di Tommaso Pantuso

VIC da zero +64

Il Buffer di tastiera

Nell'ultimo articolo di questa serie abbiamo visto, insieme ad alcune caratteristiche generali della memoria video, anche come scrivere o disegnare sullo schermo senza utilizzare l'istruzione Print, come intercettare delle parole impresse sullo schermo ed un modo rudimentale di memorizzazione di schermate. Vogliamo ora, nei due articoli che seguiranno, fare un passo avanti illustrando degli altri metodi che permettono di conservare su un supporto magnetico il contenuto di una pagina video e di richiamarlo al momento opportuno. Vedremo inoltre come sia possibile codificare il contenuto della memoria video con un insieme di numeri in maniera che risulti facilmente interpretabile e quindi riproducibile da parte di chiunque. Di schermate comunque parleremo la prossima volta perché prima è indispensabile conoscere l'argomento che segue.

Il Buffer di tastiera

Molti, leggendo il titoletto, si chiederanno cosa c'entra il Buffer della tastiera con la memoria video. Vi diremo allora che direttamente non c'è nessun legame tra queste due cose, ma noi riusciremo in un certo senso a legarle indirettamente per raggiungere una elementare codifica di una videata completa. Ma andiamo per gradi. Ogni volta che noi premiamo un tasto, il codice ASCII del carattere corrispondente viene conservato in una zona appositamente concepita per accumulare un certo numero di caratteri, per la precisione 10, in attesa di essere processati. Il tutto avviene in maniera trasparente all'utente poiché gestito per mezzo dell'interrupt del sistema (di cui abbiamo già ampiamente parlato) che viene abilitato ogni sessantesimo di secondo. Il perché della presenza di un Buffer del genere non è molto difficile da intuire: se noi premiamo un tasto prima che il sistema sia pronto ad esaminarlo, esso non andrà perduto perché il codice corrispondente sarà conservato nella zona di cui stiamo parlando. Vogliamo attirare la vostra attenzione su un primo importante comportamento del Buffer in questione: quando esso è completo, e questo avviene se riusciamo a mettere in attesa dieci caratteri, la pressione dell'undicesimo tasto ne provoca lo svuotamento dopo di che il processo può ricominciare. Vista l'ampiezza ridotta di tale zona (dieci byte) non si pensi comunque di riuscire a premere dieci tasti e battere il sistema in velocità nel senso che sarà molto difficile accumulare dieci battute e provocare l'azzeramento del Buffer prima che il suo contenuto sia stato processato. Vediamo un po' più da vicino le implicazioni di questo comportamento.

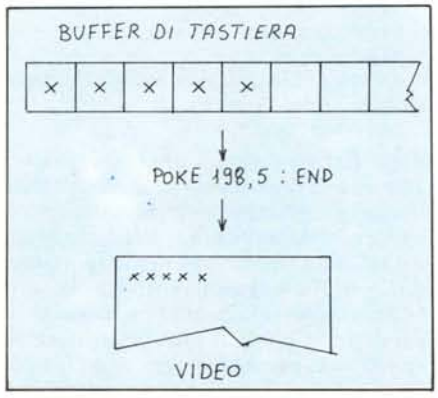

Figura 1 - Immaginiamo di riempire le prime cinque locazioni del Buffer. Dando Poke 198,5 si ottiene lo svuotamento ed il contenuto del Buffer compare sullo schermo.

Come detto, il Buffer è composto da dieci byte e si estende, sia per il Vic che per il C 64, dalla locazione 631 alla 640. L'elemento più imporatante dell'insieme di gestione è un puntatore P ad un byte situato nella locazione 198 che contiene il numero di tasti accumulati. Caricando allora il Buffer e dando un opportuno valore a P potremo forzarne lo svuotamento e sfruttare a nostro vantaggio tale comportamento. In pratica, utilizzando le proprietà della zona in questione potremo simulare dei comandi diretti...durante lo svolgimento di un programma. Sembra un controsenso, ma se ci seguirete attentamente, avrete presto le idee chiare.

Simuliamo i comandi diretti

Supponiamo di voler caricare un programma (ad esempio da cassetta) e di avviarlo automaticamente a caricamento avvenuto. La cosa è molto semplice perché basta premere il tasto Shift insieme al tasto Run/Stop, operazione che provoca l'effetto desiderato. Supponendo però di voler compiere la stessa azione durante lo svolgimento di un programma, ci troveremmo di fronte ad un enigma difficilmente risolubile con le consuete procedure. Infatti, normalmente, quando la macchina sta elaborando, ci permette di interagire con essa mettendoci a disposizione dei comandi d'I/O non sufficienti però ai nostri scopi.

Ora, se noi potessimo far fermare per un istante il sistema, scrivere sullo schermo il comando diretto richiesto, premere Return ecc. ecc. senza ... toccare la tastiera, avremmo raggiunto il nostro scopo. Il Buffer di tastiera ci offre la soluzione nei seguenti termini. La sequenza Load + Return + Run è codificata semplicemente con il numero 131; quando noi premiamo i tasti Shift + Run /Stop, tale numero viene introdotto nel Buffer di tastiera e da quel momento in poi le cose vanno da sé. Dato che il Buffer è accessibile mediante dei comandi di Poke, noi possiamo simulare abbastanza semplicemente il comportamento appena descritto andando a scrivere nella locazione interessata, la 631, il numero che codifica il comando in questione. Provate allora ad effettuare in modo diretto:

```
Poke 631,131: <Return>.
```

Non rileverete nessun cambiamento. L'operazione indicata non è infatti sufficiente a provocare lo svuotamento del Buffer e l'esecuzione dei comandi in esso contenuti. Dovremo specificare un'altra informazione utile al sistema per procedere e cioè il numero di caratteri che vogliamo "tirar fuori" dalle locazioni 631 in poi (fino alla 640) attribuendo il relativo valore al puntatore situato nella locazione 198. In altre parole, se immaginiamo di riempire le prime n locazioni del Buffer (con n che va da 1 a 10), perché la sequenza da esse codificata venga eseguita dovremo dare l'ulteriore comando:

```
Poke 198,n
```

in seguito al quale le locazioni citate ver-

ranno "svuotate", nel senso che comparirà sullo schermo il loro contenuto decodificato (cioè il carattere al posto del codice) (figura 1). Ritornando a noi, il codice del caricamento + autoLoad è 131 e quindi il comando completo occupa una sola locazione; dopo aver allora "Pokato" 131 nella posizione 631, corrispondente al primo byte del Buffer, dovremo scrivere il numero 1 nel byte rappresentante il puntatore per comunicare al sistema che deve considerare solo il primo elemento del Buffer. Il tutto si risolverà con:

```
Poke  631,131:Poke  198,1 <Return>
```

che farà comparire sullo schermo la scritta "Press Play on Tape" e genererà automaticamente il Return (figura 2).

La prima cosa importante da imparare è che, se la procedura descritta viene gestita da un programma, perché il contenuto del Buffer di tastiera venga preso in considerazione bisogna... uscire dal programma. In altre parole, dopo aver immagazzinato gli appositi valori nelle adeguate locazioni con delle Poke, bisogna scrivere di seguito il comando End. Il sistema scriverà allora Ready e solo dopo questo evento andrà a leggere il contenuto del Buffer. Ciò crea una difficoltà: il programma si ferma. Questo inconveniente può comunque essere risolto con la stessa tecnica codificando nel Buffer il comando Goto m che rimanderà l'esecuzione ad una certa linea, nel nostro caso la m. Avremo comunque modo di capire meglio questo fatto con degli esempi, fra breve.

Un altro inconveniente è dato dal fatto che, rimandando in esecuzione il programma con un Goto, viene perso il contenuto delle variabili. Anche questo inconveniente può comunque essere risolto memorizzando tali variabili in alcune locazioni inutilizzate dal sistema. Per fare un semplice esempio, se abbiamo la necessità di conservare il valore A = 54, basterà, prima del Goto, effettuare: Poke L,A dove L è una qualsiasi locazione di Ram libera che non viene modificata dal sistema nel corso dell'esecuzione di un programma, se non dietro nostra richiesta.

I primi esempi

Ricapitolando, l'impiego principale che noi possiamo trarre dai comportamenti descritti è quello di permettere ad un programma di automodificarsi, nel corso dell'esecuzione generando delle istruzioni così come verrebbero scritte dall'utente arrestando il programma. Abbiamo visto come risolvere alcune difficoltà e vogliamo accennare al modo in cui risolverne un'altra introdotta dalle modeste dimensioni del Buffer. Per fare un esempio, supponiamo di voler aggiungere, in seguito ad una determinata scelta, la linea:

```
10 PRINT "CASA".
```

Se il programma fosse fermo, basterebbe scrivere 10 Print "casa" <Return>, ma dato che supponiamo che il programma sia in esecuzione, dovremo simulare l'operazione mediante l'impiego del Buffer.

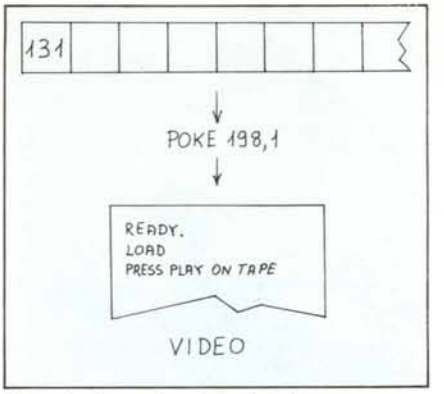

Figura 2 - Generazione di Load + Autorun.

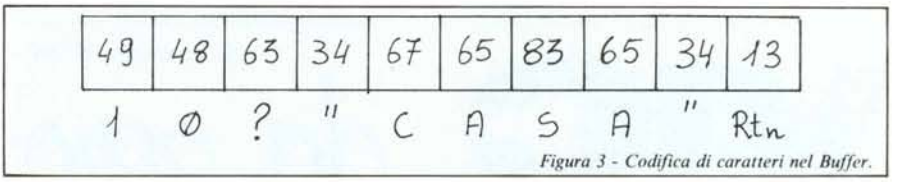

Figura 3 - Codifica di caratteri nel Buffer.

Per prima cosa dobbiamo ricavare i codici ASCII dei vari elementi costituenti la frase completa utilizzando le apposite tabelle fornite nei manuali o per mezzo del comando Print Asc("x"), dove x è il carattere di cui vogliamo ottenere il codice. Utilizzando per Print la forma ridotta, data dal punto interrogativo, otterremo la seguente tabella:

N	Carattere	Codice
1	1	49
2	0	48
3	?	63
4	"	34
5	C	67
6	A	65
7	S	83
8	A	65
9	"	34
10	Return	13

A questo punto provate ad immettere questi dati con il seguente programma:

```
 1 FOR I 0 TO 9
 2 READ A
 3 POKE 631+I,A
 4 NEXT I:POKE 198,10
20 DATA 49,4863
34,67,65,83,65,34,13
```

e date il Run. Vedrete allora comparire sullo schermo la scritta

```
10 PRINT"CASA"
```

dopo di che si genererà automaticamente il Return e la frase verrà memorizzata come linea di programma, identificata dal numero 10; potrete verificare questo fatto dando il List. Bene: fino a questo punto nessuna difficoltà. Immaginiamo però che la parola da scrivere sia composta appena da un carattere in più o il numero di linea sia formato da più di due cifre. In tali condizioni il Buffer di tastiera non avrebbe più la capacità di contenere tutti i caratteri e a prima vista ci vedremmo impossibilitati ad utilizzare i metodi precedenti. Possiamo però utilizzare una tecnica mista. Per mezzo dell'istruzione:

```
n PRINT"10PRINT"CHR$(34)"..."CHR$(34)"
```

faremo scrivere dal programma la linea che ci interessa dove n rappresenta il numero della linea ed al posto dei puntini introdurremo la parola desiderata. Chr$(34) è il codice delle virgolette ed è stato usato perché altrimenti non si potrebbero scrivere più di due virgolette sulla stessa linea. Scritta la linea sullo schermo, dovremo fare in modo che il cursore vi si "depositi" sopra e ciò potrà essere ottenuto introducendo nel Buffer, per il numero di volte sufficiente, il codice del carattere Shift + Crsr verticale. Fatto ciò non ci resta che simulare la pressione del Return con il codice 13 nel Buffer. Questa sequenza di operazioni, che "narrata" sembra complessa, è in realtà molto semplice e viene implementata con le seguenti linee:

```
1 PRINT"Shift + Clr/Home"
2 PRINT"10PRINT"CHR$(34)"CASA"CHR$(34)
3 POKE631,145:POKE632,145:POKE633,145
4 POKE634,13:POKE198,4
```

Nella linea 3, il codice 145 nei primi tre byte provoca lo spostamento del cursore di tre posizioni verso l'alto. Non è difficile osservare, dato il Run, che questo segmento funziona come il precedente con l'unica ma sostanziale differenza che la linea da aggiungere può in questo caso essere lunga molto più di quanto sia permesso dalla capienza del solo Buffer di tastiera.

Un passo avanti

Supponiamo ora che che il segmento di cui abbiamo parlato faccia parte di un programma e che, dopo aver aggiunto la linea che interessa, il controllo debba ripassare al programma in oggetto. Come già detto, il contenuto del Buffer di tastiera viene fuori per così dire solo dopo un End; nel segmento precedente l'End veniva introdotto automaticamente dopo la linea 4 perché il programma terminava, ma in molti dei casi in cui può essere impiegata la tecnica descritta il programma continua e se poniamo un "End" per provocare lo svuotamento del Buffer otterremo anche l'arresto dell'elaborazione e ciò evidentemente non sempre fa comodo.

Come però abbiamo accennato, la soluzione a questo ulteriore problema è abbastanza semplice in quanto basta far simulare, nel corso dello svuotamento, l'istruzione diretta Goto m, dove m rappresenta la linea da cui vogliamo ricominci lo svolgimento del programma. Tale istruzione la daremo in forma ridotta per risparmiare spazio, cioè come G + Shift O a cui corrispondono i codici 71 (G) e 111 (Shift O).

Per chiarire le idee facciamo il solito esempio.

Consideriamo sempre il segmento precedente e supponiamo, dopo aver aggiunto la linea 10, di voler passare il controllo di nuovo al programma mandando l'esecuzione ad una nuova linea, la 20, che provoca l'arresto momentaneo fino alla pressione di un tasto qualsiasi. La linea in questione può ad esempio essere la seguente:

```
20 GET A$:IF A$="" THEN 20;
```

aggiungendo nel nostro caso solo questa linea, quando al momento opportuno si premerà un tasto il programma avrà termine. Il programmino d'esempio proposto è il seguente:

```
0 PRINT"Clr/Home"
1 PRINT"10PRINT"CHR$(34)"CASA"CHR$
(34)
2 POKE631,145:POKE632,145:POKE633,145
3 POKE634,13:POKE635,71:POKE636,111
4 POKE637,50:POKE638,48:POKE639,13
5 POKE198,9:END
10 REM questa linea sarà modificata
20 GETA$: IFA$=" " THEN 20
```

Anche in questo caso per maggiore chiarezza presentiamo la tabellina dei caratteri introdotti nel Buffer ed i relativi codici:

N	Carattere	Codice
1	Shift + Crsr vert.	145
2	Shift + Crsr vert.	145
3	Shift + Crsr vert.	145
4	Return	13
5	G	71
6	Shift + 0	111
7	2	50
8	0	48
9	Return	13

Il segmento ha bisogno di pochi commenti. Dopo il <Run> vedrete comparire la linea 10, il cursore vi ritornerà sopra e sarà generato un primo Return dopodiché comparirà il Ready. Dopo ciò verrà scritto sullo schermo, in forma ridotta, Goto 20 e quindi generato un altro Return. Con Poke 198,9:END si ottiene lo svuotamento del Buffer.

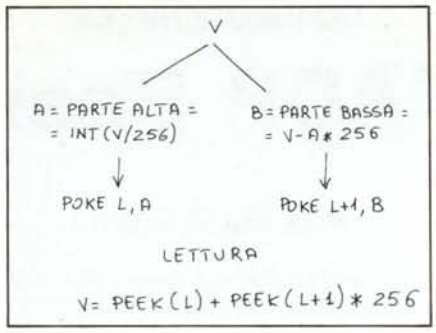

Figura 4 - Scomposizione di una variabile V per la memorizzazione in 2 byte.

Salviamo le variabili

Ci resta ora da approfondire un ulteriore aspetto che ci permetterà di utilizzare in maniera piú produttiva il Buffer di tastiera: il salvataggio delle variabili.

Nel programma precedente, quando viene generata l'istruzione Goto 20, se il sistema sta conservando qualche variabile questa va perduta. Sembrerebbe allora che in una grossa quantità di casi l'utilizzo del Buffer diventi inutile a causa di questo handicap. Anche questa volta vogliamo proporvi una soluzione del problema. Come saprete, esistono nella memoria del Vic 20 o del C 64 delle locazioni che l'utente può impiegare a suo piacimento per memorizzarvi delle piccole routine in linguaggio macchina senza che debba proteggere delle apposite zone nella parte alta della memoria. Una di queste zone accessibili è ad esempio il Buffer usato dal registratore a cassette, al quale possiamo accedere con una certa tranquillità se non sono richiesti caricamenti di dati o programmi da nastro. Di tale zona, per il nostro esperimento, utilizzeremo la locazione 820.

Il "trucco" che vi proponiamo, se così si può chiamare, consiste nel memorizzare, per mezzo del comando Poke, il valore della variabile che andrebbe perduta in una o più locazioni da cui andremo poi a ripescarla al momento opportuno. Ad esempio, se A = V con V minore di 256, il tutto si risolverà con Poke L,A dove L è la locazione in cui depositiamo il valore da salvare: lo stesso valore potrà poi essere riletto con Peek (L). Se la variabile supera il limite suddetto, potremo dividerla in una parte più significativa (parte alta) ed in una meno significativa (parte bassa) e conservare i valori in due locazioni. La suddivisione in parte alta e bassa può ad esempio essere ottenuta nel seguente modo:

```
A = parte alta  = int(V$256)
B = parte bassa = V-A*256
```

e per riavere la variabile si effettuerá:

```
V = A*256 + B.
```

Per variabili di valore più elevato si possono studiare altri metodi ma per il momento la cosa non ci riguarda.

Tornando a noi, supponiamo di voler aggiungere al programma precedente per tre volte la linea Print"casa" con numerazione che va da 1 a 3. Normalmente la prima cosa che viene in mente è di impiegare un contatore, ma nel nostro caso ciò è impossibile perché esso andrebbe perduto dopo il Goto. La soluzione completa ve la forniamo nel segmento della figura 5 che ormai dovreste essere in grado di interpretare da soli. Notare come l'incremento della locazione 820, preventivamente azzerata, venga effettuato semplicemente con Poke (830), Peek (830) + 1. Per concludere, vi forniamo in figura 6 un programmino che permette di sostituire durante l'esecuzione un'intera funzione. Noi abbiamo usato il programma per calcolare il valore dell'ordinata della funzione data l'ascissa ma sostituendo o aggiungendo le linee opportune si può far svolgere qualsiasi calcolo. Dopo il Run, voi dovrete solo introdurre la funzione di cui si vogliono calcolare i punti in maniera intellegibile al computer cioè come X*X oppure X*X+3*X+2 ecc.

Per questo mese ci fermiamo qui ma vi aspettiamo la prossima volta per vedere insieme come codificare delle schermate per renderle facilmente interpretabili e come queste possano essere salvate su nastro o disco. MC

```
5 POKE830,0:GOTO70
6 PRINT"■"
7 PRINTA;"PRINT"CHR$(34)"CASA"CHR$(34)
20 POKE631,145:POKE632,145
21 POKE633,145:POKE634,13
22 POKE635,71:POKE636,111
23 POKE637,55:POKE638,48
24 POKE639,13:POKE198,9:END
70 GETA$:IFA$=""THEN70
75 POKE830,PEEK(830)+1
80 A=PEEK(830):IFA>3THENEND
90 GOTO6
```

Figura 5 - Dopo il Run, premendo per tre volte un tasto qualsiasi vengono introdotte le tre linee desiderate.

```
5 PRINT"■"
10 REM ------------------------------
20 REM --- CALCOLO DELL'ORDINATA ---
23 REM ---   DI UNA  FUNZIONE    ---
27 REM ---    DATA L'ASCISSA     ---
30 REM ------------------------------
35 PRINT
40 INPUT"FUNZIONE";F$
50 PRINT"■99 DEF FNF(X)= ";F$
60 POKE631,19:POKE632,13:POKE633,71
65 POKE634,207:POKE635,57
70 POKE636,57:POKE637,13:POKE198,7:END
99 REM QUESTA LINEA SARA' SOSTITUITA
100 INPUT"ASCISSA";X
110 Y=FNF(X)
115 PRINT"■ORDINATA■";Y
120 IFX=999THEN40
130 GOTO100
```

Figura 6 - Durante questo programma, digitando 999 si passa all'introduzione di una nuova funzione.