Vittorio Paola · Gaetano Romano

IBM P.C. commodore 64

libreria dario flaccovio editrice

Vittorio Paola - Gaetano Romano

TECNICHE AVANZATE DI PROTEZIONE E SPROTEZIONE SOFTWARE

libreria dario flaccovio editrice

1ª Ristampa 1988

SOFTWARE

I programmi descritti nel volume sono disponibili su disco utilizzabile su:

☐ IBM p.c. e compatibili (cod. 6008) L. 80.000+IVA

☐ Commodore 64 (cod. 8010) L. 40.000+IVA

Chi lo desidera potrà farne ordinazione direttamente alla

EDITRICE DARIO FLACCOVIO
Via Ausonia, 70/74
90144 PALERMO - Tel. 091-562068

Le spedizioni vengono effettuate in Contrassegno Postale

© 1986 by Libreria Dario Flaccovio Editrice - Via Ausonia 70/74, Palermo - Tel. (091) 562068

Fotocomposizione e stampa: Priulla - Palermo 1988

PREFAZIONE

Da utilizzatori di software, ci siamo sempre chiesti cosa potesse giustificare l' elevato prezzo commerciale di certi pacchetti applicativi, o peggio, di taluni giochi; forse lo sforzo del programmatore, il quale ha dovuto faticare parecchi mesi per stendere in algoritmo la sua idea, o forse interessi commerciali che inevitabilmente fanno lievitare il prezzo dei programmi in funzione della richiesta del mercato; ma dopo una attenta analisi del problema ci siamo accorti che le softwarehouse tengono alti i prezzi per arginare le perdite derivanti dal fenomeno della "pirateria".
Il "pirata", infatti, non si contenta di copiare il programma, alle volte neanche regolarmente aquistato, per usi personali, ma trova divertente distribuirne, o peggio, venderne piu' copie possibile.
Quale arma resta alle softwarehouse per tutelare i propri programmi, le proprie vendite, ed indirettamente, gli interessi del programmatore che dopo mesi di lavoro attende la giusta ricompensa ?
Immediata viene alla mente la soluzione del problema : proteggere il programma dalle copie clandestine . Questo vuol dire mettere al lavoro schiere di programmatori, per escogitare metodi di protezione validi, ma contemporaneamente si mettono in moto migliaia di utenti che provvedono a vanificare le protezioni rendendo perfettamente copiabili i programmi.
Si innesca, cosi' una reazione a catena che vede da una parte i "buoni" , intenti a mettere a punto sistemi anticopia sempre piu' sofisticati, e dall'

altra i "cattivi" i quali lavorano per annullare il lavoro dei "buoni", e la storia si ripete.
La legge nulla puo' in questo caso non essendo, ancora regolata la materia, e chi paga le conseguenze di questo andazzo e', come al solito, il povero utente che si reca nei negozzi specializzati per acquistare un programma e paga, oltre al costo del programma stesso, il lavoro del programmatore che ha protetto il programma dalle copie.
In definitiva, chi il pirata non lo ha mai fatto, continua a non farlo, mentre chi lo ha sempre fatto continua nella sua opera traendo, anzi, motivo di soddisfazione personale nello sproteggere un programma considerato incopiabile.
Diciamolo francamente, non esistono protezioni efficaci al cento per cento, qualunque sistema anticopia, nelle mani di uno sprotettore esperto, crolla miseramente e' solo questione di tempo.
Perche', allora, si deve precludere all' utente, normale, la possibilita', peraltro legittima, di fare delle copie di riserva del proprio software, quando attorno a lui molti lo fanno abusivamente ?
Da queste considerazioni nasce il libro, non e' un delitto informare il lettore sui metodi di protezione e come fare ad eluderli, in quanto questi stessi metodi, ed altri, sono gia' noti a chi, il pirata lo fa per mestiere; mettiamo dunque i "buoni" ed i "cattivi" sullo stesso piano, almeno per la possibilita' di copiare, e lasciamo al buon senso di entrambi la decisione sull' uso delle copie prodotte.
Pensiamo che la pirateria e nata con il software e morira' con esso, cioe' mai. Cosa si puo' fare, allora ? Proprio niente, prenderla con filosofia, oppure, come stanno facendo molte case

di software americane, approfittare della diffusione piratesca dei loro prodotti, per farsi della pubblicita' gratuita, o quasi.
Prima di augurare a tutti buon lavoro volevamo precisare che per motivi tecnici (i listati erano troppo lunghi e difficili da copiare), non é stato possibile inserire tutti i programmi nel libro.
Per compensare questa mancanza, i dischi con tutti i programmi necessari per le operazioni descritte, sono disponibili sia per Commodore 64 che per PC IBM e compatibili ad un prezzo modico.

GLI AUTORI

GENERALITA'

Generalmente il software commerciale é sempre protetto contro le copie, in altre parole, un programma regolarmente acquistato su supporto magnetico, può essere normalmente caricato nel calcolatore, eseguito, e poi cancellato, ma mai copiato.
Se si tenta di duplicare uno di questi programmi, anche se alle volte sembra una operazione possibile, si ottengono i risultati più strani e le copie saranno sicuramente delle copie non funzionanti.
I metodi per ottenere delle copie funzionanti, sono molteplici e di svariata natura e soprattutto legati al tipo di calcolatore adoperato; noi da ora in poi faremo riferimento ad un Commodore 64.
Prima di addentrarci nei vari metodi sia per disco che per cassetta, vale la pena introdurre qualche premessa:
Per potere accedere ad un programma 'chiuso' bisogna avere delle nozioni sui seguenti argomenti:
- Elementi di linguaggio macchina
- Sistemi di numerazione decimale, binaria, esadecimale ecc.
- Suddivisione standard della memoria del calcolatore
- Studio del sistema operativo

Questi argomenti saranno oggetto di discussione nei capitoli successivi, e precederanno quelli che sono i metodi di sprotezione veri e propri.
Si raccomanda ai meno esperti un attento studio di questi capitoli fondamentali per l' acquisizione delle tecniche di copiatura.
Naturalmente, il lavoro diventerà più semplice via via che si saranno risolti 'casi' sempre più difficili.
Resta inteso che lme copie prodotte con le tecniche suggerite da questo libro serviranno, solo e soltanto, a creare delle copie di lavoro da destinarsi ad uso

personale.
La mentalità dello sprotettore, una figura ben precisa nel panorama informatico del nostro tempo, deve essere rivolta a ciò che pensava il programmatore nel momento in cui stava per chiudere il nostro programma; bisogna esaminare come il programma si difende dai nostri assalti e cercare di individuarne le cause.
Discorsi a parte é molto utile esaminare almeno una volta il programma da copiare, e vedere esattamente come si comporta; é buona norma non fidarsi della propria memoria e tenere accanto al tavolo di lavoro, un notes ed una penna, dove andranno annotate varie cose: indirizzi specifici del programma, locazione di partenza e di fine ecc. ecc..
Mi sembra di avere detto tutto e che possiamo cominciare lo studio vero e proprio; a tutti BUON LAVORO e se avete dei problemi l' autore é a vostra completa disposizione, naturalmente tramite la casa editrice.

SISTEMI DI NUMERAZIONE

I simboli numerici, o più semplicemente i numeri, furono inventati per facilitare le operazioni di conteggio.
I diversi sistemi di numerazione differiscono nel tipo di simboli usati e nelle combinazioni degli stessi.
I sistemi che sono realmente utili per lavorare su di un calcolatore sono tre:
- Decimale (in base dieci)
- Binario (in base due)
- Esadecimale (in base sedici)

Del sistema decimale é inutile parlarne in quanto é il sistema che adoperiamo quotidianamente e che quindi conosciamo molto bene.
Vale, soltanto, la pena dare uno sguardo su quello che é un modo abbreviato di scrivere numeri decimali molto grandi, la notazione espansa o esponenziale.
Il valore espanso serve ad evidenziare il valore rappresentato da ogni cifra condensando in pochi simboli numeri molto grandi.
Vediamo meglio di chiarire il concetto con un esempio:
il numero 5867 é perfettamente equivalente alla seguente espressione:

5 migliaia 8 centinaia 6 decine 7 unità

5*1000 + 8*100 + 6*10 + 7

oppure, che é lo stesso, si può scrivere:

5*10(E)3 + 8*10(E)2 + 6*10(E)1 + 7*10(E)0

Il simbolo (E) evidenzia l' operazione di elevazione a potenza.
I numeri 1000,100,10,1 sono tutti potenze di dieci e si ottengono moltiplicando 10 per se stesso un certo numero di volte.
Una potenza con esponente 0 é del tutto particolare; indipendentemente dalla base il suo valore é sempre uno, così ad esempio 10(E)0=1 oppure 156(E)0=1 ed anche 23(E)0=1 ecc. ecc.
Anche le frazioni ed i numeri razionali si possono esprimere con la notazione espansa assegnando ad ogni cifra a destra della virgola decimale, una potenza negativa della base 10 a partire da 10(E)-1.
Così avremo per esempio:
0.00005 che diventa 5*10(E)-5
0.0087 che diventa 8.7*10(E)-3
0.000000012 che diventa 1.2*10(E)-8
e così via. Per comprendere meglio questo meccanismo può essere utile consultare la tabella quì di seguito che rappresenta alcune potenze di dieci:

10(E)0	1	10(E)-1	0.1
10(E)1	10	10(E)-2	0.01
10(E)2	100	10(E)-3	0.001
10(E)3	1.000	10(E)-4	0.0001
10(E)4	10.000	10(E)-5	0.00001
10(E)5	100.000	10(E)-6	0.000001
10(E)6	1.000.000	10(E)-7	0.0000001
10(E)7	10.000.000	10(E)-8	0.00000001
10(E)8	100.000.000	10(E)-9	0.000000001
10(E)9	1.000.000.000	10(E)-10	0.0000000001
10(E)10	10.000.000.000		

Questo tipo di notazione in base 10, sembra a prima vista complessa, in realtà il meccanismo é di facile apprendimento e consente un reale risparmio di cifre nel rappresentare un numero molto grande o anche molto piccolo.

NUMERAZIONE IN BASE 16

Nella memoria centrale di qualunque elaboratore i dati ed il programma sono immagazzinati come sequenze di numeri binari (che vedremo in dettaglio nel prossimo capitolo) cioe numeri formati dalla combinazione di due sole cifre lo 0 e l' 1.
Con queste due cifre é possibile rappresentare tutti i numeri ed anche tutte le lettere dell' alfabeto ed altri caratteri ancora.
Ogni cifra di un codice binario prende il nome di bit; otto bit consecutivi formano un byte.
Durante la scrittura, la messa a punto di un programma o il tentativo di riprodurne delle copie, é molto importante prendere visione dei dati e dei codici memorizzati nelle varie locazioni di memoria, ma é sicuramente molto pesante visionare lunghe sequenze di uno e di zero ed é anche facile sbagliare l' interpretazione dei codici.
Per questi motivi, normalmente di adopera una rappresentazione dei dati più sintetica, quella esadecimale cioé in base sedici.
Il metodo consiste nel rappresentare quattro cifre binarie con un unico simbolo; risulta subito evidente il perché della scelta base sedici, servono, infatti sedici simboli diversi in quanto, le possibili combinazioni di quattro bit sono sedici.

La numerazione in base sedici usa sedici simboli diversi come cifre; i primi dieci sono uguali alle cifre del sistema decimale: 0,1,2.....9; le restanti 6 cifre sono le prime lettere dell' alfabeto e cioé A,B,C,D,E,F.
Ogni posizione entro una sequenza di cifre esadecimali é collegata con la relativa potenza di sedici.
Ad esempio il numero esadecimale 14 é uguale a:
1*16(E)1 + 4*16(E)0
Come si era visto per i numeri deçimali.
Ecco di seguito una tavola di conversione dei primi 16 numeri decimali in esadecimale e binario:

DECIMALE	BINARIO	ESADECIMALE
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

La notazione esadecimale é molto importante e deve

diventare per il lettore di normale uso come il sistema decimale.
Basti pensare che qualunque programma disassemblatore, strumento indispensabile per sproteggere un programma, funziona con codici esadecimali sia in assembler che in disassembler (programmi che saranno discussi in seguito).
Nel capitolo dei programmi sarà presentato il listato di un programma in grado di convertire fra di loro, numeri in qualsiasi base.

NUMERI BINARI

Il sistema in base due, binario, é quello adoperato in tutti i calcolatori e si basa sugli stati logici delle varie porte dei circuiti elettronici che compongono il calcolatore stesso.
La cifra uno sta per stato logico 'on' cioé aperto mentre la cifra zero sta per stato logico 'off' cioé chiuso.
Con queste due sole condizioni qualunque calcolatore, piccolo oppure grande, esegue tutte le sue funzioni.
Un calcolatore che funzioni ad 8 bit, cioé a 8 cifre binarie alla volta é in grado di memorizzare in ogni locazione di memoria un numero pari ad 11111111 in binario, che tradotto in decimale fa 255.

Il sistema di numerazione più in uso é, come già detto, quello esadecimale in quanto esprime un modo più sintetico per riconoscere i contenuti delle varie locazioni di memoria.

DIVISIONE DELLA MEMORIA

La memoria del COMMODORE 64 ha una suddivisione molto originale, infatti risulta divisa in pagine, formate da 256 bytes ciascuna e numerate progressivamente a partire da 0.
Complessivamente la RAM disponibile al BASIC ammonta a soli 39870 bytes la restante area di memoria tranne un' altra parte di RAM non a disposizione del BASIC, é memoria ROM che contiene i programmi applicativi, come l'interprete del basic, il sistema operativo, il generatori di caratteri, le variabili del sistema ecc. ecc..
Tramite la manipolazione di un particolare registro di memoria é tuttavia possibile cambiare l' assetto della disposizione della memoria cosiddetta 'standard' ; questo registro é situato nella prima pagina di memoria denominata pagina 0 e secondo la selezione dei bit può escludere parti di ROM trasformandole in RAM; questa RAM nascosta non é accessibile ai programmi in basic, ma può essere adoperata per scrivere programmi in linguaggio macchina.
Sui modi di attuazione di quanto appena detto saranno dedicati ampi discorsi nei prossimi capitoli.
Di seguito una tabella con la mappatura standard all' accensione del calcolatore:

```
DA 0000  A  00FF -- VARIABILI DI SISTEMA
DA 0100  A  01FF -- AREA STACK MICROPROCESSORE
DA 0200  A  02FF -- FLAG DI SALTO E PUNTATORI
DA 0300  A  03FF -- VETTORI DEL BASIC E DEL S.O.
DA 0400  A  07E7 -- MEMORIA VIDEO
DA 07E8  A  07FF -- PUNTATORI DEGLI SPRITE
```

```
DA 0800  A  9FFF -- AREA PER PROGRAMMI IN BASIC
DA 8000  A  9FFF -- AREA PER CARTUCCE ROM
vDA A000  A  BFFF -- INTERPRETE BASIC
DA C000  A  CFFF -- AREA LIBERA DI MEMORIA RAM
DA D000  A  DFFF -- DISPOSITIVI DI I/O RAM COLORE
DA E000  A  FFFF -- SISTEMA OPERATIVO KERNAL
```

La prima pagina, la pagina zero, é molto interessante per i nostri scopi, cioé quelli di copiare programmi protetti, infatti in questa area di memoria sono contenuti molti puntatori che gestiscono alcune funzioni fondamentali, ma prima di esaminarli uno per uno vediamo come il COMMODORE 64 memorizza dei numeri in due locazioni contigue.

LOCAZIONI 43-44: In queste due locazioni é codificato il punto di inizio dei programmi basic.
La codifica avviene con il modo a byte basso e byte alto.
In altre parole una locazione di memoria può contenere un numero non più grande di 255 che corrisponde in binario ad una serie di otto bit tutti posti alti cioé ad 1, se il numero da memorizzare é maggiore di 255 deve essere scomposto in due numeri ognuno dei quali minore di 255 secondo un modo logico e ripetibile, vediamo come:
Il basic a disposizione dell' utente inizia dalla locazione 2048 dove deve esserci sempre il valore 0, ed il primo byte viene allocato a partire da 2049 ($0801), questo numero viene scisso in due bytes nella seguente maniera:

```
BL=INT(LOC/256)
BH=((LOC/256)-LO)*256
```

Dove:
BL= Bytes basso
BH= Bytes alto

LOC= Locazione di memoria da codificare.
Bisogna fare attenzione a mettere il byte basso nella prima locazione, nel nostro caso 43 e poi il bytes alto nella seconda locazione nel nostro caso 44.
Queste operazioni vanno eseguite con delle semplici poke: POKE 43,BL e POKE 44,BH
In questo modo si può, ad esempio, ingannare il sistema operativo andando a modificare i contenuti delle locazioni sopra esaminate facendo iniziare il basic da una locazione diversa di quella standard.
La pagina 0 contiene altri importanti puntatori che vengono regolarmente consultati ed aggiornati dal sistema operativo per lo svolgimento di vitali funzioni del calcolatore stesso, vediamoli:
LOCAZIONI 45-46
In queste due locazioni con il solito metodo di codifica é memorizzata la fine del programma basic in atto e il successivo inizio dell' area delle variabili del programma stesso.
variando questo puntatore é possibile registrare routine in linguaggio macchina allocate in coda al programma in basic le quali risultano completamente invisibili, si proceda nella seguente maniera:
1- scrivere il programma basic nel modo tradizionale;
2- entrare in monitor allocato a 49152 ($C000)
3- dall' esame dei puntatori di fine programma stabilire l' ultima locazione occupata dal programma stesso (le ultime tre locazioni di qualunque programma sono sempre tre zeri consecutivi);
4- cominciare ad assemblare il programma in linguaggio macchina in coda a quello basic, avendo cura di NON TOCCARE gli ultimi tre zeri che indicano al sistema operativo la fine dell' ultima riga di programma;
5- annotare l' ultima locazione occupata dal programma in linguaggio macchina e codificarla in

byte basso e byte alto;
6- uscire dal monitor con il comando X
7- pokare nelle locazioni di fine basic e nella giusta sequenza, (byte basso, byte alto) i valori ottenuti
8- eseguire un normalissimo SAVE
il nostro programma in basic é, ora, regolarmente salvato con in coda il programma in linguaggio macchina.
Vediamo ora di esaminare più in particolare come il COMMODORE gestisce la memoria.
Il COMMODORE ha 64 Kbytes di memoria RAM, oltre a 20 Kbytes di memoria ROM i quali contengono tutti i programmi di supporto, come l' interprete basic, il sistema operativo, il set di caratteri standard.
Questa quantità di memoria indirizzata sembra eccessiva, infatti un computer come il COMMODORE 64 con un bus dati a 16 bit non può indirizzare più di 64 Kbytes, in condizioni normali.
Per potere controllare più memoria di quella possibile il microprocessore del calcolatore si serve di una porta di input/output (I/O) e un registro di locazione dati.
Questi due importantissimi registri sono posizionati, rispettivamente nella locazione 1 e nella locazione 0.
Nella porta di controllo di I/O della locazione 0 i bit, all' accensione sono così suddivisi:

NOME	BIT	DIREZIONE	DESCRIZIONE
LORAM	0	OUTPUT	Scambio RAM/ROM area Basic
HIRAM	1	OUTPUT	Scambio RAM/ROM S.O Kernal
CHAREN	2	OUTPUT	Scambio RAM/ROM I/O
	3	OUTPUT	Linea scrittura cassetta
	4	INPUT	Controllo cassetta
	5	OUTPUT	Controllo motore cassetta

6	Non Usato
7	Non Usato

La configurazione di bit con il basic attivato il sistema operativo pronto ed il set caratteri standard, é la seguente:

BIT	CONTENUTO
0	1
1	1
2	1
3	1
4	0
5	1

Che in decimale fa 47, questo é il valore riscontrabile nella locazione 0 all' accensione del sistema.
In altre parole é possibile variando i bit della locazione 0 ad esempio disabilitare il basic ed usare le stesse locazioni come RAM.
Naturalmente in questo caso il basic non esisterà più e i programmi vanno scritti necessariamente in linguaggio macchina o con altro interprete caricato dall' esterno.
Esaminiamo ora le diverse linee di controllo per ottenere le diverse configurazioni di memoria:
LORAM:
Questa linea é normalmente alta (posta ad 1) e se viene posta bassa (a 0) esclude il basic attivando 8 Kbytes di RAM che vanno da $A000 ad $BFFF.
HIRAM:
Questa linea é normalmente alta e se posta bassa

esclude il sistema operativo KERNAL attivando 8 Kbytes di RAM mappati da $E000 a $FFFF.
CHAREN:
Questa linea é usata per escludere dall' area indirizzata dal microprocessore una ROM di 4 Kbytes che contiene il set dei caratteri standard del COMMODORE.
Questa linea é posta generalmente alta e viene posta bassa quando si devono riprogrammare tutti i caratteri del set o parte di esso oppure per esigenze grafiche.

E' da notare che se vengono alterati i bits di queste porte il computer non si danneggia ma é molto facile provocare il blocco del sistema, in questi casi basta spegnere e riaccendere il computer per ripristinare le condizioni iniziali.
Ricapitolando, possiamo dire che il microprocessore del Commodore 64 ha un bus da 16 bit.
Questo significa che é in grado di indirizzare 65536 locazioni di memoria, cioé 64 KByte.
Di questi 64 Kbyte però, 16 sono riservati a memoria ROM e contengono l' interprete basic ed il sistema operativo; rimangono facendo la ovvia sottrazione soltanto 44 Kbyte a disposizione (tenendo conto che 4 Kbytes sono normalmente impegnati per il CIA complex interface adapter).
Il commodore 64 per indirizzare più memoria di quella matematicamente possibile, usa un sistema che consiste nel mettere la memoria aggiuntiva in parallelo a quella già esistente.
Questo vuol dire che gli indirizzi saranno gli stessi ma una porta si occuperà di commutarli anche via software, le locazioni di questa porta sono già state precedentemente prese in esame.
Facciamo, ora, qualche prova su quanto detto per

comprendere meglio come stanno le cose:
Nel Commodore 64 la memoria nascosta si trova parallela alla ROM locata a partire dall' indirizzo $A000, 40960 in decimale fino ad $FFFF in decimale 65535, ad eccezione di 4 Kbytes locati da $C000, 49152 in decimale, fino ad $CFFF, in decimale 53247, che sono di RAM non accessibile al basic e quindi rappresentano un luogo ideale per allocare pezzi di linguaggio macchina al sicuro da pericolose sovrapposizioni.
Tutto ciò si può verificare facendo:
POKE 50000,12:PRINT PEEK(50000)
Come risposta si otterrà 12, segno che questa locazione di memoria é RAM perfettamente accessibile, infatti rientra in quei 4 Kbytes compresi tra $C000 e $CFFF.
Se, invece proviamo a digitare:
POKE 60000,12:PRINT PEEK(60000)
avremo come risultato 235; questo vuol dire che quella locazione é sicuramente ROM.
Il 12, che si é tentato di scrivere nella locazione 60000, non é andato perduto, ma, é stato scritto nella RAM parallela.
Possiamo dedurre che un tentativo di scrittura in una zona ROM sarà accettato dalla RAM parallela, mentre un tentativo di lettura sarà accettato solo dalla ROM.
Selezionando i banchi di memoria, come già detto in precedenza, é possibile scambiare zone di ROM con le sottostanti zone di RAM.
Per fare ciò bisogna adottare qualche precauzione, infatti, se si scambiasse, ad esempio, la ROM dell' interprete basic con la corrispondente ram, il sistema operativo tenterebbe di leggere dati dell' interprete sempre dalle stesse locazioni solo che troverebbe al posto della ROM con il giusto

programma, della RAM disordinata, risultato, il blocco del sistema che si può ripristinare solo spegnendo e riaccendendo il calcolatore.
A titolo do esempio, vediamo come si può copiare l' interprete basic dalla ROM alla RAM per poi poterlo modificare a piacimento:

- Digitare la seguente linea basic in modo diretto:
FOR T=40960 TO 49151:POKE T,PEEK(T):NEXT
- Digitare, sempre in modo diretto:
POKE 1,54

Nella prima fase, vienre ricopiato tutto l' interprete basic nella RAM parallela, ed nella seconda si seleziona la porta informando il sistema operativo, di leggere la RAM corrispondente alla ROM del basic.
Questa volta il sistema non va in blocco perche il sistema operativo continua a trovare nelle nuove locazioni, tutte le routine che gli servono.
Come si può facilmente dedurre, queste nozioni aprono nuovi orizzonti alla programmazione; unico problema, é che bisogna saper programmare in assembler, ma con un poco di buona volontà e qualche testo specifico si può centrare qualunque risultato.

LINGUAGGIO MACCHINA DEL COMMODORE 64

IL commodore 64 é architettato su un microprocessore molto diffuso: il 6510 della famiglia CMOS direttamente derivato dal più vecchio 6502, conservando una compatibilità totale con quest' ultimo.
Il linguaggio macchina, di questo processore, non é poi molto difficile e ci si potrebbe fare ingannare dalla apparente complessità dell' argomento.
Tuttavia, é sufficiente inpadronirsi dei concetti fondamentali per, gradatamente, riuscire a programmare in questo affascinante linguaggio.
Scopo do questo libro, non é quello di fare un corso di assembly, pur tuttavia viene fornita qualche indispensabile nozione per potere svolgere il lavoro di copiatura dei programmi protetti in una qualche maniera.
Il linguaggio macchina é il linguaggio adoperato dal microprocessore per eseguire i nostri programmi.
Ogni istruzione che viene fornita alla macchina in basic, viene tradotta da un apposito programma in codice macchina, perché il codice macchina é l' unico linguaggio che il computer può eseguire.
Facciamo, ora un paragone, con un semplice programma in basic:

```
10 B=1:C=1
20 D=B+C
40 PRINT D
```

Questo programmino tutti siamo in grado di comprenderlo, vediamo come, in realtà, il computer lo recepisce:

A9 01 69 0A 8D 00 04 A9 01 8D 00 D8 60

Questo é il modo con cui il nostro Commodore esegue il semplice programma in basic sopra visto, ma questa serie di numeri, al contrario del programma in basic, difficilmente riusciremo a capirla; per questo motivo, é stato creato un linguaggio denominato assembly che permette di programmare in modo molto molto simile al codice macchina diretto, vediamo come si potrebbe scrivere in assebly il semplice programma in basic di prima:

```
LDA *$01    B=1
ADC *$01    SOMMA 1
STA $0400   SCRIVI IL RISULTATO
RTS         FINE
```

IL programma risuta certamente più comprensibile di prima.
Il linguaggio assembly é basato in definitiva, su di una serie di codici come LDA, ADC, STA, RTS, ed altri che vengono subito codificati in codice macchina e direttamente eseguiti dal microprocessore.
Prima di parlare di codici assembler bisogna parlare di come questi codici possono agire; queste tecniche si chiamano indirizzamenti:

INDIRIZZAMENTO IMMEDIATO: il codice della istruzione relativa é direttamente eseguibile senza attingere notizie o dati dalla memoria centrale

INDIRIZZAMENTO ASSOLUTO: il codice o i codici interessano una locazione di memoria specifica durante l' esecuzione della istruzione stessa

INDIRIZZAMENTO IN PAGINA ZERO: vale quanto detto per

il precedente indirizzamento, solo che le locazioni intreressate sono le prime 256 e vengono numerate da 0 a 255.
Questo indirizzamento si distingue dal precedente in quanto più veloce nell' esecuzione.

INDIRIZZAMENTO INDICIZZATO: l accesso alla memoria di questo tipo di indirizzamento, é condizionato dal valore dei registri programmabili X ed Y, come vedremo in seguito.

INDIRIZZAMENTO IMPLICATO: é un indirizzamento poco usato in quanto si usa solo con l' istruzione JMP (salto incondizionato); la locazione di memoria dove saltare con il JMP con questo tipo di indirizzamento, viene codificata come byte basso, byte alto in altre due locazioni di memoria contigue.

INDIRIZZAMENTO INDIRETTO INDICIZZATO: simile all' indirizzamento indicizzato con la differenza che le locazioni di memoria interessate sono indicizzate esclusivamente dal registro Y.

INDIRIZZAMENTO INDICIZZATO INDIRETTO: come il precedente solo che come indicizzazione vengono adoperati degli spazi in pagina zero; questo é l' unico tipo di indirizzamento indicizzato che consente di accedere a tutta la memoria del commodore 64.

Esaminiamo, ora una per una, tutte le istruzioni del microprocessore 6510:

LDA carica l' accumulatore con un numero o un byte

LDX carica il registro x con un numero o un byte

LDY carica il registro y con un numero o un byte

STA deposita il contenuto dell' accumulatore

STX deposita il contenuto del registro x

STY deposita il contenuto del registro y

TAX scambia l' accumulatore con il registro x

TAY scambia l' accumulatore con il registro y

TXA scambia il registro x con l' accumulatore

TYA scambia il registro y con l' accumulatore

NOP nessuna operazione per il 6510

JMP salto incondizionato

JSR salto ad una soubroutine

RTS ritorno al basic

INC incrementa un byte di una unità

INX incremente il registro x di una unità

INY incrementa il registro y di una unità

DEC decrementa il byte di una unità

DEX decrementa il registro x di una unità

DEY decrementa il registro y di una unità

CMP compara l' accumulatore

CPX compara il registro x

CPY compara il registro y

BEQ salta se uguale a zero

BNE salta se diverso da zero

BCC salta se il carry é zero

BCS salta se il carry é uno

BVC salta se overflow é zero

BVS salta se overflow é zero

BPL salta se maggiore di 128

BMI salta se minore di 128

BRK interruzione forzata

PHA deposita accumulatore nello stack

PHP deposita il registro st nello stack

PLA preleva il valore in cima allo stack

PLP preleva e deposita il valore dello stack

TXS deposita il registro x nello stack

AND and logico con l' accumulatore

ORA or logico con l' accumulatore

EOR or esclusivo con l' accumulatore

BIT and logico particolare

ADC somma un numero all' accumulatore

SBC sottrae un numero dall' accumulatore

SEC mette a uno il bit di carry

CLC mette a zero il bit di carry

SED inserisce il modo decimale

CLD disinserisce il modo decimale

SEI disabilita gli interrupt

CLI abilita gli interrupt

RTI ritorno da un interrupt

CLV mette a zero il bit di overflow

ROR rotazione di un bit a destra

ROL rotazione di un bit a sinistra

ASL spostamento di un bit a sinistra

LSR spostamento di un bit a destra

Naturalmente il set di istruzioni non é sufficiente a programmare in linguaggio macchina in quanto esiste una certa sintassi da rispettare, ma é utile per riconoscere e seguire il filo logico di un programma scritto da altri autori.

IL SISTEMA OPERATIVO DEL COMMODORE 64

Questo complesso programma che permette tutte le funzioni base del calcolatore pur rimanendo trasparente all' utilizzatore prende, il nome di KERNAL.
Tutto l' input e l' output e le varie gestioni interne, sono controllate dal Kernal.
All' accensione del calcolatore il Kernal parte automaticamente e prende il controllo della macchina stessa attraverso un processo che prende il nome di inizializzazione.
Questo processo consiste in più fasi, vediamole in dettaglio:

- Viene riattivato il puntatore dello stack e attivato il modo decimale

- Il kernal controlla se é presente una cartuccia nella porta posteriore, se questa é presente l' inizializzazione viene interrotta e il controllo viene ceduta alla ROM della cartuccia e, in genere, si ottiene un autostart molto difficile de inibire.
Se invece nessuna cartuccia é presente la procedura continua.

- Il Kernal inizializza tutti i dispositivi di input ed output e di conseguenza il bus seriale, viene attivato il timer e la scansione della tastiera ha inizio, viene selezionata la mappa di memoria basic e viene spento il motore del registratore a cassette.

- Il Kernal esegue, ora, un test sulla RAM inizializza la pagina zero ed il buffer di cassetta.

- Viene azzerato ed impostato lo schermo, viene attivato l' editor ed il controllo viene affidato al basic.

Il calcolatore a questo punto é pronto per accettare comandi: dalmomento dell' accensione sono trascorsi pochi secondi.
Il kernal é un programma completamente trasparente all' utente, nel senso che non ci si accorge che esiste, ma, può essere adoperato sfruttando le prezziose routine in esso contenute.
Fortunatamente é nota la tavola dei salti del Kernal che contiene tutti gli indirizzi delle routine del sistema operativo.
Vediamo, ora, come sfruttare a nostro vantaggio questa peculiarità del commodore 64.
Per usare correttamente una, o più, routine del Kernal, bisogna seguire determinate procedure che si possono riassumere in tre passi fondamentali:

1) PREPARAZIONE
2) CHIAMATA DELLA ROUTINE
3) GESTIONE DELL' ERRORE

Per preparazione si intende l' insieme di tutte quelle procedure preliminari alla chiamata della routine del Kernal, come ad esempio, caricare l' accumulatore con un certo valore che verrà poi trasmesso alla routine vera e propria oppure eseguire un azzeramento dei registri ecc. ecc..
La chiamata della routine viene eseguita tramite un jsr (salto ad una soubroutine) e alla fine il controllo viene restituito all' interprete basic oppure al programma monitor.
Al ritorno, la routine può trasmettere dei codici di errore, i quali, possono essere gestiti da programma

nella maniera più opportuna; ad esempio se si vuole caricare un file da disco con l' ausilio delle routine del Kernal, ed il file non venisse trovato sul disco, non si otterrebbe il solito "file not found" ma solo un codice di errore caricato sul' accumulatore o altro registro che può essere tranquillamente gestito dal programmatore.
Ecco quì di seguito la tavola dei salti delle varie routine del Kernal:

NOME	INDIRIZZO	DESCRIZIONE
F1	65445	accetta un byte dalla porta seriale
F2	65478	apre il canale di input
F3	65481	apre il canale di output
F4	65487	accetta un carattere dal canale
F5	65490	immette un carattere nel canale
F6	65448	trasferisce dati porta seriale
F7	65409	inizializza editor di schermo
F8	65511	chiude tutti i canali
F9	65475	chiude un file logico specifico
F10	65484	chiude tutti i canali in/out
F11	65508	prende un carattere dal buffer
F12	65523	base dell' in/out
F13	65412	inizializza l' in/out
F14	65457	dispone a ricevente il bus
F15	65493	carica la ram da dispositivo
F16	65436	imposta la base della memoria
F17	65433	imposta il top della memoria
F18	65472	apre un file logico
F19	65520	legge/imposta il cursore
F20	65415	inizializza la ram
F21	65502	legge il clock
F22	65463	legge lo status
F23	65418	ripristina in/out
F24	65496	salva la ram su dispositivo
F25	65493	scandisce la tastiera

F26	65517	ritorna la pos del cursore
F27	65427	trasmette indirizzo secondario
F28	65466	imposta gli indirizzi
F29	65424	controlla i messaggi del Kernal
F30	65469	imposta il nome del file
F31	65499	imposta il clock
F32	65442	tempo bus seriale
F33	65505	termina la scansione tastiera
F34	65460	trasmette su dispositivo
F35	65430	invia indirizzo secondario sul bus
F36	65514	incremente il clock
F37	65454	imposta il bus non ricevente
F38	65451	imposta il bus non trasmittente
F39	65421	legge/inposta in/out

A titolo di esempio, vediamo come si può adoperare la F15 che carica la ram da un dispositivo, cioé esegue un normale load.
La routine é la seguente:

```
LDA *$ device   imposta il codice del dispositivo
LDX *$ fileno   imposta il numero del file logico
LDY *$ cmd      imposta l' indirizzo secondario
JSR F15         routine Kernal F15
```

In questo modo si possono caricare programmi da disco o da nastro non da basic ma direttamente usando le routine del sistema operativo.
Questo può essere utile in quanto molti programmi protetti, usano inibire il load ed il save da basic, ma naturalmente usando le routine del Kernal, questa protezione viene a cadere.
Esaminiamo, ora, tutte le routine del sistema operativo con le relative locazioni in esadecimale:

$E000-$E042
Inizio del sistema operativo che contiene, ancora parti utili all' interprete basic.

$E043-$E08C
Converte l 'argomento di una funzione in un numero compreso fra zero ed 0.999999999.

$E08D-$E096
Routine utilizzata nella funzione RND

$E097-$E0F8
Esegue la funzione RND

$E0F9-$E10B
Controlla gli errori di i/o da basic

$E10C-$E111
Serve a prelevare un carattere dal buffer

$E118-$E11D
Manda un carattere in uscita sul canale

$E11E-$E123
Routine di apertura canale di entrata

$E124-$E129
Routine del basic che preleva un carattere dalla tastiera con una procedura simile a quella ottenuta con il comando basic get.

$E12A-$E155
Serve ad eseguire la funzione sys.controlla se dopo la funzione sys esiste una virgola e carica i valori numerici che la seguono nell' accumulatore nel registro x e nel registro y; poi procede a mandare in esecuzione la routine in liguaggio macchina indicata

dal primo parametro, infine scarica i registri nelle locazioni di memoria 780,781,782,783 e cede il controllo al basic.

$E156-$E15E
Esegue il comando SAVE, esamina il testo del comando basic ed esegue di conseguenza la routine apposita, dopo, cede il controllo alla routine successiva.

$E15F-$E164
Salva la memoria ram su di un dispositivo periferico usando una routine che vedremo in seguito locata a partire da $FFD8

$E165-$E167
Esegue il comando VERIFY, controlla il testo basic e preleva i parametri per la routine di LOAD

$E168-$E174
Esegue il comando LOAD, prelevando i parametri dal testo basic, infine salta alla routine seguente

$E175-$E1BD
Carica una zona di ram da un dispositivo periferico usando la routine di load locata a partire da $FFD5

$E1BE-$E1C0
Esegue il comando OPEN; controlla i parametri del testo basic e passa il controllo alla routine successiva.

$E1C1-$E1C6
Apre un file desiderato sfruttando la routine di open locata a partire dalla locazione$FFC0

$E1C7-$E1C9
Esegue il comando CLOSE usando la routine che segue

$E1CA-$E1D3
Chiude un file desiderato usando la routine di close situata a partire dalla locazione $FFC3.

$E1D4-$E1FF
Preleva i parametri del testo in basic per l' esecuzione delle rouitnes di SAVE,LOAD,VERIFY.

$E200-$E205
Esamina i dati dopo le virgole

$E206-$E20D
Controlla la lunghezza dei comandi, al raggiungimento dell' ultimo carattere, inizializza lo stak e setta tutti i parametri.

$E20E-$E218
Controlla che l' ultimo byte del testo non sia seguito da un due punti o un byte nullo, in questo caso stampa un SYNTAX ERROR.

$E219-$E263
Routine usata per i comandi OPEN e CLOSE.

$E264-$E26A
Esegue la funzione COS.

$E26B-$E2B3
Esegue la funzione SIN.

$E2B4-$E2DF
Esegue la funzione TAN.

$E2E0-$E2E4
Zona dove si trova memorizzata la costante in virgola mobile PIGRECO.

$E2E5-$E2E9
Zona dove si trova memorizzata la costane in virgola mobile PIGRECO*2.

$E2EA-$E2EE

Zona dove si trova memorizzata la costante in virgola mobile 0.25.

$E2EF-$E30D
zona di memoria dove sono memorizzate alcune costanti in virgola mobile utili per la funzione SIN.

$E30E-$E33D
Esegue la funzione ATN (arcotangente).

$E33E-$E393
Contiene alcune costanti utili per la funzione ATN.

$E37B-$E393
Routine di warm start; questa routine viene eseguita ogni volta che viene incontrata una istruzione di BRK oppure quando vengono premuti contemporaneamente, i tasti di RUN/STOP e RESTORE.

$E394-$E396
Routine di cold start reinizializza il basic ed esegue una routine di reset.

$E397-$E3A1
Inizializza la pagina zero della memoria, ed esegue una warm start.

$E3A2-$E3B9
Aggiorna dati in pagina zero.

$E3BA-$E3BE

Contiene dati per la funzione RND.

$E3BF-$E446
Inizializza la ram del basic e predispone la configurazione standard di memoria.

$E447-$E452
Zona di memoria in cui sono memorizzati in rom, tutti i vettori basic.

$E453-$E45F
Routine che aggiorna la pagina zero della memoria.

$E460-$E472
Zona di memoria che contiene la stringa BYTES FREE.

$E473-$E4AC
Zona di memoria che contiene i messaggi della schermata che si ottiene all' accensione del sistema o ad un suo reset.

$E4AD-$E4D9
Routine del basic per le operazione su periferica.

$E4DA-$E4DF
Contiene i parametri per il colore del bordo.

$E4E0-$E4EB
Routine di attesa per il caricamento di un programma da nastro, se si preme il tasto commodore, la routine viene saltata ed il caricamento viene subito eseguito.

$E4EC-$E4FF
Contiene costanti adoperate dall' interfaccia RS232.

$E500-$E504
Parte di sistema operativo che controlla tramite

interrupt, la gestione delle periferiche.

$E505-$E509
Routine che provvede alla organizzazione dello schermo in 25 linee e in 40 colonne.

$E50A-$E517
Routine che controlla le linee e le colonne dello schermo.

$E518-$E565
Inizializza tutte le procedure di I/O.

$E566-$E56B
Porta il cursore in posizione HOME (in alto a sinistra).

$E56C-$E599
Si occupa della gestione del cursore.

$E59A-$E59F
Reinizializza i registri del video.

$E5A0-$E5A7
Reinizializza le procedure di I/O.

$E5A8-$E5B3
Setta il VIC II.

$E5B4-$E5C9
Preleva i caratteri dal buffer di tastiera.

$E5CA-$E631
Routine che si occupa dell' INPUT

$E632-$E639
Input da tastiera.

$E63A-$E683
Input da schermo.

$E684-$E690
Ferma la procedura di tokenizzazione delle parole chiave incluse fra apici.

$E691-$E715
Stampa sullo schermo il contenuto dell' accumulatore.

$E716-$E8A0
Interpreta i due set di caratteri, i codici colore, e i caratteri di controllo tra apici.

$E8A1-$E8B2
Decrementa il contatore di linea.

$E8B3-$E8CA
Incrementa il contatore di linea.

$E8C8-$E8D9
testa il colore del cursore.

$E8EA-$EA30
Zona di memoria che contiene tutte le routine di scrolling dello schermo.

$EA31-$EA86
Routine di esecuzione di un interrupt richiesto (IRQ).

$EA87-$EB78
Routine di scansione della tastiera.

$EB79-$ED08
Tavole usate per convertire la avvenuta pressione di un tasto nel corrispondente codice ASCII.

$ED09-$ED0B
Routine che effettua un or logico tra il contenuto dell' accumulatore e $40.

$ED0C-$EDB8
Routine che esegue un or logico tra il contenuto dell' accumulatore e $20.

$ED89-$EDC6
Trasmette un indirizzo secondario al bus IEEE.

$EDC7-$EDDC
Trasmette un indirizzo secondario al bus IEEE.

$EDDD-$EDEE
Trasmette un byte attraverso il bus IEEE.

$EDEF-$EDFD
Trasmette un comando di chiusura al bus IEEE.

$EDFE-$EE12
Trasmette un comando di non ascolto al bus IEEE.

$EE13-$EEBA
Preleva un byte dal bus IEEE e lo memorizza nell' accumulatore.

$EEBB-$EF05
Sistema per la generazione di NMI (interrupt non mascherabile) usato dall' interfaccia RS232.

$EF06-$EF58
Trasmette un byte in uscita sulla interfaccia RS232.

$EF59-$F0BC
Routine che tratta gli NMI per i byte provenienti

dall' inerfaccia RS232.

$F0BD-$F12A
Zona di memoria che contiene tutti i messaggi di errore e di controllo del sistema operativo.

$F12B-$F13D
Questa routine serve a stampare sullo schermo i messaggi della routine precedente.

$F13E-$F156
Preleva un carattere dal buffer di tastiera e lo memorizza nell' accumulatore.

$F157-$F1C9
Preleva un carattere dal canale di input e lo memorizza nell' accumulatore.

$F1CA-$F20D
Trasmette ad un canale di uscita il contenuto dell' accumulatore.

$F20E-$F24F
Serve per la gestione di alcuni tipi di file particolari.

$F250-$F290
Considera un file come file di uscita dati.

$F291-$F32E
Chiude un file specifico.

$F32F-$F332
Chiude tutti i file sia di entrata che di uscita.

$F333-$F349
Questa routine reinizializza tutti i canali del

sistema.

$F34A-$F49D
Apre un file specifico.

$F49E-$F5DC
Carica in memoria un file specifico.

$F5DD-$F69A
Salva una zona di memoria su di un dispositivo.

$F69B-$F6EC
Routine di IRQ che provvede all' aggiornamento del clock, alla scansione della tastiera, e al test sulla pressione del tasto di STOP.

$F6FD-$F6FA
Controlla l' avvenuta pressione del tasto di STOP.

$F6FB-$F72B
Provvede, in caso di errore, a stampare il messaggio appropriato.

$F72C-$F80C
Trova e legge il blocco HEADER nel caricamento di un file da nastro.

$F80D-$F92B
Varie routines che controllano le operazioni con il nastro.

$F92C-$FA6F
Routines di controllo della lettura da nastro.

$FA70-$FBA5
Routines che trasferiscono dati al nastro.

$FBA6-$FCE1
Routines di controllo per la registrazione su nastro.

$FCE2-$FE42
Routine di cold start, viene eseguita all' accensione del computer; la memoria viene inizializzata e vengono abilitati tutti i collegamenti con le periferiche.
La parte iniziale della routine controlla la presenza di una cartuccia nell' apposita porta: se ne viene riscontrata la presenza il controllo viene ceduto al programma della cartuccia, viceversa, si continua con la normale routine di cold start.

$F343-$FF47
Routine di controllo per NMI.

$FF48-$FF80
Questa routine viene eseguita ad ogni chiamata di un interrupt e provvede a salvare i registri e ad analizzare la natura dell' interrupt.

Dall' attento studio di queste routine si possono fare molte esperienze utili ai nostri fini; ad esempio chiamando la routine del sistema operativo incaricata di leggere l' HEADER di cassetta locata a $F72C e facendola partire con l' apposita SYS, si può leggere questa importante parte di nastro, inibendo la presenza di un eventuale autorun presente.
Il valore delle informazioni contenute nel blocco iniziale di ogni registrazione, appunto l' HEADER, verrà discusso nel capitolo inerente.

INTERRUPT

Il concetto di interrupt, rientra con pertinenza di argomento nella nostra trattazione sulla protezione e sprotezione dei programmi, infatti molti trucchi adoperati da rinomate case di software per impedire la copiatura illegale dei propri programmi, si basano, appunto, sulla manipolazione di questo parametro della macchina.
Tutti i microprocessori, dal più semplice al più complesso, sono in grado di gestire un servizio alla volta, intendendo per servizio, qualunque operazione che il computer possa svolgere: l' esecuzione di un programma, la consultazione di una periferica, o la manipolazione di dati.
Alle volte si ha l' impressione che il computer possa eseguire più operazioni contemporaneamente, cosa naturalmente impossibile.
Ad esempio in alcuni videogiochi si sente una allegra musichetta mentre il giocatore sta sparando a delle feroci navi spaziali, l'illusione della contemporaneità é perfetta ma é, appunto, una illusione.
Il computer, in questi casi, esegue sempre una operazione alla volta, ma in modo talmente veloce che a noi umani sembrano simultanee; questa tecnica si ottiene manipolando l'interrupt.
Ogni processore contiene almeno una linea di interrupt, nella quale sono codificate tutta una serie di operazioni: per esempio, mentre nel Commodore lampeggia il cursore e la macchina aspetta istruzioni, vengono eseguile seguenti operazioni: la scansione della tastiera, la rilevazione dell' avvenuta pressione del tasto di

stop, il controllo delle periferiche, l' aggiornamento degli orologgi interni, il refresh delle memorie e del video ed altre ancora; tutte queste routine inserite nella linea di interrupt vengono eseguite ogni sessantesimo di secondo.
Esistono, nel Commodore, due interrupt:
IRQ(interrupt request)
ed NMI(not mascherable interrupt).
IL secondo tipo di interrupt gestisce operazioni particolari di competenza stretta del sistema operativo, il primo invece é facilmente manipolabile via software in quanto rappresentato da un vettore a due bytes modificabile a piacimento.
Il vettore a cui facevamo cenno é locato a 788 ($0314) ed 799 ($0315), al solito modo, byte basso e byte alto.
Per alterare il vettore di interrupt bisogna seguire una ben determinata procedura, vediamola:

- Calcolare il byte basso e il byte alto dell' indirizzo dove inizia la routine da inserire nella linea di interrupt nel seguente modo: dividere l'indirizzo per 256 e prenderne la parte intera, questo numero rappresenta il byte basso e va pokato nella locazione 788, poi prelevare il resto della divisione precedente e moltiplicarlo per 256 questo numero va pokato nella locazione 799 e rappresenta il byte alto del' indirizzo della nostra routine.
Adesso, la nostra routine inserita nell' interrupt, verrà eseguita ogni sessantesimo di secondo.
Naturalmente, quando si vanno a pokare il byte basso e il byte alto che rappresentano il nuovo indirizzo della nostra routine, quest' ultima dovrà trovarsi al suo posto !!! ciò vuol dire che la routine andrà assemblata prima di eseguire le due poke, pena il blocco del sistema e la perdita di tutto il lavoro

fatto fino al quel momento.

- La routine va assemblata segnalando al sistema operativo che stiamo inserendo qualcosa nella linea di interrupt; questo si ottiene con l' istruzione SEI, ed alla fine bisognerà comunicare che é necessario rimettere a posto la normale linea di interrupt, cosa che si ottiene con l' istruzione RTI.
La routine deve necessariamente terminare con un JMP alla locazione $EA31 per potere continuare ad eseguire tutte le normali operazione concatenate alla linea dell' interrupt; in pratica siamo riusciti ad inserire una routine nella normale catena di routine gestite dall' interrupt ed é quindi necessario non spezzare questa catena, pena spiacevoli risultati.

Facciamo un esempio, creiamo un programma in linguaggio macchina che incrementi, seguento il ritmo dell' interrupt cioé ogni sessantesimo di secondo, il colore dello schermo:

7000 SEI serve a disinserire i normali interrupt ed informa il sistema operativo che una nuova routine sta per essere inserita.
7001 LDA *$0D carica nell' accumulatore il valore del byte basso dell' indirizzo dove inizia la nuova routine da inserire, questo indirizzo é $700D.
7003 STA $0314 deposita l' accumulatore nel vettore di interrupt che contiene il byte basso, in pretica equivale ad eseguire una poke nella locazione 788.
7006 LDA *$70 carica nell' accumulatore il

valore del byte alto dell' indirizzo che codifica la nuova routine.
7008 STA $0315 e lo deposita nel vettore di interrupt che contiene il byte alto completando l' operazione di aggiornamento del vettore.
700B CLI istruzione complementare al SEI, viene, infatti, informato il sistema operativo che si siamo appena usciti da una situazione di gestione dell' interrupt.
700C RTS si ritorna al basic.
700D INC $D021 questa é la locazione $700D dove inizia la nuova routine, il comando serve ad incrementare di uno fino al massimo numero possibile che é 255 la locazione $D021 che serve a fare variare il colore dello sfondo.
7010 JMP $EA31 infine il necessario salto alla prossima routine di interrupt che é locata a $EA31 per non interrompere la catena di routine inserite nella linea dell' interrupt.
Con queste tecniche, si possono realizzare effetti sorprendenti e protezioni quasi inespugnabili, in particolare modo autorun che si autoproteggono dalle intrusioni esterne, oppure sempre tramite interrupt, il programma chiuso, controlla se in memoria risiede un programma estraneo ad esempio un monitor, ed in questo caso si autodistrugge.
Vedremo qui di seguito come fare a neutralizzare queste ed altre protezioni basate su questo principio.
Il segreto sta nel non fare caricare il programma nelle giuste locazioni, cioé, il programma funzionerà, protezione compresa, solo se al momento del caricamento verrà allocato nelle giuste posizioni.
Ma come fa il programma a sapere dove allocarsi ?, semplice, questo dato é codificato nel' HEADER.
Ma se riusciamo a modificare questa parte di header il

gioco é fatto, infatti possiamo fare caricare il programma in qualunque parte della memoria con il risultato di neutralizzare tutte le protezioni inerenti l' interrupt; non dimentichiamo che questo tipo di protezione resta efficace solo se il programma durante il caricamento é correttamente allocato.
Questo risultato si può ottenere usando il programma 'rilocatore' il cui listato é contenuto in questo libro.
Vediamo, esattamente la procedura da seguire:

- Caricare il programma 'rilocatore' e lanciarlo.

- Inserire nel drive il disco che contiene il programma da copiare.

- Tramite il programma rilocatore, variare l' indirizzo di partenza e posizionarlo, ad esempio, a 2049 (inizio del testo basic) annotando, però, il valore originale.

- Spegnere il sistema e riaccenderlo.

- Caricare il programma monitor rilocato a 49152 e lanciarlo

- Disassemblare il programma, prestando particolare attenzione se vengono variate le locazioni dove risiede il vettore dell' interrupt.

- Se viene riscontrata la condizione di cui al passo precedente, mettere nelle locazioni dell' interrupt viste in precedenza il giusto valore e cioé $EA31, mettendo nella prima locazione il byte 31 e nella seconda il byte EA.

- Registrare nuovamente sempre da monitor il programma su disco.

- Spegnere il sistema e riaccenderlo.

- Caricare il programma rilocatore e rimettere il punto di partenza al valore originale precedentemente annotato.

Il programma é ora sprotetto e pronto per essere copiato.
Dalla manipolazione di questi registri possono originarsi molte brutte sorprese; é, infatti, cosa molto frequente causare il blocco del sistema e perdere l' intero programma oppure rovinare il programma stesso, é sempre consigliabile eseguire delle copie di sicurezza del vostro lavoro, state sicuri che non é tempo perduto.

ORGANIZZAZIONE DEI DISCHETTI PER DRIVE 1541

Il dischetto del drive 1541 gestito da un commodore 64, é suddiviso in un numero standard di tracce pari a 35, ed un numero variabile di settori che va da 21 settori per le tracce più esterne a 17 settori per le tracce interne.
Il sistema operativo all' atto della formattazione del dischetto organizza una directory, dove sono memorizzati tutti i nomi dei file contenuti nel dischetto; inoltre viene organizzata una area del dischetto come BAM che vuol dire block availability map, cioé mappa di disponibilità dei blocchi.
Tutta questi dati vengono allocati nella traccia 18 che, quindi, non risulta disponibile per la memorizzazione dei dati.
I blocchi in totale, esclusi quelli della traccia 18, sono 664 ognuno di 256 bytes.
Quì di seguito una tabella, che mostra la distribuzione dei blocchi nelle varie traccie:

T R A C C I A	S E T T O R I
da 1 a 17	21
da 18 a 24	19
da 25 a 30	18
da 31 a 35	17

LA BAM DEL DRIVE 1541

La bam indica dove esiste disponibilità di blocchi nel dischetto; in altre parole, prima di scrivere un programma su disco il sistema operativo si incarica di controllare se su disco esiste lo spazio sufficiente a contenerlo ed eventualmente, quali sono i blocchi liberi dove allocarlo, tutto questo, al fine, di rendere impossibili sovrapposizioni di dati sul disco.
La bam viene aggiornata dal sistema operativo ogni volta che viene compiuta una operazione rivolta al disco, come una concellazione un salvataggio ecc. ecc..
Soltanto i comandi di lettura non provocano nessun aggiornamento della bam.
Vediamo, ora, come é organizzata la bam nella traccia 18 settore 0 di qualunque dischetto formattato dall' unità dischi 1541:

BYTE 0 contiene $12 traccia e settore del primo
BYTE 1 contiene $01 blocco della directory
BYTE 2 contiene $41 codice ascii della A
BYTE 3 contiene $00 non usato
BYTE 4-143 bit map dei blocchi liberi, un 1 indica blocco occupato ed uno 0 indica un blocco libero.

Alcune directory di programmi commerciali risultano pericolosamente alterate proprio nella bam, dico pericolosamente perché il trucco consiste nel disallocare nella bam i blocchi già occupati dal programma, mettendo a 0 i relativi byte della bam; chiaramente un' operazione di save su quel dischetto, provocherà un disastro in quanto, ci sarà sicuramente una sovrapposizione di dati con la conseguente distruzione del programma originario.

Queste protezioni sono veramente 'cattive' in quanto penalizzano fortemente chi compie certe operazioni senza particolari intenzioni piratesche ma solo per ingenuità.

LA DIRECTORY

La directory é contenuta nella traccia 18 sotto la bam e contiene le seguenti informazioni:
- Nome del disco
- Identificatore del disco (ID)
- tipo di sistema operativo per il disco (DOS)
- Nomi dei file
- Tipi dei file
- Blocchi occupati da ogni file
- Blocchi liberi nel dischetto

Sul dischetto, per motivi di spazio, possono essere allocati solo 144 file differenti.
Vediamo, ora, come sono organizzati nel disco i byte della traccia 18 settore 0 che seguono la bam, cioé dal 144 esimo in poi sino al 255 esimo:

BYTE 144-161:
Questi byte contengono il nome del disco che viene assegnato all' atto della formattazione.
I caratteri possono essere al massimo 16, se sono di meno il rimanente spazio viene riempito con spazi shiftati

BYTE 162-163:
Questi due bytes contengono l' identificatore del dischetto che viene assegnato all' atto della formattazione e, vedremo si trova su tutte le tracce per operazioni riservate al sistema operativo.

BYTE 164:
Questo byte contiene uno spazio shiftato codice $A0.

BYTE 165-166:

Questi bytes contengono i caratteri ascii del "2A" che compare in alto a destra nella directory, ed indica il tipo di unità dischi adoperata.

BYTE 167-170:
In questi byte sono contenuti soltanto spazi shiftati.

BYTE 171-225:
Questi bytes non sono adoperati dal dos.

Ecco, quì di seguito, una piccola routine che controlla il nome del disco.
Facendo partire il programma, viene caricato il nome del disco nella variabile DN$, per essere, eventualmente, testato.
Questa routine rappresenta una forma di protezione per alcuni programmi su disco, infatti, questi programmi smettono di funzionare se si prova a cambiare nome al disco, e la protezione é composta da una routine simile a questa che se non riscontra il nome giusto arresta il programma.
Usatela come credete, e se la trovate inserita in un programma commerciale, state tranquilli che lì gatta ci cova.

```
100 OPEN 15,8,15,"IO"
110 OPEN 2,8,2,"§"
120 PRINT §15,"B-R";2;0;18;0
130 PRINT §15,"B-P";2;144
140 DN$=""
160 FOR I = 1 TO 16
170 GET §2,X$
180 IF ASC(X$)=160 THEN 200
190 DN$=DN$+X$
200 NEXT
```

```
210 CLOSE 2:CLOSE 15
```

Naturalmente, come sempre il carattere § deve essere sostituito con il carattere cancelletto del commodore.

IDENTIFICATORE ID

L' identificatore é composto da due caratteri e viene dichiarato, dall' utente, all' atto della formattazione.
Il dos controlla l' identificatore ad ogni operazione di input output rivolta al dischetto, per vedere se il dischetto é stato sostituito nel frattempo.
Se, tramite la ID, il dos si accorge che il dischetto é stato sostituito, carica la bam del nuovo dischetto in mémoria.
In questo modo il drive ha sempre la bam del dischetto che é inserito in quel momento, in memoria; ciò snellisce la procedura di aggiornamento della directory.
Evidentemente, per quanto detto, é utile lavorare con dischetti che contengsno sempre una id diversa, ma questo, per vari motivi, non é sempre possibile.

BACK-UP

Per back-up si intende un programma in grado di copiare per intero un disco.
Naturalmente l' operazione risulta semplice se il disco in questione contiene programmi non protetti in un certo modo, se così non é l' operazione, pur disponendo di un back-up diventa difficoltosa.
Un programma può essere protetto nel migliore dei modi ma se io scrivo un back-up che legge byte per byte il programma da disco, come potrebbe fare una normale operazione di lettura, e lo riversa, sempre byte per byte su di un altro disco protezione compresa, ho aggirato l' ostacolo della protezione; sembra facile, ma i programmatori che studiano protezioni, oltre che proteggere il programma proteggono anche il disco da copie illegali.
Questi tipi di protezione sono molto complessi e richiedono una perfetta conoscenza sia del sistema operativo del calcolatore che di quello del drive.
Uno dei metodi più usati, si basa sulla produzione artificiale di un errore durante la lettura, errore che viene provocato su una traccia e su un settore specifico del disco.
L' errore generato viene controllato e gestito dal programma protetto; in genere se viene riscontrato l' errore previsto dal programmatore, il caricamento del programma va a buon fine altrimenti succedono le cose più strane, come la formattazione del dischetto o il blocco del sistema oppure ancora la distruzione del programma stesso secondo algoritmi molto complessi e sofisticati.
In genere il programmatore opera nella seguente maniera per proteggere un suo programma compreso il disco:

- Registra il programma sul disco

- Esamina le tracce del disco con un programma simile ad all disk contenuto nel libro, e ne individua una vuota

- Su questa traccia produce un errore con programmi simili a genera 22 contenuto nel libro.

- Inserisce nel programma da proteggere una routine che verifica l' avvenuto riscontro dell' errore durante il caricamente; se l' errore si é verificato il programma parte regolarmente, altrimenti no

Il programma così protetto, quando viene caricato; funziona nella seguente maniera:

- Si richiama il programma con LOAD"NOME",8,1

- Parte il caricamento ad un certo punto del quale il programma va a testare la traccia con l' errore, si sente un rumore nel drive come se si stesse formattando il dischetto in effetti é la testina che, per un comando del sistema operativo, viene forzata a fondo corsa fino a sbattervi ripetutamente

- Viene generato l' errore che produce l' effetto voluto.

Se si tenta di copiare con un back-up, un programma siffatto accadono le seguenti cose:

- Si carica in memoria il back-up e lo si lancia

- Il back-up comincia a leggere traccia per traccia settore per settore tutto il disco

- quando si arriva alla traccia rovinata che contiene l' errore il back-up si arresta, segnala l' errore all' operatore con un codice che varia in funzione dell' errore generato, e salta la traccia continuando a copiare o si blocca completamente; in ogni caso la traccia che contiene l' errore non é stata riprodotta, il disco é incompleto.
Se si tenta di fare partire il programma così copiato, al momento che il programma testa la traccia per constatare la presenza dell' errore, naturalmente non lo trova e va in tilt.
Questo metodo di protezione é molto efficace, ma con un pò di pazienza e con gli strumenti giusti si può arrivare ad ottenere buoni risultati, vediamo come:

- Caricare il programma "esaminaerrori" e lanciarlo.

- Introdurre nel drive il disco da copiare.

- Analizzare tramite l' opzione 1 o la 2 del programma "esaminaerrori" il disco ed annotarsi le tracce in cui compare una segnalazione di errore.

- Spegnere il sistema e riaccenderlo, compreso il drive.

- Caricare il back-up veloce ed eseguire una copia del dischetto protetto ignorando gli eventuali messaggi di errore che il back-up fornisce all' utilizzatore.

- Spegnere e riaccendere il sistema, compreso il drive.

- Caricare il programma "copiatracce" e lanciarlo.

- Copiare tramite il programma "copiatracce" le tracce 'incriminate' dal disco originale a quello di copia.

Se tutto é stato eseguito correttamente la copia dovrebbe essere uguale in tutto e per tutto all' originale, si ce ne può sincerare mandando in esecuzione il programma appena copiato.
Se ancora non si riesce a far partire la copia, conviene ritentare l' intero procedimento e se ancora non si dovesse ottenere il risultato voluto é probabile che il disco contenga qualche altro sistema di protezione.
Se invece tutto é andato bene, siamo riusciti a copiare il nostro programma, ma non a sproteggerlo, nel senso che abbiamo si la copia ma siamo sempre vincolati alle traccie che contengono gli errori.
Per sproteggere veramente il programma, bisogna eliminare completamente questa condizione; per fare ciò occorre molta esperienza di programmazione in linguaggio macchina e bisogna conoscere molto bene il sistema operativo del drive, in fondo la copia é stata fatta, ci possiamo contentare anche di questo.
Per i curiosi, spero la maggior parte di voi, dirò che esiste nel programma una routine che si occupa di gestire l' errore generato e che prende i provvedimenti del caso se questo errore non viene riscontrato in fase di caricamento.
Eliminando questa routine abbiamo eliminato la protezione che ci ha fatto tanto lavorare, il problema é riconoscere la parte di programma dove é codificata questa routine.
Per individuare la sezione che si occupa dell' errore, bisogna esaminare come si comporta il programma quando si deve difendere contro una copia illegale, ad esempio, alcuni programmi se non

riscontrano l' errore nelle copie si autocancellano tramite un reset provocato da un salto alla locazione 64738, lo stesso effetto si può ottenere da basic digitando, appunto, SYS 64738.
Per individuare il punto critico bisognerà cercare all' interno del programma i due bytes che compongono l 'indirizzo del reset e da lì seguire il percorso che il programma ci indica tramite i vari JMP o JSR.
Provateci, ma non scoraggiatevi se le prime volte sembra difficile.
Per alcuni tipi di errore basta la semplce copiatura con un back-up il quale in alcuni casi riesce a trasferire l' errore anche alla copia é il caso dell' errore con codice 23 che viene trasferito perfettamente dal back-up veloce, provare per credere.
Nei casi più ostinati può essere utile provare ad eseguire una copia con il back-up blocchi il cui listato si trova nel libro, il quale anche se un pò più lento del back-up veloce vi stupirà per le sue splendide virtù.
Se dopo tutti questi tentativi, ancora, non siete riuscire a copiare il vostro disco, vuol dire che, molto probabilmente esso contiene delle intertracce, o delle tracce al di fuori delle normali 35.
Questo é reso possibile dal fatto che le prime tracce del disco, contengono un nuovo sistema operative diretto al drive, in grado di leggere tracce o intertracce che normalmente verrebbero rifiutate.
Quando si tenta di copiare questo disco con un back-up tradizionale la testina del drive nei suoi spostamenti segue le tracce standard e procede con un certo passo, ignorando tutte le tracce che restano al di fuori da questi spostamenti.
Il back-up in questo caso dovrebbe scandire tutta la superficie del disco, con soluzione di continuità

arrestandosi non appena viene riscontrata una traccia piena di dati, copiandola.
Il super back del libro assolve questi compiti.
Per motivi tecnici non é stato possibile inserire come listato, il programma di altri tre back-up copiatutto e di altre utility per procedere alla sprotezione dei dischi; questi programmi fanno, però parte del dischetto che si può richiedere alla casa editrice.

PROCEDURA DI AUTOSTART

Per autostart si intende un programma che all' accensione del computer viene automaticamente caricato in memoria ed eseguito senza alcun intervento dall' esterno.
Questa esigenza nasce per due motivi:
- Rendere piu' confortevole l' uso dei programmi in quanto vengono eliminate le procedure di caricamento e relativo lancio dei programmi da utulizzare
- Rendere efficaci le varie protezioni disseminate nel programma stesso e che diventano attive solo dopo che il programma e' stato lanciato.

Un tipico esempio di procedura autostart é rappresentata dalle cartucce rom che si inseriscono nella porta che si trova dietro il Commodore e che contengono del software.
Queste cartucce vanno inserite a computer spento e vengono attivate quando il computer viene acceso, con il risultato che il programma viene caricato ed eseguito senza che l'operatore possa intervenire bloccando questo processo.
Nel commodore 64 non si possono realizzare procedure di questo tipo, se il programma risiede su nastro o su disco, si puo', tuttavia simulare, tale comportamento via software con appositi programmini i quali devono essere scritti necessariamente in linguaggio macchina e devono risiedere in parti specifiche della memoria.
Il programma che seguira' sara' in grado di emettere un run quando viene eseguito un load su disco o su nastro.
Diamo ora, prima di esaminare il programma, uno sguardo al sistema operativo del Commodore 64 ed in particolare rivolgiamo la nostra attenzione su di una

zona di memoria particolare, quella compresa tra $02A7 ed $0303 (i due numeri sono espressi in notazione esadecimale lo si può notare dal simbolo $ che precede il numero).
Le locazioni comprese tra $02A7 ed $02FF sono a disposizione dell' utente, ed é in queste locazioni che noi assembleremo la nostra routine che provochera' l' autostart al programma desiderato.
E' di fondamentale importanza usare queste locazioni di memoria in quanto pochi bytes più sotto si trovano due locazioni le quali contengono l' indirizzo della routine di warm start.
La routine di warm start viene eseguita dal sistema operativo ogni volta che il computer ha terminato una operazione, ad esempio, alla fine di un caricamento eseguito con load viene eseguita questa routine e successivamente quella che fa' comparire il familiare READY con il cursore lampeggiante.
Poiché l'indirizzo di inizio del warm start é conservato in due locazioni, già viste che risiedono in ram, i contenuti di queste ultime possono essere modificati a piacere facendoli puntare, ad esempio, all' inizio della nostra routine, così si potrà caricare in memoria, contemporaneamente, la routine ed il vettore modificato ed ottenere che alla fine del load venga automaticamente eseguita la routine in questione.
Vediamo ora come scrivere il programma da autostart:
- Caricare il programma MONITOR del libro con load "MONITOR",8,1 da disco e load "MONITOR",1,1 da cassetta;
- Digitare sys 49152 il programma MONITOR dovrebbe partire;
- Entrare in fase di assemblaggio digitando A 02A7
.....
dove al posto dei puntini bisognerà mettere il codice

da assemblare
- Premere Return, se tutto va bene il dato viene accettato e compare automaticamente l' indirizzo della prossima locazione per il prossimo dato; e così via
- Alla fine del caricamento PRIMA DI FARE PARTIRE IL PROGRAMMA salvarlo su disco o su nastro con la seguente sintassi: save" B00T" 08 02A7 0304 per il disco per la cassetta cambiare il codice 08 con 01
- Ora e soltanto ora provare il programma

Esaminiamo il listato in codice assembler

```
02A7   0F 08 00 0A 93        Codici ASCII linea basic
02AC   22 42 2A 22 2C        10 LOAD "B*",8,1
02B1   38 2C 31 00 00 00     +++--------------------+++
02B7   LDA *$08              Mettere 01 PER NASTRO
02B9   TAX                   Assegna numero file
02BA   LDY *$01              periferica e comando
02BC   JSR $FFBA             secondario
                             +++--------------------+++
02BF   LDA *$02              Lunghezza del nome del
02C1   LDX *$F3              file e locazione di
02C3   LDY *$02              inizio del nome
02C5   JSR $FFBD             predispone   caricamento
                             +++--------------------+++
02C8   LDA *$00              Disabilita tutti i
02CA   STA $9D               messaggi ed inizia il
02CC   JSR $FFD5             caricamento
                             +++--------------------+++
02CF   STX $2D               salva l' indirizzo di
02D1   STY $2E               fine del basic
                             +++--------------------+++
02D3   LDA *$83              Riposiziona il vero
```

```
02D5   STA $0302              Warm Start al
02D8   LDA *$A4               valore originale
02DA   STA $0303
                              +++--------------------+++
02DD   LDX *$06               Nel buffer di tastiera
02DF   STX $C6                mette CLEAR+ RETURN e
02E1   LDA $02EC,X            RUN+ RETURN
02E4   STA $0276,X
02E7   DEX
02E8   BNE $02E1
                              +++--------------------+++
02EA   JMP $(0302)            Torna al Basic
                              +++--------------------+++
02ED   43 CC 0D               Codici che servono al
02F0   52 D5 0D              ciclo di lettura di sopra
                              +++--------------------+++
02F3   20 20 20 20 20 20 20   nome del programma
                              +++--------------------+++
0300   8B E3
0302   B7 02                  Vettore da modificare
```

Notare che lo star (*) che compare nel listato, va sostituito nel Commodore con il simbolo del cancelletto (SHIFT+3).
Nei tratti in cui non esistono codici assembler bisogna digitare direttamente i codici, in quanto questi non corrispondono a vere istruzioni ma a dati; ecco come si opera:
-da MONITOR digitare M ..... dove i puntini rappresentano la locazione di inizio e premere return, comparira' una fila di otto codici

- Portarsi con i tasti cursore sul primo codice e digitare il dato voluto poi spostarsi sul successivo fino all' ottavo dato, premere return e passare agli

otto dati successivi.
Salvata la routine di autostart con il nome di BOOT, si deve procedere a salvare di seguito il programma da fare partire in autostart.
Se tutto é stato eseguito correttamente, chiamando il caricatore BOOT, questo automaticamente deve caricarsi il programma successivo.
Senza dubbio questa é già una forma di protezione, per la verità poco valida, infatti, basta premere il tasto stop perché il programma si arresti diventando vulnerabile.
Si può inoltre caricare direttamente il programma senza passare per il blocco caricatore e poi salvarlo come si vuole.
Per rendere efficace questo sistema di protezione bisogna agire su due fronti:
- Fare in modo che il programma sia caricato e quindi lanciato, solo tramite il caricatore; in altre parole, se il programma viene caricato da solo, deve autodistruggersi;
- Fare in modo che il programma una volta partito in autorun attraverso il caricatore, non possa piu' essere bloccato in nessun modo, e se accidentalmente lo fosse, il listato deve essere incomprensibile.
Vediamo, ora, come realizzare questi due obbiettivi:
Per rendere obbligatoria la procedura di autostart, il programma deve essere caricato solo tramite il caricatore che contiene la routine di autostart appena vista.
Per fare ciò bisogna inserire nel caricatore alcuni controlli che poi devono essere riscontrati nel programma madre.
Ad esempio, se si inserisce nel caricatore il seguente controllo:

```
LDA *$FF
STA $0400
```

che vuol dire carica nell' accumulatore il valore $FF e depositalo nella locazione $0400 (che equivale a fare da basic POKE $0400,$FF), alla fine del caricamento la locazione $0400 conterrà il valore $FF.
Il programma madre quando parte, dovrà verificare questa condizione con una linea basic di questo tipo:
IF PEEK($0400) diverso da $FF THEN SYS 64738
così se il caricamento é avvenuto tramite caricatore la condizione non viene verificata ed il programma avra' un regolare svolgimento; se, invece il caricamento é avvenuto senza il caricatore la condizione viene verificata (perché la locazione $0400 non contiene $FF) ed il programma si autodistrugge con il salto alla routine di reset che parte da 64738.
Naturalmente si possono inserire altri controlli o più condizioni rendendo più efficace la protezione.
A titolo di esempio si puo' adoperare un metodo di protezione usato con successo da molte case di software, inserendo dei caratteri speciali nel nome assegnato al programma stesso in fase di registrazione si possono generare dei codici mascherati che vengono successivamente testati dal programma madre, vediamo come:
Quando viene registrato un programma su nastro, il sistema operativo provvede a creare un blocco di dati preliminare al programma stesso; questo blocco di dati prende il nome di HEADER.
Nell' header vengono memorizzati in sequenza il tipo di file (programma sequenziale programma non rilocabile) l' indirizzo di partenza del programma, l' indirizzo dell' ultima locazione del programma, ed infine i caratteri ASCII che rappresentano il nome del programma.
Quando si esegue il load, tutti i dati dell header

vengono trasferiti nella memoria del calcolatore nelle seguenti locazioni di memoria: da $033C a $03FB questo spazio di memoria viene definito come BUFFER di cassetta.
Il nome viene allocato esattamente, a partire dalla locazione $0341.
Se si salva un programma nella seguente maniera, ad esempio,
A$="(colore giallo)(colore originale) name prog" e poi
SAVE A$
salveremo su nastro il programma name prog, ma con il nome preceduto da due caratteri che all' atto della stampa sullo schermo non saranno visibili, poiché il secondo carattere riporta il colore come era all' origine.
Controllando l' esistenza del codice 158 (colore giallo) nella locazione $0341 il programma madre sapra' se é stato caricato dal regolare caricatore o no.
Adesso bisogna fare in modo che il programma una volta lanciato non sia più arrestabile.
Qui di seguito vengono elencate una serie di locazioni che contengono gli indirizzi di partenza di essenziali routine del sistema operativo, vediamole:

- $0306 $0307 Vettore di list
- $0316 $0317 Vettore di brk
- $0318 $0319 Vettore di nmi e restore
- $0328 $0329 Vettore di stop
- $0332 $0333 Vettore di save

Questi vettori risiedono in ram e possono essere alterati a piacere provocando degli effetti a volte incontrollabili.
Il vettore di list punta alla routine omonima che risiede in rom, variando le locazioni di puntamento, si può ingannare il sistema operativo e fare in modo che ogni volta che viene richiesto il list di un

programma il sistema operativo salti, o ad una routine appositamente scritta dall' utente, oppure in un posto della memoria casuale, provocando il blocco del sistema.
I vettori di brk e di nmi(Not Mascherable Interrupt) funzionano come quello di list e, se alterati, servono a bloccare le funzioni dei tasti run/stop e restore.
Per proteggere un programma che parte in autorun si opera in questo modo:
- Caricare il programma Monitor rilocato a partire da $C000 (49152) e farlo partire;
- Con l' istruzione F (Fill) riempire tutta la memoria di 00 con F 0801 9FFF 00;
- Caricare da monitor il programma da proteggere con l' istruzione L "NOME" 08 oppure 01;
- Con l' istruzione M 0801 visionare tutto il programma fino alla fine, che si può individuare da tutta la serie di zeri creata al passo precedente, ed annotare questo valore;
- Assemblare con le modalità già viste, la routine di autorun nelle locazioni già eseminate;
- con il comando M del monitor modificare a piacere tutti i vettori di salto tranne quello relativo al SAVE;
- Registrare con il monitor tutto il segmento di memoria da $02A7 alla fine del programma, con la seguente istruzione: S "NOME" 08 02A7 FINE.
A questo punto spegnere tutto e provare a caricare il programma creato, il quale dovrà partire non appena caricato, in modo automatico e non dovrà essere vulnerabile alla pressione dei tasti RUN/STOP e RESTORE.

SPROTEZIONE AUTORUN

Un modo molto elegante per togliere qualsiasi autorun da disco, consiste nell' ingannare il programma facendogli credere che l' autorun non esiste !.
Per parlare di questo metodo, facciamo qualche richiamo ad un argomento già trattato nel capitolo dedicato alla gestione dei dischi.
Quando viene registrato un programma su disco, esso, viene spezzettato in blocchi di lunghezza pari a 256 bytes.
Di questi 256 bytes, solo 254 sono a disposizione del programma, perché gli altri due sono a disposizione del D.O.S. del drive; in questi due bytes risiedono due importantissime informazioni, il primo byte contiene la traccia del prossimo blocco di dati e il secondo il settore.
Se il primo byte é zero, vuol dire che si tratta dell' ultimo blocco di dati per quel programma ed in questo caso, il byte successivo indicherà la posizione dell' ultimo byte del programma all' interno di quel blocco.
Quando viene letto un programma con un normale LOAD, il dos legge uno per uno tutti i bytes del programma tranne i primi due di ogni blocco che, come si é visto, servono per concatenare i vari blocchi fra di loro.
Il terzo ed il quarto byte del programma, indicano al sistema operativo dove allocare il programma, cioé a partire da quale locazione; se il programma in questione é un programma scritto interamente in basic, possiamo essere certi che i valori corrispondenti al terzo e quarto byte saranno $01 e $08, in quanto il basic inizia alla locazione $0801.

Se la locazione di inizio del programma é minore di $0801 o 2049 in decimale, vuol dire che, quasi sicuramente questo programma é protetto da un' autorun e da altre diavolerie.
Ed ecco l' idea: basta, durante la fase di load, eliminare tutto quello che si trova sopra la locazione 2049 !.
Provate per credere, questo metodo funziona quasi sempre e con ottimi risultati; per quelle rare volte che non va, esiste un altro metodo un poco più complesso che vedremo più avanti.
Ecco ora il listato del programma che consente la copia con protezioni ed autorun:

```
10 INPUT" NOME DEL PROGRAMMA DA SPROTEGGERE";NO$
20 INPUT" NOME PROGRAMMA SPROTETTO";NS$
30 OPEN 2,8,0,NO$
40 OPEN 3,8,1,NS$
50 REM ************************************
60 REM ************************************
70 REM *** LEGGE I PRIMI DUE BYTES      ***
80 REM ************************************
90 GET§2,A$,B$
120 REM SCRIVE SUL FILE IN USCITA
140 PRINT§3,CHR$(1) CHR$(8);
190 M=ASC(A$+CHR$(0))+ASC(B$+CHR$(0))*256
240 IF M (magg) 2048 THEN 1040
250 FOR X=M TO 2048
260 GET§2,A$
270 NEXT
1040    GET§2,A$:IF    ST    (diverso)    0    THEN
PRINT§3,A$;:CLOSE 2:CLOSE 3:END
1050 PRINT§3,CHR$(ASC(A$+CHR$(0)));:GOTO 1040
```

Attenzione che il simbolo § nella digitazione del listato sul commodore va sostituito con il

cancelletto (SHIFT + 3)e le scritte fra parentesi (magg),(min),(diverso), vanno sostituite con i rispettivi segni del basic commodore.
L' altro tipo di protezione a cui facevo cenno prima, risulta un poco più laboriosa ma, fondamentalmente, si basa sullo stesso principio, solo che si evita di tosare tutto quello che c'é sopra la fatidica locazione 2049, in quanto alle volte, operare questo taglio provoca arresti al programma irreversibili.
Capita sovente, che le parti che il precedente programma taglia contengano informazioni vitali al buon funzionamento del programmo stesso; in questo caso bisogna usare un programma diverso, contenuto nel libro, chiamato ' rilocatore'.
Si procede nel seguente modo:

- Caricare il programma rilocatore dal disco.

- Inserire il disco che contiene il programma da sproteggere nel drive.

- Fare partire il programma rilocatore ed indicare il nome del programma da sproteggere.

- Viene visualizzato l' indirizzo di partenza del programma, annotare questo valore.

- Modificare l' indirizzo di partenza con 2049, e registrarlo nuovamente su disco.
in questo modo tutto il programma, compreso protezioni ed autorun, vengono allocati nell' area basic, solo che non potrà mai funzionare in queste condizioni, perché non allocato correttamente.

- Caricare il programma MONITOR $C000 e lanciarlo

con SYS 49152.

- Entrare con il comando D nel programma a partire dalla prima locazione ($0801).

- Individuare la routine di autorun che in genere, é posta all' inizio del programma e termina con un JMP ad una locazione che rappresenta la SYS del programma.

- Togliere il JMP al programma, annotandone il valore e sostituirlo con due NOP ed un RTS finale, BADANDO DI OCCUPARE GLI STESSI BYTES.

- Registrare il tutto su disco con un nuovo nome.

- Ricaricare il programma rilocatore e lanciarlo.

- Inserire il disco che contiene il programma modificato e registrato con il nuovo nome.

- Modificare la locazione di partenza, che sarà 2049, con quella originale, che si era annotata.

Il gioco é fatto, caricare il programma, il quale alla fine del caricamento, se si é operato correttamente, non partirà in autorun, e farlo partire con la SYS convertita in decimale, che era stata eliminate nella routine di autorun.
Se neanche questo metodo ha successo, conviene non perdere ulteriore tempo é usare uno dei back-up in dotazione e copiare tutto il dischetto.
Si raccomanda di non lavorare mai sul programma originale che potrebbe rovinarsi irreparabilmente, ma sempre su una copia back-up funzionante.
Naturalmente tutte queste operazioni sono da

considerare 'via software' e quindi non danneggiano mai il computer; al massimo si può perdere il programma e bisogna ricominciare di nuovo, quindi, niente paura a provare nuovi metodi, non succede nulla, anzi se nentrovate dei nuovi, fatemeli sapere.
Esistono dei metodi di sprotezione che coinvolgono l' hardware ma ritengo, siano un tantino pericolosi per la vita del computer stesso, quindi, non ne parlerò; ed ora buon lavoro a tutti.

COPIATURA PROGRAMMI PC IBM

I programmi scritti per il pc IBM e compatibili sono scritti sotto MS/DOS e per questo motivo possono girare su tutti i computer muniti di questo sistema operativo.
Naturalmente le copie prodotte con i metodi appresso descritti, DEVONO servire solo ed esclusivamente come copie di lavoro ad uso strettamete personale; non dimentichiamo che il programma é di proprieta' della casa che lo commercializza, sia IBM o altro, e che diffondere clandestinamente un programma copiato, costituisce reato perseguibile penalmente.
Fatta questa doverosa premessa veniamo al dunque:
il sistema di protezione che viene usato sui programmi per pc IBM consiste nel posizionare determinate informazioni nonsenso nelle tracce del disco.
Le informazioni nonsenso possono essere di svariata natura: un errore di controllo oppure un codice illegale o anche un dato alfanumerico in un posto sbagliato, tutto questo produce un blocco del drive durante la copia.
Per potere effettuare delle copie funzionanti di programmi protetti in questa maniera, bisognerebbe conoscere a menadito il pc IBM come sistema operativo interno, mappe di memoria e architettura delle periferiche.
Per evitarvi questa fatica e per mirare subito al sodo abbiamo pensato di scrivere un back-up che legge automaticamente i caratteri nonsenso e li riproduce fedelmente nelle tracce del disco copia.
Alle volte basta rimuovere i caratteri nonsenso per rimettere a posto le cose, e il nostro back-up

prevede anche questa possibilita' consentendo, di ottenere delle copie funzionanti e prive di errori. Il back-up funziona come un normale programma di copia e svolge automaticamente le operazioni sopra descritte.

A U T O R U N

```
5 PRINT"s]]]]]]R+ AUTO RUN 1.0                "
10 PRINT"Q]]]]]]]]]QUESTO PROGRAMMA CREA"
20 PRINT"Q]]]UN PROGRAMMA IN LINGUAGGIO MACCHINA"
25 PRINT"Q]CHE PROVOCA L'AUTO START DEL PROGRAMMA"
30 PRINT"Q]SPECIFICATO,I CUI DATI DOVRANNO ESSERE"
40 PRINT"Q]INSERITI NEL MODO RICHIESTO."
50 PRINT"QQ]]]]]]]]]SEI PRONTO ?
(S/N)":GOSUB10000:IFA$="N"THENEND
60 PRINT"qR             ATTENDERE PREGO                "
70 RESTORE:FORI=49152TO49184:READA:POKEI,A:NEXT
75 FORI=679TO767:POKEI,0:NEXT
80 FORI=679 TO 749:READ X:POKE I,X:NEXT
100 PRINT"sQQNOME DEL PROGRAMMA DA CARICARE MAX 16
CH":INPUTN$
105 IF LEN(N$)>16THEN100
110 INPUT"Q]]]DALLA PERIFERICA N.";P
120 INPUT"QQNOME DEL LOADER";NL$
122 IF LEN(NL$)>16THEN120
125 INPUT"Q]DA SALVARE SULLA PERIFERICA N.";PL
130 PRINT"RQ    NON FERMARE IL PROGRAMMA !!!          "
150 REM
160 LE=LEN (N$)
170 POKE 709,LE:POKE 702,P
175 FOR I=0 TO LE-1:POKE750+I,ASC(MID$(N$,I+1,1)):NEXT
180 REM SALVATAGGIO
190 LL=LEN(NL$)
200 FORI=0TOLL-1
210 POKE 49185+I,ASC(MID$(NL$,I+1,1))
220 NEXT
230 POKE 49155,PL:POKE 251,167
240 POKE 49162,LL:POKE 252,2
250 POKE 770,167:POKE 771,2
260 SYS 49152
270 POKE 770,131:POKE 771,164
280 PRINT"Q]]]]]]]]]RANCORA ? (S/N)"
```

```
290 GOSUB10000:IFA$<>"S"ANDA$<>"N"THEN290
300 IFA$="S"THEN70
310 END
980
DATA169,0,162,0,160,255,32,186,255,169,0,162,33,160,192,32,18
9,255,169
985 DATA251,162,4,160,3,32,216,255,169,0,32,195,255,96:REM
SAVE
990 DATA
169,131,141,2,3,169,164,141,3,3,169,142,141,40,3,169,230,141
991 DATA
41,3,169,0,162,0,160,1,32,186,255,169,0,162,238,160,2,32,189
992 DATA 255,169,0,162,255,160,255,32,213,255,134,45,132,46
993 DATA
169,82,141,119,2,169,213,141,120,2,169,13,141,121,2,169,3,133
,198,96
10000 GET A$:IF A$="" THEN 10000
10001 RETURN
```

ROM COPY

```
790 PRINT"s"
800 PRINT"]]]]]]]]]]]]]]]]ROM COPY"
810 PRINT"Q--------------------------------------"
900 PRINT"QUESTO PROGRAMMA E' STATO ADATTATO PER-"
910 PRINT"TRASFERIRE LE CARTRIDGE (HES) SU' DISCO"
920 PRINT"AL PRIMO INPUT RISPONDERE CON LA LOCAZI"
930 PRINT"ONE DA' DOVE SI PRESUME INIZI LO START-"
935 PRINT"DELLA CARTRIDGE"
940 PRINT"AL SEC. INPUT FINE LOCAZIONE DELLA"
950 PRINT"CARTRIDGE E QUINDI TITOLO"
960 PRINT"I VALORI VANNO INSERITI IN DECIMALE"
970 PRINT"QQVALORI APPROSIMATIVI START 32767"
975 PRINT"Q]]]]]]]]]]]]]]]]]]]]END   36863"
976 QQQQ$:IFA$=""THEN976
985 GETA$:IFA$=""THEN985
1000 POKE36879,14
1010
PRINT"sESTART,END,NAME":INPUTV,W,V$:R=540:FORJ=1TO43:READT:PO
KER+J+5,T:NEXT
1020
DATA169,1,162,8,160,0,32,186,255,165,0,166,49,164,50,32,189,2
55,169,128,133,157,96
1030
DATA169,0,166,251,164,252,32,213,255,96,169,251,166,253,164,2
54,32,216,255,96
1040 T=LEN(V$):POKE0,T:U=1:S=256*PEEK(50)+PEEK(49)+T
1050 IFV$=""THEN1070
1060 FORJ=1TOT:POKES-J,ASC(RIGHT$(V$,J)):NEXT
1070 SYS546:U=V:T=252:GOSUB1120:IFW=0THEN1090
1080 U=W:T=254:GOSUB1120:SYS579:END
1090
FORJ=0TO5:POKER+J,PEEK(45+J):NEXT:SYS569:FORJ=0TO5:POKE45+J,P
EEK(R+J):NEXT
1100 IFSTATUSAND48THENPRINT:PRINT"?LOAD":PRINT"ERROR";
1110 END
1120 POKET,INT(U/256):POKET-1,U-256*PEEK(T):RETURN
```

C O N V E R S I O N I

```
10 REM CONVERSIONE NUMERI A BASI DIVERSE
20 PRINT"s";TAB(240);:INPUT"BASE DEL NUMERO DA CONVERTIRE";B1
30 PRINT:PRINT:INPUT"BASE DEL NUOVO NUMERO";B2
40 PRINT"s":PRINT"NUMERO BASE";B1,"NUMERO BASE";B2:PRINT
50 DIMA$(18)
60 IFB1<>10THENGOSUB200:GOTO80
70 GOSUB100:GOTO350
80 IFB2=10THEN300
90 GOSUB105:GOTO350
100 INPUT A:IFA=0THEN END
105 X=33
110 Q=INT(A/B2):R=INT(A-B2*Q)
120 X=X-1:IFX=28 ORX230RX=18THEN X=X-1
125 IFR>9THEN160
130 PRINTTAB(X)"q";R
140 IFQ>0THEN A=Q:GOTO110
150 PRINT:RETURN
160 PRINTTAB(X)"q";"]";CHR$(R+55):GOTO140
200 INPUTA$:IFA$="0"THEN END
210 N=LEN(A$):A=0
215 FOR I=NTO1STEP-1
220 C=ASC(MID$(A$,I,1)):IFC<65THEN C=C-48:GOTO240
230 C=C-55
240 A=A+B1↑(N-I)*C:NEXT:RETURN
300 PRINTTAB(20)"q";A:PRINT
350 IFB1=10THEN70
360 GOTO60
370 PRINT"C.BASE"
```

L. M. ---> D A T A

```
59900 PRINT"sNUMERO DI LINEA DI PARTENZA":PRINT"Q(0-255)Q"
59901 INPUTK$:K=VAL(K$):POKE0,K
59902 INPUT"sLOCAZIONE DI PARTENZA";D$:D=VAL(D$)
59903 PRINT"sULTIMA LOCAZIONE":INPUTE$:E=VAL(E$)
60000 IFPEEK(0)=255THENPRINT"sQQQQQQQQNUMERO DI LINEA TROPPO
GRANDE"
60001 IFD>ETHENPRINT"sQQQQQ]]]]JFINE":STOP
60002 PRINT"sQQQQQQD="D+12":E="E":Go60000SQQ"
60010 K=PEEK(0):F$=STR$(K)+"DATA"+STR$(PEEK(D))
60100 PRINTF$;
60200 FORD=D+1TOD+10
60300 K$=","+MID$(STR$(PEEK(D)),2,3)
60400 PRINTK$;
60500 NEXT:POKE0,PEEK(0)+1
60510 PRINT"S"
60520 POKE631,13:POKE632,13:POKE633,13:POKE198,3
```

D. O. S.

```
1 rem *** D.O.S. ***
0 GOSUB196
2 PRINT"sRMENUQQQ"
4 PRINT"F1    AUTORUN DI UN FILE NEL DIRECTORY
6 PRINT"F2    INIZIALIZZAZIONE
8 PRINT"F3    COPIA CON NOME DIVERSO
10 PRINT"F4    CANCELLA UN FILE
12 PRINT"F5    MOSTRA IL DIRECTORY
14 PRINT"F6    MOSTRA LO STATUS DEL DISCO
16 PRINT"F7    CAMBIA IL NOME DI UN FILE
18 PRINT"F8    COPIA IL MINI DOS SU ALTRO SUPPORTO
20 PRINT"↑     CAMBIA NOME E ID DEL DISCO
22 PRINT"Q     FINE DEL LAVORO
24 PRINT"QQQQQSCELGA PER CORTESIA :    "
25 GETX$:IFX$=""THEN25
26 IFX$="9"THENGOSUB48
28 IFX$="i"THENGOSUB86
30 IFX$="f"THENGOSUB112
32 IFX$="j"THENGOSUB136
34 IFX$="l"THENGOSUB170
36 IFX$="Q"THENGOSUB100
38 IFX$="k"THENGOSUB184
40 IFX$="e"THENGOSUB148
42 IFX$="h"THENGOSUB124
44 IFX$="↑"THENGOSUB220
46 GOTO2
48 PRINT"s":OPEN1,8,0,"$"
50 GET#1,A$,B$
52 GET#1,A$,B$
54 GET#1,A$,B$
56 C=0:IFA$<>""THENC=ASC(A$)
58 IFB$<>""THENC=C+ASC(B$)*256
60 PRINT"R"MID$(STR$(C),2);TAB(7);"r";
62 GET#1,B$:IFST<>0THEN78
64 IFB$<>CHR$(34)THEN62
66 GET#1,B$:IFB$<>CHR$(34)THENPRINTB$;:GOTO66
```

```
68 GET#1,B$:IFB$=CHR$(32)THEN68
70 PRINTTAB(29);:C$=""
72 C$=C$+B$:GET#1,B$:IFB$<>""THEN72
74 PRINT"R"LEFT$(C$,3)
76 IFST=0THEN52
78 PRINT"BLOCCHI LIBERI "
80 CLOSE1
82 GOSUB214
84 RETURN
86 PRINT"s"
88 PRINT"INSERIRE IL DISCO DA INIZIALIZZARE":PRINT
90 PRINT"NOME DEL DISCO":INPUTDI$
92 PRINT"NUMERO DI ID":INPUTEX$
94 MA$="N."+DI$+","+EX$
96 OPEN15,8,15,MA$
98 CLOSE15:MA$="":GOSUB214:RETURN
100 REM
102 PRINT"QFINE DEL LAVORO. SICURO? (S/N)"
104 GETA$:IFA$=""THEN104
106 IFA$="N"THEN2
108 IFA$="S"THENSYS64738
110 IFA$<>"S"ORA$<>"N"THEN104
112 PRINT"s"
114 PRINT"NOME DEL FILE-SORGENTE ":INPUTDI$
116 PRINT"NOME DEL NUOVO FILE ":INPUTNW$
118 MA$="C:"+NW$+"="+DI$
120 OPEN15,8,15,MA$
122 CLOSE15:MA$="":GOSUB214:RETURN
124 PRINT"s"
126 PRINT"VECCHIO NOME DEL FILE":INPUTDI$
128 PRINT"NUOVO NOME DEL FILE":INPUTNW$
130 MA$="R:"+NW$+"="+DI$
132 OPEN15,8,15,MA$
134 CLOSE15:MA$="":GOSUB214:RETURN
136 PRINT"s"
138 PRINT"NOME DEL FILE DA CANCELLARE":INPUTDI$
140 PRINT"PREMERE -RETURN- PER CANCELLARE":INPUTX$
```

```
142 MA$="S:"+DI$
144 OPEN15,8,15,MA$
146 CLOSE15:MA$="":GOSUB214:RETURN
148 PRINT"sIL FILE DA FAR GIRARE PARTE CON RUN O "
150 PRINT"CON UNA SYS ? SCRIVERE  -R-  O  -S-"
152 GETA$:IFA$=""THEN152
154 IFA$="R"THEN160
156 IFA$="S"THEN164
158 IFA$<>"R"ORA$<>"S"THEN152
160 PRINT"SCRIVI IL NOME DEL FILE":INPUTN$
162 LOADN$,8:RUN
164 PRINT"SCRIVI L'INDIRIZZO DI PARTENZA":INPUTK
166 PRINT"SCRIVI IL NOME DEL PROGRAMMA":INPUTN1$
168 LOADN1$,8,1:SYSK
170 PRINT"s"
172 PRINT"INSERISCI IL DISCO-DESTINAZIONE":PRINT
174 PRINT"PREMI -RETURN- PER COPIARE MINI DOS":INPUTX$
176 OPEN1,8,15
178 SAVE"DOS",8
180 CLOSE1
182 GOSUB214:RETURN
184 PRINT"s":OPEN1,8,15
186 INPUT#1,A,B$,C,D
188 PRINT"STATO DEL DISCO":PRINT:PRINT"ERRORE # ";A
190 PRINTB$:PRINT"TRACCIA ";C,"SETTORE ";D
192 CLOSE1:GOSUB214:RETURN
194 REM INSERIRE ROUTINE DI RENAME DISK/ID
196 POKE53280,0:POKE53281,0:PRINT"X"
198 PRINT"sR                              DOS              "
200 PRINT"QQQ]]]<C> 1985 "
202 PRINT"QQQQ
204 PRINT"Q
206 PRINT"Q
208 PRINT"QQQQQPER CORTESIA PREMERE UN TASTO
210 GETA$:IFA$=""THEN210
212 RETURN
214 PRINT"QQUN TASTO PER IL MENU'"
```

```
216 GETA$:IFA$=""THEN216
218 RETURN
220 POKE 53280,0:POKE
53281,0:PRINT"■":DIMA$(255):OPEN1,8,15,"I"
222 OPEN3,8,3,"#"
224 PRINT#1,"U1:"3;0;18;0
226 FORI=0TO255:GET#3,A$:IFA$=""THENA$=CHR$(0)
228 A$(I)=A$:NEXT:A$="":PRINT"s@"
230 CLOSE3:CLOSE1:FORI=144TO159:A$=A$+A$(I):NEXT
232 PRINT"s@PRECEDENTE   NOME   :";A$:A$=""
234 INPUT"@NUOVO   NOME   :";A$:IFA$=""THENT=16:GOSUB262
236 A$=LEFT$(A$+"                    ",16)
238 FORI=1TOLEN(A$):A$(143+I)=MID$(A$,I,1):NEXT
240 I$=A$(162)+A$(163):A$=""
242 PRINT"@@PRECEDENTE  ID    :";I$
244 INPUT"@NUOVO   ID. = ";A$:IFA$=""THENT=2:GOSUB262
246 A$=A$+"  ":A$(162)=LEFT$(A$,1):A$(163)=MID$(A$,2,1)
248 OPEN1,8,15,"I"
250 OPEN3,8,3,"#"
252 PRINT#1,"B-P:"3;0
254 FORI=0TO255:PRINT#3,A$(I);:NEXT
256 PRINT#1,"B-P:"3;0
258 PRINT#1,"U2:"3;0;18;0
260 CLOSE3:PRINT#1,"I":CLOSE1:GOTO2
262 FORI=1TOT:INPUT"CODICE ASC ";Z:A$=A$+CHR$(Z):NEXT: RETURN
```

ON ERROR

```
5 REM *** ON ERROR ***
10 POKE56,PEEK(56)-1:POKE52,PEEK(52)-1
20 IN=PEEK(56)*256+PEEK(55)
30 FORI=0TO99:READN:POKEIN+I,N:NEXT
40 POKE768,PEEK(55):POKE769,PEEK(56)
50 REM LN%=LINEA DELL' ERRORE
60 REM ST=CODICE ERRORE
2000 DATA 138,016,003,076,116,164,133,144,164,058
2010 DATA 200,240,080,169,128,072,169,204,133,069
2030 DATA 169,197,133,070,160,000,032,235,176,160
2040 DATA 000,165,058,145,071,200,165,057,145,071
2050 DATA 169,204,133,069,169,206,133,070,160,000
2060 DATA 032,235,176,104,160,000,177,071,133,021
2070 DATA 200,177,071,133,020,165,043,166,044,032
2080 DATA 023,166,144,017,165,095,233,001,133,122
2090 DATA 165,096,233,000,133,123,104,104,076,234
3000 DATA 167,162,017,076,058,164,234,234,234,234
JREADY.
J
```

TRACK COPY

```
0 REM *** TRACK-COPY ***
1 INPUT"sQQTRACCIA DA COPIARE     ↑↑↑↑";T:IFT<1ORT>35THEN1
5 PRINT"sQ            --- TRACK COPY ---"
10 R$="INSERIRE DISCO SORGENTE & PREMERE
RETURN":I=146:GOSUB1000
20 D$="APERTURA CANALE DI ERRORE":GOSUB2000
25
OPEN1,8,15,"I0":OPEN5,8,5,"#":DIMA$(20),B$(20):D$="":GOSUB200
0:GOSUB3000
35 IFT<18THENR=20
40 IFT>17ANDT<25THENR=18
50 IFT>24ANDT<31THENR=17
60 IFT>30THENR=16
65 GOSUB4000
70 FORS=0TOR:GOSUB4000:D$="LETTURA BLOCCO DAL
DISCO":GOSUB2000
80
PRINT#1,"U1:";5;0;T;S:GOSUB3000:D$="":GOSUB2000:D$="LETTURA
DATI":GOSUB2000
90 FORI=0TO254:GET#5,A$:IFA$=""THENA$=CHR$(0)
100 A$(S)=A$(S)+A$:IFST=0THENNEXTI
105 GET#5,B$(S):IFB$(S)=""THENB$(S)=CHR$(0)
110 GOSUB2000:NEXTS
115 D$="CHIUSURA DI TUTTI I FILES E
CANALI":GOSUB2000:CLOSE1:CLOSE5
120 R$="INSERIRE NUOVO DISCO & PREMERE
RETURN":I=146:GOSUB1000
130 D$="":GOSUB2000:D$="APERTURA CANALE DI ERRORE":GOSUB2000
135 OPEN1,8,15,"I0":OPEN5,8,5,"#":GOSUB3000
140
D$="":GOSUB2000:FORS=0TOR:PRINT#1,"B-P:";5;0:D$="SCRITTURA
BLOCCO":GOSUB2000
150 GOSUB4000:PRINT#5,A$(S)B$(S);
160 PRINT#1,"U2:";5;0;T;S
170 GOSUB3000
200 NEXTS
210 D$="":GOSUB2000:D$="VUOI ALLOCARE I BLOCCHI
```

```
(S/N)?":GOSUB2000
220 GETA$:IFA$<>"N"ANDA$<>"S"THEN220
230 IFA$="N"THEN255
240 D$="":GOSUB2000:FORI=0TOR:D$="ALLOCATING
BLOCK":GOSUB2000:S=I:GOSUB4000
250 PRINT#1,"B-A:";0;T;I:NEXT:D$="":GOSUB2000
255 D$="CHIUSURA DI TUTTI I FILES E CANALI":GOSUB2000
260 CLOSE1:CLOSE5:FORI=0TO100:NEXT
265 D$="":GOSUB2000:D$="DISCO COPIATO"
270 GOSUB2000:END
1000 PRINT"SQQQQQQ"SPC((40-LEN(R$))/2)CHR$(I)R$
1010 FORV=0TO50:GETG$:IFG$=CHR$(13)THENRETURN
1020 NEXTV
1030 IFI=18THENI=146:GOTO1000
1040 IFI=146THENI=18:GOTO1000
2000 IFD$=""THEND$="r                                                    "
2002 PRINT"SQQQQQQ                                                      "
2005 PRINT"Q--------- O P E R A Z I O N E ----------"
2010 H=(40-LEN(D$))/2:PRINTSPC(H)"R"D$;
2020
PRINTTAB(80)"QQQQQQQr----------------------------------------
":RETURN
3000 PRINT"SQQQQQQQQQQQQQQQQ]]]DISK'STATUS R";
3010 INPUT#1,A$,B$,C$,D$
3020 V$=",":PRINTA$V$B$V$C$V$D$"r
":IFA$<>"00"THEN4020
3030 RETURN
4000 PRINT"SQQQQQQQQQQQQ";
4010 PRINT"]]]RTRACCIA "T"r  RSETTORE"S"I  ":RETURN
4020 PRINT"SQQQQQQQQQQQQQQQQQQQQQQQQR]]]]]]]  DISK ERROR
CONTINUO ?  "
4030 GETA$:IFA$<>"S"ANDA$<>"N"THEN4030
4040 IFA$="N"THENCLOSE1:CLOSE5:END
4050 PRINT"SQQQQQQQQQQQQQQQQQQQQQQ
     ":RETURN
JREADY.
J
```

E S A M I N A I D

```
1 DIMA$(40)
10 REM ESAMINA ID
20 REM DISCO
30 REM PER SPROTEGGERE ERRORI
40 OPEN8,8,8,"#":OPEN15,8,15
50 POKE 53280,1:POKE53281,1:PRINT"s←"
51 PRINT"R                                        ┌Q"
52 PRINT"***************z SECTOR ID ←*************** QQ"
60 INPUT"←TRACCIA DA ESAMINARE ";TR:A$(T)="TR. "+STR$(TR)
70 INPUT"SETTORE DA ESAMINARE ";SC:A$(T)=A$(T)+" SC.
"+STR$(SC)
80 PRINT#15,"U1:";8;0;TR;SC
90 INPUT#15,A$,B$,C$,D$:IFVAL(A$)<>29ANDVAL(A$)>0THEN200
100 PRINT#15,"M-R"CHR$(22)CHR$(0)
110 GET#15,I1$
120 PRINT#15,"M-R"CHR$(23)CHR$(0)
130 GET#15,I2$:A$(T)=A$(T)+" ID="+I1$+I2$
140 PRINT"Q↑ID SULLA TRACCIA";TR;"SECT.";SC;"====> R";I1$;I2$
150 PRINT"QRz SPAZIO PER CONTINUARE - F1 PER FINIRE  ┌"
160 GETX$:IFX$=""THEN160
170 IFX$="e"THEN300
180 T=T+1:GOTO50
200 REM ****** ERROR ******
210 PRINT"Q£ERRORE┌ ";A$;B$;C$;D$
220 A$(T)="TR."+STR$(TR)+" SC."+STR$(SC)+" £R"+A$+" "+B$+"←┌"
230 GOTO150
300 REM
305 PRINT"s***************z SECTOR ID ←*************** QQ"
310 PRINT"←TRACCE E SETTORI ESAMINATIQQ"
320 FORQ=0TOT
330 PRINTA$(Q)
340 NEXT
```

R E C U P E R O F I L E

```
10 REM *** RECUPERO FILE ***
100 PRINT"s0 RRECUPERO FILE  r0"
130 REM=========================
140 DIMFE$(32),TF$(5):X$=" ":R$=" "
150 FORI=0TO5:READTF$(I):NEXTI:CC=0
160 K=0:CH=9:S=1:TR=18:T=TR:DR$="0"
170 IN$="RTIPO  NOME FILE      TR.       SET.LUNG."
180 PRINT"1 RECUPERO FILE CANCELLATOQ"
190 PRINT"2 RECUPERO DI TUTTI I FILE CANCELLATIQ"
200 PRINT"3 CONVALIDAQ"
210 PRINT"4 VISUALIZZAZIONE DIRECTORY AMPLIATAQ"
220 PRINT"5 CANCELLAZIONE FILEQ":PRINT"6 FINEQ"
230 PRINT"<SPACE> PER TORNARE AL MENUQ"
240 PRINT"R1:REC1;2:RECT;3:CONV;4:DIR;5:CANC;6:ENDr";
250 GETR$:IFR$=""THEN250
270 R=VAL(R$):PRINTTR$:RT=0
280 ONRGOTO450,960,1000,1500,1800,2000
290 END:REM=========================
295 REM ===== LEGGE,STAMPA DIRECTORY=
298 REM=============================
300 TT=ASC(FE$(1)):SS=ASC(FE$(2))
310 FT=ASC(FE$(0))AND127
320 IFFT>5THENPRINT"TIPO ERRATO":FT=5
330 TF$=TF$(FT)
340 LF=ASC(FE$(28))+256*ASC(FE$(29))
350 TT$=RIGHT$("  "+STR$(TT),3)
360 SS$=RIGHT$("  "+STR$(SS),3)
370 LF$=RIGHT$("     "+STR$(LF),5)
380 PRINT"R"TF$"r "NF$;TAB(20)
390 IFSPTHENPRINT#4," R"TF$"r "NF$;
400 PRINTTT$,SS$;LF$
410 IFSPTHENPRINT#4,TT$;SS$;LF$
420 RETURN
440 REM============================
450 PRINT"RRICERCA DI CERTO FILE"
455 REM============================
460 GOSUB750
```

```
470 PRINT"NOME DEL FILE DA RECUPERARE"
480 INPUTNO$:L=LEN(NO$):FORK=1TOL
490 IFMID$(NO$,K,1)="*"THENL=K-1:K=80
500 NEXTK:IFK<80THENL=16
510 NO$=LEFT$(NO$+"                ",L)
520 PRINT"STO CERCANDO         R"NO$
525 REM==============================
530 REM==RICERCA FILE NELLA DIRECT.==
535 REM==============================
540 T=TR:OPEN1,8,15:PRINT#1,"I"+DR$
550 S=1:OPENCH,8,CH,"#"+DR$
560 PRINT#1,"U1:"CH;DR;T;S
570 PRINT#1,"B-P:"CH,0:GET#CH,T$,S$
580 FORFE=1TO8
590 GETR$:IFR$="  "THEN710
600 FORCC=0TO31:GET#CH,X$
610 IFX$=""THENX$=CHR$(0)
620 FE$(CC)=X$:NEXT:NF$=""
630 FORK=3TO18:NF$=NF$+FE$(K):NEXT
640 IFNO$=LEFT$(NF$,L)THEN1200
650 NEXTFE
660 IFT$=""THENT$=CHR$(0)
670 IFS$=""THENS$=CHR$(0)
680 T=ASC(T$):S=ASC(S$)
690 IFT<>0THEN560
700 PRINT"RNON TROVATOr"NF$
710 CLOSECH:CLOSE1:CLOSE4:GOTO240
715 REM============================
720 REM =GESTIONE ERRORI DA DISCO=
725 REM============================
730 INPUT#1,CE$,ER$,ET$,ES$:IFCE$="00"THENRETURN
740 PRINTCE$,ER$,ET$,ES$,:GOTO710
745 REM========================
750 REM===== QUALE DRIVE ?=====
755 REM========================
760 PRINT"DRIVE    ";DR$"|||";:INPUTDR$
770 IFDR$<>"0"ANDDR$<>"1"THEN760
```

```
780 DR=VAL(DR$)
790 RETURN
795 REM=========================
800 REM===SCELTA TIPO FILE===
805 REM=========================
810 PRINT"R0=DEL;1=SEQ;2=PRG;3=USR;4=REL;5=>MENU"
820 INPUT"RTIPOr  2III";R:IFR<00R0>5THEN820
830 IFR=5THEN710
840 PO=2+32*(FE-1):TP=R+128
850 PRINT#1,"U1:"CH;DR;T;S
860 PRINT#1,"B-P:"CH;PO
870 PRINT#CH,CHR$(TP);
880 PRINT#1,"U2:"CH;DR;T;S
890 RETURN
895 REM==============================
900 REM=== IL FILE VA RECUPERATO ?===
905 REM==============================
910 IFMID$(NF$,1,1)=CHR$(0)THEN1420
920 INPUT"RDA RECUPERARE (S/N/END)r    NIII";R$
930 IFR$="S"THENGOSUB800:PRINT#1,"B-P:"CH;PO+32
940 IFLEFT$(R$,1)="E"THEN1420
959 RETURN
960 REM=========================
970 PRINT"RRECUPERO TUTTI I FILE"
980 RT=1:GOTO1510
990 REM=========================
1000 PRINT"RCONVALIDA  r";:GOSUB750
1010 CLOSE1:OPEN1,8,15:PRINT#1,"V"+DR$
1020 GOSUB720:GOTO710
1200 REM=========================
1210 PRINT"TROVATO IL FILE "NF$
1220 PRINTIN$
1230 GOSUB300
1330 PRINT"CHE TIPO ERA  ?"
1340 GOSUB800
1420 INPUT"CONVALIDA (S/N)   SIII";R$
1430 IFR$="S"THEN1000
```

```
1440 GOTO710
1450 REM=========================
1500 PRINT"RVISUALIZZAZIONE/STAMPA  DIRECTORY"
1505 REM=========================
1510 GOSUB750:INPUT "STAMPA   N|||";R$
1520 SP=0:IFR$="S"THENSP=-1:OPEN4,4
1530 GOSUB1900:PRINTIN$:S=1
1540 IFSPTHENPRINT#4,IN$
1550 IFSTTHENPRINT"STAMPANTE SPENTA":END
1560 OPEN1,8,15:PRINT#1,"I"+DR$
1570 OPENCH,8,CH,"#"+DR$
1571 GOSUB720
1580 PRINT#1,"U1:"CH;DR;T;S
1590 PRINT#1,"B-P:"CH;0:GET#CH,T$,S$
1600 FORFE=1TO8
1610 GETR$:IFR$="  "THEN1730
1620 FORCC=0TO31:GET#CH,X$
1630 IFX$=""THENX$=CHR$(0)
1640 FE$(CC)=X$:NEXT:NF$=""
1650 FORK=3TO18:NF$=NF$+FE$(K):NEXT
1660 GOSUB300:IFRTAND(FT=0)THENGOSUB900
1670 IFFE$(3)=CHR$(0)THEN1730
1680 NEXTFE
1690 IFT$=""THENT$=CHR$(0)
1700 IFS$=""THENS$=CHR$(0)
1710 T=ASC(T$):S=ASC(S$)
1720 IFT<>0THEN1580
1730 IFRTTHEN1420
1740 T=TR:GOTO710
1790 REM=========================
1800 PRINT"RCANCELLAZZIONE FILE":GOSUB750
1810 PRINT"NOME DEL FILE DA CANCELLARE"
1820 INPUTR$:CLOSE1:OPEN1,8,15
1830 PRINT#1,"S"+DR$+":"+R$
1840 GOSUB720:GOTO710
1890 REM=========================
1900 REM LEGGE NOME ,IDENTIF.DISCHETTO
```

```
1910 PO=6:IFTR=18THENPO=144
1920 S=0:OPEN1,8,15,"I"+DR$:GOSUB720
1930 OPENCH,8,CH,"#":PRINT#1,"U1:"CH;DR;TR;S
1940 PRINT#1,"B-P:";CH;PO:NF$=""
1950 FORK=1TO20:GET#CH,X$:NF$=NF$+X$:NEXT
1960 PRINTDR$"   "NF$:IFSPTHENPRINT#4,DR$"   "NF$
1970 CLOSECH:CLOSE1:RETURN
1980 DATADEL,SEQ,PRG,USR,REL,XXX
1990 REM=======================
2000 PRINT"RFINE":CLOSE1:CLOSECH:CLOSE4:END
JREADY.
J
```

D I S K S T A R T

```
1 REM ****DISK. START .64****
110 PRINT"DISK FILE LOG"
130 C$=CHR$(0)
140 DATA169,0,162,4,149,98,202,16,251
145 DATA169,160,133,97,162,2,32,198,255
150 DATA230,101,208,10,230,100,208,6,230,99
155 DATA208,2,230,98,32,228,255,165,144
160 DATA240,235,32,204,255,198,97
165 DATA6,101,38,100,38,99,38,98,16,244,96
170 DATA169,0,133,139,133,140
180 DATA230,139,208,2,230,140
190 DATA162,15,32,201,255,169,80,32,210,255
200 DATA169,4,32,210,255,165,139,32,210,255
205 DATA165,140,32,210,255
210 DATA169,1,32,210,255,32,204,255
215 DATA162,15,32,198,255,32,228,255
220 DATA72,32,204,255,104,201,48,240,200,96
230 FORJ=860TO977:READX:T=T+X:POKEJ,X:NEXTJ
240 IFT<>16312THENSTOP
250 DATA"XXX","SEQ","PRG","USR","REL"
260 FORJ=0TO4:READT$(J):NEXT
270 INPUT"PRINTER (Y/N)";Z$
280 Z=3:IFASC(Z$)=89THENZ=4:INPUT"DATE";D$
290 U=8:REM UNIT 8
300 D=0:REM DRIVE 0
330 OPEN4,Z:OPEN1,U,15,"I"+CHR$(D+48):CLOSE1
340 G$="                "
350 OPEN15,U,15
360 OPEN1,U,3,"$"+CHR$(D+48)
370 GET#1,A$:A=ASC(A$+" ")
380 IFA=1ORA=65THENL1=141:L2=89:GOTO410
390 IFA=67THENL1=3:L2=735:GOTO410
400 CLOSE1:PRINT"???":STOP
410 PRINT#4,"*** DISK LOG ***  ";D$
420 FORJ=1TOL1:GET#1,A$:NEXTJ
430 PRINT#4,"  ";:FORJ=1TO23:GET#1,A$:PRINT#4,A$;:NEXTJ
```

```
440 PRINT#4:       FORJ=1TOL2:GET#1,A$:              NEXTJ
450 M=M+1:GET#1,K$,T$,S$
460 L7=-1:Z$=CHR$(160):F$="":FORJ=1TO16:GET#1,A$
470 IFA$=Z$THENL7=0
480 IFL7THENF$=F$+A$
490 NEXTJ
500 GET#1,A$,A$,A$:L%=ASC(A$+C$)
510 FORJ=1TO6:GET#1,A$:NEXTJ
530 GET#1,A$:L =ASC(A$+C$)
550 GET#1,A$:L =L+256*ASC(A$+C$):IFM<8THENGET#1,A$,A$:GOTO570
560 M=0
570 SW=ST:IFK$=""GOTO820
580 K=ASC(K$)-128:IFK<10RK>4THENK=0
620 PRINT#4,T$(K)
630 PRINT#4,RIGHT$(" "+STR$(L),3);" ";
640 PRINT#4,LEFT$(F$+G$,17);
650 IFK=0GOTO810
660 IFK=4THENPRINT#4,"L=";MID$(STR$(L%),2);
670 OPEN2,U,4,CHR$(D+48)+":"+F$+","+T$(K)
680 A=0:IFK<>2GOTO730
690 GET#2,A$,B$:A=ASC(A$+C$)
700 B=ASC(B$+C$)
710 GOSUB840
730 IFK<>4GOTO760
740 SYS915:A=PEEK(139)+PEEK(140)*256-1
750 PRINT#4,",";MID$(STR$(A),2);"R";:GOTO800
760 POKE785,92:POKE786,3:A=A+USR(0)
770 IFK<>2THENPRINT#4,A;"BYTES";:GOTO800
780 PRINT#4,"  ";:A%=A/256:A=A-A%*256:B=B+A%
790 GOSUB840
800 CLOSE2
810 PRINT#4
820 IFSW=0GOTO450
830 INPUT#15,A:CLOSE1:PRINT#4,CHR$(13):CLOSE4:CLOSE15:END
840 X=B/16:GOSUB850:X=A/16
850 FORJ=1TO2:X%=X:X=(X-X%)*16:IFX%>9THENX%=X%+7
860 PRINT#4,CHR$(X%+48);:NEXTJ:RETURN
```

D I S K M E M O R Y D U M P

DISK MEMORY DUMP

IL DRIVE DEL COMMODORE 64 E' UNA PERIFERICA INTELLIGENTE,
IN QUANTO POSSIEDE UN PROPRIO SISTEMA OPERATIVO CON TANTO
DI MICROPROCESSORE DEDICATO (6502) E 2 KBYTS DI RAM.
IL PROGRAMMA CHE SEGUE SERVE A LEGGERE ED A STAMPARE IL
CONTENUTO DELLA MEMORIA ROM/RAM DEL DRIVE.

```
17000 REM******************
17010 REM DISK MEMORY DUMP
17020 REM******************
17021 CLR:PRINT"s":POKE53281,255:POKE53280,255:POKE646,9
17030 DEV=8
17040 OPEN15,DEV,15
17050 DEFFNHI(X)=INT((X-INT(X/65536)*65536)/256)
17060 DEFFNLO(X)=X-INT(X/256)*256
17070 INPUT" START ADRESS ";T$
17080 IF LEFT$(T$,1)="$"THENT$=MID$(T$,2):GOTO17140
17090 AD=VAL(T$)
17100 IFAD<>0THEN17170
17110 FORI=1TOLEN(T$):T1$=MID$(T$,I,1)
17120 IFT1$>"9"ORT1$<"0"THENPRINT"??":GOTO17070
17130 NEXT:GOTO17240
17140 GOSUB20000
17150 IF ERR THEN PRINT"??":GOTO17070
17160 AD=T
17170 IFAD>65535 OR AD<0THENPRINT"??":GOTO17070
17180 INPUT"OUTPUT TO PRINTER (Y/N)?N|||";T$
17190 OP=T$="Y"
17200 IF NOT OP AND T$<>"N"THENPRINT"??":GOTO17180
17210 T=3
17220 IF OP THEN T=4
17230 OPEN1,T
17240 PRINT"s"
17250 FORI=0TO20
17260 GOSUB18000
17270 PRINT#1,HE$
17280 NEXT
17290 INPUT"Q CONTINUE (Y/N) ? Y|||";T$
17300 IFT$="Y"THEN17240
17310 IFT$<>"N"THENPRINT"???":GOTO17290
17320 PRINT#15,"I0"
17330 CLOSE1
17340 CLOSE15
```

```
17350 END
18000 REM****************************
18010 REM READ 8 BYTES FROM DISK
18020 REM****************************
18030 O1$="":O2$=""
18040 T1$="":T2$=""
18045 PRINT#15,"M-R" CHR$(FNLO(AD)) CHR$(FNHI(AD)) CHR$(0)
18050 FORJ=0TO7
18070 GET#15,T$:T$=LEFT$(T$+CHR$(0),1)
18080 T=ASC(T$)
18090 GOSUB 19000
18100 T1$=T1$+RIGHT$("00"+HE$,2)+" "
18110 T3$="."
18120 IF T$>CHR$(31) AND T$<CHR$(128)THENT3$=T$
18130 T2$=T2$+T3$
18140 NEXT
18150 T=AD
18160 GOSUB 19000
18170 HE$=RIGHT$("0000"+HE$,4)+" "+T1$+" "+T2$
18180 AD=AD+8
18190 RETURN
19000 REM******************************
19010 REM CONVERT T TO HEX
19020 REM******************************
19030 HE$=""
19040 T1=T-INT(T/16)*16
19050 T=INT(T/16)
19060 HE$=CHR$(T1+48-(T1>9)*7)+HE$
19070 IFT>0THEN19040
19080 RETURN
20000 REM******************************
20010 REM CONVERT HEX TO DECIMAL
20020 REM******************************
20030 T=0:ERR=0
20040 FORI=1TOLEN(T$)
20050 T1=ASC(MID$(T$,I,1))-48
20060 IFT1>9THENT1=T1-7+(T1>22)*15
```

```
20070 T=T*16+T1
20080 ERR=T1>15ORT1<00RERR
20090 NEXT
20100 RETURN
```

D I S K I N I Z I O F I N E

DISK INIZIO-FINE

QUESTO PROGRAMMA NON SERVE A SPROTEGGERE ALTRI PROGRAMMI, MA SERVE AD INDIVIDUARE SUL DISCO LA LOCAZIONE DI INIZIO E LA LOCAZIONE DI FINE DI QUALSIASI PROGRAMMA.
CIO' RISULTA MOLTO UTILE IN MOLTE OCCASIONI COME GIA' ESAMINATO IN ALTRE PARTI DEL LIBRO.
PRESTATE PARTICOLARE CURA NELLA DIGITAZIONE DEL LISTATO, CHE E' TUTTO IN BASIC, IN QUANTO CONTIENE DEI COMANDI DIRETTI AL DOS DEL DRIVE, I QUALI, SE MAL DIGITATI, POTREBBERO PROVOCARE IL BLOCCO DEL SISTEMA.
PER OGNI PRECAUZIONE, E' BUONA NORMA REGISTRARE IL PROGRAMMA PRIMA DI MANDARLO IN RUN.

```
100 PRINT"sR          DISK-inizio/fine. 1.0          "
110 PRINT"NOME PROGRAMMA ";:INPUTPN$
120 OPEN15,8,15,"I0"
130 INPUT#15,A,A$,B,C:IFA<>0THEN1000
140 OPEN1,8,3,PN$+",P,R"
150 INPUT#15,A,A$,B,C
160 IFA=0THEN200
170 PRINT"PROGRAMM ";PN$;" NON TROVATO."
180 CLOSE1:CLOSE15:GOTO110
200 PRINT"sRDATI PER ";PN$;
210 FORI=POS(0)TO39:PRINT" ";:NEXT:PRINT
220 PRINT"QSTARDADRESSE:"
230 GET#1,A$:GET#1,B$
240 IFA$=""THENA$=CHR$(0)
250 IFB$=""THENB$=CHR$(0)
260 SA=ASC(A$)+256*ASC(B$)
270 PRINT"RDECIMAL:r";SA;:SH=SA:SA$=""
280 FORI=0TO3
290 SR=SH-INT(SH/16)*16
300 SR=SR+48
310 IFSR>57THENSR=SR+7
320 SA$=CHR$(SR)+SA$
330 SH=INT(SH/16)
340 NEXT
350 PRINTTAB(20);"RHEX:r ";SA$
360 CLOSE1
370 PRINT"QBLOCCHI OCCUPATI:";
380 OPEN1,8,0,"$:"+PN$
390 GET#1,A$,B$
400 GET#1,A$,B$
410 GET#1,A$,B$
420 GET#1,B$:IFST<>0THEN900
430 IFB$<>CHR$(34)THEN420
440 GET#1,B$:IFB$<>CHR$(34)THEN440
450 GET#1,B$:IFB$=CHR$(32)THEN450
460 GET#1,B$:IFB$<>""THEN460
```

```
470 GET#1,A$:GET#1,B$:GET#1,A$:GET#1,B$
480 IFA$=""THENA$=CHR$(0)
490 IFB$=""THENB$=CHR$(0)
500 AB=ASC(A$)+256*ASC(B$)
510 PRINTAB
520 CLOSE1
530 PRINT"Q ENDADRESSE:"
540 SA=SA+256*AB
550 PRINT"RDECIMAL:r";SA;:SH=SA:SA$=""
560 FORI=0TO3
570 SR=SH-INT(SH/16)*16
580 SR=SR+48
590 IFSR>57THENSR=SR+7
600 SA$=CHR$(SR)+SA$
610 SH=INT(SH/16)
620 NEXT
630 PRINTTAB(20);"RHEX:r ";SA$
900 CLOSE1:CLOSE15
999 END
1000 PRINT"QQRDISK-ERROR:";A;B;C:PRINTA$
1010 PRINT"QBITTE BEHEBEN UND PROGRAMM NEU STARTEN."
1020 CLOSE15:END
```

E S A M I N A E R R O R I

ESAMINA ERRORI

QUASTO PROGRAMMA SI INCARICA DI CERCARE E DI MOSTRARE SU VIDEO EVENTUALI ERRORI RESIDENTI TRA LE TRACCE DEL DISCO, AL FINE DI POTERLI RIPRODURRE IN SEGUITO CON IL PROGRAMMA APPOSITO OPPURE, MEGLIO ANCORA, PER, UNA VOLTA INDIVIDUATE LE TRACCE DOVE RISIEDONO QUESTI ERRORI, POTERLE RIPRODURRE PER INTERO CON IL PROGRAMMA COPIATRACCE.
IL PROGRAMMA E' INTERAMENTE IN BASIC, QUINDI NON PRESENTA PARTICOLARI DIFFICOLTA' NELLA DIGITAZIONE.
PER QUANTO RIGUARDA LE MODALITA' D' USO, POCO DA DIRE I COMMENTI CONTENUTI NEL PROGRAMMA STESSO SONO GIA' MOLTO ESPLICATIVI E DA SOLI GUIDANO MOLTO BENE AL CORRETTO USO DEL PROGRAMMA.

```
5 rem esamina errori
10 GOTO1000
50 PRINT#15,TS$S$;
60 PRINT#15,CM$CHR$(CM);
70 PRINT#15,RD$;:GET#15,E$:ER=ASC(E$+CHR$(0)):IFER>127THEN70
80 RETURN
100 FORNT=1TORT:CM=128:GOSUB50:IFER<2THENRETURN
110 NEXTNT:IFTTTHENRETURN
120 CM=192:GOSUB50:TT=NOTTT:GOTO100
1000 POKE53281,0:POKE53280,0:CLR
1010 OPEN15,8,15,"I0"
1020 INPUT#15,A:IFATHENSTOP
1030 OPEN2,8,2,"#0":Q$=CHR$(0)
1040 ID=16+6:GOTO2000
1100
MH=INT(MEM/256):ML=MEM-MH*256:PRINT#15,"M-R";CHR$(ML);CHR$(MH
)
1200 GET#15,A$:A=ASC(A$+0$):RETURN
1300 :
1400 PRINT"$";:NL=INT(BY/16):GOSUB1500:NL=BY-16*NL
1500 PRINTMID$("0123456789ABCDEF",NL+1,1);:RETURN
1600 :
2000 PRINT"sNzesamina disco"
2020 PRINT"@@@@@@@@@@@@@@@@@@Q"
2030 PRINT"QDI vITTORIO PAOLAQ"
2031 PRINT"                  QQ"
2050 PRINT"    *** menu ***"
2060 PRINT"QQ■-Q1. eSAME ID "
2070 PRINT"Q2. rICERCA VELOCE ERRORI"
2080 PRINT"Q3. rICERCA LENTA ERRORI"
2085 PRINT"Q4. rICERCA DI TUTTI GLI ERRORI"
2090 PRINT"Q5. fINE"
2110 PRINT"QINPUT (1-5):";
2220 OPEN1,0:INPUT#1,A$:CLOSE1:A=VAL(A$):IFA=0THEN2000
2230 IFA>5THEN2000
2240 ONAGOTO3000,4000,5000,5500,6000
3000 PRINT"sQID DISPLAYQQ"
```

```
3010 FORT=1TO35
3020 PRINT#15,"U1:2,0,";T;",0"
3030 MEM=ID:GOSUB1100:IZ=A:MEM=ID+1:GOSUB1100:
3040 PRINTRIGHT$(STR$(T),2);".
ID=";CHR$(34);CHR$(IZ);CHR$(A);CHR$(34);" ";
3050 BY=IZ:GOSUB1400:PRINT",";:BY=A:GOSUB1400
3060 PRINTTAB(20);:IFT/2=INT(T/2)THENPRINT
3070 NEXTT
3500 PRINT:PRINT"0PREMI RSPAZIOr PER MENU"
3510 GETA$:IFA$<>" "THEN3510
3520 GOTO1000
4000 PRINT"srICERCA ERRORIQ"
4010 FORT=1TO35
4020 PRINTRIGHT$(STR$(T),2);". ";
4030 PRINT#15,"U1:2,0,";T;",0"
4040
INPUT#15,A$,B$,C$,D$:DD$=A$+","+B$+","+C$+","+D$:PRINTDD$
4060 NEXTT
4070 GOTO 3500
5000 PRINT"srICERCA ERRORIQ"
5010 FORT=1TO35:FORS=0TO99
5030 PRINT#15,"U1:2,0,";T;","S
5040
INPUT#15,A$,B$,C$,D$:DD$=A$+","+B$+","+C$+","+D$:IFA$="66"THE
N5080
5041 PRINT"T="T;" I S="S;" I ";
5050 IFA$="00"THENPRINT"9":GOTO5070
5060 PRINTDD$
5070 GETA$:IFA$<>" "THEN5079
5071 GETA$:IFA$=""THEN5071
5079 NEXTS
5080 NEXTT
5100 GOTO3500
5500 PRINT"srICERCA eRRORIQ"
5510 PRINT" 1> 1541":PRINT" 2> 4040":PRINT"Q SELECT 1 OR 2 >
1 I";
5520
```

```
A$="":OPEN1,0:INPUT#1,A$:CLOSE1:A=VAL(A$):IFA<1ORA>2THEN5510
5525 PRINT
5530 IFA=1THENTL=6:TH=0:CL=0:CH=0:MT=40
5540 IFA=2THENTL=35:TH=16:CL=3:CH=16:MT=36
5541 INPUT"MIN TRK]]1|||";TS
5542 PRINT"         "MT"9":INPUT"MAX TRK";MT
5543 INPUT"MAX SEC]]0|||";LS
5544 INPUT"RETRIES]]4|||";RT
5545 INPUT"OK MSG ]]N|||";A$:OK$="9":IFA$="Y"THENOK$="00, OK"
5550 TS$="M-W"+CHR$(TL)+CHR$(TH)+CHR$(2)
5560 CM$="M-W"+CHR$(CL)+CHR$(CH)+CHR$(1)
5570 RD$="M-R"+CHR$(CL)+CHR$(CH)
5571 PRINT#15,"UA:2 0 18 0"
5580 FORT=TSTOMT:TT=0:T$=CHR$(T)
5590
MS=LS:IFMS=0THENMS=20:IFT>17THENMS=18:IFT>24THENMS=17:IFT>30T
HENMS=16
5600 FORS=0TOMS:S$=T$+CHR$(S)
5610 PRINT"T="T;" | S=";S;" | ";
5620 GOSUB100:REM TRY READ
5630
ONERGOTO5640,5650,5660,5670,5680,5690,5700,5710,5720,5730,574
0
5640 ER$=OK$:GOTO5750
5650 ER$="20,BLOCK HEADER NOT FOUND":GOTO5750
5660 ER$="21,NO SYNC CHARACTER":GOTO5750
5670 ER$="22,DATA BLOCK NOT PRESENT":GOTO5750
5680 ER$="23,CHECKSUM ERROR IN DATA BLOCK":GOTO5750
5690 ER$="24,BYTE DECODING ERROR":GOTO5750
5700 ER$="25,WRITE-VERIFY ERROR":GOTO5750
5710 ER$="26,WRITE PROTECTED":GOTO5750
5720 ER$="27,CHECKSUM ERROR IN HEADER":GOTO5750
5730 ER$="28,WRITE ERROR,LONG DATA BLOCK":GOTO5750
5740 ER$="29,DISK ID MISMATCH":GOTO5750
5750 PRINTER$
5760 GETA$:IFA$=""THEN5780
5761 IFA$="←"THENS=MS:T=MT:GOTO5780
```

```
5770 GETA$:IFA$=""THEN5770
5771 IFA$="←"THENS=MS:T=MT
5780 NEXTS
5790 NEXTT
5800 GOTO3500
6000 PRINT#15,"I0":CLOSE2:CLOSE15:PRINT"s":END
```

B A C K N I B B L E

C O P I A N A S T R O

PARTE IN BASIC DEL PROGRAMMA COPIANASTRO

```
1 POKE56,PEEK(44)+2:PRINT"sQPREPARE FOR
LOAD":WAIT653,1:SYSPEEK(44)*256+292:IFPEEK(144)AND48THEN4
2 IFPEEK(828)><1ANDPEEK(828)><3THENPRINT"QQNOT A PROGRAM
FILE.":END
3 PRINT"sQPREPARE FOR
SAVE":WAIT653,1:SYSPEEK(44)*256+383:GOTO5
4 PRINT"QR£LOAD ERRORr←":PRINT"QRUN AGAIN":END
5 PRINT"QSAVE AGAIN?":POKE198,0:WAIT198,1:IFPEEK(631)=89THEN3
7 IFPEEK(631)=78THEN1
8 END
```

DATI L.M PER PROGRAMMA COPIANASTRO

```
:0920 C1 A5 A9 C2 A9 C2 CD 53
:0928 C3 D0 2A A9 01 A2 01 A0
:0930 00 20 BA FF 20 C6 F8 AD
:0938 3D 03 85 C1 A5 38 85 C2
:0940 AD 40 03 18 65 38 38 ED
:0948 3E 03 85 AF AD 3F 03 85
:0950 AE 20 C9 F8 60 A9 01 A2
:0958 01 A0 00 20 BA FF 20 47
:0960 F8 AD 3D 03 85 C1 A5 38
:0968 85 C2 AD 40 03 18 65 38
:0970 38 ED 3E 03 85 AF AD 3F
:0978 03 85 AE 20 4A F8 60 A9
:0980 C2 CD 53 C3 D0 31 A9 01
:0988 A2 01 A0 01 20 BA FF 20
:0990 54 F8 A9 69 85 AB 20 EA
:0998 F8 AD 3D 03 85 C1 A5 38
:09A0 85 C2 AD 40 03 18 65 38
:09A8 38 ED 3E 03 85 AF AD 3F
:09B0 03 85 AE 20 E6 F8 60 A9
:09B8 01 A2 01 A0 01 20 BA FF
:09C0 20 D7 F7 A9 69 85 AB 20
:09C8 6B F8 AD 3D 03 85 C1 A5
:09D0 38 85 C2 AD 40 03 18 65
:09D8 38 38 ED 3E 03 85 AF AD
```

B A C K U P B L O C C H I

```
0 REM *** BACK-UP BLOCCHI ***
1 FORI=828TO883:READA:POKEI,A:NEXTI
10 REM"D=SAVE"BACK2",D0:?DS$:CATALOGD0
20 BB=PEEK(44)+27:POKE995,BB
30 POKE998,PEEK(55):POKE999,PEEK(56):POKE55,0:POKE56,BB:CLR
40 BB=PEEK(995)
50 N=PEEK(999)-BB-1:BA=BB*256:MA=828
60 DIMBM%(35,24)
70 FORJ=0TO7:TA(J)=2↑J:NEXT
80 PRINT"s]]]RBACKUP 1541r"
90 PRINT"Q'GOTO10000'IF PROGRAM QUITS ABNORMALLY"
100 PRINT"Q"N"BUFFERS AVAILABLE"
110 OPEN1,8,15
200 REM*** MAIN FUNCTIONS ***
210 GOSUB1000
220 D$="S":GOSUB3200:I2$=IR$
230 IFDR$<>"2A"THENPRINT"RILLEGAL DOS 1.0 DISKr":GOTO10000
240 IFI2$=I1$THENPRINT"RSOURCE AND DESTINATION HAVE SAME ID
CODEr":GOTO10000
250 GOSUB2500
260 T=TS:S=0:NU=1:T1=T:S1=S
270 PRINT#1,"I0":OPEN3,8,3,"#"
280 PRINT"READING BLOCK #";
290 IFBM%(T1,S1)=0THENGOSUB2000:NU=NU+1:IFNU>NTHEN320
300 S1=S1+1:IFS1>20THENS1=0:T1=T1+1
310 IFT1<TF+1THEN290
320 PRINT"Q"
330 CLOSE3
340 D$="D":GOSUB3200:IFIR$<>I1$THENGOTO340
350 PRINT#1,"I0":OPEN3,8,3,"#"
360 PRINT"WRITING BUFFER #";
370 NU=1:T1=T:S1=S
380 IFBM%(T1,S1)=0THENGOSUB2200:NU=NU+1:IFNU>NTHEN410
390 S1=S1+1:IFS1>20THENS1=0:T1=T1+1
400 IFT1<TF+1THEN380
410 PRINT"Q"
```

```
420 CLOSE3
430 S=S1+1:IFS>20THENS=0:T1=T1+1
440 T=T1:IFT>TFTHEN500
450 D$="S":GOSUB3200:IFIR$<>I2$THEN450
460 NU=1:T1=T:S1=S:GOTO270
500 REM FINISHED XFERS
510 CLOSE1
520 POKE55,PEEK(998):POKE56,PEEK(999):CLR
530 PRINT"QQBACKUP COMPLETE"
540 OPEN1,8,0,"$0"
550 GET#1,A$:IFA$<>"R"THEN550
560 PRINTA$;:GOTO610
570 GET#1,A$:SS=ST:A=LEN(A$):IFATHENA=ASC(A$)
580 GET#1,B$:SS=ST:B=LEN(B$):IFBTHENA=ASC(B$)
590 IFSSTHEN660
600 IFA=1ANDB=1THENGOSUB630
610 GET#1,A$:IFA$=""THENPRINT:GOTO570
620 PRINTA$;:GOTO610
630 GET#1,A$:SS=ST:A=LEN(A$):IFATHENA=ASC(A$)
640 GET#1,B$:SS=ST:B=LEN(B$):IFBTHENB=ASC(B$)
650 N=B*256+A:PRINTN;:RETURN
660 CLOSE1
670 END
1000 REM HEADER DEST DISK
1010 PRINT"QINSERT DESTINATION DISK TO BE FORMATTED"
1020 INPUT"QQDISK NAME]]]]]";DN$
1030 IFDN$=" "THENPRINT"999";:GOTO1020
1040 IFLEN(DN$)>16THENCLR:GOTO40
1050 F=0:FORJ=1TOLEN(DN$):S1$=MID$(DN$,J,1)
1060 IFS1$=" "ORS1$=CHR$(34)THENF=1
1070 NEXTJ:IFFTHENPRINT"999";:GOTO1020
1080 INPUT"QUNIQUE DISK ID";I1$
1090 IFI1$=" "THENPRINT"99";:GOTO1080
1100 IFLEN(I1$)<>2THENPRINT"99";:GOTO1080
1110 PRINT#1,"N0:"+DN$+","+I1$
1120 GOSUB3000
1130 IFERTHENPRINTER$:GOTO10000
```

```
1140 RETURN
2000 REM READ BLOCK T1,S1 TO BUFFER # NU
2010 C=.
2020 PRINT#1,"U1";3;0;T1;S1
2030 GOSUB3000:IFNOTERTHEN2060
2040 C=C+1:IFC<3GOTO2020
2050
PRINTER$:FORJ=(BB+NU)*256TO(BB+NU)*256+255:POKEJ,.:NEXTJ:GOTO
2100
2060 PRINT#1,"B-P";3;0
2070 IFNU<>0THENPRINT"    |||";RIGHT$("  "+STR$(NU),3);"|||";
2080 POKE996,PEEK(3):POKE997,PEEK(4):POKE4,BB+NU:SYSMA
2085 POKE3,PEEK(996):POKE4,PEEK(997)
2090 IFST<>.ANDST<>64THENGOSUB3000:GOTO2050
2100 RETURN
2200 REM WRITE BLOCK T1,S1 FROM BUFFER # NU
2210 C=.
2220 PRINT#1,"B-A";0;T1;S1:PRINT#1,"B-P";3;0
2230 PRINT"    ||||";RIGHT$("  "+STR$(NU),3);"|||";
2240 POKE996,PEEK(3):POKE997,PEEK(4):POKE4,BB+NU:SYSMA+3
2245 POKE3,PEEK(996):POKE4,PEEK(997)
2250 IFST<>.ANDST<>64THENPRINT"RIEEE WRITE
ERROR"ST"r":GOTO10000
2260 PRINT#1,"U2";3;0;T1;S1
2270 GOSUB3000:IFNOTERTHEN2300
2280 C=C+1:IFC<3THEN2260
2290 PRINT"RUNRECOVERABLE WRITE ERROR"ER$:GOTO10000
2300 RETURN
2500 REM GET BAM TO BM%(T,S)
2510 TS=1:TF=.
2520 PRINT#1,"I0":OPEN3,8,3,"#"
2530 S9=0
2540 PRINT"QTRACK #  BLOCKS TO XFER"
2550 PRINT"##########################"
2560 NU=0:T1=18:S1=0:C0$=CHR$(.):GOSUB2000
2570 BY=4
2580 T%=(BY-4)/4+1
```

```
2590 PRINT"  ";T%;
2600
IFPEEK(BA+BY)=.THENFORJ=.TO20:BM%(T%,J)=.:NEXT:BY=BY+4:GOTO26
50
2610 S=0
2620
BY=BY+1:A0=PEEK(BA+BY):FORJ=.TO7:BM%(T%,S)=A0ANDTA(J):S=S+1:N
EXT
2630 IFS<22THEN2620
2640 BY=BY+1
2650 ES=21:IFT%>17THENES=19
2660 IFT%>24THENES=18
2670 IFT%>30THENES=17
2680 FORJ=ESTO24:BM%(T%,J)=-1:NEXT
2690 SM=.:FORJ=.TO20:IFBM%(T%,J)=.THENSM=SM+1
2700 NEXT:PRINTTAB(12);SM:S9=S9+SM
2710 IFSM=.ANDTS=T%THENTS=TS+1:GOTO2730
2720 IFSM<>.THENTF=T%
2730 IFBY<143THEN2580
2740 CLOSE3
2750 PRINT"START =";TS;" FINISH =";TF
2760 PRINT"QA TOTAL OF";S9;"BLOCKS TO XFER"
2770 S8=90+25+(.650+.980)*S9
2780 S7=INT(S8/60):PRINT"APPROX";S7":"INT(S8-S7*60);"FOR
COPY"
2790 RETURN
3000 REM READ ERR CH TO ER,ER$
3010 INPUT#1,E0$,E1$,E2$,E3$:ER$=E0$+","+E1$+","+E2$+","+E3$
3020 ER=LEN(E0$):IFERTHENER=VAL(E0$)
3030 RETURN
3200 REM INSTRUCT TO SWAP TO DISK GIVEN IN D$
3210 IFD$="D"THENS1$="DESTINATION":GOTO3230
3220 S1$="SOURCE"
3230 PRINT"QINSERT ";S1$;" DISK, PRESS R r"
3240 GETA$:IFA$<>" "THEN3240
3250 OPEN2,8,0,"$0"
3260 GOSUB3000:IFER>0THEN10000
```

```
3270 FORJ=1TO26:GET#2,A$:NEXTJ
3280 GET#2,A$:GET#2,B$:IR$=A$+B$
3290 GET#2,A$:GET#2,A$:GET#2,B$:DR$=A$+B$
3300 CLOSE2:RETURN
10000 REM DROP OUT
10010 POKE55,PEEK(998):POKE56,PEEK(999):CLR:STOP
15000
DATA76,66,3,76,91,3,162,3,32,198,255,160,0,132,3,32,207,255,1
45
15010
DATA3,165,144,208,3,200,208,244,32,204,255,96,162,3,32,201,25
5,160
15020
DATA0,132,3,177,3,32,210,255,165,144,208,3,200,208,244,32,204
,255,96
```

MONITOR C000

```
:C000 4C 0C C0 4C 3F C0 4C CF
:C008 FF 4C D2 FF 78 AD 18 03
:C010 85 11 AD 19 03 85 12 AD
:C018 F2 CE AE F3 CE 8D 16 03
:C020 8E 17 03 20 19 CD A9 00
:C028 85 F9 85 FA A9 80 20 90
:C030 FF A9 FF 85 1A A9 00 85
:C038 13 85 10 85 31 58 00 20
:C040 F9 CC 68 85 07 68 85 06
:C048 68 85 05 68 85 04 68 85
:C050 03 68 85 02 BA 86 08 38
:C058 A5 03 E9 02 85 03 A5 02
:C060 E9 00 85 02 20 19 CD A5
:C068 1A C9 FF F0 1D A6 14 86
:C070 C1 A6 15 86 C2 A0 00 20
:C078 AB CD A5 2D D0 06 A5 2E
:C080 D0 05 F0 06 4C 94 C6 4C
:C088 92 C6 20 7C C9 A2 42 A9
:C090 2A 20 64 C8 4C A8 C9 A9
:C098 3F 20 09 C0 20 7C C9 A9
:C0A0 2E 20 09 C0 A9 00 85 0F
:C0A8 85 29 85 31 20 DB C8 C9
:C0B0 2E F0 F9 C9 20 F0 F5 A2
:C0B8 14 DD AD CE D0 12 85 1E
:C0C0 8A 0A AA BD C2 CE 85 C1
:C0C8 BD C3 CE 85 C2 6C C1 00
:C0D0 CA 10 E6 30 C2 A2 02 D0
:C0D8 02 A2 00 B4 C1 D0 08 B4
:C0E0 C2 D0 02 E6 29 D6 C2 D6
:C0E8 C1 60 A9 00 85 23 20 5B
:C0F0 C2 A2 02 20 79 C9 CA D0
:C0F8 FA 60 A2 02 B5 C0 48 B5
:C100 26 95 C0 68 95 26 CA D0
:C108 F3 60 A5 27 A4 28 4C 15
:C110 C1 A5 C3 A4 C4 38 E5 C1
```

```
:C118 85 26 98 E5 C2 A8 05 26
:C120 60 A9 00 F0 02 A9 01 85
:C128 2A 20 1D C8 20 7C C9 20
:C130 11 C1 20 77 C8 90 1A 20
:C138 0A C1 B0 03 4C C1 C1 20
:C140 77 C1 E6 C3 D0 02 E6 C4
:C148 20 63 C9 A4 29 D0 49 F0
:C150 E6 20 0A C1 18 A5 26 65
:C158 C3 85 C3 98 65 C4 85 C4
:C160 20 FA C0 20 77 C1 20 0A
:C168 C1 B0 56 20 D5 C0 20 D9
:C170 C0 A4 29 D0 23 F0 EC A2
:C178 00 20 49 CD 48 20 45 CD
:C180 68 A4 2A F0 03 20 53 CD
:C188 C1 0D F0 0B 20 4E C8 20
:C190 79 C9 20 E5 C8 F0 01 60
:C198 4C 9C C0 20 36 C8 20 40
:C1A0 C8 20 DB C8 20 A9 C8 90
:C1A8 15 85 20 A6 29 D0 12 20
:C1B0 11 C1 90 0D A5 20 20 57
:C1B8 CD 20 63 C9 D0 ED 4C 97
:C1C0 C0 4C 9C C0 20 36 C8 20
:C1C8 40 C8 20 DB C8 A2 00 20
:C1D0 DB C8 C9 27 D0 13 20 DB
:C1D8 C8 95 33 E8 20 06 C0 C9
:C1E0 0D F0 20 E0 20 D0 F2 F0
:C1E8 1A 86 2C 20 B0 C8 90 CE
:C1F0 95 33 E8 20 06 C0 C9 0D
:C1F8 F0 09 20 A9 C8 90 BF E0
:C200 20 D0 ED 86 1F 20 7C C9
:C208 A2 00 A0 00 A5 C1 48 A5
:C210 C2 48 20 7C CD D5 33 D0
:C218 13 20 63 C9 E8 E4 1F D0
:C220 F1 68 85 C2 68 85 C1 20
:C228 8C C1 D0 06 68 85 C2 68
:C230 85 C1 20 63 C9 A4 29 D0
```

```
:C238 05 20 11 C1 B0 CA 4C 9C
:C240 C0 20 38 C4 20 11 C1 90
:C248 0D A0 2C 20 EA C0 20 B5
:C250 C2 20 E5 C8 D0 EE 20 97
:C258 C5 D0 E3 20 6E C9 20 4E
:C260 C8 20 79 C9 20 1B CC 48
:C268 20 CC C2 68 A2 03 20 52
:C270 C3 A2 06 E0 03 D0 13 A4
:C278 22 F0 0F A5 2B C9 E8 20
:C280 7C CD B0 1C 20 AD C2 88
:C288 D0 F1 06 2B 90 0E BD 0A
:C290 CE 20 92 C5 BD 10 CE F0
:C298 03 20 92 C5 CA D0 D4 60
:C2A0 20 C0 C2 AA E8 D0 01 C8
:C2A8 98 20 AD C2 8A 86 1F 20
:C2B0 55 C8 A6 1F 60 A5 22 20
:C2B8 BF C2 85 C1 84 C2 60 38
:C2C0 A4 C2 AA 10 01 88 65 C1
:C2C8 90 01 C8 60 A4 22 C8 98
:C2D0 48 20 2A C9 68 85 21 38
:C2D8 A5 C1 E5 21 85 C1 B0 02
:C2E0 C6 C2 A5 22 18 65 22 65
:C2E8 22 A8 A9 20 20 09 C0 C8
:C2F0 C0 07 D0 F6 60 A8 4A 90
:C2F8 0B 4A B0 17 C9 22 F0 13
:C300 29 07 09 80 4A AA BD B9
:C308 CD B0 04 4A 4A 4A 4A 29
:C310 0F D0 04 A0 80 A9 00 AA
:C318 BD FD CD 85 2B 29 03 85
:C320 22 98 29 8F AA 98 A0 03
:C328 E0 8A F0 0B 4A 90 08 4A
:C330 4A 09 20 88 D0 FA C8 88
:C338 D0 F2 60 20 7C CD 20 AD
:C340 C2 A2 01 20 F3 C0 C4 22
:C348 C8 90 F0 A2 03 C0 03 90
:C350 F2 60 A8 B9 17 CE 85 27
```

```
:C358 B9 57 CE 85 28 A9 00 A0
:C360 05 06 28 26 27 2A 88 D0
:C368 F8 69 3F 20 09 C0 CA D0
:C370 EC 4C 79 C9 AD F5 CE 48
:C378 AD F4 CE 48 A9 00 48 48
:C380 48 48 6C 18 00 85 20 48
:C388 20 DB C8 20 3F C9 D0 F8
:C390 68 49 FF 4C B7 C2 A9 40
:C398 2C A9 00 85 20 20 38 C4
:C3A0 A6 29 D0 17 20 11 C1 90
:C3A8 12 24 20 50 06 20 BE C3
:C3B0 4C B6 C3 20 0E C4 20 E5
:C3B8 C8 D0 E5 4C 56 C2 20 7C
:C3C0 C9 A2 2E A9 27 20 64 C8
:C3C8 20 79 C9 20 4E C8 A9 12
:C3D0 20 09 C0 A0 08 A2 00 20
:C3D8 49 CD 29 7F C9 20 B0 02
:C3E0 A9 2E 20 09 C0 C9 22 D0
:C3E8 0A A9 14 20 09 C0 A9 22
:C3F0 20 09 C0 20 63 C9 88 D0
:C3F8 DE A9 92 20 09 C0 A5 C1
:C400 38 E9 08 85 C1 B0 02 C6
:C408 C2 A9 08 4C 2A C9 20 7C
:C410 C9 A2 2E A9 3A 20 64 C8
:C418 20 4E C8 A9 08 4C 2A C9
:C420 20 36 C8 A9 08 20 85 C3
:C428 20 97 C5 20 0E C4 A9 3A
:C430 8D 77 02 A9 00 4C 47 C5
:C438 20 36 C8 85 C3 86 C4 20
:C440 06 C0 C9 0D F0 03 20 3B
:C448 C8 4C 7C C9 20 87 C8 85
:C450 C3 86 C4 A2 00 86 34 20
:C458 DB C8 C9 20 F0 F5 95 24
:C460 E8 E0 03 D0 F2 CA 30 11
:C468 B5 24 38 E9 3F A0 05 4A
:C470 66 34 66 33 88 D0 F8 F0
```

```
:C478 EC A2 02 20 06 C0 C9 0D
:C480 F0 20 C9 3A F0 1C C9 20
:C488 F0 F1 20 89 C5 B0 0E 20
:C490 BD C8 A4 C1 84 C2 85 C1
:C498 A9 30 95 33 E8 95 33 E8
:C4A0 D0 D9 86 27 A2 00 86 29
:C4A8 A2 00 86 20 A5 29 20 F5
:C4B0 C2 A6 2B 86 28 AA BD 57
:C4B8 CE 20 6F C5 BD 17 CE 20
:C4C0 6F C5 A2 06 E0 03 D0 12
:C4C8 A4 22 F0 0E A5 2B C9 E8
:C4D0 A9 30 B0 1D 20 6C C5 88
:C4D8 D0 F2 06 2B 90 0E BD 0A
:C4E0 CE 20 6F C5 BD 10 CE F0
:C4E8 03 20 6F C5 CA D0 D5 F0
:C4F0 06 20 6C C5 20 6C C5 A5
:C4F8 27 C5 20 F0 03 4C 79 C5
:C500 20 77 C8 A4 22 F0 2B A5
:C508 28 C9 9D D0 1C 20 11 C1
:C510 90 09 98 D0 6B A6 26 30
:C518 67 10 07 C8 D0 62 A6 26
:C520 10 5E CA CA 8A A4 22 D0
:C528 03 B9 C2 00 20 AB CD 88
:C530 D0 F7 A5 29 20 AB CD A0
:C538 41 8C 77 02 20 97 C5 20
:C540 EA C0 20 B5 C2 A9 20 8D
:C548 78 02 A5 C2 20 9C C5 8E
:C550 79 02 8D 7A 02 A5 C1 20
:C558 9C C5 8E 7B 02 8D 7C 02
:C560 A9 20 8D 7D 02 A9 07 85
:C568 C6 4C 9C C0 20 6F C5 86
:C570 1F A6 20 D5 33 F0 0C 68
:C578 68 E6 29 F0 03 4C A8 C4
:C580 4C 97 C0 E8 86 20 A6 1F
:C588 60 C9 30 90 03 C9 47 60
:C590 38 60 C5 23 D0 03 60 A9
:C598 91 4C 09 C0 48 4A 4A 4A
```

```
:C5A0 4A 20 6D C8 AA 68 29 0F
:C5A8 4C 6D C8 85 05 08 68 29
:C5B0 EF 85 04 86 06 84 07 68
:C5B8 18 69 01 85 03 68 69 00
:C5C0 85 02 A9 80 85 10 D0 3F
:C5C8 48 8A 48 98 48 BA BD 04
:C5D0 01 29 04 D0 01 58 A0 32
:C5D8 B1 F9 8D AA AA EA 78 20
:C5E0 F9 CC 68 85 07 68 85 06
:C5E8 68 85 05 D8 68 85 04 68
:C5F0 85 03 68 85 02 A5 11 8D
:C5F8 18 03 A5 12 8D 19 03 20
:C600 19 CD A9 20 8D AA AA BA
:C608 86 08 AD AA AA 8D AA AA
:C610 58 24 13 10 1F A5 14 85
:C618 C1 A5 15 85 C2 A0 00 20
:C620 7C CD 85 1A A9 00 20 AB
:C628 CD 24 13 50 05 85 13 4C
:C630 D9 C6 85 13 24 10 50 18
:C638 A5 02 C5 15 D0 58 A5 03
:C640 C5 14 D0 52 A5 2D D0 4C
:C648 A5 2E D0 46 A9 80 85 10
:C650 30 15 46 10 B0 03 4C 8A
:C658 C0 A6 08 9A AD F7 CE 48
:C660 AD F6 CE 48 4C 33 C7 20
:C668 7C C9 A5 03 A6 02 85 C1
:C670 86 C2 20 79 C9 A9 24 85
:C678 23 20 5E C2 20 E4 FF F0
:C680 FB C9 03 D0 03 4C 9C C0
:C688 C9 4A D0 4D A9 01 85 10
:C690 D0 47 C6 2E C6 2D AD 00
:C698 DC C9 FA D0 3C A2 53 4C
:C6A0 8F C0 A9 00 F0 06 A9 40
:C6A8 D0 02 A9 80 85 10 85 31
:C6B0 A4 16 84 2D A4 17 84 2E
:C6B8 20 06 C0 C9 0D F0 17 C9
```

```
:C6C0 20 F0 03 4C 67 C7 20 9A
:C6C8 C8 20 25 C9 20 06 C0 C9
:C6D0 0D F0 03 4C 67 C7 20 7C
:C6D8 C9 A5 1A C9 FF F0 35 A6
:C6E0 14 E4 03 D0 1D A4 15 C4
:C6E8 02 D0 17 84 C2 86 C1 A6
:C6F0 10 D0 25 A0 00 20 AB CD
:C6F8 A9 80 85 13 A9 C0 85 13
:C700 D0 16 A5 10 D0 12 86 C1
:C708 A4 15 84 C2 A0 00 A9 00
:C710 A8 20 AB CD A5 10 F0 17
:C718 78 A9 A0 8D AA AA A9 00
:C720 8D AA AA AD F1 CE 8D 19
:C728 03 AD F0 CE 8D 18 03 78
:C730 A6 08 9A A5 02 48 A5 03
:C738 48 A5 04 48 A5 05 48 A6
:C740 06 A4 07 A5 10 05 13 08
:C748 AD AA AA 85 32 20 3A CD
:C750 20 F9 CC 28 F0 0F A9 7F
:C758 8D AA AA A9 09 8D 06 DD
:C760 A9 00 8D 07 DD 68 40 4C
:C768 97 C0 20 87 C8 85 C1 86
:C770 C2 85 14 86 15 A0 00 20
:C778 7C CD 85 1A 98 A9 00 85
:C780 16 85 17 20 97 C8 85 16
:C788 86 17 4C 9C C0 20 1D C8
:C790 85 2F 86 30 20 97 C8 85
:C798 24 86 25 20 97 C8 85 AE
:C7A0 86 AF 20 06 C0 C9 0D F0
:C7A8 09 20 06 C0 C9 57 D0 02
:C7B0 E6 23 20 77 C8 A6 29 D0
:C7B8 18 20 0A C1 90 13 A4 23
:C7C0 D0 1A 20 7C CD 20 F5 C2
:C7C8 AA BD 17 CE D0 06 20 EA
:C7D0 C0 4C 9C C0 A4 22 C0 02
:C7D8 D0 3B F0 02 84 22 88 38
:C7E0 20 7C CD AA E5 24 C8 20
```

```
:C7E8 7C CD E5 25 90 27 88 20 .\.%■'■
:C7E8 7C CD E5 25 90 27 88 20 .\.%■'■
:C7F0 7C CD 85 0B A5 AE E5 0B .\■■ ¬.■
:C7F8 C8 20 7C CD 85 0B A5 AF  ■ .\■■ ▁
:C800 E5 0B 90 11 88 18 8A 65 .■■■■■■.
:C808 2F 20 AB CD C8 20 7C CD / ⊢\■ .\
:C810 65 30 20 AB CD 20 63 C9 .0 ⊢\ .\
:C818 88 10 FA 30 98 20 87 C8 ■■.0■ ■■
:C820 85 C3 86 C4 20 97 C8 85 ■─■─ ■■■
:C828 27 86 28 20 DB C8 20 9A '■( +■ ■
:C830 C8 85 C1 86 C2 60 20 87  ■▲■■. ■
:C838 C8 B0 F6 20 9A C8 B0 03 ■r. ■■■r■
:C840 20 97 C8 85 C3 86 C4 60  ■■■─■─.
:C848 20 FA C0 4C 97 C8 A5 C2  .─L■■ ■
:C850 20 55 C8 A5 C1 48 4A 4A  U ■ ▲HJJ
:C858 4A 4A 20 6D C8 AA 68 29 JJ .■■.)
:C860 0F 20 6D C8 48 8A 20 09 ■■ .■H■ ■■
:C868 C0 68 4C 09 C0 18 69 F6 ─.L■■─■..
:C870 90 02 69 06 69 3A 60 A2 ■■■.■■.:.■
:C878 02 B5 C0 48 B5 C2 95 C0 ■■ ─H■ ■■─
:C880 68 95 C2 CA D0 F3 60 A9 .■■ ¬■..◤
:C888 00 85 2C 20 DB C8 C9 20 ■■■, +■\
:C890 F0 F9 20 BD C8 B0 08 20 .. ┘■r■■
:C898 DB C8 20 A9 C8 90 07 AA +■ ◤■■■■
:C8A0 20 A9 C8 90 01 60 4C 97  ◤■■■.L■
:C8A8 C0 A9 00 85 2C 20 DB C8 ─◤■■, +■
:C8B0 C9 20 D0 09 20 DB C8 C9 ¬ ■■ +■\
:C8B8 20 D0 0E 18 60 20 D0 C8  ■■■. ■■
:C8C0 0A 0A 0A 0A 85 2C 20 DB ■■■■■, +
:C8C8 C8 20 D0 C8 05 2C 38 60 ■ ■■■,8.
:C8D0 C9 3A 08 29 0F 28 90 02 ¬:■)■(■■
:C8D8 69 08 60 20 06 C0 C9 0D .■■. ■─¬■
:C8E0 D0 F8 4C 9C C0 A5 91 C9 ■.L■─■ ■\
:C8E8 FE D0 07 08 20 CC FF 85 .■■■■ └.■
:C8F0 C6 28 60 20 FC C8 2C AA ─(. .■/ ■
:C8F8 AA 30 F8 60 20 E5 C8 D0  ■0.. .■■
```

```
:C7E8 7C CD E5 25 90 27 88 20
:C7F0 7C CD 85 0B A5 AE E5 0B
:C7F8 C8 20 7C CD 85 0B A5 AF
:C800 E5 0B 90 11 88 18 8A 65
:C808 2F 20 AB CD C8 20 7C CD
:C810 65 30 20 AB CD 20 63 C9
:C818 88 10 FA 30 98 20 87 C8
:C820 85 C3 86 C4 20 97 C8 85
:C828 27 86 28 20 DB C8 20 9A
:C830 C8 85 C1 86 C2 60 20 87
:C838 C8 B0 F6 20 9A C8 B0 03
:C840 20 97 C8 85 C3 86 C4 60
:C848 20 FA C0 4C 97 C8 A5 C2
:C850 20 55 C8 A5 C1 48 4A 4A
:C858 4A 4A 20 6D C8 AA 68 29
:C860 0F 20 6D C8 48 8A 20 09
:C868 C0 68 4C 09 C0 18 69 F6
:C870 90 02 69 06 69 3A 60 A2
:C878 02 B5 C0 48 B5 C2 95 C0
:C880 68 95 C2 CA D0 F3 60 A9
:C888 00 85 2C 20 DB C8 C9 20
:C890 F0 F9 20 BD C8 B0 08 20
:C898 DB C8 20 A9 C8 90 07 AA
:C8A0 20 A9 C8 90 01 60 4C 97
:C8A8 C0 A9 00 85 2C 20 DB C8
:C8B0 C9 20 D0 09 20 DB C8 C9
:C8B8 20 D0 0E 18 60 20 D0 C8
:C8C0 0A 0A 0A 0A 85 2C 20 DB
:C8C8 C8 20 D0 C8 05 2C 38 60
:C8D0 C9 3A 08 29 0F 28 90 02
:C8D8 69 08 60 20 06 C0 C9 0D
:C8E0 D0 F8 4C 9C C0 A5 91 C9
:C8E8 FE D0 07 08 20 CC FF 85
:C8F0 C6 28 60 20 FC C8 2C AA
:C8F8 AA 30 F8 60 20 E5 C8 D0
```

```
:C900 0B 20 0D C9 A9 03 85 9A
:C908 A9 00 85 99 60 08 78 AD
:C910 AA AA 09 02 8D AA AA AD
:C918 EE CE 8D 14 03 AD EF CE
:C920 8D 15 03 28 60 85 03 86
:C928 02 60 85 21 A0 00 20 79
:C930 C9 20 7C CD 20 55 C8 20
:C938 63 C9 C6 21 D0 F0 60 20
:C940 A9 C8 90 0C A2 00 20 57
:C948 CD C1 0D F0 03 4C 97 C0
:C950 20 63 C9 C6 20 60 A9 04
:C958 85 C1 A9 00 85 C2 A9 05
:C960 E6 0F 60 E6 C1 D0 06 E6
:C968 C2 D0 02 E6 29 60 98 48
:C970 20 7C C9 68 A2 2E 20 64
:C978 C8 A9 20 2C A9 0D 4C 09
:C980 C0 A5 C8 C9 02 90 21 20
:C988 06 C0 C9 42 D0 1A A5 1A
:C990 C9 FF F0 14 A6 14 A4 15
:C998 86 C1 84 C2 A0 00 20 AB
:C9A0 CD A9 FF 85 1A 4C 9C C0
:C9A8 A2 00 BD 97 CE 20 09 C0
:C9B0 E8 E0 16 D0 F5 20 7C C9
:C9B8 A2 2E A9 3B 20 64 C8 A5
:C9C0 02 20 55 C8 A5 03 20 55
:C9C8 C8 20 56 C9 20 2A C9 C6
:C9D0 0F 4C 9C C0 4C 97 C0 20
:C9D8 87 C8 20 25 C9 20 56 C9
:C9E0 85 20 20 DB C8 20 3F C9
:C9E8 D0 F8 C6 0F F0 E3 78 20
:C9F0 3A CD A6 08 9A 20 F9 CC
:C9F8 6C 02 A0 4C 97 C0 A0 01
:CA00 84 BA 84 B9 88 84 B7 84
:CA08 90 84 93 A9 00 85 BC A9
:CA10 33 85 BB 20 06 C0 C9 20
:CA18 F0 F9 C9 0D F0 1A C9 22
```

```
:CA20 D0 D9 20 06 C0 C9 22 F0
:CA28 26 C9 0D F0 0B 91 BB E6
:CA30 B7 C8 C0 10 F0 C5 D0 EA
:CA38 A5 1E C9 4C D0 E2 A9 00
:CA40 20 D5 FF 20 F3 C8 A5 90
:CA48 29 10 D0 F0 4C 9C C0 20
:CA50 06 C0 C9 0D F0 E2 C9 2C
:CA58 D0 F0 20 A9 C8 29 0F F0
:CA60 D3 C9 03 F0 FA 85 BA 20
:CA68 06 C0 C9 0D F0 CA C9 2C
:CA70 D0 E6 20 2E C8 20 06 C0
:CA78 C9 2C D0 F4 20 9A C8 85
:CA80 AE 86 AF 20 06 C0 C9 20
:CA88 F0 F9 C9 0D D0 EC A5 1E
:CA90 C9 53 D0 F8 20 7C C9 A9
:CA98 00 85 B9 A6 AE A4 AF A9
:CAA0 C1 20 D8 FF 4C 9C C0 A5
:CAA8 31 D0 04 A5 C6 D0 06 68
:CAB0 A8 68 AA 68 40 AD 77 02
:CAB8 C9 11 F0 03 4C 55 CB A5
:CAC0 D6 C9 18 D0 EA A5 D1 85
:CAC8 C3 A5 D2 85 C4 A9 19 85
:CAD0 2D A0 01 20 B3 CC C9 3A
:CAD8 F0 19 C9 2C F0 15 C9 27
:CAE0 F0 11 C6 2D F0 C9 38 A5
:CAE8 C3 E9 28 85 C3 B0 E2 C6
:CAF0 C4 D0 DE 85 1E 20 6F CC
:CAF8 B0 B5 A5 1E C9 3A F0 15
:CB00 C9 27 D0 31 18 A9 08 65
:CB08 C1 85 C1 90 02 E6 C2 20
:CB10 BE C3 4C 44 CB 18 A5 C1
:CB18 69 08 85 C1 90 02 E6 C2
:CB20 20 7C C9 A2 2E A9 3A 20
:CB28 64 C8 20 4E C8 A9 08 20
:CB30 2A C9 4C 44 CB 20 1B CC
:CB38 20 B5 C2 A9 00 85 23 A0
```

```
:CB40 2C 20 5B C2 A9 00 85 C6
:CB48 85 31 85 29 20 97 C5 20
:CB50 7C C9 4C AF CA C9 91 F0
:CB58 03 4C AF CA A5 D6 F0 03
:CB60 4C AF CA A5 D1 85 C3 A5
:CB68 D2 85 C4 A9 19 85 2D A0
:CB70 01 20 B3 CC C9 3A F0 19
:CB78 C9 2C F0 15 C9 27 F0 11
:CB80 C6 2D F0 14 18 A5 C3 69
:CB88 28 85 C3 90 E2 E6 C4 D0
:CB90 DE 85 1E 20 6F CC 90 03
:CB98 4C AF CA A5 1E C9 3A F0
:CBA0 18 C9 27 D0 33 20 23 CC
:CBA8 38 A5 C1 E9 08 85 C1 B0
:CBB0 02 C6 C2 20 C1 C3 4C CA
:CBB8 CB 20 23 CC 38 A5 C1 E9
:CBC0 08 85 C1 B0 02 C6 C2 20
:CBC8 11 C4 A9 00 85 C6 85 31
:CBD0 85 29 20 6A CC 4C AF CA
:CBD8 20 23 CC A5 C1 A6 C2 85
:CBE0 C3 86 C4 A9 10 85 2D 38
:CBE8 A5 C3 E5 2D 85 C1 A5 C4
:CBF0 E9 00 85 C2 20 1B CC 20
:CBF8 B5 C2 20 11 C1 F0 06 B0
:CC00 F3 C6 2D D0 E2 E6 22 A5
:CC08 22 20 91 C3 A2 00 86 23
:CC10 A9 2C 20 74 C9 20 5E C2
:CC18 4C CA CB A2 00 20 49 CD
:CC20 4C F5 C2 A9 00 85 AE A9
:CC28 DB 85 AF 85 B1 A9 07 85
:CC30 AD 85 C4 A9 00 85 AC A9
:CC38 28 85 C3 85 B0 A0 BF A2
:CC40 04 B1 AC 91 C3 B1 AE 91
:CC48 B0 88 D0 F5 B1 AC 91 C3
:CC50 B1 AE 91 B0 88 C6 AD C6
:CC58 C4 C6 AF C6 B1 CA D0 E1
```

```
:CC68 10 FA A9 13 4C 09 C0 C0
:CC70 28 D0 02 38 60 20 B3 CC
:CC78 C9 20 F0 F3 88 20 9E CC
:CC80 AA 20 9E CC 85 C1 86 C2
:CC88 A9 FF 85 31 85 CC A5 CF
:CC90 F0 0A A5 CE A4 D3 91 D1
:CC98 A9 00 85 CF 18 60 20 B3
:CCA0 CC 20 D0 C8 0A 0A 0A 0A
:CCA8 85 2C 20 B3 CC 20 D0 C8
:CCB0 05 2C 60 B1 C3 C8 29 7F
:CCB8 C9 20 B0 02 09 40 60 20
:CCC0 87 C8 E0 04 90 23 85 F9
:CCC8 86 FA 8D 83 02 8E 84 02
:CCD0 85 C3 86 C4 A0 00 A9 55
:CCD8 91 C3 D1 C3 D0 0B B9 00
:CCE0 00 91 C3 C8 D0 F0 4C 9C
:CCE8 C0 86 C3 05 C3 D0 07 85
:CCF0 F9 86 FA 4C 9C C0 4C 97
:CCF8 C0 48 8A 48 98 48 A5 FA
:CD00 F0 11 A0 00 B9 00 00 AA
:CD08 B1 F9 99 00 00 8A 91 F9
:CD10 C8 D0 F1 68 A8 68 AA 68
:CD18 60 AD 14 03 AE 15 03 CD
:CD20 EE CE D0 05 EC EF CE F0
:CD28 10 85 18 86 19 AD EE CE
:CD30 AE EF CE 8D 14 03 8E 15
:CD38 03 60 A5 18 8D 14 03 A5
:CD40 19 8D 15 03 60 A2 02 D0
:CD48 02 A2 00 20 63 CD A2 00
:CD50 A1 0D 60 A2 02 D0 02 A2
:CD58 00 48 20 63 CD 68 A2 00
:CD60 81 0D 60 B5 C2 D0 0E 18
:CD68 A5 F9 75 C1 85 0D A5 FA
:CD70 75 C2 85 0E 60 85 0E B5
:CD78 C1 85 0D 60 08 68 29 EF
:CD80 48 84 0C 98 18 65 C1 85
```

```
:CD90 01 B0 10 A5 0F D0 0C A5
:CD98 0D 65 F9 85 0D A5 0E 65
:CDA0 FA 85 0E A0 00 B1 0D A4
:CDA8 0C 28 60 48 20 7C CD 84
:CDB0 0C A0 00 68 91 0D A4 0C
:CDB8 60 40 02 45 03 D0 08 40
:CDC0 09 30 22 45 33 D0 08 40
:CDC8 09 40 02 45 33 D0 08 40
:CDD0 09 40 02 45 B3 D0 08 40
:CDD8 09 00 22 44 33 D0 8C 44
:CDE0 00 11 22 44 33 D0 8C 44
:CDE8 9A 10 22 44 33 D0 08 40
:CDF0 09 10 22 44 33 D0 08 40
:CDF8 09 62 13 78 A9 00 21 81
:CE00 82 00 00 59 4D 91 92 86
:CE08 4A 85 9D 2C 29 2C 23 28
:CE10 24 59 00 58 24 24 00 1C
:CE18 8A 1C 23 5D 8B 1B A1 9D
:CE20 8A 1D 23 9D 8B 1D A1 00
:CE28 29 19 AE 69 A8 19 23 24
:CE30 53 1B 23 24 53 19 A1 00
:CE38 1A 5B 5B A5 69 24 24 AE
:CE40 AE A8 AD 29 00 7C 00 15
:CE48 9C 6D 9C A5 69 29 53 84
:CE50 13 34 11 A5 69 23 A0 D8
:CE58 62 5A 48 26 62 94 88 54
:CE60 44 C8 54 68 44 E8 94 00
:CE68 B4 08 84 74 B4 28 6E 74
:CE70 F4 CC 4A 72 F2 A4 8A 00
:CE78 AA A2 A2 74 74 74 72 44
:CE80 68 B2 32 B2 00 22 00 1A
:CE88 1A 26 26 72 72 88 C8 C4
:CE90 CA 26 48 44 44 A2 C8 0D
:CE98 20 20 20 50 43 20 20 53
:CEA0 52 20 41 43 20 58 52 20
```

```
:CEA8 59 52 20 53 50 41 42 43 YR SPABC
:CEB0 44 46 47 48 49 4C 4D 4E DFGHILMN
:CEB8 51 52 53 54 57 58 2C 3A QRSTWX,:
:CEC0 3B 45 4C C4 6A C7 21 C1 ;EL¯.!!♠
:CEC8 41 C2 9B C1 A2 C6 C4 C1 A!▪♠▬¯♠
:CED0 96 C3 FE C9 99 C3 8D C7 ▒¯.▝▌⊢▣!
:CED8 A6 C6 81 C9 FE C9 25 C1 ▒▬↓↓.▝%♠
:CEE0 AA C6 EE C9 4C C4 20 C4  ∟.▝L¯ ¯
:CEE8 D7 C9 BF CC 00 C0 74 C3 o▝▀▟_⊕⊢.¯
:CEF0 C8 C5 3F C0 A7 CA AA C5  ⌐?¯ ⌐▝ ⌐
:CEF8 AA AA AA AA AA AA AA AA  ||||||||
:CF00 AA AA AA AA AA AA AA AA  ||||||||
:CF08 AA AA AA AA AA AA AA AA  ||||||||
:CF10 AA AA AA AA AA AA AA AA  ||||||||
:CF18 AA AA AA AA AA AA AA AA  ||||||||
:CF20 AA AA AA AA AA AA AA AA  ||||||||
:CF28 AA AA AA AA AA AA AA AA  ||||||||
:CF30 AA AA AA AA AA AA AA AA  ||||||||
:CF38 AA AA AA AA AA AA AA AA  ||||||||
:CF40 AA AA AA AA AA AA AA AA  ||||||||
:CF48 AA AA AA AA AA AA AA AA  ||||||||
:CF50 AA AA AA AA AA AA AA AA  ||||||||
:CF58 AA AA AA AA AA AA AA AA  ||||||||
:CF60 AA AA AA AA AA AA AA AA  ||||||||
:CF68 AA AA AA AA AA AA AA AA  ||||||||
:CF70 AA AA AA AA AA AA AA AA  ||||||||
:CF78 AA AA AA AA AA AA AA AA  ||||||||
:CF80 AA AA AA AA AA AA AA AA  ||||||||
:CF88 AA AA AA AA AA AA AA AA  ||||||||
:CF90 AA AA AA AA AA AA AA AA  ||||||||
:CF98 AA AA AA AA AA AA AA AA  ||||||||
:CFA0 AA AA AA AA AA AA AA AA  ||||||||
:CFA8 AA AA AA AA AA AA AA AA  ||||||||
:CFB0 AA AA AA AA AA AA AA AA  ||||||||
:CFB8 AA AA AA AA AA AA AA AA  ||||||||
:CFC0 AA AA AA AA AA AA AA AA  ||||||||
```

```
:CFC8 AA AA AA AA AA AA AA AA  ||||||||
:CFD0 AA AA AA AA AA AA AA AA  ||||||||
:CFD8 AA AA AA AA AA AA AA AA  ||||||||
:CFE0 AA AA AA AA AA AA AA AA  ||||||||
:CFE8 AA AA AA AA AA AA AA AA  ||||||||
:CFF0 AA AA AA AA AA AA AA AA  ||||||||
:CFF8 AA AA AA AA AA AA AA FF  |||||||.
:D000 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D008 00 00 00 00 00 00 00 00 □□□□□□□□
:D010 00 9B 24 D1 00 00 C8 00 □■►●□□ |□
:D018 15 79 F0 00 00 00 00 00 ▌y. .□□□□□
:D020 FE FE F1 F2 F3 F4 F0 F1 ........
:D028 F2 F3 F4 F5 F6 F7 FC FF ........
:D030 FF FF FF FF FF FF FF FF ........
:D038 FF FF FF FF FF FF FF FF ........
:D040 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D048 00 00 00 00 00 00 00 00 □□□□□□□□
:D050 00 1B FF D1 00 00 C8 00 □◄■.●□□ |□
:D058 15 79 F0 00 00 00 00 00 ▌y. .□□□□□
:D060 FE FE F1 F2 F3 F4 F0 F1 ........
:D068 F2 F3 F4 F5 F6 F7 FC FF ........
:D070 FF FF FF FF FF FF FF FF ........
:D078 FF FF FF FF FF FF FF FF ........
:D080 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D088 00 00 00 00 00 00 00 00 □□□□□□□□
:D090 00 1B A5 D1 00 00 C8 00 □◄■●□□ |□
:D098 15 79 F0 00 00 00 00 00 ▌y. .□□□□□
:D0A0 FE FE F1 F2 F3 F4 F0 F1 ........
:D0A8 F2 F3 F4 F5 F6 F7 FC FF ........
:D0B0 FF FF FF FF FF FF FF FF ........
:D0B8 FF FF FF FF FF FF FF FF ........
:D0C0 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D0C8 00 00 00 00 00 00 00 00 □□□□□□□□
:D0D0 00 1B FA D1 00 00 C8 00 □◄■.●□□ |□
:D0D8 15 79 F0 00 00 00 00 00 ▌y. .□□□□□
:D0E0 FE FE F1 F2 F3 F4 F0 F1 ........
```

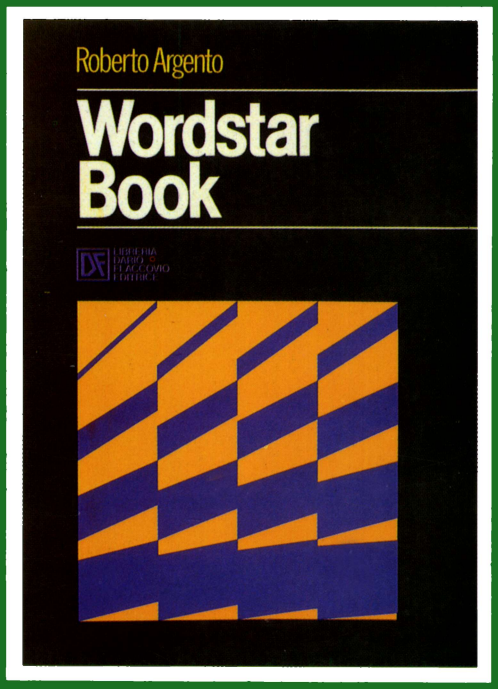
```
:D0E8 F2 F3 F4 F5 F6 F7 FC FF ........
:D0F0 FF FF FF FF FF FF FF FF ........
:D0F8 FF FF FF FF FF FF FF FF ........
:D100 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D108 00 00 00 00 00 00 00 00 □□□□□□□□
:D110 00 1B 12 D1 00 00 C8 00 □◀■●□□ |□
:D118 15 79 F0 00 00 00 00 00 ▮..□□□□□
:D120 FE FE F1 F2 F3 F4 F0 F1 ........
:D128 F2 F3 F4 F5 F6 F7 FC FF ........
:D130 FF FF FF FF FF FF FF FF ........
:D138 FF FF FF FF FF FF FF FF ........
:D140 FF 30 20 FF 00 00 00 00 .0 .□□□□
:D148 00 00 00 00 00 00 00 00 □□□□□□□□
:D150 00 1B 2A D1 00 00 C8 00 □◀*●□□ |□
:D158 15 79 F0 00 00 00 00 00 ▮..□□□□□
```

R I L O C A T O R E D. S. K.

```
:0800 00 0D 08 0A 00 9E 28 32 ▯▯▯▯▯▯▯(2
:0808 30 36 33 29 00 00 00 4C 063)▯▯▯L
:0810 98 08 86 41 84 42 A0 00 ▯▯▯A▯B ▯
:0818 B1 41 F0 06 20 D2 FF C8 ▯A▯ ▯▯ ▯
:0820 D0 F6 60 93 1F 4C 4F 41 ▯▯`▯▯LOA
:0828 44 20 41 54 20 36 34 0D D AT 64▯
:0830 0D 20 20 20 20 20 20 20 ▯
:0838 20 20 20 20 20 20 20 20
:0840 20 20 20 20 20 20 0D 0D       ▯▯
:0848 9F 00 0D 46 49 4C 45 4E ▯▯▯FILEN
:0850 41 4D 45 3A 00 0D 0D 4D AME:▯▯▯M
:0858 45 4D 20 53 54 41 52 54 EM START
:0860 20 24 00 0D 4C 4F 41 44  $▯▯LOAD
:0868 20 45 4E 44 20 49 53 20  END IS
:0870 24 00 0D 20 57 41 53 20 $▯▯ WAS
:0878 24 00 20 54 4F 20 24 00 $▯ TO $▯
:0880 0D 20 49 53 20 20 24 00 ▯ IS  $▯
:0888 30 31 32 33 34 35 36 37 01234567
:0890 38 39 41 42 43 44 45 46 89ABCDEF
:0898 A9 00 8D 20 D0 8D 21 D0 ▯▯▯ ▯▯!▯
:08A0 A0 08 A2 23 20 12 08 A0   ▯▯# ▯▯
:08A8 08 A2 4A 20 12 08 A2 00 ▯▯J ▯▯▯▯
:08B0 86 3F 20 CF FF C9 0D F0 ▯? ▯▯▯▯
:08B8 0C A6 3F E0 12 F0 D9 9D ▯▯?▯▯▯▯
:08C0 40 01 E8 D0 EB A6 3F F0 @▯▯▯▯?▯
:08C8 CF 8A A0 01 A2 40 20 BD ▯▯ ▯▯@ ▯
:08D0 FF A0 08 A2 55 20 12 08 ▯ ▯▯U ▯▯
:08D8 A9 00 85 3F 85 40 20 CF ▯▯▯?▯▯@ ▯
:08E0 FF C9 0D F0 1E A8 A2 04 ▯▯▯▯▯▯▯▯
:08E8 06 3F 26 40 CA D0 F9 98 ▯?&@▯▯▯▯
:08F0 A2 0F DD 88 08 F0 04 CA ▯▯▯▯▯▯▯▯
:08F8 10 F8 60 8A 05 3F 85 3F ▯▯`▯▯?▯?
:0900 4C DE 08 A5 3F 05 40 F0 L▯▯▯?▯@▯
:0908 F1 A9 01 A2 08 A0 00 20 ▯▯▯▯▯▯ ▯
:0910 BA FF 20 C0 FF 20 CC FF ▯▯ ▯▯ ▯▯
:0918 A9 00 20 BD FF A9 02 A2 ▯▯ ▯▯▯▯
```

```
:0920 08 A0 0F 20 BA FF 20 C0
:0928 FF A2 02 20 C6 FF A9 00
:0930 85 B7 20 E4 FF A6 B7 9D
:0938 40 01 E6 B7 C9 0D D0 F2
:0940 E8 A9 00 9D 40 01 20 CC
:0948 FF A9 0D 20 D2 FF 20 D2
:0950 FF A0 01 A2 40 20 12 08
:0958 AD 40 01 0D 41 01 C9 30
:0960 F0 05 A9 02 4C C3 FF A5
:0968 3F 8D 40 01 A5 40 8D 41
:0970 01 A2 01 20 C6 FF 20 E4
:0978 FF 8D 42 01 8D 44 01 20
:0980 E4 FF 8D 43 01 8D 45 01
:0988 20 E4 FF A0 00 91 3F E6
:0990 3F D0 02 E6 40 EE 44 01
:0998 D0 03 EE 45 01 A5 90 F0
:09A0 E7 20 CC FF A9 01 20 C3
:09A8 FF A9 02 20 C3 FF A0 08
:09B0 A2 72 20 12 08 AD 43 01
:09B8 20 F8 09 AD 42 01 20 F8
:09C0 09 A0 08 A2 7A 20 12 08
:09C8 AD 45 01 20 F8 09 AD 44
:09D0 01 20 F8 09 A0 08 A2 80
:09D8 20 12 08 AD 41 01 20 F8
:09E0 09 AD 40 01 20 F8 09 A0
:09E8 08 A2 7A 20 12 08 A5 40
:09F0 20 F8 09 A5 3F 4C F8 09
:09F8 48 4A 4A 4A 4A 20 03 0A
:0A00 68 29 0F AA BD 88 08 4C
:0A08 D2 FF 12 00 00 00 00 00
:0A10 00 00 00 00 00 00 00 00
:0A18 00 00 00 00 00 00 00 00
:0A20 00 00 00 00 00 00 00 00
:0A28 00 00 00 00 00 00 00 00
:0A30 00 00 00 00 00 00 00 00
:0A38 00 00 00 00 00 00 00 00
:0A40 00 00 00 00 00 00 00 00
```

DELLA STESSA COLLANA



ISBN 88-7758-051-8

DF 0901

L. 20.000