito VIC-II in modo che prelievi caratteri dalla RAM (modo bit map).

Si possono anche visualizzare piu' di 8 animazioni nello stesso tempo, anche se il BASIC non e' abbastanza veloce per gestire bene questa situazione; percio', se si vogliono usare le interruzioni del video, e' consigliabile lavorare in linguaggio macchina.

COMBINAZIONI CONSIGLIATE DEI COLORI DI SCHERMO E CARATTERE

I TV COLOR presentano limitazioni nell'accostamento di alcuni colori sulla stessa riga, producendo immagini confuse. La seguente tabella indica quali combinazioni colore evitare, e quali invece risaltano di piu'.

COLORE CARATTERE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	#	*	#	*	*	@	#	*	*	#	*	*	*	*	*	*
1	*	#	*	#	*	*	*	#	@	*	@	*	*	#	*	*
2	#	*	#	#	@	#	#	*	*	#	*	#	#	#	#	@
3	*	#	#	#	#	@	*	#	#	#	#	@	#	#	@	#
4	*	@	#	#	#	#	#	#	#	#	#	#	#	#	#	@
5	*	@	#	@	#	#	#	#	#	#	#	@	#	*	#	@
6	@	*	#	*	#	#	#	#	#	#	#	#	#	@	*	*
7	*	#	*	#	#	#	@	#	@	*	@	*	*	#	#	#
8	@	*	*	#	#	#	#	*	*	*	#	#	#	#	#	@
9	#	*	#	#	#	#	#	*	*	#	*	#	#	#	#	*
10	@	@	*	#	#	#	#	@	#	*	#	#	#	#	#	@
11	*	*	#	@	#	#	#	*	#	#	#	#	*	*	@	*
12	*	*	@	#	#	#	@	#	#	@	#	*	#	#	#	*
13	*	#	#	#	#	*	@	#	#	#	#	*	#	#	#	#
14	*	*	#	*	#	#	*	#	#	#	#	@	#	#	#	@
15	*	*	*	#	@	@	*	#	#	@	@	*	*	#	@	#

* = ECCELLENTE
@ = BUONO
\# = SCARSO

PROGRAMMAZIONE DELLE ANIMAZIONI UN ULTERIORE SGUARDO

Questo paragrafo intende dare un approccio semplificato alle animazioni.

COSTRUZIONE DI ANIMAZIONI DA BASIC – UN BREVE PROGRAMMA

Ci sono almeno tre differenti tecniche di programmazione BASIC che consentono di creare sul COMMODORE 64 immagini grafiche ed animazioni: si puo' usare la grafica di sistema (cfr. Appendice B), creare dei caratteri personalizzati (vd. Definizioni di Carattere), oppure, meglio di tutto....usare la "grafica animata" di sistema. Per illustrare l'estrema semplicita' d'uso, basta osservare il seguente programma, uno dei piu' corti programmi di animazione che si possono scrivere in BASIC:

```
10 PRINT "□"          (SHIFT CLR HOME)
20 POKE2040,13
30 FORS=832TO832+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV,24
80 POKEV+1,100
```

Questo programma contiene tutte le "componenti" principali necessarie per la realizzazione di qualunque animazione. I numeri delle istruzioni POKE sono stati presi dalla tabella per la costruzione delle animazioni. Questo programma definisce la prima animazione – Animazione 0 – come un cubo bianco; la descrizione del programma e' la

seguente:

LINEA 10 Azzera lo schermo

LINEA 20 Imposta il puntatore dell'animazione al valore dal quale il COMMODORE 64 deve leggere i dati dell'animazione. L'animazione 0 e' posta a 2040, la 1 a 2041...la 7 a 2047. Si possono impostare tutti e 8 i puntatori delle animazioni a 13 sostituendo la linea 40 con:

```
FOR SP=2040TO2047:POKE SP,13:NEXT SP
```

LINEA 30 Inserisce la prima animazione (ANIMAZIONE 0) nei 63 bytes della memoria RAM del COMMODORE 64 a partire dalla posizione 832 (ogni animazione richiede 63 bytes). La prima animazione e' "indirizzata" nelle locazioni di memoria da 832 a 894.

LINEA 40 Uguaglia la variabile "V" a 53248, indirizzo di partenza del circuitovideo. Cio' permette di usare la forma (V + numero) per impostare l'animazione; inoltre, al momento dell'impostazione dell'animazione con l'istruzione POKE, tale forma non modifica la memoria, consentendo cosi' di lavorare con numeri piu' piccoli. Ad esempio, alla linea 50 abbiamo scritto POKE V+21: e' equivalente a POKE 53248+21 o a POKE 53269, solo che richiede meno spazio e si ricorda meglio.

LINEA 50 Abilita l'animazione 0. Ci sono 8 animazioni, numerate da 0 a 7. Per attivare un'animazione singola o un gruppo di esse, basta scrivere POKE V+21 seguito da un numero compreso fra 0 (disattiva tutte le animazioni) e 255 (attiva tutte le animazioni). La seguente tabella illustra come attivare e disattivare una o piu' animazioni:

TUTTO ON	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7	TUTTO OFF
V+21,255	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128	V+21,0

Si possono attivare anche combinazioni di animazioni: ad esmpio, POKE V+21,129 attiva le due animazioni 0 e 7 sommando i due numeri di attivazione (vd. tabella per la costruzione delle animazioni).

LINEA 60 Imposta il colore dell'animazione 0. Ci sono 16 colori a disposizione di un'animazione, numerati da 0 (nero) a 15 (grigio). Ogni animazione richiede una POKE diversa per ogni colore impostato da V+39 a V+46. POKE V+39 colora l'animazione 0 di bianco; POKE V+46,15 colora l'animazione 7 di grigio (vd. tabella per la costruzione delle animazioni).

Quando si crea un'animazione, questa rimane in memoria finche' non viene disattivata o ridefinita, oppure non viene spenta la macchina. Si puo' cosi' cambiare colore, disposizione e pefino luminosita' all'animazione, sia in modo DIRETTO che IMMEDIATO, quest'ultimo utile in caso di stampa. Ad esempio, far girare il precedente programma, poi battere in modo DIRETTO:

```
POKE V+39,8
```

è quindi RETURN : l'animazione sullo schermo e' diventata ARANCIONE; inserendo allo stesso modo i numeri da 0 a 15 si possono visualizzare

gli altri colori. Poiche' si e' agito in modo DIRETTO, rilanciando il programma l'animazione assume il colore originale (bianco).

LINEA 70 Determina la POSIZIONE ORIZZONTALE "X" dell'animazione sullo schermo. Questo numero rappresenta la locazione dell'ANGOLO IN ALTO A SINISTRA dell'animazione. La posizione orizzontale piu' lontana visibile a sinistra e' 24, sebbene si possa spostare l'animazione FUORI SCHERMO fino alla posizione 0.

LINEA 80 Determina la POSIZIONE VERTICALE Y dell'animazione sullo schermo. Il programma posiziona l'animazione nel punto (X=24, Y=100),rispettivamente posizione orizzontale e verticale. Un'altra posizione puo' essere ottenuta digitando:

```
POKE V,24:POKE V+1,50
```

Dopo aver battuto RETURN , la figura si sposta nell'angolo in alto a sinistra dello schermo; per portarla nell'angolo in basso a sinistra, digitare:

```
POKE V,24:POKE V+1,229
```

Ogni numero da 832 a 895 rappresenta, nell'indirizzo dell'animazione 0, un blocco di 8 pixel, con 3 blocchi di 8 pixel in ogni riga orizzontale dell'animazione. Il ciclo della linea 80 da' istruzione al computer per rendere pieni i primi, i secondi..., gli ultimi 8 pixel dell'angolo basso destro dell'animazione. Il funzionamento puo' essere osservato meglio digitando:

```
POKE 833,0 (per rimetterlo a posto POKE 833,255 o lanciare (RUN) il
            programma).
```

Come si puo' vedere, il secondo gruppo di 8 pixel e' stato cancellato; allo stesso modo, inserendo nel programma la seguente linea:

```
99 FORA=836TO891STEP3:POKEA,0:NEXT A
```

si cancellano i blocchi al centro dell'animazione. Va ricordato che i pixel di cui e' formata una figura sono raggruppati in blocchi di 8; la precendete linea cancella il quinto gruppo di 8 pixel (blocco 836) ed ogni terzo blocco fino al blocco 890 Introducendo tramite POKE uno qualunque degli altri numeri da 832 a 894, si puo' visualizzare ogni blocco da pieno (255) a vuoto (0).

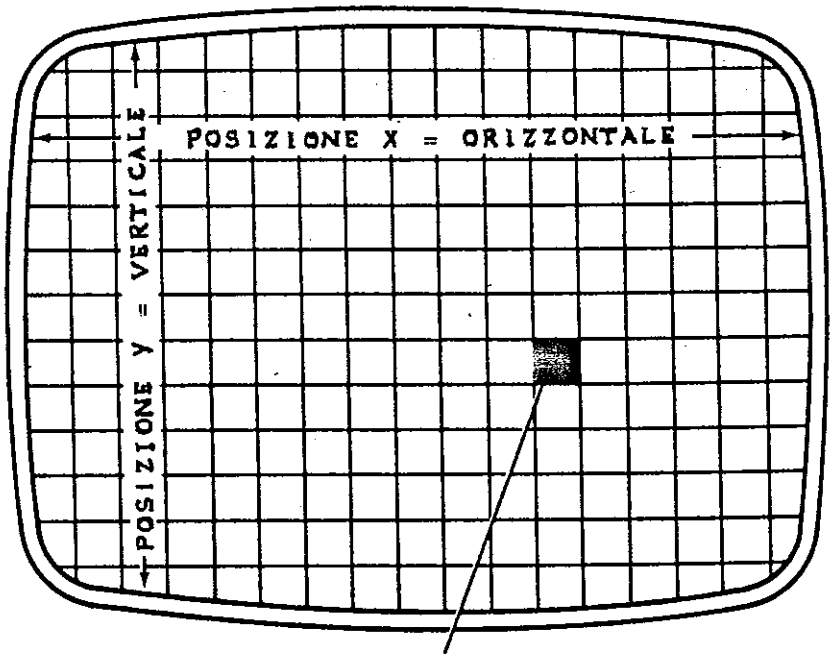
COMPATTAZIONE DEI PROGRAMMI DI ANIMAZIONE

Qella illustrata di seguito e' un'utile tecnica di compattazione. Il programma illustrato in precedenza puo' essere ulteriormente abbreviato compattandolo opportunamente come illustrato di seguito:

```
10 PRINTCHR$(147):V=53248:POKEV+21,1:POKE2040,13:POKEV+39,1

20 FORS=832TO894:POKES,255:NEXT:POKEV,24:POKEV+1,100
```

Altri modi di compattazione possono essere osservati nel paragrafo GUIDA ALLA COMPATTAZIONE.



Per essere visualizzata, un'animazione locata in questo punto deve avere impostata sia la posizione X (orizzontale) che la posizione Y (verticale)

Figura 3.4 - Divisione dello schermo video in una griglia di coordinate (X,Y)

POSIZIONAMENTO DELLE ANIMAZIONI SULLO SCHERMO

Tutto lo schermo video e' diviso in una griglia di coordinate X e Y, come una carta millimetrata. La COORDINATA X e' la posizione ORIZZONTALE sullo schermo, la Y quella VERTICALE (vd. figura 3.4).

Per posizionare qualunque animazione sullo schermo, bisogna usare POKE per due impostazioni - la posizione X e la posizione Y - che dicano al computer dove visualizzare L'ANGOLO IN ALTO A SINISTRA dell'animazione. Va ricordato che un'animazione e' formata da 504 punti, 24 orizzontali X 21 verticali; percio', quando si posiziona un'animazione nell'anglo in alto a sinistra dello schermo, l'animazione viene visualizzata come un'immagine grafica di 24 pixel ORIZZONTALI X 21 VERTICALI, iniziando dalla posizione (X,Y) precedentemente definita. L'animazione viene visualizzata partendo dall'angolo in alto a sinistra dell'intera animazione, anche se l'animazione viene definita usando solo una parte dell'area animazione di 24 X 21 pixel.

La seguente figura 3.5 illustra il funzionamento delle posizioni X e Y sullo schermo; l'area grigia indica il campo visibile sullo schermo, quella bianca la zona FUORI da tale campo.

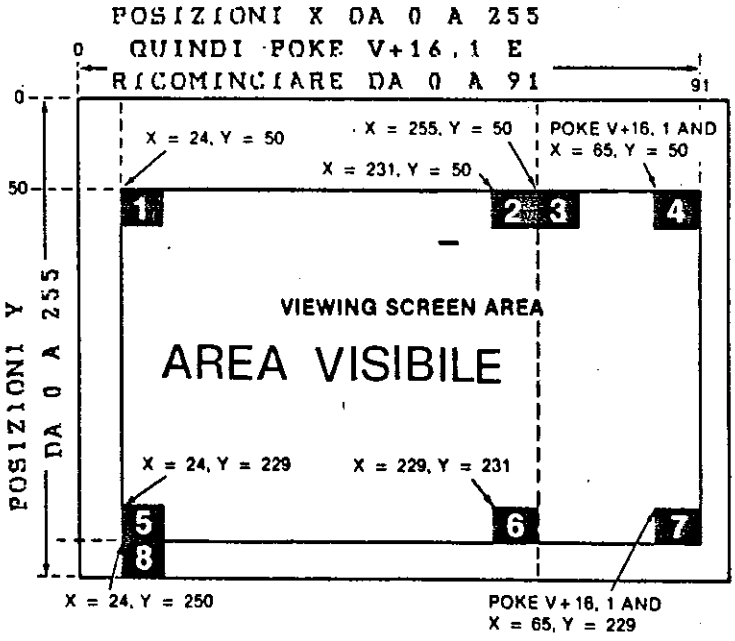


Figura 3.5 - Determinazione delle posizioni X-Y di un'animazione

Per visualizzare un'animazione in una data locazione, occorre impostare con una POKE i valori di X e Y per ogni animazione, tenendo presente che ogni animazione ha la propria POKE X e POKE Y. I valori di X e Y di tutte e 8 le animazioni sono i seguenti:

INSERIRE CON UNA POKE QUESTI VALORI PER DETERMINARE LE POSIZIONI X-Y DELL'ANIMAZIONE

	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7
X	V,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
X di DESTRA	V+16,1	V+16,2	V+16,4	V+16,8	V+16,16	V+16,32	V+16,64	V+16128

IMPOSTAZIONE DELLA POSIZIONE X - Questi valori sono compresi fra 0 e 255, partendo da sinistra. I valori da 0 a 23 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE FINO ALLA 255-esima POSIZIONE (per le posizioni oltre la 256-esima, si veda il prossimo paragrafo). Per collocare l'animazione in una di queste posizioni, usare POKE con la corrispondente POSIZIONE X; ad esempio, POKE V+2,24 posiziona l'animazione 1 nella posizione X piu' a sinistra NEL CAMPO VISIBILE.

VALORI DI X OLTRE LA 255-ESIMA POSIZIONE - Per posizionare un'animazione oltre la 255-esima posizione, e' necessaria una seconda POKE che usa i valori illustrati nella riga "X DI DESTRA" della tabella 3.5. Di solito, la numerazione orizzontale continua oltre 255, ma poiche' i registri contengono solamente 8 bit, bisogna usare un secondo registro per accedere alla PARTE DESTRA dello schermo, ricominciando la numerazione da 0. Quindi, per oltrepassare la posizione 255, bisogna impostare POKE V+16, seguito dal numero dell'animazione. Si guadagnano cosi' altre 65 posizioni orizzontali

(rinumerate da 0 a 65) nella zona visibile a DESTRA dello schermo (impostando il valore delle X sulla destra a 255, si esce dal bordo destro dello schermo visibile)

IMPOSTAZIONE DELLA POSIZIONE Y - Questi valori sono compresi fra 0 e 255. a partire dall'alto. I valori compresi fra 0 e 49 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE NELLA PARTE ALTA dello schermo; quelli compresi fra 50 e 229 la collocano nel CAMPO VISIBILE; quelli compresi fra 230 e 255 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE NELLA PARTE BASSA dello schermo.
Consideriamo il seguente programma:

```
10 PRINT"[SHIFT CLR/HOME]": V=53248:POKEV+21,2:POKE2041,13
15 FORS=832TO895:POKES,255:NEXT
20 POKEV+40,7
30 POKEV+2,24
40 POKEV+3,50
```

Questo semplice esempio illustra quali valori assegnare alle coordinate X e Y dell'animazione 1, un cubo pieno, nel caso che si voglia collocare tale animazione nell'angolo in alto a sinistra dello schermo. Se si modifica la linea 40 nel modo seguente:

```
40 POKE V+3,229
```

L'animazione viene visualizzata nell'angolo in basso a sinistra. Modificando invece la linea 30 in:

```
30 POKE V+2,255
```

l'animazione viene spostata sul LIMITE DESTRO DI X (255), per cui il bit piu' significativo del registro 16 deve essere IMPOSTATO. In altre parole, per RIPOSIZIONARE il contatore della posizione X al valore 256 (256-esimo pixel) dello schermo, occorre impostare POKE V+16 seguito dal numero mostrato nella colonna X di DESTRA della precedente tabella delle posizioni di X e Y. Quindi la linea 30 diventa:

```
30 POKE V+16,PEEK(V+16)OR 2:POKE V+2,0
```

POKE V+16,2 imposta il bit piu' significativo della posizione X dell'animazione 1, riposizionando a 256 (256-esimo pixel dello schermo). POKE V+2,0 visualizza l'animazione nella NUOVA POSIZIONE ZERO, che attualmente e' il 256-esimo pixel.
Per riposizionarsi sulla sinistra dello schermo, occorre impostare daccapo a 0 il bit piu' significativo del contatore della posizione X scrivendo:

```
POKE V+16, PEEK(V+16)AND 253
```

RIASSUMENDO: la POSIZIONE X di ogni animazione assume tutti i valori compresi fra 0 e 255. Per accedere alle posizioni dello schermo oltre a 255 (255-esimo pixel), si deve usare una POKE V+16 aggiuntiva, che imposti il bit piu' significativo della posizione X e ricominci a contare da 0 al 256-esimo pixel in poi (ad esempio, POKE V+16, PEEK(V+16)OR1 e POKE V,1 devono essere usate per posizionare l'animazione 0 al 257-esimo pixel dello schermo). Per tornare nella

posizione X di sinistra, bisogna DISATTIVARE l'impostazione del controllo digitando POKE V+16,PEEK(V+16)AND 254.

POSIZIONAMENTO SULLO SCHERMO DI PIÙ ANIMAZIONI

Il seguente programma definisce TRE DIFFERENTI ANIMAZIONI (0,1 e 2) di colore diverso, posizionandole in tre diversi punti dello schermo:

```
10 PRINT"□" :V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKE+41,8
50 POKEV,24:POKEV+1,50
60 POKEV+2,12:POKEV+3,229
70 POKEV+4,255:POKEV+5,50
```

Per comodita', le tre animazioni sono state definite come quadrati pieni che traggono i dati dalla stessa fonte: l'aspetto che qui interessa e' il loro posizionamento sullo schermo. L'animazione 0 (bianca) si trova nell'angolo in alto a sinistra; l'animazione 1 (gialla) in basso a sinistra PER META' FUORI SCHERMO (ricordiamo ancora una volta che 24 e' la posizione piu' a sinistra del campo visibile, per cui una posizione di X minore di 24 pone una parte o tutta l'animazione fuori dallo schermo; per questo si e' usata per X la posizione 12, che visualizza meta' animazione lasciando l'altra meta' fuori dallo schermo); infine, l'animazione 2 (arancione) si trova nella POSIZIONE LIMITE DI DESTRA (posizione 255). Che cosa bisognerebbe fare se si volesse visualizzare un'animazione nella zona di DESTRA DELLA POSIZIONE 255 di X? Lo saprete...al prossimo paragrafo!

VISUALIZZAZIONE DI UN'ANIMAZIONE OLTRE LA 255-ESIMA POSIZIONE DI X

Questa visualizzazione richiede una particolare POKE che imposti il bit piu' significativo della posizione X ed inizi alla 256-esima posizione (256-esimo pixel).

Innanzi tutto, occorre impostare POKE V+16 seguito dal numero dell'animazione che si vuole usare (supponiamo di usare l'animazione 0, i cui valori di X e Y devono essere rintracciati nella TAVOLA DELLE POSIZIONI X e Y). All'assegnazione della posizione X, occorre tenere presente che il contatore di X varia da 0 a 256; la linea 50 diventa quindi:

```
50 POKE V+16,1:POKE V,24:POKE V+1,75
```

La POKE V+16 di questa linea consente di "aprire" la parte detstra dello schermo del numero di posizioni richieste. Ora la nuova posizione 24 di X inizia, per l'animazione 0, a DESTRA dello schermo; modificare la linea 60 come segue:

```
60 POKE V+16,1:POKE V,65:POKE V+1,75
```

Alcune prove fatte con i valori riportati nella tabella delle posizioni X e Y forniscono i valori necessari per posizionare e muovere le animazioni da destra a sinistra. Il posizionamento delle animazioni viene ulteriormente illustrato nel paragrafo riguardante il movimento delle animazioni.

PRIORITÀ DELLE ANIMAZIONI

Precedentemente si e' esposto il principio in base al quale le animazioni sembrano muoversi DAVANTI o DIETRO l'una rispetto all'altra. Quest'illusione tridimensionale si ottiene per mezzo delle PRIORITA' DELLE ANIMAZIONI del sistema, che determinano quali animazioni hanno priorita' maggiore rispetto alle altre nel caso in cui due o piu' animazioni SI SOVRAPPONGANO sullo schermo.

La regola fondamentale e' "primo entrato, primo servito" (FIFO=First In First Out), cioe' le animazioni di bassa numerazione hanno AUTOMATICAMENTE priorita' maggiore di quelle di alta numerazione. Se, ad esempio, si visualizzano le animazioni 0 e 1 in modo che si sovrappongano, l'animazione 0 appare DAVANTI alla 1. Normalmente, l'animazione 0 ha la priorita' PIU' ALTA, essendo 0 il numero piu' basso a disposizione delle animazioni. Analogamente, l'animazione 1 ha la priorita' maggiore delle 2-7, la 2 delle 3-7, ecc.; infine, l'animazione 7 ha la priorita' PIU' BASSA DI TUTTE, ed appare sempre "DIETRO" ALLE ALTRE ANIMAZIONI CHE LE SI SOVRAPPONGONO.

Per poter vedere come funzionano le priorita', cambiare le linee 50, 60 e 70 del precedente programma nel modo seguente:

```
10 PRINT"[SHIFT CLR HOME]" :V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50:POKEV+16,0
60 POKEV+2,34:POKEV+3,60
70 POKEV+4,44:POKEV5,70
```

Dovrebbe comparire un'animazione bianca sopra un'animazione gialla sopra un'animazione arancione. Ovviamente, si puo' trarre vantaggio dalle priorita' per MUOVERE LE ANIMAZIONI e migliorare i programmi che le riguardano.

COME DISEGNARE UN'ANIMAZIONE

Disegnare un'animazione COMMODORE e' come colorare degli spazi vuoti di un libro da disegni. Ogni animazione e' formata da un insieme di punti chiamati pixel; il disegno di un'animazione viene quindi ricondotto a "colorare" alcuni pixel.

La griglia della figura 3.6 e' simile ad un'animazione vuota.

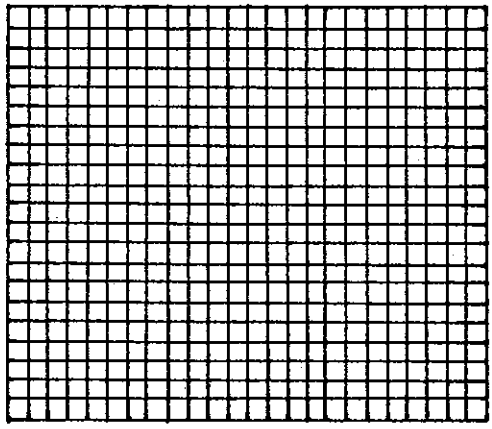


Figura 3.6 - Griglia per la costruzione di animazioni

Ogni quadratino rappresenta un pixel dell'animazione; la griglia e' formata da 24 pixel X 21, pari a 504 pixel dell'animazione completa. Per costruire un'animazione che assomigli a qualcosa, occorre colorare questi pixel usando uno speciale PROGRAMMA...m come si riesce a controllare piu' di 500 punti indipendenti? Ecco venire in aiuto la programmazione, che consente di trattare, per ogni animazione, 63 numeri invece dei 504 richiesti.

CREAZIONE DI UN'ANIMAZIONE...MINUTO PER MINUTO

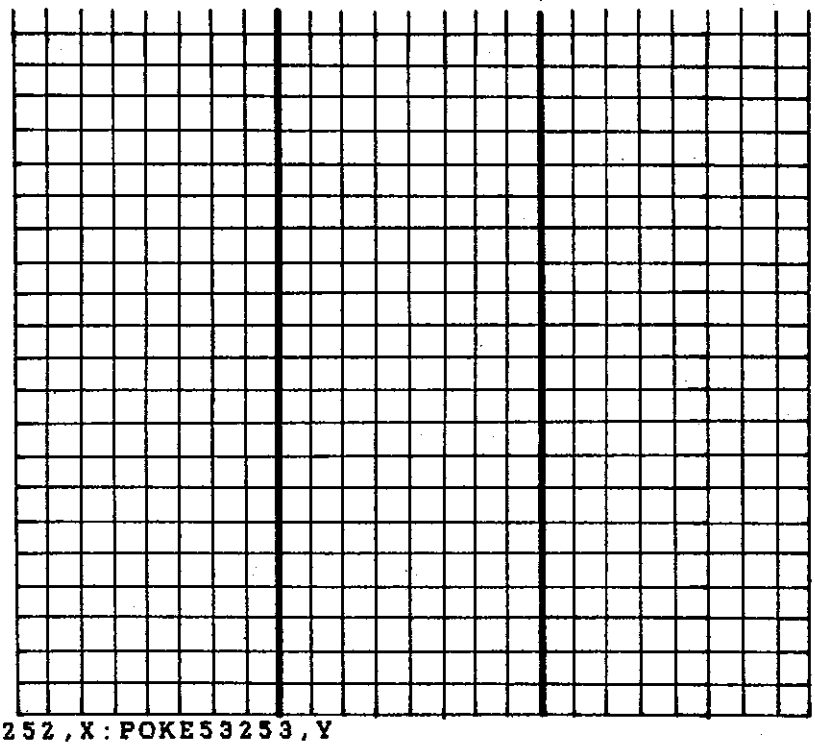
La creazione di un'animazione puo' essere riassunta nei seguenti passi:

PASSO 1:

Scrivere SU UN FOGLIO DI CARTA il programma di creazione di un'animazione illustrato sotto. Da notare la linea 100, che da' il via ad una speciale sezione di istruzioni DATA contenenti i 63 numeri necessari a creare un'animazione.

```
10 PRINT "[SHIFT CLR HOME]":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT

100 DATA255,255,255
101 DATA128,0,1
102 DATA128,0,1
103 DATA128,0,1
104 DATA144,0,1
105 DATA144,0,1
106 DATA144,0,1
107 DATA144,0,0
108 DATA144,0,1
109 DATA144,0,1
110 DATA144,0,1
111 DATA144,0,1
112 DATA144,0,1
113 DATA144,0,1
114 DATA144,0,1
115 DATA128,0,1
116 DATA128,0,1
117 DATA128,0,1
118 DATA128,0,1
119 DATA128,0,1
120 DATA255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

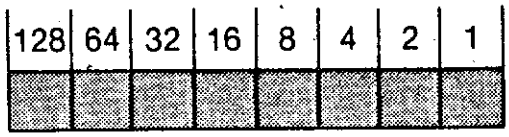


PASSO 2:

Colorare i pixel (i quadratini) della griglia della figura 3.6 (usare in alternativa un foglio di carta millimetrata, ricordando sempre che un'animazione e' formata da 24 pixel orizzontali X 21 verticali). si puo' disegnare qualunque immagine si desideri, ma per il nostro esempio disegneremo una scatola.

PASSO 3:

Consideriamo i primi 8 pixel: ogni colonna di pixel ha un numero, corrispondente ad una potenza del 2, da 128 (=2 7) a 1 (=2 0). La particolare addizione che stiamo per fare e' un tipo di ARITMETICA BINARIA molto usata dai calcolatori come modo particolare di conteggio. Gli 8 pixel considerati sono visualizzati in dettaglio qui di seguito.



PASSO 4:

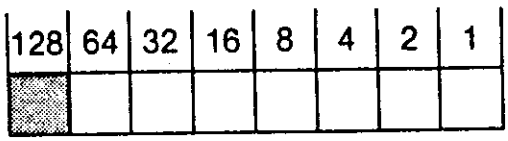
Sommare i numeri dei pixel PIENI. Poiche' il primo gruppo di pixel e' pieno, il loro totale e' 255

PASSO 5:

Introdurre tale totale come PRIMO ELEMENTO DELL'ISTRUZIONE DATA della linea 100 del programma di costruzione di un'animazione dato piu' sotto.

PASSO 6:

Consideriamo I PRIMI 8 PIXEL DELLA SECONDA RIGA dell'animazione. Sommare anche in questo caso i valori dei pixel pieni. Poiche' solo uno di questi 8 pixel e' pieno, il totale e' 128. Introdurre questo totale come primo elemento dell'istruzione DATA nella linea 101.



PASSO 7:

Sommare i valori del gruppo successivo di 8 pixel (0 in quanto sono tutti VUOTI) e posizionarli in ordine nella linea 101. Passare quindi al gruppo seguente di pixel e ripetere l'operazione per TUTTI I GRUPPI DI 8 PIXEL (3 gruppi per ogni riga, per 21 righe). Alla fine si ha un totale di 63 numeri, ognuno dei quali rappresenta un gruppo di 8 pixel, che moltiplicato per 63 da' un totale di 504 punti indipendenti. Il programma puo' essere visto anche nel modo seguente. Ogni riga del programma rappresenta UNA RIGA dell'animazione. Un gruppo si tre numeri di ogni riga rappresenta UN GRUPPO DI 8 PIXEL. Ogni numero comunica al computer quali pixel riempire e quali lasciare vuoti.

PASSO 8:

COMPATTARE IL PROGRAMMA IN MENO SPAZIO RAGGRUPPANDO TUTTE LE ISTRUZIONI DATA, COME ILLUSTRATO NEL PROGRAMMA SEGUENTE. Appare ora chiaro il motivo per cui si e' consigliato di scrivere il programma dell'animazione su un foglio di carta: LE ISTRUZIONI DATA DELLE LINEE

DA 100 A 120 del programma di cui al PASSO 1 sono state scritte in quel modo solamente per aiutare a visualizzare quali numeri mettere in relazione ai gruppi di pixel dell'animazione. Il programma finale puo' dunque essere "compattato" come segue:

```
10 PRINT" ":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255,128,0,1,128,0,1,128,0,1,144,0,1,144,0,1,144,0,
1,144,0,1
101 DATA144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,128,0,1
128,0,1
102 DATA128,0,1,128,0,1,128,0,1,128,0,1,255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

MOVIMENTO DELLE ANIMAZIONI SULLO SCHERMO

Per muovere lentamente l'animazione sullo schermo, aggiungere al programma precedente le seguenti due linee:

```
50 POKEV+5,100:FORX=24TO255:POKE V+4,X:NEXT:POKEV+16,4
55 FORX=0TO65:POKEV+4,X:NEXTX:POKEV+16,0:GOTO50
```

LINEA 50 Imposta la posizione Y a 100(provare, tanto per cambiare, i valori 50 e 229). Il successivo LOOP FOR...NEXT muove l'animazione dalla posizione 0 alla 255, rispettivamente. Quando quest'ultima viene raggiunta,viene impostata la posizione X di destra (POKE V+16,2) necessaria per attraversare la parte destra dello schermo.

LINEA 55 Contiene un loop For...Next che muove l'animazione nelle successive 65 righe dello schermo.Da notare che il valore di X e' stato rimesso a zero, ma poiche' si e' impostato l'X DI DESTRA (POKE V+16,2), X parte dalla destra dello schermo.

Queste due linee sono in ciclo infinito (GOTO 50); se si desidera che l'animazione attraversi una sola volta lo schermo e scompaia, togliere GOTO 50.

Le seguenti istruzioni muovono l'animazione AVANTI e INDIETRO:

```
50 POKE V+5,100:FOR X=24TO255:POKE V++4,X:NEXT:POKEV+16,4
51 FOR X=0TO65: POKE V+4,X:NEXT X
55 FOR X=65TO0 STEP-1:POKE V+4,X:NEXT:POKE V+16,0
56 FOR X=255TO24 STEP-1:POKE V+4,X:NEXT
60 GOTO 50
```

Questa versione del programma e' analoga alla precedente, solo che quando l'animazione raggiunge il bordo destro dello schermo, GIRA SU SE' STESSA e torna indietro. Cio'e' ottenuto per mezzo di STEP-1, che dice al programma di posizionare (POKE) l'animazione nei valori di destra compresi fra 65 e 0, poi in quelli di sinistra da 255 a 0 tornando indietro di una posizione alla volta.

SCROLLING VERTICALE

Questo tipo di movimento dell'animazione e' chiamato scrolling; per la sua realizzazione su e giu' alla posizione Y, basta UNA SOLA LINEA. CANCELLARE LE PRECEDENTI LINEE 50 e 55 battendo i rispettivi numeri di linea seguiti da RETURN:

50 (RETURN)
55 (RETURN)

e quindi impostare la LINEA 50 come segue:

```
50 POKE V+4,24:FOR Y=0TO255:POKE V+5,Y:NEXT
```

IL TOPOLINO BALLERINO – UN ESEMPIO DI PROGRAMMA DI ANIMAZIONE

Le tecniche descritte in una guida per programmatori sono talvolta di difficile comprensione, ragion per cui presentiamo qui di seguito un divertente programma di animazione chiamato "TOPOLINO BALLERINO". Questo programma usa tre differenti animazioni in rapido movimento unite ad effetti sonori. Quella seguente e' la descrizione di OGNI ISTRUZIONE, che riteniamo possa aiutare a comprendere la costruzione ed il funzionamento del programma.

```
5 S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,15:POKES+6,215
10 POKES+7,120:POKES+8,100:POKES+12,15:POKES+13,215
15 PRINT"□" V=53248:POKEV+21,1          (SHIFT CLR-HOME)
20 FORS1=12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2=12352TO12414:READQ2:POKES2,Q2:NEXT
30 FORS3=12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV+39,15:POKEV+1,68
40 PRINTTAB(160) "(CTRL 2) I AM THE DANCING MOUSE!(C= 7)"
45 P=192
50 FORX=0TO347STEP3
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO60:NEXT
85 P=P+1:IFP>194THENP=192
90 NEXT
95 END
100 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
101 DATA63,255,251,31,187,248,3,187,192,1,255,128,3,189,192,1,231,
    128,1,255,0
102 DATA31,255,0,0,124,1,1,254,0,1,199,32,3,131,224,7,1,192,1,192,0,
    3,192,0
103 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
104 DATA63,255,252,31,221,248,3,221,192,1,255,128,3,255,192,1,195,
    128,1,231,3
105 DATA31,255,255,0,124,0,0,254,0,1,199,0,7,1,128,7,0,204,1,128,
    124,7,128,56
106 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
107 DATA63,255,252,31,221,248,3,221,192,1,255,134,3,189,204,1,199,
    152,1,255,48
108 DATA1,255,224,1,252,0,3,254,0
109 DATA7,14,0,204,14,0,248,56,0,112,112,0,0,60,0,-1
200 POKES+4,12,:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

LINEA 5:

`S=54272` Uguaglia la variabile S a 54272, che e' la locazione di memoria di partenza del CIRCUITO SONORO. In seguito, anziche' impostare direttamente una locazione di memoria, usare una POKE S piu' un certo valore.

`POKES+24,15` Imposta il volume al massimo.

`POKES,220` Imposta una Bassa Frequenza nella Voce 1 per una nota che approssima un DO acuto in sesta ottava.

`POKES+1,68` Imposta un'Alta Frequenza nella Voce 1 per una nota che approssima un DO acuto in sesta ottava

`POKES+5,15` Imposta ATTACCARE/DECADERE per la VOCE 1; consiste in questo caso nel massimo livello di DECADERE senza alcun ATTACCARE, producendo cosi' un effetto "ECO".

`POKES+6,215` Imposta SUSTENERE/RILASCIARE per la Voce 1 (215 e' una combinazione di questi due valori).

LINEA 10:

`POKES+7,120` Imposta l'Alta Frequenza per la Voce 2

`POKES+8,100` Imposta la Bassa Frequenza per la Voce 2

`POKE S+12,15` Imposta ATTACK/DECAY per la Voce 2 allo stesso livello della precedente Voce 1

`POKES+13,215` Imposta ATTACCARE/DECADERE per la Voce 2 allo stesso livello della Voce 1

LINEA 15:

`PRINT"` SHIFT CLR/HOME `"` Azzera lo schermo all'inizio del programma.

`V=53248` Definisce la variabile "V" come la locazione di partenza del circuito VIC-II che controlla le animazioni; in seguito, le locazioni dell'animazione saranno definite come V + un certo valore.

`POKEV+21,1` Abilita l'animazione 1

`FORS1=12288 TO12350` Attualmete si usa una sola animazione (la #0), ma che fa capo a tre insiemi di dati dell'animazione necessari alla formazione di tre diverse figure. Per avere l'animazione, occorre assegnare i puntatori dell'animazione 0 a tre diverse locazioni di memoria in cui si sono registrati i dati che definiscono le tre differenti figure. La stessa animazione viene via via ridefinita rapidamente come tre differenti figure per riprodurre l'animazione del Topolino Ballerino. Nelle istruzioni DATa, si possono definire quante figure si

desiderano e ruotarle attorno ad una o piu' animazioni; si puo' vedere, quindi, che non occorre limitare un'animazione ad una figura, o viceversa. Un'animazione puo'assumere molte figure differenti, semplicemente cambiando il puntatore che imposta quell'animazione nelle diverse locazioni di memoria dove sono registrati i dati dell'animazione relativi alle diverse figure. Questa linea significa che i dati per la figura 1 dell'animazione sono stati collocati nelle locazioni da 12288 a 12350.

READ Q1 — Legge nell'ordine 63 numeri dalle istruzioni DATA che iniziano alla linea 100. Q1 e' un nome di variabile arbitrario.

POKE S1,Q1 — Posiziona il primo numero rilevato dalle istruzioni DATA (il primo, "Q1", e' 30) nella prima locazione di memoria che e' la 12288. Questa istruzione e' equivalente a POKE12288,30.

NEXT — Comunica al computer di considerare solo le parti comprese tra FOR e NEXT e di eseguire le istruzioni che incontra. In altre parole, l'istruzione NEXT fa leggere (READ) al computer il prossimo (NEXT) Q1 dalle istruzioni DATA. Takle Q1 e' 0, il che fa incrementare S1 di 1 unita' verso il valore successivo, che e'12289. L'istruzione NEXT fa eseguire il ciclo fino all'ultimo valore della serie, dato da POKE 12350,0.

LINEA 25:

FORS2=12352 — La seconda figura dell'animazione 0 e' definita dai dati da 12352 a 12414. DA NOTARE che si e' saltata la locazione 12351: questa infatti, e' la 64-esima locazione usata nella definizione del primo gruppo dell'animazione. Occorre ricordare, quando si definiscono animazioni in locazioni consecutive, che si usano 64 locazioni, ma i dati delle animazioni si trovano solamente nelle prime 63.

READ Q2 — Legge i 63 numeri che seguono quelli usati per la prima figura dell'animazione. Questa READ cerca semplicemente il numero immediatamente successivo nella zona occupata da DATA e comincia a leggere 63 numeri, uno alla volta.

POKES2,A2 — Posiziona i dati (Q2) nelle locazioni di memoria (S2) della seconda figura dell'animazione, che inizia alla locazione 12352.

NEXT — Come la precedente linea 20

LINEA 30:

FORS3=12416 TO12478	La terza figura dell'animazione 0 e' definita dalle istruzioni DATA locate da 12416 a 12478.
READQ3	Legge in Q3 gli ultimi 63 numeri in ordine.
POKE S3,A3	Posiziona tali numeri nelle locazioni da 12416 a 12478.
NEXT	Analogo alle linee 20 e 25.

LINEA 35:

POKE V+39,15	Imposta il colore grigio chiaro per l'animazione 0.
POKE V+1,68	Imposta l'angolo in alto a destra del quadrato dell'animazione nella posizione verticale (Y) 68. Analogamente la posizione 50 e' la posizione Y dell'angolo in alto a sinistra dello schermo.

LINEA 40:

PRINT TAB(160)	Tabula 160 spazi dello spazio carattere in alto a sinistra, cioe' si posiziona 4 righe piu' in basso della istruzione di azzeramento dello schermo;in altre parole, il messaggio da visualizzare inizia alla sesta riga dello schermo.
" CTRL WHT	Premendo contemporaneamente questi due tasti dopo le virgolette, viene visualizzata una E "reverse": il colore di qualunque cosa visualizzata da ora in poi viene impostato a bianco.
J AM THE DANCING MOUSE!	Messaggio visualizzato dalla PRINT
C= 7"	Ripristina il colore blu dopo l'istruzione PRINT. Premendo questi due tasti dopo le virgolette si visualizza un asterisco (*) "reverse".

LINEA 45:

P=192	Uguaglia la variabile P a 192. Questo numero e' il puntatore da usare per "puntare", in questo caso, l'animazione 0 alla locazione di memoria che comincia a 12288. Il segreto per usare un'animazione allo scopo di creare un effetto di movimento composto da tre diverse figure sta nello spostare questo puntatore alle locazioni delle altre due figure dell'animazione.

LINEA 50:

FORX=0TO347 STEP3	Incrementa il movimento dell'animazione di 3X posizioni alla volta, dalla posizione 0 alla 347.

LINEA 55:

`RX = INT(X/256)` Parte intera di X/256; indica che RX assume il valore 0 per X < 256, a quando X = 256. Si usa RX al momento di attivare la PARTE DESTRA dello schermo, impostando POKE V+16 a 0 o a 1

`LX=X-RX*256` Quando l'animazione e' nella posizione 0 di X, questa formula diviene LX=0-0*256=0; quando invece si trova nella posizione 1 di X, la formula diviene LX=1-0*256=1; quando infine si trova nella posizione 256 di X, si ottiene LX=256-1*256=0, mediante il quale si imposta X di nuovo a 0, operazione effettuata quando ci si muove sulla DESTRA dello schermo (POKE V+16,1).

LINEA 60:

`POKEV,LX` Imposta POKE V, senza alcun inremento, insieme ad un altro valore, quando si desidera impostare sullo schermo la POSIZIONE ORIZZONTALE (X) dell'animazione (vd. tabella per la creazione di un'animazione). Come mostrato in precedenza, il valore della posizione orizzontale LX dell'animazione varia da 0 a 255; quando quest'ultima raggiunge questo valore, viene automaticamente riportata a 0 dall'espressione di LX impostata alla linea 55

`POKE V+16,RX` Questa istruzione abilita sempre la DESTRA dello schermo oltre la posizione 256, e riporta a zero le coordinate del posizionamento orizzontale. In base alla posizione dell'animazione, determinata dalla espressione di RX alla LINEA 55, RX puo' essere 0 oppure 1.

LINEA 70:

`IFP=192THEN GOSUB200` Se il puntatore all'animazione e' impostato a 192 (prima figura dell'animazione), il controllo della forma d'onda per il primo effetto sonoro viene impostato, alla linea 200, a 129 e 128

LINEA 75:

`IFP=193THEN GOSUB300` Se il puntatore all'animazione e' impostato a 193 (seconda figura dell'animazione), il controllo della forma d'onda per il secondo effetto sonoro (Voce 2) viene impostato, alla linea 300, a 129 e 128

LINEA 80:

`POKE2040,P` Imposta il puntatore dell' animazione alla locazione 192 (come si ricordera', P e' stato posto uguale a 192 alla linea 45).

`FORT=1TO60: NEXT` Loop di ritardo per impostare il ritmo della danza del topolino (tale ritmo puo' essere variato aumentando o diminuendo il numero 60).

LINEA 85:

P=P+1 Aggiunge 1 al valore iniziale del puntatore.

IFP>194 THENP=192 Punta l'animazione a 3 sole locazioni di memoria. Il valore 192 punta alle locazioni da 12288 a 12350; 193 punta alle locazioni da 12416 a 12478. Quesra linea imposta daccapo P a 192 non appena P diventa 195, impedendo cosi' a P stesso di assumere quest'ultimo valore. In questo modo il Puntatore spazia consecutivamente le tre figure dell'animazione all'interno dei gruppi di 64 byte delle locazioni di memoria contenenti i dati.

LINEA 90:

NEXT X Solamente dopo che l'animazione ha assunto una delle tre figure definite dalle istruzioni DATA, l'animazione puo' muoversi sullo schermo. A questo punto l'animazione si muove di 3X posizioni alla volta (anziche' spostarsi lentamente di una, come pure e' possibile). Saltando 3 posizioni alla volta (istruzione STEP), il Topilino "danza"
piu' velocemente. NEXT X chiude il LOOP, aperto alla linea 50, che determina la posizione X.

LINEA 95:

END Chiude il programma, quando l'animazione esce dallo schermo.

LINEA 100-109:

DATA Le figure dell'animazione vengono lette in ordine dai numeri delle istruzioni DATA. Prima i 63 numeri che compongono la Figura 1, poi i 63 della Figura 2, infine i 63 della figura 2, quindi i 63 della figura 3. Questi dati vengono letti permanentemente nelle 3 locazioni di memoria, dopodiche' il programma punta l'animazione 0 alle 3 locazioni di memoria e l'animazione assume automaticamente la figura rappresentata dai dati presenti in quelle locazioni. Puntando l'animazione ad una figura alla volta si ottiene l'effetto "movimento". Per osservare come questi numeri influenzano ogni animazione, basta sostituire i primi 3 numeri della linea 100 con 255,255,255. Si veda anche il paragrafo sulla definizione delle figure di un'animazione.

LINEA 200:

POKES+4,129 Attiva l'effetto sonoro.

POKES+4,128 Disattiva l'effetto sonoro.

RETURN Riporta il programma alla fine della LINEA 70 dopo aver cambiato impostazione del controllo della forma d'onda; l'elaborazione riprende dalla fine della LINEA 70.

LINEA 300:

POKES+11,129 Attiva l'effetto sonoro.

POKES+11,128 Disattiva l'effetto sonoro.

RETURN Riporta l'elaborazione alla fine della LINEA 75.

PRONTUARIO PER LA COSTRUZIONE DI UN'ANIMAZIONE

	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7
Attiva una animazione	V+21,2	V+21,1	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128
Caricamento in memoria (imposta i puntatori)	2040, 192	2041, 193	2042, 194	2043, 195	2044, 196	2045, 197	2046, 198	2047, 199
Locazione dei pixel (da 12288 a 12798)	12288 to 12350	12352 to 12414	12416 to 12478	12480 to 12542	12544 to 12606	12608 to 12670	12672 to 12734	12736 to 12798
Colore animazione	V+39,C	V+40,C	V+41,C	V+42,C	V+43,C	V+44,C	V+45,C	V+46,C
Posizione X di sinistra (0-255)	V+0,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Posizione X di destra (0-255)	V+16,1 V+0,X	V+16,2 V+2,X	V+16,4 V+4,X	V+16,8 V+6,X	V+16,16 V+8,X	V+16,32 V+10,X	V+16,64 V+12,X	V+16,128 V+14,X
Posizione Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
Espansione orizzontale	V+29,1	V+29,2	V+29,4	V+29,8	V+29,16	V+29,32	V+29,64	V+29,128
Espansione verticale	V+23,1	V+23,2	V+23,4	V+23,8	V+23,16	V+23,32	V+23,64	V+23,128
Attivazione modo multicolore	V+28,1	V+28,2	V+28,4	V+28,8	V+28,16	V+28,32	V+28,64	V+28,128
Multicolore 1 (primo colore)	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C
Multicolore 2 (secondo colore)	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C
Priorità animazioni	Animazioni di numero piu' basso hanno priorita' di schermo maggiore delle animazioni di numero piu' alto. Ad esempio, l'animazione 0 ha priorita' su TUTTE le altre animazioni, e l'animazione 7 ha la priorita' piu' bassa. Cio' significa che le animazioni di numero piu' basso sembrano sempre muoversi DAVANTI o SOPRA le animazioni di numero piu' alto							
Contatto (animazione-animazione)	V+30 IF PEEK(V+30)ANDX=X THEN [azione]							
Contatto (animazione-fondo)	V+31 IF PEEK(V+31)ANDX=X THEN [azione]							

NOTE PER LA COSTRUZIONE DI UN'ANIMAZIONE

LOCAZIONI DI MEMORIA E PUNTATORI ALLA MEMORIA ANIMAZIONE ALTERNATIVI PER L'USO DEL BUFFER DEL REGISTRATORE

Memorizzazione (imposta i puntatori)	ANIM. 0 2040,13	ANIM. 1 2041,14	ANIM. 2 2045,15	Se si pensa di usare da 1 a 3 animazioni, per il buffer del registratore si possono usare le locazioni 832-1023, ma per piu' di 3 animazioni si consigliano le locazioni 12288-12798 (vd. tabella).
Pixel animaz. Locazioni per Blocchi 13-15	832 fino a 894	896 fino a 958	960 fino a 1022	

ATTIVAZIONE DI UN'ANIMAZIONE

Si puo' attivare una singola animazione usando una POKE V+21 e il numero estratto dalla tabella, ma l'attivazionre di UNA SOLA animazione DISATTIVA le altre. Per attivare DUE o PIU' animazioni, SOMMARE i numeri delle animazioni che si vuole attivare (es. POKE V+21,6 attiva le animazioni 1 e 2). Il seguente esmpio indica come attivare e disattivare un'animazione senza influenzare le altre.

ESEMPIO:

Per disattivare la sola animazione 0: POKEV+21,PEEKV+21AND(255-1). Cambiare il numero 1 di (255-1) in 1, 2, 4, 8, 16, 32, 64, 128 per le animazioni da 0 a 7. Per riattivare l'animazione senza influenzare le altre gia' attivate, usare POKEV+21,PEEK(V+21)OR1 cambiando OR1 in OR2 (ANIMAZIONE 2),OR3 (ANIMAZIONE 3), ecc.

VALORI DELLA POSIZIONE X OLTRE 255

Le posizioni di X vanno da 0 a 255 e poi RIPARTONO da 0 a 255. Per posizionare un'animazione oltre la posizione 255 di X sulla destra dello schermo, impostare prima POKE V+16 come gia' indicato, poi usare POKE per impostare un nuovo valore di X da 0 a 63, che posiziona l'animazione in una delle posizioni di X a destra dello schermo. Per ritornare alle posizioni 0-255, usare POKE V+16,0, quindi una seconda POKE i cui valori di X siano compresi fra 0 e 255.

VALORI DELLA POSIZIONE Y

Queste posizioni vanno da 0 a 255, comprendenti i valori da 0 a 49 fuori dalla parte superiore del quadro, quelli da 50 a 229 nel quadro, e quelli da 230 a 255 fuori dalla parte inferiore del quadro.

COLORI DI UN'ANIMAZIONE

Per colorare di bianco l'animazione 0, battere POKE V+39,1 (usare le IMPOSTAZIONI DEL COLORE TRAMITE POKE illustrate in tabella, ed i codici di colore individuale illustrati sotto):

0-NERO	4-PORPORA	8-ARANCIO	12-GRIGIO MEDIO
1-BIANCO	5-VERDE	9-MARRONE	13-VERDE CHIARO
2-ROSSO	6-BLU	10-ROSSO CHIARO	14-BLU CHIARO
3-AZZURRO	7-GIALLO	11-GRIGIO SCURO	15-GRIGIO CHIARO

LOCAZIONE DELLA MEMORIA

Per ogni animazione si deve riservare, nella memoria del computer, un BLOCCO separato di 64 BYTE, dei quali 63 vengono usati dai dati dell'animazione. Le impostazioni della memoria che seguono sono consigliate per l'impostazione del puntatore dell'animazione della tavola precedente. Ogni animazione e' unica e puo' essere definita come si desidera. Per fare tutte le animazioni uguali, puntare le animazioni che si desiderano uguali allo stesso registro delle animazioni.

DIFFERENTI IMPOSTAZIONI DEL PUNTATORE DELL'ANIMAZIONE

Questi sono SOLAMENTE ACCORGIMENTI da seguire per impostare il puntatore di un'animazione.
ATTENZIONE - I puntatori dell'animazione possono essere piazzati dovunque nella RAM, ma se sono troppo in "basso" un LUNGO PROGRAMMA BASIC si puo' sovrapporre ai dati dell'animazione, o viceversa. Per evitare cio', si possono piazzare le animazioni in un'area di memoria piu' alta (ad esempio 2040,192 per l'animazione 0 alle locazioni da 12288 a 12350, 2041,193 per l'animazione 1 alle locazioni da 12352 a 12414, ecc.); adeguando le locazioni di memoria da cui le animazioni traggono i "dati", si possono definire la bellezza di 64 differenti animazioni oltre a un notevole programma BASIC. A questo fine, occorre definire diverse "figure" di un'animazione nelle istruzioni DATA e poi ridefinire una particolare animazione modificando il "puntatore" in modo che tale animazione venga "puntata" a differenti aree di memoria contenenti differenti dati per la rappresentazione dell'animazione. Un esempio di questo funzionamento e' dato nel "TOPOLINIO BALLERINO" al quale si rimanda. Se si vuole che due o piu' animazioni assumano la stessa figura (pur cambiando posizione e colore di ciascuna), usare lo stesso puntatore e la stessa locazione di memoria delle animazioni che si vogliono uguagliare (ad esempio si possono puntare alla stessa locazione le animazioni 0 e 1 usando POKE 2040,192 e POKE 2041,192).

PRIORITÀ

Significa che un'animazione viene visualizzata davanti o dietro ad un'altra. Le animazioni di maggiore priorita' compaiono sempre davanti o sopra quelle di priorita' minore. La regola e' che animazioni di bassa numerazione hanno la priorita' su quelle di alta numerazione. L'animazione 0 ha priorita' massima, la 7 minima; l'animazione 1 ha priorita' su 2-7, ecc. Mettendo due animazioni nella stessa posizione, quella di priorita' maggiore appare davanti a quella di priorita' minore. L'animazione di priorita' minore viene oscurata oppure "mostrata attraverso" quella di priorita' maggiore.

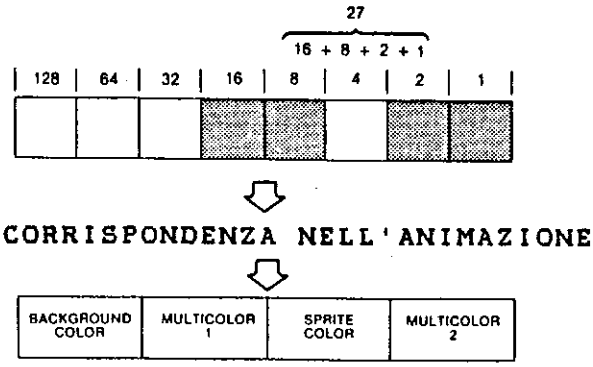
USO DEL MULTICOLORE

Si possono creare animazioni multicolore, anche se questa tecnica richiede l'uso di coppie di pixel (in altre parole, ogni "punto" o "blocco" colorato dell'animazione e' formato dal doppio di pixel per ogni lato). Si hanno a disposizione 4 colori: colore animazione (tabella precedente), multicolor 1, multicolor 2 e "colore di fondo" (ottenuto usando impostazioni a 0 che lasciano "intravedere il colore di fondo). Consideriamo un blocco di 8 pixel orizzontali del disegno di un'animazione. Il colore di ogni coppia di pixel e' determinato a

seconda che sia pieno il pixel di destra sinistra, entrambi o nessuno:

▯▯	SOTTOFONDO	(Se ENTRAMBI I PIXEL SONO VUOTI (zero), appare il colore di fondo dello schermo)
▯▒	MULTICOLORE 1	(Se il PIXEL DI DESTRA di una coppia di pixel e' PIENO, ENTRAMBI I PIXEL vengono impostati al Multicolore 1)
▒▯	COLORE ANIMAZIONE	(Se il PIXEL DI SINISTRA di una coppia di pixel e' PIENO, ENTRAMBI I PIXEL vengono impostati al Colore Animazione)
▒▒	MULTICOLORE 2	(Se ENTRAMBI I PIXEL di una coppia di pixel sono PIENI ENTRAMBI I PIXEL vengono impostati al Multicolore 2)

Consideriamo la riga di 8 pixel orizzontali illustrati sotto. Questo blocco imposta i primi due pixel al colore di fondo, i secondi due al multicolor 1, i terzi due al colore animazione ed i quarti due al multicolor 2. Il colore di ogni coppia di pixel dipende da quali bit di ogni coppia sono pieni e quali vuoti, secondo la precedente illustrazione. Dopo aver determinato i colori di ogni coppia di pixel bisogna aggiungere nel blocco di 8 pixel i valori dei pixel pieni e posizionare il risultato con una POKE nella locazione di memoria adatta. Ad esempio, se la riga di 8 pixel illustrata sotto e' il primo blocco di un'animazione che inizia alla locazione 832, il valore dei pixel pieni e' 16+8+2+1=27, percio' si deve impostare POKE832,27.



CONTATTI:

Si puo' scoprire se un'animazione e' entrata in collisione con un'altra usando questa istruzione:

IF PEEK(V+30)AND X=X THEN <AZIONE>.

Tale istruzione controlla se una data animazione e' entrata in collisione con un'altra animazione, dove X e' 1 per l'animazione 0, 2 per la 1, 4 per la 2,....,128 per la 7.

Per scoprire se l'animazione e' entrata in collisione con un "CARATTERE DI SOTTOFONDO", impostare:

IF PEEK(V+31)AND X=X THEN <AZIONE>

USO DEI CARATTERI GRAFICI NELLE ISTRUZIONI DATA

Il seguente programma permette di creare un'animazione usando spazi e cerchi pieni (SHIFT Q) nelle istruzioni data. L'animazione ed i numeri posizionati con una POKE nei registri dei dati dell'animazione vengono visualizzati.

```
10 PRINT"■" :FORI=0TO63:POKE832+I,0:NEXT
20 GOSUB60000
999 END
60000 DATA"          OOOOOOO .        "
60001 DATA"        OOOOOOOOOOO        "
60002 DATA"       OOOOOOOOOOOOO       "
60003 DATA"       OOOOO    OOOOO      "
60004 DATA"      OOOOO OOO  OOOO      "
60005 DATA"      OOOOO OOO  OOOO      "
60006 DATA"      OOOOO OOO  OOOO      "
60007 DATA"       OOOOO    OOOOO      "
60008 DATA"       OOOOOOOOOOOOO      "
60009 DATA"       OOOOOOOOOOOOO      "
60010 DATA"       O OOOOOOOOO O      "
60011 DATA"        O OOOOOOO O       "
60012 DATA"        O  OOOOO  O       "
60013 DATA"         O  OOO  O        "
60014 DATA"         O  OOO  O        "
60015 DATA"          O  O  O         "
60016 DATA"          O  O  O         "
60017 DATA"           OOOOO          "
60018 DATA"           OOOOO          "
60019 DATA"           OOOOO          "
60020 DATA"            OOO           "
60100 V=53248:POKEV,200:POKEV+1,100:POKEV+21,1:POKEV+39,14:POKE2040,13
60105 POKEV+23,1:POKEV+29,1
60110 FORI=0TO20:READA$:FORK=0TO2:T=0FORJ=0TO7:B=0
60140 IFMID$(A$,J+K*S+1,1)="O"THENB=1
60150 T=T+B*21(7-J):NEXT:PRINTT;:POKE832+I*3+K,T:NEXT:PRINT:NEXT
60200 RETURN
```

CAPITOLO 4

programmazione di suoni e musica con il commodore 64

- Introduzione
  Controllo del Volume
  Frequenze delle Onde Sonore
- Uso delle Voci Multiple
- Modifica delle Forme d'Onda
- Il Generatore di Inviluppo
- Filtratura
- Tecniche Avanzate
- Sincronizzazione e Modulazione Circolare

INTRODUZIONE

Il COMMODORE 64 e' dotato di uno dei piu' sofisticati sintetizzatori elettronici musicali disponibili su qualunque computer. Viene fornito completo di tre Voci totalmente indirizzabili, un Generatore ADSR (ATTACCARE/DECOMPORRE/SOSTENERE/RILASCIARE), filtratura, modulazione e "rumore bianco". Tutte queste capacita' sono messe a disposizione da poche istruzioni e funzioni del BASIC e/o del linguaggio assemblatore, facili da usare. Cio' significa che si possono comporre canzoni e suoni molto complessi usando programmi di progettazione relativamente semplice.

Questo Capitolo e' stato concepito per agevolare l'esplorazione di tutte le capacita' del Circuito 6581 "SID", il sintetizzatore sonoro e musicale. Vengono spiegate sia la teoria musicale, sia gli aspetti pratici che si incontrano nella conversione di tale teoria in composizioni reali complete.

Non e' necessario essere un programmatore o un musicista esperto per raggiungere con il sintetizzatore musicale dei risultati stimolanti. Questo Capitolo e' ricco di esempi di programmi, completi di spiegazione, da cui prendere spunto.

L'accesso al Generatore del Suono avviene tramite una POKE a particolari locazioni di memoria. La lista completa delle locazioni usate e' riportata nell'Appendice O. Ogni concetto viene illustrato passo dopo passo, in modo da mettere in condizione, verso la fine, di creare una varieta' di suoni quasi infinita, e di realizare esperimenti sonori in proprio.

Ogni paragrafo di questo Capitolo inizia dando un esempio di programma ed illustrandolo linea per linea, in modo da esporne le caratteristiche. La spiegazione tecnica e' a disposizione per tutte le occasioni in cui si desideri saperne di piu' su quanto sta accadendo.

L'istruzione fondamentale dei programmi musicali e' POKE; essa uguaglia la locazione di memoria indicata (MEM) ad un valore specificato (NUM):

```
POKE MEM,NUM
```

Le locazioni di memoria (MEM) usate per la sintesi musicale iniziano, sul COMMODORE 64, alla locazione 54272 ($D400 HEX). Le locazioni che vanno da 54272 a 54296, estremi inclusi, sono le locazioni da ricordare quando si fa uso della mappa registro del Circuito 6581 (SID). Un altro metodo per ricordare tali locazioni e' di fissare la sola locazzione 54272, e poi di aggiungere un numero da 0 a 27. I numeri (NUM) da usare con l'istruzione POKE devono essere compresi fra 0 e 255.

Una volta presa pratica con la composizione della musica, si puo' approfondire la conoscenza ricorrendo alla funzione PEEK: questa e' una funzione che ritorna il valore corrente contenuto nella locazione di memoria indicata:

```
X=PEEK(MEM)
```

Il valore della variabile X e' posto uguale al contenuto corrente della locazione di memoria MEM.

Ovviamente, i programmi comprendono anche altre istruzioni BASIC, per la cui spiegazione si rimanda al Capitolo Istruzioni BASIC di questo Manuale.

Proviamo ora a scrivere un semplice programma usando una sola delle tre Voci a disposizione. Predisporre il computer, battere NEW, poi il programma seguente, ed infine RUN. Salvare quindi il programma su disco o su DATASSETTE (TM) Commodore.

ESEMPIO - PROGRAMMA 1:

```
 5 S=54272
 10 FORL=STOS+24:POKEL,0:NEXT:REM AZZERA IL CIRCUITO SONORO
 20 POKES+5,9:POKES+6,0
 30 POKES+24,15                :REM IMPOSTA IL VOLUME A MASSIMO
 40 READHF,LF,DR
 50 IFHF<0THENEND
 60 POKES+1,HF:POKES,LF
 70 POKES+4,33
 80 FORT=1TODR:NEXT
 90 POKES+4,32:FORT=1TO50:NEXT
 100 GOTO40
 110 DATA25,177,250,28,214,250
 120 DATA25,177,250,25,177,250
 130 DATA25,177,125,28,214,125
 140 DATA32,94,750,25,177,250
 150 DATA28,214,250,19,63,250
 160 DATA19,63,250,19,63,250
 170 DATA21,154,63,24,63,63
 180 DATA25,177,250,24,63,125
 190 DATA19,63,250,-1,-1,-1
```

La seguente e' una descrizione linea per linea del programma appena battuto. Per quelle parti del programma non completamente comprese si rimanda a tale descrizione.

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA1 :

LINEA	DESCRIZIONE
5	Imposta S all'inizio del Circuito del Suono
10	Azzera tutti i registri del Circuito del Suono
20	Imposta ATTACCARE/DECOMPORRE per la Voce 1 (A=0, D=9) Imposta SOSTENERE/RILASCIARE per la Voce 1 (S=0, R=0)
30	Imposta il Volume al massimo
40	Legge l'Alta Frequenza, la Bassa Frequenza e la durata della Nota
50	Termina la canzone se Alta Frequenza < 0
60	Posiziona Alta e Bassa Frequenza della Voce 1
70	Introduce la Forma d'Onda "dente di sega" per la Voce 1.
80	Ciclo di tempo per la durata della nota
90	Rilascia la precedente Forma d'Onda
100	Posizionamento per la prossima nota
110-180	Dati della composizione: Alta Frequenza, Bassa Frequenza, Durata (numero di passi del ciclo) di ogni nota
190	Ultima nota della composizione e segnalazione di Termine Composizione (-1 sec)

CONTROLLO DEL VOLUME

Il registro 24 del circuito contiene l'intero controllo del Volume. Quest'ultimo puo' essere posizionato ad un qualunque valore compreso fra 0 e 15. Gli altri 4 bit sono usati per altri scopi che saranno definiti in seguito. La linea 30 del precedente programma illustra come il Volume viene impostato nel Programma 1.

FREQUENZE DELLE ONDE SONORE

Il suono e' generato in forma di onde dal movimento dell'aria. Se si getta un sasso in uno stagno, si possono osservare le onde che si allontanano a raggera dal punto dell'impatto; allo stesso modo, quando onde simili si creano in aria, siamo in grado di udirle. Se si misurano due successivi picchi d'onda, si trova il numero di secondi per ciclo dell'onda (n=numero di secondi). Il reciproco di questo numero (1/n) da' il numero di cicli per secondo; questa quantita' e' conosciuta anche come frequenza. L'acutezza o la profondita' di un suono (la nota) sono determinati dalla frequenza delle onde sonore prodotte.

Il Generatore del Suono del COMMODORE 64 usa due locazioni per determinare la frequenza; l'Appendice E riporta i valori delle frequenze necessari a riprodurre una gamma completa di otto ottave di note musicali. Per creare una frequenza piuttosto che un'altra elencata nella Tavola delle Note, si usi Fout (Frequency OUTput) e la formula seguente, per la rappresentazione della frequenza (Fn) del suono che si desidera creare. Va ricordato che ogni nota richiede un numero sia per l'Alta che per la Bassa Frequenza.

$$F_n = F_{out}/.06097$$

Stabilito il valore di Fn per la "nuova" nota, si tratta ora di creare, per quella nota, i valori di Alta e Bassa Frequenza. Innanzitutto, occorre arrotondare il valore di Fn ad un valore intero; a questo punto, il valore dell'Alta Frequenza e' dato da:

$$F_{hi}=F_n/256$$

mentre quello per la Bassa Frequenza e' dato da:

$$F_{lo}=F_n-(256*F_{hi}).$$

USO DELLE VOCI MULTIPLE

Il COMMODORE 64 ha tre Voci (Oscillatori) controllate indipendentemente; il Programma 1 ne usava solo una. Piu' avanti, viene spiegato come cambiare la qualita' del suono prodotto dalle Voci.

Il programma seguente illustra come trasformare un foglio di carta in un'orchestra...computerizzata. Battere il seguente programma, quindi salvarlo su disco o su DATASSETTE (TM); ricordarsi di battere NEW prima di iniziare la battitura del programma.

ESEMPIO – PROGRAMMA 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
```

```
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):IFNM<0THENNM=-NM:WA=1
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:GOTO120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:NEXT
230 H(K,I)=HF%:L(K,I=LF%:C(K,I)=WA-1
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(I,1):POKES,L(2,1)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,1):POKES+11,C(1,1):POKES+18,H(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592.587.585.331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,585,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,327,329,326,578,583
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1609,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA306,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,310,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,399,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```

La seguente e' una spiegazione linea per linea del Programma 2; ci occuperemo, per ora, di come sono controllate le tre Voci.

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 2:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del Circuito del Suono e azzera tutti i registri di tale Circuito
20	Dimensiona le schiere per contenere l'attivita' della composizione, 1/16 di tempo per locazione
30	Dimensiona le schiere per contenere la Frequenza di Base di ogni nota
40	Registra il byte di controllo della Forma d'Onda per ogni Voce
50	Imposta l'ampiezza dell'impulso alto per la Voce 2 Imposta l'Alta Frequenza per il taglio del filtro Imposta la risonanza per il filtro e la Voce 3 del filtro
60	Legge la frequenza base di ogni nota
100	Inizio del ciclo di decodifica di ogni Voce
110	Inizializza il puntatore alla schiera attivita'
120	Legge la nota codificata
130	Se questa e' zero passa alla Voce successiva
140	Imposta il controllo della forma d'onda alla Voce appropriata
150	Decodifica durata ed ottava
160	Decodifica la nota
170	Acquisisce la frequenza base di questa nota
180	Se e' l'ottava piu' alta, salta il ciclo di divisione
190	Divide la frequenza base per due appropriate quantita' di tempo
200	Acquisisce i bytes di Alta e Bassa Frequenza
210	Se e' la sedicesima nota, imposta la schiera attivita': Alta Frequenza, Bassa Frequenza e controllo della forma d'onda (Voce ON)
220	Per tutte le battute meno l'ultima imposta la schiera attivita': alto
230	Per l'ultima battuta, imposta la schiera attivita' : Alta Frequenza, Bassa Frequenza, controllo della forma d'onda (Voce OFF)
240	Incrementa il puntatore alla schiera attivita' Acquisisce la prossima nota
250	Se e' piu' lunga della precedente, imposta daccapo il numero delle attivita'
260	Ritorna alla prossima Voce
500	Imposta ATTACCARE/DECOMPORRE per la Voce 1 (A=0, D=0)
510	Come sopra, ma per la Voce 2 (A=5, D=5, S=8, R=5)
520	Come 500, ma per la Voce 3 (A=0, D=10, S=12, R=5)
530	Imposta il Volume a 15 ed il filtro passabasso
540	Inizio del ciclo per ogni 1/16 di tempo
550	Preleva (POKE) la Bassa Frequenza dalla schiera delle attivita' per tutte le Voci
560	Come sopra, ma per l'Alta Frequenza
570	Come 550 ma per la forma d'onda
580	Ciclo di tempo per 1/16 di tempo e ritorno per il prossimo
590	Pausa, poi disattiva il Volume
600-620	Dati della frequenza base
1000-1999	Dati della Voce 1
2000-2999	Dati della Voce 2
3000-3999	dati della Voce 3

I valori usati nelle istruzioni DATA sono stati calcolati usando la Tabella delle Note (Appendice E) e la Tavola seguente:

TIPO DI NOTA	DURATA
1/16	128
1/8	256
1/8 Punteggiato	384
1/4	512
1/4 + 1/16	640
1/4 Punteggiato	768
1/2	1024
1/2 + 1/16	1152
1/2 + 1/8	1280
1/2 Punteggiato	1536
Intero	2048

Il numero della nota preso dalla Tabella delle Note viene sommato alla precedente durata. Successivamente, si puo' introdurre ogni nota usando solamente un numero decodificato dal programma. La formula usata per codificare una nota e' la seguente:

1) Si moltiplica la durata (numero di sedicesimi del tempo) per 8
2) Si aggiunge al risultato l'ottava che si e' scelto (0-7)
3) Si moltiplica il risultato per 16
4) Si aggiunge al risultato la nota che si e' scelto

In altri termini:

((((D*8)+O) *16)+N)

essendo D=Durata, O=Ottava, N=Nota.
Il silenzio si ottiene usando il negativo della quantita' della durata (numero di sedicesimi di tempo X 128) (Questo e' soltanto uno dei possibili metodi di codifica, fra i quali si puo' scegliere quello individualmente piu' confacente).

CONTROLLO DELLE VOCI MULTIPLE

Dopo aver imparato ad usare piu' di una Voce, si puo' osservare che e' necessario coordinare il ritmo delle tre Voci. Nel programma precedente, cio' viene realizzato:

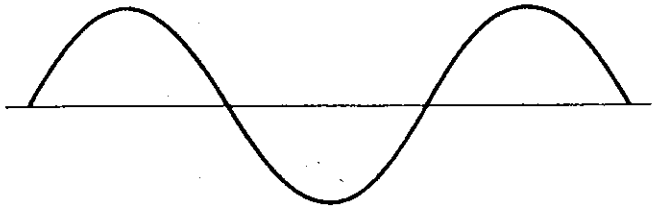
1) Dividendo ogni tempo musicale in 16 parti
2) Memorizzando in tre schiere separate gli eventi che accadono ad ogni intervallo di sedicesimo di tempo

I bytes di Alta e Bassa Frequenza sono calcolati dividendo per 2 le frequenze dell'ottava piu' alta (linee 180 e 190). Il byte di controllo della forma d'onda e' unsegnale di partenza per iniziare una nota o continuarne un'altra che sta gia' suonando; analogamente, lo stesso byte viene usato come segnale di fine nota. La scelta della forma d'onda viene effettuata, per ogni Voce, alla linea 40.
Questo e' solamente uno dei possibili metodi di controllo delle Voci multiple, fra i quali si puo' scegliere quello individualmente piu' confacente; quello che importa, tuttavia, e' di essere in grado di

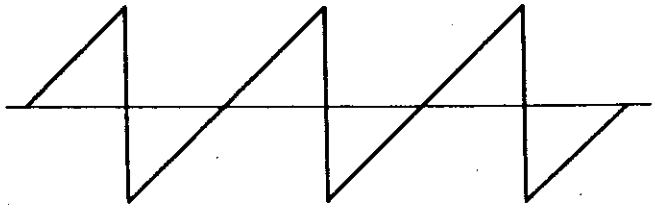
rappresentare le note per tutte le tre Voci.

MODIFICA DELLE FORME D'ONDA

La qualita' tonale di un suono si chiama TIMBRO. Il timbro di un suono e' determinato essenzialmente dalla sua "forma d'onda". Se si ricorda l'esempio del sasso gettato in acqua, si ricordera' anche che le onde si propagano uniformemente sullo stagno. Queste onde assomigliano alquanto alla prima onda sonora di cui stiamo per parlare: l'onda sinusoidale (illustrata sotto).



Per rendere un po' piu' pratico l'argomento di cui stiamo parlando, ritorniamo al Programma 1 per esaminare differenti forme d'onda, in quanto le modifiche che stiamo per apportare sono piu' facili se si usa una sola Voce. Questo programma usa una forma d'onda a "dente di sega" come la seguente:



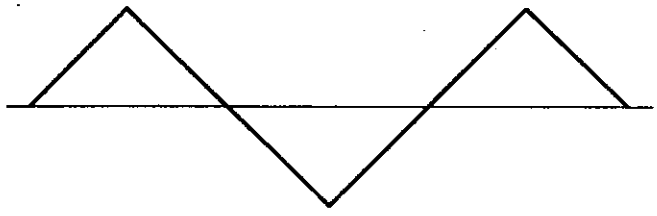
che gli deriva dal Dispositivo Generatore del Suono del Circuito 6581 SID. Se si cambia il numero di partenza della nota nella linea 70 da 33 a 17, ed il corrispondente numero di arrivo nella linea 90 da 32 a 16, si ottiene il seguente programma:

ESEMPIO - PROGRAMMA 3 (PROGRAMMA 1 MODIFICATO):

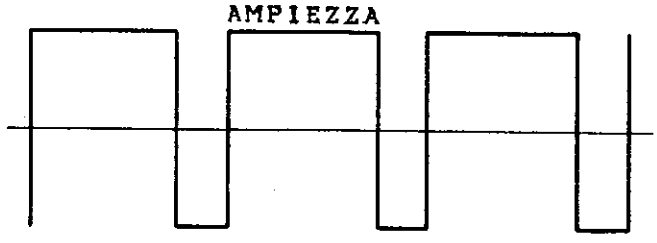
```
 5 S=54272
 10 FORL=STOS+24:POKEL,0:NEXT
 20 POKES+5,9:POKES+6,0
 30 POKES+24,15
 40 READHF,LF,DR
 50 IFHF<0THENEND
 60 POKES+1,HF:POKES,LF
 70 POKES+4,17
 80 FORT=1TODR:NEXT
```

```
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Se ora lanciamo (RUN) il programma, notiamo che la qualita' del suono e' diversa, meno stridula e piu' cupa: cio' poiche' abbiamo cambiato la forma d'onda da "dente di sega" a triangolare.



La terza forma d'onda e' detta ad impulso variabile:



Come illustrato in figura, questa e' un'onda rettangolare di cui si deve determinare la lunghezza del ciclo di pulsazione, definendo la proporzione della parte alta dell'onda. Cio' e' ottenuto, per la Voce 1, usando i registri 2 e 3: il registro 2 e' il byte basso dell'ampiezza dell'impulso (Lpw=0...255); il registro 3 sono i quattro bit alti (Hpw=0...15).

Questa coppia di registri specifica un numero a 12 bit per l'ampiezza dell'impulso; tale numero puo' essere determinato dalla seguente formula:

$$PW_n = H_{pw}*256 + L_{pw}$$

mentre l'ampiezza del suono e' determinata dalla seguente relazione:

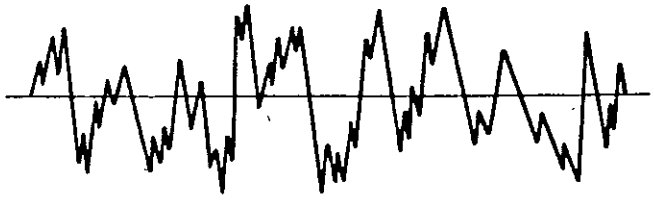
$$PW_{out} = (PW_n/40.95)\ \%$$

Quando PWn assume il valore 2048, l'onda assume una forma quadrata:

cio' vuol dire che il registro 2 (Lpw)=0 ed il registro 3 (Hpw)=8.
Aggiungiamo ora al programma in questione la seguente riga:

```
15 POKES+3,8:POKES+2,0
```

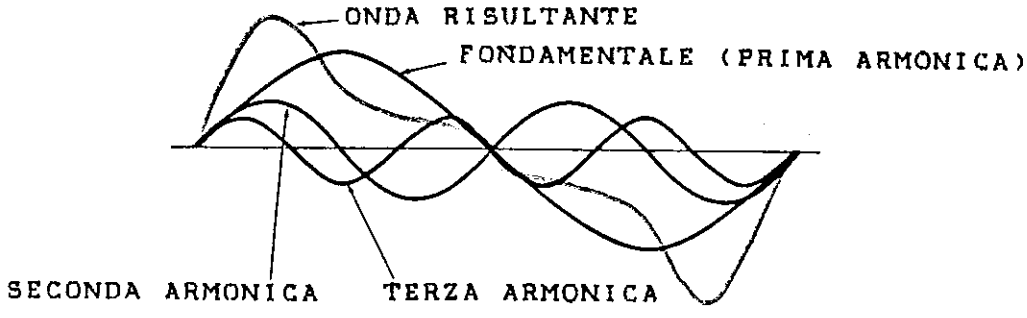
e cambiamo il numero di inizio nella linea 70 in 65, e quello di fine nella linea 90 in 64; lanciamo (RUN) infine il programma. Se a questo punto si modifica l'ampiezza della pulsazione alta (registro 3 alla linea 15) da 8 a 1, si nota una differenza di suono...drammatica: quest'ultima forma d'onda e' il rumore bianco:



Questa forma d'onda e' usata soprattutto per effetti sonori. Per udire questo suono, occorre cambiare il numero di inizio nella linea 70 in 129, ed il numero di fine nella linea 90 in 128.

INTRODUZIONE ALLE FORME D'ONDA

Una nota che viene suonata e' formata da un'onda sinusoidale, oscillante alla frequenza fondamentale, e dalle armoniche di quell'onda.
La frequenza fondamentale definisce completamente la tonalita' della nota. Le armoniche sono onde sinusoidali la cui frequenza e' un multiplo intero della frequenza fondamentale. Un'onda sonora e' composta dalla frequenza fondamentale e da tutte le armoniche richieste per formare quel suono.



La teoria musicale assume l'armonica numero 1 come frequenza fondamentale; la seconda armonica ha una frequenza doppia di quella fondamentale, la terza armonica tripla, ecc. La quantita' di ogni armonica presente in una nota da' il timbro della nota stessa.
Uno strumento acustico, come una chitarra o un violino, ha una struttura armonica molto complessa, per il fatto che tale struttura puo' variare a seconda di come viene variata una singola nota. Le forme d'onda disponibili al sintetizzatore musicale del COMMODORE 64 sono gia' state esaminate; rimane da affrontare il problema di come

funzionano le armoniche con onde triangolari, rettangolari ed a "dente di sega".

Un'onda triangolare possiede soltanto armoniche casuali; la quantita' di ogni armonica presente e' proporzionale al reciproco del quadrato del numero di armonica. In altre parole, l'armonica numero 3 e' 1/9 piu' dolce dell'armonica numero 1, in quanto il quadrato di 3 e' 9 (3 X 3 =9), ed il suo reciproco e' 1/9.

Come si puo' osservare, c'e' una somiglianza nella forma di un'onda triangolare rispetto ad un'onda sinusoidale oscillante alla frequenza fondamentale.

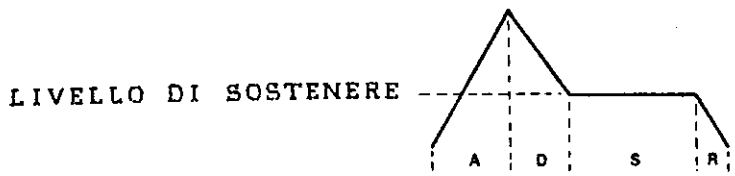
Un'onda a "dente di sega" contiene tutte le armoniche; la quantita' di ogni armonica presente e' proporzionale al reciproco del numero di armonica. Ad esempio, l'armonica numero 2 e' profonda 1/2 rispetto all'armonica numero 1.

L'onda rettangolare contiene armoniche casuali in proporzione al reciproco del numero di armonica. Onde rettangolari diverse hanno un diverso contenuto armonico. Cambiando l'ampiezza dell'impulso, viene modificato grandemente il timbro del suono di un'onda rettangolare.

Scegliendo attentamente la forma d'onda usata, si puo' dare inizio ad una struttura armonica che assomiglia in qualche modo al suono che si desidera riprodurre. Per la rifinitura di tale suono, si puo' aggiungere un'altra caratteristica della qualita' del suono disponibile sul COMMODORE 64, chiamata filtratura, della quale parleremo piu' avanti.

IL GENERATORE DI INVILUPPO

Il volume di un tono musicale cambia dal momento in cui viene percepito, via via fino alla sua scomparsa, quando non puo' piu' essere udito. Quando una nota viene suonata per la prima volta, il suo volume sale da zero al suo volume di picco. Il passo in cui cio' si verifica si chiama ATTACCARE. Successivamente, la nota scende di volume dal valore di picco ad un valore medio: questo passo prende il nome di DECOMPORRE, mentre il livello medio raggiunto si chiama SOSTENERE. Quando infine la nota cessa di suonare, il Volume passa dal livello di SOSTENERE al valore zero: il passo in cui cio' si verifica si chiama RILASCIARE. Una rappresentazione di queste quatro fasi e' data qui sotto:



Ognuno dei quattro livelli sopra menzionati conferisce ad una nota certe qualita' e restrizioni; i quattro livelli si chiamano parametri, e vengono indicati collettivamente con le rispettive iniziali (ADSR); il loro controllo avviene per mezzo di un insieme di locazioni del Circuito Generatore del Suono. Riprendiamo il primo programma di questo capitolo, carichiamolo e lanciamolo cercando di ricordarsi il suono emesso. A questo punto, cambiamo la linea 20 in modo che il

programma diventi come il seguente:

ESEMPIO - PROGRAMMA 4 (PROGRAMMA 1 MODIFICATO):

```
 5 S=54272
 10 FORL=STOS+24:POKEL,0:NEXT
 20 POKES+5,88:POKES+6,195
 30 POKES+24,15
 40 REDAHF,LF,DR
 50 IFHF<0THENEND
 60 POKES+1,HF
 70 POKES+4,33
 80 FORT=1TODR:NEXT
 90 POKES+4.32:FORT=1TO50:NEXT
 100 GOTO40
 110 DATA25.177,250,28,214,250
 120 DATA25,177,250,25,177,250
 130 DATA25,177,125,28,214,125
 140 DATA32,94,750,25,177,250
 150 DATA28,214,250,19,63,250
 160 DATA19,63,250,19,63,250
 170 DATA21.154,63,24,63,63
 180 DATA25,177,25,24,63,125
 190 DATA19.63,250,-1,-1,-1
```

I registri 5 e 6 definiscono l'ADSR per la Voce 1: l'ATTACCARE e' il semibyte alto del registro 5 (si intende per semibyte l'insieme delle quattro locazioni (bit) piu' alte o piu' basse di un registro), mentre il DECOMPORRE e' il semibyte basso. Per l'ATTACCARE si puo' scegliere qualunque numero compreso fra 0 e 15; moltiplicandolo poi per 16 ed aggiungendo un numero compreso fra 0 e 15 si ottiene il DECOMPORRE. I valori che corrispondono a questi numeri sono elencati piu' in basso.

Il livello SOSTENERE e' il semibyte alto del registro 6; e' compreso fra 0 e 15; definisce la quantita' del volume di picco posseduta. Il RILASCIARE e' il semibyte basso del registro 6.

VALORE	VALORE DI ATTACCARE (TEMPO/CICLO)	VALORE DI DECOMPORRE/RILASCIARE (TEMPO/CICLO)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

I seguenti sono solo pochi esempi di modifiche da apportare al precedente programma, che insieme alle modifiche realizzabili individualmente forniscono una varieta' di suoni veramente sbalorditiva! Se, ad esempio, si vuole riprodurre il suono di un violino, occorre modificare la linea 20 come segue:

```
20 POKES+5,88:POKES+6,89:REM A=5;D=8;S=5;R=9
```

mentre uno xilofono sara' riprodotto modificando in triangolare la forma d'onda, ed introducendo le seguenti linee:

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9
70 POKES+4,17
90 POKES+4,16: FORT=1TO50:NEXT
```

S=0;R=9

Se poi si usa un'onda rettangolare si potra' ottenere il suono di un pianoforte impostando:

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0: REM A=0;D=9;S=0;R=0
70 POKES+4,65
90 POKES+4,64:FORT=1TO50:NEXT
```

Ma i suoni piu' emozionanti sono quelli che solo il sintetizzatore possiede e che non imitano alcuno strumento acustico; ad esempio:

```
20 POKES+5,144:POKES+6,243:REM A=9;D=0; S=15;R=3
```

FILTRATURA

Il contenuto sonoro di una forma d'onda puo' essere modificato usando un filtro. Il circuito SID e' equipaggiato con tre tipi di filtri, che possono essere usati singolarmente o in combinazione. Come dimostrazione dell'uso di un filtro, torniamo a considerare il PROGRAMMA 1: ci sono diversi controlli di filtro da impostare.

Per impostare la FREQUENZA DI TAGLIO del filtro aggiungere la linea 15; la frequenza di taglio e' il punto di riferimento del filtro. I punti delle frequenze di taglio alte e basse vengono IMPOSTATE nei registri 21 e 22. L'attivazione del filtro per la Voce 1 avviene caricando (POKE) il registro 23.

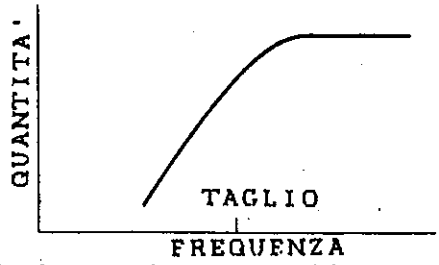
Successivamente, per mostrare che si usa un filtro passa alto occorre modificare la linea 30 (vedere la mappa dei registri del SID).

ESEMPIO – PROGRAMMA 5 (PROGRAMMA 1 MODIFICATO):

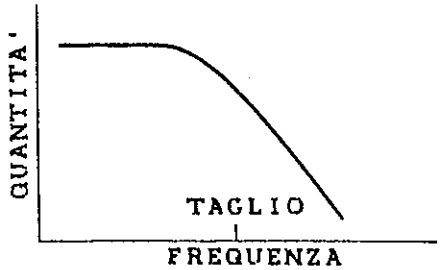
```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+2,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT1TODR:NEXT
90 POKES+24,32:FORT1=TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Se ora si prova a lanciare il programma, si notera' che i toni bassi hanno un volume ridotto; cio' provoca un cambiamento della qualita' complessiva del suono della nota, che ora sembrera' piu' metallica: infatti, il filtro passa alto usato attenua (riduce) le frequenze al disotto della frequenza di taglio usata.

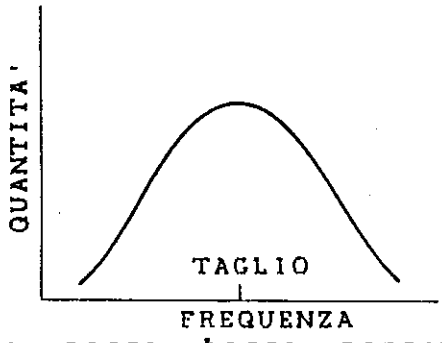
Il circuito SID del COMMODORE 64 possiede tre tipi di filtri; abbiamo appena usato il filtro passa alto, che lascia passare tutte le freguenze maggiori o uguali a quella di taglio, mentre attenua le frequenze al di sotto di quella di taglio.



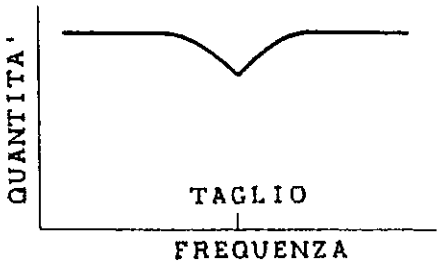
Il circuito SID possiede anche un filtro passa basso, che lascia passare le frequenze al di sotto di quelle di taglio ed attenua quelle al di sopra.



Infine si ha a disposizione un filtro passa banda, che lascia passare una banda di frequenze ristrette intorno alla frequenza di taglio, attenuando tutte le altre.



I filtri passa alto e passa basso possono essere combinati per formare un filtro di rigetto del taglio, che attenua la frequenza di taglio lasciando passare tutte le altre.



Il registro 24 determina il tipo di filtro usato: questo in aggiunta alle altre funzioni di questo registro, come il controllo completo del volume. Il bit 6 controlla il filtro passa alto (0=OFF, 1=ON), il bit 5 il filtro passa banda e il bit 4 il filtro passa basso. I tre bit bassi della frequenza di taglio sono determinati dal registro 21 (Lcf=0...7), mentre il registro 22 determina gli 8 BIT della frequenza di taglio alta (Hcf=0...255). Per mezzo di un uso oculato di filtri, si puo' cambiare la struttura armonica di qualunque forma d'onda, ottenendo cosi' il suono desiderato. Inoltre, si possono produrre effetti interessanti cambiando la filtratura di un suono durante il suo movimento attraverso le 4 fasi ADSR.

TECNICHE AVANZATE

I parametri del circuito SID possono essere modificati dinamicamente durante la riproduzione di una nota o di un suono, creando cosi' molti effetti interessanti e divertenti. Allo scopo di facilitare la loro messa in paratica, sono disponibili, rispettivamente nei registri 27 e 28, le uscite digitalizzate provenienti dall'oscillatore 3 e dal generatore di inviluppo 3. L'uscita dell'oscillatore 3 (registro 27) e' collegata direttamente alla forma d'onda scelta. Se ad esempio si sceglie la forma d'onda a "dente di sega", questo registro presenta una serie di numeri crescenti da 0 a 255 secondo un incremento determinato dalla frequenza dell'oscillatore 3. Se invece si sceglie una forma d'onda triangolare, l'uscita viene prima incrementata da 0 a 255, poi decrementata da 255 a 0. Se, infine, si sceglie la forma d'onda di un rumore, si ottiene una serie di numeri casuali. Quando

l'oscillatore 3 viene usato per la modulazione, di solito NON si desidera udire la sua uscita: cio' e' possibile impostando a 0 (OFF) il bit 7 del registro 24, che esclude l'OUTPUT audio della voce 3. Il registro 27 riporta sempre l'uscita modificata dell'oscillatore; questo registro non viene influenzato in alcun modo dal generatore d'inviluppo (ADSR).

L'accesso all'uscita del generatore d'inviluppo dell'oscillatore 3 e' data dal registro 25, che funziona quali allo stesso modo dell'uscita dell'oscillatore 3. Questo oscillatore deve essere attivato per produrre qualunque uscita da questo registro.

L'effetto "vibrato" (una rapida variazione di frequenza) puo' essere ottenuto aggiungendo l'uscita dell'oscillatore 3 alla frequenza di un'altro oscillatore; questa idea e' illustrata nel programma 6.

ESEMPIO – PROGRAMMA 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+4,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQAND255
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5103,2,5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4,8583,12,9634,2
530 DATA10207,2,10814,2,8583,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 6:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera tutte le locazioni del circuito del suono
30	Imposta la voce 1 ad alta ampiezza di pulsazione
40	Imposta ATTACCARE/DECOMPORRE per questa voce (A=2, D=9) Imposta SOSTENERE/RILASCIARE per questa voce (S=5,R=9)
50	Imposta la voce 3 a bassa frequenza
60	Imposta la forma d'onda triangolare per questa voce
70	Imposta il volume a 15; disattiva l'uscita audio della voce 3
80	Legge frequenze e durata della nota
90	Se tale frequenza e' 0 si ferma
100	Imposta all'inizio il controllo della pulsazione della forma d'onda della voce 1
110	Inizio del ciclo di tempo della battuta
120	Acquisisce una nuova frequenza usando l'uscita dell'oscillatore 3
130	Acquisisce alta e bassa frequenza
140	Imposta alta e bassa frequenza per la voce 1
150	Fine del ciclo di tempo della battuta
160	Imposta alla fine il controllo della pulsazione della forma d'onda della voce 1
170	Passa alla prossima nota.
500-550	Frequenze e battute della composizione
560	Segnale di fine composizione

Una vasta gamma di effetti sonori puo' essere ottenuta anche per mezzo di effetti dinamici. Ad esempio, il seguente programma, che

realizza l'ululato di una sirena, modifica dinamicamente l'uscita della frequenza dell'oscillatore 1, quando quest'ultimo sia stato regolato sull'uscita dell'onda triangolare dell'oscillatore 3:

ESEMPIO – PROGRAMMA 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 7:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta la bassa frequenza della voce 3
40	Imposta la forma d'onda triangolare per questa voce
50	Imposta la voce 1 ad alta ampiezza di pulsazioni
60	Imposta il volume a 15 e disattiva l'uscita audio
70	Imposta SOSTENERE/RILASCIARE per la voce 1 (S=15, R=0)
80	Imposta all'inizio il controllo della pulsazione della forma d'onda della voce 1
90	Imposta la frequenza piu' bassa per la sirena
100	Inizio del ciclo di tempo
110	Aquisisce una nuova frequenza usando l'uscita dell'oscillatore 3
120	Aquisisce alta e bassa freguenza
130	Imposta alta e bassa frequenza per la voce 1
140	Fine del ciclo di tempo
150	Disattiva il volume

La forma d'onda del rumore puo' essere usata per fornire una varieta' di effetti sonori. Questo esempio imita un applauso usando una forma d'onda di rumore filtrato:

ESEMPIO – PROGRAMMA 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

DESCRIZIONE LINEA PER LINEA DEL PROGRAMMA 8:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta alta e bassa frequenza per la voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa voce (A=0, D=8)
50	Imposta l'alta frequenza di taglio per il filtro
60	Attiva il filtro per la voce 1
70	Imposta il volume a 15 del filtro passa alto
80	Conta 15 applausi
90	Imposta all'inizio il controllo della forma d'onda rumore
100	Attesa, poi imposta alla fine il controllo della forma d'onda del rumore
110	Attesa, poi inizia il prossimo applauso
120	Disattiva il volume

SINCRONIZZAZIONE E MODULAZIONE CIRCOLARE

Il circuito 6581 SID permette di creare strutture armoniche piu' complesse per mezzo della sincronizzazione o modulazione circolare di due Voci.

Il processo di sincronizzazione consiste fondamentalmente in una AND logica di due forme d'onda; quando entrambi sono zero, il risultato e' nullo. L'esempio seguente usa questo processo per imitare una zanzara:

ESEMPIO – PROGRAMMA 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```

DESCRIZIONE LINEA PER LINEA DEL PROGRAMMA 9:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta ad alta frequenza la Voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa Voce (A=13, D=11)
50	Imposta ad alta frequenza la Voce 3
60	Imposta il Volume a 15
70	Imposta all'inizio il controllo del sincronismo e della forma d'onda triangolare della Voce 1
80	Ciclo di tempo
90	Imposta alla fine il controllo del sincronismo e della forma d'onda triangolare della Voce 1
100	Attesa, poi disattiva il Volume

La caratteristica della sincronizzazione viene attivata alla linea 70, dove vengono impostati i bit 0, 1 e 4 del registro 4. Il bit 1 attiva la funzione di sincronizzazione tra le Voci 1 e 3; i bit 0 e 4 svolgono le loro funzioni abituali di introduzione della Voce 1 e di impostazione della forma d'onda triangolare.

La modulazione circolare (realizzata, per la Voce 1, impostando a 1 il bit 3 del registro 4 nella linea 70 del programma sseguente) sostituisce l'uscita triangolare dell'oscillatore 1 con una combinazione "modulata ad anello" degli oscillatori 1 e 3. Cio' produce strutture sopratonali non armoniche che trovano impiego nell'imitazione del suono di campane o di gong. Il seguente programma esegue l'imitazione di una sveglia:

ESEMPIO - PROGRAMMA 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 10:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta l'alta frequenza per la Voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa Voce (A=0, D=9)
50	Imposta l'alta frequenza per la Voce 3
60	Imposta il Volume a 15
70	Conta il numero di rintocchi ed imposta all'inizio il controllo della forma d'onda triangolare e della modulazione circolare
80	Ciclo di tempo; imposta alla fine la forma d'onda triangolare e la modulazione circolare
90	Ciclo di tempo; rintocco successivo

Con l'uso dei parametri del circuito SID del COMMODORE 64 si hanno a disposizione effetti numerosi e vari. Solamente con delle prove personali si possono apprezzare appieno le capacita' della macchina: gli esempi dati in questo Capitolo hanno un valore puramente indicativo.

Per avere una conoscenza di tipo piu' professionale, svincolata dal gioco e dal semplice divertimento, si rimanda al testo MAKING MUSIC ON YOUR COMMODORE COMPUTER.