Este manual elucida o leitor sobre todos os passos a seguir, ao tratar profissionalmente a programação do ZX Spectrum. John Lettice mostra-lhe como explorar ao máximo as capacidades do seu microcomputador e como abordar, da maneira mais correcta, os problemas da sua programação. Conheça melhor a enorme flexibilidade do ZX Spectrum aprofundando os seus conhecimentos sobre temas como a programação interactiva, o tratamento da informação, o uso de quadros, gráficos definidos pelo utilizador, variáveis de sistema, *sprites* e animação, memória e utilização do som. Uma obra a não perder, para aqueles que pretendem alargar os seus conhecimentos sobre as potencialidades quase inesgotáveis de um pequeno computador tão espantoso como o ZX Spectrum.

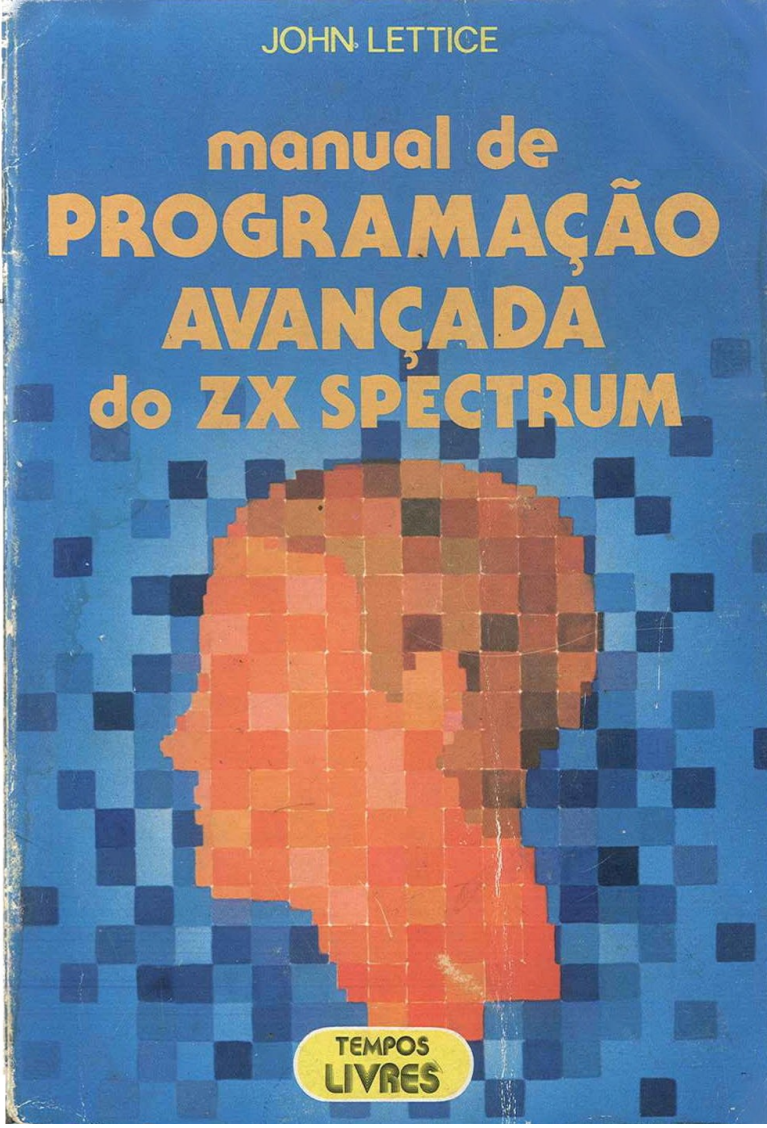
EDITORIAL PRESENÇA

198

Manual de Programação Avançada do ZX Spectrum

JOHN LETTICE

manual de PROGRAMAÇÃO AVANÇADA do ZX SPECTRUM



TEMPOS LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Peter Trifunovitch e Sava Vulkovitch*
4. ABC do Bridge, *Pierre Jais e H. Lahana*
5. Guia Prático de Fotografia, *W. D. Emanuel*
6. ABC do Judo, *E. J. Harrison*
7. Como Fazer Cinema, *Paul Petzold*
8. Bridge Moderno, *Pierre Jais e H. Lahana*
9. Fotografia – Técnicas e Truques I, *Edwin Smith*
10. ABC dos Estilos, da Arquitectura ao Mobiliário, *A. Aussel*
11. Fotografia – Técnicas e Truques II, *Edwin Smith*
12. A Pesca Submarina, *António Ribera*
13. Teoria dos Finais de Partida, *Yuri Averbach*
14. Aprenda Rádio, *B. Fighiera*
15. Guia do Cão, *Louise Laliberté-Robert e Jean-Pierre Robert*
16. ABC do Aquário, *Anthony Evans*
17. Iniciação à Electricidade e Electrónica, *Fernand Huré*
18. Os Transístores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricone*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *Curt Af Enehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez – Teoria do Meio Jogo I, *I. Bondarevsky*
35. Manual do Super 8 – I, *Myron A. Matzkin*
36. ABC da Criação de Periquitos, *Cyril H. Rogers*
37. O Livro dos Gatos, *Barbel Gerber e Horst Bielfeld*
38. Manual do Super 8 – II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. Witham Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Cocker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabian Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*
51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires e C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelhos, *P. Surdeau e R. Henaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen e Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Boëlle e D. Doyen*
63. Enxerto de Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *Luis Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*
67. Doces e Conservas de Fruta, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Rioy Genders*
70. Calendário do Horticultor, *Luis Alsina Grau*
71. Jogos Electrónicos, *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, *Alejo Rigau*
73. Aprenda Televisão a Cores, *Gordon J. King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Poda de Árvores e Arbustos, *Roy Genders*
76. Como Treinar o Seu Cão, *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, *Heinrich Stöckle*
78. A Criação de Caracóis, *Matías Josa*
79. Rádio – Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébault*
81. Iniciação à Jardinagem, *Noel Clarasó*
82. A Congelação dos Alimentos, *Suzanne Lapointe*
83. Windsurf – Prancha à Vela, *Ernstfried Prade*
84. Raças de Cães, *O. Hasselfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia, Embalsamamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging – Correr Para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos TV, *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie de Prémont e Nicole Philippi*
95. O Minitrampolim na Escola, *Sonja Helmer Jensen e Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, *B. Fighiera*
97. O Cultivo do Tomate, *Louis N. Flawn*
98. Pilhas Solares, *F. Juster*
99. A Criação Doméstica de Coelhos, *C. F. Snow*
100. Iniciação ao Futebol, *Wieland Männle e Heinz Arnold*
101. Horóscopos Chineses, *Georg Haddenbach*
102. Guia Prático de Marcenaria, *Charles H. Hayward*
103. Andebol, *Fritz e Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perús, Pintadas e Codornizes, *Jérome Sauze*
106. Crepes – Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott e M. Baker*
110. Futebol – Técnica e Táctica, *Kurt Lavall*
111. A Vaca Leiteira, *Colin T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*
113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marieanne Rank*
115. O Cubo Para Quem Já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, *M. Mrowka-W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung-Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hans-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hill*
121. Construção de um Alimentador de Corrente, *Waldemar Baittinger*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *Christa-Maria e Richard Kerler*
126. A Pesca e os seus Segredos, *Armand Deschamps*
127. O Osciloscópio, *R. Rateau*
128. Guia Prático da Banda do Cidadão, *T. M. Normand*
129. Sumos e Batidos, *Manfred Donderski*
130. Introdução à Programação de Microcomputadores, *Peter C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*
133. Guia Prático de Basic, *Roger Hunt*

134. Introdução à Electrónica Digital, *Ian R. Sinclair*
135. ABC do Vídeo, *David Matthewson*
136. Fotografia em Movimento, *Don Morley*
137. Guia de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *Sidney F. Ray*
139. Guia Moderno da Canaricultura, *Manuel Gonçalves*
140. Minielectrónica para Amadores, *Heinz Richter*
141. ABC da Programação de Computadores, *John Shelley*
142. Tarot – O Futuro Pelas Cartas, *Edwin J. Nigg*
143. ABC da Equitação, *Dorothy Johnson*
144. Como Programar o seu ZX 81, *Patrick Gueulle*
145. 100 Avarias TV e a Maneira Prática de as Detectar, *P. Duranton*
146. ABC da Horticultura, *Louis Giordano*
147. Basic Para Microcomputadores, *A. P. Stephenson*
148. Como Programar o seu ZX Spectrum, *Tim Hartnell e Dilwyn Jones*
149. Iniciação aos Motores Diesel, *David S. Maclean*
150. 60 Jogos Para o ZX Spectrum, *David Harwod*
151. As Linhas da Mão, *Rose Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *R. J. Simpson e T. J. Terrell*
154. Z80 Assembler Para o ZX Spectrum – Iniciação ao Código de Máquina, *João Paulo Fragoso*
155. Aeróbica, *Hans Schulz*
156. ABC do Atletismo, *Denis Watts*
157. 26 Programas Basic Para Microcomputadores, *Derrick Daines*
158. Aprenda Pascal no Seu Microcomputador, *Jeremy Ruston*
159. Guia Moderno da Suinicultura, *Colin Whittemore*
160. O Bar em Sua Casa – 888 Cocktails, *Aladar von Wesendonk*
161. Código de Máquina Para Principiantes, *James Walsh*
162. Código de Máquina Para Programadores Avançados, *Paul Holmes*
163. ABC da Fruticultura, *Henri Grosselin*
164. ABC da Canoagem, *Alan Byde*
165. Guia de Fortran, *Philip Ridler*
166. Manual da Secretária, *Phillippa Ramage*
167. ABC das Antenas, *Gordon J. King*
168. Programar Aventuras no Seu Computador, *Anfrew Nelson*
169. Guia do Sinclair QL, *Boris Allan*
170. Novas Aventuras no Seu ZX Spectrum, *Peter Shaw e James Mortleman*
171. O Computador no Escritório, *John Shelley*
172. Sobremesas, *Fred Timber*
173. Rádio – Do Circuito Oscilante ao Receptor de Ondas Curtas, *Richard Zierl*
174. Xadrez: ABC das Aberturas, *V. N. Panov*
175. A Construção de Pequenos Transformadores, *M. Dorian e F. Juster*
176. Guia de Pascal, *David Watt*
177. O ZX Spectrum na Educação, *Tim Hartnell, Christine Johnson e David Valentine*
178. Iniciação ao Snooker, *John Pulman*
179. Circuitos com Diacs, Tiristores e Triacs, *Fritz Bergtold*
180. O Jogo do Mah-Jong, *Ursula Eschenbach*
181. Cultura Hidropónica, Culturas sem solo, *Laura Fronty*
182. 49 Jogos Explosivos para o ZX Spectrum, *Tim Hartnell, D. Perry, G. Charlton, N. Pellinacci e M. Young*
183. Iniciação ao Golfe, *J. C. Jessop*
184. Como se Constrói um Receptor de FM, *Richard Zierl*
185. Iniciação à Navegação Marítima, *Anselmo Vieira*
186. Processamento de Dados, *T. F. Fry*
187. O ZX Spectrum na Gestão de Pequenas Empresas, *Luís de Campos*
188. Programar Grafismos no seu Computador, *Michel Rousselet*
189. O IBM PC e Sistemas Compatíveis, *Carlos Reis e João Capaz*
190. Antenas para a Banda do Cidadão, *Patrick Gueulle*
191. Paciências, *Irmgard Wolter*
192. Defesa Pessoal, *Syd Hoare*
193. Processamento de texto, *Lew Hollerbach*
194. Inteligência Artificial no Seu Spectrum e Spectrum +, *Tim Hartnell*
195. Manual do Correspondente de Inglês, *W. G. Tackle e Gomes Pitta*
196. Como Interpretar um Balanço, *Carlos Nabais*
197. Iniciação à Tecnologia da Informação, *Garry Marchall*
198. Manual de Programação Avançada do ZX Spectrum, *John Lettice*

JOHN LETTICE

MANUAL DE PROGRAMAÇÃO AVANÇADA DO ZX SPECTRUM

EDITORIAL PRESENÇA

FICHA TÉCNICA

Título original:
TURBOCHARGE YOUR ZX SPECTRUM

Edição publicada por acordo com Longman Group Limited, London
Título da edição portuguesa: *Manual de Programação Avançada do ZX Spectrum*

Tradução: *Eduardo Nogueira*
Capa: *Rogério M. Silva*
Fotocomposição: *Nova Força, Lisboa*
Impressão e Acabamento: *Tipografia Nunes, Porto*

1.ª edição, Lisboa, 1986

NOTA

Muitos dos programas e rotinas deste livro foram desenvolvidos, gravados e usados em Microdrives Sinclair. Isto significa que tais programas deverão ser adaptados para uso em cassete ou outro suporte. Em geral, este facto é mencionado no texto que introduz esses programas e rotinas. No entanto, esses programas e rotinas serão fáceis de identificar através das instruções LOAD e SAVE com a sintaxe seguinte:

```
LOAD * "M"; 1; "nome"
SAVE * "M"; 1; "nome"
```

Para adaptar os programas ao uso em cassete, substitua as instruções anteriores por:

```
LOAD "nome"
SAVE "nome"
etc.
```

No entanto, a ordem de gravação e leitura de blocos de dados e programas em alguns dos casos deverá ser alterada a fim de corresponder à ordem pela qual se encontram gravados em cassete.

1

INTRODUÇÃO AVANÇADA AO SPECTRUM

Depois de o leitor entrar em contacto com os aspectos básicos da programação do Spectrum, começará provavelmente a pensar sobre a direcção que deverá tomar em seguida. Deverá começar a aprender código-máquina, deverá comprar um Assembler, ou deve tentar criar programas mais estruturados?

Destes três rumos o terceiro é provavelmente o mais fácil de aprender, mas é muitas vezes o mais difícil de manter, especialmente numa máquina como o Spectrum, mais orientada para produzir bons resultados do que para a escrita de programas elegantes. Mas apesar de tudo, há vantagem em conhecer um pouco os princípios de estruturação, mesmo que nem sempre sejam usados.

Para que serve a estrutura?

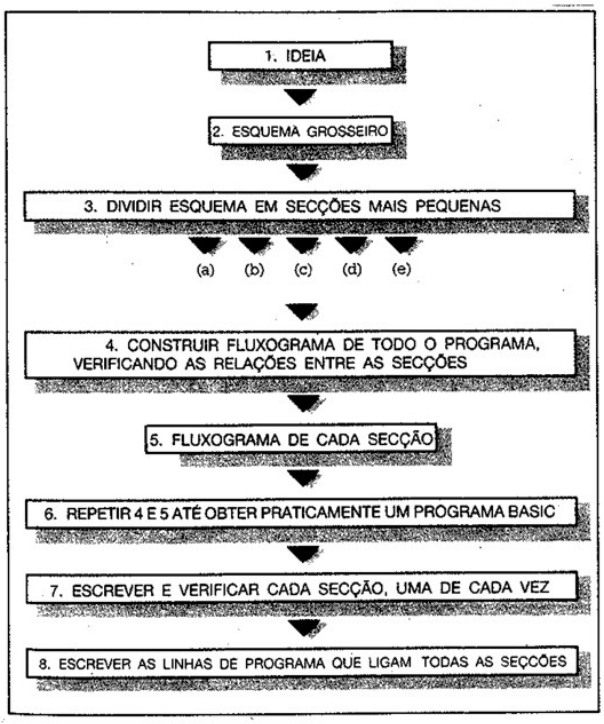
O que é então a estrutura? Em termos de uma programação eficaz, a estrutura serve para escrever programas em pequenas secções fáceis de compreender. Como estas podem ser incluídas no programa geral recorrendo a instruções GOSUB, os programas podem ser alterados facilmente, e depois de se ter estado a escrever programas deste modo durante algum tempo acaba-se por obter uma extensa gama de subrotinas que podem ser incorporadas nos programas que se venham a escrever no futuro.

À medida que a capacidade de programação melhora, e que se acrescentam impressoras, microdrives, etc., desejar-se-á

também melhorar ou modificar os programas anteriores, de tal modo que se os programas se encontram claramente divididos em secções se tornará muito mais fácil compreender o que fazem quando voltamos a eles.

O principal causador da programação não estruturada é a instrução GO TO. Suponhamos que o leitor começa a escrever um programa, e que há medida que progride lhe ocorrem magníficas ideias como seja a introdução de alguns gráficos.

Escrita de um programa



O leitor apressa-se portanto a acrescentar um GO TO, escreve uma brilhante rotina de gráficos, e acrescenta um novo GO TO que devolve a execução ao programa principal. Uma só rotina utiliza dois saltos, mas que acontecerá se o leitor tiver mais algumas ideias magníficas? O seu programa começa a parecer-se com um prato de *spaghetti*, bastante difícil de compreender mesmo quando está a ser escrito. O que acontecerá quando voltar ele ao fim de alguns dias (nem vale a pena pensar em seis meses) ou quando dá uma cópia do seu programa aos amigos?

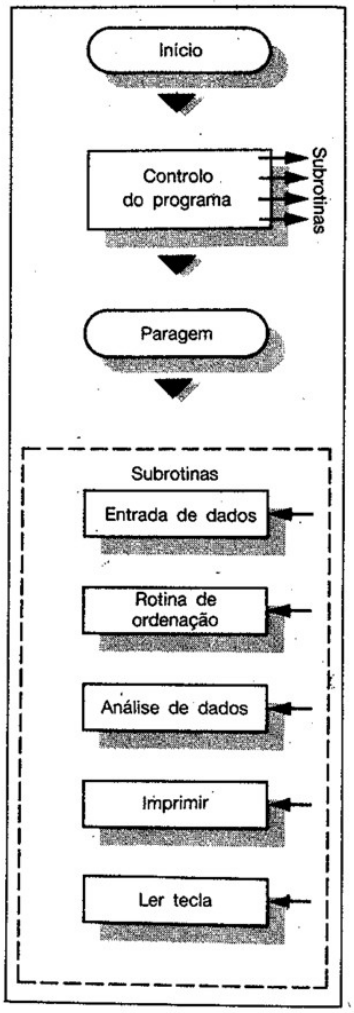
O uso de subrotinas sempre que possível evita que isto aconteça, e torna a vida de toda a gente bastante mais fácil. E pense no que está a fazer com as subrotinas. Estará a dividir o programa em pequenas secções fáceis de compreender, mas que implicará isto? Deve pensar sobre o que vai fazer com um programa antes de começar a escrevê-lo – pelo que a estruturação tem tanto a ver com o planeamento como com o uso de instruções GOSUB!

Subrotinas

A tabela da página seguinte deveria dar-lhe uma boa ideia sobre como se relaciona a estruturação e o emprego de subrotinas. Depois de se ter a ideia de um programa pode começar-se a fazê-lo, mas isso será apenas arranjar problemas, e convirá muito mais escrever o máximo possível em papel antes de se aproximar sequer do computador.

Digamos que o leitor decide escrever um programa envolvendo manuseamento de dados. Como provavelmente estará a introduzir os dados nele por uma ordem qualquer, convirá dispor de alguma rotina de ordenação que lhe permita organizá-lo. Depois de ter escrito esta rotina bastar-lhe-á gravá-la em fita ou microdrive, usando-a em programas futuros sob a forma:

```
 100 GO SUB 1000: REM Rotina de
ordenacao
 999 REM final da rotina de orde
nacao
1000 ....
1010 RETURN
```



Se se gasta muito tempo em pequenas rotinas como esta, talvez nos próximos dez anos se descubra que não é preciso escrever mais nada – basta consultar a biblioteca de rotinas! Note que colocando a REM na linha 999 em vez de o fazer na 1000 lhe permite cortá-la mais tarde para poupar espaço.

Estudando primeiro a estrutura de cada programa em papel, está a pensar-se em termos de programas consistindo em estruturas de controlo. Considerando que as subrotinas que de facto realizam o trabalho podem ser acrescentadas mais tarde está-se de facto a trabalhar a um nível muito mais elevado do que a linguagem Basic, e os puristas diriam até que todo o programa deveria ser primeiramente escrito em papel.

Mas esta ideia é de facto uma recordação dos tempos em que os programadores utilizavam computadores grandes, e em que o tempo de computador era reduzido. Se se define a estrutura em papel, bastará geralmente aperfeiçoar as subrotinas no visor. Neste caso será útil uma impressora, dado que nunca se dispõe de mais de 24 linhas no visor e se torna portanto bastante complicado tratar mais do que uma rotina simultaneamente.

Regra

Um aspecto importante das subrotinas é que correm de um modo lógico – entra-se nelas no princípio, e sai-se delas no fim. Vejamos o que se passa no exemplo seguinte:

```
 10 GO SUB 100
 20 ....
 30 ....
 99 REM inicio da subrotina
100 ....
110 ...
120 IF a$="SAIR" THEN GO TO 20
130 ....
140 RETURN
```

Escrevendo um programa como este estar-se-ia a eliminar as vantagens da subrotina, porque se estaria a incluir um salto para fora dela na linha 120. Se a subrotina for muito comprida, poderia interessar o uso de uma GO TO para apressar o

programa (evitando que o Spectrum passasse uma série de linhas completamente desnecessárias), mas isto é apenas uma boa razão para dividir a subrotina em rotinas ainda mais pequenas, e de qualquer modo continua a ser possível apressar a execução fazendo um salto para a instrução de retorno final.

Uma experiência

Para além de um fluxo lógico do programa existem boas razões para sair sempre de um programa pela linha RETURN. Escreva este pequeno programa e execute-o:

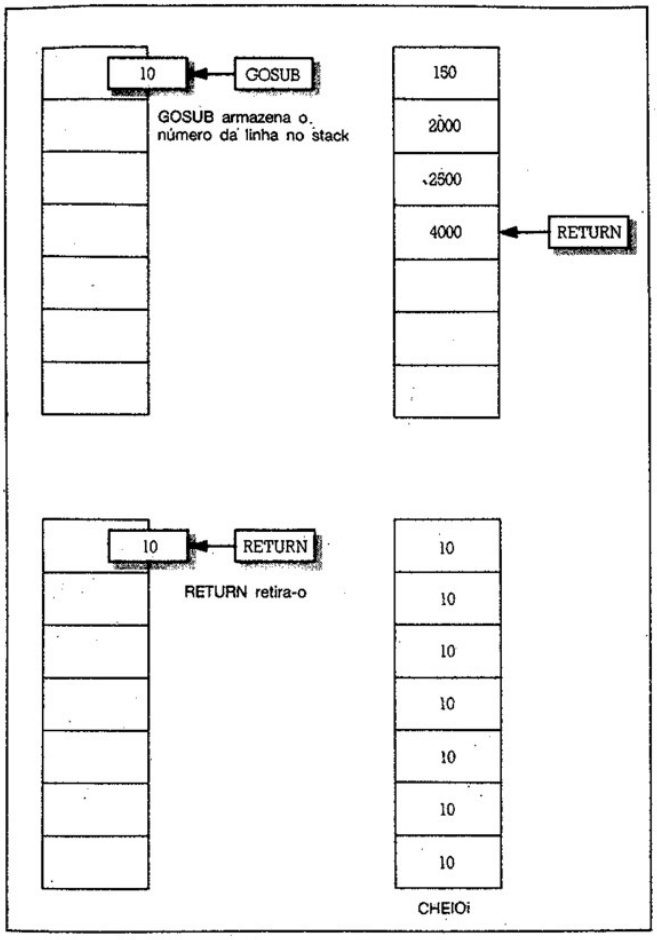
```
 10 GO SUB 100
 20 STOP
100 GO TO 10
```

Que aconteceu? Ao executar este programa, o leitor gastou completamente a memória do Spectrum conhecida pelo nome de «stack». Quando se escreve num programa GO SUB 100, como aqui se faz na linha 10, o computador recorda a linha 10, guardando-a naquilo que se chama «stack» de subrotinas. O primeiro RETURN que encontrar reenvia-o para a linha 10, pelo que escrevemos aqui um programa que diz sistematicamente ao computador para se lembrar de uma linha sem depois lhe dizer para a esquecer.

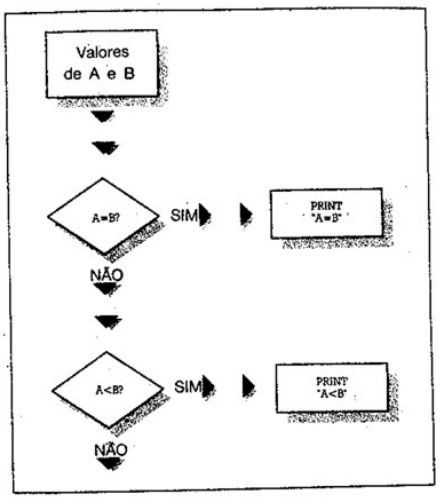
Se pensar um pouco no assunto, o leitor compreenderá como o «stack» funciona – se tiver subrotinas em rede, umas invocadas pelas outras, a execução deve executar um RETURN à segunda instrução GO SUB antes de voltar à primeira. «O último que entra é o primeiro a sair».

Saltos condicionais

A questão que interessa considerar a seguir na estruturação de um programa é a execução de saltos condicionais, que constitui basicamente a forma como o computador toma decisões. Pode pensar-se nestes saltos muito simplesmente como constituindo uma forma de levar o computador a verificar se uma



determinada coisa é verdadeira, tomando em seguida a acção apropriada. Isto toma geralmente a forma *IF condição THEN acção*.



Juntamente com a instrução IF... THEN usam-se os símbolos =, < , > e < >, que são conhecidos pela designação genérica de operadores lógicos, e significam igual a, menor do que, maior do que e diferente de, respectivamente. Outros operadores possíveis são AND, OR e NOT.

```
10 LET a=10: LET b=5
20 IF a=b THEN PRINT "A=B"
30 IF a<b THEN PRINT "A e' men
or que B"
40 IF a>b THEN PRINT "A e' mai
or que B"
50 IF a<>b THEN PRINT "A e' di
ferente de B"
```

É ainda possível combinar condições usando os operadores AND e OR:


```
60 IF a=10 OR b=10 THEN PRINT
"Uma das condicoes e' verdadeira
"
70 IF a=10 AND b=10 THEN PRINT
"Ambas as condicoes sao verdade
iras"
```

Outros dialectos Basic

A versão da Basic para o Spectrum não dispõe de algumas estruturas pensadas para tornar mais fácil a organização dos programas. Por exemplo, alguns microcomputadores utilizam instruções como a seguinte:

1 2 3 4 5

100 ON X GOSUB 200, 300, 400, 500, 600

Trata-se de uma formulação bastante prática que nos permite especificar diferentes subrotinas em função do valor de X. Se X = 1 o programa executa GOSUB 200, se X = 2 executa GOSUB 300, etc. Mas não chegará muito longe se tentar isto num Spectrum!

A forma mais simples de rodear o problema consiste em usar várias instruções do tipo IF X = 1 THEN GOSUB 200, etc., mas existe um modo de formular aquela estrutura:

```
100 GO SUB (100 AND x=1)+(200 A
ND x=2)+(300 AND x=3)....
```

O que se está a fazer aqui é aproveitar a capacidade do Spectrum de distinguir entre o que é verdadeiro e falso. Se dizemos "X = 1" e X não é igual a 1, o Spectrum pensa "O", isto é, «falso». Se pensa "1", é porque X é de facto igual a 1. Ao dizermos (100 AND X = 1), estamos a especificar a necessidade de a segunda condição ser verdadeira para que seja de facto executada a GO SUB 100. Note-se que (100 + (X = 1) significa algo completamente diferente para o computador.

Uma outra construção também não presente no Spectrum, menos útil, é a IF... THEN... ELSE. É usada do seguinte modo:

10 IF A = B THEN PRINT "A é igual a B" ELSE PRINT "A não é igual a B"

Esta instrução é simulada, de uma forma menos elegante, dizendo:

```
  10 IF a=b THEN PRINT "A e' igu
al a B": GO TO 30
  20 PRINT "A e' diferente de B"
  30 ... Continuar programa
```

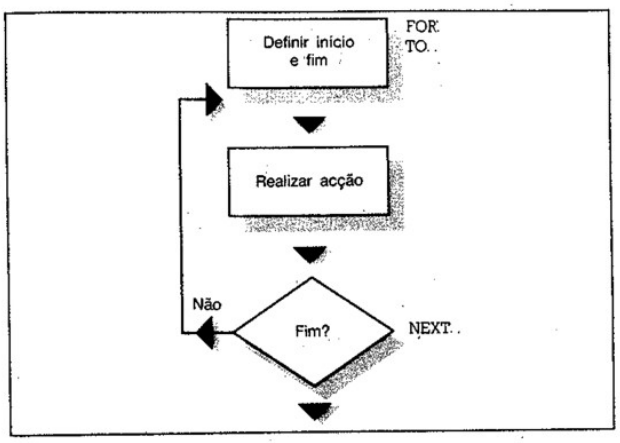
Mas é também possível usar instruções lógicas:

```
 10 PRINT ("A e' igual a B" AND
a=b)+("A e' diferente de B" AND
a<>b)
```

Ciclos

A última estrutura que estudaremos neste capítulo é o ciclo FOR... NEXT, que evita a necessidade de escrever linhas repetidas. Vejamos o seguinte exemplo:

```
10 FOR i=1 TO 20
20 BEEP 1,i
30 NEXT i
```

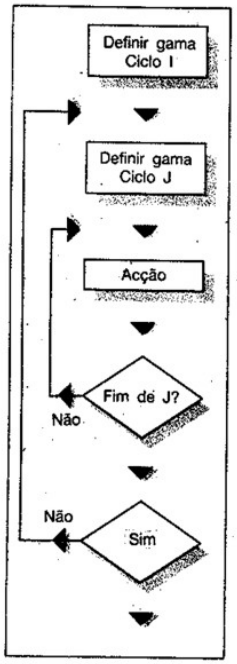


Estamos a usar aqui um simples ciclo FOR... NEXT para fazer uma coisa que de outro modo obrigaria a escrever 20 linhas.

Uma experiência

É muito fácil, e útil, recorrer a ciclos FOR... NEXT em rede, uns dentro dos outros.

```
10 FOR i=1 TO 12
20 PRINT "Tabela de ";i
30 FOR j=1 TO 12
40 PRINT i;"x";j;"=";i*j
50 NEXT j
60 NEXT i
```



Este programa imprimirá tabelas de multiplicação até 12 × 12. Note-se que é necessário terminar o último ciclo primeiro (neste caso o ciclo J), senão confundir-nos-emos a nós próprios. Tente trocar as linhas 50 e 60 para ver o que acontece.

Regra

Tal como acontece no caso das instruções GO SUB, não se deve saltar para fora de um ciclo FOR... NEXT antes de estar terminado. Por exemplo:

```
10 FOR i=1 TO 50
20 IF i=25 THEN GO TO 40
30 NEXT i
40 .... Continuar programa
```

O leitor terá aqui exactamente o mesmo tipo de problemas que teria ao saltar para fora de uma subrotina, pelo que deve usar IF I = 25 THEN LET I = 51: GO TO 30 para sair do ciclo.

RESUMO

Neste capítulo aprendemos:

1. Porque se deve tentar escrever programas estruturados.
2. Como e porquê se devem usar subrotinas (e como não devem ser usadas).
3. Como usar saltos condicionais, e como simular as instruções de dialectos Basic mais estruturados com a Basic do Spectrum.
4. Como usar IF... THEN.
5. Como e porquê usar ciclos para tarefas repetitivas.

2

FUNÇÕES DISPONÍVEIS

Todos os computadores passam o seu tempo a calcular números, e a maior parte dos programas devem manipular números de diferentes maneiras. Estes métodos de manipulação entram em duas categorias principais, a das funções e a dos operadores. Os operadores são coisas como +, −, * e /, e necessitam de um número de cada lado, por exemplo PRINT 2 + 2.

As funções, por outro lado, aceitam um número como argumento e actuam de algum modo sobre ele; por exemplo, a SIN converte o seu argumento para um seno. Produzem sempre um resultado, pelo que ao usá-las deve sempre verificar-se se esse resultado é usado, por exemplo:

```
LET A = INT (10.2)
PRINT SIN (PI)
```

RND

A função RND é uma das mais úteis do computador, se bem que não constitua estritamente uma função na medida em que não trabalha directamente sobre um argumento. Pode ser usada para várias coisas, por exemplo, para a escolha de respostas aleatórias a uma pergunta como a seguinte:

```
  10 INPUT "Como se sente hoje?
";a$
  15 PRINT "Optimo, porque me si
nto ";
  20 LET b=INT (RND*4)+1
  25 GO SUB (40 AND b=1)+(50 AND
 b=2)+(60 AND b=3)+(70 AND b=4)
  30 STOP
  40 PRINT "muito bem,": RETURN
  50 PRINT "bem.": RETURN
  60 PRINT "OK.": RETURN
  70 PRINT "mal.": RETURN
```

Mais simplesmente, podemos imprimir estrelas no visor do seguinte modo:

```
  10 FOR t=0 TO 100
  20 PLOT INT (RND*255)+1,INT (R
ND*176)+1
  30 NEXT
```

Normalmente desejaremos que o resultado desta função seja um número inteiro; como atrás, poderá ser usada a forma

[INT (RND*R) + 1,]

onde R é a gama entre 1 e R. Pode ser um pouco aborrecido escrever isto se for necessário recorrer a esta função muitas vezes. Uma alternativa consistirá em definir uma função que produza o resultado desejado, isto é:

[DEF FN (R) = INT (RND*R) + 1]

Depois de definida num dado programa, esta função pode ser usada para produzir números inteiros aleatórios do seguinte modo:

[PRINT FN R(6)]

dando neste caso um número entre 1 e 6, isto é, sublinhando o lançamento de um dado, ou sob a forma

[LET CARTA = FN R (13)]

Um aspecto a notar sobre esta função é que não é verdadeiramente aleatória dado que o computador começa a funcionar, ao ser ligado, com valores bem definidos em todo o lado. Se desligarmos e ligarmos a máquina, imprimindo RND em seguida obtemos sempre o mesmo valor.

A RND produz aquilo que é conhecido pelo nome de números pseudo-aleatórios, criados a partir de um número «semente» e realizando uma série de operações sobre ele recorrendo a uma fórmula contida na máquina. O resultado é depois apresentado como sendo produzido pela RND; e passa igualmente a «semente» a partir da qual será calculado o seguinte valor «aleatório». Isto pode ser exemplificado usando o programa que se segue, que produz números aleatórios entre 0 e 6 usando uma fórmula muito simples:

```
  10 INPUT "Origem=";a
  20 LET a=(a*75)+19200
  40 LET a=a-INT (a/256)-INT (a/
256)
  45 LET a=a-(INT (a/65536)*6553
6)
  50 PRINT INT (a/10000)
  60 GO TO 20
```

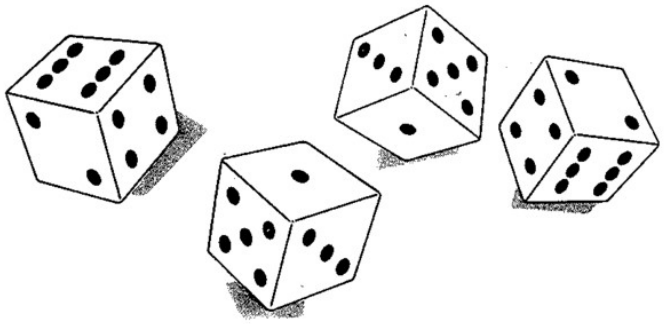
A função RND incorporada no Spectrum é bastante mais eficaz e complexa do que isto, produzindo portanto uma série mais comprida. Acabará no entanto por se repetir sempre a si mesma.

A «semente» pode ser definida no Spectrum usando a instrução RANDOMIZE, e se experimentarmos:

```
10 RANDOMIZE 1
20 PRINT RND
30 RANDOMIZE 1
40 PRINT RND
```

verificaremos que a instrução em causa leva a função RND a partir sempre do mesmo valor na série de números pseudo-aleatórios. Isto pode constituir uma vantagem ou uma desvantagem, conforme a fase do programa. Se se está a testar um programa que utiliza números aleatórios, é útil levar a RND a partir

sempre do mesmo ponto da série a fim de permitir a verificação de resultados. No entanto, depois de o programa estar terminado, o uso sistemático de uma dada semente leva os resultados a serem previsíveis, não havendo portanto qualquer interesse em usar a função RND. A forma de ultrapassar isto consiste em redefinir a RANDOMIZE no início do programa. Isto é de facto muito mais simples do que parece dado que existem algumas posições de memória no interior do Spectrum que se alteram demasiado depressa para poderem ser previsíveis, tornando-se do ponto de vista prático aleatórias. Uma destas posições é a numerada 23 762, que indica o número de imagens enviadas para a televisão, alterado em cada 20 ms. Se a primeira linha do programa for RANDOMIZE PEEK 23 672, a probabilidade de acertar sempre no mesmo número é de 1 em 256, o que produz um valor muito mais aleatório do que o obtido através de outros métodos.



INT

Na secção sobre a RND usámos a INT para obter números inteiros a partir de decimais com:

[PRINT INT (RND*10) + 1]

A função INT é usada para alterar o formato do número e dividi-lo em duas metades, uma de cada lado da vírgula decimal, apresentando como resultado o lado esquerdo, inteiro:

[PRINT INT (123.456)]

imprime 123. Convém notar que esta função arredonda o número para menos, de tal modo que INT (23.999) produz apenas 23 e não 24 como se poderia esperar. Se se pretende arredondar da forma habitual, isto é, para cima no caso de a parte fraccionária do número ultrapassar .5, deve somar-se 0,5 ao número antes de usar a função INT sobre ele: experimente o seguinte:

```
10 INPUT "Preco=";a
20 PRINT "Arredonda para: ";
30 GO TO 10
```

Este método é bastante útil quando se tratam quantias em dinheiro, dado que muitas vezes estas são arredondadas para o valor seguinte.

ABS e SGN

Uma outra função que altera o formato de um número é a ABS. É usada para retirar ao número o sinal que se encontra à sua frente, tornando-o positivo independentemente do que era antes. Nestas condições, ABS (−2) dará 2, o mesmo acontecendo a ABS (2). Esta função é útil para diversas coisas, por exemplo, para verificar que não são usados valores negativos para definir posições de impressão no visor, o que provocaria a paragem do programa com uma mensagem de erro. É ainda mais útil para obrigar ao uso de uma tecla para um qualquer efeito:

```
 10 LET t=1
 20 IF t=1 THEN PRINT "Largue o
botao"
 30 IF t=0 THEN PRINT "Carregue
no botao"
 40 LET a$=INKEY$: IF a$="" THE
N GO TO 40
 45 PAUSE 0
 50 LET t=ABS (t-1)
 60 GO TO 20
```

A SGN é uma função muito semelhante à ABS. Produz o resultado + 1 para qualquer valor positivo e − 1 para qualquer valor negativo. No caso de o número ser zero, a função SGN produz também zero, sendo incapaz de decidir se 0 é positivo ou negativo... Como exemplo, experimente:

```
  10 INPUT a
  20 PRINT a;" e' ";
  30 IF SGN (a)=-1 THEN PRINT "n
egativo.": GO TO 60
  40 IF SGN (a)=1 THEN PRINT "po
sitivo.": GO TO 60
  50 PRINT "zero."
  60 GO TO 10
```

Ou ainda:

```
  10 INPUT a
  20 PRINT a;" e' ";
  30 PRINT ("negativo." AND SGN
(a)=-1)+("positivo." AND SGN (a)
=1)+("zero." AND SGN (a)=0)
  60 GO TO 10
```

SIN, COS, TAN, ASN, ACS, ATN

São estas as funções trigonométricas usadas para tratamento de ângulos. Cabem em duas categorias opostas, dado que parte delas (por exemplo ASN) são o contrário das outras (neste caso, de SIN). Por exemplo:

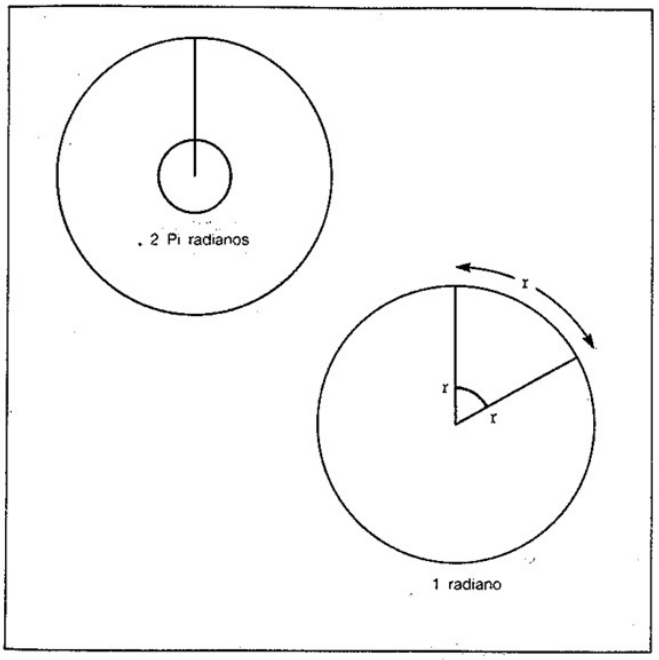
```
10 LET a=SIN (2,5)
20 LET b=ASN (a)
30 PRINT a;" e' o seno de ";b
```

Assim:

ASN é o oposto de SIN, ou o arco cujo seno é...
ACS é o oposto de COS, ou o arco cujo co-seno é...
ATN é o oposto de TAN, ou o arco cuja tangente é...

A maior parte das pessoas estão habituadas a trabalhar

com ângulos entre 0 e 360 graus, mas o Spectrum (e a maior parte dos outros computadores) usa a forma matemática de divisão do círculo em 2 PI radianos.

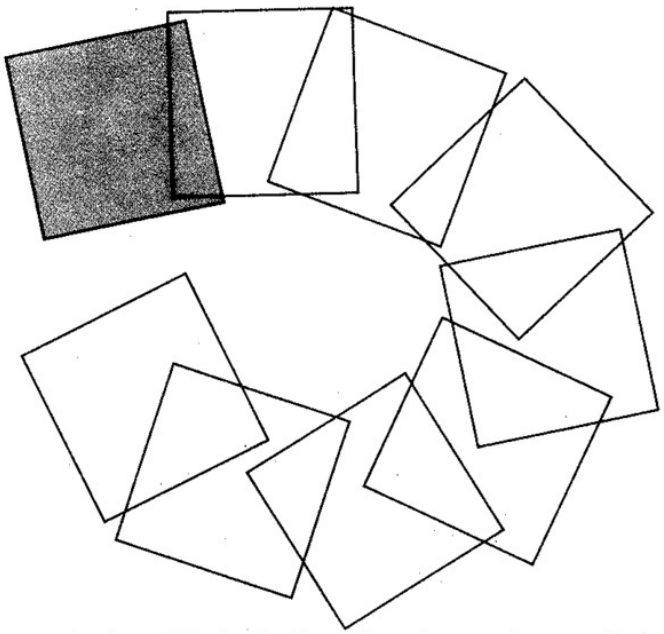


O Spectrum é melhor nesta conversão do que muitos outros computadores porque já inclui o valor PI como número previamente definido, facilitando bastante as conversões.

A SIN, COS e a TAN são definidas usando um triângulo rectângulo, e tornam-se bastante úteis para tratamento de gráficos. Por exemplo, suponhamos que se pretende desenhar um quadrado no visor com o seguinte programa:

```
 10 LET x=20: LET y=20
100 PLOT x,y
```

```
110 DRAW 20,0
120 DRAW 0,-20
130 DRAW -20,0
140 DRAW 0,20
```



e se deseja rodá-lo de 45º de modo a obter um losango. Pode usar-se SIN e COS para calcular as novas posições das linhas. Alteremos o programa dos quadrados do seguinte modo:

```
  10 LET x=40: LET y=40
  15 INPUT "Angulo=";a
  16 LET a=FN r(a)
  20 GO SUB 100: REM desenhar
quadrado
  30 STOP
  40 DEF FN r(a)=a*PI/180
 100 PLOT x,y
 110 DRAW 20*SIN (a),20*COS (a)
```



```
 120 DRAW 20*SIN (a+(PI/2)),20*C
OS (a+(PI/2))
 130 DRAW 20*SIN (a+PI),20*COS (
a+PI)
 140 DRAW 20*SIN (a+(3*PI/2)),20
*COS (a+(3*PI/2))
 150 RETURN
```

Acrescentando e alterando as linhas que se seguem obteremos um padrão agradável.

```
 15 FOR t=0 TO 360 STEP 10
 16 LET a=FN r(t)
 25 NEXT t
 30 STOP
```

Os resultados de todas estas funções podem ser facilmente ilustrados usando gráficos, o mais fácil dos quais é o da função SIN. Experimente o seguinte:

```
 10 FOR t=0 TO 2*PI STEP .1
 20 PLOT t*10,SIN (t)*10+50
 30 NEXT t
```

Este programa apresenta a forma de uma onda sinusoidal, e se substituírmos SIN por COS ou até RND teremos uma melhor ideia daquilo que estas funções fazem. Pode usar-se o programa seguinte para apresentar no visor as diversas funções, e até misturas delas, como, por exemplo, SIN (1 − SIN (3*T)).

```
  5 GO SUB 100: REM   desenhar
eixos
  10 INPUT "Indique funcao: ";a$
  20 FOR t=0 TO 2*PI STEP .1
  30 PLOT t*23,VAL a$*50+50
  40 NEXT t
  50 GO TO 10
  99 REM desenhar eixos
 100 PLOT 0,0
 110 DRAW 0,100
 120 PLOT 0,50
 130 DRAW 150,0
 140 RETURN
```

Note-se que quando se indicam estas funções no programa se deve usar a tecla apropriada, onde a respectiva abreviatura se encontra gravada.

Conversão de graus para radianos

Como 2 PI radianos é igual a 360 graus, para converter graus para radianos basta multiplicar por PI/180 ou definir a função:

```
DEF FN r(A) = A * PI/180
```

Para realizar a conversão inversa, deve multiplicar-se por 180/PI ou usar a função:

```
DEF FN d(A) = A * 180/PI
```

RESUMO

Neste capítulo estudámos:

1. O modo como funciona a RND, e como nos é possível definir uma função que nos permita obter uma determinada gama de valores.
2. A forma como actua a função INT, e quando a devemos usar.
3. O modo de funcionamento das funções ABS e SGN, e qual a respectiva utilidade.
4. Como funcionam as funções SIN, COS, TAN, ASN, ACS e ATN e o modo de as usar na produção de gráficos.
5. Como programar algumas funções definidas pelo utilizador a fim de converter graus para radianos e vice-versa.

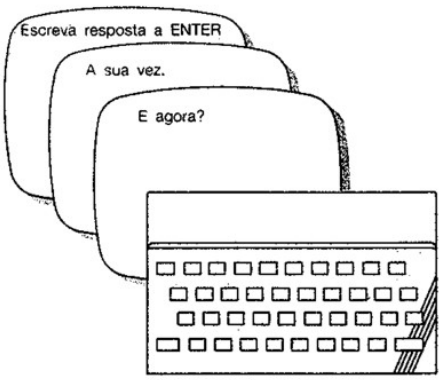
PROJECTOS

1. Use o último programa de construção de gráficos com uma mistura de funções incluindo, por exemplo, a ABS e a SGN para verificar como actuam. Veja se consegue melhorar a apresentação dos eixos dos gráficos atribuindo-lhes etiquetas apropriadas.
2. Tente escrever o seu próprio programa gerador de números aleatórios, capaz de produzir uma extensa sequência de números.

3

PROGRAMAÇÃO INTERACTIVA

Quando falamos de «programação interactiva», referimo-nos a algo bastante simples – como é possível a cada um de nós trabalhar em conjunto com o seu Spectrum. Podemos conceber um programa praticamente não interactivo, por exemplo, um programa de demonstração de gráficos que se limite a executar um ciclo apresentando uma série de imagens sem necessidade de qualquer interferência do utilizador, mas na maior parte dos casos será necessário carregar fisicamente em teclas e dar ao computador alguma informação relativa a diversas passagens do programa.



Portanto a «programação interactiva» trata da maneira como se apresenta a informação ao computador, e como este nos apresenta os resultados, no visor ou numa impressora. Naturalmente, se quisermos que os programas sejam tão úteis quanto possível, desejaremos igualmente apresentar a informação do computador de uma forma tão clara quanto possível, tal como desejaremos enviar dados à máquina da forma mais simples possível. Este capítulo foi precisamente pensado para nos ajudar nisto.

Porque procedemos deste modo? Certamente que quem programa não é idiota, mas se algum dia quiser vender os seus programas desejará que sejam tão pouco dependentes de quem os utilize quanto possível. Consideremos, por exemplo, as linhas que se seguem:

```
  10 INPUT "Indique um numero: "
;a
  20 GO TO 10
```

Dizemos ao utilizador que escreva um número, e se for escrito 1, 2, 3, etc., tudo correrá bem. Mas aquilo que nós entendemos como número e o que o computador entende pode não ser a mesma coisa. Se escrevermos «um», veremos que o computador não nos compreende. E convirá recordar que apesar de nós considerarmos 1000 como um número, o computador entende que não o é, apenas aceitando 1000. No caso aqui apresentado estaremos apenas a estragar um programa de duas linhas, mas suponhamos que estas fazem parte de um programa mais vasto, contendo 100 pessoas e respectivas moradas, e que acabámos de escrever 100 destas moradas, mas que nos enganamos e carregamos na tecla errada – desagradável, não é?

Como obter a informação

O Spectrum oferece-nos duas maneiras de obtermos informação de um utilizador, recorrendo às instruções INPUT e INKEY$. Alguns outros computadores dispõem ainda da instrução GET, ou GET$, que é usada no formato GET A ou GET A$, e que indica ao computador que deve esperar pelo

toque numa tecla. É semelhante à INKEY$, mas esta não espera que o utilizador carregue numa tecla.

Podemos no entanto simular esta instrução do seguinte modo:

```
10 IF INKEY$="" THEN GO TO 10
```

O Spectrum passará assim a esperar nesta linha até carregarmos numa tecla, passando depois à linha seguinte.

Mas que acontece então ao nosso pequeno problema da INPUT, atrás mencionado? Perguntemos primeiro a nós próprios o que leva o Spectrum a imprimir uma mensagem de erro quando escrevemos «um». O que acontece é que estamos a escrever uma cadeia alfabética quando a máquina espera um número. Devemos portanto prepararmo-nos para o pior caso possível, o que significa escrevermos programas que possam aceitar a escrita de cadeias alfabéticas sem pararem.

Experimentemos o seguinte:

```
  10 INPUT "Indique um numero en
tre 0 e 9";a$
  20 IF CODE a$<48 OR CODE a$>57
 THEN BEEP .5,1: GO TO 10
  30 LET a=VAL a$
```

O que este pequeno programa faz é aceitar qualquer cadeia escrita pelo utilizador, verificar o valor do seu código, e passar apenas à linha 30 se este valor se encontrar entre 48 e 57. Os códigos 48 a 57 são evidentemente os correspondentes aos algarismos 0 a 9.

Este método poderia ser igualmente empregue para números maiores, mas o problema é que a função CODE só indica o valor do primeiro caracter de uma cadeia. Se escrevermos por exemplo 2c32 o programa lerá apenas o código de 2, e o problema subsistirá ainda para os outros caracteres. É portanto necessário levar o Spectrum a verificar individualmente todos os caracteres de A$.

Um modo de o fazer consiste em programar um ciclo que verifique todos os caracteres:

```
  10 INPUT "Indique um numero en
tre 0 e 9";a$
```

```
  20 FOR n=1 TO LEN a$
  30 IF CODE a$(n TO n)<48 OR CO
DE a$(n TO n)>57 THEN BEEP .5,1:
 GO TO 10
  40 NEXT n
  50 LET a=VAL a$
```

Passamos assim a dispor de uma rotina que verifica todos os caracteres escritos, um a um, e faz soar um petulante BEEP no caso de cometermos um erro. A expressão A$(N TO N) é o modo descritivo habitual no Spectrum para especificar um caracter individual de uma cadeia, pelo que no caso de N ser 5 estaremos de facto a referirmo-nos a A$(5 TO 5), isto é, ao quinto caracter da cadeia. A$(5 TO 6) especifica o quinto e o sexto caracteres, etc.

Poderemos começar por dimensionar a cadeia com DIM A$(X), o que terá aliás a vantagem de permitir um controlo imediato da dimensão do número (ou daquilo que virá a ser considerado como tal) através da limitação das entradas ao comprimento máximo X. Neste caso é ainda possível fazer um programa mais à prova de «bomba» verificando os possíveis erros por comparação do comprimento da cadeia escrita com N, e voltando novamente à linha 10 no caso de o comprimento ser excessivo.

Estamos portanto a pensar em formas de apanhar os erros antes de estes provocarem uma paragem de execução. Obviamente, não será possível apanhá-los todos quando estamos a escrever o programa, mas se pensarmos no que estamos a fazer, e actualizarmos os nossos programas quando passarmos para um novo problema, acabaremos por obter um produto acabado com uma aparência muito mais profissional.

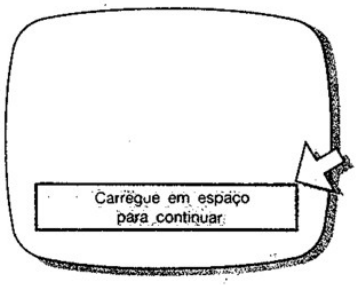
Regra

Quando escrevemos programas devemos tentar não misturar excessivamente as instruções INPUT e INKEY$. Obviamente, no caso de o programa estar a pedir um número ou um nome de ficheiro (ambos com mais de um caracter), só a INPUT nos poderá ser útil; mas de qualquer modo a mistura de ambas as instruções é confusa, dado que o utilizador tenderá a carregar em «Enter» enquanto o programa está a executar.

Uma experiência

É possível determinar os erros das entradas definindo como ilegais todos os caracteres desejados. Para tal, deve consultar-se os códigos apresentados no Apêndice A do Manual Spectrum. Basta-nos então usar um método semelhante ao do programa seguinte para eliminarmos todos os caracteres cujos códigos se encontram fora da gama escolhida:

```
1000 PAUSE 0
1010 IF INKEY$=CHR$ (13) OR INKE
Y$=CHR$ (32) THEN GO TO 1030
1020 IF CODE INKEY$<65 OR (CODE
INKEY$>90 AND CODE INKEY$<97) OR
 CODE INKEY$>122 THEN GO TO 1000
1030 PRINT INKEY$
1040 GO TO 1000
```



Se se executar este programa, será fácil verificar que actua um pouco como uma máquina de escrever. Considerámos ilegais todos os caracteres excepto as letras do alfabeto (maiúsculas e minúsculas). Aliás tratar-se-á de uma má máquina de escrever, dado que não nos permite usar qualquer sinal de pontuação.

Se o leitor verificar os códigos indicados no Manual, descobrirá que CHR$ 32 equivale a um espaço, e CHR$ 13 a «Enter». Portanto, na linha 1010 estaremos a verificar se a tecla premida é uma destas duas, e se assim for a obrigar o computador a passar sobre a linha seguinte, que de outro modo as consideraria ilegais. O leitor pode eliminar a linha 1010 para verificar o que foi dito.

Outros pontos de interesse são a instrução PAUSE na linha 1000, que estamos a usar a fim de evitar a repetição das teclas carregadas, e o separador da linha 1030, que garante que o caracter seguinte seja impresso imediatamente a seguir ao anterior.

RESUMO

Neste capítulo aprendemos:

1. O que é a programação interactiva.
2. As diferenças entre a INPUT e a INKEY$
3. Como usar CODE para verificar se o utilizador de um programa está a usar as teclas correctas ou não.

PROJECTOS

1. Escreva uma rotina curta que aceite nomes e endereços, excluindo uma entrada numérica na primeira linha (onde deve encontrar-se o nome). Poderá também exigir que uma dada parte da linha seguinte seja numérica, se bem que deva ter em conta a possibilidade de o endereço em causa não conter número de polícia, que será então substituído por um símbolo especial, por exemplo, um "=". Tente construir o programa de tal modo que seja impossível pará-lo com entradas incorrectas.
2. Reescreva o programa incipiente de tratamento de texto já apresentado de modo a poder admitir igualmente sinais de pontuação e números.

4

TRATAMENTO DA INFORMAÇÃO

O manuseamento de informação pode não parecer muito interessante, mas se pensarmos um pouco no assunto verificaremos que é crucial para nós podermos escrever programas que sejam interessantes no nosso Spectrum. De facto, que faz um computador? Guarda informação sob a forma de números, de tal modo que quando lhe dizemos para fazer alguma coisa consulta a informação que possui e actua de acordo com ela.

E não interessa sequer que a informação tratada seja um invasor espacial ou a nossa conta bancária – do ponto de vista do Spectrum estes dois tipos de informação são precisamente equivalentes. O que de facto interessa é a eficácia da forma como dizemos ao Spectrum que deve tratar a informação.

Blocos de notas

Os computadores que utilizam a linguagem Basic dispõem de diversos modos de armazenamento da informação, sendo o mais óbvio de todos o recurso a instruções DATA. Uma instrução DATA é essencialmente apenas uma lista de números ou letras que incluimos no final de um programa:

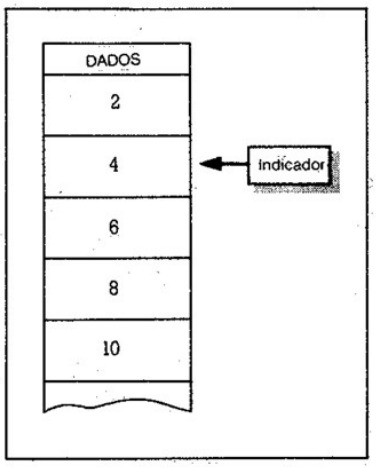
```
10 FOR a=1 TO 10
20 READ b
30 PRINT a;" vezes dois e' ";b
40 NEXT a
50 DATA 2,4,6,8,10,12,14,16,18
,20
```

Talvez isto pareça ao leitor um programa estúpido, particularmente quando se conhece a tabela de multiplicação por dois e se conhece também que:

```
10 FOR a=1 TO 10
20 PRINT a;" vezes dois e' ";a
#2
30 NEXT a
```

constitui um modo mais simples de fazer a mesma coisa. Mas pensemos um pouco melhor no assunto – como já dissemos, o leitor conhece a tabela de multiplicação por dois, e portanto para quê ter de a descobrir sempre que se quer usá-la? As declarações DATA, como é fácil ver, são listas de informações que já possuímos, e muitas vezes é mais rápido para um programa encontrar um valor numa lista do que ter de o calcular.

O que acontece na nossa versão com declarações DATA é que a linha 10 conta dez valores diferentes para A, e para cada valor lê B uma vez. READ B significa que o programa deve ler o valor de ordem A (isto é, o primeiro, o segundo, o terceiro, etc.) de DATA, atribuindo assim a B um valor diferente de cada vez. Depois restará ao programa fazer a B o que tivermos dito para fazer.



Um aspecto a ter presente sobre as declarações DATA é que apenas é possível ler (READ) estas uma vez, a menos que se use a instrução RESTORE. No caso anterior, diríamos RESTORE 50 (isto é, reactivar os dados a partir da linha 50) para ser possível lê-los uma segunda vez, a partir de uma outra linha READ, por exemplo. A instrução RESTORE altera algo a que se chama «indicador de DATA», usado para definir o ponto onde nos encontramos na leitura dos dados, e que guarda o valor do último dado lido. Podemos utilizar tantos ou tão poucos DATA quantos quisermos em cada linha – podemos escrever por exemplo:

```
50 DATA 2,4,6,8,10
60 DATA 12,14,16,18,20
```

o que é perfeitamente equivalente à forma usada na listagem anterior.

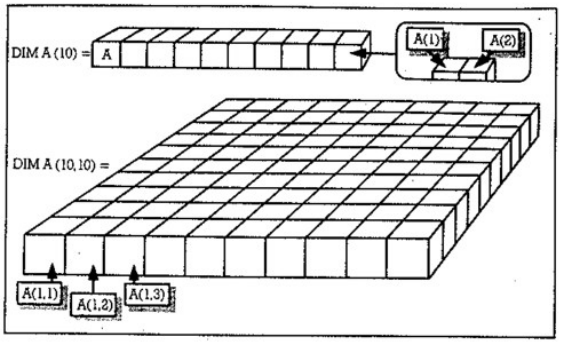
Sabemos que READ e DATA são úteis para guardar a informação que já conhecemos, como os gráficos definidos pelo utilizador ou as notas de passagens musicais; mas como poderemos armazenar a informação que não conhecemos? Nesse caso teremos que procurar outro método.

Uso dos quadros

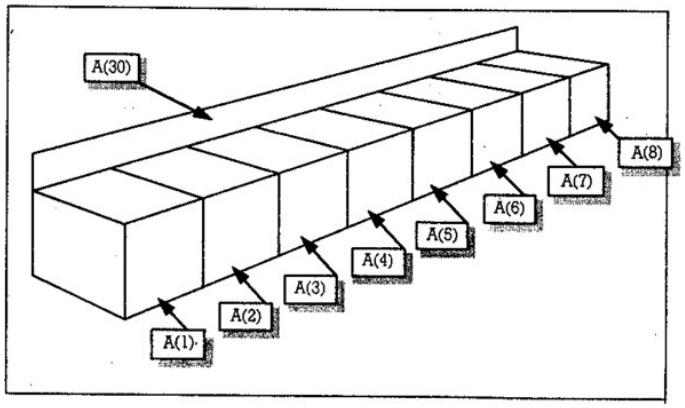
Observámos já o nosso «bloco de notas» e descobrimos as suas limitações, mas felizmente o Spectrum dispõe também de um ficheiro próprio! O modo mais simples de pensar num quadro *(array)* consiste em imaginá-lo como uma rede de caixas, ou uma tabela, sendo-nos permitido decidir quantas caixas possui.

Define-se o número de caixas usando a declaração DIM, e se observarmos a ilustração da página seguinte veremos como isto é feito. Ao dizermos DIM A(10) estamos a reservar um espaço suficiente para guardar dez elementos de informação, que depois podemos designar por A(1) a A(10). Dizendo A(10,10), estamos a reservar espaço para 10×10, isto é, 100 elementos de informação, desde A(1,1) até A(10,10). Podemos imaginar um quadro 10×10×10, obtido com a instrução DIM A(10,10,10), e é até possível construir quadros com quatro ou

mais dimensões – tendo como único limite a quantidade de memória disponível no Spectrum.



Podemos pensar no assunto do seguinte modo: DIM A(30) indica ao Spectrum que deve construir uma caixa grande com 30 caixas mais pequenas no seu interior, designadas por A(1), A(2), etc. Estas caixas menores são consideradas «indiciadas» – talvez isto lhe permita entender o que o Spectrum quer dizer quando imprime a mensagem de erro «subscript wrong» – «índice errado».



Organização

Até agora tratámos apenas de quadros contendo números, isto é, quadros numéricos, mas é possível guardar dados alfanuméricos (incluindo gráficos definidos pelo utilizador) em quadros do mesmo tipo (alfanuméricos, ou de cadeias). A única diferença relativamente aos anteriores é que estes são definidos por uma instrução DIM ligeiramente diferente: DIM A$(X).

Suponhamos que queremos manter um registo de um conjunto de dados, como endereços, pormenores sobre a nossa colecção de discos, etc. A informação pode ser dividida num certo número de secções diferentes que podem ser tratadas com facilidade, cada uma das quais guardando uma informação diferente. Estas secções são designadas por «campos», e a colecção de campos recebe o nome de «registo». Pode dizer-se que um conjunto de registos constitui um ficheiro ou base de dados.

Os requisitos de um programa para tratamento de uma base de dados são:

1. Permitir a entrada e alteração de registos
2. Dispor de um mec. para armazenamento permanente da b. de dados
3. Permitir a consulta de um dado registo através da espec. de um campo
4. Qualquer outra coisa que o utilizador necessite!

Uma experiência

Quando se está a conceber um programa deste tipo devemos antes do mais definir como deverão ser guardados os dados, quais os quadros necessários, etc. Por exemplo, consideremos uma colecção de discos – é necessário reservar espaço para guardar os títulos, os artistas, as composições, e a data de gravação. Tudo isto deve ser definido logo de início, apesar de ser fácil introduzir alterações mais tarde em caso de necessidade.

Comecemos então pela seguinte parte do programa:

```
  10 REM definir quadros
  20 DIM t$(30,15): REM 30 disco
s com titulos em t$
  30 DIM b$(30,15): REM artistas
  40 DIM r$(30,14,15): REM 14 pi
stas em cada disco
  50 DIM d$(30,8)
  60 DIM p$(15): REM titulo do f
icheiro
  65 DIM s$(15)
  70 LET ptr=1: REM indicador do
 disco actual
```

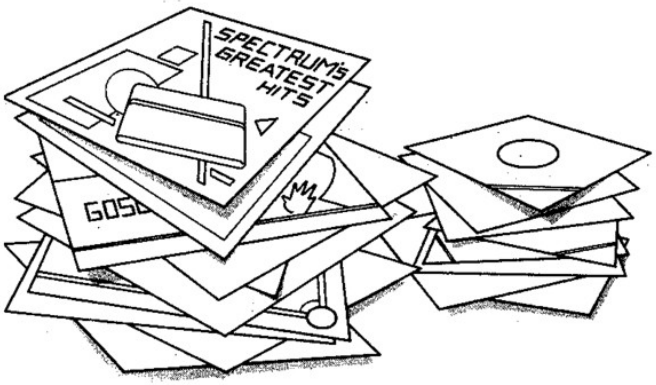
Em seguida teremos de definir um menú que nos dê acesso a todas as opções relevantes:

```
  99 REM menu
 100 CLS
 110 PRINT " FICHEIRO DE ";p$
 120 PRINT
 130 PRINT " 1...Alterar dados"
 140 PRINT " 2...Procurar dados"
 150 PRINT " 3...Carregar dados"
 160 PRINT " 4...Gravar dados"
 170 PRINT " 0...Terminar"
 175 PRINT "O disco actual e': "
;ptr
 180 LET a$=INKEY$
 190 IF a$="1" THEN GO SUB alter
ar: GO TO 100
 200 IF a$="2" THEN GO SUB procu
rar: GO TO 100
 210 IF a$="3" THEN GO SUB carre
gar: GO TO 100
 220 IF a$="4" THEN GO SUB grava
r: GO TO 100
 230 IF a$="0" THEN GO SUB fim:
GO TO 100
 240 GO TO 180
```

Note a forma como estamos a escrever o programa – até agora decidimos o que queremos fazer, reservámos espaço para os quadros necessários, e depois considerámos a questão logicamente apresentando ao utilizador uma série de opções, todas elas tratadas por uma instrução GOSUB. Por enquanto o programa ainda não faz nada, pelo que convirá começarmos a escrever as subrotinas!

Construção do programa

Uma vantagem do Spectrum é a sua capacidade de usar instruções como GOSUB A, onde A indica um dado valor numérico – o número de uma linha. Na secção que se segue veremos o interesse disto, por nos permitir designar as subrotinas recorrendo a mnemónicas muito práticas. Todas estas subrotinas poderão ser escritas agora, deixando-se para mais tarde a definição dessas mnemónicas. Por exemplo, iniciaremos a secção ALTERAR na linha 300:




```
 299 REM alterar comeca em 300
 300 CLS
 310 PRINT "Menu de alteracoes"
 320 PRINT
 330 PRINT " 1...Escolher disco"
 340 PRINT " 2...Alterar disco a
ctual"
 350 PRINT " 3...Avancar um disc
o"
 360 PRINT " 4...Atrasar um disc
o"
 370 PRINT " 0...Menu principal"
 375 PRINT " O disco actual e' "
;ptr
 380 LET a$=INKEY$
 390 IF a$="1" THEN GO SUB escol
her: GO TO 300
```

```
 400 IF a$="2" THEN GO SUB altdi
s: GO TO 300
 410 IF a$="3" THEN GO SUB avanc
ar: GO TO 300
 420 IF a$="4" THEN GO SUB atras
ar: GO TO 300
 430 IF a$="0" THEN RETURN
 440 GO TO 380
```

O leitor já terá compreendido que uma boa maneira de conceber programas deste tipo consiste em escrever primeiro os menús apropriados.

Consideremos agora o tipo de alterações mais simples:

```
 499 REM Avancar (500), Incremen
tar indicador de disco
 500 LET ptr=ptr+(ptr<31)
 510 RETURN
 519 REM Atrasar (520), Decremen
tar indicador de disco
 520 LET ptr=ptr-(ptr>0)
 530 RETURN
 539 REM Escolher (540) um disco
 pelo numero
 540 CLS
 550 INPUT "Indique numero do di
sco ";a$
 560 IF VAL a$<1 OR VAL a$>30 TH
EN RETURN
 570 LET ptr=VAL a$
 580 RETURN
```

Tratemos agora da secção, um pouco mais complexa, que modifica os campos. Aqui seria bom dispor de uma subrotina que imprimisse o registo no visor, e é precisamente disso que GOSUB 1000 trata:

```
 599 REM Alterar campos
 600 GO SUB 1000: REM imprimir d
isco
 610 INPUT "Indique letra ou num
ero do campoa alterar, X termina
, ";a$
 615 IF a$="*" THEN INPUT "Novo
titulo do ficheiro: ",p$
 620 IF a$="A" THEN INPUT "Novo
titulo: ";t$(ptr): GO TO 600
 630 IF a$="B" THEN INPUT "Nova
data: ";d$(ptr): GO TO 600
```



```
 640 IF a$="C" THEN INPUT "Novo
artista: ";b$(ptr)
 650 IF a$="X" THEN RETURN
 655 IF CODE a$>65 THEN GO TO 60
0
 660 IF VAL a$<1 OR VAL a$>14 TH
EN GO TO 600
 670 PRINT "Nova pista ";VAL a$;
":";
 680 INPUT r$(ptr,VAL a$)
 690 GO TO 600
```

Apresentação dos discos

```
1000 CLS
1005 PRINT "*,FICHEIRO DE: ";p$
1006 PRINT "Numero de registo: "
;ptr
1010 PRINT "A,Titulo: ";t$(ptr)
1020 PRINT "B,Data: ";d$(ptr)
1030 PRINT "C,Artista: ";b$(ptr)
1040 FOR i=1 TO 20 STEP 2
1050 PRINT i;" ";r$(ptr,i);" ";i
+1;" ";r$(ptr,i+1)
1060 NEXT i
1070 RETURN
```

Depois disto poderemos tratar de algumas outras opções simples contidas no mesmo menú:

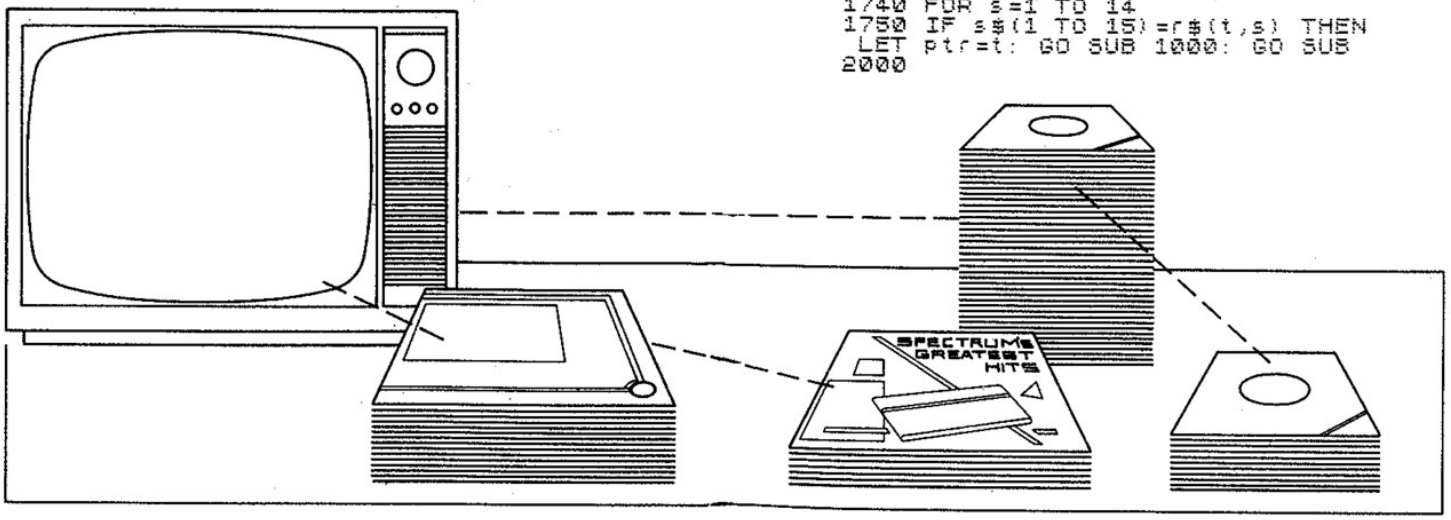
```
 799 REM gravar p$(),t$(),b$(),r
$(),d$()
 800 INPUT "Indique nome do fich
eiro: ",f$
 810 SAVE f$+"p" DATA p$()
 820 SAVE f$+"t" DATA t$()
 830 SAVE f$+"b" DATA b$()
 840 SAVE f$+"r" DATA r$()
 850 SAVE f$+"d" DATA d$()
 860 RETURN
 899 REM carregar p$(),t$(),b$()
 r$(),d$()
 900 INPUT "Indique nome do fich
eiro: ",f$
 910 LOAD f$+"p" DATA p$()
 920 LOAD f$+"t" DATA t$()
 930 LOAD f$+"b" DATA b$()
 940 LOAD f$+"r" DATA r$()
 950 LOAD f$+"d" DATA d$()
 960 RETURN
```

Tudo isto poderia ser facilmente convertido para Microdrive prefixando os nomes de ficheiros pela expressão *"m";1; obteríamos assim um sistema de ficheiros muito rápido.

À procura da palavra perdida

A última secção do programa é a rotina de «procura», que descobre no ficheiro um dado campo e apresenta depois no visor o registo, ou registos, que o contêm. É aqui que a rotina de apresentação visual dos registos se torna muito útil.

```
1499 REM procura
1500 CLS
1510 LET ptr=1
1520 PRINT "PROCURA"
1530 PRINT
1540 INPUT "Indique cadeia a pro
curar",a$
1550 LET s$=a$
1559 REM procurar titulos
1560 GO SUB 2100
1570 FOR t=1 TO 30
1580 IF s$(1 TO 15)=t$(t) THEN L
ET ptr=t: GO SUB 1000: GO SUB 20
00
1590 NEXT t
1599 REM procurar datas
1600 LET s$=a$
1610 IF LEN s$<8 THEN LET s$=s$+
" ": GO TO 1610
1620 FOR t=1 TO 30
1630 IF s$(1 TO 8)=d$(t) THEN LE
T ptr=t: GO SUB 1000: GO SUB 200
0
1640 NEXT t
1659 REM procurar artistas
1660 LET s$=a$
1670 GO SUB 2100
1680 FOR t=1 TO 30
1690 IF s$(1 TO 15)=a$(t) THEN L
ET ptr=t: GO SUB 1000: GO SUB 20
00
1700 NEXT t
1709 REM procurar pistas
1710 LET s$=a$
1720 GO SUB 2100
1730 FOR t=1 TO 30
1740 FOR s=1 TO 14
1750 IF s$(1 TO 15)=r$(t,s) THEN
 LET ptr=t: GO SUB 1000: GO SUB
2000
```



```
1760 NEXT s: NEXT t
1770 RETURN
2000 PRINT AT 20,0;"Carregue em
SPACE para continuar"
2010 LET c$=INKEY$: IF c$<>" " T
HEN GO TO 2010
2020 RETURN
2099 REM dar a s$ 15 caracteres
2100 IF LEN (s$)<15 THEN LET s$=
s$+" ": GO TO 2100
2110 RETURN
2199 REM final inicia-se em 2200
2200 INPUT "Tem a certeza? ";a$
2210 IF a$<>"S" THEN RETURN
2220 STOP
```

O leitor dispõe agora do rudimento de um ficheiro muito simples, mas será ainda necessário referir um aspecto. Reveja novamente o programa, e defina as variáveis que serão usadas nas instruções GOSUB. Estas variáveis serão indicadas numa nova linha, a linha 10, que apresentamos em seguida. A função «Alterar», por exemplo, inicia-se na linha 300, pelo que escreveremos LET Alterar=300:

```
  10 LET alterar=300: LET procur
ar=1500: LET carregar=900: LET g
ravar=800: LET altdis=600: LET e
scolher=540: LET avancar=500: LE
T atrasar=520: LET fim=2200
```

Dispomos assim de um ficheiro que guarda um máximo de 30 discos, dispondo de campos com um comprimento máximo de 15 caracteres. Pode alterar-se estes números em função da memória disponível, e do tempo que quisermos esperar durante a sua carga a partir de cassetes.

Resumo

Neste capítulo aprendemos:

1. As diferenças entre quadros e instruções DATA, e o modo como poderemos usar ambos estes métodos de armazenamento da informação nos nossos programas.
2. Como usar RESTORE.
3. Como escrever um programa de ficheiros de uma forma lógica, usando uma combinação de GOSUB's e menús.

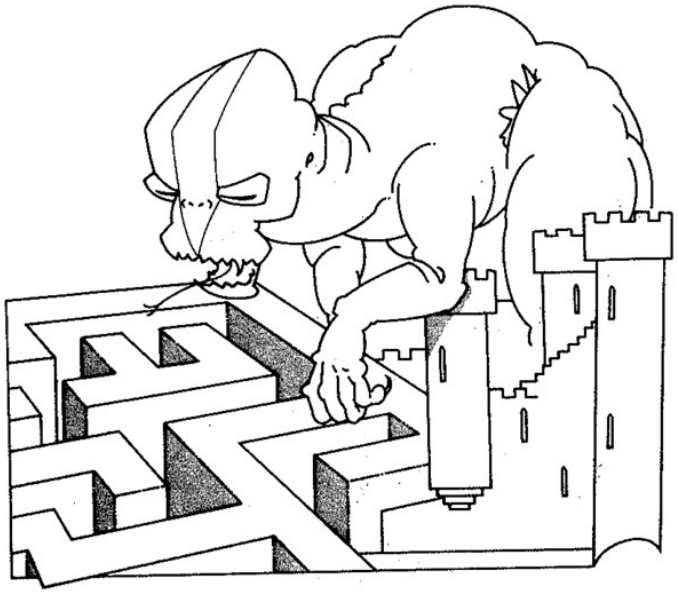
Projectos

Reescreva o programa de ficheiro de modo a este catalogar uma colecção de livros, ou os endereços dos seus amigos.

5

USO DE QUADROS

Um dos usos principais dos quadros, tanto numéricos como alfanuméricos, ocorre em programas de aventuras. Estes permitem ao utilizador mover-se por uma espécie de labirinto recolhendo tesouros e enfrentando vários monstros.



Vejamos como podemos escrever um programa que nos permita definir as salas e ligações entre estas, e mover-nos através delas. Tal como acontece no caso do programa de tratamento de ficheiros, o melhor modo de começar o programa consiste em definir quais os quadros que virão a ser necessários.

Como o programa necessita de um conjunto de descrições das salas, etc., necessitamos de uma cadeira que contenha estas descrições. O número total de posições possíveis é definido pela quantidade de memória de que dispomos. No programa que se segue, definimos este número como igual a 10, usando para tal a variável M. Este número pode ser aumentado até se usar toda a memória. O comprimento da cadeia descritiva é definido por T; também este comprimento pode ser alterado, mas note-se que consome bastante memória. Convirá portanto decidir entre muitas posições de memória com descrições curtas, ou menos com descrições compridas.

Para usar o quadro descritivo necessitamos igualmente de um conjunto de indicadores que permitirão a ligação das salas entre si de diferentes maneiras. Necessitaremos ainda de definir os valores iniciais deste quadro de indicadores. A secção que se segue faz precisamente isto:

```
  6 BORDER 3
  7 INK 3
  8 LET t=30
 10 LET m=10
 20 DIM d$(m,t)
 25 DIM s(m,4)
 30 DIM e(m,4)
 31 LET b$="NAO"
 35 FOR t=1 TO m: FOR s=1 TO 4:
LET e(t,s)=-1: NEXT s: NEXT t
```

Em seguida teremos de programar um qualquer tipo de controlo da execução. Este controlo assumirá a forma de um menú principal, contendo como opções a possibilidade de alterar as descrições, carregar e gravar em fita, ligação das salas de forma aleatória, e de jogo:

```
 40 CLS
 45 DEF FN r(r)=INT (RND*r)+1
 50 PRINT "AVENTURA SIMPLES"
 60 PRINT
```

```
  70 PRINT " 1... Jogar"
  80 PRINT " 2... Alterar"
  90 PRINT " 3... Carregar"
 100 PRINT " 4... Gravar"
 110 PRINT " 5... Ligar salas"
 115 PRINT " 6... Terminar"
 120 LET a$=INKEY$: IF a$="" THE
N GO TO 120
 130 IF a$="1" THEN GO SUB 600:
GO TO 40
 140 IF a$="2" THEN GO SUB 230:
GO TO 40
 150 IF a$="3" THEN GO SUB 900:
GO TO 40
 160 IF a$="4" THEN GO SUB 800:
GO TO 40
 170 IF a$="5" THEN GO SUB 450:
GO TO 40
 180 IF a$="6" THEN GO SUB 200:
GO TO 40
 190 GO TO 120
```

Considerando primeiro a opção mais fácil, definiremos o modo de terminar a execução. Para termos a certeza de que não paramos o programa antes de gravarmos os dados, é boa ideia pedir confirmação ao utilizador antes desta e doutras opções drásticas:

```
200 INPUT "Tem a certeza? ";b$
210 IF b$="SIM" THEN STOP
220 RETURN
```

As secções que se seguem constituem as partes principais do programa. Consideremos primeiro a parte de alterações nas salas. Esta é controlada novamente por um menú, contendo todas as opções apropriadas. O menú é escrito exactamente da mesma forma que o menú principal; notemos que não são usadas declarações REM, a fim de poupar espaço.

```
225 REM Alterar
230 CLS
240 PRINT "MENU DE ALTERACOES"
250 PRINT
260 PRINT "1... Listar locais"
270 PRINT "2... Alterar local"
280 PRINT "3... Menu principal"
290 LET a$=INKEY$
300 IF a$="1" THEN GO SUB 340:
GO TO 230
 310 IF a$="2" THEN GO SUB 410:
GO TO 230
 320 IF a$="3" THEN GO SUB 1000:
 GO TO 230
 330 GO TO 290
```

Considerando estas opções pela ordem em que ocorrem no menú, vejamos uma rotina que lista os locais disponíveis. Como é óbvio, se existir um vasto número de locais é boa ideia programar um qualquer tipo de mecanismo de fuga que nos permita voltar ao menú anterior.

```
335 REM Listar locais
340 PRINT "Carregue em 'S' para
parar"
350 FOR t=1 TO m
360 PRINT t;"==>";d$(t)
365 PRINT "As saidas sao ";
366 FOR s=1 TO 4: PRINT ("Norte
" AND e(t,s)=0 AND s=1);("Sul "
 AND e(t,s)=0 AND s=2);("Este "
AND e(t,s)=0 AND s=3);("Oeste "
AND e(t,s)=0 AND s=4);: NEXT s
 367 PRINT
 370 LET a$=INKEY$
 380 IF a$="S" THEN LET t=m
 390 NEXT t
 400 RETURN
```

Um dos aspectos interessantes da versão da Basic empregue no Spectrum é a possibilidade de executar a instrução de impressão da linha 366 apresentando apenas as palavras apropriadas em função da presença de um zero no quadro E.

Esta especificação das direcções é realizada na secção seguinte. Pede-se ao utilizador o número do local, esperando-se em seguida um conjunto de direcções de saída possíveis. No máximo poderão existir quatro saídas: «Norte», «Sul», «Este» e «Oeste». Podem acrescentar-se direcções como «Subir», «Descer», etc, alterando a instrução DIM das linhas 25 e 30 de modo a dispor do espaço necessário nos quadros E e S.

```
405 REM Alterar local
410 INPUT "Indique local a alte
rar ";l
 420 PRINT d$(l)
```

```
430 INPUT "==>";d$(l)
431 INPUT "Indique saidas N/S/E
/O ";a$
```

Depois de dar entrada à informação, não é necessário usá-la toda. A secção que se segue verifica as entradas e coloca-se nas posições adequadas do quadro E, sendo Norte E (T, 1), Sul E (T,2), e Este e Oeste E (T,3) e E (T,4) respectivamente. O uso desta rotina permite indicar as saídas ao programa por uma ordem errada sem causar quaisquer problemas.

```
432 FOR t=1 TO 4
433 FOR s=1 TO LEN a$
434 LET e(l,t)=((a$(s TO s)="N"
) AND t=1)+((a$(s TO s)="S") AND
 t=2)+((a$(s TO s)="E") AND t=3)
+((a$(s TO s)="O") AND t=4)-1
435 IF e(l,t)=0 THEN LET s=5
437 NEXT s: NEXT t
440 RETURN
```

Depois de terem sido definidas todas as salas e de terem sido incorporadas as descrições correspondentes, é necessário relacionar as salas entre si. Normalmente, uma aventura terá estas ligações já previamente definidas, fazendo parte do jogo. Como esta é uma aventura simples e não contém quaisquer objectos móveis, teremos de lhe acrescentar um pouco de interesse. Fá-lo-emos permitindo a ligação aleatória das salas entre si, mas garantindo em qualquer caso que quando se sai por sul, se entra na sala seguinte pelo norte.

```
444 REM mistura
450 GO SUB 1060
455 FOR t=1 TO m
460 FOR s=1 TO 4
470 IF e(t,s)=-1 THEN GO TO 500
480 IF e(t,s)<>0 THEN GO TO 500
490 GO SUB 520
500 NEXT s
510 RETURN
```

A rotina de mistura das salas é dividida em três partes principais. A primeira é uma subrotina que redefine as direcções originais no quadro E (subrotina 1060). Em seguida são avaliadas todas as posições, verificando-se se possuem ou não saídas

válidas. Se contêm zero, uma saída, a rotina salta para a linha 520 e procura em 100 posições aleatórias até encontrar uma que faça corresponder Norte a Sul, Este a Oeste, etc. Os números das posições são então incluídos no quadro E, realizando assim uma ligação.

```
515 REM trocar saidas
520 FOR u=1 TO 100
530 LET v=FN r(m)
540 IF (s=1 AND e(v,2)<>0) OR (
s=2 AND e(v,1)<>0) OR (s=3 AND e
(v,4)<>0) OR (s=4 AND e(v,3)<>0)
THEN GO TO 570
545 LET e(v,(s=2)+((s=1)*2)+((s
=3)*4)+((s=4)*3))=t
550 LET e(t,s)=v: LET u=100
570 NEXT u
580 RETURN
```

Quando tiverem sido cobertas todas as posições, a rotina regressa ao menú principal.

A secção mais importante que se segue permite a execução do programa. Primeiramente é apresentada a descrição da posição 1, sendo depois indicadas as saídas. Depois de indicar a direcção a seguir, as linhas 640 a 650 verificam se a direcção é válida ou se não foi atribuída. A linha 680 atribui o número da nova posição a L, que contém assim a posição actual.

```
 599 REM seccao principal
 600 LET l=1
 605 PRINT d$(l)
 620 PRINT "As saidas sao ";("No
rte" AND e(l,1)<>-1);(" Sul" AND
 e(l,2)<>-1);(" Este" AND e(l,3)
<>-1);(" Oeste" AND e(l,4)<>-1)
 625 PRINT a$;" ---------------
-----------"
 630 INPUT "Direccao: ";a$
 635 LET a$=a$(1 TO 1)
 640 IF (a$="N" AND e(l,1)=-1) O
R (a$="S" AND e(l,2)=-1) OR (a$=
"E" AND e(l,3)=-1) OR (a$="O" AN
D e(l,4)=-1) THEN PRINT "Nao pod
e ir por ai',": GO TO 620
 650 IF (a$="N" AND e(l,1)=0) OR
 (a$="S" AND e(l,2)=0) OR (a$="E
" AND e(l,3)=0) OR (a$="O" AND e
(l,4)=0) THEN PRINT "Se eu fosse
 a si voltava atras,": GO TO 610
 660 IF a$="A" THEN GO SUB 700
 670 IF a$="SIM" THEN RETURN
 680 IF a$="E" THEN GO TO 610
 682 LET l=((a$="N")*e(l,1))+((a
$="S")*e(l,2))+((a$="E")*e(l,3))
+((a$="O")*e(l,4))
 685 IF l=0 THEN PRINT "Eh??": L
ET l=1
 690 GO TO 610
```

As outras opções são a L, que imprime a descrição da sala actual, e a F que permite voltar ao menú principal. Para evitar confusões na execução, é novamente conveniente pedir confirmação ao utilizador.

```
700 INPUT "Tem a certeza? ";b$
710 RETURN
799 REM gravar d$(), e(),s()
```


As subrotinas finais têm geralmente a ver com o tratamento de dados, e permitem a gravação do quadro principal para fita ou microdrive, assim como a sua carga:

```
 800 INPUT "Nome da aventura? ";
f$

 815 SAVE *"m";1;f$+"S" DATA s()
 820 SAVE *"m";1;f$+"D" DATA d$(
)
 830 SAVE *"m";1;f$+"P" DATA e()

 840 RETURN
 899 REM carregar s(), e(), d$()
 900 INPUT "Nome do ficheiro: ";
f$
 910 LOAD *"M";1;F$+"S" DATA S()
 915 LOAD * "M";1;F$+"D" DATA D$
()
 920 LOAD *"M";1;F$+"P" DATA E()
 930 RETURN
```

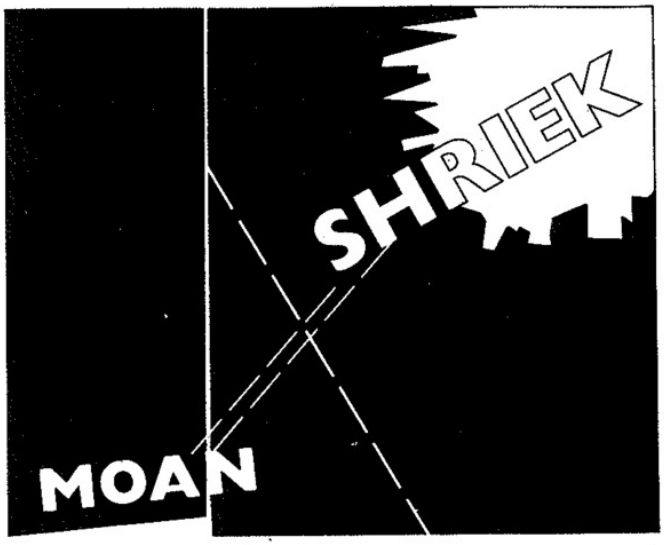
Como o programa altera o quadro E, para voltar ao início convém guardar em algum local as direcções originais. As duas rotinas que se seguem transferem os dados entre os quadros E e S:

```
 999 REM guardar direccoes origi
nais
1000 FOR t=1 TO m
1010 FOR s=1 TO 4
1020 LET s(t,s)=e(t,s)
1030 NEXT s
1040 NEXT t
1050 RETURN
1059 REM recuperar direccoes ori
ginais
1060 FOR t=1 TO m
1070 FOR s=1 TO 4
1080 LET e(t,s)=s(t,s)
1090 NEXT s
1100 NEXT t
1110 RETURN
```

E terminamos assim o programa. O leitor conseguirá provavelmente pensar em centenas de melhoramentos a introduzir nele, e foi escrito de tal modo que permita a introdução fácil de modificações. A arte da construção destes programas de

aventuras consiste em tornar as salas tão interessantes que adquiram suficiente valor próprio. Se se quiser introduzir monstros, basta fazer uma descrição do seguinte tipo:

«Você está numa sala escura.
Ouve num dos cantos um grito agudo
que termina num gemido surdo».



Resumo

Neste capítulo, o leitor deverá ter aprendido:

1. Como o Spectrum pode tratar quadros múltiplos, e como pode relacioná-los entre si.
2. A usar declarações REM na linha anterior ao início de uma subrotina, de modo a não interferirem na leitura do programa

3. A forma como se pode usar a instrução AND para escolher a opção adequada num programa.
4. O modo de usar GOSUB's aleatórios de modo a introduzir no programa um elemento de acaso.

Projecto

Tente acrescentar uma secção que lhe permita recolher tesouros neste programa.

6

INTRODUÇÃO AOS GRÁFICOS

As potencialidades gráficas do Spectrum são relativamente fáceis de usar – os gráficos são muito mais complicados em muitos outros computadores domésticos. No Spectrum, os gráficos pertencem basicamente a duas categorias – gráficos definidos pelo utilizador, baseados nas 32 × 22 posições de caracter disponíveis no visor do Spectrum, e o grupo de instruções PLOT e DRAW que permitem ao utilizador produzir grafismos ao nível dos mais simples elementos de imagem, os pixels. O Spectrum possui 256 × 176 pixels.

Mas existe uma desvantagem importante no trabalho gráfico do Spectrum. Só se podem definir as cores de INK e PAPER à escala das posições de caracter, não à dos pixels, e se bem que este problema possa ser evitado quando tido em conta, a imagem pode tornar-se extremamente confusa se os cálculos estiverem errados.

Uma experiência

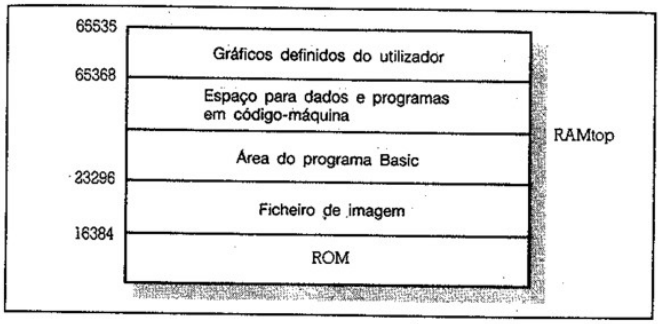
Suponhamos que o leitor deseja desenhar uma rede no visor do Spectrum; utilizará provavelmente um programa do tipo:

```
10 CLS
20 FOR n=5 TO 253 STEP 8
30 INK 5: PLOT n,0: DRAW 0,175
40 NEXT n
50 FOR n=5 TO 173 STEP 8
60 INK 6: PLOT 0,n: DRAW 255,0
```



```
70 NEXT n
80 GO TO 20
```

O que se deveria ter aqui seria um programa que desenhasse linhas a cores verticalmente, e depois as desenhasse na horizontal. Mas como o segundo grupo de linhas passa pelas posições de caracter já ocupadas pelo primeiro grupo de linhas, altera a cor destas linhas para a cor das segundas.



O leitor obterá o mesmo efeito quaisquer que sejam as instruções de tratamento do visor que utilizar, pelo que mesmo no caso de recorrer aos gráficos definidos pelo utilizador verificará que a representação de algo numa dada posição de caracter com determinada cor alterará a cor de tudo o que já nela se encontrar. Isto constitui geralmente um problema, mas em certos casos pode tornar-se útil.

Digamos que o leitor quer conceber um jogo do tipo «invasores», usando a PLOT e a DRAW para tratar os lasers. Se um feixe laser branco penetra na posição de caracter onde se encontra o invasor, o facto pode ser utilizado para colorir este de branco antes de o fazer desaparecer ou explodir.

Definição de imagens

É óptimo poder desenhar imagens e colori-las, e não será um grande problema gravá-las em fita a seguir, mas o leitor

já terá notado quanto tempo é necessário para carregar uma imagem. Não haverá problema se quiser apresentar um desenho no visor enquanto está a carregar um jogo comprido, mas se quiser carregar na máquina uma série de imagens durante a execução do jogo, este método tornar-se-á bastante aborrecido.

É neste ponto que se torna prático aprender um pouco sobre o modo como o Spectrum organiza a sua memória. O diagrama aqui apresentado na página anterior mostra parte do mapa de memória do Spectrum – poderá obter mais informação sobre o assunto se consultar o capítulo 24 do manual do Spectrum. O Spectrum de 48 K possui 65 536 posições de memória, que podem ser consideradas como caixas onde é possível guardar números.

Cada um destes endereços pode guardar um byte, representando um número entre zero e 255. Pensemos agora no modo como se definem caracteres gráficos do utilizador – cada linha do caracter gráfico definido contém um número binário com oito algarismos entre 00000000 e 11111111 que, se for convertido para decimal, dará qualquer coisa entre 0 e 255.

Podem armazenar-se números nesta gama em *todas* as posições de memória, e não apenas nas posições inicialmente dedicadas aos gráficos definidos pelo utilizador – o capítulo 25 do Manual do Spectrum mostra-nos o que se encontra nas posições entre 23 552 e 23 733, e o que se pode conseguir alterando os valores contidos nestas posições.

A maior parte da memória acima de 23 296 é usada para programação Basic, estendendo-se até à RAMTOP, para além da qual se encontram os gráficos definidos pelo utilizador. Um programa Basic não interferirá com qualquer coisa que se encontre acima da RAMTOP (literalmente, «topo da RAM»), sendo portanto possível armazenar informação acima desta, e desde que o programa Basic não POKE qualquer número nesta área torna-se possível invocar rotinas em código-máquina usando instruções Basic.

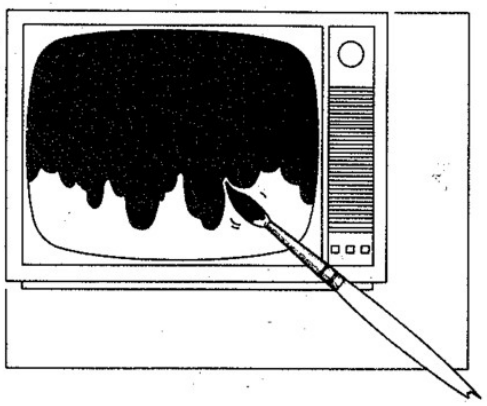
Uma experiência

Como o leitor descobrirá dentro em breve, o ficheiro de imagem do Spectrum está organizado de uma forma bizarra e enervante – não se preocupe muito com este assunto por enquanto, porque lhe indicaremos como pode resolver o problema.

```
10 FOR n=16384 TO 23295
20 POKE n,INT (RND*256)
30 NEXT n
```

Pergunte a si mesmo o que será feito neste programa – as posições de memória entre 16 384 e 22 528 controlam a imagem no visor, isto é, constituem o ficheiro de imagem. Estamos a usar a instrução POKE para colocar um número aleatório entre 0 e 255 em cada uma destas posições de memória, o que produz uma combinação de pontos e cores para cada posição no visor. Dependendo do seu gosto, o efeito resultante é horrível ou agradavelmente vivo.

Mas o leitor notará ainda que o Spectrum desenha o visor de uma forma estranha, por secções pequenas, introduzindo no final da cor. Isto deve-se ao modo um tanto estranho como armazena em memória a informação relativa à imagem. Se acrescentarmos a instrução 40 GO TO 40, a propósito, poderemos suprimir a mensagem na parte inferior do visor, pelo que a imagem poderá ocupar 24 linhas no total em vez das habituais 22. É de facto possível apresentar uma imagem também nas duas linhas inferiores.



Guardando tesouros

Do que aprendemos até agora devemos ter concluído qual a forma de guardar em memória e invocar de novo qualquer imagem. Como o estado do visor é guardado sob a forma de números entre 0 e 255 nas posições de memória 16 384 e 22 528, basta-nos dispor de uma rotina que leia estes números, os guarda noutra posição de memória e os vá buscar novamente quando deles necessitarmos.

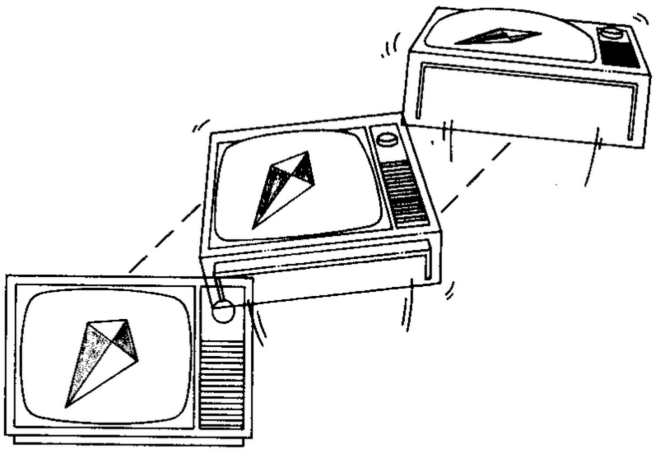
Uma experiência

É possível diminuir a dimensão da área de programa Basic deslocando a RAMTOP mais para baixo. Isto dá-nos mais espaço para os programas em código-máquina que, nas condições já indicadas, não interferem nos programas Basic.

```
  10 CLEAR 58430
  20 FOR n=58431 TO 58450
  30 READ b: PRINT n;b: NEXT n
  40 DATA 3,0,64,17,83,228,24,6,
33,83,228,17,0,64,1,0,27,237,176
,201
  50 NEW
```

Este programa desce a RAMTOP para o endereço 58 430, e POKE um programa em código-máquina na área que se encontra acima deste endereço. Em seguida o programa desaparece, devido ao uso da instrução NEW, eliminando-se assim as linhas Basic mas mantendo o programa em código-máquina acima da RAMTOP.

Torna-se assim possível desenhar a imagem que se quiser, e invocar a rotina citada para guardar essa imagem noutro ponto da memória. A instrução necessária para tal é LET imagem = USR 58 431. Vai-se novamente buscar a imagem, guardada nos 6912 endereços que se seguem a 58 451, usando a instrução LET imagem = USR 58 439. Esta quantidade, 6912, é obtida calculando oito linhas por caracter × 32 colunas ×24 linhas, mais 32 × 24 endereços para definição dos atributos de cada posição de caracter.



Poderemos usar qualquer variável que quisermos em vez de «imagem», mas devemos ter em conta que tal variável não deverá ser usada pelo programa Basic.

Uma das mais úteis rotinas gráficas não existentes originalmente no Spectrum é a de preenchimento de formas com determinadas cores. Existem vários métodos para o fazer, a maior parte dos quais são bastante lentos em Basic.

A rotina que se segue parte de um ponto no interior da área a preencher, e verifica os quatro pontos a ele adjacentes a fim de verificar se já foram ou não preenchidos. Se não, preenche-os e guarda as respectivas posições em quadros (um para os X e outro para os Y). Este preenchimento continua até todos os pontos no interior da área terem sido preenchidos. O rebordo pode ser considerado como uma trincheira escavada no solo, junto à qual um incêndio deixará de se expandir.

A dimensão da área a preencher é limitada pela dimensão do quadro usado para guardar os pontos adjacentes. Isto tem a desvantagem de limitar a área a um quadrado de cerca de 200 pixels por lado. A vantagem é que não é usada muita memória, e no caso de uma rotina em código-máquina adquirimos a certeza de que a rotina acabará por parar.

Experimente o seguinte:

```
  10 LET a=200: REM dimensao do
quadro
  20 DIM x(a): DIM y(a)
  30 CIRCLE 100,100,30
  40 LET x=105: LET y=105
  50 GO SUB 1000: REM encher
  60 STOP
 999 REM encher relva comecando
em x,y
1000 LET p1=1: LET p2=2
1010 LET x(1)=x: LET y(1)=y
1020 FOR d=1 TO 4
1030 LET t=-(d=1)+(d=3): LET s=-
(d=4)+(d=2)
1040 LET px=x(p1)+t: LET py=y(p1
)+s
1050 IF POINT (px,py)=1 THEN GO
TO 1100
1060 PLOT px,py
1070 LET x(p2)=px: LET y(p2)=py
1080 LET p2=p2+1
1090 IF p2>a THEN LET p2=1
1100 NEXT d
1110 LET p1=p1+1
1120 IF p1>a THEN LET p1=1
1130 IF p1<>p2 THEN GO TO 1020
1140 RETURN
```

Escreva agora a rotina seguinte e grave-a em fita ou microdrive, usando as instruções:

```
  SAVE "ENCHER"CODE 30519,173
```

ou

```
  SAVE *"M";1;"ENCHER"CODE 30519
,168
```

```
   5 REM versao em codigo-maquin
a
  10 CLEAR 30000
  15 LET c=0
  20 FOR t=30519 TO 30519+168
  30 READ a: POKE t,a
  40 LET c=c+a
  50 NEXT t
  60 IF c<>15116 THEN PRINT "Err
o. Verifique os dados."
  70 PRINT "POKE em 30514 a coor
denada X"
  80 PRINT "POKE em 30515 a coor
denada Y"
  90 PRINT "Execute com RANDOMIZ
E USR 30519"
 100 DATA 62,0,50,48,119,62,1,50
,49,119
 110 DATA 33,48,117,58,50,119,11
9,33,48,118
 120 DATA 58,51,119,119,62,4,50,
52,119,33
 130 DATA 48,117,58,48,119,95,22
,0,25,78
 140 DATA 33,48,118,25,70,58,52,
119,254,1
 150 DATA 202,130,119,254,2,202,
126,119,254,3
 160 DATA 202,122,119,13,195,131
,119,4,195,131
 170 DATA 119,12,195,131,119,5,1
20,50,54,119
 180 DATA 121,50,53,119,58,54,11
9,71,58,53
 190 DATA 119,79,205,206,34,205,
213,45,254,1
 200 DATA 202,198,119,58,54,119,
71,58,53,119
 210 DATA 79,205,229,34,33,48,11
7,58,49,119
 220 DATA 95,22,0,25,58,53,119,1
19,33,48
 230 DATA 118,25,58,54,119,119,5
8,49,119,60
 240 DATA 50,49,119,58,52,119,61
,50,52,119
 250 DATA 194,84,119,58,48,119,6
0,50,48,119
 260 DATA 71,58,49,119,184,194,7
9,119,201,75
 270 DATA 69,78,78
```

Veja como o código-máquina é muito mais rápido recorrendo ao exemplo seguinte:

```
  10 CLEAR 30000
  20 LOAD *"M";1;"ENCHER"CODE 30
519: REM Carregar de microdrive;
 tirar *"M";1; no caso de usar c
assette
```

```
30 CIRCLE 100,100,30
40 POKE 30514,100
50 POKE 30515,100
60 RANDOMIZE USR 30519
70 PRINT "QUE TAL, EH?"
```

A posição de memória 30514 é usada para guardar a posição horizontal de um ponto no interior da forma, e 30515 guarda a sua posição vertical. Chamando a rotina em 30519 colocamos o código-máquina em execução.

A instrução CLEAR 30000, no início do programa, liberta não só uma certa área da memória para guardar o código-máquina, como ainda reserva memória para os quadros.

A rotina em código-máquina funciona exactamente da mesma maneira que o programa Basic mas, como facilmente podemos constatar, é muito mais rápida.

O leitor poderá preencher qualquer forma usando esta rotina. A única coisa em que convém ter cuidado consiste em verificar se o rebordo da área não contém falhas, senão o enchimento espalhar-se-á à região exterior à forma.

Resumo

Neste capítulo o leitor aprendeu:

1. A forma como o Spectrum organiza a imagem apresentada no visor, e o modo como esta imagem se encontra guardada em memória.
2. Um pouco sobre o mapa de memória do Spectrum, e a forma como se deve reservar espaço para introdução de código-máquina recorrendo à instrução CLEAR.
3. Como é possível guardar acima da RAMTOP qualquer imagem que se encontre no visor, invocando-a e imprimindo-a novamente sempre que for necessário.

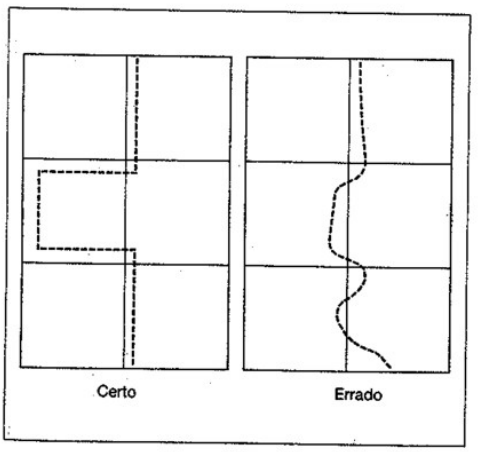
Projecto

Escreva um programa de desenho que inclua uma rotina escrita em Basic que permita ler os atributos de cada uma das posições de caracter existentes no visor, e que em seguida altere as cores correspondentes. Para a execução deste projecto o leitor deverá consultar as referências contidas no manual do Spectrum sobre a instrução ATTR.

7

A COR

Se o leitor tentou alguma vez desenhar imagens complicadas no seu Spectrum, e se tiver visto ainda as imagens de título de alguns dos jogos comerciais mais espectaculares, já terá perguntado a si próprio como será que os programadores profissionais conseguem estes resultados. Aparentemente estes jogos dispõem de imagens de título bastante detalhadas, onde ocorrem pixeis individuais marcados em cores diferentes.



O leitor sabe provavelmente que o Spectrum permite facilmente o uso de cores de INK e PAPER diferentes, mas que só pode usar uma INK e uma PAPER para cada posição de caracter – como conseguirão portanto os profissionais obter imagens de tanta qualidade?

A resposta a isto é que, de facto, não existe qualquer forma de colorir a imagem pixel a pixel no Spectrum. Isto pode parecer decepcionante, mas o facto de serem possíveis soberbas imagens no Spectrum mostra que, com um pouco de imaginação, o problema pode ser torneado.

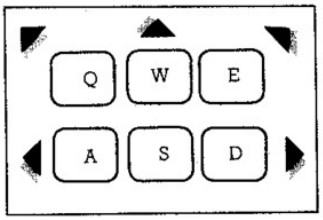
É apenas necessário desenhar as linhas de tal modo que se adaptem facilmente a diferentes posições de caracter – observe o diagrama da página anterior e compreenderá o que pretendemos dizer.

Um programa de desenho

Já dispomos de uma rotina de preenchimento das formas desenhadas, mas o desenho através de instruções PLOT e DRAW pode ser bastante aborrecido, sendo fácil cometer erros. Aquilo de que necessitamos é um modo de desenhar formas com facilidade, e de apagar os nossos erros. Naturalmente queremos usar a cor na nossa rotina, pelo que necessitaremos também de uma forma de verificar quando estamos a sair dos limites das posições de caracter. Se conseguirmos ver no visor estes limites, deixaremos de cometer tantos erros:

```
  10 LET ink=1: LET x=128: LET y
=88
  20 LET grid=200: LET report=90
0: LET print=300: LET stop=500
  30 GO SUB grid: REM definir re
de
  40 GO SUB report: REM definir
impressao
  49 REM definir modos
  50 IF INKEY$="I" THEN LET ink=
1: GO SUB print
  60 IF INKEY$="O" THEN LET ink=
3: GO SUB print
  70 IF INKEY$="P" THEN LET ink=
0
  75 GO SUB print
  80 IF INKEY$="S" THEN GO SUB s
top
  99 REM mover cúrsor
 100 IF INKEY$="A" THEN LET x=x-
1
 110 IF INKEY$="D" THEN LET x=x+
1
 120 IF INKEY$="W" THEN LET y=y+
1
 130 IF INKEY$="X" THEN LET y=y-
1
 140 IF INKEY$="Q" THEN LET x=x-
1: LET y=y+1
 150 IF INKEY$="E" THEN LET x=x+
1: LET y=y+1
 160 IF INKEY$="Z" THEN LET x=x-
1: LET y=y-1
 170 IF INKEY$="C" THEN LET x=x+
1: LET y=y-1
 180 GO TO 50
```

Neste programa a variável «ink» deve ser escrita letra a letra, e as variáveis «print» e «stop» também. Não convirá portanto confundi-las com as instruções INK, PRINT e STOP. São variáveis, não palavras-chave.



Este programa segue o formato habitual. Exceptuando as linhas 100-170, que activam um conjunto de teclas de cursos à volta da tecla S, o programa consiste ainda, e apenas, é em GOSUB's. Notemos, porém, que existe uma GOSUB para um menú, o que é algo de novo.

Existe uma razão perfeitamente lógica para isto. Estamos a construir aqui um programa de desenho, pelo que necessitamos de todo o visor para tal. Se imprimíssemos um menú no visor, não poderíamos desenhar. Recorremos assim às linhas de men-

sagem, não disponíveis para outros efeitos em Basic; e esta não é obviamente a tarefa que primeiro interessa resolver neste caso.

Trataremos do assunto mais adiante; por agora, falemos das outras subrotinas que teremos de escrever. A rotina da rede é bastante fácil. Bastar-nos-á produzir um padrão, tipo tabuleiro de xadrez, que indique os limites das posições de caracter em todo o visor. O tratamento de INK e PAPER é óbvio, indicando se estamos a desenhar uma linha ou a apagá-la, enquanto STOP nos permite sair elegantemente do programa. PRINT utilizará a instrução POINT para descobrir se a imagem se encontra ou não no visor.

O menú

O leitor já sabe como pode obter uma imagem nas duas linhas inferiores do visor, usando a instrução POKE para colocar os números adequados na parte apropriada do ficheiro de imagem – isto pode parecer uma forma aborrecida de trabalhar, mas tenha em conta que alguns micros chegam a usar a POKE para imprimir em todo o visor.

No entanto, o leitor estará provavelmente a pensar sob qual será a maneira de descobrir os números que deve colocar em cada endereço – parece aborrecido, não é? Felizmente não. Basta procurar na ROM os padrões que queremos imprimir, e de POKE em seguida estes números na parte do ficheiro de imagem corresponde à posição de caracter onde queremos imprimir.

Pode-se simplificar isto ainda mais, determinando quais os DATA necessários, usando depois a POKE para os introduzir, juntamente com um ciclo FOR... NEXT! Mas não iremos fazê-lo agora, na medida em que o problema das declarações DATA consiste em só o autor as compreender. E, por outro lado, a subrotina que se segue, além de provocar dores de cabeça ao autor enquanto esteve a ser produzida, permite-nos imprimir quaisquer caracteres que se queira nas duas linhas de mensagem da parte inferior do visor. Bastará alterar o que se encontra contido em A$:


```
 899 REM imprimir em linhas de m
ensagem (report)=900
 900 DIM a$(61)
 910 LET a$="MODO = DESENHO  CUR
SOR = 128,88 D - DESENHAR,  S -
PARAR"
 920 FOR n=1 TO 61
 930 LET b=CODE a$(n)
 940 LET c=15360+b*8
 950 LET d=20671+n
 960 FOR p=0 TO 7
 970 POKE d+(256*p),PEEK (c+p)
 980 NEXT p: NEXT n
 990 RETURN
```

Se quisermos ver este programa em acção, deveremos acrescentar um PAUSE O na linha 985 – isto manterá a imagem estática até carregarmos numa qualquer tecla. Que estamos afinal a fazer aqui? As linhas 900 e 910 são fáceis de compreender, dimensionando uma cadeia alfanumérica com 61 caracteres e definindo-a em seguida. É necessário que a cadeia em causa tenha de facto 61 caracteres, senão obteremos mensagens de erro.

A linha 920 inicia um ciclo FOR... NEXT que primeiramente obtém o CODE (código) do caracter actual, definindo em seguida C como equivalendo a 15360 mais oito vezes esse código. O endereço de memória onde se inicia o conjunto de caracteres é 15360, e como cada caracter possui oito linhas de padrões de pixels, consumindo oito endereços, o endereço do início de cada caracter é oito vezes o seu código mais 15360.

Sabemos agora onde se inicia o caracter que queremos descobrir; mas temos ainda de aprender a forma de o imprimir na posição adequada. O endereço 20672 é o endereço da primeira fiada de pixels do primeiro caracter da linha 22 do visor, e como para cada caracter os oito endereços se sucedem por saltos de 256, teremos de POKE um número em D, depois outro em D + 256, depois em D + 512, etc. É isto que faz a primeira parte da linha 970.

Mas para que o leitor não morra de desespero o conjunto de caracteres em ROM ocupa endereços consecutivos, pelo que a segunda parte da linha 970 descobre, recorrendo à instrução PENK, quais são esses valores. Passamos depois ao caracter seguinte, e à posição de caracter, seguinte, somando

1 a D; com efeito, o primeiro endereço da nova posição de caracter é imediatamente a seguir ao da anterior.

Compreendo perfeitamente que o leitor pergunte quem foi o maníaco que concebeu o ficheiro de imagem do Spectrum.

A tabela que se segue mostra as posições ocupadas pelos primeiros sete caracteres da primeira linha do ficheiro de imagem. Note que os endereços de memória sobem um a um quando observamos o visor da esquerda para a direita, mas que os números na vertical parecem desordenados. Se o ficheiro da imagem do Spectrum estivesse organizado de uma forma que pudesse ser considerada lógica, teríamos 32 números ao longo do visor (na horizontal), correspondentes a 32 posições de caracter, pelo que o primeiro número da segunda fiada de pixels seria 16384 + 32, ou seja, 16416.

	Coluna:						
Linha:	1	2	3	4	5	6	7
	16384	16385	16386	16387	16388	16389	16390
	16640	16641	16642	16643	16644	16645	16646
	16896	16897	16898	16899	16900	16901	16902
	17152	17153	17154	17155	17156	17157	17158
	17408	17409	17410	17411	17412	17413	17414
	17664	17665	17666	17667	17668	17669	17670
	17920	17921	17922	17923	17924	17925	17926
	18176	18177	18178	18179	18180	18181	18182

Se estudarmos a tabela verificaremos que 16146 será o endereço da primeira fiada de pixels da posição de caracter 1, 0, e se voltarmos ao nosso programa de enchimento do visor verificaremos que o Spectrum desenha cada imagem uma linha de cada vez, começando pela sua primeira fiada de pixels, passando à primeira fiada de pixels da linha seguinte, etc., até terminar as oito primeiras linhas; passa em seguida ao segundo bloco de oito linhas, e finalmente ao terceiro (que inclui as duas linhas da janela de mensagens inferior).

É portanto fácil compreender que qualquer rotina que faça algo simples, como PEEK os endereços onde se encontra um caracter em ROM, seguida de POKE dos mesmos valores noutros endereços, será necessariamente um tanto complicada.

Se quiser fazer um POKE para qualquer parte do ficheiro de imagem, deverá consultar a tabela que se segue para determinar os endereços apropriados:

Mapa do ficheiro de imagem

Linha	Início da linha	Final da linha
0	16384	16415
1	16416	16447
2	16448	16479
3	16480	16511
4	16512	16543
5	16544	16575
6	16576	16607
7	16608	16639
8	18432	18463
9	18464	18495
10	18496	18527
11	18528	18559
12	18560	18591
13	18592	18623
14	18624	18655
15	18656	18687
16	20480	20511
17	20512	20543
18	20544	20575
19	20576	20607
20	20608	20639
21	20640	20671
22	20672	20703
23	20704	20735

O leitor notará certamente que existe um modo matemático de deduzir estes endereços. O início de cada fiada encontra-se 32 endereços acima do início da fiada anterior (falamos aqui de fiadas de pixels, obviamente), excepto no caso do salto entre as linhas 15 e 16, onde é necessário acrescentar 1824.

O cursor

O posicionamento do cursor é provavelmente a parte mais importante deste nosso programa, porque se não soubermos onde se encontra o cursor teremos grandes problemas quando for necessário preencher uma forma. Normalmente pensamos no cursor como tratando-se de um quadrado cintilante ou numa forma em cruz, mas para este efeito particular necessitamos de algo que mostre os limites do quadrado onde nos encontramos.

Poderíamos portanto organizá-lo como um ponto com uma linha em torno da posição de caracter onde se encontra. Mas seria necessário mover esta caixa antes de o cursor entrar em contacto com ela pois em caso contrário estragaria o desenho. Pela mesma razão, tudo o que envolva uma cor de INK poderia ser difícil de tratar, porque duas cores diferentes no mesmo caracter causariam o caos.

Isto conduz-nos à opção pelo PAPER. Se introduzirmos, recorrendo à instrução POKE, diferentes atributos de PAPER alternadamente em cada posição de caracter, obteremos uma

Linha	Início da linha	Final da linha
0	22528	22559
1	22560	22591
2	22592	22623
3	22624	22655
4	22656	22687
5	22688	22719
6	22720	22751
7	22752	22783

8	22784	22815
9	22816	22847
10	22848	22879
11	22880	22911
12	22912	22943
13	22944	22975
14	22976	23007
15	23008	23039
16	23040	23071
17	23072	23103
18	23104	23135
19	23136	23167
20	23168	23199
21	23200	23231
22	23232	23263
23	23264	23295

rede com diferentes cores de fundo. Podemos desenhar sobre ela sempre que quisermos, e ao terminar eliminaremos a rede e usaremos a rotina de enchimento já apresentada.

```
199 REM definir rede (grid)=200
200 FOR n=22528 TO 23168 STEP 6
4
210 FOR p=0 TO 31 STEP 2
220 POKE n+p,40: POKE n+p+33,40
230 NEXT p: NEXT n
240 RETURN
```

Aqui N trata novamente posições de memória desta vez a área usada para guardar os atributos de uma dada posição de caracter. Cada uma destas posições de memória guarda um número específico indicando uma determinada combinação de INK, PAPER, BRIGHT e FLASH, bastando apenas escrever uma rotina que POKE o valor correcto nas posições adequadas. A tabela acima indica os endereços relevantes para cada posição de caracter.

Resta-nos apenas construir um método de alterar o indicador de modo na linha de mensagens. Seria possível incorporá-lo na rotina anterior que imprime estas linhas, mas não o fazer não complica muito o programa:

```
299 REM print
300 PLOT INK ink;x,y
399 REM alterar janela
400 IF ink=1 THEN LET a$="DESEN
HAR"
410 IF ink=0 THEN LET a$="PARAR
"
420 FOR n=1 TO 4
430 LET b=CODE a$(n)
440 LET c=15360+b*8
450 LET d=20578+n
460 FOR p=0 TO 7
470 POKE d+(256*p),PEEK (c+p)
480 NEXT p: NEXT n
490 RETURN
```

Poderíamos ainda dar um último retoque na rotina de paragem:

```
499 REM stop
500 FOR n=22528 TO 23199
510 POKE n,56
520 NEXT n: STOP
```

A cor

PAPER	INK							
	Preto	Azul	Verm.	Magenta	Verde	Ciáo	Amarelo	Branco
Preto	0	1	2	3	4	5	6	7
Azul	8	9	10	11	12	13	14	15
Verm.	16	17	18	19	20	21	22	23
Magenta	24	25	26	27	28	29	30	31
Verde	32	33	34	35	36	37	38	39
Ciáo	40	41	42	43	44	45	46	47
Amarelo	48	49	50	51	52	53	54	55
Branco	56	57	58	59	60	61	62	63

NOTA: Se se pretende obter um caracter em BRIGHT, acrescenta-se 64 ao valor indicado; para obter FLASH acrescenta-sa ainda 128.

Depois de tratarmos do ficheiro de imagem do Spectrum, notemos finalmente que a obtenção de cor é relativamente simples. Desde que coloquemos as cores pretendidas nas posições de caracter adequadas, basta alterar os atributos das posições de caracter que desejamos colorir. A tabela da página anterior que se segue mostra quais os valores a utilizar para diferentes efeitos.

O leitor não terá dificuldade em juntar o nosso programa de desenho ao de enchimento, de modo a obter um programa de desenho bastante convincente. Claro que haverá muitas coisas que o programa não conseguirá fazer – em particular, seria muito bom que pudesse ampliar a posição de caracter onde o cursor se encontra para podermos observar os pixels separados, mas também isto poderia ser acrescentado.

Resumo

Neste capítulo aprendemos o seguinte:

1. Como evitar dificuldades ao desenhar devido a problemas no uso de DRAW, INK e PAPER.
2. Como descobrir qual o endereço do ficheiro de imagem onde convirá utilizar a POKE para alterar o desenho, e onde a mesma função pode ser usada para modificar a cor.
3. Como ler padrões de pontos existentes em ROM.

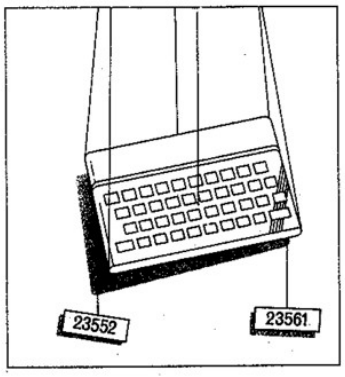
Projectos

1. Escrever uma nova subrotina que altere os atributos do conjunto do visor de qualquer maneira que se pretenda.
2. Escrever uma rotina tipo máquina de escrever que aceite texto escrito no teclado e o imprima nas linhas de mensagem.

8

AS VARIÁVEIS DE SISTEMA

As variáveis de sistema são aquilo que o computador utiliza para manter as suas contas em dia. Saber, por exemplo, qual a linha do programa Basic que está a ser executada, o estado do teclado, etc., depende do armazenamento dos valores adequados na parte da memória dedicada às variáveis de sistema, isto é, nos endereços 23552 a 23732. O utilizador pode ler qualquer destas variáveis mas não convirá escrever (isto é, alterar usando a POKE) em algumas delas, dado que pode levar ao estoiro do sistema. Em geral, este estoiro é causado por não se saber exactamente para que serve uma dada posição, usando no entanto a POKE para introduzir nela qualquer coisa no momento errado.



Algumas das variáveis de sistema podem no entanto ser bastante úteis e permitem ao Spectrum ser adaptado de diversas formas, por exemplo redefinindo o conjunto de caracteres ou escrevendo nas linhas de mensagem.

A fim de evitar resultados inesperados, desligue e volte a ligar o seu Spectrum entre cada uma das rotinas que se seguem.

Este capítulo será preenchido por uma lista das variáveis de sistema, dando alguns pormenores sobre as mais úteis e algumas ideias quanto à forma como poderão ser usadas.

Antes de começarmos, convirá notar que poderemos definir algumas funções muito úteis. A primeira permite descobrir um endereço ou valor guardado em dois bytes, usando a instrução PEEK:

```
DEF FN A(X) = PEEK X + 256 * PEEK (X + 1)
```

Como veremos adiante, esta função torna-se muito útil para observarmos indicadores que apontam para outros.

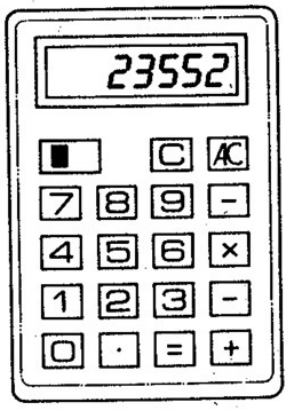
Para dividir um número (N) nos seus dois componentes de maior e menor ordem, usaremos o método seguinte:

```
Maior ordem = INT (N/256)
Menor ordem = N – 256 * INT (N/256)
```

Estas expressões serão também usadas muitas vezes, dado que a maior parte dos indicadores contidos nas variáveis de sistema são guardados em dois bytes, permitindo a indicação de endereços e números de linha entre 0 e 65535. Algumas das variáveis de sistema são tão transitórias ou tão inúteis para nós que não interessará perdermos tempo com o seu uso em rotinas Basic. Algumas delas poderão ser úteis aos programadores em código-máquina, mas como este assunto se encontra para além do tema deste livro o leitor deverá estudá-lo noutras obras.

23552, KSTATE

Os endereços entre 23552 e 23559 são usados pelo sistema para armazenar o estado do teclado. Dividem-se em dois con-



juntos de quatro bytes. O segundo é usado para controlar a detecção da primeira tecla em que se carrega. Se for premida outra tecla, libertando-se depois a primeira, os primeiros quatro endereços substituirão o segundo conjunto. Em cada um dos conjuntos os quatro endereços têm as mesmas funções; considerando o segundo:

23556 contém 255 quando não se está a carregar em qualquer tecla, senão contém o código ASCII da letra maiúscula representada pela tecla premida.

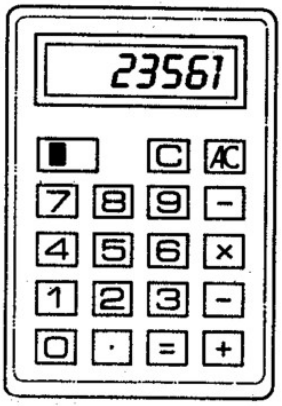
23552 é o contador de repetição, que depende da velocidade de repetição, normalmente 5, que é copiada de 23562.

23558 guarda o valor de atraso entre o momento em que se carrega numa tecla e o início das repetições. Este valor, contido em 23561, é normalmente 35. O endereço citado funciona como contador, contando até 0 antes de a tecla começar a repetir.

23559 contém o código ASCII da letra minúscula correspondente à tecla premida.

23560, LASTK

Este endereço é usado pela rotina de varrimento do teclado para armazenar o valor ASCII da última tecla premida. Seria possível usar a PEEK para ler aqui esse valor, em vez de se usar a INKEY$, eliminando a necessidade de usar a instrução CODE.



23561, REPDEL

É guardado aqui o número que especifica o tempo de atraso antes de se iniciar a repetição de cada tecla. Pode fazer-se POKE neste endereço de qualquer valor entre 0 e 255 – 0 desliga completamente a repetição, 1 não permite praticamente qualquer atraso (começando a tecla a repetir imediatamente), e 255 torna este processo o mais lento possível.

23562, REPPLR

Guarda-se aqui a velocidade de repetição de cada tecla, sendo também possível alterá-la para qualquer valor entre 0 e 255. O não elimina completamente a repetição, obrigando apenas o contador a dar «a volta» – passa de 0 a 255, 254, etc. O valor 1 apressa muito a repetição.

23563, DEFADD

Estes dois endereços guardam o endereço da função definida pelo utilizador que está a ser avaliada. Por exemplo:

```
  10 FOR t=FN x(0) TO FN x(0)+50
  20 PRINT t;" ";
  30 PRINT PEEK t;" ";
  40 IF PEEK t>31 THEN PRINT CHR
$ PEEK t: GO TO 60
  50 PRINT
  60 NEXT t
  70 DEF FN x(r)=PEEK 23563+256*
PEEK 23564
```

Se executarmos este programa teremos uma ideia do modo como as funções são guardadas na memória. Note-se que no caso de não estarem a ser avaliadas quaisquer funções, a variável DEFADD contém zero.

23565, K DATA

Esta posição guarda o número da cor indicada directamente nos controlos de cor escritos no teclado.

23566, TVDATA

Esta variável é semelhante à anterior, mas utiliza um segundo byte que indica a posição de caracter (na horizontal) correspondente ao último TAB ou AT usados.

23568-23605, STRMS

Nestes endereços são guardados os endereços dos diversos canais ligados a «streams». De início, os primeiros 14 bytes guardam os dados correspondentes aos canais −3 a 3. À medida que se acrescentam outros «streams», é inserida informação até um total de 19.

23606/7, CHARS

Estas duas posições guardam o endereço do conjunto de caracteres usado pela máquina, inicialmente em ROM. O endereço indicado é de facto inferior em 256 ao endereço real do primeiro caracter, o que pode parecer estranho à primeira vista. Se considerarmos que não é possível imprimir os primeiros 31 caracteres, ou seja, que estes constituem simplesmente códigos de controlo, aquela diferença torna-se lógica, dado que o primeiro caracter que pode ser impresso é o «espaço», com o código 32, e 32×8 = 256. Para descobrir o endereço de um caracter, o processador necessita então de multiplicar apenas por 8 o código respectivo, somando depois este valor a CHARS.

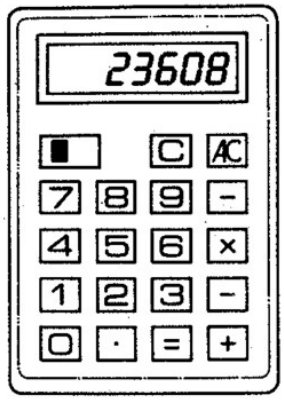
Uma vantagem desta variável é que nos permite fazer um novo conjunto de caracteres e utilizá-lo em vez do normal. O programa que se segue relocata (altera a localização) do conjunto de caracteres, passando-o para a RAM, e depois redefine parte da letra 'A' (código ASCII 65).

```
  10 CLEAR 39999
  20 PRINT "AAAAA".
  30 PRINT PEEK 23606,PEEK 23607
  40 LET ch=PEEK 23606+256*PEEK
23607
  50 LET ch=ch+256
  60 FOR t=0 TO 127*8
  70 POKE 40000+t,PEEK (15616+t)
  80 NEXT t
  90 POKE 23607,INT ((40000-256)
/256)
 100 POKE 23606,40000-256*INT (4
0000/256)
 110 REM redefinir parte de "A"
 120 POKE 39744+(8*65),255
 130 PRINT "AAAAA"
```

Para voltar ao conjunto de caracteres original, deve-se POKE 23606,0 e POKE 23607,60.

23608, RASP

O valor aqui guardado especifica o comprimento do sinal de aviso.



23609, PIP

Esta variável indica a duração do ruído produzido por qualquer tecla, podendo ser introduzido aqui um novo valor a fim de tornar este ruído mais audível.

23610, ERR NR

É aqui guardado o código de erro menos um; se se introduz aqui um determinado número, provoca-se o erro correspondente.

23611, FLAGS

Este endereço contém um certo número de «flags» (indicadores de estado) usadas pelo Spectrum para diversas operações. Os seus oito bits têm o seguinte significado:

O bit 1 encontra-se ao valor 1 quando se pretende usar o «stream» 3 para saída de uma instrução PRINT. Se for zero, está a ser usado o «stream» 2 (3 corresponde normalmente à impressora e 2 ao visor). O bit 2 é passado a 1 quando se

está a imprimir em modo "L", ou a 0 quando se usa o modo "K". O bit 3 é igual a 1 quando se realiza um "input" em modo "L", e a zero quando este é feito em modo "K".

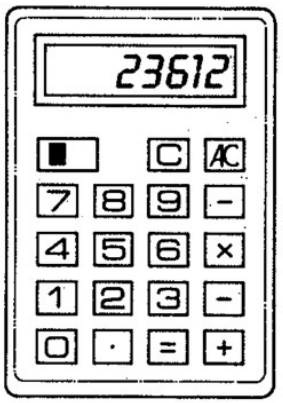
O bit 5 é passado a 1 quando se carrega numa tecla depois de ter sido passado a 0.

O bit 6 é usado para indicar se a expressão actual é um número (0) ou uma cadeia (1).

O bit 7 é zero quando o interpretador Basic está a verificar a sintaxe de uma linha, e um quando está a ser executado um programa.

A maior parte das «flags» são pouco úteis ao utilizador, mas podem constituir interessantes indicadores de estado que podem ser lidos recorrendo à instrução PEEK.

23612, TVFLAG



Este conjunto de «flags» é usado para indicar o estado do visor.

O bit 0 está ao valor um quando se está a usar a parte inferior do visor, e zero quando se está a tratar a janela principal do visor.

O bit 3 indica que o modo actual (K, L, etc.) pode ter mudado, e necessita de ser verificado.

O bit 4 é um se está a ser impressa uma listagem automática. Se não é zero.

O bit 5 é usado para indicar que é necessário limpar a janela inferior do visor.

23613/4, ERR SP

Esta variável é usada para indicar a linha onde foi invocado um GOSUB. Por exemplo, experimente:

```
 10 PRINT FN a(FN a(23613)+2)
 20 GO SUB 1000
 30 STOP
100 DEF FN a(x)=PEEK x+256*PEEK
 (x+1)
1000 PRINT FN a(FN a(23613)+2)
1010 GO SUB 2000
1020 RETURN
2000 PRINT FN a(FN a(23613)+2)
2010 RETURN
```

A função FN A é usada aqui para obter o endereço de 16 bits da posição especificada em argumento. Assim, quando usada duas vezes, obtém o número indicado pelo endereço da primeira execução. O coeficiente +2 é necessário devido ao facto de os números de linha de GOSUB serem guardados num «stack», e de o indicador de «stack» ter sido aumentado de dois quando nos encontramos na subrotina. O programa apresentado pode ser usado num programa que tenha dificuldade com as suas subrotinas, dado que nos indica de que ponto vem cada GOSUB.

23617, MODE

Esta variável define o cursor e modo de «input» (entrada) a usar. Experimente por exemplo:

```
 10 INPUT "Indique numero entre
e 255 ";a
 20 POKE 23617,a
 30 GO TO 10
```

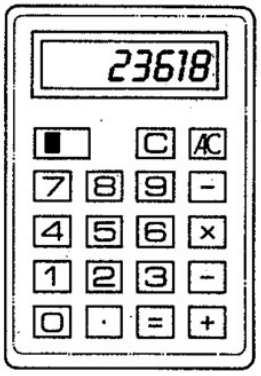

Note como ocorrem cursores diferentes conforme os números indicados. Experimente, por exemplo, os valores 10 e 255 e note como o cursor se altera para "O" ou "I". Isto pode ser útil quando se dá entrada a dados que devem ser aceites num determinado modo, por exemplo, em maiúsculas ou gráficos.

Este programa só pode ser executado uma vez com a instrução RUN.

23618/9, NEWPPC
23620, NSPPC

Estes três endereços podem ser considerados em conjunto dado que é possível usá-los para definir o número de linha e instrução dentro desta que serão executados em seguida. Por exemplo, experimente:

```
  10 PRINT "Linha 10"
  20 POKE 23618,1000-256*INT (10
00/256): POKE 23619,INT (1000/25
6): POKE 23620,3
  30 PRINT "Linha 30"
  40 STOP
1000 PRINT "Linha 1000 Inst.1":
PRINT "Linha 1000 Inst.2": PRINT
 "Linha 1000 Inst.3"
1010 GO TO 30
```



Este programa saltará directamente para a instrução 3 da linha 1000, e esta ideia pode ser usada de diversas maneiras, incluindo em código-máquina, para saltar directamente para uma determinada linha Basic.

23621/2, PPC
23623, SUBPPC

Também estes três endereços podem ser considerados em conjunto, dado que apontam directamente para a instrução em execução. Não são de facto muito úteis em Basic, mas podem ser usados por uma rotina de código-máquina funcionando por «interrupt» para rastrearem a execução de um programa em Basic.

23624, BORDCR

Esta variável contém a cor da margem, multiplicada por oito. Os bits 6 e 7 (equivalentes a 64 a 128 em decimal) podem ser usados para levar a janela inferior do visor a ser impressa em FLASH ou BRIGHT. Realizando POKE4s de valores neste endereço pode experimentar-se os resultados.

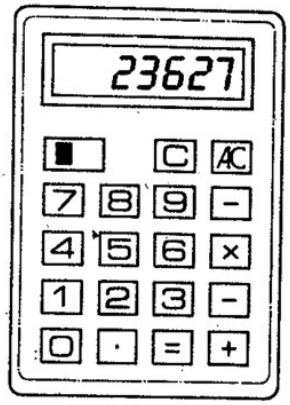
23625/6, E PPC

Quando é executada a instrução LIST, ou é forçada uma listagem automática, estas posições guardam o número da linha que contém o cursor. Atribuindo valores a esta variável usando a POKE, alteraremos a linha onde o cursor se encontra. Uma possível utilidade disto consiste em sair de um programa com o cursor numa dada posição.

Alternativamente, pode remover-se o cursor colocando zero em ambos os endereços ocupados pela variável em causa.

23627/8, VARS

Nesta variável é indicado o início da zona de memória onde são guardadas as variáveis em execução. Este indicador pode ser útil para aqueles que querem aceder à zona de variáveis a partir de rotinas em código-máquina, permitindo a transferência de dados sem recorrer a PEEK's e POKE's. A constituição da área de variáveis é indicada no manual do Spectrum.



23629/30, DEST

Esta variável contém o endereço da primeira letra do nome da variável que está a ser usada em Basic. Se se trata de uma variável nova, apontam para o endereço imediatamente anterior a E LINE, onde deve ser guardado o início da nova variável.

23635/6, PROG

O endereço do início do programa Basic é guardado aqui. Normalmente este endereço não pode ser alterado, não havendo possibilidade de colocar mais de um programa em memória a não ser que se alterem muitos outros indicadores.

23637/8, NXTLN

O endereço da linha Basic a ser executada a seguir é guardado nesta variável. Uma vez mais, esta variável não será muito útil, a não ser para permitir aos programas alterarem-se a si mesmos. Experimente o seguinte e veja o que acontece:

```
 10 POKE FN a(23637)+6,65
 20 REM "Ola"
100 DEF FN a(x)=PEEK x+256*PEEK
(x+1)
```

Se agora alterar a linha 10 para:

```
10 POKE FN a(23637)+4,245
```

e voltar a executar o programa, verificará como é possível levar os programas a alterarem-se a si mesmos. Note que 245 é uma palavra-chave.

23639/40, DATADD

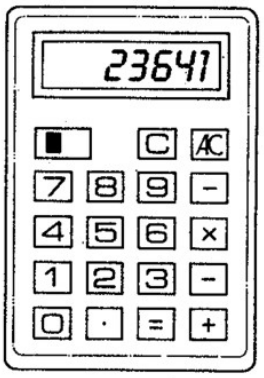
O endereço aqui guardado é usado para conhecer qual o último elemento dos dados usados. Se não existirem mais dados depois desta declaração, ocorre um erro «Out of data».

23641/2, E LINE

Esta variável guarda o endereço do início da área de montagem («edit») e aponta para a linha que no momento está a ser montada.

23659, DF SZ

Esta variável contém o número de linhas, incluindo uma em branco, correspondentes à janela inferior do visor (janela de mensagens). Contém geralmente o valor 2, mas pode ser



modificada para zero a fim de juntar as duas linhas inferiores ao resto do visor. O inconveniente é que deve ser novamente passada para 2 antes do final de um programa, senão a máquina ficará em «crash».

O número de linha aqui especificado pode também ser aumentado levando a mensagem «scroll?» a surgir mais acima no visor. O problema é que no caso de se responder sim, obtém-se um erro «out of screen». Só serve portanto para aumentar o número de linhas para 24, do seguinte modo:

```
 10 LET a$=INKEY$
 20 IF a$=" " THEN GO TO 100
 30 POKE 23659,0: REM ver scrol

 40 PRINT "AA"
 50 GO TO 10
100 POKE 23659,2
```

Note que se para a máquina («break») enquanto as duas linhas inferiores estão cheias, provocará um «crash». O mesmo acontecerá se usar uma instrução PRINT AT.

23660/1, S TOP

Esta variável contém o número de linha onde se inicia a

listagem automática, pelo que a alteração deste valor com um POKE equivale ao uso da instrução LIST número de linha.

23662/3, OLDPPC

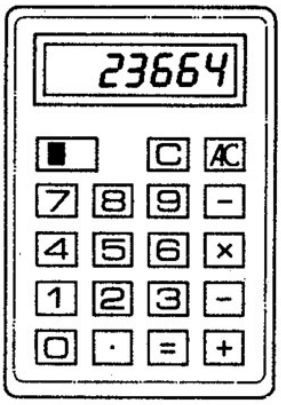
Quando se usa a ordem CONTINUE, o sistema consulta esta variável para conhecer a linha que estava em execução antes de o programa ser parado. Se executarmos:

```
 10 PRINT AT 0,0;"Linha 0"
 20 STOP
100 PRINT "Continua 100"
```

e fizermos POKE 23662,100, POKE 23663,0, escrevendo em seguida CONTINUE, obteremos o resultado esperado.

23664, OSPPC

Esta variável pode ser usada do mesmo modo que OLDPPC, mas indica a instrução da linha que está a ser executada.



23670/1, SEED

Trata-se da «semente» usada para gerar números aleatórios. Introduzindo valores neste endereço obtemos o mesmo efeito produzido pela instrução RANDOMIZE, pelo que:

```
10 RANDOMIZE 1
20 PRINT RND
30 POKE 23670,1: POKE 23671,0
40 PRINT RND
```

produz o mesmo número pseudo-aleatório.

23672, FRAMES

Umà das coisas que parece faltar ao Spectrum é um relógio de tempo real, ou será que não?

Pode usar-se a variável FRAMES para obter tempos relativamente rigorosos, com um erro de 1/50 de segundo. Experimente:

```
  10 LET t=PEEK 23672+(256*PEEK
23673)+(65536*PEEK 23674)
  20 PRINT AT 0,0;INT ((t/50)-60
*INT ((t/50)/60))
  30 GO TO 10
```

para obter segundo. O número de imagens enviadas para o visor é contado em 3 bytes, e o número máximo de 1/50 avos de segundo em 24 horas é 4320000, enquanto o maior número que pode ser guardado nestes três bytes é 16777216. Isto significa que existe «espaço» suficiente para um relógio de 24 horas. Basta determinar os números adequados e introduzi-los em FRAMES. Uma subrotina semelhante à anterior, que produz segundos, poderá então fornecer hora correcta.

23675/6, UDG

Tal como o indicador CHARS, que referencia o conjunto de caracteres, a variável UDG indica a posição dos gráficos

definidos pelo utilizador. Este indicador pode ser usado de diversas maneiras. Primeiro pode ser alterado de modo a indicar um endereço maior, deixando mais espaço livre para uma rotina em código-máquina. Alternativamente, pode ser usado para apontar para um certo número de conjuntos de caracteres diferentes, definidos em RAM. Isto é mais simples do que alterar o conjunto de caracteres standard, dado que USR "A" produz sempre o endereço do conjunto de UDG's indicado por 23675/6, podendo as mesmas rotinas serem usadas para definir diferentes conjuntos UDG's. Mudar de umas para outras obrigará simplesmente a alterar este indicador.

23677/8, COORDS

São aqui guardadas as coordenadas horizontal e vertical do último ponto marcado com a instrução PLOT. Isto aplica-se igualmente à instrução DRAW, sendo portanto possível usar estas ligações para obter uma ordem de movimento absoluto, permitindo reposicionar a PLOT sem necessidade de recorrer a INVERSE. Experimente:

```
10 PLOT 0,0
20 DRAW 10,10
30 POKE 23677,50
40 POKE 23678,60
50 DRAW 10,10
```

23684, DF CC

Esta variável contém o endereço da posição de impressão no ficheiro de imagem, e pode ser usada para construir uma rotina de impressão alternativa.

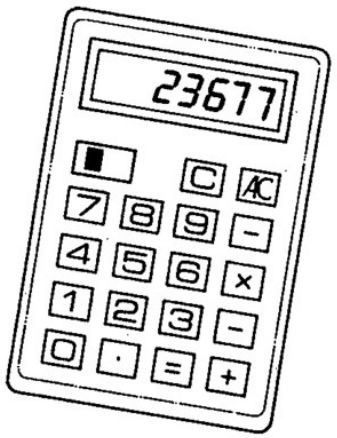
23686/7, DFCCL

Esta variável é geralmente mais útil do que a anterior, pois normalmente indicam o endereço do início da janela inferior do visor. Este endereço pode ser usado, juntamente com outros

já discutidos, para construir uma rotina capaz de imprimir mensagens nesta janela. O programa que se segue faz exactamente isto, usando o indicador CHARS para obter a informação relativa à forma dos caracteres.

```
 10 BORDER 0
 20 DIM a$(31)
 30 LET s=FN a(23686)
 40 INPUT a$
 50 FOR t=1 TO LEN a$
 60 FOR s=0 TO 7
 70 POKE s-1+(s*256)+t,PEEK (C
ODE a$(t TO t)*8+FN a(23606)+s)
 80 NEXT s
 90 NEXT t
100 PRINT "Use BREAK para parar
o programa"
110 GO TO 100
1000 DEF FN a(x)=PEEK x+256*PEEK
 (x+1)
```

Note que se pode usar exactamente a mesma rotina para imprimir no resto do visor, bastando para tal dar a LS o valor 16384 na linha 30.

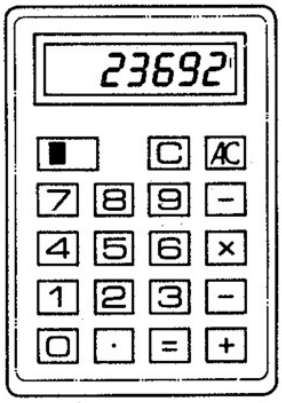


23688/9, S POSN

É aqui guardada a posição de impressão no visor. É estranho que aceite como origem o canto inferior esquerdo do visor, pelo que a vulgar instrução PRINT AT 0,0; "a" levará estes dois endereços a conterem respectivamente 33 e 22 e não 0,0. Pode sofrer POKE's, substituindo a PRINT AT, mas provavelmente isto produzirá um «crash» do sistema.

23692, SCR CT

Esta variável é usada pelo sistema para controlar o rolamento (scrolling) do visor, e quando atinge o valor 1 produz a mensagem «Scrol»? Isto pode ser evitado introduzindo na variável de valor 0 ou 2, através de um POKE, antes de cada instrução de PRINT. Este método pode ser usado para tornar a rotina que aumenta o tamanho do visor (ver DF SZ) um pouco mais segura; basta acrescentar 35 POKE 23692,0 para nos libertarmos da tentação para parar o programa usando BREAK.



Como podemos ver, um bom número de variáveis de sistema podem ser bastante úteis e, apesar de por vezes ser necessário um certo cuidado, não se deve temer a experimentação neste campo, dado que é sempre possível desligar o Spectrum e voltar a ligá-lo. A alteração do conjunto de caracteres e a impressão em locais do visor invulgares podem tornar-se bastante úteis nos nossos próprios programas e, se o leitor alguma vez adquirir a «mania» do código-máquina, verificará provavelmente que as variáveis de sistema mais obscuras lhe permitem fazer algo de verdadeiramente original, o que em última análise é um dos aspectos mais agradáveis da programação.

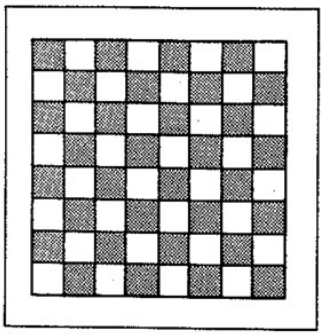
9

GRÁFICOS DEFINIDOS PELO UTILIZADOR

Até certo ponto é bastante fácil compreender a forma como os gráficos definidos pelo utilizador actuam, dado que se trata simplesmente de atribuir oito números binários a uma matriz 8 × 8 que forma cada novo caracter. Por exemplo:

```
10 POKE USR "a"+0, BIN 10101010
20 POKE USR "a"+1, BIN 01010101
30 POKE USR "a"+2, BIN 10101010
40 POKE USR "a"+3, BIN 01010101
50 POKE USR "a"+4, BIN 10101010
60 POKE USR "a"+5, BIN 01010101
70 POKE USR "a"+6, BIN 10101010
80 POKE USR "a"+7, BIN 01010101
90 PRINT "A": REM Udg
```

As oito linhas anteriores introduzirão o padrão de um tabuleiro de xadrez no gráfico "A"; e é fácil compreender o tipo de padrão obtido observando as posições dos uns e zeros. Como é óbvio, dado que estamos a especificar BIN antes de cada número – e se não o fizermos o Spectrum pensará que se trata de um valor decimal, imprimindo uma mensagem "Integer out of range" – podem evitar-se os problemas de escrita con-



vertendo o número para decimal. É igualmente possível POKE números a partir de declarações DATA, usando um ciclo FOR... NEXT:

```
10 FOR n=0 TO 7: READ b: POKE
USR "a"+n,b: NEXT n
20 DATA 170,85,170,85,170,85,1
70,85
```

Se o leitor estiver atento, terá notado um aspecto interessante dos números decimais que não é imediatamente óbvio quando se observa a versão em binário, a saber, que cada número correspondente a um dos bits é igual a metade do imediatamente anterior. Compreenderá porquê observando a versão em binário, onde é cortado o algarismo da extremidade. A razão disto é que dividir um número binário por dois é equivalente a dividir um decimal por dez.

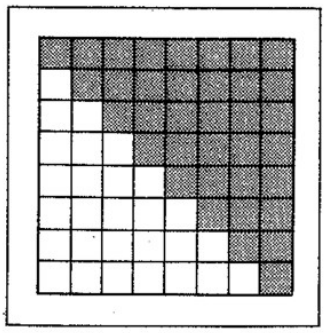
O assunto torna-se ainda mais claro se experimentarmos o seguinte:

```
10 FOR n=0 TO 7: POKE USR "a"+
n,INT (255/(2↑n)): NEXT n
```

Realizando divisões sistemáticas por dois e aplicando INT aos resultados, estaremos a produzir a série de números 255, 127, 63, 31, 15, 7, 3, 1, produzindo uma forma em cunha.

O leitor deveria começar a compreender por que razão é possível usar operações aritméticas com UDG's. A realização de cálculos sobre os números contidos em posições correspon-

dentes e UDG's pode ser importante para obter uma animação – exploraremos este assunto mais tarde, mas será que o leitor já compreende como se consegue isto?



Gráficos e Memória

A dificuldade que muitos utilizadores do Spectrum enfrentam quando estão a tratar gráficos por eles definidos consiste em entenderem-se com o que de facto acontece quando estes são programados. Neste sentido os UDG's são o seu pior inimigo, porque são tão fáceis de programar que o utilizador tende a perder uma oportunidade de aprender alguma coisa sobre a forma como a memória do Spectrum funciona.

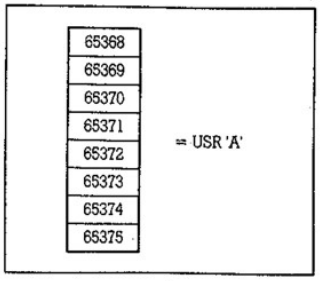
É necessário compreender que a área UDG da memória do Spectrum é apenas mais um «stack». Por exemplo:

```
PRINT USR "A"
```

Esta instrução imprime a localização do primeiro de oito endereços que formam o gráfico A do Spectrum, 65368, pelo que a expressão USR "A" é apenas um modo de evitar a necessidade de recordar uma determinada posição de memória.

Experimente o seguinte:

```
10 FOR n=1 TO 10
20 FOR i=0 TO 7
30 POKE 65368+i,INT (RND*256)
40 NEXT i
50 PRINT AT 10,16;"A": REM car
acter udg
60 NEXT n
```

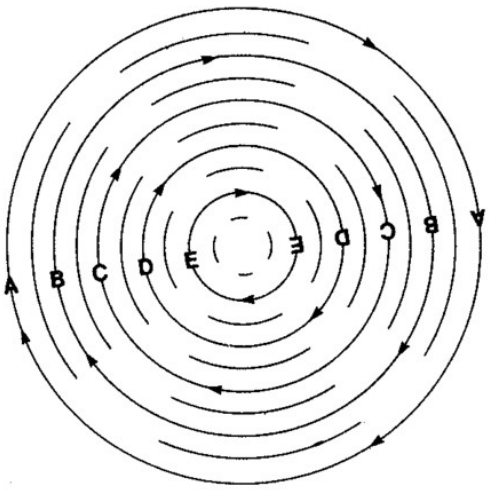


Este pequeno programa redefine o símbolo gráfico A introduzindo nele números aleatórios, e executa o mesmo processo dez vezes – experimente passar ao modo GRAPHICS e escrever A, verificando que 65368 é de facto o mesmo que USR "A". Uma pequena rotina como esta é sempre fácil de introduzir num programa, mas existem modos mais sistemáticos de tratar os UDG's:

```
 1 LET p=167
10 FOR n=0 TO 167
20 POKE USR "a"+n,PEEK (15880+
p)
30 LET p=p-1
40 NEXT n
```

Este programa constitui simplesmente um ciclo que conta N e P, de tal modo que quando N é zero P é 167, e até ao valor 167, de N, ao qual corresponde um N igual a zero. Na linha 20 introduzimos valores nos 168 endereços (21 gráficos vezes 8 posições). O primeiro endereço do caracter A do Spectrum, em ROM, é 15880, e como estes caracteres se encontram organizados exactamente da mesma forma que o conjunto de gráficos definidos pelo utilizador, 15880 mais 167 dará o último endereço do caracter U. Lemos portanto os valores para trás

a partir daqui, e introduzimo-los na área reservada aos UDG's, obtendo deste modo um conjunto de caracteres invertido!



Se realizarmos esta operação no conjunto de UDG's, a propósito, verificamos que a primeira metade deste conjunto é uma imagem ao espelho da segunda metade.

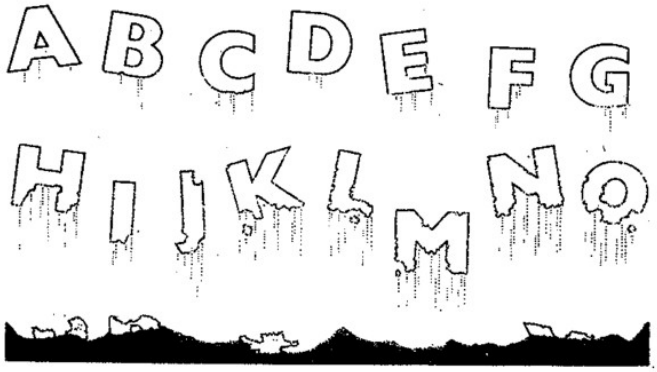
Mas com aquilo que apresentámos aqui não obtemos de facto um conjunto de caracteres completo. Apenas para verificarmos que existe pouca diferença entre a memória UDG e as outras áreas, experimentemos o seguinte:

```
10 CLEAR 59999
20 FOR n=0 TO 1024
30 POKE 60000+n,PEEK (15360+n)
40 NEXT n
50 POKE 23607,234
60 POKE 23606,96
```

Executemos este programa e, desde que não cometamos erros, não encontraremos qualquer diferença. Aqueles que encontrarem diferenças (o que significará que o novo conjunto de caracteres se encontra corrompido) devem escrever:

POKE 23606,0: POKE 23607,60

e verificar novamente. Estamos aqui a reservar espaço acima do endereço 59999 a fim de realizarmos uma cópia completa do conjunto de caracteres do Spectrum. A linha 30 copia os padrões de pontos a partir de 15360 para os endereços acima de 60 000.



As linhas importantes são a 50 e a 60, que alteram o endereço indicado na variável de sistema CHARS, a qual serve para indicar o local onde o Spectrum guarda o seu conjunto de caracteres. Como o endereço em causa é obviamente superior a 255, o maior número que é possível guardar num único byte, tornam-se evidentemente necessárias duas posições de memória. CHARS deve ser calculada do seguinte modo:

PRINT PEEK 23606 + 256 * PEEK 23607

Trabalhando para trás a partir daqui, e como pretendemos apontar CHARS para 60000 (poderíamos preferir que qualquer outro endereço razoável), dividimos 60000 por 256, obtendo 234,375, pelo que o número a guardar em 23607 deve ser 234. Em seguida multiplica-se 234 por 256, subtrai-se de 60 000, e obtemos 96, que é o número que deve ser introduzido em 23606.

Para mostrarmos como se pode utilizar tudo isto, acrescentemos as seguintes linhas:

```
 60 FOR n=96 TO 120
 70 POKE 23606,n
 80 LIST
 90 NEXT n
100 POKE 23606,96
```

Quando dissemos que o conjunto de caracter era corrompido, não era estritamente verdade. As linhas que acrescentámos limitam-se a deslocar de um endereço de cada vez o indicador do conjunto de caracteres, pelo que para cada passagem sucessiva pelo segundo ciclo o conjunto de caracteres desloca-se para cima, até acabar por se tornar completamente ininteligível.

Pode usar-se isto como método de segurança, mas pode tornar-se as listagens igualmente ilegíveis usando a POKE para introduzir números em 23606 e 23607 sem deslocar o conjunto de caracteres.

Dando caracter...

Usando o método citado, é fácil ver que seria bastante fácil levar o Spectrum a usar um conjunto de caracteres alternativo, mas isso obriga a perder algum tempo com papel milimétrico e desenhar um conjunto de caracteres. Ou não?

De facto, não. A menos que se fale do chinês ou do árabe, os conjuntos de caracteres têm muito em comum – o que é lógico, porque se não tivessem não seríamos capazes de os ler. Se portanto pensarmos no que queremos fazer é muitas vezes possível realizar uma operação sistemática sobre o conjunto de caracteres, obteremos um novo tipo de letra sem necessidade de grande trabalho.

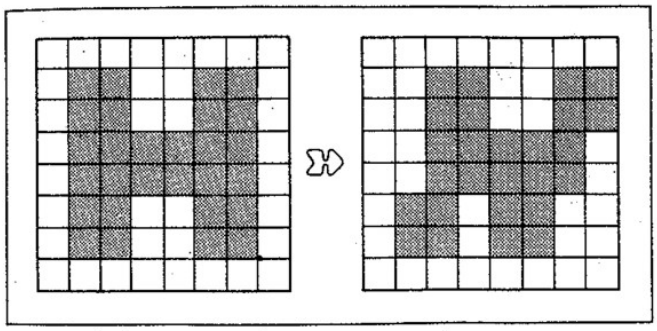
Uma experiência

Consideremos o problema da produção de um tipo itálico. Trata-se essencialmente de um tipo inclinado, pelo que movendo a fiada superior de pontos para um dos lados, e a fiada inferior para o outro, podemos produzir uma simulação aceitável de itálico:

```
 20 CLEAR 64529
 30 FOR t=32 TO 127
 40 FOR s=0 TO 8
 50 LET a=PEEK (15360+(t*8)+s)
 60 IF s>=0 AND s<4 THEN LET a=
a/2
 70 IF s=6 OR s=7 THEN LET a=a*
2
 80 LET a=a-((a>255)*256)
 90 POKE 64273+(t*8)+s,a
100 NEXT s
110 NEXT t
120 GO SUB 1000
140 PRINT "Eis a aparencia que
as letras",
150 GO SUB 2000
160 PRINT "ITALICAS ": GO SUB 1
000: PRINT "tem. ": GO SUB 2000:
 PRINT "OK!"
170 FOR t=0 TO 5
180 GO SUB 1000: PRINT "ABCDEFG
HIJKLM"
190 GO SUB 2000: PRINT "ABCDEFG
HIJKLM"
200 NEXT t
210 STOP
1000 POKE 23606,0: POKE 23607,60
1010 RETURN
2000 POKE 23606,64273-256*INT (6
4273/256)
2010 POKE 23607,INT (64273/256)
2020 RETURN
```

Deveria ser bastante simples descobrir o que se passa neste programa. O endereço 15360 na linha 50 é o início do conjunto de caracteres, mas a parte que nos interessa começa um pouco mais adiante. Não tenho bem ideia do que possa ser um espaço em itálico, mas aceitá-lo-emos...

A linha 60 utiliza S para verificar se o programa está a tratar as três linhas de pixels da parte superior do caracter, e se assim for desloca-se um pixel para a direita dividindo o número por dois. A quarta e quinta linhas são deixadas como estão, e as seguintes são multiplicadas por dois, deslocando-as um pixel para a esquerda.



A linha 80 verifica se o número resultante é demasiado grande para caber num endereço, e se assim for corta 256.

Resta apenas introduzir o novo valor através do POKE da linha 90. O resto do programa mostra-nos a aparência do conjunto de caracteres, e as subrotinas em 1000 e 2000 respectivamente comutam do conjunto de caracteres normal para o conjunto de caracteres itálico. O programa demora um pouco até produzir resultados.

UDG's e visor

Se construírmos os nossos programas usando apenas gráficos definidos pelo utilizador estaremos de facto a perder alguma coisa. É obviamente fácil construir as imagens de um programa usando blocos de gráficos deste tipo, mas o Spectrum dispõe de outras características que nos podem ser vantajosas. Usando o rolamento de imagem, por exemplo, podemos produzir facilmente efeitos interessantes.

Uma experiência

Como se sabe, o visor do Spectrum encontra-se organizado de uma forma bastante complexa, pelo que a sua manipulação pode ser difícil. O programa que se segue, no entanto, incorpora

duas rotinas em código-máquina que rolam os terços superior e inferior da imagem. Foram organizadas assim a fim de permitir o uso separado destas rotinas se assim o quisermos:

```
  10 CLEAR 59999
  20 GO SUB 500
  30 LET x=10: LET z=16
  40 FOR n=60000 TO 60033
  50 READ e
  60 POKE n,e
  70 NEXT n
 100 LET y=40: LET yy=120
 110 LET d=INT (RND*2): LET pp=(
1 AND d=1)+(-1 AND d=0)
 120 LET c=INT (RND*2): LET p=(1
 AND c=1)+(-1 AND c=0)
 130 IF y=0 THEN LET p=1
 140 IF y=40 THEN LET p=-1
 150 IF yy=174 THEN LET pp=-1
 160 IF yy=115 THEN LET pp=1
 170 LET y=y+p: LET yy=yy+pp
 180 PLOT 0,y: DRAW 0,-y: PLOT 0
,yy: DRAW 0,175-yy
 190 PRINT AT x,z;" "
 200 IF INKEY$="1" THEN LET x=x-
1
 210 IF INKEY$="q" THEN LET x=x+
1
 220 PRINT AT x,z;"A": REM carac
ter udg
 230 LET b=USR 60000: LET a=USR
60017
 240 GO TO 110
 499 REM definir udg
 500 FOR n=0 TO 7
 510 READ e
 520 POKE USR "a"+n,e
 530 NEXT n
 540 DATA 7,30,124,255,124,30,7,
0
 550 RETURN
 680 DATA 33,0,80,62,63,6,32,183
,203,30,35,16,251,61,32,245,201
 690 DATA 33,0,64,62,63,6,32,183
,203,30,35,16,251,61,32,245,201
```

Os elementos cruciais deste programa são as linhas 680 e 690. Trata-se basicamente da mesma rotina em ambas, excepto no que se refere ao terceiro bloco de dados, que determina

o endereço do ficheiro de imagem onde a rotina se inicia. No caso da primeira rotina o número é 680, e se se usar o mesmo método a que já recorremos para tratar de CHARS, isto é, multiplicar por 256 e somar o número que se encontra no endereço anterior, obtemos 20480, que é o endereço inicial do terço inferior do visor. A mesma operação realizada sobre a rotina da linha 690 dá-nos 16384, endereço inicial do terço superior do visor.

Daqui deduz-se que estamos a rolar a parte superior e a inferior do visor independentemente, deixando a nave espacial, formada por UDG's, no meio. Usamos PLOT e DRAW para produzir o cenário, o que torna o programa um pouco lento, mas poderíamos apressá-lo não preenchendo as manchas do cenário, podendo-se ainda acrescentar uma rotina que verificasse quando a nave bate nas paredes da caverna. Isto envolveria apenas verificar se os atributos de X,y são ou não iguais a um espaço quando se imprime a nave.

No caso de se querer aproveitar as rotinas de «scrolling» para uso noutros jogos, convirá notar que a correspondente ao terço inferior é invocada por LET B=USR 60000 e a da parte inferior por LET A=USR 60017. Poder-se-ia usar igualmente RANDOMIZE USR em vez de LET A=USR.

Resumo

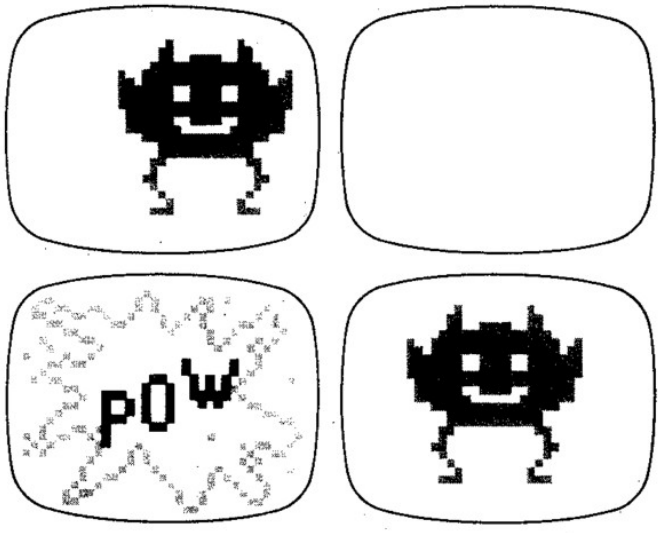
Neste capítulo aprendemos:

1. Que USR "A" é apenas uma forma de indicar um dos endereços da memória do Spectrum, e a forma como os padrões de pontos que formam os caracteres se encontram armazenados na área de UDG's.
2. Como se pode manipular a forma de um caracter no visor realizando operações aritméticas sobre o número guardado na memória.
3. Como introduzir conjuntos de caracteres completamente novos alterando o valor armazenado na variável de sistema CHARS, produzindo tantos gráficos do utilizador quantos os que se queira.
4. Como usar rotinas de «scrolling» misturadas com UDG's para produzir jogos simples.

10

SPRITES E ANIMAÇÃO

«Animação» é basicamente uma forma de levar imagens a moverem-se no visor. Isto pode ser feito a diversos níveis, sendo o mais simples o uso de uma rotina Basic que imprima um caracter no visor, o apague, e o imprima em seguida na posição seguinte, como neste exemplo:




```
 10 PRINT AT 10,0;"M";
 20 FOR t=0 TO 30
 30 PRINT AT 10,t;" ";
 40 PRINT AT 10,t+1;"M";
 50 NEXT t
 60 FOR t=31 TO 0 STEP -1
 70 PRINT AT 10,t;" ";
 80 PRINT AT 10,t-1;"M";
 90 NEXT t
100 GO TO 10
```

Depois de escrever o programa, o leitor obterá um "M" movendo-se para trás e para a frente no visor. O problema deste método é que, como leitor terá notado, produz um movimento descontínuo, «apagando» e «acendendo» continuamente o caracter. A ideia geral, no entanto, constitui a base dos «sprites» e do movimento de qualquer tipo de figura.

O visor do Spectrum

Para termos uma ideia da forma como os «sprites» funcionam, convém termos uma ideia mais pormenorizada do que é o visor do Spectrum.

A imagem é formada por uma série de linhas horizontais, cada uma das quais se apresenta novamente dividida num certo número de pontos. Estas linhas e pontos dão a resolução da imagem em pixels. O visor do Spectrum, tal como acontece em outros microcomputadores, encontra-se ainda dividido em duas secções principais, a margem e o visor principal. Quando este é impresso, 50 vezes por segundo, o feixe de varrimento do televisor é de facto ligado e desligado conforme a posição em que se encontra e o conteúdo do ficheiro de imagem. O hardware responsável por realizar todo o tratamento de dados do visor é o famoso circuito ULA da Sinclair.

Durante a primeira parte do varrimento, o ULA limita-se a enviar um sinal de cor, que forma a parte superior da margem. Assim que é atingida a parte do visor capaz de fornecer uma imagem, o ULA envia primeiro o sinal de margem e depois escrutina a RAM vídeo, ou ficheiro de imagem, enviando para o televisor a informação relativa a um ponto quando encontra

um bit contendo um, ou um espaço quando encontra um bit contendo zero. Depois de a linha ter sido impressa, o ULA volta a enviar o sinal de margem, continuando do mesmo modo até atingir a margem inferior, momento em que se limita a enviar um simples sinal de cor.

Além de varrer a parte de imagem da RAM, entre 16384 e 22527, o ULA varre igualmente a secção de atributos, entre 22528 e 23295, de modo a produzir as cores no visor. Os sinais de cor levam ao envio para os diferentes canhões do visor da informação sobre a cor da imagem.

Quando se movem caracteres pelo visor, observa-se muitas vezes uma tremura dos caracteres causada pela interferência entre o varrimento do visor realizado pelo ULA e a escrita de nova informação no ficheiro de imagem pela CPU. Como o objecto deve ser apagado e depois impresso de novo a fim de dar a aparência de movimento, o varrimento efectuado pelo ULA apanhará provavelmente a imagem a meio caminho durante a parte do programa que a apaga, ou em algum ponto durante o seu redesenhar, dando origem a fantasmas e severas instabilidades da imagem.

Felizmente, nem tudo está perdido, dado que existem momentos em que o ULA não está de facto a varrer a RAM vídeo, por exemplo quando está a imprimir a margem, ou quando o feixe volta ao topo da imagem. É nestes momentos que a imagem deve ser modificada.

Voltando aos «sprites», que poderemos definir como elementos gráficos em movimento ou transformação, envolvendo possivelmente um espaço de imagem correspondente a várias posições de caracter, vamos apresentar um conjunto de rotinas em código-máquina que nos permitem colocar caracteres em qualquer ponto do visor (em qualquer das posições horizontais 0 a 255 e das verticais 0 a 175).

O programa que se segue é usado para carregar em memória o código-máquina, armazenando-o em seguida em fita ou microdrive.

```
  10 REM programa que carrega sp
rites
  20 FOR t=30022 TO 30175
  30 READ a: POKE t,a
  40 NEXT t
  50 SAVE *"M";1;"SPRT.BIN"CODE
30000,175
  60 STOP
 100 DATA 205,129,117,205,119,11
7,201,205,170
 110 DATA 117,201,42,68,117,1,56
,117,22,8,126,2
 120 DATA 8,35,21,32,249,201,205
,170,117
 130 DATA 58,66,117,50,64,117,58
,67,117,50,65
 140 DATA 117,205,129,117,205,11
9,117,201,205
 150 DATA 81,117,33,56,117,205,1
73,117,201,33
 160 DATA 48,117,237,75,64,117,2
2,8,30,8
 170 DATA 197,213,229,205,205,34
,205,213,45
 180 DATA 225,209,193,203,47,203
,22,12,29
 190 DATA 32,236,35,5,121,214,8,
79,21,32
 200 DATA 225,201,33,48,117,237,
75,64,117
 210 DATA 22,8,30,8,203,38,218,1
91
 220 DATA 117,62,12,50,145,92,19
7,213,229
 230 DATA 205,229,34,225,209,193
,62,0,50
 240 DATA 145,92,12,29,32,228,12
1,214,8,79
 250 DATA 35,5,21,194,179,117,20
1
 260 DATA 80,11,114,99,101
```

Este código-máquina permite o deslocamento de qualquer caracter no visor. Antes de o «sprite» ser deslocado para uma dada posição, são guardados os dados relativos a esta posição. Depois de o «sprit»e ter saído dela, os dados são repostos de modo a não corromper a imagem de fundo.

Para usar o programa é necessário ter em conta os seguintes endereços importantes:

30020 (ADL) guarda o byte menos significativo do «sprite».

30021 (ADH) guarda o byte mais significativo desse mesmo endereço.
30016 (X) guarda a posição inicial do «sprite«, na horizontal.
30017 (Y) guarda a posição inicial do «sprite», na vertical.
30018 (X1) guarda a coordenada horizontal da posição para onde o «sprite» deve ser deslocado usando MSPR.
30019 (Y1) guarda a coordenada vertical da posição para onde o «sprite» deve ser movido.
30022 (SPON) A execução desta rotina «liga» o «sprite».
30029 (SPOF) Esta rotina «desliga» o «sprite».
30049 (MSPR) Esta rotina move o «sprite» de X,Y para X1,Y1, e quando terminada substitui X por X1 e Y por Y1.

Com estas informações, o leitor dispõe de tudo aquilo de que necessita para poder deslocar o «sprite» no visor. Note que, na medida em que a posição especificada por X e Y é a posição de um pixel, o caracter pode ser deslocado para qualquer ponto do visor.

Vejamos uma demonstração:

```
  10 GO SUB 1000: REM carregar
codigo-maquina
  20 LET x=30016
  30 LET y=30017
  40 LET x1=30018
  50 LET y1=30019
  60 LET cl=30020
  70 LET ch=30021
  80 LET spon=30022
  90 LET spof=30029
 100 LET mspr=30049
 110 LET dx=1: LET dy=1
 120 LET sx=10: LET sy=10
 130 POKE ch,255: POKE cl,88
 140 PRINT x,sx: POKE y,sy
 150 RANDOMIZE USR spon
 160 LET sx=sx+dx: LET sy=sy+dy
 170 IF sx>240 OR sx<10 THEN LET
dx=-dx
 180 IF sy>160 OR sy<10 THEN LET
dy=-dy
 190 POKE x1,sx: POKE y1,sy
 200 RANDOMIZE USR mspr
 210 GO TO 160
1000 LOAD *"M";1;"SPRT.BIN"CODE
1010 RETURN
```


Apenas para provar que o fundo não será apagado, acrescentemos a linha seguinte:

```
105 FOR t=0 TO 600: PRINT "b";:
NEXT t
```

A outra utilidade das rotinas de «sprites» consiste em permitir a impressão de caracteres normais em qualquer ponto do visor experimentemos o seguinte:

```
  10 GO SUB 1000: REM carregar
codigo-maquina
  20 LET x=30016: LET y=30017: L
ET x1=30018: LET y1=30019: LET c
l=30020: LET ch=30021: LET spon=
30022: LET spof=30029: LET mspr=
30049
 110 FOR t=65 TO 85
 120 POKE cl,(USR CHR$ (t))-256*
INT (USR CHR$ (t)/256)
 130 POKE ch,INT (USR CHR$ (t)/2
56)
 140 POKE x,FN r(200)+10
 150 POKE y,FN r(100)+10
 160 RANDOMIZE USR spon
 170 NEXT t
 180 DEF FN r(x)=INT (RND*x)-1
 999 STOP
1000 LOAD *"M";1;"SPRT.BIN"CODE
1010 RETURN
```

Alterando o endereço introduzido em CH e CL de modo a indicar o conjunto de caracteres em ROM, definido através da soma do código do caracter a

(33 * 8) + 256 + PEEK 23606 + 256 * PEEK 23607

pode imprimir-se qualquer caracter em qualquer posição do visor.

Animação de todo o visor

Usa-se ainda outro tipo de animação, consistindo numa sequência de instantâneos de uma imagem, cada um deles ligeiramente diferente do anterior. No Spectrum, podemos obter este efeito desenhando o primeiro instantâneo no visor, e trans-

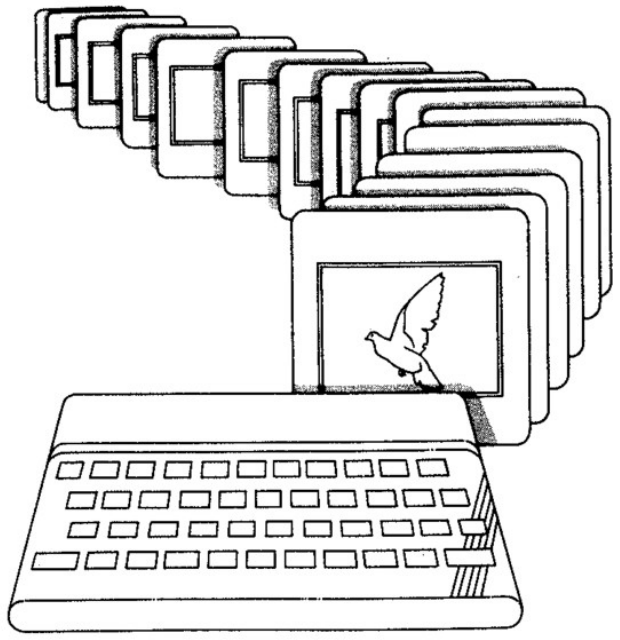
ferindo depois toda a imagem para a memória. Desenha-se depois a imagem seguinte, e assim por diante.

Todas as imagens são transferidas para a memória, constituindo uma série que pode ser passada sequencialmente para o ficheiro de imagem. Existem dois inconvenientes principais neste processo, sendo o primeiro que cada imagem ocupa uma grande quantidade de memória (cerca de 7 K). O segundo inconveniente é que, usando a Basic, seria necessário muito tempo para transferir os dados.

Felizmente, a velocidade a que isto pode ser feito em código-máquina é bastante maior. O microprocessador que se encontra no «coração» do Spectrum dispõe de uma instrução especial para cópia de partes da memória de umas posições para outras a alta velocidade, pelo que a rotina que deveremos escrever se torna bastante curta.

Se o leitor experimentar o programa seguinte, verificará que as imagens podem ser substituídas com bastante rapidez:

```
 10 CLEAR 50000
 20 LET swap=50000
 30 GO SUB 200
 40 FOR t=0 TO 255 STEP 5
 50 PLOT 0,0
 60 DRAW t,175
 70 NEXT t
 80 RANDOMIZE USR swap
 90 CLS
100 FOR t=-255 TO 0 STEP 5
110 PLOT 255,0
120 DRAW t,175
130 NEXT t
140 RANDOMIZE USR swap
150 GO TO 140
200 FOR t=swap TO swap+29
210 READ a: POKE t,a
220 NEXT t
230 RETURN
240 DATA 33,0,64,17,0,224,1,0
250 DATA 27,126,245,25,119,241,
18,11
260 DATA 35,19,120,177,32,243,1
77,32
270 DATA 243,201,107,101,110,11
0
```

A primeira imagem, depois de desenhada, é guardada na memória usando RANDOMIZE USR SWAP, para um local onde não se encontra qualquer outra imagem. A seguinte é desenhada e, em seguida, trocada pela anterior.

Um aspecto a notar é que esta rotina pode ser «relocatada», isto é, colocada em qualquer ponto da RAM bastando alterar o valor de SWAP. No entanto, devemos ter ainda em conta a necessidade de alterar a instrução CLEAR. Este método não é porém o mais rápido, e além disso a imagem treme.

Um método muito melhor consiste em usar a sequência de programas que vamos apresentar para guardar as imagens em memória e chamá-las depois ao visor.

O primeiro programa carrega o código-máquina a partir de uma série de declarações DATA, gravando depois a secção

apropriada da memória em fita ou microdrive. O código poderá ser novamente carregado em qualquer momento usando:

LOAD *"M";1;"TRN.BIN"CODE 65280

ou no caso da fita:

LOAD "TRN.BIN" CODE 65280

Esta rotina em código-máquina é usada para copiar uma secção da memória cujo endereço deve ser colocado nos endereços 65280 e 65281. O primeiro guarda o byte menos significativo do endereço, obtido a partir de:

END.BAIXO=END−256*INT(END/256)

O endereço 65281 guarda o byte mais significativo, obtido do seguinte modo.

END.ALTO=INT(END/256)

Depois de estes dois números terem sido introduzidos, o programa em código-máquina passa a conhecer a localização para onde se devem deslocar os dados; ao fazermos RANDOMIZE USR 65282 executamos este código, deslocando a porção de memória para o visor. Notemos que a memória de atributos não é usada, permitindo à rotina funcionar ainda um pouco mais rapidamente.

Vejamos agora o programa carregador:

```
   9 REM carregar codigo-maquina
, e gravar como "trn.bin". Usado
 para transferir da memoria para
 o visor
  10 FOR t=65280 TO 65295
  20 READ a: POKE t,a
  30 NEXT t
  40 SAVE *"M";1;"TRN.BIN"CODE 6
5280,15
  50 DATA 192,165,42,0,255,17,0,
64,1,0
  60 DATA 24,237,276,201,80,11,1
14,99,101
```


O programa que se segue demonstra a forma como cada parte da animação é construída e copiada para a memória. Existe espaço em memória para um máximo de cinco a seis imagens sem atributos dependendo do comprimento do programa usado para as desenhar. Este programa necessita de bastante tempo para copiar a imagem de um ponto para outro, e demonstra a extraordinária velocidade do código-máquina Z80 quando comparado com a Basic. A cópia pode também ser realizada em código-máquina, usando-se para tal uma rotina muito semelhante à usada para invocar as imagens no programa anterior:

```
ORG 65280
início DEFW 0      ;reservar alguma memória para o endereço

LD HL,16284        ;colocar o endereço do ficheiro de imagem
                    no registo HL

LD DE, (início)    ;carregar no registo DE o endereço da memória
                    para onde a imagem deve ser transferida.

LD BC,6144         ;carregar em BC o comprimento da imagem, 6144 bytes

LDIR               ;é esta instrução que realiza a transferência
                    de memória, usando os registos
                    HL, DE e BC.

RET                ;voltar à Basic
```

O leitor não se deverá preocupar muito se não compreender a listagem anterior. Basta-lhe com efeito saber como usar esta rotina.

Para tal, substitui a subrotina do programa de desenho já apresentado, que se inicia na linha 100, por:

```
  100 LET ad=30000+(s*6192)
  110 POKE 65280,ad-256*INT (ad/2
56)
  120 POKE 65281,INT (ad/256)
```

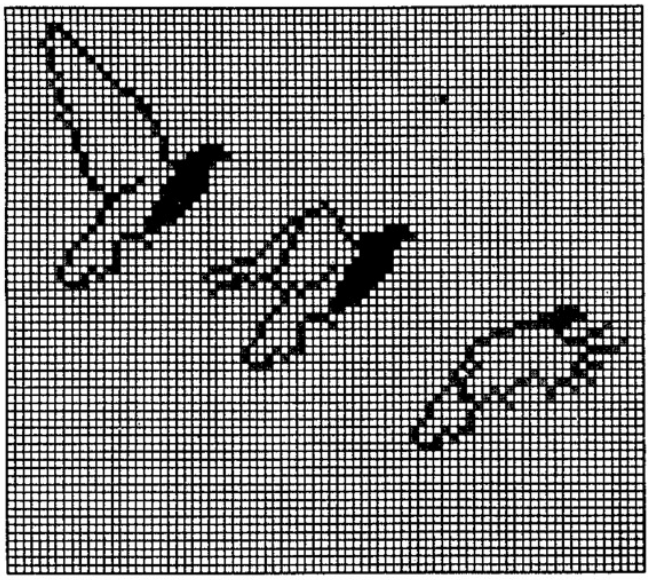
```
130 RANDOMIZE USR 65282
140 RETURN
```

e acrescente ainda as seguintes linhas:

```
  5 GO SUB 200
200 FOR t=65280 TO 65295
210 READ a: POKE t,a
220 NEXT t
230 RETURN
240 DATA 0,0,33,0,64,237,91,0,2
55,1,0
250 DATA 24,237,176,201,74,111,
104,110
```

Todo o processo de transferência se torna agora bastante mais rápido.

```
10 FOR t=10 TO 50 STEP 10
15 LET s=(t/10)-1
20 CLS
```



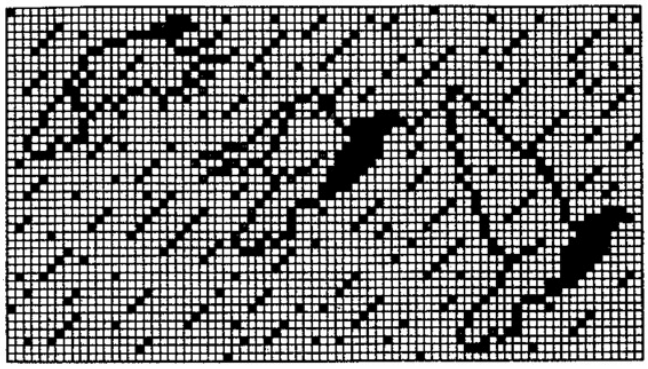

```
 30 CIRCLE 100,100,t: CIRCLE 15
0,90,t/3: CIRCLE 140,110,t/2
 40 PRINT "Imagem n. ";s: PAUSE
100
 50 GO SUB 100
 60 NEXT t
 70 SAVE *"M";1;"CHUVA"CODE 300
00,35280
 80 STOP
 99 REM ler dados da imagem par
a a memoria
100 FOR d=0 TO 6144
110 POKE 30000+d+(6912*s),PEEK
(16384+d)
120 NEXT d
130 RETURN
```

Programa de demonstração

É necessário em seguida animar as imagens. Isto significa que se deve carregar no ficheiro de imagem os vários blocos de dados, sequencialmente, usando a rotina em código-máquina já apresentada.

```
 10 CLEAR 30000
 20 GO SUB 100
 30 FOR t=30000 TO 30000+(4*691
2) STEP 6912
 40 POKE 65280,t-256*INT (t/256
)
 50 POKE 65281,INT (t/256)
 60 RANDOMIZE USR 65282
 70 NEXT t
 80 GO TO 30
 99 REM carregar codigo-maquina
e dados da imagem
100 PRINT "ESTOU A CARREGAR COD
IGO"
105 LOAD *"M";1;"TRN.BIN"CODE 6
5280
110 LOAD *"M";1;"CHUVA"CODE 300
00
120 RETURN
```

Depois de executar este programa, compreenderá por que razão o programa é chamado «CHUVA».



PROJECTOS

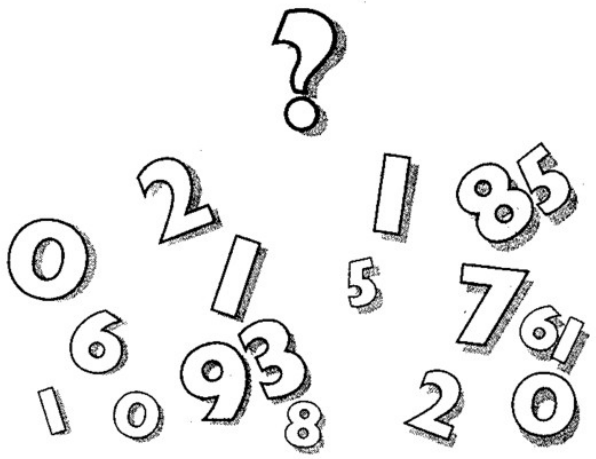
1. Tente construir uma sequência de animação mais completa.
2. Altere a parte Basic da rotina de «sprites» de modo a poder usar mais do que um «sprite».
3. Altere o programa de animação do visor de modo a incluir o ficheiro de atributos, obtendo cores.
4. Inclua este processo de animação do visor no programa de aventuras descrito no capítulo 3, de modo a conseguir representar imagens das várias salas, em vez de se limitar a descrevê-las com texto.
5. Verifique se consegue codificar os dados relativos às imagens de modo a permitir a inclusão de um maior número de imagens. Uma sugestão: note que muitos bytes seguidos, na imagem, contêm os mesmos valores (por exemplo zero, quando não contêm qualquer desenho), e que é possível contá-los em vez de os guardar a todos em memória.

11

A MEMÓRIA DO SPECTRUM

Várias vezes ao longo deste livro o leitor encontrou frases do tipo «não é necessário que o leitor compreenda já isto». Muitas vezes estas frases referiam-se a conjuntos de números indicados em declarações DATA que, quando são introduzidos na memória, parecem fazer coisas incríveis, e muito mais depressa do que a Basic.

O leitor já terá provavelmente visto a mesma coisa em revistas, e neste caso só lhe é dito geralmente que se trata de «programas em código-máquina». Se pretende saber mais



alguma coisa sobre o que é o código-máquina esta simples informação não lhe é de grande utilidade. Todos esses números terão certamente algum significado!

Para termos uma ideia do que se passa devemos dar um passo atrás e pensarmos no que o Spectrum é de facto, e na forma como a sua memória se encontra estruturada. Trata-se essencialmente de uma série de interruptores que podem estar «ligados» ou não, e que interactuam entre si de modo a guardarem números. De facto o Spectrum só é capaz de guardar números, pelo que quando está a «guardar» texto, gráficos, etc., está apenas a armazenar em memória uma representação numérica daqueles.

Nestas condições, sempre que se escreve uma instrução Basic o interpretador alojado no Spectrum deve traduzir essa instrução para que a máquina possa actuar sobre ela, e esta tradução é realizada de tal modo que a instrução é transformada num número, ou numa série de números, e executada em seguida. É esta a principal razão de a Basic ser bastante lenta quando comparada com o código-máquina.

O que estamos a fazer quando introduzimos dados numéricos decimais em declarações DATA é falar ao Spectrum na linguagem que ele entende, e de facto os números em causa constituem uma série de instruções que formam um programa em código-máquina. O leitor já sabe que o código-máquina deve ser guardado em memória acima de um endereço designado por RAMTOP, e que portanto deve baixar esta RAMTOP de modo a dispor de espaço suficiente acima dela para guardar os seus programas (basta-lhe descer a RAMTOP do número de bytes equivalente à quantidade de números indicados nas declarações DATA); sabe ainda que para executar o código deve usar a instrução RANDOMIZE USR (endereço), onde «endereço» é a primeira posição acima da RAMTOP.

Porquê RANDOMIZE USR? É vulgar nos microcomputadores disporem de uma instrução EXEC ou CALL para executarem código-máquina, sendo estas palavras muito mais claras no seu significado («executar», «chamar»). Já não acontece o mesmo no caso do Spectrum, mas quando o leitor tiver compreendido o que se passa considerará as suas instruções também suficientemente claras.

Se consultar o manual verificará que RANDOMIZE é usada como um indicador, e que ao acrescentar USR está a indicar ao Spectrum que deseja apontar para um determinado endereço. O Spectrum salta então para esse endereço, lê a primeira instrução e executa-a.

Sistema hexadecimal

Não entre em pânico se continua a não perceber o significado dos números. Em termos gerais o leitor tem razão; como os computadores pensam em termos de interruptores, apesar de aceitarem números decimais como instruções, não se encontram de facto organizados de uma forma particularmente lógica. No entanto, se traduzisse estes números para a base 16, ou seja, para o sistema numérico hexadecimal, começaria a aperceber-se de que existe algo de organizado neles.

Assembler

Mesmo que consiga dominar as tabelas de multiplicação em hexadecimal, continuará a enfrentar ainda um problema de comunicação. Terá ainda o problema de pensar em português e de tentar comunicar directamente com algo que pensa em números. É precisamente este problema que é resolvido pelos programas «Assembler». Um Assembler é um programa que lhe permite utilizar mnemónicas fáceis de recordar e as traduz para a linguagem numérica que o computador utiliza. Note porém que existe uma diferença fundamental entre a linguagem Assembler e a linguagem Basic.

No caso da Basic, o utilizador fornece à máquina uma série de instruções que ele armazena e interpreta depois, uma de cada vez; no caso da Assembler, está-se de facto a introduzir directamente a informação em memória, como se se usasse a instrução POKE, apesar de não parecer que é isso que se faz. Este livro não tentará explicar código-máquina ou programas Assembler, mas é importante que o leitor compreenda o que estes são, quanto mais não seja para referência futura.

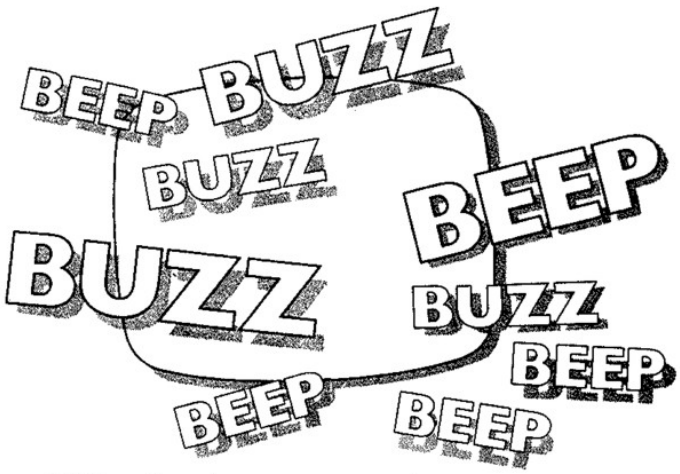
Uma experiência

O som é um bom exemplo daquilo que se pode fazer em código-máquina entrando directamente em contacto com a memória do Spectrum. Se já experimentou as instruções de produção de som do Spectrum terá provavelmente ficado bastante desiludido, particularmente se já ouviu o som produzido por outras máquinas.

O problema mais importante do som do Spectrum é o facto de todas as suas operações serem controladas pelo seu processador Z80. Isto significa que o som é apenas mais uma das operações executadas pela CPU, enquanto em muitos outros micros se trata de uma função controlada por um processador separado. O resultado disto é que, no Spectrum e quando se programa em Basic, tudo pára enquanto a máquina produz sons.

Em código-máquina, no entanto, é possível tornear o facto. Podem usar-se interrupções para produzir ruído enquanto o programa parece continuar em execução – se bem que de facto pare brevemente a intervalos regulares, e se possam produzir também versões do tipo de sons que geralmente é associado aos jogos de acção, como no seguinte caso:

```
  10 CLEAR 65205
  20 FOR x=65206 TO 65260
  30 READ a
  40 POKE x,a
  50 NEXT x
  60 DATA 58,72,92,31,31,31,230,
7,14,255
  70 DATA 38,0,68,203,231,211,25
4,16,254,68
  80 DATA 203,167,211,254,16,254
,203,231,211,254
  90 DATA 16,254,203,167,211,254
,16,254,36,13
 100 DATA 32,226,201
 110 DATA 58,72,92,31,31,31,230,
7,225,229
 120 DATA 95,14,0,22,15,126,230,
16,131,211
 130 DATA 254,65,16,254,35,21,32
,243,13,32
 140 DATA 238,201
 150 FOR n=1 TO 3: RANDOMIZE USR
 65206: RANDOMIZE USR 65247: NEX
T n
```

O leitor já terá certamente pensado em programas como este. Armazena uma rotina – neste caso duas – acima da RAMTOP, e chama-a com uma RANDOMIZE USR para o endereço onde se inicia. Neste caso temos dois programas em código-máquina que produzirão o som de um laser e uma explosão, neste último caso alterando a margem a fim de sublinhar a intenção.

Se agora contar os números contidos nas declarações DATA verificará que a primeira rotina, que termina em 65246, termina de facto no final da linha 100. A segunda termina no final da 140, e o leitor conseguirá certamente observar uma semelhança – ambas terminam com o número 201. Se observar o conjunto de instruções Z80, verificará que este número corresponde à instrução RET, ou retorno. Notará ainda que existem outros números que se repetem, e se tenciona interessar-se por código-máquina acabará por se familiarizar com eles; mas por agora interessa apenas que constate a existência de certos padrões na organização destes números.

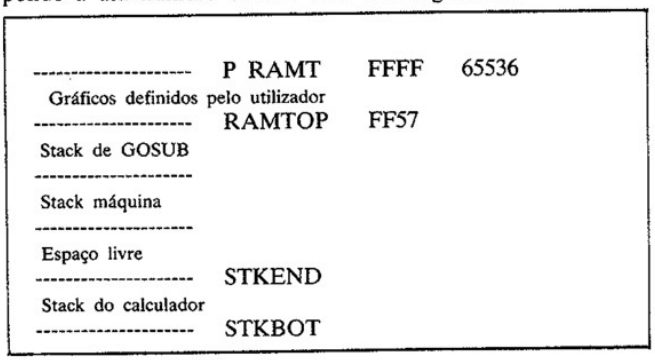
No caso do programa aqui apresentado o leitor deveria detectar uma diferença entre o que é possível obter em Basic e o que se consegue em código-máquina. Em Basic podemos

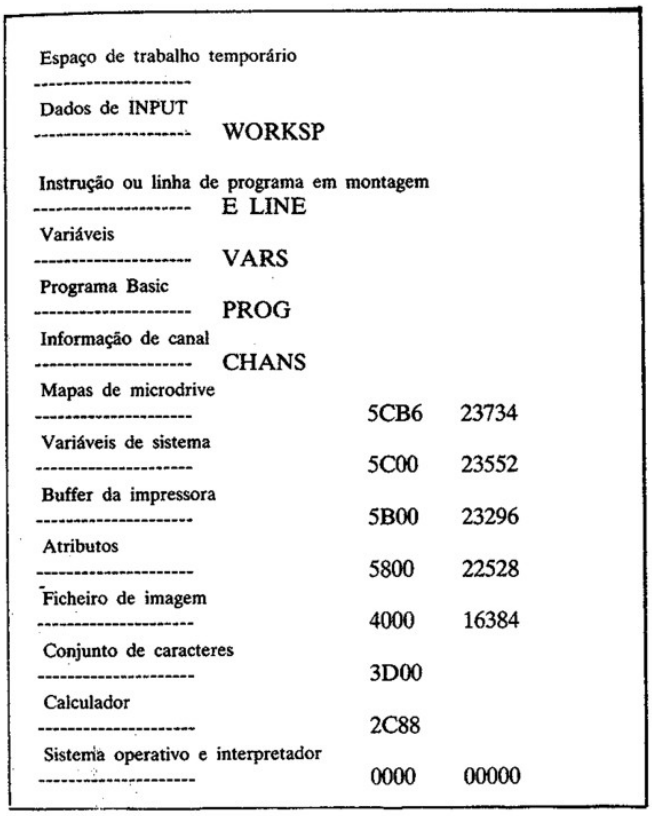
usar um ciclo FOR... NEXT para produzir uma série de notas, mas trata-se de notas separadas – não se consegue obter uma variação gradual dos tons. Mas o código-máquina é deste ponto de vista muito mais flexível.

O BEEP do Spectrum é basicamente apenas um ruído curto. O microfone encontra-se ligado a um dos «ports» de saída do Z80, e sempre que o bit D4 que lhe corresponde é passado ao valor 1 ouve-se esse ruído. A tonalidade da nota que se ouve é determinada pelo número de vezes por segundo que D4 passa ao valor 1 e ao valor zero. Assim, em vez de variarmos a tonalidade e a duração em Basic, podemos alterar a velocidade dos «clicks» produzidos em código-máquina, o que nos permite obter uma variação mais gradual da nota.

O mapa de memória

O diagrama que se segue mostra a forma como se encontra organizada a memória do Spectrum, e como os números decimais se relacionam com as suas versões hexadecimais. É lógico que tratemos aqui na numeração hexadecimal. A memória pode ser considerada como uma longa fila de caixas numeradas, de 0000h (**h** designa hexadecimal,) até FFFFh (7FFFh no caso do Spectrum de 16 K). Cada uma destas caixas contém um byte de oito bits, que do ponto de vista dos simples mortais corresponde a um número binário com oito algarismos.





A memória encontra-se dividida entre ROM (Read Only Memory) e RAM (Random Access Memory). Pode alterar-se aquilo que se encontra em RAM, mas não se pode modificar a ROM – sendo no entanto possível usar algumas das suas rotinas como indicámos no capítulo sobre as variáveis de sistema.

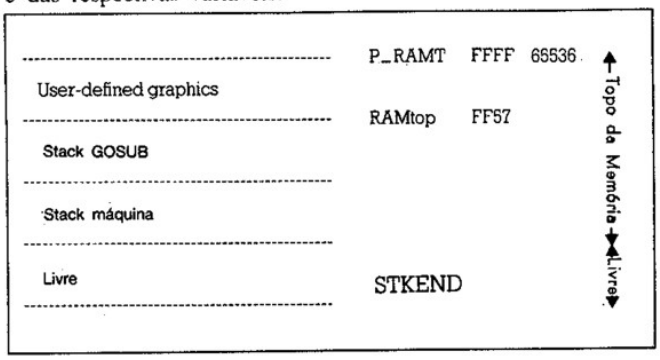
A ROM ocupa as posições de memória entre 0000 e 3FFFh. Trata-se basicamente de um conjunto de programas escritos

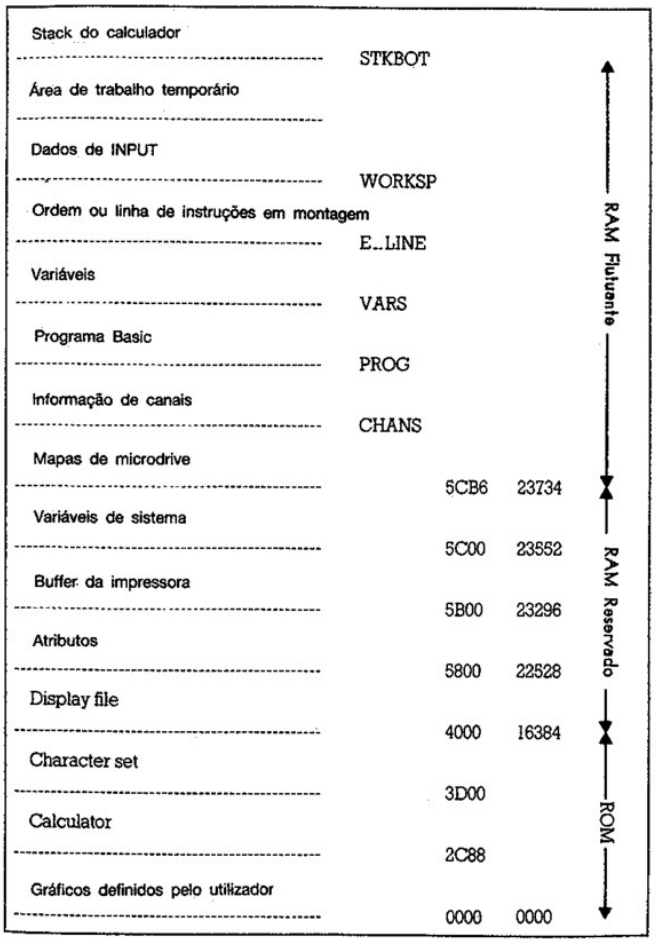
em linguagem-máquina Z80, podendo dizer-se que é aquilo que transforma o Spectrum num Spectrum em vez de ser, por exemplo, um Memotech. Se vier a conhecer bem a linguagem Assembly, verificará que várias partes do programa em ROM podem de facto ser usadas como rotinas nos seus programas.

Se quiser dedicar-se verdadeiramente a este assunto, necessitará de uma lista das instruções existentes em ROM, já traduzidas para as mnemónicas Assembly. Existem vários livros com esta listagem, e apesar de não serem muito fáceis de compreender certamente que um pouco de dedicação lhe trará grandes benefícios.

Imediatamente a seguir à ROM encontra-se a RAM, mais exactamente a área desta que é reservada pelo sistema operativo. É constituída por endereços fixos, usados pela ROM para tarefas como a formação dos ficheiros de imagem e de atributos, já tratados no capítulo sobre a cor. No que nos diz respeito, a outra área mais importante é aquela que contém as variáveis usadas pela ROM para controlar o Spectrum – as chamadas variáveis de sistema.

Passando esta secção atingimos a RAM livre, flutuante, onde as secções não têm limites definidos, apesar de a ROM conhecer estes limites em cada situação através dos valores que se encontram guardados nas variáveis de sistema. Estas secções incluem informações de canal e mapas de Microdrive, e precedem as áreas de armazenamento de programas Basic e das respectivas variáveis.





Em seguida encontram-se as secções que tratam a montagem de instruções, a área de trabalho temporária e o «stack» do

calculador, associadas à execução de programas Basic. O espaço livre acima disto tem dimensões variáveis, em função do tamanho do programa Basic e da quantidade de variáveis que este utiliza, mas pode ser usado para guardar dados ou para programas em código-máquina.

Depois, e imediatamente abaixo da RAMTOP, encontramos o «stack» máquina e o «stack» de GOSUB. Ao sair desta zona e passando a RAMTOP, abandonando assim a área da memória RAM que pode ser limpa usando a instrução NEW, encontramos os gráficos definidos pelo utilizador. A variável de sistema UDG aponta normalmente para o primeiro dos bytes que se seguem à RAMTOP, mas é possível alterar isto em caso de necessidade.

Note-se que ao baixarmos a RAMTOP usando a instrução CLEAR (endereço) obtemos uma maior área de memória livre do funcionamento em Basic.

RESUMO

Neste capítulo o leitor aprendeu:

1. A forma como se encontra organizada a memória do Spectrum.
2. A forma de usar a instrução POKE ou um programa Assembler para comunicar com a memória, e para guardar programas nesta.
3. Como invocar rotinas em código-máquina usando RANDOMIZE USR.

12

SOM

Os computadores estão actualmente a ser usados cada vez mais para tocar música. Os métodos, e os resultados são muitos e variados. Desde notas semi-aleatórias até composições fantásticas, como as do Fairlight que pode ser usado para gravar vários sons digitalmente, como copos partindo-se, cães a ladrar, etc., que depois podem ser reproduzidos a diferentes velocidades permitindo a produção de obras musicais.

Uma das vantagens da música é que pode ser representada de diversas maneiras. Todas estas se encontram relacionadas entre si matematicamente, tornando fácil o seu tratamento por computador. A diferença entre as notas em diferentes oitavas é definida pela frequência, isto é, o Dó acima do Dó central tem uma frequência exactamente dupla da deste. Para subir uma oitava, basta portanto multiplicar por dois.

Uma oitava é definida como o intervalo entre uma frequência e o dobro desta, dividido em oito. De facto divide-se em 13 meios-tons, tendo o maior o dobro da frequência do primeiro.

Infelizmente, é aqui que as coisas se começam a tornar um pouco mais complexas dado que o ouvido ocidental está habituado à escala de oito notas quando de facto existem treze. Por exemplo, ouçamos os resultados produzidos pelos dois programas que se seguem:

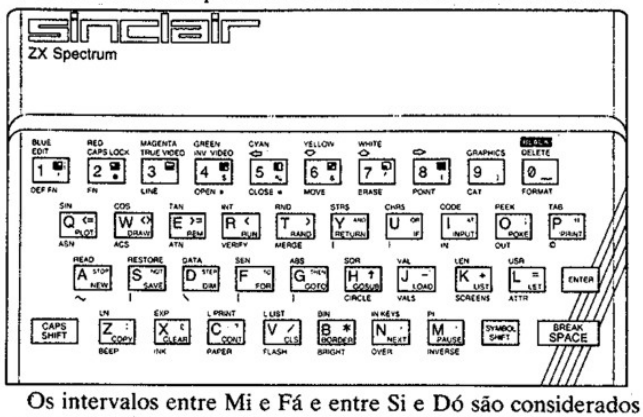
```
10 FOR t=0 TO 12
20 BEEP .5,t
30 NEXT t

10 FOR t=0 TO 7
```

```
20 READ a
30 BEEP .5,a
40 NEXT t
50 DATA 0,2,4,5,7,9,11,12
```

O primeiro toca todos os meios-tons da escala, e o segundo apenas as notas principais, sem notas alteradas.

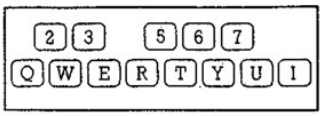
Para resolver este problema, todos os instrumentos de teclado, como o piano, grande parte dos sintetizadores, etc., possuem os respectivos teclados divididos em duas partes, as notas negras e as notas brancas. A primeira contém todas as notas alteradas.



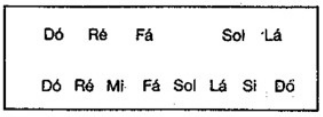
Os intervalos entre Mi e Fá e entre Si e Dó são considerados como meios-tons, enquanto os outros são de tom inteiro. O resultado de tudo isto é que é muito fácil tocar melodias simples em Dó (sem as notas negras). A maior parte dos programas que se seguem usarão apenas esta escala, se bem que não fosse necessário limitarem-se a este tom. O tom pode ser alterado deslocando um pouco o valor das notas. Por exemplo, o programa que se segue, um desenvolvimento do anterior, é ilustrativo:

```
10 FOR s=0 TO 12
20 FOR t=0 TO 7
30 READ a
40 BEEP .1,a+s
50 NEXT t
60 RESTORE
70 NEXT s
80 DATA 0,2,4,5,7,9,11,12
```


O teclado do Spectrum, do tipo máquina de escrever, pode ser alterado de modo a simular um piano desde que se usem as teclas Q W E R T Y U I para representar as notas brancas, e as teclas 2 3 5 6 7 para representar as pretas. A distribuição das notas será portanto a seguinte:



produzindo as notas:



Será muito útil uma subrotina que descodifique as teclas e produza os valores apropriados das notas. Isto é feito lendo o teclado com a instrução INKEY$ e usando o operador de igualdade paara obter o valor correcto. Os valores das notas são mantidos num quadro por uma questão de conveniência, permitindo a sua alteração segundo a vontade do utilizador, sem necessidade de alterar uma linha Basic excessivamente comprida. A rotina é a seguinte:

```
100 LET a$=INKEY$
110 IF a$="" THEN GO TO 100
115 LET n=((a$="Q")*s(1))+((a$=
"2")*s(2))+((a$="W")*s(3))+((a$=
"3")*s(4))+((a$="E")*s(5))+((a$=
"R")*s(6))+((a$="5")*s(7))+((a$=
"T")*s(8))+((a$="6")*s(9))+((a$=
"Y")*s(10))+((a$="7")*s(11))+((a
$="U")*s(12))+((a$="I")*s(13))
120 BEEP .1,N
130 GO TO 100
```

Para construir o quadro, será necessário executar primeiramente a rotina seguinte:

```
1000 DIM s(13)
1010 FOR t=1 TO 13
```

```
1020 READ s(t)
1030 NEXT t
1040 RETURN
1050 DATA -3,-2,-1,0,1,2,3,4,5,6
,7,8,9
```

A primeira linha do programa deverá ser:

```
10 LET a$="": GO SUB 1000
```

Obteremos assim um teclado musical.

Existem vários inconvenientes neste programa, em particular a sua falta de flexibilidade.

Um dos problemas é que as notas se repetem com excessiva rapidez. Uma solução para isto consistirá em alterar as linhas seguintes:

```
  10 IF s$=a$ THEN GO TO 100
 100 LET s$=a$: LET a$=INKEY$
1005 LET a$=" "
```

o que nos permitem evitarmos as repetições.

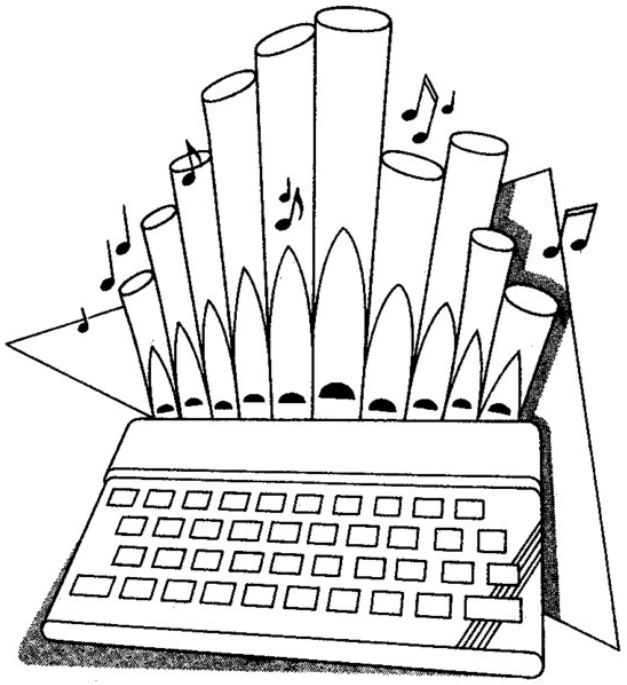
Um dos outros problemas principais da instrução de som do Spectrum é que, depois de iniciada, não pode ser interrompida. Convém portanto garantir que o comprimento da nota na linha 120 não é excessivo.

A partir disto, é possível acrescentar diversas funções que tornarão o programa muito mais útil e versátil.

A primeira destas é um efeito de vibrato que tornará o som um pouco mais interessante. Seria igualmente bom que fosse possível «ligar» e «desligar» este efeito. Para obter o vibrato, é necessário alterar do seguinte modo a frequência de beep:

```
10 BEEP .01,0
20 BEEP .01,7
30 GO TO 10
```

Usando esta ideia, podem acrescentar-se as linhas seguintes a fim de obter o vibrato. Consegue-se ainda, através desta subrotina, evitar o problema da interrupção da nota ao tocar noutra tecla. Dado que o BEEP é de facto bastante curto, é possível detectar a pressão noutra tecla e substituir a nota actual por outra.



Um outro aspecto é a possibilidade de calibrar o teclado para qualquer tom ou oitava desejada, e dado que vão de facto existir várias funções, será chegado o momento de pensar no uso de um menú. As linhas seguintes permitir-nos-ão iniciar o trabalho:

```
  10 LET a$="": GO SUB 1000
  20 GO TO 3000
  99 REM tocar
 106 IF s$=a$ THEN GO TO 100
 107 IF a$="K" THEN RETURN
 120 GO SUB 2000
 160 RETURN
 999 REM definir variaveis e qua
dros
```

```
1005 LET a$=""
1006 LET d=0: LET l=10: LET o=0
1999 REM nova rotina de beep
2000 FOR t=0 TO l
2004 IF INKEY$="" THEN LET t=l:
GO TO 2020
2005 BEEP .01,n+o
2010 BEEP .01,n+o+d
2020 NEXT t
2030 RETURN
2999 REM menu principal
3000 CLS : PRINT "MENU PRINCIPAL
"
3010 PRINT
3020 PRINT "Z... Mais agudo"
3030 PRINT "X... Mais grave"
3040 PRINT "M... Oitava acima"
3050 PRINT "N... Oitava abaixo"
3060 PRINT "B... Alongar nota"
3070 PRINT "V... Encurtar nota"
3090 PRINT "S... Aumentar vibrat
o"
3100 PRINT "D... Diminuir vibrat
o"
3110 PRINT "L... Tocar"
3120 PRINT "K... Menu"
3200 LET a$=INKEY$
3210 IF a$="Z" THEN LET o=o+.5:
GO SUB 3500: GO TO 3000
3220 IF a$="X" THEN LET o=o-.5:
GO SUB 3500: GO TO 3000
3230 IF a$="M" THEN LET o=o+12:
GO SUB 3500: GO TO 3000
3240 IF a$="N" THEN LET o=o-12:
GO SUB 3500: GO TO 3000
3250 IF a$="B" THEN LET l=l+1: G
O SUB 3600: GO TO 3000
3260 IF a$="V" THEN LET l=l-1: G
O SUB 3600: GO TO 3000
3270 IF a$="S" THEN LET d=d+.1:
GO SUB 3600: GO TO 3000
3280 IF a$="D" THEN LET d=d-.1:
GO SUB 3600: GO TO 3000
3290 IF a$="L" THEN CLS : PRINT
"TOCAR": GO SUB 100: GO TO 3000
3300 GO TO 3200
3499 REM tocar escala
3500 FOR n=0 TO 12
3510 GO SUB 2000
3520 NEXT n
```



```
3530 RETURN
3599 REM tocar uma nota
3600 LET n=0
3610 GO SUB 2000
3620 RETURN
```

Este programa permite agora tocar melodias no teclado do Spectrum (utilize letras maiúsculas quando escolhe as opções). Em seguida teremos de considerar a necessidade de gravar e reproduzir essas melodias. É ainda possível mostrar as notas no visor, no local adequado da pauta. O melhor modo de o fazer consiste em recorrer aos gráficos definidos pelo utilizador:

```
4000 RESTORE 4000
4005 FOR t=0 TO 159
4010 READ a: POKE USR "s"+t,a
4020 NEXT t
4100 DATA 0,255,0,0,255,8,8,255,
8,8,255,8,8,255,24,24
4110 DATA 0,255,0,8,255,8,8,255,
8,8,255,8,24,255,24,0
4120 DATA 0,255,8,8,255,8,8,255,
8,8,255,24,24,255,0,0
4130 DATA 0,255,8,8,255,8,8,255,
8,24,255,24,0,255,0,0
4140 DATA 8,255,8,8,255,8,8,255,
24,24,255,0,0,255,0,0
4145 DATA 0,255,0,0,255,0,24,255
,24,16,255,16,16,255,16,16
4150 DATA 0,255,0,0,255,24,24,25
5,16,16,255,16,16,255,16,16
4160 DATA 0,255,0,24,255,24,16,2
55,16,16,255,16,16,255,16,0
4170 DATA 0,255,24,24,255,16,16,
255,16,16,255,16,16,255,0,0
4180 DATA 24,255,24,16,255,16,16
,255,16,16,255,16,0,255,0,0
4190 RETURN
```

Depois de serem introduzidos estes dados, podem ser usados pela subrotina que se segue.

Note-se que os caracteres das linhas 4205 e 4210 são caracteres gráficos (UDG's).

```
4200 LET g=g+1: IF g>30 THEN LET
 m=m+3: LET g=0
4201 IF m>21 THEN LET m=2: LET g
=0
4205 PRINT AT m,g;("A" AND n=-3)
```

```
;("C" AND n=-1);("E" AND n=1);("
G" AND n=2);("I" AND n=4);("K" A
ND n=6);("M" AND n=8);("O" AND n
=10);
4210 PRINT AT m+1,9;("B" AND n=-
3);("D" AND n=-1);("F" AND n=1);
("H" AND n=2);("J" AND n=4);("L"
 AND n=6);("N" AND n=8);("P" AND
 n=10);
4220 RETURN
```

Devemos ainda acrescentar as seguintes linhas:

```
   11 GO SUB 4000
  125 GO SUB 4200
 1007 LET m=2: LET g=0
```

e alterar do seguinte modo a linha 3290:

```
3290 IF a$="L" THEN LET m=2: LET
 g=0: CLS : GO SUB 100: GO TO 30
00
```


Agora, sempre que se tocar uma nota será apresentada no visor uma sua representação. Para gravar as notas, é necessário inclui-las num quadro, pelo que vamos dimensionar Q() e incluir uma opção de gravação e carga de músicas a partir de cassete ou microdrive. Acrescentemos o seguinte ao menú principal:

```
3130 PRINT "G... MENU DE MONTAGE
M"
3295 IF a$="g" THEN GO SUB 6000:
 REM menu de montagem
6000 CLS
6010 PRINT "MENU DE MONTAGEM"
6020 PRINT
6030 PRINT "A... Nova musica"
6040 PRINT "B... Tocar musica"
6050 PRINT "C... Montar musica"
6060 PRINT "D... Gravar musica"
6070 PRINT "E... Carregar musica
"
6080 PRINT "0... Menu principal"
6200 LET a$=INKEY$
6210 IF a$="A" THEN GO SUB 6300:
 GO TO 6000
6220 IF a$="B" THEN GO SUB 6400:
 GO TO 6000
6230 IF a$="C" THEN GO SUB 6700:
 GO TO 6000
6240 IF a$="D" THEN GO SUB 6500:
 GO TO 6000
6250 IF a$="E" THEN GO SUB 6600:
 GO TO 6000
6255 IF a$="0" THEN RETURN
6260 GO TO 6200
6299 REM nova musica
6300 LET r=1: REM definir flag
6310 LET c=1: LET m=2: LET g=0:
CLS : GO SUB 100
6320 LET nts=c: LET r=0
6330 RETURN
```

Acrescentemos, na linha 126:

```
126 IF r=1 THEN LET q(c)=n: LET
c=c+1
```

e as linhas:

```
1001 DIM q(64)
1008 LET r=0: LET nts=1
6399 REM tocar musica
```

```
6400 CLS : PRINT "CARREGUE EM B
PARA OUVIR MUSICA"
6410 LET m=2: LET g=0
6420 FOR c=1 TO nts
6430 LET n=q(c): GO SUB 2000: GO
 SUB 4200
6440 NEXT c
6445 GO SUB 1500
6450 RETURN
```

Acrescentemos ainda:

```
1500 PRINT AT 21,0;"Carregue em
SPACE para continuar"
1510 IF INKEY$<>" " THEN GO TO 1
510
1520 RETURN
6499 REM gravar musica
6500 INPUT "Indique nome ";f$
6510 SAVE *"M";1;f$ DATA q()
6520 RETURN
6599 REM carregar musica
```



```
6600 INPUT "Indique nome ";f$
6610 LOAD *"M";1;f$ DATA q()
6620 RETURN
```

Convém incluir uma opção de montagem, permitindo a inclusão de alterações nas músicas. Basta para tal alterar a linha 115, transformando-a numa subrotina de alteração. Deve-se portanto invocar a linha 115 e modificá-la. Passemo-la para 150, transformando a 115 em GOSUB 150. Introduzimos depois a linha 160 RETURN.

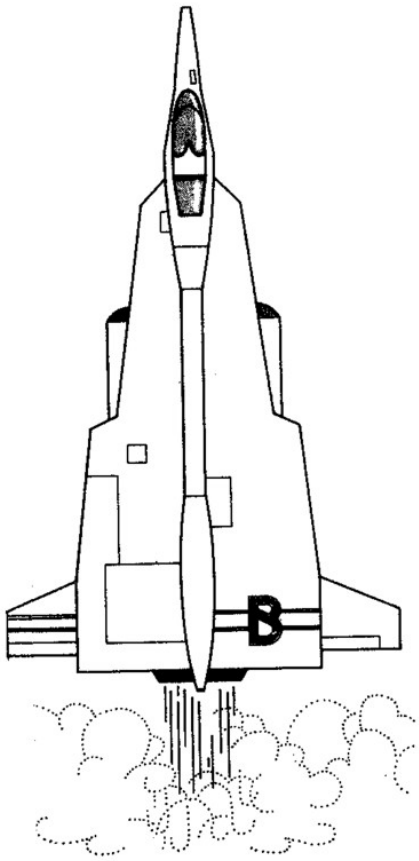
O bloco seguinte permite usar separadamente a rotina de leitura do teclado.

```
6699 REM alterar musica
6700 CLS : PRINT "Passe a musica
 carregando em     SPACE. Use X p
ara parar."
6710 PRINT "Para mudar uma nota,
 carregue na tecla que lhe corres
ponde."
6715 LET m=8: LET g=0
6720 FOR c=1 TO 4
6730 LET n=q(c): GO SUB 4200: BE
EP .1,n
6740 LET a$=INKEY$: IF a$="" THE
N GO TO 6740
6745 IF a$="X" THEN LET c=64
6750 IF a$<>" " THEN GO SUB 150:
 BEEP .1,n: LET q(c)=n: LET g=g-
1: GO SUB 4200
6755 IF c>nts THEN LET nts=nts+1
6770 NEXT c
6780 RETURN
```

Como é óbvio, é possível introduzir melhoramentos neste programa, mas dispomos já de uma base satisfatória de um programa de composição. O modo como foi construído ilustra alguns dos métodos de modificação e construção de programas, particularmente a necessidade de documentar os programas usando **REM's**.

A principal utilidade do gerador de som do Spectrum consiste em criar ruídos de fundo para jogos. A maior parte destes são fáceis de obter e encontram-se documentados em muitos livros e manuais. Incluiremos no entanto aqui alguns desses efeitos, a fim de dar ao leitor uma ideia do que é possível obter. Por exemplo, ouçamos uma nave levantando voo:

```
10 FOR t=0 TO 69
20 FOR s=0 TO 5
30 BEEP .005,t
40 NEXT s
50 NEXT t
```



Para a fazer aterrar, basta alterar a primeira linha do seguinte modo:

```
10 FOR t=69 TO 0 STEP -1
```

Para obter uma ambulância, usemos:

```
10 BEEP .5,10
20 BEEP .5,5
30 GO TO 10
```

No caso de ruídos mais eficazes, convirá aceder directamente ao altifalante do Spectrum. É possível fazê-lo facilmente usando a instrução OUT 254. Infelizmente, esta instrução afecta igualmente a margem do visor, pelo que devemos indicar a cor correcta, além do som. Por exemplo:

```
10 OUT 254,16
20 OUT 254,0
30 GO TO 10
```

produz um agradável som, mas a margem muda de cor. Os primeiros três bits de cada byte controlam as cores vermelha, azul e verde. Para manter a cor de margem actual é necessário garantir que se usa o valor adequado. É possível descobri-lo experimentando um pouco.

Experimente o seguinte:

```
10 FOR t=0 TO 7
20 OUT 254,t
30 PRINT t
40 PAUSE 0
50 NEXT t
```

Pode constatar assim quais as cores que é possível obter. Basta acrescentar ou diminuir 16 à cor escolhida para ligar e desligar o som. Escolhamos como, exemplo, a cor branca para a margem, e vejamos o resultado:

```
10 OUT 254,23
20 OUT 254,7
30 GO TO 10
```

onde 23 é 17+7. Pode alterar-se a linha 10 para OUT 254,16+7 o que se traduz imediatamente numa mudança da frequência

do som, devido à necessidade do Spectrum de avaliar o valor de 16+7 sempre que executa o ciclo.

Para obtermos um contador Geiger afastando-se de uma fonte radioactiva, poderemos usar:

```
10 FOR t=0 TO 100
20 OUT 254,23
30 OUT 254,7
40 FOR s=0 TO t: NEXT s
50 NEXT t
```

e para obter impulsos aleatórios:

```
10 OUT 254,23: OUT 254,7
20 FOR t=0 TO INT (RND*10)-1:
NEXT t
30 GO TO 10
```

A quantidade pela qual se multiplica RND determina a frequência média destes impulsos.

Existem muitos outros truques que podem ser obtidos acedendo directamente ao altifalante, mas para conseguir efeitos verdadeiramente bons é necessário recorrer ao código-máquina. Os sons produzidos por um helicóptero, etc., são fáceis de obter com método ligeiramente para além dos apresentados neste livro.

PROJECTOS

1. Melhore o programa de composição musical de modo a obter um programa de montagem que permita alterar as notas e imprimi-las (experimente a instrução COPY). Este programa de composição poderia ainda ser modificado de modo a produzir notas alteradas juntamente com os gráficos respectivos, se bem que talvez seja necessário alterar o conjunto de caracteres a fim de obter os caracteres necessários.
2. Experimente a instrução BEEP de modo a obter diferentes efeitos sonoros, e modifique depois os seus programas de modo a conseguir o mesmo resultado usando OUT 254.

13

INTERFACE 1

O Spectrum que todos podemos comprar nas lojas é extremamente útil, mas depois de o leitor programar durante algum tempo descobrirá alguns inconvenientes, incluindo a dificuldade de obter uma cópia em impressora e o tempo necessário para gravar ou carregar programas de fita. A popularidade do Spectrum conduziu porém ao desenvolvimento de várias soluções para estes problemas, entre as quais o Interface 1 da Sinclair.

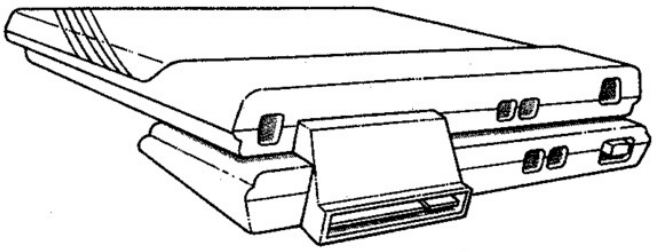
Um extra essencial

Para compreender o que é o Interface 1 é necessário estudar um pouco a concepção básica do Spectrum. Tal como é vendida, a máquina constitui uma unidade compacta, que comunica com o mundo exterior apenas através da sua saída TV e das tomadas de ligação ao gravador. Na parte traseira do computador existe ainda um ligador, que basicamente equivale a uma extensão do circuito interno da máquina. Se observarmos este ligador veremos um conjunto de linhas que formam canais de potencial comunicação.

Nos computadores mais caros estas linhas são transformadas num certo número de «tomadas» que permitem ligar essas máquinas a impressoras, monitores, etc. Mas é certamente mais barato deixar as linhas à vista, como acontece no Spectrum, e deixar para mais tarde os problemas das «fichas» que será necessário usar.

O que o Interface 1 faz é precisamente utilizar estas linhas

para três fins específicos – uma interface de impressora, controlo dos Microdrives Sinclair, e a possibilidade de formação de uma rede. Vale a pena considerarmos aqui estas três áreas de um modo um pouco mais profundo.



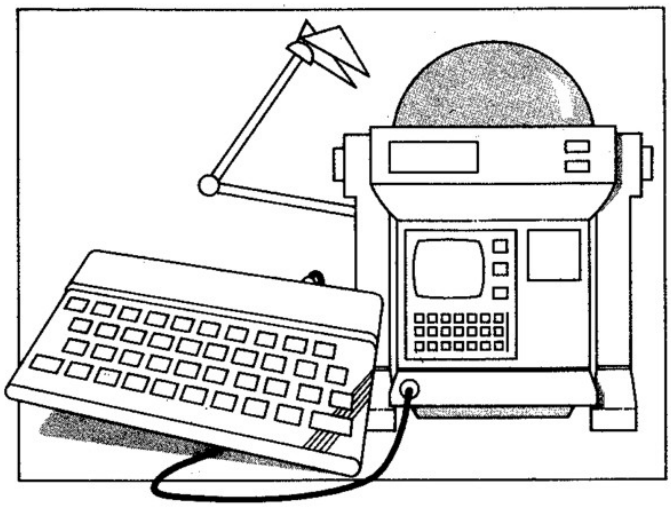
A interface da impressora

Existem essencialmente duas interfaces para impressoras em uso nos computadores: a Centronics, ou paralela, e a RS232, ou serial. A Interface 1 utiliza esta última. Sem entrarmos excessivamente em pormenores técnicos, a forma mais simples de exemplicar a diferença entre ambas é que a interface paralela envia dados simultaneamente ao longo de várias linhas, enquanto a interface serial envia os dados, um de cada vez, por uma mesma linha. Pode alterar-se neste último caso a velocidade e a forma da transferência de dados, mas não no caso de uma interface paralela.

Uma consequência disto é que podemos usar diversos métodos para transportar dados seriais – podemos enviá-los ao longo de linhas telefónicas, por exemplo, bastando para tal converter o sinal em som e vice-versa. O RS232 dá-nos portanto a oportunidade de comunicarmos com outros micros, ou com dispositivos como os *robots* ou alarmes contra ladrões. Este interface serve portanto para comunicações e controlo, bastando-

-nos escrever os programas necessários para imediatamente podermos entrar neste novo mundo...

Se desejamos simplesmente uma interface de impressora, a fim de obtermos saídas em papel, para tratamento de texto ou para listar os nossos programas, a interface Centronics é provavelmente a mais apropriada. Não tem a flexibilidade da RS232, mas como não é necessário definir velocidades de transferência de dados tende a causar menos problemas. Se bem que a Interface 1 não permita este tipo de funcionamento, é possível obter várias interfaces Centronics para o Spectrum, e no final do capítulo encontrará uma pequena lista delas.



A rede

A Interface 1 permite ainda algo a que se chama «rede local» – basicamente um grupo de computadores trabalhando em conjunto, ligados por cabos. O sistema Sinclair permite a ligação de dois a sessenta e quatro Spectrums por este método,

e apesar de o leitor conseguir facilmente descobrir algumas aplicações destas redes para o seu caso pessoal, a sua utilidade mais flagrante é nas escolas.

A utilização destas redes é bastante simples, empregando versões de outras instruções Spectrum/Interface 1 para efeitos de controlo. Para enviar informação, por exemplo, usa-se uma variante da instrução SAVE, enquanto para receber se usa a LOAD – a única diferença é que se está a SAVE e a LOAD de ou para outro Spectrum, e não para uma cassete.

Se tiver alguns amigos que possuam a Interface 1, talvez valha a pena desenvolver um qualquer jogo interactivo que lhe permita utilizar esta possibilidade de formação de redes – já são vendidos alguns comercialmente, mas é ainda possível desenvolver muitas ideias neste campo.

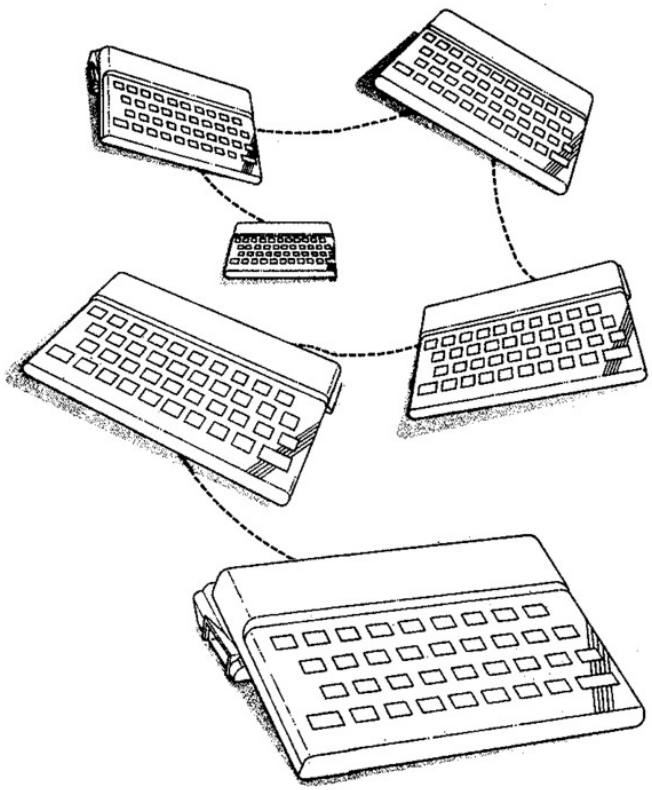
Os microdrives

O objectivo do Microdrive Sinclair consiste em constituir uma alternativa barata e eficaz à memória periférica em disco, e em termos gerais consegue fazê-lo. A unidade básica é uma caixa negra com as dimensões de um punho, capaz de conter uma «cartridge» dentro da qual se encontra uma fita contínua. No caso do armazenamento em disco pode normalmente carregar-se programas usando «acesso aleatório» – isto é, pode aceder-se directamente a informação que se encontra em qualquer ponto do disco, tal como no caso de um gira-discos é possível aceder imediatamente qualquer composição independentemente da ordem em que se encontra.

No caso das unidades de banda ou fita magnética, é necessário desenrolar toda a fita até atingir o programa que se deseja. O microdrive é uma unidade de fita, mas possui a vantagem de correr tão rapidamente que necessita apenas de oito segundos para percorrer toda a fita – pelo que os seus tempos de acesso são de facto comparáveis aos de algumas unidades de disco.

Uma experiência

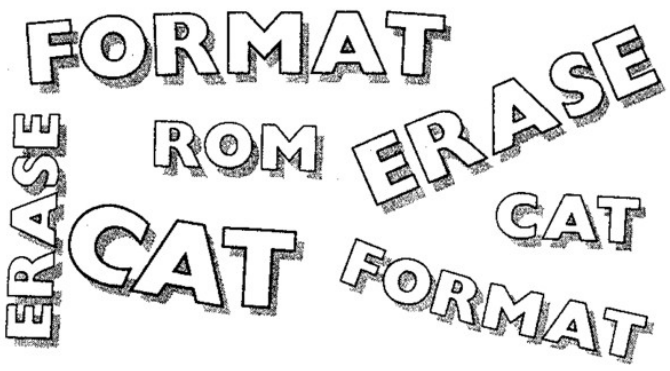
Devido a estes bons tempos de acesso, é possível fazer



coisas com os microdrives que são impossíveis quando apenas se usam cassetes. Suponhamos que queremos demonstrar os nossos talentos artísticos aos nossos amigos, e que, dispondo de um óptimo programa de desenho, possuímos tantas imagens que não conseguimos guardá-las todas em memória simultaneamente.

```
  10 CLS
  20 PRINT
  30 PRINT "SESSAO DE CINEMA"
  40 PRINT
  50 INPUT "Indique numero de im
agens ";a$
  60 IF VAL a$<1 OR VAL a$>20 TH
EN BEEP 1,1: GO TO 50
  70 LET a=VAL a$: CLS
  80 PRINT
  90 PRINT "Escolha modo"
 100 PRINT "1... Imagens continu
as"
 105 PRINT "2... Pausa entre ima
gens"
 110 PRINT "3... Escolha de uma
imagem"
 120 IF INKEY$="1" THEN FOR n=1
TO a: LOAD *"M";1;STR$(n) SCREEN
$: NEXT n
 130 IF INKEY$="2" THEN FOR n=1
TO a: LOAD *"M";1;STR$(n) SCREEN
$: PAUSE 0: NEXT n
 140 IF INKEY$="3" THEN PRINT: I
NPUT "Numero da imagem? ";n: LOA
D *"M";1;STR$(n) SCREEN$
 150 GO TO 90
```

Esta rotina constitui a base de um interessante programa de apresentação de *slides*. É fácil ver que é bastante mais rápido do que a fita, mas ainda um pouco lento. Como no entanto sabemos que o Spectrum dispõe de espaço em memória para guardar vários «screens» simultaneamente, pode usar-se este programa como base de construção de uma espécie de sequência animada. Tal como se encontra, o programa pode ser usado para invocar qualquer imagem gravada que esteja identificada por um número. Se quisermos construir um programa mais desenvolvido, poderemos facilmente desenvolvê-lo a partir daqui.



As instruções

Como se pode constatar observando a tabela de instruções básicas para o microdrive que apresentamos em seguida, e ignorando a FORMAT, a CAT e a ERASE, são de facto semelhantes às instruções usuais no Spectrum para comando do tratamento de ficheiros em cassete, se bem que a sintaxe seja ligeiramente mais complicada. É necessário especificar um nome na carga, gravação, etc., e convém ainda especificar o número do microdrive (N, na tabela)

FORMAT "M";N;"nome"
CAT N
ERASE "M";N;"nome"
SAVE* "M";N;"nome"
LOAD* "M";N;"nome"
VERIFY* "M";N;"nome"
MERGE* "M";N;"nome".

O interface possui uma ROM própria, que é «ligada» em vez da ROM normal do Spectrum sob a forma de uma espécie de intercepção de erro. Normalmente obtém-se uma mensagem

de erro quando se tenta usar uma instrução de microdrive, mas quando o interface 1 se encontra ligado o Spectrum verifica primeiro o conteúdo da ROM do Interface 1, e se a instrução está correcta executa-a. Note que se dispõe aqui de um possível bónus – se se consegue substituir as rotinas em ROM que realizam este tratamento, é possível acrescentar instruções ao Spectrum.

Interface 2

O segundo periférico de interesse para o Spectrum é o Interface 2. É mais simples do que o Interface 1, na medida em que apenas possibilita dois usos extra, de «cartridges» ROM e *joysticks*. As «cartridges» são úteis porque constituem o modo mais rápido possível de carregar um jogo, devido ao facto de paginarem na verdade uma nova área de memória; mas não é possível gravar programas nossos nelas.

Estas «cartridges» abrem no entanto várias possibilidades, como seja a substituição de sistemas operativos, e sem dúvida que mais tarde ou mais cedo alguém começará a fazer algo deste género.

Os *joysticks*, no entanto, são algo que se pode incorporar nos nossos programas. A tabela que se segue mostra como os dois *joysticks* emulam as teclas habituais:

Tecla	Movimento do joystick	Joystick
1	Esquerda	2
2	Direita	2
3	Para baixo	2
4	Para cima	2
5	Tiro	2
6	Esquerda	1
7	Direita	1
8	Para baixo	1
9	Para cima	1
0	Tiro	1

Assim, se se ligarem dois *joysticks* ao Interface 2 obtém-se

o efeito correspondente ao uso destas teclas, desde que evidentemente o programa o permita.

Normalmente usa-se INKEY$ para este controlo, mas esta instrução tem a desvantagem de ser incapaz de ler duas teclas simultaneamente, não sendo portanto possível mover e disparar ao mesmo tempo. É no entanto possível usar uma outra instrução.

A função IN e a declaração OUT são usadas para controlar os «ports» de entrada/saída do Spectrum.

Escrevendo:

```
IN endereço
```

obtemos o byte do «port» cujo endereço indicámos. Enviamos um valor para esse «port» escrevendo:

```
OUT endereço, valor
```

Por agora interessa-nos a IN. Os endereços que governam os dois «ports» de *joystick* são 61438 no caso do *joystick* 1 e 63486 no caso de *joystick* 2. Escrevendo IN 61438, obtemos portanto um número de oito bits que nos dá informações sobre o estado do *joystick* 1. O «port» trata cinco movimentos, pelo que só nos interessa os primeiros cinco dos oito bits:

Movimento	Joystick 1	Joystick 2
	IN 61438	IN 63486
Tiro	Bit 0	Bit 4
Subir	Bit 1	Bit 3
Descer	Bit 2	Bit 2
Direita	Bit 3	Bit 1
Esquerda	Bit 4	Bit 0

A situação é aqui análoga à das variáveis de sistema KSTATE. Movendo o *joystick* 1 altera-se o número guardado em 61438, e como se pode verificar na tabela acima cada um dos primeiros cinco bits desse número determina uma acção bem definida. Assim, a fim de usar IN para mover algo no visor

teremos de examinar estes bits individualmente. Primeiro necessitamos de uma variável para cada movimento:

```
  10 LET f=0: LET u=0: LET d=0:
LET r=0: LET l=0
```

Dispomos assim de cinco variáveis, uma para cada movimento, e tornamo-las iguais a zero. O passo seguinte consiste em ler o valor em 61438, e em eliminar os bits 5, 6 e 7, dado que não necessitamos deles:

```
20 LET n=255-IN 61438
30 IF n>127 THEN LET n=n-128
40 IF n>63 THEN LET n=n-64
50 IF n>31 THEN LET n=n-32
```

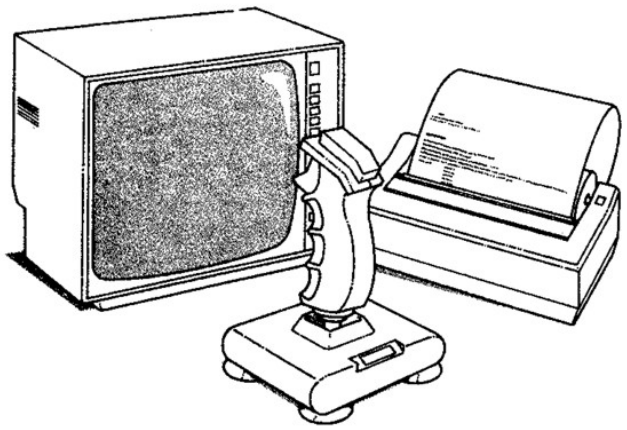
Temos agora tudo aquilo de que necessitamos. Vamos agora alterar o estado das nossas cinco variáveis em função do que está a ser feito com o *joystick*:

```
  60 IF n>15 THEN LET n=n-16: LE
T l=1
  70 IF n>7 THEN LET n=n-8: LET
r=1
  80 IF n>3 THEN LET n=n-4: LET
d=1
  90 IF n>1 THEN LET n=n-2: LET
u=1
 100 IF n=1 THEN LET f=1
```

Basta-nos agora escrever uma rotina para o *joystick* 2, definindo o movimento em função dos valores das variáveis.

Outros periféricos

Este livro não constitui certamente o melhor local para tratar os periféricos do Spectrum, mas existem alguns pormenores que talvez interessem ao leitor. Se quiser adquirir um interface Centronics, em vez ou para além do Interface 1, note que existem interfaces comutáveis Centronics/RS232, fabricados pela Euroelectronics e pela Morex. Ambos são compatíveis com o Interface 1, e isto é importante dado que os periféricos construídos por algumas empresas não funcionam com o Interface 1 ligado.



Dois outros dispositivos que talvez interessem ao leitor são um monitor e um teclado de maior qualidade para o Spectrum. Na maior parte dos casos um novo teclado obrigará a invalidar a garantia do seu Spectrum, dado que obrigará a abrir a caixa deste, mas noutros casos tal não acontecerá. Provavelmente os melhores são os da Transform, que exigem a abertura da caixa; e entre os que não têm esta exigência talvez o Stonechip mereça alguma atenção.

Só se pode de facto ligar um monitor se se souber fazer soldaduras, ou se conhece alguém que o saiba. Observe o desenho do ligador traseiro no final do capítulo 23 do Manual do Spectrum, e procure as linhas Video e 0V. Ligue a linha Video ao núcleo central do cabo de televisão e a linha 0V ao exterior deste, e disporá de uma saída vídeo composta. No caso da versão 3 do Spectrum (posterior ao Natal de 1983) bastar-lhe-á proceder deste modo, mas nos modelos anteriores é necessário realizar ainda outra ligação. De qualquer modo, não se arrisque a fazer estas ligações a menos que saiba exactamente o que está a fazer.

Existem muitos outros extras que pode ligar ao seu Spectrum, e se quiser expandir o seu sistema o melhor que pode fazer

é ler o que é escrito e publicitado nas revistas especializadas. Uma última palavra de aviso – verifique sempre se a empresa que fabrica o material que lhe interessa ainda existe, e se tem esse produto em *stock*, antes de enviar o seu dinheiro pelo correio e ter grandes desilusões.

RESUMO

Neste capítulo, o leitor aprendeu:

1. A diferença entre um interface RS232 e um interface Centronics.
2. O que é uma rede local.
3. A diferença entre acesso aleatório e acesso serial.
4. Como adaptar os seus programas ao uso do Interface 2.

APÊNDICE

Códigos ASCII do conjunto de caracteres do Spectrum

Código	Caracter	Hex	Assembler Z80	Após CB	Após ED
0	Não usados	00	NOP	RLC B	
1		01	LD BC,nn	RLC C	
2		02	LD (BC),A	RLC D	
3		03	INC BC	RLC E	
4		04	INC B	RLC H	
5		05	DEC B	RLC L	
6	Imprimir vírgula	06	LD B,n	RLC (HL)	
7	EDIT	07	RLCA	RLC A	
8	Cursor para a esquerda	08	EX AF,AF'	RRC B	
9	Cursor para a direita	09	ADD HL,BC	RRC C	
10	Cursor para baixo	0A	LD A,(BC)	RRC D	
11	Cursor para cima	0B	DEC BC	RRC E	
12	DELETE	0C	INC C	RRC H	
13	ENTER	0D	DEC C	RRC L	
14	Número	0E	LD C,n	RRC (HL)	
15	Não usado	0F	RRCA	RRC A	
16	Controlo de INK	10	DJNZ n	RL B	
17	Controlo de PAPER	11	LD DE,nn	RL C	
18	Controlo de FLASH	12	LD (DE),A	RL D	
19	Controlo de BRIGHT	13	INC DE	RL E	
20	Controlo de INVERSE	14	INC D	RLH	
21	Controlo de OVER	15	DEC D	RL L	
22	Controlo AT	16	LD D,n	RL(HL)	
23	Controlo TAB	17	RLA	RL A	
24	Não usados	18	JR n	RR B	
25		19	ADD HL,DE	RR C	
26		1A	LD A,(DE)	RR D	
27		1B	DEC DE	RR E	
28		1C	INC E	RR H	
29		1D	DEC E	RR L	
30		1E	LD E,n	RR(HL)	
31		1F	RRA	RR A	

32	Espaço	20	JR NZ,n	SLA B
33	!	21	LD HL,nn	SLA C
34	"	22	LD (nn),HL	SLA D
35	#	23	INC HL	SLA E
36	$	24	INC H	SLA H
37	%	25	DEC H	SLA L
38	&	26	LD H,n	SLA (HL)
39	'	27	DAA	SLA A
40	(	28	JR Z,n	SRA B
41	)	29	ADD HL,HL	SRA C
42	*	2A	LD HL,(nn)	SRA D
43	+	2B	DEC HL	SRA E
44	,	2C	INC L	SRA
45	-	2D	DEC L	SRA L
46	.	2E	LD L,n	SRA (HL)
47	/	2F	CPL	SRA A
48	0	30	JR NC,n	
49	1	31	LD SP,nn	
50	2	32	LD (nn),A	
51	3	33	INC SP	
52	4	34	INC (HL)	
53	5	35	DEC (HL)	
54	6	36	LD (HL),n	
55	7	37	SCF	
56	8	38	JR C,n	SRL B
57	9	39	ADD HL,SP	SRL C
58	:	3A	LD A,(nn)	SRL D
59	;	3B	DEC SP	SRL E
60	<	3C	INC A	SRL H
61	=	3D	DEC A	SRL L
62	>	3E	LD A,n	SRL (HL)
63	?	3F	CCF	SRL A
64	@	40	LD B,D	BIT 0,B	IN B,(C)
65	A	41	LD B,C	BIT 0,C	OUT (C),B
66	B	42	LD B,D	BIT 0,D	SBC HL,BC
67	C	43	LD B,E	BIT 0,E	LD (nn),BC
68	D	44	LD B,H	BIT 0,H	NEG
69	E	45	LD B,L	BIT 0,L	RETN
70	F	46	LD B,(HL)	BIT 0,(HL)	IM 0
71	G	47	LD B,A	BIT 0,A	LD I,A
72	H	48	LD C,B	BIT 1,B	IN C,(C)
73	I	49	LD C,C	BIT 1,C	OUT (C),C
74	J	4A	LD C,D	BIT 1,D	ADC HL,BC
75	K	4B	LD C,E	BIT 1,E	LD BC,(nn)
76	L	4C	LD C,H	BIT 1,H	
77	M	4D	LD C,L	BIT 1,L	RETI
78	N	4E	LD C,(HL)	BIT 1,(HL)	
79	O	4F	LD C,A	BIT 1,A	LD R,A
80	P	50	LD D,B	BIT 2,B	IN D,(C)
81	Q	51	LD D,C	BIT 2,C	OUT (C),D
82	R	52	LD D,D	BIT 2,D	SBC HL,DE

83	S	53	LD D,E	BIT 2,E	LD (nn),DE
84	T	54	LD D,H	BIT 2,H	
85	U	55	LD D,L	BIT 2,L	
86	V	56	LD D,(HL)	BIT 2,(HL)	IM 1
87	W	57	LD D,A	BIT 2,A	LD A,I
88	X	58	LD E,B	BIT 3,B	IN E,(C)
89	Y	59	LD E,C	BIT 3,C	OUT (C),E
90	Z	5A	LD E,D	BIT 3,D	ADC HL,DE
91	[	5B	LD E,E	BIT 3,E	LD DE,(nn)
92	/	5C	LD E,H	BIT 3,H	
93	]	5D	LD E,L	BIT 3,L	
94	↑	5E	LD E,(HL)	BIT 3,(HL)	IM 2
95	—	5F	LD E,A	BIT 3,A	LD A,R
96	£	60	LD H,B	BIT 4,B	IN H,(C)
97	a	61	LD H,C	BIT 4,C	OUT (C),H
98	b	62	LD H,D	BIT 4,D	SBC HL,HL
99	c	63	LD H,E	BIT 4,E	LD (nn),HL
100	d	64	LD H,H	BIT 4,H	
101	e	65	LD H,L	BIT 4,L	
102	f	66	LD H,(HL)	BIT 4,(HL)	
103	g	67	LD H,A	BIT 4,A	RRD
104	h	68	LD L,B	BIT 5,B	IN L,(C)
105	i	69	LD L,C	BIT 5,C	OUT (C),L
106	j	6A	LD L,D	BIT 5,D	ADC HL,HL
107	k	6B	LD L,E	BIT 5,E	LD HL, (nn)
108	l	6C	LD L,H	BIT 5,H	
109	m	6D	LD L,L	BIT 5,L	
110	n	6E	LD L,(HL)	BIT 5,(HL)	
111	o	6F	LD L,A	BIT 5,A	RLD
112	p	70	LD (HL),B	BIT 6,B	IN F,(C)
113	q	71	LD (HL),C	BIT 6,C	
114	r	72	LD (HL),D	BIT 6,D	SBC HL,SP
115	s	73	LD (HL),E	BIT 6,E	LD (nn),SP
116	t	74	LD (HL),H	BIT 6,H	
117	u	75	LD (HL),L	BIT 6,L	
118	v	76	HALT	BIT 6,(HL)	
119	w	77	LD (HL),A	BIT 6,A	
120	x	78	LD A,B	BIT 7,B	IN A,(C)
121	y	79	LD A,C	BIT 7,C	OUT (C),A
122	z	7A	LD A,D	BIT 7,D	ADC HL,SP
123	{	7B	LD A,E	BIT 7,E	LD SP,(nn)
124	|	7C	LD A,H	BIT 7,H	
125	}	7D	LD A,L	BIT 7,L	
126	~	7E	LD A,(HL)	BIT 7,(HL)	
127	©	7F	LD A,A	BIT 7,A	
128	□	80	ADD A,B	RES 0,B	
129	▗	81	ADD A,C	RES 0,C	
130	▖	82	ADD A,D	RES 0,D	
131	▄	83	ADD A,E	RES 0,E	
132	▝	84	ADD A,H	RES 0,H	
133	▐	85	ADD A,L	RES 0,L	
134	▚	86	ADD A,(HL)	RES 0,(HL)	

135	▘	87	ADD A,A	RES 0,A	
136	▝	88	ADC A,B	RES 1,B	
137	▀	89	ADC A,C	RES 1,C	
138	▖	8A	ADC A,D	RES 1,D	
139	▌	8B	ADC A,E	RES 1,E	
140	▞	8C	ADC A,H	RES 1,H	
141	▛	8D	ADC A,L	RES 1,L	
142	▗	8E	ADC A,(HL)	RES 1,(HL)	
143	▐	8F	ADC A,A	RES 1,A	
144	(a)	90	SUB B	RES 2,B	
145	(b)	91	SUB C	RES 2,C	
146	(c)	92	SUB D	RES 2,D	
147	(d)	93	SUB E	RES 2,E	
148	(e)	94	SUB H	RES 2,H	
149	(f)	95	SUB L	RES 2,L	
150	(g) Gráficos	96	SUB (HL)	RES 2,(HL)	
151	(h) do utilizador	97	SUB A	RES 2,A	
152	(i)	98	SBC A,B	RES 3,B	
153	(j)	99	SBC A,C	RES 3,D	
154	(k)	9A	SBC A,D	RES 3,D	
155	(l)	9B	SBC A,E	RES 3,E	
156	(m)	9C	SBC A,H	RES 3,H	
157	(n)	9D	SBC A,L	RES 3,L	
158	(o)	9E	SBC A,(HL)	RES 3,(HL)	
159	(p)	9F	SBC A,A	RES 3,A	
160	(q)	A0	AND B	RES 4,B	LDI
161	(r)	A1	AND C	RES 4,C	CPI
162	(s)	A2	AND D	RES 4,D	INI
163	(t)	A3	AND E	RES 4,E	OUTI
164	(u)	A4	AND H	RES 4,H	
165	**RND**	A5	AND L	RES 4,L	
166	**INKEY$**	A6	AND (HL)	RES 4,(HL)	
167	**PI**	A7	AND A	RES 4,A	
168	**FN**	A8	XOR B	RES 5,B	LDD
169	**POINT**	A9	XOR C	RES 5,C	CPD
170	**SCREEN$**	AA	XOR D	RES 5,D	IND
171	**ATTR**	AB	XOR E	RES 5,E	OUTD
172	**AT**	AC	XOR H	RES 5,H	
173	**TAB**	AD	XOR L	RES 5,L	
174	**VAL$**	AE	XOR (HL)	RES 5,(HL)	
175	**CODE**	AF	XOR A	RES 5,A	
176	**VAL**	B0	OR B	RES 6,B	LDIR
177	**LEN**	B1	OR C	RES 6,C	CPIR
178	**SIN**	B2	OR D	RES 6,D	INIR
179	**COS**	B3	OR E	RES 6,E	OTIR
180	**TAN**	B4	OR H	RES 6,H	
181	**ASN**	B5	OR L	RES 6,L	
182	**ACS**	B6	OR (HL)	RES 6,(HL)	
183	**ATN**	B7	OR A	RES 6,A	
184	**LN**	B8	CP B	RES 7,B	LDDR
185	**EXP**	B9	CP C	RES 7,C	CPDR
186	**INT**	BA	CP D	RES 7,D	INDR

187	**SQR**	BB	CP E	RES 7,E	OTDR
188	**SGN**	BC	CP H	RES 7,H	
189	**ABS**	BD	CP L	RES 7,L	
190	**PEEK**	BE	CP (HL)	RES 7,(HL)	
191	**IN**	BF	CP A	RES 7,A	
192	**USR**	C0	RET NZ	SET 0,B	
193	**STR$**	C1	POP BC	SET 0,C	
194	**CHR$**	C2	JP NZ,nn	SET 0,D	
195	**NOT**	C3	JP nn	SET 0,E	
196	**BIN**	C4	CALL NZ,nn	SET 0,H	
197	**OR**	C5	PUSH BC	SET 0,L	
198	**AND**	C6	ADD A,n	SET 0.(HL)	
199	**<=**	C7	RST 0	SET 0,A	
200	**>=**	C8	RET Z	SET 1,B	
201	**<>**	C9	RET	SET 1,C	
202	**LINE**	CA	JP Z,nn	SET 1,D	
203	**THEN**	CB		SET 1,E	
204	**TO**	CC	CALL Z, nn	SET 1,H	
205	**STEP**	CD	CALL nn	SET 1,L	
206	**DEF FN**	CE	ADC A,n	SET 1,(HL)	
207	**CAT**	CF	RST 8	SET 1,A	
208	**FORMAT**	D0	RET NC	SET 2,B	
209	**MOVE**	D1	POP DE	SET 2,C	
210	**ERASE**	D2	JP NC,nn	SET 2,D	
211	**OPEN**	D3	OUT (n),A	SET 2,E	
212	**CLOSE**	D4	CALL NC,nn	SET 2,H	
213	**MERGE**	D5	PUSH DE	SET 2,L	
214	**VERIFY**	D6	SUB n	SET 2,(HL)	
215	**BEEP**	D7	RST 16	SET 2,A	
216	**CIRCLE**	D8	RET C	SET 3,B	
217	**INK**	D9	EXX	SET 3,C	
218	**PAPER**	DA	JP C,nn	SET 3,D	
219	**FLASH**	DB	IN A,(n)	SET 3,E	
220	**BRIGHT**	DC	CALL C,nn	SET 3,H	
221	**INVERSE**	DD	Prefixa instruções usando IX	SET 3,L	
222	**OVER**	DE	SBC A,n	SET 3,(HL)	
223	**OUT**	DF	RST 24	SET 3,A	
224	**LPRINT**	E0	RET PO	SET 4,B	
225	**LLIST**	E1	POP HL	SET 4,C	
226	**STOP**	E2	JP PO,nn	SET 4,D	
227	**READ**	E3	EX (SP),HL	SET 4,E	
228	**DATA**	E4	CALL PO,nn	SET 4,H	
229	**RESTORE**	E5	PUSH HL	SET 4,L	
230	**NEW**	E6	AND n	SET 4,(HL)	
231	**BORDER**	E7	RST 32	SET 4,A	
232	**CONTINUE**	E8	RET PE	SET 5,B	
233	**DIM**	E9	JP (HL)	SET 5,C	
234	**REM**	EA	JP PE,nn	SET 5,D	
235	**FOR**	EB	EX DE,HL	SET 5,E	
236	**GO TO**	EC	CALL PE,nn	SET 5,H	

237	**GO SUB**	ED		SET 5,L
238	**INPUT**	EE	XOR n	SET 5,(HL)
239	**LOAD**	EF	RST 40	SET 5,A
240	**LIST**	F0	RET P	SET 6,B
241	**LET**	F1	POP AF	SET 6,C
242	**PAUSE**	F2	JP P,nn	SET 6,D
243	**NEXT**	F3	DI	SET 6,E
244	**POKE**	F4	CALL P,nn	SET 6,H
245	**PRINT**	F5	PUSH AF	SET 6,L
246	**PLOT**	F6	OR n	SET 6,(HL)
247	**RUN**	F7	RST 48	SET 6,A
248	**SAVE**	F8	RET M	SET 7,B
249	**RANDOMIZE**	F9	LD SP,HL	SET 7,C
250	**IF**	FA	JP M,nn	SET 7,D
251	**CLS**	FB	EI	SET 7,E
252	**DRAW**	FC	CALL M,nn	SET 7,H
253	**CLEAR**	FD	Prefixa instruções usando IY	SET 7,L
254	**RETURN**	FE	CP n	SET 6,(HL)
255	**COPY**	FF	RST 56	SET 7,A

As variáveis de sistema

O número da coluna 1 é o número de bytes da variável. Quando possui dois bytes, o primeiro é o menos significativo.

Bytes	Endereço	Nome	Conteúdo
8	23552	KSTATE	Usada para leitura do teclado.
1	23560	LAST K	Indica a última tecla premida.
1	23561	REPDEL	Tempo (em quinquagésimos de segundo) durante o qual se deve carregar numa tecla antes de entrar em repetição. O seu valor inicial é 35.
1	23562	REPPER	Atraso (em quinquagésimos de segundo) entre repetições sucessivas da mesma tela: inicialmente 5.
2	23563	DEFADD	Endereço dos argumentos de uma função definida pelo utilizador se está a ser avaliada; senão zero
1	23565	K DATA	Guarda 2º byte de controlos de cor escritos no teclado.
2	23566	TVDATA	Guarda bytes de cor, e controlos de AT e TAB para a televisão.
38	23568	STRMS	Endereço do canal ligado a «streams».
2	23606	CHARS	Indicador do conjunto de caracteres.
1	23608	RASP	Duração do sinal de aviso.
1	23609	PIP	Duração do sinal das teclas.
1	23610	ERR NR	Código da mensagem de erro (menos 1).
1	23611	FLAGS	Flags Basic.

1	23612	TV FLAG	Flags da televisão
2	23613	ERR SP	Endereço de retorno de erros.
2	23615	LIST SP	Endereço de retorno de listagem automática.
1	23617	MODE	Tipo de cursor: K, L, C, E ou G.
2	23618	NEWPPC	Linha Basic para onde deve saltar
1	23620	NSPPC	Número da instrução na linha para onde deve saltar.
2	23621	PPC	Número da linha Basic onde se encontra a instrução em execução.
1	23623	SUBPPC	Número da instrução no interior da linha que está a ser executada.
1	23624	BORDCR	Cor da margem vezes 8, e atributos usados na janela inferior do visor.
2	23625	E PPC	Número da linha actual (com cursor do programa).
2	23627	VARS	Endereço das variáveis Basic.
2	23629	DEST	Endereço da variável em atribuição.
2	23631	CHANS	Endereço dos dados do canal.
2	23633	CURCHL	Endereço da informação que está a ser usada para entrada e saída.
2	23635	PROG	Endereço do programa Basic.
2	23637	NXTLIN	Endereço da linha seguinte do programa Basic.
2	23639	DATADD	Endereço do separador do último elemento de DATA.
2	23641	E LINE	Endereço da ordem que está a ser escrita.
2	23643	K CUR	Endereço do cursor.
2	23645	CH ADD	Endereço do caracter a interpretar a seguir.
2	23647	X PTR	Endereço do caracter que se segue ao marcador «?»
2	23649	WORKSP	Endereço da área de trabalho temporário.
2	23651	STKBOT	Endereço da base do «stack» do calculador.
2	23653	STKEND	Endereço do início do espaço livre.
1	23655	BREG	Registo B do calculador.
2	23656	MEM	Endereço da área usada para a memória do calculador.
1	23658	FLAGS2	Flags.
1	23659	DF SZ	Número de linhas (incluindo uma em branco) na janela inferior do visor.
2	23660	STOP	Número da superior linha do programa em listagens automáticas.
2	23662	OLDPPC	Número de linha para onde CONTINUE salta.
1	23664	OSPCC	Número da instrução dentro da linha para onde CONTINUE salta.
1	23665	FLAGX	Flags.
2	23666	STRLEN	Comprimento da cadeia em atribuição.
2	23670	SEED	Semente de RND. Variável definida por RANDOMIZE.
3	23672	FRAMES	Contador de imagens, com 3 bytes (o primeiro é o menos significativo). Incrementado em cada 20 ms.
2	23675	UDG	Endereço do 1º gráfico definido pelo utilizador.
1	23677	COORDS	Coordenada x do último ponto marcado

			no visor.
1	23678		Coordenada y do último ponto marcado no visor.
1	23679	P POSN	Número (33 colunas) da posição de impressão.
1	23680	PR CC	Byte menos significativo do endereço da posição seguinte para LPRINT (no buffer da impressora).
1	23681		Não usada.
2	23682	ECHO E	Número de coluna (33) e número de linha (24) do final do buffer de impressora.
2	23684	DF CC	Endereço da posição PRINT no ficheiro da imagem.
2	23686	DFCCL	Como DF CC para a parte inferior do visor.
1	23688	S POSN	Número de coluna (33) da posição de PRINT.
1	23689		Número de linha (24) da posição de PRÍNT.
2	23690	SPOSNL	Como S POSN para parte inferior do visor.
1	23692	SCR CT	Contador de «scrolls».
1	23693	ATTR P	Cores permanentes actuais.
1	23694	MASK P	Usado para cores transparentes, etc.
1	23695	ATTR T	Cores temporárias actuais.
1	23696	MASK T	Como MASK P, mas temporária.
1	23697	P FLAG	Flags
30	23698	MEMBOT	Área de memória do calculador.
2	23728		Não usada.
2	23730	RAMTOP	Endereço do último byte da área BASIC.
2	23732	PRAMT	Endereço do último byte da RAM física.

Tabela de conversão hexadecimal

Hexadecimal	Decimal	Binário
00	0	00000000
01	1	00000001
02	2	00000010
03	3	00000011
04	4	00000100
05	5	00000101
06	6	00000110
07	7	00000111
08	8	00001000
09	9	00001001
0A	10	00001010
0B	11	00001011
0C	12	00001100
0D	13	00001101
0E	14	00001110
0F	15	00001111
10	16	00010000
11	17	00010001
12	18	00010010
13	19	00010011
14	20	00010100
15	21	00010101
16	22	00010110
17	23	00010111
18	24	00011000
19	25	00011001
1A	26	00011010
1B	27	00011011
1C	28	00011100
1D	29	00011101
1E	30	00011110
1F	31	00011111
20	32	00100000
21	33	00100001
22	34	00100010
23	35	00100011
24	36	00100100
25	37	00100101
26	38	00100110
27	39	00100111
28	40	00101000
29	41	00101001
2A	42	00101010
2B	43	00101011
2C	44	00101100
2D	45	00101101
2E	46	00101110
2F	47	00101111
30	48	00110000
31	49	00110001
32	50	00110010
33	51	00110011
34	52	00110100
35	53	00110101
36	54	00110110
37	55	00110111
38	56	00111000
39	57	00111001
3A	58	00111010
3B	59	00111011
3C	60	00111100
3D	61	00111101
3E	62	00111110
3F	63	00111111
40	64	01000000

41	65	01000001
42	66	01000010
43	67	01000011
44	68	01000100
45	69	01000101
46	70	01000110
47	71	01000111
48	72	01001000
49	73	01001001
4A	74	01001010
4B	75	01001011
4C	76	01001100
4D	77	01001101
4E	78	01001110
4F	79	01001111
50	80	01010000
51	81	01010001
52	82	01010010
53	83	01010011
54	84	01010100
55	85	01010101
56	86	01010110
57	87	01010111
58	88	01011000
59	89	01011001
5A	90	01011010
5B	91	01011011
5C	92	01011100
5D	93	01011101
5F	94	01011110
5F	95	01011111
60	96	01100000
61	97	01100001
62	98	01100010
63	99	01100011
64	100	01100100
65	101	01100101
66	102	01100110
67	103	01100111
68	104	01101000
69	105	01101001
6A	106	01101010
6B	107	01101011
6C	108	01101100
6D	109	01101101
6E	110	01101110
6F	111	01101111
70	112	01110000

71	113	01110001
72	114	01110010
73	115	01110011
74	116	01110100
75	117	01110101
76	118	01110110
77	119	01110111
78	120	01111000
79	121	01111001
7A	122	01111010
7B	123	01111011
7C	124	01111100
7D	125	01111101
7E	126	01111110
7F	127	01111111
80	128	10000000
81	129	10000001
82	130	10000010
83	131	10000011
84	132	10000100
85	133	10000101
86	134	10000110
87	135	10000111
88	136	10001000
89	137	10001001
8A	138	10001010
8B	139	10001011
8C	140	10001100
8D	141	10001101
8E	142	10001110
8F	143	10001111
90	144	10010000
91	145	10010001
92	146	10010010
93	147	10010011
94	148	10010100
95	149	10010101
96	150	10010110
97	151	10010111
98	152	10011000
99	153	10011001
9A	154	10011010
9B	155	10011011
9D	156	10011100
9D	157	10011101
9E	158	10011110
9F	159	10011111
A0	160	10100000

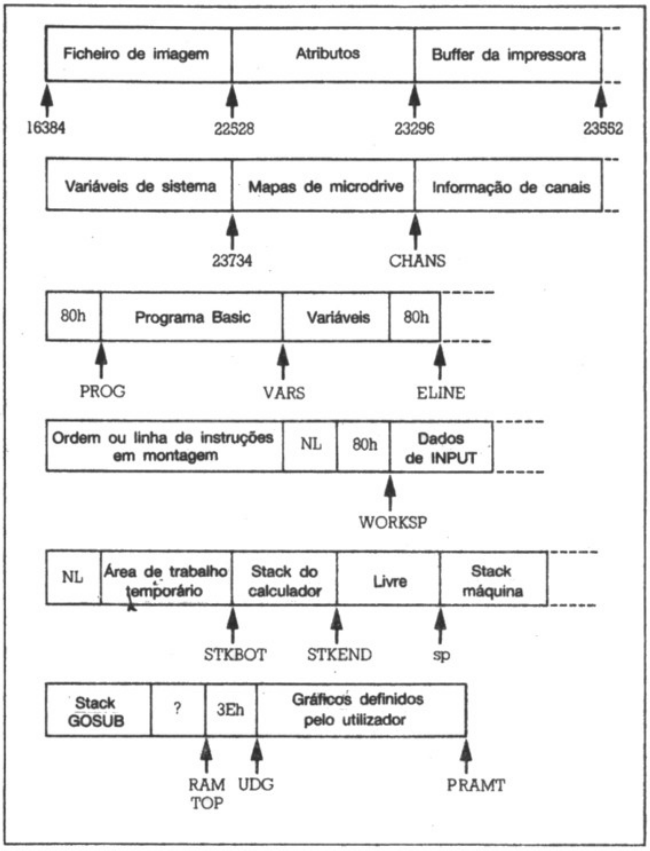
A1	161	10100001
A2	162	10100010
A3	163	10100011
A4	164	10100100
A5	165	10100101
A6	166	10100110
A7	167	10100111
A8	166	10101000
A9	169	10101001
AA	170	10101010
AB	171	10101011
AC	172	10101100
AD	173	10101101
AE	174	10101110
AF	175	10101111
B0	176	10110000
B1	177	10110001
B2	178	10110010
B3	179	10110011
B4	180	10110100
B5	181	10110101
B6	182	10110110
B7	183	10110111
B8	184	10111000
B9	185	10111001
BA	186	10111010
BB	187	10111011
BC	188	10111100
BD	189	10111101
BE	190	10111110
BF	191	10111111
C0	192	11000000
C1	193	11000001
C2	194	11000010
C3	195	11000011
C4	196	11000100
C5	197	11000101
C6	198	11000110
C7	199	11000111
C8	200	11001000
C9	201	11001001
CA	202	11001010
CB	203	11001011
CC	204	11001100
CD	205	11001101
CE	206	11001110
CF	207	11001111
D0	208	11010000

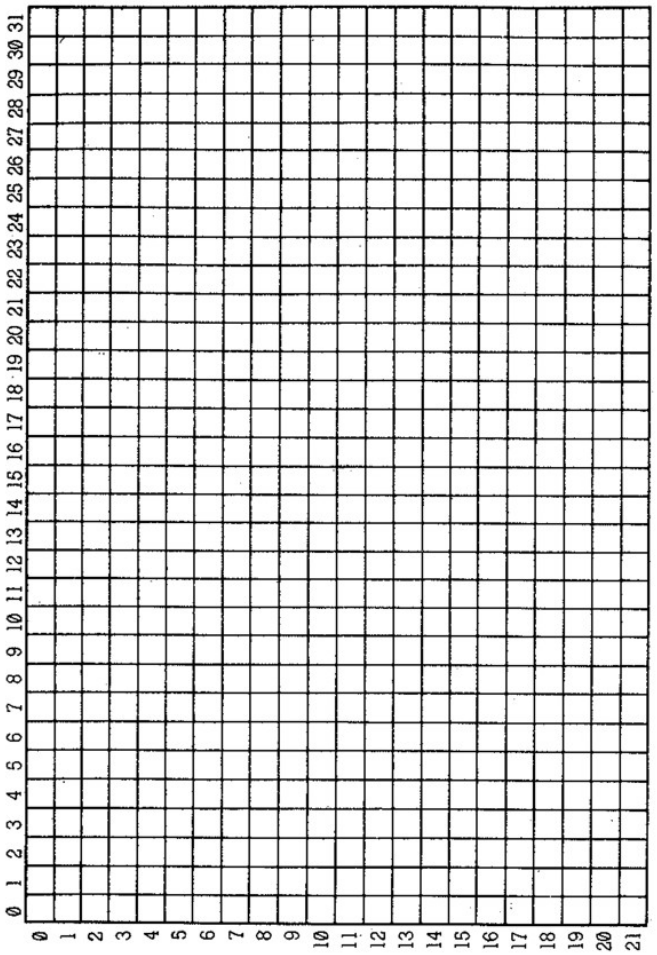
D1	209	11010001
D2	210	11010010
D3	211	11010011
D4	212	11010100
D5	213	11010101
D6	214	11010110
D7	215	11010111
D8	216	11011000
D9	217	11011001
DA	218	11011010
DB	219	11011011
DC	220	11011100
DD	221	11011101
DE	222	11011110
DF	223	11011111
E0	224	11100000
E1	225	11100001
E2	226	11100010
E3	227	11100011
E4	228	11100100
E5	229	11100101
E6	230	11100110
E7	231	11100111
E8	232	11101000
E9	233	11101001
EA	234	11101010
EB	235	11101011
EC	236	11101100
ED	237	11101101
EE	238	11101110
EF	239	11101111
F0	240	11110000
F1	241	11110001
F2	242	11110010
F3	243	11110011
F4	244	11110100
F5	245	11110101
F6	246	11110110
F7	247	11110111
F8	248	11111000
F9	249	11111001
FA	250	11111010
FB	251	11111011
FC	252	11111100
FD	253	11111101
FE	254	11111110
FF	255	11111111

Tabela de conversão hexadecimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Mapa de memória





ÍNDICE