CAPITOLO 5

dal basic al linguaggio macchina

- Che cos'è un Linguaggio Macchina?
- Come si scrivono i programmi in Linguaggio Macchina?
- Notazione Esadecimale
- Modi di Indirizzamento
- Indicizzazione
- Sottoprocedure
- Suggerimenti utili per il Principiante
- Un compito più impegnativo
- Insieme delle istruzioni del microprocessore MCS6510
- Gestione della memoria sul Commodore 64
- Il KERNAL
- Attività di inizializzazione del KERNAL
- Uso del Linguaggio Macchina da BASIC
- Mappa della memoria del Commodore 64

CHE COS'È IL LINGUAGGIO MACCHINA?

Il cuore di ogni microcomputer e' costituito da un microprocessore centrale, un particolare microcircuito (chip) che rappresenta il cervello del computer. Il COMMODORE 64 non fa eccezione. Ciascun microprocessore comprende le istruzioni scritte nel proprio linguaggio. Per essere piu' precisi, il Linguaggio Macchina e' il solo linguaggio di programmazione che il COMMODORE 64 comprenda. Esso e' il linguaggio NATURALE della macchina.

Se il Linguaggio Macchina e' il solo linguaggio che il COMMODORE 64 comprende, allora come puo' capire il linguaggio di programmazione CBM BASIC ? Il CBM BASIC NON e' il Linguaggio Macchina del COMMODORE 64. Che cosa, quindi, fa si' che il COMMODORE 64 capisca istruzioni CBM BASIC come PRINT o GOTO ?

Per avere una risposta, si deve vedere per prima cosa, cio' che accade dentro il COMMODORE 64. A prescindere dal microprocessore, che e' il cervello del COMMODORE 64, c'e' un programma in Linguaggio Macchina che e' memorizzato in uno speciale tipo di memoria in modo tale da non poter essere modificato. E, cosa ancora piu' importante, esso non viene perduto quando il COMMODORE 64 viene spento, diversamente da quanto accade ad un programma scritto da un Utente. Questo programma in Linguaggio Macchina e' chiamato SISTEMA OPERATIVO del COMMODORE 64. Il COMMODORE 64 sa che cosa fare quando e' acceso perche' il suo Sistema Operativo "gira" automaticamente.

Il Sistema Operativo e' incaricato di "organizzare" tutta la memoria della macchina affinche' svolga varie mansioni. Oltre ad un certo numero di altre funzioni, esso considera inoltre quali caratteri vengono digitati sulla tastiera e li riporta sullo schermo. Il Sistema Operativo si puo' pensare come "l'intelligenza e la personalita'" del COMMODORE 64 (o di ogni altro computer di quella portata). Cosi', quando si accende il COMMODORE 64, il Sistema Operativo prende il controllo della macchina, e dopo aver terminato il suo compito, dice:

```
READY.
■
```

Il Sistema Operativo del COMMODORE 64 permette, quindi, di digitare sulla tastiera e di usare l'EDITOR SCHERMO del COMMODORE 64. L'Editor di Schermo permette di muovere il cursore, di cancellare (DEL), di immettere (INS), ecc., ed e', percio', l'unica parte del Sistema Operativo incorporata a vantaggio dell'Utente.

Tutti i comandi che sono disponibili nel CBM BASIC sono semplicemente riconosciuti da un altro vasto programma in linguaggio macchina incorporato nel COMMODORE 64. Questo vasto programma "elabora" (RUN) il segmento appropriato del linguaggio macchina collegato al comando in CBM BASIC che sta per essere eseguito. Questo programma e' chiamato l'INTERPRETE BASIC, perche' interpreta ciascun comando, uno ad uno, a meno che incontri un comando che non capisce, nel qual caso appare il familiare messaggio di:

```
?SYNTAX ERROR

READY.
■
```

A CHE COSA ASSOMIGLIA IL CODICE MACCHINA?

Si dovrebbe avere familiarita' con i comandi PEEK e POKE del linguaggio CBM BASIC, con i quali si modificano le locazioni di memoria. Probabilmente, sono gia' stati usati per la grafica sullo schermo e per gli effetti sonori. Ciascuna locazione di memoria ha il proprio numero di identificazione. Tale numero e' conosciuto come l'"indirizzo" di una locazione di memoria. Se si immagina la memoria del COMMODORE 64 come una strada circondata da edifici, allora il numero su ciascuna porta e', ovviamente, l'indirizzo. Vediamo ora per quali scopi vengono utilizzate le varie parti della strada.

SEMPLICE MAPPA DELLA MEMORIA DEL COMMODORE 64

INDIRIZZO	DESCRIZIONE
0 & 1	Registri del 6510
2	Inizio della memoria
Fino a: 1023	Memoria usata dal Sistema Operativo
Da 1024 a 2039	Memoria dello schermo
Da 2040 a 2047	Puntatori alle animazioni
Da 2048 a 40959	Memoria UTENTE. Qui vengono memorizzati il BASIC ed i programmi in Linguaggio Macchina (o entrambi)
Da 40960 a 49151	Interprete CBM BASIC (8K)
Da 49152 a 53247	Area RAM per programmi speciali
Da 53248 a 53294	Registri del VIC-II
Da 55296 a 56296	RAM colore
Da 56320 a 57343	Registri di I/O
Da 57344 a 65535	Sistema Operativo CBM KERNAL (8K)

Se non si comprende il significato della descrizione appena data di ciascuna parte della memoria, risultera' piu' chiaro in altre parti di questo Manuale. I programmi in Linguaggio Macchina sono costituiti da istruzioni che possono o no avere operandi (parametri) associati ad essi. Ciascuna istruzione occupa una locazione di memoria, ed ogni operando e' contenuto in una o due locazioni contigue all'istruzione. Nei programmi in BASIC, parole come PRINT e GOTO vanno ad occupare solamente una locazione di memoria, invece che una per ogni carattere della parola. I contenuti della locazione che rappresenta una particolare parola riservata del BASIC e' chiamato "token". Nel Linguaggio Macchina, ci sono "token" diversi per differenti istruzioni, anch'essi occupanti un solo byte (locazione di memoria = byte).

Le istruzioni del Linguaggio Macchina sono molto semplici. Percio', ciascuna istruzione individuale non puo' realizzare moltissimo. Le istruzioni in Linguaggio Macchina modificano il contenuto di una locazione di memoria, oppure cambiano uno dei registri interni

(particolari locazioni di memoria) del microprocessore. I registri interni rappresentano la vera base del linguaggio macchina.

I REGISTRI DEL MICROPROCESSORE 6510

ACCUMULATORE

Questo e' il registro piu' importante del microprocessore. Diverse istruzioni in Linguaggio Macchina consentono di copiare il contenuto di una locazione di memoria nell'accumulatore, di copiare il contenuto dell'accumulatore in una locazione di memoria, di modificare direttamente il contenuto dell'accumulatore o di qualche altro registro, senza interessare la memoria. E' l'unico registro fornito di istruzioni per eseguire calcoli aritmetici.

REGISTRO INDICE X

Ci sono istruzioni relative a quasi tutte le trasformazioni che si possono fare all'accumulatore. Ma ci sono altre istruzioni per cose che solamente il registro X puo' effettuare. Diverse istruzioni in Linguaggio Macchina permettono di copiare il contenuto di una locazione di memoria del registro X, di copiare il contenuto dal registro X in una locazione di memoria e di modificare il contenuto del registro X o di qualche altro registro direttamente, senza interessare la memoria.

REGISTRO INDICE Y

Ci sono istruzioni per quasi tutte le trasformazioni che si possono effettuare sull'accumulatore e sul registro X. Ma ci sono altre istruzioni che solo il registro Y puo' eseguire. Numerose istruzioni in Linguaggio Macchina consentono di copiare il contenuto di una locazione di memoria nel registro Y, di copiare il contenuto del registro Y in una locazione di memoria, e di modificare il contenuto di Y o di qualche altro registro direttamente, senza interessare altra memoria.

REGISTRO DI STATO

Registro costituito da 8 "flag" (flag = indicatore di un fatto, avvenuto o no).

CONTATORE DI PROGRAMMA (PROGRAM COUNTER)

Contiene l'indirizzo dell'istruzione corrente in Linguaggio Macchina che sta per essere eseguita. Poiche' il Sistema Operativo e' sempre "caricato" (RUNning) sul COMMODORE 64 (o su qualche altro computer con queste caratteristiche), il contatore di programma viene sempre modificato. Puo' essere fermato solamente dall'arresto del microprocessore.

PUNTATORE ALLO STACK (STACK POINTER)

Contiene la locazione del primo posto vuoto sullo "stack" (pila). Lo stack viene usato dai programmi in Linguaggio Macchina e dal computer per memorizzazioni temporanee.

PORTA DI INPUT/OUTPUT

Questo registro appare nella locazioni di memoria 0 (per il registro DATA DIRECTION) e 1 (per la PORTA attuale). Si tratta di una porta di input/outpu a 8 bit. Sul COMMODORE 64 questo registro e' usato per la gestione della memoria, per permettere al circuito di controllare piu' di 64K di memoria RAM e ROM. I particolari di questi registri verranno esposti piu' avanti.

COME SI SCRIVONO I PROGRAMMI IN LINGUAGGIO MACCHINA?

Poiche' i programmi in Linguaggio Macchina risiedono in memoria, e dato che non e' facile scrivere ed editare programmi in Linguaggio Macchina sul COMMODORE 64, per fare cio' si deve usare un programma particolare, oppure scrivere un programma in BASIC che permetta di scrivere in Linguaggio Macchina.

I piu' comuni metodi usati per scrivere programmi in Linguaggio Macchina sono i programmi assembler. Questi pacchetti ("packages") consentono di scrivere istruzioni in Linguaggio Macchina in un formato mnemonico standardizzato, che rende il programma in Linguaggio Macchina molto piu' leggibile di un flusso di numeri. Riassumendo: un programma che permette di scrivere in formato mnemonico programmi in Linguaggio Macchina e' chiamato assemblatore. Per inciso, un programma che risultasse scritto in linguaggio macchina in formato mnemonico, e' chiamato disassembler. A disposizione del COMMODORE 64 c'e' una cartuccia (con assembler/disassembler, ecc.), costruita dalla Commodore, per il controllo dei programmi in Linguaggio Macchina:

64MON

La cartuccia 64MON permette di uscire dal mondo del CBM BASIC, per entrare nel Linguaggio Macchina. Essa e' in grado di visualizzare il contenuto dei registri interni del microprocessore 6510, permettendo quindi di visualizzare parti della memoria e di cambiarle sullo schermo tramite l'Editor di Schermo. Incorpora anche l'assembler ed il disassembler, oltre a molte altre caratteristiche che permettono di scrivere e di editare facilmente programmi in Linguaggio Macchina. Per scrivere in Linguaggio Macchina NON E' NECESSARIO usare un assembler, ma con quest'ultimo il compito diventa considerevolmente piu' facile. Se si desidera scrivere programmi in Linguaggio Macchina, e' fortemente consigliabile acquistare un assembler, in assenza del quale si dovra' probabilmente "caricare" (POKE) il programma in Linguaggio Macchina nella memoria, che e' totalmente intrattabile. Da ora in poi, questo manuale riportera' gli esempi nel formato utilizzato dal 64MON. Quasi tutti i formati assembler sono gli stessi, quindi gli esempi in linguaggio macchina riportati sono quasi certamente compatibili con qualsiasi assembler. Ma prima di esporre altre caratteristiche del 64MON, occorre spiegare il sistema di numerazione esadecimale.

NOTAZIONE ESADECIMALE

La notazione esadecimale e' usata dalla maggior parte dei programmatori in linguaggio macchina quando trattano un numero o un

indirizzo in linguaggio macchina.

Alcuni assembler consentono di riferirsi ad indirizzi e numeri in decimale (base 10), in binario (base 2), in ottale (base 8) ed in esadecimale (base 16 - indicata in questo manuale con "HEX"). Questi assembler effettuano la conversione da una base all'altra in modo automatico.

Probabilmente, l'esadecimale risulta un po' difficile da capire all'inizio, ma come la maggior parte delle cose, con la pratica non ci vorra' molto tempo per apprenderlo.

Osservando i numeri decimali (base 10), si puo' vedere che ciascuna cifra risulta compresa tra zero ed il numero della base diminuito di uno (in questo caso, 9). QUESTO E' VERO PER TUTTE LE BASI NUMERICHE. I numeri binari (base 2) sono rappresentati con cifre comprese fra zero e uno (quest'ultimo valore si ottiene diminuendo la base 2 di una unita'). Similmente, i numeri esadecimali hanno cifre comrese fra zero e quindici, solo che non ci sono singole cifre in grado di rappresentare i numeri da dieci a quindici; pertanto, vengono utilizzate le prime sei cifre dell'alfabeto:

DECIMALE	ESADECIMALE	BINARIO
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	A	00001010
11	B	00001011
12	C	00001100
13	D	00001101
14	E	00001110
15	F	00001111
16	10	00010000

Un esempio di come costruire una base (numero) decimale puo' essere il seguente:

Base elevata a
potenze decrescenti... . 10^3 10^2 10^1 10^0

Equivale a ..... .1000 100 10 1

Esempio: 4569 (base 10) 4 5 6 9

$=(4\times1000)+(5\times100)+(6\times10)+9$

In maniera del tutto analoga si puo' costruire una base (numero) esadecimale:

Base elevata a
potenze decrescenti... 16^3 16^2 16^1 16^0

Equivale a 4096 256 16 1

Esempio: 11D9 (base 16) 1 1 D 9

$=1\times4096+1\times256+13\times16+9$

Quindi, 4569 (base 10) = 11D9 (base 16).
L'intervallo delle locazioni di memoria indirizzabili (come visto precedentemente) e' 0...65535; in notazione esadecimale, diviene 0...FFFF.
Di solito i numeri esadecimali sono preceduti dal simbolo "$", in modo da poterli distinguere dai numeri decimali. Numeri esadecimali possono essere osservati, usando la cartuccia 64MON, visualizzando il contenuto di una parte di memoria. Digitando:

```
B
      PC SR AC XR YR SP
.; 0401 32 04 5E 00 F6      (possono comparire anche valori differenti)

.M 0000 0020.  (seguito da RETURN ).
```

compaiono file di 6 numeri "HEX". Il primo numero di 4 cifre rappresenta l'indirizzo del primo byte di memoria visualizzato; gli altri cinque numeri costituiscono il contenuto reale della locazione di memoria che inizia da quell'indirizzo.
E' cosigliabile cercare di imparare a "pensare" in esadecimale: non e' poi cosi' difficile, poiche' non e' necessaria la riconversione in decimale; infatti, non fa alcuna differenza dire che un particolare valore e' memorizzato in)14ED anziche' 5357 (corrispondente valore decimale).

LA PRIMA ISTRUZIONE IN LINGUAGGIO MACCHINA

LDA - Carica l'Accumulatore

Nel linguaggio assembly del 6510, i codici mnemonici sono quasi sempre di tre caratteri. LDA sta per "carica l'accumulatore con..."; cio' che deve essere caricato nell'accumulatore e' specificato dal parametro (o dai parametri) associato all'istruzione stessa. L'assembler sa che il "token" e' rappresentato dal corrispondente codice mnemonico, e quando "assembla" un'istruzione trasferisce semplicemente in memoria (a qualunque indirizzo specificato) il "token" ed i parametri associati. Alcuni assembler ritornano messaggi di errore, oppure avvertono quando si cerca di assemblare qualcosa che l'assembler o il microprocessore 6510 non possono fare.
Se al parametro associato all'istruzione si antepone il simbolo "#", cio' significa che si intende caricare il registro specificato nell'istruzione con il "valore" che segue "#". Per esempio:

Questa istruzione mette $05 (5 esadecimale) nel registro accumulatore. Nell'indirizzo specificato per questa istruzione, l'assembler carica $A9 (che in questo caso e' il "token"

dell'istruzione) nell'indirizzo specificato, e $05 nella prima locazione seguente quella contenente l'istruzione stessa ($A9). Se il parametro che deve essere utilizzato e' preceduto da "#", cioe' se il parametro rappresenta un "valore" piuttosto che il contenuto di una locazione di memoria o di un altro registro, allora l'istruzione si dice "immediata". Per illustrare meglio quanto detto, vediamo un altro modo.

Se si vuole trasferire il contenuto della locazione di memoria $102E nell'accumulatore, si deve usare la seguente istruzione "assoluta":

```
LDA $102E
```

L'assembler e' in grado di distinguere tra i due differenti modi in quanto il secondo non ha "#" prima del parametro. Il microprocessore 6510 puo' distinguere tra il modo immediato ed il modo assoluto per il fatto che sono formati da "token" leggermente diversi: LDA (immediato) ha come "token" $09, mentre LDA (assoluto) ha come "token" $AD.

Il codice mnemonico che rappresenta un'istruzione indica in genere l'azione svolta dall'istruzione stessa. Consideriamo ad esempio un'altra istruzione: LDX; che cosa pensiamo che faccia ?

Se si risponde "carica il registro X con...", allora...siamo i primi della classe ! Altrimenti, non c'e' di che preoccuparsi, imparare il linguaggio macchina richiede pazienza, e non puo' essere appreso in un solo giorno.

I vari registri interni possono essere pensati come locazioni di memoria speciali, poiche' possono contenere anche un solo byte. Il sistema di numerazione binario (base 2) non richiede particolari spiegazioni: valgono per esso le stesse regole riportate per i precedenti sistemi esadecimale e decimale; ci limitiamo a ricordare che un "bit" rappresenta una cifra binaria, e che otto bit costituiscono un byte. Cio' significa che il massimo numero che un byte puo' contenere e' dato dal piu' grande numero esprimibile con otto cifre binarie, vale a dire 11111111 (binario), equivalente a $FF (esadecimale) e a 255 (decimale). Probabilmente, ci si potra' meravigliare del fatto che in una locazione di memoria possa essere contenuto un numero compreso solamente fra 0 e 255. Se si prova a digitare POKE 7680,260 (istruzione BASIC che dice di "caricare il numero 260 nella locazione di memoria 7680), allora, dato che l'interprete BASIC sa che una locazione di memoria puo' contenere un numero comreso solamente fra 0 e 255, il COMMODORE 64 replichera':

```
?ILLEGAL QUANTITY ERROR

READY.
■
```

Se il limite di un byte e' $FF (HEX), come viene espresso in memoria il parametro indirizzo contenuto nell'istruzione assoluta "LDA $10E" ? Elementare: con due byte (ovvio, visto che uno non basta...). Le due cifre basse (le piu' a destra) dell'indirizzo esadecimale formano il "byte basso" dell'indirizzo, quelle alte (le piu' a sinistra) il "byte alto".

Il 6510 richiede che ciascun indirizzo venga specificato ponendo per primo il byte basso, seguito da quello alto. Cio' significa che l'istruzione "LDA $102E" e' rappresentata in memoria dai tre valori consecutivi:

$AD, $2E, $10

Per poter scrivere un programma e' necessario conoscere ancora un'istruzione: BRK. Una completa descrizione di questa istruzione si trova in un Manuale di Programmazione del 6502 M.O.S. Per il momento, si puo' pensare a questa istruzione come corrispondente in linguaggio macchina della END del BASIC.

Se si scrive un programma usando la cartuccia 64MON, sistemando l'istruzione BRK alla fine del programma, questa istruzione, una volta eseguito e terminato il programma, fa tornare alla 64MON. Se cio' non accade, vuol dire che si e' verificato un errore nel programma, oppure l'istruzione BRK non e' stata mai raggiunta (proprio come se non si fosse mai raggiunta una END in un programma BASIC). Si capisce cosi' che se il COMMODORE 64 non avesse una chiave di STOP, non saremmo in grado di abortire i programmi BASIC!

IL PRIMO PROGRAMMA

Se si e' usata l'istruzione POKE in un programma BASIC per trasferire i caratteri sullo schermo, allora si e' gia' a conoscenza del fatto che i codici carattere per "modificare tramite POKE" sono diversi dai valori carattere del CBM ASCII. Se, ad esempio, digitiamo:

PRINT ASC("A") (premere poi RETURN)

il COMMODORE 64 risponde con:

```
65

READY.
■
```

Tuttavia, per inserire una "A" sullo schermo tramite l'istruzione POKE, digitare (il codice per "A" e' 1):

SHIFT CLR/HOME per pulire lo schermo

POKE 1024,1 (e RETURN) (1024 e' l'inizio della memoria schermo)

La lettera "P" dell'istruzione POKE dovrebbe essere ora una "A". Proviamo ora in linguaggio macchina: digitiamo quanto segue su 64MON (da questo momento, il cursore dovrebbe essere intermittente a fianco del"."):

.A 1400 LDA #$01 (premere poi RETURN)

Il COMMODORE 64 risponde con:

```
.A 1400 LDA #$01
.A 1402 ■
```

A questo punto digitiamo:

.A 1402 STA $0400

(l'istruzione STA memorizza il contenuto dell'accumulatore in una locazione di memoria specifica). La risposta del COMMODORE 64 e':

```
.A 1405 ■
```

Digitiamo infine:

```
.A 1405 BRK
```

Puliamo lo schermo e digitiamo:

```
G 1400
```

Se tutto e' stato eseguito correttamente, la "G" si trasforma in "A". Si e' scritto cosi' un primo programma in linguaggio macchina, che ha lo scopo di memorizzare un carattere ("A") nella prima locazione della memoria schermo. Possiamo a questo punto introdurre altre istruzioni e principi.

MODI DI INDIRIZZAMENTO

PAGINA ZERO

Come abbiamo gia' visto, gli indirizzi assoluti sono espressi in termini di byte alto e basso. Il byte alto si riferisce alla pagina di memoria. Per esempio, l'indirizzo $1637 si trova nella pagina $16 (22 decimale), mentre l'indirizzo $0277 si trova nella pagina $02 (2 decimale). Esiste, tuttavia, un particolare modo di indirizzamento, conosciuto come "indirizzamento di pagina zero", associato, come spiegato dallo stesso nome, all'indirizzamento delle locazioni di memoria di pagina zero. Questi indirizzi, percio', hanno SEMPRE il byte alto a zero. Questo modo di indirizzamento si aspetta solo un byte per la descrizione dell'indirizzo, invece dei due byte usati per un indirizzo assoluto. Il modo di indirizzamento di pagina zero dice al microprocessore di assumere zero come indirizzo alto, per cui puo' fare riferimento a locazioni di memoria i cui indirizzi vanno da $0000 a $00FF. Questo ora puo' sembrare poco importante, ma presto ci serviranno i principi dell'indirizzamneto di pagina zero.

LO STACK

Il microprocessore 6510 ha uno "stack" (pila), che viene usato sia dal programmatore che dal microprocessore per memorizzazioni temporanee, come ad esempio una lista ordinata di eventi. L'istruzione GOSUB del BASIC, che permette al programmatore di richiamare una sottoprocedura, deve tener presente in quale punto del programma sta per essere chiamata, in modo che, quando nella sottoprocedura viene eseguita l'istruzione RETURN, l'interprete BASIC "sappia" da quale punto del programma riprendere l'esecuzione. Quando incontra in un programma l'istruzione GOSUB, l'interprete BASIC, prima di mandare in esecuzione la sottoprocedura, ne carica la posizione attuale sullo "stack"; dopo l'esecuzione dell'istruzione RETURN, l'interprete preleva dallo "stack" l'informazione che gli comunica in quale punto del programma si trovava prima che fosse eseguita la sottoprocedura chiamata. L'interprete fa uso di istruzioni come PHA, che carica sullo "stack" il contenuto dell'accumulatore, e come PLA (opposta di PHA).

che preleva un valore dallo "stack" caricandolo nell'accumulatore. Anche il registro di stato puo' essere caricato o scaricato, facendo uso rispettivamente delle istruzioni PHP e PLP.

Lo stack e' lungo 256 byte ed e' allocato nella pagina 1 della memoria: si estende, quindi, da $0100 a $01FF, e nella memoria e' organizzato in senso inverso. In altre parole, la prima posizione dello "stack" ha indirizzo $01FF, e l'ultima $0100. Un altro registro del microprocessore 6510 e' il PUNTATORE ALLO STACK, che indica sempre la prima posizione disponibile sullo "stack". Quando un dato viene inserito nello stack, trova posto nella locazione indicata dal puntatore allo stack, quindi il puntatore allo stack viene fatto scendere alla prossima locazione (decrementato). Quando invece un dato viene tolto dallo stack, il puntatore allo stack viene incrementato ed il byte puntato da questo puntatore viene sistemato nel registro specificato.

Fino a questo punto sono state esaminate le istruzioni scritte in modo immediato, pagina zero ed assoluto; introduciamo ora il modo "implicito". Il modo implicito indica che l'informazione e' implicita nell'istruzione stessa, cioe' a quali registri, indicatori (flag) e memoria fa riferimento l'istruzione. Gli esempi che abbiamo visto sono PHA, PLA, PHP e PLP, che si riferiscono, rispettivamente, all'elaborazione dello stack (le prime due) ed ai registri di stato e dell'accumulatore (le seconde due)

NOTA: Da ora in avanti, indicheremo con X il registro X, con A l'accumulatore, con Y il registro Y, con S il puntatore allo stack e con P lo stato del processore.

INDICIZZAZIONE

L'indicizzazione occupa una parte importante nell'elaborazione del microprocessore 6510. Puo' essere definita come "la generazione di un indirizzo attuale ottenuto sommando all'indirizzo base il contenuto di entrambi i registri indice X e Y".

Per esempio, se X contiene $05 ed il microprocessore esegue un'istruzione LDA nel "modo indicizzato assoluto X" con un indirizzo base (ad esempio, $9000), allora la locazione reale riportata nel registro A e' data da $9000+$05=$9005. Il formato del codice mnemonico di un'istruzione indicizzata assoluta e' lo stesso di quello di un'istruzione assoluta, ad eccezione di una X o di una Y indicanti che l'indice viene sommato all'indirizzo.

ESEMPIO:

```
LDA $9000,X
```

Disponibili sul microprocessore 6510 ci sono modi di indirizzamento indicizzato assoluto, indicizzato di pagina zero, indicizzato indiretto ed indiretto indicizzato.

INDICIZZATO INDIRETTO

E' l'unico modo che permette l'uso del registro Y come indice. L'indirizzo reale puo' essere solamente nella pagina zero. Il modo dell'istruzione e' chiamato indiretto perche' l'indirizzo di pagina zero, specificato nell'istruzione, contiene il byte basso dell'indirizzo reale, ed il byte successivo a questo il byte alto.

ESEMPIO:

Supponiamo che la locazione $01 contenga $45 e la locazione $02 contenga $1E. Se viene eseguita l'istruzione per caricare l'accumulatore nel modo indicizzato indiretto, e se l'indirizzo di pagina zero specificato e' $01, allora l'indirizzo reale sara':

```
Byte basso = Contenuto di $01
Byte  alto = Contenuto di $02
Registro Y = $00
```

Quindi l'indirizzo reale sara' dato da $1045+Y=$1045.
L'intestazione di questo modo implica in effetti un principio indiretto, anche se a prima vista risulta difficile da afferrare. In un'altra forma, tale principio puo' suonare come segue: "Sto andando a recapitare questa lettera all'ufficio postale di indirizzo $01, MEMORY ST.. e l'indirizzo riportato sulla lettera si trova $05 abitazioni oltre $1600, MEMORY ST.". Traducendo in codice:

```
LDA #$00     - Carica l'indirizzo basso di base attuale
STA $02      - Imposta il byte basso del'indirizzo indiretto
LDA #$16     - Carica l'indirizzo indiretto alto
STA $03      - Imposta il byte alto dell'indirizzo indiretto
LDY #$05     - Imposta l'indice indiretto
LDA ($02),Y  - Carica indirettamente indicizzato da Y
```

INDIRETTO INDICIZZATO

E' l'unico modo che consente l'uso del registro X come indice. Questo modo e' analogo al modo indicizzato indiretto, ad eccezione del fatto che viene indicizzato l'indirizzo di pagina zero del PUNTATORE, anziche' l'indirizzo di base attuale. Percio', l'indirizzo di base attuale E' l'indirizzo attuale, in quanto l'indice e' gia' stato usato in modo indiretto. Il modo indiretto indicizzato puo' essere usato anche nel caso in cui nella memoria di pagina zero venga allocata una TABELLA di puntatori indiretti; in questo caso, il registro X specifica quale puntatore indiretto usare.

ESEMPIO:

Supponiamo che la locazione $02 contenga $45 e la locazione $03 contenga $10. Se si esegue l'istruzione per caricare l'accumulatore nel modo indiretto indicizzato, e se l'indirizzo di pagina zero specificato e' $02, allora l'indirizzo attuale sara':

```
Byte basso = Contenuto di ($02+X)
Byte  alto = Contenuto di ($03+X)
Registro X = $00
```

Percio' il puntatore attuale si trovera' in $02+X=$02, e l'indirizzo attuale sara' l'indirizzo indiretto contenuto in $02, cioe' di nuovo $1045.

L'intestazione di questo modo rende in effetti implicito il principio, anche se a prima vista risulta difficile da afferrare. In un'altra forma, tale principio puo' suonare come segue: "Sto andando a recapitare questa lettera al quinto ufficio postale di indirizzo $01, MEMORY ST., e l'indirizzo riportato sulla lettera sara' poi consegnato a $1600, MEMORY ST.". Traducendo in codice:

```
LDA #$00      - Carica l'indirizzo basso di base attuale
STA $06       - Imposta il byte basso dell'indirizzo indiretto
LDA #$16      - Carica l'indirizzo indiretto alto
STA $07       - Imposta il byte alto dell'indirizzo indiretto
LDX #$05      - Imposta l'indice indiretto (X)
LDA ($01,X)   - Carica indicizzato indirettamente da X
```

NOTA: Dei due modi indiretti di indirizzamento, quello di uso piu' comune e' il primo (indicizzato indiretto)

SALTI E CONTROLLO

Un altro principio molto importante nel linguaggio macchina e' dato dalla capacita' di testare e di scoprire certe condizioni, in modo simile alla struttura "IF...THEN, IF...GOTO" del CBM BASIC.

I numerosi indicatori posti nel registro di stato vengono interessati da istruzioni diverse in maniera diversa. Per esempio, c'e' un indicatore che viene impostato (ON) quando un'istruzione da' come risultato zero, e viene disattivato (OFF) quando il risultato e' diverso da zero.

L'istruzione:

```
LDA #$00
```

attiva l'indicatore di risultato zero, in quanto questa istruzione ha come risultato, nell'accumulatore, proprio lo zero.

Esistono alcune istruzioni che in particolari condizioni fanno passare il controllo ad un'altra parte del programma. Un esempio di istruzione di salto e' BEQ, che significa "salta se il risultato e' uguale a zero". Le istruzioni di salto vengono eseguite se la condizione risulta vera, altrimenti il programma continua dalla prossima istruzione, come se non fosse accaduto nulla. Le istruzioni di salto vengono eseguite non in base al risultato della precedente (o delle precedenti) istruzione, ma dall'esame interno del registro di stato. Come si e' gia' detto, nel registro di stato c'e' un indicatore per risultato zero. Quindi, l'istruzione BEQ salta se l'indicatore di risultato zero (conosciuto come Z) e' attivato. Ogni istruzione di salto ha la sua complementare. L'istruzione BEQ ha l'istruzione complementare BNE, che significa "salta se il risultato non e' uguale a zero" (se cioe' Z non e' attivato).

I registri indice hanno un numero di istruzioni associate che ne modificano il contenuto. Per esempio, l'istruzione INX incrementa il registro indice X. Se il registro indice X, prima di essere incrementato, conteneva $FF (il massimo numero che il registro X puo' contenere), "ritornera'" zero. Se si vuole che il programma continui a fare qualcosa fintantoche' si incrementa l'indice X fino a portarlo a zero, si puo' usare l'istruzione BNE, che continua a testare X finche'

non lo trova a zero.
Il contrario dell'istruzione INX e' DEX, che decrementa il registro indice X. Se il contenuto del registro X e' zero, DEX lo riporta a $FF. In maniera del tutto analoga si comportano le istruzioni INY e DEY, relative al registro Y.
Ma che cosa fare se un programma non desidera aspettare che X e Y abbiano raggiunto (oppure no) lo zero ?. Bene, ci sono le istruzioni di confronto, CPX e CPY, che consentono (al programmatore che lavora in Linguaggio Macchina) di testare il registro indice con valori specifici, o addirittura con il contenuto delle locazioni di memoria. Se si volesse sapere se il registro X contiene $40, si dovrebbe usare la seguente istruzione:

```
CPX #$40    - Confronta X con il "valore" $40
BEQ         - Se questa condizione e' vera, salta ad un'altra
              parte del programma
```

Il confronto e le istruzioni di salto coprono la parte piu' importante di ogni Linguaggio Macchina.
L'operando specificato in un'istruzione di salto, usando la 64MON, rappresenta l'indirizzo della parte di programma alla quale si salta quando si incontrano le dovute condizioni. In ogni caso, l'operando e' solamente un "offset", che conduce il programma dal punto in cui si trova attualmente all'indirizzo specificato. Tale "offset" e' di appena un byte, e percio' l'intervallo a cui un istruzione di salto puo' saltare e' limitato: infatti, sono consentiti salti di 128 byte all'indietro e di 127 in avanti.

NOTA: Questo e' un intervallo totale di 255 byte, che rappresenta, naturalmente, il massimo intervallo di valori che un byte puo' contenere.

La cartuccia 64MON segnala un "salto fuori dall'intervallo", rifiutandosi di "assemblare" quella particolare istruzione. Il salto e' un'istruzione "veloce" data dagli standard del Linguaggio Macchina, grazie al principio dell'"offset", che e' l'opposto di un indirizzo assoluto. La 64MON permette di scrivere un indirizzo assoluto, ed e' essa a calcolare l'esatto "offset". Questa e' una delle "comodita'" date dall'uso di un assembler.

NOTA: NON e' possibile trattare ogni singola istruzione di salto. Per ulteriori informazioni si rimanda alla sezione bibliografica (Appendice F)

SOTTOPROCEDURE

Nel Linguaggio Macchina (analogamente a quanto avviene nel BASIC) e' possibile chiamare una sottoprocedura. L'istruzione che chiama una sottoprocedura e' JSR (salta alla procedura), seguita da uno specifico indirizzo assoluto.
C'e' una sottoprocedura, incorporata nel Sistema Operativo, che scrive un carattere sullo schermo. Il codice CBM ASCII dei caratteri deve trovarsi nell'accumulatore, prima della chiamata alla sottoprocedura, il cui indirizzo e' $FFD2; percio', per scrivere "HI" sul video, si deve caricare il seguente programma:

```
.A 1400 LDA #$48   - Carica il codice CBM ASCII della lettera "H"
.A 1402 JSR $FFD2  - Stampa tale lettera
.A 1405 LDA #$49   - Carica il codice CBM ASCII della lettera "I"
.A 1407 JSR $FFD2  - Stampa tale lettera
.A 140A LDA #$0D   - Stampa un ritorno carrello
.A 140C JSR $FFD2
.A 140F BRK        - Ritorna a 64MON
.G 1400            - Stampa "HI" e ritorna a 64MON
```

La procedura "stampa di un carattere", che abbiamo appena usato, fa parte della tavola di salto del KERNAL. L'istruzione equivalente alla GOTO del BASIC e' JMP, che significa "salto all'indirizzo assoluto specificato". Il KERNAL e' una lunga lista di sottoprocedure "standardizzate", che controllano tutto l'input e l'output del COMMODORE 64. Ciascuna entrata nel KERNAL salta ad una sottoprocedura del Sistema Operativo. Questa "tavola di salto" si trova fra le locazioni di memoria $FF84 e $FFF5. Una completa spiegazione del KERNAL e' disponibile nella "sezione di riferimento al KERNAL" di questo Manuale. Tuttavia, certe procedure vengono usate qui di seguito per mostrare quanto sia efficace e di facile uso il KERNAL.

Usiamo ora i nuovi principi appresi in un altro programma, che ci consente di inserire le istruzioni nel contesto.

Questo programma visualizza l'alfabeto usando una routine del KERNAL. L'unica nuova istruzione introdotta in questo programma e' TXA, che trasferisce il contenuto del registro indice X nell'accumulatore.

```
.A 1400 LDX #$41   - X = CBM ASCII di "A"
.A 1402 TXA        - A = X
.A 1403 JSR $FFD2  - Scrive il carattere
.A 1406 INX        - Incrementa il contatore
.A 1407 CPX #$5B   - Si e' oltrepassata la "Z" ?
.A 1409 BNE $1402  - Se no, torna all'istruzione 1402 e ricomincia
.A 1408 BRK        - Se si, ritorna a 64MON
```

Per far stampare l'alfabeto al COMMODORE 64, usare il solito comando:

```
.G 1400
```

Il commento che si trova a lato del programma ne spiega la logica ed il funzionamento. E' sempre consigliabile scrivere un programma prima sulla carta, e quindi verificarlo, possibilmente, un segmento alla volta.

SUGGERIMENTI UTILI PER IL PRINCIPIANTE

Uno dei modi migliori per imparare il Linguaggio Macchina e' osservare programmi scritti in Linguaggio Macchina da altre persone, pubblicati di solito sulle riviste e sugli articoli della stampa specializzata. Non ha alcuna importanza se il computer per cui sono scritti e' diverso dal COMMODORE 64, basta che anch'esso usi il microprocessore 6510 (o 6502). Si dovrebbe essere in grado di comprendere completamente i codici che vi sono riportati. Fare questo richiede perseveranza, specialmente quando si incontra una tecnica nuova mai vista prima.

Dopo aver osservato attentamente gli altri programmi in Linguaggio Macchina, se ne dovrebbero scrivere alcuni, come ad esempio programmi

di utilita' per altri programmi in BASIC, oppure programmi scritti interamente in Linguaggio Macchina.
Inoltre, si dovrebbero usare le "utilities", disponibili o NEL computer o in un programma, che saranno di aiuto nella scrittura, nell'editazione o nell'individuazione degli errori di un programma in Linguaggio Macchina. Un esempio puo' essere fornito dal KERNAL, che permette di controllare la tastiera, di stampare testi, di controllare i dispositivi periferici come dischi, stampanti, modem, ecc., la gestione della memoria e dello schermo. E' estrememente potente, ed il suo uso e' fortemente raccomandato (cfr. Gestione del KERNAL).
Vantaggi derivanti dallo scrivere i programmi in Linguaggio Macchina:

1. Velocita' - Il Linguaggio Macchina e' centinaia, talvolta migliaia di volte, piu' veloce di un linguaggio ad alto livello come il BASIC.

2. Tenuta - Un Linguaggio Macchina puo' essere reso completamente impermeabile, cioe' si puo' mettere l'Utente nella condizione di eseguire solo quello che gli consente di fare il programma. In un linguaggio ad alto livello si puo' solo sperare che l'Utente non faccia "saltare" l'interprete BASIC, inserendo ad esempio uno zero che, successivamente, comporti un:

```
?DIVISION BY ZERO ERROR IN LINE 830

READY.
■
```

In essenza, la programmazione in Linguaggio Macchina non puo' che valorizzare maggiormente il computer.

UN COMPITO PIÙ IMPEGNATIVO

Affrontando un compito piu' impegnativo, si e' di solito presi da pensieri. Si cerca di intuire come vengono eseguiti certi processi in Linguaggio Macchina. Quando si e' all'inizio di un compito, e' buona norma riportarlo su carta. E' consigliabile inoltre fare uso di diagrammi a blocchi di uso della memoria, di moduli funzionali di codici e di flussi di programma. Supponiamo di voler scrivere il gioco della roulette in Linguaggio Macchina; ad esempio:

```
* Visualizzare il titolo
* Chiedere se il giocatore richiede istruzioni
* Se si, visualizzarle e saltare a START
* Se no, saltare direttamente a START
* START - Inizializzare tutto
* MAIN  - Visualizzare il tavolo della roulette
* Raccogliere le puntate
* Far girare la ruota
* Far rallentare la ruota fino a fermarla
* Confrontare le puntate con il risultato
* Informare il giocatore
* Ha perso ?
* Se si, saltare a MAIN
* Se no, informarlo della vincita e saltare a START
```

Questo e' lo schema principale. Quando ciascun modulo e' stato individuato, puo' essere scomposto ulteriormente. Se si deve affrontare un problema complesso che puo' essere scomposto in blocchi piu' piccoli per poter essere compreso, allora siamo in grado di affrontare qualcosa che sembra impossibile, e vederlo realizzato.
Questo processo migliora solo con la pratica, percio' TENETE DURO!...

INSIEME DELLE ISTRUZIONI DEL MICROPROCESSORE MCS6510

SEQUENZA ALFABETICA

ADC	Somma la Memoria all'Accumulatore, con Riporto
AND	"AND" fra Memoria ed Accumulatore
ASL	Scorrimento (Shift) a Sinista di un bit (Memoria o Accumulatore)
BCC	Salto sull'azzeramento del Riporto
BCS	Salto sull'impostazione del Riporto
BEQ	Salto su Risultato Zero
BIT	Confronta i bit nella Memoria con l'Accumulatore
BMI	Salto su Risultato Meno
BNE	Salto su Risultato Non-Zero
BPL	Salto su Risultato Piu'
BRK	Interruzione (break) forzata
BVC	Salto sull'Azzeramento dell'Overflow
BVS	Salto sull'Impostazione dell'Overflow
CLC	Azzera l'Indicatore di Riporto
CLD	Azzera il Modo Decimale
CLI	Azzera l'Interruzione e Disabilita il Bit
CLV	Azzera l'Indicatore di Overflow
CMP	Compara Memoria ed Accumulatore
CPX	Compara Memoria ed Indice X
CPY	Compara Memoria ed Indice Y
DEC	Decrementa la Memoria di uno
DEX	Decrementa l'Indice X di uno
DEY	Decrementa l'Indice Y di uno
EOR	OR esclusivo della Memoria con l'Accumulatore
INC	Incrementa la Memoria di uno
INX	Incrementa l'Indice X di uno
INY	Incrementa l'Indice Y di uno
JMP	Salto a Nuova Locazione
JSR	Salto a Nuova Locazione e salvataggio dell'indirizzo di ritorno
LDA	Carica l'Accumulatore con la Memoria
LDX	Carica l'Indice X con la Memoria
LDY	Carica l'Indice Y con la Memoria
LSR	Scorrimento a Destra (Shift) di un Bit (Memoria o Accumulatore)
NOP	Nessuna Operazione

ORA	OR della Memoria con l'Accumulatore
PHA	Posiziona l'Accumulatore sullo Stack
PHP	Posiziona lo Stato del Processore sullo Stack
PLA	Ritira l'accumulatore dallo Stack
PLP	Ritira lo Stato del Processore dallo Stack
ROL	Ruota a Sinitra di un Bit (Memoria o Accumulatore)
ROR	Ruota a Destra di un Bit (Memoria o Accumulatore)
RTI	Ritorno da Interruzione
RTS	Ritorno da Sottoprocedura
SBC	Sottrae la Memoria dall'Accumulatore, con Prestito
SEC	Imposta l'Indicatore di Riporto
SED	Imposta il Modo Decimale
SEI	Imposta lo Stato di Disabilitazione dell'interruzione
STA	Registra l'Accumulatore in Memoria
STX	Registra l'Indice X in Memoria
STY	Registra l'Indice Y in Memoria
TAX	Trasferisce l'Accumulatore all'Indice X
TAY	Trasferisce l'Accumulatore all'Indice Y
TSX	Trasferisce il Puntatore allo Stack all'Indice X
TXA	Trasferisce l'Indice X all'Accumulatore
TXS	Trasferisce l'Indice X al Registro dello Stack
TYA	Trasferisce l'Indice Y all'Accumulatore

Al riassunto che segue vanno applicate le seguenti notazioni:

A	Accumulatore
X,Y	Registri indice
M	Memoria
P	Registro di Stato del Processore
S	Puntatore allo Stack
	Cambio
	Nessun cambio
+	Addizione
	AND logico
-	Sottrazione
	OR esclusivo logico
	Trasferimento dallo Stack
	Trasferimento allo Stack
	Trasferimento a
	Trasferimento da
V	OR logico
PC	Contatore di Programma
PCH	Contatore di Programma Alto
PCL	Contatore di Programma Basso
OPER	Operando
#	Modo di indirizzamento immediato

NOTA: In cima a ciascuna tavola e' riportato fra parentesi un numero di riferimento (Ref: xx) che rimanda l'Utente alla relativa sezione nel Manuale di Programmazione, in cui l'istruzione e' definita e discussa

ADC — Somma la memoria all'accumulatore, con riporto — ADC

Operazione: A+M+C → A, C

(Ref.: 2.2.1)

N	Z	C	I	D	V
✓	✓	✓	≃	—	✓

Modo di indirizzamento	Forma in linguaggio Assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	ADC #Oper	69	2	2
Pagina Zero	ADC Oper	65	2	3
Pagina Zero, X	ADC Oper, X	75	2	4
Assoluto	ADC Oper	6D	3	4
Assoluto, X	ADC Oper, X	7D	3	4 (*)
Assoluto, Y	ADC Oper, Y	79	3	4 (*)
(Indiretto, X)	ADC (Oper, X)	61	2	6
(Indiretto), Y	ADC (Oper), Y	71	2	5 (*)

(*) Aggiunge 1 se si e' oltrepassato il limite della pagina di memoria

AND — "AND" logico tra memoria ed accumulatore — AND

Operazione: A∧M→A

(Ref.: 2.2.3.0)

N	Z	C	I	D	V
✓	✓	—	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	AND #Oper	29	2	2
Pagina Zero	AND Oper	25	2	3
Pagina Zero, X	AND Oper, X	35	2	4
Assoluto	AND Oper	2D	3	4
Assoluto, X	AND Oper, X	3D	3	4 (*)
Assoluto, Y	AND Oper, Y	39	3	4 (*)
Indiretto, X	AND (Oper, X)	21	2	6
Indiretto, Y	AND (Oper), Y	31	2	5

(*) Aggiunge 1 se si e' oltrepassao il limite della pagina di memoria

ASL — Scorrimento a sinistra di un bit — ASL

Operazione: C ← | 7 | 6 | 5 | 4 | 3 | 2 | 1 | ∅ | ← ∅

(Ref.: 10.2)

N	Z	C	I	D	V
✓	✓	✓	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ASL A	∅A	1	2
Pagina Zero	ASL Oper	∅6	2	5
Pagina Zero, X	ASL Oper, X	16	2	6
Assoluto	ASL Oper	∅E	3	6
Assoluto, X	ASL Oper, X	1E	3	7

BCC Salto sull'azzeramento del riporto BCC

Operazione: Salto su C=ø

N Z C I D V
– – – – – –

(Ref.: 4.1.1.3)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BCC Oper	9ø	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BCS Salto sull'impostazione del riporto BCS

Operazione: Salto su C=1

N Z C I D V
– – – – – –

(Ref.:4.1.1.4)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Relativo	BCS Oper	Bø	2	2 (*)

(*) Aggiuge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BEQ Salto su risultato zero BEQ

Operazione: Salto su Z=1

N Z C I D V
– – – – – –

(Ref.: 4.1.1.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BEQ Oper	Fø	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BIT — Confronta i bit nella memoria con l'accumulatore

BIT

Operazione: A ∧ M, M7 → N, M6 → V

N Z C I D V

I bit 6 e 7 sono trasferiti nel registro di stato. Se il risultato di A ∧ M e' 0, allora Z=1, altrimenti Z=0

M_7 √ — — — M_6

(Ref.: 4.2.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	BIT Oper	24	2	3
Assoluto	BIT Oper	2C	3	4

BMI — Salto su risultato meno

BMI

Operazione: Salto su N=1

N Z C I D V

— — — — — —

(Ref.: 4.1.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BMI Oper	30	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BNE — Salto su risultato non-zero

BNE

Operazione: Salto su Z=0

N Z C I D V

— — — — — —

(Ref.: 4.1.1.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BNE Oper	D0	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BPL Salto su risultato piu' BPL

Operazione: Salto su N=ø

N Z C I D V
— — — — — —

(Ref.: 4.1.1.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BPL Oper	1ø	2	2 (*)

(*) Aggiunge 1 se si e' oltrepassato il limite della pagina di memoria

BRK Interruzione forzata BRK

Operazione: Interruzione forzata PC+2↓P↓

N Z C I D V
— — — 1 — —

(Ref.: 9.11)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	BRK	øø	1	7

1. Un comando BRK non puo' essere forzato da un'impostazione di I.

BVC Salto sull'azzeramento dell'overflow

Operazione: Salto su V=0

N Z C I D V
— — — — — —

(Ref.: 4.1.1.8)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BVC Oper	5ø	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BVS Salto sull'impostazione dell'overflow BVS

Operazione: Salto su V=1

N Z C I D V
— — — — — —

(Ref.:4.1.1.7)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Relativo	BVS Oper	7ø	2	2 (*)

(*) Aggiuge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

CLC — Azzera l'indicatore di riporto — CLC

Operazione: $\phi \rightarrow C$

N Z C I D V
— — ϕ — — —

(Ref.: 3.0.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLC	18	1	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

CLD — Azzera il modo decimale — CLD

Operazione: $\phi \rightarrow D$

N Z C I D V
— — — — ϕ —

(Ref.: 3.3.2.)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLD	D8	1	2

CLI — Azzera l'interruttore e disabilita il bit — CLI

Operazione: $\phi \rightarrow I$

N Z C I D V
— — — ϕ — —

(Ref.: 3.2.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLI	58	1	2 (*)

CLV — Azzera l'indicatore di overflow — CLV

Operazione: $\phi \rightarrow V$

N Z C I D V
— — — — — ϕ

(Ref.: 3.6.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLV	B8	1	2

CMP Compara memoria ed accumulatore

Operazione: A-M

N Z C I D V

(Ref.: 4.2.1)

✓ ✓ ✓ — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CMP #Oper	C9	2	2
Pagina Zero	CMP Oper	C5	2	3
Pagina Zero, X	CMP Oper,X	D5	2	4
Assoluto	CMP Oper	CD	3	4
Assoluto, X	CMP Oper, X	DD	3	4 (*)
Assoluto, Y	CMP Oper, Y	D9	3	4 (*)
(Indiretto, X)	CMP (Oper, X)	C1	2	6
(Indiretto), Y	CMP (Oper), Y	D1	2	5

(*) Aggiunge 1 se viene superato il limite delle pagine di memoria

CPX Compara memoria ed indice X

Operazione: X-M

N Z C I D V

(Ref.: 7.8)

✓ ✓ ✓ — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CPX #Oper	E0	2	2
Pagina Zero	CPX Oper	E4	2	3
Assoluto	CPX Oper	EC	3	4

CPY Compara memoria ed indice Y

Operazione: Y-M

N Z C I D V

(Ref.: 7.9)

✓ ✓ ✓ — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CPY #Oper	C0	2	2
Pagina Zero	CPY Oper	C4	2	3
Assoluto	CPY Oper	CC	3	4

DEC — Decrementa la memoria di uno — DEC

Operazione: M-1→M

(Ref.: 10.7)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	DEC Oper	C6	2	5
Pagina Zero, X	DEC Oper, X	D6	2	6
Assoluto	DEC Oper	CE	3	6
Assoluto, X	DEC Oper, X	DE	3	7

DEX — Decrementa l'indice X di uno — DEX

Operazione: X-1→X

(Ref.: 7.6)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	DEX	CA	1	2

DEY — Decrementa l'indice Y di uno — DEY

Operazione: Y-1→Y

(Ref.: 7.7)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	DEY	88	1	2

EOR — OR esclusivo fra memoria ed accumulatore — EOR

Operazione: A⊻M→A

(Ref.: 2.2.3.2)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	EOR #Oper	49	2	2
Pagina Zero	EOR Oper	45	2	3
Pagina Zero, X	EOR Oper, X	55	2	4
Assoluto	EOR Oper	4D	3	4
Assoluto, X	EOR Oper, X	5D	3	4 (*)
Assoluto, Y	EOR Oper, Y	59	3	4 (*)
(Indiretto, X)	EOR (Oper, X)	41	2	6
((Indiretto), Y	EOR (oper), Y	51	2	5

(*) Aggiungere 1 se si supera il limite della pagina di memoria.

INC — Incrementa la memoria di uno

Operazione: M+1 → M

(Ref.: 10.6)

N	Z	C	I	D	V
✓	✓	—	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	INC Oper	E6	2	5
Pagina Zero, X	INC Oper, X	F6	2	6
Assoluto	INC Oper	EE	3	6
Assoluto, X	INC Oper, X	FE	3	7

INX — Incrementa l'indice X di uno

Operazione: X+1 → X

(Ref.: 7.4)

N	Z	C	I	D	V
✓	✓	—	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	INX	E8	1	2

INY — Incrementa l'indice Y di uno

Operazione: Y+1 → Y

(Ref.: 7.5)

N	Z	C	I	D	V
✓	✓	—	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	INY	C8	1	2

JMP — Salto a nuova locazione

Operazione: (PC+1) → PCL (Ref.: 4.0.2)
(PC+2) → PCH (Ref.: 9.8.1)

N	Z	C	I	D	V
—	—	—	—	—	—

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Assoluto	JMP Oper	4C	3	3
Indiretto	JMP (Oper)	6C	3	5

JSR — Salto a nuova locazione e salvataggio dell'indirizzo di ritorno

JSR

Operazione: PC+2↓, (PC+1) → PCL
(PC+2) → PCH

N Z C I D V
– – – – – –

(Ref.: 8.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Assoluto	JSR Oper	20	3	6

LDA — Carica l'accumulatore con il contenuto della memoria

LDA

Operazione: M → A

N Z C I D V
✓ ✓ – – – –

(Ref.: 2.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDA #Oper	A9	2	2
Pagina Zero	LDA Oper	A5	2	3
Pagina Zero, X	LDA Oper, X	B5	2	4
Assoluto	LDA Oper	AD	3	4
Assoluto, X	LDA Oper, X	BD	3	4 (*)
Assoluto, Y	LDA Oper, Y	B9	3	4 (*)
(Indiretto, X)	LDA (Oper, X)	A1	2	6
(Indiretto), Y	LDA (Oper), Y	B1	2	5 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LDX — Carica l'indice X con la memoria

LDX

Operazione: M → X

N Z C I D V
✓ ✓ – – – –

(Ref.: 7.0)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDX #Oper	A2	2	2
Pagina Zero	LDX Oper	A6	2	3
Pagina Zero, Y	LDX Oper, Y	B6	2	4
Assoluto	LDX Oper	AE	3	4
Assoluto, Y	LDX Oper, Y	BE	3	4 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LDY — Carica l'indice Y con la memoria — LDY

Operazione: M → Y

N Z C I D V
✓ ✓ – – – –

(Ref.: 7.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDY #Oper	A0	2	2
Pagina Zero	LDY Oper	A4	2	3
Pagina Zero, X	LDY Oper, X	B4	2	4
Assoluto	LDY Oper	AC	3	4
Assoluto, X	LDY Oper, X	BC	3	4 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LSR — Scorrimento a destra di un bit (memoria o accumulatore — LSR

Operazione: ϕ → [7][6][5][4][3][2][1][0] → C

N Z C I D V
ϕ ✓ ✓ – – –

(Ref.: 10.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	LSR A	4A	1	2
Pagina Zero	LSR Oper	46	2	5
Pagina Zero, X	LSR Oper, X	56	2	6
Assoluto	LSR Oper	4E	3	6
Assoluto, X	LSR Oper, X	5E	3	7

NOP — Nessuna operazione — NOP

Operazione: Nessuna Operazione (2 cicli)

N Z C I D V

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	NOP	EA	1	2

ORA — OR della memoria con l'accumulatore

ORA

Operazione: A V M → A

N Z C I D V
✓ ✓ — — — —

(Ref.: 2.2.3.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	ORA #Oper	ø9	2	2
Pagina Zero	ORA Oper	ø5	2	3
Pagina Zero, X	ORA Oper, X	15	2	4
Assoluto	ORA Oper	øD	3	4
Assoluto, X	ORA Oper, X	1D	3	4 (*)
Assoluto, Y	ORA Oper, Y	19	3	4 (*)
(Indiretto, X)	ORA (Oper, X)	ø1	2	6
(Indiretto), Y	ORA (Oper), Y	11	2	5

(*) Aggiunge 1 se si supera la pagina di memoria

PHA — Posiziona l'accumulatore sullo stack

PHA

Operazione: A ↓

N Z C I D V
— — — — — —

(Ref.: 8.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PHA	48	1	3

PHP — Posiziona lo stato del processore sullo stack

PHP

Operazione: P ↓

N Z C I D V
— — — — — —

(Ref.: 8.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PHP	ø8	1	3

PLA — Ritira l'accumulatore dallo stack

PLA

Operazione: A ↑

N Z C I D V
✓ ✓ — — — —

(Ref.: 8.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PLA	68	1	4

PLP — Ritira lo stato del processore dallo stack — PLP

Operazione: P ↑

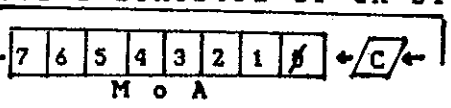
N Z C I D V
dallo stack

(Ref.: 8.12)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PLP	8	1	4

ROL — Ruota a sinistra di un bit (memoria o accumulatore) — ROL

Operazione:



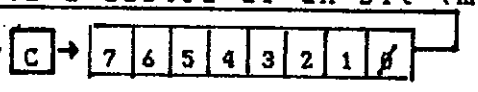
N Z C I D V
✓ ✓ ✓ – – –

(Ref.: 10.3)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ROL A	2A	1	2
Pagina Zero	ROL Oper	26	2	5
Pagina Zero, X	ROL Oper, X	36	2	6
Assoluto	ROL Oper	2E	3	6
Assoluto, X	ROL Oper, X	3E	3	7

ROR — Ruota a destra di un bit (memoria o accumulatore) — ROR

Operazione:



N Z C I D V
✓ ✓ ✓ – – –

(Ref.: 10.4)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ROR A	6A	1	2
Pagina Zero	ROR Oper	66	2	5
Pagina Zero, X	ROR Oper, X	76	2	6
Assoluto	ROR Oper	6E	3	6
Assoluto, X	ROR Oper, X	7E	3	7

RTI — Ritorno da un'interruzione

RTI

Operazione: P↑ PC↑

N Z C I D V
dallo stack

(Ref.: 9.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	RTI	40	1	6

RTS — Ritorno da una subroutine

RTS

Operazione: PC↑, PC+1→PC

N Z C I D V
— — — — — —

(Ref.: 8.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	RTS	60	1	6

SBC — Sottrae la memoria dall'accumulatore, con prestito

SBC

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

$\overline{C}$ = BORROW

N Z C I D V
✓ ✓ ✓ — — ✓

(Ref.: 2.2.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	SBC #Oper	E9	2	2
Pagina Zero	SBC Oper	E5	2	3
Pagina Zero, X	SBC Oper, X	F5	2	4
Assoluto	SBC Oper	ED	3	4
Assoluto, X	SBC Oper, X	FD	3	4 (*)
Assoluto, Y	SBC Oper, Y	F9	3	4 (*)
(Indiretto, X)	SBC (Oper, X)	E1	2	6
(Indiretto), Y	SBC (Oper), Y	F1	2	5

(*) Aggiunge 1 se si supera il limite della pagina di memoria

SEC — Imposta l'indicatore di riporto

SEC

Operazione: 1→C

N Z C I D V
— — 1 — — —

(Ref.: 3.0.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	SEC	38	1	2

SED — Imposta il modo decimale

Operazione: 1 → D

N Z C I D V
— — — — 1 —

(Ref.: 3.3.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	SED	F8	1	2

SEI — Imposta lo stato di disabilitazione dell'interruzione

Operazione: 1 → I

N Z C I D V
— — — 1 — —

(Ref.: 3.2.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	SEI	78	1	2

STA — Registra l'accumulatore in memoria

Operazione: A → M

N Z C I D V
— — — — — —

(Ref.: 2.1.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STA Oper	85	2	3
Pagina Zero, X	STA Oper, X	95	2	4
Assoluto	STA Oper	8D	3	4
Assoluto, X	STA Oper, X	9D	3	5
Assoluto, Y	STA Oper	99	3	5
(Indiretto, X)	STA (Oper, X)	81	2	6
(Indiretto), Y	STA (Oper), Y	91	2	6

STX — Registra l'indice X in memoria

Operazione: X → M

N Z C I D V
— — — — — —

(Ref.: 7.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STX Oper	86	2	3
Pagina Zero, Y	STX Oper, X	96	2	4
Assoluto	STX Oper	8E	3	4

STY Registra l'indice Y in memoria STY

Operazione: Y → M

N Z C I D V
— — — — — —

(Ref.: 7.3)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STY Oper	84	2	3
Pagina Zero, X	STY Oper, X	94	2	4
Assoluto	STY Oper	8C	3	4

TAX Trasferisce l'accumulatore all'indice X TAX

Operazione: A → X

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.11)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TAX	AA	1	2

TAY Trasferisce l'accumulatore all'indice Y TAY

Operaione: A → Y

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.13)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TAY	A8	1	2

TSX Trasferisce il puntatore allo stack nell'indice X TSX

Operazione: S → X

N Z C I D V
✓ ✓ — — — —

(Ref.: 8.9)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TSX	BA	1	2

TXA Trasferisce l'indice X all'accumulatore TXA

Operazione: X → A

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.12)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TXA	8A	1	2

TXS Trasferisce l'indice X al registro dello stack TXS

Operazione: X → S

N Z C I D V
— — — — — —

(Ref.:8.8)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TXS	9A	1	2

TYA Trasferisce l'indice Y all'accumulatore TYA

Operazione: Y → A

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.14)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TYA	98	1	2

MODI DI INDIRIZZAMENTO DELLE ISTRUZIONI E RELATIVI TEMPI DI ESECUZIONE (ESPRESSI IN CICLI DI CLOCK)

ACC = Accumulatore
IMM = Immediato
PZ = Pagina Zero
PZX = Pagina Zero, X
PZY = Pagina Zero, Y
ASS = Assoluto
ASX = Assoluto, X
ASY = Assoluto, Y
IMP = Implicito
REL = Relativo
INX = Indiretto, X
INY = Indiretto, Y
IA = Indiretto Assoluto

	ACC	IMM	PZ	PZX	PZY	ASS	ASX	ASY	IMP	REL	INX	INY	IA
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	6	7	.	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

	ACC	IMM	PZ	PZX	PZY	ASS	ASX	ASY	IMP	REL	INX	INY	IA
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Aggiungere un ciclo se l'indicizzazione supera il limite della pagina di memoria

** Aggiungere un ciclo se si effettua un salto, piu' un ciclo se l'operazione di salto supera il limite della pagina di memoria

ØØ - BRK
Ø1 - ORA - (Indirect,X)
Ø2 - Future Expansion
Ø3 - Future Expansion
Ø4 - Future Expansion
Ø5 - ORA - Zero Page
Ø6 - ASL - Zero Page
Ø7 - Future Expansion
Ø8 - PHP
Ø9 - ORA - Immediate
ØA - ASL - Accumulator
ØB - Future Expansion
ØC - Future Expansion
ØD - ORA - Absolute
ØE - ASL - Absolute
ØF - Future Expansion
1Ø - BPL
11 - ORA - (Indirect),Y
12 - Future Expansion
13 - Future Expansion
14 - Future Expansion
15 - ORA - Zero Page,X
16 - ASL - Zero Page,X
17 - Future Expansion
18 - CLC
19 - ORA - Absolute,Y
1A - Future Expansion
1B - Future Expansion
1C - Future Expansion
1D - ORA - Absolute,X
1E - ASL - Absolute,X
1F - Future Expansion
2Ø - JSR
21 - AND - (Indirect,X)
22 - Future Expansion
23 - Future Expansion
24 - BIT - Zero Page
25 - AND - Zero Page
26 - ROL - Zero Page
27 - Future Expansion
28 - PLP
29 - AND - Immediate
2A - ROL - Accumulator
2B - Future Expansion
2C - BIT - Absolute
2D - AND - Absolute
2E - ROL - Absolute
2F - Future Expansion
3Ø - BMI
31 - AND - (Indirect),Y
32 - Future Expansion
33 - Future Expansion
34 - Future Expansion
35 - AND - Zero Page,X
36 - ROL - Zero Page,X
37 - Future Expansion
38 - SEC
39 - AND - Absolute,Y
3A - Future Expansion
3B - Future Expansion
3C - Future Expansion
3D - AND - Absolute,X
3E - ROL - Absolute,X
3F - Future Expansion
4Ø - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
5Ø - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion
6Ø - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
7Ø - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion
8Ø - Future Expansion
81 - STA - (Indirect,X)
82 - Future Expansion
83 - Future Expansion
84 - STY - Zero Page
85 - STA - Zero Page
86 - STX - Zero Page

87 - Future Expansion
88 - DEY
89 - Future Expansion
8A - TXA
8B - Future Expansion
8C - STY - Absolute
8D - STA - Absolute
8E - STX - Absolute
8F - Future Expansion
9Ø - BCC
91 - STA - (Indirect),Y
92 - Future Expansion
93 - Future Expansion
94 - STY - Zero Page,X
95 - STA - Zero Page,X
96 - STX - Zero Page,Y
97 - Future Expansion
98 - TYA
99 - STA - Absolute,Y
9A - TXS
9B - Future Expansion
9C - Future Expansion
9D - STA - Absolute,X
9E - Future Expansion
9F - Future Expansion
AØ - LDY - Immediate
A1 - LDA - (Indirect,X)
A2 - LDX - Immediate
A3 - Future Expansion
A4 - LDY - Zero Page
A5 - LDA - Zero Page
A6 - LDX - Zero Page
A7 - Future Expansion
A8 - TAY
A9 - LDA - Immediate
AA - TAX
AB - Future Expansion
AC - LDY - Absolute
AD - LDA - Absolute
AE - LDX - Absolute
AF - Future Expansion
BØ - BCS
B1 - LDA - (Indirect),Y
B2 - Future Expansion
B3 - Future Expansion
B4 - LDY - Zero Page,X
B5 - LDA - Zero Page,X
B6 - LDX - Zero Page,Y
B7 - Future Expansion
B8 - CLV
B9 - LDA - Absolute,Y
BA - TSX
BB - Future Expansion
BC - LDY - Absolute,X
BD - LDA - Absolute,X
BE - LDX - Absolute,Y
BF - Future Expansion
CØ - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
DØ - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion
EØ - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
FØ - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

GESTIONE DELLA MEMORIA SUL COMMODORE 64

Il COMMODORE 64 ha 64K byte di RAM, oltre a 20K di ROM contenenti il BASIC, il SIstema Operativo ed il set di caratteri standard. Una porzione di memoria di 4K e' inoltre destinata ai dispositivi di accesso di I/O. Come puo' essere possibile tutto cio' su un computer dotato di un bus indirizzi a 16 bit, capace solamente di indirizzare, in codizioni normali, 64K ?

Il segreto sta nel microprocessore 6510: su questo circuito si trova una porta di input/output che serve a controllare la RAM, la ROM o l'I/O che compaiono in determinate parti della memoria del sistema. Questa porta e' usata anche per controllare il DATASSETTE(TM), percio' e' importante considerare solamente i bit appropriati.

La porta di I/O del 6510 compare nella locazione 1. Il registro direzione dati per questa porta compare nella locazione 0. La porta e' controllata come ogni altra porta di input/output presente nel sistema...il registro direzione dati controlla se un certo bit e' di ingresso o di uscita, e se il trasferimento di dati avviene attraverso questa porta.

Nella porta di controllo del 6510, le righe sono definite nel modo seguente:

NOME	BIT	DIREZIONE	DESCRIZIONE
LORAM	0	OUTPUT	Controllo per RAM/ROM da $A000 a $BFFF (BASIC)
HIRAM	1	OUTPUT	Controllo per RAM/ROM da $E000 a $FFFF (KERNAL)
CHAREN	2	OUTPUT	Controllo per I/O/ROM da $D000 a $DFFF
	3	OUTPUT	Linea di scrittura Cassette
	4	INPUT	Senso interruttori Cassette
	5	OUTPUT	Controllo motore Cassette

Il valore appropriato del registro direzione dati e' il seguente:

BITS	5	4	3	2	1	0
	1	0	1	1	1	1

dove 1 indica output e 0 input.

La somma di tali cifre binarie e' 47 decimale. Il COMMODORE 64 imposta automaticamente il registro direzione dati a questo valore.

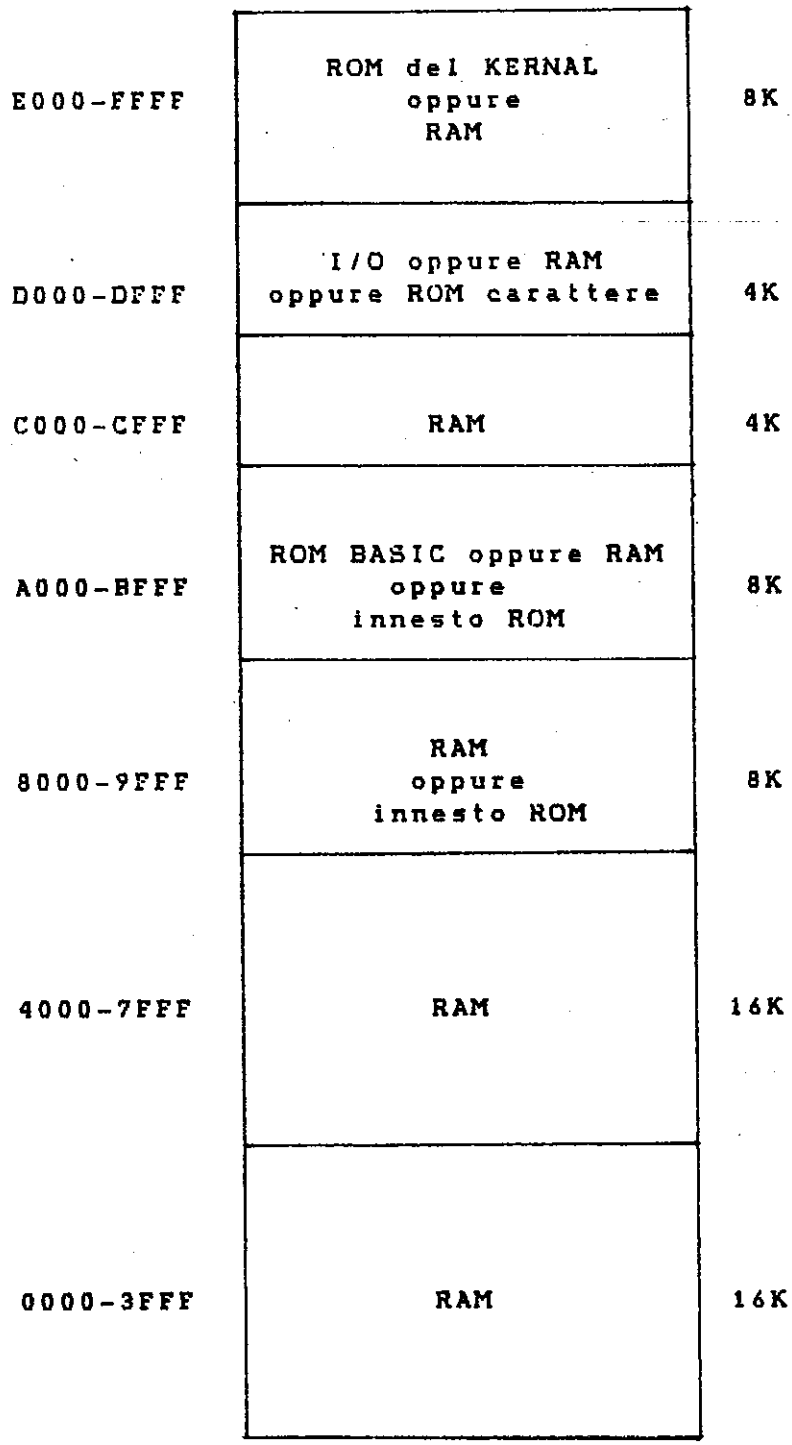
In generale, le linee di controllo eseguono la funzione riportata nella loro descrizione. Occasionalmente, si usa una combinazione di linee di controllo per ottenere una particolare configurazione della memoria.

LORAM (bit 0) puo' essere generalmente pensata come una linea di controllo che inserisce o esclude dall'area indirizzabile del microprocessore la ROM di 8K del BASIC. Per le operazioni BASIC questa linea e' normalmente ALTA. Se questa linea e' predisposta BASSA, la ROM del BASIC scompare dalla mappa di memoria ed e' rimpiazzata dagli 8K byte di RAM che vanno dalla locazione $A000 alla locazione $BFFF.

HIRAM (bit 1) puo' essere generalmente pensato come una linea di controllo che inserisce o esclude dall'area indirizzabile del microprocessore la ROM di 8K bytes del KERNAL. Per le operazioni in BASIC questa linea e' normalmente ALTA. Se questa linea viene predisposta BASSA, la ROM del KERNAL scompare dalla mappa di memoria ed e' rimpiazzata dagli 8K byte di RAM che vanno dalla locazione $E000 alla locazione $FFFF.

CHAREN (bit 2) e' usata solamente per inserire o escludere dall'area indirizzabile del microprocessore la ROM di 4K byte del generatore di caratteri. Dal punto di vista del processore, la ROM dei caratteri occupa la stessa area indirizzabile dei dispositivi di I/O ($D000-$DFFF). Quando la linea CHAREN e' impostata a 1 (condizione normale), i dispositivi di I/O appaiono nell'area indirizzabile del microprocessore e la ROM dei caratteri non e' piu' accessibile. Quando il bit CHAREN e' azzerato, la ROM dei caratteri compare nell'area indirizzabile del microprocessore e i dispositivi di I/O non sono accessibili (il microprocessore ha bisogno di accedere alla ROM dei caratteri solo quando scarica nella RAM l'insieme dei caratteri della ROM. Per questo e' consigliabile vedere nel Capitolo riservato alla GRAFICA la Sezione dei CARATTERI PROGRAMMABILI). In certe configurazioni di memoria, CHAREN puo' essere sovrapposto da un'altra linea di controllo. Senza i dispositivi di I/O, CHAREN non ha alcun effetto sulla configurazione della memoria. La RAM appare invece nelle locazioni da $D000 a $DFFF.

NOTA: In ciascuna mappa di memoria contenente la ROM, un'istruzione WRITE (o POKE) rivolta ad una locazione ROM provoca la memorizzazione dei dati RAM "sotto" la ROM. La scrittura in una locazione ROM comporta la memorizzazione dei dati nella RAM "riservata". Questo permette, ad esempio, di mantenere uno schermo ad alta risoluzione sotto il controllo di una ROM, e di poterlo modificare riportandolo nello spazio di indirizzamento del processore. Normalmente, una READ ad una locazione ROM riporta il contenuto della ROM, ma non della RAM "riservata".



DESCRIZIONE DELLE LOCAZIONI DI I/O

D000-D3FF	VIC (Controllore Video)	1K Byte
D400-D7FF	SID (Sintetizzatore del Suono)	1K Byte
D800-DBFF	RAM colore	500 Byte
DC00-DCFF	CIA 1 (Tastiera)	256 Byte
DD00-DDFF	CIA 2 (Bus seriale, Porta Utente/RS-232)	256 Byte
DE00-DEFF	Presa aperta di I/O #1 (Abilitatore CP/M)	256 Byte
DF00-DFFF	Presa aperta di I/O #2 (Disk)	256 Byte

Le due prese aperte di I/O sono state previste per scopi generali dell'Utente e per cartucce speciali di I/O (come IEEE); sperimentalmente, sono state predisposte per supportare la cartuccia Z-80 (opzione CP/M), e per fare da interfaccia ad un sistema di dischi ad alta velocita' e di basso costo.

Il sistema provvede allo "start automatico" dei programmi contenuti nella Cartuccia Espansione del COMMODORE 64. Il programma della cartuccia ha inizio se i primi 9 byte della cartuccia ROM a partire dalla locazione 32768 ($8000 HEX) contengono dati specifici: i primi due byte devono contenere il vettore dello Start a Freddo, usato dal programma della cartuccia; i due byte successivi, che partono da 32770 ($8002 HEX), devono contenere il vettore dello Start a Caldo, usato dal programma della cartuccia; i tre byte successivi contengono le lettere CBM, con il bit 7 impostato in ciascuna lettera; infine, gli ultimi due byte contengono la cifra 80 codificata in PET ASCII.

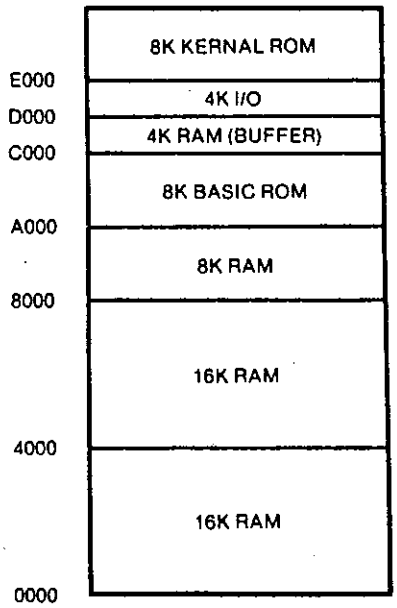
MAPPE DELLA MEMORIA DEL COMMODORE 64

Le seguenti tabelle riportano la lista delle varie configurazioni di memoria disponibili sul COMMODORE 64, gli stati delle linee di controllo che selezionano ciascuna mappa di memoria e l'applicazione particolare di ciascuna di esse.

1) Mappa di default della memoria BASIC; fornisce il BASIC 2.0 e 38K contigui di RAM Utente.

LORAM = 1 HIRAM = 1 GAME = 1 EXROM = 1

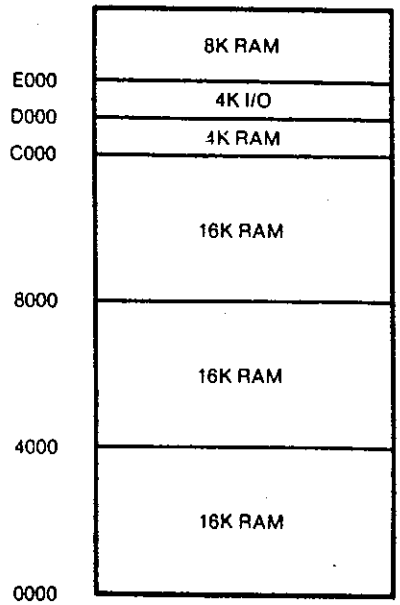
(X=non usato, 0=basso, 1=alto)



2) Mappa comprendente 60K di RAM e di dispositivi di I/O. Le procedure di I/O di accesso ai dischi sono a carico dell'Utente (devono essere scritte da quest'ultimo)

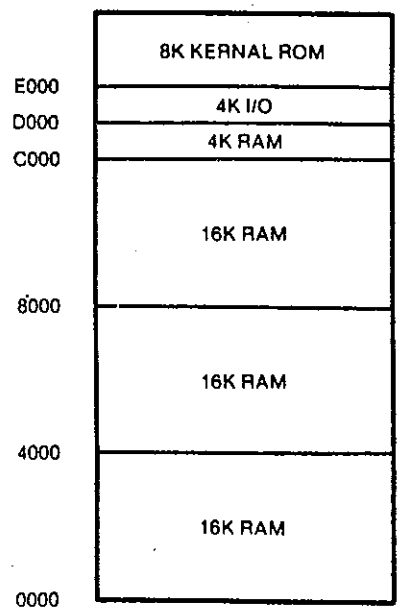
LORAM = 1 HIRAM = 0 GAME = 1 EXROM = X,
LORAM = 1 HIRAM = 0 GAME = 0 EXROM = 0
(La ROM carattere non e' accessibile dalla CPU in questa mappa)

(X=non usato, 0=basso, 1=alto)



3) Mappa costruita per l'uso dei programmi "softload" (compreso il CP/M); comprende 52K byte contigui di RAM Utente, i dispositivi di I/O e le routine di I/O per l'accesso ai dischi

LORAM = 0 HIRAM = 1 GAME = 1 EXROM = X
(X=non usato, 0=basso, 1=alto)

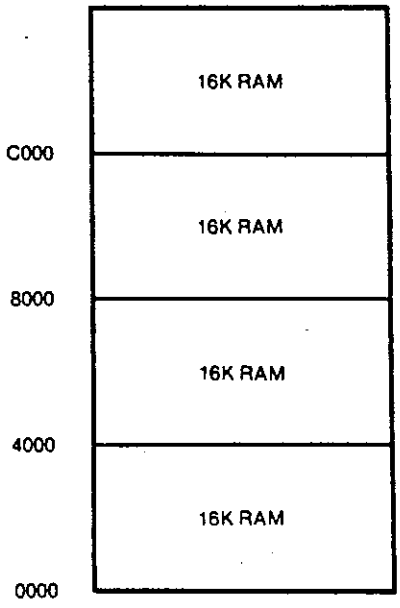


4) Mappa costruita per l'accesso a tutti i 64K byte RAM. I dispositivi di I/O devono essere riportati nell' area indirizzabile del processore per ogni operazione di I/O

LORAM = 0 HIRAM = 0 GAME = 1 EXROM = X

LORAM = 0 HIRAM = 0 GAME = X EXROM = 0

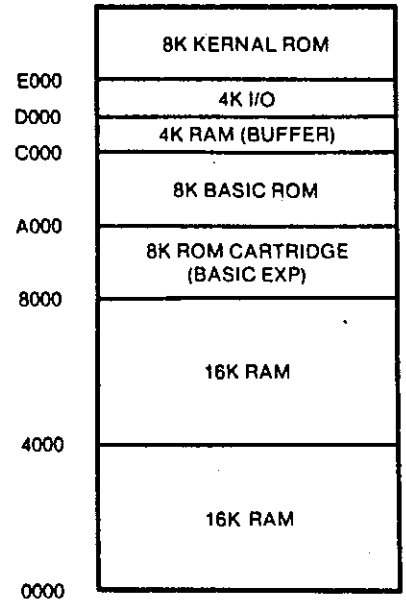
(X=non usato, 0=basso, 1=alto)



5) Mappa rappresentante la configurazione standard di un sistema BASIC con una ROM espansa BASIC. Fornisce 32K RAM contigui Utente e fino a 8K bytes di "arricchimento" BASIC

LORAM = 1 HIRAM = 1 GAME = 0 EXROM = 0

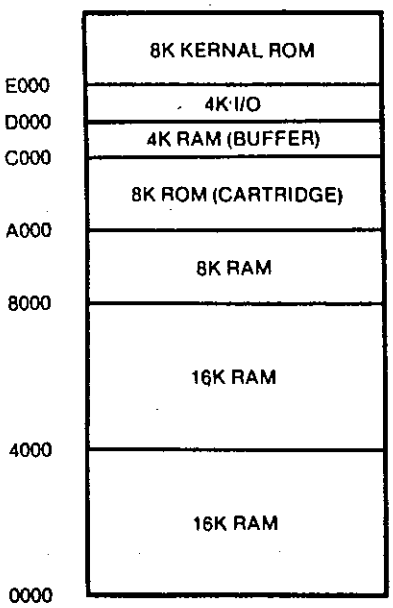
(X=non usato, 0=basso, 1=alto)



6) Mappa rappresentante una configurazione di 40K byte contigui di RAM Utente; mette a disposizione fino a 8K byte di innesto ROM per applicazioni speciali basate sulla ROM, che non richiedono il BASIC

LORAM = 0 HIRAM = 1 GAME = 0 EXROM = 0

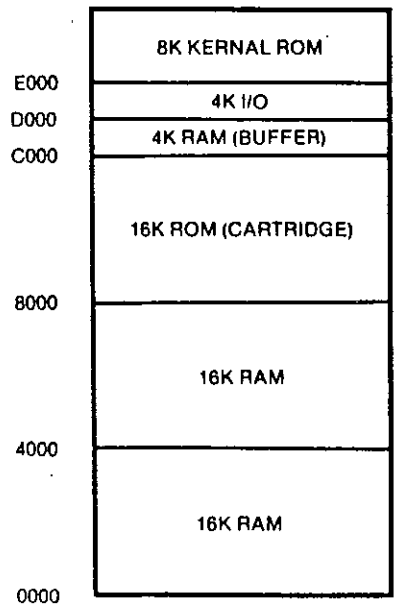
(X=non usato, 0=basso, 1=alto)



7) Questa mappa mette a dispsizione 32K byte contigui di RAM Utente, lasciando fino a 16K bytes di innesto ROM per applicazioni speciali basate sulla ROM non richiedenti il BASIC (word processor, altri linguaggi, ecc.)

LORAM = 1 HIRAM = 1 GAME = 0 EXROM = 0

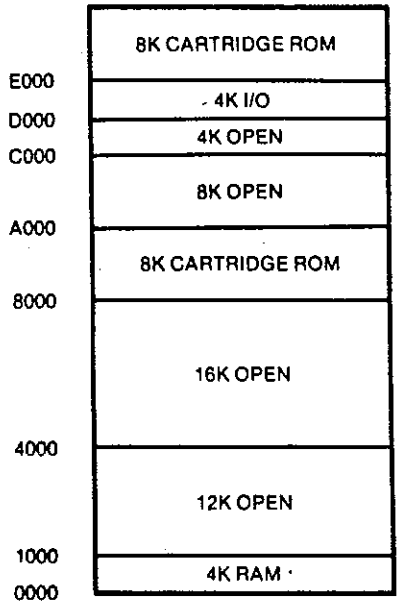
(X=non usato, 0=basso, 1=alto)



8) Mappa ULTIMAX per videogiochi. Si noti che, se richiesto, si puo' accedere dall'esterno del COMMODORE 64 a 2K byte per ULTIMAX, senza considerare alcuna RAM della cartuccia

LORAM = X HIRAM = X GAME = 0 EXROM = 1

(X=non usato, 0=basso, 1=alto)



IL KERNAL

Uno dei problemi che deve essere affrontato dai programmatori nel campo dei microcomputer e' la questione di che cosa fare quando la casa costruttrice modifica il Sistema Operativo del computer. I programmi in linguaggio macchina, che richiedono molto tempo per essere sviluppati, potrebbero non funzionare piu', costringendo ad apportare modifiche complesse al programma. Per rendere meno oneroso questo problema, la Commodore ha sviluppato un metodo chiamato KERNAL, che interviene in difesa degli scrittori di software.

Praticamente, il KERNAL e' una TABELLA DEI SALTI standardizzata rivolta all'input, all'output ed alle routine di gestione della memoria del Sistema Operativo. Le locazioni di ciascuna routine che si trova in ROM puo' essere cambiata quando viene rinnovato il sistema, ma la tavola dei salti KERNAL sara' modificata continuamente per adattarsi. Modificare le sottoprocedure scritte in Linguaggio Macchina e' molto piu' veloce se esse usano solamente la routine di sistema in ROM, tramite il KERNAL.

Il KERNAL costituisce il Sistema Operativo del COMMODORE 64. Tutto l'input, l'output e la gestione della memoria e' controllato dal KERNAL. Per semplificare i programmi in Linguaggio Macchina, e per essere sicuri che la futura versione del Sistema Operativo del COMMODORE 64 non renda inutilizzabili i programi scritti in Linguaggio Macchina, il KERNAL comprende una tabella dei salti. Traendo vantaggio dalle 39 routine di input/output e da altre "utilities" disponibili sulla tabella, non solo si risparmia tempo, ma e' anche piu' facile trasferire i programmi dal COMMODORE 64 ad un altro computer.

La tabella dei salti ha la sua locazione nell'ultima pagina di memoria, nella Memoria a Sola Lettura (ROM).

Per usare la tabella dei salti del KERNAL e' necessario, per prima cosa, impostare i parametri necessari al funzionamento della procedura KERNAL. Quindi occorre saltare alla sottoprocedura, tramite l'istruzione JSR, nel punto esatto della tabella dei salti del KERNAL. Dopo che il KERNAL ha esaurito le sue funzioni, riporta il controllo al programma in Linguaggio Macchina. A seconda di quale routine del KERNAL si sta usando, certi registri possono ritornare parametri al programma. I registri propri di ciascuna routine del KERNAL possono essere individuati nelle singole descrizioni delle sottoprocedure del KERNAL.

A questo punto, una buona domanda potrebbe essere la seguente: "Perche' usare la tavola dei salti ? Perche' JSR non va ad interessare direttamente la sottoprocedura del KERNAL ?". Si usa la tabella dei salti perche',se il KERNAL o il BASIC vengono modificati, i programmi in linguaggio macchina possono essere ancora validi. In un futuro Sistema Operativo, le routine potranno trovarsi allocate in un'altra parte della mappa della memoria...ma la tavola dei salti funzionera' ancora bene !

ATTIVITÀ DI INIZIALIZZAZIONE DEL KERNAL

1) All'inizializzazione, il KERNAL riattiva innanzitutto il puntatore allo stack, poi azzera il modo decimale.
2) Il KERNAL passa poi a controllare la presenza di una cartuccia per l'avviamento automatico della ROM, nella locazione $8000 HEX (32768 decimale). Se questa e' presente, l'inizializzazione normale viene sospesa ed il controllo viene trasferito al codice della cartuccia; in caso contrario, continua il normale sistema di inizializzazione.
3) Successivamente il KERNAL inizializza tutti i dispositivi di input/output, ed anche il bus seriale. Entrambi i circuiti CIA 6526 sono impostati ai valori richiesti per la scansione della tastiera; viene attivato il timer a 60 Hz. Il circuito SID viene azzerato. Viene selezionata la mappa della memoria BASIC e viene spento il motore del registratore.
4) Il KERNAL esegue a questo punto un test sulla RAM, impostando i puntatori in cima ed in fondo alla memoria. Viene inizializzata anche la Pagina Zero, e viene impostato il buffer del nastro.
   Il test sulla RAM e' una procedura non distruttiva, che parte dalla locazione $0300 e si muove verso l'alto. Una volta raggiunta la prima locazione non RAM, la cima della RAM si ritrova il puntatore impostato. La base della memoria e' sempre impostata alla locazione $0800, mentre la memoria dello schermo parte dalla locazione $0400.
5) Infine, il KERNAL esegue le altre attivita'. I vettori di I/O vengono impostati a valori di default. La tabella dei salti indiretti viene fissata nella memoria di base. Lo schermo viene azzerato, e vengono riattivate tutte le variabili dell'editor di schermo. Quindi viene usato l'indirizzo posto nella locazione $A000 per dare il via al BASIC.

COME USARE IL KERNAL

Quando si scrivono programmi in Linguaggio Macchina e' conveniente usare le routine riguardanti l'input/output che fanno gia' parte del Sistema Operativo, l'accesso al clock di sistema, la gestione della memoria ed altre operazioni simili. Dato il facile accesso al Sistema Operativo, che rende piu' spedita la programmazione in Linguaggio Macchina, la riscrittura di tali routine non risulta altro che uno sforzo inutile.

Come si e' gia' detto, il KERNAL e' una tabella dei salti, costituita da un insieme di istruzioni JMP che consentono di saltare alle varie routine del Sistema Operativo.

Per usare una procedura del KERNAL e' necessario eseguire per prima cosa tutte le operazioni richieste dalla procedura stessa. Ad esempio, se una procedura del KERNAL richiede di essere chiamata prima di un'altra, essa deve essere chiamata. Se la procedura si aspetta l'inserimento di un numero nell'accumulatore, allora questo numero si deve trovare lì'; diversamente, la procedura ha poche probabilita' di funzionare nel modo previsto.

Dopo essersi preparati, occorre chiamare la procedura per mezzo dell'istruzione JSR. Tutte le procedure del KERNAL a cui si puo'

accedere sono strutturate come SOTTOPROCEDURE, e devono terminare con un'istruzione RTS. Quando la procedura del KERNAL ha terminato il suo compito, il controllo ritorna all'istruzione successiva alla JSR.

Molte procedure del KERNAL restituiscono i codici d'errore nella "Status Word", oppure nell'accumulatore, se insorgono problemi durante il loro svolgimento. La buona pratica della programmazione ed il successo dei programmi scritti in Linguaggio Macchina richiedono un trattamento adeguato di queste procedure: ignorare un ritorno d'errore potrebbe causare il fallimento del resto del programma.

Quando si vuole usare il KERNAL si devono eseguire i tre semplici passi seguenti:

1) Avviamento
2) Chiamare la routine
3) Gestire l'errore

Nella descrizione delle routine del KERNAL si usano le seguenti convenzioni:

NOME DELLA FUNZIONE - Nome della routine del KERNAL.

INDIRIZZO DI CHIAMATA - Indirizzo di chiamata della routine del KERNAL, espresso in esadecimale.

REGISTRI DI COMUNICAZIONE - I registri di questa voce vengono usati per passare/ritornare i parametri alla/dalla rotine del KERNAL.

ROUTINE DI PREPARAZIONE - Alcune routine del KERNAL richiedono che siano passati i dati prima che esse possano operare. Le routine necessarie sono elencate qui di seguito.

RITORNO DELL'ERRORE - Un ritorno da una routine del KERNAL con il riporto impostato indica che durante l'elaborazione si e' verificato un errore. Il numero di tale errore e' contenuto nell'accumulatore.

RICHIESTE DELLO STACK - Numero attuale di byte dello stack usati dalla routine del KERNAL.

REGISTRI INTERESSATI - Tutti i registri usati dalla routine del KERNAL sono riportati qui di seguito.

DESCRIZIONE - Un breve elenco delle funzioni svolte dalle routine del KERNAL e' riportato qui di seguito.

La seguente e' la lista delle routine del KERNAL.

ROUTINE DEL KERNAL RICHIAMABILI DALL'UTENTE

NOME	INDIRIZZO		FUNZIONE
	HEX	DECIMALE	
ACPTR	$FFA5	65445	Accetta un byte dalla porta seriale
CHKIN	$FFC6	65478	Apre il canale di input
CHKOUT	$FFC9	65481	Apre il canale di output
CHRIN	$FFCF	65487	Accetta un carattere dal canale
CHROUT	$FFD2	65490	Immette un carattere nel canale
CIOUT	$FFA8	65448	Trasferisce un byte alla porta seriale
CINT	$FF81	65409	Inizializza l'editor di schermo
CLALL	$FFE7	65511	Chiude tutti i canali ed i files
CLOSE	$FFC3	65475	Chiude un file logico specifico
CLRCHN	$FFCC	65484	Chiude i canali di input e di output
GETIN	$FFE4	65508	Prende il carattere dalla coda (buffer) della tastiera
IOBASE	$FFF3	65523	Ritorna l'indirizzo di base dei dispositivi di I/O
IOINIT	$FF84	65412	Inizializza I'I/O
LISTEN	$FFB1	65457	Dispone a RICEVENTE i dispositivi sul bus seriale
LOAD	$FFD5	65493	Carica la RAM da un dispositivo
MEMBOT	$FF9C	65436	Legge/imposta la base della memoria
MEMTOP	$FF99	65433	Legge/imposta la cima della memoria
OPEN	$FFC0	65472	Apre un file logico
PLOT	$FFF0	65520	Legge/imposta la posizione X, Y del cursore
RAMTAS	$FF87	65415	Inizializza la RAM, alloca il buffer del nastro, imposta lo schermo a $0400
RDTIM	$FFDE	65502	Legge il clock del tempo
READST	$FFB7	65463	Legge la parola di stato di I/O
RESTOR	$FF8A	65418	Ripristina il vettore di default di I/O
SAVE	$FFD8	65496	Salva la RAM su un dispositivo
SCNKEY	$FF9F	65439	Fa la scansione della tastiera
SCREEN	$FFED	65517	Ritorna il sistema di coordinate X, Y di schermo
SECOND	$FF93	65427	Invia I'indirizzo secondario dopo RICEZIONE
SETLFS	$FFBA	65466	Imposta gli indirizzi primario, secondario e logico
SETMSG	$FF90	65424	Controlla i messaggi del KERNAL
SETNAM	$FFBD	65469	Imposta il nome del file
SETTIM	$FFDB	65499	Imposta il clock del tempo
SETTMO	$FFA2	65442	Imposta il supero tempo sul bus seriale
STOP	$FFE1	65505	Termina la scansione della tastiera
TALK	$FFB4	65460	Imposta a TRASMITTENTE il dispositivo del bus seriale
TKSA	$FF96	65430	Invia l'indirizzo secondario dopo TRASMISSIONE
UDTIM	$FFEA	65514	Incrementa il clock del tempo
UNLSN	$FFAE	65454	Imposta il bus seriale a NON-RICEVENTE
UNTLK	$FFAB	65451	Imposta il bus seriale a NON-TRASMITTENTE
VECTOR	$FF8D	65421	Legge/imposta il vettore di I/O

B.1 - Nome della funzione: ACPTR

Scopo: Prende i dati dal bus seriale
Indirizzo di chiamata: $FFA5 (HEX), 65445 (decimale)
Registri di comunicazione: .A
Procedura di preparazione: TALK, TKSA
Errori di ritorno: Vedere READST
Richiesta dello stack: 13
Registri interessati: .A, .X

Descrizione: - Questa routine deve essere usata quando si vogliono ricevere informazioni da un dispositivo sul bus seriale, come ad esempio un disco. Questa routine preleva dal bus seriale un byte del dato usando l'"Handshacking". Il dato viene riportato nell'accumulatore. Per preparare questa routine e' necessario chiamare la routine TALK, che ordina al dispositivo sul bus seriale di trasmettere dati attraverso il bus. Se il dispositivo ha bisogno di un comando secondario, quest'ultimo deve essere inviato per mezzo della routine del KERNAL TKSA, prima che la routine in esame venga chiamata. Gli errori sono riportati nella parola di stato, che viene letta dalla routine READST.

Come si usa:

0) Ordinare a un dispositivo sul bus seriale di predisporsi all'invio dati al COMMODORE 64 (usare le routine del KERNAL TALK e TKSA)
1) Chiamare queste routine (usando JSR)
2) Memorizzare, oppure usare, i dati

ESEMPIO:

```
;PRENDE UN BYTE DAL BUS
  JSR ACPTR
  STA DATA
```

B.2 – Nome della funzione: CHKIN

Scopo: Apre un canale di input
Indirizzo di chiamata: $FFC6 (HEX), 65478 (decimale)
Registri di comunicazione: .X
Procedure di preparazione: (OPEN)
Errori di ritorno:
Richiesta dello stack: Nessuna
Registri interessati: .A, .X

Descrizione: - Ogni file logico che e' gia' stato aperto dalla routine del KERNAL OPEN puo' essere definito, da questa routine, come canale di input. Normalmente, il dispositivo sul canale deve essere un dispositivo di input; diversamente, si verifica un errore e la routine "abortisce"
All'atto della ricezione di dati provenienti da qualunque altra parte diversa dalla tastiera, prima di usare le routine del KERNAL CHRIN o CETIN si deve chiamare questa routine. Se si vuole usare l'input da tastiera, e non sono aperti altri canali di input, le chiamate a questa routine ed alla routine OPEN non sono necessarie.

Quando si usa questa routine in congiunzione ad un dispositivo sul bus seriale, essa provvede ad inviare automaticamente sul bus l'indirizzo di chiamata (e l'indirizzo secondario, se specificato nella routine OPEN).

Come si usa:

0) Aprire (OPEN) il file logico (se necessario: vedere descrizione precedente).
1) Caricare il registro .X con il numero di file logico che deve essere usato.
2) Chiamare questa routine (usando il comando JSR)

Possibili errori:

#3 : File non aperto
#5 : Dispositivo non presente
#6 : Il file non e' un file di input

ESEMPIO:

```
;SI PREPARA PER UN INPUT PROVENIENTE DAL FILE LOGICO 2
LDX #2
JSR CHKIN
```

B.3 – Nome della funzione: CHKOUT

Scopo: Apre un canale di output
Indirizzo di chiamata: $FFC9 (HEX), 65481 (decimale)
Registri di comunicazione: .X
Procedure di preparazione: (OPEN)
Errori di ritorno: 0, 3, 5, 7 (vedere READST)
Richiesta dello stack: 4+
registri interessati: .A, .X

Descrizione: - Ciascun numero di file logico, creato dalla routine del KERNAL OPEN, puo' essere definito come un canale di output. Naturalmente, il dispositivo che si intende usare per aprire un canale deve essere di output, altrimenti si verifica un errore e la routine "abortisce".
Questa routine deve essere chiamata prima che qualsiasi dato venga inviato su un dispositivo di output, a meno che quest'ultimo non sia lo schermo del COMMODORE 64. In questo caso, se non ci sono altri canali di output gia' predisposti, la chiamata a questa routine ed alla routine OPEN non e' necessaria.
Quando si usa questa routine per aprire un canale di un dispositivo sul bus seriale, essa invia automaticamente l'indirizzo di RICEZIONE specificato dalla routine OPEN (e l'indirizzo secondario, se specificato).

Come si usa:

RICORDARE: Questa routine NON E' NECESSARIA per inviare i dati sullo schermo.

0) Usare la routine OPEN del KERNAL per specificare il numero di file logico, l'indirizzo di RICEZIONE e, se specificato, l'indirizzo secondario.
1) Caricare il registro .X con il numero del file logico usato nella istruzione di apertura.
2) Chiamare questa routine usando JSR

ESEMPIO:

```
;DEFINISCE IL FILE LOGICO 3 COME CANALE DI OUTPUT
LDX #3
JSR CHKOUT
```

Possibili errori:

#3 : File non aperto
#5 : Dispositivo non presente
#7 : File non di output

B.4 – Nome della funzione: CHRIN

Scopo: Preleva un carattere dal canale di input
Indirizzo di chiamata: $FFCF (HEX), 65487 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: (OPEN, CHKIN)
Errori di ritorno: 0 (vedere READST)
Richiesta dello stack: 7+
Registri interessati: .A, .X

Descrizione: – Questa routine preleva un byte del dato da un canale gia' predisposto come canale di input dalla routine CHKIN del KERNAL. Se CHKIN non e' stata usata per definire un altro canale di input, allora si prevede che i dati provengano dalla tastiera. Il byte del dato viene sistemato nell'accumulatore. Dopo la chiamata, il canale rimane aperto.

L'input da tastiera e' gestito in modo particolare. Per prima cosa, viene attivato il cursore, che continua a lampeggiare finche' non viene digitato sulla tastiera un ritorno carrello. Tutti i caratteri che si trovano sulla linea (fino a 88) vengono memorizzati nel buffer di input del BASIC, per poi essere ripresi uno alla volta da questa routine. Quando viene chiamato il ritorno carrello, tutta la linea e' stata elaborata. Alla chiamata successiva di questa routine, questo processo, a partire dal lampeggiamento del cursore, viene eseguito di nuovo.

Come si usa:

DA TASTIERA
1) Recuperare un byte del dato tramite la chiamata a questa routine
2) Memorizzare il byte del dato
3) Verificare se si tratta dell'ultimo byte del dato
4) Se non lo e', ritornare al passo 1

ESEMPIO:

```
        LDY $#00        ;PREPARA IL REGISTRO .Y ALLA MEMORIZZAZIONE DEL DATO
    RD  JSR CHRIN
        STA DATA,Y      ;MEMORIZZA L'Y-ESIMO BYTE NELLA Y-ESIMA LOCAZIONE
                         DELL'AREA DATI
        INY
        CMP #CR         ;E' UN RITORNO CARRELLO?
        BNE RD          ;SE NON LO E', PRENDI UN ALTRO BYTE DEL DATO
```

DA ALTRI DISPOSITIVI:

0) Usare le routine OPEN e CHKIN del KERNAL
1) Chiamare questa routine (usando l'istruzione JSR)
2) Memorizzare il dato

ESEMPIO:

```
JSR CHRIN
STA DATA
```

B.5 – Nome della funzione: CHROUT

Scopo: Invia in output un carattere
Indirizzo di chiamata: $FFD2 (HEX), 65490 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: (CHKOUT, OPEN)
Errori di ritorno: 0 (vedere READST)
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: – Questa routine invia in output un carattere su un canale gia' aperto. Prima di chiamare questa routine, usare le routine OPEN e CHKOUT per predisporre il canale di output. Se la chiamata non viene effettuata, il dato viene inviato ad un dispositivo di output standard (il numero 3, cioe' lo schermo). Prima di chiamare questa routine, il byte del dato viene caricato nell'accumulatore, per poi essere inviato al dispositivo di output specificato. Dopo la chiamata, il canale rimane aperto.

NOTA:: Questa routine deve essere trattata con cura quando si intenda inviare il dato ad uno specifico dispositivo seriale, poiche' il dato viene inviato a tutti i canali di output aperti sul bus. Tutti i canali di output aperti sul bus seriale, oltre a quello inteso per la destinazione, devono essere chiusi tramite una chiamata alla routine CLRCHN del KERNAL, a meno che non si desideri diversamente.

Come si usa:

0) Usare la routine CHKOUT del KERNAL, se necessario (vedere la descrizione sopra).
1) Caricare nell'accumulatore il dato da inviare in output.
2) Chiamare questa routine.

ESEMPIO:

```
;DUPLICATO DELL'ISTRUZIONE BASIC CMD 4,"A"
  LDX #4          ;FILE LOGICO #4
  JSR CHKOUT      ;APRE IL CANALE DI OUTPUT
  LDA #'A
  JSR CHROUT      ;INVIA IL CARATTERE
```

B.6 – Nome della funzione: CIOUT

Scopo: Trasmette un byte sul bus seriale
Indirizzo di chiamata: $FFA8 (HEX), 65448 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: LISTEN,[SECOND]
Errori di ritorno: vedere READST
Richiesta dello stack: 5
Registri interessati: nessuno

Descrizione: -- Questa routine e' usata per inviare informazioni ai dispositivi sul bus seriale. Una chiamata a questa routine inserisce il byte del dato sul bus seriale usando l'"Handshacking" seriale. Prima di chiamare questa routine, e' necessario chiamare la routine LISTEN del KERNAL, che ordina ad un dispositivo sul bus seriale di tenersi pronto alla ricezione di dati (se un dispositivo ha bisogno di un indirizzo secondario, si deve inviare anche questo, tramite la routine SECOND del KERNAL). L'accumulatore viene caricato con un byte che viene trasmesso come dato sul bus seriale. Un dispositivo deve essere predisposto a ricevente, altrimenti la parola di stato rileva un "fuori sincronismo". Questa routine "bufferizza" sempre un carattere (la routine, cioe', trattiene il dato precedente a quello ritornato). Percio', quando si chiama la routine UNLSN del KERNAL per porre fine alla trasmissione dei dati, il carattere "bufferizzato" viene inviato con una "End Or Identify" (EOI) impostata; successivamente, al dispositivo viene inviato il comando UNLSN.

Come si usa:

0) Usare la procedura LISTEN del KERNAL (se necessario, anche la procedura SECOND)
1) Caricare l'accumulatore con un byte del dato
2) Chiamare questa procedura per inviare il byte del dato

ESEMPIO:

```
  LDA #'X         ;INVIA UNA X AL BUS SERIALE
  JSR CIOUT
```

B.7 – Nome della funzione: CINT

Scopo: Inizializza l'editor di schermo ed il circuito di controllo del video 6567
Indirizzo di chiamata: $FF81 (HEX), 65409 (decimale)
Registri di comunicazione: nessuno
Procedure di preparazione: nessuna
Errori di ritorno: nessuno
Richiesta dello stack: 4
Registri interessati: .A, .X, .Y

Descrizione: **– Questa routine predispone il circuito di controllo del video 6567del COMMODORE 64 ad un'operazione normale. Viene inizializzato anche l'editor di schermo. Questa routine deve essere chiamata da una cartuccia programma del COMMODORE 64.**

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR CINT
JMP RUN          ;INIZIO DELL'ESECUZIONE
```

B.8 – Nome della funzione: CLALL

Scopo: Chiude tutti i files
Indirizzo di chiamata: $FFE7 (HEX), 65511 (decimale)
Registri di comunicazione: nessuno
Errori di ritorno: nessuno
Richiesta dello stack: 11
Registri interessati: .A, .X

Descrizione: **– Questa routine chiude tutti i files aperti. Quando questa routine viene chiamata, i puntatori alla tabella dei file aperti vengono impostati daccapo a zero, chiudendo tutti i files. Inoltre, viene chiamata automaticamente la routine CLRCHN, che imposta daccapo i canali di I/O.**

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
    JSR CLALL      ;CHIUDE TUTTI I FILES E SELEZIONA I CANALI DI I/O DI
DEFAULT
    JMP RUN        ;INIZIO DELL'ESECUZIONE
```

B.9 – Nome della funzione: CLOSE

Scopo: Chiude un file logico
Indirizzo di chiamata: $FFC3 (HEX), 65475 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: 0, 240 (vedere READST)
Richiesta dello stack: 2+
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine serve per chiudere un file logico dopo che su questo file sono state completate tutte le operazioni di I/O. Questa routine viene chiamata dopo che si e' caricato l'accumulatore con il numero del file logico che deve essere chiuso (il numero e' lo stesso usato per la sua apertura, avvenuta tramite la routine OPEN).

Come si usa:

1) Caricare l'accumulatore con il numero del file logico che deve essere chiuso.
2) Chiamare questa routine.

ESEMPIO:

```
;CLOSE 15
LDA #15
JSR CLOSE
```

B.10 – Nome della funzione: CLRCHN

Scopo: Azzera i canali di I/O
Indirizzo di chiamata: $FFCC (HEX), 65484 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno:
Richiesta di stack: 9
Registri interessati: .A, .X

Descrizione: – Questa routine viene chiamata per azzerare tutti i canali aperti e per restituire ai canali di I/O il loro valore standard originale. Di solito, viene chiamata dopo aver aperto altri canali di I/O (come nastro o disco) e dopo averli usati per operazioni di I/O. Il dispositivo standard di input e' il numero 0 (tastiera), quello standard di output e' il numero 3 (schermo).
Se uno dei dispositivi da chiudere e' la porta seriale, prima di tutto viene inviato@un segnale di UNTLK per azzerare il canale di input, oppure un segnale di@UNLISTEN per azzerare il canale di output. Se questa routine non viene chiamata (ed il ricevitore (o i ricevitori) viene lasciato attivo sul bus seriale), lo stesso dato puo' essere ricevuto contemporaneamente da piu' di un dispositivo del COMMODORE 64. Un modo per trarre vantaggio da tutto cio' e' quello di ordinare alla stampante di COMUNICARE (TALK) e al disco di RICEVERE (LISTEN), permettendo cosi' la stampa diretta di un file su disco.
Questa routine viene chiamata direttamente durante l'esecuzione della routine CLALL del KERNAL.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.

ESEMPIO:

```
JSR CLRCHN
```

B.11 – Nome della funzione: GETIN

Scopo: Prende un carattere
Indirizzo di chiamata: $FFE4 (HEX), 655508 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: CHKIN, OPEN
Errori di ritorno: vedere READST
Richiesta dello stack: 7+
Registri interessati: .A (.X, .Y)

Descrizione: - Se il canale e' la tastiera, questa sottoprocedura rimuove un carattere dalla coda della tastiera e lo restituisce all'accumulatore sottoforma di valore in ASCII. Se la coda e' vuota, il valore che viene riportato nell'accumulatore e' zero. I caratteri sono inseriti automaticamente nella coda da una routine di scansione della tastiera, pilotata da interruzioni, che si chiama SCNKEY. Il buffer della tastiera puo' contenere un massimo di 10 caratteri. Quando il buffer e' pieno, i caratteri in eccedenza sono ignorati fino alla prossima rimozione di un carattere dalla coda. Se il canale e' l'RS-232, allora viene usato solamente il registro .A e viene ritornato un singolo carattere. Per il controllo della validita' si veda READST. Se il canale e' di tipo seriale, oppure e' il registratore o lo schermo, allora chiamare la routine BASIN.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.
2) Controllare se nell'accumulatore c'e' zero (buffer vuoto).
3) Elaborare i dati.

ESEMPIO:

```
;ATTENDE UN CARATTERE
 WAIT JSR GETIN
 CMP #0
 BEQ WAIT
```

B.12 – Nome della funzione: IOBASE

Scopo: Definisce la pagina di memoria per l'I/O
Indirizzo di chiamata: $FFF3 (HEX), 65523 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno:
Richiesta dello stack: 2
Registri interessati: .X, .Y

Descrizione: – Questa routine imposta i registri X e Y all'indirizzo del segmento di memoria dove sono allocati i dispositivi di I/O mappati in memoria. Questo indirizzo puo' quindi essere usato come "offset" per accedere ai dispositivi di I/O mappati nella memoria del COMMODORE 64. L'"offset" rappresenta il numero delle locazioni dall'inizio della pagina sulle quali si vuole allocare il registro di I/O. Il registro .X contiene il byte basso dell'indirizzo, mentre il registro .Y contiene il byte alto dell'indirizzo.
Questa routine e' stata creata per garantire la compatibilta' tra il COMMODORE 64, il VIC 20 ed i prossimi modelli del COMMODORE 64. Se le locazioni per l'I/O, limitatamente ad un programma in Linguaggio Macchina, vengono assegnate tramite una chiamata a questa routine, allora risultano ancora compatibili con le prossime versioni del COMMODORE 64, del KERNAL e del BASIC.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.
2) Memorizzare i registri .X e .Yin due locazioni consecutive
3) Caricare il registro .Y con l'"offset".
4) Accedere a quella locazione di I/O.

ESEMPIO:

```
;IMPOSTA A ZERO (INPUT) IL REGISTRO DIREZIONE DATI DELLA PORTA UTENTE
    JSR IOBASE
    STX POINT      ;IMPOSTA I REGISTRI BASE
    STY POINT+1
    LDY #2
    LDA #0         ;OFFSET PER DDR DELLA PORTA UTENTE
    STA (POINT), Y ;IMPOSTA DDR A ZERO
```

B.13 – Nome della funzione: IOINIT

Scopo: Inizializza i dispositivi di I/O
Indirizzo di chiamata: $FF84 (HEX), 65412 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nesuna
Errori di ritorno:
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine inizializza tutti i dispositivi e le routine di I/O e le rotine. Viene normalmente chiamata come parte della procedura di inizializzazione di una cartuccia programma del

ESEMPIO:

```
JSR IOINIT
```

B.14 – Nome della funzione: LISTEN

Scopo: Predispone il dispositivo sul bus seriale a ricezione
Indirizzo di chiamata: $FFB1 (HEX), 65457 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: Nessuno
Registri interessati: .A

Descrizione: – Questa routine ordina al dispositivo sul bus seriale di ricevere dati. Prima di effettuare la chiamata a questa routine, l'accumulatore deve essere caricato con un numero di dispositivo compreso fra 0 e 31. La funzione LISTEN esegue una OR su tutti i bit del numero per convertirlo ad un indirizzo di ascolto, e per trasmettere poi questo dato sul bus seriale sotto forma di comando. Il dispositivo specificato viene posto nella condizione di ricevere, ed e' quindi pronto ad accettare l'informazione.

Come si usa:

1) Caricare l'accumulatore con il numero del dispositivo a cui si desidera comandare di RICEVERE.
2) Chiamare questa routine usando l'istruzione JSR.

ESEMPIO:

```
;COMANDA AL DISPOSITIVO #8 DI RICEVERE
   LDA #8
   JSR LISTEN
```

B.15 – Nome della funzione: LOAD

Scopo: Carica da dispositivo in RAM
Indirizzo di chiamata: $FFD5 (HEX), 65493 (decimale)
Registri di comunicazione: .A, .X, .Y
Errori di ritorno: 0,4,5,8,9, READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine carica i byte del dato da qualsiasi dispositivo di input nella memoria del COMMODORE 64. Puo' essere usata anche per operazioni di verifica, confrontando il dato sul dispositivo con quello gia' in memoria e lasciando invariato il dato memorizzato in RAM. L'accumulatore viene impostato a 0 per un caricamento e a 1 per una verifica. Se il dispositivo di input e' aperto con indirizzo secondario 0, l'informazione contenuta nell'etichetta di testa

proveniente dal dispositivo viene ignorata. In questo caso, i registri .X e .Y devono contenere l'indirizzo di partenza per il caricamento. Se il dispositivo e' indirizzato con indirizzo secondario 1, 0 o 2, allora il dato viene caricato in memoria a partire dalla locazione di memoria specificata dalla testata. Questa routine ritorna l'indirizzo della piu' alta locazione RAM caricata.
Prima di effettuare la chiamata per questa routine, occorre chiamare le routine SETLFS e SETNAM del KERNAL.

NOTA: NON E' POSSIBILE caricare da tastiera (0), da RS-232 (2) o da schermo (3).

Come si usa:

0) Chiamare le routine SETLFS e SETNAM. Se si desidera un caricamento rilocato, usare la routine SETLFS per inviare un indirizzo secondario 0.
1) Impostare il registro .A a 0 per caricamento e a 1 per verifica.
2) Se si desidera un caricamento rilocato, occorre impostare i registri .X e .Y
all'indirizzo di partenza per il caricamento.
3) Chiamare la routine usando l'istruzione JSR.

ESEMPIO:

```
          ;CARICA UN FILE DA REGISTRATORE
          LDA #DEVICE1        ;IMPOSTA IL NUMERO DI DISPOSITIVO
          LDX #FILENO         ;IMPOSTA IL NUMERO LOGICO DEL FILE
          LDY CMD1            ;IMPOSTA L'INDIRIZZO SECONDARIO
          JSR SETLFS
          LDA #NAME1-NAME     ;CARICA .A CON IL NUMERO DI CARATTERI DEL
                              NOME DEL FILE
          LDX #<NAME          ;CARICA .X CON L'INDIRIZZO DEL NOME DEL
FILE
          LDY #>NAME          ;CARICA .Y CON L'INDIRIZZO DEL NOME DEL
FILE
          JSR SETNAM
          LDA #0              ;IMPOSTA L'INDICATORE A CARICAMENTO
          LDX #$FF            ;ALTERNA LA PARTENZA
          LDY #$FF
          JSR LOAD
          STX VARTAB          ;TERMINE CARICAMENTO
          JMP START
NAME      .BYT 'FILE NAME'
NAME1     ;
```

B.16 – Nome della funzione: MEMBOT

```
Scopo: Imposta la base della memoria
Indirizzo di chiamata: $FF9C (HEX), 65436 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: Nessuna
Registri interessati: .X, .Y
```

Descrizione: - Questa routine e' usata per impostare la base della memoria. Se il bit di riporto dell'accumulatore risulta impostato, allora, quando questa routine viene chiamata, nei registri .X e .Y viene riportato il valore del puntatore al byte piu' basso della RAM. Sul COMMODORE, 64 privo di espansioni di memoria, il valore iniziale di questo puntatore e' $0800 (2048 decimale). Se il bit di riporto dell'accumulatore e' a zero al momento della chiamata della routine, i valori dei registri .X e .Y vengono trasferiti rispettivamente al byte basso ed al byte alto del puntatore all'inizio della RAM.

Come si usa:

```
PER LEGGERE IL VALORE DALLA BASE DELLA RAM

 1) Impostare il riporto
 2) Chiamare questa routine

PER IMPOSTARE IL VALORE DELLA BASE DELLA RAM

 1) Azzerare il riporto
 2) Chiamare questa routine
```

ESEMPIO:

```
  ALZA LA BASE DELLA MEMORIA DI UNA PAGINA
      SEC        ;LEGGE LA BASE DELLA MEMORIA
      JSR MEMBOT
      INY
      CLC        ;IMPOSTA LA BASE DELLA MEMORIA AL NUOVO VALORE
      JSR MEMBOT
```

B.17 – Nome della funzione: MEMTOP

```
Scopo: Imposta la cima della RAM
Indirizzo di chiamata: $FF99 (HEX), 65433 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .X, .Y
```

Descrizione: - Questa routine e' usata per impostare la cima della RAM. Quando si chiama questa routine con il bit di riporto impostato, nei registri .X e .Y viene caricato il valore del puntatore alla cima della RAM. Se invece il bit di riporto dell'accumulatore e' azzerato, nella cima della RAM viene caricato il contenuto dei registri .X e .Y, cambiando cosi' la cima della memoria.

ESEMPIO:

```
;DISALLOCA IL BUFFER DELL'RS-232
   SEC          ;LEGGE LA CIMA DELLA MEMORIA
   JSR MEMTOP
   DEX
   CLC
   JSR MEMTOP   ;IMPOSTA LA NUOVA CIMA DELLA MEMORIA
```

B.18 – Nome della funzione: OPEN

Scopo: Apre un file logico
Indirizzo di chiamata: $FFC0 (HEX), 65472 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: SETLFS, SETNAM
Errori di ritorno: 1,2,4,5,6,240,READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine e' usata per aprire un file logico. Una volta predisposto, il file logico puo' essere usato per le operazioni di input/output. La maggior parte delle routine di I/O del KERNAL chiamano questa routine per creare il file logico su cui lavorare. Prima di usare questa routine, occorre chiamare le routine SETLFS e SETNAM del KERNAL.

Come si usa:

0) Usare la routine SETLFS.
1) Usare la routine SETNAM.
2) Chiamare questa routine.

ESEMPIO:

Implementazione dell'istruzione BASIC OPEN 15,8,15,"I/O"

```
        LDA #NAME2-NAME     ;LUNGHEZZA DEL NOME DEL FILE PER SETLFS
        LDY #>NAME          ;INDIRIZZO DEL NOME DEL FILE
        LDX #<NAME
        JSR SETNAM
        LDA #15
        LDX #8
        LDY #15
        JSR SETLFS
        JSR OPEN
NAME    .BYT 'I/O'
NAME2
```

B.19 – Nome della funzione: PLOT

Scopo: Imposta la locazione del cursore
Indirizzo di chiamata: $FFF0 (HEX), 65520 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Una chiamata a questa routine, con impostato l'indicatore di riporto dell'accumulatore, causa il caricamento della posizione attuale del cursore sullo schermo (espressa nelle coordinate X e Y) nei registri .X e .Y. Il numero di colonna (0...79) della posizione del cursore e' rappresentato da Y, mentre il numero di riga (0...24) della posizione del cursore e' rappresentato da X. Una chiamata con il bit di riporto azzerato posiziona il cursore nel punto di coordinate X,Y, come determinato dai registri .X e .Y

Come si usa:

PER LEGGERE LA LOCAZIONE DEL CURSORE

1) Impostare l'indicatore di riporto.
2) Chiamare la routine.
3) Prelevare la posizione di X e Y rispettivamente dai registri .X e .Y.

PER IMPOSTARE LA LOCAZIONE DEL CURSORE

1) Azzerare l'indicatore di riporto
2) Impostare i registri .X e .Y alla locazione di cursore desiderata.

ESEMPIO:

```
;POSIZIONA IL CURSORE NEL PUNTO DI COORDINATE (5,10)
;(RIGA=5, COLONNA=10)
  LDX #10
  LDY #5
  CLC
  JSR PLOT
```

B.20 – Nome della funzione: RAMTAS

Scopo: Esegue un test sulla RAM
Indirizzo di chiamata: $FF87 (HEX), 65415 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nesuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine e' usata per eseguire un test sulla RAM e per impostare di conseguenza i puntatori alla cima ed alla base della memoria; inoltre, azzera le locazioni da $0000 a $0101 e da

$0200 a $0BFF, alloca il buffer del registratore ed imposta la base dello schermo a $03FF. Normalmente, questa routine viene chiamata durante il processo di inizializzazione della cartuccia programma del COMMODORE 64.

ESEMPIO:

```
JSR RAMTAS
```

B.21 – Nome della funzione: RDTIM

Scopo: Legge il clock di sistema
Indirizzo di chiamata: $FFDE (HEX), 65502 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine e' usata per leggere il clock di sistema. La risposta del clock e' di 1/60 di secondo. La routine restituisce tre byte. L'accumulatore contiene il byte piu' significativo; il registro indice X contiene il byte piu' significativo seguente; il registro Y contiene l'ultimo byte piu' significativo.

ESEMPIO:

```
JSR RDTIM
STY TIME
STX TIME+1
STA TIME+2
...
TIME *=*+3
```

B.22 – Nome della funzione: READST

Scopo: Legge la parola di stato
Indirizzo di chiamata: $FFB7 (HEX), 65463 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A

Descrizione: Questa routine riporta nell'accumulatore lo stato attuale dei dispositivi di I/O. Solitamente, questa routine viene chiamata dopo aver iniziato una nuova comunicazione verso un dispositivo di I/O. La routine restituisce informazioni sullo stato del dispositivo, oppure gli errori che si sono verificati durante le operazioni di I/O. I bit ritornati nell'accumulatore contengono le seguenti informazioni:

POSIZIONE DEL BIT DI STATO	VALORE NUMERICO DELLO STATO	LETTURA DA REGISTRATORE	R/W SERIALE	VERIFICA+CARICO DEL NASTRO
0	1		Supero Tempo scrittura	
1	2		Supero Tempo lettura	
2	4	Blocco Corto		Blocco Corto
3	8	Blocco Lungo		Blocco Lungo
4	16	Errore lettura irrecuperabile		Qualsiasi Errore
5	32	Errore controllo sommatoria		Errore controllo sommatoria
6	64	Fine File	Fine Linea	
7	-128	Fine Nastro	Dispositivo non presente	Fine Nastro

Come si usa:

1) Chiamare questa routine.
2) Decodificare l'informazione contenuta nel registro .A quando essa si riferisce al programma.

ESEMPIO:

```
;CONTROLLA LA FINE DEL FILE DURANTE UNA LETTURA
 JSR READST
 AND #64          ;CONTROLLA IL BIT DI FINE FILE
 BNE EOF          ;SALTA SE E' FINE FILE
```

B.23 – Nome della funzione: RESTOR

Scopo: Ripristina il sistema standard ed i vettori di interruzione
Indirizzo di chiamata: $FF8A (HEX), 65418 (decimale)
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: 2
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine ripristina i valori standard di tutti i vettori di sistema usati dalle routine e dalle interruzioni del BASIC e del KERNAL (vedere la mappa di memoria per il contenuto di default dei vettori). La routine VECTOR del KERNAL e' usata per leggere e modificare individualmente i vettori di sistema.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
 JSR RESTOR
```

B.24 – Nome della funzione: SAVE

Scopo: Salva la memoria su un dispositivo
Indirizzo di chiamata: $FFD8 (HEX), 65496 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: SETLFS, SETNAM
Errori di ritorno: 5,8,9,READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine salva un segmento di memoria a partire da un indirizzo indiretto, contenuto sulla pagina zero e specificato dall'accumulatore, fino all'indirizzo memorizzato nei registri .X e .Y; successivamente, viene trasferita ad un file logico su un dispositivo di input/output. Prima di usare questa routine, devono essere richiamate le routine SETLFS e SETNAM. Da notare che un salvataggio sul dispositivo 1 (registratore Datassette [TM]) non richiede necessariamente il nome del file. Qualsiasi altro tentativo di salvataggio su altri dispositivi senza usare il nome del file genera un errore.

NOTA: I dispositivi 0 (tastiera), 2 (RS-232) e 3 (schermo) non possono essere salvati.
Qualunque tentativo in questo senso genera un errore e l'interruzione di questa routine.

Come si usa:

0) Usare la routine SETLFS e SETNAM (a meno che non si desideri fare un salvataggio senza alcun nome di file su un registratore).
1) Caricare due locazioni consecutive su pagina zero con un puntatore all'inizio del salvataggio.
2) Caricare l'accumulatore con l'"offset" del singolo byte di pagina zero per il puntatore.
3) Caricare i registri .X e .Y rispettivamente con i byte basso ed alto della locazione dove termina il salvataggiio.
4) Chiamare questa routine.

ESEMPIO:

```
LDA #1              ;DISPOSITIVO 1 = REGISTRATORE
JSR SETLFS
LDA #0              ;NESSUN NOME DI FILE
JSR SETNAM
LDA PROG            ;CARICA L'INDIRIZZO DI PARTENZA DEL SALVATAGGIO
STA TXTTAB          ;BYTE BASSO
LDA PROG+1
STA TXTTAB+1        ;BYTE ALTO
LDX VARTAB          ;CARICA .X CON IL BYTE BASSO DI FINE SALVATAGGIO
LDY VARTAB+1        ;CARICA .Y CON IL BYTE ALTO
LDA #<TXTTAB
JSR SAVE
```

B.25 – Nome della funzione: SCNKEY

Scopo: Scandisce la tastiera
Indirizzo di chiamata: $FF9F (HEX), 65439 (decimale)

Registri di comunicazione: Nessuno
Procedura di preparazione: IOINIT
Errori di ritorno: Nessuno
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine esegue la scansione della tastiera del COMMODORE 64 individuando i tasti premuti. Questa routine viene chiamata anche dal gestore delle interruzioni. Se si preme un tasto, il corrispondente valore ASCII viene riportato nella coda della tastiera. Questa routine viene chiamata solamente se viene superata l'interruzione IRQ normale.

Come si usa:

0) Chiamare questa routine.

ESEMPIO:

```
GET  JSR SCNKEY     ;SCANDISCE LA TASTIERA
     JSR GETIN      ;PRELEVA UN CARATTERE
     CMP #0         ;E' IL CARATTERE NULLO ("") ?
     BEQ GET        ;SE SI, RICOMINCIA LA SCANSIONE
     JSR CHROUT     ;ALTRIMENTI, STAMPA IL CARATTERE
```

B.26 – Nome della funzione: SCREEN

Scopo: Ritorna il formato dello schermo
Indirizzo di chiamata: $FFED (HEX), 65517 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Richiesta dello stack: 2
Registri interessati: .X, .Y

Descrizione: - Questa routine ritorna il formato dello schermo, cioe' il numero delle colonne (40) e' contenuto nel registro .X, mentre il numero delle righe (25) e' contenuto nel registro .Y. Questa routine puo' essere usata per determinare su quale tipo di macchina il programma sta funzionando. Questa funzione e' stata implementata sul COMMODORE 64 per facilitare la compatibilita' dei programmi.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR SCREEN
STX MAXCOL
STY MAXROW
```

B.27 – Nome della funzione: SECOND

Scopo: Invia l'indirizzo secondario per LISTEN
Indirizzo di chiamata: $FF93 (HEX), 65427 (decimale)
Registri di comunicazione: .A

Procedure di preparazione: LISTEN
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine e' usata per inviare un indirizzo secondario ad un dispositivo di I/O dopo che e' stata effettuata la chiamata alla routine LISTEN, predisponendo il dispositivo per la ricezione. Questa routine non puo' essere usata per inviare un indirizzo secondario dopo che si e' effettuata una chiamata alla routine TALK.
L'indirizzo secondario si usa di solito per inviare ad un dispositivo informazioni di preparazione prima di iniziare le operazioni di I/O.

Come si usa:

1) Caricare l'accumulatore con l'indirizzo secondario da inviare.
2) Chiamare la routine.

ESEMPIO:

```
;INDIRIZZA IL DISPOSITIVO #8 CON IL COMANDO (INDIRIZZO SECONDARIO)#15
    LDA #8
    JSR LISTEN
    LDA #15
    JSR SECOND
```

B.28 – Nome della funzione: SETLFS

Scopo: Predispone un file logico
Indirizzo di chiamata: $FFBA (HEX), 65466 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: Nessuno

Descrizione: - Questa routine imposta il numero del file logico, l'indirizzo del dispositivo e l'indirizzo secondario (numero del comando) per le routine del KERNAL. Il numero di file logico e' usato dal sistema come chiave di accesso alla tabella dei files creata dalla routine OPEN di apertura dei files. I valori degli indirizzi del dispositivo va da 0 a 31. I seguenti codici sono utilizzati dal COMMODORE 64 per supportare i dispositivi CBM riportati qui di seguito:

INDIRIZZO	DISPOSITIVO
0	Tastiera
1	Datassette [TM] #1
2	Dispositivo RS-232C
3	CRT Video
4	Stampante a bus seriale
8	Unita' disk CBM a bus seriale

I dispositivi di numero maggiore o uguale a 4 fanno automaticamente riferimento a dispositivi sul bus seriale. Un comando ad un dispositivo viene inviato come indirizzo secondario sul bus seriale dopo l'invio del numero del dispositivo stesso, durante la sequenza seriale che si occupa dell'"handshacking". Se non viene inviato alcun indirizzo secondario, allora il registro indice .Y deve essere impostato a 255.

Come si usa:

1) Caricare l'accumulatore con il numero di file logico.
2) Caricare il registro indice .X con il numero del dispositivo.
3) Caricare il registro indice .Y con il comando.

ESEMPIO:

PER IL FILE LOGICO 32, DISPOSITIVO #4, E NESSUN COMANDO:

```
LDA #32
LDX #4
LDY #255
JSR SETLFS
```

B.29 – Nome della funzione: SETMSG

Scopo: Controlla i messaggi di output del sistema
Indirizzo di chiamata: $FF90 (HEX), 65424 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: 2
Registri interessati: .A

Descrizione: – Questa routine controlla la stampa degli errori ed i messaggi per mezzo del KERNAL. Impostando l'accumulatore al momento della chiamata della routine, si puo' scegliere di stampare i messaggi di errore oppure i messaggi di controllo. FILE NOT FOUND e' un esempio di messaggio di errore, mentre PRESS PLAY ON CASSETTE e' un esempio di messaggio di controllo.
I bit 6 e 7 di questo valore determinano la provenienza del messaggio: se il bit 7 contiene 1, viene stampato uno dei messaggi di errore provenienti dal KERNAL; se invece viene impostato il bit 6, vengono stampati i messaggi di controllo.

Come si usa:

1) Impostare l'accumulatore al valore desiderato.
2) Chiamare questa routine.

ESEMPIO:

```
LDA #$40
JSR SETMSG      ;ATTIVA I MESSAGGI DI CONTROLLO
LDA #$80
JSR SETMSG      ;ATTIVA I MESSAGGI DI ERRORE
LDA #0
JSR SETMSG      ;DISATTIVA TUTTI I MESSAGGI DEL KERNAL
```

B.30 – Nome della funzione: SETNAM

Scopo: Predispone il nome del file
Indirizzo di chiamata: $FFBD (HEX), 65469 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedura di preparazione: Nessuna
Richiesta dello stack: Nessuna
Registri interessati: Nessuno

Descrizione: - Questa routine e' usata per predisporre il nome del file per le routine 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 nel formato standard del 6502 byte basso/byte alto. L'indirizzo puo' essere un qualsiasi indirizzo di memoria valido nel sistema dove e' memorizzata la stringa di caratteri usata per il nome del file. Se non si vuole dare un nome al file, occorre impostare nell'accumulatore il valore 0, che rappresenta una lunghezza di file pari a zero. In questo caso i registri .X e .Y possono essere impostati a qualsiasi indirizzo di memoria.

Come si usa:

1) Caricare l'accumulatore con la lunghezza del nome del file.
2) Caricare il registro indice .X con l'indirizzo piu' basso del nome del file.
3) Caricare il registro indice .Y con l'indirizzo piu' alto del nome del file.
4) Chiamare questa routine.

ESEMPIO:

```
LDA #NAME2-NAME     ;CARICA LA LUNGHEZZA DEL NOME DEL FILE
LDX #<NAME          ;CARICA L'INDIRIZZO DEL NOME DEL FILE
LDY #>NAME
JSR SETNAM
```

B.31 – Nome della funzione: SETTIM

Scopo: Imposta il clock di sistema
Indirizzo di chiamata: $FFDB (HEX), 65499 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: Nessuno

Descrizione: - Il clock di sistema viene gestito da una routine di interruzione che aggiorna il clock ogni 1/60 di secondo (questa quantita' viene anche indicata con "jiffy"). Il clock e' lungo tre byte, che gli consentono di contare fino a 5.184.000 "jiffy", dopodiche' ricomincia da zero. Prima di chiamare questa routine, l'accumulatore deve contenere il byte piu' significativo, il registro .X il byte piu' significativo seguente ed il registro .Y l'ultimo byte piu' significativo dell'impostazione iniziale del tempo (espresso in

"jiffy").

Come si usa:

1) Caricare l'accumulatore con l'MSB del numero di tre byte per impostare il clock.
2) Caricare il registro .X con il byte successivo.
3) Caricare il registro .Y con l'LSB (Byte meno significativo).
4) Chiamare questa routine.

ESEMPIO:

```
;IMPOSTA IL CLOCK A 10 MINUTI = 3600 "JIFFY"
 LDA #0              ;BYTE PIU' SIGNIFICATIVO
 LDX #>3600
 LDY #<3600          ;BYTE MENO SIGNIFICATIVO
 JSR SETTIM
```

B.32 – Nome della funzione: SETTMO

Scopo: Imposta l'indicatore di supero tempo della scheda del bus IEEE
Indirizzo di chiamata: $FFA2 (HEX), 65442 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richieste dello stack: 2
Registri interessati: Nessuno

NOTA: Questa routine e' usata solo con una scheda aggiuntiva IEEE.

Descrizione: - Questa routine imposta l'indicatore di supero tempo per il bus IEEE. Quando l'indicatore di supero tempo e' impostato, il COMMODORE 64 attende per 64 millisecondi la comparsa di un dispositivo sulla porta IEEE. Se tale dispositivo non risponde al segnale di Indirizzo Dati Valido (DAV - Data Address Valid) del COMMODORE 64 entro tale tempo, il COMMODORE 64 emette una condizione di errore lasciando la sequenza di "handshacking". Quando si chiama questa routine con il bit 7 dell'accumulatore impostato a 0, viene attivato il supero tempo. Se invece questo bit e' a 1, il supero tempo viene disabilitato.

NOTA: Il COMMODORE 64 usa la caratteristica del supero tempo per comunicare che non ha trovato un file su disco, durante un tentativo di apertura di file usando una scheda IEEE.

Come si usa:

PER IMPOSTARE L'INDICATORE SUPERO TEMPO:
1) Impostare a 0 il bit 7 dell'accumulatore.
2) Chiamare questa routine.

PER AZZERARE L'INDICATORE DI SUPERO TEMPO:
1) Impostare a 0 il bit 7 dell'accumulatore.
2) Chiamare questa routine.

ESEMPIO:

```
;DISABILITA IL SUPERO TEMPO
 LDA #0
 JSR SETTMO
```

B.33 – Nome della funzione: STOP

Scopo: Controlla se e' stato premuto il tasto STOP
Indirizzo di chiamata: $FFE1 (HEX), 65505 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: Nessuna
Registri interessati: .A, .X

Descrizione: – Se si preme il tasto STOP sulla tastiera durante una chiamata alla routine UDTIM, viene impostato l'indicatore Z. Inoltre, i canali vengono ripristinati ai valori standard. Tutti gli altri indicatori rimangono immutati. Se il tasto STOP non e' stato premuto, allora l'accumulatore contiene un byte che rappresenta l'ultima linea di scansione della tastiera. Questa routine puo' essere usata dall'Utente per controllare anche gli altri tasti.

Come si usa:

0) Chiamare la funzione UDTIM.
1) Chiamare questa routine.
2) Controllare se l'indicatore e' a zero.

ESEMPIO:

```
    JSR UDTIM     ;SCANDISCE LA TASTIERA PER CONTROLLARE SE SI E' PREMUTO
                  ;IL TASTO STOP
    JSR STOP
    BNE *+5       ;TASTO NON PREMUTO
    JMP READY     ;STOP
```

B.34 – Nome della funzione: TALK

Scopo: Ordina ad un dispositivo sul bus seriale di comunicare
Indirizzo di chiamata: $FFB4 (HEX), 65460 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: – Per usare questa routine e' necessario innanzitutto caricare nell'accumulatore il numero del dispositivo (che deve essere compreso fra 0 e 31). Una volta chiamata, questa routine esegue una OR bit per bit per convertire il numero del dispositivo in un indirizzo di colloquio. Successivamente, questo dato viene trasmesso come

comando sul bus seriale.

Come si usa:

1) Caricare l'accumulatore con il numero del dispsitivo
2) Chiamare questa routine.

ESEMPIO:

```
;ORDINA AL DISPOSITIVO #4 DI COMUNICARE
 LDA #4
 JSR TALK
```

B.35 – Nome della funzione: TKSA

Scopo: Invia un indirizzo secondario al dispositivo predisposto al colloquio
Indirizzo di chiamata: $FF96 (HEX), 65430 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: TALK
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine trasmette un indirizzo secondario sul bus seriale al dispositivo che deve colloquiare. Per chiamare questa routine, l'accumulatore deve contenere un numero compreso fra 0 e 31. La routine invia questo numero sul bus seriale sottoforma di comando relativo all'indirizzo secondario. Questa routine puo' essere chiamata solamente dopo aver chiamato la funzione TALK; chiamata invece dopo la funzione LISTEN, non avra' effetto.

Come si usa:

0) Chiamare la routine TALK.
1) Caricare l'accumulatore con l'indirizzo secondario.
2) Chiamare questa routine.

ESEMPIO:

```
;COMUNICA AL DISPOSITIVO #4 DI COLLOQUIARE CON IL COMANDO #7
  LDA #4
  JSR TALK
  LDA #7
  JSR TALKSA
```

B.36 – Nome della funzione: UDTIM

Scopo: Aggiorna il clock di sistema
Indirizzo di chiamata: $FFEA (HEX), 65514 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno

Richiesta di stack: 2
Registri interessati: .A, .X

Descrizione: - Questa routine aggiorna il clock di sistema. Di solito, questa routine viene chiamata da una normale routine di interruzione del KERNAL ogni 1/60 di secondo. Se il programma utente elabora interruzioni proprie, per aggiornare il tempo e' necessario chiamare questa routine. Se, inoltre, si desidera che il tasto STOP rimanga funzionale, e' necessario chiamare la routine STOP.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR UDTIM
```

B.37 – Nome della funzione: UNLSN

Scopo: Invia un comando di non-ascolto
Indirizzo di chiamata: $FFAE (HEX), 65454 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine ordina a tutti i dispositivi sul bus seriale di interrompere la ricezione di dati dal COMMODORE 64 (UNLSN = UNLISTEN - Non-ascolto). Una chiamata a questa routine provoca la trasmissione sul bus seriale di un comando UNLISTEN (non-ascolto). Questo comando interessa solamente i dispositivi precedentemente individuati. Normalmente, questa routine viene usata dopo che il COMMODORE 64 ha terminato l'invio di dati ad un dispositivo esterno. Il comando UNLISTEN, inviato a dispositivi posti in ascolto, permette di lasciare il bus seriale in modo tale che quest'ultimo possa essere impiegato per altri scopi.

Come si usa:

1) Chiamare questa routine.

ESEMPIO:

```
JSR UNLSN
```

B.38 – Nome della funzione: UNTLK

Scopo: Invia un comando di non-colloquio
Indirizzo di chiamata: $FFAB (HEX), 65451 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna

Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: – Questa routine trasmette sul bus seriale un comando di interruzione-trasmissione. Tutti i dispositivi precedentemente disposti al colloquio interrompono l'invio di dati dal momento della ricezione di questo comando.

Come si usa:

1) Chiamare questa routine.

ESEMPIO:

```
JSR UNTALK
```

B.39 – Nome della funzione: VECTOR

Scopo: Gestisce i vettori della RAM
Indirizzo di chiamata: $FF8D (HEX), 65421 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richieste dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine gestisce tutti gli indirizzi di salto del vettore di sistema memorizzato nella RAM. Se si chiama questa routine quando il bit di riporto dell'accumulatore e' impostato, si ottiene una lista, puntata dai registri .X e .Y, comprendente la memorizzazione del contenuto attuale dei vettori della RAM. Se questa routine viene chiamata quando il bit di riporto e' azzerato, allora la lista utente puntata dai registri .X e .Y viene trasferita nei vettori della RAM di sistema.

NOTA: L'uso di questa routine richiede una certa attenzione. Il modo migliore di usare questa routine consiste nel leggere innanzitutto il contenuto del vettore nell'area utente; poi, modificare i vettori desiderati; quindi, copiare di nuovo il contenuto nei vettori di sistema.

Come si usa:

PER LEGGERE I VETTORI DELLA RAM DI SISTEMA:
1) Impostare il riporto.
2) Impostare i registri .X e .Y all'indirizzo in cui posizionare i vettori.
3) Chiamare questa routine.

PER CARICARE I VETTORI DELLA RAM DI SISTEMA:
1) Impostare il riporto.
2) Impostare i registri .X e .Y all'indirizzo della lista dei vettori della RAM da caricare
3) Chiamare questa routine.

ESEMPIO:

```
;CARICA LE ROUTINE DI INPUT NEL NUOVO SISTEMA
LDX #<USER
LDY #>USER
SEC
JSR VECTOR          ;LEGGE I VECCHI VETTORI
LDA #<MYINP         ;CAMBIA L'INPUT
STA USER+11
LDX #<USER
LDY #>USER
CLC
JSR VECTOR          ;CAMBIA IL SISTEMA
...
USER *=*+26
```

CODICI ERRORE

La seguente lista riporta i messaggi di errore che possono verificarsi usando le routine del KERNAL. Se si verifica un errore durante lo svolgimento di una routine del KERNAL, viene impostato il bit di riporto dell'accumulatore, e nel medesimo viene riportato il numero corrispondente al messaggio di errore.

NOTA: Alcune routine di I/O del KERNAL non usano questi codici di errore. In questo caso gli errori possono essere identificati usando la routine READST del KERNAL.

CODICE	SIGNIFICATO
0	Routine terminata dal tasto STOP
1	Troppi file aperti
2	File gia' aperto
3	File non aperto
4	File non trovato
5	Dispositivo non presente
6	File non di input
7	File non di output
8	Manca il nome del file
9	Numero di dispositivo illegale
240	La cima della memoria modifica l'allocazione/disallocazione del buffer RS-232

USO DEL LINGUAGGIO MACCHINA DA BASIC

Esistono numerosi modi per usare il BASIC ed il linguaggio macchina sul COMMODORE 64, comprese particolari istruzioni come parte del CBM BASIC, come pure locazioni chiave nella macchina. Ci sono cinque modi principali di usare sul COMMODORE 64 le routine in linguaggio macchina:

1) L'istruzione BASIC SYS
2) La funzione BASIC USR
3) La modifica di uno dei vettori di I/O della RAM
4) La modifica di uno dei vettori di interruzione della RAM
5) La modifica della routine CHRGET

1) L'istruzione BASIC SYS X provoca il salto ad una sottoprocedura in linguaggio macchina allocata nell'indirizzo X. Questa routine deve terminare con un'istruzione RTS (ReTurn from Subroutine - RiTorno da Sottoprocedura), che ritorna il controllo al BASIC. Tra la routine in linguaggio macchina ed il programma in BASIC, i parametri vengono passati usando le istruzioni BASIC PEEK e POKE ed i loro equivalenti in linguaggio macchina.
Il comando SYS e' il metodo piu' comune per unire il BASIC al linguaggio macchina. La PEEK e la POKE consentono un facile passaggio di parametri multipli. In un programma ci possono essere diverse istruzioni SYS, ciascuna destinata a differenti (in qualche caso, alla stessa) routine in linguaggio macchina.

2) La funzione BASIC USR(X) passa il controllo alla sottoprocedura in Linguaggio Macchina allocata nella posizione indicata dall'indirizzo memorizzato nelle locazioni 785 e 786 (tale indirizzo e' memorizzato nel formato standard byte basso/byte alto). Il valore di X viene valutato e passato alla sottoprocedura in Linguaggio Macchina tramite l'accumulatore reale #1, il cui inizio e' stabilito all'indirizzo $61 (per ulteriori dettagli si veda la mappa di memoria). Al programma BASIC puo' essere ritornato un valore, preventivamente sistemato nell'accumulatore reale; il ritorno al BASIC si verifica se la routine termina con l'istruzione RTS.
L'istruzione USR(X) e' diversa da SYS, in quanto necessita dell'impostazione di un vettore indiretto, che rappresenta il formato attraverso il quale la variabile viene passata (formato in virgola mobile). Se si usa piu' di una routine in Linguaggio Macchina, e' necessario modificare il vettore.

3) Ciascuna routine interna di input/output o del BASIC, a cui si fa accesso per mezzo della tabella dei vettori allocata a pagina 3 (vedere MODI DI INDIRIZZAMENTO, PAGINA ZERO), puo' essere sostituita o modificata da un codice utente. Ogni vettore di due byte e' formato da un indirizzo, composto da un byte alto ed un byte basso, a disposizione del sistema operativo.
La routine VECTOR del KERNAL rappresenta il modo piu' sicuro per modificare uno qualunque di tali vettori; tuttavia, un singolo vettore puo' essere modificato anche dall'istruzione POKE. In

questo caso, un nuovo vettore punta alla routine preparata dall'Utente e che ha lo scopo di sostituire o incrementare la routine standard del sistema. La routine Utente viene lanciata al momento dell'esecuzione del comando BASIC appropriato. Se dopo l'esecuzione di questa routine e' necessario eseguire una normale routine di sistema, allora il programma Utente deve prevedere un'istruzione di salto (JMP) all'indirizzo precedentemente conetnuto nel vettore. In caso contrario, la routine deve terminare con un'istruzione RTS che restituisce il controllo al BASIC.

4) Si puo' modificare il VETTORE DELLE INTERRUZIONI HARDWARE (IRQ). Ogni 1/60 di secondo il sistema operativo passa il controllo alla routine, specificata da questo vettore, che di solito viene usata per la sincronizzazione, per la scansione della tastiera, ecc. Se si usa questa tecnica occorre trasferire sempre il controllo alla routine di gestione dell'IRQ normale, a meno che la routine di sostituzione sia in grado di operare sul circuito CIA (se CIA e' gestito dalla routine, RICORDARSI di terminare la routine con l'istruzione RTI [ReTurn from Interrupt - RiTorno da Interruzione]).
Questo metodo e' valido per la determinazione di quello che si deve verificare in concorrenza ad un programma BASIC, ma presenta lo svantaggio di essere piu' difficile.

NOTA: PRIMA DI MODIFICARE QUESTO VETTORE, DISABILITARE SEMPRE LE INTERRUZIONI !

5) La routine CHRGET e' usata dal BASIC per prelevare ogni carattere o simbolo, cosi' da semplificare l'aggiunta di nuovi comandi BASIC. Naturalmente, ciascun nuovo comando deve essere eseguito da una sottoprocedura in Linguaggio Macchina scritta dall'Utente. Un modo comune per usare questo metodo e' quello di specificare un carattere (@, per esempio) che deve comparire prima di ogni altro nuovo comando. La nuova routine CHRGET cerca i caratteri speciali: se non ne vengono trovati, il controllo passa alla normale routine CHRGET del BASIC; in caso contrario, il nuovo comando viene interpretato ed eseguito dal programma in Linguaggio Macchina. Si minimizza cosi' il tempo extra di esecuzione necessario alla ricerca dei comandi aggiuntivi. Questa tecnica viene spesso indicata come "wedge".

DOVE MEMORIZZARE LE ROUTINE IN LINGUAGGIO MACCHINA

Il posto migliore per allocare le routine in linguaggio macchina sul COMMODORE 64 sono le locazioni da $C000 a $CFFF, assumendo che le routine siano piu' corte di 4K byte. Questo segmento di memoria non e' influenzato dal BASIC.
Se non e' possibile, o non si desidera, inserire le routine in linguaggio macchina a partire da $C000, per esempio perche' sono piu' lunghe di 4K byte, allora e' necessario proteggere dal BASIC un'area di memoria a partire dalla cima della memoria. Quest'ultima e' allocata in $9FFF, e puo' essere modificata usufruendo della routine MEMTOP del KERNAL, oppure della seguente istruzione:

```
10 POKE51,L:POKE52,H:POKE55,L:POKE56,H:CLR
```

dove H e L sono rispettivamente la porzione alta e bassa della nuova cima della memoria. Ad esempio, per riservare al Linguaggio Macchina l'area compresa fra $9000 e $9FFF, si puo' usare la seguente istruzione:

```
10 POKE51,0:POKE52,144:POKE55,0:POKE56,144:CLR
```

COME SI ACCEDE AL LINGUAGGIO MACCHINA

Per aggiungere programmi in linguaggio macchina ad un programma BASIC ci sono tre modi comuni:

1) ISTRUZIONI DATA:

Le routine in linguaggio macchina possono essere implementate dalla lettura di istruzioni DATA e dall'inserimento di valori (tramite l'istruzione POKE) in memoria all'inizio del programma. Questo e' il metodo piu' facile: non sono necessari metodi speciali per salvare le due parti del programma, ed e' facile da correggere. Gli inconvenienti comportano l'occupazione di maggiore spazio di memoria e l'attesa del caricamento del programma. Percio', questo metodo e' consigliabile per routine brevi.

ESEMPIO:

```
10 RESTORE:FORX=1TO9:READA:POKE12*4096+X,A:NEXT
.
.
.
BASIC PROGRAM
.
.
.
1000 DATA 161,1,204,204,204,204,204,204,96
```

2) MONITOR DEL LINGUAGGIO MACCHINA (64MON):

Questo programma permette di accedere ad un programma in entrambi i codici ESADECIMALE e SIMBOLICO, e di salvare il segmento di memoria in cui si trova il programma. I vantaggi di questo metodo sono un piu' facile accesso alla routine in linguaggio macchina, semplificazioni della messa a punto ed un buon numero di mezzi veloci di salvataggio e caricamento. L'inconveniente e' rappresentato dalla richiesta al programma BASIC di caricare all'inizio la routine in linguaggio macchina da cassetta o da disco (per ulteriori chiarimenti sulla cartuccia 64MON si veda la sezione sul linguaggio macchina).

ESEMPIO:

Il seguente e' un esempio di programma BASIC che usa una routine in Linguaggio Macchina preparata dalla cartuccia 64MON. La routine e' memorizzata su cassetta.

```
10 IF FLAG=1 THEN 20
15 FLAG=1:LOAD "MACHINE LANGUAGE ROUTINE NAME",1,1
20
  .
  .
  .
  PARTE RIMANENTE DEL PROGRAMMA BASIC
```

3) PACKAGE EDITOR/ASSEMBLER:

I vantaggi e gli svantaggi sono simili all'uso del monitor del linguaggio macchina, solo che l'accesso ai programmi e' piu' semplice.

MAPPA DELLA MEMORIA DEL COMMODORE 64

LABEL	INDIRIZZO ESADECIMALE	LOCAZIONE ESADECIMALE	DESCRIZIONE
D6510	0000	0	Registro direzione dati del circuito 6510
R6510	0001	1	Registro a 8 bit di Input/Output del circuito 6510
	0002	2	Non usato
ADRAY1	0003-0004	3-4	Vettore salti: Conversione reale-intero
ADRAY2	0005-0006	5-6	Vettore salti: Conversione intero-reale
CHARAC	0007	7	Carattere di ricerca
ENDCHR	0008	8	Indicatore: Cerca le virgolette alla fine di una stringa
TRMPOS	0009	9	Colonna di schermo dopo l'ultima TAB
VERCK	000A	10	Indicatore: 0=Carica, 1=Verifica
COUNT	000B	11	Puntatore buffer di input/numero indici
DIMFLG	000C	12	Indicatore: Dimensione di default di una schiera
VALTYP	000D	13	Tipo di dato: $FF=Stringa, $00=Numerico
INTFLG	000E	14	Tipo di dato: $80=Intero, $00=Reale
GARBFL	000F	15	Scansione istruzione DATA/Virgolette istruzione LIST/"Garbage Collection"
SUBFLG	0010	16	Indicatore: Riferimento indice/Chiamata di funzione Utente
INPFLG	0011	17	Indicatore: $00=INPUT, $40=GET, $98=READ
TANSGN	0012	18	Indicatore: Simbolo TAN/Risultato di un confronto
	0013	19	Indicatore: Richiesta di INPUT
LINNUM	0014-0015	20-21	Transiente: Valore intero
TEMPPT	0016	22	Puntatore: Stack stringhe tansienti
LASTPT	0017-0018	23-24	Ultimo indirizzo stringhe transienti
TEMPST	0019-0021	25-33	Stack stringhe transienti
INDEX	0022-0025	34-37	Area puntatori programmi di utilita'
RESHO	0026-002A	38-42	Prodotto di moltiplicazione reale
TXTTAB	002B-002C	43-44	Puntatore: Inizio del testo BASIC
VARTAB	002D-002E	45-46	Puntatore: Inizio variabili del BASIC
ARYTAB	002F-0030	47-48	Puntatore: Inizio schiere del BASIC
STREND	0031-0032	49-50	Puntatore: Fine schiere del BASIC (+1)
FRETOP	0033-0034	51-52	Puntatore: Base della memoria stringa
FRESPC	0035-0036	53-54	Puntatore stringa programmi di utilita'
MEMSIZ	0037-0038	55-56	Puntatore: Indirizzo piu' alto usato dal BASIC
CURLIN	0039-003A	57-58	Numero di linea corrente del BASIC
OLDLIN	003B-003C	59-60	Numero di linea precedente del BASIC
OLDTXT	003D-003E	61-62	Puntatore: Istruzione BASIC per CONT
DATLIN	003F-0040	63-64	Numero di linea DATA corrente
DATPTR	0041-0042	65-66	Puntatore: Indirizzo elemento corrente dell'istruzione DATA
INPPTR	0043-0044	67-68	Vettore: Routine di INPUT
VARNAM	0045-0046	68-69	Nome variabile corrente del BASIC
VARPNT	0047-0048	70-71	Puntatore: Dato variabile corrente

			del BASIC
FORPNT	0049-004A	73-74	Puntatore: Variabile indice per il ciclo FOR...NEXT
	004B-0060	75-96	Area puntatore / dati transiente
FACEXP	0061	97	Accumulatore reale #1: Esponente
FACHO	0062-0065	98-101	Accumulatore reale #1: Mantissa
FACSGN	0066	102	Accumulatore reale #1: Segno
SGNFLG	0067	103	Puntatore: Costante di valutazione delle serie
BITS	0068	104	Accumulatore reale #1: Cifra di overflow
ARGEXP	0069	105	Accumularore reale #2: Esponente
ARGHO	006A-006D	106-109	Accumulatore reale #2: Mantissa
ARGSGN	006E	110	Accumulatore reale #2: Segno
ARISGN	006F	111	Risultato di confronto del segno: Accumulatore #1 contro Accumulatore #2
FACOV	0070	112	Accumulatore reale #1. Byte basso (arrotondamento)
FBUFPT	0071-0072	113-114	Puntatore: Buffer cassetta
CHRGET	0073-008A	115-138	Sottoprocedura: Preleva il prossimo byte del testo BASIC
CHRGOT	0079	121	Ingresso per un nuovo prelievo dello stesso byte di testo
TXTPTR	007A-007B	122-123	Puntatore: Byte corrente del testo BASIC
RNDX	008B-008F	139-143	Valore reale del seme della funzione RND
STATUS	0090	144	Parola di stato dell'I/O del KERNAL: ST
STKEY	0091	145	Indicatore: Tasto STOP / Tasto RVS
SVXT	0092	146	Costante di misura del tempo per nastro
VERCK	0093	147	Indicatore: 0=Carica, 1=Verifica
C3PO	0094	148	Indicatore: Bus seriale - Carattere bufferizzato di output
BSOUR	0095	149	Carattere bufferizzato per bus seriale
SYNO	0096	150	Numero di sincronismo cassetta
	0097	151	Area dati transiente
LDTND	0098	152	Numero file aperti/Indice della tabella dei file
DFLTN	0099	153	Dispositivo di input di default (0)
DFLTO	009A	154	Dispositivo di output (CMD) di default (3)
PRTY	009B	155	Parita' carattere nastro
DPSW	009C	156	Indicatore: Ricevuto byte da nastro
MSGFLG	009D	157	Indicatore: $80=Modo diretto, $00=Modo Programma
PTR1	009E	158	Registro errore passo 1 del nastro
PTR2	009F	159	Registro errore passo 2 del nastro
TIME	00A0-00A2	160-162	Clock in tempo reale (approsimato) ad 1/60 di secondo ("Jiffy")
	00A3-00A4	163-164	Area dati transiente
CNTDN	00A5	165	Contatore a ritroso di sincronizzazione cassetta
BUFPNT	00A6	166	Puntatore: Buffer di I/O del nastro
INBIT	00A7	167	Bit di input dell'RS-232/Cassetta Transiente
BITCI	00A8	168	Contatore bit di input dell'RS-232/ Cassetta transiente

RINONE	00A9	169	Indicatore RS-232: Controllo del bit di partenza
RIDATA	00AA	170	Buffer del byte di input dell'RS-232/ Cassetta tansiente
RIPRTY	00AB	171	Parita' input dell'RS-232/Contatore corto cassetta
SAL	00AC-00AD	172-173	Puntatore: Buffer nastro/Scorrimento schermo
EAL	00AE-00AF	174-175	Indirizzi di Fine nastro/Fine programma
CMP0	00B0-00B1	176-177	Costanti di misura del tempo del nastro
TAPE1	00B2-00B3	178-179	Puntatore: Inizio buffer del nastro
BITTS	00B4	180	Contatore bit di output dell'RS-232/ Cassetta transiente
NXTBIT	00B5	181	Prossimo bit dell'RS-232 da inviare/ Indicatore di fine nastro (EOT)
RODATA	00B6	182	Buffer del byte di output dell'RS-232
FNLEN	00B7	183	Lunghezza del nome del file corrente
LA	00B8	184	Numero file logico corrente
SA	00B9	185	Indirizzo secondario corrente
FA	00BA	186	Numero del dispositivo corrente
FNADR	00BB-00BC	187-188	Puntatore: Nome del file corrente
ROPRTY	00BD	189	Parita' output dell'RS-232/Cassetta transiente
FSBLK	00BE	190	Contatore blocco Read/Write cassetta
MYCH	00BF	191	Buffer parola seriale
CAS1	00C0	192	Arresto motore del nastro
STAL	00C1-00C2	193-194	Indirizzo di partenza dell'I/O
MEMUSS	00C3-00C4	195-196	Carico nastro transiente
LSTX	00C5	197	Tasto corrente premuto: CHR$(n) 0=Nessun tasto
NDX	00C6	198	Numero caratteri nel buffer della tastiera (coda)
RVS	00C7	199	Indicatore: Stampa caratteri inversi - 1=Si, 0=Non usato
INDX	00C8	200	Puntatore: Fine linea logica per INPUT
LXSP	00C9-00CA	201-202	Posizione (X,Y) del cursore all'inizio di INPUT
SFDX	00CB	203	Indicatore: Stampa i caratteri ottenuti tenendo premuto il tasto SHIFT
BLNSW	00CC	204	Abilitatore del lampeggio: 0=Lampeggio
BLNCT	00CD	205	Timer: Conto alla rovescia per cursore bistabile
GDBLN	00CE	206	Carattere sotto il cursore
BLNON	00CF	207	Indicatore: Ultima impostazione cursore (lampeggio/fisso)
CRSW	00D0	208	Indicatore: INPUT o GET da tastiera
PNT	00D1-00D2	209-210	Puntatore: Indirizzo della linea di schermo corrente
PNTR	00D3	211	Colonna del cursore sulla linea corrente
QTSW	00D4	212	Indicatore: Editor modo "quote", $00=NO
LNMX	00D5	213	Lunghezza linea di schermo fisica
TBLX	00D6	214	Numero linea fisica attuale del cursore
	00D7	215	Area dati transiente
INSRT	00D8	216	Indicatore: Modo inserimento >0 = # INST
LDTBI	00D9-00F2	217-242	Tavola collegamenti della linea dello

			schermo/Editor transiente
USER	00F3-00F4	243-244	Puntatore: Locazione corrente della RAM colore dello schermo
KEYTAB	00F5-00F6	245-246	Vettore: Tavola di decodificazione della tastiera
RIBUF	00F7-00F8	247-248	Puntatore al buffer di input dell'RS-232
ROBUF	00F9-00FA	249-250	Puntatore al buffer di output dell'RS-232
FREKZP	00FB-00FE	251-254	Libera Pagina 0 per programmi Utente
BASZPT	00FF	255	Area dati transiente del BASIC
	0100-01FF	256-511	Area stack sistema del microprocessore
	0100-010A	256-266	Fluttuante per area di lavoro stringa
BAD	0100-013E	256-318	Registro errori di input del nastro
BUF	0200-0258	512-600	Buffer di INPUT del sistema
LAT	0259-0262	601-610	Tabella KERNAL: Numero file logici attivi
FAT	0263-026C	611-620	Tabella KERNAL: Numero dispositivo per ogni file
SAT	026D-0276	621-630	Tabella KERNAL: Indirizzo secondario di ogni file
KEYD	0277-0280	631-640	Coda del buffer della tastiera (FIFO)
MEMSTR	0281-0282	641-642	Puntatore: Base della memoria per Sistema Operativo
MEMSIZ	0283-0284	643-644	Puntatore: Cima della memoria per Sistema Operativo
TIMOUT	0285	645	Indicatore: Variabile KERNAL per supero Tempo dell'IEEE
COLOR	0286	646	Codice colore del carattere corrente
GDCOL	0287	647	Colore di fondo sotto il cursore
HIBASE	0288	648	Cima della memoria schermo (pagina)
XMAX	0289	649	Misura del buffer della tastiera
RPTFLG	028A	650	Indicatore: Ripete il tasto battuto, $80=Ripete
KOUNT	028B	651	Ripete il contatore velocita'
DELAY	028C	652	Ripete il contatore ritardo
SHFLAG	028D	653	Indicatore: Tasto SHIFT della tastiera/ Tasto CTRL/Tasto C=
LSTSHF	028E	654	Ultima configurazione ottenuta con il tasto SHIFT della tastiera
KEYLOG	028F-0290	655-656	Vettore: Preparazione tabella tastiera
MODE	0291	657	Indicatore: $00=Disabilita tasti SHIFT, $80=Abilita tasti SHIFT
AUTODN	0292	658	Indicatore: Scorrimento automatico verso il basso, 0=ON
M51CTR	0293	659	RS-232: Immagine registro di controllo del 6551
M51CDR	0294	660	RS-232: Immagine del registro di comando del 6551
M51AJB	0295-0296	661-662	BPS RS-232 USA non standard (Tempo/2-100)
RSSTAT	0297	663	RS-232: Immagine del registro di stato del 6551
BITNUM	0298	664	Numero di bit dell'RS-232 rimasti da inviare
BAUDOFF	0299-029A	665-666	Trasmittanza dell'RS-232: Tempo per un bit completo (nsec)

RIDBE	029B	667	Indice RS-232 per termine buffer input
RIDBS	029C	668	Inizio del buffer di input dell'RS-232 (pagina)
RODBS	029D	669	Inizio del buffer di output dell'RS-232 (pagina)
RODBE	029E	670	Indice RS-232 per termine buffer output
IRQTMP	029F-02A0	671-672	Contiene il vettore IRQ durante l'I/O del nastro
ENABL	02A1	673	Abilita l'RS-232
	02A2	674	Lettura di TOD durante I/O cassetta
	02A3	675	Memorizzazione transiente per lettura cassetta
	02A4	676	Indicatore DIIRQ transiente per lettura cassetta
	02A5	677	Transiente per indice di linea
	02A6	678	Indicatore PAL/NTSC, 0=NTSC, 1=PAL
	02A7-02FF	679-767	Non usati
IERROR	0300-0301	768-769	Vettore: Stampa i messaggi di errore del BASIC
IMAIN	0302-0303	770-771	Vettore: Partenza a caldo del BASIC
ICRNCH	0304-0305	772-773	Vettore: Testo BASIC "tokenizzato"
IQPLOP	0306-0307	774-775	Vettore: Lista del testo BASIC
IGONE	0308-0309	776-777	Vettore: Invio caratteri BASIC
IEVAL	030A-030B	778-779	Vettore: Valutazione "token" del BASIC
SAREG	030C	780	Memorizzazione del registro .A del 6502
SXREG	030D	781	Memorizzazione del registro .X
SYREG	030E	782	Memorizzazione del registro .Y
SPREG	030F	783	Memorizzazione del registro .SP
USRPOK	0310	784	Istruzione di salto della funzione USR
USRADD	0311-0312	785-786	Byte basso/alto dell'indirizzo di USR
	0313	787	Non usato
CINV	0314-0315	788-789	Vettore: Interruzione hardware di IRQ
CBINV	0316-0317	790-791	Vettore: Interruzione istruzione BRK
NMINV	0318-0319	792-793	Vettore: Interruzione non mascherabile
IOPEN	031A-031B	794-795	Vettore routine OPEN del KERNAL
ICLOSE	031C-031D	796-797	Vettore routine CLOSE del KERNAL
ICHKIN	031E-031F	798-799	Vettore routine CHKIN del KERNAL
ICKOUT	0320-0321	800-801	Vettore routine CHKOUT del KERNAL
ICLRCH	0322-0323	802-803	Vettore routine CLRCHN del KERNAL
IBASIN	0324-0325	804-805	Vettore routine CHRIN del KERNAL
IBSOUT	0326-0327	806-807	Vettore routine CHROUT del KERNAL
ISTOP	0328-0329	808-809	Vettore routine STOP del KERNAL
IGETIN	032A-032B	810-811	Vettore routine GETIN del KERNAL
ICLALL	032C-032D	812-813	Vettore routine CLALL del KERNAL
USRCMD	032E-032F	814-815	Vettore definito dall'Utente
ILOAD	0330-0331	816-817	Vettore routine LOAD del KERNAL
ISAVE	0332-0333	818-819	Vettore routine SAVE del KERNAL
	0334-033B	820-827	Non usati
TBUFFR	033C-03FB	828-1019	Buffer di I/O del nastro
	03FC-03FF	1020-1023	Non usati
VICSCN	0400-07FF	1024-2047	Area memoria schermo (1024 byte)
	0400-07E7	1024-2023	Matrice video (25 linee X 40 colonne)
	07F8-07FF	2040-2047	Puntatori ai dati animazione
	0800-9FFF	2048-40959	Spazio normale dei programmi BASIC
	8000-9FFF	32768-40959	ROM cartuccia VSP (8192 byte)
	A000-BFFF	40960-49151	ROM BASIC (8192 byte - 8K RAM)
	C000-CFFF	49152-53247	RAM (4096 byte)

D000-DFFF	53248-57343	Dispositivi di I/O e RAM colore, oppure ROM generatore caratteri, oppure RAM (4096 byte)
E000-EFFF	57344-65535	ROM del KERNAL (8192 byte oppure 8K RAM)

ASSEGNAZIONI DI INPUT/OUTPUT DEL COMMODORE 64

ESADECIMALE	DECIMALE	BIT	DESCRIZIONE
0000	0	7-0	Registro direzione dati del 6510 MOS (xx101111) Bit=1:Output, Bit=0:Input, x=Non usato
0001	1		Porta I/O del circuito microprocessore 6510 MOS
		0	Segnale /LORAM(0=Disattiva ROM BASIC)
		1	Segnale /HIRAM(0=Disattiva ROM KERNAL)
		2	Segnale /CHAREN(0=Attiva ROM carattere)
		3	Linea dati di output della cassetta
		4	Lettura interruttore cassetta 1 = Interruttore chiuso
		5	Controllo motore cassetta: 0=ON, 1=OFF
		6-7	Non definiti
D000-D02E	53248-54271		CONTROLLORE INTERFACCIA VIDEO (VIC) MOS 6566
D000	53248		Posizione X dell'animazione 0
D001	53249		Posizione Y dell'animazione 0
D002	53250		Posizione X dell'animazione 1
D003	53251		Posizione Y dell'animazione 1
D004	53252		Posizione X dell'animazione 2
D005	53253		Posizione Y dell'animazione 2
D006	53254		Posizione X dell'animazione 3
D007	53255		Posizione Y dell'animazione 3
D008	53256		Posizione X dell'animazione 4
D009	53257		Posizione Y dell'animazione 4
D00A	53258		Posizione X dell'animazione 5
D00B	53259		Posizione Y dell'animazione 5
D00C	53260		Posizione X dell'animazione 6
D00D	53261		Posizione Y dell'animazione 6
D00E	53262		Posizione X dell'animazione 7
D00F	53263		Posizione Y dell'animazione 7
D010	53264		Posizione X delle animazioni 0-7 (MSB delle coordinate X)
D011	53265		Registro di controllo del VIC
		7	Comparatore di quadro (Bit 8): Vedere 53266
		6	Modo Testo colore esteso 1=Abilitato
		5	Modo Bit Map - 1=Abilitato
		4	Riempie lo schermo con il colore del bordo - 0=Vuoto
		3	Seleziona le righe di testo (24 o 25) video 1=25 Righe
		2-0	Scorrimento rallentato fino alla posizione Y di un punto (0-7)
D012	53266		Lettura/Scrittura del valore di quadro per confronto con IRQ
D013	53267		Posizione X del "latch" penna ottica
D014	53268		Posizione Y del "latch" penna ottica

D015	53269		Abilitatore animazione di schermo 1=Abilitato
D016	53270		Registri di controllo del VIC
		7-6	Non usati
		5	QUESTO BIT DEVE ESSERE SEMPRE 0
		4	Modo Multicolore - 1=Abilitato (Testo o Bit Map)
		3	Seleziona le colonne sdel testo video 1=40 Colonne
		2-0	Scorrimento rallentato a posizione X
D017	53271		Espansione (2X) verticale (Y) animazioni 0-7
D018	53272		Registro di controllo della memoria del VIC
		7-4	Indirizzo base della matrice video (interno al VIC)
		3-1	Indirizzo base del punto di un carattere (interno al VIC)
D019	53273		Registro indicatore di interruzione (Bit=1: si e' verificata una IRQ)
		7	Imposta a ON qualunque condizione IRQ abilitata del VIC
		3	Indicatore IRQ triggerato per penna ottica
		2	Indicatore IRQ di contatto fra due animazioni
		1	Indicatore IRQ di contatto animazione-fondo
		0	Indicatore IRQ di comparazione quadro
D01A	53274		Registro maschera IRQ - 1=Interruzione abilitata
D01B	53275		Priorita' di schermo animazione-fondo 1=Animazione
D01C	53276		Seleziona il Modo Multicolore per le animazioni 0-7 - 1=Modo Multicolore
D01D	53277		Espansione (2X) orizzontale (X) delle animazioni 0-7
D01E	53278		Scoperta contatto fra due animazioni
D01F	53279		Scoperta di contatto animazione-fondo
D020	53280		Colore del bordo
D021	53281		Colore di fondo 0
D022	53282		Colore di fondo 1
D023	53283		Colore di fondo 2
D024	53284		Colore di fondo 3
D025	53285		Registro 0 animazione multicolore
D026	53286		Registro 1 animazione multicolore
D027	53287		Colore animazione 0
D028	53288		Colore animazione 1
D029	53289		Colore animazione 2
D02A	53290		Colore animazione 3
D02B	53291		Colore animazione 4
D02C	53292		Colore animazione 5
D02D	53293		Colore animazione 6
D02E	53294		Colore animazione 7
D400-D7FF	54272-55295		DISPOSITIVO MOS 6581 INTERFACCIA DEL SUONO (SID)
D400	54272		Voce 1: Controllo frequenza

			Byte basso
D401	54273		Voce 1: Controllo frequenza Byte alto
D402	54274		Voce 1: Ampiezza forma d'onda Pulsazione - Byte basso
D403	54275	7-4	Non usati
		3-0	Voce 1: Ampiezza forma d'onda Pulsazione - Semibyte alto
D404	54276		Voce 1: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda triangolare 1=ON
		3	Bit di controllo: 1=Disabilita l'Oscillatore 1
		2	Modulazione ad anello Oscillatore 1 con output Oscillatore 3 - 1=ON
		1	Sincronizza Oscillatore 1 con frequenza Oscillatore 3 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ DECADERE/SOSTENERE, 0=Attiva RILASCIO
D405	54277		Generatore 1 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona la durata del ciclo ATTACCARE: 0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D406	54278		Generatore 1 dell'inviluppo: ciclo di controllo SOSTENERE/RILASCIARE
		7-4	Seleziona la durata del ciclo SOSTENERE: 0-15
		3-0	Seleziona la durata del ciclo RILASCIARE: 0-15
D407	54279		Voce 2: Controllo frequenza Byte basso
D408	54280		Voce 2: Controllo frequenza Byte alto
D409	54281		Voce 2: Ampiezza forma d'onda pulsazione - Byte basso
D40A	54282	7-4	Non usati
		3-0	Voce 2: Ampiezza forma d'onda Pulsazione - Semibyte alto
D40B	54283		Voce 2: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda Triangolare 1=ON
		3	Bit di controllo: 1=Disabilita Oscillatore 2
		2	Modulazione ad anello Oscillatore 2

			con l'uscita Oscillatore 1 - 1=ON
		1	Sincronizza Oscillatore 2 con frequenza Oscillatore 1 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ SOSTENERE/DECADERE, 0=Attiva RILASCIO
D40C	54284		Generatore 2 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona durata ciclo ATTACCARE:0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D40D	54285		Generatore 2 dell'inviluppo: ciclo di controllo SOSTENERE/RILASCIARE
		7-4	Seleziona la durata del ciclo SOSTENERE: 0-15
		3-0	Seleziona la durata del ciclo RILASCIARE: 0-15
D40E	54286		Voce 3: Controllo frequenza Byte basso
D40F	54287		Voce 3: Controllo frequenza-Byte alto
D410	54288		Voce 3: Ampiezza forma d'onda PULSAZIONE - Byte basso
D411	54289	7-4	Non usati
		3-0	Voce 3: Ampiezza forma d'onda PULSAZIONE - Semibyte alto
D412	54290		Voce 3: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda Triangolare 1=ON
		3	Bit di controllo: 1=Disabilita l'Oscillatore 3
		2	Modulazione ad anello Oscillatore 3 con uscita Oscillatore 2 - 1=ON
		1	Sincronizza Oscillatore 3 con frequenza Oscillatore 2 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ DECADERE/SOSTENERE 0=Attiva RILASCIARE
D413	54291		Generatore 3 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona durata ciclo ATTACCARE 0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D414	54292		Generatore 3 dell'inviluppo: ciclo di controllo DECADERE/RILASCIARE
		7-4	Seleziona durata ciclo SOSTENERE:0-15
		3-0	Seleziona durata ciclo RILASCIARE: 0-15
D415	54293		Frequenza di taglio del filtro: semibyte basso (bit 2-0)
D416	54294		Frequenza di taglio del filtro: semibyte alto
D417	54295		Controllo risonanza filtro/Controllo ingresso voce

		7-4	Seleziona risonanza del filtro: 0-15
		3	Ingresso esterno filtro: 1=Si, 0=No
		2	Uscita Voce 3 del filtro: 1=Si, 0=No
		1	Uscita Voce 2 del filtro: 1=Si, 0=No
		0	Uscita Voce 1 del filtro: 1=Si, 0=No
D418	54296		Seleziona modo e volume del filtro
		7	Uscita taglio della Voce 3: 1=Si, 0=No
		6	Seleziona modo passa alto del filtro 1=ON
		5	Seleziona modo passa banda del filtro 1=ON
		4	Seleziona modo passa basso del filtro 1=ON
		3-0	Seleziona volume di uscita: 0-15
D419	54297		Convertitore analogico/digitale: Paddle 1 (0-255)
D41A	54298		Convertitore analogico/digitale: Paddle 2 (0-255)
D41B	54299		Generatore numeri casuali Oscillatore 3
D41C	54300		Uscita generatore 3 dell'inviluppo
D500-D7FF	54528-55295		IMMAGINI DEL SID
D800-DBFF	55296-56319		RAM colore (Semibytes)
DC00-DCFF	56320-56575		Adattatore Interfaccia Complessa (CIA) #1 del 6526 MOS
DC00	56320		Porta dati A (tastiera, joystick, paddle, penna ottica)
		7-0	Scrive i valori della colonna della tastiera per la sua scansione
		7-6	Legge le Paddle sulla porta A/B (01=Porta A, 10=Porta B)
		4	Pulsante sparo joystick A - 1=Fuoco
		3-2	Pulsanti sparo Paddle
		3-0	Direzione joystick A (0-15)
DC01	56321		Porta dati B (tastiera, joystick, paddle)
		7-0	Legge i valori della colonna della tastieraper la sua scansione
		7	Timer B: Uscita bistabile/pulsazione
		6	Timer A: Uscita bistabile/pulsazione
		4	Pulsante sparo joystick 1 - 1=Fuoco
		3-2	Pulsanti sparo paddle
		3-0	Direzione joystick 1
DC02	56322		Registro direzione dati Porta A(56320)
DC03	56323		Registro direzione dati Porta B(56321)
DC04	56324		Timer A: Byte basso
DC05	56325		Timer A: Byte alto
DC06	56326		Timer B: Byte basso
DC07	56327		Timer B: Byte alto

DC08	56328		Clock tempo-del-giorno: Decimi di secondo
DC09	56329		Clock tempo-del-giorno: Secondi
DC0A	56330		Clock tempo-del-giorno: Minuti
DC0B	56331		Clock tempo-del-giorno: Ore+Indicatore AM/PM (bit 7)
DC0C	56332		Buffer dati I/O seriale sincrono
DC0D	56333		Registri controllo interruzione CIA (IRQ lettura/Maschera scrittura)
		7	Indicatore IRQ (1=Si e' verificata una IRQ)/Indicatore Imposta-Azzera
		4	IRQ indicatore 1 (lettura cassetta/ Input SRQ del bus seriale)
		3	Interruzione porta seriale
		2	Interruzione allarme clock tempo-del-giorno
		1	Interruzione Timer B
		0	Interruzione Timer A
DC0E	56334		Registro A di controllo del CIA
		7	Frequenza del clock tempo-del-giorno 1=50 Hz, 0=60 Hz
		6	Modo I/O porta seriale 1=Output, 0=Input
		5	Conteggio Timer A: 1=Segnali di CNT, 0=Clock O2 di sistema
		4	Caricamento forzato Timer A - 1=Si
		3	Modo funzionamento del Timer A 1=Monostabile, 0=Continuo
		2	Modo uscita per PB6 del Timer A 1=Bistabile, 0=Pulsazione
		1	Uscita per PB6 del Timer A-1=Si,0=No
		0	Attiva/Ferma Timer A-1=Attiva,0=Ferma
DC0F	56335		Registro B di controllo del CIA
		7	Imposta Allarme/Clock TOD 1=Allarme, 0=Clock
		6-5	Seleziona modo Timer B: 00 = Conteggio pulsazioni clock O2 di sistema 01 = Conteggio transizioni positive di CNT 10 = Conteggio pulsazioni underflow Timer A 11 = Conteggio pulsazioni underflow Timer A mentre CNT e' positivo
		4-0	Come registro A di controllo del CIA, ma per Timer B
DD00-DDFF	56576-56831		Adattatore interfaccia Complessa (CIA) #2 del 6526 MOS
DD00	56576		Porta dati A (bus seriale, RS-232, Controllo Memoria del VIC)
		7	Ingresso dati bus seriale
		6	Ingresso pulsazioni clock bus seriale
		5	Uscita dati bus seriale
		4	Uscita pulsazioni clock bus seriale
		3	Uscita segnale ATN del bus seriale
		2	Uscita dati dell'RS-232

			01 = Conteggio transizioni positive di CNT 10 = Conteggio pulsazioni underflow del Timer A 11 = Conteggio pulsazioni underflow Timer A mentre CNT e' positivo
		4-0	Come registro di controllo A del CIA, ma per Timer B
DE00-DEFF	56832-57087		Riservati a future espansioni di I/O
DF00-DFFF	57088-57343		Riservati a future espansioni di I/O

CAPITOLO 6

guida all'input/output

- Introduzione
- Output su TV
- Output su altri dispositivi
- Porte Giochi
- Descrizione dell'Interfaccia RS-232
- Porta Utente
- Bus Seriale
- Porta Espansione
- Cartuccia con Microprocessore Z-80

INTRODUZIONE

Le capacita' fondamentali degli elaboratori sono tre: calcolare, prendere decisioni e comunicare. Il calcolo e' probabilmente la cosa piu' facile da programmare, essendo familiari la maggior parte delle regole della matematica. Prendere delle decisioni non e' cosa troppo difficile, poiche' le regole della logica sono relativamente poche.

L'aspetto piu' complesso e' la comunicazione, perche' coinvolge il piu' piccolo insieme di leggi ben definite. Questa non e' una tascuratezza del progetto del computer: le regole offrono un'enorme flessibilita' nel comunicare virtualmente qualunque cosa, nei diversi modi possibili. L'unica vera regola e' la seguente: qualsiasi fonte di informazione deve presentare l'informazione stessa in maniera comprensibile al ricevitore.

OUTPUT SU TV

La forma piu' semplice di output messa a disposizione dal linguaggio BASIC e' l'istruzione PRINT: essa utilizza come dispositivo di output lo schermo della TV; gli occhi invece sono dispositivi di input, con i quali si sfrutta l'informazione presente sullo schermo.

Obiettivo principale della scrittura (con l'istruzione PRINT) sullo schermo e' la costruzione dell'informazione in modo che questa risulti facile da leggere. Si deve cercare di pensare come gli artisti grafici, usando i colori, posizionando lettere maiuscole e minuscole, ricorrendo pure alla grafica per comunicare l'informazione nel migliore dei modi. Basta ricordare che non e' importante l'eleganza del programma, quanto piuttosto la sua capacita' di far capire il significato dei risultati.

L'istruzione PRINT usa numerosi codici carattere come "comandi" per il cursore. Il tasto CRSR non visualizza nulla: permette solamente al cursore di cambiare posizione. Altri comandi cambiano i colori, puliscono lo schermo, ed inseriscono o tolgono gli spazi. Il tasto RETURN ha codice carattere (CHR$) 13; la tabella completa di questi codici e' contenuta nell'Appendice C.

Nel linguaggio BASIC ci sono due funzioni che operano assieme alla funzione PRINT: TAB, che posiziona il cursore sulla posizione assegnata a partire dal margine sinistro dello schermo, e SPC, che sposta il cursore verso destra di un dato numero di spazi a partire dalla posizione attuale.

I due punti (:) nell'istruzione PRINT servono a separare ed a costruire l'informazione, il punto e virgola (;) separa due voci senza alcuno spazio tra di loro. Se quest'ultimo e' l'ultimo carattere della linea, il cursore rimane sulla linea appena stampata senza andare a capo alla linea successiva, sopprimendo (o sostituendo) cosi' il carattere di RETURN, normalmente stampato a fine linea.

La virgola separa dati di stampa all'interno delle colonne. Il COMMODORE 64 ha sullo schermo 4 colonne di 10 caratteri ciascuna; quando viene incontrata una virgola, il computer sposta il cursore all'inizio della colonna successiva. Come per il punto e virgola, se questo e' l'ultimo carattere della linea, il RETURN viene soppresso.

Gli apici distinguono il testo letterale delle variabili; il primo apice delinea l'inizio dell'area letterale, il secondo la fine. Per inciso, non si deve riportare un apice conclusivo alla fine di una linea.

Il codice di RETURN (codice CHR$ 13) fa si' che il cursore si posizioni sulla prossima linea logica dello schermo, che non e' sempre la linea immediatamente successiva. Quando si digita oltre la fine della linea, questa viene concatenata alla linea successiva, per cui il computer sa che entrambi le linee costituiscono in realta' una sola linea di programma. Questi "agganci" sono contenuti nella tabella di "aggancio" delle linee (la costruzione di questa tabella e' spiegata nella mappa della memoria).

Una linea logica puo' essere costituita da una o due linee dello schermo, a seconda di cio' che e' stato digitato o stampato. La linea logica su cui si trova il cursore determina il punto dove il tasto RETURN invia il cursore stesso. La linea logica all'inizio dello schermo determina se il video avanza di una o due linee alla volta.

Ci sono altri modi di usare la TV come dispositivo di output. Il capitolo sulla grafica descrive i comandi che servono per creare oggetti in movimento sullo schermo. La sezione sul circuito VIC illustra come si possono cambiare dimensioni, colori e contorno dello schermo. Il capitolo sul suono mostra come l'altoparlante della TV possa creare musica ed effetti speciali.

OUTPUT SU ALTRI DISPOSITIVI

Spesso e' necessario inviare output su dispositivi diversi dallo schermo, quali registratori, stampanti, unita' a disco o modem. L'istruzione OPEN del BASIC crea un "canale" di colloquio con uno di uesti dispositivi. Una volta che tale canale e' stato aperto, l'istruzione PRINT# invia caratteri a quel dispositivo.

ESEMPIO DI ISTRUZIONI OPEN E PRINT :

```
100 OPEN 4,4:PRINT#4, "SCRITTURA SU STAMPANTE"
110 OPEN 3,8,3,"0:DISK-FILE,S,W":PRINT#3, "INVIATO A DISCO"
120 OPEN 1,1,1,"TAPE-FILE":PRINT#1, "SCRITTURA SU NASTRO"
130 OPEN 2,2,0,CHR$(10):PRINT#2, "INVIATO A MODEM"
```

L'istruzione OPEN e' diversa per ciascun dispositivo. I parametri per l'istruzione OPEN relativi a ciascun dispositivo sono elencati nella seguente tabella:

TABELLA DEI PARAMETRI DELL'ISTRUZIONE OPEN

Formato: OPEN file#,dispositivo#, numero, stringa

DISPOSITIVO	NUMERO DEL DISPOSITIVO	NUMERO	STRINGA
REGISTRATORE	1	0=Input 1=Output 2=Output senza EOT	Nome del file
MODEM	2	0	Registri di controllo
SCHERMO	3	0,1	
STAMPANTE	4 o 5	0=Maiuscole/Grafica 7=Maiuscole/ Minuscole	Stampa il testo
DISCO	da 8 a 11	2-14=Canale dati 15=Comando Canale	Numero Drive, nome del file, tipo del file, lettura / scrittura Comando

OUTPUT SU STAMPANTE

La stampante e' un dispositivo simile allo schermo. Quando si inviano dati alla stampante, l'interesse principale e' quello di creare un formato di facile lettura. Questo compito e' agevolato dai caratteri "reverse", in doppia ampiezza, maiuscoli e minuscoli, come pure dalla grafica programmabile per punti.

La funzione SPC funziona per la stampante allo stesso modo dello schermo. Altrettanto non si puo' dire invece dell'istruzione TAB: essa infatti calcola la posizione attuale sulla linea basandosi sulla posizione del cursore sullo schermo, e non sulla carta.

L'istruzione OPEN usata per la stampante crea il canale di comunicazione, specificando anche quale insieme di caratteri viene usato, se quello "maiuscole e grafica" oppure quello "maiscole e minuscole".

ESEMPI DI ISTRUZIONE OPEN PER STAMPANTE:

```
OPEN 1,4    : REM MAIUSCOLE/GRAFICA

OPEN 1,4,7  : REM MAIUSCOLE/MINUSCOLE
```

Quando si lavora con un insieme di caratteri, si possono stampare singole linee usando l'apposito insieme di caratteri. Quando si lavora con l'insieme maiuscole/grafica, il carattere "cursore verso il basso" [CHR$(17)] predispone i caratteri all'insieme maiuscole/minuscole. Quando invece si lavora con l'insieme maiuscole/minuscole, il carattere "cursore verso l'alto" [CHR$(145)] permette di usare l'insieme maiuscole/grafica.

Altre funzioni speciali della stampante vengono controllate attraverso particolari codici carattere. Tutti questi codici sono semplicemente stampati come ogni altro carattere.

TABELLA DEI CODICI CARATTERE PER IL CONTROLLO DELLA STAMPANTE:

CODICE CHR$	SCOPO
10	Alimentazione linea
13	RETURN (alimentazione linea automatica su stampanti CBM)
14	Inizio carattere a doppia ampiezza
15	Termine carattere a doppia ampiezza
18	Inizio caratteri "reverse"
146	Termine caratteri "reverse"
17	Imposta l'insieme maiuscole/minuscole
145	Imposta l'insieme maiuscole/grafica
16	Tabula la posizione dei successivi due caratteri
27	Spostamento alla posizione del punto specificato
8	Inizio grafica programmabile per punti
26	Ripete i dati della grafica

Per dettagli relativi all'uso dei codici di comando si veda il manuale Commodore della stampante.

OUTPUT SU MODEM

Il modem e' un semplice dispositivo in grado di tradurre i codici carattere in impulsi acustici, e viceversa, permettendo cosi' al computer di comunicare per mezzo delle linee telefoniche. L'istruzione OPEN relativa al modem imposta i parametri per controllare la velocita' ed il formato dell'altro computer con il quale si vuole comunicare. La stringa inviata al termine dell'istruzione OPEN puo' contenere due caratteri.

Le posizioni dei bit del primo codice carattere determinano la trasmittanza (velocita' di manipolazione di una linea), il numero di bit del dato ed il numero di bit di stop. Il secondo codice e' opzionale, ed i suoi bit specificano la parita' ed il duplex della trasmissione. Per particolari specifici su questo dispositivo si veda la sezione sull'RS-232 oppure il manuale **VICMODEM** .

ESEMPIO DI ISTRUZIONE OPEN PER IL MODEM:

```
OPEN 1,2,0,CHR$(6)                :REM 300 BAUD
OPEN 2,2,0,CHR$(163 CHR$(112):REM 110 BAUD, ECC.
```

La maggior parte dei computer usa il Codice Standard Americano per l'Interscambio delle Informazioni (ASCII - American Standard Code for Information Interchange). Questo insieme standard di codici carattere e' un po' diverso dai codici usati dal COMMODORE 64. Quando il COMMODORE 64 deve comunicare con altri computer, i soui codici carattere devono essere tradotti nei corrispondenti codici ASCII. La tabella dei codici ASCII standard e' riportata in Appendice C.

L'output su modem e' un compito assolutamente complicato, eccezion fatta per la necessita' di traduzione dei caratteri. In ogni caso si deve conoscere bene il dispositivo ricevente, specialmente quando si scrivono programmi dove il computer "colloquia" con un altro computer senza l'intervento umano. Un esempio di quanto detto potrebbe essere un programma da terminale che digita automaticamente un numero o una parola d'ordine segreta. Per riuscire a fare cio', si devono contare attentamente i caratteri ed i RETURN. altrimenti il computer,

ricevendo i caratteri, non saprebbe di che cosa farsene.

USO DEI REGISTRATORI A CASSETTA

I registratori hanno una capacita' di memorizzazione dati quasi illimitata: piu' infatti il nastro e' lungo e piu' informazioni puo' memorizzare. Tuttavia, la loro maggiore limitazione e' costituita dal tempo: piu' numerosi sono i dati memorizzati e piu' lungo e' il tempo necessario per la ricerca dei dati stessi.

Quando si lavora con memorizzazioni su nastro e' necessario provare a minimizzare il fattore tempo. La pratica piu' comune consiste nel leggere l'intero file dati da cassetta nella memoria RAM, quindi elaborare e riscrivere tutti i dati su nastro. Questo procedimento permette di eseguire sort, editazioni e controlli sui dati, limitando pero' la dimensione dei file in base alla RAM disponibile.

Se il file dati e' piu' grande della memoria RAM disponibile, e' decisamente preferibile optare per i floppy disk, che permettono la lettura di dati a partire da qualunque posizione, senza la necessita' di leggere tutti i dati precedenti a quello cercato. Inoltre, si possono scrivere dati sopra altri dati piu' vecchi senza perturbare l'ordine della rimanente parte del file. Per questo il disco viene usato in tutte le applicazioni commerciali, quali i libri mastri ed i registri della corrispondenza.

L'istruzione PRINT# formatta i dati allo stesso modo dell'istruzione PRINT; anche tutta la punteggiatura si comporta allo stesso modo. Bisogna pero' tenere ben presente che non si sta lavorando con lo schermo. La formattazione deve essere eseguita tenendo ben presente l'istruzione INPUT#.

Consideriamo l'istruzione INPUT#1,A$,B$,C$. Quando viene usata con lo schermo, le virgole di separazione delle variabili inseriscono spazi bianchi sufficienti a sistemare ogni elemento su una colonna ampia 10 caratteri. Su cassetta, invece, vengono aggiunti da 1 a 10 spazi, a seconda della lunghezza delle stringhe, comportando cosi' uno spreco di spazio sul nastro.

Ancora peggiore e' cio' che accade quando l'istruzione INPUT# cerca di leggere queste stringhe: l'istruzione INPUT#1,A$,B$,C$ non trova alcun dato per B$ e C$, mentre A$ contiene tutte e tre le variabili separate dagli spazi bianchi. Vediamo che cosa e' accaduto al file su nastro:

```
A$="DOG" B$="CAT" C$="TREE"
PRINT# 1, A$, B$, C$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

D O G           C A T                     T R E E  RETURN
```

L'istruzione INPUT# funziona come l'istruzione INPUT: quando si digitano i dati nell'istruzione INPUT, i dati vengono separati dal tasto RETURN oppure dalle virgole. L'istruzione PRINT# inserisce, alla fine della linea, un RETURN, proprio come l'istruzione PRINT. A$ contiene tutti e tre i valori perche' sul nastro non e' presente alcun separatore fra essi (il separatore compare solo alla fine di tutti e tre i valori).

I separatori piu' indicati per il nastro sono la virgola e RETURN ; il codice di quest'ultimo viene inserito automaticamente alla fine dell'istruzione PRINT o PRINT#. Un modo per inserire il codice di RETURN fra gli elementi e' quello di usare solamente una voce per

l'istruzione PRINT#. Un modo ancora migliore consiste nell'impostare una variabile al codice CHR$ di RETURN [CHR$(13)], oppure nell'uso di una virgola. In quest'ultimo caso l'istruzione e' R$=",":PRINT#1,A$ R$ B$ R$ C$. Non si devono inserire virgole o qualsiasi altro carattere di punteggiatura tra i nomi delle variabili, perche' il COMMODORE 64 considera tali variabili separatamente, provocando cosi' uno spreco di spazio nel programma.

Un file registrato su nastro in maniera corretta deve essere simile al seguente:

```
1 2 3 4 5 6 7 8 9 10 11 12 13

D O G , C A T , T R E E RETURN
```

L'istruzione GET# preleva dati da nastro un carattere alla volta, compreso il codice di RETURN e di tutto il resto della punteggiatura. Il codice CHR$(0) viene interpretato come stringa vuota, non come una stringa di un carattere di codice 0. Il tentativo di usare la funzione ASC con una stringa vuota si risolve nel messaggio di errore **ILLEGAL QUANTITY ERROR.**

La riga GET#1,A$:A=ASC(A$) viene usata comunemente per esaminare da programma i dati registrati sul nastro. Per evitare messaggi di errore, la precedente riga puo' essere modificata nel modo seguente:

```
GET#1,A$ : A=ASC(A$+CHR$(0))
```

CHR$(0) posto al termine della stringa mette al riparo da eventuali stringhe vuote, ma non interessa la funzione ASC quando A$ contiene altri caratteri.

MEMORIZZAZIONI DI DATI SU FLOPPY DISK

I dischetti permettono tre diverse forme di memorizzazione. I file sequenziali si comportano come quelli su nastro, ma se ne possono adoperare contemporaneamente piu' di uno. I file relativi consen ono di organizzare i dati in record, e quindi di leggere ed allocare individualmente i record nel file. I file random consentono di lavorare con dati memorizzati in qualunque zona del disco; sono organizzati in ssegmenti di 256 byte chiamati blocchi.

Le limitazioni dell'istruzione PRINT# su disco sono analoghe a quelle riguardanti i nastri. Anche in questo caso sono necessarie le virgole o i RETURN per separare i dati, e CHR$(0) viene ancora letto dall'istruzione GET# come stringa vuota.

I file relativi e quelli random usano entrambi dati separati e comandi di "canale". I dati scritti su disco attraversano il canale dati, dove vengono memorizzati in un buffer transiente situato nella RAM del disco. Quando il blocco e' completo, attraverso un canale di comando viene inviato un comando che comunica al drive dove inserire i dati, quindi l'intero buffer viene scritto.

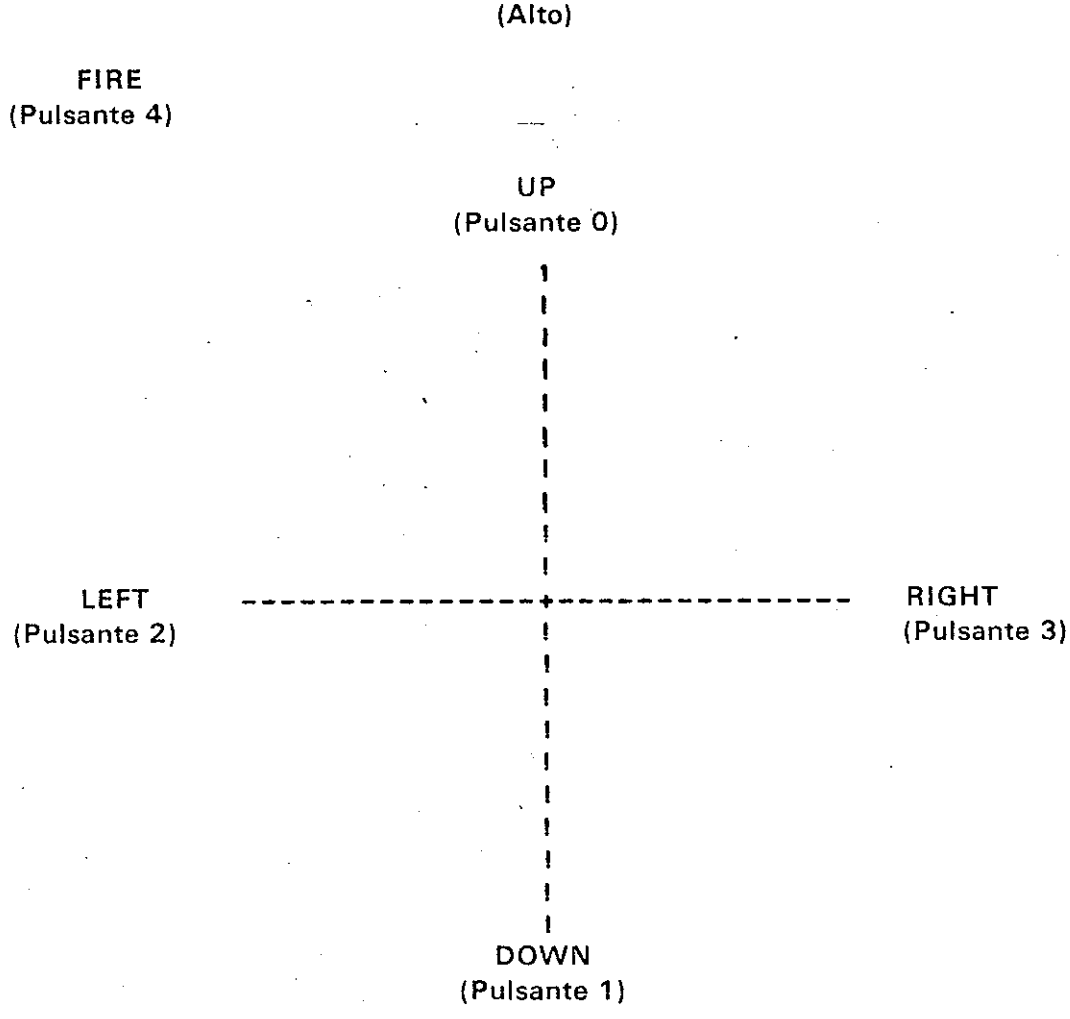
Applicazioni richiedenti un grande numero di dati da elaborare trovano adeguata sistemazione nei file relativi su disco. Una simile organizzazione richiede poco tempo di elaborazione pur garantendo una buona flessibilita' al programma. Una guida completa per la programmazione e l'uso dei file su disco e' riportata nel manuale dell'unita' disco.

PORTE-GIOCHI

Il COMMODORE 64 ha due Porte Giochi a 9 pin che permettono l'uso di joystick, paddle e penna ottica. Ogni porta accetta un joystick o una paddle. La porta A e' l'unica a consentire l'inserimento della penna ottica da adibire a particolari controlli grafici, ecc. Questo paragrafo riporta vari esempi dell'uso di joystick e paddle sia da BASIC che da linguaggio macchina.

Il joystick digitale e' collegato con il CIA#1 (Adattatore Interfaccia Complessa 6526 MOS). Questo dispositivo di input/output gestisce anche i pulsanti di sparo della paddle ed esegue la scansione della tastiera. Il circuito CIA 6526 ha 16 registri locati in memoria da 56320 a 56335 (\$DC00-\$DC0E HEX). I dati della Porta A compaiono nella locazione 56320 (\$DC00 HEX), quelli della Porta B nella locazione 56321 (\$DC01 HEX).

Un joystick digitale ha cinque pulsanti distinti, quattro dei quali sono usati per le direzioni ed uno come pulsante di sparo. La disposizione dei pulsanti del joystick e' la seguente:



Questi interruttori corrispondono ai 5 bit bassi del dato contenuti nelle locazioni 56320 o 56321. Normalmente, un bit e' impostato a 1 se NON viene scelta la direzione o se NON viene premuto il pulsante di sparo. Quando invece quest'ultimo pulsante viene premuto, il corrispondente bit (bit 4, in questo caso) cambia e passa a 0. La lettura da BASIC del joystick puo' essere ottenuta usando la sottoprocedura seguente:

```
10 FORK=0TO10:REM IMPOSTA STRINGA DIREZIONE
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"GOING...";
60 GOSUB100:REM LEGGE IL JOYSTICK
65 IFDR$(JV)=""THEN80:REM CONTROLLA SE SI E' SCELTA UNA DIREZIONE
70 PRINTDR$(JV);" ";:REM STAMPA QUALE DIREZIONE
80 IFFR=16THEN60:REM CONTROLLA SE E' STATO PREMUTO IL PULSANTE DI
81 REM                SPARO
90 PRINT"-----F-----I-----R-----E-----!!!":GOTO60
100 JV=PEEK(56320):REM PRELEVA IL VALORE DEL JOYSTICK
110 FR=JVAND16:REM PREPARA LO STATO DEL PULSANTE DI SPARO
120 JV=15-(JVAND15):REM PREPARA IL VALORE DELLA DIREZIONE
130 RETURN
```

NOTA: Per il secondo joystick impostare JV=PEEK(56321)

I valori di JV corispondono alle seguenti direzioni:

JV	DIREZIONE
0	Nessuna
1	Alto
2	Basso
3	-
4	Sinistra
5	Alto e sinistra
6	Basso e sinistra
7	-
8	Destra
9	Alto e destra
10	Basso e destra

Lo stesso compito viene svolto dalla seguente sottoprocedura:

```
1000 .PAGE (JOYSTICK.8/5)  JOYSTICK - LETTURA PULSANTE DI SPARO
1010 ;
1020 ;AUTORE - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR   LDA $DC00    ; PRELEVA L'INPUT DALLA SOLA PORTA A
1080 DJRRB  LDY #0       ; QUESTA ROUTINE LEGGE E DECODIFICA I DATI
1090        LDX #0       ; DEL PULSANTE DI SPARO DEL JOYSTICK IN INPUT
1100        LSR A        ; ALL'ACCUMULATORE. I 5 BIT MENO SIGNIFICATIVI
1110        BCS DJR0     ; CONTENGONO L'INFORMAZIONE DI CHIUSURA
1120        DEY          ; DEL PULSANTE. SE UN PULSANTE VIENE CHIUSO,
1130 DJR0   LSR A        ; SI PRODUCE UN BIT 0, SE VIENE APERTO SI
1140        BCS DJR1     ; PRODUCE UN BIT 1. LE DIREZIONI DEL JOYSTICK
1150        INY          ; SONO DESTRA, SINISTRA, AVANTI, INDIETRO
1160 DJR1   LSR A        ; BIT3=DESTRA, BIT2=SINISTRA, BIT1=AVANTI,
1170        BCS DJR2     ; BIT0=INDIETRO, BIT4=FUOCO.
1180        DEX          ; AL TEMPO RTS, DX E DY CONTENGONO IL
                         ; COMPLEMENTO
1190 DJR2   LSR A        ; A 2 DEI NUMERI DI DIREZIONE, CIOE' $FF=-1,
```

```
1200        BCS DJR3      ; $00=0,$01=1. DX=1 (DESTRA),DX=-1 (SINISTRA),
1210        INX           ; DX=0 (NESSUN CAMBIAMENTO DI X), DY=-1 (ALTO),
1220 DJR3   LSR A         ; D1=1 (BASSO), DY=0 (NESSUN CAMBIAMENTO DI Y)
1230        STX DX        ; LA POSIZIONE "INDIETRO" DEL JOYSTICK
                          ; CORRISPONDE
1240        STY DY        ; AL MOVIMENTO VERSO L'ALTO DELLO SCHERMO, LA
1250        RTS           ; POSIZIONE "AVANTI" AL MOVIMENTO VERSO IL
1260 ;                      BASSO DELLO SCHERMO. AL TEMPO RTS,
1270 ; L'INDICATORE DI RIPORTO CONTIENE LO STATO DEL PULSANTE
1280 ; DI SPARO. SE C=1 IL PULSANTE NON E' PREMUTO, SE C=0 E' PREMUTO
1290 ;
1300 .END
```

PADDLES

Una paddle puo' essere connessa sia al circuito CIA#1 che al circuito SID (Dispositivo Interfaccia del Suono 6581 MOS) attraverso una porta giochi. Il valore della paddle e' letto attraverso i registri SID 54297 ($D419 HEX) e 54298 ($D41A HEX). LA LETTURA ESCLUSIVAMENTE DA BASIC DELLE PADDLES NON E' ATTENDIBILE!!!! Il modo migliore di usare le paddle, da BASIC o da codice macchina, e' usare la seguente sottoprocedura in linguaggio macchina (accesso da BASIC consentito dall'istruzione SYS, quindi lettura tramite l'istruzione PEEK delle locazioni di memoria usate dalla sottoprocedura)...

```
1000 ;*****************************************************************
1010 ;* ROUTINE DI LETTURA DI 4 PADDLE (UTILIZZABILE ANCHE PER 2) *
1020 ;*****************************************************************
1030 ; AUTORE - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150    LDX #1          ; PER 4 PADDLE OPPURE 2 JOYSTICK
1160 PDLRD0             ; PUNTO DI INGRESSO PER UNA COPPIA
1170    SEI
1180    LDA CIDDRA      ; PRELEVA IL VALORE CORRENTE DI DDR
1190    STA BUFFER      ; E LO SALVA
1200    LDA #$C0
1210    STA CIDDRA      ; IMPOSTA LA PORTA A PER L'INPUT
1220    LDA #$80
1230 PDLRD1
1240    STA PORTA       ; INDIRIZZA UNA COPPIA DI PADDLE
1250    LDY #$80        ; BREVE PAUSA
1260 PDLRD2
1270    NOP
1280    DEY
1290    BPL PDLRD2
1300    LDA SID+25      ; PRELEVA IL VALORE DI X
1310    STA PDLX,X
```

```
1320     LDA SID+26       ; PRELEVA IL VALORE DI Y
1330     STA PDLY,X
1340     LDA PORTA        ; TEMPO LETTURA PULSANTI DI SPARO DELLA PADDLE
1350     ORA #$80         ; COME SOPRA, PER LA SECONDA COPPIA
1360     STA BTNA         ; BIT 2 CORRISPONDE A PDL X E BIT 3 A PDL Y
1370     LDA #$40
1380     DEX              ; SONO STATE LETTE TUTTE LE COPPIE ?
1390     BPL PDLRD1       ; NO
1400     LDA BUFFER
1410     STA CIDDRA       ; RIPRISTINA IL PRECEDENTE VALORE DI DDR
1420     LDA PORTA+1      ; PER LA SECONDA COPPIA -
1430     STA BTNB         ; BIT 2 CORRISPONDE A PDL X E BIT 3 A PDL Y
1440     CLI
1450     RTS
1460 .END
```

Le paddle possono essere lette usando il seguente programma BASIC:

```
10 C=12*4096:REM IMPOSTA IL PUNTO DI PARTENZA DELLA SOTTOPROCEDURA
11 REM SCRIVE TRAMITE POKE NELLA SOTTOPROCEDURA DI LETTURA
12 REM DELLE PADDLE
15 SYSC:REM RICHIAMA LA SOTTOPROCEDURA DELLE PADDLE
30 P1=PEEK(C+257):REM IMPOSTA IL VALORE DELLA PADDLE 1
40 P2=PEEK(C+258):REM  "   "       "  "   "        "   2
50 P3=PEEK(C+259):REM  "   "       "  "   "        "   3
60 P4=PEEK(C+260):REM  "   "       "  "   "        "   4
61 REM LEGGE LO STATO DEL PULSANTE DI SPARO
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM STAMPA I VALORI DELLE PADDLE
72 REM STAMPA LO STATO DEL PULSANTE DI SPARO
75 PRINT:PRINT"FIRE A ";S1;"FIRE B ";S2
80 FORW=1TO50:NEXT:REM BREVE ATTESA

90 PRINT"[SHIFT CLR/HOME]" :PRINT:GOTO20:REM AZZERA LO SCHERMO E RICOMINCIA
95 REM DATI PER SOTTOPROCEDURA IN LINGUAGGIO MACCHINA
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,1,220,141
140 DATA3,193,88,96
```

PENNA OTTICA

L'ingresso penna ottica registra in un circuito "latch", sul fianco di un impulso in caduta, la corrente posizione dello schermo, utilizzando una coppia di registri (LPX, LPY). Il registro 19 ($13 HEX) posizione X contiene gli 8 MSB della posizione X all'istante della transizione. Poiche' la posizione X e' definita da un contatore a 512 posizioni (9 bit), viene fornita una risoluzione di due punti orizzontali. Analogamente, la posizione Y viene registrata nel circuito "latch" del registro 20 ($14 HEX); in questo caso, gli 8 bit forniscono, all'interno dello schermo visibile, una risoluzione di quadro singola. Il circuito "latch" della penna ottica puo' essere triggerato solamente una volta per quadro, per cui tutti gli scatti seguenti non hanno alcun effetto. Occorre percio' eseguire diverse prove (mediamente da tre in su) prima di iniziare ad operare sullo schermo con la penna ottica; il numero di prove da eseguire varia

tuttavia in base alle caratteristiche della penna ottica impiegata.

DESCRIZIONE DELL'INTERFACCIA RS-232

SCHEMA GENERALE

Il COMMODORE 64 incorpora un'interfaccia per il collegamento con un qualunque modem, stampante o altri dispositivi compatibili con l'RS-232. Per collegare un dispositivo al COMMODORE 64 occorrono una certa capacita' ed un po' di programmazione.

L'RS-232 installata sul COMMODORE 64 risponde al formato RS-232 standard, ma le tensioni elettriche sono di livello TTL (0...5V) piuttosto che dell'intervallo -12...+12 V. Le necessarie conversioni della tensione di voltaggio tra il COMMODORE 64 e l'RS-232 sono svolte dalla cartuccia interfaccia Commodore RS-232.

L'accesso al software dell'interfaccia RS-232 puo' avvenire sia da BASIC che da KERNAL, per la programmazione in linguaggio macchina.

A livello BASIC, l'RS-232 usa i normali comandi del BASIC: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#; le variabili riservate ST.INPUT# e GET# vanno a prelevare i dati dal buffer ricevente, mentre PRINT# e CMD vanno a sistemare i dati nel buffer trasmittente. Piu' avanti in questo capitolo sara' spiegato piu' dettagliatamente l'uso di questi comandi.

I gestori dell'RS-232 a livello di bit e byte del KERNAL funzionano sotto il controllo dei timer e delle interruzioni del dispositivo 6526 CIA #2. Un'elaborazione RS-232 genera all'interno del circuito 6526 una serie di richieste NMI (Non Maskable Interrupt - Interruzioni Non Mascherabili). Questo permette un'elaborazione RS-232 di fondo durante l'esecuzione di programmi BASIC ed in linguaggio macchina. Per evitare la distruzione di dati memorizzati, o per prevenirne la trasmissione attraverso le NMI generate dalle sottoprocedure dell'RS-232, all'interno delle sottoprocedure del KERNAL, del registatore e del bus seriale sono state previste particolari aree di memoria riservate. Durante le attivita' del registratore o del bus seriale, NON si possono ricevere dati da dispositivi RS-232. Ma, poiche' tali aree di memoria riservata sono solamente locali (supponendo che la programmazione sia stata accorta), non si genera alcuna interferenza.

Nell'interfaccia RS-232 del COMMODORE 64 ci sono due buffer che aiutano a prevenire eventuali perdite di dati durante la ricezione o la trasmissione di informazioni RS-232.

I buffer RS-232 del KERNAL del COMMODORE 64 sono costituiti da due buffer "FIFO" (First In, First Out - primo entrato, primo uscito), ciascuno lungo 256 byte, situati all'inizio della memoria. L'apertura di un canale RS-232 alloca automaticamente per questi buffer 512 byte di memoria. Se oltre la fine di un programma BASIC non c'e' spazio libero a sufficienza, non viene stampato alcun messaggio di errore, ma la fine del programma viene distrutta. Quindi, ATTENZIONE!

Con il comando CLOSE questi buffer vengono automaticamente rimossi dalla memoria.

APERTURA DI UN CANALE RS-232

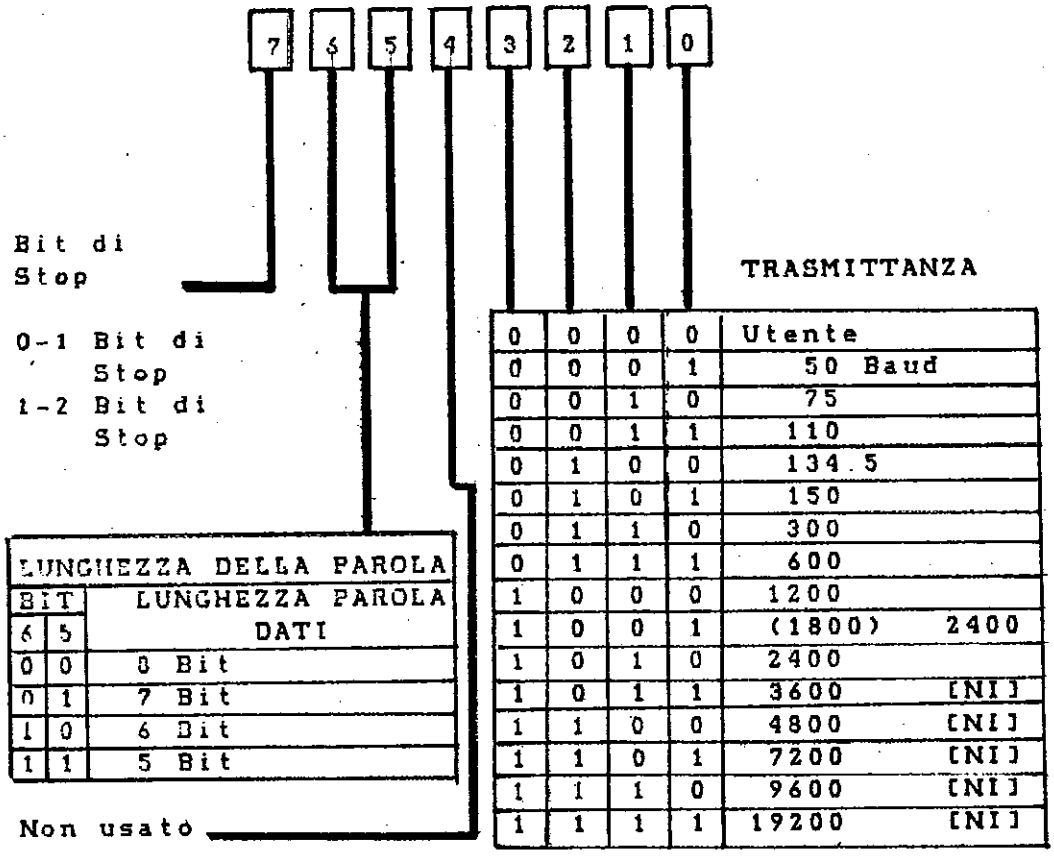
Non si puo' tenere aperto piu' di un canale RS-232 alla volta: una seconda istruzione OPEN causa infatti la nuova impostazione dei puntatori al buffer, provocando di conseguenza la perdita di tutti i

caratteri contenuti nel buffer trasmittente o ricevente.
Il campo nome del file puo' contenere fino a 4 caratteri: i primi due sono i caratteri di controllo e di comando dei registri, gli altri due sono riservati alle future opzioni di sistema. velocita' di trasmissione (espressa in baud), parita' ed altre opzioni possono essere selezionate attraverso questa caratteristica.
Sulla parola di controllo non viene eseguito alcun controllo di errore per scoprire una trasmittanza non implementata. Ciascuna parola di controllo illecita fa si' che l'output di sistema lavori a velocita' molto basse (al di sotto di 50 baud).

SINTASSI BASIC:

OPEN lfn,2,0, "<registro di controllo> <registro di comando> <opt baud low> <opt baud high>"

LFN - Numero di file logico; e' un qualunque numero compreso fra 1 e 255. Da notare che un numero di file maggiore di 127 comporta un avanzamento automatico delle linee dopo ogni ritorno carrello.



LUNGHEZZA DELLA PAROLA		
BIT		LUNGHEZZA PAROLA DATI
6	5	
0	0	8 Bit
0	1	7 Bit
1	0	6 Bit
1	1	5 Bit

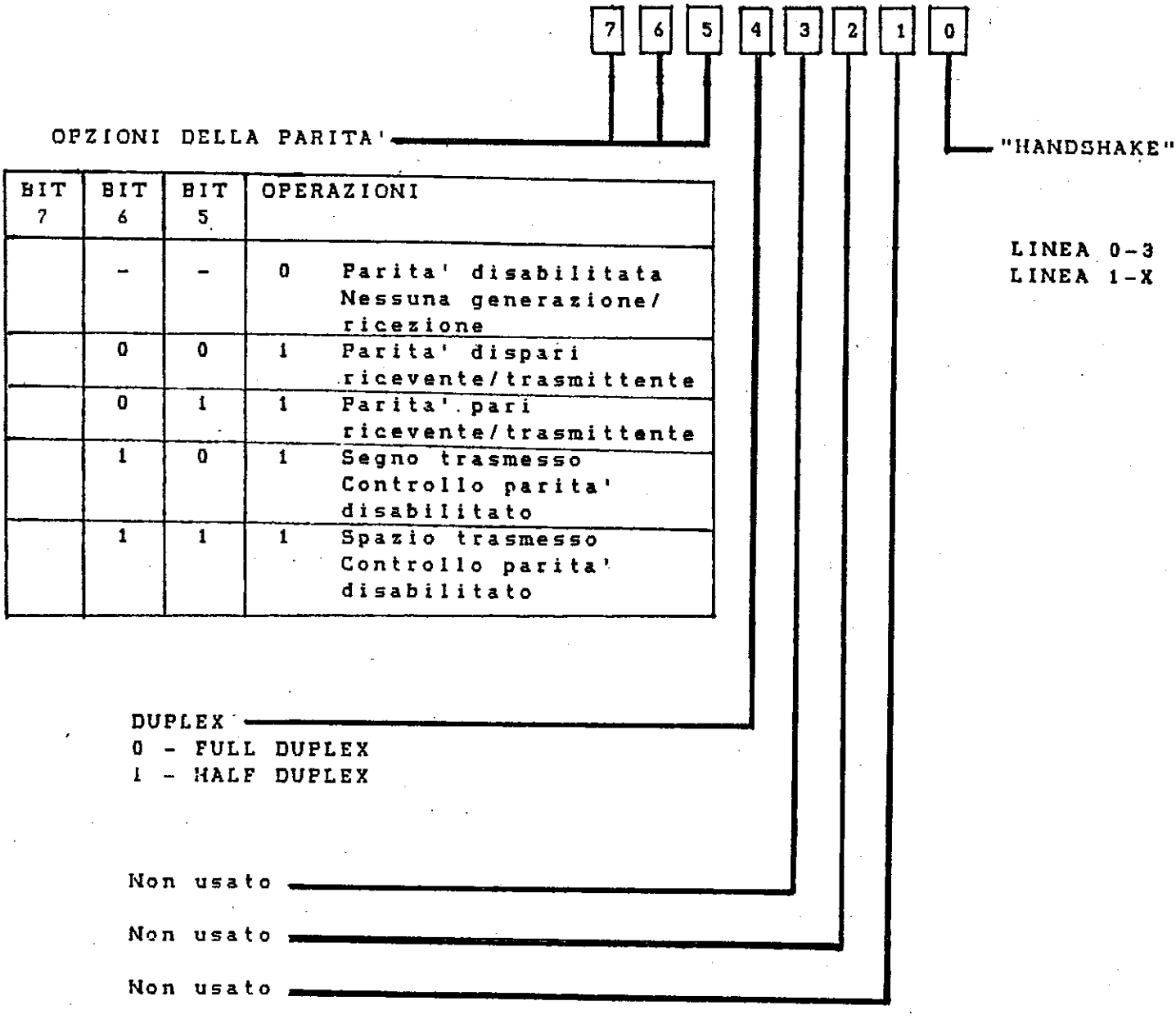
0	0	0	0	Utente
0	0	0	1	50 Baud
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	(1800) 2400
1	0	1	0	2400
1	0	1	1	3600 [NI]
1	1	0	0	4800 [NI]
1	1	0	1	7200 [NI]
1	1	1	0	9600 [NI]
1	1	1	1	19200 [NI]

Figura 6.1 - Mappa del registro di controllo

<registro di controllo> - Carattere contenuto in un singolo byte (vd. fig. 6.1), richiesto per specificare la trasmittanza. Se il piu' basso dei 4 bit della trasmittanza e' uguale a zero, i caratteri <opt baud low> e <opt baud high> forniscono una tasmittanza calcolata come segue:

<opt baud low> = <frequenza di sistema/velocita'/2-100 - <opt baud high> * 256

(opt baud high) = INT[(frequenza di sistema/velocita'/2-100) / 256]



BIT 7	BIT 6	BIT 5	OPERAZIONI
	-	-	0 Parita' disabilitata Nessuna generazione/ ricezione
	0	0	1 Parita' dispari ricevente/trasmittente
	0	1	1 Parita' pari ricevente/trasmittente
	1	0	1 Segno trasmesso Controllo parita' disabilitato
	1	1	1 Spazio trasmesso Controllo parita' disabilitato

Figura 6.2 - Mappa del registro di comando

Le formule precedentemente riportate si basano sulle seguenti considerazioni:

Frequenza di sistema = 1.02273E6 NTSC (North American TV Standard -Standard TV nordamericano)
= 0.98525E6 PAL (Standard TV inglese e di molti paesi europei)

<registro di comando> - Carattere su singolo byte (vd. fig. 6.2) che definisce altri parametri del terminale. NON e' obbligatorio.

INGRESSO KERNAL

OPEN ($FFC0) (Per ulteriori informazioni sulle condizioni e sulle istruzioni di ingresso si veda la descrizione dettagliata del KERNAL).

NOTA IMPORTANTE: In un programma BASIC, il comando OPEN dell'RS-232 puo' essere effettuato prima di creare qualunque variabile o schiera, perche' viene eseguito un CLR automatico al momento dell'apertura di un canale RS-232 (cio' grazie ai 512 byte allocati all'inizio della memoria. Da ricordare che il programma viene distrutto se i 512 byte dello spazio non sono disponibili al momento dell'istruzione OPEN.

PRELIEVO DI DATI DA UN CANALE RS-232

Al momento del prelievo di dati da un canale RS-232, il buffer ricevente del COMMODORE 64 arresta l'ingresso dei caratteri al 255-esimo, prima che il buffer stesso vada in overflow. Questo fatto e' riportato nella parola di stato dell'RS-232 (ST in BASIC, RSSTAT in linguaggio macchina). Se si verifica un overflow, vengono persi tutti i caratteri ricevuti dopo che il buffer e' risultato pieno. E' quindi consigliabile tenere il buffer piu' vuoto possibile.
Se si desidera che la ricezione dei dati avvenga ad alta velocita' (il BASIC puo' solo andare velocemente, anche in considerazione del "garbage collection", e causare di conseguenza l'overflow del buffer ricevente), e' necessario usare sottoprocedure in linguaggio macchina per trattare questo tipo di flusso di dati.

SINTASSI BASIC:

Consigliata: GET# lfn, <variabile stringa>

SCONSIGLIATA: INPUT# lfn, <lista variabile>

INGRESSI KERNAL:

CHKIN ($FFC6) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

GETIN ($FFE4) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

CHRIN ($FFCF) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

NOTE:

* Se la lunghezza della parola e' inferiore a 8 bit, tutti i bit inutilizzati vengono impostati a zero.
* Se GET# non trova alcun dato nel buffer, ritorna un carattere nullo ("").
* Se si usa INPUT#, il sistema resta in condizione di attesa fino alla ricezione di un carattere nullo ed un successivo ritorno carrello. Tuttavia, se scompare la linea CTS (Clear To Send - Azzera per invio) o DSR (DataSsette Ready - registratore disponibile) durante INPUT#, il sistema rimane nello stato solo-ripristino. Ecco perche' NON sono consigliabili le sottoprocedure INPUT# e CHRIN.

La sottoprocedura CHKIN tratta l'"handshaking" sulla x-esima linea seguente lo standard EIA (agosto 1979) per l'interfaccia RS-232 (le linee RTS (Request To Send - richiesta di invio), CTS e DCD (segnale di linea ricevuta) sono implementate sul COMMODORE 64 definito come dispositivo terminale di dati).

INVIO DI DATI AD UN CANALE RS-232

Al momento dell'inoltro di dati occorre tener presente che il buffer di output puo' contenere 255 caratteri, dopodiche' si verifica un trabocco del buffer pieno (overflow). Il sistema attende, nella sottoprocedura CHROUT, che venga attivata la trasmissione, oppure che vengano premuti i tasti RUN/STOP e RESTORE per ripristinare il sistema con una "partenza a caldo".

SINTASSI BASIC:

CMD lfn - Agisce come riportato nelle specifiche del BASIC

PRINT#lfn, ⟨lista variabile⟩

INGRESSI KERNAL:

CHKOUT ($FFC9) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

CHROUT ($FFD2) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

NOTE IMPORTANTI: All'interno del canale di output non avviene alcun ritardo del ritorno carrello: cio' significa che una normale stampante RS-232 non puo' stampare correttamente, a meno che non sia dotata di qualche area di memoria riservata (che richieda un'attesa al COMMODORE 64), o di un buffer interno. L'area di memoria riservata puo' essere facilmente implementata in un programma. Se si implementa un "handshake" CTS (linea X), il buffer del COMMODORE 64 viene riempito, permettendo quindi al dispositivo RS-232 di autorizzare l'area di memoria riservata all'invio di altro output finche' permane la condizione di trasmissione. L'"handshake" della linea X e' una procedura di "handshake" che usa linee multiple per la trasmissione e la ricezione di dati.

L'"handshake" della linea X viene trattato dalla sottoprocedura CHKOUT, che segue lo standard EIA (agosto 1979) per le intarfacce RS-232. Le linee RTS, CTS e DCD vengono implementate sul COMMODORE 64 definendo quest'ultimo come Dispositivo Terminale Dati.

CHIUSURA DI UN CANALE DATI RS-232

La chiusura di un file RS-232 provoca la perdita di tutti i dati (trasmessi, stampati oppure no) contenuti nei buffer al momento dell'esecuzione, l'arresto di tutte le attivita' di trasmissione e ricezione dell'RS-232, e la rimozione di tutti e due i buffer dell'RS-232.

SINTASSI BASIC:

```
CLOSE lfn
```

INGRESSO KERNAL:

CLOSE ($FFC3) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

NOTA: Prima di chiudere un canale, assicurarsi che tutti i dati siano stati trasmessi; cio' si ottiene da BASIC usando la forma:

```
100 SS=ST: IF(SS=0 OR SS=8) THEN 100
110 CLOSE lfn
```

(Locazioni $DD00 – $DD0F del dispositivo 2 6526)

PIN	6526	DESCRIZIONE	EIA	ABBR.	IN/OUT	MODI
C	PB0	Dati ricevuti	(BB)	Sin	IN	1 2
D	PB1	Richiesta di invio	(CA)	RTS	OUT	1(*) 2
E	PB2	Terminale dati disponibile	(CD)	DTR	OUT	1(*) 2
F	PB3	Indicatore anello	(CE)	RI	IN	3
H	PB4	Ricevuto segnale di linea	(CF)	DCD	IN	2
J	PB5	Non asegnato	-	XXX	IN	3
K	PB6	Azzera per invio	(CB)	CTS	IN	2
L	PB7	Insieme dati disponibile	(CC)	DSR	IN	2
B	FLAG2	Dati ricevuti	(BB)	Sin	IN	1 2
M	PA2	Dati trasmessi	(BA)	Sout	OUT	1 2
A	GND	Linea di terra (protezione	(AA)	GND	-	1 2
N	GND	Terra del segnale	(AB)	GND	-	1 2 3

MODI:

1) Interfaccia linea 3 (Sin, Sout, GND)
2) Interfaccia linea X
3) Disponibile solo all'Utente (Non usata / non implementata nel codice)

*) Queste linee sono mantenute alte durante il modo LINEA-3.

Tabella 6.1 - Linee Porta Utente

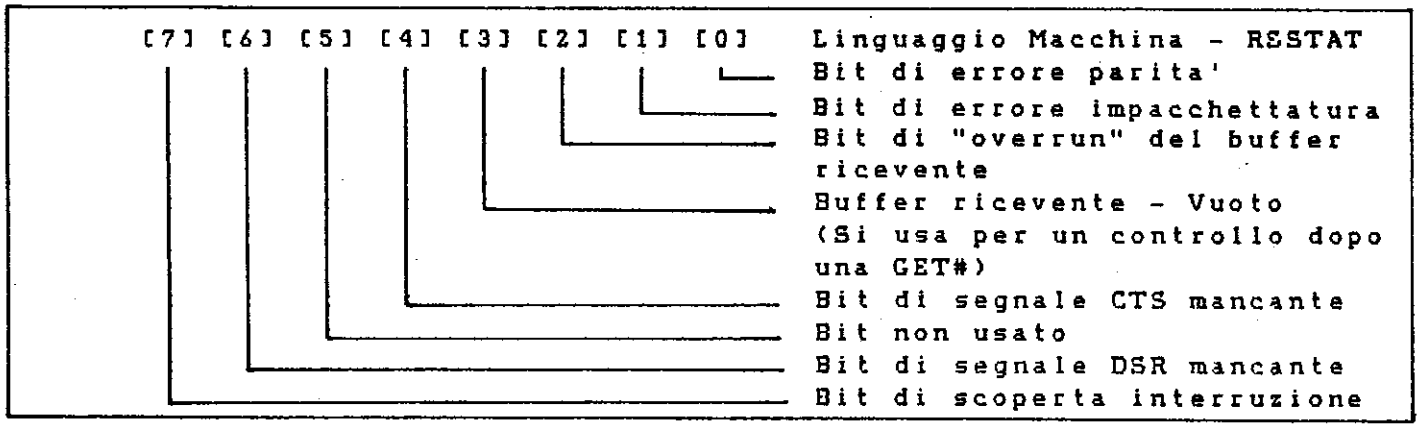


Figura 6.3 - Registro di stato dell'RS-232

NOTE: * Se BIT=0, non si e' incontrato alcun errore.
* Il registro di stato dell'RS-232 puo' essere letto da BASIC usando la variabile ST.
* Se ST e' letta da BASIC oppure usando la sottoprocedura READST del KERNAL, la parola di stato dell'RS-232 viene azzerata all'uscita. Se e' la parola di stato e' necessaria piu' volte, ST deve essere assegnata ad un'altra variabile. Ad esempio:

```
SR=ST: REM ASSIGNS ST TO SR
```

* Lo stato dell'RS-232 viene letto (ed azzerato) solo quando il canale RS-232 e' stato l'ultimo I/O esterno usato.

```
10 REM QUESTO PROGRAMMA INVIA/RICEVE DATI A/DA UN TERMINALE 700
11 REM SILENZIOSO MODIFICATO PER IL PET ASCII
20 REM PREDISPOSIZIONE DEL TERMINALE 700 SILENZIOSO: 300 BAUD, ASCII
21 REM A 7 BIT, PATITA' DEL CARATTERE, FULL DUPLEX
30 REM PREDISPOSIZIONE DEL COMPUTER: COME SOPRA, USANDO L'INTERFACCIA
31 DELLA LINEA 3
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM APERTURA DEL CANALE
110 GET#2,A$:REM ATTIVA IL CANALE RICEVENTE (LANCIA UN CARATTERE
111 REM           NULLO)
200 REM CICLO PRINCIPALE
210 GET B$:REM PRELIEVO DALLA TASTIERA DEL COMPUTER
220 IF B$ <> "" THEN PRINT#2,B$;:REM INVIA AL TERMINALE IL CARATTERE
225 REM                                BATTUTO
230 GET#2,C$:REM PRELEVA UN CARATTERE DAL TERMINALE
240 PRINT B$:C$;:REM STAMPA TUTTI I CARATTERI IN INGRESSO ALLO
241 REM               SCHERMO DEL COMPUTER
250 SR=ST:IF SR=0 OR SR=8 THEN 200:REM CONTROLLA LO STATO, SE E'
260 REM                                  TUTTO A POSTO
260 REM MESSAGGIO DI ERRORE
310 PRINT "ERROR: ";
320 IF SR AND 1 THEN PRINT "PARITA'"
330 IF SR AND 2 THEN PRINT "PACCHETTO"
340 IF SR AND 4 THEN PRINT "BUFFER RICEVENTE PIENO"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF (PEEK(673) AND 1) THEN 360:REM ATTENDE TUTTI I CARATTERI
370 CLOSE 2:END
```

```
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=16:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0 THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$="" OR ST <> 0 THEN 360
330 PRINT " "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34  THEN POKE212,0
350 GOTO 310
360 PRINT CHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$ <> "" THEN PRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THEN CT=0:RV=164-RV
410 GOTO 310
```

PUNTATORI ALLA LOCAZIONE DI BASE DEL BUFFER RICEVENTE/TRASMITTENTE

$00F7-RIBUF Puntatore composto da due byte per la locazione di base del buffer ricevente.

$00F9-ROBUF Puntatore composto da due byte per la locazione di base del buffer trasmittente.

Queste due locazioni sono impostate dalla sottoprocedura OPEN del KERNAL; ciascuna punta ad un differente buffer di 256 byte. Queste locazioni vengono disallocate quando la sottoprocedura CLOSE del KERNAL imposta uno zero nel byte di ordine piu' alto ($00F8 e $00F9 HEX); la loro allocazione e disallocazione e' consentita anche a chi programma in linguaggio macchina per scopi propri, ed e' ottenibile rimuovendo o creando solamente i buffer richiesti. L'allocazione di questi buffer eseguita attraverso il linguaggio macchina richiede una certa attenzione nel corretto posizionamento dei puntatori all'inizio della memoria, specialmente se nello stesso tempo sono presenti in macchina altri programmi BASIC.

LOCAZIONI DELLA MEMORIA DI PAGINA ZERO ED USO DELL'INTERFACCIA DI SISTEMA RS-232

$00A7-INBIT Memorizzazione transiente bit di input del ricevente.
$00A8-BITCI Conteggio bit in ingresso al ricevitore.
$00A9-RINONE Indicatore controllo bit di partenza del ricevente.
$00AA-RIDATA Locazione del byte buffer/assembly del ricevente.
$00AB-RIPRTY Memorizzazione del bit di parita' del ricevente.
$00B4-BITTS Conteggio bit di uscita dal trasmittente.
$00B5-NXTBIT Prossimo bit del trasmittente da inviare.
$00B6-RODATA Locazione byte buffer/disassembly del trasmittente.

Tutte le precedenti locazioni di pagina zero vengono usate localmente e sono state riportate come guida per la comprensione delle sottoprocedure associate. Queste non possono essere utilizzate direttamente da BASIC o da KERNAL per realizzare impieghi simili a quelli dell'RS-232. Per tali impieghi devono essere usate le sottoprocedure dell'RS-232.

LOCAZIONI DELLA MEMORIA DELLE ALTRE PAGINE ED USO DELL'INTERFACCIA DI SISTEMA RS-232

Memoria generale dell'RS-232:

$0293-M51CTR Registro di controllo pseudo 6551 (vd. fig. 6.1).
$0294-M51COR Registro di comando pseudo 6551 (vd. fig. 6.2).
$0295-M51AJB Coppia di byte che segue i registri di controllo e di comando. Queste locazioni contengono la trasmittanza relativa alla partenza del bit di controllo che ha luogo durante l'attivita' dell'interfaccia, usata alternativamente per il calcolo della tasmittanza.
$0297-RSSTAT Registro di stato dell'RS-232 (vd. fig. 6.3).
$0298-BITNUM Numero di bit trasmessi / ricevuti.
$0299-BAUDOF Coppia di byte uguagliata al tempo di una cella di bit (basata sul clock di sistema/trasmittanza).
$029B-RIDBE Indice del byte posto al termine del buffer FIFO del ricevente.
$029C-RIDBS Indice del byte posto all'inizio del buffer FIFO del

ricevente.

$029D-RODBS Indice del byte posto all'inizio del buffer FIFO del trasmittente.

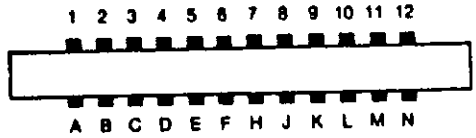
$029E-RODBE Indice del byte posto al termine del buffer FIFO del trasmittente.

$02A1-ENABL Contiene le interruzioni di corrente attive nell'ICR del CIA #2. Quando il bit 4 e' impostato, significa che il sistema sta aspettando il fianco dell'impulso ricevente. Quando il bit 1 e' impostato, il sistema e' in fase di ricezione dati. Quando il bit 0 e' impostato, il sistema e' in fase di tasmissione dati.

PORTA UTENTE

La Porta Utente serve per collegare il COMMODORE 64 al mondo esterno. Usando le linee di collegamento disponibili per questa porta, si puo' collegare il COMMODORE 64 ad una stampante, ad un "type and talk" Votrax, ad un modem, perfino ad un altro computer.

La porta del COMMODORE 64 e' collegata direttamente ad uno dei circuiti CIA 6526. Da programma, il CIA puo' collegarsi con molti altri dispositivi.



DESCRIZIONE DEI PIN DI PORTA

PIN	DESCRIZIONE	NOTE
CIMA		
1	TERRA	
2	+5V	(Max. 100 mA)
3	RIPRISTINO	Connettendo questo pin a terra, il COMMODORE 64 esegue una PARTENZA A FREDDO, risettandosi completamente. I puntatori ad un programma BASIC vengono impostati daccapo, senza pero' che la memoria sia azzerata. Questo pin funziona anche come uscita RIPRISTINO per dispositivi esterni
4	CNT1	Contatore Porta Seriale da CIA#1 (vd. specifiche CIA)
5	SP1	Porta Seriale da CIA#1 (vd. specifiche CIA 6526)
6	CNT2	Contatore Porta Seriale da CIA#2 (vd. specifiche CIA)
7	SP2	Porta Seriale da CIA#2 (vd. specifiche CIA 6526)
8	PC2	Linea di "handshacking" da CIA#2 (vd. specifiche CIA)
9	ATN SERIALE	Pin connesso alla linea ATN del bus seriale
10	9VAC+fase	Connessi direttamente al trasformatore del
11	9VAC-fase	COMMODORE 64 (Max. 50 mA)
12	GND	
FONDO		
A	GND	Il COMMODORE 64 permette di controllare la PORTA B del circuito CIA#1, mettendo a disposizione 8 linee di input/output e 2 linee di handshacking con un dispositivo esterno. Le linee di I/O della PORTA B sono controllate da 2 locazioni, di cui una, la PORTA stessa, sita in 56577 ($DD01 HEX). Ovviamente, la lettura avviene tramite PEEK e la scrittura tramite POKE. Ogni linea di I/O puo' essere impostata sia come INPUT che come OUTPUT impostando in maniera adeguata il REGISTRO DIREZIONE DATI.
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
G	PB4	
H	PB5	
J	PB6	
K	PB7	
L	PA2	
N	GND	

Il REGISTRO DIREZIONE DATI e' allocato in 56579 ($DD03 HEX). Ciascuna delle otto linee della PORTA ha un corrispondente BIT nel REGISTRO DIREZIONE DATI a 8 bit (DDR), che controlla se la linea deve essere di input o di output. Se un bit del DDR si trova impostato a UNO, allora la corrispondente linea della PORTA e' di OUTPUT, mentre se si trova a ZERO la linea e' di INPUT. Ad esempio, se il bit 3 di DDR e' impostato a 1, allora la linea 3 della PORTA e' di OUTPUT. Un ulteriore esempio puo' essere il seguente. Supponiamo che DDR sia impostato nel modo seguente:

BIT # :	7	6	5	4	3	2	1	0
VALORE:	0	0	1	1	1	0	0	0

Si puo' vedere che le linee 5,4 e 3 sono di output poiche' i relativi bit sono a 1, mentre le rimanenti linee sono di input, dato che i relativi bit sono a 0.

Per eseguire una PEEK o una POKE sulla Porta UTENTE, e' necessario usare sia DDR che la PORTA stessa. Si ricordi che le istruzioni PEEK e POKE vogliono un numero compreso fra 0 e 255. Affinche' i numeri riportati nell'esempio possano essere usati, devono prima subire la trasformazione in decimale: tale valore e':

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

Da notare che il numero di bit (BIT #) per DDR e' lo stesso numero ottenuto dall'elevamento di 2 a potenza, in modo tale da ritornare il valore del bit:

(16 = 2↑4=2×2×2×2, 8 = 2↑3=2×2×2)

Le altre due linee, FLAG1 e PA2, sono diverse dal resto della PORTA UTENTE: queste due linee sono riservate esclusivamente all'"HANDSHACKING", e sono programmate diversamente dalla porta B.

L'"handshacking" e' necessario quando due dispositivi comunicano tra loro. Dato che uno dei dispositivi puo' eleborare ad una velocita' diversa da quella usata dall'altro, e' necessario far sapere a ciascun dispositivo cio' che sta facendo l'altro. L'"handshacking" e' necessario anche quando i due dispositivi lavorano alla stessa velocita', per permettere all'altro di sapere quando si deve inviare il dato, e se il dato e' stato ricevuto. La linea FLAG1 ha delle caratteristiche particolari che la rendono adatta per l'"handshacking".

FLAG1 e' un ingresso sensibile al fianco negativo di un impulso, che puo' essere usato come input ad una interruzione di carattere generale. Ogni transizione negativa sulla linea FLAG imposta il bit di interruzione di FLAG. Se viene abilitata l'interruzione FLAG, questa causa una RICHIESTA DI INTERRUZIONE. Se il bit di FLAG non e' abilitato, puo' essere interrogato dal registro interruzione sotto il controllo del programma.

PA2 e' il bit 2 della PORTA A del CIA; viene controllato come ogni altro bit della porta. Quest'ultima e' allocata nella posizione 56576 ($DD00 HEX), mentre il registro direzione dati e' allocato in 56578 ($DD02 HEX).

PER ULTERIORI INFORMAZIONI SUL 6526 SI VEDANO LE SPECIFICHE DI CIRCUITO RIPORTATE IN APPENDICE M.

BUS SERIALE

Il bus seriale e' una combinazione a "daisy chain" designata per permettere al COMMODORE 64 di comunicare con dispositivi quali il DISK DRIVE VIC-1541 e la STAMPANTE GRAFICA VIC-1525. Il vantaggio apportato dal bus seriale sta nel fatto che quest'ultimo puo' essere collegato con la porta meglio di un altro dispositivo. Al bus seriale possono essere collegati contemporaneamente non piu' di 5 dispositivi.

Il bus seriale consente tre tipi di operazione: CONTROLLO, TRASMISSIONE E RICEZIONE. Un dispositivo CONTROLLORE e' preposto alle funzioni del bus seriale; il TRASMETTITORE e' preposto all'invio di dati sul bus seriale; il RICEVITORE e' preposto alla ricezione dei dati dal bus seriale.

Il controllore del bus seriale e' il COMMODORE 64: questo si comporta anche da TRASMETTITORE (ad esempio, quando invia dati alla stampante) e da RICEVITORE (ad esempio, quando carica un programma da disco). Anche altri dispositivi possono fungere da RICEVITORE (stampante), da TRASMETTITORE (disco) o da entrambi (disco), ma solamente il COMMODORE 64 puo' comportarsi da CONTROLLORE.

Tutti i dispositivi collegati al bus seriale ricevono tutti i dati trasmessi sul bus. Per permettere al COMMODORE 64 di distribuire i dati alle proprie destinazioni, ciascun dispositivo ha un INDIRIZZO di bus, tramite il quale il COMMODORE 64 puo' controllare l'accesso al bus; gli indirizzi sul bus seriale coprono un intervallo che va da 4 a 31.

Il COMMODORE 64 puo' COMANDARE ad un particolare dispositivo di TRASMETTERE o RICEVERE. Quando ad un dispositivo viene ordinato di TRASMETTERE, questo comincia ad immettere dati sul bus seriale; quando invece gli viene ordinato di RICEVERE, il dispositivo indirizzato si trova pronto per prelevare i dati (dal COMMODORE 64 o da un altro dispositivo che si trova sul bus). Nello stesso istante, solamente un dispositivo puo' INVIARE dati sul bus, altrimenti si potrebbero verificare collisioni fra i dati, gettando il sistema nella confusione. Tuttavia, qualsiasi numero di disositivo puo' RICEVERE nello stesso istante da un TRASMETTITORE.

INDIRIZZI DEL BUS SERIALE COMUNE

NUMERO	DISPOSITIVO
4 o 5	Stampante grafica VIC-1525
8	Disk Drive VIC-1541

Sono consentiti anche altri numeri di dispositivi. Ciascun dispositivo ha il proprio indirizzo. Alcuni dispositivi (come la stampante del COMMODORE 64) forniscono all'Utente la possibilita' di scegliere fra due indirizzi.

L'INDIRIZZO SECONDARIO consente al COMMODORE 64 di trasmettere l'informazione ad un dispositivo. Ad esempio, per aprire un collegamento sul bus della stampante e per far si' che essa stampi nel formato MAIUSCOLO/MINUSCOLO, si puo' usare la seguente istruzione:

```
OPEN 1,4,7
```

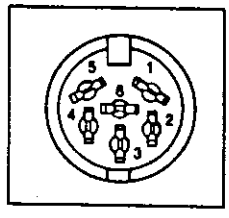
dove:

1 e' il numero del file logico (il numero da usare con PRINT#),
4 e' l'INDIRIZZO della stampante, e
7 e' l'INDIRIZZO SECONDARIO che dice alla stampante di assumere il modo di stampa MAIUSCOLO/MINUSCOLO.

Per le operazioni sul bus seriale si hanno a disposizione sei linee - tre di input e tre di output. Le tre linee di input trasferiscono dati e segnali di controllo e del tempo al COMMODORE 64, le tre di output inviano dati e segnali di controllo e del tempo dal COMMODORE 64 ai dispositivi esterni posti sul bus seriale.

SPINOTTI DEL BUS SERIALE

PIN	DESCRIZIONE
1	Ingresso SRQ seriale
2	GND
3	Ingresso/Uscita ATN seriale
4	Ingresso/Uscita CLK seriale
5	Ingresso/Uscita dati seriali
6	Nessuna connessione



INGRESSO SRQ SERIALE (INGRESSO RICHIESTA DI SERVIZIO SERIALE)

Qualunque dispositivo posto sul bus seriale puo' abbassare (LOW) questo segnale quando richieda l'attenzione del COMMODORE 64; in questo caso, quest'ultimo si prende cura del dispositivo che ha causato tale abbassamento (vd. fig. 6-4).

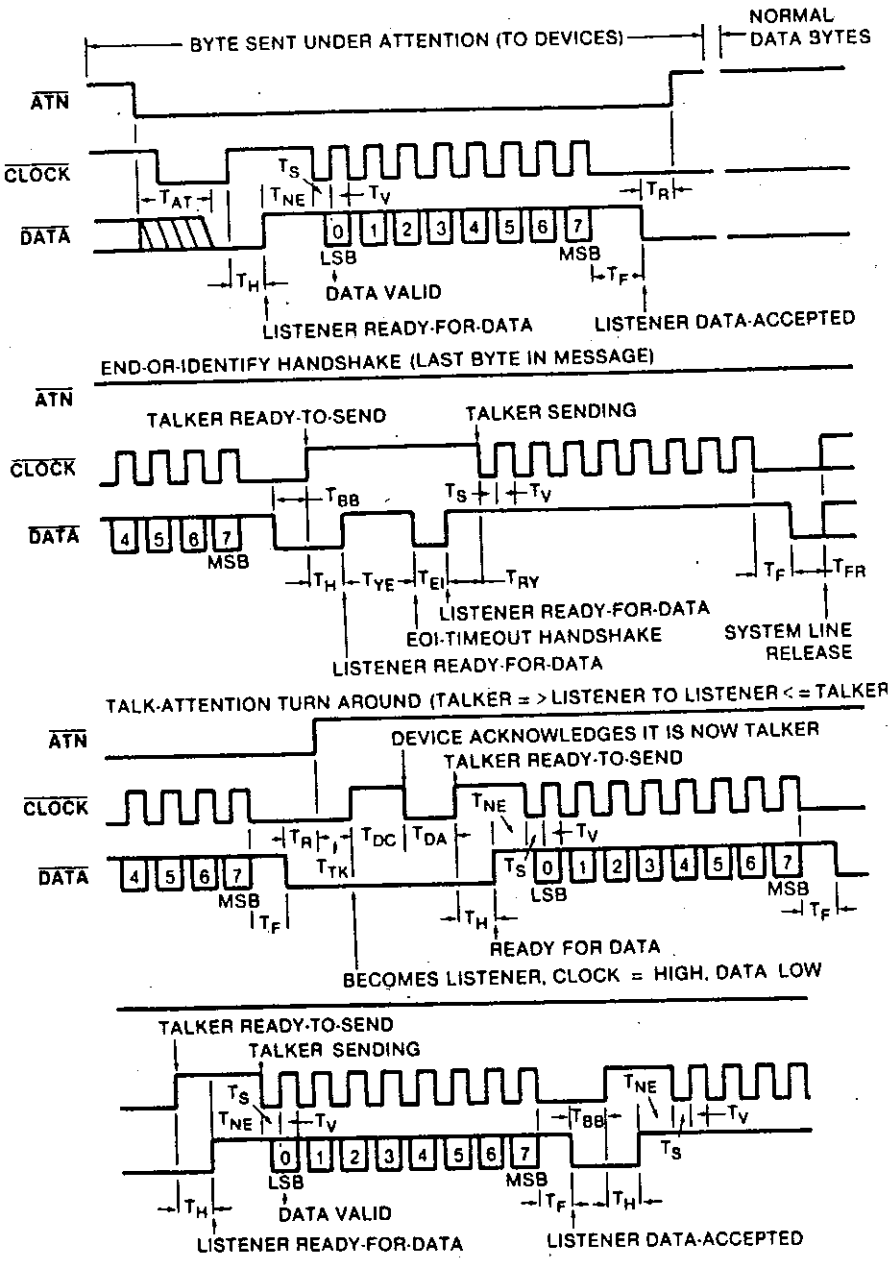


Figura 6.4 - Tempi di risposta del bus seriale

TEMPI DI RISPOSTA DEL BUS SERIALE

DESCRIZIONE	SIMBOLO	MIN	MED	MAX
Risposta ATN (richiesta) (1)	Tat	-	-	1000 ns
Predisposizione Ricevente	Th	0	-	**
Risposta non-EOI a RFD (2)	Tne	-	40 ns	200 ns
Predisposizione Trasmittente	Ts	20 ns	70 ns	-
Convalida dati Handshake	Tv	20 ns	20 ns	-
pacchetto (3)	Tf	0	20 ns	1000 ns
Pacchetto di ATN da rilasciare	Tr	20 ns	-	-
Intervallo fra due byte	Tbb	-	-	-
Risposta di EOI	Tye	200 ns	250 ns	-
Trattenimento risposta di EOI	Tei	60 ns	-	-
Limite di risposta del trasmittente	Try	0	30 ns	60 ns
Riconoscimento del byte	Tpr	20 ns	30 ns	-

Note:

1. Se si supera il massimo tempo consentito, errore di dispositivo non presente.
2. Se si supera il massimo tempo consentito, richiesta di risposta EOI.
3. Se si supera il massimo tempo consentito, errore di pacchetto.
4. Affinche' un dispositivo esterno sia assunto come trasmittente, il valore minimo di Tv e Tpr deve essere 60 ns.

INGRESSO/USCITA ATN SERIALE (INGRESSO/USCITA ATTENZIONE SERIALE)

Il COMMODORE 64 usa questo segnale per far partire una sequenza di comandi destinati ad un dispositivo sul bus seriale. Dal momento in cui il COMMODORE 64 abbassa (LOW) il segnale, tutti i dispositivi sul bus seriale si tengono pronti a ricevere l'indirizzo trasmesso dal COMMODORE 64. Il dispositivo indirizzato deve rispondere immediatamente, altrimenti il COMMODORE 64 assume che il dispositivo chiamato non e' presente sul bus, riportando un errore nella PAROLA DI STATO (vd. fig. 6.4).

INGRESSO/USCITA CLK SERIALE (INGRESSO/USCITA CLOCK SERIALE)

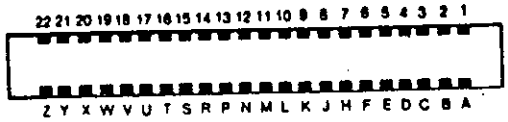
Questo segnale viene usato per la temporizzazione dei dati inviati sul bus seriale (vd. fig. 6.4).

INGRESSO/USCITA DATI SERIALI

I dati presenti sul bus seriale vengono trasmessi bit per bit su questa linea (vd. fig. 6.4).

PORTA ESPANSIONE

Il connettore di espansione e' una presa femmina a 44 pin (22/22) posta sul retro del COMMODORE 64, sull'estrema destra osservando la macchina dal davanti. Il collegameno con questo connettore e' possibile usando un connettore maschio a 44 pin (22/22).
Questa porta viene usata per espansioni di sistema del COMMODORE 64 richiedenti l'accesso al bus indirizzi o al bus dati del computer. L'uso del bus espansione richiede una certa prudenza, in quanto un malfunzionamento del dispositivo allacciato a questa porta puo' causare danni al COMMODORE 64.
L'organizzazione del bus espansione e' la seguente:



I segnali disponibili sul connettore sono riportati nella tabella seguente.

NOME	PIN	DESCRIZIONE
GND	1	Terra del sistema
+5 VDC	2	(Assorbimento massimo sopportabile dai dispositivi
+5 VDC	3	PORTA UTENTE e CARTUCCIA: 450 mA)
IRQ	4	Linea di RIchiesta Interruzione per 6502
R/W	5	Lettura/Scrittura
DOT CLOCK	6	Timer punti video (8.18 MHz)
I/O1	7	Blocco 1 di I/O ($DE00-$DEFF) non bufferizzato @
GAME	8	Input ls ttl
EXROM	9	Input ls ttl
I/O2	10	Blocco 2 di I/O ($DE00-$DEFF) bufferizzato @ output ls ttl
ROML	11	Blocco di 8K RAM/ROM decodificato ($8000) @ output ls ttl
BA	12	Segnale di bus disponibile dal circuito VIC-II non bufferizzato - carico massimo 1 ls
DMA	13	Linea di richiesta di accesso diretto alla memoria input ls ttl
D7	14	Bit 7 del bus dati
D6	15	Bit 6 del bus dati
D5	16	Bit 5 del bus dati
D4	17	Bit 4 del bus dati
D3	18	Bit 3 del bus dati
D2	19	Bit 2 del bus dati
D1	20	Bit 1 del bus dati
D0	21	bit 0 del bus dati
GND	22	Terra del sistema
GND	A	
ROMH	B	Blocco bufferizzato di 8K di RAM/ROM decodificati @ ($E000)
RESET	C	Pin di RESET del 6502 out ttl bufferizzato/in non bufferizzato
NMI	D	Interruzione non mascherabile del 6502 out ttl bufferizzato, in non bufferizzato
O2	E	Timer di sistema per Fase 2
A15	F	Bit 15 del bus indirizzi
A14	H	Bit 14 del bus indirizzi
A13	J	Bit 13 del bus indirizzi
A12	K	Bit 12 del bus indirizzi
A11	L	Bit 11 del bus indirizzi
A10	M	Bit 10 del bus indirizzi
A9	N	Bit 9 del bus indirizzi
A8	P	Bit 8 del bus indirizzi
A7	R	Bit 7 del bus indirizzi
A6	S	Bit 6 del bus indirizzi
A5	T	Bit 5 del bus indirizzi
A4	U	Bit 4 del bus indirizzi
A3	V	Bit 3 del bus indirizzi
A2	W	Bit 2 del bus indirizzi
A1	X	Bit 1 del bus indirizzi
A0	Y	Bit 0 del bus indirizzi
GND	Z	Terra del sistema

I nomi soprasegnati sono attivi a segnale basso

La seguente e' una descrizione di alcune linee importanti della porta espansione:

Tutte le linee sono connesse alla terra di sistema.

DOT CLOCK - Clock a punti del video a 8.18 MHz. Tutte le temporizzazioni del sistema derivano da questo orologio.

BA (Bus Available - bus disponibile) - segnale originato dal circuito VIC-II che si mantiene basso per tre cicli prima che il VIC-II acceda ai canali di trasferimento delle informazioni tra registri, rimanendo basso fino alla fine della ricerca dell'informazione da visualizzare svolta dal VIC-II.

DMA (Direct Memory Access - Accesso Diretto alla Memoria). - Quando si abbassaquesta linea, il bus indirizzi, il bus dati e la linea di lettura/scrittura del circuito processore 6510 entrano nello stato ad alta impedenza, permettendo ad un processore esterno di prendere il controllo dei canali di trasferimento delle informazioni tra registri ("busses") del sistema. Questa linea puo' essere abbassata solamente quando il clock O2 e' basso. Inoltre, poiche' il circuito VIC-II continua a visualizzare DMA, il dispositivo esterno deve essere conforme alla temporizzazione del VIC-II (vedere il diagramma di temporizzazione del VIC-II). Sul COMMODORE 64 questa linea viene tenuta alta.

CARTUCCIA CON MICROPROCESSORE Z-80

La lettura di questo libro e l'uso del COMMODORE 64 mettono in luce solamente una parte della versatilita' di questo computer. Le capacita' della macchina possono essere aumentate in maniera considerevole con l'aggiunta di strumenti periferici, quali registratori Datassette[TM], unita' disco, stampanti e modem. Tutte queste unita' possono essere aggiunte al COMMODORE 64 tramite le varie porte e prese poste sul retro della macchina. Cio' che rende valide le periferiche del COMMODORE 64 e' che queste sono "intelligenti": durante il loro funzionamento, infatti, non sfruttano lo spazio della Memoria ad Accesso Random, lasciando quindi libero l'Utente di usare tutti i 64K byte del COMMODORE 64.

Un altro vantaggio offerto dal COMMODORE 64 e' che i programmi scritti per questa macchina sono adattabili anche ad altri computer Commodore, acquistabili in futuro: parte di questo merito va alle qualita' del Sistema Operativo (OS).

Quest'ultimo, pero', non puo' rendere i programmi compatibili ad un computer prodotto da un'altra Societa'. Data comunque la semplicita' d'uso del COMMODORE 64, questa necessita' non si presentera' mai!...Ma per quegli Utenti occasionali che vogliono trarre vantaggio dal software non disponibile nello standard del COMMODORE 64, e' stata creata dalla Commodore una cartuccia CP/M [R].

Il CP/M [R] non e' un Sistema Operativo "dipendente dal computer". Al contrario, si appoggia ad una parte dello spazio di memoria normalmente disponibile per la programmazione, sfruttando questo spazio per far girare il proprio sistema operativo. Questo modo di procedere presenta vantaggi e svantaggi. Gli svantaggi sono che i programmi scritti per funzionare con il CP/M [R] devono essere piu'

corti di quelli scritti per funzionare con il sistema operativo interno del COMMODORE 64; inoltre, NON si puo' sfruttare appieno la potenza di editing dello schermo del COMMODORE 64. I vantaggi risiedono nella possibilita' di usare una grande quantita' di software creati appositamente per il CP/M [R] e per il microprocessore Z-80; inoltre, i programmi scritti usando il sistema operativo CP/M [R] possono essere trasferiti e fatti girare su qualsiasi altro computer dotato di CP/M [R] e di una scheda Z-80.

Ad esempio, la maggior parte dei computer che usano un microprocessore Z-80 richiedono che sia l'Utente stesso ad inserire nel computer la scheda Z-80. Questo metodo richiede molta attenzione a non danneggiare i delicati circuiti che attraversano il resto del computer. La cartuccia CP/M [R] della Commodore elimina questi inconvenienti, in quanto la cartuccia Z-80 viene installata sul retro del COMMODORE 64, velocemente e facilmente, senza fili confusi che in seguito potrebbero causare problemi.

COME USARE IL CP/M [R] COMMODORE

La cartuccia Z-80 della Commodore fa si' che i programmi predisposti per un microprocessore Z-80 siano in grado di funzionare sul COMMODORE 64. La cartuccia e' provvista di un dischetto contenente il sistema operativo CP/M [R] della Commodore.

ELABORAZIONE SOTTO CP/M [R] COMMODORE

Per far girare il CP/M [R]:

1) Caricare (LOAD) il programma CP/M [R] dall'unita' disco.
2) Battere RUN.
3) Premere il tasto RETURN .

A questo punto, i 64K byte di RAM del COMMODORE 64 sono accessibili al processore centrale 6510, OPPURE sono disponibili i 48K byte di RAM per il processore centrale Z-80. L'uso di questi due processori e' alternabile, ma non e' consentito il loro uso contemporaneo all'interno di un singolo programma. Cio' e' reso possibile dal sofisticato meccanismo di temporizzazione del COMMODORE 64.

La trasformazione degli indirizzi di memoria resa necessaria dall'uso della cartuccia Z-80 e' riportata nella seguente tabella. Si puo' notare che, addizionando 4096 byte alle locazioni di memoria usate in CPM [R] $1000 (HEX), si ritrovano gli indirizzi di memoria del normale sistema operativo del COMMODORE 64. La corrispondenza degli indirizzi di memoria tra Z-80 e 6510 e' la seguente:

INDIRIZZI Z-80		INDIRIZZI 6510	
DECIMALE	HEX	DEC.	HEX
0000-4095	0000-0FFF	4096-8191	1000-1FFF
4096-8191	1000-1FFF	8192-12287	2000-2FFF
8192-12287	2000-2FFF	12288-16383	3000-3FFF
12288-16383	3000-3FFF	16384-20479	4000-4FFF
16384-20479	4000-4FFF	20480-24575	5000-5FFF
20480-24575	5000-5FFF	24576-28671	6000-6FFF
24576-28671	6000-6FFF	28672-32767	7000-7FFF
28672-32767	7000-7FFF	32768-36863	8000-8FFF
32768-36863	8000-8FFF	36864-40959	9000-9FFF
36864-40959	9000-9FFF	40960-45055	A000-AFFF
40960-45055	A000-AFFF	45056-49151	B000-BFFF
45056-49151	B000-BFFF	49152-53247	C000-CFFF
49152-53247	C000-CFFF	53248-57343	D000-DFFF
53248-57343	D000-DFFF	57344-61439	E000-EFFF
57344-61439	E000-EFFF	61440-65535	F000-FFFF
61440-65535	F000-FFFF	0000-4095	0000-0FFF

Per ATTIVARE la cartuccia Z-80 e DISATTIVARE il circuito 6510, digitare il seguente programma:

```
10 REM QUESTO PROGRAMMA DEVE ESSERE USATO CON LA CARTUCCIA Z-80.
20 REM INNANZITUTTO MEMORIZZA I DATI DELLA Z-80 NELLA LOCAZIONE $1000
21 REM (Z80=$0000)
30 REM POI DISATTIVA LE IRQ DEL 6510 ED ATTIVA LA CARTUCCIA Z-80.
40 REM PER RIABILITARE IL SISTEMA 6510, OCCORRE DISABILITARE LA
41 REM CARTUCCIA Z-80
50 REM
100 REM MEMORIZZA I DATI Z-80
110 READ B: REM PRELEVA LA QUANTITA' DI CODICE Z80 CHE DEVE ESSERE
111 REM          RIMOSSO
120 FOR I=4096 TO 4096+B-1: REM RIMUOVE IL CODICE
130 READ A:POKE I,A
140 NEXT I
200 REM LANCIA IL CODICE Z-80
210 POKE 56333,127:REM DISATTIVA LE IRQ DEL 6510
220 POKE 56832,00 :REM ATTIVA LA CARTUCCIA Z-80
230 POKE 56333,129:REM ATTIVA LE IRQ DEL 6510 QUANDO SI RILASCIA
231 REM                LA Z-80
240 END
1000 REM SEZIONE DATI DEL CODICE DEL LINGUAGGIO MACCHINA Z-80
1010 DATA 18:REM MISURA DEI DATI DA PASSARE
1100 REM CODICE DI ATTIVAZIONE Z-80
1110 DATA 00,00,00:REM LA CARTUCCIA Z-80 RICHIEDE DI ATTIVARE IL
1111 REM              TEMPO A $0000
1200 REM SEZIONE DATI Z-80 DA ELABORARE
1210 DATA 33,02,245:REM LD  HN, NN (LOCAZIONI DELLO SCHERMO)
1220 DATA 52:REM INCREMENTA LA LOCAZIONE HL
1300 REM DATI Z-80 DI AUTO-ESCLUSIONE
1310 DATA 62,01:REM LD  A,N
1320 DATA 50,00,206:REM LD (NN),A : LOCAZIONE DI I/O
1330 DATA 00,00,00:REM NOP:NOP:NOP
1340 DATA 195,00,00:REM JMP $0000
```

Per maggiori dettagli sul CP/M [R] Commodore ed il microprocessore Z-80 si veda la Guida di Riferimento della cartuccia e della Z-80.