

Este guia descreve as características e o funcionamento do apaixonante ZX SPECTRUM e explica a programação em Basic e código-máquina. O «software» e o «hardware» são completamente cobertos, para além de valiosas indicações sobre os periféricos. Não são esquecidas as potencialidades de tratamento do som e da cor. O livro inclui ainda programas originais em Basic e código-máquina e muitos exemplos de aplicação prática que permitem ao leitor explorar as capacidades do ZX SPECTRUM.



R. J. SIMPSON e T. J. TERRELL
**MANUAL
DO
ZX
SPECTRUM**





**SEDE R. Passos Manuel 67 B
Tel. 55 55 80 / 56 27 43
1100 Lisboa**

**Loja 538
Centro Comercial
Terminal do Rossio**

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Petar Trifunovitch e Sava Vukovitch*
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 Pezold*
8. Bridge Moderno, *Pierre Jais e H. Lahana*
9. Fotografia — Técnicas e Truques I, *Edwin Smith*
10. ABC dos Estilos, da Arquitetura 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 Transistores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricono*
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, *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. Coker 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 Coelho, *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. Böelle e B. 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, *Roy Genders*
70. Calendário do Horticultor, *Luis Alsina Grau*
71. Jogos Electrónicos, *F. G. Royer*
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, *Maitas Josa*
79. Rádio — Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébaud*
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, Embalsamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging — Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos T.V., *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 Coelho, *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. Perus, Pintadas e Codornizes, *Jérôme 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 Tática, *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 Ränk*
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, *Hanns-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hil*
121. Construção de um Alimentador de Corrente, *Waldemar Baitinger*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *Christu-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, *Goger Hunt*
134. Introdução à Electrónica Digital, *Ian Sinclair*
135. ABC do Vídeo, *David K. 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. Durantou*
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, *Devid Harwood*
151. As Linhas da Mão, *Rosé Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *Simpson e Terrel*

ROBERT J. SIMPSON e TREVOR J. TERREL

MANUAL DO ZX SPECTRUM

PREFÁCIO

Os computadores desenvolveram-se rapidamente a partir do momento em que foram descritos pelo matemático inglês A.M. Turing, na década de 1930, e da subsequente demonstração do Eniac (calculador e integrador numérico electrónico) por T.P. Eckert e J.W. Mauchly na Universidade da Pensylvania na década seguinte. A tecnologia micro-electrónica actual permitiu já o desenvolvimento de computadores compactos, baratos e com considerável sofisticação, e os efeitos destes na indústria, no comércio, na educação e na vida quotidiana tem sido vasto.

A produção do ZX Spectrum colocou o microcomputador ao alcance de todos. No entanto, algumas características do Spectrum podem ser difíceis de compreender para muitos utilizadores, e consequentemente o objectivo deste livro consiste em explicar as características, aspectos de programação e de funcionamento do ZX Spectrum e dos seus periféricos. O livro fornece ainda ao utilizador informações sobre o microcomputador que permitem usá-lo eficazmente e explorar as suas potencialidades em aplicações práticas.

O livro cobre os aspectos mais complicados da programação em Basic, os princípios de programação em código-máquina, detalhes de hardware e princípios de ligação a hardware externo através do ligador existente na parte traseira da máquina. São incluídos exemplos apropriados, exercícios e programas de demonstração em Basic e código-máquina para permitir ao leitor adquirir uma experiência prática e investigar e verificar os aspectos referidos no texto. O livro deve servir igualmente como útil fonte de referência ao programar e usar o computador ZX Spectrum.

Título original:

ZX SPECTRUM USER'S HANDBOOK

Originalmente publicado por Butterworth Group

© *Copyright* by Butterworth & Co. (Publishers) Ltd., Inglaterra, 1983

Tradução de Conceição Jardim e Eduardo Nogueira

Capa de Rogério Silva

Reservados todos os direitos

para a língua portuguesa à

EDITORIAL PRESENÇA, LDA.

Rua Augusto Gil, 35-A — 1000 LISBOA

Desejamos agradecer a Sue Wasson pelo seu trabalho na dactilografia do manuscrito. Agradecemos às United Technologies Mostek pela sua autorização de uso do resumo do Conjunto de Instruções do Z80A. Agradecemos ainda a Tom Izatt e às nossas filhas Andrea (S), Janet (T) e Lesley (T) por nos terem auxiliado a compreender os rudimentos das notas musicais produzidas pelo ZX Spectrum. Expressamos ainda os nossos agradecimentos a David Platt pelo seu auxílio na construção do amplificador de som e do interesse citado no texto.

Agradecemos sinceramente às nossas esposas Meryl (S) e Jennifer (T) pelo seu constante apoio e encorajamento.

Outubro de 1982

Robert J. Simpson
Trevor J. Terrell

A Jennifer e Meryl
e a Andrea, Janet e Lesley

CAPÍTULO I

COMECEMOS

Introdução

O leitor pode começar a trabalhar com o seu ZX Spectrum ligando a ficha do transformador à tomada de 9 V contínuos do computador, e ligando o cabo TV de UHF entre a tomada de antena do seu televisor a preto e branco ou a cores e a tomada TV do Spectrum.

Depois de ligar a alimentação e o aparelho de televisão, escolha um canal de televisão e sintonize-o até a frase

©1982 Sinclair Research Ltd

aparecer a negro na parte inferior do visor branco do aparelho. Provavelmente achará melhor baixar ao mínimo o volume de som do televisor quando trabalhar com o computador.

Como verificação simples do bom funcionamento do seu computador, sugerimos que carregue nas seguintes teclas pela ordem indicada. Primeiro PRINT, depois 9, depois + (obtido carregando simultaneamente em SYMBOL SHIFT e em K), depois em 8 e finalmente em ENTER. O computador calcula então a soma $9 + 8$, apresentando o resultado, 17, no canto superior esquerdo do visor. A mensagem em código apresentada pela máquina depois de cumprir esta ordem, no canto inferior esquerdo do visor, será:

0 OK, 0 : 1

o que indica que a instrução dada foi cumprida com êxito (na Tabela 1.1 apresentamos uma lista completa das mensagens fornecidas pela máquina).

O leitor pode experimentar mais algumas somas a fim de se familiarizar com o teclado e a apresentação do visor.

O Teclado

Cada uma das quarenta teclas do Spectrum está associada a pelo menos seis caracteres, símbolos e palavras, e todas elas excepto CAPS SHIFT e SYMBOL SHIFT repetem continuamente os caracteres enquanto forem premidas mais de um segundo; aproximadamente. Normalmente existe um cursor na parte inferior da imagem, indicando que pode dar à máquina qualquer instrução.

Depois de ligar o seu ZX Spectrum e de obter a frase inicial, carregue em qualquer tecla excepto CAPS SHIFT ou SYMBOL SHIFT. Notará que a frase desaparece, sendo substituída no caso de ter carregado numa tecla correspondente a um número ou a SPACE pelo número correspondente, ou um espaço, seguido do cursor K; este cursor indica que o teclado está a ser lido em modo «palavra chave» (Keyword). Pelo contrário, se carregou numa tecla correspondente a uma palavra chave, esta aparece na parte inferior do visor seguida de um cursor L, indicativo do modo «letras» (Letter).

O cursor K é sempre apresentado automaticamente pelo sistema quando este exige de si que escreva um número de linha de programa (qualquer inteiro positivo na gama 1 a 9999) ou uma palavra chave (as palavras escritas nas teclas ou nos intervalos entre estas). Quando se está a dar entrada a um programa na máquina, o cursor K ocorre automaticamente sempre que o ZX Spectrum espera uma palavra chave válida.

Depois de dar entrada à palavra chave o computador passa a apresentar o cursor L, indicando que espera de si um símbolo alfanumérico (número ou letra), SPACE ou ENTER, ou seja, os caracteres indicados a branco nas teclas, ou um dos símbolos ou palavras chave indicados a vermelho nas mesmas. Os caracteres alfabéticos indicados à máquina são sempre impressos no visor em letras minúsculas, a menos que se esteja a carregar simultaneamente em CAPS SHIFT quando se introduz o carácter. A máquina apresenta então um novo cursor, o cursor C, indicando letras maiúsculas (Capitals); os números podem também ser fornecidos à máquina em modo C. O modo C é obtido carregando simultaneamente nas teclas CAPS SHIFT e 2. O computador manter-se-á em modo C até se carregar novamente nestas duas teclas, passando então ao modo L. O símbolo ou palavra chave escritos a vermelho

nas teclas obtêm-se carregando na tecla SYMBOL SHIFT (canto inferior direito do teclado) e na tecla que corresponde à palavra ou símbolo desejados.

O modo *extenso* (Extended) é obtido carregando na tecla CAPS SHIFT e simultaneamente na SYMBOL SHIFT. O novo modo de leitura do teclado é indicado pelo cursor E, e permite a escolha de qualquer das palavras apresentadas a verde e a vermelho acima e abaixo de cada tecla, respectivamente. Para usar as palavras a verde basta carregar na tecla respectiva depois de a máquina se encontrar no modo E, enquanto que para obter as palavras a vermelho é necessário, depois de passar ao modo E, carregar simultaneamente na tecla SHIFT e em qualquer das teclas pretendidas.

Se tentar dar entrada a ordens ou dados não válidos na versão BASIC usada pelo ZX Spectrum, o sistema indicará o erro apresentando um ponto de interrogação (o cursor de erro de sintaxe «?») junto ao local onde foi cometido o erro, não aceitando a entrada. Para apagar o erro deve usar a função DELETE, carregando simultaneamente nas teclas CAPS SHIFT e 0, para remover a parte da linha que se encontra depois do erro. Alternativamente pode deslocar o cursor para a esquerda, usando ◀ (CAPS SHIFT e 5), ou para a direita usando ▶ (CAPS SHIFT e 8), até ao ponto da frase onde se encontra o erro. Pode então eliminar o erro usando DELETE e substituindo pela instrução correcta.

Para dar entrada a símbolos gráficos no seu programa em BASIC, deve escolher o modo gráfico. Para estabelecer este modo deve carregar na tecla GRAPHICS (CAPS SHIFT e 9), surgindo imediatamente no visor o cursor «G». Poderá então dar entrada a qualquer dos 16 símbolos gráficos do conjunto de caracteres do ZX Spectrum (códigos 128 a 143 da Tabela 7.1) ou a qualquer dos 21 símbolos definidos pelo utilizador (ver capítulo 7). Para obter um dos 8 símbolos gráficos impressos nas teclas 1 a 8 (códigos 128 a 135), deve então carregar numa destas teclas; para obter os 8 símbolos gráficos inversos dos anteriores (códigos 136 a 143) deve carregar conjuntamente na tecla equivalente e em CAPS SHIFT. Para obter um dos símbolos gráficos definidos pelo utilizador, que deve primeiramente ser definido do modo indicado no capítulo 7, carregará numa das teclas A a U. Finalmente, para sair do modo G, deve carregar novamente na tecla 9; volta então ao cursor (e ao modo) L.

Para se familiarizar no uso do teclado incluímos em seguida uma descrição do modo como deve ser dada entrada a um simples programa BASIC com três linhas, que ao ser executado calcula e apresenta no visor a raiz quadrada de um número.

Comece por carregar na tecla NEW em modo K e depois em ENTER, o que limpa a memória do computador e apresenta novamente a frase inicial no visor. Em seguida realize as acções indicadas na figura 1 e note o que vai aparecendo no visor.

Acção	Imagem obtida	
	em baixo	em cima
premir 5	5 K	nada
premir INPUT	5 INPUT L	nada
premir y	5 INPUT y L	nada
premir ENTER	K	5 > INPUT y
escrever 15	15 K	5 INPUT y
premir PRINT	15 PRINT L	5 > INPUT y
passar ao modo "extended" (CAPS SHIFT e SYMBOL SHIFT)	15 PRINT E	5 > INPUT y
premir SQR (tecla H)	15 PRINT SQR L	5 > INPUT y
premir y	15 PRINT SQR y L	5 > INPUT y
premir ENTER	K	5 INPUT y
		15 > PRINT SQR y
escrever 25	25 K	5 INPUT y
		15 > PRINT SQR y
premir STOP (SYMBOL SHIFT e A)	25 STOP L	5 > INPUT y
premir ENTER	K	15 > PRINT SQR y
		5 INPUT y
		15 PRINT SQR y
		25 > STOP

Figura 1

É possível incluir observações ou comentários num programa usando a declaração REM (REMark). Os comentários que se seguem à declaração REM não são lidos (e portanto não são executados) pelo ZX Spectrum. Se quiser, pode verificar isto incluindo a linha

2 REM Programa de raízes quadradas

no programa anterior.

Notará que quando cada uma das linhas surge na parte superior do visor é acompanhada pelo símbolo >. Trata-se de um indicador usado para montagem do programa. Discutiremos mais adiante, neste mesmo capítulo, as características do ZX Spectrum na montagem de programas.

O leitor acabou portanto de dar entrada ao programa, podendo em seguida pô-lo em execução. Para tal carregue em RUN e depois em ENTER. A listagem do programa desaparece do visor, surgindo em vez dela no canto inferior da imagem o cursor L, pedindo-lhe que indique o número cuja raiz quadrada quer calcular. Deve portanto dar entrada a esse número, carregando seguidamente em ENTER. Por exemplo, se o número for 25 a máquina apresentará o resultado 5 no canto superior esquerdo do visor, assim como a mensagem

9 STOP statement, 25:1

no canto inferior esquerdo. Esta mensagem indica que terminou a execução do programa na instrução STOP da linha 25 (1.ª instrução desta linha).

Para executar novamente o programa é necessário escrever RUN seguido de ENTER. Se deseja utilizar este programa para vários números diferentes talvez convenha substituir a linha 25 pelo seguinte:

25 GOTO 5

e neste caso o programa não pára depois de apresentar o resultado, sendo novamente dirigido para a linha 5. O programa encontra-se agora num ciclo (loop) contínuo, calculando e apresentando o resultado de 22 números diferentes linha a linha. Se for dada entrada a um novo número, a imagem rola, deslocando-se uma linha para cima a fim de admitir o novo número e removendo a linha superior.

Quando o programa está a executar um ciclo mas o utilizador deseja parar a execução, deve escrever STOP (SYMBOL SHIFT e A) seguido de ENTER. O programa será interrompido, surgindo no visor a mensagem.

H STOP in INPUT, 5:1

Para continuar o programa depois de um STOP escreva CONT (de CONTINUE) seguido de ENTER.

Consideremos um segundo exemplo: um programa de uma

única linha que ilustra o uso do modo G (gráficos) e o modo como o Spectrum recusa os erros de sintaxe. Depois de parar a execução do programa anterior, volte a colocar a máquina no seu estado inicial escrevendo NEW e ENTER. Em seguida faça o que se indica na figura 2, notando o que surge no visor depois de cada acção sua.

Acção	Imagem obtida	
	em baixo	em cima
premir 5	5 K	nada
premir PRINT	5 PRINT L	nada
premir " (SYMBOL SHIFT e P)	5 PRINT " L	nada
passar ao modo GRAPHICS (CAPS SHIFT e 9)	5 PRINT " G	nada
escrever ████ (tecla 6)	5 PRINT " ████ G	nada
sair do modo GRAPHICS (CAPS SHIFT e 9)	5 PRINT " ████ L	nada
premir ENTER	5 PRINT " ████ ? L	nada

Figura 2

O cursor "??" indica um erro de sintaxe: neste caso por termos esquecido o separador necessário, ", que deve ser sempre escrito no final de uma declaração PRINT. A correcção é feita do seguinte modo:

Acção	Imagem obtida	
	em baixo	em cima
premir " (SYMBOL SHIFT e P)	5 PRINT " ████" L	nada
premir ENTER	K	5 > PRINT " ████"

Figura 3

Para executar este programa carregue em RUN seguido de ENTER. Verá os três símbolos gráficos impressos no canto superior esquerdo do visor, e a mensagem

0 OK, 5 : 1

no seu canto inferior esquerdo. Esta mensagem indica uma vez mais que o programa foi executado com êxito, terminando na primeira instrução da linha 5.

Quando o utilizador mantém a pressão sobre uma dada tecla (tendo ou não carregado ao mesmo tempo numa das duas teclas SHIFT), o carácter correspondente à tecla premida é atribuído à variável de cadeia INKEY\$ é uma cadeia nula (''). Pode-se verificar isto introduzindo na máquina e fazendo executar o programa seguinte:

```
5 IF INKEY$ = "" THEN GO TO 5
15 PRINT INKEY$
25 GO TO 5
```

A execução deste ciclo contínuo pode ser terminada carregando simultaneamente nas teclas CAPS SHIFT e BREAK, o que terá como resultado a apresentação da mensagem

L BREAK into program, 5:2

O valor da variável de cadeia INKEY\$ pode ser atribuído a uma variável de cadeia (uma única letra seguida de \$) usando a declaração LET. Por exemplo, podemos modificar o programa anterior do seguinte modo:

```
5 IF INKEY$ = "" THEN GO TO 5
15 LET K$ = INKEY$
25 PRINT K$
35 GO TO 5
```

A forma e a utilidade das cadeias e variáveis de cadeia são descritas no capítulo 4. Se quiser limpar o visor use a declaração CLS (de CLEAR SCREEN).

Listagem e montagem do programa

Para listar as linhas de um programa no visor pode-se utilizar a declaração LIST. Se se carrega em LIST e depois em ENTER, o ZX Spectrum mostra as primeiras 22 linhas do programa, escrevendo em baixo

scroll?

quando o programa tem mais de 22 linhas. Esta mensagem surge porque o visor já está cheio. Se se deseja ver as 22 linhas de programa que se seguem responde-se à pergunta carregando em qualquer

tecla excepto "n", SPACE ou STOP. Este processo pode ser continuado até ser apresentada a última linha do programa. Em seguida o Spectrum apresenta a mensagem

0 OK, 0 : 1

no canto inferior esquerdo do visor. Se o programa tem menos de 22 linhas estas são obviamente apresentadas todas de uma vez.

Se se carrega em N, SPACE ou STOP em resposta à mensagem

scroll?

é apresentada no visor a mensagem

D BREAK — CONT repeats, 0 : 1

podendo-se em seguida digitar o que se quiser.

Um método alternativo de listar 22 linhas de programa consiste em usar a instrução

LIST número de linha

seguida de ENTER. Consegue-se assim listar a linha desejada na parte superior do visor, seguida pelas 21 linhas imediatas. Deste modo torna-se possível examinar qualquer bloco de 22 linhas no seu programa.

Quando deseja alterar uma linha de programa pode fazê-lo de duas maneiras. O primeiro método envolve muito simplesmente dar entrada a uma nova linha de programa do modo habitual, que substitui a linha que anteriormente possuía o mesmo número ao carregar-se na tecla ENTER. O outro método consiste em utilizar o modo de fazer "montagem" de linhas permitido pela máquina. Para alterar uma linha começa-se por colocar no visor a linha desejada, usando a ordem

LIST número de linha a montar

Se surgir a mensagem "scroll?", carregue por exemplo em "n", a fim de impedir a imagem de rolar. Em seguida accione o modo EDIT (CAPS SHIFT e 1) e a linha a montar surgirá imediatamente na parte inferior do visor, com o cursor K colocado imediatamente à frente do número de linha.

O movimento deste cursor para a direita e para a esquerda é realizado premindo as teclas CAPS SHIFT e 8 ou 9, respectiva-

mente. O carácter alfanumérico ou palavra que se encontra imediatamente à esquerda do cursor pode ser removido usando a função DELETE (CAPS SHIFT e 0), sendo possível inserir um carácter alfanumérico ou palavra escrevendo-a no teclado. Depois de alterar a linha dá-se entrada a esta, que irá substituir a original assim que se carrega na tecla ENTER.

Vale a pena notar que, para auxiliar este trabalho, é possível deslocar o cursor de uma linha para outra usando as teclas CAPS SHIFT e 7 ou 6. Note ainda que a linha que foi alterada volta à listagem com o cursor > colocado depois do número de linha.

Gravação de programas em cassette

O ZX Spectrum guarda os programas e a informação na sua memória volátil, o que significa que quando se desliga a alimentação é perdido o programa e quaisquer dados contidos na máquina. No entanto, quando se deseja guardar um programa para uso ulterior pode-se copiá-lo da memória do microcomputador para uma cassette; quando se deseja executar novamente o programa carrega-se este da cassette para a memória do ZX Spectrum.

Para transferir um programa do ZX Spectrum para uma cassette é necessário ligar a tomada MIC do Spectrum à correspondente entrada do gravador. É necessário assegurar que o outro dos dois cabos que ligam a máquina ao gravador, ou seja, o ligado à tomada EAR, é removido. Em seguida, com a fita na posição em que se deseja gravar o programa, escreve-se no teclado

SAVE "nome do programa"

e carrega-se na tecla ENTER. Note que o nome do programa só pode ter até dez caracteres. O computador responde imprimindo na última linha do visor a mensagem:

Start tape, then press any key

avisando-o de que deve pôr a cassette em movimento e carregar em seguida em qualquer tecla. Se estiver a usar um televisor a cores observará neste linhas horizontais em movimento, de cores vermelho e azul claro, seguidas de faixas azuis e amarelas (escuras e claras no caso de um televisor a preto e branco). Quando estas linhas desaparecem surge no visor a mensagem:

0 OK, 0 : 1

Esta mensagem indica que a gravação foi terminada. Pode em seguida parar o gravador de cassettes.

Para verificar se a gravação ficou bem feita pode usar a ordem VERIFY. Para isso deve ligar novamente o cabo à tomada EAR, rebobinando a fita para o ponto onde iniciou a gravação. Em seguida escreva no teclado:

VERIFY "nome do programa"

e ponha o gravador em movimento. Depois de apresentar uma sucessão de diferentes cores no rebordo do visor, surgem faixas azuis e amarelas semelhantes às observadas durante a gravação. Em seguida surge novamente no visor a mensagem

0 OK, 0 : 1

indicando que a gravação está correcta. Deve em seguida parar o gravador de cassettes.

Se o programa não for correctamente gravado o ZX Spectrum apresentará, ao verificar a gravação, a mensagem

R Tape Loading error, 0 : 1

Deve então verificar as ligações, regular os comandos de volume e tonalidade do gravador, e repetir em seguida as operações de gravação e verificação.

Para transferir um programa de uma cassette para o ZX Spectrum é necessário colocar a fita no ponto onde se inicia a gravação, verificar se a tomada EAR da máquina está ligada à tomada correspondente do gravador e em seguida escrever no teclado

LOAD "nome do programa"

pondo o gravador em movimento depois de carregar na tecla ENTER. No final o computador apresenta a mensagem

0 OK 0 : 1

O nome do programa será impresso no visor durante a carga. Depois de esta ter terminado, carrega-se na tecla RUN para começar a executar o programa.

Quando o computador recebe ordem de carregar um programa gravado em cassette, a máquina começa por "apagar" o pro-

grama e variáveis que se encontram na sua memória. No entanto, é por vezes conveniente manter um programa já existente em memória ou parte dele, carregando simultaneamente um outro a partir de uma cassette. Pode-se conseguir isto usando a declaração MERGE. Quando se utiliza esta, só serão eliminadas as linhas do programa anterior que possuem um número igual ao das linhas do novo programa. Do mesmo modo, só são eliminadas as variáveis em memória que têm as mesmas características das variáveis do novo programa. Esta operação é realizada escrevendo no teclado

MERGE "nome do programa"

e accionando o gravador do modo já descrito.

É igualmente possível gravar em cassette conjuntos de números, caracteres ou bytes, assim como voltar a carregá-los na máquina.

A ordem

SAVE "nome do programa" DATA nome do quadro ()

é usada para gravar os dados contidos num quadro (array) com um determinado nome, podendo esse quadro ser depois carregado novamente na máquina. Nos capítulos 3 e 4 serão fornecidos pormenores sobre os quadros numéricos e de cadeias.

Para gravar o conteúdo de quaisquer bytes da memória do computador usa-se a ordem

SAVE "nome de Bytes" CODE endereço do primeiro
byte a gravar, número de bytes a gravar

Para carregar na máquina usa-se a instrução

LOAD "nome de Bytes" CODE endereço do primeiro
byte a carregar, número de bytes a carregar

Durante esta operação o visor apresenta os dizeres

Bytes: nome de bytes

e depois de ser terminada é impressa a mensagem

0 OK, 0 : 1

na última linha do visor.

A imagem apresentada no visor é guardada em 6912 bytes da memória do computador, 6144 dos quais guardam o ficheiro de imagem (as figuras, números ou caracteres presentes no visor) e os 768 bytes restantes o ficheiro de atributos, isto é, as características de cor de tinta, cor de fundo, brilho e cintilação de cada carácter. O primeiro destes bytes tem o endereço 16 384, pelo que se escrever

```
SAVE "visor" CODE 16384, 6912
```

gravará em cassette uma cópia daquilo que o visor apresenta nesse momento. Para carregar esta imagem de novo para a máquina deve usar a ordem

```
LOAD "visor" CODE 16384, 6912
```

A declaração CODE 16384, 6912 é tão usada que a máquina aceita uma outra em sua substituição, facilitando a escrita das ordens de gravação e carga de imagens: a declaração SCREEN\$. Com efeito, pode-se usar a ordem

```
SAVE "visor" SCREEN$
```

para gravar em cassette o conteúdo da imagem do televisor, e

```
LOAD "visor" SCREEN$
```

para carregar na máquina essa mesma imagem. Também neste caso o nome "visor" pode ter até 10 caracteres.

Mensagens

Ao utilizar o computador verificará que este imprime frequentemente mensagens na última linha do visor, e aliás já apresentámos até agora algumas destas mensagens.

O computador apresenta uma mensagem sempre que acaba de executar uma tarefa, tanto no caso de a realizar correctamente como quando ocorre um erro. Todas as mensagens incluem um código alfanumérico simples e uma frase curta indicando o que se passou, e a linha e o número de instrução dessa linha onde o programa parou.

O formato da mensagem é portanto o seguinte:

código da mensagem, descrição curta,
último número de linha executado,
número de instrução dentro dessa linha

As instruções contidas em cada linha de programa podem ser identificadas por um número. A instrução 1 é logicamente a primeira, a instrução 2 é aquela que se encontra depois da primeira declaração THEN ou do primeiro separador ":", etc. Se a máquina está a executar uma ordem e não uma linha de programa, indica em vez do número de linha o valor 0; este é, de facto, reservado para as ordens digitadas directamente no teclado. A Tabela 1.1 apresenta um resumo das mensagens e dos respectivos códigos.

Tabela 1.1

MENSAGENS ZX SPECTRUM

Código	Significado
0	OK
1	NEXT usada sem um FOR anterior
2	Variável desconhecida pela máquina
3	Índice de variável incorrecto
4	Falta de espaço de memória
5	Falta de espaço de visor
6	Número demasiado grande
7	RETURN usada sem um GO SUB anterior
8	Final de ficheiro
9	Declaração STOP
A	Argumento não válido
B	Inteiro fora da gama aceite
C	Sem sentido em BASIC
D	BREAK — CONTINUE repete
E	Sem dados
F	Nome de ficheiro não válido
G	Falta de espaço para a linha
H	STOP em INPUT
I	FOR usado sem NEXT correspondente
J	Dispositivo de entrada/saída não válido

K	Cor não válida
L	BREAK no programa
M	RAMTOP não aceitável
N	Instrução perdida
O	Fluxo inválido
P	FN usada sem DEF
Q	Erro de parâmetros
R	Erro na carga do programa

O ligador externo

O ligador externo do ZX Spectrum contém ligações de circuito ao microprocessador Z80A e linhas de comando especiais que permitem a ligação de dispositivos periféricos ao seu Spectrum. Por exemplo, a impressora ZX Printer e o ZX Microdrive foram concebidos para ligação directa nessa tomada.

Como o ligador externo contém pernes de ligação ao microprocessador Z80A, é possível conceber circuitos de interface e comando que permitam o uso do Spectrum para comandar sistemas exteriores. Pode-se consegui-lo utilizando alguns dos interfaces existentes no mercado, mas é também possível conceber e fabricar um especialmente para as nossas necessidades. No capítulo 9 consideraremos alguns aspectos dos sistemas de entradas/saídas do ZX Spectrum, discutindo o ligador externo e os sinais nele disponíveis.

A impressora ZX Spectrum

A impressora permite-nos obter um registo permanente da imagem do visor de televisão, assim como a impressão de longos programas e de listas de resultados. A saída da impressora consiste em 32 colunas, tal como o visor, e tantas linhas quanto se quiser. Para obter cópias usam-se as instruções LPRINT, LLIST e COPY.

A instrução LPRINT é usada do mesmo modo que a instrução PRINT, diferindo apenas por produzir uma saída na impressora em vez de comandar o visor. Se por exemplo se escrever

LPRINT "Gosto de computadores"

seguido de ENTER, a mensagem "Gosto de computadores" será impressa em papel.

Para copiar linhas do programa na impressora ZX Printer usa-se a instrução LLIST. Esta funciona de modo semelhante à declaração LIST usada para apresentar um máximo de 22 linhas de programa no visor de televisão, mas a declaração LLIST provoca a impressão de *todas* as linhas do programa. Esta operação pode ser terminada carregando nas teclas CAPS SHIFT e BREAK. Para começar a imprimir a partir de uma linha especificada, escreve-se

LLIST número de linha

A declaração COPY é usada para obter uma cópia de toda a imagem do visor. Pode-se recorrer a ela para obter gráficos, tabelas de dados, etc.

Deve-se notar que é sempre possível parar a impressora carregando em BREAK (CAPS SHIFT e SPACE).

Observações finais

Neste capítulo apresentámos o ZX Spectrum e estudámos o teclado e o seu funcionamento. Apresentámos também as operações de programação, montagem de programas, gravação destes e de dados em cassette, o significado das mensagens do computador e a utilidade do ligador externo. Estas indicações serão úteis ao estudar os capítulos subsequentes.

CAPÍTULO II

NÚMEROS BINÁRIOS E HEXADECIMAIS

Introdução

Usamos tanto os números decimais na nossa vida quotidiana actual que não é surpreendente verificar que os dados usados nos programas BASIC se apresentam geralmente nessa forma. A típica instrução de programa.

25 PRINT 17.4/6.29

utiliza o número decimal 25 como número de linha, e os dois valores decimais 17,4 e 6,29 como dados da instrução. O ZX Spectrum executa a instrução imprimindo a resposta 2.7662957 (outro número decimal) no visor. Não esqueça que a vírgula decimal é representada, em inglês, por um ponto.

Os circuitos electrónicos que se encontram no interior do microcomputador não são concebidos para tratar directamente números decimais. De facto, todos os números decimais usados num programa são convertidos pelo sistema em formas equivalentes susceptíveis de serem aceites pelos circuitos electrónico. Estas formas equivalentes são de facto representações dos mesmos valores na base binária. Usam-se números binários inteiros, por exemplo, quando incluímos instruções em código-máquina num programa, ou quando definimos os *atributos* de uma character impresso no visor (se bem que também seja possível fazê-lo em decimal).

Os números binários

O ZX Spectrum utiliza circuitos integrados capazes de assumir dois estados diferentes. Chama-se-lhes estados binários, e representam dois estados de tensão diversos, por exemplo 0 e + 5 volts. É prático representar o nível de tensão zero pelo símbolo 0 e o nível de tensão + 5 pelo símbolo 1. Os símbolos 0 e 1 são chamados *bits* (binary digits, algarismos binários).

Os números podem ser representados por uma combinação destes algarismos binários, dado que existe uma correspondência biunívoca entre cada representação decimal e binária. Por exemplo, o número binário 11100010 é um modo de escrever:

$$\begin{aligned} & (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + \\ & + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = \\ & = 128 + 64 + 32 + 20 + 0 + 0 + 2 + 0 = \\ & = 226 \end{aligned}$$

Exercício

Use este método para verificar que

$$10101010 = 170$$

Talvez considere mais fácil utilizar a pseudo-função BIN para converter este número binário em decimal, escrevendo

```
PRINT BIN 10101010
```

Note que esta pseudo-função BIN define o número como estando em forma binária, mas não esqueça que BIN só pode tratar números binários inteiros com um comprimento máximo de 16 bits (*palavra* de 16 bits). Se quiser converter um número binário contendo mais do que dezasseis bits para a sua forma decimal pode utilizar o esquema geral indicado em seguida.

Um número binário inteiro de i bits dividido em j bits (grupo mais significativo) e k bits (grupo menos significativo), onde $0 \leq i \leq 32$, $0 \leq j \leq 16$ e $0 \leq k \leq 16$, possui um valor decimal equivalente dado por

$$\begin{aligned} & (2^k \text{ valor decimal da palavra de } j \text{ bits}) + \\ & + \text{valor decimal da palavra de } k \text{ bits} \end{aligned}$$

Em Basic pode ser calculado fazendo

```
PRINT 2↑k * BIN palavra de j bits + BIN  
palavra de k bits
```

Por exemplo, consideremos a palavra de 23 bits 11011110111101100101110 dividida do seguinte modo:

$$\begin{array}{cc} \underbrace{11011110111101}_{j = 14 \text{ bits}} & \underbrace{110010110}_{k = 9 \text{ bits}} \end{array}$$

Isto é convertido para decimal fazendo

```
PRINT 2↑9 * BIN11011110111101 + BIN110010110
```

e produz como resposta 7306134.

Exercício

Verifique que a palavra de 23 bits anterior dividida sob a forma

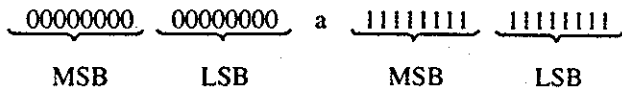
$$\begin{array}{cc} \underbrace{1101111011}_{j = 10 \text{ bits}} & \underbrace{1101110010110}_{k = 13 \text{ bits}} \end{array}$$

produz o mesmo resultado decimal (7306134) usando

```
PRINT 2↑12 * BIN1101111011 + BIN1101110010110
```

Uma palavra binária de 8 bits é designada por *byte*. Um único byte pode representar qualquer inteiro na gama zero (00000000) a 255 (11111111). Alternativamente, um byte do ZX Spectrum pode ser usado para representar um carácter do Conjunto de Caracteres. Por exemplo, o byte 00100100 (código 36) representa o carácter \$ (ver a Tabela 7.1).

Um número binário de dezasseis bits, consistindo em dois bytes, um dos quais é o menos significativo (LSB em inglês) e o outro o mais significativo (MSB), pode representar qualquer byte na gama



ou seja, 0 a 65535.

Equivalente binário de um número decimal

O número decimal a converter é primeiramente separado nas suas partes inteira e fraccionária, sendo cada uma delas convertida separadamente. Depois da conversão juntam-se as duas partes de modo a obter o resultado.

O equivalente binário da parte inteira do número decimal pode ser obtida dividindo repetidamente o número decimal por 2, formando em cada divisão um quociente e um resto. O processo de divisão é continuado até o quociente ser zero, e os restos constituem o número na sua forma binária. O último resto é o bit menos significativo do número binário. O exemplo seguinte ilustra este método.

	Quociente	Resto
226/2 =	113	0
113/2 =	66	1
56/2 =	28	0
28/2 =	14	0
14/2 =	7	0
7/2 =	3	1
3/2 =	1	1
1/2 =	0	1

Ler de baixo para cima

ou seja, 226 (decimal) = 11100010 (binário).

Exercício

Use este método para verificar que

$$174 = 10101110$$

$$240 = 11110000$$

O programa 1 do Apêndice A pode ser usado para converter um decimal inteiro no seu equivalente binário.

A parte fraccionária do número decimal é convertida multiplicando repetidamente a fracção por 2 e anotando o número inteiro assim produzido. Este processo é repetido até ter sido obtida a aproximação suficiente — pode não ser possível obter um equivalente exacto. O exemplo que se segue ilustra este método:

Virgula binária →	0,8
	× 2
	1 ← 1,6
	× 2
	1 ← 1,2
	× 2
	0 ← 0,4
	× 2
	0 ← 0,8
	× 2
	1 ← 1,6
	× 2
	1 ← 1,2
	× 2
	0 ← 0,4
	× 2
	0 ← 0,8 etc.

Ler de cima para baixo

ou seja, 0,8 decimal ~ 0,11001100 em binário.

Exercícios

1. Use este método para verificar que

$$0,140625 = 0,0010010$$

$$0,9013671875 = 0,1110011011$$

2. Mostre que o equivalente binário do número decimal 25,34375 é 11001,01011.

Equivalente decimal de um número binário

O número binário a converter é primeiramente dividido nas suas partes inteira e fraccionária, sendo cada uma delas convertida separadamente. Depois da conversão as duas partes são combinadas de modo a obter o resultado.

O equivalente decimal da parte inteira do número binário é obtido multiplicando o bit mais significativo do número binário por 2 e somando o bit mais significativo a seguir ao resultado do produto. O resultado é então multiplicado por 2. O bit menos significativo que se segue é então somado ao resultado, multiplicando-se novamente por 2. Continuamos este processo até terem sido usados todos os algarismos binários. O exemplo seguinte ilustra este método.

$$\begin{array}{r}
 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \\
 \times 2 \\
 \hline
 2 \quad + 0 \\
 \hline
 4 \quad + 0 \\
 \hline
 8 \quad + 1 \\
 \hline
 18 \quad + 0 \\
 \hline
 36 \quad + 0 \\
 \hline
 72 \quad + 1 \\
 \hline
 146 \quad + 1 \\
 \hline
 147
 \end{array}$$

ou seja, 10010011 (binário) = 147 (decimal).

O programa 1 do Apêndice A pode também ser usado para converter um número binário inteiro na sua forma decimal equivalente.

Exercício

Use este método para verificar que

$$\begin{array}{l}
 101101 = 45 \\
 11011000 = 216
 \end{array}$$

O equivalente decimal da parte fraccionária do número binário é obtido dividindo o bit menos significativo do número binário por 2 e somando o bit menos significativo seguinte ao resultado da divisão. Este processo é continuado até ter sido somado o bit mais significativo do número binário original, e o resultante dividido por 2.

O exemplo seguinte ilustra este método

$$\begin{array}{r}
 \text{Vírgula} \\
 \text{binária} \rightarrow , \\
 \begin{array}{r}
 0 \\
 1 \\
 1 \\
 0
 \end{array}
 \end{array}
 \begin{array}{l}
 \\
 \\
 2)1,25 = 0,625 \\
 2)0,5 = 0,25
 \end{array}
 \begin{array}{l}
 \\
 \\
 \\
 \\
 \end{array}
 \begin{array}{l}
 \text{Vírgula} \\
 \text{decimal} \\
 \downarrow \\
 2)0,8125 = ,40625 \text{ (final)} \\
 2)1,625 = 0,8125 \\
 2)1,25 = 0,625 \\
 2)0,5 = 0,25 \\
 \end{array}$$

(Início) $\overline{)1} = 0,5$

ou seja, $0,01101$ (binário) = $0,40625$ (decimal)

Exercício

1. Use este método para verificar que

$$\begin{array}{l}
 0,11011 = 0,84375 \\
 0,101001 = 0,650625
 \end{array}$$

Mostre que o equivalente decimal do número binário $1011001,100111$ é $89,609375$.

Números hexadecimais

Os números hexadecimais constituem uma representação muito útil dos seus equivalentes binários. Esta representação utiliza os símbolos 0 a 9 e os caracteres alfabéticos A a F, como se mostra na Tabela 2.1.

Podemos ver na tabela 2.1 que um número binário de oito bits é representado por dois algarismos hexadecimais. O menos significativo destes representa os 4 bits menos significativos da palavra binária e o mais significativo representa os 4 bits mais significativos da palavra binária. Em função da posição que ocupa num número é atribuído um valor a cada algarismo; o menos significativo possui um valor posicional (peso) de $1(16^0)$, e o mais significativo o peso de $16(16^1)$.

Por exemplo B7 (10110111) pode ser escrito sob a forma

$$\begin{aligned} & (B \times 16^1) + (7 \times 16^0) \\ &= (11 \times 16) + (7 \times 1) \\ &= 183 \end{aligned}$$

Exercício

Usando este método tente demonstrar que

$$E6 = 230$$

Tabela 2.1

Tabela de conversão de códigos

Decimal	Hexadecimal	Binário
0	00	00000000
1	01	00000001
2	02	00000010
3	03	00000011
4	04	00000100
5	05	00000101
6	06	00000110

7	07	00001111
8	08	00001000
9	09	00001001
10	0A	00001010
11	0B	00001011
12	0C	00001100
13	0D	00001101
14	0E	00001110
15	0F	00001111
16	10	00010000
17	11	00010001
:	:	:
254	FE	11111110
255	FF	11111111

Note que os números hexadecimais não utilizam apenas dois símbolos (dois algarismos). Um exemplo típico do uso de quatro algarismos hexadecimais no ZX Spectrum é a utilização de uma palavra de 16 bits (dois bytes) para indicar o endereço de uma posição de memória. Por exemplo, o número hexadecimal 5CB2 escrito em forma binária é:

5	C	B	2
↓	↓	↓	↓
0101	0100	1011	0010

ou seja, 5CB2 (hex) = 0101110010110010 (binário).

Exercício

Usando este método mostre que

$$\begin{aligned} C3 &= 11000011 \\ BAD1 &= 1011101011010001 \end{aligned}$$

Quando se converte um número hexadecimal inteiro para a forma decimal equivalente pode ser prático convertê-lo primeiro para a forma binária e usar depois a pseudo-função BIN para converter de binário para decimal, como se descreveu anteriormente.

O programa 1 do Apêndice A pode ser usado para converter um número inteiro decimal no seu número hexadecimal equivalente.

A parte fraccionária do número decimal é convertida multiplicando repetidamente a fracção decimal por 16 e anotando a parte inteira produzida (que é substituída pelo algarismo hexadecimal equivalente). Este processo é repetido até ter sido obtida uma aproximação suficiente — pode não ser possível obter um equivalente exacto. O exemplo seguinte ilustra este método.

Vírgula hexadecimal	0,8
→ ,	× 16 (princípio)
C ← subs. por ←	12,8
equivalente hexadecimal	× 16
C ←	12,8
	× 16
C ←	12,8
	etc.

ou seja, 0,8 (decimal) ~ 0,CCC (hex).

Exercícios

- Use este método para verificar que

$$\begin{aligned} 0,890625 &= 0,E4 \\ 0,9013671875 &= 0,E6C \end{aligned}$$

- Mostre que o equivalente hexadecimal do número decimal

$$33,84375 \text{ é } 21,D8$$

Equivalente decimal de um número hexadecimal

O número hexadecimal a converter é primeiramente separado nas suas partes inteira e fraccionária, que são uma vez mais convertidas separadamente. Depois da conversão são combinadas de modo a obter o resultado final.

O equivalente decimal da parte inteira do número hexadecimal é obtido convertendo o caracter mais significativo deste para o seu equivalente decimal e multiplicando-o por dezasseis; depois converte-se o algarismo mais significativo a seguir e soma-se ao produto anterior. O resultado é em seguida multiplicado por 16. Passa-se sucessivamente aos outros algarismos, até todos os caracteres do número hexadecimal terem sido usados. O exemplo seguinte ilustra este método.

(Princípio)	F	2	A	6
substituir por equivalente decimal			substituir por equivalente decimal	
	15			
	× 16			
	240			
		+ 2		
		242		
		× 16		
		3872	+ 10	
			3882	
			× 16	
			62112	+ 6
				62118 (Fim)

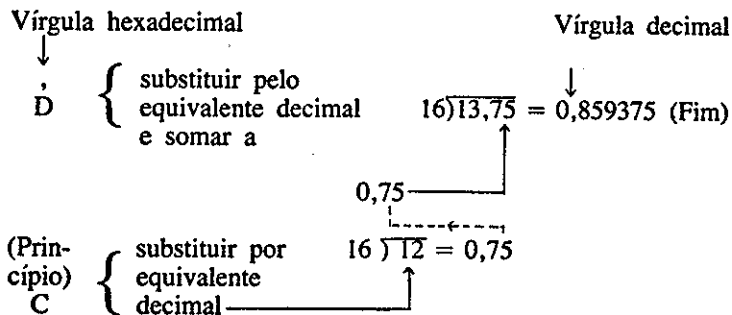
ou seja, F2A6 (hex) = 62118 (decimal).

O programa 1 do Apêndice A pode ser usado para converter um número inteiro hexadecimal na sua forma decimal equivalente.

O equivalente decimal da parte fraccionária do número hexadecimal é obtido convertendo o caracter menos significativo deste no seu equivalente decimal e dividindo em seguida por 16. O algarismo menos significativo seguinte é convertido para o seu equivalente decimal e somado ao resultado da divisão. Este método é prosseguido até ter sido convertido o algarismo mais significativo da parte fraccionária e ter-se somado o seu equivalente decimal ao

valor anterior; em seguida divide-se uma vez mais por dezasseis.

O exemplo seguinte ilustra este método:



ou seja, 0,DC (hex) = 0,859375 (decimal).

As partes inteira e fraccionária são combinadas de modo a dar o resultado final:

$$F2A6,DC = 62118,859375$$

Exercício

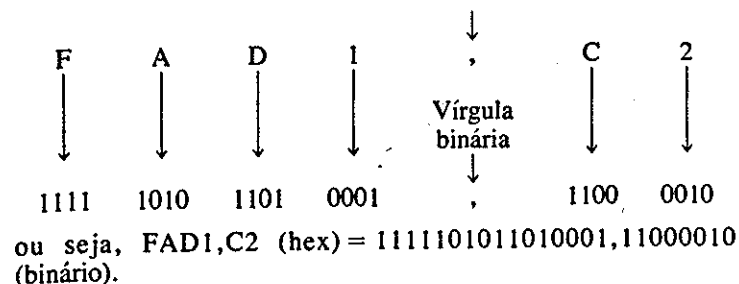
Mostre que o equivalente decimal do número hexadecimal

$$B2,F \text{ é } 178,9375$$

Equivalente binário de um número hexadecimal

Já vimos na Tabela 2.2 que a forma hexadecimal de um número constitui uma representação mais simples do seu equivalente binário. Consequentemente, para converter um número hexadecimal no seu equivalente binário basta substituir cada algarismo hexadecimal pelo algarismo binário equivalente. Por exemplo:

Vírgula hexadecimal



Exercício

Mostre que o equivalente binário de E49,D é

$$111001001001,11010001$$

Números armazenados na memória

O microprocessador Z80A utiliza uma palavra binária de 16 bits para endereçar uma posição de memória. No entanto, é mais conveniente usar o equivalente decimal da palavra de endereçamento de 16 bits, e portanto o programa interpretador da BASIC contido no ZX Spectrum permite-nos indicar um endereço de memória usando um número decimal.

Para examinar o conteúdo de qualquer posição de memória utilizamos a função PEEK e o endereço da posição de memória em forma decimal. Por exemplo

PRINT PEEK 1983

apresentará no visor o conteúdo da posição 1983 da memória de leitura (ROM), que possui um valor fixo de 58. Sugiro ao leitor que faça esta experiência para verificar o resultado.

Use o programa seguinte para apresentar no visor o conteúdo de um bloco de bytes de memória ROM, que ocupa os endereços 0 a 16383. Se escolher um bloco de mais de 22 posições de memória ou bytes pode "rolar" (scroll) a imagem carregando em qualquer tecla excepto N, SPACE ou STOP, pois qualquer destas provoca um BREAK no cumprimento da ordem dada ao computador.

```

2 PRINT "Indique primeiro endereço" : INPUT pe :
  PRINT pe
3 PRINT "Indique último endereço" : INPUT ue : PRINT
  ue
5 PAUSE 50
7 CLS
8 IF ue < pe THEN PRINT "O último endereço é inferior
  ao", "primeiro — repita." : GO TO 2
10 FOR n = pe TO ue
20 PRINT n,
25 PRINT PEEK n
30 NEXT n

```

Note que pode usar o programa 4 do Apêndice A para obter uma listagem hexadecimal do conteúdo de todas as posições da ROM.

Só é possível alterar o conteúdo das posições de memória da RAM (Random Access Memory, memória de acesso aleatório). Podemos fazê-lo usando a declaração POKE juntamente com o endereço da posição de memória e os dados em forma decimal. Por exemplo

```
POKE 17111,129
```

carregará a palavra binária 10000001 (129) na posição de memória 0100001011010111 (17111), correspondente ao ficheiro de imagem. O padrão de oito bits correspondentes é apresentado na posição do visor correspondente (ver o capítulo 7 para mais detalhes sobre gráficos). Talvez o leitor compreenda agora porque razão a representação decimal é mais prática do que a binária. É evidentemente possível verificar se o valor pretendido foi armazenado no byte desejado usando a função PEEK:

É possível guardar números inteiros na gama 0 a 255 usando a declaração POKE. Se tentar armazenar em memória números inteiros maiores do que 255 o computador responderá com a seguinte mensagem:

B Integer out of range

indicando o uso de um número inteiro excessivamente grande.

Deve notar que no caso de executar um POKE com números inteiros na gama -1 a -255 estes são aceites e guardados sob a forma de complemento. Isto significa que se, por exemplo, der ao computador a ordem POKE -123 este valor será de facto guardado sob a forma 133. Note que $133 = 256 - 123$, o que significa que o inteiro armazenado é o complemento de 123 para 256, obtido por subtracção do valor indicado de 256.

Se tentar guardar em memória um número não inteiro, este será arredondado para o inteiro mais próximo e guardado nesta forma. Por exemplo, experimente fazer POKE 4.73 na posição de memória 17111. Pode em seguida executar um PEEK sobre este byte, verificando que o número nele guardado é 5. Se experimentar fazer POKE 5.5 verificará que é guardado sob a forma de 6, enquanto que 5.4999 é guardado sob a forma de 5. Um número negativo será arredondado e guardado na sua forma complementar. Por exemplo, -13,417 é guardado sob a forma 243 (isto é, $256 - 13$).

Aritmética binária

O microprocessador Z80A usado no ZX Spectrum realiza operações aritméticas usando representações binárias de números, e portanto, ao nível de código-máquina, convém que o utilizador compreenda um pouco dos conceitos da aritmética binária. Para realizar operações aritméticas binárias é necessário apenas conhecer e aplicar as regras básicas da adição, subtracção, multiplicação e divisão.

Adição

As regras básicas da adição binária são

$$\begin{aligned}0 + 0 &= 0 \\0 + 1 &= 1 \\1 + 0 &= 1 \\1 + 1 &= 0 \text{ com transporte de } 1 \\1 + 1 + 1 &= 1 \text{ com transporte de } 1\end{aligned}$$

Em seguida apresentamos dois exemplos de adições de dois números binários de quatro bits, com os valores decimais equivalentes:

Decimal	Binário	Decimal	Binário
8	1000	7	0111
+ 7	+ 0111	+ 5	+ 0101
<hr/>		<hr/>	
15	1111	12	1100

Nestes dois exemplos simples vemos que a soma de dois números binários de quatro bits produz um resultado também de quatro bits. No entanto, se a soma exceder 15 o resultado será uma palavra de cinco bits. Por exemplo

Decimal	Binário
9	1001
+ 8	+ 1000
<hr/>	
17	10001

Vemos assim que a soma de duas palavras de quatro bits requer de facto $n+1$ bits para representar correctamente o resultado.

Exercício

Usando as regras da soma binária mostre que

$$1101 + 0110 = 10011$$

Subtracção

A subtracção binária pode ser realizada usando as regras básicas da subtracção binária, que são as seguintes:

$$\begin{aligned}0 - 0 &= 0 \\1 - 0 &= 1 \\1 - 1 &= 0 \\0 - 1 &= 1 \text{ com transporte } - 1 \\1 - 1 - 1 &= 1 \text{ com transporte } - 1\end{aligned}$$

Em seguida mostram-se dois exemplos de subtracção de dois números binários de quatro algarismos com os seus valores decimais correspondentes.

Decimal	Binário	Decimal	Binário
14	1110	12	1100
- 3	- 0011	- 7	- 0111
<hr/>		<hr/>	
11	1011	5	0101

Nestes dois exemplos de subtracção os resultados são números positivos, mas o resultado de uma subtracção pode ser positivo ou negativo em função das grandezas relativas de ambos. A fim de distinguir o sinal de um número usa-se o bit mais à esquerda (o mais significativo, MSB) como *bit de sinal*. Normalmente usa-se a convenção que atribui o valor 1 ao sinal negativo e 0 ao sinal positivo. Consequentemente o microprocessador Z80A, que possui oito bits para armazenamento de dados, utiliza sete deles para o dado propriamente dito e o último para indicar o sinal.

São necessários complexos circuitos electrónicos para subtrair directamente os números binários, e nos microcomputadores é habitual realizar as subtracções somando o complemento para dois do diminuidor ao diminuendo. Isto significa que se torna possível dispensar o circuito que executa a subtracção, realizando esta através de circuitos somadores, inclusivamente ao determinar a forma complementar do diminuidor.

O complemento para dois de um número binário é determinado invertendo cada um dos algarismos do número, ou seja, transformando todos os 0s em 1s e todos os 1s em zeros, e somando um ao resultado. Por exemplo, o complemento para dois de 76,

em representação binária, é determinado do seguinte modo:

$$\begin{array}{r}
 \text{Bit de sinal} \\
 \downarrow \\
 + 76 = 01001100 \\
 \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \text{ Inverter os zeros e os uns} \\
 10110011 \\
 \hline
 + 1 \text{ Somar 1} \\
 \hline
 10110100 = -76 \text{ em forma "complemento para dois"}.
 \end{array}$$

Para realizar a subtração usando a complementação para dois é necessário obter o complemento para dois do diminuidor, que é em seguida somado ao diminuendo. Como exemplo consideremos a subtração de 44 (decimal) a 15 (decimal), usando palavras binárias de oito bits.

$$\begin{array}{cc}
 \text{Bit de sinal} & \text{Bit de sinal} \\
 \downarrow & \downarrow \\
 15 = 00001111, & 44 = 00101100
 \end{array}$$

de onde o complemento para dois de 44 é -44:

$$\begin{array}{c}
 \text{Bit de sinal} \\
 \downarrow \\
 11010100
 \end{array}$$

Nestas condições:

$$\begin{array}{r}
 \text{Bit de sinal} \\
 \downarrow \\
 0\ 0001111 \\
 + 1\ 1010100 \\
 \hline
 \end{array}$$

$1\ 1100011 = -29$ na forma "complemento para dois"; isto é, trata-se do complemento para dois de +29.

Neste caso o bit de sinal indica um número negativo que se encontra na forma complemento para dois.

Exercício

Sejam A e B dois números binários na forma complemento para dois, tais que $A = 010111$ e $B = 010001$. Mostre que $A - B = 000110$ e $B - A = 111010$.

Como novo exemplo do método de subtração usando a complementação para dois iremos considerar a subtração da fração decimal 0,25 da fração decimal 0,875 usando palavras binárias de oito bits. Neste caso, devemos antes de mais converter ambas as frações para as suas representações binárias equivalentes. Isto é feito do seguinte modo:

$$0,875 = 0,111$$

Do mesmo modo, $0,25 = (0 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3})$ isto é, $0,25 = 0,010$

Usando as palavras binárias de oito bits para os números binários, com a vírgula ocorrendo depois do quarto bit, teremos

$$\begin{array}{cc}
 \text{bit de sinal} & \text{vírgula binária} \\
 \downarrow & \downarrow \\
 0,875 = 0000,1110
 \end{array}$$

$$\begin{array}{cc}
 \text{bit de sinal} & \text{vírgula binária} \\
 \downarrow & \downarrow \\
 0,25 = 0000,0100
 \end{array}$$

e portanto a forma complementar de -0,25 é

$$\begin{array}{cc}
 \text{bit de sinal} & \text{vírgula binária} \\
 \downarrow & \downarrow \\
 1111,1100
 \end{array}$$

$$\begin{array}{cc}
 \text{bit de sinal} & \text{vírgula binária} \\
 \downarrow & \downarrow \\
 0\ 000,1110 (= 0,875) \\
 + 1\ 111,1100 (= -0,25) \\
 \hline
 1\ 0\ 000,1010 (= 0,625) \\
 \uparrow \\
 \text{(ignorar excesso)}
 \end{array}$$

Exercícios

1. Usando palavras binárias de oito bits, com cinco para a parte fracionária, mostre que a representação em complemento para 2 de 0,53125 é 000,10001 e de -0,6875 é 111,01010.

2. Verifique que

$$\begin{array}{r} 000,10001 (= 0,53125) \\ + 111,01010 (= -0,6875) \\ \hline 111,11011 (= -0,15625) \end{array}$$

Multiplicação

Um método muito comum de realizar a multiplicação binária utiliza o princípio de deslocamento e soma de algarismos. O multiplicando é multiplicado por cada bit do multiplicador, bit a bit, e o produto resultante obtém-se somando todos os produtos parciais convenientemente deslocados. Como os bits do multiplicador são 0 ou 1, os termos do produto parcial são iguais a zero ou ao multiplicando, ou versões deslocadas deste.

Ilustremos este método considerando a multiplicação dos dois números decimais 193 e 21. Utilizamos um acumulador de 16 bits para guardar o resultado, e trabalharemos com representações binárias oito bits para o multiplicando e o multiplicador.

Multiplicando	11000001 = 193
Multiplicador	00010101 = .21
	<hr/>
	11000001
	11000001 Produtos parciais
	11000001 convenientemente deslocados
	<hr/>
	111111010101 Soma dos termos

Exercício

Usando o método anterior mostre que

$$1101 \times 10111 = 100101011$$

Quando se multiplicam dois números binários o produto contém um número de bits igual à soma dos bits contidos nos dois números binários. O número máximo de somas requerido para a multiplicação usando o método de deslocamento e soma é igual ao número de bits do multiplicador.

Divisão

A divisão binária é realizada usando um método de deslocamento e subtração. Diminui-se repetidamente o divisor do dividendo depois de este ser convenientemente deslocado, e inspeciona-se o sinal do resto após cada subtração. Se o sinal do resto é positivo o valor do quociente é 1, mas se o sinal é negativo o valor é zero, e o dividendo é reconduzido ao seu valor anterior somando de novo o divisor. Depois de a subtração dar um quociente positivo, ou depois do tratamento anterior no caso de ter dado um quociente negativo, o divisor é deslocado de uma posição para a direita, sendo incluído o bit seguinte e repetindo-se a operação até todos os bits do divisor terem sido usados.

Ilustraremos este método considerando a divisão de 90 por 9, usando uma vez mais as correspondentes representações binárias de oito bits.

	00001010
00001001	01011010
	- 00001001
	11110111
	+ 00001001
	00000001
	- 00001001
	11111000
	+ 00001001
	00000010
	- 00001001
	11111001
	+ 00001001
	000000101
	- 00001001
	11111100
	+ 00001001
	00001011
	- 00001001
	00000100
	- 00001001
	11111011
	+ 00001001
	00001001
	- 00001001
	00000000
	- 00001001
	1110111
	+ 00001001
	00000000

O resultado é negativo; portanto o valor do quociente é zero. Soma-se o divisor.

Deslocar divisor para a direita e subtrair. O resultado é negativo; o quociente é zero. Somar divisor.

Deslocar divisor para a direita e subtrair. O resultado é negativo; o quociente é zero. Somar divisor.

Deslocar divisor para a direita e subtrair. O resultado é negativo; o quociente é zero. Somar divisor.

Deslocar divisor para a direita e subtrair. O resultado é positivo; o quociente é 1.

Deslocar o divisor para a direita e subtrair. O resultado é negativo; o quociente é 0. Somar divisor.

Deslocar divisor para a direita e subtrair. Resultado positivo; quociente 1.

Deslocar divisor para a direita e subtrair. Resultado negativo; quociente 0. Somar divisor.

Resto.

Observações finais

Neste capítulo vimos a representação de números em forma decimal, binária e hexadecimal, e aprendemos a convertê-los entre si e a usar as funções PEEK e POKE. Foram também introduzidas as operações de soma, subtração; multiplicação e divisão binárias.

Exercício

Usando o método apresentado mostre que 01011/011 dá o quociente 0011 com resto 010.

CAPÍTULO III

TRATAMENTO DE NÚMEROS

Uma característica importante do ZX Spectrum é a sua capacidade de manipular números e de realizar operações matemáticas. Este capítulo examina o modo como o computador representa números em forma de vírgula flutuante e explica como são realizados os cálculos aritméticos. Descreve também as funções matemáticas e trigonométricas existentes na máquina e o modo de definir funções novas. O capítulo inclui igualmente uma explicação dos quadros de variáveis numéricas, mas deixaremos os quadros de sequências ou cadeias alfanuméricas para o capítulo 4.

O *software* (programas) de jogos vídeo pode incluir rotinas usando números aleatórios, pelo que descreveremos o modo como estes são produzidos.

Representação de números em vírgula flutuante

Quando se usa a linguagem Basic no ZX Spectrum, os números decimais são representados usando uma notação binária de vírgula flutuante. Isto permite manipular números decimais na gama $\pm 1,7 \times 10^{38}$ usando pelo menos oito algarismos decimais significativos.

Em notação de vírgula flutuante os números decimais são representados da seguinte forma:

$$\text{Número} = m \times 2^{\varepsilon}$$

onde ε é o expoente e m a mantissa. A gama desta é limitada a

$$1/2 \leq m < 1$$

e o expoente ε é um inteiro na gama

$$-127 \leq \varepsilon \leq +127$$

A mantissa é armazenada sob a forma de um número fracionário de quatro bits, e devido à sua gama limitada o bit mais significativo é sempre 1. Isto significa que o bit mais significativo da mantissa é redundante, sendo portanto possível usá-lo para indicar o sinal do número. A convenção adoptada para este bit de sinal consiste em usar 0 para os números positivos e 1 para os negativos.

O expoente é guardado usando apenas um byte. Para evitar complicações desnecessárias relativamente ao sinal o expoente é armazenado sob a forma de um número inteiro positivo na gama 1 a 255, conseguida somando o número decimal 128 ao valor real do expoente, ε .

Como exemplo, mostramos o modo como é armazenado o número decimal $-28,84375$ usando cinco bytes de memória. Consideremos então

$$\begin{aligned} -28,84375 &= -0,9013671875 \times 32 \\ &= -0,9013671875 \times 2^5 \\ &\quad \quad \quad m \quad \quad \quad \varepsilon \end{aligned}$$

A representação fraccionária binária da grandeza da mantissa, m , é 0,1110011011. No entanto, o bit mais significativo deste número é sempre 1, e como é redundante é substituído pelo bit de sinal. Neste exemplo, como a mantissa é negativa, o bit de sinal é 1, e usando quatro bytes para o sinal e para a grandeza da mantissa obtém-se:

$$\begin{array}{cccc} 11100110 & 11000000 & 00000000 & 00000000 \\ \uparrow & & & \\ \text{bit de sinal} & & & \end{array}$$

O valor do expoente é 5, acrescentando-se 128 para dar 133, valor armazenado sob a forma de um inteiro positivo de um só byte, 10000101.

O ZX Spectrum representa o zero colocando os quatro bytes da mantissa e o de expoente a zero.

Exercício

Mostre que o número decimal 140 é armazenado em notação de vírgula flutuante sob a forma

$$\begin{array}{ccccccc} & & \text{bit de sinal} & & & & \\ & & \downarrow & & & & \\ 1000010000 & 00001100 & 00000000 & 00000000 & 00000000 & & \\ \hline \varepsilon + 128 & & & m & & & \end{array}$$

Usando o formato de vírgula flutuante, o maior número que o ZX Spectrum pode tratar corresponde a todos os bits iguais a 1 no byte de expoente, representando $\varepsilon = +127$, e a todos os bits (32) dos quatro bytes da mantissa também a 1, representando $m = (1 - (1/2^{32}))$. O maior número possível é portanto

$$\begin{aligned} &= 2^{127} \times (1 - (1/2^{32})) \\ &\sim 2^{127}, \text{ porque } (1 - 1/2^{32}) \sim \\ &\sim 1,7014 \times 10^{38} \end{aligned}$$

Pode-se verificar que o ZX Spectrum pode tratar este número fazendo PRINT 1.7014E + 38, caso em que o computador responde apresentando o número sob a mesma forma no visor, o que indica que aceita. Se no entanto aumentarmos este valor, para por exemplo $1,7015 \times 10^{38}$, indicado sob a forma PRINT 1.7015E + 38, o computador não o aceitará porque está fora da gama válida.

O menor número positivo que pode ser representado no ZX Spectrum corresponde ao valor de expoente $\varepsilon = 1 - 127$, e ao valor de mantissa de 0,5. É portanto:

$$\begin{aligned} &= 2^{-127} \times 0,5 \\ &= 2^{-128} \\ &\approx 2,9388 \times 10^{-39} \end{aligned}$$

Pode-se confirmar que o ZX Spectrum pode tratar este número fazendo PRINT 2.9388E - 39, caso em que o computador apresenta o número no visor indicando a sua aceitação. Se no entanto tentarmos dar entrada a um valor inferior a este, por exemplo $2,92 \times 10^{-39}$, fazendo PRINT 2.92E - 39, o computador

não o aceitará, apresentando no entanto no visor o valor positivo mais pequeno que aceita, isto é, $2,92 \times 10^{-39}$. Dará sempre este resultado para valores compreendidos entre este e $1,49367939E - 39$, abaixo do qual imprimirá no visor o valor 0.

O ZX Spectrum utiliza sempre o mesmo formato de sinal e grandeza para qualquer número, tendo como consequência que as gamas de números negativos ou positivos aceites são idênticas.

Exercício

Use o programa seguinte para investigar a gama de números aceitáveis no ZX Spectrum.

```
5 PRINT "Indique número"
15 INPUT X
25 PRINT "O número é";X
35 PRINT
45 GO TO 5
```

Variáveis numéricas

Quando usamos o ZX Spectrum para obter a soma de três números, por exemplo 27,6, 8,7 e 9,1, podemos simplesmente ordenar ao computador que

```
PRINT 27.6 + 8.7 + 9.1
```

e a máquina imprimirá o valor 45,4.

Por outro lado, se quisermos programar o computador para somar quaisquer três números, devemos definir três símbolos que irão representar esses três números, usando-os depois num programa Basic. Por exemplo, apresentamos a seguir um programa muito simples que permite obter a soma de quaisquer três números representados por x, y e z.

```
10 INPUT x
20 INPUT y
30 INPUT z
40 PRINT x + y + z
```

Os símbolos x, y e z podem representar qualquer valor numérico, e consequentemente são chamados "variáveis numéricas". Note que as declarações INPUT usadas nas linhas 10, 20 e 30 pedem ao utilizador que dê entrada a três números (dados) através do teclado, os quais serão atribuídos a cada uma das três variáveis.

Uma variável numérica pode ser representada usando caracteres alfabéticos (maiúsculas e minúsculas), caracteres numéricos e espaços, mas o primeiro carácter deve ser sempre alfabético. Não se podem usar caracteres alfabéticos em maiúsculas e minúsculas para representar duas variáveis numéricas diferentes. O ZX Spectrum tratá-los-á como uma mesma variável numérica. Vejamos alguns exemplos de variáveis permitidas:

```
Jaime
salmão ou Salmão
u12 ou U12
O dividendo final é
```

No entanto, as seguintes variáveis numéricas não serão aceites:

```
4 Venda — porque se inicia por um carácter numérico.
U12? — porque utiliza um carácter não válido (?)
```

A declaração LET pode ser usada num programa para atribuir um valor constante a uma variável; por exemplo

```
LET Juro = 6
```

Pode também ser usada para atribuir o valor do segundo membro de uma expressão matemática, envolvendo variáveis numéricas previamente definidas, a uma nova variável. Exemplos típicos:

```
LET x = 2 * Y + B
LET SOMA 1 = - (4.9/t + A)
```

Um outro método de atribuir valores constantes a variáveis no interior de um programa consiste em usar a declaração READ e a correspondente declaração DATA. Podemos ilustrar isto executando o seguinte programa.

```

15 DATA 6,-7,2.9,149.21
25 READ x,y
35 PRINT x+y
45 READ p,q
55 PRINT p * q

```

Variável em atribuição :

Dados:

x	y	p	q
6	-7	2.9	149.21

Linha de Programa: 15

(a)

Variável em atribuição :

Dados:

x	p	q	Não usado
6	-7	2.9	149.21

Linha de programa:
(ponto de referência)

15 32 65

(b)

Figura 3.1 — Formato de dados: (a), único ponto de referência; (b), três pontos de referência

Note que a linha 15 define uma lista de dados, apresentada na figura 3.1 (a), que é usada pelas instruções READ. A linha 25 atribui os dois primeiros valores dos dados às variáveis x e y e, depois de estes serem usados, a linha 45 atribui os dois seguintes às variáveis p e q. As linhas 35 e 55 são usadas para formar a soma $x + y$ e o produto $p \times q$ respectivamente.

Os valores de dados das instruções DATA são apresentados sob a forma de uma simples lista de dados, sendo acedidos sequencialmente por instruções READ. No programa anterior, os dados podem ser definidos usando mais do que uma instrução DATA; por exemplo

```

15 DATA 6
25 READ x,y
32 DATA -7,2.9
35 PRINT x+y
45 READ p,q
55 PRINT p * q
65 DATA 149.21

```

O leitor pode verificar que este programa dá os mesmos resultados numéricos do programa anterior, -1 no caso da soma $x+y$ e $432,709$ no caso do produto $p * q$. O interesse da divisão dos dados por diversas instruções DATA reside em permitir às instruções READ acederem dados a partir de pontos de referência especificados na lista de dados, como se mostra na figura 3.1 (b). Um ponto de referência é definido pela declaração RESTORE, seguida do número de linha de programa onde se encontra a instrução DATA onde se pretende ler os dados. Pode-se examinar este aspecto inserindo

40 RESTORE 32

no programa anterior. Neste caso verificará que a soma é ainda -1 , mas que o produto passa a $-20,3$ porque a instrução RESTORE obriga o indicador de referência dos dados a apontar para a linha 32 do programa. De facto seria possível obter os mesmos resultados indicando na instrução RESTORE qualquer linha entre 16 e 32, porque este valor é sempre interpretado pelo Spectrum como uma instrução para aceder aos dados da instrução DATA que se encontra a seguir ou na linha indicada por RESTORE. Para iniciar a primeira instrução DATA pode-se usar uma instrução RESTORE sem acrescentar qualquer número de linha.

Exercícios

1. Alterar a linha 40 do programa anterior para

40 RESTORE 18

e verificar que os resultados são agora -1 e $-20,3$.

2. Alterar a linha 40 do programa anterior para

0 RESTORE

e verificar que os resultados são agora -1 e -42 .

Quando se grava um programa em cassette os últimos dados atribuídos a variáveis numéricas são também gravados, passando a constituir os dados iniciais para a execução do programa quando este é novamente carregado. A declaração CLEAR limpa no en-

tanto todas as variáveis definidas, libertando a memória que foi atribuída ao armazenamento das variáveis.

Simple cálculos aritméticos

O ZX Spectrum pode realizar cinco operações aritméticas, a soma (+), a subtração (-), a multiplicação (*), a divisão (/) e a exponenciação (e elevar a uma potência) (\uparrow). É importante notar que o computador não pode realizar a exponenciação de um número negativo. Quando são usadas várias operações aritméticas num mesmo cálculo o computador começa por calcular as exponenciações, depois as multiplicações e as divisões e só em seguida as somas e subtrações. No entanto, se os cálculos forem colocados entre parêntesis serão executados antes de quaisquer outros.

O melhor modo de compreender como são realizados os cálculos no ZX Spectrum consiste em estudar alguns exemplos. Experimente os seguintes:

1. Cálculo pretendido: $18,6 \times 3,7 - 2^{1,6}$
Instrução ZX Spectrum: PRINT 18.6 * 3.7 - 2 \uparrow 1.6
Resposta: 65.788567
2. Cálculo pretendido: $18,6 \times (3,7 - 2^{1,6})$
Instrução ZX Spectrum: PRINT 18.6 * (3.7 - 2 \uparrow 1.6)
Resposta: 12.435344
3. Cálculo pretendido: $-0,06/2,417 + 2,3E4$
Instrução ZX Spectrum: PRINT -0.06/2.417 + 2.3E4
Resposta: 22999.975
4. Cálculo pretendido: $49^{7,02}$
Instrução ZX Spectrum: LET A = 49 \uparrow - 7.02: PRINT A
Resposta: 1.3640287E-12
5. Cálculo pretendido: $-10,66^{(4,2 + 1,915)}$
Instrução ZX Spectrum: LET Resposta = -10.66 \uparrow (4.2 + 1.915);
PRINT Resposta
Resposta: -1.8460829E+8

Note que as respostas apresentadas para os exemplos 4 e 5 são-no em notação científica, onde E-12 representa 10^{-12} e E+8 representa 10^8 , esta forma de notação é um método simples de indicar números muito grandes ou muito pequenos, sendo por vezes designada por representação numérica de vírgula flutuante.

Pode-se ver pelos exemplos 4 e 5 que é possível incluir variáveis numéricas em expressões aritméticas, e que o computador usará os respectivos valores numéricos no cálculo. Se quisermos escrever um programa para calcular a média de quaisquer três números comunicados à máquina através do teclado, poderemos usar as variáveis numéricas a, b e c, para representar os números, e calcular a média usando o programa listado em seguida. Experimente você mesmo.

```
15 INPUT a: PRINT a
25 INPUT b: PRINT b
35 INPUT c: PRINT c
45 PRINT
55 PRINT "Média = ";(a+b+c)/3
```

Como segundo exemplo, vejamos um programa que calcula o quadrado e a raiz quadrada de um número indicado ao computador através do teclado.

```
15 INPUT A: PRINT A
20 PRINT
25 PRINT A  $\uparrow$  2, A  $\uparrow$  .5
30 PRINT
35 PRINT A  $\uparrow$  2: A  $\uparrow$  .5
40 PRINT
45 PRINT A  $\uparrow$  2: " " : A  $\uparrow$  .5
50 PRINT
55 PRINT A  $\uparrow$  2'A  $\uparrow$  .5
```

Sugerimos que experimente este programa e note os quatro diferentes formatos de resposta criados pelas quatro diferentes instruções PRINT das linhas 25, 35, 45 e 55.

Exercício

Utilize o computador como máquina de calcular, demonstrando que:

- 1) $17,2^2 - 94,6 = 201,24$
- 2) $(8,3/7,2 + 19,6)^{1/2} = 4,5555217$
- 3) $1/2 + 7/16 - 8,2^{0,6} = -2,5966771$

Funções matemáticas

Quando se utilizam funções matemáticas deve-se notar que, na ausência de parêntesis, estas expressões são avaliadas ao realizar cálculos segundo uma expressão envolvendo apenas as operações aritméticas simples: +, -, * e /. A regra segundo a qual as expressões encerradas em parêntesis são avaliadas primeiro aplica-se mesmo quando a expressão total contém uma ou mais funções matemáticas.

ABS

A função ABS é usada quando se pretende obter o valor Absoluto de um número ou de uma expressão matemática em avaliação.

A expressão "valor absoluto" refere-se apenas à grandeza do número ou da expressão avaliada, ignorando o sinal. Use o seu Spectrum para obter os resultados do seguinte:

```
15 PRINT ABS 118.97,ABS-118.97
25 PRINT ABS 2.9/1.12 * 2.81, ABS2.9/-1.12 * 2.81
```

Notou que o quarto resultado é prefixado pelo sinal -? A razão disto é que, como a quarta expressão não contém quaisquer parêntesis, o ZX Spectrum determina primeiramente o valor absoluto de 2,9, divide em seguida este valor por -1,12 e termina multiplicando o resultado por 2,81.

SGN

A função SGN é usada para determinar se um número, ou uma expressão matemática depois de calculada, é nulo, positivo ou negativo. O resultado obtido quando se usa esta função é 0, +1 ou -1, correspondendo respectivamente aos resultados nulo, positivo e negativo. Uma aplicação típica desta função é quando se deseja obrigar o computador a aceitar apenas números positivos superiores a zero. Experimente o seguinte:

```
5 PRINT "Indique número positivo > 0"
15 INPUT N
25 LET A = SGN N
35 IF A = 0 THEN GOTO 5
45 IF A = - 1 THEN GOTO 5
55 PRINT N; "Indique número seguinte"
65 GO TO 15
```

Este programa aceita um número, e se este é positivo mostra-o no visor e pede um novo número. Um número negativo ou zero não serão aceites, sendo impressa uma mensagem pedindo um número positivo.

INT

A função INT é usada para arredondar para o inteiro imediatamente inferior qualquer número. Um número positivo diminuirá portanto em grandeza ao ser arredondado, a não ser que já seja um número inteiro, enquanto que um número negativo aumentará de grandeza a menos que já seja um inteiro. Por exemplo, +17,632 é reduzido para +17, mas -17,632 é aumentado para -18. Vejamos um programa simples que permite verificar os resultados desta função.

```
5 PRINT "Indique um número"
15 INPUT A
25 LET B = INT A
35 PRINT "O valor inteiro do número é: ";B
45 GO TO 5
```

SQR

Para obter a raiz quadrada de um número pode-se usar a função SQR. Experimente o programa seguinte, que lhe permite determinar a raiz quadrada de um número positivo. Note que usamos a função SGN para assegurar a aceitação de números positivos.

É necessário usar apenas números positivos porque o ZX Spectrum não pode calcular a raiz quadrada de um número negativo.


```

5 PRINT "Indique número positivo > 0":PRINT
15 INPUT n: PRINT n: PRINT
25 LET b = SGN n
35 IF b = 0 THEN GO TO 5
45 IF b = 1 THEN GO TO 5
55 LET c = SQR n
65 PRINT "A raiz quadrada de ";n;" é"
68 PRINT : PRINT c
75 PRINT : GO TO 5

```

LN

Para determinar o logaritmo natural de um número pode-se usar a função LN. O logaritmo natural de um número, ou \log_e , como por vezes é designado, está relacionado com o logaritmo na base 10 (\log_{10}) pela expressão

$$\log_{10} x = \log_e x / 2,302585093$$

Podemos portanto usar esta expressão para obter o logaritmo decimal de um número. O programa listado em seguida utiliza a função LN e a relação citada para determinar o logaritmo decimal de um número positivo.

```

5 PRINT "Indique número positivo > 0":PRINT
15 INPUT n: PRINT n: PRINT
25 LET b = SGN n
35 IF b = 0 THEN GO TO 5
45 IF b = - 1 THEN GO TO 5
55 LET x = LN n/2.302585093
65 PRINT "O log 10 de ";n;" é"
68 PRINT : PRINT x
75 PRINT : GO TO 5

```

Podemos, evidentemente, determinar o antilog₁₀ de um número x recorrendo à relação

$$y = \text{antilog}_{10} x = 10^x$$

que pode ser obtida em Basic usando

```
LET y = 10 ↑ x
```

EXP

Podemos determinar a função exponencial, e^x , de um número x recorrendo à função EXP. Esta relação pode ser usada, por exemplo, para avaliar o crescimento ou decréscimo exponenciais. O programa seguinte ilustra o uso desta função para determinar o crescimento exponencial da expressão matemática Ke^{2t} , para $t = 0, 1, 2, \dots, 15$.

```

3 INPUT K: PRINT K: PRINT
5 FOR t = 0 TO 15
15 LET x = K * EXP (2 * t)
25 PRINT x
35 NEXT t

```

Sugerimos-lhe que execute o programa e altere depois a linha 15 para

```
15 LET x = K * EXP (- 2 * t)
```

Execute novamente o programa e observe o correspondente decréscimo exponencial.

Funções trigonométricas

Quando se usam funções trigonométricas deve-se notar que, na ausência de parêntesis, estas funções são avaliadas em expressões que envolvem apenas operações aritméticas simples: +, -, * e /. A regra segundo a qual as expressões encerradas em parêntesis são avaliadas primeiro aplica-se mesmo quando a expressão total contém uma ou mais funções matemáticas.

SIN, COS e TAN

O ZX Spectrum pode avaliar o seno, o co-seno e a tangente de um ângulo usando respectivamente as funções SIN, COS e TAN. O ângulo deve ser expresso em radianos, pelo que se se quiser exprimi-lo em graus é necessário usar a expressão

$$\text{ângulo (em radianos)} = \text{ângulo (em graus)} * \pi/180$$

O programa seguinte calcula o seno, o co-seno e a tangente de um ângulo indicado à máquina em graus, no qual a constante matemática pi é igual a 3,14155927 e existe no teclado do ZX Spectrum.

```
15 PRINT "Indique ângulo em graus"
25 INPUT a: PRINT a: PRINT
35 LET x=a * PI/180
45 PRINT "Sen a=", SIN x,"Cos a=", COS x, "Tan a=",
  TAN x
55 PRINT: GO TO 15
```

Experimente este programa para diferentes ângulos positivos e negativos. Notará que não consegue obter a tangente de 90 ou de 270 graus porque, como se sabe, este valor é infinito e encontra-se portanto fora da gama de números tratada pelo computador.

Exercício

Usando as funções SIN, COS e TAN escreva instruções Basic que demonstrem que

- (1) $\tan a = \frac{\text{sen } A}{\text{cos } A}$
- (2) $\text{sen}^2 a + \text{cos}^2 a = 1$
- (3) $\text{sen } 2x = 2\text{sen}x \text{ cos}x$
- (4) $\text{cos } 2x = 2\text{cos}^2 x - \text{sen}^2 x$

ASN, ACS e ATN

Quando se conhece o seno de um ângulo é possível determinar o ângulo correspondente recorrendo à função "arco cujo seno", ASN. Do mesmo modo, é possível determinar um ângulo a partir do seu cosseno, recorrendo à função "arco cujo cosseno", ACS, ou a partir da sua tangente, recorrendo a ATN.

Estas três funções dão o ângulo resultante em radianos, sendo portanto necessário usar a relação

$$\text{ângulo (em graus)} = \text{ângulo (em radianos)} * 180/\pi$$

para converter esse valor em graus.

O programa seguinte calcula o ângulo em graus correspondente a um valor de seno.

```
15 PRINT "Indique Valor do Seno": PRINT
25 INPUT s: PRINT s: PRINT
35 LET a = ASN s
45 LET b = a * 180/PI
55 PRINT "O ângulo é ";b;" graus"
65 PRINT : GO TO 15
```

Quando se executa este programa para valores positivos e negativos do arco no domínio válido +1 a -1, nota-se que o programa apresenta como resposta um ângulo que varia na gama +90 a -90 graus.

Se alterar a linha 15 para

```
15 PRINT "Indique Valor do Co-seno": PRINT
```

e a linha 35 para

```
LET a = ACS s
```

o mesmo programa poderá ser usado para calcular o ângulo em graus correspondente a um dado valor de co-seno. O programa apresentará então uma resposta na gama 0 a 180 graus, correspondente aos valores válidos do arco, variando entre +1 e -1.

Se alterar a linha 15 para

```
15 PRINT "Indique Valor da Tangente": PRINT
```

e a linha 35 para

```
35 LET a = ATN s
```

pode usar o mesmo programa para obter o ângulo em graus correspondente a uma dada tangente trigonométrica. Neste caso verificará que o programa devolve ângulos na gama +90 a -90 graus. Experimente esta última variante na gama +90 000 a -90 000.

Funções matemáticas definidas pelo utilizador

É possível definir as nossas próprias funções para uso no

interior de um programa Basic. Chama-se-lhes *funções definidas pelo utilizador*, e o formato da instrução necessária para tal é: número de linha DEF FN letra (argumento) = função matemática

Por exemplo:

```
35 DEF FN A(r) = PI * r ↑ 2
```

define a função do utilizador A(r) como sendo igual a πr^2 (a área do círculo com um raio r). Depois de definir a função é possível usá-la referenciando-a pela expressão FN A(r), e o computador determinará o valor da expressão para qualquer valor do argumento. Talvez o leitor esteja interessado em verificar por si mesmo que a ordem PRINT FN A(2) avaliará a expressão e apresentará o resultado 12,566371, correspondente à área do círculo com um raio (argumento) igual a 2.

Exercício

Execute o seguinte programa para diversos valores de raio.

```
5 PRINT "Indique valor do raio"
15 INPUT r: PRINT r
25 PRINT
35 DEF FN A(r) = PI * r ↑ 2
45 DEF FN F(r) = 4/3 * PI * r ↑ 3
55 PRINT "A área do círculo com um raio ";r;" é ";FN A(r)
70 PRINT
75 PRINT "O volume da esfera com um raio ";r;" é ";FN V(r)
```

Se não sabe porque razão o computador responde com a mensagem de erro A Invalid Argument, 55:1 quando indica um valor negativo para o raio, recorde o que já foi dito sobre a exponenciação de números negativos.

Um outro exemplo em que pode ser conveniente empregar funções definidas pelo utilizador é na avaliação de funções hiperbólicas: seno, co-seno e tangente hiperbólicos de x.

Usando a série que representa o seno hiperbólico de x:

$$\sinh x \sim x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \dots$$

podemos definir a função em Basic do seguinte modo

```
DEF FN s(x) = x + x ↑ 3/6 + x ↑ 5/120 + x ↑ 7/5040
```

O desenvolvimento em série que nos dá $\cosh x$ é

$$\cosh x \sim 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots$$

o que pode ser definido em Basic do seguinte modo:

```
DEF FN c(x) = 1 + x ↑ 2/2 + x ↑ 4/24 + x ↑ 6/720
```

Podemos definir agora $\tanh x$ como

```
DEF FN t(x) = FN s(x)/FN c(x)
```

O programa listado a seguir pode ser usado para avaliar estas funções hiperbólicas para qualquer valor positivo de xx. Experimente executar este programa e compare as respostas com os valores listados em tabelas matemáticas, ou com os resultados obtidos numa máquina de calcular.

```
35 DEF FN s(x) = x + x ↑ 3/6 + x ↑ 5/120 + x ↑ 7/5040
40 DEF FN c(x) = 1 + x ↑ 2/2 + x ↑ 4/24 + x ↑ 6/720
50 DEF FN t(x) = FN s(x)/FN c(x)
130 PRINT "Indique valor de x"
135 INPUT x
140 PRINT "For x = ";x;" sinh x = "; FN s(x) " cosh x = ";FN
c(x) " tanh x = ";FN t(x)
150 PRINT
155 PRINT "Nova execução (Sim/Não)?"
165 INPUT k$
175 IF K$ = "S" THEN GO TO 130
185 IF k$ = "N" THEN STOP
```

Note que se for necessário maior rigor nos valores calculados das funções hiperbólicas, necessitará de usar mais termos dos desenvolvimentos em série de seno hiperbólico e co-seno hiperbólico. Por exemplo, os termos seguintes de ambas as séries são $x^9/9!$ e $x^8/8!$, devendo ser acrescentados para aumentar o desenvolvimento em série das funções seno hiperbólico e co-seno hiperbólico respectivamente de mais um termo.

Exercício

Investigue o efeito de acrescentar $+x \uparrow 9/362880$ no final da linha 35 no programa anterior; e de $+x \uparrow 8/40320$ no final da linha 40 do mesmo programa.

Alternativamente, e com maior rigor, as definições matemáticas de seno hiperbólico e co-seno hiperbólico de x são:

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

e

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

e ambas estas expressões podem ser definidas no programa anterior alterando as linhas 35 e 40 do seguinte modo:

```
35 DEF FN s(x) = (EXP x - EXP - x)/2
```

```
40 DEF FN c(x) = (EXP x + EXP - x)/2
```

Exercício

Para $x = 0,8$ e $x = 3,1$ use:

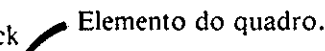
- (1) O programa anterior de desenvolvimento em série com
 - (a) os quatro termos originais;
 - (b) cinco termos (ver exercício anterior);
- (2) as definições alternativas dadas atrás;

para determinar o seno hiperbólico, o co-seno hiperbólico e a tangente hiperbólica de x . Compare os resultados com os valores listados em tabelas trigonométricas, ou determinados usando uma máquina de calcular.

Quadros de variável numérica

Se diversas variáveis possuírem uma característica comum, por exemplo número de elementos em «stock», é conveniente designar as variáveis usando uma representação simbólica dessa ca-

racterística comum e distinguir as variáveis por um número de identificação incluído no próprio nome da variável. Se por exemplo o leitor vender revistas, pode querer registrar o seu «stock» de seis publicações, por exemplo (a) *Your Computer*, (b) *Reader's Digest*, (c) *Popular Science*, (d) *Trout and Salmon*, (e) *Stamp Monthly* e (f) *She*. Necessitará então de seis variáveis para designar as revistas e, como a característica comum é o número de exemplares de cada revista, podemos indicá-lo por $n(1)$, $n(2)$, $n(3)$, $n(4)$, $n(5)$ e $n(6)$, como se mostra na figura 3.2.

Quadro de espécies em stock  Elemento do quadro.

$n(1)$ Your Computer	$n(2)$ Reader's Digest	$n(3)$ Popular Science	$n(4)$ Trout and Salmon	$n(5)$ Stamp Monthly	$n(6)$ She
----------------------------	------------------------------	------------------------------	-------------------------------	----------------------------	---------------

Figura 3.2 — Quadro de uma dimensão com seis elementos

O arranjo apresentado na figura 3.2 é conhecido por quadro («array») de uma dimensão, contendo neste caso seis elementos. Para usar este quadro num programa Basic é necessário reservar primeiramente as posições de memória necessárias, recorrendo à instrução DIM. Neste exemplo, como existem seis elementos no quadro de uma dimensão, a instrução de dimensionamento necessária é DIM $n(6)$. É importante notar que o número que distingue as variáveis se encontra no interior de parêntesis. Isto significa que $n(1)$, $n(2)$, $n(3)$, $n(4)$, $n(5)$ e $n(6)$ são as únicas variáveis numéricas válidas neste quadro. Só se podem usar índices (valores dentro de parêntesis) num nome de variável quando se estão a usar quadros.

Não somos obrigados a usar apenas quadros de uma dimensão. Por exemplo, é possível ampliar o quadro de uma só dimensão acima considerado dando-lhe uma forma bi-dimensional, de modo a permitir o seu uso pelo importador das revistas. Na figura 3.3 mostramos um quadro de duas dimensões representando as mesmas revistas e a quantidade de cada uma delas necessária para abastecer quatro lojas diferentes. Vemos que este quadro contém 24 elementos dispostos em quatro linhas e seis colunas. Cada elemento é identificado por uma variável usando o formato:

n(número de linha, número de coluna)

	Your Computer	Reader's Digest	Popular Science	Trout and Salmon	Stamp Monthly	She
N.º1	n(1,1)	n(1,2)	n(1,3)	n(1,4)	n(1,5)	n(1,6)
N.º2	n(2,1)	n(2,2)	n(2,3)	n(2,4)	n(2,5)	n(2,6)
N.º3	n(3,1)	n(3,2)	n(3,3)	n(3,4)	n(3,5)	n(3,6)
N.º4	n(4,1)	n(4,2)	n(4,3)	n(4,4)	n(4,5)	n(4,6)

Figura 3.3 — Quadro bi-dimensional com quatro linhas e seis colunas

Por exemplo, n(2,4) refere-se ao número de *Trout and Salmon* encomendados pelo agente n.º 2. Para utilizar este quadro num programa Basic devemos evidentemente reservar as necessárias posições de memória no início do programa usando

DIM n(4,6)

O programa listado em seguida aceita dados para o quadro bi-dimensional de 24 elementos já citado, examina os dados e apresenta na saída a quantidade total de cada publicação existente no stock de representante.

```

5 DIM n(4,6)
10 PRINT "Pode escolher:" "1. Alterar elementos" "2.
  Imprimir elementos" "3. Imprimir total por coluna"
20 PRINT: PRINT "Indique 1, 2 ou 3"
25 INPUT N
35 IF N = 1 THEN GO TO 300
45 IF N = 2 THEN GO TO 300
55 IF N = 3 THEN GO TO 500
65 CLS
75 GO TO 10
300 PRINT "Indique n.º de linha": INPUT R: PRINT R
302 IF R > 4 THEN GO TO 300

```

```

305 PRINT "Indique n.º de coluna": INPUT C: PRINT C
307 IF C > 6 THEN GO TO 305
310 IF N = 2 THEN GO TO 320
312 PRINT "Indique dados para elemento escolhido": INPUT D:
  PRINT "Dado:"D
315 LET n(R,C) = D
317 PRINT : GO TO 10
320 PRINT "Dado no elemento escolhido:"n(R,C)
325 PRINT : GO TO 10
500 PRINT "Pode escolher total para:" "1. Your Computer"
  "2. Reader's Digest" "3. Popular Science" "4. Trout
  and Salmon" "5. Stamp Monthly" "6. She"
525 PRINT "Indique 1, 2, 3, 4, 5 ou 6": INPUT T
530 LET Total = 0
555 FOR R = 1 TO 4
560 LET Total = n(R,T) + Total
575 NEXT R
580 PRINT "Total da coluna:"T:"="Total
590 GO TO 20

```

Sugerimos-lhe que carregue este programa na máquina e investigue a sua execução.

Teoricamente não existe qualquer restrição quanto ao tamanho do quadro nem ao número de dimensões deste, mas na prática o leitor verificará que a capacidade de armazenamento de dados do ZX Spectrum é um factor limitativo. Note que o número de posições de memória que devem ser reservadas para um quadro é o *produto* do número total de elementos pelo número de dimensões. Por exemplo:

DIM R(8, 10, 6)

requer 480 posições de memória para poder armazenar todos os dados do quadro.

Números aleatórios

Pode-se usar a palavra-chave RANDOMIZE (abreviada para RAND no teclado) ou a função RND para produzir números com um certo grau de aleatoriedade, se bem que não satisfaçam de

facto uma análise estatística rigorosa em termos de casualidade.

A função RND produz uma sequência definida de 65 536 números na gama zero a 0,99998474, e a instrução RANDOMIZE que lhe está associada:

```
RANDOMIZE X
```

onde X é um inteiro positivo na gama 1 a 65535, define o ponto de partida da sequência de números produzidos pela função RND. No entanto, para obter uma casualidade dos resultados convém levar a função RND a começar por um qualquer número da sequência por ela produzida; isto consegue-se omitindo o número x, de partida da sequência de números produzidos pela função RND. No entanto, para obter uma casualidade dos resultados convém levar a função RND a começar porse deseja que o programa siga a mesma sequência de números aleatórios de cada vez que é executado. Por exemplo, usando a instrução

```
RANDOMIZE 45438
```

obteremos uma sequência definida partindo do número 0. Se usarmos o programa de duas linhas

```
10 RANDOMIZE 45438
20 PRINT RND: GO TO 20
```

os primeiros sete números da sequência produzida serão

```
0
0,0011291504
0,08581543
0,43719482
0,79025269
0,2691803
0,18934631
```

Se omitirmos o número depois da instrução RANDOMIZE obteremos uma sequência diferente de cada vez que o programa for executado, e a sequência parecerá de facto aleatória.

O programa referido em seguida pode ser usado para produzir números aleatórios positivos. Este programa exige que o utilizador indique a grandeza máxima do número a obter. Depois de o ter feito, o programa produz e imprime uma sequência de números.

```
15 DIM a$(12)
25 PRINT "Indique valor máximo desejado"
35 INPUT n: PRINT "Valor máximo = ";n: PRINT
65 PRINT "Carregue em n para obter o número"
70 IF INKEY$ <> "n" THEN GO TO 70
72 LET x = INT (RND*n) + 1
74 PRINT AT 8,14;a$
75 PRINT AT 8,0;"Número seguinte =";x
78 IF INKEY$ <> " " THEN GO TO 78
85 GO TO 70
```

Se fizer n = 6 obterá o equivalente a um dado; e se pensar um pouco pode até fazer previsões para o Totobola (espero que tenha sorte...)!

Observações finais

Estudando as informações dadas neste capítulo verificará que o ZX Spectrum é uma potente máquina para tratamento de números.

No capítulo seguinte descreveremos o modo de utilizar ca-deias neste computador.

CAPÍTULO IV

CADEIAS

Introdução

Quando desejamos incluir texto nos nossos programas podemos usar a instrução PRINT seguida do texto inserido dentro de aspas. O texto é uma *cadeia (string)* de caracteres. Por exemplo, "Bom Dia" é uma cadeia de sete caracteres, que pode ser impressa no visor usando

```
PRINT "Bom Dia"
```

Numa cadeia podemos usar todos os caracteres existentes no teclado excepto as próprias aspas, porque estas serão interpretadas como final da cadeia. Se no entanto quiser mesmo incluir aspas numa cadeia, pode utilizar a aspa simples (tecla "symbol shift" e tecla 7) ou repetir as aspas normais o número de vezes necessário. Por exemplo:

```
PRINT "Bom Dia 'Caro Amigo'"
```

e

```
PRINT "Bom Dia ""Caro Amigo"""
```

imprimirão no visor as frases

```
Bom Dia 'Caro Amigo'
```

e

```
Bom Dia ""Caro Amigo""
```

Se quiser que qualquer destas cadeias seja impressa noutro ponto do visor use a tecla PRINT seguida pela pseudo-função AT, segundo o formato:

PRINT AT número de linha, número de coluna; "caracteres"
onde as linhas são numeradas entre 0 e 21 a começar do topo, e as colunas 0 e 31 a começar da esquerda. Por exemplo

```
25 PRINT AT 21,1; "Bom Dia 'Caro Amigo'"
```

imprimirá esta cadeia na última linha de uso normal do visor (acima da janela de comunicação com a máquina). Experimente alterar o número da coluna para 6 para ver o que acontece.

Quando uma cadeia não contém quaisquer caracteres entre as aspas o computador sabe que se trata de uma cadeia vazia.

As cadeias podem ser acrescentadas entre si usando a operação +. Por exemplo, experimente o seguinte

```
PRINT AT 10,4; "O meu nome é" + "Guilherme"
```

Note que só pode usar a operação '+' para tratamento de cadeias.

Variáveis de cadeia

Pode-se atribuir uma cadeia de caracteres a uma variável, e nesses casos o nome da variável será indicado por uma única letra seguida do símbolo \$. No entanto, note que a máquina não distingue entre a\$ e A\$, pelo que só pode utilizar um máximo de 26 variáveis de cadeia no seu programa de Basic. O seguinte programa simples utiliza cinco variáveis de cadeia: A\$, B\$, C\$, D\$ e K\$.

```
15 LET A$ = "Parabéns"  
25 LET B$ = " "  
35 LET C$ = "PAI"  
45 LET D$ = "MÃE"  
55 LET K$ = A$ + B$ + C$  
65 PRINT AT 11, 8; K$
```

Exercício

Verifique que o programa anterior imprime Parabéns PAI.

Altere a variável C\$ para D\$ na linha 55 e execute novamente o programa.

Neste simples programa as variáveis de cadeia são definidas usando instruções LET. No entanto, é possível dar entrada a variáveis de cadeia usando instruções INPUT, que podem assumir uma das duas formas seguintes:

```
INPUT carácter$
```

ou

```
INPUT LINE carácter$
```

Se se usar uma instrução INPUT sem LINE, o sistema operativo do computador imprime automaticamente aspas de cada um dos lados do cursor "L", pedindo portanto ao utilizador que dê entrada à cadeia pretendida. Pelo contrário, quando se inclui LINE na instrução INPUT a máquina apresenta simplesmente o cursor (como se estivesse a pedir ao utilizador uma variável numérica).

Exercício

Altere a linha 15 do programa anterior para 15 INPUT A\$ e a linha 35 para 35 INPUT LINE C\$. Execute o programa e escreva quando lhe for pedido as cadeias apropriadas.

Subcadeias

Uma subcadeia é qualquer conjunto de caracteres consecutivos extraídos de uma cadeia já existente. Para definir uma subcadeia podemos usar a cadeia ou a variável de cadeia juntamente com a declaração TO, sob a forma:

```
Cadeia  
ou  
Variável de cadeia } (Princípio TO fim)
```

Nestas condições, se a cadeia existente for "Manual do Utilizador Spectrum", podemos defini-la como cadeia A\$:

```
15 LET A$ = "Manual do Utilizador Spectrum"
```

Podemos em seguida imprimir apenas "Utiliza" recorrendo a

```
25 PRINT AS (11 TO 17)
```

Alternativamente, para imprimir esta mesma subcadeia na linha 5 e na coluna 13, usariamos

```
25 PRINT AT 5,13:AS (11 TO 17)
```

Para imprimir a subcadeia "M" podemos usar

```
25 PRINT AS (1 TO 1)
```

Se a posição de início da cadeia for omitida ao definir a subcadeia, a máquina considerada que desejamos começar pelo primeiro caracter da cadeia original.

Assim, para imprimir "Manual" podemos escrever

```
25 PRINT AS (TO 6)
```

a máquina considera que desejamos terminar no último caracter da cadeia original. Por exemplo, para escrever "Spectrum" podemos usar:

```
25 PRINT AS (25 TO)
```

Quando a posição do primeiro caracter da subcadeia é maior do que a do último caracter da cadeia original é criada uma cadeia vazia.

Note que a última posição da subcadeia não deve exceder a posição do último caracter da cadeia original. Por outro lado, não se podem usar números negativos para definir as posições inicial e final de uma subcadeia.

Exercício

Se a cadeia x\$ = "Ponte de Londres", verifique que a subcadeia "on" pode ser impressa usando

```
PRINT x$ (2 TO 3)
```

ou

```
PRINT x$ (11 TO 12)
```

e que a subcadeia "dres" pode ser impressa usando

```
PRINT x$ (13 TO)
```

Funções de cadeia

No capítulo 3 vimos que o ZX Spectrum pode avaliar funções matemáticas e trigonométricas próprias ou definidas pelo utilizador. Discutiremos agora as quatro funções de cadeia LEN, VAL, VALS e STR\$, além das funções de cadeia definidas pelo utilizador.

LEN

Pode-se determinar o comprimento (LENGth) de uma cadeia usando a função LEN. Por exemplo, a cadeia "Tom and Jerry" possui 13 caracteres, o que pode ser confirmado executando

```
PRINT LEN "Tom and Jerry"
```

O computador imprimirá "13" no visor. Do mesmo modo, se dermos à máquina a ordem

```
PRINT LEN ("Tom" + "and" + "Jerry")
```

obteremos a resposta "11" (não incluímos os espaços entre as palavras).

VAL

Esta função é usada para avaliar cadeias que são expressões aritméticas. Por exemplo:

```
PRINT VAL "6 ↑ 6 ↑ 6"
```

avalia $((6)^6)^6 = 6^{36} = 1,0314425E + 28$.

Pode-se igualmente usar a função VAL com instruções gráficas; por exemplo

```
PRINT AT VAL "3 ↑ 2 + 2", VAL "4 ↑ 2";"©"
```

apresenta © no centro do visor.

Exercício

Mostrar que PRINT LEN ("Hoje é" + "Segunda-feira") tem o mesmo valor que PRINT VAL "3 * 5", ou seja 15.

VALS

Esta função avalia uma cadeia contida no interior de três grupos de aspas, e dá como resultado a mesma cadeia contida apenas num conjunto de aspas. Um dos seus usos consistirá portanto em avaliar subcadeias de cadeias contendo aspas no seu interior. Por exemplo

```
10 LET a$ = "Quando bem usados, ""os computadores
    são úteis""
20 LET b$ = a$ (19 TO)
30 PRINT b$
```

Este programa imprimirá no visor a cadeia

```
    "os computadores são úteis"
    Se substituirmos a linha 20 por
20 LET b$ = VAL$ a$ (19 TO)
```

a máquina imprimirá a cadeia

```
    os computadores são úteis
```

Exercício

Verifique os resultados obtidos em ambos os casos se substituir no programa anterior a linha 30 por

```
30 PRINT LEN VAL$ b$
```

STR\$

A função STR\$ permite-lhe converter números decimais em cadeias. Por exemplo, a declaração

PRINT STR\$ 562

apresenta no visor o número 562. É equivalente a usar a instrução
PRINT "562"

Exercício

Confirme que a instrução

```
PRINT LEN STR$ 691982.14
```

é equivalente a

```
PRINT LEN "691982.14"
```

Funções definidas pelo utilizador

O leitor pode definir as suas próprias funções de cadeia para uso num seu programa Basic. Estas serão designadas "funções definidas pelo utilizador", e a sua forma geral é

número de linha DEF FN letra\$ (argumento\$) = cadeia

Apresentamos em seguida três exemplos de formas válidas de funções de cadeia definidas pelo utilizador:

```
15 DEF FN a$() = "Maria"
25 DEF FN H$(b$) = "Maria" + b$
35 DEF FN T$(b$.c$) = "Maria" + b$ + c$
```

Note no entanto que antes de poder usar a função definida por si na linha 25 é necessário definir a variável de cadeia b\$, e que antes de poder usar a linha 35 deve igualmente definir a variável de cadeia c\$.

Exercício

Execute o seguinte programa e investigue o modo como utiliza as funções definidas

```

5 INPUT b$
10 PRINT b$
15 INPUT c$
25 PRINT c$
115 DEF FN a$( ) = "Maria"
125 DEF FN H$(b$) = FN a$( ) + b$
135 DEF FN T$(b$, c$) = FN H$(b$) + c$
145 PRINT FN a$( ); FN H$(b$)
160 PRINT FN T$(b$, c$)

```

Quadros de cadeias alfanuméricas

É possível incluir cadeias alfanuméricas num quadro. Um quadro de uma só dimensão com N elementos, onde N é um número inteiro positivo, deve ser DIMENSIONADO usando um único caracter alfabético seguido de um \$ e pelas dimensões do quadro. Por exemplo:

```
DIM D$ (1,N)
```

Cada elemento é identificado por um índice, usando o formato

```
D$(1, número do elemento)
```

como se mostra na figura 4.1.

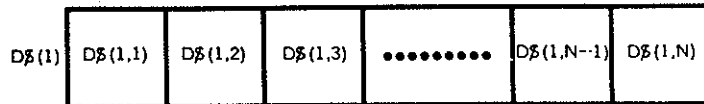


Figura 4.1 — Quadro de cadeias alfanuméricas de uma só dimensão

Considere a cadeia "TELEVISÃO", que contém 9 caracteres. Para definir esta cadeia podemos usar o seguinte:

```

15 DIM D$ (1,9)
25 LET D$ (1) = "TELEVISÃO"

```

obtendo-se o quadro da figura 4.2. Assim, para imprimir um único caracter deste quadro, por exemplo S, usaremos

```
PRINT D$ (1,7)
```

Por outro lado, para imprimir no visor a subcadeia LEV usá-riamos a instrução

```
PRINT D$ (1,3 TO 5)
```

enquanto que para imprimir toda a cadeia usá-riamos

```
PRINT D$ (1)
```

ou

```
PRINT D$ (1, TO)
```



Figura 4.2 — Quadro de uma só dimensão contendo uma cadeia com nove caracteres

Usando este método podemos indicar caracteres isolados no interior da cadeia, uma qualquer subcadeia, ou a cadeia inteira.

Um quadro de duas dimensões é construído de modo semelhante. No entanto, a instrução de DIMENSIONAMENTO deve reservar um espaço de memória suficiente para acomodar a cadeia mais comprida. Se por exemplo utilizarmos as quatro cadeias "Janeiro", "Fevereiro", "Março" e "Abril" poderemos dimensionar o quadro usando

```
DIM M$ (4,9)
```

onde 4 é o número de cadeias e 9 o número de caracteres da cadeia mais comprida. O arranjo do correspondente quadro a duas dimensões é apresentado na figura 4.3.

M\$(1)	J	A	N	E	I	R	O		
M\$(2)	F	E	V	E	R	E	I	R	O
M\$(3)	M	A	R	Ç	O				
M\$(4)	A	B	R	I	L				

Figura 4.3 — Quadro alfanumérico a duas dimensões, definindo quatro cadeias

Pode-se definir este quadro bi-dimensional usando

```
10 DIM M$ (4,8)
20 LET M$ (1) = "Janeiro"
30 LET M$ (2) = "Fevereiro"
40 LET M$ (3) = "Março"
50 LET M$ (4) = "Abril"
```

Exercício

Mostre que a subcadeia "eiro" pode ser obtida do quadro bi-dimensional citado usando

```
PRINT M$ (1,4 TO 7)
ou PRINT M$ (1,4 TO)
ou PRINT M$ (2,5 TO 8)
ou PRINT M$ (2,5 TO)
```

Vale a pena notar que

```
PRINT M$ (3,6 TO)
```

imprime no visor uma cadeia nula (quatro espaços) porque M\$ (3,6), M\$ (3,7), M\$ (3,8) e M\$ (3,9) são subcadeias vazias.

É igualmente possível usar um quadro de cadeias alfanuméricas sem qualquer dimensão! Nestes casos, por estranho que pareça, o quadro deve ser ainda dimensionado usando um único caractere alfabético seguido de \$, e do número de elementos (espaços) vazios. Por exemplo,

```
DIM c$ (5)
```

dimensiona o quadro alfanumérico vazio c\$ que contém cinco espaços. Trata-se de um método alternativo e mais eficaz (em termos de requisitos de memória) de definir espaços para uso no texto.

Exercício

Introduza o programa seguinte na máquina e execute-o a fim de verificar que o quadro c\$ (5) produz o mesmo efeito que o uso da cadeia S\$.

```
5 DIM c$ (5)
10 LET a$ = "Nome"
20 LET b$ = "Endereço"
30 LET S$ = " "
40 PRINT a$ + S$ + b$
50 PRINT a$ + c$ + b$
```

Observações finais

Estudando este capítulo o leitor deve aprender a usar cadeias alfanuméricas, variáveis de cadeia, subcadeias e quadros de cadeias. Conhecerá ainda as funções de cadeia LEN, VAL, VAL\$ e STR\$, descobrindo o modo de definir e usar as suas próprias funções de cadeia.

Verificará que as potencialidades de tratamento de cadeias do ZX Spectrum lhe fornecem um método versátil e poderoso de imprimir texto no visor.

CAPÍTULO V

DECISÕES LÓGICAS

Introdução

Neste capítulo vamos mostrar que as operações e decisões lógicas podem ser realizadas usando portas lógicas e que isto pode ser-nos útil quando ligamos o nosso ZX Spectrum a aparelhagem externa. Mostraremos ainda que as operações lógicas podem ser realizadas numa base bit a bit entre dois bytes de dados, o que é aplicável quando se escrevem programas em código-máquina. Concluiremos o capítulo com uma secção sobre a instrução condicional IF. Trata-se de uma declaração muito útil da Basic, que permite realizar decisões complexas nos nossos programas.

Operações lógicas AND, OR e NOT

O ZX Spectrum utiliza elementos lógicos (circuitos electrónicos) que produzem uma saída dependente do valor de uma ou mais variáveis de entrada. Os valores de entrada e saída encontram-se ao nível lógico 0 (0 V) ou ao nível lógico 1 (+ 5 V).

A porta AND produz uma saída lógica 1 quando todas as entradas se encontram a esse mesmo nível lógico; em qualquer outro caso a saída é 0. O símbolo e a tabela de verdade da porta AND de duas entradas são apresentados na figura 5.1.

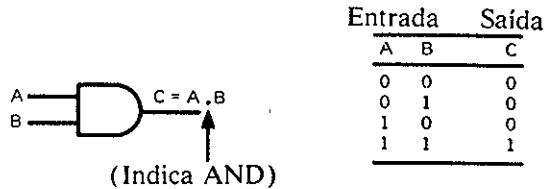


Figura 5.1 — Símbolo e tabela de verdade de uma porta AND de duas entradas

A porta OR (OR inclusivo) produz uma saída lógica 1 quando pelo menos uma das entradas se encontra ao nível lógico 1. Isto significa que a saída só é 0 quando todas as entradas o são igualmente. O símbolo e a tabela de verdade da porta OR de duas entradas são apresentados na figura 5.2.

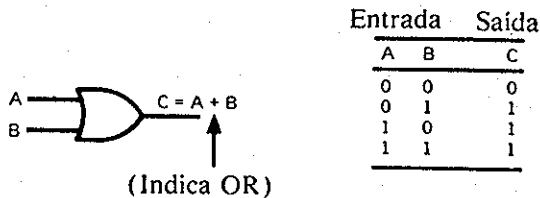


Figura 5.2 — Símbolo e tabela de verdade de uma porta OR de duas entradas

A porta NOT (porta inversora ou complemento) produz uma saída inversa (complementar) da entrada. O símbolo e a tabela de verdade desta função são apresentados na figura 5.3.

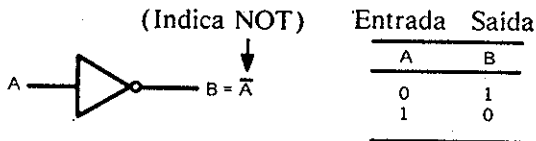


Figura 5.3 — Símbolo e tabela de verdade da porta NOT

No microprocessador Z80A as funções AND e OR são realizadas bit a bit entre dois bytes de dados. Se os dois bytes de dados forem $A = 100001010$ e $B = 11100111$, a operação AND ("E") é realizada do seguinte modo:

$$A = 10001010$$

$$B = 11100111$$

$$\text{Resultado} = A.B = 10000010$$

e a operação OR ("OU") produz

$$A = 10001010$$

$$B = 11100111$$

$$\text{Resultado} = A + B = 11101111$$

O microprocessador Z80A pode realizar a operação NOT (negação lógica) sobre um byte de dados usando uma instrução de código-máquina (CPL), que transforma 0s em 1s e 1s em 0s. Como exemplo, se $A = 11011011$ então o complemento de A, ou seja NOT A, é

$$A = 11011011$$

$$\text{Resultado} = \bar{A} = 00100100 = \text{NOT } A$$

Exercício

Considerando dois bytes de dados $A = 10101011$ e $B = 11110111$, mostre que numa operação bit a bit

$$A.B = 10100011$$

$$A + B = 11111111$$

$$\bar{A} = 01010100$$

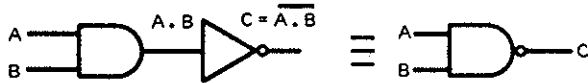
$$A + \bar{B} = 10101011 = A$$

Note que pode utilizar AND, OR e NOT em instruções condicionais IF nos seus programas Basic. No entanto, é necessário compreender nestes casos que, em vez de interpretar as operações lógicas em termos de 0s e 1s, o ZX Spectrum interpreta-os como falsos ou verdadeiros respectivamente. Este assunto será aprofundado mais tarde.

Operações lógicas NAND e NOR

Podem-se executar estas operações lógicas usando circuitos de porta. A porta NAND é obtida combinando uma porta AND e

uma porta NOT. Isto é mostrado na figura 5.4 para o caso de um elemento NAND de duas entradas. Esta porta só fornece uma saída lógica 0 quando todas as entradas se encontram ao nível lógico 1; em todos os outros casos produz uma saída 1. O símbolo e a tabela de verdade de uma porta NAND de duas entradas são apresentados na figura 5.4.



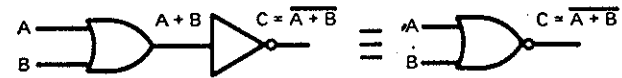
Entrada		Saída
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Figura 5.4 — Símbolo e tabela de verdade de uma porta NAND de duas entradas

A porta NOR é obtida combinando uma porta OR e uma porta NOT. Isto é ilustrado na figura 5.5 para o caso de um elemento NOR de duas entradas. Esta porta só produz uma saída lógica 1 quando todas as entradas têm o nível lógico 0; em todos os outros casos produz uma saída 0. O símbolo e a tabela de verdade de uma porta NOR de duas entradas são apresentados na figura 5.5.

Consideremos o modo como estas operações lógicas funcionam para dois bytes de dados numa base bit a bit. Usando $A = 11110101$ e $B = 01111100$ obteremos

$$\begin{array}{r}
 A = 11110101 \\
 B = 01111100 \\
 \hline
 \text{Resultado} = A \cdot B = 10001011 \\
 A = 11110101 \\
 B = 01111100 \\
 \hline
 \text{Resultado} = A + B = 00000010
 \end{array}$$



Entrada		Saída
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Figura 5.5 — Símbolo e tabela de verdade de uma porta NOR de duas entradas

É igualmente interessante notar que as operações NAND e NOR sobre as variáveis numéricas A e B podem ser substituídas por uma forma equivalente, obtida usando as Leis de Morgan, a saber

$$\begin{array}{l}
 \text{(NAND)} \quad \overline{A \cdot B} = \overline{A} + \overline{B} \\
 \text{(NOR)} \quad A + B = \overline{\overline{A} \cdot \overline{B}}
 \end{array}$$

Estas formas são indicadas nos diagramas lógicos da figura 5.6.

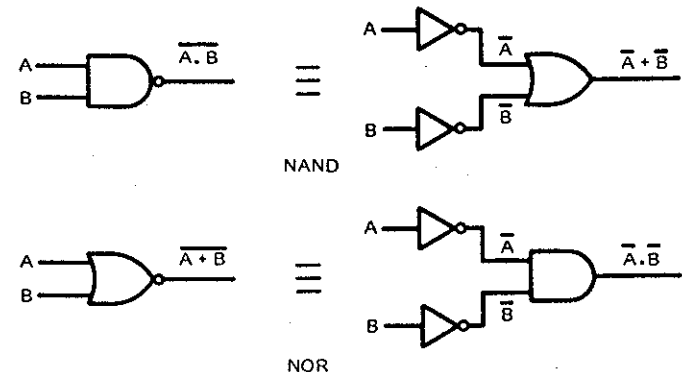


Figura 5.6 — Portas equivalentes usando as Leis de Morgan

Exercícios

1. dados dois bytes A = 10111000 e B = 00111001, mostrar que numa base bit a bit

Exercícios

1. dados dois bytes A = 10111000 e B = 00111001, mostrar que numa base bit a bit

$$\begin{array}{r} \overline{A \cdot B} = 11000111 \\ A + B = 01000110 \end{array}$$

2. Usando as Leis de Morgan prove que

$$\begin{array}{l} a + B = \overline{\overline{a} \cdot \overline{B}} \\ A \cdot B = \overline{\overline{A} + \overline{B}} \end{array}$$

Operação lógica OR-exclusivo

Pode-se executar esta operação usando um circuito de porta. A porta OR exclusiva dá uma saída lógica 1 quando qualquer das entradas, mas não todas elas, está ao nível lógico 1. Isto implica que a saída só estará ao nível lógico 0 quando todas as entradas tiverem o mesmo valor lógico, 0 ou 1. O símbolo e a tabela de verdade de uma porta Or-exclusivo de duas entradas são apresentados na figura 5.7.

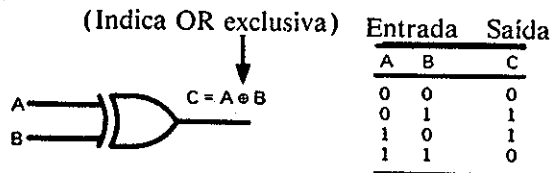


Figura 5.7 — Símbolo e tabela de verdade de uma porta OR exclusiva de duas entradas

Esta operação lógica pode ser realizada sobre dois bytes bit a bit. Se considerarmos A = 10000111 e B = 01110000 obteremos

$$\begin{array}{r} A = 10000111 \\ B = 01110000 \end{array}$$

$$\text{Resultado} = A \oplus B = 11110111$$

A operação OR-exclusivo pode ser realizada em código-máquina do Z80A utilizando uma instrução XOR.

Exercício

Dados dois bytes de dados, A=11100011 e B=10101010, mostrar que numa base bit a bit

$$A \oplus B = 01001001$$

Instrução condicional IF

Talvez o leitor necessite num programa Basic de tomar decisões tendo em conta certas condições. Para o fazer poderá utilizar instruções IF condicionais, que devem ser escritas sob a forma

IF condição THEN instrução

Na declaração Basic, a instrução que se segue a THEN pode ser usada condicionalmente para dirigir o programa para outra linha, por exemplo recorrendo a uma instrução GO TO; se a condição não for satisfeita o programa prosseguirá para a linha de programa que se segue imediatamente. Isto pode ser observado considerando as instruções:

```
20 IF h >= 12 AND h <= 112 THEN GO TO 45
30 STOP
```

que significam que no caso de a variável numérica *h* se encontrar na gama entre 12 e 112 a execução passa para a linha 45; no caso contrário passará para a linha seguinte, ou seja, para STOP. Podemos representar isto sob a forma de uma tabela de verdade:

h > = 12 (h maior ou igual a 12?)	h > = 112 (h menor ou igual a 112?)	Execução do programa passa para
Verdadeiro	Verdadeiro	Linha 45
Verdadeiro	Falso	Linha 30
Falso	Verdadeiro	Linha 30
Falso	Falso	Linha 30

As condições entre expressões matemáticas ou números podem ser definidas usando as seguintes relações matemáticas. Apresentam-se igualmente no ZX Spectrum os símbolos que as representam.

Igual a	Símbolo =
Diferente de	Símbolo < >
Menor do que	Símbolo <
Menor ou igual a	Símbolo < =
Maior do que	Símbolo >
Maior ou igual a	Símbolo > =

O programa listado em seguida ilustra um modo de determinar os efeitos do uso destas relações.

```

5 PRINT
15 PRINT "Indique valores para A e B"
25 INPUT A
35 INPUT B
45 CLS
55 IF condição THEN GO TO 85
65 PRINT "Condição falsa"
75 GO TO 5
85 PRINT "Condição verdadeira"
95 GO TO 5

```

Sugerimos ao leitor que introduza o programa na máquina e na sua primeira tentativa escreva entre IF e THEN a condição A = B (linha 55 do programa). Ao executar, verificará que para valores de A iguais a B o programa passa para a linha 85, enquanto que para valores de A diferentes de B o programa passa para a linha 65.

Exercício

Altere a linha 55 no programa anterior a fim de verificar cada uma das outras relações condicionais.

```

A < > B
A < B
A < = B
A > B
A > = B

```

É igualmente possível utilizar a declaração condicional IF para determinar o estado de cadeias. As relações condicionais consideradas acima podem ser usadas para verificar se duas cadeias são iguais ou se uma delas precede alfabeticamente a outra. Considerando que as duas cadeias são A\$ e B\$, podemos resumir as possíveis condições do seguinte modo:

```

A$ = B$   A$ igual a B$
A$ < > B$  A$ diferente de B$
A$ > B$   A$ segue-se a B$ em ordem alfabética
A$ < B$   A$ precede alfabeticamente B$
A$ > = B$ A$ segue-se a B$ em ordem alfabética ou é igual a ela.
A$ < = B$ A$ precede alfabeticamente B$ ou é igual a ela.

```

Por exemplo, se utilizarmos

```

55 IF A$ < B$ THEN GO TO 85
65 Linha seguinte do programa

```

isto será interpretado pelo ZX Spectrum como significando: se a cadeia A\$ precede alfabeticamente a cadeia B\$ então a execução deve passar para a linha de programa número 85, senão passará para a linha seguinte (número 65).

O programa listado em seguida, onde se deve introduzir a condição desejada na linha 55, fornece-lhe um método de investigação das condições já citadas.

```

5 PRINT
15 PRINT "Indique as cadeias A$ e B$"
25 INPUT A$

```

```

35 INPUT B$
45 CLS
55 IF condição THEN GO TO 85
65 PRINT "Condição falsa"
75 GOTO 5
85 PRINT "Condição verdadeira"
95 GO TO 5

```

É possível combinar decisões entre si recorrendo a operações lógicas. A Basic usada pelo ZX Spectrum possui as operações AND, OR e NOT para este fim.

A operação AND é usada para verificar se todas as condições usadas na declaração condicional IF são verdadeiras. Por exemplo, se alterarmos a linha 55 do exemplo anterior para

```
55 IF A$ = "Z" AND B$ = "W" THEN GO TO 85
```

o programa passa para a linha 85 apenas quando a cadeia A\$ é igual ao carácter único Z e a cadeia B\$ é igual ao carácter único W; noutras condições a execução passa para a linha seguinte.

A operação OR é usada para verificar se pelo menos uma das relações usadas na declaração condicional IF é verdadeira. Por exemplo, se alterarmos a linha 55 do programa anterior para

```
55 IF A$ = "Z" OR B$ = "W" THEN GO TO 85
```

o programa passa para a linha 85 quando a cadeia A\$ é igual ao carácter único Z ou a cadeia B\$ é igual ao carácter único W, ou ainda quando ambas as condições são verdadeiras. No caso contrário execução passa para a linha de programa seguinte.

A função NOT é usada para verificar se a relação que se segue a ela é ou não falsa. Por exemplo, se alterarmos a linha 55 do programa anterior para

```
55 IF NOT A$ = "Z" AND B$ = "W" THEN GO TO 85
```

o programa passa para a linha 85 quando a cadeia A\$ não for igual ao carácter único Z e a cadeia B\$ não for igual ao carácter W; de outro modo a execução passa para a linha de programa que se segue.

Observações finais

Neste capítulo aprendemos a maneira de realizar operações lógicas e decisões recorrendo a portas lógicas (hardware) e a instruções condicionais IF (software). O leitor nunca deve esquecer que os processos lógicos têm de ser definidos sem qualquer ambiguidade, e para o conseguir se torna normalmente necessário construir um fluxograma. Esta questão é discutida no capítulo que se segue.

CAPÍTULO VI

FLUXOGRAMAS, CICLOS E SUBROTINAS

Introdução

A primeira consideração a ter em conta ao preparar um programa relativamente complexo consiste em definir a sequência lógica dos passos a executar pelo microcomputador para atingir o resultado desejado. A preparação inicial pode envolver o desenho de um *fluxograma* que ilustre os passos do programa, e inclua os ciclos (*loops*) e subrotinas envolvidas. Cada passo da sequência lógica do fluxograma deve então ser convertido em funções ou segmentos Basic apropriados.

Fluxogramas

Um fluxograma é constituído por símbolos gráficos ligados entre si por linhas rectas. São desenhadas setas nas linhas que ligam estes símbolos gráficos de modo a indicarem a direcção do fluxo do programa. Na figura 6.1 apresenta-se uma selecção dos símbolos gráficos mais usados em fluxogramas.

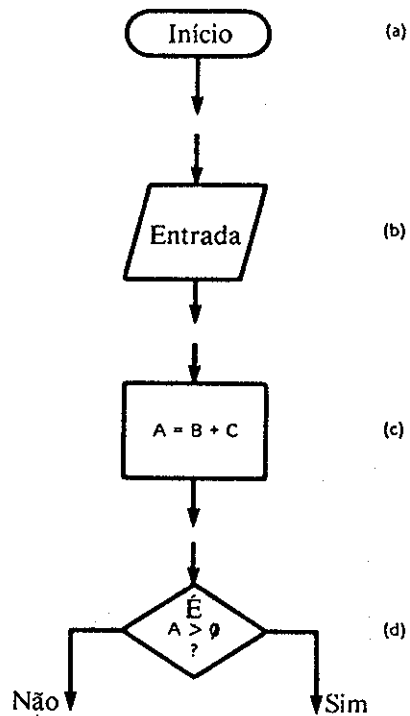


Figura 6.1 — Símbolos de fluxogramas. (a) símbolo terminal; (b) símbolo entrada/saída; (c) símbolo de execução; (d) símbolo de decisão

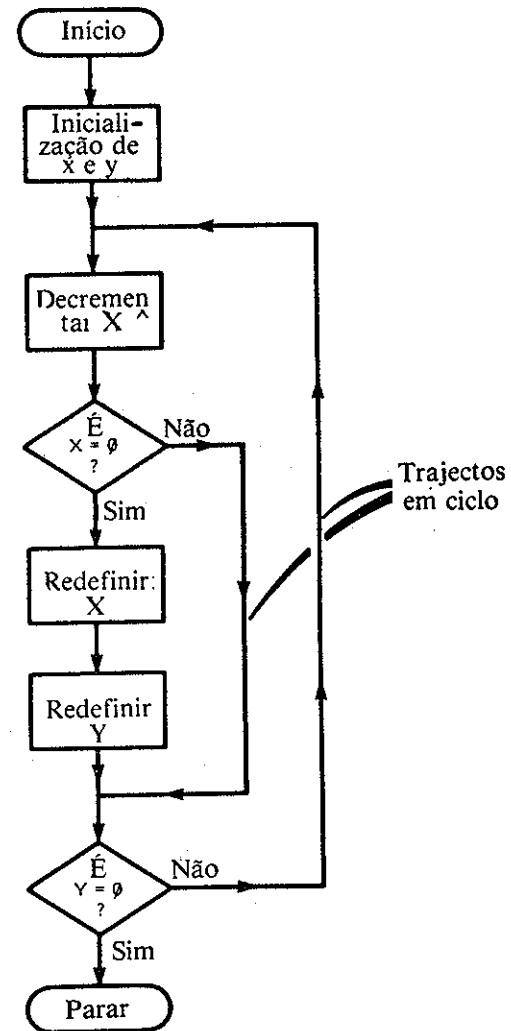


Figura 6.2 — Fluxograma de um programa que produz atrasos de tempo

O início e o fim do fluxograma é indicado pelo símbolo terminal mostrado na figura 6.1(a), e obviamente este símbolo é ligado ao resto do fluxograma apenas por uma linha. O paralelograma apresentado na figura 6.1(b) é usado para indicar uma operação específica a realizar por um dispositivo de entrada ou saída. O retângulo da figura 6.1(c) indica a realização de uma dada tarefa. Uma declaração no interior do retângulo especifica a tarefa em causa, que pode aliás ser identificada em português normal ou, alternativamente, ser substituída por uma expressão lógica ou algébrica. O losango da figura 6.1(d) é usado para indicar o ponto de uma programa em que é necessário realizar uma decisão. Por exemplo, pode ser necessário descobrir em algum ponto do programa se a variável numérica A é maior do que zero, ver figura 6.1(d), e obviamente a resposta a esta pergunta é sim ou não. O programa prosseguirá ao longo de dois trajetos alternativos conforme a decisão tomada.

Como exemplo, a figura 6.2 mostra o fluxograma básico que pode ser usado para representar a produção de um atraso. Este fluxograma contém dois ciclos, pelo que é chegado o momento de considerar as operações de ciclo antes de passarmos à tradução de um fluxograma em programa.

Ciclos

Um ciclo é uma sequência de instruções repetidas pelo computador. É possível levá-lo a repetir a sequência um número especificado de vezes usando as instruções FOR, NEXT, TO e STEP no formato:

```
FOR variável = valor inicial TO valor final STEP passo
Sequência de instruções
NEXT variável
```

Por exemplo, se tivermos um quadro numérico de uma dimensão contendo seis elementos que desejamos inicializar com os dados apropriados, podemos utilizar o seguinte ciclo para dar entrada os dados:

```
10 DIM C(6)
20 FOR X = 1 TO 6
30 INPUT d
35 PRINT d
40 LET C(X) = d
50 NEXT X
```

O leitor notará que a linha 20 não inclui STEP e que nestas condições o passo admitido pelo computador é 1.

O programa começa por atribuir a X o valor de 1 e em seguida executa as linhas 30 a 40 inclusive, isto é, recebe dados para o elemento C(1) do quadro; a linha 50 dirige novamente o programa para a linha 30 (funcionamento em ciclo) e atribui a X o valor inicial incrementando-o de um (valor do passo); o computador volta a executar a sequência de instruções, recebendo dados para o elemento C(2) do quadro numérico. Esta execução repete-se até ser dada entrada aos dados relativos a C(6).

Exercício

Altere a linha 20 do programa anterior para

```
20 FOR X = 6 TO 1 STEP - 1
```

e verifique que o programa inicializa o quadro numérico pela ordem inversa.

Um método alternativo de execução de um ciclo consiste em usar uma instrução GO TO juntamente com a instrução condicional IF discutida no capítulo 5.

Para realizar a operação de inicialização dos seis elementos do quadro numérico, poderemos usar a declaração GO TO e IF do modo indicado em seguida:

```
10 DIM C(6)
20 LET X = 1
30 INPUT d
```



```

35 PRINT d
40 LET C(X) = d
50 LET X = X + 1
60 IF X <> 7 THEN GO TO 30

```

Exercício

Alterar as linhas 20, 50 e 60 do programa para

```

20 LET X = 6
50 LET X = X - 1
60 IF X <> 0 THEN GO TO 30

```

e verifique que o programa inicializa o quadro de uma dimensão por ordem inversa.

Tradução de um fluxograma em programa

Na figura 6.2 mostra-se um fluxograma que pode ser usado para representar a produção de um atraso de tempo. Note que a duração do atraso depende dos valores atribuídos às variáveis inteiras positivas X e Y. Sugerimos-lhe que estude este fluxograma a fim de compreender o significado de cada passo da sequência de operações. Verificará que é prático atribuir valores a X e a Y, por exemplo 3 e 2 respectivamente, e determinar usando papel e lápis os valores intermédios de X e Y depois de cada passo. Note que ao alterar os valores de X e Y o número de passos é modificado de modo correspondente, dado que o tempo requerido para executar a sequência de operações pode ser variado, obtendo-se de facto um atraso variável entre o início e o fim.

Para cada passo da sequência lógica do fluxograma temos de escolher as funções ou declarações Basic apropriadas à execução da operação desejada.

A fim de ilustrar os princípios aqui envolvidos vamos escrever um programa de atraso de tempo correspondente ao fluxograma da figura 6.2. Os passos envolvidos na tradução do programa são indicados em seguida sob a forma de uma tabela.

<i>Fluxograma</i>	<i>Instruções Basic apropriadas</i>
Início	Início manual da execução do programa usando RUN ou ENTER
Definir valores de X e Y	INPUT X INPUT Y
Decrementar X	LET X = X - 1
X é igual a 0?	IF X <> 0 THEN GO TO n.º de linha
Redefinir X	LET X = S (S deve ter sido inicializado para o primeiro valor de X)
Decrementar Y	LET Y = Y - 1
Y é igual a 0?	IF Y <> 0 THEN GO TO n.º de linha
Parar	STOP

Para escrever o programa, cada instrução requer obviamente um número de linha (é conveniente deixar bastante espaço entre números de linha sucessivos a fim de inserir novas linhas em caso de necessidade); é ainda necessário inserir o número de linha apropriado nas declarações condicionais IF. O leitor deve notar ainda que o valor inicial de X deve ser reestabelecido durante o programa, o que se consegue atribuindo-o a uma variável S imediatamente depois de ser iniciada a variável X.

O programa é listado em seguida; foram incluídas instruções PRINT nos pontos apropriados a fim de dar ao utilizador informações sobre o que acontece durante a execução.

```

5 PRINT "Programa de demonstração de Atrasos de
Tempo" "Indique inteiro positivo X"
15 INPUT X
25 PRINT X
55 LET S = X
65 PRINT "Indique inteiro positivo Y"
75 INPUT Y
85 PRINT Y
95 LET X = X - 1
115 IF X <> 0 THEN GO TO 135
120 LET X = S
125 LET Y = Y - 1
135 IF Y <> 0 THEN GOTO 95
145 PRINT "Fim de execução"

```

Sugerimos ao leitor que execute este programa para $X = Y = 25$. Verificará que o tempo necessário para executar o programa é aproximadamente 10 segundos. Experimente outros valores de X e Y mas não utilize números demasiado elevados a menos que disponha de muito tempo para esperar pela resposta "Fim de execução".

O programa anterior deve utilizar valores inteiros positivos para X e Y , e para assegurar que os números negativos, não inteiros e diferentes de 0 sejam rejeitados pelo programa pode-se incluir as duas linhas adicionais:

```
20 IF X <= 0 OR (X - INT X) <> 0 THEN GO TO 5
80 IF Y <= 0 OR (Y - INT Y) <> 0 THEN GO TO 65
```

Utilizámos esta forma de atraso no programa 2 no Apêndice A para produzir um atraso de aproximadamente 1 segundo, que é incorporado num cronómetro. Talvez lhe interesse experimentar este programa.

Subrotinas

Em muitos programas o leitor verificará que o mesmo conjunto de instruções pode ser necessário mais do que uma vez, e nestes casos é conveniente programar-se esse conjunto comum de instruções sob a forma de uma subrotina.

Uma subrotina é um conjunto de instruções que podem ser chamadas a partir de qualquer ponto do programa. É normalmente colocada no início ou no fim do programa principal. O processo é ilustrado na figura 6.3. Para explicar o princípio podemos considerar o problema de um temporizador que utiliza uma subrotina de atraso de tempo.

O problema considerado consiste em usar o ZX Spectrum para apresentar no visor um valor inteiro positivo que é decrementado segundo a segundo até atingir zero. O valor apresentado deve indicar o número de segundos restante em cada semiperíodo, e qual destes está a diminuir.

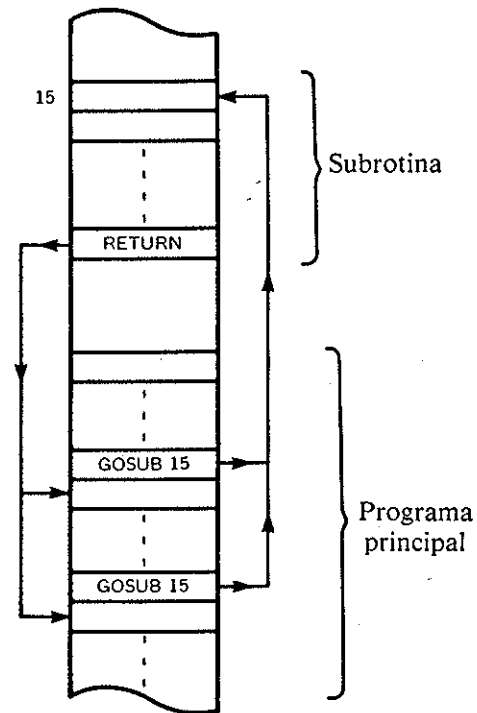


Figura 6.3 — Ilustração do GOSUB e RETURN de subrotinas

No programa em causa a subrotina (linhas 150 a 256) é usada para criar o atraso requerido de cerca de um segundo. O leitor pode verificar que o conjunto de instruções a repetir é acedido através uma instrução GOSUB em dois pontos diferentes do programa principal, nas linhas 55 e 115, e que o necessário RETURN (ver a figura 6.3) é colocado no final da subrotina, na linha 265.

Talvez o programa seguinte lhe interesse para cronometrar jogos de competição que utilizem períodos de tempo previamente definidos, por exemplo o xadrez, etc.

```

5 PRINT "Indique número de segundos em cada metade"
10 INPUT N
12 IF N - INT N <> 0 THEN GO TO 135
20 CLS
24 PRINT "Número de segundos restante" " " "na primeira
metade:"
25 FOR J = N TO 0 STEP - 1
55 GO SUB 150
65 NEXT J
66 CLS
70 PRINT "Número de segundos restante" " " "na segunda
metade:"
75 FOR J = N TO 0 STEP - 1
115 GO SUB 150
120 NEXT J
125 STOP
135 CLS
145 GO TO 5
150 PRINT AT 1, 18;" "
170 PRINT AT 1, 18; J
175 LET X = 14
185 LET S = X
195 LET Y = 4
215 LET X = X - 1
225 IF X <> 0 THEN GO TO 255
235 LET X = S
245 LET Y = Y - 1
255 IF Y <> 0 THEN GO TO 215
265 RETURN

```

Observações finais

Neste capítulo o leitor aprendeu que os fluxogramas ajudam a determinar os passos necessários de um programa e que a sequência de passos resumida no fluxograma pode ser facilmente traduzida para programa.

Verificará que a eficácia dos seus programas pode ser muitas vezes melhorada usando ciclos e subrotinas apropriados.

CAPÍTULO VII GRÁFICOS A CORES

Introdução

Podem-se criar imagens a cores num televisor apropriado usando as oito cores disponíveis no ZX Spectrum (ver a fotografia 7.1). As cores são indicadas acima das teclas 0 a 7 e listam-se e seguir.

Tecla	Cor	
0	Negro	(máximo da escala de cinzentos)
1	Azul	↓ Gama decrescente de cinzentos
2	Vermelho	
3	Magenta	
4	Verde	
5	Ciã	
6	Amarelo	
7	Branco	

Se estiver a usar um receptor de televisão monocromático pode usar as teclas de números 0 a 7 para escolher e apresentar no visor uma gama de cinzentos — o que é também indicado na lista anterior.

A região do visor onde se imprimem caracteres é chamada "papel" (*paper*) e a região que a circunda é a margem (*border*). Por exemplo, se utilizar

BORDER 3: PAPER 6

e ENTER estas ordens, só será apresentada a cor da margem.

magenta; a de papel não se altera. No entanto, se carregar novamente em ENTER o papel muda para amarelo. Isto ocorre devido ao facto de a ordem PAPER alterar o *atributo* do papel, mas a cor deste só se altera de facto quando o computador recebe ordem para enviar caracteres para o visor.

O leitor pode investigar isto dando entrada e executando o programa seguinte.

```
5 BORDER 3
10 PAPER 6
15 CLS
18 STOP
20 PAPER 1
35 PRINT "***"
40 STOP
45 PAPER 4
80 LIST
```

Neste programa poderá notar que a linha 5 define a cor da margem, e que a linha 10 inicializa o atributo do papel para todos os caracteres (amarelo). A linha 15 ordena ao computador que imprima espaços usando os atributos definidos até então, pelo que todo o papel muda para amarelo. Para continuar escreve-se a ordem CONTINUE, e nota-se que a linha 20 define o atributo de papel para todos os caracteres como sendo 1 (azul), e a linha 35 imprime os caracteres "***" que agora aparecem sobre papel azul tal como foi definido. Note que, como a linha 35 só exige a impressão de três caracteres, só a parte do papel que lhes diz respeito é modificada para azul.

Quando escreve novamente CONTINUE, verifica que na linha 45 o papel passa para a cor 4 (verde), e a linha 80 ordena ao computador que liste o programa. Só os caracteres correspondentes à listagem são apresentados sobre fundo verde, porque estes são os únicos caracteres impressos pela ordem LIST.

O leitor verificará ainda que os caracteres apresentados pelo programa anterior são todos negros, isto é, que a tinta (*ink*) é negra. Para alterar a cor dos caracteres usa-se a ordem INK seguida do número da cor pretendida. Se por exemplo escrever

```
INK 2
```

todos os caracteres impressos subsequentemente serão vermelhos.

até se redefinir a cor de tinta.

Além de usar os números 0 a 7 para definir uma cor, pode usar os números 8 e 9 depois das ordens PAPER e INK. O número 8 é usado com estas ordens para significar "sem alteração", isto é, para manter a cor de papel ou de tinta como estava anteriormente. O número 9 é usado com estas ordens para significar "contraste": isto é, o papel ou a tinta são tornados brancos se o outro é negro, azul, vermelho ou magenta, ou negro se o outro for verde, cião, amarelo ou branco.

O leitor pode estar interessado em apresentar caracteres no visor cintilando ou não, o que se consegue recorrendo à ordem FLASH seguida de 0, 1 ou 8, onde 0 define uma imagem estável, 1 uma imagem cintilante e 8 a não alteração do atributo anterior. Quando os caracteres cintilam as cores de papel e tinta são trocadas periodicamente.

A ordem BRIGHT pode ser usada para apresentar os caracteres com uma intensidade normal ou aumentada. Recorre-se para tal à ordem BRIGHT seguida de 0, 1 ou 8, onde 0 define um visor normal, 1 um visor mais brilhante e 8 mantém o atributo anterior.

Quando se pretende trocar as cores de papel e tinta (inverter a imagem) pode-se usar a ordem

```
INVERSE 1
```

e para voltar às cores de papel e tinta normais usa-se a ordem INVERSE 0.

Uma outra ordem muito útil é a OVER, que quando utilizada sob a forma

```
OVER 1
```

permite imprimir novos caracteres sobre os que já se encontram no visor. Para mais detalhes veja a secção seguinte. A ordem

```
OVER 0
```

apaga os caracteres anteriores sempre que imprime novos caracteres no mesmo local.

Na secção seguinte vamos examinar o conjunto de caracteres do ZX Spectrum e descrever o modo de definir caracteres próprios.

O conjunto de caracteres

O conjunto de caracteres do ZX Spectrum é resumido na tabela 7.1. O leitor deve notar que cada caracter possui um único código decimal na gama 0 a 255, mas que alguns dos códigos não são usados e outros são usados como caracteres de comando. Existem no total 224 caracteres que podem ser impressos directamente no visor, a saber, entre 32 e 255.

Nas aplicações em que se requer o código do primeiro caracter de uma cadeia, utilizamos a função CODE. Pelo contrário, quando queremos obter um caracter a partir do código que lhe corresponde usamos a função CHR\$. Se executarmos

```
PRINT CODE "Foguete"
```

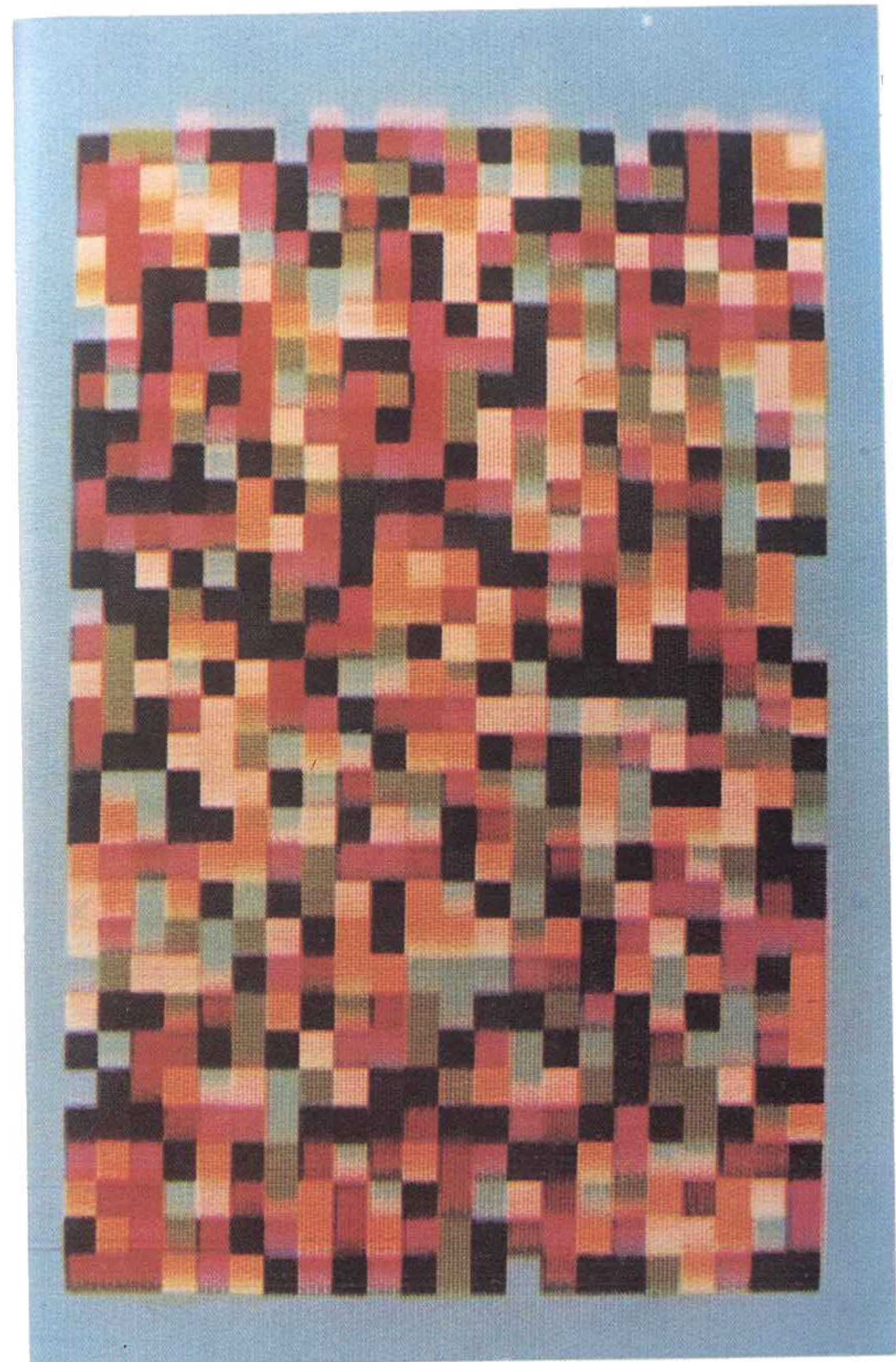
veremos impresso o número 70, código correspondente a F. Se executarmos

```
PRINT CHR$ 36
```

veremos o caracter \$ impresso no visor.

Tabela 7.1 — Conjunto de caracteres do ZX Spectrum

Código	Caracter	Hexadecimal
0 a 5	Não usado	00 a 05
6	Separador vírgula da ordem PRINT	06
7	Edit	07
8	Cursor esquerda	08
9	Cursor direita	09
10	Cursor para baixo	0A
11	Cursor para cima	0B
12	Delete	0C
13	Enter	0D
14	número	0E
15	Não usado	0F
16	Comando Ink	10
17	Comando Paper	11
18	Comando Flash	12
19	Comando Bright	13
20	Comando Inverse	14
21	Comando Over	15
22	Comando AT	16
23	Comando TAB	17
24 a 31	Não usado	18 a 1F
32	espaço	20



Rede de caracteres com atributos de cor definidos aleatoriamente.

33 !
 34 "
 35 #
 36 \$
 37 %
 38 &
 39 '
 40 (
 41)
 42 *
 43 +
 44 ,
 45 -
 46 .
 47 /
 48 0
 49 1
 50 2
 51 3
 52 4
 53 5
 54 6
 55 7
 56 8
 57 9
 58 :
 59 ;
 60 <
 61 =
 62 >
 63 ?
 64 @
 65 A
 66 B
 67 C
 68 D
 69 E
 70 F
 71 G
 72 H
 73 I
 74 J
 75 K
 76 L
 77 M
 78 N
 79 O
 80 P

Saída produzida pelo programa de impressão dos códigos em ROM.

33	!	21	81	Q	51
34	"	22	82	R	52
35	#	23	83	S	53
36	\$	24	84	T	54
37	%	25	85	U	55
38	&	26	86	V	56
39	'	27	87	W	57
40	(28	88	X	58
41)	29	89	Y	59
42	*	2A	90	Z	5A
43	+	2B	91	[5B
44	,	2C	92	/	5C
45	-	2D	93]	5D
46	.	2E	94	t	5E
47	/	2F	95	_	5F
48	0	30	96	£	60
49	1	31	97	a	61
50	2	32	98	b	62
51	3	33	99	c	63
52	4	34	100	d	64
53	5	35	101	e	65
54	6	36	102	f	66
55	7	37	103	g	67
56	8	38	104	h	68
57	9	39	105	i	69
58	:	3A	106	j	6A
59	;	3B	107	k	6B
60	<	3C	108	l	6C
61	=	3D	109	m	6D
62	>	3E	110	n	6E
63	?	3F	111	o	6F
64	@	40	112	p	70
65	A	41	113	q	71
66	B	42	114	r	72
67	C	43	115	s	73
68	D	44	116	t	74
69	E	45	117	u	75
70	F	46	118	v	76
71	G	47	119	w	77
72	H	48	120	x	78
73	I	49	121	y	79
74	J	4A	122	z	7A
75	K	4B	123	{	7B
76	L	4C	124		7C
77	M	4D	125	}	7D
78	N	4E	126	-	7E
79	O	4F	127	©	7F
80	P	50	128	ll	80

129 █
 130 █
 131 █
 132 █
 133 █
 134 █
 135 █
 136 █
 137 █
 138 █
 139 █
 140 █
 141 █
 142 █
 143 █
 144 (a) █
 145 (b) █
 146 (c) █
 147 (d) █
 148 (e) █
 149 (f) █
 150 (g) █
 151 (h) █
 152 (i) █
 153 (j) █
 154 (k) █
 155 (l) █
 156 (m) █
 157 (n) █
 158 (o) █
 159 (p) █
 160 (q) █
 161 (r) █
 162 (s) █
 163 (t) █
 164 (u) █
 165 RND █
 166 INKEY\$ █
 167 PI █
 168 FN █
 169 POINT █
 170 SCREENS █
 171 ATTR █
 172 AT █
 173 TAB █
 174 VAL\$ █
 175 CODE █
 176 VAL █

Gráficos do
 utilizador

81
 82
 83
 84
 85
 86
 87
 88
 89
 8A
 8B
 8C
 8D
 8E
 8F
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 9A
 9B
 9C
 9D
 9E
 9F
 A0
 A1
 A2
 A3
 A4
 A5
 A6
 A7
 A8
 A9
 AA
 AB
 AC
 AD
 AE
 AF
 B0

177 LEN
 178 SIN
 179 COS
 180 TAN
 181 ASN
 182 ACS
 183 ATN
 184 LN
 185 EXP
 186 INT
 187 SQR
 188 SGN
 189 ABS
 190 PEEK
 191 IN
 192 USR
 193 STR\$
 194 CHR\$
 195 NOT
 196 BIN
 197 OR
 198 AND
 199 < =
 200 > =
 201 <>
 202 LINE
 203 THEN
 204 TO
 205 STEP
 206 DEF FN
 207 CAT
 208 FORMAT
 209 MOVE
 210 ERASE
 211 OPEN #
 212 CLOSE #
 213 MERGE
 214 VERIFY
 215 BEEP
 216 CIRCLE
 217 INK
 218 PAPER
 219 FLASH
 220 BRIGHT
 221 INVERSE
 222 OVER
 223 OUT
 224 LPRINT

B1
 B2
 B3
 B4
 B5
 B6
 B7
 B8
 B9
 BA
 BB
 BC
 BD
 BE
 BF
 C0
 C1
 C2
 C3
 C4
 C5
 C6
 C7
 C8
 C9
 CA
 CB
 CC
 CD
 CE
 CF
 D0
 D1
 D2
 D3
 D4
 D5
 D6
 D7
 D8
 D9
 DA
 DB
 DC
 DD
 DE
 DF
 E0

225 LLIST
 226 STOP
 227 READ
 228 DATA
 229 RESTORE
 230 NEW
 231 BORDER
 232 CONTINUE
 233 DIM
 234 REM
 235 FOR
 236 GO TO
 237 GO SUB
 238 INPUT
 239 LOAD

E1
 E2
 E3
 E4
 E5
 E6
 E7
 E8
 E9
 EA
 EB
 EC
 ED
 EE
 EF

240 LIST
 241 LET
 242 PAUSE
 243 NEXT
 244 POKE
 245 PRINT
 246 PLOT
 247 RUN
 248 SAVE
 249 RANDOMIZE
 250 IF
 251 CLS
 252 DRAW
 253 CLEAR
 254 RETURN
 255 COPY

F0
 F1
 F2
 F3
 F4
 F5
 F6
 F7
 F8
 F9
 FA
 FB
 FC
 FD
 FE
 FF

Exercicio

Utilize o programa seguinte para obter uma imagem de todos os caracteres que podem ser impressos no visor (ver a fotografia 7.2).

```
10 FOR x = 255 TO 32 STEP - 1
15 PRINT CHR$ x:
20 NEXT x
```

Para imprimir os 16 símbolos gráficos, códigos 128 a 143 na tabela 7.1, deve-se colocar o teclado em modo gráfico (cursor G). Para definir este modo carrega-se em GRAPHICS (CAPS SHIFT e 9), e para sair dele carrega-se novamente em GRAPHICS. Note que para obter os oito caracteres gráficos com códigos 128 a 135, se carrega na tecla apropriada, 1 a 8; mas que para obter os símbolos gráficos inversos, de códigos 136 a 143, se carrega na tecla apropriada, 1 a 8, e na tecla SYMBOL SHIFT.

Exercicio

Escreva

```
PRINT CODE "█"
```

e verifique que é impresso o número 143.

Caracteres guardados em ROM (Read Only Memory)

Os caracteres listados na tabela 7.1 que possuem códigos na gama 32 a 127 inclusive são guardados num bloco de ROM (Read Only Memory) entre os endereços 15616 a 16383. Cada caracter é definido pelo conteúdo de oito bytes sucessivos. O caracter espaço (código 32) é guardado nos oito bytes entre 15616 e 15623, e os códigos que se lhe seguem são guardados em grupos sucessivos de oito bytes.

Os oito bytes de qualquer destes caracteres são guardados em memória desde o endereço 15616 + (8 * (código - 32)) até ao endereço 15623 + (8 * (código - 32)). Por exemplo, o caracter "+" (código 43) é guardado nos endereços de memória 15704 a 15711. Vamos descobrir o conteúdo destas posições de memória usando a função PEEK:

```
10 FOR x = 0 TO 7
15 PRINT PEEK (15704 + x)
20 NEXT x
25 STOP
```

Os valores obtidos (a função PEEK lê os valores guardados num byte) são

Endereço de memória	Conteúdo em decimal
15704	0
15705	0
15706	8
15707	8
15708	62
15709	8
15710	8
15711	0

O leitor recorda do capítulo 2 que o conteúdo de uma posição de memória é armazenado sob a forma de uma palavra binária de 8 bits, pelo que o caracter + é guardado em oito bytes sob a forma binária

Endereço de memória	Conteúdo em binário
15704	00000000
15705	00000000
15706	00001000
15707	00001000
15708	00111110
15709	00001000
15710	00001000
15711	00000000

Quando o computador imprime este caracter +, interpreta o padrão de bits considerando 0 como papel e 1 como tinta, e imprime o resultado usando uma rede de 8 x 8 elementos (*pixels*), como se mostra na figura 7.1.

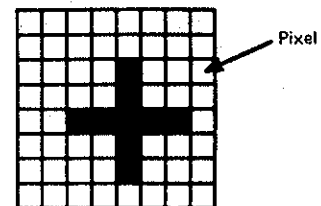


Figura 7.1 — Rede de pixels definindo o caracter +

Exercício

Verifique que o caracter £ (código 96) é impresso usando a rede de pixels da figura 7.2.

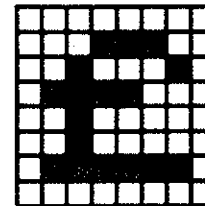


Figura 7.2 — Rede de pixels definindo o caracter £

Pode-se usar o programa 3 do Apêndice para imprimir, para um dado código de entrada, a rede de 8×8 bits correspondente ampliada de um factor de 64. A fotografia 7.3 mostra um exemplo da saída deste programa.

Caracteres definidos pelo utilizador

Na secção anterior vimos como se define e guarda em memória (neste caso ROM) um conjunto de caracteres. Estes caracteres não podem ser alterados, mas quando se deseja utilizar caracteres diferentes pode-se defini-los e guardá-los numa área apropriada da memória volátil (RAM; Random Access Memory).

É possível definir até 21 novos caracteres gráficos, o que requer 168 bytes de memória, e a área da RAM reservada para este uso corresponde a 32600 a 32767 no Spectrum de 16 K e a 65368 a 65535 no Spectrum de 48 K.

Pode-se obter o endereço do primeiro byte em memória de qualquer dos gráficos definidos pelo utilizador recorrendo à função USR sob a forma

```
PRINT USR "símbolo gráfico"
```

e o computador imprimirá o endereço apropriado.

Por exemplo:

```
PRINT USR "D"
```

imprime o endereço 32624 no Spectrum de 16 K, e 65392 no de 48 K.

A área da RAM usada para os caracteres definidos pelo utilizador é inicializada, ao ligar o computador, com bytes de dados que reproduzem os caracteres alfabéticos A a U. Os oito bytes de qualquer destes caracteres são armazenados na memória a partir do endereço $\text{End} + 8 * (\text{código} - 144)$ até ao endereço $\text{End} + 7 + 8 * (\text{código} - 144)$, onde código possui um valor na gama 144 a 164 correspondente aos caracteres definidos pelo utilizador A a U, como se indica na tabela 7.1, e End é igual a 32600 no caso do Spectrum de 16 K e 65368 no de 48 K.

Por exemplo, no Spectrum de 16 K, se examinarmos o conteúdo dos oito bytes de memória com endereços entre 32672 e 32679 inclusive, veremos que o caracter J está guardado sob a

forma binária

Endereço de memória	Conteúdo em binário
32672	00000000
32673	00000010
32674	00000010
32675	00000010
32676	01000010
32677	01000010
32678	00111100
32679	00000000

Um modo simples de verificar isto consiste em executar o programa 3, dado no Apêndice A, alterando a linha 30 para

```
30 LET j = 32599 + 8 * (código - 144) + X
(Spectrum de 16 K)
```

ou

```
30 LET j = 65367 + 8 * (código - 144) + X
(Spectrum de 48 K)
```

Exercício

Execute o programa 3 do Apêndice A, com a linha 30 alterada do modo indicado, e verifique que os caracteres inicializados para os códigos 145 e 162 são B e S respectivamente.

Os caracteres definidos pelo utilizador não são obviamente muito úteis sob a forma em que se encontram inicializados, dado que os mesmos caracteres A a U existem já como parte do conjunto de caracteres guardado em ROM. No entanto, como os caracteres definidos pelo utilizador são guardados em RAM, pode-se produzir outros carregando os correspondentes grupos de oito bytes em qualquer das 21 áreas que lhes correspondem. Por exemplo, para obter um caracter com a forma apresentada na figura 7.3, é necessário definir primeiramente o padrão de bits de cada byte do caracter. Neste exemplo teremos

```

00001000
00010100
00100010
01000001
00100010
00010100
00001000
00000000

```

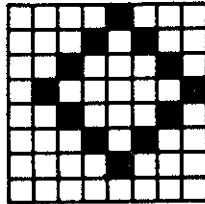


Figura 7.3 — Losango definido pelo utilizador

Consideremos que este caracter deve ser identificado como correspondendo ao caracter definido pelo utilizador D. Teremos de guardar os bytes indicados nos endereços de memória End a End+7, sendo End igual a 32624 no caso do Spectrum de 16 K e 65392 no de 48 K. Podemos fazê-lo recorrendo à função POKE, que coloca directamente num byte o valor que quisermos:

```

POKE End, BIN 00001000
POKE END+1, BIN 00010100

```

```

.....
POKE End+7, BIN 00000000

```

Para examinar o padrão de bits e a definição da rede de pixels deste caracter podemos utilizar novamente o programa 3 do Apêndice A, alterando a linha 30 para

```

30 LET j = 32599 + 8 * (código - 144) + X
(Spectrum de 16 K)

```

ou

```

30 LET j = 65367 + 8 * (código - 144) + X
(Spectrum de 48 K)

```

Para incluir um caracter definido pelo utilizador num programa Basic passa-se ao modo gráfico e acede-se o caracter carregando na tecla com o símbolo alfabético apropriado, A a U. No caso do exemplo acima, para apresentar o caracter "losango" no centro do visor escreve-se

```
PRINT AT 10,15;"◇"
```

obtendo-se ◇ passando ao modo gráfico e carregando na tecla D.

Exercício

Usando a área em RAM de USR "P" a USR "P" + 7, guarde o padrão binário correspondente ao caracter apresentado na figura 7.4. Imprima o caracter no visor e verifique se foi correctamente guardado.

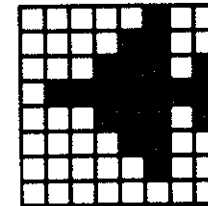


Figura 7.4 — Caracter gráfico representando um avião

A rede de 8 × 8 pixels permite-nos representar $2^8 \times 2^8$ (1,8446744E + 19) caracteres diferentes, mas só podemos utilizar 21 caracteres gráficos diferentes de cada vez (para além dos próprios da máquina). Para criar mais caracteres, pode-se no entanto recorrer à ordem OVER 1, que permite sobrepor dois ou mais caracteres; mas não esqueça que os pixels de tinta coincidentes são impressos com a cor de papel.

Como exemplo, consideremos o caracter produzido sobrepondo > e <. As definições de > e < são apresentadas nas figuras 7.5 (a) e (b) respectivamente, e o caracter resultante é apresentado na figura 7.5 (c). Podemos imprimir o novo caracter executando as seguintes linhas de programa:

```

10 PRINT AT 10,15;"<"
15 OVER 1
20 PRINT AT 10,15;">"

```

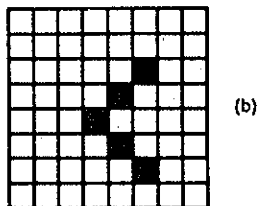
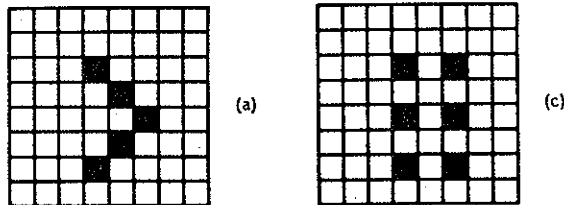


Figura 7.5 — Exemplo de uso da instrução OVER 1

A imagem no visor

O "papel" do visor é formado por uma rede de 22 × 32 caracteres, como se mostra na figura 7.6. As linhas são numeradas de 0 a 21 começando por cima, e as colunas são numeradas de 0 a 31 começando pelo lado esquerdo.

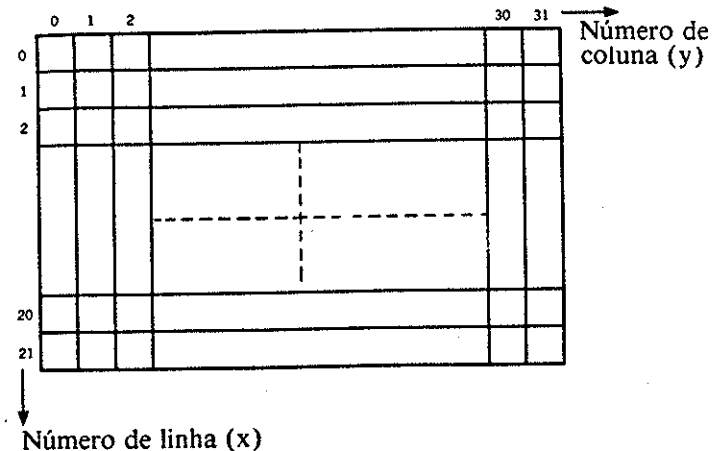


Figura 7.6 — Rede de caracteres

Pode-se imprimir um caracter, ou uma cadeia de caracteres, em posições definidas da rede usando a palavra-chave PRINT seguida pela pseudo-função AT com o formato

```
PRINT AT n.º de linha, n.º de coluna; "caracteres"
```

ou alternativamente

```
PRINT AT n.º de linha, n.º de coluna, CHR$ n.º de código
```

Para deslocar a posição PRINT de n colunas ao longo de uma dada linha a partir do lado esquerdo da rede pode-se usar a pseudo-função TAB. Esta tem o formato TAB n, onde n é um número (na gama 0 a 31) que especifica a posição onde se começa a imprimir a cadeia de caracteres.

O programa seguinte demonstra o uso de PRINT AT, TAB e CLS. Ao executar o programa observará os caracteres da figura 7.7 surgirem na parte inferior do visor e moverem-se sobre este.

```

5 FOR n = 20 TO 0 STEP - 1
10 PRINT AT n,9;"↑Para Cima↑"
20 PRINT AT n + 1,9; CHR$ 94; TAB 21; CHR$ 94

```

```

25 PAUSE 20
30 CLS
35 NEXT n

```

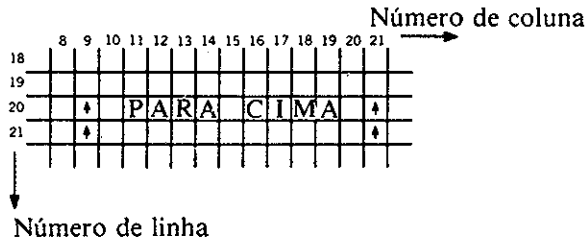


Figura 7 7 — Exemplo de impressão de caracteres

Um caracter impresso numa dada posição da rede pode ser lido pelo programa usando a instrução SCREEN\$ com o formato

SCREEN\$ n.º de linha, n.º de coluna

Por exemplo, se executarmos

```

10 PRINT AT 4,4;"A"
20 PRINT CODE SCREEN$(4,4)

```

veremos o ZX Spectrum imprimir o caracter A na posição 4,4, e o código de A (65) no início da linha seguinte do visor. Note no entanto que não pode usar a instrução SCREEN\$ para ler caracteres definidos pelo utilizador.

No caso de aplicações gráficas o "papel" é formado por uma rede de 176 × 256 elementos, conhecidos pelo nome de pixels. A rede de pixels sobrepõe-se à rede de caracteres de tal modo que cada quadrado desta contém sessenta e quatro pixels (8 × 8). As colunas na direcção X são numeradas de 0 a 255 a partir do lado esquerdo, e as linhas na direcção Y são numeradas de 0 a 175 a partir da parte inferior do visor. Este arranjo dos pixels é ilustrado na figura 7.8.

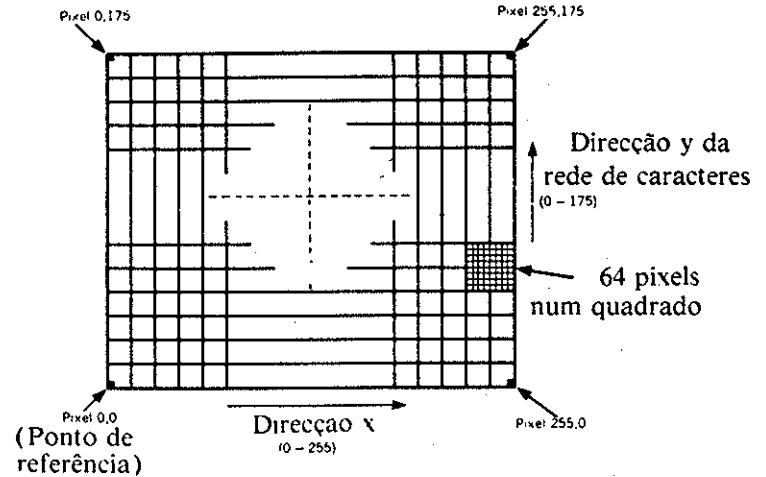


Figura 7 8 Rede de pixels gráficos

O ZX Spectrum imprime um pixel de tinta numa dada posição da rede de pixels usando a palavra-chave PLOT, seguida pelas coordenadas X e Y da rede. O formato é

PLOT x,y

Pode-se apagar um pixel de tinta usando

PLOT INVERSE I: x,y

Para examinar qualquer pixel a fim de determinar se foi impresso a cor de tinta ou de papel, usa-se a função POINT sob a forma

POINT (x,y)

e o resultado é zero quando se trata de um pixel de "papel" e 1 quando se trata de um pixel de "tinta".

Por exemplo, a linha de programa

```
15 LET x = POINT (200,100)
```

definirá a variável x igual a 0 quando pixel 200,100 for papel, e 1 no caso contrário.

Desenho de linhas e círculos

O ZX Spectrum possui declarações DRAW e CIRCLE para desenhar linhas rectas, arcos e círculos, como se mostra nos exemplos da fotografia 7.4.

Para desenhar uma linha recta a partir de um ponto de referência usa-se a declaração DRAW com o formato

DRAW h,v

onde h e v são o número de deslocamentos horizontais e verticais, em pixels, a partir do ponto de referência. O sinal de h deve ser positivo se o deslocamento for executado para a direita do ponto de referência, e negativo se o deslocamento for para a esquerda. O sinal de v deve ser positivo se o deslocamento vertical é executado para cima, e negativo no caso contrário. O ponto de referência possui as coordenadas do último ponto impresso (com PLOT, DRAW ou CIRCLE), e se nenhuma destas instruções tiver sido usada o computador considera que o ponto de referência é 0,0. Este ponto de referência é sempre repostado em 0,0 depois de qualquer das instruções NEW, RUN, CLS ou CLEAR.

Pode ser usada uma outra forma da declaração DRAW para desenhar o arco de um círculo. Neste caso a declaração assume a forma

DRAW h,v,a

onde h e v são definidos do mesmo modo que anteriormente e "a" é o ângulo do arco em radianos. Se "a" possui um valor positivo, o arco é desenhado no sentido contrário ao dos ponteiros do relógio, e se é negativo o arco é desenhado no sentido dos ponteiros do relógio.

Exercício

Execute o programa seguinte, que imprime um motivo simples em forma de rosa-dos-ventos.

```
10 PLOT 196, 125
20 DRAW - 51,0
24 PLOT 150, 125
```

```
25 DRAW 40,0, PI
26 DRAW - 40,0, PI
39 PLOT 171, 125
40 DRAW 0,27:DRAW 0, - 51
60 PRINT AT 1,21;''N''
70 PRINT AT 2,21;''↑''
```

No exercício acima, o leitor notará que o círculo do motivo impresso é desenhado usando dois arcos semicirculares (linhas 25 e 26). No entanto, existe um modo alternativo de desenhar círculos usando a instrução CIRCLE.

Esta instrução tem o seguinte formato:

CIRCLE x,y,r

onde r designa o raio em pixels. Neste programa poderia eliminar as linhas 24, 25 e 26, substituindo-as por

```
50 CIRCLE 171,125,20
```

que desenhará todo o círculo.

Note que no caso de querer utilizar instruções DRAW a seguir a uma instrução CIRCLE deve redefinir novamente o ponto de referência usando a instrução PLOT.

É possível incluir todas as declarações de atributos (INK, PAPER, BRIGHT, FLASH, OVER e INVERSE) nas instruções PLOT, DRAW ou CIRCLE, devendo aquelas ser escritas imediatamente a seguir à palavra-chave e terminadas por separadores apropriados (ponto e vírgula, vírgula).

Exercício

Escreva as seguintes duas linhas de programa

```
10 DRAW PAPER 6,44,78
20 CIRCLE INK 4; FLASH 1; 100,78,15
```

e observe o efeito resultante.

Atributos da rede de caracteres

A rede de caracteres do visor consiste na rede «de papel» que já estudámos (ver a figura 7.6) mais duas linhas adicionais, as linhas 22 e 23, cada uma delas com os habituais 32 caracteres (numerados de 0 a 31). Estas linhas adicionais formam a janela inferior da imagem, usada para imprimir as linhas de programa quando estão a ser montadas, e em geral para comunicação com o computador.

Cada um dos 768 elementos (24×32) da rede de caracteres do visor possui um atributo que define as cores do papel e da tinta do carácter impresso, e o modo de apresentação geral do carácter em causa.

Os atributos de cada carácter são armazenados em certas posições da memória RAM, entre os endereços 22528 e 23295 (chamada Ficheiro de Atributos). Cada elemento da rede de caracteres do visor utiliza um byte desta área de memória para definir as suas cores e modo de impressão (ver a fotografia 7.5 e o formato dos bytes de atributos indicado na figura 7.9).

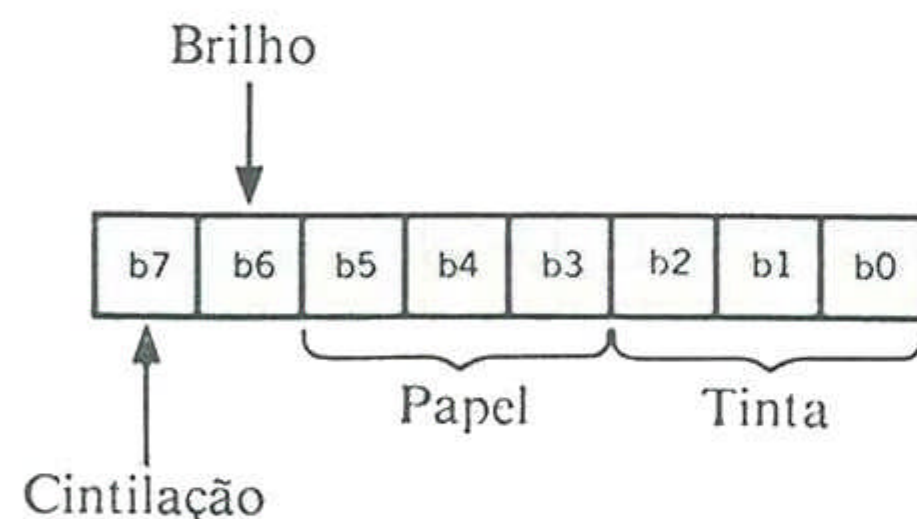
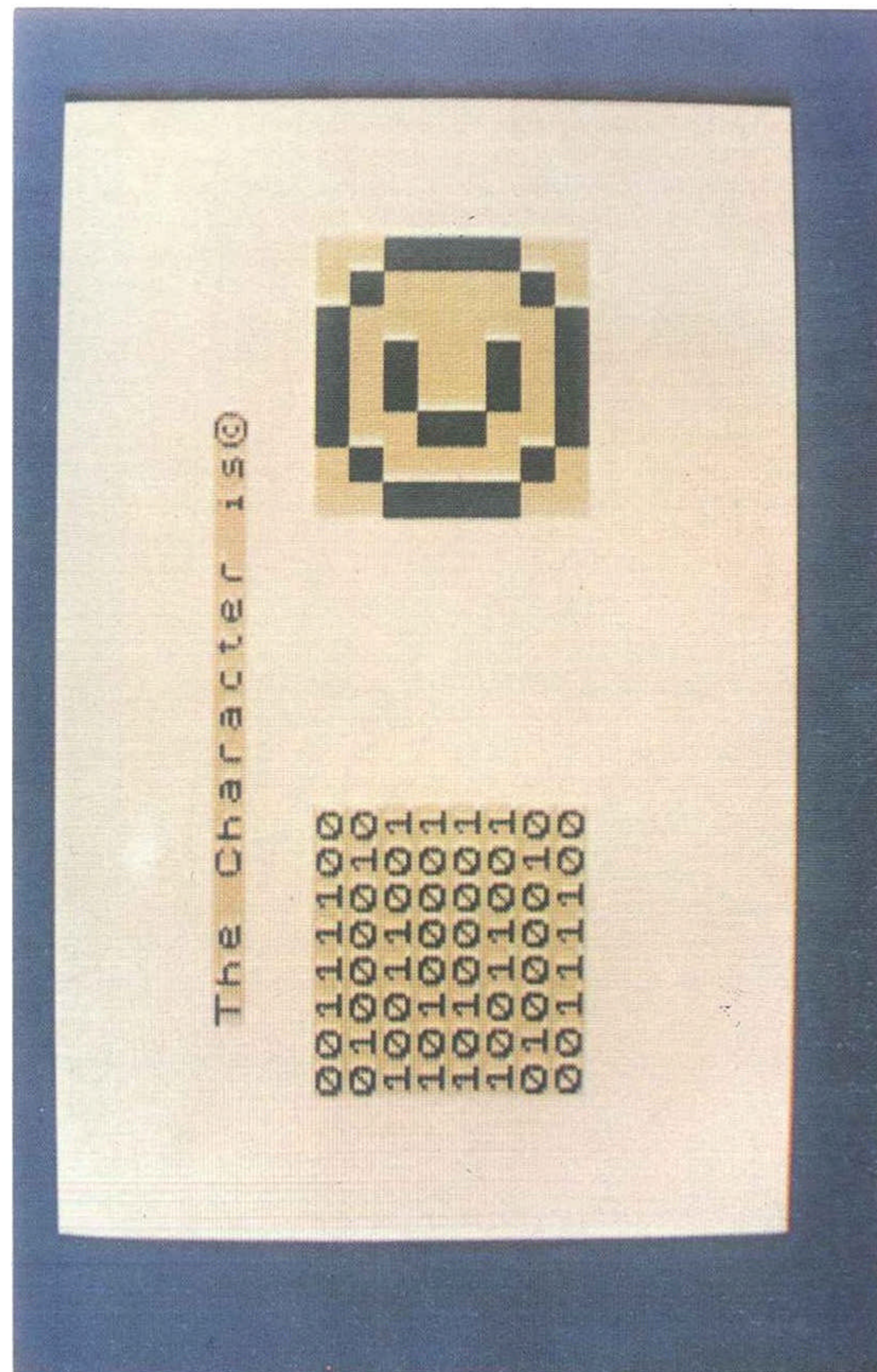
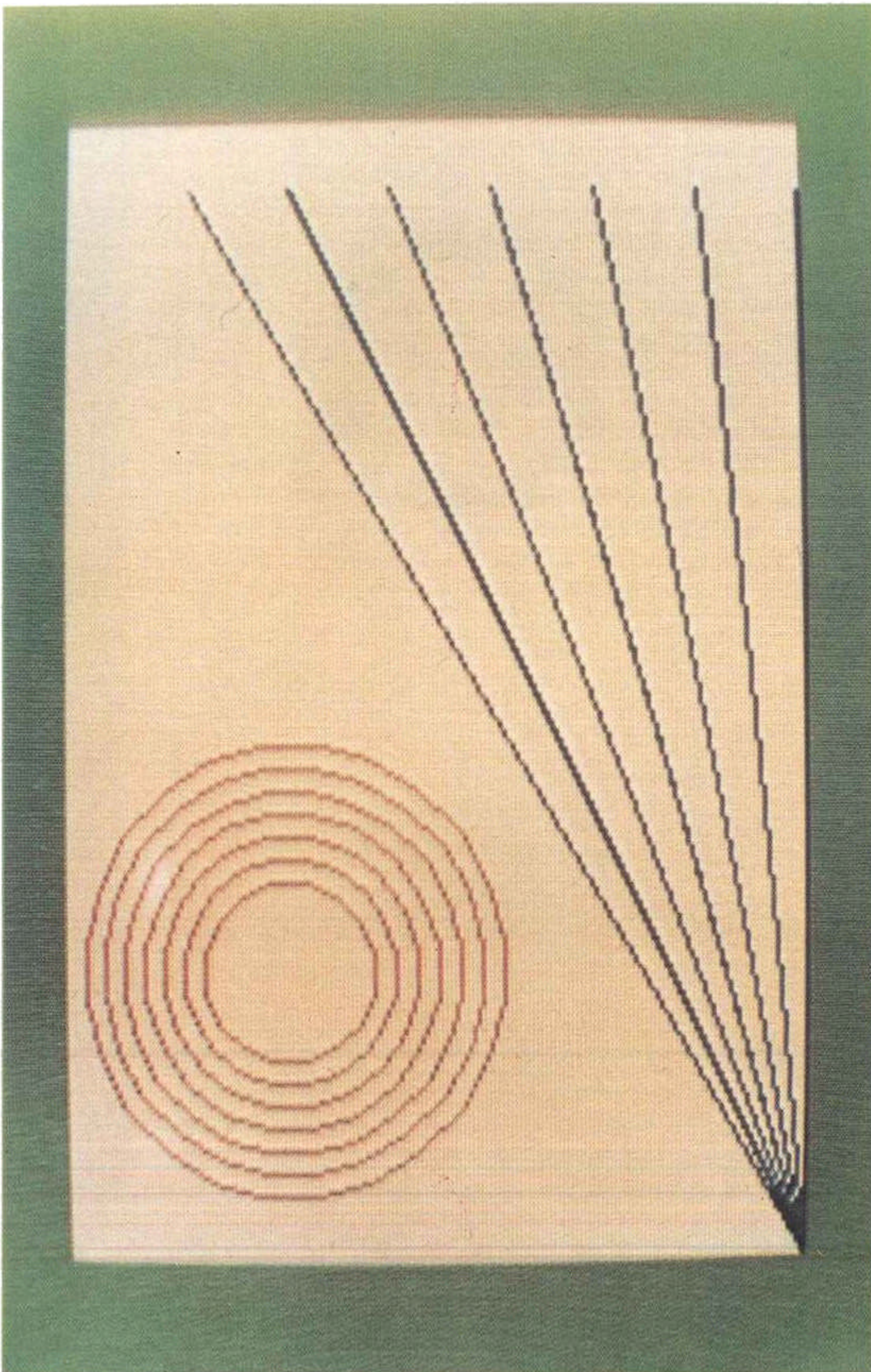


Figura 7.9 — Formato de um byte de atributos.

O bit mais significativo, b7, de um byte de atributos é usado para definir se o carácter cintila ou não (*flash*). Se este bit tiver o valor 1 o carácter cintila, se não a sua imagem é estável. O bit mais significativo a seguir, b6, é usado para definir o modo brilhante. Quando tem o valor 1 o carácter é impresso com maior brilho, e quando tem o valor zero o seu brilho é normal. Os bits b5, b4 e b3 são usados para definir a cor do papel do elemento e os bits b2, b1 e b0 definem a cor de tinta do carácter.





Exemplo do desenho de linhas e círculos.

O equivalente decimal do bit de cintilação, b7, quando ao valor 1, é 128; se este bit estiver ao valor zero o seu valor decimal será 0. O equivalente decimal do modo brilhante (b6 ao valor 1) é 64, ou 0 no caso contrário. Os bits de cor de papel possuem um equivalente decimal igual a oito vezes o número de cada cor, enquanto que os bits de cor de tinta possuem um equivalente decimal dado pelo próprio número da cor. O atributo pode portanto ter um valor equivalente decimal entre 0 e 255, que é determinado usando a expressão:

$$\text{Valor decimal} = (b7 \times 128) + (b6 \times 64) + (\text{n.º da cor de papel} \times 8) + (\text{n.º da cor de tinta})$$

O atributo de qualquer elemento da rede de caracteres do visor pode ser examinado usando a função ATTR com o formato ATTR (n.º de linha, n.º de coluna)

Por exemplo, se depois de escrevermos NEW fizermos executar as linhas de programa

```
10 PAPER 6: INK 3
20 PRINT ATTR (0,0)
```

o Spectrum imprimirá no visor o valor 51, indicando que o número da cor de papel é 6, o número de tinta é 3 e o modo de apresentação é estável e de brilho normal.

Para definir os atributos de um elemento da rede de caracteres do visor usa-se a função POKE com o formato

```
POKE endereço do ficheiro de atributos,
valor decimal dos atributos
```

onde o endereço do ficheiro de atributos está relacionado com a linha e a coluna da rede de caracteres através da expressão

$$\text{Endereço do ficheiro de atributos} = 22527 + (\text{n.º da coluna}) + ((\text{n.º da linha}) \times 32)$$

Exercício

Verifique que quando escreve

```
POKE 23231,207
```


o elemento da rede de caracteres no canto inferior direito cintila em modo brilhante com PAPER 1 (azul) e INK 7 (branco). Verifique que se escrever em seguida

```
PRINT ATTR (21,31)
```

o ZX Spectrum responde indicando os atributos correctos.

Outras observações sobre o visor

Já vimos anteriormente que é possível imprimir caracteres e pixels no visor, e as figuras 7.6 e 7.8 definem as correspondentes redes. Os bytes de dados que definem os caracteres e pixels impressos são armazenados na área da memória volátil conhecida por ficheiro de imagem. O ficheiro de imagem corresponde aos endereços entre 16384 e 22527 inclusive, mas os oito bytes de cada caracter não são armazenados sequencialmente e, por outro lado, o computador não escreve sequencialmente as 192 linhas da imagem (44 linhas de caracteres x 8 bytes/caracter = 92 linhas).

A rede de caracteres apresentada na figura 7.6, mais as duas linhas usadas para comunicação com o computador, são dispostas sob a forma de três secções de 32 colunas e 8 fiadas, como se mostra na figura 7.10. Os caracteres são escritos no visor, começando pela primeira linha de pixels da linha 0 e coluna 0 da rede de caracteres e continuando pela primeira linha de pixels de todas as colunas na linha 0 da rede de caracteres, segue-se a segunda linha de pixels de cada um dos 32 caracteres da linha 1 da rede de caracteres, etc., até ser terminado o primeiro terço, secção 0. Em seguida o computador guarda todas as primeiras linhas de pixels dos 32 caracteres da linha 8 da rede de caracteres (linha 0 da secção 1), etc. Este processo relativamente complicado de armazenar os bytes de imagem pode ser observado usando

```
20 FOR n = 16384 TO 22527
30 POKE n,255
40 NEXT n
```

Para calcular o endereço de qualquer byte do ficheiro de imagem pode-se usar

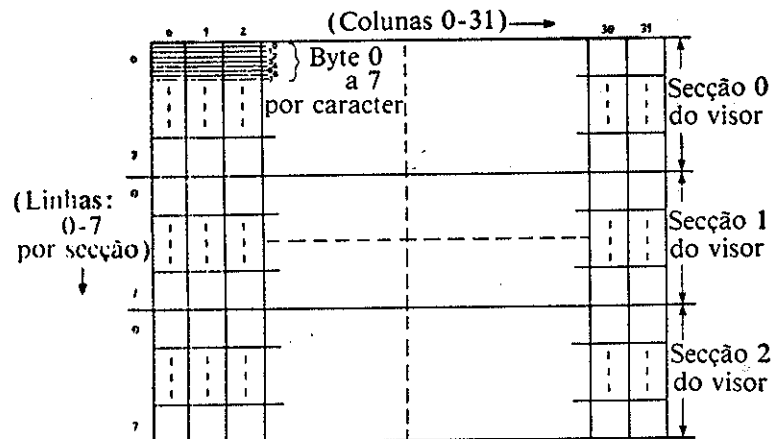


Fig. 7.10.

Endereço do ficheiro de imagem = 16384 + n.º da coluna de caracteres + (32 × n.º da linha de caracteres) + (256 × número do byte do caracter) + (2048 × n.º da secção do visor)

onde o número da coluna, o número da linha, o número do byte do caracter e o número da secção são definidos na figura 7.10. Por exemplo, o endereço do byte 6, na linha de caracteres 5, coluna 27, secção 2, será

$$16384 + 27 + (32 \times 5) + (256 \times 6) + (2048 \times 2) = 22203$$

e se o leitor fizer

```
POKE 22203,255
```

observará que a máquina imprime uma fiada de pixels no local exacto do visor. Não se esqueça de que 255 = 11111111, isto é, que um byte define oito pixels.

A possibilidade de colocar directamente, através da função POKE, valores em bytes do ficheiro de imagem é uma característica particularmente útil quando se pretende programar em código-máquina.

Movimentação de gráficos

Vimos já o modo como se pode imprimir um carácter gráfico num determinado quadrado da rede de caracteres usando PRINT AT x,y; «carácter», onde x e y representam os números de linha e coluna respectivamente. Se mantivermos x constante e incrementarmos y de 0 a 31, o carácter será impresso em todas as colunas da linha x. Por exemplo, o programa

```
5 FOR y = 0 TO 31
15 PRINT AT 10,y;"*"
25 NEXT y
```

imprime uma linha de asteriscos na linha 10. Quando se executa este programa nota-se que o carácter gráfico se desloca da esquerda para a direita, deixando porém a sua imagem ao longo do trajecto percorrido. É possível remover esta imagem alterando a linha 15 do seguinte modo:

```
15 PRINT AT 10,y;" *"
```

ou seja, incluindo um espaço imediatamente antes do asterisco. Note que o programa apresenta agora um asterisco móvel que parte da coluna 1 e termina na coluna 0 depois de se deslocar por todas as colunas.

Para inverter a direcção do movimento, digamos quando o asterisco atinge o lado direito do visor, o programa pode ser escrito de modo a detectar quando y se torna igual a um limite previamente especificado, podendo em seguida ser decrementado a fim de inverter a direcção do movimento. Para evitar que se mantenha a imagem nas posições percorridas pelo asterisco ao mover-se em sentido contrário deve-se incluir um espaço também à frente do asterisco.

Se executar o programa seguinte, o leitor verificará que o asterisco se move para diante e para trás ao longo do visor.

```
20 FOR y = 0 TO 30
25 PRINT AT 10,y;" * "
30 IF y = 30 THEN GO TO 45
35 NEXT y
45 FOR y = 30 TO 0 STEP -1
55 PRINT AT 10,y;"* "
```

```
60 IF y = 0 THEN GO TO 15
65 NEXT y
```

Para obter um movimento vertical do asterisco, mantém-se constante o parâmetro y e altera-se o parâmetro x em função do movimento requerido. Como é óbvio, é necessário incluir declarações PRINT AT que imprimam espaços acima e abaixo do asterisco de modo a eliminar as imagens que deixa atrás. Por outro lado, em algumas aplicações pode ser aconselhável comandar o movimento do carácter gráfico através do teclado. Para investigar o movimento vertical do asterisco comandado pelo teclado sugerimos ao leitor que execute o programa seguinte. Carregando nas teclas U ou D o leitor poderá deslocar o asterisco para cima e para baixo respectivamente entre os limites superior e inferior do visor.

```
15 LET x = 10
25 LET y = 15
35 LET K$ = INKEY$
45 IF K$ = "U" THEN LET x = x - 1
50 IF x < 0 THEN LET x = 0
55 IF K$ = "D" THEN LET x = x + 1
60 IF x > 20 THEN LET x = 20
75 PRINT AT x - 1,y;" "
85 PRINT AT x,y;" * "
95 PRINT AT x + 1,y;" "
115 GO TO 35
```

Para estudar as potencialidades de uso de gráficos do ZX Spectrum um pouco melhor, consideremos agora o modo de obter um movimento do asterisco numa trajectória aproximadamente circular. Neste caso será necessário avaliar os parâmetros x e y para cada posição de impressão. No programa que se segue as coordenadas da rede de caracteres são avaliadas nas linhas 45 e 55, sendo o asterisco impresso em seguida na posição apropriada. Note como são usadas duas instruções LET nas linhas 36 e 37 para guardar a posição de impressão anterior, tornando assim possível apagá-la na linha 58. Execute o programa e veja como o asterisco se move sobre o visor.

```

5 LET H = 0
7 LET U = 0
35 FOR A = 0 TO (2 * PI) STEP PI/8
36 LET M = H
37 LET N = U
45 LET H = 15 + INT (8 * COS A)
55 LET U = 10 + INT (8 * SIN A)
58 PRINT AT N,M: " "
65 PRINT AT U,H: " * "
75 NEXT A
95 GO TO 35

```

No programa 5 do Apêndice A apresenta-se um jogo simples utilizando os princípios discutidos.

É também possível produzir movimentos tintando e apagando pixels. Por exemplo, o programa seguinte pode ser usado para imprimir um único pixel que viaja continuamente para a frente e para trás na horizontal ao longo do visor.

```

3 BORDER 2: PAPER 6: INK 0
5 LET y = 85
10 FOR x = 0 TO 255
15 INVERSE 0 : PLOT x,y
18 PAUSE 2
20 INVERSE 1 : PLOT x,y
35 NEXT x
45 FOR x = 255 TO 0 STEP - 1
55 INVERSE 0 : PLOT x,y
65 PAUSE 2
75 INVERSE 1 : PLOT x,y
95 IF x = 0 THEN GO TO 5
100 NEXT x

```

Note que o pixel é impresso usando as instruções INVERSE 0 e PLOT x,y (linhas 15 e 55), e que é apagado usando as instruções INVERSE 1 e PLOT x,y (ver as linhas 20 e 75). As declarações PAUSE das linhas 18 e 65 são necessárias para evitar o apagamento indesejado da imagem — verifique este facto removendo estas duas linhas e observando o resultado.

Exercício

Execute o programa seguinte e veja como o pixel se desloca verticalmente no visor.

```

3 BORDER 3: PAPER 4: INK 2
5 LET x = 128
10 FOR y = 0 TO 175
15 INVERSE 0 : PLOT x,y
18 PAUSE 2
20 INVERSE 1 : PLOT x,y
35 NEXT y
45 FOR y = 175 TO 0 STEP - 1
55 INVERSE 0 : PLOT x,y
65 PAUSE 2
75 INVERSE 1 : PLOT x,y
95 IF y = 0 THEN GO TO 5
100 NEXT y

```

Observações finais

Neste capítulo estudámos as potencialidades em termos de gráficos coloridos do ZX Spectrum, e vimos como este armazena e imprime caracteres gráficos no visor. Tratámos ainda os gráficos definidos pelo utilizador, a movimentação de gráficos, e o modo de desenhar círculos e linhas. Todas estas características deverão ser bastante úteis no desenvolvimento de programas para aplicações envolvendo gráficos a cores.

CAPÍTULO VIII

SONS

Introdução

O ZX Spectrum pode produzir sons, o que o leitor pode constatar sempre que carrega numa tecla. Se verificar que ao fazê-lo o nível sonoro é muito baixo, pode alterá-lo intervindo na variável de sistema PIP, no endereço 23609, e colocando nela um valor apropriado recorrendo à instrução POKE:

POKE 23609, inteiro na gama 0 a 255

Sugerimos-lhe que investigue isto usando vários valores diferentes para a variável PIP (ver Apêndice C).

Consideremos agora o modo de programar um gerador de sons capaz de produzir notas com uma dada duração e tonalidade

Produção de sons

Pode-se programar o ZX Spectrum para produzir sons criados electronicamente com uma dada tonalidade e duração, usando à declaração BEEP sob a forma

BEEP duração, tonalidade

onde a duração e a tonalidade podem ser expressões numéricas ou valores constantes, inteiros ou não. A duração deve ser expressa em segundos, e a tonalidade, a frequência da nota produzida, é expressa como o número de meios-tons relativamente ao Dó normal; a tonalidade é portanto considerada zero para a frequência de referência de 216,6 Hz da escala temperada (definida na

Conferência Internacional de Londres de Maio de 1939). Nesta escala um meio-tom é o intervalo entre duas frequências adjacentes, tais que a frequência f_n está relacionada com o meio-tom seguinte, f_{n+1} , por

$$f_{n+1} = \sqrt[12]{2} f_n = 1,0594631 f_n$$

Por uma questão de conveniência, quando se usa a declaração BEEP identifica-se a frequência do meio-tom como equivalendo a um certo número de meios-tons de distância relativamente ao Dó central. Os valores de tonalidade positivos correspondem a meios-tons mais altos do que o Dó central, e os negativos a meios-tons mais baixos. Na tabela 8.1 listam-se 27 meios-tons, o correspondente valor de tonalidade no ZX Spectrum e a frequência real da nota. Observando esta tabela verifica-se que após 12 meios-tons, ou seja uma oitava, a frequência das notas é duplicada.

Tabela 8.1 — Escala temperada de notas

Nota musical	Valor de tonalidade no ZX Spectrum	Frequência da nota (Hz)
Dó	— 12	130,8
Dó#, Réb	— 11	138,6
Ré	— 10	146,8
Ré #, Mi _b	— 9	155,6
Mi	— 8	164,8
Fá	— 7	174,6
Fá #, Sol _b	— 6	185,0
Sol	— 5	196,0
Sol #, Lá _b	— 4	207,7
Lá	— 3	220,0
Lá #, Si _b	— 2	233,1
Si	— 1	246,9
Dó	0	261,6
Dó #, Ré _b	1	277,2
Ré	2	293,7

Ré #, Mi _b	3	311,1
Mi	4	329,6
Fá	5	349,2
Fá #, Sol _b	6	370,0
Sol	7	392,0
Sol #, Lá _b	8	415,3
Lá	9	440,0
Lá #, Si _b	10	466,2
Si	11	493,2
Dó	12	523,2
Dó #, Ré _b	13	554,4
Ré	14	587,4

O programa simples apresentado a seguir produzirá a escala de duas oitavas, começando uma oitava abaixo da central e terminando uma oitava acima do Dó central.

```

3 PRINT "Indique duração do BEEP"
4 INPUT y
5 FOR x = - 12 TO 12
15 BEEP y,x
25 NEXT x

```

Sugerimos-lhe que execute este programa com um valor de duração na gama 0,0015 segundos a 2 segundos.

Verificará que é interessante alterar a tonalidade em função duma expressão matemática. Por exemplo, o programa listado a seguir utiliza a expressão $(\sin x + \cos x)x$ para definir o valor da duração.

```

3 PRINT "Indique duração do BEEP"
4 INPUT y
5 FOR x = - 12 TO 12
15 BEEP y, (SIN x + COS x) *x
25 NEXT x
30 GO TO 5

```

Sugerimos que execute este programa com valores de duração de 0,035 segundos. Para sair do programa carregue simplesmente em CAPS SHIFT e BREAK.

Para introduzir uma *pausa* na sequência de notas musicais pode utilizar a instrução com o mesmo nome, PAUSE, que tem o formato

PAUSE n

onde n é um inteiro na gama 0 a 65535. Se $n = 0$, a pausa dura até se carregar em qualquer tecla; se n for diferente de zero a pausa dura $n/50$ segundos. Consequentemente, a pausa máxima que é possível obter usando esta instrução é 21,845 minutos.

É interessante inserir a instrução

18 PAUSE ABS x + 1

no programa anterior, com um valor de 0,035 segundos.

Exercício

Experimente várias expressões matemáticas na definição do valor de tonalidade na declaração BEEP da linha 15 do programa anterior. Investigue por outro lado o efeito da alteração do valor da duração.

Ao realizar o exercício referido, o leitor poderá produzir as suas próprias sequência sonoras. No entanto, poderá querer traduzir música publicada de tal modo que o Spectrum possa "tocar" essas melodias. Para o fazer deve compreender os rudimentos da escrita musical, e na secção seguinte daremos uma breve introdução a este assunto. Se sabe ler música talvez seja melhor passar directamente ao exemplo ilustrativo.

Rudimentos de música

A música para piano, violino, viola, etc., é muitas vezes escrita na clave de Sol. O símbolo correspondente é colocado no lado esquerdo das cinco linhas a que se chama "pauta". A clave de Sol rodeia a segunda linha da pauta (a contar de baixo), e é isto que define a posição de todas as notas; a nota Sol será sempre representada na segunda linha (ver figura 8.1).

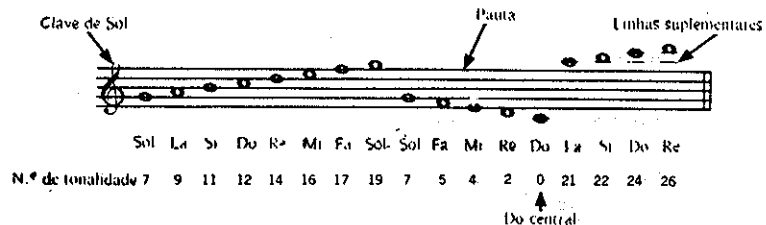


Figura 8.1 — Notas musicais correspondentes aos valores de tonalidade do ZX Spectrum

A pauta é dividida por barras verticais, formando "compassos" com uma duração igual; esta duração é especificada pelo número fraccionário inserido no início da pauta, a seguir à clave. Este número indica o número de notas de valor igual em cada compasso, indicando o valor de cima o número de batimentos em cada compasso e o de baixo o tipo de nota que corresponde a um batimento. Por exemplo, o compasso 4/4 terá quatro "tempos" (batimentos) por compasso, e cada batimento corresponde a uma "semínima".

Existem vários tipos de notas, cada uma das quais representa uma certa duração. Na figura 8.2 mostram-se a semibreve, a mínima, a semínima e a colcheia num compasso 4/4. O número de batimentos por cada semibreve é quatro, por cada mínima é dois, por cada semínima é 1, sendo igualmente de 1 por cada grupo de duas colcheias. O símbolo $\overline{\sim}$ escrito sobre uma nota indica que esta deve ser prolongada.



Figura 8.2 — Notas musicais

Para indicar que a tonalidade de uma nota é "alterada", usam-se símbolos apropriados antes da nota:

- 1) O símbolo de "sustenido", #, indica que a tonalidade da nota deve ser aumentada de um meio-tom;
- 2) O símbolo de "bemol", b, indica que a tonalidade deve ser diminuída de meio-tom;
- 3) O símbolo "bequadro", ♯, anula as anteriores alterações da nota;
- 4) O "sustenido duplo", ×, aumenta a tonalidade de um tom inteiro;
- 5) O "bemol duplo", bb, baixa a tonalidade de um tom inteiro.

As pausas indicam durações iguais às das notas que lhes dão o nome. Na figura 8.3 apresentam-se as pausas de semibreve, mínima, semínima, colcheia e semi-colcheia.



Figura 8.3 — Pausas musicais

Um ponto colocado à frente de uma nota ou pausa ("ponto de aumento") aumenta a duração destas de metade do seu valor. Na figura 8.4(a) apresentam-se alguns exemplos. Dois pontos colocados à frente e atrás de uma barra indicam que a música entre barras deve ser repetida, como acontece na figura 8.4(b). O sinal Del Segno (D.S.) indica que se deve voltar ao sinal "C" e tocar até ao fim, e o sinal Da Capo (D.C.) significa que se deve voltar ao princípio e tocar até ao fim.

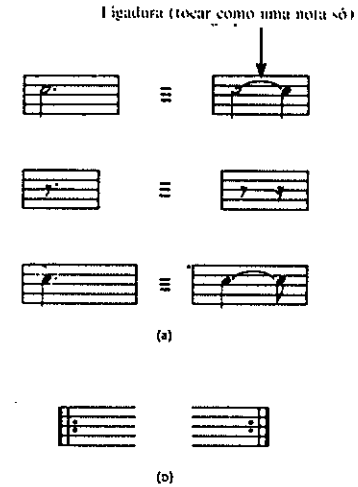


Figura 8.4(a) — Pontos de aumento
(b) — Pontos que indicam repetição

Todas as claves possuem dois nomes, conforme a música e em modo maior ou menor. O modo da música é indicado pelo número de sustenidos ou bemóis no seu início (ver a figura 8.5).

Figure 8.5 displays two staves of musical notation. The top staff contains notes with the following mode names: Dó maior, Ré maior, Mi maior, Fá maior, Lá maior, Si maior, and Fá# maior. The bottom staff contains notes with the following mode names: Ré menor, Mi menor, Fá menor, Lá menor, Si menor, Dó# maior, and Ré# menor.

Figura 8.5 — Claves e modos

Utilizemos agora alguns destes rudimentos no exemplo seguinte.

Exemplo ilustrativo

Na figura 8.6 mostramos a música e os correspondentes valores de tonalidade no ZX Spectrum para uma melodia bem conhecida. O programa seguinte produz os sons correspondentes. O leitor notará que o valor da duração é indicado pelo utilizador, e que depois de este valor ser indicado a música é imediatamente executada. Sugerimos-lhe que utilize uma duração de 0,3 segundos para começar, experimentando depois outros valores.

Figure 8.6 displays two staves of musical notation. The top staff has pitch numbers: 5, 5, 12, 12, 14, 14, 12, 10, 10, 9, 9, 7, 7, 5, 12, 12, 10, 10, 9, 9, 7. The bottom staff has pitch numbers: 12, 12, 10, 10, 9, 9, 7, 5, 5, 12, 12, 14, 14, 12, 10, 10, 9, 9, 7, 7, 5.

Figura 8.6

```

5 PRINT "Twinkle, Twinkle, little Star"
10 INPUT d:PRINT d:
15 BEEP d,5:BEEP d,5:BEEP d,12:BEEP d,12
20 BEEP d,14:BEEP d,14:BEEP 2 * d,12
25 BEEP d,10:BEEP d,10:BEEP d,9:BEEP d,9
30 BEEP d,7:BEEP d,7:BEEP 2 * d,5
35 BEEP d,12:BEEP d,12:BEEP d,10:BEEP d,10
40 BEEP d,9:BEEP d,9:BEEP 2 * d,7
44 BEEP d,12:BEEP d,12:BEEP d,10:BEEP d,10
47 BEEP d,9:BEEP d,9:BEEP 2 * d,7
50 BEEP d,5:BEEP d,5:BEEP d,12:BEEP d,12
55 BEEP d,14:BEEP d,14:BEEP 2 * d,12
60 BEEP d,10:BEEP d,10:BEEP d,9:BEEP d,9
65 BEEP d,7:BEEP d,7:BEEP 2 * d,5

```

Amplificador de som

O sinal sonoro do gerador de sons do ZX Spectrum é enviado para as tomadas de MIC e EAR. Assim, utilizando uma ficha tipo "jack" de 3,5 mm, é possível conduzir este sinal à entrada de um amplificador de som. Na figura 8.7 indicamos o circuito de amplificador apropriado. O circuito em causa possui um comando de volume (potenciômetro de 10 kohms) e utiliza um amplificador integrado bastante barato, tipo LM380. Na figura 8.8 ilustramos a distribuição dos pinos deste circuito.

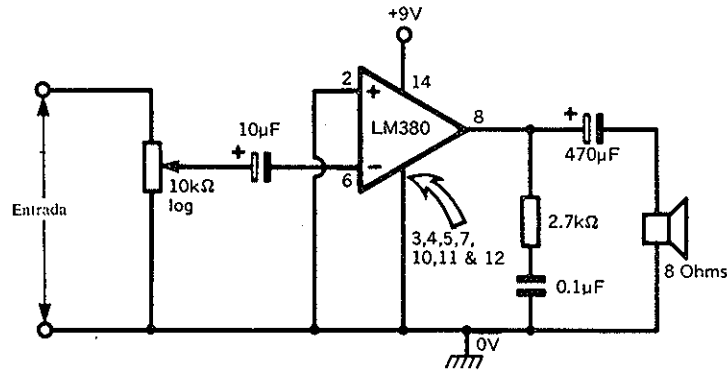


Figura 8.7 — Circuito amplificador de som

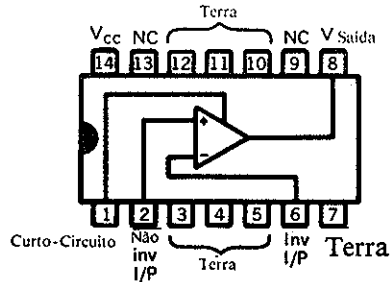


Figura 8.8 — Distribuição dos pinos do amplificador integrado LM380

Observações finais

Neste capítulo introduzimos os conceitos básicos da produção de sons usando a declaração BEEP, e mostrámos como é possível traduzir música para a linguagem compreendida pelo ZX Spectrum. Isto permite-nos incluir instruções de produção de sons nos nossos programas. Como exemplo consulte o programa 5 do Apêndice A.

Considerámos apenas a produção de sons usando a declaração BEEP, mas é possível produzir sons usando código-máquina; este assunto será considerado mais detalhadamente no Capítulo 10.

CAPÍTULO IX

HARDWARE

Introdução

O computador ZX Spectrum contém um certo número de circuitos integrados ligados entre si que fazem parte do "hardware" do sistema. Na figura 9.1 mostra-se um simples diagrama de blocos do computador. O sistema deste consiste numa unidade microprocessadora (MPU), na memória necessária para armazenar as instruções e dados, e em certas ligações de entrada/saída.

O microprocessador é o coração de todo o sistema, consistindo o seu papel em executar operações matemáticas ou lógicas de acordo com o programa de instruções que lhe foi fornecido. São usados dois tipos de memória; a memória volátil, de acesso aleatório (RAM), para armazenamento do programa, de dados, de variáveis de sistema, etc.; e memória fixa, apenas de leitura (ROM), com um comprimento de 16 K, onde se guarda o sistema operativo e o interpretador Basic do ZX Spectrum.

A informação é comunicada ao microcomputador usando o teclado, o gravador de cassettes, o microdrive e um interface. As informações de saída do microcomputador podem ser conduzidas para o gravador de cassettes, para uma impressora, para um receptor de televisão através de um codificador PAL e um modulador UHF/VHF, para um microdrive, um interface, o circuito sonoro e um altifalante.

Para assegurar que o sistema operativo funciona correctamente sob o comando do utilizador ou de um programa, são necessários extensos circuitos de tempo e de comando. É evidente que a unidade ULA (Uncommitted Logic Array) da Ferranti (tipo ULA5C102E no Spectrum de 16 K e ULA5C112E no Spectrum

de 48 K) foi concebida de modo a produzir a temporização e o comando adequados do *hardware*. Se bem que os fabricantes não forneçam indicações completas sobre este circuito lógico integrado, sabe-se pelo menos que serve para:

- selecção de circuitos e descodificação de endereços em RAM e ROM;
- temporização e comando do microprocessador Z80A;
- leituras do teclado;
- produção de sinais de comando para condicionamento e transferência de informação de entrada e saída.

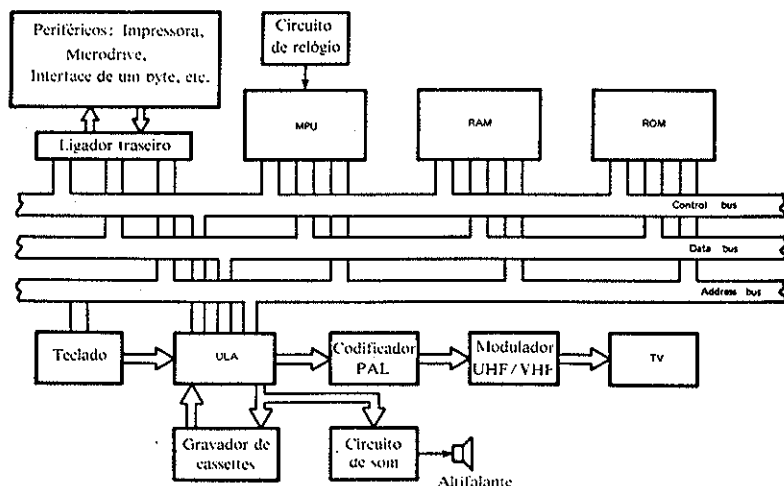


Figura 9.1 — Simple diagrama de blocos do ZX Spectrum

Vamos considerar agora alguns conceitos dos sistemas digitais que nos ajudarão a compreender o microprocessador Zilog Z80A utilizado pelo ZX Spectrum. Este conhecimento dos sistemas digitais ser-nos-á igualmente útil quando quisermos saber como ligar aparelhagens externas ao nosso microcomputador.

Flip-Flops, Flags, Registos e Contadores

Os *flip-flops* são elementos bastante importantes num computador e podem ser usados como células de memória para armazenar algarismos binários (bits), aceitando valores lógicos 0 ou 1.

O simples *flip-flop* SR (Set-Reset; *set* coloca-o ao nível lógico 1, *reset* passa-o ao nível lógico 0) pode ser construído usando portas NAND ou NOR. O símbolo lógico e a construção básica de uma porta NAND são apresentados na figura 9.2, e podemos ver nela que quando a entrada Set é passada ao valor lógico 0 a saída Q passa para o valor lógico 1, e correspondentemente \bar{Q} passa a 0. As saídas mantêm-se a estes níveis lógicos mesmo depois de a entrada Set voltar ao valor 1, e só voltarão respectivamente aos valores 0 e 1 quando for apresentado um valor 0 na entrada Reset. Este tipo de flip-flop possui valores indeterminados nas saídas quando as entradas S e R se encontram a 0, pelo que este estado é em princípio evitado por concepção do circuito.

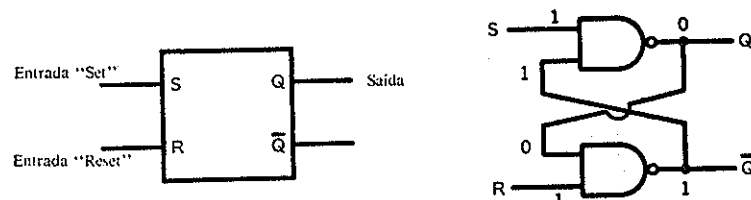


Figura 9.2 — Símbolo e porta NAND correspondentes ao *flip-flop* Set-Reset

O *flip-flop* é muitas vezes usado em aplicações que requerem apenas a existência da saída Q, e nestes casos o *flip-flop* é considerado como "flag" ("bandeira") de estado, ou seja como um indicador de estado, e a saída Q é chamada saída de *flag*.

As *flags* são usadas para indicar a ocorrência de acontecimentos relevantes no microprocessador. Por exemplo, se a execução de uma operação aritmética tem como consequência a obtenção de um valor igual a zero, tal facto provoca a passagem de uma *flag* ao valor 1. Se o resultado não é zero a *flag* é limpa ("cleared" ou "reset"). A *flag* aqui referida é chamada *flag Z* (Z de

zero) e constitui um dos indicadores de estado contidos num grupo de bits que formam o byte a que se chama Registo F (F de Flag).

O *flip-flop* de tipo D é semelhante ao *flip-flop* SR mas possui uma entrada de dados (D), uma entrada de relógio (C) e duas saídas Q e \bar{Q} . O valor binário (dado) a armazenar é aplicado à entrada D, e quando é recebido o bordo positivo de um impulso de relógio na entrada respectiva o valor binário presente na entrada D é armazenado no *flip-flop* e aparece na saída Q. O valor inverso (negado) da entrada D é aplicado à saída \bar{Q} . O símbolo deste tipo de *flip-flop* é apresentado na figura 9.3.

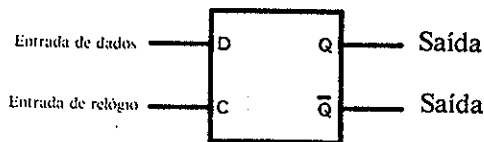
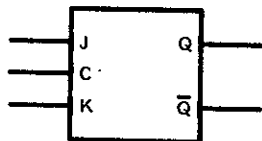


Figura 9.3 — Símbolo de um *flip-flop* de tipo D

O *flip-flop* J-K temporizado é semelhante ao *flip-flop* SR, pois possui duas entradas, J e K, que comandam o estado do *flip-flop* juntamente com um impulso de relógio que lhe é aplicado. A condição de entrada $J = K = 1$ leva a saída Q a mudar de estado ao receber cada um dos bordos positivos do impulso de relógio. O símbolo e a tabela de função de um *flip-flop* J-K são indicados na figura 9.4.



Entradas		Saída
J	K	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	\bar{Q}_n

Onde Q_n = estado de *flip-flop* antes do impulso de relógio, e Q_{n+1} = estado do *flip-flop* depois do impulso de relógio

Figura 9.4 — Símbolo e tabela de função de um *flip-flop* J-K temporizado

Vimos já que um *flip-flop* pode armazenar um bit de informação. Quando os *flip-flops* são organizados de modo a guardarem uma palavra binária o arranjo é designado por "registo". Se os dados são introduzidos e lidos simultaneamente de todos os *flip-flops*, o registo é considerado de entrada e saída "em paralelo", enquanto que no caso de os dados serem introduzidos um bit de cada vez e lidos simultaneamente o registo é considerado de entrada serial e saída em paralelo.

Na figura 9.5 mostra-se um registo de entrada e saída em paralelo, construído usando *flip-flops* de tipo D. Quando a linha de "escrita" no registo recebe um impulso positivo a palavra de entrada com oito bits é armazenada nos *flip-flops*. É incluída uma porta AND de duas entradas; uma entrada de cada porta corresponde à saída Q do correspondente *flip-flop* de tipo D, sendo a outra entrada da porta ligada à linha de escrita. Consequentemente, a verdadeira palavra de saída só surgirá nas linhas de saída de dados quando for aplicado um sinal de leitura.

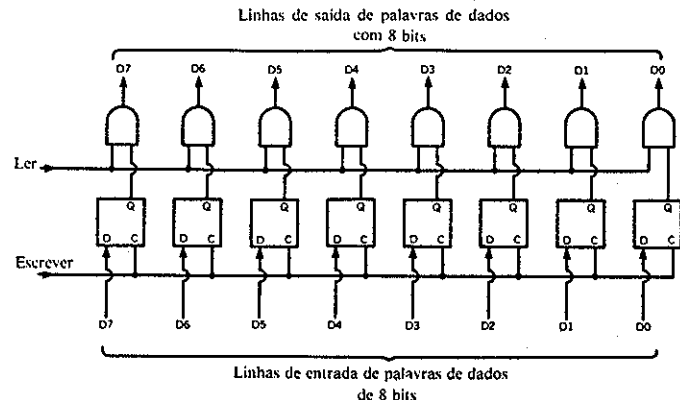


Figura 9.5 — Um registo em paralelo de oito bits

Quando é necessário manipular dados em forma serial utiliza-se um registo de deslocamento. A figura 9.6 ilustra um registo de deslocamento de oito bits construído com oito *flip-flops* J-K. O primeiro bit de dados é transferido para o *flip-flop* da esquerda quando é recebido o primeiro bordo negativo de impulso de relógio.

gio. Em cada bordo negativo sucessivo deste impulso de relógio cada um dos bits de dados é deslocado para a direita, passando ao flip-flop seguinte, e o flip-flop da esquerda recebe um bit novo. São necessários portanto oito impulsos de relógio para armazenar uma palavra serial de oito bits neste registo. Se a saída dos oito flip-flops for lida apenas ao fim de oito impulsos, tratar-se-á de uma saída em paralelo. Este é um método de conversão de serial para paralelo.

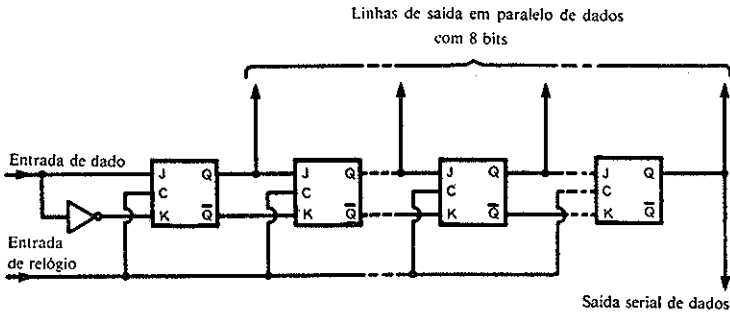


Figura 9.6 — Registo de deslocamento de oito bits construído usando *flip-flops* J-K

Vale a pena notar que o microprocessador Z-80 do ZX Spectrum contém vários registos de oito bits e dezasseis bits para fins especiais, que estudará melhor no capítulo 10.

Os *flip-flops* podem ser ligados entre si de modo a formarem diferentes tipos de contadores. O *flip-flop* J-K está particularmente adaptado a esta aplicação. A figura 9.7 mostra como é possível ligar quatro *flip-flops* J-K de modo a formar um contador binário entre 0000 e 1111. Podemos considerar a saída de cada *flip-flop* como tendo um valor calibrado: o primeiro (o da esquerda) possui um calibre 1, o segundo 2, o terceiro 4 e o último 8. Este contador aceita os sucessivos impulsos de entrada aumentando a contagem, e quando recebe um novo impulso depois de atingir o valor 1111 volta a 0000.

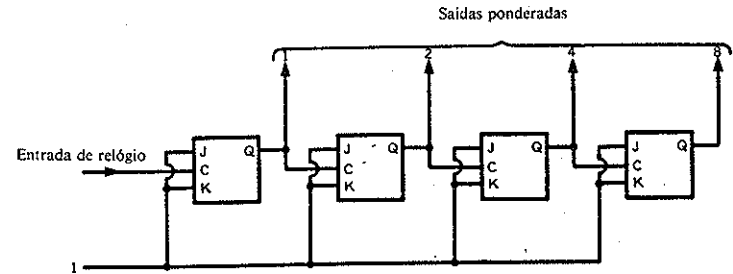


Figura 9.7 — Contador binário de quatro andares

Acrescentando outros *flip-flops* pode-se aumentar o valor máximo atingido pela contagem. É fácil ver que um contador binário formado por N *flip-flops* poderá contar $2^N - 1$ impulsos antes de voltar a 0.

O Z80A contém um Contador de Programa de 16 bits onde é guardado o endereço da instrução que deve ser executada a seguir. O contador pode ser carregado previamente com um dado valor, e o seu conteúdo modifica-se durante a execução.

ROM (Read Only Memory)

A memória ROM é usada no ZX Spectrum para armazenar o interpretador Basic e o sistema operativo. Esta ROM guarda informações sob a forma de palavras binárias (ver o Capítulo 2), encontrando-se cada palavra binária numa posição de memória endereçável, e depois de programada o seu conteúdo não pode ser modificado. Por outro lado, como a ROM é estática, não é necessário "refrescá-la", isto é, repor continuamente em cada byte o valor adequado; o conteúdo dos bytes nunca se altera. Note por outro lado que ao remover-se a alimentação o conteúdo da ROM não é perdido, pelo que esta memória é considerada não-volátil.

A ROM pode ser considerada como um quadro de posições de memória equivalentes a um bit arranjadas em grupos de oito bits (cada um deles designado por byte) formando posições de memória endereçáveis como se mostra na figura 9.8. Cada posição de memória é programada pelo fabricante durante o processo de

fabrico, de acordo com os requisitos do cliente, de tal modo que cada bit fica permanentemente com o valor 0 ou 1. A pastilha integrada que forma a ROM contém ainda um decodificador de endereços, e alguma lógica de comando e de interface. Trata-se de um dispositivo programável por máscara com 16384 palavras de oito bits, funcionando com uma alimentação de 5 V e estando organizada por bytes. Na figura 9.9(a) mostra-se um diagrama de blocos da ROM HN1312P, sendo a distribuição dos pernes indicada na figura 9.9(b).

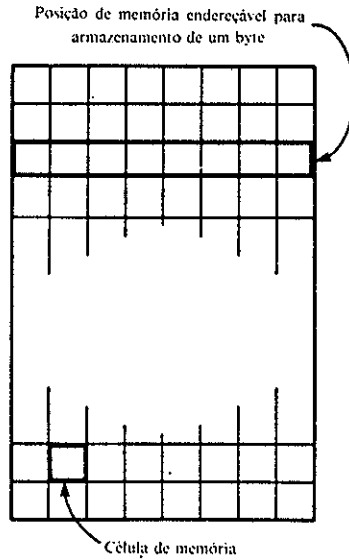


Figura 9.8 — Quadro de posições de memória

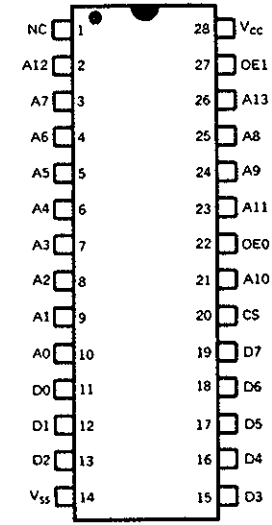
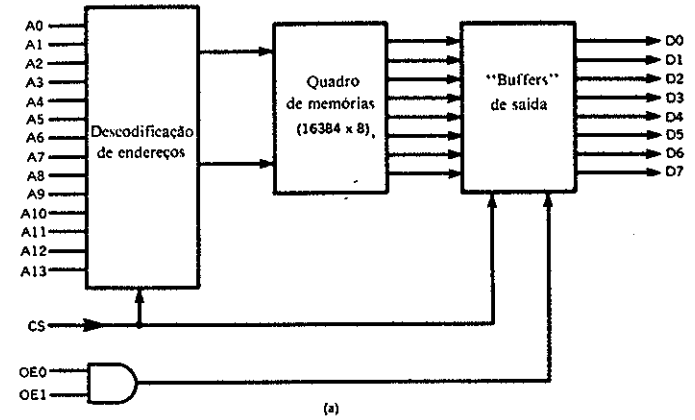


Figura 9.9(a) — Diagrama de blocos da ROM HN613128P de 16 K
(b) — Distribuição dos pernes

Existem oito ligações de saída de dados unidireccionais, D0 a D7, que são usadas para enviar para o "bus" (ligador) de dados o conteúdo de uma posição de memória endereçada numa operação de leitura. Possui catorze ligações ao "bus" de endereços. A0 a A13, sendo portanto possível aceder (tempo máximo de acesso 250 ns) qualquer das 16384 (2^{14}) palavras de que dispõe. A entrada de selecção, CS, quando activa (o nível "activo" é definido pelo utilizador) permite accionar a lógica de descodificação de endereços e passa o circuito ao seu estado de funcionamento normal (normalmente 50 mV) a partir do seu estado "inactivo" (normalmente μ W). As duas entradas de comando das saídas, OE0 e OE1 (os respectivos níveis activos são também definidos pelo utilizador) são usadas para accionar os *buffers* de saída.

No capítulo 7 descrevemos a forma dos caracteres armazenados na ROM, e no Capítulo 2 explicámos o modo de examinar o conteúdo de qualquer posição de memória. Convirá recordar que o programa 4 do Apêndice A pode ser usado para imprimir no visor todo o conteúdo da ROM.

RAM dinâmica

A memória dinâmica de acesso aleatório (RAM) é usada pelo ZX Spectrum para armazenar o programa, os dados e as variáveis de sistema. Armazena esta informação sob a forma de palavras binárias (ver Capítulo 2), sendo cada uma destas palavras armazenada numa posição de memória endereçável. Esta forma de memória é *volátil*, isto é, os dados só se manterão nela enquanto a alimentação não for anulada. No Spectrum de 16 K, para armazenar um byte de informação o computador utiliza oito 16384×1 bit pastilhas integradas de RAM dinâmica, tipo μ PD416, 4116 ou equivalente. Isto implica que cada uma das 16384 células de memória de um bit, que estão arrançadas sob a forma de uma matriz de 128 linhas por 128 colunas, tenha um endereço único; a escolha de um byte de memória na RAM é conseguida fornecendo o endereço e sinais de comando apropriados às oito pastilhas integradas simultaneamente. A figura 9.10(a) mostra o diagrama de blocos de uma pastilha RAM dinâmica de 16 K, e a figura 9.10(b) indica o arranjo dos seus pines. O dispositivo possui sete entradas de endereço, A0 a A6, e dois sinais de comando, RAS e CAS. No início

de um ciclo de leitura/escrita em memória, os bits de endereçamento do microprocessador, A0 a A6, são accionados e temporizados por RAS de modo a seleccionarem uma das 128 linhas de células de memória. Este sinal de temporização inicia por outro lado a contagem que acciona os 128 amplificadores de coluna. Depois de uma pausa previamente determinada o endereço da linha é eliminado e os sete bits de endereço de ordem superior, A7 a A13, são aplicados às sete entradas RAM A0 a A6. Esta parte do endereço é em seguida temporizada e enviada para o dispositivo sob o comando do sinal CAS. Assim, dois dos 128 amplificadores de coluna são escolhidos por A1 a A6, e A0 escolhe um entre dois "buses" de dados. Termina então a operação de escolha do bit; em seguida pode-se escrever ou ler dados para a célula de memória considerada.

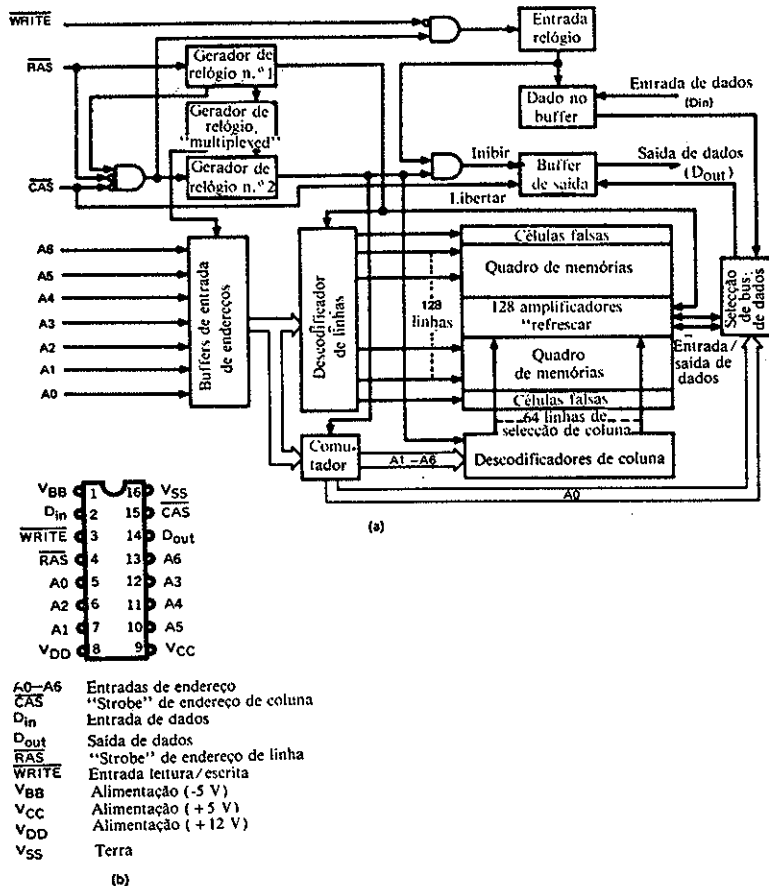


Figura 9.10(a) — Diagrama de blocos da RAM dinâmica de 16 Kx1 bit
 (b) — Distribuição dos pinos do circuito integrado de RAM dinâmica

As pastilhas de circuito RAM usadas no ZX Spectrum são dinâmicas, pelo que a informação em cada célula é armazenada sob a forma de uma pequena quantidade de carga eléctrica; normalmente o valor lógico 0 é representado por uma carga nula e o valor lógico 1 por uma carga da ordem dos 500 femto coulombs ($c \times 10^{-15}$). O condensador responsável pelo armazenamento de informação nas células de memória perde com o tempo a sua carga, e portanto é necessário regenerar esta carga ("refrescar" a memória) pelo menos em cada dois milisegundos. O Z80A fornece um sinal de saída RFSH e possui um registo encarregue de refrescar a memória, R, que pode ser usado como parte desta operação. Mais adiante, neste mesmo Capítulo e no Capítulo 10, serão fornecidos mais detalhes sobre o assunto.

No Spectrum de 48 K, 16 K da memória RAM consistem em 16 K x 1 bit RAM's dinâmicas do tipo já descrito, e os adicionais 32 K de memória são fornecidos por oito 32 K x 1 bit de RAM's dinâmicas tipo TMS4532 ou equivalente. Estes 32 K de RAM adicional funcionam de modo semelhante aos 16 K anteriores, mas os seus endereços são descodificados na gama 32768 a 65535.

É evidentemente possível examinar o conteúdo de qualquer posição RAM endereçável usando a função PEEK, ou escrever num dado endereço usando a declaração POKE como já se disse no Capítulo 2. O leitor recorda que usámos estas funções PEEK e POKE no Capítulo 7 nas secções sobre gráficos definidos pelo utilizador, atributos da rede de caracteres, etc. Mais adiante discutiremos os modos de as usar sobre toda a memória.

O mapa de memória

Quando o microprocessador Z80A está a endereçar memória, está de facto a endereçar uma posição RAM ou ROM usando o endereço apropriado, que é enviado para o "bus" de endereços. Como é óbvio, cada posição de memória, tanto em RAM como em ROM, possui um endereço único definido pelo bus de endereços e pelas linhas de selecção em cada pastilha integrada.

O sistema microcomputador apresentado na figura 9.1 utiliza a lógica de descodificação de endereços no interior da pastilha integrada ULA Ferranti, permitindo o acesso a qualquer das 16 K

posições de memória da ROM ou das 16 K posições da RAM (ou 48 K se se possui a máquina de 48 K). As correspondentes gamas de endereços ROM e RAM são:

ROM 00000 a 16383
 RAM { 16384 a 32767 (RAM de 16 K)
 16384 a 65535 (RAM de 48 K)

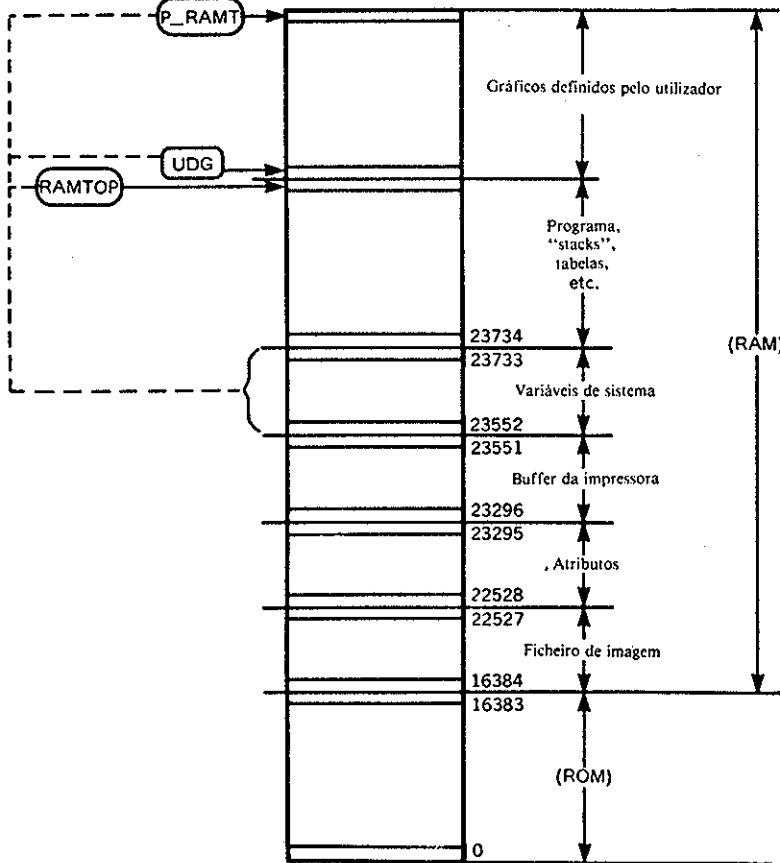


Figura 9.11. — Mapa de memória do Spectrum

As gamas de memória especificadas são muitas vezes ilustradas em diagramas sob a forma conhecida pela designação de “mapa de memória”, que indica as gamas de endereços destinadas a cada periférico ou secção do sistema. A figura 9.11 mostra os endereços na ROM e RAM do ZX Spectrum.

Os sinais de entrada e saída do microprocessador Z80A

Para permitir ao utilizador ligar hardware periférico ao ZX Spectrum (impressora, microdrive, dispositivos de entrada/saída, etc.) as ligações dos pinos do microprocessador são conduzidas ao ligador na parte traseira do computador (excepto 0, ver adiante). Só quando se compreende a função de cada sinal do microprocessador (MPU) se torna possível conceber e ligar novos circuitos de interface. Para auxiliar os interessados fornecemos em seguida o esquema da distribuição dos pinos do Z80A na figura 9.12 e resumimos as funções dos sinais.

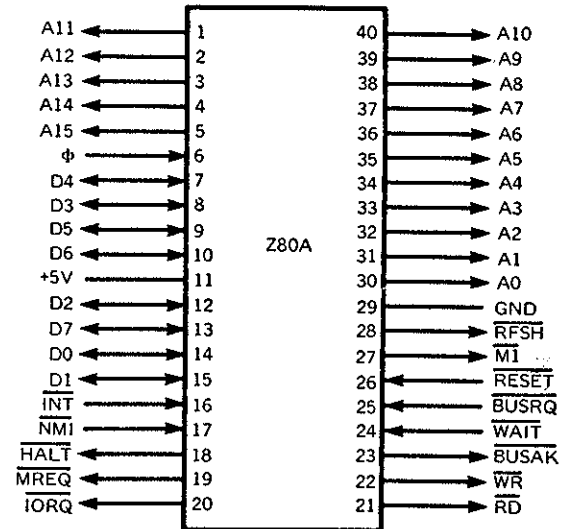


Figura 9.12 — Distribuição dos pinos do microprocessador Z80A.

O microprocessador de 40 pernas em linha dupla NMOS Z80A é accionado por uma alimentação de + 5V.

O sinal de relógio ϕ , produzido externamente, acciona a lógica de comando e temporização do microprocessador, e define o período de um ciclo (ver o Capítulo 10). A onda ϕ não surge no ligador, mas o perne CK deste contém um sinal com a mesma frequência (3,5 MHz) se bem que apenas com uma amplitude de 1,5 V relativamente à linha de 0 V (ver a figura 10.3).

As linhas de dados bidireccionais de entrada/saída de oito bits, activas ao nível elevado, D0 (bit menos significativo) a D7 (bit mais significativo), são usadas para transferências de dados entre a unidade microprocessadora (MPU) e a memória e os dispositivos de entrada-saída. As linhas de saída de endereço unidireccionais, de 16 bits e activas ao nível elevado, A0 (bit menos significativo) a A15 (bit mais significativo), são usadas para definir o endereço da memória e as transferências de dados para dispositivos de entrada/saída.

O bus de comando consiste em cinco ligações de entrada e oito de saída. As cinco entradas são:

1. — $\overline{\text{WAIT}}$. Trata-se de uma entrada activa ao nível baixo que leva a MPU a entrar num estado de espera. Este estado consiste num certo número de ciclos ineficazes inseridos num ciclo-máquina, ficando a MPU em “ponto morto” até $\overline{\text{WAIT}}$ passar ao nível lógico 1. É portanto possível comandar a MPU de tal modo que espere até a memória ou os dispositivos de entrada/saída estarem prontos para realizar transferências de dados.

2. — $\overline{\text{RESET}}$. Trata-se de uma entrada activa ao nível 0 que coloca o Contador de Programa em zero, desliga o *flip-flop* de accionamento de interrupções, passa a operação de interrupções para o modo 0, e limpa os registos R e I.

3. — $\overline{\text{INT}}$. Trata-se de uma entrada de pedido de interrupção activa ao nível 0 que corresponde a um sinal produzido por um dispositivo de entrada/saída quando o *flip-flop* de accionamento de interrupções (IFF) é activado, e quando o sinal $\overline{\text{BUSRQ}}$ está inactivo. A resposta a um sinal de entrada $\overline{\text{INT}}$ aceite é determinada pelo código do modo de interrupção especificado (ver a figura 10.6).

4. — $\overline{\text{NMI}}$. Trata-se da entrada de interrupções não “mascarável”, accionada pelos bordos negativos do sinal e activa ao ní-

vel 0. Possui uma prioridade superior a $\overline{\text{INT}}$ e é posta em execução no final da instrução em execução independentemente do estado do *flip-flop* de accionamento de interrupções (IFF). A resposta a um pedido $\overline{\text{NMI}}$ aceitável consiste em endereçar a MPU para a posição de memória 102 (ou seja, 0066 em hexadecimal). O sinal $\overline{\text{NMI}}$ não será aceite se $\overline{\text{WAIT}}$ e $\overline{\text{BUSRQ}}$ estiverem activos.

5. — $\overline{\text{BUSRQ}}$. Trata-se da entrada de pedido de “bus”, activa ao nível 0, e quando activa acciona os sinais dos *buses* de endereço e de dados da MPU e os sinais de saída de comando, passando-os para o estado de alta impedância. É usada nos casos em que dispositivos externos comandam os *buses* (linhas de ligação entre os vários componentes do sistema microcomputador). O estado de alta impedância é activado no final do ciclo-máquina em que o sinal $\overline{\text{BUSRQ}}$ o é.

As oito ligações de saída são:

1. — $\overline{\text{BUSAk}}$. Trata-se de uma saída de aceitação de *bus*, activa ao nível 0, que quando activa indica que os *buses* de endereço e dados da MPU e os sinais de comando de *bus* se encontram no estado de alta impedância.

2. — $\overline{\text{HALT}}$. Depois de executar uma instrução de software Halt esta saída torna-se activa ao nível 0, e a MPU continua executando NOP's (instrução de não funcionamento em código-máquina). Mantém-se neste estado até ser recebido um sinal de interrupção ($\overline{\text{NMI}}$ ou $\overline{\text{INT}}$).

3. — $\overline{\text{RFSH}}$. Trata-se de uma saída activa ao nível 0 que pode ser usada com memórias dinâmicas para refrescar os dados armazenados. Quando as linhas do bus de endereço A0 a A6 contêm um endereço a “refrescar” esta saída encontra-se activa.

4. — $\overline{\text{WR}}$. Quando o *bus* de dados contém dados válidos esta saída activa ao nível 0 indica essa condição a um dispositivo endereçado, podendo os dados ser escritos nesse dispositivo.

5. — $\overline{\text{RD}}$. Quando esta saída é activa (nível 0), pode ser usada por um dispositivo endereçado para reconhecer a disponibilidade da MPU para ler dados do *bus* de dados.

6. — $\overline{\text{IORQ}}$. Esta saída, activa ao nível 0, indica que se encontra presente um endereço de memória válido nas linhas de en-

dereço A0 a A7.

7. — $\overline{\text{MREQ}}$. esta saída activa ao nível 0 indica que se encontra um endereço de memória válido nas linhas de endereçamento A0 a A15.

8. — $\overline{\text{MI}}$. Esta saída estará activa (ao nível 0) durante o ciclo-máquina M1 (ver o Capítulo 10).

Na secção que se segue mostramos como se podem usar alguns dos sinais Z80A ao montar uma interface "memory-mapped" de um byte, e no capítulo 10 estudaremos a programação em código-máquina do microprocessador Z80A.

Interface "Memory-Mapped" de um byte

As ligações dos *buses* de endereço, dados e comando do ZX Spectrum encontram-se presentes no ligador externo (parte traseira; ver a figura 9.13 e a tabela 9.1).

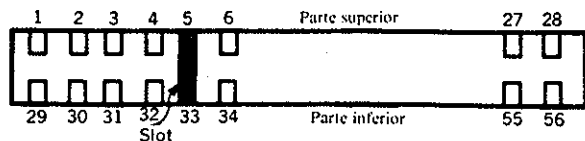


Figura 9.13. — Numeração dos pinos no ligador externo.

Para ler e escrever dados de ou para o *hardware* externo, por exemplo usando PEEK e POKE, é necessário dispor de uma lógica de descodificação de endereços apropriada a fim de assegurar que o *hardware* seja convenientemente endereçado ("memory-mapped"). Por outro lado, é necessário incluir uma lógica de comando a fim de determinar se o computador deve executar uma operação de leitura ou escrita (ver a figura 9.14). A fonte de dados deve fornecer sinais binários aceitáveis ao *bus* de dados para serem aproveitados durante o ciclo de leitura de memória do ZX Spectrum. No entanto, quando o *bus* de dados é requerido por outro dispositivo, as ligações da fonte devem ser isoladas do *bus* de dados. Isto pode ser conseguido usando *buffers*. Por outro lado,

quando o ZX Spectrum escreve dados para o *bus*, o dispositivo onde estes são armazenados deve ser capaz de capturar os dados durante o ciclo de escrita em memória. Deve portanto existir um *buffer* octal para esse fim.

Tabela 9.1 — Contactos do ligador externo

N.º do pino	Sinal	N.º do pino	Sinal
1	A15	29	A14
2	A13	30	A12
3	D7	31	+ 5V
4	Não usada	32	+ 9V
5	SLOT	33	SLOT
6	D0	34	0V
7	D1	35	0V
8	D2	36	CK
9	D6	37	A0
10	D5	38	A1
11	D3	39	A3
12	<u>D4</u>	40	A3
13	<u>INT</u>	41	IORQGE
14	<u>NMI</u>	42	0V
15	<u>HALT</u>	43	VIDEO
16	<u>MREQ</u>	44	Y
17	<u>IORQ</u>	45	V
18	<u>RD</u>	46	U
19	<u>WR</u>	47	BUSRQ
20	- 5V	48	RESET
21	WAIT	49	A7
22	+ 12V	50	A6
23	- 12V	51	A5
24	<u>MI</u>	52	A4
25	<u>RFSH</u>	53	<u>ROMCS</u>
26	A8	54	<u>BUSACK</u>
27	A10	55	A9
28	Não usado	56	A11

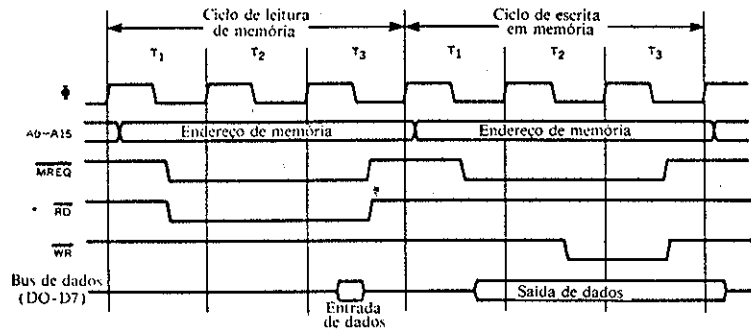


Figura 9.14 — Ciclos de leitura ou escrita da MPU

A Figura 9.15 mostra o diagrama lógico de uma interface *memory-mapped* de um byte. A lógica de descodificação de endereços atribui este "port" de entrada/saída ao endereço RAM 32767 (7FFF), que é o último byte na área da RAM dedicado aos gráficos definidos pelo utilizador no Spectrum de 16 K. Durante o ciclo de leitura de memória os *buffers* octais de dados (74LS244) são accionados pelo sinal \overline{RD} , ligando as entradas externas ao *bus* de dados. Durante o ciclo de escrita em memória o *buffer* octal (74LS373) é accionado pelo sinal \overline{WR} e os dados são aceites e apresentados o quadro de díodos emissores de luz (LED). Para efeitos de referência a distribuição dos pernos dos circuitos integrados usados no interface de um byte da figura 9.15 são indicados na figura 9.16, (a) a (f).

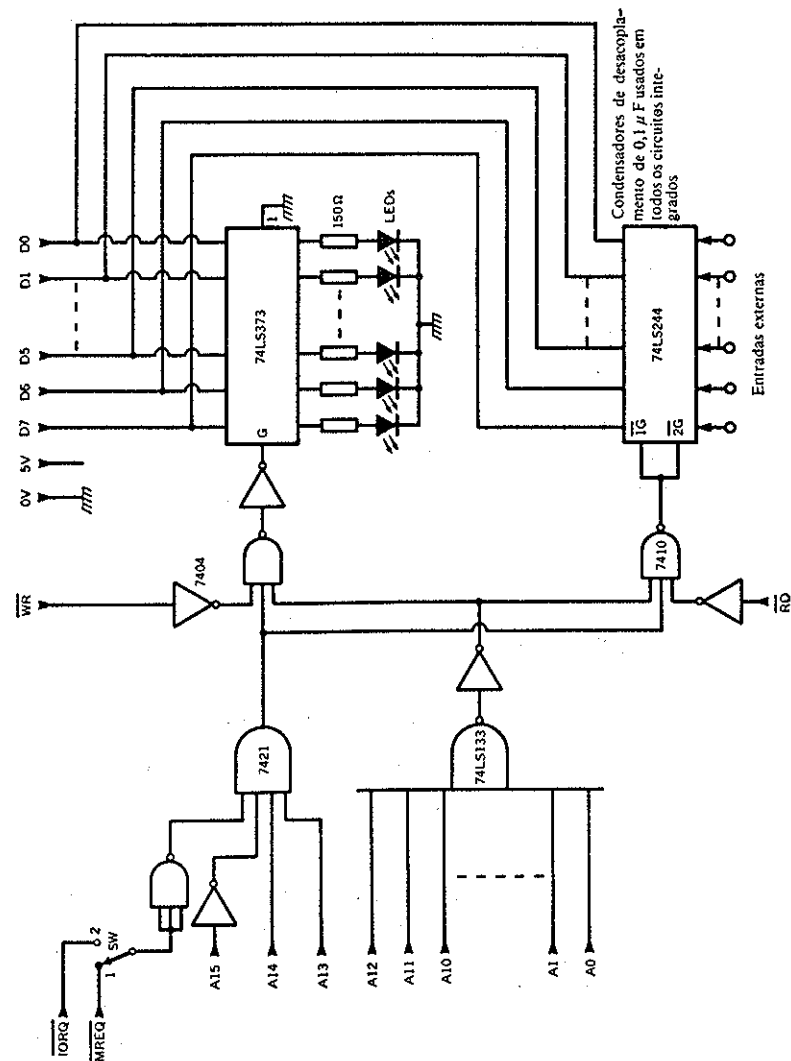


Figura 9.15 — Interface *memory-mapped* de um byte (todos os cabos de ligação entre a lógica de interface e o ZX Spectrum devem ser tão curtos quanto possível).

Para ler e apresentar um byte de informação obtido a partir deste hardware externo usamos

```
PRINT PEEK 32767
```

Podemos usar o simples programa seguinte para testar o *port* de entrada.

```
3 LET x = PEEK 32767
5 PRINT AT 10,14;x
7 PAUSE 20
8 PRINT AT 10,14;"   ": GO TO 3
```

Este programa lê periodicamente o byte de dados indicado pelo utilizador e apresenta o equivalente decimal do número de oito bits no visor.

Para enviar um byte de dados para o hardware externo use

```
POKE 32767, byte de dados
```

O programa simples que se segue pode ser usado para verificar o *port* de saída.

```
5 FOR n = 1 TO 255
6 PAUSE 20
10 POKE 32767,n
30 NEXT n
```

Este programa envia periodicamente um byte de dados para o quadro LED, começando pelo número binário igual a 1, e produz uma operação de contagem binária que continua até todos os LED's emitirem luz.

Um modo alternativo de comunicar com um *port* de entrada/saída consiste em usar a função IN e a declaração OUT. A função IN assume a forma

```
IN endereço de port
```

e lê o byte de dados do *port* endereçado. A declaração OUT assume a forma

```
OUT endereço do port, dados
```

e envia os dados para o *port* endereçado.

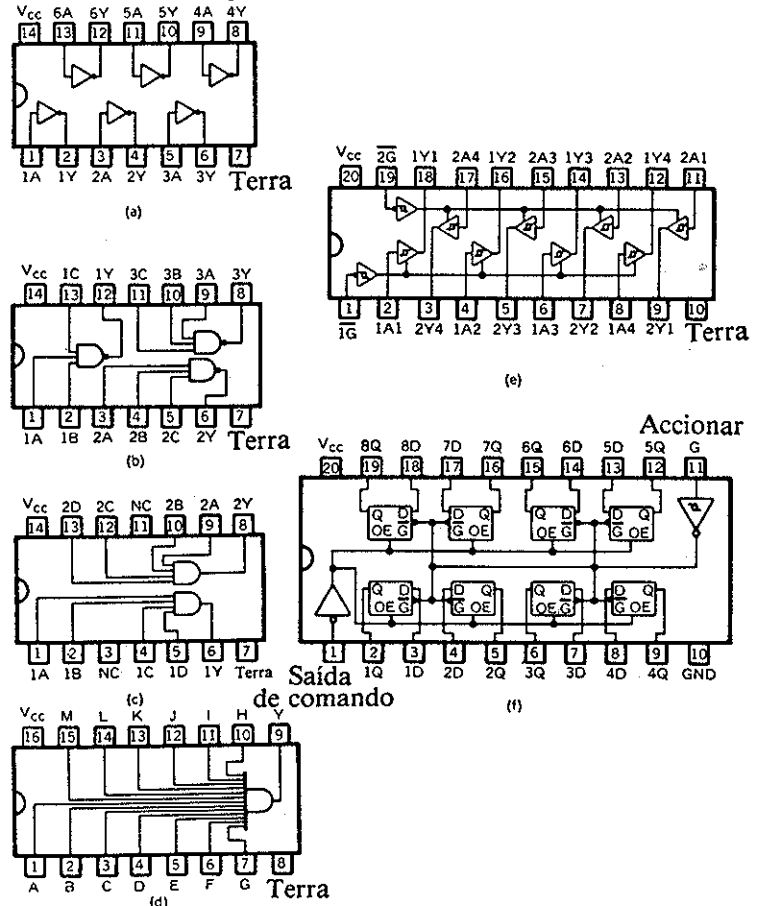


Figura 9.16 — Utilização dos pinos para um interface *memory-mapped* de um byte. (a) 7404 (inversor hexadecimnal), (b) 7410 (porta NAND tripla de três entradas), (c) 7421 (porta AND dupla de quatro entradas), (d) 74133 (porta NAND de 13 entradas), (e) 74LS244 (pilotagem de três estados), (f) 74LS373 (buffer de tipo D).

Estas instruções podem parecer semelhantes a PEEK e POKE, mas de facto IN e OUT não alteram o conteúdo da posição de memória que contém o mesmo endereço do *port* de entrada/saída. Pode-se verificar isto usando

```

5 FOR n = 1 TO 255
6 PAUSE 20
10 OUT 32767, n
30 NEXT n

```

Consequentemente, não é necessário reservar endereços de memória para os *ports* de entrada/saída.

No capítulo 10 o *port* de entrada/saída "memory-mapped" de um byte é usado com as versões em código-máquina de IN e OUT.

Para utilizar a interface de um byte apresentada na figura 9.15 com IN e OUT, é necessário assegurar que o comutador SW se encontra na posição 2, escolhendo portanto o sinal $\overline{\text{IORQ}}$ para obter a temporização requerida (ver a figura 9.17).

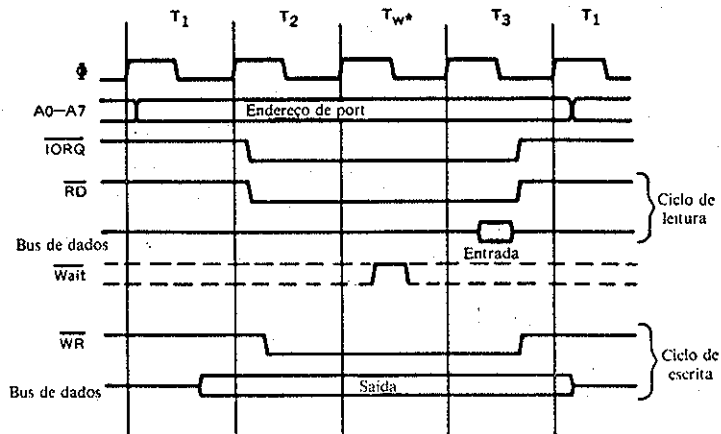


Figura 9.17 — Ciclos de entrada/saída da MPU (* período de espera inserido pelo Z80A).

O programa simples que se segue pode ser utilizado para verificar o *port* de entrada

```

3 LET x = IN 32767
5 PRINT AT 10,14;x
7 PAUSE 20
8 PRINT AT 10,14;" " : GO TO 3

```

Observações finais

Neste capítulo fizemos um resumo de alguns dos aspectos mais importantes do hardware de computador, e o material de referência aqui fornecido será certamente útil ao leitor se quiser conceber e utilizar hardware periférico.

CAPÍTULO X

PROGRAMAÇÃO EM CÓDIGO-MÁQUINA

Introdução

Quando escrevemos programas em Basic, as nossas declarações, funções e números devem ser armazenados e subsequentemente traduzidos em código-máquina equivalente pelo interpretador Basic e pelo sistema operativo contidos no ZX Spectrum. As ordens Basic são interpretadas de tal modo que o microprocessador Z80 A as possa executar, sendo necessário todo este trabalho de tradução dos programas porque a máquina só é capaz de compreender uma linguagem, o código-máquina.

O processo de interpretação leva um tempo finito a ser realizado, e se este puder ser eliminado escrevendo os programas directamente em código-máquina e não em Basic conseguir-se-á evidentemente poupar um tempo significativo. Esta consideração pode tornar-se importante em aplicações que envolvam o uso gráfico ou o comando de aparelhagem periférica.

O número de bytes de memória necessários para um programa escrito em Basic é bastante superior ao requerido para o programa em código-máquina equivalente, pelo que se consegue uma poupança significativa de memória escrevendo os programas em código-máquina.

Consideremos o modo como o microprocessador Z80 A realiza as instruções que lhe são dadas, e examinemos o conjun-

to de instruções que aceita e os modos de endereçamento possíveis. Veremos ainda o modo de fazer executar programas em código-máquina no ZX Spectrum.

Execução de instruções pelo microprocessador Z80 A

Na figura 10.1 pode ver-se um diagrama de blocos do Z80A, observando-se a existência de dezasseis linhas de endereçamento, oito de dados e treze de comando. O leitor recorda certamente que no capítulo 9 descrevemos a distribuição de pines e sinais do Z80A.

O Z80A (MPU) endereça os dispositivos do sistema (RAM, ROM e entradas/saídas) através do *bus* de endereços de 16 bits, sendo portanto capaz de endereçar 65536 (2^{16}) posições de memória na gama 0000 a FFFF (hexadecimal). As 16 linhas de endereçamento podem ser divididas entre dois bytes: os bits de endereço A0-A7 formam o byte menos significativo do endereço, e os bits A8-A15 formam o byte mais significativo do endereço. Por exemplo:

11000000 10011110

MSB

LSB

bus de endereçamento de 16 bits \equiv C09E

onde MSB (most significant byte) designa o byte mais significativo, e LSB (least significant byte) designa o menos significativo.

O bus bidireccional de dados, de oito bits, da MPU é usado para transmitir dados ou receber dados, da RAM, ROM e dos dispositivos de entrada/saída. Uma palavra do *bus* de dados terá por exemplo a forma

100001111

bus de dados de 8 bits \equiv 8F

O endereço de 16 bits guardado no registo Contador de Programa da MPU aponta para o código de operação seguinte (*op-code*) a transferir para o Registo de Instruções da MPU ao serem recebidos os sinais apropriados de temporização e comando. O conteúdo do Registo de Instruções é então descodificado, sendo a instrução nele indicada executada pela MPU.

Para ilustrar o princípio da operação de recolha e execução em MPU consideremos a instrução que carrega imediatamente o Registo D da MPU com o dado 6E. A partir do Conjunto de Instruções indicado na tabela 10.1, vemos que o código da operação que indica a carga de Registo D usando endereçamento imediato (LD r,n) é 00010110 (correspondente a 16 em hexadecimal), valor que consideraremos armazenado na posição de memória 27000 (6978 em hexadecimal). O dado 6E estará então guardado na posição de RAM imediatamente a seguir ao código de operação, isto é, no endereço 27001 (6979 em hexadecimal).

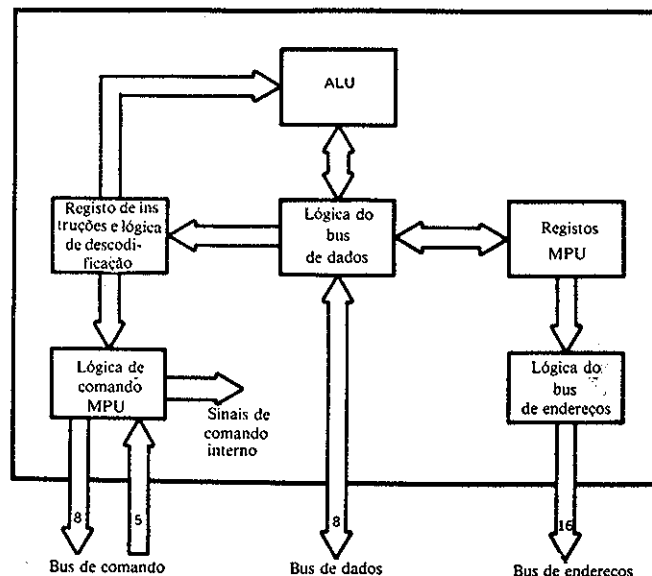


Figura 10.1. — Diagrama de blocos funcional do Z80A.

Os dois ciclos M têm uma duração diferente: M1 contém quatro ciclos-T, enquanto que M2 contém 3 ciclos-T, correspondendo cada ciclo-T ao período de relógio 0 da MPU. Obviamente o número de ciclos-T por ciclo de instrução determina o tempo de recolha e execução da instrução, o que depende da complexidade desta.

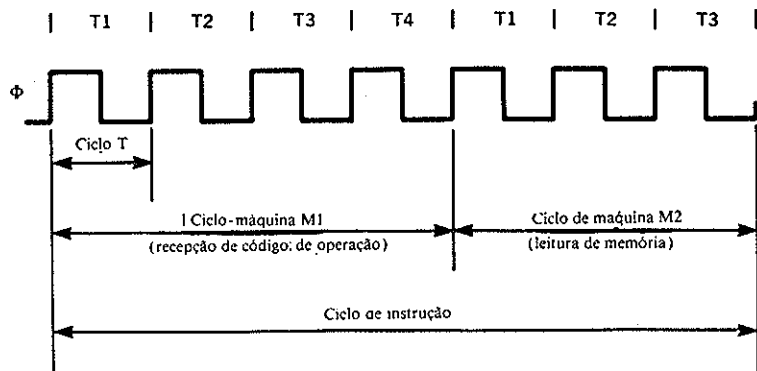


Figura 10.2. — Diagrama de temporização do exemplo de endereçamento imediato referido no texto.

A onda indicada na figura 10.3 é um oscilograma do impulso de relógio usado no ZX Spectrum, que é fornecido ao perne CK do ligador externo. O valor medido no período de cada ciclo-T é 285 nanosegundos ($285 \times 10^{-9}s$). Nestas condições, no exemplo anterior envolvendo sete ciclos-T o tempo necessário para recolher e executar a instrução é 7×285 nanosegundos = 1,995 microsegundos ($1,995 \times 10^{-6}s$). Este tempo corresponde ao uso do microprocessador Z80A a 3,5 MHz.

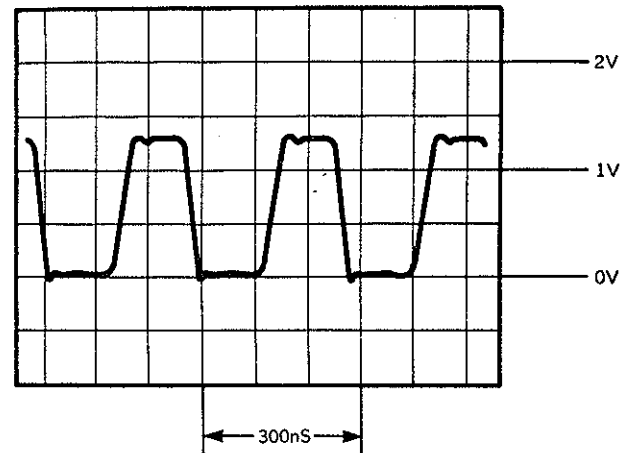


Figura 10.3. — Onda do perne CK do ligador externo. Registos acessíveis do Z80A

A configuração dos registos do MPU Z80A é indicada na figura 10.4, podendo-se verificar que contém 22 registos internos que podem ser acedidos por programa.

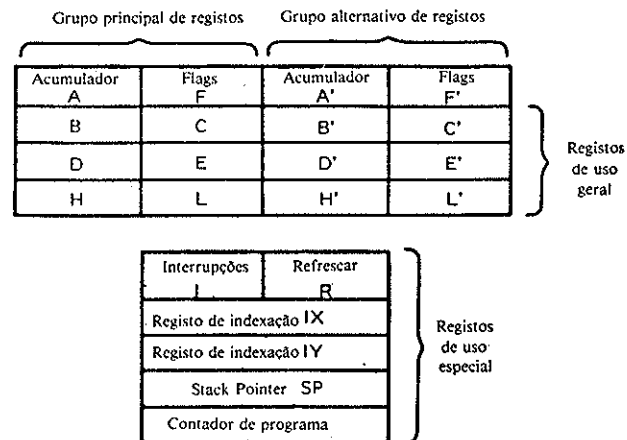


Figura 10.4. — Registos do Z80A acessíveis por programa.

Existem dois conjuntos de registos de uso geral, cada um deles contendo dois registos de 8 bits, e dois conjuntos de registos acumuladores e de *flags*. Os registos de oito bits para uso geral existentes em cada conjunto podem ser concatenados (emparelhados) de modo a formarem pares de registos com 16 bits (BC, DE e HL no caso do conjunto principal de registos, e B'C', D'E' e H'L' no caso do conjunto alternativo).

O programador pode, recorrendo a uma simples instrução de "troca" (*exchange*), trabalhar com o conjunto principal de registos ou com o conjunto alternativo. Este facto é bastante útil quando se pretende obter uma resposta rápida a uma interrupção. Nesses casos é possível programar a rotina de serviço da interrupção de modo a substituir rapidamente o conjunto principal pelo conjunto alternativo de registos, voltando àquele ao terminar a rotina de interrupção.

O registo acumulador e o de *flags* existentes no conjunto principal e alternativo de registos, são trocados recorrendo a uma única instrução "exchange". O acumulador, de oito bits, guarda o resultado das operações aritméticas e lógicas e o registo de *flags* guarda os estados de funcionamento correspondentes à ocorrência de certos acontecimentos específicos resultantes da execução de instruções (ver a figura 10.5). Por exemplo, a *flag* de transporte (*carry*), C, contém o bit de maior ordem do acumulador em função da instrução executada.

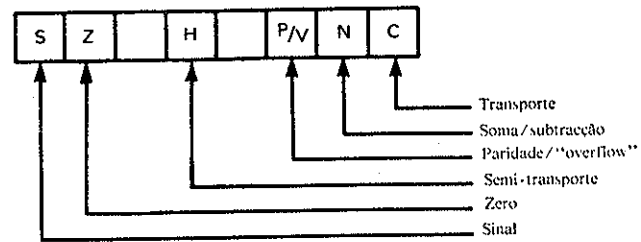


Figura 10.5. — Conteúdo do registo de *Flags* (registo F).

O Z80A contém ainda quatro registos de 16 bits, IX, IY, SP e PC, um registo I de oito bits, e um registo R de sete bits.

Estes registos executam tarefas especiais e serão descritos adiante.

Os dois registos IX e IY, de 16 bits, são registos de indexação independentes, usados para guardar endereços. Utilizam-se em operações que envolvam endereçamento em modo indexado, nas quais um inteiro com sinal em formato complemento para dois (incluído sob a forma de um byte adicional na intrusão) especifica o deslocamento relativamente ao endereçamento existente no registo de indexação x, definindo assim a posição de memória a aceder.

O registo de 16 bits SP (*stack-pointer*) contém o endereço superior do *stack*, isto é, da "pilha" de dados em uso pelo microprocessador, que se encontra em RAM e aceita ou fornece dados de tal modo que o último a ser aceite é o primeiro a ser retirado. Isto é, o *stack* permite guardar temporariamente dados existentes em registos especificados da MPU, sendo possível em seguida recolhê-los, começando pelo último, nos mesmos ou noutros registos.

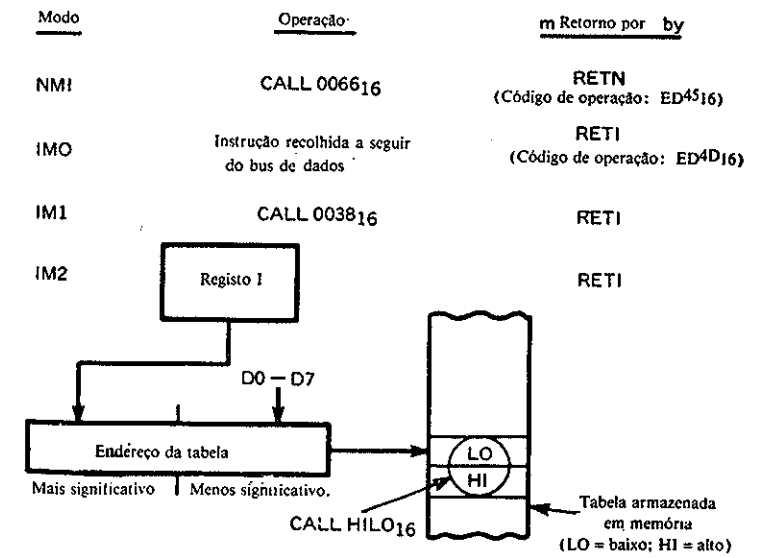


Figura 10.6. — Execução de uma interrupção

O registo Contador de Programa PC, também de 16 bits, contém o endereço da instrução que está a ser usada. O seu conteúdo altera-se em função dos requisitos de execução.

O registo I, de oito bits, é usado quando a MPU toma conhecimento da realização de uma interrupção, e armazena os oito bits de maior ordem do endereço da tabela de vectores (ver a figura 10.6.).

O registo R, de sete bits, é um contador automaticamente incrementado para cada ciclo M1 e encarregado de refrescar a memória (ver a figura 10.2.). Contém os endereços A0 a A6 que podem ser utilizados para aceder posições de memória dinâmica, e que são em seguida “refrescadas” pelo sinal RFSH do Z80A. Esta operação de “refrescamento” não interfere com o funcionamento normal da MPU e é transparente do ponto de vista do programador (o que significa que este não tem conhecimento dela).

Conjunto de Instruções do Z80A

É útil para o utilizador dispor dos 158 tipos diferentes de instruções do Z80A classificados em grupos do modo indicado nas Tabelas 10.1 a 10.11. Fornecem-se para cada instrução a mnemónica em linguagem *Assembler*, o símbolo da operação, o estado dos bits do Registo F (*flags*), o respectivo código de operação (em forma binária ou hexadecimal) e as informações relevantes quanto a armazenamento e temporização. Incluem-se ainda alguns comentários e a notação das *flags*.

Veremos mais adiante o modo de utilizar algumas destas instruções quando se escrevem programas em código-máquina.

Tabela 10.1. — Instruções de carga de registos de oito bits

Mnemónica	Operação simbólica	Flags								Código de operação	N.º de bytes	N.º de ciclos M	N.º de Estados T	Comentários.	
		S	Z	AC	H	OV	N	OV	C						
LD r, s	r ← s	•	•	•	•	•	•	•	•	•	•	•	•	•	000 B
LD r, n	r ← n	•	•	•	•	•	•	•	•	•	•	•	•	•	001 C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	•	•	•	•	•	•	•	010 D
LD r, (IX+ <i>d</i>)	r ← (IX+ <i>d</i>)	•	•	•	•	•	•	•	•	•	•	•	•	•	011 E
LD r, (IY+ <i>d</i>)	r ← (IY+ <i>d</i>)	•	•	•	•	•	•	•	•	•	•	•	•	•	100 H
LD (HL), r	(HL) ← r	•	•	•	•	•	•	•	•	•	•	•	•	•	101 L
LD (IX+ <i>d</i>), r	(IX+ <i>d</i>) ← r	•	•	•	•	•	•	•	•	•	•	•	•	•	111 A
LD (IY+ <i>d</i>), r	(IY+ <i>d</i>) ← r	•	•	•	•	•	•	•	•	•	•	•	•	•	
LD (HL), n	(HL) ← n	•	•	•	•	•	•	•	•	•	•	•	•	•	
LD (IX+ <i>d</i>), n	(IX+ <i>d</i>) ← n	•	•	•	•	•	•	•	•	•	•	•	•	•	

Mnemónica	Operação simbólica	S	Z	H	IV	M	C	76	543	210	Hex	DD	4	6	20	Par	
LD (nn), IX	(nn+1) ← IXH (nn) ← IXL	•	•	•	•	•	•	•	•	•	•	11 011 101 00 100 010	00 22	4	6	20	00 01 01 10 11
LD (nn), IY	(nn+1) ← IYH (nn) ← IYL	•	•	•	•	•	•	•	•	•	•	FD 22	4	6	20		
LD SP, HL	SP ← HL	•	•	•	•	•	•	•	•	•	•	F9	1	1	6		
LD SP, IX	SP ← IX	•	•	•	•	•	•	•	•	•	•	F9	2	2	10		
LD SP, IY	SP ← IY	•	•	•	•	•	•	•	•	•	•	FD	2	2	10		
PUSH qq	(SP-2) ← qqL (SP-1) ← qqH	•	•	•	•	•	•	•	•	•	•	DD	2	3	11		
PUSH IX	(SP-2) ← IXL (SP-1) ← IXH	•	•	•	•	•	•	•	•	•	•	E5	2	4	15		
PUSH IY	(SP-2) ← IYL (SP-1) ← IYH	•	•	•	•	•	•	•	•	•	•	FD	2	4	15		
POP qq	qqH ← (SP+1) qqL ← (SP)	•	•	•	•	•	•	•	•	•	•	E5	1	3	10		
POP IX	IXH ← (SP+1) IXL ← (SP)	•	•	•	•	•	•	•	•	•	•	DD	2	4	14		
POP IY	IYH ← (SP+1) IYL ← (SP)	•	•	•	•	•	•	•	•	•	•	E1 FD	2	4	14		

Notas: *dd* é qualquer registo dos pares de registos BC, DE, HL, SP.

qq é qualquer dos pares de registos AF, BC, DE, HL. (PAR)H, (PAR)L refere-se aos oito bits de maior e menor ordem respectivamente do par de registos, por exemplo: BCL = C; AFH = A

Notação das flags: — flag não afectada, 0 = flag limpa, 1 = flag ao nível 1, X = flag desconhecida, † = flag afectada em função do resultado da operação.

Tabela 10.3. — Instruções Z80A de Troca de Registos e Transfêrencia e busca de blocos

Mnemónica	Operação simbólica	Flags								Código de operação			N.º de bytes	N.º de ciclos M	N.º de Estados T	Comentários
		S	Z	H	IV	M	C	76	543	210	Hex					
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	•	•	•	•	E8	1	4	4	Troca de grupo principal por grupo alternativo de registos.
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	•	•	•	•	08	1	4		
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	•	•	•	•	•	•	•	•	•	•	09	1	4		
EX (SP), HL	H ↔ (SP+1) L ← (SP)	•	•	•	•	•	•	•	•	•	•	E3	1	5	19	
EX (SP), IX	IXH ↔ (SP+1) IXL ← (SP)	•	•	•	•	•	•	•	•	•	•	DD	2	6	23	
EX (SP), IY	IYH ↔ (SP+1) IYL ← (SP)	•	•	•	•	•	•	•	•	•	•	FD	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	•	0	•	•	•	•	•	•	FD A0	2	4	16	Carregar (HL) para (DE), incrementar 'indicadores (pointers) e decrementar o byte contador (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	•	0	•	•	•	•	•	•	ED	2	5	21	Se BC ≠ 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	•	0	•	•	•	•	•	•	BD	2	4	16	Se BC = 0

INC (HL)	(HL) ← (HL) + 1	1	1	X	1	X	V	0	•	100 110 [100]	1	3	11
INC (IX+d)	(IX+d) ← (IX+d) + 1	1	1	X	1	X	V	0	•	11 011 101	DD 3	6	23
INC (IY+d)	(IY+d) ← (IY+d) + 1	1	1	X	1	X	V	0	•	11 111 101	FD 3	6	23
DEC s	s ← s - 1	1	1	X	1	X	V	1	•	00 110 [100]			

s é qualquer de r, (HL), (IX + d), (IY + d) como em INC. DEC possui os mesmos estados e formato que INC. Substitui o grupo 100 por 101 no código de operação.

Notas: O símbolo V na coluna da flag P/V indica que esta contém o excesso do resultado da operação. Do mesmo modo, o símbolo P indica paridade. V = 1 significa excesso (over/low), V = 0 indica ausência deste, P = 1 indica paridade par do resultado, P = 0 indica paridade ímpar do resultado.

Notação das flags: • = flag não afectada, 0 = flag limpa, 1 = flag ao nível 1, X = flag desconhecida, † = flag afectada em função do resultado da operação.

Tabela 10.5. — Instruções Z80A de comando e uso geral

Mnemónica	Operação simbólica	Flags						Código de operação			N.º de bytes	N.º de ciclos de estados			Comentários		
		S	Z	H	P/V	N	C	76	543	210		Hex	M	I		ss	
ADD HL, ss	HL ← HL + ss	•	•	X	X	X	•	0	†	00	ss	001	1	3	11	ss	Reg.
ADC HL, ss	HL ← HL + ss + CY	†	†	X	X	X	V	0	†	11	101	101	ED	2	4	15	01 DE 10 HL 11 SP
SBC HL, ss	HL ← HL - ss - CY	†	†	X	X	X	V	1	†	11	101	101	ED	2	4	15	
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	†	11	011	101	DD	2	4	15	pp Reg. BC DE IX SP
ADD IY, rr	IY ← IY + rr.	•	•	X	X	X	•	0	†	11	111	101	FD	2	4	15	rr Reg. BC DE IY SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00	ss	011		1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11	011	101	DD	2	2	10	
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11	111	101	FD	2	2	10	
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00	ss	011		1	1	6	
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11	011	101	DD	2	2	10	
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11	111	101	FD	2	2	10	

Notas: ss é qualquer dos pares de registos BC, DE, HL, SP pp é qualquer dos pares registos BC, DE, IX, SP rr é qualquer dos pares de registos BC, DE, IY, SP.

Notação das flags: • = flag não afectada, 0 = flag limpa, 1 = flag ao nível 1, X = flag desconhecida, † = flag afectada em função do resultado da operação.

Tabela 10.6. — Instruções Aritméticas em 16 bits
 Código de operação



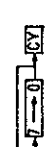

Mnemónica	Operação simbólica	Flags								N.º de bytes	N.º de ciclos M	N.º de Estados T	Comentários		
		S	Z	X	H	V	N	C	Hex						
DAA	Converte o conteúdo do ac. em BCD compacto após somas e subtrações com operandos compactos.	†	†	X	†	X	†	†	†	†	†	†	†	†	Acumulador de ajuste decimal
CPL	$A \rightarrow \bar{A}$	•	•	X	†	X	•	†	•	†	†	†	†	†	Acumulador (completamente para 1)
NEG	$A \rightarrow \bar{A} + 1$	†	†	X	†	X	V	†	†	†	†	†	†	†	Acumulador negação (complemento para 2)
CCF	$CY \rightarrow \bar{CY}$	•	•	X	X	X	•	†	†	†	†	†	†	†	Flag C do complemento
SCF	$CY \rightarrow 1$	•	•	X	0	X	•	†	†	†	†	†	†	†	Flag C ao nível 1
NOP	Não actua CPU parada	•	•	X	•	X	•	•	†	†	†	†	†	†	
HALT	HALT	•	•	X	•	X	•	•	†	†	†	†	†	†	
DI*	IFF = 0	•	•	X	•	X	•	•	†	†	†	†	†	†	
EI*	IFF = 1	•	•	X	•	X	•	•	†	†	†	†	†	†	
IM 0	Define modo 0 de interrupção	•	•	X	•	X	•	•	†	†	†	†	†	†	
IM 1	Define modo 1 de interrupção	•	•	X	•	X	•	•	†	†	†	†	†	†	
IM 2	Define modo 2 de interrupção	•	•	X	•	X	•	•	†	†	†	†	†	†	

Notas: IFF indica o *flip-flop* de accionamento de interrupção
 CY indica o *flip-flop* "carry" (C)

Notação de flags: • = flag não afectada, 0 = flag limpa
 1 = flag ao nível 1, X = flag desconhecida, † = flag afectada em função do resultado da operação.

* As interrupções não são reconectadas no final de EI ou DI

Tabela 10.7. — Instruções Z80A de Rotação e Deslocamento

Mnemónica	Operação simbólica	Flags								N.º de bytes	N.º de ciclos M	N.º de Estados T	Comentários		
		S	Z	X	H	V	N	C	Hex						
RLCA		•	•	X	0	X	•	†	†	†	†	†	†	†	Acumulador de rotação circular para a esquerda
RLA		•	•	X	0	X	•	†	†	†	†	†	†	†	Acumulador de rotação para a esquerda
RRCA		•	•	X	0	X	•	†	†	†	†	†	†	†	Acumulador de rotação circular para a direita
RRA		•	•	X	0	X	•	†	†	†	†	†	†	†	Acumulador de rotação para a direita
RLC r		†	†	X	0	X	P	†	†	†	†	†	†	†	Registro r de rotação circular para a esquerda
RLC (HL)		†	†	X	0	X	P	†	†	†	†	†	†	†	Reg. r
RLC (HX+d)	$r(HL)(HX+d)(HX+d)$	†	†	X	0	X	P	†	†	†	†	†	†	†	000 B
															001 C
															010 D
															011 E
															100 H
															101 L
															111 A

sideremos a instrução POP IX. Esta instrução carrega no byte menos significativo do registo IX o conteúdo existente no endereço de memória indicado pelo conteúdo do registo SP (Stack Pointer), e carrega no byte mais significativo do registo de indexação o dado guardado na posição de memória indicada pelo Stack Pointer mais um. Na Tabela 10.2 vemos que o código de operação para esta instrução é

DD
E1

Endereçamento imediato

Neste modo de endereçamento o byte de dados segue-se imediatamente ao código de operação. Por exemplo, a instrução com o formato LD r,n carrega o byte n no registo r. Assim, para carregar em E o valor F8 vemos na tabela 10.1 que devemos usar a instrução

1E código de operação
F8 dados

Endereçamento imediato extenso

Esta forma de endereçamento pode ser usada para carregar valores em pares de registos, incluindo registos especiais de 16 bits. Por exemplo, usando a tabela 10.2, vemos que a instrução

DD } Código de operação
21 }
FF Byte menos significativo de dados
00 Byte mais significativo de dados
carrega no registo de indexação IX o dado 00FF, e a instrução
01 Código de operação
BA Byte menos significativo de dados
AB Byte mais significativo de dados
carrega no par de registos BC o dado ABBA

Endereçamento extenso

Os últimos dois bytes da instrução são usados para especificar o endereço da posição de memória onde se encontram os dados a usar, ou contêm um endereço usado numa instrução de salto. Por exemplo, a instrução LD A, (nn) carrega no acumulador A o conteúdo da posição de memória com o endereço nn. Na tabela 10.1 vemos que para carregar o acumulador A com o conteúdo da posição de memória 1982 usamos a instrução

3A Código de operação
82 Byte menos significativo do endereço
19 Byte mais significativo do endereço

Um outro exemplo é a instrução com a forma LD dd, (nn), que carrega o par de registos dd (ver o código de dois bits na tabela 10.2) com dados dos endereços de memória *nn* e *nn + 1*, onde o conteúdo do byte de memória endereçado por *nn* é carregado no registo menos significativo do par e o conteúdo do byte de memória endereçado por *nn + 1* é carregado no registo mais significativo do par. Assim, para carregar o par de registos BC com os dados guardados nas posições de memória 1982 e 1983, usa-se a instrução

ED } Código de operação
4B }
82 LSB } Endereço dos dados para o registo C
19 MSB }

Uma forma simples de instrução de salto que utiliza esta forma de endereçamento é o Salto Incondicional, JP, *nn* (ver a tabela 10.9). Por exemplo, a instrução

C3 Código de operação
21 Byte menos significativo do endereço
C4 Byte mais significativo do endereço

fará o programa saltar para o endereço de memória C421, aceitando assim o primeiro byte da instrução seguinte.

Por outro lado a instrução de salto pode ser tornada condicional em função do estado de uma dada *flag* usando a instrução JP cc,nn indicada na tabela 10.9. Por exemplo, a instrução

CA	Código de operação
2B	Byte menos significativo do endereço
14	Byte mais significativo do endereço

levará o programa a saltar para a posição de memória 142B se o resultado da última instrução for zero. Se não, a instrução de salto é ignorada sendo executada a instrução de programa que se segue.

Endereçamento modificado de página zero

As oito instruções "restart" RST p, de um byte indicadas na tabela 10.10, utilizam endereçamento modificado de página zero para dirigir o comando de programa para um de oito endereços previamente definidos, que possuem um byte mais significativo de 00, e daí o nome dado a este modo de endereçamento. A instrução que permite ter acesso à posição de memória previamente definida requer apenas um código de operação de um byte, sem necessidade de quaisquer outros bytes indicando o endereço. Consegue-se assim poupar memória e tempo de execução. Trata-se de um modo muito útil de aceder subrotinas frequentemente usadas.

A ROM do ZX Spectrum utiliza as oito instruções RST nas suas rotinas de monitoragem. Por exemplo, a RST para a posição de memória 0008 é usada para aceder o endereço inicial da Rotina de Processamento de Mensagens de Erros. Como se mostra na tabela 10.10 o código de operação CF é usado para aceder 0008. Para utilizar isto em programas em código-máquina para gerar as nossas próprias mensagens de erro devemos usar o código de operação CF seguido pelo byte de dados requerido para definir o carácter do código alfanumérico da mensagem a apresentar. O byte de dados usado para as mensagens A a R é obtido subtraindo 38 (hexadecimal) ao código de operação, como se define na Tabela 7.1. Por exemplo, para apresentar a mensagem P (código hexadecimal 50), use a instrução

CF	Código de operação
18	Byte de dados (50-38)

O byte de dados usado para as mensagens 0-9 é obtido subtraindo 31 (hexadecimal) ao código hexadecimal indicado na tabela 7.1.

Endereçamento implícito

As instruções que trabalham com o conteúdo de um registo específico são conhecidas como instruções de endereçamento implícito. Por exemplo, é este o caso das operações aritméticas e lógicas sobre o conteúdo do acumulador A. Esta forma de endereçamento é ilustrada nos exemplos seguintes:

1. Para incrementar o conteúdo de um registo escolhido podemos usar a instrução INC r. Por exemplo, para incrementar o conteúdo do registo B podemos usar o código de operação 04, obtido na tabela 10.4.

2. Para formar o complemento para um do conteúdo do acumulador A utilizamos a instrução CPL. O código de operação que lhe corresponde é 2F, ver a tabela 10.5.

3. Para deslocar o conteúdo do registo H uma posição para a esquerda, transferindo um 0 para o bit menos significativo de H e o bit mais significativo deste para o bit de transporte (*carry*), podemos usar a instrução SLA s. O código de operação relevante pode ser obtido na tabela 10.7:

CB

24

O comentário na tabela 10.7 segundo o qual o formato da instrução e os estados são os mesmos das instruções RLC r, define o primeiro byte como sendo CB e o segundo como igual a 00100100, isto é, 24.

Endereçamento de bits

É possível endereçar um único bit num dado registo ou posição de memória e "limpá-lo" (torná-lo igual a 0) ou colocá-lo ao nível 1 em função da instrução especificada.

Os exemplos seguintes ilustram desta forma de endereça-

mento. Note que os bits de um byte são numerados entre 0 e 7, sendo o bit 0 o menos significativo e o 7 o mais significativo.

1. A instrução BIT b,r (indicada na tabela 10.8) tornará a *flag Zero (flag Z)* igual ao complemento do estado lógico do bit endereçado, b, do registo r escolhido. Por exemplo, para tornar a *flag Zero* igual ao complemento do bit 3 do registo L necessitaremos do código de operação

· CB
5D

2. A instrução RES b, (HL) é usada para limpar o bit b da posição endereçada pelo conteúdo do par de registos HL, ver a tabela 10.8. Por exemplo, a instrução

CB
AE (10101110)

limpará o bit 5 da posição de memória RAM endereçada pelo conteúdo do par de registos HL no momento em que a instrução é executada.

Endereçamento indexado

O microprocessador Z80A utiliza esta forma de endereçamento para aceder o byte de dados que se pretende usar com a instrução. Define o endereço de memória desejado somando o conteúdo do registo de indexação em causa, IX ou IY, a um *byte de deslocamento*, d, com o conteúdo do mesmo registo e o byte de deslocamento não alterado quando a instrução é executada.

O byte de deslocamento é um número binário de oito bits na forma complemento para dois (ver o Capítulo 2), sendo portanto possível que este byte tenha um valor na gama +127 (01111111) a -128 (10000000) inclusive. Consequentemente, as posições de memória são endereçadas usando (IX + d) ou (IY + d), onde d varia entre -128 e +127.

A instrução LD r, (IX + d), indicada na Tabela 10.1, será usada para ilustrar este modo de endereçamento. Para este efeito vamos supor que o conteúdo do registo de indexação IX é

0200, e que é necessário copiar os dados contidos na posição de memória 01FB para o registo E. A instrução requerida (ver a tabela 10.1) será

DD
5E (01011110)
FB (11111011 = -5 = 01FB-0200)

Endereçamento relativo

Este modo de endereçamento aplica-se a algumas das instruções de salto do Z80A, que são constituídas pelo código de operação apropriado seguido por um byte de deslocamento na forma complemento para dois indicado por e-2.

O microprocessador consegue o desejado salto do comando de programa através da alteração do conteúdo do contador de programa, PC, permitindo assim o acesso à instrução desejada.

O número binário complemento para dois igual a e é somado ao endereço de memória do código de operação da instrução de salto (guardado em PC) de modo a obter a desejada modificação em PC. Como e-2 é um número binário de oito bits na forma complemento para dois representando números na gama -128 a +127, conclui-se que é possível fazer saltar o comando do programa para qualquer posição na gama -126 a +129 relativamente ao endereço do código de operação da instrução de salto. Isto significa que como o comando de programa não salta para um endereço de memória especificado, pode ser possível alterar facilmente a posição do programa na memória. É portanto melhor utilizar instruções em modo de endereçamento relativo do que em modo de endereçamento extenso. O exemplo seguinte utiliza a instrução JR e, indicada na Tabela 10.9, para ilustrar este modo de endereçamento.

Suponhamos que o código de operação da instrução requerida, 18, se encontra guardado na posição de memória 4AFC, e que pretendemos fazer saltar o comando de programa para a posição de memória 4B02; a instrução em código-máquina para o efeito, obtida na Tabela 10.9, é

18 Código de operação
 04 e-2 (e = 6; 4AFC + 6 = 4B02)

No nosso estudo dos modos de endereçamento vimos os códigos-máquina para cada instrução. Quando se armazenam estas instruções em posição de memória seguidas cria-se um programa em código-máquina que o microprocessador pode executar recolhendo de cada vez uma delas e executando-a.

Como ilustração muito simples de um programa em código-máquina, listamos em seguida os códigos de um programa que carrega no acumulador A o número hexadecimal 8D, copia em seguida o conteúdo de A para o registo B, forma o complemento para 1 do conteúdo de A, e finalmente copia o conteúdo de A para o registo C.

3E	Código de operação	}	LD A,8D	Tabela 10.1
8D	Dados			
47	Código de operação		LD B,A	Tabela 10.1
2F	Código de operação		CPL	Tabela 10.5
4F	Código de operação		LD C,A	Tabela 10.1

A execução destas quatro instruções resulta em (A) = 72, (B) = 8D e (C) = 72, onde () indica o conteúdo do registo.

Armazenamento e execução de programas em código-máquina

É possível guardar programas em código-máquina em dois locais: na mesma área onde se guardam os programas Basic ou numa área separada criada para este fim entre a RAMTOP e os gráficos definidos pelo utilizador (ver a figura 9.11).

Para incluir código-máquina no interior de um programa Basic pode-se usar uma declaração REM. No entanto, de um ponto de vista prático, a escolha do local onde se deve encontrar esta declaração REM é determinada pela necessidade de conhecer o endereço exacto do primeiro byte do programa em código-máquina. Quando a REM ocupa a primeira linha do programa Basic, o endereço do primeiro byte usado para este programa é determinado lendo o valor da variável de sistema

PROG (ver o Apêndice C). Pode-se fazê-lo do seguinte modo:

```
10 LET x = PEEK 23635
20 LET y = PEEK 23636
30 PRINT "Endereço do primeiro byte de Basic;"
x + 256 * y
```

O endereço do primeiro byte do programa em código-máquina será obtido usando o programa anterior mais cinco, porque o número da linha Basic, o comprimento desta e a palavra-chave REM ocupam cinco bytes. O leitor verificará que o endereço do primeiro byte do seu programa Basic é 23755 quando não usa microdrives, e nessas condições o código-máquina pode ser armazenado a partir do endereço 23760 inclusive. Se colocar a instrução REM em qualquer outro ponto do programa, terá de determinar o endereço em RAM a partir do qual pode guardar o código-máquina, o que pode não ser uma tarefa fácil.

Ao usar a declaração REM na primeira linha do seu programa Basic, o número de caracteres entre a palavra-chave REM e ENTER corresponde ao número de bytes que podem ser usados para guardar o código-máquina.

O método de inclusão de um programa em código numa declaração REM envolve portanto duas fases. Na primeira cria-se uma declaração REM necessária para reservar o espaço de memória necessário para guardar o programa em código-máquina, e na segunda convertem-se os bytes de código nos seus equivalentes decimais e transferem-se estes (usando POKE) para as posições de memória apropriadas.

Consideremos agora como é possível guardar deste modo os seguintes três bytes de código-máquina.

```
04 INC B
0D DEC C
C9 RET
```

A primeira operação consiste em criar a declaração REM com três bytes livres:

```
1 REM bbb
```

onde os caracteres bbb correspondem aos três bytes necessários

para o código-máquina. Os bytes do código devem em seguida ser convertidos nos seus equivalentes decimais (ver a tabela 7.1.), isto é

04	Converte para 04
0D	Converte para 13
C9	Converte para 201

e depois armazenados nos endereços de memória 23760; 23761 e 23762. Note que ao terminar esta operação em duas fases a declaração REM da linha 1 será listada pelo computador sob a forma

1 REM?

Quando se inclui código-máquina numa declaração REM aquele passa a fazer parte de um programa Basic e portanto a estar sujeito a todas as ordens Basic, por exemplo EDIT, LIST, SAVE, NEW (que apaga todo o programa), etc.

Um programa em código é invocado por um programa Basic usando a função USR. O endereço de memória do primeiro byte de código-máquina é o usado depois da função USR para executar o código. O equivalente hexadecimal deste número é carregado no par de registos BC do Z80A e, depois de executar a última instrução de código, ou seja a essencial instrução de retorno (C9), o conteúdo do par de registos BC volta à função USR.

Nestas condições, para aceder e executar os três bytes de código-máquina listados atrás depois de terem sido armazenados no computador, podemos usar

PRINT USR 23760

seguido de ENTER. O resultado apresentado é 24015, que é o resultado correcto. Ou seja, antes de as instruções em código-máquina serem executadas, os valores armazenados nos registos B e C do microprocessador Z80A são

(B) = 92 e (C) = 208

(não esqueça que $23760 = (92 \times 256) + 208$), e depois de ser executada a primeira instrução, INC B, o conteúdo dos registos será

(B) = 93 e (C) = 208.

Depois de executada a segunda instrução, DEC C, o conteúdo dos registos é

(B) = 93 e (C) = 207

e depois da instrução RET ser executada a função USR dá o valor

$(93 \times 256) + 207 = 24015$

Vimos neste exemplo que a função USR é usada numa linha do programa Basic para aceder ao programa em código-máquina; convirá notar ainda que, no caso de se usar

25 LET x = USR 23760

depois de o programa em código-máquina ser executado o valor 24015 será atribuído à variável x.

Note ainda que a instrução de retorno (RET) deve ser incluída sempre como último código de operação do programa em código. Se assim não for a máquina executará todos os códigos que existirem em RAM depois do programa, e daí resultará quase certamente um "crash" (estoiro) do sistema. O sistema operativo do ZX Spectrum perde o controlo da situação e torna o computador incapaz de fazer qualquer coisa de útil. Por exemplo, é fácil reconhecer esta situação quando não se consegue comunicar seja o que for à máquina através do teclado. Infelizmente o único modo de terminar esta situação consiste em desligar momentaneamente a alimentação, o que evidentemente provoca a perda do programa guardado em RAM.

A outra área da RAM onde é possível armazenar um programa em código-máquina é a que podemos criar acima da RAMTOP e abaixo dos gráficos definidos pelo utilizador. Esta área é sempre usada quando não se pretende que a instrução NEW apague o programa em código-máquina.

No Spectrum, os gráficos definidos pelo utilizador ocupam os últimos 168 bytes da RAM, e a RAMTOP (limite superior da RAM utilizável em Basic) encontra-se normalmente no byte imediatamente inferior ao primeiro daqueles (ver a figura 9.11). Assim, no ZX Spectrum de 16 K a RAMTOP encontra-se normalmente no endereço 32599, enquanto que no Spectrum de 48 K se encontra no endereço 65367. É possível redefi-

nir o endereço da RAMTOP inserindo o novo endereço numa declaração CLEAR, com o formato

CLEAR valor desejado da RAMTOP

Note ainda que CLEAR também limpa todas as variáveis do programa, do ficheiro de imagem (CLS) e da pilha de endereços de GOSUB, esta última colocada imediatamente abaixo da RAMTOP; além disso CLEAR provoca uma acção de RESTORE dos dados em programa e volta a posição de PLOT para a origem.

Depois de ter redefinido a RAMTOP de modo a reservar uma quantidade de memória suficiente para o seu programa em código, pode-se guardar este nos bytes obtidos. O programa em código será acedido usando novamente a funçãoUSR seguida do endereço inicial do programa, como se disse anteriormente.

Por exemplo, considerando que se usa um Spectrum de 48 K e o pequeno programa de três bytes indicado mais atrás, ou seja:

```
04    INC B
0D    DEC C
C9    RET
```

iremos guardar o primeiro byte do código no endereço da memória 60001. Teremos portanto de redefinir a RAMTOP escrevendo

```
CLEAR 60000
```

Em seguida converteremos os códigos nos seus equivalentes decimais, o que neste caso, como já vimos anteriormente, dá

```
04    INC B
13    DEC C
201   RET
```

Estes valores são em seguida guardados nos endereços de memória 60001, 60002 e 60003 respectivamente, podendo o programa ser executado escrevendo

```
PRINT USR 60001
```

O leitor notará que, no caso de dispor de uma máquina de 48 K, o Spectrum dá como resposta o valor 60256. Ou seja, antes de as instruções em código-máquina serem executadas os valores armazenados nos registos B e C são

(B) = 234 e (C) = 97

(recordar que $60001 = (234 \times 256) + 97$)

e depois de executar as três instruções a funçãoUSR produz o valor

$(235 \times 256) + 96 = 60256$

Exercício

No seu Spectrum de 16 ou 48 K defina a RAMTOP para o valor 30 000, em seguida armazene nos três bytes que se seguem a esse os códigos

```
04    INC B
0D    DEC C
C9    RET
```

e execute o programa. Verifique que depois de executar o programa a funçãoUSR fornece o resultado 30256.

Note que depois de ter redefinido a RAMTOP esta se mantém no novo endereço até que a alteremos de novo ou desliguemos a alimentação.

Pode-se guardar os programas em código-máquina em cassettes utilizando a declaração SAVE sob a forma

SAVE "nome do programa em código" CODE endereço inicial, n.º de bytes do programa

Por exemplo, no caso do exercício anterior utilizaríamos:

```
SAVE "programa" CODE 30001,3
```

Neste caso designámos os três bytes pelo simples nome de "programa".

O tédio provocado pela conversão de bytes de programas em código para os respectivos equivalentes decimais e pelo seu armazenamento nas posições de memória desejadas pode ser eliminado em boa parte recorrendo ao programa 6 do Apêndice A.

qual é determinado usando a equação indicada no Capítulo 7, ou seja:

$$(BC) = 16384 + y + x(32xr) + (256x \text{ n.}^\circ \text{ do byte do caracter}) + (2048 \times \text{n.}^\circ \text{ da secção do visor})$$

onde r é o n.º da linha do caracter na gama 0 a 7 (ver a figura 7.10). O n.º do byte do caracter é igual a zero para o primeiro byte do Ficheiro de Imagem, e o número da secção do visor é determinado pelo valor de x : nestas condições, segue-se que

$$(BC) = 16384 + y + (32xx) \text{ para } (x=r) < 8, \text{ ou}$$

$$(BC) = 16384 + y + (32x(x-8)) \text{ para } r=(x-8) \text{ a } 8 \leq x \leq 15, \text{ ou}$$

$$(BC) = 20480 + y + (32x(x-16)) \text{ para } r=(x-16) \text{ e } x > 15$$

O fluxograma mostra que o endereço do primeiro byte do Ficheiro de Imagem, endereçado por (BC), é então guardado no Registo de Indexação IY. Isto liberta o par de registos BC, que é em seguida usado para guardar o endereço de memória do primeiro byte do caracter armazenado em ROM, especificado pelo código do Conjunto de Caracteres. Ou seja:

$$(BC) = 15616 + (8x(\text{código}-32)) + (E)$$

Exemplos Ilustrativos

Programa pseudo — Print At

O principal objectivo deste exemplo de programação em código-máquina, que executa uma pseudo-operação PRINT AT x,x é:

a) demonstrar o modo como se pode usar o conjunto principal de registos do microprocessador Z80A com diferentes tipos de endereçamento;

b) Mostrar como é possível copiar um caracter armazenado em ROM para o Ficheiro de Imagem, e como se pode definir os atributos do caracter usando o Ficheiro de Atributos (questões já referidas no Capítulo 7).

O fluxograma deste exemplo é apresentado na figura 10.7. A primeira parte do processo envolve a definição do conteúdo do par de registos HL, nele guardando o endereço do Ficheiro de Atributos, isto é, K

$$(HL) = 22528 + y + (32xx)$$

onde x e y correspondem aos números da linha e da coluna da rede de caracteres (ver a figura 7.6). O acumulador A é em seguida carregado com os atributos do caracter, sendo estes guardados no byte do Ficheiro de Atributos endereçado pelo conteúdo do par de registos HL.

Neste ponto do processo os atributos do caracter já se encontram definidos.

A parte seguinte do processo envolve a definição do conteúdo do par de registos BC igual ao endereço de memória do primeiro byte no Ficheiro de Imagem do caracter a imprimir, o

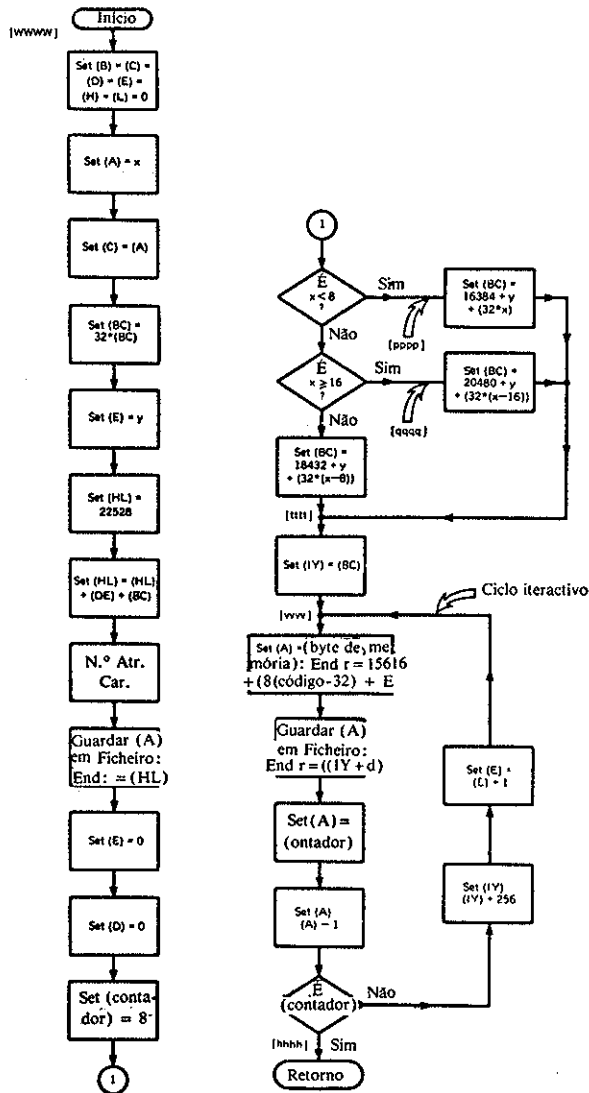


Figura 10.7 — Fluxograma de um programa executando uma pseudo PRINT AT

onde $(E) = K$ dá acesso ao byte de ordem K do carácter para $0 \leq k \leq 7$. O primeiro byte do carácter em ROM é guardado no acumulador A, e em seguida o conteúdo de A, (A) é armazenado no byte do Ficheiro de Imagem endereçado usando o conteúdo já definido do Registo de Indexação IY. Neste ponto do processo o primeiro byte do carácter ROM já foi copiado para o primeiro byte do carácter no Ficheiro de Imagem.

Ao entrar no ciclo pela primeira vez a variável de contagem não é zero, e portanto o ciclo é repetido. No entanto, para obter acesso ao byte seguinte do carácter em ROM e ao segundo byte do carácter no Ficheiro de Imagem, é necessário actualizar (E) e (IY) respectivamente, como se mostra no fluxograma. A variável de contagem é decrementada de cada vez que o ciclo é executado, e consequentemente o programa conta o número de bytes do carácter em ROM que são copiados para o Ficheiro de Imagem. Quando os oito bytes do carácter em ROM já se encontram copiados é executada a instrução RET, novamente essencial.

O programa em código-máquina para executar todo este processo é listado mais abaixo. Antes de usar o programa convém no entanto notar os pontos seguintes:

b) É necessário inicializar as variáveis de programa x, y, código c N.º Atr. Car., fazendo POKE's dos números requeridos para as respectivas posições de memória.

c) As gamas de valores aceites são: $0 \leq x \leq 21$, $0 \leq y \leq 31$ e $32 \leq \text{código} \leq 127$.

Considerando o ponto (a) acima, note que no caso de as variáveis do programa e o programa, tal como é listado abaixo, forem carregados imediatamente acima da RAMTOP, então os endereços gerais usados na listagem terão valores relativos à RAMTOP, como se segue:

mmmm	RAMTOP + 1, ou seja RT + 1
aaaa	RT + 2
bbbb	RT + 3
cccc	RT + 4
dddd	RT + 5
www	RT + 6

tttt RT + 109
 vvvv RT + 115
 pppp RT + 166
 qqqq RT + 196
 hhhh RT + 228

No caso do Spectrum de 16 K ou do de 48 K, quando RT é tornado igual a 32300 os endereços do programa são

	Decimal	Hexadecimal
mmmm	32301	7E2D
aaaa	32302	7E2E
bbbb	32303	7E2F
cccc	32304	7E30
dddd	32305	7E31
wwwwww	32306	7E32
tttt	32409	7E99
vvvv	32415	7E9F
pppp	32466	7ED2
qqqq	32496	7EF0
hhhh	32528	7F10

Listagem do programa

Endereço	Byte de memória	Mnemônica	Observações	Tabelas de instruções
mmmm	Contador	—	Variável do programa	—
aaaa	x	—	Variável do programa	—
bbbb	y	—	Variável do programa	—
cccc	Código	—	Variável do programa	—
dddd	N. Atr. Car.	—	Variável do programa	—
wwwwww	06	LD r,n	(B) = 0	10.1
	00	LD r,n	(C) = 0	10.1
	0E			
	00	LD r,n	(D) = 0	10.1
	16			
	00	LD r,n	(E) = 0	10.1
	1E			
	00	LD r,n	(H) = 0	10.1
	26			
	00	LD r,n	(L) = 0	10.1
	2E			
	00	LD A,(nn)	(A) = x	10.1
	3A			
	aa (Byte inferior de Ender; n)	}	(A) = x	
	aa (Byte superior de Ender; n)			

Endereço	Byte de memória	Mnemônica	Observações	Tabelas de instruções
	4F	LD r,s	(C) = x, isto é (BC) = x	10.1
	CB	SLA s		10.7
	21	RLs	2*(BC)	10.7
	CB			
	10	SLAs	4*(BC)	10.7
	CB			
	21	RLs		10.7
	CB			
	10	SLAs	8*(BC)	10.7
	CB			
	21	RLs	16*(BC)	10.7
	CB			
	10	SLAs	32*(BC)	10.7
	CB			
	21	RLs		10.7
	CB			
	10	LD A,(nn)	(A) = y	10.1
	3A			
	bb (Byte inferior de Ender; n)	LD r,s	(E) = y	10.1
	bb (Byte superior de Ender; n)			
	5F	LD dd,nn	(HL) = 5800 (hex)	10.2
	21			
	00 (Byte inferior de dados)	}	= 22528 (decimal)	
	58 (Byte superior de dados)			
	19	ADD HL,ss	(HL) = (HL) + (DE)	10.6
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	3A	LD A,(nn)		10.1
	dd (Byte inferior de Ender; n)	}	Guardar (A): Ender = (HL)	10.1
	dd (Byte superior de Ender; n)			
	77	LD(HL),r	(E) = 0	10.1
	1E	LD r,n		
	00	LD r,n	(D) = 0	10.1
	16			
	00	LD r,n	(A) = 8	10.1
	3E			
	08	LD(nn),A	(Count) = 8	10.1
	32			
	mm (Byte inferior de Ender; n)	LD A,(nn)	(A) = x	10.1
	mm (Byte superior de Ender; n)			
	3A	ADD A,n	(A) = (A) - 8	10.4
	aa (Byte inferior de Ender; n)			
	aa (Byte superior de Ender; n)	JP cc,nn	Se x < 8 saltar para pppp, senão herwise continuar	10.9
	C6			
	F8	LD A,(nn)	(A) = x	10.1
	FA			
	pp (Byte inferior de Ender; n)	}		
	pp (Byte superior de Ender; n)			
	3A	LD A,(nn)	(A) = x	10.1
	aa (Byte inferior de Ender; n)			
	aa (Byte superior de Ender; n)	}		
	aa (Byte superior de Ender; n)			

Endereço	Byte de memória	Mnemônica	Observações	Tabelas de instruções
	C6	ADD A,n	(A) = (A) - 16	10.4
	F0	(= -16)		
	F2	JP cc,nn	Se x > 15 saltar para qqqq, senão otherwise continuar	10.9
	qq (Byte inferior de Ender; n)			
	qq (Byte superior de Ender; n)			
	3A	LD A,(nn)	(A) = x	10.1
	aa (Byte inferior de Ender; n)			
	aa (Byte superior de Ender; n)			
	C6	ADD A,n	(A) = x-8	10.4
	F8	(= -8)		
	CB	SLA r	(A) = 2*(x-8)	10.7
	27			
	CB	SLA r	(A) = 4*(x-8)	10.7
	27			
	CB	SLA r	(A) = 8*(x-8)	10.7
	27			
	CB	SLA r	(A) = 16*(x-8)	10.7
	27			
	CB	SLA r	(A) = 32*(x-8)	10.7
	27			
	21	LD dd,nn		10.2
	00 (Byte inferior de dados)		(HL) = 4800 (hex)	
	48 (Byte superior de dados)		= 18432 (decimal)	
	06	LD r,n	(B) = 0	10.1
	00			
	4F	LD r,s	(C) = (A)	10.1
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	3A	LDA,(nn)		10.1
	bb (Byte inferior de Ender; n)		(A) = y	
	bb (Byte superior de Ender; n)			
	4F	LD r,s	(C) = y	10.1
	09	Add HL,ss	(HL) = (HL) + (BC)	10.6
	44	LD r,s	(B) = (H)	10.1
	4D	LD r,s	(C) = (L)	10.1
	FD	LD IY,nn	(BC) = (HL)	10.1
ttt	21		(IY) = 0	10.2
	00			
	00			
	FD	ADD IY,rr	Ender do byte do Ficheiro de Imagem e byte (H) = 0	10.6
vvv	09			10.1
	26	LD r,n		10.1
	00			
	3A	LDA,(nn)	(A) = Código	10.1
	cc (Byte inferior de Ender; n)			
	cc (Byte superior de Ender; n)			
	C6	Add A,n	(A) = Código 32	10.4
	E0	(= -32)		
	6F	LD r,s	(L) = Código 32	10.1
	CB	SLA,r	(L) = 2x (código 32)	10.7
	25			
	CB	RL,r	(HL) = 2x (código 32)	10.7
	14			
	CB	SLA,r	(L) = 4x (código 32)	10.7
	25			

Endereço	Byte de memória	Mnemônica	Observações	Tabelas de instruções
	CB	RL,r	(HL) = Código 32	10.7
	14			
	CB	SLA,r	(L) = 8x (código 32)	10.7
	25			
	CB	RL,r	(HL) = 8* Código 32	10.7
	14			
	19	ADD HL,ss	(HL) = (HL) + (DE)	10.6
	01	LD dd,nn		10.2
	00 (Byte inferior de dados)		(BC) = 3D00 (hex)	
	3D (Byte superior de dados)		= 15616 (decimal)	
	09	ADD HL,ss	(HL) = 15616 + 8* (código 32) + (E)	10.6
	7E	LD r,(HL)	(A) = (byte de memória): Ender = (HL)	10.1
	FD	LD(IY+d),r	Guarda (A): Ender d = (IY+d)	10.1
	77			
	00	d		
	3A	LDA,(nn)	(A) = (Contador)	10.1
	mm (Byte superior de Ender; n)			
	mm (Byte superior de Ender; n)			
	3D	DECs	(A) = (A) - 1	10.4
	CA	JP cc,nn	Se (A) = 0 saltar para hhhh, senão otherwise continuar	10.9
	hh (Byte inferior de Ender; n)			
	hh (Byte superior de Ender; n)			
	32	LD(nn),A	(Contador) = (A), isto e, (Contador) = (Contador) - 1	10.1
	mm (Byte inferior de Ender; n)			
	mm (Byte superior de Ender; n)			
	06	LD r,n	(B) = 0	10.1
	00		(BC) = 255	10.1
	0E	LDr,n	(C) = FF	10.1
	FF			
	FD	INC IY	(IY) = (IY) + 1	10.6
	23			
	FD	ADD IY,rr	(IY) = (IY) + (BC)	10.6
	09		ie (IY) = (IY) + 256	
	1C	INC r	(E) = (E) + 1	10.4
	C3	JP nn		10.9
	vv (Byte inferior de Ender; n)		Salto incondicional para vvvv vvv	
	vv (Byte superior de Ender; n)			
pppp	3A	LDA,(nn)	(A) = x	10.1
	aa (Byte inferior de Ender; n)			
	aa (Byte superior de Ender; n)			
	CB	SLA r	(A) = 2*x	10.7
	27			
	CB	SLA r	(A) = 4*x	10.7
	27			
	CB	SLA r	(A) = 8*x	10.7
	27			
	CB	SLA r	(A) = 16*x	10.7
	27			
	CB	SLA r	(A) = 32*x	10.7
	27			
	21	LD dd,nn		10.2
	00 (Byte inferior de dados)		(HL) = 4000 (hex)	
	40 (Byte superior de dados)		= 16384 (decimal)	

Endereço	Byte de memória	Mnemónica	Observações	Tabelas de instruções
	06	LD r,n	(B) = 0	10.1
	00			
	4F	LD r,s	(C) = (A)	10.1
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	3A	LDA,(nn)		10.1
	bb (Byte inferior de Ender; n)		(A) = y	
	bb (Byte superior de Ender; n)			
	4F	LD r,s	(C) = y	10.1
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	44	LD r,s	(B) = (H) } (BC) = (HL)	10.1
	4D	LD r,s	(C) = (L) } -16384 + y + 32*x	10.1
	C3	JP nn		10.9
	tt (Byte inferior de Ender; n)		Salto incondicional para ttt ttt	
	tt (Byte superior de Ender; n)			
qqqq	3A	LDA,(nn)		10.1
	aa (Byte inferior de Ender; n)		(A) = x	
	aa (Byte superior de Ender; n)			
	C6	ADD A,n	(A) = x - 16	10.4
	F0	(= -16)		
	CB	SLA r	(A) = 2*(x - 16)	10.7
	27			
	CB	SLA r	(A) = 4*(x - 16)	10.7
	27			
	CB	SLA r	(A) = 8*(x - 16)	10.7
	27			
	CB	SLA r	(A) = 16*(x - 16)	10.7
	27			
	CB	SLA r	(A) = 32*(x - 16)	10.7
	27			
	21	LD dd,nn		10.2
	00 (Byte inferior de dados)		(HL) = 5000 (hex)	
	50 (Byte superior de dados)		= 20480 (decimal)	
	06	LD r,n	(B) = 0	10.1
	00			
	4F	LD r,s	(C) = 32*(x - 16)	10.7
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	3A	LDA,(nn)		10.1
	bb (Byte inferior de Ender; n)		(A) = y	
	bb (Byte superior de Ender; n)			
	4F	LD r,s	(C) = y	10.1
	09	ADD HL,ss	(HL) = (HL) + (BC)	10.6
	44	LD r,s	(B) = (H) } (BC) = (HL)	10.1
	4D	LD r,s	(C) = (L) } -20480 + y + 32*(x - 16)	10.1
	C3	JP nn		10.9
	tt (Byte inferior de Ender; n)		Salto incondicional para ttt ttt	
	tt (Byte superior de Ender; n)			
hhhh	C9	RET	Retorno	10.10

Exercício

Usando o Programa 6 do Apêndice 6, baixe a RAMTOP para um endereço apropriado e carregue o programa em código-máquina que acabámos de listar. Não se esqueça de usar os endereços hexadecimais correctos relativamente à RAMTOP, e de os indicar nos pontos apropriados da listagem. Inicialize os valores de x, y, Código e Número de Atributo do Caracter, e em seguida faça executar o programa em código-máquina usando

```
5 GO TO USR RAMTOP + 6
```

```
7 STOP
```

onde RAMTOP é o valor numérico que usou no programa de carga.

Programa de entrada/saída

O principal objectivo deste exemplo de programação em código-máquina consiste em demonstrar o modo como é possível usar o microprocessador Z80A para realizar uma operação de entrada/saída ao nível de código-máquina. Consegue-se isto usando a interface "memory-mapped" de um byte descrita no capítulo 9. Por outro lado, neste exemplo, usa-se o microprocessador. É usado para determinar o número de bits no conjunto de bytes de entrada iguais a 1, e em seguida apresenta o resultado sob a forma de um número binário através do port de entrada/saída.

O fluxograma deste exemplo é apresentado na figura 10.8. A primeira parte do processo envolve a definição do conteúdo do par de registos BC, igual ao endereço da interface "memory-mapped" de um byte. O passo seguinte deste processo consiste em dar entrada ao byte de dados da interface para o acumulador A. Usa-se o Registo D como contador, sendo o seu valor inicial igual a zero. A instrução de deslocamento para a esquerda move o conteúdo do acumulador A uma posição para a esquerda, passando o bit mais significativo do byte de dados para o bit de transporte e tornando-se o bit menos significativo igual a zero. Consequentemente, se esta acção passa ao nível 1

a *flag C* o contador é incrementado de um, senão não o é. Quando o conteúdo do acumulador A é zero, isto é, nenhum dos seus bits é igual a 1, o processo de deslocamento é terminado e o valor binário guardado no registo D é apresentado utilizando os LED's da interface.

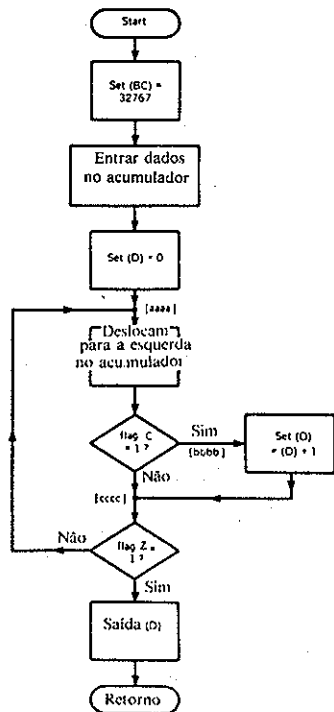


Fig. 10.8 — Fluxograma de um programa de entrada/saída em código-máquina.

O programa em código para este processo é listado em seguida; note que é conveniente exprimir os endereços do programa relativamente à RAMTOP, RT. Os endereços gerais usados na listagem possuem os seguintes valores relativos:

aaaa RT + 9
 bbbb RT + 20
 cccc RT + 14

Note que para as operações de entrada/saída o comutador de selecção da interface, SW, deve ser passado para a posição 2 (ver a figura 9.15).

Endereço	Byte de memória	Mnemónica	Observações	Tabelas de instruções
RT + 1	06 7F 0E FF ED 78 16 00	LD r,n LD r,n IN r,(C) LD r,n	(B) = 7F (C) = FF (A) = Dado de entrada (D) = 0	(BC) = 32767 10.1 10.1 10.11 10.1
aaaa	CB 27 DA	SLAs JP cc,nn	2° (A)	10.7
cccc	bb bb	JP cc,nn	Se C = 1 saltar para bbbb. senão continuar	10.9
	aa aa			
bbbb	ED	OUT (C),r	Apresentar resultado	10.11
	51	RET	Retorno	10.9
	C9	INC r	(D) = (D) + 1	10.4
	14 C3 cc cc	JP nn	Salto incondicional para cccc	10.9

Em particular, quando a RAMTOP é igual a 32300, os endereços usados pelo programa são

	Decimal	Hexadecimal
aaaa	32309	7E35
bbbb	32314	7E3A
cccc	32320	7E40

Exercício

Usando o Programa 6 do Apêndice A, defina um valor

apropriado para a RAMTOP e carregue o programa anterior acima dela. Não se esqueça de usar os endereços hexadecimais correctos relativamente à RAMTOP, e indique-os nos pontos apropriados da listagem do programa. Execute-o fazendo

```
200 LET A = USR RAMTOP + 1
210 GO TO GO 200
```

onde RAMTOP é o valor numérico que usou no programa de carga.

Rotina de produção de som: CALL 03B5

Um programa em código-máquina pode aceder a rotinas Basic presentes em ROM, através do uso da instrução CALL seguida do endereço da rotina a executar. A rotina de produção de sons, por exemplo, é acedida através de CALL 03B5

Antes de chamar esta rotina é necessário carregar o par de registos DE com o número que define a duração do som produzido, e o par de registos HL com o número que define a tonalidade da nota.

Os seguintes dez bytes de código-máquina permitirão usar esta rotina.

RAMTOP + 1	11	LD DE,nn
	n	Byte inferior do valor da duração
	n	Byte superior do valor da duração
	21	LD HL,nn
RAMTOP + 5	n	Byte inferior da tonalidade
	n'	Byte superior da tonalidade
	CD	CALL
	B5	Byte inferior do endereço da rotina
	03	Byte superior do endereço da rotina
RAMTOP + 10	C9	RET

Exercício

Usando o programa 6 do Apêndice A, indique um valor apropriado para a RAMTOP e carregue o anterior programa em código-máquina. Utilize 05FF para a duração e 00C8 para

a tonalidade. Execute o programa fazendo

```
135 LET x = USR RAMTOP + 1
```

onde RAMTOP é o valor numérico usado no programa de carga.

Introduza na máquina e faça executar o seguinte programa, observando o efeito da alteração do tom da nota produzida.

```
300 POKE RAMTOP + 5,100
350 LET x = USR RAMTOP + 1
375 POKE RAMTOP + 5,255
400 LET x = USR RAMTOP + 1
425 GO TO 300
```

Observações finais

Neste capítulo apresentaram-se os conceitos básicos de programação do microprocessador Z80A usando técnicas de programação em código-máquina. Examinámos os modos de endereçamento e as instruções disponíveis no Z80A, mostrando o seu uso em alguns exemplos simples. Recomendamos ao leitor que estude os exemplos ilustrativos, que foram incluídos para o auxiliar a compreender o grupo principal de registos do microprocessador Z80A, os seus modos de endereçamento, e o modo de realizar a transferência de dados de entrada/saída.

APÊNDICE A

PROGRAMAS

Programa 1: Conversão de Números

Este programa pode ser usado para converter números inteiros positivos, binários e hexadecimais, para os respectivos equivalentes decimais. Pode ser igualmente usado para converter números decimais inteiros positivos em equivalentes binários ou hexadecimais de vinte e cinco algarismos (para converter para binário o maior número será 33554431, e para hexadecimal aproximadamente $1,2676506E + 30$).

```
20 CLS
```

```
25 PRINT "Este programa converte:
```

1. Binário para decimal
2. Hexadecimal para decimal
3. Decimal para binário
4. Decimal para hexadecimal

Escolha 1, 2, 3 ou 4"

```
45 INPUT N
```

```
55 IF N = 1 THEN GO TO 105
```

```
65 IF N = 2 THEN GO TO 205
```

```
75 IF N = 3 THEN GO TO 300
```

```
85 IF N = 4 THEN GO TO 405
```

```
95 GO TO 20
```

```
105 PRINT
```

```
108 PRINT "Escreva número binário"
```

```
109 LET radix = 2
```

```
110 INPUT B$
```

```
115 LET P = 1
```

```
120 LET R = 0
```

```
125 LET L = LEN B$
```

```
130 FOR K = L TO 1 STEP -1
```

```
131 LET j = 0
```

```
133 IF CODE B$(K) > 57 THEN LET i = 7
```

```
135 LET a = (CODE (B$(K)) - 48 - j) * P
```

```
145 LET P = P * radix
```

```
155 LET R = R + a
```

```
165 NEXT K
```

```
175 PRINT
```

```
176 IF radix = 16 THEN GO TO 225
```

```
178 PRINT "O número binário"; B$; "converte em"; R
```

```
200 GO TO 500
```

```
205 PRINT
```

```
208 PRINT "Escreva número hexadecimal"
```

```
209 LET radix = 16
```

```
210 GO TO 110
```

```
225 PRINT "O número hexadecimal"; B$; "converte em"; R
```

```
230 GO TO 500
```

```
300 LET b = 2
```

```
305 DIM f$(25)
```

```
306 PRINT
```

```
307 PRINT "Escreva número decimal"
```

```
310 INPUT S: LET d = S
```

```
315 FOR i = 1 TO 25
```

```
320 LET x = d / b
```

```
325 LET r = d - (INT (x) * b)
```

```
326 IF r > 9 THEN GO TO 420
```

```
330 LET f$(i) = CHR$(48 + r)
```

```
335 LET d = INT (x)
```

```
345 NEXT i
```

```
355 PRINT
```

```
365 PRINT "O número decimal"; S "converte em"
```

```
375 FOR i = 25 TO 1 STEP -1
```

```
385 PRINT f$(i);
```

```

395 NEXT i
400 GO TO 500
405 LET b = 16
410 GO TO 305
420 LET r = r + 7: GO TO 330
500 PRINT : PRINT
520 PRINT "Escreva R para recomençar e P para parar"
555 INPUT K$
565 IF K$ = "P" THEN STOP
666 IF K$ = "R" THEN GO TO 20

```

Programa 2: Cronómetro

Este programa permite obter um relógio de 24 horas e um cronómetro. O utilizador indica o tempo inicial e em seguida carrega em S para iniciar a contagem do tempo; carregando em H pára a contagem. O tempo em horas, minutos e segundos é apresentado durante toda a execução. A diferença entre o tempo inicial e o tempo final tempo (decorrido) é apresentada quando o relógio é parado.

```

3 DIM a$(7)
5 PRINT "Indique hora inicial"
15 INPUT h: PRINT AT 0,22; "=" ; h
18 LET h0 = h
25 PRINT "Indique minuto inicial"
35 INPUT m: PRINT AT 1, 22; "=" ; m
37 LET m0 = m
45 PRINT "Indique segundo inicial"
55 INPUT sec: PRINT AT 2,22; "=" ; sec
57 LET sec0 = sec
60 IF NOT (h <= 23 AND m <= 59 AND sec <= 59) THEN GO
    TO 400
64 PRINT "Carregue em S para comeenar, H para parar" fold
    Down Key H"
66 IF INKEY$ < > "S" THEN GO TO 66
75 CLS

```

```

95 PRINT "Horas Minutos Segundos"
05 LET X = 16
15 LET S = X
45 LET Y = 3
55 LET X = X - 1
65 IF X < > 0 THEN GO TO 195
175 LET X = S
185 LET Y = Y - 1
195 IF Y < > 0 THEN GO TO 155
200 PAUSE 2
202 LET sec = sec + 1
205 IF sec = 60 THEN LET m = m + 1
210 IF sec = 60 THEN PRINT AT 1,18;a$
215 IF sec = 60 THEN LET sec = 0
225 IF m = 60 THEN LET h = h + 1
230 IF m = 60 THEN PRINT AT 1,8;a$
235 IF m = 60 THEN LET m = 0
240 IF h = 24 THEN PRINT AT 1,0;a$
245 IF h = 24 THEN LET h = 0
255 LET a = 1
340 IF h < 10 THEN LET a = 2
342 PRINT AT 1,a;h
343 LET b = 10
346 IF m < 10 THEN LET b = 11
347 PRINT AT 1,b;m
349 LET c = 20
351 IF sec < 10 THEN LET c = 21
353 PRINT AT 1,c;sec
360 IF INKEY$ = "H" THEN GO TO 365
363 GO TO 105
365 PRINT AT 10,0; AT 10,0; "Liberte a tecla H"
370 IF INKEY$ < > " " THEN GO TO 370
375 GO SUB 500
380 STOP
400 CLS
410 GO TO 5
500 LET h0 = h h0
510 LET m0 = m m0
515 LET sec0 = sec sec0

```

```

520 LET t = h0*3600 + m0*60 + sec0
540 LET h1 = INT (t/3600)
545 LET m1 = INT ((t - h1*3600)/60)
555 LET sec1 = t - m1*60 - h1*3600
565 PRINT AT 5,0; "Tempo decorrido"
570 LET a = 1
572 IF h1 < 10 THEN LET a = 2
575 PRINT AT 7,a;h1
577 LET b = 10
585 IF m1 < 10 THEN LET b = 11
590 PRINT AT 7,b;m1
600 LET c = 20
610 IF sec1 < 10 THEN LET c = 21
620 PRINT AT 7,c;sec1
630 RETURN

```

Note que neste programa o menor período de tempo usado pelo cronómetro é de aproximadamente um segundo. Este período pode ser alterado modificando a duração da pausa na linha 200 do programa, ou alterando os valores de x e y nas linhas 105 e 145 respectivamente. Experimente por exemplo x = 15, y = 3 e PAUSE 1.

Programa 3: Apresentação dos caracteres em ROM

No capítulo foi descrita a forma de guardar os caracteres na ROM, e na fotografia 7.3 apresentou-se o padrão de 8 x 8 bits e a correspondente definição de 8 x 8 pixels, com um aumento de 64 vezes, para o carácter ©. O programa listado em seguida permite obter o mesmo resultado.

Podem-se usar neste programa caracteres com códigos entre 32 e 127 inclusivé (ver a tabela 7.1), apresentando o resultado sob a forma indicada na fotografia 7.3.

```

4 PAPER 6
5 INPUT code
12 PRINT AT 5,6; "O caracter é"; CHR$ code
25 FOR x = 1 TO 8

```

```

30 LET j = 115615 + 8*(code - 32) + x
32 LET g = PEEKj
40 GOSUB 150
50 NEXT x
150 FOR n = 1 TO 8
155 LET y = g/2
165 LET r = g - (INT y*2)
175 LET a$ = CHR$ (48 + r)
177 PRINT AT 7 + x,12 - n;a$
178 IF a$ = "0" THEN LET a$ = " "
179 IF a$ = "1" THEN LET a$ = "■"
180 PRINT AT 7 + x,28 - n;a$
185 LET g = INT y
195 NEXT n
200 IF x = 8 THEN PAPER 4
210 IF x = 8 THEN STOP
225 RETURN

```

O programa pode ser modificado de modo a obter a imagem do padrão de 8 x 8 bits e da definição da rede de pixels de um carácter indicado através do teclado. Neste caso o carácter indicado pelo teclado é convertido no código respectivo usando a função INKEY\$. Para tal devem-se alterar as linhas 5,12 e 30 do programa para

```

5 IF INKEY$ " " THEN GO TO 5
12 PRINT AT 5,6; "O caracter é"; Z$
30 LET i = 15615 + 8*(CODE Z$ - 32) + x
e acrescentar as linhas adicionais
3 PAUSE 10
10 LET Z$ = INKEY$

```

Note que esta forma modificada do programa não permite dar entrada a todos os caracteres na gama 32 a 127. Por exemplo o carácter (código 125) está excluído. No entanto, a principal vantagem do uso do programa nesta forma modificada consiste em permitir-lhe dar entrada directamente à maior parte dos caracteres sem necessidade de procurar os respectivos códigos na tabela 7.1.

Programa 4: Conteúdo da ROM

Este programa pode ser usado para imprimir os códigos decimais guardados na ROM. O valor de 'a' definido na linha 5 corresponde ao primeiro endereço da ROM a listar, e o valor de 'a' definido na linha 87 indica o último endereço da ROM a listar. No caso da listagem apresentada em seguida, com os valores a = 0 (linha 5) e a = 16384 (linha 87), todo o conteúdo da ROM será listado. Para examinar pequenas parcelas da ROM bastará definir de outro modo os endereços inicial e final nas linhas 5 e 87 respectivamente.

Na linha 77 usa-se a função LPRINT pelo que, se o leitor não tiver uma impressora, altere esta palavra para PRINT e verá o conteúdo da ROM ser impresso no visor (ver a fotografia A.4).

```

5 LET a = 0
8 DIM h$(1,24)
15 LET d = 0
17 LET x = PEEK a
20 FOR i = 2 TO 1 STEP -1
30 LET h = x/16
40 LET r = x - INT (h)*16
50 LET h$(1,i + d) = CHR$(48 + r)
55 IF r > 9 THEN LET h$(1,i + d) = CHR$(55 + r)
60 LET x = INT h
70 NEXT i
72 LET d = d + 3
77 IF d = 24 THEN LPRINT h$(1);" "; a - 3
80 IF d = 24 THEN GO TO 15
85 LET a = a + 1
87 IF a = 16384 THEN STOP
90 GO TO 17

```

Ao executar o programa verificará que o conteúdo da ROM é indicado em hexadecimal, cabendo oito posições sucessivas de memória em cada linha e terminando pela indicação do primeiro endereço dessa linha. Em seguida apresentamos um pouco da listagem produzida pela máquina.

```

F3 AF 11 FF FF C3 CB 11 0
11 2A 5D 5C 22 5F 5C 18 7
18 43 C3 F2 15 FF FF FF 14
FF FF FF 2A 5D 5C 7E CD 21
CD 7D 00 D0 CD 74 00 18 28
18 F7 FF FF FF C3 5B 33 35
33 FF FF FF FF FF C5 2A 42
2A 61 5C E5 C3 9E 16 F5 49
F5 E5 2A 78 5C 23 22 78 56
78 5C 7C B5 20 03 FD 34 63
34 40 C5 D5 CD BF 02 D1 70
D1 C1 E1 F1 FB C9 E1 6E 77
6E FD 75 00 ED 7B 3D 5C 84
5C C3 C5 16 FF FF FF FF 91
FF FF FF FF F5 E5 2A B0 98

```

Programa 5: Descubra o Tesouro

Este programa é um jogo. Foi escondido um tesouro numa das posições da rede de caracteres do visor. O leitor deverá tentar descobrir o tesouro deslocando o carácter (gráfico D) para cima e para baixo (teclas u e d respectivamente), e para a esquerda ou para a direita (teclas l e r), até o carácter se encontrar no quadrado onde foi escondido o tesouro. A máquina imprime então o número de movimentos necessários para descobrir o tesouro.

Note que no início do programa se encontram dois blocos de linhas, tendo estas a numeração 1 a 8 em ambos os casos. O bloco a usar depende de se ter um Spectrum de 16 ou de 48 K.

```

1 POKE 32624,BIN 00001000
2 POKE 32625,BIN 00010100
3 POKE 32626,BIN 00100010
4 POKE 32627,BIN 01000001
5 POKE 32628,BIN 00100010
6 POKE 32629,BIN 00010100
7 POKE 32630,BIN 00001000
8 POKE 32631,BIN 00000000

```

} 16K Spectrum

```

1 POKE 65392,BIN 00001000
2 POKE 65393,BIN 00010100
3 POKE 65394,BIN 00100010
4 POKE 65395,BIN 01000001
5 POKE 65396, BIN 00100010
6 POKE 65397, BIN 00010100
7 POKE 65398, BIN 00001000
8 POKE 65399, BIN 00000000
10 BORDER 5: PAPER 6: INK 0: FLASH 0
12 LET n = 0
14 CLS
25 LET r INT (RND*20)
35 LET c = INT (RND*31)
45 LET x = INT (RND*20)
55 LET y = INT (RND*31)
60 IF x = r AND y = c THEN GO TO 45
65 PRINT AT x,y; "◇"
85 LET k$ = INKEY$
90 IF k$ = " " THEN GO TO 85
91 IF NOT (k$ = "u" OR k$ = "d" OR k$ = "l" OR k$ = "r") THEN GO TO 85
92 IF INKEY$ < > " " THEN GO TO 92
93 PRINT AT x,y; " "
95 IF k$ = "u" THEN LET x = x - 1
115 IF x < 0 THEN LET x = 0
125 IF k$ = "d" THEN LET x = x + 1
135 IF x > 20 THEN LET x = 20
145 IF k$ = "l" THEN LET y = y - 1
155 IF y < 0 THEN LET y = 0
165 IF k$ = "r" THEN LET y = y + 1
175 IF y > 31 THEN LET y = 31
225 PRINT AT x,y; "◇"
235 IF x = r AND y = c THEN GO TO 350
240 LET n = n + 1
250 PRINT AT 21,0; "Número de movimentos ="; n
260 GO TO 85
350 FOR g = 10 TO 0 STEP - 1
355 BEEP. 1,g
365 NEXT g
370 FLASH 1: PAPER 4

```

48K Spectrum

```

375 CLS: PRINT AT 10,10; "ÓPTIMO!"
385 PRINT AT 12,3; "Tesouro descoberto em"; n;
    "movimentos"
390 PRINT AT 16,4; "Carregue em G para novo jogo"
400 PRINT AT 18,4; "ou em S para parar"
420 IF NOT (INKEY$ = "G" OR INKEY$ = "S")
    THEN GO TO 420
425 IF INKEY$ = "G" THEN GO TO 10
430 IF INKEY$ = "S" THEN STOP

```

Programa 6: Programa para carregar código-máquina acima da RAMTOP

Este programa pede-lhe que indique, sob a forma de número decimal, o endereço da RAMTOP. Esta é em seguida definida usando a declaração CLEAR da linha 75, que possui o formato

CLEAR valor da RAMTOP

Em seguida dá-se entrada a cada byte do programa em código-máquina sob a forma de dois caracteres hexadecimais, que serão armazenados em posições sucessivas da memória começando no endereço que se segue à RAMTOP. Depois de dar entrada ao último byte do programa, escreva Z para terminar a operação de carga. Para aceder e executar o programa em código-máquina use a função USR seguida pelo endereço inicial que é, evidentemente, RAMTOP mais um.

```

10 PRINT "Indique endereço da RAMTOP"
20 INPUT RT
25 PRINT RT
28 LET A = RT
30 PRINT "Escreva byte de código" "ou Z para parar"
35 INPUT H$
36 PRINT H$
37 IF H$ = "Z" THEN GO TO 75
38 LET RT = RT + 1
40 LET x CODE H$
45 IF x > 57 THEN LET x = x - 7

```

```
50 LET y = CODE H$(2)
55 IF y > THEN LET y = y - 7
60 POKE RT,x*16 + y - 816
70 GO TO 30
75 CLEAR A
```

APÊNDICE B

GLOSSÁRIO DE TERMOS

ACUMULADOR — Um registo que pode ser usado como fonte de um operando e destino do resultado de operações.

ALFANUMÉRICO — Refere-se a caracteres alfabéticos e numéricos.

ARGUMENTO — Um valor numérico usado numa declaração de programa.

ARQUITECTURA — A estrutura de um programa.

ASCII — (American National Standard Code for Information Interchange). O código standardizado, usando um conjunto de caracteres codificado, que é utilizado para troca de informações entre sistemas de processamento de dados, equipamento de comunicações e outro equipamento associado.

ASSEMBLER — Um programa de computador usado para traduzir instruções mnemónicas nos códigos binários que lhes correspondem e para atribuir posições de memória para dados e instruções.

ATRIBUTOS — Código de elementos da rede de caracteres do visor que define as cores do papel e da tinta para impressão dos caracteres.

BASIC — (Beginners All-Purpose Symbolic Instruction Code) — Uma linguagem de alto nível, muito usada em microcomputadores

BAUD — Unidade da velocidade de transmissão de informação em bits por segundo.

BINÁRIO — Um sistema de numeração com base 2.

BIT — Um algarismo binário.

BIT DE PARIDADE — Um bit do estado que normalmente se encontra ao nível lógico 1 quando a última operação deu um resultado com:

a) uma paridade par, se se utiliza esta paridade, ou (b) uma paridade ímpar se se usa paridade ímpar.

BIT DE SINAL — O bit mais significativo de uma palavra de dados que indica o sinal destes. Usa-se o nível lógico 0 para indicar um número positivo e o nível 1 para indicar um número negativo.

BIT DE TRANSPORTE — O bit usado para indicar a ocorrência de um transporte do bit mais significativo de uma palavra.

BUG — Um erro num programa.

BUS — Condutores paralelos que ligam dois ou mais dispositivos.

BUS BI-DIRECCIONAL — Um bus ao longo do qual os sinais podem viajar em ambos os sentidos.

BYTE — Grupo de oito bits considerado pelo microprocessador como uma palavra binária.

CADEIA NULA — Uma cadeia que não contém quaisquer caracteres.

CICLO — Uma sequência de instruções a repetir pelo microcomputador.

CICLO DE INSTRUÇÃO — Ciclo de recolha, descodificação e execução de uma instrução.

CICLO-MÁQUINA — Ciclo básico do microprocessador. É o tempo requerido para recolher dados da memória ou para executar uma instrução de um byte.

CICLO-T — Ciclo de temporização do microprocessador Z80A.

CÓDIGO DE OPERAÇÃO — A parte de uma instrução em código-máquina usada para especificar a operação a realizar durante o ciclo que se segue.

CÓDIGO-MÁQUINA — A linguagem que o microcomputador pode interpretar directamente.

COMPLEMENTO PARA DOIS — Complemento para um de um número binário mais um.

COMPLEMENTO PARA UM — O complemento bit a bit de um número binário.

COMPLEMENTO DE PALAVRA — O número de bits da palavra de um microprocessador.

CONDICIONAMENTO DE SINAL — Alteração de um sinal para torná-lo compatível com um dado dispositivo.

CONJUNTO DE CARACTERES — Todos os caracteres acei-

tes pelo teclado.

CONJUNTO DE INSTRUÇÕES — Conjunto de instruções que podem ser interpretadas pelo microprocessador.

CONTADOR — Um dispositivo que muda de estado quando lhe é aplicado um impulso de relógio. Normalmente a sua saída indica o número total de impulsos de relógios recebidos (até à sua capacidade máxima).

CONTADOR DE PROGRAMA — Um registo que armazena o endereço da instrução a executar a seguir.

DEBUG — Eliminação dos erros de um programa.

DECIMAL EM CODIFICAÇÃO BINÁRIA — Um código em que cada algarismo decimal é codificado usando quatro algarismos binários.

DESLOCAMENTO ARITMÉTICO — Uma operação de deslocamento (para a esquerda ou direita) em que se mantém o valor do bit de sinal.

DISPOSITIVOS MOS — Elementos semicondutores que utilizam transistores de efeito de campo.

DISPOSITIVOS CMOS — Elementos lógicos complementares construídos com transistores de efeito de campo de canal n ou p de modo a obter dispositivos com baixo consumo e alta imunidade ao ruído.

DUMP — Transferência de dados para uma memória de apoio, para o ficheiro de imagem ou cópia em papel.

EAROM — Memória de leitura apenas alterável electricamente

EEROM } Memória de leitura apenas, apagável

E2PROM } electricamente

ENDEREÇO — Localização codificada de um byte de memória.

EPROM — Memória de leitura apenas programável

ESTOIRO — “Crash” do computador; perda de comando do programa.

ETIQUETA — Um nome dado a uma instrução ou declaração num programa a fim de a identificar.

FIELD PROGRAMMABLE ROM — Uma memória apenas de leitura que pode ser programada pelo utilizador.

FIRMWARE — Microprogramas existentes em memória apenas de leitura.

FLAG — Um *flip-flop* que se encontra normalmente ao nível lógico 1 depois da ocorrência de um determinado acontecimen-

to.

FLUXOGRAMA — Uma representação gráfica de um conjunto de instruções de microcomputador.

FUNCIONAMENTO ASSÍNCRONO — Funcionamento sem utilizar uma referência de tempo.

FUNCIONAMENTO SINCRONO — Funcionamento em períodos regulares de tempo relativamente a um tempo de referência.

HARDWARE — Aparelhos físicos que constituem o microcomputador.

HEXADECIMAL — Um sistema de numeração na base 16. Utiliza os caracteres alfanuméricos 0 a 9 e A a F.

IMAGEM INVERTIDA (INVERSE VIDEO) — Apresentação de um carácter de tal modo que as cores de papel e de tinta se encontram trocadas.

INSTRUÇÃO — Um grupo de bits usado para especificar uma operação do microprocessador.

INTEIRO — Um número positivo ou negativo sem parte fracionária.

INTERFACE — Fronteira entre o computador e os seus periféricos.

INTERRUPÇÃO — Uma entrada do microprocessador que transfere temporariamente o comando do programa principal para uma rotina de interrupção independente.

INTERRUPÇÃO DE SOFTWARE — Uma instrução que leva um programa a saltar para um endereço específico.

INTERRUPÇÃO MASCARÁVEL — Uma interrupção que pode ser contrariada pelo sistema.

INTERRUPÇÕES PRIORITÁRIAS — Interrupções que podem ser obedecidas antes de outras, ou podem interromper outras.

KBYTES — 1024 bytes.

LIMPAR (CLEAR) — Uma entrada de um dispositivo que conduz o seu estado ao nível lógico 0.

LINGUAGEM ASSEMBLY — Uma linguagem em que se utilizam instruções mnemónicas, etiquetas e nomes. Estas podem ser traduzidas por um "assembler" directamente em código-máquina.

LINGUAGEM DE ALTO NÍVEL — Uma linguagem de computador cujas declarações representam procedimentos (*proce-*

dures) e não simples instruções-máquina.

MAPA DE MEMÓRIA — Um método gráfico de ilustrar secções de memória.

MÁSCARA — (a) um padrão de bits que separa um ou vários bits de outros; (b) uma chapa fotográfica usada no fabrico de circuitos integrados para definir padrões de difusão de impurezas.

MÁSCARA DE INTERRUPÇÕES — Uma técnica que permite ao microprocessador especificar as interrupções que serão aceites.

MEMÓRIA — Um elemento que pode armazenar bits lógicos.

MEMÓRIA DE APOIO (BACKING STORE) — Um meio de grande capacidade para guardar informação, por exemplo uma cassette de fita.

MEMÓRIA DINÂMICA — Uma memória que lentamente perde o seu conteúdo, necessitando portanto de ser "refrescada".

MEMÓRIA ESTÁTICA — Uma memória que não necessita de ser "refrescada".

MEMÓRIA VOLÁTIL — Uma memória que perde o seu conteúdo quando se desliga a alimentação.

MICROCOMPUTADOR — O conjunto de um microprocessador, elementos de memória e elementos de interface.

MICRO-INSTRUÇÃO — Uma das sequências organizadas de sinais de comando que constituem instruções ao nível de comando.

MICROPROCESSADOR — A unidade central de processamento de um microcomputador.

MNEMÓNICA — Nome ou abreviatura simbólica.

MODEM — Um elemento modulador/desmodulador que utiliza a frequência de uma portadora para permitir comunicações num canal de alta frequência.

MODOS DE ENDEREÇAMENTO — Modos que especificam os endereços em memória ou os registos de microprocessador a usar em cada instrução.

MONITOR — Um sistema operativo simples que permite ao utilizador dar entrada e fazer executar programas.

MULTIPROCESSAMENTO — Uso de mais de um microprocessador num mesmo sistema.

NIBBLE — Um grupo de quatro bits.

OFFSET — Um número que é acrescentado a outro para formar um endereço útil.

PALAVRA — O grupo de bits que o microprocessador pode manipular.

PARIDADE — Um código de um bit que é acrescentado a uma palavra de modo a tornar o número total de bits “um” par (paridade par) ou ímpar (paridade ímpar).

PASTILHA INTEGRADA — O suporte de um circuito integrado

PEEK — Uma instrução de Basic usada para examinar o conteúdo de um endereço de memória.

PIXEL — Um elemento de imagem.

POINTER (INDICADOR) — Um registo ou posição de memória que contém um endereço.

POKE — Uma instrução de Basic usada para escrever dados numa posição de memória.

POLLING — Exame sucessivo do estado dos periféricos.

POP — Remoção de um operando do “stack” da máquina.

PORT DE ENTRADA/SAÍDA — Uma ligação ao microcomputador que pode ser programada para transferência de dados através de um interface.

PROGRAMA — Uma sequência de instruções correctamente ordenadas e que executam uma dada tarefa.

PROM — Memória programável apenas de leitura.

PSEUDO-ALEATÓRIO — Acontecimento que parece aleatório, mas não o é de facto.

PUSH — Guardar um operando no “stack” da máquina.

QUADRO (ARRAY) — Um conjunto de variáveis sendo cada uma destas identificada por um único carácter representando o quadro e por índices representando as respectivas posições dentro do quadro.

RAM — Uma memória que pode ser lida e alterada livremente. É designada por Memória de Acesso Aleatório.

REFRESCAR — Processo de restauração do conteúdo de uma memória dinâmica.

REGISTO — Um grupo de células de memória usado para armazenar palavras no interior do microprocessador.

REGISTO DE INDEXAÇÃO — Um registo do microprocessador que é usado para armazenar endereços de memória.

RELÓGIO — Um sinal periódico de temporização usado para

controlar o sistema operativo.

RELÓGIO EM TEMPO REAL — Um elemento que interrompe um microprocessador em intervalos de tempo regulares.

ROM — Memória apenas de leitura.

ROTINA DE INTERRUPTÃO — Um programa que é executado em resposta a um sinal de interrupção.

RS232 — Um interface “standard” para comunicações seriais entre computadores e dispositivos periféricos.

SALTO CONDICIONAL — Uma instrução que provoca um salto para uma parte diferente de um programa quando é verdadeira uma condição especificada.

SCROLL — Movimento da informação presente no visor de modo a ser substituída por outra.

SINTAXE — Regras de estrutura das declarações numa linguagem de programação.

SOFTWARE — Programas para o computador.

STACK — Pilha de elementos de memória RAM que são normalmente acedidos de tal modo que o primeiro a retirar da pilha foi o último nela guardado.

STACK POINTER — Um registo usado para endereçar a posição de “stack” seguinte.

SUBROTINA — Um subprograma que pode ser acedido a partir de diferentes locais de um programa principal.

TABELA DE VERDADE — Tabela que indica os estados de entrada e de saída de uma operação lógica.

TEMPO DE ACESSO — Tempo entre a recepção de um endereço por um elemento de memória e o momento em que os dados presentes nesse endereço surgem na saída.

TEMPO DE ATRASO — Tempo entre um sinal de pedido de informação e o aparecimento da correspondente resposta na saída.

TEMPO DE EXECUÇÃO DA INSTRUÇÃO — Tempo requerido para recolher, descodificar e executar uma instrução.

TERMINAL — Um dispositivo de entrada/saída usado para comunicar com um sistema microprocessador.

TTL — Lógica Transistor-Transistor. Tecnologia bipolar muito usada em circuitos integrados.

ULA (Uncommitted Logic Array) — Usado para realizar funções lógicas complexas.

UNIDADE LÓGICA E ARITMÉTICA (ALU) — Um elemen-

to que pode realizar várias funções aritméticas e lógicas.

VARIÁVEIS DE SISTEMA — Bytes guardados em RAM identificados por uma mnemónica e usados pelo computador para tarefas específicas.

VARIÁVEL DE CADEIA — Uma variável, consistindo num único carácter alfabético seguido do símbolo \$, à qual é possível atribuir uma cadeia (*string*).

APÊNDICE C

VARIÁVEIS DE SISTEMA

A área de memória em RAM com endereços entre 23552 e 23733 é usada pelo interpretador Basic e pelo sistema operativo do ZX Spectrum para guardar os valores das **variáveis de sistema**. Em seguida apresenta-se um resumo destas variáveis e dos respectivos endereços. Note que no caso das variáveis que ocupam dois bytes o primeiro contém sempre o byte menos significativo, e o segundo o mais significativo.

Endereço em RAM	Nome da variável	Comentários
23552	KSTATE	8 bytes. Usada em leituras de teclado.
23560	LAST K	1 byte. Guarda o código da última tecla usada.
23561	REPDEL	1 byte. Tempo em 1/50 de segundo durante o qual qualquer tecla pode estar carregada antes de repetir. Possui à partida o valor 35, mas este pode ser alterado com POKE's.
23562	REPPER	1 byte. Atraso de tempo entre repetições sucessivas de uma tecla. De início possui o valor 5, e o atraso é medido em 1/50

23563	DEADD	de segundo. 2 bytes. Guarda endereços de argumentos de funções definidas pelo utilizador quando usadas. Se não possui o valor zero.	23613	ERR SP	2 bytes. Endereço do elemento do "stack" usado como retorno de erro. Não executar POKE's.
			23615	LIST SP	2 bytes. Contém endereço de retorno em listagem automática.
23565	KDATA	1 byte. Guarda o segundo byte dos comandos de cor indicadas por teclado.	23617	MODE	1 byte. Especifica o cursor K, L, G, C, E.
23566	TVDATA	2 bytes. Guarda bytes de cor, comandos AT e TAB indo para o televisor.	23618	NEWPPC	2 bytes. Linha para onde o programa deve saltar.
			23620	NSPPC	1 byte. Contém número da declaração de uma linha para onde o programa deve saltar.
23568	STRMS	38 bytes. Guarda os endereços dos canais ligados a "streams". Não se deve usar a função POKE sobre esta variável, porque o sistema pode estoirar.	23621	PCC	2 bytes. Contém o número de linha da instrução em execução.
23606	CHARS	2 bytes. 256 menos do que o endereço do conjunto de caracteres.	23623	SUBPPC	1 byte. Número da declaração de uma linha que está em execução.
23608	RASP	1 byte. Define a duração do sinal de excesso de comprimento de linha.	23624	BORDCR	1 byte. Atributos da margem.
			23625	EPPC	2 bytes. Número da linha actual.
23609	PIP	1 byte. Define a duração do som produzido pelas teclas.	23627	VARs	2 bytes. Contém o endereço das variáveis. Não executar POKE's.
23610	ERRNR	1 Byte. 1 menos do que o código das mensagens.	23629	DEST	2 bytes. Contém o endereço da variável em uso.
23611	FLAGS	1 byte. <i>Flags</i> que comandam o Basic. Não executar POKE's.	23631	CHANS	2 bytes. Contém o endereço dos dados de canal. Não executar POKE's.
23612	TV FLAG	1 byte. Flags relativas à TV. Não executar POKE's.	23633	CURCHL	2 bytes. Endereço da informação em uso para entradas/saídas. Não executar POKE's.
			23635	PROG	2 bytes. Contém endereço inicial do programa Basic. Não executar PO-

23637	NXTLIN	KE's. 2 bytes. Contém o endereço da linha seguinte do programa. Não executar POKE's.	23658 23659	FLAGS 2 DF SZ	1 byte. Flags. 1 byte. Número de linhas na parte inferior do visor (incluindo uma em branco). Não executar POKE's.
23639	DATADD	2 bytes. Endereço do separador final do último elemento de dados. Não executar POKE's.	23660	STOP	2 bytes. Número da linha superior de um programa em listagem automática.
23641	E LINE	2 bytes. Endereço da ordem que está a ser escrita. Não executar POKE's.	23662	OLDPPC	2 bytes. Número da linha para onde salta a ordem CONT.
23643	K CUR	2 bytes. Endereço do cursor.	23664	OSPCC	1 byte. Número da declaração de uma linha para onde salta a ordem CONT.
23645	CH ADD	2 bytes. Endereço do carácter a interpretar em seguida. Não executar POKE's.	23665 23666	FLAGX STRLEN	1 byte. Flags. 2 bytes. Comprimento da cadeia em atribuição.
23647	X PTR	2 bytes. Contém endereço do carácter que se segue ao cursor?.	23668	T ADDR	2 bytes. Endereço do elemento seguinte da tabela de sintaxe.
23649	WORKSP	2 bytes. Contém endereço do espaço de trabalho temporário. Não executar POKE's.	23670	SEED	2 bytes. Função RND. É definida por RANDOMISE.
23651	STKBOT	2 bytes. Contém endereço do limite inferior do «stack» do microprocessador. Não executar POKE's.	23672	FRAMES	3 bytes. Contador de imagens incrementado a intervalos de 20 ms.
23653	STKEND	2 bytes. Endereço do início da área de memória livre. Não executar POKE's.	23675 23677	UDG COORDS	2 bytes. Endereço do primeiro gráfico definido pelo utilizador. 1 byte. Coordenada x do último ponto marcado no visor.
23655	BREG	1 byte. Registo B do microprocessador	23678	COORDS	1 byte. Coordenada y do último ponto marcado no visor.
23656	MEM	2 bytes. Primeiro endereço da área usada para memória do microprocessador.	23679	P POSN	1 byte. Usado para registar o número da coluna da posição de impressão,

23680	PR CC	inicializado para 33. 1 byte. Byte menos significativo do endereço da posição seguinte se LPRINT.	23694	MASK P	1 byte. Usada para cores transparentes.
			23695	ATTR T	1 byte. Usada para cores actuais temporárias.
			23696	MASK T	1 byte. Usada para cores transparentes actuais.
23681	Não usado		23697	P FLAG	1 byte. Flags.
23682	ECHOE	2 bytes. Usada para registar os números de coluna e linha do final do buffer de entradas, inicializados respectivamente para 33 e 24.	23698	MEMBOT	30 bytes. Usada para números que não podem ser incluídos no "stack" do microprocessador.
			23728	Não usada	
23684	DF CC	2 bytes. Contém endereço no ficheiro de imagem da posição de PRINT.	23730	RAMTOP	2 bytes. Endereço do último byte de memória da área Basic.
23686	DF CCL	2 bytes. Como DF CC, mas para a parte inferior do visor.	23732	P-RAMT	2 bytes. Endereço do último byte de memória RAM.
23688	S POSN	1 byte. Usado para registar o número de coluna da posição PRINT, inicializado para 33. Não executar POKE's.			
23689	S POSN	1 byte. Usado para registar o número de linha da posição PRINT, inicializada para 24. Não executar POKE's.			
23690	SPOSNL	2 bytes. Como S POSN mas para a parte inferior do visor. Não executar POKE's.			
23692	SCRCT	1 byte. Um mais do que o número de rolamentos ("scrolls") da imagem antes de a mensagem "scroll?" ser apresentada.			
23693	ATTR P	1 byte. Usado para cores actuais permanentes.			

ÍNDICE

PREFACIO	7
CAPITULO 1 — COMECEMOS	9
Introdução	9
O Teclado	10
Listagem e montagem do programa	15
Gravação de programas em cassette	17
Mensagens	20
O ligador externo	22
A impressora ZX Spectrum	22
CAPITULO II — NÚMEROS BINÁRIOS E HEXADECIMAIS	25
Introdução	25
Os números binários	26
Equivalente binário de um número decimal	28
Equivalente decimal de um número binário	30
Números hexadecimais	32
Equivalente hexadecimal de um número binário	34
Equivalente hexadecimal de um número decimal	35
Equivalente decimal de um número hexadecimal	36
Equivalente binário de um número hexadecimal	38
Números armazenados na memória	39
Aritmética binária	41
CAPITULO III — TRATAMENTO DE NÚMEROS	51
Representação de números em vírgula flutuante	51
Variáveis numéricas	54
Simples cálculos aritméticos	58
Funções matemáticas	60
Funções trigonométricas	63
Funções matemáticas definidas pelo utilizador	65
Quadros de variável numérica	68
Números aleatórios	71

CAPITULO IV — CADEIAS	75
Introdução	75
Variáveis de cadeia	76
Subcadeias	77
Funções de cadeia	79
Funções definidas pelo utilizador	81
Quadros de cadeias alfanuméricas	82
CAPITULO V — DECISÕES LÓGICAS	87
Introdução	87
Operações lógicas AND, OR e NOT	87
Operações lógicas NAND e NOR	89
Instrução condicional IF	93
CAPITULO VI — FLUXOGRAMAS, CICLOS E SUBRO- TINAS	99
Introdução	99
Fluxogramas	99
Ciclos	102
CAPITULO VII — GRÁFICOS A CORES	109
Introdução	109
O conjunto de caracteres	112
Caracteres guardados em ROM (Read Only Me- mory)	116
Caracteres definidos pelo utilizador	118
A imagem no visor	122
Desenho de linhas e círculos	126
Atributos da rede de caracteres	128
Outras observações sobre o visor	130
Movimentação de gráficos	132
CAPITULO VIII — SONS	137
Introdução	137
Produção de sons	137
Rudimentos de música	140
Exemplo ilustrativo	145
Amplificador de som	146

CAPITULO IX — HARDWARE	149
Introdução	149
Flip-Flops, Flags, Registos e Contadores	151
ROM (Read Only Memory)	155
RAM dinâmica	158
O mapa de memória	161
Os sinais de entrada e saída do microprocessador Z80A	163
Interface «Memory-Mapped» de um byte	166

CAPITULO X — PROGRAMAÇÃO EM CÓDIGO-MA- QUINA	175
Introdução	175
Execução de instruções pelo microprocessador Z80A	176
Conjunto de Instruções do Z80A	182
Modos de endereçamento	203
Endereçamentos de Registos	203
Endereçamento de registos indirecto	203
Endereçamento imediato	204
Endereçamento imediato extenso	204
Endereçamento extenso	205
Endereçamento modificado de página zero	206
Endereçamento implícito	207
Endereçamento de bits	207
Endereçamento indexado	208
Endereçamento relativo	209
Armazenamento e execução de programas em código- máquina	210
Exemplos Ilustrativos	216
Listagem do programa	220
Programa de entrada/saída	225
Rotina de produção de som: CALL 03B5	228

APÊNDICE A — PROGRAMAS	230
Programa 1: Conversão de Números	230
Programa 2: Cronómetro	232
Programa 3: Apresentação dos caracteres em ROM	234
Programa 4: Conteúdo da ROM	236
Programa 5: Descubra o Tesouro	237
Programa 6: Programa para carregar código-má- quina acima da RAMTOP	239

APÊNDICE B — GLOSSÁRIO DE TERMOS	241
---	-----

APÊNDICE C — VARIÁVEIS DE SISTEMA	249
--	-----

