Grafica avanzata con lo ZX SPECTRUM

S. Nicholls

Grafica avanzata con lo ZX SPECTRUM

Grafica avanzata con lo ZX SPECTRUM

S. Nicholls

McGRAW-HILL Book Company GmbH

Amburgo · New York · St Louis · San Francisco · Auckland · Bogotá · Città del Guatemala · Città del Messico · Johannesburg · Lisbona · Londra · Madrid · Montreal · Nuova Delhi · Panama · Parigi · San Juan · San Paolo · Singapore · Sydney · Tokyo · Toronto

Titolo originale: *The Spectrum Graphics Machine*

Realizzazione editoriale: EDIGEO srl, via del Lauro 3, 20121 Milano
Traduzione: Giuseppe Zappalà
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelox, Trento

ISBN 88-7700-020-1

1ª edizione Marzo 1985

Indice

A Fran

Prefazione

Questo libro è dedicato ai programmatori dello Spectrum 48K in codice macchina, così come a quelli BASIC, che vogliono inserire un tocco di professionalità nei loro giochi d'avventura.
Esso contiene tutte le specifiche di una "ROM alternativa", che è stata chiamata GOLDMINE (miniera d'oro), ideata apposta per sfruttare al meglio le capacità e la velocità dello Spectrum in tutti i campi che concorrono alla creazione dei giochi.
Ai programmatori in BASIC viene mostrato come accedere alle più importanti routine in codice macchina del sistema GOLDMINE, per aver accesso al controllo dell'intera area dello schermo, inclusa una routine di movimento di sprite di ogni dimensione.
I principianti dell'Assembler, d'altra parte, possono usare tutte le routine senza la necessità di uno sforzo particolare per capire a fondo il loro funzionamento. Le routine del sistema GOLDMINE sono illustrate brevemente nella Guida Rapida dell'Appendice A, assieme ai parametri fondamentali, come gli stati di registri e variabili in entrata e in uscita.
Coloro che invece hanno già dimestichezza con Assembler e codice macchina troveranno sia la teoria che i passi più importanti delle routine discussi in dettaglio; ciò permetterà loro di apportare le modifiche necessarie per adattarle alle esigenze dei loro programmi di animazione. Le routine del sistema GOLDMINE possono sostituire i comandi e le istruzioni contenuti nella ROM dello Spectrum — come PRINT, PLOT, POINT, DRAW, RND, SCREEN$ — e ne includono altri, come il movimento di sprite a pixel con rilevamento di collisione, roll e scroll a pixel di finestre, effetti sonori e contatori di ogni tipo. Tutte queste routine lavorano molto più velocemente di quelle della ROM e permettono l'utilizzo di tut-

to lo schermo: il comando CIRCLE è più veloce perfino in BASIC! Con il sistema GOLDMINE, il programmatore in codice macchina non ha bisogno di accedere a nessuna delle routine della ROM, poiché è stato incluso anche un generatore di caratteri che rende il sistema totalmente indipendente.

Benché questo libro contenga — nell'Appendice B — i listati del sistema GOLDMINE disassemblato, esso trova il suo naturale complemento nel software su cassetta *Routines in Assembler per la grafica avanzata con lo ZX Spectrum*; questo software include anche le procedure presentate nel precedente volume di Nicholls *Tecniche avanzate in Assembler con lo ZX Spectrum.*

Tutti i programmi presentati sono inoltre stati progettati per essere compatibili con lo *ZX Spectrum Machine Code Assembler.*

Capitolo 1

Scansione della tastiera

Parte essenziale di ogni programma di giochi d'animazione è costituita dalle routine di lettura della tastiera e riconoscimento dei tasti premuti. Il dato così ottenuto potrà essere poi sfruttato per controllare il movimento di un oggetto sullo schermo, per selezionare il livello di difficoltà richiesto, o per introdurre le iniziali del giocatore in una tabella di punteggi record.
La routine contenuta nella ROM dello Spectrum è piuttosto complessa: essa riconosce infatti i tasti non solo in modo normale, ma anche in modo L e CAPS SHIFT, in modo E shiftati e normali e infine in SYMBOL SHIFT. Le routine relative, insieme alle opportune tabelle, possono essere trovate nella ROM dello Spectrum agli indirizzi $0205–$03B4. Ai nostri fini interessa solamente rilevare la pressione di un tasto, senza alcun riguardo agli SHIFT, semplificando, e quindi accelerando, la routine.
Considereremo innanzi tutto una routine di scansione dell'intera tastiera, in grado di restituire il codice di carattere dell'ultimo tasto premuto.

1.1 KEY: GOLDMINE 63043

Per poter usare facilmente il *Machine Code Assembler* della *McGraw-Hill*, conserveremo i codici Spectrum/ASCII per le lettere maiuscole e i numeri. La routine inizia all'indirizzo 63043, finisce in 63090 e sfrutta una tabella chiamata KDATA all'indirizzo 62451. La routine principale KEY inizia chiamando la subroutine SCAN (vedi oltre), poi somma il numero del tasto all'indirizzo base della tabella KDATA, puntando quindi all'indiriz-

zo della cella che contiene il codice del carattere corrispondente, che viene posto nella variabile di sistema KEYV all'indirizzo 62374.

SCAN: GOLDMINE 63057

Questa subroutine legge consecutivamente ogni semiriga, usando l'istruzione *in a,(c)*; quindi, dopo aver provveduto preventivamente a mascherare i bit 0-4 dell'accumulatore, ne completa il valore, e controlla se sia zero. In caso di valore compreso fra 1 e 31, esso viene trasferito al registro H e, usando l'istruzione *srl h*, la routine è in grado di riconoscere quale tasto di quella riga sia stato premuto. Il registro E contiene il numero del tasto corrente all'inizio di ogni semiriga, mentre il registro L contiene il numero dell'ultimo tasto che è stato premuto. Come potete vedere, in caso di pressione di più tasti la routine restituirà il numero dell'ultimo tasto trovato premuto. Il registro D viene usato per indicare il numero di semirighe contenenti i tasti premuti e contiene normalmente il valore zero. Questo valore viene posto nella variabile di sistema KNO, all'indirizzo 62381 (o IY+7).
Nota: IY deve essere posto a 62374 per poter usare questa e varie altre routine GOLDMINE, poiché IY è sfruttato come puntatore alle variabili di sistema.
L'ordine di scansione della semiriga è:

1. da CHAPS SHIFT a V
2. da A a G
3. da Q a T
4. da 1 a 5
5. da 0 a 6
6. da P a Y
7. da ENTER a H
8. da SPACE a B

A ogni tasto corrisponde un unico numero da 39 a 0, in funzione dell'ordine di scansione della tastiera: CAPS SHIFT=39, Z=38, X=37, C=36, V=35, A=34, S=33... B=0.
Dopo aver eseguito la scansione della tastiera tramite SCAN, possiamo convertire il numero del tasto premuto al suo codice di carattere, usando la tabella KDATA. A questo provvede la routine principale KEY, descritta prima.

in BASIC
Se desiderate usare questa routine nel vostro programma BASIC per leggere la tastiera, potrete chiamarla usando, per esempio:

```
10 LET a$ = CHR$ USR 62264
```

Questa riga fa sì che a$ contenga il CHR$ del tasto premuto (ricordate che stiamo usando solo lettere maiuscole). Se nessun tasto è premuto, o se sono premuti i tre tasti di controllo, a$ conterrà il carattere corrispondente a CHR$(0). Potete, naturalmente, usare il codice del tasto premuto come segue:

```
10 IF USR 62264 THEN ...
```

Quest'istruzione non farà eseguire il comando successivo al THEN nel caso non sia premuto alcun tasto.

in Assembler
Questa routine dovrebbe essere chiamata direttamente usando call 63043. Essa usa tutti i registri e ritorna avendo caricato il codice nel registro A dell'ultimo tasto premuto e con il registro D contenente il numero delle semirighe relative ai tasti premuti. Questi valori, naturalmente, vengono anche trasferiti alle variabili GOLDMINE di sistema KEYV (62374 o IY+0) e KNO (62381 o IY+7). I programmatori più esperti noteranno che il valore contenuto nella coppia dei registri BC viene modificato dopo la scansione di ogni semiriga, in modo da contenere l'indirizzo per l'istruzione *in a,(c)* e per indicare quando sia stata completata la scansione di tutte le otto semirighe. Ciò si ottiene con:

```
63083 CB 00 rlc b
63085 38 E9 jr c,L3
```

Il registro B contiene inizialmente il valore $FE: solo il bit 0 è di reset e, dopo ogni istruzione *rlc*, trasferisce un bit posto a 1 al carry per sette passaggi successivi attraverso il loop L3. Il valore contenuto in BC cambia ad ogni passo nel modo seguente:

1. 65278
2. 65022
3. 64510
4. 63486
5. 61438
6. 57342
7. 49150
8. 32766

Quando il bit di reset raggiunge il bit 7,*B* e viene trasferito al carry, si esce dal ciclo L3 e si ritorna al programma chiamante.

1.2 LETT: GOLDMINE 65210

È una routine di scansione che acquisisce esclusivamente lettere, costruita a partire dalla routine KEY. Ciò può essere utile, per esempio, per introdurre le iniziali in una tabella di punteggi massimi, poiché essa ritorna solo in seguito alla pressione di un tasto corrispondente a una lettera. La routine inizia all'indirizzo 65214.

in BASIC
Questa routine viene chiamata usando:

```
10 LET a$ = CHR$ USR 62254
```

e ritorna solo in seguito alla pressione di un tasto corrispondente a una lettera; a$ conterrà di nuovo la lettera maiuscola.

in Assembler
Usate call 65210. Vengono sfruttati tutti i registri; al ritorno, il registro A contiene il codice del carattere corrispondente al tasto premuto.

1.3 NUMB: GOLDMINE 65195

Si tratta di una routine simile a quella appena presentata, che ritorna solo in seguito alla pressione di un tasto numerico; essa è utile per la selezione delle opzioni di gioco da un menu contenente fino a nove scelte. La routine inizia all'indirizzo 65198.

in BASIC
Analogamente a quanto già fatto prima, usate:

```
10 LET a$ = CHR$ USR 62244.
```

in Assembler
Usate call 65195. Vengono sfruttati tutti i registri e si ritorna con il carattere del tasto numerico premuto contenuto nel registro A.

1.4 LSCAN: GOLDMINE 63233

Una routine utile per controllare un carattere attraverso lo schermo è LSCAN. Questa routine inizia all'indirizzo 63233 ed è una versione semplificata di KEY, che scandisce consecutivamente ogni semiriga, controllando l'eventuale pressione di un tasto. Eseguita la scansione di tutte le

semirighe, si finisce con il registro E contenente un valore da 0 a 255. Ogni bit del registro E viene usato per indicare lo stato di una semiriga:

il bit 0 indica lo stato della semiriga CAPS SHIFT – V;
il bit 1 indica lo stato della semiriga A – G e così via, fino al bit 7, che indica lo stato della semiriga SPACE – B.

Se una semiriga contiene un tasto premuto, il bit relativo sarà posto a 1, mentre resterà a 0 in ogni altro caso. Questo valore è immagazzinato nella variabile di sistema FLAG1 (62382). Questo valore viene ulteriormente modificato per produrre un numero nel campo 0-31:

il bit 0 indica lo stato della semiriga SPACE – B;
il bit 1 indica lo stato della semiriga CAPS SHIFT – V;
il bit 2 indica lo stato della riga A – ENTER;
il bit 3 indica lo stato della riga Q – P;
il bit 4 indica lo stato della riga 1 – 0.
Questo valore modificato viene immagazzinato nella variabile di sistema FLAG2 (62383).

Il valore finale può essere usato per analizzare lo stato di ogni bit di FLAG1, usando *rra* e controllando lo stato del flag di carry, o l'intero valore.

Il bit 0 indica lo spostamento a sinistra;
il bit 1 indica lo spostamento a destra;
il bit 2 indica lo spostamento verso il basso;
il bit 3 indica lo spostamento verso l'alto;
il bit 4 avanza e potrebbe essere usato come controllo di fuoco.

Usando in questo modo le semirighe e le righe, è possibile lasciare al giocatore un'ampia scelta di tasti, sia per destri che per mancini.

in BASIC
Come esempio dell'uso di LSCAN, esamineremo ora un metodo di spostamento di un punto attraverso lo schermo, visualizzandolo via via che si muove nelle quattro direzioni principali e in diagonale. In questo esempio, useremo la routine PLOT dello Spectrum e ci limiteremo a una finestra di 256×176 pixel. Il seguente programma visualizza un punto nel centro dello schermo, che può essere spostato in una delle otto direzioni ammesse usando i controlli già descritti.

```
10 LET x=127 : LET y=88
20 CLS : GOTO 100
30 LET x=x+(x>0)     : RETURN
```

```
40 LET x=x-(x<255)  : RETURN
50 LET y=y+(y<175)  : RETURN
60 LET y=y-(y>0)    : RETURN
100 PLOT x, y,
110 LET b=USR 62274
120 IF b>14 OR b=0 OR b=3 OR b=12 THEN
    GOTO 110
130 IF b=1 OR b=13 THEN GOSUB 30
140 IF b=2 OR b=14 THEN GOSUB 40
150 IF b=8 OR b=11 THEN GOSUB 50
160 IF b=4 OR b=7 THEN GOSUB 60
170 IF b=6 THEN GOSUB 40 : GOSUB 60
180 IF b=9 THEN GOSUB 30 : GOSUB 50
190 IF b=10 THEN GOSUB 40 : GOSUB 50
200 IF b=5 THEN GOSUB 30 : GOSUB 60
210 GOTO 100
```

in Assembler

Usate call 63233. La routine sfrutta tutti i registri e ritorna con un valore variabile da 0 a 31 nel registro A. È molto più facile interpretare questo valore in codice macchina, poiché si è in grado di testare lo stato di ogni bit separatamente, usando il bit *N,A*. Lo stato di ogni semiriga è contenuto nel registro E e può prestarsi ad eventuali successive elaborazioni. La routine è piuttosto semplice e non dovrebbe essere difficile comprenderne il funzionamento.

Capitolo 2

Visualizzazione sullo schermo

La ROM dello Spectrum contiene una routine per visualizzare un carattere il cui codice varia da 6 a 255 (da $09F4 a $11B6). Se consultate l'Appendice A del vostro manuale Spectrum, noterete che questo campo non contiene solo i caratteri (codici 32-127), ma anche simboli grafici (codici 128-143), caratteri grafici definibili dall'utente (UDG, codici 144-164), parole chiave (codici 165-255) e codici di controllo (6-23). La routine deve controllare il campo cui appartiene il codice ed agire di conseguenza, cioè espandere le parole chiave usando l'opportuna tabella, oppure creare la sagoma a otto byte di un carattere grafico (che non è contenuto nel generatore di caratteri), modificare la posizione di visualizzazione e/o gli attributi temporanei, come predisposti dai codici di controllo (virgola, AT, TAB, FLASH, BRIGHT...). Deve anche considerare la posizione di destinazione della visualizzazione, cioè lo schermo, le linee di edit, o la stampante.
Come vedete, questa routine è eccessivamente complessa per le necessità di un normale programma di giochi. Dopo tutto, noi possiamo modificare gli attributi e la posizione di PRINT AT semplicemente alterando le variabili di sistema. Le parole chiave e i caratteri grafici non sono normalmente usati e possiamo limitarci a sfruttare le lettere maiuscole, i numeri e alcuni segni di interpunzione. Tenendo presente ciò, possiamo riscrivere la routine, in modo che tratti solo i codici da 32 a 100.
Innanzi tutto è necessario produrre un generatore di caratteri GOLDMINE, memorizzato come tabella avente come indirizzo base 62491. Ho lasciato disponibili nel generatore 20 caratteri non definiti, sfruttabili per gli UDG, anche se possono essere usati i normali UDG dello Spectrum; ho scelto di formare i caratteri secondo una matrice di 7×7 pixel, di facile

lettura. Potete variare queste caratteristiche in modo che rispondano alle vostre necessità. I codici di controllo, come ho già fatto notare, sono stati omessi, e sono disponibili alterando opportunamente le variabili di sistema. In GOLDMINE queste variabili sono:

COL 62375
LINE 62376
FLAG 62377
ATTR 62378

Le variabili COL, LINE e ATTR lavorano in modo analogo alle variabili di sistema dello Spectrum, con l'unica differenza che LINE ora ha campo di variazione da 0 a 23 e ATTR è "permanente".
FLAG viene usata per immagazzinare lo stato di OVER e INVERSE. OVER 1 viene indicato dal bit 0 posto a 1, mentre OVER 0 viene indicato dal bit 0 posto a 0. Il bit 1 viene usato in modo simile per indicare lo stato di INVERSE. Ho lasciato disponibile il codice di controllo AT (codice 22) per la visualizzazione di stringhe di caratteri. Questo codice si rivela estremamente utile nella visualizzazione di tabelle o istruzioni, delle cui stringhe può essere parte integrante, rendendo quindi inutile ripristinare le variabili di sistema COL/LINE ad ogni cambio di linea.
Ora che disponiamo di un nostro generatore di caratteri e delle nuove variabili di sistema, possiamo esaminare le routine di stampa.

2.1 PRINT: GOLDMINE 63091

Alla chiamata della routine, il registro A contiene il codice del carattere e, durante la visualizzazione di stringhe, la coppia di registri HL viene usata come puntatore ai caratteri della stringa. La prima verifica è per il codice di controllo AT. Se questo viene trovato, vengono prelevati i due byte successivi della stringa come parametri COL/LINE e aggiornate le variabili di sistema; nel registro A viene poi posto il successivo carattere della stringa.
La routine successiva trova l'indirizzo di schermo relativo ad una data combinazione colonna/linea. Essa inizia con LINE contenuto nel registro H e COL contenuto nel registro L; finisce poi con l'indirizzo di schermo del superiore degli otto byte che formano il carattere contenuto nella coppia di registri HL. Questo indirizzo viene anche immagazzinato, per usi successivi, nella variabile di sistema SCREEN (62379/62380). Viene ora eseguita una verifica sugli UDG. Se il codice di carattere è maggiore di 128, il che implica il bit 7 dell'accumulatore posto a 1, si suppone che sia un UDG (codici 144-164) e l'indirizzo base del generatore di UDG viene posto nella coppia di registri DE. Per i nostri scopi, possiamo usare la

stessa area di memoria già riservata dallo Spectrum per gli UDG, il cui indirizzo base è contenuto nella variabile GOLDMINE di sistema UDG (62443/62444). Se il bit 7 dell'accumulatore è posto a 0, si presuppone che il codice corrisponda ad un normale carattere nel campo 32-100 e in DE viene allora posto l'indirizzo base del generatore di caratteri CHR meno 256; questo valore è contenuto nella variabile GOLDMINE di sistema CHAROM (62439/62440). Trasferiamo ora il codice del carattere alla coppia dei registri HL, moltiplichiamo per otto e sommiamo il valore così ottenuto al contenuto della coppia dei registri DE (*nota:* il codice degli UDG è modificato sottraendovi 144): HL conterrà ora l'indirizzo del primo degli otto byte che formano il carattere nel generatore. La routine successiva considera lo stato di OVER e INVERSE:

se OVER è abilitato, il registro B contiene il valore 255;
se OVER è disabilitato il registro B contiene il valore 0;
se INVERSE è abilitato il registro C contiene il valore 255;
se INVERSE è disabilitato il registro C contiene il valore 0.

Gli otto byte vengono poi copiati dal generatore alla memoria video e vengono modificati come richiesto dai registri B e C.
Copiati gli otto byte, si passa alla routine di predisposizione degli attributi. La coppia di registri HL contiene l'indirizzo SCREEN che è modificato dalla routine **FIND2**, una delle più utili routine di schermo. Essa trova l'indirizzo degli attributi corrispondenti a un determinato indirizzo di schermo; la coppia di registri HL contiene in ingresso l'indirizzo di schermo e in uscita l'indirizzo degli attributi. Il valore di ATTR viene poi copiato in questo indirizzo. L'ultima parte della routine PRINT aggiorna la variabile di sistema COL/LINE. Notate che se la visualizzazione raggiunge COL 31 LINE 23 la successiva posizione di visualizzazione sarà 0,0.

in BASIC
Pensando che questa routine non possa esservi di particolare utilità, non ho predisposto alcun accesso. Comunque, il generatore di caratteri è disponibile e due semplici POKE vi consentiranno di usarlo:

```
POKE 23606,27
POKE 23607,243
```

Questi abiliteranno i caratteri GOLDMINE, mentre

```
POKE 23606,0
POKE 23607,60
```

vi permetteranno di ritornare al set di caratteri Spectrum. Ricordatevi di selezionare CAPS LOCKS prima di usare i caratteri GOLDMINE.

in Assembler
Usate call 63091. In ingresso il registro A contiene il codice del carattere e le variabili di sistema COL, LINE, ATTR e FLAG dovranno essere opportunamente predisposte prima della chiamata. COL e LINE vengono automaticamente aggiornate dopo la visualizzazione del carattere. La routine usa tutti i registri, ma salva HL per la visualizzazione della stringa.

2.2 PSTRING: GOLDMINE 63224

Questa è un'utile routine che visualizza una *stringa di caratteri* contenuta sotto forma di dati. La coppia di registri HL viene usata come puntatore all'indirizzo che contiene il codice del carattere, dovrà quindi essere predisposta all'indirizzo base dei dati prima della chiamata della routine. La fine dei dati deve essere contrassegnata da un'istruzione *nop* (o *defb 0*). Come per la routine PRINT, è consentito il codice di controllo AT e i due byte successivi verranno considerati come parametri di colonna/linea.

2.3 PRSPC: GOLDMINE 63310

È un'altra routine che visualizza BC *spazi* dalla corrente posizione di visualizzazione, usando gli attributi correnti, utile per ripulire parti dello schermo.

2.4 CLS: GOLDMINE 63281

Questa routine ripulisce lo schermo usando il valore corrente degli attributi e resettando la posizione di visualizzazione a 0,0. Notate che lo schermo è 24 linee×32 colonne.

in BASIC
Quest'ultima routine è disponibile e ripulisce tutte le 24 linee dello schermo usando il valore di attributo contenuto nella variabile GOLDMINE di sistema ATTR 62378. Per esempio, per pulire lo schermo usando INK 7 PAPER 0:

```
10 POKE 62378,7
20 LET a=USR 62100
30 PAUSE 0
```

L'istruzione PAUSE 0 è necessaria per mostrare l'intero schermo ripulito. Notate, comunque, che questa è solo una predisposizione temporanea degli attributi, in quanto l'interprete BASIC userà le sue proprie variabili di sistema per il valore permanente degli attributi.

in Assembler

Per dimostrare quanto spiegato finora, il seguente programma mostra come trasformare lo Spectrum in una semplice macchina da scrivere senza sfruttare alcuna routine della ROM. Questo programma visualizza qualunque tasto valido premuto ignorando i tre tasti di controllo e usando come cursore il ridefinito codice di carattere 33.

```
        equ 63043 KEY
        equ 63091 PRINT
        equ 63281 CLS
        equ 62375 COL
        equ 62376 LINE
        equ 62377 FLAG
        equ 62378 ATTR
        equ 63704 BREAK
        di
        ld iy,62374          REM setta il puntatore alla variabi-
                                 le di sistema
        ld a,56
        ld (ATTR),a          REM setta INK 0 PAPER 7
        call CLS
LOOP2   ld hl,(COL)          REM salva la posizione del cursore
        push hl
        ld a,33
        call PRINT
LOOP1   call KEY
        and a                REM attende che non sia premuto
                                 alcun tasto
        jr nz,LOOP1
        pop hl
        ld (COL),hl          REM riposiziona il cursore
LOOP3   call BREAK
        jr nc,END
        call KEY
        and a                REM attende che sia premuto un
                                 tasto
```

```
        jr z,LOOP3
        call PRINT          REM lo stampa
        jr LOOP2            REM ritorna al loop principale
END     ei                  REM resetta IY e abilita l'interrupt
                                per ritornare al BASIC
        ld iy,23610
        ret
```

2.5 BREAK: GOLDMINE 63704

L'unica routine del programma precedente non ancora presentata è BREAK 63704. Si tratta di una breve routine che esamina entrambe le semirighe da CAPS SHIFT a V e da SPACE a B, verificando l'eventuale pressione contemporanea dei tasti CAPS SHIFT e SPACE. In tal caso, il flag di carry viene azzerato, mentre viene posto a 1 in tutti gli altri casi. Ciò è utile durante l'esecuzione di subroutine in cui può essere effettuata in qualunque istante un'interruzione per ritornare tanto al BASIC quanto alla routine principale in codice macchina.
Il seguente programma dimostra come visualizzare una stringa di caratteri AT 11,7; PAPER 6; INK 1; BRIGHT 1; FLASH 0; OVER 1; INVERSE 1:

```
        equ 63224 PSTRING
        equ 62377 FLAG
        equ 62378 ATTR
        di
        ld iy,62374
        ld a,113            REM setta gli attributi PAPER 6 INK 1
                                BRIGHT 1
        ld (ATTR),a         REM pone FLASH 0
        ld a,3              REM pone OVER 1 e INVERSE 1
        ld (FLAG),a
        ld hl,DATA
        call PSTRING
        ei
        ld iy,23610
        ret
DATA    defb 22711          REM AT 11,7
        defs MCGRAW-HILL
        defb 0
```

Infine, il set di caratteri GOLDMINE può essere visualizzato usando il seguente programma:

```
        equ 63091 PRINT
        equ 63281 CLS
        di
        ld iy,62374
        call CLS
        ld a,32              REM parte del codice 32 (SPACE)
LOOP    push af
        call PRINT
        pop af
        inc a
        cp 101               REM ripete fino al codice 100
        jr nz,LOOP
        ld iy,23610
        ei
        ret
```

2.6 SWAP: GOLDMINE 64593

Si tratta di una routine estremamente semplice e breve che può essere usata per fornire interessanti effetti in molti giochi d'animazione. Il suo scopo è il reperimento di uno specifico attributo e la sua sostituzione con un nuovo valore, ottenendo così cambi istantanei dei colori visualizzati. La coppia di registri DE contiene l'indirizzo della variabile di sistema OLD 62425; la coppia di registri HL viene usata come puntatore agli indirizzi degli attributi. Il valore contenuto nell'indirizzo puntato da HL viene confrontato al valore in OLD e, in caso di corrispondenza, questo valore viene sostituito con quello contenuto nella variabile di sistema NEW 62426. In caso di mancata corrispondenza, si passa al successivo indirizzo di attributo, ripetendo fino al controllo di tutti gli indirizzi.

in BASIC
I vecchi e nuovi valori vanno posti nelle appropriate variabili GOLDMINE di sistema. La routine può essere poi chiamata tramite USR 62160. Per esempio, per cambiare PAPER 2 INK 0 in PAPER 1 INK 7, usate:

```
10 POKE 62425,16: POKE 62426,15
20 LET a=USR 62160
```

Ricordate che questa routine controlla l'intero schermo, comprese le linee di edit.

in Assembler
Usate call 64593. I valori di OLD e NEW possono essere predisposti tramite:

```
equ 62425 OLD
equ 64593 SWAP

ld hl,$0F10          REM cambia PAPER 2 INK 0 con PAPER 1
ld (OLD),hl              INK 7
call SWAP
ecc...
```

2.7 PRB: GOLDMINE 64903

Questa è la prima di tre routine di tipo "PRINT AT" e permette la visualizzazione di un carattere iniziando da una specifica riga di pixel (da 0 a 191), nella colonna di carattere 0-31; essa supporta tutte le funzioni della routine PRINT (vedi i paragrafi precedenti), OVER, INVERSE, ATTRIBUTE e permette anche la visualizzazione dei caratteri GOLDMINE e degli UDG dello Spectrum.
Questa routine viene usata per visualizzare un singolo carattere il cui codice è contenuto nel registro A, iniziando a una particolare riga di pixel e in una particolare colonna di caratteri, i cui valori sono contenuti nella variabile di sistema Cc 62403(colonna)/62404(riga). Queste coordinate vengono poste nella coppia di registri BC e il valore di colonna viene moltiplicato per otto per ottenere una coppia di coordinate x,y nel campo 0,0-248,191 convertibile a un indirizzo di schermo usando la subroutine FIND1 (vedi paragrafo 3.1). Quindi, come nella routine PRINT, controlla se si tratti di caratteri normali o UDG e ne trova l'inizio nell'opportuno generatore. Vengono poi considerati gli stati di OVER e INVERSE e i byte del carattere vengono posti negli opportuni indirizzi di schermo. Poiché gli otto byte possono risiedere su due linee di visualizzazione normale, o zone di schermo diverse, viene usata un'altra subroutine FIND che trovi l'indirizzo della linea di pixel sottostante e, se questo indirizzo è superiore alla linea 191, che passi in cima allo schermo, con un effetto di avvolgimento. Questa subroutine è **FIND3 63476**.
Vengono poi considerati gli attributi, la qual cosa rende necessaria un'ulteriore subroutine FIND. Eseguita la copia di tutti gli otto byte, la coppia di registri HL contiene l'indirizzo di schermo della linea di pixel inferiore all'ultima visualizzata; è quindi necessaria una subroutine che trovi la linea ad essa superiore, per poter tornare all'indirizzo di schermo dell'ultimo byte di carattere copiato. La routine opportuna si chiama FIND4 64673. Quest'ultimo indirizzo di schermo viene convertito, tramite la su-

broutine FIND2, nel suo indirizzo di attributi, in cui viene copiato il valore corrente di ATTR. L'indirizzo di schermo del primo byte viene poi prelevato da SCREEN 62379/62380 e analogamente convertito al suo indirizzo di attributo, dove viene copiato il valore corrente di ATTR. Si aggiorna infine la variabile di sistema Cc, ottenendo un effetto di avvolgimento quando viene raggiunto il fondo dello schermo.

2.8 PBST: GOLDMINE 65021

L'indirizzo contenuto nella variabile di sistema STDATA 62431/62432 viene posto nella coppia di registri HL. L'uso di questa routine è previsto solo dal BASIC.

2.9 PBST1: GOLDMINE 65024

Questa routine visualizza stringhe di caratteri utilizzando la routine PRB. La coppia di registri HL contiene l'indirizzo base della stringa, che deve terminare con un *nop* o un *defb 0.*

in BASIC
Questa routine può essere usata per visualizzare caratteri GOLDMINE e UDG iniziando in qualunque riga di pixel e colonna di caratteri. Lo stato di OVER e INVERSE deve essere posto in FLAG 62377 e il valore di attributo in ATTR 62378. La colonna (0-31) deve essere posta nell'indirizzo 62403 e la riga di pixel (0-191) nell'indirizzo 62404. L'indirizzo iniziale dei dati va posto in STDATA 62431/62432.
Vediamo, ad esempio, come visualizzare "HELLO" in colonna 4, riga 170:

```
10 REM HELLO?
20 POKE 23765,0: REM uno zero deve seguire la parola HELLO
30 POKE 62403,4: POKE 62404,170
40 POKE 62431,208: POKE 62432,92
50 LET a=USR 62214
```

Nota: l'indirizzo del primo carattere seguente la REM nella prima linea di programma è normalmente 23760; questo indirizzo può variare in caso di collegamento con il Microdrive. Considerando comunque la configurazione base, si avranno per STDATA i valori 208 (byte meno significativo) e 92 (byte più significativo): 92*256+208=23760.

in Assembler
Usate call 65024. La coppia di registri HL punta all'indirizzo base della stringa. Per esempio, se si vuole visualizzare "HELLO" in colonna 23, linea 42 usando gli attributi correnti, OVER e INVERSE bisogna fare:

```
        equ 65024 PBST1
        equ 62403 Cc
        di
        ld hl,$2a17
        ld (Cc),hl
        ld hl,DATA
        call PBST1
        ei
        ret
DATA    defs HELLO
        defb 0
```

Capitolo

Tracciamento di linee

3

3.1 PLOT: GOLDMINE 63346

Questa routine GOLDMINE è praticamente la stessa contenuta nella ROM Spectrum da $22DC a $2303 tranne che per i seguenti punti:

1. Le coordinate 0,0 vengono considerate l'angolo superiore sinistro dello schermo.
2. È possibile visualizzare sull'intero schermo, cioè è permesso tanto PLOT 255,191 quanto PLOT 255,255. Quest'ultima posizione non è effettivamente sullo schermo, ma non genera messaggio d'errore, come farebbe la routine della ROM. Questa caratteristica viene sfruttata in seguito nei comandi CIRCLE e DRAW.
3. Gli attributi non possono essere incorporati nel comando PLOT, ma non penso che questa sia una caratteristica particolarmente richiesta.

Le variabili di sistema sono XY 62384 e Y 62385. È anche possibile eseguire PLOT OVER e PLOT INVERSE, il cui stato corrente è contenuto nella variabile di sistema FLAG (vedi Capitolo 2).
La routine PLOT dispone di due entry point:

1. PLOT 63346, che preleva le coordinate x,y dalle variabili di sistema XY e Y.
2. **PLOT1 63350**, in cui il registro B contiene il valore di y e il registro C contiene il valore di x.

I valori di x e y sono contenuti, come è stato detto, nella coppia di regi-

stri BC. Il primo controllo è per y<192: in caso contrario viene saltata la routine PLOT. È pertanto chiaro che possono essere forniti valori di y da 192 a 255 senza generare un messaggio d'errore. Viene poi chiamata la subroutine **FIND1**, che trova l'indirizzo di schermo partendo dalle coordinate di un pixel. Questa routine richiede in ingresso le coordinate nella coppia di registri BC e restituisce l'indirizzo di schermo nella coppia di registri HL. Il valore contenuto in questo indirizzo definisce lo stato di otto pixel: per esempio se il valore era 20, la sua conversione in binario darà 00010100. Da questo possiamo vedere che il quarto e il sesto pixel sono "accesi", mentre gli altri non lo sono. Notate che leggiamo i pixel da sinistra verso destra. La routine successiva trova il bit che definisce lo stato del pixel di coordinate x,y. All'inizio di ogni riga di pixel dello schermo, il valore di x è sempre 0 ed esistono 32 indirizzi di schermo per riga, ognuno dei quali contiene lo stato di otto pixel. Mascherando quindi il valore di x, in modo da lasciare valori da 0 a 7 (in codice macchina usate *ld a,c*; *and 7*), ci resta un numero che definisce il bit di quel byte all'indirizzo di schermo che controlla il pixel sotto esame (iniziando dal bit 7, verso il bit 0). La routine inizia con il numero 01111111 contenuto nel registro A e finisce, dopo la modifica, con il bit corrispondente al pixel richiesto posto a 0. Per esempio, se il valore di x era 115 (01110011 binario), l'istruzione *and 7* restituirà 00000011 = 3, indicando che il quarto bit dalla sinistra del byte posto a INT (115/8)+1 è uguale al quindicesimo indirizzo dall'estremità sinistra dello schermo, mentre il registro A conterrà 11101111.
L'ultimo passo consiste nel prelevare il byte a questo indirizzo di schermo e modificarlo, tenendo presente lo stato di OVER e INVERSE, e quindi sostituirlo. I più esperti programmatori in codice macchina potranno trovare interessante l'esame della routine FIND1.

in BASIC
Questa routine permette di "accendere" punti sull'intero schermo, ricordandosi che 0,0 è l'angolo superiore sinistro. Per visualizzare un pixel, per esempio quello alle coordinate 200,190 usate:

```
10 POKE 62384,200: POKE 62385,190
20 LET a=USR 62106
34 PAUSE 0
```

Questa routine prevede un controllo d'errore, nel senso che, pur essendo accettabili valori fino a 255×255, essi non vengono visualizzati se posti al di fuori dello schermo. Valori maggiori di 255 non possono essere inseriti in x e y con un POKE senza ottenere un messaggio d'errore dall'interprete BASIC.

in Assembler
Usate call 63346 se i valori di x e y devono essere prelevati dalle variabili di sistema XY e Y.
Usate invece call 63350 se i valori di x e y sono contenuti nella coppia di registri BC.
La routine sfrutta i registri BC, HL e A, ma non la coppia di registri DE.

3.2 CIRCLE: GOLDMINE 63787 (con RAD≤100)

Come ogni programmatore BASIC sa bene, il comando CIRCLE contenuto nella ROM dello Spectrum ($2320-$24FA) è praticamente inutile nei programmi di giochi. Esso è estremamente lento e permette di disegnare solo cerchi completi, e quelli di raggio molto piccolo sono tutto fuorché simmetrici. Esistono in effetti solo 88 diverse misure di cerchi che possono essere disegnati dallo Spectrum, perché il valore di RAD prima del disegno è arrotondato ad intero e qualunque valore maggiore di 88 genera un "fuori schermo" con uno spiacevole messaggio di errore.
Per ovviare a tutti questi problemi, considereremo ora un metodo completamente differente per produrre circonferenze aventi 100 misure differenti, con l'aggiunta di un incremento della velocità in funzione della misura del cerchio, la possibilità di porre il centro della circonferenza in qualunque punto su una griglia di 256×256 pixel, l'opzione di disegnare archi di circonferenza, (cioè cerchi che possono anche uscire dallo schermo e rientrarvi senza originare messaggi di errore), nonché l'uso dell'intero schermo di 256×192 pixel. In questa routine noi useremo la solita formula di tracciamento di una circonferenza: x ↑ 2+y ↑ 2=r ↑ 2, dove r è il raggio e x,y la posizione di PLOT lungo la circonferenza. Al momento della definizione del cerchio noi conosciamo il centro x,y e RAD, quindi è estremamente facile adattare la formula per ottenere le coordinate dei pixel necessari al tracciamento della circonferenza. Il metodo usato dalla routine è il seguente:

1. Controlla se RAD=0; se è vero ritorna senza alcun effetto.
2. Controlla se RAD=1; se è vero ritorna tracciando solo x,y.
3. Controlla se RAD< =100; ritorna se è maggiore.
4. Inizializza Y2 per contenere l'indirizzo BASE della tabella dei quadrati, cioè lo pone a 0 ↑ 2.
5. Inizializza X2 all'indirizzo nella tabella dei quadrati che contiene (RAD−1) ↑ 2.
6. Inizializza R2 all'indirizzo nella tabella dei quadrati che contiene RAD ↑ 2.
7. Copia i valori contenuti negli indirizzi X2 e Y2 agli indirizzi di xm e ym.

8. Visualizza otto pixel: 1. x+xm, y+ym
   2. x+xm, y−ym
   3. x+ym, y+xm
   4. x+xm, y−ym
   5. x−ym, y+ym
   6. x−xm, y−ym
   7. x−xm, y+ym
   8. x−ym, y−ym

   Se un valore non appartiene allo schermo, ovvero è maggiore di 255, PLOT non va chiamato per evitare l'effetto di avvolgimento intorno allo schermo.
9. Controlla che ym sia minore di xm al ritorno dalla routine.
10. Incrementa il puntatore Y2 al successivo quadrato della tabella, cioè l'indirizzo Y2 conterrà ora (y+1) ↑ 2.
11. Controlla che x ↑ 2+y ↑ 2<r ↑ 2 e riprende dal punto 7 se la condizione è soddisfatta.
12. Mantiene il vecchio puntatore Y2 e diminuisce il puntatore X2.
13. Ritorna al punto 7.

Da quanto appena esposto, potete vedere che a ogni passo del loop vengono visualizzati otto pixel e che, per ottenere un ulteriore incremento di velocità, si sfrutta una tabella per i quadrati da 0 a 100 (da qui il limite di 100 circonferenze), risparmiando quindi tempo di elaborazione. L'intera routine, comprensiva della tabella di 100 quadrati, occupa solo 423 byte: è la più lunga routine presente in GOLDMINE.

in BASIC
Il comando CIRCLE può essere utilizzato come segue. Volendo eseguire per esempio CIRCLE 50,190,90:

```
10 REM predispone x,y,r
20 POKE 62384,50: POKE 62385,190: POKE 62394,90
30 LET a=USR 62112
40 PAUSE 0
```

in Assembler
Usate call 63787 se il valore RAD è contenuto nella variabile di sistema RAD 62394; usate invece call 63790 se il valore di RAD è contenuto nel registro A.
La routine è di facile comprensione e consiste di otto subroutine per sommare o sottrarre combinazioni di xm e ym, tracciando ognuna come richiesto e aggiornando i puntatori X2 e Y2 alla tabella dei quadrati. Vengono utilizzati tutti i registri. Come esempio d'uso delle routine CIRCLE e PLOT, il seguente programma dimostra la velocità dei comandi

GOLDMINE, tracciando tutti e 100 i cerchi o parti di essi per dare l'impressione di un cilindro. Notate che sono permessi anche CIRCLE OVER e INVERSE.

```
        equ 62100 CLS
        equ 63787 CIRCLE
        equ 64008 SQU
        equ 63704 BREAK
        equ 62394 RAD
        equ 62384 XY
        equ 62385 Y
        di
        ld iy,62374
        ld a,5
        out ($fe),a          REM setta BORDER 5
        ld (iy+4),7          REM setta PAPER 0 INK 7
LOOP1   call CLS
        ld a,1               REM parte con RAD=1
        ld (RAD),a
        ld hl,SQU
        ld de,0              REM parte con DE=0 e HL che punta
                                 a SQU di 1
        inc hl
        inc hl
LOOP2   push hl
        push de
        ld c,(hl)            REM setta BC=(HL)
        inc hl
        ld b,(hl)
        rr b                 REM BC=INT BC/16
        rr c
        rr b
        rr c
        rr b
        rr c
        rr b
        rr c
        ld a,c
        add a,40
        ld (XY),a            REM X=C+40 quindi x=(r ↑ 2/16)+40
        ld a,e
        add a,5
        ld (Y),a             REM Y=E+5 quindi y=2*r+5
        call CIRCLE
```

```
pop de
pop hl
inc hl
inc hl              REM impone a HL di puntare al SQU
                        successivo
ld a,(RAD)
inc a               REM RAD=RAD+1; ld(RAD),a
ld (RAD), a
cp 101              REM verifica se RAD=101
jr z,LOOP
add a,a
ld e,a              REM E=RAD*2
call BREAK          REM verifica se è stato premuto un
                        BREAK
jr c,LOOP2          REM ripete tutto per il RAD successivo
ld iy,23610
ei
ret
```

3.3 DRAW: GOLDMINE 64210

(su una griglia di 256×256 pixel)

La routine DRAW della ROM, da $2382 a $24FA, parte integrante della routine CIRCLE, permette di disegnare archi e linee rette. Comunque, per le necessità dei giochi d'avventura, ci interessa solo riprodurre il comando DRAW per tracciare linee rette, perché queste sono usate molto più frequentemente (gli archi non sembrano avere un grande utilizzo, sono più difficili da programmare e il loro tracciamento richiede un tempo maggiore).
Il comando DRAW dello Spectrum per linee rette richiede che le coordinate iniziali corrispondano a quelle dell'ultimo punto visualizzato e che la posizione finale sia definita da due valori di offset x e y, che possono essere positivi o negativi.
Pertanto, per tracciare un quadrato, sarà, ad esempio, necessario:

```
10 PLOT 50,50,
20 DRAW 10,0
30 DRAW 0,10
40 DRAW -10,0
50 DRAW 0,-10
```

Ritengo che questa notazione possa essere fonte di possibili confusioni; preferirei quindi usare le vere coordinate al posto dei valori di offset:

```
10 PLOT 50,50
20 DRAW TO 60,50
30 DRAW TO 60,60
40 DRAW TO 50,60
50 DRAW TO 50,50
```

Con ciò ben presente in mente, riscriviamo la routine della ROM, in modo da definire le coordinate iniziali come quelle dell'ultimo punto visualizzato e le finali come insieme di coordinate di schermo. Come con la routine PLOT, le coordinate 0,0 identificano l'angolo superiore sinistro dello schermo, e noi possiamo specificare l'inizio e la fine del tracciamento in qualunque posto della griglia di 256×256 pixel, permettendo quindi il tracciamento di linee che escano dallo schermo e vi rientrino, per esempio:

```
10 PLOT 150,100
20 DRAW TO 50,250
30 DRAW TO 250,50
```

La nostra routine sfrutta la coppia di registri H'L', il cui valore dovrà essere pertanto salvato all'inizio della routine e ripreso alla fine, per ottenere un sicuro ritorno al BASIC. La coppia di registri HL viene usata per contenere le coordinate dell'ultimo punto visualizzato (x,y) mentre la coppia di registri BC contiene le coordinate di DRAW TO (xm, ym). La coppia di registri DE contiene il segno (SGN) dei valori di offset xm−x e ym−y, il registro D contiene SGN (xm−x) e il registro E contiene SGN (ym−y). Se SGN è negativo, il valore è 255; il valore 1 identifica il segno positivo.
Possiamo così copiare la routine della ROM da $23BB a $24F6, escludendo i controlli di fuori schermo, che sono inutili poiché la specifica stessa delle coordinate di DRAW TO assicura la permanenza su una griglia di 256×256 pixel.

in BASIC
Per disegnare una linea, dovete inserire (tramite dei POKE) le coordinate iniziali nelle variabili GOLDMINE di sistema XY e Y e le coordinate di DRAW TO in XM e YM.
Ad esempio, il comando PLOT 150,150: DRAW TO 160,200 sarà eseguito tramite:

```
10 POKE 62384,150: POKE 62385,150
20 POKE 62399,160: POKE 62400,200
30 LET a=USR 62106: REM PLOT
40 LET a=USR 62118: REM DRAW TO
50 PAUSE 0
```

in Assembler
Usate call 64210. I valori di x, y, xm, ym sono contenuti nelle variabili di sistema:

```
XY 62384
Y  62385
XM 62399
YM 62400
```

Vengono utilizzati tutti i registri (compresa la coppia di registri H'L'). Notate che le coordinate iniziali non fanno parte della linea visualizzata: questo pixel deve essere controllato tramite un PLOT e le coordinate finali vengono poi prese come quelle dell'ultimo punto visualizzato e permettono così la successione di linee, una di seguito all'altra.
È pure possibile usare DRAW OVER e INVERSE, ma non è possibile variare il valore degli attributi.
Esempio: PLOT 150,150; DRAW TO 160,200

```
equ 64210 DRAW
equ 63346 PLOT
equ 62384 XY
equ 62399 XM
di
ld hl,$9696
ld (XY),hl
ld hl,$c8a0
ld (XM),hl
call PLOT
call DRAW
ei
ret
```

Capitolo

Rilevamento di caratteri e punti

4

4.1 Att: GOLDMINE 65183

Analogamente alla routine ATTRIBUTE della ROM, questa routine restituisce un valore che definisce l'attribuito di una "casella" dello schermo, identificata da una coppia linea/colonna.
La routine GOLDMINE permette di scegliere la coppia LINE/COL su qualunque posizione di una griglia di 24×32 caratteri, comprendendo quindi l'intero schermo.
Si tratta di una routine estremamente corta che inizia con la coppia di registri BC contenente la posizione di schermo ATL/C, variabili di sistema 62436/62437. Usando la subroutine FIND6, converte questo valore a un indirizzo di attributi contenuto nella coppia di registri HL. Il contenuto di questo indirizzo viene posto nella variabile di sistema ATT? 62438.

in BASIC
Questa routine può essere usata per trovare il valore degli attributi di qualunque parte dello schermo:
Esempio: ATTRIBUTE linea 23, colonna 31 (0,0 è l'angolo superiore sinistro):

```
10 REM predisposizione LINE,COL
20 POKE 62436,31: POKE 62437,23
30 PRINT USR 62238
40 LET a=USR 62238
```

La linea 30 visualizza il valore di attributo, mentre la linea 40 lo pone nella variabile BASIC a.

in Assembler
Usate call 65183 se la posizione LINE,COL è contenuta nella variabile di sistema ATL/C 62436/62437; usate invece call 65187 se la coppia di registri BC contiene le coordinate (B contiene COL e C contiene LINE). La routine restituisce nel registro A il valore di attributo che è contenuto anche nella variabile di sistema ATT? 62438.
La routine usa solo i registri HL, BC e A, e sfrutta un'altra routine FIND (**FIND6 65167**) che, partendo da una posizione di carattere LINE,COL restituisce un indirizzo di attributo.

4.2 POINT: GOLDMINE 63716

Questa routine è usata analogamente a quella del BASIC Spectrum. POINT x,y restituisce il valore 1 se il punto di coordinate x,y è visualizzato, 0 in caso contrario.
La versione GOLDMINE permette la scelta delle coordinate su una griglia di 256×192 punti; 0,0 identifica l'angolo superiore sinistro.
Le coordinate x,y sono contenute nella coppia di registri BC; sfruttando la subroutine FIND1, la routine converte questo valore a un indirizzo di schermo (vedi PLOT). Il valore contenuto in questo indirizzo viene prelevato e, tramite un loop, il pixel richiesto viene spostato nel bit 0; l'uso dell'istruzione *and 1* ci permette di mascherare i bit da 1 a 7, lasciando quindi nel registro A il valore 1 o 0. Viene così definito lo stato del pixel x,y il cui valore viene poi posto nella variabile di sistema POIV 62386.

in BASIC
Questa routine permette di eseguire il POINT dell'intero schermo, comprese le linee di EDIT, ricordandosi che 0,0 identifica l'angolo superiore sinistro; ad esempio, POINT 150,180 sarà eseguito tramite:

```
10 POKE 62384,150: POKE 62385,180
20 PRINT USR 62297
30 LET a=USR 62297
40 IF USR 62297 THEN...
```

La linea 10 inserisce le coordinate nelle variabili di sistema XY e Y. La linea 20 viene usata per visualizzare il valore, mentre la linea 30 lo pone nella variabile BASIC a. La linea 40 è utile quando deve essere eseguita una certa azione se il punto è "acceso".

in Assembler
Usate call 63716 se le coordinate sono contenute nella variabile di sistema XY 62384/62385, o call 63720 se il registro C contiene x e il registro B contiene y. Il registro A al ritorno conterrà il valore 0 o 1, in funzione dello stato del pixel x,y. Questo valore viene anche posto nella variabile di sistema POIV 62417. La routine sfrutta solo i registri HL, BC e A.

Le routine SCREEN$

La più grave lacuna della routine SCREEN$ della ROM consiste nella sua incapacità di rilevare facilmente gli UDG, ritornando una stringa vuota in caso di rinvenimento di un UDG. È possibile ovviamente alterare la variabile di sistema dello Spectrum CHARS 23606/23607 per far credere alla ROM che il generatore di caratteri abbia una diversa posizione nella RAM, ma questo non è eccessivamente conveniente. Un altro difetto è dato dal doppio immagazzinamento della stringa di output (un *bug* della ROM), che può portare a risultati equivoci.
Tramite GOLDMINE è possibile correggere tutti questi difetti.

4.3 SCR1: GOLDMINE 65089

Questa è la prima delle tre routine SCREEN$ e controlla i caratteri aventi codice da 32 a 127. Inizialmente, la coppia di registri BC contiene le coordinate x,y (COL/LINE), prelevate dalle variabili GOLDMINE di sistema SCR$ 62433/62434. Nella coppia di registri HL viene posto il contenuto delle variabili di sistema CHARS 62441/62442, che normalmente contiene l'indirizzo base meno 256 del generatore di caratteri dello Spectrum. L'istruzione *inc h* ci fornisce l'indirizzo base. La subroutine **FIND5 65074** converte le coordinate x,y all'indirizzo di schermo contenente il primo byte della forma del carattere. La successiva routine **Chr 65035** è una copia della routine ROM da $25AF a $257C, che controlla l'eventuale corrispondenza di tutti gli 8 byte dello schermo con i blocchi di 8 byte del generatore di caratteri, ritornando con il carry posto a 1 se non è stata trovata alcuna corrispondenza, o con il registro B contenente il numero del carattere trovato. Notate come il controllo venga eseguito anche per caratteri visualizzati in INVERSE.
Se il carry è posto a 1 al ritorno dalla subroutine Chr, si salta alla routine SCR2; in caso contrario, il registro A viene usato per contenere il codice del carattere trovato nella posizione x,y che viene posto nella variabile di sistema CHR? 62435.

4.4 SCR2: GOLDMINE 65114

Questa routine esegue un controllo analogo al precedente, ma sui caratteri GOLDMINE. La coppia di registri HL contiene l'indirizzo base del generatore di caratteri GOLDMINE, contenuto nella variabile di sistema CHAROM 62439/62440. In caso di carry, si salta alla routine SCR3.

4.5 SCR3: GOLDMINE 65138

Quest'ultima routine controlla l'eventuale rinvenimento di un UDG. La coppia di registri HL contiene l'indirizzo base degli UDG contenuto nella variabile di sistema UDGS 62443/62444. Se nemmeno questa routine trova un carattere sullo schermo, la variabile di sistema CHR? 62435 conterrà 0.

in BASIC
Come potete vedere, è possibile eseguire un controllo per tutti i caratteri, per i soli caratteri GOLDMINE e UDG, o per i soli UDG.
Per esempio, per verificare la presenza di un qualsiasi carattere in 30,23 potete usare:

```
10 POKE 62433,30: POKE 62434,23
20 LET a$=CHR$ USR 62220
30 LET a=USR 62220
40 IF USR 62220 THEN...
```

La linea 20 pone SCREEN$ 30,23 nella variabile stringa BASIC a$.
La linea 30 restituisce CODE SCREEN$ 30,23 nella variabile BASIC a.
La linea 40 ha effetto solo in caso di ritrovamento di un carattere.
Per cercare esclusivamente caratteri GOLDMINE e UDG, sostituite la riga 30 con:

```
30 LET a=USR 62226.
```

Desiderando ricercare i soli UDG sostituite la riga 30 con:

```
30 LET a=USR 62232.
```

in Assembler
Usate call 65089 per eseguire il controllo su tutti i caratteri, call 65114 per eseguire il controllo sui soli caratteri GOLDMINE e UDG e infine call 65138 per eseguire il controllo sui soli UDG.

La variabile di sistema SCR$ 62443/62444 viene usata per contenere le coordinate x,y (COL,LINE); al ritorno dalla routine, il registro A contiene il codice del carattere trovato, oppure 0 in caso di mancato ritrovamento. Questo codice viene anche posto nella variabile di sistema CHR? 62435. Vengono usati tutti i registri.

Capitolo

Musica ed effetti sonori

5

Tutti i giochi di animazione dovrebbero avere buoni effetti sonori, ma, come ogni programmatore BASIC ben sa, non è possibile ottenerli usando il solo comando BEEP. Ho pertanto incluso 4 routine di effetti sonori, i cui parametri possono essere variati tanto dai programmatori BASIC quanto da quelli in codice macchina.

5.1 BEEP: GOLDMINE 63501

Questa routine è una copia di quella della ROM da $03B6 a $03F5; è copiata perché non è possibile programmare facilmente in altro modo l'accurata temporizzazione richiesta per la musica. All'ingresso di questa routine, le coppie di registri DE e HL contengono i valori di durata e tono della nota richiesta. La coppia di registri DE contiene il prodotto della frequenza (in Hz) per il tempo (in secondi); la coppia di registri HL contiene il risultato di: 437500/frequenza (in Hz) meno 30,125. Come potete vedere, il valore contenuto in HL è costante per una data frequenza, mentre il valore contenuto nella coppia di registri DE varia linearmente con il tempo; ciò rende possibile, fornendo i valori di DE e HL per una scala di tempi da 0,01s a un secondo (Tabella 5.1), produrre la tabella per una ottava musicale.

in BASIC
Questa routine non è disponibile, perché il comando BEEP è molto più facile da usare e da programmare.

Note	Frequenza Hz	Tempo (secondi)							
			1	0,5	0,25	0,1	0,05	0,025	0,01
		HL	DE						
Do	261.63	1643	262	131	65	26	13	7	3
Do#	277.18	1549	277	139	73	28	14	7	3
Re	293.66	1461	294	147	73	29	15	7	3
Re#	311.13	1377	311	156	78	31	16	8	3
Mi	329.63	1298	330	165	82	33	17	8	3
Fa	349.23	1223	349	175	87	35	18	9	4
Fa#	369.99	1152	370	185	92	37	19	9	4
Sol	392.00	1086	392	196	98	39	20	10	4
Sol#	415.30	1023	415	207	104	42	21	10	4
La	440.00	964	440	220	110	44	22	11	4
La#	466.16	909	466	233	117	47	23	12	5
Si	493.88	856	494	247	123	49	25	12	5

Tabella 5.1 Valori di HL e DE per ogni nota, in funzione del tempo

in Assembler
Usate call 63501. La coppia di registri HL può essere caricata dalla tabella (indirizzo base 63562). Ad esempio: *ld hl, (Do#)*, o direttamente usando la Tabella 5.1: *ld hl, 1549.*
Il valore per la coppia di registri DE può essere prelevato da questa tabella. Potete, volendo, sfruttare la formula ricavando così i valori per altre note o per differenti durate.
Desiderando, ad esempio, ottenere un Fa per 0,5 secondi:

```
equ 63501 BEEP
di
ld hl,1223
ld de,175
call BEEP
ei
ret
```

Notate che, desiderando riprodurre della musica, occorre introdurre un loop di ritardo fra le note, per evitarne la fusione. Un tipico loop di ritardo è costituito da:

```
        ld hl,1000
DELAY   dec hl
        ld a,h
        or l
        jr nz,DELAY
```

Come già detto, GOLDMINE contiene 4 routine per effetti sonori, ognuno dei quali programmabile in durata e/o tono.

5.2 BP1: GOLDMINE 64474

Questa routine usa il comando BEEP e preleva i valori delle coppie di registri HL, BC e DE dalle variabili di sistema GOLDMINE:

BC 62418/62419
HL 62420/62421
DE 62422/62423

Il registro B contiene il numero di passaggi attraverso la routine; ad ogni passaggio viene incrementato il valore contenuto nella coppia di registri HL e si ottiene così un progressivo smorzamento del suono.

5.3 BP2: GOLDMINE 64498

Questa routine controlla direttamente l'altoparlante attraverso l'istruzione *out (254),a*. Utilizziamo la variabile di sistema DE (vedi BP1) come controllo. Il registro D contiene il tono e il registro E la durata. Notate che il colore della cornice viene preventivamente posto nei bit 0-2 del registro A, in modo che l'istruzione *out* non cambi il colore corrente della cornice.

5.4 BP3: GOLDMINE 64530

Questa routine produce un effetto di rumore bianco, con la cornice lampeggiante; è ideale per simulare esplosioni, la cui durata è contenuta nella variabile di sistema HL.

5.5 BP4: GOLDMINE 64557

Anche questa routine controlla direttamente l'altoparlante. Il tono è contenuto nella variabile di sistema BC (*nota:* viene usato solo il valore contenuto nel registro C); la durata è contenuta nella variabile di sistema HL.

in BASIC
Sono disponibili tutti e quattro gli effetti sonori. Provate a sperimentare differenti valori per le variabili di sistema BC, HL e DE.
Per accedere alle routine di effetti sonori usate:

```
BP1: LET a=USR 62136
BP2: LET a=USR 62142
BP3: LET a=USR 62148
BP4: LET a=USR 62154
```

Cercate di non eccedere con i valori dei parametri, poiché varie routine possono proseguire per tempi estremamente lunghi (specialmente per valori alti di BC).

in Assembler
Predisponete i valori nelle opportune variabili di sistema, quindi usate:

```
call 64474 per BP1 (vengono usati tutti i registri)
call 64494 per BP2 (vengono usati i registri A, B e DE)
call 64530 per BP3 (vengono usati solo i registri A e HL)
call 64557 per BP4 (vengono usati i registri A, HL, BC ed E).
```

Nota: mantenete ogni effetto sonoro il più breve possibile, per evitare il rallentamento del movimento degli oggetti (anche usando un interrupt di modo 2).

Capitolo

Punteggi, conteggi e numeri casuali

6

Ogni gioco d'animazione richiede qualche routine per tener conto del punteggio, del numero di vite rimanenti, o forse del tempo ancora disponibile. Per questo scopo, GOLDMINE contiene un insieme di routine che permettono il conteggio crescente, decrescente e la ricerca dei valori massimi e minimi. La grandezza del numero trattabile è limitata solo dalla memoria disponibile.

6.1 C/UP: GOLDMINE 63587

È una routine per il *conteggio crescente.* Il massimo numero richiesto viene immagazzinato come stringa di dati in byte consecutivi, preceduti e seguiti da *defb 0* (o *nop*). Se, per esempio, il massimo numero richiesto è 9999, sono necessari 6 byte di dati così inizializzati: 0 48 48 48 48 0 (48 è il codice CHR$ di "0"). GOLDMINE contiene un'area per questo tipo di dati, idonea al massimo numero di 9.999.999, il cui indirizzo base è 62357; BUF3 è la label del byte corrispondente al primo numero. La routine inizia predisponendo la coppia di registri HL come puntatore a BUF3; viene poi eseguito un controllo sul contenuto degli indirizzi seguenti BUF3 per trovare il byte 0 e ritornare all'indirizzo delle unità per l'inizio della routine principale di conteggio. Innanzitutto controlla se è presente un byte numerico nella posizione riservata all'unità, in caso contrario ritorna. Viene poi incrementato il valore delle unità e si controlla se è diventato un codice 58 ("9"+1), indicante il riporto di uno. In tal caso, quell'indirizzo viene forzato a contenere il codice 48 ("0") e si passa all'indirizzo delle decine, lo si incrementa eseguendo simili control-

li sul suo valore, e così via, fino a trovare un indirizzo che non generi riporto, oppure l'indirizzo iniziale che contiene 0. Completato il conteggio crescente, si passa alla subroutine Cend 63616, che visualizza il nuovo valore in colonna, linea, prelevati dalla variabile GOLDMINE di sistema CLUP 62445(colonna), 62446(linea). Questi valori vengono posti nella variabile di sistema COL; la coppia di registri HL viene poi caricata con l'indirizzo base BUF3 prima di cedere il controllo alla routine PSTRING per visualizzare il numero. Notate che il byte 0 (marker di fine stringa) è parte integrante dei dati del nostro numero. Viene anche usata la routine **Numb? 63628** che controlla se il registro A contiene un codice corrispondente a un numero; il flag di carry viene posto a 1 nel caso che il codice non corrisponda a un numero e posto a 0 in caso contrario.

6.2 C/DN: GOLDMINE 63635

Questa routine di *conteggio decrescente* è simile a quella di conteggio crescente, in quanto il numero di partenza viene salvato come dato nello stesso modo. Essa usa BUF4 62366. Trovato l'indirizzo delle unità, ne decrementiamo il contenuto e controlliamo se è diventato il codice 47 ("0" − 1) sostituendolo in tal caso con il codice 57 ("9") e passando poi all'indirizzo delle decine e così via. La routine di visualizzazione del numero, Cdend 63663, preleva i suoi valori di linea/colonna dalla variabile GOLDMINE di sistema CLDN 62447(colonna)/62448(linea).

6.3 Chup9: GOLDMINE 63676

Questa routine scandisce i dati controllando se tutti i codici rappresentino la cifra "9" e ritorna 0 nel registro A in caso di raggiungimento del valore massimo, o il codice della prima cifra diversa da "9" in caso contrario.
Senza questa routine, il massimo valore consentito traboccherebbe resettando il conteggio a 0.

6.4 Chdn0: GOLDMINE 63690

Analogamente alla precedente, questa routine controlla i codici contenuti in BUF4 per riscontrare se essi contengano tutti "0" e ritorna ancora con 0 nel registro A in caso di soddisfacimento della condizione, o con il codice della prima cifra diversa da 0.

in BASIC

Queste routine sono disponibili e facili da includere nei vostri programmi, con la limitazione di un valore massimo pari a 9.999.999. Per usare la routine è necessario innanzitutto predisporre le coordinate di PRINT AT, introducendo la colonna (0-31) in 62445, la linea (0-23) in 62446 e il valore di attributi in 62378.
È quindi necessario predisporre il valore iniziale ponendo, tramite dei POKE, i codici del numero negli indirizzi da 62358 a 62365, ricordandosi che l'indirizzo delle unità deve essere seguito da un indirizzo contenente 0. Per esempio, per predisporre 00 000, da visualizzare in linea 3, colonna 10, usate:

```
10 POKE 62445,10: POKE 62446,3
20 FOR a=62358 TO 62362
30 POKE a,48: NEXT a
40 POKE 62363,0
```

Per il conteggio crescente usate semplicemente LET a=USR 62313 e per controllare il raggiungimento di un valore massimo, sfruttate IF USR 62325 THEN ...
Per esempio:

```
10 IF INKEY$="7" THEN LET a=USR 62313
20 IF USR 62325 THEN GOTO 10
30 PRINT AT 11,7; "CONTEGGIO TERMINATO"
```

Questa routine, in seguito alla pressione del tasto 7, eseguirà un conteggio crescente visualizzando "CONTEGGIO TERMINATO" al raggiungimento di 99.999.
In modo analogo, è possibile programmare e predisporre il conteggio decrescente, ponendo i valori di colonna e linea in 62447/62448 rispettivamente e il numero negli indirizzi da 62366 a 62372. Per chiamare la routine usate:

```
LET a=USR 62319
```

e, per controllare l'eventuale raggiungimento dello 0, usate:

```
IF USR 62331 THEN ...
```

Notate che il numero viene visualizzato utilizzando i caratteri GOLDMINE; per usare il set di caratteri dello Spectrum dovrete inserire nella variabile GOLDMINE di sistema CHAROM i seguenti valori:

POKE 62439,0: POKE 62440,60

e per ripristinare le condizioni iniziali:

POKE 62439,27: POKE 62440,243

in Assembler

CONTEGGIO CRESCENTE

Se desiderate usare BUF3 per contenere la stringa del numero, sfruttate call 63587; in caso contrario, la coppia di registri HL deve contenere l'indirizzo base dei dati del numero, ma ricordate che questi devono essere preceduti e seguiti da *defb 0*, e usate call 63590.
Per controllare l'eventuale raggiungimento di un valore massimo, usate call 63676 se avete utilizzato BUF3, o call 63679 se l'indirizzo base dei dati del numero è contenuto nella coppia di registri HL.

CONTEGGIO DECRESCENTE

Usate call 63635 se state sfruttando BUF4; call 63638 se l'indirizzo base del numero è contenuto nella coppia di registri HL.
Per controllare il raggiungimento dello 0, usate call 63690 se state sfruttando BUF4 o call 63694 se l'indirizzo base del numero è contenuto nella coppia di registri HL.
Come esempio di conteggio crescente da 0000 a 9.999 e, contemporaneamente, di conteggio decrescente partendo da 1111 (senza controllo di 0), usate il seguente programma:

```
equ 62356 BUF3
equ 62366 BUF4
equ 62378 ATTR
equ 63281 CLS
equ 62445 CLUP
equ 62447 CLDN
equ 63587 C/UP
equ 63635 C/DN
equ 63676 Chup9
di
ld a,56                 REM inizializzazione
ld (ATTR),a
call CLS
ld hl,$0314
ld (CLUP),hl
ld hl,$0514
ld (CLDN),hl
```

```
        ld hl,BUF3
        ld b,4
LOOP1   ld (hl),48
        inc hl
        djnz LOOP1
        ld (hl),0
        ld hl,BUF4
        ld b,4
LOOP2   ld (hl),57
        inc hl
        djnz LOOP2
        ld (hl),0
LOOP3   call C/UP              REM inizio della routine
        call C/DN
        call Chup9
        and a
        jr nz, LOOP3
        ei
        ret
```

6.5 RAND: GOLDMINE 63738

I numeri casuali sono molto importanti e vengono usati spesso nei giochi d'avventura. Tuttavia, non possiamo ottenere un equivalente esatto della routine della ROM, perché non siamo in grado di gestire numeri nella loro rappresentazione a 5 byte in virgola mobile. Ciò nonostante, ci è possibile scrivere una routine che produca numeri sufficientemente casuali per i nostri scopi.
Analogamente alla routine della ROM, sfruttiamo una variabile di sistema SEED 62392/62393 che contiene un numero iniziale. La routine seguente modifica questo numero (la routine è il risultato di una serie di tentativi per ottenere un numero finale il più casuale possibile). Il numero così modificato viene posto nuovamente in SEED per ulteriori usi futuri.
Poniamo poi nella coppia di registri DE il valore prelevato da RND 62390/62391, che definisce il campo di variazione del numero richiesto, e controlliamo se questo valore è minore di 256; in tal caso saltiamo alla routine 63778. Il valore SEED viene poi diviso per RND e il resto viene preso come numero casuale, nel campo da 0 a RND−1. Questo numero casuale viene poi posto nella coppia di registri BC prima del ritorno. Notate che se il numero RND è minore di 256, dividiamo RND per il valore contenuto nel solo registro L, velocizzando quindi la routine di divisione.

in BASIC
Poiché questa routine non produce un numero veramente casuale, la sua utilità è molto relativa. Potete tuttavia vederne i risultati eseguendo il seguente programma:

```
10 INPUT "NUMERO CASUALE 0-65535";a
20 LET b=INT (a/256)
30 LET c=a-b*256
40 POKE 62390, c: POKE 62391, b
50 PRINT USR 63738;"  ";
60 GOTO 50
```

Questa routine continuerà a visualizzare numeri casuali nel campo di variazione scelto fino alla pressione del tasto BREAK.

in Assembler
Usate call 63738. Il campo di variazione di RND deve essere posto nella variabile di sistema RND 62390/62391. Per esempio, per ottenere un numero casuale INT (RND*35), usate:

```
equ 63738 RAND
equ 62390 RND
di
ld hl,35
ld (RND),hl
call RAND                    REM il numero casuale è ora in BC
```

Capitolo 7

Roll e scroll di una finestra

Presentiamo ora un altro utile insieme di routine di facile programmazione e grande utilità nei giochi d'animazione. Queste routine vi permettono di definire la posizione di una finestra usando le coordinate Ca (0-31), Ya (0-191) dell'angolo superiore sinistro della finestra, la cui misura sarà definita tramite Cb (1-32), Yb (2-192). Come vedete, siamo limitati alle colonne dei caratteri nella definizione della finestra, ottenendo in cambio un movimento molto più veloce. Se cercate di usare ampiezze di pixel invece che di caratteri, è necessaria una routine di visualizzazione molto più complicata e quindi più lenta. Inoltre, possiamo muovere solo i byte dello schermo senza gli attributi, perché non siamo in grado di eseguire movimenti a pixel su quest'area di memoria.

7.1 WUR: GOLDMINE 64614 (Roll verso l'alto)

Questa e le routine successive usano le variabili GOLDMINE di sistema Ca 62427, Ya 62428, Cb 62429, Yb 62430 per contenere le coordinate iniziali e la misura, come appena descritto. Esse iniziano prelevando innanzitutto i valori di Ca,Yb e convertendoli in un indirizzo di schermo moltiplicando Ca per otto e usando la subroutine FIND1. Il passo successivo consiste nell'immagazzinare la riga in cima alla finestra ponendo sullo stack i contenuti degli indirizzi di schermo Cb. Notate che Ca+Cb deve essere nel campo da 1 a 32, cioè la finestra non deve avvolgersi attorno allo schermo. Si trasferisce poi la seconda riga di pixel della finestra nella prima riga. Ciò viene eseguito ponendo nella coppia di registri HL l'indirizzo iniziale della finestra (contenuto nella variabile GOLDMINE di si-

stema SCREEN) e usando la subroutine FIND3 per trovare l'indirizzo di schermo della linea di pixel sottostante. La coppia di registri DE viene quindi inizializzata con l'indirizzo iniziale della finestra.
A questo punto l'indirizzo iniziale della prima linea è nella coppia di registri DE, l'indirizzo iniziale della linea successiva nella coppia di registri HL, l'altezza della finestra nel registro B e l'ampiezza della finestra nel registro C. Vengono salvati sullo stack i registri HL e BC prima di azzerare il registro B e, usando *ldir*, si trasferiscono i contenuti degli indirizzi in HL agli indirizzi in DE, per un totale di "C" indirizzi.
Vengono poi prelevati dallo stack le coppie di registri BC e HL, decrementato il valore contenuto nel registro B e ripetuta la routine di roll fino al trasferimento di tutte le linee, cioè fino all'azzeramento del valore contenuto nel registro B.
L'ultimo passo consiste nel prelevare dallo stack i byte della prima linea e porli negli indirizzi dell'ultima linea della finestra. Notate che bisogna iniziare dalla fine di questa linea, perché i valori prelevati sono in ordine inverso.

7.2 WDR: GOLDMINE 64700 (Roll verso il basso)

Questa routine è simile a quella di roll verso l'alto, tranne che inizia dall'ultima linea della finestra e usa la subroutine **FIND4 64673** per trovare l'indirizzo della linea di pixel soprastante.

7.3 WRR: GOLDMINE 64765 (Roll a destra)

Analogamente alla routine di roll verso l'alto, si trova l'indirizzo di schermo contenente l'angolo superiore sinistro della finestra tramite FIND1 e lo si immagazzina. Per muovere il byte contenuto in questo indirizzo a destra di un pixel viene usata l'istruzione *rr(hl)*, sfruttando il fatto che il bit passato nel carry sarà prelevato dalla successiva istruzione *rr(hl)*. In questo modo, si opera sulla linea superiore per "C" indirizzi. L'effetto di roll è ottenuto considerando lo stato del flag di carry dopo l'ultima istruzione *rr(hl)*, ripristinando l'indirizzo iniziale della linea nella coppia di registri HL e ponendo a 1 il bit *7,hl* se il carry è posto a 1. Questo loop è ripetuto per "B" linee e usa FIND3 per la scansione della finestra.

7.4 WLR: GOLDMINE 64803 (Roll a sinistra)

Si tratta di una routine simile a quella di roll a destra, tranne che si inizia alla fine della prima linea e viene usata l'istruzione *rl(hl)* per ruotare i byte a sinistra anziché a destra.

7.5 WUS: GOLDMINE 64847 (Scroll verso l'alto)

La presenza del nostro programma in RAM anziché in ROM ci permette di modificare temporaneamente le nostre routine di roll per far loro eseguire lo scroll. In questo caso, viene modificata la routine di roll verso l'alto, con la sostituzione dell'istruzione *ld a,(hl)* in 63634 con *xor a*, ponendo cioè sullo stack degli zeri anziché i byte della prima riga. Questo ha l'effetto di ripulire l'ultima linea della finestra al momento in cui gli zeri prelevati dallo stack vengono posti nella memoria video. In effetti, noi modifichiamo la routine di roll verso l'alto per ottenere lo scroll, la eseguiamo e quindi la resettiamo a roll.

7.6 WDS: GOLDMINE 64861 (Scroll verso il basso)

Questa routine usa lo stesso metodo presentato nella routine di scroll verso l'alto.

7.7 WRS: GOLDMINE 64875 (Scroll a destra)

Viene modificata la routine di roll, sostituendo l'istruzione *jr nc,Rc* con *jr,Rc*, rimpiazzando cioè il codice 48 con 24 ed evitando di porre a 1 il bit *7,(hl)*.

7.8 WLS: GOLDMINE 64889 (Scroll a sinistra)

In questa routine viene sostituito *jr nc,Lc* con *jr,Lc* analogamente a quanto eseguito nella routine di scroll a destra.

in BASIC
Sono disponibili tutte le otto routine sopra presentate. Occorre predisporre le seguenti variabili di sistema GOLDMINE:

Ca 62427 = colonna iniziale della finestra (0-31)
Ya 62428 = linea iniziale della finestra (0-191)
Cb 62429 = ampiezza della finestra (1-32), ricordandosi che deve essere $1 \leq Ca+Cb \leq 32$
Yb 62430 = altezza della finestra (2-192)

Come potete vedere, la finestra può avvolgersi fra la cima e il fondo dello schermo, ma non attorno ai lati. Le routine sono così chiamate:

Roll verso l'alto:	LET a=USR 62166
Roll verso il basso:	LET a=USR 62172
Roll a destra:	LET a=USR 62178
Roll a sinistra:	LET a=USR 62184
Scroll verso l'alto:	LET a=USR 62190
Scroll verso il basso:	LET a=USR 62196
Scroll a destra:	LET a=USR 62202
Scroll a sinistra:	LET a=USR 62208

in Assembler
Predisponete le quattro variabili GOLDMINE di sistema, Ca, Ya, Cb, Yb e usate:

Roll verso l'alto:	call 64614	usa tutti i registri
Roll verso il basso:	call 64700	usa tutti i registri
Roll a destra:	call 64765	usa i registri HL, BC e A
Roll a sinistra:	call 64803	usa i registri HL, BC e A
Scroll verso l'alto:	call 64847	usa tutti i registri
scroll verso il basso:	call 64861	usa tutti i registri
Scroll a destra:	call 64875	usa i registri HL, BC e A
Scroll a sinistra:	call 64889	usa i registri HL, BC e A

Ad esempio, per ottenere un continuo roll a sinistra di una finestra, con l'angolo superiore sinistro in colonna 4, linea 25 e ampiezza 8 colonne per 131 righe di altezza, usate il seguente programma:

```
        equ 62427 Ca
        equ 62429 Cb
        equ 64803 WLR
        equ 63704 BREAK     REM prima predisponete la fine-
                                stra con un programma BA-
                                SIC o in codice macchina
        di
        ld hl,$1904
        ld (Ca),hl
        ld hl,$8308
        ld (Cb),hl
LOOP    call WLR
        call BREAK
        jr c,LOOP
        ei
        ret
```

Ho incluso la subroutine BREAK, per poter tornare al BASIC.
I programmatori più esperti noteranno come vengono modificate le routine di roll per convertirle temporaneamente in routine di scroll.

Capitolo

Movimento di sprite 8

Esamineremo ora un insieme di routine per ottenere il movimento a pixel dei nostri caratteri su tutto lo schermo.
Avrete senza dubbio notato che, poiché lo schermo è mappato in memoria, non possiamo ottenere facilmente o velocemente il movimento a pixel di caratteri a sinistra e a destra, pur essendoci possibile il movimento a pixel in su o in giù. Per risolvere questo problema ed ottenere i movimenti orizzontali, sfrutteremo il movimento a passi di carattere e un piccolo trucco.
Per ottenere l'impressione del movimento a pixel, definiremo la nostra misura di sprite un carattere più ampia del necessario e conserveremo i dati del nostro sprite sotto forma di otto figure separate, ognuna slittata a destra di un bit rispetto alla precedente. Se visualizziamo le nostre otto figure, una di seguito all'altra, usando la stessa posizione iniziale di schermo, lo sprite sembrerà muoversi di un pixel nell'ambito di quella unità di carattere. Aggiornando poi la posizione di schermo a passi di un carattere e rivisualizzando le otto figure avremo l'impressione di un continuo movimento a pixel.
Questo è il principio di base del primo metodo di movimento di sprite a passi di un pixel ed il solo disponibile ai programmatori BASIC. Poiché desideriamo muovere anche gli attributi degli sprite lungo tutto lo schermo, limiteremo la nostra misura a passi interi di un carattere, definendo l'altezza dello sprite pari alla sua larghezza. Anche se non è essenziale, ciò serve a dare una migliore impressione di movimento.
Ciò comunque implica che se lo sprite è ad esempio, 4×4 caratteri, ci saranno necessari $4\times4\times8\times8=1$ kbyte di memoria per immagazzinare le otto figure. Poiché ci sembra un po' eccessivo, eseguiremo i movimenti a

sinistra e a destra di due pixel ed avremo così bisogno di solo 4 figure e di 512 byte di memoria. Per compensare il fatto che stiamo usando movimenti a passi di due pixel, possiamo rendere le nostre 4 figure degli sprite lievemente differenti per ottenere, attraverso il movimento, un effetto di animazione. Ad esempio, lo sprite di un cane potrà avere le zampe in posizioni differenti per dare l'impressione del movimento... Il primo passo consiste allora nel disegnare le nostre quattro figure e vedremo come utilizzare una griglia di 3×2 caratteri (2×2 sprite), per disegnare una ruota avente i raggi in posizioni differenti, così da ottenere l'impressione del rotolamento contemporaneamente al movimento verso destra o verso sinistra.

Metodo 1

Noterete che dobbiamo iniziare con la nostra ruota compresa entro la zona riservata ai primi 2×2 caratteri, senza occupare la prima e l'ultima linea e la prima e la seconda colonna; la quarta figura avrà la ruota compresa nel secondo, terzo, quinto e sesto carattere, lasciando intatte le ultime due colonne. Useremo una routine di visualizzazione che sovrapporrà la nuova figura a quanto precedentemente visualizzato. Immagini consecutive sono shiftate a destra di due bit. Questa regola generale deve essere applicata a tutte le misure di sprite.
Dobbiamo ora convertire le nostre quattro figure in una stringa di byte di dati. Iniziando dalla figura 1, colonna 1, linea 1, esaminiamo ogni singola riga, ottenendo i relativi valori dei byte, fino ad arrivare alla figura 4, colonna 24, linea 16, per un totale di 3×2×8×4=192 byte esaminati. Ritengo che il modo più semplice per calcolare i valori dei byte e, contemporaneamente, avere un'idea dell'aspetto dello sprite in movimento, consista nell'uso di un programma fornito insieme allo Spectrum che permette di ridefinire i caratteri da A a T e di rappresentare quindi i singoli caratteri componenti ogni figura. Le figure sono rappresentate nella Tavola 8.1.
Il procedimento per ottenere questi UDG è mostrato nella Tavola 8.2. Tutto ciò di cui abbiamo bisogno ora è un breve programma BASIC che ci mostri il rotolamento della ruota verso destra, per poterne controllare l'aspetto. Il programma è:

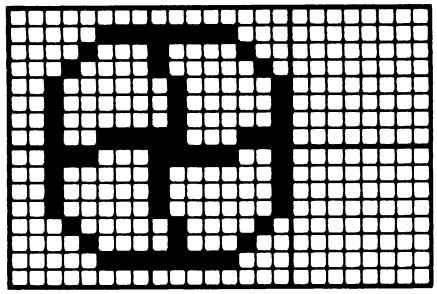
```
10 LET x=0 : LET y=0
20 PRINT AT x,y;"AB ";AT x+1,y;"CD "
25 PAUSE 5
30 PRINT AT x,y;"EFG";AT x+1,y;"HIJ"
35 PAUSE 5
```

```
40 PRINT AT x,y;"KLM';AT x+1,y; "NOP"
45 PAUSE 5
50 PRINT AT x,y;" QR";AT x+1,y;" ST"
60 PAUSE 5
70 LET y=y+1 : IF y=30 THEN CLS : GOTO 10
80 GOTO 20
```

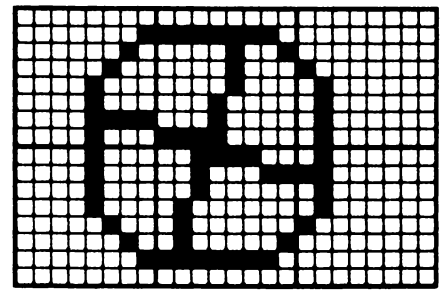
Sarà ora possibile esaminare accuratamente il movimento dello sprite, eventualmente modificandolo prima di passare il controllo al codice macchina.
Abbiamo così creato i nostri sprite. Eseguito ogni necessario controllo estetico, si può passare a predisporre le variabili necessarie e a descrivere la vera e propria routine di visualizzazione degli sprite.

8.1 SPRITE: GOLDMINE 63388

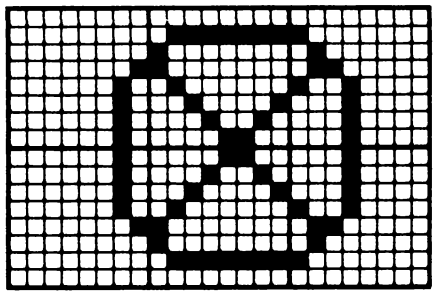
Questa routine inizia prelevando i valori delle variabili GOLDMINE di sistema CO2 62412/62413, FORMAT 62414/62415 e MAN1 62388/62389. CO2 viene usata per contenere le coordinate del pixel cui corrisponde il primo bit dello sprite, cioè l'angolo superiore sinistro. FORMAT contiene la misura dello sprite in caratteri (in 62414 l'ampiezza e in 62415 l'altezza). MAN1 contiene l'indirizzo iniziale dei dati della figura correntemente in uso. Notate che le coordinate x della prima figura devono essere un multiplo di otto, in modo da ottenere una corretta sequenza delle quattro figure.
Il primo passo consiste nel trovare l'indirizzo di schermo per CO2 usando la subroutine FIND1 e poi nel convertire l'altezza dei caratteri in righe di pixel, moltiplicando per 8 il valore contenuto nel registro B. Il secondo passo consiste nel controllare lo stato del bit 0 della variabile di sistema HITVAR 62387 (IY+13). Se questo bit è a 1, viene eseguito solamente un controllo di collisione con un altro oggetto; in caso contrario si prosegue copiando i byte di dati negli indirizzi dello schermo.
Il controllo di collisione agisce sul pixel e non sull'attributo, come fanno molti programmatori; ciò permette di ottenere una particolare accuratezza, in quanto il controllo è eseguito su ogni singolo byte della figura, con l'indirizzo corrente di schermo. Il primo controllo verifica che all'indirizzo corrente di schermo sia effettivamente presente il byte in esame della nostra figura e assicura quindi che allo sprite non si sia sovrapposto un altro oggetto. In caso di mancata corrispondenza, viene controllato che l'oggetto sovrapposto abbia toccato un pixel della figura. In tal caso, il valore dell'attributo corrispondente a quell'indirizzo di schermo viene confrontato con il valore di un'altra variabile di sistema, HATT 62405, che contiene il valore di attributo di un oggetto che entrerà in collisione



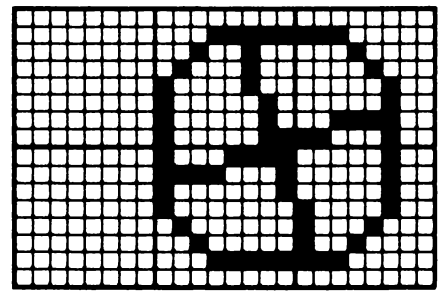
(a)



(b)



(c)



(d)

Tavola 8.1 Le quattro figure necessarie per visualizzare la ruota in movimento:
(a) figura 1: A B SPACE C D SPACE
(b) figura 2: E F G H I J
(c) figura 3: K L M N O P
(d) figura 4: SPACE Q R SPACE S T

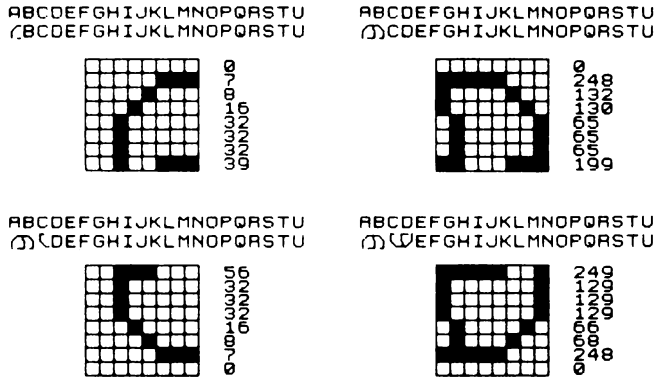


Tavola 8.2

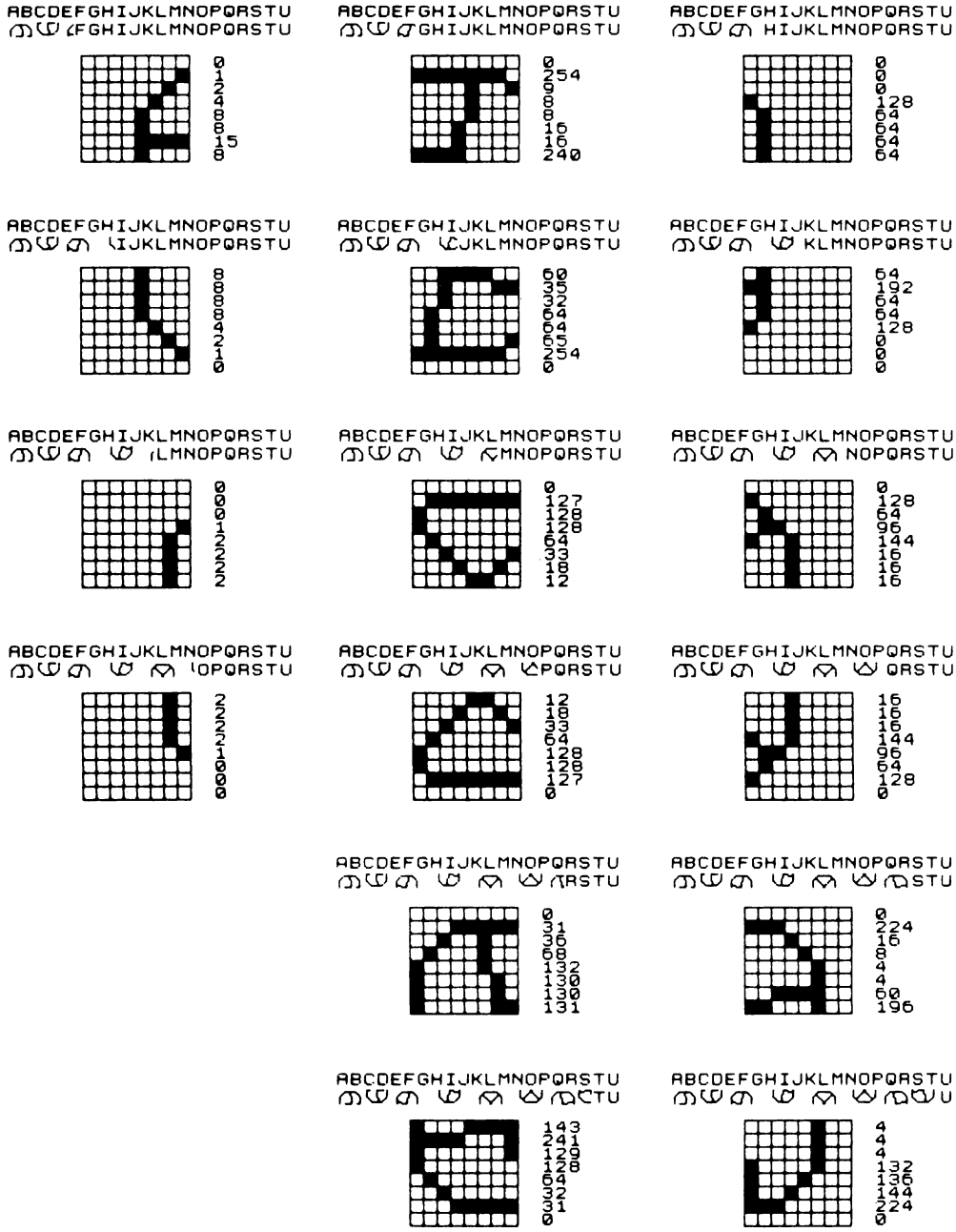


Tavola 8.2 Procedimento di costruzione delle quattro figure per lo sprite della ruota

con lo sprite (causando, per esempio, la perdita di una vita) piuttosto che attraversarlo senza danni. Se viene trovato un simile attributo, il flag di carry viene posto a 1.
Noterete che lo sprite si avvolgerà intorno allo schermo nelle 4 direzioni principali e che è stata usata la routine FIND3 per trovare la linea di pixel sottostante.

8.2 MOVE: GOLDMINE 64313

La successiva routine di questo gruppo legge la tastiera sfruttando la subroutine LSCAN e usa il risultato per aggiornare le coordinate della posizione dello sprite.
Essa è abbastanza semplice: preleva il valore di FLAG2, ottenuto dopo la chiamata a LSCAN e aggiorna le coordinate CO1 62410/62411 ponendole in CO2. La routine dà un effetto di avvolgimento.

8.3 MOVE?: GOLDMINE 64393

Controlliamo ora gli attributi in questa nuova posizione per vedere se per caso è presente una barriera, cioè un oggetto che impedisce il movimento dello sprite in quella direzione.
Le coordinate aggiornate vengono convertite al loro indirizzo di attributi. Viene poi eseguito il controllo per un'altezza di uno superiore a quella del formato. Ciò si ottiene verificando se il valore nel registro H rappresenti o meno il valore iniziale di un carattere, cioè se l'istruzione *and* 7 dia o meno 0; il valore contenuto nel registro B viene incrementato se questo non è un inizio di carattere.
I valori degli attributi della nuova posizione di sprite vengono confrontati col valore della variabile di sistema BATT (attributo di ostacolo 62406) e, se c'è coincidenza, si resettano i valori di CO1 e CO2 per la successiva chiamata alla routine MOVE e si pone l'attributo di sprite in questo indirizzo, usando il valore della variabile di sistema SPRAT 62407. Questa routine sfrutta le subroutine FIND1 e FIND2 per trovare l'indirizzo dell'attributo dell'inizio dello sprite.
Resta solo da stabilire quale figura deve essere visualizzata dalla routine SPRITE.

8.4 MAN?: GOLDMINE 64369

Questa routine usa anche le variabili di sistema MBYTE (numero di byte per figura: 62416/62417) e MADD (indirizzo iniziale dei dati relativi alle

quattro figure: 62408/62409). Supponendo di avere come coordinata x iniziale un multiplo di 8, possiamo sfruttare il valore contenuto nel registro L per trovare quale figura visualizzare. Il valore viene elaborato tramite l'istruzione *and 6*, che fornisce un risultato di 0,2,4 o 6. La cifra così ottenuta viene ulteriormente modificata usando *rra*, in modo da restituire un risultato finale di 0, 1, 2 o 3. Questo valore viene usato per trovare l'indirizzo iniziale della figura da visualizzare, che è posto nella variabile di sistema MAN1 a disposizione della routine SPRITE.
Per dare l'impressione di un movimento a passi di un pixel, saranno necessarie otto figure e una piccola modifica alla routine, consistente nella sostituzione di *and 6* con *and 7* e di *rra* con *nop*, in modo da selezionare una figura su otto.

in BASIC
Sapete ora come ottenere i dati dei vostri sprite. Questi dati vanno posti tramite dei POKE in un'area libera di memoria, preferibilmente nella zona alta, dove possono essere protetti abbassando la RAMTOP, prestando però la massima attenzione a non sovrapporsi all'area GOLDMINE. Potreste immagazzinarli nella prima linea del vostro programma in una REM fittizia di sufficiente lunghezza, ma non è una scelta raccomandabile perché i listati BASIC possono subire modifiche involontarie.
Se, ad esempio, scegliamo 48128 come indirizzo iniziale dei nostri 192 byte di dati della ruota, potremo usare i comandi DATA e READ per inserire con un POKE i valori in indirizzi consecutivi a ogni esecuzione del programma, o inserire i valori direttamente negli indirizzi di memoria usando un semplice programma di caricamento:

```
10 FOR a=48128 TO 48319
20 INPUT "DATA BYTE";byte
30 POKE a,byte
40 NEXT a
```

Quindi, come parte della nostra routine di SAVE, potremo salvare sul nastro i 192 byte di dati, in modo che un successivo LOAD li ricarichi negli indirizzi correnti:

```
9000 CLEAR 48127: LOAD ""CODE
9010 LOAD ""CODE: GOTO START
9020 SAVE "Basic" LINE 9000
9030 SAVE "Sprite"CODE 48128,192
9040 SAVE "Goldmine"CODE 62000,3240
9050 REM Per salvare il vostro programma battete GOTO 9020
```

Posti i dati degli sprite in memoria, possiamo predisporre le variabili di sistema:

HATT	62405
BATT	62406
SPRAT	62407
MADD	62408/62409
CO1	62410/62411
CO2	62412/62413
FORMAT	62414/62415
MBYTE	62416/62417

CO1 e CO2 andranno inizializzate alle coordinate dello stesso pixel, con il valore dell'asse x contenuto in 62410 e 62412 e il valore dell'asse y in 62411 e 62413, ricordando che il valore iniziale di x deve essere un multiplo di 8.
FORMAT è la misura della figura in caratteri; 62414 contiene la larghezza e 62415 l'altezza.
MADD è l'indirizzo iniziale delle 4 figure, con il byte meno significativo contenuto in 62408 e il byte più significativo in 62409. In questo esempio noi dividiamo 48128 per 256, ottenendo come risultato un quoziente intero (188 nel byte più significativo) e un resto (0 nel byte meno significativo).
HATT è il valore di attributo di un oggetto di cui si desideri rilevare la collisione col nostro sprite.
BATT è il valore di attributo di un oggetto tale da impedire il movimento del nostro sprite.
SPRAT è il valore di attributo del nostro sprite; normalmente il valore di PAPER sarà lo stesso dello schermo, ma notate che lo sprite lascerà, nel suo movimento lungo lo schermo, una scia di attributi. Comunque, esistono modi per evitare questo effetto.
Le routine MOVE, MOVE?, MAN? e SPRITE sono tutte disponibili separatamente usando le seguenti chiamate:

```
MOVE: LET a=USR 62303
```

Questa routine aggiornerà CO2 in funzione dei tasti premuti. Il valore restituito nella variabile a non ha alcun significato.

```
MOVE?: LET a=USR 62130
```

Questa routine controlla l'eventuale presenza di un attributo di barriera alla nuova posizione CO2, aggiornando/resettando CO1/CO2 conseguentemente. La routine predispone anche gli attributi di sprite.

MAN?: LET a=USR 62124

Questa routine trova l'indirizzo iniziale della figura da visualizzare da parte della routine SPRITE.

SPRITE: LET a=USR 62284

Questa routine svolge due funzioni:

a) se HITVAL è 1, la routine controlla la posizione di sprite per l'eventuale collisione su un pixel e ritorna con la variabile a=1 in caso di collisione, o a=0 in caso contrario.
b) se HITVAL è 0, viene visualizzato lo sprite. Per il normale uso BASIC, tuttavia, ho incluso 4 routine che dovrebbero soddisfare gran parte delle necessità.

1. MOVESPRITE (usando gli attributi di schermo ATTR)

LET a=USR 62000

Questa routine chiama le varie routine appena presentate nell'ordine corretto, ma considera come attributo di sprite quello già predisposto sullo schermo. Così, per muovere il vostro sprite lungo tutto lo schermo, potrete usare:

10 LET a=USR 62000: GOTO 10

2. MOVESPRITE (lascia una scia di attributi di sprite SPRAT)

LET a=USR 62012

Questa routine preleva i valori di attributi dello sprite dalla variabile di sistema SPRAT, ma lascia una scia di valori di SPRAT nel movimento. Questo effetto non sarà visibile se il colore di PAPER dello schermo è lo stesso di SPRAT.

3. MOVESPRITE (senza lasciare alcuna scia di attributo)

LET a=USR 62037

Questa routine cancellerà l'attributo di sprite prima di rivisualizzarlo; notate che durante il movimento dello sprite potrebbe essere presente un certo sfarfallio dei colori.

4. CHECKSPRITE (controllo posizione sprite)

LET a=USR 62082

Questa routine pone HITVAL a 1 e chiama la routine SPRITE. La variabile a contiene 1 in caso di collisione, 0 in caso contrario. Con questo ben presente in mente possiamo ora scrivere un programma dimostrativo per mostrare gli effetti di questi metodi:

```
   5 REM **********************
  10 REM * Dati degli SPRITE *
  15 REM **********************
  20 DATA 0,0,0,7,248,0,8,132,0,16,130,0,32,65,0,32,65,0
,32,65,0,39,199,0
  30 DATA 56,249,0,32,129,0,32,129,0,32,129,0,16,66,0,8,
68,0,7,248,0,0,0,0
  40 DATA 0,0,0,1,254,0,2,9,0,4,8,128,8,8,64,8,16,64,15,
16,64,8,240,64
  50 DATA 8,60,64,8,35,192,8,32,64,8,64,64,4,64,128,2,65
,0,1,254,0,0,0,0
  60 DATA 0,0,0,0,127,128,0,128,64,1,128,96,2,64,144,2,3
3,16,2,18,16,2,12,16
  70 DATA 2,12,16,2,18,16,2,33,16,2,64,144,1,128,96,0,12
8,64,0,127,128,0,0,0
  80 DATA 0,0,0,0,31,224,0,36,16,0,68,8,0,132,4,0,130,4,
0,130,60,0,131,196
  90 DATA 0,143,4,0,241,4,0,129,4,0,128,132,0,64,136,0,3
2,144,0,31,224,0,0,0
  91 REM ***************************
  95 REM * Predisposizione SPRITE *
  99 REM ***************************
 100 FOR a=48128 TO 48319
 110 READ b: POKE a,b: NEXT a
 115 REM ************************************************
 120 REM * Predisposizione chiamate al codice macchina *
 125 REM ************************************************
 130 LET move=62000
 140 LET check=62082
 150 LET sprite=62284
 160 LET man=62124
 170 LET hit=62388
 180 LET beep3=62148
 185 REM ***************************************
 190 REM * Predisposizione variabili di sistema *
 195 REM ***************************************
 215 DATA 23,32,48,0,188,0,0,0,0,3,2,48,0
 220 FOR a=62405 TO 62417
 230 READ b: POKE a,b: NEXT a
 240 LET b$=CHR$ 22+CHR$ 4+CHR$ 20+CHR$ 17+CHR$ 6+CHR$ 1
6+CHR$ 0+"AREA SICURA"
```

```
250 LET a$=CHR$ 22+CHR$ 15+CHR$ 15+CHR$ 17+CHR$ 2+CHR$
16+CHR$ 7+"AREA ESPLOSIVA"
260 REM ********************************
280 REM * Posizione iniziale di schermo *
285 REM ********************************
290 PRINT AT 10,10; PAPER 4; INK 0;"BARRIERA"
315 POKE hit,0
325 REM ************************
330 REM * Routine di movimento *
331 REM ************************
335 LET a=USR move
360 PRINT b$
370 PRINT a$
375 REM ****************************
380 REM * Controllo di collisione *
385 REM ****************************
390 IF NOT USR check THEN  GO TO 335
410 LET a=USR beep3
420 PRINT #0;AT 0,0;"Hai colpito una mina!"
430 PAUSE 0: CLS
435 RESTORE 215
440 GO TO 130
```

Il programma così come è presentato, dimostra la routine 1. Per dimostrare la routine 2, al 62000 nella linea 130 va sostituito 62012; per la routine 3 gli va invece sostituito 62037.
Noterete che *Area sicura* e *Area esplosiva* vanno rivisualizzate in seguito ad ogni movimento dello sprite, in quanto questo può sovrapporsi ai loro attributi. Consiglio di mantenere queste aree da rivisualizzare il più piccole possibile, per evitare il rallentamento dell'esecuzione. Provate per curiosità a rimuovere "Area sicura" e vedrete l'aumento di velocità!
Per aumentare ulteriormente la velocità, potreste trascurare il controllo di collisione a pixel e usare al suo posto il valore degli attributi. Nel nostro esempio su uno sprite 3×2, ciò vorrà dire esaminare solo i due caratteri di mezzo dello sprite, dato che questi conterranno sicuramente una parte di esso. Questo procedimento non è accurato come il rilevamento di collisione a pixel e la routine dovrà essere in codice macchina per fornire un aumento della velocità.

in Assembler
Se leggete le istruzioni dedicate ai programmatori BASIC, troverete il seguente ordine di chiamata per i tre tipi di movimenti di sprite:

Con uso degli attributi di schermo:	ld hl,64423
	ld (hl),13
	call MOVE
	call MOVE?

```
                        call MAN?
                        call SPRITE
```

Con una scia di SPRAT:

```
                        ld hl,64423
                        ld (hl),9
                        call MOVE
                        call MOVE?
                        call MAN
                        call SPRITE
```

Senza alcuna scia:

```
                        ld hl,64423
                        ld (hl),9
                        ld a,(SPRAT)
                        push af
                        ld a,(ATTR)
                        ld (SPRAT),a
                        call MOVE?    REM cancella SPRAT
                        call MOVE
                        call MOVE?
                        pop af
                        ld (SPRAT),a
                        call MOVE?    REM resetta SPRAT
                        call MAN?
                        call SPRITE
```

Il nostro esempio BASIC può essere riprodotto in codice macchina nella maniera seguente:

```
org 32512
equ 63224 PSTRING
equ 64313 MOVE
equ 64393 MOVE?
equ 64369 MAN?
equ 63388 SPRITE
equ 63281 CLS
equ 64530 BEEP3
equ 63704 BREAK
equ 62405 HATT
equ iy+13 HITVAR
equ iy+4 ATTR
Dati degli sprite
DATA1
defb 0 0 0 7 248 0 8 132 0
defb 16 130 0 32 65 0 32 65 0
defb 32 65 0 39 199 0 56 249 0
defb 32 129 0 32 129 0 32 129 0
```

```
defb 16 66 0 8 68 0 7 248 0
defb 0 0 0 0 0 0 1 254 0

defb 2 9 0 4 8 128 8 8 64
defb 8 16 64 15 16 64 8 240 64
defb 8 60 64 8 35 192 8 32 64
defb 8 64 64 4 64 128 2 65 0
defb 1 254 0 0 0 0 0 0 0
defb 0 127 128 0 128 64 1 128 96
defb 2 64 144 2 33 16 2 18 16
defb 2 12 16 2 12 16 2 18 16
defb 2 33 16 2 64 144 1 128 96
defb 0 128 64 0 127 128 0 0 0
defb 0 0 0 0 31 224 0 36 16
defb 0 68 8 0 132 4 0 130 4
defb 0 130 60 0 131 196 0 143 4
defb 0 241 4 0 129 4 0 128 132
defb 0 64 136 0 32 144 0 31 224
defb 0 0 0
*********************************
Predisposizione variabili
di sistema
DATA2
defb 23 32 48 0 127 0 0 0 0 3 2

defb 48 0
*********************************
Dati di visualizzazione

DATA3
defb 22 20 4
defs AREA SICURA
defb 0
DATA4
defb 22 15 15
defs AREA ESPLOSIVA
defb 0
DATA5
defb 22 10 10
defs BARRIERA
defb 0
*********************************
Inizializzazione
START
32762 F3          di
32763 FD 21 A6 F3 ld iy,62374

32767 FD 36 04 38 ld (iy+4),56
32771 CD 31 F7    call CLS
32774 21 C0 7F    ld hl,DATA2
```

```
32777 11 C5 F3     ld de,HATT
32780 01 0D 00     ld bc,13
32783 ED B0        ldir
32785 21 A7 FB     ld hl,64423
32788 36 0D        ld (hl),13
32790 FD 36 0D 00  ld (iy+13),0
*********************************
Visualizzazione
32794 FD 36 04 20  ld (iy+4),32
32798 21 EE 7F     ld hl,DATA5
32801 CD F8 F6     call PSTRING
LOOP
32804 CD 39 FB     call MOVE
32807 CD 89 FB     call MOVE?
32810 CD 71 FB     call MAN?
32813 CD 9C F7     call SPRITE
32816 FD 36 04 17  ld (iy+4),23
32820 21 DC 7F     ld hl,DATA4

32823 CD F8 F6     call PSTRING
32826 FD 36 04 30  ld (iy+4),48
32830 21 CD 7F     ld hl,DATA3
32833 CD F8 F6     call PSTRING
32836 CD D8 F8     call BREAK
32839 30 0C        jr nc,END
32841 FD 36 0D 01  ld (iy+13),1
32845 CD 9C F7     call SPRITE
32848 30 D2        jr nc,LOOP
*********************************
Fine routine
32850 CD 12 FC     call BEEP3
END
32853 FD 21 3A 5C  ld iy,23610
32857 FB           ei
32858 C9           ret
```

Per aumentare la velocità degli sprite, esistono vari metodi disponibili al programmatore in codice macchina:

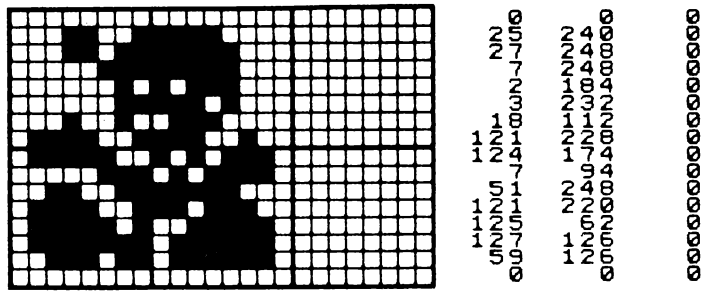
1. Usare una misura standard, in modo da evitare la necessità delle variabili di sistema MBYTE e FORMAT. Se lo sprite è sempre grande tre caratteri, le routine SPRITE e MOVE? possono essere riscritte per rimuovere il loop di "ampiezza" usando al suo posto *inc l* tre volte (è necessario controllare ogni volta l'eventuale raggiungimento del bordo destro dello schermo).
2. Limitarsi ad un effetto di avvolgimento solo in senso orizzontale.
3. Modificare la routine MOVE per leggere solo una semiriga di tasti, da sfruttare per il controllo dello sprite.

4. Non avere "Aree sicure".
5. Usare il rilevamento di collisione per attributi, analogo al rilevamento di barriere, ma controllare solo le colonne centrali di uno sprite da 3×2 caratteri, in quanto contengono sempre parte dello sprite.

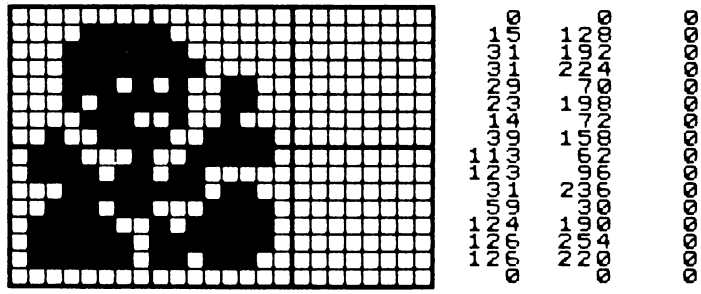
Metodo 2

L'uso di questo metodo è riservato ai soli programmatori in codice macchina, essendo esso più lento del metodo 1 e quindi possibile fonte di problemi per i programmatori BASIC.
Invece di usare, come nel primo esempio, 4 figure separate, questo metodo modifica i dati dello sprite via via che viene richiesto un aggiornamento della posizione di schermo. Il vantaggio di questo metodo è che possiamo riservare memoria solo per due figure, una destinata alla posizione iniziale dello sprite, e l'altra, inizialmente una copia, da manipolare. Lo svantaggio invece è nella sua lentezza e nella impossibilità di ottenere effetti di animazione, sebbene sia possibile alternare due o più sprite diversi per ottenere un'animazione a "fotogramma singolo". Ciò però richiede ulteriore memoria.
Questo metodo usa le stesse routine MOVE, MOVE? e SPRITE già sfruttate nel metodo 1, ma richiede una nuova routine MAN?. Poiché essa non è disponibile ai programmatori BASIC, non ho incluso questa routine in GOLDMINE; vi mostrerò, comunque, come ottenere una routine su misura per le vostre necessità.
Il primo passo consiste nel progettare gli sprite, un carattere più ampio del necessario. Per questa dimostrazione userò due sprite, che si sovrappongono di continuo per ottenere una parvenza di animazione. La forma dei miei sprite è mostrata nella Tavola 9.3. Noterete che questa volta ogni sprite occupa i primi 2×2 caratteri; è necessario lasciare intorno a ognuno una cornice di pixel e lasciar vuota la posizione del terzo carattere. Questi sprite rappresentano la posizione iniziale.
Come col metodo 1, immagazziniamo i byte di dati degli sprite in indirizzi consecutivi, ma questa volta creiamo un buffer della stessa misura contenente gli sprite modificati per la visualizzazione. La nostra routine utilizza l'indirizzo iniziale di ogni sprite in questo buffer BUF1 e BUF2 e l'indirizzo finale BUFEND. Nel movimento a sinistra e a destra, noi modifichiamo l'intera area di buffer, mediante rotazioni a destra/sinistra, in modo analogo alle routine di scroll della finestra, usando *rr(hl)* e *rl(hl)*.
Il seguente programma dimostra i principi di base del procedimento. Noterete che sono necessari due cont[illegible] prima di muovere i dati:

1. Usando OLDX, che è una copia di CO2 prima della chiamata a MOVE?,



(a)



(b)

Tavola 9.3 Le figure dei due sprite di esempio per il metodo 2

si controlla l'avvenuto movimento a sinistra, a destra, o nessun movimento.

2. Se è richiesto un movimento a destra, controlliamo che le coordinate x non siano passate alla successiva posizione di carattere, cioè che *and* 7 non fornisca come risultato 0. In tal caso, non solo muoviamo di un pixel a destra i dati, ma li spostiamo anche a sinistra di un carattere usando *lddr*, in effetti resettandoli alla posizione iniziale. Analogamente, per un movimento a sinistra noi controlliamo la successiva posizione di carattere e sfruttiamo *rl(hl)* e/o *lddr*.
   Per controllare il tasso di scambio fra sprite1 e sprite2, usiamo un contatore, incrementandolo ad ogni passaggio attraverso il loop e controllando lo stato del bit 6. Lo stato di questo bit viene usato per commutare l'indirizzo di sprite (MAN1) fra BUF1 e BUF2. Per ottenere una maggiore velocità, usate il bit 5 o il bit 4.

```
org 32512
equ 62412 CO2
equ 64313 MOVE
equ 64393 MOVE?
equ 62388 MAN1
equ 63388 SPRITE
equ 63281 CLS
equ 64530 BEEP3
equ 63704 BREAK
equ 62405 HATT
equ iy+13 HITVAR
equ iy+4 ATTR
Dati degli sprite
BUF1
defb 0 0 0 0 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0 0 0 0 0
BUF2
defb 0 0 0 0 0 0 0 0 0 0 0 0

defb 0 0 0 0 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0 0 0 0
BUFEND
defb 0
PIC
defb 0 0 0 25 240 0 27 248 0
defb 7 248 0 2 184 0 3 232 0
defb 18 112 0 121 228 0
defb 124 170 0 7 94 0 51 248 0
defb 121 220 0 125 62 0
defb 127 126 0 59 126 0 0 0 0
defb 0 0 0 15 128 0 31 192 0
defb 31 224 0 29 70 0 23 198 0
defb 14 72 0 39 158 0 113 62 0
defb 123 96 0 31 236 0 59 30 0
defb 124 190 0 126 254 0
defb 126 220 0 0 0 0
*********************************
Predisposizione variabili
di sistema
DATA2

defb 23 32 48 0 127 0 0 0 0 3 2
defb 48 0
OLDX
32705 00          nop
COUNTER
32706 00          nop
*********************************
```

```
START
32707 F3          di
32708 FD 21 A6 F3 ld iy,62374
32712 21 54 7F    ld hl,PIC
32715 11 00 7F    ld de,BUF1
32718 01 60 00    ld bc,96
32721 ED B0       ldir
32723 11 C5 F3    ld de,HATT
32726 01 0D 00    ld bc,13
32729 ED B0       ldir
32731 36 00       ld (hl),0
32733 23          inc hl
32734 36 00       ld (hl),0
32736 21 A7 FB    ld hl,64423

32739 36 0D       ld (hl),13
32741 FD 36 0D 00 ld (iy+13),0
32745 FD 36 04 38 ld (iy+4),56
32749 CD 31 F7    call CLS
Quale sprite
LOOP
32752 21 C2 7F    ld hl,COUNTER
32755 34          inc (hl)
32756 7E          ld a,(hl)
32757 CB 77       bit 6,a
32759 28 05       jr z,C1
32761 21 00 7F    ld hl,BUF1
32764 18 03       jr C2
C1
32766 21 30 7F    ld hl,BUF2
C2
32769 22 B4 F3    ld (MAN1),hl
32772 CD 39 FB    call MOVE
32775 CD 89 FB    call MOVE?
32778 CD 1B 80    call ADJUST
32781 CD 9C F7    call SPRITE

32784 CD D8 F8    call BREAK
32787 38 DB       jr c,LOOP
32789 FD 21 3A 5C ld iy,23610
32793 FB          ei
32794 C9          ret
ADJUST
32795 21 C1 7F    ld hl,OLDX
32798 3A CC F3    ld a,(CO2)
32801 BE          cp (hl)
32802 C8          ret z
32803 34          inc (hl)
32804 BE          cp (hl)
32805 28 1E       jr z,DESTRA
```

```
SINISTRA
32807 77          ld (hl),a
32808 E6 07       and 7
32810 FE 07       cp 7
32812 20 0B       jr nz,S1
32814 21 53 7F    ld hl,BUFEND
32817 54          ld d,h
32818 5D          ld e,l

32819 2B          dec hl
32820 01 5F 00    ld bc,95
32823 ED B8       lddr
S1
32825 21 53 7F    ld hl,BUFEND
32828 06 60       ld b,96
32830 A7          and a
S2
32831 CB 16       rl (hl)
32833 2B          dec hl
32834 10 FB       djnz S2
32836 C9          ret
DESTRA
32837 E6 07       and 7
32839 20 0B       jr nz,D1
32841 21 00 7F    ld hl,BUF1
32844 54          ld d,h
32845 5C          ld e,h
32846 23          inc hl
32847 01 5F 00    ld bc,95
32850 ED B0       ldir

D1
32852 21 00 7F    ld hl,BUF1
32855 06 60       ld b,96
32857 A7          and a
D2
32858 CB 1E       rr (hl)
32860 23          inc hl
32861 10 FB       djnz D2
32863 C9          ret
```

Sprite controllati dal computer

I due metodi precedenti sono progettati per il movimento di sprite controllati dal giocatore. Non c'è motivo per cui le stesse routine non possano essere usate per gli sprite controllati dal computer, i cui parametri potrebbero essere immagazzinati sotto forma di dati, con i primi due by-

te usati per costituire l'indirizzo base delle figure degli sprite (metodo 1), seguiti da tre byte per ogni sprite, contenenti le seguenti informazioni:

BYTE 1 : byte di attributi

Bit 0	→ HIT?	se uguale a 1, lo sprite è entrato in collisione, quindi non si effettua la visualizzazione se uguale a 0, si effettua la visualizzazione
Bit 1	→ X?	se uguale a 1, vengono aggiornate le coordinate x se uguale a 0, si passa al bit 3
Bit 2	→ +/−	se uguale a 1, allora x=x+1 se uguale a 0, allora x=x−1
Bit 3	→ Y?	se uguale a 1, vengono aggiornate le coordinate y se uguale a 0, si passa al bit 5
Bit 4	→ +/−	se uguale a 1, allora y=y+1 se uguale a 0, allora y=y−1
Bit 5		
Bit 6	→ colore di INK	
Bit 7		

BYTE 2 : coordinata x
BYTE 3 : coordinata y

Usando il metodo appena presentato, possiamo ottenere il movimento continuo di uno sprite in una certa direzione, fino al raggiungimento di una barriera o a una collisione. In caso di barriera, riconosciuta attraverso la routine MOVE?, potremmo variare casualmente i bit da 1 a 4 per ottenere una nuova posizione al di fuori della barriera stessa.
Questo principio è usato molto più facilmente dal metodo 1, perché il disegno dello sprite viene trovato direttamente dalle coordinate x. Il metodo 2 richiederebbe una modifica dello sprite per ognuna delle coordinate x prelevate dai dati.
Il seguente breve programma dimostra quanto esposto, con il computer che controlla 14 ruote di differenti colori in movimento in tutte le direzioni.

```
org 32512
equ 62412 X
equ 62413 Y
equ 64393 MOVE?
equ 64369 MAN?
equ 63388 SPRITE
equ 63281 CLS
equ 62408 MADD
equ 62409 MADDH
```

```
equ 63704 BREAK
equ 62405 HATT
equ 62407 SPRAT
Dati degli sprite
DATA1
defb 0 0 0 7 248 0 8 132 0
defb 16 130 0 32 65 0 32 65 0
defb 32 65 0 39 199 0 56 249 0
defb 32 129 0 32 129 0 32 129 0
defb 16 66 0 8 68 0 7 248 0
defb 0 0 0 0 0 0 1 254 0

defb 2 9 0 4 8 128 8 8 64
defb 8 16 64 15 16 64 8 240 64
defb 8 60 64 8 35 192 8 32 64
defb 8 64 64 4 64 128 2 65 0
defb 1 254 0 0 0 0 0 0 0
defb 0 127 128 0 128 64 1 128 96
defb 2 64 144 2 33 16 2 18 16
defb 2 12 16 2 12 16 2 18 16
defb 2 33 16 2 64 144 1 128 96
defb 0 128 64 0 127 128 0 0 0
defb 0 0 0 0 31 224 0 36 16
defb 0 68 8 0 132 4 0 130 4
defb 0 130 60 0 131 196 0 143 4
defb 0 241 4 0 129 4 0 128 132
defb 0 64 136 0 32 144 0 31 224
defb 0 0 0
**********************************
Predisposizione variabili
di sistema
DATA2
defb 23 32 56 0 127 0 0 0 0 3 2

defb 48 0
**********************************
COORDS
defb 0 127
defb 62 10 10 202 0 20
defb 166 50 70 130 200 50
defb 120 100 40 72 30 100
defb 58 25 25 14 240 130
defb 62 100 100 202 75 75
defb 166 125 125 130 90 90
defb 10 0 0 94 180 180
**********************************
Inizializzazione
START
32761 F3          di
32762 FD 21 A6 F3 ld iy,62374
```

```
32766 FD 36 04 38 ld (iy+4),56
32770 CD 31 F7    call CLS
32773 21 C0 7F    ld hl,DATA2
32776 11 C5 F3    ld de,HATT
32779 01 0D 00    ld bc,13

32782 ED B0       ldir
32784 21 A7 FB    ld hl,64423
32787 36 09       ld (hl),9
32789 FD 36 0D 00 ld (iy+13),0
*********************************
Acquisisce l'indirizzo
iniziale dello sprite
LOOP
32793 21 CD 7F    ld hl,COORDS
32796 7E          ld a,(hl)
32797 32 C8 F3    ld (MADD),a
32800 23          inc hl
32801 7E          ld a,(hl)
32802 32 C9 F3    ld (MADDH),a
*********************************
Acquisisce 14 parametri
di sprite
32805 06 0E       ld b,14
L1
32807 C5          push bc
32808 23          inc hl

Acquisisce il byte di attributo
32809 7E          ld a,(hl)
32810 23          inc hl
32811 1F          rra
32812 30 03       jr nc,X?
32814 23          inc hl
32815 18 48       jr NEXT
Controlla se occorre aggiornare
la coordinata X
X?
32817 1F          rra
32818 F5          push af
32819 30 07       jr nc,Y?
LX
32821 1F          rra
32822 30 03       jr nc,XM
XP
32824 34          inc (hl)
32825 18 01       jr Y?
XM
32827 35          dec (hl)
```

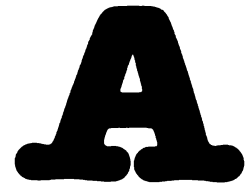
```
Pone la nuova coordinata X
Y?
32828 7E          ld a,(hl)
32829 32 CC F3    ld (X),a
32832 F1          pop af
Controlla se occorre aggiornare
la coordinata Y
32833 23          inc hl
32834 1F          rra
32835 1F          rra
32836 F5          push af
32837 30 15       jr nc,INK1
32839 1F          rra
32840 30 09       jr nc,YM
YP
32842 7E          ld a,(hl)
32843 3C          inc a
32844 FE C0       cp 192
32846 20 0B       jr nz,OK1
32848 AF          xor a
32849 18 08       jr OK1

YM
32851 7E          ld a,(hl)
32852 3D          dec a
32853 FE FF       cp 255
32855 20 02       jr nz,OK1
32857 3E BF       ld a,191
Pone la nuova coordinata Y
OK1
32859 77          ld (hl),a
INK1
32860 7E          ld a,(hl)
32861 32 CD F3    ld (Y),a
Predispone il colore di INK
dello sprite
INK
32864 F1          pop af
32865 1F          rra
32866 E6 07       and 7
32868 4F          ld c,a
32869 3A C7 F3    ld a,(SPRAT)
32872 E6 F8       and 248

32874 81          add a,c
32875 32 C7 F3    ld (SPRAT),a
32878 E5          push hl
Visualizza lo sprite
32879 CD 71 FB    call MAN?
32882 CD 89 FB    call MOVE?
```

```
32885 CD 9C F7     call SPRITE
32888 E1           pop hl
NEXT
32889 C1           pop bc
32890 CD D8 F8     call BREAK
32893 30 04        jr nc,END
32895 10 A6        djnz L1
32897 18 96        jr LOOP
END
32899 FD 21 3A 5C  ld iy,23610
32903 FB           ei
32904 C9           ret
```

Una volta compresi a fondo i principi del movimento degli sprite, troverete senz'altro facile adattare la routine alle vostre necessità.
La parte più difficile consisterà nel disegno degli sprite e nella ideazione di nuovi giochi. Vi auguro buona fortuna e tante ore di divertimento davanti al vostro computer.

Appendice

Guida rapida al sistema GOLDMINE



Questa appendice contiene le descrizioni sintetiche delle routine GOLDMINE; essa è organizzata in modo da essere di veloce consultazione ed è divisa in 4 parti:

1. Routine in sostituzione di quelle originali della ROM dello ZX Spectrum 48K;
2. Routine di utilità;
3. Un gruppo di utili routine FIND per la gestione dello schermo;
4. Un elenco delle variabili di sistema GOLDMINE con i relativi indirizzi e funzioni.

In tutte le schede, a fianco del nome della routine, sono stati indicati il suo indirizzo ed il paragrafo del libro in cui vengono spiegate dettagliatamente.

GOLDMINE dispone anche di un gruppo di routine che permette un movimento a pixel di sprite di qualunque misura su tutto lo schermo, con possibilità di avvolgimento nelle quattro direzioni, sia sotto il controllo del giocatore che del computer. Queste routine sono troppo complesse per essere incluse nella guida rapida; si consiglia quindi di esaminare il Capitolo 8 prima di tentarne l'uso.

A.1 Routine di sostituzione di quelle originali della ROM dello ZX Spectrum

Il primo gruppo di routine sostituisce quelle presenti nella ROM dello Spectrum con le seguenti differenze:

1. Lo schermo dei caratteri è definito come 24 linee (numerate da 0 a 23) per 32 colonne (numerate da 0 a 31), cioè senza linee di input. 0,0 è l'angolo superiore sinistro.
2. Lo schermo dei pixel può essere considerato come uno schermo visibile, alto 192 pixel numerati da 0 a 191 e largo 256 pixel numerati da 0 a 255. Questo rende possibile disegnare anche al di fuori dallo schermo visibile, permettendo alle linee e ai cerchi tracciati l'uscita (e il successivo rientro) dallo schermo visibile.
   0,0 rappresenta l'angolo superiore sinistro dello schermo visibile.
   255,191 rappresenta l'angolo inferiore destro dello schermo visibile.
   0,192 rappresenta l'angolo superiore sinistro dello schermo invisibile.
   255,255 rappresenta l'angolo inferiore destro dello schermo invisibile.
3. La nuova routine PRINT AT richiede la posizione iniziale sotto forma di colonna/linea; la routine Spectrum richiede invece linea/colonna.
4. Il registro IY è usato come puntatore all'inizio delle nuove variabili di sistema e dovrà contenere 62374 per molte delle routine (LSCAN, KEY...). Per le routine della ROM dovrà contenere normalmente 23610.

I programmatori BASIC dovrebbero notare che, dopo l'esecuzione di una routine in codice macchina, si ritorna al BASIC con il valore contenuto nella coppia di registri BC disponibile al programmatore.
Svariati sono i modi di utilizzo di questo valore:

1. Nella routine SCREEN$, ad esempio, usate

   LET *variabile stringa*=CHR$ USR 62220

   Questo restituirà, nella variabile stringa scelta, il carattere il cui codice era contenuto nella coppia di registri BC.
2. Quando il valore restituito è numerico, per esempio di attributi, usate

   LET *variabile*=USR *indirizzo*

3. Quando il valore restituito può essere 0, come ad esempio dalle routine POINT e SCREEN$, potete usare

   IF USR *indirizzo* THEN...

   oppure

   IF NOT USR *indirizzo* THEN....

Att 65183

Paragrafo 4.1

INPUT	
Registri	Non è richiesta alcuna predisposizione
Variabili di sistema	È necessario predisporre ATL/C
OUTPUT	
Registri	A contiene il valore dell'attributo
Variabili di sistema	ATT? contiene il valore dell'attributo
Registri usati	A, BC, HL
Commento	Restituisce il valore dell'attributo di schermo su uno schermo di 24×32 caratteri
BASIC USR	62238 LET a=62238 IF USR 62238 THEN...

BEEP 63501

Paragrafo 5.1

INPUT	
Registri	HL e DE contengono la combinazione durata/timbro
Variabili di sistema	Nessuna
OUTPUT	
Registri	Nessuno
Variabili di sistema	Nessuna
Registri usati	A, BC, DE, HL, IX
Commento	Produce note musicali analogamente al comando BEEP. Il valore per HL e DE può essere ottenuto dalla tabella nel Capitolo 5.
BASIC USR	Non disponibile

BP1 64474

Paragrafo 5.2

INPUT Registri	
Variabili di sistema	Vanno predisposti HL, BC e DE
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL, IX
Commento	L'effetto sonoro varia col valore delle variabili di sistema HL, BC, DE
BASIC USR	62136 LET a=USR 62136

BP2 64498

Paragrafo 5.3

INPUT Registri	
Variabili di sistema	Vanno predisposti DE e BC (MSB)
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, B, DE
Commento	L'effetto sonoro varia in funzione del valore delle variabili di sistema. Uscita diretta dall'altoparlante. Viene conservato il colore della cornice
BASIC USR	62142 LET a=USR 62142

BP3 64530

Paragrafo 5.4

INPUT Registri	
Variabili di sistema	Va predisposto HL
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, HL
Commento	Rumore bianco di durata dipendente dal valore di HL. La cornice lampeggia durante l'emissione del suono
BASIC USR	62148 LET a=USR 62148

BP4 64557

Paragrafo 5.5

INPUT Registri	
Variabili di sistema	Vanno predisposti HL, BC, DE (LSB)
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, B, DE, HL
Commento	L'effetto sonoro varia in funzione dei valori delle variabili di sistema. Viene conservato il colore della cornice
BASIC USR	62154 LET a=USR 62154

CIRCLE 63787

Paragrafo 3.2

INPUT	
Registri	Nessuna
Variabili di sistema	RAD, XY,Y, FLAG
OUTPUT	
Registri	Nessuna
Variabili di sistema	Nessuna
Registri usati	A, BC, DE, HL
Commento	Traccia un cerchio di centro x,y con raggio nel campo 0-100. Il centro può essere al di fuori dello schermo su una griglia di 256×256 pixel. Il cerchio può anche fuoriuscire dallo schermo
BASIC USR	62112 LEF a=USR 62112

CLS 63281

Paragrafo 2.4

INPUT	
Registri	
Variabili di sistema	ATTR
OUTPUT	
Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Svuota l'intero schermo (24×32) usando per attributo il valore di ATTR
BASIC USR	62100 LET a=USR 62100

DRAW 64210

Paragrafo 3.3

INPUT	
Registri	
Variabili di sistema	XY,Y, XM,YM e FLAG
OUTPUT	
Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL e H'L'
Commento	Traccia una linea da x,y a xm,ym su una griglia di 256×256 pixel. Le linee fuoriuscite dallo schermo possono rientrarvi
BASIC USR	62106 LET a=USR 62106

KEY 63043

Paragrafo 1.1

INPUT	
Registri	IY dovrà contenere 62374 (puntatore alle variabili di sistema)
Variabili di sistema	
OUTPUT	
Registri	A contiene il codice dell'ultimo tasto premuto
Variabili di sistema	KEYV contiene il codice dell'ultimo tasto premuto
Registri usati	A, BC, DE, HL IY
Commento	Restituisce il codice dell'ultimo tasto premuto in modo normale. Restituisce il valore 0 se non è stato premuto alcun tasto, o se è stato premuto CAPS SHIFT, SYMBOL SHIFT, o ENTER Usa la subroutine SCAN 63057
BASIC USR	62264 1. LET a$=CHR$ USR 62264 2. IF USR 62264 THEN .....

LETT 65210

Paragrafo 1.2

INPUT	
Registri	IY dovrà contenere 62374 (puntatore alle variabili di sistema)
Variabili di sistema	
OUTPUT	
Registri	A contiene il codice dell'ultimo tasto alfabetico premuto
Variabili di sistema	KEYV contiene il codice dell'ultimo tasto alfabetico premuto
Registri usati	A, BC, DE, HL IY
Commento	Ritorna solo in seguito alla pressione di un tasto alfabetico
BASIC USR	62254 LET a$=CHR$ USR 62254

LSCAN 63233

Paragrafo 1.4

INPUT	
Registri	IY dovrà contenere 62374 (puntatore alle variabili di sistema)
Variabili di sistema	
OUTPUT	
Registri	A=0-31 E=stato di ogni semiriga
Variabili di sistema	FLAG1 e FLAG2 (vedi Capitolo 9 per l'interpretazione dei valori)
Registri usati	A, BC, E IY
Commento	Restituisce nell'accumulatore un valore rappresentante la codifica dello stato delle 8 semirighe. Usato nelle routine di controllo sprite
BASIC USR	62274 LET a=USR 62274

NUMB 65195

Paragrafo 1.3

INPUT	
Registri	IY dovrà contenere 62374 (puntatore alle variabili di sistema)
Variabili di sistema	
OUTPUT	
Registri	A contiene il codice dell'ultimo tasto numerico premuto
Variabili di sistema	KEYV contiene il codice dell'ultimo tasto numerico premuto
Registri usati	A, BC, DE, HL IY
Commento	Ritorna solo in seguito alla pressione di un tasto numerico
BASIC USR	62244 LET a$=CHR$ USR 62244

PBST 65021
PBST1 65024

Paragrafo 2.8
Paragrafo 2.9

INPUT	
Registri	HL contiene l'indirizzo iniziale (solo PBST1)
Variabili di sistema	Va predisposto Cc Va predisposto FLAG Va predisposto ATTR Va predisposto STDATA
OUTPUT	
Registri	
Variabili di sistema	Viene aggiornata Cc. Se l'ultima posizione era l'angolo inferiore destro dello schermo, riprenderà da 0,0
Registri usati	A, BC, DE, HL
Commento	Vedi il Capitolo 2 per l'uso
BASIC USR	Vedi il Capitolo 2 per l'uso

PLOT 63346 — Paragrafo 3.1
PLOT1 63350 — Paragrafo 3.1

INPUT	
Registri	Va predisposta la coppia di registri BC (solo PLOT1)
Variabili di sistema	Va predisposto XY/Y Va predisposto FLAG
OUTPUT	
Registri	
Variabili di sistema	
Registri usati	A, BC, HL
Commento	Visualizza un pixel su una riga di 256×192 pixel (ma permette la visualizzazione anche al di fuori dello schermo su una griglia di 256×256 pixel; vedi DRAW e CIRCLE). È possibile ottenere OVER e INVERSE usando FLAG. 0,0 rappresenta l'angolo superiore sinistro
BASIC USR	62106 LET a=USR 62106

POINT 63716 — Paragrafo 4.2

INPUT	
Registri	
Variabili di sistema	Va predisposto XY/Y
OUTPUT	
Registri	A=0 o 1 in funzione dello stato del pixel
Variabili di sistema	Predispone POIV
Registri usati	A, BC, HL
Commento	Restituisce lo stato di un pixel su una griglia di 256×192 pixel. 0=spento 1=acceso
BASIC USR	62297 1. PRINT USR 62297 2. LET a=USR 62297 3. IF USR 62297 THEN ...

PRB 64903

Paragrafo 2.7

INPUT	
Registri	
Variabili di sistema	Va predisposto Cc Va predisposto FLAG Va predisposto ATTR
OUTPUT	
Registri	
Variabili di sistema	Viene aggiornata Cc. Se l'ultima posizione era l'angolo inferiore destro dello schermo, riprenderà da 0,0
Registri usati	A, BC, DE, HL
Commento	Vedi il Capitolo 2 per l'uso
BASIC USR	Vedi il Capitolo 2 per l'uso

PRINT 63091

Paragrafo 2.1

INPUT	
Registri	A contiene il codice del carattere
Variabili di sistema	Va predisposto COL/LINE Va predisposto FLAG Va predisposto ATTR
OUTPUT	
Registri	Viene salvato HL
Variabili di sistema	Vengono aggiornate COL/LINE. Se l'ultima posizione era in 23,31, riprenderanno da 0,0
Registri usati	A, BC, DE, HL
Commento	Permette la visualizzazione su uno schermo di 24 linee per 32 colonne; è possibile definire OVER e INVERSE tramite FLAG
BASIC USR	Non disponibile

PRSPC 63310

Paragrafo 2.3

INPUT	
Registri	BC contiene il numero di spazi richiesti
Variabili di sistema	Va predisposto COL/LINE Va predisposto FLAG Va predisposto ATTR
OUTPUT	
Registri	BC=0
Variabili di sistema	Vengono aggiornate COL/LINE. Se l'ultima posizione era in 23,31, riprenderanno da 0,0
Registri usati	A, BC, DE, HL
Commento	Visualizza una stringa di spazi. Usato per svuotare parti dello schermo
BASIC USR	Non disponibile

PSTRING 63224

Paragrafo 2.2

INPUT	
Registri	HL contiene l'indirizzo iniziale dei dati della stringa
Variabili di sistema	Va predisposto COL/LINE Va predisposto FLAG Va predisposto ATTR
OUTPUT	
Registri	
Variabili di sistema	Vengono aggiornate COL/LINE. Se l'ultima posizione era in 23,31, riprenderanno da 0,0
Registri usati	A, BC, DE, HL
Commento	Visualizza una stringa di caratteri ASCII fino al raggiungimento del byte 0 segnalatore di fine (*nop*). È ammesso PRINT AT colonna/linea
BASIC USR	Non disponibile

RAND 63738

Paragrafo 6.5

INPUT Registri	
Variabili di sistema	Va predisposto RND
OUTPUT Registri	BC contiene il numero casuale (da 0 a RND-1)
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Produce numeri interi pseudo casuali nel campo da 0 a RND-1
BASIC USR	Non disponibile

SCR1 65089

Paragrafo 4.3

INPUT Registri	
Variabili di sistema	Va predisposto SCR$
OUTPUT Registri	A contiene il codice del carattere
Variabili di sistema	CHR? contiene il codice del carattere trovato
Registri usati	A, BC, DE, HL
Commento	Restituisce il codice di CHR, nel campo da 32 a 127; nel caso il carattere non appartenga a questo campo, passa alla routine SCR2. Opera su una griglia di 24×32 caratteri
BASIC USR	62220 1. LET a$=CHR$ USR 62220 2. LET a=USR 62220 3. IF USR 62220 THEN ....

SCR2 65114

Paragrafo 4.4

INPUT	
Registri	
Variabili di sistema	Va predisposto SCR$
OUTPUT	
Registri	BC contiene il codice del carattere
Variabili di sistema	CHR? contiene il codice del carattere trovato
Registri usati	A, BC, DE, HL
Commento	Restituisce il codice dei caratteri GOLDMINE nel campo da 32 a 100 o, se il carattere non appartiene a questo campo, passa alla routine SCR3. Opera su una griglia di 24×32 caratteri
BASIC USR	62226 1. LET a$=CHR$ USR 62226 2. LET a=USR 62226 3. IF USR 62226 THEN .....

SCR3 65138

Paragrafo 4.5

INPUT	
Registri	
Variabili di sistema	Va predisposto SCR$
OUTPUT	
Registri	A contiene il codice del carattere
Variabili di sistema	CHR? contiene il codice del carattere trovato
Registri usati	A, BC, DE, HL
Commento	Restituisce il codice UDG nel campo da 144 a 164. Se il carattere non appartiene a questo campo, viene restituito uno 0. Usa una griglia di 24×32 caratteri
BASIC USR	62232 1. LET a$=CHR$ USR 62232 2. LET a=USR 62232 3. IF USR 62232 THEN ....

A.2 Routine di utilità

Questo secondo gruppo di routine è costituito da quelle più frequentemente usate nei giochi d'avventura, ma limitato a quelle ritenute più utili nel movimento a passi di un pixel.
Non è incluso il movimento a passi di un carattere poiché è relativamente semplice e si ritiene pertanto che il lettore abbia già prodotto le proprie routine. Se questo non è ancora il vostro caso, documentatevi leggendo il mio precedente libro intitolato: *Tecniche avanzate in Assembler per giochi veloci con lo ZX Spectrum.*

BREAK 63704 Paragrafo 2.5

INPUT Registri Variabili di sistema	
OUTPUT Registri Variabili di sistema	
Registri usati	A
Commento	Controlla la pressione di CAPS SHIFT e BREAK. Se entrambi sono premuti, ritorna con carry = 1, altrimenti con carry = 0
BASIC USR	Non disponibile

C/UP 63587

Paragrafo 6.1

INPUT	
Registri	IY dovrà contenere il valore 62734
Variabili di sistema	CLUP BUF 3 (vedi Capitolo 6)
OUTPUT	
Registri	
Variabili di sistema	COL/LINE aggiornate
Registri usati	A, BC, DE, HL, IY
Commento	Routine di conteggio che incrementa una stringa di numeri ASCII e li visualizza AT COL, LINE, come definito da CLUP
BASIC USR	Vedi Capitolo 6

C/DN 63635

Paragrafo 6.2

INPUT	
Registri	IY dovrà contenere il valore 62734
Variabili di sistema	CLDN BUF 4 (vedi Capitolo 6)
OUTPUT	
Registri	
Variabili di sistema	COL/LINE aggiornate
Registri usati	A, BC, DE, HL, IY
Commento	Routine di conteggio che decrementa una stringa di numeri ASCII e li visualizza AT COL, LINE come definito da CLDN
BASIC USR	Vedi Capitolo 6

Chdn0 63690

Paragrafo 6.4

INPUT Registri Variabili di sistema	 BUF 4 (vedi Capitolo 6)
OUTPUT Registri Variabili di sistema	 A contiene 0 se è stato raggiunto il minimo
Registri usati	A, HL
Commento	Controlla se tutti i numeri ASCII in una stringa sono "0"
BASIC USR	Vedi Capitolo 6

Chr 65035

Paragrafo 4.3

INPUT Registri Variabili di sistema	
OUTPUT Registri Variabili di sistema	 B contiene il codice ASCII del carattere trovato
Registri usati	A, B, BC, DE, HL
Commento	Controlla la corrispondenza di un blocco di byte sullo schermo con un carattere del generatore. Ritorna con carry = 1 in caso negativo e carry = 0 in caso affermativo
BASIC USR	non disponibile

Chup9 63676

Paragrafo 6.3

INPUT Registri Variabili di sistema	 BUF 3 (vedi Capitolo 6)
OUTPUT Registri Variabili di sistema	 A contiene 0 se è stato raggiunto il massimo
Registri usati	A, HL
Commento	Controlla se tutti i numeri ASCII in una stringa sono "9"
BASIC USR	Vedi Capitolo 6

Numb? 63628

Paragrafo 6.1

INPUT Registri Variabili di sistema	 A contiene il codice ASCII del carattere da verificare
OUTPUT Registri Variabili di sistema	 A contiene il codice ASCII del carattere verificato
Registri usati	A
Commento	Controlla che il codice ASCII nel registro A sia un codice numerico (48–57). Ritorna con carry = 1 in caso positivo, carry = 0 in caso negativo
BASIC USR	Non disponibile

SWAP 64593

Paragrafo 2.6

INPUT Registri	
Variabili di sistema	Va predisposto OLD Va predisposto NEW
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, DE, HL
Commento	Scambia i valori degli attributi contenuti in OLD con quelli contenuti in NEW su uno schermo di 24×32 caratteri
BASIC USR	62160 LET a=USR 62160

WDR 64700

Paragrafo 7.2

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Roll verso il basso di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (1-192)
BASIC USR	62172 LET a=USR 62172

WDS 64861

Paragrafo 7.6

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Scroll verso il basso di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62196 LET a=USR 62196

WLR 64803

Paragrafo 7.4

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Roll a sinistra di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62184 LET a=USR 62184

WLS 64889

Paragrafo 7.8

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Scroll a sinistra di un pixel di una finestra La finestra è definita da: Cb/Yb=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62208 LET a=USR 62208

WRR 64765

Paragrafo 7.3

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Roll a sinistra di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (1-192)
BASIC USR	62184 LET a=USR 62184

WRS 64875

Paragrafo 7.7

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Scroll a destra di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62202 LET a=USR 62202

WUR 64614

Paragrafo 7.1

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Roll verso l'alto di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62166 LET a=USR 62166

WUS 64847

Paragrafo 7.5

INPUT Registri	
Variabili di sistema	Va predisposto Ca/Ya Va predisposto Cb/Yb
OUTPUT Registri	
Variabili di sistema	
Registri usati	A, BC, DE, HL
Commento	Scroll verso l'alto di un pixel di una finestra La finestra è definita da: Ca/Ya=colonna (0-31), linea (0-191) dell'angolo superiore sinistro della finestra Cb/Yb=ampiezza finestra (1-32), altezza (2-192)
BASIC USR	62190 LET a=USR 62190

A.3 Routine di gestione dello schermo

Quest'ultimo gruppo di subroutine consente al programmatore di manipolare lo schermo in quanto permette di convertire le coordinate dello schermo, espresse sotto forma di pixel o di caratteri, in indirizzi di schermo o di attributo. Esse sono le routine FIND, che forniscono l'indirizzo del byte superiore o inferiore, utilizzate in tutti i capitoli del libro.

FIND1 63322

INPUT Registri	BC contiene le coordinate x,y del pixel
Variabili di sistema	Nessuna
OUTPUT Registri	HL contiene l'indirizzo di schermo del bit corrispondente al pixel BC contiene x, y
Variabili di sistema	Nessuna
Registri usati	A, BC, HL
Commento	Trova un indirizzo di schermo partendo dalle coordinate x, y. Il bit richiesto può essere calcolato partendo dal valore di x (vedi Capitolo 3)
BASIC USR	Non disponibile

FIND2 63214

INPUT	
Registri	HL contiene l'indirizzo di schermo
Variabili di sistema	Nessuna
OUTPUT	
Registri	HL contiene l'indirizzo di attributo
Variabili di sistema	Nessuna
Registri usati	A, HL
Commento	Trova un indirizzo di attributo partendo da un indirizzo di schermo
BASIC USR	Non disponibile

FIND3 63476

INPUT	
Registri	HL contiene un indirizzo di schermo
Variabili di sistema	Nessuna
OUTPUT	
Registri	HL contiene l'indirizzo di schermo della linea di pixel sottostante (nella stessa colonna di carattere)
Variabili di sistema	Nessuna
Registri usati	A, HL
Commento	Se l'indirizzo di schermo in entrata corrisponde alla linea di pixel 191, la linea inferiore sarà linea 0; si determina l'effetto di avvolgimento
BASIC USR	Non disponibile

FIND4 64673

INPUT	
Registri	HL contiene l'indirizzo di schermo
Variabili di sistema	Nessuna
OUTPUT	
Registri	HL contiene l'indirizzo di schermo
Variabili di sistema	Nessuna
Registri usati	A, HL
Commento	Se l'indirizzo di schermo in entrata è alla linea di pixel 0, la linea soprastante sarà linea 191; si determina l'effetto di avvolgimento
BASIC USR	Non disponibile

FIND5 65074

INPUT	BC contiene COL/LINE del carattere
Registri	C=COL (0-31): B=LINE (0-23)
Variabili di sistema	Nessuna
OUTPUT	DE contiene l'indirizzo di schermo del primo byte del carattere
Registri	BC contiene COL/LINE
Variabili di sistema	Nessuna
Registri usati	A, HL, DE
Commento	Usata in SCREEN$
BASIC USR	Non disponibile

FIND6 65167

INPUT Registri Variabili di sistema	BC contiene LINE/COL del carattere C=LINE (0-23): B=COL (0-31) Nessuna
OUTPUT Registri Variabili di sistema	HL contiene l'indirizzo di attributo BC contiene LINE/COL Nessuna
Registri usati	A, HL
Commento	Trova un indirizzo di attributo partendo dalle coordinate
BASIC USR	Non disponibile

A.4 Variabili di sistema

Note: o Deve essere predisposta prima della chiamata della routine relativa
x Usata dalla routine relativa, non richiede predisposizione
ox Può essere cambiata dal programmatore per puntare a un diverso generatore di caratteri

Nome	Indirizzo	Note	Funzione
KEYV	62374	x	Contiene il codice dell'ultimo tasto premuto dopo la chiamata alla routine KEY
COL/LINE	62375/6	o	Le coordinate di schermo della posizione di visualizzazione corrente
FLAG	62377	o	Contiene lo stato di OVER e INVERSE Bit 0 = 1 OVER 1 Bit 0 = 0 OVER 0 Bit 1 = 1 INVERSE 1 Bit 1 = 0 INVERSE 0
ATTR	62378	o	Il valore corrente di attributo
SCREEN	62379/80	x	Usata nella routine PRINT per contenere l'indirizzo di schermo della posizione corrente di PRINT
KNO	62381	x	Usata nella subroutine SCAN per contenere il numero di semirighe aventi tasti premuti
FLAG1	62382	x	Usata nella routine LSCAN per contenere lo stato di ogni semiriga
FLAG2	62383	x	Usata per contenere la versione modificata di FLAG1
XY/Y	62384/5	o	Contiene le coordinate di PLOT correnti Campo di y=0-255 (visualizzabili 0-191) Campo di x=0-255 0,0=angolo superiore sinistro
POIV	62386	x	Contiene lo stato del pixel x,y dopo l'esecuzione della routine POINT Valore 0=spento Valore 1=acceso
HITVAR	62387	o	Usata nella routine di visualizzazione di sprite Se bit 0=0, lo sprite viene visualizzato Se bit 0=1, si esegue un controllo di collisione

Nome	Indirizzo	Note	Funzione
MAN1	62388/9	o	Contiene l'indirizzo iniziale dello sprite attualmente visualizzato
RND	62390/1	o	Contiene il campo di variazione richiesto per i numeri casuali. La routine RAND restituisce INT (RND – 1)
SEED	62392/3	x	Usata nella routine RAND
RAD	62394	o	Il raggio del cerchio che si desidera tracciare
X2	62395/6	x	Usata nella routine CIRCLE per contenere il valore di x↑2
Y2	62397/8	x	Usata nella routine CIRCLE per contenere il valore di y↑2
XM/YM	62399/400	x	Usata nella routine CIRCLE per contenere le coordinate di PLOT dei punti della circonferenza
R2	62401/2	x	Usata nella routine CIRCLE per contenere il valore di r↑2
Cc	62403/4	o	Contiene la colonna di caratteri/riga di pixel per la routine PRB (vedi Capitolo 7) 62403 contiene la colonna 62404 contiene la riga
HATT	62405	o	Contiene il valore di attributo di uno sprite o di un carattere che segnalerà una collisione se un suo pixel verrà ad occupare le stesse coordinate di uno sprite controllato
BATT	62406	o	Contiene il valore di attributo di un carattere o di uno sprite che impedirà il movimento di uno sprite controllato
SPRAT	62407	o	Contiene il valore di attributo dello sprite correntemente visualizzato
MADD	62408/9	o	Contiene l'indirizzo iniziale della serie di figure relative allo sprite corrente
CO1	62410/1	o	Contiene le coordinate x,y della figura corrente di sprite

Nome	Indirizzo	Note	Funzione
CO2	62412/3	o	Inizializzata alle stesse coordinate x,y di CO1. Modificata come necessario dalla routine di movimento degli sprite
FORMAT	62414/5	o	Contiene la misura dello sprite corrente, in caratteri 62414=larghezza: 62415=altezza
MBYTE	62416/7	o	Contiene il numero di byte di dati della figura di sprite corrente
BC	62418/9	o	Contiene il valore per la coppia di registri BC nelle routine di effetti sonori
HL	62420/1	o	Come sopra per la coppia di registri HL
DE	62422/3	o	Come sopra per la coppia di registri DE
BDRCL	62424	o	Posta al colore corrente della cornice. Usata nel comando BEEP
OLD	62425	o	Contiene il vecchio attributo per la routine SWAP
NEW	62426	o	Contiene il nuovo attributo per la routine SWAP
Ca/Ya	62427/8	o	Usata nelle routine di roll/scroll della finestra per definirne il byte superiore sinistro Ca=Colonna (0-31) Ya=Linea (0-191)
Cb/Yb	62429/30	o	Usata nelle routine di roll/scroll della finestra per definirne la misura Cb=larghezza finestra (1-32) tale che Ca+Cb<33 Yb=altezza finestra (2-192)
STDATA	62431/2	o	Contiene l'indirizzo iniziale dei dati per la routine PBST (per uso dal BASIC)
SCR$	62433/4	o	Contiene COL/LINE per la routine SCREEN$ 62433=COL (0-31) 62434=LINE (0-23)
CHR?	62435	o	Contiene il codice dell'ultimo carattere trovato dalla routine SCREEN$, o 0 se non è stato trovato alcun carattere

Nome	Indirizzo	Note	Funzione
ATL/C	62436/7	o	Contiene LINE/COL del carattere di cui si richiede l'attributo 62436=LINE (0-23) 62437=COL (0-31)
ATT?	62438	o	Contiene il valore di attributo dello spazio di carattere definito da ATC/L
CHAROM	62439/40	ox	Contiene l'indirizzo iniziale del nuovo generatore di caratteri meno 256
CHARS	62441/2	ox	Contiene l'indirizzo iniziale del generatore di caratteri Spectrum meno 256
UDGS	62443/4	ox	Contiene l'indirizzo iniziale dei normali UDG Spectrum
CLUP	62445/6	o	Contiene COL/LINE della posizione iniziale di visualizzazione per la routine di conteggio crescente 62445=COL (0-31) 62446=LINE (0-23)
CLDN	62447/8	o	Come CLUP, ma per la routine di conteggio decrescente

Appendice

Listati delle routine GOLDMINE

B

```
PRINTER NOW ENABLED
org 49000 62000
********************************
Routine chiamabili dal BASIC
********************************
Movesprite
(usando gli attributi
di schermo ATTR)

62000 F3          di
62001 FD 21 A6 F3 ld iy,62374
62005 21 A7 FB    ld hl,64423
62008 36 0D       ld (hl),13
62010 18 0A       jr Bsprite
********************************
Movesprite
(lasciando una scia di
attributi di sprite SPRAT)

62012 F3          di
62013 FD 21 A6 F3 ld iy,62374

62017 21 A7 FB    ld hl,64423
62020 36 09       ld (hl),9
Bsprite
62022 CD 39 FB    call MOVE
62025 CD 89 FB    call MOVE?
62028 CD 71 FB    call MAN?
62031 CD 9C F7    call SPRITE
```

```
62034 C3 81 F3     jp RET1
*******************************
Movesprite
(senza lasciare alcuna scia)

62037 F3           di
62038 FD 21 A6 F3  ld iy,62374
62042 21 A7 FB     ld hl,64423
62045 36 09        ld (hl),9
62047 3A C7 F3     ld a,(SPRAT)
62050 F5           push af
62051 3A AA F3     ld a,(ATTR)
62054 32 C7 F3     ld (SPRAT),a
62057 CD 89 FB     call MOVE?
62060 CD 39 FB     call MOVE
62063 CD 89 FB     call MOVE?
62066 F1           pop af
62067 32 C7 F3     ld (SPRAT),a
62070 CD 89 FB     call MOVE?
62073 CD 71 FB     call MAN?
62076 CD 9C F7     call SPRITE
62079 C3 81 F3     jp RET1
*******************************
Controllo posizione sprite

62082 F3           di
62083 FD 21 A6 F3  ld iy,62374
62087 FD 36 0D 01  ld (iy+13),1
62091 CD 9C F7     call SPRITE
62094 FD 7E 0D     ld a,(iy+13)
62097 C3 81 F3     jp RET1
org 49100 62100
*******************************
Svuotamento schermo

62100 F3           di
62101 CD 31 F7     call CLS
62104 FB           ei
62105 C9           ret
*******************************
Plot x,y
(0,0=angolo superiore sinistro)

62106 F3           di
62107 CD 72 F7     call PLOT
62110 FB           ei
62111 C9           ret
*******************************
Circle
```

```
62112 F3          di
62113 CD 2B F9    call CIRCLE
62116 FB          ei
62117 C9          ret
********************************
Draw


62118 F3          di
62119 CD D2 FA    call DRAW
62122 FB          ei
62123 C9          ret
********************************
Man?

62124 F3          di
62125 CD 71 FB    call MAN?
62128 FB          ei
62129 C9          ret
********************************
Move?

62130 F3          di
62131 CD 89 FB    call MOVE?
62134 FB          ei
62135 C9          ret
********************************
Beep 1


62136 F3          di
62137 CD DA FB    call BP1
62140 FB          ei
62141 C9          ret
********************************
Beep 2

62142 F3          di
62143 CD F2 FB    call BP2
62146 FB          ei
62147 C9          ret
********************************
Beep 3

62148 F3          di
62149 CD 12 FC    call BP3
62152 FB          ei
62153 C9          ret
********************************
Beep 4
```

```
62154 F3          di
62155 CD 2D FC    call BP4
62158 FB          ei
62159 C9          ret
********************************
Scambio attributi

62160 F3          di
62161 CD 51 FC    call SWAP
62164 FB          ei
62165 C9          ret
********************************
ROLL di una finestra
VERSO L'ALTO

62166 F3          di
62167 CD 66 FC    call WUR
62170 FB          ei
62171 C9          ret
********************************

ROLL di una finestra
VERSO IL BASSO

62172 F3          di
62173 CD BC FC    call WDR
62176 FB          ei
62177 C9          ret
********************************
ROLL di una finestra
A DESTRA

62178 F3          di
62179 CD FD FC    call WRR
62182 FB          ei
62183 C9          ret
********************************
ROLL di una finestra
A SINISTRA

62184 F3          di
62185 CD 23 FD    call WLR
62188 FB          ei
62189 C9          ret
********************************
SCROLL di una finestra
VERSO L'ALTO

62190 F3          di
62191 CD 4F FD    call WUS
```

```
62194 FB          ei
62195 C9          ret
*******************************
SCROLL di una finestra
VERSO IL BASSO

62196 F3          di
62197 CD 5D FD    call WDS
62200 FB          ei
62201 C9          ret
*******************************
SCROLL di una finestra
A DESTRA


62202 F3          di
62203 CD 6B FD    call WRS
62206 FB          ei
62207 C9          ret
*******************************
SCROLL di una finestra
A SINISTRA

62208 F3          di
62209 CD 79 FD    call WLS
62212 FB          ei
62213 C9          ret
*******************************
Visualizzazione stringa

62214 F3          di
62215 CD FD FD    call PBST
62218 FB          ei
62219 C9          ret
*******************************
Screen$
(Tutti i caratteri)

62220 F3          di
62221 CD 41 FE    call SCR1
62224 18 73       jr RET
*******************************
Screen$
(Caratteri GOLDMINE + UDG)

62226 F3          di
62227 CD 5A FE    call SCR2
62230 18 6D       jr RET
*******************************
```

```
Screen$
(Solo UDG)

62232 F3          di
62233 CD 72 FE    call SCR3
62236 18 67       jr RET

*******************************
Attributo x,y

62238 F3          di
62239 CD 9F FE    call Att
62242 18 61       jr RET
*******************************
Scansione tastiera per
soli tasti numerici

62244 F3          di
62245 FD 21 A6 F3 ld iy,62374
62249 CD AB FE    call NUMB
62252 18 53       jr RET1
*******************************
Scansione tastiera per
soli tasti alfabetici MAIUSCOLI

62254 F3          di
62255 FD 21 A6 F3 ld iy,62374
62259 CD BA FE    call LETT
62262 18 49       jr RET1
*******************************
Scansione tastiera

62264 F3          di
62265 FD 21 A6 F3 ld iy,62374
62269 CD 43 F6    call KEY
62272 18 3F       jr RET1
*******************************
Scansione linea

62274 F3          di
62275 FD 21 A6 F3 ld iy,62374
62279 CD 01 F7    call LSCAN
62282 18 35       jr RET1
*******************************
Visualizzazione sprite

62284 F3          di
62285 FD 21 A6 F3 ld iy,62374
62289 CD 9C F7    call SPRITE
```

```
62292 FD 7E 0D     ld a,(iy+13)
62295 18 28        jr RET1
*******************************
Point x,y
(0,0=angolo superiore sinistro)

62297 F3           di
62298 CD E4 F8     call POINT
62301 18 22        jr RET1
*******************************
Movimento

62303 F3           di
62304 FD 21 A6 F3  ld iy,62374
62308 CD 39 FB     call MOVE
62311 18 18        jr RET1
*******************************
Conteggio verso l'ALTO

62313 F3           di
62314 CD 63 F8     call C/UP

62317 FB           ei
62318 C9           ret
*******************************
Conteggio verso il BASSO

62319 F3           di
62320 CD 93 F8     call C/DN
62323 FB           ei
62324 C9           ret
*******************************
Controllo raggiungimento del
valore MAX (9 999 999)

62325 F3           di
62326 CD BC F8     call Chup9
62329 18 0A        jr RET
*******************************
Controllo raggiungimento del
valore MIN (0 000 000)

62331 F3           di
62332 CD CA F8     call Chdn0
62335 18 04        jr RET
RET1
62337 FD 21 3A 5C  ld iy,23610
RET
62341 06 00        ld b,0
62343 4F           ld c,a
```

```
62344 FB            ei
62345 C9            ret
org 49357 62357
62357 00            nop
BUF3
defb 48 48 48 48 0 0 0
62365 00            nop
BUF4
defb 57 57 57 57 0 0 0
62373 00            nop
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
VARIABILI DI SISTEMA

org 49374 62374

*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
KEYV
62374 00            nop
COL
62375 00            nop
LINE
62376 00            nop
FLAG
62377 00            nop
ATTR
defb 56
SCREEN
62379 00            nop
62380 00            nop
KNO
62381 00            nop
FLAG1
62382 00            nop
FLAG2
62383 00            nop
XY
62384 00            nop
Y
62385 00            nop
POIV
62386 00            nop
HITVAR
62387 00            nop
MAN1
62388 00            nop
62389 00            nop
RND
62390 00            nop
62391 00            nop
SEED
```

```
defb 1
62393 00          nop
RAD
62394 00          nop
X2
62395 00          nop
62396 00          nop
Y2
62397 00          nop
62398 00          nop
XM
62399 00          nop
YM
62400 00          nop
R2
62401 00          nop
62402 00          nop
Cc
62403 00          nop
62404 00          nop
HATT
62405 00          nop
BATT
62406 00          nop
SPRAT
62407 00          nop
MADD
62408 00          nop
62409 00          nop
CO1
62410 00          nop
62411 00          nop
CO2
62412 00          nop
62413 00          nop
FORMAT
62414 00          nop
62415 00          nop
MBYTE
62416 00          nop
62417 00          nop
BC
defb 20
defb 5
HL
defb 30
62421 00          nop
DE
defb 5
62423 00          nop
```

```
BDRCL
62424 00          nop
OLD
62425 00          nop
NEW
62426 00          nop
Ca
defb 1
Ya
defb 8
Cb
defb 3
Yb
defb 48
STDATA
62431 00          nop
62432 00          nop
SCR$
62433 00          nop
62434 00          nop
CHR?
62435 00          nop
ATL/C
62436 00          nop
62437 00          nop
ATT?
62438 00          nop
CHAROM
defb 27
defb 243
CHARS
defb 0
defb 60
UDGS
defb 88
defb 255
CLUP
62445 00          nop
62446 00          nop
CLDN
62447 00          nop
62448 00          nop
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
INIZIO ROM/RAM

org 49451 62451
KDATA
defb 66 78 77 0 32 72 74 75 76 0
defb 89 85 73 79 80 54 55 56 57
defb 48 53 52 51 50 49 84 82 69
```

```
defb 87 81 71 70 68 83 65 86 67
defb 88 90 0
CHR
defb 0 0 0 0 0 0 0 0
defb 170 85 170 85 170 85 170 85
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 32 32 64
defb 0 0 0 0 62 0 0 0
defb 0 0 0 0 0 96 96 96
defb 0 1 2 4 8 16 32 64
NUMERI
defb 0 62 99 103 107 115 99 62
defb 0 56 88 24 24 24 24 126
defb 0 62 99 3 62 96 96 127
defb 0 62 99 3 30 3 99 62
defb 0 96 96 108 108 127 12 12
defb 0 127 96 96 126 3 99 62
defb 0 62 99 96 126 99 99 62
defb 0 127 3 3 2 4 8 16
defb 0 62 99 99 62 99 99 62
defb 0 62 99 99 63 3 99 62
SU
defb 0 24 60 126 24 24 24 24
GIU'
defb 0 24 24 24 24 126 60 24
<
defb 0 16 48 127 127 48 16 0

_
defb 0 255 255 0 0 0 0 0
>
defb 0 4 6 127 127 6 4 0
defb 0 62 99 3 30 24 0 24
defb 0 0 0 0 0 0 0 0
LETTERE
defb 0 62 99 99 127 99 99 99
defb 0 126 99 99 126 99 99 126
defb 0 62 99 96 96 96 99 62
defb 0 126 99 99 99 99 99 126
defb 0 127 96 96 124 96 96 127
defb 0 127 96 96 124 96 96 96
defb 0 62 99 96 111 99 99 63
```

```
defb 0 99 99 99 127 99 99 99
defb 0 126 24 24 24 24 24 126
defb 0 15 3 3 3 3 99 62
defb 0 98 100 104 120 100 98 97
defb 0 96 96 96 96 96 96 127
defb 0 99 119 119 107 99 99 99
defb 0 99 115 115 107 103 103 99
defb 0 62 99 99 99 99 99 62
defb 0 126 99 99 126 96 96 96
defb 0 62 99 99 115 107 103 62
defb 0 126 99 99 126 100 98 97
defb 0 62 99 96 62 3 99 62
defb 0 126 24 24 24 24 24 24
defb 0 99 99 99 99 99 99 62
defb 0 99 99 99 99 34 20 8
defb 0 99 99 99 107 107 119 34
defb 0 65 34 20 8 20 34 65
defb 0 129 66 36 24 24 24 24
defb 0 127 3 4 8 16 96 127
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
defb 0 0 0 0 0 0 0 0
INIZIO ROM
################################
KEY
63043 CD 51 F6     call SCAN
63046 11 F3 F3     ld de,KDATA
63049 26 00        ld h,0
63051 19           add hl,de
63052 7E           ld a,(hl)
63053 32 A6 F3     ld (KEYV),a
63056 C9           ret
################################
SCAN
63057 11 28 00     ld de,$0028
63060 6B           ld l,e
63061 01 FE FE     ld bc,$fefe
L3
63064 ED 78        in a,(c)
63066 2F           cpl
63067 E6 1F        and 31
63069 28 08        jr z,L1
63071 67           ld h,a
```

```
63072 6B          ld l,e
L2
63073 2D          dec l
63074 CB 3C       srl h
63076 30 FB       jr nc,L2
63078 14          inc d
L1
63079 7B          ld a,e
63080 D6 05       sub 5
63082 5F          ld e,a
63083 CB 00       rlc b
63085 38 E9       jr c,L3
63087 FD 72 07    ld (iy+7),d
63090 C9          ret
################################
PRINT
63091 FE 16       cp 22
63093 20 0A       jr nz,L4
63095 23          inc hl
63096 11 A7 F3    ld de,COL
63099 01 02 00    ld bc,2
63102 ED B0       ldir
63104 7E          ld a,(hl)
L4
63105 E5          push hl
63106 F5          push af
63107 11 00 40    ld de,16384
63110 2A A7 F3    ld hl,(COL)
63113 19          add hl,de
63114 7C          ld a,h
63115 E6 07       and 7
63117 0F          rrca
63118 0F          rrca
63119 0F          rrca
63120 B5          or l
63121 6F          ld l,a
63122 3E F8       ld a,248
63124 A4          and h
63125 67          ld h,a
63126 F1          pop af
63127 E5          push hl
63128 22 AB F3    ld (SCREEN),h
63131 CB 7F       bit 7,a
63133 28 08       jr z,NGo
63135 ED 5B EB F3 ld de,(UDGS)
63139 D6 90       sub 144
63141 18 04       jr Go
NGo
63143 ED 5B E7 F3 ld de,(CHAROM
Go
```

```
63147 6F          ld l,a
63148 26 00       ld h,0
63150 29          add hl,hl
63151 29          add hl,hl
63152 29          add hl,hl
63153 19          add hl,de
63154 D1          pop de
63155 3A A9 F3    ld a,(FLAG)
63158 06 FF       ld b,255
63160 1F          rra
63161 38 01       jr c,L5
63163 04          inc b
L5
63164 1F          rra
63165 9F          sbc a,a
63166 4F          ld c,a
63167 3E 08       ld a,8
L6
63169 F5          push af
63170 1A          ld a,(de)
63171 A0          and b
63172 AE          xor (hl)
63173 A9          xor c
63174 12          ld (de),a
63175 23          inc hl
63176 14          inc d
63177 F1          pop af
63178 3D          dec a
63179 20 F4       jr nz,L6
63181 2A AB F3    ld hl,(SCREEN
63184 CD EE F6    call FIND2
63187 3A AA F3    ld a,(ATTR)
63190 77          ld (hl),a
63191 2A A7 F3    ld hl,(COL)
63194 2C          inc l
63195 CB 6D       bit 5,l
63197 28 0A       jr z,L7
63199 CB AD       res 5,l
63201 24          inc h
63202 3E 18       ld a,24
63204 BC          cp h
63205 20 02       jr nz,L7
63207 26 00       ld h,0
L7
63209 22 A7 F3    ld (COL),hl
63212 E1          pop hl
63213 C9          ret
################################
FIND2
63214 7C          ld a,h
```

```
63215 0F          rrca
63216 0F          rrca
63217 0F          rrca
63218 E6 03       and 3
63220 F6 58       or 88
63222 67          ld h,a
63223 C9          ret
################################
PSTRING
L8
63224 7E          ld a,(hl)
63225 A7          and a
63226 C8          ret z
63227 CD 73 F6    call PRINT
63230 23          inc hl
63231 18 F7       jr L8
################################
LSCAN
63233 1E 00       ld e,0
63235 01 FE FE    ld bc,$fefe
L9
63238 ED 78       in a,(c)
63240 2F          cpl
63241 E6 1F       and 31
63243 28 01       jr z,L10
63245 37          scf
L10
63246 CB 13       rl e
63248 CB 00       rlc b
63250 38 F2       jr c,L9
63252 FD 73 08    ld (iy+8),e
63255 7B          ld a,e
63256 07          rlca
63257 CB 7F       bit 7,a
63259 28 02       jr z,L11
63261 CB D7       set 2,a
L11
63263 CB 77       bit 6,a
63265 28 02       jr z,L12
63267 CB DF       set 3,a
L12
63269 CB 6F       bit 5,a
63271 28 02       jr z,L13

63273 CB E7       set 4,a
L13
63275 E6 1F       and 31
63277 FD 77 09    ld (iy+9),a
63280 C9          ret
################################
```

```
CLS
63281 21 00 40     ld hl,16384
63284 11 01 40     ld de,16385
63287 01 00 18     ld bc,6144
63290 36 00        ld (hl),0
63292 ED B0        ldir
63294 01 00 03     ld bc,768
63297 3A AA F3     ld a,(ATTR)
63300 77           ld (hl),a
63301 ED B0        ldir
63303 21 00 00     ld hl,0
63306 22 A7 F3     ld (COL),hl
63309 C9           ret
################################
PRSPC
63310 3E 20        ld a,32
63312 C5           push bc
63313 CD 73 F6     call PRINT
63316 C1           pop bc
63317 78           ld a,b
63318 B1           or c
63319 20 F5        jr nz,PRSPC
63321 C9           ret
################################
Trova gli indirizzi di schermo
FIND1
63322 79           ld a,c
63323 07           rlca
63324 07           rlca
63325 07           rlca
63326 A8           xor b
63327 E6 C7        and $c7
63329 A8           xor b
63330 07           rlca
63331 07           rlca
63332 6F           ld l,a
63333 78           ld a,b
63334 A7           and a
63335 1F           rra
63336 37           scf
63337 1F           rra
63338 A7           and a
63339 1F           rra
63340 A8           xor b
63341 E6 F8        and $f8
63343 A8           xor b
63344 67           ld h,a
63345 C9           ret
hl=address
################################
```

```
PLOT
Predisporre variabile XY
(0 0=angolo superiore sinistro)
63346 ED 4B B0 F3 ld bc,(XY)
PLOT1
63350 78          ld a,b
63351 D6 C0       sub 192
63353 D0          ret nc
63354 CD 5A F7    call FIND1
63357 79          ld a,c
63358 E6 07       and 7
63360 47          ld b,a
63361 04          inc b
63362 3E FE       ld a,$fe
P1
63364 0F          rrca
63365 10 FD       djnz P1
63367 47          ld b,a
63368 7E          ld a,(hl)
63369 F5          push af
63370 3A A9 F3    ld a,(FLAG)
63373 4F          ld c,a
63374 F1          pop af
63375 CB 41       bit 0,c
63377 20 01       jr nz,P2
63379 A0          and b
P2
63380 CB 49       bit 1,c
63382 20 02       jr nz,P3
63384 A8          xor b
63385 2F          cpl
P3
63386 77          ld (hl),a
63387 C9          ret
################################
Visualizzazione sprite
SPRITE
63388 2A CC F3    ld hl,(CO2)
63391 ED 4B CE F3 ld bc,(FORMAT
63395 ED 5B B4 F3 ld de,(MAN1)
63399 C5          push bc
63400 44          ld b,h
63401 4D          ld c,l
63402 CD 5A F7    call FIND1
63405 C1          pop bc
63406 CB 20       sla b
63408 CB 20       sla b
63410 CB 20       sla b
S6
63412 C5          push bc
```

```
63413 E5          push hl
S1
63414 1A          ld a,(de)
63415 FD CB 0D 46 bit 0,(iy+13)
63419 28 15       jr z,S7
63421 BE          cp (hl)
63422 28 13       jr z,S8
63424 A6          and (hl)
63425 28 10       jr z,S8
63427 E5          push hl
63428 CD EE F6    call FIND2
63431 3A C5 F3    ld a,(HATT)
63434 BE          cp (hl)
63435 E1          pop hl
63436 20 05       jr nz,S8
63438 37          scf
63439 E1          pop hl
63440 C1          pop bc
63441 C9          ret
S7
63442 77          ld (hl),a
S8
63443 2C          inc l
63444 7D          ld a,l
63445 E6 1F       and 31
63447 20 04       jr nz,S5
63449 7D          ld a,l
63450 D6 20       sub 32
63452 6F          ld l,a
S5
63453 13          inc de
63454 0D          dec c
63455 20 D5       jr nz,S1
63457 E1          pop hl
63458 C1          pop bc
63459 CD F4 F7    call FIND3
63462 10 CC       djnz S6
63464 FD CB 0D 86 res 0,(iy+13)
63468 A7          and a
63469 C9          ret
63470 00          nop
63471 00          nop
63472 00          nop
63473 00          nop
63474 00          nop
63475 00          nop
################################
Trova la linea sottostante
FIND3
63476 24          inc h
```

```
63477 7C          ld a,h
63478 E6 07       and 7
63480 C0          ret nz
63481 7D          ld a,l
63482 C6 20       add a,32
63484 6F          ld l,a
63485 3F          ccf
63486 9F          sbc a,a
63487 E6 F8       and 248
63489 84          add a,h
63490 67          ld h,a
63491 FE 58       cp 88
63493 C0          ret nz
63494 26 40       ld h,64
63496 C9          ret
63497 00          nop
63498 00          nop
63499 00          nop
63500 00          nop
################################
GENERATORE DI SUONI
BEEP
63501 7D          ld a,l
63502 CB 3D       srl l
63504 CB 3D       srl l
63506 2F          cpl
63507 E6 03       and 3
63509 4F          ld c,a
63510 06 00       ld b,0
63512 DD 21 28 F8 ld ix,B1
63516 DD 09       add ix,bc
63518 3A D8 F3    ld a,(BDRCL)
63521 E6 38       and $38
63523 0F          rrca
63524 0F          rrca
63525 0F          rrca
63526 F6 08       or 8
B1
63528 00          nop
63529 00          nop
63530 00          nop
63531 04          inc b
63532 0C          inc c
B2
63533 0D          dec c
63534 20 FD       jr nz,B2
63536 0E 3F       ld c,$3f
63538 05          dec b
63539 20 F8       jr nz,B2
63541 EE 10       xor 16
```

```
63543 D3 FE        out (254),a
63545 44           ld b,h
63546 4F           ld c,a
63547 CB 67        bit 4,a
63549 20 08        jr nz,B3
63551 7A           ld a,d
63552 B3           or e
63553 C8           ret z
63554 79           ld a,c
63555 4D           ld c,l
63556 1B           dec de
63557 DD E9        jp (ix)
B3
63559 4D           ld c,l
63560 0C           inc c
63561 DD E9        jp (ix)
DO
defw 1643
DO#
defw 1549
RE
defw 1461
RE#
defw 1377
MI

defw 1298
FA
defw 1223
FA#
defw 1152
SOL
defw 1086
SOL#
defw 1023
LA
defw 964
LA#
defw 909
SI
defw 856
################################
C/UP
63587 21 96 F3     ld hl,BUF3
63590 E5           push hl
Cu1
63591 7E           ld a,(hl)
63592 CD 8C F8     call Numb?
63595 23           inc hl
63596 30 F9        jr nc,Cu1
```

```
63598 2B          dec hl
Cu2
63599 2B          dec hl
63600 7E          ld a,(hl)
63601 CD 8C F8    call Numb?
63604 38 0A       jr c,Cend
63606 3C          inc a
63607 FE 3A       cp 58
63609 20 04       jr nz,Cu3
63611 36 30       ld (hl),48
63613 18 F0       jr Cu2
Cu3
63615 77          ld (hl),a
Cend
63616 2A ED F3    ld hl,(CLUP)
63619 22 A7 F3    ld (COL),hl
63622 E1          pop hl
63623 CD F8 F6    call PSTRING
63626 C9          ret
63627 00          nop
################################
Numb?
63628 FE 30       cp 48
63630 D8          ret c
63631 FE 3A       cp 58
63633 3F          ccf
63634 C9          ret
################################
C/DN
63635 21 9E F3    ld hl,BUF4
63638 E5          push hl
Cd1
63639 7E          ld a,(hl)
63640 CD 8C F8    call Numb?
63643 23          inc hl
63644 30 F9       jr nc,Cd1
63646 2B          dec hl
Cd2
63647 2B          dec hl
63648 7E          ld a,(hl)
63649 CD 8C F8    call Numb?
63652 38 0A       jr c,Cdend
63654 3D          dec a
63655 FE 2F       cp 47
63657 20 04       jr nz,Cd3
63659 36 39       ld (hl),57
63661 18 F0       jr Cd2
Cd3
63663 77          ld (hl),a
Cdend
```

```
63664 2A EF F3     ld hl,(CLDN)
63667 22 A7 F3     ld (COL),hl
63670 E1           pop hl
63671 CD F8 F6     call PSTRING
63674 C9           ret
63675 00           nop
################################
Chup9
63676 21 96 F3     ld hl,BUF3
Ch1
63679 7E           ld a,(hl)
63680 CD 8C F8     call Numb?
63683 D8           ret c
63684 FE 39        cp 57
63686 23           inc hl
63687 28 F6        jr z,Ch1
63689 C9           ret
################################
ChdnO
63690 21 9E F3     ld hl,BUF4
Ch2
63693 7E           ld a,(hl)
63694 CD 8C F8     call Numb?
63697 D8           ret c
63698 FE 30        cp 48
63700 23           inc hl
63701 28 F6        jr z,Ch2
63703 C9           ret
################################
BREAK
63704 3E 7F        ld a,$7f
63706 DB FE        in a,($fe)
63708 1F           rra
63709 D8           ret c
63710 3E FE        ld a,$fe
63712 DB FE        in a,($fe)
63714 1F           rra
63715 C9           ret
################################
POINT
63716 ED 4B B0 F3 ld bc,(XY)
63720 CD 5A F7     call FIND1
63723 79           ld a,c
63724 E6 07        and 7
63726 47           ld b,a
63727 04           inc b
63728 7E           ld a,(hl)
LP
63729 07           rlca
63730 10 FD        djnz LP
```

```
63732 E6 01       and 1
63734 32 B2 F3    ld (POIV),a
63737 C9          ret
################################
RAND
63738 2A B8 F3    ld hl,(SEED)
63741 54          ld d,h
63742 7E          ld a,(hl)
63743 5F          ld e,a
63744 1A          ld a,(de)
63745 67          ld h,a
63746 ED 6A       adc hl,hl
63748 ED 5A       adc hl,de
63750 19          add hl,de
63751 ED 6A       adc hl,hl
63753 29          add hl,hl
63754 ED 6A       adc hl,hl
63756 19          add hl,de
63757 ED 6A       adc hl,hl
63759 22 B8 F3    ld (SEED),hl
63762 ED 5B B6 F3 ld de,(RND)
63766 AF          xor a
63767 BA          cp d
63768 28 08       jr z,LP1
LP2
63770 ED 52       sbc hl,de
63772 30 FC       jr nc,LP2
63774 19          add hl,de
63775 E5          push hl
63776 C1          pop bc
63777 C9          ret
LP1
63778 7D          ld a,l
LP3
63779 93          sub e
63780 30 FD       jr nc,LP3
63782 83          add a,e
63783 06 00       ld b,0
63785 4F          ld c,a
63786 C9          ret
################################
CIRCLE
63787 3A BA F3    ld a,(RAD)
63790 A7          and a
63791 C8          ret z
63792 FE 01       cp 1
63794 20 04       jr nz,LP4
63796 CD 72 F7    call PLOT
63799 C9          ret
LP4
```

```
63800 47          ld b,a
63801 3E 64       ld a,100
63803 90          sub b
63804 D8          ret c
63805 21 08 FA    ld hl,SQU
63808 22 BD F3    ld (Y2),hl
63811 05          dec b
63812 78          ld a,b
LP6
63813 23          inc hl
63814 23          inc hl
63815 10 FC       djnz LP6
63817 22 BB F3    ld (X2).hl
63820 23          inc hl
63821 23          inc hl
63822 01 02 00    ld bc,2
63825 11 C1 F3    ld de,R2
63828 ED B0       ldir
63830 32 BF F3    ld (XM),a
63833 AF          xor a
63834 32 C0 F3    ld (YM),a
LP7
63837 2A B0 F3    ld hl,(XY)
63840 ED 5B BF F3 ld de,(XM)
63844 7C          ld a,h
63845 82          add a,d
63846 38 34       jr c,SK5
63848 47          ld b,a
63849 7D          ld a,l
63850 83          add a,e
63851 38 06       jr c,SK2
63853 4F          ld c,a
63854 E5          push hl
63855 CD 76 F7    call PLOT1
63858 E1          pop hl
SK2
63859 7C          ld a,h
63860 83          add a,e
63861 38 18       jr c,SK4
63863 47          ld b,a
63864 7D          ld a,l
63865 82          add a,d
63866 38 06       jr c,SK3
63868 4F          ld c,a
63869 E5          push hl
63870 CD 76 F7    call PLOT1
63873 E1          pop hl
SK3
63874 7C          ld a,h
63875 83          add a,e
```

```
63876 47          ld b,a
63877 7D          ld a,l
63878 92          sub d
63879 38 31       jr c,SK7
63881 4F          ld c,a
63882 E5          push hl
63883 CD 76 F7    call PLOT1
63886 E1          pop hl
SK4
63887 7C          ld a,h
63888 82          add a,d
63889 47          ld b,a
63890 7D          ld a,l
63891 93          sub e
63892 38 15       jr c,SK6
63894 4F          ld c,a
63895 E5          push hl
63896 CD 76 F7    call PLOT1
63899 E1          pop hl
SK5
63900 7C          ld a,h
63901 92          sub d
63902 38 36       jr c,SK9
63904 47          ld b,a
63905 7D          ld a,l
63906 93          sub e
63907 38 06       jr c,SK6
63909 4F          ld c,a
63910 E5          push hl
63911 CD 76 F7    call PLOT1
63914 E1          pop hl
SK6
63915 7C          ld a,h
63916 93          sub e
63917 38 1A       jr c,SK8
63919 47          ld b,a
63920 7D          ld a,l
63921 92          sub d
63922 38 06       jr c,SK7
63924 4F          ld c,a
63925 E5          push hl
63926 CD 76 F7    call PLOT1
63929 E1          pop hl
SK7
63930 7C          ld a,h
63931 93          sub e
63932 38 0B       jr c,SK8
63934 47          ld b,a
63935 7D          ld a,l
63936 82          add a,d
```

```
63937 38 13        jr c,SK9
63939 4F           ld c,a
63940 E5           push hl
63941 CD 76 F7     call PLOT1
63944 E1           pop hl
SK8
63945 7C           ld a,h
63946 92           sub d
63947 47           ld b,a
63948 7D           ld a,l
63949 83           add a,e
63950 38 06        jr c,SK9
63952 4F           ld c,a
63953 E5           push hl
63954 CD 76 F7     call PLOT1
63957 E1           pop hl
SK9
63958 21 C0 F3     ld hl,YM
63961 34           inc (hl)
63962 7E           ld a,(hl)
63963 2B           dec hl
63964 96           sub (hl)
63965 D0           ret nc
63966 2A BD F3     ld hl,(Y2)
63969 23           inc hl
63970 23           inc hl
63971 22 BD F3     ld (Y2),hl
63974 5E           ld e,(hl)
63975 23           inc hl
63976 56           ld d,(hl)
63977 2A BB F3     ld hl,(X2)
63980 4E           ld c,(hl)
63981 23           inc hl
63982 46           ld b,(hl)
63983 2A C1 F3     ld hl,(R2)
63986 EB           ex de,hl
63987 09           add hl,bc
63988 ED 52        sbc hl,de
63990 DA 5D F9     jp c,LP7
63993 21 BF F3     ld hl,XM
63996 35           dec (hl)
63997 2A BB F3     ld hl,(X2)
64000 2B           dec hl
64001 2B           dec hl
64002 22 BB F3     ld (X2),hl
64005 C3 5D F9     jp LP7
SQU
defb 0 0 1 0 4 0 9 0 16 0 25 0
defb 36 0
defb 49 0 64 0 81 0 100 0 132 0
```

```
defb 144 0 169 0 196 0 225 0 0 1
defb 33 1
defb 68 1 105 1 144 1 185 1
defb 228 1
defb 17 2 64 2 113 2 164 2 217 2
defb 16 3 73 3 132 3 193 3 0 4
defb 65 4
defb 132 4 201 4 16 5 89 5 164 5
defb 241 5 64 6 145 6 228 6 57 7
defb 144 7 233 7 68 8 161 8 0 9
defb 97 9 196 9 41 10 144 10
defb 249 10 100 11 209 11 64 12
defb 177 12 36 13 153 13 16 14
defb 137 14 4 15 129 15 0 16
defb 129 16 4 17 137 17
defb 16 18
defb 153 18 36 19 177 19 28 20
defb 209 20 100 21 249 21 144 22
defb 41 23 160 23 97 24 0 25
defb 161 25
defb 68 26 233 26 144 27 57 28
defb 228 28 145 29 64 30 241 30
defb 164 31 89 32 16 33 201 33
defb 132 34 65 35 0 36 193 36
defb 132 37 73 38 16 39
################################
DRAW
64210 D9          exx
64211 E5          push hl
64212 D9          exx
64213 2A B0 F3    ld hl,(XY)
64216 ED 4B BF F3 ld bc,(XM)
64220 78          ld a,b
64221 94          sub h
64222 16 01       ld d,1
64224 30 04       jr nc,DR1
64226 16 FF       ld d,255
64228 7C          ld a,h
64229 90          sub b
DR1
64230 47          ld b,a
64231 79          ld a,c
64232 95          sub l
64233 1E 01       ld e,1
64235 30 04       jr nc,DR2
64237 1E FF       ld e,255
64239 7D          ld a,l
64240 91          sub c
DR2
64241 4F          ld c,a
```

```
64242 B8          cp b
64243 30 06       jr nc,DR3
64245 69          ld l,c
64246 D5          push de
64247 AF          xor a
64248 5F          ld e,a
64249 18 11       jr DR4
DR3
64251 B1          or c
64252 20 09       jr nz,DR5
64254 ED 4B B0 F3 ld bc,(XY)
64258 CD 76 F7    call PLOT1
64261 18 2E       jr DR10
DR5
64263 68          ld l,b
64264 41          ld b,c
64265 D5          push de
64266 16 00       ld d,0
DR4
64268 60          ld h,b
64269 78          ld a,b
64270 1F          rra
DR9
64271 85          add a,l
64272 38 03       jr c,DR6
64274 BC          cp h
64275 38 07       jr c,DR7
DR6
64277 94          sub h
64278 4F          ld c,a
64279 D9          exx
64280 C1          pop bc
64281 C5          push bc
64282 18 04       jr DR8
DR7
64284 4F          ld c,a
64285 D5          push de
64286 D9          exx
64287 C1          pop bc
DR8
64288 2A B0 F3    ld hl,(XY)
64291 78          ld a,b
64292 84          add a,h
64293 47          ld b,a
64294 79          ld a,c
64295 85          add a,l
64296 4F          ld c,a
64297 ED 43 B0 F3 ld (XY),bc
64301 CD 76 F7    call PLOT1
64304 D9          exx
```

```
64305 79          ld a,c
64306 10 DB       djnz DR9
64308 D1          pop de
DR10
64309 D9          exx
64310 E1          pop hl
64311 D9          exx
64312 C9          ret
################################
MOVE
64313 2A CA F3    ld hl,(CO1)
64316 E5          push hl
64317 CD 01 F7    call LSCAN
64320 E1          pop hl
64321 3A AF F3    ld a,(FLAG2)
64324 E6 0F       and 15
64326 28 25       jr z,J16
64328 F5          push af
64329 E6 03       and 3
64331 28 07       jr z,J4
64333 CB 47       bit 0,a
64335 28 02       jr z,J5
64337 2D          dec l
64338 2D          dec l
J5
64339 2C          inc l
J4
64340 F1          pop af
64341 E6 0C       and 12
64343 28 14       jr z,J16
64345 CB 57       bit 2,a
64347 28 02       jr z,J7
64349 24          inc h
64350 24          inc h
J7
64351 25          dec h
64352 7C          ld a,h
64353 FE C0       cp 192
64355 20 02       jr nz,J8
64357 26 00       ld h,0
J8
64359 FE FF       cp 255
64361 20 02       jr nz,J16
64363 26 BF       ld h,191
J16
64365 22 CC F3    ld (CO2),hl
64368 C9          ret
################################
MAN?
64369 2A CC F3    ld hl,(CO2)
```

```
64372 ED 5B D0 F3 ld de,(MBYTE)
64376 7D          ld a,l
64377 E6 06       and 6
64379 1F          rra
64380 2A C8 F3    ld hl,(MADD)
64383 28 04       jr z,J9
J10
64385 19          add hl,de
64386 3D          dec a
64387 20 FC       jr nz,J10
J9
64389 22 B4 F3    ld (MAN1),hl
64392 C9          ret
################################
MOVE?
64393 2A CC F3    ld hl,(CO2)
64396 ED 4B CE F3 ld bc,(FORMAT
64400 7C          ld a,h
64401 E6 07       and 7
64403 28 01       jr z,M4
64405 04          inc b
M4
64406 C5          push bc
64407 E5          push hl
64408 C1          pop bc
64409 CD 5A F7    call FIND1
64412 C1          pop bc
64413 CD EE F6    call FIND2
M5
64416 C5          push bc
64417 E5          push hl
M2
64418 3A C6 F3    ld a,(BATT)
64421 BE          cp (hl)
64422 20 09       jr nz,M6
64424 2A CA F3    ld hl,(CO1)
64427 22 CC F3    ld (CO2),hl
64430 E1          pop hl
64431 C1          pop bc
64432 C9          ret
M6
64433 3A C7 F3    ld a,(SPRAT)
64436 77          ld (hl),a
64437 2C          inc l
64438 7D          ld a,l
64439 E6 1F       and 31
64441 20 04       jr nz,M1
64443 7D          ld a,l
64444 D6 20       sub 32
64446 6F          ld l,a
```

```
M1
64447 0D          dec c
64448 20 E0       jr nz,M2
64450 E1          pop hl
64451 11 20 00    ld de,32
64454 19          add hl,de
64455 7C          ld a,h
64456 FE 5B       cp 91
64458 20 02       jr nz,M3
64460 26 58       ld h,88
M3
64462 C1          pop bc
64463 10 CF       djnz M5
64465 2A CC F3    ld hl,(CO2)
64468 22 CA F3    ld (CO1),hl
64471 C9          ret
64472 00          nop
64473 00          nop
################################
BP1
64474 ED 4B D2 F3 ld bc,(BC)
64478 2A D4 F3    ld hl,(HL)
64481 ED 5B D6 F3 ld de,(DE)
Ba
64485 E5          push hl
64486 C5          push bc
64487 D5          push de
64488 CD 0D F8    call BEEP
64491 D1          pop de
64492 C1          pop bc
64493 E1          pop hl
64494 23          inc hl
64495 10 F4       djnz Ba
64497 C9          ret
################################
BP2
64498 ED 5B D6 F3 ld de,(DE)
64502 3A D8 F3    ld a,(BDRCL)
64505 E6 38       and 56
64507 0F          rrca
64508 0F          rrca
64509 0F          rrca
64510 F6 08       or 8
Bb
64512 EE 10       xor 16
64514 D3 FE       out (254),a
64516 42          ld b,d
Bc
64517 10 FE       djnz Bc
64519 EE 10       xor 16
```

```
64521 D3 FE        out (254),a
64523 42           ld b,d
Bg
64524 10 FE        djnz Bg
64526 1D           dec e
64527 20 EF        jr nz,Bb
64529 C9           ret
################################
BP3
64530 2A D4 F3     ld hl,(HL)
Bd
64533 7E           ld a,(hl)
64534 00           nop
64535 D3 FE        out (254),a
64537 00           nop
64538 00           nop
64539 D3 FE        out (254),a
64541 00           nop
64542 00           nop
64543 D3 FE        out (254),a
64545 00           nop
64546 00           nop
64547 D3 FE        out (254),a
64549 00           nop
64550 23           inc hl
64551 7C           ld a,h
64552 D6 3C        sub 60
64554 38 E9        jr c,Bd
64556 C9           ret
################################
BP4
64557 ED 4B D2 F3 ld bc,(BC)
64561 21 D4 F3     ld hl,HL
64564 3A D8 F3     ld a,(BDRCL)
64567 E6 38        and 56
64569 0F           rrca
64570 0F           rrca
64571 0F           rrca
64572 F6 08        or 8
64574 5F           ld e,a
Be
64575 EE 10        xor 16
64577 D3 FE        out (254),a
64579 41           ld b,c
Bf
64580 10 FE        djnz Bf
64582 EE 10        xor 16
64584 D3 FE        out (254),a
64586 0C           inc c
64587 79           ld a,c
```

```
64588 BE          cp (hl)
64589 7B          ld a,e
64590 20 EF       jr nz,Be
64592 C9          ret
################################
SWAP
64593 11 D9 F3    ld de,OLD
64596 21 00 58    ld hl,22528
Aa
64599 1A          ld a,(de)
64600 BE          cp (hl)
64601 20 04       jr nz,Ab
64603 3A DA F3    ld a,(NEW)
64606 77          ld (hl),a
Ab
64607 23          inc hl
64608 7C          ld a,h
64609 FE 5B       cp 91
64611 20 F2       jr nz,Aa
64613 C9          ret
################################
WUR
64614 ED 4B DB F3 ld bc,(Ca)
64618 CB 21       sla c
64620 CB 21       sla c
64622 CB 21       sla c
64624 CD 5A F7    call FIND1
64627 22 AB F3    ld (SCREEN),h
64630 3A DD F3    ld a,(Cb)
64633 47          ld b,a
Wa
64634 7E          ld a,(hl)
64635 F5          push af
64636 2C          inc l
64637 10 FB       djnz Wa
64639 ED 4B DD F3 ld bc,(Cb)
64643 2A AB F3    ld hl,(SCREEN
64646 05          dec b
Wb
64647 E5          push hl
64648 CD F4 F7    call FIND3
64651 D1          pop de
64652 E5          push hl
64653 C5          push bc
64654 06 00       ld b,0
64656 ED B0       ldir
64658 C1          pop bc
64659 E1          pop hl
64660 10 F1       djnz Wb
64662 79          ld a,c
```

```
64663 85          add a,l
64664 3D          dec a
64665 6F          ld l,a
64666 41          ld b,c
Wc
64667 F1          pop af
64668 77          ld (hl),a
64669 2D          dec l
64670 10 FB       djnz Wc
64672 C9          ret
################################
FIND4
64673 25          dec h
64674 7C          ld a,h
64675 EE 07       xor 7
64677 C0          ret nz
64678 7D          ld a,l
64679 D6 20       sub 32
64681 6F          ld l,a
64682 3F          ccf
64683 9F          sbc a,a
64684 E6 07       and 7
64686 84          add a,h
64687 67          ld h,a
64688 FE 40       cp 64
64690 C0          ret nz
64691 26 58       ld h,88
64693 C9          ret
64694 00          nop
64695 00          nop
64696 00          nop
64697 00          nop
64698 00          nop
64699 00          nop
################################
WDR
64700 ED 4B DB F3 ld bc,(Ca)
64704 3A DE F3    ld a,(Yb)
64707 80          add a,b
64708 3D          dec a
64709 47          ld b,a
64710 CB 21       sla c
64712 CB 21       sla c
64714 CB 21       sla c
64716 CD 5A F7    call FIND1
64719 22 AB F3    ld (SCREEN),h
64722 3A DD F3    ld a,(Cb)
64725 47          ld b,a
Da
64726 7E          ld a,(hl)
```

```
64727 F5          push af
64728 2C          inc l
64729 10 FB       djnz Da
64731 ED 4B DD F3 ld bc,(Cb)
64735 2A AB F3    ld hl,(SCREEN
64738 05          dec b
Db
64739 E5          push hl
64740 CD A1 FC    call FIND4
64743 D1          pop de
64744 E5          push hl
64745 C5          push bc
64746 06 00       ld b,0
64748 ED B0       ldir
64750 C1          pop bc
64751 E1          pop hl
64752 10 F1       djnz Db
64754 79          ld a,c
64755 85          add a,l
64756 3D          dec a
64757 6F          ld l,a
64758 41          ld b,c
Dc
64759 F1          pop af
64760 77          ld (hl),a
64761 2D          dec l
64762 10 FB       djnz Dc
64764 C9          ret
################################
WRR
64765 ED 4B DB F3 ld bc,(Ca)
64769 CB 21       sla c

64771 CB 21       sla c
64773 CB 21       sla c
64775 CD 5A F7    call FIND1
64778 ED 4B DD F3 ld bc,(Cb)
Ra
64782 E5          push hl
64783 C5          push bc
64784 A7          and a
Rb
64785 CB 1E       rr (hl)
64787 2C          inc l
64788 0D          dec c
64789 20 FA       jr nz,Rb
64791 C1          pop bc
64792 E1          pop hl
Rd
64793 30 02       jr nc,Rc
```

```
64795 CB FE        set 7,(hl)
Rc
64797 CD F4 F7     call FIND3
64800 10 EC        djnz Ra
64802 C9           ret
################################
WLR
64803 ED 4B DB F3 ld bc,(Ca)
64807 3A DD F3     ld a,(Cb)
64810 81           add a,c
64811 3D           dec a
64812 4F           ld c,a
64813 CB 21        sla c
64815 CB 21        sla c
64817 CB 21        sla c
64819 CD 5A F7     call FIND1
64822 ED 4B DD F3 ld bc,(Cb)
La
64826 E5           push hl
64827 C5           push bc
64828 A7           and a
Lb
64829 CB 16        rl (hl)
64831 2D           dec l
64832 0D           dec c
64833 20 FA        jr nz,Lb
64835 C1           pop bc
64836 E1           pop hl
Ld
64837 30 02        jr nc,Lc
64839 CB C6        set 0,(hl)
Lc
64841 CD F4 F7     call FIND3
64844 10 EC        djnz La
64846 C9           ret
################################
WUS
64847 3E AF        ld a,175
64849 32 7A FC     ld (Wa),a
64852 CD 66 FC     call WUR
64855 3E 7E        ld a,126
64857 32 7A FC     ld (Wa),a
64860 C9           ret
################################
WDS
64861 3E AF        ld a,175
64863 32 D6 FC     ld (Da),a
64866 CD BC FC     call WDR
64869 3E 7E        ld a,126
64871 32 D6 FC     ld (Da),a
```

```
64874 C9          ret
################################
WRS
64875 3E 18       ld a,24
64877 32 19 FD    ld (Rd),a
64880 CD FD FC    call WRR
64883 3E 30       ld a,48
64885 32 19 FD    ld (Rd),a
64888 C9          ret
################################
WLS
64889 3E 18       ld a,24
64891 32 45 FD    ld (Ld),a
64894 CD 23 FD    call WLR
64897 3E 30       ld a,48
64899 32 45 FD    ld (Ld),a
64902 C9          ret
################################
PRB
64903 F5          push af
64904 ED 4B C3 F3 ld bc,(Cc)
64908 CB 21       sla c
64910 CB 21       sla c
64912 CB 21       sla c
64914 CD 5A F7    call FIND1
64917 F1          pop af
64918 22 AB F3    ld (SCREEN),h
64921 E5          push hl
64922 CB 7F       bit 7,a
64924 28 08       jr z,Pg
64926 ED 5B EB F3 ld de,(UDGS)
64930 D6 90       sub 144
64932 18 04       jr Pf
Pg
64934 ED 5B E7 F3 ld de,(CHAROM
Pf
64938 6F          ld l,a
64939 26 00       ld h,0
64941 29          add hl,hl
64942 29          add hl,hl
64943 29          add hl,hl
64944 19          add hl,de
64945 D1          pop de
64946 3A A9 F3    ld a,(FLAG)
64949 06 FF       ld b,255
64951 1F          rra
64952 38 01       jr c,Pa
64954 04          inc b
Pa
64955 1F          rra
```

```
64956 9F          sbc a,a
64957 4F          ld c,a
64958 3E 08       ld a,8
Fb
64960 F5          push af
64961 1A          ld a,(de)
64962 A0          and b
64963 AE          xor (hl)
64964 A9          xor c
64965 12          ld (de),a
64966 EB          ex de,hl
64967 13          inc de
64968 CD F4 F7    call FIND3
64971 EB          ex de,hl
64972 F1          pop af
64973 3D          dec a
64974 20 F0       jr nz,Fb
64976 EB          ex de,hl
64977 CD A1 FC    call FIND4
64980 CD EE F6    call FIND2
64983 3A AA F3    ld a,(ATTR)
64986 F5          push af
64987 77          ld (hl),a
64988 2A AB F3    ld hl,(SCREEN
64991 CD EE F6    call FIND2
64994 F1          pop af
64995 77          ld (hl),a
64996 2A C3 F3    ld hl,(Cc)
64999 2C          inc l
65000 CB 6D       bit 5,l
65002 28 0B       jr z,Pc
65004 CB AD       res 5,l
65006 7C          ld a,h
65007 C6 08       add a,8
65009 67          ld h,a
65010 D6 C0       sub 192
65012 38 01       jr c,Pc
65014 67          ld h,a
Pc
65015 22 C3 F3    ld (Cc),hl
65018 C9          ret
65019 00          nop
65020 00          nop
################################
PBST
65021 2A DF F3    ld hl,(STDATA
PBST1
Pd
65024 7E          ld a,(hl)
65025 A7          and a
```

```
65026 C8          ret z
65027 E5          push hl
65028 CD 87 FD    call PRB
65031 E1          pop hl
65032 23          inc hl
65033 18 F5       jr Pd
################################
Chr               .
65035 C5          push bc
65036 D5          push de
65037 E5          push hl
65038 1A          ld a,(de)
65039 AE          xor (hl)
65040 28 04       jr z,Sa
65042 3C          inc a
65043 20 12       jr nz,Sb
65045 3D          dec a
Sa
65046 4F          ld c,a
65047 06 07       ld b,7
Sc
65049 14          inc d
65050 23          inc hl
65051 1A          ld a,(de)
65052 AE          xor (hl)
65053 A9          xor c
65054 20 07       jr nz,Sb
65056 10 F7       djnz Sc
65058 C1          pop bc
65059 C1          pop bc
65060 C1          pop bc
65061 AF          xor a
65062 C9          ret
Sb
65063 E1          pop hl
65064 11 08 00    ld de,8
65067 19          add hl,de
65068 D1          pop de
65069 C1          pop bc
65070 10 DB       djnz Chr
65072 37          scf
65073 C9          ret
################################
FIND5
65074 79          ld a,c
65075 0F          rrca
65076 0F          rrca
65077 0F          rrca
65078 E6 E0       and 224
65080 A8          xor b
```

```
65081 5F          ld e,a
65082 79          ld a,c
65083 E6 18       and 24
65085 F6 40       or 64
65087 57          ld d,a
65088 C9          ret
################################
SCR1
65089 ED 4B E1 F3 ld bc,(SCR$)
65093 2A E9 F3    ld hl,(CHARS)
65096 24          inc h
65097 CD 32 FE    call FIND5
65100 06 60       ld b,96

65102 CD 0B FE    call Chr
65105 38 07       jr c,SCR2
65107 3E 80       ld a,128
65109 90          sub b
65110 32 E3 F3    ld (CHR?),a
65113 C9          ret
################################
SCR2
65114 ED 4B E1 F3 ld bc,(SCR$)
65118 21 1B F4    ld hl,CHR
65121 CD 32 FE    call FIND5
65124 06 44       ld b,68
65126 CD 0B FE    call Chr
65129 38 07       jr c,SCR3
65131 3E 64       ld a,100
65133 90          sub b
65134 32 E3 F3    ld (CHR?),a
65137 C9          ret
################################
SCR3
65138 ED 4B E1 F3 ld bc,(SCR$)
65142 2A EB F3    ld hl,(UDGS)
65145 CD 32 FE    call FIND5
65148 06 15       ld b,21
65150 CD 0B FE    call Chr
65153 38 07       jr c,NULL
65155 3E A5       ld a,165
65157 90          sub b
65158 32 E3 F3    ld (CHR?),a
65161 C9          ret
NULL
65162 AF          xor a
65163 32 E3 F3    ld (CHR?),a
65166 C9          ret
################################
FIND6
```

```
65167 79          ld a,c
65168 0F          rrca
65169 0F          rrca
65170 0F          rrca
65171 4F          ld c,a
65172 E6 E0       and 224

65174 A8          xor b
65175 6F          ld l,a
65176 79          ld a,c
65177 E6 03       and 3
65179 F6 58       or 88
65181 67          ld h,a
65182 C9          ret
################################
Att
65183 ED 4B E4 F3 ld bc,(ATL/C)
65187 CD 8F FE    call FIND6
65190 7E          ld a,(hl)
65191 32 E6 F3    ld (ATT?),a
65194 C9          ret
################################
NUMB
65195 CD 43 F6    call KEY
65198 A7          and a
65199 28 FA       jr z,NUMB
65201 FE 30       cp 48
65203 38 F6       jr c,NUMB

65205 FE 3A       cp 58
65207 30 F2       jr nc,NUMB
65209 C9          ret
################################
LETT
65210 CD 43 F6    call KEY
65213 A7          and a
65214 28 FA       jr z,LETT
65216 FE 41       cp 65
65218 38 F6       jr c,LETT
65220 FE 5B       cp 91
65222 D8          ret c
65223 FE 61       cp 97
65225 38 EF       jr c,LETT
65227 FE 7B       cp 123
65229 30 EB       jr nc,LETT
65231 FD CB 00 AE res 5,(iy+0)
65235 C9          ret
```

Appendice

Mappa della memoria del video

C

Sfruttando questa tabella potrete agevolmente trovare le celle di memoria corrispondenti alle varie zone del video.
Esempio: il byte della quinta riga del carattere in linea 13, colonna 27 è dato da

indirizzo base riga 5, linea 13	4BA0+
numero colonna	1B=
indirizzo del byte corrispondente	4BBB

Linea		Linea		Linea	
0	4000 4100 4200 4300 4400 4500 4600 4700	8	4700 4800 4900 4A00 4B00 4C00 4D00 4E00	16	5000 5100 5200 5300 5400 5500 5600 5700
1	4020 4120 4220 4320 4420 4520 4620 4720	9	4720 4820 4920 4A20 4B20 4C20 4D20 4E20	17	5020 5120 5220 5320 5420 5520 5620 5720
2	4040 4140 4240 4340 4440 4540 4640 4740	10	4740 4840 4940 4A40 4B40 4C40 4D40 4E40	18	5040 5140 5240 5340 5440 5540 5640 5740
3	4060 4160 4260 4360 4460 4560 4660 4760	11	4760 4860 4960 4A60 4B60 4C60 4D60 4E60	19	5060 5160 5260 5360 5460 5560 5660 5760
4	4080 4180 4280 4380 4480 4580 4680 4780	12	4780 4880 4980 4A80 4B80 4C80 4D80 4E80	20	5080 5180 5280 5380 5480 5580 5680 5780
5	40AO 41AO 42AO 13AO 44AO 45AO 46AO 47AO	13	47AO 48AO 49AO 4AAO 4BAO 4CAO 4DAO 4EAO	21	50AO 51AO 52AO 53AO 54AO 55AO 56AO 57AO
6	40CO 41CO 42CO 43CO 44CO 45CO 46CO 47CO	14	47CO 48CO 49CO 4ACO 4BCO 4CCO 4DCO 4ECO	22	50CO 51CO 52CO 53CO 54CO 55CO 56CO 57CO
7	40EO 41EO 42EO 43EO 44EO 45EO 46EO 47EO	15	47EO 48EO 49EO 4AEO 4BEO 4CEO 4DEO 4EEO	23	50EO 51EO 52EO 53EO 54EO 55EO 56EO 57EO

Colonna																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F

Appendice

Codici macchina dello Z80

D

Tabella D.1 Codici macchina (mnemonici) dello Z80

(HL) – è il numero contenuto nella locazione il cui indirizzo è puntato da HL
NN – supponendo che sia scritto come N_1N_2, N_2 è inserito nel byte basso (da 0 a 255), seguito da N_1 nel byte alto (da 0 a 255 per 256). Per esempio, il numero decimale 4000 è inserito come 3840+160=160 (N_2), 15 (N_1), poiché 15×256=3840.
Registro A – è l'accumulatore.

Decimale	Byte	Es.	Codifica	Descrizione
0	1	00	nop	Nessuna operazione
1	3	01	ld bc,NN	Carica nella coppia di registri BC il valore NN
2	1	02	ld (bc),a	Immagazzina l'accumulatore nella locazione di memoria indirizzata dalla coppia di registri BC
3	1	03	inc bc	Incrementa di 1 il contenuto della coppia di registri BC
4	1	04	inc b	Incrementa di 1 il contenuto del registro B
5	1	05	dec b	Diminuisce di 1 il contenuto del registro B
6	2	06	ld b,N	Carica nel registro B il valore N
7	1	07	rlca	Rotazione circolare verso sinistra dell'accumulatore
8	1	08	ex af, af	Scambia il contenuto di A e di F con quello di A' e F'
9	1	09	add hl,bc	Somma il contenuto della coppia di registri HL con quello di BC, lasciando il risultato in HL
10	3	0A	ld a,(bc)	Immagazzina nell'accumulatore il contenuto della locazione di memoria indirizzata dalla coppia di registri BC
11	1	0B	dec bc	Diminuisce di 1 il contenuto della coppia di registri BC
12	1	0C	inc c	Incrementa di 1 il contenuto del registro C
13	1	0D	dec c	Diminuisce di 1 il contenuto del registro C
14	2	0E	ld c,N	Carica nel registro C il valore N
15	1	0F	rrca	Rotazione circolare dell'accumulatore verso destra
16	2	10	djnz x	Diminuisce di 1 il contenuto del registro B, e, se il risultato è zero, salta in avanti o indietro di x
17	3	11	ld de,NN	Carica nella coppia di registri DE il valore NN
18	1	12	ld(de),a	Immagazzina l'accumulatore nella locazione di memoria indirizzata dalla coppia di registri DE

Decimale	Byte	Es.	Codifica	Descrizione
19	1	13	inc de	Incrementa di 1 il contenuto della coppia di registri DE
20	1	14	inc d	Incrementa di 1 il contenuto del registro D
21	1	15	dec d	Diminuisce di 1 il contenuto del registro D
22	2	16	ld d,N	Carica nel registro D il valore N
23	1	17	rla	Rotazione circolare verso sinistra dell'accumulatore attraverso il flag di carry (riporto)
24	2	18	jr x	Salta in avanti o indietro di x
25	1	19	add hl,de	Somma il contenuto della coppia di registri HL con quello di DE, lasciando il risultato in HL
26	3	1A	ld a,(de)	Immagazzina nell'accumulatore il contenuto della locazione di memoria indirizzata dalla coppia di registri DE
27	1	1B	dec de	Diminuisce di 1 il contenuto della coppia di registri DE
28	1	1C	inc e	Incrementa di 1 il contenuto del registro E
29	1	1D	dec e	Diminuisce di 1 il contenuto del registro E
30	2	1E	ld e,N	Carica nel registro E il valore N
31	1	1F	rra	Rotazione circolare verso destra dell'accumulatore attraverso il flag di carry
32	2	20	jr nz, x	Salta in avanti o indietro di x, se il flag Z è a zero (cioè, se è non-zero)
33	3	21	ld hl,NN	Carica nella coppia di registri HL il valore NN
34	3	22	ld (NN),hl	Immagazzina il contenuto della coppia di registri HL nella locazione NN
35	1	23	inc hl	Incrementa di 1 il contenuto della coppia di registri HL
36	1	24	inc h	Incrementa di 1 il contenuto del registro H
37	1	25	dec h	Diminuisce di 1 il contenuto del registro H
38	2	26	ld h,N	Carica nel registro H il valore N
39	1	27	daa	Aggiustamento decimale dell'accumulatore
40	2	28	jr z, x	Salta in avanti o indietro di x, se il flag Z è a uno (cioè, se è zero)
41	1	29	add hl,hl	Raddoppia il contenuto della coppia di registri HL
42	3	2A	ld hl,(NN)	Immagazzina nella coppia di registri HL il contenuto della locazione NN
43	1	2B	dec hl	Diminuisce di 1 il contenuto della coppia di registri HL
44	1	2C	inc l	Incrementa di 1 il contenuto del registro L
45	1	2D	dec l	Diminuisce di 1 il contenuto del registro L
46	2	2E	ld l,N	Carica nel registro L il valore N
47	1	2F	cpl	Complementa l'accumulatore
48	2	30	jr nc, x	Salta in avanti o indietro di x se il flag C è a zero (cioè, se è non-carry)
49	3	31	ld sp,NN	Carica nel puntatore allo stack (stack pointer) il valore NN
50	3	32	ld (NN),a	Immagazzina l'accumulatore nella locazione NN
51	1	33	inc sp	Incrementa di 1 il contenuto dello stack pointer
52	1	34	inc (hl)	Incrementa di 1 il contenuto della locazione di memoria indirizzata dalla coppia di registri HL
53	1	35	dec (hl)	Diminuisce di 1 il contenuto della locazione di memoria indirizzata dalla coppia di registri HL
54	2	36	ld (hl),N	Carica il valore N nella locazione di memoria indirizzata dalla coppia di registri HL
55	1	37	scf	Pone a uno il flag C
56	2	38	jr c, x	Salta in avanti o indietro di x se il flag C è a uno (cioè, se è carry)
57	1	39	add hl,sp	Somma al contenuto della coppia di registri HL il contenuto del registro SP (stack pointer) lasciando il risultato in HL
58	3	3A	ld a,(NN)	Immagazzina nell'accumulatore il contenuto della locazione NN
59	1	3B	dec sp	Diminuisce di 1 il contenuto del registro SP (stack poiriter)
60	1	3C	inc a	Incrementa di 1 il contenuto dell'accumulatore

Decimale	Byte	Es.	Codifica	Descrizione
61	1	3D	dec a	Diminuisce di 1 il contenuto dell'accumulatore
62	2	3E	ld a,N	Immagazzina nell'accumulatore il valore N
63	1	3F	ccf	Completa il flag C
64	1	40	ld b,b	Copia il contenuto del registro B nel registro B
65	1	41	ld b,c	Copia il contenuto del registro C nel registro B
66	1	42	ld b,d	Copia il contenuto del registro D nel registro B
67	1	43	ld b,e	Copia il contenuto del registro E nel registro B
68	1	44	ld b,h	Copia il contenuto del registro H nel registro B
69	1	45	ld b,l	Copia il contenuto del registro L nel registro B
70	1	46	ld b,(hl)	Carica il registro B dalla locazione di memoria indirizzata dalla coppia di registri HL
71	1	47	ld b,a	Copia il contenuto dell'accumulatore nel registro B
72	1	48	ld c,b	Copia il contenuto del registro B nel registro C
73	1	49	ld c,c	Copia il contenuto del registro C nel registro C
74	1	4A	ld c,d	Copia il contenuto del registro D nel registro C
75	1	4B	ld c,e	Copia il contenuto del registro E nel registro C
76	1	4C	ld c,h	Copia il contenuto del registro H nel registro C
77	1	4D	ld c,l	Copia il contenuto del registro L nel registro C
78	1	4E	ld c,(hl)	Carica il registro C dalla locazione di memoria indirizzata dalla coppia di registri HL
79	1	4F	ld c,a	Copia il contenuto dell'accumulatore nel registro C
80	1	50	ld d,b	Copia il contenuto del registro B nel registro D
81	1	51	ld d,c	Copia il contenuto del registro C nel registro D
82	1	52	ld d,d	Copia il contenuto del registro D nel registro D
83	1	53	ld d,e	Copia il contenuto del registro E nel registro D
84	1	54	ld d,h	Copia il contenuto del registro H nel registro D
85	1	55	ld d,l	Copia il contenuto del registro L nel registro D
86	1	56	ld d,(hl)	Carica il registro D dalla locazione di memoria indirizzata dalla coppia di registri HL
87	1	57	ld d,a	Copia il contenuto dell'accumulatore nel registro D
88	1	58	ld e,b	Copia il contenuto del registro B nel registro E
89	1	59	ld e,c	Copia il contenuto del registro C nel registro E
90	1	5A	ld e,d	Copia il contenuto del registro D nel registro E
91	1	5B	ld e,e	Copia il contenuto del registro E nel registro E
92	1	5C	ld e,h	Copia il contenuto del registro H nel registro E
93	1	5D	ld e,l	Copia il contenuto del registro L nel registro E
94	1	5E	ld e,(hl)	Carica il registro E dalla locazione di memoria indirizzata dalla coppia di registri HL
95	1	5F	ld e,a	Copia il contenuto dell'accumulatore nel registro E
96	1	60	ld h,b	Copia il contenuto del registro B nel registro H
97	1	61	ld h,c	Copia il contenuto del registro C nel registro H
98	1	62	ld h,d	Copia il contenuto del registro D nel registro H
99	1	63	ld h,e	Copia il contenuto del registro E nel registro H
100	1	64	ld h,h	Copia il contenuto del registro H nel registro H
101	1	65	ld h,l	Copia il contenuto del registro L nel registro H
102	1	66	ld h,(hl)	Carica il registro H dalla locazione di memoria indirizzata dalla coppia di registri HL
103	1	67	ld h,a	Copia il contenuto dell'accumulatore nel registro H
104	1	68	ld l,b	Copia il contenuto del registro B nel registro L
105	1	69	ld l,c	Copia il contenuto del registro C nel registro L
106	1	6A	ld l,d	Copia il contenuto del registro D nel registro L
107	1	6B	ld l,e	Copia il contenuto del registro E nel registro L
108	1	6C	ld l,h	Copia il contenuto del registro H nel registro L
109	1	6D	ld l,l	Copia il contenuto del registro L nel registro L
110	1	6E	ld l,(hl)	Carica il registro L dalla locazione di memoria indirizzata dalla coppia di registri HL
111	1	6F	ld l,a	Copia il contenuto dell'accumulatore nel registro L

Decimale	Byte	Es.	Codifica	Descrizione
112	1	70	ld (hl),b	Copia il contenuto del registro B nella locazione indirizzata da HL
113	1	71	ld (hl),c	Copia il contenuto del registro C nella locazione indirizzata da HL
114	1	72	ld (hl),d	Copia il contenuto del registro D nella locazione indirizzata da HL
115	1	73	ld (hl),e	Copia il contenuto del registro E nella locazione indirizzata da HL
116	1	74	ld (hl),h	Copia il contenuto del registro H nella locazione indirizzata da HL
117	1	75	ld (hl),l	Copia il contenuto del registro L nella locazione indirizzata da HL
118	1	76	halt	Blocca la CPU
119	1	77	ld (hl),a	Copia il contenuto dell'accumulatore nella locazione indirizzata da HL
120	1	78	ld a,b	Copia il contenuto del registro B nell'accumulatore
121	1	79	ld a,c	Copia il contenuto del registro C nell'accumulatore
122	1	7A	ld a,d	Copia il contenuto del registro D nell'accumulatore
123	1	7B	ld a,e	Copia il contenuto del registro E nell'accumulatore
124	1	7C	ld a,h	Copia il contenuto del registro H nell'accumulatore
125	1	7D	ld a,l	Copia il contenuto del registro L nell'accumulatore
126	1	7E	ld a,(hl)	Carica l'accumulatore dalla locazione di memoria indirizzata dalla coppia di registri HL
127	1	7F	ld a,a	Copia il contenuto dell'accumulatore nell'accumulatore
128	1	80	add a,b	B+A→A
129	1	81	add a,c	C+A→A
130	1	82	add a,d	D+A→A
131	1	83	add a,e	E+A→A
132	1	84	add a,h	H+A→A
133	1	85	add a,l	L+A→A
134	1	86	add a,(hl)	(HL)+A→A
135	1	87	add a,a	A+A→A
136	1	88	adc a,b	B+A+carry→A
137	1	89	adc a,c	C+A+carry→A
138	1	8A	adc a,d	D+A+carry→A
139	1	8B	adc a,e	E+A+carry→A
140	1	8C	adc a,h	H+A+carry→A
141	1	8D	adc a,l	L+A+carry→A
142	1	8E	adc a,(hl)	(HL)+A+carry→A
143	1	8F	adc a,a	A+A+carry→A
144	1	90	sub b	A−B→A
145	1	91	sub c	A−C→A
146	1	92	sub d	A−D→A
147	1	93	sub e	A−E→A
148	1	94	sub h	A−H→A
149	1	95	sub l	A−L→A
150	1	96	sub (hl)	A−(HL)→A
151	1	97	sub a	A−A→A
152	1	98	sbc a,b	A−B−carry→A
153	1	99	sbc a,c	A−C−carry→A
154	1	9A	sbc a,d	A−D−carry→A
155	1	9B	sbc a,e	A−E−carry→A
156	1	9C	sbc a,h	A−H−carry→A
157	1	9D	sbc a,l	A−L−carry→A
158	1	9E	sbc a,(hl)	A−(HL)−carry→A
159	1	9F	sbc a,a	A−A−carry→A
160	1	A0	and b	A and B→A

Decimale	Byte	Es.	Codifica	Descrizione
161	1	A1	and c	A and C→A
162	1	A2	and d	A and D→A
163	1	A3	and e	A and E→A
164	1	A4	and h	A and H→A
165	1	A5	and l	A and L→A
166	1	A6	and (hl)	A and (HL)→A
167	1	A7	and a	A and A→A
168	1	A8	xor b	A exclusive or B→A
169	1	A9	xor c	A exclusive or C→A
170	1	AA	xor d	A exclusive or D→A
171	1	AB	xor e	A exclusive or E→A
172	1	AC	xor h	A exclusive or H→A
173	1	AD	xor l	A exclusive or L→A
174	1	AE	xor (hl)	A exclusive or (HL)→A
175	1	AF	xor a	A exclusive or A→A
176	1	B0	or b	A or B→A
177	1	B1	or c	A or C→A
178	1	B2	or d	A or D→A
179	1	B3	or e	A or E→A
180	1	B4	or h	A or H→A
181	1	B5	or l	A or L→A
182	1	B6	or(hl)	A or (HL)→A
183	1	B7	or a	A or A→A
184	1	B8	cp b	Sottrae il contenuto del registro B dall'accumulatore, scartando il risultato
185	1	B9	cp c	Sottrae il contenuto del registro C dall'accumulatore, scartando il risultato
186	1	BA	cp d	Sottrae il contenuto del registro D dall'accumulatore, scartando il risultato
187	1	BB	cp e	Sottrae il contenuto del registro E dall'accumulatore, scartando il risultato
188	1	BC	cp h	Sottrae il contenuto del registro H dall'accumulatore, scartando il risultato
189	1	BD	cp l	Sottrae il contenuto del registro L dall'accumulatore, scartando il risultato
190	1	BE	cp (hl)	Sottrae dall'accumulatore il contenuto della locazione indirizzata dalla coppia di registri HL, scartando il risultato (cioè lasciando invariato il contenuto dell'accumulatore)
191	1	BF	cp a	Sottrae il contenuto dell'accumulatore, scartando il risultato
192	1	C0	ret nz	Ritorna se il flag Z è a zero (cioè se è non-zero)
193	1	C1	pop bc	Carica dallo stack la coppia di registri BC
194	3	C2	jp nz,NN	Salta alla locazione NN se il flag Z è a zero
195	3	C3	jp NN	Salta alla locazione NN
196	3	C4	call nz,NN	Chiama la routine alla locazione NN se il flag Z è a zero
197	1	C5	push bc	Salva nello stack il contenuto della coppia di registri BC
198	2	C6	add a,N	A+N→A
199	1	C7	rst 0	Chiama la routine iniziante alla locazione 0000
200	1	C8	ret z	Ritorna se il flag Z è a uno (cioè se è zero)
201	1	C9	ret	Ritorna
202	3	CA	jp z,NN	Salta alla locazione NN se il flag Z è a uno
203		CB		Vedi l'insieme di routine CB
204	3	CC	call z,NN	Chiama la routine alla locazione NN se il flag Z è a uno
205	3	CD	call NN	Chiama la routine alla locazione NN
207	1	CF	rst 8	Chiama la routine di errore all'indirizzo 0008
208	1	D0	ret nc	Ritorna se il flag C è a zero (cioè se non-carry)
209	1	D1	pop de	Carica dallo stack la coppia di registri DE
210	3	D2	jp nc,NN	Salta alla locazione NN se il flag C è a zero

Decimale	Byte	Es.	Codifica	Descrizione
211	2	D3	out(N),a	Emette il contenuto dell'accumulatore alla porta N
212	3	D4	call nc,NN	Chiama la routine alla locazione NN se il flag C è a zero
213	1	D5	push de	Salva nello stack il contenuto della coppia di registri DE
214	2	D6	sub N	A−N→A
215	1	D7	rst 16	Chiama la routine di stampa all'indirizzo 0010
216	1	D8	ret c	Ritorna se il flag C è a uno
217	1	D9	exx	Scambia i registri principali con quelli alternativi
218	3	DA	jp c,NN	Salta alla locazione NN se il flag C è a uno
219	2	DB	in a,(N)	Carica l'accumulatore dalla porta N
220	3	DC	call c,NN	Chiama la routine alla locazione NN se il flag C è a uno
221		DD	[prefissi delle istruzioni che usano ix]	Vedi Tabella D.2
222	2	DE	sbc a,N	A−N−carry→A
223	1	DF	rst 24	Chiama la routine di carattere all'indirizzo 0018
224	1	E0	ret po	Ritorna se il flag P/O è a zero
225	1	E1	pop hl	Carica dallo stack la coppia di registri HL
226	3	E2	jp po,NN	Salta alla locazione NN se il flag P/O è a zero
227	1	E3	ex(sp),hl	Scambia il contenuto del registro SP con quello della coppia di registri HL
228	3	E4	call po,NN	Chiama la routine all'indirizzo NN se il flag P/O è a zero
229	1	E5	push hl	Salva nello stack il contenuto della coppia di registri HL
230	2	E6	and N	A and N→A
231	1	E7	rst 32	Chiama la routine di carattere alla locazione 0020
232	1	E8	ret pe	Ritorna se il flag P/O è a uno
233	1	E9	jp (hl)	Salta alla locazione indirizzata dalla coppia di registri HL
234	3	EA	jp pe,NN	Salta alla locazione NN se il flag P/O è a uno
235	1	EB	ex de,hl	Scambia il contenuto della coppia di registri DE con quello della coppia HL
236	3	EC	call pe,NN	Chiama la routine all'indirizzo NN se il flag P/O è a uno
237		ED		Vedi l'insieme di routine ED
238	2	EE	xor N	A exclusive or N→A
239	1	EF	rst 40	Chiama la routine calcolatore all'indirizzo 0028
240	1	F0	ret p	Ritorna se il flag P è a zero
241	1	F1	pop af	Carica dallo stack la coppia di registri AF
242	3	F2	jp p,NN	Salta alla locazione NN se il flag P è a zero
243	1	F3	di	Disabilita gli interrupt
244	3	F4	call p,NN	Chiama la routine all'indirizzo NN se il flag P è a zero
245	1	F5	push af	Salva nello stack il contenuto della coppia di registri AF
246	2	F6	or N	A or N→A
247	1	F7	rst 48	Chiama la routine spazio di lavoro all'indirizzo 0030
248	1	F8	ret m	Ritorna se il flag P è a uno
249	1	F9	ld sp,hl	Copia il contenuto del registro HL nel registro SP (stack pointer)
250	3	FA	jp m,NN	Salta alla locazione NN se il flag P è a uno
251	1	FB	ei	Abilita gli interrupt
252	3	FC	call m,NN	Chiama la routine all'indirizzo NN se il flag P è a uno
253		FD	[prefissi delle istruzioni che usano iy]	Vedi Tabella D.2
254	2	FE	cp N	Sottrae al contenuto dell'accumulatore il valore N scaricando il risultato
255	1	FF	rst 56	Chiama la routine all'indirizzo 0038

Tabella D.2 Codici utilizzati con DD e FD

d significa spostamento (displacement): dd e dddd significano rispettivamente numeri di uno e due byte.
La tabella si riferisce agli mnemonici DD (221 decimale) in relazione col registro IX.
Gli mnemonici FD (253 decimale) usano operazioni simili, ma in relazione col registro IY.

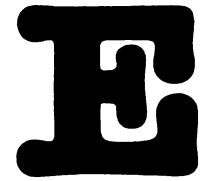
DD 09	ADD	IX,BC	DD CB d 06	RLC	(IX+d)
DD 19	ADD	IX,DE	DD CB d 0E	RRC	(IX+d)
DD 21 +dddd	LD	IX,+dddd	DD CB d 16	RL	(IX+d)
DD 22 addr	LD	(addr),IX	DD CB d 1E	RR	(IX+d)
DD 23	INC	IX	DD CB d 26	SLA	(IX+d)
DD 29	ADD	IX,IX	DD CB d 2E	SRA	(IX+d)
DD 2A addr	LD	IX,(addr)	DD CB d 3E	SRL	(IX+d)
DD 2B	DEC	IX	DD CB d 46	BIT	0,(IX+d)
DD 34 d	INC	(IX+d)	DD CB d 4E	BIT	1,(IX+d)
DD 35 d	DEC	(IX+d)	DD CB d 56	BIT	2,(IX+d)
DD 36 d +dd	LD	(IX+d),+dd	DD CB d 5E	BIT	3,(IX+d)
DD 39	ADD	IX,SP	DD CB d 66	BIT	4,(IX+d)
DD 46 d	LD	B,(IX+d)	DD CB d 6E	BIT	5,(IX+d)
DD 4E d	LD	C,(IX+d)	DD CB d 76	BIT	6,(IX+d)
DD 56 d	LD	D,(IX+d)	DD CB d 7E	BIT	7,(IX+d)
DD 5E d	LD	E,(IX+d)	DD CB d 86	RES	0,(IX+d)
DD 66 d	LD	H,(IX+d)	DD CB d 8E	RES	1,(IX+d)
DD 6E d	LD	L,(IX+d)	DD CB d 96	RES	2,(IX+d)
DD 70 d	LD	(IX+d),B	DD CB d 9E	RS	3,(IX+d)
DD 71 d	LD	(IX+d),C	DD CB d A6	RES	4,(IX+d)
DD 72 d	LD	(IX+d),D	DD CB d AE	RES	5,(IX+d)
DD 73 d	LD	(IX+d),E	DD CB d B6	RES	6,(IX+d)
DD 74 d	LD	(IX+d),H	DD CB d BE	RES	7,(IX+d)
DD 75 d	LD	(IX+d),L	DD CB d C6	SET	0,(IX+d)
DD 77 d	LD	(IX+d),A	DD CB d CE	SET	1,(IX+d)
DD 7E d	LD	A,(IX+d)	DD CB d D6	SET	2,(IX+d)
	ADD	A,(IX+d)	DD CB d DE	SET	3,(IX+d)
DD 8E d	ADC	A,(IX+d)	DD CB d E6	SET	4,(IX+d)
DD 96 d	SUB	(IX+d)	DD CB d EE	SET	5,(IX+d)
DD 9E d	SBC	A,(IX+d)	DD CB d F6	SET	6,(IX+d)
DD A6 d	AND	(IX+d)	DD CB d FE	SET	7,(IX+d)
DD AE d	XOR	(IX+d)	DD E1	POP	IX
DD B6 d	OR	(IX+d)	DD E3	EX	(SP),IX
DD BE d	CP	(IX+d)	DD E5	PUSH	IX
			DD E9	JP	(IX)
			DD F9	LD	SP,IX

Tabella D.3 Istruzioni ED (237 decimale)

ED 40 IN B,(C)	ED 50 IN D,(C)	ED 60 IN H,(C)		ED A0 LDI	ED B0 LDIR
ED 41 OUT (C),B	ED 51 OUT (C),D	ED 61 OUT (C),H		ED A1 CPI	ED B1 CPIR
ED 42 SBC HL,BC	ED 52 SBC HL,DE	ED 62 SBC HL,HL	ED 72 SBC HL,SP	ED A2 INI	ED B2 INIR
ED 43 (addr),BC	ED 53 (addr),DE	ED 63 (addr),HL	ED 73 (addr),SP	ED A3 OUTI	ED B3 OTIR
ED 44 NEG					
ED 45 RETN					
ED 46 IM 0	ED 56 IM 1	ED 66 IM 2			
ED 47 LD I,A	ED 57 LD A,I	ED 67 RRD			
ED 48 IN C,(C)	ED 58 IN E,(C)	ED 68 IN L,(C)	ED 78 IN A,(C)	ED A8 LDD	ED B8 LDDR
ED 49 OUT (C),C	ED 59 OUT (C),E	ED 69 OUT (C),L	ED 79 OUT (C),A	ED A9 CPD	ED B9 CPDR
ED 4A ADC HL,BC	ED 5A ADC HL,DE	ED 6A ADC HL,HL	ED 7A ADC HL,SP	ED AA IND	ED BA INDR
ED 4B LD BC,(addr)	ED 5B LD DE,(addr)	ED 6B LD HL,(addr)	ED 7B LD SP,(addr)	ED AB OUTD	ED BB OTRD
ED 4D RETI					
ED 4F LD R,A	ED 5F LD A,R	ED 6F RLD			

Appendice

Indice del sistema GOLDMINE



Questa appendice è il vero e proprio indice del sistema: in essa sono elencate tutte le routine GOLDMINE, così come si trovano nella RAM dello Spectrum una volta caricato il sistema, in ordine progressivo dalla locazione 62000 alla 65235.
Il sistema è diviso in quattro parti distinte:

1. Routine chiamabili dal BASIC, nelle locazioni 62000-62373
2. Variabili di sistema (per le quali bisogna fare riferimento all'Appendice A), nelle locazioni 62374-62448
3. Tabella dei dati, nelle locazioni 62449-63042
4. Routine in codice macchina, nelle locazioni 63043-65235

Di fianco all'indirizzo iniziale, al nome e alla descrizione di ogni routine, troverete il riferimento alla pagina del testo in cui essa viene descritta in dettaglio

E.1 Routine chiamabili dal BASIC

E.2 Routine in codice macchina

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

88 7700 001 5 J. Heilborn e R. Talbott, *Guida al Commodore 64*
88 7700 002 3 C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
88 7700 003 1 T. Woods, *L'Assembler per lo ZX Spectrum*
88 7700 004 X R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
88 7700 005 8 G. Bishop, *Progetti hardware con lo ZX Spectrum*
88 7700 006 6 H. Mullish e D. Kruger, *Il BASIC Applesoft*
88 7700 007 4 N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
88 7700 008 2 H. Peckham, *Il BASIC e il PC-IBM in pratica*
88 7700 009 0 H. Peckham, *Il BASIC e il Commodore 64 in pratica*
88 7700 010 4 S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*
88 7700 011 2 K. Skier, *L'Assembler per il Commodore 64 e il VIC-20*
88 7700 012 0 S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
88 7700 013 9 A. Pennell, *Guida allo ZX Microdrive e all'Interface 1*
88 7700 015 5 P. Cohen, *Grafica e animazione con gli Apple II*
88 7700 016 3 C. Duff, *Guida al Macintosh*
88 7700 017 1 G. Kane, *Il manuale MC68000*
88 7700 018 X P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*
88 7700 020 1 S. Nicholls, *Grafica avanzata con lo ZX Spectrum*
88 7700 021 X L.J. Graham e T. Field, *Guida al PC-IBM*
88 7700 022 8 T. Field, *Come usare MacWrite e MacPaint*

In preparazione

88 7700 024 4 H. Peckham, *Il BASIC e gli Apple II in pratica*
88 7700 025 2 C. Morgan e M. Waite, *Il manuale 8086/8088*
88 7700 026 0 W. Ettlin, *Come usare il Multiplan*

I giochi di animazione rappresentano uno dei campi di applicazione più divertenti dello ZX Spectrum e le caratteristiche di questo microcomputer sono tali da permettere la creazione di giochi a livello quasi professionale. Gli strumenti a disposizione nell'hardware fornito - in particolare le routine della ROM - sono però carenti in termini di flessibilità e velocità, elementi fondamentali per i giochi veloci; per questo motivo Stuart Nicholls ha ideato un "sistema alternativo", chiamato GOLDMINE (miniera d'oro), che contiene una serie di routine specialmente orientate alla creazione di giochi.
Le routine GOLDMINE coprono l'intera gamma delle utility necessarie alla grafica avanzata, suddivise nei seguenti argomenti:
- scansione della tastiera
- visualizzazione sullo schermo
- tracciamento di linee
- rilevamento di caratteri e punti
- musica ed effetti sonori
- punteggi, conteggi e numeri casuali
- roll e scroll di una finestra
- movimento di sprite.

Un generatore di caratteri e una serie di routine di gestione dello schermo completano il sistema GOLDMINE rendendolo indipendente, più veloce ed efficiente.
Ogni routine è spiegata sia al programmatore BASIC che al principiante e all'esperto di codice macchina.
Il testo è completato da una guida rapida del sistema e dal listato Assembler di tutte le procedure GOLDMINE, disponibili anche su cassetta col titolo **Routines in Assembler per la grafica avanzata con lo ZX Spectrum.**

Lire 18 000
(IVA 2% inclusa)

ISBN 88 7700 020 1