4

Este livro estimula o desenvolvimento do principiante através de exemplos simples e de fácil aprendizagem e também apresenta referências utilíssimas para o trabalho do perito. Está dividido em duas partes:

Secção A:
Três capítulos explicando tudo o que é necessário saber acerca do código máquina do *ZX Spectrum* – como estão organizados os registos, a memória, a pilha, o ficheiro de imagem.

Secção B:
40 rotinas, entre as quais as que se destinam a rotação de *écran,* manipulação de imagem, trocas de caracteres, compactação de programas, manipulação de programas, etc., e ainda rotinas utilitárias, como as de procura de comprimento de programa, áreas livres da RAM, procurar e substituir, renumeração de linhas (incluindo GOSUBs, GOTOs, RUN, etc.) e muitas outras.

Todos os programas deste livro foram verificados e testados pelo Gabinete Verbo de Informática.

BIBLIOTECA VERBO DE INFORMÁTICA

AS MELHORES ROTINAS

Verbo

AS MELHORES ROTINAS

PARA O ZX SPECTRUM

J. HARDMAN / A. HEWSON

VERBO
BIBLIOTECA DE INFORMÁTICA

JOHN HARDMAN
ANDREW HEWSON

As Melhores Rotinas

40 Rotinas em Código Máquina para o ZX Spectrum

Verbo

Título original: *40 Best Machine Code Routines for the ZX Spectrum*
Tradução de Eng. Paulo Alexandre Monteiro
Capa de Luís Anglin
© *Copyright* by Hewson Consultants 1984
Direitos reservados para a Língua Portuguesa
Editorial Verbo. Lisboa / São Paulo
N.º Ed.-1596
Composto por Fotocompográfica
Impresso por Empresa Litográfica do Sul
em Março de 1985
Depósito Legal n.º 7933/85

Índice

Secção A

Pág.

1. Introdução 11
Porquê usar código máquina? 13
Como aprender código máquina? 15

2. Estrutura interna do ZX Spectrum 16
O mapa de memória 16
PEEK e POKE 18
O ficheiro de *écran* 20
Atributos 22
O *buffer* de impressão 23
Zona de programas BASIC 24
Formato numérico de cinco *bytes* 26
Zona de variáveis 27
As rotinas da ROM 28

3. A linguagem máquina do Z80A 29
Bits 30
Bytes 31
Endereços 31
Registos do *Z80A* 32
Acumulador 33
O registo de *flags* 33
Registos de contagem 34
Registos de endereço 35
Registos de índex 35
O ponteiro de pilha 36
O contador de programa 36
Registos alternativos 37
Acerca do conjunto de instruções 37
Glossário das instruções de código máquina 39
Não operar 39
Carregar 39
Push e *Pop* 39
Troca 40
Adição e subtracção a 8 *bits* 40
And, or e *xor* («e» lógico, «ou» lógico e «ou» exclusivo lógico) a 8 *bits* 40
Comparar 40
Incremento e decremento a 8 *bits* 40

Incremento e decremento a 16 *bits* 40
Soma e subtracção a 16 *bits* 40
Salto, chamada e retorno de sub-rotinas 40
Instruções a nível de *bit* 41
Rotação esquerda de um dígito 42
Rotação direita de um dígito 42
Operações com o acumulador 42
Restart 43
Manipulação de blocos 43

Secção B

4. Introdução 47
Carregador de código máquina 48

5. Rotinas de rotação de écran 53
Rotação esquerda de atributos 53
Rotação direita de atributos 54
Rotação superior de atributos 56
Rotação inferior de atributos 57
Rotação à esquerda de um carácter 58
Rotação à direita de um carácter 60
Rotação superior de um carácter 61
Rotação inferior de um carácter 63
Rotação à esquerda de um *pixel* 65
Rotação à direita de um *pixel* 66
Rotação superior de um *pixel* 68
Rotação inferior de um *pixel* 70

6. Rotinas de manipulação de imagem 73
Mistura de imagens 73
Inversão vídeo do *écran* 75
Inversão vertical de caracteres 77
Inversão horizontal de caracteres 77
Rotação horária de caracteres 78
Mudança de atributos 81
Troca de atributos 82
Preenchimento de zonas 84
Tabelas de desenho 90
Cópia e ampliação do *écran* 95

7. Rotinas de manipulação de programas 104
Eliminação de blocos de programa 104
Troca de carácter 106
Eliminação de REMs 107
Criação de REMs 111
Compactação de programas 115
Carregamento de código máquina em instruções DATA .. 116
Conversão minúsculas/maiúsculas 122

8. Rotinas utilitárias 124
Renumeração 124
Memória disponível 134
Comprimento de um programa 135
Endereço de uma linha 135
Cópia de um bloco de memória 137
Posicionar variáveis a zero 139
Listar variáveis 142
Procurar e listar 145
Procurar e substituir 150
Procurar na ROM 153
Instr$ 156

Apêndice A 162

À minha família, pela sua paciência, mas especialmente a Debbie, John e Mairi, pelo seu infindável encorajamento.

J. H.

Obrigado a Janet, Gordon e Louise.

A. D. H.

Secção A

1. Introdução

O objectivo deste volume é fornecer, quer ao iniciado quer ao utilizador experiente de computadores, uma fonte de referências com base numa série de úteis, interessantes ou divertidas rotinas em código máquina para o *ZX Spectrum*. Para isso, dividimos o livro em duas secções. A Secção A descreve as características do *Spectrum* que têm interesse para o programador em código máquina — o significado de uma rotina em código máquina, as características internas e rotinas importantes do *Spectrum* e a estrutura do código máquina em si.

A Secção B apresenta as rotinas propriamente ditas. São apresentadas em formato normalizado explicado pormenorizadamente no início dessa secção. As rotinas funcionam autonomamente, para que possam ser carregadas individualmente, sem referências a quaisquer outras.

Não é necessário compreender o funcionamento de uma rotina para a utilizar, uma vez que cada rotina pode ser carregada utilizando o acessível carregador de C/M listado no início da Secção B. Desta forma, ao utilizador que pretenda testar, por exemplo, a rotina de listar variáveis, basta-lhe simplesmente passar à página respectiva, introduzir na máquina o carregador de C/M, executar (RUN) este programa e passar à introdução dos números decimais listados na coluna de cabeçalho «Números a introduzir». Uma vez inseridos todos os números, deve comparar o valor do teste-soma apresentado pelo carregador de C/M com o valor apresentado com a rotina. Se estes valores forem iguais, podemos ter a certeza de que a introdução dos números foi correcta (a menos que existam dois ou mais erros que se anulem mutuamente), estando a rotina pronta a utilizar.

Quem não estiver suficientemente confiante para testar uma rotina longa como a de listar variáveis, mas desejar, mesmo assim, iniciar-se rapidamente no código máquina, pode tentar uma rotina mais curta. Desta forma, se se perder ou cometer demasiados erros, não perderá muito tempo. A rotina ideal para este efeito é a de rotação inferior de atributos. Uma vez mais, trata-se apenas de introduzir e executar o carregador de C/M listado no início da Secção B e copiar os números dados na coluna de cabeçalho «Números a introduzir». É necessário não esquecer a verificação final do teste-soma.

Para quem não for importante o uso imediato das rotinas em código máquina, é preferível fazer a leitura integral do livro. Até ao final deste capítulo, faz-se uma introdução às ideias de base e prossegue-se com uma explicação mais minuciosa do conteúdo da obra. Aconselha-se aos iniciados uma leitura cuidadosa desta informação, que tem menos interesse para os utilizadores mais experientes.

O microprocessador *Z80A*, que comanda o *ZX Spectrum*, não entende directamente termos em BASIC como PRINT, IF, TAB, etc. Em vez disso, obedece a uma linguagem específica, chamada «código máquina». As instruções na ROM *Sinclair*, que dá ao *Spectrum* a sua «personalidade», estão escritas nesta linguagem e consistem num grande número de rotinas para introduzir, listar, interpretar e executar o tipo particular de BASIC que o *Spectrum* utiliza. Com efeito, as rotinas são grupos de instruções do tipo «que fazer se». Elas indicam ao *Z80A*, por exemplo, «que fazer se» o próximo comando BASIC for a palavra PRINT, e depois «que fazer se» o item seguinte for um nome de variável; e em seguida «que fazer se» o próximo item for uma vírgula, etc.

O código máquina consiste numa sequência de números inteiros positivos, todos inferiores a 256, e comanda a acção do *Z80A* através do posicionamento de oito «interruptores» lógicos, de acordo com o formato em binário equivalente ao número. O equivalente em binário a 237 é, por exemplo, 11101101, de forma que, quando surge este valor, os oito interruptores são posicionados em *on, on, on, off, on, on, off, on*, respectivamente.

Apesar de a máquina só obedecer à versão binária do número, não há necessidade de o utilizador se preocupar com a instrução neste formato. Como estamos mais familiarizados com o uso de números decimais, são estes números que aceita o carregador de C/M da Secção B. Contudo, mesmo com números decimais, uma longa sequência é difícil de interpretar, pelo que estes são normalmente convertidos de novo numa linguagem especial, designada linguagem *assembly* (*assembler* na terminologia informática corrente) que tem um aspecto bizarro, mas não é muito difícil de utilizar na prática. Cada rotina da Secção B é listada quer em linguagem *assembly* quer através dos números decimais.

A linguagem *assembly* ou *assembler* é assim designada devido ao facto de se poder utilizar um programa especial, chamado «assemblador», de forma a juntar *(assemble)* várias instruções em código máquina e formar um programa. Os assembladores são programas complexos, uma vez que o código máquina é muito extenso e também eles são normalmente escritos em código máquina. Existe uma grande quantidade desses assembladores já disponíveis para o *Spectrum* e, apesar de as rotinas deste livro poderem ser introduzidas utilizando um assemblador, convém realçar que o carregador de C/M serve perfeitamente para este fim.

Um só número é suficiente para especificar as instruções mais simples do *Z80A*. Por exemplo, a instrução para copiar o conteúdo do registo c para o registo d é 81, em decimal (o significado da palavra «registo» será explicado com mais minúcia no capítulo 3. Por agora é suficiente pensarmos em c e d como variáveis em BASIC). Para estas instruções, existe uma correspondência de um para um entre o número decimal e a versão em *assembler*, de forma que o decimal 81, por exemplo, é escrito em *assembler* na forma: ld d,c. «ld» significa «carregar» *(load)*. Muitas das instruções do *assembler* consistem em

abreviaturas simples deste tipo, pelo que é usual designá-las por «mnemónicas».

Instruções mais complexas necessitam de dois, três ou mesmo quatro número decimais para serem completamente especificadas, sendo mesmo assim utilizada uma só mnemónica *assembler* na sua representação. A tabela 1.1 apresenta alguns exemplos destes números, respectivas mnemónicas e uma breve explicação.

Ref	*Decimal*	*Assembler*	*Comentário*
(a)	81	ld d,c	Carregar d com o conteúdo de c
(b)	14 27	ld c,27	Colocar o número 27 em c
(c)	14 13	ld c,13	Colocar o número 13 em c
(d)	33 27 52	ld hl,13339	Colocar 13339 no par de registos hl. Note-se que 27+256*52=13339; 27 é colocado em l; 52 é colocado em h
(e)	221 33 27 52	ld ix,133339	Colocar 13339 no registo ix

Tabela 1.1 *Alguns exemplos de instruções em código máquina para o* Z80A.

A linha (a) da tabela é o exemplo ld d,c acima estudado. As linhas (b) e (c) mostram como um número inteiro positivo menor que 255 pode ser carregado num registo, utilizando-se dois números consecutivos — o primeiro especifica a acção a executar e o segundo indica o número a ser carregado. A linha (d) mostra como carregar um número positivo grande em dois registos, h e l, conjuntamente. Desta vez são o segundo e terceiros números que especificam a quantidade a ser carregada. O último exemplo, na linha (e), ilustra a utilização de uma codificação com quatro números, para carregar o registo ix com um número positivo grande. De notar que três dos quatro números utilizados surgem também na linha (d). Com efeito, o primeiro número apenas especifica o registo ix em vez do par hl.

A estrutura da linguagem máquina será explicada mais minuciosamente no capítulo 3, sendo apresentada no Apêndice A uma lista completa das mnemónicas *assembler*. Uma questão mais importante a responder, de momento é o texto seguinte.

PORQUÊ USAR CÓDIGO MÁQUINA?

Quaisquer que sejam as linguagens de programação ou os computadores, surgem sempre, aparentemente, tarefas que o utilizador pretende que a

máquina execute e que não podem ser convenientemente escritas na linguagem disponível ou que, quando são escritas nessa linguagem, resultam de execução demasiado lenta. O *ZX Spectrum* não constitui excepção a esta regra.

Considere-se, por exemplo, o problema de salvaguardar a totalidade do conteúdo de um *écran* no topo da RAM ou de o recuperar, talvez com o objectivo de criar um efeito de animação, através da rápida mudança entre várias imagens. O ficheiro de *écran* e os atributos ocupam um total de 6912 *bytes*, sendo assim necessário deslocar o cimo da pilha *(ramtop)* para 32767−6912=25855 [na máquina de 16 k, para obter o espaço necessário para a cópia da imagem para fora da área reservada ao BASIC (65535−6912=58623) na versão de 48 k]. O seguinte pequeno programa em BASIC faz a salvaguarda de imagem pretendida, mas demora bastante tempo a ser executado — cerca de 70 segundos:

```
10 FOR i = 0 TO 6911
20 POKE 25856 + i, PEEK (16384 + i)
30 NEXT i
```

A razão de ser desta demora é o facto de o *Spectrum* passar a maior parte do seu tempo a descodificar os comandos, antes de os executar. É também gasto algum tempo na conversão de números do formato inteiro a dois *bytes*, que o *Z80A* entende, para o formato decimal de 5 *bytes*, utilizado pelo contador de ciclos, e também na execução de operações aritméticas neste último formato. São estes os passos seguidos:

1) Somar i a 16384.
2) Converter o resultado para o formato a dois *bytes*.
3) Retirar o conteúdo do endereço a que se faz PEEK.
4) Somar i a 25856.
5) Converter o resultado para o formado a dois *bytes*.
6) Armazenar o valor obtido no endereço a que se faz POKE.
7) Somar um ao valor de i e guardar o resultado.
8) Subtrair i de 6911. Se o resultado for positivo ou nulo, voltar a 1).

De cada vez que o ciclo é repetido, o *Spectrum* tem que descodificar de novo cada um dos comandos, uma vez que já não se recorda das anteriores operações. É fácil verificar que o computador gasta cerca de 99% do tempo de execução de uma tarefa com a sua preparação. Não constitui, desta forma, surpresa o facto de observarmos que uma rotina em código máquina para salvaguardar o conteúdo do *écran* se executa mais ou menos instantaneamente. Na Secção B dá-se uma rotina demonstrativa deste facto.

COMO APRENDER CÓDIGO MÁQUINA

A linguagem máquina do microprocessador *Z80A* é muito complexa e a compreensão de todas as suas potencialidades requer um bom livro de referência, muito raciocínio e bastante prática. Existem no mercado alguns destes livros. A referência normalmente utilizada é *How to Program the Z80* de Rodney Zacks. Esta obra contém bastante informação sobre a organização *hardware* do microprocessador e uma descrição minuciosa do conjunto de instruções, mas o iniciado decerto o achará demasiado pesado, uma vez que contém mais de 600 páginas de informação.

Mais legível é o *Z80 and 8080 Assembley Language Programming*, de Kate Spracklen. Começa a um nível elementar, cobrindo, no entanto, os aspectos mais importantes do *software*, mas ignorando quase por completo o *hardware*.

A presente obra pretende não ser apenas uma introdução ao código máquina para iniciados, mas tornar-se também útil aos utilizadores com algumas experiência. Dá-se ao leitor um forte incentivo para a aprendizagem de código máquina, através da apresentação de rotinas que este pode incorporar nos seus próprios programas em BASIC ou código máquina, com ou sem adaptações.

A estrutura da maior parte das rotinas está intimamente dependente da estrutura do *ZX Spectrum*, pelo que o capítulo seguinte cobre este tópico com algum pormenor. Apresenta, por exemplo, o formato do ficheiro de *écran*, as zonas de programa e de variáveis, explica a forma de armazenamento das linhas de programa BASIC e introduz a aritmética de vírgula flutuante a cinco *bytes*.

O terceiro capítulo explica mais minuciosamente a linguagem máquina do *Z80*, descrevendo alguns dos items que, mais tarde, se consideram adquiridos. Contém um glossário do conjunto de instruções que cobre os factos mais importantes, sem pretender fazer, no entanto, uma cobertura tão pormenorizada como a do livro de Zacks.

2. A estrutura interna do «ZX Spectrum»

Um computador é uma máquina capaz de armazenar uma sequência de instruções e executá-las de seguida. Necessita, assim, de uma memória para armazenar essas instruções. O *ZX Spectrum* tem dois tipos distintos de memória. O primeiro destes tipos é a memória só de leitura (ROM: *Read Only Memory*) que contém um conjunto fixo de instruções implementadas na máquina pelo fabricante. O segundo tipo é a memória de acesso aleatório (RAM: *Random Access Memory*).

A memória de acesso aleatório é o bloco-notas do *Spectrum*. Durante a execução de uma tarefa, o computador está continuamente a verificar o conteúdo da RAM («lendo» a memória) e a alterá-lo («escrevendo» na memória). O *Spectrum* não utiliza o seu bloco-notas ao acaso. Usa zonas diferentes da RAM para armazenar tipos diferentes de informação. Por exemplo, um programa em BASIC introduzido pelo utilizador, é armazenado numa determinada zona da RAM, ao passo que as variáveis utilizadas pelo programa são armazenadas noutra zona. O tamanho do bloco-notas é limitado, pelo que a máquina é obrigada a reservar o espaço apenas suficiente para a informação de que necessita. Assim, o espaço de sobra é arrumado num só bloco, de forma a que se o utilizador quer, por exemplo, acrescentar uma linha ao seu programa, a informação na RAM é redistribuída de forma a ocupar algum do espaço livre e acomodar essa linha.

Grande parte deste capítulo é dedicado à explicação minuciosa da forma como o *Spectrum* organiza a RAM, uma vez que muitas das rotinas da Secção B se destinam à manipulação desta memória. Assim, se o leitor pretende compreender a construção das rotinas, em vez de pura e simplesmente as utilizar, deve apreender o conteúdo deste capítulo, que descreve o ficheiro de *écran*, os atributos, o *buffer* de impressão, as variáveis do sistema, a zona de programas e a zona de variáveis. A parte final descreve as rotinas da ROM que são referenciadas ao longo da Secção B.

O MAPA DA MEMÓRIA

Existem 16384 posições de memória na RAM do *Spectrum* de 16 k (a versão de 48 k contém mais 32768 posições, num total de 49152). Cada uma destas posições pode conter um número inteiro positivo entre 0 e 255, inclusive, e é identificada pelo seu endereço, que é também um número inteiro positivo.

Os endereços de 0 a 16383 estão atribuídos à porção fixa da memória, a ROM, pelo que o primeiro endereço atribuído à RAM é 16384. A tabela 2.1

contém o mapa de memória do *Spectrum*, que mostra a forma como a RAM é utilizada, a partir do endereço 16384. O ficheiro de *écran*, por exemplo, que contém a informação que está constantemente a ser apresentada no *écran*, ocupa as posições 16384 a 22527. Os atributos, que determinam a cor, brilho, etc., da imagem apresentada, vêm logo de seguida, nas posições 22528 a 23295.

Os primeiros cinco endereços iniciais na coluna 1 da tabela 2.1 são todos valores fixos, pois o ficheiro de *écran*, os atributos, etc., ocupam sempre uma porção fixa do espaço de memória. A quinta área está atribuída aos mapas de *microdrives*. Se ligarmos um *microdrive* ao *Spectrum*, esta área passa a conter informações sobre a forma como os dados estão distribuídos no *microdrive*. Caso não haja nenhum *microdrive* ligado, esta zona torna-se desnecessária, pelo que a sexta zona, a informação de canais, é posicionada imediatamente após a quarta, as variáveis do sistema, de acordo com a prática do *Spectrum* de poupar espaço onde possível. Desta forma, o endereço inicial da zona de informação de canais, e de todas as zonas subsequentes, não é fixo, podendo ser deslocado para cima e para baixo na RAM.

O *Spectrum* mantém-se ao corrente do endereço inicial de todas estas zonas, armazenando o valor corrente de cada endereço dentro da zona de variáveis do sistema. A zona de variáveis do sistema está situada antes do mapa de *microdrives*, nas posições de endereços 23552 a 23733, inclusive, pelo que fica fora de questão o seu deslocamento na RAM. Os endereços, dentro desta zona, que contêm os endereços iniciais de todas as zonas deslocáveis, são listados na coluna dois da tabela 2.1. O endereço da zona de programas BASIC, por exemplo, está contido na posição 23635 e, portanto, dentro da zona de variáveis do sistema.

Endereço inicial ou variável do sistema	*Localização da variável do sistema*	*Conteúdo da memória*
16384	—	Ficheiro de *écran*
22528	—	Atributos
23296	—	*Buffer* de impressão
23552	—	Variáveis do sistema
23734	—	Mapa de *microdrives*
CHANS	23631	Informação de canais
PROG	23635	Programas BASIC
VARS	23627	Variáveis
E_LINE	23641	Comando/linha em correcção
WORKSP	23649	Dados em introdução
STKBOT	23651	Pilha *(stack)* da calculadora
STKEND	23653	Espaço livre

sp	—	Pilha/máquina e pilha/ /GOSUB
RAMTOP	23730	Rotinas em código máquina do utilizador
UDG	23675	Formas gráficas definíveis pelo utilizador
PRAMT	23732	Fim da RAM

Tabela 2.1 *Mapa de memória. O ponteiro de pilha, sp, não está contido na RAM, mas, sim, no registo sp do microprocessador* Z80A.

Seria pouco esclarecedor se nos referíssemos às variáveis de sistema apenas pelo respectivo endereço, pelo que foi atribuído a cada uma delas um nome — PROG, por exemplo, para a posição que contém o endereço da zona de programas BASIC. Estes nomes existem apenas por conveniência de utilização, uma vez que não são identificados pelo *Spectrum*. Assim, se introduzirmos o comando:

PRINT PROG

o *Spectrum* devolve-nos uma mensagem de erro «2 Variable not found» (Variável desconhecida), a menos que alguma variável de nome PROG tenha sido, por coincidência, definida por um programa ou pelo utilizador. O valor desta variável não teria, obviamente, nada a ver com o valor da variável do sistema PROG.

PEEK E POKE

O mapa de memória é a chave para a compreensão do uso da RAM pelo *Spectrum*, mas as chaves para a exploração da RAM são os termos PEEK e POKE do BASIC, que permitem ao utilizador verificar e alterar o conteúdo de cada posição de memória.

PEEK é uma função da forma:

PEEK endereço

O endereço pode ser qualquer número inteiro positivo entre 0 e 65535 ou ainda uma expressão aritmética que, quando calculada, dê como resultado um número nesse domínio. É importante a colocação entre parênteses das expressões aritméticas, uma vez que:

PEEK 16384+2

é interpretado como a soma de 2 com o resultado de:

PEEK 16384

ao passo que:

PEEK (16384+2)

é interpretado como:

PEEK 16386

O resultado que se obtém da função PEEK é o número actualmente contido no endereço em questão, que será sempre um número inteiro positivo entre 0 e 255, inclusive. Explicámos acima que a variável do sistema PROG está contida no endereço 23635, mas o valor de PROG, representando um endereço na RAM, é sempre muito maior que 255, pelo que são necessários dois endereços adjacentes, 23635 e 23636, para o armazenar. Este valor pode ser obtido através do comando:

PRINT "PROG="; PEEK 23635+256*PEEK 23636

Todos os endereços são guardados desta forma, podendo assim ser obtidos por um comando do tipo:

PRINT PEEK *primeiro endereço*+256*PEEK *endereço seguinte*

Se um *Spectrum* estiver, por exemplo, a ser utilizado sem nenhum *microdrive* ligado, não existirá a zona do mapa de *microdrives*, e a informação de canais seguir-se-á imediatamente à zona de variáveis de sistema. Assim, o valor da variável de sistema CHANS será igual ao endereço inicial do mapa de *microdrives*, se este existisse, isto é, 23734. CHANS está nas posições 23631 e 23632, pelo que o comando:

PRINT PEEK 23631+256* PEEK 23632

dará como resultado o valor 23734.

A função PEEK pode ser utilizada para verificar o conteúdo de quaisquer posições de memória, inclusive das instruções fixas da ROM. Representa, assim, uma ferramenta muito útil. A utilização de PEEK, seja em que endereço for, nunca provoca falha *(crash)* ou alteração nos programas ou variáveis. O seu resultado, ocasionalmente, não se encontra actualizado, uma vez que o conteúdo da posição de memória a que se faz PEEK pode ter sido alterado durante, ou imediatamente após, a execução da instrução. Por exemplo, se fizermos PEEK aos endereços que estão atribuídos ao canto superior esquerdo do *écran*, o resultado é apresentado nessa mesma zona do *écran*, pelo que a informação já está desactualizada quando o utilizador a vê.

O comando POKE é algo mais perigoso que a função PEEK, uma vez que o utilizador pode interferir com o funcionamento do *Spectrum*, ao invocá-lo. É possível, com este comando, forçar o aparecimento na RAM de valores disparatados, provocando uma falha da máquina ou a sua paragem e o aparecimento de uma mensagem de erro. O formato deste comando é:

POKE *endereço, número*

Uma vez mais o endereço deve ser um número inteiro positivo entre 0 e 65535, inclusive, ou uma expressão aritmética cujo valor represente um número nesse domínio. Neste caso não é essencial o uso de parênteses englobando a expressão, uma vez que POKE é um comando, não uma função, não podendo assim ser avaliada como um todo. O número a que se vai aplicar o POKE, ou seja, o número que vai ser introduzido no endereço acima referido, deve ser um valor entre 0 e 255, inclusive.

O *Spectrum* aceitará e tentará executar um comando POKE que pretenda carregar um número num endereço da ROM (isto é, num endereço entre 0 e 16383), mas esse número nunca chegará ao seu destino. Este facto pode ser demonstrado pela execução do seguinte programa:

```
10 PRINT PEEK 0
20 POKE 0,92
30 PRINT PEEK 0
```

As linhas 10 e 30 provocarão ambas o aparecimento no *écran* do mesmo número (243), que é o conteúdo da posição de endereço 0. A linha 20 não tem, portanto, qualquer efeito.

O FICHEIRO DE «ÉCRAN»

O formato normal de apresentação de imagens no *écran* consiste em 24 linhas, contendo cada uma 32 caracteres. Vimos, no entanto, que o ficheiro de *écran (display file)* ocupa as posições de endereços 16384 a 22527, ou seja, um total de 6144 posições, pelo que o número de posições ocupado por cada carácter é:

6144/(24*32)=8

A forma mais simples de ficarmos com uma ideia geral da organização em memória da informação que cria as imagens é criar uma imagem no *écran*, gravar (*save*) essa imagem, limpar o *écran* e carregar *(load)* de novo essa imagem a partir do gravador. O programa P2.1 faz *save* e *load* ao *écran*, tal como acima se refere, utilizando o carácter gráfico 5 na criação da imagem inicial.

```
 100 FOR i=0 TO 703
 110 PRINT "▌";
 120 NEXT i
 130 SAVE "Imagem"SCREEN$
 140 CLS
 150 INPUT "Rebobinar e ler a ca
ssette e premir qualquer tecla";
z$
 160 LOAD "Imagem"SCREEN$
```

Programa P2.1 *Programa que guarda um* écran *em* cassete (save), *limpa esse* écran *e torna a carregá-lo.*

Observando a imagem a ser carregada a partir do gravador, torna-se claro que o *écran* é dividido em três secções, cada uma delas com oito linhas de caracteres. Cada linha de caracteres é ainda subdividida em oito linhas de *pixels*. Surpreendentemente, o *Spectrum*, ao fazer LOAD SCREEN$, não carrega em primeiro lugar as oito linhas de *pixels* que formam a primeira linha de caracteres, seguidas das oito linhas de *pixels* que formam a segunda linha de caracteres, etc. Em vez disso, são primeiro carregadas as oito linhas de *pixels* que constituem o topo das oito primeiras linhas de caracteres, seguindo-se as oito linhas de *pixels* seguintes das mesmas oito linhas de caracteres e assim sucessivamente. A secção superior do *écran*, constituída por oito linhas de caracteres, e as oito linhas finais de caracteres formam o centro e a base do *écran*, respectivamente.

Outra forma de compreender o formato do ficheiro de *écran* é analisar o posicionamento neste dos oito *bytes* que são utilizados para construir o carácter situado no canto superior esquerdo. O primeiro *byte* é o que define a parte superior do carácter, estando situado no início do ficheiro de *écran*, no endereço 16384. Uma rápida experiência demonstra que:

POKE 16384,0

escurece a linha de oito *pixels* que forma o topo do primeiro carácter, enquanto que:

POKE 16384,255

faz com que todos os *pixels* que a constituem fiquem iluminados. Se fizermos POKE com valores entre 0 e 255, obtemos uma linha salpicada.

A segunda linha de oito *pixels*, a contar de cima, do primeiro carácter no *écran* não é formada com base no número contido na posição 16385, pois esta posição é usada pela linha de oito *pixels* que forma o topo do carácter adjacente ao primeiro. Existem 32 caracteres em cada linha e 8 linhas por secção, pelo que a segunda linha de oito *pixels* do primeiro carácter é formada com base no número contido na posição:

16384+32*8=16640

```
  10 REM Rotina para construir
um caracter situado no canto su-
perior esquerdo do ecran"
  20 INPUT "Um caracter e formad
o por oito  bytes situados entre
 0 e 255 in-clusive. Introduzir
o numero do byte (0 a 7) ";n
  30 IF n<0 OR n>7 OR n<>INT n T
HEN BEEP .2,24: GO TO 20
  40 INPUT "Introduzir o conteud
o do byte ";m
  50 IF m<0 OR m>255 OR m<>INT m
 THEN BEEP .2,24: GO TO 40
  60 POKE 16384+8*32*n,m
```

Programa P2.2 *Programa para construir o carácter situado no canto superior esquerdo do* écran.

O mesmo tipo de argumentação é aplicável às restantes seis linhas de oito *pixels*, pelo que a forma do carácter situado no canto superior esquerdo do *écran* é descrita pelo conteúdo dos endereços:

16384, 16440, 16896, 17152, 17408, 17664, 17920, 18176

O programa P2.2 dá ao utilizador a possibilidade de experimentar carregar vários números nestas oito posições.

Cada posição do ficheiro de *écran* controla o estado de oito *pixels* no *écran*. Este controle é exercido pela conversão do número contido numa dada posição para o seu formato binário, sendo os oito *pixels* posicionados de acordo com o padrão zero/um formado pelos oito dígitos binários. Por exemplo, 240, após conversão em binário, fica:

11110000

Assim, uma posição que contenha o número 240 terá quatro dos seus *pixels* iluminados e os restantes apagados.

Resumindo, o ficheiro de *écran* é constituído por 6144 posições de memória, sendo atribuídas oito posições a cada zona de carácter no *écran*. Cada uma destas posições comanda o estado de uma barra horizontal de oito *pixels*. As posições atribuídas a uma dada zona de carácter não são adjacentes, estando em vez disso o *écran* dividido em oito secções, no interior das quais existem 256 posições a separar os *bytes* constituintes de cada zona de carácter.

OS ATRIBUTOS

O conteúdo do ficheiro de *écran* determina apenas quais dos *pixels* estão ou não iluminados. A cor do «fundo» onde o *Spectrum* escreve (PAPER), da «tinta» com que ele escreve (INK) e as condições de maior brilho (BRIGHT) e de piscar (FLASH) são determinadas pelos atributos. A zona de atributos ocupa as posições 22528 a 23295, sendo cada uma destas posições atribuída a cada uma das 768 zonas de carácter do *écran*. Contrariamente ao ficheiro de *écran*, estas posições são atribuídas às zonas de carácter de uma forma óbvia, ou seja, começando pelo canto superior esquerdo e seguindo da esquerda para a direita e de cima para baixo.

Cada posição determina os parâmetros INK e PAPER da zona de carácter correspondente, com base numa das oito cores que estão descritas por cima da primeira linha de teclas do *Spectrum*. Determina, ainda, se essa posição deve ter mais brilho (BRIGHT) e se deve piscar (FLASH). Estes quatro parâmetros são codificados pela seguinte expressão:

Valor do atributo = 128*FLASH + 64*BRIGHT + 8*PAPER + INK

Os parâmetros FLASH e BRIGHT tomam o valor um se a respectiva condição se verificar e zero no caso contrário. PAPER e INK tomam o valor da cor pretendida, de acordo com o que está no teclado (2 para o vermelho, por exemplo). O programa P2.3 é um descodificador de atributos, isto é: dado um

determinado valor de atributo, imprime no *écran* os valores correspondentes dos pârametros PAPER, INK, etc.

```
  10 REM Descodificador de atrib
uto
  20 DATA "Preto  ","Azul   ","V
ermelho ","Magenta","Verde  ",
"Cyan   ","Amarelo","Branco  ","
Brilho ","Piscar"
  30 DIM c$(8,7)
  40 FOR i=1 TO 8
  50 READ c$(i)
  60 NEXT i
 100 REM Descodificador de atrib
uto
 110 INPUT "Introduzir um numero
 entre 0 e  255. Este programa d
escodifica asua interpretacao no
 ficheiro deatributos";n
 120 IF n<0 OR n>255 OR n<>INT n
 THEN BEEP .2,24: GO TO 110
 200 PRINT "A cor da tinta e ";c
$(1+n-8*INT (n/8))
 210 PRINT "A cor do fundo e ";c
$(1+INT (n/8)-8*INT (n/64))
 220 IF INT (n/64)=1 OR INT (n/6
4)=3 THEN PRINT "O caracter e BR
IGHT "
 230 IF n>127 THEN PRINT "O cara
cter seria FLASH "
 300 PRINT AT 6,0;"████████████████
████████████████"
 310 FOR i=22720 TO 22751
 320 POKE i,n
 330 NEXT i
 500 INPUT "Carregue em ENTER pa
ra repetir";z$
 510 CLS
 520 GO TO 110
```

Programa P2.3. *Programa para descodificar um atributo.*

O «BUFFER» DE IMPRESSÃO

Na RAM, utilizam-se as 256 posições que se seguem à zona de atributos para guardar temporariamente uma linha incompleta de caracteres, que serão mais tarde enviados para a impressora. O *buffer* torna-se necessário pelo facto de um programa BASIC poder mandar imprimir (LPRINT) apenas parte de uma linha, terminada por um ponto e vírgula para indicar que a linha está incompleta. Nalguns casos, o comando TAB actua de forma similar. Uma

linha incompleta não pode ser transmitida para a impressora, uma vez que esta tem que imprimir completamente uma linha para poder avançar o papel e passar à seguinte. Assim, qualquer linha incompleta é guardada temporariamente no *buffer* de impressão, até que o programa a complete.

Muitas das rotinas da Secção B utilizam o *buffer* de impressão para passar dados, a partir do BASIC ou do teclado, para o seu programa. O *buffer* é indicado para este fim, pois ocupa posições fixas em memória e é pouco provável que o utilizador precise dele para qualquer outro fim durante a execução de rotinas em código máquina.

ZONA DE PROGRAMAS BASIC

Se estiver ligado ao *Spectrum* algum *microdrive*, o início da zona de programas BASIC determina-se com base na variável do sistema PROG, situada na posição 23635. Sem *microdrives*, esta zona começa sempre na posição 23755. Nesta análise, assume-se que não estão ligados *microdrives*.

O programa P2.4 imprime no *écran* o conteúdo das 18 posições iniciais da zona de programas, como se mostra na figura F2.1. Estas 18 posições são utilizadas pela primeira linha do programa, isto é:

10 REM Peek program

Analisando a figura F2.1 chegamos a algumas importantes conclusões sobre a forma como são codificados os programas.

```
10 REM Peek program
20 FOR i=23755 TO 23772
30 PRINT i,PEEK i
40 NEXT i
```

Programa P2.4. *Programa que imprime no* écran *(PRINT) as primeiras dezoito posições da zona de programas.*

```
23755       0
23756       10
23757       14
23758       0
23759       234
23760       80
23761       101
23762       101
23763       107
23764       32
23765       112
23766       114
23767       111
23768       103
23769       114
23770       97
23771       109
23772       13
```

Figura F2.1. *Forma como é armazenada na zona de programas a linha 10 REM Peek program.*

O número de linha, 10, é guardado nas primeiras duas posições, no formato:

n.º linha = 256*PEEK primeiro endereço+PEEK segundo endereço

Note-se que não se aplica a convenção do *Z80A* de multiplicar o conteúdo do segundo endereço por 256 e somá-lo ao conteúdo do primeiro endereço.

A referida convenção aplica-se às duas posições seguintes, 23757 e 23758, que guardam, conjuntamente, o comprimento da restante parte da linha, a partir da posição 23759. No nosso caso, o número obtido é:

14+256*0=14

Assim, a próxima linha começa na posição:

23759+14=23773

A posição 23759 guarda o número 234, que é o código de carácter de REM. As 12 posições seguintes guardam os códigos de carácter do conjunto de onze letras e um espaço que constituem o título do programa:

Peek program

Finalmente, na posição 23772, temos o código de ENTER, que é o número 13, indicando fim da linha. Na tabela 2.2 apresenta-se um resumo do método de codificação dos programas na zona de programas.

Posições	*Conteúdo*
1 e 2	Número de linha, guardado segundo uma convenção inversa da do *Z80A*
3 e 4	Comprimento da linha, exceptuando os quatro primeiros caracteres
5	Código do comando
Final	Código do carácter ENTER

Tabela 2.2. *Método usado para codificar linhas de programa.*

Nesta descrição foi omitido um item, que é o método de armazenar valores numéricos do programa. Podemos analisá-lo através da substituição da linha 10 por:

10 LET a=1443

no programa P2.4. A figura 2.2 mostra o resultado que se obtém executando o programa obtido.

```
23755   0
23756   10
23757   14
23758   0
23759   241
23760   97
23761   61
23762   49
23763   52
23764   52
23765   51
23766   14
23767   0
23768   0
23769   163
23770   5
23771   0
23772   13
```

Figura 2.2. *Forma como é armazenada na zona de programas a linha: 10 LET A=1443.*

As posições 23755 a 23758 têm o mesmo significado. Seguem-se os códigos de LET, a, =, e dos quatro dígitos que formam o número 1443. O item seguinte, na posição 23766, é 14, o código de carácter que indica que as cinco posições seguintes guardam o número 1443 em formato numérico. A linha termina na posição 23772, como o código de ENTER, tal como anteriormente.

FORMATO NUMÉRICO A CINCO «BYTES»

São utilizadas cinco posições de memória para guardar cada número que apareça num programa BASIC (excepto números de linha, como já vimos). Números inteiros entre −65535 e 65535 são armazenados numa forma semelhante à convenção do *Z80*. Para números neste domínio, as duas primeiras e última posições contêm todas zero, sendo o número guardado na terceira e na quarta, na forma:

número=PEEK terceira posição+256*PEEK quarta posição

16553, por exemplo, é guardado desta forma nas cinco posições referidas:

0 0 169 64 0

uma vez que:

169+256*64=16553

Números não inteiros são guardados com base na representação em vírgula flutuante, com o expoente na primeira posição e a mantissa nas quatro seguintes, ou seja:

número=mantissa+2 ↑ expoente

A primeira posição da mantissa serve também para determinar o sinal do número. Se esta posição contiver um valor entre 0 e 127, o número é positivo, sendo negativo no caso contrário.

O programa P2.5, que a seguir se apresenta, serve para reconstruir um número não inteiro a partir dos valores dos seus cinco componentes.

```
  10 PRINT "Introduzir o expoent
e e os  qua-tro bytes da mantiss
a. Todos de-vem situar-se entre
0 e 255  in-clusive"
  20 INPUT e,a,b,c,d
  30 PRINT  ,,"O expoente e = ";
e
  40 PRINT "A mantissa e = ",a,,
b,,c,,d
  50 PRINT ,,"O numero = ";(2*(a
<128)-1)*2↑(e-160)*(((256*(a+128
*(a<128))+b)*256+c)*256+d)
```

Programa P2.5. *Este programa reconstrói um número não inteiro a partir dos valores dos seus cinco componentes.*

A ZONA DE VARIÁVEIS

O endereço inicial da zona de variáveis está guardado na variável do sistema VARS, situada na posição 23627. Sempre que se define uma nova variável, quer por um programa quer directamente através do teclado, cria-se nesta zona o espaço necessário para a guardar.

Todos os nomes de variável devem começar por uma letra, não sendo distinguidas letras maiúsculas e minúsculas. Estas restrições permitem ao *Spectrum* manipular o código de carácter atribuído à primeira letra de cada variável, de forma a distinguir cada um dos seis tipos de variáveis permitidos pela simples verificação da gama de valores a que pertence o referido código. Todas as variáveis numéricas com nomes de um só carácter, por exemplo, têm códigos entre 97 e 122; 97 para a letra a; 98 para o b; 99 para o c, etc. De forma

similar, aos *arrays* numéricos atribuem-se códigos entre 129 e 153; 129 para o a; 130 para o b; 131 para o c, etc. Os códigos atribuídos a cada tipo de variável são apresentados na tabela 2.3. Apresenta-se, também, o comprimento da zona de variáveis ocupado por cada um dos tipos.

Tipo de variável	*Códigos atribuídos*	*Comprimento na zona de variáveis*
Numéricas (nome de um só carácter)	97 a 122	6
Numéricas (nome com mais de um carácter)	161 a 186	5+comprimento do nome
Array numérico	129 a 154	4+2*número de dimensões +5*número total de elementos
Variável de controle dum ciclo FOR-NEXT	225 a 250	18
String	65 a 90	3+comprimento do *string*
Array de caracteres	193 a 218	4+2*número de dimensões +número total de elementos

Tabela 2.3. *Variáveis, códigos atribuídos e comprimento na zona de variáveis.*

ROTINAS DA ROM

Algumas das rotinas da Secção B usam as seguintes rotinas da ROM:

rst 16
Imprime no *écran* o conteúdo do acumulador.

call 3976
Carrega, no endereço da RAM dado pelo par de registos hl, o carácter contido no acumulador.

call 4210
Apaga um carácter no endereço da RAM apontado pelo par de registos hl.

call 6326
Se o acumulador contiver o código indicativo de número (14), activa a *flag* de zero e incrementa o par de registos hl cinco vezes.

call 6510
Devolve, em hl, o endereço na RAM da linha cujo número foi passado para a rotina em hl.

3. A linguagem máquina do «Z80A»

Este capítulo começa por explicar o significado de alguns dos termos mais importantes, tais como *bit, byte,* endereço e registo, que se terá como adquirido no resto do livro. A quantidade e a diversidade de registos do *Z80A* são, de seguida, examinadas, com referência particular a um pequeno número de instruções, a título de exemplo. Finalmente, é apresentado um resumo do conjunto de instruções.

O aspecto mais difícil que se apresentará ao iniciado em programação em código máquina é, talvez, a quantidade de novos termos e conceitos a fixar. Assim, examinaremos primeiro uma só instrução, como exemplo do que virá a seguir, antes de partir para a parte principal deste capítulo. Consideremos a seguinte instrução composta, que será encontrada em muitas das rotinas da Secção B:

```
ld hl, (23627)
```

Esta instrução deve ser lida da seguinte forma: *carregar (load)* o *par de registos hl* com os *bytes* contidos nos *endereços* 23627 e 23628. Cada uma das palavras em itálico será explicada mais pormenorizadamente ainda neste capítulo.

A instrução é codificada sob a forma de três números decimais — 42, 75, 92. O primeiro número indica:

```
ld hl, (       )
```

ou seja, carregar o par de registos hl com o conteúdo de dois endereços de memória consecutivos. Os endereços em questão são especificados pelos segundo e terceiro números, através do seguinte cálculo:

endereço menor = primeiro número+256*segundo número
endereço maior = endereço menor+1

ou, no nosso caso:

endereço menor = 75+256*92=23627
endereço maior = 23627+1=23628

O termo «carregar» não é mais que uma outra forma de dizer «copiar» e h e l podem ser vistos como duas posições de memória especiais, contidas no *Z80A*, que são utilizadas para guardar números. Assim, o significado global da instrução é copiar o conteúdo de 23627 para o registo l e de 23628 para o registo h. Note-se que, na transferência, o endereço menor (*l*ower) é a origem de destino *l* e o endereço maior (*h*igher) é a origem de destino *h*.

«BITS»

Um *bit* é a unidade fundamental de memória num computador, pois só pode apresentar dois possíveis estados. Estes dois estados podem ser vistos como representando ON ou OFF; VERDADEIRO ou FALSO; SIM ou NÃO; MACHO ou FÊMEA ou quaisquer outros pares de condições logicamente opostas. O mecanismo de funcionamento da memória de um computador não é importante para nós, mas no *Spectrum* o estado de um *bit* é memorizado pelo posicionamento de um microscópico interruptor electrónico em ON ou OFF, conforme se pretenda.

A notação normal refere um destes estados como o estado ZERO e outro como o estado UM. Um *bit* é considerado «activo» se estiver no estado representado como UM e «inactivo» no caso contrário. Esta notação permite-nos falar de uma dada configuração de *bits* em termos do seu equivalente em numeração binária e, através da conversão do número binário em decimal, designar cada configuração por intermédio de um e um só número decimal inteiro e positivo.

Consideremos, por exemplo, 8 *bits*, dos quais estão activos os quatro da direita e inactivos os quatro da esquerda. Esta configuração de *bits* é ilustrada através da tabela 3.1.

Interruptor	Off	Off	Off	Off	On	On	On	On
Posicionamento	Inact	Inact	Inact	Inact	Act	Act	Act	Act
Configuração binária	0	0	0	0	1	1	1	1
N.º de ordem do *bit*	7	6	5	4	3	2	1	0

Tabela 3.1. *Grupo de oito* bits, *com os quatro da esquerda inactivos e os restantes activos.*

A configuração binária pode ser convertida para decimal, se nos lembrarmos que, em binário, a coluna da direita é a das unidades; a seguinte, para a esquerda, é a dos «dois»; a seguinte, também para a esquerda, é a dos «quatro» e assim sucessivamente, duplicando em cada movimento para a esquerda. O equivalente decimal de 00001111 é, desta forma:

0*128+0*64+0*16+1*8+1*4+1*2+1*1=15

uma vez que existem uns nas colunas dos «uns», «dois», «quatros», e «oitos», e zeros nas restantes.

É obviamente incómodo referir os *bits* como o «mais à direita» ou o «segundo a partir da esquerda», e daí a convenção adoptada de numerar os *bits* da direita para a esquerda, começando em zero. Não é apenas por coincidência que, nesta convenção, o número de ordem do *bit* é também o expoente de base 2 de onde se obtém o número da coluna. Isto é:

2 ↑ número de ordem = valor da coluna

O *bit* 3, por exemplo, surge na coluna dos «oitos» e 2↑ 3=8.

«BYTES»

O microprocessador *Z80A*, que está situado no coração do *ZX Spectrum*, opera com oito *bits* de cada vez. (O termo «opera» cobre todas as diferentes tarefas contidas no conjunto de instruções, tais como adição, subtracção, rotação, «E» lógico, etc. O formato destas instruções será explicado em pormenor mais adiante, neste mesmo capítulo.) Assim, apesar de os *bits* serem a unidade fundamental de memória do computador, são normalmente manipulados em grupos de oito. Designamos por *bytes* (que se pronuncia «báites») estes grupos de oito *bits*.

Cada um dos *bytes* da RAM pode ser usado para guardar um número inteiro positivo entre 0 e 255, inclusive, activando-se ou não cada um dos seus oito *bits*, de acordo com o equivalente, em binário, ao número. O *byte* da tabela 3.1, por exemplo, contém o número decimal 15.

Existem 16384 *bytes* na memória só de leitura (ROM) do *ZX Spectrum*, sendo o conteúdo desses *bytes*, juntamente com os dispositivos electrónicos, que dá ao computador as suas características específicas. O conteúdo da ROM é implementado nesta durante o fabrico do *Spectrum*, não podendo ser posteriormente alterado. É por esta razão que esta memória se designa por «memória só de leitura» — o conteúdo pode ser lido, mas não alterado.

O *Spectrum* de 16 k contém ainda 16384 *bytes* de memória de acesso aleatório (RAM). A designação «de acesso aleatório» é algo enganadora, não pretendendo significar que a memória é utilizada ao acaso, mas sim que temos acesso a qualquer *byte* imediatamente, em qualquer altura. Esta propriedade contrasta com a de uma memória de acesso sequencial, como uma fita de *cassette*, para a qual se torna necessária a deslocação ao longo da fita até se atingir o ponto onde está a informação pretendida.

Como princípio, 16384 parece ser um estranho número de *bytes* a usar. De facto, é uma quantidade muito conveniente, pois 2↑14=16384 (isto é, 16384 é igual a 2 multiplicado 14 vezes por si próprio). No universo dos computadores, potências de 2 são «números arredondados», tal como as potências de dez — dezenas, milhares, milhões — são «números arredondados» no nosso dia-a-dia. Um «número arredondado» particularmente importante é 1024, que representa 2 elevado a 10. 1024 é suficientemente semelhante a mil para justificar que se utilize o símbolo k na sua representação. (k é utilizado para representar um milhar no sistema métrico, tal como no quilograma — kg —, quilómetro — km —, etc.) Assim, 1024 é representado por 1 k e 16384, que é 16×1024, é representado por 16 k.

ENDEREÇOS

Um computador deve estar apto a identificar cada uma das suas posições de memória, de forma a poder copiar de e para a posição certa. Assim, atribui-se

um *endereço* único a cada posição. Um endereço é sempre um número inteiro positivo ou zero.

Muitas das instruções do *Z80* são do tipo «copiar o conteúdo do seguinte endereço para este ou aquele registo ou par de registos». A instrução:

ld hl, (23627)

descrita no início deste capítulo, é deste tipo. O endereço que se segue à instrução está guardado em dois *bytes*, pelo que o número máximo de posições a que o processador tem acesso é limitado pelo número de endereços que se podem codificar com dois *bytes*. Este número é o mesmo que o número de diferentes configurações que se podem obter com os 16 *bits* que constituem o endereço de dois *bytes*, ou seja 2↑16=65536.

Um endereço de dois *bytes* é interpretado da seguinte forma:

endereço = primeiro *byte*+256*segundo *byte*

Os dois *bytes* são por vezes designados por *byte* «menor» e «maior», respectivamente. A representação a dois *bytes* de 16384 (o início da RAM no *Spectrum*) é, por exemplo, *byte* menor = 0; *byte* maior = 64, uma vez que:

0+256*64=16384

OS REGISTOS DO «Z80A»

Um computador nunca altera directamente o conteúdo da memória quando da execução de um programa, copiando sempre, previamente, cada posição de memória que pretende manipular para um *registo* e trabalhando depois o conteúdo desse registo. Os registos têm, na linguagem máquina, uma função semelhante à das variáveis no BASIC, uma vez que armazenam números e podem ser também utilizados como elementos controladores de decisões. São, no entanto, diferentes das variáveis BASIC, uma vez que existem em número muito limitado e estão implementados no interior do processador, ao contrário das variáveis, que estão na RAM. Outra das diferenças é o facto de cada registo só poder conter um *byte* ou dois (no caso dos pares de registos).

O *Z80A* é um microprocessador muito poderoso, pois dispõe de muitos registos, podendo assim guardar simultaneamente bastantes números, o que reduz a necessidade de constantes transferências entre o processador e a memória, bastante mais lentas que as operações internas. Muitos destes registos têm algumas particularidades especiais.

O registo acumulador – a

O acumulador é o registo mais importante, pois a maior parte das operações aritméticas, tais como a adição, por exemplo, e as instruções lógicas, como o «OU» lógico, são efectuadas sobre este registo. O acumulador deve o seu nome ao facto de serem nele acumulados os resultados das operações sucessivas.

Algumas das instruções que se referem ao acumulador utilizam como origem dos dados um segundo registo ou um endereço da memória. Por

exemplo, a instrução add a,b indica ao processador que deve somar o conteúdo do registo b ao registo a, colocando o resultado em a.

O registo de «flags» – f

Na maior parte, os registos agrupam-se em pares, no sentido de que algumas instruções se executam conjuntamente sobre o par de registos. O registo f, ou registo de *flags*, faz par com o registo a neste sentido, apesar de a ligação entre eles ser muito fraca, pois limita-se às instruções *push*, *pop* e *exchange*.

O registo f é, funcionalmente, bastante diferente de todos os outros, pois os seus oito *bits* são usados individualmente com a função de *flags* (sinalizadores), registando e controlando a sequência de execução dos programas. Cada *flag* é utilizada como indicador da ocorrência de um de dois eventos logicamente opostos, como por exemplo a *flag* de zero, que indica se o resultado da última adição, subtracção, etc., foi ou não zero. Somente quatro das oito *flags* têm interesse para o utilizador. Na tabela 3.2 apresenta-se um resumo das suas propriedades.

Flag	*Mnemónica flag activa*	*Mnemónica flag inactiva*	*Função*
Sinal	M	P	Activa se o último resultado foi positivo
Zero	Z	NZ	Activa se o último resultado foi zero ou se uma comparação teve êxito
Transporte (*carry*)	C	NC	Activa se o último resultado foi demasiado grande para ser armazenado num só *byte* (ou em dois *bytes* para operações com pares de registos)
Paridade/ /*Overflow*	PE	PO	Paridade activa se o último resultado teve paridade par *Overflow*-activa se o *bit* sete foi alterado por uma operação como resultado de *overflow* com origem noutros *bits*

Tabela 3.2. *As quatro* flags *que controlam a maior parte das operações do* Z80A.

A *flag* de sinal é a mais simples de todas. Por convenção, se um dado *byte* estiver a representar um número com sinal, o seu *bit* sete guarda o sinal, estando activo se o número for negativo e inactivo no caso contrário. A *flag* de sinal reflecte o sinal do último resultado ocorrido.

A *flag* de zero está activa quando o resultado da última operação foi zero. É também utilizada em instruções de comparação, que não passam de subtracções em que o resultado é ignorado.

A *flag* de transporte *(carry)* regista o valor a transportar se o resultado de uma adição for demasiado grande para ser armazenado no registo de resultado ou se ocorrer um transporte de subtracção. Existem também algumas instruções de rotação, nas quais os *bits* do registo a são rodados à esquerda ou à direita, com passagem dos *bits* 7 e 0 «de» ou «para» a *flag* de transporte.

A *flag* de paridade/*overflow* exerce funções de duas *flags* numa só. É usada como *flag* de *overflow* pelas instruções aritméticas, indicando se o *bit* sete foi afectado por um transporte de adição ou subtracção proveniente do *bit* seis. Serve, assim, para verificar se o *bit* de sinal foi indevidamente alterado. As intruções lógicas utilizam esta mesma *flag* como indicativo da paridade do resultado. (A paridade de um número binário é dada pela quantidade de *bits* activos que ele contém. Se for ímpar, o número binário diz-se de paridade ímpar; se for par, diz-se de paridade par.) Esta *flag* está activa após a ocorrência de um resultado de paridade par.

O efeito de algumas instruções depende do estado actual de uma dada *flag*. Por exemplo, a instrução

```
jr z,d
```

faz com que o *Z80* salte por cima das d instruções seguintes, se a *flag* de zero estiver activa. Se esta *flag* estiver inactiva, o processador passa à execução da próxima instrução, como habitualmente faz. O registo de *flags* é, pois, importante, uma vez que permite ao processador decidir se deve ou não seguir para outra parte do programa.

Os registos de contagem – b e c

O registo b e, até certo ponto, o registo c, que faz par com ele, serve para diversos fins, mas a sua finalidade principal é o uso como contador. Já vimos como o fluir de um programa é controlado pelo uso da *flag* de zero, na instrução jr z,d. Existe outra instrução:

```
djnz d
```

que também usa a *flag* de zero para permitir a construção de ciclos de programa em código máquina, utilizando o registo b como contador, de forma análoga aos ciclos FOR/NEXT em BASIC.

Ao encontrar estra instrução, o *Z80A* decrementa o conteúdo de b, ou seja, diminui o seu conteúdo de uma unidade. Se o resultado for nulo, é executada a instrução seguinte da sequência. Se o resultado for não nulo, a rotina executa um salto de instruções. Se o programa usar um valor negativo para d, o salto é para trás e, assumindo que não há outros saltos pelo meio, o processador

executará, de novo, a mesma instrução. Assim, se carregarmos em b um valor conveniente e posicionarmos correctamente o deslocamento d, um bloco de programa é executado sucessivamente um dado número de vezes.

O registo b só pode conter um *byte*, pelo que só admite números entre 0 e 255. Utilizando o mecanismo acima referido, podemos provocar um máximo de 255 passagens pelo mesmo troço de programa.

Não existe qualquer instrução similar para obter mais de 255 passagens num ciclo, mas existem algumas instruções muito poderosas, que utilizam todos os 16 bits do par de registos bc como contador até 65535. Um exemplo possível é a instrução:

```
cpdr
```

Quando o *Z80A* encontra esta instrução, efectua as seguintes operações:

1) decrementa bc de uma unidade;
2) decrementa o conteúdo de hl (hl é um outro par de registos) — ver próximo item;
3) compara o conteúdo do acumulador, a, com o conteúdo da posição de memória cujo endereço está contido em hl.

O processador repete este conjunto de acções até ser detectada uma igualdade entre a e a memória ou até bc=0. Assim, esta instrução varre a memória à procura de um endereço cujo conteúdo seja um dado número.

Os registos de endereço – de e hl

Os registos d e e não têm nenhuma função particular, sendo geralmente utilizados como memória temporária de acesso rápido. Podem ser usados conjuntamente para armazenar endereços de posições de memória com interesse.

A principal função dos registos h e l é guardarem também conjuntamente o endereço de uma dada posição de memória, tendo já sido apresentados exemplos de instruções poderosas que utilizam hl com este fim. A designação de h vem de *high byte* (*byte* maior) e a de l de *low byte* (*byte* menor), sendo um endereço guardado da seguinte forma:

endereço = 256*h+1

donde se obtém um máximo de 65536 endereços possíveis (isto é, de 0 a 65535, inclusive).

Os registos de índex – ix e iy

Os registos ix e iy são ambos de 16 *bits* e apenas podemos usá-los como tal, contrastando com os registos bc, de e hl, que podemos usar quer aos pares, como registos de 16 *bits*, quer individualmente, como registos de 8 *bits*. Utilizamos geralmente estes registos com fins semelhantes aos do par de registos hl, apesar de as instruções que os manipulam terem sempre mais um *byte* de comprimento que as equivalentes para hl.

Por exemplo:

```
add hl,bc
```

é uma instrução de um *byte*, que faz com que o *Z80A* some os conteúdos dos pares de registos hl e de e coloque o resultado em hl. A mesma instrução, mas usando ix,

add ix,bc

é uma instrução de dois *bytes*.

ix e iy têm uma potencialidade extra, em relação a hl, pois se usam conjuntamente com um deslocamento, d. Isto significa que uma instrução com referência a (ix+d) não utiliza a posição de memória cujo endereço está contido em ix. Soma-se o valor de d ao de ix, obtendo-se um novo endereço, o da posição de memória utilizada. É devido a esta propriedade que estes registos têm a designação de «registos de índex».

O ponteiro de «stack» (pilha) – sp

A pilha é uma zona situada no topo da RAM, ou junto a este, que armazena temporariamente o conteúdo de pares de registos. Está construída de forma a «descer» na RAM conforme vai sendo carregada e a «subir», de novo, na RAM quando se lhe retiram dados. A base da pilha é fixa, estando, no *ZX Spectrum*, situada imediatamente abaixo da posição apontada pela variável do sistema RAMTOP. O topo da pilha situa-se abaixo da respectiva base, uma vez que este cresce para baixo e decresce para cima. O endereço da posição actual do topo da pilha está guardado no registo sp.

As transferências de e para a pilha são efectuadas por intermédio das instruções *push* e *pop*. Por exemplo, a instrução:

push hl

faz com que o processador:

1) decremente sp;
2) copie o conteúdo de h para a posição apontada por sp;
3) decremente sp;
4) copie o conteúdo de l para a posição apontada por sp.

A instrução *pop* faz exactamente o contrário. Desta forma, os pares de valores mais recentemente introduzidos na pilha são sempre os primeiros a ser retirados. Obtém-se, assim, um método simples e eficaz de armazenar temporariamente o conteúdo de registos durante, por exemplo, a execução de uma sub-rotina. Desde que haja o cuidado de retirar os registos pela ordem inversa daquela por que foram introduzidos, atingir-se-ão os objectivos desejados.

O contador de programa – pc

O contador de programa, pc, é um registo de 16 *bits* extremamente importante, pois contém o endereço na memória da instrução executada a seguir.

A sequência normal de acontecimentos, na execução de um programa, é a seguinte:

1) cópia do conteúdo da posição apontada por pc para um registo especial situado no interior do processador;

2) se a instrução for de mais de um *byte*, incremento de pc e cópia do conteúdo da posição seguinte para um segundo registo especial;
3) incremento de pc, de forma a que este aponte para a instrução a executar a seguir;
4) execução da instrução que acabou de ser copiada.

Uma instrução de salto, tipo djnz d ou jr z, d, altera o fluir normal dos acontecimentos através da manipulação de pc, durante a execução do passo 4). Note-se que esta manipulação só é efectuada após o incremento de pc, pelo que o valor de um deslocamento, d, deve sempre ser calculado relativamente à posição da instrução seguinte à que contém o deslocamento.

Os registos alternativos – af', bc', de', hl'

O *Z80A* possui duplicados de cada um dos seus registos internos: a, b, c, d, e, f, h, e l. Os duplicados distinguem-se pelo uso de uma pelica na sua representação, sendo, por exemplo, a' o duplicado do registo a. Nenhuma instrução opera directamente com estes duplicados, mas existem instruções de troca, que retiram de circulação dois ou mais registos e trazem os duplicados para uso, em sua substituição.

As instruções de troca são executadas muito rapidamente, bastante mais que as instruções de *push* e *pop*, por exemplo. Os conteúdos não são copiados de um registo para outro. Em vez disso, altera-se um conjunto de interruptores lógicos internos, de forma a que o registo ou os registos «operacionais» passem a «inoperativos» e vice-versa.

ACERCA DO CONJUNTO DE INSTRUÇÕES

Existem mais de 600 elementos no conjunto de instruções do *Z80A*, como se verifica pela listagem no Apêndice A. Devido a só existirem 256 arranjos diferentes de 8 *bits* (pois 2↑8=256), apenas menos de metade das instruções podem ser descritas por um *byte*. As restantes são descritas por dois, ou mesmo três, *bytes*. O primeiro *byte* de uma instrução de dois *bytes* é sempre 203, 221, 237 ou 253 (CB, DD, ED ou FD em hexadecimal). Os dois primeiros *bytes* de uma instrução de três *bytes* são sempre um dos pares 221, 203 ou 253, 203 (DD, CB ou FD, CB em hexadecimal).

Algumas instruções são seguidas de um deslocamento representado por um *byte*, d, ou de um número de um *byte*, n, ou de um número ou endereço de dois *bytes*, nn, aos quais a instrução se refere. Desta forma, uma só instrução pode ocupar um total de quatro *bytes*. Por exemplo, a instrução:

jn nz,d

já nossa conhecida, necessita de um *byte* para descrever a instrução em si (32 em decimal, 20 em hexadecimal) e de um segundo *byte* para representar o deslocamento, d.

Neste capítulo, todas as instruções são referenciadas pelas respectivas mnemónicas em linguagem *assembly (assembler)* ou código de operação (Op Code) na terminologia informática. As mnemónicas são uma forma abreviada de descrever cada instrução e existem apenas para conveniência dos utilizadores. O *Spectrum* não identifica as mnemónicas, a não ser por intermédio de um programa assemblador.

Descrevemos a seguir algumas convenções:

1) Registos de um *byte* são designados pelas respectivas letras, por exemplo b. Pares de registos são designados por ordem alfabética, por exemplo bc.
2) Um deslocamento, d, é tomado como positivo se estiver contido entre 0 e 127 e como negativo se estiver contido entre 128 e 255. Não se admitem números inferiores ou superiores aos referidos.
   Calcula-se o valor negativo de um número subtraindo-se d de 256. Por exemplo, a instrução de salto incondicional:
   jr d
   provoca um salto em frente de 8 *bytes* se d=8 e um salto para trás de 8 *bytes* se d=248 (=256−8). Convém recordar, no cálculo de um deslocamento, que um salto é sempre efectuado a partir do endereço do primeiro *byte* a seguir à intrução.
3) Um número de um *byte*, n, está contido entre 0 e 255, inclusive.
4) Um número de dois *bytes*, ou um endereço, é representado por nn e está contido entre 0 e 65535, inclusive. Calcula-se o seu valor adicionando o primeiro n ao produto do segundo por 256.
5) nn entre parênteses — isto é, (nn) — significa «o conteúdo da posição de memória cujo endereço é nn», enquanto nn sem parênteses significa «o número nn». Assim,
   ld hl,(23627)
   significa «carregar o par de registos hl com o conteúdo das posições de memória 23627 e 23628», enquanto:
   ld hl,23627
   significa «carregar hl com o número 23627». Do mesmo modo (hl) significa «o conteúdo da posição de memória cujo endereço está em hl», enquanto hl, sem parênteses, significa «o número contido em hl».
6) O destino do resultado de uma operação figura sempre em primeiro lugar. Por exemplo:
   add a,b
   significa «somar o conteúdo de b ao conteúdo de a, guardando o resultado em a».

GLOSSÁRIO DAS INSTRUÇÕES DE CÓDIGO MÁQUINA

Esta secção apresenta um sumário da maior parte das instruções do *Z80A*. Foram omitidas algumas instruções mais específicas para trabalhar com *interrupts*, etc.

«No operation» (não operar) — nop

Esta é, de todas, a instrução mais simples, estando implícito no seu nome que o processador nada faz quando a encontra. Apresenta, no entanto, extrema utilidade na fase de teste e detecção de erros numa rotina, uma vez que pode ser temporariamente substituída por uma instrução suspeita, sem alterar o funcionamento do resto da rotina. Pode também ser usada para preencher espaços vazios provocados por pequenas alterações a um programa ou para provocar pausas, especialmente se for introduzida num ciclo repetitivo convenientemente dimensionado. O seu código decimal é 0.

«Load» (carregar) — ld

Usam-se as instruções de *load* para movimentar um ou dois *bytes* entre registos ou entre registos e a memória. Existe mais de uma centena de instruções deste tipo, classificáveis em oito grupos:

1) Registo para registo a 8 *bits*.
   Os conteúdos de qualquer um dos registos a, b, c, d, e, h ou l podem ser copiados entre si.
2) Memória para registo a 8 *bits*.
   (hl), (ix+d) ou (iy+d) podem ser copiados para quaisquer dos registos a, b, c, d, e, h ou l. (bc), (de) ou (nn) podem ser copiados para a.
3) Registo para memória a 8 *bits*.
   a, b, c, d, e, h ou l podem ser copiados para (hl), (ix+d) ou (iy+d). a pode ser copiado para (bc), (de) ou (nn).
4) Imediato para registo ou memória.
   Imediato indica um número que é lido do programa em si, em vez de partir de um registo ou de um endereço em memória. Um dado número, n, pode ser carregado em a, b, c, d, e, h, l, (hl), (ix+d) ou (iy+d).
5) Registo para registo a 16 *bits*.
   Os conteúdos de hl, ix ou iy podem ser copiados para sp.
6) Memória para registo a 16 *bits*.
   (nn) pode ser copiado para bc, de, hl, ix, iy ou sp.
7) Registo para memória a 16 *bits*.
   bc, de, hl, ix, iy ou sp podem ser copiados para (nn).
8) Imediato para registo a 16 *bits*.
   nn pode ser carregado em bc, de, hl, ix, iy ou sp.

«Push» e «pop» (empurra para e retira de) — push, pop

Uma instrução *push* copia o conteúdo de um dado registo de 16 *bits* para a pilha e decrementa o ponteiro de pilha duas vezes. Uma instrução *pop* faz

exactamente o inverso, pelo que as duas instruções podem ser utilizadas em conjunto para guardar temporariamente valores de registos e obtê-los, de novo, mais tarde, dentro de um mesmo programa. *Push e pop* podem fazer actuar os pares de registos af, bc, de, hl, ix e iy.

«Exchange» (troca) ex

Podem ser efectuadas trocas de conteúdo entre hl e de, hl e (sp), ix e (sp), iy e (sp), af e af' e entre bcdehl e bcdehl' (uma só intrução faz a troca de todos estes seis registos de oito *bits*).

Adição e subtracção a 8 «bits» add, sub, etc.

a, b, c, d, e, h, l, (hl), n, (ix+d), e (iy+d) podem ser adicionados ou subtraídos ao registo a, com ou sem *flag* de transporte *(carry)*. As intruções que envolvem também esta *flag* terminam por c.

«And», «or» e «xor» («e» lógico, «ou» lógico e «ou» exclusivo) a 8 «bits» and, etc.

a, b, c, d, e, h, l, (hl), n, (ix+d), e (iy+d) podem formar combinações com o registo a, utilizando qualquer uma destas três operações lógicas. A operação *and* activa cada *bit* do resultado que estiver activo em ambos os operandos; a operação *or* activa cada *bit* do resultado que estiver activo pelo menos em um dos operandos e a operação *xor* activa cada *bit* do resultado que estiver activo num ou noutro dos operandos, mas não em ambos.

Compare (comparar) cp

A instrução cp é semelhante à subtracção, só que apenas afecta as *flags*, não alterando o conteúdo de a. Podemos comparar a, b, c, d, e, h, l, (hl), n, (ix+d), e (iy+d) com o acumulador.

Incremento e decremento a 8 «bits» inc, dec

a, b, c, d, e, h, l, (hl), n, (ix+d) e (iy+d) podem ser incrementados ou decrementados.

Incremento e decremento a 16 «bits» inc, dec

bc, de, hl, ix, iy e sp podem ser incrementados ou decrementados.

Adição e subtracção a 16 «bits» add, sub, etc.

bd, de, hl, ix podem ser adicionados, com ou sem transporte, ou subtraídos, apenas com transporte, ao par hl. bc, de, sp, ix podem ser adicionados, com transporte, a ix. bc, de, sp e iy podem ser adicionados, com transporte, a iy.

«Jump» (salto), «call» e «return» (chamada e retorno de sub-rotinas)

O registo de *flags*, f, contém uma *flag* de transporte, c, uma *flag* de paridade, p, que está activa após um resultado de paridade par, uma *flag* de sinal, s, que

está activa após um resultado negativo, uma *flag* de *overflow*, v, que está activa após a ocorrência de um *overflow*, e uma *flag* de zero, z, que está activa após um resultado nulo. Usam-se estas *flags* para controlar saltos *(jump)*, chamadas a sub-rotinas *(call)* e retorno de sub-rotinas *(ret)*.

1) *Jump* jp ou jr

São possíveis os seguintes saltos para um endereço nn:
salto absoluto incondicional (jp); salto se zero ou não (jp z e jp nz); salto se transporte ou não (jp c e jp nc); salto se positivo ou negativo (jp p e jp m); salto se p/v=1 ou p/v=0 (jp pe ou jp po).

São ainda possíveis os seguintes saltos, para um endereço d, relativo à posição actual, onde d é interpretado como um número de −128 a 127; salto relativo incondicional (jr); salto relativo se zero ou não (jr z ou jr nz); salto relativo se transporte ou não (jr c ou jr nc).

Podem ainda ser efectuados saltos para os endereços contidos em hl, ix ou iy [jp (hl), jp (ix), jp (iy)]. A instrução djnz decrementa o registo b e salta d instruções se b não for zero.

2) *Call* call

Esta instrução tem uma função semelhante ao comando GOSUB do BASIC. Se a condição de chamada for verdade, o programa passa para a instrução contida no endereço nn. Podem fazer-se as seguintes chamadas: *call* absoluto (call); *call* se zero ou não (call z ou call nz); *call* se transporte ou não (call c ou call nc); *call* se positivo ou não (call p ou call m); *call* se p/v=1 ou p/v=0 (call pe ou call po).

3) *Return* ret

Esta instrução tem uma função semelhante ao comando RETURN do BASIC. Existem também, disponíveis, condições de retorno para todas as condições já vistas, admitindo-se também retornos de pedidos de *interrupt* (INT) e de pedidos de «*interrupt* não mascarável» (NMI) — reti e retn, respectivamente.

Instruções a nível de «bit»

Os oito *bits* de cada registo estão numerados de 0 a 7, da direita para a esquerda. Podem ser executadas as seguintes operações em qualquer um dos registos a, b, c, d, e, h, l e ainda em (hl), (ix+d) e (iy+d).

1) *bit test* (teste de *bit*). bit

O teste de *bit* posiciona a *flag* de zero no estado oposto ao estado do *bit* testado. Qualquer dos *bits* pode ser testado.

2) *Bit set* (activação de *bit*) set

Qualquer dos *bits* pode ser posicionado no estado 1.

3) *Bit reset* (desactivação de *bit*) res

Qualquer dos *bits* pode ser posicionado no estado 0.

4) *Rotate left* (rotação à esquerda) rl
O bit 7 é copiado para a *flag* de transporte, esta é copiada para o *bit* 0 e todos os outros *bits* são deslocados de uma posição para a esquerda.

5) *Rotate right* (rotação à direita) rr
O *bit* 0 é copiado para a *flag* de transporte, esta é copiada para o *bit* 7 e todos os outros *bits* são deslocados uma posição para a direita.

6) *Rotate left circular* (rotação circular à esquerda) rlc
O *bit* 7 é copiado para o *flag* de transporte e para o *bit* 0. Os restantes *bits* são deslocados de uma posição para a esquerda.

7) *Rotate right circular* (rotação circular à direita) rrc
O *bit* 0 é copiado para a *flag* de transporte e para o *bit* 7. Os restantes *bits* são deslocados uma posição para a direita.

8) *Shift left arithmetic* (deslocação aritmética à esquerda) sla
Todos os *bits* são deslocados, para a esquerda, de uma posição, o *bit* 7 é copiado para a *flag* de transporte e o *bit* 0 é desactivado.

9) *Shift right arithmetic* (deslocação aritmética à direita) sra
Todos os *bits* são deslocados, para a direita, de uma posição, o *bit* 0 é copiado para a *flag* de transporte e o *bit* 7 é copiado para dentro de si mesmo.

10) *Shift right logical* (deslocação lógica à direita) srl
Uma deslocação aritmética à direita, em que o *bit* 7 é desactivado.

Rotação esquerda de um dígito («rotate left digit») rld

Os *bits* 0 a 3 de a são copiados para os *bits* 0 a 3 de (hl); os *bits* 0 a 3 de (hl) são copiados para os *bits* 4 a 7 de (hl); os *bits* 4 a 7 de (hl) são copiados para os *bits* 0 a 3 de a.

Rotação direita de um dígito («rotate right digit») rrd

Os *bits* 0 a 3 de a são copiados para os *bits* 4 a 7 de (hl); os *bits* 4 a 7 de (hl) são copiados para os *bits* 0 a 3 de (hl); os *bits* 0 a 3 de (hl) são copiados para os *bits* 0 a 3 de a.

Operações com o acumulador

1) *Complement* (complemento lógico) cpl
Todos os *bits* de a activos são desactivados e vice-versa.

2) *Negate* (troca de sinal) neg
Complementa logicamente a e soma-lhe uma unidade (complementação aritmética).

3) *Complement carry flag* (complemento da *flag* de transporte) ccf
Activa a *flag* de transporte, se estiver inactiva, e vice-versa.

4) *Set carry flag* (activação da *flag* de transporte) scf
Activa a *flag* de transporte.

5) *Decimal adjust* (ajuste decimal) daa
Corrige o valor de a, após uma operação em código DCB.

«Restart» rst

Guarda o contador de programa na pilha e salta para a posição de memória 8*n, onde n é o número contido na posição que se segue à instrução.

Manipulação de blocos

Estas instruções compostas destinam-se a deslocar ou a procurar dados em memória.

1) *Load and increment* (carregar e incrementar) ldi
Desloca um *byte* de (hl) para (de). Incrementa hl e de e decrementa bc.

2) *Load, increment and repeat* (carregar, incrementar e repetir) ldir
Desloca um *byte* de (hl) para (de). Incrementa hl e de e decrementa bc. Repete até bc=0.

3) *Load and decrement* (carregar e decrementar) ldd
Desloca um *byte* de (hl) para (de) e decrementa hl, de e bc.

4) *Load, decrement and repeat* (carregar, decrementar e repetir) lddr
Desloca um *byte* de (hl) para (de) e decrementa hl, de e bc. Repete até bc=0.

5) *Compare and increment* (comparar e incrementar) cpi
Compara a com (hl). Incrementa hl e decrementa bc.

6) *Compare, increment and repeat* (comparar, incrementar e repetir) cpir
Compara a com (hl). Incrementa hl e decrementa bc. Repete até a=(hl) ou bc=0.

7) *Compare and decrement* (comparar e decrementar) cpd
Compara a com (hl). Decrementa hl e bc.

8) *Compare, decrement and repeat* (comparar, decrementar e repetir) cpdr
Compara a com (hl). Decrementa hl e bc. Repete até a=(hl) ou bc=0.

Secção B

4. Introdução

As 40 rotinas em código máquina da Secção B são todas apresentadas no mesmo formato, a fim de simplificar a sua utilização. Esta introdução descreve esse formato e apresenta um programa BASIC que carrega as rotinas em memória.

Comprimento:
Representa o comprimento, em *bites*, da rotina.

Número de variáveis:
Controla-se a execução de uma rotina através da alteração dos valores de uma ou mais variáveis, passadas para a rotina através do *buffer* de impressão.

Teste soma:
Cada rotina é apresentada com uma sequência de números inteiros positivos, carregados (POKE) em posições de memória sucessivas. Dá-se o teste soma (isto é, a soma de todos os números que constituem a rotina) para que o utilizador verifique se carregou correctamente a rotina.

Operação:
Dá-se uma breve explicação da tarefa executada pela rotina.

Variáveis:
Apresentam-se os nomes, comprimentos e endereços, no *buffer* de impressão, de cada variável. Uma variável com o comprimento de um *byte* deve ser um número inteiro positivo entre 0 e 255, inclusive, sendo passada para a rotina a partir do BASIC ou do teclado, fazendo:

```
POKE endereço, valor
```

Uma variável de dois *bytes* passa através de dois comandos:

```
POKE endereço, valor−256*INT (valor/256)
POKE endereço+1, INT (valor/256)
```

Os endereços utilizados pertencem ao *buffer* de impressão.

Execução:
A execução das rotinas é comandada pela função USR, que deve para tal ser incorporada num comando. Se não se pretende que a rotina de código máquina passe dados no retorno ao BASIC, recomenda-se o uso do comando RAND, ou seja:

RAND USR endereço

Se se pretende obter, no retorno, o conteúdo do par de registos bc, usa-se:

LET A = USR endereço

ou

PRINT USR endereço

dependendo de se pretender que o valor de retorno seja guardado numa variável BASIC ou apresentado no *écran*.

Verificação de erros:

Explicam-se as verificações, efectuadas pela rotina, da ocorrência de valores de variáveis ilógicos ou incompatíveis, etc.

Comentários:

Apresentam-se variantes simples à rotina de base.

Listagem do código máquina:

A rotina é apresentada em linguagem *assembly* (assembler), com os respectivos códigos decimais na terceira coluna, de cabeçalho «Números a introduzir». Para carregar a rotina, os números da terceira coluna são introduzidos pelo comando POKE, sequencialmente, em memória. Todos os números apresentados são decimais.

Como funciona:

Explica-se, com base na listagem em código máquina, o modo como a rotina é executada.

CARREGADOR DE CÓDIGO MÁQUINA

Quase todas as rotinas apresentadas neste volume são «relocatáveis», o que significa que funcionarão correctamente, qualquer que seja a sua localização na RAM. Para as rotinas não «relocatáveis» explica-se, nos comentários, quais as alterações a efectuar se se pretender posicionar a rotina num local diferente do proposto.

Vimos, no capítulo 2 da Secção A, que o *Spectrum* usa zonas diferentes da RAM para diferentes funções, sendo a área contida entre as posições apontadas pelas variáveis do sistema RAMTOP e UDG reservada para o armazenamento de rotinas em código máquina.

O programa BP permite carregar, alterar e deslocar na memória uma rotina em código máquina. O utilizador, através deste programa, reposiciona o ponteiro RAMTOP a fim de obter mais espaço livre para uma dada rotina; introduzir uma rotina através do teclado; «percorrer» a rotina passo a passo para corrigir erros e inserir ou eliminar partes da rotina.

Quando se executa o programa (RUN), imprime-se no *écran* o menor endereço possível para armazenar uma rotina, ou seja o endereço que se segue a RAMTOP, e o espaço disponível entre esse endereço e o fim da RAM.

Nas máquinas de 16 K, o menor endereço é inicialmente 32600, a menos que o utilizador tenha alterado a variável do sistema RAMTOP. Do mesmo modo, nas máquinas de 48 K, o menor endereço é, no início, 65368.

Os últimos 168 *bytes* da RAM são normalmente reservados para os caracteres gráficos definíveis pelo utilizador *(user defined graphics)*, mas o programa permite que o utilizador use também esta área, se o desejar. Em alternativa, pode escolher um novo menor endereço possível, que o programa introduz no RAMTOP através do comando CLEAR. Não aceitará, no entanto, a introdução de um endereço inferior a 27800, uma vez que, desta forma, a rotina iria ser escrita por cima do espaço ocupado pelo programa em si, inutilizando-o. Pede-se, para cada rotina, a introdução do endereço inicial. Assim, o utilizador reserva espaço para várias rotinas e carrega cada uma delas separadamente.

Depois de dar ao utilizador a oportunidade de alterar a sua selecção, se não estiver satisfeito, o programa imprime o *écran* principal. A figura BF1 mostra o aspecto desse *écran* depois de ter sido carregada a rotina «Inversão de *écran*», a partir do endereço 32000. A primeira coluna apresenta os endereços ocupados, a segunda o conteúdo desses endereços e a terceira apresenta os testes soma parciais. A rotina «Inversão de *écran*» tem 18 *bytes* de comprimento e o seu teste soma é 1613. Ocupa, assim, as posições 32000 a 32017, sendo o teste soma para a posição 32017, isto é, a soma dos conteúdos das posições 32000 a 32017, de 1613.

Ao apresentar-se o *écran* principal, orienta-se a atenção do utilizador para uma determinada zona, que está a piscar. Essa zona é a chamada posição *corrente*, que é, no começo, o endereço inicial seleccionado para a rotina. O utilizador deve então introduzir, através do teclado, um número inteiro positivo entre 0 e 255, inclusive, que será por sua vez introduzido pelo programa na posição corrente, passando o endereço seguinte a ser a nova posição corrente. Desta forma, pode introduzir-se completamente uma rotina, sendo o *écran* principal actualizado (e rodado para cima, se necessário) em cada passo da introdução.

O utilizador, se não quiser introduzir um número, escolherá uma das opções apresentadas na tabela BT1. Esta possibilidade admite a execução de correcções.

Código	*Opção*
b	Deslocar a posição corrente, para trás, um endereço.
b *número*	Deslocar a posição corrente, para trás, um dado *número* de endereços.

f	Deslocar a posição corrente, para a frente, um endereço.
f *número*	Deslocar a posição corrente, para a frente, um dado *número* de endereços.
i *número*	Inserir um dado *número* de *bytes*, todos contendo zero, a partir da posição corrente.
d *número*	Apagar um dado *número* de *bytes*, a partir da posição corrente
t	Terminar o programa.

Tabela BT1. *Opções disponíveis para edição do código máquina.*

Programa BP. *Carregador de código máquina.*

```
 100 GO SUB 8100
 200 REM *****Calcula a memoria
disponivel
 210 LET min=1+PEEK 23730+256*PE
EK 23731
 220 LET p=PEEK 23732+256*PEEK 2
3733
 230 LET t=p-min+1
 400 REM *****Le o endereco de i
nicio
 410 PRINT "Menor endereco util
= ";min,,,"Espaco maximo disponi
vel = ";t
 420 INPUT "Deseja alterar o men
or endereco util (S ou N)? ";z$
 430 IF z$="S" OR z$="s" THEN GO
 TO 7000
 440 INPUT "Introduza o endereco
 inicial de carregamento do codi
go ";a
 450 IF a<min OR a>p THEN BEEP .
2,24: GO TO 440
 500 GO SUB 8100
 510 LET t=t-a+min
 520 PRINT "Pode usar ate ";t;"
bytes",,,
 530 LET u=PEEK 23675+256*PEEK 2
3676
 540 IF a<u AND u<p THEN PRINT "
Se usar mais de ";u-a;" bytes, a
"'"zona dos caracteres graficos"
'"sera alterada."
 550 IF a>=u THEN PRINT "Vai alt
erar a zona dos caracte- res gra
ficos."
 560 INPUT "Esta' certo (S ou N)
? ";z$
```



```
 570 IF z$="N" OR z$="n" THEN GO
 TO 7000
 580 IF z$<>"S" AND z$<>"s" THEN
 BEEP .2,24: GO TO 560
 700 REM *****Carregamento do co
digo
 710 LET l=a
 750 GO SUB 8200
 760 INPUT "Introduza um numero
ou b,f,i,d,t";z$
 770 IF z$="" THEN BEEP .2,24: G
O TO 760
 780 LET a$=CHR$ (CODE z$(1)-32*
(z$(1)>"£"))
 790 GO TO 800+200*(a$="B")+300*
(a$="F")+400*(a$="I")+500*(a$="D
")+600*(a$="T")
 800 LET x=VAL z$
 810 IF l>p THEN BEEP .2,24: GO
TO 750
 820 IF x<0 OR x>255 THEN BEEP .
2,24: GO TO 760
 830 POKE l,x
 840 LET l=l+1
 850 GO TO 740
1000 REM *****Avancar
1010 LET l=l-1
1020 IF LEN z$>1 THEN LET l=l+1-
VAL z$(2 TO )
1030 IF l<a THEN LET l=a
1040 GO TO 740
1100 REM *****Recuar
1110 LET l=l+1
1120 IF LEN z$>1 THEN LET l=l-1+
VAL z$(2 TO )
1130 IF l>p THEN LET l=p
1140 GO TO 740
1200 REM *****Inserir
1210 IF LEN z$=1 THEN LET n=1: G
O TO 1225
1220 LET n=VAL z$(2 TO ): IF n<1
 OR n>p-l OR n<>INT n THEN BEEP
.2,24: GO TO 740
1225 CLS : GO SUB 8100: PRINT TA
B 7;"Insercao em curso"
1230 FOR j=p TO l+n STEP -1
1240 POKE j,PEEK (j-n)
1250 NEXT j
1260 FOR j=l TO l+n-1
1270 POKE j,0
1280 NEXT j
1290 GO TO 740
1300 REM *****Apagar
```

```
1310 IF LEN z$=1 THEN LET n=1: G
O TO 1330
1320 LET n=VAL z$(2 TO ): IF n<1
 OR n>p-l OR n<>INT n THEN BEEP
.2,24: GO TO 740
1330 IF n<0 OR n>p-l THEN BEEP .
2,24: GO TO 1320
1340 CLS : GO SUB 8100: PRINT TA
B 6;"Apagamento em curso"
1350 FOR j=l TO p-n
1360 POKE j,PEEK (j+n)
1370 NEXT j
1380 GO TO 740
1400 PRINT AT 21,7;"Programa ter
minado"
1410 STOP
7000 REM *****Modificar RAMTOP
7010 INPUT "Introduza novo endre
co de inicio";a
7020 IF a<27000 OR a>p THEN BEEP
 .2,24: GO TO 7010
7030 CLEAR a-1
7040 RUN
7999 STOP
8100 CLS
8110 PRINT ;"<<Carregador de cod
igo maquina>>",,
8120 RETURN
8200 REM *****Imprime a memoria
8210 GO SUB 8100
8220 PRINT "Endereco  Decimal
Teste soma"
8230 LET c=0
8240 LET s=l-8: IF s<a THEN LET
s=a: GO TO 8280
8250 FOR j=a TO s-1
8260 LET c=c+PEEK j
8270 NEXT j
8280 LET f=s+17: IF f>p THEN LET
 f=p
8290 FOR j=s TO f
8300 LET c=c+PEEK j
8310 PRINT AT j-s+3,1;j;TAB 12;P
EEK j;TAB 22;c
8320 NEXT j
8400 LET pos=l-s+3
8410 PRINT AT pos,12; FLASH 1;PE
EK l
8420 RETURN
```


<<Carregador de codigo maquina>>

Endereco	Decimal	Teste soma
32000	33	33
32001	0	33
32002	64	97
32003	1	98
32004	0	98
32005	24	122
32006	22	144
32007	255	399
32008	122	521
32009	150	671
32010	119	790
32011	35	825
32012	11	836
32013	120	956
32014	177	1133
32015	32	1165
32016	247	1412
32017	201	1613

Figura BF1. *O* écran *apresentado pelo carregador de código máquina depois da introdução da rotina de inversão de* écran, *a partir da posição 32000.*

5. Rotinas de rotação de «écran»

ROTAÇÃO ESQUERDA DE ATRIBUTOS

Comprimento: 23
Número de variáveis: 1
Teste soma: 1574

Operação

Esta rotina provoca a rotação à esquerda, de um carácter, dos atributos de todos os caracteres no *écran*.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
novo atr	1	23296	Atributo a introduzir na coluna da direita

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Esta rotina é útil para sobreiluminar áreas de texto e gráficos. Para rodar as 22 linhas superiores do *écran*, deve-se substituir 24 (assinalado com *) por 22.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22528	33 0 88
	ld a, (23296)	58 0 91
	ld c, 24	14 24*
próx linha	ld b, 31	6 31
próx caract	inc hl	35
	ld e, (hl)	94
	dec hl	43
	ld (hl),e	115
	inc hl	35
	djnz, próx caráct	16 249
	ld (hl),a	119
	inc hl	35
	dec c	13
	jr nz, próx linha	32 242
	ret	201

Como funciona

O par de registos hl é carregado com o endereço da zona de atributos. O acumulador é carregado com o valor do atributo a ser introduzido na coluna da direita. O registo c é carregado com o número de linhas a rodar, a fim de funcionar como contador de linhas. O registo b é posicionado com o número de caracteres por linha, menos um, para funcionar como contador.

hl é incrementado, apontando para o próximo atributo, que se carrega no registo e. hl é decrementado, introduzindo-se o conteúdo de e no endereço para onde este aponta. hl é incrementado de novo, apontando para o atributo seguinte. O registo b é decrementado e, se o seu conteúdo não for zero, salta-se para «próx caráct». hl aponta agora para a coluna da direita, que se carrega com o valor contido no acumulador. hl é incrementado, de forma a apontar para o início da próxima linha. O contador de linhas, ou seja, o registo c, é decrementado. Se o valor resultante for não nulo, a rotina volta a «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO DIREITA DE ATRIBUTOS

Comprimento: 23
Número de variáveis: 1
Teste soma: 1847

Operação

Esta rotina provoca a rotação à direita, de um carácter, dos atributos de todos os caracteres no *écran*.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
novo atr	1	23296	Atributo a introduzir na coluna da esquerda.

Execução

RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é útil para sobreiluminar áreas de texto e gráficos. Para rodar apenas as 22 linhas superiores do *écran*, devemos substituir 24 (assinalado com *) por 22.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 23295	33 255 90
	ld a, (23296)	58 0 91
	ld c, 24	14 24*
próx linha	ld b, 31	6 31
próx caráct	dec hl	43
	ld e, (hl)	94
	inc hl	35
	ld (hl),e	115
	dec hl	43
	djnz, próx caráct	16 249
	ld (hl),a	119
	dec hl	43
	dec c	13
	jr nz, próx linha	32 242
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do último *byte* da zona de atributos. O acumulador é carregado com o valor do atributo a introduzir na coluna da esquerda. O registo c é carregado com o número de linhas a rodar, a fim de funcionar como contador de linhas. O registo b é posicionado com o número de caracteres por linha, menos um, para funcionar como contador.

hl é decrementado, apontando para o próximo atributo, que se carrega no registo e. hl é incrementado, introduzindo-se o conteúdo de e no endereço para onde este aponta. hl é decrementado de novo, apontando para o atributo seguinte. O contador, no registo b, é decrementado e, se o seu conteúdo não for zero, efectua-se um salto para «próx caráct».

hl aponta agora para a coluna da esquerda, que se carrega com o valor contido no acumulador. hl é decrementado, de forma a apontar para o fim da próxima linha. O contador de linhas, ou seja, o registo c, é decrementado e, se o valor resultante for não nulo, a rotina volta a «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO SUPERIOR DE ATRIBUTOS
Comprimento: 21
Número de variáveis: 1
Teste soma: 1591

Operação
Esta rotina provoca a rotação dos atributos de todos os caracteres no *écran*, para cima, de um carácter.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
novo atr	1	23296	Atributo a introduzir na linha de baixo.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é útil para sobreiluminar áreas de texto e gráficos. Para rodar apenas as 22 linhas superiores do *écran*, deve-se substituir 224 (assinalado com *) por 160.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22560	33 32 88
	ld de, 22528	17 0 88
	ld bc, 736	1 224* 2
	ldir	237 176
	ld a, (23296)	58 0 91
	ld b, 32	6 32
próx caráct	ld (de),a	18
	inc de	19
	djnz próx caráct	16 252
	ret	201

Como funciona
hl é carregado com o endereço da segunda linha de atributos, de é carregado com o endereço da primeira linha e bc com o número de *bytes* a deslocar.

Efectua-se uma cópia de bc *bytes*, a partir de hl, para de, através da instrução «ldir». Como resultado de fica a apontar para a linha inferior de atributos. O acumulador é carregado com o código de atributo a introduzir na linha de baixo. O registo b é, então, carregado com o número de caracteres por linha, para funcionar como contador.

Carrega-se o conteúdo do acumulador no endereço apontado por de, sendo este par incrementado, de forma a apontar para o próximo *byte*. O contador é decrementado e, se o seu conteúdo não for zero, efectua-se um salto para «próx caráct».

A rotina regressa, então, ao BASIC.

ROTAÇÃO INFERIOR DE ATRIBUTOS

Comprimento: 21
Número de variáveis: 1
Teste soma: 2057

Operação

Esta rotina provoca a rotação dos atributos de todos os caracteres no *écran*, para baixo, de um carácter.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
novo atr	1	23296	Atributo a introduzir na linha de cima.

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Esta rotina é útil para sobreiluminar áreas de texto e gráficos. Para rodar apenas as 22 linhas superiores do *écran*, devem efectuar-se as seguintes alterações:

223* para 159
255** para 160
224*** para 160

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 23263	33 223* 90
	ld de, 23295	17 255** 90
	ld bc, 736	1 224*** 2
	lddr	237 184
	ld a, (23296)	58 0 91
	ld b, 32	6 32
próx caráct	ld (de),a	18
	dec de	27
	djnz próx caráct	16 252
	ret	201

Como funciona

hl é carregado com o endereço do último atributo da 23.ª linha.

de é carregado com o endereço do último atributo da 24.ª linha.

Carrega-se bc com o número de *bytes* a deslocar. A instrução «lddr» transfere os bc *bytes* que terminam em hl de forma a que terminem em de. Isto tem como resultado ficar de a apontar para o endereço do último atributo da primeira linha.

O acumulador é carregado com o código de atributo a introduzir na linha de cima. O registo b é, então, carregado com o número de caracteres por linha, para funcionar como contador. O conteúdo do acumulador é carregado no endereço apontado por de, sendo este par decrementado, de forma a apontar para o próximo *byte*. O contador é decrementado e, se o seu conteúdo não for zero, efectua-se um salto para «próx caráct».

A rotina regressa, então, ao BASIC.

ROTAÇÃO À ESQUERDA DE UM CARÁCTER

Comprimento: 21
Número de variáveis: 0
Teste soma: 1745

Operação

Esta rotina roda o conteúdo do ficheiro de *écran* de um carácter para a esquerda.

Execução

RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é útil para a utilização do *écran* como «janela», apresentando apenas uma pequena porção de uma área de imagem muito maior. A «janela» move-se dentro dessa área através das rotinas de deslocação de imagem.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld d,l	85
	ld a,192	62 192
próx linha	ld b,31	6 31
próx byte	inc hl	35
	ld e, (hl)	94
	dec hl	43
	ld (hl),e	115
	inc hl	35
	djnz próx byte	16 249
	ld (hl),d	114
	inc hl	35
	dec a	61
	jr nz, próx linha	32 242
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do ficheiro de *écran*, sendo o registo d posicionado a zero. O acumulador é carregado com o número de linhas do *écran*. O registo b é carregado com um menos o número de caracteres por linha, pois é este o número de *bytes* a copiar.

hl é incrementado, de forma a apontar para o próximo *byte*, e carrega-se o registo e com este valor. hl é decrementado, carregando-se o conteúdo de e na posição para que este aponta. hl é incrementado, para apontar o próximo *byte*, e o contador no registo b é decrementado. Se o seu conteúdo não for nulo, efectua-se um salto para «próx *byte*».

Se o conteúdo do registo b for nulo, já foi copiado o último *byte* da linha e hl aponta para o seu *byte* mais à direita. Este *byte* é carregado com zero e hl é incrementado, de forma a apontar para a próxima linha. O contador de linhas, no acumulador, é decrementado e, se o seu conteúdo não for nulo, efectua-se um salto para «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO À DIREITA DE UM CARÁCTER
Comprimento: 22
Número de variáveis: 0
Teste soma: 1976

Operação
Esta rotina roda o conteúdo do ficheiro de *écran* de um carácter para a direita.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é útil para a utilização do *écran* como «janela», apresentando apenas uma pequena porção de uma área de imagem muito maior. A «janela» move-se dentro dessa área através das rotinas de deslocação de imagem.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22527	33 255 87
	ld d, O	22 O
	ld a, 192	62 192
próx linha	ld b, 31	6 31
próx byte	dec hl	43
	ld e, (hl)	94
	inc hl	35
	ld (hl),e	115
	dec hl	43
	djnz próx byte	16 249
	ld (hl),d	114
	dec hl	43
	dec a	61
	jr nz, próx linha	32 242
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do último *byte* do ficheiro de *écran*, posicionando-se a zero o registo d. Carrega-se o acumulador com o número de linhas do *écran*. O registo b é carregado com um menos o número de caracteres por linha, para funcionar como contador.

O par de registos hl é decrementado, de forma a apontar para o próximo *byte*, e o seu valor é carregado no registo e. hl é incrementado, carregando-se o conteúdo de e na posição para que este aponta. hl é decrementado, para apontar o próximo *byte*, e o contador no registo b é decrementado. Se o seu conteúdo não for nulo, efectua-se um salto para «próx *byte*».

Se o conteúdo do registo b for nulo, hl aponta para o *byte* mais à esquerda da linha. Este *byte* é carregado com zero e hl é decrementado, de forma a apontar para a próxima linha. O contador no acumulador é decrementado e, se o seu conteúdo não for nulo, efectua-se um salto para «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO SUPERIOR DE UM CARÁCTER

Comprimento: 68
Número de variáveis: 0
Teste soma: 6328

Operação

Esta rotina roda o conteúdo do ficheiro de *écran*, para cima, de oito *pixels*.

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Nenhum

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld de, 16416	17 32 64
guardar	push hl	229
	push de	213
	ld c,23	14 23
próx linha	ld b,32	6 32
copiar byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	and 7	230 7
	cp 1	254 1
	jr nz, próx byte	32 2
	sub a	151
	ld (de),a	18
próx byte	inc hl	35
	inc de	19
	djnz copiar byte	16 241
	dec c	13
	jr z, devolver	40 19
	ld a,c	121
	and 7	230 7
	cp O	254 O
	jr z, próx bloco	40 22
	cp 7	254 7
	jr nz, próx linha	32 225
	push de	213
	ld de, 1792	17 0 7
	add hl,de	25
	pop de	209
	jr próx linha	24 217
devolver	pop de	209
	pop hl	225
	inc d	20
	inc h	36
	ld a,h	124
	cp 72	254 72
	jr nz, guardar	32 204
	ret	201
próx bloco	push hl	229
	ld hl, 1792	33 0 7
	add hl,de	25
	ex de,hl	235
	pop hl	225
	jr próx linha	24 198

Como funciona

O par de registos hl é carregado com o endereço do ficheiro de *écran* e o par de com o endereço do *byte* situado oito linhas abaixo. hl e de são guardados na pilha. O registo c é carregado com um menos o número total de linhas no

écran. O registo b é carregado com o número de *bytes* de uma linha de caracteres, para funcionar como contador.

O acumulador é carregado com o *byte* endereçado por de, sendo este byte carregado no endereço apontado por hl. Carrega-se o acumulador com o conteúdo do registo c e, se este for 1, 9 ou 17, o endereço apontado por de é carregado com zero. hl e de são incrementados, de forma a apontarem os bytes seguintes. O contador no registo b é decrementado e, se o conteúdo não for nulo, efectua-se um salto para «copiar *byte*».

O contador de linhas no registo c é decrementado. Se o seu conteúdo for nulo, efectua-se um salto para «devolver». Se for 8 ou 16, o salto é para «próx bloco». Se não for 7 nem 15, a rotina volta a «próx linha». Adiciona-se 1792 a hl, para que este aponte para o próximo bloco de *écran*. A rotina salta, então, para «próx linha».

Em «devolver», de e hl são retirados da pilha, adicionando-se 256 a cada um deles. Assim, de e hl apontam para a linha imediatamente abaixo da do ciclo anterior. Se hl contém 18432, a rotina regressa ao BASIC. Se não, efectua-se um salto para «guardar». Em «próx bloco» adiciona-se 1792 a de, de forma a que este aponte para o próximo bloco de *écran*. A rotina salta, finalmente para «próx linha».

ROTAÇÃO INFERIOR DE UM CARÁCTER

Comprimento: 73
Número de variáveis: 0
Teste soma: 7987

Operação

Esta rotina roda o conteúdo do ficheiro de *écran*, para baixo, de oito *pixels*.

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Nenhum

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22527	33 255 87
	ld de, 22495	17 223 87
guardar	push hl	229
	push de	213
	ld c, 23	14 23
próx linha	ld b,32	6 32
copiar byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	and 7	230 7
	cp 1	254 1
	jr nz, próx byte	32 2
	sub a	151
	ld (de),a	18
próx byte	dec hl	43
	dec de	27
	djnz copiar byte	16 241
	dec c	13
	jr z, devolver	40 21
	ld a,c	121
	and 7	230 7
	cp O	254 O
	jr z, próx bloco	40 24
	cp 7	254 7
	jr nz, próx linha	32 225
	push de	213
	ld de, 1792	17 0 7
	and a	167
	sbc hl,de	237 82
	pop de	209
	jr próx linha	24 215
devolver	pop de	209
	pop hl	225
	dec d	21
	dec h	37
	ld a,h	124
	cp 79	254 79
	ret z	200
	jr guardar	24 201
próx bloco	push hl	229
	ld hl, 1792	33 0 7
	ex de, hl	235
	and a	167

sbc hl,de	237 82
ex de, hl	235
pop hl	225
jr próx linha	24 193

Como funciona
O par de registos hl é carregado com o endereço do último *byte* do ficheiro de *écran* e o par de é carregado com o endereço do *byte* situado oito linhas acima. hl e de são guardados na pilha. O registo c é carregado com um menos o número total de linhas no *écran*. O registo b é carregado com o número de *bytes* de uma linha de caracteres, para ser utilizado como contador.

O acumulador é carregado com o *byte* endereçado por de, sendo este *byte* carregado no endereço apontado por hl. O acumulador é carregado com o conteúdo do registo c e, se este for 1, 9 ou 17, o endereço apontado por de é carregado com zero. hl e de são decrementados, de forma a apontarem os *bytes* seguintes. O contador no registo b é decrementado e, se o seu conteúdo não for nulo, efectua-se um salto para «copiar *byte*».

O contador de linhas no registo c é decrementado. Se o seu conteúdo for nulo, efectua-se um salto para «devolver». Se for 8 ou 16, o salto é para «próx bloco». Se não for 7 nem 15, a rotina volta a «próx linha». Subtrai-se 1792 a hl, para que este aponte para o próximo bloco de *écran*. A rotina salta, então, para «próx linha».

Em «devolver», de e hl são retirados da pilha, subtraindo-se 256 de cada um deles. Assim, de e hl apontam para a linha imediatamente acima da do ciclo anterior. Se hl contém 20479, a rotina regressa ao BASIC. Caso contrário, efectua-se um salto para «salvar». Em «próx bloco» subtrai-se 1792 a de, de forma a que este aponte para o próximo bloco de *écran*. A rotina salta, finalmente, para «próx linha».

ROTAÇÃO À ESQUERDA DE UM «PIXEL»
Comprimento: 17
Número de variáveis: 0
Teste soma: 1828

Operação
Esta rotina roda o conteúdo do ficheiro de *écran*, para a esquerda, de um *pixel*.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina provoca um deslocamento da imagem mais suave que a rotina de rotação à esquerda de um carácter, mas são necessárias oito execuções para deslocar o *écran* de um só caracter.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22527	33 255 87
	ld c, 192	14 192
próx linha	ld b, 32	6 32
	or a	183
próx byte	rl (hl)	203 22
	dec hl	43
	djnz próx byte	16 251
	dec c	13
	jr nz, próx linha	32 245
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do último *byte* do ficheiro de *écran* e o registo c é carregado com o número de linhas do *écran*, para ser utilizado como contador de linhas. O registo b é carregado com o número de *bytes* por linha para funcionar como contador. Posiciona-se a zero a *flag* de transporte.

O *byte* endereçado por hl roda um *bit* para a esquerda, sendo a *flag* de transporte copiada para o *bit* mais à direita e o *bit* mais à esquerda copiado para esta *flag*. O par hl é decrementado, apontando para o próximo *byte*, e o contador no registo b é decrementado. Se o seu conteúdo não for nulo, a rotina volta para «próx *byte*». O contador de linhas é decrementado e, se não for igual a zero, efectua-se um salto para «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO À DIREITA DE UM «PIXEL»
Comprimento: 17
Número de variáveis: 0
Teste soma: 1550

Operação
Esta rotina roda o conteúdo do ficheiro de *écran*, para a direita, de um *pixel*.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

COMENTÁRIOS
Esta rotina provoca um deslocamento da imagem mais suave que a rotina de rotação à direita de um carácter, mas são necessárias oito execuções para deslocar o *écran* de um só carácter.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld c, 192	14 192
próx linha	ld b, 32	6 32
	or a	183
próx byte	rr (hl)	203 30
	inc hl	35
	djnz próx byte	16 251
	dec c	13
	jr nz, próx linha	32 245
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do ficheiro de *écran* e o registo c é carregado com o número de linhas do *écran*, para funcionar como contador de linhas. O registo b é carregado com o número de *bytes* por linha, para funcionar como contador. A *flag* de transporte posiciona-se a zero. O *byte* endereçado por hl roda um *bit* para a direita, copiando-se a *flag* de transporte para o *bit* mais à esquerda e o *bit* mais à direita para esta *flag*. O par hl é incrementado, apontado para o próximo *byte*, e o contador no registo b é decrementado. Se o seu conteúdo não for nulo, a rotina volta para «próx *byte*». O contador de linhas é decrementado e, se não for igual a zero, efectua-se um salto para «próx linha».

A rotina regressa, então, ao BASIC.

ROTAÇÃO SUPERIOR DE UM «PIXEL»
Comprimento: 91
Número de variáveis: 0
Teste soma: 9228

Operação
Esta rotina roda o conteúdo do ficheiro de *écran*, para cima, de um *pixel*.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Nenhuns

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld de, 16640	17 0 65
	ld c, 192	14 192
próx linha	ld b, 32	6 32
copiar byte	ld a, (de)	26
	ld (hl),a	119
	ld a,c	121
	cp 2	254 2
	jr nz, próx byte	32 2
	sub a	151
	ld (de),a	18
próx byte	inc de	19
	inc hl	35
	djnz copiar byte	16 243
	push de	213
	ld de, 224	17 224 O
	add hl,de	25
	ex (sp),hl	227
	add hl,de	25
	ex de,hl	235
	pop hl	225
	dec c	13
	ld a,c	121
	and 7	230 7

	cp O	254 O
	jr nz, subtrair	32 10
	push de	213
	ld de, 2016	17 224 7
	and a	167
	sbc hl,de	237 82
	pop de	209
	jr próx bloco	24 14
subtrair	cp 1	254 1
	jr nz, próx bloco	32 10
	push hl	229
	ex de,hl	235
	ld de, 2016	17 224 7
	and a	167
	sbc hl,de	237 82
	ex de,hl	235
	pop hl	225
próx bloco	ld a,c	121
	and 63	230 63
	cp O	254 O
	jr nz, somar	32 6
	ld a,7	62 7
	add a,h	132
	ld h,a	103
	jr próx linha	24 187
somar	cp 1	254 1
	jr nz, próx linha	32 183
	ld a,7	62 7
	add a,d	130
	ld d,a	87
	ld a,c	121
	cp 1	254 1
	jr nz, próx linha	32 174
	ret	201

Como funciona

O par de registos hl é carregado com o endereço do ficheiro de *écran* e o par de registos de é carregado com o endereço do primeiro *byte* da segunda linha de *écran*. O registo c é carregado com o número de linhas no *écran*. O registo b é carregado com o número de *bytes* por linha, para funcionar como contador.

O acumulador é carregado com o *byte* endereçado por de, que se introduz no endereço apontado por hl. O acumulador é carregado com o conteúdo do

registo c. Se for dois, de aponta para a última linha do *écran*, pelo que é carregado a zeros. de e hl são incrementados, ficando apontados para os *bytes* seguintes. O contador no registo b é decrementado e, se o seu conteúdo não for nulo, a rotina volta para «copiar *byte*».

Adiciona-se 224, quer a hl quer a de, de forma a que eles apontem para a próxima linha de *écran*. O contador de linhas, no registo c, é decrementado. Se o valor em c não for múltiplo de oito, efectua-se um salto para «subtrair». Subtrai-se 2016 de hl e salta-se para «próx bloco». A rotina fica, assim, posicionada para o próximo conjunto de oito linhas.

Em «subtrair», se o valor (c−1) não for um múltiplo de oito, salta-se para «próx bloco». Caso contrário subtrai-se 2016 de de, para que este aponte para um novo conjunto de oito linhas. Em «próx bloco», se c for um múltiplo de 64, soma-se 1792 a hl e salta-se para «próx linha», de forma a que hl aponte o próximo bloco de 64 linhas. Em «somar», se (c−1) for um múltiplo de 64, soma-se 1792 a de, de forma a que este par de registos aponte para o próximo bloco de 64 linhas. Se o conteúdo de c não for 1, a rotina salta para «próx linha», regressando ao BASIC no caso contrário.

ROTAÇÃO INFERIOR DE UM «PIXEL»

Comprimento: 90
Número de variáveis: 0
Teste soma: 9862

Operação

Esta rotina roda o conteúdo do ficheiro de *écran*, para baixo, de um *pixel*.

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Nenhuns

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22527	33 255 87
	ld de, 22271	17 255 86

```
            ld c, 192            14 192
próx linha  ld b, 32             6 32
copiar byte ld a, (de)           26
            ld (hl),a            119
            ld a,c               121
            cp 2                 254 2
            jr nz, próx byte     32 2
            sub a                151
            ld (de),a            18
próx byte   dec de               27
            dec hl               43
            djnz copiar byte     16 243
            push de              213
            ld de, 224           17 224 O
            and a                167
            sbc hl,de            237 82
            ex (sp), hl          227
            and a                167
            sbc hl,de            237 82
            ex de,hl             235
            pop hl               225
            dec c                13
            ld a,c               121
            and 7                230 7
            cp O                 254 O
            jr nz, somar         32 8
            push de              213
            ld de, 2016          17 224 7
            add hl,de            25
            pop de               209
            jr próx bloco        24 11
somar       cp 1                 254 1
            jr nz, próx bloco    32 7
            push hl              229
            ld hl, 2016          33 224 7
            add hl,de            25
            ex de,hl             235
            pop hl               225
próx bloco  ld a,c               121
            and 63               230 63
            cp O                 254 O
            jr nz, subtrair      32 6
```



```
            ld a,h               124
            sub 7                214 7
            ld h,a               103
            jr próx linha        24 188
subtrair    cp 1                 254 1
            jr nz, próx linha    32 184
            ld a,d               122
            sub 7                214 7
            ld d,a               87
            ld a,c               121
            cp 1                 254 1
            jr nz, próx linha    32 175
            ret                  201
```

Como funciona

O par de registos hl é carregado com o endereço do último *byte* do ficheiro de *écran* e o par de registos de é carregado com o endereço do *byte* situado uma linha acima do último. O registo c é carregado com o número de linhas do *écran*. O registo b é carregado com o número de *bytes* por linha, para ser utilizado como contador.

O acumulador é carregado com o *byte* endereçado por de, que é introduzido no endereço apontado por hl. É, depois, carregado com o conteúdo do registo c. Se este for dois, de aponta para a primeira linha do *écran*, pelo que é carregado a zeros. de e hl são decrementados, ficando apontados para os *bytes* seguintes. O contador no registo b é decrementado e, se o seu conteúdo não for nulo, a rotina volta para «copiar *byte*».

Subtrai-se 224, quer a hl quer a de, de forma a que eles apontem para a próxima linha de *écran*. O contador de linhas, no registo c, é decrementado. Se o valor em c não for um múltiplo de oito, salta-se para «somar». Soma-se 2016 a hl e salta-se para «próx bloco». hl fica, assim, posicionado para o próximo conjunto de oito linhas.

Em «somar», se o valor (c−1) não for um múltiplo de oito, salta-se para «próx bloco». Caso contrário soma-se 2016 a de, para que este aponte para um novo conjunto de oito linhas. Em «próx bloco», se c for um múltiplo de 64, subtrai-se 1792 de hl, de forma a que este par aponte o próximo bloco de 64 linhas, e salta-se para «próx linha». Em «subtrair», se (c−1) for um múltiplo de 64, subtrai-se 1792 de de, de forma a que este par de registos aponte para o próximo bloco de 64 linhas. Se o conteúdo de c não for 1, a rotina salta para «próx linha», regressando ao BASIC no caso contrário.

6. Rotinas de manipulação de imagem

MISTURA DE IMAGENS
Comprimento: 21
Número de variáveis: 1
Teste soma: 1709

Operação
Esta rotina mistura uma imagem armazenada na RAM (utilizando a rotina de «Cópia» descrita mais à frente), com a imagem presente no *écran*. Os atributos não são alterados.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
zona *écran*	2	23296	Endereço na RAM da imagem a misturar.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Para misturar imagens, devemos utilizar esta rotina tal como se apresenta. Contudo, obtêm-se efeitos bastante interessantes, substituindo a instrução «or(hl) 182» por «xor(hl) 174» ou «and(hl) 166».

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld de, (23296)	237 91 0 91
	ld bc, 6144	1 0 24
próx byte	ld a, (de)	26
	or (hl)	182
	ld (hl),a	119
	inc hl	35
	inc de	19
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, próx byte	32 246
	ret	201

Como funciona
O par de registos hl é carregado com o endereço do ficheiro de *écran* e o par de registos de é carregado com o comprimento da imagem em memória, para funcionar como contador.

O acumulador é carregado com *byte* cujo endereço está armazenado em de, efectuando-se de seguida um «OU lógico» deste *byte* com o próximo *byte* do ficheiro de *écran*. O resultado desta operação é armazenado no *byte* do ficheiro de *écran*.

hl e de apontam para as posições seguintes e o contador é decrementado. Se o contador não tiver ainda atingido zero, a rotina volta a repetir o ciclo para o *byte* seguinte.

INVERSÃO VÍDEO DO «ÉCRAN»
Comprimento: 18
Número de variáveis: 0
Teste soma: 1613

Operação
Inverte todo o ficheiro de *écran* — iluminam-se todos os pontos apagados e apagam-se todos os pontos iluminados.

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina, em programas de jogos, produz um efeito eficaz de explosão. Este efeito aumenta se se chamar a rotina várias vezes seguidas, acompanhadas com a produção de sons.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 16384	33 0 64
	ld bc, 6144	1 0 24
	ld d, 255	22 255
próx byte	ld a,d	122
	sub (hl)	150
	ld (hl),a	119
	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, próx byte	32 247
	ret	201

Como funciona

O par de registos hl é carregado com o endereço do ficheiro de *écran* e bc com o respectivo comprimento. O registo d posiciona-se com 255. De cada vez que a rotina repete o ciclo, o acumulador é carregado a partir de d. Este método é preferível ao uso da instrução «ld a,255», uma vez que a execução de «ld a,d» demora cerca de metade do tempo em relação a «ld a,255». O valor do *byte* armazenado em hl é subtraído do acumulador e armazenado de novo no mesmo *byte*, que fica assim complementado.

hl é incrementado para apontar para o próximo *byte* e o contador, bc, é decrementado. Se o contador não estiver a zero, a rotina volta para «próx *byte*» e repete-se o processo.

INVERSÃO VERTICAL DE CARACTERES

Comprimento: 20
Número de variáveis: 1
Teste soma: 1757

Operação

Esta rotina inverte verticalmente um carácter, p. ex.: uma flecha para cima resulta numa flecha para baixo e vice-versa.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
caráct	2	23298	Endereço na RAM da descrição do carácter

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Esta rotina é útil em jogos de certos tipos, uma vez que os símbolos mudam de direcção sem que usemos mais que um carácter.

Listagem de código máquina

Label	*Assembler*	*Números introduzidos*
	ld hl, (23296)	42 0 91
	ld d,h	84
	ld e,l	93
	ld b,8	6 8
próx byte	ld a, (hl)	126
	inc hl	35
	push af	245
	djnz próx byte	16 251
	ld b,8	6 8
substituir	pop af	241
	ld (de),a	18
	inc de	19
	djnz substituir	16 251
	ret	201

Como funciona

O par de registos hl é carregado com o endereço, na RAM, da descrição do carácter, que é depois carregado em de. O registo b posiciona-se a 8, para funcionar como contador.

Para cada *byte*, o acumulador é carregado com o seu valor actual, hl é incrementado para apontar o próximo *byte* e o acumulador é guardado na pilha. O contador é decrementado e, se não for zero, a rotina salta para trás, repetindo o processo para o *byte* seguinte. O registo b é de novo posicionado a 8, para servir mais uma vez como contador.

Para cada *byte*, o acumulador é retirado da pilha e o seu conteúdo carregado no endereço apontado por de. de é decrementado, apontado para o próximo *byte*, e o contador é também decrementado. Se não for zero, a rotina salta para «substituir». Caso contrário, regressa ao BASIC.

INVERSÃO HORIZONTAL DE CARACTERES

Comprimento: 19
Número de variáveis: 1
Teste soma: 1621

Operação

Esta rotina inverte horizontalmente um carácter, por exemplo: uma flecha para a esquerda resulta numa flecha para a direita e vice-versa.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
caráct	2	23296	Endereço na RAM da descrição do carácter

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Nenhum

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld a, 8	62 8
próx byte	ld b, 8	6 8
próx pixel	rr (hl)	203 30
	rl c	203 17
	djnz próx pixel	16 250
	ld (hl),c.	113
	inc hl	35
	dec a	61
	jr nz, próx byte	32 243
	ret	201

Como funciona

O par de registos hl é carregado com o endereço na RAM da descrição do carácter e o acumulador é carregado com o número de *bytes* a inverter. O registo b é carregado com o número de *bits* por cada *byte*, para funcionar como contador.

O *byte* contido no endereço apontado por hl roda à direita, de forma a que o seu *bit* da direita seja copiado para a *flag* de transporte. O registo c roda à esquerda, de forma a que a *flag* de transporte seja copiada para o *bit* mais à direita. O contador guardado no registo b é decrementado. Se o seu conteúdo não for nulo, efectua-se um salto, de volta para «próx *pixel*». O *byte* invertido, que fica no registo c, transfere-se finalmente para o endereço de onde veio.

hl é incrementado, apontado para o próximo *byte*, e o acumulador é decrementado. Se o conteúdo deste não for nulo, efectua-se um salto, de volta para «próx *byte*».

A rotina regressa, então, ao BASIC.

ROTAÇÃO HORÁRIA DE CARACTERES

Comprimento: 42
Número de variáveis: 1
Teste soma: 3876

Operação

Esta rotina provoca a rotação de 90°, no sentido horário, de um dado carácter. Por exemplo uma flecha para a cima resulta numa flecha para a direita.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
caráct	2	23296	Endereço na RAM da descrição do carácter

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Esta rotina é útil, quer para jogos, quer para aplicações mais sérias, por exemplo para etiquetar gráficos.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld e, 128	30 128

```
próx bit      push hl             229
              ld c, O             14 0
              ld b,1              6 1
próx byte     ld a, e             123
              and (hl)            166
              cp O                254 0
              jr z, não activo    40 3
              ld a,c              121
              add a,b             128
              ld c,a              79
não activo    sla b               203 32
              inc hl              35
              jr nc, próx byte    48 242
              pop hl              225
              push bc             197
              srl e               203 59
              jr nc, próx bit     48 231
              ld de,7             17 7 0
              add hl,de           25
              ld b,8              6 8
substituir    pop de              209
              ld (hl),e           115
              dec hl              43
              djnz substituir     16 251
              ret                 201
```

Como funciona

Cada carácter é constituído por um conjunto de 8×8 *pixels*, cada um dos quais pode ser activado (=1) ou desactivado (=0). Considere-se um *bit* qualquer B_2 do *byte* B_1 da figura B_1. Os valores situados na posição (B_2, B_1) da matriz serão:

$$\begin{bmatrix} N_1 & N_3 \\ N_2 & N_4 \end{bmatrix}$$

onde:

N_1 = o *byte* em que o *pixel* (B_2, B_1) será inserido após a rotação.
N_2 = o *bit* de N_1 onde o *pixel* será inserido.
N_3 = o valor corrente do *bit*.
N_4 = o valor do *bit* N_2.

Os *bytes* do carácter rodado serão construídos, um a um, pela acumulação dos valores de todos os bits N_2 que estarão no novo *byte*.

O par de registos hl é carregado com o endereço na RAM do primeiro *byte* do carácter. O registo e é carregado com o valor binário que tem o *bit* 7 activo e os restantes inactivos, isto é, 128. O par de registos hl é guardado na pilha. O registo c, no qual se irão acumular dados que permitirão obter o novo valor do *byte* em construção, é carregado com zero. O registo b é carregado com o valor binário que tem o *bit* 0 activo e os restantes inactivos, isto é, 1.

																Byte (B_1)
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	2
1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	3
2	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	4
3	8	3	8	3	8	3	8	3	8	3	8	3	8	3	8	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	5
4	16	4	16	4	16	4	16	4	16	4	16	4	16	4	16	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	6
5	32	5	32	5	32	5	32	5	32	5	32	5	32	5	32	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	7
6	64	6	64	6	64	6	64	6	64	6	64	6	64	6	64	
1	128	2	64	3	32	4	16	5	8	6	4	7	2	8	1	8
7	128	7	128	7	128	7	128	7	128	7	128	7	128	7	128	
7		6		5		4		3		2		1		0		*Byte* (B_2)

Figura B1. *Chave da rotina de rotação de caracteres.*

O acumulador é carregado com o conteúdo do registo e, (N_3). Efectua-se um «E lógico» (AND) entre este valor e o *byte* cujo endereço está contido em hl. Se o resultado for zero, a rotina salta para «não activo», pois o *pixel*

endereçado por e e hl está inactivo. Se estiver activo, o acumulador é carregado com o valor actual do *byte* respectivo, (N_1). O registo b, (N_4), é adicionado ao acumulador e este é carregado em c. O registo é, então, ajustado de forma a apontar para o próximo *byte*, (B_1). Se o *byte* N_1 não estiver completo, a rotina salta, de volta para «próx *byte*».

hl é retirado da pilha, apontando de novo para o primeiro *byte* do carácter. bc é guardado na pilha, armazenando o valor do último *byte* a completar em c. O registo e é ajustado, de forma a endereçar o próximo *bit* de cada *byte*. Se a rotação não estiver completa, a rotina salta para «próx *bit*».

de é carregado com 7, valor adicionado a hl, para que hl aponte para o último *byte* que descreve o carácter. Para cada *byte*, copia-se o novo valor em e, sendo depois carregado no endereço apontado por hl. hl é decrementado, para apontar para o próximo *byte*, e o contador em b é decrementado. Se o contador ainda não tiver atingido o zero, a rotina salta para «substituir».

A rotina regressa, então, ao BASIC.

MUDANÇA DE ATRIBUTOS

Comprimento: 21
Número de variáveis: 2
Teste soma: 1952

Operação

Esta rotina altera os atributos de todos os caracteres no *écran*, de uma determinada forma. Por exemplo, altera a cor da «tinta» (INK), põe a piscar todo o *écran*, etc.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
bits salv	1	23296	*Bits* de atributo a não alterar
nov dados	1	23297	Novos *bits* a serem introduzidos nos atributos

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Cada *bit* dos atributos, individualmente, pode ser alterado, através do uso das instruções de código máquina «and» e «or».

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22528	33 0 88
	ld bc, 768	1 0 3
	ld de, (23296)	237 91 0 91
próx byte	ld a, (hl)	126
	and e	163
	or d	178
	ld (hl), a	119
	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, prox byte	32 246
	ret	201

Como funciona

O par de registos hl é carregado com o endereço da zona de atributos e o par de registos bc é carregado com o número de caracteres no *écran*. O registo d é carregado com o valor de «nov dados» e o registo e com o valor de «*bits* salv».

O acumulador é carregado com o *byte* endereçado por hl, e ajustados os seus *bits*, de acordo com os valores dos registos d e e. Devolve-se o resultado ao endereço de origem (hl). hl é incrementado, apontando o próximo *byte*, e o contador em bc é decrementado. Se o seu conteúdo não for nulo, a rotina salta, de novo para «próx *byte*».

A rotina regressa, então, ao BASIC.

TROCA DE ATRIBUTOS

Comprimento: 22
Número de variáveis: 2
Teste soma: 1825

Operação

Esta rotina varre a zona de atributos até encontrar um dado valor, substituindo cada ocorrência por um outro valor.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
valor ant.	1	23296	Vor a substituir
novo valor	1	23297	Novo valor do *byte* modificado

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é útil para sobreiluminar áreas de texto e de caracteres gráficos.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, 22528	33 0 88
	ld bc, 768	1 0 3
	ld de, (23296)	237 91 0 91
próx byte	ld a, (hl)	126
	cp e	187
	jr nz, não modif	32 1
	ld (hl), d	114
não modif	inc hl	35
	dec bc	11
	ld a,b	120
	or c	177
	jr nz, próx byte	32 245
	ret	201

Como funciona
O par de registos hl é carregado com o endereço da zona de atributos e bc carregado com o número de caracteres no *écran*. O registo e é carregado com «valor ant» e o registo d com «novo val».

O acumulador é carregado com o *byte* endereçado pelo par de registos hl. Se os conteúdos do acumulador e do registo e forem semelhantes o conteúdo do registo d é carregado no *byte* endereçado por hl. hl é incrementado, apontando para o próximo *byte*, e o contador em bc é decrementado. Se o seu valor não for nulo, a rotina salta, de novo para «próx *byte*».

A rotina regressa, então, ao BASIC.

PREENCHIMENTO DE ZONAS
Comprimento: 263
Número de variáveis: 2
Teste soma: 26647

Operação
Esta rotina preenche uma área do *écran*, limitada por uma linha de *pixels* ou pelo bordo do *écran*.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
coord x	1	23296	Coordenada x da posição inicial
coord y	1	23297	Coordenada y da posição inicial

Execução
RAND USR endereço

Verificação de erros
Nenhuma

Comentários
Esta rotina é não relocatável, sendo o seu endereço inicial 31955. Para copiá-la para um outro endereço, deve ser utilizado o método dado na rotina de «RENUMERAÇÃO». Se utilizarmos 31955 como endereço inicial desta rotina e 32218 como endereço inicial de «RENUMERAÇÃO», as rotinas podem coexistir em memória. Quando se preenchem zonas muito irregulares, pode ser necessária uma grande quantidade de RAM disponível. Se esta não existir, a rotina pode originar uma falha *(crash)*.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld a,h	124
	cp 176	254 176
	ret nc	208
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	ret nz	192
	ld bc, 65535	1 255 255
	push bc	197

direita	ld hl, (23296)	42 0 91
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	jr nz, esquerda	32 9
	ld hl, (23296)	42 0 91
	inc l	44
	ld (23296),hl	34 0 91
	jr nz, direita	32 236
esquerda	ld de, O	17 0 0
	ld hl, (23296)	42 0 91
	dec l	45
	ld (23296),hl	34 0 91
ponto	ld hl, (23296)	42 0 91
	push hl	229
	call sub-rotina	205 143* 125*
	or (hl)	182
	ld (hl),a	119
	pop hl	225
	ld a,h	124
	cp 175	254 175
	jr z, baixo	40 44
	ld a,e	123
	cp O	254 O
	jr nz, apagar	32 16
	inc h	36
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	jr nz, apagar	32 7
	ld hl, (23296)	42 0 91
	inc h	36
	push hl	229
	ld e,l	30 1
apagar	ld hl, (23296)	42 0 91
	ld a,e	123
	cp 1	254 1
	jr nz, baixo	32 15
	inc h	36
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O

	jr z, baixo	40 6
	ld e, O	30 0
	jr baixo	24 2
salto longo	jr direita	24 167
baixo	ld hl, (23296)	42 0 91
	ld a,h	124
	cp O	254 O
	jr z, próx pixel	40 40
	ld a,d	122
	cp O	254 O
	jr nz, restaurar	32 16
	dec h	37
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	jr nz, restaurar	32 7
	ld hl, (23296)	42 0 91
	dec h	37
	push hl	229
	ld d,1	22 1
restaurar	ld a,d	122
	cp 1	254 1
	jr nz, próx pixel	32 14
	ld hl, (23296)	42 0 91
	dec h	37
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	jr z, próx pixel	40 2
	ld d, O	22 O
próx pixel	ld hl, (23296)	42 O 91
	ld a,l	125
	cp O	254 O
	jr z, retirar	40 12
	dec l	45
	ld (23296),hl	34 0 91
	call sub-rotina	205 143* 125*
	and (hl)	166
	cp O	254 O
	jr z, ponto	40 129
retirar	pop hl	225

```
              ld (23296),hl      34 0 91
              ld a, 255          62 255
              cp h               188
              jr nz, salto longo 32 177
              cp l               189
              jr nz, salto longo 32 174
              ret                201
sub-rotina    push bc            197
              push de            213
              ld a, 175          62 175
              sub h              148
              ld h,a             103
              push hl            229
              and 7              230 7
              add a, 64          198 64
              ld c,a             79
              ld a,h             124
              rr a               203 31
              rr a               203 31
              rr a               203 31
              and 31             230 31
              ld b, a            71
              and 24             230 24
              ld d,a             87
              ld a,h             124
              and 192            230 192
              ld e,a             95
              ld h,c             97
              ld a,l             125
              rr a               203 31
              rr a               203 31
              rr a               203 31
              and 31             230 31
              ld l,a             111
              ld a,e             123
              add a,b            128
              sub d              146
              ld e,a             95
              ld d, O            22 O
              push hl            229
              push de            213
              pop hl             225
```



```
              add hl,hl          41
              add hl,hl          41
              add hl,hl          41
              add hl,hl          41
              add hl,hl          41
              pop de             209
              add hl,de          25
              pop de             209
              ld a,e             123
              and 7              230 7
              ld b, a            71
              ld a, 8            62 8
              sub b              144
              ld b,a             71
              ld a,l             62 1
rodar         add a,a            135
              djnz rodar         16 253
              rr a               203 31
              pop de             209
              pop bc             193
              ret                201
```

Como funciona

Esta rotina executa *plots* de linhas horizontais de *pixels* adjacentes, que se designam por «RUNS» no interior de áreas limitadas por *pixels* iluminados.

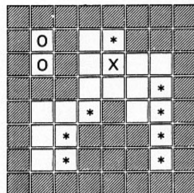

Figura B2. *Ilustração da técnica utilizada para preencher uma região. Os quadrados cinzentos estão já iluminados e definem a região a preencher. X é a posição de partida, * são inícios de RUNS e 0 ficarão não preenchidos.*

Cada RUN é memorizado pelo armazenamento das coordenadas do seu *pixel* da direita. Partindo das coordenadas especificadas, a rotina preenche cada RUN, anotando as posições de cada RUN não preenchido, acima ou abaixo. Depois de completado cada um dos RUNS, o último conjunto de coordenadas anotado é recuperado, preenchendo-se os correspondentes RUNS. O processo repete-se até não existirem mais RUNS por preencher.

A figura B2 ilustra esta técnica. Os quadrados representam *pixels* iluminados, x marca a posição de partida no interior da área a preencher e * marcam os *pixels* da direita dos RUNS.

A rotina preenche a linha horizontal que contém a posição de partida e guarda na pilha as posições de partida dos RUNS das linhas imediatamente acima e abaixo. De seguida, preenche a linha acima e a linha abaixo anotando, no segundo caso, que existem mais dois RUNS a partir dessa linha, e assim sucessivamente. Qualquer posição no interior da zona a preencher pode servir de posição de partida, mas note-se que os dois *pixels* assinalados com zeros não são alterados, pois ficam fora da área a preencher.

O registo h é carregado com a coordenada y especificada, e o registo l com a coordenada x. Se o valor da coordenada y for superior a 175, a rotina regressa ao BASIC. Invoca-se a «sub-rotina», devolvendo o endereço na memória do *bit* (x,y). Se este *bit* estiver activo, a rotina regressa ao BASIC.

O número 65535 é guardado na pilha, como marcação do primeiro valor a ser nele armazenado. Mais tarde, nessa rotina, cada número retirado da pilha será utilizado como par de coordenadas. Contudo, se esse número for 65535, a rotina regressa ao BASIC, pois já retirámos todos os números.

O registo h é carregado com a coordenada y e o registo l com a cordenada x. Invoca-se a sub-rotina devolvendo-nos em hl o endereço do *bit* (x, y). Se o *bit* estiver activo, salta para «esquerda». Se não, a coordenada x é incrementada e repete-se o ciclo a partir de «direita», enquanto x for menor que 256.

Em «esquerda», de é posicionado a zero. Os registos d e e serão utilizados como *flags*, d indicando «para baixo» e e indicando «para cima». A coordenada x é decrementada. Invoca-se a «sub-rotina» e desenha-se o ponto (x,y). Se a coordenada y for 175, a rotina salta para «baixo». Se a *flag* «para cima» estiver posicionada a um, salta para «apagar». Se o *bit* (x,y+1) estiver inactivo, os valores de x e de y+1 são guardados na pilha e posiciona-se a um a *flag* «para cima».

Em «apagar», se a *flag* «para cima» estiver a zero, salta «para baixo». Se o *bit* (x,y+1) estiver activo, a *flag* «para cima» posiciona-se a zero. Em «baixo», se a coordenada y for zero, salta para «próx *pixel*». Se a *flag* «para baixo» estiver a um, salta para «restaurar». Se o *bit* (x,y−1) estiver inactivo, os valores de x e de y−1 são guardados na pilha e posiciona-se a um a *flag* «para baixo».

Em «restaurar», se a *flag* «para baixo» estiver a zero, salta para «próx *pixel*». Se o *bit* (x,y−1) estiver activo, posiciona-se a zero a *flag* «para baixo». Em «próx *pixel*», se a coordenada x for zero a rotina salta para «retirar». A

coordenada x é decrementada e, se o novo *bit* (x,y) estiver inactivo, salta para «ponto». Em «retirar», retira-se da pilha um par de coordenadas (x,y). Se x e y forem ambos iguais a 255, a rotina regressa ao BASIC, pois a região está completamente preenchida. Caso contrário, salta, de novo para «direita».

A sub-rotina tem de calcular o endereço do *bit* (x,y) na memória. Em BASIC, este endereço seria:

16384+INT(Z/8)+256*(Z−8*INT(Z/8))
+32*(64*INT(Z/64)+INT(Z/8)−8*INT(Z/64))

onde Z=175−y

Os pares de registos bc e de são guardados na pilha. O acumulador é carregado com 175 e subtrai-se daí a coordenada y. O resultado é devolvido ao registo h. hl é guardado na pilha. Os cinco *bits* da esquerda do acumulador são posicionados a zero, adicionando-se-lhe depois 64. O resultado é copiado para o registo c. Quando multiplicado por 256, resulta em 16384+256*(Z−8*INT(Z/8)). O acumulador é carregado com Z e dividido por oito, sendo o resultado copiado para o registo b. Este resultado representa INT(Z/8). Posicionando os três *bits* da direita a zero, obtém-se o valor 8*INT(Z/64), que é carregado no registo d.

O acumulador é carregado com Z e posicionam-se a zero os seus seis *bits* da direita, produzindo o valor 64*INT(Z/64). Este valor é carregado no registo e. O valor no registo c é copiado para h. O acumulador é carregado com a coordenada x, que se divide por oito, indo o resultado para l.

O acumulador é então carregado com o valor em e, adicionando-se-lhe o conteúdo de b. Subtrai-se o valor de d e o resultado é carregado em de. Guarda-se na pilha o par de registos hl, sendo depois carregado com o valor de de. Multiplica-se este valor por 32, retira-se de da pilha e soma-se a hl. Assim, hl contém agora o endereço do *bit* (x,y).

O acumulador é carregado com o valor original de x. Posicionando a zero os cinco *bits* da esquerda, obtém-se o valor x−8*INT(x/8). O registo b é carregado com oito menos o valor no acumulador, para funcionar como contador. O acumulador é posicionado com um, que se multiplica por dois b−1 vezes.

Neste ponto o acumulador deve ter um só *bit* posicionado a um, que corresponde ao bit (x,y) endereçado por hl. de e bc são retirados da pilha e a sub-rotina regressa à rotina principal.

TABELAS DE DESENHO

Comprimento: 196
Número de variáveis: 2
Teste soma: 20278

Operação
Esta rotina desenha uma forma de qualquer tamanho no *écran*.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
X inicial	1	23 296	Coordenada X do primeiro *pixel*
Y inicial	1	23 297	coordenada Y do primeiro *pixel*

Execução
RAND USR endereço

Verificação de erros
Se A$ não existir, tiver comprimento nulo ou não contiver quaisquer parâmetros de forma, a rotina regressa imediatamente ao BASIC. O mesmo sucede se Y inicial for superior a 175.

Comentários
Método eficaz de armazenar formas em memória, de modo a serem rapidamente desenhadas no *écran*.

O método de utilização desta rotina é:

(i) LET A$= «parâmetros da forma»
(ii) POKE 23296, coordenada X do primeiro *pixel*
(iii) POKE 23297, coordenada Y do primeiro *pixel*
(iv) RAND USR endereço

Os parâmetros da forma são armazenados numa cadeia de caracteres *(string)*, com os seguintes significados:

«0» desenhar ponto
«5» decrementar coordenada X
«6» decrementar coordenada Y
«7» incrementar coordenada X
«8» incrementar coordenada Y

Quaisquer outros caracteres são ignorados.
A rotina permite à coordenada X sair do *écran*, pois prevê a continuação do desenho no lado oposto.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23627)	42 75 92
próx variável	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	bit 7,a	203 127
	jr nz, for next	32 23
	cp 96	254 96
	jr nc, número	48 11
	cp 65	254 65*
	jr z, encontrado	40 35
string	inc hl	35
	ld e, (hl)	94
	inc hl	35
	ld d, (hl)	86
somar	add hl,de	25
	jr incrementar	24 5
número	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
incrementar	inc hl	35
	jr próx variável	24 225
for next	cp 224	254 224
	jr c, próx bit	56 5
	ld de, 18	17 18 0
	jr add	24 236
próx bit	bit 5,a	203 111
	jr z, string	40 228
próx byte	inc hl	35
	bit 7,(hl)	203 126
	jr z, próx byte	40 251
	jr número	24 228
encontrado	inc hl	35
	ld c, (hl)	78
	inc hl	35
	ld b, (hl)	70
	inc hl	35

	ex de,hl	235
	ld a, (23297)	58 1 91
	cp 176	254 176
	ret nc	208
repetir	ld hl, (23296)	42 0 91
	ld a,b	120
	or c	177
	ret z	200
	dec bc	11
	ld a, (de)	26
	inc de	19
	cp 48	254 48
	jr nz, não desenhar	32 78
	push bc	197
	push de	213
	ld a, 175	62 175
	sub h	148
	ld h,a	103
	push hl	229
	and 7	230 7
	add a,64	198 64
	ld c,a	79
	ld a,h	124
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld b,a	71
	and 24	230 24
	ld d,a	87
	ld a,h	124
	and 192	230 192
	ld e,a	95
	ld h,c	97
	ld a,l	125
	rr a	203 31
	rr a	203 31
	rr a	203 31
	and 31	230 31
	ld l,a	111
	ld a,e	123
	add a,b	128

	sub d	146
	ld e,a	95
	ld d, O	22 O
	push hl	229
	push de	213
	pop hl	225
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	add hl,hl	41
	pop de	209
	add hl,de	25
	pop de	209
	ld a,e	123
	and 7	230 7
	ld b,a	71
	ld a, 8	62 8
	sub b	144
	ld b,a	71
	ld a,1	62 1
rodar	add a,a	135
	djnz rodar	16 253
	rr a	203 31
	pop de	209
	pop bc	193
	or (hl)	182
	ld (hl),a	119
aqui	jr repetir	24 165
não desenhar	cp 53	254 53
	jr nz, abaixo	32 1
	dec l	45
abaixo	cp 54	254 54
	jr nz, acima	32 8
	dec h	37
	ld a,h	124
	cp 255	254 255
	jr nz, guardar	32 19
	ld h, 175	38 175
acima	cp 55	254 55
	jr nz, direita	32 8

```
                inc h              36
                ld a,h             124
                cp 176             254 176
                jr nz, guardar     32 7
                ld h, O            38 O
direita         cp 56              254 56
                jr nz, guardar     32 1
                inc l              44
guardar         ld (23296),hl      34 0 91
                jr aqui            24 215
```

Como funciona

Acha-se o endereço da cadeia a$ através de uma adaptação ao primeiro troço da rotina «Instr $».

O comprimento da cadeia é carregado em bc e o endereço do primeiro carácter de A$ em de. Posiciona-se o acumulador com o valor de y inicial e, se este valor for superior a 175, a rotina regressa ao BASIC. O registo h é carregado com a coordenada Y, e l com a coordenada X. Se o conteúdo do par bc for nulo, a rotina regressa ao BASIC, pois atingimos o fim da cadeia. bc é decrementado, indicando que foi tratado mais um carácter. O próximo carácter é carregado no acumulador e de é incrementado, para apontar para o próximo *byte*. Se o acumulador não contiver 48, a rotina salta para «não desenhar». O ponto (X,Y) é desenhado através da «sub-rotina» da rotina de «Preenchimento de zonas». Depois disto, a rotina salta para executar, de novo, «repetir».

Em «não desenhar», se o conteúdo do acumulador for 53, a coordenada X é decrementada. Em «abaixo», se esse conteúdo não for 54, salta para «acima». A coordenada Y é decrementada e, se o resultado for −1, é posicionada com 175.

Em «acima», se o conteúdo do acumulador não for 55, salta para «direita». A coordenada Y é incrementada e, se o resultado for 176, posiciona-se a 0. Em «direita», se o conteúdo do acumulador for 56, a coordenada X é incrementada. Em «guardar», as coordenadas X e Y são carregadas na memória e a rotina salta para «aqui».

CÓPIA E AMPLIAÇÃO DO «ÉCRAN»

Comprimento: 335
Número de variáveis: 8
Teste soma: 33663

Operação

Esta rotina copia uma zona do *écran* para outra posição no mesmo *écran*, ampliando a cópia nos planos x ou y.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
y superior	1	23296	Coordenada y da linha limite superior
y inferior	1	23297	Coordenada y da linha limite inferior
x direita	1	23298	Coordenada x da coluna limite da direita
x esquerda	1	23299	Coordenada x da coluna limite da esquerda
esc horiz	1	23300	Ampliação no plano x
esc vertic	1	23301	Ampliação no plano y
novo x esq	1	23302	Coordenada x da coluna mais à esquerda da zona destino da cópia
novo y inf	1	23303	Coordenada y da linha inferior da zona destino da cópia

Execução

RAND USR endereço

Verificação de erros

A rotina regressa imediatamente ao BASIC, se alguma das seguintes condições se verificar:

(i) escala horizontal=0
(ii) escala vertical=0
(iii) y superior maior que 175
(iv) novo y inf maior que 175
(v) y inferior maior que y superior
(vi) x esquerda maior que x direita

No entanto, para não tornar esta rotina demasiado longa, não se faz qualquer verificação de que a cópia efectuada cabe no *écran*. Se tal não acontecer, a rotina pode provocar uma falha. O mesmo pode suceder se a memória disponível não for suficiente, pois esta rotina necessita de bastante memória disponível para trabalhar.

Comentários

Esta rotina é não relocatável, devido à existência de uma sub-rotina de *plot/point*. Está situada no endereço 65033, pelo que só pode ser usada nas máquinas com 48 k de RAM. A rotina pode ser reposicionada na memória, usando os procedimentos descritos para a rotina de «renumeração». No entanto, se se pretende copiar grandes zonas de *écran*, torna-se necessária a existência de bastante RAM disponível, pelo que o endereço inicial deve ser tão alto quanto possível.

Se se pretende que a área destino da cópia tenha o mesmo tamanho do original, devemos posicionar as escalas a um. Para duplicar o tamanho, posicionar as escalas a dois, para triplicar posicionar a três, etc.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld ix, 23296	221 33 0 91
	ld a, 175	62 175
	cp (ix + O)	221 190 0
	ret c	216
	cp (ix + 7)	221 190 7
	ret c	216
	sub a	151
	cp (ix + 4)	221 190 4
	ret z	200
	cp (ix + 5)	221 190 5
	ret z	200
	ld hl, (23296)	42 0 91
	ld b, 1	69
	ld a,l	125
	sub h	148
	ret c	216
	ld (23296),a	50 0 91
	ld e,a	95
	ld hl, (23298)	42 2 91
	ld c,l	77
	ld a,l	125
	sub h	148
	ret c	216
	ld (23298),a	50 2 91
	push bc	197
	ld l,a	111
	ld h, O	38 O
	inc hl	35
	push hl	229

	pop bc	193
	inc e	28
somar	dec e	29
	jr z, resto	40 3
	add hl,bc	9
	jr somar	24 250
resto	ld a,l	125
	and 15	230 15
	ld b,a	71
	pop hl	225
	ld c,l	77
	jr nz, guardar	32 2
cheio	ld b,16	6 16
guardar	push hl	229
	call sub-rotina	205 13* 255*
	and (hl)	166
	jr z, off	40 2
	ld a, 1	62 1
off	pop hl	225
	rr a	203 31
	rl e	203 19
	rl d	203 18
	ld a,l	125
	cp (ix + 3)	221 190 3
	jr z, prox coluna	40 6
	dec l	45
próx bit	djnz guarda	16 231
	push de	213
	jr cheio	24 226
próx coluna	ld l,c	105
	ld a,h	124
	cp (ix + 1)	221 190 1
	jr z, copiar	40 3
	dec h	37
	jr próx bit	24 241
copiar	push de	213
	ld b, O	6 O
	ld h,b	96
	ld l,b	104
apagar	ld (23306),hl	34 10 91
	ld a,b	120

	or a	183
	jr nz, retirar	32 3
	pop de	209
	ld b, 16	6 16
retirar	sub a	151
	dec b	5
	rr d	203 26
	rr e	203 27
	rl a	203 23
	push de	213
	push bc	197
	push af	245
	ld h,l	38 1
ciclo	ld l, l	46 1
manter	ld (23304),hl	34 8 91
	ld a, (23307)	58 11 91
	ld hl, O	33 0 0
	ld de, (23301)	237 91 5 91
	ld d,l	85
multiplicar	or a	183
	jr z, calcular	40 6
	add hl,de	25
	dec a	61
	jr multiplicar	24 249
salto longo	jr apagar	24 208
calcular	ld a, (23303)	58 7 91
	add a,l	133
	ld hl, (23304)	42 8 91
	add a,l	133
	dec a	61
	push af	245
	ld a, (23306)	58 10 91
	ld hl, O	33 0 0
	ld de, (23300)	237 91 4 91
	ld d, l	85
repetir	or a	183
	jr z, continuar	40 4
	add hl,de	25
	dec a	61
	jr repetir	24 249
continuar	ld a, (23302)	58 6 91

	add a,l	133
	ld hl, (23305)	42 9 91
	add a, l	133
	dec a	61
	ld l,a	111
	pop af	241
	ld h,a	103
	pop af	241
	push af	245
	or a	183
	jr nz, desenhar	32 7
	call sub-rotina	205 13* 255*
	cpl	47
	and (hl)	166
	jr Poke	24 4
desenhar	call sub-rotina	205 13* 255*
	or (hl)	182
Poke	ld (hl),a	119
	ld hl, (23304)	42 8 91
	inc l	44
	ld a, (23301)	58 5 91
	inc a	60
	cp l	189
	jr nz, manter	32 165
	inc h	36
	ld a, (23300)	58 4 91
	inc a	60
	cp h	188
	jr nz, ciclo	32 155
	pop af	241
	pop bc	193
	pop de	209
	ld hl, (23306)	42 10 91
	inc l	44
	ld a, (23298)	58 2 91
	inc a	60
	cp l	189
	jr nz, salto longo	32 164
	ld l, O	46 O
	inc h	36
	ld a, (23296)	58 0 91
	inc a	60

```
          cp h                   188
          jr nz, salto longo     32 154
          ret                    201
sub-rotina push bc               197
          push de                213
          ld a, 175              62 175
          sub h                  148
          ld h,a                 103
          push hl                229
          and 7                  230 7
          add a, 64              198 64
          ld c,a                 79
          ld a,h                 124
          rr a                   203 31
          rr a                   203 31
          rr a                   203 31
          and 31                 230 31
          ld b,a                 71
          and 24                 230 24
          ld d,a                 87
          ld a,h                 124
          and 192                230 192
          ld e,a                 95
          ld h, c                97
          ld a,l                 125
          rr a                   203 31
          rr a                   203 31
          rr a                   203 31
          and 31                 230 31
          ld l,a                 111
          ld a,e                 123
          add a,b                128
          sub d                  146
          ld e,a                 95
          ld d, O                22 O
          push hl                229
          push de                213
          pop hl                 225
          add hl,hl              41
          add hl,hl              41
          add hl,hl              41
          add hl,hl              41
```



```
          add hl,hl              41
          pop de                 209
          add hl,de              25
          pop de                 209
          ld a,e                 123
          and 7                  230 7
          ld b,a                 71
          ld a,8                 62 8
          sub b                  144
          ld b,a                 71
          ld a, 1                62 1
rodar     add a,a                135
          djnz rodar             16 253
          rr a                   203 31
          pop de                 209
          pop bc                 193
          ret                    201
```

Como funciona

ix é carregado com o endereço do *buffer* de impressão, para ser utilizado como ponteiro das variáveis. Se «y superior» ou «novo y inf» forem superiores a 175, a rotina regressa ao BASIC. O mesmo acontece se a escala horizontal ou a escala vertical forem zero.

O registo h é carregado com «y inferior» e o registo 1 com «y superior». Copia-se o registo 1 para o registo b e para o acumulador. Subtrai-se do acumulador o registo h e a rotina regressa ao BASIC se o resultado for negativo.

O valor acumulado é carregado no endereço 23298, para funcionar como contador. Guarda-se na pilha o par de registos bc.

O registo h1 é carregado com o valor contido no acumulador, incrementado e copiado para o registo bc. bc é somado «e» vezes a h1, sendo o resultado, em h1, igual ao número total de *pixels* a copiar. O acumulador é carregado com o valor do registo 1 e posicionam-se a zero os seus quatro *bits* da esquerda. O resultado copia-se para o registo b, para funcionar como contador.

Retira-se da pilha o par de registos h1 e copia-se o registo 1 para o registo c. Se o conteúdo do registo c for nulo, contém dezasseis, pois é este o número de *bits* de um par de registos. Invoca-se a «sub-rotina» e o acumulador é carregado com o valor PONTO (1,h). O par de registos de roda à esquerda, sendo o valor do acumulador introduzido no *bit* mais à direita do registo e.

Se o registo e for igual a «x esquerda», a rotina salta para «próx coluna». Caso contrário, o registo 1 é decrementado, seguido do registo b. Se o conteúdo do registo b for não nulo, guarda-se na pilha o par de registos e efectua-se um salto para «cheio».

Em «próx coluna», o registo l é carregado com «x direita» e o acumulador é carregado com o valor contido no registo h. Se o valor do acumulador for igual a «y inferior», salta para «copiar», pois o último *pixel* a copiar já foi carregado em de. No caso contrário, o registo h é decrementado, de forma a apontar para a próxima coluna e a rotina salta para «próx *bit*».

Em «copiar», guarda-se de na pilha e posicionam-se a zero os registos b, h e l, para uso como contadores. O conteúdo do par hl é carregado nos endereços 23306/7, para que hl também funcione como contador de ciclos mais longos, sem recorrência à pilha. Se o conteúdo de b for zero, retira-se da pilha e o registo b é reinicializado a dezasseis, indicando o número de *pixels* armazenados em de. O registo b é decrementado, indicando que vai ser retirado de de um *bit* de informação. O *bit* da direita do registo e é carregado no acumulador e o par de registos de roda à direita. de, bc e af são guardados na pilha, enquanto se efectuam alguns cálculos.

Os registos h e l estão ambos carregados com um, para uso como contadores, sendo hl carregado nos endereços 23304/5. O acumulador é carregado com o valor do *byte* situado no endereço 23307, que é um dos contadores que aqui foram armazenados. O registo de é carregado com a escala vertical. Este valor é depois multiplicado pelo valor no acumulador, sendo o resultado guardado em hl. Este resultado soma-se a «novo y inf», no acumulador. O *byte* situado no endereço 23304 soma-se ao acumulador e o resultado é decrementado.

O acumulador contém agora a coordenada y do próximo *pixel* a ser desenhado. Este valor é guardado na pilha, enquanto se calcula a coordenada x por um processo muito semelhante ao descrito. Depois de calculada, é carregada no registo l. A coordenada y, retirada da pilha, é carregada no registo h. Posiciona-se o acumulador com o último valor contido. Se for um, o ponto (x,y) deve ser desenhado, devendo ser «apagado» no caso contrário. Invoca-se a sub-rotina e desenrola-se a acção apropriada.

O par de registos hl é carregado com os contadores de ciclos, armazenados nos endereços 23304/5. O registo l é incrementado e, se o seu conteúdo não for igual a (1+escala vertical), a rotina salta de volta para «manter». O registo h é incrementado e, se o seu conteúdo não for igual a (1+escala horizontal), salta para «ciclo».

Retiram-se da pilha os registos af, bc e de e o par de registos hl é carregado com o segundo conjunto de contadores de ciclos, que estão armazenados nos endereços 23306/7. O registo l é incrementado e é efectuado um salto para «apagar», se o resultado não for igual a (x direita−x esquerda+1). Posiciona-se a zero o registo l, sendo esse o valor inicial do contador de ciclos. O registo l é, então, incrementado e a rotina salta de volta para «apagar», se o resultado não for igual a (y superior−y inferior+1).

A rotina regressa, então, ao BASIC.

A «sub-rotina» é idêntica à utilizada pela rotina de «Preenchimento de zonas».

7. Rotinas de manipulação de «écran»

ELIMINAÇÃO DE BLOCOS DE PROGRAMA

Comprimento: 42
Número de variáveis: 2
Teste soma: 5977

Operação

Esta rotina elimina blocos de programa BASIC, entre duas linhas especificadas pelo utilizador.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
linha inic	2	23296	Primeira linha a ser apagada
linha fim	2	23298	Última linha a ser apagada

Execução

RAND USR endereço

Verificação de erros

Se for detectado algum dos seguintes erros, a rotina pára, sem ter apagado qualquer das linhas do BASIC:

(i) Último número de linha menor que o primeiro;
(ii) não existem instruções de BASIC entre as duas linhas dadas;
(iii) uma das linhas especificadas, ou ambas, é zero.

Comentários

Esta rotina é bastante lenta, quando se pretende eliminar um grande troço de programa, mas, mesmo assim, é muito mais rápido e cómodo utilizá-la do que eliminar as linhas manualmente. Não devem ser introduzidos números de linha maiores que 9999.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91

Label	Assembler	Números a introduzir
	ld a,h	124
	or l	181
	ret z	200
	ld a,d	122
	or e	179
	ret z	200
	push de	213
	call 6510	205 110 25
	ex (sp), hl	227
	inc hl	35
	call 6510	205 110 25
	pop de	209
	and a	167
	sbc hl,de	237 82
	ret z	200
	ret c	216
	ex de,hl	235
próx caráct	ld a,d	122
	or e	179
	ret z	200
	push de	213
	push hl	229
	call 4120	205 24 16
	pop hl	225
	pop de	209
	dec de	27
	jr próx caráct	24 243

Como funciona

Os pares de registos hl e de são carregados com os números de linha inicial e final, respectivamente. Verificam-se os seus valores e, se um deles ou ambos é, ou são, zero, a rotina regressa ao BASIC.

Chama-se a rotina da ROM situada a partir do endereço 6510, devolvendo-nos o endereço da primeira linha. Chama-se de novo, desta vez para determinar o endereço do carácter que se segue ao «ENTER», na linha final. O par de registos h1 é carregado com a diferença entre estes dois endereços e, se essa diferença for igual ou inferior a zero, a rotina regressa ao BASIC.

Copia-se para de o conteúdo do par de registos h1, para servir como contador. Quando o contador atinge o zero, a rotina termina. Enquanto não atinge este valor, chama-se a rotina da ROM situada a partir de 4210, que elimina um carácter. A rotina salta, de volta para «próx caráct».

TROCA DE CARÁCTER

Comprimento: 46
Número de variáveis: 2
Teste soma: 5000

Operação

Troca todas as ocorrências de um carácter especificado, num programa BASIC, por outro carácter escolhido pelo utilizador. Por exemplo, todas as instruções PRINT podem ser substituídas por LPRINT.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
car antigo	1	23296	carácter a substituir
novo car	1	23297	carácter a introduzir

Execução

RAND USR endereço

Verificação de erros

Se não existir em memória qualquer programa BASIC ou se algum dos caracteres especificados tiver código inferior a 32, a rotina regressa ao BASIC.

Comentários

Esta rotina é bastante rápida mas, como é óbvio, quanto maior for o programa BASIC mais tempo demora a sua execução.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld, bc, (23296)	237 75 0 91
	ld a,31	62 31
	cp b	184
	ret nc	208
	cp c	185
	ret nc	208
	ld hl, (23635)	42 83 92
próx caráct	inc hl	35
	inc hl	35
	inc hl	35
verificar	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	ret nc	208

	add hl,de	25
	inc hl	35
	ld a, (hl)	126
	inc hl	35
	cp 13	254 13
	jr z, próx caráct	40 237
	cp 14	254 14
	jr nz, comparar	32 3
	inc hl	35
	jr prox caráct	24 230
comparar	dec hl	43
	cp c	185
	jr nz, verificar	32 229
	ld (hl), b	112
	jr verificar	24 226

Como funciona

Os registos b e c são carregados, respectivamente, com o antigo e o novo carácter. Se algum dos caracteres tiver código inferior a 32, a rotina regressa ao BASIC.

O par de registos hl é carregado com o endereço inicial do programa BASIC. Este par é depois incrementado e comparado com o endereço da zona de variáveis. Se o conteúdo de hl não for menor que esse endereço, a rotina regressa ao BASIC.

O par hl é incrementado, apontando para o próximo carácter. O código deste carácter é carregado no acumulador e hl é incrementado de novo. Se o conteúdo do acumulador for 13 ou 14 (ENTER ou NUMBER), a rotina salta, de volta para «próx caráct», e hl é incrementado, apontando para o próximo carácter. Se o conteúdo do acumulador não for 13 ou 14, é comparado com «car antigo». Se for igual, este carácter é substituído por «novo car».

A rotina salta para «verificar», testando se já foi atingido o fim do programa.

ELIMINAÇÃO DE REMS

Comprimento: 132
Número de variáveis: 0
Teste soma: 13809

Operação

Esta rotina elimina todas as instruções «REM» do programa BASIC presentemente em memória.

Execução

RAND USR endereço

Verificação de erros

Se não existir em memória nenhum programa BASIC, a rotina não tem qualquer efeito.

Comentários

A rotina da ROM, utilizada para eliminar caracteres, não é muito rápida, pelo que a execução desta rotina pode demorar algum tempo.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23635)	42 83 92
	jr verificar	24 31
próx linha	push hl	229
	inc hl	35
	inc hl	35
	ld c, (hl)	78
	inc hl	35
	ld b, (hl)	70
próx caráct	inc hl	35
	ld a,(hl)	126
	cp 33	254 33
	jr c, próx caráct	56 250
	cp 234	254 234
	jr nz, procurar	32 26
	inc bc	3
	inc bc	3
	inc bc	3
	inc bc	3
	pop hl	225
elimin linha	push bc	197
	call 4120	205 24 16
	pop bc	193
	dec bc	11
	ld a,b	120
	or c	177

```
                jr nz, elimin linha      32 246
verificar       ld de, (23627)           237 91 75 92
                and a                    167
                sbc hl,de                237 82
                ret nc                   208
                add hl,de                25
                jr próx linha            24 214
procurar        inc hl                   35
                ld a, (hl)               126
                cp 13                    254 13
                jr nz, não enter         32 8
enter achado    pop hl                   225
                add hl,bc                9
                inc hl                   35
                inc hl                   35
                inc hl                   35
                inc hl                   35
                jr verificar             24 231
não enter       cp 14                    254 14
                jr nz, não número        32 7
                inc hl                   35
                inc hl                   35
                inc hl                   35
                inc hl                   35
                inc hl                   35
                jr procurar              24 231
não número      cp 33                    254 33
                jr c, procurar           56 227
                cp 34                    254 34
                jr nz, não aspas         32 8
achar aspas     inc hl                   35
                ld a, (hl)               126
                cp 34                    254 34
                jr nz, achar aspas       32 250
                jr procurar              24 215
não aspas       cp 58                    254 58
                jr nz, procurar          32 211
                ld d,h                   84
                ld e, l                  93
achar enter     inc hl                   35
                ld a, (hl)               126
```



```
                cp 13                    254 13
                jr z, enter achado       40 209
                cp 33                    254 33
                jr c, achar enter        56 246
                cp 234                   254 234
                jr nz, não aspas         32 236
                ld h,d                   98
                ld l,e                   107
elimin caráct   push bc                  197
                call 4120                205 24 16
                pop bc                   193
                dec bc                   11
                ld a, (hl)               126
                cp 13                    254 13
                jr nz, elimin caract     32 245
                pop hl                   225
                inc hl                   35
                inc hl                   35
                ld (hl),c                113
                inc hl                   35
                ld (hl),b                112
                dec hl                   43
                dec hl                   43
                dec hl                   43
                jr verificar             24 160
```

Como funciona

O par de registos hl é carregado com o endereço inicial da zona de programas BASIC, executando-se um salto para a rotina que verifica se já atingimos o fim da zona de programas. No caso afirmativo, a rotina regressa ao BASIC.

A rotina salta para «prox linha». Esta secção guarda na pilha o endereço contido em hl, para utilização posterior, carregando em bc o comprimento da linha de BASIC encontrada. A rotina «próx caract» incrementa o endereço contido em hl e carrega o acumulador com o carácter contido nesse endereço. Se esse carácter tiver código inferior a 33, indicando espaço ou carácter de controle, a rotina salta para nova execução deste troço. Se o carácter encontrado não for «REM», salta para «procurar».

Se foi encontrado um «REM», o registo bc é incrementado quatro vezes, para ser usado como contador, e retira-se hl do topo da pilha. Eliminam-se bc caracteres a partir do endereço hl, através da rotina da ROM situada no endereço 4120. A rotina vai então executar de novo o troço «verificar».

Quando salta para a rotina «procurar», hl é incrementado, apontando para o próximo carácter, que é carregado no acumulador. Se for um carácter «ENTER», hl é retirado da pilha, incrementado até apontar o início da próxima instrução e efectua-se um salto para «verificar».

Se o conteúdo do acumulador for o carácter «NUMBER» (14), hl é incrementado até apontar o primeiro carácter após o número e repete-se o processo de busca.

Efectua-se um teste para verificar se existem caracteres de código inferior a 33 e, se for encontrado algum, a rotina salta, de volta para «procurar». Se for encontrado um carácter de aspas (34), a rotina executa um ciclo de busca de um segundo carácter de aspas, após o que continua a busca interrompida. Se o carácter encontrado não for «dois pontos», indicando uma linha multi-instrução, a busca é repetida. Copia-se então hl para de, a fim de ser armazenado o endereço dos «dois pontos» e hl é incrementado, apontando para o próximo carácter. Se for um carácter de «ENTER», salta para «*enter* achado». Caso contrário, se for um caracter de controle ou espaço, a rotina volta para «achar *enter*».

Se o carácter não for um «REM», efectua-se um salto para «não aspas». Achando-se um «REM», hl é carregado com o endereço dos últimos «dois pontos» encontrados, sendo de seguida eliminados todos os caracteres, a partir de hl e até ao próximo «ENTER». Corrigem-se os ponteiros de linha, posiciona-se hl com o início da linha e a rotina salta para «verificar».

CRIAÇÃO DE REMS

Comprimento: 85
Número de variáveis: 3
Teste soma: 9526

Operação

Esta rotina cria uma instrução «REM» numa dada linha, contendo um determinado número de caracteres. O carácter é escolhido pelo utilizador.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
num linha	2	23296	Linha onde se pretende inserir o «REM»
num caráct	2	23298	Número de caracteres a introduzir depois do «REM»
cod caráct	1	23300	Código dos caracteres a introduzir

Execução

RAND USR endereço

Verificação de erros

Se o número de linha for zero ou superior a 9999, ou ainda se já existir no programa uma linha com esse número, a rotina regressa ao BASIC.

Comentários

Esta rotina não verifica se existe memória disponível para a inserção de novas instruções. Essa existência deve ser verificada, antes de executarmos a rotina em si, invocando a rotina «Memória disponível», apresentada mais à frente.

Os caracteres a introduzir devem, preferencialmente, ter códigos superiores a 31, pois os caracteres de controle (0-31) podem confundir a rotina da ROM que produz as listagens de BASIC (LIST).

A rotina da ROM chamada para inserir os caracteres é lenta, pelo que a execução desta rotina pode demorar bastante tempo.

A instrução de «REM» criada através desta rotina serve para armazenar código máquina ou dados, bastando para isso carregá-los no endereço correcto.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld, a,h	124
	or l	181
	ret z	200
	ld de, 10000	17 16 39
	and a	167
	sbc hl,de	237 82
	ret nc	208
	add hl,de	25
	push hl	229
	call 6510	205 110 25
	jr nz, criar	32 2
	pop hl	225
	ret	201
criar	ld bc, (23298)	237 75 2 91
	push bc	197
	push bc	197
	ld a, 13	62 13
	call 3976	205 136 15
	inc hl	35

	pop bc	193
próx caráct	push bc	197
	ld a,b	120
	or c	177
	jr z, inserir REM	40 11
	ld a, (23300)	58 4 91
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	dec bc	11
	jr próx caráct	24 240
inserir REM	pop bc	193
	ld a, 234	62 234
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	inc bc	3
	inc bc	3
	ld a,b	120
	push bc	197
	call 3976	205 136 15
	pop bc	193
	inc hl	35
	ld a,c	121
	call 3976	205 136 15
	inc hl	35
	pop bc	193
	ld a,c	121
	push bc	197
	call 3976	205 136 15
	pop bc	193
	inc hl	35
	ld a,b	120
	jp 3976	195 136 15

Como funciona

O par de registos hl é carregado com o número de linha especificado. Compara-se este número com zero e, se for igual, a rotina regressa ao BASIC. Do mesmo modo, se hl contém um número superior a 9999 (o maior número de linha possível), regressa-se ao BASIC.

Chama-se uma rotina da ROM que devolve, em hl, o endereço da linha cujo

número foi enviado em hl. Se a *flag* de zero estiver activa, a linha já existe, pelo que a rotina regressa ao BASIC.

Se a *flag* de zero não estiver activa, a rotina salta para «criar». bc é carregado com o número de caracteres a inserir após o «REM», sendo este número guardado na pilha. O acumulador é carregado com 13, o código de carácter de «ENTER». Chama-se a rotina da ROM situada no endereço 3976 para inserir este carácter. Retira-se da pilha o registo bc. Depois de bc ser de novo guardado na pilha, testa-se o seu conteúdo, verificando-se se há mais caracteres a inserir. Se não, efectua-se um salto para «inserir REM». Caso haja mais algum carácter para inserir, o acumulador é carregado com o respectivo código e chama-se de novo a rotina no endereço 3976 da ROM. O contador em bc é decrementado e a rotina volta a testar se bc é zero. Quando a rotina passa a «inserir REM», é inserido um código de «REM», através da mesma rotina da ROM. bc é carregado com o comprimento da nova linha, criando-se os respectivos ponteiros. O número de linha é retirado da pilha e, finalmente, inserido, antes do retorno ao BASIC.

COMPACTAÇÃO DE PROGRAMAS

Comprimento: 71
Número de variáveis: 0
Teste soma: 7158

Operação

Esta rotina elimina todos os caracteres de controle e espaços desnecessários num programa BASIC, aumentando assim a quantidade de RAM disponível.

Execução

RAND USR endereço

Verificação de erros

Se não existir em memória qualquer programa BASIC, a rotina regressa, de imediato, ao BASIC.

Comentários

Esta rotina assume que todas as instruções de REM foram já retiradas do programa BASIC. No entanto, caso tal não tenha acontecido, o computador não entra em *crash*. O tempo de execução desta rotina é proporcional ao tamanho do programa BASIC.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23635)	42 83 92
próx linha	inc hl	35

```
            inc hl                 35
verificar   ld de, (23627)         237 91 75 92
            and a                  167
            sbc hl,de              237 82
            ret nc                 208
            add hl,de              25
comprimento push hl                229
            ld c, (hl)             78
            inc hl                 35
            ld b, (hl)             70
próx byte   inc hl                 35
carregar    ld a, (hl)             126
            cp 13                  254 13
            jr nz, número          32 8
restaurar   pop hl                 225
            ld (hl),c              113
            inc hl                 35
            ld (hl),b              112
            add hl,bc              9
            inc hl                 35
            jr próx linha          24 227
número      cp 14                  254 14
            jr nz, aspas           32 7
            inc hl                 35
            inc hl                 35
            inc hl                 35
            inc hl                 35
            inc hl                 35
            jr próx byte           24 231
aspas       cp 34                  254 34
            jr nz, controle        32 12
achar aspas inc hl                 35
            ld a, (hl)             126
            cp 34                  254 34
            jr z, próx byte        40 221
            cp 13                  254 13
            jr z, restaurar        40 223
            jr achar aspas         24 244
controle    cp 33                  254 33
            jr nc, próx byte       48 211
            push bc                197
```



```
            call 4120              205 24 16
            pop bc                 193
            dec bc                 11
            jr carregar            24 204
```

Como funciona

O par de registos hl é carregado com o endereço do programa BASIC. hl é incrementado duas vezes, de forma a apontar para os dois *bytes* que contêm o comprimento da linha. O par de registos de é carregado com o endereço da zona de variáveis. Se este endereço for menor ou igual ao endereço do programa, a rotina regressa ao BASIC, uma vez que atingimos o fim da zona de programas.

Guarda-se na pilha o endereço em hl, bc é carregado com o comprimento actual da linha e hl é incrementado, de forma a apontar para o próximo *byte*. O *byte* em hl é carregado no acumulador. Se o conteúdo do acumulador não for treze, salta para «número».

Torna-se necessário, para cada passagem no troço «restaurar», que tenha sido encontrado o fim da presente linha. O endereço dos «ponteiros» de linha é carregado, a partir da pilha, em hl, sendo o comprimento actual nele inserido. O comprimento de linha é somado a hl, este é incrementado e a rotina salta para «próx linha».

Se a rotina atinge o troço «número», testa-se o acumulador, verificando-se se contém o código de carácter de NUMBER (14). No caso afirmativo, hl é incrementado cinco vezes, para que o número não seja alterado, e a rotina salta para «próx *byte*».

Se o acumulador não contém o código do carácter «aspas», a rotina salta para «controle». Se forem encontradas aspas, a rotina executa um ciclo até encontrar o fim da linha, ou outras aspas. No primeiro caso, salta para «restaurar» e no segundo para «próx *byte*».

Em «controle», o caracter é testado, para se verificar se contém um código inferior a treze. Se não, a rotina salta, de volta para «próx *byte*».

Se for encontrado um espaço ou um carácter de controle, chama-se a rotina do endereço 4120 da ROM, para o eliminar. O comprimento da linha, armazenado em bc, é decrementado e efectua-se um salto para «carregar».

CARREGAMENTO DE CÓDIGO MÁQUINA EM INSTRUÇÕES DATA

Comprimento: 179
Número de variáveis: 2
Teste soma: 19181

Operação
Esta rotina produz uma instrução DATA na primeira linha de um programa BASIC, preenchendo-a com dados retirados da memória.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
inic dados	2	23296	Endereço inicial dos dados
comp dados	2	23298	Número de *bytes* a copiar

Execução
RAND USR endereço

Verificação de erros
Se o número de *bytes* a copiar for zero ou já existir uma linha um no programa, a rotina regressa, de imediato, ao BASIC. A rotina não verifica se existe memória disponível em quantidade suficiente para armazenar a nova linha, o que terá que ser feito pelo utilizador.

A rotina necessita de dez *bytes* por cada *byte* de dados, mais cinco para ponteiros e números de linha, etc. A rotina da ROM invocada utiliza grandes zonas de trabalho em memória, o que deve também ser levado em conta. Se não houver memória em quantidade suficiente, os ponteiros de linha não serão correctamente posicionados, o que deturpa completamente as listagens do programa BASIC.

Comentários
O tempo de execução desta rotina é proporcional ao comprimento da zona de memória a ser copiada.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld de, (23296)	237 91 0 91
	ld bc, (23298)	237 75 2 91
	ld a,b	120
	or c	177
	ret z	200
	ld hl, (23635)	42 83 92
	ld a, (hl)	126
	cp O	254 0
	jr nz, continuar	32 6
	inc hl	35
	ld a, (hl)	126
	cp 1	254 1
	ret z	200

Label	*Assembler*	*Números a introduzir*
	dec hl	43
continuar	push hl	229
	push bc	197
	push de	213
	sub a	151
	call 3976	205 136 15
	ex de,hl	235
	ld a, l	62 1
	call 3976	205 136 15
	ex de,hl	235
	call 3976	205 136 15
	ex de,hl	235
	call 3976	205 136 15
	ex de,hl	235
	ld a, 228	62 228
	call 3976	205 136 15
	ex de,hl	235
próx byte	pop de	209
	ld a, (de)	26
	push de	213
	ld c,47	14 47
centenas	inc c	12
	ld b, 100	6 100
	sub b	144
	jr nc, centenas	48 250
	add a,b	128
	ld b,a	71
	ld a,c	121
	push bc	197
	call 3976	205 136 15
	ex de,hl	235
	pop bc	193
	ld a,b	120
	ld c,47	14 47
dezenas	inc c	12
	ld b,10	6 10
	sub b	144
	jr nc, dezenas	48 250
	add a,b	128
	ld b,a	71
	ld a,c	121

```
               push bc             197
               call 3976           205 136 15
               pop bc              193
               ex de,hl            235
               ld a,b              120
               add a,48            198 48
               call 3976           205 136 15
               ex de,hl            235
               ld a,14             62 14
               ld b, 6             6 6
próx zero      push bc             197
               call 3976           205 136 15
               pop bc              193
               ex de,hl            235
               sub a               151
               djnz próx zero      16 247
               pop de              209
               push hl             229
               dec hl              43
               dec hl              43
               dec hl              43
               ld a, (de)          26
               ld (hl),a           119
               pop hl              225
               inc de              19
               pop bc              193
               dec bc              11
               ld a,b              120
               or c                177
               jr z, enter         40 10
               push bc             197
               push de             213
               ld a, 44            62 44
               call 3976           205 136 15
               ex de, hl           235
               jr prox byte        24 173
enter          ld a,13             62 13
               call 3976           205 136 15
               pop hl              225
               ld bc, 0            1 0 0
               inc hl              35
               inc hl              35
```



```
               ld d,h              84
               ld e,l              93
               inc hl              35
ponteiros      inc hl              35
               inc bc              3
               ld a, (hl)          126
               cp 14               254 14
               jr nz, fim?         32 12
               inc bc              3
               inc bc              3
               inc bc              3
               inc bc              3
               inc bc              3
               inc hl              35
               inc hl              35
               inc hl              35
               inc hl              35
               inc hl              35
               jr ponteiros        24 237
fim?           cp 13               254 13
               jr nz, ponteiros    32 233
               ld a,c              121
               ld (de),a           18
               inc de              19
               ld a,b              120
               ld (de),a           18
               ret                 201
```

Como funciona

O par de registos de é carregado com o endereço dos *bytes* a copiar e o par de registos bc é carregado com o número desses *bytes*. Se o conteúdo de bc é nulo, a rotina regressa, de imediato, ao BASIC.

O par hl é carregado com o endereço do programa BASIC. O acumulador é carregado com o *byte* de endereço apontado por hl. Este *byte* representa o *byte* mais alto do número de linha. Se não for nulo, a linha um não existe ainda e a rotina salta para «continuar». Se o *byte* mais alto for nulo, o acumulador é carregado com o *byte* mais baixo. Se estiver posicionado a um, a linha um já existe, pelo que a rotina regressa ao BASIC.

O endereço do *byte* mais alto do número de linha é guardado na pilha. É também guardado na pilha o número de *bytes* a copiar, seguido do endereço dos dados.

O acumulador é carregado com zero, que é o valor do *byte* mais alto do novo número de linha. Invocando a rotina no endereço 3976 da ROM, insere-se o carácter contido no acumulador no endereço apontado por hl. Reposiciona-se hl com o valor que continha antes desta operação. O acumulador é carregado com um, valor que se insere três vezes. Na primeira vez posiciona-se o *byte* mais baixo do número de linha, posicionando-se o ponteiro de linha nas restantes duas. O acumulador é carregado com o código de carácter de «DATA», que é inserido.

O endereço do próximo *byte* de dados é retirado da pilha e carregado em de. O acumulador é carregado com este *byte* e de é de novo colocado na pilha. O registo c é carregado com um menos o código de carácter de «0». Este registo é incrementado e o registo b é carregado com 100. Subtrai-se o registo b do acumulador e, se o resultador não for negativo, a rotina salta, de volta para «centenas».

Soma-se uma vez o registo b ao acumulador, para que este contenha um valor positivo, valor este que é devolvido ao registo b. O acumulador é carregado com o conteúdo de c e bc é guardado na pilha. Chama-se a rotina situada no endereço 3976 da ROM, que mais uma vez insere o carácter contido no acumulador no endereço apontado por hl. O par bc é retirado da pilha e o acumulador é carregado com o valor do registo b. O processo acima descrito repete-se para b=10. O acumulador é, então, incrementado de 48 e inserido o carácter resultante.

A rotina acima introduziu o valor decimal do *byte* de dados encontrado na instrução DATA. Deve inserir-se agora a representação binária. É assinalada pelo carácter de «NUMBER» (14) inserido em primeiro lugar, seguido de cinco zeros. O valor do *byte* a ser copiado vai substituir o terceiro zero. de é incrementado, apontando para o próximo *byte* de dados. Retira-se da pilha para bc o número de *bytes* a copiar, decrementando-se bc. Se o resultado do decremento for zero, a rotina salta para *enter*. Caso contrário, os pares bc e de são de novo carregados na pilha, insere-se uma vírgula e a rotina salta, de volta para «próx *byte*».

Em *enter*, introduz-se o código de carácter de ENTER, marcando o fim da instrução de DATA. hl é carregado com o endereço inicial da linha e posiciona-se bc a zero. hl é incrementado, de forma a apontar para o menor *byte* do ponteiro de linha, e copia-se este novo endereço para de. hl é incrementado, de forma a apontar para o maior *byte* do ponteiro de linha. hl e bc são, então, incrementados e o acumulador é carregado com o *byte* situado no endereço apontado por hl.

Sendo 14 o conteúdo do acumulador, foi encontrado um número, pelo que hl e bc são incrementados cinco vezes, para apontarem o primeiro carácter após o número, e a rotina salta, de volta para «ponteiros».

Se o acumulador não contém 14 nem 13, salta para «ponteiros».

Ao chegarmos a este ponto, deve já ter sido encontrado o carácter de ENTER que marca o fim de linha. bc contém agora o comprimento da linha,

que é carregado no respectivo ponteiro, cujo endereço está em de.

A rotina regressa, então, a BASIC.

CONVERSÃO MINÚSCULAS/MAIÚSCULAS

Comprimento: 41
Número de variáveis: 0
Teste soma: 4685

Operação

Esta rotina converte todas as letras minúsculas num programa BASIC em maiúsculas ou vice-versa.

Execução

RAND USR endereço

Verificação de erros

Se não existir qualquer programa BASIC em memória, a rotina regressa imediatamente ao BASIC.

Comentários

Para alterar esta rotina, de modo a que ela converta maiúsculas em minúsculas, devem ser alterados os seguintes números na terceira coluna:

97* para 65
91** para 123

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23635)	42 83 92
	ld de, (23627)	237 91 75 92
saltar	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
alterado	inc hl	35
próx byte	and a	167
	sbc hl,de	237 82
	ret nc	208
	add hl,de	25

ld a, (hl)	126
cp 13	254 13
jr z, saltar	40 241
cp 14	254 14
inc hl	35
jr z, saltar	40 236
sub 97	214 97*
jr c, próx byte	56 237
sub 26	214 26
jr nc, próx byte	48 233
add a,91	198 91*
dec hl	43
ld (hl), a	119
jr alterado	24 226

Como funciona

O par de registos hl é carregado com o endereço do programa BASIC e de é carregado com o endereço da zona de variáveis. hl é incrementado, saltando por cima do número e ponteiros de linha. Se hl não for menor que de, a rotina regressa ao BASIC, pois foi atingido o fim do programa.

O acumulador é carregado com o *byte* armazenado em hl. Se este *byte* for um código de carácter «ENTER», a rotina salta, de volta para «saltar». Se for um código de «NUMBER», a rotina salta também para «saltar», tendo já sido incrementado hl. Evitam-se, assim, os cinco *bytes* após o carácter 14.

Subtrai-se 96 ao acumulador. Se o resultado for negativo, a rotina salta para «próx *byte*», porque o carácter não pode ser uma letra minúscula. Subtrai-se, ainda, 26 ao acumulador. Se o resultado não for negativo, salta para «próx *byte*», pois o carácter tem um código demasiado elevado para poder ser uma letra minúscula. Soma-se 90 ao acumulador, o que dá como resultado o código da respectiva letra maiúscula. hl é decrementado, apontando para o carácter a substituir. Este endereço é carregado com o conteúdo do acumulador e a rotina salta para «alterado».

8. Rotinas utilitárias

RENUMERAÇÃO

Comprimento: 382
Número de variáveis: 2
Teste soma: 41629

Operação

Esta rotina renumera um programa BASIC, alterando GOTOs, GOSUBs, etc.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
nº pr lin	2	23296	Número pretendido para a primeira linha do programa
passo	2	23298	Diferença entre números de linha consecutivos.

Execução

RAND USR endereço

Verificação de erros

Se o número da primeira linha ou o passo forem zero, a rotina retorna imediatamente ao BASIC. O mesmo sucede se não existir na RAM nenhum programa BASIC. Expressões numéricas (p. ex., GOTO 7*A), números com ponto decimal (p. ex., GOTO 7.8), números negativos (p. ex., GOTO −1) ou números superiores a 9999 (p. ex., GOTO 20170) são ignorados. Se o passo for excessivamente grande podem surgir números de linha repetidos, inutilizando o programa. A rotina aumenta o comprimento na RAM do programa, pelo que devemos previamente verificar se existe algum espaço livre na memória.

Comentários

O tempo de execução desta rotina é proporcional ao tamanho do programa BASIC.

A rotina é não relocatável, devendo ser introduzida a partir do endereço 32218. O seguinte algoritmo altera este endereço.

(i) X = novo endereço − 32218
(ii) H = INT(X/256)
L = X − 256*H

(iii) Para cada par de números assinalados com «*» na listagem:

LI = L + primeiro número
HI = H + segundo número

Se LI maior que 255, deve-se fazer:

HI = HI + 1
LI = LI − 256

Basta-nos finalmente substituir cada par de números pelos correspondentes LI e HI.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld a, h	124
	or L	181
	ret z	200
	ld hl, (23298)	42 2 91
	ld a,h	124
	or L	181
	ret z	200
	ld hl, (23635)	42 83 92
	ld de, (23296)	237 91 0 91
próx linha	call verificar	205 76* 127*
	jr nc, procurar GOTO	48 22
	ld b, (hl)	70
	ld (hl),d	114
	inc hl	35
	ld c, (hl)	78
	ld (hl),e	115
	inc hl	35
	ld (hl),c	113
	inc hl	35
	ld (hl),b	112
	inc hl	35
	push hl	229
	ld hl, (23298)	42 2 91
	add hl,de	25
	ex de, hl	235
	pop hl	225
	call fim linha	205 65* 127*
	jr prox linha	24 229
procurar GOTO	ld hl, (23635)	42 83 92
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
procurar	call encontrar	205 235* 126*
	jp nc, restaurar	210 184* 126*
	ld d,h	84
	ld e, l	93
	ld b, O	6 0
próx dígito	inc b	4
	inc hl	35
	ld a, (hl)	126
	cp 46	254 46
	jr nz, continuar	32 3
encontrar próx	ex de,hl	235
	jr procurar	24 236
continuar	cp 14	254 14
	jr nz, próx dígito	32 242
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	ld a, (hl)	126
	cp 58	254 58
	jr z, encontrado	40 4
	cp 13	254 13
	jr nz, encontrar próx	32 234
encontrado	ld a, b	120
comparar	cp 4	254 4
	jr z, calcular	40 16
	jr nc, encontrar próx	48 227
	push de	213
	ld h,d	98
	ld l,e	107
	push af	245
	ld a, 48	62 48
	call 3976	205 136 15
	pop af	241
	inc a	60

```
                pop de              209
                jr comparar         24 236
calcular        ld b,d              66
                ld c,e              75
                push de             213
                ld hl, O            33 O O
                ld de, 1OOO         17 232 3
                call somar          205 226* 126*
                ld de, 100          17 100 0
                call somar          205 226* 126*
                ld e, 10            30 10
                call somar          205 226* 126*
                ld a, (bc)          10
                sub 48              214 48
                ld e, a             95
                add hl,de           25
                ld b,h              68
                ld c,l              77
                ld hl, (23635)      42 83 92
procur linha    inc hl              35
                inc hl              35
fim programa    call verificar      205 76* 127*
                jr c, existe        56 3
                pop hl              225
                jr procurar         24 153
existe          ld a, (hl)          126
                cp c                185
                jr nc, próx byte    48 7
                inc hl              35
linha errada    inc hl              35
                call fim linha      205 65* 127*
                jr procur linha     24 235
próx byte       inc hl              35
                ld a, (hl)          126
                cp b                184
                jr c, linha errada  56 245
                dec hl              43
                dec hl              43
                ld c, (hl)          78
                dec hl              43
                ld h, (hl)          102
```



```
                ld l, c             105
                pop bc              193
                push bc             197
                push hl             229
                ld de, 1000         17 232 3
                call inserir        205 212* 126*
                ld de, 100          17 100 0
                call inserir        205 212* 126*
                ld e, 10            30 10
                call inserir        205 212* 126*
                ld e, 1             30 1
                call inserir        205 212* 126*
                inc bc              3
                sub a               151
                ld (bc),a           2
                inc bc              3
                ld (bc),a           2
                inc bc              3
                pop hl              225
                ld a,l              125
                ld (bc),a           2
                inc bc              3
                ld a,h              124
                ld (bc),a           2
                inc bc              3
                sub a               151
                ld (bc),a           2
                pop hl              225
                jp procurar         195 15* 126*
restaurar       ld hl, (23635)      42 83 92
linha seguinte  inc hl              35
                inc hl              35
                call verificar      205 76* 127*
                ret nc              208
                ld d,h              84
                ld e,l              93
                inc hl              35
                inc hl              35
                call fim linha      205 65* 127*
                push hl             229
                scf                 55
                sbc hl,de           237 82
```

```
                dec hl                  43
                ex de,hl                235
                ld (hl),e               115
                inc hl                  35
                ld (hl),d               114
                pop hl                  225
                jr linha seguinte       24 231
inserir         ld a, 48                62 48
subtrair        and a                   167
                sbc hl,de               237 82
                jr c, poke              56 3
                inc a                   60
                jr subtrair             24 248
poke            add hl,de               25
                ld (bc),a               2
                inc bc                  3
                ret                     201
somar           ld a, (bc)              10
                inc bc                  3
                sub 47                  214 47
repetir         dec a                   61
                ret z                   200
                add hl,de               25
                jr repetir              24 251
encontrar       ld a, (hl)              126
                call verificar          205 76* 127*
                ret nc                  208
                cp 234                  254 234
                jr nz, não REM          32 13
procur ENTER    inc hl                  35
                ld a, (hl)              126
                cp 13                   254 13
                jr nz, procur ENTER     32 250
incrementar     inc hl                  35
                inc hl                  35
                inc hl                  35
                inc hl                  35
                inc hl                  35
                jr encontrar            24 234
não REM         cp 34                   254 34
```



```
                jr nz, não cadeia       32 9
próx carácter   inc hl                  35
                ld a, (hl)              126
                cp 34                   254 34
                jr nz, próx carácter    32 250
                inc hl                  35
                jr encontrar            24 221
não cadeia      cp 13                   254 13
                jr z, incrementar       40 232
                call 6326               205 182 24
                jr z, encontrar         40 212
                cp 237                  254 237
                jr z, verif dígito      40 27
                cp 236                  254 236
                jr z, verif dígito      40 23
                cp 247                  254 247
                jr z, verif dígito      40 19
                cp 240                  254 240
                jr z, verif dígito      40 15
                cp 229                  254 229
                jr z, verif dígito      40 11
                cp 225                  254 225
                jr z, verif dígito      40 7
                cp 202                  254 202
                jr z, verif dígito      40 3
                inc hl                  35
                jr encontrar            24 181
verif dígito    inc hl                  35
                ld a, (hl)              126
                cp 48                   254 48
                jr c, encontrar         56 175
                cp 58                   254 58
                jr nc, encontrar        48 171
                ret                     201
fim linha       ld a, (hl)              126
repetir         call 6326               205 182 24
                jr z, repetir           40 251
                cp 13                   254 13
                inc hl                  35
                jr nz, fim linha        32 245
verificar       push hl                 229
```

```
push de          213
ld de, (23627)   237 91 75 92
and a            167
sbc hl,de        237 82
pop de           209
pop hl           225
ret              201
```

Como funciona

O par de registos hl é carregado com o primeiro número de linha. Se for zero, a rotina retorna ao BASIC. hl é carregado com o passo, o mesmo sucedendo se este for zero.

hl é carregado com o endereço do programa BASIC e posiciona-se de para apontar o primeiro número de linha. Chama-se a sub-rotina «verificar» e, se já tivermos atingido o fim do programa, a rotina salta para «procurar GOTO». bc é carregado com o anterior número de linha e este substituído por de, copiando-se bc para os ponteiros de linha.

Guarda-se hl na pilha, carregada com o passo e incrementada de de. O resultado copia-se para de, passando a ser o próximo número de linha. Retira-se então hl da pilha, sendo incrementada pela sub-rotina «fim linha», de forma a apontar a próxima linha. A rotina salta para «prox linha».

Em «procurar GOTO», hl é carregado com o endereço do programa BASIC e incrementado, para apontar o primeiro carácter da primeira linha. Chama-se então a sub-rotina «Encontrar»; se não existirem mais GOTOs, GOSUBs, etc., por alterar, a rotina salta para «restaurar». Caso contrário, no retorno desta sub-rotina, hl guarda o endereço do primeiro dígito após o GOTO, GOSUB, etc., copia-se este endereço para de e posiciona-se o registo b a zero. Este registo conta os dígitos do número que se segue.

O registo b é incrementado, o mesmo sucedendo com hl, para apontar o próximo carácter. Este carácter é carregado no acumulador. Se for um ponto decimal, hl é carregado com de e a rotina salta para «procurar», em busca do próximo GOTO. Se o carácter não for o indicativo de número (14), a rotina salta para «próx dígito».

hl é incrementado, apontando o carácter que se segue ao indicativo de número. Se este não for «dois pontos» ou ENTER, a rotina salta para «encontrar próx», uma vez que este GOTO tem um argumento que não é um número inteiro. O acumulador é carregado com o conteúdo do registo b. Se for quatro, a rotina salta para «calcular»; se for superior a quatro, salta para «encontrar próx», pois números de linha superiores a 9999 não são válidos.

Guarda-se de na pilha e copia-se para hl. O acumulador é também guardado na pilha e carregado com o código do carácter zero. Introduz-se este código no endereço apontado por hl através da rotina da ROM situada no endereço 3976. O acumulador é retirado da pilha e incrementado. Contém agora o número actual de dígitos do número de linha. Retira-se de da pilha e a rotina salta para «comparar».

Em «calcular», o endereço contido em de é copiado para bc, que é, por sua vez, salvo na pilha. hl é carregado com zero e de é carregado com 1000. Chama-se a sub-rotina «somar», adicionando a hl a casa dos milhares do número de linha em análise. Repete-se este processo para as centenas, as dezenas e as unidades, carregando assim hl com o número de linha. O par de registos bc é carregado com o resultado.

O par de registos hl é carregado com o endereço do programa BASIC. Chama-se a sub-rotina «verificar» e, caso tenha sido atingido o fim da zona de programas, retira-se hl da pilha e a rotina salta para «procurar», uma vez qúe o GOTO aponta para um número de linha que não existe. Se o *byte* endereçado por hl for menor que o valor contido no registo c, hl é incrementado, apontando a próxima linha, e a rotina salta para «procur linha». Caso contrário, hl é incrementado, apontando para o próximo *byte* do número de linha em teste. Se este for inferior ao valor contido no registo b, salta para «linha errada».

Neste ponto, deve já ter sido encontrado o endereço da linha destino do GOTO. hl é decrementado, apontando o início da linha, sendo depois carregado com o seu novo número de linha. bc é carregado com o endereço guardado na pilha, que é substituído pelo conteúdo de hl. bc contém, assim, o endereço para onde deve ser copiado o número de linha. de é carregado com 1000 e chama-se a sub-rotina «inserir». Esta sub-rotina calcula os milhares contidos em hl, adiciona-lhes 48, para produzir um valor interpretável pela máquina, e carrega o resultado no endereço apontado por bc. bc é incrementado, apontando para o próximo carácter. Este processo repete-se para as centenas, dezenas e unidades.

É, agora, construída a representação binária do número de linha; bc é incrementado, apontando para o carácter que se segue ao código indicativo de número (14), e posicionam-se a zero os dois *bytes* seguintes. Retira-se hl da pilha e o seu conteúdo é carregado nos dois *bytes* seguintes. O quinto *byte* é carregado com zero. Retira-se hl da pilha e a rotina salta para «procurar», repetindo-se o processo para o próximo GOTO.

Em «restaurar», hl é carregado com o endereço da zona de programas BASIC e incrementado duas vezes, para endereçar os ponteiros da linha seguinte, que neste momento contêm o anterior número de linha. Chama-se a sub-rotina «verificar» e, caso tenha sido atingido o fim do programa, a rotina regressa ao BASIC. bc é carregado com o endereço contido em hl e chama-se a sub-rotina «fim linha». Esta sub-rotina devolve-nos, em hl, o endereço do carácter de «ENTER» mais um. Guarda-se hl na pilha. Subtrai-se bc de hl e este é decrementado duas vezes, obtendo-se assim os novos ponteiros de linha, que são carregados nos endereços bc e bc+1. Retira-se hl da pilha e a rotina salta para «linha seguinte».

Sub-rotinas

Inserir:

O acumulador é carregado com o código de carácter zero. Subtrai-se de de hl e, se o resultado for negativo, a rotina salta para «*poke*». Caso contrário, o acumulador é incrementado e salta para «subtrair».

Em «*poke*», soma-se de a hl, para produzir um valor positivo. O acumulador é carregado no endereço apontado por bc e este é incrementado, apontado para o próximo *byte*. A sub-rotina regressa, então, à rotina principal (ret).

Somar:

O acumulador é carregado com o *byte* endereçado por bc e bc é incrementado, apontando o próximo *byte*. Subtai-se 47 do acumulador. Este registo é depois decrementado e, se o resultado for zero, regressa-se à rotina principal. Caso contrário, adiciona-se de a hl e a rotina salta para «repetir».

Encontrar:

O acumulador é carregado com o *byte* endereçado por hl. Chama-se a sub-rotina «verificar» e, caso tenha sido atingido o fim do programa BASIC, regressa-se à rotina principal. Se o carácter no acumulador não for o código de «REM», a sub-rotina salta para «não REM». hl é sucessivamente incrementado, até atingir o fim da linha. Este registo é de novo incrementado, para apontar o primeiro carácter da linha seguinte e a sub-rotina salta para «encontrar».

Em «não REM», se o acumulador não contém o código de carácter de aspas, efectua-se um salto para «não *string*». Caso contrário, hl é sucessivamente incrementado, até se encontrar um segundo código de aspas. hl é mais uma vez incrementado, para apontar o próximo carácter, e a sub-rotina salta para «encontrar».

Em «não *string*», se o acumulador contém o código de «ENTER», a sub-rotina salta para o troço «incrementar». Se contém o indicativo de número, salta para «encontrar». Se não for encontrada nenhuma das instruções GOSUB, GOTO, RUN, LIST, RESTORE, LLIST ou LINE, hl é incrementado e a sub-rotina salta para «encontrar». hl é incrementado e o acumulador é carregado com o carácter seguinte. Se o seu código não estiver entre 48 e 57, salta também para «encontrar». Regressa-se, então, à rotina principal.

Fim linha:

O acumulador é carregado com o *byte* endereçado por hl. Se este for o indicativo de número, hl é incrementado e a sub-rotina salta para «repetir».

hl, é incrementado. Se o acumulador não contiver o código de «ENTER», salta para «fim linha». Depois de efectuar um teste para verificar se já foi atingido o fim do programa BASIC, a sub-rotina regressa à rotina principal.

MEMÓRIA DISPONÍVEL

Comprimento: 14
Número de variáveis: 0
Teste soma: 1443

Operação

Esta rotina imprime no *écran* a quantidade de RAM disponível, em *bytes*.

Execução

PRINT USR endereço

Verificação de erros

Nenhuma

Comentários

Esta rotina deve ser executada antes da utilização de quaisquer rotinas que aumentem o tamanho de um programa, a fim de se verificar se existe RAM disponível em quantidade suficiente.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, O	33 O O
	add hl, sp	57
	ld de, (23653)	237 91 101 92
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Como funciona

O par de registos hl é posicionado a zero, sendo-lhe adicionado o endereço final da zona de RAM disponível (armazenado em sp). O par de registos de é carregado com o endereço inicial dessa mesma zona e é subtraído de hl. Copia-se hl para bc e a rotina regressa ao BASIC.

COMPRIMENTO DE UM PROGRAMA

Comprimento: 13
Número de variáveis: 0
Teste soma: 1544

Operação

Esta rotina imprime no *écran* o comprimento, em *bytes*, do programa BASIC presentemente em memória.

Execução

PRINT USR endereço

Verificação de erros

Nenhuma

Comentários

Nenhum

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23627)	42 75 92
	ld de, (23635)	237 91 83 92
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Como funciona

O par de registos hl é carregado com o endereço da zona de variáveis e de é carregado com o endereço do programa BASIC. de é subtraído de hl, operação que dá como resultado o comprimento do programa. Copia-se hl para bc e a rotina regressa ao BASIC.

ENDEREÇO DE UMA LINHA

Comprimento: 29
Número de variáveis: 1
Teste soma: 2351

Operação

Esta rotina obtém o endereço do primeiro carácter, após o código de «REM», numa dada linha.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
num linha	2	23296	Número da linha, que deve conter um «REM»

Execução

LET A=USR endereço

Verificação de erros

Se a linha especificada não existir ou não for uma instrução «REM», a rotina devolve o valor zero.

Comentários

Esta rotina pode servir para encontrar o endereço onde devemos introduzir código máquina, se pretendermos para tal utilizar uma instrução «REM».

A rotina posiciona a variável BASIC A (ou qualquer outra que se utilize de forma similar) com o endereço pretendido, ou com zero, se ocorrer algum erro. Não devem ser introduzidos números de linha superiores a 9999.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld a,h	124
	or l	181
	jr z, erro	40 5
	call 6510	205 110 25
	jr z, continuar	40 4
erro	ld bc, O	1 0 0
	ret	201
continuar	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	ld a, 234	62 234
	cp (hl)	190
	jr nz, erro	32 243

```
inc hl      35
ld b,h      68
ld c,l      77
ret         201
```

Como funciona

O par de registos hl é carregado com o número de linha especificado. Se este número for zero, a rotina salta para «erro». Caso contrário, invoca-se a rotina da ROM situada no endereço 6510, que nos devolve, em hl, o endereço da linha. Se a *flag* de zero estiver activa, salta para «continuar». Se estiver inactiva, a linha não existe e a rotina prossegue para «erro», onde bc é encarregado com zero e se executa o regresso ao BASIC.

Se a rotina passar por «continuar», hl é incrementado de quatro unidades, apontando para a primeira instrução da linha referida. Se esta instrução não for um «REM» (código 234), a rotina salta para «erro». Caso contrário, hl é incrementado, apontando o próximo carácter. Copia-se hl para bc e a rotina regressa ao BASIC.

CÓPIA DE UM BLOCO DE MEMÓRIA

Comprimento: 33
Número de variáveis: 3
Teste soma: 4022

Operação

Esta rotina copia um bloco de memória de um endereço para outro.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
início	2	23296	Endereço de origem
destino	2	23298	Endereço de destino
comprim	2	23300	Número de *bytes* a copiar

Execução

RAND USR endereço

Verificação de erros

Nenhuma

Comentários

Com esta rotina consegue-se produzir «filmes» de animação, através do seguinte método:

(i) desenhar a primeira imagem
(ii) copiar o *écran* para a zona acima de RAMTOP
(iii) repetir para as imagens seguintes
(iv) copiar as imagens de volta para o ficheiro de *écran*, em rápida sucessão.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91
	ld bc, (23300)	237 75 4 91
	ld a,b	120
	or c	177
	ret z	200
	and a	167
	sbc hl,de	237 82
	ret z	200
	add hl,de	25
	jr c, lddr	56 3
	ldir	237 176
	ret	201
lddr	ex de,hl	235
	add hl,bc	9
	ex de,hl	235
	add hl,bc	9
	dec hl	43
	dec de	27
	lddr	237 184
	ret	201

Como funciona

O par de registos hl é carregado com o endereço do primeiro *byte* a copiar, de é carregado com o endereço para onde este vai ser copiado e bc é carregado com o número de *bytes* a copiar. Se bc for zero ou hl=de, a rotina regressa ao BASIC. Se hl for maior que de, a cópia é efectuada através da instrução «ldir» e a rotina regressa ao BASIC.

Se de for maior que hl, adiciona-se bc−1 a ambos os pares de registos e efectua-se a cópia através da instrução «lddr», após o que a rotina regressa ao BASIC.

POSICIONAR VARIÁVEIS A ZERO

Comprimento: 108
Número de variáveis: 0
Teste soma: 10717

Operação

Colocam-se a zero todas as variáveis numéricas, posicionam-se a espaços todas as cadeias *(strings)* dimensionadas e todas as cadeias não dimensionadas são posicionadas com comprimento zero (cadeia nula).

Execução

RAND USR endereço

Verificação de erros

Se não existirem variáveis definidas em memória, a rotina regressa, de imediato, ao BASIC.

Comentários

Esta rotina é um excelente utilitário para detecção de erros em programas.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23627)	42 75 92
verificar	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	ld de, 1	17 1 0
	bit 7,a	203 127
	jr nz, próx bit	32 32
	bit 5,a	203 111
	jr z, 'string'	40 9
zero	ld b,5	6 5
próx byte	inc hl	35
	ld (hl),d	114
	djnz próx byte	16 252
	add hl,de	25
	jr verificar	24 232
string	inc hl	35
	ld c,(hl)	78
	ld (hl),d	114
	inc hl	35

Label	*Assembler*	*Números a introduzir*
	ld b,(hl)	70
	ld (hl),d	114
	inc hl	35
eliminar	ld a,b	120
	or c	177
	jr z, verificar	40 221
	push bc	197
	call 4120	205 24 16
	pop bc	193
	dec bc	11
	jr eliminar	24 244
próx bit	bit 6,a	203 119
	jr nz, bit 5	32 45
	bit 5,a	203 111
	jr z, array	40 7
número	inc hl	35
	bit 7, (hl)	203 126
	jr z, número	40 251
	jr zero	24 213
array	sub a	151
obter compr	puch af	245
	inc hl	35
	ld c, (hl)	78
	inc hl	35
	ld b, (hl)	70
	inc hl	35
	push hl	229
	ld l, (hl)	110
	ld h, d	98
	add hl,hl	41
	pop de	209
obter elem	inc de	19
	dec bc	11
	dec hl	43
	ld a,h	124
	or l	181
	jr nz, obter elem	32 249
	dec bc	11
apagar	inc de	19
	dec bc	11
	pop af	241

```
                push af               245
                ld (de),a             18
                ld a,b                120
                or c                  177
                jr nz, apagar         32 247
                pop af                241
restaurar       inc de                19
                rx de,hl              235
                jr verificar          24 164
bit 5           bit 5,a               203 111
                jr z, string array    40 5
                ld de, 14             17 14 0
                jr 'zero'             24 170
string array    ld a, 32              62 32
                jr obter compr        24 210
```

Como funciona

O par de registos hl é carregado com o endereço inicial da zona de variáveis. O acumulador é carregado com o *byte* apontado por hl. Se o valor deste *byte* for 128, a rotina regressa ao BASIC, pois o código 128 assinala o fim da zona de variáveis. O registo de é carregado com o valor um, para uso posterior. Se o *bit* 7 do acumulador estiver activo, a rotina salta para «próx *bit*» onde, se o *bit* 5 estiver a zero, salta para «*string*».

Se atingirmos «zero» sem saltar ao longo da rotina, a variável encontrada tem de ser um número cujo nome é constituído por apenas uma letra. O registo b posiciona-se em cinco, para funcionar como contador, e hl é incrementado, apontando o próximo *byte*, que é carregado com o valor zero. O contador é decrementado e, se ainda não atingiu o zero, a rotina salta para «próx *byte*». Soma-se de a hl, apontando para a próxima variável, e efectua-se um salto para «verificar».

Se a rotina executar o troço *string*, hl é incrementado, apontando para os *bytes* onde está contido o comprimento da cadeia encontrada. O anterior comprimento é carregado em bc, para funcionar como contador, e o novo comprimento é posicionado a zero. hl é de novo incrementado, apontando o texto da cadeia. Quando o contador atinge o zero, hl aponta para a variável seguinte, pelo que a rotina salta para «verificar». Enquanto tal não sucede, guarda-se bc na pilha e chama-se a rotina da ROM situada no endereço 4120, para eliminar um carácter. Retira-se o contador da pilha, após o que é decrementado, e a rotina salta para «eliminar».

Com «próx *bit*», testa-se o *bit* seis do acumulador. Se estiver posicionado a um, a rotina salta para «*bit* 5», pois a variável em teste é um *array* de caracteres ou uma variável de controle de um ciclo FOR/NEXT. Se estiver posicionado a zero e o *bit* cinco também estiver inactivo, salta para «*array*».

Se a rotina executar o troço «número», a variável encontrada deve ser um número cujo nome é constituído por mais de um carácter. O par de registos hl é incrementado, apontando para o próximo *byte*, operação que se repete até se encontrar um *byte* com o *bit* sete posicionado a um. Quando tal acontece, a rotina salta para «zero», onde a variável se posiciona a zero.

Se for executado o troço «*array*», o acumulador é carregado com zero, pois é com este valor que os elementos do *array* serão carregados mais adiante.

Em «obter compr», guarda-se o acumulador na pilha e hl é incrementado, apontando para os *bytes* que contêm o comprimento do *array*. Copia-se este comprimento para bc, para funcionar como contador. hl é de novo incrementado, apontando agora o *byte* que contém o número de dimensões, e depois guarda-se na pilha. hl é carregado com este número, que se multiplica por dois. Posiciona-se de com o endereço guardado na pilha, após o que este registo é decrementado hl vezes e o registo bc é decrementado (hl+1) vezes. De novo de é incrementado e bc decrementado. Agora de aponta para o próximo elemento do *array* e bc contém o número de *bytes* que faltam até ao fim deste. Retira-se da pilha o acumulador, que é carregado no endereço apontado por de. O contador em bc é decrementado e, se não tiver atingido ainda o zero, a rotina salta para «apagar». O valor em hl é ajustado de forma a apontar a próxima variável e a rotina salta para «verificar».

Em «*bit* 5», efectua-se um teste para verificar se a variável encontrada é um *array* de caracteres. No caso afirmativo, o acumulador é carregado com o código de carácter de «espaço» e a rotina salta para «obter compr». Se a rotina executar este troço, a variável encontrada deve ser uma variável de controle de um ciclo FOR/NEXT. Posiciona-se de com 14, uma vez que este valor, adicionado a (hl+5), aponta a próxima variável. A rotina salta, então, para «zero».

LISTAR VARIÁVEIS

Comprimento: 94
Número de variáveis: 0
Teste soma: 10295

Operação

Esta rotina lista os nomes de todas as variáveis actualmente em memória.

Execução

RAND USR endereço

Verificação de erros

Se não existirem variáveis em memória, a rotina regressa de imediato ao BASIC.

Comentários

Esta rotina mostra-se um precioso auxiliar na detecção de erros em programas, em especial os longos e complexos.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	res O, (IY + 2)	253 203 2 134
	ld hl, (23627)	42 75 92
próx variável	ld a, 13	62 13
	rst 16	215
	ld a, 32	62 32
	rst 16	215
	ld a, (hl)	126
	cp 128	254 128
	ret z	200
	bit 7,a	203 127
	jr z, bit 5	40 62
	bit 6,a	203 119
	jr z, próx bit	40 31
	bit 5,a	203 111
	jr z, string array	40 9
	sub 128	214 128
	ld de, 19	17 19 0
imprimir	rst 16	215
	add hl,de	25
	jr próx variável	24 225
string array	sub 96	214 96
	rst 16	215
	ld a, 36	62 36
parênteses	rst 16	215
	ld a, 40	62 40
	rst 16	215
	ld a, 41	62 41
ponteiros	inc hl	35
	ld e, (hl)	94
	inc hl	35
	ld d, (hl)	86
	inc hl	35
	jr print	24 234
próx bit	bit 5,a	203 111
	jr z, array	40 19
	sub 64	214 64
	rst 16	215
próx carácter	inc hl	35
	ld a, (hl)	126
	bit 7,a	203 127
	jr nz, último caráct	32 3
	rst 16	215
	jr próx carácter	24 247
último caráct	sub 128	214 128
saltar	ld de, 6	17 6 0
	jr print	24 211
array	sub 32	214 32
	jr parênteses	24 216
bit 5	bit 5,a	203 111
	jr nz, saltar	32 243
	add a, 32	198 32
	rst 16	215
	ld a, 36	62 36
	jr ponteiros	24 211

Como funciona

Posiciona-se a zero o *bit* 0 no endereço 23612, a fim de assegurar que quaisquer caracteres a imprimir surgirão na parte superior do *écran*. hl é carregado com o endereço da zona de variáveis. O acumulador é carregado com o código do carácter «ENTER», que se envia para o *écran* através da rotina da ROM situada no endereço 16. O acumulador é então carregado com o código de espaço, que se envia para o *écran* pelo mesmo processo.

O conteúdo do endereço apontado por hl é carregado no acumulador. Se o seu valor for 128, a rotina regressa ao BASIC, uma vez que atingimos o fim da zona de variáveis.

Se o *bit* 7 do acumulador estiver a zero, a rotina salta para «*bit* 5», pois a variável é uma cadeia ou um número com nome de uma só letra. É testado o *bit* 6 do acumulador. Se estiver a zero, a rotina salta para «próx bit», pois a variável em análise é um *array* ou um número cujo nome tem mais de uma letra. Se o *bit* 5 do acumulador estiver a zero, a rotina salta para «*string array*».

Atinge-se este ponto se a variável for controle de um ciclo FOR/NEXT.

Subtrai-se 128 ao acumulador, sendo o resultado o código de carácter a apresentar no *écran*. de é carregado com 19, para que aponte para a próxima variável depois de somado a hl. Envia-se para o *écran* o carácter contido no acumulador, soma-se de a hl e a rotina salta para «próx variável».

Se o troço «*string array*» for executado, subtrai-se 96 ao acumulador, para se obter o código do nome do *array* encontrado. Este código é enviado para o *écran*, através da rotina da ROM. Enviam-se depois para o *écran* um cifrão ($) e uma abertura de parênteses. O acumulador é carregado com o código de fecho de parênteses. hl é incrementado, apontando para os *bytes* que contêm o comprimento do *array*. Este comprimento é carregado em de e, uma vez somado a hl, permite obter o endereço da próxima variável. A rotina salta para «*print*», onde se envia para o *écran* o fecho de parênteses e efectua a soma entre hl e de.

Em «próx *bit*», testa-se o *bit* 5 do acumulador. Se estiver a zero, a rotina salta para «*array*». Se estiver a um, encontrámos um número cujo nome tem mais de uma letra. Subtrai-se 64 do acumulador e envia-se o resultado para o *écran*. Executa-se um ciclo onde são enviados para o *écran* todos os caracteres encontrados até que um deles tenha o *bit* sete posicionado a um. Subtrai-se 128 do código deste carácter, carrega-se de com o deslocamento necessário até à próxima variável e efectua-se um salto para «*print*».

Se for encontrado um *array*, subtrai-se 32 do acumulador, a fim de se obter o código correcto, e salta-se para «parênteses».

Em «*bit* 5», se tiver sido encontrado um número com nome de uma só letra, a rotina salta para «saltar».

Esta secção é executada se a variável encontrada for uma cadeia. Subtraindo-se 32 do acumulador, obtém-se o código a enviar para o *écran*. O acumulador é, finalmente, carregado com o cifrão ($) e a rotina salta para «ponteiros».

PROCURAR E LISTAR

Comprimento: 155
Número de variáveis: 2
Teste soma: 17221

Operação

Esta rotina procura, dentro de um programa BASIC, uma cadeia de caracteres especificada pelo utilizador, listando as linhas que a contêm.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
inic dados	2	23296	Endereço do primeiro *byte* de dados
comp cadei	1	23298	Número de caracteres na cadeia especificada

Execução

RAND USR endereço

Verificação de erros

Se não existir em memória nenhum programa BASIC, ou se a cadeia de caracteres tiver comprimento nulo, a rotina regressa imediatamente ao BASIC.

Comentários

O tempo de execução desta rotina é proporcional, quer ao comprimento da cadeia, quer ao tamanho do programa BASIC.

A cadeia a pesquisar carrega-se acima de RAMTOP, o endereço do seu primeiro *byte* deve ser colocado em 23296/7 e o comprimento da cadeia em 23298.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	res O, (IY + 2)	253 203 2 134
	ld ix, (23296)	221 42 0 91
	ld hl, (23635)	42 83 92
reinício	ld a, (23298)	58 2 91
	ld e, a	95
	cp O	254 0
	ret z	200
	push hl	229
restaurar	push ix	221 229
	pop bc	193
	ld d, O	22 0
	inc hl	35
	inc hl	35
	inc hl	35
verificar	inc hl	35
	push de	213
	ld de, (23627)	237 91 75 92
	and a	167
	sbc hl,de	237 82
	add hl,de	25
	pop de	209
	jr c, enter	56 4
	pop hl	225

```
                ret                  201
salto longo     jr reinício          24 223
enter           ld a, (hl)           126
                cp 13                254 13
                jr nz, número        32 5
                inc hl               35
                pop bc               193
                push hl              229
                jr restaurar         24 221
número          call 6326            205 182 24
                jr nz, comparar      32 8
                dec hl               43
diferente       push ix              221 229
                pop bc               193
                ld d, O              22 0
                jr verificar         24 216
comparar        ld a, (bc)           10
                cp (hl)              190
                jr nz, diferente     32 245
                inc bc               3
                inc d                20
                ld a,d               122
                cp e                 187
                jr nz, verificar     32 206
                ld a, 13             62 13
                rst 16               215
                pop hl               225
                push hl              229
                ld b, (hl)           70
                inc hl               35
                ld l, (hl)           110
                ld h, b              96
                ld de, 1000          17 232 3
                ld a, 47             62 47
milhares        inc a                60
                and a                167
                sbc hl,de            237 82
                jr nc, milhares      48 250
                add hl, de           25
                rst 16               215
                ld de, 100           17 100 0
```



```
                ld a, 47             62 47
centenas        inc a                60
                and a                167
                sbc hl, de           237 82
                jr nc, centenas      48 250
                add hl,de            25
                rst 16               215
                ld de, 10            17 10 0
                ld a, 47             62 47
dezenas         inc a                60
                and a                167
                sbc hl,de            237 82
                jr nc, dezenas       48 250
                add hl, de           25
                rst 16               215
                ld a,l               125
                add a, 48            198 48
                rst 16               215
                pop hl               225
                inc hl               35
                inc hl               35
                inc hl               35
próx caráct    inc hl               35
                ld a, (hl)           126
fim linha       cp 13                254 13
                jr nz, carácter 14   32 4
                rst 16               215
                inc hl               35
                jr salto longo       24 155
carácter 14     call 6326            205 182 24
                jr z, fim linha      40 243
                cp 32                254 32
                jr c, próx caráct    56 237
                rst 16               215
                jr próx caráct       24 234
```

Como funciona

O *bit* 0 do *byte* no endereço 23612 é desactivado, para assegurar que quaisquer caracteres enviados para o *écran* serão impressos no seu topo. ix é carregado com o endereço do primeiro *byte* de dados. Podemos assim carregar o endereço noutros pares de registos com menos referências do *buffer* de impressão. hl é carregado com o endereço do programa BASIC.

O acumulador é carregado com o comprimento da cadeia de caracteres, que se copia para o registo e. Se este comprimento for zero, a rotina regressa imediatamente ao BASIC. Guarda-se na pilha o endereço contido em hl, preservando-se, desta forma, o endereço da linha em análise.

Para se tornar mais acessível, copia-se o endereço dos dados de ix para bc. O registo d é carregado com zero, ou seja, com o número de caracteres iguais à cadeia a testar até agora encontrados. O par hl é incrementado de três unidades, apontando o maior *byte* do ponteiro de linha. É incrementado de novo, apontando, assim, o proximo carácter. O par de fica guardado na pilha.

de é carregado com o endereço da zona de variáveis, que se subtrai de hl. Se o resultado desta operação for negativo, a rotina salta para *«enter»*, após restaurar hl com o seu valor inicial e retirar de da pilha. Se o resultado for positivo, posiciona-se a pilha com o seu conteúdo inicial e a rotina regressa ao BASIC, pois foi atingido o fim do programa.

Em *«enter»*, o acumulador é carregado com o *byte* contido no endereço apontado por hl. Se este não for o código de «ENTER», a rotina salta para «número». No caso afirmativo, hl é incrementado até apontar o início da próxima linha. Retira-se da pilha o endereço da linha anterior, sendo substituído pelo novo valor, contido em hl. A rotina salta para «restaurar».

Em «número», chama-se a rotina da ROM situada no endereço 6326. Se o carácter no acumulador for o indicativo de número (14), hl é incrementado até apontar o primeiro carácter após a representação binária desse número. Se não for encontrado o indicativo de número, a rotina salta para «comparar». Caso contrário, hl é decrementado e a rotina segue para «diferente». bc é copiado de ix, posiciona-se a zero o número de caracteres encontrado e a rotina salta para «verificar».

Em «comparar», o acumulador é carregado com o *byte* cujo endereço é apontado por bc. Se não for igual ao *byte* apontado por hl, a rotina salta para «diferente». bc é incrementado, apontando para o próximo *byte* de dados e o número de caracteres encontrado é incrementado. Se este número não for igual ao comprimento total da cadeia, a rotina salta para «verificar».

O acumulador é carregado com o código de «ENTER», que se envia para o *écran* através da rotina da ROM situada no endereço 16. O endereço da linha a enviar para o *écran* é retirado da pilha, para hl. O número de linha é carregado em hl através do registo b, de com 1000 e o acumulador com um menos o código do carácter «O». O acumulador é incrementado e de é sucessivamente subtraído de hl, até este ficar negativo. Soma-se então de a hl, produzindo-se um resultado positivo. Envia-se então para o *écran* o carácter no acumulador.

O processo acima descrito repete-se para de=100 e de=10. O resultado é carregado no acumulador, ao qual ainda se adicionam 48 unidades, após o que segue para o *écran*.

Retira-se da pilha o endereço inicial da linha e carrega-se em hl. Este é incrementado até apontar o maior *byte* do ponteiro de linha. É de novo incrementado sendo o *byte* apontado carregado no acumulador. Se este *byte* não for o código de «ENTER», a rotina salta para «carácter 14». Caso contrário, «ENTER» segue para o *écran*, hl é incrementado e salta para «reinício».

Em «carácter 14», chama-se a rotina da ROM situada no endereço 6326. Se o carácter no acumulador for o indicativo de número, hl é incrementado até apontar o primeiro carácter após esse número. Este carácter é carregado no acumulador e a rotina salta para «fim linha». Neste ponto, se o carácter contido no acumulador tiver código inferior a 32, a rotina salta para «próx carácter». Se esse código for superior a 31, o carácter achado segue para o *écran* e a rotina salta também para «próx caráct».

PROCURAR E SUBSTITUIR

Comprimento: 85
Número de variáveis: 3
Teste soma: 8518

Operação

Esta rotina procura, num programa BASIC, uma dada cadeia de caracteres, substituindo cada ocorrência por uma outra cadeia do mesmo comprimento.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
iníc ant	2	23296	Endereço da cadeia a substituir
comp cad	1	23298	Comprimento da cadeia a substituir
iníc novo	2	23299	Endereço da cadeia de substituição

Execução

RAND USR endereço

Verificação de erros

Se o comprimento da cadeia for zero ou se não existir em memória nenhum programa BASIC, a rotina regressa, imediatamente, ao BASIC.

Comentários

O tempo de execução desta rotina depende do comprimento da cadeia e do tamanho do programa BASIC.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld ix, (23296)	221 42 0 91

```
                ld hl, (23635)        42 83 92
                ld a, (23298)         58 2 91
                ld e, a               95
                cp O                  254 O
                ret z                 200
                dec hl                43
nova linha      inc hl                35
                inc hl                35
                inc hl                35
                inc hl                35
                jr apagar             24 23
verificar       inc hl                35
                push de               213
                ld de, (23627)        237 91 75 92
                and a                 167
                sbc hl,de             237 82
                add hl,de             25
                pop de                209
                ret nc                208
                ld a, (hl)            126
                cp 13                 254 13
                jr z, nova linha      40 233
                call 6326             205 182 24
                jr nz, comparar       32 8
                dec hl                43
apagar          push ix               221 229
                pop bc                193
                ld d, O               22 0
                jr verificar          24 226
comparar        ld a, (bc)            10
                cp (hl)               190
                jr nz, apagar         32 245
                inc bc                3
                inc d                 20
                ld a,d                122
                cp e                  187
                jr nz, verificar      32 216
                push hl               229
                ld d, O               22 0
                and a                 167
                sbc hl,de             237 82
                ld d,e                83
```



```
                ld bc, (23299)        237 75 3 91
                inc d                 20
próx carácter   inc hl                35
                dec d                 21
                jr z, fim             40 5
                ld a, (bc)            10
                ld (hl),a             119
                inc bc                3
                jr próx carácter      24 247
fim             pop hl                225
                jr apagar             24 215
```

Como funciona

ix é carregado com o endereço da cadeia a procurar. Este endereço deve estar situado acima de RAMTOP. hl é carregado com o endereço da zona de programas e o acumulador é carregado com o comprimento da cadeia. Copia-se este comprimento para o registo e, a fim de ser utilizado mais adiante. Se o comprimento da cadeia é zero, a rotina regressa ao BASIC. hl ajusta-se de forma a apontar para o maior *byte* do próximo ponteiro de linha e a rotina salta para «apagar».

Em «verificar», hl é incrementado, apontando o carácter seguinte. Guarda-se na pilha de, que é carregado com o endereço da zona de variáveis. Se hl não for menor que de, atingimos o fim do programa, pelo que, após recuperar o conteúdo de de armazenado na pilha, a rotina regressa ao BASIC.

O acumulador é carregado com o carácter endereçado por hl. Se for o carácter «ENTER», a rotina salta para «nova linha». Se esse carácter não for o indicativo de número (14), salta para «comparar». No caso contrário, hl é incrementado cinco vezes, apontando agora o quinto *byte* do número encontrado.

Em «apagar», bc é carregado com o endereço da cadeia a procurar. O registo d é posicionado a zero, para acumular o número de caracteres da cadeia encontrados. A rotina salta, então, para «verificar».

Em «comparar», o acumulador é carregado com o carácter da cadeia apontado por bc. Se for diferente do apontado por hl, a rotina salta para «apagar». bc é incrementado, apontando o carácter seguinte da cadeia, e o contador em d é também incrementado. Se não for igual ao comprimento total da cadeia, a rotina salta para «verificar».

Se se encontrou a cadeia pretendida, guarda-se hl na pilha para que a rotina possa iniciar a próxima busca a partir deste endereço. de é carregado com o comprimento da cadeia, que é subtraído de hl, dando como resultado um menos o endereço inicial. O comprimento da cadeia é também carregado em d, para funcionar como contador. bc é carregado com o endereço inicial da nova cadeia e o registo d é incrementado. O registo hl é incrementado, apontando a

nova posição, e o contador é decrementado. Se o seu conteúdo for nulo, retira-se hl da pilha e a rotina salta para «apagar», em busca de nova ocorrência. O acumulador é carregado com o carácter apontado por bc, que se introduz no endereço apontado por hl. bc é incrementado, apontando o carácter seguinte, e a rotina salta para «próx caráct».

PROCURAR NA ROM

Comprimento: 58
Número de variáveis: 3
Teste soma: 6533

Operação

Esta rotina procura na ROM uma dada cadeia (*string*) de caracteres definida pelo utilizador.

Variáveis

Nome	*Comprimento*	*Posição*	*Comentário*
end inic	2	23296	Endereço inicial da busca
comp *string*	1	23298	Número de *bytes* da *string* a pesquisar
end *string*	2	23299	endereço na RAM da *string* a pesquisar

Execução

PRINT USR endereço

Verificação de erros

Se o comprimento da cadeia for nulo, a rotina regressa imediatamente ao BASIC, dando como resultado o endereço inicial da cadeia. Não se encontrando a cadeia, o resultado apresentado é 65535.

Comentários

Rotina útil na construção de rotinas em código máquina, pois por ela o utilizador descobre rotinas da ROM, desde que conheça parte da sua constituição.

Atendendo a que a maior parte das rotinas da ROM do *Spectrum* foram adaptadas a partir do *ZX81*, podemos alterar facilmente algumas rotinas originalmente escritas para esta máquina, de forma a utilizá-las no *Spectrum*. Por exemplo, a rotina «Endereço de uma linha» chama a rotina da ROM

situada no endereço 6510. No *ZX81*, esta rotina começa no endereço 2520. Desassemblando esta rotina, obtemos:

```
push hl
ld hl, program
ld d,h*
ld e,l*
pop bc*
call 09EA
```

Os três *bytes* assinalados com asterisco são idênticos no *Spectrum*, e podem ser encontrados usando a rotina de procura. De facto, a rotina dá como resultado, neste caso, o endereço 6514, que representa quatro mais o endereço da rotina da ROM pretendida.

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	ld hl, (23296)	42 0 91
	ld de, (23298)	237 91 2 91
reinício	ld bc, (23299)	237 75 3 91
	ld a,e	123
	cp O	254 O
	ret z	200
	push hl	229
	ld d, O	22 O
comparar	ld a, (bc)	10
	cp (hl)	190
	jr z, igualdade	40 25
	pop hl	225
	inc hl	35
	push de	213
	push hl	229
	ld hl, 16384	33 0 64
	l d, O	22 0
	and a	167
	sbc hl,de	237 82
	inc hl	35
	pop de	209
	and a	167
	sbc hl,de	237 82
	ex de,hl	235
	pop de	209

	jr nz, reinício	32 220
	ld bc, 65535	1 255 255
	ret	201
igualdade	inc d	20
	ld a,d	122
	cp e	187
	jr nz, próx byte	32 2
	pop bc	193
	ret	201
próx byte	inc hl	35
	inc bc	3
	jr comparar	24 216

Como funciona

O par de registos hl é carregado com o endereço da primeira posição de memória a testar. Para se obter a primeira ocorrência na ROM, o endereço inicial deve ser zero. O registo e é carregado com o número de *bytes* da cadeia a pesquisar. O par de registos bc é carregado com o endereço da cadeia, introduzida pelo utilizador, na RAM. O acumulador é carregado com o comprimento da cadeia e, se for zero, a rotina regressa ao BASIC.

Guarda-se na pilha o endereço em hl. O acumulador é carregado com o *byte* apontado pelo par bc. Se for igual ao *byte* apontado por hl, a rotina salta para «igualdade». Se forem diferentes, hl é carregado com o endereço que guardou na pilha. Este endereço é, então, incrementado para apontar a posição de memória seguinte.

Guardam-se na pilha os registos de e hl, sendo hl carregado com o endereço do primeiro *byte* de RAM e de carregado com o comprimento da cadeia a pesquisar. de é subtraído de hl, obtendo-se o mais alto endereço inicial possível para a cadeia. Este resultado é incrementado, obtendo-se o primeiro endereço em que a cadeia não pode estar.

O endereço no topo da pilha é carregado em de e, por sua vez, subtraído de hl. Enquanto se analisa o resultado desta operação, hl é carregado com o conteúdo de de e este com o valor na pilha. Se o resultado foi zero, bc é carregado com 65535 e a rotina regressa ao BASIC, uma vez que a cadeia especificada não existe na ROM. Se o resultado não foi zero, a rotina salta para «reinício».

Em «igualdade», o registo d é incrementado, contabilizando as igualdades sucessivas. Se este número igualar o comprimento da cadeia, retira-se bc da pilha e a rotina regressa ao BASIC. Se o registo d não continha uma contagem igual ao comprimento total da cadeia, hl e bc são incrementados, apontando para os *bytes* seguintes, e a rotina salta para «comparar».

INSTR$

Comprimento: 168
Número de variáveis: 0
Teste soma: 19875

Operação

Esta rotina calcula a posição de uma subcadeia (B$), no interior de uma cadeia principal (A$), dando resultado zero se ocorrer algum erro.

Execução

Let P=USR endereço

Verificação de erros

Se qualquer uma das cadeias não existir, o comprimento da subcadeia for zero ou maior que o comprimento da cadeia principal, a rotina dá como resultado zero.

Se não ocorrer nenhum erro, mas não se encontrar a subcadeia, o resultado é também zero.

Comentários

No retorno desta rotina, a variável P (ou qualquer outra utilizada da mesma forma) contém o resultado pretendido. As cadeias referenciadas não podem ser DIMensionadas (*arrays* de caracteres). Para utilizar um outro par de cadeias, os números assinalados com asterisco devem ser alterados. 66* representa a subcadeia e 65* é a cadeia principal. Para efectuar esta alteração, os números acima referidos devem ser substituídos pelos códigos de carácter pretendidos (A a Z=65 a 90).

Listagem de código máquina

Label	*Assembler*	*Números a introduzir*
	sub a	151
	ld b,a	71
	ld c,a	79
	ld d, a	87
	ld e,a	95
	ld hl, (23627)	42 75 92
próx variável	ld a, (hl)	126
	cp 128	254 128
	jr z, não encontr	40 95
	bit 7,a	203 127
	jr nz, for-next	32 41
	cp 96	254 96
	jr nc, número	48 29

	cp 65	254 65*
	jr nz, substring	32 2
	ld d,h	84
	ld e,l	93
substring	cp 66	254 66*
	jr nz, verificar	32 2
	ld b,h	68
	ld c,l	77
verificar	ld a,d	122
	or e	179
	jr z, string	40 4
	ld a, b	120
	or c	177
	jr nz, encontrado	32 38
string	push de	213
	inc hl	35
	ld e, (hl)	94
	inc hl	35
	ld d, (hl)	86
somar	add hl,de	25
	pop de	209
	jr incrementar	24 5
número	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
	inc hl	35
incrementar	inc hl	35
	jr próx variável	24 206
for-next	cp 224	254 224
	jr c, próx bit	56 6
	push de	213
	ld de, 18	17 18 0
	jr add	24 234
próx bit	bit 5,a	203 111
	jr z, string	40 225
próx byte	inc hl	35
	bit 7, (hl)	203 126
	jr z, próx byte	40 251
	jr número	24 227

encontrado	ex de,hl	235
	inc hl	35
	inc hl	35
	push hl	229
	push hl	229
	inc bc	3
	push bc	197
	ld a, (bc)	10
	ld e,a	95
	inc bc	3
	ld a, (bc)	10
	ld d,a	87
	or e	179
	jr z, compr zero	40 11
	push de	213
	ld a, (hl)	126
	dec hl	43
	ld l, (hl)	110
	ld h,a	103
	and a	167
	sbc hl,de	237 82
	jr nc, continuar	48 8
	pop bc	193
compr zero	pop bc	193
	pop bc	193
erro	pop bc	193
não encontr	ld bc, O	1 0 0
	ret	201
continuar	pop ix	221 225
	pop bc	193
	ex de,hl	235
	pop hl	225
	inc bc	3
	inc bc	3
guardar	inc hl	35
	push hl	229
	push bc	197
	push ix	221 229
	push de	213
comparar	ld a, (bc)	10
	cp (hl)	190

	jr z, igualdade	40 12
	pop de	209
	pop ix	221 225
	pop bc	193
	pop hl	225
	ld a,d	122
	or e	179
	jr z, erro	40 225
	dec de	27
	jr guardar	24 234
igualdade	inc hl	35
	inc bc	3
	push hl	229
	dec ix	221 43
	push ix	221 229
	pop hl	225
	ld a,h	124
	or l	181
	pop hl	225
	jr nz, comparar	32 227
	pop de	209
	pop de	209
	and a	167
	sbc hl,de	237 82
	pop de	209
	pop de	209
	pop de	209
	and a	167
	sbc hl,de	237 82
	ld b,h	68
	ld c,l	77
	ret	201

Como funciona

Posicionam-se a zero o acumulador, o par de registos bc e o par de registos de. Mais adiante, bc será posicionado com o endereço de B$ e de com o endereço de A$. hl é carregado com o endereço da zona de variáveis.

O acumulador é carregado com o *byte* endereçado por hl. Se o seu conteúdo for 128, a rotina salta para «não encontr», pois atingimos o fim da zona de variáveis. Se o *bit* 7 do acumulador estiver posicionado a um, a rotina salta para *«for/next»*, pois a variável encontrada não é uma cadeia nem um número com nome de uma só letra. Se o acumulador contém um número superior a 95, salta para «número».

Ao chegar a este troço, a rotina deve já ter encontrado uma cadeia. Se o acumulador contém 65, localizámos A$, sendo o conteúdo de hl copiado para de. Se o acumulador contém 66, localizámos B$, sendo hl copiado para bc. Se nem de nem bc contém zero, ambas as cadeias foram localizadas, pelo que a rotina salta para «encontrado».

Se a rotina executar o troço *«string»*, guarda-se de na pilha e carrega-se com o comprimento da cadeia encontrada. Este comprimento soma-se ao endereço do maior *byte* dos ponteiros da cadeia e o resultado é guardado em hl. Retira-se de da pilha e a rotina salta para «incrementar».

Em «número», hl é incrementado cinco vezes, apontando para o último *byte* de qualquer número encontrado. É de novo incrementado, apontando para a variável seguinte, e a rotina salta para «próx variável».

Em *«for/next»*, se o conteúdo do acumulador for inferior a 224, a rotina salta para «prox *bit*», uma vez que a variável encontrada não é controle de um ciclo FOR/NEXT. Se o valor no acumulador for superior a 223, soma-se dezoito a hl, que fica a apontar o último *byte* da variável e a rotina salta para «incrementar».

Se a rotina executar o troço «próx *bit*» e o *bit* 5 do acumulador estiver posicionado a zero, é efectuado um salto para *«string»*, onde hl é carregado com o endereço da variável seguinte, pois a variável em análise é um *array*.

Se a rotina executar o troço «próx *byte*», foi encontrado um número de nome constituído por mais de dois caracteres. Assim, hl é incrementado até apontar o último carácter do nome da variável e a rotina salta para «número».

Em «encontrado», hl é carregado com o endereço de A$, que é incrementado duas vezes para apontar o endereço do maior *byte* dos ponteiros. Guarda-se este valor duas vezes na pilha. bc é incrementado, apontando para o menor *byte* dos ponteiros de B$. O endereço contido em bc é, então, guardado na pilha. de é carregado com o comprimento de B$ e, se este for zero, a rotina salta para «compr zero». de também se guarda na pilha. hl é carregado com o comprimento de A$ e, se este comprimento não for inferior a de, a rotina salta para «continuar». Repõe-se a pilha com o seu anterior conteúdo, bc é carregado com zero e a rotina regressa ao BASIC.

Em «continuar» ix é posicionado com o comprimento de B$ e posiciona-se bc para apontar o menor *byte* de endereço dos ponteiros de B$. de é carregado com a diferença de comprimentos entre A$ e B$ e hl é carregado com o endereço do maior *byte* dos ponteiros de A$. bc é decrementado duas vezes, obtendo-se, assim, o endereço do primeiro carácter de B$. hl é incrementado, apontando para o próximo carácter de A$. Guardam-se na pilha hl, bc, ix e de. O acumulador é carregado com o *byte* endereçado por bc e, se este for igual ao *byte* endereçado por hl, a rotina salta para «igualdade». Retiram-se da pilha de, ix, bc e hl. Se de contém zero, salta para «erro», pois B$ não está contida em A$. O contador em de é decrementado e a rotina salta para «guardar».

Se for executado o troço «igualdade», hl e bc são ambos incrementados, a fim de apontarem os caracteres seguintes em A$ e B$, respectivamente.

Guarda-se hl na pilha. O contador em ix é decrementado e, após retirar hl da pilha, se ix não contiver zero, a rotina salta para «comparar».

Se a rotina chegar a este ponto, foi detectada uma ocorrência de B$ em A$. O comprimento de B$ e, depois, o endereço do maior *byte* dos ponteiros de A$ são subtraídos de hl. O resultado destas operações é a posição de B$ em A$. Copia-se este resultado para o registo bc e a rotina regressa ao BASIC.

Apêndice A

Neste apêndice existem, basicamente, duas tabelas. A Tabela A2 apresenta as instruções de um *byte* e as de dois *bytes* que são precedidas de 203 (CB em hexadecimal) ou de 237 (ED em hexadecimal). A Tabela A3 apresenta as instruções referentes a registos de índex.

Existem muitas sistematizações no conjunto de instruções. Por exemplo, os registos estão quase sempre na ordem b, c, d, e, h, l, (hl), a, tal como sucede no grupo de instruções de cópia de registo para registo a oito *bits*, de números 64 a 127. De forma similar, os códigos referentes aos registos de índex imitam os códigos referentes a hl, mas precedidos de 221 (DD em hexadecimal) quando referidos ao registo ix e de 253 (FD em hexadecimal) quando referidos ao registo iy.

Algumas das instruções utilizam a seguinte simbologia:

n — inteiro de um *byte*, entre 0 e 255, inclusive.

d — deslocamento a um *byte*, entre 0 e 255 (instruções com registos de índex) ou entre −127 e 128 (instruções de salto). Valores negativos obtém-se subtraindo o valor absoluto de 256.

mn — inteiro de dois *bytes*, entre 0 e 65535, inclusive. O *byte* mais significativo vem em segundo lugar; por exemplo: guarda-se 16384 (=0*256*64) na forma 0,64.

Estes símbolos surgem sempre no ou nos *bytes* subsequentes à instrução a que se referem, excepto nas instruções com registos de índex a três *bytes* (colunas 5 e 6 da tabela A3), onde surgem entre os segundo e terceiro *bytes*. Vejam-se os exemplos dados na Tabela A1.

Tabela A1. *Alguns exemplos do conjunto de instruções do* Z80A. *Na coluna 1 é referenciada a coluna correspondente nas tabelas A2 e A3.*

Tabela e número de coluna	*Forma geral*	*Exemplo específico*	*Decimal*			
A2,3	—	inc b	4			
A2,3	ld e,n	ld e,25	30	25		
A2,3	ld a, (nn)	ld a, (23296)	58	0	91	
A2,4	—	res 2,d	203	146		
A2,5	ld (nn),de	ld (23760),de	237	83	208	92
A3,3	—	add ix,bc	221	9		

A3,3	ld (ix + d),n	ld (ix + 193),5	221	54	193	5
A3,4	—	add iy,bc	253	9		
A3,4	ld (nn), iy	ld (23760),iy	253	34	208	92
A3,5	rrc (ix + d)	rrc (ix + 5)	221	203	5	14
A3,6	rrc (iy + d)	rrc (iy + 5)	253	203	5	14

Tabela A2. *Instruções do Z80A, excepto as referentes aos registos de índex (ver Tabela A3).*

Decimal	*Hex*	*Op Code*	*Após 203 (hex CB)*	*Após 237 (hex ED)*
0	00	nop	rlc b	
1	01	ld bc,nn	rlc c	
2	02	ld (bc),a	rlc d	
3	03	inc bc	rlc e	
4	04	inc b	rlc h	
5	05	dec b	rlc l	
6	06	ld b,n	rlc (hl)	
7	07	rlca	rlc a	
8	08	ex,af,af'	rrc b	
9	09	add hl,bc	rrc c	
10	0A	ld a, (bc)	rrc d	
11	0B	dec bc	rrc e	
12	0C	inc c	rrc h	
13	0D	dec c	rrc l	
14	0E	ld c,n	rrc (hl)	
15	0F	rrca	rrc a	
16	10	djnz d	rl b	
17	11	ld de,nn	rl c	
18	12	ld (de),a	rl d	
19	13	inc de	rl e	
20	14	inc d	rl h	
21	15	dec d	rl l	
22	16	ld d,n	rl (hl)	
23	17	rla	rl a	
24	18	jr d	rr b	
25	19	add hl,de	rrc	
26	1A	ld a, (de)	rr d	
27	1B	dec de	rr e	
28	1C	inc e	rr h	
29	1D	dec e	rr l	
30	1E	ld e,n	rr (hl)	

31	1F	rra	rr a	
32	20	jr nz,d	sla b	
33	21	ld hl,nn	sla c	
34	22	ld (nn),hl	sla d	
35	23	inc hl	sla e	
36	24	inc h	sla h	
37	25	dec h	sla l	
38	26	ld h,n	sla (hl)	
39	27	daa	sla a	
40	28	jr z,d	sra b	
41	29	add hl,hl	sra c	
42	2A	ld hl, (nn)	sra d	
43	2B	dec hl	sra e	
44	2C	inc l	sra h	
45	2D	dec l	sra l	
46	2E	ld l,n	sra (hl)	
47	2F	cpl	sra a	
48	30	jr nc,d		
49	31	ld sp,nn		
50	32	ld (nn),a		
51	33	inc sp		
52	34	inc (hl)		
53	35	dec (hl)		
54	36	ld (hl),n		
55	37	scf		
56	38	jr c,d	srl b	
57	39	add hl,sp	srl c	
58	3A	ld a, (nn)	srl d	
59	3B	dec sp	srl e	
60	3C	inc a	srl h	
61	3D	dec a	srl l	
62	3E	ld a,n	srl (hl)	
63	3F	ccf	srl a	
64	40	ld b,b	bit 0,b	in b, (c)
65	41	ld b,c	bit 0,c	out (c),b
66	42	ld b,d	bit 0,d	sbc hl,bc
67	43	ld b,e	bit 0,e	ld (nn),bc
68	44	ld b,h	bit 0,h	neg
69	45	ld b,l	bit 0,l	retn
70	46	ld b, (hl)	bit 0, (hl)	im 0
71	47	ld b,a	bit 0,a	ld i,a
72	48	ld c,b	bit 1,b	in c, (c)

73	49	ld c,c	bit 1,c	out (c),c
74	4A	ld c,d	bit 1,d	adc hl,bc
75	4B	ld c,e	bit 1,e	ld bc,(nn)
76	4C	ld c,h	bit 1,h	
77	4D	ld c,l	bit 1,l	reti
78	4E	ld c, (hl)	bit 1, (hl)	
79	4F	ld c,a	bit 1,a	ld r,a
80	50	ld d,b	bit 2,b	in d, (c)
81	51	ld d,c	bit 2,c	out (c),d
82	52	ld d,d	bit 2,d	sbc hl,de
83	53	ld d,e	bit 2,e	ld (nn),de
84	54	ld d,h	bit 2,h	
85	55	ld d,l	bit 2,l	
86	56	ld d, (hl)	bit 2, (hl)	im 1
87	57	ld d,a	bit 2,a	ld a,i
88	58	ld e,b	bit 3,b	in e,(c)
89	59	ld e,c	bit 3,c	out (c),e
90	5A	ld e,d	bit 3,d	adc hl,de
91	5B	ld e,e	bit 3,e	ld de,(nn)
92	5C	ld e,h	bit 3,h	
93	5D	ld e,l	bit 3,1	
94	5E	ld e, (hl)	bit 3, (hl)	im 2
95	5F	ld e,a	bit 3,a	ld a,r
96	60	ld h,b	bit 4,b	in h, (c)
97	61	ld h,c	bit 4,c	out (c),h
98	62	ld h,d	bit 4,d	sbc hl,hl
99	63	ld h,e	bit 4,e	ld (nn),hl
100	64	ld h,h	bit 4,h	
101	65	ld h,l	bit 4,l	
102	66	ld h, (hl)	bit 4, (hl)	
103	67	ld h,a	bit 4,a	rrd
104	68	ld l,b	bit 5,b	in l,(c)
105	69	ld l,c	bit 5,c	out (c),l
106	6A	ld l,d	bit 5,d	adc hl,hl
107	6B	ld l,e	bit 5,e	ld hl,(nn)
108	6C	ld l,h	bit 5,h	
109	6D	ld l,l	bit 5,l	
110	6E	ld l, (hl)	bit 5, (hl)	
111	6F	ld l,a	bit 5,a	rld
112	70	ld (hl),b	bit 6,b	in f,(c)
113	71	ld (hl),c	bit 6,c	
114	72	ld (hl),d	bit 6,d	sbc hl,sp

115	73	ld (hl),e	bit 6,e	ld (nn),sp
116	74	ld (hl),h	bit 6,h	
117	75	ld (hl),l	bit 6,l	
118	76	halt	bit 6, (hl)	
119	77	ld (hl),a	bit 6,a	
120	78	ld a,b	bit 7,b	in a,(c)
121	79	ld a,c	bit 7,c	out (c),a
122	7A	ld a,d	bit 7,d	adc hl,sp
123	7B	ld a,e	bit 7,e	ld sp, (nn)
124	7C	ld a,h	bit 7,h	
125	7D	ld a,l	bit 7,1	
126	7E	ld a, (hl)	bit 7, (hl)	
127	7F	ld a,a	bit 7,a	
128	80	add a,b	res 0,b	
129	81	add a,c	res 0,c	
130	82	add a,d	res 0,d	
131	83	add a,e	res 0,e	
132	84	add a,h	res 0,h	
133	85	add a,l	res 0,l	
134	86	add a, (hl)	res 0, (hl)	
135	87	add a,a	res 0,a	
136	88	adc a,b	res 1,b	
137	89	adc a,c	res 1,c	
138	8A	adc a,d	res 1,d	
139	8B	adc a,e	res 1,e	
140	8C	adc a,h	res 1,h	
141	8D	adc a,l	res 1,l	
142	8E	adc a, (hl)	res 1, (hl)	
143	8F	adc a,a	res 1,a	
144	90	sub b	res 2,b	
145	91	sub c	res 2,c	
146	92	sub d	res 2,d	
147	93	sub e	res 2,e	
148	94	sub h	res 2,h	
149	95	sub l	res 2,l	
150	96	sub (hl)	res 2, (hl)	
151	97	sub a	res 2,a	
152	98	sbc a,b	res 3,b	
153	99	sbc a,c	res 3,c	
154	9A	sbc a,d	res 3,d	
155	9B	sbc a,e	res 3,e	
156	9C	sbc a,h	res 3,h	

157	9D	sbc a,l	res 3,l	
158	9E	sbc a, (hl)	res 3, (hl)	
159	9F	sbc a,a	res 3,a	
160	A0	and b	res 4,b	ldi
161	A1	and c	res 4,c	cpi
162	A2	and d	res 4,d	ini
163	A3	and e	res 4,e	outi
164	A4	and h	res 4,h	
165	A5	and l	res 4,l	
166	A6	and (hl)	res 4, (hl)	
167	A7	and a	res 4,a	
168	A8	xor b	res 5,b	ldd
169	A9	xor c	res 5,c	cpd
170	AA	xor d	res 5,d	ind
171	AB	xor e	res 5,e	outd
172	AC	xor h	res 5,h	
173	AD	xor l	res 5,l	
174	AE	xor (hl)	res 5, (hl)	
175	AF	xor a	res 5,a	
176	B0	or b	res 6,b	ldir
177	B1	or c	res 6,c	cpir
178	B2	or d	res 6,d	inir
179	B3	or e	res 6,e	otir
180	B4	or h	res 6,h	
181	B5	or l	res 6,l	
182	B6	or (hl)	res 6, (hl)	
183	B7	or a	res 6,a	
184	B8	cp b	res 7,b	lddr
185	B9	cp c	res 7,c	cpdr
186	BA	cp d	res 7,d	indr
187	BB	cp e	res 7,e	otdr
188	BC	cp h	res 7,h	
189	BD	cp l	res 7,l	
190	BE	cp (hl)	res 7, (hl)	
191	BF	cp a	res 7,a	
192	C0	ret nz	set 0,b	
193	C1	pop bc	set 0,c	
194	C2	jp nz, nn	set 0,d	
195	C3	jp nn	set 0,e	
196	C4	call nz,nn	set 0,h	
197	C5	push bc	set 0,l	
198	C6	add a,n	set 0, (hl)	
199	C7	rst 0	set 0,a	
200	C8	ret z	set 1,b	
201	C9	ret	set 1,c	
202	CA	jp z,nn	set 1,d	
203	CB	—	set 1,e	
204	CC	call z,nn	set 1,h	
205	CD	call nn	set 1,l	
206	CE	adc a,n	set 1, (hl)	
207	CF	rst 8	set 1,a	
208	D0	ret nc	set 2,b	
209	D1	pop de	set 2,c	
210	D2	jp nc,nn	set 2,d	
211	D3	out (n),a	set 2,e	
212	D4	call nc,nn	set 2,h	
213	D5	push de	set 2,l	
214	D6	sub,n	set 2, (hl)	
215	D7	rst 16	set 2,a	
216	D8	ret c	set 3,b	
217	D9	exx	set 3,c	
218	DA	jp c,nn	set 3,d	
219	DB	in a,(n)	set 3,e	
220	DC	call c,nn	set 3,h	
221	DD	—	set 3,l	
222	DE	sbc a,n	set 3, (hl)	
223	DF	rst 24	set 3,a	
224	E0	ret po	set 4,b	
225	E1	pop hl	set 4,c	
226	E2	jp po,nn	set 4,d	
227	E3	ex (sp),hl	set 4,e	
228	E4	call po,nn	set 4,h	
229	E5	push hl	set 4,l	
230	E6	and n	set 4, (hl)	
231	E7	rst 32	set 4,a	
232	E8	ret pe	set 5,b	
233	E9	jp (hl)	set 5,c	
234	EA	jp pe,nn	set 5,d	
235	EB	ex de,hl	set 5,e	
236	EC	call pe,nn	set 5,h	
237	ED	—	set 5,l	
238	EE	xor n	set 5, (hl)	
239	EF	rst 40	set 5,a	
240	F0	ret p	set 6,b	

241	F1	pop af	set 6,c
242	F2	jp p,nn	set 6,d
243	F3	di	set 6,e
244	F4	call p,nn	set 6,h
245	F5	push af	set 6,1
246	F6	or n	set 6, (hl)
247	F7	rst 48	set 6,a
248	F8	ret m	set 7,b
249	F9	ld sp,hl	set 7,c
250	FA	jp m,nn	set 7,d
251	FB	ei	set 7,e
252	FC	call m,nn	set 7,h
253	FD	—	set 7,l
254	FE	cp n	set 7, (hl)
255	FF	rst 56	set 7,a

Tabela A3. *Instruções referentes aos registos de índex. As colunas 3 e 5 referem-se ao registo ix. As colunas 4 e 6 referem-se ao registo iy. Todas as instruções desta tabela são réplicas prefixadas das equivalentes para o par de registos hl, da tabela A2.*

Decimal	*Hex*	*Após 221 (hex DD)*	*Após 253 (hex FD)*	*Após 221,203 (hex DD,CB)*	*Após 253,203 (hex FD,CB)*
6	06			rlc (ix + d)	rlc (iy + d)
9	09	add ix, bc	add iy,bc		
14	OE			rrc (ix + d)	rrc (iy + d)
22	16			rl (ix + d)	rl (iy + d)
25	19	add ix,de	add iy,de		
30	IE			rr (ix + d)	rr (iy + d)
33	21	ld ix,nn	ld iy,nn		
34	22	ld (nn),ix	ld (nn),iy		
35	23	inc ix	inc iy		
38	26			sla (ix + d)	sla (iy + d)
41	29	add ix,ix	add iy,iy		
42	2A	ld ix,(nn)	ld iy,(nn)		
43	2B	dec ix	dec iy		
46	2E			sra (ix + d)	sra (iy + d)
52	34	inc (ix + d)	inc (iy + d)		
53	35	dec (ix + d)	dec (iy + d)		
54	36	ld (ix + d),n	ld (iy + d),n		
57	39	add ix,sp	add iy,sp		
62	3E			srl (ix + d)	srl (iy + d)
70	46	ld b, (ix + d)	ld b, (iy + d)	bit 0,(ix + d)	bit 0,(iy + d)
78	4E	ld c,(ix + d)	ld c,(iy + d)	bit 1,(ix + d)	bit 1,(iy + d)
86	56	ld d,(ix + d)	ld d,(iy + d)	bit 2,(ix + d)	bit 2,(iy + d)
94	5E	ld e,(ix + d)	ld e,(iy + d)	bit 3,(ix + d)	bit 3,(iy + d)
102	66	ld h,(ix + d)	ld h,(iy + d)	bit 4,(ix + d)	bit 4,(iy + d)
110	6E	ld l,(ix + d)	ld l,(iy + d)	bit 5,(ix + d)	bit 5,(iy + d)
112	70	ld (ix + d),b	ld (iy + d),b		
113	71	ld (ix + d),c	ld (iy + d),c		
114	72	ld (ix + d),d	ld (iy + d),d		
115	73	ld (ix + d),e	ld (iy + d),e		
116	74	ld (ix + d),h	ld (iy + d),h		
117	75	ld (ix + d),l	ld (iy + d),l		
118	76			bit 6, (ix + d)	bit 6, (iy + d)
119	77	ld (ix + d),a	ld (iy + d),a		
126	7E	ld a,(ix + d)	ld a,(iy + d)	bit 7, (ix + d)	bit 7,(iy + d)
134	86	add a,(ix + d)	add a,(iy + d)	res 0,(ix + d)	res 0,(iy + d)
142	8E	adc a,(ix + d)	adc a,(iy + d)	res 1,(ix + d)	res 1,(iy + d)
150	96	sub (ix + d)	sub (iy + d)	res 2,(ix + d)	res 2,(iy + d)
158	9E	sbc a,(ix + d)	sbc a,(iy + d)	res 3,(ix + d)	res 3,(iy + d)
166	A6	and (ix + d)	and (iy + d)	res 4,(ix + d)	res 4,(iy + d)
174	AE	xor (ix + d)	xor (iy + d)	res 5,(ix + d)	res 5,(iy + d)
182	B6	or (ix + d)	or (iy + d)	res 6,(ix + d)	res 6,(iy + d)
190	BE	cp (ix + d)	cp (iy + d)	res 7, (ix + d)	res 7, (iy + d)
198	C6			set 0,(ix + d)	set 0,(iy + d)
206	CE			set 1,(ix + d)	set 1,(iy + d)
214	D6			set 2,(ix + d)	set 2,(iy + d)
222	DE			set 3,(ix + d)	set 3,(iy + d)
225	E1	pop ix	pop iy		
227	E3	ex (sp),ix	ex (sp),iy		
229	E5	push ix	push iy		
230	E6			set 4,(ix + d)	set 4,(iy + d)
233	E9	jp ix	jp iy		
238	EE			set 5,(ix + d)	set 5,(iy + d)
246	F6			set 6,(ix + d)	set 6,(iy + d)
249	F9	ld sp,ix	ld sp,iy		
254	FE			set 7,(ix + d)	set 7,(iy + d)

Guia do Principiante do Spectrum

Escrito num estilo simples e prático, este guia proporciona o primeiro passo da aprendizagem no uso do ZX Spectrum.

Explica-se ao leitor como escrever programas simples, como usar variáveis numéricas e de texto, obter som, cores e formas gráficas.

Guia do Computador Pessoal

Guia profusamente ilustrado e com textos muito claros para os utilizadores de qualquer micro.

Se está a pensar em comprar um computador, se já o comprou ou se está simplesmente a tentar compreender a actual expansão dos computadores, o *Guia do Computador Pessoal* é o livro de que necessita.

BIBLIOTECA VERBO DE INFORMÁTICA

Jogos Dinâmicos para o ZX Spectrum

de *Tim Hartnell*

Este livro abre perspectivas novas aos apaixonados dos jogos de computador, que encontram nele os conselhos, artifícios, sugestões e técnicas indispensáveis a quem quiser aprender a construir os seus próprios programas de jogos.

Destes, os 20 que apresenta são, por si sós, fascinantes; cobrem toda esta área da informática, incluindo jogos de tabuleiro, de «arcada», de aventuras e ainda de palavras e outros, que representam um desafio à agilidade intelectual do jogador.

BIBLIOTECA VERBO DE INFORMÁTICA

Aprofundar o BASIC do Spectrum

de *Mike Lord*

Esta obra, de grande utilidade quer para principiantes quer para programadores experientes, é um magnífico complemento do manual do *Spectrum;* explica as características principais do BASIC Sinclair e mostra como podemos manejá-las para a criação de programas de jogos ou de aplicação autêntica.

Inclui mais de 50 programas completos, acompanhados de explicações minuciosas, além de numerosas pequenas rotinas e três apêndices informativos.

BIBLIOTECA VERBO DE INFORMÁTICA

O Domínio do Código Máquina no ZX Spectrum

de *Toni Baker*

Este livro põe à disposição do leitor os elementos essenciais da programação em código máquina e desfaz a falsa ideia de que esta linguagem não é para todos.

O código máquina é uma linguagem de computador, exactamente como é o BASIC.

Quem conhece instruções de BASIC compreenderá facilmente as instruções do código máquina e poderá vir a construir os seus programas de forma muito avançada, até atingir a incrível velocidade desta linguagem de alta resolução.

Próximo volume da

BIBLIOTECA VERBO DE INFORMÁTICA

Os 20 Melhores Programas para o Spectrum

de *Andrew Hewson*

Escrever bons programas é uma arte que se pode adquirir, e o propósito desta obra é ensiná-la através de exemplos.

Estes 20 programas, que ilustram inúmeras das técnicas mais proveitosas no mundo da informática, são todos de grande utilidade: indexação de ficheiros, gráficos de pontos e de funções, renumeração de linhas de programa, cálculo de comprimento de programa, carregador de código máquina, manipulação de imagem, de movimento, de cor, etc.

Trata-se de uma boa base prática para quem não quer depender de programação alheia.