

Se se cansou da lentidão dos seus programas em BASIC, e se deseja escrever programas mais rápidos e melhores — em particular de jogos — então este livro é-lhe destinado.

Este trabalho parte do princípio de que não possui conhecimento da linguagem máquina. E começa por explicar quais são os equivalentes em linguagem máquina para instruções BASIC tais como LET, IF, FOR/NEXT, PRINT, GOSUB, ADDITION e SUBTRACTION. Prossegue com a descrição pormenorizada das rotinas individuais dum simples jogo do Invasor Espacial, e de como essas rotinas estão interligadas. Para terminar são indicadas algumas rotinas simples que podem ser usadas com programas BASIC. São também incluídos alguns jogos para demonstrar a eficácia da programação em linguagem máquina.

- Aprenda como é simples incorporar nos seus programas BASIC rotinas em linguagem máquina.

- Use a linguagem máquina para criar muito melhores efeitos sonoros do que os que podem ser conseguidos com o comando BASIC.

Quer pretenda simplesmente escrever uma rotina para melhorar os seus programas BASIC, quer seja para escrever um programa em linguagem máquina com pés e cabeça, este livro será muito útil para si. Ele elimina a teoria que envolve a linguagem máquina, apresentando-a de um modo prático e agradável de aprender.

Há no decurso do livro um certo número de perguntas que permitirão testar os seus conhecimentos de linguagem máquina.

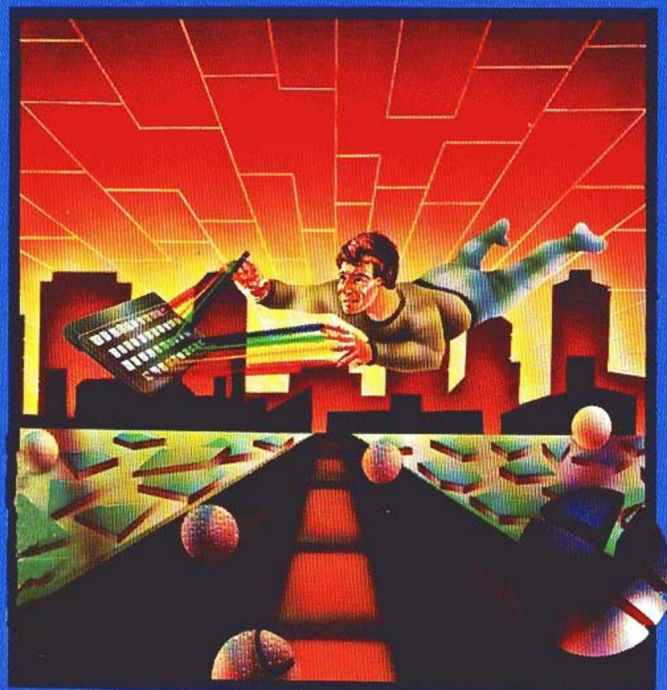
ARTE
DE
VIVER.

STEVE WEBB

São prática em LINGUAGEM MÁQUINA para o Spectrum

E.A.

Programação prática em LINGUAGEM MÁQUINA para o Spectrum



PUBLICAÇÕES EUROPA-AMÉRICA

STEVE WEBB

**Programação prática em
LINGUAGEM MÁQUINA
para o Spectrum**

*ARTE
DE
VIVER.*

PUBLICAÇÕES EUROPA-AMÉRICA

Titulo original: Practical Spectrum Machine Code Programming

Tradução do Eng.º Eduardo Mesquita de Abreu

Capa: estúdios P. E. A.

© 1984 Steve Webb
First published in Great Britain
by Virgin Books Ltd.

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

ÍNDICE

	Pág.
Introdução	9
Que é a linguagem máquina?	11
Instruções em BASIC e os seus equivalentes em linguagem máquina	17
Rotinas em linguagem máquina (Programa «Invasor Espacial»)	43
Dois jogos e algumas rotinas úteis	72
Apêndice	111

Editor: Francisco Lyon de Castro

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

Edição n.º 33 089/3897

Execução técnica:
Gráfica Europam, Lda.,
Mem Martins

Introdução

Este livro tem a intenção de constituir uma introdução ao mundo da programação em linguagem máquina. O leitor não precisa de ter nenhum conhecimento prévio de linguagem máquina para ser capaz de usar as rotinas e os jogos nele contidos. Deve no entanto necessitar dum conhecimento sobre o funcionamento da programação BASIC. E por tal eu quero significar que deve ser capaz de imprimir o seu nome e fazê-lo mover-se através do *écran* em jeito de comboio expresso.

Para todos aqueles que pretendem tornar-se programadores a sério de linguagem máquina, cada rotina é acompanhada de uma descrição pormenorizada e dum fluxograma.

Se você não pretende tornar-se programador de linguagem máquina, não há razão para que não use algumas rotinas que vêm no livro para valorizar a sua programação em BASIC.

Esta obra tem a finalidade de ensinar os funcionamentos da programação em linguagem máquina e permitir-lhe aprender como desenvolver os seus próprios programas. Se bem que tenha incluído no texto alguns jogos, fi-lo mais com a finalidade de ensinar do que divertir. Se os utilizar simplesmente com o fim de jogar ficará desapontado.

Espero sinceramente que o leitor ache este livro mais prático que a maioria dos livros do tipo «Aprenda Linguagem Máquina» disponíveis.

As rotinas e os programas contidos no livro não foram escritos da maneira mais eficiente ou de modo a poupar a memória do computador. Muitos deles são mais longos do que é necessário, com a finalidade de os tornar mais compreensíveis. À medida que você se

Não é com frequência que tenho a oportunidade de, em público, agradecer aos meus amigos. Por isso peço que me seja perdoada esta página de auto-indulgência.

Quero agradecer à Susan por ter dactilografado este trabalho e ao Tracy por tudo o que fez.

tornar mais proficiente em programação será capaz de criar os seus próprios métodos mais abreviados.

No decurso deste livro o leitor depara com várias perguntas. Deve tentar dar-lhes resposta (estas podem ser encontradas no fim do livro); se verificar que errou, deve voltar a ler a secção correspondente, até que seja capaz de responder correctamente.

Que é a linguagem máquina?

O BASIC é uma linguagem de programação que tem por fim permitir ao homem comunicar com o funcionamento interno de um computador; pode ser olhado como um *interface* (um tradutor) entre o homem e o computador. O BASIC é extremamente inclinado para o homem e, acredite-se ou não, é mais fácil para si entendê-lo do que é para o computador.

O BASIC é uma língua estrangeira para o computador. A sua língua mãe é a linguagem máquina. Cada vez que ao computador se depara uma instrução em BASIC ele precisa do equivalente a um intérprete para o compreender. E isto leva tempo. Se um programa for escrito em linguagem máquina, o computador agirá de acordo com as instruções quase instantaneamente, poupando assim um tempo considerável. Os programas escritos em linguagem máquina são geralmente cinquenta a cem vezes mais rápidos do que os equivalentes programas em BASIC.

Os números e o seu armazenamento na memória

Creio que uma das coisas mais importantes a compreender antes de aprender linguagem máquina é como estão armazenados na memória os números. A memória é feita de locais individualmente numerados.

Cada local pode conter um número entre 0 e 255, inclusive. O facto de só poder conter um número com o valor máximo 255 é

obviamente uma limitação e tem de se encontrar um método para poder armazenar números de valor superior. O exemplo seguinte mostra como isto se faz.

Digamos que o número que se quer armazenar é 29248. Em primeiro lugar dividimos este número por 256 e arredondamos o resultado por defeito para o número inteiro mais próximo.

Assim: 29248 dividido por 256 = 114,25, arredondado por defeito = 114. A este chamamos a componente superior do número a ser armazenado. Representa o número total de vezes que o número 256 se contém no número em questão.

A componente superior (114) deste exemplo é então multiplicada por 256 e o resultado é em seguida subtraído do número que queremos armazenar (29248).

$$\begin{aligned} 114 \times 256 &= 29184 \\ 29248 - 29184 &= 64 \end{aligned}$$

64 chama-se a componente inferior do número que pretendemos armazenar.

Para o fazer pomos a componente inferior, que é 64, numa localização da memória e a componente superior, 114, na seguinte.

Para se armazenar um número de valor superior a 255 precisamos de utilizar duas posições de memória. Assim, se ler neste livro que um número, como por exemplo 29248, está armazenado na posição 30000, já sabe que na realidade ele se encontra armazenado nas posições 30000 e 30001.

Quais são as componentes superior e inferior do número 45621?

Se a componente inferior de um número for 31 e a sua componente superior tiver o valor 64, de que número se trata?

Quando damos entrada a um programa em linguagem máquina não usamos o nosso habitual sistema de numeração decimal, mas sim um outro que se designa por HEXADECIMAL (e que abreviadamente se designa por HEX). Para dar uma ideia como é o HEX veja-se este quadro, onde temos os números decimais e os seus equivalentes em HEX.

DECIMAL	HEX
0	00
9	09
10	0A
15	0F
16	10
255	FF

Anexa a este livro há uma tabela completa dos HEX e seus equivalentes em números decimais.

Para evitar confusões usa-se neste trabalho a letra D para significar número decimal e H para o HEX. E assim

$$\begin{aligned} 8d &= 8 \text{ decimal} \\ 12h &= 12 \text{ HEX} \end{aligned}$$

Qual é o número decimal equivalente a E3h?

Se FBh é a componente superior dum número e CBh a sua componente inferior, qual é o seu equivalente no sistema decimal?

Para dar as respostas a estas duas perguntas deve usar-se a tabela do anexo.

Escrevemos um programa especial para permitir ao leitor dar facilmente entrada e testar os seus programas em linguagem máquina, ao qual nos referiremos como *Hexent*. Segue-se uma listagem do Hexent que deve ser batido no teclado e cuidadosamente verificado, guardando-o em seguida usando

SAVE "HEXENT" LINE 5

```

5 CLEAR 28999
10 CLS
15 PRINT AT 5,0; "Press key E
to input HEX code."
20 PRINT AT 7,0; "Press key C
to check HEX code."
35 PRINT AT 9,0; "Press key X
to start HEX count."
40 PRINT AT 11,0; "Press key Q
to stop program."

```

```

45 IF INKEY $="e" OR INKEY $="E"
THEN GO TO 155
50 IF INKEY $="c" OR INKEY $="C"
THEN GO TO 70
55 IF INKEY $="x" OR INKEY $="X"
THEN GO TO 265
60 IF INKEY $="q" OR INKEY $="Q"
THEN STOP
65 GO TO 45
Q 70 LET AS="0123456789ABCDEF"
75 CLS
80 PRINT "INPUT STARTING ADDRESS."
85 INPUT A
90 PRINT "INPUT FINISHING ADDRESS."
95 INPUT B
100 CLS
105 FOR C=A TO B
110 LET D=PEEK C
115 LET E=INT(D/16)
120 LET F=D-16*E
125 PRINT C; " "; AS(E+1); AS(F+1)
130 IF INKEY $="M" OR INKEY $="m"
THEN GO TO 15
135 NEXT C
140 PRINT AT 5,12; "PRESS M FOR
MENU."
145 IF INKEY $="M" OR INKEY $="m"
THEN GO TO 10
150 GO TO 145
155 CLS
160 PRINT "PUT CAPS LOCK ON."
165 PRINT "INPUT STARTING ADDRESS."
170 INPUT X
175 PRINT "INPUT HEX CODE."
180 LET AS=""
185 IF AS="" THEN INPUT AS
190 IF AS="M" THEN GO TO 10
195 LET B=0: LET A=1
200 FOR D=1 TO 8 STEP -1
205 LET C=CODE AS(A)
210 LET A=A+1
215 IF C>=48 AND C<=57 THEN LET
C=C-48: GOT TO 230
220 IF C>=65 AND C<=70 THEN LET
C=C-55: GOT TO 230
225 GO TO 180
230 LET B=B+C*16+D
235 NEXT D
240 POKE X,B
245 PRINT X; " "; CHR$(CODE AS(1));
CHR$(CODE AS(2))
250 LET X=X+1
255 LET AS=AS (3 TO)
260 GO TO 185
265 CLS
270 PRINT "INPUT STARTING ADDRESS."
275 LET D=0
280 INPUT A

```

```

285 PRINT "INPUT END ADDRESS."
290 INPUT B
295 FOR C=A TO B
300 LET D=D+PEEK C
305 NEXT C
310 PRINT "HEX COUNT = "; D
315 PRINT "PRESS M TO RETURN TO
MENU."
320 IF INKEY $="M" OR INKEY $="m"
THEN GO TO 10
325 GO TO 320

```

Como utilizar o Hexent

Quando se carrega o Hexent ele executa-se automaticamente e aparece (em inglês)* o seguinte menu:

Pressionar a tecla E para dar entrada ao código HEX.
 Pressionar a tecla C para verificar o código HEX.
 Pressionar a tecla X para iniciar a contagem HEX.
 Pressionar a tecla Q para parar o programa.

Para dar entrada à linguagem máquina deve bater-se a tecla E. Quando o tiver feito o computador pedirá que pressione a tecla que fará com que a impressão seja em maiúsculas. Depois deve fazer-se o «input» da posição de memória em que se deseja que se inicie a linguagem máquina. Feito isto, logo surgirá o indicativo « » de pronto a funcionar, e significando que o *Spectrum* aguarda agora que alguma coisa dê entrada.

Na primeira rotina a primeira linha a entrar é 2108 3E (o que dá três bytes de Hex). Impressa esta linha deve pressionar-se ENTER. O que tiver sido batido no teclado aparece então no canto superior esquerdo do *écran*, juntamente com as posições de memória em

* Por motivos técnicos, optou-se por manter em inglês as mensagens (o que está escrito entre aspas, « »), nas listagens de programas. O leitor já saberá que o que está escrito entre aspas não afecta o trabalho do computador. Por outro lado, decidiu-se aceitar como linguagem normal termos como *byte*, *opcode*, etc., que, não pertencendo ainda à nossa língua, são usados correntemente. (N. do E. port.)

que os bytes foram colocados. Prossegue-se de modo semelhante dando entrada ao resto das linhas. E, quando se deu entrada à última linha, tem de se bater a tecla M e depois ENTER, o que o fará voltar ao menu.

Se agora se pressionar a tecla C entrar-se-á na rotina de verificação, que lhe pedirá os endereços do início e do fim da linguagem máquina a serem verificados. Esta imprimirá os endereços e a linguagem máquina, que pode então ser verificada com as listagens para correcção. Para continuar a proceder à verificação do resto do programa tem de se pressionar a tecla C. No final bate-se a tecla M para voltar ao menu de controlo.

Agora é a vez de se pressionar X, o que dará início à verificação de contagem do HEX. Primeiro surgirá o pedido para dar entrada aos endereços inicial e a final da linguagem máquina a ser verificada. O Hexent adicionará então os valores dos bytes entre essas posições e imprimirá o total. No fim de cada rotina indicou-se qual deve ser esse valor. O total para a primeira rotina é 1585. Isto serve como uma dupla verificação de que se deu entrada correctamente à linguagem máquina.

É importante uma verificação exhaustiva, dado que ao contrário do BASIC que pára se houver erro, se a linguagem máquina tiver erro dará como resultado que o *Spectrum* aborte e provavelmente terá de se desligar antes de prosseguir. Como precaução adicional deve também guardar-se o programa em linguagem máquina antes de na realidade o executar.

Se se pressionar Q o Hexent parará, o que permite guardar a linguagem máquina e/ou escrever pequenos programas em BASIC acima da linha 325 para testar a linguagem máquina.

Quando se está a ler ou a dar entrada à linguagem máquina não se pode ter a certeza se se faz entrar a letra l ou o número um ou a letra O ou o número zero. Para apurar isto lembre-se que em HEX não temos as letras l e O.

Instruções em BASIC e os seus equivalentes em linguagem máquina

Nesta secção vamos indicar os equivalentes em linguagem máquina às mais correntes instruções em BASIC. Estes equivalentes mostrarão que a linguagem máquina não é tão difícil como se pode ter acreditado.

Na programação BASIC estávamos acostumados a usar variáveis tais como A, B, C, ..., X, Y, Z. Na programação em linguagem máquina não há tais variáveis e as suas mais próximas equivalentes são os registos. Há muito poucos registos e aqueles que principalmente se utilizarão são rotulados:

A, B, C, D, E, H, L

Cada registo pode ser considerado como uma posição específica da memória e consequentemente só pode conter um número entre 0 e 255. Para podermos usar os registos para armazenar números superiores a 255, seis deles foram constituídos num grupo de três pares, como se segue:

H L
B C
D E

Esta formação de pares não nos impede de usar os registos individualmente.

Todos estarão familiarizados com as instruções BASIC tais como $LET A=5$. O equivalente em linguagem máquina é $LD A,5$, que se lê como «carregar o registo A com 5». $LD A$ chama-se um *opcode*¹. No «Anexo» há uma lista de opcodes. Cada um deles é representado por um número HEX. O HEX para $LD A,X$ é $3EX$, onde X representa um número que pode ir de 0 a 255. A instrução $LD A,5$ ocupa duas posições de memória. Se $LD A,5$ for a primeira instrução duma rotina em linguagem máquina que começa na posição 30000, então, se atentarmos nas duas primeiras posições usando o Hexent, veremos que:

```
30000  3E
30001  05
```

É impossível ter uma instrução tal como $LD A,827$, pois que 827 é maior do que o número máximo, 255, que um simples registo pode armazenar. Se pretendermos armazenar um número maior do que 255 temos de utilizar um dos três pares de registos, tal como

LD HL, 827

Já mencionámos que cada registo individual pode ser considerado como uma posição de memória. Se nos lembrarmos como os números são armazenados, sabemos que a componente superior de $827=3$ ($827:256$, arredondado o quociente por defeito) e que a componente inferior é 59.

Quando executarmos uma instrução como $LD HL,827$ a componente superior, 3, é colocada no registo H e a componente inferior, 59, vai para o registo L.

No par de registos BC, B é a componente superior; no registo DE, a componente superior é D.

Os opcodes podem ter o comprimento de um ou dois bytes e podem ser seguidos por mais um ou dois bytes adicionais. Há mais de 600 variantes de opcodes e está fora do âmbito desta obra dar uma descrição de cada um deles. Contudo, no «Anexo» encontra-se uma lista completa dos opcodes. No decurso deste livro aprender-

-se-á os opcodes suficientes para permitir ao leitor escrever os seus próprios programas.

Mostraremos seguidamente as instruções BASIC mais frequentemente usadas e quais são os seus equivalentes em linguagem máquina.

```
BASIC : Let A = A + 3
OPCODE : ADC, x
HEX : CE 03
```

```
BASIC : Let A = B
OPCODE : LD A, B
HEX : 78
```

```
BASIC : Let A = A - 3
OPCODE : SBC A, x
HEX : DE 03
```

BASIC

```
For A = 1 to 100
  Rotina a fazer
  100 vezes
Next A
```

OPCODE

```
LD A, 100
→ Rotina a fazer
100 vezes
SBC A, 1
JRNZ
```

HEX

```
3E 64
Rotina a fazer
100 vezes
DE 01
20 x {— número de bytes a saltar
      para chegar ao início da rotina)
```

Para simular um ciclo FOR/NEXT carregamos primeiro o registo A com 100, ou fazemo-lo tantas vezes quantas as necessárias para realizar a rotina dentro do ciclo. No fim da rotina subtraímos

¹ Abreviatura de «operational code». (N. do T.)

1 do registo A, e fazemos a verificação para ver se A é zero. Isto faz-se com o opcode JRNZ, o qual significa salto relativo se o último cálculo efectuado não teve resultado zero, JRNZ é então seguido pelo número de bytes que se deve saltar para chegar ao começo da rotina. Digamos que a rotina que tem de ser executada 100 vezes consiste em 50 bytes. Com o fim de saltar para trás para o seu começo temos de fazer um salto negativo de 54 bytes, a saber, os 50 bytes da rotina mais os 2 bytes de SBC A,1, mais ainda 2 bytes do JRNZ. O HEX para JRNZ — 54 é 20 CA.

No fim do livro há um «Apêndice» onde estão todos os códigos HEX para saltar negativamente (isto é, para trás). Veremos que só se pode saltar para trás o máximo 128 bytes. Também não se pode saltar para a frente mais do que 127 bytes. Com o fim de ajudar o leitor a utilizar o «Apêndice» no fim do livro, damos a seguir alguns exemplos de saltos relativos e dos seus códigos HEX:

Salto relativo + 49	18 31
Salto relativo — 49	18 CF
Salto relativo — 94	18 A2
Salto relativo + 94	18 5E

Se um opcode é 18 E5, quantos bytes teremos de saltar, negativa (para trás) ou positivamente (para a frente)?

Qual é o opcode para JR + 98?

A finalidade da utilização da linguagem máquina é tirar partido da sua alta velocidade. No entanto acontece que, por vezes, é demasiado rápida para certas aplicações e é preciso introduzir um processo que permite reduzir a velocidade. A rotina anterior é excelente para produzir um retardamento e seria usada do seguinte modo:

LD A, 100	3E 64
SBC, 1	DE 01
JRNZ	20 FC (-4)

Isto pode ser considerado como o equivalente em linguagem máquina do que em BASIC é a instrução de pausa cuja duração é regulada pela carga inicial do registo A.

Notar-se-á provavelmente que, nos exemplos dados dos equivalentes em linguagem máquina, usámos constantemente o registo A e não fazemos uso de nenhum dos outros registos. Isto acontece porque A é o único registo do qual se pode directamente subtrair ou adicionar um número. Não podemos ter opcodes como os seguintes

```
ADC H, 1
SBC H, 1
    ou
SBC B, 3
```

Se quisermos adicionar um número, por exemplo 8, ao registo H, temos primeiro de carregar o registo A como o número 8 e seguidamente juntar os dois registos.

```
ADC A, H
```

A resposta aparecerá no registo A.

Pode-se adicionar ou subtrair qualquer dos seis registos, B, C, D, E, H, L, ao ou do registo A, mas lembremo-nos de que a resposta aparecerá sempre no registo A.

Podem-se adicionar ou subtrair pares de registos, mas somente ao ou do registo HL, com a resposta a aparecer neste. Assim, poderemos ter

```
HL + DE
HL + BC
```

mas não

```
DE + BC
BC - DE
```

Não se pode somar ou diminuir um registo simples a ou dum registo par.

Sugere-se que se evite efectuar cálculos em que se espera que o resultado seja um número negativo. Claro que se pode trabalhar com números negativos, mas sugerimos que os principiantes os evitem.

BASIC	:	GOTO 265
OPCODE	:	JP x x
HEX	:	C3 x x

O leitor já terá tomado consciência de que em linguagem máquina não temos números de linha. Se queremos «ir para» qualquer parte temos de especificar a posição particular na memória. Suponhamos que no fim de um jogo queremos ir para o início do mesmo, na posição 30000. O opcode completo para o fazer é

C3	30	75
	componente	componente
	inferior	superior
	de 30000	de 30000

Em todos os opcodes apropriados, os números são escritos sempre da maneira que acima se indicou — isto é, a componente inferior seguida da componente superior.

```

BASIC      : GOSUB 1500
OPCODE     : CALL x x
HEX        : CD x x

```

O equivalente a GOSUB em linguagem máquina é CALL. Uma vez mais, como sucedeu com GOTO, temos de chamar uma posição específica de memória. Assim como com as sub-rotinas em BASIC, uma sub-rotina em linguagem máquina tem de terminar com um opcode de retorno.

O opcode completo para chamar uma sub-rotina na posição 30000 é

CD 30 75

Na posição 30000 podemos ter a rotina para abrandamento que foi previamente descrita:

```

3E 64
DE 01
20 FC
C9

```

Como se vê, ela termina com C9, que é o opcode para retorno.

BASIC: IF

Já se nos deparou o equivalente em linguagem máquina para «IF» sob a forma de Salto Relativo se o último resultado não é zero.

Há muitas mais variações de «IF», existindo uma lista no «Apêndice» dos opcodes na parte final deste livro. Damos a seguir uma descrição de algumas das variações que mais provavelmente se utilizarão.

J R Z	Se o último resultado calculado for zero, fazer salto relativo de x bytes
J P Z	Se o último resultado calculado for zero, saltar para o endereço x x
J P N Z	Se o último resultado calculado não for zero, saltar para o endereço x x
J P M	Se o último resultado calculado for negativo, saltar para o endereço x x
J P P	Se o último resultado calculado for positivo, saltar para o endereço x x

Também podemos ter CALLs condicionais (GOSUBs).

CALL NZ	Chamar o endereço x x se o último resultado calculado não for zero
CALL Z	Chamar o endereço x x se o último resultado calculado for zero
CALL M	Chamar o endereço x x se o último resultado calculado for negativo
CALL P	Chamar o endereço x x se o último resultado calculado for positivo

BASIC: LET A = PEEK 30000

Quando se programa em BASIC, PEEK e POKE são as instruções que mais se aproximam normalmente da linguagem máquina. O opcode para LET A = PEEK 30000 é

```

LD A, (30000)
HEX = 3A 30 75

```

Os parênteses curvos à volta de 30000 significam que estamos a carregar o registo A com o que estiver contido na posição 30000. Assim, se esta posição 30000 contiver o número 81 e executarmos LD A, (30000), o registo conterà agora 81.

Podemos também usar os três pares de registos para PEEK em posições.

LOAD HL, (30000)
HEX = 2A 30 75

Isto significa carregar o registo L com o que estiver contido na posição 30000 e carregar o registo H com o conteúdo da posição 30001. Você será capaz de ver como isto se liga com o modo como os números são armazenados na memória e em registos pares.

Se a posição 30000 contém o número 5d e a posição 30001 contém o número 15d, qual será o valor total de HL após a instrução LD HL, (30000)?

O oposto a PEEK é POKE. O opcode para POKE 30000,A é:

LD (30000), A
HEX 32 30 75

Os parênteses curvos à volta de 30000 indicam que estamos a carregar a posição 30000 com o que estiver contido no registo A.

Podemos também usar POKE em posições com o valor contido por qualquer dos três registos pares.

LD (30000), HL
HEX 22 30 75
LD (30000), DE
HEX ED 53 30 75
LD (30000), BC
HEX ED 43 30 75

Depois da instrução LD (30000), HL, a posição 30000 conterà o valor do registo L e a posição 30001 o valor do registo H.

Se HL contém o valor 35621d, quais serão os valores decimais contidos nas posições 30000 e 30001 depois de se ter dado a instrução LD (30000), HL?

Em BASIC podemos ter o seguinte

LET A = 5
LET B = 30000
POKE B, A

o que significa que podemos POKE (escrever) uma variável por via de uma outra. Podemos fazer isto facilmente em linguagem máquina:

OPCODE LD(HL), A
HEX 77

Digamos que HL contém o valor 30000 e que o registo A tem um valor de 5. Após a instrução LD(HL),A a posição 30000 conterà o valor 5.

Em BASIC, o oposto disto é

LET A = PEEK B

e é fácil de fazer em linguagem máquina:

LD A, (HL)
HEX 7E

Como resultado deste opcode, o registo A ficará a conter o que estava na posição contida por HL.

BASIC: READ e DATA

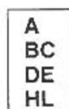
Em linguagem máquina não há equivalentes directos para READ e DATA, porém é bastante simples simulá-los. Já se explicou que os opcodes exigem que isso se faça. Na secção que trata dos dois jogos mostrar-se-á como é possível armazenar e ler dados.

PUSH e POP

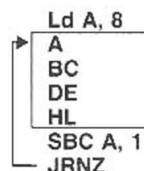
PUSH e POP são dois opcodes da linguagem máquina que na realidade não têm equivalentes em BASIC. Os fluxogramas que se

seguem ajudarão a explicar o que na realidade PUSH e POP fazem.

Suponhamos que temos uma rotina que usa os sete registos que já mencionámos:



Se em BASIC quiséssemos executar várias vezes esta rotina usaríamos A para próximo ciclo. Isto foi já explicado anteriormente e vamos então experimentar:

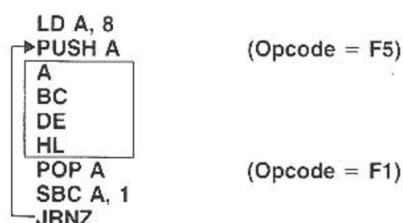


Estamos certos de que poderá ver que o programa acima escrito não funcionaria. É como montar um ciclo FOR/NEXT tal como

FOR A = 1 TO 8

e então usar a variável A na rotina no interior do ciclo. Aconteceriam coisas muito estranhas; o valor de A a controlar o ciclo FOR/NEXT seria destruído se o A fosse utilizado nesta rotina. Na programação BASIC é muito fácil evitar este problema, pois que há muitas variáveis que podem ser usadas.

Em linguagem máquina é um verdadeiro problema e usamos PUSH e POP para o evitar da maneira que a seguir se indica.



Depois de termos carregado o registo A com 8 usamos o opcode PUSH, o que em termos muito simples significa que se coloca o valor de A num local «seguro» que se designa por pilha. Podemos então executar a rotina principal sem receio de destruir o valor original de A. No fim da rotina usamos o opcode POP A, que permite recuperar o valor original de A.

PUSH e POP podem ser usados para quaisquer registos pares mas não para registos individuais. Podemos já ouvir todos a dizer: «Mas acabamos de ver a utilização de PUSH e POP para o registo A, que é um registo individual.»

Admitimos que assim foi, mas fizemo-lo para evitar confusões. O opcode F5 significa PUSH o par de registos AF. F é um registo que não se mencionou previamente. Trata-se de um registo que não pode ser usado como os outros sete registos e não voltará a ser mencionado neste livro.

Imprimir um carácter no «écran»

Se estivermos a programar em BASIC é bastante simples imprimir um carácter. Na programação em linguagem máquina é ligeiramente mais difícil e requer compreender como os caracteres estão armazenados na memória e como funciona o visor. Primeiramente vamos descrever em pormenor como se imprime a letra A no canto superior esquerdo do *écran*. Em seguida mostraremos como se imprime um carácter em qualquer local do mesmo.

Se o leitor estiver familiarizado com a programação BASIC sabe que cada carácter é formado numa grelha 8 x 8.

O diagrama da página seguinte mostra como se constrói a letra A.

Cada uma das oito filas da grelha do carácter pode ser convertida num número decimal (veja-se o «Apêndice» que diz respeito ao sistema binário, se se pretender saber mais sobre este assunto). Os oito números ficam então armazenados na memória. Consequentemente cada carácter precisa de oito posições de memória para armazenar os seus dados. Os dados para a letra A começam na posição 15880. Isto pode ser verificado fazendo entrar e passando o se-

	128	64	32	16	8	4	2	1	Valor decimal
Fila 1									0
Fila 2									60
Fila 3									66
Fila 4									66
Fila 5									126
Fila 6									66
Fila 7									66
Fila 8									0

guinte programa, que imprimirá os valores de cada uma das filas como acima foi indicado.

```

400 LET X = 15880
405 FOR A = 0 to 7
410 PRINT PEEK (X + A)
415 NEXT A
420 STOP

```

Há 768 posições possíveis para imprimir um carácter no *écran* (24 filas x 32 colunas). Sabemos já que cada carácter usa oito posições de memória, do que se segue que o total do *écran* utiliza 6144 posições de memória (8 x 768). A memória para o *écran* tem início na posição 16384 e vai até à posição 22527. Quando pretendemos que apareça algo no *écran* temos de o colocar na sua área de memória. O *hardware* do *Spectrum* encarrega-se então de o enviar para o televisor.

Se se introduzir o exemplo seguinte, começar-se-á a ver como é que um carácter é impresso.

```

POKE 16384, 0 (ENTER)
POKE 16640, 60 (ENTER)
POKE 16896, 66 (ENTER)
POKE 17152, 66 (ENTER)
POKE 17408, 126 (ENTER)
POKE 17664, 66 (ENTER)
POKE 17920, 66 (ENTER)
POKE 18176, 0 (ENTER)

```

(Observe-se o canto superior esquerdo do *écran* cada vez que se carrega ENTER)

Temos agora a letra A impressa no canto superior esquerdo. O que na realidade se fez colocar os valores do exemplo da grelha da letra A na área de memória do visor.

A única coisa que se não terá compreendido é por que razão «pokámos» as posições particulares. Isso será explicado mais tarde.

Antes de se imprimir um carácter no *écran* é preciso saber duas coisas.

- 1) O endereço inicial dos oito dados do carácter que se pretende imprimir. No caso da letra A, o endereço inicial dos oito dados é 15880.
- 2) Também se precisa de saber o endereço inicial das oito posições na memória do visor que devem ser «pokadas» com os dados do carácter. No caso do canto superior esquerdo do *écran*, a primeira posição é 16384.

O fluxograma que se segue mostra como imprimir o carácter A no canto superior esquerdo do *écran*.

Carregar o par de registos HL com 15880 (o endereço inicial dos dados para a letra A)

Carregar o par de registos BC com 16384 (o endereço inicial da posição de imprimir no canto superior esquerdo)

Carregar A,8 (este é o número de dados que formam o carácter)

→ PUSH AF (pôr o valor de A num lugar seguro)

Carregar A com os dados no endereço contido por HL

Carregar o endereço contido em BC com os dados contidos em A

Aumentar o valor de HL até ao ponto do próximo dado

Aumentar B. Dado que B é a parte de valor mais elevado do par de registos BC, esta instrução é equivalente à de somar 256 ao par de registos BC. É preciso adicionar 256 a BC porque esta é onde o próximo carácter tem de ser «pokado». Se nos voltarmos a referir ao exemplo onde «pokámos» 8 números no *écran* para produzir a letra A, verificamos que as posições dos números são incrementadas em lances de 256

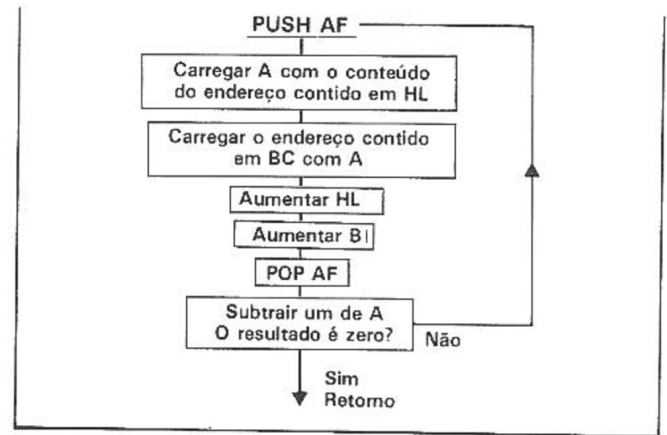
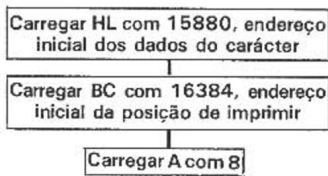
POP AF

Subtrair 1 de A

Se o resultado final não for igual a zero, então saltar para o início do ciclo

Retorno (isto é, o fim da rotina)

FLUXOGRAMA PARA IMPRIMIR A LETRA A



Temos a seguir a listagem em linguagem máquina para imprimir a letra A no canto superior esquerdo. O programa contém 12 linhas com uma instrução em cada linha. A primeira coluna indica, em HEX, o endereço inicial de cada instrução. Assim, o endereço da primeira instrução é 29000. A segunda coluna contém a correspondente linguagem máquina, também em HEX. A primeira instrução é 21083E, que é três bytes de HEX. Na terceira coluna temos os opcodes, sendo o primeiro carregar o registo par HL com 3E08, que é 15880 decimal.

Endereço, em HEX, do primeiro byte de cada instrução	A correspondente linguagem máquina	O opcode
7148	21083E	LD HL,3E08
714B	010040	LD BC,4000
714E	3E08	LD A,08
7150	F5	PUSH AF
7151	7E	LD A,(HL)
7152	02	LD (BC),A
7153	23	INC HL
7154	04	INC B
7155	F1	POP AF

Endereço, em HEX, do primeiro byte de cada instrução	A correspondente linguagem máquina	O opcode
7156	D601	SUB 01
7158	20F6	JR NZ, 7150
715A	C9	RET

Esta é a primeira rotina útil a que podemos dar entrada usando HEX. Carregue-se Hexent e bata-se a tecla E quando o menu aparece. Será então pedido para introduzir o endereço inicial; bater no teclado 29000 e pressionar ENTER. Pode então começar a dar entrada ao programa acima. A primeira linha a dar entrada é 21083E, depois pressione ENTER. A segunda linha é 010040. Quando tiver dado entrada à última linha, C9, deve bater a tecla M e pressionar ENTER, o que o fará regressar ao menu.

Agora bata C e ENTER e entrará no modo de verificação. O computador perguntará qual o endereço inicial do código que quer verificar, que é 29000, e a seguir o endereço final, que é 29018. Os conteúdos desses endereços serão então visualizados no *écran* e terão de ser verificados para se saber se se deu entrada ao programa correctamente.

Pressione M para regressar ao menu e a seguir carregue em X para começar a verificação da contagem do HEX. O Hex pedir-lhe-á uma vez mais que dê entrada aos endereços iniciais e finais do código que quer verificar. Após uma curta pausa será visualizado um número no *écran*, que é o total HEX. Neste caso será 1585. Se o número impresso = 1585, pode ter a certeza absoluta de que deu entrada ao código correctamente e que pode fazê-lo trabalhar tecando RANDOM USR 29000. No entanto, antes de o fazer tem de quebrar o programa Hexent imprimindo Q na fase do menu. Como precaução adicional deve guardar a linguagem antes de a executar, e para o fazer escreva no teclado

SAVE "PRINT" CODE 29000, 19

Se o programa abortar enquanto o está a executar, pode facilmente proceder à sua recarga, primeiro carregando o Hexent e depois escrevendo

LOAD "" CODE

o que lhe permitirá fazer o exame do programa e esperar com confiança que vai encontrar o erro.

A rotina «Imprimir um carácter» que acabámos de desenvolver será usada na maior parte dos programas deste livro. Ocorrerá uma ligeira diferença, consistindo em que as duas primeiras instruções (carregar HL e carregar BC) não serão incluídas. Quando queremos imprimir um carácter, HL deve ser carregado com o endereço inicial dos dados do carácter, e BC deve por sua vez ser carregado com o endereço inicial da posição de impressão desejada; e então será chamada a rotina de impressão.

Esta sub-rotina de impressão pode também ser usada para imprimir determinados gráficos do utilizador. Não existe diferença entre imprimir um carácter comum e um UDG¹. Só o que se tem de fazer é carregar HL com o endereço inicial dos dados do carácter. O endereço inicial da letra A do UDG é

65368 = 48K SPECTRUM
32600 = 16K SPECTRUM

Não foram utilizados os UDG nos programas deste livro, pois não teria utilidade quando se está a aprender e envolveria muita confusão pelas diferenças entre 48K e 16K.

Existe um outro método para imprimir um carácter que implica chamar uma das sub-rotinas na «read only memory» (ROM). Contudo este método não é tão versátil como aquele que até agora descrevemos e não se teria aprendido tanto sobre linguagem máquina se se tivesse usado uma sub-rotina.

Já se mostrou como, uma vez que se saiba o endereço inicial da posição em que se quer imprimir um carácter e depois, adicionando sucessivamente o número 256 ao endereço inicial, calcular onde devem ser postas as sete restantes partes do carácter.

O endereço inicial do primeiro carácter da fila zero é 16384, o segundo começa em 16385, o terceiro em 16386 e assim por diante até ao 32.º carácter da fila zero, que tem início em 16415, o que é muito fácil de compreender. Desloquemo-nos para a fila um, em que o primeiro carácter começa na posição 16416, o segundo na 16417 e por aí adiante, até ao 32.º carácter, o qual se inicia na posi-

¹ User Defined Graphics = gráficos definidos pelo utilizador. (N. do T.)

ção 16447. O endereço inicial do primeiro carácter da fila dois é 16448 e prossegue até ao 32.º carácter exactamente da mesma maneira que na fila precedente. Já notaram a relação entre os endereços iniciais dos primeiros caracteres de cada linha? Há uma diferença de 32 entre cada um deles.

O primeiro carácter na fila zero começa em 16384, o primeiro na fila um começa em 16416 e o primeiro na fila três começa em 16448. A sequência continua até, e incluindo, a fila sete. O salto entre a fila sete e a oito não é usual: tem de se adicionar 1824. Da fila oito até à quinze, inclusive, continua a sequência, com o formato + 32. Uma vez mais o salto entre a linha quinze e a dezasseis exige uma adição de 1824, e então a sequência normal prossegue. A tabela que se segue indica o endereço inicial do primeiro e do último carácter em cada fila, mostrando onde ocorrem os saltos entre as linhas.

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

Com a ajuda desta tabela será possível encontrar o endereço inicial de qualquer posição de impressão no *écran*.

Daremos um exemplo: como encontrar o endereço inicial do décimo carácter na fila 9. Sabemos que o primeiro carácter na fila 9 começa na posição 18464; por conseguinte, o décimo carácter começa em $(18464 + 9) = 18473$.

E, se pretende estar seguro de ter compreendido o que se explicou atrás, pode, como exercício, converter a rotina que se utilizou para imprimir a letra A no canto esquerdo superior numa outra para colocar a mesma letra na terceira posição da fila 10 (o equivalente em BASIC seria: PRINT AT 10, 2;).

Atributos

Para cada posição de carácter existe uma correspondente posição na memória que é usada para registar a cor da tinta e a cor do papel e se o carácter é brilhante/normal ou flamejante/normal. Esta área de memória é chamada o arquivo dos atributos e começa na posição 23528, prosseguindo até 23295. $(32 \times 24 = 768$ posições de caracteres.)

Para mostrar como funciona o arquivo dos atributos, vamos experimentar o exemplo seguinte. Primeiro devemos assegurar-nos de que estamos a trabalhar com tinta preta e papel branco. Em seguida escreve-se o comando directo: PRINT AT 0,0; «X».

Teremos então a letra X impressa a preto no papel branco no canto superior esquerdo.

Agora:

POKE 22528, 7

e o X passará a estar impresso a tinta branca em papel preto.

Escreva:

POKE 22528, 135

e teremos o X flamejante.

A posição 22528 é o atributo que corresponde às primeiras posições de impressão na linha zero. Em primeiro lugar «pokámos» es-

ta posição com o número 7, que significa tinta branca e papel preto. Em seguida «pokámos» com o 135, que significa flamejar.

O arranjo do ficheiro dos atributos é muito directo. A posição 22528 corresponde à primeira posição de impressão na linha zero, 22529 é a segunda posição. A posição que corresponde à última impressão na linha zero é 22559. 22560 diz respeito à primeira posição de impressão na linha um, 22561 é a segunda na mesma linha e por aí adiante. No «Apêndice» indicam-se as posições que correspondem ao primeiro e ao último carácter em cada linha.

No quadro seguinte indicam-se os códigos que devem ser «pokados» no arquivo dos atributos para que se consiga o resultado desejado.

COR DO PAPEL	COR DA TINTA							
	PRETO	AZUL	ENC.	MAGENTA	VERDE	TURQUE	AMARELO	BRAN.
PRETO	0	1	2	3	4	5	6	7
AZUL	8	9	10	11	12	13	14	15
ENC.	16	17	18	19	20	21	22	23
MAGENTA	24	25	26	27	28	29	30	31
VERDE	32	33	34	35	36	37	38	39
TURQUE	40	41	42	43	44	45	46	47
AMARELO	48	49	50	51	52	53	54	55
BRANCO	56	57	58	59	60	61	62	63

Se se pretender papel magenta e tinta encarnada teremos de fazer entrar 26 na correspondente posição, dado que, como verificamos no quadro acima, o número 26 encontra-se na intersecção do papel magenta e da tinta encarnada.

Para tornar brilhante um carácter adiciona-se 64 ao código obtido do quadro acima. E assim, para ter um carácter impresso em tinta encarnada brilhante em papel magenta o código seria 90 (26 + 64).

Para obter um carácter flamejante tem de se somar 128 aos códigos obtidos do quadro. Se quisermos um carácter flamejante impresso a tinta vermelha em papel magenta o código será 154 (26 + 128).

E, para terminar, se se pretender um carácter brilhante, flame-

jante, em tinta encarnada e impresso em papel magenta, teremos o código 218 (26 + 64 + 128).

Qual é o código para papel turquesa e tinta verde?

Qual é o código para papel branco, tinta turquesa e flamejante?

Detectar qual a tecla que está pressionada

Em BASIC, quando queremos detectar uma tecla, usualmente servimo-nos de qualquer coisa como:

```
IF INKEY$ = "5" THEN GOTO
IF INKEY$ = "8" THEN GOTO
```

Se procedermos a um exame veremos que estamos activamente a procurar saber se uma determinada tecla foi pressionada; e do mesmo modo se procede em linguagem máquina.

Com a finalidade de ler qual a tecla que está a ser pressionada pode considerar-se o teclado dividido em 8 blocos, cada um contendo 5 teclas. O diagrama seguinte mostra como está dividido o teclado.

BLOCOS		COLUNAS				
		10h	08h	04h	02h	01h
FE	V	C	X	Z	Shift	
FD	G	F	D	S	A	
FB	T	R	E	W	Q	
F7	5	4	3	2	1	
EF	6	7	8	9	Ø	
DF	Y	U	I	O	P	
BF	H	J	K	L	Enter	
7F	B	N	M	SYM	Space	

Como vemos, a cada um dos blocos foi dado um número HEX. O primeiro bloco tem a etiqueta FE, o segundo FD. Os blocos por sua vez estão divididos em cinco colunas, a cada uma das quais igualmente se atribui um número HEX. A coluna da direita tem o número 01.

O exemplo seguinte mostra como escrever uma rotina para detectar uma tecla específica.

3E (Bloco)	Ld A, x
D3 FF	OUT 255, A
DB FE	IN A, 254
E6 (Coluna)	AND, A

Suponhamos que queremos ver se a tecla K está a ser pressionada. A primeira instrução será

3E BF

Esta instrução carrega o registo A com o número do bloco em que está a tecla que queremos detectar.

A segunda instrução mantém-se a mesma, independentemente da tecla que estamos a verificar.

D3 FF OUT 255, A

A terceira instrução também se mantém a mesma.

DB FE IN A, 254

Isto lê o bloco do teclado que seleccionámos na primeira instrução. No registo A são postos diferentes valores, dependendo de qual das cinco teclas deste bloco está a ser pressionada.

A última instrução será:

E604 AND A, 04

04 mostra que a tecla que queremos detectar está na coluna 04, como está a tecla K. Esta última instrução porá o registo A a zero se a tecla K está a ser pressionada. A rotina final para detectar a tecla K é

**3E BF
D3 FF
DB FE
E6 04**

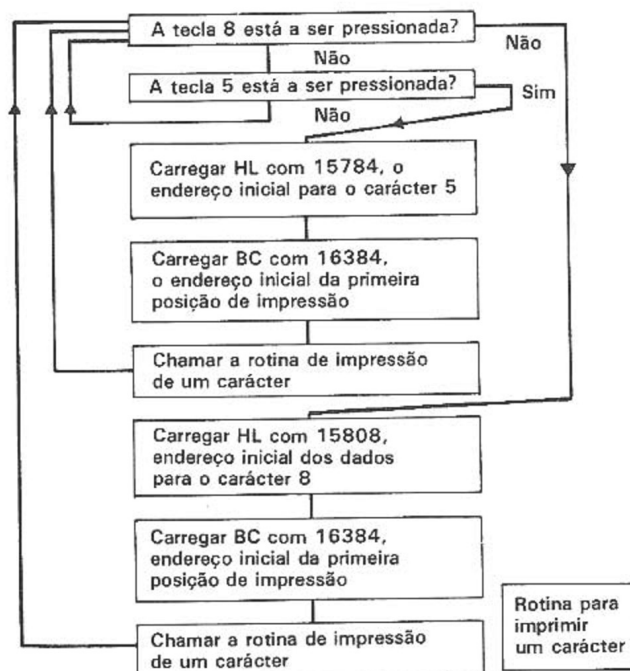
Para detectar qualquer outra tecla basta seleccionar o bloco e a coluna certas.

Por si própria, a rotina acima não tem utilidade e seria normalmente seguida por uma outra instrução tal como: se A é zero (a tecla foi pressionada), saltar então para uma rotina que executa seja o que for que aconteça quando aquela tecla for pressionada.

Vamos agora desenvolver uma rotina que mostrará um meio prático de detectar teclas. Nesta rotina, se a tecla 5 está pressionada, aparecerá um 5 impresso no canto superior esquerdo e teremos um 8 se for a tecla 8 que tiver sido pressionada.

O fluxograma para esta rotina é o seguinte:

FLUXOGRAMA PARA DETECTAR AS TECLAS 5 E 8



O fluxograma não necessita de explicações. Vamos descrever o que acontece se for pressionada a tecla 8. O ciclo principal procura constantemente saber se a tecla 8 ou a 5 está pressionada. Ao detectar-se a tecla 8 salta-se para uma rotina que coloca em HL o valor 15808, que é o endereço inicial dos dados para 8. O registo BC passa a conter o valor 16384, que é o início da posição onde queremos imprimir o carácter. É então chamada a rotina para imprimir um carácter, que é idêntica a uma que já se escreveu e testou anteriormente. A diferença consiste em que HL e BC devem ser colocados antes de se chamar a rotina da pesquisa de teclas.

Listagem em linguagem máquina para detectar as teclas 5 e 8

Endereço inicial 29000
Endereço final 29060
HEX total 7789

7148	3EEF	LD	A,EF
714A	D3FF	OUT	(FF), A
714C	DBFE	IN	A, (FE)
714E	E604	AND	04
7150	CA6071	JP	Z,7160
7153	3EF7	LD	A,F7
7155	D3FF	OUT	(FF), A
7157	DBFE	IN	A, (FE)
7159	E610	AND	10
715B	CA6C71	JP	Z,716C
715E	18E8	JR	7148
7160	21C03D	LD	HL,3DC0
7163	010040	LD	BC,4000
7166	CD7871	CALL	7178
7169	C34871	JP	7148
716C	21A83D	LD	HL,3DA8
716F	010040	LD	BC,4000
7172	CD7871	CALL	7178
7175	C34871	JP	7148
7178	3E08	LD	A,08
717A	F5	PUSH	AF
717B	7E	LD	A, (HL)
717C	02	LD	(BC), A
717D	23	INC	HL

717E	04	INC	B
717F	F1	POP	AF
7180	D601	SUB	01
7182	20F6	JR	NZ,717A
7184	C9	RET	

Não seria usual só querermos imprimir os números 5 ou 8, e normalmente as duas chamadas da rotina de impressão seriam substituídas por rotinas de chamada, que seriam algo como

IF KEY 8 IS PRESSED THEN LET X = X + 1
IF KEY 5 IS PRESSED THEN LET X = X - 1

onde X representa uma variável tal como a posição horizontal dum canhão. Na verdade usaremos este método na próxima secção, quando se desenvolver o jogo do «Invasor Espacial».

Deve notar-se que, desde que se tenha iniciado esta rotina, a única maneira de a parar é desligar da corrente. Pedimos desculpas por isto, mas foi inteiramente intencional para terem de voltar a carregar o Hexent e ter memória disponível pronta para a secção seguinte.

Diferenças fundamentais entre a linguagem máquina e o BASIC

As programações em linguagem máquina e em BASIC são mundos à parte: há duas diferenças da maior importância que se devem ter em mente quando se desenvolvem programas em linguagem máquina.

Quando se escreve um programa em BASIC é usual numerar-se as linhas 10, 15, 20, 25, etc. Isto permite acrescentar linhas entre as existentes, se o quisermos fazer, sem causar grande perturbação. Em linguagem máquina as linhas estão efectivamente numeradas 1, 2, 3, 4, 5, etc. Consequentemente é impossível inserir linhas extra sem ter de mudar a posição de todas as linhas seguintes. Quando, na próxima secção, desenvolvermos o programa «Invasor Espacial» veremos como ultrapassar este problema intercalando «linhas em branco» entre as rotinas individuais dentro do programa global.

Assim, se precisarmos de acrescentar mais linhas a uma rotina, basta mudar a posição das linhas no seio da própria rotina e ter assim espaço disponível de modo a não afectar as rotinas subsequentes.

A segunda maior diferença é a portabilidade das rotinas em linguagem máquina. Se estudarmos a rotina «Imprimir um carácter» veremos que é possível colocá-la em qualquer parte da memória disponível e ela funcionará. Podemos experimentar fazê-la entrar em diferentes posições usando o Hexent; podemos, por exemplo, fazê-la entrar nas posições 29500 ou 29611, mas temos de lembrar-nos de estabelecer o acesso à rotina por meio da instrução RAN-DOM X, em que X representa o endereço inicial no qual se começou. A rotina impressão pode considerar-se portátil, dado que pode ser colocada em qualquer ponto da memória sem quaisquer mudanças.

A maior parte das rotinas em linguagem máquina não são portáteis. Se atentarmos, por exemplo, na rotina que desenvolvemos para detectar duas teclas, veremos que ela contém instruções tais como C3 48 71, o que significa saltar para a posição 29000. É óbvio que isto não é o que queremos que aconteça se tivermos deslocado a rotina para a posição 35000. Devíamos ter intercalado uma rotina inteiramente nova na posição 29000.

Com o fim de dar nova colocação a uma rotina não portátil teríamos de alterar os saltos absolutos e chamadas relevantes. Na próxima secção aprenderemos como desenvolver um mapa de memória, que ajudará a evitar os dois problemas acima mencionados.

Rotinas em linguagem máquina (programa «Invasor Espacial»)

O programa «Invasor Espacial»

Este capítulo descreve em pormenor como se escreve um simples programa «Invasor Espacial». O facto de o jogo ser simples não significa que as rotinas que vão ser descritas não possam ser usadas para criar um jogo muito mais interessante. Devemos lembrar-nos que mesmo o programa mais complexo pode ser dividido numa série de blocos e que mesmo esses podem ainda ser divididos num conjunto de rotinas. No fim deste capítulo o leitor terá ganho suficientes conhecimentos que lhe permitirão começar a escrever os seus próprios programas.

Este capítulo iniciará-se pela descrição do fluxograma para o programa. Se já tiver escrito programas em BASIC não terá dificuldade em compreender um fluxograma.

A seguir discutiremos como formar um mapa de memória. Na programação em BASIC não houve necessidade de o fazer, mas é muito importante quando se inicia o desenvolvimento dos programas em linguagem máquina.

Vamos prosseguir com a descrição pormenorizada de cada um dos blocos do fluxograma. Por exemplo: um dos blocos tem a etiqueta «mover a bala para cima no écran». O modo como isto se consegue será descrito com a maior exactidão e, quando for necessário, indicar-se-á um subfluxograma adicional da presente rotina.

Neste programa as rotinas foram escritas de tal modo que podem ser testadas cada uma de per si ou em conjugação com uma rotina que tenha sido previamente testada. Isto permitirá testar o programa passo a passo e ver exactamente o que cada rotina faz.

Na altura em que se escreveu este livro a intenção do autor era que o leitor se sentasse ao computador e não o largasse até que tivesse acabado a secção do livro. Contudo, agora compreende-se que em dada altura o leitor terá de devolver a televisão à família, de modo que ela possa receber a sua ração diária de telenovela. Por conseguinte, em qualquer altura, é fácil guardar aquilo que já deu entrada e recarregá-lo posteriormente. Primeiro faz-se parar o programa que está a ser executado, carregando na tecla Q do menu. Em seguida escreve-se:

SAVE "INVADER" CODE 29000, 3010

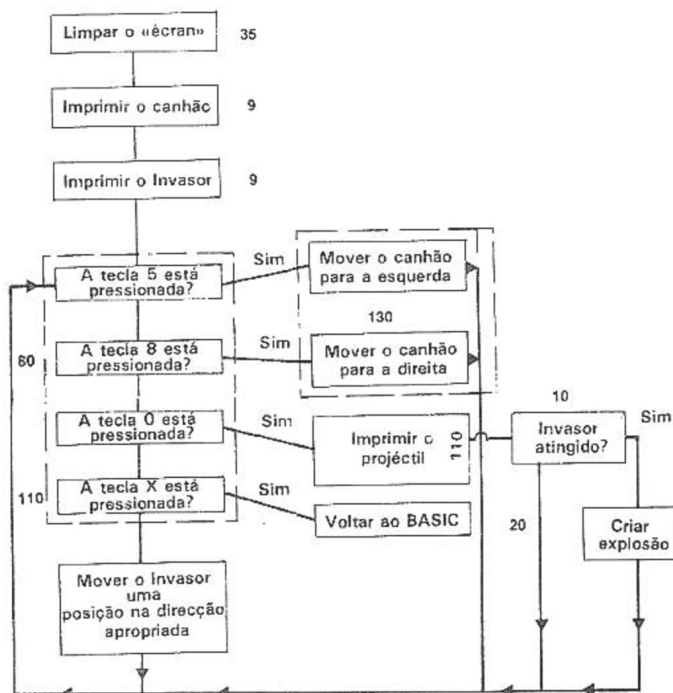
Isto permite guardar o que já tiver dado entrada em linguagem máquina. Para voltar a carregar a linguagem máquina (depois da telenovela), primeiro carrega-se o programa Hexent e pára-se. Então imprime-se

LOAD "" CODE

Quando a linguagem máquina estiver carregada escreve-se «executar» (RUN) e pode continuar-se onde se tinha interrompido.

FLUXOGRAMA

Uma das primeiras coisas a fazer quando se constrói um programa é desenhar um fluxograma que mostre com clareza como o dito programa vai funcionar. O diagrama que temos a seguir é o fluxograma do programa «Invasor Espacial».



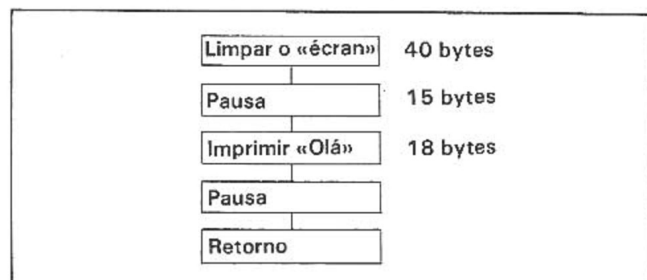
Concordam por certo que este fluxograma é relativamente fácil de compreender. O leitor deve estudá-lo e compreender bem como vai funcionar o programa e verá então como cada rotina se interliga com as outras.

Mapa de memória

Já explicámos que em linguagem máquina não temos números de linhas. Consequentemente, se precisarmos de inserir uma nova instrução teremos de mover todas as instruções que estão a seguir à

posição onde a vamos colocar. Ora, isto causa imensos problemas com saltos absolutos e chamadas. Suponhamos que temos uma rotina de pausa na posição 30000 do programa. Para ter acesso a esta rotina, seja de que parte for do programa, temos a instrução de chamada 30000. E se, por qualquer razão, precisarmos de acrescentar uma instrução de uns meros três bytes no fim da rotina que termina na posição 29999 precisamos, como é evidente, usar as posições 30000/30001/30002. E assim teremos de mover a rotina de pausa para a posição 30003. Feito isto teremos de seguir o programa e alterá-lo desde a chamada 30000 até à 30003. Escrever um programa desta maneira leva muito tempo.

Ora, acontece que estes problemas podem ser ultrapassados utilizando o fluxograma principal para criar um mapa de memória. Para isso precisamos de saber aproximadamente quantos bytes há em cada rotina. Então atribuem-se a cada rotina posições específicas de memória e deixa-se também entre cada rotina um espaço de memória disponível. Estes espaços em branco permitirão aumentar cada rotina sem afectar as seguintes. O fluxograma abaixo pode ser usado para mostrar como se faz um mapa de memória.



Como vemos acima, cada rotina está marcada com o número aproximado de bytes que utiliza.

Podemos colocar a rotina «limpar o écran» na posição 30000, a da pausa em 30050 e a de impressão na 30080. Isto deixará alguns bytes entre cada rotina, de modo que cada uma pode ser aumentada, se for necessário, sem ser preciso «rebaralhar» todo o programa.

Se se atentar no fluxograma do programa do «Invasor Espacial» verifica-se que ao longo de cada rotina se indicou o número

aproximado de bytes que se terá de usar. Utilizando esta informação elaborou-se o mapa de memória que se segue. Na coluna da esquerda temos o nome de cada rotina e na da direita o endereço inicial respectivo.

MAPA DE MEMÓRIA DO «INVASOR ESPACIAL»

Limpar o «écran»	29000
Colocar e imprimir o canhão e o Invasor	29050
Detectar as teclas	29100
Mover o canhão	29150
Pausa	29230
Mover o projectil	29257
Alvo atingido	29370
Explosão	29385
Mover o Invasor	29470
Imprimir um carácter	29610
Verificar a impressão	29700
Variáveis	32000
Controle	29630

Neste mapa de memória considerou-se uma área da memória para um bloco designado por «controle», que não está no fluxograma. A rotina de controle pode ser considerada como um gestor do programa. Uma outra área considerada no mapa de memória, mas não no fluxograma, é «teste de impressão», que será explicada mais adiante.

Para se obter o máximo de benefício desta secção deve dar-se entrada e testar cada rotina de per si, como foi indicado. Recomenda-se ainda que, depois de fazer entrar cada rotina, se guarde a linguagem máquina utilizando

SAVE «INVADER» CODE 29000, 3010

Caso algo desagradável suceda, pode ficar-se com a certeza de se poder voltar a carregar a linguagem, evitando-se os inconvenientes de se ter de a voltar a escrever.

Imprimir um carácter

Esta rotina é idêntica à rotina de impressão que já foi desenvolvida. Quando queremos imprimir um carácter, deve colocar-se HL com o endereço inicial do carácter que se quer imprimir e BC com o inicial da posição de impressão. A seguir chama-se a rotina por meio de 29610 (CD AA 73).

Linguagem máquina para a rotina «Imprimir um carácter»

Endereço inicial 29610
Endereço final 29622
HEX total 1417

73AA	3E08	LD	A,08
73AC	F5	PUSH	AF
73AD	7E	LD	A, (HL)
73AE	02	LD	(BC), A
73AF	23	INC	HL
73B0	04	INC	B
73B1	F1	POP	AF
73B2	D601	SUB	01
73B4	20F6	JR	NZ,73AC
73B6	C9	RET	

Não é possível testar esta rotina só por si, porque HL e BC devem ter os seus valores colocados antes de se ter acesso à rotina. Tem de se dar entrada a uma rotina de teste de pequena dimensão, cujo código é este

7404	21083E	LD	HL,3E08
7407	010040	LD	BC,4000
740A	CDAA73	CALL	73AA
740D	C9	RET	

Esta rotina de teste colocará HL ao ponto dos dados para a letra A, e BC para o início da primeira posição de impressão; nesta altura a rotina de impressão será chamada.

ENDEREÇO INICIAL = 29700
ENDEREÇO FINAL = 29709
HEX TOTAL = 859

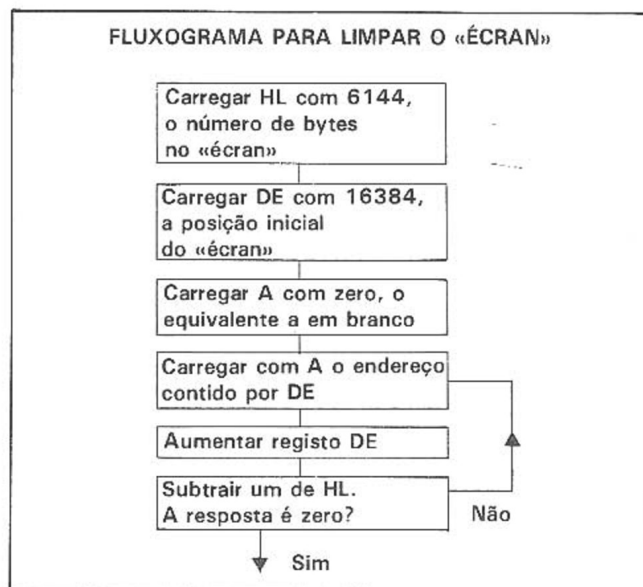
Para testar a rotina de impressão deve bater-se no teclado:

RANDOMIZE USR 29700

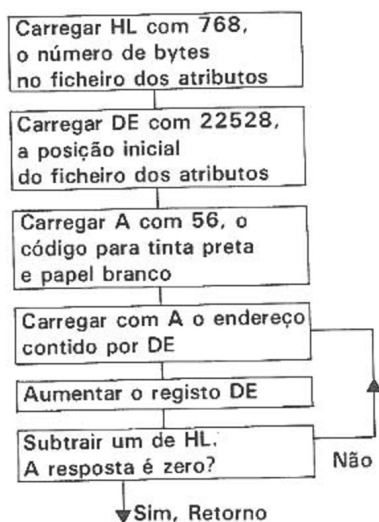
A letra A deve aparecer impressa no canto superior esquerdo.

Limpar o «écran»

Esta rotina encherá totalmente a área da memória do écran com zeros, por conseguinte nada aparecerá no écran. Isto é o equivalente a imprimir em branco em cada posição de carácter. Esta rotina colocará também os atributos das cores em tinta preta e papel branco. O fluxograma seguinte mostra como o programa funciona.



FLUXOGRAMA PARA LIMPAR O FICHEIRO DOS ATRIBUTOS



Linguagem máquina da rotina para limpar o «écran»

Endereço inicial	29000
Endereço final	29037
HEX total	2185

7148	210018	LD	HL,1800
714B	110040	LD	DE,4000
714E	3E00	LD	A,00
7150	12	LD	(DE), A
7151	13	INC	DE
7152	010100	LD	BC,0001
7155	ED42	SBC	HL,BC
7157	20F7	JR	NZ,7150
7159	210003	LD	HL,0300
715C	110058	LD	DE,5800

715F	3E38	LD	A,38
7161	12	LD	(DE), A
7162	13	INC	DE
7163	010100	LD	BC,0001
7166	ED42	SBC	HL,BC
7168	20F7	JR	NZ,7161
716A	328D5C	LD	(5C8D), A
716D	C9	RET	

A rotina para limpar o *écran* pode ser testada pela entrada do programa BASIC que se segue:

```

400 PAPER 2
405 CLS
410 INK 7
415 PAPER 5
420 PRINT AT 10,10; "HELLO"
425 PAUSE 100
430 RANDOMIZE USR 29000
435 STOP
  
```

Por meio deste programa teremos primeiro o *écran* em encarnado, a seguir aparecerá a palavra «hello» em tinta branca em papel turquesa. Depois de uma curta pausa teremos acesso à rotina «limpar o *écran*», o que dará como consequência o *écran* ficar em branco. Mais uma curta pausa, o controle regressará ao BASIC e pode-se testar que se tem então palavras em tinta preta impressas em papel branco.

Posicionar as variáveis e imprimir o canhão

Dada a falta de variáveis e o pequeno número de registos da linguagem máquina, é preciso encontrar outra maneira de armazenar os valores de coisas que mudam de posição, tais como a posição horizontal do canhão no jogo «Invasor Espacial». Como se viu no mapa de memória, atribuíram-se para as variáveis a posição 32000 e seguintes, o que dará uma indicação sobre a forma como as variáveis estão armazenadas. As posições 32000 e 32001 conterão o endereço inicial em que o canhão se situa no *écran*. Para começar, este será impresso no meio da linha do fundo. Assim, a posição 32000 armazenará o número 20720, que é o endereço inicial da décima sétima posição de impressão na fila mais baixa. Utilizando es-

te método podem armazenar-se tantas variáveis quanto se quiser, desde que se tenha espaço disponível de memória. O exemplo seguinte mostra como deste modo se faz uso das variáveis armazenadas.

Suponhamos que a posição 32000 contém o canhão e que o vamos deslocar uma posição de impressão para a direita.

```
LD HL (32000)
INC HL
LD (32000), HL
```

e disto resulta que se carrega o registo HL com a posição do canhão.

Incrementa-se HL (teríamos de diminuir, se se quisesse levar o canhão para a esquerda) e voltamos a pôr o novo valor de HL em 32000. E que não haja preocupações por se não compreender isto, pois será descrito todo o processo quando chegarmos a deslocar o canhão para a esquerda e para a direita.

Em programação BASIC estávamos habituados à colocação de variáveis no início, tal como LET A = 20720. Pois bem, neste caso é também igualmente importante no arranque dum programa em linguagem máquina colocar as variáveis. E é também boa altura para imprimir o canhão na sua posição inicial.

A rotina seguinte colocará o canhão variável e fará a sua impressão usando a respectiva rotina, a que já foi dada entrada.

Endereço inicial	29050
Endereço final	29063
HEX total	1639

717A	21683D	LD	HL,3D68
717D	01F050	LD	BC,50F0
7180	ED43007D	LD	(7D00),BC
7184	CDAA73	CALL	73AA
7187	C9	RET	

Como funciona a rotina

HL é colocado no início dos dados do carácter do canhão. (Está-se a utilizar o sinal — para representar o canhão). BC é colocado por sua vez no endereço inicial da posição de impressão em

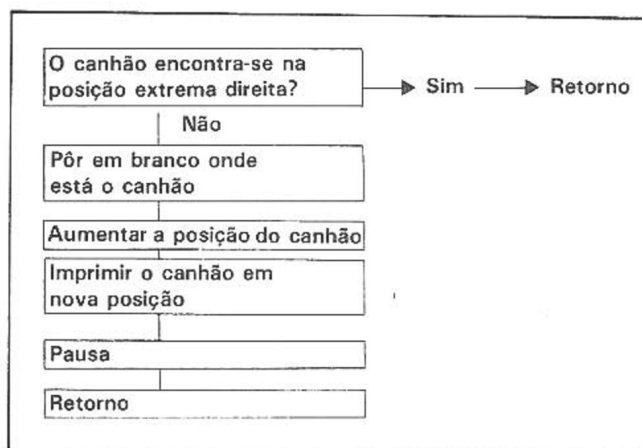
que o canhão estará no começo do jogo. O endereço variável do canhão (32000) é então carregado com BC. A seguir chama-se a rotina de impressão, que imprimirá o canhão no centro da fila de baixo. A instrução final desta rotina é RETURN.

Para efectuar o teste desta rotina faz-se entrar o seguinte programa em BASIC:

```
400 RANDOMIZE USR 29050
405 GOTO 405
```

Deslocar o canhão

Para a rotina «Mover o canhão» há duas partes principais, muito parecidas uma com a outra e que se designam por «mover o canhão para a direita» e «mover o canhão para a esquerda». A parte adequada é executada conforme é pressionada a tecla 5 ou a 8. Que fique porém bem claro que a rotina para detectar as teclas 5 ou 8 não se encontra nesta parte e sim na zona para onde o programa salta se uma das teclas for pressionada. O fluxograma seguinte é a parte do programa que controla o que sucede se a tecla 8 for batida.



Como funciona a rotina

Se a tecla 8 foi detectada chama-se a sub-rotina. A tecla 8 é usada para mover o canhão para a direita e portanto a primeira coisa que se deve fazer é ver se o canhão está na posição extrema da direita do *écran*. Se isso acontecer não se pode, como é óbvio, deslocar o canhão ainda mais para a direita; assim, volta-se à rotina para detectar uma tecla. A maneira que se usa para detectar se o canhão está na extrema direita é carregar HL com a corrente posição do canhão da localização 32000, carregar o registo BC com 20735 (este valor é o início da extrema direita posição de impressão na fila de baixo). Subtraímos BC de HL. Se o resultado for zero é porque o canhão deve estar no limite da direita e regressamos à rotina de procura da tecla. Se o resultado não for zero, então pomos o canhão em branco carregando HL com o endereço de início para o carácter espaço, carregamos BC com a corrente posição do canhão e então chamamos a rotina de impressão. Para deslocar o canhão para a direita, carregamos BC com a corrente posição do canhão, sendo BC assim aumentado. O seu novo valor é colocado na posição 32000, isto é, na nova posição do canhão. Coloca-se HL no início dos dados do carácter e é chamada a rotina de impressão. Antes de finalmente se voltar para a rotina de procura da tecla, faz-se uma pausa que controla a velocidade máxima a que o canhão se moverá. Se a não incluíssemos e se a tecla 5 ou a 8 continuasse pressionada, o canhão mover-se-ia tão depressa que dificilmente poderia ser visto. A parte desta rotina que controla o movimento do canhão para a esquerda é muito semelhante à rotina do movimento para a direita, com excepção de que a primeira coisa a fazer é verificar se ele se encontra na posição esquerda limite e, antes de o imprimir, diminuir a posição em que se encontra.

Temos a seguir uma listagem da rotina para mover o canhão que deve ter entrada na posição 29150.

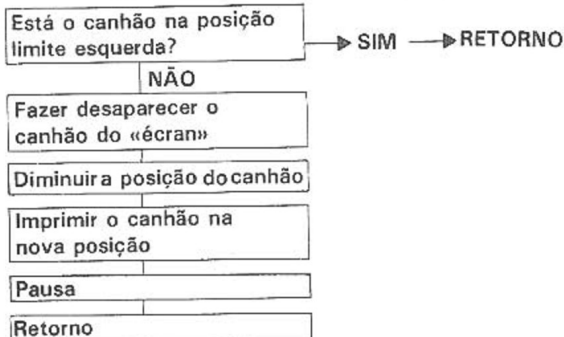
Rotina em linguagem máquina para mover o canhão

Endereço inicial 29150
Endereço final 29225
HEX total 8277

71DE 2A007D LD HL,(7D00)
71E1 01FF50 LD BC,50FF

71E4 ED42 SBC HL,BC
71E6 C8 RET Z
71E7 21003D LD HL,3D00
71EA ED4B007D LD BC,(7D00)
71EE CDAA73 CALL 73AA
71F1 ED4B007D LD BC,(7D00)
71F5 03 INC BC
71F6 ED43007D LD (7D00),BC
71FA 21683D LD HL,3D68
71FD CDAA73 CALL 73AA
7200 CD2E72 CALL 722E
7203 C9 RET
7204 2A007D LD HL,(7D00)
7207 01E050 LD BC,50E0
720A ED42 SBC HL,BC
720C C8 RET Z
720D 21003D LD HL,3D00
7210 ED4B007D LD BC,(7D00)
7214 CDAA73 CALL 73AA
7217 ED4B007D LD BC,(7D00)
721B 0B DEC BC
721C ED43007D LD (7D00),BC
7220 21683D LD HL,3D68
7223 CDAA73 CALL 73AA
7226 CD2E72 CALL 722E
7229 C9 RET

FLUXOGRAMA DA ROTINA QUE É EXECUTADA SE SE PRESSIONAR A TECLA 5



Como se vê no fluxograma, a rotina para mover o canhão requer uma rotina de pausa. Esta rotina de pausa está na posição 29230 e a ela deve ser dada entrada antes que a rotina para mover o canhão seja testada.

Linguagem máquina para a rotina de pausa para mover o canhão

Endereço inicial 29230
Endereço final 29240
HEX total 843

722E	010100	LD	BC,0001
7231	210014	LD	HL,1400
7234	ED42	SBC	HL,BC
7236	20FC	JR	NZ,7234
7238	C9	RET	

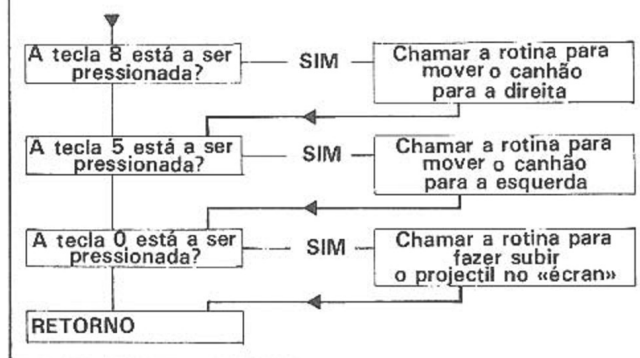
Pode proceder-se ao teste da rotina para mover o canhão dando entrada ao seguinte programa em BASIC. Inicia-se o programa BASIC batendo no teclado GOTO 400. Se se pressionar a tecla 1 será chamada a rotina para mover o canhão para a esquerda. Se for pressionada a tecla 2 será chamada a rotina para mover o canhão para a direita.

```
400 IF INKEY$ = "2" THEN RANDOMIZE USR 29150
405 IF INKEY$ = "1" THEN RANDOMIZE USR 29188
410 GOTO 400
```

Detectar as teclas

Esta rotina permitirá saber se a tecla pressionada foi a 5, a 8 ou a 0. No primeiro caso será chamada a rotina para mover o canhão para a esquerda. Se foi a 8, a correspondente para a direita. Sendo a 0, a rotina que será chamada é a que fará o disparo. O fluxograma seguinte não precisa de explicações.

FLUXOGRAMA PARA DETECTAR AS TECLAS 5, 8 E 0



Linguagem máquina da rotina para detectar as teclas

Endereço inicial 29100
Endereço final 29133
HEX total 5892

71AC	3EEF	LD	A,EF
71AE	D3FF	OUT	(FF),A
71B0	DBFE	IN	A,(FE)
71B2	E604	AND	04
71B4	CCDE71	CALL	Z,71DE
71B7	3EF7	LD	A,F7
71B9	D3FF	OUT	(FF),A
71BB	DBFE	IN	A,(FE)
71BD	E610	AND	10
71BF	CC0472	CALL	Z,7204
71C2	3EEF	LD	A,EF
71C4	D3FF	OUT	(FF),A
71C6	DBFE	IN	A,(FE)
71C8	E601	AND	01
71CA	CC4972	CALL	Z,7249
71CD	C9	RET	

Quando se está a proceder ao teste da rotina para detectar as teclas não se pode bater a tecla 0 se ainda se não tiver dado entrada à rotina «fazer um disparo» na posição 29257. No entanto, se se quiser ter a certeza de que o *Spectrum* não aborta se inadvertidamente se pressionar 0, então escreva-se como comando directo:

POKE 29257, 201

201 é o código para retorno e, assim se se pressionar o 0, o programa saltará para a posição 29257 — como devia fazer —, mas fá-lo-á directamente.

Para testar a rotina para detectar as teclas faz-se entrar e executar o seguinte programa em BASIC:

```
400 RANDOMIZE USR 29050
405 RANDOMIZE USR 29100
410 GOTO 405
```

Rotina para fazer um disparo

Esta rotina será chamada se se pressionar a tecla 0. A rotina fará que o projectil suba ao topo do canhão até ao topo do *écran*. Movimentar um objecto da esquerda para a direita através do *écran*, como vimos na rotina para mover o canhão, é bastante simples. Tudo o que se tem a fazer é somar ou subtrair um desde o início da posição de impressão. Mover um objecto para cima ou para baixo é mais difícil, devido à configuração da memória do *écran*.

O exemplo que vamos dar em seguida mostrará como fazer subir o projectil na vertical. Suponhamos que o canhão se encontra no limite inferior esquerdo da fila de impressão do fundo do *écran*. A primeira posição em que o projectil deve ser impresso é mesmo acima do canhão, na fila 22. Sabemos que o endereço inicial onde o canhão está impresso é 20704 (veja-se no «Anexo» a configuração da memória do *écran*). Para se ter o endereço inicial de onde o projectil deve em primeiro lugar estar impresso subtraímos 32 do endereço inicial do canhão — o que dará 20762, que é o endereço inicial da primeira posição do carácter na linha 22. Para mover o projectil para a linha 21, temos primeiro de o apagar e depois subtrair 32 do início da sua posição de impressão, para dar 20640, que é a posição inicial de impressão do primeiro carácter na linha 21. Este método para fazer subir o carácter no *écran* pode continuar até à linha 16,

depois temos de efectuar o salto não sequencial da linha 16 para a 15. Continuamos a subtrair 32 até chegar a linha 8, onde temos de fazer outro salto não sequencial para a linha 7. São estes dois saltos não sequenciais que tornam mais difícil fazer subir o carácter do que deslocá-lo lateralmente. Mover o carácter de uma linha para a linha acima, dentro dum bloco de oito linhas, faz-se sempre da mesma maneira, independentemente de qual dos três blocos se trate. Portanto podemos escrever uma sub-rotina para tratar do simples processo de mover o carácter para a linha de cima.

A rotina para fazer subir o projectil funciona da seguinte maneira: antes de se pressionar a tecla que provoca o disparo não existe projectil no *écran*. Assim, a primeira coisa que a rotina deve fazer é imprimir a bala logo acima do canhão, o que se consegue subtraindo 32 do endereço inicial da posição de impressão do canhão. Há então um retardamento para diminuir a velocidade da bala, fazendo-se depois desaparecer.

O fluxograma seguinte mostra a sub-rotina para subtrair 32 da posição corrente de impressão do canhão, imprimir a bala e depois, após um curto retardamento, fazê-la desaparecer do *écran*.

FLUXOGRAMA PARA SUBIR O PROJÉCTIL UMA LINHA

Subtrair 32 da posição de impressão corrente contida em BC
Imprimir a bala
Retardamento
Fazer desaparecer o projectil do «écran»
Retorno

Para deslocar a bala da linha 32 para a linha 16, a rotina acima deve ser chamada sete vezes, o que se faz estabelecendo um ciclo FOR/NEXT, utilizando o registo A.

Depois de termos executado sete vezes o ciclo, o registo BC — que contém a posição corrente de impressão do projectil — precisará de ser ajustado, com o fim de fazer o salto não sequencial da linha 16 para a 15. Este ajustamento é feito subtraindo 1792 de BC. Para deslocar a bala da linha 15 para a 8, a sub-rotina tem de ser

chamada oito vezes, o que se faz estabelecendo um ciclo utilizando o registo A. Da linha 8 para a 7 tem de se proceder ao ajustamento do registo BC subtraindo 1792. Para mover o projectil da linha 7 para a 0 chama-se outra vez a sub-rotina oito vezes.

O fluxograma que se segue ajudará a compreender como funciona a rotina «Mover o projectil».

FLUXOGRAMA PARA MOVER VERTICALMENTE A BALADA POSIÇÃO CORRENTE DO CANHÃO PARA O TOPO DO «ÉCRAN»

Carregar BC com o endereço do canhão
Carregar A com 7. (Ciclo para mover o projectil da linha 22 para a 16)
Chamar «Ciclo de controle e rotina de salto de linha não sequencial»
Carregar A com 8. (Ciclo para mover o projectil da linha 15 para a 8)
Chamar «Ciclo de controle e rotina de salto de linha não sequencial»
Carregar A com 8. (Ciclo para mover o projectil da linha 7 para a 0)
Chamar «Ciclo de controle e rotina de salto de linha não sequencial»
RETORNO

«Ciclo de controle e rotina de salto de linha não sequencial»

PUSH A
Chamar «Rotina para fazer subir o projectil uma linha»
POP A
Subtrair um de A. Saltar para o início desta rotina se o resultado não é zero
Subtrair 1792 de BC
RETORNO

«Rotina para fazer o projectil subir uma linha»

Subtrair 32 da posição de impressão corrente contida por BC
Imprimir o projectil
Retardamento
Fazer desaparecer o projectil do «écran»
RETORNO

Linguagem máquina para fazer subir o projectil no «écran»

Endereço inicial 29257
Endereço final 29337
HEX total 9206

7249	ED4B007D	LD	BC,(7D00)
724D	3E07	LD	A,07
724F	CD5F72	CALL	725F
7252	3E08	LD	A,08
7254	CD5F72	CALL	725F
7257	3E08	LD	A,08
7259	CD5F72	CALL	725F
725C	C9	RET	
725D	00	NOP	
725E	00	NOP	
725F	F5	PUSH	AF
7260	CD7272	CALL	7272
7263	F1	POP	AF
7264	DE01	SBC	A,01
7266	20F7	JR	NZ,725F
7268	60	LD	H,B
7269	69	LD	L,C
726A	010007	LD	BC,0700
726D	ED42	SBC	HL,BC
726F	44	LD	B,H
7270	4D	LD	C,L
7271	C9	RET	
7272	C5	PUSH	BC
7273	E1	POP	HL
7274	012000	LD	BC,0020

7277	ED42	SBC	HL,BC
7279	44	LD	B,H
727A	4D	LD	C,L
727B	C5	PUSH	BC
727C	21E03F	LD	HL,3FE0
727F	CDAA73	CALL	73AA
7282	CD8F72	CALL	728F
7285	C1	POP	BC
7286	C5	PUSH	BC
7287	21003D	LD	HL,3D00
728A	CDAA73	CALL	73AA
728D	C1	POP	BC
728E	C9	RET	
728F	010100	LD	BC,0001
7292	21FF14	LD	HL,14FF
7295	ED42	SBC	HL,BC
7297	20FC	JR	NZ,7295
7299	C9	RET	

Para testar esta rotina faz-se entrar o seguinte programa em BASIC:

```
400 RANDOMIZE USR 29050
405 RANDOMIZE USR 29100
410 GOTO 405
```

Para mover o alvo (o Invasor)

Esta rotina fará um inofensivo Invasor Espacial deslocar-se de um lado para o outro ao longo do topo do «écran». O Invasor começará a deslocar-se para a direita a partir do canto superior esquerdo. Quando atingir a posição extrema à direita mudará de direcção e começa a deslocar-se para a esquerda. Tal como se fez para o caso do canhão, é preciso colocar as variáveis do Invasor antes da rotina para o fazer movimentar-se.

O Invasor tem duas variáveis. A primeira é a sua posição no écran, a qual está contida nas posições 30003 e 30004. A segunda diz respeito ao facto de ele estar a deslocar-se da esquerda para a direita ou da direita para a esquerda. Esta variável está contida na

posição 30002. Se o Invasor se está a mover da esquerda para a direita, a posição 30002 contém um 1. Se ele está a movimentar-se da direita para a esquerda, tem um zero. Esta posição actua como uma «agulha» e cada vez que chamamos a rotina para fazer mover o Invasor ele diz-nos se a deslocação é para a esquerda ou para a direita. O fluxograma mostra claramente como funciona.

A linguagem máquina para estabelecer as variáveis e inicialmente fazer a sua impressão começará na posição 29063, isto é, directamente a seguir às variáveis do canhão e da sua impressão. De modo que, na versão final do jogo, quando se chama as variáveis do canhão, o mesmo acontece para as do Invasor.

Linguagem máquina para mover o Invasor Espacial

Endereço inicial	29470
Endereço final	29560
HEX total	8082

731E	3A027D	LD	A,(7D02)
7321	DE01	SBC	A,01
7323	202A	JR	NZ,734F
7325	2A037D	LD	HL,(7D03)
7328	011F40	LD	BC, 401F
732B	ED42	SBC	HL,BC
732D	2006	JR	NZ,7335
732F	3E00	LD	A,00
7331	32027D	LD	(7D02),A
7334	C9	RET	
7335	ED4B037D	LD	BC,(7D03)
7339	21003D	LD	HL,3D00
733C	CDAA73	CALL	73AA
733F	ED4B037D	LD	BC,(7D03)
7343	03	INC	BC
7344	ED43037D	LD	(7D03),BC
7348	21503D	LD	HL,3D50
734B	CDAA73	CALL	73AA
734E	C9	RET	
734F	2A037D	LD	HL,(7D03)
7352	010040	LD	BC,4000
7355	ED42	SBC	HL,BC
7357	2006	JR	NZ,735F
7359	3E01	LD	A,01

735B	32027D	LD	(7D02),A
735E	C9	RET	
735F	ED4B037D	LD	BC, (7D03)
7363	21003D	LD	HL,3D00
7366	CDAA73	CALL	73AA
7369	ED4B037D	LD	BC,(7D03)
736D	0B	DEC	BC
736E	ED43037D	LD	(7D03),BC
7372	21503D	LD	HL,3D50
7375	CDAA73	CALL	73AA
7378	C9	RET	

Antes que a rotina para mover o Invasor possa ser testada é necessário fazer entrar a rotina que se segue, que primeiro imprime o Invasor e coloque a «agulha» em 1, o que significa que o Invasor se está a mover da esquerda para a direita.

Endereço inicial	29063
Endereço final	29081
HEX total	1602

7187	21503D	LD	HL,3D50
718A	010040	LD	BC,4000
718D	ED43037D	LD	(7D03),BC
7191	CDAA73	CALL	A,01
7194	3E01	LD	A,01
7196	32027D	LD	(7D02),A
7199	C9	RET	

A rotina do Invasor Espacial pode ser testada utilizando o programa BASIC que se segue. Escreva no teclado GOTO 400 e verá o Invasor a mover-se continuamente através do topo do écran.

```

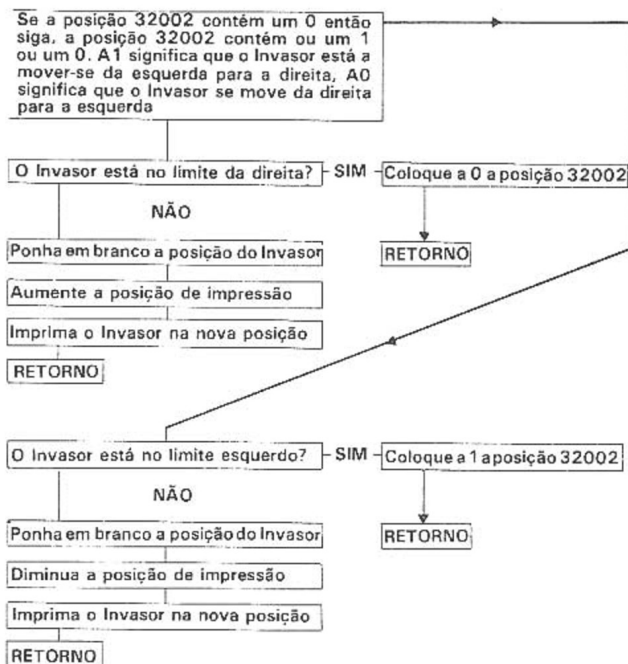
400 RANDOMIZE USR 29063
405 RANDOMIZE USR 29470
410 GOTO 405

```

O fluxograma incluído a seguir mostra como funciona a rotina para movimentar o Invasor. Esta rotina é chamada da rotina de controle que estará no final do jogo. Quando a ela se tiver tido acesso, decidirá em que direcção o Invasor deve ser movido, fará

com que ele se desloque uma posição nessa direcção e voltará à rotina de controle. Se o Invasor estiver num dos limites, a rotina colocará a «agulha» de direcção na direcção oposta, de modo que quando se tem em seguida acesso à rotina o Invasor inverterá a sua direcção.

FLUXOGRAMA PARA MOVER O INVASOR



Como produzir uma explosão

Se o projectil que disparou atingir o Invasor ocorrerá obviamente uma espectacular explosão.

Antes que ela ocorra é necessário detectar que o projectil coli-

diu com o Invasor. A curta rotina para o fazer será descrita após termos escrito e testado a rotina de explosão.

Quando examinarem o fluxograma da explosão perdoamos-lhes por terem pensado que a explosão não parece ser muito espectacular. Tudo o que ela faz é encher o *écran* com preto, depois azul, vermelho, magenta, verde, turquesa e amarelo, o que se repete ciclicamente nove vezes. Quando isto é feito com a veloz rotina em linguagem máquina o resultado é um verdadeiro espectáculo.

FLUXOGRAMA DA EXPLOÇÃO

Carregar HL com 9 (número de vezes que o ciclo principal é executado).

PUSH HL

Colocar amarelo à cor de papel
Chamar «Rotina para limpar atributos»
Colocar azul à cor de papel
Chamar «Rotina para limpar atributos»
Colocar encarnado à cor de papel
Chamar «Rotina para limpar atributos»
Colocar magenta à cor de papel
Chamar «Rotina para limpar atributos»
Colocar verde à cor de papel
Chamar «Rotina para limpar atributos»
Colocar turquesa à cor de papel
Chamar «Rotina para limpar atributos»
Colocar branco à cor de papel
Chamar «Rotina para limpar atributos»

POP HL

Subtrair um de HL. Se o resultado não é zero, saltar então para a segunda instrução desta rotina

RETORNO

«Rotina para limpar atributos»

Esta rotina é exactamente a mesma que a descrita no início desta secção para limpar o «écran»

Linguagem máquina para a rotina da explosão

Endereço inicial 29385
Endereço final 29458
HEX total 7466

72C9	210900	LD	HL,0009
72CC	E5	PUSH	HL
72CD	3E30	LD	A,30
72CF	CDF972	CALL	72F9
72D2	3E08	LD	A,08
72D4	CDF972	CALL	72F9
72D7	3E10	LD	A,10
72D9	CDF972	CALL	72F9
72DC	3E18	LD	A,18
72DE	CDF972	CALL	72F9
72E1	3E20	LD	A,20
72E3	CDF972	CALL	72F9
72E6	3E28	LD	A,28
72E8	CDF972	CALL	72F9
72EB	3E38	LD	A,38
72ED	CDF972	CALL	72F9
72F0	E1	POP	HL
72F1	010100	LD	BC,0001
72F4	ED42	SBC	HL,BC
72F6	20D4	JR	NZ,72CC
72F8	C9	RET	
72F9	210003	LD	HL,0300
72FC	110058	LD	DE,5800
72FF	12	LD	(DE),A
7300	13	INC	DE
7301	010100	LD	BC,0001
7304	ED42	SBC	HL,BC
7306	20F7	JR	NZ,72FF
7308	210100	LD	HL,0001
730B	010100	LD	BC,0001
730E	ED42	SBC	HL,BC
7310	20FC	JR	NZ,730E
7312	C9	RET	

A rotina da explosão pode ser testada fazendo entrar como comando directo

RANDOMIZE USR 29385 (ENTER)

Detectar quando o Invasor é atingido

Com o fim de detectar se o projectil atingiu o Invasor, tudo o que é necessário fazer é comparar as posições do canhão e do Invasor. Se eles se situam na mesma vertical, isto quer dizer que obviamente o projectil atingiu o alvo. O projectil terminou a sua viagem para o topo do *écran* e directamente será dado acesso à rotina «Detectar alvo atingido». No caso de ter sido detectado que o alvo foi atingido será então dado acesso a sub-rotina da explosão.

Não incluímos no texto um fluxograma desta rotina, dado que é muito curto e fácil de compreender. O registo HL é primeiro carregado com o endereço do canhão. Subtrai-se 4320 de HL. BC é carregado com o endereço do Invasor e é subtraído de HL. Se o resultado é zero, quer dizer que o projectil atingiu o alvo. Para confirmar que isto é assim, suponhamos que o canhão está situado no canto inferior esquerdo, e que assim o seu endereço é 20704, e que o Invasor está no canto superior esquerdo, no endereço 16384. Ora, isto significa que o Invasor e o canhão estão na mesma vertical e que o projectil vai atingir aquele. Carrega-se HL com 20704 e dele subtrai-se 4320, ficando com o valor 16384. BC é carregado com o endereço do Invasor, que é 16384. HL - BC = zero, por conseguinte a rotina salta para a explosão.

Linguagem máquina para a rotina «Detectar alvo atingido»

Endereço inicial 29370
Endereço final 29384
HEX total 1646

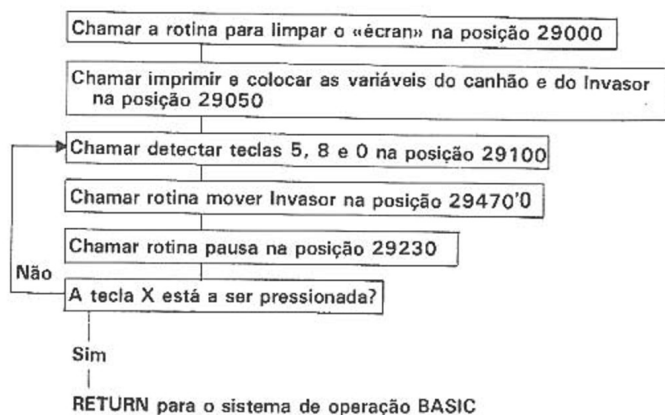
72BA	2A007D	LD	HL,(7D00)
72BD	01E010	LD	BC,10E0
72C0	ED42	SBC	HL,BC
72C2	ED4B037D	LD	BC,(7D03)
72C6	ED42	SBC	HL,BC
72C8	C0	RET	NZ

Faz-se entrar os três bytes seguintes na posição 29276. Isto é uma instrução que faz o programa saltar para o início da rotina para detectar o alvo atingido depois de o projectil ter alcançado o topo do *écran*: C3 BA 72.

A rotina de controle

Tudo o que resta para ser feito agora é dar entrada à rotina de controle. Esta começa na posição 29630 e todo o jogo será posto em funcionamento pela instrução RANDOM USR 29630. A primeira instrução é chamar 29000, que é a rotina para limpar o *écran*. Seguidamente a rotina para estabelecer a variável e imprimir o canhão é chamada para a posição 29050. A terceira instrução é chamar 29100, que é a rotina para detectar as teclas. A próxima rotina a ser chamada do controle é a rotina para mover o Invasor. Como a linguagem máquina é tão rápida, é introduzida outra pausa chamando a rotina de pausa na posição 29230. A parte final da rotina de controle é usada para detectar se a tecla X está a ser pressionada. Se estiver, volta-se então ao sistema de operação BASIC. Se a tecla X não estiver a ser pressionada, então o programa salta para trás, para a terceira instrução da rotina de controle. Esta é apenas um ciclo contínuo que constantemente chama as duas rotinas principais — a saber: a rotina para detectar as teclas e a outra que faz mover o Invasor.

FLUXOGRAMA DA ROTINA DE CONTROLE



Posição inicial 29630
 Posição final 29657
 HEX total 4026

73BE	CD4871	CALL	7148
73C1	CD7A71	CALL	717A
73C4	CDAC71	CALL	71AC
73C7	CD1E73	CALL	731E
73CA	CD2E72	CALL	722E
73CD	3EFE	LD	A,FE
73CF	D3FF	OUT	(FF),A
73D1	DBFE	IN	A,(FE)
73D3	E604	AND	04
73D5	2802	JR	Z,73D9
73D7	18EB	JR	73C4
73D9	C9	RET	

Todo o jogo pode agora ser testado batendo no teclado RANDOM USR 29630. Pode ver-se o Invasor a mover-se continuamente através do *écran*. Utilizam-se as teclas 5 e 8 para mover o canhão para a esquerda e para a direita. Usa-se a tecla 0 para efectuar um disparo contra o Invasor. Quando se estiver cansado de jogar este jogo, carrega-se na tecla X para voltar ao sistema de operação BASIC.

Como todos estarão de acordo, o jogo «Invasor Espacial» agora acabado é bastante simples. O fim desta secção não era proporcionar um entretenimento mas sim mostrar como escrever e testar pequenas rotinas e em seguida interligá-las para formar um jogo completo. Se tiverem compreendido como funciona cada rotina, nada os impedirá de escrever os vossos pequenos jogos. Uma das maiores fraquezas do jogo que acabámos de escrever é que o Invasor pára quando o projectil é disparado. Para os que tenham estudado os fluxogramas será fácil compreender por que é que isto acontece. E é sem dúvida devido ao facto de que nada é feito até que o projectil atinja o topo do *écran*. Aham que há algum modo de modificar o programa de forma que o Invasor continue a deslocar-se mesmo depois de o projectil ter sido disparado?

Notaram que neste programa se usaram algumas rotinas de pausa. Num programa mais profissional, isso seria considerado um

desperdício de tempo. Normalmente estas rotinas seriam substituídas por algo mais útil, mas produzindo o mesmo efeito. Por exemplo, podia usar-se o tempo para dar animação a alguns dos gráficos normalmente estáticos no *écran* ou para mover o fundo do jogo. Ou ainda se poderia usar o tempo disponível para criar efeitos sonoros ou tocar uma melodia, se bem que isto seja ligeiramente mais complicado.

Esperamos que ao terminar esta secção tenham aprendido os princípios fundamentais da programação em linguagem máquina. Podíamos ter feito o «Invasor Espacial» muitíssimo mais complexo, arranjando um conjunto de Invasores que fossem morrendo à medida que os alvejassemos. Podíamos ter dado mais cores e sons. Porém, deliberadamente, quisemos fazer um programa simples e sem complicações, esperando que pudesse ser compreendido com maior facilidade.

Quando estiverem a desenvolver as vossas próprias rotinas em linguagem máquina atribuam uma área específica para cada uma. Experimentem escrevê-las de modo que possam ser testadas individualmente. Não devem esquecer que é também importante demarcar uma área da memória para as variáveis. Quando estiverem a escrever as vossas próprias rotinas não terão o benefício dum verificador do HEX, de modo que devem sempre guardar a linguagem antes de a experimentar.

Como uma ajuda para um mais profundo conhecimento da linguagem máquina, experimentem neste programa fazer qualquer coisa como, por exemplo, alterar os valores dos ciclos de pausa ou modificar o programa de controle de modo que a rotina para detectar as teclas seja chamada duas vezes em vez de uma.

Dois jogos e algumas rotinas úteis

Este capítulo contém dois jogos e algumas rotinas úteis para o leitor escrever. Não são descritos com aquela abundância de pormenores do jogo do «Invasor Espacial», mas, com a ajuda dos fluxogramas, poderá aprender alguns truques.

Os jogos são a preto e branco por duas razões principais. Em primeiro lugar para diminuir a quantidade de escrita a realizar; em segundo lugar para tornar mais fáceis de compreender os fluxogramas e as descrições. Nos vossos próprios programas vão ver que é relativamente simples juntar as cores. Lembrem-se que quando se tem de mover um objecto, como o canhão do programa do «Invasor Espacial», é também preciso mover os seus atributos das cores.

Dados

Descrição do programa

Este programa simula o lançar de um dado. O dado continuará sempre a rolar até que a tecla S seja pressionada e ficará imóvel até que ela seja libertada.

Como funciona

O ficheiro dos atributos das cores é usado para «imprimir» no *écran*. Se este estiver completamente branco e se introduzir a instrução POKE 22862,0 ver-se-á aparecer um ponto negro no meio do *écran*. E isto acontece porque a posição 22862 é o atributo que corresponde ao meio do *écran* e 0 é o código para preto. Na secção que trata do BASIC e seus equivalentes em linguagem máquina há uma descrição completa do ficheiro dos atributos das cores. As posições de memória do ficheiro dos atributos que se utilizarão para «imprimir» as sete pintas possíveis de um dado são indicadas no diagrama seguinte.

22796 X		22800 X
22860 X	22862 X	22864 X
22924 X		22928 X

Para visualizar o lado três do dado, tudo o que é preciso fazer é utilizar a instrução POKE das três posições adequadas com 0. Se se experimentar o que a seguir se indica ver-se-á o lado 3:

```
POKE 22796, 0 (ENTER)
POKE 22862, 0 (ENTER)
POKE 22928, 0 (ENTER)
```

Neste programa também usamos a instrução POKE para as quatro restantes posições com 63 (63 é o código para o branco). Isto assegura que o que quer que fosse que estivesse aí será apagado.

É necessário armazenar os valores relativos a cada um das seis faces do dado, isto é, quais os pontos que estão ligados (código 0) e quais os que estão desligados (código 63). Cada face do dado tem sete pontos que podem, em certas combinações, estar ligados ou desligados. Por conseguinte precisamos de 42 bytes (6×7) da memória para armazenar os valores relativos ao total dos seis lados. Neste programa escolheu-se armazenar os valores nas posições 30000 a 30041, inclusive. Os valores para o lado um estão armazenados nas posições 30000 a 30006 do modo seguinte:

Posição	Valores (HEX)
30000	3F
30001	3F
30002	3F
30003	00
30004	3F
30005	3F
30006	3F

Os valores correspondentes à face dois estão contidos nas posições desde 30007 a 30013, e por aí adiante no que respeita às outras faces. Para imprimir no *écran* os valores de uma face temos de primeiro colocar um apontador para o início dos valores da face que queremos imprimir. No caso da face um, colocaremos o apontador para 30000. Depois colocamos um outro apontador para 30050. As posições de 30050 a 30063 contêm os sete endereços do ficheiro dos atributos para os quais transferimos os valores.

Posição	Conteúdos
30050/1	22796
30052/3	22800
30054/5	22860
30056/7	22862
30058/9	22864
30060/1	22924
30062/3	22928

FLUXOGRAMA PARA IMPRIMIR UMA FACE DO DADO

Colocar DE no início dos valores do lado do dado que quer imprimir. Se se tratar do lado um, DE deve ser colocado em 30000

Colocar HL em 30050

Carregar A.(DE)

Carregar (HL),A

INC HL Executar este ciclo sete vezes
INC HL para transferir os sete valores
INC DE para o ficheiro dos atributos

A rotina acima é na realidade uma sub-rotina do programa principal. Se se quiser imprimir um dos lados do dado basta colocar DE no início dos valores do lado que se quer imprimir, e depois chamar esta sub-rotina.

Vejamos como funciona o programa. Em primeiro lugar coloca-se DE no início dos valores da face um, e então chama-se a sub-rotina de impressão. No fim desta foi acrescentada uma pequena rotina para detectar se a tecla S está pressionada. Se assim for, o programa espera até que ela seja libertada antes de deixar a sub-rotina de impressão. O programa principal coloca a seguir DE no início dos valores para a face dois e chama a sub-rotina de impressão.

Esta sequência prossegue até à face seis e nessa altura o programa salta para trás, para o início do ciclo. Deste modo, os seis lados do dado estão a ser constantemente mostrados um após outro. Porém acontece que a impressão é tão rápida que nunca se pode realmente ver qual o lado que está a ser impresso. Só quando se faz parar o programa é que então se pode ver qual é o lado.

Listagem em linguagem máquina para o programa dos dados

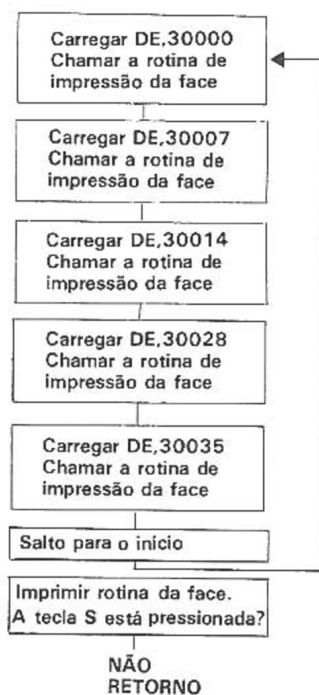
Endereço inicial 30000
Endereço final 30138
HEX total 11210
Começar por RANDOMIZE USR 30064

7530	3F
7531	3F
7532	3F
7533	00
7534	3F
7535	3F
7536	3F
7537	00
7538	3F
7539	3F
753A	3F
753B	3F
753C	3F
753D	00
753E	00

753F 3F
 7540 3F
 7541 00
 7542 3F
 7543 3F
 7544 00
 7545 00
 7546 00
 7547 3F
 7548 3F
 7549 3F
 754A 00
 754B 00
 754C 00
 754D 00
 754E 3F
 754F 00
 7550 3F
 7551 00
 7552 00
 7553 00
 7554 00
 7555 00
 7556 3F
 7557 00
 7558 00
 7559 00
 755A 00
 755B 00
 755C 00
 755D 00
 755E 00
 755F 00
 7560 00
 7561 00
 7562 0C
 7563 59
 7564 1059
 7566 4C
 7567 59
 7568 4E
 7569 59
 756A 50

756B	59		
756C	8C		
756D	59		
756E	90		
756F	59		
7570	113075	LD	DE,7530
7573	CD9675	CALL	7596
7576	113775	LD	DE,7537
7579	CD9675	CALL	7596
757C	113E75	LD	DE,753E
757F	CD9675	CALL	7596
7582	114575	LD	DE,7545
7585	CD9675	CALL	7596
7588	114C75	LD	DE,754C
758B	CD9675	CALL	7596
758E	115375	LD	DE,7553
7591	CD9675	CALL	7596
7594	18DA	JR	7570
7596	3E62	LD	A,62
7598	329F75	LD	(759F),A
759B	3E07	LD	A,07
759D	F5	PUSH	AF
759E	2A0075	LD	HL,(7500)
75A1	1A	LD	A,(DE)
75A2	77	LD	(HL),A
75A3	13	INC	DE
75A4	3A9F75	LD	A,(759F)
75A7	3C	INC	A
75A8	3C	INC	A
75A9	329F75	LD	(759F),A
75AC	F1	POP	AF
75AD	3D	DEC	A
75AE	20ED	JR	NZ,759D
75B0	3EFD	LD	A,FD
75B2	D3FF	OUT	(FF),A
75B4	DBFE	IN	A,(FE)
75B6	E602	AND	02
75B8	28F6	JR	Z,7580
75BA	C9	RET	

FLUXOGRAMA COMPLETO DO PROGRAMA DOS DADOS



Som

O *Spectrum* não desfruta de fama pelas suas capacidades de produzir sons. Se o leitor possuir na sua colecção alguns dos jogos mais sofisticados, sabe que se podem produzir sons e melodias bastante razoáveis. Não temos a pretensão de ensinar a escrever música,

mas indicaremos como se podem produzir várias notas. Para criar som usaremos uma sub-rotina no ROM, e que é quase tão simples como usar uma das suas próprias sub-rotinas. Chama-se simplesmente a rotina usando a instrução CD XX. O endereço da sub-rotina do som é 949d, por conseguinte, a instrução completa será CD B5 03. Antes de se chamar a rotina os pares de registos HL e DE devem ser estabelecidos com valores que se relacionam com a frequência e a duração do som.

Suponhamos que queremos produzir um som cuja frequência é 350 Hz e que dura dois segundos. A fórmula para achar o valor de DE é:

$$\text{FREQUÊNCIA} \times \text{DURAÇÃO EM SEGUNDOS}$$

Portanto, o valor de DE deve ser

$$350 \times 2 = 700$$

E a fórmula para encontrar o valor do registo HL é

$$(437\,500/\text{FREQUÊNCIA}) - 30,125$$

Por conseguinte, HL deve ter o valor

$$(437\,500/350) = 1250 - 30,125 = 1219,875$$

O valor a dar a HL deve ser 1219 (arredondando sempre por defeito).

A rotina que se segue dará para DE o valor 700 e para HL o de 1219, e então deve chamar-se a sub-rotina do som. Dá-se entrada a esta rotina na posição 29000, imprimindo depois RANDOM USR 29000. E ouvir-se-á o som durante dois segundos.

```

11 BC 02
21 C3 04
CD B5 03
C9
  
```

Endereço inicial	29000
Endereço final	29009
HEX total	1029

Que valores devem ter HL e DE para produzir uma frequência de 1200 Hz com a duração de três segundos?

Se se quiser criar sons do tipo «Invasor Espacial», tais como *la-sers*, *bipebipe* e *pumpum*, o melhor método é o da experimentação. A rotina seguinte consiste num ciclo com uma chamada da rotina do som. O valor de DE permanece constante, mas o valor de HL é aumentado do valor de BC cada vez que o ciclo se completa. O registo A é inicialmente colocado no valor 10, o que determina o número de vezes que se completa o ciclo. Tem de se dar entrada à rotina seguinte na posição 29000 e começar-se por utilizar RANDOM USR 29000.

Endereço inicial 29000
Endereço final 29023
HEX total 2345

7148	215703	LD	HL,0357
714B	3E0A	LD	A,0A
714D	F5	PUSH	AF
714E	113200	LD	DE,0032
7151	E5	PUSH	HL
7152	CDB503	CALL	03B5
7155	E1	POP	HL
7156	013200	LD	BC,0032
7159	09	ADD	HL,BC
715A	F1	POP	AF
715B	DE01	SBC	A,01
715D	20EE	JR	NZ,714D
715F	C9	RET	

Uma vez que se tenha dado entrada e se tenha testado a rotina pode começar-se a fazer experiências com ela. Pode mudar-se o número de vezes em que o ciclo é completado por meio de POKE 29001,X, em que X pode ter um valor entre 1 e 255. Poder-se-ia alterar os valores iniciais de HL e DE. Efectuando experiências com os valores é possível produzir uma frequência tão alta que não pode ser ouvida pelo homem mas que um cão pode ouvir! Se se der a DE um valor demasiado alto, pode acontecer que se esperem horas para que o som termine. Podia também ser alterado o valor de BC.

As combinações destes valores diferentes produzirão muitos sons diferentes. Como regra geral, quando se estiverem a fazer estas experiências, o valor de cada coisa deve ser alterado somente em pequena quantidade e assim se saberá o que está a causar um determinado efeito.

O quadro que se segue indica quais os valores de HL e DE que devem ser atribuídos para produzir cada uma das doze notas de música contidas na primeira coluna durante um segundo.

	HL	DE
Mi	1642	261
Mi sustenido	1548	277
Fá	1459	293
Fá sustenido	1406	311
Sol	1297	329
Lá	1222	349
Lá sustenido	1152	369
Si	1085	392
Si sustenido	1023	415
Dó	964	440
Dó sustenido	908	466
Ré	855	493

As rotinas de som previamente indicadas usam ciclos quer para aumentar quer para diminuir os vários registos. Na realidade este método é adequado para produzir algo de musical. A rotina seguinte mostra como produzir uma sequência de notas musicais. Claro que primeiro é preciso saber quais as notas que se vão tocar. Neste exemplo só teremos a sequência das doze notas que constam da tabela anterior, de Mi a Ré.

Os dois valores de HL e DE que são precisos para produzir uma das notas estão contidos em quatro consecutivas posições de memória. Os da primeira nota estão em 30000 até 30003, os da segunda de 30004 a 30007 e por aí adiante até à décima segunda nota, cujos valores estão contidos nas posições 30044 e 30047. O fluxograma seguinte mostra como funciona o programa.

Ld BC 30000 Colocar o apontador para o início do quadro de dados	
Ld A 12 Colocar o número de notas a tocar	
PUSH A	
Ld A (BC) Ld L,A INC BC Ld A(BC) Ld H,A	Obter o valor de HL do quadro de dados
Ld A (BC) Ld E,A INC BC Ld A (BC) Ld D,A	Obter o valor de DE do quadro de dados
PUSH BC	
Chamar a rotina do som	
POP BC INC BC	Colocar o apontador para o início dos dados da nota seguinte
POP A Subtrair 1 de A Se o resultado não é zero saltar para PUSH A	As 12 notas foram tocadas
RETURN. Fim da rotina	

Como funciona

O registo BC é usado como um apontador para os dados e começa com o valor de 30000. Os dados nessa posição são carregados no registo A usando a instrução LD A,(BC). O conteúdo do registo A é então transferido para o registo L utilizando a instrução LD L,A.

BC é então aumentado para a posição 30001. Nesta altura a informação é transferida para o registo H. O registo BC é mais uma vez aumentado e a informação transferida para o registo E. Os dois registos HL e DE estão agora carregados com os dados para produzir a primeira nota, o que se faz chamando a rotina do som.

Se se quiser produzir as doze notas junta-se um ciclo no início, utilizando o registo A, que é carregado com 12. No final da rotina principal o registo A é diminuído e efectua-se uma verificação para ver se ele atingiu o zero. Se isso não acontecer, salta-se para o início da rotina e a nota seguinte é tocada. A entrada dos dados e do programa faz-se da maneira corrente usando Hexent. Para iniciar o programa bate-se no teclado RANDOMIZE USR 30043. (30048 é onde se inicia o presente programa. As posições 30000 a 30047 contêm os dados correspondentes às notas.)

Linguagem máquina da rotina para tocar notas armazenadas

Endereço inicial	30000
Endereço final	30076
HEX total	5533

7530	6A
7531	06
7532	05
7533	01
7534	0C
7535	06
7536	15
7537	01
7538	B3
7539	05
753A	25
753B	01

753C	7E		
753D	05		
753E	37		
753F	01		
7540	11		
7541	05		
7542	49		
7543	01		
7544	C6		
7545	04		
7546	5D		
7547	01		
7548	80		
7549	04		
754A	71		
754B	01		
754C	3D		
754D	04		
754E	88		
754F	01		
7550	FF		
7551	03		
7552	9F		
7553	01		
7554	C4		
7555	03		
7556	B8		
7557	01		
7558	8C		
7559	03		
755A	D2		
755B	01		
755C	57		
755D	03		
755E	ED		
755F	01		
7560	013075	LD	BC,7530
7563	3E0C	LD	A,0C
7565	F5	PUSH	AF
7566	0A	LD	A,(BC)
7567	6F	LD	L,A
7568	03	INC	BC

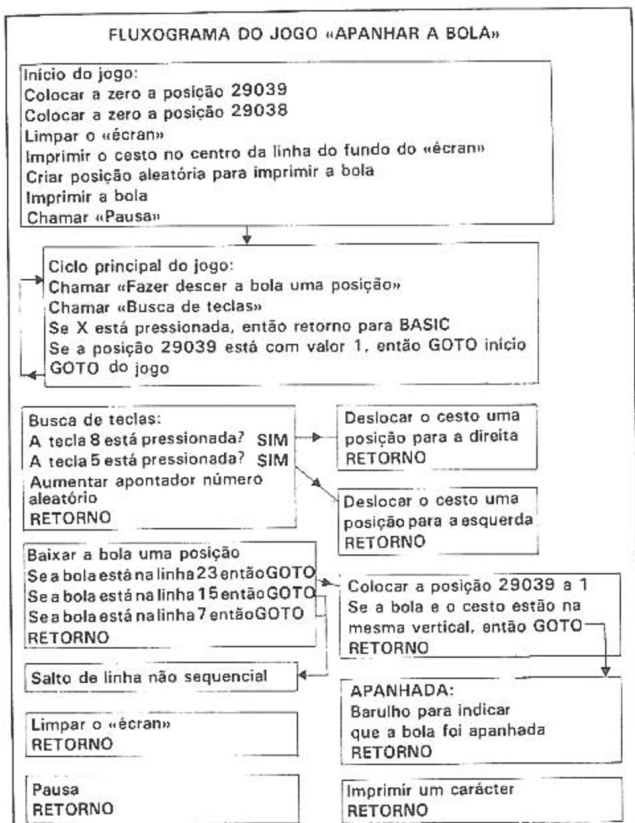
7569	0A	LD	A,(BC)
756A	67	LD	H,A
756B	03	INC	BC
756C	0A	LD	A,(BC)
756D	5F	LD	E,A
756E	03	INC	BC
756F	0A	LD	A,(BC)
7570	57	LD	D,A
7571	C5	PUSH	BC
7572	CDB503	CALL	03B5
7575	C1	POP	BC
7576	03	INC	BC
7577	F1	POP	AF
7578	DE01	SBC	A,01
757A	20E9	JR	NZ,7565
757C	C9	RET	

Se se usar o método indicado para produzir som, seremos capazes de armazenar as notas para tocar qualquer melodia. (Se o leitor não for dotado para a música, como sucede com o autor deste livro, trabalhar com notas para armazenar pode constituir um problema!)

Apanhar a bola

Neste jogo aparecerá uma bola em posição aleatória ao longo da fila de cima do *écran* e que em seguida vai cair verticalmente. O que se tem de fazer é apanhar a bola, utilizando um cesto que se pode deslocar de um lado para outro ao longo da fila do fundo do *écran*. Deve estudar-se o fluxograma que apresentamos a seguir, para compreender o programa na sua globalidade.

Para dar início à rotina do jogo é preciso estabelecer as variáveis e proceder à impressão da bola e do cesto. No início este está sempre na mesma posição, a saber, no centro da linha do fundo do *écran*. E a bola está, por seu lado, sempre em posição aleatória no topo do mesmo. Esta posição aleatória consegue-se do modo seguinte: as posições desde 29000 até 29031 contêm a sequência de



números de 1 a 31. As posições 29032/3 contêm um número entre 29000 e 29031, que se chama o apontador para a tabela de números aleatórios. As três instruções seguintes são então usadas para calcular a posição aleatória inicial da bola.

Load DE, (29032)

O registo DE conterá agora um número entre 29000 e 29031

Load A, (DE)

O registo A conterá agora um valor entre 1 e 32

Load HL, 16383

O valor de A é agora acrescentado a HL, procedendo repetidamente ao aumento de HL e à diminuição de A até que o valor deste seja zero. HL conterá agora um valor entre 16384 e 16415, que será a posição inicial da bola. Este valor é colocado na posição 29036/7 e será utilizado quando se fizer a bola descer ao longo do *écran*. O apontador na posição 29032/3 está continuamente a mudar de local, dado que no início de cada jogo a bola é impressa numa posição aleatória. Mais adiante vai ser descrito o modo como o apontador está constantemente a mudar de local. Tendo-se inicialmente imprimido a bola, segue-se uma curta pausa, o suficiente para dar ao jogador uma melhor oportunidade de a apanhar.

Inicia-se então o principal ciclo de controle do jogo, o qual consiste no seguinte:

- 1) Chama-se a rotina que faz baixar a bola uma posição.
- 2) Proceder-se a uma verificação para ver se a bola está no fundo do *écran*. Se tal acontecer volta-se ao princípio.
- 3) Chama-se a rotina de busca de teclas para ver se as teclas 5 e 8 estão a ser pressionadas.
- 4) Volta-se para o item 1 do início deste ciclo.

A rotina que se usa para fazer descer a bola uma posição é muito semelhante à usada no programa «Invasor Espacial» para fazer subir o projectil. A corrente posição da bola é primeiramente obtida da posição 29036/7. Então esta é usada para fazer desaparecer a bola do *écran*. Seguidamente calcula-se e armazena-se em 29036/7 a nova posição de impressão e imprime-se a bola. Quando se faz descer a bola no *écran* é necessário tomar em consideração os saltos de linha não sequenciais, o que se consegue mantendo-se a verificação da variável na posição 29038. No início de cada jogo esta variável é posta a zero e representa o número da linha onde a bola está. Cada vez que se faz descer a bola uma posição, a variável na posi-

ção 29038 é aumentada; quando atinge 8 ou 16 sabemos que temos de fazer um salto de linha não sequencial. Quando a posição 29038 contém um valor 22, a bola atingiu a linha justamente acima daquela em que se desloca o cesto. Uma vez que a bola tenha atingido a linha do fundo do *écran* podemos chamar a rotina que faz a verificação para saber se o cesto a apanhou. Esta rotina usa o endereço do cesto que está contido na posição 29034/5 e o endereço da bola que está na posição 29036/7. Por conseguinte, do endereço do cesto subtrai-se 32 e o resultado é comparado com o endereço da bola; se forem iguais é chamada a sub-rotina do ruído, para indicar que a bola foi apanhada. A posição 29039 é então colocada no valor 1. Isto é usado pelo ciclo de controle principal para indicar que pode ser começado um novo jogo.

A rotina de busca das teclas verifica se as teclas 5 e 8 estão a ser pressionadas e moverá o cesto uma posição na direcção apropriada. A posição 29034/5 é actualizada para indicar a nova posição do cesto. No fim da rotina de busca das teclas o apontador para a tabela de números aleatórios é aumentado e reposto em 29000 se atingir 29032. Dado que a rotina para busca das teclas é chamada muitas vezes por segundo, obviamente o apontador do número aleatório estará continuamente a mudar, dando por conseguinte a garantia de que a bola começará em posição aleatória no início de cada jogo.

O ciclo de controle principal verifica constantemente a posição 29039. Se ela contém um 1, então deve ser começado um novo jogo. No início de cada jogo, a posição 29039 é levada a 0.

Listagem em linguagem máquina para apanhar a bola

Endereço inicial	29000
Endereço final	29401
HEX total	41790

Inicia-se o programa batendo no teclado RANDOM USR 29040. Lembremo-nos de que as posições 29000/29039 contêm variáveis e dados para o programa funcionar.

7148	01
7149	02
714A	03

714B	04		
714C	05		
714D	06		
714E	07		
714F	08		
7150	09		
7151	0A		
7152	0B		
7153	0C		
7154	0D		
7155	0E		
7156	0F		
7157	10		
7158	11		
7159	12		
715A	13		
715B	14		
715C	15		
715D	16		
715E	17		
715F	18		
7160	19		
7161	1A		
7162	1B		
7163	1C		
7164	1D		
7165	1E		
7166	1F		
7167	20		
7168	58		
7169	71		
716A	F0		
716B	50		
716C	00		
716D	00		
716E	00		
716F	00		
7170	3E00	LD	A,00
7172	326F71	LD	(716F),A
7175	326E71	LD	(716E),A
7178	CDC772	CALL	7207
717B	21A83E	LD	HL,3EA8
717E	01F050	LD	BC,50F0
7181	ED436A71	LD	(716A),BC
7185	CD3F72	CALL	723F
7188	ED5B6871	LD	DE,(7168)
718C	1A	LD	A,(DE)
718D	01FF3F	LD	BC,3FFF
7190	03	INC	BC
7191	3D	DEC	A
7192	C29071	JP	NZ,7190
7195	ED436C71	LD	(716C),BC
7199	21783E	LD	HL,3E78
719C	CD3F72	CALL	723F
719F	CD3472	CALL	7234
71A2	CD4C72	CALL	724C
71A5	CD3472	CALL	7234
71A8	CDBE71	CALL	71BE

71AB	3EFE	LD	A,FE
71AD	D3FF	OUT	(FF),A
71AF	DBFE	IN	A,(FE)
71B1	E604	AND	04
71B3	C8	RET	Z
71B4	3A6F71	LD	A,(716F)
71B7	3D	DEC	A
71B8	CA7071	JP	Z,7170
71BB	C3A271	JP	71A2
71BE	3EEF	LD	A,EF
71C0	D3FF	OUT	(FF),A
71C2	DBFE	IN	A,(FE)
71C4	E604	AND	04
71C6	CCE871	CALL	Z,71E8
71C9	3EF7	LD	A,F7
71CB	D3FF	OUT	(FF),A
71CD	DBFE	IN	A,(FE)
71CF	E610	AND	10
71D1	CC0E72	CALL	Z,720E
71D4	2A6871	LD	HL,(7168)
71D7	23	INC	HL
71D8	226871	LD	(7168),HL
71DB	016871	LD	BC,7168
71DE	ED42	SBC	HL,BC
71E0	C0	RET	NZ
71E1	214871	LD	HL,7148
71E4	226871	LD	(7163),HL
71E7	C9	RET	
71E8	2A6A71	LD	HL,(716A)
71EB	01FF50	LD	BC,50FF
71EE	ED42	SBC	HL,BC
71F0	C8	RET	Z
71F1	21003D	LD	HL,3D00
71F4	ED4B6A71	LD	BC,(716A)
71F8	CD3F72	CALL	723F
71FB	ED4B6A71	LD	BC,(716A)
71FF	03	INC	BC
7200	ED436A71	LD	(716A),BC
7204	21A83E	LD	HL,3EA8
7207	CD3F72	CALL	723F
720A	CD3472	CALL	7234
720D	C9	RET	
720E	2A6A71	LD	HL,(716A)
7211	01E050	LD	BC,50E0
7214	ED42	SBC	HL,BC
7216	C8	RET	Z
7217	21003D	LD	HL,3D00
721A	ED4B6A71	LD	BC,(716A)
721E	CD3F72	CALL	723F
7221	ED4B6A71	LD	BC,(716A)
7225	0B	DEC	BC
7226	ED436A71	LD	(716A),BC
722A	21A83E	LD	HL,3EA8
722D	CD3F72	CALL	723F
7230	CD3472	CALL	7234
7233	C9	RET	
7234	010100	LD	BC,0001
7237	210014	LD	HL,1400

723A	ED42	SBC	HL,BC
723C	20FC	JR	NZ,723A
723E	C9	RET	
723F	3E08	LD	A,08
7241	F5	PUSH	AF
7242	7E	LD	A,(HL)
7243	02	LD	(BC),A
7244	23	INC	HL
7245	04	INC	B
7246	F1	POP	AF
7247	DE01	SBC	A,01
7249	20F6	JR	NZ,7241
724B	C9	RET	
724C	3A6E71	LD	A,(716E)
724F	3C	INC	A
7250	326E71	LD	(716E),A
7253	DE17	SBC	A,17
7255	CAA972	JP	Z,72A9
7258	3A6E71	LD	A,(716E)
725B	DE0F	SBC	A,0F
725D	CA8972	JP	Z,7289
7260	3A6E71	LD	A,(716E)
7263	DE07	SBC	A,07
7265	CA8972	JP	Z,7289
7268	ED4B6C71	LD	BC,(716C)
726C	21003D	LD	HL,3D00
726F	CD3F72	CALL	723F
7272	ED4B6C71	LD	BC,(716C)
7276	212000	LD	HL,0020
7279	ED4A	ADC	HL,BC
727B	226C71	LD	(716C),HL
727E	ED4B6C71	LD	BC,(716C)
7282	21783E	LD	HL,3E78
7285	CD3F72	CALL	723F
7288	C9	RET	
7289	ED4B6C71	LD	BC,(716C)
728D	21003D	LD	HL,3D00
7290	CD3F72	CALL	723F
7293	2A6C71	LD	HL,(716C)
7296	012007	LD	BC,0720
7299	ED4A	ADC	HL,BC
729B	226C71	LD	(716C),HL
729E	ED4B6C71	LD	BC,(716C)
72A2	21783E	LD	HL,3E78
72A5	CD3F72	CALL	723F
72A8	C9	RET	
72A9	3E01	LD	A,01
72AB	326F71	LD	(716F),A
72AE	2A6A71	LD	HL,(716A)
72B1	012000	LD	BC,0020
72B4	ED42	SBC	HL,BC
72B6	ED4B6C71	LD	BC,(716C)
72BA	ED42	SBC	HL,BC
72BC	C0	RET	NZ
72BD	11BC02	LD	DE,02BC
72C0	21C304	LD	HL,0403
72C3	CDB503	CALL	03B5
72C6	C9	RET	

72C7	210018	LD	HL,1800
72CA	110040	LD	DE,4000
72CD	3E00	LD	A,00
72CF	12	LD	(DE),A
72D0	13	INC	DE
72D1	010100	LD	BC,0001
72D4	ED42	SBC	HL,BC
72D6	C2CF72	JP	NZ,72CF
72D9	C9	RET	

As oito rotinas a seguir apresentadas são típicas dos programas em linguagem máquina que se podem escrever e utilizar para melhorar os programas em BASIC.

Rolar o «écran»

Podem fazer-se três coisas:

- 1) Rolar os atributos das cores.
- 2) Rolar as informações que estão no écran.
- 3) Ou, o que é mais útil, fazer ao mesmo tempo as duas coisas.

Vamos primeiro mostrar como se faz cada uma isoladamente; depois, é simples combinar os dois métodos.

Rolar os atributos da esquerda para a direita

Quando esta rotina for chamada fará mudar a posição das 32 colunas dos atributos de cor da esquerda para a direita. A coluna um será mudada para a coluna dois, a coluna dois para a três e por aí adiante até à coluna 32. Esta será mudada para a coluna 1, para dar um efeito de envolvimento.

Como funciona

Este exemplo mostra como a fila inferior do écran é mudada de posição da esquerda para a direita. O atributo mais à direita desta fila está na posição 23295. O registo A está carregado com o con-

teúdo desta posição. DE é colocado a 23295, HL a 23294 e BC a 31. Os dados no endereço contido por HL são então transferidos para o endereço contido por DE. HL e DE são então diminuídos e a transferência volta a efectuar-se. Tem de se fazer isto 31 vezes. Esta transferência de dados e a diminuição dos registos é efectuada por uma simples instrução chamada LDDR, a qual significa: «Carregar o endereço contido pelo registo DE com o conteúdo do endereço contido por HL.»

DE é então diminuído.

HL é então diminuído.

BC é então diminuído.

Se BC não é igual a zero verifica-se então uma outra transferência entre HL e DE. O ciclo de transferência e diminuição prossegue até que o valor de BC seja zero.

Dado que BC está carregado com 31, esta instrução prosseguirá até que o atributo na posição 23264 tenha sido transferido para a posição 23265. Temos então de carregar a posição 23264 com o conteúdo do registo A (lembramo-nos de que A estava carregado com um atributo na extrema direita); isto dá o efeito de envolvimento que desejamos.

As restantes 23 filas horizontais são mudadas de posição do mesmo modo.

FLUXOGRAMA DO PROGRAMA PARA ROLAR OS ATRIBUTOS DA ESQUERDA PARA A DIREITA

Carregar B,24.

O número de linhas a serem roladas

Carregar DE,24.

O endereço do atributo do lado direito do fundo

Carregar H L,23294

PUSH BC

Carregar B C,31.

O número de atributos numa linha a ser mudado para a direita

Carregar A,(DE).

Armazenar o atributo do lado direito no registo A

LDDR. Mudar o 31.º atributo para a 32.ª posição.

Mudar o 30.º atributo para a 31.ª posição e por aí adiante para mudar o primeiro atributo para o segundo

Carregar (HL),A. Pôr o valor do registo A que contém o 32.º atributo original no primeiro atributo. Isto dá o desejado efeito de envolvimento.

Colocar DE no endereço do atributo do lado direito da linha seguinte acima da que acaba de ser mudada

Colocar HL em um menos DE

POP BC

DEC BC

Salto se não for zero

RETORNO

Listagem em linguagem máquina do rolar atributos da esquerda para a direita

Endereço inicial 29000
Endereço final 29023
HEX total 2467

7148	0618	LD	B,18
714A	11FF5A	LD	DE,5AFF
714D	D5	PUSH	DE
714E	E1	POP	HL
714F	2B	DEC	HL
7150	C5	PUSH	BC
7151	011F00	LD	BC,001F
7154	1A	LD	A,(DE)
7155	EDB8	LDDR	
7157	23	INC	HL

7158	77	LD	(HL),A
7159	2B	DEC	HL
715A	2B	DEC	HL
715B	1B	DEC	DE
715C	C1	POP	BC
715D	10F1	DJNZ	7150
715F	C9	RET	

A rotina para mover os atributos da direita para a esquerda é muito semelhante à que serve para efectuar o movimento da esquerda para a direita. As diferenças são que começamos à esquerda da fila do topo e que a instrução LDDR é substituída pela LDIR, o que significa:

Carregar o endereço contido pelo registo DE com o conteúdo do endereço contido por HL.

DE é então diminuído.

HL é aumentado.

BC é diminuído.

Se BC não é igual a zero repete-se a transferência. A instrução LDIR é semelhante à instrução LDDR, com excepção de que os registos HL e DE são diminuídos.

Listagem em linguagem máquina para rolar os atributos da direita para a esquerda

Endereço inicial 29024
Endereço final 29047
HEX total 2178

7160	0618	LD	B,18
7162	110058	LD	DE,5800
7165	D5	PUSH	DE
7166	E1	POP	HL
7167	23	INC	HL
7168	C5	PUSH	BC
7169	011F00	LD	BC,001F
716C	1A	LD	A,(DE)

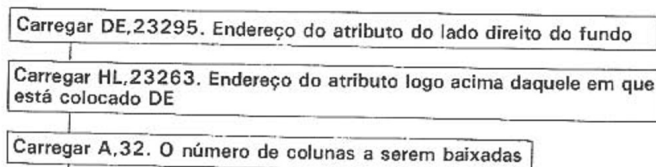
716D	EDB0	LDIR	
716F	2B	DEC	HL
7170	77	LD	(HL),A
7171	23	INC	HL
7172	23	INC	HL
7173	13	INC	DE
7174	C1	POP	BC
7175	10F1	DJNZ	7168
7177	C9	RET	

Rolar os atributos para baixo

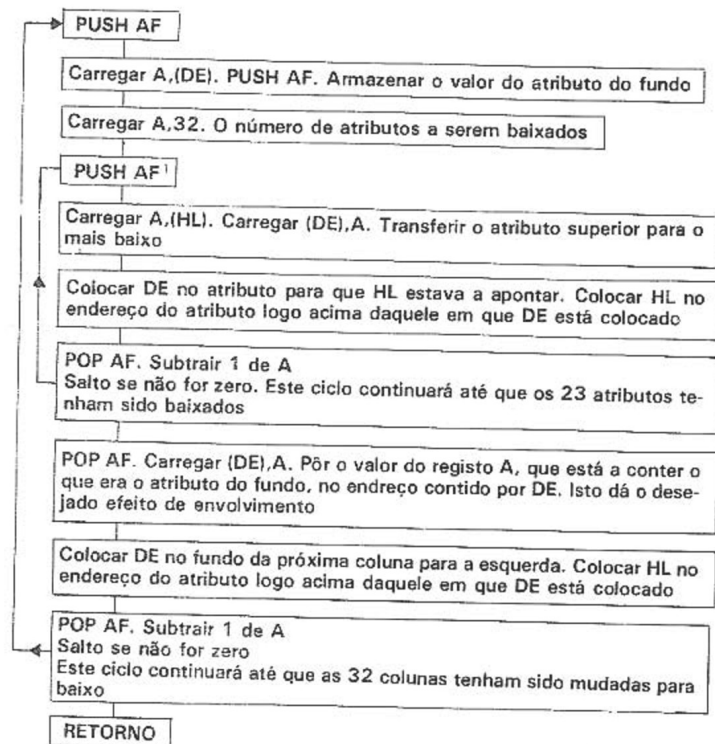
Os princípios fundamentais para rolar os atributos para baixo são muito semelhantes aos que usamos para os rolar para a esquerda ou direita. Cada uma das 32 colunas é baixada individualmente, começando pela da direita e prosseguindo até chegar à da esquerda. Cada coluna é mudada de posição, começando por armazenar o valor do atributo do fundo e em seguida fazendo o mesmo para os outros. No fim, o atributo armazenado é posto no topo da coluna.

O fluxograma seguinte diz respeito à rotina para baixar os atributos.

FLUXOGRAMA PARA ROLAR OS ATRIBUTOS PARA BAIXO



96



Listagem em linguagem máquina
para rolar os atributos para baixo

Endereço inicial	29048
Endereço final	29095
HEX total	5819

7178	11FF5A	LD	DE,5AFF
717B	21DF5A	LD	HL,5ADF
717E	3E20	LD	A,20

A. V. 89 - 7

97

7180	F5	PUSH	AF
7181	1A	LD	A,(DE)
7182	F5	PUSH	AF
7183	3E17	LD	A,17
7185	F5	PUSH	AF
7186	7E	LD	A,(HL)
7187	12	LD	(DE),A
7188	E5	PUSH	HL
7189	D1	POP	DE
718A	012000	LD	BC,0020
718D	ED42	SBC	HL,BC
718F	F1	POP	AF
7190	DE01	SBC	A,01
7192	20F1	JR	NZ,7185
7194	F1	POP	AF
7195	12	LD	(DE),A
7196	01FF02	LD	BC,02FF
7199	ED4A	ADC	HL,BC
719B	E5	PUSH	HL
719C	D1	POP	DE
719D	012000	LD	BC,0020
71A0	ED42	SBC	HL,BC
71A2	F1	POP	AF
71A3	DE01	SBC	A,01
71A5	20D9	JR	NZ,7180
71A7	C9	RET	

Rolar os atributos para cima

Como se esperaria, a maneira de o fazer é muito semelhante à que serviu para rolar os atributos para baixo. Cada coluna é mudada individualmente de posição, começando-se pela da esquerda e prosseguindo até chegar à da direita.

Listagem em linguagem máquina para rolar os atributos para cima

Endereço inicial	29096
Endereço final	29143
HEX total	5477

71A8	110058	LD	DE,5800
71AB	212058	LD	HL,5820
71AE	3E20	LD	A,20
71B0	F5	PUSH	AF
71B1	1A	LD	A,(DE)
71B2	F5	PUSH	AF
71B3	3E17	LD	A,17
71B5	F5	PUSH	AF
71B6	7E	LD	A,(HL)
71B7	12	LD	(DE),A
71B8	E5	PUSH	HL
71B9	D1	POP	DE
71BA	012000	LD	BC,0020
71BD	ED4A	ADC	HL,BC
71BF	F1	POP	AF
71C0	DE01	SBC	A,01
71C2	20F1	JR	NZ,71B5
71C4	F1	POP	AF
71C5	12	LD	(DE),A
71C6	01FF02	LD	BC,02FF
71C9	ED42	SBC	HL,BC
71CB	E5	PUSH	HL
71CC	D1	POP	DE
71CD	012000	LD	BC,0020
71D0	ED4A	ADC	HL,BC
71D2	F1	POP	AF
71D3	DE01	SBC	A,01
71D5	20D9	JR	NZ,71B0
71D7	C9	RET	

O programa em BASIC que se segue pode ser usado para demonstrar as quatro rotinas para rolar os atributos. Dependendo de qual dos quatro cursores for pressionado, assim será chamada uma determinada rotina. Por exemplo, se for batida a tecla 5 será chamada a rotina para rolar os atributos da direita para a esquerda.

Este mesmo programa também mostra como usar a linguagem máquina para melhorar os seus programas em BASIC. Com efeito, a linguagem máquina torna-se uma sub-rotina do programa em BASIC. Podem ter-se na memória tantas rotinas quanto se quiser e chamá-las individualmente do programa em BASIC.

É também uma boa altura para se falar de RAMTOP. Se olharmos para a primeira linha do programa Hexent, vemos que ela con-

siste em CLEAR 28999. Isto pode ser considerado como uma barreira entre o programa em BASIC e a linguagem máquina. Sem ele a linguagem máquina seria provavelmente sobreposta pelo programa em BASIC. Limita a dimensão do programa em BASIC que pode ser escrito e os possuidores de *Spectrum 48K* acharão mais prático colocar RAMTOP a 39999 e escrever as rotinas em linguagem máquina acima deste endereço.

```

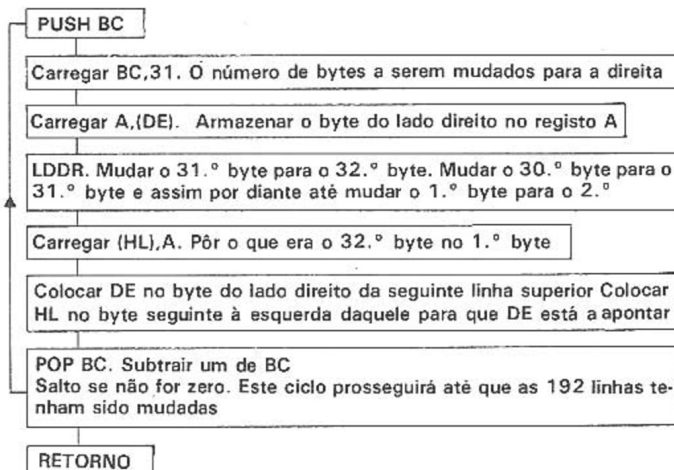
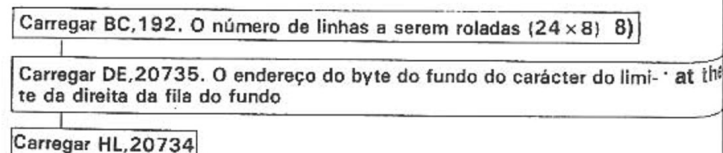
400 LET I=8
405 FOR A=22528 TO 23295
410 POKE A,I
415 LET I=I+8
420 IF I=64 THEN LET I=8
425 NEXT A
430 IF INKEYS="5" THEN RANDOMIZE USR 29024
435 IF INKEYS="8" THEN RANDOMIZE USR 29000
440 IF INKEYS="6" THEN RANDOMIZE USR 29048
445 IF INKEYS="7" THEN RANDOMIZE USR 29096
450 GOTO 430

```

Rolar o texto da esquerda para a direita

A rotina para rolar o texto da esquerda para a direita é quase idêntica à rotina para rolar os atributos da esquerda para a direita; as diferenças óbvias são os endereços iniciais e o facto de que têm de ser mudadas 192 filas em vez de 24.

FLUXOGRAMA DO PROGRAMA PARA ROLAR O TEXTO DA ESQUERDA PARA A DIREITA



Listagem em linguagem máquina para rolar o texto da esquerda para a direita

Endereço inicial 29144
Endereço final 29167
HEX total 2932

71D8	06C0	LD	B,C0
71DA	11FF57	LD	DE,57FF
71DD	D5	PUSH	DE
71DE	E1	POP	HL
71DF	2B	DEC	HL
71E0	C5	PUSH	BC
71E1	011F00	LD	BC,001F
71E4	1A	LD	A,(DE)
71E5	EDB8	LDDR	
71E7	23	INC	HL
71E8	77	LD	(HL),A
71E9	2B	DEC	HL
71EA	2B	DEC	HL
71EB	1B	DEC	DE
71EC	C1	POP	BC
71ED	10F1	DJNZ	71EQ
71EF	C9	RET	

Rolar o texto da direita para a esquerda

Como já devem ter pensado, esta rotina é quase idêntica à rotina usada para rolar os atributos da esquerda para a direita.

Listagem em linguagem máquina para rolar o texto da direita para a esquerda

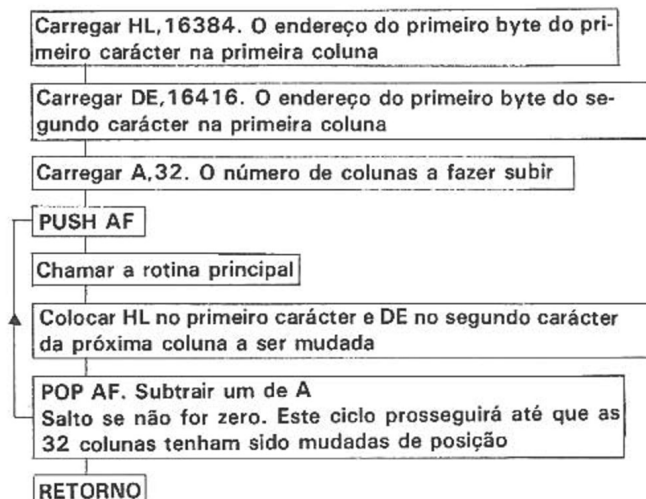
Endereço inicial 29168
Endereço final 29191
HEX total 2322

71F0	06C0	LD	B,C0
71F2	110040	LD	DE,4000
71F5	D5	PUSH	DE
71F6	E1	POP	HL
71F7	23	INC	HL
71F8	C5	PUSH	BC
71F9	011F00	LD	BC,001F
71FC	1A	LD	A,(DE)
71FD	EDB0	LDIR	
71FF	2B	DEC	HL
7200	77	LD	(HL),A
7201	23	INC	HL
7202	23	INC	HL
7203	13	INC	DE
7204	C1	POP	BC
7205	10F1	DJNZ	71F8
7207	C9	RET	

Rolar o texto para cima

As rotinas para rolar o texto do *écran* para cima e para baixo são as mais difíceis, principalmente por causa dos agora famosos saltos não sequenciais entre as linhas 7 e 8 e 15 e 16. O fluxograma seguinte mostra a rotina de controle deste programa.

FLUXOGRAM DA ROTINA DE CONTROLE PARA MOVER O TEXTO PARA CIMA



A tarefa da rotina de controle é colocar HL no primeiro byte do primeiro carácter e DE no primeiro byte do segundo carácter da coluna que é movida para cima. A rotina principal é então chamada, e realiza a tarefa de mover a coluna para cima.

A rotina principal

A primeira coisa a fazer é colocar em lugar seguro os oito bytes do primeiro carácter. Depois de a coluna ter sido deslocada para cima esses oito bytes são recuperados e colocados na base da coluna para dar o efeito de envolvimento. Por causa dos saltos de linha não sequenciais, temos agora de nos concentrar em cada terço da coluna de per si. No terço superior há sete caracteres que precisam de ser mudados para cima na sequência seguinte:

1 para 0
2 para 1
3 para 2
4 para 3
5 para 4
6 para 5
7 para 6

Isto faz-se carregando o registo A com 7, número de caracteres a ser movido, depois chama-se a rotina chamada «Mudar bloco». Esta rotina fará mover os caracteres no sentido ascendente. Depois de se ter mudado a posição do terço superior da coluna é preciso fazer o mesmo para o terço do meio. Primeiro HL e DE devem ser ajustados para o salto de linha não sequencial, o que se faz chamando uma rotina designada por «Salto de linha». Depois de se ter ajustado HL e DE, o registo A é então carregado com 8, e é chamada a rotina para mudar o bloco. Para mudar o terço inferior da coluna chama-se a rotina de salto de linha, A é carregado com 8 e chama-se a rotina «Mudar bloco». A seguir o carácter que estava no topo da coluna é retirado e colocado na base da mesma. Feito isto, faremos RETURN para a rotina de controle.

Rotina para mudar bloco

Esta rotina fará subir sete ou oito caracteres, dependendo de que valor se colocou no registo A por meio da rotina principal antes de ela ter sido chamada. Viu-se anteriormente que um carácter é constituído por oito bytes e por conseguinte para o subir é preciso que os oito bytes o façam. A parte principal desta rotina compete efectuar a transferência dos oito bytes de uma posição em que está o carácter para a posição seguinte.

Rotina do salto de linha

Esta rotina ajusta o valor de DE de modo que ele é correcto para o próximo terço do écran. HL também será colocado no endereço correcto devido à acção da rotina para mudar bloco.

Listagem em linguagem máquina para rolar o texto para cima

Endereço inicial 29192
Endereço final 29333
HEX total 15983

7208	112040	LD	DE,4020
720B	210040	LD	HL,4000
7210	3E20	LD	A,20
7211	F5	PUSH	AF
7214	CD2772	CALL	7227
7217	11DF18	LD	DE,18DF
7219	ED52	SBC	HL,DE
721A	E5	PUSH	HL
721D	012000	LD	BC,0020
721E	09	ADD	HL,BC
721F	54	LD	D,H
7220	5D	LD	E,L
7221	E1	POP	HL
7222	F1	POP	AF
7224	DE01	SBC	A,01
7226	20EA	JR	NZ,7210
7227	C9	RET	
7228	E5	PUSH	HL
722B	019672	LD	BC,7296
722D	3E08	LD	A,08
722E	F5	PUSH	AF
722F	7E	LD	A,(HL)
7230	02	LD	(BC),A
7231	24	INC	H
7232	03	INC	BC
7233	F1	POP	AF
7235	DE01	SBC	A,01
7237	20F6	JR	NZ,722D
7238	E1	POP	HL
723A	3E07	LD	A,07
723D	CD6B72	CALL	726B
7240	CD6072	CALL	7260
7242	3E08	LD	A,08
7244	CD6B72	CALL	726B
7245	CD6072	CALL	7260
7248	3E08	LD	A,08
724A	CD6B72	CALL	726B
724D	110001	LD	DE,0100
7250	019672	LD	BC,7296
7253	3E08	LD	A,08
7255	F5	PUSH	AF
7256	0A	LD	A,(BC)
7257	77	LD	(HL),A
7258	19	ADD	HL,DE
7259	03	INC	BC
725A	F1	POP	AF
725B	DE01	SBC	A,01
725D	20F6	JR	NZ,7255
725F	C9	RET	

7260	E5	PUSH	HL
7261	D5	PUSH	DE
7262	E1	POP	HL
7263	010007	LD	BC,0700
7266	09	ADD	HL,BC
7267	E5	PUSH	HL
7268	D1	POP	DE
7269	E1	POP	HL
726A	C9	RET	
726B	F5	PUSH	AF
726C	3E08	LD	A,08
726E	F5	PUSH	AF
726F	1A	LD	A,(DE)
7270	77	LD	(HL),A
7271	010001	LD	BC,0100
7274	09	ADD	HL,BC
7275	E5	PUSH	HL
7276	D5	PUSH	DE
7277	E1	POP	HL
7278	09	ADD	HL,BC
7279	E5	PUSH	HL
727A	D1	POP	DE
727B	E1	POP	HL
727C	F1	POP	AF
727D	DE01	SBC	A,01
727F	20ED	JR	NZ,726E
7281	D5	PUSH	DE
7282	E1	POP	HL
7283	010008	LD	BC,0800
7286	ED42	SBC	HL,BC
7288	E5	PUSH	HL
7289	012000	LD	BC,0020
728C	09	ADD	HL,BC
728D	E5	PUSH	HL
728E	D1	POP	DE
728F	E1	POP	HL
7290	F1	POP	AF
7291	DE01	SBC	A,01
7293	20D6	JR	NZ,726B
7295	C9	RET	

Rolar o texto para baixo

O programa para rolar o texto para baixo é muito semelhante ao programa que o faz rolar para cima. As principais diferenças são:

- 1) Começa-se com a coluna da extrema direita, que é deslocada para baixo, e continua-se a mover as colunas até se chegar à da extrema esquerda.

- 2) O carácter do fundo de cada coluna é armazenado e quando se baixou a coluna é posto no topo, para lhe dar a volta.
- 3) A rotina bloco move os caracteres para baixo em vez de para cima.

Listagem em linguagem máquina para rolar o texto para baixo

Endereço inicial 29342
Endereço final 29489
HEX total 18002

729E	11DF50	LD	DE,50DF
72A1	21FF50	LD	HL,50FF
72A4	3E20	LD	A,20
72A6	F5	PUSH	AF
72A7	CDBE72	CALL	72BE
72AA	11DF08	LD	DE,08DF
72AD	ED5A	ADC	HL,DE
72AF	E5	PUSH	HL
72B0	012000	LD	BC,0020
72B3	ED42	SBC	HL,BC
72B5	54	LD	D,H
72B6	5D	LD	E,L
72B7	E1	POP	HL
72B8	F1	POP	AF
72B9	DE01	SBC	A,01
72BB	20E9	JR	NZ,72A6
72BD	C9	RET	
72BE	E5	PUSH	HL
72BF	013273	LD	BC,7332
72C2	3E08	LD	A,08
72C4	F5	PUSH	AF
72C5	7E	LD	A,(HL)
72C6	02	LD	(BC),A
72C7	24	INC	H
72C8	03	INC	BC
72C9	F1	POP	AF
72CA	DE01	SBC	A,01
72CC	20F6	JR	NZ,72C4
72CE	E1	POP	HL
72CF	3E07	LD	A,07
72D1	CD872	CALL	72F8
72D4	CD2673	CALL	7326
72D7	3E08	LD	A,08
72D9	CD872	CALL	72F8
72DC	CD2673	CALL	7326
72DF	3E08	LD	A,08
72E1	CD872	CALL	72F8
72E4	110001	LD	DE,0100
72E7	013273	LD	BC,7332
72EA	3E08	LD	A,08
72EC	F5	PUSH	AF
72ED	0A	LD	A,(BC)

72EE	77	LD	(HL),A
72EF	ED5A	ADC	HL,DE
72F1	03	INC	BC
72F2	F1	POP	AF
72F3	DE01	SBC	A,01
72F5	20F5	JR	NZ,72EC
72F7	C9	RET	
72F8	F5	PUSH	AF
72F9	3E08	LD	A,08
72FB	F5	PUSH	AF
72FC	1A	LD	A,(DE)
72FD	77	LD	(HL),A
72FE	010001	LD	BC,0100
7301	ED4A	ADC	HL,BC
7303	E5	PUSH	HL
7304	D5	PUSH	DE
7305	E1	POP	HL
7306	ED4A	ADC	HL,BC
7308	E5	PUSH	HL
7309	D1	POP	DE
730A	E1	POP	HL
730B	F1	POP	AF
730C	DE01	SBC	A,01
730E	20EB	JR	NZ,72FB
7310	D5	PUSH	DE
7311	E1	POP	HL
7312	010008	LD	BC,0800
7315	ED42	SBC	HL,BC
7317	E5	PUSH	HL
7318	012000	LD	BC,0020
731B	ED42	SBC	HL,BC
731D	E5	PUSH	HL
731E	D1	POP	DE
731F	E1	POP	HL
7320	F1	POP	AF
7321	DE01	SBC	A,01
7323	20D3	JR	NZ,72F8
7325	C9	RET	
7326	E5	PUSH	HL
7327	62	LD	H,D
7328	6B	LD	L,E
7329	010007	LD	BC,0700
732C	ED42	SBC	HL,BC
732E	54	LD	D,H
732F	5D	LD	E,L
7330	E1	POP	HL
7331	C9	RET	

O seguinte programa em BASIC pode ser usado para demonstrar as quatro rotinas para rolar o texto.

```

400 FOR A=0 TO 21
405 FOR B=0 TO 31
410 PRINT AT A,B; CHR$(B+65)
415 NEXT B

```

```

420 NEXT A
425 IF INKEYS = "5" THEN RANDOMIZE USR 29168
430 IF INKEYS = "8" THEN RANDOMIZE USR 29144
435 IF INKEYS = "6" THEN RANDOMIZE USR 29342
440 IF INKEYS = "7" THEN RANDOMIZE USR 29192
445 GOTO 425

```

A rotina BASIC acima indicada encherá o *écran* com texto. Depois, se se pressionar as teclas 5, 6, 7 ou 8, pode-se rolar o texto em qualquer direcção.

O seguinte programa em BASIC mostra como combinar as rotinas para rolar de modo que as cores e o texto sejam rolados juntamente:

```

400 LET I=8
401 LET X=22528
404 FOR A=0 TO 21
405 FOR B=0 TO 31
410 PRINT AT A,B;CHR$(B+65)
411 POKE X,I
412 LET X=X+1
413 LET I=I+8
414 IF I=64 THEN LET I=8
415 NEXT B
420 NEXT A
425 IF INKEYS="5" THEN GO SUB 500
430 IF INKEYS="8" THEN GO SUB 550
435 IF INKEYS="6" THEN GO SUB 600
440 IF INKEYS="7" THEN GO SUB 650
445 GO TO 425
500 RANDOMIZE USR 29168
505 RANDOMIZE USR 29024
510 RETURN
550 RANDOMIZE USR 29144
555 RANDOMIZE USR 29000
560 RETURN
600 RANDOMIZE USR 29342
605 RANDOMIZE USR 29048
610 RETURN
650 RANDOMIZE USR 29192
655 RANDOMIZE USR 29096
660 RETURN

```

APÊNDICE

- 1) OPCODES Z80
- 2) TABELA DE CONVERSÃO DE DECIMAL EM HEX
- 3) TABELA DE CÁLCULO DO SALTO RELATIVO
- 4) TABELA DO FICHEIRO VISUAL
- 5) TABELA DO FICHEIRO DE ATRIBUTOS
- 6) O SISTEMA DE NUMERAÇÃO BINÁRIA
- 7) RESPOSTAS

Nota do autor

Espero ter realizado o que pretendi fazer, que era proporcionar ao leitor uma introdução suave ao mundo da programação em linguagem máquina. Devem nesta altura ter descoberto se se dão bem com a linguagem máquina. Existem muitos opcodes Z80 de que não tratei neste livro; se pretenderem conhecê-los, existem alguns livros que tratam especificamente deles com grandes pormenores.

Quando estiverem a escrever programas em linguagem máquina pode acontecer que se lhes deparem alguns problemas que não sejam capazes de resolver. Se me escreverem por intermédio dos Editores desta obra, terei muito prazer em tentar dar-lhe uma ajuda na sua resolução, mas só o farei se me enviarem um envelope selado e com o endereço.

Já expliquei que os programas são muito mais extensos do que o necessário e que isso acontece intencionalmente, para tornar mais fácil a sua compreensão. Por isso não me enviem cartas a dizer coisas tais como «consegui encurtar determinada rotina 10 bytes» ou «encontrei um modo melhor de deslocar os atributos». Tais cartas dêito-as para o cesto dos papéis.

STEVE WEBB
Londres
Maio de 1984

Opcodes Z80

ADC A, (HL)	8E	AND C	A1
ADC A, (IX + d)	DD8Ed	AND D	A2
ADC A, (IY + d)	FD8Ed	AND E	A3
ADC A, A	8F	AND H	A4
ADC A, B	88	AND L	A5
ADC A, C	89	AND n	E6n
ADC A, D	8A	BIT 0, (HL)	CB46
ADC A, E	8B	BIT 0, (IX + D)	DDCBd46
ADC A, H	8C	BIT 0, (IY + d)	FDCBd46
ADC A, L	8D	BIT 0, A	CB47
ADC A, n	CEn	BIT 0, B	CB40
ADC HL, BC	ED4A	BIT 0, C	CB41
ADC HL, DE	ED5A	BIT 0, D	CB42
ADC HL, HL	ED6A	BIT 0, E	CB43
ADC HL, SP	ED7A	BIT 0, H	CB44
ADD A, (HL)	86	BIT 0, L	CB45
ADD A, (IX + d)	DD86d	BIT 1, (HL)	CB4E
ADD A, (IY + d)	FD86d	BIT 1, (IX + d)	DDCBd4E
ADD A, A	87	BIT 1, (IY + d)	FDCBd4E
ADD A, B	80	BIT 1, A	CB4F
ADD A, C	81	BIT 1, B	CB48
ADD A, D	82	BIT 1, C	CB49
ADD A, E	83	BIT 1, D	CB4A
ADD A, H	84	BIT 1, E	CB4B
ADD A, L	85	BIT 1, H	CB4C
ADD A, n	C6n	BIT 1, L	CB4D
ADD HL, BC	09	BIT 2, (HL)	CB56
ADD HL, DE	19	BIT 2, (IX + d)	DDCBd56
ADD HL, HL	29	BIT 2, (IY + d)	FDCBd56
ADD HL, SP	39	BIT 2, A	CB57
ADD IX, BC	DD09	BIT 2, B	CB50
ADD IX, DE	DD19	BIT 2, C	CB51
ADD IX, IX	DD29	BIT 2, D	CB52
ADD IX, SP	DD39	BIT 2, E	CB53
ADD IY, BC	FD09	BIT 2, H	CB54
ADD IY, DE	FD19	BIT 2, L	CB55
ADD IY, IY	FD29	BIT 3, (HL)	CB5E
ADD IY, SP	FD39	BIT 3, (IX + d)	DDCBd5E
AND(HL)	A6	BIT 3, (IY + d)	FDCBd5E
AND (IX + d)	DDA6d	BIT 3, A	CB5F
AND (IY + d)	FDA6d	BIT 3, B	CB58
AND A	A7	BIT 3, C	CB59
AND B	A0	BIT 3, D	CB5A
		BIT 3, E	CB5B
		BIT 3, H	CB5C
		BIT 3, L	CB5D
		BIT 4, (HL)	CB66

BIT 4, (IX + d)	DDCBd66	CALL Z, nn	CCnn	IM 2	ED5E	LD (BC), A	02
BIT 4, (IY + d)	FDCBd66	CCF	3F	IN A, (C)	ED78	LD (DE), A	12
BIT 4, A	CB67	CP (HL)	BE	IN A, (n)	DBn	LD (HL), A	77
BIT 4, B	CB60	CP (IX + d)	DDBE d	IN B, (C)	ED40	LD (HL), B	70
BIT 4, C	CB61	CP (IY + d)	FDBE d	IN C, (C)	ED48	LD (HL), C	71
BIT 4, D	CB62	CP A	BF	IN D, (C)	ED50	LD (HL), D	72
BIT 4, E	CB63	CP B	B8	IN E, (C)	ED58	LD (HL), E	73
BIT 4, H	CB64	CP C	B9	IN H, (C)	ED60	LD (HL), H	74
BIT 4, L	CB65	CP D	BA	IN L, (C)	ED68	LD (HL), L	75
BIT 5, (HL)	CB6E	CP E	BB	INC (HL)	34	LD (HL), n	36n
BIT 5, (IX + d)	DDCBd6E	CP H	BC	INC (IX + d)	DD34d	LD (IX + d), A	DD77d
BIT 5, (IY + D)	FDCBd6E	CP L	BD	INC (IY + d)	FD34d	LD (IX + d), B	DD70d
BIT 5, A	CB6F	CP n	FEn	INC A	3C	LD (IX + d), C	DD71d
BIT 5, B	CB68	CPD	EDA9	INC B	04	LD (IX + d), D	DD72d
BIT 5, C	CB69	CPDR	EDB9	INC BC	03	LD (IX + d), E	DD73d
BIT 5, D	CB6A	CPI	EDA1	INC C	0C	LD (IX + d), H	DD74d
BIT 5, E	CB6B	CPiR	EDB1	INC D	14	LD (IX + d), L	DD75d
BIT 5, H	CB6C	CPL	2F	INC DE	13	LD (IX + d), n	DD36dn
BIT 5, L	CB6D	DAA	27	INC E	1C	LD (IY + d), A	FD77d
BIT 6, (HL)	CB76	DEC (HL)	35	INC H	24	LD (IY + d), B	FD70d
BIT 6, (IX + d)	DDCBd76	DEC (IX + d)	DD35d	INC HL	23	LD (IY + d), C	FD71d
BIT 6, (IY + d)	FDCBd76	DEC (IY + d)	FD35d	INC IX	DD23	LD (IY + d), D	FD72d
BIT 6, A	CB77	DEC A	3D	INC IY	FD23	LD (IY + d), E	FD73d
BIT 6, B	CB70	DEC B	05	INC L	2C	LD (IY + d), H	FD74d
BIT 6, C	CB71	DEC BC	0B	INC SP	33	LD (IY + d), L	FD75d
BIT 6, D	CB72	DEC C	0D	IND	EDAA	LD (IY + d), n	FD36dn
BIT 6, E	CB73	DEC D	15	INDR	EDBA	LD (nn), A	32nn
BIT 6, H	CB74	DEC DE	1B	INI	EDA2	LD (nn), BC	ED43nn
BIT 6, L	CB75	DEC E	1D	INIR	EDB2	LD (nn), DE	ED53nn
BIT 7, (HL)	CB7E	DEC H	25	JP (HL)	E9	LD (nn), HL	22nn
BIT 7, (IX + d)	DDCBd7E	DEC HL	2B	JP (IX)	DDE9	LD (nn), IX	DD22nn
BIT 7, (IY + d)	FDCBd7E	DEC IX	DD2B	JP (IY)	FDE9	LD (nn), IY	FD22nn
BIT 7, A	CB7F	DEC IY	FD2B	JP C, nn	DAnn	LD (nn), SP	ED73nn
BIT 7, B	CB78	DEC L	2D	JP M, nn	FAnn	LD A, (BC)	0A
BIT 7, C	CB79	DEC SP	3B	JP NC, nn	D2nn	LD A, (DE)	1A
BIT 7, D	CB7A	DI	F3	JP nn	C3nn	LD A, (HL)	7E
BIT 7, E	CB7B	DJNZ, d	10d	JP NZ, nn	C2nn	LD A, (IX + d)	DD7Ed
BIT 7, H	CB7C	E1	FB	JP P, nn	F2nn	LD A, (IY + d)	FD7Ed
BIT 7, L	CB7D	EX (SP), HL	E3	JP PE, nn	EAnn	LD A, (nn)	3Ann
CALL C, nn	DCnn	EX (SP), IX	DDE3	JP PO, nn	E2nn	LD A, A	7F
CALL M, nn	FCnn	EX (SP), IY	FDE3	JP Z, nn	CAnn	LD A, B	78
CALL NC, nn	D4nn	EX AF, AF	08	JR C, d	38d	LD A, C	79
CALL nn	CDnn	EX DE, HL	EB	JR, d	18d	LD A, D	7A
CALL NZ, nn	C4nn	EXX	D9	JR NC, d	30d	LD A, E	7B
CALL P, nn	F4nn	HALT	76	JR NZ, d	20d	LD A, H	7C
CALL PE, nn	ECnn	IM 0	ED46	JR Z, d	28d	LD A, I	ED57
CALL PO, nn	E4nn	IM 1	ED56				

LD A, L 7D
 LD A, n 3En
 LD B, (HL) 46
 LD B, (IX + d) DD46d
 LD B, (IY + d) FD46d
 LD B, A 47
 LD B, B 40
 LD B, C 41
 LD B, D 42
 LD B, E 43
 LD B, H 44
 LD B, L 45
 LD B, n 06n
 LD B, C (nn) ED4Bnn
 LD B, C nn 01nn
 LD C, (HL) 4E
 LD C, (IX + d) DD4Ed
 LD C, (IY + d) FD4Ed
 LD C, A 4F
 LD C, B 48
 LD C, C 49
 LD C, D 4A
 LD C, E 4B
 LD C, H 4C
 LD C, L 4D
 LD C, n 0En
 LD D, (HL) 56
 LD D, (IX + d) DD56d
 LD D, (IY + d) FD56d
 LD D, A 57
 LD D, B 50
 LD D, C 51
 LD D, D 52
 LD D, E 53
 LD D, H 54
 LD D, L 55
 LD D, n 16n
 LD DE, (nn) ED58nn
 LD DE, nn 11nn
 LD E, (HL) 5E
 LD E, (IX + d) DD5Ed
 LD E, (IY + d) FD5Ed
 LD E, A 5F
 LD E, B 58
 LD E, C 59
 LD E, D 5A
 LD E, E 5B

LD E, H 5C
 LD E, L 5D
 LD E, n 1En
 LD H, (HL) 66
 LD H, (IX + d) DD66d
 LD H, (IY + d) FF66d
 LD H, A 67
 LD H, B 60
 LD H, C 61
 LD H, D 62
 LD H, E 63
 LD H, H 64
 LD H, L 65
 LD H, n 26n
 LD HL, (nn) 2Ann
 LD HL, nn 21nn
 LD I, A ED47
 LD IX, (nn) DD2Ann
 LD IX, nn DD21nn
 LD IY, (nn) FD2Ann
 LD IY, nn FD21nn
 LD L, (HL) 6E
 LD L, (IX + d) DD6Ed
 LD L, (IY + d) FD6Ed
 LD L, A 6F
 LD L, B 68
 LD L, C 69
 LD L, D 6A
 LD L, E 6B
 LD L, H 6C
 LD L, L 6D
 LD L, n 2En
 LD SP, (nn) ED7Bnn
 LD SP, HL F9
 LD SP, IX DDF9
 LD SP, IY FDF9
 LD SP, nn 31nn
 LDD EDA8
 LDDR EDB8
 LDI EDAO
 LDIR EDB0
 NEG ED44
 NOP 00
 OR (HL) B6
 OR (IX + d) DOB6d
 OR (IY + d) FDB6d
 OR A B7

OR B B0
 OR C B1
 OR D B2
 OR E B3
 OR H B4
 OR L B5
 OR n F6n
 OTDR EDBB
 OTIR EDB3
 OUT (C), A ED79
 OUT (C), B ED41
 OUT (C), C ED49
 OUT (C), D ED51
 OUT (C), E ED59
 OUT (C), H ED61
 OUT (C), L ED69
 OUT (n), A D3n
 OUTD EDAB
 OUTI EDA3
 POPAF F1
 POP BC C1
 POP DE D1
 POP HL E1
 POP IX DDE1
 POP IY FDE1
 PUSH AF F5
 PUSH BC C5
 PUSH DE D5
 PUSH HL E5
 PUSH IX DDE5
 PUSH IY FDE5
 RES 0, (HL) CB86
 RES 0, (IX + d) DDCBd86
 RES 0, (IY + d) FDCBd86
 RES 0, A CB87
 RES 0, B CB80
 RES 0, C CB81
 RES 0, D CB82
 RES 0, E CB83
 RES 0, H CB84
 RES 0, L CB85
 RES 1, (HL) CB8E
 RES 1, (IX + d) DDCBd8E
 RES 1, (IY + d) FDCBd8E
 RES 1, A CB8F
 RES 1, B CB88
 RES 1, C CB89

RES 1, D CB8A
 RES 1, E CB8B
 RES 1, H CB8C
 RES 1, L CB8D
 RES 2, (HL) CB96
 RES 2, (IX + d) DDCBd96
 RES 2, (IY + d) FDCBd96
 RES 2, A CB97
 RES 2, B CB90
 RES 2, C CB91
 RES 2, D CB92
 RES 2, E CB93
 RES 2, H CB94
 RES 2, L CB95
 RES 3, (HL) CB9E
 RES 3, (IX + d) DDCBd9E
 RES 3, (IY + d) FDCBd9E
 RES 3, A CB9F
 RES 3, B CB98
 RES 3, C CB99
 RES 3, D CB9A
 RES 3, E CB9B
 RES 3, H CB9C
 RES 3, L CB9D
 RES 4, (HL) CBA6
 RES 4, (IX + d) DDCBdA6
 RES 4, (IY + d) FDCBdA6
 RES 4, A CBA7
 RES 4, B CBA0
 RES 4, C CBA1
 RES 4, D CBA2
 RES 4, E CBA3
 RES 4, H CBA4
 RES 4, L CBA5
 RES 5, (HL) CBAE
 RES 5, (IX + d) DDCBdAE
 RES 5, (IY + d) FDCBdAE
 RES 5, A CBAF
 RES 5, B CBA8
 RES 5, C CBA9
 RES 5, D CBAA
 RES 5, E CBAB
 RES 5, H CBAC
 RES 5, L CBAD
 RES 6, (HL) CBB6
 RES 6, (IX + d) DDCBdB6
 RES 6, (IY + d) FDCBdB6

RES 6, A	CBB7
RES 6, B	CBB0
RES 6, C	CBB1
RES 6, D	CBB2
RES 6, E	CBB3
RES 6, H	CBB4
RES 6, L	CBB5
RES 7, (HL)	CBBE
RES 7, (IX + d)	DDCBdBE
RES 7, (IY + d)	FDCBdBE
RES 7, A	CBBF
RES 7, B	CBB8
RES 7, C	CBB9
RES 7, D	CBBA
RES 7, E	CBBB
RES 7, H	CBBC
RES 7, L	CBBD
RET	C9
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET P0	E0
RET Z	C8
RETI	ED4D
RETN	ED45
RL (HL)	CB16
RL (IX + d)	DDCBd16
RL (IY + d)	FDCBd16
RL A	CB17
RL B	CB10
RL C	CB11
RL D	CB12
RL E	CB13
RL H	CB14
RL L	CB15
RLA	17
RLC (HL)	CB06
RLC (IX + d)	DDCBd06
RLC (IY + d)	FDCBd06
RLC A	CB07
RLC B	CB00
RLC C	CB01
RLC D	CB02
RLC E	CB03

RLC H	CB04
RLC L	CB05
RLCA	07
RLD	ED6F
RR (HL)	CB1E
RR (IX + d)	DDC8d1E
RR (IY + d)	FDCBd1E
RR A	CB1F
RR B	CB18
RR C	CB19
RR D	CB1A
RR E	CB1B
RR H	CB1C
RR L	CB1D
RRR	1F
RRC (HL)	CB0E
RRC (IX + d)	DDCBd0E
RRC (IY + d)	FDCBd0E
RRC A	CB0F
RRC B	CB08
RRC C	CB09
RRC D	CB0A
RRC E	CB0B
RRC H	CB0C
RRC L	CB0D
RRC A	0F
RRD	ED67
RST 0	C7
RST10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF
RST 8	CF
SBC A, (HL)	9E
SBC A, (IX + d)	DD9Ed
SBC A, (IY + d)	FD9Ed
SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D
SBC A, n	DEn
SBC HL, BC	ED42

SBC HL, DE	ED52
SBC HL, HL	ED62
SBC HL, SP	ED72
SCF	37
SET 0, (HL)	CBC6
SET 0, (IX + d)	DDCBdC6
SET 0, (IY + d)	FDCBdC6
SET 0, A	CBC7
SET 0, B	CBC0
SET 0, C	CBC1
SET 0, D	CBC2
SET 0, E	CBC3
SET 0, H	CBC4
SET 0, L	CBC5
SET 1, (HL)	CBCE
SET 1, (IX + d)	DDCBdCE
SET 1, (IY + d)	FDCBdCE
SET 1, A	CBCF
SET 1, B	CBC8
SET 1, C	CBC9
SET 1, D	CBCA
SET 1, E	CBCB
SET 1, H	CBCC
SET 1, L	CBCE
SET 2, (HL)	CBD6
SET 2, (IX + d)	DDCBdD6
SET 2, (IY + d)	FDCBdD6
SET 2, A	CBD7
SET 2, B	CBD0
SET 2, C	CBD1
SET 2, D	CBD2
SET 2, E	CBD3
SET 2, H	CBD4
SET 2, L	CBD5
SET 3, (HL)	CBDE
SET 3, (IX + d)	DDCBdDE
SET 3, (IY + d)	FDCBdDE
SET 3, A	CBDF
SET 3, B	CBD8
SET 3, C	CBD9
SET 3, D	CBDA
SET 3, E	CBDB
SET 3, H	CBDC
SET 3, L	CBDD
SET 4, (HL)	CBE6
SET 4, (IX + d)	DDCbE6
SET 4, (IY + d)	FDCBdE6

SET 4, A	CBE7
SET 4, B	CBE0
SET 4, C	CBE1
SET 4, D	CBE2
SET 4, E	CBE2
SET 4, H	CBE4
SET 4, L	CBE5
SET 5, (HL)	CBEE
SET 5, (IX + d)	DDCBdEE
SET 5, (IY + d)	FDCBdEE
SET 5, A	CBEF
SET 5, B	CBE8
SET 5, C	CBE9
SET 5, D	CBEA
SET 5, E	CBEB
SET 5, H	CBEC
SET 5, L	CBED
SET 6, (HL)	CBF6
SET 6, (IX + d)	DDCBdF6
SET 6, (IY + d)	FDCBdF6
SET 6, A	CBF7
SET 6, B	CBF0
SET 6, C	CBF1
SET 6, D	CBF2
SET 6, E	CBF3
SET 6, H	CBF4
SET 6, L	CBF5
SET 7, (HL)	CBFE
SET 7, (IX + d)	DDCBdFE
SET 7, (IY + d)	FDCBdFE
SET 7, A	CBFF
SET 7, B	CBF8
SET 7, C	CBF9
SET 7, D	CBFA
SET 7, E	CBFB
SET 7, H	CBFC
SET 7, L	CBFD
SLA (HL)	CB26
SLA (IX + d)	DDCBd26
SLA (IY + d)	FDCBd26
SLA A	CB27
SLA B	CB20
SLA C	CB21
SLA D	CB22
SLA E	CB23
SLA H	CB24
SLA L	CB25

SRA (HL)	CB2E	SUB (IX + d)	DD96d
SRA (IX + d)	DDCBd2E	SUB (IY + d)	FD96d
SRA (IY + d)	FDCBd2E	SUB A	97
SRA A	CB2F	SUB B	90
SRA B	CB28	SUB C	91
SRA C	CB29	SUB D	92
SRA D	CB2A	SUB E	93
SRA E	CB2B	SUB H	94
SRA H	CB2C	SUB L	95
SRA L	CB2D	SUB n	D6n
SRL (HL)	CB3E	XOR (HL)	AE
SRL (IX + d)	DDCBd3E	XOR (IX + d)	DDAEd
SRL (IY + d)	FDCBd3E	XOR (IY + d)	FDAEd
SRL A	CB3F	XOR A	AF
SRL B	CB38	XOR B	A8
SRL C	CB39	XOR C	A9
SRL D	CB3A	XOR D	AA
SRL E	CB3B	XOR E	AB
SRL H	CB3C	XOR H	AC
SRL L	CB3D	XOR L	AD
SUB (HL)	96	XOR n	EEn

Tabela de conversão HEX/Decimal

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

Tabela de cálculo do salto relativo

Este quadro serve para calcular os saltos relativos. Por exemplo, o HEX para o salto relativo + 73 é 18 49; para -73 é 18 B7.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
1	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Tabela do ficheiro visual

A tabela seguinte contém os endereços iniciais dos caracteres no princípio e no fim de cada uma das 24 linhas no *écran*.

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

Tabela do ficheiro de atributos

Esta tabela mostra os endereços da memória dos atributos que correspondem ao primeiro e último carácter em cada uma das 24 linhas.

LINHA	INÍCIO	FIM
0	22528	22559
1	22560	22591
2	22592	22623
3	22624	22655
4	22656	22687
5	22688	22719
6	22720	22751
7	22752	22783
8	22784	22815
9	22816	22847
10	22848	22879
11	22880	22911
12	22912	22943
13	22944	22975
14	22976	23007
15	23008	23039
16	23040	23071
17	23072	23103
18	23104	23135
19	23136	23167
20	23168	23199
21	23200	23231
22	23232	23263
23	23264	23295

O sistema de numeração binária

No capítulo em que se tratou do armazenamento dos números terá aprendido que cada localização pode armazenar um byte, o qual tem um valor entre 0 e 255. Os computadores só podem lidar com dois números, o zero e o um, e todos os números estão armazenados como uma série de uns e zeros.

Cada byte divide-se em oito bits com cada bit representando um número específico do seguinte modo:

128 64 32 16 8 4 2 1

Cada bit pode ser representado por um 1 ou um 0. Dependendo quais os bits que estão ligados (representados por um 1) ou desligados (representados por um 0), qualquer número compreendido entre 0 e 255 pode ser armazenado.

O exemplo seguinte mostra quais os bits que devem estar ligados e quais os que devem estar desligados de modo a representar o número 55.

128 64 32 16 8 4 2 1
0 0 1 1 0 1 1 1

Se se proceder à soma dos números acima dos bits que estão ligados obter-se-á o número 55. Se nunca tomou conhecimento do sistema binário pode ficar um pouco confuso. Felizmente não precisa de compreender para escrever os programas em linguagem máquina. É suficiente saber que se realizar uma instrução BASIC tal como POKE 30000,55, o 55 está de facto armazenado como 8 bits na localização 30000. E, de modo semelhante, se realizar uma instrução em linguagem máquina tal como LD 30000,A o valor de A está armazenado no local 30000 do mesmo modo em 8 bits.

Respostas

- 1) A componente superior de 45621 é 178 e a inferior é 53.
- 2) Se a componente inferior de um número é 31 e a sua componente superior 64, trata-se do número 16415.
- 3) O equivalente de E3h no sistema decimal é 227.
- 4) Se FBh é a componente superior de um número e CBh a sua componente inferior, então o número é 64459.
- 5) O opcode 18E5 significa salto relativo menos 27 bytes.
- 6) O opcode para salto relativo mais 98 bytes é 18 62.
- 7) Se a posição 30000 contém 5d e a 30001 contém 15d, após a instrução LD HL,(30000), HL conterá o valor 3845.
- 8) Se HL contém 35621, após a instrução LD(30 000),HL, a posição 30000 conterá 37 e a 30001 terá 139.
- 9) O código para papel turquesa e tinta verde é 44.
- 10) O código para papel branco, tinta turquesa e flamejante é 189.
- 11) Para produzir uma nota de 1200 Hz durante três segundos, HL deve ser 334 e DE 3600.