Este livro contém toda a informação de que você necessita para usar em pleno o *ZX Microdrive.* Com explicações claras, muitos exemplos, tanto é adequado para o leitor relativamente recém-chegado ao BASIC como para o experimentado programador em código máquina.

São explicadas as *streams* e os canais fundamentais. Há uma secção da maior importância dedicada à manipulação de ficheiros que explica muitas características que normalmente só estão disponíveis nas máquinas baseadas em discos, incluindo a criação de ficheiros de programas do BASIC.

O livro contém também um programa de base de dados da maior importância para usar com o *microdrive* e um capítulo onde se explica como proteger os programas. Tal como acontece com o *microdrive,* são explicadas duas outras características do ZX Interface 1 — a entrada *(port)* do RS232 e a formação da rede *(networking).*

São fornecidos abundantes pormenores de como usar o RS232 como uma rede para o código máquina, incluindo explicações sobre as mais importantes rotinas ROM. Inclui-se um programa que torna fácil utilizar ordens para o *microdrive,* com uma ampla explicação sobre como você pode acrescentar elementos por sua própria iniciativa.

Domine o seu ZX Microdrive é um guia fundamental para os utilizadores.

Andrew Pennell é um estudante universitário, programador em regime livre e um colaborador regular do *Popular Computing Weekly.*

ARTE DE VIVER 98

3309885

ARTE DE VIVER 98

ANDREW PENNELL

DOMINE O SEU ZX MICRODRIVE

Programas, código máquina e formação da rede



PUBLICAÇÕES EUROPA-AMÉRICA

DOMINE O SEU ZX MICRODRIVE ANDREW PENNELL

3309885

DOMINE O SEU ZX MICRODRIVE
Programas, código-máquina e formação da rede

Obras publicadas nesta colecção:

1 — *111 Receitas com Ovos*, Etelvina Lopes de Almeida
2 — *O Livro do Casal*, Pierre-Marie Brémond
3 — *Aprenda a Fotografar*, Antoine Desilets
4 — *Guia da Interpretação dos Sonhos*, Louis Stanké
5 — *A Arte de bem Receber*, Marguerite du Coffre
6 — *Guia do Comportamento Sexual*, Dubois-Caballero
7 — *Como Reparar Avarias na Estrada — Manual de Todo o Automobilista*, Miguel de Castro Vicente
8 — *Guia Prático e Completo da Costura*, Lise Chartier
9 — *Guia Íntimo das Relações Sexuais*, Pierre Valinieff
10 — *Guia dos Jovens — A Vida e o Amor*, Dr. Benjamin Spock
11 — *111 Receitas de Tapas e Entradas*, Etelvina Lopes de Almeida
12 — *Guia da Futura Mãe durante a Gravidez*, Dr. José M.ª Carrera
13 — *Como Suprimir as Suas Dores com a Simples Pressão de Um Dedo*, Dr. Roger Dalet
14 — *O Livro das Boas Maneiras*, Marcelle Fortin-Jacques
15 — *111 Receitas de Frango*, Jacky Davin
16 — *Doenças Transmitidas pelas Relações Sexuais*, Dr. Lionel Gendron
17 — *Hatha-Yoga*, Suzanne Piuze
18 — *Os Segredos do Amor Táctil*, A. Vignati e O. Caballero
19 — *Como Socorrer o Seu Filho*, Marie Hermand
20 — *Métodos Anticonceptivos e Planeamento Familiar*, Santiago Dexeus e Margarita Riviere
21 — *A Técnica da Fotografia*, Antoine Desilets
22 — *Amor, Sexo e Astrologia*, Teri King
23 — *Como Vencer a Timidez*, François Suzzarini
24 — *111 Receitas de Coelho*, Anne Vernon
25 — *Os Remédios da Avozinha*, Barbara Kamir
26 — *A Mulher depois dos 40 Anos*, Santiago Dexeus e Teresa Pàmies
27 — *Viver bem depois dos 50 Anos*, Dr. Hugues Destrem
28 — *Conservas, Compotas e Xaropes*, Maria Emília Abreu Semedo
29 — *Como Proteger a Saúde e a Beleza com a Simples Pressão de Um Dedo*, Dr. Roger Dalet
30 — *111 Receitas para Emagrecer*, Dr. Jean-Paul Ostigny
31 — *Eu... Tu... e os Outros*, Anna Boyer e Isabelle Nicolas
32 — *O Rosto, Espelho do Carácter*, Louis Stanké
33 — *O Seu Aquário de Peixes Tropicais*, Brian Ward
34 — *Pílula — A Solução Mortal*, Dr. Dominique Chatain
35 — *O Seu Futuro nas Cartas*, Louis Stanké
36 — *111 Refeições Naturistas*, Maria Cândida de Albuquerque Cardoso
37 — *A Bíblia do Bridge*, Claude Derwy
38 — *A Congelação dos Alimentos*, Pamela Dotter
39 — *A Celulite*, Gerald J. Leonard
40 — *Guia Sexual da Moça Moderna*, Wardel B. Pomeroy
41 — *101 Conselhos aos Diabéticos*, Prof. Georges Tchobroutsky
42 — *A Beleza pela Saúde*, Dr. Pierre Fournier
43 — *Plantas de Interior*, Brian Ward e Tom Wellsted
44 — *111 Receitas de Cozinha Africana*, Maria de Lourdes Chantre
45 — *Saber Maquilhar-Se*, Josette Ghedin
46 — *111 Receitas de Massas*, Anne Vernon
47 — *Alimentação Natural*, José Lyon de Castro
48 — *A Mulher e o Sexo*, Dr. Lionel Gendron
49 — *A Menopausa*, Dr. Lionel Gendron
50 — *111 Receitas de Arroz*, Déda Frachon
51 — *Trate o Seu Cão, o Seu Gato, os Seus Pássaros com a Simples Pressão de Um Dedo*, Roger Dalet
52 — *A Chave da Longevidade*, Dr. Hugues Destrem
53 — *Tudo sobre Acupunctura*, Dr. Jean Vibes
54 — *101 Respostas sobre a Depressão*, Dr.ª Marie Claude Navikoff e Dr. Jean Pierre Olié
55 — *111 Receitas para Panela de Pressão*, Janet Warren
56 — *Como Vencer as Enxaquecas*, Dr. Claude Loisy e Dr. Sidney Pélage
57 — *Como Viajar de Avião sem Ter Medo*, Afra Botteri/Cécile Gateff
58 — *Tempo Que Mata, Tempo Que Cura*, Dr. Fernand Attali
59 — *Como Manter a Virilidade*, Paul Stanley
60 — *A Alimentação da Criança*, Louise Lambert-Lagacé
61 — *O Sexo e o Amor no Casamento*, Bernard Delon e Germaine Lanoë
62 — *Conheça-Se a Si Próprio — I*
63 — *Conheça-Se a Si Próprio — II*
64 — *Ténis Prático — Técnica — Conselhos — Campos*, Christian Collin
65 — *101 Segredos da Medicina Natural*, Dr. Péron-Autret
66 — *Manual de Protecção contra o Crime*, Ira A. Lipman
67 — *111 Receitas de Caça*, Ana Isabel de Castro
68 — *Manual Médico da Família*, Dr. David Kellett Carding
69 — *A Cozinha Astrológica*, Marie Goberg e Monique Maine
70 — *Doenças de Cães e Gatos Transmissíveis a Crianças*, Silva Leitão
71 — *Manual de Sobrevivência na Situação de Guerra Nuclear — Como Viver durante e após um Ataque Nuclear*, Barry Popkess
72 — *Conheça os Computadores, John Shelley*
73 — *Como Tratar o Seu Filho com a Simples Pressão de Um Dedo*, Dr. Tan Poch, Choon
74 — *Tudo sobre Astrologia*, H.-M. de Campigny
75 — *Horóscopos Árabes*, Paula Delsol
76 — *Horóscopos Chineses*, Paula Delsol
77 — *Aventuras com o Spectrum*, Tony Bridge e Roy Carnell
78 — *Enciclopédia dos Pontos Que Curam*, Dr. Roger Dalet
79 — *A Dianética*, L. Ron Hubbard
80 — *Auto-Análise*, L. Ron Hubbard
81 — *Como Vencer no Trabalho e na Vida*, L. Ron Hubbard
82 — *Como Planear e Construir a Sua Lareira*, Margaret e Wilbur F. Eastman Jr.
83 — *As Previsões Astrológicas para 1985*, Catherine Aubier
84 — *Faça Você Mesmo — I, Alvenaria — Telhados — Carpintaria*
85 — *Faça Você Mesmo — II, Electricidade — Canalização — Pintura — Vidraria*
86 — *Faça Você Mesmo — III, Revestimentos — Isolamentos — Refrigeração*
87 — *8 Exercícios para Um Corpo Perfeito*, Sheri Blair
88 — *Guia Prático da Sorte*, Cécile Donner e Jean-Luc Caradeau
89 — *Programação Prática para o Spectrum em Linguagem Máquina*, Steve Webb
90 — *Aplicações Domésticas no seu Microcomputador*, Mike Grace
91 — *Como Fazer Amor com a Simples Pressão de Um Dedo... e não só*, Hsuan Tsai Su-Nu
92 — *111 Receitas de Cozinha Indiana*
93 — *Como Fazer Amor com Um Homem*, Régine Dumay
94 — *Terapêutica Biológica*, Adriano de Oliveira
95 — *Inteligência Artificial no Spectrum*, Keith e Steven Brain
96 — *111 Receitas de Cozinha Chinesa*
97 — *O Spectrum Funcional*, David Lawrence
98 — *Domine o Seu ZX Microdrive*, Andrew Pennell

ANDREW PENNELL

DOMINE O SEU ZX MICRODRIVE

Programas, código máquina e formação da rede

ARTE DE VIVER®

PUBLICAÇÕES EUROPA-AMÉRICA

Título original: Master your ZX Microdrive

Tradução de Eng.º Mesquita de Abreu

Capa: estúdios P. E. A.

© *Andrew Pennell*
First published in English 1983 by:
Sunshine Books (an imprint of Scot Books Ltd)
12/13 Little Newport Street
LONDON WC2H 7PP

Direitos reservados por
Publicações Europa-América, Lda.

Nenhuma parte desta publicação pode ser reproduzida ou transmitida por qualquer forma ou por qualquer processo, electrónico, mecânico ou fotográfico, incluindo fotocópia, xerocópia ou gravação, sem autorização prévia e escrita do editor. Exceptua-se naturalmente a transcrição de pequenos textos ou passagens para apresentação ou crítica do livro. Esta excepção não deve de modo nenhum ser interpretada como sendo extensiva à transcrição de textos em recolhas antológicas ou similares donde resulte prejuízo para o interesse pela obra. Os transgressores são passíveis de procedimento judicial

Editor: Francisco Lyon de Castro

PUBLICAÇÕES EUROPA-AMÉRICA, LDA.
Apartado 8
2726 MEM MARTINS CODEX
PORTUGAL

Edição n.º 33 098/4005

Execução técnica:
Gráfica Europam, Lda.,
Mira-Sintra — Mem Martins

ÍNDICE

Pág.

Nota do editor português 11
Introdução 13

1 Streams e canais 15
2 *O arranque com os microdrives* 27
3 *Manipulação do ficheiro do microdrive* 41
4 *Unificha* 71
5 *Protecção dos programas* 89
6 *Utilização da interface RS232* 97
7 *Utilização da rede* 111
8 *O código-máquina e a interface* 129
9 *Acrescentando comandos BASIC* 157

Apêndice A — *As variáveis do sistema de interface* 171
Apêndice B — *Listagens Assembly* 175
Apêndice C — *«Bugs» da interface* 193

Pormenorização dos assuntos

CAPÍTULO 1
Streams e canais
Explicações sobre streams e canais, o uso de OPEN#, CLOSE#, CLEAR# e MOVE. Programa de «listagem de stream» e a rotina «Stream 14-Z$».

CAPÍTULO 2
O arranque com os microdrives
Como funcionam os microdrives, montagem de um cartucho *(cartridge)*, armazenamento de programas, armazenamento de variáveis, cópias múltiplas, programas em cadeia.

CAPÍTULO 3
Manipulação do ficheiro do microdrive
Ficheiros em série, ficheiros de escrita e de leitura, o uso de MOVE, acrescentando ficheiros, o uso de LIST#, a detecção do fim do ficheiro *(End of File)* (incluindo a rotina ON EOF GOTO), a utilização de programas como ficheiros de dados usando a rotina «OPEN# qualquer coisa», a criação de ficheiros não-dados, a rotina «Status», comandos de cores, e a rotina «Alta pontuação de jogos (Recordes)».

CAPÍTULO 4
Unificha
Um muito poderoso programa de base de dados para se usar com os microdrives e (opcionalmente) com uma impressora RS232.

CAPÍTULO 5
Protecção dos programas
A execução automática, a rotina ON ERROR GOTO, os POKE que ocasionam falhas.

CAPÍTULO 6

A utilização da interface RS232

Tipos de periféricos e velocidade. Canais de texto, canais binários, accionamento duma impressora incluindo a implementação TAB, cópias do écran, e outras utilizações.

CAPÍTULO 7

Utilização da rede

O envio de programa através da rede, o envio de dados, a emissão, programas escravos para impressora e microdrive.

CAPÍTULO 8

O código-máquina e a interface

A ROM sombra, o mecanismo de paginação, como funcionam as streams e os canais, variáveis dos canais, rotinas ROM: microdrive, RS232, rede e diversas. Sumário do *Hook code*.

CAPÍTULO 9

Acrescentando comandos BASIC

Como alterar a sintaxe, um programa para mudar os comandos do microdrive, e como acrescentar comandos a partir do código-máquina.

Apêndice A

As variáveis da interface, as suas posições, mnemónicas e os seus usos.

Apêndice B

Listagens Z80 da stream 14-Z$, ON EOF GOTO, OPEN# qualquer coisa, Status, ON ERROR GOTO e RS232 TAB, com anotações.

Apêndice C

Bugs da interface — Uma lista incluindo os mais importantes defeitos que afectam o uso da interface em BASIC e código-máquina.

Nota do editor português

A informática, como todas as disciplinas do conhecimento humano, tem a sua linguagem própria — e essa linguagem é quase na totalidade baseada no inglês. E se certas palavras (*byte*, por exemplo) não têm equivalente na nossa língua, outras (como, por exemplo, *net* ou *network*) têm traduções que só parcialmente correspondem ao conteúdo da palavra original.

Ao editar livros dirigidos a um público específico houve a preocupação de traduzir tanto quanto possível o que tem tradução, seguindo o critério de pôr entre parênteses a palavra original e usando o tipo de letra corrente para aquelas que não tenham tradução ou cuja tradução registada seja menos corrente, entre os utilizadores, que a palavra original.

Tomemos como exemplo a palavra inglesa *interface:* poder-se-iam utilizar termos como *tradutor, comunicador de via dupla, interpretador, adaptador*... Mas a realidade do objecto em si é, para os interessados na informática (mormente o público mais restrito a quem este livro se dirige), muito melhor definido pela palavra *interface*.

Que os puristas da língua tenham isto em consideração e não esqueçam que a aquisição de termos por apropriação literal é um dos meios de ampliação de uma língua viva.

Introdução

O meu desígnio ao escrever este livro é revelar as melhores maneiras de utilizar o Sinclair ZX Interface 1, com particular referência para os ZX Microdrives. A interface aumenta grandemente a utilidade do Spectrum e o microdrive oferece um acesso mais rápido ao armazenamento em massa por um custo baixo.

As características padrão são todas claramente explicadas, mas são suplementadas por uma quantidade de características «escolhidas» que tornam a interface muito mais útil. Incluem-se várias rotinas de código-máquina que aumentam grandemente o seu potencial, e elas foram apresentadas de modo que qualquer pessoa pode facilmente introduzi-las e usá-las sem qualquer conhecimento de código-máquina. Para aqueles leitores que conhecem o código-máquina incluí uma secção sobre rotinas interface ROM e como as usar, assim como listagens das minhas próprias rotinas.

Os programas em código-máquina são apresentados sob a forma de séries de números em instruções DATA, e um ciclo FOR para as colocar (POKE) no lugar. Sempre que possível, o código-máquina tem sido relocável. Isto significa que o leitor pode colocá-lo em qualquer posição da memória e ainda assim funcionar correctamente. O melhor local para o código-máquina é acima do RAMTOP (veja-se a p. 179 do manual do Spectrum), o qual pode ser deslocado para baixo com a instrução CLEAR. Outros locais para armazenar o código-máquina são as áreas de gráficos definidos pelo utilizador (168 bytes) e a unidade-tampão *(buffer)* da impressora (256 bytes). Usando a interface devem ser evitadas as instruções REM, por razões técnicas. Para facilidade de uso, cada rotina tem duas posições recomendadas (para as máquinas de 16K e de 48K), que permitirão que todas as rotinas estejam presentes si-

multaneamente. Se elas são usadas, o RAMTOP deve primeiro ser descido suficientemente — para todas as rotinas, CLEAR 32009 para 16K ou CLEAR 64779 para 48K reservarão espaço suficiente. Como algumas das rotinas mais compridas contêm muitos números, que são susceptíveis de erro, cada rotina forma uma «*checksum*» (verificação de soma) à medida que é colocada (POKE). Se o total não está correcto, a rotina parará (STOP), com uma mensagem apropriada. Os dados devem então ser reverificados, e corrigidos onde for necessário.

Há também secções sobre a segurança dos programas e sobre o acrescentar de comandos adicionais do BASIC. Os Apêndices contêm informações técnicas, assim como documentam os conhecidos problemas (ou «*bugs*») quando se usa a interface.

A própria produção deste livro começa a mostrar o valor potencial do Spectrum como uma máquina «séria» — eu usei um Spectrum de 48K com um teclado verdadeiro e interface de impressora, juntamente com um processador de texto e uma impressora por pontos.

ANDREW PENNELL
Cliftonville, Kent
Setembro de 1983

CAPÍTULO 1

Streams e canais

Antes de discutir as muitas características extras que a interface ZX dá ao Spectrum, devem ser compreendidos os conceitos de streams e canais, pois que eles são fundamentais para o uso eficiente de microdrives, RS232 e rede.

Um *canal* é uma parte dum sistema computador para o qual os dado podem ser enviados e do qual podem ser recebidos, tal como o teclado para a entrada *(input)* de dados, e o écran para a saída *(output)* de dados. Os percursos ao longo dos quais os dados passam para os canais são chamados *streams*.

No Spectrum existem dezasseis streams disponíveis do BASIC, quatro das quais estão inicialmente atribuídas a certos canais, como se mostra no quadro abaixo:

STREAM N.º	ESPECIFICADOR DO CANAL	USO EM ENTRADA	USO EM SAÍDA
# 0	«K»	Parte inferior do écran.	Teclado
# 1	«K»	Parte inferior do écran.	Teclado
# 2	«S»	Parte superior do écran.	Nenhum
# 3	«P»	Impressora ZX.	Nenhum

As streams 0 e 1 são idênticas e estão ligadas ao canal «K». Este faz sair caracteres para a parte inferior do écran (onde ocorrem mensagens de erro de INPUT) e faz a entrada a partir do teclado.

A stream 2 é o canal «S», que imprime caracteres na parte superior do écran, e a stream 3 é o canal «P», o qual envia caracteres para a impressora ZX. O sistema do Spectrum não permite a entrada pelas streams 2 e 3. Se, por exemplo, se tentar a instrução INPUT #3;a$ produzir-se-á a mensagem de erro *Invalid I/O device* (Dispositivos de entrada/saída não válidos).

O símbolo para stream é o sinal cardinal (#, tecla *symbol shift* e tecla 3), que pode ser acrescentado a PRINT, a INPUT e a alguns outros comandos. Por exemplo, para se imprimir nas linhas da parte inferior do écran, que não estão normalmente disponíveis, você pode usar as streams 0 ou 1 — isto é:

```
PRINT #0;AT 0,0;"Linha 23";AT 1,
0;"Linha 24";:PAUSE 0
```

A pausa é necessária para evitar que a mensagem «OK» destrua o que se mandou escrever. Quando para imprimir algo se usa o canal «K», a parte inferior do écran pode inesperadamente rolar para a parte superior. Isto pode ser evitado pelo uso dos comandos AT e fazendo terminar qualquer texto com um ponto e vírgula

A instrução PRINT #2; é a mesma que a usual instrução PRINT, que envia mensagens para o canal «S», isto é, a parte superior do écran. PRINT #3; é o mesmo que LPRINT, que imprime caracteres na impressora ZX (canal «P»). Uma vantagem da utilização das streams 2 e 3 nos programas é que pode facilmente ser usada uma variável para determinar se o que se faz entrar vai para o écran ou para a impressora. O programa seguinte demonstra o uso desta técnica:

```
1000 INPUT "Impressora (S/N)";a$
1010 LET c=2: IF a$="s" OR a$="S
" THEN LET c=3
1020 PRINT #c;"Isto e um ZX Spec
trum"
```

É também possível misturar streams na mesma instrução PRINT, isto é:

```
2000 PRINT #2;"O ecran";#3;"A im
pressora ZX"
```


As instruções INPUT e LIST podem também ser usadas com streams. Com um Spectrum comum quer dizer, sem estar ligado a uma interface) somente são permitidas as instruções INPUT # 0; e INPUT # 1; e são equivalentes à normal instrução INPUT. Contudo, como iremos ver em capítulos posteriores, a instrução INPUT # é extremamente útil quando for usada a interface.

A instrução LIST # n, onde «n» é o número da stream, pode também ser usada, e dirige uma listagem dum programa BASIC para a stream seleccionada. LIST # 3 é equivalente ao comando LLIST, que produz uma listagem na impressora ZX, e LIST # 0 produz um interessante (se bem que de modo algum útil) efeito no écran.

A instrução INKEY # n pode também ser usada com streams, mas não tem utilidade com os canais padrão. INKEY # 0 e INKEY # 1 normalmente produzem como resultado cadeias *(strings)* nulas, e # 2 e # 3 não são permitidos. Em capítulos posteriores vai-se mostrar como isto pode ser usado com os canais de interface extras.

OPEN # CLOSE

A instrução OPEN #, com um Spectrum padrão, é bastante limitada. A sua finalidade é ligar um canal a uma stream, e tem a forma geral

```
OPEN #n,f$
```

onde «n» é o número da stream, e «f$» o especificador do canal. Deve notar-se que o sinal # faz parte da palavra-chave *(keyword)* OPEN #, obtida premindo o 4 no modo extensivo. O valor de «n» deve estar compreendido entre 0 e 15 (de outro modo ocorrerá um erro *Invalid stream*) e o especificador do canal deve ser uma simples letra e especificar um canal válido (quer em maiúsculas, quer em minúsculas), isto, é, «K», «k», «S», «s», «P», «p». Se não for válido ocorrerá um erro *Invalid filename*. Com a interface ligada, podem também ser aceites especificadores de canal «M», «N», «B» e «T» e as suas versões em letras minúsculas, que representam respectivamente os canais de microdrive, rede Binary RS232 e Text RS232. Estes nomes adicionais de canais permitem também o uso

do ponto e vírgula como separador na instrução OPEN #, assim como o uso duma vírgula, e alguns requerem também serem seguidos por dados adicionais. Nos capítulos adequados serão fornecidos mais pormenores.

Como exemplo:

```
OPEN #3,"s"
```

fará que toda a saída da stream 3, normalmente para a impressora ZX, vá para o canal «S», a parte superior do écran, assim como LLIST produzirá uma listagem no écran. Isto poderá ser útil para evitar erros *(bugs)* de programas que usam a impressora ZX, enquanto não está ligada, ou para poupar papel. O oposto disto é

```
OPEN #2,"p"
```

o que faz que toda a saída da stream 2, normalmente o écran, seja enviada para a impressora. Se ambos estes comandos forem usados simultaneamente, a saída do programa pode ser bastante confusa — e é o menos que se pode dizer!

Pode também usar-se OPEN # para utilizar streams adicionais. Normalmente as streams 4 a 15 são «sem canal», e qualquer tentativa para as usar (por exemplo, PRINT # 6; «6»;) produzirá a mensagem de erro *Invalid stream*. Contudo estas streams extras podem ser abertas da maneira normal — por exemplo:

```
OPEN #4,"s"
PRINT #4;"Ei, tu ai!"
```

produzirá as palavras «Ei, tu ai!» no écran, canal «S». As streams 4 a 15 são extensivamente usadas com periféricos Sinclair extras, usando a interface.

Para desfazer uma instrução OPEN #n, deve ser usado o seu complemento, o comando CLOSE #n, com «n» a referir-se ao número da stream. Fechar as streams 0 a 3 fá-las-á voltar aos seus canais usuais, ao passo que fechar as streams 4 a 15 restabelecê-las-á para nenhum canal.

IMPORTANTE: Deve acautelar-se quanto ao uso da instrução CLOSE # com as streams 4 a 15 sem estar a interface ligada — devido a uma falha no BASIC, podem ocorrer coisas estranhas se a stream já está fechada. Usualmente é gerada uma curiosa mensagem de erro, mas ocasionalmente o computador pode perder todo o programa e os dados (RESET)! O problema não existe com a interface ligada, pois que ela corrige o defeito.

CLEAR E MOVE

Há uma outra instrução que pode ser usada com canais — nomeadamente CLEAR # (note-se o sinal de cardinal). Isto não tem nada a ver com a instrução CLEAR normal — a sua finalidade é fechar (CLOSE) as streams de 4 a 15 e restabelecer as streams de 0 a 3 para os seus canais iniciais. Deve ter-se cuidado no seu uso com os tipos de canais extra da interface.

O comando final para uso com canais é MOVE. Pode ter diferentes formas de sintaxe, mas é basicamente:

```
MOVE #a TO #b
```

Onde «a» e «b» são números de streams (e TO é uma palavra-chave). O que ele faz é ler um carácter duma stream «a» e imprimi-lo numa stream «b». Repete este processo até que uma certa condição é satisfeita — a condição depende do tipo de canal a que está ligada a stream número «a». É um comando muito poderoso, e as suas diferentes aplicações são explicitadas em capítulos subsequentes.

É possível, para rotinas em código-máquina, criar streams especiais para usos particulares. Por exemplo, o sistema operacional BASIC usa internamente o canal «R» com a stream −1 (que não está disponível nos programas BASIC) para «imprimir» caracteres na área interna de trabalho da memória. É também possível modificar canais existentes — todas as impressoras do tipo Centronics produzidas independentemente modificam o canal «p», de modo que toda a saída da stream 3 (por exemplo, LPRINT) envia caracteres para uma impressora externa via o seu interface, em vez da impressora ZX.

Quando se está a usar várias streams ao mesmo tempo é muitas vezes difícil conseguir saber que canal está ligado com qual stream, e quais as streams que não estão sequer a ser usadas.

O programa seguinte produz um quadro que mostra todos os pormenores relevantes de cada stream, incluindo as streams — 3 a — 1, que só são disponíveis para os programas em código-

-máquina. Quando o programa é executado (RUN) imprime um sumário do uso de cada stream e pode ser usado para entradas, ou para saídas, ou para ambas. Quando o sumário estiver terminado, é-lhe pedido a si um número de stream. Quando o introduzir, é dado maior número de pormenores sobre essa particular stream. O programa, como está feito, somente dá informações sobre os canais normais. No decurso deste livro, nos capítulos adequados, serão acrescentadas características adicionais.

Listagem da stream

```
1000 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
1005 CLS: PRINT TAB 10; INVERSE
1;"USO DA STREAM"
1010 PRINT "Stream";TAB 7;"Entra
da/Saida";TAB 14;"Nome";TAB 19;"
Uso"
1020 FOR s=-3 TO 15
1025 PRINT TAB 2;s;
1030 LET d=FN p(s*2+23566+8)
1040 IF d=0 THEN PRINT TAB 19; I
NVERSE 1;"Nao usado": GO TO 1200
1050 LET d=d+FN p(23631)-1
1060 IF FN p(d+2)<>5572 THEN PRI
NT TAB 7;"Entrada";
1070 PRINT TAB 9;"/";
1080 IF FN p(d)<>5572 THEN PRINT
 TAB 10;"Saida";
1090 LET f$=CHR$ PEEK (d+4)
1100 PRINT TAB 14;"""";f$;"""";T
```



```
AB 19;
1110 IF f$="K" THEN PRINT "Parte
 inferior do ecran": GO TO 1200
1120 IF f$="S" THEN PRINT "Parte
 superior do ecran": GO TO 1200
1130 IF f$="P" THEN PRINT "Impre
ssora": GO TO 1200
1140 IF f$="M" THEN PRINT "Micro
drive": GO TO 1200
1150 IF f$="N" THEN PRINT "Rede"
: GO TO 1200
1170 IF f$="T" THEN PRINT "RS232
": GO TO 1200
1180 IF f$="R" THEN PRINT "Espac
o de trabalho": GO TO 1200
1190 PRINT FLASH 1;"Nao especifi
cado"
1200 NEXT s
1210 PRINT AT 0,0;
1500 INPUT "Introduzir numero st
ream ou ENTER para saida ?";LINE
 S$
1505 IF S$="" THEN STOP
1510 CLS
1515 LET S=VAL S$
1520 PRINT TAB 10; INVERSE 1;"Nu
mero stream ";S''
```

```
1530 LET D=FN P(S*2+23566+8)
1540 IF D=0 THEN PRINT "Stream f
echada": GO TO 1500
1550 LET D=D+FN P(23631)-1
1560 LET F$=CHR$ PEEK (D+4)
1570 PRINT "Especificador de can
al:";f$
1580 REM Verificar cada tipo
1600 IF F$="K" THEN GO TO 2000
1610 IF F$="S" THEN GO TO 2020
1620 IF F$="P" THEN GO TO 2040
1660 IF F$="R" THEN GO TO 2050
1670 PRINT FLASH 1;"Especificado
r desconhecido"
1700 GO TO 1500
1800 PRINT "Rotina de saida  :";
FN P(D)
1810 PRINT "Rotina de entrada:";
FN P(D+2)
1820 IF FN P(D)=5572 THEN PRINT
FLASH 1;"Somente entrada"
1830 IF FN P(D+2)=5572 THEN PRIN
T FLASH 1;"Somente saida"
1870 RETURN
1999 REM Canal K
2000 PRINT INVERSE 1;"Parte infe
rior do ecran/teclado"
```



```
2010 GO TO 2060
2019 REM Canal S
2020 PRINT INVERSE 1;"Parte supe
rior do ecran"
2030 GO TO 2060
2039 REM Canal P
2040 PRINT INVERSE 1;"Impressora
 ZX"
2045 GO TO 2060
2049 REM Canal R
2050 PRINT INVERSE 1;"Espaco de
trabalho"
2060 GO SUB 1800
2070 GO TO 1500
```

O programa funciona chamando (PEEK) várias posições de memória — leia esta explicação somente se ela lhe interessar.

O ciclo *for-next «s»* (linha 1020) atravessa cada stream uma após outra. A variável «d» torna-se o deslocamento na área STRMS (1030) para a stream seleccionada, o qual será zero se a stream está fechada (1040). O deslocamento é então adicionado à variável de sistema CHANS (1050) com «d» indicando para um número de bytes em cada área de memória CHANS. Os canais estabelecidos (isto é, K, S, P e R) ocupam cada um o mínimo de cinco bytes na área CHANS (dando um total de 20 bytes). Em cada secção de cinco bytes, os dois primeiros indicam a rotina «saída de um carácter», os dois segundos indicam a rotina «entrada de um carácter», e o quinto byte é o especificador de canal em maiúsculas. A posição 5572 é o salto para o erro *Invalid I/O device* (para ver, experimente RANDOMIZE USR 5572), e isto é verificado nas linhas 1060 e 1080. O especificador do canal é impresso na linha 1100, e depois verificado para os canais conhecidos nas linhas 1110-1180, e é impresso um adequado número de dispositivo. Depois do sumá-

rio, o utilizador pode fazer entrar um número de stream (1500), e são então dados pormenores adicionais. Para os canais correntes, isto é somente as posições de entrada e saída, usando a sub-rotina em 1800, mas isto será ampliado mais tarde. A figura 1 mostra uma saída típica do programa.

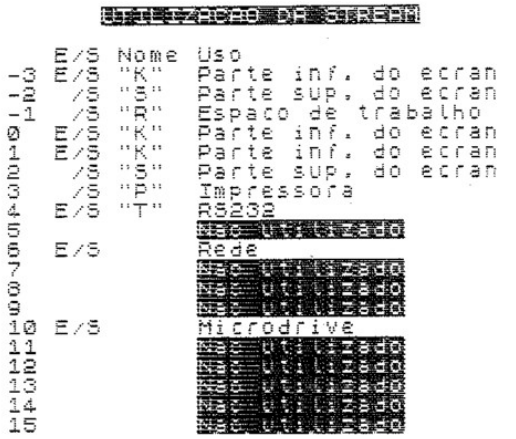


Fig. 1 — *Saída do programa «Listagem de stream»*

Programa «Stream 14-z$»

A seguinte rotina em código-máquina cria a stream 14 de tal modo que ela «imprime» caracteres numa cadeia variável BASIC. Quando a rotina é chamada com uma instrução GO SUB 8000, a variável «st» deve conter o lugar da memória para onde a rotina deve ir. O seu comprimento é de 101 bytes, e as posições recomendadas são 65260 (48K) e 32490 (16K). Depois de a rotina ter sido chamada, a saída da stream 14 será acrescentada ao fim da cadeia de variáveis z$ (a qual não deve ser dimensionada). Se z$ não existe, ou está dimensionada, ocorrerá um erro *Variable not found.* É

possível fazer PRINT #14; e LIST #14, e, talvez com mais utilidade, é usado no próximo capítulo para CATalogar um microdrive numa cadeia *(string)*. Há contudo uma restrição: instruções que imprimem cadeias directamente; por exemplo:

```
PRINT #14;"string"
```

Listagem

acrescentará incorrectamente «ssssss» ao fim de z$. Há duas maneiras de ultrapassar isto, quer imprimindo cada carácter individualmente (isto é, PRINT #14;«s»,«t»,«r»,«i»,«n»,«g», quer usando uma outra variável textual *(string)* (LET a$ = «string»: PRINT #14;a$). Para programadores avançados: a razão é que esta rotina muda secções da memória para cima para inserir os caracteres, e também muda para baixo o espaço de trabalho, onde a cadeia temporária está armazenada; deste modo, o primeiro carácter é sempre acrescentado.

Listagem de «Stream 14-z$»

```
7995 REM**************************
7996 REM* Rotina Stream 14-z$  *
7997 REM**************************
7998 REM st=posicao de inicio
7999 REM recomendado:65260 / 324
90
8000 RESTORE 8070
8005 LET c=0
8010 FOR i=st TO st+100
8020 READ a: POKE i,a: LET c=c+a
8030 NEXT i
8035 IF c<>10557 THEN PRINT "Err
o de verificacao da soma": STOP
```

```
8040 CLOSE #14
8050 RANDOMIZE USR st
8060 RETURN
8070 DATA 42,83,92,43,197,229,1,
11
8075 DATA 0,205,85,22,209,33,59,
0
8080 DATA 193,9,213,235,115,35,1
14,35
8085 DATA 235,1,247,255,9,1,9,0
8090 DATA 237,176,225,35,237,75,
79,92
8095 DATA 167,237,66,34,50,92,1,
0
8100 DATA 0,201,196,21,90,40,0,4
0
8105 DATA 0,11,0,245,42,75,92,12
6
8110 DATA 254,90,40,11,254,128,2
02,112
8115 DATA 6,205,184,25,235,24,24
0,35
8120 DATA 78,35,70,3,197,229,9,2
05
8125 DATA 82,22,35,235,225,193,1
12,43
8130 DATA 113,241,18,167,201
```


CAPÍTULO 2

O arranque com os microdrives

Como funcionam

Um cartucho de microdrive contém um anel contínuo de cerca de 16 pés de fita, duma qualidade melhor do que a que é usada nas audiocassetes. Os microdrives propriamente ditos consistem de um motor, para fazer a fita rodar, e uma cabeça de gravação/leitura como a de um leitor de cassetes. A fita é passada pela cabeça a alta velocidade, e pode ter dados armazenados por gravação, ou dela lidos. É realmente um muito rápido sistema de cassete, sem a complicação de ter de rebobinar as fitas, etc.

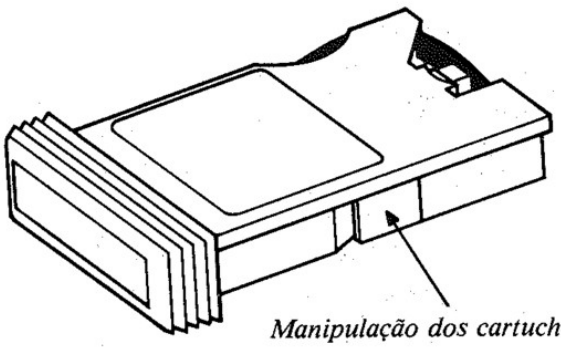
O sistema de banda contínua tem contudo desvantagens — a principal é o tempo de acesso. Se a secção de fita com o vosso programa acabou justamente de passar a cabeça, e você o quer carregar, toda a fita tem de passar pela cabeça de leitura antes de o voltar a encontrar, visto que não é possível rebobinar a fita. Isto significa que pode demorar até sete segundos para encontrar um programa ou alguns dados na fita.

Também pode ser bastante vagaroso quando se pretende escrever dados num cartucho. Se o cartucho está quase novo, com poucos outros programas, o computador cedo será capaz de encontrar uma secção disponível onde o armazenar. Contudo, se se trata dum cartucho bastante usado, o computador terá de procurar por toda a fita e armazenar pequenas partes do programa em diferentes partes da fita, o que pode levar um certo tempo. Isto também fará com que seja mais vagarosa a próxima leitura.

Protecção do que está escrito

As audiocassetes têm na parte de trás duas pequenas patilhas que podem ser puxadas para fora para evitar que se grave qualquer coisa onde já alguma outra esteja gravada. São chamadas patilhas de protecção da escrita e são também usadas nos cartuchos do microdrive Sinclair, com um fim semelhante.

Como se pode ver na figura 2, cada cartucho tem uma pequena patilha de plástico num dos lados. Se a levantar com uma chave de parafusos ou um objecto semelhante, impedi-lo-á, ou a qualquer outra pessoa, de escrever dados no cartucho. Isto é muito importante para programas comerciais e pode ser muito útil se não quiser que os seus próprios programas sejam apagados. Se, posteriormente a ter removido a patilha, quiser voltar a utilizar o cartucho, pode pôr uma fita gomada sobre o local do corte. A fita não deve ser tão larga que se sobreponha às superfícies do fundo e do topo do cartucho, de outro modo não se adaptaria correctamente no microdrive.



Manipulação dos cartuchos

Fig. 2 — *Patilha de protecção da escrita*

A fita que o cartucho contém é muito delicada. Nunca a deve tocar com os dedos ou qualquer outra coisa, e quando não estiver a ser utilizada deve colocá-la sempre na caixa para a proteger.

Quando inserir cartuchos no microdrive, assegure-se de que eles estão em posição correcta (com a grande etiqueta voltada para cima) e empurre com firmeza. Uma vez que o cartucho esteja inserido, assegure-se de que o empurrou tanto quanto ele poderá entrar, de outro modo podem ocorrer erros quando estiver a ser usado. Há duas regras muito importantes a que se se deve obedecer ao usar os cartuchos:

1. Nunca retirar um cartucho enquanto o LED encarnado está aceso.
2. Nunca ter um cartucho no microdrive quando se vai ligar à corrente, ou desligar.

O não comprimento destas regras pode resultar em dano permanente do cartucho, ou do próprio microdrive.

Os cartuchos não duram muito tempo e em tão boas condições como as fitas das cassetes, o que se deve às suas severas condições de trabalho. Deve sempre conservar-se num outro cartucho, ou melhor ainda numa cassete, cópias de programas importantes ou dados.

Montagem dum cartucho usando FORMAT

Quando se tem um cartucho novo, em branco, a fita dentro dele conterá informação magnética aleatória. Qualquer tentativa para o usar produzirá um erro *Microdrive not present*. Para permitir que ele seja usado tem de lhe ser dada uma configuração, usando o comando FORMAT. Para os microdrives isto tem a forma geral:

```
FORMAT "m";d;"nome"
```

onde «d» é o número do drive (de 1 a 8) e «nome» é o título que será permanentemente atribuído ao cartucho, com um comprimento até dez letras. Isto demorará cerca de 30 segundos, e durante este tempo o computador configurará o cartucho, armazenará o seu título e testará cada secção da fita várias vezes. Se ele encontrar uma secção que não pode ser usada por estar estragada por exemplo,

marcará este espaço para ser evitado em futuras operações. Durante este processo a margem do écran piscará. Antes de proceder à formatação, você deve assegurar-se de que o cartucho está devidamente colocado no microdrive. O primeiro cartucho em que procedi à formatação não estava empurrado até ao fundo, do que resultou que o computador pensou que só podia ser usado um terço dele! Quando, depois de o ter empurrado até ao fundo, repeti o comando, a formatação foi executada correctamente. Com cartuchos novinhos em folha, o melhor é fazer, em rápida sucessão, dois comandos FORMAT.

O comando FORMAT vai destruir quaisquer programas que estejam previamente no cartucho, de modo que se deve sempre efectuar uma dupla verificação antes de se executar o comando. Verificar-se-á um erro se tentar formatar um cartucho com a patilha de protecção de escrita removida.

Após um FORMAT você pode verificar o processo fazendo entrar *CAT*d (onde «d» é número do drive), que, depois de passados uns sete segundos, deve imprimir o título do cartucho, seguido dum número. Este número é o número de kilobytes (K) que são utilizáveis no cartucho. Depois de um FORMAT, este deve ser pelo menos 85K. O máximo que já tive num cartucho foi 92K.

Programas e microdrives

Uma das principais vantagens dos microdrives é a sua grande velocidade de guardar e carregar as cassetes. Para os usar são necessários comandos semelhantes aos das cassetes, com algumas modificações. Por exemplo, para guardar o programa da listagem da stream que vimos no capítulo precedente, você podia usar:

```
SAVE "streams"
```

Para usar estes comandos com os microdrives têm de se inscrever várias coisas na linha a seguir ao comando. Primeiramente deve ser acrescentado um asterisco (*), depois «m»; para mostrar que é uma operação de microdrive; por último «d»; para mostrar qual o drive que a operação vai usar. Para guardar o programa de stream num cartucho no drive 2, você podia usar:

```
SAVE *"m";2;"stream"
```


(Você pode usar «m» ou «M» — não há distinção entre eles.) Os comandos adicionais não serão aceites por um verificador de sintaxe dum Spectrum sem a interface ligada.

Quando ocorre qualquer operação de microdrive, o motor rodará e o LED encarnado acender-se-á. Se um drive for posto em funcionamento sem um cartucho no seu lugar, emitirá um som agudo e será produzido o erro *Microdrive not present* (microdrive não presente). Você pode normalmente saber, pelo som que fazem, se a combinação drive/cartucho está a funcionar correctamente. Quando for uma operação normal sai do drive um zumbido suave. Se algo está errado, por exemplo, um cartucho não inserido de maneira correcta, o drive produz um som forte. Se isto suceder, você deve empurrar com firmeza o cartucho para dentro do drive. Se persistir este ruído, faça BREAK (premindo CAPS e SPACE) e examine o cartucho, substitua-o e experimente outra vez. Se o drive produz um som estridente quando o cartucho está no seu lugar, e ocorre uma mensagem de erro, isso significa provavelmente que o cartucho tem um defeito e a fita não pode mais rodar. Deve ser substituído. Se estiver tudo bem, durante um SAVE o drive funcionará pelo menos sete segundos, enquanto se efectua uma busca do nome do ficheiro no cartucho. Se encontra um, ocorrerá o erro *Writing to a 'read' file* (A escrever num ficheiro de leitura). A margem do écran pisca em qualquer altura em que os dados estejam a ser escritos num cartucho.

Tal como acontece com as cassetes, é possível usar a função LINE numa instrução SAVE, de modo que um programa em BASIC começa a funcionar automaticamente quando for carregado. Para fazer o programa acima começar a trabalhar a partir da linha 1000, você podia usar:

```
SAVE *"m";2;"stream" LINE 1000
```

Depois de um SAVE, pode verificar a correcta operação do equipamento usando a função de verificação, isto é:

```
VERIFY *"m";2;"stream"
```

Se existir qualquer diferença, produzir-se-á o erro *Verification failed* (Verificação falhada). Os programas do microdrive devem sempre verificar perfeitamente — se tal não acontecer, deve examinar de modo exaustivo quer o cartucho quer o microdrive. Se a falta persistir, deve substituir-se um deles, ou ambos.

Para armazenar um programa a partir do cartucho é usado o comando LOAD com a sintaxe:

```
LOAD *"m";d;"nome"
```

onde «d» é o número do drive e «nome» é o nome do programa que se quer. Ao contrário do que se passa com a cassete, não é possível armazenar o primeiro programa na fita usando como nome do ficheiro uma cadeia vazia. Se se tentar fazê-lo, por exemplo:

```
LOAD *"m";1;""
```

produzir-se-á o erro *Invalid filename* (Nome do ficheiro não válido). Se se não puder encontrar num cartucho o nome do ficheiro, verificar-se-á o erro *File not found* (Ficheiro não encontrado).

A fusão de funções também funciona com os microdrives, e tem uma sintaxe semelhante à dos comandos descritos, isto é:

```
MERGE *"m";d;"nome"
```

Isto armazenará o programa «nome» do drive «d» e fundi-lo-á com o programa e variáveis existentes, substituindo quaisquer que lá estejam pelos do cartucho. Ao contrário do que acontece com a instrução semelhante para cassete, não é possível fundir (MERGE) um programa de execução automática, guardado com LINE xxx. Uma tentativa para o fazer produzirá a mensagem *Merge error* (Erro de fusão).

Um comando adicional, que não é necessário para ficheiros de cassete, é ERASE (apagar). O seu fim é apagar qualquer ficheiro do cartucho e tem a sintaxe:

```
ERASE "m";d;"nome"
```

A instrução de apagar não requer um asterisco, ao contrário dos outros comandos do microdrive. Não é permitido fazer seguir o nome do ficheiro com qualquer outra coisa, por exemplo, LINE 2000.

Se se faz uma paragem (BREAK) durante uma operação SAVE, o ficheiro ficará incompleto e não se poderá armazenar. Tem de se apagar qualquer ficheiro que tenha sido parcialmente criado com um SAVE interrompido. Se se tentar armazenar um programa que já está no cartucho, ocorrerá o erro *Writing to a 'read' file* (A escrever num ficheiro de leitura). O antigo programa deve ser apagado antes que o novo seja armazenada.

Programas em cadeia

Usando-se a facilidade SAVE* «m» ...LINE, um programa pode arrancar automaticamente e carregar um outro programa, talvez algum código-máquina. O problema é que os programas em cadeia somente funcionarão num drive particular — para contornar isto, o primeiro programa deve ter uma linha tal como:

```
LET d=PEEK 23766
```

Então a variável «d» conterá o drive donde o programa foi carregado. Se o segundo programa for chamado «prog2», para o carregar a partir do primeiro podia usar:

```
100 LOAD *"m";d;"prog2"
```

Os dois programas serão então executados independentemente de quais os drives em que estão. A propósito, qualquer coisa como:

```
100 LOAD *"m";PEEK 23766;"prog2"
```

não funcionará, por razões técnicas. Use sempre uma variável.

Quadros e código-máquina no cartucho

Tal como se pode armazenar programas no cartucho, é também possível armazenar quadros *(arrays)* código-máquina, e écrans, acrescentando certas palavras-chave depois do nome do ficheiro nos comandos adequados. Para armazenar quadros faça seguir o nome do ficheiro com DATA, e a seguir o nome do quadro, numérico ou de cadeia. Por exemplo, para guardar o quadro «b ( )» no drive 2 sob o nome de «salários», use:

```
SAVE *"m";2;"salarios" DATA b()
```

Com as cassetes é possível guardar na fita cadeias de variáveis vulgares, não dimensionadas, mas quando voltarem a ser carregadas ficam corrompidas. Isto foi evitado no microdrive. Para o mostrar, escreva no teclado:

```
LET a$="teste": SAVE *"m";1;"c
adeia" DATA a$()
```

o que produzirá *Nonsense in BASIC* (Disparate em BASIC).

Código-máquina e bytes podem ser armazenados acrescentando a palavra-chave CODE seguida de dois números, separados por uma vírgula. O primeiro número é o endereço de início, e o segundo é o número de bytes. Ambos os números são opcionais nas instruções LOAD e VERIFY. Também podem ser armazenadas figuras do écran, usando SCREEN$. Ao contrário do que acontece com ficheiros de cassetes, é possível verificar (VERIFY) figuras guardadas.

Um erro *Wrong file type* (Tipo de ficheiro errado) pode ser causado por se tentar carregar um ficheiro com o comando errado, isto, é:

```
LOAD *"m";1;"teste" code
```

se «teste» era um programa.

CATálogo

Para ficar a saber que ficheiros se tem num cartucho deve ser usado o comando CAT (abreviatura de catálogo). Ele tem a forma:

```
CAT d
```

onde «d» é o número do drive. Isto produz no écran o título do cartucho, depois uma linha em branco, a seguir uma lista alfabética de todos os ficheiros do cartucho, e finalmente um número. O número indica, em kilobytes (K) quanto do cartucho não está utilizado, e deve ser sempre pelo menos 85K num cartucho em branco. O

catálogo pode ser enviado para outros canais que não o écran usando:

```
CAT #n,d
```

onde «n» é o número da stream. Por exemplo:

```
CAT #3,2
```

produzirá um catálogo do drive 2 na stream 3, normalmente a impressora ZX. Ele somente listará os primeiros 50 ficheiros que encontra: assim, se tiver mais do que 50, um CAT não os mostrará todos.

Que acontece se você quiser saber que ficheiros estão num cartucho respeitantes a um certo programa? Uma maneira bastante rudimentar e lenta seria limpar o écran, fazer um CAT, e então ler cada carácter do écran usando a função SCREEN$. Este método é bastante incómodo e sem utilidade se existem mais de 20 ficheiros no cartucho, pois que o écran rola para cima, perdendo-se os primeiros nomes de ficheiros. A melhor maneira é usar a rotina 14-z$, mencionada no capítulo antecedente para estabelecer a stream 14, para acrescentar caracteres à variável z$. Utilize:

```
LET z$="": CAT #14,d
```

e z$ conterá todo o catálogo, para você o examinar como quiser. Para ser capaz de usar a informação em z$, o seu formato deve ser conhecido e é como segue.

Em primeiro lugar há dez caracteres que são o título do cartucho, seguidos por dois códigos de nova linha (CHR$ 13). Cada nome de ficheiro está armazenado alfabeticamente, cada um consistindo de dez caracteres, depois uma nova linha. Finalmente há uma nova linha, depois um ou dois caracteres que dão o número de kilobytes ainda disponíveis, e para terminar uma outra nova linha. O formato é tornado mais claro pela sub-rotina seguinte, que produz no écran um catálogo bem apresentado. O código-máquina da stream 14-z$ deve ter entrado e sido estabelecida antes do GO SUB 2000.

Listagem «Catálogo perfeito»

```
1999 REM Catalogo perfeito
2000 LET z$="": CAT #14,d
2010 CLS
2020 PRINT "Nome do cartucho"; I
NVERSE 1;z$( TO 10)
2025 PRINT
2030 LET z$=z$(13 TO )
2040 LET f=0
2050 IF LEN z$(10 THEN GO TO 210
0
2060 LET f=f+1
2070 PRINT f;". ";z$( TO 10),
2080 LET z$=z$(12 TO )
2090 GO TO 2050
2100 PRINT ''f;"Ficheiros restan
tes";z$(2 TO LEN z$-1);"K"
2110 RETURN
```

Armazenamento de variáveis

Se tiver um programa grande, que requeira uma certa quantidade de quadros e variáveis cruciais, guardar a totalidade do programa e variáveis (usando SAVE normal) ou guardar cada quadro com um nome de ficheiro diferente (usando SAVE... DATA) pode ser um processo que gasta muito tempo. As sub-rotinas que se seguem permitem guardar todas as variáveis dum programa num cartucho e depois voltar a carregá-las numa fase subsequente, possivelmente num programa diferente. As variáveis são guardadas no drive sob os nomes «var1» e «var2», mas isto poder ser mudado por conveniência alterando as linhas 4040, 4050, 4110 e 4130. Não é possível fazer os nomes de ficheiros e os números dos drives em variáveis, por causa da técnica usada para as guardar. A propósito, não é um erro de impressão — as linhas 4010 e 4015 são iguais, e ambas são necessárias. A rotina SAVE pode ser chamada por GO SUB 4000, mas a rotina LOAD deve ser iniciada com GO TO 4100 — GO SUB não pode ser usada por conter uma instrução CLEAR, que limpa todos os precedentes GO SUB. Deste modo, a linha 4140 deve ser corrigida para saltar para trás até à linha correcta do programa de chamada.

Listagem «Guardar e carregar variáveis»

```
3999 REM SAVE as variaveis
4000 DEF FN p(p)=PEEK p+256*PEEK
 (p+1)
4010 LET l=FN p(23641)-FN p(2362
7)
4015 LET l=FN p(23641)-FN p(2362
7)
4020 POKE 23565,l-256*INT (l/256
)
4030 POKE 23566,INT (l/256)
4040 SAVE *"m";1;"var1"CODE 2356
5,2
4050 SAVE *"m";1;"var2"CODE FN p
(23627),l
4060 RETURN
```

```
4098
4099 REM LOAD as variaveis
4100 CLEAR
4110 LOAD *"m";1;"var1"CODE 2356
5
4120 DIM a$(FN p(23565)-7)
4130 LOAD *"m";1;"var2"CODE FN p
(23627)
4140 GO TO xxxx: REM resto do pr
ograma
```

Cópias múltiplas

Como foi mencionado, recomenda-se que se conserve sempre cópias de apoio de programas e dados importantes em cartuchos alternados, ou em cassetes. Contudo, incorporado no Spectrum está uma facilidade não documentada de salvar um ficheiro um número qualquer de vezes no mesmo cartucho, com o mesmo nome de ficheiro. Contudo, se tentar guardar um ficheiro com o mesmo nome com o BASIC verificar-se-á um erro. Para fazer uma cópia múltipla de um programa ou de qualquer outro tipo de ficheiro, mesmo antes de fazer o SAVE, deve introduzir a instrução POKE 23791,x, onde «x» representa o número de cópias, desde 1 a 255, embora o valor 2 seja usualmente melhor. Então, quando é feito o SAVE, o ficheiro é guardado *x* vezes. O número de cópias é reposto em 1 depois do SAVE. Isto não é visível para si, visto que uma instrução CAT imprimirá só uma vez o nome do ficheiro. A vantagem deste método de cópias múltiplas é que se você, ou qualquer outra pessoa, acidentalmente apagar o ficheiro, somente será apagada uma cópia. As restantes cópias manter-se-ão no cartucho e podem ser carregadas da maneira normal. Se fizer *x* cópias dum ficheiro, para

apagá-las todas tem de usar o comando ERASE *x* vezes. É claro que *x* cópias ocuparão *x* vezes mais espaço do cartucho. Note-se que tais cópias múltiplas **NÃO** estão em segurança se se executar o comando FORMAT «m»;, o qual apagará duma vez todas as cópias.

CAPÍTULO 3

Manipulação do ficheiro do microdrive

Ficheiros em série

Um ficheiro é um bloco de dados ao qual podem ser acrescentados itens, e do qual os itens podem ser lidos. Com ficheiros *em série* os dados têm de ser lidos e escritos numa certa ordem. Se, por exemplo, você quiser o décimo item, terá de começar por ler os nove primeiros itens de dados antes de poder ler o décimo. No Spectrum, os ficheiros são normalmente armazenados num cartucho, e são do tipo em série. [O outro tipo de ficheiro, acesso aleatório *(random acess),* não é disponível com o BASIC.]

No cap. 1 explicou-se como funcionam as streams e foram descritos os canais «K», «S» e «P». Para armazenar dados num microdrive, de outro modo que não seja guardando-os como programas ou quadros, é utilizado um outro especificador de canal — «m» (ou «M»), para microdrive. De modo semelhante ao que já foi descrito, a instrução OPEN # deve ser usada para ligar o canal «m» a uma stream, contudo a sua sintaxe é ligeiramente diferente, visto que o Spectrum requer mais informação. Tem a forma:

```
OPEN #n;"m";d;"nome
```

o que criará um novo canal com o especificador «m» no drive «d» com o nome «nome», e liga depois o novo canal à stream número

«n». Como o ficheiro «nome» acaba mesmo de ser criado, não contém dados; desse modo, é designado por ficheiro de escrita *(write file)*.

Ficheiros de escrita

Estes ficheiros não existiam antes de o comando OPEN# ter sido executado e são criados por ele. O principal comando para enviar dados para um ficheiro do microdrive é a instrução PRINT#n; que funciona duma maneira semelhante à instrução PRINT normal, mas «imprime» os dados num cartucho. Como uma simples demonstração, execute o programa seguinte e verifique quando o microdrive na realidade opera:

```
100 OPEN #4;"m";1;"fich teste"
110 FOR i=1 TO 500
120 PRINT i;"";
130 PRINT #4;i
140 NEXT i
150 CLOSE #4
```

Você devia esperar que o drive funcionasse a maior parte do tempo, à medida que imprime cada número, mas tal não acontece! Depois dum comando OPEN#, o drive funciona por instantes enquanto procede à busca dum ficheiro com o mesmo nome, o que ele não deve encontrar por ser um ficheiro de escrita. O que OPEN# também faz é criar uma área de memória chamada *buffer*, que usa como armazenamento temporário dos caracteres. Quando se lhe deparar uma declaração PRINT# ele armazenará os caracteres no buffer, mas não os escreverá no cartucho a não ser que o buffer esteja cheio. Quando ele estiver cheio, o drive funcionará durante cerca de um segundo enquanto transfere os 512 caracteres do buffer para o cartucho, e depois limpa o buffer, que fica pronto para receber mais dados. É esta a razão por que o drive somente funciona durante umas breves ocasiões, pois que o buffer enche-se

mais de quatro vezes. Quando se usam ficheiros de escrita é absolutamente necessária uma instrução CLOSE#, porque, se o buffer não está cheio e você não usar CLOSE, os últimos dados que você pensou que enviou por meio de PRINT# nunca ficarão no cartucho, e o ficheiro ficará incompleto. Uma instrução CLOSE# enviará para o cartucho o que ficou no buffer, após o que retira o buffer da memória. Se você der uma instrução CLEAR# enquanto uma stream de escrita está aberta, o ficheiro ficará incompleto e não utilizável. Um tal ficheiro parcial deve ser apagado. Se você tentar abrir um ficheiro de escrita que está já aberto noutra stream, ocorrerá a mensagem *Reading from a 'write' file* (A ler dum ficheiro de escrita).

Ficheiros de leitura

Uma vez que um ficheiro que tenha sido criado por OPEN# tenha recebido dados e sido fechado, ele torna-se um ficheiro de «leitura», e é visível no catálogo do cartucho. Para dele serem lidos dados, ele tem de ser aberto, usando outra vez OPEN# com a mesma sintaxe anterior. A leitura dos dados é usualmente feita com o comando INPUT#. Para ler no ficheiro criado no programa precedente, introduza o seguinte:

```
100 OPEN #4;"m";1;"fich teste"
110 INPUT #4;i
120 PRINT i;"";
130 GO TO 110
```

Isto cria inicialmente um canal de leitura de ficheiro na stream 4. A variável «i» é então lida do ficheiro (linha 110) e impressa no écran. O programa faz então um ciclo para trás mas eventualmente pára com um erro *Fim do ficheiro (End of file)*. (Veremos num capítulo mais adiante como detectar o fim do ficheiro antes de ele ocorrer.) Como se deve ter apercebido enquanto o programa estava em execução, é também usado um buffer com os ficheiros de leitura, e mais uma vez tem um comprimento de 512 caracteres. Quan-

do é encontrado um INPUT#, é lido do cartucho um bloco de 512 caracteres (ou menos, se é o último bloco), o qual é armazenado no buffer. Quando a rotina de INPUT (ou INKEY$#) necessita de um carácter, ele é extraído do buffer. Se é o último carácter que está no buffer, é então lido do cartucho um outro bloco de 512, até que não haja mais no ficheiro, momento em que ocorrerá um erro *End of file* (Fim do ficheiro).

Há uma outra maneira de ler caracteres dum ficheiro, à parte usar INPUT# — a função INKEY$#, a qual foi descrita brevemente no cap. 2. Ao contrário de INPUT, que junta uma sequência de caracteres até que atinge o fim duma linha e depois a copia para uma variável, INKEY$# n lê só um simples carácter. Ao contrário da função INKEY$ normal, ela «espera» por um carácter — INKEY$# com uma stream do microdrive nunca dará como resultado uma cadeia vazia («»). Quando se tiver acabado de ler um ficheiro, deve encerrar-se com o comando CLOSE#. Se bem que não seja tão importante como no caso dos ficheiros de escrita, é aconselhável fazê-lo sempre por uma questão de boa apresentação e para evitar que vastas porções da memória sejam usadas por buffers desnecessários. É perfeitamente correcto fazer CLEAR# com ficheiros de leitura abertos.

Quando se usar instruções INPUT#, o contador usado pelo Spectrum para produzir a pergunta «scroll?» («rolar?») é reposto a zero; assim, se você estiver a ler um ficheiro e a imprimi-lo no écran, ele rolará continuamente. Se isto não é desejável no seu programa, use uma linha como:

```
LET sc=PEEK 23692: INPUT (seja o
que for): POKE 23692,sc
```

A razão disto é que o contador é posto a zero por qualquer declaração INPUT, o que usualmente nunca é notado. De facto, uma maneira simples de desarmar o contador de «scroll?» num programa é usar a declaração:

```
INPUT ""
```


Notas sobre PRINT e INPUT

Deve ter muito cuidado quando usar as instruções PRINT# e INPUT# com os ficheiros do microdrive, por causa do efeito dos separadores usados entre os argumentos. Quando estiver a imprimir no écran ou na impressora, uma vírgula (,) levá-lo-á para o início da seguinte meia linha, e um apóstrofe (') fará uma nova linha. Contudo não há «linhas» nos ficheiros do microdrive, e imprimir uma vírgula num ficheiro (por exemplo: PRINT#4;a,b) enviará na realidade o código de controle para vírgula, isto é um CHR$ 6. De modo semelhante, um apóstrofe enviará CHR$ 13 (um código de nova linha). O código de carácter 6 (CHR$ 6) pode confundir o Spectrum quando você tentar usar INPUT# (mas não INKEY$#). Tente, por exemplo, o seguinte programa:

```
10 OPEN #4;"m";1;"teste erro"
20 LET a$="primeiro": LET b$="se
gundo"
30 PRINT #4;a$,b$
40 CLOSE #4
50 OPEN #4;"m";1;"teste erro"
60 INPUT #4;a$
70 INPUT #4;b$
80 CLOSE #4
```

Você não esperava o erro *Fim de ficheiro* na linha 70, mas o PRINT na linha 30 na realidade enviou para o cartucho o texto «primeiro», depois um carácter vírgula (CHR$ 6), seguindo-se o texto «segundo», e finalmente uma nova linha (CHR$ 13). Uma instrução INPUT# pressupõe que as variáveis num ficheiro sejam separadas por CHR$ 13, assim ela lê «a$» como «primeiro» + CHR$ 6 + «segundo», e esvazia o ficheiro; depois, tentando ler a seguir «b$», produziu o erro.

Os separadores nas instruções INPUT # também criam problemas inesperados. Se você tentar:

```
10 OPEN #4;"m";1;"teste fich"
20 INPUT #4;a$,b$
```

(partindo do princípio de que o ficheiro «teste fich» existe) verificar-se-á o erro *A escrever num ficheiro de leitura*. Isto sucede porque a vírgula na linha 20 está a tentar enviar um CHR$ 6 para um ficheiro de leitura. Se as aspas (») estão incluídas nos ficheiros de dados, a leitura de cadeias usando INPUT # pode causar problemas; por isso, deve sempre ser usado INPUT # ... LINE. As regras de ouro para usar PRINT # e INPUT # são:

1. Separar sempre as variáveis com apóstrofes quando usar PRINT # ou usar linhas separadas, por exemplo: PRINT # 4; a$'d$ ou PRINT # 4; a$: PRINT; # 4b$
2. Usar sempre o ponto e vírgula como separador nas instruções INPUT, e usar LINE para variáveis em cadeia, por exemplo: INPUT # 4; LINE a$; LINE B$

O uso de MOVE com os ficheiros do microdrive

O comando MOVE tem muitos usos, mas aqui só serão discutidos os que são directamente aplicáveis aos ficheiros dos microdrives. Se você tiver um ficheiro de dados e quiser ver o que nele está contido sem ter de o abrir e ler carácter a carácter, pode fazer:

```
MOVE "m";d;"nome" TO #2
```

(onde TO é uma palavra-chave), o que imprime o conteúdo de um determinado ficheiro na stream 2, o écran. [Se tentar aplicar MOVE a qualquer outro tipo de ficheiro, tal como programas, verificar-se-á um erro *Wrong file type* (Tipo de ficheiro errado).] Se quiser uma cópia do ficheiro em papel, pode usar:

```
MOVE "m";d;"nome" TO #3
```


o que imprimirá na stream 3, a impressora ZX. Pode também usar MOVE para copiar ficheiros de dados:

```
MOVE "m";d1;"ficheiro 1" TO "m";
d2;"ficheiro 2"
```

que copia os dados do «ficheiro 1» no drive «d1» para o drive «d2» com o nome «ficheiro 2».

Como acrescentar ficheiros

Suponha que tem um ficheiro contendo uma porção grande de dados (como, por exemplo, a lista do seu software) e quer acrescentar alguns itens no fim dele. Como não pode imprimir num ficheiro de leitura, tem de criar um novo ficheiro, copiar o conteúdo do velho ficheiro para ele e, enquanto ainda está aberto, juntar os novos dados. Uma maneira seria ler cada item do antigo ficheiro e depois imprimir no novo, mas isto seria bastante lento e ineficiente, e não funcionaria com certas formas de dados. O melhor modo de o fazer é usar o comando MOVE, como se mostra no exemplo seguinte, onde se pretende adicionar as cadeias «Invasores Espaciais» e «25 de Dezembro de 1983» ao ficheiro «software»:

```
10 OPEN #4;"m";1;"software2"
20 MOVE "m";1;"software" TO #4
30 PRINT #4;"Invasores espaciais
""25 Dezembro 1983"
40 CLOSE #4
```

Se você quisesse que o novo ficheiro tivesse o mesmo nome que o antigo, podia juntar as linhas:

```
50 ERASE "m";1;"software"
60 MOVE "m";1;"software2" TO "m"
;1;"software"
70 ERASE "m";1;"software2"
```

O programa acima será executado muito mais depressa se os dois ficheiros estiverem armazenados em microdrives diferentes.

O comando MOVE é também muito útil para adicionar dois ficheiros de dados — por exemplo, formar o ficheiro «terceiro» adicionando o ficheiro «primeiro» com o ficheiro «segundo»:

```
10 OPEN #4;"m";1;"terceiro"
20 MOVE "m";1;"primeiro" TO #4
30 MOVE "m";1;"segundo" TO #4
40 CLOSE #4
```

A velocidade deste programa pode também ser grandemente melhorada se se usarem dois ou mesmo três microdrives, com um ficheiro cada um.

Utilização de LIST

Se bem que o método principal de escrever dados num cartucho seja usando PRINT #, a instrução LIST # pode também ser usada. Ela fará a listagem dum programa em BASIC numa stream, que pode ser um canal do microdrive. Quando você voltar a ler o ficheiro, um símbolo (por exemplo, STOP) será apenas um carácter (CHR$ 226 é o código para STOP) e não caracteres individuais com espaços entre as palavras-chave. Converter um programa em forma de caracteres usando LIST # pode ser muito útil se você deseja processar ou procurar através do seu programa BASIC, talvez com um processador de texto. O programa seguinte, «Programa de busca», efectuará uma busca no ficheiro de dados criado por instruções OPEN #, LIST # e depois CLOSE # para qualquer sequência particular, e imprimirá no écran qualquer linha que a contenha. Isto é muito útil para a procura de nomes de variáveis, ou certas instruções GO TO. Pode também ser usado para procurar através de qualquer tipo de ficheiro de dados, não só para listagens de programas. Eventualmente parará com a mensagem *End of file* (Fim do ficheiro).

```
 999 REM Programa de procura
1000 INPUT "No. do drive?";d;"No
me do ficheiro?";LINE f$
1005 CLOSE #4: OPEN #4;"m";d;f$
1010 INPUT "Cadeia a procurar?";
LINE b$
1020 INPUT #4;LINE a$
1030 FOR i=1 TO LEN a$-LEN b$
1035 IF a$(i TO i+LEN b$-1)=b$ T
HEN PRINT a$: PAUSE 50: GO TO 10
50
1040 NEXT i
1050 GO TO 1020
```

Se você quer procurar símbolos (por exemplo, GO TO) deve escrever THEN, o que põe o Spectrum no modo K, depois premir a tecla adequada (G para GO TO) e em seguida apagar a palavra THEN.

End of File (EOF) (Fim do ficheiro)

Quando estiver a ler um ficheiro, você eventualmente atingirá o fim dele. Se tentar ler qualquer coisa mais será produzida a mensagem *Fim do ficheiro*. Há, contudo, várias maneiras de detectar um EOF antes que ele ocorra, o que torna os programas mais bem apresentados e mais eficientes.

A maneira mais simples de detectar um EOF é imprimir um carácter especial ou uma sequência de caracteres, que não seria nor-

malmente usada no fim dum ficheiro, mesmo antes de o fechar. A minha preferência pessoal é usar a cadeia CHR$#0+CHR$#0, que nunca usaria para qualquer outro fim. Este é o melhor método para a maior parte dos programas, mas não pode ser usado se você tiver um programa destinado a ler todas as espécies de ficheiros de dados, ou que pode ler ficheiros de programas ou código-máquina (usando um método que será explicado mais adiante).

Outros BASIC baseados em discos têm usualmente a função ON EOF GO TO ou mais geralmente ON ERROR GO TO, que faz o salto de controle para uma particular linha se ocorre um EOF (ou qualquer outro erro). Infelizmente o Spectrum não tem essa função, mas umas tantas linhas de BASIC podem detectar 99% dos EOF. Se juntar a um programa as linhas que se seguem (tão perto do princípio quanto possível), então a função FN E(x), onde «x» é o número da stream, dará 1 se o ficheiro está vazio, de outro modo dará 0.

```
 10 DEF FN E(A)=((INT (PEEK (FN
D(A)+67)/2)-2*INT (PEEK (FN D(A
)+67)/4)AND(FN P(FN D(A)+11))=FN
P(FN D(A)+69))))
 11 DEF FN P(P)=PEEK P+256*PEEK
(P+1)
 12 DEF FN D(D)=FN P(D*2+23574)
+FN P(23631)-1
```

(se a stream «x» não está aberta ou não é um canal «M», o resultado não terá significado). O 1% das vezes em que este método falha ocorre quando o número de caracteres no buffer é um múltiplo exacto de 512, o que graças a Deus é raro. Quando estivermos a criar um ficheiro de escrita, a meneira de termos a certeza de que isto não acontece é usar a função:

```
FN P(FN D(X)+11)
```

mesmo antes de fechar a stream «x». Se o resultado é zero, então imprima de antemão um carácter extra na stream.

Alternativamente, a seguinte rotina em código-máquina pode ser usada para executar a função ON EOF GO TO. Tem um comprimento de 67 bytes e pode ser posicionada em qualquer parte da memória. Antes de fazer um GO SUB 8150 deve estabelecer-se a variável «eof» na posição necessária e «line» no necessário número de linha (que deve existir). A rotina coloca (POKE) o código e executa-o. Depois disto, quando ocorre qualquer EOF não será produzida mensagem — o interpretador saltará para a linha especificada. A função é cancelada quando ocorre qualquer erro, incluindo um EOF. Se quer mudar o número de linha do salto de erro, mude «line» e GO SUB 8220. Quando ocorre um EOF, para encontrar em que linha ele ocorreu use

```
LET errlinha=PEEK 23753+256*PEEK
 23754
```

Faça isto antes de quaisquer operações do microdrive, de outro modo dará um resultado não válido.

Listagem «ON EOF GOTO»

```
8145 REM ************************
8146 REM *      ON EOF GOTO     *
8147 REM ************************
8148 REM eof=posicao de inicio,l
inha=salto de erro
8149 REM recomendado:65190/32490
8150 RESTORE 8250
8160 LET c=0
8170 FOR i=eof TO eof+66
8180 READ a: LET c=c+a
8190 IF a()260 THEN POKE i,a
8200 NEXT i
```

```
8210 IF c<>6456 THEN PRINT "Erro
 de verificacao da soma": STOP
8220 POKE eof+49,linha-256*INT (
linha/256)
8225 POKE eof+50,INT (linha/256)
8230 RANDOMIZE USR eof
8240 RETURN
8250 DATA 33,17,0,9,235,42,61,92
8260 DATA 115,35,114,207,49,1,0,
0
8270 DATA 201,42,61,92,58,58,92,
254
8280 DATA 7,194,3,19,94,35,86,21
3
8290 DATA 205,176,22,253,203,55,
174,205
8300 DATA 110,13,42,69,92,34,201
,92
8310 DATA 17,260,260,33,66,92,11
5,35
8320 DATA 114,35,54,1,253,54,0,2
55
8330 DATA 195,125,27
```


Utilização de programas, etc., como ficheiros de dados

Normalmente, se você tentar abrir um canal do microdrive com o nome do ficheiro dum programa, ou qualquer outro tipo de ficheiro não-dados, ocorrerá o erro *Wrong type file* (Tipo de ficheiro errado). Pode ser na verdade muito útil ler tais ficheiros alternados do BASIC, e isto pode ser feito com a rotina em código-máquina chamada «OPEN # qualquer coisa». É quase equivalente à instrução OPEN, mas abrirá qualquer ficheiro, independentemente do seu tipo. Também o informará se o ficheiro existe ou não, o que em si mesmo é muito útil. Antes do GO SUB 8350, a variável «open» deve estar na posição requerida para a rotina (recomenda-se 32320 para 16K e 65090 para 48K). O GO SUB colocará o código, mas não o executará.

Listagem «OPEN # qualquer coisa»

```
8344 REM ************************
8345 REM *OPEN # qualquer coisa*
8346 REM ************************
8347 REM open=posicao de inicio
8348 REM recomendado:65090/32320
8350 RESTORE 8400: LET c=0
8360 FOR i=open TO open+93
8370 READ a: LET c=c+a
8375 POKE i,a
8380 NEXT i
8385 IF c<>10725 THEN PRINT "Err
o de verificacao da soma": STOP
8390 RETURN
```

```
8400 DATA 58,216,92,205,39,23,33
,17
8410 DATA 0,175,237,66,1,0,0,216
8420 DATA 50,215,92,33,10,0,34,2
18
8430 DATA 92,42,123,92,34,220,92
,58
8440 DATA 216,92,135,33,22,92,95
,22
8450 DATA 0,25,217,229,217,229,2
07,34
8460 DATA 221,203,24,70,40,13,17
5,207
8470 DATA 33,207,44,225,217,225,
217,1
8480 DATA 1,0,201,221,203,4,190,
175
8490 DATA 229,207,33,209,225,115
,35,114
8500 DATA 221,126,67,230,4,198,2
,79
8510 DATA 6,0,217,225,217,201
```

Quando você a quiser usar, tem de lhe dizer qual o drive, stream e ficheiro em que está interessado. Para isto deve fazer POKE 23766 com o número do drive, POKE 23768 com o número da stream, e POKE com as dez primeiras posições na área de gráficos definida pelo utilizador com o nome do ficheiro. Por exemplo,

se quisesse OPEN # 6; «m»; 2; «teste» (onde «teste» é qualquer tipo de ficheiro), as intruções seriam:

```
3500 POKE 23766,2: REM numero do
 drive
3510 POKE 23768,6: REM numero da
 stream
3520 LET a$="test"
3530 FOR i=1 TO 10
3540 IF i>LEN a$ THEN POKE USR"a
"+i-1,32: GO TO 3560
3550 POKE USR "a"+i-1,CODE a$(i)
3560 NEXT i
3570 LET a=USR open: REM chamar
a rotina
```

De notar que se o nome do ficheiro tem menos de dez caracteres de comprimento, deve ser acrescentado o apropriado número de espaços (CHR$ 32) e colocado o (POKE) (linha 3540). O valor devolvido pela função «USR open» («a» no exemplo) é como se segue:

0 — stream já aberta
1 — ficheiro não encontrado
2 — ficheiro de dados
6 — não é um ficheiro de dados

Se o ficheiro não existe, a stream é fechada e o ficheiro não é criado. Por conseguinte o método serve apenas para ficheiros de leitura — ficheiros de escrita não são por ele criados, a não ser o comando OPEN #.

Se o número devolvido é 6, isto é, se não é um ficheiro de dados, como sabe você que tipo de ficheiro é? A resposta reside nos primeiros nove caracteres lidos do ficheiro. Estes caracteres contêm

todos os atributos do ficheiro, incluindo o tipo. O primeiro código de carácter determina o tipo, por conseguinte:

0 programa BASIC
1 quadro numérico
2 quadro de cadeia
3 bytes

Os oito bytes seguintes contêm pormenores tais como o comprimento, o início, número da linha, etc., e são na realidade tomados das variáveis do sistema HD_00 a HD_11 (ver o Apêndice A para os pormenores exactos). Depois dos primeiros nove bytes, o resto é o ficheiro real. O programa seguinte, «Verdadeiro catálogo», é uma versão muito melhorada da função CAT. Além de fornecer cada nome de ficheiro, dá também todos os outros pormenores acerca do ficheiro — o seu tipo, comprimento, número da linha do arranque automático, etc. Para funcionar precisa das rotinas em código-máquina «Stream 14-z$« e «Open # qualquer coisa». É mais lento do que um CAT, mas dá uma grande quantidade de informações úteis.

Listagem «Verdadeiro catálogo»

```
4997 REM **************************
4998 REM * VERDADEIRO CATALOGO *
4999 REM **************************
5000 LET z$="": CAT #14,d
5010 CLS : POKE 23766,d: POKE 23
768,15
5020 PRINT INVERSE 1;"Titulo:";z
$( TO 10)
5030 LET z$=z$(13 TO )
5040 IF LEN z$(10 THEN GO TO 533
0
```



```
5050 FOR i=1 TO 10
5060 POKE USR "a"+i-1,CODE z$(i)
5070 NEXT i
5080 CLOSE #15: LET z=USR open
5090 PRINT z$( TO 10);"  ";
5100 IF z=2 THEN PRINT "Ficheiro
 de dados": GO TO 5300
5110 LET a$="": FOR i=1 TO 9
5120 LET a$=a$+INKEY$#15: NEXT i
5130 LET z=CODE a$(1)
5140 GO TO 5150+z*40
5149 REM z=0  Programa
5150 PRINT "Programa  LINE ";COD
E a$(8)+256*CODE a$(9)
5160 GO TO 5300
5189 REM z=1  quadro numerico
5190 PRINT "Quadro ";CHR$ (CODE
a$(6)-32);"()"
5200 GO TO 5300
5229 REM z=2  quadro de cadeia
5230 PRINT "Quadro ";CHR$ (CODE
a$(6)-96);"$()"
5240 GO TO 5300
5269 REM z=3  Codigo
5270 PRINT "Codigo ";CODE a$(4)+
256*CODE a$(5);
5280 PRINT ",";CODE a$(2)+256*CO
```

```
DE a$(3)
5300 CLOSE #15
5310 LET z$=z$(12 TO )
5320 GO TO 5040
5330 PRINT 'z$(2 TO LEN z$-1);"K
 bytes restantes"
5340 RETURN
```

Criação de ficheiros do tipo não-dados

Tal como a capacidade para ler a partir dum cartucho ficheiros do tipo não-dados, é possível criar qualquer tipo de ficheiro dum programa BASIC. O uso desta capacidade pode ser bastante difícil, e não se recomenda aos principiantes que o façam, visto que os erros com o seu uso podem fazer falhar o computador.

O método para o conseguir é fazer seguir a um comando OPEN #, para criar o ficheiro, linhas tais como:

```
100 DEF FN p(p)=PEEK p+256*PEEK(
p+1)
110 POKE FN p(s*2+23566+8)+FN p(
23631)+66,4
```

onde «s» é o número da stream no comando OPEN. Isto leva o sistema a criar fragmentos de ficheiro do tipo não-dados quando quaisquer dados da stream estão escritos num cartucho. Depois do POKE você deve imprimir (PRINT) nove caracteres na stream escolhida. O primeiro define o tipo do ficheiro e os oito seguintes determinam os parâmetros do ficheiro e são os mesmos que os mencionados acima para a leitura de ficheiros — as variáveis de sistema HD_00 a HD_11 (ver o Apêndice A).

Como um exemplo, o programa seguinte guarda todas as variáveis BASIC num programa como a rotina do cap. 2, mas com uma grande diferença: em lugar de as guardar como dois ficheiros CODE, guarda-as como um ficheiro programa, podendo desse modo ser combinados (MERGE) com outros programas. Pode ser bastante mais lento do que o outro método, mas é muito mais flexível.

Listagem «Guardar variáveis»

```
3999 REM guardar as variaveis co
mo programa
4000 OPEN #4;"m";1;"variaveis"
4010 DEF FN p(p)=PEEK p+256*PEEK
(p+1)
4020 LET d=FN p(4*2+23566+8)+FN
p(23631)-1
4030 IF PEEK (d+4)<>CODE "M" THE
N PRINT "Erro": STOP
4040 POKE d+67,4: REM passar tem
po
4045 FOR i=1 TO 2: NEXT i
4050 LET z=FN p(23641)-FN p(2362
7)-1
4055 LET z=FN p(23641)-FN p(2362
7)-1
4060 PRINT #4;CHR$ 0;:REM Progra
ma flag
4070 PRINT #4;CHR$ (z-256*INT (z
/256));CHR$ INT (z/256);
4080 PRINT #4;CHR$ PEEK 23627;CH
```

```
R$ PEEK 23628;: REM Inicio
4090 PRINT #4;CHR$ 0;CHR$ 0;: RE
M comprimento do programa
4100 PRINT #4;CHR$ 255;CHR$ 255;
:REM 'numero da linha'
4110 FOR i=FN p(23627) TO FN p(2
3627)+z-1
4120 PRINT #4;CHR$ PEEK i;
4130 NEXT i
4150 CLOSE #4
4160 RETURN
```

A linha 4000 cria o ficheiro de escrita, a linha 4040 faz o necessário POKE. As linhas 4045 e 4050 podem parecer supérfluas, mas ambas são vitais. A variável «z» contém então o número de bytes na área das variáveis (linha 4055), depois a linha 4060 imprime o primeiro carácter, para assinalar um ficheiro programa. A linha 4070 imprime em seguida os dois caracteres para definir o comprimento do ficheiro, e a linha 4080 imprime dois para definir o início da área variável. A linha 4090 estabelece o comprimento do programa como 0, e a linha 4100 estabelece 65535 como número de linha para o arranque automático (isto é, não há arranque automático). O ciclo de 4110 a 4130 envia então cada byte das variáveis para o ficheiro, antes de a linha 4150 o fechar.

Outras maneiras de ler dados

Se bem que INPUT # e INKEY$ # sejam os principais comandos para ler ficheiros, o MOVE pode ser mais útil. Se a rotina «Stream 14-z$» está activa, a linha:

```
LET z$="": MOVE "m";1;"ficheiro"
 TO #14
```


lerá imediatamente todo o ficheiro de dados a partir do cartucho e colocá-lo-á, desde que a memória o permita, na variável de cadeia «z$». Ele pode então ser manipulado e modificado como se quiser, e depois voltar a ser escrito num outro ficheiro com uma simples instrução PRINT, por exemplo:

```
OPEN #4;"m";1;"ficheiro 2": PRIN
T #4;z$;: CLOSE #4
```

Este método tem um efeito mínimo no uso e desgaste dos cartuchos, o que pode ser um factor importante quando se está a lidar com grandes ficheiros de dados usados frequentemente, visto que o drive só funcionará duas vezes — uma vez para ler e outra para escrever.

Rotina «Status»

A próxima rotina, chamada «Status», permite que um programa determine se um certo microdrive está ou não ligado, se nele há um cartucho, e se este é de escrita protegida. Antes de fazer um GO SUB 8550, a variável «stat» deve ser estabelecida na desejada posição da memória.

Listagem "Status"

```
8545 REM ************************
8546 REM *       Status         *
8547 REM ************************
8548 REM stat=endereco de inicio
8549 REM recomendado:64960/32190
8550 RESTORE 8630: LET c=0
8560 LET x2=INT ((stat+100)/256)
: LET x1=stat-256*x2+100
8570 FOR i=stat TO stat+128
```

```
8580 READ a: LET c=c+a
8590 POKE i,a
8600 NEXT i
8610 IF c<>14341+4*(x1+x2) THEN
PRINT "Erro de verificacao da so
ma": STOP
8620 RETURN
8630 DATA 58,214,92,243,24,33,33
,136
8640 DATA 19,43,125,180,32,251,3
3,136
8650 DATA 19,6,6,219,239,230,4,3
2
8660 DATA 4,16,248,24,84,43,124,
181
8670 DATA  32,239,1,0,0,24,34,17
8680 DATA 0,1,237,68,198,9,79,6,
8690 DATA 8,13,32,20,122,50,247,
0
8700 DATA 62,238,211,239,205,x1,
x2,62
8710 DATA 236,211,239,205,x1,x2,
24,17
8720 DATA 62,239,211,239,123,211
,247,205
8730 DATA x1,x2,62,237,211,239,2
05,x1
```



```
8740 DATA x2,16,214,122,211,247,
62,238
8750 DATA 211,239,24,162,197,245
,1,135
8760 DATA 0,11,120,177,32,251,24
1,193
8770 DATA 201,219,239,230,1,1,1,
0
8780 DATA 32,1,3,197,175,207,33,
193
8790 DATA 201
```

O GO SUB só põe o código-máquina no lugar, não o executa. Antes de usar, faça POKE 23766 com o número do drive e depois use algumas linhas, tais como:

```
2500 LET a=USR stat
2510 IF a=0 THEN PRINT "Microdri
ve nao ligado"
2520 IF a=1 THEN PRINT "Cartucho
 presente"
2530 IF a=2 THEN PRINT "Cartucho
 de escrita protegida"
```

Comandos de cores

No manual da interface Sinclair é feita uma referência de passagem ao facto de os comandos de cores não funcionarem depois de se usar outros canais que não os «K», «S» ou «P». É de facto

um problema sério, de que se deve estar particularmente consciente quando se estiver a lidar com ficheiros.

O problema é que depois de se usar os canais da interface como saída, as instruções permanentes de cores (como, por exemplo, PAPER 3) não têm efeito aparente. E, o que é mais importante, eles na realidade mandam dados para ficheiros de escrita. Para corrigir isto, use uma instrução «PRINT;» muda antes de estabelecer as cores permanentes. Para mostrar este defeito em acção, experimente o seguinte:

```
10 OPEN #4;"m";1;"corteste"
20 PRINT #4;"Primeira linha"
30 FLASH 1: PAPER 4: CLS
40 PRINT #4;"Segunda linha"
50 CLOSE #4
60 MOVE "m";1;"corteste" TO #2
```

A linha 10 cria um ficheiro de escrita «corteste», e a linha 20 manda para ele uma linha de dados. A linha 30 deve fazer todo o écran de cor verde flamejante, mas em vez disso envia os códigos de cor para o cartucho. As linhas 40 e 50 enviam uma outra linha de dados para o ficheiro e depois fecham-no. Finalmente a linha 60 imprime todo o ficheiro no écran, revelando a não desejada inserção dos códigos de cores. A maneira de neste caso arrumar o assunto é acrescentar a linha:

```
25 PRINT;
```

Acrescentos microdrive à listagem «Stream»

As linhas seguintes podem ser acrescentadas à listagem «Stream» para lhe permitir dar maior pormenor sobre as streams de microdrives.

```
1630 IF f$="M" THEN GO TO 2500
1840 IF FN P(D)<>8 AND FN P(D+2)
<>8 THEN RETURN
1850 PRINT "Saida da ROM sombra:
";FN P(D+5)
1860 PRINT "Entrada da ROM sombr
a :";FN P(D+7)
2499 REM Canal M
2500 PRINT INVERSE 1;"MICRODRIVE
"
2510 GO SUB 1800
2520 PRINT "Numero do drive  :";
PEEK (D+25)
2530 LET M=FN P(D+26)
2540 PRINT "Mapa do anel de fita
:"
2550 FOR I=0 TO 31: FOR J=1 TO 8
2560 POKE 16384+2048+2*32+J*256+
31-I,PEEK (M+I)
2570 NEXT J: NEXT I
2575 PLOT 0,95: DRAW 255,0: PLOT
 0,88: DRAW 255,0
2580 PRINT '"Area do mapa     :"
;M;"-";M+31
2590 PRINT "Nome do cartucho :";
2600 FOR I=D+44 TO D+53
2610 PRINT CHR$ PEEK I;
2620 NEXT I: PRINT
2630 PRINT "Nome do ficheiro :";
```

```
2640 FOR I=D+14 TO D+23
2650 PRINT CHR$ PEEK I;
2660 NEXT I: PRINT '"Espaco livr
e      :";
2670 LET F=0
2680 FOR I=0 TO 255
2690 LET F=F+NOT POINT (I,90)
2700 NEXT I
2710 PRINT F/2;"Kbytes"
2720 GO TO 1500
```

Listagem «Stream 2»

Os pormenores extras incluem o número do drive, o número do cartucho, o nome do cartucho e o nome do ficheiro, bem como o espaço livre no cartucho escolhido. É também mostrado um «mapa do anel *(loop)* de fita», o qual é uma espécie de código de barras que mostra graficamente quais as secções da fita que estão ocupadas. A figura 3 mostra a saída dum programa com um cartucho novo em folha. Quaisquer barras pretas no mapa do anel de fita marcam áreas que ou estão ocupadas ou não existem. A grande área a preto à esquerda mostra uma secção da fita que simplesmente não existe, pois que o sistema pode usar fitas mais compridas. As barras no meio mostram onde está a junra da fita, e a barra na extrema direita mostra o primeiro sector, que nunca é usado. O mapa do anel de fita para um cartucho usado conterá mais barras verticais, mostrando onde estão armazenados os programas e os dados.

A primeira adição principal para a listagem «Stream» consiste nas linhas 1840-1860. Estas pesquisam a existência de tipos de canais do interface — se eles existem, então as posições das rotinas de entrada e saída do ROM sombra são impressas. Estas linhas extras são também necessárias para as adições de RS232 e de rede ao programa. As linhas 2500-2720 imprimem vários pormenores sobre a stream. O modo como é impresso o mapa do anel de fita é chaman-


```
NUMERO DE STREAM :0
Especificador de canal:M
MICRODRIVE
Rotina de saida        :8
Rotina de entrada      :8
Saida de ROM sombra    :4568
Entrada de ROM sombra  :4368
Numero do drive        :1
Mapa do anel da fita:
[                                ]
Area do mapa           :23792-23823
Nome do cartucho       :21,8,83 3
Nome do ficheiro       :teste
Espaco livre           :90Kbytes
```

Fig. 3 — *Saída da listagem «Stream» para um cartucho novo*

do directamente ao écran os dados armazenados na memória, nas linhas 2530-2575. O espaço livre é calculado contando o número de barras brancas do mapa no écran (2670-2710).

Rotinas de alta pontuação de jogos (Recordes)

Para demonstrar uma aplicação de ficheiros de dados, temos a seguir algumas rotinas que podem adicionar uma pontuação de recordes a um jogo, armazenando a tabela de pontuação num ficheiro. Apresenta-se sob a forma de duas sub-rotinas principais (linhas 9000-) lidas no quadro, e deve ser usado antes que o jogo comece, e as linhas 9100-, que actualizam o quadro se é atingida uma nova pontuação mais elevada.

Rotina de «Alta pontuação de jogos (Recordes)»

```
8996 REM **************************
8997 REM *       Recordes         *
8998 REM **************************
```

```
8999 REM Ler os dados
9000 LET ns=5: DIM s(ns): DIM s$
(ns,10)
9010 CLOSE #4: OPEN #4;"m";1;"Re
cordes"
9020 FOR i=1 TO ns
9030 INPUT #4;s(i); LINE s$(i)
9040 NEXT i: CLOSE #4
9050 GO TO 9300
9098
9099 REM refazer o ficheiro se u
m recorde
9100 LET ns=5: IF sc(s(ns) THEN
RETURN
9110 PRINT ' FLASH 1;"       Um
novo recorde!!!       "
9120 INPUT "Introduzir o seu nom
e (max. 10 letras)"; LINE a$
9130 FOR i=1 TO ns
9140 IF sc(=s(i) THEN NEXT i
9150 ERASE "m";1;"recordes"
9155 OPEN #4;"m";1;"recordes"
9160 FOR j=1 TO ns
9170 IF j(i THEN PRINT #4;s(j)'s
$(j)
9180 IF j=i THEN PRINT #4;sc'a$
9190 IF j)i THEN PRINT #4;s(j-1)
's$(j-1)
9200 NEXT j
9210 CLOSE #4
9220 RETURN
9299 REM Inprimir quadro de pont
uacoes
9300 CLS
9310 PRINT FLASH 1; INK 7; PAPER
 0;TAB 8;"RECORDES";TAB 31;" "
9320 FOR i=1 TO ns
9330 PRINT PAPER i; INK 9,,TAB 6
;i;" ";s$(i);TAB 20;s(i),,,
9340 NEXT i
9350 RETURN
9988
9989 REM definir novo ficheiro
9990 OPEN #4;"m";1;"recorde"
9992 FOR i=1 TO 5
9994 PRINT #4;0'" "
9996 NEXT i: CLOSE #4
```

O programa armazena a tabela como dez elementos, pela ordem: 1.ª pontuação, 1.° nome, 2.ª pontuação, 2.° nome, etc. As linhas 9900- para cima só devem ser executadas à medida que o ficheiro recebe dados nulos. Para ler a tabela são usadas as linhas de 9000 a 9050. A linha 9000 estabelece o número de itens da tabela (em «ns»), e os quadros «s( )», para cada pontuação, e «s$( )», para cada nome. A linha 9010 abre o ficheiro para leitura, e as linhas 9020-9040 lêm cada pontuação e nome e colocam-nos nos quadros. O ficheiro é fechado, e a tabela é impressa a partir da li-

nha 9300-. Depois de o jogo terminar, a pontuação deve ser colocada na variável «sc», e é executado um GO SUB 9100. A linha 9100 verifica se a pontuação era menor do que a mais baixa da tabela. Nesse caso é então feito um RETURN, visto que o jogador não se classificou para entrar na tabela. Se o jogador se qualificou, então o seu nome é introduzido (linha 9120) e a sua nova posição na tabela é calculada nas linhas 9130-9140, e colocada em «i». A linha 9150 apaga o antigo ficheiro, e a linha 9155 cria um novo ficheiro de escrita. O ciclo de 9160 a 9200 imprime então no ficheiro a nova tabela, inserindo a nova pontuação e o nome, e mudando quaisquer dados inferiores uma posição para baixo. Finalmente a linha 9210 fecha o ficheiro.

CAPÍTULO 4

Unificha

O programa deste capítulo é um importante programa de base de dados, usando os microdrives e, opcionalmente, uma impressora (do tipo ZX ou RS232). Baseia-se muito de perto no programa «Unificha 1», do livro *O Spectrum Funcional*[1], cujo autor é David Lawrence, sendo aqui apresentado com a sua autorização.

Destina-se aos possuidores do 48K, se bem que mais adiante sejam dados pormenores de uma versão reduzida, para quem tem o 16K. O programa não foi renumerado e, assim, as linhas do programa original são as mesmas.

Listagem de «Unificha»

```
1000 PAPER 7: CLS: BORDER 7: INK
 6: PAPER 0: PRINT PAPER 2;"UNIF
ICHA";F$;TAB 31;" "
1010 PRINT '"FUNCOES DISPONIVEIS
;"
1020 PRINT '"    1) DEFINIR UM F
ICHEIRO"
1030 PRINT '"    2) INTRODUZIR I
```

[1] Publicado nesta colecção com o número 97.

```
NFORMACAO"
1040 PRINT '"    3) BUSCAR/MOSTR
AR/ALTERAR"
1050 PRINT '"    4) STOP"
1055 PRINT '"    5) SAVE/LOAD/CA
T"
1060 PRINT ''"      INTRODUZA A
SUA ORDEM"
1070 PAUSE 0: LET Z$=INKEY$
1080 CLS
1090 IF Z$="1" THEN GO SUB 1210
1100 IF Z$="2" THEN GO SUB 1440
1110 IF Z$="3" THEN GO SUB 2180
1120 IF Z$="4" THEN GO SUB 1150
1125 IF Z$="5" THEN GO TO 3500
1130 CLS
1140 GO TO 1000
1150 PRINT AT 10,5; INK 7; PAPER
 2;"SISTEMA DE FICHEIRO FECHADO"
1160 BEEP 2,2
1180 INPUT "Ja fez entrar as nov
as informacoes que quer guardar?
 (S/N)";Q$: IF Q$="N" THEN STOP
1190 SAVE "UNIFICHA": PRINT '"Re
bobine a cassete, prima qualquer
 tecla para verificar": PAUSE 0:
 VERIFY "UNIFICHA": STOP
```



```
1200 REM ************************
1210 REM * ESTRUTURA DE ENTRADA*
1220 REM ************************
1230 PRINT PAPER 2;"      ESTRUT
URA DO FICHEIRO     "
1235 GO SUB 4200
1240 PRINT '"QUANTOS ITENS EM CA
DA ENTRADA?"
1250 INPUT X
1260 CLS
1270 DIM A$(X,20)
1280 PRINT PAPER 2;"         NOM
E DOS ITENS         "
1290 FOR I=1 TO X
1300 PRINT "ITEM ";I;":";
1310 GO SUB 2780
1320 PRINT Q$(2 TO )
1330 LET A$(I)=Q$
1340 NEXT I
1350 DIM B$(28000)
1360 LET B$(1 TO 4)=CHR$ 2+CHR$
0+CHR$ 2+CHR$ 255
1370 DEF FN A()=256*CODE Y$(2*S-
1)+CODE Y$(2*S)
1380 DEF FN A$()=B$(C TO C+CODE
B$(C)-1)
1390 LET P=5
```

```
1400 LET Y$=CHR$ 0+CHR$ 1+CHR$ 0
+CHR$ 3
1410 LET N=2
1420 RETURN
1430 REM ************************
1440 REM *    SAIDA  NORMAL    *
1450 REM ************************
1460 LET R$=""
1470 PRINT PAPER 2;"           E
NTRADAS              "
1480 PRINT '"COMANDOS DISPONIVEI
S:"
1490 PRINT '">ENTRAR ITEM ESPECI
FICADO"''")""ZZZ"" PARA FIM"
1500 PRINT "**********************
**************"
1510 PRINT "TAMANHO DO FICHEIRO:
";P-1;"/";LEN B$
1520 FOR I=1 TO X
1530 GO SUB 2810
1540 GO SUB 2780
1580 PRINT Q$(2 TO )
1590 IF Q$(2 TO )="ZZZ" THEN  RE
TURN
1600 LET R$=R$+Q$
1610 NEXT I
1620 CLS
```



```
1630 GO SUB 1650
1640 GO TO 1440
1650 REM ************************
1660 REM * MEMORIZAR OS DADOS  *
1670 REM ************************
1680 IF P+LEN R$-1<LEN B$ THEN G
O TO 1730
1690 PRINT AT 14,10;"FICHEIRO CO
MPLETO"
1700 PRINT ''"Prima qualquer tec
la para  continuar"
1710 PAUSE 0
1720 RETURN
1730 LET POWER=INT (LN (N-1)/LN
2)
1740 LET S=2↑POWER
1750 LET T$=R$(2 TO CODE R$(1))
1760 FOR K=POWER-1 TO 0 STEP -1
1770 LET C=FN A()
1780 LET U$=FN A$()(2 TO )
1790 LET S=S+(2↑K)*(T$)U$)-(2↑K)
*(T$<U$)
1810 IF S>N-1 THEN LET S=N-1
1820 IF S<2 THEN LET S=2
1830 NEXT K
1840 LET C=FN A()
1850 LET U$=FN A$()(2 TO )
```

```
1860 IF T$<U$ THEN LET S=S-1
1870 LET B$(P TO P+LEN R$-1)=R$
1880 LET N=N+1
1890 LET Y$=Y$(1 TO 2*S)+CHR$ IN
T (P/256)+CHR$ (P-256*INT (P/256
))+Y$(2*(S+1)-1 TO )
1900 LET P=P+LEN R$
1910 RETURN
1920 REM ************************
1930 REM * MODIFICAR A ENTRADA *
1940 REM ************************
1950 LET S=S-1
1960 LET C=FN A()
1970 LET R$=""
1980 LET PRINT "ENTRADA";S-1;":-
"
1990 FOR I=1 TO X
2000 GO SUB 2810
2010 GO SUB 2830
2020 PRINT AT 17,0;PAPER 2;"
           EMENDA          "
2030 PRINT "COMANDOS DISPONIVEIS
:"
2040 PRINT ">""ENTER"" DEIXA COM
O ESTAVA"'">""ZZZ"" APAGA A ENTR
ADA"'">INTRODUZIR O NOVO ITEM"
2050 GO SUB 2780
```



```
2060 IF LEN Q$=1 THEN LET R$=R$+
B$ (C TO C+CODE B$(C)-1)
2070 LET C=C+CODE B$(C)
2080 CLS
2090 IF LEN Q$=1 THEN GO TO 2120
2100 IF Q$(2 TO )="ZZZ" THEN GO
TO 2130
2110 LET R$=R$+Q$
2120 NEXT I
2130 GO SUB 3130
2140 IF Q$(2 TO )="ZZZ" THEN RET
URN
2150 GO SUB 1660
2160 RETURN
2170 REM ************************
2180 REM *        BUSCA         *
2190 REM ************************
2200 LET S=2
2205 LET ST=2
2210 PRINT PAPER 2;"
 BUSCA              "
2220 PRINT ''"COMANDOS DISPONIVE
IS:"
2230 PRINT ">INTRODUZA O ITEM PA
RA BUSCA NORMAL"'">PRECEDER DE "
"SSS"" PARA"'"BUSCA ESPECIAL"'">
PRECEDER DE ""III"" PARA BUSCA"'
```

```
"PELA PRIMEIRA LETRA DA ENTRADA"
'">""ENTER"" PARA O PRIMEIRO ITE
M NO FICHEIRO"
2235 PRINT ">""PPP""SAIDA PARA "
'"IMPRESSORA " AND ST=2;"ECRAN"
AND ST=3
2240 PRINT "**********************
**********"
2250 PRINT ''"INTRODUZA ITEM A P
ROCURAR:";
2260 GO SUB 2780
2265 IF Q$=CHR$ 4+"PPP" THEN LET
 ST=5-ST: CLS : GO TO 2210
2270 PRINT Q$(2 TO )
2280 LET S$=Q$
2290 IF LEN S$=1 THEN GO TO 2510
2300 LET C=FN A()
2310 IF LEN S$(5 THEN GO TO 2430
2320 IF S$(2 TO 4)()"III" THEN G
O TO 2390
2330 FOR I=S TO N
2340 LET S=I
2350 LET C=FN A()
2360 IF B$(C+1)=S$(5) THEN GO TO
 2510
2370 NEXT I
2380 RETURN
```



```
2390 IF S$(2 TO 4)()"SSS" THEN G
O TO 2430
2400 GO SUB 2920
2410 IF C4=1 THEN GO TO 2510
2420 RETURN
2430 FOR I=1 TO X
2440 IF FN A$()=S$ THEN GO TO 25
10
2450 IF FN A$()=CHR$ 2+CHR$ 255
THEN RETURN
2460 LET C=C+CODE B$(C)
2470 NEXT I
2480 LET S=S+1
2490 LET C=FN A()
2500 GO TO 2430
2510 LET C=FN A()
2520 LET C4=0
2530 IF FN A$()=CHR$ 2+CHR$ 255
THEN RETURN
2540 CLS
2545 IF ST()2 THEN GO TO 4300
2550 PRINT "ENTRADA ";S-1;":-"
2560 GO SUB 2850
2570 LET S=S+1
2580 PRINT AT 15,0;PAPER 2;"
        BUSCA                "
2590 PRINT "COMANDOS DISPONIVEIS
```

```
;"
2600 PRINT ">""ENTER"" PARA VER
PROXIMO ITEM"'">""ZZZ"" PARA FIM
"'">""AAA"" PARA EMENDAR"'">""CC
C"" PARA CONTINUAR BUSCA"
2610 INPUT P$
2620 CLS
2630 IF P$="CCC" THEN GO TO 2300
2640 IF P$="" THEN GO TO 2510
2650 IF P$()"AAA" THEN GO TO 271
0
2660 LET C=FN A()
2670 CLS
2680 GO SUB 1930
2710 IF P$="ZZZ" THEN RETURN
2720 IF P$="AAA" THEN RETURN
2730 CLS
2740 GO TO 2260
2750 REM**************************
2760 REM*SUB-ROTINAS FUNCIONAIS*
2770 REM**************************
2780 INPUT Q$
2790 LET Q$=CHR$ (LEN Q$+1)+Q$
2800 RETURN
2810 PRINT A$(I,2 TO CODE A$(I,1
));":";
2820 RETURN
```



```
2830 PRINT FN A$()(2 TO )
2840 RETURN
2850 FOR I=1 TO X
2860 GO SUB 2810
2870 GO SUB 2830
2880 LET C=C+CODE B$(C)
2890 NEXT I
2900 RETURN
2910 REM ************************
2920 REM *   BUSCA ESPECIAL    *
2930 REM ************************
2940 LET C4=0
2950 FOR H=S TO N-1
2960 LET S=H
2970 LET C=FN A()
2980 LET C1=C
2990 FOR I=1 TO X
3000 LET C1=C1+CODE B$(C1)
3010 NEXT I
3020 FOR J=C+1 TO C1- LEN S$+5
3030 IF B$(J TO J+LEN S$-5)()S$(
5 TO ) THEN GO TO 3060
3040 LET C4=1
3050 RETURN
3060 NEXT J
3070 NEXT H
3080 LET C4=0
```

```
3090 RETURN
3100 REM ************************
3110 REM *FICHEIRO TELESCOPICO *
3120 REM ************************
3130 LET C=FN A()
3140 LET SHIFT=1000
3150 LET C1=C
3160 LET C3=C
3170 FOR I=1 TO X
3180 LET C1=C1+CODE B$(C1)
3190 NEXT I
3200 LET C2=C1-C
3210 FOR I=C1 TO LEN B$-1 STEP S
HIFT
3220 IF LEN B$-I+1(SHIFT THEN LE
T SHIFT=LEN B$-I+1
3230 LET S$=B$(I TO I+SHIFT-1)
3240 LET B$(C TO C+SHIFT-1)=S$
3250 LET C=C+SHIFT
3260 NEXT I
3270 LET Y$=Y$(1 TO 2*(S-1))+Y$(
2*(S+1)-1 TO )
3280 FOR I=1 TO N-1
3290 LET S=1
3300 LET C=FN A()
3310 IF C(=C3 THEN GO TO 3350
3320 LET C=C-C2
```



```
3330 LET Y$(2*I-1)=CHR$ INT (C/2
56)
3340 LET Y$(2*I)CHR$ (C-256*INT
(C/256))
3350 NEXT I
3360 LET P=P-C2
3370 LET N=N-1
3380 RETURN
3390 FOR I=1 TO 20: PRINT CODE Y
$(I): NEXT I
3500 CLS
3510 PRINT "UNIFICHA - OPERACOES
COM MICRODRIVE"
3520 PRINT '" 1) SAVE "; INVERSE
 1;F$
3530 PRINT '" 2) LOAD novos dado
s"
3540 PRINT '" 3) CATalogar um ca
rtucho"
3545 PRINT '" 4) RETURN para men
u principal"
3550 PAUSE 0: LET Z$=INKEY$
3560 IF Z$="1" THEN GO SUB 3900
3570 IF Z$="2" THEN GO TO 3800
3580 IF Z$="3" THEN GO SUB 3700
3590 IF Z$="4" THEN GO TO 1000
3600 GO TO 3500
```

```
3670 REM ************************
3680 REM *  ROTINA DE CATALOGO *
3690 REM ************************
3700 CLS: PRINT "CATALOGO DO CAR
TUCHO:"
3710 INPUT "NUMERO DO DRIVE? (0
para saida)";D
3720 IF D=0 OR D>8 THEN RETURN
3730 CAT D
3740 PRINT #0;AT 0,0;FLASH 1;"Pr
ima uma tecla para continuar";
3750 PAUSE 0
3760 RETURN
3770 REM ************************
3780 REM *    Rotina de LOAD    *
3790 REM ************************
3800 PRINT '' FLASH 1;"ATENCAO:
QUANDO LER DADOS NOVOS APAGA OS
ANTERIORES NA MEMORIA"
3805 PRINT '"Premir C para conti
nuar, ou qualquer outra tecla pa
ra sair."
3810 PAUSE 0: LET Z$=INKEY$: IF
Z$<>"C" AND Z$<>"c" THEN GO TO 3
500
3820 GO SUB 4200
3830 INPUT "NUMERO DO DRIVE (1-8
)?";D
3840 IF D<0 OR D>8 THEN GO TO 38
30
3850 LOAD *"M";D;F$+CHR$ 128
3860 GO TO 1000
3870 REM ************************
3880 REM *   Rotina de SAVE     *
3890 REM ************************
3900 PRINT ''"(NOME EXISTENTE=";
F$;")"
3905 GO SUB 4200
3910 INPUT "NUMERO DO DRIVE? ";D
3920 IF D<0 OR D>8 THEN RETURN
3930 CLS
3940 PRINT "UNIFICHA-";F$
3950 PRINT AT 15,0;"Quer ";F$;"
no drive ";D;" apagado? (S/N)"
3960 INPUT LINE Z$
3965 PRINT AT 15,0;,,,,
3970 IF Z$<>"S" AND Z$<>"s" THEN
 GO TO 4000
3980 ERASE "M";D;F$+CHR$ 128
3990 PRINT '"FICHEIRO APAGADO"
4000 SAVE *"M";D;F$+CHR$ 128 LIN
E 1000
4010 PRINT "DADOS GRAVADOS - PRO
CEDENDO A   VERIFICACAO"
```

```
4020 VERIFY a"M";D;F$+CHR$ 128 L
INE 1000
4030 RETURN
4170 REM**************************
4180 REM*Alterar nome ficheiro *
4190 REM**************************
4200 INPUT "NOME DO FICHEIRO? ";
LINE F$
4210 IF LEN F$<10 AND LEN F$>0 T
HEN RETURN
4220 INPUT "(MAX. 9 CARACTERES)
"; LINE F$
4230 GO TO 4210
4270 REM**************************
4280 REM* Saida pela impressora*
4290 REM**************************
4300 LPRINT "ENTRADA ";S-1;":-"
4310 FOR I=1 TO X
4320 LPRINT A$(I,2 TO CODE A$(I,
1));";";
4330 LPRINT FN A$()(2 TO )
4340 LET C=C+CODE B$(C)
4350 NEXT I
4360 GO TO 2570
```

Quando o tiver introduzido, escreva o seguinte:

```
CLEAR
LET F$=""
```


e então salve o programa com LINE 1000. Para o fazer arrancar, faça GO TO 1 — nunca use RUN, pois limparia os dados.

Cada ficheiro de dados pode contar até 28 000 caracteres compostos de um número de itens, definidos quando o ficheiro é inicializado. Um exemplo seria um ficheiro do livro de moradas, com três itens por entrada — nome, endereço e número de telefone.

A opção 1 possibilita-lhe criar um novo ficheiro, e também lhe solicita que dê um nome ao seu ficheiro.

A opção 2 permite-lhe introduzir informações no ficheiro.

A opção 3 é de todas a mais útil: permite-lhe fazer a procura em todo o ficheiro, imprimindo as entradas mais apropriadas no écran ou na impressora. A opção «busca especial» pesquisará em todas as partes de cada entrada a ocorrência duma cadeia especificada, e não é muito rápida. Se, por exemplo, você fizer entrar *SSSKENT* com o ficheiro do livro de moradas de todos os que vivem no Kent, ou cujo nome é Kent, todos seriam impressos. Uma opção mais rápida é a busca normal, a qual só verifica o primeiro carácter de cada entrada.

A opção 4 permite que o programa e os dados sejam guardados em cassete para fins de apoio.

A opção 5 permite que os dados correntes sejam guardados, ou outros dados sejam carregados, ou ainda, que um cartucho seja catalogado. Os ficheiros são armazenados em cartucho com um CHR$ 128 que se lhes acrescenta no fim, para os distinguir dos ficheiros normais de programas. O carácter extra não é notado ao executar a instrução CAT(álogo), pois que ele imprime como um espaço.

Não vou tentar explicar o funcionamento deste programa — David Lawrence fá-lo no seu livro. Explicarei no entanto a sub-rotina da impressora de modo a que os utilizadores a possam alterar para se ajustar aos requisitos dos seus ficheiros. A rotina fica situada entre 4300 e 4350 — «S-1» é a posição de entrada no ficheiro e «X» é o número de itens por entrada. A linha 4320 imprime o título de cada entrada e a linha 4330 imprime os dados existentes. A linha 4320 incremento um apontador. Se, por exemplo, você tiver um ficheiro construído como um livro de moradas, como vimos acima, e quiser imprimir as entradas num formato de etiqueta de morada, uma possibilidade podia ser:

```
4300 LPRINT "Ref. no. ";S-1
4310 LPRINT FN A$()(2 TO ): REM
```

```
o nome
4320 LET C=C+CODE B$(C)
4330 LPRINT FN A$()(2 TO ): REM
o endereco
4340 LPRINT
4350 LET C=C+CODE B$(C)
4360 LPRINT "Telefone:";FN A$()(
2 TO ): REM o numero
4370 LET C=C+CODE B$(C)
4380 GO TO 2570
```

Aumentos adicionais

Existem ainda mais características que podiam ser acrescentadas. Uma poderia ser um CAT de somente programas «Unificha», usando a rotina «Stream 14-z$» e examinando «Z$» para CHR$ 128. Um outro melhoramento seria converter as linhas 3130-3380 em código-máquina, o que aumentaria grandemente a velocidade de apagar e emendar.

Versão 16K

No 16K Spectrum pode ser usada uma versão limitada do «Unificha», com as seguintes modificações:

Anular todas as instruções REM e as linhas 3540, 3580 e 3700-3760

Emendar as linhas:

```
1350 DIM B$(1000)
3140 LET SHIFT=100
3375 LET S$=""
```


CAPÍTULO 5

Protecção de programas

Quando tiver escrito um muito bom pedaço de software e o desejar vender, não quererá que o António, o Luís ou o Eduardo possam fazer tantas cópias quantas quiserem da sua obra-prima para dar a todos os seus amigos. Se bem que nenhum programa possa possivelmente ser cem por cento incopiável, podem ser usadas muitas técnicas simples para dificultar o mais possível a sua cópia. Há algumas características incorporadas que iremos discutir, bem como alguns outros métodos mais dissimulados. Deve deixar-se bem claro que nenhum método é imbatível — um programador de código-máquina, com determinação, será sempre capaz de derrotar os métodos de protecção de qualquer programa desde que disponha de tempo e de conhecimentos.

AUTO-EXECUÇÃO

A maneira mais simples de tornar os programas difíceis de copiar é fazê-los de execução automática, usando a função SAVE* ...LINE. Um programa assim não pode ser fundido com outro: quando carregado, arrancará automaticamente. Se der a um programa o nome «run» (todo em minúsculas), então depois da ligação à corrente ou de um NEW, escrevendo RUN carregará o programa e ele arrancará (se estiver guardado com LINE).

É também muito útil ter os nomes de programas começados por CHR$ 0, pois assim não serão mostrados no catálogo do cartucho. Se você fizer o programa «run» carregar o programa «invisível», o potencial copiador do software nem sequer fica a saber o

nome do programa principal. Uma outra maneira é fazer o nome do programa todo em espaços em branco — a maior parte dos utilizadores nem sequer o notará no catálogo.

POKEs que fazem bloquear o computador

Uma coisa que deve ser evitada é que o utilizador possa fazer parar um programa uma vez que ele arrancou, ou no meio de uma instrução LOAD. No caso acima ser-lhe-ia possível ver o nome do ficheiro invisível, ao examinar a listagem. Uma maneira de o conseguir, que não é muito subtil mas funciona lindamente, é fazer a primeira linha:

```
POKE 23659,0
```

O que isto faz é dizer ao Spectrum que não há linhas na secção inferior do écran. Então, quando se verifica qualquer erro [tal como *Break into programa* (interromper o programa)] todo o sistema ficará «pendurado». porque não há lugar algum onde imprimir a mensagem de erro. O utilizador terá então de desligar e voltar a ligar o computador, impedindo-o, por conseguinte, de examinar a listagem ou copiá-la.

Deve ter cuidado quando usar este POKE particular: um comando CLS, ou qualquer tentativa para imprimir nas duas linhas do fundo (por exemplo, usando INPUT) fará bloquear o computador. O POKE, na realidade, é útil apenas para programas em código-máquina enquanto se estão a carregar — para programas em BASIC ele é realmente inadequado, pois que é demasiado limitado. Uma alternativa é usar a linha:

```
LET p=PEEK 23613+256*PEEK 23614:
 POKE p,0: POKE p+1,0
```

a qual fará bloquear o sistema quando ocorrer qualquer erro, mas permite a impressão na parte inferior do écran.

On Error GOTO

Muitos outros computadores têm uma função *On Error GOTO,* que força um salto para uma certa linha quando ocorre um erro. Infelizmente o Spectrum carece de tal função, e assim a seguinte rotina em código-máquina foi escrita para o mesmo fim.

Listagem «On error»

```
8794 REM ************************
8795 REM *   ON ERROR GO TO     *
8796 REM ************************
8797 REM err=endereco de inicio,
linha=salto do erro
8798 REM recomendado: 64780/3201
0
8800 RESTORE 8900: LET c=0
8810 FOR i=err TO err+170
8820 READ a: LET c=c+a
8830 IF a<256 THEN POKE i,a
8840 NEXT i
8850 IF c<>17439 THEN PRINT "Err
o de verificacao da soma": STOP
8860 POKE err+149,linha-256*INT
(linha/256)
8870 POKE err+150,INT (linha/256
)
8875 RETURN
```

```
8880 RANDOMIZE USR err
8885 POKE 23734,0: RETURN
8890 RANDOMIZE USR err
8895 POKE 23734,4: RETURN
8900 DATA 33,17,0,9,235,42,61,92
8905 DATA 115,35,114,207,49,1,0,
0
8910 DATA 201,34,201,92,42,61,92
,17
8915 DATA 3,19,213,205,176,22,25
3,203
8920 DATA 55,174,205,110,13,33,1
85,23
8925 DATA 34,237,92,58,58,92,50,
211
8930 DATA 92,245,207,50,33,129,0
,237
8935 DATA 91,201,92,241,167,237,
82,32
8940 DATA 20,253,203,124,70,32,1
4,254
8945 DATA 7,40,10,62,100,50,211,
92
8950 DATA 62,63,215,24,21,60,71,
254
8955 DATA 10,56,2,198,7,205,239,
21
```



```
8960 DATA 62,32,215,120,17,145,1
9,205
8965 DATA 10,12,175,17,54,21,205
,10
8970 DATA 12,237,75,69,92,237,67
,201
8975 DATA 92,205,27,26,62,58,215
,253
8980 DATA 78,13,6,0,205,27,26,33
8985 DATA 59,92,203,174,251,203,
110,40
8990 DATA 252,33,66,92,17,260,27
0,115
8994 DATA 35,114,35,54,1,253,54,
0
8995 DATA 255,253,54,124,0,205,1
10,13
8996 DATA 195,125,27
```

Tem um comprimento de 171 bytes, e é de posição independente. Antes do GO SUB 8800 para o tornar operacional, a variável «err» deve ser enviada para a necessária posição de memória, e «line» para o número da linha de erro. Para operações normais (não-interface), faça um GO SUB 8800 para activar a rotina. Quando o erro ocorre, a correspondente mensagem será impressa no local normal. Esperará então que seja premida uma tecla, antes de limpar a mensagem e saltar para a linha correspondente. A característica é desactivada quando se verifica o erro. O número da linha onde o erro ocorreu pode ser lido por:

```
PEEK 23753+256*PEEK 23754
```

e o número do erro com:

```
PEEK 23763
```

Para operações com interface, deve fazer um GO SUB 8890 para activar a rotina. Note que, se ocorrer um erro de interface, a correspondente mensagem não será produzida — em seu lugar será impresso um ponto de interrogação, e a posição 23763 conterá 100.

Depois de um comando da interface, a rotina é desarmada como resultado da sua operação.

Se desejar desactivar a rotina, faça:

```
LET p=PEEK 23613+256*PEEK 23614:
 POKE p,3: POKE p+1,19: POKE 237
34,0
```

Esta rotina não é compatível com a rotina «ON EOF GOTO» do capítulo precedente. Para verificar para EOF quando ocorre um erro, PEEK 23763 será 7 se é um EOF.

Enquanto se estiver a fazer a depuração de erros, prestar atenção aos ciclos infinitos. Se você tivesse a linha 1000 como linha de erro, e era

```
1000 GO SUB 8880
```

você podia nunca ser capaz de interromper o programa, de modo que deve sempre incluir em qualquer parte do seu programa a opção de fazer POKE para o desactivar.

Para a inclusão desta rotina num programa comercial, o melhor é acrescentar a seguinte linha:

```
8855 POKE err+23,94: POKE err+24
,35: POKE err+25,86
```

Estes POKEs alteram a rotina, de modo que ela não é cancelada quando ocorre um erro.

Se você quer incluir esta ou qualquer outra rotina em código-máquina deste livro num programa comercial, *deve* ser pedida a minha autorização antes de o fazer, dado que sobre elas tenho direitos de autor.

Como se viu, o código-máquina e alguns POKEs podem ser usados com muita eficácia para proteger programas, mas a inversa é também verdadeira: o código-máquina pode também ser usado para quebrar as defesas de programas, desfazendo os métodos de protecção. Para quebrar um particular método, o programador deve saber o que foi usado. É por isso que é relativamente simples para um programador em código-máquina possuído de determinação conseguir, por exemplo, um catálogo de ficheiros invisíveis, ou fundir programas de arranque automático. É um desafortunado subproduto da abundante disponibilidade de dados técnicos em todos os produtos Sinclair.

CAPÍTULO 6

Utilização da interface RS232

Uma outra característica da interface 1 é a facilidade de transmitir e de receber dados via RS232. A RS232 é um (quase) padrão universal de transmissão em série, e há um muito grande número de periféricos, bem como a maior parte dos outros computadores, que podem comunicar por meio da RS232. Para o uso com o Spectrum, a aplicação mais vulgar é fazer accionar impressoras de papel vulgares e largas, para produzir listagens e bem elaboradas saídas de programas.

A velocidade a que os dados são enviados ou recebidos é conhecida como taxa *baud* (ou baudagem), e é aproximadamente dez vezes o número de bytes enviados por segundo. Existem nove taxas baud padrão, e o Spectrum suporta-as todas, nomeadamente 50, 110, 300, 600, 1200, 2400, 4800, 9600 e 19200. Os teletipos usam normalmente as taxas lentas de 110 ou 300 baud, com terminais CRT (tubos de raios catódicos) funcionando a altas velocidades de 2400 baud e mais.

Tal como necessitam de taxas baud diferentes, os diversos dispositivos podem precisar de formatos de dados ligeiramente diferentes. No caso do Spectrum, o formato é fixado em:

— Não paridade
— Bits de oito dados
— Um bit de separação

Para utilizar a facilidade RS232 com o Spectrum são usados mais dois especificadores da canais — «t» e «b». Estas streams são diferentes, no seu funcionamento, pela maneira como tratam cada carácter.

Canal de texto «t»

O canal de texto é o mais adequado para accionamento de impressoras, visto que faz certas coisas para cada carácter. Com referência ao conjunto de caracteres no Anexo A do manual do Spectrum, o canal «t» imprime o seguinte com cada carácter de código:

0-12	(códigos de controle) são ignorados
13	(nova linha)
14-31	(códigos de controle) são ignorados
32-127	(códigos ASCII) são enviados não modificados
128-164	(códigos de gráficos) são enviados como «?»
165-255	(símbolos) são dessimbolizados em caracteres individuais

Esta conversão de carácter é ideal para enviar listagens de programa, bem como saídas de programa, mas muitas impressoras usam códigos abaixo de 32 para realizar funções especiais — por exemplo, as séries Epson usam CHR$14 para imprimir texto ampliado, mas um tal carácter não pode ser enviado por um canal «t». Acresce que a extremamente útil função TAB é ignorada, o que torna muito mais difícil a saída dum bem apresentado programa. Um método alternativo é usar o canal «b».

Canal binário «b»

O canal binário é muito adequado para comunicar com dispositivos que não sejam as impressoras, porque não é feita conversão de carácter. Cada carácter é enviado sem ser modificado, o que é ideal para se enviar dados em bruto e programas. É também necessário para enviar códigos de controle para as impressoras, visto que eles não podem ser enviados por um canal «t».

Antes de que o seu Spectrum possa enviar ou receber quaisquer dados via RS232, ele deve saber a taxa baud do seu periférico. Isto é feito pela instrução FORMAT «b»;, seguida pela taxa baud. Por exemplo, para se usar a taxa baud de 110, use:

```
FORMAT "b";110
```


(Podia usar também FORMAT «t»; 110 — produz o mesmo efeito.)

Se entrar com uma taxa baud não reconhecida não ocorrerá uma mensagem de erro, e o Spectrum usará a taxa reconhecida imediatamente acima. Para verificar a taxa correntemente seleccionada use:

```
PRINT INT (3500000/((PEEK 23747+
256*PEEK 23748+2)*26))
```

(Para velocidades mais elevadas, isto pode ser incorrecto em alguns baud.)

Se a taxa baud não está especificada, o sistema usualmente não o notará, e podem resultar estranhas velocidades de transferência. Inicialmente a taxa baud é estabelecida para a taxa padrão de 9600.

Accionamento de uma impressora RS232

Antes de pela primeira vez se estabelecer a ligação da impressora RS232 com o Spectrum, deve ser estabelecido o formato de dados que o Spectrum usa. Isto pode ser normalmente realizado mudando alguns interruptores miniaturais no interior da impressora, e deve somente ser feito uma vez. Ele deve ser estabelecido para a mais alta taxa baud permitida (e tome nota dela, para o caso de se esquecer!), e as opções para o formato dos dados devem ser não paridade, 8 dados e um bit de separação. Se ele oferece a opção *line-feed* (avanço de linha) automático após *carriage return* (fim de linha ou o equivalente a «retrocesso do carro» nas máquinas de escrever) (em calão, auto CR depois de LF), a opção deve ser desactividada, visto que o Spectrum pode criar as *line-feeds* necessárias.

Quando tiverem sido escolhidas todas as opções requeridas, a impressora pode ser ligada à interface por meio de um cabo adequado que se insere na tomada com 9 pinos, do tipo D.

Quando se utilizar uma impressora RS232, é aconselhável ter ambos os canais «t» e «b» abertos simultaneamente — o último para texto e listagens e o primeiro para códigos de controle. É me-

lhor usar a stream 3 para os canais «t» da impressora, utilizar as instruções LLIST e LPRINT nos programas existentes, por exemplo:

```
  10 FORMAT "t";1200: REM taxa b
aud
  20 OPEN #3;"t": REM texto na s
tream 3
  30 OPEN #15;"b": REM binario n
a stream 15
  40 LLIST: REM listagem no cana
l "t"
  50 LPRINT "Tamanho normal"
  60 PRINT #15;CHR$ 14; #3; Larg
ura dupla
70 CLOSE #3: CLOSE #15
```

No exemplo acima, a stream 15 é usada como um ficheiro «b» para enviar códigos de controle, e a stream 3 é usada como um canal «t» para x texto e listagens.

Quando transmitindo pela RS232, a margem do écran torna-se preta se o disposivo de recepção (neste caso a impressora) está ocupado e não pode aceitar os dados. O Spectrum esperará até que ele esteja pronto antes de enviar os dados, ou até que seja premida a tecla BREAK. Quando qualquer stream RS232 está fechada é transmitido um carácter 13, para esvaziar os buffers da impressora ou outros. Quando o comando CLEAR # é usado não é enviado um carácter 13.

Não é possível ter mais de um dispositivo RS232 ligado de cada vez, se bem que diferentes streams possam ligar ao mesmo canal. Também não é possível ter diferentes taxas baud em diferentes canais ou streams.

O único problema importante quando se accionam impressoras via RS232 é a falta do comando TAB. A rotina em código-máquina que se segue abrirá a stream 3 como um canal «t», mas com uma ligeira modificação. Ela contará o número de caracteres enviados, e por conseguinte permite que a função TAB seja plenamente implementada. E perguntamos a nós próprios por que é que este pequeno melhoramento não foi incluído na máquina original. Ela corrige também a falha no software que incorrectamente produz espaços duplos nas listagens (por exemplo THEN PRINT). O código está escrito para ficar na área de memória do buffer da impressora ZX, que nunca é usada quando a stream 3 é redefinida. Note-se que nem um CLOSE # 3 ou um CLEAR # fará a stream 3 reverter para a impressora ZX, e não enviará um *carriage return*.

Rotina TAB

```
8997 REM **********************
8998 REM *    TAB   RS232      *
8999 REM **********************
9000 RESTORE 9100: LET c=0
9010 FOR i=23296 TO 23467
9020 READ a: LET c=c+a
9030 POKE i,a
9040 NEXT i
9050 POKE 23540,80: REM largura
da impressora
9055 IF c<>19420 THEN PRINT "Err
o de verificacao da soma": STOP
9060 RANDOMIZE USR 23296
9070 RETURN
9100 DATA 42,79,92,1,15,0,9,17
9110 DATA 23,91,115,35,114,1,0,0
```

```
9120 DATA 33,245,91,112,35,112,2
01,254
9130 DATA 32,48,93,254,13,32,26,
33
9140 DATA 246,91,203,70,203,134,
192,33
9145 DATA 246,91,203,134,43,54,0
,62
9150 DATA 13,205,169,91,62,10,19
5,169
9160 DATA 91,254,23,63,192,17,71
,91
9170 DATA 42,81,92,115,35,114,20
1,50
9180 DATA 15,92,17,79,91,24,241,
17
9190 DATA 23,91,205,64,91,58,15,
92
9200 DATA 87,33,244,91,150,210,1
08,4
9210 DATA 35,122,150,213,220,39,
91,209
9220 DATA 122,253,150,187,200,71
,62,32
9230 DATA 197,217,215,217,193,16
,247,201
9240 DATA 254,165,56,5,214,165,1
```



```
95,16
9250 DATA 12,253,203,188,134,33,
59,92
9260 DATA 203,134,254,32,32,2,20
3,198
9270 DATA 254,128,56,2,62,63,205
,169
9280 DATA 91,33,245,91,52,126,43
,190
9290 DATA 192,205,39,91,253,203,
188,198
9300 DATA 201,207,30,201
```

Cópias do écran

Uma outra possibilidade útil numa impressora é a produção de cópias em papel do que está no écran. Na sua forma mais simples isto consiste no pequeno programa que se segue, que trabalha com qualquer impressora que tenha pelo menos 32 caracteres por linha), o qual lê cada carácter e o envia depois para a impressora.

```
 100 FORMAT "T";BAUD: REM valor
adequado
 110 OPEN #4;"T"
 120 FOR Y=0 TO 21
 130 FOR X=0 TO 31
 140 LET A$=SCREEN$(Y,X)
 150 IF A$="" THEN LET A$=""
```

```
160 PRINT #4;A$;
170 NEXT X
180 PRINT #4
190 NEXT Y
```

O programa funciona lendo cada posição de carácter utilizando SCREEN$, e ou o imprime ou deixa um espaço se ele não é reconhecível (linha 150). Depois de cada linha de 32 caracteres, é enviada uma nova linha (linha 180). Se bem que isto seja um pouco rudimentar, pode ainda ser uma sub-rotina muito útil, em particular porque é independente da impressora.

Como pode você fazer, se quiser, um desenho de alta resolução, ou uma listagem com caracteres gráficos e invertidos? Se tiver uma impressora adequada pode fazer verdadeiras cópias deles, como com o comando COPY na impressora ZX. Em BASIC é, contudo, um processo lento, mas os resultados valem a pena. Apresento aqui duas versões — uma para a gama de impressoras Epson (com uma placa RS232 montada), e uma para a Seikosha GP100.

Cópia de alta resolução Epson

```
1000 FORMAT "b";1200: REM mudar
para se adaptar ao interface
1010 OPEN #3;"b"
1020 LPRINT CHR$ 27;"A";CHR$ 8;
1030 FOR y=175 TO 0 STEP -8
1040 LPRINT CHR$ 27;"K";CHR$ 0;C
HR$ 1;
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (128*POINT (x,y
)+64*POINT (x,y-1)+32*POINT (x,y
-2)+16*POINT (x,y-3)+8*POINT (x,
```



```
y-4)+4*POINT (x,y-5)+2*POINT (x,
y-6)+POINT (x,y-7));
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 27;"A";CHR$ 12
```

Cópia de alta resolução Seikosha

```
1000 FORMAT "b";1200: REM mudar
para se adaptar ao interface
1010 OPEN #3;"b"
1020 LPRINT CHR$ 18
1030 FOR y=174 to 0 STEP -7
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (POINT (x,y)+2*
POINT (x,y-1)+4*POINT (x,y-2)+8*
POINT (x,y-3)+16*POINT (x,y-4)+3
2*POINT (x,y-5)+64*POINT (x,y-6)
+128);
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 30
```

Os programas funcionam usando a função POINT para calcular um número binário que é enviado para a impressora. Note-se que é usado um canal «b», porque não deve ocorrer conversão de

carácter. A figura 4 mostra um exemplo da saída duma rotina Epson.

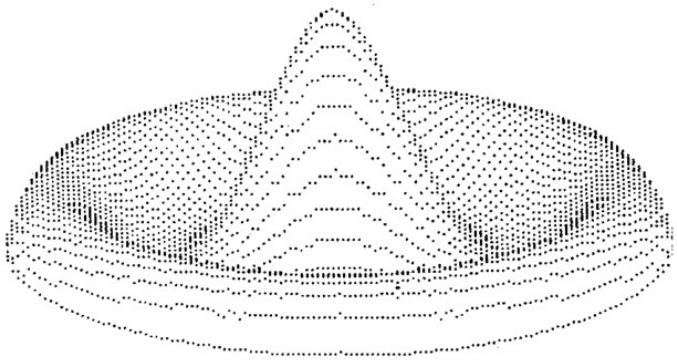


Fig. 4 — *Amostra duma cópia Epson*

Outros usos para o RS232

O canal «b» pode também ser usado para enviar programas, dados, código máquina e écrans, de maneira semelhante aos canais «m», mas não são necessários nomes de ficheiros. Isto pode ser muito útil para o envio de programas pelo telefone, por exemplo, se se comprar um modem com RS232. Com o adequado hardware, para enviar para um amigo na outra extremidade da linha um programa via um modem de 300 baud você deve primeiro escrever

```
FORMAT "b";300:LOAD *"b"
```

de modo que o seu Spectrum fique preparado para o esperar. Depois deve escrever no teclado

```
FORMAT "b";300:SAVE *"b"
```

O canal «t» só deve ser usado para transmitir listagens ou textos. Podem ser usados com o RS232 todos os comandos LOAD*, SAVE*, MERGE*, e VERIFY*, e podem também usar-se OPEN# e CLOSE#. Nomes de ficheiros e os números seguintes não são precisos com os canais da RS232. Se bem que sejam aceites pelo Spectrum, serão ignorados. Por exemplo, SAVE#*«b»;5;«nome» é aceite, mas os pormenores extra são ignorados.

Uma outra aplicação pode ser utilizar o Spectrum como um terminal RS232, para transmitir caracteres escritos para um outro computador:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 PAUSE 0: LET a$=INKEY$
40 IF a$)CHR$ 127 THEN GO TO 30
50 IF a$=CHR$ 6 THEN POKE 23658
8-PEEK 23658: GO TO 30
60 PRINT #4;a$
70 GO TO 30
```

O programa funciona lendo o teclado (linha 30), ignorando símbolos (por exemplo STOP) não padronizados (40), e usando CAPS LOCK para mudar entre os modos C e L (linha 50). Se for válido, informa disso, pela stream 4, o computador recebedor. Alguns computadores podem reagir de modo estranho a alguns dos códigos do Spectrum, tal como INV VIDEO; assim, pode ser necessário fazer testes adicionais do lado do Spectrum.

Uma aplicação adicional podia ser usar o Spectrum como um terminal de visualização, imprimindo nele toda a entrada de RS232:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 LET a$=INKEY$#4
40 IF a$="" THEN GO TO 30
50 IF a$=CHR$ 12 THEN CLS: GO T
O 30
```

```
60 PRINT a$;
70 GO TO 30
```

Quando se usar o RS232 para entrada, basta premir só a tecla SPACE para fazer *Break* (parar) — não é necessário CAPS SHIFT.

É possível ligar dois Spectrums directamente via RS232 por meio dum fio adequado, mas poucas vantagens há quando se comparar com a formação da rede, que é muito mais fácil de realizar, visto que faz mais rapidamente uma coisa semelhante enquanto deixa livre para os periféricos a muito útil porta *(port)* RS232.

Acrescentos à «Listagem de Stream» para o RS232

Devem ser acrescentadas as linhas seguintes ao programa «Listagem de Stream» para lhe permitir manusear os canais RS232:

Listagem «Stream 3»

```
1640 IF F$="T" THEN GO TO 3000
2999 REM Canal T
3000 PRINT INVERSE 1;"RS232  ";
3010 IF FN P(D+5)=3132 THEN PRIN
T "Texto": GO TO 3040
3020 IF FN P(D+5)=3162 THEN PRIN
T "Binario": GO TO 3040
3030 PRINT "(Desconhecido)"
3040 GO SUB 1800
3050 PRINT "Taxa baud:";INT (350
0000/((FN P(23747)+2)*26));" (ap
```



```
rox.)"
3060 GO TO 1500
```

Ambos os tipos de canais RS232 têm o mesmo especificador de canal em memória — «T». Deste modo, as linhas 3010 e 3020 funcionam quer seja Texto quer Binário. A taxa baud é também impressa.

CAPÍTULO 7

Utilização da rede

«Formação da rede» é um método pelo qual muitos computadores podem estar ligados de modo que comunicam rapidamente entre si. No caso do Spectrum, esta velocidade é para cima de 3 K bytes por segundo, a qual é muito maior do que a mais elevada taxa *baud* do RS232, 19200. Podem estar ligados uns aos outros até 64 Spectrums, e há vários usos para esta facilidade — o primeiro é nas salas de aula, onde se pode ter um certo número de Spectrums ligados uns aos outros de modo que eles podem «partilhar» os periféricos, como as impressores e os microdrives. Um outro uso é fazer a ligação entre os computadores de dois amigos, de modo que eles podem rapidamente trocar programas e dados. Uma utilização adicional é para o desenvolvimento de programas, particularmente código-máquina. Um Spectrum pode conter o Assembler e programas utilitários e descarregar o desenvolvimento do programa para um segundo Spectrum. A verdade é que se o segundo Spectrum bloqueia devido a um erro, pode rapidamente ter uma versão modificada recarregada do primeiro.

Para usar a rede são utilizados canais e streams, com o especificador de canal «N» para *Network* (Rede). Antes de usar a rede, tem de se decidir sobre um número para o seu Spectrum privativo (ou *estação*). Isto estabelece-se com o comando:

```
FORMAT "n";t
```

onde «t» é o número que se deseja, de 1 a 64. Quando o Spectrum está ligado, «t» é estabelecido para o número 1. Se há somente dois Spectrums na rede, não é preciso um FORMAT em cada um, pois

ambos podem ser a estação número 1. Se você se esquecer qual é o número da sua estação, a função:

```
PEEK 23749
```

dar-lhe-á o seu número.

Para ligar as estações entre si, use os cabos fornecidos com cada interface e introduza-os nas tomadas que ele tem, por debaixo das tomadas para cassete, formando uma cadeia. Deve sempre haver um fio deixado de fora, pois você não deve formar um ciclo fechado — a figura 5 mostra os pormenores das ligações para três estações.

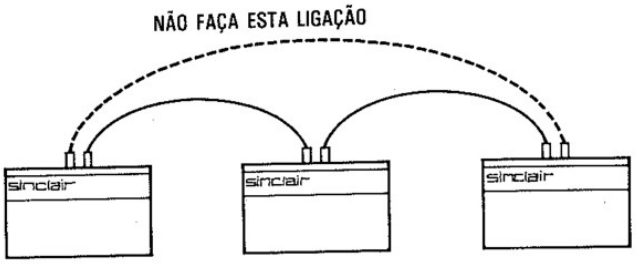


Fig. 5 — *Um Exemplo de rede*

Enviando programas através da rede

Tal como com os microdrives e o RS232, os programas e dados podem ser enviados por meio da rede usando comandos semelhantes aos usados pelo RS232, mas com o especificador «N», seguido do número da estação. Se, por exemplo, você está na estação 10 e um amigo que queria o seu programa está na estação 20, você deve teclar:

```
SAVE *"n";20
```


e a margem do seu écran ficará preta enquanto o seu computador espera que a estação 20 do seu amigo aceite o programa. O seu amigo deve então teclar:

```
LOAD *"n";10
```

para aceitar o programa. Quando ele fizer isto, as margens dos écrans das duas estações irão piscar enquanto o programa estiver a ser transferido. Se ele tivesse teclado LOAD primeiro, o seu Spectrum teria esperado que você teclasse SAVE e o programa teria sido transferido perfeitamente. Por razões que veremos mais adiante, é uma boa prática para aquele que vai receber bater o LOAD antes que o que vai enviar tenha batido o SAVE.

Se o seu amigo deseja confirmar que o programa foi transferido correctamente, deve escrever:

```
VERIFY *"n";10
```

e você deve escrever:

```
SAVE *"n";20
```

outra vez. Todas as funções usuais podem ser acrescentadas ao comando, por exemplo:

```
SAVE *"n";10 LINE 10
LOAD *"n";20 SCREEN$
```

O envio de dados através da rede

Para se enviar dados, através da rede, para qualquer outra estação, são usadas streams e o canal «N». Para estabelecer uma stream é usado o comando:

```
OPEN #s;"n";x
```

em que «s» é o número da stream (0 a 15) e «x» é o número da outra estação, de 0 a 64 (0 tem uma função especial, que veremos mais adiante). Isto cria um buffer semelhante ao usado com os canais do tipo «M», mas com cerca de metade do tamanho — 255 caracteres, para ser exacto. Quando uma stream de rede é aberta, ela

pode ser quer um ficheiro de leitura, quer um ficheiro de escrita, mas não ambos. Você determina o que ela é pela primeira acção que é feita com ela — se PRINT # nela, torna-se um ficheiro de escrita, ou se lê a partir dele (usando INPUT # ou INKEY$ #) torna-se um ficheiro de leitura. Tal como com os microdrives, os comandos CLOSE # são vitalmente importantes com os ficheiros de escrita — se você não usar a instrução CLOSE #, os últimos dados no buffer não serão enviados e, o que ainda é mais importante, a estação recebedora ficará para sempre à espera deles!

Enquanto estão a ser efectuadas quaisquer comunicações pela rede (isto é, enquanto a margem está a piscar), a tecla CAPS não é necessária para Break — será suficiente premir SPACE. Nunca se deve ligar ou desligar um Spectrum quando se estão a fazer comunicações.

Estação 0 — Emissão

Quando se está a receber ou enviar dados para ou de outra estação, realiza-se normalmente o *«handshaking» (o que poderia traduzir-se como «cumprimento entre computadores »)* — isto significa que se a estação recebedora não está pronta para o fazer, a que envia os dados espera até que tal aconteça, e os dados não se perderão. Contudo, foi incluída na interface uma facilidade muito útil — a de *broadcasting* (emissão), usando o número de estação zero, que não obedece à regra indicada.

Quando você enviar dados para a «estação 0», eles são enviados para todos os Spectrums ligados à rede, mas não se espera até qualquer deles estar pronto. Se eles estão à espera de entrada vinda da estação 0, então eles obtêm esses dados, mas se não estão prontos os dados são «perdidos». Quando se transmitir programas pela rede para várias estações, cada estação recebedora deve introduzir:

```
LOAD *"n";0
```

e as margens do seu écran ficarão pretas enquanto esperam pelo programa. Então a estação com o programa deve introduzir:

```
SAVE *"n";0
```


e todas as estações «que estão à escuta» carregarão o programa. Quando você emite qualquer coisa não há uma maneira fácil para saber quem (se é que alguém) a recebeu. Contudo, se você lê alguma coisa da rede, para saber de que estação ela vem, pode:

```
PEEK 23759
```

que dará o número da estação que a está a enviar.

Programa «escravo» da impressora e microdrive

A grande vantagem da criação da rede é que muitos Spectrums podem partilhar periféricos, tais como impressoras e microdrives. Seguem-se dois programas que permitem que cada estação «escravo» (como as dos alunos) use a impressora e os microdrives na estação «senhor» (tal como a do professor). Os programas foram escritos para permitir que os periféricos sejam usados o mais facilmente possível, com comandos tais como SAVE e LPRINT. Um programa é para a estação «senhor» e o outro para cada «escravo». Para começar, cada estação deve carregar o programa «Escravo», e a maneira mais simples é a estação «senhor» emiti-lo para cada uma das outras. O programa «Senhor» deve então ser carregado na estação com a impressora e os microdrives, e o programa corre. Quando o código-máquina foi introduzido, a margem do écran ficará preta, e o sistema está pronto.

Cada estação «escravo» pode formatar-se a si própria para ser qualquer número de estação, com excepção de 32 e 64. Podem ser escritos e depurados programas nos «escravos», desde que os números de linhas se mantenham inferiores a 9000, para evitar conflitos com o programa «Escravo». Se um «escravo» deseja usar um dos periféricos do «senhor» deve teclar GO TO 9000. Será apresentada uma ementa e o utilizador pode usar a impressora, guardar e carregar programas, ou catalogar os seus programas.

Se ele escolhe a opção 1, fica ligado via «senhor» a qualquer impressora com ele ligado — quer seja ZX quer seja do tipo RS232. Pode então usar LPRINT, LLIST, até ter acabado, e depois dar a instrução CLOSE#3. A opção 2 permite ao utilizador salvar um programa de qualquer tipo de ficheiro (por exemplo SCREEN$)

num cartucho da estação «senhor». A opção 3 permite-lhe carregar qualquer ficheiro guardado com a opção 2 do cartucho do «senhor». A opção 4 imprime um catálogo de todos os programas do utilizador, e a opção 5 é para sair.

Listagem do programa «Escravo»

```
9000 CLS
9010 PRINT '"  1. Usar a impress
ora"
9020 PRINT '"  2. SAVE um progra
ma"
9030 PRINT '"  3. LOAD um progra
ma"
9040 PRINT '"  4. CAT o seu cart
ucho"
9050 PRINT '"  5. Desistir"
9060 PAUSE 0: LET a$=INKEY$
9070 IF a$="1" THEN GO TO 9150
9080 IF a$="2" THEN GO TO 9300
9090 IF a$="3" THEN GO TO 9400
9100 IF a$="4" THEN GO TO 9500
9110 IF a$<>"5" THEN GO TO 9060
9120 GO TO 9000
9147 REM ************************
9148 REM *Enviar p/ impressora *
9149 REM ************************
9150 LET a$="imprimir": GO SUB 9
200
9160 CLOSE #3: OPEN #3;"n";64
9170 PRINT "A impressora esta pr
onta. Usar LPRINT, LLIST etc. Qu
ando acabar, faca CLOSE #3"
9197 REM ************************
9198 REM *Difundir a$ e esperar*
9199 REM ************************
9200 PRINT INVERSE 1;"Contactand
o estacao ""Senhor"""
9205 CLOSE #4: OPEN #4;"n";0
9210 PRINT #4;a$
9220 CLOSE #4
9230 OPEN #4;"n";64
9240 IF INKEY$#4="" THEN GO TO 9
205
9250 CLOSE #4
9260 PRINT "Contacto estabelecid
o"
9270 RETURN
9297 REM ************************
9298 REM *   SAVE um programa   *
9299 REM ************************
9300 LET a$="guardar": GO SUB 92
00
9310 INPUT "Nome de ficheiro par
a SAVE:"; LINE f$
```

```
9320 OPEN #4;"n";64
9330 PRINT #4;f$
9340 CLOSE #4
9350 PRINT "Microdrive pronto. F
zer          SAVE *""n"";64 etc"
9360 GO TO 16000
9397 REM ************************
9398 REM *   LOAD um programa   *
9399 REM ************************
9400 LET a$="carregar": GO SUB 9
200
9410 INPUT "Nome do ficheiro par
a LOAD:"; LINE f$
9420 OPEN #4;"n";64
9430 OPEN #4;f$
9440 CLOSE #4
9450 OPEN #4;"n";64
9460 INPUT #4;st
9470 CLOSE #4
9480 IF st<>6 THEN PRINT "Fichei
ro nao encontrado": STOP
9490 PRINT "Microdrive pronto. F
azer        LOAD *""n"";64 etc"
9495 GO TO 16000
9497 REM ************************
9498 REM *       CATalogo       *
9499 REM ************************
```



```
9500 LET a$="cat": GO SUB 9200
9510 CLS
9520 PRINT "CATALOGO:"
9530 MOVE "n";64 TO #2
9540 PRINT ' FLASH 1;"Prima qual
quer tecla para       continuar"
9550 PAUSE 0
9560 GO TO 9000
```

Listagem do programa «Senhor»

```
  1 REM ************************
  2 REM *  Programa "Senhor"   *
  3 REM ************************
 10 CLEAR 65089
 20 LET st=65260: GO SUB 8000
 30 LET open=65090: GO SUB 8350
 40 DEF FN p(p)=PEEK p+256*PEEK
(p+1)
 100 FORMAT "n";64
 110 CLOSE #4
 120 OPEN #4;"n";0
 130 INPUT #4; LINE a$
 140 CLOSE #4
 150 LET n=PEEK 23759
 160 IF a$="imprimir" OR a$="gua
rdar" OR a$="carregar" OR a$="ca
```

```
t" THEN GO TO 180
 170 GO TO 120
 180 OPEN #4;"n";n
 190 PRINT #4
 200 CLOSE #4
 210 IF a$="guardar" THEN GO TO
300
 220 IF a$="carregar" THEN GO TO
 600
 230 IF a$="cat" THEN GO TO 800
 237 REM ************************
 238 REM *Necessario impressora*
 239 REM ************************
 240 PRINT "Impressora a ser usa
da pela estacao ";n
 250 MOVE "n";n TO #3
 260 LPRINT ''''''''
 270 PRINT "(Impressao terminada
)"
 280 GO TO 110
 297 REM ************************
 298 REM *   Necessario SAVE    *
 299 REM ************************
 300 OPEN #4;"n";n
 310 PRINT "Fazendo SAVE para a
estacao ";n
 320 DIM f$(10): INPUT #4; LINE
```



```
f$
 330 CLOSE #4
 340 GO SUB 500: LET f$(10)=CHR$
 n
 350 FOR i=1 TO 10
 360 POKE USR "a"+i-1,CODE f$(i)
 370 NEXT i
 380 POKE 23766,d: POKE 23768,5
 390 LET a=USR open: CLOSE #5
 400 IF a<>1 THEN ERASE "m";d;f$
 410 OPEN #5;"m";d;f$
 420 POKE FN p(5*2+23566+8)+FN p
(23631)-1+67,4
 430 MOVE "n";n TO #5
 440 CLOSE #5
 450 PRINT "(Programa gravado)"
 460 GO TO 110
 497 REM ************************
 498 REM *   Calcular D de N    *
 499 REM ************************
 500 LET d=1
 510 RETURN
 597 REM ************************
 598 REM *   LOAD um programa   *
 599 REM ************************
 600 OPEN #4;"n";n
 610 DIM f$(10): INPUT #4; LINE
```

```
f$
 620 CLOSE #4: LET f$(10)=CHR$ n
 630 PRINT "Fazendo LOAD para a
estacao ";n
 640 GO SUB 500
 670 FOR i=1 TO 10
 680 POKE USR "a"+i-1,CODE f$(i)
 690 NEXT i
 700 POKE 23766,d: POKE 23768,5
 710 LET a=USR open
 720 OPEN #4;"n";n
 730 PRINT #4;a
 740 CLOSE #4
 750 IF a<>6 THEN GO TO 770
 760 MOVE #5 TO "n";n
 770 CLOSE #5
 780 PRINT "(Programa carregado)
"
 790 GO TO 110
 797 REM **************************
 798 REM *   CAT um cartucho    *
 799 REM **************************
 800 PRINT "CATalogar para a est
acao ";n
 805 GO SUB 500
 810 LET z$="": CAT #14,d
 820 OPEN #4;"n";n
```



```
 830 LET z$=z$(13 TO )
 840 IF LEN z$(10 THEN GO TO 880
 850 IF z$(10)=CHR$ n THEN PRINT
 #4;z$( TO 9)
 860 LET z$=z$(12 TO )
 870 GO TO 840
 880 PRINT #4: CLOSE #4
 890 PRINT "(CAT terminado)"
 900 GO TO 110
 910 REM
```

De notar que o programa «Senhor» exige que as rotinas «OPEN # qualquer coisa» e «Stream 14-z$» estejam presentes das linhas 7995-8510.

Explicarei ambos os programas juntamente, de modo que você pode ver o que acontece em ambas as estações:

A primeira coisa que o «Senhor» faz é estabelecer programas em código-máquina — as estações «senhor» de 16K devem substituir:

```
10 CLEAR 32489
20 LET st=32490: GO SUB 8000
30 LET open=32490: GO SUB 8350
```

O «senhor» deve ser a estação 64 — a linha 100 estabelece-o como tal.

Um «escravo» deve fazer primeiro contacto com o «senhor», e isto é feito emitindo uma palavra através da rede, e o «senhor» acusa a recepção dela quando a tiver notado. Os ficheiros do «escravo» estão armazenados num cartucho como nove letras do nome, e a décima letra é CHR$ N, onde «N» é o número da estação «escravo». Isto proporciona um método de manter separados os programas dos escravos.

Quando um «escravo» usa a opção 1 para pedir acesso para a impressora, ele emite repetidamente «imprimir» («print»), até que

um carácter é recebido da estação 64. Este estabelecimento de contacto é feito por uma sub-rotina em 9200, a qual abre a stream 4 como uma emissão (9205), envia «a$» (9210), e depois fecha a stream. Vê então se houve qualquer resposta da estação 64 (9230-9240). Se não houve resposta, «a$» é outra vez emitido. Quando não há resposta, a stream é fechada (9250). Uma vez que o contacto tenha sido estabelecido, a linha 9160 abre a stream 3, de modo que ele envia dados para a estação «senhor». O GO TO 16000 em 9180, e outras linhas, é para parar o programa com a mensagem *Programa terminado*.

O programa «Senhor» lê continuamente cadeias da stream de emissão, usando as linhas 120-140. Verifica então se há um pedido dum «escravo» (160). Se não há, continua a fazer de monitor das emissões (170), de outro modo é calculado o número do «escravo» (150) e enviada uma resposta para ele (180-200). Cada palavra-pedido é então testada, e é feito um GOTO apropriado. Se a palavra «imprimir» é emitida, é impressa uma mensagem adequada na televisão do «senhor», e o poderoso comando MOVE é usado para transferir toda a entrada da estação «escravo» para todo e qualquer dispositivo que esteja ligado à stream 3 (linha 250). Quando o «escravo» faz entrar CLOSE # 3, o comando MOVE acabará, e será impressa uma outra mensagem. Ele então volta a entrar no ciclo para de novo fazer de monitor da emissão.

Quando um «escravo» escolhe a opção 2 para salvar o seu programa, a palavra «save» é emitida até que o «senhor» responda (linha 9300). É-lhe pedido um nome de ficheiro para o programa, o qual é enviado para o «senhor» (9310-9340). É impressa uma mensagem para ele, e ele pode então fazer SAVE *«n»; 64 seguido por seja qual for o tipo de ficheiro. Se se trata apenas de guardar programas, substituir a linha:

```
9350 SAVE *"n";64: GO TO 9000
```

Quando o «senhor» lê a palavra «save» proveniente duma estação no canal de emissão abre uma stream para o «escravo» e imprime uma mensagem (300-310). A linha 340 usa então a sub-rotina em 500 para calcular qual o microdrive que corresponde com a estação que está a chamar. Nesta versão o drive 1 é sempre seleccionado; mas, se há um grande número de «escravos», isto pode ser impraticável. Por exemplo, se há 63 «escravos», e o senhor tem oito drives, cada um deles podia ser usado por oito estações, e a linha 500 seria:

```
500 LET d=1+INT (n/8)
```

Tendo encontrado o número do drive, o décimo carácter do nome do ficheiro é estabelecido para marcar a estação à qual o programa pertence (340) e ele é colocado na área dos gráficos do utilizador (350-370). Isto, conjuntamente com a linha 380, estabelece tudo para usar a rotina «OPEN qualquer coisa» na stream 5, linha 390. Se o ficheiro solicitado lá existe, então é apagado (400). A linha 410 abre um ficheiro de escrita com um nome adequado, e 420 engana o sistema fazendo-o pensar que não é um ficheiro de dados. O programa que é enviado do «escravo» é escrito no ficheiro, uma vez mais usando MOVE (430). Quando o SAVE acabou, a stream é fechada e é impressa uma mensagem (440-450).

Quando o «escravo» escolhe a opção 3, carregar um programa, a palavra «load» (carregar) é difundida, e é esperada uma resposta (9400). É então pedido um nome de ficheiro e enviado para o «senhor» (9450-9470) e é feita uma verificação para ver se o ficheiro existe. Se ele existe, pode ser carregado com:

```
LOAD *"n";64
```

seguido por quaisquer que sejam os atributos requeridos.

Se se pretendem somente programas, substituir a linha:

```
9490 LOAD *"n";64: GO TO 90000
```

Quando o «senhor» recebe a palavra «load» (carregar) (220), lê o nome do ficheiro do «escravo» (600-620) e estabelece o décimo carácter para marcar a origem. O número do drive é calculado (64) e o nome do ficheiro é colocado na área de gráficos do utilizador. A linha 700 faz mais colocações antes de usar «OPEN # qualquer coisa» (710) para verificar a presença do programa pedido. O número retornado é enviado de volta para a estação «escravo». Só se o ficheiro é do tipo correcto é que será carregado do drive e enviado para o «escravo», usando MOVE (760). A stream do microdrive é então fechada e é impressa uma mensagem (770-780).

Se a estação «escravo» escolhe a opção 4 para catalogar os seus programas, a palavra «cat» é emitida até que seja recebido um reconhecimento. O écran é limpo e é impresso um título, sendo en-

tão impressa toda a entrada da estação «senhor» (isto é, os nomes do ficheiro) é impressa, usando MOVE (9530).

Quando a estação «senhor» recebe um pedido «cat» (230), é impressa uma mensagem e calculado o número do drive (800-805). Usando «Stream 14-z$», é dirigida uma ordem «CATalogue» para «z$» (810), e é aberta uma stream para o «escravo» (820). O título é removido da cadeia (830), e se há a seguir um nome de cartucho é testado o décimo carácter. Se se verificar ser CHR$ N, isto é a chamada do programa «escravo», o seu nome é então enviado para o «escravo» (850). Quando não há mais nomes de ficheiros, a stream é fechada e impressa a mensagem (880-890).

Melhorias adicionais

Alguns utilizadores podem querer características adicionais nestes programas. Um melhoramento seria armazenar uma lista de «códigos de acesso» ou «palavras de passe» *(passwords)* para cada estação (quer em quadro quer em ficheiro de dados), de modo que uma estação a fingir ser outra não pudesse destruir os programas de um outro qualquer. Se os nomes dos utilizadores estivessem também armazenados, as saídas da impressora podiam ter um cabeçalho com o nome do utilizador para os separar.

Adicionais à rede para o programa «Listagem de Stream»

As linhas que se seguem devem ser acrescentadas ao programa para lhe permitir reconhecer as streams da rede. Imprimirá o número da estação e o número do destino, e indicará se está a ser usada a característica de emissão.

Listagem «Stream 4»

```
1650 IF F$="N" THEN GO TO 3500
3499 REM Canal N
3500 PRINT INVERSE 1;"REDE"
3510 GO SUB 1800
3520 PRINT "Numero da estacao:";
PEEK 23749
3530 PRINT "transmitindo para:";
PEEK (D+11);"(emitindo)" AND PEE
K (D+11)=0
3540 GO TO 1500
```

CAPÍTULO 8

O código-máquina e a interface

Este capítulo é escrito para os leitores que têm conhecimentos do código-máquina Z80, se bem que os que o não conhecem possam mesmo assim achar úteis algumas partes.

A ROM 8K «Sombra»

Adicionalmente à ROM 16K do Spectrum, a interface 1 tem uma ROM 8K que está paginada na mesma área de memória, isto é, a partir de 0000-1FFFh (os números seguidos por um «h» são em hexadecimal). Esta disposição muito inteligente do mecanismo de paginação tornou muito simples, para o sistema operacional do Spectrum, usar as rotinas extras na ROM sombra, mas torna a vida bastante mais complicada para os que ainda não têm experiência em código-máquina que as desejam usar. Antes de aprofundar mais o sistema operacional e a ROM, deve ser explicado o engenhoso mecanismo de paginação. Para evitar qualquer confusão, as posições da ROM sombra são, neste livro, precedidas por um «X».

O mecanismo de paginação

Com uma única excepção, a ROM sombra só é paginada quando o contador de programa do Z80 atinge 0008, isto é, quando ocorre um RST 8. Na ROM 16K, este rearranque é seguido por um

byte de dados que determina que mensagem de erro vai ser produzida enquanto um programa BASIC está a ser executado (ou para produzir um ponto de interrogação a piscar se a sintaxe duma linha está errada). Se é feita uma tentativa para usar qualquer dos comandos adicionais (por exemplo, CAT) ou uma sintaxe ampliada (por exemplo, LOAD*) o BASIC 16K forçará um erro, usando RST 8. Com a interface ligada, isto despagina o BASIC, e pagina a ROM sombra. Calcula então o que causou o erro, e se foi uma operação de BASIC ampliado. Se não foi, é usada a rotina normal de manipulação do erro (na ROM 16K). Se foi um comando ampliado, a ROM sombra actua então nele (fazendo-o ou verificando a sua sintaxe), e depois despagina-se a si própria após ter limpo o erro que o fez paginar. Deve dizer-se que é um método brilhante para acrescentar comandos sem mudar um só byte da ROM 16K do Spectrum, mas faz que seja bastante cheio de artimanhas o uso das rotinas da ROM sombra para os próprios programas de cada um.

Para o ajudar a ultrapassar este obstáculo, os autores do ROM sombra usaram o que eles chamaram «*Hook codes*», que permitem que uma rotina tenha acesso a certas rotinas da ROM sombra. Estes *hook codes* são bytes de dados que seguem um comando RST 8, e vão desde 1Bh até 32h, inclusive. Os códigos de FFh a 1Ah produzem as mensagens de erro normais, ao passo que os códigos para cima de 32h produzem um *Hook code error* (Erro Hook code), não documentado neste manual.

Alguns dos *hook codes* são muito úteis, e outros não o são tanto, mas o programador deve estar consciente do estado das «interrupções mascaráveis» *(Maskable Interrupts)* na entrada e saída para algumas rotinas. Se vai ocorrer um retorno para o BASIC (depois duma função USR), então o valor do par de registos HL deve sempre ser preservado quando se usarem as rotinas do microdrive. A falha em o preservar pode bloquear o calculador de vírgula flutuante, e em seguida o sistema.

O *hook code* 32h é talvez o mais útil, contudo ele é oficialmente designado como *Reservado para futura expansão pelo SRL*. Isto é feito para esconder o facto de que ele permite a uma rotina do utilizador chamar a maior parte das rotinas da ROM sombra, assim como permite que a ROM sombra seja paginada e despaginada como bem se quiser.

Alguns leitores podem estar interessados em desassemblar a ROM 8K, mas sem dúvida um PEEK ou um LD A,(HL) só lerá a

ROM 16K. Se se pretender uma total desassemblagem, a maneira que eu arranjei de o fazer foi com o programa seguinte num Spectrum 48K:

```
  10 CLEAR 32767
  20 SAVE *"m";1;"8K ROM" CODE 0
,8192
  30 LOAD *"m";1;"8K ROM" CODE 3
2768
```

Quando o SAVE é executado, a ROM sombra está no local e guarda uma cópia dele no cartucho. A linha 30 só o carrega de volta em 8000h acima de onde ele normalmente está, mas lembre-se de substrair 8000h de todos os endereços absolutos.

À parte RST 8, a outra ocasião em que a ROM sombra é paginada é quando PC = 1708h, o que é o meio da rotina do comando CLOSE #. Isto é porque ele contém um *bug* fatal e é bastante inadequado para lidar com os tipos de canais extras usados com a interface.

Como funcionam as streams e os canais

De maneira semelhante ao BASIC, as streams e os canais estão disponíveis para os programas em código-máquina. As streams usam a área STRMS da memória, e estão ligadas com os canais que usam a área CHANS. A área STRMS é fixa e está situada de 5C10h a 5C35h, inclusive, e consiste de 19 pares de bytes, um par para cada stream de FDh a 0Fh. Cada par forma um deslocamento de 16 bits, que é 0000 se a stream não tem canal associado com ela. Se não é zero, é usado para indiciar na área CHANS. Quando inicializada, a área CHANS consiste de 20 bytes, de quatro grupos de cinco, com cinco bytes por canal. A área CHANS tem o início na posição apontada pela variável de sistema CHANS em 5C4Fh, e termina dois bytes antes da zona de PROG (zona em que o computador armazena os programas em BASIC).

Inicialmente a informação do canal contém dados em quatro canais, com cinco bytes cada, na ordem «K», «S», «R» e «p». Os primeiros dezasseis bytes da área STRMS contêm os bytes que se indicam no quadro seguinte, com o canal seleccionado:

NO. DA STREAM	LOCALIZAÇÃO DA STREAM	DESLOCAMENTO DO CANAL	CANAL
FD	5C10	0001	«K»
FE	5C12	0006	«S»
FF	5C14	000B	«R»
00	5C16	0001	«K»
01	5C18	0001	«K»
02	5C1A	0006	«S»
03	5C1C	0010	«R»
04	5C1E	0000	Nenhum

Para achar o canal adequado para uma stream, o deslocamento do canal para ele é adicionado a CHANS (depois de se verificar se não é zero), e então diminuído de um. O resultado aponta então para o inicio dos dados correspondentes, que é pelo menos cinco bytes. A primeira parte dos bytes aponta para a rotina de saída da ROM, o segundo par para a rotina de entrada, e o quinto é o especificador do canal em maiúsculas. Por exemplo, o canal «S» é armazenado como se segue:

```
CHANS + 5   DEFW 09F4h
            DEFW 15C4h
            DEFM «S»
```

De facto, 09F4h é usado como a rotina de saída para os canais «K», «S» e «P», com diferentes acções a serem tomadas consoante o especificador do canal.

Os canais da interface

Para usar qualquer das características da interface, o sistema requer 58 bytes extras para as variáveis do sistema, a serem inseridos a partir da posição 5CB6h. Esta área deve ser estabelecida

antes que sejam usadas quaisquer rotinas ROM sombra ou *Hook code,* e antes que sejam criados quaisquer tipos de canais extras (tal como a rotina «Stream 14-z$»). O modo de se fazer isto é com as instruções:

```
CF RST 08
31 DEFB 31h
```

Se a interface não está ligada, será produzido comunicado de erro não-existente «i». Se ela está ligada, os 58 bytes são inseridos, a nenhum erro ocorrerá. As variáveis são explicadas no Apêndice A.

As variáveis de sistema extras são normalmente criadas quando:

i) Ocorre qualquer erro (na sintaxe ou na execução);
ii) Qualquer comando BASIC ampliado (modo extensivo) é introduzido ou executado;
iii) É executado um comando CLOSE #.

Inicialmente a variável VECTOR em 5CB7h é estabelecida para 01F0h, mas isto pode ser mudado para permitir que sejam adicionados comandos extra ao BASIC. Este aspecto da interface podia encher por si próprio um livro, contudo num capítulo subsequente são dados alguns pormenores.

A variável IOBORD em 5CC6h é usada para determinar a cor da margem durante certas operações de I/O (entrada/saída), nomeadamente RS232, formação da rede, e quando se estiver a escrever para um cartucho. Se você preferir uma determinada cor da margem, pode fazer POKE 23750, com a cor que se pretende. Por outro lado, se você não gosta mesmo da margem a piscar, deve fazer:

```
POKE 23750,INT (PEEK 23624/8)
```

o que faz a cor do I/O a mesma que a cor normal da margem.

Os canais extra da interface «M», «N», «T» e «B» usam um formato ampliado para armazenar os seus pormenores na área CHANS.

O canal «M»

Este ocupa 595 bytes na área CHANS, e é endereçado pelo registo IX na ROM sombra. A localização dos bytes é como se segue (todos os deslocamentos em decimal):

0	DEFW 8	rotina «saída»
1	DEFW 8	rotina «entrada»
4	DEFM «M»	especificador do canal
5	DEFW 11D8h	saída da ROM sombra
7	DEFW 1122h	entrada da ROM sombra
9	DEFW 595	comprimento desta área CHANS
11	CHBYTE	posição corrente do buffer (0-512)
13	CHREC	posição do registo ao ficheiro (0-255)
14	CHNAME	10 bytes do nome do ficheiro
24	CHFLAG	bit 0 — *reset:* ficheiro de leitura *set:* ficheiro de escrita
25	CHDRIVE	número do drive (1-8)
26	CHMAP	localização do drive MAP
28	HPREAM	12 bytes do cabeçalho do preâmbulo
40	HDFLAG	bit 0 — *set* para assinalar o cabeçalho bits 1-7 não usados (1, 3, 4, 5 *set*, 2, 6, 7 *reset*)
41	HDNUM	número do sector (0-255)
42	DEFW 0E31h	não usado
44	HDNAME	10 bytes do nome do cartucho
54	HDCHK	verificação de soma do cabeçalho
55	DPREAM	12 bytes do preâmbulo dos dados
67	RECFLG	bit 0 — *reset* para assinalar «não um cabeçalho» *(header)* bit 1 — *reset:* não EOF *set:* EOF bit 2 — *reset:* um ficheiro PRINT *set:* não é um ficheiro PRINT bits 3-7 — não usados (todos *reset*)
68	RECNUM	número do registo (0-255)
69	RECLEN	número de bytes em registo (0-512)
71	RECNAM	10 bytes do nome do ficheiro
81	DESCHK	verificação de soma de RECFLG a DESCHK—1
82	CHDATA	primeiro dos 512 bytes do buffer
593		último byte do buffer
594	DCHK	verificação de soma do buffer

Cada microdrive em uso também tem 32 bytes postos à sua disposição na área MAP do Microdrive, de 5CFOh a CHANS—1. Cada sector de 32 bytes é um bit-mapade cada sector do cartucho. Se um bit é *set*, então o sector não é utilizável, pois que ele ou contém dados ou está danificado.

O canal «N»

Este toma 276 bytes da área CHANS, e é endereçado pelo registo IX na ROM sombra. A localização dos bytes é a seguinte:

0	DEFW 8	
2	DEFW 8	
4	DEFM «N»	especificador do canal
5	DEFW 0D6Ch	rotina saída da sombra
7	DEFW 0D0Ch	rotina entrada da sombra
8	DEFW 276	comprimento desta área CHANS
11	NCIRIS	número do posto de destino (0-64)
12	NCSELF	número deste posto (0-64)
13	NCNUMB	número do blocoi (0-65535)
15	NCTYPE	tipo do pacote *(packet)* — 0: dados, 1: fim do ficheiro
16	NCOBL	bytes em bloco de dados (ou 0 se saída)
17	NCDCS	verificação de soma dos dados
18	NCHCS	verificação de soma do cabeçalho
19	NCCUR	posição do buffer de entrada
20	NCIBL	número de bytes no buffer de entrada (0 se saída)
21	NCB	início dos 255 bytes do buffer de entrada

Canais «T» e «B»

Cada um destes ocupa 11 bytes na área CHANS, o mínimo para canais que não sejam canais-padrão. A colocação dos bytes é:

```
0 DEFB 8
2 DEFB 8
4 DEFB «T»                        especificador (para «b» e «t»)
5 DEFW 0C3Ch(T) ou 0C5Ah(B)       saída da sombra
7 DEFW 0B6Fh(T) ou 0B75h(B)       entrada da sombra
9  DEFW 11                        comprimento desta área CHANS
```

Quando se usar SAVE*, etc., MOVE, FORMAT e ERASE do BASIC ou código-máquina, são estabelecidos canais «temporários». Estes são semelhantes aos canais acima, mas com quatro diferenças importantes:

i) O byte especificador tem o bit superior *set;*
ii) Nunca estão ligados a uma stream;
iii) São anulados quando o comando terminou, ou quando ocorre qualquer erro;
iv) Estão sempre no fim de CHANS — um canal permanente nunca deve estar mais elevado na memória do que um canal temporário.

Com canais que não sejam padrão, o número mínimo de bytes, por canal na área CHANS é 11, como usado pelos canais «T» e «B». Os dois primeiros pares de bytes, usualmente os apontadores da rotina de entrada e saída, são ambos 0008, e obrigam a ROM sombra a paginar. Isso detecta que é precisa uma rotina de canal, e usa então o par de bytes localizado cinco mais adiante na área CHANS seleccionada, isto é, a rotina da ROM sombra. Quando cada rotina acaba, a ROM sombra despagina-se a si própria.

O par de bytes nas posições nona e décima são um número de 16 bits que diz ao sistema o número de bytes que o canal ocupa na área CHANS. É um par de bytes extremamente importante, visto

que é usado quando ocorre qualquer erro para atravessar a área CHANS à procura de canais temporários para fechar (CLOSE). Se eles são omitidos, ou são incorrectos, o sistema bloqueará quando ocorrer o primeiro erro.

Usando streams

Para usar uma stream para entrada ou saída, é utilizada a rotina STRM_OPEN em 1601h. (Não é de modo algum semelhante à instrução BASIC OPEN#). Na entrada o acumulador deve conter um número de stream válido (FDh a 0Fh). A partir dele é encontrado o deslocamento respeitante em STRMS, e é produzido um erro *Stream inválida* se é zero. Se não é zero, é calculada a localização correspondente em CHANS, e o seu valor é armazenado na variável de sistema CURCHL em 5C51h. O valor de CURCHL é também estabelecido por todas as rotinas OPEN em código-máquina.

Para a saída de caracteres para stream corrente, carrega-se o acumulador com o código e chama-se a rotina OUTPUT em 0010h com a instrução RST 10. Os registos BC, DE e HL não são alterados.

Para fazer entrar caracteres duma stream corrente, chamar a rotina INPUT em 15E6h. Os registos BC, DE e HL permanecerão inalterados. No retorno, a *carry flag* será *set* se A contém um carácter de leitura válido. A *flag* zero será *set* se não tiver sido lido um carácter, de outro modo *carry* e zero serão ambos *reset* se ocorreu uma situação *End of file* (Fim do ficheiro).

A utilização dos canais

Não existe, infelizmente, em código-máquina um equivalente simples para a instrução OPEN#. Para experimentar, as streams que se pretendem podiam ser abertas do BASIC, e usadas em código-máquina, mas isto é provavelmente insatisfatório para uma utilização final. Os equivalentes em código-máquina para cada tipo

de declaração OPEN# são dados mais adiante neste capítulo, na secção adequada.

O equivalente em código-máquina para CLOSE# é como se segue:

```
LD A, stream
RES 1,(1Y+124)      ;assinalar «não uma CLEAR#»
LD HL11718h         ;(rotina CLOSE da ROM sombra)
LDz(HD__11),HL
RST 8
DEFB 32h
RET
```

Esta rotina usa o *hook code* 32h para chamar a rotina CLOSE# da ROM sombra.

Rotinas ROM

Temos a seguir uma lista das rotinas da ROM sombra para usar as facilidades da interface. São dados pormenores sobre o modo de usar a rotina (usualmente por *hook codes*) e as condições de entrada e saída dos registos e as interrupções. São também dados os registos cujos valores são alterados pela rotina, assim como a localização real das rotinas.

ROTINAS DO MICRODRIVE

OPEN__M — cria um canal «M» temporário na área CHANS.

Para usar	:RST 8 DEFB 22h
Entrada	:INT ligadas
Saída	:IX = CURCHL = início da área, HL = deslocamento da stream (= IX-CHANS + 1), INT ligadas
Regs	:AF, BC, DE, HL, B'C', D'E', H'L', IX
Localização	:X1B29h

Operação: Antes de usar isto, a variável de sistema D__STR1 deve conter o número do drive, N__STR1 o comprimento do nome do ficheiro, e T__STR1 deve apontar para o início do nome do ficheiro. No fim da área CHANS (desde que o espaço de memória o permita) deve primeiro ser inserida uma área de 595 bytes e os atributos correspondentes copiados para dentro dela. Se não existe uma área do mapa para o drive escolhido, é criada uma e é cheia com FFS. O drive é então posto em funcionamento e é efectuada a procura no cartucho do nome do ficheiro. O bite 0 de IX + 25(CHFLAG) é *reset* se é um ficheiro PRINT, de outro modo ele é *set* (para programas, bytes, etc.). Note-se que a palheta de protecção da escrita não é verificada. Para o fazer, depois da rotina faça:

```
IN A,(EFH)
AND 1               ;Z se está protegida
```

Deve notar-se que o motor do drive não será desligado por esta rotina. O canal criado é temporário. Para o tornar permanente e o incorporar numa stream, faça o seguinte:

```
LD A, stream
ADD A,A
LD HL,5C16h
LD E,A
LD D,O
ADD HL,DE
PUSH HL             ;guardar a localização da stream
RST 8
DEFB OPEN__M
PUSH HL             ;guardar o deslocamento da stream
XOR A
RST 8
DEFB MOTOR          ;desligar os motores
POP DE
POP HL
LD(HL),E
INC HL
LD(HL),D            ;armazenar novos dados em STRMS
RES 7,(IX+4)        ;torne-o permanente
RET
```

Nota do autor: Aparentemente, pressupunha-se que o código *hook* 2Bh devia fazer o que está acima, mas devido a um erro de impressão no código de origem da ROM sombra salta para a mesma rotina como código *hook* 22h!

MOTOR — ligue um motor do drive ou desligue-se todos.

Para usar	:RST 8 DEFB 21h
Entrada	:A = 1 a 8 para ligar o drive, A = 0 para os desligar todos
Saída	:INT desligadas se A< >0, ou ligadas se A = 0
Regs	:AF, BC, DE, HL
Localização	:X17F7h

Operação: Se o acumulador não é zero, o motor do microdrive correspondente é ligado e todos os outros são desligados, e as interrupções são desactivadas. Se A é zero, todos os drives são desligados e as interrupções são activadas. Se o drive seleccionado não está ligado, ocorrerá o erro *Microdrive não presente*.

CLOSE_M — Fechar o canal «M».

Para usar	:RST 8 DEFB 23h
Entrada	:IX = início da área do canal «M»
Saída	:INT ligadas
Regs	:AF, BC, DE, HL
Localização	:X12A9

Operação: Se o canal escolhido é um ficheiro de escrita, o conteúdo remanescente do buffer é enviado. O motor do drive é desligado, e é reclamada a área de 595 bytes A correspondente área do mapa é também reclamada, se não é usada por outro canal.

ERASE_M — apagar um ficheiro do microdrive.

Para usar	:RST 8 DEFB 24h
Entrada	:nenhuma
Saída	:INT ligadas
Regs	:AF, BC, DE, HL, B'C', D'E', H'L', IX
Localização	:X1D6Eh

Operação: Antes de usar isto, as variáveis de sistema D_STR1, T_STR1 e N_STR1 devem ser estabelecidas para o pretendido número de drive e cadeia do ficheiro a ser apagada. É criado o canal temporário «m» e efectua-se a procura do ficheiro no cartucho. Se é encontrado é apagado (contudo só uma cópia dele é apagada se há cópias múltiplas) desde que esteja presente a patilha de protecção de escrita. Se não estiver, ocorrerá um erro. Quando tiver terminado, o motor é desligado e o canal temporário é anulado.

RECLAIM_M — reclamar uma área «M» de 595 bytes.

Para usar	:RST 8 DEFB 2Ch
Entrada	:IX = início da área do canal «M»
Saída	:nenhuma
Regs	:AF, BC, DE, HL
Localização	:X10C4h

Operação: Primeiro que tudo, a área de 595 bytes é reclamada, e quaisquer streams apontando para ela são fechadas. A área do mapa de 32 bytes só é reclamada se não é usada para qualquer outro canal.

CAT — catalogar um cartucho do microdrive

Para usar	:LD HL, 1C58 LD (HD_11), HL RST 8 DEFB 32h
Entrada	:INT ligadas
Saída	:nenhuma
Regs	:AF, BC, DE, HL, A'F', B'C', D'E', H'L', IX
Localização	:X1C58h

Operação: Antes de ser usado, S_STR1 deve conter o número da stream para o catálogo, e D_STR1 o número do drive. Como não há código *hook* para CAT, o código 32h é usado em conjunção com HD_11. Primeiramente, a stream escolhida é aberta (usando 1601h), e então criado um canal «M» temporário. Cada cabeçalho no cartucho é examinado, e o nome do ficheiro armazenado no buffer de 512 bytes na área CHANS, a não ser que ele já esteja aí ou se começa com CHR$ 0. É inserido na posição alfabética correcta no buffer. Quando todo o cartucho tiver sido completamente lido, ou quando tiverem sido encontrados 50 nomes de ficheiros, o drive é desligado, e o nome do cartucho é impresso na stream escolhida. Cada nome de ficheiro é então extraído do buffer e impresso, e finalmente é impresso o número de kilobytes livres. Este último número é calculado a partir de metade do número de bits livres na área do mapa correspondente. Por último, a área «M» é reclamada.

FORMAT_M — Formatar um cartucho.

Para usar	:LD HL, 1B6Eh LD (HD_11), HL RST 8 DEFB 32h
Entrada	:Nenhuma
Saída	:INT ligadas
Regs	:AF, BC, DE, HL, B'C', D'E', H'L', IX
Localização	:X1B6Eh

Operação: Antes de usar isto, D_STR1 deve conter o número do drive, e N_STR1 e T_STR1 apontar para o título do cartucho desejado. Inicialmente, é criada uma área «M» temporária, e o drive é ligado. Após um curto período a patilha de protecção da escrita é testada, e produzido um erro se solicitado. A área do buffer é cheia com FCh, e o buffer é repetidamente escrito no cartucho com um cabeçalho numerado e um nome de ficheiro «invisível». Quando todo o cartucho tiver sido escrito, tudo é verificado várias vezes. Finalmente o motor é parado, e a área «M» reclamada.

Formato dos dados do cartucho

Antes de explicar as mais avançadas rotinas do microdrive deve ser explicada a disposição real dos dados na fita. O cartucho é dividido em cerca de 180 sectores, cada um contendo 512 bytes de dados. As rotinas na realidade fornecem 256 sectores, mas os sectores 0 e os acima de cerca de 180 ou não existem ou nunca são usados. Além disso, dois ou três sectores não são utilizáveis, pois estão situados onde está a junta do anel de fita.

Antes de cada sector, há um cabeçalho consistindo de 12 bytes de preâmbulo (que é dez zeros e dois FF), depois 15 bytes das variáveis HDFLAG-HDCHK. Cada cabeçalho tem um número diferente (HDNUMB) de 1 até cerca de 180, armazenados sequencialmente no cartucho.

Em seguida ao cabeçalho está o próprio sector, consistindo de 12 bytes de preâmbulo (como acima), os 15 bytes de variáveis (RECFLG a DESCHK, depois os 512 bytes de dados, e finalmente um byte de verificação de soma (DCHK). Um sector não está usado se o bit 1 de RECFLG e o bit 1 de RECLENhi (IX+70) estão ambos *reset*.

Como a maior parte dos ficheiros é demasiado grande para caber num sector, eles são divididos em registos de 512 bytes e cada registo é numerado sequencialmente, começando em zero. Se um sector contém menos de 512 bytes, o bit 1 de RECFLG é *set* para mostrar que é o último. Ficheiros tipo «não-PRINT» tais como programas, armazenam os seus atributos nos primeiros nove bytes, no registo número 0. Esses atributos são HD_00 a HD_11, explicados no Apêndice A.

Os restantes *hook codes* do microdrive são como se segue:

READ_P — Ler um registo a partir dum ficheiro PRINT.

Para usar	:RST 8 DEFB 27h
Entrada	:IX = início da área do canal «M»
Saída	:INT desligadas, motor ligado
Regs	:AF, BC, DE, HL
Localização	:X1A17h

Operação: Antes de usar esta rotina, CHDRIV deve conter o número do drive, CHREC o número do registo pretendido e CHNAME o nome do ficheiro. Em primeiro lugar, o drive escolhido é ligado e SECTOR estabelecido para 04FB (= 5 rotações completas). É então efectuada a procura no cartucho do número do registo escolhido do ficheiro escolhido. Se é encontrado e é um ficheiro PRINT, então o sector é lido para o buffer e é feito um retorno. Se se verifica que não é um ficheiro PRINT, então a área é reclamada e ocorre um *Wrong type file* (Tipo de ficheiro errado). Se o registo do ficheiro não é encontrado após cinco buscas na fita, ocorrerá o erro *File not found* (Ficheiro não encontrado), e a área é reclamada, se era temporária.

READ_NP — Ler o registo seguinte dum ficheiro PRINT.

Para usar	:RST 8
	DEFB 25h
Entrada	:IX = início da área «M»
Saída	:INT desligadas, motor ligado
Regs	:AF, BC, DE, HL
Localização	:X1A09h

Operação: Esta rotina é semelhante a READ_P, e CHDRIV deve conter o número do drive e CHNAME o nome do ficheiro. Em primeiro lugar, o bit 1 de RECFLG é verificado para ver se o registo na memória era o último — se era, então ocorrerá *End of file* (Fim de ficheiro). Se não era o último, então o conteúdo de CHREC é incrementado, e o controle continua em READ_P.

READ_S — Ler o sector seguinte.

Para usar	:RST 8
	DEFB 29h
Entrada	:IX = início da área «M»
Saída	:INT desligadas, motor ligado
Regs	:AF, BC, DE, HL
Localização	:X1A86h

Operação: Antes de chamar esta rotina, estabelecer CHDRIV e CHNAME para adaptar. O próximo cabeçalho e sector na fita

são lidos na área do canal. Se é um ficheiro PRINT, é feito um retorno com o *carry flag reset*. Se não é um ficheiro PRINT, o conteúdo do buffer é obliterado e é feito um retorno com *carry set*.

READ_R — Ler aleatoriamente um sector PRINT.

Para usar	:RST 8
	DEFB 28h
Entrada	:IX = início da área «M»
Saída	:INT desligadas, motor ligado
Regs	:AF, BC, DE, HL
Localização	:X1A4Bh

Operação: É feita a busca no cartucho para encontrar o número do sector CHREC. Se não é encontrado, o buffer é carregado. Se é um ficheiro PRINT, a *carry flag* é limpa e é feito um retorno. Se não é um ficheiro PRINT, então o conteúdo do buffer é obliterado e é feito *carry set*. Se o sector correspondente não é encontrado após uma rotação, ocorrerá então *File not found* (Ficheiro não encontrado).

WRITE_S — Escrever um sector em série.

Para usar	:RST 8
	DEFB 26h
Entrada	:IX = início da área «M»
Saída	:INT ligadas, motor desligado
Regs	:AF, BC, DE, HL
Localização	:X11FFh

Operação: Antes de chamar esta rotina, estabelecer as variáveis CHBYTE, CHREC, CHNAME e CHDRIVE e a área do buffer para se adaptar. Em primeiro lugar, o motor respectivo é ligado, e é feito um teste a todo o ficheiro. O nome do ficheiro é então copiado de CHNAME para RECNAN, e CHBYTE e CHREC copiados por RECLEN e RECNUM. São calculadas as verificações de soma DESCHK e DCHK, e a área de RECFLG a DCHK é escrita no primeiro sector do cartucho (desde que não seja de escrita protegida). É então estabelecido no mapa o bit correspondente, CHBYTE é levado a zero e CHREC incrementado. Finalmente o motor é desligado.

WRITE_R — Escrever um sector aleatoriamente.

Para usar	:RST 8
	DEFB 2Ah
Entrada	:IX = início da área «M»
Saída	:INT desligadas, motor ligado
Regs	:AF, BC, DE, HL
Localização	:X1A9h

Operação: Antes de chamar esta rotina, CHREC e CHDRIV devem ser estabelecidos para os valores desejados, e CHNAME conter o nome do sector a ser guardado. É efectuada uma vez uma busca no cartucho do número do sector CHREC. Se ele não pode ser encontrado, ocorrerá *File not found* (Ficheiro não encontrado). Se é encontrado, a patilha de protecção da escrita é testada. Se está presente, os bytes do sector e de dados (RECFLG-DCHK) são escritos no cartucho, e é estabelecido no mapa o bit correspondente. De notar que o mapa não é testado antes da escrita; deste modo, o teste deve ser feito pelo utilizador, se o desejar.

Rotinas da RS232

Antes de se usar qualquer das rotinas para a RS232, as variáveis extras do sistema devem estar presentes, e deve ser seleccionada uma adequada taxa baud. O primeiro pode ser feito com o *hook code* 31h, e o último alertando BAUD em 5CC3h. Como referência, a tabela de identificação para converter cada taxa baud para um adequado valor em BAUD está situada entre X0AEFh e X0B12h, e consiste de nove pares de bytes. O primeiro par é a taxa baud, e o segundo par o correspondente valor 16-bit para ir para BAUD.

Não há *hook codes* para abrir ou fechar canais da RS232, somente para a entrada e saída de caracteres, usando o canal «B».

232IN — Ler um byte a partir da RS232.

Para usar	:RST 8
	DEFB 1Dh
Entrada	:nenhuma

Saída	:*carry set* se ler um byte, A = byte, INT ligadas
Regs	:AF, BC, DE, HL
Localização	:X0B1h

Operação: Lê um byte a partir da *port* da RS232, a não ser que tenha de esperar de mais ou SPACE for premido (caso em que ocorrerá *Break into the program* — Interrupção do programa). A *carry flag* é *set* se um byte tiver sido lido.

32OUT — Enviar um byte a partir da RS232.

Para usar	:RST 8
	DEFB 1Eh
Saída	:A = byte código
Entrada	:INT ligadas
Regs	:AF, BC, DE, HL
Localização	:X0C5Ah

Operação: É enviado um byte através da *port* à velocidade correcta com o necessário *handshaking*. Se BREAK for pressionado durante a transmissão, ocorrerá um erro.

Se você desejar usar o equivalente ao canal «T» para a saída de RS232, utilize o *hook code* 32h chamar W0C3Ch, com A contendo o código do carácter.

OPEN_R — Abrir um canal da RS232.

Para usar	:LD HL, 0B13h
	LD (HD_11), HL
	RST 8
	DEFB 32h
Entrada	:INT ligadas
Saída	:DE = início da nova área CHANS
Regs	:AF, BC, DE, HL
Localização	:X0B13h

Operação: Isto cria uma área de 11 bytes no fim de CHANS para um canal «B» ou «T». Normalmente criará um canal «T», a não ser que L_STR1 contenha «B». Quando assim for, criará um canal «B».

Rotinas da formação de rede

A formação da rede é implementada no Spectrum por uma ligação contínua entre as *ports* de cada posto. Só pode ocorrer uma comunicação de cada vez, e todos os dados são transmitidos em série (mas, ao contrário do que sucede na RS232, não há bits de início e de separação, nem bits de paridade). Os dados são divididos em pacotes numerados, cada um com o máximo de 255 bytes. Um pacote é transferido da maneira seguinte: em primeiro lugar, o emissor verifica se a rede já está ocupada; se está, então espera. Quando não está ocupada, envia pelo cabo oito bytes dum cabeçalho. composto das variáveis de sistema NTDEST e NTHCS. Este cabeçalho contém várias informações acerca dos dados que se seguem, e em particular os números do emissor e do receptor, bem como duas verificações de soma. A não ser que esteja emitindo, o emissor faz um teste para o reconhecimento do receptor, na forma de um simples byte do número da estação receptora. Se o byte não é recebido, então o cabeçalho é outra vez transmitido. Se estiver a emitir, não é feito teste do byte de resposta, e os dados são transmitidos independentemente disso. A secção de dados não é enviada se não há nela bytes, isto é, NTLEN = 0, de outro modo é enviado o adequado número de bytes. É então testado um outro byte de resposta (a não ser que seja uma emissão); se não é recebido um, então repete-se o processo, e o cabeçalho volta a ser transmitido. Há quatro *hook codes* para usar na rede, contudo infelizmente um não é utilizável.

OPEN__N — Abrir um canal «N» temporário.

Para usar	:RST 8 DEFB 2Dh
Entrada	:nenhuma
Saída	:IX = DE = (CURCHL) = início da área «N» em CHANS
Regs	:AF, BC, DE, HL
Localização	:X0EA9h

Operação: Antes de chamar esta rotina, D__STR1 deve conter o número de destino e NSTAT o próprio número do Spectrum. No

fim da área CHANS é criada uma área de 265 bytes e nela são armazenados os dados. O número de destino é copiado de D__STR1 para NCIRIS, o número do posto de NSTAT para NCSELF, e a área de NCNUMB até ao último carácter no buffer é cheia com zeros. O canal é feito temporário usando o bit 7 do especificador de canal. A variável CURCHL é feita apontar para o início da área criada.

Para fazer permanente um canal da rede, use o código seguinte (depois de estabelecer D__STR1 e NTSTAT):

```
LD A, stream
ADD A;A
LD HL, 5C16h
LD E,A
LD D,O
ADD HL,DE          ;HL = localização adequada em STRMS
PUSH HL            ;guardá-lo
RST 8
DEFB OPEN__N       ;abrir um «N» temporário
RES 7,(IX+4)       ;torná-lo permanente
LD HL,(CHANS)
EX DE,HL
AND A
SBC HL,DE
INC HL             ;HL = deslocamento da stream
POP DE             ;DE = localização STRMS
EX DE,HL
LD (HL),E          ;armazenar o deslocamento em STRMS
INC HL
LD (HL),D
RET
```

CLOSE__N — Fechar um canal «N».

Para usar	:RST 8 DEFB 2Eh
Entrada	:(CURCHL) = início da área «N» em CHANS
Saída	:INT ligadas
Regs	:AF, BC, DE, HL, IX
Localização	:X1A24h

Operação: Se o canal é um ficheiro de escrita (isto é, NCOBL <>0), então o conteúdo remanescente do buffer é enviado num pacote marcado EOF. A área de 265 bytes é então reclamada, mas quaisquer streams com ela ligadas *não* são fechadas. A área «N» deve ser, quer temporária, o último canal na área CHANS, de outro modo outras streams e canais podem corromper-se.

WRITE_N — Enviar um pacote através da rede.

Para usar	:RST 8 DEFB 30h
Entrada	:A=0 para dados, 1 para EOF, IX=início da área «N»
Saída	:INT ligadas, A=número de destino (Z se em emissão)
Regs	:AF, BC, DE, HL
Localização	:X0DB2h

Operação: Antes de chamar esta rotina, as variáveis NCOBL e MCNUMB, o conteúdo do buffer e a área do cabeçalho (excluindo as verificações de soma) devem ser estabelecidas para se ajustar. Na entrada, o acumulador deve normalmente conter 0, ou 1 se o pacote é o último numa sequência, isto é, um marcador EOF. O valor é armazenada em NCTYPE, e a cor da margem estabelecida para IOBORD. Uma verificação de soma do conteúdo do buffer é armazenado em NCDCS, e uma verificação de soma de NCIRIS a NCDCS armazenada em NCHCS. As interrupções são desarmadas, e o pacote enviado. Quando o pacote tiver sido recebido (se não estiver em emissão) então NCNUMB é aumentado, a cor da margem restaurada e as interrupções armadas.

READ_N — (com a intenção de) Ler um pacote a partir da rede.

Para usar	:RST 8 DEFB 2Fh
Entrada	:IX=início da área «N»
Saída	:INT ligadas
Regs	:AF, BC, DE, HL
Localização	:X1A31h

Operação: Este *hook code* deveria ler um pacote de bytes duma rede, mas não é utilizável devido a um erro no seu fim. A *carry flag* tinha a intenção de indicar se um pacote era ou não lido, mas fica corrompido pois que a saída é por meio da rotina de restauração da margem, que destrói o valor prévio.

A melhor maneira de ler caracteres da rede é por meio da rotina INPUT do ROM 16K, em 15E6h, como foi mencionado previamente. Lembre-se de preservar CURCHL se usar quaisquer outros canais entre chamadas para ele.

Diversos códigos «hook»

PAUSE — Esperar que uma tecla seja premida (ou repetir).

Para usar	:RST 8 DEFB 1Bh
Entrada	:nenhuma
Saída	:INT ligadas, A=código chave
Regs	:AF, BC, DE, HL
Localização	:X19D9h

Operação: As interrupções estão activadas e cada cinquentésimo de segundo é chamada a rotina para ler o teclado no ROM 16K, até que uma tecla seja premida. O valor da tecla é lido a partir de LAST_K.

PRINT — Imprimir um carácter no écran.

Para usar	:RST 8 DEFB 1Ch
Entrada	:A=carácter código, INT ligadas
Saída	:nenhuma
Regs	:AF, BC, DE, HL, A'F', B'C', D'E'
Localização	:X19ECh

Operação: A variável SCR_CT é estabelecida para FF para permitir o rolamento automático, a stream FEh (canal «S») é aberta e o carácter impresso.

LPRINT — Imprimir um carácter na impressora ZX.

Para usar	:RST 8
	DEFB IFh
Entrada	:A = carácter código, INT ligadas
Saída	:nenhuma
Regs	:AF, BC, DE, HL, A'F', B'C', D'E'
Localização	:X19FCh

Operação: A stream 3 (canal «P» usualmente») é aberta e o carácter impresso.

SCAN_K — Ler a matriz do teclado.

Para usar	:RST 8
	DEFB 20h
Entrada	:nenhuma
Saída	:NZ se a tecla premida
Regs	:AF, BC, DE, HL
Localização	:X1A01h

Operação: O teclado é lido directamente e se qualquer tecla (incluindo quaisquer das teclas shift) é mantida em baixo, então a *flag* Z é *reset*. As interrupções não são necessárias.

NEWVARS — Criar as variáveis de sistema extras.

Para usar	:RST 8
	DEFB 31h
Entrada	:nenhuma
Saída	:nenhuma
Regs	:AF, BC, DE, HL
Localização	:X19A8h

Operação: Se as 58 variáveis de sistema extras (no Anexo A) não existem, então elas são criadas, desde que a memória o permita. (De facto, a rotina X19A8h contém só a instrução RET — as variáveis extras são criadas pelo próprio RST 8.)

SHADOW — Chamar qualquer rotina do ROM sombra.

Para usar	:RST 8
	DEFB 32h
Entrada	:(HD_11) = localização
Saída	:nenhuma
Regs	:não especificados (depende da rotina)
Localização	:X19A4h

Operação: É chamada a rotina do ROM sombra apontada por HD_11. Designado oficialmente por *Reservado para expansão futura por SRL,* é o mais poderoso código *hook* no ROM. Só o valor do registo A pode ser passado para a rotina, se bem que todos os valores sejam dele retornados. Pode mesmo ser usado para na realidade despaginar a ROM BASIC, com o seguinte código:

```
        :LD HL,PAGOUT
        :LD (HD_11),HL
        :RST 8
        :DEFB 32h
PAGOUT:POP HL           ;remover 0700h da pilha
        :POP HL         ;remover PAGOUT da pilha
                        ;resto do programa
```

Quando você quiser repaginar a ROM 16K, faça um CALL 0700h.

Sumário dos códigos «hook»

Este é um sumário de cada código *hook,* o seu nome, localização na ROM sombra e função.

CÓDIGO	NOME	LOCALIZAÇÃO	FUNÇÃO
1B	PAUSE	19D9	Esperar que uma tecla seja premida — valor em A.

CÓDIGO	NOME	LOCALIZAÇÃO	FUNÇÃO
1C	PRINT	19EC	Imprimir CHR$ A na stream FE (o écran).
1D	232IN	0B81	Entrada RS232 em A — *carry* se válido.
1E	232OUT	0C51	Saída RS232 — código do carácter em A.
1F	LPRINT	19FC	Imprimir CHR$ A na stream 3 (a impressora).
20	SCAN_K	1A01	Ler a matriz do teclado — NZ se qualquer premida.
21	MOTOR	17F7	Ligar ou desligar os motores dos microdrives.
22	OPEN_M	1B29	Abrir um canal temporário «M».
23	CLOSE_M	12A9	Fechar um canal «M».
24	ERASE	1D6E	Apagar um ficheiro dum cartucho.
25	READ_NP	1A09	Ler no próximo buffer PRINT.
26	WRITE_S	11FF	Enviar o buffer «M» para o cartucho.
27	READ_P	1A17	Ler num buffer PRINT.
28	READ_R	1A4B	Procurar ficheiro aleatório.
29	READ_S	1A86	Procurar ficheiro em série.
2A	WRITE_R	1A91	Escrever um bloco aleatoriamente.
2B	OPEN_M	1B29	O mesmo que 22 — um erro — FAULTY (não funciona).
2C	RECLAIM	10C4	Reclamar uma área «M» de 595 bytes
2D	OPEN_N	0EA9	Abrir um canal «N» temporário.
2E	CLOSE_N	1A24	Fechar um canal «N».
2F	READ_N	1A31	Ler um pacote da rede — FAULTY (não funciona).
30	WRITE_N	0DB2	Escrever um pacote para a rede.
31	NEWVARS	19A8	Criar o sistema de variáveis extras se necessário.
32	SHADOW	19A4	Chamar a rotina ROM sombra; endereço por HD_11.

Notas

Há um certo número de coisas de que um programador em código-máquina deve ter consciência quando usar a interface:

i) Usar a função USR somente em instruções LET ou RANDOMIZE. Se ela é usada em instruções BASIC ampliadas (modo extensivo), e ocorre um erro, a ROM sombra pode bloquear o sistema.

ii) Ter em conta o estado das interrupções na entrada e saída de algumas rotinas ROM sombra. As interrupções devem sempre estar ligadas (ON) antes de se regressar ao BASIC.

iii) Resguardar sempre o valor de H'L' ao usar as rotinas de microdrive se voltar ao BASIC.

iv) Nunca usar cartuchos importantes quando se fizer a depuração das rotinas de um microdrive. Pode criar-se uma situação muito embaraçosa se você acidentalmente formatar o cartucho que contém todo o seu código de origem! As patilhas de protecção da escrita não são garantia de segurança para erros selvagens de código-máquina.

v) Assegurar-se de que foram criadas as variáveis de sistema extras, usando o código *hook* 31h.

vi) Não usar instruções REM para rotinas de interface. A criação das áreas CHANS movê-las-ão no mapa de memória enquanto executam, e podem fazer bloquear o sistema.

CAPÍTULO 9

Acrescentando comandos BASIC

Tal como os comandos BASIC ampliados (modo extensivo) acrescentados ao Spectrum pela interface 1, pode também o programador acrescentar os seus próprios comandos. Isto é uma facilidade muito útil, presente em muitos outros computadores, mas sempre em falta na gama Sinclair. A razão principal da sua prévia omissão é o facto de, até agora, todos os comandos no Spectrum terem de ser palavras-chave, e o conjunto de caracteres não ter brechas para acrescentar comandos. O que a interface 1 faz é acrescentar a opção de saltar para um programa em código-máquina acrescentado se encontra um erro de sintaxe numa linha, quer quando ela é introduzida quer quando é executada.

Isto permite que a sintaxe da maior parte dos comandos existentes possa ser modificada para alterar a sua função, se bem que os comandos da interface não possam normalmente ser modificados. Para os comandos que não sejam usados pela interface, existem várias maneiras de acrescentar funções:

i) Acrescentar caracteres em locais estranhos, por exemplo, COPY #
ii) Mudar tipos de parâmetros, como por exemplo POKE 30000,«x»
iii) Acrescentar parâmetros, por exemplo, PLOT x,y, «BANG»
iv) Deixar de fora parâmetros, por exemplo, CIRCLE 10,30
v) Usar funções do modo E, por exemplo, LINE 10,30

Pode também usar os seus próprios comandos, escritos por extenso mas precedidos de um carácter não alfabético (por exemplo !REPEAT). O carácter não alfabético é necessário para tirar o Spectrum do modo K quando se bater a linha no teclado.

As palavras-chave cuja sintaxe não pode normalmente ser alterada são LOAD, SAVE, MERGE, VERIFY, CAT, FORMAT, OPEN, CLOSE, CLS, CLEAR e MOVE. E eu digo normalmente porque elas na realidade podem ser alteradas se você se não importar de fazer bastante exercício com os dedos quando as está a escrever! A maneira pela qual elas podem ser modificadas é precedê-las de um carácter mudado, tal como «*». Para as palavras-chave isto significa escrever a palavra, mover o cursor para a esquerda, escrever no teclado «*», mover o cursor para a direita, e depois escrever o resto da linha!

Para criar os seus próprios comandos, você deve ter um bom conhecimento de código-máquina e do sistema de funcionamento do Spectrum, e para os leitores a quem isso acontece temos os métodos descritos na segunda metade deste capítulo. A primeira metade contém um programa que altera a sintaxe dos comandos vulgares para microdrive, e pode ser introduzido e usado por qualquer leitor.

Para informar a interface de que alguns novos comandos foram acrescentados, você tem de usar duas instruções POKE que *devem* ser feitas numa linha de multienunciado. Não devem ser feitas antes de quaisquer operações de interface após um NEW. O meio mais simples é utilizar CLOSE #, o qual é realmente uma instrução muda. Para fazer interface voltar a utilizar somente os comandos normais, deve escrever:

```
POKE 23735,240: POKE 23736,1
```

uma vez mais numa só linha.

Devo admitir que não sou grande admirador da sintaxe escolhida para carregar e guardar programas no microdrive. Do meu ponto de vista ele é desnecessariamente não rendível e exige uma imensa quantidade de pressionamentos de teclas; por isso, escrevi o programa que se segue para lhes acrescentar comandos mais simples. Todos eles são sob a forma de um asterisco seguido pela adequada letra (maiúscula ou minúscula) de comando e pormenores. Para carregar um programa chamado «teste» do drive 2 o novo comando seria *L«2teste», o que é equivalente a LOAD*«m»;2;«teste». De modo semelhante, SAVE *«m», VERIFY e MERGE são suplementados por *S, *V, *M seguidos de uma simples cadeia. A primeira letra da cadeia deve ser o número do drive. Um outro comando acrescentado é *C, equivalente a CAT 1, e *C seguido de um número para catalogar os outros drives no écran. O novo comando final é *run, o qual carregará um programa chamado «run» do drive 1, é é equivalente a NEW seguido de RUN.

Listagem de comandos extra

```
  10 REM ************************
  20 REM *  Ampliar sintaxe M  *
  30 REM ************************
  40 REM                  para 16K
  50 CLEAR 65169: REM 32401
  60 LET ex=65170: REM 32402
  65 LET x2=INT ((ex+124)/256):
LET x1=ex+124-256*x2
  70 RESTORE 200: LET c=0
  80 FOR i=ex TO ex+194
  90 READ a: LET c=c+a
 100 POKE i,a
 110 NEXT i
 115 IF c<>22003+2*(x1+x2) THEN
PRINT "Erro de verificacao da so
ma": STOP
 120 CLOSE #0
 130 POKE 23735,ex-256*INT (ex/2
56): POKE 23736,INT (ex/256)
```

```
 140 PRINT "Novos comandos:"
 150 PRINT "*L LOAD"'"*S SAVE"'"
*M MERGE"'"*V VERIFY"
 160 PRINT "*C CAT"'"*RUN"
 170 STOP
 200 DATA 254,32,194,240,1,215,3
2,0
 210 DATA 246,32,254,115,40,22,2
54,108
 220 DATA 40,28,254,118,40,34,25
4,109
 230 DATA 40,36,254,99,40,38,254
,114
 240 DATA 40,62,24,222,253,203,1
24,238
 250 DATA 205,x1,x2,195,54,8,253
,203
 260 DATA 124,230,205,x1,x2,195,
165,8
 270 DATA 253,203,124,254,24,244
,253,203
 280 DATA 124,246,24,238,33,214,
92,54
 290 DATA 1,35,54,0,35,54,2,215
 300 DATA 32,0,254,13,40,7,254,5
8
 310 DATA 40,3,205,30,6,195,169,
```



```
4
 320 DATA 215,32,0,246,32,254,11
7,194
 330 DATA 40,0,215,32,0,246,32,2
54
 340 DATA 110,32,244,215,32,0,20
5,183
 350 DATA 5,195,149,10,62,77,50,
217
 360 DATA 92,215,32,0,215,140,28
,215
 370 DATA 24,0,223,202,35,7,215,
241
 380 DATA 43,33,11,0,167,237,66,
218
 390 DATA 76,6,120,177,202,76,6,
26
 400 DATA 214,48,254,1,56,27,254
,9
 410 DATA 48,23,50,214,92,175,50
,215
 420 DATA 92,11,19,237,67,218,92
,237
 430 DATA 83,220,92,215,24,0,195
,35
 440 DATA 7,231,4
```

Acho estes novos comandos muito mais fáceis e rápidos de usar, e tenho o programa guardado como «run» no meu cartucho do desenvolvimento do programa principal. Quando se introduzir este programa, devem ser alteradas as linhas 50 e 60 para se adequar à mudança da nova posição de RAMTOP e à localização de memória para onde o programa irá. As posições recomendadas são incompatíveis com as posições recomendadas de algumas outras rotinas deste livro. Se você desejar que todas elas sejam compatíveis, então acrescente as linhas:

```
 50 CLEAR 64579: REM 31809 para
16K
 60 LET mc=64580: REM 31810 par
a 16K
```

Quaisquer comandos extra serão inutilizados depois de um NEW; por isso, você deve criar no cartucho um ficheiro de programa vazio «new». Então, para apagar um programa corrente retendo os comandos, faça:

```
*L"new"
```

Como acrescentar comandos

Esta metade do capítulo é para os programadores em código-máquina. Seria também útil um certo conhecimento do sistema de funcionamento do Spectrum — para isso posso recomendar *The Complete Spectrum ROM Disassembly,* da autoria do Dr. Ian Logan, cuja rotina «Etiquetas I do ROM 16K» eu usei neste livro.

A variável VECTOR da interface é crucial para a facilidade dos comandos extra, sendo a posição da variável em 5CB7h. Isto normalmente contém 01F0h, que é uma posição da ROM sombra que executa a rotina de erro normal (depois de copiar CHADD__ de volta para CH__ADD).

Para explicar como alterar VECTOR, devem ser explicadas as

acções da ROM sombra quando ocorre um erro. Os números entre parênteses referem-se às posições ROM em hexadecímal.

Primeiramente HL é feito para conter o endereço de retorno (0013) (do RST 8), e é verificado para ver se é 0000 ou 15FE. Se é 0000, então uma rotina «16KROM» foi chamada do ROM sombra, ou se é 15FE então um canal interface foi solicitado. Se não é nenhum destes, então as variáveis da interface são criadas se ainda não existem, e certos sinais são enviados para o ULA nele. O número de erro é então encontrado (00C7) (do byte seguinte ao RST 8), e verificado para ver se é um byte *hook code.* Se não é, é verificado para ver se o erro era *Disparate em BASIC, Nome de ficheiro inválido,* ou *Cadeia não-válida.* Se não é nenhum destes, então a velha rotina de erro da ROM é usada (00F8). Se era um dos erros acima indicados, então o conteúdo do CH__ADD é armazenado em CHADD__ (que confusas mnemónicas são usadas!). O bit 3 de FLAG3 é verificado — se está *set,* a rotina normal de erro é usada. Depois de ter calculado o primeiro carácter da linha em que ocorreu o erro (0126-01E9), ele é comparado com os símbolos da interface (01B5-01E9). Se não é reconhecido, é então efectuado um salto para a rotina apontada por VECTOR. Normalmente isto produz um erro, mas se o VECTOR é mudado então você pode acrescentar comandos que falham a verificação de sintaxe normal.

Assim, se você modificou VECTOR para apontar para a sua rotina extra, que deve esta fazer? Primeiramente ela deve verificar o registo A para o primeiro símbolo ou carácter de código dos seus escolhidos comandos extras, menos 206. Se não é reconhecido, então JP 01F0h para produzir um erro. Se é reconhecido, você deve saltar para a rotina de comando, que deve verificar a sua sintaxe, avaliar quaisquer parâmetros e executá-los (os dois últimos somente durante o tempo de execução). Há algumas coisas de que se deve estar consciente quando se escrever uma tal rotina:

1. A ROM sombra está paginada, não o 16K BASIC.
2. Todos os *restarts* Z80 são diferentes.
3. NÃO tente usar os *hook codes* — chame directamente as rotinas.
4. Mesmo enquanto as interrupções estão ligadas, o teclado *não* é examinado e FRAMES não é incrementada.

Os *restarts* da sombra fazem o seguinte:

RST 0 — *resets* FLAGS3 e retorna para 16K ROM
RST 8 — NUNCA USAR
RST 10 — Chama (CALL) rotinas 16K ROM — seguido por dois bytes do endereço
RST 19 — BIT 7, (FLAGS) — Z se verificação de sintaxe, NZ se em execução.
RST 20 — faz um erro da ROM sombra. Seguir com um byte de dados:

FF	Programa acabado	00E7
00	Disparate em BASIC	0139
01	Número de stream não válido	0663
02	Expressão do dispositivo não válido	062D
03	Nome não válido	064C
04	Número do drive não válido	0681
05	Número do posto não válido	05F6
06	Falta o nome	068D
07	Esquecimento do número do posto	06A1
08	Esquecimento do número do drive	0683
09	Esquecimento da taxa baud	06B7
0A	Erro do cabeçalho inadequado	(nunca usado em ROM)
0B	Stream já aberta	052F
0C	Escrever para um ficheiro de «leitura»	0D78
0D	Lendo num ficheiro de escrita	0D1C
0E	Drive de «escrita» protegida	128D
0F	Microdrive cheio	1219
10	Microdrive não presente	1828
11	Ficheiro não encontrado	11A3
12	Erro de «hook code»	1985
13	Erro CODE	092E
14	Erro MERGE	07D8
15	Falhou a verificação	0930
16	Tipo de ficheiro errado	0902

(As localizações em hexadecimal acima são para onde saltar para produzir cada erro.)

RST 28 — faz um erro da 16K ROM. Carregar ERRNR com a mensagem de código solicitada antes do RST.
RST 30 — cria variáveis da interface, se não existentes.
RST 38 — liga as interrupções (é por isto que o teclado não é examinado).

Rotinas de leitura de linha

Sejam quais forem os comandos que você acrescentar, terá de ler a linha BASIC por causa da sintaxe ou da execução. A sua rotina de comando será chamada duas vezes por cada linha introduzida — uma vez para a verificação da sintaxe, a outra quando é executada. Para dizer qual é, use RST 18. Há uma grande quantidade de rotinas para leitura da linha na velha ROM que podem ser usadas.

Para ler a linha carácter a carácter, use estas rotinas da velha ROM:

GET_CHAR 0018h	— O registo A é o corrente byte na linha.
NEXT_CHAR 0020h	— O registo A é o byte seguinte na linha, ignorando códigos de controle e espaços.

As principais rotinas para avaliação são:

CLASS_061C82h	— Verificar/avaliar parâmetro numérico e pôr valor na pilha do calculador se em tempo de execução, de outro modo inserir na linha bytes invisíveis.
CLASS_0A1C8Ch	— Verificar/avaliar parâmetro de cadeia e pôr valor na pilha do calculador se em tempo de execução.

Na entrada das duas últimas rotinas, CH_ADD deve apontar para o primeiro carácter da expressão. No retorno, aponta para o

primeiro carácter fora do lugar, com A contendo o seu código (por exemplo «,»).

Há algumas rotinas de leitura na ROM sombra que podem ser chamadas directamente.

PARAMS #0701	— A rotina para verificar se uma cadeia e um número estão presentes, isto é os caracteres a seguir a LOAD, SAVE, etc. Todos os parâmetros são passados para D__STR1, N__STR1, S__STR1, L__STR1 e HD__00 para HD__11 durante o tempo de execução.
EVALBC #061E	— Avaliar um número de 16-bits, retornado em BC e em D__STR1.
CHKEND #05B7	— Verifica o fim duma instrução. Se não é o fim, então ocorrerá um erro. Se é o fim, então é feito um RET durante o tempo de execução, de outro modo o endereço de retorno é removido e o controle regressa ao 16K ROM por meio de 05C1.
CHKDRV066D	— Verifica se o conteúdo de D__STR1 está na gama 1-8. Se não está, ocorre o erro *Número do drive não válido*.
CHKST$ #062F	— Avalia uma cadeia — se em tempo de execução é verificada para estar abaixo de 11 caracteres, e os parâmetros armazenados em N__STR1 e T__STR1.

Quando toda a sintaxe tiver sido verificada e considerada correcta deve ser feito um salto para 05C1h (a não ser que seja feito um retorno indirecto por meio de CHKEND). Quando tiver sido executado com sucesso um comando, deve ser feito um salto semelhante. A pilha e o código de erro é limpo e feito um retorno para o 16K ROM.

Como pode ver-se, acrescentar comandos é bastante difícil, mas vale a pena. Abre o Spectrum a todos os tipos de programas utilitários, assim como oferece a possibilidade de utilizar o editor de linha BASIC para introduzir linhas para outras linguagens, ou *assemblers*.

Temos aqui a listagem *assembly* dos comandos extras acrescentados na primeira metade do capítulo. Não é de posição independente ou auto-recolocação — o carregador BASIC altera automaticamente os bytes CALL, à medida que os faz entrar.

```
0010;          Vector para alterar sintaxe
0030           ORG   40000      Origem da depuração de erros
0040           CP    5Ch        Verificação para «#» — 206
0050 ERROR     JP    NZ, 01FOh  Erro se não
0060           RST   10h
0070           DEFW  0020h      Chamar NEXT-CHAR
0080           OR    20h        Fazê-lo minúsculas
0090           CP    «s»
0100           JR    Z,SAVE     Saltar se «s» ou «S»
0110           CP    «l»
0120           JR    Z,LOAD     Saltar se «l» ou «L»
0130           CP    «v»
0140           JR    Z,VERIF    Saltar se «v» ou «V»
0150           CP    «m»
0160           JR    Z,MERGE    Saltar se «m» ou «M»
0170           CP    «c»
0180           JR    Z,CAT      Saltar se «c» ou «C»
0190           CP    «r»
0200           JR    Z,RUN      Saltar se «r» ou «R»
0210           JR    ERROR      De outro modo faz um erro
0220 SAVE      SET   5,(IY+124) Assinalar SAVE
0230           CALL  VARS       Avaliar a cadeia
0240           JP    0836h      Fazer o SAVE
0250 LOAD      SET   4,(IY+124) Assinalar LOAD
0260 DOIT      CALL  VARS       Avaliar a cadeia
0270           JP    08A5h      Fazer LOAD/
                                VERIFY/MERGE
0280 VERIF     SET   7,(IY+124) Assinalar VERIFY
0290           JR    DOIT       Fazê-lo
0300 MERGE     SET   6,(IY+124) Assinalar MERGE
0310           JR    DOIT       Fazê-lo
0320 CAT       LD    HL,5CD6h   HL=D__STRI
```

```
0330          LD    (HL),1      Fazê-lo drive 1
0340          INC   HL
0350          LD    (HL),0      Zero D__STR1 hi
0360          INC   HL
0370          LD    (HL),2      Fazer a stream 2
0380          RST   10h
0390          DEFW  0020h       Chamar NEXT-CHAR
0400          CP    13
0410          JR    Z,CAT2      Saltar se nova linha
0420          CP    «:»
0430          JR    Z,CAT2      Saltar se separador
0440          CALL  061Eh       Avaliar um número
0450 CAT2     JP    04A9h       Fazer o CAT
0460 RUN      RST   10h
0470          DEFW  0020h       Chamar NEXT-CHAR
0480          OR    20h         Fazê-lo em minúsculas
0490          CP    «u»
0500 ERR2     JP    NZ,0028h    Erro se não «u»
0510          RST   10h
0520          DEFW  0020h       Chamar NEXT-CHAR
0530          OR    20h
0540          CP    «n»
0550          JR    NZ,ERR2     Erro se não «n»
0560          RST   10h
0570          DEFW  0020h       Chamar NEXT-CHAR
0580          CALL  05B7h       Chamar CHKEND
0590          JP    0A95h       Carregar o programa «run»
0600 VARS     LD    A, «M»
0610          LD    (5CD9h),A   Fazer L__STR1 = «M»
0620          RST   10h
0630          DEFW  0020h       Chamar NEXT-CHAR
0640          RST   10h
0650          DEFW  1C8Ch       Chamar CLASS__0A (cadeia)
0660          RST   10h
0670          DEFW  0018h       Chamar GET__CHAR
0680          RST   18h         Verificar a sintaxe
0690          JP    Z,0723h     Não executa isto se somente a avaliar a sintaxe
0700          RST   10h
```



```
0710          DEFW  2BF1h       Chamar STK__FETCH (BC = comprimento, DE = arranque)
0720          LD    HL,11
0730          AND   A
0740          SBC   HL,BC
0750          JP    C,064Ch     Erro se mais de 11 caracteres
0760          LD    A,B
0770          OR    C
0780          JP    Z,064Ch     Erro se cadeia vazia
0790          LD    A,(DE)      A = primeiro carácter código
0800          SUB   «0»         A = número do drive
0810          CP    1
0820          JR    C,INVDR     Erro se <1
0830          CP    9
0840          JR    NC,INVDR    Erro se >8
0850          LD    (5CD6h),A   Armazenar o número em D__STR1
0860          XOR   A
0870          LD    (5CD7h),A   Limpar D__STR1 hi
0880          DEC   BC          Diminuir o comprimento
0890          INC   DE          Aumentar posição de arranque
0900          LD    (5CDAh),BC  Armazenar o comprimento
0910          LD    (5CDCh),DE  Armazenar o início
0920          RST   10h
0930          DEFW  0018h       Chamar GET__CHAR
0940          JP    0723h       Verificar argumentos e depois RET
0950 INVDR    RST   20h         Fazer um novo erro
0960          DEFB  4           Byte para «No. do drive não válido»
```

Como se pode ver, saltei para algumas muito obscuras posições da ROM sombra. Estas localizações não devem ser chamadas usando o código *hook* 32h, visto que usam rotinas de leitura de linha e retornam para o BASIC, não para a rotina de chamada em código-máquina.

APÊNDICE A

As variáveis do sistema da interface

Tal como as variáveis do sistema normais, os seguintes 58 bytes são acrescentados ao mapa de memória, com as seguintes funções:

DECIMAL	HEX	NOME	U S O
23734	5CB6	FLAGS3	Bit 0 — *set* enquanto se fizer comando ampliado Bit 1 — *set* enquanto se fizer um CLEAR # Bit 2 — *set* quando ERRSP alterado Bit 3 — *set* quando da formação da rede Bit 4 — *set* quando fazendo LOAD Bit 5 — *set* quando fazendo SAVE Bit 6 — *set* quando fazendo MERGE Bit 7 — *set* quando fazendo VERIFY
23735	5CB7	VECTOR	Usado para ampliar intérprete — normalmente 01F0
23737	5CB9	SBRT	Rotina usada pela ROM som-

			bra para chamar rotinas 16K, da forma: LD HL, valor CALL rotina LD(5CBAh), HL RET
23747	5CC3	BAUD	Taxa baud da RS232 — inicialmente 000Ch, 19200 — aproximadamente (3500000/(26*taxa))—2
23749	5CC5	NTSTAT	Número do posto da rede (1 a 64)
23750	5CC6	IOBORD	Cor da margem durante as operações de entrada/saída I/O
23751	5CC7	SER__FL	Área de trabalho da RS232 — o primeiro byte é uma *flag*, o segundo um carácter entrado
23753	5CC9	SECTOR	Espaço de trabalho do microdrive — normalmente usado para contar sectores, começando de FFh ou 04FBh
23755	5CCB	CHADD—	Armazenamento temporário para CH__ADD enquanto se verificar sintaxe ampliada das linhas
23757	5CCD	NTRESP	Código de resposta da rede —recebendo números de postos— o reconhecimento início de 8 bytes cabeçalho da rede
23758	5CCE	NTDEST	Número do posto de destino da rede
23759	5CCF	NTSRCE	Número do posto de origem da rede
23760	5CD0	NTNUMB	Número do bloco da rede (0 a 65535)
23762	5CD2	NTTYPE	Código do tipo de cabeçalho da rede — 0 dados, 1 EOF

23763	5CD3	NTLEN	Comprimento do bloco de dados da rede (0 a 255)
23764	5CD4	NTDCS	Verificação de soma do bloco de dados da rede
23765	5CD5	NTHCS	Verificação de soma do cabeçalho da rede (de NTDEST-NTDCS)
23766	5CD6	D__STR1	Início dos primeiros 8 bytes do especificador do ficheiro: 2 bytes do número do drive (1 a 8) ou número de destino da rede ou taxa baud
23768	5CD8	S__STR1	Número da stream (0 a 15)
23769	5CD9	L__STR1	Especificador do canal (em maiúsculas)
23770	5CDA	N__STR1	Comprimento do nome do ficheiro
23772	5CDC	T__STR1	Início do nome do ficheiro (normalmente no espaço de trabalho)
23774	5CDE	D__STR2	Início dos segundos 8 bytes do especificador do ficheiro
23782	5CE6	HD__00	Tipo ficheiro: 0 — programa 1 — quadro numérico 2 — quadro alfanumérico 3 — bytes
23783	5CE7	HD__0B	Comprimento dos dados
23785	5CE9	HD__0D	Início dos dados
23787	5CEB	HD__0F	Comprimento do programa (ou nome do quadro)
23789	5CED	HD__11	Número da linha do arranque automático (também usado pelo *hook code* 32h)
23791	5CEF	COPIES	Número de cópias múltiplas feitas por SAVE — *reset* para 1 depois de SAVE

Notas: FLAGS3 é normalmente levado a zero enquanto a ROM sombra despagina. Pode ser utilmente endereçado por IY + 124.

Os parâmetros na sub-rotina SBRT são colocados pela ROM sombra. O RETorno vai para o endereço 8, mas há sempre uma entrada 0000 seguinte na pilha para o diferenciar de um erro.

Os nomes de variáveis HD_00 a HD_11 são directamente equivalentes aos usados pelas rotinas de cassete da ROM 16K, endereçados por IX + 0 a IX + 11h.

APÊNDICE B

Listagens Assembly

Este apêndice contém listagens Assembler Z80 das rotinas usadas neste livro. São baseadas nos originais criados com «Editor assembler» da Picturesque, o qual indica os números hexadecimais seguindo-os com um «h». Os números de linhas são usados pelo assembler, e não têm significado aqui. No lado direito foram acrescentados comentários. Todos os ORG são arbitrários, pois que a maior parte das rotinas ou são de posição independente ou se colocam a si próprias. Aquelas que se autorrecolocam usam a facilidade de, na entrada para elas, o registo BBC conter os seus endereços de início.

«Stream 14-z$»

Esta cria um novo canal «Z», que liga à stream 14, a qual se supõe estar já fechada. Também se parte do princípio de que as 58 variáveis extra sistema já tenham sido criadas. Isto porque o comando CLOSE # está incluído no carregador BASIC; deste modo, ele faz ambas estas tarefas.

```
0010        ;           «Rotina de impressão para
                        cadeia»
0030        ;           «Rotina para fazer #14»
0040        ;           «Inserir caracteres em z$»
0050        ;           «© A. Pennel 1983»
0060        PROG EQU    5C53h
```

```
0070        VARS  EQU   5C4Bh
0080              ORG   40000
0090        SETUP LD    HL,(PROG)
0100              DEC   HL          HL = PROG—1 = fim de
                                    CHANS
0110              PUSH  BC          Guardar o endereço de iní-
                                    cio
0120              PUSH  HL          Guardar início área novos
                                    dados
0130              LD    BC,11       Preciso 11 bytes
0140              CALL  1655h       Chamar MAKE_ROOM
0150              POP   DE          DE = início novos dados
0160              LD    HL, OUTCH-SETUP
0170              POP   BC          Recuperar SETUP
0180              ADD   HL,BC       HL = OUTCH
0200              PUSH  DE          Guardar o início de dados
0210              EX    DE,HL
0220              LD    (HL),E      Armazenar OUTCH em no-
                                    va área
0230              INC   HL
0240              LD    (HL),D
0250              INC   HL
0260              EX    DE,HL
0270              LD    BC,LABEL-OUTCH
0280              ADD   HL,BC       HL = LABEL
0290              LD    BC,9        Mais 9 bytes de dados
0300              LDIR              Copiar os dados para
                                    CHANS
0310              POP   HL          HL = início de dados
0320              INC   HL
0330              LD    BC,(5C4Fh)  BC = CHANS
0340              AND   A
0350              SBC   HL,BC
0360              LD    (5C32h),HL  Armazenar o deslocamento
                                    STRMS em STRMS para
                                    #14
0370              LD    BC,0
0380              RET               Regressar para BASIC com 0

                                    Os restantes dados para
                                    irem CHANS:
```



```
0390 LABEL DEFW  15C4h    A rotina de entrada
0400       DEFM  «Z»      O nome do canal
0410       DEFW  28h      «ROM sombra o/p»
0420       DEFW  28h      «ROM sombra i/p»
0430       DEFW  11       O número de bytes
                          A rotina de saída:

0440 OUTCH PUSH  AF       Guardar o código de carác-
                          ter
0450       LD    HL,(VARS)
0460 L1    LD    A,(HL!)
0470       CP    5Ah
0480       JR    Z,FOUND  Saltar se encontrado Z$
0490       CP    80h
0500       JP    Z,0670h  «Var não encontrada» se
                          não mais
0510       CALL  19B8h    Chamar NEXT_ONE
0520       EX    DE,HL    HL = início da variável se-
                          guinte
0530       JR    L1       Anel circular

0540 FOUND INC   HL       HL = comprimento cadeia lo

0550       LD    C,(HL)
0560       INC   HL       HL = comprimento cadeia
                          hi
0570       LD    B,(HL)   BC = comprimento cadeia
0580       INC   BC       Aumentar comprimento
0590       PUSH  BC       Guardar o novo compri-
                          mento
0600       PUSH  HL       Guardar posição 1.° caracter
0610       ADD   HL,BC    HL = fim da cadeia
0620       CALL  1652h    Chamar ONE_SPACE (fa-
                          zer espaço para carácter)
0630       INC   HL       HL = novo espaço
0640       EX    DE,HL    DE = novo espaço
0650       POP   HL       HL = Posição 1.° carácter
0660       POP   BC       BC = novo comprimento
```

```
0670        LD    (HL),B     Armazenar novo comprimento
0680        DEC   HL
0690        LD    (HL),C
0700        POP   AF         Voltar obter o código de carácter
0710        LD    (DE),A     Armazenar o carácter na cadeia
0720        AND   A          Limpar carry (para evitar erros)
0730        RET              Voltar ao O/S
0740        END
```

ON EOF GOTO

Esta rotina altera ERRSP para apontar para uma adequada rotina para manipulação de erro.

```
0010;               «ON EOF goto»
0020;               «PIC code»
0040;       ORG     40000
0050 SETUP  LD      HL,START-SETUP Alter. ERRSP:
0060        ADD     HL,BC          HL = START
0070        EX      DE,HL          DE = START
0080        LD      HL,(ERRSP)     HL = ERRSP
0090        LD      (HL),E         Armazenar a nova rotina de erro na pilha adequada
0100        INC     HL
0110        LD      (HL),D         Posição
0120        RST     8              Assegurar-se de que as variáveis interface estão ali
0130        DEFB    31h
0140        LD      BC,0
0150        RET                    Volta ao BASIC com 0
                                   Manipulação de erro:
0160 START  LD      HL,(ERRSP)     HL = posição da pilha
```



```
0170        LD    A,(5C3Ah)    A = código de erro
0180        CP    EOF          Era um EOF?
0190        JP    NZ,1303h     Saltar para rotina ROM se não era
0200        LD    E,(HL)       De outro modo
                                DE = START
0210        INC   HL
0220        LD    D,(HL)
0230        PUSH  DE           Pôr START no fundo da pilha
0240        CALL  16B0h        Limpar espaço de trabalho e áreas de edição
0250        RES   5,(IY = 37h) Sinal «pronto para nova tecla»
0260        CALL  0D6Eh        Limpar a parte inferior do écran e abrir stream 0
0270        LD    HL,(5C45h)   HL = número de linha corrente
0280        LD    (5CC9h),HL   Armazená-lo em SECTOR
0290        LD    DE,LINE      DE = linha para que saltar
0300        LD    HL,5C42h     HL = NEWPPC
0310        LD    (HL),E       Armazenar o novo número de linha
0320        INC   HL
0330        LD    (HL),D
0340        INC   HL
0350        LD    (HL),1       Fazer NSPPC = 1, isto é, 1.ª instrução saltar para A ROM
0370        JP    1B7Dh
0380 ERRSP  EQU   5C3Dh
0390 EOF    EQU   7            Código de mensagem eof
0400 LINE   EQU   1000         Erro número de linha
0410        END
```

Rotina «OPEN # qualquer coisa»

Esta rotina abre uma stream especificada para um canal do microdrive, mas o tipo de ficheiro não se restringe a ficheiros de dados. Ele é, basicamente, a rotina do comando OPEN # com a verificação de erro modificada.

```
0010;                «OPEN qualquer coisa»
0030;
0040;
0050 OPENM EQU  22h          Hook codes
0060 CLOSM EQU  23h
0070 MOTOR EQU  21h
0080       ORG  40000
0090 START LD   A,(5CD8h)    A = S_STR1 = número da
                             stream
0100       CALL 1727h        Chamar STR_DATA1
0110       LD   HL,17
0120       XOR  A
0130       SBC  HL,BC
0140       LD   BC,0
0150       RET  C            Voltar para BASIC com 0
                             se a stream já está aberta
0160       LD   (5CD7h),A    Zero D_STR1 hi
0170       LD   HL,10
0180       LD   (5CDAh),HL   Fazer comprimento nome
                             do ficheiro = 10
0190       LD   HL,(5C7Bh)
0200       LD   (5CDCh),HL   Fazer início nome = área
                             de gráficos do utilizador
0210       LD   A,(5CD8h)    A = número da stream
0220       ADD  A            Calcular o STRM adequa-
                             do
0230       LD   HL,5C16h
0240       LD   E,A
0250       LD   D,0
0260       ADD  HL,DE        HL = posição da stream
```



```
0270       EXX
0280       PUSH HL           Guardar H'L'
0290       EXX
0300       PUSH HL           Guardar a posição de
                             STRM
0310       RST  8            Abrir um canal «M» tem-
                             porário
0320       DEFB OPENM
0330       BIT  0,(IX + 24)
0340       JR   Z,READ       Saltar se é um ficheiro de
                             leitura
0350       XOR  A            Ficheiro não encontrado,
                             assim todos os motores
                             desligados
0360       RST  8
0370       DEFB MOTOR
0380       RST  8            e reclamar a área
0390       DEFB 2Ch
0400       POP  HL           Recarregar a pilha
0410       EXX
0420       POP  HL           Recuperar H «L»
0430       EXX
0440       LD   BC,1
0450       RET               Retorno com 1 se não en-
                             contrado
0460 READ  RES  7,(IX + 4)   Fazê-lo permanente
0470       XOR  A
0480       PUSH HL           Guardar o deslocamento da
                             stream
0490       RST  8            Desligar todos os motores
0500       DEFB MOTOR
0510       POP  DE           DE = deslocamento
0520       POP  HL           HL = posição
0530       LD   (HL),E       Armazenar o novo desloca-
                             mento na área STRMS
0540       INC  HL
0550       LD   (HL),D
0560       LD   A,(IX + 67)  A = RECFLG
0570       AND  4            Mascarar o bit do PRINT
                             ficheiro
```

```
0580            ADD   2          Adicionar 2
0590            LD    C,A        Pôr em BC
0600            LD    B,0
0610            EXX
0620            POP   HL         Recuperar H'L'
0630            EXX
0640            RET              Retorno com 2 se um fi-
                                 cheiro PRINT ou 6 se não
0650            END
```

Rotina «Status»

Esta rotina é semelhante à rotina MOTOR na ROM sombra, da posição X182Ah até X1871h, mas com a manipulação do erro modificada. O seu fim é determinar se um dado microdrive está ou não presente, e se o cartucho nele contido é de escrita protegida. As etiquetas são baseadas nas posições equivalente da ROM sombra. O ORG é uma origem depuradora de erros — o código *não* é de posição independente — o carregador BASIC faz a recolocação dos quatro CALL.

(Não posso reivindicar que sei exactamente o que cada instrução *port* I/0 faz — as notas aqui são conjecturas informadas!)

```
0010;                STATUS routine
0020 MOTOR EQU   21h          hook code do motor
0030       ORG   40000        Origem da depuração
0040       LD    A,(5CD6h)    A = D_STR1 = número do
                                drive
0550       DI                 Interrupções desligadas
0060       JR    X182A        Saltar por cima disto
0070 X1806 LD    HL,1388h     HL = 5000
0080 X1809 DEC   HL
0090       LD    A,L
0100       OR    H
0110       JR    NZ,X1809     Esperar um momento
0120       LD    HL,1388h
0130 X1811 LD    B,6
0140 X1813 IN    A,(0EFh)     Ler port EF
```



```
0150       AND   4            Somente bit 2
0160       JR    NZ,X1820     Saltar se o drive não pre-
                                sente
0170       DJNZ  X1813        Assegurar-se ali 6 vezes
0180       JR    PRESN        Neste caso retorno
0190 X1820 DEC   HL           Diminuir o contador
0200       LD    A,H
0210       OR    L
0220       JR    NZ,X1811     Experimentá-lo 5000 vezes
0230       LD    BC,0         Não pode ser ligado assim
0240       JR    NOTPR        Retorno com 0
0250 X182A LD    DE,0100h     D = 1,E = 0
0260       NEG                Neg. número do drive
0270       ADD   9            A = 9 – número do drive
0280       LD    C,A          C = drive seleccionando
0290       LD    B,8          B = contador do drive
0300 X1835 DEC   C            Diminuir o drive selecio-
                                nado
0310       JR    NZ,X184B     Saltar se não o que está a
                                ser investigado
                              Pôr um motor ON
                                (ligado):
0320       LD    A,D
0330       LD    (0F7h),A     Enviar 1 para port F7 —
                                ON
0340       LD    A,0EEh
0350       OUT   (0EFh),A     Enviar EE para port EF
0360       CALL  X1867        Esperar um momento
0370       LD    A,0ECh
0380       OUT   (0EFh),A     Enviar EC para port EF
0390       CALL  X1867        Esperar um momento
0400       JR    X185C        Fazer o seguinte
                              Desligar (OFF) um drive
0410 X184B LD    A,0EFh
0420       OUT   (0EFh),A     Enviar EF para port EF
0430       LD    A,E
0440       OUT   (0F7h),A     Enviar 0 para port F7 —
                                OFF
0450       CALL  X1867        Esperar um momento
0460       LD    A,0EDh
```

```
0470          OUT   (0EFh),A    Enviar ED para port EF
0480          CALL  X1867       Esperar um momento
0490 X185C    DJNZ  X1835       Fazer todos os 8 drives
0500          LD    A,D
0510          OUT   (0F7h),A    Enviar 1 para port F7
0520          LD    A,0EEh
0530          OUT   (0EFh),A    Enviar EE para port EF
0540          JR    X1806       Testar o seu estado
0550 X1867    PUSH  BC          Rotina de abrandamento:
0560          PUSH  AF          guardar regs
0570          LD    BC,0087h    BC = 135
0580 X18FB    DEC   BC
0590          LD    A,B
0600          OR    C
0610          JR    NZ,X18FB    Repetir 135 vezes
0620          POP   AF          Rearmazenar regs
0630          POP   BC
0640          RET               Retorno
0650 PRESN    IN    A,(0EFh)    O drive está presente:
0660          AND   1           teste a patilha de protecção
                                da escrita
0670          LD    BC,1
0680          JR    NZ,NOTPR    Saltar se patilha está aí
                                com 1
0690          INC   BC          ou com 2 se não está
0700 NOTPR    PUSH  BC          Guardar o valor de retorno
0710          XOR   A
0720          RST   8
0730          DEFB  MOTOR       Desligar todos os motores
0740          POP   BC          Rearmazenar o valor
0750          RET               Voltar ao BASIC
0760          END
```

«ON ERROR GO TO»

Esta rotina altera ERR_SP para indicar uma nova rotina de erro. Isto só por si não é suficiente para os erros da interface — para eles, o bit 2 de FLAGS3 deve igualmente ser *set*, de outro modo será usada a rotina de erro normal. Se o erro ocorreu enquanto o ROM sombra estava paginada, HL igualará 81h.

```
0010;         «ON ERR GO TO»
0020;         Versão interface
0030 ERRSP    EQU   5C3Dh       Estabelecer constantes
0040 SECTR    EQU   5CC9h       Salto erro de linha
0050 LINE     EQU   1000
0060          ORG   40000       Origem arbitrária
0070 SETUP    LD    HL,START-SETUP
0080          ADD   HL,BC       HL = START
0090          EX    DE,HL       DE = START
0100          LD    HL,(ERRSP)  HL = erro posição da pilha
0110          LD    (HL),E      Armazenar a nova rotina
                                de erro na pilha
0120          INC   HL
0130          LD    (HL),D
0140          RST   8
0150          DEFB  31h         Criar as variáveis extra
0160          LD    BC,0
0170          RET               Retorno com 0
                                O novo erro de manipula-
                                ção:
0180 START    LD    (SECTR),HL  Armazenar o valor de HL
0190          LD    HL,(ERRSP)
0200          LD    DE,1303h    DE = manipulação da velha
                                ROM
0210          PUSH  DE          Colocar isto na pilha para
                                desligar a função next time
0220          CALL  16B0h       Limpar as várias áreas
0230          RES   5,(IY + 37h) Limpar a tecla bit em
                                FLAGS
0240          CALL  0D6Eh       Limpar parte inferior do
                                écran e abrir a stream 0
0250          LD    HL,17B9h    HL = rotina sombra para
                                reclamar todos os canais
                                temporários e desligar to-
                                dos os motores
0260          LD    (5CEDh),HL  Armazenar em HD_11
```

```
0270        LD    A,(5C3Ah)    A = número de erro (velho
                                erro)
0280        LD    (23763),A    Armazenar em NTLEN
0290        PUSH  AF           Guardar código de erro
0300        RST   8
0310        DEFB  32h          Chamar X17B9h
0320        LD    HL,0081h
0330        LD    DE,(SECTR)   DE = valor de HL depois
                                de erro
0340        POP   AF           A = código de erro da ve-
                                lha ROM
0350        AND   A            Limpar carry
0360        SBC   HL,DE
0370        JR    NZ,OLD       Saltar se HL na entrada
                                <> 81h, isto é, um velho
                                erro ROM
0380        BIT   0,(IY + 124) Testar FLAGS3
0390        JR    NZ,OLD       Para um velho erro no
                                meio duma instrução
0400        CP    7
0410        JR    Z,OLD        Saltar se não era um EOF
0420        LD    A,100        Erro de interface, assim ar-
                                mazenar 100 em NTLEN
0430        LD    (23763),A
0440        LD    A,«?»
0450        RST   10h          Imprimir um «?»
0460        JR    PRNTN        Então imprimir números de
                                linha, etc.
                               Erro da velha ROM:
0470        INC   A
0480        LD    B,A          Armazenar o número de
                                erro em B
0490        CP    10
0500        JR    C,2
0510        ADD   7            Formar o carácter de aviso
0520        CALL  15EFh        Imprimi-lo
0530        LD    A, « »
0540        RST   10h          Imprimir um espaço
0550        LD    A,B
```



```
0560        LD    DE,1391h     Imprimir mensagem de er-
                                ro
0570        CALL  0C0Ah
0580 PRNTN  XOR   A
0590        LD    DE,1536h
0600        CALL  0C0Ah        Imprimir «,»
0610        LD    BC,(5C45h)   BC = PCP = linha corrente
0620        LD    (SECTR),BC   Armazenar em SECTOR
0630        CALL  1A1Bh        Imprimir o número de li-
                                nha
0640        LD    A,«:»
0650        RST   10h          Imprimir uma vírgula
0660        LD    C,(IY + 13)
0670        LD    B,0
0680        CALL  1A1Bh        Imprimir o número da ins-
                                trução
0690        LD    HL,5C3Bh     HL = FLAGS
0700        RES   5,(HL)
0710        EI                 Interrupções ligadas
0720 WAIT   BIT   5,(HL)
0730        JR    Z,WAIT       Esperar que uma tecla seja
                                premida
0740        LD    HL,5C42h     HL = NEWPPC
0750        LD    DE,LINE      DE = linha de erro para
                                salto para armazenar a li-
                                nha
0760        LD    (HL),E
0770        INC   HL
0780        LD    (HL),D
0790        INC   HL
0800        LD    (HL),1       Fazê-lo instrução 1
0810        LD    (IY + 0),255 Limpar o erro
0820        LD    (IY + 124),0 Limpar FLAGS3
0830        CALL  0D6Eh        Limpar a parte inferior do
                                écran
0840        JP    1B7Dh        Saltar para a ROM 16K
0850        END
```

Rotina «RS232 TAB»

Esta rotina altera os dados do canal «P» de modo que LPRINT, etc., serão enviados para o *port* da RS232, semelhante ao canal «T», mas com a função TAB implementada. Usa três das suas próprias variáveis de sistema — WIDTH = largura da cabeça da impressora, POSN = posição corrente da cabeça de impressão, CONTR = uma *flag*. Quando TAB é interpretado, primeiramente é enviado um carácter 23, e então o LSB do número seguinte, o MSB.

```
0010;       «Rotina RS232 TAB«
0020        ORG   23296          Localizar no buffer da im-
                                 pressora
0030 SETUP  LD    HL,(23631)     HL = CHANS
0040        LD    BC,15
0050        ADD   HL,BC          HL = área do canal «P»
0060        LD    DE,START       DE = nova rotina de saída
0070        LD    (HL),E         Armazenar a posição da no-
                                 va rotina na área «P»
0080        INC   HL
0090        LD    (HL),D
0100        LD    BC,0
0110        LD    HL,POSN
0120        LD    (HL),B         Zero POSN
0130        INC   HL
0140        LD    (HL),B         Zero CONTR
0150        RET                  Retorno com 0
                                 A rotina O/P:A = código
0160 START  CP    « »
0170        JR    NC,NORM        Saltar se > = « »
0180        CP    13
0190        JR    NZ,NNL         Saltar se não nova linha
0200        LD    HL,CONTR       Nova linha:
0210        BIT   0,(HL)         Testar a auto flag
0220        RES   0,(HL)         Limpar a auto flag
0230        RET   NZ             Não fazer uma N/L se set
0240 NEWLI  LD    HL,CONTR
0250        RES   0,(HL)         Limpar a auto flag
0260        DEC   HL             HL = POSN
```



```
0270        LD    (HL),0         Zero POSN
0280        LD    A,13
0290        CALL  OUTCH          Enviar uma N/L(13)
0300        LD    A,10
0310        JP    OUTCH          Enviar uma line-feed (10)
0320 NNL    CP    23             É o carácter TAB?
0330        CCF
0340        RET   NZ             Retorno se não é
0350        LD    DE,TAB2        Comando TAB:
0360 REDO   LD    HL,(5C51h)     HL = CURCHL
0370        LD    (HL),E         Armazenar DE em
                                 CURCHL, isto é, alterar a
                                 posição O/P para DE
0380        INC   HL
0390        LD    (HL),D
0400        RET                  Retorno
                                 Chegar aqui com LSB da
                                 posição TAB:
0410 TAB2   LD    (5C0Fh),A      Armazenar o LSB em
                                 TVDATA2
0420        LD    DE,TAB3
0430        JR    REDO           Fazer a rotina O/P TAB3
                                 Chegar aqui com MSB da
                                 posição TAB:
0440 TAB3   LD    DE,START
0450        CALL  REDO           Rearmazenar O/P para nor-
                                 mal
0460        LD    A,(5C0Fh)      A = posição TAB
0470        LD    D,A            Armazenar em D
0480 TAB4   LD    HL,WIDTH
0490        SUB   (HL)           A = TAB — WIDTH
0500        JP    NC,046Ch       «Int out of range» se o car-
                                 ro não está suficientemente
                                 afastado
0510        INC   HL             HL = POSN
0520        LD    A,D            A = TAB
0530        SUB   (HL)           A = TAB—POSN
0540        PUSH  DE             Guardar D
0550        CALL  C,NEWLI        Fazer uma nova linha se não
                                 se adaptar
```

```
0560          POP   DE          Rearmazenar D
0570          LD    A,D         A = TAB
0580          SUB   (IY+POSY)   A = TAB—POSN
0590          RET   Z           Retorno se no lugar certo
0600          LD    B,A         B = número de espaços ne-
                                cessários
0610 TABB     LD    A,« »
0620          PUSH  BC          Guardar BC
0630          EXX               Trocar regs
0640          RST   10h         Imprimir um espaço
0650          EXX               Voltar a trocar
0660          POP   BC          Rearmazenar BC
0670          DJNZ  TABB        Imprimir os espaços apro-
                                priados
0680          RET               Retorno quando feito
                                Vir aqui com códigos «nor-
                                mais»
0690 NORM     CP    0A5h
0700          JR    C,NORM2     Saltar se não um símbolo
0710          SUB   0A5h
0720          JP    0C10h       Dessimbolizá-lo
0730
NORM2         RES   0,(IY+CONY) Limpar a auto flag
0740          LD    HL,5C3Bh    HL = FLAGS
0750          RES   0,(HL)      Limpar o «espaço precede-
                                nte» flag (que a ROM não
                                faz)
0760          CP    « »
0770          JR    NZ,NSPAC    Deixá-lo se não um espaço
0780          SET   0,(HL)      De outro modo set a flag
0790 NSPAC    CP    128
0800          JR    C,OUTC2     Imprimir o carácter se não
                                gráficos
0810          LD    A,«?»       Caracteres gráficos, então
                                imprimir «?»
0820 OUTC2    CALL  OUTCH       Enviar o byte para a impres-
                                sora
0830          LD    HL,POSN
0840          INC   (HL)        Aumentar POSN
0850          LD    A,(HL)      A = novo POSN
```



```
0860          DEC   HL          HL = WIDTH
0870          CP    (HL)        Comparar com WIDTH
0880          RET   NZ          Retorno se não o mesmo
0890          CALL  NEWLI       Ao fim da linha, assim fazer
                                uma NL
0900          SET   0,(IY+CONY) e set a auto flag
0910          RET               Retorno
                                Enviar um carácter para a
                                RS232
0920
OUTCH         RST   8           Hook code
0930          DEFB  1Eh         Byte 2320P
0940          RET               Retorno
                                Constantes:
0950
WIDTH         EQU   23540       No buffer da impressora
0960 POSN     EQU   WIDTH+1
0970 CONTR    EQU   WIDTH+2
0980 POSY     EQU   0BBh        Deslocamentos IY
0990 CONY     EQU   0BCh
1000          END
```

APÊNDICE C

«Bugs» da interface

Com a interface ligada, é usada uma ROM adicional de 8K para as novas características. Isto está de maneira geral muito bem escrito, se bem que contenha alguns erros (ou «*bugs*»).

1. Falha na verificação da sintaxe

Quando foi escrita a original 16K BASIC ROM, a sintaxe dos comandos da interface foi incorrectamente antecipada. Infelizmente esta sintaxe incorrecta é ainda aceite pelo verificador, mas não se executará. Estas formas incorrectas da sintaxe são:

ERASE <*cadeia*>	por exemplo, ERASE «teste»
MOVE <*cadeia*>,<*cadeia*>,	por exemplo, MOVE «a», «b»
FORMAT <*cadeia*>	por exemplo, FORMAT «teste»
CAT (sem número)	

2. Comandos de cor

Se bem que não seja estritamente um erro causado pela ROM da interface, o problema explicado no cap. 3, respeitante aos comandos de cor, é causado por um pedaço descuidado de código no 16K BASIC. Para o sanar, faça preceder cada comando de cor permanente por PRINT;.

3. Inconsistência do Break

Parece haver uma inconsistência geral sobre o método de interromper as operações da interface. Durante as operações do microdrive e a saída da RS232, CAPS SHIFT e SPACE devem ambas estar premidas; mas com a rede a trabalhar e a entrada da RS232 é suficiente estar premida a tecla SPACE.

4. Operações da cassete

Se um nome de ficheiro numa operação de cassete não é válido sob qualquer aspecto (como, por exemplo, mais comprido do que dez caracteres), o erro que deve ocorrer (*Invalid filename*—Nome de ficheiro não válido) é sobreposto com o inútil *Nonsense in Basic* (Disparate em BASIC).

5. *Hook codes* do código-máquina

Se bem que a grande maioria da ROM sombra esteja bem escrita, os *hook codes* parece terem sido acrescentados como uma apressada reflexão. Há duas faltas neles — o *hook code* READ_N não é utilizável, pois que a *carry flag* está corrompida no fim da rotina, e o *hook code* 2B é o mesmo que 22 — um desperdício dum *hook code* causado por uma entrada incorrecta na tabela de salto de *hook code* na posição X19C9h.

6. Duplos espaços da RS232

Quando se usar um canal «t» para listar programas, é impresso um duplo espaço entre as palavras-chave, por exemplo, PRINT PAPER ou THEN PRINT. Ele é causado pela falha em fazer qualquer coisa com o bit 0 de FLAGS.

7. Problema de close

Se você fizer *Break* no meio duma instrução CLOSE# referente a um canal tipo interface, a memória usada pela stream não é reclamada do mapa de memória. Isto pode usar grandes quantidades de memória, e não é com facilidade que lhe podemos fazer face. Uma declaração CLEAR# não a reclamará, somente uma NEW. Podia ser corrigido juntando a instrução SET 7,(HL) por volta da posição X1741h.

Bibliografia

The Working Spectrum, David Lawrence, (Sunshine Books), publicado nesta colecção com o título *O Spectrum Funcional,* n.º 97.
Programming the Z80, Rodney Zaks (Sybex Inc.)
ZX Spectrum BASIC Programming, Steven Vickers e Robin Bradbeer (Sinclair Research Ltd.).
ZX Microdrive and Interface 1 manual (Sinclair Research Ltd).
The Complete Spectrum ROM Disassembly, Dr. Ian Logan e Dr. Frank O'Hara (Melbourne House Ltd.).

