VIC da zero +64

di Tommaso Pantuso

Generatore di caratteri e modifica dei registri

La volta scorsa abbiamo visto come il sistema operativo sia in grado di risalire alla forma di un carattere partendo dal codice scritto in una locazione della memoria video (di schermo).

Oggi, come promesso, approfondiremo un po' l'argomento gettando le prime basi utili all'apprendimento del modo in cui leggere il contenuto delle mappe dei caratteri del C 64 e del VIC 20 per poi modificarle, al fine di generare dei caratteri personalizzati. Faremo inoltre vedere come sia possibile manipolare uno o più bit di una certa locazione di memoria "in maniera ponderata", cioè agendo in modo da cambiare solo i bit che ci interessano e lasciando invariato il resto della locazione: tutto questo, senza conoscere a priori il contenuto di quella locazione.

La mappa dei caratteri

Come abbiamo visto, la mappa (generatore) dei caratteri contiene, opportunamente codificate, le forme con le quali ciascun carattere, in base al codice ad esso assegnato, viene riprodotto sullo schermo. Ogni forma è contenuta in otto byte e l'intera mappa occupa, nella memoria del computer (C 64 e Vic) 4096 byte. Dato che per ogni carattere vengono spesi 8 byte, non è difficile concludere che il numero di forme stivate in questa specie di magazzino sono 4096/8 = 512 (non tutte differenti). Esse, nella Rom che nel C 64 è situata a partite dalla locazione 53248 e nel Vic 20 dalla 32768, sono così suddivise (figura 1):

	C 64	Vic 20	caratteri
1	53248	32768	Maiuscoli
	53760	33280	Grafici
	54272	33792	Maiuscoli reverse
	54784	34304	Grafici reverse
2	55296	34816	Minuscoli
	55808	35328	Maiuscoli/Grafici
	56320	35843	Minuscoli reverse
	56832	36355	Maiuscoli/Grafici reverse

Nella tabella abbiamo indicato i punti di partenza di ciascun gruppo di caratteri. Come non è difficile osservare, ciascun gruppo dista dall'altro 512 byte quindi, in ognuno di essi, sono contenuti 512/8 = 64 caratteri.

Se ora fate un po' di conti, tenendo presente che un carattere può avere un codice rappresentativo — nella memoria di schermo — contenuto tra 0 e 255, riscontrerete un'apparente incongruenza tra il numero di caratteri (512) indicati nella tabella ed il numero di codici disponibili. In altre parole, se con la prima gamma di codici viene coperto, ad esempio, l'intervallo di caratteri da 53248 a 55296, come è possibile accedere agli altri? La risposta è semplice. Se consideriamo il generatore composto da due grandi insiemi, come indicato a fianco della tabella con i numeri 1 e 2, si deduce che non è possibile accedere contemporaneamente ad essi quindi c'è bisogno di un "deviatore" che permetta di passare da un insieme all'altro. Il passaggio comunque è molto semplice e si ottiene ad esempio con la pressione contemporanea del logo Commodore (in basso a sinistra) e del tasto Shift oppure fornendo il codice di controllo CHR$(14) (vedi figura 2). Esistono altri modi per effettuare la commutazione, ma di essi non ci occuperemo per il momento. Come ultima cosa osserviamo che ciascuno degli insiemi da 2048 byte del generatore, può a sua volta essere suddiviso in due sottoinsiemi: ciascuno di essi contiene gli stessi caratteri con la sola differenza che, nel secondo sottoinsieme di ogni insieme con codici maggiori di 127, si ottengono caratteri in campo inverso.

Manipolazione dei registri

Visto che per ridefinire dei caratteri, sia sul C 64 che sul Vic, dovremo modificare il contenuto dei registri caratteristici di alcuni chip specializzati, esaminiamo un po' di teoria riguardante il modo migliore per

Figura 1 - Suddivisione dei vari tipi di caratteri all'interno del generatore.

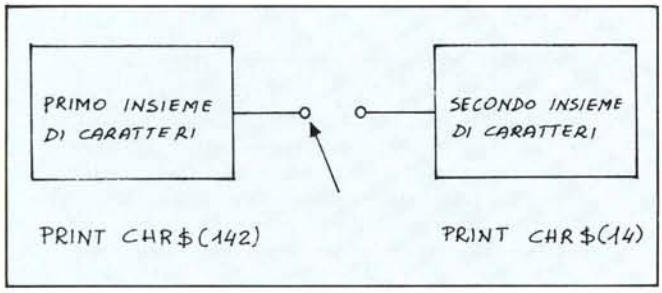

Figura 2 - Commutazione tra i due macroinsiemi di caratteri.

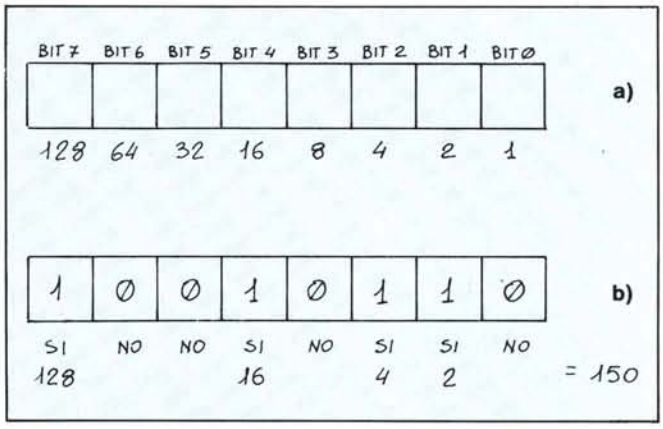

Figura 3 - Esempio di conversione di un numero da binario a decimale.

effettuare queste modifiche. La prima cosa da sapere, per comprendere quanto diremo, è la corrispondenza esistente tra i numeri in notazione decimale (che possiamo leggere con Peek) contenuti nella memoria del computer e gli stessi in notazione binaria.

Per prima cosa, in ciascuna locazione di memoria del C 64 o del Vic 20, il numero massimo che vi si può memorizzare è 255. Ciò equivale a dire che ciascuna locazione occupa 8 bit (un byte, capirete meglio tra breve), cioè in essa è possibile memorizzare un numero binario composto da otto bit dove, ciascun bit, è l'unità elementare di un numero binario e può essere 0 o 1. Senza addentarci troppo in questo tipo di teoria, facciamo subito un esempio che chiarisca le idee. Osservate la costruzione fatta nella sezione a della figura 3. Lì abbiamo rappresentato ciò che dovrebbe essere una locazione di memoria del nostro computer. Sopra la "locazione" abbiamo indicato la posizione di ciascun bit che va da zero, per il primo, a sette, per l'ottavo. Sotto abbiamo riportato dei numeri (potenze di 2) da 1 a 128. Se ora prendiamo il nostro numero binario, composto da una certa quantità di "0" e di "1", e lo inseriamo nelle caselline, come indicato nella sezione b della stessa figura, e poi sommiamo tutti i numeri (da 1 a 128) che compaiono sotto gli "1" — ignorando quelli che stanno sotto gli "0" — , otteniamo il numero decimale corrispondente a quel numero binario. Nel caso della figura, il numero decimale corrisponde al binario 10010110 è 150. Per il procedimento inverso, cioè la trasformazione da binario a decimale non daremo un metodo, essendo molto facile, con un po' di pratica, ottenere la corrispondenza sostituendo degli "1" e degli "0" nelle varie caselline facendo qualche tentativo fino ad ottenere il risultato voluto.

Le prime cose da sapere

Detto ciò, vediamo come agire, con l'uso di comandi di Peek e Poke, sui singoli bit di un registro modificando solo quelli che ci interessano. La prima cosa da fare è quella di definire due operazioni, And e Or, tra due numeri binari. Diamo subito le modalità con cui svolgere le suddette operazioni e facciamo poi qualche esempio sul loro uso. Esse sono definite dalle seguenti tabelle:

AND			OR		
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

Facciamo subito un esempio (figura 4). Prendiamo due numeri binari, 10001001 e 01101110, ed eseguiamo l'And e l'Or tra

Figura 4 - Esempio di And tra due numeri. Notate che si ottiene come risultato "0" quando almeno uno degli operandi è "0".

Figura 5 - Esempio di Or tra due numeri. Notate che si ottiene come risultato "1" quando almeno uno degli operandi è "1".

Figura 6 - In questa figura è illustrato come modificare, ponendolo a zero, un certo numero di bit di una locazione di memoria di cui non si conosce a priori il contenuto. Naturalmente i bit non interessati all'operazione non subiscono alcuna modificazione.

Figura 7 - È la stessa logica di quanto esposto nella didascalia della figura 6. In questo caso viene esposto il modo con cui porre a "1" un certo numero di bit.

di essi dopo averli incolonnati l'uno sotto l'altro. Per fare l'And, rispetteremo le regole imposte dalla prima tabella, cioè quando si troveranno ad esempio incolonnati due "0" oppure un "1" e uno "0", daremo come risultato "0" mentre, trovando due "1", attribuiremo come risultato "1": questo per ogni coppia di bit. Per aver immediatamente le idee chiare, osservate quanto segue:

```
  10001001
  01101110
-----------
And 00001000
```

I numeri cha abbiamo sottoposto all'operazione di And, in notazione decimale, sono rispettivamente 137 e 110 ed il risultato ottenuto è 8. Potete rendervene conto effettuando la conversione mediante il procedimento descritto poc'anzi. Se ora vi mettere un attimo alla tastiera del computer e fate eseguire:

```
PRINT 137 AND 110
```

otterrete appunto 8 come risultato.

In modo analogo ci comporteremo (figura 5) se vogliamo calcolare l'Or tra il primo ed il secondo numero, seguendo le indicazioni date dalla seconda tabella:

```
  10001001
  01101110
-----------
Or 11101111
```

Il risultato, 11101111, come potete facilmente verificare è il corrispondente del numero decimale 239, lo stesso numero che otterrete con:

```
PRINT 137 OR 110.
```

Manipolazione di una locazione.

Supponiamo ora (figura 6), avendo il numero 10001001 (137), — che potrebbe benissimo essere il contenuto di un registro di memoria — di voler azzerare i suoi primi 4 bit, quelli da 0 a 3. La cosa è molto semplice: basta fare l'And del numero in questione, 10001001, con un altro che abbia i primi quattro bit contenenti "0". In altre parole faremo:

```
  1000 1001
  1111 0000
-------------
And 1000 000
```

Come è facile verificare, sono cambiati solo i bit che intendevamo modificare e nessun altro. Se invece, ad esempio, avessimo voluto modificare il primo e l'ultimo bit, avremmo dovuto utilizzare un numero contenente degli "0" solo in queste posizioni, cioè: 01111110. Ritornando al primo caso, il numero impiegato per azzerare i primi quattro bit, in notazione decimale equivale a 240 ed il risultato dell'And è 128. Per verificarlo, scrivete:

```
PRINT 137 AND 240
```

e dopo il <Return> otterrete appunto 128.

Prendiamo ora il numero ottenuto (figura 7) 10000000 (128), e supponiamo di voler porre ad "1" i bit 1 e 2. Avrete capito che, questa volta, dovremo usare l'altra operazione, l'Or, impiegando un numero binario che ora contenga degli "1" in corrispondenza dei bit da "alzare", cioè mettere in stato "1". L'operazione sarà la seguente:

```
  10000000
  00000110
-----------
Or 10000110
```

dalla quale non è difficile osservare che, effettivamente, sono stati modificati solo i bit 1 e 2, essendo rimasti invariati gli altri. Anche in questo caso, possiamo effettuare l'altra verifica, cioè quella ottenuta eseguendo l'Or tra decimali. Questa volta il primo numero è 128, il secondo 6 e il risultato, 128 Or 6, è 134.

Apprese queste semplici nozioni, abbiamo tutti gli elementi necessari per poter manipolare a nostro piacimento qualsiasi locazione di memoria "senza conoscerne il contenuto" a priori ed è questo l'aspetto importantissimo di tutta la faccenda. Infatti, se vogliamo ad esempio azzerare il bit 3 (ricordare che i bit si contano partendo da 0) della locazione N, senza curarci di cosa essa contenga, prenderemo il numero 11110111, che in decimale equivale a 247, e scriveremo:

```
POKE N, (PEEK(N)AND247)
```

sicuri di ottenere l'effetto desiderato. Allo stesso modo, se volessimo essere sicuri che lo stesso bit della stessa locazione sia posto a 1, potremmo eseguire l'Or del contenuto di N con 00001000 (8), cioè:

```
POKE N,PEEK(N)OR8.
```

La prossima volta vedremo delle applicazioni pratiche di quanto appreso in questo articolo. MC