

# GUIDA AL COMMODORE 64

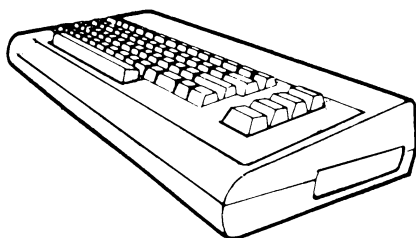
COD. 6900



*E U M - COMPUTER*



# **GUIDA AL COMMODORE 64**



*EUM COMPUTERS*

Copyright 1984 E.V.M Computers

Tutti i diritti sono riservati. Nessuna parte di questo manuale puo' essere riprodotta o posta in sistemi di archiviazione elettronici, meccanici o fotocopiata senza autorizzazione scritta.

V Edizione Luglio 1984

E.V.M Computers  
Via Marconi 9/a  
52025 MONTEVARCHI (AR)



## INTRODUZIONE

Sicuramente il CBM 64 e' uno dei prodotti piu' sofisticati che siano presenti in questo momento sul mercato.

Per quanto riguarda poi il rapporto prezzo/prestazioni, i programmi esistenti, il numero delle periferiche disponibili e i promessi sviluppi futuri e' veramente qualcosa di eccezionale.

Questo manuale vuole essere solo una introduzione all'uso che si puo' fare di questo computer e quindi parleremo del BASIC, cioe' del linguaggio che vi permette di colloquiare con il CBM 64, senza avere la pretesa di scrivere un trattato sul linguaggio.

Parleremo delle possibilita' musicali e delle grandissime capacita' grafiche, anche in questo caso senza scendere eccessivamente nel dettaglio un po' perche' questo richiederebbe una serie di volumi, di cui parte pubblicati e parte in preparazione, un po' perche' riteniamo che debba essere lasciato all'utente di realizzare quello che il CBM 64 puo' dare ed e' veramente molto.

Presto saranno disponibili altri libri su questo computer per aiutare alla scoperta ed alla conoscenza dell'INFORMATICA da parte di CHIUNQUE. Ed e' questo il vero significato del CBM 64.

Questo manuale contiene anche un' accurata mappa di memoria del Sistema Operativo perche' chi voglia approfondire questo interessantissimo settore trovi gli strumenti idonei almeno per iniziare il lavoro.

Abbiamo inoltre ritenuto utile mettere una serie di capitoli che si riferiscono ai FILES ed alla gestione almeno della periferica piu' diffusa: la cassetta, mentre ci siamo riservati una specifica trattazione dell'unita' a dischi e della stampante in apposito e separato volume. In fondo a questo volume troverete una scheda da riempire e da rinviare alla EVM-COMPUTER ne ha curato la messa a punto.

Rispedendola compilata riceverete in omaggio il nostro catalogo prodotti HARDWARE e SOFTWARE COMMODORE. Sarete quindi inseriti in un MAILING LIST e potrete ricevere continui aggiornamenti sul computer in vostro possesso.

Potrete inoltre ordinare un disco o una cassetta contenente i programmi riportati nel presente manuale.

Questo nostro lavoro, primo di una serie di manuali che riguardano questa macchina e' il frutto di una ricerca approfondita ed e' stato rivisto e corretto piu' volte in collaborazione con i lettori.

Tuttavia non e' certamente privo di errori e di imprecisioni che l'utente ci aiuterà a correggere come per gli altri nostri lavori.

Invitiamo quindi tutti a collaborare con la nostra redazione mentre da parte nostra ci impegniamo a rispondere a tutte le richieste che ci verranno fatte.

## CAPITOLO PRIMO

### LA TASTIERA

E' necessario familiarizzarsi con una delle parti piu' importanti del CBM 64: la tastiera.

Troverete che la tastiera e' simile a quella di una comune macchina da scrivere anche se per noi italiani, alcuni tasti risultano in altra posizione rispetto alle macchine da scrivere standard.

Ci sono inoltre alcuni tasti di colore beige sulla destra con le scritte F1....F8 che sono i tasti FUNZIONE. Ognuno di questi tasti, come vedremo in seguito, puo' essere programmato appunto per eseguire una funzione particolare.

Quanto segue e' una breve descrizione dei tasti principali presenti sul vostro CBM 64, le cui funzioni saranno viste piu' approfonditamente nel seguito del manuale.

### RETURN

Questo tasto serve ad informare il computer che le informazioni che appaiono sul video devono essere trasferite al suo interno. Normalmente questo tasto che corrisponde al RITORNO CARRELLO della macchina da scrivere viene utilizzato per dare qualsiasi informazione al computer, o per essere piu' precisi, consente il passaggio di un' informazione dal video, ( cioe' dalla memoria video) alla memoria interna del computer.

### SHIFT

Anche questo tasto opera come in una macchina da scrivere standard per le maiuscole e le minuscole. Come e' possibile vedere direttamente sui singoli tasti,

molti di loro sono in grado di trasmettere al computer non solo caratteri alfabetici o numerici, ma anche 2 lettere o simboli e due caratteri grafici.

All' accensione della macchina ci troviamo in quello che potremo definire come modo normale, (UPPER/LOWER CASE) la tastiera da come abilitati cioe', come disponibili, i caratteri superiori.

Premendo il tasto SHIFT ed entrando quindi nel modo shift (UPPER/CASE GRAPHIC) si visualizzano i caratteri grafici presenti sulla parte destra del tasto.

Nel caso dei tasti funzione saranno attivati le funzioni corrispondenti alla parte inferiore del tasto. In pratica quelle relative ai numeri pari.

Un altro modo per selezionare i caratteri maiuscoli o minuscoli e' quello di digitare questi comandi di cui vedremo in seguito sia il significato che le possibili applicazioni:

POKE 53272,20 per i maiuscoli

POKE 53272,22 per i minuscoli

oppure, e vedremo dopo il come ed il perche':

CHR\$ (14) per i minuscoli

CHR\$ (142) per i maiuscoli

## NOTA

Altro modo, molto semplice per passare dal maiuscolo al minuscolo e viceversa e quello di premere contemporaneamente il tasto SHIFT ed il tasto con il simbolo COMMODORE.

## EDITING DEL CBM 64

Per quanti paragoni si possano fare riteniamo che nessun computer abbia delle funzioni di EDITOR paragonabili a quelli della linea Commodore, e il CBM 64 non fa eccezione a questa regola.

L'alto livello di sofisticazione dei comandi relativi a questa funzione permette una facilità di programmazione e di correzione di cui ci si rende conto solo passando ad un altro tipo di computer.

Vediamone i tasti uno per uno.

### CRSR

Ci sono due tasti con questo simbolo che è l'abbreviazione della parola inglese CURSOR, uno con delle freccette rivolte in alto ed in basso e l'altro con le freccette rivolte a destra e sinistra. Infatti questi due tasti servono per muovere il quadratino lampeggiante o CURSORE, a destra e sinistra, in alto ed in basso sull'area di schermo.

Nel modo normale, cioè senza il tasto di SHIFT premuto, i due cursori si spostano rispettivamente in basso ed a destra.

Usando lo SHIFT i tasti CRSR si muovono nelle direzioni opposte, cioè in alto ed a sinistra dello schermo.

Sul CBM 64 questi tasti hanno inserita una speciale funzione chiamata REPEAT AUTOMATICO, per cui se si tiene premuto il tasto si ha una ripetizione automatica della funzione scelta. Si potrà muoversi cioè a destra o sinistra, in alto o in basso continuamente senza premere più volte il tasto relativo.

### INST/DEL

Attivando il tasto di INST/DEL il cursore si muoverà di uno spazio a sinistra cancellando (DEleting) il carattere scritto.

Se ci si trova nel mezzo di una linea, cioè fra due caratteri o gruppi di caratteri, allora il carattere a sinistra sarà cancellato e tutti i caratteri sulla destra si muoveranno di uno spazio a sinistra, per cui il primo carattere a destra del cursore andrà ad occupare lo spazio lasciato libero.

Nel modo SHIFT, cioè premendo questo tasto insieme con il tasto INST/DEL consente di inserire (INSErT) informazioni, dati o caratteri in una linea.

Se avete commesso un errore all'inizio di una linea, ad esempio lasciando la lettera di un nome, potrete usare i tasti CRSR per riposizionarvi nel punto giusto e poi lo INST/DEL + SHIFT per creare uno o più spazi dove inserire le correzioni.

Anche su questo tasto è attiva la funzione di REPEAT.

## CLR/HOME

Questo tasto serve a posizionare il cursore nella parte sinistra più in alto dello schermo, appunto la così detta posizione HOME, se usato in modo normale.

Usato con lo SHIFT cancellerà tutto lo schermo e invierà il cursore nella posizione HOME.

## RESTORE

Questo tasto serve a riportare il computer al normale stato in cui era prima che avvenissero cambiamenti dovuti a programmi o comandi diretti.

Per ovvii motivi di sicurezza questo tasto da solo non dà nessun risultato, ma deve essere azionato insieme al RUN/STOP.

## CTRL

Il tasto CTRL (ConTRoL) consente di fissare i colori sul vostro schermo e di manipolare altre funzioni

specializzate. Anche le funzioni relative a questo tasto saranno viste e approfondite in seguito.

## RUN/STOP

Normalmente premendo questo tasto si arresta l'esecuzione di un programma BASIC.

Usando il RUN/STOP con il tasto di SHIFT premuto si invia al computer il segnale di caricare un programma da cassetta. Infatti il CBM 64 risponderà con messaggio:

PRESS PLAY AND RECORD ON TAPE

e resterà in attesa che abbiate eseguito questa funzione.

## IL TASTO COMMODORE

Il tasto a sinistra in basso del CBM 64 ha un simbolo strano: è il simbolo della COMMODORE la casa che ha costruito il vostro computer.

Quando il computer viene acceso ci troviamo nel modo NORMALE o UPPER CASE/GRAPHIC nel quale le lettere sul video appaiono in maiuscolo. Come abbiamo detto in precedenza usando il tasto SHIFT potremo utilizzare i caratteri grafici sulla parte destra del singolo tasto.

Se si preme il tasto COMMODORE insieme allo SHIFT allora la visualizzazione sullo schermo cambierà da maiuscolo in minuscolo.

Invece se sarà premuto un qualsiasi tasto insieme al tasto COMMODORE sarà visualizzato il simbolo grafico sulla sinistra del tasto.

Per ripassare dal modo minuscolo nuovamente al maiuscolo premere nuovamente il tasto C= e il tasto SHIFT.

La seconda funzione di questo tasto è quella di selezionare uno degli otto colori del testo.

Attivando il tasto COMMODORE ed uno dei tasti con i numeri, il testo scritto da quel punto in poi apparirà

con il colore prescelto.

Vedremo in seguito un piu' preciso utilizzo di questo tasto in rapporto ai colori anche con qualche esempio.

## NOTA

Prima di procedere oltre nella lettura cercate di familiarizzarvi con l' uso della tastiera ed in particolare dei tasti citati o TASTI DI CONTROLLO.

Soprattutto, e lo ripeteremo spesso nel seguito, ricordarsi che a qualsiasi istruzione diretta o in modo programma e' necessario far seguire SEMPRE il RETURN.

E' bene ricordare inoltre che la maggior parte delle funzioni, descritte per l' utilizzazione, fino a questo punto, in modo diretto, possono essere eseguite in modo programma.



## CAPITOLO SECONDO

### GLI ELEMENTI FONDAMENTALI DEL BASIC

Il processo creativo di combinare diverse istruzioni per risolvere un particolare problema e' chiamato PROGRAMMAZIONE.

Non possiamo dare ovviamente in questo manuale tutte le ricette per risolvere i problemi che possono presentarsi ne' pretendere di fornire un corso completo di Basic, ma possiamo dare gli strumenti in modo da poter utilizzare il CBM 64 alla soluzione del problema stesso.

#### L'ISTRUZIONE PRINT

Il computer puo' manipolare un numero impressionante di dati ed eseguire a grande velocita' operazioni complesse, ma tutto cio' non avrebbe senso se il risultato del calcolo non potesse essere visualizzato in qualche modo. Per questo e per altri motivi che vedremo in seguito, la descrizione dei comandi BASIC iniziera' dall' istruzione PRINT, cioe' STAMPA, anche se per il momento ci riferiamo alla stampa su video o VISUALIZZAZIONE.

Sia in modo programma che nell' utilizzo in modo diretto l' istruzione PRINT puo' anche essere abbreviata con un punto interrogativo ? per cui avremo:

```
PRINT"ciao"
```

e' equivalente a:

```
?"ciao"
```

Questa appena vista e' una istruzione al calcolatore che fa si che esso visualizzi sullo schermo quanto e' stato posto fra virgolette o apici. Nel caso in oggetto una breve espressione di saluto.

L' utilizzo dell' istruzione PRINT in forma diretta fa sì che il CBM 64 possa essere usata come una SUPER-CALCOLATRICE.

Infatti il Basic esegue immediatamente il comando che è stato immesso dopo l' istruzione PRINT non appena viene premuto il tasto RETURN (ritorno carrello).

Ad esempio se si vuole che il CBM 64 addizioni due numeri scriveremo:

PRINT 5+3

e dopo aver premuto il RETURN sullo schermo comparirà il numero:

8

Ready.

che è appunto il risultato dell' operazione e con il cursore lampeggiante sotto e dove Ready sta per la frase:

SONO PRONTO AD ESEGUIRE UN' ALTRO LAVORO

Il segno + che abbiamo messo fra i due valori si chiama:

OPERATORE

è cioè un carattere che viene interpretato dal Basic come una funzione, in questo caso aritmetica, che deve essere eseguita.

OPERATORI

Il Basic permette di utilizzare i seguenti operatori singolarmente o in combinazione fra loro:

- + per la somma
- per la sottrazione
- \* per la moltiplicazione
- / per la divisione
- ↑ per l'elevazione a potenza

Con un' istruzione tipo:

```
PRINT 1024 * 8
```

stampa sullo schermo il risultato del prodotto 1024\*8  
cioe' 8192

Con il BASIC e' possibile stampare piu' di un valore per volta, cioe' con una singola istruzione PRINT possiamo scrivere sulla solita riga piu' dati utilizzando o dei separatori es. (,;) o delle istruzioni tipo TAB, SPC.

Un esempio puo' essere:

```
PRINT 1024*8,7+5,"risultati"
```

che fara' stampare il prodotto, la somma e la stringa contenuta fra virgolette spaziati di un certo numero di caratteri.

```
8192      12      risultati
```

Nei successivi capitoli saranno contenuti i dettagli sull' esatto formato di stampa, mentre qui si vuol far solo presente che un certo numero di dati possono essere stampati sia su una serie di linee successive che su una stessa linea.

Sebbene come abbiamo detto la rappresentazione fisica sullo schermo sia di 25 linee di 40 caratteri ciascuna, la stampa di linee fino ad 80 caratteri viene fatta eseguendo automaticamente un A CAPO dopo il quarantesimo carattere su ciascuna linea.

Il CBM 64 fa scorrere automaticamente la visualizzazione sullo schermo di una o di due linee quando raggiunge i mille caratteri che costituiscono nel modo normale, l' area disponibile dello schermo.

Il comando PRINT ha due forme principali. La prima permette la stampa di una singola variabile nello spazio ad essa destinato.

Se il dato viene presentato in questa forma, il campo viene stampato dalla posizione corrente sullo schermo e facendo seguire la stampa da un ritorno a capo.

Se i dati sono presentati nella forma :

PRINT A,B

allora il BASIC automaticamente, stampa A, poi dopo aver imposto una spaziatura di 10 caratteri, stampa B seguito da un ritorno carrello.

Es.

PRINT 5+5,6\*2

avremo

10-----12

Ready.

NOTA

Lo spazio tratteggiato sta ad indicare i 10 spazi.

Per impedire che il Basic effettui il ritorno a capo dopo la stampa si usera' il punto e virgola";".

L' istruzione

PRINT 3+2;8\*6;

avra' come risultato:

6 48

Ed il cursore viene posizionato alla fine della seconda

variabile. Nel nostro caso cioè' accanto a 48.

Se la variabile A (cioè' la prima) ha piu' di 7 caratteri, naturalmente come risultato, allora B viene stampato dopo una spaziatura di 20 caratteri se viene usata la forma:

PRINT A,B

Il Basic in stampa usa le seguenti regole.

Quando il dato da stampare e' una stringa cioè' un insieme di caratteri alfanumerici , allora non vi e' nessun carattere che preceda o segua tale stringa.

Se deve stampare invece un numero per prima cosa il Basic esamina la sua grandezza.

Se il numero e':

minore di 0,01

o

maggiore o = a 999999999,2

allora viene stampato usando la notazione scientifica.  
Es.

0,0034

verra' stampato come

3,4 E -03

mentre il numero

-1234567890,5

verra' stampato come

-1,2345678 E+09

Valgono cioè' le seguenti regole:

1)- Se il numero da stampare ha un valore compreso fra 0,01 e 999999999,2 vengono stampate le cifre piu' significative piu' il punto decimale se necessario.

2)- Gli zeri non significativi dopo il punto decimale non vengono stampati.

Il BASIC stampa inoltre un carattere bianco dopo un numero ( a meno che questi non venga stampato sotto forma di stringa ).

Per questo motivo i seguenti valori:

0.25            12.36000

saranno intesi e visualizzati come:

.25            12.36

#### NOTA

Per utilizzare al meglio le possibilita' del CBM 64 di comporre un testo sullo schermo, e' bene notare che a differenza di parecchi tipi di Basic quello della Commodore, fa si che quello che appare fra i campi di stampa e' sempre un carattere di salto ottenuto mediante lo spostamento a destra del cursore.

#### TERMINI E OPERATORI

I numeri sono sempre presentati sullo schermo in forma decimale sebbene il microprocessore 6510 inserito nel CBM 64 lavori in modo binario.

Quando il Basic riceve sia in modo diretto che in modo programma una intera linea interpreta i caratteri ricevuti e li divide in diverse classi.  
In particolare li dividera' in

## COMANDI

## FUNZIONI

## OPERATORI

Come abbiamo visto la frase PRINT e' un comando al CBM 64 di eseguire una determinata istruzione.

Una funzione puo' essere ad esempio una radice quadrata o una variabile o una funzione speciale.

Non appena si premera' sulla tastiera il tasto del PI GRECO sara' disponibile ed usabile in una qualsiasi espressione la costante 3.14159265.

Un operatore e' un carattere che viene interpretato dal computer come una funzione aritmetica che deve essere eseguita nel calcolare una espressione.

Il Basic permette di usare un vasto gruppo di operatori che sono stati visti nel precedente capitolo.

Il segno + fa si che due numeri siano addizionati. La precisione e' limitata a 9 cifre significative, per cui eseguendo un' addizione con piu' di nove cifre avremo come risultato un numero con notazione esponenziale, a meno di non ricorrere a particolari accorgimenti.

Es.

```
PRINT 2 + 2
```

```
4
```

```
Ready.
```

Il segno - sottrae il valore che si trova alla destra del segno dal valore che si trova alla sua sinistra.

Es.

```
PRINT 4 - 6
```

```
-2
```

Ready.

L' asterisco e' il simbolo usato per la moltiplicazione. Il valore che si trova alla destra del segno e' moltiplicato per il valore che si trova alla sua sinistra. Viene tenuto conto dei segni come in una comune calcolatrice.

Es.

PRINT 7 \* 9

63

Ready.

Per la divisione si usa la barra rivolta a destra. Tutti i numeri che si trovano alla sinistra della barra vengono divisi per il numero o l'espressione che si trova alla destra del segno.

Es.

PRINT 12 / 2

6

Ready.

La freccia rivolta verso l' alto e' il simbolo usato per l' elevamento a potenza. Alla sinistra di tale segno si trova la base ed a destra l' esponente.

Es.

PRINT 4 é 2

16

Ready.

## PARENTESI

Le parentesi chiuse ed aperte fanno si che i valori al loro interno siano una sola espressione.



Il loro uso e' quello comune nella matematica. Da ricordare che nel Basic si usa un solo tipo di parentesi, quelle tonde ().

Ci devono essere tante parentesi aperte quante parentesi chiuse e le operazioni avvengono a partire dalla coppia piu' interna di parentesi.

Come in matematica cosi' nel Basic esiste un ordine gerarchico nella esecuzione delle operazioni.

L' ordine gerarchico degli operatori e' il seguente :

1)-Per prima viene fatta la moltiplicazione seguita dalla divisione, poi dalla somma ed infine dalla sottrazione.

2)-Le espressioni all' interno delle parentesi sono calcolate per prime partendo dalla prima coppia di parentesi, cioe' da quella piu' interna.

Vediamo un po' di esempi commentati.

Es.

? (4+8)/2

6

Ready.

In questo caso per prima e' sta eseguita la somma all' interno delle parentesi e successivamente il risultato diviso per 2.

Nel caso non si fossero utilizzate le parentesi si avrebbe avuto un risultato diverso.

Infatti:

? 4+8/2

8

Ready.

Perche' in questo caso prima sarebbe stata eseguita la divisione e POI la somma con il risultato di 8 invece di 6.

## CAPITOLO TERZO

### LA PROGRAMMAZIONE

Con l'evolversi delle tecnologie sono stati modificati certi concetti operativi il che ha permesso di affrontare i problemi della programmazione sotto un' ottica diversa rendendo piu' accessibile e soprattutto piu' funzionale l' avvicinamento all' informatica.

Il compito essenziale del programmatore e' rimasto comunque quello di esprimere sotto forma di algoritmo il problema in esame e tradurlo in linguaggio, in questo caso il BASIC, sotto forma di programma.  
Il concetto di algoritmo, che in matematica puo' definirsi come:

"Quell' insieme di regole atte a definire un procedimento di calcolo al fine di ottenere un determinato risultato partendo da determinati dati iniziali"

Assume in informatica un significato piu' specifico."

### ALGORITMO INFORMatico

In informatica l' algoritmo deve soddisfare le seguenti caratteristiche:

- 1) Essere finito e terminare dopo un numero determinato di operazioni.
- 2) Essere definito e preciso. Ogni istruzione deve essere definita incontestabilmente.
- 3) Essere precisato il campo di applicazione dei dati di entrata.

4) Possedere almeno un dato in uscita.

5) Essere possibile. Devono cioè poter essere effettuate tutte le operazioni in modo esatto e finito nel tempo, da un uomo che impieghi i mezzi manuali.

Enunciato il problema, il primo compito del programmatore è quello di costruire un algoritmo che soddisfi le caratteristiche elencate, procedendo quindi alla stesura di un programma che conduca ad un risultato.

Lo schema sarà quindi:

PROBLEMA

ALGORITMO

PROGRAMMA

RISULTATO

Teniamo presente che:

A)-Non per tutti i problemi è possibile costruire un algoritmo, ma ciò non significa che il problema non abbia una soluzione.

B)-L' algoritmo che risolve il problema può non essere unico.

C)-Il fatto che esista una soluzione per un problema, in uno o più casi particolari, non significa che il problema sia sempre risolubile.

D)-Il fatto che un problema non sia risolubile, non significa che non ha soluzioni, ma solo che non esiste o non siamo in grado di costruire, un algoritmo capace di dare una soluzione in tutti i casi in cui essa esista.

Tradizionalmente, in informatica, viene usato un sistema grafico per la stesura dell' algoritmo. Questo grafico che prende il nome di schema a blocchi o FLOW CHART, consiste nella descrizione delle diverse operazioni, in forma di schema simbolico descrivente i diversi ordini e le diverse condizioni trattate dall' algoritmo stesso.

L' utilizzo di linguaggi come il Basic, anche se non completamente strutturati, o almeno spesso utilizzati in modo non strutturato, ha ridotto notevolmente l' importanza dello schema a blocchi, riservando l' uso per casi complessi.

Molto spesso, infatti, e' sufficiente scrivere con esattezza cio' che si vuol fare, per riuscire a stendere il programma.

Visto poi che il Basic e' un linguaggio interprete, il programma, se si dispone di un calcolatore, puo' venire direttamente inserito e controllato passo passo.

E' buon sistema comunque, studiare a fondo il problema e stendere una traccia scritta di cio' che si vuol fare, prima di passare alla stesura del programma definitivo per il CBM 64. Questo perche', in caso di errori, la macchina segnala solo quelli sintattici o di procedura e non quelli di logica.

Questi ultimi devono essere individuati dal programmatore, analizzando i risultati eventualmente ottenuti.

Le situazioni in cui ci si imbatte programmando sono riassumibili in tre punti essenziali:

## I) SEQUENZA

Le operazioni devono svolgersi una dopo l' altra, secondo uno schema preordinato.

## II) DIRAMAZIONE

Si deve operare una scelta in funzione del risultato ottenuto.

## III) ITERAZIONE (ciclo)

Si deve ripetere un determinato numero di volte una sequenza di operazioni.

I concetti qui esposti rappresentano una sintetizzazione di quelli che sono i problemi e le tecniche della programmazione.

Lo studio del Basic, almeno come e' riportato in questo manuale, servira' a farvi entrare in possesso degli strumenti idonei a risolvere il vostro problema.

## LE RIGHE DI UN PROGRAMMA

Fino a questo momento e' stata usata una tecnica di programmazione che fa si che il CBM 64 risponda direttamente al comando dato.

Cioe' il Basic esegue immediatamente il comando dato dopo che e' stato effettuato un ritorno carrello.

In questa maniera, come abbiamo detto, il CBM 64 viene usato come un supercalcolatore.

Ad esempio se si vuole che il CBM 64 esegua una addizione scriveremo :

? 2+8

ed avremo come risultato:

10

Ready.

Il CBM 64 cioe' obbedisce istantaneamente a qualsiasi istruzione assegnatagli attraverso la tastiera tranne quando e' in esecuzione un programma Basic.

Il modo diretto e' valido e quindi utilizzabile quando si vogliono correggere programmi.

Tuttavia tranne questi casi per utilizzare in pieno le capacita del computer e' necessario scrivere dei veri

programmi Basic.

La differenza piu' evidente in questo modo di operare sul CBM 64 e' che con un programma parecchie istruzioni possono essere raggruppate assieme e il Basic le eseguirà in successione logica cioe' una dietro l'altra.

Le operazioni necessarie per scrivere un programma sono molto semplici.

Qualsiasi istruzione si desideri che venga trattata dal Basic del CBM 64 come un' istruzione di programma, deve essere preceduta da un numero di linea:

### nn Linea di programma

Un numero di linea puo' essere un qualsiasi numero compreso fra 0 e 63999.

Tuttavia e' buona abitudine, quando si sviluppi un programma, assegnare i numeri di linea incrementandoli di 10 o di 100. Usare cioe' 10,20,30,40, ecc. invece di 1,2,3,4 ecc.

Questo vi permettera' di avere uno spazio fra una istruzione e l' altra in modo da poter aggiungere successivamente delle nuove linee ed effettuare agevolmente correzioni del vostro programma.

Infatti il linguaggio e' stato messo a punto in modo tale che dovendo per esempio inserire fra le linee di istruzione 10 e 20 la linea 15 sia sufficiente digitare:

### 15 comandi o funzioni

e questa nuova linea viene AUTOMATICAMENTE immessa fra la 10 e la 20 senza nessun bisogno di intervento manuale.

Al RUN, cioe' dopo aver dato il comando RUN, il Basic eseguirà ciascuna istruzione in ordine crescente di numero di linea (tranne i salti come vedremo in seguito).

Si supponga di voler visualizzare la frase CBM 64 in forma verticale anziche' in orizzontale.

Questa operazione puo' essere facilmente eseguita con un programma, mentre non e' altrettanto semplice attraverso comandi diretti.

Per quanto detto, sara' sufficiente inserire in ciascuna linea di programma, che supponiamo parta dalla linea 10, una serie di comandi PRINT come nell' esempio seguente:

```
10?"C"  
20?"B"  
30?"M"  
40?"6"  
50?"4"
```

Si ricordi che come nel caso dei comandi diretti, la fine di ogni linea di programma deve terminare con un RETURN, in modo da far passare l' istruzione dalla memoria di schermo a quella del Sistema Operativo.

Il programma e' ora in memoria. Per eseguirlo bastera' digitare il comando diretto RUN e sara' visualizzata:

```
C  
B  
M  
6  
4
```

e cio' naturalmente fino a quando non si spenga il computer.

Si noti che non esiste nessuno spazio fra la CBM e la cifra 64. Una delle ragioni per cui si sono usati dei numeri di linea spaziati di 10 e' che ora si puo' inserire una linea di correzione fra la linea 30 e 40. Inseriamo ora:

```
35?
```

Dopo il ritorno carrello eseguiamo il LIST e notiamo che questa nuova linea e' stata inserita al punto giusto. Facendo girare ora il programma otterremo la stampa:

C  
B  
M

6  
4

Abbiamo cioe' inserito un' istruzione PRINT che visualizzera' quindi una riga vuota fra le lettere CBM e la cifra 64.

Questo piccolo esempio dimostra l' uso che si puo' fare dei numeri di linea e la possibilita' di inserire delle linee per eseguire correzioni.

## TECNICHE DI PROGRAMMAZIONE

### L' ISTRUZIONE INPUT

A questo punto sara' necessario iniziare ad esaminare la possibilita' di immettere dei dati che non siano solo stampati direttamente sullo schermo, come abbiamo visto fino a questo momento, ma che possano essere anche elaborati.

La sintassi dell' istruzione per l' ingresso di dati e' fornita dal comando INPUT seguita da una lista di variabili.

Tali variabili verranno caricate dai valori assegnati attraverso la tastiera secondo la sequenza specificata nella lista.



Es.

10 INPUT A,B

Verra' eseguita nel modo seguente.

Quando il programma girando incontrera' questa istruzione, che non necessariamente deve essere la prima, il Basic stampera' un punto interrogativo sullo schermo e fara' lampeggiare il cursore in attesa dei dati di ingresso.

Ora dovremo immettere accanto al punto interrogativo i dati necessari al funzionamento del programma e premere il tasto RETURN per un ritorno carrello. In questo caso si tratta di valori o espressioni numeriche.

Dopo che questo ritorno carrello viene ricevuto dal computer i dati sono consegnati al Basic un carattere per volta.

Il Basic, utilizzando i suoi Buffers di ingresso e le sue routines di traduzione del Sistema Operativo interpreta quindi questi dati.

Nel caso si debba dare piu' di una informazione o dato come nell' esempio precedente, questi dati devono essere separati fra loro da una virgola.

Quando si cerchi di dare piu' dati di quelli richiesti dal programma( proviamo, sempre con riferimento all' esempio precedente a cercare di immettere 3 o piu' dati invece dei due richiesti) il CBM 64 rispondera' con un messaggio:

?EXTRA IGNORED

e considerera' validi solo tanti dati quanti sono necessari a soddisfare la lista d' ingresso.

Notiamo che parliamo di dati anziche' di cifre o di caratteri perche' in questo caso si possono dare come valore ANCHE delle espressioni purché naturalmente numeriche.

Cioe' in risposta all' istruzione INPUT A, potremo dare

per esempio:

$(12+3)*14-(11.3*4.5)$

Al contrario quando non vengono date informazioni in numero sufficiente il CBM 64 rispondera' con un messaggio:

??

accanto al quale lampeggera' il cursore in attesa dei dati che mancano.

Se come nell' esempio fatto avremo una richiesta di dati numerici e verranno immessi dei dati alfanumerici, cioe' dei caratteri o dei simboli, il CBM 64 rispondera' con un messaggio:

?REDO FROM START

#### NOTA

Durante la fase di attesa dati il tasto di RUN/STOP viene disabilitato, cioe' non puo' essere usato per interrompere l' esecuzione del programma.

Per uscire da questa situazione, qualora si desideri dovremo premere contemporaneamente il tasto di RUN/STOP e di RESTORE.

Vediamo a questo punto un semplice esempio di programma.

```
10 INPUT A,B
20 C=A*B
30 PRINT C
```

Questo programma consente di risolvere il problema della moltiplicazione o piu' precisamente :

E' l'ALGORITMO necessario a risolvere il problema della moltiplicazione.

Naturalmente l' algoritmo scelto in questo caso e' molto semplice, ma serve per avvicinarsi alla programmazione. Inoltre partendo dall' esempio appena visto creeremo un programma un po' piu' complesso esaminando passo passo le relative istruzioni.

Se proviamo ad inserire nel computer questo programma e successivamente a digitare il comando RUN seguito da un RETURN, vedremo sullo schermo comparire un punto interrogativo (?).

Il Basic chiederà' cioè' il valore che si desidera assegnare alla variabile A.

Ammettiamo di dare a questa variabile il valore 4, digitando appunto un 4 seguito da un RETURN.

Comparirà' allora la cifra ed il cursore si posizionerà' immediatamente sulla seconda riga dove appariranno due punti interrogativi.

Ci viene cioè' richiesto di inserire il valore della seconda variabile cioè' di B.

Mettiamo di assegnare a B il valore 8.

Dopo il solito RETURN verrà' visualizzato :

32

Ready.

che e' appunto il risultato della moltiplicazione seguita dalla parola Ready. che sta a significare che il computer e' pronto a fare qualche altra cosa.

Un programma, mentre può' essere chiaro al programmatore o comunque a colui che l' ha scritto non e' detto che lo sia anche all' utente. Per questo sarà' necessario immettere dei chiarimenti ad uso dell' utilizzatore.

Introduciamo `a questo punto un ampliamento dell' istruzione INPUT.

Infatti l'istruzione INPUT permette di eseguire delle funzioni speciali quali quelle di indicare all' utente che tipo di caratteri in ingresso siano desiderati.

Un' espressione letterale racchiusa fra virgolette che segua il comando INPUT viene stampata prima che compaia il punto interrogativo.

Ad esempio possiamo sostituire la linea 10 del programma precedente con:

```
10 INPUT"Valori da moltiplicare";A,B
```

Dopo il RUN verra' visualizzato:

```
Valori da moltiplicare?
```

ed il cursore restera' lampeggiante dopo il ?

Nello stesso modo potremo modificare la linea 30 con:

```
30 PRINT"Il risultato e' ";C
```

Facendo girare avremo:

```
Valori da moltiplicare? 4
```

```
??8
```

```
Il risultato e' 32
```

```
Ready.
```

Se le istruzioni sono state seguite correttamente non ci sono difficolta' a far funzionare questo piccolo programma, tuttavia noterete che lo schermo resta SPORCO, cioe' scorre (SCROLL) mano a mano che facciamo girare il programma, mantenendo pero' le scritte, le correzioni e tutta una serie di informazioni che possono disturbare.

Possiamo quindi desiderare che tutte le volte che si fa girare il programma, cioe' che si da un comando di RUN, lo schermo venga pulito per accettare i dati.

Proviamo quindi ad aggiungere una linea 9 con l'istruzione:

```
9 INPUT"SHIFT+CLR/HOME "
```

Fra virgolette otterremo un simbolo che deriva appunto dal fatto di premere insieme i tasti SHIFT e CLR/HOME.

Come ricordiamo dal capitolo dedicato all' uso della tastiera, mentre il tasto CLR/HOME serve per riposizionare in alto a sinistra il cursore, se premuto insieme allo shift esegue ANCHE la pulizia dello schermo. Nell' istruzione alla riga 9 non abbiamo fatto altro che inserire questa funzione, vista fino ad ora in modo diretto in MODO PROGRAMMA.

Volendo migliorare ancora possiamo inserire altre istruzioni ad esempio per far in modo che fra i dati di ingresso e la stampa del risultato venga lasciato un po' di spazio.

Scriviamo quindi una riga di programma nuova:

```
25 PRINT:PRINT:PRINT
```

Il Basic del CBM 64 automaticamente posizionera' questa nuova serie di istruzioni fra le righe 20 e 30.

Il risultato sara' che fra i dati di ingresso e la stampa dei risultati saranno saltate 3 righe.

Infatti un comando di PRINT a vuoto, cioe' senza nessun valore fa spaziare di una intera riga.

Notiamo inoltre che per separare una istruzioe da l' altra nell' ambito della stessa riga e' necessario usare i due punti (:).

Proviamo ad aggiungere ancora qualche istruzione.

Si voglia ripetere piu' volte la moltiplicazione senza digitare sempre il comando RUN.

Per far questo, cioe' per ripetere il programma tutte le volte che si desidera, aggiungiamo una linea 40 con l' ordine di tornare indietro.

```
.....
```

```
40 GOTO 10
```

Questo nuovo comando indicherà al computer una volta che il risultato è stato visualizzato, di tornare alla riga 10 e di riproporre una nuova moltiplicazione

NOTA

Non è possibile dare un'istruzione:

40 GOTO 9

perché il programma funzionerebbe così:

- a) Chiede i dati
- b) Stampa i risultati
- c) Torna a chiedere i dati

però l'esecuzione delle fasi b) e c) sarebbe così veloce che in pratica non si avrebbe il tempo materiale di leggere i risultati sul video.

Non dimentichiamo infatti che la linea 9 contiene la pulizia dello schermo.

NOTA

Anche questa nuova istruzione GOTO, come il comando LIST visto in precedenza sarà approfondita in seguito.

## CAPITOLO QUARTO

### FUNZIONI MATEMATICHE

Vi e' tutta una serie di funzioni matematiche, trigonometriche e di altro tipo nel Basic del CBM 64 caratteristiche sia di questo computer che di questo linguaggio. Vediamole una per una prima di passare ai comandi della programmazione e quindi a problemi piu' complessi.

#### NUMERI

Il numero piu' grande che il Basic del CBM 64 puo' manipolare e':

+ o - 1.7014118 E+38

Qualsiasi tentativo di usare un numero piu' grande dara' luogo alla stampa di un errore:

? OVERFLOW

Il numero piu' piccolo che si puo' usare, cioe' che puo' essere distinti dallo ZERO e':

2.93873588 E-39

Anche in questo caso il tentativo di utilizzare un numero inferiore dara' luogo ad una segnalazione di errore.

#### ABS

La funzione ABS(x) permette di calcolare il valore assoluto di un numero.

La funzione calcolerà il valore dell'espressione contenuta fra parentesi come un numero positivo. Non vi è nessuna perdita di precisione.

Es.

? ABS (-190)

190

Ready.

? ABS (-12.2\*3)

36.6

Ready.

## INT

La funzione INT(X) arrotonda per difetto un qualsiasi numero all'intero più vicino.

Es.

?INT (.23)

0

?INT (-2.5)

-3

?INT (1.79)

1

## NOTA

La funzione INT non introduce nessuna perdita di precisione al di là del troncamento della parte decimale. Tuttavia piccole imprecisioni nell'argomento della funzione possono dare luogo a seri problemi. Ad esempio il numero 4 può essere memorizzato dal Basic



come 3.999999999.

Quando cio' avviene e sia usato come argomento della funzione INT il risultato sara' 3 anziche' 4.

## SGN

La funzione SGN(X) da come risultato 1 se il segno del suo argomento e' positivo, 0 se l' argomento e' 0 e -1 se il segno dell' argomento e' negativo.

Es.

?SGN (-96)

-1

?SGN (63.2)

1

?SGN (0)

0

## SQR

La funzione SQR(X) calcola la radice quadrata di qualsiasi numero maggiore di ZERO contenuto nell' argomento.

Se come argomento viene usato un numero minore di ZERO il risultato e' la stampa di un messaggio di errore:

?ILLEGAL QUANTITY ERROR

Es.

? `SQR(16)

4

Ready.

Non disponiamo, come funzione diretta che di SQR, ma volendo calcolare radici di ordine superiore ricordiamo che la radice n-sima di X equivale a:

$$X \acute{e} (1/n)$$

Es.

Radice cubica di 81

$$81 \acute{e} (1/3)$$

NOTA

Le funzioni seguenti lavorano sulla base del numero naturale di Nepero e che si ricorda e':

$$2.71828183$$

EXP

Il parametro di tale funzione definisce la potenza alla quale la base viene innalzata. E' cioe' l' esponente da dare al risultato per ottenere la base ( il Cologaritmo).Il risultato e' l' elevamento a potenza di e per il numero dato.

I limiti di tale parametro sono :

$$88.02969189$$

Quando per argomento della EXP(X) si usa un numero che in valore assoluto sia maggiore del limite citato si avra' un errore di ? OVERFLOW ERROR.

LOG

La funzione LOG(X) calcola il logaritmo Naturale o Neperiano dell' argomento.

Es.

? LOG (562)

6.33150185

Per calcolare il logaritmo in base 10 e' sufficiente dividere il logaritmo in base E per il logaritmo in base E del numero 10. Cioe' il logaritmo decimale di X e' uguale al logaritmo naturale di X fratto il logaritmo naturale di 10.

$\log \text{ decimale di } X = \text{LOG}(X)/\text{LOG}(10)$

Es.

?LOG(1000)/LOG(10)

3

Se il valore dell' argomento e' 0 (ZERO) oppure un valore negativo avremo come risposta un messaggio di :?ILLEGAL QUANTITY ERROR.

ATN

Questa funzione restituisce l' arcotangente del numero. Il risultato e' l' angolo in radianti, di cui la tangente e' il numero dato.

Il risultato sara' sempre compreso nell' intervallo da PIGRECO/2 a + PIGRECO/2.

Es.

```
10 INPUT X:REM VALORE IN GRADI
20 Y= ATN (X)
30 REM CONVERSIONE IN GRADI
40 Z=Y*180/PIGRECO
50 PRINT Z: REM VALORE IN GRADI
```

COS

Questa funzione calcola il coseno di un numero, dove il

numero e' un angolo in radianti.

Es.

```
10 INPUT X:REM VALORE IN GRADI
20 REM CONVERSIONE GRADI IN RADIANTI
30 Y=X*PIGRECO/180
40 Z=COS(Y)
50 PRINT Z
```

## SIN

Questa funzione da il seno dell' argomento, in radianti.  
Ricordiamo che il coseno di un numero  $\cos(X)$  e' eguale a  $\sin(X + \text{PIGRECO}/2)$

Es.

```
10 INPUT A:REM IN RADIANTI
20 PRINT COS(A),SIN(A+PIGRECO/2)
```

## TAN

Riporta la tangente del valore del numero o dell' espressione numerica, in radianti.

Es.

```
10 X=.785398163 : Y=TAN(X):PRINT Y
```

il risultato sara':

1

## FUNZIONI NON MATEMATICHE

### PEEK

La funzione PEEK(X) permette all' utente di ispezionare qualsiasi locazione di memoria del CBM 64.

L' argomento della funzione PEEK e' l' indirizzo di memoria espresso in decimale ed il risultato e' un numero

decimale compreso fra 0 e 255.

L'argomento della funzione PEEK puo' andare da 0 a 65535. Se si tenta di leggere una locazione minore di 0 o maggiore di 65535 avremo un ?ILLEGAL QUANTITY ERROR.

Es.

?PEEK(50)

23

che e' appunto il contenuto della posizione di memoria 50 espresso in decimale.

Es.

?PEEK(53280) AND 15

Che serve per leggere il valore corrispondente al colore del bordo.

## POKE

In effetti POKE non e' una funzione ma un comando. Esso permette di caricare un valore in una determinata locazione di memoria secondo i parametri specificati.

La sintassi del comando e:

POKE X,Y

dove X sta ad indicare la locazione di memoria nella quale deve essere caricato il valore decimale Y.

Il primo parametro deve avere un valore fra 0 e 65535 mentre il secondo puo' andare da 0 a 255.

Alcune locazioni di memoria (ROM) non possono essere cambiate, mentre per altre e' bene procedere con notevole cautela.

Anche per POKE valgono le limitazioni viste per il precedente comando non potendo scrivere un valore maggiore di 255 e minore di 0 in una locazione minore di 0 o maggiore di 65535.

Es.

POKE 1024,1

che fara' visualizzare la lettera A nella prima locazione in alto a sinistra.

## USR

Serve per eseguire un salto ad una subroutine in linguaggio macchina il cui indirizzo di partenza e' dato dal contenuto delle locazioni di memoria 785-786. Questo indirizzo viene fissato, prima di far entrare in funzione USR, utilizzando dei comandi POKE.

## FRE

La funzione FRE(X) informa sul quantitativo di memoria disponibile.

Malgrado essa sia una vera e propria funzione in quanto puo' essere usata in MODO PROGRAMMA, essa viene normalmente usata in modo diretto nella forma:

? FRE (0)

Questa funzione da luogo ad una azione Basic che viene chiamata:

## GARBAGE COLLECTION

che significa letteralmente:

## RACCOLTA DELLE IMMONDIZIE

Tale azione consiste praticamente nel riunire tutti i Bytes liberi della memoria RAM in un blocco di grosse dimensioni e quindi di visualizzare il totale di questi Bytes in risposta alla domanda di PRINT.

## NOTA

La funzione FRE e' tipica di tutte le versioni del Basic. Tuttavia nel CBM 64 si possono ottenere in risposta dei

valori negativi e quindi apparentemente assurdi.  
Si suggerisce quindi di usare direttamente o in programma la seguente formula invece della funzione diretta:

FR=FRE(0):IF FR THEN FR=FR+65536:?FR

CLR

Questo comando rende nuovamente disponibile la memoria RAM che era stata usata in precedenza ma che non e' piu' necessario tenere occupata.

Il programma BASIC presente in memoria resta inalterato, ma tutte le variabili, le matrici, gli indirizzi di GOSUB, i cicli di FOR...NEXT, i valori presenti nelle funzioni definite dall' utente ( cioe' quelli usati con il comando DEF FN che vedremo in seguito) oltre al contenuto dei files, sono cancellati dalla memoria e questo spazio viene reso disponibile per nuovi valori di variabili, ecc.

Si tratta di un' azione spesso necessaria durante il trattamento di grossi quantitativi di dati in particolare provenienti da periferiche.

Questo comando puo' essere usato sia in modo diretto sia in programma.

Es.

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

facendo girare, cioe' dando RUN, avremo come risultato:

0

NOTA

E' necessario fare molta attenzione ad utilizzare questa istruzione con files aperti su cassetta o su disco.

## FUNZIONI DI STAMPA

### POS

Riporta l' attuale posizione del cursore che e' in un' intervallo compreso fra 0 (cioe' il carattere piu' a sinistra) fino a 79 su una linea LOGICA di schermo di 80 caratteri.

Poiche' il CBM64 ha uno schermo di 40 colonne ogni posizione del cursore da 40 a 79 si riferira' alla seconda linea di schermo. Ecco perche' abbiamo parlato di linee LOGICHE.

Il formato del comando e':

POS(data)

Data e' un argomento convenzionale che puo' essere quindi un qualsiasi valore.

Come abbiamo detto POS restituisce la posizione corrente del cursore. Se non e' visualizzato nessun cursore, viene riportata l' attuale posizione del carattere entro una linea di programma o una variabile stringa.

La posizione del carattere va da 0 alla posizione piu' a sinistra del carattere stesso.

Es.

?POS(1)

all' inizio della linea riportera' un valore 0

### SPC

Questa funzione di formattazione stampa il numero di salti specificati nell' argomento di SPC(X).

Il campo di valori consentiti e' da 0 a 255.

Notare che la funzione SPC con argomento 0 da luogo a 256 salti

Proviamo a far girare un semplice programma:

10 INPUT A,B



```
20 C= A*B
30 PRINT SPC(4) C
```

Osserviamo tuttavia che la stampa avviene in effetti non a 4 ma a 5 spazi dalla sinistra dello schermo perche' il Basic tiene conto del segno.

## TAB

La funzione TAB(arg) muove il cursore verso destra alla specificata colonna contenuta nell' argomento della funzione.

Il valore nell' argomento deve andare da 0 a 255.

Questa funzione deve essere usata solo con PRINT e, non come la precedente, anche con PRINT# per le periferiche, perche' non avra' effetto.

## NOTA IMPORTANTE

La differenza fra SPC e TAB e' che con TAB ci si sposta dal bordo dello schermo di tanti spazi quanti sono quelli dell' argomento della funzione stessa:

Es.

```
10 INPUT A, B
20 PRINT TAB(10)A,TAB(20)B
```

Facendo girare questo programma, il valore dato ad A sara' stampato a partire dalla decima colonna ed il valore di B a partire dalla ventesima colonna di schermo. Mentre invece con SPC:

Es.

```
10 INPUT A, B
20 PRINT SPC(10)A,SPC(20)B
```

il valore dato ad A sara' stampato a 10 spazi dal bordo, MA il valore di B sara' stampato a 20 spazi non dal bordo, ma dal valore A.

Consigliamo di effettuare semplici esercizi per familiarizzarsi con queste funzioni.

## L' OROLOGIO INTERNO

La funzione TI\$ ed il valore TI sono due modi di comunicare con l' orologio in tempo reale che si trova all' interno del CBM 64.

TI\$ puo' essere espresso anche come TIME\$. Per fissare l' orologio interno usare il seguente formato:

TI\$="hhmmss"

dove:

hh= le ore tra 0 e 23

mm= i minuti tra 0 e 59

ss= i secondi tra 0 e 59

Alla mezzanotte o quando il ciclo si e' concluso TI\$ va naturalmente a 0

Per fissare l' orologio digiteremo per esempio:

TI\$="100150"

Mentre per visualizzare che ore sono eseguiremo in modo diretto o in programma:

?TI\$ o TIME\$

ed, avremo ad esempio:

121000

Il CBM 64 tiene conto del tempo in "JIFFIES". Un JIFFIES corrisponde ad un sessantesimo di secondo.

TI o TIME e' una variabile riservata che si incrementa automaticamente ogni sessantesimo di secondo. TI e' posta a 0 all' accensione e ritorna a 0 dopo 51.839.999 cicli, cioe' dopo esattamente 24 ore come del resto TI\$.

Questi valori sono importanti sia con riferimento alla funzione RND sia per calcolare i tempi di esecuzione di un programma.

Esempio.

Il seguente piccolo programma ci mostra un esempio di

rilevamento della velocita' di esecuzione di semplici routines.

Provate a cambiare i valori delle variabili nei cicli di FOR oppure a modificare le operazioni delle linee 30 e 70.

```
10 PRINT"TEST":PRINT
15 A=TI
20 FORI=32 TO 127
30 PRINT CHR$(I);
40 NEXT I
50 FORJ=161 TO 255
60 PRINT CHR$(J);
70 NEXT J
75 B=TI
80 PRINT"FINE":PRINT:PRINT
90 PRINT:PRINT"TI = ";B-A
```

#### NOTA

Ricordiamo che durante l' uso del registratore a causa dell' intervento dell' INTERRUPT, il valore restituito da TI\$ non e' da prendersi in considerazione come valido.

#### FUNZIONI DEFINIBILI DALL' UTENTE

Fino a questo punto sono state spiegate le funzioni intrinseche del Basic.

Nel campo della matematica si usano pero' molte piu' funzioni, specialmente trigonometriche.

Si potrebbe pensare di scrivere un programmino per approssimare in linea certe funzioni, ma cio' e' veramente noioso e inoltre dal punto di vista della documentazione una espressione per quanto semplice potrebbe divenire non facile a capirsi.

Nel Basic per fortuna esiste la possibilita' di definire funzioni in termini di altre funzioni.

Una funzione viene definita in una istruzione DEF la cui forma e':

DEF FN nome della variabile(argomento)= espressione.

E' necessario spendere qualche parola per l'utilita' che questa funzione puo' dare e per la non facilissima comprensione nel suo uso.

La funzione e' specificata da una espressione che puo' essere una espressione aritmetica, contenere una qualsiasi combinazione di costanti numeriche, variabili e/o operatori.

L' argomento e' una variabile cosiddetta DUMMY, cioe' di comodo che appare in una espressione.

L' argomento e' inoltre la sola variabile che puo' essere specificata quando ci si riferisce al nome della variabile.

Vediamo degli esempi di utilizzo prima di procedere oltre.

Es.

```
10DEF FNC(R)= pg*R*2
```

Si definisce cioe' una funzione che calcola la circonferenza di un cerchio, necessita di un solo argomento R e restituisce un singolo valore numerico, appunto la circonferenza del cerchio.

```
10DEF FNA (X)=X+7
```

```
20DEF FNAA (X)=Y*Z
```

```
30DEF FNA9 (Q)=INT(RND(1)*Q+1)
```

Come la precedente, anche questa funzione puo' essere richiamata piu' tardi nel programma usando il nome della funzione con una variabile in parentesi.

Questo nome di funzione e' quindi usato come una qualsiasi altra variabile ed il suo valore e' calcolato automaticamente.

Esempio di utilizzo:

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

L' intero comando DEF FN non deve superare le 80 colonne, tuttavia una funzione preventivamente definita puo' essere inclusa in una espressione, in modo tale che l' utente in pratica puo' definire una funzione molto piu' complessa.

## CAPITOLO QUINTO

### IL COMANDO LIST

Il comando LIST ha diverse opzioni che aiutano la correzione dei programmi.

Questo comando ha la funzione di rintracciare i programmi e di visualizzarne una riga dopo l'altra sullo schermo.

Usato senza parametri LIST parte dal primo numero di riga di istruzione del programma presente in memoria e lista tutte le righe fino all'ultima.

Si abbia per esempio il seguente programma:

```
10 INPUT A
20 INPUT B
30 INPUT C
40 PRINT A + B
50 PRINT A * B
60 PRINT C * A
70 PRINT A - C
```

E' possibile listare la singola riga:

Es.

```
LIST 20
```

```
20 INPUT B
```

fara' cioe' apparire sullo schermo solo il contenuto della linea 20.

```
LIST 20-50
```

```
20 INPUT B
30 INPUT C
40 PRINT A + B
50 PRINT A * B
```

cioe' listera' tutte le linee dalla linea 20 alla 50

inclusa.

LIST-50

```
10 INPUT A
20 INPUT B
30 INPUT C
40 PRINT A + B
50 PRINT A * B
```

listera' tutte le linee i cui numeri sono compresi fra il primo presente in memoria ( ricordiamo che un programma puo' iniziare anche dalla linea 0 ) e la linea 50 inclusa.

LIST 50-

```
50 PRINT A * B
60 PRINT C * A
70 PRINT A - C
```

listera' il programma a partire dalla linea 50 fino alla fine.

Il comando LIST e' usato sempre in forma diretta, tuttavia puo' essere inserito anche in un passo di programma.

In questo caso, dopo aver eseguito il LIST parziale o totale il sistema andra' in :Ready.

CONT

Questo comando fa ripartire l' esecuzione di un programma che era stato fermato o dal tasto RUN/STOP o per aver incontrato nel programma stesso un' istruzione di STOP o di END.

Il programma reiniziera' dall' esatta posizione alla quale si era fermato.

Mentre l' esecuzione del programma e' arrestata si puo' sia listare il programma che portarvi dei cambiamenti,

sia controllare il valore che hanno assunto a quel momento le variabili, sia variarne il valore o il contenuto.

Durante la prova (il così detto DEBUGGING) di un programma può rivelarsi una funzione molto utile.

Verrà visualizzato un messaggio:

CAN'T CONTINUE

se invece il programma si è fermato per un errore oppure se è stato commesso un errore, (tipico quello di dare un colpo di return sulla scritta READY.) prima di CONT.

NOTA

Si ha la segnalazione di errore come quella appena detta perché il Basic interpreta come l'invio di un comando diretto READ Y, il cui significato preciso sarà visto in seguito.

## L'ISTRUZIONE GET

Il maggior problema presentato dall'istruzione INPUT è che essa non permette di lavorare in REAL-TIME.

Durante il tempo adoperato dall'utente per introdurre i dati e premere il tasto di ritorno carrello, tutti i procedimenti di elaborazione vengono arrestati.

Il Basic del CBM 64 è stato fornito di una speciale funzione che permette di accettare dalla tastiera un carattere alla volta o di accertarsi se un tasto viene premuto.

Il comando da dare è GET. Come sintassi GET è identico all'INPUT per cui è possibile specificare una lista di variabili, ma in generale tale modo di operare non è il migliore in quanto lo scopo del GET è di tenere sotto sorveglianza la tastiera e ritornare al programma non appena un tasto viene premuto.



Quando viene richiesto un valore numerico :  
Es.

```
10 GET A
```

solo tasti numerici vengono accettati in ingresso, mentre la pressione di qualsiasi altro tasto dara' luogo ad un :

```
?SYNTAX ERROR
```

L' uso di un valore numerico puo' dar luogo a confusione perche' il valore restituito e' ZERO se nessun tasto viene premuto.

L' uso migliore del GET si ha con le variabili stringa. Se nessun tasto viene premuto la stringa avra' valore nullo e lunghezza pari a 0, mentre in caso contrario la stringa conterra' il carattere corrispondente al tasto premuto.

Per un approfondimento nel suo uso e' bene vedere il capitolo relativo all' uso delle stringhe.

Comunque diamo subito un esempio operativo di questa istruzione.

```
10GET A$  
20IF A$=""THEN
```

Queste istruzioni permettono di ottenere un ciclo di attesa finche' un tasto non sia premuto.

Come abbiamo detto, questo comando puo' essere anche utilizzato per inserire piu' variabili:

```
10 GET A$  
20 IF A$="" THEN  
30 GET A$,B$,C$,D$,A
```

Normalmente la forma di GET come alla linea 30 non viene utilizzata in quanto non e' controllabile come OUTPUT. Viene invece usata spesso nel colloquio con le periferiche nella forma GET#.

## GOTO E IF...THEN

Come abbiamo visto una delle maggiori comodita' della programmazione in Basic e' la possibilita' di ritornare indietro e rieseguire alcune linee di programma. Cio' puo' essere fatto in due modi:

a) Senza condizioni con l' istruzione GOTO

b) Con una istruzione condizionata ( detta anche istruzione di SALTO condizionato), come IF...THEN

GOTO e' scritto per specificare un numero di linea alla quale il programma deve ritornare.

GO TO puo' essere anche scritto con uno spazio fra il GO ed il TO perche' il Basic del CBM 64 riconosce entrambe le forme.

L' istruzione IF...THEN ha tre forme

IF (seguita dalla condizione) THEN (seguito da una istruzione).

IF(seguita dalla condizione)GOTO ad un numero di linea

IF(seguito da condizione) THEN(seguito da un numero di linea).

L' istruzione IF...THEN permette di eseguire l' istruzione che segue immediatamente il THEN solo se la specifica condizione si e' verificata.

La condizione puo' essere fissata ponendo fra le due espressioni uno dei sei operatori relazionali.

Fino a questo momento sono stati descritti programmi che eseguono delle funzioni singole in ordine seriale. Per cui a questo punto e' abbastanza familiare il concetto che viene eseguita per prima la linea 10 successivamente la linea 20 e cosi' via per le altre linee numerate in ordine crescente.

Se si vuole ad esempio calcolare e stampare per i numeri fra 1 e 20, il numero, il quadrato e l' area di cerchio corrispondente, si potrebbe scrivere un programma nella sua forma lineare:

```
10PRINT1,1*1,1*3.1415
20PRINT2,2*2,2*2*3.1415
30PRINT3,3*3,3*3*3.1415
```

e cosi' via poniamo fino alla linea 200  
Facendo ricorso al concetto di variabile e con l' aggiunta di quanto esposto si puo' scrivere un programma molto piu' corto e perciò piu' efficiente.  
Es.

```
10 PRINT"VAL","QUAD","A.CERCHIO"
20 I=I+1
30 PRINT I,I*I,I*I*3.1415
40 GOTO20
```

#### NOTA

Come al solito ci soffermeremo a lungo su questo piccolo programma per chiarire i concetti delle istruzioni e per introdurre le subroutines, i salti ecc.

La linea 10 stampa una intestazione per le colonne di numeri.Essa viene eseguita una sola volta.

La prima volta che la linea 20 viene eseguita la variabile I non e' mai stata usata e quindi il suo valore e'= 0.

La linea 30 stampa il valore assunto da I, dal suo quadrato e calcola l' area del cerchio con raggio I e corrisponde alla linea 10 e seguenti del programma precedente con l' unica differenza che le costanti sono rimpiazzate da variabili.

La linea 40 contiene un comando di GOTO che fa ritornare l' esecuzione indietro e la fa ripartire dalla linea 20. Mettendo in esecuzione il programma si vedra' che esso

continuera' a stampare i valori di I finche' non si premera' il tasto di RUN/STOP.

Se si vuole ripartire dal punto in cui ci siamo fermati si puo' digitare il comando:

CONT

mentre se si vuol ripartire dall' inizio occorrera' digitare nuovamente RUN.

Per quanto riguarda l' esempio fatto si puo' osservare che, malgrado tale programma usi il GOTO esso non ha realmente risolto il problema posto e cioe' di stampare i primi numeri sullo schermo.

Tuttavia prima di considerare questo aspetto si introduca un piccolo errore nel programma in modo da esaminare uno dei piu' comuni inconvenienti che un uso poco corretto del comando GOTO puo' generare.

Proviamo a modificare l' istruzione alla linea 40 nel seguente modo:

40 GOTO 10

Passando all' esecuzione si vedra' che il programma visualizzera' alternativamente l' intestazione e i valori calcolati anziche' stampare una unica intestazione.

Infatti il salto ad una linea di programma errata e' il piu' comune errore che si puo' fare durante la programmazione.

## ANELLI CONDIZIONATI

L' istruzione IF THEN permette di eseguire l' istruzione che segue immediatamente il THEN solo se una specifica condizione si e' verificata. La condizione puo' essere fissata mettendo uno dei sei operatori relazionali fra due espressioni.

Gli operatori relazionali sono:

- = uguale
- <> diverso
- > maggiore di
- < minore di
- >= maggiore o uguale di
- <= minore o uguale

Es.

IFA<B THEN PRINT"A MINORE DI B"

Se l'espressione e' vera l'istruzione di PRINT, in questo caso, viene eseguita.

Se l'espressione e' falsa, il programma salta alla prossima linea.

Nell'esempio precedente la linea 40 puo' essere sostituita:

40 IF I = 20 THEN GOTO 20

Come si vede l'istruzione IF THEN permette di prendere delle decisioni durante l'esecuzione del programma.

Cio' permette quindi di limitare il programma stesso e di eseguire delle azioni non appena si e' verificata una condizione.

Nel caso in oggetto il programma viene eseguito per I che va da 1 a 20 e passa oltre l'istruzione 40 non appena I diventa maggiore di 20.

## L' ISTRUZIONE GOSUB E LE SUBROUTINES

Si e' visto ora che le funzioni definite dall'utente, con un singolo argomento, possono essere usate come qualsiasi altra funzione intrinseca.

La maggiore limitazione delle funzioni definite dall'utente sta nel fatto che esse sono realizzate mediante una singola espressione algebrica.

In molti casi tuttavia e' necessario ripetere un certo

numero di espressioni durante lo svolgimento di un programma.

Queste istruzioni possono essere raccolte in un unico gruppo ed eseguite mediante una istruzione di GOSUB.

Le linee di programma che sono così riunite vengono chiamate:

### SUBROUTINES

La sintassi dell' istruzione GOSUB è la seguente:

```
100 GOSUB X
```

dove X è il numero di linea al quale inizia la subroutine.

L' istruzione GOSUB differisce dal GOTO in quanto RICORDA la linea dalla quale si è partiti per andare alla subroutine stessa e ritorna alla linea immediatamente successiva in modo automatico dopo che il sottoprogramma è stato eseguito.

L' ultima linea della subroutine deve essere una istruzione di RETURN che appunto permette la corretta esecuzione di questo programma.

Es.

```
10 INPUT A,B  
20 GOSUB 40  
30 PRINT C:END  
40 C= A*B  
50 RETURN
```

Nel programma descritto le linee verranno eseguite con la sequenza:

```
10-20-40-50-30
```

La limitazione fisica sul numero di GOSUB che possono

essere contemporaneamente in funzione e' di 23 richiami a subroutines.

I richiami a subroutines dall' interno di un' altra subroutine vengono chiamati:

### RICHIAMI INTERNI

e tale tecnica di operare viene chiamata

### ANNIDAMENTO DI SUBROUTINES.

E' inoltre abbastanza frequente il caso di programmi che utilizzano subroutines annidate entro altre subroutines e cosi' via.

L' unico limite a tale tecnica operativa e' la disponibilita' di memoria.

### PRECAUZIONI DA OSSERVARE

Un errore comune che si commette nell' uso delle subroutines e' quello di permettere al programma principale di invadere una subroutine che lo segue.

Cio' dara' luogo ad un messaggio di errore :

### RETURN WITHOUT GOSUB ERROR.

In conclusione si puo' dire che le subroutines sono degli strumenti estremamente potenti per la programmazione e che permettono di strutturare i programmi in blocchi.

### NOTA

1)Ricordiamo che un' istruzione GOTO non seguita da nessun numero e' equivalente a GOTO 0.

2)Ogni volta che il programma esegue una istruzione GOSUB, il numero di linea e la posizione della linea nel programma sono salvati in una speciale area di memoria

chiamata area di STACK, che puo' contenere fino ad un massimo di 256 Bytes, che sara' quindi il limite all' ammontare dei dati che possono essere salvati in STACK. Ecco quindi il motivo per cui il numero degli indirizzi a subroutines e di conseguenza il numero stesso dei salti deve essere limitato.

3) Ricordiamo ancora una volta di fare PARTICOLARE attenzione a che ad ogni comando di GOSUB corrisponda il suo RETURN.

### L' ISTRUZIONE 'FOR...NEXT

L' introduzione del concetto di subroutine ci porta necessariamente all' uso ripetitivo di particolari gruppi di istruzione. Di conseguenza sara' necessario qualcosa che registri il numero di passaggi altrimenti infiniti. In altre parole il concetto di:

### CONTATORE

o meglio di variabile contatore.

Lo schema generalizzato e' il seguente:

```
10 I=A
20 I=I+C
30 IF I<B THEN20
```

da un punto di vista strettamente matematico cio' e' un vero controsenso in quanto non e' pensabile di dare 2 assegnazioni alla stessa variabile, ma per la programmazione cio' e' logico e vediamo come con un esempio.

```
5 A=1:REM VALORE INIZIALE
6 B=5:REM VALORE FINALE
7 C=1:REM INCREMENTO
10 I=A
```



```

20 I=I+C
30 PRINTI*I*3.1415
40 IF I<B THEN20

```

il risultato di questo programma sara' la visualizzazione di:

```

12.566
28.2735
50.264
78.5375

```

Introduciamo ora il comando FOR...NEXT...STEP. Con questo nuovo comando il problema precedente sarebbe stato risolto:

```

10 FORI=1TO5(STEP1)
20 PRINTI*I*3.1415
30 NEXT

```

La sintassi generale del comando FOR...NEXT e' la seguente:

```

FOR I=A TO B STEP C

```

con alla fine del ciclo:

```

NEXT

```

Il valore di I variera' quindi da A a B con un incremento di C. Quando l' incremento C e'=1 allora puo' essere sottinteso.

I valori A,B,C possono essere delle variabili o possono risultare da dei procedimenti di calcolo.

Es.

```

FORI=3*4.18 TO J*2 STEP-1:REM PER I 6.27

```

Vediamo un esempio di calcolo di un fattoriale.

```
5 INPUT N
10 NF=1
20 FOR I=1TON
30 NF=NF*I
40 NEXT
```

## ANELLI CONSECUTIVI

I cicli FOR...NEXT, come abbiamo detto a proposito delle subroutines, possono essere annidati uno dentro l'altro. E' tuttavia molto importante che per ciascun anello venga usata una variabile contatore diversa come nell' esempio seguente.

Es.

```
10 FORI=1TO10
20 PRINT I
30 FORJ=1TO10
40 PRINT J*J
50 FORH=1TO10
60 PRINTH-350
70 NEXTH,J,I
```

## CAPITOLO SESTO

### LE VARIABILI

Abbiamo già detto che il CBM 64 può essere usato come una calcolatrice di grosse dimensioni che esegue funzioni matematiche direttamente e ne visualizza quindi i risultati.

Tuttavia è spesso necessario avere a disposizione dei valori intermedi o eseguire operazioni interattive.

Per effettuare programmi a tutti i livelli è stato necessario implementare l'uso di funzioni che possono assumere quindi una varietà di valori di volta in volta. Una funzione che può assumere diversi valori viene definita in matematica come in programmazione con il nome di :

### VARIABILE

Fino a questo momento le abbiamo semplicemente utilizzate, vediamo ora di approfondirne sia il concetto che i modi di utilizzo.

Nel BASIC del CBM 64 le variabili sono definite mediante un massimo di due caratteri alfanumerici.

Se la variabile è una variabile numerica allora non vi è nessun carattere particolare nel suo nome per cui il carattere A è considerato come la variabile numerica A.

Il carattere AA sarà allora una diversa variabile. Al sarà una terza variabile diversa dalle prime due.

Nel Basic del CBM 64 esistono 4 tipi di variabili:

#### VARIABILI NUMERICHE

#### VARIABILI INTERE

#### VARIABILI STRINGA

#### MATRICI

Le variabili numeriche sono quelle appena viste.

Per introdurre il concetto di variabili intere e' necessario fare una premessa circa la memorizzazione.

Ricordando che il Bit e' l' unita' piu' piccola di memoria e che puo' essere 0 o 1, mentre il Byte e' l' insieme di 8 Bits, vediamo , almeno a grandi linee, come vengono memorizzate le variabili.

Il Basic opera sempre in virgola mobile, cioe' non memorizza il valore numerico, ma assegna 5 bytes per memorizzare il numero.

Uno di questi Byte viene utilizzato per l' esponente e gli altri 4 per la mantissa in modo da ottenere 9 cifre significative.

In molti casi quando la quantita' da rappresentare e' una quantita' intera la variabile puo' essere espressa in maniera piu' semplice.

Per poter avere una maggiore efficienza della memoria il CBM 64 ha la possibilita' di memorizzare numeri come valori interi in 2 soli Bytes.

Cio' da origine ad un tipo di variabile chiamata appunto:

### VARIABILE INTERA

Questi numeri devono essere compresi fra -32767 e +32767

#### NOTA

Per chi voglia saperne di piu' segnaliamo che siamo in presenza di una rappresentazione binaria pura di due Bytes, cioe' di 16 bits in cui il bit di ordine piu' alto contiene il segno.

Se la variabile contiene dati alfanumerici essa viene definita una stringa. Una variabile stringa deve terminare con un carattere \$.

Es.

A\$ AA\$ B1\$

Sono tutte variabili diverse.

Ma una variabile stringa puo' contenere anche una funzione, un numero, uno o piu' caratteri grafici. Esiste tutta una serie di comandi che permettono di operare sulle variabili stringa o come si dice piu' semplicemente sulle:

### STRINGHE

Le matrici sono il quarto tipo di variabile del Basic e si differenziano dalle altre variabili per le parentesi che le seguono.

Anche di questo tipo di variabile daremo un dettaglio quando parleremo dell' istruzione DIM.

### TECNICHE DI PROGRAMMAZIONE AVANZATA

In quanto precede sono state descritte quasi esclusivamente funzioni numeriche, ma la maggior parte dei programmi operano anche con grandezze alfanumeriche.

Il Basic del CBM 64 contiene un insieme di funzioni che permettono di lavorare con questi dati.

I dati alfanumerici possono essere definiti come una connessione continua di caratteri che vengono visti dal Basic come una singola variabile.

Quasi tutte le regole che si applicano alle normali variabili si applicano anche alle variabili stringa. Si provi a scrivere:

```
10 A$="BUONGIORNO"  
20 PRINT A$
```

con queste istruzioni abbiamo detto al Basic di definire una stringa e di visualizzarla.

In questo modo si puo' definire una stringa che puo' avere un massimo di 75 caratteri, cioe' due righe dello schermo.

Tuttavia la limitazione del numero di caratteri che possono essere memorizzati in una variabile stringa e' di 255.

Infatti si possono costruire stringhe piu' lunghe di quanti caratteri possono entrare in un' unica linea.

L'accumulazione di caratteri viene fatta attraverso la concatenazione di stringhe semplicemente usando l' operatore "+".

Si puo' modificare l' esempio precedente in modo:

```
10 A$="BUONGIORNO"  
20 B$="A TUTTI"  
30 PRINT A$+B$
```

in modo da avere:

```
BUONGIORNO A TUTTI
```

## OPERAZIONI E COMANDI PER LE STRINGHE

Due stringhe possono essere confrontate e si possono eseguire numerose operazioni sulle stringhe tipo quella che abbiamo appena visto.

Per questo motivo spiegheremo ora i comandi necessari alla manipolazione delle stringhe seguiti da qualche breve esempio.

### STR\$

Serve per convertire un numero in una stringa. L'argomento di STR\$(X) deve quindi essere numerico.

I numeri positivi verranno preceduti nella loro stringa equivalente da uno spazio bianco, mentre i numeri negativi avranno il segno meno nella posizione corrispondente.

Es.

```
10 X=1024.25
20 PRINT STR$(X)
```

verra' visualizzato

1024.25

preceduto da uno spazio bianco.

VAL

E' la funzione complementare di STR\$. Essa converte una stringa in un numero che puo' essere usato per i calcoli. Se il primo carattere diverso dallo spazio bianco che si incontra nella stringa e' un carattere non numerico allora la funzione VAL restituirà un valore nullo. In altre parole la funzione VAL puo' convertire tanti caratteri numerici quanti ne incontra prima di trovare un carattere non valido.

Es.

```
?VAL("3.1415 CBM")
```

sara' visualizzato il valore:

3.1415

CHR\$

Riporta il carattere ASCII corrispondente al codice numerico dell' argomento.

Es.

```
?CHR$(65)
```

Stampera' la lettera A.

Come abbiamo già visto in precedenza in una stringa si

possono avere tutti normali caratteri ASCII introducendoli o attraverso una stringa racchiusa fra virgolette o tramite organi di INPUT.

Tuttavia parecchi dispositivi richiedono l'uso di caratteri di controllo che non possono essere prodotti con i normali mezzi.

Ad esempio una stampante COMMODORE usa il ritorno carrello assieme al tasto delle maiuscole come carattere di termine per indicare che il carrello della stampante deve ritornare indietro ma senza eseguire un avanzamento carta in modo da effettuare sovrastampe.

La funzione CHR\$ permette allora di specificare questi caratteri di controllo semplicemente assegnando il numero di codice ASCII del carattere stesso.

Per un approfondimento sulle periferiche vedi "LE PERIFERICHE COMMODORE" edito da EVM.

La funzione CHR\$ converte cioè un numero nella rappresentazione interna ASCII.

Il valore dell' argomento deve essere compreso fra 0 e 255.

ASC(X\$)

La funzione ASC converte un carattere in un codice ASCII e tale codice può essere quindi usati in calcoli numerici.

Il parametro d' ingresso della funzione ASC è una stringa.

Es.

?ASC("A")

65

Ready.

In pratica è la funzione complementare della precedente CHR\$.

Se la stringa è formata da più di un carattere la funzione ASC riporterà il codice SOLO del primo



carattere della stringa.

Es.

```
?ASC("BIANCO")
```

```
66
```

```
Ready.
```

Cioe' solo il codice del carattere B.

Se non esiste nessun carattere nella stringa, se cioe' siamo in presenza di una stringa vuota, allora avremo un errore di:

```
?ILLEGAL QUANTITY
```

## SEGMENTI DI STRINGA

In molti casi e' necessario poter avere solo una parte di una stringa ad esempio quando si voglia sviluppare una lista in ordine alfabetico.

Vediamo ora le funzioni disponibili sul CBM 64 per l'analisi e la manipolazione delle stringhe e successivamente degli esempi di lavoro sulle stesse.

### LEFT\$

La funzione LEFT\$(X\$,I)riporta gli I caratteri piu' a sinistra della stringa che appare in argomento.

Es.

```
10 X$="GIOVANNI"  
20 PRINT LEFT$(X$,3)
```

si avra' come risultato di stampa

```
GIO
```

```
10 X$="GIOVANNI"  
20 PRINT LEFT$(X$,5)
```

si avra' come risultato:

GIOVA

Se I e' negativo, zero o maggiore di 255 allora sullo schermo apparira' il messaggio:

?ILLEGAL QUANTITY ERROR

RIGHT\$

E' una funzione simile alla precedente solo che in questo caso, come dice il nome stesso, il conto inizia da destra, cioe' dall' ultimo carattere della stringa piu' a destra.

Es.

```
10 X$="FRANCESCO"  
20 PRINT RIGHT$(X$,2)
```

si avra' come risultato di stampa:

CO

mentre

```
10 X$="FRANCESCO"  
20 PRINT RIGHT$(X$,4)
```

si avra' come stampa

ESCO

Anche per questa funzione valgono le limizioni di quella precedente.

MID\$

Per questa funzione esistono due forme.

La prima, la piu' generale e':

```
MID$(stringa,I,J)
```

dove I e' la posizione di partenza all' interno della stringa.

Invece J sta ad indicare il numero dei caratteri della stringa argomento della funzione.

Es.

```
10 A$="BUON"  
20 B$="GIORNO DI' ANNO"  
30 PRINT A$ +MID$ (B$,8,3)
```

che visualizzera':

```
BUON DI'
```

Se I e' maggiore della lunghezza della stringa viene restituita appunto una stringa nulla.

Naturalmente I e J non possono essere negativi ne maggiori della lunghezza massima di una stringa cioe' 255 perche' in questo caso verra' visualizzato un messaggio di

```
?ILLEGAL QUANTITY ERROR
```

Se J e' maggiore del numero dei caratteri che possono essere prelevati dalla stringa, allora vengono restituiti tutti i caratteri dalla posizione I alla fine della stringa stessa.

La seconda forma e':

```
MID$(stringa,i)
```

ed ha lo stesso effetto della forma precedente quando J e' maggiore della lunghezza della stringa.

Vengono cioe' riportati tutti i caratteri che partono dalla posizione I fino alla fine della stringa stessa.

## LEN

La funzione LEN(X\$) fornisce l' esatto numero di caratteri contenuto in una stringa. Vengono conteggiati anche i caratteri che non appaiono in stampa sullo schermo e gli spazi bianchi.

Es.

```
10 A$="COMMODORE"  
20 PRINT LEN(A$)
```

avremo come risultato 9 che e' appunto il numero di caratteri della stringa A\$.

Es.

```
10 A$=" COMMODORE"  
20 PRINT LEN(A$)
```

avremo invece come risultato 11 perche' vengono contati anche gli spazi iniziali e finali.

## LE MATRICI ED IL DIM

### DIM

Se non altrimenti specificato il Basic del CBM 64 classifica una variabile come una matrice ad una sola dimensione.

Tale matrice ha 11 possibili valori cioe' da 0 a 10.

Per cui se si desidera dimensionare una variabile con piu' di 11 elementi sara' necessario usare un comando di DIM cioe' di dimensionamento.

Le matrici possono dimensionare qualsiasi tipo delle variabili viste, cioe' numeriche, intere o stringhe.

Per le matrici numeriche metteremo una variabile numerica, mentre se vogliamo definire matrici di dati alfanumerici aggiungeremo un \$ e matrici di valori interi(INTEGER) il segno di percentuale (%).

Il numero totale degli elementi di una matrice puo' essere calcolato moltiplicando le dimensioni di ciascun indice per l' altro indice, tenendo presente che si parte da 0. Per esempio la matrice A(6,9) contiene 70 elementi.(6+1 \*9+1).

Puo' essere definita quindi come abbiamo visto una tabella ad una sola entrata, una matrice a doppia, tripla,quadrupla, ecc. entrate.

I limiti del numero di indici ed il valore massimo di ciascun indice sono in funzione della memoria disponibile, tuttavia il numero di indici, (e ci sembra ovvio che si tratta di limite teorico) non puo' superare 255.

Il comando che definisce una matrice e' appunto il comando DIM.

Esempi di uso del comando DIM

```
10 DIM A(10)
20 DIM A%(5,6),B%(7,9),C%(100,3)
30 DIM A$(20,10,5),B$(6,4,75)
```

Nella riga 10 e' dimensionata una matrice numerica, nella linea 20 una matrice di integer e nella riga 30 una matrice per dati alfanumerici.

## CONSUMO DELLA MEMORIA

Riportiamo una tabella di consumo della memoria utilizzando il comando DIM

```
5 Bytes per il nome della matrice
2 Bytes per ogni DIM
2 Bytes per ogni elemento di variabile integer
5 Bytes per ogni elemento di variabile numerica
3 Bytes per ogni elemento di variabile stringa
1 Byte per ogni carattere in ogni elemento di stringa
```

## I COMANDI DATA, READ, RESTORE

Quando una o piu' variabili necessitano di dati si possono usare i comandi DATA e READ invece di fare assegnazioni di valori con il LET o con il segno =.

Si riesce cosi' ad assegnare numerosi valori con una sola linea di programma.

Es.

```
100 DATA 20,12,16,70
```

che si possono rileggere con un comando:

```
120 READA,B,C,D
```

dovendo eseguire piu' volte la lettura dei valori contenuti nei DATA sara' pero' necessario usare il comando RESTORE.

Per chiarire l'uso e la funzione di questi comandi porteremo degli esempi per vedere come funzionano con e senza il RESTORE.

Es.

```
100 DATA10,20,30,40,50,60,70,80,90,100
```

```
200 READA,B,C
```

verranno assegnati i seguenti valori:

```
A = 10
```

```
B = 20
```

```
C = 30
```

proseguendo si potra' avere

```
300 READD,E,F
```

e verranno assegnati questi valori:

D = 40  
E = 50  
F = 60

continuando:

400 READA,D,F,G

verranno assegnati i seguenti valori

A = 70  
D = 80  
F = 90  
G = 100

per cui alle variabili A e D si assegnano a secondo del momento in cui avviene la lettura i valori:

|        |        |
|--------|--------|
| A = 10 | A = 70 |
| D = 40 | D = 80 |

e questo perche' il puntatore di inizio di lettura dei caratteri si incrementa continuamente.

Usando invece il comando RESTORE si ha sempre un ritorno indietro del puntatore.

Es.

100 DATA10,20,30,40,50,60,70,80,90,100

200 READ A,B,C

leggera':

A = 10  
B = 20  
C = 30

210 RESTORE

.....

300 READ A,B,C

sara'

A = 10

B = 20

C = 30

che invece, stante all' esempio precedente, cioe' senza  
l' uso del comando RESTORE sarebbe stato:

A = 40

B = 50

C = 60



## CAPITOLO SETTIMO

### LA FUNZIONE RANDOM

Dedichiamo un' attenzione particolare alla funzione Random RDN per l' importanza che essa riveste nel calcolo e nella simulazione statistica e nel campo dei giochi. La funzione generatrice di numeri casuali e':

RND(argomento)

Questa funzione il cui argomento puo' essere positivo, zero o negativo e' utilmente impiegata in giochi, debugging dei programmi, statistica.

Se e' usata con un numero positivo come argomento RND genera una serie di valori casuali sempre minori di zero ed e' comunque una serie che prende per riferimento un numero all' accensione della macchina. Naturalmente questo numero varia da macchina a macchina. Es.

FORI=1TO6:PRINTRND(1):NEXT

e verranno visualizzati dei numeri come segue:(ATT. e' solo un esempio)

.523696687  
.236945124  
.541246987  
.365474445  
.233335698  
.556547852

Provate a ripetere questo esempio accendendo e spegnendo piu' volte il vostro CBM 64.

Se la funzione e' usata con in argomento un numero

negativo si avra' una serie di numeri casuali, ma sempre uguali per lo stesso valore messo in argomento.

Si provi l'esempio precedente sostituendo -1 o altro valore negativo, anche questa volta ripetutamente, ma senza spegnere il computer.

Nel caso si usi lo 0 come argomento avremo allora realmente una serie di numeri casuali.

I valori a nove o dieci cifre che appaiono sullo schermo possono non essere utilizzabili per i nostri scopi.

Vediamo allora qualche esempio di applicazioni pratiche.

I) Si voglia generare n un programma di gioco il valore di uscita di un dado per dieci lanci.

```
10 FOR I=1 TO 10
20 PRINT INT(6*RND(1))+1
30 NEXT
```

II) Si desidera simulare l'uscita dei numeri del lotto su tutte le 10 ruote.

```
10 FOR I=1 TO 50
20 PRINT INT(90*RND(1))+1
30 NEXT
```

In generale la formula per la generazione di numeri casuali, ma compresi in un certo RANGE sara':

$$\text{NUMERO} = \text{INT}((\text{limite superiore} - \text{limite inferiore} + 1) * \text{RND}(1)) + \text{limite inferiore}$$

Vediamo delle applicazioni grafiche

```
10 PRINT CHR$(17)
20 PRINT CHR$(205.5+RND(1));
30 GOTO 20
```

La linea 10 serve a pulire lo schermo. La chiave del programma e' naturalmente la linea 20.

Come abbiamo visto, e per i particolari vedere nelle tavole, la funzione CHR\$(X), con X variabile fra 0 e 255 ci consente di stampare sullo schermo un carattere o un segno grafico del CBM 64.

Usando la formula 205.5+RND(1) il computer sceglierà un numero fra i valori 205.5 e 206.5 con una probabilità del 50 % di eventi.

Poiché CHR\$ ignora la parte decimale di un numero avremo la visualizzazione di caratteri corrispondenti alternativamente ma in modo casuale a CHR\$(205) e CHR\$(206), con l'effetto che potrete notare sullo schermo.

Provate, ad esempio, a selezionare altri caratteri complementari. E' questa la strada per generare ad esempio un labirinto!!!

## GLI OPERATORI LOGICI

Oltre agli operatori matematici e agli operatori di relazione visti in precedenza esiste una terza categoria di operatori detti appunto operatori logici.

Gli operatori logici AND, OR e NOT possono essere usati per modificare le risultanze di operatori relazionali o per produrre risultati aritmetici essi stessi.

Gli operatori logici, conosciuti come operatori BOOLEANI, possono essere usati per consentire operazioni logiche con bit in due operandi.

Non vogliamo però approfondire eccessivamente l'argomento, e ci limiteremo quindi a spiegare il funzionamento e l'uso di ogni comando operatore, rimandando al linguaggio macchina il trattamento dei Bit.

## AND

E' usato in operazioni booleane per controllare bits secondo le tavole della verita'. E' anche usato, e qui ci interessa di piu' per controllare la verita' di entrambi gli operatori.

Es.

```
10 IF X=7 AND W=3 THEN GOTO200
```

Cioe' se si verifica la condizione che sia X e' = a 7 che W e' = a 3 allora salta alla riga 200.

## NOT

E' usato in negazioni. Nel seguente esempio, se il complemento a due di AA e' = a BB e se BB non e' uguale a CC allora l' espressione e' vera

```
10 IF NOT AA=BB AND NOT(BB= CC) THEN 200
```

## OR

Anche l' uso di questo operatore puo' essere spiegato meglio con un esempio che con lunghi giri di parole.

```
100IF(A=B) OR (X=20) THEN 200
```

Cioe' se A=B o X=20 allora vai a 200.

## I TASTI FUNZIONE

Sulla parte destra del CBM64, fra l' altro allo stesso modo del VIC 20, sono presenti 4 tasti di diverso colore comunemente conosciuti come :

## TASTI FUNZIONE

e che infatti recano i numeri da F1 a F8.

A questi tasti e' possibile assegnare un valore o un comando qualsiasi che potra' essere utilizzato in qualsiasi momento ai quali sono stati assegnati i seguenti valori CHR\$:

```
F1 = CHR$(133)
F2 = CHR$(137)
F3 = CHR$(134)
F4 = CHR$(138)
F5 = CHR$(135)
F6 = CHR$(139)
F7 = CHR$(136)
F8 = CHR$(140)
```

Quelli appena visti sono i valori da assegnare ai tasti funzione tramite i comandi CHR\$.

Avrete visto infatti in molti programmi la visualizzazione di un messaggio di questo tipo:

PER INIZIARE PREMERE F1

Un semplice sistema per controllare se ad esempio il tasto F1 e' stato premuto e' quello di far ricorso al GET con il metodo seguente:

```
100GET A$:IF A$ CHR$(133) THEN 100
```

Altro e piu' importante utilizzo che sicuramente i possessori del VIC 20 gia' conoscono per averlo implementato con il SUPER EXPANDER, e' quello di assegnare dei comandi ai tasti funzione che possano, ad esempio, essere utilizzati durante la programmazione.

Il seguente programma consente di definire i tasti funzione cosi' che si possa stampare una qualsiasi stringa di una lunghezza massima di 8 caratteri e che nel nostro caso abbiamo utilizzato per comandi Basic.

Le linee da 200 a 320 immettono il codice macchina a

partire dalla locazione \$C000.

Abbiamo inserito anche una routine di controllo errore per cui se il programma che digiterete si ferma alla linea 320 e stampa quindi il messaggio di errore contenuto nella stringa relativa dovrete ricontrollare i valori dei DATA inseriti.

I comandi dalla linea 2 alla 9 possono essere variati con molta semplicità'.

```
0 REM PROGRAMMA PER LA DEFINIZIONE DEI TASTI FUNZIONE
1 GOSUB 300
2 F$(1)=" LIST"+CHR$(13)
3 F$(2)=" RUN"+CHR$(13)
4 F$(3)=" GOSUB"
5 F$(4)=" SAVE"+CHR$(34)
6 F$(5)=" GOTO"
7 F$(6)=" CHR$("
8 F$(7)=" LOAD"
9 F$(8)=" RETURN"
10 SYS12*4096+64
20 V=12*4096-1
30 FORI=1TO8 : K=I-1 : V1=V+K*8
40 FORJ=1TO LEN(F$(I))
50 POKEV1+J,ASC(MID$(F$(I),J,1))
60 NEXT: NEXT: END
200 DATA 120,169,87,141,20,3,169,192,141,21
210 DATA 3,88,162,63,169,0,157,0,192,202
220 DATA 16,250,96,165,197,201,64,208,6,141
230 DATA 151,192,76,148,192,205,151,192,240,44
240 DATA 141,151,192,162,3,221,152,192,240,5
250 DATA 202,16,248,48,29,138,174,141,2,240
260 DATA 3,24,105,4,10,10,10,168,162,0
270 DATA 185,0,192,157,119,2,200,232,224,8
280 DATA 208,244,134,198,76,49,234,64,4,5,6,3,0
300 FORI=0TO92:READA:Z=Z+A
310 POKE12*4096+64+I,A:NEXT
320 IFZ<>10771 THEN PRINT"ERRORE NEI DATA": STOP
330 RETURN
```

## GLI ULTIMI COMANDI

Vediamo ora una breve sintesi dei restanti comandi Basic che fino a questo momento non abbiamo trattato.

### CMD

Questo comando serve per trasferire l' uscita dei dati da una periferica ad un' altra.

Normalmente l' uscita dati o OUTPUT e' sulla periferica primaria cioe' lo schermo TV o il monitor e questa istruzione serve appunto per far uscire i dati su stampante, nastro, disco o altro.

Quindi quando viene eseguito questo comando qualsiasi istruzione PRINT o comando LIST puo' inviare il testo su un generico file aperto preventivamente con un comando OPEN. Ricordiamo fra l' altro che qualsiasi errore di sistema verra' pero' visualizzato sullo schermo.

Ammettiamo di avere un programma in memoria e che si desideri listarlo su stampante. Si potra' farlo utilizzando il seguente esempio (che non e' il programma inmemoria ma la serie di comandi DIRETTI da dare).

Es.

```
OPEN4,  
CMD4  
LIST  
CLOSE 4
```

Con questo metodo avremo il listato su carta, mentre l' ultima istruzione verra' data non appena sara' riabilitato l' accesso alla tastiera.

### END

Questa istruzione fa terminare l' esecuzione di un programma e visualizza il messaggio READY. restituendo il controllo del computer all' utente. Puo' essere usata piu' volte all' interno di un programma ed e' utile in

fase di prova programmi.

L'istruzione END e' simile allo STOP e l'unica differenza e' che STOP visualizza il messaggio BREAK IN LINE XX, mentre END visualizza solo il READY.

## LET

Questa istruzione puo' essere usata per assegnare un valore ad una variabile. Pur essendo classica del Basic standard e pertanto obbligatoria in molte versioni, in quella in uso nei computer COMMODORE e' opzionale e puo' essere sottintesa.

Es.

10 LET A=12    equivale a    10 A=12

## NEW

Questo comando viene usato per cancellare un programma presente nella memoria del computer e per azzerarne tutte le variabili.

Puo' essere usato sia in modo diretto che in modo programma. Ricordiamo che abbiamo in pratica un NEW automatico tutte le volte che si esegue un LOAD da cassetta o da disco.

## REM

Questa istruzione permette di commentare un programma in modo tale da renderlo piu' comprensibile al momento di listarlo. Il commento puo' essere costituito da una qualsiasi parola o simbolo compreso le parole chiave del Basic purché sia preceduto da questa istruzione.

Quindi tutto cio' che viene scritto dopo il REM e che sta sullo stesso numero di linea viene ignorato dal Basic dal punto di vista della funzionalita' e viene semplicemente visualizzato quando si esegue il LIST.

Es.



10 REM CALCOLO DI UN FATTORIALE

20 FOR I=1 TO 10 :REM CICLO.

STOP

Viene usata per fermare l' esecuzione di un programma che sta girando in quel momento e per ritornare in modo diretto.

Si ottiene lo stesso effetto premendo il tasto RUN/STOP e il Basic visualizzera' il solito messaggio: ?BREAK IN LINE XX.

Ricordiamo che qualsiasi file aperto viene mantenuto e che le variabili mantengono i valori che hanno raggiunto in quel momento.

Il programma puo' essere fatto ripartire usando un comando CONT o GOTO ad un determinata linea.

SYS

Consente di saltare ad una determinata locazione di memoria indicata nel parametro che segue il SYS e che quindi non dovra' essere maggiore di 65535.

E' il sistema piu' comune per utilizzare insieme a programmi Basic delle sub-routines in linguaggio macchina.

La precauzione importante da prendere e' quella di far terminare la parte in Linguaggio Macchina con un istruzione RTS (RETURN FROM SUBROUTINE) in modo tale che si possa cosi' tornare in ambito Basic ed in modo da far riprendere al programma l' esecuzione dall' istruzione successiva alla SYS.

Una routine famosa e' quella detta di COLD START alla quale si salta con un SYS 64738 e che serve per resettare il computer.

WAIT

Questa istruzione fa si che l' esecuzione di un programma

venga sospesa finche' in una data zona di memoria non venga registrato un dato valore.

WAIT la cui sintassi e':

WAIT indirizzo,x,y

estrae il valore da una data locazione di memoria ed esegue un AND logico con il primo parametro del comando e, se e' presente il secondo parametro (y), allora viene eseguita una operazione di OR esclusivo fra il risultato del primo calcolo e il secondo parametro.

Es.

WAIT 36868,144,16

## I COLORI

E' necessario premettere che questo paragrafo e' solo una introduzione parziale a quanto vedremo piu' approfonditamente nei capitoli seguenti.

I colori disponibili sul CBM 64 sono 16, come riportiamo in tabella usando per praticita' il nome inglese:

- 1- BLACK
- 2- WHITE
- 3- RED
- 4- CYAN
- 5- PURPLE
- 6- GREEN
- 7- BLUE
- 8- YELLOW
- 9- ORANGE
- 10- BROWN
- 11- LIGHT RED
- 12- GRAY 1
- 13- GRAY 2
- 14- LIGHT GREEN
- 15- LIGHT BLUE
- 16- GRAY 3

E' possibile selezionare questa gamma rispettivamente per:

-Ciascun carattere

-Il bordo esterno

-L' interno

Si possono quindi utilizzare ben 4096 combinazioni diverse di colori.

I colori dei singoli caratteri possono essere selezionati direttamente utilizzando i tasti CTRL ed il tasto con il simbolo COMMODORE unitamente ai tasti dei numeri dall' 1 all' 8, sotto i quali infatti e' riportata l'abbreviazione del colore.

Mostriamo nelle tabelle riportate sotto i colori selezionabili con i tasti numerici e contemporaneamente con l' uso dei tasti CTRL o COMMODORE

#### USANDO IL TASTO CTRL

1 =BLACK  
2 =WHITE  
3 =RED  
4 =CYAN  
5 =PURPLE  
6 =GREEN  
7 =BLUE  
8 =YELLOW

## CON IL TASTO COMMODORE

1 =ORANGE  
2 =BROWN  
3 =LIGHT RED  
4 =GRAY 1  
5 =GRAY 2  
6 =LIGHT GREEN  
7 =LIGHT BLUE  
8 =GRAY 3

Questi colori sono inoltre selezionabili tramite la funzione CHR\$(X), sostituendo ad X il numero di riferimento del colore come si puo' vedere nell'appendice.

In questo modo quindi i colori possono essere visualizzati direttamente inserendoli in una istruzione PRINT.

```
10 PRINTCHR$(147):REM PULIZIA DI SCHERMO  
20 PRINTCHR$(30):REM CAMBIO COLORE
```

Provate a far girare e vedrete la parte interna ,cioe' il testo, diventare verde.

Esistono delle locazioni di memoria nelle quali inserendo un valore tramite l'istruzione POKE, o come si usa dire adoperando un brutto neologismo POKEggiando, si cambia rapidamente il colore del testo o del bordo.

Queste locazioni di memoria si chiamano registri e sono:

53280 per il bordo

53281 per il testo o interno

Provate a digitare:

POKE53281,5

e vedrete l'interno cambiare in verde come con la precedente istruzione:

PRINTCHR\$(30)

I valori da inserire tramite i POKE per cambiare colori sia nel bordo che nell' interno sono quelli riportati nella tavola all' inizio, per cui digitando anche in forma diretta:

POKE53280,0

POKE53281,1

avremo bordo nero e schermo bianco.

e:

POKE53280,7

POKE53280,2

avremo bordo giallo e interno rosso.

Per vedere una combinazione di colori provate il seguente programmino:

```
10FOR E=0TO15
20FOR I=0TO15
30POKE53280,E
40POKE53281,I
50FOR X=1TO1000:NEXT
60NEXT I:NEXTE
```

Da notare che alla linea 50 e' inserito un ciclo di temporizzazione per rallentare la visualizzazione.

## MAPPA DEI COLORI

Come abbiamo detto in precedenza lo schermo del CBM 64 e' una matrice di 40 colonne per 25 righe, cioe' per un totale di 1000 cellette entro ognuna delle quali puo' essere rappresentato un carattere o un simbolo grafico. Il contenuto della memoria di ogni singola posizione e' immagazzinato in una serie di posizioni di memoria che vanno dalla locazione decimale 1024 alla locazione decimale 2023.

La locazione 1024 e' la piu' in alto a sinistra sullo schermo e la 2023 quella piu' in basso a destra.

I comandi relativi alle opzioni di stampa indirizzano automaticamente i caratteri, ma queste locazioni possono essere indirizzate anche tramite comandi di POKE.

Ad esempio per far apparire la lettera A nella posizione piu' alta dello schermo sara' sufficiente:

POKE1024,65

La formula per inserire tramite il comando POKE un simbolo Z sullo schermo sara' allora:

Il punto  $P=1024+colonna+40*riga$

percio'

POKE P,Z

Digitando ad esempio:

POKE1524,81

vedremo apparire una pallina nel centro dello schermo.

Il CBM 64 e' un computer particolarmente dedicato alla gestione della grafica e del colore per cui non solo si puo' indirizzare un segno grafico in una particolare locazione di schermo che come abbiamo visto corrisponde

ad una locazione di memoria, ma possiamo anche dargli un colore che si desidera e cio' indipendentemente dai colori del bordo o del testo.

Oltre alla mappa di schermo esiste una mappa di colore per ogni locazione dello schermo.

Questa mappa di colori e' localizzata agli indirizzi da 55296 a 56295 anche questa quindi per 1000 caratteri.

In pratica il punto piu' in alto a sinistra e' identificato come locazione 1024 dal punto di vista del contenuto (cioe' del simbolo ivi presente), e come locazione 55296 come contenuto del colore.

Riportandosi all' esempio precedente in cui per scrivere la lettera A nella posizione piu' in alto a sinistra avevamo scritto:

POKE 1024,81

per dare un colore, mettiamo il rosso a questa lettera dovremo aggiungere:

POKE55296,2

Ricordando che i colori variano secondo la tabella iniziale da 0 a 15.

La formula generale per l' assegnazione del colore ad un dato segno S sara' quindi:

Colore C =55296+colonna+40\*riga

per cui:

POKE C,S

## CAPITOLO OTTAVO

### LA GRAFICA

Tutte le grandi capacita' grafiche del CBM 64 derivano dall' uso dell' integrato 6567 Video Interface Chip ( conosciuto anche come VIC-II).

Questo integrato consente una grande varieta' di modi grafici, incluso fra l' altro il video di 40 colonne per 25 linee, un modo di visualizzazione di 320 per 200 punti in alta risoluzione ed i famosi SPRITES, un nuovo modo di gestire la grafica utilissimo in modo particolare per i giochi e per qualsiasi altro disegno di animazione.

Oltre tutto questi modi possono essere gestiti contemporaneamente sullo schermo. E' possibile per esempio, definire mezzo schermo in alta risoluzione e l' altra meta' in modo testo. Gli SPRITES poi possono muoversi facilmente e combinarsi fra loro.

Ricordiamo che per molti indirizzi di memoria sono forniti sia gli indirizzi in DECIMALE che in ESADECIMALE. Questi ultimi sono facilmente riconoscibili perche' sempre di 4 valori e spesso li indichiamo preceduti dal simbolo \$.

### LOCAZIONI GRAFICHE

Per prima cosa delle informazioni di carattere generale. Lo schermo del CBM 64 e' di 25 righe per 40 colonne per cui ci sono 1000 possibili locazioni sullo video del CBM64.

Normalmente lo schermo inizia dalla locazione decimale 1024 ( nella notazione esadecimale \$0400) e va fino alla locazione decimale 2023 ed ognuna di queste locazione e' di un Byte cioe' di 8 bits. Questo consente quindi di manipolare un numero variabile da 0 a 255.

Connessa alla memoria di schermo c' e' un gruppo di altre



1000 locazioni di memoria chiamata MEMORIA COLORE o COLOR RAM. Questa inizia dalla locazione 55296 (\$D800) e va fino alla locazione 56295 (\$DBE7).

Ognuna di queste locazioni colore in RAM e' grande 4 bits il che consente di poter manipolare numeri da 0 a 15. Per questo e' possibile sfruttare i 16 colori disponibili sul CBM64.

In piu' ci sono ben 256 caratteri che possono essere visualizzati in qualsiasi momento.

Per la normale visualizzazione, ciascuna delle 1000 locazioni della memoria di schermo contiene un numero di codice che comunica all' integrato VIC-II quale carattere visualizzare in quella locazione.

I vari modi grafici sono selezionati da ben 47 Registri di controllo dell' integrato 6567. Molte funzioni grafiche possono essere controllate e manipolate POKEggiando un determinato valore entro uno di questi registri.

Questi registri e quindi in pratica gli indirizzi dell' integrato stesso vanno dalla locazione decimale 53248 (\$D000) fino alla 53294 (\$D02E).

## NOTA

Si consiglia di vedere le tavole di memoria riportate in appendice per un' esatta dislocazione e distribuzione di queste ed altre zone di memoria e per il loro utilizzo.

## SELEZIONE DEI BANCHI DI MEMORIA

Il VIC-II puo' accedere ("VEDERE") a 16k-Bytes di memoria contemporaneamente.

Dato che sul CBM64 ci sono 64k di memoria come e' possibile accedere a tutte?

La memoria in effetti e' divisa in 4 possibili BANCHI o SEZIONI di 16k-Bytes ciascuna.

Il BANK SELECT BIT che consente di accedere a tutte le differenti sezioni di memoria e' localizzato nel:

## 6526 COMPLEX INTERFACE ADAPTER CHIP 2

(CIA 2).

I comandi Basic PEEK e POKE ( o la loro versione in linguaggio macchina) sono usati per selezionare un banco controllando i bits 0 e 1 della porta A del CIA 2 ( locazione decimale 56576 ( o \$DD00)).

Questi due bit devono essere fissati per consentire il cambiamento dei banchi di memoria.

Vediamo l' esempio e la tabella:

POKE56578,PEEK(56578) OR 3:REM CONTROLLO CHE I BITS 0 E 1 SIANO FISSATI PER L' OPERAZIONE.

POKE56576,(PEEK(56576)AND252)OR A: REM CAMBIO DEI BANCHI DI MEMORIA

Dove la variabile A deve avere uno dei seguenti valori:

| VALORE<br>DI A | BITS | BANK | INIZIO | RANGE<br>VIC-II |
|----------------|------|------|--------|-----------------|
| 0              | 00   | 3    | 49152  | \$C000-\$FFFF   |
| 1              | 01   | 2    | 32768  | \$8000-\$BFFF   |
| 2              | 01   | 1    | 16384  | \$4000-\$7FFF   |
| 3              | 11   | 0    | 0      | \$0000-\$3FFF   |

Questo concetto di banchi a 16k e' una parte importante del funzionamento del VIC-II, ed e' quindi importante impararlo per conoscere in quale banco di memoria ci troviamo.

Ricordiamo che all' accensione del CBM64 i bits 0 e 1 della locazione 56576 sono automaticamente fissati sul BANK 0 cioe' di indirizzo \$0000-\$3FFF.

## MEMORIA DI SCHERMO.

La locazione di memoria di schermo puo' essere facilmente cambiata con un comando POKE al registro di controllo di indirizzo 53272 (\$D018).

Tuttavia questo registro e' anche usato per controllare quale set di caratteri e' usato per cui e' necessario porre una particolare cura per non disturbare questa parte del registro di controllo.

I quattro bits piu' alti controllano la locazione della memoria di schermo.

Per muovere lo schermo si puo' usare il seguente gruppo di comandi:

POKE53272,(PEEK(53272)AND15)OR A

Dove la variabile A puo' avere uno dei seguenti valori:

| A   | BITS     | LOCAZIONE<br>DECIM. | ESA    |
|-----|----------|---------------------|--------|
| 0   | 0000XXXX | 0                   | \$0000 |
| 16  | 0001XXXX | 1024                | \$0400 |
| 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 |

## ATTENZIONE

Ricordarsi che il BANK ADRSS del VIC-II deve essere aggiunto.

## MEMORIA COLORE

La memoria colore non puo' essere mossa. Essa e' sempre localizzabile da 55296 a 56295 (\$D800-DBE7).

La memoria di schermo, cioe' le 1000 locazioni che partano dall' indirizzo 1024, e la memoria di colore sono usate in maniera diversa nei differenti modi grafici.

Infatti un disegno creato in un modo grafico apparira' completamente diverso in un' altro modo grafico.

## MEMORIA CARATTERE

Il punto esatto in cui il VIC-II prende le informazioni per il carattere, conosciuto anche MAPPA DEI CARATTERI poiche' riporta il disegno di ogni singolo carattere disponibile del SET standard e' molto importante per la gestione della grafica in particolare e della programmazione in generale.

Normalmente il disegno del carattere che si desidera visualizzare e' prelevato dal:

## CHARACTER GENERATOR ROM

In questo integrato sono immagazzinati i disegni relativi ai vari caratteri, siano questi lettere, numeri, punteggiature e comunque tutto cio' che potete osservare sulla tastiera.

Una delle caratteristiche del CBM64 e' la capacita' di

utilizzare disegni presenti nella memoria RAM.

Chiaramente questi disegni su RAM sono creati da voi e cio' consente che si possa utilizzare una varieta' pressoché infinita di simboli grafici, caratteri speciali sia per applicazioni di giochi ma anche scientifiche come potrebbero essere i simboli matematici speciali o, ad esempio, i caratteri dell' alfabeto greco.

Un normale set contiene 256 caratteri in cui ognuno è definito da 8 Bytes.

La memoria occupata da un set completo sarà quindi di  $256 \times 8 = 2k$  Bytes.

Poiché come abbiamo detto il VIC-II può controllare un banco di memoria di 16k per volta vorrà dire che potranno essere disponibili un massimo di 8 set completi per volta.

Naturalmente non è necessario riempire completamente il set ma sarà comunque necessario partire da una delle locazioni disponibili.

La posizione della memoria di carattere è controllata da 3 bits del registro di controllo che ha come indirizzo 53272 (\$D018). I bits 3, 2, 1 controllano dove è localizzato il set di caratteri nei 2k di memoria, mentre il bit 0 viene ignorato.

Ricordiamo che questo è lo stesso registro che determina dove è localizzata la memoria di schermo e di cui precedentemente abbiamo utilizzato i bits più alti, mentre avevamo messo una X nei bits che utilizziamo ora. Per cambiare la locazione della memoria carattere, può essere usato il seguente comando Basic:

```
POKE53272,(PEEK(53272)AND240) OR A
```

Dove A può assumere uno dei seguenti valori:

| VALORE<br>DI A | INDIRIZZI DEI CARATTERI |        |               |
|----------------|-------------------------|--------|---------------|
|                | BITS                    | DECIM. | ESA           |
| 0              | XXXX000X                | 0      | \$0000-\$07FF |
| 2              | XXXX001X                | 2048   | \$0800-\$0FFF |
| 4              | XXXX010X                | 4096   | \$1000-\$17FF |
| 6              | XXXX011X                | 6144   | \$1800-\$1FFF |
| 8              | XXXX100X                | 8192   | \$2000-\$27FF |
| 10             | XXXX101X                | 10240  | \$2800-\$2FFF |
| 12             | XXXX110X                | 12288  | \$3000-\$37FF |
| 14             | XXXX111X                | 14336  | \$3800-\$3FFF |

Quello che abbiamo definito come ROM IMAGE nella precedente tavola si riferisce al generatore di caratteri in ROM ed appare al posto della RAM nella suddetta locazione nel BANK 00. Compare anche nella corrispondente RAM alla locazione da 36864 a 40959 (\$9000-\$9FFF).

La locazione ed il contenuto del set di caratteri presente in ROM e' quello riportato in tabella:

| INDIRIZZO |           | VIC-II    | CONTENUTO                      |
|-----------|-----------|-----------|--------------------------------|
| DEC.      | ESA       | IMMAG.    |                                |
| 53248     | D000-D1FF | 1000-11FF | Caratteri maiuscoli            |
| 53760     | D200-D3FF | 1200-13FF | Caratteri grafici              |
| 54272     | D400-D5FF | 1400-15FF | Maiuscoli in reverse           |
| 54784     | D600-D7FF | 1600-17FF | Grafici in reverse             |
| 55296     | D800-D9FF | 1800-19FF | Minuscoli                      |
| 55808     | DA00-DBFF | 1A00-1BFF | Maiuscoli e grafici            |
| 56320     | DC00-DDFF | 1C00-1DFF | Minuscoli in reverse           |
| 56832     | DE00-DFFF | 1E00-1FFF | Maiuscoli e grafici in reverse |

Le locazioni occupate dal CHARACTER ROM sono le stesse che occupano i registri di controllo del VIC-II. Cio' e' ovviamente possibile perche' l' occupazione delle stesse locazioni non avviene nello stesso tempo.

Quando al VIC-II necessita di accedere ai dati dei caratteri viene attivata la ROM. I dati necessari sono trasferiti nel banco di memoria da 16 k che il VIC-II tiene sotto controllo in quel momento.

In caso contrario questa area e' occupata dai registri di controllo dell'I/O.

In questo caso si devono disabilitare i registri di I/O, abilitare la ROM CHARACTER ed effettuare la copia. Alla fine eseguire l' operazione inversa rimettendo in funzione i registri di I/O.

Durante il processo di copia ( quando i registri di I/O sono disabilitati) non e' ammesso l' uso di INTERRUPTS. Se dimenticate questo particolare possono succedere delle strane cose.

Anche la tastiera deve essere disabilitata durante il processo di copia.

Per disabilitare la tastiera e le altre normali funzioni di Interrupt usare i seguenti comandi:

```
POKE56334,PEEK(56334) AND 254
```

che disabilitera' le funzioni di Interrupt.

```
POKE56334,PEEK(56334) OR 1
```

che riabilita le funzioni di Interrupt.

Questa ultima funzione deve essere quindi utilizzata per continuare nell' esecuzione del programma e per riabilitare la tastiera quando e' terminata la funzione di caricamento dei caratteri da ROM.

#### NOTA

Per un approfondimento in particolare del significato ed uso degli Interrupt si consiglia di consultare un manuale di programmazione Assembler e ci permettiamo di consigliare l' INTRODUZIONE ALL' ASSEMBLER ed. COMMODORE oppure MANUALE ASSEMBLER PER CBM 64 ed.EVM.



## CAPITOLO NONO

### MODO STANDARD DEI CARATTERI

Quando il COMMODORE 64 viene acceso ci troviamo nel modo di caratteri standard che e' quello che si adopera per la programmazione.

Come abbiamo detto i caratteri possono essere prelevati sia da ROM che da RAM, ma normalmente sono prelevati da ROM.

Quando si desiderano speciali caratteri grafici per un programma e' necessario definirli in RAM e segnalare al VIC-II di prelevarli di qui anziche' dalla ROM.

Tutto cio' sara' spiegato in dettaglio nella seguente sezione.

Per visualizzare i caratteri a colori, il VIC-II accede alla memoria di schermo per determinare il codice del carattere per quella data locazione sullo schermo. Allo stesso tempo accede alla memoria di colore per definire in quale colore si desidera visualizzare quel carattere. Il codice del carattere e' trasferito dal VIC-II entro l'indirizzo d'inizio del blocco di 8 byte che e' localizzato nella memoria di carattere.

Questo trasferimento non e' eccessivamente complicato ma sono necessari alcuni comandi per generare l'indirizzo desiderato.

Per prima cosa il codice del carattere che si usa per eseguire un POKE sulla memoria di schermo deve essere moltiplicato per 8. E' necessario poi aggiungere l'inizio della memoria del carattere (Vedi la parte del CHARACTER MEMORY).

Ecco una semplice formula che illustra cosa succede:

INDIRIZZO DEL CARATTERE = CODICE DI SCHERMO x 8 + ( SET  
DEI CARATTERI x 2048) + (BANK SELECT BITS x 16384)

## DEFINIZIONE DEL CARATTERE

Ogni carattere puo' essere visto come contenuto in una griglia di 8 x 8 punti e dove ogni punto puo' essere OFF o ON. Le immagini di ogni singolo carattere del CBM 64 sono contenute in ROM. Qui dentro i caratteri sono immagazzinati a gruppi di 8 bytes per ogni carattere. Immaginando una griglia di 8 x 8, ogni Byte rappresenta una riga della griglia e ogni bit un punto entro la griglia.

Quando il relativo bit sara' a 1 vorra' dire che il punto e' attivato (ON) e visibile, mentre quando e' a zero e' disattivato (OFF) e quindi invisibile.

La memoria di caratteri in ROM inizia dalla locazione 53248 ( quando i registri di I/O sono disattivati come detto in precedenza ).

Quindi i primi 8 bytes dalla locazione 53248 (\$D000) alla 53255 (\$D007) contengono il disegno della A commerciale che ha un codice di carattere del valore 0 nella memoria di schermo. I successivi 8 bytes dalla locazione 53256 alla 53263 (\$D008-\$D00F) contengono il disegno della lettera A.

| INDIR. | IMMAGINE | BINARIO  | PEEK |
|--------|----------|----------|------|
| 53256  | XX       | 00011000 | 24   |
| 53257  | XXXX     | 00111100 | 60   |
| 53258  | XX XX    | 01100110 | 102  |
| 53259  | XXXXXX   | 01111110 | 126  |
| 53260  | XX XX    | 01100110 | 102  |
| 53261  | XX XX    | 01100110 | 102  |
| 53262  | XX XX    | 01100110 | 102  |
| 53263  |          | 00000000 | 0    |

Ricordiamo che ogni set completo di caratteri occupa 2K di memoria perche' e' formato da 8 bytes per carattere e da 256 caratteri.

Poiche' ci sono in ROM due set completi di caratteri, uno per le maiuscole e la grafica e l' altro per le maiuscole e le minuscole la ROM CHARACTER GENERATOR sara' di 4K.

## CARATTERI PROGRAMMABILI

Come abbiamo accennato in precedenza, per creare un nuovo set di caratteri alternativo a quello standard fornito dal Sistema Operativo, e' necessario operare sui registri programmabili dell' integrato VIC-II.

Il nuovo o i nuovj set di caratteri possono essere caricati in un punto qualsiasi della memoria RAM.

Tuttavia ci sono almeno due punti della memoria che NON dovrebbero essere usati quando si programma in Basic: le locazioni che partano da 0 e quelle da 2048.

Le prime locazioni di memoria non devono essere usate perche' il Sistema Operativo immagazzina importantissime informazioni appunto in pagina ZERO e la seconda perche' e' da qui che inizia il programma Basic.

La miglior posizione per inserire il nuovo set di caratteri, per essere poi usato entro un programma Basic, almeno a livello sperimentale e' di far iniziare il caricamento a partire dalla locazione decimale 12288 (Esa \$3000).

Questo puo' essere fatto con la seguente istruzione:

```
POKE 53272, (PEEK(532772) AND 240)+12
```

Per riportare alla visualizzazione normale il CBM64 premere:RUN/STOP e RESTORE.

Proviamo ora a creare dei nuovi caratteri grafici.  
Innanzitutto per proteggere il nuovo set di caratteri da

eventuali sovrascritture del Basic si deve ridurre la memoria.

Intendiamoci, non e' che si tolga memoria al Computer!!! Semplicemente si abbassano i puntatori di fine programma Basic in modo tale che il programma che scriveremo non vada ad occupare e quindi a sovrascrivere il nuovo set di caratteri.

Digitiamo:

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

Il numero visualizzato in risposta a questa interrogazione e' l' ammontare di spazio di memoria libero per i programmi. Digitiamo ora:

```
POKE 52,48 : POKE 56,48 :CLR
```

ed ora di nuovo:

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

Se osservate i due dati vedrete che sono diversi, nel senso che il risultato della prima interrogazione e' superiore alla seconda ed il Basic non utilizzerà quella parte di memoria che rimane così disponibile per scrivere il nuovo set di caratteri.

Il passo successivo e' di immettere i caratteri in RAM. Quando si inizia questa operazione ricordiamoci che le locazioni RAM sulle quali caricheremo ora i nuovi o vecchi caratteri di inizio 12288 (\$3000) contengono valori casuali.

Il seguente programma sposta 64 caratteri dalla memoria ROM nella RAM che utilizzeremo:

```

5 PRINT CHR$(142)
10 POKE 52,48 : POKE56,48:CLR
20 POKE56334,PEEK(56334) AND 254
30 POKE1,PEEK(1) AND 251
40 FOR I=0TO511: POKEI+12288, PEEK(I+12288) :NEXT
50 POKE1,PEEK(1) OR 4
60 POKE56334,PEEK(56334) OR 1
70 END

```

## COMMENTO

```

LINEA 5 Passa al maiuscolo
LINEA 10 Riserva una zona di memoria per i caratteri
LINEA 20 Disabilita l' Interrupt
LINEA 30 Switch dei caratteri
LINEA 40 Carica i caratteri
LINEA 50 Switch di I/O
LINEA 60 Riabilita l' Interrupt

```

Eseguiamo ora un POKE alla locazione 53272 con:

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

Vediamo pero' che non accade niente. Perche'?

Il CBM64 sta prendendo ora le informazioni sui caratteri non piu' da ROM ma da RAM, ma poiche' i caratteri sono stati integralmente copiati e NON ricreati non appare niente di particolare.

Ora' pero' che abbiamo il set di caratteri in RAM possiamo facilmente cambiarli. Vediamo come.

Eseguiamo per prima cosa un clear di schermo e digitiamo il carattere corrispondente alla a commerciale (ochiocciola). Muoviamoci poi con il cursore due spazi piu' in basso e scriviamo:

FORI=12288 TO 12288+7:POKEI,255-PEEK(I):NEXT

dopo il RETURN comparirà lo stesso carattere visualizzato in precedenza ma in REVERSE.

Per ritornare al carattere normale è sufficiente ridare un RUN al programma.

Con il sistema appena visto e ricordando che ogni carattere è una matrice di 8x8, vediamo come è possibile creare un nuovo carattere.

Per prima cosa facciamo girare il precedente programma e poi digitiamo NEW.

Prendiamo poi un foglio di carta a quadretti e disegniamo un quadrato di 8 x 8 con i riferimenti sottodescritti:

|     | 7<br>(128) | 6<br>(64) | 5<br>(32) | 4<br>(16) | 3<br>(8) | 2<br>(4) | 1<br>(2) | 0<br>(0) |
|-----|------------|-----------|-----------|-----------|----------|----------|----------|----------|
| R.0 |            |           | X         | X         | X        | X        |          |          |
| R.1 |            | X         |           |           |          |          | X        |          |
| R.2 | X          |           | X         |           |          | X        |          | X        |
| R.3 | X          |           |           |           |          |          |          | X        |
| R.4 | X          |           | X         |           |          | X        |          | X        |
| R.5 | X          |           |           | X         | X        |          |          | X        |
| R.6 |            | X         |           |           |          |          | X        |          |
| R.7 |            |           | X         | X         | X        | X        |          |          |

Eseguiamo poi la somma dei corrispondenti valori tra parentesi riga per riga.

Così per la prima riga (riga 0) avremo:

$$32+16+8+4 = 60$$

per la seconda riga (riga 1)

$$64+2 = 66$$

e così via fino al termine

Digitiamo poi il seguente programma di due sole righe nel quale immetteremo i valori dei DATA le somme ottenute riga per riga:

```
10FOR I=12448 TO 12445: READ A:POKE I,A:NEXT
20DATA 60,66,165,129,165,153,66,60
```

Dopo aver dato il RUN vedremo che al premere della lettera T apparirà il carattere disegnato nella nostra matrice.

E' successo infatti che nel nostro set di caratteri alla lettera ( o meglio alla matrice di punti che corrisponde alla formazione della lettera) T e' stata sostituita una nuova serie di valori.

A scopo di esercizio vi consigliamo di cambiare altri caratteri utilizzando anche set diversi.

## NOTA

E' disponibile un programma di nome CHARACTER GENERATOR che consente di creare rapidamente ed a colori un numero praticamente infinito di caratteri, di salvarli su cassetta o su disco, di posizionarli in particolari aree di memoria e di selezionarli quando si desidera.

## GRAFICA IN MODO MULTICOLORE

L' alta risoluzione grafica STANDARD abilita il controllo di tutti i punti dello schermo.

Ogni punto nella memoria di carattere puo' avere 2 possibili valori:

1 = ATTIVO (ON)                      = Il punto appare

2 = NON ATTIVO (OFF) = Non appare

Quando il punto e' in OFF, per riempire comunque quello spazio vuoto viene utilizzato il colore di schermo.

Se invece il registro relativo al punto e' attivato (ON), questi assume il colore del carattere scelto per quella posizione di schermo.

Quando si sta usando l' alta risoluzione grafica standard, tutti i punti entro la griglia del carattere ( ricordarsi che e' una griglia di 8x8 punti ), possono



avere un colore interno o esterno.  
Possono pero' esserci dei problemi quando si incrociano 2 diverse linee di colori. Con il modo multicolore si ha una soluzione a questo tipo di problemi.  
Infatti ogni punto dello schermo in modo multicolore puo' avere 4 colori:

COLORE DI SCHERMO

COLORE DEL REGISTRO 1

COLORE DEL REGISTRO 2

COLORE DEL CARATTERE

La sola perdita e' nella risoluzione orizzontale perche' ogni punto in modo MULTI-COLORE e' due volte piu' largo di un punto in ALTA RISOLUZIONE.  
Riteniamo pero' che questa piccola perdita di risoluzione sia ampiamente compensata da cio' che e' possibile poi fare in modo multicolore.

## MULTI-COLORE

Per passare al modo multicolore e' necessario mettere a 1 il bit 4 del registro di controllo 53270 (\$D016) con il seguente comando:

POKE 53270,PEEK(53270) OR 16

Per uscire dal modo MULTICOLORE sara' necessario quindi riportare a 0 lo stesso quarto bit del registro 53270 con il seguente comando di POKE:

POKE 53270,PEEK(53270) AND 239

Il modo MULTICOLORE puo' essere infatti abilitato o disabilitato per qualsiasi punto dello schermo in modo tale che si possa utilizzare contemporaneamente il MULTICOLORE e l' ALTA RISOLUZIONE (HI-RES).

Questo processo e' controllato dal bit 3 della memoria di colore.

La memoria di colore inizia dalla locazione 55296 (\$D800).

Se il numero nella memoria di colore e' minore di 8 (cioe' va da 0 a 7) allora il corrispondente spazio sullo schermo sara' in ALTA RISOLUZIONE standard e nel colore da 0 a 7 scelto.

Se invece il numero presente nella memoria colore e' uguale o maggiore di 8 (cioe' da 8 a 15 ) allora questo spazio sara' visualizzato in modo MULTICOLORE.

Eseguendo un comando POKE di un numero apposito entro la memoria di colore, si puo' cambiare il colore del carattere in quella data posizione di schermo.

Con un POKE di un numero da 0 a 7 diamo un normale colore al carattere, mentre con un POKE fra 8 e 15 fisseremo il modo multicolore.

In altre parole mettendo a 1 il Bit 3 (ON), sceglieremo il modo MULTICOLORE, mentre mettendo lo stesso Bit a 0 (OFF) sceglieremo l' ALTA RISOLUZIONE.

Vediamo l' immagine della lettera A:

| IMMAGINE | BIT PATTERN |
|----------|-------------|
| **       | 00011000    |
| ****     | 00111100    |
| ** **    | 01100110    |
| *****    | 01111110    |
| ** **    | 01100110    |
| ** **    | 01100110    |
| ** **    | 01100110    |
|          | 00000000    |

Il colore di schermo e' visualizzato dove i Bits sono a 0, mentre il colore del carattere e' visualizzato dove i Bits sono a 1.

Il modo MULTICOLORE usa i Bits in coppia nel modo seguente:

| IMMAGINE | BIT PATTERN |
|----------|-------------|
| AABB     | 00 01 10 00 |
| CCCC     | 00 11 11 00 |
| AABBAABB | 01 10 01 10 |
| AACCCBBB | 01 11 11 10 |
| AABBAABB | 01 10 01 10 |
| AABBAABB | 01 10 01 10 |
| AABBAABB | 01 10 01 10 |
|          | 00 00 00 00 |

Nella figura appena vista gli spazi AA assumeranno il colore BACKGROUND 1, gli spazi BB e gli spazi CC useranno il colore del carattere.

Questo viene determinato dall' uso delle coppie di Bits secondo la seguente tabella:

| COPPIE REGISTRO                      | COLORE | INDIRIZZO     |
|--------------------------------------|--------|---------------|
| 00 Background<br>(colore di schermo) | 0      | 53281(\$D021) |
| 01 "                                 | 1      | 53282(\$D022) |
| 10 "                                 | 2      | 53283(\$D023) |
| 11 Vedi nota                         |        | RAM           |

NOTA

Colore specificato dal contenuto dei 3 bits piu' bassi della memoria di colore.

Digitiamo il seguente programma dimostrativo:

```
100 POKE 53281,1
110 POKE 53282,3
120 POKE 53283,8
130 POKE53270,PEEK(53270) OR16
140 C=13x4096+8x256
150 PRINTCHR$(147) "AAAAAAA"
160 FOR L=0TO9
170 POKE C+L,8
180 NEXT
```

Provate a far girare questo programma e vedrete che lo schermo sara' bianco, il colore del carattere nero mentre un registro colore CYAN e l' altro arancio.

Come esercizio vi consigliamo di completare il precedente programma utilizzando i caratteri programmabili, e cambiando i colori dei vari registri.

Un' altro metodo per cambiare i colori e' l' uso del tasto Commodore per mettere in funzioni i tasti numerici sotto i quali ( cioe' nella loro parte inferiore) sono scritti i colori utilizzabili in forma abbreviata.

## MODO DI COLORE EXTENDED BACKGROUND

Questo modo colore abilita il controllo sul colore di sfondo di ogni carattere.

In questo modo si puo' visualizzare per esempio un carattere Bleu con sfondo giallo su schermo bianco.

Ci sono 4 registri disponibili per il modo EXTENDED BACKGROUND ed ognuno di questi registri puo' essere caricato con l' indirizzo di uno dei 16 colori

disponibili consentendo quindi una gamma di scelta particolarmente vasta.

Tuttavia questo modo di rappresentazione ha un limite nel numero dei diversi caratteri che si possono visualizzare. Infatti quando si usa il modo EXTENDED BACKGROUND solo i primi 64 caratteri nella memoria ROM ( oppure i primi 64 caratteri del set costruito e quindi presente in RAM ) possono essere usati. E cio' perche' due dei bits del codice di carattere sono utilizzati per selezionare il colore di sfondo.

Il codice del carattere della lettera A e' 1. Quando e' attivato il modo EXTENDED BACKGROUND se si esegue un POKE del valore 1 sullo schermo, dovrebbe essere visualizzata una A.

Se si esegue un POKE di 65 dovrebbe invece apparire una A in reverse. Cio' invece non succede nel modo di colore EXTENDED BACKGROUND.

La seguente tavola mostra l' uso dei codici:

| CODICE CARATTERE |      |      | REG. COLORE BACKGROUND |               |
|------------------|------|------|------------------------|---------------|
| RANGE            | BIT7 | BIT6 | NUM.                   | 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) |

Per attivare il modo EXTENDED BACKGROUND e' necessario mettere a 1 il sesto bit del registro di indirizzo 53265 (\$D011) dell' integrato VIC-II con il comando:

POKE 53265, PEEK(53265) OR 64

Mentre per disattivarlo occorre mettere a 0 lo stesso VI bit con l' istruzione:

POKE 53265, PEEK(53265) AND 191

## IL BIT MAPPING NELLA GRAFICA

Quando si scrivono giochi, si disegnano tabelle per le applicazioni commerciali o altri tipi di programmi prima o poi si arriva al punto di utilizzare l' ALTA RISOLUZIONE GRAFICA, ed il CBM64 e' stato costruito per farlo.

L' HIGH-RESOLUTION e' disponibile ed utilizzabile attraverso la mappa di memoria di bit dello schermo (BIT MAPPING).

Il BIT MAPPING e' il metodo in cui ad ogni possibile punto che puo' apparire sullo schermo e' assegnato un bit della memoria.

Se il contenuto di questo bit e' 1 allora il punto sara' visualizzato, mentre invece se il bit ha valore 0 il punto non appare.

Il disegno in HIGH-RESOLUTION ha una gestione dei bit che molto particolare ma che comunque non utilizza tutta la memoria teorica allo stesso tempo.

Poiche' ogni carattere e' 8x8 e ci sono 40 linee con 25 caratteri per ogni linea la risoluzione e' di 320x200 PIXELS per l' intera superficie di schermo.

Cio' da un totale di 64000 punti separati ognuno dei quali richiedera' 1 bit di memoria.

In altre parole sono necessari 8000 bytes per avere la mappa di memoria di tutto lo schermo.

Generalmente le operazioni in alta risoluzione sono eseguite da semplici e ripetitive routines, ma sfortunatamente questo modo di operare e' troppo lento quando si opera in ambito Basic.

Al contrario questo modo di scrivere routines di gestione, che cioe' siano CORTE, SEMPLICI e RIPETITIVE e' quanto di meglio possa dare il Linguaggio Macchina.

La soluzione migliore quindi ci sembra quindi di scrivere il programma generale in Basic e di inserire Subroutines per la grafica entro questo programma con dei DATA oppure di memorizzare queste routines in Linguaggio Macchina e di richiamarle con dei SYS opportuni dal programma principale.

Cio' consentira' di accoppiare la facilita' della

programmazione in Basic alla velocita' tipica di esecuzione del Linguaggio Macchina.

Il BIT MAPPING e' una delle tecniche grafiche piu' popolari ed e' usata per creare disegni con alto dettaglio di risoluzione.

Quando siamo in questo modo si puo' controllare direttamente DOVE si trova e se e' attivato un singolo punto sullo schermo.

Ci sono due tipi di BIT MAPPING disponibili sul CBM64:

- Lo STANDARD ( alta risoluzione ) BIT MAP MODE con una risoluzione di 320x200.

- Il MULTI-COLOR BIT MAP MODE con 160x200 punti di risoluzione.

#### NOTA

Sono in commercio alcuni package grafici che consentono di gestire l' alta risoluzione in modo semplice e che aggiungono una serie di comandi.

#### STANDARD BIT MAP MODE

Questo modo di risoluzione vi da la possibilita' di operare su 320 punti orizzontali per 200 verticali con una scelta di due colori in ogni sezione di 8x8.

Lo STANDARD HIGH RESOLUTION BIT MAP MODE e' attivato mettendo ad 1 il bit 5 del registro di controllo 53265 (\$D011) con il seguente comando:

```
POKE53265, PEEK(53265) OR 32
```

e viene disattivato mettendo a 0 lo stesso bit con il comando:

```
POKE53265, PEEK(53265) AND 233
```

## MODO DI COLORE MULTI-COLOUR BIT MAP

Allo stesso modo del MULTI COLOUR CHARACTER MODE il MULTI-COLOUR BIT MAP consente di visualizzare fino a 4 diversi colori in ogni sezione 8 x 8 della mappa dei bits.

Anche qui tuttavia dobbiamo sopportare una minore risoluzione nella definizione orizzontale. Si passa infatti da 320 a 160 punti disponibili in orizzontale.

Inoltre questo modo o metodo usa 8 K Bytes di memoria per la mappa dei bits.

Per attivarlo e' necessario mettere ad 1 il bit 5 del registro di locazione 53265 (\$D011) ed il bit 4 del registro di indirizzo 53270 (\$D016) con i seguenti comandi:

```
POKE 53264,PEEK (53265) OR 32
```

```
POKE 53270,PEEK (53270) OR 16
```

Mentre per disattivarlo occorre mettere a 0 i bit anzidetti con i comandi :

```
POKE 53265,PEEK (53265) AND 223
```

```
POKE 53270,PEEK (53270) AND 239
```

Come abbiamo detto in questo modo grafico la risoluzione e' piu' bassa , in orizzontale, a causa della gestione colore che avviene, per ogni due bits, nel modo seguente:

| BITS | PROVENIENZA INFORMAZIONI |
|------|--------------------------|
|------|--------------------------|

|    |                   |
|----|-------------------|
| 00 | Colore di schermo |
|----|-------------------|

|    |                                      |
|----|--------------------------------------|
| 01 | 4 bits alti della memoria di schermo |
|----|--------------------------------------|



- 10 4 bits bassi della memori di schermo
- 11 Nybble di colore

## NOTA

Ricordiamo che un Nybble equivale a 4 bits o mezzo Byte.

## LO SCROLLING LENTO

L' integrato VIC-II consente uno scrolling lento dello schermo, sia in orizzontale che in verticale. Questo tipo di movimento viene effettuato spostando l' intero schermo a destra o a sinistra, in alto o in basso di un solo punto per volta.

Mentre il VIC-II esegue una gran parte del lavoro, lo scrolling attualmente deve essere fatto agendo con programmi in codice macchina.

L' integrato ha la capacita' di mettere lo schermo video in una qualsiasi delle otto posizioni orizzontali o verticali.

Esiste un modo per cosi' dire ridotto di schermo che consta di 38 colonne per 24 righe.

Questo PICCOLO SCHERMO e' utilizzato per riservare un po' di posto per i data dello scrolling.

Vediamo i passi necessari per eseguire queste funzioni sia da un punto di vista della logica della programmazione sia per quanto riguarda i comandi necessari. Al termine verra' descritto un breve esempio.

1) Stringere lo schermo (si espande quindi il bordo).

2) Fissare il registro di scrolling al massimo ( o al minimo in dipendenza dalla direzione dello scrolling

stesso).

3) Immettere i nuovi data nella giusta ( coperta) porzione di schermo.

4) Incrementare ( o decrementare ) il registro di scrolling fino a trovare il massimo (o il minimo) valore.

5) A questo punto utilizzare la routine in linguaggio macchina per spostare l' intero schermo di un carattere nella direzione di scroll.

6) Tornare al passo 2

Vediamo ora i comandi necessari per la riduzione dello schermo.

Per immettersi nel modo 38 colonne, deve essere messo a 0 il bit 3 della locazione di memoria 53270 (\$D016) con il seguente comando di POKE:

POKE 53270, PEEK (53270) AND 247

Per tornare alle 40 colonne normali naturalmente occorre riportare ad uno quel bit stesso con:

POKE 53270, PEEK (53270) OR 8

Per immettersi nel modo 24 righe, deve essere messo a zero il bit 3 della locazione 53265 (\$D011) con il seguente comando POKE:

POKE 53265, PEEK (53265) AND 247

Anche in questo caso per ritornare alle normali 25 righe rimettere ad 1 lo stesso bit con:

POKE 53265, PEEK (53265) OR 8

Quando si esegue lo scrolling nella direzione dell' asse X o da destra a sinistra o viceversa o comunque in direzione orizzontale e' necessario eseguire le operazione per porsi nel modo 38 colonne.

Quando si esegue lo scrolling a sinistra i nuovi dati dovrebbero essere immessi sulla destra, mentre quando si esegue lo scrolling a destra i nuovi dati dovrebbero essere immessi nella direzione opposta.

#### NOTA

Ricordarsi che nella memoria video ci sono SEMPRE 40 colonne, solo che in questo modo solo 38 sono visibili.

Quando si esegue lo scrolling in direzione Y o verticale e' necessario selezionare il modo 24 righe con i comandi visti in precedenza.

Di conseguenza quindi al ragionamento fatto a proposito dello scrolling orizzontale ne deriva che quando si esegue lo spostamento in ALTO sara' necessario immettere i nuovi dati nell' ultima riga.

Al contrario, quando si esegue lo scrolling in BASSO i dati saranno nella prima riga.

#### NOTA

E' da notare che mentre nello scrolling orizzontale vengono eliminate due colonne in quello verticale viene eliminato ( o meglio nascosta) solo una riga.

Per eseguire lo scrolling in direzione X o orizzontale,

il registro e' localizzabile nei bits da 2 a 0 del registro di controllo del VIC-II di indirizzo 53270 (\$D016).

Al solito e' necessario operare solo su questi bits con la seguente istruzione:

```
POKE 53270, (PEEK(53270) AND 248) + X
```

dove X e' la posizione (X) dello schermo da 0 a 7.

Per eseguire lo scrolling in direzione Y o verticale il relativo registro e' localizzabile nei bits da 2 a 0 del registro di controllo del VIC-II di indirizzo 53265 (\$D011).

Il comando relativo a questo registro e':

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

dove Y e' la posizione (Y) dello schermo variabile da 0 a 7

Vediamo ora un piccolo programma per l' uso di questa opzione grafica:

```
10 POKE53265, PEEK(53265) AND 247
20 PRINTCHR$ (147)
30 FOR X =1 TO24:PRINTCHR$ (17);:NEXT
40 POKE53265,(PEEK(53264)AND248)+7: PRINT
50 PRINT"    BUONGIORNO"
60 FOR P=6TO0 STEP-1
70 POKE 53265,(PEEK(53265)AND248)+P
80 FOR X=1TO50:NEXT
90 NEXT: GOT040
```

## CAPITOLO DECIMO

### GLI SPRITES

Uno Sprite e' uno speciale tipo di carattere definibile che puo' essere posizionato in una qualsiasi parte dello schermo.

Questa straordinaria opzione grafica e' gestita interamente dall' integrato VIC-II.

COME DEVE ESSERE L' IMMAGINE

DI QUALE COLORE

DOVE DEVE COMPARIRE

sono le uniche informazioni da dare al computer. Al resto pensa l' integrato VIC-II.

Insieme agli Sprites si possono usare sia tutti e 16 i colori disponibili, sia tutti gli altri modi grafici.

Si possono avere contemporaneamente fino a 8 immagini Sprites, ma in effetti, usando la tecnica del RASTER INTERRUPT, che vedremo in seguito, questo numero non costituisce un limite insormontabile.

Le principali caratteristiche degli Sprites sono:

- 1) Definizione dello Sprite in una matrice di 24 x 21 punti.
- 2) Controllo individuale di ogni Sprite.
- 3) Possibilita' di usare il modo MULTICOLOUR.
- 4) Ingrandimento automatico ( con rapporto 2x) in orizzontale, verticale o in entrambe le direzioni.

5) Movimento controllato degli Sprites verso lo sfondo, o bordo.

6) Definizione della priorita' nei movimenti fra le figure.

7) Controllo di incontro ( o scontro) fra gli sprites.

8) Controllo di incontro con priorita' assegnabile e selezionabile fra gli Sprites ed il Bordo schermo.

Tutte queste caratteristiche consentono di programmare delle figure con gli Sprites in maniera completa e veloce.

Anche la grafica relativa ai giochi, di solito cosi' complessa da manipolare, diviene facile e gestibile dal Basic, grazie alla gestione Hardware degli Sprites direttamente dal sistema.

Come abbiamo detto, possiamo , almeno per ora, occuparci della gestione degli sprites da 0 a 7 (cioe' 8 in tutto) che il sistema ci consente senza particolari tecniche.

## COME DEFINIRE UNO SPRITE

Gli Sprites sono definiti allo stesso modo dei Carattereri Programmabili visti in precedenza. Tuttavia saranno necessari un maggior numero di Bytes di memoria perche' la figura generata da questa grafica e' di dimensioni maggiori del carattere.

Uno Sprite puo' essere composto entro una griglia massima di 24 x 21 punti per un totale quindi di 504 punti (  $24 \times 21 = 504$  ).

Viene quindi ad occupare 63 Bytes di memoria ( 504 diviso 8 = 63 ).

Questi 63 Bytes sono suddivisi e disposti in 21 colonne di 3 bytes ciascuna come nello schema sotto:

|         |         |         |
|---------|---------|---------|
| BYTE 0  | BYTE 1  | BYTE 2  |
| BYTE 3  | BYTE 4  | BYTE 5  |
| BYTE 6  | BYTE 7  | BYTE 8  |
| BYTE 9  | BYTE 10 | BYTE 11 |
| ...     | ...     | ...     |
| ...     | ...     | ...     |
| ...     | ...     | ...     |
| BYTE 60 | BYTE 61 | BYTE 62 |

All' interno di ogni Byte, ognuno degli 8 Bits che lo compongono puo' essere ON cioe' 1 o OFF cioe' 0.

Se il bit relativo e' a 0 cioe' disabilitato, allora quel punto dello Sprite sara' trasparente sullo sfondo dello schermo e percio' invisibile.

Se il bit e' a 1 (ON) allora' sara' visualizzato con il colore selezionato.

Questo avviene quando si usano gli Sprites nel modo STANDARD che come possiamo notare e' simile al modo di caratteri standard.

Usando invece gli Sprites in modo multicolore avremo la solita limitazione osservata appunto nel modo MULTICOLORE dei caratteri.

Anche in questo caso infatti ci dovremo accontentare del dimezzamento della risoluzione, disponendo quindi di una matrice non piu' di 24x21, ma di 12x21.

Il numero di colori visualizzabile in ogni Sprite salira' pero' da 1 a 4.

## PUNTATORI DEGLI SPRITES

Come abbiamo appena visto, per definire dal punto di vista grafico uno Sprite, sono necessari 63 Bytes. Poiche' e' tuttavia necessario mettere, e vedremo il perche', un' altro Byte alla fine di ogni figura, lo Sprite occupa in effetti 64 Bytes di memoria. Ad ognuno degli 8 Sprite definibile e' associato un registro di un Byte chiamato:

### SPRITE POINTER

Questo registro ha il compito di controllare in quale locazione di memoria si trovi lo Sprite (cioe' a quale Sprite PUNTA).

Gli SPRITES POINTERS sono localizzabili alla fine del primo KByte della memoria di schermo.

La definizione non deve stupire, perche', mentre alla memoria di schermo e' dedicato effettivamente 1 K Byte, cioe' 1024 Bytes, in effetti lo schermo indirizza, in modo normale, solo 1000 caratteri, essendo infatti una matrice di 25 righe per 40 colonne.

Gli indirizzi della memoria di schermo saranno percio', in modo normale, da 1024 a 2023 ( e quindi non 2047), mentre i puntatori degli Sprites saranno da 2040 a 2047 (\$07F8 a \$07FF).

Poiche' la memoria di schermo puo' essere spostata ( o rilocata), allora lo saranno anche questi puntatori ed e' quindi necessario che il programmatore ne tenga il dovuto conto.

Essendo ogni SPRITE POINTER di 1 Byte puo' assumere un valore da 0 a 255, che stara' ad indicare dove (cioe' da quale locazione di memoria) prendere i 64 Bytes che definiscono lo Sprite stesso.

Se lo Sprite Pointer 0, alla locazione 2040, contiene il numero 14, cio' vuol dire che lo Sprite 0 sara' visualizzato usando i 64 Bytes che iniziano alla locazione  $14 \times 64 = 896$  che e' il Buffer di cassetta. Generalizzando, potremo avere la seguente formula:



LOCAZIONE = (BANCO DI MEMORIA X 16384) + (VALORE DEL PUNTATORE X 64)

Dove per BANCO di memoria si intende un blocco di 16 K Bytes che l' integrato VIC-II puo' controllare . I banchi vanno da 0 a 3.

Quando il nostro integrato e' fissato sui banchi 0 o 2, avremo una immagine ROM del set di carateri presente in alcune locazioni di questi banchi come del resto abbiamo gia' detto prima e come vedremo in seguito.

In questo caso le definizioni degli sprites non possono essere immagazzinate qui.

Se, per una ragione qualsiasi, si necessita di piu' di 128 diverse definizioni o luoghi di definizioni sprites, allora sara' necessario utilizzare uno dei banchi di memoria privi di immagini ROM, cioe' i banchi 1 e 3.

## MESSA IN FUNZIONE DEGLI SPRITES

Il registro di indirizzo 53269 (\$D015) sempre sull' integrato VIC-II, e' noto come:

### SPRITE ENABLE REGISTER

Cioe' come registro che abilita gli sprites.

Questo registro, di 1 Byte cioe' 8 bits, controlla l' attivazione o la disattivazione di ogni Sprite in maniera molto semplice.

Ad ogni bit corrisponde infatti uno sprite, per cui se il bit relativo e' a 0 (OFF), lo sprite e' disabilitato, cioe' non funziona, mentre se e' a 1 (ON) e' in funzione. La disposizione dei bits nel registro corrisponde a quella degli sprite :

\$ D015 BITS 7 6 5 4 3 2 1 0

per cui, per attivare per esempio lo sprite n. 1 sara' necessario mettere in ON ( o al valore 1) il bit corrispondente con l' istruzione:

POKE53269, PEEK(53269) OR 2

Piu' in generale la formula di attivazione di un singolo Sprite sara':

POKE53269, PEEK(53269) OR (2 elevato alla SN)

dove SN e' il numero corrispondente dello Sprite che si vuole attivare e variabile quindi da 0 a 7.

Per disattivare lo stesso registro, cioe' per disabilitare lo sprite, metteremo a 0 il bit relativo, che nel caso dell' esempio precedente sara':

POKE53269, PEEK(53269) AND 253

In generale useremo la seguente formula:

POKE53269, PEEK(53269) AND (255-2elevato a SN)

dove anche in questo caso, SN e' il numero dello sprite sul quale si vuol operare ( o SPRITE NUMBER).

I COLORI NEGLI SPRITES

Uno Sprite puo' avere uno qualsiasi dei colori disponibili sul CBM64.

Per questo motivo ad ogni sprite e' assegnato un registro che contenga l' informazione sul colore con il quale deve essere visualizzato.

La seguente tabella mostra i registri colori assegnati ad ogni Sprite:

| INDIRIZZI    | DESCRIZIONI         |
|--------------|---------------------|
| 53287 \$D027 | R.C. DELLO SPRITE 0 |
| 53288 \$D028 | " 1                 |
| 53289 \$D029 | " 2                 |
| 53290 \$D02A | " 3                 |
| 53291 \$D02B | " 4                 |
| 53292 \$D02C | " 5                 |
| 53293 \$D02D | " 6                 |
| 53294 \$D02E | " 7                 |

Come abbiamo detto in precedenza tutti i punti dello Sprite vengono attivati con il colore scelto, mentre gli altri , cioe' i punti non attivati, saranno trasparenti.

## MODO MULTICOLORE

Questo modo di visualizzazione, a differenza del precedente, nel quale tutta la figura generata dallo Sprite, era in un solo colore consente di avere fino ad un massimo di 4 colori presenti contemporaneamente sulla figura stessa.

Tuttavia, come abbiamo visto nelle altre gestioni grafiche, avremo una perdita del 50 % nella risoluzione orizzontale.

In altre parole, quando si sta lavorando con gli Sprites

in modo multicolore, invece di disporre di 24 colonne di punti se ne dispone di 12 coppie. Ogni coppia di punti e' chiamata BIT PAIR ed ogni BIT PAIR deve essere visto come un singolo punto di risoluzione al quale e' assegnato un colore.

La seguente tabella mostra i valori di BIT PAIR necessari per attivare uno dei quattro colori che si possono scegliere:

| BIT PAIR | DESCRIZIONE                              |
|----------|--|
| 00       | Trasparenza, colore schermo.             |
| 01       | Multicolore, registro 0 (53285 - \$D025) |
| 10       | Registro di colore sprite                |
| 11       | Multicolore, registro 1 (53286 - \$D026) |

#### COME FISSARE LO SPRITE IN MODO MULTICOLORE

Per visualizzare uno sprite nel modo MULTICOLORE occorre attivare un registro ,(mettere in ON) del nostro VIC-II. Si tratta del registro di indirizzo 53276 (\$D01C) che verra' quindi messo in grado di operare con :

POKE 53276, PEEK(53276) OR (2 elevato SN)

dove SN e' il solito Sprite Number che varia da 0 a 7.

Per disattivarlo:

POKE 53276, PEEK(53276) AND (255- 2 elevato SN)

## EXPANDED SPRITES

Sempre via Hardware, cioe' per mezzo dell' integrato VIC-II esiste la capacita di espandere o ingrandire uno Sprite, sia in direzione verticale ( o dell' asse Y o direzione Y) che in direzione orizzontale ( o dell' asse X o direzione X) che in entrambe le direzioni.

Quando si espande una figura, vuol dire che ogni punto raddoppia praticamente in altezza, larghezza o in entrambe le direzioni.

Anche in questo caso esistono 2 registri di 8 bit ciascuno, di cui uno verra' utilizzato per l' espansione dello sprite in orizzontale e l' altro per l' espansione in verticale.

I registri in oggetto sono contenuti nelle locazioni di memoria:

53277 (\$D01D) per l' asse X o orizzontale

53271 (\$d017) per l' asse Y o verticale

Ricordando sempre che SN sta per numero dello sprite sul quale si opera, vediamo le formule per questo tipo di operazioni:

POKE 53277, PEEK(53277) OR (2 elevato a SN)

che serve per espandere lo sprite relativo ( con rapporto 2x) in direzione orizzontale.

POKE 53277, PEEK(53277) AND (255 -2 elevato a SN)

per l' operazione contraria.

POKE 53271, PEEK(53271) OR (2 elevato a SN)

serve per espandere lo Sprite in direzione Verticale o  
asse delle Y.

POKE 53271, PEEK(53271) AND (255 -2 elevato a SN)

anche questo per l' operazione opposta.

## POSIZIONE DEGLI SPRITES

La creazione di una figura, per quanto di notevole interesse e con le possibilita' applicative che tutto quanto abbiamo visto consente, non deve pero' necessariamente essere quella di un' immagine statica. Per darle un' animazione, cioe' per muoverla sulla superficie dello schermo, il CBM64 usa 3 registri detti di posizione:

- 1) Registro di posizione X
- 2) Registro di posizione Y
- 3) Bit piu' significativo del registro di posizione X.

Ad ogni Sprite e' possibile associare ( o collegare) uno di questi registri e questo ci permette quindi di

posizionare queste figure in modo molto accurato. Si hanno infatti a disposizione ben 512 possibili posizioni per X e 256 possibili posizioni per Y. I registri di posizione inoltre lavorano sempre in coppia. Le locazioni dei registri X e Y appaiono in memoria come segue:

Prima il registro X per lo sprite 0, poi il registro Y per lo stesso Sprite. Successivamente il registro X per lo Sprite 1 e poi il registro Y per lo sprite 1 e così via. Al termine dei 16 registri (8 sprite per 2 registri ciascuno) viene il MSB (= Most Significant Bit) indirizzabile nel suo stesso registro.

La seguente tabella mostra l' indirizzo di memoria per ogni coppia di registri di ogni Sprite sui quali sarà necessario lavorare con gli appositi POKE.

| INDIRIZZI    | DESCRIZIONE           |
|--------------|-----------------------|
| 53248 \$D000 | R.P. X DELLO SPRITE 0 |
| 53249 \$D001 | R.P. Y DELLO SPRITE 0 |
| 53250 \$D002 | R.P. X DELLO SPRITE 1 |
| 53251 \$D003 | R.P. Y DELLO SPRITE 1 |
| 53252 \$D004 | R.P. X DELLO SPRITE 2 |
| 53253 \$D005 | R.P. Y DELLO SPRITE 2 |
| 53254 \$D006 | R.P. X DELLO SPRITE 3 |
| 53255 \$D007 | R.P. Y DELLO SPRITE 3 |
| 53256 \$D008 | R.P. X DELLO SPRITE 4 |
| 53257 \$D009 | R.P. Y DELLO SPRITE 4 |
| 53258 \$D00A | R.P. X DELLO SPRITE 5 |
| 53259 \$D00B | R.P. Y DELLO SPRITE 5 |
| 53260 \$D00C | R.P. X DELLO SPRITE 6 |
| 53261 \$D00D | R.P. Y DELLO SPRITE 6 |
| 53262 \$D00E | R.P. X DELLO SPRITE 7 |
| 53263 \$D00F | R.P. Y DELLO SPRITE 7 |
| 53264 \$D010 | MSB X                 |

Dove R.P. e' il registro di posizione.

La posizione di uno Sprite e' calcolata dalla posizione piu' in alto a sinistra (TOP LEFT) dell' area di 21x24 in cui puo' essere disegnato lo Sprite stesso.

Ricordiamo inoltre che non ha importanza il numero di punti presenti in una figura per il calcolo della posizione che deve essere sempre tenuta presente per evitare che nulla compaia sullo schermo.

## POSIZIONAMENTO VERTICALE

Sull' asse Y o in posizione verticale, ci sono 200 diversi punti che possono essere individualmente programmati sullo schermo.

Il registro di posizione Y di uno Sprite puo' manipolare fino a 255 valori. Cio' consente di avere a disposizione piu' posizioni di movimento in alto ed in basso di quanti ne sarebbero necessari.

Desiderando pero' muovere con semplicita' lo Sprite all' interno e fuori dello schermo richiede pero' piu' di 200 posizioni.

Per questo il primo valore che puo' apparire sullo schermo, dal suo massimo (TOP), nella direzione Y per uno sprite non espanso e' 30. Cioe' al disotto di questo valore non e' che lo sprite non esista, ma semplicemente non appare.

Per uno Sprite in dimensione espansa, poiche' e' di grandezza doppia e quindi il calcolo va rifatto per questi nuovi valori, il numero di cui si parlava prima e' 9.

Il primo valore Y con il quale uno Sprite ( espanso o non espanso) e' pienamente visualizzato ( cioe' con tutte le sue POSSIBILI linee visibili sullo schermo) e' 50.

L' ultimo valore Y in cui uno Sprite non espanso e' visualizzato in pieno sullo schermo e' 229, mentre per lo sprite espanso questo valore e' 208.



Quindi a partire da valori maggiori di quelli ricordati (229 e 208) la figura tendera' a scomparire dallo schermo.

Il primo valore Y per cui la figura e' TOTALMENTE fuori schermo e' 250.

Vediamo un esempio:

```
9 REM ESEGUI LA PULIZIA DI SCHERMO
10 PRINT "shift+ clear/home"
20 POKE 2040,13
30 FOR I=0 TO 62 : POKE 832+I,129 : NEXT
40 V=53248
50 POKE V+21,1
60 POKE V+39,1
70 POKE V+1,100
80 POKE V+16,0 : POKE V,100
```

Provate a farlo girare e a cambiarne i valori.

## CAPITOLO UNDICESIMO

### IL POSIZIONAMENTO ORIZZONTALE

Il posizionamento in direzione orizzontale e' molto piu' complesso perche' ci sono piu' di 256 posizioni. Ricordiamoci infatti che NON stiamo operando ne' come video ne' come figure su dei quadrati, ma sempre su rettangoli.

Mentre infatti per l' asse Y i 256 valori avanzavano ora no sono piu' sufficienti.

Si rendera' pertanto indispensabile usare un NONO BIT per controllare la posizione dello Sprite sull' asse X o orizzontale.

Infatti, aggiungendo questo bit extra avremo, quando necessario, disponibili 512 posizioni e quindi anche in questo caso piu' combinazioni possibili di quante ne possano essere visualizzate.

Con il NONO bit o MBS ogni Sprite puo' avere una posizione da 0 a 511, tuttavia solo i valori compresi fra 24 e 343 sono visibili sullo schermo.

Se la posizione X dello Sprite e' piu' grande di 255, per quanto detto, dovra' essere attivato il NONO bit mettendo a 1 il bit dell' apposito registro di indirizzo 53264.

Se invece e' inferiore a 255 lo stesso bit dello stesso registro (53264- \$D010) dovra' essere messo a 0.

Il seguente programma muove una figura :

```
10 PRINT"shift + clear/home"
20 POKE2040,13
30 FORI =0TO 62:POKE832 +I,129: NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
```

```
70 POKEV+1,100
80 FORJ =OT0347
90 HX=INT(J/256):LX=J-256*HX
100POKEV, LX:POKEV+16,HX: NEXT
```

Ricordando che precedentemente abbiamo detto che e' disponibile anche uno schermo ridotto, diamo un sommario delle posizioni visibili degli Sprites espansi e non.

#### SPRITE NON ESPANSO

Nel modo normale per essere una figura almeno parzialmente visibile:

X deve essere maggiore di 1 e minore di 343.

Y deve essere maggiore di 30 e minore di 249.

Con schermo ridotto:

X deve essere maggiore di 8 e minore di 334.

Y deve essere maggiore di 34 e minore di 245.

#### SPRITE ESPANSO

Nel modo normale per essere una figura almeno parzialmente visibile:

X deve essere compreso fra 489 e 343

Y deve essere compreso fra 9 e 249

Con schermo ridotto:

X deve essere compreso fra 496 e 334

Y deve essere compreso fra 13 e 245

## NOTA

Ricordiamo che nel modo a schermo ridotto i parametri sono indipendenti. Cioe' puo' essere usata la riduzione sull' asse X senza quella sull' Y , viceversa o insieme.

## PRIORITA' DI VISUALIZZAZIONE

Durante il loro movimento, cioe' durante gli spostamenti sullo schermo, gli Sprites possono prendere una strada qualsiasi e quindi incrociare, di fronte o di spalle altre figure presenti sullo schermo. Cio' consente una reale rappresentazione tridimensionale utilissima, ad esempio, per i giochi.

Per mettere un certo ordine e per ricavarne soprattutto dei risultati e' necessario disporre di un certo numero di informazioni, la prima delle quali sara' la priorita' o precedenza.

La priorita' fra gli Sprites e' fissata dal numero d'ordine delle stesse figure.

Cosi' lo Sprite 0 ha una priorita' piu' alta rispetto allo Sprite 1. Lo Sprite 1 avra' una priorita' piu' alta rispetto al 2 e cosi' via fino ad arrivare allo Sprite 7 che ha la priorita' piu' bassa.

La precedenza consiste nell' incrociare per primo.

In altre parole se lo Sprite 1 e lo Sprite 6 vengono posizionati o manipolati per incrociare una qualsiasi altra figura presente sullo schermo, lo Sprite 1 sara' DAVANTI allo Sprite 6.

In questo modo, dovendo per esempio creare un effetto scenico di movimento, per le figure in primo piano sara' necessario assegnare un numero basso ( come Sprite) e per

le figure di sfondo un numero piu' alto.

La priorita' di uno Sprite rispetto allo sfondo e' controllabile per mezzo del registro di indirizzo 53275 (\$D01B) detto anche :

#### SPRITE BACK-GROUND PRIORITY REGISTER

Anche in questo caso ad ogni bit di questo registro e' associato uno sprite.

Se il relativo Bit e' a 0 allora quello sprite avra' una priorita' piu' alta rispetto allo sfondo, cioe' lo Sprite apparira' DAVANTI o di fronte, rispetto ai dati dello sfondo.

Se il bit e' a 1 allora lo Sprite apparira' DIETRO perche' avra' una priorita' piu' bassa.

#### AVVERTENZA

Si consiglia di effettuare un po' di esercizi sulla manovra delle figure. All' inizio si puo' limitarsi a manovrare in tutti i modi possibili due sprites sullo schermo dove, tramite comandi Basic di POKE si sia magari creato una terza figura fissa.

#### SEGNALAZIONE DI SCONTRI O COLLISION DETECT

##### Introduzione

Un' altra capacita', utilissima durante la creazione di movimenti, e' la segnalazione degli scontri (cioe' di quando due figure si incontrano) o COLLISION DETECT.

Anche in questo caso lo scontro o incontro puo' essere segnalato quando avviene fra due Sprites o fra uno Sprite e la figura di sfondo.

La collisione avviene quando una parte dello Sprite che

non sia 0, cioe' quella parte che abbia dei punti attivati, va a finire in contatto o sopra una stessa parte di un' altro Sprite o di una figura o carattere dello sfondo.

## Scontro fra Sprites

L' incontro o scontro fra due Sprites e' riconosciuto dal computer per mezzo di un registro di indirizzo 53278 (\$D01E) conosciuto come:

## SPRITE TO SPRITE COLLISION REGISTER

Il cui funzionamento e' molto semplice.

All' interno di questo registro, con i principi e le tecniche di disposizione gia' viste per gli altri registri esaminati fino ad ora, ogni Sprite ha il suo Bit.

Normalmente questo bit e' a 0 (OFF). Quando avviene lo scontro il Bit va ad 1 e rimane ON fino a quando non venga letto ( di solito con un PEEK) nel qual caso viene resettato cioe' rimesso a 0.

## NOTA

Si ha sempre una segnalazione di scontro, cioe' il relativo valore del Bit va a 1, quando lo Sprite per effetto del movimento va fuori dallo schermo.

## Scontro fra Sprite e dati di sfondo

Esattamente come detto prima e con le stesse funzioni, esiste un registro di indirizzo 53279 (\$D01F) per

segnalare gli scontri fra uno Sprite ed un' altro tipo di figura (potrebbe essere anche un carattere).  
Questo registro, che ripetiamo, si comporta come il precedente si chiama:

## SPRITE TO DATA COLLISION REGISTER

## ALTRE POSSIBILITA' GRAFICHE

### Lo Screen Blanking

Questa funzione consente di far apparire lo sfondo dello schermo dello stesso colore del bordo.

In pratica tutti i dati presenti non vengono piu' visualizzati. Non e' che vengano cancellati, semplicemente non si vedono piu'.

Questa funzione e' controllata dal registro di indirizzo 53265 (\$D011) il cui bit 4 quando viene messo a 1 attiva la funzione di Blanking e quando viene messo a 0 la disattiva.

I comandi necessari per questa operazione sono i seguenti:

### ATTIVAZIONE BLANKING

POKE53265,PEEK(53265) AND 239

### DISATTIVAZIONE

POKE53265,PEEK(53265) OR 16

### NOTA

E' stato riscontrato che attivando questa funzione i programmi girano piu' rapidamente, anche se non di molto.

## RASTER REGISTER

All' indirizzo 53266 (\$D012) si trova un registro con una doppia funzione:

### IL RASTER REGISTER

Quando si legge questo registro vengono riportati gli 8 bits piu' bassi dell' attuale posizione raster. La posizione raster del bit piu' significativo e' di indirizzo 53265 (\$D011).

Si puo' utilizzare il registro raster per temporizzare il video o meglio i cambiamenti video.

I cambiamenti dovrebbero essere fatti quando il raster non e' nell' area di visualizzazione visibile, cioe' quando e' fuori video.

### INTERRUPT STATUS REGISTER

Questo registro mostra lo stato corrente di ogni sorgente di interruzione.

Il bit 2 di questo registro sara' a 1 quando avviene lo scontro fra due Sprites.

Lo stesso, con una corrispondenza di relazione 1 a 1, per i bits 0-3 riportati di seguito.

Anche il bit 7 viene messo ad 1 ogni volta che c'e' un INTERRUPT.

L' interrupt status register e' di indirizzo 53273 (\$D019) ed ha la seguente distribuzione interna:

### CHIAVE BIT DESCRIZIONE

|      |   |  |
|------|---|--|
| IRST | 0 | Viene fissato quando il CURRENT RASTER = STORED RASTER |
|------|---|--|



|      |   |   |
|------|---|---|
| IMDC | 1 | Fissato dallo SPRITE DATA Collision (solo la I volta) |
| IMMC | 2 | Come sopra ma da SPRITE-SPRITE collision              |
| ILP  | 3 | Fissato dalla Light Pen                               |
| IRQ  | 7 | Fissato dalla chiusura                                |

Quando viene fissato (o settato) un bit di interrupt, questi viene "CHIUSO" e per essere di nuovo a 0 (CLEARED) si deve scrivere un 1 nello stesso registro quando sara' di nuovo necessario manipolarlo.

Cio' consente una manipolazione selettiva di interrupt senza dover immagazzinare gli altri bit di interrupt.

L' INTERRUPT ENABLE REGISTER e' un registro di indirizzo 53274 (\$D01A).

Ha lo stesso formato del registro precedente.

Tuttavia, a meno che il corrispondente bit del registro precedente sia messo a 1, nessun di queglii interrupt puo' avere luogo.

L' interrupt status register puo' essere visto come fonte d' informazione, ma non sara' consentita nessuna interruzione.

Per consentire una richiesta di interrupt, il corrispondente bit dell' Interrupt enable Register deve essere messo a 1.

Questa potente possibilita' operativa consente grandi risultati per quanto riguarda la manipolazione grafica. Si potra' avere mezzo schermo in BIT MAP MODE, e mezzo in modo testo, piu' di otto Sprites allo stesso tempo.

Riteniamo tuttavia che per l' uso di queste sofisticate tecniche, per quanto utilizzabili in Basic sia molto meglio passare attraverso i codici macchina e i programmi in Assembler.

## CAPITOLO DODICESIMO

### ALTRO MODO DI TRATTARE GLI SPRITES

Questa sezione del capitolo e' stata messa a punto per due scopi. Il primo e' di dare informazioni sugli Sprites da un' altro punto di vista ed il secondo scopo e' di avvicinarsi alla programmazione di queste figure grafiche in modo piu' elementare.

#### Un breve programma

Ci sono come abbiamo gia' visto e come vedremo in seguito, anche su altri testi e capitoli, molte strade per creare immagini in movimento sul CBM64.

Comunque il metodo migliore, e da qualsiasi punto di vista lo si voglia osservare, restano gli Sprites, data la grande mole di lavoro che per il loro utilizzo svolge direttamente l' integrato VIC-II.

Per dimostrare la facilita' dell' uso degli Sprites scriveremo e commenteremo un piccolo programma di "MANIPOLAZIONE SPRITES " in Basic.

```
10 PRINT"shift+clr/home"  
20 POKE 2040,13  
30 FORS=832 TO 832+62:POKES,255:NEXT  
40 V=53248  
50 POKEV+21,1  
60 POKEV+39,1  
70 POKEV,24  
80 POKEV+1,100
```

Questo programma racchiude in se,per quanto breve sia, tutti gli "INGREDIENTI".

Vi consigliamo di avere sottomano la:

## CARTA DI MANIPOLAZIONE DEGLI SPRITES

Il nostro programma definisce il primo Sprite ( lo Sprite 0) come un quadrato bianco. Vediamo ora di commentarlo linea per linea.

LINEA 10 Esegue il Clear di schermo

LINEA 20 Fissa il Puntatore degli Sprites ( o SPRITES POINTER o SP ) con il quale il CBM64 leggerà i valori costituenti gli Sprites.

Lo SP è fissato a 2040 per lo Sprite 0, per l' 1 a 2041, per il 2 a 2042 e così via fino al 7 di indirizzo 2047. Si possono fissare tutti e 8 gli Sprites in modo veloce usando un ciclo. Sostituendo cioè la linea 20 con:

```
20 FOR SP=2040 TO 2047:POKESP,13:NEXT SP
```

LINEA 30 Immette i valori del primo Sprite (cioè dell' SP 0) entro 63 Bytes della memoria RAM del CBM64, che in questo caso inizia dalla locazione 832.

Poiché ogni Sprite necessita di 63 Bytes di memoria in questo caso l' indirizzo dello Sprite 0 sarà da 832 a 894.

### NOTA

Anche in questo caso abbiamo utilizzato la memoria del Buffer di Cassetta. Ricordarsi di non utilizzarlo contemporaneamente a programmi in codice macchina che in specie se non molto lunghi si ha l' abitudine di immagazzinare in questa parte di RAM.

LINEA 40 Assegna alla variabile V il valore 53248 che corrisponde all' indirizzo di inizio dell' Integrato Video.

Questo metodo viene utilizzato per facilitare il lavoro, in particolare mnemonico, e per accorciare (CRUNCH) i programmi.

La memoria del CBM64 conservera' in un' apposita locazione il valore di V ( 1 solo Byte) e vi permettera' di operare con numeri piu' piccoli (come vedremo) e quindi piu' facili da manipolare e da ricordare.

Per esempio nella linea 50 di questo programma abbiamo scritto POKE V+21 che equivale a :

POKE 53248 + 21 cioe'

POKE 53269

ma V+21 richiede uno spazio minore ed e' piu' facile da ricordare.

LINEA 50 Abilita o mette in grado di funzionare (TURN ON) lo sprite 0. Come sappiamo ci sono 8 Sprites possibili, numerati da 0 a 7.

Per attivare uno Sprite o una combinazione di Sprites, si deve eseguire un comando POKEV+21,X dove X e' un numero variabile da 0 a 255 che consentira' di attivare uno o piu' Sprites contemporaneamente.

Vediamo la seguente mappa di attivazione ripresa dalla piu' completa MAPPA gia' citata:

POKE V+21,255 Attiva tutti gli Sprites

POKE V+21,1 Attiva lo Sprite 0

POKE V+21,2 Attiva lo Sprite 1

POKE V+21,4 Attiva lo Sprite 2

POKE V+21,8 Attiva lo Sprite 3  
POKE V+21,16 Attiva lo Sprite 4  
POKE V+21,32 Attiva lo Sprite 5  
POKE V+21,64 Attiva lo Sprite 6  
POKE V+21,128 Attiva lo Sprite 7  
POKE V+21,0 Disattiva tutti gli S.

E' importante notare che con uno stesso comando di POKE si possono attivare contemporaneamente piu' sprite. Basta infatti mettere nel secondo operando del comando POKE la somma dei valori riportati in tabella degli Sprite che si vogliono attivare.  
Così per attivare S. 2 e S. 7:

POKEV+21, 130 (cioe' 128+2)

e per S. 0, S. 3 e S. 5

POKEV+21, 41 (cioe' 1+8+32)

LINEA 60 Fissa il colore dello Sprite 0.

Ci sono 16 possibili colori per gli Sprites, numerati da 0 a 15.

Ogni Sprite richiede un comando POKE diverso per fissare il colore, cioe' da POKEV+39, X (per lo Sprite 0 e dove X e' il colore scelto) a POKEV+46, X ( per lo Sprite 7). Vediamo alcuni esempi:

POKEV+39,1 assegna allo Sprite 0 il colore, per tutti i punti visualizzati, BIANCO.

POKEV+46,15 assegna allo Sprite 7 il colore GRIGIO.

Quando si crea uno Sprite, come abbiamo appena fatto, questi resta in memoria fino a quando non lo si metta in OFF, oppure non lo si ridefinisca o non si spenga il computer.

Tutto questo dà la possibilità di cambiare colore, posizione e contenuto dello stesso Sprite in modo diretto o immediato che è utile per le funzioni di editing e le eventuali prove connesse.

Facciamo, per esempio girare il programma e dopo digitiamo, in modo DIRETTO cioè senza riga di programma:

POKEV+39,8

vedremo che lo Sprite assume la colorazione ORANGE.

Allo stesso modo possiamo cambiare in un colore qualsiasi.

Eseguendo il RUN del programma dall'inizio, lo sprite ritornerà al BIANCO perché verrà ripristinato il comando della linea 60 che lo fissa a 1.

LINEA 70 Determina la posizione orizzontale o sull'asse X dello Sprite .

LINEA 80 Determina la posizione verticale o sull'asse Y dello Sprite.

In questo programma abbiamo dato come posizione orizzontale (vedi linea precedente) a 24 e quella verticale a 100.

Per visualizzare su altre posizioni lo Sprite proviamo i seguenti comandi diretti:

POKEV,24:POKEV+1,50

che visualizzerà la figura nell'angolo in alto a sinistra dello schermo.

POKEV,24:POKEV+1,229

che visualizzera' la figura nell' angolo basso a sinistra.

Ogni numero da 832 a 895 nel nostro Sprite 0 rappresenta un blocco di 8 PIXELS (punti grafici) con 3 blocchi di 8 PIXELS quindi in ogni riga orizzontale dello Sprite.

Il ciclo della linea 80 comunica al computer il comando di POKE832,255 che riempie completamente il primo dei tre blocchi di 8 PIXEL. Poi il comando POKE833,255 eseguirà la stessa funzione sul secondo blocco della prima riga e così via fino all'ultima locazione a 849.

(Per lo Sprite queta ultima locazione si tratta del terzo blocco di 8 PIXEL dell' ultima riga).

In questo modo ecco perche' ci troveremo come risultato del disegno un rettangolo pieno di 24x21 punti, cioè 3blocchi da 8 PIXEL ripetuti per 21 righe.

Per capire meglio quanto abbiamo appena detto proviamo a cancellare in modo diretto i punti che compongono il secondo blocco di PIXEL della prima riga con:

POKE833,0

Questa operazione puo' essere fatta in modo diretto. Per tornare alla figura originaria sara' sufficiente eseguire:

POKE833,255

o

Eeguire il RUN del programma dall' inizio.

La seguente linea di comandi, che potrete anche aggiungere nell' interno del programma, cancellerà i BLOCCHI di PIXELS e quindi i punti relativi del centro della figura Sprite:

```
90 FORA=836 TO891 STEP3:POKEA,0:NEXTA
```

## NOTA

E' possibile eseguire un CRUNCH cioe' un accorciamento o compattazione dei programmi.

Le tecniche sono numerose per risparmiare memoria e sono state anche condensate in un programma disponibile chiamato:

## COMPACTOR

Senza tuttavia scendere in particolari diamo un esempio di come il precedente programma, scritto e commentato, pur abbastanza corto possa essere ulteriormente ridotto addirittura in due sole linee:

```
10PRINTCHR$(147):V=53248 : POKEV+21,1 :POKE2040,13  
:POKE4+39,1  
20FORS=832TO894: POKES,255: NEXT:POKEV,24:POKEV+1,100
```

## POSIZIONAMENTO DEGLI SPRITES SU SCHERMO

Ancora qualcosa sugli Sprites.

L' intero schermo puo' essere considerato come diviso in una griglia di coordinate X (orizzontale) e Y (verticale).

Si tratta quindi di dare al computer le coordinate , appunto X e Y che gli consentano di posizionare le figure



sullo schermo.

Ricordiamo ancora una volta che uno Sprite consiste di 504 punti disposti su una matrice 24x21.

Per visualizzare uno Sprite in una determinata locazione di schermo sara' quindi necessario comunicare al computer con i relativi ( uno per ogni Sprite) comandi POKE sia per l' asse X che per l' asse Y.

POKE sulla posizione X

I possibili valori di X sono da 0 a 255 contando da sinistra a destra.

I valori da 0 a 23 immettono in tutto o in parte lo Sprite fuori dall' area visibile dello schermo, mentre i valori da 24 a 255 lo immettono nell' area visibile ( per i valori oltre 255 vedi dopo).

Quindi per immettere uno sprite in una di queste posizioni, sempre rispetto alla' asse orizzontale, e' sufficiente:

POKE V+K,X

dove:

K = e' lo Sprite sul quale si opera

X = la posizione

Esempio:

POKEV+2,24

Immetterà lo Sprite 1 nella I colonna di schermo.( la piu' a sinistra)

NOTA

Colonna in questo caso ha il significato di zona. Inoltre noterete che non abbiamo detto in alto a sinistra perche' manca di fissare la posizione Y.

## Valori di X oltre la 255ma posizione

Per andare oltre la 255ma posizione orizzontale di schermo e' necessario eseguire un secondo comando di POKE usando i valori della riga "RIGHT X" della tabella mostrata.

Di norma infatti la numerazione orizzontale di X, dovrebbe continuare passata la 255ma posizione di X passando quindi alla 256ma, 257ma e cosi' via fino al termine dello schermo visibile.

Tuttavi poiche' i registri usati sono di 8 bit cio' vuol dire che non possono contenere un valore maggiore di 255. Sara' quindi necessario usare un secondo registro per accedere a quella parte destra dello schermo altrimenti indisponibile.

Questo risultato si ottiene settando appunto il RIGHT X con il comando :

POKE V+16, S

Dove S e' il corrispondente Sprite.

Questa tecnica ci consnte altre 65 posizioni visibili che nella figura precedente vanno dalla linea tratteggiata in verticale fino alla fine dello schermo.

Naturalmente si puo' usare un numero nmaggiore di 65 ma lo Sprite tendera' a scomparire dallo schermo.

POKE di Y.

I possibili valori di Y vanno da 0 a 255 partendo dalla parte alta dello schermo.

Tuttavia gli Sprites sono visibili completamente solo per i valori da 50 a 249, mentre per i valori compresi fra 0 e 49 gli Sprites saranno presenti ma invisibili oltre la parte alta dello schermo e fra 230 e 255 invisibili oltre la parte bassa.

POSIZIONAMENTO DI PIU' ANIMAZIONI

Il seguente programma definisce 3 differenti Sprites ( lo 0, 1 e 2) di colore diverso posizionandoli in tre diversi punti dello schermo.

```
10 PRINT"shift+clr/home"  
15 V=53248:FORS=832 TO 895:POKES,255:NEXT  
20 FORM=2040TO2042:POKEM,13:NEXT  
30 POKE+21,7  
40 POKEV+39,1:POKEV+40,7:POKEV+41,8  
50 POKEV,24:POKEV+1,50  
60 POKEV+2,12:POKEV+3,229  
70 POKEV+4,255:POKEV+5,50
```

Per comodita' i tre Sprites sono stati definiti come quadrati pieni che traggono i dati dalla stessa fonte. Infatti l' aspetto che qui interessa e' il loro posizionamento sullo schermo.

Provate a digitare questo programma e vedrete cosa accade.

A conclusione di questi capitoli sulla grafica vogliamo segnalare la presenza sul mercato di una serie di pacchetti pronti che l' appassionato potra' utilizzare con grande soddisfazione.

Sta per uscire inoltre un MANUALE DELLA GRAFICA PER CBM64, a cura della EVM che conterra' non solo un approfondito discorso sulla materia e numerosi esempi applicativi ma anche una serie di programmi su cassetta per impadronirsi rapidamente e approfonditamente di questo argomento.

## CAPITOLO TREDICESIMO

### LA PROGRAMMAZIONE DEI SUONI

#### Introduzione

Il vostro CBM64 e' dotato di uno dei piu' sofisticati sintetizzatori musicali oggi disponibile su un computer di queste dimensioni.

Questo sintetizzatore consente l' utilizzo delle seguenti possibilita':

3 VOCI

INDIRIZZABILITA' TOTALE

ATTACK

DECAY

SUSTAIN

RELEASE

FILTRI

MODULAZIONI

RUMORE BIANCO

#### NOTA

Malgrado i termini ATTACK, DECAY, SUSTAIN, RELEASE, siano traducibili, anche se male, con ATTACCO, DECADIMENTO, SOSTEGNO e RILASCIO noi utilizzeremo la scrittura inglese

ed anche la forma abbreviata ADSR per indicarli tutti e quattro insieme.

Tutte queste funzioni sono direttamente disponibili attraverso comandi e programmi sia in Basic che in codice macchina.

Cio' consente che si possano generare forme sonore e musicali, anche complesse, con relativa facilità.

Questa sezione del manuale e' stata scritta per aiutarvi ad esplorare tutte le capacità dell' integrato 6581 "SID" il sintetizzatore musicale e sonoro che si trova all' interno del CBM64.

Per comprendere bene quello che segue non e' necessario conoscere la musica, infatti ogni parte di questo capitolo e' scritto iniziando da un esempio e poi proseguendo con le spiegazioni.

Vedrete che, quasi gli unici comandi usati in questo capitolo, sono i POKE ed i DATA.

La forma generale del POKE, lo ricordiamo e' :

POKE MEM, NUM

Dove :

MEM = alla locazione di memoria nella quale si vuole immettere un determinato NUM = numero o valore.

Le locazioni di memoria utilizzate per il sintetizzatore iniziano dall' indirizzo 54272 (\$D400) e vanno fino a 54296, mentre ovviamente il NUM potrà avere un valore da 0 a 255.

Allo stesso modo che abbiamo visto per la parte riferentesi agli Sprite anche in questo caso sarà conveniente sia dal punto di vista mnemonico sia per quanto riguarda l' occupazione di memoria assegnare una costante, ad esempio S alla locazione 54272 ed usare per i comandi POKE degli incrementi a questo valore.

Senza perderci in altri discorsi, resettiamo il nostro computer e scriviamo il seguente programma che, dopo averlo salvato su cassetta, commenteremo:

## PROGRAMMA 1

```
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<0THEN END
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125,
140 DATA 32,94,75,0,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1
```

Ed ecco ora una descrizione, passo passo, del programma appena scritto.

Riferendosi a questa descrizione potrete accuratamente ricontrollare il vostro programma.

```
5 Fissa la costante di inizio
10 Esegue un clear su tutti i registri musicali
20 Fissa l' Attack/Decay per la voce 1 (A=0, D=9)
Fissa il Sustain/Release per la voce 1 (S=0, R=0)
```

30 Fissa il volume al massimo  
40 Legge l' alta frequenza, la bassa frequenza e la durata delle note  
50 Si ferma quando trova per l' alta frequenza un valore minore di 0  
60 Immette l' alta e la bassa frequenza per la voce l  
70 Fissa la forma d' onda SAWTOOTH per la voce l  
80 Ciclo di temporizzazione per la durata delle note  
90 Rilascia SAWTOOTH per la voce l  
100 Ritorno per la successiva nota  
110-180 DATA per il suono: alta frequenza, bassa frequenza, durata per ogni nota  
190 Ultima nota e serie di -1 che segnalano la fine del programma da eseguire.

## CONTROLLO VOLUME

Il registro 54296 (cioe' S+24) e' diviso in due NYBBLE il primo del quale viene utilizzato per controllare il volume da un minimo di 0 ad un massimo di 15, mentre gli altri 4 bits sono utilizzati per funzioni che vedremo in seguito.

## FREQUENZE D' ONDA SONORA

Il suono e' creato dal movimento dell' aria in onde, infatti dove non c' un mezzo di trasmissione, l' aria (ma potrebbe essere anche l' acqua), non esiste la possibilita' di emettere e soprattutto di ascoltare suoni. Se misuriamo il tempo che intercorre tra il picco o punta massima dell' onda ed il successivo troveremo il numero di secondi per un ciclo d' onda.

Ammettiamo di chiamare n questo numero avremo allora che il reciproco di questo numero, cioe' uno su n ( $1/n$ ) dara' i cicli per secondo, che sono molto piu' comunemente conosciuti come:

## FREQUENZA

Il generatore sonoro del CBM64 usa due locazioni per determinare la frequenza.

In appendice sono riportati i valori necessari a riprodurre una gamma completa di otto ottave.

## USO DI VOCI MULTIPLE

Il vostro CBM64 ha tre controlli di voci indipendenti cioe' 3 Oscillatori.

Nel nostro primo esempio riportavamo SOLO l' uso di una voce. Piu' tardi impareremo come puo' essere cambiata la qualita' del suono stesso attraverso le voci, ma per il momento vediamo di farle suonare tutte e tre insieme.

L' esempio seguente mostra come far suonare le tre voci cioe' in pratica come far suonare il CBM 64 come un' orchestra.

Non dimenticate di salvare il vostro programma su nastro o su disco e ricordatevi sempre di aver resettato, con un NEW e un comando CLEAR, il vostro computer.

## 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 OTHENNMM=-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=0TOOC%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 IM=1
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(1,I):POKES+14,L(2,I)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,I):POKES+11,C(1,I):POKES+18,C(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38593,40830
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 DATA326,327,329,327,329,326,578,583
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 DATA1577,299,295,306,310,311,304  
 3100 DATA562,546,1575  
 3999 DATA0

I valori utilizzati nei comando DATA del precedente programma sono stati ricavati per mezzo dell' apposita tavola di conversione che si trova al termine del volume e dalla tabella seguente:

| TIPO NOTA | DURATA |
|-----------|--------|
| 1/16      | 128    |
| 1/8       | 256    |
| 1/8 PUNT. | 384    |
| 1/4       | 512    |
| 1/4+1/16  | 640    |
| 1/4 PUNT. | 768    |
| 1/2       | 1024   |
| 1/2+1/16  | 1152   |
| 1/2+1/8   | 1280   |
| 1/2 PUNT. | 1536   |
| INTERO    | 2048   |

Il numero della nota e' aggiunto alla durata appena descritta.

Per questo ogni nota puo' essere inserita usando un solo numero che e' codificato dal programma.

Naturalmente questo e' solo un metodo per codificare il valore delle note per cui ne potrete usare altri che possano risultarvi piu' convenienti o piu' comodi e facili.

La formula qui utilizzata per codificare le note e' la seguente:

1) La durata ( numero di sedicesimi della misura) e' moltiplicato per 8

2) Il risultato del passo 1 e' aggiunto all' ottava che e' stata scelta

3) Il risultato del passo 2 e' quindi moltiplicato per 16

4) Aggiungere la nota scelta (0-11) al risultato dell' operazione nel passo 3.

In altre parole:

$$((((D*8)+O)*16)+N)$$

Dove D= durata

O= ottava

N= nota

Il silenzio e' ottenuto usando il negativo del numero di durata.

CONTROLLO DI VOCI MULTIPLE

Ora che avete imparato ad usare contemporaneamente piu' di una voce troverete che le tre voci debbano essere coordinate come esecuzione temporale.

Questo viene ottenuto con:

- 1) Dividere ogni misura musicale in 16 parti
- 2) Immagazzinare cio' che accade in ogni sedicesimo in 3 matrici separate.

I Bytes di frequenza alta e bassa sono calcolati dividendo le frequenze dell' ottava piu' alta per 2 ( linee 180 e 190 del precedente programma).

Il Byte di controllo della forma d' onda e' un segnale di partenza per iniziare una nota o per continuare una nota che sta gia' suonando. Esiste anche un segnale di Stop per finire una determinata nota.

Sempre nel precedente programma la scelta della nota e' fatta una volta per tutte e per ogni singola voce alla linea 40.

Ripetiamo ancora che questo e' solo uno dei tanti sistemi che possono essere utilizzati per controllare voci multiple. Diciamo che e' un metodo qualsiasi e che potete svilupparne uno altrettanto valido se non di piu' e che comunque soddisfi le esigenze di programmazione sonora.

## CAMBIO DELLE FORME D' ONDA

La qualita' tonale di un suono e' chiamata TIMBRO ed il timbro di un suono e' determinato prima di tutto dalla sua forma d' onda.

Senza addentrarci in particolari spiegazioni su cosa sia il suono e sul come venga riprodotto partiamo subito con un esempio.

Provate a ricaricare da nastro o da disco il programma che abbiamo visto all' inizio di questo capitolo.

Diciamo subito che il suono prodotto da questo programma, rispetto alla forma d' onda e' WAVEFORM SAWTOOTH o forma d' onda a dente di sega.

Provate a cambiare le note d' inizio alla riga 70 da 33 a 17 e le note di stop nella riga 90 da 32 a 16. Il vostro nuovo programma dovrebbe essere il seguente:

```
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<0THEN END
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125
140 DATA 32,94,75,0,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1
```

Eseguite il RUN del programma e noterete come la qualita' del suono sia differente.

Cio' perche' siamo passati da una forma d' onda a dente di sega ad una forma d' onda triangolare che' e' un suono meno secco e piu' allungato.

La terza forma d' onda musicale e' chiamata onda pulsante (PULSE WAVE).

Questa e' un' onda rettangolare e si puo' determinare la lunghezza del ciclo di PULSE definendo la proporzione dell' onda che sara' alta.

Tutto questo viene determinato per la voce 1 usando i

registri 2 e 3.

Il registro 2 e' il Byte basso dell' ampiezza d' impulso (da 0 a 255), mentre il registro 3 sono i 4 bits alti (da 0 a 15).

Percio' questi registri specificano un numero di 12 bits per l' ampiezza d' impulso che potete determinare e poi fissare usando il seguente metodo:

$$PW=Hpw*256+Lpw$$

L' ampiezza d' impulso e' determinata dalla seguente equazione:

$$PWout=(PWn/40.95)\%$$

Quando  $PWn$  assume un valore di 2048 avrete una forma d' onda quadra. Cioe' il registro 2 ( $Lpw$ ) sia = a 0 e il registro 3 ( $Hpw$ ) =8.

Provate ora ad aggiungere questa riga al vostro programma:

$$15 \text{ POKES}+3,8:\text{POKES}+2,0$$

Cambiate poi il numero d' inizio o di START della riga 70 in 65 ed il numero di fine della linea 90 in 64 e fate girare il programma. Noterete la notevole differenza di suono.

Questo sistema e' molto utilizzato piu' che altro per EFFETTI sonori.

## IL GENERATORE DI ENVELOPE

Il volume di un tono musicale cambia, cioè varia dal momento in cui inizia lungo la sua durata fino alla fine dell' emissione stessa.

In altre parole quando viene generata una nota essa passa dal suo volume zero al suo picco o massimo ed il passaggio in cui ciò avviene è chiamato :

#### ATTACK

Il tempo in cui la nota va a finire, cioè si riporta al suo volume ZERO è chiamato:

#### DECAY

Il volume stesso, nel suo intervallo di mezzo è chiamato:

#### SUSTAIN LEVEL

E, per finire, quando la nota ha terminato di produrre suoni essa passa dal livello SUSTAIN al volume zero.

L' intervallo nella quale cade è chiamato:

#### RELEASE

Ognuno dei passi o intervalli di tempo menzionati contribuisce a caratterizzare la nota stessa.

Infatti i parametri visti cioè:

#### ATTACK/DECAY/SUSTAIN/RELEASE

detti anche in maniera abbreviata ADSR, possono essere controllati da un gruppo di registri dell' integrato generatore di suoni.

Passando ad un altro esempio, proviamo ancora una volta a

caricare il nostro programma numero 1 e a farlo girare cercando di ricordare i suoni emessi ed il loro tipo. Successivamente proviamo a cambiare la riga 20 in modo tale che il programma risulti essere in definitiva il seguente:

## PROGRAMMA

```
5  S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THEN END
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125
140 DATA 32,94,75,0,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1
```

I registri 5 e 6 definiscono l' ADSR per la voce 1.

L' ATTACK e' il NYBBLE alto del registro 5.

Ricordiamo che per nybble intendiamo mezzo Byte, in altre parole i quattro BITS alti o bassi del registro che di solito e' costituito da un Byte intero.

Il DECAY e' il nybble basso del registro 5.

Si puo' scegliere un numero qualsiasi compreso nell' intervallo fra 0 e 15 per il parametro ATTACK, moltiplicarlo per 16 ed aggiungerci un numero a scelta



evidenziati nella tabella seguente.

Il livello di SUSTAIN e' il nybble alto del registro 6 e puo' essere fra 0 e 15 e ricordiamo che definisce l'intervallo di tempo in cui DURA il picco, cioe' la parte piu' alta della nota.

L' intervallo di RELEASE e' il nybble basso del registro 6.

| VALORE | ATTACK | DECAY/RELEASE |
|--------|--------|---------------|
| 0      | 2 ms   | 6 ms          |
| 1      | 8      | 24            |
| 2      | 16     | 48            |
| 3      | 24     | 72            |
| 4      | 38     | 114           |
| 5      | 56     | 168           |
| 6      | 68     | 204           |
| 7      | 80     | 240           |
| 8      | 100    | 300           |
| 9      | 250    | 750           |
| 10     | 500    | 1.5 s         |
| 11     | 800    | 2.4           |
| 12     | 1 s    | 3             |
| 13     | 3      | 9             |
| 14     | 5      | 15            |
| 15     | 8      | 24            |

NOTA

Naturalmente per ms si intendono i millisecondi e per s i secondi di durata.

Vediamone qualche breve applicazione eseguendo dei cambiamenti al programma originale.

Per un suono tipo il violino:

20 POKES+5,88:POKES+6,89

Proviamo ora a cambiare la forma d' onda a triangolo e inseriamo un tipo di suono a XILOFONO usando le righe di programma seguente:

```
20 POKES+5,9:POKES+6,9
70 POKES+4,17
90 POKES+4,16:FORT=1T050:NEXT
```

Proviamo ora a immettere una forma d' onda quadra e proviamo ad eseguire il cambiamento per ottenere un tipo di suono PIANOFORTE:

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0
70 POKES+4,65
90 POKES+4,64:FORT=1T050:NEXT
```

## FILTRI

Il contenuto armonico di una forma d' onda puo' essere cambiato usando un filtro ed il SID e' dotato di ben 3 tipi di filtri che possono essere usati separatamente o in combinazione fra loro.

Torniamo ancora una volta al nostro semplice programma iniziale ed aggiungiamo la linea 15 per fissare la frequenza di taglio del filtro. Questa frequenza di taglio o CUTOFF FREQUENCY e' il punto di riferimento per il filtro.

Nel seguente programma si fissano i punti di frequenza di taglio nei registri 21 e 22. Per mettere ON, cioe' per attivare il filtro per la voce 1, eseguiamo un POKE sul registro 23.

Il cambiamento successivo alla riga 30 mostra l' uso di un filtro PASSA-ALTO.

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
```

```

15 POKES+22,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THEN END
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125
140 DATA 32,94,75,0,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1

```

Provate a far girare questa nuova versione del programma e notate i cmbiamenti di tono.

Non vogliamo approfondire concetti che ai piu' risulterebbero abbastanza oscuri e che saranno quindi approfonditi in altri e separati volumi.

Ricordiamo solo che i filtri disponibili sul nostro CBM 64 sono:

FILTRO PASSA-ALTO

FILTRO PASSA-BASSO

FILTRO PASSA-BANDA

Puo' essere usata la combinazione di un filtro Passa-alto con un filtro Passa-basso.

## CAPITOLO QUATTORDICESIMO

### Introduzione alle periferiche

E' normalmente molto difficile che un computer possa essere utilizzato come sola unita' centrale.

In questo caso infatti ci si dovrebbe limitare ad utilizzare un mezzo sicuramente meno potente, ma altrettanto sicuramente piu' facile da usare, come ad esempio una calcolatrice delle quali ne esistono versioni sempre piu' sofisticate e con grandi capacita'.

Esiste pur sempre, utilizzando un computer piccolo o grande che sia, il problema di immagazzinare programmi scritti dall' utente stesso oppure la necessita' di caricare programmi acquistati.

Percio' un computer diventa realmente un SISTEMA DI ELABORAZIONE DATI solo quando dispone come minimo di un' unita' di massa o di memoria esterna.

Abbiamo cercato di spiegare con parole facili una serie di concetti, quelli relativi ai files, che risultano da sempre lo scoglio maggiore ad un corretto apprendimento della programmazione, ma senza i quali a nostro modesto parere, l' utente resta solo un mezzo programmatore, non potendo utilizzare in pieno le reali capacita' di un computer di qualsiasi tipo si tratti.

### NOTA

Quanto segue e' stato tratto, integralmente o parzialmente dal volume LE PERIFERICHE COMMODORE ediz. EVM al quale si rimanda per approfondimenti e per tutta la parte riguardante sia il disco che la stampante e le porte di uscita con relativi collegamenti.

## I FILES

Un sistema di computer e' composto da qualcosa in piu' di una tastiera, di uno schermo e della stessa unita' centrale, cuore del computer.

Per avere un programma a disposizione tutte le volte che si vuole senza doverlo reinserire nella memoria della unita' centrale con una lunga e noiosa serie di operazioni durante la quale e' oltretutto facile sbagliarsi e' necessario disporre di una unita' di registrazione che potra' essere una unita' a cassetta o un floppy disk.

Prendiamo per ipotesi un programma per la gestione di indirizzi o MAIL PROGRAM.

Questo programma, per non riscriverlo tutte le volte, dovra' essere immagazzinato in una cassetta o in un dischetto.

Inoltre un MAIL PROGRAM e' usato per creare una lista di nomi e di indirizzi che per essere utilizzata deve essere immagazzinata anche questa.

Infine per un completo utilizzo sara' necessario disporre di una stampante per le lettere, gli elenchi.

Per questo motivo difficilmente potremo limitarci alla sola unita' centrale.

Volendo sintetizzare un computer ha essenzialmente tre grandi capacita':

CALCOLO

SCELTA

COMUNICAZIONE

Fino a questo momento, se e' stata letta e studiata

accuratamente la parte relativa al Basic ed alla programmazione, dovrebbero essere conosciute le prime due capacita', vediamo ora la terza.

La capacita' di comunicare con l' esterno e', in un computer, certamente la piu' complessa perche' e' necessario per prima cosa conoscere una serie di regole precise per l' invio e la ricezione dei dati, e poi adeguarsi a queste regole, cioe' metterle in pratica, renderle operative.

Le capacita' di un computer di comunicare con il mondo esterno (esterno al microprocessore naturalmente) sono veramente notevoli, tuttavia noi ci limiteremo per il momento a prendere in considerazione solo la possibilita' di leggere, scrivere e modificare dati e programmi su nastro che e' la periferica piu' usata per il suo costo.

## IL CONCETTO DI FILE

Sia su dischi che su cassette le informazioni sono immagazzinate come " FILES".

Per comprendere il concetto di FILE immaginiamo un comune schedario da tavolo entro il quale conserveremo, per esempio delle schede contenenti indirizzi di nostri amici, clienti o altro.

Avremo quindi che lo schedario sara' il contenitore del nostro FILE, cioe' la cassetta o il dischetto. Non l' unita' a cassette ma il nastro stesso.

L' insieme delle schede sara' il FILE vero e proprio e che quindi potra' ampliarsi in rapporto alla grandezza fisica del contenitore.

E' intuitivo che le singole schede saranno i RECORDS, mentre i dati della scheda saranno i campi o FIELDS.

Per un utente di computer il concetto di FILE deve essere considerato come di primaria importanza, ma non e' difficile da comprendere.

Quando si apre (OPEN) un file, tutte le informazioni ivi immagazzinate diventano accessibili e lo rimangono fino a

quando non si chiude (CLOSE).

Rifacendosi all' esempio iniziale e' come aprire il contenitore.

Quando un computer scrive un programma o dei dati su cassetta o disco, crea un nuovo file o aggiunge qualcosa ad uno vecchio.

Un file puo' avere una lunghezza qualsiasi, limitata sola dalla capacita' della cassetta o del disco.

Si puo' creare un nuovo file senza scrivere niente dentro, cio' equivale ad avere una serie di schede bianche senza alcun contenuto.

Si possono avere numerosi files per ogni dischetto (dipende da che tipo di unita' CBM.sì sta adoperando ), mentre non esiste limite per la cassetta.

L' ammontare della memoria non ha nessun effeto sulla grandezza di un file di dati.

Un file di dati puo' essere piu' grande della memoria disponibile del vostro computer. Aprendo un file di dati infatti si puo' leggere un solo carattere o piu' informazioni e passarle alla memoria centrale del computer.

Quando si scrive un file di dati, le informazioni che passano dalla memoria del computer a quella di massa possono essere, e di norma e' cosi',aggiunte a dati immagazzinati in precedenza sulla cassetta o sul floppy.

## FILES PROGRAMMI

Ci sono due differenti tipi di files:

### FILES PROGRAMMI

### FILES DATI

Un file programmi, come e' implicito nel nome, contiene un serie di comandi in Basic, in Assembler, in Pascal o qualsiasi linguaggio si stia usando, messi insieme come programma.

Si crea un file programmi utilizzando un comando SAVE. Cioe' quando si usa il comando SAVE automaticamente viene creato sulla periferica un file PROGRAMMA.

Per creare un file di dati e' invece indispensabile usare piu' di un comando il che ci riporta al concetto di programma piu' o meno piccolo.

## FILES PROGRAMMA

Un file puo' avere un nome, per cui il nome che assegnerete ad ogni file programma sara' posto in testa al programma.

### \*\*NOTA\*\*

Il discorso del nome e' riferito SOLO alla cassetta perche', come vedremo si puo' anche registrare un gruppo di informazioni SENZA dare nessun nome.

Non cosi' su disco.

Stante alle specifiche fornite dalle case costruttrici i computers CBM dovrebbero riconoscere nomi di files fino ad un massimo di 128 caratteri, ma solo i primi 16 caratteri sono visualizzati sullo schermo.

Tuttavia non si riesce a capire bene come facciano per i 128 caratteri in quanto da nessuna parte del sistema operativo di cassetta o sulla directory del disco e' possibile trovare tanto spazio.

Esperimenti condotti dagli autori in tal senso hanno dato sempre esito negativo.

I nomi dei files disco possono avere 16 o meno caratteri per questo sara' un buon principio di restringere tutti i



nomi di files a 16 caratteri o meno.

L' ammontare di memoria del vostro computer ha effetto sulla grandezza massima di un PROGRAM FILE.

Cio' e' perche' si crea un singolo file programma quando si salva ( SAVE) un programma su cassetta o su disco.

Quando si carica un programma in memoria si carica l' intero contenuto del file programma. Non si puo' caricare una parte di un file programma in memoria.

Per questo la grandezza massima di un file programma deve essere minore della capacita' di memoria programma del vostro computer.

La domanda e' come si faccia allora a caricare un programma molto grande.

Se e' necessario caricare un programma molto grande e quindi non e' disponibile una memoria del computer sufficientemente vasta si puo' suddividere il programma in tanti sottoprogrammi, ognuno dei quali potra' essere contenuto nella memoria del computer.

Quando ogni sezione del programma ha completato il lavoro, cioe' la sua esecuzione, semplicemente caricheremo la sezione successiva in memoria e la faremo girare.

In questo modo si puo' eseguire l' intero programma.

Successivamente descriveremo i passi necessari per eseguire un programma in questo modo.

Un vantaggio dei files programma e' che il salvataggio ed il caricamento avviene tramite il DOS, cioe' il DISK OPERATING SYSTEM, o per la cassetta, tramite il sistema operativo del computer.

Sara' quindi necessario applicare un identificatore al file programma, tramite il suo nome o la sua locazione, per caricarlo nella memoria, ma e' l' unica operazione necessaria.

FIGURA

| FILE MAILIG LIST |         |         |   |  |
|------------------|---------|---------|---|--|
| RECORD 1         | cognome | Campo 1 |   |  |
|                  | nome    | "       | 2 |  |
|                  | via     | "       | 3 |  |
|                  | CAP     | "       | 4 |  |
|                  | città   | "       | 5 |  |
| RECORD 2         | cognome | Campo 1 |   |  |
|                  | nome    | "       | 2 |  |
|                  | via     | "       | 3 |  |
|                  | CAP     | "       | 4 |  |
|                  | città   | "       | 5 |  |
| RECORD 3         | cognome | Campo 1 |   |  |
|                  | nome    | "       | 2 |  |
|                  | via     | "       | 3 |  |
|                  | CAP     | "       | 4 |  |
|                  | città   | "       | 5 |  |
| RECORD 4         | cognome | Campo 1 |   |  |
|                  | nome    | "       | 2 |  |
|                  | via     | "       | 3 |  |
|                  | CAP     | "       | 4 |  |
|                  | città   | "       | 5 |  |

## COMANDI PER I FILES PROGRAMMI

I comandi per i files programmi sono in verita' molto semplici ed essenzialmente si riducono a 3:

LOAD

SAVE

VERIFY

Il formato di LOAD e':

LOAD"nome del programma",d

dove:

d = il numero della periferica interessata. Nel caso della cassetta non e' necessario mettere il parametro d.  
Esempio

Caricare da cassetta il programma "MAILING LIST".

LOAD"MAILING LIST"

Il sistema rispondera' con un messaggio:

PRESS PLAY ON TAPE

e dopo aver eseguito l' operazione sul tasto indicato verra' visualizzato un:

OK

quindi il sistema iniziera' il caricamento del programma che sara' copiato dal nastro sulla memoria RAM del

computer.

### \*\*\* APPROFONDIMENTO \*\*\*

Il Sistema Operativo del computer genera automaticamente un' istruzione di OPEN (vedi dopo) usando l' indirizzo secondario appropriato per effettuare l' operazione di LOAD.

Sulla periferica cosi' attivato viene iniziata una ricerca per trovare il programma il cui nome e' specificato nell' istruzione LOAD.

Dopo che il programma e' stato trovato esso viene letto automaticamente dalla periferica e caricato nella memoria partendo dall' indirizzo specificato nell' intestazione del file.

Gli errori di lettura che possono verificarsi durante l' esame del primo blocco vengono automaticamente corretti dal secondo blocco perche' ricordiamo che le registrazioni sono fatte in doppio.

Alla fine del ciclo viene eseguita una somma di prova o:

#### CHECKSUM

Se si ha un errore di Checksum o se siamo in presenza di un errore che non e' correggibile il sistema operativo visualizza un messaggio di:

? LOAD ERROR

ed arresta il caricamento del programma.

Se il caricamento del programma viene fatto in modo diretto, alla fine dell' operazione viene eseguita la funzione di CLEAR (CLR) inizializzando tutte le variabili.

Se l' operazione di caricamento e' invece chiamata da un programma, allora il computer COMMODORE tratta questa operazione come una OVERLAY.

Il nuovo programma cioe' viene caricato nello spazio usato dal precedente programma, ma i valori di tutte le variabili vengono mantenuti uguali a quelli raggiunti nel programma precedente.

Cio' permette ad un programma di chiamarne un' altro passandogli tutti i parametri.

L' unico vincolo a quanto appena esposto sta nel fatto che il programma chiamato deve avere dimensioni uguali o minori del programma chiamante.

Poiche' il Basic rimpiazza completamente il programma corrente, non e' possibile avere un singolo programma principale e diverse subroutines in overlay. Tuttavia includendo il programma principale in ciascuna overlay, si puo' ottenere lo stesso effetto con una piccola perdita di velocita'.

L' uso dei nomi dei files e degli overlay permettono di scrivere dei programmi strutturati di grande ampiezza e con una notevole complessita'.

## VERIFY

Questo comando ha la stessa sintassi del comando LOAD:

VERIFY"nome del programma",d

In effetti l' istruzione di VERIFY e' un caso speciale di LOAD che dovrebbe venir eseguita dopo aver registrato qualsiasi programma.

Il comando VERIFY fa si che il Basic esegua le stesse operazioni dell' istruzione LOAD, con la differenza che i dati non vengono caricati in memoria ma vengono confrontati con il contenuto della memoria.

Se vengono incontrati degli errori, sia nel primo che nel

secondo passo, il computer visualizzera' un messaggio:

? VERIFY ERROR

e sara' necessario registrare una seconda volta il programma in quanto la copia precedente non e' utilizzabile.

SAVE

Anche l' istruzione SAVE provoca un' operazione di apertura e chiusura automatica di un file.

Ha il seguente formato:

SAVE"nome del programma",d

dove d e' il numero della periferica.

Se la periferica e' una unita' a nastro il Sistema Operativo del Computer inizia automaticamente a registrare, naturalmente dopo aver premuto i relativi tasti, un' intestazione ed apre un file su nastro con un nome appropriato.

Se il dispositivo e' un' unita' IEEE-488, cioe' di norma un disco, viene inviato uno speciale messaggio di apertura che sta ad indicare che il computer sta inviando un file programma.

Immediatamente dopo il programma viene scritto direttamente dalle sue locazioni di memoria o sul nastro o sulla periferica collegata alla porta IEEE-488.

**\*\*NOTA\*\***

Per quanto riguarda il caricamento di programmi su cassetta da parte di unita' VIC 20 e CBM 64 si ha un BLANK di schermo al momento stesso che inizia l'operazione di ricerca.

E' un problema che la COMMODORE avrebbe potuto benissimo evitare e che spesso induce il principiante a spaventarsi ingiustificatamente.

Quando il programma viene trovato si ha un' attesa di circa 30 secondi e poi il nastro riparte da solo.

Se si vuole evitare questa attesa e' sufficiente premere il tasto con il simbolo della COMMODORE posto a sinistra del computer.

## CAPITOLO QUINDICESIMO

### DATA FILES

Un DATA FILE o un files di dati come dovrebbe essere implicito nel nome, contiene informazioni che devono essere interpretati come dati in opposizione a comandi di programma.

I files dati sono creati, scritti e letti tramite i programmi, cioe' non possono essere scritti o letti direttamente come un file programma per mezzo delle istruzioni LOAD e SAVE.

### RECORDS E FIELDS

I data files sono divisi in records che a sua volta sono suddivisi in FIELDS o campi.

Un singolo field contiene informazioni che possono essere rappresentate tramite il nome di una singola variabile.

Per questo motivo un field puo' contenere un numero intero, un numero in virgola mobile o una singola variabile stringa.

Un record contiene uno o piu' field.

I records di solito rappresentano unita' d' informazioni ripetitive entro il file, ma puo' non essere sempre cosi'.

Prendiamo per esempio una mailing list.

L' intera mailig list puo' essere considerata come un singolo file di dati.

Ogni nome e indirizzo entro la mailing list sara' un record entro il file.

Un esempio di file che gestisce un indirizzario potrebbe essere caricato con i seguenti dati. In questo caso ogni record conterra' 5 campi:



Il cognome

Il nome

La via

Il CAP

La citta'

Un file puo' contenere uno o piu' record. Ogni record puo' contenere uno o piu' fields.

Il numero di records in un file e la massima lunghezza di un record varia con il tipo di file come descriviamo successivamente in questo manuale.

Tuttavia in pratica la grandezza di un file e' limitata solo dalla capacita' di memoria di massa.

Nessuna restrizione invece alla lunghezza di un record su cassetta.

Un record puo' avere una lunghezza qualsiasi che entri pero' nella lunghezza del nastro e questo per il semplice motivo che non puo' essere diviso in due nastri fisici.

### \*\*\* APPROFONDIMENTO \*\*\*

#### TRASFERIMENTO DATI VERSO E DA PERIFERICHE.

Istintivamente ci si puo' aspettare che sia il disco che il nastro si muovano in risposta ad ogni comando di lettura o di scrittura. Si intende come movimento fisico.

Cioe' l' unita' a cassetta dovrebbe muovere fisicamente il nastro e lo stesso per il disco.

Qualche volta si possono osservare entrambe queste

attività', mentre altre volta rimane tutto invisibile. Cio' e' dovuto al fatto che un piccolo ammontare di memoria agisce come DATA BUFFER connettendo il computer con l' unita' a cassetta o con il disco.

**\*\*NOTA\*\***

Il BUFFER e' una zona di memoria che serve di solito come magazzino temporaneo di dati.

Quando il computer legge da una di queste periferiche sono letti abbastanza dati per riempire il buffer.

I dati che devono essere scritti su cassetta o su disco sono prima scritti sul buffer data.

Non appena il data buffer e' pieno, tutto il contenuto del buffer e' inviato alla cassetta o al disco, ecco che allora vedremo qualche attività' delle periferiche. Vedremo cioe' qualcosa in funzione fisicamente.

Mentre non vedremo nessuna altra attività' fino a quando il BUFFER non sia riempito.

Il buffer di cassetta e' contenuto nella memoria del computer, e' lungo 192 bytes e puo' contenere percio' fino a 191 bytes di data.

Invece il buffer del disco e' nella stessa unita' a disco e non nella memoria del computer.

Ogni buffer data del disco e' grande 256 bytes e puo' manipolare fino a 254 bytes di dati.

I Buffer del disco e della cassetta sono relativamente grandi di conseguenza queste periferiche sono inattive per la maggior parte del tempo che il computer impiega per leggere o scrivere data.

Cio' e' di notevole vantaggio perche' consente in pratica di risparmiare la meccanica delle periferiche ed accelera inoltre le procedure di lettura/scrittura.

## FILES LOGICI E UNITA' FISICHE

Si usa il termine "INPUT/OUTPUT PROGRAMMING" per descrivere la logica di programmazione che consente il trasferimento dati fra il computer e le unita' esterne. I dischi, le cassette e le stampanti sono quindi unita' fisiche esterne.

Per consentire una qualsiasi operazione di INPUT/OUTPUT (ingresso/uscita) il programma deve identificare l'unita' fisica esterna alla quale si deve accedere.

Pensiamo il problema in termini di programmazione. Questo concetto e' facile da capire se si pensa alla tastiera ed al video come unita' esterne rispetto al computer propriamente detto, come in effetti esse sono.

Quando viene eseguito un comando INPUT i dati che abbiamo immesso con la tastiera sono specificati in un parametro di input.

Quando il comando :

```
10 INPUT A
```

e' eseguito, alcuni numeri che l'operatore fa entrare attraverso la tastiera sono assegnati a una variabile (in virgola mobile).

Nello stesso modo il comando di PRINT visualizzera' variabili o costanti.

Così il comando PRINT:

```
20 PRINT A
```

prende i valori assegnati alla variabile in virgola mobile A e mostra questo valore sullo schermo.

Quando viene eseguito un comando di INPUT l'unita' fisica esterna e' vista come nel caso precedente e' stato per la tastiera.

Quando viene eseguito invece un comando di PRINT l'unita' fisica esterna viene vista come il video. La programmazione degli INPUT/OUTPUT diventa molto piu' complessa quando i dati sono trasferiti da/a e dalla cassetta, il disco, la stampante e altre unita' fisiche esterne al di fuori della tastiera e del video. Per queste piu' complesse operazioni di INPUT o OUTPUT dovreste prima di tutto aprire un :

## CANALE DI COMUNICAZIONE

tra il programma e l'unita' fisica selezionata. Dopo aver eseguito l'operazione richiesta di INPUT/OUTPUT dovreste richiudere il canale. Il Basic del CBM identifica canali singoli usando un numero di canali che puo' andare da 0 a 255.

Si apre un canale usando il comando :

## OPEN

I Parametri di questo comando identificano l'unita' fisica alla quale si deve accedere, mentre per la natura di questo accesso vedremo in dettaglio nelle parti seguenti.

Fino a che il canale non sia chiuso, ogni comando di INPUT/OUTPUT necessita solo che sia specificato il numero del canale per descrivere completamente la natura della operazione di input o di output.

Ogni unita' fisica ha di per se un solo numero di riconoscimento fisico. Questo numero e' usato come un parametro quando si apre un canale per identificare l'unita' fisica alla quale si vuole accedere.

I numeri di canale sono per questo riportati frequentemente come "LOGICAL FILES NUMBERS" o "LOGICAL UNITS NUMBERS".

Il nome LOGICAL FILE describe un canale molto accuratamente, perche' un canale stabilisce un legame tra un programma e un file di dati.

I FILES LOGICI sono un concetto di programmazione.

Si puo' iniziare una qualsiasi operazione di I/O usando un comando di OPEN.

Uno dei parametri del comando OPEN e' il canale o il numero di FILE logico.

Gli altri parametri identificano l' unita' fisica, i dati ai quali si deve avere accesso ed il mezzo in cui occorre questo accesso.

Dopo che una operazione di input o output e' stata completata bisogna eseguire un comando CLOSE che richiudera' il canale.

Il comando CLOSE richiede solo un parametro:

#### IL CANALE O NUMERO DI FILE LOGICO

Questo numero di file logico unisce quindi un comando CLOSE ad un comando OPEN.

Tra un comando OPEN ed un CLOSE tutti i comandi di I/O usano un canale o un logical file number per identificare l' unita' alla quale si deve accedere e l' operazione che deve essere eseguita.

Il LOGICAL FILE NUMBER mette in relazione i comandi di :

OPEN

CLOSE

GET

PRINT

INPUT

Con qualsiasi altro.

Fino a quando state usando un numero di file logico in un comando, non potete riutilizzare lo stesso numero di file logico per fissare un diverso canale di I/O fino a che il LOGICAL FILE non sia chiuso.

Se lo farete il Basic del computer rispondera' con un :

## FILE OPEN ERROR

D'altra parte nessun'altra limitazione e' presente nel metodo di assegnare un numero di File Logico entro il vostro programma.

Il numero di DEVICE o di periferica identifica l'unita' fisica alla quale il computer inviera' i suoi dati o dalla quale li riceverà'.

Il numero di device appare come un parametro nel comando OPEN.

Ogni unita' fisica che possa comunicare con un computer CBM ha assegnato in permanenza un numero di DEVICE.

Non appena venga trovato un numero di device in un comando OPEN il computer attiva un'appropriata logica elettronica per stabilire una comunicazione con l'unita' specifica identificata nel numero di device.

Teoricamente sono disponibili 256 numeri di periferiche in un range compreso fra 0 e 255.

Tuttavia solo i numeri di device fra 0 e 30 sono correntemente usati.

In aggiunta alla definizione del numero di device, molte unita' fisiche rispondono ad un vasto gruppo di indirizzi secondari.

## TAVOLA

| Device                                  | Device Number                      | Secondary Address               | Operation Performed   |
|---|------------------------------------|---------------------------------|---|
| Keyboard                                | 0                                  | None                            |   |
| Cassette Drive #                        | 1 (Default)                        | 0<br>1                          | Open per letture<br>Open per scritture  |
| Cassette Drive #2                       | 2                                  | 2                               | Open per scritture ma con EOI dopo la chiusura.   |
| Video Display                           | 3                                  | None                            |   |
| Line Printer Models 2022 and 2023       | 4                                  | 0<br>1<br>2<br>3<br>4<br>5<br>6 | VEDERE CAPITOLO STAMPANTI   |
| Disk Drives (all models)                | 8                                  | 0<br>1<br>2-14<br>15            | LOAD di un file programma<br>SAVE di un file programma<br>Non assegnati<br>OPEN sul canale di comando |
| Other devices connected to IEEE 488 Bus | 5,6,7 and 9 through 31             |                                 | VEDERE SPECIFICHE SUI MANUALI DELLE SINGOLE PERIFERICHE   |
|   | 32 to 255 unavailable at this time |                                 |   |

- Numero di periferiche con indirizzo secondario.

## INDIRIZZO SECONDARIO

Oltre ad avere un numero di unita' fisica a molte periferiche puo' essere assegnato un indirizzo secondario o SECONDARY ADRESS.

L' indirizzo secondario e' un comando che parte dal computer e che dice all' unita' fisica quale operazione deve prepararsi ad eseguire.

Gli indirizzi secondari sono riportati in sommario in appendice per le unita' fisiche che sono piu' comunemente connesse ad un computer CBM.

Non dovrete impegnarvi in uno studio particolare degli indirizzi secondari, perche' successivamente quando descriveremo i programmi di I/O in dettaglio la funzione di indirizzo secondario diventera' familiare ed ovvia per il suo frequente uso.

Il programma seguente illustra molto bene l' uso dei parametri nei comandi di I/O.

```
100 OPEN 4,1,2,"MAILING LIST"  
200 PRINT#4,CN$  
210 PRINT#4,NO$  
220 PRINT#4,VP$  
230 PRINT#4,CA$  
240 PRINT#4,LO$  
300 CLOSE4
```

I 5 comandi di PRINT# che appaiono nelle linee dalla 200 alla 240 scrivono 5 parti di nome ed indirizzo in un file chiamato :

### MAILING LIST

localizzato su un nastro dell' unita' a cassetta.

Tutte le volte che si incontra un comando di PRINT #i l



computer sa cosa deve fare perche' controlla il numero di File logico che appare dopo il carattere# .

Nel programma questo numero di File logico e' 4, percio' nel comando di OPEN e' specificato il file logico 4 che descrive la natura dell' operazione.

Questo comando di OPEN e' presente nella linea 100 del nostro programma.

Se il computer non dovesse trovare un comando di OPEN con il richiesto numero di unita' logica, questi non potrebbe mettere in funzione le operazioni di I/O poiche' non saprebbe cosa fare.

Nel programma c'e' un comando di OPEN sul File Logico n. 4. Questo comando specifica l' unita' logica n. 1 che appunto sta ad indicare che e' selezionata la cassetta.

L' indirizzo secondario e' 2 perche' in questa occasione e' possibile scrivere sulla cassetta del drive 1 ma non e' possibile leggerci.

Quando questa operazione e' chiusa verra' scritto un fine nastro sulla cassetta per prevenire che un qualsiasi dato possa essere successivamente aggiunto.

Il comando OPEN specifica inoltre che il Data file al quale si deve accedere ha il nome MAILIG LIST.

Sulla linea 300 e' presente un comando di CLOSE (chiudi). Questo comando specifica il numero 4 come File Logico, di conseguenza tutto quanto e' stato aperto con il comando OPEN nella linea 100 sara' chiuso con questo comando alla linea 300.

Poiche' il comando OPEN alla linea 100 specifica un numero di indirizzo secondario 2, il comando CLOSE alla linea 300, quando sara' eseguito causera' una EOF (END OF FILE) sulla cassetta.

In questo modo il file logico n 4 che e' presente nei comandi dalle linee 200 alla linea 300 congiunge questi comandi con un comando OPEN alla linea 100.

Parametri addizionali appaiono sui comandi OPEN alla linea 100 per descrivere le operazioni che devono essere eseguite.

Prima di procedere oltre con la programmazione vediamo alcuni concetti, in particolare per le variabili di

controllo che sono essenziali per la gestione delle periferiche.

## FISICAL UNIT STATUS

Una stampante puo' ricevere informazioni da un computer, cioe' si possono preparare delle stringhe da far stampare su una stampante, tuttavia da una stampante i dati non possono passare ad un computer.

Per questo motivo non e' necessario specificare il numero di indirizzo secondario quando si esegue un comando di OPEN su una periferica tipo stampante.

Al contrario una cassetta puo' ricevere dati dal computer o trasmetterglieli, per cui l' indirizzo secondario usato nel comando OPEN che inizializza la cassetta dovra' specificare se l' operazione e' di lettura o di scrittura.

Quando si esegue un comando di PRINT , GET o INPUT e' necessario fare attenzione a quello che si vuol fare. In altre parole non sara' possibile eseguire dei comandi di INPUT o di GET quando l' unita' a cassetta sara' stata preparata solo per operazioni di scrittura.

Se questo dovesse avvenire avremo una registrazione di errore di STATUS.

L' unita' fisica riporta l' informazione sullo status di seguito ad ogni operazione di INPUT o di OUTPUT quando questa sia stata eseguita con successo o con insuccesso.

In pratica tutte le volte che si accede ad una unita' periferica, considerando pero' in questo caso come periferiche anche la tastiera ed il video, viene attivato un registro di 8 bit che e' appunto:

## REGISTRO DI STATUS

Questo registro ha come riferimento la variabile Basic ST. Per esempio il comando 10:

10 X= ST

Assegnera' al registro di status il valore della variabile X.

Per quanto riguarda le cassette riportiamo gli errori che tramite ST possono essere rilevati:

ST=4 Blocco Corto.

Leggendo un blocco di dati da nastro si incontra un segnali di spaziatura prima di aver letto i caratteri che ci si aspettava di trovare.

ST=8 Blocco Lungo.

Leggendo un blocco non si trova il segnale di spaziatura dopo aver letto il numero di caratteri che si aspettava di trovare.

ST=16 Errore di lettura irrecuperabile.

Si sono riscontrati piu' di 31 errori nel primo blocco del blocco di controllo oppure si e' trovato un errore che non puo' essere corretto perche' presente in tutti e due i blocchi di registrazione.

Ricordiamo che la registrazione su nastro e' SEMPRE doppia.

ST=32 Errore di Checksum.

Durante la lettura o il LOAD di dati viene calcolata una somma di controllo sui bits dei bytes letti e viene confrontata con la somma di controllo registrata sul nastro al momento della scrittura. Se viene generato questo errore le due somme non coincidono.

**\*\*NOTA\*\***

Questi ultimi due non sono, come si vede dalla descrizione dei veri e propri errori.

ST=64 End of File.

Viene cioe' incontrato il fine di un file.

ST=-128 End of Tape.

Viene incontrato il segnale di fine nastro.

Da quanto detto e' evidente che il nostro computer CBM non e' in grado di riconoscere errori durante la scrittura su nastro.

La normale tecnica di programmazione e' quella che fa eseguire ad un' istruzione di INPUT#o GET#un test sul valore dello stesso. Poiche' la parola di stato e lo stato cambia per ciascun nuovo comando di I/O lo stato e' una grandezza variabile rapidamente. Ad esempio:

```
100 INPUT#2,A
110 INPUT#5,B
120 IF TS=0 THEN 200
```

Questo programma testa il risultato del trasferimento di dati SOLO dal file logico 5. Il risultato della lettura del file logico 2 e' perso.

Un sistema corretto per usare la parola ST e' la seguente:

```
100 INPUT#2 ,A,B,C
110 IF ST=0 THEN 200
120 IF ST=64 THEN 300
130 IF ST=2 THEN 400
```

In tal modo ciascun errore puo' essere individuato e si possono intraprendere le successive operazioni che si rendano necessarie attraverso l' istruzione:

```
140 IF ST AND mask THEN nnn
```

dove mask rappresenta il bit che si vuol testare.

| Device<br>Operation                     | Status                               |   |  |   |                             |  |                            |
|---|--------------------------------------|---|--|---|-----------------------------|--|----------------------------|
|   | 00000001<br>Read as 1                | 00000010<br>Read as 2                   | 00000100<br>Read as 4  | 00001000<br>Read as 8   | 00010000<br>Read as 16      | 00100000<br>Read as 32   | 01000000<br>Read as 64     |
| Read from<br>Cassette drive<br>#1 or #2 | Operation OK                         | Operation OK                            | Short Block.<br>Data Block<br>read had<br>fewer bytes<br>than expected | Long Block<br>Data Block<br>read had<br>more bytes<br>than expected | Unrecoverable<br>read error | Checksum<br>error. One<br>or more data<br>bits read<br>incorrectly | End of file<br>encountered |
| Verify<br>cassette drive<br>#1 or #2    |                                      |   |  |   | Any verify<br>mismatch      |  | None                       |
| Disk<br>drives<br>(all models)          | Receiving<br>device not<br>available | Transmitting<br>device not<br>available | None   | None  | None                        | None   | Disk drive<br>not present  |
| IEEE 488<br>Bus                         | Time out<br>on listener              | Time out<br>on talker                   |  |   |                             |  | Device<br>not present      |

- Valore del byte di Status letto da periferiche con la variabile SI.

## MANIPOLAZIONE DI DATI SU CASSETTA

Veniamo ora a descrivere i passi di programma necessari per la manipolazione dei files su cassetta. Descriveremo come i FILES DI DATI sono creati, letti o modificati sotto controllo di programma.

Molti dei comandi Basic che appaiono in questo capitolo sono dati per scontati, cioè si suppone che il lettore abbia una discreta conoscenza generale del Basic.

Potete programmare il computer per scrivere dati su cassetta o per rileggerli, ma non potete programmare il movimento fisico della cassetta.

E' importante che comprendiate il modo in cui opera fisicamente il DRIVE, cioè l' unita'. In altre parole dovete tenere presente che per eseguire operazioni sulla cassetta non sarete mai in grado di manipolare il movimento fisico del nastro.

### \*\*NOTA\*\*

In effetti questo discorso non e' completamente vero perche' si puo' programmare, ad esempio agendo su determinati registri, l' arresto del motore.

I files sono immagazzinati in modo sequenziale su nastro, cioè uno di seguito all' altro.

Un HEADER cioè una testata, precede il primo file e un fine nastro (EOT) segue l' ultimo file. Ogni fine di file e' segnato da un EOF.

La testata e' scritta automaticamente all' inizio del nastro, cioè quando ci scrivete per la prima volta.

A questo punto potete notare che l' attivita' della cassetta o almeno questa parte di attivita' della cassetta, non vi riguarda. In altre parole l' esistenza di un HEADER viene scritta automaticamente, cioè non ha bisogno di vostre operazioni.

Il computer puo' trovare i File mentre il nastro sta

girando piano, cioe' alla velocita':

PLAY

ma non e' in grado di trovarli quando sta girando in FF, cioe' in :

FAST FORWARD

E questo perche' durante la fase di LETTURA/SCRITTURA non e' in contatto fisicamente con il nastro.

Come abbiamo detto la fine file e' identificata da un segno particolare ( o meglio da un carattere particolare).

In altre parole, come si puo' vedere dalla precedente tabella, uno status pari a 64 identifica un END OF FILE. Uno Status di -128 identifica una fine nastro.

Il computer non puo' eseguire una operazione di riavvolgimento diretto veloce ne puo' trovare niente sulla cassetta mentre il nastro si sta riavvolgendo.

Si deve iniziare il movimento sulla cassetta manualmente premendo il tasto relativo in seguito ad una istruzione fornita dalla unita' centrale.

Si raccomanda di non premere nessun tasto prima che un messaggio venga visualizzato.

In seguito potremo comportarci diversamente dopo aver pero' preso un po' di pratica nelle operazioni.

Esaminiamo ora l' impatto sulle operazioni dell' unita' a cassetta.

Quando si stanno scrivendo dati sulla cassetta, il nastro deve essere correttamente posizionato ad inizio scrittura e cio' e' messo sotto la responsabilita' dell' operatore. A questo punto e' necessario ricordarsi che se non si posiziona correttamente il nastro e' facile avere delle sovrascritture.



Inoltre se la parte iniziale trasparente e' posizionata sulla testina di scrittura, l' unita' cerchera' di scrivere le informazioni che pero' non saranno registrate.

E' importante ricordarsi di questo perche' il computer non e' capace di distinguere la superfice magnetica dalla superfice non magnetica.

Il metodo che consente la massima sicurezza e' di iniziare a scrivere su nastro vergine o su una cassetta i cui dati non servano piu' e di posizionare il nastro all' inizio della superfice magnetica.

Si possono cosi' tranquillamente scrivere records e files uno dietro l' altro fino al termine fisico del nastro stesso.

Il Sistema Operativo dell' unita' centrale nella parte relativa all' uso dell' unita' a dischi si assicurera' che venga lasciato uno spazio sufficiente fra la fine di un record o di un file e l' inizio del successivo, per cui non sara' necessario che l' operatore si occupi di questo.

Quando si leggono files di dati gia' registrati, e' necessario assicurarsi che il nastro sia riavvolto fino all' inizio del primo file che si vuole rileggere.

Il computer puo' trovare un qualsiasi definito file di dati purché questo sia DOPO il punto in cui l' abbiamo fatto partire, ma non puo' certo tornare indietro a cercarselo da solo.

Non si deve mai cercare di riscrivere anche una piccola parte di file su nastro perche' l' operazione e' troppo rischiosa.

Supponiamo per esempio di aver immagazzinato su un file cassetta 10 nomi ed indirizzi e che si desideri variare il quinto nome ed il relativo indirizzo.

Teoricamente, si potrebbe leggere i primi 4 nomi ed indirizzi e questa operazione ci dovrebbe lasciare il nastro posizionato all' inizio del quinto nome.

Si potrebbe quindi scrivere il nuovo quinto nome sul vecchio.

In pratica e' meglio non farlo.

Infatti l' unita' a cassetta non e' molto precisa e c'

e' una buona probabilita' che il nuovo nome ed indirizzo sia scritto un po' prima o un po' dopo del vecchio.

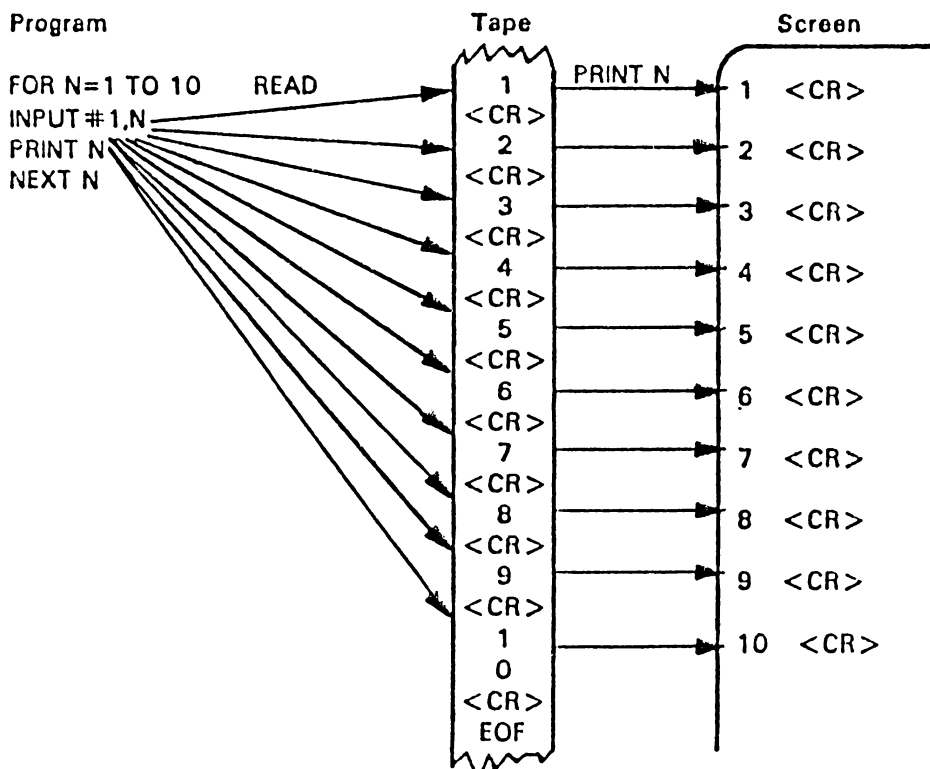
Infatti, a parte il fatto che dovrebbero essere della stessa identica lunghezza e questo risulta difficile da ottenere, e' sufficiente che un solo carattere vada fuori posto, cioe' o troppo prima o dopo, che non saremo in grado di rileggere i dati.

Per aggiornare quindi un file di dati e' necessario caricare il file stesso nella memoria del computer, aggiornarlo e quindi riscriverlo.

E' molto improbabile che sul CBM 64 a differenza di quanto avveniva per il VIC 20, sia necessaria una espansione di memoria per cui l' utilizzo della ricordata tecnica di OVERLAY dovrebbe essere piu' che sufficiente.

Potrebbe sembrare che la trattazione di questi problemi sia stata troppo lunga, ma per esperienza diretta e per le numerose lettere che ci sono pervenute, possiamo assicurarvi che i guai sulla registrazioni su nastro possono far perdere piu' tempo che non la scrittura stessa dei programmi.

# TAVOLA



- Diagramma di distribuzione e di comparazione di dati numerici su nastro e su schermo.

## IL FORMATO DEI FILE SU CASSETTA

La descrizione dei file dati presentata all' inizio di questo capitolo e' una descrizione concettualmente esatta e completa di come i dati sono trattati in generale da un computer.

I file di dati sono suddivisi in RECORD e FIELDS.

Si puo' mantenere questo tipo di organizzazione che definiremo CLASSICA usando dei programmi appropriati e ci permettiamo di raccomandarvelo.

### FIELDS NUMERICI

Ogni field numerico deve essere seguito da un ritorno carrello effettuato con l' inserzione del carattere (CHR\$(13)).

Percio' un file che consiste solo di campi numerici dovrebbe essere visto come una sequenza di numeri separati da un carattere di ritorno carrello come e' illustrato nell' esempio seguente dove N sta' per il numero e CR per il carattere di ritorno carrello:

```
N--CR--N--CR--N--CR
```

Per questo all' interno di un file di dati puramente numerico non esiste nessuna ripartizione di fields in record o distinzione fra i vari records.

E' interamente demandato quindi alla logica del programma la formazione di record intesi come sequenza di fields sempre che questo sia necessario.

Le variabili stringa da memorizzarle possono invece essere divisi facilmente in fields e questi eventualmente raggruppati in records.

Si possono usare i due punti (CHR\$(44)) per separare i fields entro un record, mentre un ritorno carrello (CHR\$(13)) seguira' l' ultimo field del record stesso. Nell' esempio seguente viene mostrato la struttura fisica di registrazione su nastro di un file che contiene

solo variabili stringa con 5 file per record. Ricordiamo che, come nell' esempio precedente CR sta per il ritorno carrello mentre S per la stringa:

CR--S---CR--S---CR--S---

Se si usano i due punti ed il ritorno carrello come separatori per dividere il file stringa in Fields e record come illustrato precedentemente, allora tutti i fields di ogni record devono essere letti con un singolo comando di INPUT£.

Non e' obbligatorio usare come separatori di variabili stringa i due punti e il ritorno carrello, ma sara' probabilmente meglio il tenere separato tutti i campi di una variabile stringa usando un ritorno carrello. Come per i file di dati numerici,e' rimesso alla logica del programma che dovra' gestirli il raggruppamento di campi in records.

Prendendo come esempio il programma di gestione indirizzi si vede come la logica di programmazione richieda essa stessa che vari fields siano organizzati in record in maniera evidente.

Non e' necessario insegnare come un programmatore debba vedere che ogni nome ed indirizzo diventa un record, mentre parte del nome e dell' indirizzo devono essere trattati come singolo field.

Le strade per dividere un nome ed indirizzo nei singoli fields sono numerose e in pratica, purché risponda allo scopo che ci siamo prefissi una strada vale l' altra.

L' organizzazione di un file deve essere dettata dalle necessita' del programma piuttosto che dalla struttura dei data files su cassetta del vostro computer.

Le difficoltà eventuali di programmazione saranno relative alla sintassi dei comandi PRINT#eINPUT#.

Ora partendo da un semplice programma cercheremo di approfondire la sintassi dei comandi e tramite piccole continue modifiche spiegheremo cio' che e' concesso e cio' che non e' concesso fare.

Digitiamo il seguente programma:

```

10 OPEN1,1,1
20 FORI=1TO10
30 PRINT#1,I+100
40 NEXT
50 CLOSE 1
60 STOP
70 OPEN1
80 FORI=1TO10
90 INPUT#1,J
100 PRINTJ
110 NEXT
120 CLOSE1
132 STOP

```

Il comando OPEN alla linea 10 apre il file logico 1, selezionando l' unita' a cassetta per una operazione di scrittura.

Il ciclo di FOR-NEXT alle linee 20,30 e 40 scrivono 10 numeri su nastro.

I numeri sono seguiti da un carattere di ritorno carrello per far si che il comando di PRINT# alla linea 30 forzi un ritorno carrello ad ogni scrittura su nastro, allo stesso modo che un comando di PRINT forza un a capo sullo schermo.

Il file logico e' chiuso alla linea 50.

La struttura fisica della registrazione su nastro sara' la seguente:

CR--1--CR--2--CR--3...10--CR

I comandi dalla linea 70 alla linea 120 leggono e visualizzano i dieci numeri che erano stati scritti su nastro dai comandi che vanno dalle linea 20 alla linea 50. Eseguite il programma e osservate cosa succede seguendo le indicazioni che daremo.

Digitate il programma e salvatelo su una cassetta programmi. Prendete poi un cassetta vergine ed inseritela facendo in modo che la superfice magnetica della cassetta sia posizionata sulla finestrella perche' altrimenti scriverebbe sulla parte bianca/trasparente del nastro e quindi non scriverebbe affatto.

Assicuratevi che nessun tasto sia premuto e digitate RUN.  
Verra' visualizzato il seguente messaggio:

PRESS PLAY AND RECORD ON TAPE

Premere i tasti indicati sulla unita' a cassette. Il computer visualizzera' un :

OK

sotto la frase precedente.

Il nastro comincera' a girare mentre i numeri da 101 a 110 saranno registrati.

Dopo che i 10 numeri sono stati scritti la cassetta si ferma e sullo schermo viene visualizzato il seguente messaggio:

BREAK IN 60  
READY.

con il cursore che lampeggia sotto la scritta Ready.

Il comando di STOP alla linea 60 ha causato l'interruzione.

Premere ora il tasto di STOP della cassetta per far rialzare i tasti di PLAY e di RECORD.

Riavvolgere la cassetta con il tasto REWIND e ripremere il tasto STOP per far rialzare il REWIND. Infatti anche se quest' ultimo si rialza da se la tensione sul nastro puo' far si che questo si spezzi.

Eseguiamo la seconda parte del programma digitando:

GOTO 70

Sara' visualizzato il messaggio:

PRESS PLAY ON TAPE 1

Fate attenzione a premere SOLO il PLAY sulla cassetta ed, il computer rispondera' con un:

OK

dopo l' operazione il nastro comincerà a girare e dopo aver superato la parte bianca della cassetta e letto i 10 numeri scritti prima visualizzerà su una colonna verticale dello schermo:

```
101
102
103
104
105
106
107
108
109
110
BREAK IN 130
READY.
```

Il messaggio finale è dovuto all' esecuzione del comando STOP che è presente alla linea 130 del nostro programma. Se avete dimenticato di riavvolgere il nastro prima di digitare GOTO 70, l' unità cercherà per tutto il nastro, che si è detto doveva essere vergine per questo esempio, e non trovando niente andrà fino alla fine fisica del nastro.

Se siete incorsi in questo errore dovete per prima cosa premere il tasto STOP della unità a cassetta e poi fermare il programma premendo il RUN/STOP del calcolatore.

Successivamente riavvolgere il nastro, ma prima di far ripartire il programma dovete eseguire una operazione di chiusura in forma diretta digitando:

CLOSE 1

e quindi ripartire con un:

GOTO 70



Chiarito il primo errore che si può commettere e che è più frequente di quanto non si creda passiamo ad effettuare delle piccole modifiche al nostro programma. Listiamo il programma ed aggiungiamo al comando PRINT nella linea 100 un punto e virgola in modo da avere:

```
100 PRINT J;
```

Riavvolgiamo il nastro e digitiamo di nuovo:

```
GOTO 70
```

Dopo avere eseguito l'operazione di lettura come precedentemente descritto sullo schermo apparirà:

```
101 102 103 104 105 106 107 108 109 110
```

```
BREAK IN 130  
READY.
```

A titolo sperimentale proviamo a cambiare i comandi dalla linea 80 alla 110 in modo tale che i dieci numeri siano inseriti usando un solo comando di INPUT, per cui il nostro programma sarà ora come segue:

```
10 OPEN1,1,1  
20 FORI=1TO10  
30 PRINT#1,I+100  
40 NEXT  
50 CLOSE 1  
60 STOP  
70 OPEN1  
80 FORI=1TO10  
90 INPUT#1,J  
100 PRINTJ  
110 NEXT  
120 CLOSE1  
132 STOP
```

Riavvolgiamo ancora la cassetta ed eseguiamo la seconda parte del programma, cioè la parte di sola lettura. Come nell' esempio appena visto sull' uso del punto e virgola, i dieci numeri saranno letti dal nastro e visualizzati in una sola riga.

Questo esempio serve a dimostrare che non esiste nessuna differenza nella lettura dei dieci numeri sia eseguendo il comando INPUT# con 10 variabili come suoi parametri che eseguendo lo stesso comando con una sola variabile ma 10 volte.

Continuiamo il nostro ciclo di esperimenti modificando il sistema di separazione all' interno del file numerico. Proviamo ora a cambiare la prima parte, cioè quella relativa alla scrittura. Il programma sarà come segue:

```
10 OPEN 1,1,1
20 FOR I =1 TO 10
30 M(I) =I + 100
40 NEXT
45 C$=CHR$(59)
46 PRINT#1, M(1);C$; M(2);C$; M(3);C$; M(4);C$; M(5)
47 PRINT#1, M(6);C$; M(7);C$; M(8);C$; M(9);C$; M(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 FOR I =1 TO 10
90 INPUT#1, J
100 PRINT J
110 NEXT
120 CLOSE 1
130 STOP
```

dove CHR\$(59) rappresenta un punto e virgola(;).

Eseguiamo nuovamente l' operazione di scrittura nastro ricordando di far apparire sulla finestrella la prima parte della superficie magnetica.

Dopo che sarà apparso il solito:

## PRESS PLAY AND RECORD ON TAPE

eseguiamo le operazioni abituali e dopo la registrazione avremo:

BREAK IN 60  
READY.

Riavvolgiamo il nastro e digitiamo il GOTO 60 per far eseguire l'operazione di lettura.  
Dopo aver premuto il tasto PLAY sulla cassetta sarà visualizzata una scritta:

FILE DATA ERROR IN 90  
READY

Cio' vorrà dire che i dati non sono stati letti correttamente. Perché'?

Perché' non si può usare nessuna altra forma di punteggiatura e quindi di separazione nel trattamento di dati numerici all'infuori del ritorno carrello.

Si possono invece usare i due punti o il ritorno carrello per separare i campi in una stringa.

Proviamo a cambiare il programma come segue:

5 DATA UNO, DUE, TRE, QUATTRO, CINQUE, SEI, SETTE, OTTO,  
NOVE, DIECI

6 REM -----

10 OPEN 1,1,1

20 FOR I =1 TO 10

30 READ M\$(I)

40 NEXT

45 C\$=CHR\$(44)

46 PRINT#1, M\$(1);C\$; M\$(2);C\$; M\$(3);C\$; M\$(4);C\$; M\$(5)

47 PRINT#1, M\$(6);C\$; M\$(7);C\$; M\$(8);C\$; M\$(9);C\$;

M\$(10)

50 CLOSE 1

60 STOP

70 OPEN 1

```
80 FOR I =1 TO 10
90 INPUT#1, J$
100 PRINT J$
110 NEXT
120 CLOSE 1
130 STOP
```

Riavvolgiamo quindi il nastro e eseguiamo la prima parte di scrittura del programma.

Vedremo che i dati saranno scritti correttamente e sarà visualizzata la scritta:

```
BREAK IN 60
READY
```

A questo punto riavvolgiamo la cassetta ed eseguiamo il solito GOTO 70

Dopo aver premuto il tasto PLAY avremo la seguente visualizzazione:

```
UNO SEI
```

ed immediatamente sotto sullo schermo:

```
STRING TOO LONG ERROR IN 90
READY.
```

Cosa è successo?

Il problema sta nel comando INPUT #della linea 90.

Un comando INPUT #leggerà tutti i campi della stringa fino al primo ritorno carrello.

Quindi le variabili da M\$(1) a M\$(5) sono in INPUT alla prima esecuzione del comando INPUT #alla riga 90. Tuttavia solo il valore di M\$(1) è stato letto da J\$ perché la virgola è interpretata come un separatore di campo e non come un segnale di termine.

La seconda volta il comando INPUT # alla linea 90 e' eseguito, da M\$(6) fino a M\$(10) sono in input, poiche' questi sono due campi situati fra due ritorni carrello. Ancora una volta solo M\$(6) e' assegnato a J\$ poiche' la virgola e' interpretata come campo.

La terza volta che il comando INPUT# alla linea 90 viene eseguito non ci sono piu' dati da leggere e viene segnalato pertanto un errore.

Per risolvere questo problema e' necessario eseguire i comandi di INPUT # con lo stesso numero di variabili presenti nel comando PRINT#.

Consideriamo il seguente programma:

```
5 DATA UNO, DUE, TRE, QUATTRO, CINQUE, SEI, SETTE, OTTO,
NOVE, DIECI
6 REM -----
10 OPEN 1,1,1
20 FOR I = 1 TO 10
30 READ M$(I)
40 NEXT
45 C$=CHR$(44)
46 PRINT#1, M$(1);C$; M$(2);C$; M$(3);C$; M$(4);C$; M$(5)
47 PRINT#1, M$(6);C$; M$(7);C$; M$(8);C$; M$(9);C$;
M$(10)
50 CLOSE 1
60 STOP
70 OPEN 1
80 INPUT#1, N$(1), N$(2), N$(3), N$(4), N$(5)
90 INPUT#1, N$(6), N$(7), N$(8), N$(9), N$(10)
100 FOR I = 1 TO 10
105 PRINT N$(I);" ";
110 NEXT
120 CLOSE 1
130 STOP
```

Se non commetteremo nessun errore nel posizionamento del nastro avremo, stavolta correttamente:

UNO DUE TRE QUATTRO ecc.

sul video seguiti da:

BREAK IN 130  
READY.

E' probabile che siamo stati noiosi, ma vi assicuriamo che la manipolazione dei dati su cassetta o su disco e' veramente la parte piu' importante della programmazione. Per questo vi consigliamo di continuare con gli esperimenti fra cui vi consigliamo:

1)Puo' un singolo comando di INPUT #leggere un gruppo di variabili stringa separate da un ritorno carrello?

Per provare basta cambiare nell' ultimo programma la linea 45 in modo tale che a C\$ sia assegnato il valor di CHR\$(13).

2)Cosa accade se si mischiano numeri e stringhe in un singolo file di dati?

3)Creare 10 variabili stringa M\$(I) come nell' ultimo programma, ma aggiungere 10 variabili numeriche M(I) aggiungendo per esempio:

35 M(I)=I+100

## LETTURA DI DATI DA CASSETTA

Ci sono tre passi di programma necessari per leggere un file dati da cassetta:

APRIRE IL FILE con un OPEN

LEGGERE IL FILE con INPUT #

CHIUDERE IL FILE con un CLOSE

Un file di dati deve essere aperto per la lettura utilizzando lo stesso nome (NAME FILE) che era stato adoperato per scriverlo.

Mentre puo' essere assegnato un diverso numero di file logico, l' indirizzo secondario deve essere posto a 0 appunto per l' opzione di lettura.

Ricordiamo come regola generale:

SCRITTURA: OPEN1,1,2,"DATA"

LETTURA: OPEN1,1,0,"DATA"

Sono disponibili due comandi di lettura da cassetta:

INPUT # e GET #

Per leggere un campo (FIELD) numerico o una stringa useremo il comando INPUT#.

Il comando GET# leggerà invece un carattere per volta.

Ricordiamo poi di eseguire un CLOSE dopo che il file e' stato letto.

E naturalmente chiudiamo lo stesso file logico che e' stato aperto. per cui nell' esempio di prima:

CLOSE 1

Un buon sistema di chiudere un file e' quello di controllare l' esistenza di un carattere EOF (end-of-file) attraverso il registro di STATUS.

Quando un file viene scritto, un carattere di EOF viene posto alla fine del file.

Quando viene letto un carattere di EOF, il registro di status assume il valore di 64 ed il file puo' essere chiuso.

Si puo' controllare con questo semplice comando inserito alla fine della lettura:

```
IF ST=64 THEN CLOSE 1
```

Cioe' quando lo status e' 64 il file viene chiuso.

Precedentemente abbiamo scritto un programma per scrivere una serie di numeri da 1 a 10 in un file dati su cassetta chiamato.

Ora scriveremo un programma per leggere i dieci numeri dal file di dati NUMBERS e visualizzarli sullo schermo.

```
10 PRINT"** LETTURA DI DATI NUMERICI"  
15 PRINT  
20 PRINT"INSERIRE LA CASSETTA E PREMERE IL RETURN QUANDO  
SIETE PRONTI":  
25 PRINT  
30 GET A$:IFA$=""THEN 30  
40 PRINT"APERTURA":OPEN1,1,0 "NUMERI"  
45 PRINT  
50 FOR I=1 TO10  
60 INPUT# 1,N  
70 PRINT N  
80 NEXT I  
90 PRINT"CHIUSURA": CLOSE 1  
100 END
```

Il comando INPUT #legge un campo per volta.

Le prime 3 linee di del programma dicono all' utente come deve operare.

I comandi sono identici a quelli del programma di scrittura.



Alla linea 30 c'e' un loop di attesa che da all' operatore il tempo di montare il nastro sulla cassetta. Dopo il montaggio fisico del nastro, premere RETURN e il programma passa alla linea successiva. Prima che ogni dato sia letto, il file deve essere aperto.

I comandi alla linea 40 apriranno il FILE#1,sulla periferica 1, con l' indirizzo secondario 0 per l' operazione di lettura sul file con il nome NUMBERS.

Nelle linee da 50 a 80, il cuore del programma, con il ciclo FOR-NEXT vengono letti i primi 10 dati dal nastro e visualizzati sullo schermo.

Il comando INPUT#1 della linea 60 legge un numero per l' esecuzione.

Dopo che i dati sono stati letti il file viene chiuso alla linea 90, mentre alla linea 100 si trova un END che ricordiamo pero' e' opzionale.

Il comando INPUT#1 puo' anche leggere campi che contengano variabili stringa.

Come abbiamo visto in precedenza con il programma di numeri alfabetizzati, invece dei numeri nella loro rappresentazione decimale li abbiamo scritto in forma letterale.

Per leggere le stringhe, in questo caso, e' sufficiente una piccola modifica al programma di lettura precedente. Come possiamo vedere dal listato che segue sono necessari solo dei cambiamenti alla linea 40 per fargli rileggere un file diverso da quello di prima ed alla linea 60 per far rileggere una stringa anziche' un numero.

Cambieremo pertanto:

```
60 INPUT#1,N
70 PRINTN
```

con:

```
60 INPUT#1,N$
70 PRINTN$
```

## CAPITOLO SEDICESIMO

### PROGRAMMAZIONE DI FILE DI DATI

Per accedere ai dati su cassetta sono necessari tre blocchi di programma:

-1 Aprire il File (OPEN)

-2 Eseguire un comando di lettura o scrittura ( INPUT o PRINT)

-3 Chiudere il file (CLOSE)

vediamo i singoli passi uno per uno.

#### APERTURA O OPEN

Si deve aprire un comando OPEN per aprire un file di dati qualsiasi operazione si desideri eseguire.

Il formato di questo comando e':

OPEN N,D,S,Nome del File

Cioe' apri il file logico N, seleziona sulla periferica D il file di dati scelto con "Nome del file" e predisponi per eseguire l' operazione specificata con l' indirizzo secondario S.

Si puo' usare il comando OPEN con una qualsiasi combinazione di questi parametri.

N e' il solo parametro che deve essere presente, mentre

se D e' assente viene ritenuto uguale a 1.  
Se e' assente S viene assunto = 0, mentre se manca il  
nome del file verra' selezionato il primo file che si  
incontra sulla cassetta.  
Quando viene eseguito un comando OPEN su cassetta per la  
lettura di dati, verra' visualizzato il seguente  
messaggio:

PRESS PLAY ON TAPE

e dopo che il tasto della cassetta sara' premuto sara'  
visualizzato:

OK

Il computer incomincera' allora a leggere il nastro.  
Nell' ipotesi che il comando sia dato in modo immediato e  
che il file cercato non sia il primo sul quale si e'  
posizionata la testina di lettura, avremo la  
visualizzazione dei seguenti messaggi:

SEARCHING FOR (Nome del file)

FOUND Test

FOUND Ross

FOUND Chess

FOUND

dove i primi tre stanno ad indicare i nomi di files  
incontrati, mentre il quarto indica che si e' incontrato  
un file senza nome.

FOUND (Nome del file scelto)

Ready.

L' ultimo messaggio prima del Ready sara' visualizzato quando il file scelto sara' trovato.

**\*\*NOTA\*\***

In modo programma invece questo blocco di messaggi non sara' visualizzato.

Quando il comando OPEN viene eseguito per una operazione di scrittura il computer visualizzera' il seguente messaggio:

PRESS PLAY & RECORD ON TAPE

Anche questo messaggio sara' seguito da un OK quando vengano premuti i tasti.

Il computer scrivera' il TAPE HEADER, poi si fermara' in attesa di dati.

Vediamo alcuni esempi di utilizzo del comando OPEN commentandoli:

OPEN 1

Viene aperto il file logico 1. Nessuna periferica e' specificata per cui come periferica sara' assunta la cassetta.

Non essendo inoltre specificato nessun indirizzo secondario il computer si prepara ad una operazione di indirizzo secondario 0 cioe' di lettura.

E dato che nemmeno il nome del file e' specificato,

verra' letto il primo file che incontra su cassetta.

OPEN 1,1

Come per il precedente, ma in questo caso e' specificata la periferica.

OPEN 1,1,0

Come il precedente salvo che in questo caso sono dichiarati tre parametri.

OPEN 1,1,0,"data"

Come sopra ma con una apertura per leggere il file di nome "data" sulla cassetta.

OPEN 3,1,2

Apri il file logico 3 per la cassetta 1. Scrive un nuovo file e un carattere End of Tape alla fine del file. Il file che si registra non ha nessun nome.

OPEN 3,1,2,"Paolo"

Come sopra ma questa volta con un file di nome PAOLO.

## CHIUSURA DI UN FILE

I comandi di apertura e di chiusura di un file sono fra se strettamente correlati.

Ricordiamoci che il comando CLOSE deve essere l' ultimo comando che si da nella sequenza logica della programmazione e che mentre se non si esegue l' apertura

di un file e' immediatamente tangibile che non si puo' accedere ad esso, il fatto di usare il comando CLOSE e' lasciato all' accortezza del programmatore, perche' il sistema non dara' alcuna segnalazione.  
Il formato del comando e':

CLOSE N

dove N e' l' unico parametro da specificare e deve corrispondere al numero di file logico precedentemente dichiarato nel comando OPEN.

Ricordiamo che quando e' stata eseguita un' operazione di chiusura su un file dopo la lettura non sono piu' consentiti accessi per la lettura sullo stesso file, per cui sara' necessario riaprirlo.

#### NOTA

Non e' indispensabile eseguire la chiusura di un file dopo la lettura, tuttavia non sara' cattiva pratica di programmazione prendere costante confidenza con questa operazione che in altri casi e' invece indispensabile.

E' da notare pero' che e' invece ASSOLUTAMENTE necessario eseguire la chiusura di un file dopo una operazione di scrittura.

Infatti i dati non vengono mai trasferiti direttamente dalla memoria centrale del computer al nastro, ma passano attraverso un BUFFER, cioe' attraverso una zona polmone che provvede a comunicare i dati stessi a gruppi di 192 Bytes.

Quando questo BUFFER e' stato riempito allora i dati passano al nastro. Ma solo allora, oppure quando si esegue un CLOSE per chiudere appunto il file al quale stiamo riferendoci.

#### NOTA

Tutte queste notizie sono approfondite sia dalpunto di

vista SOFTWARE che HARDWARE che come tabelle ed indirizzi di salto nel citato volume LE PERIFERICHE COMMODORE.

Per questo motivo accade quasi sempre che se al termine di un' operazione di scrittura non si esegue la chiusura un po' di bytes che quasi sicuramente sono sul BUFFER vengono persi.

Inoltre quando si esegue una chiusura di un file nastro dopo un' operazione di scrittura, verra' anche inviato un carattere di fine file o END OF FILE (EOF) che verra' quindi scritto sul nastro stesso.

Al sistema necessita questo carattere di separazione fra un file e il successivo. Senza di questo infatti il computer potrebbe, successivamente in fase di lettura, continuare a leggere i dati del file successivo sul quale magari siamo andati a scrivere sopra.

#### **\*\*NOTA\*\***

E' importante notare infatti che mentre in fase di input dati siamo certi della quantita' di dati da scrivere, altrettanto non avviene in fase di rilettura.

Quando si chiude un file preventivamente con un INDIRIZZO SECONDARIO 2, su cassetta verra' scritto un fine nastro END OF TAPE (EOT) alla fine del file stesso.

In questo caso il computer non andra' ad eseguire una ricerca di altri files.

#### **COME ACCEDERE AI DATA FILES**

Dopo aver aperto un file con un comando OPEN si puo' accedere a questo file sia per leggerci che per scriverci fino a quando il file stesso non sia chiuso con un comando CLOSE.

Ricordiamoci ancora una volta che sia che si scriva sia che si legga, queste operazioni devono essere fatte in modo SEQUENZIALE.

Cioe' il primo record scritto o letto sara' sempre il primo record del FILE, per cui se si vuole leggere il decimo record di un file sara' necessario prima leggere i primi nove.

Nessun tasto della cassetta deve essere premuto prima che l' apposito messaggio non sia apparso sullo schermo.

E' bene ricordare ancora una volta che la posizione esatta della cassetta e' sotto la responsabilita' dell' operatore. Infatti l' unita' a cassette iniziera' a scrivere immediatamente i dati non appena saranno premuti gli appositi tasti senza controllare ne che sul nastro sia scritto qualcosa ne che il nastro sia correttamente posizionato sull' inizio della superfice magnetica.

Per scrivere data su cassetta e' necessario usare il comando PRINT# che ha il seguente formato:

PRINT#f,data

dove f e' il numero di file logico che e' stato assegnato con un comando OPEN e che sara' riutilizzato con il comando CLOSE. Puo' avere un valore fra 1 e 255.

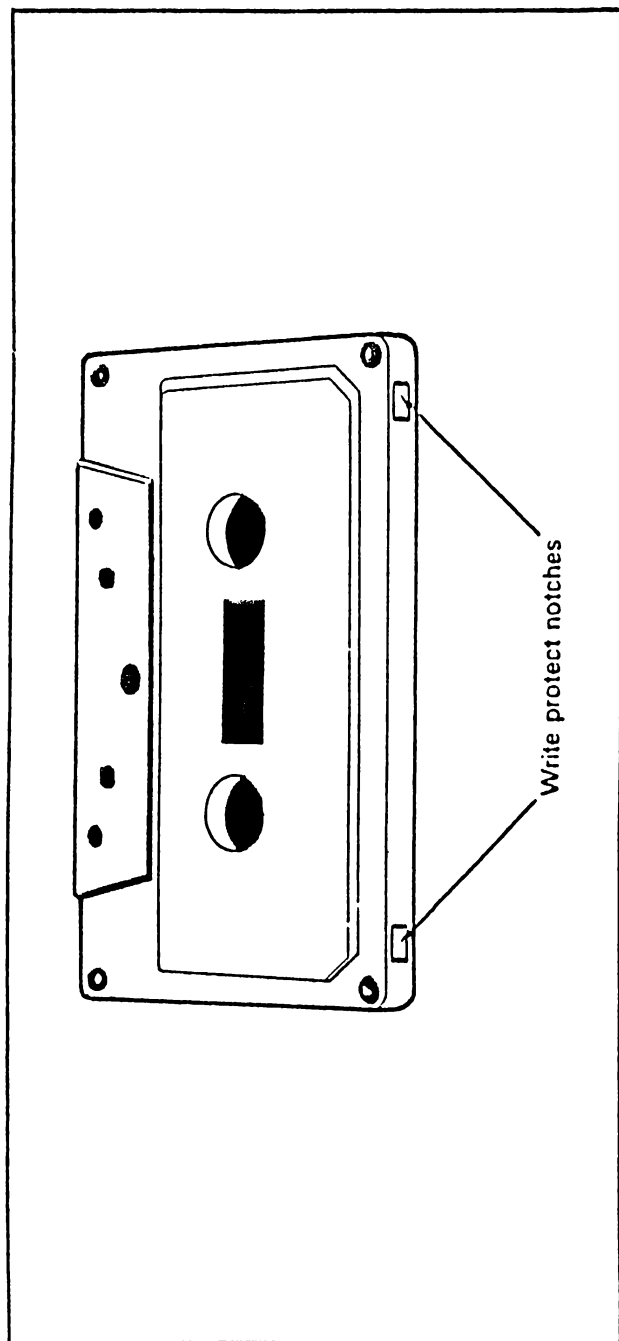
Data saranno invece i dati da caricare sul file.

Il comando PRINT #non puo' essere scritto nella forma abbreviata ?#, ma deve essere digitato per intero.

Il comando PRINT #trasferisce i dati dalla memoria centrale del computer al buffer di cassetta. Quando il buffer e' stato completamente caricato nei suoi 191 Bytes di capacita' i dati sono scaricati su nastro appunto in BLOCCHI di 191 caratteri per volta.

Con il comando appena visto e' possibile scrivere sia dati numerici che alfanumerici o stringhe.





- Linguette da togliere per proteggere da scrittura i FILES su cassetta.

## APPROFONDIMENTO HARD/SFTWARE

Tutti i computers COMMODORE, comprese le serie professionali e con la sola esclusione attuale del CBM64 SX ( quello portatile) hanno la possibilita' di connettere una singola unita' a cassette esterna che puo' essere utilizzata per immagazzinare programmi e dati. L' unita' e' connessa al computer per mezzo delle linee:

WRITE

READ

MOTOR

SENSE

due linee di alimentazione composte da una terra (GND) e un +5 Volts.

La cassetta e' controllata nelle sue linee di I/O per mezzo di due integrati di tipo VIA.

Le linee di alimentazione del motore sono connesse agli integrati di interfaccia attraverso 3 piloti transistor utilizzati per elevare la tensione ed il voltaggio e che consentono quindi che il motore sia alimentato e guidato direttamente dal computer.

L' uscita al motore e' un +9 Volts non regolata a una potenza di 500Ma.

Il controllo della linea di Input, la linea PA6 sul VIA 1, e' connessa ad un' interruttore sull' unita' a cassette che controlla quando sia il tasto PLAY, REWIND o FAST FORWARD sono stati premuti.

Purtroppo pero' la connessione dell' interruttore viene controllata solo durante le fasi di lettura e scrittura per vedere se il tasto PLAY e' premuto .

Per questo motivo se il tasto di REWIND e di FAST FORWARD vengono premuti accidentalmente invece del tasto PLAY, il

sistema non e' in grado di comprendere la differenza ed agisce come se il tasto PLAY fosse premuto. Per lo stesso motivo, durante l' esecuzione di una routine di registrazione il tasto RECORD dovrebbe essere premuto prima del tasto PLAY perche' la fase di registrazione inizia nello stesso momento in cui il tasto PLAY viene premuto senza controllare che sia stato premuto anche il tasto di RECORD.

La linea di lettura "READ LINE" e' connessa al CIA numero 1 tramite l' integrato VIA 2, e la linea di scrittura "WRITE LINE" alla linea PB3 del VIA 2.

Durante un' operazione di lettura il Sistema Operativo utilizza l' INTERRUPT FLAG del CIA numero 1 per controllare il passaggio di dati sulla linea di lettura cassetta.

Il funzionamento delle linee di lettura scrittura e' controllato interamente dal S.O. e la sola operazione Hardware richiesta e' l' amplificazione del segnale.

## OPERAZIONI SU CASSETTA

Nel normale utilizzo all' unita' a cassetta e' assegnato un numero di periferica di Input/Output.

La cassetta e' la periferica numero 1, il numero della periferica utilizzata in quel momento e' immagazzinata nella locazione decimale 186 per il VIC 20 e per il CBM64.

Il numero di periferica, il numero di file logico e l' indirizzo secondario sono utilizzati quando si salvano o si ricercano file di dati da cassetta.

Il numero di file logico puo' essere uno qualsiasi purché compreso nell' intervallo da 1 a 255 ed e' utilizzato per consentire la manipolazione di Files multipli sulla stessa periferica.

Tutto questo e' di poco utilizzo per l' unita' a

cassette, perche' non si possono manipolare contemporaneamente piu' files.

Lavorando normalmente e' usuale avere lo stesso numero sia per il file logico che per la periferica. Il numero di file logico del file sul quale si sta lavorando e' immagazzinato nella locazione decimale 184.

L'indirizzo secondario e' importante perche' determina il modo di operare della cassetta. L'indirizzo secondario e' immagazzinato nella locazione di memoria 185 il cui valore normale dovrebbe essere 0.

Se l'indirizzo secondario e' 0 allora il nastro e' aperto per un' operazione di lettura. Se e' messo a 1 allora il nastro e' aperto per un' operazione di scrittura mentre se il contenuto della locazione decimale 185 e' 2 sul nastro verra' eseguita un' operazione di scrittura con l' immissione di un carattere di END OF TAPE quando il file viene chiuso.

Il Sistema Operativo del computer e' configurato in modo tale da consentire che due diversi tipi di file possano essere immagazzinati su cassetta:

Files programmi

Files data

L' utilizzo di questi due nomi e' pero' una convenzione poiche' un programma puo' essere immagazzinato come file di dati mentre dei dati possono essere immagazzinati come file programma.

La differenza fra questi due tipi di files non e' nella loro applicazione ma nel sistema in cui i contenuti della memoria del computer sono registrati.

Infatti invece di FILES PROGRAMMI e FILES DATA si dovrebbe parlare piuttosto di FILES BINARI e FILES ASCII. Un file binario e' normalmente utilizzato per immagazzinare programmi perche' questo tipo di files viene creato e gestito dal sistema operativo che immagazzina i contenuti di memoria presenti e compresi

fra una locazione o un' indirizzo di inizio ed un' indirizzo di fine.

## FILES BINARI

E' chiamato file binario perche' immagazzina su nastro il valore di ogni locazione di memoria compresa nell' intervallo richiesto o desiderato.

I comandi Basic sono immagazzinati in memoria utilizzando i TOKENS.

L' uso dei TOKENS consente che i comandi non siano quindi immagazzinati nello stesso sistema in cui sono listati cioe' visualizzati oppure immessi tramite la tastiera.

I comandi sono invece immagazzinati in memoria in una maniera che potremmo definire parzialmente codificata.

Essendo parzialmente codificata un file binario e' un sistema veloce e molto efficiente di immagazzinare programmi.

Inoltre i files binari sono ESSENZIALI, questo nel senso che si devono obbligatoriamente usare, quando si caricano o si salvano programmi in codice macchina.

L' indirizzo di inizio dal quale un file di tipo binario deve essere salvato e' immagazzinato sia per il VIC 20 che per il CBM 64 nelle locazioni di memoria 172 e 173.

Queste locazioni di memoria sono utilizzate dalla routine di SAVE che conterra' l' indirizzo di inizio della BASIC TEXT AREA diversa pero' da macchina a macchina.

I contenuti delle locazioni di memoria 172 e 173 possono essere cambiati per consentire che la routine di SAVE punti ad una qualsiasi scelta locazione di memoria.

L' indirizzo di fine dell' area di memoria che deve essere salvata su nastro e' immagazzinata nelle locazioni 174 e 175.

Normalmente quando si salva un programma Basic queste due locazioni contengono l' indirizzo in cui nel programma si trovano due Bytes a ZERO che corrispondono al punto di

fine o di termine del programma Basic stesso.

Anche l' indirizzo di fine puo' essere cambiato per far si che la routine di SAVE punti ad un' altro qualsiasi desiderato indirizzo.

Per cambiare entrambi questi indirizzi non possiamo utilizzare la normale routine di SAVE poiche' questa automaticamente inizializza queste locazioni, cioe' punta o indirizza automaticamente a queste locazioni.

Dovremo quindi scrivere una piccola routine in codice macchina per modificare i valori presenti in detti indirizzi.

In mancanza di modifiche il comando SAVE scrivera' un file binario da quegli indirizzi ed un comando LOAD li leggerà come file binario.

## FILES ASCII

I files ASCII sono normalmente utilizzati per immagazzinare dati, (ma possono essere utilizzati per immagazzinare anche programmi).

Il formato dei dati e' lo stesso che viene visualizzato sullo schermo o immesso tramite tastiera.

I files ASCII sono creati o letti esclusivamente tramite istruzioni contenute entro un programma Basic.

Un file binario e' creato o letto molto spesso per mezzo di istruzioni date in modo diretto, sebbene i comandi LOAD e SAVE possano essere utilizzati entro un programma.

Un file ASCII deve per prima cosa essere aperto con un comando OPEN che specifichi il file logico, il numero di periferica, l' indirizzo secondario ed il nome del file.

Il Sistema Operativo interpreta questi parametri, cioe' quelli contenuti nel comando OPEN, e consente all' utente di leggere o scrivere il file su una data periferica.

Mentre un file binario e' quindi caricato come una serie di successive locazioni di memoria, un file ASCII e' caricato come una serie di variabili stringa.

Per questo motivo il file ASCII dovrebbe essere manipolato immagazzinando pochi Bytes per volta e

fermando e facendo quindi ripartire il nastro.

Il Sistema Operativo del computer supera questo problema perche' dispone di un BUFFER di nastro di 192 Bytes entro il quale tutti dati che devono essere scritti su nastro o da questi letti vengono caricati.

Solo quando questo Buffer e' pieno il motore del nastro si avvia.

I dati sono quindi immagazzinati su nastro in blocchi di 192 Bytes e cosi' il motore si attiva e si spegne tra i blocchi in un intervallo di due secondi cosa che consente al motore stesso un' accurato movimento di accelerazione e di decelerazione.

L' inizio del buffer contenente i 192 caratteri e' posto all' indirizzo 828. Con questo intendiamo naturalmente dire che l' area del Buffer inizia da qui.

Il puntatore all' inizio del Buffer e' dato dal contenuto dei puntatori 178 e 179.

Il numero dei caratteri o il numero di Bytes presenti nel buffer e' immagazzinato nell' indirizzo di memoria 166.

Questa locazione di memoria puo' essere utilizzata dal programmatore per controllare l' ammontare di spazio lasciato in un file di dati.

Sia che il file da immagazzinare sia di tipo binario o ASCII il metodo di registrazione e' lo stesso.

Si tratta infatti di un metodo di codifica messo a punto dalla COMMODORE per consentire la massima sicurezza di registrazione e quindi poi di rilettura.

Ogni Byte di dati o di programma e' codificato dal Sistema Operativo del computer utilizzando impulsi di 3 distinte audiofrequenze.

Questi sono impulsi di lunga durata con una frequenza di impulsi di 1488 Hz, impulsi di media durata con una frequenza di 1953 Hz ed impulsi di corta durata con una frequenza di 2480 Hz.

Tutti questi impulsi sono del tipo ad ONDA QUADRA con un' intervallo costante ( cioe' di 1:1).

Il Sistema Operativo richiede circa 9 millisecondi per registrare un Byte di dati che consiste di 8 bits, una parola di segnale, ed un bit di parita'.

I bits di dati possono essere composti sia da 1 che da 0 e sono codificati da una sequenza di impulsi medi e corti.

Un UNO e' composto da un ciclo di impulsi di media lunghezza seguito da un ciclo d' impulsi di corta lunghezza, mentre uno ZERO e' composto da un ciclo d' impulso di corta lunghezza seguito da un ciclo d' impulso di lunghezza media.

Ogni bit consiste quindi di due cicli d' impulsi ad onda quadra uno corto ed uno medio per una durata totale di 864 mcs.

Il BIT di parita' o PARITY BIT e' richiesto per il controllo d' errore ed e' codificato come 8 bits di dati utilizzando un impulso lungo ed uno corto.

Il suo stato e' determinato dal contenuto degli 8 data bits. Il WORD MARKER, cioe' il segnalatore di parola, separa ogni Byte di dati e segnala anche al Sistema Operativo l' inizio di ogni Byte.

Il WORD MARKER e' codificato come un ciclo d' impulsi lunghi seguito da un ciclo d' impulsi medi.

## **\*\*NOTA\*\***

Poiche' un BYTE di dati e' registrato in appena 8.96 millisecondi, un blocco di 192 Bytes di dati di un file ASCII dovrebbe essere registrato in poco piu' di 1.7 secondi.

Tuttavia ad un controllo di temporizzazione risulta che un Byte di dati richiede circa 5.7 secondi.

Ci sono due cause per questa discrepanza nella temporizzazione.

Prima di tutto per ridurre la possibilita' di errori audio i dati sono registrati due volte.

Secondo fra due record di 192 bytes viene immesso un INTER RECORD GAP della lunghezza di circa 2 secondi.



## CONTROLLO DI ERRORE

L'uso massiccio delle tecniche di controllo errore (ERROR CHECKING) e' una delle ragioni per le quali il sistema a nastri implementato sui computer COMMODORE e' di gran lunga migliore di quello disponibile in molti altri personal computer.

Ci sono due livelli di controllo d' errore.

Il primo divide i dati in blocchi di 8 bytes e successivamente inserisce un nono Byte chiamato CHECKSUM DIGIT.

Il CHECKSUM e' ottenuto aggiungendo all' ottavo Byte un nuovo Byte.

Il CHECKSUM e' il Byte meno significativi di risultato.

Il secondo livello di controllo errore consiste nel registrare ogni blocco di dati due volte.

Questo consente di rilevare gli errori controllando il digit che deve essere corretto durante la seconda lettura del blocco di 192 Bytes di dati.

Registrando i dati due volte una verifica e' possibile confrontando il contenuto dei due blocchi.

Cio' consentira' di rilevare quei piccoli errori che non sono stati controllati dal CHECKSUM.

L' utilizzo di sequenze di impulsi invece di due differenti frequenze come normalmente avviene nelle registrazioni, ha un grande vantaggio perche' consente al Sistema Operativo di compensare facilmente la variazione o le variazioni nella velocita' di registrazione.

I computers COMMODORE utilizzano infatti il sistema SOFTWARE per controllare la corretta registrazione dei dati.

Prima di iniziare la registrazione dei dati o dei programmi su nastro viene scritta una testata di 10 secondi.

Questa testata ha due funzioni. Prima di tutto consente che il motore del nastro possa incominciare a scorrere ad una velocita' corretta.

Secondariamente la sequenza di impulsi corti scritta nella testata e' utilizzata per sincronizzare la routine di lettura temporizzata in modo tale che il nastro sia correttamente temporizzato.

In questo modo il Sistema Operativo puo' quindi produrre un fattore di correzione che consente un largo raggio di variazioni nella velocita' del nastro senza che ne abbia a risentire la fase di lettura.

La temporizzazione di sistema utilizzata per consentire sia le operazioni di scrittura che di lettura e' molto accurata perche' e' basata su un sistema di CLOCK e sui TIMER 1 e 2 (cioe' sui TEMPORIZZATORI 1 e 2) del VIA n.2.

Gli INTERRECORD GAPS sono utilizzati solamente con i Files ASCII e la loro funzione e' di permettere che il motore dell' unita' a nastri possa decelerare prima che sia fermato ed accelerare alla corretta velocita' prima che riparta.

In questo modo viene consentita una migliore lettura o scrittura di dati.

Ogni INTERRECORD GAP ha una durata di circa 2 secondi ed e' registrato come una sequenza di impulsi corti allo stesso modo dei 10 secondi della testata.

E' presente anche un GAP , cioe' un' interruzione fra i blocchi.

Quando il primo blocco di 192 Bytes e' stato registrato, questi e' seguito da 50 cicli di impulsi corti e poi inizia la registrazione del secondo blocco di 192 Bytes. Sia che stiamo registrando un file ASCII che un File di tipo binario, il primo record scritto su nastro dopo la testata di 10 secondi e' un file di 192 caratteri chiamato FILE HEADER BLOCK.

Il FILE HEADER contiene il nome del file, l' indirizzo di partenza e l' indirizzo di fine.

In un file ASCII questi indirizzi sono l' inizio e la fine del BUFFER di nastro, mentre in un file di tipo binario questi indirizzi contengono l' indicazione dell' area di memoria nel quale il programma deve essere immagazzinato.

Il nome del file puo' essere lungo al massimo 128 Bytes. La lunghezza del nome del file e' immagazzinata nell' indirizzo di memoria decimale 183 e quando viene eseguita un' operazione di lettura, questa lunghezza verra' confrontata con il nome del file del comando LOAD. Se il nome del file trovato e' identico allora il Sistema Operativo consentira' la lettura del file. Se e' invece diverso proseguira' nella ricerca.

Durante un' operazione di lettura o scrittura il nome del file e' immagazzinato in un blocco di memoria il cui indirizzo di partenza e' dato dal contenuto delle locazioni di indirizzo 187 e 188. Una volta compiuta questa operazione le due locazioni di memoria precedenti sono azzerate per puntare ad una locazione indicata dal Sistema Operativo.

L' indirizzo di partenza e' normalmente fissato all' inizio della memoria utente che pero' varia da macchina a macchina.

Tuttavia questa puo' essere cambiato per puntare ad un' altra qualsiasi locazione.

Questo sistema e' impiegato quando si registra un programma in codice macchina utilizzando il monitor e quando si eseguono delle protezioni sui programmi.

## CAPITOLO DICIASSETTESIMO

### Messaggi di errore e rimedi possibili

Diamo qui di seguito un' elenco dei messaggi di errore insieme ad una serie di cause che possono generarli e dei possibili rimedi.

#### BAD SUBSCRIPT

E' stato eseguito un tentativo di riferirsi ad un elemento di una matrice che e' fuori della dimensione della matrice stessa.

Cio' puo' essere successo o perche' e' stata dimensionata ( con il comando DIM) una matrice piu' piccola di quella che poi il programma avrebbe utilizzato oppure si e' tentato di utilizzare un elemento di indice maggiore di 10 senza aver eseguito un dimensionamento.

Si ricorda infatti che le matrici non dimensionate assumono automaticamente indice 10 ( o meglio da 0 a 9).

#### BAD DATA

E' stato tentato di immettere una stringa quando il programma attendeva dei dati numerici.

Correggere il dato in INPUT oppure cambiare il programma per far in modo che accetti la stringa.

#### BAD DISK

Errore del Sistema Operativo Disco. Puo' essere dovuto sia alla mancanza di dischetto nel Drive, al tentativo di scrivere su un disco con la tacca protetta o anche ad un

difetto sulla superficie magnetica del dischetto stesso.  
Controllare che il dischetto sia presente e correttamente inserito.  
Togliere la protezione dalla finestrella se e' presente.

## CAN'T CONTINUE

E' stato dato un comando di CONT, ma l' esecuzione del programma non puo' riprendere. I motivi possono essere diversi: il programma e' stato cambiato, e' stato aggiunto qualcosa o e' stato eseguito un CLEAR in modo diretto.

Oppure l' esecuzione del programma e' stata interrotta da un errore.

Infatti l' esecuzione non puo' riprendere, nel senso di continuare dallo stesso punto, dopo un messaggio d' errore.

Eseguire la correzione dell' errore.

La strada migliore e' quella di far partire il programma di nuovo e da capo con un RUN, tuttavia si puo' tentare di far riprendere l' esecuzione di un programma da un punto determinato tramite un' istruzione di GOTO diretta.

## DEVICE NOT PRESENT

Durante un comando di INPUT o di OUTPUT il computer si accorge che la periferica alla quale il comando stesso e' indirizzato non risponde al segnale.

Controllare per prima cosa che la periferica sia correttamente connessa al sistema e che sia stata accesa. Controllare anche che sia corretto il comando (OPEN o altri).

Controllare l' indirizzo.

## DIVISION BY ZERO

E' stato tentato di eseguire una divisione con un divisore = 0 cosa che non e' consentita.

Controllare il valore delle variabili ( o anche di costanti) nel numero di linea indicata.

Eseguire i necessari cambiamenti nel programma in modo tale che non si trovi a dover eseguire un tipo di operazione di questo genere.

## EXTRA IGNORED

In risposta ad una richiesta di INPUT, cioe' di ingresso dati, e' stato tentato di immettere piu' dati del necessario. E quindi gli altri sono ignorati.

## FILE NOT FOUND

Il nome del file dato con un comando di LOAD o OPEN non e' stato trovato su quella particolare periferica.

Controllare che sull' unita' a dischi o sulla cassetta ci siano rispettivamente il floppy o il nastro giusto.

Controllare che sia stato digitato correttamente il nome, controllandone cioe' sia le lettere che gli eventuali spazi.

Ricordiamo che mentre sul disco la risposta e' immediata, il nastro verra' fatto scorrere fino a quando non termina o comunque non rileva un segnale di END-OF-TAPE.

## FILE NOT OPEN

E' stato tentato di eseguire un accesso ad un file che non era stato preventivamente aperto per mezzo di un comando OPEN.

L' unico rimedio suggerito e' quindi di aprire il File.

## FILE OPEN

Si e' tentato di aprire un file che era gia' stato aperto con un comando OPEN oppure e' stato tentato di aprire un file con il numero di un file gia' aperto.

Controllare il numero di file logico ( il primo parametro nel comando OPEN) per assicurarsi che sia utilizzato un diverso numero per ogni file.

Inserire un comando CLOSE se si desidera riaprire lo stesso file per una diversa operazione di I/O.

## FILE EXIST

Il nome del file sorgente, cioe' del file che deve essere copiato, e' gia' presente sul dischetto sul quale deve essere copiato.

Cancellare il file sul dischetto copia prima di eseguire il COPY oppure utilizzare un' altro dischetto.

## FORMULA TOO COMPLEX

Questo non e' un errore vero e proprio ma indica che un' espressione , nel programma stesso, e' troppo complessa o intricata per essere manipolata dal Basic.

Si consiglia di spezzare l' espressione stessa in due o piu' parti e di far girare il programma, cosi' corretto, dall' inizio.

## ILLEGAL DIRECT

E' stato dato un comando in modo immediato che invece puo' essere dato solo in modo programma.

I seguenti comandi devono essere dati solo in modo programma:

DATA  
DEF FN  
GET  
GET #  
INPUT  
INPUT #

## ILLEGAL QUANTITY

Si ha questo tipo di errore quando si esegue o si tenta di eseguire una funzione con valori dati ai parametri fuori degli intervalli previsti per quella funzione.

E' necessario rivedere i parametri possibili per ciascuna funzione.

Questo errore inoltre puo' essere visualizzato se si tenta di far eseguire al programma una funzione USR prima che siano immagazzinati gli indirizzi delle subroutines nelle locazioni di memoria 1 e 2.

Poiche' il valore del parametro puo' derivare anche da un calcolo, eseguire un controllo nel programma per vedere che il valore stesso sia sempre nell' intervallo consentito.

## NEXT WITHOUT FOR

E' stato incontrato un NEXT senza che sia preceduto dal relativo FOR.

Puo' accadere cipe, come vediamo negli esempi sotto, sia che venga incontrato un NEXT senza il FOR sia che ci si riferisca con il NEXT ad una variabile diversa da quella indicata dal FOR.

FOR I=1 TO 10: NEXT:NEXT

oppure



FOR I=1 TO 10:NEXT J

Riesaminare quindi con piu' cura il programma.

#### NOT INPUT FILE

Si e' tentato di eseguire una lettura da nastro su di un file che era stato aperto solo per scriverci.

Controllare i parametri dei comandi ricordando che la lettura richiede che il terzo parametro del comando OPEN sia 0.

#### NOT OUTPUT FILE

Si e' tentato di eseguire una scrittura su un file presente su cassetta che era stato aperto per la lettura.

Anche in questo caso come nel precedente sara' necessario controllare la correttezza dei parametri.

Ricordiamo che il terzo parametro del comando OPEN deve essere = a 1 per eseguire un' operazione di lettura ( o = a 2 se si desidera un END OF TAPE ).

#### OUT OF DATA

E' stato eseguito un READ ma tutti i DATA presenti nel programma sono gia' stati letti.

Ad ogni variabile in un comando READ deve corrispondere un elemento di DATA.

Quindi o il programma tenta di leggere piu' DATA di quelli che sono presenti oppure i DATA non sono sufficienti.

Il possibile rimedio consiste nell' aggiungere altri elementi ai DATA oppure di restringere il campo di

lettura del comando READ.

Se si desidera rileggere DATA precedentemente utilizzati ricordarsi di immettere un comando RESTORE.

Anche il RETURN (ritorno carrello) sul messaggio READY. presente sul video causera' questo tipo di errore perche' verra' interpretato come un READ Y.

## OUT OF MEMORY

Questo messaggio puo' apparire non solo mentre sta girando, ma anche mentre si sta inserendo o listando un programma.

Dice in pratica che tutta la memoria a disposizione dell' utente e' stata utilizzata.

Inoltre mentre gira il programma ricordiamo che la definizione o la creazione di variabili occupano altra memoria.

E' da notare anche che grossissime parti di memoria sono occupate da dimensionamenti di matrici anche in presenza di programmi cortissimi.

Questo messaggio puo' anche essere causato da un eccessivo uso di cicli di FOR...NEXT e/o di GOSUB che riempiono completamente l' area di memoria dello STACK.

In questo ultimo caso e' sufficiente eseguire un ?FRE(0). Se il numero di Bytes a disposizione e' ancora abbastanza ampio allora e' proprio il caso di ridurre il numero di FOR...NEXT o di GOSUB.

Nei casi precedenti e' bene rivedere il programma, cercando di ottimizzare gli spazi, controllando accuratamente l' area di dimensionamento delle matrici o spezzando ulteriormente il programma con la tecnica di utilizzo chiamata OVERLAY.

Per ultimo ricordiamo che una Subroutine che termina con un GOTO invece che con un RETURN puo' causare questo errore, sempre dovuto all' occupazione dell' area di STACK.

**\*\*NOTA\*\***

Questo problema si presenta spesso sul VIC 20 in configurazione base, cioe' senza espansione per le ridotte capacita' di memoria.

**OVERFLOW**

In questo caso un calcolo ha dato un risultato maggiore del numero piu' grande manipolabile dal BASIC. Il numero piu' grande consentito e' 1.70141184 E + 38. Eseguire un controllo nel programma. Benché sia difficile cadere in questo errore e' possibile correggerlo cambiando l'ordine di esecuzione dei calcoli.

**\*\*Non esiste un errore di UNDERFLOW ma un numero minore di 2.93873587 E-39 non viene distinto dallo ZERO.**

**REDIM'D ARRAY**

Il nome di una matrice appare in piu' di un comando DIM. In pratica si tenta di dimensionare piu' volte la stessa matrice. Oppure si assegna il valore ad una variabile e poi si tenta di dimensionarla:

```
10 A(5)=21
20 DIM A(10,10)
```

Dopo aver eseguito il RUN avremo un:

**?REDIM'D ARRAY ERROR**

E' necessario solo un po' di attenzione per prevenire questo tipo di errore ed eventualmente per correggerlo. Inmettere i comandi DIM vicino all'inizio del programma per abitudine in modo di averli rapidamente sotto

controllo.

Controllare per vedere che ogni comando DIM sia eseguito solo una volta.

Ricordare inoltre che un comando DIM non deve essere inserito in un ciclo di FOR...NEXT o in una subroutine dove possa essere eseguito piu' di una volta.

## REDO FROM START

Non e' un errore di programma, ma un messaggio diagnostico eseguito durante un' operazione di INPUT. Sta ad indicare che si e' tentato di rispondere ad una richiesta di INPUT con un tipo di dati errati, cioe' una stringa al posto di dati numerici o viceversa. Immettere il tipo di dati giusti richiesti dal programma.

## RETURN WIHOUT GOSUB

E' stato incontrato un comando di RETURN senza che sia stato prima eseguito un comando di GOSUB. Inserire un comando di GOSUB o togliere il RETURN. L' errore spesso e' causato da salti a subroutines non ben calcolati ed e' quindi necessario correggere la logica di funzionamento del programma.

Suggeriamo in fase di prova programmi di mettere degli END o STOP nelle subroutines.

## STRING TOO LONG

E' stato tentato di concatenare ( ricordiamo con l' operatore +), due o piu' stringhe per creare una nuova stringa piu' grande di 255 caratteri. Spezzare la stringa in 2 o piu' parti piu' corte.

Si consiglia inoltre di usare la funzione LEN per calcolare la lunghezza delle stringhe prima di procedere alla concatenazione per evitare questo errore.

## SYNTAX

Si può avere un errore di sintassi sia in modo immediato sia durante l'esecuzione di un programma.

E' l'errore più comune e può essere dovuto a numerose cause come:

- l'utilizzo di parole non riconosciute dal Basic
- la punteggiatura non corretta
- parole chiave scritte male
- uso di parentesi non corretto, ecc.

Se il SYNTAX viene segnalato durante l'esecuzione del programma, esaminare con cura la linea dove viene segnalato l'errore e correggerla.

Ricordiamo che questo tipo di errore in modo programma, viene segnalato al momento che il programma stesso è eseguito e non al momento della digitazione da tastiera.

## TYPE MISMATCH

Si è tentato di assegnare valori numerici ad una variabile stringa o viceversa.

Oppure ad una funzione che aveva per parametri dei numeri si è tentato di assegnare una stringa o viceversa.

A\$=5

? TYPE MISMATCH ERROR

oppure

A="CIAO"

? TYPE MISMATCH ERROR

UNDEF'D STATEMENT

Si e' tentato di eseguire un comando GOTO, GOSUB o THEN ad una linea di programma o a un comando che non esiste. Inserire il comando con il necessario numero di linea o far eseguire il salto ad un' altro punto del programma.

UNDEF'D FUNCTION

Si e' tentato di utilizzare una funzione definita dall' utente senza che sia stato usato preventivamente una istruzione DEF FN per definire la funzione stessa. Eseguire le opportune correzioni.

VERIFY ERROR

Questo errore e' conseguenza dell' esecuzione del comando VERIFY.

In questo caso il programma presente in memoria e' stato confrontato con quello su nastro o su disco e non e' stato trovato uguale.

## CAPITOLO DICIOTTESIMO

### IL LINGUAGGIO MACCHINA

Al centro funzionale e logico di ogni computer e' presente un microprocessore che e' un circuito integrato particolare e costituisce il CERVELLO stesso della macchina.

Il CBM64 non fa eccezione a questa regola ed e' dotato di un nuovo microprocessore il MOS 6510.

Ogni microprocessore ha un suo linguaggio di istruzioni chiamate istruzioni in linguaggio macchina.

Anzi, per essere piu' precisi, il linguaggio macchina o codice macchina e' il solo linguaggio che il CBM64 comprende, cioe' a dire il LINGUAGGIO NATIVO.

La prima domanda che viene da porsi e' quindi come faccia il CBM64 a comprendere un linguaggio come il BASIC che non e' il suo proprio.

Per rispondere a questa domanda dobbiamo per prima cosa vedere cosa accade all' interno del CBM64.

A parte il microprocessore e' residente un programma in linguaggio macchina che e' immagazzinato in memoria ROM cosi' che non possa essere cambiato, e cosa molto piu' importante, non si perde allo spegnimento della macchina come invece accade per i programmi scritti da voi ( e che non siano salvati su cassetta o disco).

Questo programma in linguaggio macchina e' chiamato SISTEMA OPERATIVO. Ed il vostro CBM64 sa cosa deve fare all' accensione perche' questo programma gira immediatamente ed automaticamente.

Il Sistema Operativo ha l' INCARICO di organizzare tutta la memoria della vostra macchina per le varie funzioni siano queste il controllo del tasto premuto, la visualizzazione di caratteri, l' utilizzo delle funzioni e dei comandi.

Infatti quando accendete la macchina il Sistema Operativo

esegue una serie di controlli e quindi si dichiara pronto ad operare (READY.)

Un particolare programma dell' intero Sistema Operativo, che dovrebbe essere visto come un insieme di procedure, e' il Basic interprete, cosi' chiamato perche' interpreta ogni comando che diamo alla macchina e se incontra un comando o una funzione che non comprende, cioe' che non riconosce, risponde con un :

?SYNTAX ERROR

A questo punto dovrebbero esservi abbastanza familiari i concetti dei comandi PEEK e POKE cioe' di quei comandi che vanno a leggere e scrivere in determinate locazioni di memoria.

In altre parole avete gia', pur utilizzando comandi e funzioni Basic, scritto e letto DIRETTAMENTE nell' interno del vostro computer sia come memoria che come indirizzi e comandi.

I programmi in linguaggio macchina non sono altro che una serie di istruzioni ad ognuna delle quali puo' o non puo' essere vicino un operando.

Non e' particolarmente difficile utilizzare questo nuovo tipo di linguaggio in modo particolare se si dispone di un programma ASSEMBLER che ne facilita la messa a punto, tuttavia esula dai limiti di questo manuale una trattazione completa di questo linguaggio e del suo utilizzo.

Poiche' pero' qualcuno potrebbe avere gia' conoscenze di questo linguaggio che e' notevolmente piu' potente del Basic e che permettendo di utilizzare e conoscere a fondo il computer, riserva delle soddisfazioni che ripagano abbondantemente la fatica di studiarlo, riportiamo in appendice:



LE ISTRUZIONI DELL' ASSEMBLER

LE ROUTINES KERNAL COMMENTATE

LE MAPPE DI MEMORIA.

Tuttavia non volendo lasciare l' utente ed il lettore completamente all' oscuro di termini che puo' frequentemente udire o leggere riporteremo alcune brevissime annotazioni tratte dal manuale di ASSEMBLER -ed. EVM- in corso di preparazione.

Riprendendo il discorso iniziato in precedenza possiamo affermare che un microelaboratore e' un sistema elettronico in grado di ricevere dall' esterno segnali elettrici, immagazzinarli, elaborarli secondo un certo programma, prendere delle decisioni, naturalmente sempre in base al programma o agli ordini dati ed alle sue intrinseche capacita' operative e quindi emettere dei segnali elettrici utilizzabili all' esterno.

Essenzialmente un micro e' composto da 5 grandi blocchi funzionali:

CPU - Central Processing Unit

cioe' l' unita' centrale di elaborazione, nel nostro caso costituita come detto dal processore 6510.

RAM - Random Access Memory

le memorie di lettura e scrittura.

ROM - Read Only Memory

cioe' memorie a sola lettura

I/O - Input/Output

cioe' tutti i dispositivi di ingresso e uscita e che quindi permettono di colloquiare con il mondo esterno sia alla CPU che al computer propriamente detto.

## BUS

Insieme di linee sulle quali si muovono i segnali elettrici e che collegano un blocco funzionale all'altro.

Abbiamo detto che all' interno di un microelaboratore circolano dei segnali elettrici che passano da un blocco all' altro attraverso il BUS dei dati.

Questi particolari segnali elettrici si chiamano BIT ed altro non sono che livelli alti o bassi di tensione.

Con la parola BIT che e' la contrazione delle due parole inglesi BINARY DIGIT si intende l' unita' elementare di informazione nell' elettronica ( appunto DIGITALE ) cioe' uno dei possibili stati logici o livelli logici ZERO e/o UNO.

## NOTA

Ricordarsi che lo Ø e' usato in una forma di scrittura particolare cioe' con la sbarra in diagonale per distinguerlo dalla lettera O, ma che spesso non e' evidenziato per ragioni tipografiche.

Tornando al nostro discorso dunque il BIT puo' avere due e due soli stati.

Se si considera il classico esempio di una lampadina potremo avere che quest' ultima e' in una delle due condizioni:

Accesa cioe' ON

Spenta cioe' OFF

e di conseguenza con un solo BIT possiamo avere uno dei due stati.

Se avessimo due lampadine ovverosia due BIT potremo avere invece quattro combinazioni diverse:

0 0  
1 0  
0 1  
1 1

Da questo se ne ricava che la formula generale delle combinazioni che possono avere piu' BITS usati contemporaneamente e':

$2 \text{ elevato alla } n$

dove  $n$  e' il numero di BITS trattati.

Da questo ne consegue che un microprocessore ad 8 BITS, cioe' che puo' trattare fino ad 8 BITS contemporaneamente, puo' ricevere informazioni in 256 combinazioni diverse.

Nel caso del 6510 abbiamo un microprocessore che tratta informazioni a 8 BITS, ma che puo' utilizzare 16 BITS di indirizzi quindi pari a 65536 cioe' 2 alla 16ma.

Abbiamo pero' trovato altri termini per la rappresentazione dei dati:

Nibble cioe' 4 BITS

BYTE cioe' 8 BITS

## I REGISTRI DEL 6510

Sempre senza la pretesa di approfondire vediamo quali

sono i registri principali del nostro microprocessore.

## L' ACCUMULATORE

Come dice il nome stesso e' un' area di immagazzinamento. E' il piu' importante registro del microprocessore de e' controllabile tramite una nutrita serie di comandi in codice macchina.

Questi comandi o istruzioni consentono di copiare il contenuto di una locazione di memoria nell' Accumulatore, di eseguire l' operazione inversa, di modificarne i contenuti e di accdervi direttamete senza effetto sulle altre locazioni di memoria.

## I REGISTRI INDICE X , Y

Altre due locazioni di memoria che potremo definirre complementari o di aiuto alle funzioni dell' Accumulatore e che comunque consentono una notevole agilita' operativa.

Le operazioni consentite su questi registri sono simili a quelle relative all' Accumulatore ma altre devono essere svolte SOLO da uno o tutti e due i registri.

## PROGRAM COUNTER

Un programma in codice macchina consiste essenzialmente in una serie di istruzioni come:

Prendi il contenuto della locazione di memoria X

Sommalo al contenuto della locazione di memoria Y

Metti il risultato nella locazione di memoria Z.

Questo, come del resto altri programmi piu' complessi, puo' essere eseguito dalla CPU.

E' evidente pero' che la CPU cioe' il processore non puo' iniziare ad eseguire istruzioni partendo da un punto qualsiasi della memoria e seguendo un ordine casuale, ma deve iniziare partendo da un punto ben preciso che rappresenta l' inizio del programma.

Notare che questo avviene anche nell' esecuzione di un programma scritto in Basic.

Nella CPU si trova un registro chiamato PROGRAM COUNTER o contatore di programma che contiene la locazione di memoria dalla quale verra' prelevata la prossima istruzione del programma che la CPU stessa deve eseguire. In linguaggio tecnico questo registro viene chiamato anche POINTER cioe' puntatore perche' punta, cioe' guarda alla prossima istruzione da eseguire e viene abbreviato come le lettere PC.

Dopo aver eseguito un' istruzione del programma il PC si posiziona automaticamente all' indirizzo dell' istruzione da eseguire immediatamente dopo e quindi non si richiede nessun intervento da parte dell' utente il quale tuttavia potra' eventualmente modificarne il contenuto per far SALTARE il programma non all' istruzione immediatamente seguente ma da altra parte.

Come si vede nelle tavole riportate in fondo, le istruzioni del 6510 non sono tutte di una sola locazione di memoria, ma possono interessare da 1 a 3 locazioni. In questo caso il PC si incrementara' di conseguenza.

Abbiamo insistito in modo particolare su questo registro di 16 BITS per le analogie che presenta con i relativi registri di pagina ZERO del Basic che come abbiamo detto hanno funzioni similari.

LO STATUS REGISTER

Anche questo e' un registro importantissimo nell'architettura della CPU.  
Ognuno degli otto BITS che lo compongono e che in questo caso sono denominati FLAGS ha una funzione particolare ed e' programmabile con particolari comandi.

## NOTA FINALE

Come gia' ricordato per approfondire l' argomento e' necessario un apposito manuale come quelli citati.  
Tuttavia una prima esperienza potrebbe essere fatta usando un MONITOR (ottimo il SUPERMON purché contenga le istruzioni in italiano) che consentendo di implementare piccole routines in codice macchina può evidenziare le potenti capacità di questo modo di operare.

## LA MEMORIA DEL CBM64

Il CBM64 ha 64 K Bytes di memoria RAM cosa che in parte da origine al nome della macchina.

Al suo interno sono presenti anche 20 K Bytes di memoria ROM che contengono il Basic, il Sistema Operativo ed il set di caratteri Standard.

La prima domanda che ci si pone e' quindi come sia possibile che un microprocessore a 8 bit che e' quindi capace di indirizzare un massimo di 64 K Bytes possa gestire tanta memoria.

Il segreto e' nello stesso integrato 6510.

Nell' integrato e' presente una porta di ingresso/uscita (I/O PORT). Questa porta e' impiegata per controllare che le RAM, le ROM o gli organi di I/O vengano gestite in certe parti del sistema di memoria.

La porta e' usata anche per controllare la DATASETTE , cioè la cassetta del CBM64.

La porta I/O del 6510 e' di indirizzo 1 , mentre il

registro di direzione dati, sempre per questa porta, e' di indirizzo 0.

La porta e' controllata allo stesso modo delle altre porte di I/O del sistema, mentre il registro di direzione dati controllera' appunto che un certo bit possa essere in entrata o in uscita ed il trasferimento di dati passera' attraverso la porta stessa.

Le linee della porta di controllo sono riportate in tabella:

| NOME   | BIT      | DIREZIONE | DESCRIZIONE                                   |
|--------|----------|-----------|---|
| LORAM  | 0        | Uscita    | Controllo per RAM/ROM a \$A000-\$BFFF (Basic) |
| HIRAM  | 1        | "         | Controllo per RAM/ROM a \$E000-\$FFF (Kernal) |
| CHAREN | 2        | "         | Controllo per I/O/ROM a \$D000-\$DFFF         |
| 3      | "        |           | Linea di scrittura cassetta                   |
| 4      | Ingresso |           | Switch di cassetta                            |
| 5      | Uscita   |           | Controllo motore della cassetta               |

Il valore normale per il registro di direzione dati e' il seguente:

BIT 5 4 3 2 1 0

1 0 1 1 1 1

dove 1 e' un OUTPUT e 0 un INPUT

Cio' da un valore di 47 in decimale. Il CBM64 fissa automaticamente il registro a questo valore.

Le linee di controllo, in generale, consentono la funzione data nella loro descrizione vista nella tabella. Tuttavia una combinazione delle linee di controllo e' usata occasionalmente per particolari configurazioni di memoria.

LORAM (bit0) puo' essere generalmente vista come una linea di controllo che gestisce gli 8 K Bytes di ROM del Basic.

Normalmente questa linea e' alta (cioe' a 1) per le operazioni Basic. Se invece viene messa bassa (0) allora il Basic ROM scomparira' dalla mappa di memoria e sara' rimpiazzato da 8 K Bytes di RAM di indirizzo da \$A000 a \$BFFF.

HIRAM (bit1) puo' essere generalmente vista come una linea di controllo che gestisce gli 8 K Bytes di ROM delle routines Kernal.

Normalmente anche questa linea e' alta (cioe' a 1) per le operazioni Basic. Se invece viene messa bassa (0) allora le Rotines Kernal verranno accantonate (anche in questo caso si intende che non sono sotto controllo del microprocessore) dalla mappa di memoria e sara' rimpiazzato da 8 K Bytes di RAM di indirizzo da \$E000 a \$FFFF.

CHAREN (bit2) e' usata solo per gestire i 4 K Bytes del generatore di caratteri in ROM.

Dal punto di vista del 6510 la ROM dei caratteri occupa lo stesso spazio di indirizzo delle periferiche di I/O (\$D000-\$DFFF).

Quando la linea di CHAREN e' a 1( cioe' in modo normale), gli I/O periferiche compaiono nello spazio di indirizzo del microprocessore e la ROM dei caratteri non e' accessibile.

Quando invece il Bit di CHAREN e' messo a 0, accade esattamente il contrario.

Cioe' il microprocessore PUNTA al generatore dei caratteri e gli I/O sono esclusi.

Il microprocessore ha la necessita' di accedere alla ROM caratteri SOLO quando deve trasferirne il contenuto in



RAM ed ecco perciò spiegato il mistero della sezione relativa ai caratteri programmabili quando si doveva appunto mettere a 0 questo bit e si escludeva pertanto gli interrupts.

## CLASSIFICAZIONE DEGLI I/O

|  |           |
|--|-----------|
| D000-D3FF VIC (VIDEO CONTROLLER)       | 1K        |
| D400-D7FF SID (SINTETIZZ. SONORO)      | 1K        |
| D800-DBFF RAM COLORE                   | 1/2K      |
| DC00-DCFF CIA1 (TASTIERA)              | 256 BYTES |
| DD00-DDFF CIA2 (BUS SERIALE,U/P,RS232) | 256 BYTES |
| DE00-DEFF Open I/O slot 1 (per CP/M)   | 256 BYTES |
| DF00-DFFF Open I/O slot 2 (Disco)      | 256 BYTES |

I due Slots di input/output sono per funzioni generali di I/O, funzioni speciali di cartridges in I/O ( come l' interfaccia IEEE-488) e sono stati progettati per il cartridge Z-80 ( per il CP/M) e per interfacciare con un sistema a dischi ad alta velocità'.

## I CARTRIDGES

Anche su questo computer, come viene fatto largamente per il VIC-20, e' previsto, ed in parte impiegato (vedi prodotti disponibili) il sistema di espansione tramite Cartridges.

Il programma su Cartridges va in esecuzione se i primi nove Bytes della ROM o EPROM del Catridge stesso di locazione 32768 (\$8000) contengono determinati dati. Vediamo quali sono.

a) I primi due Bytes devono contenere il vettore di COLD START che deve essere usato dal programma appunto su Cartridge.

b) I secondi due Bytes, quindi di indirizzo 32770, 32771 (\$8002, \$8003) devono contenere il vettore di WARM START.

c) I successivi 3 bytes devono contenere le lettere CBM, con il bit 7 settato in ogni lettera.

d) Infine gli ultimi due Bytes devono contenere 80 in CBM ASCII.

## LE ROUTINE KERNAL

Senza voler tenere un corso di linguaggio macchina ne un approfondimento sul Sistema Operativo ci e' sembrato indispensabile comunque citare per esteso le routines del S.O. che riguardano il CBM 64 e che si riferiscono al controllo dei dati e delle informazioni in ingresso ed uscita dal computer.

Malgrado quindi che l' uso di queste routines sia riservato ai programmatori piu' esperti, l' utilizzo di questi strumenti non mancherà di riservare notevoli soddisfazioni a chiunque voglia cimentarsi con questa parte del funzionamento del computer stesso.

Comunque ci auguriamo che quanto scritto possa essere di aiuto alla comprensione di fenomeni di trasmissione dati e di funzionamento complessivo del sistema di trattamento dati da e verso le periferiche.

Non ultimo la conoscenza degli indirizzi permettera' oltre che una migliore risposta in termini di programmazione e di efficienza complessiva anche la possibilita' di effettuare variazioni al sistema stesso di manipolazione dati e consentira' di essere preparati a tutte le modifiche che il costruttore vorra' apportare in futuro.

Anche per chi opera in Linguaggio Macchina e' molto piu' semplice limitarsi a delle modifiche che non riscrivere completamente delle routines e poi provarle.

Le routines KERNAL di INPUT e OUTPUT non sono altro che quella parte del Sistema Operativo che consentono al computer di colloquiare con le periferiche esterne. Vediamo ora come funzionano e le descrizioni che forniremo serviranno ad ottenere un collegamento particolarmente efficace con, ad esempio, il DOS, cioe' il Sistema Operativo residente su disco.

Non dimentichiamo infatti che, per i prodotti COMMODORE, la periferica disco e' sempre un' unita' intelligente.

## ATTIVITA' DEL S.O. E DEL KERNAL IN PARTICOLARE.

Vediamo come si comportano queste routine alle quali di solito si accede attraverso un' istruzione in codice macchina JSR cioe' JUMP TO SUBROUTINE.

1) All' accensione, il Sistema Operativo per prima cosa resetta lo Stack Pointer ed azzerà il modo decimale.

2) Per seconda cosa controlla la presenza di una eventuale ROM dotata di AUTOSTART, cioe' di partenza automatica all' indirizzo esadecimale \$8000 (32768 in decimale).

Se viene incontrata la routine di autostart sul cartridge, ed e' bene anche ricordare che non tutti i cartridges sono dotati di questa routine, allora il controllo viene trasferito al programma presente nella ROM stessa, mentre in caso contrario prosegue la procedura di inizializzazione.

E' bene anche ricordare che di solito con le informazioni contenute in cartridge viene modificato solo in piccola parte il Sistema Operativo , ad esempio con la disabilitazione del RUN/STOP e RESTORE, del LIST ed altro.

Tutto cio' spesso per problemi di sicurezza e di non copiabilita' del programma.

3) Successivamente le routines Kernal provvedono ad inizializzare tutte le periferiche di INPUT/OUTPUT come pure il bus seriale.

Entrambi gli integrati CIA 6526 sono fissati al giusto valore per la scansione di tastiera e viene attivato anche il timer da 60 Hz ed azzerato il SID.

Viene selezionata fra l' altro la normale mappa di memoria del Basic e viene eseguito lo spegnimento del motore della cassetta.

4) Le routines Kernal eseguono anche un controllo della RAM fissando i puntatori di inizio e fine memoria. Viene inizializzata la pagina zero con gli opportuni valori e fissata la zona del buffer di cassetta.

5) Vengono eseguite altre funzioni fra le quali quelle che ci interessano particolarmente in questa sede sono l'impostazione a valori normali di procedimento, cioè la loro inizializzazione, dei valori delle tavole dei valori di salto indiretti.

## COME UTILIZZARE LE KERNAL

Quando si scrivono programmi in Linguaggio Macchina e' conveniente usare le routines riguardanti l' input/output che fanno già parte del sistema operativo.

Con quanto segue infatti e' reso abbastanza agevole l' accesso al Sistema Operativo stesso il che rende, come già detto la riscrittura di queste o di parte di queste routine, uno sforzo inutile.

Come abbiamo già detto il sistema KERNAL e' costituito da una serie di indirizzi di salto utilizzabili attraverso un' istruzione JMP che consentono appunto di saltare alle varie routines.

Per poter utilizzare una procedura del Sistema Operativo e' necessario per prima cosa eseguire tutte le operazioni richieste dalla procedura stessa.

Per esempio se una routine KERNAL deve essere chiamata prima di un' altra, questa operazione deve essere eseguita nella sequenza logica ordinata, ricordandosi che quando si opera in Linguaggio Macchina vengono meno le protezioni previste dal Basic a livello di Errori di sintassi. Altro esempio. Se la procedura richiede che preventivamente sia inserito un numero nell' Accumulatore o nel Registro X, questo deve essere fatto perche' in caso contrario otterremo dei risultati strani o addirittura la procedura non funzionera' per niente.

Dopo aver preparato accuratamente il piano di lavoro occorre chiamare in funzione la routine per mezzo dell'istruzione JSR.

Tutte le routine Kernal a cui si accede sono strutturate come Subroutines in Linguaggio Macchina e quindi, per non finire in un LOOP o ciclo infinito e' necessario che terminino con un' istruzione RTS, cioe' RETURN FROM SUBROUTINE.

**\*\*NOTA\*\***

E' da ricordare che quando la routine Kernal ha terminato la sua funzione, di solito il controllo viene restituito al programma, (stiamo sempre parlando di programmi in Linguaggio Macchina), immediatamente dopo l' istruzione JSR.

Molto spesso gli eventuali errori generati dall' utilizzo di queste subroutines, come del resto si puo' vedere nella descrizione delle stesse, sono riportati nella PAROLA DI STATO o STATUS WORD o nell' Accumulatore che pertanto andranno letti per vedere che errore e' segnalato.

Oltre ad una certa pratica che si potra' acquisire con il tempo, una gestione degli errori facilitera' di molto il compito del programmatore che ricordiamo non e' assistito dalle procedure automatiche di errore tipiche del Basic. Quindi lo schema dei passi essenziali per l' uso delle Kernal e' il seguente:

1- AVVIAMENTO

2- SALTO ALLA ROUTINE

3- GESTIONE D' ERRORE

Nella descrizione delle routines useremo i seguenti termini convenzionali:

FUNZIONE: che e' il nome. Ovviamente e' solo un' indicazione mnemonica, della routine stessa.

INDIRIZZO: viene riportato l' indirizzo di inizio della routine sia in esadecimale che in decimale.

REGISTRI DI COMUNICAZIONE: i registri qui riportati sono utilizzati per passare parametri da e alla subroutine Kernal.

ROUTINES DI PREPARAZIONE: alcune routine richiedono che siano chiamate in funzione ma SOLO dopo che alcuni dati sono stati trattati da altre routines. Diamo quindi le routines di preparazione per l' utilizzo della Kernal stessa.

ERRORI: un ritorno da una routine Kernal con il FLAG di CARRY settato indica che durante la fase operativa della routine stessa e' stato incontrato un errore. Il numero dell' errore e di conseguenza il suo significato sara' riportato nell' Accumulatore.

BYTES DI STACK: e' il numero di bytes dello STACK usati dalla routine stessa.

REGISTRI USATI: vengono riportati uno o piu' registri utilizzati dalla routine durante il suo lavoro o in I/O.

DESCRIZIONE: una spiegazione, per quanto sommaria della funzione e del modo di funzionamento della routine Kernal.

# LE ROUTINES KERNAL

| NOME   | INDIRIZZO |       | FUNZIONE                              |
|--------|-----------|-------|---------------------------------------|
|        | HEX       | DEC   |                                       |
| ACPTR  | \$FFA5    | 65445 | INGRESSO BYTE DALLA PORTA SERIALE     |
| CHKIN  | \$FFC6    | 65478 | APRE UN CANALE PER INPUT              |
| CHKOUT | \$FFC9    | 65481 | APRE UN CANALE PER OUTPUT             |
| CHRIN  | \$FFCF    | 65487 | INGRESSO DI UN CARATTERE DA UN CANALE |
| CHROUT | \$FFD2    | 65490 | OUTPUT DI UN CARATTERE A UN CANALE    |
| CIOUT  | \$FFA8    | 65448 | OUTPUT DI UN BYTE ALLA PORTA SERIALE  |
| CINT   | \$FF81    | 65409 | INIZIALIZZA L' EDITOR DI SCHERMO      |
| CLALL  | \$FFE7    | 65511 | CHIUDI TUTTI I CANALI E I FILES       |
| CLOSE  | \$FFC3    | 65475 | CHIUDI UN DATO FILE LOGICO            |
| CLRCHN | \$FFCC    | 65484 | CHIUDI I CANALI IN I/O                |
| GETIN  | \$FFE4    | 65508 | RICEVE UN CARATT. DALLA CODA DI TAST. |
| IOBASE | \$FFF3    | 65523 | RIPORTA INDIRIZZO DI BASE PERIF. I/O  |
| IOINIT | \$FF84    | 65412 | INIZIALIZZA INPUT/OUTPUT              |
| LISTEN | \$FFB1    | 65457 | COMANDO AL BUS SERIALE PER LISTEN     |
| LOAD   | \$FFD5    | 65493 | CARICA RAM DA PERIFERICA              |



|        |        |       |  |
|--------|--------|-------|--|
| MEMBOT | \$FFC9 | 65436 | LEGGE/FISSA IL MINIMO DI MEMORIA       |
| MEMTOP | \$FF99 | 65433 | LEGGE/FISSA IL MASSIMO DI MEMORIA      |
| OPEN   | \$FFC0 | 65472 | APRE UN FILE LOGICO                    |
| PLOT   | \$FFF0 | 65520 | LEGGE/FISSA POSIZIONE DEL CURSORE      |
| RAMTAS | \$FF87 | 65415 | INIZ. RAM, DISPONE PER BUFFER NASTRO   |
| RDTIM  | \$FFDE | 65502 | LEGGE OROLOGIO IN TEMPO REALE          |
| READST | \$FFB7 | 65463 | LEGGE IN I/O LO STATO DELLA PAROLA     |
| RESTOR | \$FF8A | 65418 | REINTEGRA ASSENZA DEI VETTORI DI I/O   |
| SAVE   | \$FFD8 | 65496 | SALVA RAM SU PERIFERICA                |
| SCNKEY | \$FF9F | 65439 | SCANSIONE DI TASTIERA                  |
| SCREEN | \$FFED | 65517 | RIPORTA ORGANIZZ. SCHERMO DI X,Y       |
| SECOND | \$FF93 | 65427 | INVIA L'INDIRIZO SECON. DOPO LISTEN    |
| SETLFS | \$FFBA | 65466 | FISSA INDIR LOGICO E SECONDARIO        |
| SETMSG | \$FF90 | 65420 | CONTROLLO MESSAGGI KERNAL              |
| SETNAM | \$FFBD | 65469 | FISSA IL NOME DEL FILE                 |
| SETTIM | \$FFDB | 65499 | FISSA L' OROLOGIO (R.T.CLOCK)          |
| SETTMO | \$FFA2 | 65442 | FISSA IL TIMEOUT (fuori tempo)SUL BUS. |
| STOP   | \$FFE1 | 65505 | SCANSIONE DEL TASTO DI STOP            |

|        |        |       |                                     |
|--------|--------|-------|-------------------------------------|
| TALK   | \$FFB4 | 65460 | COM. SUL BUS SER. PER TALK          |
| TKSA   | \$FF96 | 65430 | INVIA IND. SEC. DOPO TALK           |
| UDTIM  | \$FFEA | 65514 | INCREMENTA OROLOGIO                 |
| UNLSN  | \$FFAE | 65454 | COMANDO AL BUS SERIALE PER UNLISTEN |
| UNTLK  | \$FFAB | 65451 | COMANDO AL BUS SERIALE PER UNTALK   |
| VECTOR | \$FF8D | 65421 | LEGGE/FISSA GLI I/O VETTORIZZATI    |

## DESCRIZIONE DELLE ROUTINES KERNAL

B-1 Nome della funzione: ACPTR

FUNZIONE:riceve dati dal bus seriale

INDIRIZZO:\$FFA5 - 65445

REGISTRI DI COMUNICAZIONE:.A

ROUTINES DI PREPARAZIONE: TALK, TKSA

ERRORI:Vedi READST

BYTES DI STACK USATI: 13

REGISTRI USATI: A,X

### DESCRIZIONE:

Questa e' la routine che si usa quando si desidera ricevere informazioni da una periferica attraverso il BUS seriale, per esempio da disco.

Questa routine riceve un Byte di dati dal Bus usando un HANDSHAKING pieno ed il dato e' riportato in Accumulatore.

La routine TALK deve essere chiamata in funzione prima di ordinare alla periferica di inviare dati sul Bus.

Se la periferica in ingresso necessita di un comando secondario, questo deve essere inviato usando la routine TKSA prima di ACPTR.

Se ci sono errori saranno riportati nella PAROLA di STATO (STATUS WORD) il cui contenuto potra' essere letto dalla routine READST.

## MODO DI UTILIZZO

- 1) Ordinare alla periferica collegata al Bus seriale di prepararsi ad inviare dati al CBM64 usando le routines TALK e TKSA.
- 2) Saltare, con JSR, quindi alla routine ACPTR.
- 3) Immagazzinare o utilizzare i dati ricevuti.

B-2 Nome della funzione: CHKIN

FUNZIONE: Apre un canale per INPUT

INDIRIZZO: \$FFC6 - 65478

REGISTRI DI COMUNICAZIONE: .X

ROUTINES DI PREPARAZIONE: (OPEN)

ERRORI: Vedi nota

BYTES DI STACK USATI: Nessuno

REGISTRI USATI: .A, .X

### DESCRIZIONE:

Un qualsiasi File logico che sia stato aperto per mezzo della routine OPEN puo' essere definito come un canale di Input per mezzo di questa routine.

Naturalmente la periferica sul canale deve essere una periferica in input, perche' altrimenti avremo un errore e la CHKIN non avra' effetto.

Se si stanno ricevendo dati da una qualsiasi altra parte che non sia la tastiera, questa routine (OPEN) deve essere chiamata prima di usare sia le routine CHRIN che GETIN per l' Input dei dati.

Se si desidera usare l' Input da tastiera, e nessun altro canale di input e' aperto, allora la chiamata a questa Routine e alla routine OPEN non e' necessaria.

Quando questa routine e' utilizzata con una periferica sul bus Seriale, essa invia automaticamente l' indirizzo di chiamata ( e l' indirizzo secondario se questo e' specificato in OPEN) sul Bus.

## MODO DI UTILIZZO

- 1) Eseguire l'OPEN del file logico (se necessario)
- 2) Caricare il registro X con il numero del file logico che deve essere usato.
- 3) Saltare a questa routine.

## POSSIBILI ERRORI

- # 3: File non aperto
- # 5: Periferica non presente ( o non collegata)
- # 6: Il file non e' un file input.

B-3 Nome della funzione: CHKOUT

FUNZIONE: Apre un canale per OUTPUT

INDIRIZZO: \$FFC9 - 65481

REGISTRI DI COMUNICAZIONE: X

ROUTINES DI PREPARAZIONE: OPEN

ERRORI: 0,3,5,7

BYTES DI STACK USATI: 4+

REGISTRI USATI: A,X

#### DESCRIZIONE:

Un qualsiasi numero di File logico che sia stato creato dalla routine OPEN puo' essere definito come un canale di OUTPUT.

Percio' la periferica deve essere una periferica in OUTPUT cioe' in uscita perche' in caso contrario avremo una segnalazione di errore.

Questa routine (CHKOUT) deve essere messa in funzione prima che un qualsiasi dato sia inviato ad una periferica (naturalmente in uscita) a meno che non si desideri usare lo schermo in funzione di periferica in uscita.

Quando e' usata per aprire un canale per una periferica sul Bus seriale, questa nostra routine inviera' automaticamente l' indirizzo di LISTEN specificato dalla routine OPEN e l' indirizzo secondario se esiste.

#### MODO DI UTILIZZO

1) Utilizzare la routine OPEN per specificare un numero di file logico, l' indirizzo di LISTEN e l' indirizzo secondario se necessario.

2) Caricare il registro X con il numero di File logico usato nel comando OPEN.

3) Chiamare ora la CHKOUT

## POSSIBILI ERRORI

- # 3: File non aperto
- # 5: Periferica non presente ( o non collegata)
- # 7: Non e' un file in uscita

B-4 Nome della funzione: CHRIN

FUNZIONE: Riceve un carattere da un canale di Input

INDIRIZZO: \$FFCF - 65487

REGISTRI DI COMUNICAZIONE:A

ROUTINES DI PREPARAZIONE: (OPEN, CHKIN)

ERRORI:Vedi READST

BYTES DI STACK USATI: 7+

REGISTRI USATI: A, X

### DESCRIZIONE:

Questa routine riceve un byte di dati da un canale gia' selezionato per mezzo della routine CHKIN come canale in INPUT.

Se CHKIN non e' stata usata per definire un diverso canale di input allora i dati saranno attesi da tastiera. Il Byte di dati e' caricato in accumulatore ed il canale resta aperto.

L' ingresso da tastiera e' manipolato in maniera particolare.

Per prima cosa e' attivato il cursore che lampeggera' fino alla digitazione di un ritorno carrello da tastiera (cioe' fino a quando non sia premuto il RETURN).

Tutti i caratteri della linea ( max 88) sono immagazzinati nel BASIC INPUT BUFFER.

Questi caratteri sono recuperati ad uno ad uno per mezzo di tanti salti a questa routine quanti sono questi caratteri.

Quando viene incontrato il ritorno carrello l' intera linea e' stata manipolata.

## MODO DI UTILIZZO

Da tastiera

1) Rintraccia un Byte di dati per mezzo del salto a questa routine.

2) Immagazzina il Byte

3) Controlla se e' l' ultimo Byte, cioe' se e' un RETURN

4) Se non e' un RETURN, ritorna al passo 1).

Da altre periferiche

1) Usare OPEN e CHKIN

2) Saltare a CHRIN

3) Immagazzinare i dati

B-5 Nome della funzione: CHROUT



FUNZIONE: Uscita di un carattere

INDIRIZZO: \$FFD2 - 65490

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:(CHKOUT, OPEN)

ERRORI: Vedi READST

BYTES DI STACK USATI: 8+

REGISTRI USATI: A

#### DESCRIZIONE:

Questa routine fa uscire un carattere su un canale già aperto.

E' necessario usare le routines OPEN e CHKOUT per fissare un canale di uscita prima di chiamare questa routine.

Nel caso che queste chiamate siano omesse i data saranno inviati alla periferica base in uscita, cioè la numero 3 il video.

I Byte che devono uscire , cioè che sono in Output sono caricati nell' Accumulatore, viene chiamata la routine CHROUT e successivamente i dati sono inviati alla periferica selezionata, mentre il canale viene lasciato aperto.

#### MODO DI UTILIZZO

- 1) Usare la CHKOUT se necessario. Vedi quanto già scritto.
- 2) Caricare i dati nell' accumulatore
- 3) Saltare ora a CHROUT

B-6 Nome della funzione: CIOUT

FUNZIONE: Trasmette un Byte sul bus seriale.

INDIRIZZO:\$FFAB - 65448

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE: LISTEN (SECOND)

ERRORI:Vedi READST

BYTES DI STACK USATI:5

REGISTRI USATI:

DESCRIZIONE:

Questa routine e' utilizzata per inviare informazioni a periferiche collegate al bus seriale.

Percio' la messa in funzione di questa routine avra' come conseguenza l' immissione di un byte di dati sul bus seriale usando un HANDSHAKING seriale pieno.

Prima di chiamare questa routine, deve essere chiamata la routine LISTEN che ordinerà alla periferica sul BUS seriale di tenersi pronta a ricevere i dati. (Se alla periferica necessita un indirizzo secondario questo deve essere inviato attraverso l' utilizzo della routine SECOND che vedremo in seguito).

La periferica deve essere in ascolto o sarà generato, attraverso la parola di stato , un errore di fuori tempo (TIMEOUT).

MODO DI UTILIZZO

1) Usare per prima cosa la routine LISTEN e poi la SECOND se necessaria.

2) Caricare l' Accumulatore con un Byte di dati.

3) Saltare a questa routine per inviare il Byte di dati

B-7 Nome della funzione: CINT

FUNZIONE: Inizializza l' editor di schermo e l' integrato 6567

INDIRIZZO:\$FF81 - 65409

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:4

REGISTRI USATI:A, X, Y

DESCRIZIONE:

Questa routine abilita l' integrato 6567 (VIDEO CONTROLLER) nel CBM64 per le normali operazioni. Viene inizializzato anche il KERNAL SCREEN EDITOR. Dovrebbe essere chiamata in funzione da un catridge.

MODO DI UTILIZZO

1) Salto a questa routine da cui si dovrebbe passare al RUN automatico

B-8 Nome della funzione: CLALL

FUNZIONE: Chiude tutti i Files

INDIRIZZO:\$FFE7 - 65511

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:11

REGISTRI USATI: A, X

DESCRIZIONE:

Questa routine serve per chiudere tutti i files aperti. Quando entra in funzione questa routine i puntatori della tavola dei file aperti sono resettati, chiudendo così tutti i files.

Anche la routine CLRCHN viene chiamata per resettare tutti i canali di I/O.

MODO DI UTILIZZO

1) Saltare semplicemente a questa routine.

B-9 Nome della funzione: CLOSE

FUNZIONE: Chiude un file logico

INDIRIZZO:\$FFC3 - 65475

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:Vedi READST

BYTES DI STACK USATI:2+

REGISTRI USATI:A, X, Y

#### DESCRIZIONE:

Questa routine e' utilizzata per chiudere un file logico dopo che tutte le operazioni di I/O sullo stesso file sono state eseguite.

La routine e' chiamata dopo che l' accumulatore e' stato caricato con il numero di file logico che deve essere chiuso.

Naturalmente questo sara' lo stesso numero usato quando il file era stato aperto con OPEN.

#### MODO DI UTILIZZO

1) Caricare l' accumulatore con il numero di file logico che deve essere chiuso.

2) Eseguire la routine.

B-10 Nome della funzione: CLRCHN

FUNZIONE: Pulisci i canali di I/O

INDIRIZZO:\$FFCC - 65484

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:9

REGISTRI USATI:A, X

DESCRIZIONE:

Questa routine e' utilizzata per eseguire il CLEAR di tutti i canali aperti e ripristinare gli stessi canali ai loro valori originari.

La periferica normale di INPUT e' 0 (cioe' la tastiera) , mentre la periferica normale di OUTPUT e' 3 ( cioe' il video).

Se uno dei canali che deve essere chiuso e' su una porta seriale, viene inviato per prima cosa un segnale di UNTALK per eseguire la pulizia del canale di Input o un segnale di UNLISTEN per la pulizia del canale di Output.

Non eseguendo la chiamata a questa routine e quindi lasciando gli ascoltatori ( LISTENERS ) attivi sul bus seriale, diverse periferiche possono ricevere gli stessi dati dal CBM64 allo stesso tempo.

Un sistema per utilizzare questa particolarita' potrebbe essere quello di mettere la stampante in TALK e il disco in LISTEN per consentire la stampa diretta di un file disco.

La routine CLRCHN entra automaticamente in funzione dopo l' esecuzione di CLALL.

#### MODO DI UTILIZZO

1) Saltare a questa routine con JSR

B-11 Nome della funzione: GETIN

FUNZIONE: Riceve un carattere da periferica.

INDIRIZZO:\$FFE4 - 65508

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE: CHKIN OPEN

ERRORI:Vedi READST

BYTES DI STACK USATI:5+

REGISTRI USATI: A, (X, Y)

#### DESCRIZIONE:

Se il carattere e' da tastiera, questa routine prende un carattere dalla coda di tastiera (KEYBOARD QUEUE) e lo riporta nell' Accumulatore come valore ASCII.

Se il buffer ( cioe' la coda di tastiera che e' appunto un buffer) e' vuota allora il carattere caricato nell' Accumulatore sara' zero.

I caratteri sono immessi nella coda di tastiera

utilizzando sia una parte HARDWARE (INTERRUPT DRIVEN KEYBOARD) sia la routine di scansione della tastiera SCNKEY.

Il buffer puo' contenere al massimo 10 caratteri per cui se e' pieno gli altri caratteri che si tentera' di immettere saranno ignorati fino a quando almeno un carattere non sia rimosso dalla coda.

Se il canale invece di essere la tastiera e' l' RS-232 allora viene usato solo il registro A e viene riportato un solo carattere e sara' necessario utilizzare READST per il controllo di validita'.

Se il canale invece e' seriale, cassetta o schermo e' chiamata la routine BASIN.

## MODO DI UTILIZZO

- 1) Chiamare la routine con una istruzione JSR.
- 2) Eseguire il controllo per 0 nell' Accumulatore
- 3) Manipolare i dati.

B-12 Nome della funzione: IOBASE

FUNZIONE: Definisce la pagina di memoria I/O

INDIRIZZO:\$FFF3 - 65523

REGISTRI DI COMUNICAZIONE: X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:



BYTES DI STACK USATI:2

REGISTRI USATI:X, Y

#### DESCRIZIONE:

Questa routine fissa i registri X e Y all' indirizzo della sezione di memoria che definisce dove sono localizzate le periferiche I/O.

Questo indirizzo puo' essere utilizzato come linea di deviazione (OFFSET) per accedere alla memoria disegnata per le periferiche I/O del CBM64.

La linea di deviazione e' il numero di locazioni dall' inizio della pagina sulla quale si desidera che i registri I/O siano immessi.

Naturalmente il registro X conterra' il Byte di indirizzo piu' basso mentre Y la parte alta dello stesso indirizzo.

#### MODO DI UTILIZZO

- 1) Eseguire il JSR a questa routine
- 2) Immagazzinare i due registri X e Y in due indirizzi consecutivi.
- 3) Caricare il registro Y con OFFSET
- 4) Accedera a quella locazione di I/O.

B-13 Nome della funzione: IOINIT

FUNZIONE: Inizializza periferiche I/O

INDIRIZZO:\$FF84 - 65412

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:

REGISTRI USATI: A, Y, X

DESCRIZIONE:

Questa routine inizializza tutte le periferiche di I/O e le routines.

E' normalmente chiamata come parte di una procedura di inizializzazione di un programma su cartridge

B-14 Nome della funzione: LISTEN

FUNZIONE: Invia un comando di ascolto ad una periferica sul Bus seriale

INDIRIZZO:\$FFB1 - 65457

REGISTRI DI COMUNICAZIONE:A

ROUTINES DI PREPARAZIONE:

ERRORI:Vedi READST

BYTES DI STACK USATI:

REGISTRI USATI: A

#### DESCRIZIONE:

Questa routine ordina ad una periferica sul Bus seriale di ricevere dati.

L' Accumulatore deve essere caricato con un numero compreso fra 0 e 31 prima di chiamare questa routine.

LISTEN eseguirà un OR logico sul numero bit per bit per convertirlo in un indirizzo di ascolto e poi trasmetterà questo dato come comando sul bus seriale.

La periferica specificata si metterà allora in modo di ascolto e sarà pronta per ricevere informazioni.

#### MODO DI UTILIZZO

1) Carica l' Accumulatore con il numero della periferica alla quale deve essere inviato l' ordine di ascolto.

2) Vai alla routine con un comando di JSR.

B-15 Nome della funzione: LOAD

FUNZIONE: Carica RAM da una periferica

INDIRIZZO:\$FFD5 - 65493

REGISTRI DI COMUNICAZIONE:A,X,Y

ROUTINES DI PREPARAZIONE:SETLFS, SETNAM

ERRORI:Vedi READST

BYTES DI STACK USATI:

REGISTRI USATI: A, Y, X

#### DESCRIZIONE:

Questa routine carica Bytes di dati da una qualsiasi periferica in INPUT direttamente entro la memoria del CBM64.

Puo' anche essere usata per una operazione di verifica che avviene confrontando i dati presenti sulla periferica con quelli in memoria e lasciando i dati in memoria inalterati.

L' Accumulatore deve essere messo a 0 per un' operazione di LOAD o messo a 1 per un' operazione di verifica.

Se la periferica in Input e' aperta con un' indirizzo secondario di 0, allora sara' ignorata la testata (HEADER) dell' informazione.

In questo caso i registri X e Y devono contenere l' indirizzo di partenza per LOAD.

Se la periferica e' collegata con un ' indirizzo secondario 1 0 2, allora i dati saranno caricati in memoria con partenza dall' indirizzo specificato dalla testata.

Questa routine inoltre riporta l' indirizzo della piu' alta locazione di RAM caricata.

Prima di chiamare questa routine e' necessario chiamare le routines SETLFS e SETNAM.

#### NOTA

Non si puo' eseguire il LOAD da Tastiera (0), RS-232 (2) o schermo (3).

## MODO DI UTILIZZO

- 1) Eseguire per prima cosa le routines SETLFS e SETNAM
- 2) Mettere A a 0 per LOAD o a 1 per VERIFY.
- 3) Saltare alla routine con JSR

B-16 Nome della funzione: MEMBOT

FUNZIONE: Fissa la parte piu' bassa della memoria

INDIRIZZO:\$FF9C - 65436

REGISTRI DI COMUNICAZIONE: X Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:

REGISTRI USATI: Y, X

## DESCRIZIONE:

Questa routine e' usata per mettere a 1 la parte piu' bassa della memoria.

Se il bit di Carry dell' Accumulatore e' a 1 quando viene chiamata questa routine, allora un puntatore che indica il Byte piu' basso della RAM e' riportato in X e Y.

Normalmente sul CBM 64 il puntatore e' \$0800 cioe' 2048.

Se il bit di Carry dell' Accumulatore e' a 0 quando viene chiamata questa routine, allora il valore dei

registri X e Y sarà trasferito rispettivamente al più basso e più alto Byte del puntatore all' inizio della RAM.

## MODO DI UTILIZZO

Per leggere il punto di partenza della RAM:

- 1) Mettere il Carry a 1
- 2) Chiamare la Routine

Per leggere il punto di partenza della Memoria:

- 1) Mettere il Carry a 0
- 2) Chiamare la routine

B-17 Nome della funzione: MEMTOP

FUNZIONE: Fissa la parte alta della memoria

INDIRIZZO: \$FF99 - 65433

REGISTRI DI COMUNICAZIONE: X Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: Y, X

#### DESCRIZIONE:

Questa routine e' utilizzata per fissare il punto massimo della memoria RAM.

Il funzionamento e' simile alla routine precedente (MEMBOT).

Infatti anche in questo caso quando si utilizza questa Routine con il bit di Carry dell' Accumulatore a 1, il puntatore alla fine della memoria RAM e' caricato nei registri X e Y.

Quando invece la si utilizza con il bit di carry a 0, allora il contenuto dei registri X e Y e' caricato nel puntatore al massimo della memoria.

B-18 Nome della funzione: OPEN

FUNZIONE: Apre un file logico

INDIRIZZO:\$FFC0 - 65472

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE: SETLFS , SETNAM

ERRORI: Vedi READST

BYTES DI STACK USATI:

REGISTRI USATI: A, Y, X

#### DESCRIZIONE:

Questa routine e' utilizzata per eseguire la funzione di

apertura di un File logico.

Non appena il file logico e' stato fissato questi .puo' essere utilizzato per operazioni di I/O.

Molte delle Routines del sistema operativo fanno uso di OPEN.

Non sono necessari argomenti o operandi ma prima di utilizzare questa routine sara' necessario metterne in funzione altre due cioe' la SETLFS e la SETNAM.

#### MODO DI UTILIZZO

- 1) Utilizzare la SETLFS
- 2) Utilizzare quindi la SETNAM
- 3) Saltare a OPEN

B-13 Nome della funzione: PLOT-

FUNZIONE: Legge e fissa la posizione del cursore

INDIRIZZO:\$FFFF - 65520

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A, Y, X

DESCRIZIONE:



Quando si salta a questa routine con il carry dell' accumulatore a 1, allora la posizione attuale del cursore , nelle sue coordinate X e Y sara' caricata nel registri Y e X dove Y sara' il numero di colonna del cursore ( da 0 a 79) e X il numero di riga occupato dal cursore ( da 0 a 24).

Se invece il carry e a 0 allora verranno letti i valori dei registri X e Y e il cursore posizionato a quei valori.

## MODO DI UTILIZZO

Lettura della locazione del cursore:

- 1) Metti a 1 il carry
- 2) Vai alla routine
- 3) Ricava la posizione del cursore dalla lettura del contenuto dei registri X e Y per le posizioni X e Y cioe' ascissa e ordinata.

Fissa l' indirizzo del cursore

- 1) Metti a 0 il carry
- 2) Carica nei registri X e Y la posizione del cursore
- 3) Esegui la routine

B-20 Nome della funzione: RAMTAS

FUNZIONE: Controlla le RAM, fissa aree per buffer nastro e schermo

INDIRIZZO: \$FF87 - 65415

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A, X, Y

DESCRIZIONE:

Questa routine e' utilizzata per controllare la memoria RAM e fissare i puntatori della memoria sia in alto che in basso.

Esegue anche il clear delle locazioni \$0000 fino a \$0101 e da \$0200 a \$03FF.

Fissa anche il buffer di cassetta e fissa la locazione base di schermo a partire da \$0400.

Normalmente questa routine e' chiamata come parte di un processo di inizializzazione di-un Cartridge.

B-21 Nome della funzione: RDTIM

FUNZIONE: Legge l'orologio in tempo reale

INDIRIZZO: \$FFDE - 65502

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A, X, Y

DESCRIZIONE:

Questa routine e' utilizzata per leggere il clock o orologio di sistema.

La risoluzione del clock, cioe' il tempo minimo e' di 1 60mo di secondo.

Il risultato della lettura di questa routine e' di 3 Bytes che sono riportati rispettivamente nell' Accumulatore , nel registro X e Y..

Operando con questi tre registri e, come vedremo poi con la routine SETTIM e' possibile leggere e variare il contenuto dell' orologio del sistema.

B-22 Nome della funzione: READST

FUNZIONE: Legge lo STATUS WORD

INDIRIZZO: \$FFB7 - 65463

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A

## DESCRIZIONE:

Questa routine riporta lo stato attuale delle periferiche in I/O nell' accumulatore.

E' utilizzata di norma dopo ogni colloquio con le periferiche e riporta le informazioni sullo stato delle periferiche stesse o eventuali errori incontrati durante operazioni di I/O.

I BITS di questa routine, caricati nell' Accumulatore contengono le informazioni illustrate nella seguente tavola:

| POSIZ.<br>BIT | VALORE<br>NUM. | DESCRIZIONE              |
|---------------|----------------|--------------------------|
| 0             | 1              | Time out in scrittura    |
| 1             | 2              | " " lettura              |
| 2             | 4              | Blocco corto             |
| 3             | 8              | Blocco lungo             |
| 4             | 16             | Err. di lettura irrecup. |
| 5             | 32             | Errore di CHECKSUM       |
| 6             | 64             | END OF FILE              |
| 7             | -128           | END OF TAPE *            |

\* Questo tipo di errore sul Bus seriale sta a significare invece un : DEVICE NOT PRESENT.

## MODO DI UTILIZZO

1) Richiamare questa routine

2) Decodificare l' informazione presente nell' Accumulatore ricavandone i significati appena esposti per utilizzarli eventualmente in programma.

B-23 Nome della funzione: RESTOR

FUNZIONE: Reintegra i vettori di sistema.

INDIRIZZO: \$FF8A - 65418

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A, X, Y

DESCRIZIONE:

Questa routine reintegra i valori mancanti dei vettori di tutto il sistema usati sia nelle KERNAL che nel BASIC come routines e come interrupts.

MODO DI UTILIZZO

1) Saltare a questa routine

B-24 Nome della funzione: SAVE

FUNZIONE: Salva la memoria RAM su periferica

INDIRIZZO: \$FF8D - 65496

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:SETLFS, SETNAM

ERRORI:Vedi READST

BYTES DI STACK USATI:

REGISTRI USATI: A, X, Y

## DESCRIZIONE:

Questa routine e' utilizzata per eseguire l' operazione di SAVE di una parte di memoria.

La memoria e' salvata da un' indirizzo indiretto in pagina 0 specificato dall' Accumulatore a un' indirizzo immagazzinato nei registri X e Y.

Sara' quindi inviato ad un File logico su una periferica. Le routines SETLFS e SETNAM devono essere utilizzate prima di accedere a questa routine.

Tuttavia non e' necessario dare un nome al file che si desidera salvare su cassetta, mentre e' necessario per qualsiasi altra periferica.

## MODO DI UTILIZZO

1) Usare SETLFS e SETNAM

2) Caricare due locazioni consecutive di memoria in pagina 0 con un puntatore all' inizia della memoria che si desidera salvare. Ricordarsi che il formato deve essere quello standard del 6502, cioe' prima il Byte basso e poi

il Byte alto dell' indirizzo.

3) Caricare l' Accumulatore con il singolo Byte di pagina zero deviato al puntatore.

4) Caricare i registri X e Y rispettivamente con il Byte basso, Byte alto dell' indirizzo di fine memoria. (cioe' della fine della memoria che si vuole caricare).

5) Saltare a SAVE

B-25 Nome della funzione: SCNKEY

FUNZIONE: Esegue la scansione di tastiera

INDIRIZZO: \$FF9F - 65439

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE: IOINIT

ERRORI:

BYTES DI STACK USATI: 5

REGISTRI USATI: A, X, Y

DESCRIZIONE:

Questa routine esegue la scansione (cioe' la lettura) della tastiera e controlla se ci sono tasti premuti. E' la stessa routine chiamata per mezzo della manipolazione di Interrupt. Se un tasto e' premuto, allora il suo valore ASCII e' immesso nella coda di tastiera.

## MODO DI UTILIZZO

1) Salto alla routine stessa

B-26 Nome della funzione: SCREEN

FUNZIONE: Riporta il formato dello schermo

INDIRIZZO: \$FFED - 65517

REGISTRI DI COMUNICAZIONE: X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: X, Y

## DESCRIZIONE:

Questa routine restituisce il formato dello schermo di 40 colonne per 25 linee nei registri X e Y.  
E' utilizzata o puo' essere utilizzata per vedere che tipo di programma sta girando data la grande flessibilita' del CBM64 ad assumere anche altri sistemi operativi.

## MODO DI UTILIZZO

1) Chiamata della routine stessa



B-27 Nome della funzione: SECOND

FUNZIONE: Invia un indirizzo secondario per la funzione di ascolto ( LISTEN)

INDIRIZZO: \$FF93 - 65427

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE: LISTEN

ERRORI: Vedi READST

BYTES DI STACK USATI: 8

REGISTRI USATI: A

#### DESCRIZIONE:

Questa routine e' utilizzata per inviare un indirizzo secondario ad una periferica in I/O dopo che e' stata effettuata una chiamata alla routine LISTEN e quindi e' stato ordinato alla periferica di porsi in ascolto.

Questa routine non puo' essere usata per inviare un indirizzo secondario dopo un salto alla routine TALK (chiamata).

Normalmente un indirizzo secondario e' usato per comunicare il tipo di informazione che si desidera inviare alla periferica.

#### MODO DI UTILIZZO

1) Caricava l' Accumulatore con l' indirizzo secondario che deve essere inviato.

2) Eseguire quindi la routine

B-28 Nome della funzione: SETLFS

FUNZIONE: Fissa il file logico ( in maniera completa)

INDIRIZZO: \$FFBA - 65466

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI:

DESCRIZIONE:

Questa routine fissa il numero di file logico, l'indirizzo della periferica e l'indirizzo secondario per le altre routines.

Il numero di file logico e' usato dal sistema come chiave di riferimento alla tavola creata dalla routine OPEN file.

L'indirizzo della periferica puo' essere un numero dell'intervallo da 0 a 31.

I seguenti codici sono usati dal CBM64 per gli indirizzi di tabella:

| INDIRIZZO | PERIFERICA                |
|-----------|---------------------------|
| 0         | TASTIERA                  |
| 1         | DATASETTE                 |
| 2         | PERIFERICA RS232C         |
| 3         | SCHERMO                   |
| 4         | STAMPANTE SUL BUS SERIALE |
| 5         | DISCO CBM SUL BUS SERIALE |

I numeri di periferica uguali o maggiori di 4 si riferiscono a periferiche sul bus seriale.

Un comando alla periferica e' inviato come un indirizzo secondario al bus seriale dopo che il numero della stessa periferica e' stato inviato durante la sequenza di HANDSHAKING ATTENTION.

Se non viene inviato nessun indirizzo secondario, allora il registro Y dovrebbe essere messo a 255.

## MODO DI UTILIZZO

- 1) Carica l' accumulatore con il numero di file logico.
- 2) Carica il registro X con il numero di device
- 3) Carica il registro Y con il comando.

B-29 Nome della funzione: SETMSG

FUNZIONE: Controllo dei messaggi di sistema in uscita

INDIRIZZO: \$FF90 - 65420

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI: A

DESCRIZIONE:

Questa routine controlla la stampa di errore ed i messaggi di controllo delle Kernal routines.

Sia la stampa dei messaggi di errore come la stampa dei messaggi di controllo possono essere selezionate, cioè scelte, fissando l' accumulatore quando viene chiamata la routine.

FILE NOT FOUND e' un esempio di messaggio d' errore.

PRESS PLAY ON TAPE e' un esempio di messaggio di controllo.

I bits 6 e 7 di questo valore determinato da dove viene il messaggio.

Se il bit 7 e' a 1, allora verra' stampato un messaggio di errore proveniente dalle KERNAL.

Se il bit 6 e' a 1, viene stampato un messaggio di controllo.

MODO DI UTILIZZO

- 1) Fissare l' accumulatore al valore desiderato.
- 2) Chiamare in funzione la routine

B-30 Nome della funzione: SETNAM

FUNZIONE: Fissa il nome del file

INDIRIZZO: \$FFBD - 65496

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:

REGISTRI USATI:

DESCRIZIONE:

Questa routine e' utilizzata per fissare il nome del file per le routine di OPEN, SAVE e LOAD.

L' Accumulatore deve essere caricato con la lunghezza del nome del file.

I registri X e Y devono essere caricati con l' indirizzo del nome del file secondo il formato 6510 cioe' prima il byte basso e poi il byte alto.

L' indirizzo puo' essere un qualsiasi proponibile indirizzo di memoria del sistema dove sia appunto immagazzinata una stringa di caratteri che e' il nome del file.

Se non si desidera nessun nome, allora l' Accumulatore deve essere messo a 0 che rappresenterà un file di lunghezza zero.

In questo caso i registri X e Y possono essere fissati ad un qualsiasi indirizzo di memoria.

#### MODO DI UTILIZZO

1) Carica l' Accumulatore con la lunghezza del nome del file.

2) Carica il registro indice X con la parte bassa dell' indirizzo del nome del file

2) Carica il registro indice Y con la parte alta dell' indirizzo del nome del file

3) Esegui SETNAM

B-31 Nome della funzione: SETTIM

FUNZIONE: Fissa i valori del clock di sistema

INDIRIZZO: \$FFDB - 65499

REGISTRI DI COMUNICAZIONE: A, X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI:

## DESCRIZIONE:

L' orologio di sistema e' mantenuto da una routine di interrupt che lo aggiorna ogni sessantesimo di secondo (un "JIFFY" o ciclo ).

Il sistema di clock occupa 3 Bytes che da una capacita' di contare fino a 5.184.000 cicli ( o JIFFY) per un totale di 24 ore dopo di che l' orologio torna a zero.

Prima di chiamare questa routine che serve per modificare i valori dell ' orologio l' accumulatore deve contenere il Byte piu'. significativo del tempo fissato nell' orologio, il registro X il secondo Byte ed il registro Y il Byte meno significativo.

## MODO DI UTILIZZO

- 1) Carica l' Accumulatore con MSB del numero di 3 byte.
- 2) Carica il regsitro X con il Byte successivo.
- 3) Carica il registro Y con LSB.
- 4) Esegui ora la routine.

B-32 Nome della funzione: SETTMO

FUNZIONE: Fissa`il flag di fuori tempo ( TIME-OUT) sul bus IEEE

INDIRIZZO: \$FFA2 - 65442

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI: 2

REGISTRI USATI:

DESCRIZIONE:

Questa routine fissa il flag di Fuori tempo per la IEEE. Quando questo flag e' messo a 1 il CBM64 attendera' una risposta da una periferica sulla IEEE per 64 millisecondi.

Se la periferica non rispondera' al segnale DAV (cioe' DATA ADDRESS VALID) entro questo tempo allora il CBM64 riconoscerà una condizione di errore ed abbandonerà la sequenza di HANDSHAKE.

Quando questa routine e' chiamata ed il bit 7 dell' accumulatore contiene uno 0 allora il TIMEOUT e' abilitato, mentre un 1 nello stesso bit dell' accumulatore lo disabilita.

NOTA

Questa routine e' usata solo con il cartridge della IEEE. Il CBM64 usa la possibilita' del TIMEOUT per riconoscere che un file su disco, sempre collegato alla IEEE, non e' stato trovato

MODO DI UTILIZZO

Per fissare il flag di TIMEOUT:

1) Mettere a 0 il bit 7 dell' accumulatore.



2) Eseguire la routine

Per resettare il flag di TIMEOUT

1) Mettere a 1 il bit 7 dell' accumulatore.

2) Eseguire la routine

B-33 Nome della funzione: STOP

FUNZIONE: Controlla se il tasto di STOP e' premuto.

INDIRIZZO:\$FFE1 - 65505

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:

REGISTRI USATI: A, X

DESCRIZIONE:

Se il tasto di STOP era premuto durante la chiamata alla routine UDTIM, la chiamata a questa routine mette a 1 il flag Z.

Per di piu' i canali saranno resettati per mancanza di valori, mentre tutti gli altri flags rimarranno immutati. Se il tasto di STOP non era premuto allora l' Accumulatore conterra' un Byte che rappresenta l' ultima

riga della scansione di tastiera.

L'utente con questo metodo può anche controllare alcuni altri tasti.

#### MODO DI UTILIZZO

1) Prima di questa routine dovrebbe essere chiamata la routine UDTIM

2) Chiamare ora STOP

3) Controllare il flag Z.

B-34 Nome della funzione: TALK

FUNZIONE: Comando ad una periferica sul BUS seriale di TALK.

INDIRIZZO:\$FFB4 - 65460

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI: Vedi READST

BYTES DI STACK USATI:8

REGISTRI USATI: A

#### DESCRIZIONE:

Per utilizzare questa routine per prima cosa l'

Accumulatore deve essere caricato con un numero di periferica fra 0 e 31.

Quando e' chiamata questa routine, allora viene eseguito un OR logico bit per bit per convertire il numero della periferica in un indirizzo di chiamata.

Quindi questi dati saranno trasmessi come comando sul Bus seriale.

## MODO DI UTILIZZO

- 1) Carica l' accumulatore con il numero di periferica.
- 2) Salta alla routine.

B-35 Nome della funzione: TKSA

FUNZIONE: Invia un indirizzo secondario ad una periferica dopo la routine TALK

INDIRIZZO:\$FF96 - 65430

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE: TALK

ERRORI:Vedi READST

BYTES DI STACK USATI: 8

REGISTRI USATI: A

DESCRIZIONE:

Questa routine trasmette un indirizzo secondario ad una periferica in attesa di TALK sul bus seriale.

Prima di chiamarla deve esserci un numero fra 0 e 31 nell' accumulatore.

La routine invia questo numero come un comando di indirizzo secondario sul bus seriale.

TKSA puo' essere messa in funzione dopo la chiamata a TALK mentre non operera' dopo una routine o un comando di LISTEN.

#### MODO DI UTILIZZO

- 1) Utilizzare per prima TALK
- 2) Caricare l' accumulatore con l' indirizzo secondario.
- 3) Eseguire TKSA

B-36 Nome della funzione: UDTIM

FUNZIONE: Incrementa l' orologio del sistema

INDIRIZZO:\$FFEA - 65514

REGISTRI DI COMUNICAZIONE: A

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:2

REGISTRI USATI: A, X

## DESCRIZIONE:

Questa routine incrementa l' orologio del sistema. Normalmente questa routine e' chiamata dalla normale routine KERNAL di interrupt ogni sessantesimo di secondo. Se l' utente si programma da se gli interrupt questa routine DEVE essere chiamata per incrementare il temporizzatore.

Per di piu',se il tasto STOP e' funzionante, cioe' non e' stato disabilitato, deve essere chiamata anche la routine di STOP che abbiamo visto prima.

## MODO DI UTILIZZO

- 1) Eseguire la routine alle condizioni dette.

B-37 Nome della funzione: UNLSN

FUNZIONE: Invia un comando di UNLISTEN

INDIRIZZO:\$FFAE - 65454

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI:Vedi READST

BYTES DI STACK USATI:8

REGISTRI USATI: A

## DESCRIZIONE:

Questa routine ordina a tutti le periferiche sul bus seriale di fermare la ricezione dei dati dal CBM64.

In altre parole chiamando questa routine viene inviato un comando di UNLISTEN sul bus seriale. Cio' naturalmente avra' effetto solo sulle periferiche alle quali era stato in precedenza inviato un comando di LISTEN.

Di norma questa routine e' usata dopo che il CBM64 ha terminato di inviare dati alle periferiche esterne.

Ricordiamo che inviando un comando di UNLISTEN il bus seriale viene reso disponibile per altri usi dal momento che le periferiche aperte sono escluse.

## MODO DI UTILIZZO

1) Eseguire il salto alla routine.

B-38 Nome della funzione: UNTLK

FUNZIONE: Invia un comando di UNTALK

INDIRIZZO:\$FFAB - 65451

REGISTRI DI COMUNICAZIONE:

ROUTINES DI PREPARAZIONE:

ERRORI: Vedi READST

BYTES DI STACK USATI: 8

REGISTRI USATI: A

## DESCRIZIONE:

Come la precedente solo che invia un messaggio di UNTALK. Naturalmente anche in questo caso avremo una disabilitazione delle periferiche dal bus seriale .

## MODO DI UTILIZZO

1) Salto alla routine

B-39 Nome della funzione: VECTOR

FUNZIONE: Manipola i vettori su RAM

INDIRIZZO:\$FF8D - 65421

REGISTRI DI COMUNICAZIONE: X, Y

ROUTINES DI PREPARAZIONE:

ERRORI:

BYTES DI STACK USATI:2

REGISTRI USATI: A, X, Y

## DESCRIZIONE:

Questa routine manipola tutti i sistemi di indirizzi di salto vettorizzati immagazzinati in RAM. Chiamando questa routine con il bit di carry dell' accumulatore a 1, l' attuale contenuto dei vettori della RAM viene immagazzinato in una lista a cui puntano X e Y.

Chiamando invece questa routine con lo stesso bit a 0, una lista dell' utente indirizzata dal contenuto dei registri X e Y e' trasferita nel sistema dei vettori RAM

## NOTA

Questa routine richiede cautela nell' uso.

Il miglior sistema e' quello di immettere il contenuto dei vettori in un' area di memoria preventivamente scelta, cmabiare quello che si desidera e solo allora copiare il contenuto nell' area RAM.

## MODO DI UTILIZZO

Leggere i vettori RAM

- 1) Mettere a 1 il carry
- 2) Fissare i registri X e Y
- 3) Eseguire la routine

Caricamento dei vettori

- 1) Mettere a 0 il bit di carry
- 2) Fissare i registri Y e X
- 3) Eseguire la routine.

## MESSAGGI DI ERRORE

La seguente e' una lista dei messaggi di errore che possono capitare quando si eseguono delle routine Kernal.



Ricordiamo che se si incappa in un errore durante l'esecuzione di una di queste routine il bit di carry dell'Accumulatore viene messo a 1 ed il numero del messaggio di errore viene immesso nell' accumulatore. Pero' alcuni routines non usano questi messaggi o codici d' errore ma quelli, gia' visti della routine READST.

| NUMERO | SIGNIFICATO   |
|--------|---|
| 0      | Routine finita con tasto STOP                           |
| 1      | Troppi files aperti contemporaneamente                  |
| 2      | File gia' aperto  |
| 3      | File non aperto   |
| 4      | File non trovato  |
| 5      | Periferica non presente                                 |
| 6      | Il File non e' in input                                 |
| 7      | Il file non e' in output                                |
| 8      | Manca il nome del file                                  |
| 9      | Numero della periferica illogico                        |
| 240    | Punto massimo della memoria. Cambiare il buffer RS-232. |

**M A P P E**  
**D I**  
**M E M O R I A**

| LABEL  | INDIRIZZI |       | DESCRIZIONE  |
|--------|-----------|-------|--|
| D6510  | 0000      | 0     | Registro di direzione dati del 6510                      |
| R6510  | 0001      | 1     | "      ingresso uscita del 6510                          |
|        | 0002      | 2     | Non usato  |
| ADRAY1 | 0003-0004 | 3-4   | Vettore di salto.Conv da Floating in intero              |
| ADRAY2 | 0005-0006 | 5-6   | Vettore di salto.Conv da intero a Floating               |
| CHARAC | 0007      | 7     | Ricerca carattere.Imm. temp. durante INT                 |
| ENDCHR | 0008      | 8     | Flag di ricerca apici a fine stringa                     |
| TRMPOS | 0009      | 9     | Colonna video dopo l' ultimo TAB                         |
| VERCK  | 000A      | 10    | Flag: se 0=LOAD se 1=Verify                              |
| COUNT  | 000B      | 11    | Puntatore Buffer Ingresso/N. indici                      |
| DIMFLG | 000C      | 12    | Flag per mancanza dimens matrici.                        |
| VALTYP | 000D      | 13    | Flag categ. dati \$FF=Stringa \$00=Numerico              |
| INTFLG | 000E      | 14    | Flag categ. dati \$80=Intero \$00=Virgola mobile         |
| GARBFL | 000F      | 15    | Flag scans DATA/LIST/ Garbage Colle.                     |
| SUBFLG | 0010      | 16    | Flag Indice rifer./Chiamata funz. Utente                 |
| INPFLG | 0011      | 17    | Flag per ingresso dati: \$00=INPUT \$40=GET<br>\$98 READ |
| TANSGN | 0012      | 18    | Flag per segno TAN/confronto risultati                   |
| CHANNL | 0013      | 19    | Flag per INPUT   |
| LINNUM | 0014-0015 | 20-21 | Temp: Valore intero                                      |
| TEMPPT | 0016      | 22    | Puntatore tempo. allo stack di stringa                   |
| LASTPT | 0017-0018 | 23-24 | Ultimo indirizzo, temporaneo, di stringa.                |
| TEMPST | 0019-0021 | 25-33 | Stack temporaneo per stringa                             |
| INDEX  | 0022-0025 | 34-37 | Area di utilizzo puntatori<br>di cui                     |
| INDEX1 | 0022-0023 | 34-35 | Primo puntatore  |
| INDEX2 | 0024-0025 | 36-37 | Secondo puntatore  |
| RSHO   | 0026-002A | 38-42 | Risultato in virgola mobile di molt. e divisione         |
| TXTTAB | 002B-002C | 43-44 | Puntatore: inizio programma Basic                        |
| VARTAB | 002D-002E | 45-46 | Puntatore: inizio variabili Basic                        |

| LABEL  | INDIRIZZI |       | DESCRIZIONE  |
|--------|-----------|-------|--|
| ARYTAB | 002F-0030 | 47-48 | Puntatore: inizio matrici del Basic                      |
| STREND | 0031-0032 | 49-50 | Puntatore: fine matrici (+1)                             |
| FRETOP | 0033-0034 | 51-52 | Puntatore della fine zona immag.stringhe                 |
| FRESPC | 0036-0036 | 53-54 | Puntatore di utilita' generale per le stringhe           |
| MEMSIZ | 0037-0038 | 55-56 | Puntatore indirizzo piu' alto usato dal Basic.           |
| CURLIN | 0039-003A | 57-58 | Numero linea che il Basic sta trattando.                 |
| OLDLIN | 003B-003C | 59-60 | Numero linea che il Basic trattava prima dell' attua.    |
| OLDTXT | 003D-003E | 61-62 | Puntatore Basic per comando CONT                         |
| DATLIN | 003F-0040 | 63-64 | Numero di linea in cui e' l' attuale DATA                |
| DATPTR | 0041-0042 | 65-66 | Puntatore dell' indirizzo del DATA attuale               |
| INPPTR | 0043-0044 | 67-68 | Vettore per la routine di INPUT                          |
| VARNAM | 0045-0046 | 69-70 | Nome dell' attuale variabile                             |
| VARPNT | 0047-0048 | 71-72 | Puntatore dell' attuale variabile                        |
| FORPNT | 0049-004A | 73-74 | Puntatore della variabile indice per FOR-NEXT            |
| VARTXT | 004B-004C | 75-76 | Immag. temp. per rout. TXTPTR durante READ,INPUT,GET     |
| OPMASK | 004D      | 77    | MASK usata durante FRMEVL                                |
| TEMPF3 | 004E-0052 | 78-82 | Immagazzinam temp. per valore FLPT                       |
| FOUR6  | 0053      | 83    | Lunghezza variabile stringa durante GARBAGE              |
| COLLEC |           |       |  |
| JMPER  | 0054-0056 | 84-86 | Vett.salto usato in funz.JMP(\$4C) seguita da indir.     |
| TEMPF1 | 0057-005B | 87-91 | Imm. temporaneo per valore FLPT                          |
| TEMPF2 | 005C-0060 | 92-96 | Come sopra   |
| FAC    | ----      |       | zona dell' accumulatore 1 per i valori in virgola mobile |
| FACEXP | 0061      | 97    | FAC esponente  |

| LABEL  | INDIRIZZI         | DESCRIZIONE  |
|--------|-------------------|--|
| FACHO  | 0062-0065 98-101  | FAC mantissa   |
| FACSGN | 0066 102          | FAC segno  |
| SGNFLG | 0067 103          | Puntatore: costante valutazione successiva               |
| BITS   | 0068 104          | Indicatore di OVERFLOW per FAC 1                         |
| AFAC   | ----              | zona dell' Accumulatore 2 per i valori in virgola mobile |
| ARGEXP | 0069 105          | AFAC esponente   |
| ARGHO  | 006A-006D 106-109 | AFAC mantissa  |
| ARGSGN | 006E 110          | AFAC segno   |
| ARISGN | 006F 111          | Risultato confronto segni fra FAC e AFAC                 |
| FACOV  | 0070 112          | Arrotondamento del valore di FAC                         |
| FBUFPT | 0071-0072 113-114 | Puntatore: Buffer di cassetta                            |
| CHRGOT | 0073-008a 115-138 | Subrout.: riceve il prossimo Byte dal testo Basic.       |
| CHRGOT | 0079 121          | Entrata per ottenere ancora lo stesso byte d. testo      |
| TXTPTR | 007A-007B 122-123 | Puntatore: Byte attuale del testo Basic                  |
| RNDX   | 008B-008F 139-143 | Funzione RND mobile                                      |
| STATUS | 0090 144          | Status W. per rout. Ker. in I/O                          |
| STKEY  | 0091 145          | Flag per tasti STOP e RVS                                |
| SVXT   | 0092 146          | Costante di tempo per il nastro.                         |
| VERCK  | 0093 147          | Flags: 0=LOAD 1=VERIFY                                   |

| LABEL  | INDIRIZZI |         | DESCRIZIONE   |
|--------|-----------|---------|---|
| C3PO   | 0094      | 148     | Flag: caratt. in uscita sul B. seriale bufferizzato.  |
| BSOUR  | 0095      | 149     | Carattere memorizzato per uscita su B. seriale  |
| SYNO   | 0096      | 150     | Carattere di sincronizz. da cassetta(vedi ST)   |
| TEMPX  | 0097      | 151     | Immag temp. di X durante l'esecuzione di CHRIN  |
| TEMPY  | 0097      | 151     | Immag.temp. di Y durante RS-232   |
| LDTND  | 0098      | 152     | N. files aperti/indice tavola Files   |
| DFLNT  | 0099      | 153     | Standard ingresso dati (0= tastiera)  |
| DFLTO  | 009A      | 154     | Standard uscita dati (CMD) (3=tastiera)   |
| PRTY   | 009B      | 155     | Carattere di parita' del nastro   |
| DPSW   | 009C      | 156     | Flag: ricevuto Byte da nastro   |
| MSGFLG | 009D      | 157     | Flag:\$00=M.programma, soppressione messaggi di errore.\$40=M.Kernal invio errori.\$80=M.diretto,tutti gli errori |
| PTR1   | 009E      | 158     | Errore logico da nastro. Passo 1  |
| PTR2   | 009F      | 159     | Errore logico da nastro. Passo 2  |
| TIME   | 00A0-00A2 | 160-162 | Orologio int.(clock) in tempo reale.  |
| TSFCNT | 00A3      | 163     | Contat. per bit da nastro R. o W./Flag di EOI   |
| TBTCNT | 00A4      | 164     | Conteggio ciclo.  |
| CNTDN  | 00A5      | 165     | Conteggio rovescia di sincronizz. cassetta  |
| BUFPNT | 00A6      | 166     | Puntatore: Buffer di I/O nastro   |
| INBIT  | 00A7      | 167     | RS232 imm. temp ricez. bit/Temporizz. di cassetta.  |
| BITCI  | 00A8      | 168     | RS232 contat. ingresso bit/Temporizz. di cassetta   |
| RINONE | 00A9      | 169     | Flag per RS232: controllo bit di partenza   |
| RIDATA | 00AA      | 170     | Buffer ingresso Byte per RS232/Temporizz. cassetta  |
| RIPRTY | 00AB      | 171     | Input di parita' per RS232/Controllo cassetta   |
| SAL    | 00AC-00AD | 172-173 | Puntatore: Buffer del nastro/Scrolling di schermo   |

| LABEL  | INDIRIZZI |         | DESCRIZIONE  |
|--------|-----------|---------|--|
| EAL    | 00AE-00AF | 174-175 | Indirizzo di fine nastro/Fine del programma          |
| CMPO   | 00B0-00B1 | 176-177 | Costanti di temporizzazione nastro                   |
| TAPE1  | 00B2-00B3 | 178-179 | Puntatore: inizio buffer nastro                      |
| BITTS  | 00B4      | 180     | Cont. bit uscita per RS232/Flag tempor. lett. nastro |
| NXTBIT | 00B5      | 181     | Pross. bit da invi. a RS232/Lett.nastro per EOT      |
| RODATA | 00B6      | 182     | Buffer per Byte in uscita su RS232                   |
| FNLEN  | 00B7      | 183     | N. di caratteri nel File                             |
| LA     | 00B8      | 184     | N. dell' attuale File logico                         |
| SA     | 00B9      | 185     | Attuale indirizzo secondario                         |
| FA     | 00BA      | 186     | Attuale n. di periferica                             |
| FNADR  | 00BB-00BC | 187-188 | Puntatore al nome del file attuale                   |
| ROPRTY | 00BD      | 189     | Parita' di uscita per RS232/Temp. di cassetta        |
| FSBLK  | 00BE      | 190     | Contatore dei blocchi in I/O per nastro              |
| MYCH   | 00BF      | 191     | Buffer per parola seriale                            |
| CAS1   | 00C0      | 192     | Interruttore per motore cassetta                     |
| STAL   | 00C1-00C2 | 193-194 | Indirizzo di partenza per I/O                        |
| MEMUSS | 00C3-00C4 | 195-196 | Temporizz. di LOAD per cassetta                      |
| LSTX   | 00C5      | 197     | Valore ultimo tasto premuto \$40=nessun tasto        |
| NDX    | 00C6      | 198     | N. di caratteri nella coda (BUFFER-QUEUE) tastiera   |
| RVS    | 00C7      | 199     | Flag per REVERSE \$01=on \$00=off                    |
| INDX   | 00C8      | 200     | Puntatore: fine linea per input                      |
| LXSP   | 00C9-00CA | 201-202 | Posizioni X-Y cursore a inizio INPUT                 |
| SFDX   | 00CB      | 203     | Flag: Stampa caratteri shiftati.                     |
| BLNSW  | 00CC      | 204     | Lampeggio cursore \$00=abilitato \$01=disabilitato   |
| BLNTC  | 00CD      | 205     | Temporizz. di conto alla rovescia per lamp. cursore  |
| GDBLN  | 00CE      | 206     | Carattere presente sotto il cursore                  |
| BLNON  | 00CF      | 207     | Flag: Stato del cursore \$00=off \$01=on             |

| LABEL  | INDIRIZZI |         | DESCRIZIONE   |
|--------|-----------|---------|---|
| CRSW   | 00D0      | 208     | Flag:ingresso da tastiera di GET o INPUT              |
| PNT    | 00D1-00D2 | 209-210 | Puntatore all' attuale linea di indirizzo schermo.    |
| PNTR   | 00D3      | 211     | Posiz del cursore sulla colonna dell'attuale linea.   |
| QTSW   | 00D4      | 212     | Indicatore se cursore fra apici. \$00=no              |
| LNMX   | 00d5      | 213     | Lunghezza dell' attuale linea di schermo              |
| TBLX   | 00D6      | 214     | Attuale linea di schermo del cursore                  |
| SCHAR  | 00D7      | 215     | Valore dell' attuale car. input/Ultimo caratt. uscito |
| INSRT  | 00D8      | 216     | Flag:n.volte di att. tasto INST                       |
| LDTB1  | 00D2-00F2 | 217-242 | Tavola di collegam. linee schermo/Immag.temp Editor   |
| USER   | 00F3-00F4 | 243-244 | Puntatore per attuale colore di schermo in RAM        |
| KEYTAB | 00F5-00F6 | 246-246 | Vettore alla tavola di decodifica tastiera            |
| RIBUF  | 00F7-00F8 | 247-248 | Puntatore buffer ingresso RS232                       |
| ROBUF  | 00F9-00FA | 249-250 | Puntatore buffer uscita RS232                         |
| FREKZP | 00FB-00FE | 251-254 | Spazio per utente in pag.0                            |
| BASZPT | 00FF      | 255     | Area per dati Basic                                   |
| ASCWRK | 00FF-010A | 255-266 | Area assem. per conversione FAC in ASCII              |
| BAD    | 0100-013E | 256-318 | Errore logico ingresso da nastro                      |
| STACK  | 0100-01FF | 256-*** | Area di Stack Hardware per 6510                       |
| BSTACK | 013F-01FF | 319-*** | Area di stack per il basic.                           |
| BUF    | 0200-0258 | 512-600 | Buffer input di sistema                               |
| LAT    | 0259-0262 | 601-610 | Tavola KERNAL: n. di F.logico attivo                  |
| FAT    | 0263-026C | 611-620 | Tavola KERNAL: n. di periferica per ogni file         |
| SAT    | 026D-0276 | 621-630 | Tavola KERNAL: indirizzo seconadario di ogni file     |
| KEYD   | 0277-0280 | 631-640 | Coda del Buffer di tastiera (metodo FIFO)             |
| MEMSTR | 0281-0282 | 641-642 | Puntatore al punto piu' basso memoria per S.O.        |



| LABEL  | INDIRIZZI |         | DESCRIZIONE   |
|--------|-----------|---------|---|
| MEMSIZ | 0283-0284 | 643-644 | Puntatore al massimo memoria per S.O.   |
| TIMOUT | 0285      | 645     | Flag: variabile KERNAL per fuoritempo IEEE  |
| COLOR  | 0286      | 646     | Codice dell' attuale colore del carattere   |
| GDCOL  | 0287      | 647     | Colore di fondo sotto il cursore  |
| HIBASE | 0288      | 648     | Byte alto indirizzo memoria di schermo  |
| XMAX   | 0289      | 649     | Massimo n. di Byte nel Buffer di tastiera   |
| RPTFLG | 028A      | 650     | Flag di repeat\$00=cursore, inst, spazio/ \$40 =<br>nessun tasto/\$80=tutti i tasti |
| KOUNT  | 028B      | 651     | Contatore di velocita' di repeat  |
| DELAY  | 028C      | 652     | Contatore del ritardo di repeat   |
| SHFLAG | 028D      | 653     | Flag tasto SHIFT (vedi nota 1)  |
| LSTSHF | 028E      | 654     | Ultimo tasto shift premuto  |
| KEYLOG | 028F-0290 | 655-656 | Routine che determina tavola di tastiera per<br>SHIFT                               |
| MODE   | 0291      | 657     | Flag \$00=abilita SHIFT \$80= disabilita  |
| AUTODN | 0292      | 658     | Flag scorrimento (SCROLLING) video  |
| M51CTR | 0293      | 659     | RS232 registro di controllo immagine d.6551   |
| M51CDR | 0294      | 660     | RS232 registro di comando immagine 6551   |
| M51AJB | 0295-0296 | 661-662 | RS232 BPS non standard (USA)  |
| BITNUM | 0298      | 664     | RS232 N. di Bits da inviare   |
| BAUDOF | 0299-029A | 665-666 | RS232 Baud rate   |
| RIDBE  | 029B      | 667     | RS232 indice della fine Buffer di Input   |
| RIDBS  | 029C      | 668     | RS232 inizio del Buffer di input  |
| RODBS  | 029D      | 669     | RS232 Inizio del Buffer di output   |
| RODBE  | 029E      | 670     | RS232 indice fine Buffer di output  |
| IRQTMP | 029F-02A0 | 671-672 | Imm. temp. vettore IRQ durante oper. su nastro                                      |
| ENABL  | 02A1      | 673     | RS232 Abilitazione  |
| TODSN  | 02A2      | 674     | Controllo sens. (TOD) cassetta durante operaz.                                      |

| LABEL  | INDIRIZZI |         | DESCRIZIONE   |
|--------|-----------|---------|---|
|        |           |         | I/O   |
| TRDTMP | 02A3      | 675     | Immag temp. per lettura cassetta                      |
| TD1IRQ | 02A4      | 676     | Indic. temp D1 IRQ durante lettura cassetta           |
| TLNIDX | 02A5      | 677     | Indic. temp per incice linea schermo                  |
| TVSFLG | 02A6      | 678     | Flag per telev \$00= NTSC \$01= PAL                   |
| -----  | 02A7-02FF | 679-767 | Area non usata attualmente                            |
| IERROR | 0300-0301 | 768-769 | Vettore: stampa messaggi errore Basic conten (\$E38B) |
| IMAIN  | 0302-0303 | 770-771 | Vettore: ingr. indir. per lin.I Basic conten (\$A483) |
| ICRNCH | 0304-0305 | 772-773 | Vettore: ingr. indir. per Rout. TOKEN conten (\$A57C) |
| IQPLOP | 0306-0307 | 774-775 | Vettore alla routine LIST contenente normalm (\$A71A) |
| IGONE  | 0308-0309 | 776-777 | Vettore per la ricerca parole chiave conten (\$A7E4)  |
| IEVAL  | 030A-030B | 778-779 | Vettore valut comandi Basic conten (\$AE86)           |
| SAREG  | 030C      | 780     | Immagazz. di A durante l' esecuzione di SYS           |
| SXREG  | 030D      | 781     | Immagazz. di X durante l' esecuzione di SYS           |
| SYREG  | 030E      | 782     | Immagazz. di Y durante l' esecuzione di SYS           |
| SPREG  | 030F      | 783     | Immagazz. di SP durante l' esecuzione di SYS          |
| USRPOK | 0310      | 784     | Funzione USR per JMP (\$4C)                           |
| USRADD | 0311-0312 | 785-786 | Indirizzo USR Byte basso/Byte alto                    |
| ----   | 0313      | 787     | Non usato   |
| CINV   | 0314-0315 | 788-789 | Vettore Hardware per indirizzo IRQ conten (\$EA31)    |

| LABEL  | INDIRIZZI         | DESCRIZIONE  |
|--------|-------------------|--|
| CBINV  | 0316-0317 790-791 | Vettore indirizzo BRK contenuto normale (\$FE66)   |
| NMINV  | 0318-0319 792-793 | Vettore Hardware per indirizzo NMI conten (\$FE47) |
| IOPEN  | 031A-031B 794-795 | Vettore per la routine di OPEN (\$F34A)            |
| ICLOSE | 031C-031D 796-797 | Vettore per la routine di CLOSE (\$F291)           |
| ICKIN  | 031E-031F 798-799 | Vettore per la routine di CHKIN (\$F20E)           |
| ICKOUT | 0320-0321 800-801 | Vettore per la routine di CKOUT (\$F250)           |
| ICLRC  | 0322-0323 802-803 | Vettore per la routine di CLRCHN (\$F333)          |
| IBASIN | 0324-0325 804-805 | Vettore per la routine di CHRIN (\$F157)           |
| IBSOUT | 0326-0327 806-807 | Vettore per la routine di CHROUT (\$F1CA)          |
| ISTOP  | 0328-0329 808-809 | Vettore per la routine di STOP (\$F6ED)            |
| IGETIN | 032A-032B 810-811 | Vettore per la routine di GETIN (\$F13E)           |
| ICLALL | 032C-032D 812-813 | Vettore per la routine di CLALL (\$F32F)           |
| USRCMD | 032E-032F 814-815 | Vettore definito dall' utente di solito (\$FE66)   |
| ILOAD  | 0330-0331 816-817 | Vettore per la routine di LOAD (\$F4A5)            |
| ISAVE  | 0332-0333 818-819 | Vettore per la routine di SAVE (\$F5ED)            |
| -----  | 0334-03FB 820-827 | Non usato normalmente                              |

| LABEL   | INDIRIZZI             | DESCRIZIONE   |
|---------|-----------------------|---|
| TBUFFER | 033C-03FB 829-1019    | Buffer del nastro per operazioni di I/O               |
| -----   | 03FC-03FF 1020-1023   | Normalmente non usato                                 |
| VICSCN  | 0400-07FF 1024-2047   | Area di memoria dello schermo                         |
|         | 0800-9FFF 2048-40959  | Area normale per i programmi Basic                    |
|         | 8000-9FFF 32768-40959 | Spazio opzionale per ROM del cartridge<br>(8192-Byte) |
|         | A000-BFFF 40960-49151 | ROM Basic o 8K-RAM                                    |
|         | C000-CFFF 49152-53247 | RAM per 4096 Bytes                                    |
|         | D000-DFFF 53248-57334 | Sistema operativo, RAM alter., ecc                    |
|         | E000-FFFF 57344-65535 | KERNAL ROM o 8K RAM.                                  |

#### NOTA

Data la grande flessibilita' di utilizzo della memoria del CBM64 e' necessario riferirsi al relativo capitolo per quanto riguarda la disposizione da \$8000 a \$FFFF

**M A P P E**  
**D E G L I**  
**I N T E G R A T I**  
**V I C E C I A**

# VIC- VIDEO INTERFACE CONTROLLER

(MOS 6566)

Da 53248 a 54241 (\$D000 a \$D02E)

| INDIRIZZI |       | FUNZIONI  |
|-----------|-------|---|
| ESA       | DEC   |   |
| D000      | 53248 | Posizione asse X dello Sprite 0                   |
| D001      | 53249 | Posizione asse Y dello Sprite 0                   |
| D002      | 53250 | Posizione asse X dello Sprite 1                   |
| D003      | 53251 | Posizione asse Y dello Sprite 1                   |
| D004      | 53252 | Posizione asse X dello Sprite 2                   |
| D005      | 53253 | Posizione asse Y dello Sprite 2                   |
| D006      | 53255 | Posizione asse X dello Sprite 3                   |
| D007      | 53256 | Posizione asse Y dello Sprite 3                   |
| D008      | 53257 | Posizione asse X dello Sprite 4                   |
| D009      | 53258 | Posizione asse Y dello Sprite 4                   |
| D00A      | 53259 | Posizione asse X dello Sprite 5                   |
| D00B      | 53260 | Posizione asse Y dello Sprite 5                   |
| D00C      | 53261 | Posizione asse X dello Sprite 6                   |
| D00D      | 53262 | Posizione asse Y dello Sprite 6                   |
| D00E      | 53263 | Posizione asse X dello Sprite 7                   |
| D00F      | 53264 | Posizione asse Y dello Sprite 7                   |
| D010      | 53264 | Posizione del MBS delle coord. X Sprites da 0 a 7 |

| INDIRIZZI  | BITS | FUNZIONI  |
|------------|------|---|
| D011 53265 |      | Registro di controllo dell' integrato VIC                   |
|            | 7    | Comparazione dei valori (vedi 53266)                        |
|            | 6    | Modo di estensione colori testo 1= abilitato                |
|            | 5    | Bit map mode 1= abilitato                                   |
|            | 4    | Colore del bordo 0=Blank                                    |
|            | 3    | Seleziona n. righe testo 1= 25 righe                        |
|            | 2-0  | Scorrimento lento alla posizione Y                          |
| D012 53266 |      | Lettura/scrittura immagine video/valore per confr IRQ       |
| D013 53267 |      | Posizione sull' asse X della Penna Luminosa                 |
| D014 53268 |      | Posizione sull' asse Y della Penna Luminosa                 |
| D015 53269 |      | Abilita (1) disabilita (0) gli Sprites (Bit0=Sprite 0, ecc) |
| D016 53270 |      | Registro di controllo del VIC                               |
|            | 7-6  | Non usati   |
|            | 5    | DEVE ESSERE SEMPRE A 0                                      |
|            | 4    | Modo multicolore 1= abilitato                               |
|            | 3    | Selettore n. colonne 1=40 colonne                           |
|            | 2-0  | Scorrimento lento alla posizione X                          |
| D017 53271 |      | Espansione Sprites rapp.2x su asse Y                        |
| D018 53272 |      | Registro di controllo della memoria del VIC                 |
|            | 7-4  | Indirizzo di base della matrice video                       |
|            | 3-1  | Indirizzo del punto relativo al carattere                   |
| D019 53273 |      | Registro dei Flags di interrupt (Bit 1= interrupt avvenuto) |

| INDIRIZZI  | BITS | FUNZIONI  |
|------------|------|---|
|            | 7    | Per qualsiasi condizione di Interrupt verificata =1 |
|            | 3    | Flag IRQ per penna luminosa                         |
|            | 2    | Flag IRQ per collisioni Sprite con Sprite           |
|            | 1    | Flag IRQ per collisione Sprite con lo sfondo        |
|            | 0    | Flag IRQ di comparazione video                      |
| D01A 53274 |      | Registro per maschera IRQ 1= Interrupt abilitato    |
| D01B 53275 |      | Priorita' di visualizzazione Sprite sullo sfondo    |
| D01C 53276 |      | Selezione del modo multicolore (MCM) per Sprites    |
| D01D 53277 |      | Espansione Sprites rapp.2x su asse X                |
| D01E 53278 |      | Controllo di collisione fra gli Sprites             |
| D01F 53279 |      | Controllo di collisione fra Sprite e sfondo         |
| D020 53280 |      | Colore del bordo                                    |
| D021 53281 |      | Colore 0 dello sfondo                               |
| D022 53282 |      | Colore 1 dello sfondo                               |
| D023 53283 |      | Colore 2 dello sfondo                               |
| D024 53284 |      | Colore 3 dello sfondo                               |
| D025 53285 |      | Registro 0 per Sprite in modo multicolore           |
| D026 53286 |      | Registro 1 per Sprite in modo multicolore           |
| D027 53287 |      | Registro di colore dello Sprite 0                   |
| D028 53288 |      | Registro di colore dello Sprite 1                   |
| D029 53289 |      | Registro di colore dello Sprite 2                   |
| D02A 53290 |      | Registro di colore dello Sprite 3                   |
| D02B 53291 |      | Registro di colore dello Sprite 4                   |
| D02C 53292 |      | Registro di colore dello Sprite 5                   |



| INDIRIZZI  | BITS | FUNZIONI                          |
|------------|------|-----------------------------------|
| D02D 53293 |      | Registro di colore dello Sprite 6 |
| D02E 53294 |      | Registro di colore dello Sprite 7 |

#### SID- SOUND INTERFACE DEVICE

(MOS 6581)

Da 54272 a 55295 (\$D400 a \$D7ff)

| INDIRIZZI |       | BITS | FUNZIONI   |
|-----------|-------|------|--|
| ESA       | DEC   |      |  |
| D400      | 54272 |      | Voce 1: controllo della frequenza Byte basso                 |
| D401      | 54273 |      | Voce 1: controllo della frequenza Byte alto                  |
| D402      | 54274 |      | Voce 1: ampiezza d' onda pulsante Byte basso                 |
| D403      | 54275 | 7-4  | Non usati  |
|           |       | 3-0  | Voce 1: ampiezza d' onda pulsante Nybble alto                |
| D404      | 54276 |      | Voce 1: registro di controllo                                |
|           |       | 7    | Selezione forma d' onda rumore bianco casuale 1=ON           |
|           |       | 6    | Selezione forma d' onda pulsante 1=ON                        |
|           |       | 5    | Selezione forma d' onda a dente di sega 1=ON                 |
|           |       | 4    | Selezione forma d' onda triangolo 1=ON                       |
|           |       | 3    | Controllo Bit se =1 disabilita l' oscillatore 1              |
|           |       | 2    | Modulazione suono oscill. 1 con oscill. 2 in uscita 1= ON    |
|           |       | 1    | Sincroniz. oscill.1 con frequen. oscill. 3 1=ON              |
|           |       | 0    | Bit di controllo 1= partenza Att/Dec/Sust/ 0= part. Release. |

| INDIRIZZI  | BITS | FUNZIONI   |
|------------|------|--|
| D405 54277 |      | Generatore Envelop 1: controllo di ciclo per Attack/Decay    |
|            | 7-4  | Selezione durata ciclo di Attack (da 0 a 15)                 |
|            | 3-0  | Selezione durata ciclo di Decay (da 0 a 15)                  |
| D406 54278 |      | Generatore Envelop 1: controllo di ciclo per Sustain/Release |
|            | 7-4  | Selezione durata ciclo di Sustain (da 0 a 15)                |
|            | 3-0  | Selezione durata ciclo di Release (da 0 a 15)                |
| D407 54279 |      | Voce 2: controllo della frequenza Byte basso                 |
| D408 54280 |      | Voce 2: controllo della frequenza Byte alto                  |
| D409 54281 |      | Voce 2: ampiezza d' onda pulsante Byte basso                 |
| D40A 54282 | 7-4  | Non usati  |
|            | 3-0  | Voce 2: ampiezza d' onda pulsante Nybble alto                |
| D40B 54283 |      | Voce 2: registro di controllo                                |
|            | 7    | Selezione forma d' onda rumore bianco casuale l=ON           |
|            | 6    | Selezione forma d' onda pulsante l=ON                        |
|            | 5    | Selezione forma d' onda a dente di sega l=ON                 |
|            | 4    | Selezione forma d' onda triangolo l=ON                       |
|            | 3    | Controllo Bit se =1 disabilita l' oscillatore 2              |
|            | 2    | Modulazione suono oscill. 2 con oscill. 1 in uscita l= ON    |
|            | 1    | Sincroniz. oscill.2 con frequen. oscill. 1 l=ON              |
|            | 0    | Bit di controllo l= partenza Att/Dec/Sust/ 0= part. Release. |
| D40C 54284 |      | Generatore Envelop 2: controllo di ciclo per Attack/Decay    |
|            | 7-4  | Selezione durata ciclo di Attack (da 0 a 15)                 |
|            | 3-0  | Selezione durata ciclo di Decay (da 0 a 15)                  |
| D40D 54285 |      | Generatore Envelop 2: controllo di ciclo per                 |

| INDIRIZZI  | BITS | FUNZIONI   |
|------------|------|--|
|            |      | Sustain/Release  |
|            | 7-4  | Selezione durata ciclo di Sustain (da 0 a 15)                |
|            | 3-0  | Selezione durata ciclo di Release (da 0 a 15)                |
| D40E 54286 |      | Voce 3: controllo della frequenza Byte basso                 |
| D40F 54287 |      | Voce 3: controllo della frequenza Byte alto                  |
| D410 54288 |      | Voce 3: ampiezza d' onda pulsante Byte basso                 |
| D411 54289 | 7-4  | Non usati  |
|            | 3-0  | Voce 3: ampiezza d' onda pulsante Nybble alto                |
| D412 54290 |      | Voce 3: registro di controllo                                |
|            | 7    | Selezione forma d' onda rumore bianco casuale l=ON           |
|            | 6    | Selezione forma d' onda pulsante l=ON                        |
|            | 5    | Selezione forma d' onda a dente di sega l=ON                 |
|            | 4    | Selezione forma d' onda triangolo l=ON                       |
|            | 3    | Controllo Bit se =1 disabilita l' oscillatore 3              |
|            | 2    | Modulazione suono oscill. 3 con oscill. 2 in uscita l= ON    |
|            | 1    | Sincroniz. oscill.3 con frequen. oscill. 2 l=ON              |
|            | 0    | Bit di controllo l= partenza Att/Dec/Sust/ 0= part. Release. |
| D413 54291 |      | Generatore Envelop 3: controllo di ciclo per Attack/Decay    |
|            | 7-4  | Selezione durata ciclo di Attack (da 0 a 15)                 |
|            | 3-0  | Selezione durata ciclo di Decay (da 0 a 15)                  |
| D414 54292 |      | Generatore Envelop 3: controllo di ciclo per Sustain/Release |
|            | 7-4  | Selezione durata ciclo di Sustain (da 0 a 15)                |
|            | 3-0  | Selezione durata ciclo di Release (da 0 a 15)                |

| INDIRIZZI  | BITS | FUNZIONI  |
|------------|------|---|
| D415 54293 |      | Filtro: frequenza di taglio Nybble basso bits 2-0     |
| D416 54294 |      | Filtro: frequenza di taglio Byte alto                 |
| D417 54295 |      | Controllo risonanza filtro/Controllo voce in ingresso |
|            | 7-4  | Selettore filtro di risonanza da 0 a 15               |
|            | 3    | Ingresso filtro esterno 1=si 0=no                     |
|            | 2    | Filtro voce 3 in uscita 1=si 0=no                     |
|            | 1    | Filtro voce 2 in uscita 1=si 0=no                     |
|            | 0    | Filtro Voce 1 in uscita 1=si 0=no                     |
| D418 54296 |      | Selettore modo filtro e volume                        |
|            | 7    | Taglio voce 3 in uscita 1=OFF 0=ON                    |
|            | 6    | Selettore filtro Passa-Alto 1=ON                      |
|            | 5    | Selettore filtro Passa-Banda 1=ON                     |
|            | 4    | Selettore filtro Passa-Basso 1=ON                     |
|            | 3-0  | Selettore uscita volume da 0 a 15                     |
| D419 54297 |      | Convertitore analogico digitale Paddle 1 valori 0-255 |
| D41A 54298 |      | Convertitore analogico digitale Paddle 2 valori 0-255 |
| D41B 54299 |      | Oscillatore 3: generatore numeri casuali              |
| D41C 54230 |      | Uscita generatore Envelop 3                           |

#### CIA 1 - COMPLEX INTERFACE ADAPTER

(MOS 6526)

Da 56320 a 54575 (\$DC00 a \$DCFF)

| INDIRIZZI  | BITS | FUNZIONI  |
|------------|------|---|
| DC00 56320 |      | Porta A dati per Tastiera, Joystick, Paddles, Light-Pen |
|            | 7-0  | Scrittura valori colonna tastiera per scansione stessa. |
|            | 7-6  | Lettura Paddles su porta A e B ( 01=porta A 10=Porta B) |
|            | 4    | Pulsante Joystick 1= Pulsante premuto                   |
|            | 3-2  | Pulsanti delle Paddles                                  |
|            | 3-0  | Direzione del Joystick A (valori da 0 a 15)             |
| DC01 56321 |      | Porta B dati per Tastiera, Joystick, Paddles            |
|            | 7-0  | Scrittura valori riga di tastiera per scansione stessa. |
|            | 7    | Temporizzatore B impulso/taglio in uscita               |
|            | 6    | Temporizzatore A impulso/taglio in uscita               |
|            | 4    | Pulsante Joystick 1= Pulsante premuto                   |
|            | 3-2  | Pulsante delle Paddles                                  |
|            | 3-0  | Pulsante delle Joysticks                                |
| DC02 56322 |      | Registro di direzione dati - Porta A (56320)            |
| DC03 56323 |      | Registro di direzione dati - Porta B (56321)            |
| DC04 56324 |      | Temporizzatore A : Byte Basso                           |
| DC05 56325 |      | Temporizzatore A : Byte alto                            |
| DC06 56326 |      | Temporizzatore B : Byte Basso                           |
| DC07 56327 |      | Temporizzatore B : Byte alto                            |
| DC08 56328 |      | Orologio : 1/10 di secondo                              |
| DC09 56329 |      | Orologio : misurazione dei secondi                      |
| DC0A 56330 |      | Orologio : misurazione dei minuti                       |
| DC0B 56331 |      | Orologio : ore. Il bit 7 indica se AM o PM              |
| DC0C 56332 |      | Buffer per dati seriali sincroni in I/O                 |

| INDIRIZZI  | BITS | FUNZIONI  |
|------------|------|---|
| DCOD 56333 |      | Registro di controllo interrupts del CIA                  |
| IRQ)       | 7    | Flag di IRQ ( Interrupt avvenuto)/ Set-Clear del Flag     |
|            | 4    | Flag di IRQ (Lettura cassetta/ingresso sul B. seriale di  |
|            | 3    | Interrupt su porta seriale                                |
|            | 2    | Interrupt per ALARM su orologio                           |
|            | 1    | Interrupt per temporizzatore B                            |
|            | 0    | Interrupt per temporizzatore A                            |
| DCOE 56334 |      | Registro di controllo A del CIA                           |
|            | 7    | Frequenza orologio : 1=50 Hz 0=60 Hz                      |
|            | 6    | Porta seriale di I/O: 1=Output 0=Input                    |
|            | 5    | Contatore temporizzatore A                                |
|            | 4    | Caricamento forzato del temporizz. A (1=attivo)           |
|            | 3    | Modo di RUN del temporizz. A (0=Continuo 1=passo-passo)   |
|            | 2    | Modo di uscita del temp. A su PB6 (1=Taglio 0=Impulso)    |
|            | 1    | Uscita Temp. A su PB6 1=On 0=Off                          |
|            | 0    | Partenza/Arresto del Temporizz. A (1=Start 0=Stop)        |
| DCOF 56335 |      | Registro di controllo B del CIA                           |
|            | 7    | Fissa sveglia orologio 1=ALARM                            |
|            | 6-5  | Selezione del modo del temporizzatore B (vn**)            |
|            | 4-0  | Stessi significati registro di controllo A per temporizz. |

## CIA 2 - COMPLEX INTERFACE ADAPTER

(MOS 6526)

Da 56576.a 56831 (\$DD00 a \$DDFF)

| INDIRIZZI |       | BITS | FUNZIONI  |
|-----------|-------|------|---|
| ESA       | DEC   |      |   |
| DD00      | 56575 |      | Porta A dati p.Bus seriale, RS-232, controllo memoria VIC |
|           |       | 7    | Input dati su Bus seriale                                 |
|           |       | 6    | Input impulso CLOCK su bus seriale                        |
|           |       | 5    | Output dati Bus seriale                                   |
|           |       | 4    | Output dati CLOCK su bus seriale                          |
|           |       | 3    | Output segnale ATN (ATTENTION) Bus seriale                |
|           |       | 2    | Output dati per RS232 (user port)                         |
|           |       | 1-0  | Selezione banchi di memoria del VIC (normale =11)         |
| DD01      | 56577 |      | Porta B dati per User port, RS232                         |
|           |       | 7-0  | User port/  |
|           |       | 7    | RS232 segnale di dati pronti                              |
|           |       | 6    | Funzione di CLEAR per invio su RS232                      |
|           |       | 4    | Controllo di invio su RS232                               |
|           |       | 3    | Indicatore di anello per RS232                            |

| INDIRIZZI  | BITS | FUNZIONI   |
|------------|------|--|
|            | 2    | DATA TERMINAL READY per RS232                            |
|            | 1    | Richiesta di invio per RS232                             |
|            | 0    | Ricezione dati per RS232                                 |
| DD02 56578 |      | Registro di direzione dati - Porta A                     |
| DD03 56579 |      | Registro di direzione dati - Porta B                     |
| DD04 56580 |      | Temporizzatore A : Byte Basso                            |
| DD05 56581 |      | Temporizzatore A : Byte alto                             |
| DD06 56582 |      | Temporizzatore B : Byte Basso                            |
| DD07 56583 |      | Temporizzatore B : Byte alto                             |
| DD08 56584 |      | Orologio : 1/10 di secondo                               |
| DD09 56585 |      | Orologio : misurazione dei secondi                       |
| DD0A 56586 |      | Orologio : misurazione dei minuti                        |
| DD0B 56587 |      | Orologio : ore. Il bit 7 indica se AM o PM               |
| DD0C 56588 |      | Buffer per dati seriali sincroni in I/O                  |
| DD0D 56589 |      | Registro di controllo interrupts del CIA                 |
| IRQ)       | 7    | Flag di NMI ( Interrupt avvenuto)/ Set-Clear del Flag    |
|            | 4    | Flag di NMI (Lettura cassetta/ingresso sul B. seriale di |
|            | 3    | Interrupt su porta seriale                               |
|            | 2    | Interrupt per ALARM su orologio                          |
|            | 1    | Interrupt per temporizzatore B                           |
|            | 0    | Interrupt per temporizzatore A                           |
| DD0E 56590 |      | Registro di controllo A del CIA                          |
|            | 7    | Frequenza orologio : 1=50 Hz 0=60 Hz                     |
|            | 6    | Porta seriale di I/O: 1=Output 0=Input                   |
|            | 5    | Contatore temporizzatore A                               |
|            | 4    | Caricamento forzato del temporizz. A (1=attivo)          |



| INDIRIZZI | BITS | FUNZIONI  |
|-----------|------|---|
|           | 3    | Modo di RUN del temporizz. A (0=Continuo 1=passo-passo)   |
|           | 2    | Modo di uscita del temp. A su PB6 (1=Taglio 0=Impulso)    |
|           | 1    | Uscita Temp. A su PB6 1=On 0=Off                          |
|           | 0    | Partenza/Arresto del Temporizz. A (1=Start 0=Stop)        |
| DDOF 5659 |      | Registro di controllo B del CIA                           |
|           | 7    | Fissa sveglia orologio 1=ALARM                            |
|           | 6-5  | Selezione del modo del temporizzatore B (vn**)            |
|           | 4-0  | Stessi significati registro di controllo A per temporizz. |

|            |             |                          |
|------------|-------------|--------------------------|
| -DE00-DEFF | 56832-57087 | Riservata per espansioni |
| -DF00-DFFF | 57088-57343 | Riservata per espansioni |

# **T A V O L E**

## TAVOLA ISTRUZIONI 6510

ADC Add memory to accumulator with carry.

Somma il contenuto della locazione di memoria specificata con il contenuto dell' Accumulatore sommando anche il bit di Carry.

Il risultato dell' operazione viene riportato nell' Accumulatore.

AND And memory with accumulator.

Esegue un' operazione di AND logico fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata.

ASL Shift left one bit

Esegue uno spostamento verso sinistra del contenuto dell' Accumulatore o della locazione di memoria specificata.

Il bit piu' significativo del contenuto di partenza viene riportato nel Carry.

Il bit che resterebbe vuoto viene riempito con ZERO.

BCC Branch on carry clear

Esegue il salto specificato se il bit di carry e uguale a ZERO.

BCS Branch on carry set

Esegue il salto specificato se il bit di Carry e a UNO.

BEQ Branch on result zero

Esegue il salto specificato se il bit del flag ZERO=1.

BIT Test bits in memory with accumulator.

Esegue un AND logico fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata. Il risultato dell' operazione non compare nei registri della macchina, pero' se il risultato e' =0 allora il bit ZERO dello Status viene messo a 1.

BMI Branch on result minus

Esegue il salto specificato se il bit del FLAG NEGATIVE e' uguale a 1.

BNE Branch on result not zero

Esegue il salto specificato se il bit del FLAG ZERO =0.

BPL Branch on result plus.

Esegue il salto specificato se il bit del FLAG NEGATIVE =0

BRK Force break

Forza un' interruzione non mascherabile tramite il bit I dello status.

Salva automaticamente il contenuto del PC. Mette a 1 il bit I dello Status.

BVC Branch on overflow clear

Esegue il salto specificato se il bit di OVERFLOW e' =0.

BVS Branch on Overflow set.

Esegue il salto specificato se il bit di OVERFLOW e' =1.

CLC Clear carry flag

Mette a =0 il bit di Carry.

CLD Clear decimal mode.

Mette a zero il bit del FLAG DECIMAL.

Le operazioni aritmetiche svolte successivamente a questa istruzione sono svolte in binario.

CLI Clear interrupt disable bit

Mette a zero il bit del FLAG di INTERRUPT. Dopo questa istruzione la CPU e' abilitata ad eseguire le interruzioni che dovessero arrivare.

CLV Clear overflow flag

Mette a 1 il bit del FLAG di OVERFLOW.

CMP Compare memory and Accumulator.

Confronta il contenuto della locazione di memoria specificata con il contenuto dell' Accumulatore.

CPX Compare memory and index X.

Confronta il contenuto della locazione di memoria specificata con il contenuto del registro indice X.

CPY Compare memory and index Y.

Confronta il contenuto della locazione di memoria specificata con il contenuto del registro Y.

DEC Decrement memory by 1.

Toglie 1 al contenuto della locazione di memoria specificata.

DEX Decrement index X by 1

Toglie 1 al contenuto del Registro indice X.

DEY Decrement index Y by 1

Toglie 1 al contenuto del Registro indice Y.

EOR Exclusive-OR memory with accumulator.

Esegue un' operazione di OR esclusivo fra il contenuto della locazione di memoria specificata ed il contenuto dell' Accumulatore.

Il risultato dell' operazione viene riportato nell' Accumulatore.

INC Increment memory by 1

Somma 1 al contenuto della locazione di memoria specificata.

INC Increment index X by 1

Incrementa il registro indice X di 1.

INY Increment index Y by 1.

Incrementa il registro indice Y di 1.

JMP Jump to new location

Esegue un salto incondizionato alla locazione di memoria specificata.

JSR Jump to new location saving return address.

Salta ad un nuovo indirizzo salvando il PC nello Stack.

LDA Load accumulator with memory.

Carica nell' Accumulatore il contenuto della memoria specificato.

LDX Load index X with memory.

Carica nel registro indice X il contenuto della memoria specificato.

LDY Load index Y with memory.

Carica nel registro indice Y il contenuto della memoria specificato.

LSR Shift right one bit

Esegue uno spostamento verso destra del contenuto dell' Accumulatore o di una dichiarata locazione di memoria. Il bit meno significativo del contenuto di partenza viene riportato nel Carry. Il bit che si libera viene riempito con uno 0.

NOP No operation

Non esegue alcuna operazione significativa.

ORA OR memory with accumulator.

Esegue un' operazione di OR logico fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata.

Il risultato dell' operazione viene riportato nell' Accumulatore.

PHA Push accumulator on stack

Esegue la memorizzazione del contenuto dell' Accumulatore nello Stack.

Lo Stack Pointer viene decrementato di 1.

PHP Push processor status on Stack

Esegue la memorizzazione del registro di Status nello Stack.

Lo Stack Pointer viene decrementato di 1.

PLA Pull accumulator from Stack.

Carica nell' Accumulatore il contenuto della locazione dell' area di Stack il cui indirizzo e' in quel momento nello Stack Pointer.

Lo Stack Pointer viene incrementato di 1.

PLP Pull processor status from Stack.

Esegue il caricamento del registro di Status con il contenuto della locazione indirizzata dallo Stack Pointer.



Lo Stack Pointer viene decrementato di 1.

ROL Rotate one bit left.

Esegue la rotazione verso sinistra del contenuto dell' Accumulatore o del contenuto della locazione di memoria specificata.

ROR Rotate one bit right.

Esegue la rotazione verso destra del contenuto dell' Accumulatore o del contenuto della locazione di memoria specificata.

RTI Return from interrupt.

Esegue il ritorno dall' interruzione (INTERRUPT).  
Vengono ripristinati i valori dello Status e del PC prelevandoli dallo Stack.

RTS Return from subroutine.

Esegue il ritorno da una subroutine ripristinando il PC con i dati provenienti dallo Stack.

SBC Subtract memory from accumulator with borrow.

Esegue la sottrazione aritmetica fra il contenuto dell' Accumulatore ed il contenuto della locazione di memoria specificata, sottraendo anche il Carry negato.  
Il risultato dell' operazione viene riportato nell' Accumulatore.

SEC Set carry flag

Mette a 1 il bit di Carry.

SED Set decimal mode.

Mette a 1 il bit del FLAG DECIMAL.

Le operazioni aritmetiche svolte successivamente a questa istruzione sono effettuate in modo decimale.

SEI Set interrupt disable status.

Mette a 1 il bit di INTERRUPT.

Dopo questa istruzione la CPU non risponde piu' alle richieste di interruzione.

STA Store accumulator in memory

Esegue la scrittura del contenuto dell' Accumulatore nella locazione di memoria specificata.

STX Store index X in memory.

Esegue la scrittura del contenuto del registro indice X nella locazione di memoria specificata.

STY Store index Y in memory.

Esegue la scrittura del contenuto del registro indice Y nella locazione di memoria specificata.

TAX Transfer accumulator to index X.

Copia il contenuto dell' Accumulatore nel registro indice X.

L' Accumulatore resta invariato.

TAY Transfer accumulator to index Y.

Copia il contenuto dell' Accumulatore nel registro indice Y.

L' Accumulatore resta invariato.

TYA Transfer index Y to accumulator.

Copia il contenuto del registro Y nell' Accumulatore.  
Il registro Y resta invariato.

TSX Transfer Stack Pointer to index X.

Copia il contenuto dello Stack Pointer nel registro indice X.

Lo Stack Pointer resta invariato.

TXA Transfer index X to accumulator.

Copia il contenuto del registro X nell' Accumulatore.  
Il registro X resta invariato.

TXS Transfer index X to Stack Pointer.

Copia il contenuto del registro indice X nello Stack Pointer.

Il registro X rimane invariato.

TYA Transfer index Y to accumulator.

Copia il contenuto del registro Y nell' Accumulatore.  
Il registro Y resta invariato.

































## ABBREVIAZIONI PER LE PAROLE CHIAVE DEL BASIC

Il Basic del CBM 64 consente di abbreviare gran parte delle parole chiave consentendo un risparmio di tempo e soprattutto la possibilita' di inserire lunghe righe di programma.

Ad esempio la parola PRINT puo' essere abbreviata con un singolo punto interrogativo (?). Per le altre sara' sufficiente scrivere la prima o le prime due lettere normalmente e la seconda o la terza con l' uso del tasto SHIFT.

La tabella seguente mostra come fare ed il risultato provvisorio sullo schermo.

| COMANDO | ABB.              | SCHERMO | COMANDO | ABB.              | SCHERMO |
|---------|-------------------|---------|---------|-------------------|---------|
| ABS     | A <b>SHIFT</b> B  | A       | END     | E <b>SHIFT</b> N  | E       |
| AND     | A <b>SHIFT</b> N  | A       | EXP     | E <b>SHIFT</b> X  | E       |
| ASC     | A <b>SHIFT</b> S  | A       | FN      | NONE              | FN      |
| ATN     | A <b>SHIFT</b> T  | A       | FOR     | F <b>SHIFT</b> O  | F       |
| CHR\$   | C <b>SHIFT</b> H  | C       | FRE     | F <b>SHIFT</b> R  | F       |
| CLOSE   | CL <b>SHIFT</b> O | CL      | GET     | G <b>SHIFT</b> E  | G       |
| CLR     | C <b>SHIFT</b> L  | C       | GET#    | NONE              | GET#    |
| CMD     | C <b>SHIFT</b> M  | C       | GOSUB   | GO <b>SHIFT</b> S | GO      |
| CONT    | C <b>SHIFT</b> O  | C       | GOTO    | G <b>SHIFT</b> O  | G       |
| COS     | NONE              | COS     | IF      | NONE              | IF      |
| DATA    | D <b>SHIFT</b> A  | D       | INPUT   | NONE              | INPUT   |
| DEF     | D <b>SHIFT</b> E  | D       | INPUT#  | I <b>SHIFT</b> N  | I       |
| DIM     | D <b>SHIFT</b> I  | D       | INT     | NONE              | INT     |

| COMANDO | ABB.              | SCHERMO  | COMANDO | ABB.              | SCHERMO   |
|---------|-------------------|--|---------|-------------------|---|
| LEFT\$  | LE <b>SHIFT</b> F | LE    | RIGHT\$ | R <b>SHIFT</b> I  | R    |
| LEN     | NONE              | LEN  | RND     | R <b>SHIFT</b> N  | R    |
| LET     | L <b>SHIFT</b> E  | L     | RUN     | R <b>SHIFT</b> U  | R    |
| LIST    | L <b>SHIFT</b> I  | L     | SAVE    | S <b>SHIFT</b> A  | S    |
| LOAD    | L <b>SHIFT</b> O  | L     | SGN     | S <b>SHIFT</b> G  | S    |
| LOG     | NONE              | LOG  | SIN     | S <b>SHIFT</b> I  | S    |
| MID\$   | M <b>SHIFT</b> I  | M     | SPC(    | S <b>SHIFT</b> P  | S    |
| NEW     | NONE              | NEW  | SQR     | S <b>SHIFT</b> Q  | S    |
| NEXT    | N <b>SHIFT</b> E  | N     | STATUS  | ST                | ST  |
| NOT     | N <b>SHIFT</b> O  | N     | STEP    | ST <b>SHIFT</b> E | ST   |
| ON      | NONE              | ON   | STOP    | S <b>SHIFT</b> T  | S    |
| OPEN    | O <b>SHIFT</b> P  | O     | STR\$   | ST <b>SHIFT</b> R | ST   |
| OR      | NONE              | OR   | SYS     | S <b>SHIFT</b> Y  | S    |
| PEEK    | P <b>SHIFT</b> E  | P     | TAB(    | T <b>SHIFT</b> A  | T    |
| POKE    | P <b>SHIFT</b> O  | P     | TAN     | NONE              | TAN   |
| POS     | NONE              | POS  | THEN    | T <b>SHIFT</b> H  | T    |
| PRINT   | ?                 | ?  | TIME    | TI                | TI  |
| PRINT#  | P <b>SHIFT</b> R  | P    | TIMES   | TI\$              | TI\$  |
| READ    | R <b>SHIFT</b> E  | R   | USR     | U <b>SHIFT</b> S  | U  |
| REM     | NONE              | REM  | VAL     | V <b>SHIFT</b> A  | V  |
| RESTORE | RE <b>SHIFT</b> S | RE  | VERIFY  | V <b>SHIFT</b> E  | V  |
| RETURN  | RE <b>SHIFT</b> T | RE  | WAIT    | W <b>SHIFT</b> A  | W  |















## CODICI DELLO SCHERMO VIDEO






















































La seguente tabella elenca tutti i gruppi (SET) di caratteri presenti in ROM. Mostra cioe' i parametri numerici da usare con il comando POKE per la memoria di schermo che ricordiamo va da 1024 a 2023.

Ad esempio se desideriamo visualizzare la lettera A nella locazione di schermo 1500 eseguiremo: POKE 1500,1. Vedi le prime pagine di questo manuale per passare da maiuscolo a minuscolo e viceversa.

Aggiungendo 128 ai valori presentati nella tabella otteniamo lo stesso simbolo, ma in reverse.

Ricordiamo che le locazioni di memoria video possono anche essere lette con PEEK(locazione di memoria).






























| SET 1 | SET 2 | POKE | SET 1 | SET 2 | POKE | SET 1   | SET 2 | POKE |
|-------|-------|------|-------|-------|------|---|-------|------|
| @     |       | 0    | C     | c     | 3    | F   | f     | 6    |
| A     | a     | 1    | D     | d     | 4    | G   | g     | 7    |
| B     | b     | 2    | E     | e     | 5    | H   | h     | 8    |
| I     | i     | 9    | %     |       | 37   |    | A     | 65   |
| J     | j     | 10   | &     |       | 38   |    | B     | 66   |
| K     | k     | 11   | '     |       | 39   |    | C     | 67   |
| L     | l     | 12   | (     |       | 40   |   | D     | 68   |
| M     | m     | 13   | )     |       | 41   |  | E     | 69   |
| N     | n     | 14   | *     |       | 42   |  | F     | 70   |
| O     | o     | 15   | +     |       | 43   |  | G     | 71   |
| P     | p     | 16   | ,     |       | 44   |  | H     | 72   |
| Q     | q     | 17   | -     |       | 45   |  | I     | 73   |
| R     | r     | 18   | .     |       | 46   |  | J     | 74   |
| S     | s     | 19   | /     |       | 47   |  | K     | 75   |
| T     | t     | 20   | 0     |       | 48   |  | L     | 76   |
| U     | u     | 21   | 1     |       | 49   |  | M     | 77   |
| V     | v     | 22   | 2     |       | 50   |  | N     | 78   |

| SET 1   | SET 2   | POKE | SET 1   | SET 2   | POKE | SET 1   | SET 2   | POKE |
|---|---|------|---|---|------|---|---|------|
| W   | w   | 23   | 3   |   | 51   |    | O   | 79   |
| X   | x   | 24   | 4   |   | 52   |    | P   | 80   |
| Y   | y   | 25   | 5   |   | 53   |    | Q   | 81   |
| Z   | z   | 26   | 6   |   | 54   |    | R   | 82   |
| [   |   | 27   | 7   |   | 55   |    | S   | 83   |
| £   |   | 28   | 8   |   | 56   |    | T   | 84   |
| ]   |   | 29   | 9   |   | 57   |    | U   | 85   |
| ↑   |   | 30   | :   |   | 58   |    | V   | 86   |
| ←   |   | 31   | ;   |   | 59   |    | W   | 87   |
| <b>SPACE</b>  |   | 32   | <   |   | 60   |    | X   | 88   |
| !   |   | 33   | =   |   | 61   |    | Y   | 89   |
| "   |   | 34   | >   |   | 62   |    | Z   | 90   |
| #   |   | 35   | ?   |   | 63   |    |   | 91   |
| \$  |   | 36   |    |   | 64   |    |   | 92   |
|    |   | 93   |    |  | 105  |    |   | 117  |
|    |  | 94   |    |   | 106  |    |   | 118  |
|    |  | 95   |    |   | 107  |    |   | 119  |
| <b>SPACE</b>  |   | 96   |    |   | 108  |    |   | 120  |
|   |   | 97   |   |   | 109  |   |   | 121  |
|  |   | 98   |  |   | 110  |  |  | 122  |
|  |   | 99   |  |   | 111  |  |   | 123  |
|  |   | 100  |  |   | 112  |  |   | 124  |
|  |   | 101  |  |   | 113  |  |   | 125  |
|  |   | 102  |  |   | 114  |  |   | 126  |
|  |   | 103  |  |   | 115  |  |   | 127  |
|  |   | 104  |  |   | 116  |   |   |      |







































































I CODICI DA 128 A 255 SONO LE IMMAGINI IN REVERSE  
DEI CODICI 0-127

# CODICI ASCII E CHR\$

Questa tabella mostra i caratteri e le funzioni che si possono ottenere con l'istruzione CHR\$(X) appunto al variare di X. Naturalmente puo' essere utilizzata la funzione inversa ASC(A\$), per cui a PRINT CHR\$(67) avremo una visualizzazione della lettera C mentre con PRINT ASC("C") avremo come risposta 65.

| PRINTS   | CHR\$ | PRINTS  | CHR\$ | PRINTS  | CHR\$ | PRINTS  | CHR\$ |
|--|-------|---|-------|---|-------|---|-------|
|  | 0     |    | 17    | "   | 34    | 3   | 51    |
|  | 1     |    | 18    | #   | 35    | 4   | 52    |
|  | 2     |    | 19    | \$  | 36    | 5   | 53    |
|  | 3     |    | 20    | %   | 37    | 6   | 54    |
|  | 4     |   | 21    | &   | 38    | 7   | 55    |
|   | 5     |   | 22    | .   | 39    | 8   | 56    |
|  | 6     |   | 23    | (   | 40    | 9   | 57    |
|  | 7     |   | 24    | )   | 41    | :   | 58    |
| DISABLES   | 8     |   | 25    | *   | 42    | ;   | 59    |
| ENABLES    | 9     |   | 26    | +   | 43    | <   | 60    |
|  | 10    |   | 27    | ,   | 44    | =   | 61    |
|  | 11    |   | 28    | -   | 45    | >   | 62    |
|  | 12    |  | 29    | .   | 46    | ?   | 63    |
|   | 13    |  | 30    | /   | 47    | @   | 64    |
|   | 14    |  | 31    | 0   | 48    | A   | 65    |
|  | 15    |  | 32    | 1   | 49    | B   | 66    |
|  | 16    | !   | 33    | 2   | 50    | C   | 67    |
| D  | 68    |  | 97    |  | 126   |  | 155   |
| E  | 69    |  | 98    |  | 127   |  | 156   |
| F  | 70    |  | 99    |   | 128   |  | 157   |
| G  | 71    |  | 100   |  | 129   |  | 158   |
| H  | 72    |  | 101   |   | 130   |  | 159   |



| PRINTS  | CHR\$ | PRINTS  | CHR\$ | PRINTS  | CHR\$ | PRINTS  | CHR\$ |
|---|-------|---|-------|---|-------|---|-------|
| I   | 73    |    | 102   |   | 131   |    | 160   |
| J   | 74    |    | 103   |   | 132   |    | 161   |
| K   | 75    |    | 104   | f1  | 133   |    | 162   |
| L   | 76    |    | 105   | f3  | 134   |    | 163   |
| M   | 77    |    | 106   | f5  | 135   |    | 164   |
| N   | 78    |    | 107   | f7  | 136   |    | 165   |
| O   | 79    |    | 108   | f2  | 137   |    | 166   |
| P   | 80    |    | 109   | f4  | 138   |    | 167   |
| Q   | 81    |    | 110   | f6  | 139   |    | 168   |
| R   | 82    |    | 111   | f8  | 140   |    | 169   |
| S   | 83    |    | 112   |    | 141   |    | 170   |
| T   | 84    |    | 113   |    | 142   |    | 171   |
| U   | 85    |    | 114   |   | 143   |    | 172   |
| V   | 86    |    | 115   |    | 144   |    | 173   |
| W   | 87    |    | 116   |    | 145   |    | 174   |
| X   | 88    |    | 117   |    | 146   |    | 175   |
| Y   | 89    |    | 118   |    | 147   |    | 176   |
| Z   | 90    |    | 119   |    | 148   |    | 177   |
| [   | 91    |    | 120   |    | 149   |    | 178   |
| £   | 92    |    | 121   |    | 150   |    | 179   |
| ]   | 93    |    | 122   |  | 151   |    | 180   |
| ↑   | 94    |   | 123   |  | 152   |   | 181   |
| ←   | 95    |  | 124   |  | 153   |  | 182   |
|  | 96    |  | 125   |  | 154   |  | 183   |
|  | 184   |  | 186   |  | 188   |  | 190   |
|  | 185   |  | 187   |  | 189   |  | 191   |

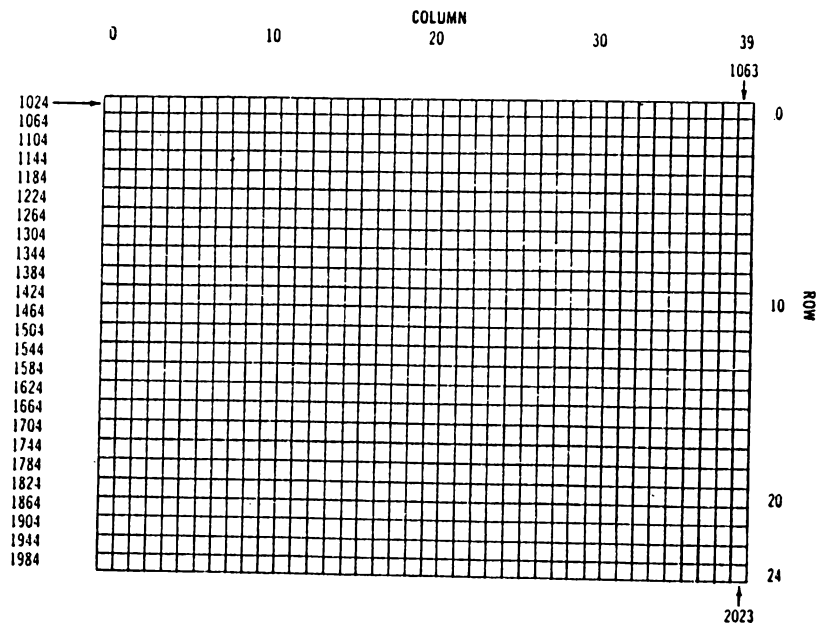
CODICI 192-223  
 CODICI 224-254  
 CODICE 255

COME  
 COME  
 COME

96-127  
 160-190  
 126

MAPPA DELLA MEMORIA DI SCHERMO

La seguente tabella riporta una griglia delle locazioni di memoria video o di schermo che potra' essere utile fotocopiare per eseguire disegni o videate con la massima precisione.



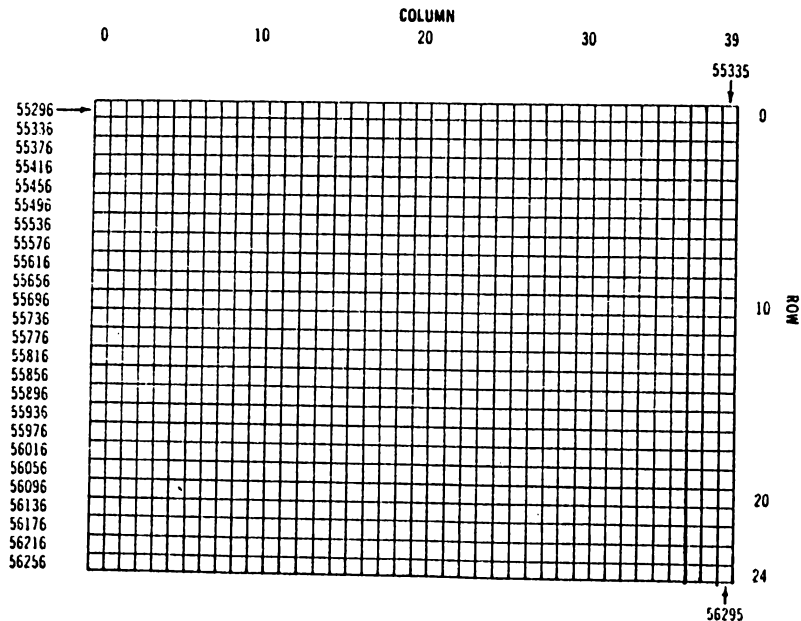
## MAPPA DELLA MEMORIA COLORE

La seguente griglia, complementare a quella appena vista mostra le locazioni di colore di schermo.

Ricordiamo che i valori da dare alle singole locazioni per ottenere un dato colore sono i seguenti:

|           |                  |
|-----------|------------------|
| 0-NERO    | 8-ARANCIO        |
| 1-BIANCO  | 9-MARRONE        |
| 2-ROSSO   | 10-ROSA          |
| 3-AZZURRO | 11-GRIGIO 1      |
| 4-PORPORA | 12-GRIGIO 2      |
| 5-VERDE   | 13-GRIGIO CHIARO |
| 6-BLEU    | 14-BLEU CHIARO   |
| 7-GIALLO  | 15-GRIGIO 3      |

Per esempio qualora si desideri che il carattere nella terza posizione a partire dall' alto a sinistra dello schermo sia in rosso, scriveremo POKE 55298,2.



## VALORE DELLE NOTE MUSICALI

Questa tabella contiene la serie completa dei valori da dare ai registri HI e LOW FREQUENCY del sintetizzatore sonoro per ottenere le note secondo il sistema anglosassone.

| NOTE MUSICALI |        | FREQUENZA OSCILLATORE |    |     |
|---------------|--------|-----------------------|----|-----|
| NOTE          | OTTAVE | DECIMALE              | HI | LOW |
| 0             | C-0    | 268                   | 1  | 12  |
| 1             | C#-0   | 284                   | 1  | 28  |
| 2             | D-0    | 301                   | 1  | 45  |
| 3             | D#-0   | 318                   | 1  | 62  |
| 4             | E-0    | 337                   | 1  | 81  |
| 5             | F-0    | 358                   | 1  | 102 |
| 6             | F#-0   | 379                   | 1  | 123 |
| 7             | G-0    | 401                   | 1  | 145 |
| 8             | G#-0   | 425                   | 1  | 169 |
| 9             | A-0    | 451                   | 1  | 195 |
| 10            | A#-0   | 477                   | 1  | 221 |
| 11            | B-0    | 506                   | 1  | 250 |
| 16            | C-1    | 536                   | 2  | 24  |
| 17            | C#-1   | 568                   | 2  | 56  |
| 18            | D-1    | 602                   | 2  | 90  |
| 19            | D#-1   | 637                   | 2  | 125 |
| 20            | E-1    | 675                   | 2  | 163 |
| 21            | F-1    | 716                   | 2  | 204 |
| 22            | F#-1   | 758                   | 2  | 246 |
| 23            | G-1    | 803                   | 3  | 35  |
| 24            | G#-1   | 851                   | 3  | 83  |
| 25            | A-1    | 902                   | 3  | 134 |
| 26            | A#-1   | 955                   | 3  | 187 |
| 27            | B-1    | 1012                  | 3  | 244 |
| 32            | C-2    | 1072                  | 4  | 48  |

| NOTE MUSICALI |        | FREQUENZA OSCILLATORE |    |     |
|---------------|--------|-----------------------|----|-----|
| NOTE          | OTTAVE | DECIMALE              | HI | LOW |
| 33            | C#-2   | 1136                  | 4  | 112 |
| 34            | D-2    | 1204                  | 4  | 180 |
| 35            | D#-2   | 1275                  | 4  | 251 |
| 36            | E-2    | 1351                  | 5  | 71  |
| 37            | F-2    | 1432                  | 5  | 152 |
| 38            | F#-2   | 1517                  | 5  | 237 |
| 39            | G-2    | 1607                  | 6  | 71  |
| 40            | G#-2   | 1703                  | 6  | 167 |
| 41            | A-2    | 1804                  | 7  | 12  |
| 42            | A#-2   | 1911                  | 7  | 119 |
| 43            | B-2    | 2025                  | 7  | 233 |
| 48            | C-3    | 2145                  | 8  | 97  |
| 49            | C#-3   | 2273                  | 8  | 225 |
| 50            | D-3    | 2408                  | 9  | 104 |
| 51            | D#-3   | 2551                  | 9  | 247 |
| 52            | E-3    | 2703                  | 10 | 143 |
| 53            | F-3    | 2864                  | 11 | 48  |
| 54            | F#-3   | 3034                  | 11 | 218 |
| 55            | G-3    | 3215                  | 12 | 143 |
| 56            | G#-3   | 3406                  | 13 | 78  |
| 57            | A-3    | 3608                  | 14 | 24  |
| 58            | A#-3   | 3823                  | 14 | 239 |
| 59            | B-3    | 4050                  | 15 | 210 |
| 64            | C-4    | 4291                  | 16 | 195 |
| 65            | C#-4   | 4547                  | 17 | 195 |
| 66            | D-4    | 4817                  | 18 | 209 |
| 67            | D#-4   | 5103                  | 19 | 239 |
| 68            | E-4    | 5407                  | 21 | 31  |
| 69            | F-4    | 5728                  | 22 | 96  |
| 70            | F#-4   | 6069                  | 23 | 181 |
| 71            | G-4    | 6430                  | 25 | 30  |
| 72            | G#-4   | 6812                  | 26 | 156 |
| 73            | A-4    | 7217                  | 28 | 49  |
| 74            | A#-4   | 7647                  | 29 | 223 |
| 75            | B-4    | 8101                  | 31 | 165 |
| 80            | C-5    | 8583                  | 33 | 135 |
| 81            | C#-5   | 9094                  | 35 | 134 |

| NOTE MUSICALI |        | FREQUENZA OSCILLATORE |     |     |
|---------------|--------|-----------------------|-----|-----|
| NOTE          | OTTAVE | DECIMALE              | HI  | LOW |
| 82            | C-0    | 9634                  | 37  | 162 |
| 83            | C#-0   | 10207                 | 39  | 223 |
| 84            | D-0    | 10814                 | 42  | 62  |
| 85            | F-5    | 11457                 | 44  | 193 |
| 86            | F#-5   | 12139                 | 47  | 107 |
| 87            | G-5    | 12860                 | 50  | 60  |
| 88            | G#-5   | 13625                 | 53  | 57  |
| 89            | A-5    | 14435                 | 56  | 99  |
| 90            | A#-5   | 15294                 | 59  | 190 |
| 91            | B-5    | 16203                 | 63  | 75  |
| 96            | C-6    | 17167                 | 67  | 15  |
| 97            | C#-6   | 18188                 | 71  | 12  |
| 98            | D-6    | 19269                 | 75  | 69  |
| 99            | D#-6   | 20415                 | 79  | 191 |
| 100           | E-6    | 21629                 | 84  | 125 |
| 101           | F-6    | 22915                 | 89  | 131 |
| 102           | F#-6   | 24278                 | 94  | 214 |
| 103           | G-6    | 25721                 | 100 | 121 |
| 104           | G#-6   | 27251                 | 106 | 115 |
| 105           | A-6    | 28871                 | 112 | 199 |
| 106           | A#-6   | 30588                 | 119 | 124 |
| 107           | B-6    | 32407                 | 126 | 151 |
| 112           | C-7    | 34334                 | 134 | 30  |
| 113           | C#-7   | 36376                 | 142 | 24  |
| 114           | D-7    | 38539                 | 150 | 139 |
| 115           | D#-7   | 40830                 | 159 | 126 |
| 116           | E-7    | 43258                 | 168 | 250 |
| 117           | F-7    | 45830                 | 179 | 6   |
| 118           | F#-7   | 48556                 | 189 | 172 |
| 119           | G-7    | 51443                 | 200 | 243 |
| 120           | G#-7   | 54502                 | 212 | 230 |
| 121           | A-7    | 57743                 | 225 | 143 |
| 122           | A#-7   | 61176                 | 238 | 248 |
| 123           | B-7    | 64814                 | 253 | 46  |

## FISSAGGIO

## FILTRI

| INDIRIZZO | CONTENUTO  |
|-----------|--|
| 54293     | Low cutoff frequency (0-7)   |
| 54294     | High cutoff frequency (0-255)  |
| 54295     | Resonance (bits 4-7)<br>Filter voice 3 (bit 2)<br>Filter voice 2 (bit 1)<br>Filter voice 1 (bit 0) |
| 54296     | High pass (bit 6)<br>Bandpass (bit 5)<br>Low pass (bit 4)<br>Volume (bits 0-3)                     |

# Programmi



# DATA BASE

```

10 OPEN15,8,15,"I0"
100 DIM A$(300)
110 REM*** DATA BASE 29/03/83 ***
120 POKE36879,42:PRINT"☐";
130 PRINT"(CLEAR)(3 DOWN)"
140 PRINTTAB(11)"████████████████████":PRINTTAB(
11)"☐"
150 PRINTTAB(11)"☐ | "
160 PRINTTAB(11)"☐ | DATA BASE | ☐":PRINTTAB(
11)"☐ | "
170 PRINTTAB(11)"☐"
PRINTTAB(11)"████████████████████"
190 PRINTTAB(17)"(2 DOWN)BY":PRINTTAB(11)"
(DOWN)E.V.M. COMPUTERS":PRINTTAB(14)"(DOWN)(
C) 1983"
200 FOR T=1TO3000:NEXT
210 POKE650,128:POKE36879,110:PRINT"☐";
220 GOTO 530
230 REM *** LETTURA FILE ***
240 PRINT"(CLEAR) (RVS) LETTUR
A FILE (RVSOFF)"
250 PRINT"(4 DOWN) D - DA DISCO
"
260 PRINT"(2 DOWN) N - DA NASTR
O"
"(2 DOWN) T - RITORNA
MENU"
280 GETA$:IFA$=""THEN280
290 IFA$="D"THEN2520
300 IFA$="N"THEN330
310 IFA$="T"THEN530
320 GOTO 280
330 PRINT"(CLEAR) (RVS) RICERC
A FILE (RVSOFF)"
340 INPUT "(2 DOWN)(RVS)FILE:(RVSOFF)";F$
350 OPEN1,1,0,F$:PRINT"(DOWN)FILE: (RVS)"F
$(RVSOFF)"

```

## DATA BASE

```

360 INPUT# 1,CA:FOR M=1TOCA:INPUT# 1,CA$(M
   ):INPUT# 1,DD(M):NEXT M
370 INPUT# 1,Y:FOR I=1TOY:INPUT# 1,A$(I):NEXT
   I:CLOSE1:GOTO 530
380 REM *** REGISTRA FILE ***
390 PRINT"(CLEAR)          (RVS)  . REGISTR
   A FILE  (RVSOFF)"
400 PRINT"(4 DOWN)          D - SU DISCO
   "
410 PRINT"(2 DOWN)          N - SU NASTR
   O"
420 PRINT"(2 DOWN)          T - TORNA ME
   NU"
430 GETA$:IFA$=""THEN430
440 IFA$="D"THEN2650
450 IFA$="N"THEN480
460 IFA$="T"THEN530
470 GOTO 430
480 PRINT"(CLEAR)          (RVS)  REGIST
   RA FILE (RVSOFF)"
490 PRINT"(2 DOWN)SCRIVI NOME  FILE:":INPUT
   "(DOWN)(RVS)FILE:(RVSOFF)":F$
500 OPEN1,1,1,F$:PRINT"(DOWN)FILE: (RVS)"F
   $"(RVSOFF)"
510 PRINT#1,CA:FOR M=1TOCA:PRINT#1,CA$(M):
   PRINT#1,DD(M):NEXT M
520 PRINT#1,Y:FOR I=1TOY:PRINT#1,A$(I):NEXT
   I:CLOSE1:GOTO 530
530 PRINT"(CLEAR)          (RVS)  DATA -
   BASE (RVSOFF)"
540 PRINT"(DOWN)          1- LEGGO FILE "
550 PRINT"(DOWN)          2- REGISTRO FIL
   E"
560 PRINT"(DOWN)          3- CREA ARCHIVI
   O"
570 PRINT"(DOWN)          4- CONTINUO ARC
   HIVO"

```

## DATA BASE

```

580 PRINT"(DOWN)          5- ELENCO RECOR
   D"
590 PRINT"(DOWN)          6- SORT RECORD"
600 PRINT"(DOWN)          7- RICERCAECO
   RD"
610 PRINT"(DOWN)          S- STAMPA RECOR
   D"
620 PRINT"(DOWN)          M- MEMORIA FREE
   .
622 PRINT"(DOWN)          E- END "
630 PRINT"(DOWN)          (RVS)CAMPI=(RVSOFF)"
   CA:PRINTTAB(20)"(UP)(RVS)RECORD=(RVSOFF)"
   Y
640 GETE$:IFE$=""THEN640
650 IFE$="1"THEN230
660 IFE$="2"THEN380
670 IFE$="3"THEN750
680 IFE$="4"THEN970
690 IFE$="5"THEN8000
700 IFE$=""THEN2280
710 IFE$="7"THEN1690
720 IFE$="M"THEN1330
730 IFE$="S"THEN1450
732 IFE$="E"THENPRINT"(CLEAR)(10 DOWN)
   (RVS)FINE  PROGRAMMA.(RVSOFF)"
   :END
740 GOTO 640
750 REM *** CRED ARCHIVIO ***
760 PRINT"(CLEAR)(DOWN)          (RVS)
   NOTA BENE (RVSOFF)"
770 PRINT"(2 DOWN)DEFINISCI IL NUMERO ED I
   NOMI DEL CAMP,(DOWN)(MASSIMO (RVS)9(RVSOFF)
   CAMPI DA (RVS)10(RVSOFF) ";
780 PRINT"CARATTERI).":PRINT"(DOWN)POI DEF
   INISCI LA LUNGHEZZA DEL CAMPO, ";
781 PRINT"AL(DOWN)MASSIMO DI (RVS)25(RVSOFF)
   CARATTERI OGNUNO."

```

## DATA BASE

```

790 PRINT"(DOWN)LA LUNGHEZZA DI UN RECORD
    PUO' ESSERE DI(DOWN)(RVS)88(RVSOFF) C
    ARATTERI IN TOTALE."
792 PRINT" (ESCLUSI I NOMI(DOWN)DEI CAMPI)
    ."
800 PRINT(2 DOWN)          (RVS)PREMI
    UN TASTO(RVSOFF)
810 GETAU$:IPAUS$=""THEN810
820 PRINT"(CLEAR)          (RVS) CREDI A
    RCHIVIO (RVSOFF)";PRINT"(2 DOWN)QU
    ANTI CAMPI ? (RVS)"
830 GETR$:IFR$=""THEN830
840 CA=VAL(R$)
850 PRINTCA:PRINT"(DOWN) (RVS) NOME DEL C
    AMPO (RVSOFF) (RVS) LUNGHEZZA CAMP
    O(RVSOFF)":PRINT
860 AH=0:FOR K=1TOCA
870 PRINTK:;INPUT " ";CA$(K):IFLEN CA$(K))
    >10THENPRINT"(2 UP)":GOTO 870
880 PRINT"(UP)(25 RIGHT)":;INPUT DD(K):IFD
    D(K)>25THEN880
890 AH=AH+(DD(K)):IFAH>88THEN880
900 NEXT
910 BH=88-AH:PRINT"(HOME)(18 DOWN)
    "BH"(RVS)CARATTERI LIBERI(RVSOFF)"
920 PRINT" (DOWN)VUOI CORREGGERE
    ?"
930 GETAP$:IFAP$=""THEN930
940 IFAP$="S"THEN961
950 IFAP$="N"THEN970
960 GOTO 930
961 INPUT "(DOWN)N. DEL CAMPO: ";NC:IFNC>CA
    +1THEN961
962 INPUT "(DOWN)(RVS)NOME CAMPO(RVSOFF)";
    CA$(NC):INPUT "(DOWN)(RVS)LUNGHEZZA(RVSOFF)"
    ;DD(NC)
963 PRINT"(CLEAR)(DOWN) (RVS) NOME DEL CA
    MPO (RVSOFF)":;PRINT" (RVS) LUNGHE
    ZZA (RVSOFF)":PRINT:IFNC>CATHENCA=NC

```

## DATA BASE

```

964 AH=0:FOR K=1TOCA:PRINT"          ";CA$(K)
965 PRINT"(UP)(25 RIGHT)"DD(K):AH=AH+DD(K)
    :NEXT
966 BH=88-AH:PRINT"(HOME)(18 DOWN)
    "BH"(RVS)CARATTERI LIBERI(RVSOFF)"
968 PRINT"          (DOWN)VUOI CORREGGERE
    ?"
969 GOTO 930
970 IFCA(1)THEN540
980 PRINT"(CLEAR)          (RVS) CREA A
    RCHIVIO (RVSOFF)"
990 Y=Y+1:I=Y
1000 PRINT"(2 DOWN) PER TORNARE BATTI (RVS)
    * (RVSOFF)"
1010 PRINT"(DOWN)NUMERO "I:PRINT"(UP)(12 RIGHT)
    (RVS)1234567890123456789012345(RVSOFF)"
    :PRINT
1020 A$(I)="":FOR K=1TOCA
1030 PRINT"(RVS)"CA$(K)"(RVSOFF)":PRINT"(UP)
    (12 RIGHT)";:FOR E=1TODD(K):PRINT"-";:
    NEXT E
1040 FOR U=1TODD(K)+2:PRINT"(LEFT)";:NEXT U

1050 INPUT AB$(K)
1060 IFLENK AB$(K))>DD(K)THEN1030
1070 A$(I)=A$(I)+AB$(K)
1080 IFLEFT$(AB$(K),1)="*"THENY=Y-1:GOTO 53
    0
1090 NEXT K:IFCC=99THEN2500
1100 GOTO 980

1120 PRINT"(DOWN)(RVS)I=INSERT D=DEL. C=CO
    RR. M=MENU V=VEDI(RVSOFF)"
1130 GETA$:IFA$=""THEN1130
1140 IFA$="I"THEN2460
1150 IFA$="C"THEN1390
1152 IFA$="D"THEN3000

```

## DATA BASE

```

1160 IFA$="M"THEN530
1170 IFA$="V"THEN1190
1180 GOTO 1130
1190 G=0:PRINT"(CLEAR)":GOTO 1260
1200 REM
1202 PRINT"(CLEAR)          (RVS)          ELEN
      CO          (RVSOFF)"
"(DOWN)QUANTI DATI VIDEO ?"
1212 GETGG$:IFGG$=""THEN1212
1214 GG=VAL(GG$)
1220 G=0:FOR I=1TOY:S=1:PRINT"(UP)
      RECORD N.(RVS)"I"(RVSOFF)":PRINT
1230 FOR M=1TOCA:AB$(M)=MID$(A$(I),S,DD(M))
      :S=S+DD(M)
1240 PRINT"(RVS)"CA$(M)" (RVSOFF)":PRINT"(UP)
      (10 RIGHT)"AB$(M)
1250 NEXT M:G=G+1:IFG=GGTHEN1120
1260 PRINT"(DOWN)":NEXT I
1270 PRINT"(DOWN)(RVS)I=INSERT =CRREGGI
      D=DELETE M=MENU(RVSOFF)"
1280 GETA$:IFA$=""THEN1280
1290 IFA$="I"THEN2460
1300 IFA$="C"THEN1390
1310 IFA$="M"THEN530
1312 IFA$="D"THEN3000
1320 GOTO 1280
1330 REM** MEMORIA **
1340 PRINT"(CLEAR)":FR=FRE(0)
1350 PRINT"(5 DOWN)          (RVS)MEMORIA
      (RVSOFF) ="FR
1360 PRINT"(2 DOWN)          (RVS)PREMI U
      N TASTO (RVSOFF)"
1370 GETA$:IFA$=""THEN1370
1380 GOTO 530
1390 I=0:INPUT "(DOWN)NUM. DA CORREGGERE";I
      :IFI>YTHEN1390
1400 PRINT"(CLEAR)(2 DOWN)"I:A$(I)="" :FOR M
      =1TOCA

```

## DATA BASE

```

1410 PRINT"(DOWN)(RVS)"CA$(M)"(RVSOFF)";:PRINT
    "    ";:FOR E=1TODD(M):PRINT"-";:NEXT E

1420 FOR U=1TODD(M)+2:PRINT"(LEFT)";:NEXT U
    :INPUT AB$(M)
1430 A$(I)=A$(I)+AB$(M)
1440 NEXT :GOTO 1380
1450 REM*** STAMPA DATI ***
1460 PRINT"(CLEAR)(4 DOWN)    LA STAMPATE E
    ' COLLEGATA ?"
1470 GETAL$:IFAL$=""THEN1470
1480 IFAL$="N"THEN530
1490 IFAL$="S"THEN1510
1500 GOTO 1470
1510 OPEN4,4:GOSUB2130
1520 FOR I=1TOY:S=1
1530 GOSUB1540:NEXT I:CLOSE4:GOTO 530
1540 PRINT#4,I;:A$(I)=A$(I)
1550 FOR M=1TOCA:AB$(M)=MID$(A$(I),S,DD(M))
    :S=S+DD(M)
1560 PRINT#4,LEFT$(DC$,DD);
1570 PRINT#4,AB$(M);
1580 PRINT#4,LEFT$(DC$,DD);
1590 NEXT M:PRINT#4"":RETURN
1600 OPEN4,4
1610 PRINT#4,CHR$(14)"                ELENCO  D
    ATI"
1620 PRINT#4,CHR$(15)
1630 FOR I=1TOY:S=1
1640 PRINT#4,"    DATI N."I
1650 FOR M=1TOCA:AB$(M)=MID$(A$(I),S,DD(M))
    :S=S+DD(M)
1660 PRINT#4,CA$(M);
1670 PRINT#4,CHR$(16)*20: "AB$(M)
1680 NEXT :PRINT#4,"":NEXT I:CLOSE4:GOTO 53
    0
1690 REM*** RICERCA DATI ***

```

# DATA BASE

```

1700 PRINT"(CLEAR)                                (RVS)NOTA
      BENE(RVSOFF)"
1710 PRINT"(DOWN) LA RICERCA DEI DATI PUO'
      ESSERE FATTA (DOWN)SU OGNI CAMPO."
1720 PRINT"(DOWN)BASTA UN CARATTERE PER ELE
      NCARE TUTTI I (DOWN)DATI CHE INIZIANO
      CON QUEL ";
1730 PRINT"CARATTERE.":PRINT"(DOWN)PIU' CAR
      ATTERI RENDONO LA RICERCA PIU' (DOWN)S
      ELETTIVA."
1740 PRINT(4 DOWN)                                (RVS)PREMI
      UN TASTO(RVSOFF)"
1750 GETA$: IFA$=""THEN1750
1760 PRINT"(CLEAR)                                (RVS) RICERCA
      DATI (RVSOFF)"
1770 JJ=0:PRINT"(DOWN)PREMI (RVS)S(RVSOFF)
      PER STAMPARE, O UN TASTO PER (DOWN)P
      ROSEGUIRE."
1780 GETA$: IFA$=""THEN1780
1790 IFA$="S"THEN2070
1800 INPUT "(RVS)(DOWN)DATI:(RVSOFF)";AW$:A
      W=LEN(AW$)
1810 SR=0:FOR I=1TOY:S=1:FOR M=1TOCA
1820 IFA$=MID$(A$(I),S,DD(M))THEN1970
1840 IFAW$=LEFT$(MID$(A$(I),S,DD(M)),AW)THEN
      1970
1850 S=S+DD(M)
1920 NEXT M
1930 NEXT I
1940 PRINT"(DOWN)                                (RVS) FINE D
      ATI (RVSOFF)":PRINT"                                (RVS)PR
      EMI PER TORNARE(RVSOFF) "
1941 CLOSE4
1950 GETA$: IFA$=""THEN1950
1960 GOTO 530
1970 S=1:A$(I)=A$(I):PRINT"(DOWN) NUM."I
1980 FOR N=1TOCA:AB$MID$(A$(I),S,DD(N)):S=
      S+DD(N)

```



# DATA BASE

```

1990 PRINT"(RVS)"CA$(N)" (RVSOFF)":PRINT"(UP)
      (11 RIGHT)"AB$:NEXT N:IFJJ=999THENI=1:
      GOTO 2090
2000 IFJJ=999THEN1930
2010 I=1:SR=SR+1:IFSR=2THEN2030
2020 GOTO 1930
2030 PRINT"(DOWN)                (RVS) PREMI PER
      VEDERE (RVSOFF)"
2040 GETA$:IFA$=""THEN2040
2050 PRINT"(CLEAR)                (RVS)      RICERCA
      DATI      (RVSOFF) "
2060 SR=0GOTO 2020
2070 OPEN#4,4:JJ=999:GOSUB2130
2080 GOTO 1800
2090 PRINT#4,I;
2100 FOR M=1TOCA:PRINT#4,LEFT$(DC$,DD);:PRINT#
      4,AB$(M);:PRINT#4,LEFT$(DC$,DD);
2110 NEXT :PRINT#4,""
2120 GOTO 2000
2130 DA=0:DB=0:FOR M=1TOCA:CA$(M)=CA$(M):DA
      =DA+LEN(CA$(M)):DB=DB+DD(M)
2140 NEXT
2150 IFDA>75THEN1600
2160 IFDB>75THEN1600
2170 DC=75-DA:DC=DC/CA/2:DC=INT(DC)
2180 D=75-DB:DD=DD/CA/2:DD=INT(DD):DC$="
      "
2190 PRINT#4,CHR$(14)"                ELENCO D
      ATI"
2200 PRINT#4,CHR$(15)
2210 FOR M=1TOCA
2220 PRINT#4,LEFT$(DC$,DC);
2230 PRINT#4,CA$(M);
2240 PRINT#4,LEFT$(DC$,DC);
2250 NEXT :PRINT#4,""
2260 FOR F=1TO79:PRINT#4,"-";:NEXT F:PRINT#
      4,""

```

## DATA BASE

```

2270 RETURN
2280 PRINT"(CLEAR)(DOWN) (RVS) NOME DEL CA
      MPO (RVSOFF)";PRINT" (RVS) LUNGHE
      ZZA (RVSOFF)";PRINT:IFNC)CATHENCA=NC
2282 FOR K=1TOCA:PRINTK;" ";CA$(K)
2284 PRINT"(UP)(25 RIGHT)"DD(K):NEXT
2290 PRINT"(DOWN) (RVS)SU QUALE CAMPO VUO
      I IL SORT?(RVSOFF)"
2292 GETR$:IFR$=""THEN2292
2294 R=VAL(R$):S=1:FOR M=1TOCA:S=S+DD(M):A(
      M)=S-DD(M):NEXT :A(1)=1:A1=A(R)
2296 INPUT "(2 DOWN) (RVS)QUANTE LETTERE
      DI PRECISIONE(RVSOFF)";B1
2310 PRINT"(CLEAR)(6 DOWN)"TAB(14)"(RVS)SHE
      LL SORT(RVSOFF)";PRINT"(4 DOWN)"TAB(1
      5)"(RVS)ATTENDERE(RVSOFF)"
2312 TI$="000000":FOR J=1TOY :M=1
2313 M=2*M:IFM=YTHEN2313
2314 M=INT(M/2):IFM=0THEN2430
2320 FOR J=1TOY-M:C=J
2330 D=C+M:IFMID$(A$(C),A1,B1)<=MID$(A$(D),
      A1,B1)THEN2350
2340 AA$=A$(C):A$(C)=A$(D):A$(D)=AA$:C=C-M:
      IFC)0THEN2330
2350 NEXT :GOTO 2314
2430 PRINTTAB(6)"(4 DOWN)"RIGHT$(TI$,4)" S
      ECONDI PER IL SORT!":FOR T=1TO2000:NEXT
2450 GOTO 530
2460 CC=0:C=0:INPUT "N.DA INSERIRE";C
2480 Y=Y+1:FOR I=YTOCSTEP-1:A$(I)=A$(I-1):NEXT
2490 I=C:CC=99:PRINT"(CLEAR)":GOTO 1010
2500 CC=0:GOTO 1270
2520 PRINT"(CLEAR) (RVS) LETTURA F
      ILE DISCO (RVSOFF)"
2530 PRINT"(2 DOWN)SCRIVI NOME FILE:";INPUT
      "(DOWN)(RVS)FILE(RVSOFF)";F$

```

## DATA BASE

```

2540 OPEN2,8,2,F$+" ,S,R":GOSUB2740
2550 INPUT# 2,CA:RS=ST:GOSUB2740:FOR M=1TOC
    A
2560 INPUT# 2,CA$(M):RS=ST:GOSUB2740
2570 INPUT# 2,DD(M):RS=ST:GOSUB2740:NEXT M
2580 INPUT# 2,Y:RS=ST:GOSUB2740:FOR I=1TOY
2590 INPUT# 2,A$(I)
2600 IFRS=64THEN2630
2610 IFRS<>0THEN2640
2620 NEXT :CLOSE2:GOTO 530
2630 CLOSE2:GOTO 530
2640 PRINT"ERRORE DI STATO":CLOSE2:GOTO 550

2650 PRINT"(CLEAR)          (RVS)  REGISTRA
    FILE DISCO (RVSOFF)":CR$=CHR$(13)
2660 PRINT"(2 DOWN)SCRIVI NOME FILE:"INPUT
    "(DOWN)(RVS)FILE(RVSOFF) ";F$
2670 OPEN2,8,2,"@:"+F$+" ,S,W":GOSUB2740
2680 PRINT#2,STR$(CA)CR$;:GOSUB2740:FOR M=1
    TOCA
2690 PRINT#2,CA$(M)CR$;
2700 PRINT#2,STR$(DD(M))CR$;:NEXT M:GOSUB27
    40
2710 PRINT#2,STR$(Y)CR$;:GOSUB2740:FOR I=1TO
    Y
2720 PRINT#2,A$(I)CR$;:GOSUB2740
2730 NEXT I:CLOSE2:GOTO 530
2740 INPUT# 15,EN,EM$,ET,ES
2750 IFEN=0THENRETURN
2760 PRINT"OPERAZIONE ERRATA":CLOSE2:FOR T=
    1TO1000:NEXT :RETURN
3000 INPUT "(DOWN)NUMERO DA CANCELLARE":C:Y
    =Y-1
3010 FOR I=CTOY:A$(I)=A$(I+1):NEXT :GOTO 53
    0
8000 PRINT"(CLEAR)          (RVS)  ELEN
    CO (RVSOFF)"

```

## DATA BASE

```

8100 PRINT"(DOWN)  ";:FOR =1TOCA:PRINT"(RVS)"
      CA$(M)"      ";:NEXT :PRINT
8220 G=0:FOR I=1TOY:S=1:PRINT"(RVS)"I"(RVSOFF)"
      ;
8230 FOR M=1TOCA
8240 PRINTMID$(A$(I),S,DD(M))"      ";:S=S+DD(
      M)
8250 NEXT M:G=G+1:IFG=20THEN8300
8260 PRINT"(DOWN)":NEXT I
8270 GOTO 1270
8300 PRINT"(DOWN)(RVS)I=INSERT D=DEL. C=CO
      RR. M=MENU V=VEDI(RVSOFF)"
8310 GETA$:IFA$=""THEN8310
8340 IFA$="I"THEN2460
8350 IFA$="C"THEN1390
8360 IFA$="M"THEN530
8362 IFA$="D"THEN3000
8370 IFA$="V"THEN8400
8380 GOTO 1130
8400 PRINT"(DOWN)":NEXT I
8410 GOTO 1270

```

# GESTIONE CONTO CORRENTE

```

115 REM*** CONTO CORRENTE ***
120 K=100
130 DIM G(K),D$(K),M(K)
140 GOSUB5000
150 PRINT"    PROGRAMMA GESTIONE CONTI CORR
    ENTI
160 GOSUB5010
180 PRINT"    ELABORAZIONE SU CONTO ESISTEN
    TE =1"
190 PRINT"    CREAZIONE NUOVO CONTO CORRENT
    E =2"
192 PRINT:PRINT"    QUALE";
195 GETX$:IFX$=""THEN195
200 IFX$="1"THEN400
202 IFX$="2"THEN220
204 GOTO 195
220 GOSUB5030
250 PRINT"    - PRENDERE UNA NUOVA CASSETT
    A"
260 GOSUB5060
280 GOSUB5100
310 GOSUB5000
320 PRINT"N. DI C/C (MAX 10 CIFRE) ? ";
330 GOSUB6000
332 A$=LEFT$(Q$,10)
340 PRINT"DOTAZIONE INIZIALE ? ";
350 GOSUB6040
352 C=Q:T=0:G(0)=-1:GOTO 600
400 GOSUB5030
410 PRINT"    - PRENDERE LA CASSETTA DEL C
    /C IN"
420 PRINT"    ESAME"
430 GOSUB5060
460 GOSUB5100
490 OPEN1,1,0,"CONTO"
500 INPUT# 1,A$
501 IFST>0THENGOSUB5130

```

# GESTIONE CONTO CORRENTE

```

503 INPUT# 1,C
504 IFST>0THEN GOSUB5130
506 N=0
510 INPUT# 1,G(N)
511 IFST>0THEN GOSUB5130
512 IFG(N)=-1THEN540
513 INPUT# 1,D$(N)
514 IFST>0THEN GOSUB5130
516 INPUT# 1,M(N)
517 IFST>0THEN GOSUB5130
525 IFN=KTHEN GOSUB5130
530 N=N+1:GOTO 510
540 T=N
541 CLOSE1
543 PRINT"      -- PREMERE STOP SULLA CASSETT
      A"
545 GOSUB5100
600 GOSUB5000
610 PRINT"L'ARCHIVIO IN ESAME COMPRENDE ";
      T
620 PRINT"TRANSAZIONI SUL C/C ";A$
625 PRINT"DOTAZIONE INIZIALE ";C
630 GOSUB5010
640 PRINT"  1= INTRODUZIONI"
650 PRINT"  2= VARIAZIONI"
660 PRINT"  3= ANNULLAMENTI":PRINT
670 PRINT"  4= RICHIESTA VISUALIZZAZIONI"
690 PRINT"  5= "
700 PRINT"  6= RICHIESTA ORDINAMENTO"
710 PRINT"  7= FINE PROCEDURA":PRINT
715 PRINT".    = ";
720 GETX$:IFX$=""THEN720
722 IFASC(X$)<49ORASC(X$)>55THEN720
724 IZ=VAL(X$)
730 ONIZGOTO 800,950,1100,1200,600,1800,23
      00
800 GOSUB5200

```

# GESTIONE CONTO CORRENTE

```

860 IFT=KTHEN930
870 GOSUB5300
880 IFZ1=0THEN600
885 W%=1
890 GOSUB5700
892 PRINT"(CLEAR)"
895 GOTO 860
930 PRINT"ARCHIVIO PIENO":GOTO 630
950 GOSUB5200
955 PRINT"(CLEAR)"
960 PRINT"NUMERO DI TRANSAZIONE DA VARIARE
"
961 PRINT" (O PER FINE ATTIVITA')      ?";
962 GOSUB6040
964 I=Q:IFI<0ORI>TTHEN975
966 IFI=0THEN600
967 PRINT"GIORNO ? "":GOSUB6040
968 Z1=Q:GOSUB5320
970 GOTO 995
975 GOSUB6100
977 GOTO 960
995 W%=2
1000 GOSUB5700
1002 GOTO 955
1100 GOSUB5000
1105 PRINT"INTRODURRE INUMERI DELLE TRANSAZ
IONI DA"
1110 PRINT"ANNULLARE (O PER STOP)"
1120 PRINT"? "":GOSUB6040
1122 N=Q:IFN>0ANDN<=TTHEN1140
1125 IFN=0GOTO 600
1130 PRINT" INESISTENTE":GOTO 1120
1140 PRINT:G(N-1)=0:M(N-1)=0
1150 D$(N-1)="":GOTO 1120
1200 GOSUB5000
1210 PRINT"      SCELTA VISUALIZZAZIONE"
1220 PRINT

```

# GESTIONE CONTO CORRENTE

```

1230 PRINT" 1- SALDO TOTALE"
1240 PRINT" 2- SALDI PARZIALI"
1250 PRINT" 3- TRANSAZIONI (MAX 15)"
1260 PRINT" 4- RIEPILOGO MOVIMENTI"
1270 PRINT" 5. FINE ATTIVITA"
1275 PRINT:PRINT" ";
1280 GETX$:IFX$=""THEN1280
1282 IFASC(X$)<49ORASC(X$)>53THEN1280
1284 IZ=VAL(X$)
1286 PRINT"(CLEAR)"
1290 ONIZGOTO 1300,1310,1550,1400,600
1300 T1=C:Y=T-1:GOTO 1340
1310 PRINT"SALDO A QUALE TRANS. ? ";;GOSUB 6
    040
1315 IZ=INT(Q):IFIZ>0ANDIZ<=TTHEN1330
1320 PRINT"NUMERO INESISTENTE":PRINT:GOTO
    1310
1330 T1=C:Y=IZ-1
1340 FOR L=0TOY:T1=T1+M(L):NEXT L:PRINT
1350 PRINT"ALLA TRANS. ";Y+1;"IL SALDO E'L
    ";T1
1360 PRINT:GOTO 1210
1400 PRINT:
1405 PRINT"MOVIMENTI DALLA TRANSAZ. ? ";;GOSUB
    6040
1410 FZ=INT(Q):IFFZ>0ANDFZ<=TTHEN1420
1415 PRINT"NUMERO INESISTENTE":PRINT:GOTO 1
    405
1420 PRINT"          ALLA TRANSAZ. ? ";;GOSUB
    6040
1425 HZ=INT(Q):IFHZ>0ANDHZ<=TTHEN1440
1430 PRINT"NUMERO INESISTENTE":PRINT:GOTO 1
    420
1440 B1=0:B2=0
1445 F1=FZ-1:F2=HZ-1
1450 FOR L=F1TOF2:IFM(L)<0THENB1=B1+M(L)
1460 IFM(L)>0THENB2=B2+M(L)

```



# GESTIONE CONTO CORRENTE

```

1465 NEXT L
1470 PRINT"PRELIEVI      L.  ";ABS(B1)
1480 PRINT"DEPOSITI     L.  ";ABS(B2)
1490 PRINT"TOTALE       L.  ";B1+B2
1500 PRINT:GOTO 1210
1550 PRINT"TRANSAZ. D'INIZIO ? ";:GOSUB6040

1555 IZ=INT(Q):IFIZ>0ANDIZ<=1THEN1565
1560 PRINT"NUMERO INESISTENTE ":PRINT:GOTO
      1550
1565 Y=IZ-1
1570 T1=Y+14:IFT1>T-1THENT1=T-1
1575 PRINT"(CLEAR)"
1580 PRINT". ";:FOR I=1TO37:PRINT". ";:NEXT I

1590 PRINT"."
1600 PRINT".NUM. . DATA. DENOMINAZIONE . IM
      PORTO ."
1605 PRINT". ";:FOR I=1TO37:PRINT". ";:NEXT I

1610 PRINT"."
1620 FOR I=YTOT1:F=G(I)-INT(G(I)/100)*100
1630 H=INT(G(I)/100):H=H-INT(H/100)*100
1632 X$=STR$(F):Y$=STR$(H)
1640 PRINT". ";I+1;TAB(6);". ";MID$(X$,2);TAB(
      9);"/";
1650 PRINTMID$(Y$,2);TAB(12);". ";D$(I);TAB(
      28);". ";
1660 PRINTM(I);TAB(38);"."
1670 NEXT I
1680 PRINT". ";:FOR I=1TO37:PRINT". ";:NEXT I

1690 PRINT". ":PRINT:PRINT
1700 PRINT"PREMERE + PER VISUALIZ. LE SUCCE
      SSIVE"
1710 PRINT"15 TRANSAZIONI,-- PER LE PRECEDEN
      TI 15 ,"
```

# GESTIONE CONTO CORRENTE

```

1715 PRINT"QUALSIASI TASTO PER NUOVA ATTIVI
    TA'"
1720 GETX$:IFX$=""THEN1720
1725 IFX$="+"THEN1750
1730 IFX$="-"THEN1760
1735 GOTO 1200
1750 Y=T1:GOTO 1570
1760 T1=Y:Y=T1-14:IFY<0THENY=0
1762 GOTO 1575
1765 GOTO 1200
1800 IFT>0THEN1805
1802 PRINT"ARCHIVIO VUOTO":PRINT:GOTO 640
1805 I=-1
1810 I=I+1:IFG(I)=-1THEN1830
1820 IFG(I)=0THENG(I)=999999
1825 GOTO 1810
1830 I=-1
1840 I=I+1:IFG(I+1)=-1THEN2000
1850 IFG(I)<=G(I+1)THEN1840
1855 F=G(I+1):G(I+1)=G(I):G(I)=F
1860 X$=D$(I+1):D$(I+1)=D$(I):D$(I)=X$
1870 F=M(I+1):M(I+1)=M(I):M(I)=F
1880 GOTO 1830
2000 I=-1:T=0
2010 I=I+1:IFG(I)=-1ORG(I)=999999THEN2030
2020 T=T+1:GOTO 2010
2030 G(I)=-1
2040 PRINT"(CLEAR)"
2041 PRINT"ORDINAMENTO ESEGUITO"
2050 PRINT"RICHIEDERE NUOVA STAMPA":PRINT:GOTO
    640
2300 GOSUB5030
2305 PRINT"      - RIAVV. LA CASSETTA(REW -ST
    OP)'"
2310 GOSUB5100
2320 OPEN1,1,2,"CONTO"
2330 PRINT#1,A$

```

## GESTIONE CONTO CORRENTE

```

2331 IFST>0THEN5500
2333 PRINT#1,C
2334 IFST>0THEN5500
2336 N=0
2360 PRINT#1,G(N)
2361 IFST>0THEN5500
2362 IFG(N)=-1THEN2382
2363 PRINT#1,D$(N)
2364 IFST>0THEN5500
2366 PRINT#1,M(N)
2367 IFST>0THEN5500
2370 N=N+1:GOTO 2360
2382 CLOSE1
2390 END
5000 PRINT"(CLEAR)":PRINT:PRINT:PRINT:RETURN

5010 PRINT:PRINT"  SCELTA ATTIVITA'"
5015 PRINT:RETURN
5030 GOSUB5000
5040 PRINT"  ESEGUIRE LE SEGUENTI AZIONI"
5050 PRINT:RETURN
5060 PRINT"    - RIAVVOLGERLA (REW -STOP)"
5065 RETURN
5100 PRINT"    - PREMERE SULLA TASTIERA QUA
    LSIASI"
5110 PRINT"      TASTO"
5115 GETX$:IFX$=""THEN5115
5120 RETURN
5130 PRINT"ERRORE IN LETTURA.PREMERE STOP S
    ULLA"
5140 PRINT"TASTIERA E SULLA CASSETTA E RICO
    MIN-"
5150 PRINT"CIARE":RETURN
5200 GOSUB5000
5205 PRINT"IMPOSTARE I DATI RICHIESTI ED IN
    OL-"
5207 PRINT"TRARLI PREMENDO IL TASTO RETURN"

```

# GESTIONE CONTO CORRENTE

```

5210 PRINT:RETURN
5300 PRINT
5302 PRINT"GIORNO (O PER FINE ATTIVITA') ?
";
5305 GOSUB6040
5310 Z1=Q:IFZ1=0THEN5445
5320 IFZ1<32THEN5340
5325 GOSUB6100
5330 GOTO 5302
5340 PRINT"MESE (IN NUMERI) ? ";
5345 GOSUB6040
5350 Z2=Q:IFZ2<13ANDZ2>0THEN5370
5355 GOSUB6100
5360 GOTO 5340
5370 PRINT"ANNO ? ";
5375 GOSUB6040
5380 Z3=Q:IFZ3>1900THEN5400
5385 GOSUB6100
5390 GOTO 5370
5400 PRINT"DENOMIN.(MAX 15 CAR)?";
5405 GOSUB6000
5410 X$=Q$
5415 PRINT"IMPORTO (-SE PRELIEVO) ? ";
5420 GETP$:IFP$=""THEN5420
5425 IFASC(P$)=45THEN5437
5427 Q$=""
5430 GOSUB6060
5435 Z4=Q:RETURN
5437 PRINTP$;
5438 GOSUB6040
5440 Z4=-Q
5445 RETURN
5500 PRINT"ERRORE IN SCRITTURA.ATTENDERNE L
A"
5510 PRINT"FINE E RIPETERE LA RICHIESTA"
5520 GOTO 640
5700 IFW%=1THEN5730

```

# GESTIONE CONTO CORRENTE

```

5720 IFW%=1THEN5730
5724 N=I-1:GOTO 5740
5730 T=T+1:N=T-1:G(N+1)=-1
5740 M(N)=Z4:D$(N)=LEFT$(X$,15)
5750 G(N)=(Z3-INT(Z3/100)*100)*10000+Z2*100
      +Z1
5760 RETURN
6000 Q$=""
6010 GETP$:IFP$=""THEN6010
6020 IFASC(P$)=13THEN6035
6030 PRINTP$;:Q$=Q$+P$:GOTO 6010
6035 PRINT:RETURN
6040 Q$=""
6050 GETP$:IFP$=""THEN6050
6060 IFASC(P$)=13THEN6090
6070 IFASC(P$)<48ORASC(P$)>57THEN6050
6080 PRINTP$;:Q$=Q$+P$:GOTO 6050
6090 PRINT:Q=VAL(Q$):RETURN
6100 PRINT"DATO INCOERENTE":RETURN

```

# INDICE

## INDICE

|              |        |
|--------------|--------|
| Introduzione | pag. 1 |
|--------------|--------|

### CAPITOLO PRIMO

|                    |     |
|--------------------|-----|
| La tastiera        | " 3 |
| Editing del CBM 64 | " 4 |
| Il tasto Commodore | " 7 |

### CAPITOLO SECONDO

|                           |      |
|---------------------------|------|
| Gli elementi fondamentali | " 9  |
| L'istruzione PRINT        | " 9  |
| Operatori                 | " 10 |
| Termini e operatori       | " 14 |
| Parentesi                 | " 16 |

### CAPITOLO TERZO

|                            |      |
|----------------------------|------|
| La programmazione          | " 18 |
| Le righe di un programma   | " 21 |
| Tecniche di programmazione | " 24 |
| L'istruzione INPUT         | " 24 |

### CAPITOLO QUARTO

|                          |      |
|--------------------------|------|
| Funzioni matematiche     | " 31 |
| ABS                      | " 31 |
| INT                      | " 32 |
| SGN                      | " 33 |
| SQR                      | " 33 |
| EXP                      | " 34 |
| LOG                      | " 34 |
| ATN                      | " 35 |
| COS                      | " 35 |
| SIN                      | " 36 |
| TAN                      | " 36 |
| Funzioni non matematiche | " 36 |

|                     |      |    |
|---------------------|------|----|
| PEEK                | pag. | 36 |
| POKE                | "    | 37 |
| USR                 | "    | 38 |
| FRE                 | "    | 38 |
| CLR                 | "    | 39 |
| Funzioni di stampa  | "    | 40 |
| POS                 | "    | 40 |
| SPC                 | "    | 40 |
| TAB                 | "    | 41 |
| L' orologio interno | "    | 42 |
| Funzioni definibili | "    | 43 |

## CAPITOLO QUINTO

|                          |   |    |
|--------------------------|---|----|
| Il comando LIST          | " | 46 |
| CONT                     | " | 47 |
| GET                      | " | 48 |
| GOTO e IF...THEN         | " | 50 |
| Anelli condizionati      | " | 52 |
| GOSUB e subroutines      | " | 53 |
| L' istruzione FOR...NEXT | " | 56 |
| Anelli consecutivi       | " | 58 |

## CAPITOLO SESTO

|                               |   |    |
|-------------------------------|---|----|
| Le variabili                  | " | 59 |
| Tecniche di programmazione    | " | 61 |
| Operazioni e comandi stringhe | " | 62 |
| STR\$                         | " | 62 |
| VAL                           | " | 63 |
| CHR\$                         | " | 63 |
| ASC                           | " | 64 |
| Segmenti di stringa           | " | 65 |
| LEFT\$                        | " | 65 |
| RIGHT\$                       | " | 66 |
| MID\$                         | " | 66 |
| LEN                           | " | 68 |
| Le matrici                    | " | 68 |
| DIM                           | " | 68 |
| Consumo della memoria         | " | 69 |
| DATA, READ ,RESTORE           | " | 70 |



## CAPITOLO SETTIMO

|                          |         |
|--------------------------|---------|
| La funzione RANDOM       | pag. 73 |
| Gli operatori logici     | " 75    |
| AND                      | " 76    |
| NOT                      | " 76    |
| OR                       | " 76    |
| I tasti funzione         | " 77    |
| Programma tasti funzione | " 78    |
| CMD                      | " 79    |
| END                      | " 79    |
| LET                      | " 80    |
| NEW                      | " 80    |
| REM                      | " 80    |
| STOP                     | " 81    |
| SYS                      | " 81    |
| WAIT                     | " 81    |
| I colori                 | " 82    |
| Mappa dei colori         | " 86    |

## CAPITOLO OTTAVO

|                              |      |
|------------------------------|------|
| La grafica                   | " 88 |
| Locazioni grafiche           | " 88 |
| Selezione dei banchi memoria | " 89 |
| Memoria di schermo           | " 91 |
| Memoria colore               | " 92 |
| Memoria carattere            | " 92 |
| Character generator ROM      | " 92 |

## CAPITOLO NONO

|                              |       |
|------------------------------|-------|
| Modo standard dei caratteri  | " 97  |
| Definizione del carattere    | " 98  |
| Caratteri programmabili      | " 99  |
| Grafica in multicolore       | " 104 |
| Multi-colore                 | " 105 |
| Extended background          | " 108 |
| Il BIT MAPPING nella grafica | " 110 |
| Standard Bit Map Mode        | " 111 |
| Multi colour Bit Map         | " 112 |

Lo scrolling lento pag. 113

## CAPITOLO DECIMO

|                           |   |     |
|---------------------------|---|-----|
| Gli Sprites               | " | 117 |
| Come definire uno Sprite  | " | 118 |
| Puntatori degli Sprites   | " | 119 |
| Messa in funzione Sprites | " | 121 |
| I colori negli Sprites    | " | 122 |
| Modo multicolore          | " | 123 |
| Lo Sprite in Multicolore  | " | 124 |
| Expanded Sprites          | " | 125 |
| Posizione degli sprites   | " | 126 |
| Posizionamento verticale  | " | 128 |

## CAPITOLO UNDICESIMO

|                              |   |     |
|------------------------------|---|-----|
| Posizionamento orizzontale   | " | 130 |
| Sprite non espanso           | " | 131 |
| Sprite espanso               | " | 131 |
| Priorita' di visualizzazione | " | 132 |
| Collision detect             | " | 133 |
| Collision register           | " | 134 |
| Altre possibilita' grafiche  | " | 135 |
| Raster register              | " | 136 |
| Interrupt status register    | " | 136 |

## CAPITOLO DODICESIMO

|                               |   |     |
|-------------------------------|---|-----|
| Altro modo per Sprites        | " | 138 |
| Programma commentato          | " | 139 |
| Posizionamento Sprites shermo | " | 144 |

## CAPITOLO TREDICESIMO

|                          |   |     |
|--------------------------|---|-----|
| Programmazione dei suoni | " | 148 |
| Controllo volume         | " | 151 |
| Frequenza d'onda sonora  | " | 151 |
| Uso di voci multiple     | " | 152 |
| Controllo voci multiple  | " | 155 |
| Cambio forme d' onda     | " | 156 |

|                           |          |
|---------------------------|----------|
| Il generatore di envelope | pag. 158 |
| Filtri                    | " 162    |

## CAPITOLO QUATTORDICESIMO

|                               |       |
|-------------------------------|-------|
| Introduzione alle periferiche | " 164 |
| I Files                       | " 165 |
| Il concetto                   | " 166 |
| File programmi                | " 167 |
| File Mailing List             | " 170 |
| Comandi per files programmi   | " 171 |
| LOAD                          | " 171 |
| Approfondimento sui files     | " 172 |
| VERIFY                        | " 173 |
| SAVE                          | " 174 |

## CAPITOLO QUINDICESIMO

|                               |       |
|-------------------------------|-------|
| Data files                    | " 176 |
| Records e fields              | " 176 |
| Trasferimento dati            | " 177 |
| Files logici e unita' fisiche | " 179 |
| Il canale                     | " 181 |
| Indirizzo secondario          | " 184 |
| Fisical unit status           | " 186 |
| Registro di Status            | " 186 |
| Manipolazione dati            | " 191 |
| Il formato dei Files          | " 196 |
| Fields numerici               | " 196 |
| Lettura dati da cassetta      | " 207 |

## CAPITOLO SEDICESIMO

|                             |       |
|-----------------------------|-------|
| Programmazione di file dati | " 210 |
| Chiusura di un file         | " 213 |
| Come accedere ai data files | " 215 |
| Approfondimento H/S         | " 218 |
| Operazioni su cassetta      | " 219 |
| Files Binari                | " 221 |
| Files ASCII                 | " 222 |
| Controllo di errore         | " 225 |

## CAPITOLO DICIASSETTESIMO

Messaggi di errore e rimedi pag. 228

## CAPITOLO DICIOTTESIMO

|                            |   |     |
|----------------------------|---|-----|
| Il linguaggio Macchina     | " | 239 |
| I registri del 6510        | " | 243 |
| L' Accumulatore            | " | 244 |
| I registri indice          | " | 244 |
| Program counter            | " | 244 |
| Lo status register         | " | 245 |
| La memoria del CBM64       | " | 246 |
| I cartridges               | " | 249 |
| Le routines Kernal         | " | 251 |
| Attività del S.O. e Kernal | " | 252 |
| Come utilizzare le Kernal  | " | 253 |
| Tavola Kernal              | " | 256 |
| Descrizione delle Kernal   | " | 259 |

|                   |   |     |
|-------------------|---|-----|
| TAVOLE DI MEMORIA | " | 307 |
|-------------------|---|-----|

|                             |   |     |
|-----------------------------|---|-----|
| VIDEO INTERFACE CONTROLLER  | " | 318 |
| SOUND INTERFACE DEVICE      | " | 321 |
| COMPLEX INTERFACE ADAPTER 1 | " | 324 |
| COMPLEX INTERFACE ADAPTER 2 | " | 327 |
| TAVOLA ISTRUZIONI 6510      | " | 331 |

|                             |   |     |
|-----------------------------|---|-----|
| Abbreviazione parole chiave | " | 340 |
| Codici dello schermo video  | " | 342 |
| Codici ASCII e CHR\$        | " | 344 |
| Mappa memoria schermo       | " | 346 |
| Mappa memoria colore        | " | 347 |
| Valore note musicali        | " | 348 |

## PROGRAMMI

|                         |   |     |
|-------------------------|---|-----|
| Data base               | " | 353 |
| Gestione conto corrente | " | 365 |

SEGNALAZIONE ERRORI \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

DESIDERO RICEVERE

- ☐ CATALOGO EVM
- ☐ ESSERE INSERITO NELLA MAILING LIST E RICEVERE TUTTI GLI AGGIORNAMENTI
- ☐ PROGRAMMI Cross Reference, Data Base, Gestione conto corrente + SPRITE GENERATOR+SOUND GENERATOR + DEMO
- ☐ NASTRO            L. 15.000 (IVA compresa)
- ☐ DISCO            L. 20.000 (IVA compresa)
- ☐ Effettuate la spedizione contrassegno aggiungendo L. 4.000 per contributo spese postali.
- ☐ Allego assegno per l' importo piu' L. 2.000 per contributo spese postali.

NOME \_\_\_\_\_

VIA \_\_\_\_\_

CAP \_\_\_\_\_ CITTA' \_\_\_\_\_

**SPEDIRE IN BUSTA A:**

**E.V.M. COMPUTERS**

**VIA MARCONI N°9/A**

**52025 MONTEVARCHI (AR)**

---







# GUIDA AL COMMODORE 64

*EURO*