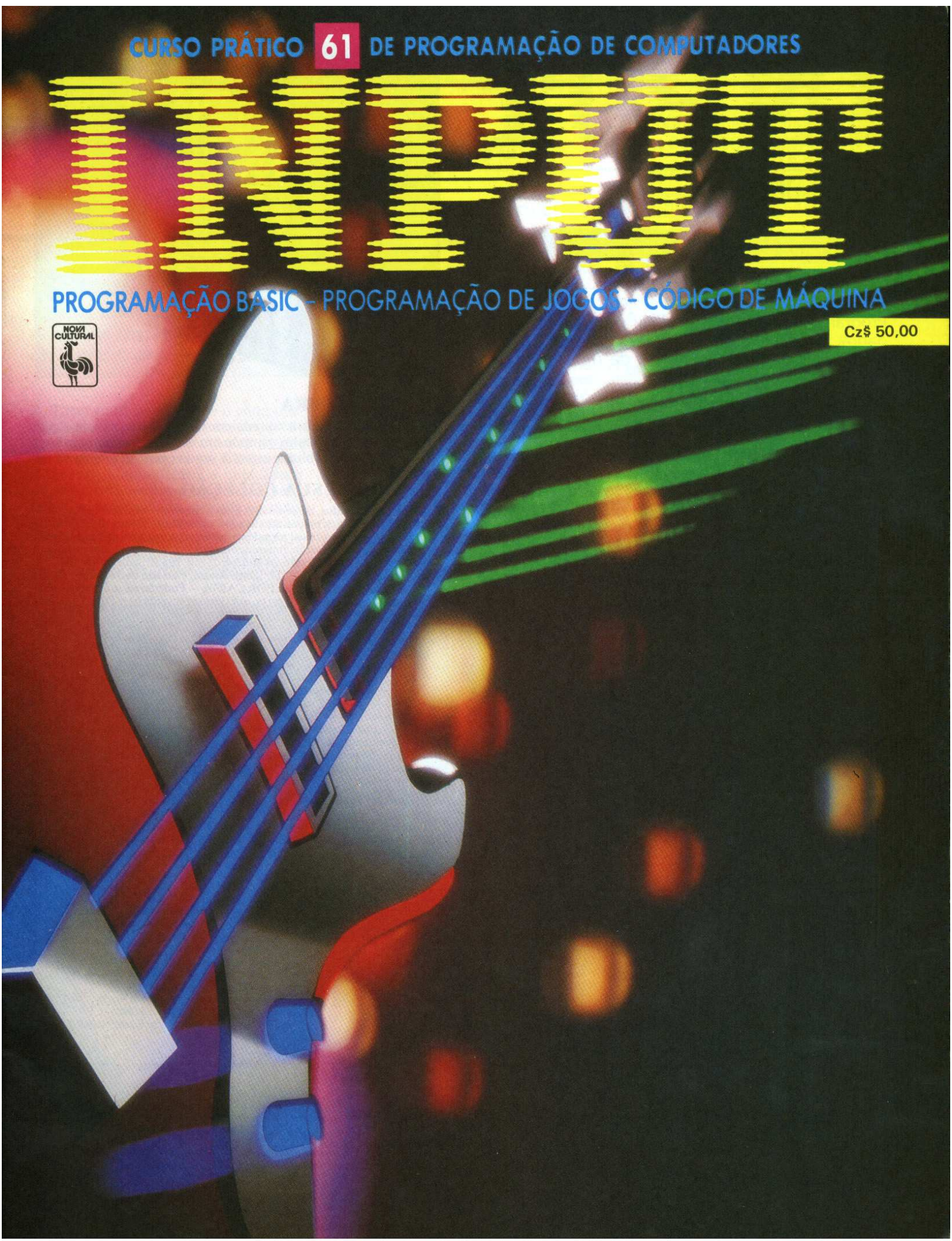
CURSO PRÁTICO 61 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

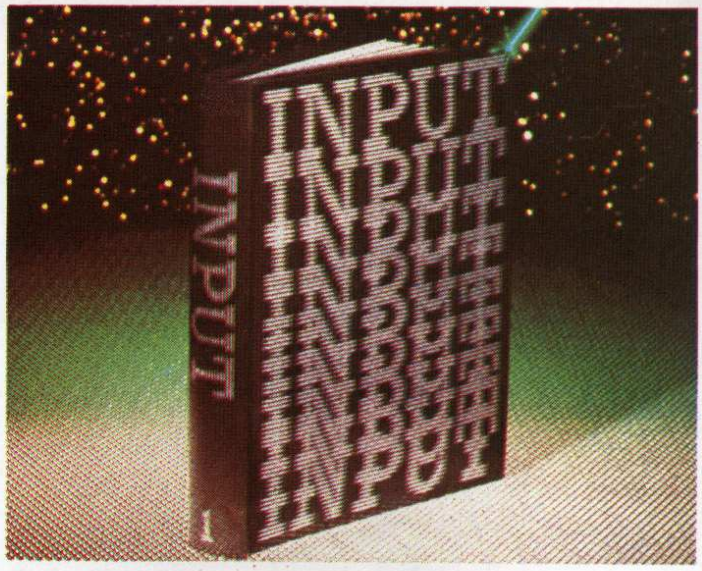
Cz$ 50,00



INPUT

Vol. 5 N.º 61

NESTE NÚMERO



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o nº (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

Atenção: Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

Obs.: Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO
Diretor Editorial: Carmo Chagas

Editores Executivos: Antonio José Filho, Berta Sztark Amar

Editor Chefe: Paulo de Almeida
Editora de Texto: Ana Lúcia B. de Lucena
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Dagmar Bastos Sampaio, Grace Alonso Arruda, Monica Lenardon Corradi
Secretária de Redação/ Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES

Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda., Campinas, SP
Tradução, adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Marcos Huascar Velasco, Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atílio Roberto Bonon

Preparador de Texto/Coordenador: Eliel Silveira Cunha
Preparadores de Texto: Alzira Moreira Braz, Ana Maria Dilguerian, Levon Yacubian, Luciano Tasca, Maria Teresa Galluzzi, Maria Teresa Martins Lopes, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria de Fátima Cardoso, Nair Lucia de Britto
Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, nº 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

COMPRESSÃO DE MELODIAS

- PROGRAMAÇÃO DE UMA MELODIA
- COMO COMPRIMIR OS DADOS
- DIVISÃO DA MÚSICA
- ECONOMIA DE NOTAS
- MODIFICAÇÃO DO ANDAMENTO

Amplie a capacidade musical de seu micro reduzindo os dados necessários à execução de uma melodia.
As técnicas apresentadas aplicam-se à compressão de diferentes tipos de dados.

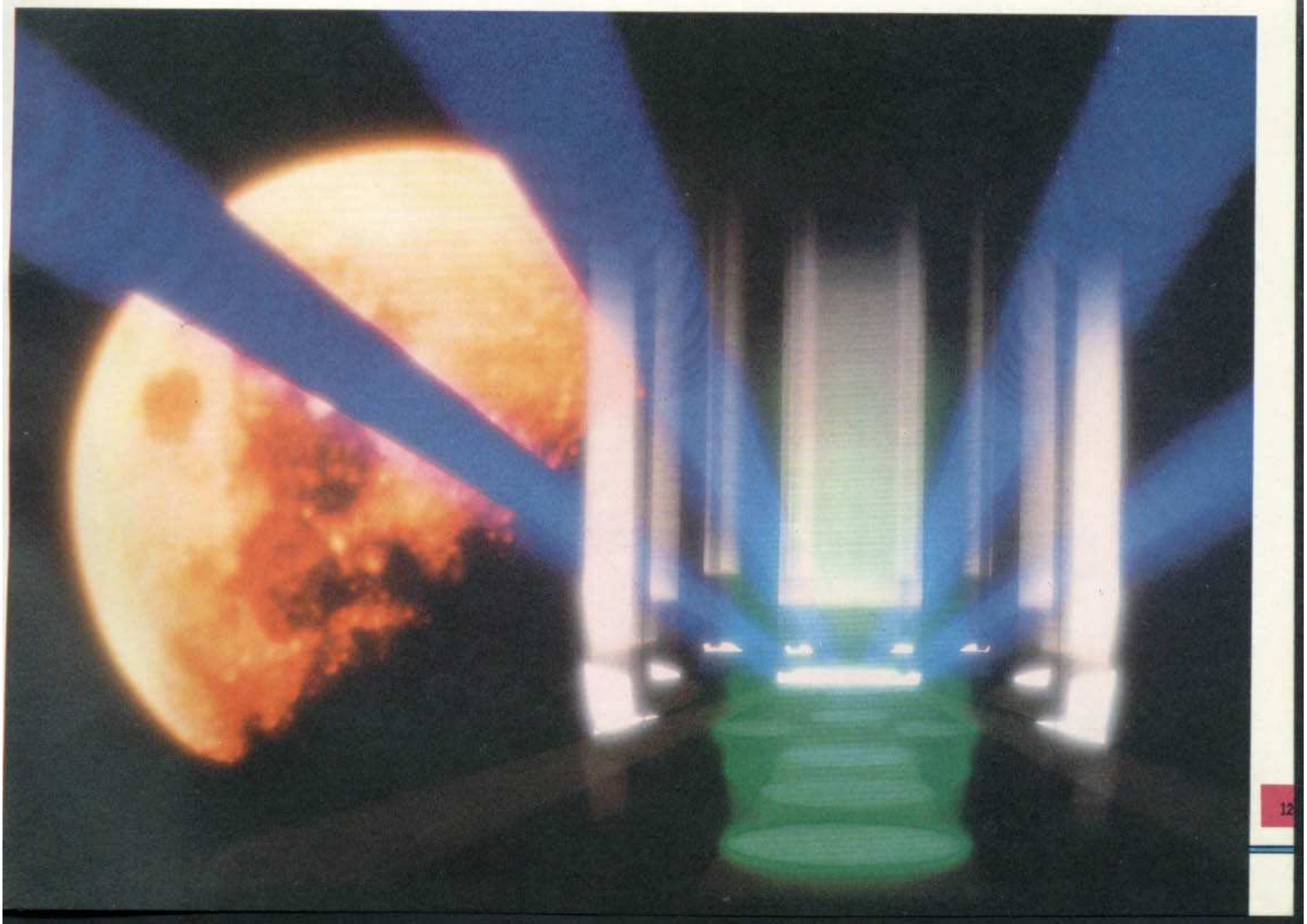
Tocar música utilizando o computador é uma experiência fascinante, principalmente quando se é o autor da peça. Existem, porém, várias dificuldades a superar, e uma delas é a quantidade de dados que um programa requer para executar determinada melodia — mais de uma tela cheia de números, na maioria dos casos. Além de serem difíceis de digitar, esses dados ocupam muito espaço na memória. Neste artigo, você verá como comprimir os dados de suas melodias dentro dos programas BASIC, de maneira que eles tomem o menor espaço possível, liberando parte da memória RAM para outro uso.

A aplicação das técnicas de compressão de dados não se restringe, naturalmente, aos programas destinados à execução de música. Estende-se a dados de todo tipo, desde que sejam repetitivos e limitados a certos valores.

UMA NOVA MELODIA

Seja qual for seu estilo, as melodias que o computador toca seguem um certo padrão que permite sua compressão. Tomemos como exemplo uma música simples, com um total de doze compassos. Para executá-la, precisaremos escrever um programa em BASIC contendo um determinado número de linhas **DATA**, como o apresentado a seguir:

```
 10 LET T=.2
 20 RESTORE 100
 30 READ D
 50 IF D=255 THEN  GOTO 20
 60 SOUND T,D
 70 GOTO 30
100 DATA 12,12,15,16,19,19,21,
19
110 DATA 12,24,22,21,19,17,16,
14
120 DATA 12,12,15,16,19,19,21,
19
130 DATA 12,24,22,21,19,17,16,
14
140 DATA 17,17,20,21,24,24,26,
```

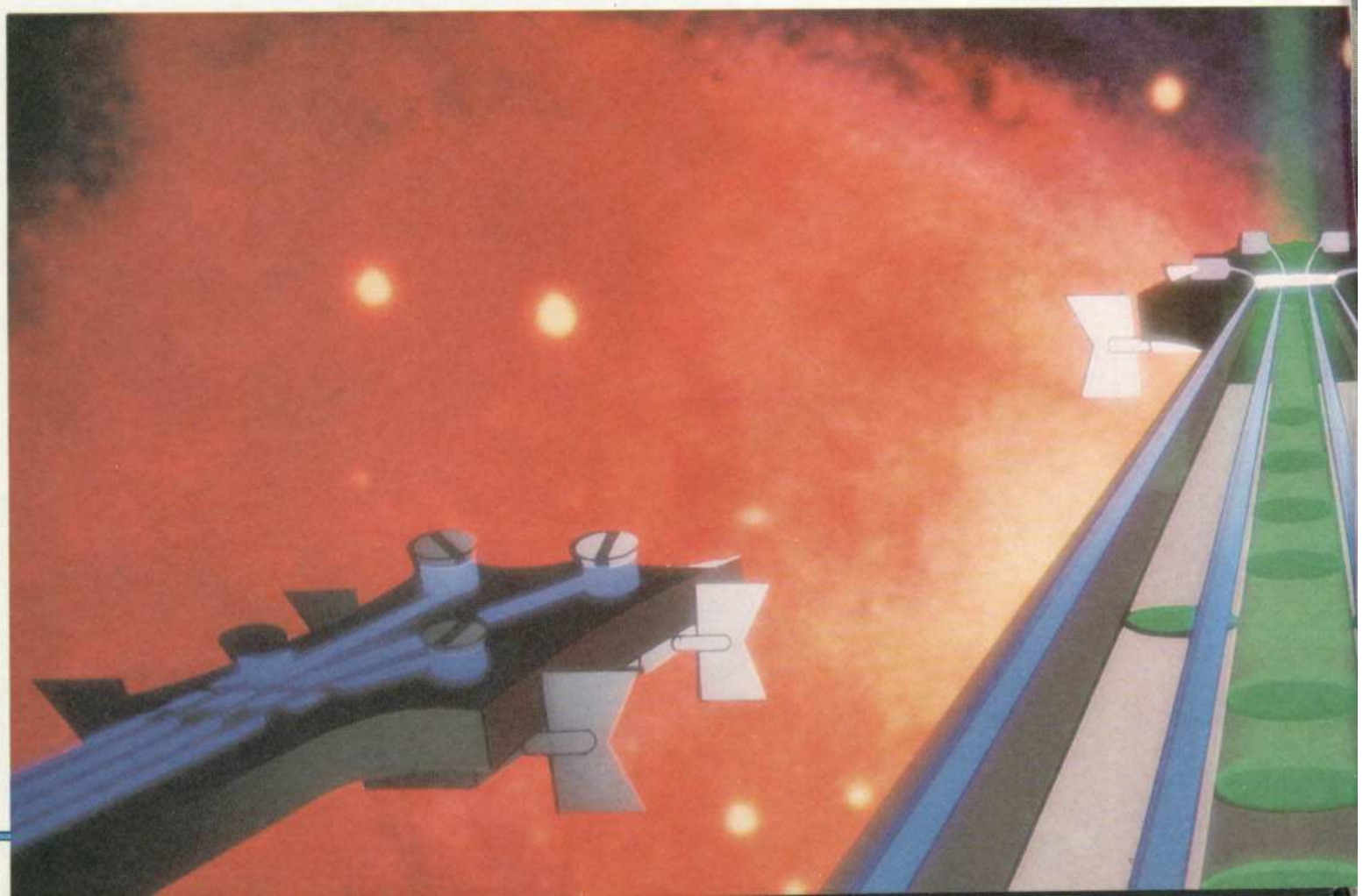


```
24
150 DATA 17,24,22,21,19,17,16,
14
160 DATA 12,12,15,16,19,19,21,
19
170 DATA 12,24,22,21,19,17,16,
14
180 DATA 19,19,23,24,26,26,24,
23
190 DATA 17,17,20,21,24,24,20,
21
200 DATA 12,12,15,16,19,19,21,
19
210 DATA 12,24,22,21,19,17,16,
14
220 DATA 255
```

```
10 SOUND 7,56:SOUND 8,15
20 SOUND 1,0:RESTORE
30 READ D
50 IF D=255 THEN 20
60 SOUND 0,D
70 FOR T=0 TO 100:NEXT:GOTO 30
100 DATA 213,213,179,169,142,14
2,126,142
110 DATA 213,106,119,126,142,15
9,169,189
120 DATA 213,213,179,169,142,14
2,126,142
130 DATA 213,106,119,126,142,15
9,169,189
140 DATA 159,159,134,126,106,10
6,94,106
150 DATA 159,106,119,126,142,15
9,169,189
160 DATA 213,213,179,169,142,14
2,126,142
170 DATA 213,106,119,126,142,15
9,169,189
180 DATA 142,142,112,106,94,94,
106,112
190 DATA 159,159,134,126,106,10
6,134,126
200 DATA 213,213,179,169,142,14
2,126,142
210 DATA 213,106,119,126,142,15
9,169,189
220 DATA 255
```

```
10  RESTORE :T = 80
20  FOR I = 0 TO 22: READ A: PO
KE 800 + I,A: NEXT
30  DATA  160,0,174,133,3,238,4
8,192,136,208,5,206,132,3,240,6
,202,208,245,76,34,3,96
40  READ D
50  IF D = 255 THEN 10
60  POKE 900,T: POKE 901,D: CAL
L 800
70  GOTO 40
100  DATA  47,47,40,37,31,31,28
,31
110  DATA  47,23,26,28,31,35,37
,42
120  DATA  47,47,40,37,31,31,28
,31
130  DATA  47,23,26,28,31,35,37
,42
140  DATA  35,35,29,28,23,23,21
,23
150  DATA  35,23,26,28,31,35,37
,42
160  DATA  47,47,40,37,31,31,2
8,31
170  DATA  47,23,26,28,31,35,3
7,42
180  DATA  31,31,25,23,21,21,23
,25
190  DATA  35,35,29,28,23,23,29
,28
200  DATA  47,47,40,37,31,31,2
8,31
210  DATA  47,23,26,28,31,35,3
7,42
220  DATA  255
```

```
10 T=3
20 RESTORE
30 READ D
50 IF D=255 THEN 20
60 SOUND D,T
70 GOTO 30
100 DATA 175,175,189,193,204,20
4,210,204
110 DATA 175,218,213,210,204,19
7,193,185
120 DATA 175,175,189,193,204,20
4,210,204
130 DATA 175,218,213,210,204,19
7,193,185
140 DATA 197,197,207,210,218,21
8,223,218
```



```
150 DATA 197,218,213,210,204,19
7,193,185
160 DATA 175,175,189,193,204,20
4,210,204
170 DATA 175,218,213,210,204,19
7,193,185
180 DATA 204,204,216,218,223,22
3,218,216
190 DATA 197,197,207,210,218,21
8,207,210
200 DATA 175,175,189,193,204,20
4,210,204
210 DATA 175,218,213,210,204,19
7,193,185
220 DATA 255
```

Se quiserem, os usuários do TK-2000 poderão modificar este e os próximos programas a fim de usar o comando **SOUND**. Mas isto não é necessário: tal como estão, os programas funcionam tão bem em seu micro quanto no Apple.

Em geral, o MSX requer o dobro dos dados utilizados pelos outros micros, já que emprega dois bytes para definir a nota musical — o byte mais significativo e o menos significativo. A melodia que escolhemos, no entanto, é tão simples que o byte mais significativo de todas as suas notas é igual a 0, o que nos permite especificar só os bytes menos significativos.

A variável **T** estabelece um fator de tempo para a velocidade da música. A linha 20 acerta o apontador das notas no início dos dados. O laço **FOR ... NEXT** entre as linhas 30 e 70 lê os valores das notas — os números nas linhas **DATA** —, colocando-os dentro do comando musical apropriado na linha 60. A linha 50 detecta o fim da melodia, marcado por um valor arbitrário — 255.

Se quisermos introduzir pausas no meio da melodia, podemos definir um outro valor arbitrário, 254 por exemplo, e incluir um teste para esse valor na linha 40. Ao surgir uma pausa no meio dos dados, a linha 40 desviaria o programa para uma outra linha. Esta produziria um certo atraso, voltando depois para a linha 30 a fim de continuar a execução da música.

Tal como o programa está, as notas são fornecidas aos comandos musicais do BASIC e emitidas conforme vão sendo lidas nas linhas **DATA**.

Embora esse processo funcione a contento, ele exige um número muito grande de dados, mesmo para uma melodia curta. Além de dificultar a digitação e tomar espaço na memória, o excesso de dados traz ainda um outro inconveniente: enquanto a música está sendo lida e processada, o computador não pode se ocupar com outra tarefa.

Alguns micros, como o MSX, solucionam parcialmente esse problema com um *buffer* musical, capaz de guardar algumas notas. Se houver espaço no buffer para o som que está sendo emitido, o computador é liberado para outras atividades. Porém, havendo mais notas a processar, o micro fica novamente comprometido. Seja como for, o buffer não alivia o trabalho de digitação das linhas **DATA** nem reduz o espaço por elas ocupado na memória.

Para contornar efetivamente esses problemas, o usuário só tem uma alternativa: encontrar um meio de comprimir os dados, tornando mais fácil seu processamento e economizando espaço.

MÃOS À OBRA

A compressão de dados é viabilizada pela existência, em uma melodia, de padrões que se repetem. Quanto mais padrões pudermos reconhecer, maior será o grau de compressão.

Para identificar esses padrões de dados, convém, em primeiro lugar, ouvir ou tocar a melodia em um instrumento, tentando distinguir as passagens que se repetem. Depois, escreva a música em um papel, preocupando-se apenas com a freqüência das notas — ignore pautas, claves, compassos etc.

Não há dificuldade em escrever, em

seqüência, as letras equivalentes às notas, se estas têm sempre a mesma duração. Mas, e se as notas tiverem duração variada? Nesse caso, o trabalho torna-se um pouco mais complicado, pois precisaremos levar em conta, além da tonalidade, o tempo de cada uma — o que duplicará o volume de dados.

Outra saída consiste em repetir as notas de maior duração. Se, por exemplo, uma nota dura duas batidas, ela será registrada duas vezes. Esta foi a alternativa que utilizamos ao transcrever a melodia do programa anterior. Obtivemos a seguinte tabela:

TABELA 1

G	G	A#	B	D	D	E	D	G
G2	F	E	D	C	B	A		
G	G	A#	B	D	D	E	D	G
G2	F	E	D	C	B	A		
C	C	D#	E	G2	G2	A2	G2	C
G2	F	E	D	C	B	A		
G	G	A#	B	D	D	E	D	G
G2	F	E	D	C	B	A		
D	D	F#	G2	A2	A2	G2	F#	
C	C	D#	E	G2	G2	D#	E	
G	G	A#	B	D	D	E	D	G
G2	F	E	D	C	B	A		

Se você examinar a tabela 1 com atenção, verá que a melodia se compõe de apenas cinco seqüências, ou minimelodias — identificadas como T1, T2, etc., como mostra a tabela 2 —, em sua maioria tocadas repetidas vezes.

TABELA 2

T1 =	G	G	A#	B	D	D	E	D	G
T2 =	G2	F	E	D	C	B	A		
T3 =	C	C	D#	E	G2	G2	A2	G2	C
T4 =	D	D	F#	G2	A2	A2	G2	F#	
T5 =	C	C	D#	E	G2	G2	D#	E	

Podemos agora utilizar uma técnica de compressão: as minimelodias não serão digitadas sempre que aparecem na música completa, mas uma única vez, junto a códigos que especificam a seqüência em que devem ser tocadas. Como costuma ocorrer quando comprimimos dados, o programa que lê a melodia assim digitada é maior que a anterior:

```
 10 LET C=0: LET T=.2
 20 RESTORE 100
 30 FOR N=1 TO C+1: READ P:
NEXT N
 40 IF P=0 THEN  GOTO 10
 50 RESTORE P
 60 READ N
 70 IF N>=255 THEN  LET C=C+1:
  GOTO 20
 80 SOUND T,N
 90 GOTO 60
100 DATA 110,120,110,120,130,
120,110,120,140,150,110,120,0
110 DATA 12,12,15,16,19,19,21,
19,12,255
120 DATA 24,22,21,19,17,16,14,
255
130 DATA 17,17,20,21,24,24,26,
24,17,255
140 DATA 19,19,23,24,26,26,24,
23,255
150 DATA 17,17,20,21,24,24,20,
21,255
```

```
1 SOUND 7,56:SOUND 8,15
2 SOUND 1,0
10 C=0:T=120
20 RESTORE
27 FORZ=1TOC+1:READP:NEXT
28 IFP=0THEN10
29 RESTORE:FORW=1TOP:READWW:NEX
T
30 READK:IFK=255THENC=C+1:GOTO2
0
50 SOUND 0,K
60 FORZ=1TOT:NEXT
70 GOTO30
100 DATA 13,23,13,23,31,23,13,2
3,41,50,13,23,0
110 DATA 213,213,179,169,142,14
2,126,142,213,255
120 DATA 106,119,126,142,159,16
9,189,255
130 DATA 159,159,134,126,106,10
6,94,106,159,255
140 DATA 142,142,112,106,94,94,
106,112,255
150 DATA 159,159,134,126,106,10
6,134,126,255
10013 ,23,13,23,0
```

```
1  FOR I = 0 TO 22: READ A: POK
E 800 + I,A: NEXT
2  DATA   160,0,174,133,3,238,4
8,192,136,208,5,206,132,3,240,6
,202,208,245,76,34,3,96
10 C = 0:T = 100
20  RESTORE
27  FOR Z = 1 TO C + 24: READ P
: NEXT
28  IF P = 0 THEN 10
29  RESTORE : FOR W = 1 TO P +
23: READ WW: NEXT
30  READ K: IF K = 255 THEN C =
 C + 1: GOTO 20
40  POKE 900,T: POKE 901,K
50  CALL 800
70  GOTO 30
100  DATA  13,23,13,23,31,23,13
,23,41,50,13,23,0
110  DATA   47,47,40,37,31,31,2
8,31,47,255
120  DATA   23,26,28,31,35,37,4
2,255
130  DATA    35,35,29,28,23,23,
21,23,35,255
140  DATA   31,31,25,23,21,21,2
3,25,255
150  DATA   35,35,29,28,23,23,2
9,28,255
220  DATA  255
```

```
1 DIM A(5,1):FOR K=1 TO 13:READ
 P:NEXT:GOTO 3
2 READ P:IF P<>255 THEN 2
3 N=N+1:A(N,0)=PEEK(51):A(N,1)=
PEEK(52):IF N<5 THEN 2
10 C=0:T=3
20 RESTORE
30 FOR N=1 TO C+1:READ P:NEXT
40 IF P=0 THEN 10
50 POKE 51,A(P,0):POKE 52,A(P,1
)
60 READ N
70 IF N=255 THEN C=C+1:GOTO 20
80 SOUND N,T
90 GOTO 60
100 DATA 1,2,1,2,3,2,1,2,4,5,1,
2,0
110 DATA 175,175,189,193,204,20
4,210,204,175,255
120 DATA 218,213,210,204,197,19
3,185,255
130 DATA 197,197,207,210,218,21
8,223,218,197,255
140 DATA 204,204,216,218,223,22
3,218,216,255
150 DATA 197,197,207,210,218,21
8,207,210,255
```

Note que o número de dados necessários para a execução da melodia foi muito reduzido — no Spectrum, por exemplo, diminuiu de 97 para 59 bytes. Rodando o programa, você verá que a música é exatamente a mesma do programa anterior. Os usuários do Spectrum vão estranhar o andamento da melodia, mas logo terão uma explicação.

Os dados que definem as minimelodias estão nas linhas 110 a 150 e a seqüência principal — isto é, a ordem em que as minimelodias são executadas — fica na linha 100. O laço na linha 30 acerta o valor de **P** (variável de trabalho utilizada na leitura das linhas **DATA**) de acordo com a seqüência principal, definindo a minimelodia a ser tocada. Na realidade, a seqüência princi-

pal é uma lista de números — ou de valores que, combinados com uma constante, resultam em números de linha — correspondentes às linhas onde se encontram as minimelodias.

Identificada a minimelodia a ser executada, a linha 50 calcula o número da linha **DATA** correspondente. Só o Spectrum permite o uso da instrução **RESTORE** durante a leitura do bloco de dados. O Apple e o MSX empregam laços **FOR...NEXT** e o TRS-Color tem os pontos de entrada gravados na linha 3.

A linha 10 utiliza a variável **C** para a contagem do número de minimelodias já tocadas. A duração de cada nota é controlada por **T**, que também determina o andamento da música. A linha 40 (28, no Apple e no MSX) verifica a última minimelodia, que é marcada com 0 na linha 100. O número 255 assinala o final de cada minimelodia. Ao encontrá-lo, o computador volta à seqüência principal para saber qual será a próxima minimelodia.

Existem várias alternativas para a divisão de uma música em seqüência principal e minimelodias. De um modo geral, quanto menores as minimelodias, maior é a seqüência principal. Devemos buscar um equilíbrio entre esses dois elementos, a fim de evitar que minimelodias muito curtas exijam uma seqüência principal tão longa que anule as vantagens da compressão.

DIVIDINDO PARA VENCER

A segunda técnica de compressão é ainda mais eficiente. Ela parte do princípio de que, embora os micros possam tocar um grande número de notas — geralmente 256 —, poucas delas são de fato utilizadas durante a execução de uma melodia. Assim, não nos interessa — e não é econômico — empregar um sistema planejado para aceitar um grande número de notas.

Cada posição de memória em seu microcomputador de oito bits é capaz de armazenar um número inteiro entre 0 e 255. Restringindo a quantidade de notas a serem armazenadas, será possível colocar duas delas em cada posição — uma em cada quatro bits. Por exemplo, o número binário 10100010, de oito bits, pode ser interpretado como um só valor decimal — 162 — ou como dois valores — 10 (1010 em binário) e 2 (0010 em binário).

Dividir por dois o espaço disponível diminui drasticamente o intervalo de valores que podemos armazenar: 0 a 15. Contudo, muitas vezes isso é suficiente, pois o número de notas diferentes em uma melodia simples raramente ultrapassa dezesseis.

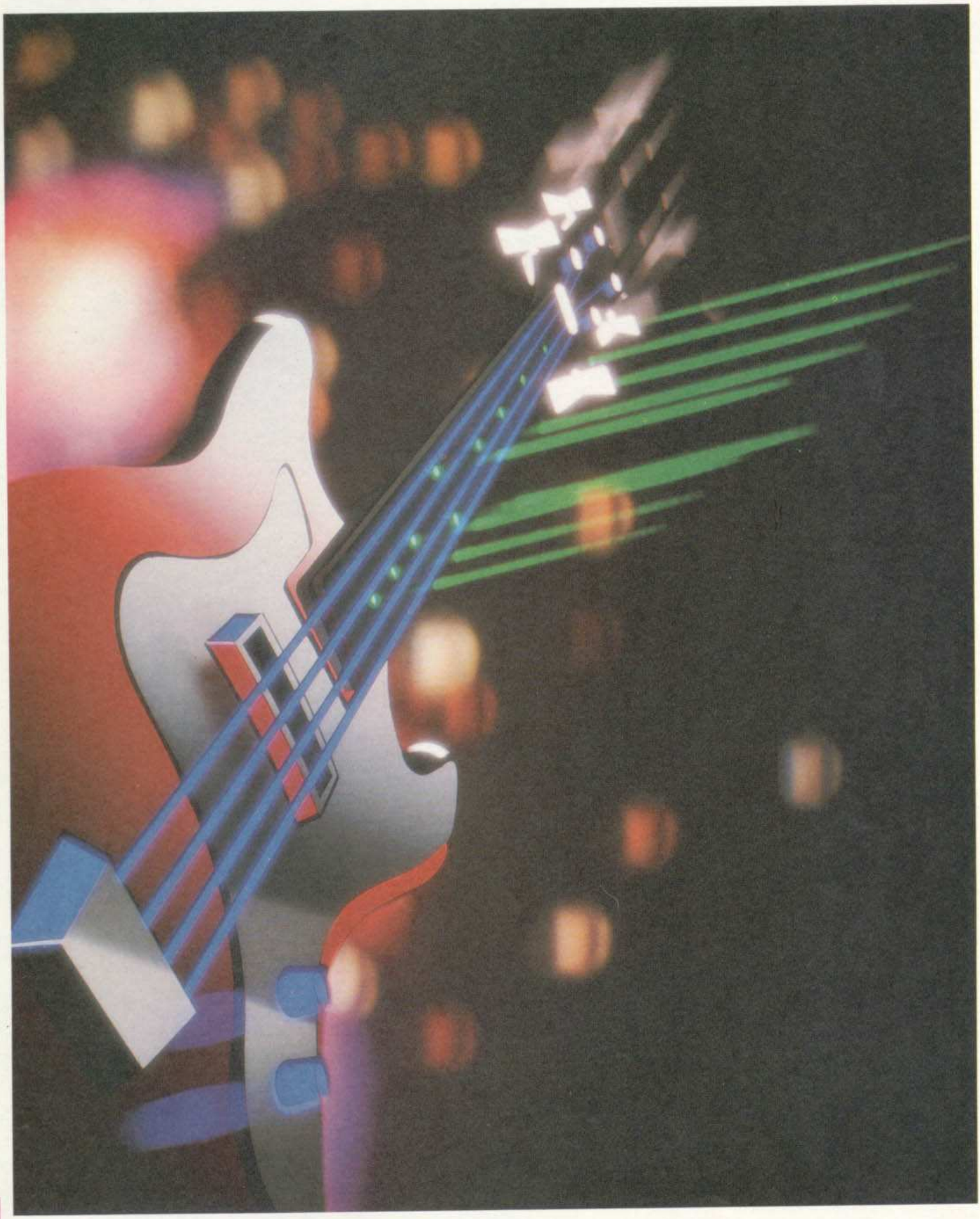
Para utilizar essa técnica de compressão, precisamos reduzir o número de notas a quinze, deixando o 16º valor como um código de controle. Já comentamos que, quanto mais comprimidos os dados, maior é o programa necessário para interpretá-los. Aqui, o programa deve decifrar cada nota abreviada que vai lendo nas linhas **DATA**.

O usuário, por sua vez, terá o trabalho de selecionar as notas que serão usadas na melodia, classificando-as em ordem crescente, conforme a freqüência, a partir da nota mais baixa da primeira oitava — nesse caso, G1. Em seguida, precisará calcular os números codificados que aparecerão nas linhas **DATA**. Nossa melodia inclui doze notas: G, A, A#, B, C, D, D#, E, F, F#, G2 e A2.

Digite o terceiro programa e veja o sistema em ação.

```
 12 GOSUB 1000: LET T=.15: LET
  NT=130: LET MS=170: LET MT=
 210: GOSUB 300
 90 STOP
100 DATA 12,14,15,16,17,19,20,
21,22,23,24,26,0,0,0,0
200 DATA 1,2,1,2,3,2,1,2,4,5,1
,2,0
210 DATA 0,35,85,117,15,255
220 DATA 168,117,67,31,255
230 DATA 68,103,170,186,79,255
240 DATA 85,154,187,169,255
250 DATA 68,103,170,103,255
310 RESTORE NT
320 FOR N=23410 TO 23425
330 READ X: POKE N,X: NEXT N
345 LET NM=0
350 RESTORE MS: LET HL=23426
360 READ X
365 IF X>NM THEN  LET NM=X
370 IF X=0 THEN  GOTO 400
380 POKE HL,X: LET HL=HL+1:
GOTO 360
400 POKE HL,X: LET HL=HL+1
401 LET X=HL: GOSUB 600
402 POKE 23403,LSB
403 POKE 23404,MSB
430 RESTORE MT
440 FOR N=1 TO NM
450 READ X
460 IF X=255 THEN  GOTO 500
470 POKE HL,X: LET HL=HL+1:
GOTO 450
500 POKE HL,X: LET HL=HL+1
510 NEXT N
511 RAND USR 23371
512 POKE 23409,0
530 LET X=USR 23296
540 IF X=255 THEN  RETURN
550 SOUND T,X: GOTO 530
600 LET MSB=INT (X/256)
610 LET LSB=X-(MSB*256):
RETURN
1000 RESTORE 2000: LET TO=0: LE
T L=2000
1030 FOR N=23296 TO 23296+111 S
TEP 8
1040 FOR K=0 TO 7: READ A: LET
TO=TO+A: POKE K+N,A: NEXT K
1050 READ A: IF A<>TO THEN  GOT
O 1080
1060 LET L=L+10: LET TO=0: NEXT
 N
1065 RESTORE : RETURN
1080 PRINT "ERRO DE DADOS NA LI
NHA ";L: STOP
2000 DATA 42,109,91,235,42,111,
91,58,779
2010 DATA 113,91,70,183,202,24,
91,175,949
2020 DATA 8,35,62,15,160,195,36
,91,602
2030 DATA 61,1,8,203,56,203,56,
203,791
2040 DATA 56,203,56,120,254,15,
202,65,971
2050 DATA 91,34,111,91,235,34,1
09,91,796
2060 DATA 8,50,113,91,33,114,91
,22,522
2070 DATA 0,8,95,25,126,6,0,79,
339
2080 DATA 201,26,19,183,194,81,
91,1,796
2090 DATA 255,0,201,17,130,91,1
95,65,954
2100 DATA 91,71,42,107,91,43,62
,255,762
2110 DATA 16,5,35,175,195,10,91
,35,562
2120 DATA 190,194,103,91,195,88
,91,35,987
2130 DATA 195,96,91,0,0,0,0,0,3
82
```

```
10 SOUND 7,56:SOUND 8,15
20 SOUND 1,0
23 DIM X(16):RESTORE:FOR N=1 TO
 16:READ X(N):NEXT
25 C=0:T=100
26 RESTORE1000
27 FORZ=1TOC+1:READP:NEXT
28 IFP=0THEN25
29 RESTORE1010:FORW=1TOP+2:READ
WW:NEXT
50 READN:SS=N
60 N=INT(N/16)
70 IFN=15THENC=C+1:GOTO26
80 GOSUB130
90 N=SS:N=15ANDN
100 IFN=15THENC=C+1:GOTO26
110 GOSUB130
120 GOTO50
130 SOUND0,X(N+1)
140 FORZ=1TOT:NEXT
150 RETURN
450 DATA 213,189,179,169,159,14
2,134,126,119,112,106,94,0,0,0,
0
1000 DATA 5,10,5,10,15,10
1010 DATA 5,10,20,25,5,10,0
1110 DATA 0,35,85,117,15
1120 DATA 168,117,67,31,255
```



```
1130 DATA 68,103,170,186,79
1140 DATA 85,154,187,169,255
1150 DATA 68,103,170,103,255
```

```
1  FOR I = 0 TO 22: READ A: POK
E 800 + I,A: NEXT
2  DATA   160,0,174,133,3,238,4
8,192,136,208,5,206,132,3,240,6
,202,208,245,76,34,3,96
23  DIM X(16): FOR N = 1 TO 16:
 READ X(N): NEXT
25 C = 0:T = 100
26  RESTORE
27  FOR Z = 1 TO C + 40: READ P
: NEXT
28  IF P = 0 THEN 25
29  RESTORE : FOR W = 1 TO P +
47: READ WW: NEXT
50  READ N:SS = N
60 N =  INT (N / 16)
70  IF N = 15 THEN C = C + 1: G
OTO 26
80  GOSUB 130
90 N = SS - N * 16
100  IF N = 15 THEN C = C + 1:
GOTO 26
110  GOSUB 130
120  GOTO 50
130  POKE 900,T: POKE 901,X(N +
 1)
140  CALL 800: RETURN
450  DATA  47,42,40,37,35,31,29
,28,26,25,23,21,0,0,0,0
1000  DATA  5,10,5,10,15,10
1010  DATA  5,10,20,25,5,10,0
1110  DATA  0,35,85,117,15
1120  DATA  168,117,67,31,255
1130  DATA  68,103,170,186,79
1140  DATA  85,154,187,169,255
1150  DATA  68,103,170,103,255
```

```
10 DIM A(5,1),X(16):FOR K=1 TO
16:READ X(K):NEXT:A(0,0)=PEEK(5
1):A(0,1)=PEEK(52)
15 FOR K=1 TO 13:READ P:NEXT:GO
TO 30
20 FOR K=1 TO 5:READ P:NEXT
30 N=N+1:A(N,0)=PEEK(51):A(N,1)
=PEEK(52):IF N<5 THEN 20
40 C=0:T=3
50 POKE 51,A(0,0):POKE 52,A(0,1
):FOR N=1 TO C+1:READ P:NEXT:IF
 P=0 THEN 40
60 POKE 51,A(P,0):POKE 52,A(P,1
)
65 READ N:S=N
70 N=INT(N/16)
75 IF N=15 THEN C=C+1:GOTO 50
80 GOSUB 130
90 N=S:N=15 AND N
100 IF N=15 THEN C=C+1:GOTO 50
110 GOSUB 130
120 GOTO 65
130 SOUND X(N+1),T:RETURN
450 DATA 175,185,189,193,197,20
4,207,210,213,216,218,223,0,0,0
,0
1000 DATA 1,2,1,2,3,2
1010 DATA 1,2,4,5,1,2,0
1110 DATA 0,35,85,117,15
1120 DATA 168,117,67,31,255
1130 DATA 68,103,170,186,79
1140 DATA 85,154,187,169,255
1150 DATA 68,103,170,103,255
```

Os valores das freqüências dessas notas estão codificados na linha 450 (100 no Spectrum). Essa linha deve conter dezesseis valores. No MSX, poderíamos precisar de 32 valores para outras melodias, já que ele requer dois parâmetros para um valor de nota.

Em todas as versões, codificamos só doze notas; assim, quatro espaços são preenchidos com 0. Os quatro dígitos binários (chamados *nybble*) que formam cada nota das minimelodias estão codificadas nas linhas 1000 e 1010 (200 e 210 no Spectrum). Para entender como se calculam esses valores, tomemos como exemplo a minimelodia T1. Veja os números da seqüência principal na linha 450 (100 no Spectrum). Chamemos a primeira nota dessa linha de 0, a segunda de 1 e assim por diante, até a última nota, que será 15. Se escrevermos a minimelodia T1 com esses códigos, teremos T1 = 0, 0, 2, 3, 5, 5, 7, 5, 0, 15. O número 15, no caso, indica o final da minimelodia, e não uma nota. Cada um desses valores cabe em um nybble; assim, precisamos combiná-los em pares para obter os bytes.

O primeiro par de nybbles é 0 e 0, valores que correspondem em binário, a 0000 e 0000. Combinados em um byte, resultam no decimal 0. O primeiro valor da linha **DATA** 1110 (210 no Spectrum), que equivale a T1, é, portanto, 0. O próximo par é 2 e 3 — em binário, 0010 e 0011. Combinados, esses valores resultam em 00100011, ou 35 em decimal. Este é o segundo valor da linha 1110 (ou 210). Procedendo dessa maneira com a minimelodia seguinte, teremos T2 = 10, 9, 7, 5, 4, 3, 1, 15. Combinados, os valores 10 e 9 — 1010 e 1001 em binário — resultam 168 em decimal. Este é o primeiro valor de T2 na linha 1120 (220 no Spectrum).

Utilizamos esse método para calcular os valores correspondentes às minimelodias, armazenados nas linhas 1110 a 1150 (210 a 250 no Spectrum). Como no programa anterior, a seqüência principal (linhas 1000 e 1010, ou 200 no Spectrum) aponta para as linhas **DATA** onde estão as minimelodias, na ordem em que devem ser tocadas.

O PROCESSO INVERSO

Outra parte importante do programa é a que se encarrega de extrair os nybbles dos números decimais lidos nas linhas **DATA**. O Spectrum faz isso colocando o valor em uma variável (linha 401) e chamando, em seguida, uma sub-rotina (linhas 600 e 610) que separa os dois nybbles do byte. Devidamente processados, esses valores são utilizados para emitir duas notas.

Nos outros micros, as linhas 50 a 100 cuidam da decodificação. Primeiro, o byte em exame, **N**, é armazenado na linha 50. A linha 60 extrai o nybble mais significativo e a linha 70 verifica o código de final de melodia. A linha 80 chama a sub-rotina que executa a nota correspondente e a linha 90 extrai o nybble menos significativo, usando a operação lógica **AND** (menos no Apple e no TK-2000, cuja função **AND** não permite o cálculo de valores diferentes de 0 e 1). A nota é então emitida. Note que o byte em exame é preservado, sem o que não seria possível recuperar o segundo nybble.

Ao executar esse programa, os usuários do Spectrum observarão que os problemas de andamento causados pela compressão já não ocorrem. O defeito era provocado por um tempo extra de processamento, necessário ao realinhamento dos ponteiros, ou seja, ao uso de **RESTORE**. O programa reduziu esse tempo de processamento por meio de uma rotina em código, armazenada nas linhas 1000 a 2130. Esta é a razão das diferenças de tamanho e numeração entre o Spectrum e os demais micros.

Para alterar a velocidade de execução, altere o valor de **T**. Ele fica na linha 12, no Spectrum, e na linha 40, nos outros computadores. Observe, porém, que o intervalo de tempo entre o final de uma minimelodia e o início da seguinte torna-se mais acentuado à medida que **T** diminui.

MICRO DICAS

MAIS MELODIAS

Como exercício, tente codificar outras melodias, utilizando o último programa deste artigo. Identifique as minimelodias e as seqüências principais, calcule os valores correspondentes e coloque-os em linhas **DATA**. Será preciso redefinir as matrizes — **X( )** — da seqüência principal para cada nova melodia, o que pode exigir a renumeração do programa.

AS CINCO VIDAS DE WILLIE

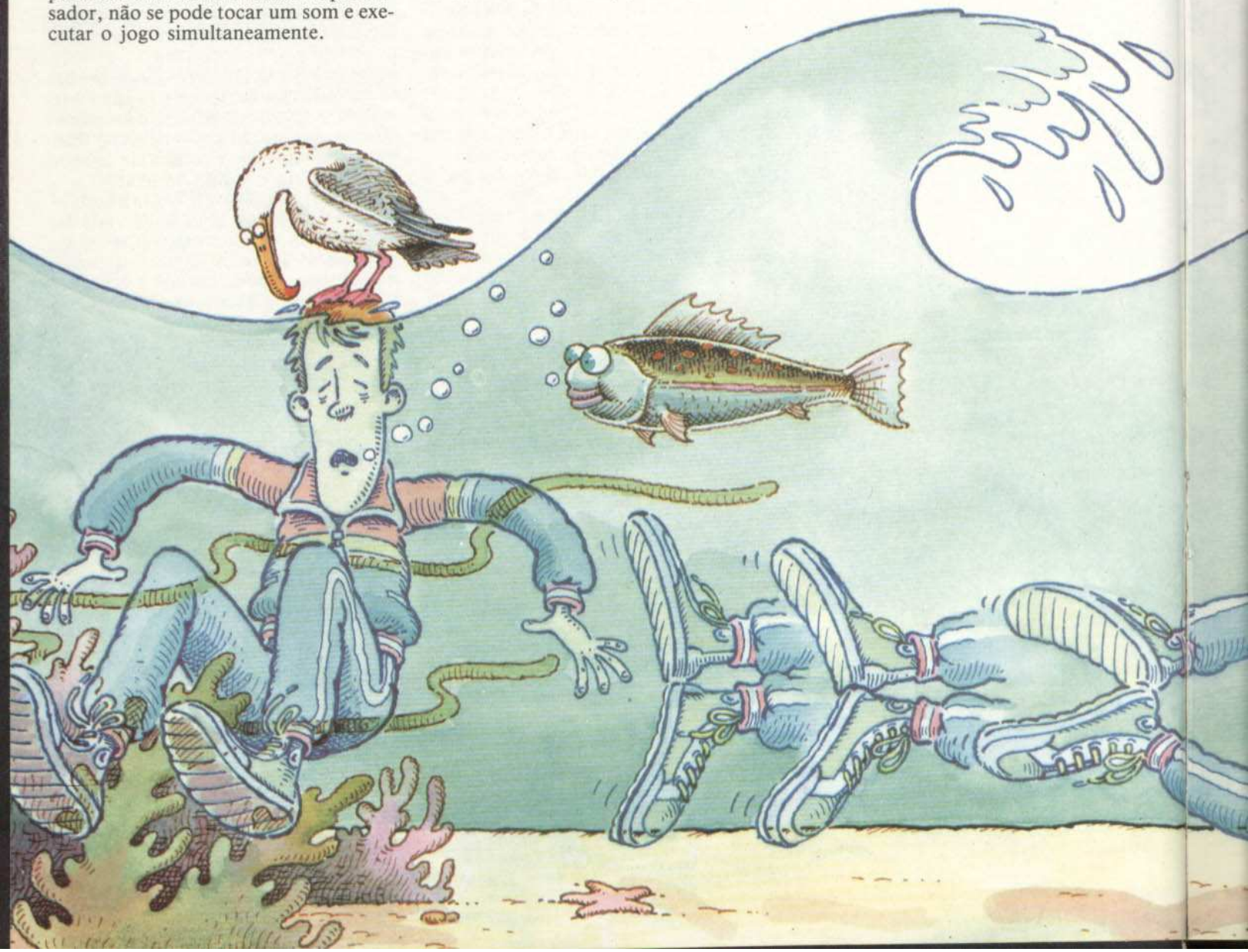
A morte vem para todos — mas não tão insistentemente como para Willie. Ele morre cinco vezes em cada jogo... e está sempre pronto a reiniciar a aventura na hora que você quiser!

Mesmo que você seja um exímio jogador, mais cedo ou mais tarde a morte levará Willie. Mas não se afobe. Ele tem cinco vidas — nem tantas como um gato; bem mais, porém, do que os pobres mortais. Quando a fatalidade atingir Willie definitivamente, você deverá enterrá-lo, imprimir o placar final e ajustar todas as variáveis.

Os sons emitidos pelo Spectrum são muito simples, acompanhando a execução dos programas. O TRS-Color pode produzir sons sofisticados. Mas, como para isso é necessário utilizar o processador, não se pode tocar um som e executar o jogo simultaneamente.

A rotina apresentada a seguir promove a morte de Willie e verifica se ele pode ser ressuscitado.

```
10 REM org 59652
20 REM die ld de,196
30 REM ld hl,1086
40 REM call 949
50 REM ld de,131
60 REM ld hl,1646
70 REM call 949
80 REM dead ld hl,(57332)
 90 REM ld bc,15616
100 REM ld a,45
110 REM call 58217
120 REM ld de,32
130 REM add hl,de
140 REM ld (57332),hl
150 REM ld bc,57000
160 REM ld a,40
170 REM ld de,258
180 REM call 58970
190 REM ld de,30
200 REM ld hl,878
210 REM ld a,r
```



```
220 REM ld l,a
230 REM call 949
240 REM ld hl,(57332)
250 REM ld de,704
260 REM sbc hl,de
270 REM jr c,dead
280 REM ld a,9
290 REM ld (60005),a
300 REM ld a,(57343)
310 REM dec a
320 REM ld (57343),a
330 REM jp nz,58606
340 REM ld hl,330
350 REM ld a,142
360 REM ld b,11
370 REM ld ix,57992
380 REM call me
390 REM call 58939
400 REM ld de,261
410 REM ld hl,1646
420 REM call 949
430 REM ld de,392
440 REM ld hl,1086
450 REM call 949
460 REM ld de,261
470 REM ld hl,1646
480 REM call 949
490 REM ld a,50
500 REM ld (58732),a
510 REM jp 58576
```

Se Willie se atogou no mar, caiu num buraco, foi picado por uma cobra ou atingido por uma pedra, os sinos tocam, anunciando sua morte. Para que soem as duas badaladas fúnebres, a rotina **BEEPER** é chamada em 949 duas vezes — em cada uma delas, com parâmetros diferentes nos pares HL e DE.

DESCIDA PARA O INFERNO

Quando Willie morre, ele desce para o inferno, que se situa no fundo da tela. Para criar esse efeito, começamos por apagar sua cabeça — se não o fizermos, teremos uma coluna de cabeças descendo até o fundo da tela!

Assim, a posição de Willie na tela — que aponta para a posição de sua cabeça — é carregada da variável em 57332 e 57333 para o par HL. BC é carregado com o endereço de um caractere limpo de céu do topo da tela. A é ajustado para ciano sobre ciano. Finalmente, a rotina **print** em 58217 é chamada, apagando a cabeça de Willie.

O valor 32 é então somado ao apontador de tela em HL, através do par DE, o que o faz se mover para a linha de baixo da tela. O resultado é colocado de volta em 57332 e 57333 para ajustar também a variável.

BC é carregado com o endereço inicial da figura de Willie com as pernas juntas. A é ajustado com o código da cor de Willie, azul sobre ciano. DE é carregado com 258, especificando um bloco de um por dois — o conteúdo do registro D define a largura do bloco, e o conteúdo do registro E, sua profundidade: 1 x 256 + 2 = 258. A rotina de impressão de bloco em 58970, **blk**, é chamada, exibindo na tela a figura de Willie com as pernas juntas, uma posição abaixo da que ele ocupava antes.

Enquanto Willie desce, ouve-se uma música "celestial". Mais uma vez, os pares de registros HL e DE são carregados com parâmetros que a rotina **BEEPER** utilizará, quando for chama-

da em 949. Desta vez, porém, em vez de um som constante, o programa precisa de um som intermitente. Para isso, carrega-se o conteúdo do registrador R na parte menos significativa de HL, que é o registro L. A operação é realizada via acumulador porque não existe nenhuma instrução **ld l,r**.

Usamos o registro R para manter a memória dinâmica. Este é um recurso do hardware com o qual você não precisa se preocupar. Para nossos fins, basta saber que o valor desse registro é variável. Assim, cada vez que esta parte da rotina for acessada irá produzir um som diferente.

SEIS PÉS DE PROFUNDIDADE

A posição da cabeça de Willie é carregada da variável em 57332 e 57333 para o par HL; 704 é carregado em DE e subtraído do conteúdo de HL. O número 704 — 32 × 22 — indica o começo da penúltima linha. Como Willie tem dois caracteres de altura, essa subtração colocará o valor 1 na baliza *carry* até que os pés de Willie tenham tocado o fundo da tela.

Enquanto a baliza carry contiver 1, a instrução **jr c, dead** manda o processador de volta ao início da rotina de enterro do personagem. Willie será afundado mais uma posição e os sinos voltarão a tocar — num tom diferente. Quando os pés de Willie tocarem o inferno, ele estará completamente enterrado. A baliza carry não será ajustada com 1 pela subtração, não haverá salto e o processador passará a executar a próxima instrução.

Tendo Willie chegado ao seu destino final, convém tocar uma canção confortadora. Assim, carrega-se A com 9 e a rotina **sound** em 60006 é chamada. Ela executa uma versão curta da melodia inicial do jogo. Para obter maiores detalhes sobre rotina da música, reveja o artigo da página 788.

Como nosso personagem perdeu uma vida, precisamos reajustar — ou melhor, decrementar — o número de vidas. O valor dessa variável é carregado de 57343 para o acumulador, decrementado e recolocado em 57343.

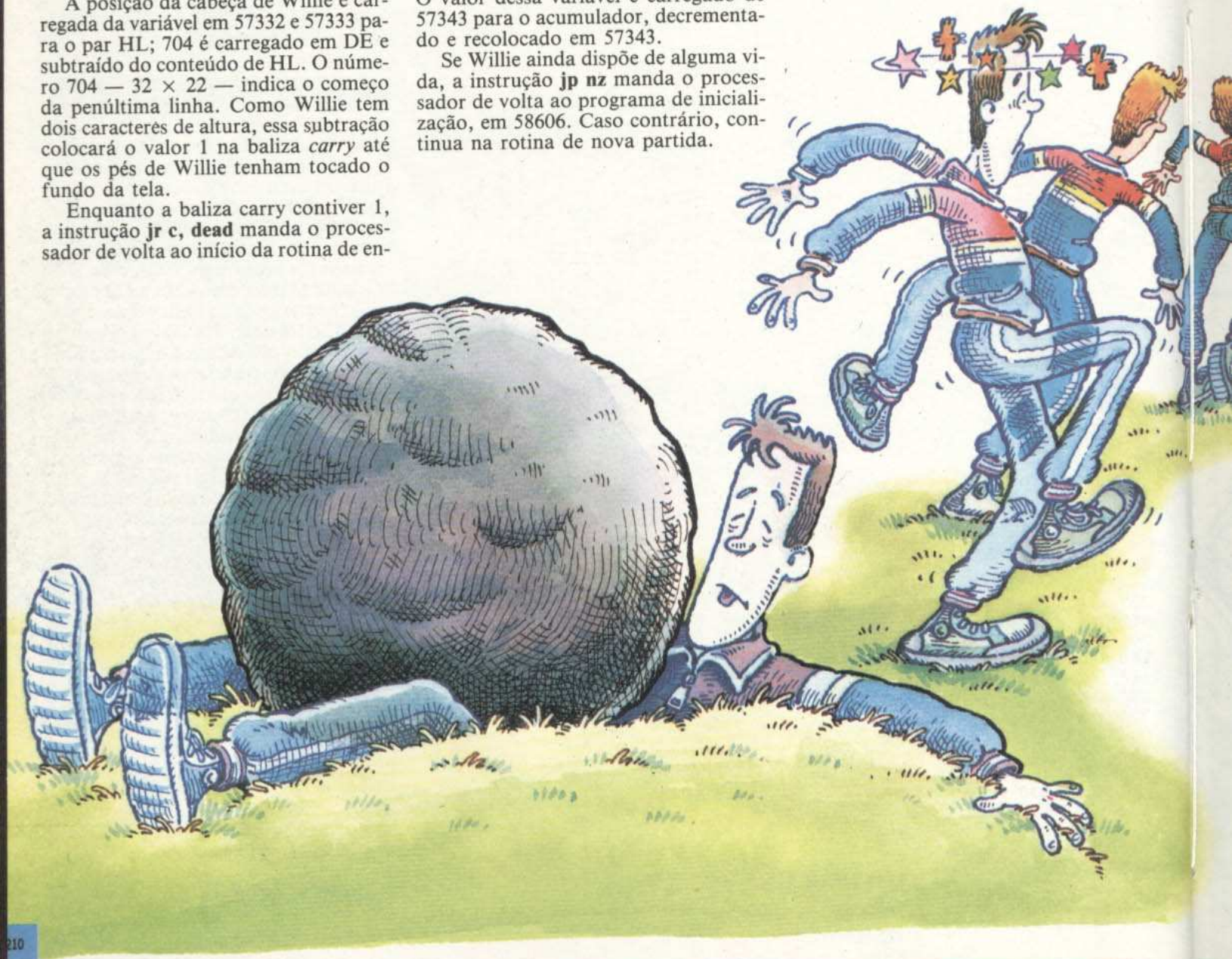
Se Willie ainda dispõe de alguma vida, a instrução **jp nz** manda o processador de volta ao programa de inicialização, em 58606. Caso contrário, continua na rotina de nova partida.

ACABARAM-SE AS VIDAS

Quando Willie tiver morrido pela quinta vez, o jogo acaba. Precisamos, então, imprimir a mensagem "GAME OVER!". Para isso, chamamos a rotina de mensagens, ou **me**, em 58155. Como sempre, os parâmetros devem ser colocados nos registros HL, A, B e IX.

A posição de impressão é carregada em HL. A cor é ajustada em A. B carrega o comprimento da mensagem e IX é usado como apontador de dados. Os dados para a mensagem estão em 57892.

A rotina que imprime o escore em 58939 é chamada para imprimir o placar. Em seguida, ouve-se o toque que marca o final do som no jogo. Para



produzi-lo, a rotina **BEEPER** é chamada em 949 três vezes, com valores diferentes em HL e DE a cada vez.

Para reajustar o atraso inicial, carrega-se o acumulador com 50 e transfere-se o valor na posição 58732. Esse endereço ainda não foi mencionado: ele pertence à rotina final do jogo e é incumbido de controlar o atraso geral. O processador salta para 58578, ajustando-o de maneira que você possa dar reinício à aventura, se quiser.

A rotina a seguir coloca Willie em seu túmulo:

```
10  ORG 20560
20  DIE LDA #136
```

```
30  LDX #140
40  JSR SOUND
50  LDA #131
60  LDX #213
70  JSR SOUND
80  DI LDX 18249
90  LDU #1536
100  JSR CHARPR
110  LEAX 254,X
120  STX 18249
130  LDU #17774
140  JSR CHARPR
150  LEAX 254,X
160  JSR CHARPR
170  LDA #30
180  LDX #113
190  JSR SOUND
200  LDX 18249
210  CMPX #6912
220  BLO DI
230  DEC 18239
240  LBNE NLV
250  PB LDA #5
260  PAA LDX #65535
270  LEAX -1,X
280  BNE PAA
290  DECA
300  BNE PB
310  JSR CLS
320  LDA $FF22
330  ANDA #15
340  STA $FF22
350  STA $FFC2
360  STA $FFC4
370  STA $FFC6
380  LDY #$50B
390  LDX #$701
400  STX ,Y++
410  LDX #$D05
420  STX ,Y++
430  LDX #$200F
440  STX ,Y++
450  LDX #$1605
460  STX ,Y++
470  LDX #$1221
480  STX ,Y++
490  LDA #200
500  LDX #255
510  JSR SOUND
520  LDA #200
530  LDX #200
540  JSR SOUND
550  LDA #255
560  LDX #255
570  JSR SOUND
580  LDA #100
590  STA DLL+1
600  LBRA GBIN
610  SOUND EQU $5133
620  CHARPR EQU $4BCA
630  DLL EQU $51ED
640  GBIN EQU $4BE2
650  NLV EQU $4BF7
660  CLS EQU $4ACC
```

OS SINOS

Quando a rotina começa, ouve-se um sino bater duas vezes. A homenagem sonora ao finado Willie, porém, é promovida pela rotina **SOUND**, que será fornecida no próximo artigo de *Avalanche*.

Antes de testar o programa aqui apresentado, lembre-se de colocar um **RTS** no endereço inicial de **SOUND**, $5133. Caso contrário, haverá um erro.

SOUND requer dois parâmetros que especifiquem o tipo e a duração do som. Eles são utilizados pela rotina nos registradores A e X. Como as duas notas produzidas diferem uma da outra, a rotina **SOUND** trabalha com parâmetros distintos a cada chamada.

DESCENDO PARA O INFERNO

Willie descerá na tela em direção ao inferno. Sua figura anterior deve, assim, ser apagada.

X é carregado com o conteúdo de 18249, que aponta para a posição que o personagem ocupa na tela; U é carregado com 1536, o endereço do céu no topo da tela. Em seguida, a rotina **CHARPR** é chamada e imprime um bloco de céu, apagando a metade superior da figura de Willie. O conteúdo de X é então adicionado a 254 e recolocado em 18249, para que o apontador de Willie desça um caractere na tela.

U é carregado com 17774, o endereço inicial dos dados para a figura de Willie com as pernas juntas. Depois, o processador salta para a rotina **CHARPR**, que imprime a metade superior de Willie uma posição abaixo da última impressão. Novamente, X é incrementado com 254 e **CHARPR** é chamada para imprimir a metade inferior da figura.

Enquanto Willie está descendo na tela, ouve-se um grito de agonia — para isso, A e X são carregados de novo e o processador salta para **SOUND**.

Willie deve chegar ao fundo da tela. Verificamos se isso já ocorreu carregando X com o conteúdo de 18249 e comparando esse valor com 6912, o início de 28ª linha da tela. Se a posição de Willie em X for menor do que 6912, a instrução **BLO** manda o processador reiniciar o laço **DI** e move Willie um caractere para baixo. Caso contrário, o processador segue em frente, pois Willie chegou ao fundo da tela.

O conteúdo de 18249 — que armazena o número de vidas que restam a Willie — é então decrementado e **LBNE NLV** manda o processador para a rotina que dá nova vida a Willie e traz de volta a última tela. Uma instrução de desvio longo é usada neste ponto, porque a rotina **NLV** encontra-se numa parte bem anterior do programa — certamente mais do que 128 bytes, o limite para um desvio curto.

OUTRA PARTIDA

Se o conteúdo de 18249 foi reduzido a 0, Willie não tem mais nenhuma vida, o jogo terminou e o processador executa a instrução seguinte.

A é carregado com 5; X é carregado com 65535 e, depois, decrementado. O processador fica nesse laço até que X seja 0. Em seguida, A é decrementado e X volta a ser carregado com 65535. O processo se repete até que A também tenha sido reduzido a 0.

O processador fica nesse laço 5 x 65535 vezes — pausa suficiente para que o jogador medite sobre o trágico destino de Willie. Ao sair do laço de atraso, o processador salta para a rotina **CLS**, que limpa a tela.

O conteúdo de $FF22 é então carregado em A, onde se executa a operação lógica **AND** com o valor 15. O resultado é armazenado nas posições de memória $FF22, $FFC2, $FFC4 e $FFC6.

Informamos, assim, ao VDG (gerador do sinal de vídeo) e ao SAM (multiplexador síncrono de endereços) que estamos voltando ao modo texto. O procedimento é o inverso do que utilizamos ao mudar para o modo gráfico.

Y é carregado com $50B, posição onde iremos imprimir a expressão "GAME OVER!". X, por sua vez, é carregado com $701, códigos de tela para as letras GA. Essas letras são colocadas na tela na posição apontada por Y, que é incrementado duas vezes.

Em seguida, X é carregado com os códigos de tela para ME e as duas letras são impressas na nova posição apontada por Y, dois blocos à direita do início de GA. Por fim, os últimos caracteres — O, VE e R! — são carregados e impressos nas posições adequadas.

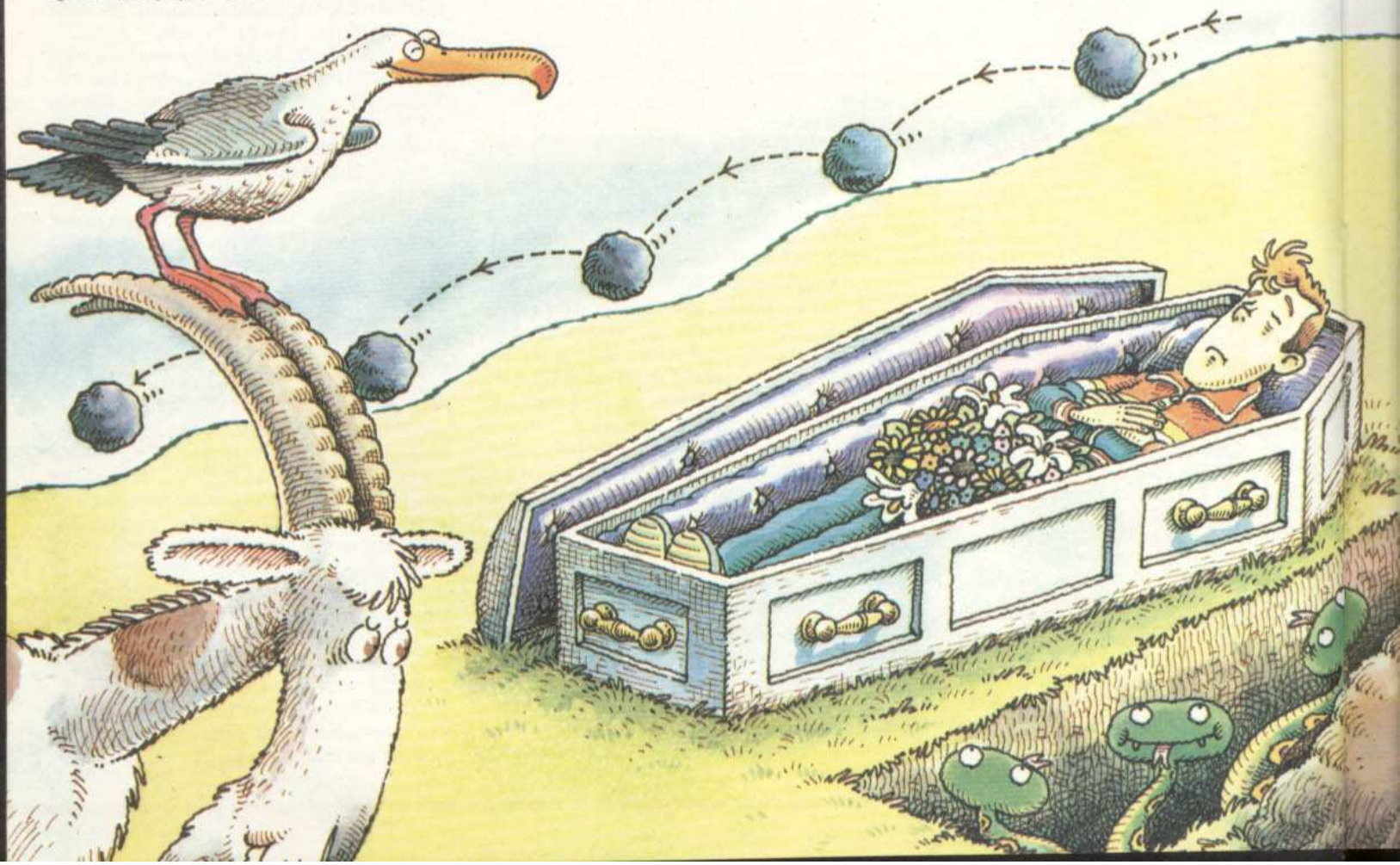
Um som anuncia o fim do jogo. Como esse som é formado por três notas, A e X são carregados e a rotina **SOUND** é chamada três vezes.

O número 100 é carregado no acumulador e armazenado em $51EE. Com isso, ajustamos o atraso com o valor inicial. Quando o jogo for novamente executado, começará com a mesma velocidade.

Encerrando a execução da rotina, o processador faz um longo desvio para o início do programa em **GBIN**.

A rotina a seguir enterra Willie e verifica se pode ser ressuscitado:

```
10    org 55334
20    ld hl,(-5205)
30    ld de,(62407)
40    add hl,de
50    inc hl
60    ld a,255
70    push hl
80    call 77
90    pop hl
100    dec hl
110    dec hl
120    ld a,255
130    push hl
140    call 77
150    pop hl
160    ld de,32
170    add hl,de
180    ld a,255
190    push hl
200    call 77
210    pop hl
220    inc hl
230    inc hl
240    ld a,255
250    push hl
260    call 77
270    pop hl
280    inc hl
290    ld a,255
300    call 77
310    mo ld hl,(-5205)
320    push hl
330    ld de,(62407)
340    add hl,de
350    ld a,255
360    call 77
370    pop hl
380    ld de,32
390    add hl,de
400    ld (-5205),hl
410    ld de,(62407)
420    add hl,de
430    ld a,0
440    push hl
450    call 77
460    pop hl
470    ld de,32
480    add hl,de
490    ld a,1
500    call 77
510    ld b,255
520    atr ld a,255
530    dl dec a
540    jr nz,dl
550    djnz atr
560    ld hl,(-5205)
570    ld de,704
580    sbc hl,de
590    jr c,mo
600    ld a,(-5221)
610    dec a
620    ld (-5221),a
```



```
630    jp nz,53888
640    ld de,234
650    ld a,114
660    ld b,12
670    tt push bc
680    push de
690    push af
700    ld hl,(62407)
710    add hl,de
720    call 77
730    pop af
740    inc a
750    pop de
760    inc de
770    pop bc
780    djnz tt
790    ld b,30
800    tu ld a,255
810    du ld c,255
820    su dec c
830    jr nz,su
840    dec a
850    jr nz,du
860    djnz tu
870    call 54023
880    ld a,50
890    ld (54133),a
900    jp 53855
910    end
```

Essa rotina é executada quando nosso personagem morre — depois de se afogar no mar, cair num buraco, ser picado por uma cobra ou ser atingido e esmagado por uma pedra.

APAGANDO OS VESTÍGIOS

Antes de mandar Willie para o inferno, a rotina apaga seus vestígios na superfície. Se ele foi atingido por uma pedra, esta também será apagada, evitando a deformação de figuras. Para isso, coloca-se o padrão do céu, código 255, em cinco posições ao redor de Willie. Examinando todas as situações de colisão, você verá que essas são as posições necessárias.

O par HL é carregado com a posição de Willie na tela, que está no apontador em −5205 e −5204; DE é carregado com o endereço inicial da TN da VRAM e somado em HL, o apontador na impressão. Em seguida, a rotina 77 da ROM é chamada cinco vezes, com o valor de HL apontando para cinco diferentes posições ao redor de Willie. Lembre-se de que a rotina 77 — utilizada no decorrer do jogo para imprimir um padrão na tela — coloca o valor de A no endereço da VRAM apontado por HL.

DESCIDA PARA O INFERNO

Willie chega até o fundo da tela, em sua descida ao inferno. Para criar esse efeito, começamos por apagar sua cabeça — se não o fizermos, veremos no vídeo uma coluna de cabeças!

A posição de Willie, que aponta para a sua cabeça, é carregada da variável em −5205 e −5204 para HL, par de registros guardado na pilha. O endereço inicial da TN na VRAM é carregado em DE e somado em HL. A rotina 77 coloca o padrão 255 nessa posição, apagando a cabeça de Willie. A posição é recuperada da pilha para HL, sendo somada a 32 através de DE e colocada de volta em −5205 e −5204. Assim, Willie afunda uma posição na tela.

Esse novo valor é somado com o endereço inicial da TN na VRAM através de DE. O acumulador é carregado com 0, o código do primeiro padrão da figura de Willie com as pernas juntas. O apontador é colocado na pilha e a rotina 77 é chamada. O apontador é recuperado da pilha e somado com 32, movendo-se para a posição de baixo. A é carregado com 1, o código do segundo padrão da figura de Willie. Chama-se a rotina 77 e Willie desce uma posição.

Em seguida, B e A são carregados com 255. O acumulador é decrementado até 0 no laço **dl**. Quando chega a esse valor, o processador, por meio da instrução **djnz atr**, decrementa B, reajusta A com 255 e repete o processo até que B também seja 0. Esse laço é executado 255 x 255 vezes e tem como único fim atrasar a descida de Willie, para que o jogador possa vê-la.

A posição de Willie é carregada da variável em −5205 e −5204 para HL, onde é subtraída de 704 através de DE. Esse valor indica o começo da penúltima linha na tela. Como Willie tem dois caracteres de altura, a subtração irá colocar o valor 1 na baliza carry até que seus pés tenham tocado o fundo da tela. Enquanto a baliza tiver esse valor, a instrução **jr c,mo** manda o processador de volta ao início da rotina do enterro, que afunda Willie mais uma posição. Quando seus pés tocarem o chão, ele estará completamente enterrado; a baliza carry não terá mais o valor 1 e não haverá o salto. O processador passa, então, a executar a próxima instrução.

O número de vidas precisa ser reajustado — ou melhor, decrementado, pois o pobre Willie perdeu uma vida. O número de vidas é carregado no acumulador em −5221, decrementado e recolocado no mesmo endereço. Se ainda restam vidas a Willie, o processador salta para o programa de inicialização, em 53888. Caso contrário, continua na rotina de nova partida.

ACABARAM-SE AS VIDAS

Quando Willie tiver consumido suas cinco vidas, o jogo acaba. Devemos, então, imprimir "NOVA PARTIDA", com os padrões previamente colocados na tabela de padrões da VRAM pela rotina fornecida no artigo da página 1001.

O par DE é carregado com 234, posição da tela onde a primeira letra será impressa. A contém o código de padrão da primeira letra; B, o número de padrões (doze, incluindo o espaço em branco). O laço **tt** é executado doze vezes, usando a rotina 77 para imprimir a mensagem no centro da tela. Note que este trabalho foi bem simplificado, porque os padrões para a mensagem já estavam na VRAM, em seqüência.

Depois, o laço formado por **tu**, **du** e **su** é executado 30 x 255 x 255 vezes, promovendo o atraso necessário para a mensagem ser lida na tela. A rotina que imprime o escore em 54023 é chamada para imprimir o placar final.

Para reajustar o atraso geral do jogo, carregamos o acumulador com 50 e transferimos esse valor para 54133 — endereço que pertence à rotina final e controla a velocidade do jogo.

O processador salta para 53855, o início da rotina principal, para que você tente de novo, se quiser.

OPERAÇÕES COM CADEIAS

A programação dos blocos básicos de um processador de textos em BASIC não é difícil. Aprenda aqui a remover espaços em branco e a converter maiúsculas em minúsculas e vice-versa.

A linguagem BASIC tem excelentes recursos para programação com variáveis literais (alfanuméricas). Diversas funções de manipulação de cadeias de caracteres (*string*) garantem a essa linguagem uma grande versatilidade. Vamos relembrar, resumidamente, o que faz cada uma dessas funções.

Funções gerais

LEN(X$) - Retorna o comprimento da cadeia **X$** (um número inteiro entre 0 e 255). Existe para todas as versões de BASIC. Nos microcomputadores ZX-81 e Spectrum não é necessário colocar **X$** entre parênteses.

ASC(X$) - Retorna o código ASCII (um número inteiro entre 0 e 255) do primeiro caractere da cadeia **X$**. No micro Sinclair ZX-81 denomina-se **CODE**.

CHR$(X) - Tem a função inversa de **ASC**, ou seja, retorna o caractere correspondente ao código ASCII indicado pela variável **X**.

STRING$(X,Y) - Retorna uma cadeia com **X** caracteres iguais, com código ASCII estipulado por **Y**. Não existe para os microcomputadores das linhas Apple, Sinclair ZX-81 e Spectrum.

Funções de subcadeias

LEFT$ (X$,N) -Extrai os **N** primeiros caracteres da cadeia **X$**.

RIGHT$ (X$,N)-Extrai os **N** últimos caracteres da cadeia **X$**.

MID$ (X$,N,L) -Extrai **L** caracteres a partir da posição **N** da cadeia **X$**. Pode ser utilizado também no formato **MID$(X$,N)**, extraindo então todos os caracteres que existirem na cadeia **X$**, a partir da posição **N**.

Nos micros da linha Sinclair — ZX-81 e Spectrum — não são necessárias funções especiais para tratamento de subcadeias. Damos, abaixo, a equivalência entre a referência padronizada a subcadeias na linha Sinclair e as funções anteriores mencionadas.

X$(1 TO N)	**⇔ LEFT$(X$,N)**
X$(LEN X$-N TO+1)	**⇔ RIGHT$(X$,N)**
X$(N TO N+L-1)	**⇔ MID$(X$,N,L)**
X$(N TO)	**⇔ MID$(X$,N)**

INSTR (X$,Y$) - A função *instring* retorna a posição em que se está o caractere **Y$** (ou o primeiro caractere da cadeia **Y$**), dentro da cadeia **X$**.

Essa função pode ser simulada nos computadores onde não existe, por meio de um laço de repetição em que cada caractere em **X$** é extraído, sucessivamente, sendo comparado com **Y$**. Isso pode ser feito com a função **MID$(X$,I,1)**, onde **I** é o indicador do laço.

Funções de conversão

Números podem ser representados na forma de uma cadeia de dígitos, mais os sinais de ponto (.), mais (+), menos (-), e as letras E ou D (notação científica em precisão simples ou dupla, respectivamente. Esta última só existe nos micros das linhas TRS-80, TRS-Color e MSX). Para efetuar a conversão da representação numérica para cadeia, ou vice-versa, existem duas funções especiais em ASCII:

STR$(X) - Retorna a cadeia correspondente ao número **X**. Um espaço em branco é adicionado ao início da cadeia.

VAL (X$) - Efetua a operação inversa, ou seja, retorna o valor numérico correspondente à cadeia **X$**. Se **X$** não contiver um número, não ocorre erro: o valor que se retorna é igualado a 0.

Nem todas as versões do BASIC dipõem do conjunto completo de funções string. Além disso, faltam certas funções, necessárias para a programação avançada com cadeias (sobretudo em processamento de textos):

- eliminação de espaços em branco no começo ou no fim de uma cadeia;
- conversão de minúsculas para maiúsculas e vice-versa;
- extração de palavras de uma frase.

Neste artigo, examinaremos como resolver alguns dos problemas mencionados acima por meio de pequenas sub-rotinas, que podem ser acrescentadas a qualquer programa.

REMOÇÃO DE ESPAÇOS

Para muitas aplicações de processamento de cadeias, é preciso remover antes os espaços em branco existentes no começo ou no final de um string. A sub-rotina que apresentamos a seguir mostra como remover brancos à direita. Nela, a variável **A$** contém a cadeia que deve ser processada, e **B$**, o resultado da rotina:

```
1000 FOR I=LEN(A$) TO 1 STEP -1
1010 IF MID$(A$,I,1)<>" " THEN
1030
1020 NEXT I
1030 B$=LEFT$(A$,I):RETURN
```

```
1000 FOR I=LEN A$ TO 1 STEP -1
1010 IF A$(I TO I)<>" " THEN
GOTO 1030
1020 NEXT I
1030 LET B$=A$(TO I)
1040 RETURN
```

Para testar a rotina, acrescente este pequeno programa:

```
10 PRINT "ENTRE A CADEIA ";
20 INPUT A$
```

- MANIPULAÇÃO DE CADEIAS DE CARACTERES
- CONJUNTO DE FUNÇÕES STRING
- NOVAS ROTINAS DE PROCESSAMENTO DE TEXTOS
- ELIMINAÇÃO DE ESPAÇOS EM BRANCO
- CONVERSÃO PARA MAIÚSCULAS
- CONVERSÃO PARA MINÚSCULAS

```
30 GOSUB 1000
40 PRINT B$
50 GOTO 10
```

A linha 1000 da sub-rotina percorre a cadeia de caracteres do fim ao começo, recuando um caractere por vez. A linha 1010 verifica se o caractere é um espaço em branco. Enquanto for, o laço se repete. A ocorrência do primeiro caractere não branco causa a interrupção do laço e a extração dos **I** caracteres à esquerda (linha 1030).

Ao testar o programa nos micros das linhas TRS-80, TRS-Color, MSX, Apple ou TK-2000, não se esqueça de digitar a cadeia entre aspas, pois, normalmente, o comando **INPUT** já realiza a função de retirar os espaços colocados à direita da mesma.

Nas versões do BASIC que permitem funções programadas (**DEF FN**), a rotina anterior pode ser substituída por algo bem mais compacto:

```
5 DEF FNTB$(A$)=LEFT$(A$,INSTR(
A$+"  ","  ")-1)
```

Para fazer um teste, troque a linha 30 do programa de verificação por:

```
30 B$=FNTB$(A$)
```

Observe que a função só opera corretamente se não houver mais do que um espaço em branco entre as palavras de uma cadeia.

Para remover os espaços em branco à esquerda podemos usar a seguinte versão da rotina anterior:

```
1200 FOR I=1 TO LEN(A$)
1210 IF MID$(A$,I,1)<>" " THEN
1230
1220 NEXT I
1230 B$=MID$(A$,I):RETURN
```

```
1200 FOR I=1 TO LEN A$
1210 IF A$(I TO I)<>" " THEN
GOTO 1230
1220 NEXT I
1230 LET B$=A$(I TO)
1240 RETURN
```

Para testar a rotina, substitua a linha 30 do programa de verificação por **GOSUB 1200**.

Note que apenas a primeira e a última linhas da sub-rotina são diferentes, pois é necessário percorrer a cadeia do primeiro ao último caractere, até encontrar o primeiro não branco. Feito isso, eliminam-se todos os caracteres à direita.

MINÚSCULAS E MAIÚSCULAS

Outro recurso útil é a conversão de todas as letras minúsculas de uma cadeia em maiúsculas, ou vice-versa. Lembre-se de que o código ASCII de uma letra minúscula é igual ao código da maiúscula correspondente, mais o valor 32. O código de A, por exemplo, é 65, e o de a, 97. Portanto, para converter maiúsculas em minúsculas, precisamos apenas somar 32 ao código das letras — quando o código ASCII for menor do que 65, que é a letra A, ou maior do que 90, que é a letra Z, o caractere não deve ser convertido.

```
1300 B$=""
1305 FOR I=1 TO LEN(A$)
1310 C=ASC(MID$(A$,I,1))
1320 IF C>64 AND C<91 THEN
C=C+32
1330 B$=B$+CHR$(C)
1340 NEXT I:RETURN
```

```
1300 LET b$=""
1305 FOR i=1 TO LEN a$
1310 LET c=ASC a$(i TO i)
1320 IF c>64 AND c<91 THEN LET
c=c+32
1330 LET b$=b$+CHR$ c
1340 NEXT i
1350 RETURN
```

Não apresentamos uma versão para os compatíveis com o ZX-81, pois estes não têm letras minúsculas.

Para testar a rotina acima, substitua a linha 30 de nosso programa de verificação por **GOSUB 1300**.

O funcionamento da rotina é simples: o laço que vai de 1305 a 1340 percorre a cadeia de entrada, tomando um caractere de cada vez, na linha 1310. Se o seu código estiver entre 65 e 90, trata-se de uma letra maiúscula; portanto, devemos somar 32 ao mesmo. A linha 130 concatena o novo caractere à cadeia de saída.

Para converter minúsculas em maiúsculas basta alterar apenas uma linha da rotina:

```
1320 IF C>96 AND C<123 THEN
C=C-32
```

A rotina verifica se o código ASCII do caractere está compreendido entre 97 e 122 (letras a a z, minúsculas). Se estiver, simplesmente subtrai 32 desse código.

Como uma cadeia é armazenada na memória do computador?

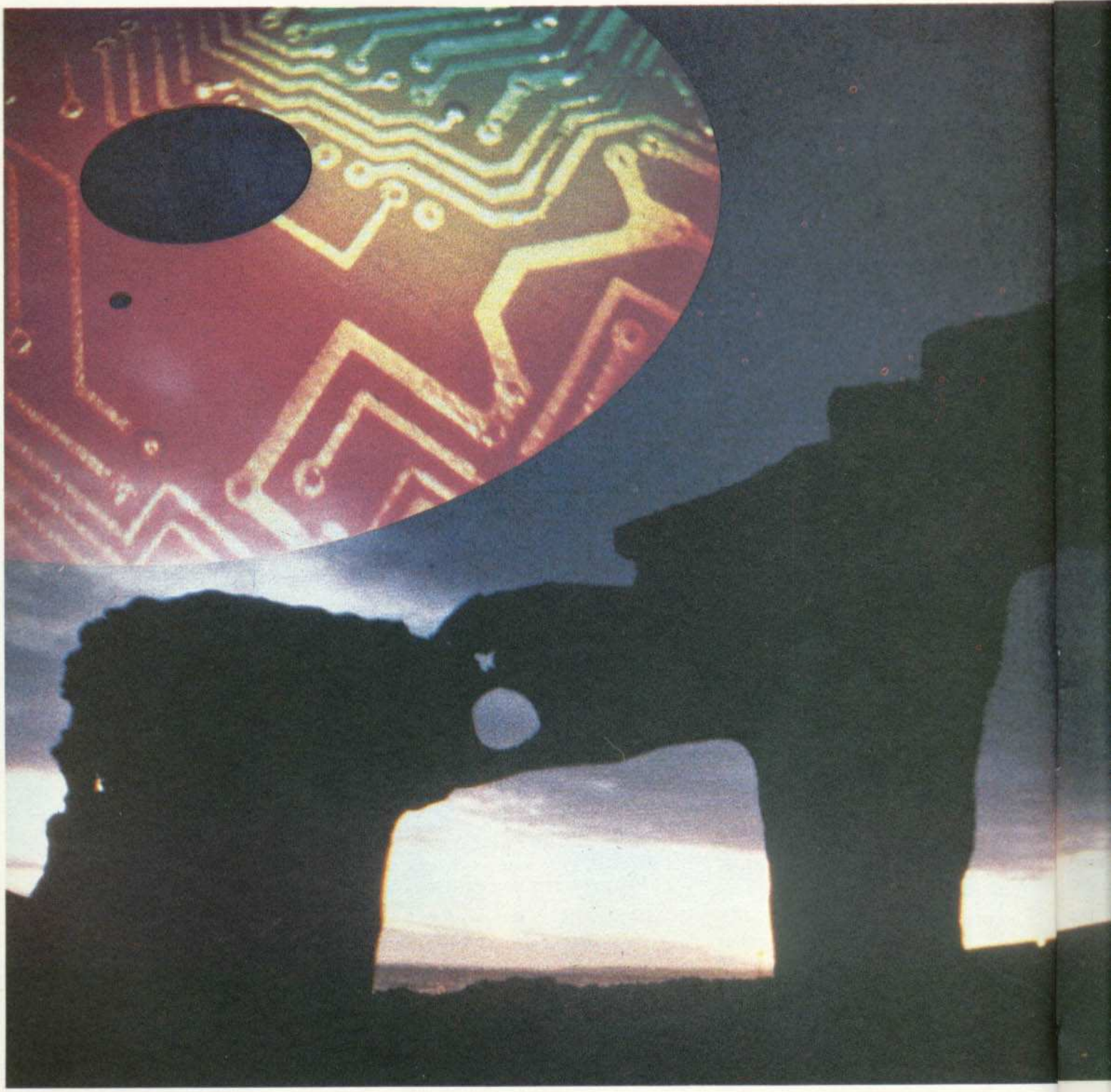
A cadeia é uma estrutura de dados muito útil, pois não exige um espaço fixo para armazenagem de cada variável literal, como acontece com as variáveis numéricas. Uma variável de precisão simples sempre ocupa quatro bytes na memória; uma variável de precisão dupla (quinze decimais) ocupa seis bytes e assim por diante.

Manter esse esquema de alocação fixa de espaço na memória para as cadeias levaria a um grande desperdício, pois o comprimento das cadeias usadas em um programa pode variar muito. Assim, o interpretador BASIC coloca todas as cadeias em um único espaço, reservado para esse fim.

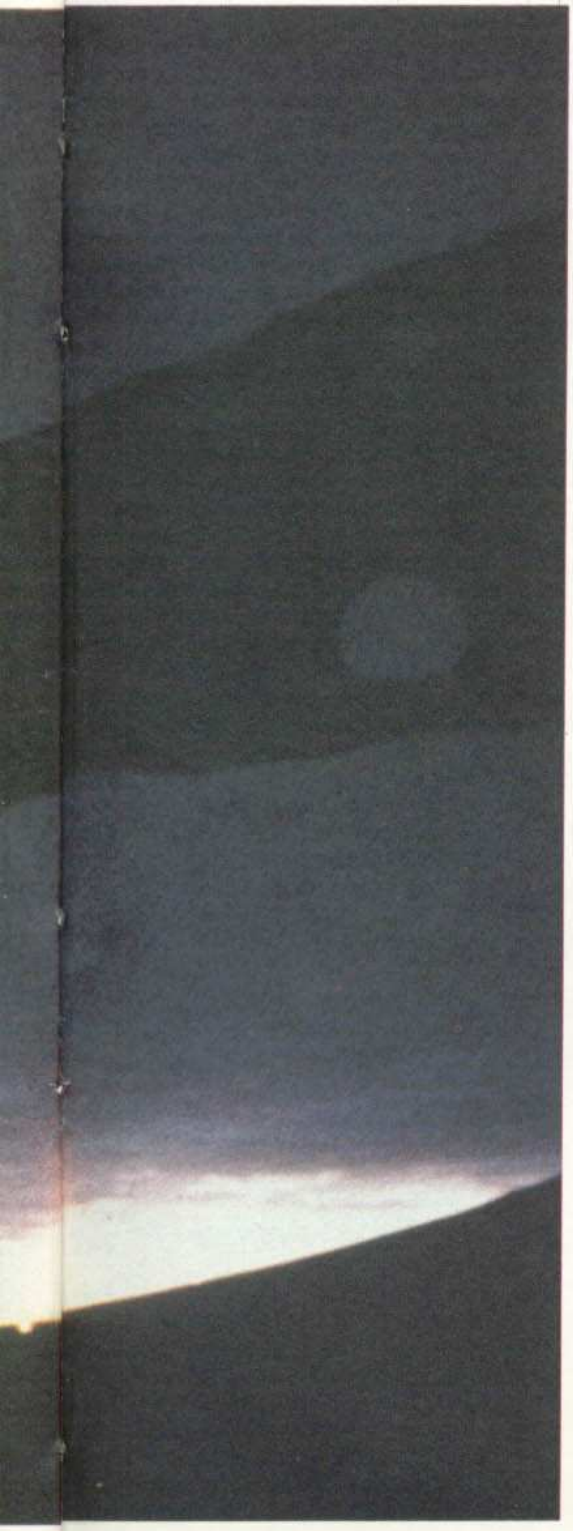
Para saber onde começa e acaba cada cadeia, o interpretador mantém duas outras estruturas de dados: uma série de apontadores indica as locações de memória onde começam as cadeias. No byte 0 da cadeia está indicado o seu comprimento, em bytes. É por isso que o comprimento máximo de uma cadeia, no dialeto Microsoft BASIC, é 255 (o maior número decimal representável em oito bits).

TRS-COLOR: UM EDITOR DE DISCOS

Com o programa apresentado neste artigo, os usuários do TRS-Color terão acesso direto à informação contida nos disquetes, podendo alterá-la ou recuperá-la em caso de acidente.



- ACESSO DIRETO
- LEITURA E GRAVAÇÃO
- FORMATO DO DISCO
- O DIRETÓRIO
- COMO UTILIZAR
- O PROGRAMA
- LEITURA DE UM SETOR
- EXECUTANDO AS ALTERAÇÕES



A tarefa de gravar e ler arquivos em um disquete é gerenciada pelo sistema operacional de disco (DOS). Em geral, o modo como se organiza a informação no disco ou como se dá a transferência de dados não constitui problema para o usuário — o DOS cuida de tudo, transformando a unidade de disquete em uma extensão do computador.

No entanto, podemos acessar partes individualizadas do disco, manipulando e alterando diretamente a informação nele contida. Este é um recurso oferecido pelo DOS que abre inúmeras possibilidades de aplicação. Permite, por exemplo, que se alterem itens de um diretório e nomes ou dados de arquivos. Sua aplicação mais interessante talvez seja a recuperação de informações: com o acesso direto você pode, entre outras coisas, recuperar arquivos que foram apagados, fechar arquivos ou restabelecer apontadores, corrigindo o encadeamento dos setores que compõem um arquivo em particular.

O acesso direto a blocos individuais de informação em um disco funciona de modo semelhante a um monitor destinado a examinar a memória do micro.

A operação do monitor de disco (ou editor de disco) é viabilizada, basicamente, pela armazenagem temporária, em uma parte da memória, da informação lida do disco ou a ser gravada. Enquanto está na memória, a informação pode ser alterada à vontade, sendo, em seguida, substituída no disco.

A GEOGRAFIA DO DISCO

Algum conhecimento sobre a distribuição dos dados no disco é fundamental para a manipulação do bloco de informações em uma trilha ou setor determinado. O diagrama, normalmente referido como formato do disco, é definido pelas rotinas de formatação do DOS.

Usa-se a notação em hexadecimal em todas as referências sobre acesso direto ao disco — por isso, nossas referências também adotarão essa notação. Como não empregaremos a notação decimal em nosso programa, será útil ter uma tabela de conversão de decimais para hex e outra de hex para ASCII.

TRILHAS E SETORES

Para utilizar nosso programa-editor, você deverá ter um mapa do formato do disco. Veremos aqui seus aspectos mais relevantes; outros detalhes serão encontrados no manual do DOS.

Se você vai trabalhar com um disco que reúne dados importantes, recomendamos que faça uma cópia dele. Qualquer erro cometido ao usar este programa poderá danificar toda a informação nele armanezada.

O TRS-Color tem uma interface especial que contém o sistema operacional. Este não precisa, assim, estar gravado no disco para ser lido quando se liga o computador — como ocorre com a maioria dos micros.

O disco divide-se em 34 trilhas, cada uma com dezoito setores de 338 bytes. Destes, apenas 256 são usados para armazenamento de dados; os 82 restantes são utilizados para controle do sistema operacional.

O diretório, que controla os arquivos do disco, fica armazenado na trilha 17. Ele pode ser acessado por meio do comando **DIR**, que mostra todos os arquivos, discriminando seu tipo e também o número de blocos empregados. Vejamos como está dividida a trilha 17, a partir do setor 3:

Byte	Conteúdo
0-7	Nome do arquivo. Ocupando um número de bytes menor que oito, é preenchido por espaços em branco. Byte 0 igual a 0 indica arquivo apagado. Byte 0 igual a FF significa que este e os próximos itens do diretório ainda não foram usados.
8-10	Extensão do nome do arquivo.
11	Tipo de arquivo: 0 programa BASIC; 1 arquivo de dados BASIC; 2 programa em linguagem de máquina; 3 arquivo de texto.
12	00 arquivo em formato binário. FF arquivo em formato ASCII.

13 Número do primeiro bloco do arquivo (0-67).

14-15 Número de bytes em uso no último setor do arquivo.

16-31 Sem uso.

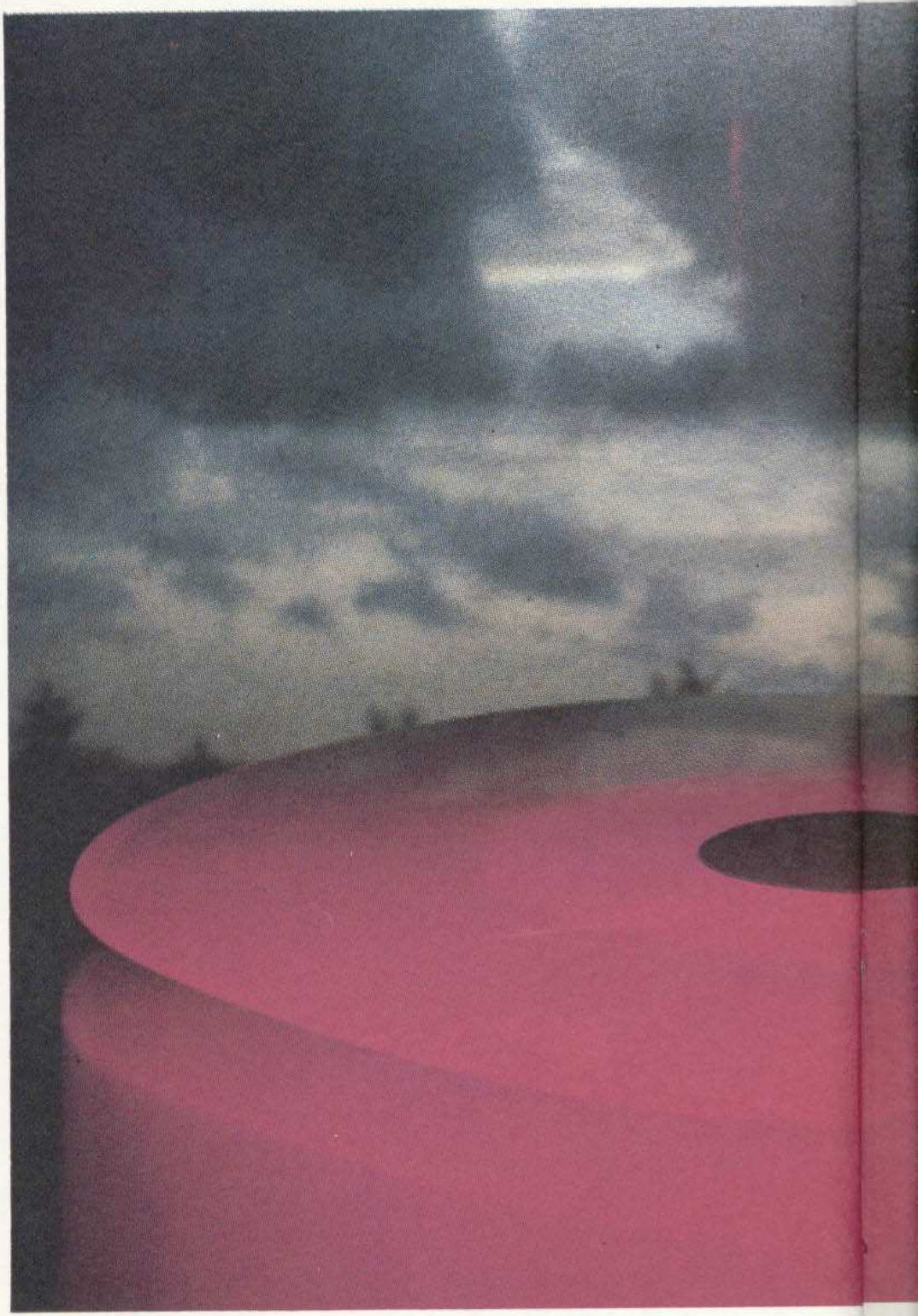
Para distribuir os arquivos no disco, o computador usa como referência *blocos*, que nada mais são que metade de uma trilha. Assim, o bloco 0 será composto pelos setores 1 a 9 da trilha 0; o bloco 1, pelos setores 10 a 18 da trilha 0; o bloco 2, pelos setores 1 a 9 da trilha 1 e assim por diante, até a trilha 16. A trilha 17 — que contém o diretório — não é levada em consideração, mas a divisão recomeça a partir da trilha 18 — onde os setores 1 a 9 compõem o bloco 34 — e, com isso, prossegue até o fim do disco...

A identificação dos blocos livres para uso é feita por meio do setor 2 da trilha 17. Os primeiros 68 bytes desse setor representam os blocos de 0 a 67. Quando determinado byte contém o valor FF, o bloco está livre. Um valor de 00 a 43 indica sua ocupação por um arquivo cuja referência é dada pelo número decimal obtido após a conversão do valor. Um valor de C0 a C9 indica que o bloco em questão é o único ou o último do arquivo; o segundo dígito especifica o número de setores que estão ocupados no bloco.

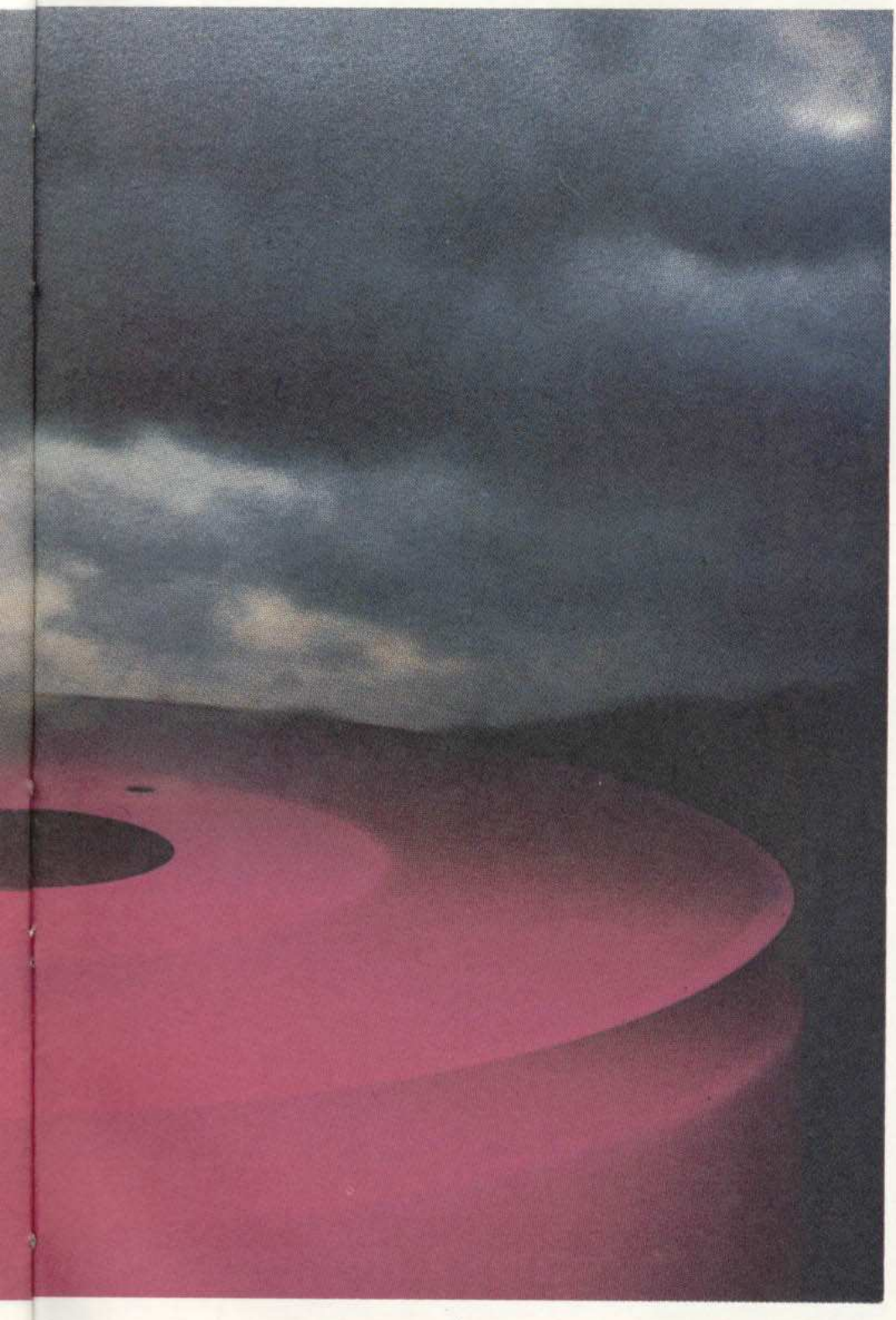
Essas informações são importantes, especialmente quando se trata de recuperar arquivos apagados por engano — tarefa, aliás, nada fácil, mas que pode ter bons resultados se realizada com um pouco de paciência.

No caso da recuperação de um arquivo pequeno, que não ocupa mais de um bloco, o trabalho é mais simples. Em primeiro lugar, você deve localizar o programa desejado no diretório. Para restaurar seu nome inicial, substitua o 00 da primeira posição pela letra adequada. Depois procure o 13º byte do item (o terceiro após a última letra da extensão do nome do arquivo). Transforme seu valor hexadecimal em decimal, para identificar o bloco usado pelo seu programa. Em seguida, grave essa informação no setor e carregue o setor 2 da trilha 17. Procure pelo byte correspondente ao bloco (lembre-se de que a contagem começa no 0). Ele deve ter o valor FF. Substitua esse valor por C9 e grave no setor. Saia do programa-editor, carregue o seu programa na memória e volte a gravá-lo, para que todos os apontadores sejam atualizados. Pronto, seu programa está totalmente recuperado!

Quando o programa que você quer recuperar é maior, a operação torna-se complicada, pois o DOS passa para FF o valor de todos os blocos usados pelo programa que foi apagado. Assim, começamos o trabalho conhecendo apenas o primeiro bloco. Os demais têm que ser procurados no disco, setor por setor. O programa só poderá ser carregado na memória do computador quando tivermos montado a seqüência correta de blocos. Inicie a busca pelos blocos vizinhos do inicial e vá se afastando até ter localizado todos eles. É um trabalho árduo, mas, certamente, haverá casos em que valerá a pena.

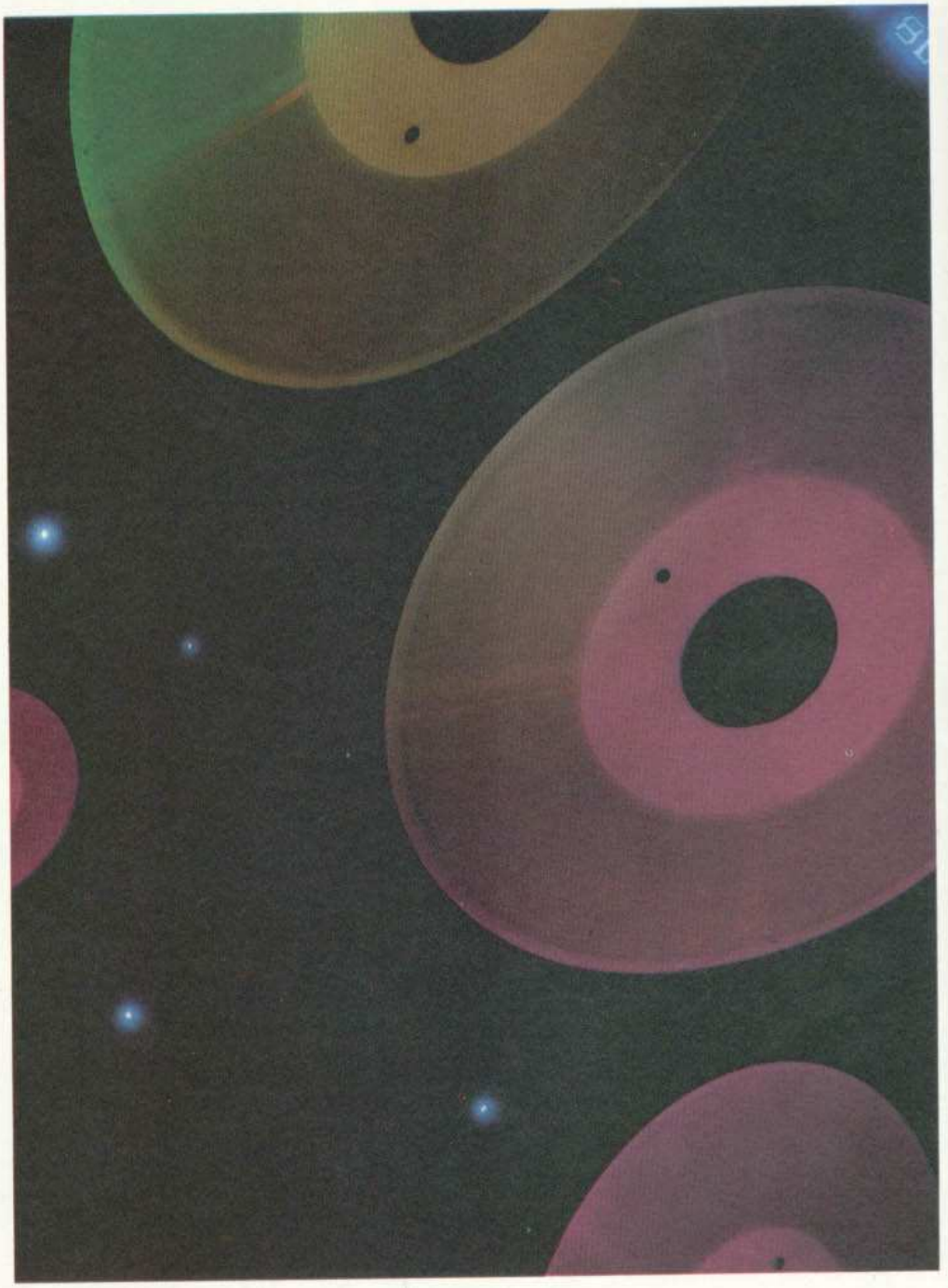


```
10 CLEAR 5000:DIM A$(1),D$(1),D
(160):C$="^"+CHR$(10)+CHR$(8)+C
HR$(9)+"AH"+CHR$(13)+" ":D=1
15 TP$(0)="BAS":TP$(1)="DAD":TP
$(2)="LMA":TP$(3)="TXT"
20 CLS:PRINT @13,"menu"
30 PRINT @105,"cARREGAR SETOR":
PRINT @169,"vER/EDITAR SETOR":P
RINT @233,"sALVAR SETOR":PRINT
@297,"dIRETORIO"
40 R$=INKEY$:IF R$="" THEN 40
50 R=INSTR("CVSD",R$):IF R=0 TH
EN 40
60 IF SL=0 AND (R=2 OR R=3)THEN
 PRINT:PRINT"NENHUM SETOR CARRE
GADO":FOR K=1 TO 2000:NEXT:GOTO
 20
70 CLS:ON R GOSUB 1000,2000,300
0,4000
80 GOTO 20
```



```
218 V$=CHR$(VAL("&H"+H$)):P=H+Y
*11+X
1000 SL=1:GOSUB 5000
1010 DSKI$ D,T,S,A$(0),A$(1)
1020 RETURN
2000 F=1:H=1:CLS:PRINT"aSCII OU
 hEX ?"
2010 R$=INKEY$:IF R$<>"A" AND R
$<>"H" THEN 2010
2020 AZ=0:IF R$="A" THEN AZ=1
2030 IF F=0 THEN 2050
2040 PK=96:CP=1535:IF AZ=1 GOSU
B 2320 ELSE GOSUB 2280
2050 POKE CP,PK:CP=1024+Y*32+X*
3:PK=PEEK(CP):POKE CP,239
2060 PRINT @321,"BYTE SUPERIOR=
";H
2070 R$=INKEY$:IF R$="" THEN 20
70
2080 R=INSTR(C$,R$):IF R=0 THEN
 2070
2090 F=0:ON R GOTO 2100,2110,21
20,2130,2140,2150,2160,2170
2100 Y=Y-1:GOTO 2210
2110 Y=Y+1:GOTO 2210
2120 X=X-1:GOTO 2210
2130 X=X+1:GOTO 2210
2140 AZ=1:GOTO 2040
2150 AZ=0:GOTO 2040
2160 RETURN
2170 PRINT @384,"NOVO CONTEUDO
(HEX) ";:INPUT H$
2180 V$=CHR$(VAL("&H"+H$)):P=H+
Y*11+X
2190 MID$(A$(P/128),P+128*(P>12
8),1)=V$
2200 F=1:GOTO 2030
2210 IF Y<0 THEN H=H-44:Y=0:F=1
2220 IF Y>7 THEN H=H+44:Y=7:F=1
2230 IF X<0 THEN X=10:Y=Y-1:IF
Y<0 THEN H=H-11:Y=0:F=1
2240 IF X>10 THEN X=0:Y=Y+1:IF
Y>7 THEN Y=7:H=H+11:F=1
2250 IF H= 10 OR H=-43 THEN H=1
:F=0:ELSE IF H<1 THEN H=1:F=1
2260 IF H=179 OR H=212 THEN H=1
68:F=0 ELSE IF H>168 THEN H=168
:F=1
2270 GOTO 2030
2280 CLS:FOR J=H TO H+87 STEP 1
1:FOR TT=0 TO 10
2290 PRINT RIGHT$("0"+HEX$(ASC(
MID$(A$(J/128),J+TT+128*((J+TT)
>128)))),2);" ";
2300 NEXT:PRINT CHR$(8);:NEXT
2310 RETURN
2320 CLS:FOR J=H TO H+87 STEP 1
1:FOR TT=0 TO 10
2330 G=ASC(MID$(A$(J/128),J+TT+
128*((J+TT)>128))):IF G<32 THEN
 2350
2340 PRINT " ";CHR$(G);" ";:GOT
O 2360
2350 PRINT LEFT$("0"+HEX$(G),2)
;" ";
2360 NEXT:PRINT CHR$(8);:NEXT:R
ETURN
3000 CLS:PRINT"SALVAR NO MESMO
SETOR (S/N) ?"
3010 R$=INKEY$:IF R$<>"S" AND R
$<>"N" THEN 3010
3020 IF R$="S" THEN 3040
 030 CLS:GOSUB 5000
```

```
3040 PRINT:PRINT"VOCE TEM CERTE
ZA (S/N) ?"
3050 R$=INKEY$:IF R$<>"S" AND R
$<>"N" THEN 3050
3060 IF R$="N" THEN RETURN
3070 DSKO$ D,T,S,A$(0),A$(1)
3080 RETURN
4000 GOSUB 5050
4010 PRINT #PR,TAB(14);"INICIO
NO."
4020 PRINT #PR,"  NOME    EXT T
IPO   DEL"
4030 FOR J=0 TO 8:DSKI$ D,17,J+
3,D$(0),D$(1)
4040 FOR K=1 TO 256 STEP 32
4050 GOSUB 6000
4060 IF ASC(V$)=255 THEN K=256:
J=8:GOTO 4120
4065 IF ASC(V$)=0 THEN MID$(V$,
1,1)=" ":DT=1
4070 PRINT #PR,MID$(V$,1,8);TAB
(9);".";MID$(V$,9,3);
4080 TP=VAL(MID$(V$,11,1))
4100 PRINT #PR,TAB(15);TP$(TP);
TAB(20)CHR$(42*DT)
4110 DT=0
4120 NEXT K,J:R$=INKEY$:IF PR=-
2 THEN 4140
4130 R$=INKEY$:IF R$="" THEN 41
30
4140 RETURN
5000 INPUT"NUMERO DA TRILHA (0-
34) ";T
5010 INPUT"NUMERO DO SETOR (1-1
8) ";S
5020 INPUT"NUMERO DO DRIVE (0-3
) ";D
5030 IF D>3 OR D<0 OR T>34 OR T
<0 OR S>18 OR S<1 THEN 5000
5040 RETURN
5050 PR=0:IF(PEEK(150))<>1 THEN
 RETURN
5060 PRINT"SAIDA PARA A IMPRESS
ORA (S/N)?"
5070 R$=INKEY$:IF R$<>"S" AND R
$<>"N" THEN 5070
5080 IF R$="S" THEN PR=-2
5090 RETURN
6000 V$=MID$(D$(K/128),K+128*(K
>128),32):IF LEN(V$)<32 THEN V$
=V$+MID$(D$(1+K/128),1,32-LEN(V
$))
6010 RETURN
```



COMO USAR O PROGRAMA

Carregue o programa na memória e escolha o disco no qual você vai trabalhar. Enquanto está aprendendo, convém utilizar um disquete que não contenha dados importantes.

Digitando **RUN**, você terá quatro opções: *Carregar setor* (**C**), *Ver/editar setor* (**V**), *Salvar setor* (**S**), *Diretório* (**D**). Pressione **D** para obter uma listagem detalhada dos arquivos do disco, incluindo os que foram apagados anteriormente. Estes estarão marcados com um asterisco na coluna DEL. Lembre-se de que a inclusão do nome de um arquivo no diretório não garante que ele esteja disponível no disco — os blocos usados por ele podem ter sido reutilizados por um outro arquivo. Se você tem uma impressora conectada ao Micro, digite **POKE 150,1** antes de executar o programa: a opção de imprimir o diretório lhe será oferecida.

Em seguida, digite a tecla **C** para carregar um setor. Tente primeiro o diretório, pedindo a trilha 17, setor 3. Digite **V** para visualizar os dados. Pressionando A ou H, você poderá obter, a qualquer momento, os dados no formato ASCII ou hexadecimal. O primeiro é o mais conveniente para a identificação de nomes de arquivos e linhas BASIC. O segundo deve ser usado quando se quer encontrar os valores hexadecimais de um determinado byte.

As setas movem o cursor para o ponto que se desejar. Para alterar um valor, pressione a barra de espaços e forneça o novo byte, sempre em hexadecimal. Note que apenas uma parte do setor é exibida. À medida que se movimenta o cursor para baixo, a tela vai rolando e mostrando o que resta.

Ao terminar a edição de um setor, pode-se gravá-lo usando a opção **S** do menu. Entre as informações solicitadas pelo programa... e pronto!

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO BASIC

O que é recursão? A máquina inteligente. Procedimentos recursivos. Limitações. Como evitar erros.

CÓDIGO DE MÁQUINA

Níveis de dificuldade em *Avalanche*. Aceleração do jogo. Contagem de pontos. Exibição do placar. De volta ao sopé.

PROGRAMAÇÃO DE JOGOS

Pôquer com dados: as regras do jogo, estratégia, UDG dos dados, rolando os dados, o placar.