VIC da zero +64

a cura di Tommaso Pantuso

Riprendiamo il discorso, iniziato nel numero di luglio-agosto, sul funzionamento e i segreti del drive 1541. Contrariamente a quanto avevamo previsto, l'argomento non sarà esaurito in questa puntata, ma sarà riaffrontato nella prossima.

Il 1541: i comandi

di Luigi Tavolato

Prima di passare alla parte critica per la gestione del 1541 è opportuno vedere o rivedere i vari comandi Basic che abbiamo a disposizione, anche se molti posseggono già una buona dimestichezza nel loro uso. Possono essere suddivisi nelle seguenti categorie in funzione delle loro caratteristiche:

— Comandi per la gestione di file programmi
— Comandi per la manipolazione di file dati
— Comandi Disco
— Comandi Block
— Comandi User
— Comandi Memory

Comandi per file programmi

Su di essi non c'è moltissimo di nuovo da dire, essendo sicuramente, fra tutti i comandi Basic (e non solo), i più noti.

LOAD: carica di memoria in file programma specificato.

```
LOAD «nome file», dv, cmd
```

«dv» è il numero di periferica da cui si effettua il caricamento (8 o 9 per il drive, 1 per il datasette).

Come abbiamo visto, quando viene effettuato il «salvataggio» di un programma, in testa ad esso viene registrato il suo indirizzo iniziale. «cmd», normalmente valorizzato ad 1, se presente, segnala al computer di caricarlo in memoria a cominciare appunto da quest'indirizzo. In caso contrario, cioè se il cmd non fosse presente, il caricamento verrà effettuato a cominciare dall'indirizzo di default $0800 (2049).

Esistono dei caratteri speciali atti a sostituire parzialmente o totalmente il nome del file da caricare: «*» e «?».L'asterisco («*») sostituisce la parte finale di un nome o il nome stesso. LOAD "0:*", 8 caricherà il primo programma presente nella directory del disco azzerando contemporaneamente lo «status» del drive.

Il punto interrogativo invece sotituisce le lettere intermedie di un nome. Dovranno essere digitati tanti «?» quante sono le lettere da sostituire e nelle posizioni ad esse corrispondenti.

Questi due caratteri hanno un po' la funzione di «Jolly» nella gestione file anche non di tipo programma; vengano indicati come il nome di «wildcard flags».

SAVE: registra sull'unità periferica specificata il contenuto dell'area di memoria delimitata dai due puntatori di inizio e fine programma. Questi puntatori, espressi nella consueta forma di byte basso/byte alto, sono:

43-44: inizio del programma e dell'area di cui effettuare il salvataggio.

45-46: fine del programma o dell'area di memoria.

```
PRINT PEEK(43)+PEEK(44)•256, PEEK(45)+PEEK(46)•256
```

visualizzerà il loro contenuto.

Normalmente gestiti dal S.O., possono però essere alterati, tramite delle POKE, in modo da poter registrare una qualsiasi area di memoria.

Supponendo che le variabili INIZIO e FINE contengano rispettivamente le locazioni iniziale e finale della nuova area, la sequenza di operazioni da eseguire sarà:

```
POKE43, INIZIO—(INT(INIZIO/256))•256
POKE44, INT(INIZIO/256))•256
POKE45, FINE—(INT(FINE/256))•256
POKE46, INT(FINE/256)
SAVE «nome file», dv
```

VERIFY: una richiesta di verifica non modifica alcuna cosa né su disco né in memoria ma controlla che il file specificato sia identico a quello che è presente in quel momento in memoria.

```
VERIFY «nome file», dv
```

Particolarità

Quando effettuiamo un SAVE, il nome del file può essere costruito in maniera tale da creare effetti inconsueti nella successiva visualizzazione della directory:

```
CHR$(160)+ «nome file»
```

entrambi i doppi apici precederanno il nome del file. È un modo di proteggere il programma da scratch erronei.

```
CHR$(20)+ «nome file»
```

non verrà visualizzato il primo doppio apice.

```
«nome file»+CHR$(160)+CHR$(147)
```

verrà effettuato un HOME prima della visualizzazione del successivo nome della directory.

```
'«nome file»+CHR$(0)
```

il S.O. cercherà di indentificare i dati di un nuovo file dai caratteri che seguono visualizzando le cose più imprevedibili. Effetti altrettanto strani si avranno inserendo un CHR$(0) in mezzo al nome del file.

```
«nome file» + CHR$(0) + CHR$(0) + CHR$(0)
```

la visualizzazione della Directory verrà interrotta dopo il nome di questo file. I file che seguono resteranno comunque accessibili.

```
«nome file»+CHR$(160)+CHR$(13)
```

visualizzerà il tipo file a capo.

«nome file» + CHR$(160) + CHR$(COL)

sostituendo alla variabile COL il codice ASCII di un colore, i nomi dei file che seguono verranno visualizzati con quel colore. Aggiungendo un diverso codice a ciascun nome file, potremo così creare delle directory multicolori.

Se le lettere del nome del file hanno un valore ASCII maggiore di 128 verrà visualizzata una directory piuttosto caotica.

Un CHR$(160) in mezzo al nome determinerà la visualizzazione del secondo apice al suo posto.

Se invece inseriamo un CHR$(20), la lettera che lo precede scomparirà.

«nome file» + CHR$(160) + CHR$(8) + CHR$(14)

causerà, ogni volta che visualizzeremo la directory, il passaggio automatico al set di caratteri minuscolo e disabiliterà lo SHIFT-COMMODORE che effettua il passaggio da un set all'altro.

Provate ad esempio a salvare un file il cui nome sia costituito dalla sequenza di caratteri:

CHR$(20) + CHR$(20) + CHR$(20) + CHR$(20) + CHR$(20) + CHR$(20) + «Pro» + CHR$(160) + CHR$(20) + «va»

Comparirà la parola «prova» allineata con il bordo sinistro dello schermo e non delimitata da alcun apice. Dunque, codici «strani» per effetti «strani».

Queste particolarità, singolarmente o combinate, possono, già, di per sé, costruire un tipo di protezione, anche se non insormontabile, che metterà in seria difficoltà chiunque tenterà di caricare un file così trattato senza conoscere la «chiave di lettura». «Folli» directory potranno essere così generate: provare per credere.

Alcune routine di esempio

```
10 REM*** MUOVE LA TESTINA DI TRACCIA IN TRACCIA ***
20 :
30 OPEN1,8,15,"I0"
40 FORTR=1TO35
50 PRINT#1,"M-W"CHR$(6)CHR$(0)CHR$(1)CHR$(TR):REM*** IMPOSTA TRACCIA E  SETTORE
60 PRINT#1,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(176):REM* CMD LETTURA - MUOVE R/W HEAD
70 FORI=1TO600:NEXT
80 NEXTTR
90 CLOSE1
```

```
10 REM******  LEGGE ID  ******
20 :
30 OPEN1,8,15,"I0"
40 PRINT#1,"M-R"CHR$(18)CHR$(0)CHR$(2)
50 INPUT#1,A$:PRINTA$
60 CLOSE1
```

```
10 REM*** LEGGE BLOCCHI LIBERI ***
20 :
30 OPEN1,8,15,"I0"
40 PRINT#1,"M-R"CHR$(250)CHR$(2):GET#1,LO$
50 PRINT#1,"M-R"CHR$(252)CHR$(2):GET#1,HI$
60 PRINTASC(LO$+CHR$(0))+ASC(HI$+CHR$(0))*256
70 CLOSE1
```

```
10 REM*** LEGGE IL NOME DEL DRIVE ***
20 REM**** E LA VERSIONE DEL DOS ****
30 :
40 OPEN1,8,15
50 PRINT#1,"M-R"CHR$(192)CHR$(229)CHR$(8)
60 INPUT#1,A$
70 PRINT"DRIVE   "MID$(A$,5,3)CHR$(ASC(RIGHT$(A$,1))AND127)
80 PRINT"DOS     "LEFT$(A$,3)
90 CLOSE1
```

```
10 REM**** CONTROLLA SE IL FLOPPY ****
20 REM*** E' PROTETTO DA SCRITTURA ***
30 :
40 OPEN1,8,15
50 PRINT#1,"M-R"CHR$(0)CHR$(28)
60 INPUT#1,L$:L=ASC(L$+CHR$(0))AND16
70 IFL=0THENPRINT"WRITE PROTECT ON !"
80 CLOSE1
```

Apertura di un canale di comunicazione

Ogni qualvolta desideriamo far compiere ad una unità periferica, nel nostro caso il drive una o più operazioni è necessario comunicare per prima cosa che intendiamo «colloquiare» con lei e, successivamente, quali operazioni desideriamo che effettui.

Tramite l'operazione di OPEN, stabiliremo uno o più canali di comunicazione con essa, un po' come si fa componendo un numero telefonico; attraverso essi invieremo o riceveremo tutte le informazioni (dati, comandi, messaggi...). Abbiamo a disposizione quindici canali, numerati da 1 a 15 e 5 buffer (0-4).

Questi ultimi sono delle aree di memoria RAM di 256 byte ciascuna in cui vengono immessi e lasciati a disposizione dell'utente i dati che sono stati letti dal floppy o inviati dal C-64: queste aree, che si estendono nel 1541 da $0300 a $07FF, vengono anche utilizzate per ospitare i vari programmi di utilità (scritti in assembler), che permetteranno di potenziare le possibilità del drive (ex. fast format, fast copy, speed DOS, i vari TURBO...).

Il primo e l'ultimo canale di comunicazione vengono usati per inviare comandi, mentre i restanti sono dedicati alla gestione dati. Unico limite nel loro uso è che, contemporaneamente, non possiamo tenerne aperti più di cinque né usare più di tre buffer (dato che il DOS se ne riserva uno per memorizzarsi la BAM ed un altro per effettuare le proprie operazioni di gestione).

OPEN ch, up, is

è la corretta sintassi per aprire un canale, dove «ch» è il canale che vogliamo aprire, «up» è l'unità periferica con cui intendiamo comunicare (8 o 9 per il drive) e «is» è l'indirizzo secondario. Nel caso in cui quest'ultimo non venga specificato, il sistema assume per default lo zero.

Vengono utilizzati 16 indirizzi secondari (0-15): lo zero viene usato per operazioni di caricamento di programmi, l'uno è invece usato per salvarli, il canale quindici permette di accedere allo STATUS del disco. I restanti indirizzi sono usati per operare sui dati.

Un canale, che non sia di comando (1o15), può essere associato o ad un buffer o ad un file. Avremo allora, nel primo caso:

OPEN ch, up, is, «#bu»

mentre per il secondo

OPEN ch, up, is, «0: nome file, tf, op»

dove:
bu è il buffer che intendiamo selezionare (0-4).
tf è il tipo di file che vogliamo gestire (PRG-USR-SEQ-REL).
op è l'operazione da effettuare («W» per scrivere un file o «R» per leggerlo).

Una volta effettuata la OPEN, potremo comunicare con il drive. Tramite un PRINT# potremo inviare i dati o i comandi, mentre con un GET# o un INMPUT# riceveremo dati o i messaggi sullo stato del drive.

Concettualmente, il funzionamento di questi comandi è identico a quello dei corrispettivi PRINT, GET e INPUT.

In linea generale, prima di usare uno qualsiasi dei comandi disponibili,

bisogna avere preventivamente aperto almeno due canali: uno per trasmettere i comandi e ricevere i messaggi sullo STATUS del drive, ed un altro per inviare o ricevere i dati. Fanno eccezione i soli comandi Memory: se intendiamo operare unicamente con essi, sarà necessario aprire il solo canale di comando (#1 o #15).

```
10 OPEN 1,8,15
20 OPEN 6,8, 6, "#2"
```

In questo modo abbiamo aperto il canale di comando (#1) ed il canale per la gestione dati (#6), al quale abbiamo associato il buffer #2, in esso transiteranno i dati di arrivo o in partenza.

È opportuno controllare sempre lo STATUS del disco dopo una OPEN (o dopo aver effettuato una qualsiasi operazione su disco). Una ulteriore riga di programma sarà:

```
30 INPUT#1,ER,ME$,TR,SE: IF ER > 19
THEN PRINT «Errore:» ER,ME$,TR,SE
```

Dove ER è il numero dell'errore, ME$ ne è la descrizione; TR ed SE danno indicazione rispettivamente della traccia e del settore in cui questo è stato riscontrato.

Comandi disco

Questi comandi verranno inviati, normalmente in forma abbreviata, insieme al numero del drive verso cui sono indirizzati (0 o 1), tramite un PRINT#, seguiti, quando richiesto, da una serie di parametri:

```
PRINT#ch, «cmd: «+» parametri»
```

INIZIALIZZAZIONE (I): determina il posizionamento della testina di lettura/scrittura sulla traccia 1, per controllare eventuali problemi di allineamento, e quindi effettua il caricamento in memoria della BAM, del nome del disco e dell'identificatore. Finite queste operazioni, viene comunicato sull'opportuno canale la situazione riscontrata.

È opportuno inviarlo prima di cominciare ad operare su di un floppy per controllarne lo stato.

VALIDATE (V): effettua una riorganizzazione del floppy recuperando eventuali settori non più utilizzati, ma ancora segnalati come tali nella BAM, e ottimizzando l'occupazione di memoria disco. Attenzione però che sul floppy non siano presentati file Random poiché i blocchi da essi occupati verrebbero disallocati. Questo avviene perché il DOS considera occupati soltanto quei blocchi facenti parte dei file che compaiono nella directory.

NEW (N): diverso dal NEW del BASIC, comunica al drive di effettuare la formattazione del floppy. Insieme ad esso vanno specificati il nome del disco ed il suo identificatore.

```
40 PRINT#1, «NO: nome disco, identificatore»
```

Se il floppy era già stato formattato e non intendiamo modificare l'ID, è sufficiente specificare il solo nome del disco: la routine di formattazione si limiterà unicamente a cancellare la directory e ad azzerare la BAM, registrando quindi il nuovo nome del floppy.

SCRATCH (S): cancella il file specificato.

```
40 PRINT#1, «S0: nome file»
```

RENAME (R): cambia il nome di un file.

```
40 PRINT#1, «R0: nuovo nome=vecchio nome»
```

COPY (C): duplica un file sul medesimo floppy (se non si dispone di due drive), ma con un nome diverso.

```
40 PRINT#1, «C0: nome file copia=nome file origine»
```

È possibile, tramite di esso, effettuare anche il merge di più file:

```
40 PRINT#1, «C0: nome file merge=0:file 1,0: file 2,...0: file N»
```

Lo zero che precede ogni nome file indica da quale drive esso deve essere letto.

Comandi Block

Possono essere definiti comandi ad accesso diretto, poiché permettono di accedere direttamente al settore specificato, per poterlo successivamente leggere ed aggiornare. Una qualsiasi richiesta di un settore non previsto dal DOS genererà un «ILLEGAL TRACK OR SECTOR».

BLOCK-READ (B-R): immette il blocco dati specificato nel buffer che stiamo utilizzando (specificato nella OPEN del file di manipolazione dati). In quest'ultimo, tramite delle GET#, leggeremo i dati presenti nel settore richiesto. Con:

```
40 PRINT#1, "B-R:"6,0; 18;0
```

richiediamo di leggere il settore 0 della traccia 18 del floppy che si trova nel drive 0 e di immettere il contenuto nel buffer 2 associato al canale 6. Questo comando, però, per come è stato realizzato, ignora del tutto i primi due byte del settore, che non verranno letti. Se volessimo effettuare la lettura completa del blocco, dovremmo utilizzare il comando U1, mantenendo la medesima sintassi:

```
40 PRINT#1, "U1:"6;0;18;0
50 GET#6,A$, B$
```

La riga 50 effettua la lettura dei primi due byte disponibili nel buffer, immettendoli nelle variabili A$ e B$. Tramite ulteriori GET# potremo leggere i successivi byte.

Ad ogni richiesta di un nuovo byte viene aggiornato il valore del BUFFER POINTER: esso contiene la posizione nel buffer del successivo byte disponibile.

È possibile assegnare a questo puntatore un valore predeterminato in modo da poter accedere senza difficoltà ad un qualsiasi byte del buffer in ogni momento senza dover per questo scorrere tutti quelli che lo precedono. Per leggere il centottantesimo byte presente nel buffer associato al canale 6 digiteremo:

```
60 PRINT#1, "B-P"6;180
70 GET#6, A$
```

Naturalmente la posizione all'interno del buffer sarà espressa mediante un numero compreso tra 0 e 255.

Se invece volessimo scrivere un byte, la procedura sarà la medesima, con la differenza che al GET#6, A$ sostituiremo un PRINT#6, A$. È importante mettere il punto e virgola dopo A$, poiché altrimenti verrebbe inviato anche un CHR$(13) che modificherebbe, magari indesideratamente, il byte successivo a quello interessato.

BLOCK-WRITE (B-W): tutte le variazioni vengono effettuare all'interno del buffer selezionato e non direttamente sul settore richiesto, è pertanto necessario, ultimate tutte le modifiche, riscrivere il settore aggiornato. A questo scopo useremo il comando di Block-Write, Questo ha però il medesimo difetto del precedente: ignora i primi due byte del buffer (e del settore). Questo inconveniente può essere facilmente superato utilizzando in sua vece il comando U2.

Supponiamo di voler cambiare il nome di un floppy senza doverlo riformattare: le variabili N$ ed S$ contengono rispettivamente il nuovo nome degli shifted space (CHR$(160)). La procedura sarà:

```
40 PRINT#1, "U1:"6;0;18;0
50 PRINT#1, "B-P"6;144
60 PRINT#6,LEFT$(N$+S$,16);
70 PRINT#1, "U2:"6;0;18;0
```

Tramite la U2 abbiamo scritto nel

settore 0 della traccia 18 il contenuto del buffer associato al canale 6.

BLOCK-ALLOCATE (B-A): richiede al DOS di controllare se un settore è disponibile o meno. Nel caso in cui lo sia, aggiorna la BAM segnalandolo come occupato, altrimenti invia un messaggio di «NO BLOCK».

Viene utilizzato prevalentemente nella gestione dei file Random, per cautelarsi da rischi di sovrascrittura e quindi di una deleteria e irreversibile perdita di dati.

BLOCK-FREE (B-F): effettua l'operazione inversa della precedente, rendendo disponibile un settore segnalato nella BAM come occupato.

BLOCK-EXECUTE (B-E): legge il programma in linguaggio macchina contenuto nel settore specificato, lo immette nel buffer selezionato e quindi lo esegue.

Comandi User

Questi comandi determinano l'esecuzione di routine in linguaggio macchina ad iniziare da indirizzi predeterminati:

```
UA o U1 –> $CD5F (Block Read)
UB o U2 –> $DC97 (Block Write)
UC o U3 –> $0500
UD o U4 –> $0503
UE o U5 –> $0506
UF o U6 –> $0509
UG o U7 –> $050C
UH o U8 –> $050F
UI o U9 –> $FF01 (Set Speed or Indirect Jump Over)
UJ o U: –> $EAA0 (Reset Vector)
```

I comandi da U3 a U8 effettuano un salto ai corrispondenti indirizzi del buffer 2 ($0500–$05FF).

Il comando UJ determina l'esecuzione della routine di RESET del drive. Con UI e U9 viene settata la velocità del drive se insieme ad esso è stato inviato un «+» o un «–», altrimenti viene eseguita la routine il cui indirizzo è stato memorizzato nelle locazioni RAM $65 e $66 (101-102). I comandi U1 e U2 li conosciamo già.

Comando	Sintassi
Block-Read	B-R;canale;drive;traccia;settore
	U1:canale;drive;traccia;settore
Block-Write	B-W;canale;drive;traccia;settore
	U2:canale;drive;traccia;settore
Block-Allocate	B-A;canale;drive;traccia;settore
Block-Free	B-F;canale;drive;traccia;settore
Block-Execute	B-E;canale;drive;traccia;settore
Buffer-Pointer	B-P;canale;Posizione
Memory-Read	M-R;CHR$(BB)CHR$(BA)CHR$(Numero Bytes)
Memory-Write	M-W;CHR$(BB)CHR$(BA)CHR$(N.Bytes)CHR$(Byte 1)...CHR$(Byte N)
Memory-Execute	M-E;CHR$(BB)CHR$(BA)

Questo è un piccolo programma di utilità che potrà ulteriormente chiarire qualche idea sull'uso dei comandi appena visti. È diviso in due parti: la prima ha lo scopo di proteggere un floppy da scrittura o da scratch permanentemente, mentre la seconda serve ad eliminare la protezione.

Comandi Memory

Tramite essi è possibile accedere facilmente ad ogni area di memoria del 1541, byte per byte, senza difficoltà.

Scrivere nuove routine in linguaggio macchina, leggere o alterare i parametri del DOS (blocchi liberi, identificatore, nome del disco, contatori...) sarà un'operazione estremamente semplice, dovremo però imparare a conoscere bene il modo di operare di questo drive e del suo microprocessore per sfruttare completamente gli «strumenti» che ci vengono messi a disposizione. In seguito vedremo come è organizzata la memoria del 1541.

Gli indirizzi su cui opereremo saranno sempre espressi sotto forma di Byte Basso/Byte Alto (BB/BA).

MEMORY-READ (M-R): tramite esso è possibile leggere il contenuto di una o più locazione successive RAM o ROM.

Se volessimo leggere il contenuto della locazione 28 ($1C), il quinto bit della quale indica se il disco su cui stiamo lavorando è protetto da scrittura o meno, non dovremmo fare altro che inserire nei nostri programmi questa routine:

```
40 PRINT#1, "M-R"CHR$(23)CHR$(0)
50 GET#1, A$:A = ASC(A$ + CHR$(0))
AND 16
60 IF A=0 THEN PRINT «PROTETTO!»
```

Per leggere il contenuto di più locazioni consecutive tramite un unico comando, dovremo inviare, insieme all'indirizzo, anche il numero di byte (NB) che si intendono ricevere, fino ad un massimo di 80. Quindi tramite un INPUT# chiederemo al DOS di immetterli nella variabile specificata. Volendo leggere, ad esempio, la versione del DOS, digiteremo:

```
40 PRINT#1, "M-R" CHR$(192) CHR$(229)
CHR$(3)
50 INPUT#1, A$: PRINT A$
```

Quest'ultimo modo di usare la Memory-Read non funziona sempre: ci sono alcune aree di memoria nelle quali non è possibile accedere a più di un byte per volta.

MEMORY-WRITE (M-W): mediante esso protremo scrivere nella memoria del drive le nostre routine in linguaggio macchina o modificare i parametri con cui opera il drive. Possiamo trasmettere contemporaneamente fino a 32 byte, con modalità molto simili a quelle del precedente comando.

Per modificare il numero di device del nostro drive via software (da 8 a 9), dovremo digitare:

```
40 PRINT#1, "M-W" CHR$(119) CHR$(0)
CHR$(2) CHR$(9 + 32)CHR$(9+64)
```

MEMORY-EXCUTE (M-E): determina l'esecuzione del programma in linguaggio macchina, presente nella memoria del drive, a partire dalla locazione di memoria specificata (sempre nella forma BB/BA).

Mandando in esecuzione la routine che inizia a $C12C potremo far lampeggiare il LED del drive come se si fosse verificata una condizione di errore:

```
40 PRINT#1, "M-E"CHR$(44)CHR$(193)
```

N.B. I comandi Block, Memory e User funzionano soltanto in modo programma, in modo diretto genereranno un ? SYNTAX ERROR.

```
100 OPEN1,8,15,"I0"
110 OPEN6,8,6,"#3"
120 PRINT#1,"U1:"6;0;18;0
130 PRINT#1,"B-P"6;2
140 PRINT#6,CHR$(76);
150 PRINT#1,"U2"6;0;18;0
160 CLOSE6: CLOSE1: END
```

```
200 OPEN1,8,15,"I0":
210 OPEN6,8,6,"#3"
220 PRINT#1,"U1:"6;0;18;0
230 PRINT#1,"B-P"6;2
240 PRINT#1,"M-W"CHR$(1)CHR$(1)CHR$(1)CHR$(65)
250 PRINT#6,CHR$(65);
260 PRINT#1,"U2:"6;0;18;0
270 CLOSE6: CLOSE1
```