Commodore 64:

CARATTERI UTENTE & SPRITE

Quello della grafica e, più in generale, della presentazione delle informazioni sullo schermo è un problema che interessa sempre gli utenti di una macchina. Questo articolo è dedicato a quei possessori di un Commodore 64 che vogliono saperne di più sulla definizione dei caratteri e sui cosiddetti "sprite", fondamentali nella realizzazione di qualsiasi gioco che comporti l'automazione (veloce) di figure sullo schermo.

di Andrea de Prisco e Leo Sorge

La sezione video del 64

Il circuito integrato che gestisce la sezione video del Commodore 64 è il 6567 — in Europa: negli States è 6566 —, detto VIC II in conseguenza del fatto che il 6561 usato nel VIC 20 era il primo Video Interface Chip. Il 6567 si programma tramite 47 registri ad 8 bit, mappati a partire dalla locazione decimale 53248 (D000 in esadecimale), che consentono di abilitare ben 7 modi di funzionamento:

(1) caratteri standard;
(2) caratteri multicolor;
(3) caratteri extended background color;
(4) alta risoluzione bit-map;
(5) alta risoluzione bit-map multicolor;
(6) gestione sprite;
(7) gestione sprite multicolor.

Sostanzialmente abbiamo a disposizione una pagina testo con 3 varianti (normale, multicolor e multicolor esteso), una pagina in alta risoluzione con 2 varianti (a mappa semplice e multicolor) ed infine 2 tipi di sprite, ai quali dedichiamo la seconda parte dell'articolo e quindi adesso non ci soffermeremo nel guardarli.

La pagina testo

In pagina testo ogni carattere va inteso come una griglia di 8×8 punti; i parametri in gioco per determinare quanta memoria è necessaria per ogni carattere sono due, la risoluzione grafica e quella cromatica (= dei colori). Un punto sullo schermo può essere visibile o no, quindi serve almeno 1 bit per ognuno, ma può esser colorato in una tonalità a scelta tra diverse possibilità: il numero di bit per ogni punto raddoppia ogni due scelte di colore disponibili.

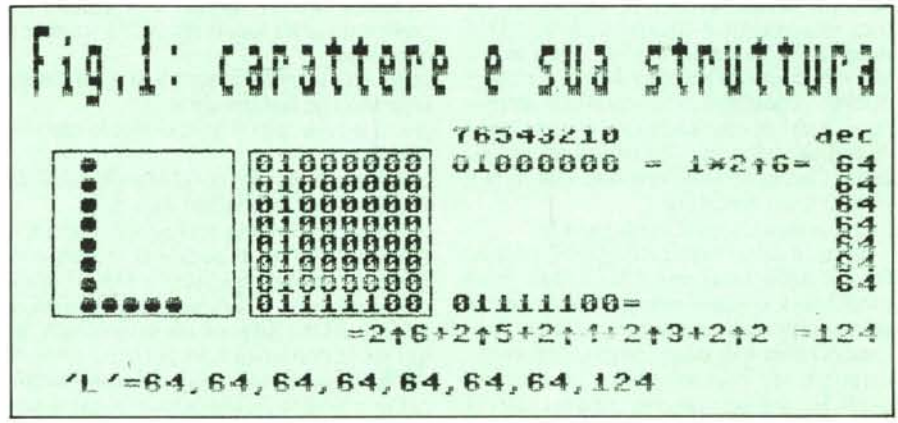
Nel modo testo abbiamo un solo colore possibile per i punti accesi, ai quali corrisponde un bit in memoria, per cui la definizione di un carattere richiede $8 \times 8/8 = 8$ byte (sembra un gioco!). Alcuni di questi vengono resi visibili (mettendo un 1), mentre gli altri assumono il colore dello sfondo (mettendo uno 0): per avere i dati relativi a quel carattere basterà prendere le otto righe di dati binari e convertirle in otto numeri interi decimali; tutto ciò è mostrato in figura 1. Il modo multicolor permette di usare 3 tonalità — selezionabili tra 8 — per i punti accesi; poiché aumenta la risoluzione di colore si dimezza la risoluzione grafica, mettendo a disposizione una griglia 4×8 i cui punti orizzontali sono però lunghi il doppio di quelli normali, per cui lo spazio occupato rimane lo stesso. Il 6567 prende allora le combinazioni di bit, a due a due, e le interpreta come codice del colore di quel punto lungo; tutto ciò è mostrato in figura 2.

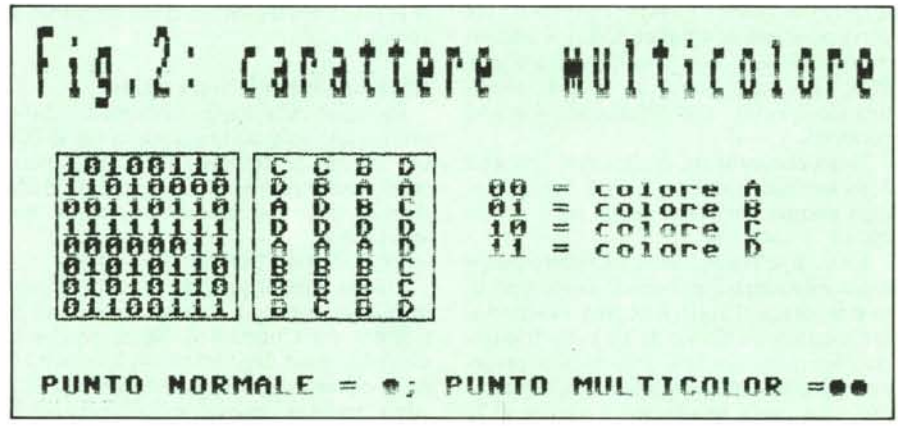
Anche con il multicolor rimane il fatto che la parte della griglia che non viene accesa, ovvero lo sfondo del carattere, viene colorato nella stessa tinta dello sfondo dello schermo: questa situazione è risolta dal modo a colore di sfondo esteso (extended background color), che permette proprio di colorare lo sfondo del carattere in modo diverso da quello dello schermo. La risoluzione grafica è quella massima, 8×8, e quella cromatica, che consente la scelta tra 4 tonalità, viene fatta a discapito del numero di caratteri che possono esser mostrati, che scende a 64.

Il Multicolor e l'Extended Background Color non possono essere usati allo stesso tempo.

Definiamo i nostri caratteri

Il 6567, per le sue molteplici funzioni, ha accesso a ben 16K byte di memoria. Poiché abbiamo 64K byte di RAM, corrisponden-





ti a 4 blocchi da 16K l'uno, il VIC II può scegliere uno di questi 4, che iniziano alle locazioni 0 (esadecimale 0000), 16384 (\$ 4000), 32768 (\$8000) e 49152 (\$ C000): la selezione va fatta agendo sui registri del secondo 6526 del 64, il CIA n. 2.

Prima di iniziare abbiamo bisogno di spazio in RAM per metterci il nostro set di caratteri. Per far ciò dovremo quindi sfruttare ancora l'istruzione POKE:

```
POKE 44, 16: POKE 43,1: POKE 4096,0: NEW
[return]
```

E adesso veniamo al grosso del lavoro. La prima operazione da compiere è di dire al CIA che gli daremo dei dati su due bit; la ricezione viene abilitata con un

```
POKE 56578, PEEK (56578) OR 3
```

che pone ad 1 i primi due bit senza modificare gli altri. A questo punto potremo selezionare il banco da 16K, agendo sulla locazione 56576 nel seguente modo:

```
POKE 56576, PEEK((56576) AND 252) OR A
```

sapendo che

```
A=0 seleziona il banco 3, C000-FFFF
A=1      "          2,  8000-BFFF
A=2      "          1,  4000-7FFF
A=3      "          0,  0000-3FFF
```

e che all'accensione viene usato il banco 0 (valore di default).

Il modo di funzionamento standard, presente all'accensione, mette a disposizione tutti i caratteri disponibili sul 64: maiuscoli, minuscoli, grafici, inversi etc, mappati in opportune locazioni della ROM interna. Come abbiamo visto, ogni carattere è descritto da 8 numeri interi — minori di 256 — contenuti in 8 locazioni di memoria successive; ogni carattere ha un numero di codice, cui si fa riferimento per individuare l'inizio della descrizione, che starà a partire dalla locazione

```
(inizio mappa) + (cod. schermo) × 8.
```

L'inizio della mappa, codificato nel contenuto della locazione 53272, può esser modificato, consentendoci di sostituire la solita mappa con una a nostra scelta. Solitamente non si desidera rimpiazzare tutti i caratteri, ma solo una parte, ad esempio quelli in carattere inverso: in quel caso la nuova mappa sarà realizzata andando a leggere (con delle PEEK) i gruppi di 8 byte corrispondenti ai soliti caratteri, e andando a porli nella nuova mappa (tramite le POKE), per poi completare l'opera modificando i valori che definiscono i nostri caratteri.

Dato che ogni set di caratteri contiene 256 elementi, come visto di 8 byte l'uno, ogni mappa dovrà estendersi per

```
256 × 8 = 2048 byte
```

Le mappe del 64 sono come detto due, e sono selezionabili premendo contemporaneamente i tasti SHIFT e CBM. Noi potremo scegliere un blocco da 2K byte all'interno della zona da 16K prescelta in precedenza agendo sulla locazione 56576 (sempre dopo aver specificato il valore della

```
1 FOR T=704TO752:READA:POKET,A:NEXT
2 DATA 0,0,0,255,0,255,255,123,255,15
3 DATA 199,224,3,199,128,0,238,0,0,238
4 DATA 0,0,238,0,0,255,0,1,255,128,3
6 DATA 147,192,7,147,224,15,147,240,31
8 DATA 239,248,31,125,240,15,131,240,7
9 DATA 131,224,3,131,192,1,255,128,0,127
```

Listato A

locazione 56578): questo si fa agendo sul contenuto della locazione 53272 tramite il comando

```
POKE 53272, (PEEK (53272) AND 240) OR B)
```

sapendo che la formula è

```
(inizio blocco 2K) = (inizio blocco 16K) +
1K × B;
```

ovviamente, poiché ci servono blocchi da 2K, useremo solo valori pari di B.

Colpo di scena! Se tra i banchi da 16K si seleziona quello di codice 0, o quello di codice 2, e si sceglie a \$1000 e \$1800 l'inizio del generatore di caratteri, si ha l'immagine della ROM già presente nella macchina, quindi in definitiva non potremo alterare nulla. Per usare una mappa RAM, modificabile a nostro piacimento, bisognerà scegliere inizi diversi da quei due valori, oppure tra i blocchi da 16K selezionare quello di codice 1 o 3.

Interrompiamo le interruzioni

Per agire sulla mappa di memoria, differentemente da quanto accadeva sul VIC, il 64 richiede la disabilitazione degli interrupt: questa può esser realizzata da Basic tramite un'opportuna istruzione di mascheratura:

```
10 POKE 56334, PEEK (56334) AND 254.
```

Questa riga di programma, come pure quelle che seguono, vanno messe in un listatino con i numeri di linea, poiché la disabilitazione degli interrupt ha come effetto collaterale la sconnessione della tastiera, il che rende ovviamente impossibile continuare a digitare le istruzioni in modo diretto. Inoltre bisognerà momentaneamente sconnettere l'intero blocco della zona di Input-Output, con una

```
20 POKE 1, PEEK (1) AND 251;
```

entrambe queste funzioni andranno riabilitate quando avremo finito di trasferire la nostra mappa.

Trasferiremo nella nuova RAM il contenuto della solita ROM, in modo da avere un set di caratteri in RAM: successivamente modificheremo parzialmente questo set. Per copiare la ROM in RAM basterà la linea

```
30 FOR T=0 TO 2047: POKE 2048+T, PEEK
(53248+T): NEXT
```

e poi dovremo invertire le operazioni sugli interrupt e sulla selezione del banco I/O:

```
40 POKE 1, PEEK (1) OR 4: POKE 56334, PEEK
(56334) OR 1.
```

Adesso abbiamo un completo set di caratteri, quello di maiuscole, a partire dalla locazione 2048 fino alla 4095: per stabilire da dove partono i dati del carattere che andremo a modificare vale la regola

```
(indir. inizio mappa) × (codice di schermo del
caratt.) × 8;
```

e i codici di schermo sono a pag. 132 del manuale in inglese.

I dati del carattere di codice 0 (la chiocciola) partiranno dunque dalla locazione di inizio della mappa, cioè da 2048. Per modificarli dovremo sostituircene di nuovi: ad esempio, per mettere una nota musicale al loro posto, come si capisce dalla figura 3 dovremo far girare il seguente programmino (anche stavolta con i numeri di linea per la presenza dei READ-DATA):

```
50 FOR T=2048 TO 2048+7
60 READ G: POKE T,G
70 NEXT
90 DATA 24,20,20,18,48,112,96,0
```

e adesso, premendo la chiocciola, avrete il grazioso simbolo musicale.

Il Multicolor

Ripetiamo che il funzionamento a caratteri multicolor differisce da quello standard basilarmente perché la corrispondenza tra bit della mappa e punti dello schermo non è 1 a 1, bensì di 1 a 2: ogni punto sullo schermo viene identificato da due bit consecutivi della mappa, che contengono il

```
5 POKE53280,3:POKE53281,1:FORI=49152TO49185:READII:POKEI,II:NEXT
10 PRINT"■        ■■■■  *** CHAR - EDITOR ***  ■"
20 PRINT"■■         ■1■ MAPPA RAM"
30 PRINT"■          ■2■ EDITOR"
40 PRINT"■          ■3■ SAVE MAPPA"
50 PRINT"■          ■4■ LOAD MAPPA"
60 PRINT"■■         ■  (C) 1984 ADP-SOFTWARE  ■"
70 GET#8:IFA$<"1"ORA$>"4"THEN70
80 ONVAL(A$)GOTO100,150,500,700
81 ************************************
82 *                                  *
83 *  C H A R  -  E D I T O R   6 4   *
84 *                                  *
85 *   (C)   1984   ADP - SOFTWARE    *
86 *                                  *
87 ************************************
88 *                                  *
89 *  PRIMA DI DIGITARE O CARICARE    *
90 *   QUESTO  PROGRAMMA, ESEGUIRE    *
91 *                                  *
92 *   POKE 44,16:POKE 4096,0:NEW     *
93 *                                  *
94 * PER SPOSTARE L'INIZIO PROGRAMMI  *
95 *    BASIC, VARIABILI E ARRAY,     *
96 *                                  *
99 ************************************
100 POKE53272,(PEEK(53272)AND240)OR2
105 POKE56334,PEEK(56334)AND254
110 POKE1,PEEK(1)AND251
115 SYS49152
120 POKE1,PEEK(1)OR4
125 POKE56334,PEEK(56334)OR1
130 GOTO70
150 INPUT"■■S■TANDARD O ■M■ULTICOLOR";A$:IFA$<>"S"ANDA$<>"M"THEN150
160 T=(A$="S")+2:INPUT"■CARATTERE DA MODIFICARE ";A$:PRINT"■■";A$:"■"
170 C=PEEK(1024):D=ASC(A$):CO=0:RI=0:PU=1024
180 PRINT"■          ■"
185 PRINT"          ■  F1 - BLANK      "
190 PRINT"          ■  F3 - COL 1      "
195 PRINT"          ■  F5 - COL 2      "
200 PRINT"          ■  F7 - COL 3      "
205 PRINT"          ■"
210 PRINT"          ■  [RET] MEMORIZZA "
215 PRINT"          ■"
225 PRINT"■■■■■■■■■■"
230 FORI=0TO7:A$(I)="00000000":NEXT
250 GET#8:POKEPU,PEEK(PU)+128*((PEEK(PU)>127)+.5)*2:IFA$=""THEN250
255 POKEPU,PEEK(PU)AND127
260 IFA$="■"THEN:GOSUB400:CO=CO+T:GOSUB470
265 IFA$="■"THENA$="01":GOSUB410:CO=CO+T:GOSUB470
270 IFA$="■"THENA$="10":GOSUB410:CO=CO+T:GOSUB470
275 IFA$="■"THENA$="11":GOSUB410:CO=CO+T:GOSUB470
280 IFA$="■"THEN170
285 IFA$="■"THENRI=0:CO=0:PU=1024
290 IFA$="■"THENRI=RI+1:GOSUB470
292 IFA$="■"THENRI=RI-1:GOSUB470
294 IFA$="■"THENCO=CO-T:GOSUB470
295 IFA$="■"THENCO=CO+T:GOSUB470
297 IFA$=CHR$(13)THEN300
299 GOTO250
300 FORI=0TO7:X=0:FORJ=8TO1STEP-1
310 X=X-(MID$(A$(I),J,1)="1")*2↑(8-J):NEXT
320 POKE2048+C*8+I,X:NEXT:IFT=2THEN340
330 PRINTCHR$(D):WAIT197,191:GETZ$:GOTO10
340 INPUT"COLORE 1";A:POKE53282,A
345 INPUT"COLORE 2";A:POKE53283,A
350 INPUT"COLORE 3";A:POKE55976,A0R8:POKE1704,C
355 POKE53270,PEEK(53270)OR16:FORI=1TO2000:NEXT:WAIT197,191
360 GETZ$:POKE53270,PEEK(53270)AND239:GOTO10
400 IFT=1THENA$(RI)=LEFT$(A$(RI),CO)+"0"+MID$(A$(RI),CO+2):POKEPU,32:RETURN
405 A$(RI)=LEFT$(A$(RI),CO)+"00"+MID$(A$(RI),CO+2):POKEPU,32:POKEPU+1,32
408 RETURN
410 IFT=1THENA$(RI)=LEFT$(A$(RI),CO)+"1"+MID$(A$(RI),CO+2):POKEPU,81:RETURN
415 A$(RI)=LEFT$(A$(RI),CO)+A$+MID$(A$(RI),CO+2):A=VAL(A$)+2
420 POKEPU,81:POKEPU+54272,A:POKEPU+1,81:POKEPU+54273,A:RETURN
470 IFCO=8THENCO=0:RI=RI+1
475 IFCO<0THENCO=CO+8:RI=RI-1
480 IFRI=8THENRI=0
485 IFRI=-1THENRI=7
490 PU=1024+CO+RI*40:RETURN
500 OPEN4,8,4,"CARATTERI,S,W":FORI=2048TO4095:PRINT#4,CHR$(PEEK(I));:NEXT
510 CLOSE4:GOTO10
700 OPEN4,8,4,"CARATTERI,S,R":FORI=2048TO4095:GET#4,A$:IFA$=""THENA$=CHR$(0)
710 POKEI,ASC(A$):NEXT:CLOSE4:GOTO10
1000 DATA169,0,133,252,133,254,169,208,133,255,169,8,133,253,160,0,177
1010 DATA254,145,252,200,208,249,230,255,230,253,165,255,201,215,208,239,96
```

codice del colore con cui si vuole visualizzarlo, mentre le dimensioni raddoppiano in orizzontale per poter coprire la stessa superficie. Quindi, ad esempio, considerando la seguente riga di 8 cifre binarie:
00100111
come una delle 8 righe che definiscono la matrice di un carattere in multicolor, avremo la seguente disposizione grafica e cromatica:
AABBCCDD
ove A, B, C e D sono i 4 colori di codici rispettivi 00 (sfondo), 10, 01, 11, ognuno considerato per due punti elementari successivi. Il risultato può esser considerato come una griglia 4×8 in cui i singoli elementi hanno la dimensione orizzontale raddoppiata (vedi sempre fig. 2).

Il modo multicolor, come al solito, va abilitato, e la locazione su cui agire è la 53270:

```
POKE 53270, PEEK (53270) OR 2↑4.
```

Quando si vorrà tornare al modo standard si dovrà rimettere tutto a posto, usando quindi la seguente istruzione:

```
POKE 53270, PEEK (53270) AND (255 - 2↑4).
```

Tornando alla gestione di questo tipo di visualizzazione, i quattro codici di colore sono contenuti nelle locazioni che partono da 53281: quello dello sfondo (o di codice 00) sta proprio nella 53281; quello che corrisponde a 01 in 53282; quello di 10 in 53283; quello di 11 in 53284. Abbiamo a disposizione 8 colori, che si abilitano modificando i tre bit meno significativi delle locazioni citate; il quarto bit andrà comunque posto ad 1, mentre i quattro più significativi sono tenuti alti (pari ad 1) dal sistema stesso. Per questo motivo se andiamo a leggere (con una PEEK) il contenuto di queste locazioni, troviamo sempre un numero maggiore di 240, anche se ci andiamo a mettere (tramite una POKE) un valore diverso: il sistema considererà solo i bit più bassi, quindi

```
POKE 53281, 13
```

porterà in lettura, tramite una

```
PRINT PEEK (53281)
```

il valore $240 + 13 = 253$.

Un editore di caratteri

Prima di caricare il programma, sia da nastro che da disco, è necessario spostare l'inizio della memoria destinata al Basic, onde creare lo spazio necessario alla nuova mappa di caratteri in RAM. Ciò avviene — come visto — con la seguente sequenza di comandi:

```
POKE 44,16: POKE 43,1: POKE 4096,0: NEW [return],
```

di modo che lo spazio di memoria tra 2048 e 4095 (2K byte) ospiterà la citata mappa.

Al RUN farà seguito un menu di 4 opzioni:

(1) trasferimento in RAM della ROM, e sua attivazione (in linguaggio macchina);

(2) scelta del tipo di carattere (quale,

standard o multicolor), e successiva gestione di una griglia 8 × 8, i cui caratteri corrispondono ai bit della mappa;

(3) salvataggio della mappa su memoria di massa;

(4) caricamento della mappa da memoria di massa.

Aggiungiamo alcune note sulla seconda opzione. Quando si è in ambiente di progetto del carattere, ovvero all'interno della griglia 8 × 8, ci si muove sfruttando gli usuali tasti del cursore. La determinazione dei punti accesi viene fatta con i tasti funzione sulla destra: in modo standard, F1 corrisponde ad un punto non acceso (colore dello sfondo), mentre F3 è un punto visibile (il cui colore, lo ricordiamo, è contenuto in 53281); in multicolor F3, F5 ed F7 scelgono il colore del punto (doppio) da accendere, mentre F1 continua a cancellare.

Gli sprite

Gli sprite, al maschile (ed invariante nel numero) come vuole la lingua italiana per le parole straniere di uso corrente, sono griglie da 24 × 21 punti completamente indipendenti dalla schermata sottostante, sia questa di testo o in alta risoluzione. Il manuale della Commodore, almeno quello inglese, riporta a pag. 70 una tipica griglia per la costruzione degli sprite: per ogni punto che si desidera riempire viene impiegato 1 bit (quindi per l'intera griglia serviranno 24 × 21/8 = 63 byte), e bisogna porre un 1 nella griglia; terminato il disegno basterà prendere 8 per volta i 24 × 21 = 504 valori binari, partendo in alto a sinistra, e convertirli in 504/8 = 63 numeri decimali (magari usando il semplice programmino indicato, sempre sul manuale inglese, a pag. 78) che andranno immagazzinati in opportune zone di memoria, come vedrete nel seguito. Ammettendo di poterli mettere a partire dalla locazione decimale 704, con riferimento al listato A (pag. 100), abbiamo ora da qualche parte un grazioso coniglietto. Per usarlo, seguiteci nel resto dell'articolo.

Il 64 mette a disposizione 8 sprite contemporaneamente sullo schermo, ma noi possiamo definirne quanti ne desideriamo, ed eventualmente selezionare i dati spostando i puntatori della zona di memoria riservata.

Sugli sprite possiamo compiere diverse operazioni:

(1) di inizializzazione;
(2) di movimento;
(3) di controllo.

(1) Inizializzazione

Ogni griglia occupa 63 byte; per semplicità di calcolo, alla fine di ogni gruppo di dati si lascia un byte vuoto, così da avere tutti i gruppi di dati ogni 2↑6 = 64 locazioni di memoria, a partire da un valore prestabilito che si ottiene dividendo per 64 la locazione di partenza: questo numero va messo, con una POKE, nella locazione 2040 per il primo sprite (quello di ordine 0), nella 2041 per il secondo, e così via fino alla locazione 2047 che contiene l'analogo valore per l'ottavo sprite. Il sistema operativo ha alcune zone vuote, che permettono di allocare un numero limitato di sprite (4) senza usare la memoria accessibile da Basic. Questi buffer sono nelle seguenti locazioni: 704-766 (il valore da porre nel registro puntatore è 704/64 = 11); 832-894 (13); 896-958 (14); 960-1022 (15).

(2) Movimento

Per dire al 64 che lo sprite va considerato inserito e che quindi deve visualizzarlo, bisogna agire direttamente sugli opportuni registri del chip video, il 6567, che partono dalla locazione v = 53248. Quelli che ci interessano comandano l'abilitazione, la posizione, la dimensione e gli urti sia tra sprite che con lo schermo per ognuna delle otto griglie. È evidente che nel caso di condizioni a due possibilità, come l'abilitazione o gli (eventuali) urti, basterà un solo bit per ogni sprite, e quindi per tutti e 8 basterà un byte. Per le coordinate, dato che il 64 in

```
10000 REM *********************
10010 REM **  DISEGNA SPRITE  **
10020 REM *********************
10100 IF PEEK(44)=16ANDPEEK(43)=1ANDPEEK(4096)=0THEN GOTO 12000
10200 POKE 53280,10:POKE53281,2
10210 PRINTTAB(210)"ATTENZIONE!"
10220 PRINT"DIGITARE POKE44,16:POKE43,1:POKE4096,0:NEW"
10230 PRINT"E RILEGGERE IL PROGRAMMA"
10240 STOP:END
12000 REM*************************
12002 REM AZIONA PUNTATORI SPRITE
12005 REM*************************
12010 SP=2040:SE=53269:SS=2048
12020 DEF FNS(SN)=SS+64*(SN)
12030 SN=0:DIM TT%(20,22):POKE53281,6
13000 REM*************************
13002 REM  DISEGNA LA GRIGLIA
13005 REM*************************
13020 POKE53269,1:POKE53287,1
13030 POKESP,FNS(SN)/64
13040 POKE53248,0:POKE53264,1:POKE53249,90
13050 PRINT"▒";:FORI=1TO11:PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ "
13060 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒":NEXT
13070 PRINT"▒▒                        "
13080 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
13090 PRINT"▒▒";:FORI=FNS(SN)TOFNS(SN)+62STEP3:FORJ=0TO2
13100 FORK=7TO0STEP-1
13110 IF(PEEK(I+J)AND2↑K)=2↑KTHENPRINT"●";
13120 IF(PEEK(I+J)AND2↑K)=0THENPRINT" ";
13130 NEXTK,J:PRINT:NEXT
13140 PRINT"▒▒NUM   PROGETTO":SN
13150 INPUT"DI QUANTO MUOVO IL PUNT.(0=REDEF.)";MM:SN=SN+MM
13160 IF SN< 0 THEN SN=0
13170 IF SN>31 THEN SN=31
13180 IF MM=0 THEN POKE 53248,0:POKE53264,1:POKE53249,200:GOTO 14000
13190 GOTO 13000
14000 REM*************************
14002 REM   RIDEFINISCE  SPRITE
14005 REM*************************
14030 PRINT"▒                                   ▒"
14040 F$="▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
14050 PRINTF$;"▒/I/ INVERTE"
14060 PRINTF$;"▒/M/ SPECCHIO"
14070 PRINTF$;"▒/R/ RETURN"
14080 PRINTF$;"▒/1/ SCRIVE"
14090 PRINTF$;"▒/0/ CANCELLA"
14100 PRINTF$;"▒/T/ GIRA  "
14110 PRINTF$;"▒/P/ MEMORIA"
14112 PRINTF$;"▒/Y/ SCAMBIA"
14114 PRINTF$;"▒/C/ COLORE"
14120 PRINTF$;"▒/S/ SALVA "
14130 PRINTF$;"▒/L/ CARICA"
14140 PRINTF$;"▒/E/ FINE  "
14141 PRINTF$;"▒/F1/Y-ESP "
14142 PRINTF$;"▒/F3/Y-ESP "
14143 PRINTF$;"▒/F5/X-RID "
14144 PRINTF$;"▒/F7/Y-RID "
14150 PRINTF$;"▒CURS. MUOVE"
14160 PRINT"▒";
14170 GET A$
14180 CC=PEEK(211)+PEEK(210)*256+PEEK(209):PP=PEEK(CC):C1=55296+CC-1024
14190 C2=PEEK(C1)
14200 POKECC,42:POKEC1,1
14210 FORI=1TO15:NEXT:POKECC,PP:POKEC1,C2:IFA$=""THEN14170
14220 P1=INT((CC-1024)/40):P2=CC-(1024+40*P1)
14230 IF(P1>0ANDA$="▒")OR(P1<20ANDA$="▒")THENPRINTA$;:GOTO14170
14240 IF(P2>0ANDA$="▒")OR(P2<23ANDA$="▒")THENPRINTA$;:GOTO14170
14250 IFA$="1"THENPRINT"▒●▒";:GOTO14170
14260 IFA$="0"THENPRINT"▒ ▒";:GOTO14170
```

alta risoluzione è organizzato come matrice di 256 punti verticali per 320 punti orizzontali, avremo bisogno di 1 byte per ogni ordinata e di 9 bit per ogni ascissa. Considerando per ora solo una zona utile di 256 × 256 punti (quindi tralasciando il nono bit delle ascisse) per le coordinate abbiamo i seguenti registri:

sprite	asse X	asse Y
0	v	v+1
1	v+2	v+3
...		
7	v+14	v+15

Il byte di abilitazione è in v+21, e ovviamente ogni bit abilita lo sprite di ordine corrispondente.

Sappiamo già quanto basta per vedere qualcosa sullo schermo. Infatti, per realizzare uno sprite dovremo:

(a) destinare 64 byte liberi, dalla locazione Y in poi [Y dev'essere multiplo di 64]: noi l'abbiamo già fatto, caricando i valori da 704 in poi;

(b) dire al 64 dove stanno i dati [con una POKE 2040+n° sprite, Y/64];

(c) abilitare l'oggetto [con una POKE v+21, 2↑(n° sprite)];

(d) dargli delle coordinate in un quadrato 256 × 256, ove il punto che indichiamo va inteso essere quello in alto a sinistra nella griglia [con POKE v+(2 × n° sprite), ascissa; POKE v+(2 × n° sprite)+1, ordinata].

Come esempio pratico abilitiamo lo sprite 3, usando i dati precedentemente posti nel blocco 11, che parte da 704. Con riferimento ai punti da (a) a (d), ponendo Y=704; n° sprite=3, v=53248; ascissa=100; ordinata=100, le istruzioni, da battere in modo diretto, sono:

v=53248 (d'ora in poi non date mai RESTORE né CLR né NEW);
(b) POKE 2040+3, 704/64;
(c) POKE v+21, 2↑(3);
(d) POKE v+(2•3), 200: POKE v+(2•3) +1, 100

e vedrete apparire sullo schermo un coniglietto: quello è uno sprite.

Per cambiare la posizione dell'oggetto basterà mutare i valori messi nelle locazioni di cui al punto (d); comunque non riusciremo a portarlo nella parte più a destra dello schermo. Infatti l'ascissa viene regolata da una strana legge: per ogni griglia è disponibile un bit nella locazione v+16, e se il bit di ordine k vale 1, allora l'ascissa della griglia di stesso ordine viene accresciuta di 256. Quindi per avere lo sprite di ordine 3 nell'ascissa 255 bisognerà fare

POKE v+(2•3), 255

ma per portarlo a 256 bisognerà fare

POKE v+(2•3), 0: POKE v+16, 2↑3.

(3) Controllo

Altre possibilità che ci interessano sono: (e) il colore dello sprite; (f) il colore dello sfondo; (g) la possibilità di raddoppiare le dimensioni; (h) il modo multicolor.

Vediamoli.

(e) Il colore di ogni sprite può essere selezionato ponendo un valore tra 0 e 15 nei registri da v+39 a v+46; per esigenze del sistema i contenuti di questi registri vengono forzati ad assumere un valore pari a 240+(codice colore), per cui ad una scrittura del valore 10 (tramite una POKE v+39+n° sprite, 10) corrisponde una lettura del valore 240+10=250.

(f) Il colore dello sfondo dello schermo viene modificato agendo sul contenuto della locazione v+33 (53281), anche lui forzato ad assumere valori da 240 a 255. In 53280 troviamo invece la chiave per modificare il colore del bordo, sempre alle stesse condizioni. Per verificare tutto questo, provate il seguente programmino (sempre in modo diretto e ricordando che il nostro sprite è quello di n° d'ordine pari a 3):

FOR I=1 TO 10: POKE v+39+3,I : POKE v+32, 15-I : FOR T=0 TO 400 :NEXTT,I

(g) Entrambe le dimensioni delle griglie, poi, possono essere raddoppiate. Come per tutti i parametri a due valori, anche l'eventuale espansione necessita di un bit per

```
14270 IFA$<>"I"THEN14330
14280 FORI=0TO20:FORJ=0TO23
14290 IFPEEK(1024+I*40+J)=32THEN14310
14300 POKE1024+40*I+J,32:POKE55296+40*I+J,180:GOTO14320
14310 POKE1024+40*I+J,81:POKE55296+40*I+J,181
14320 NEXTJ,I:GOTO14170
14330 IFA$="R"THENGOTO13030
14340 IFA$<>"M"THEN14420
14350 FORI=0TO20:FORJ=0TO23:TT%(I,J)=0:NEXTJ,I
14360 FORI=0TO20:FORJ=0TO23
14370 IFPEEK(1024+40*I+J)=81THENTT%(I,J)=1
14380 NEXTJ,I
14390 PRINT"■";:FORI=0TO20:FORJ=23TO0STEP-1:IFTT%(I,J)=1THENPRINT"■■■";
14400 IFTT%(I,J)=0THENPRINT"■ ";
14410 NEXTJ:PRINT:NEXTI:PRINT"■":GOTO14170
14420 IFA$<>"T"THEN14500
14430 FORI=0TO20:FORJ=0TO23:TT%(I,J)=0:NEXTJ,I
14440 FORI=0TO20:FORJ=0TO20
14450 IFPEEK(1024+40*I+J)=81THENTT%(20-I,20-J)=1
14460 NEXTJ,I
14470 PRINT"■";:FORI=0TO20:FORJ=23TO0STEP-1:IFTT%(I,J)=1THENPRINT"■■■";
14480 IFTT%(I,J)=0THENPRINT"■ ■";
14490 NEXTJ:PRINT:NEXTI:PRINT"■":GOTO14170
14500 IFA$<>"P"THEN14560
14510 FORI=0TO20:FORJ=0TO2:TT%(I,J)=0:NEXTJ,I
14520 FORI=0TO20:FORJ=0TO2:FORK=0TO7
14530 IFPEEK(1024+40*I+J*8+K)=81THENTT%(I,J)=TT%(I,J)OR2↑(7-K)
14540 NEXTK,J,I
14550 FORI=0TO20:FORJ=0TO2:POKEFNS(SN)+I*3+J,TT%(I,J):NEXTJ,I:GOTO13030
14560 IFA$<>"S"THEN14620
14570 INPUT"■■■■■■■■■■■■■■■■■■■■■■■■HOW MANY SPRITES TO BE SAVED";NN
14580 IFNN<1ORNN>32THEN14570
14590 OPEN1,1,1,"SPRITES":PRINT#1,NN
14600 FORI=0TONN*64-1:TX=PEEK(SP+I):PRINT#1,TX:NEXT
14610 CLOSE1:PRINT"■"
14620 IFA$<>"L"THEN14660
14630 OPEN1,1,0,"SPRITES":INPUT#1,NN
14640 FORI=0TO64*NN-1:INPUT#1,T:POKESP+I,T:NEXT
14650 CLOSE1:PRINT"■"
14660 IFA$<>"E"THEN14680
14670 POKE53269,0:POKE43,1:POKE44,8:POKE2048,0:CLR:END
14680 IFA$<>"X"THEN14740
14690 INPUT"■■■■■■■■■■■■■■■■■■■■■■■■NUMBER TO EXCHANGE WITH:";S2
14700 IFS2<0ORS2>31THEN14690
14710 FORI=FNS(SN)TOFNS(SN)+62:T1=PEEK(I):POKEI,PEEK(I-FNS(SN)+FNS(S2))
14720 POKEI-FNS(SN)+FNS(S2),T1:NEXT
14730 GOTO13000
14740 IFA$<>"C"THEN14840
14750 IFPEEK(53276)AND1=1THENPOKE53276,0:GOTO14170
14760 INPUT"■■■■■■■■■■■■■■■■■■■■■■■■INPUT COLOR FOR 01 (0-15)";C1
14770 IFC1<0ORC1>15THENPRINT"■                    ":GOTO14760
14780 INPUT"■■■■■■■■■■■■■■■■■■■■■■■■INPUT COLOR FOR 10 (0-15)";C2
14790 IFC2<0ORC2>15THENPRINT"■                    ":GOTO 14780
14800 INPUT"■■■■■■■■■■■■■■■■■■■■■■■■INPUT COLOR FOR 11 (0-15)";C3
14810 IFC3<0ORC3>15THENPRINT"■                    ":GOTO 14800
14820 POKE53285,(PEEK(53285)AND240)ORC1:POKE53287,(PEEK(53287)AND240)ORC3
14830 POKE53286,(PEEK(53286)AND240)ORC2:POKE53276,1:PRINT"■":GOTO14170
14840 IFA$<>"■"THEN14870
14850 POKE53277,255
14860 GOTO14170
14870 IFA$<>"■"THEN14900
14880 POKE53271,255
14890 GOTO14170
14900 IFA$<>"■"THEN14930
14910 POKE53277,0
14920 GOTO14170
14930 IFA$<>"■"THEN14950
14940 POKE53271,0
14950 GOTO14170
```

ogni sprite, ed è quindi un solo byte a contenere tutte le informazioni necessarie; si tratta del registro v+23 = 53271 per l'asse Y, e del registro v+29 = 53277 per l'asse X. Andando a mettere 1 nell'opportuno bit, osserveremo la desiderata espansione; rimettendoci 0, tutto tornerà normale.

Nell'esempio che ancora avete sullo schermo, lo sprite abilitato è il numero 3, quindi dovrete agire come segue:

— POKE 53271, 2↑3 per espandere le ordinate;

— POKE 53277, 2↑3 per espandere le ascisse;

per annullare entrambi gli effetti, fate contemporaneamente

— POKE 53271, 0 : POKE 53277, 0

(h) Il modo multicolor per sprite, come quello per caratteri, mette a disposizione dell'utente 3 colori per ogni sprite, con le stesse regole per la risoluzione grafica, che viene dimezzata, per cui ci viene messa a disposizione una griglia 12 × 21. Il modo multicolor si abilita ponendo a 1 il bit opportuno del registro v+28 (53276).

Rimangono almeno altre due questioni da studiare, ovvero (i) le collisioni, tra sprite e sprite e delle sprite con lo sfondo, ed inoltre la cosiddetta (l) priorità tra sprite.

(i) Trattandosi di condizioni a due possibilità, avremo a disposizione un bit per ogni eventuale collisione: il byte 53278, o v+30, terrà la situazione degli sprite tra loro, ponendo ad 1 gli opportuni bit di entrambe le griglie che vengano in contatto; un altro byte, 53279 o v+31, avrà il compito di segnalare, sempre con degli 1 nei bit corrispondenti, se lo sprite di ordine opportuno sarà andato a sbattere su qualcosa nello schermo (sia in modo testo che in alta risoluzione); in v+27, infine, si trovano gli 8 bit che, se accesi, indicano che gli sprite di ordine corrispondente dovranno passare sopra lo sfondo e non sotto. È interessante notare che il confronto di collisione non viene fatto tra le griglie 24 × 21 (o 12 × 21 nel caso del multicolor), bensì sui soli punti visibili sullo schermo (ovvero nel multicolor per i punti di codice di colore diverso da quello dello sfondo, che è 00). Attenzione al fatto che le collisioni possono avvenire anche fuori dal campo visibile! Quindi fate buon uso delle coordinate.

Per stabilire se avvengono collisioni dovrete usare una verifica di tipo:

IF (PEEK(v+30) AND (2↑SN1+2↑SN2)) = 2↑SN1+2↑SN2;

se questa è vera, c'è contatto tra i punti visibili dello sprite di codice SN1 e quello di codice SN2.

Per verificare la collisione con lo sfondo basterà invece che sia vera la condizione

IF (PEEK(v+31) AND 2↑SN).

(l) Le priorità tra sprite sono automatiche, nel senso che quello a priorità più alta è quello di ordine 0, e ovviamente quello a priorità più bassa è quello di ordine 7. In caso i due sprite passassero per le stesse coordinate, si avrebbe che quello di ordine minore passa sopra a quello di ordine maggiore. È anche interessante osservare che se gli sprite non sono pieni, ovvero se hanno dei buchi, lì si vede quello che passa sotto (sfondo o un altro sprite).

Un editore di sprite

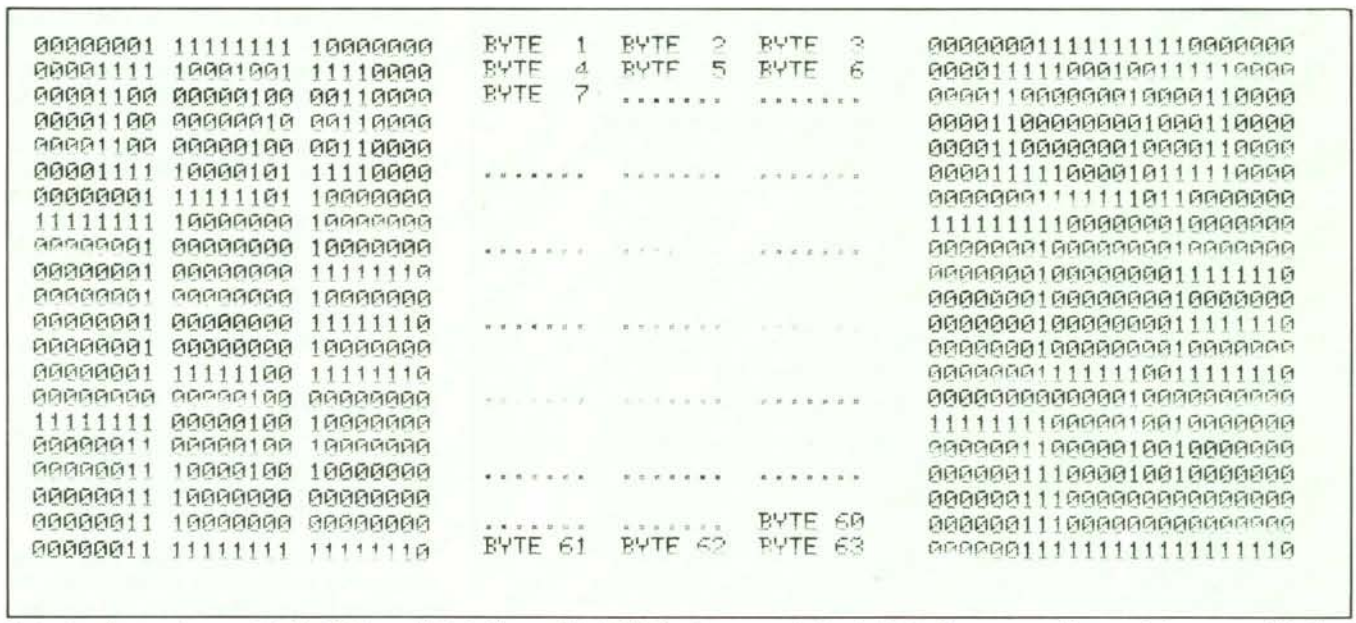
Per finire presentiamo uno sprite editor in Basic che realizza diverse funzioni: in virtù del fatto che memorizza i dati in una matrice di interi (TT%), questo programma compie sugli sprite diverse manipolazioni, consentendo — oltre che di formare l'immagine — di scambiare due sprite, di averne la speculare, l'inversa e quella girata di 90° in senso orario. Può immagazzinare fino a 32 griglie, poste dalla locazione 2048 alla 4095: ricordatevi di modificare la RAM del Basic tramite la

POKE 44,16: POKE 43,1: POKE 4096,0: NEW [return]

altrimenti il programma vi segnalerà l'errore e dovrete rileggerlo con il LOAD.

L'editore consente anche di abilitare il modo multicolor, nonché in ogni momento di espandere o riportare alle normali dimensioni la griglia al momento in opera. Per porre ad 1 un punto della griglia basta premere l'1: per cancellarlo, lo 0. Ci si sposta con il cursore.

Questo programma è direttamente derivato da quello presentato da David Laurence in The Working 64, pgg. 60 e segg. Rispetto a quella versione ci sono alcune modifiche inessenziali, ma anche alcune correzioni fondamentali senza le quali il programma era praticamente quasi inutilizzabile; abbiamo scelto questo perché ci è sembrato particolarmente interessante per alcune sue caratteristiche, come ad esempio la gestione dei dati su cassetta e l'immagazzinamento dei suddetti prima del Basic (cosa che permette di salvarli insieme al programma stesso).



Costruzione di uno sprite: una griglia 24 × 21 viene suddivisa in 63 numeri binari di 8 cifre, che — convertite in valori decimali — vanno messi in memoria in zone prestabilite (leggi articolo) creando una corrispondenza di 1 ad 1 tra punti accesi sullo sprite e bit in memoria.