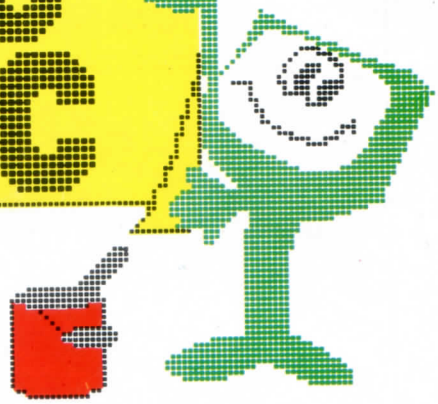
VIDEO BASIC



20 LIÇÕES DE BASIC
PARA APRENDER COM O SPECTRUM

EDIÇÕES LATINAS

JACKSON

Os modernos dispositivos de entrada: rato, trackball, ecran sensível

Software proficional: processadores de texto, folhas de cálculo, bases de dados

Instruções, comparações e saltos em código máquina

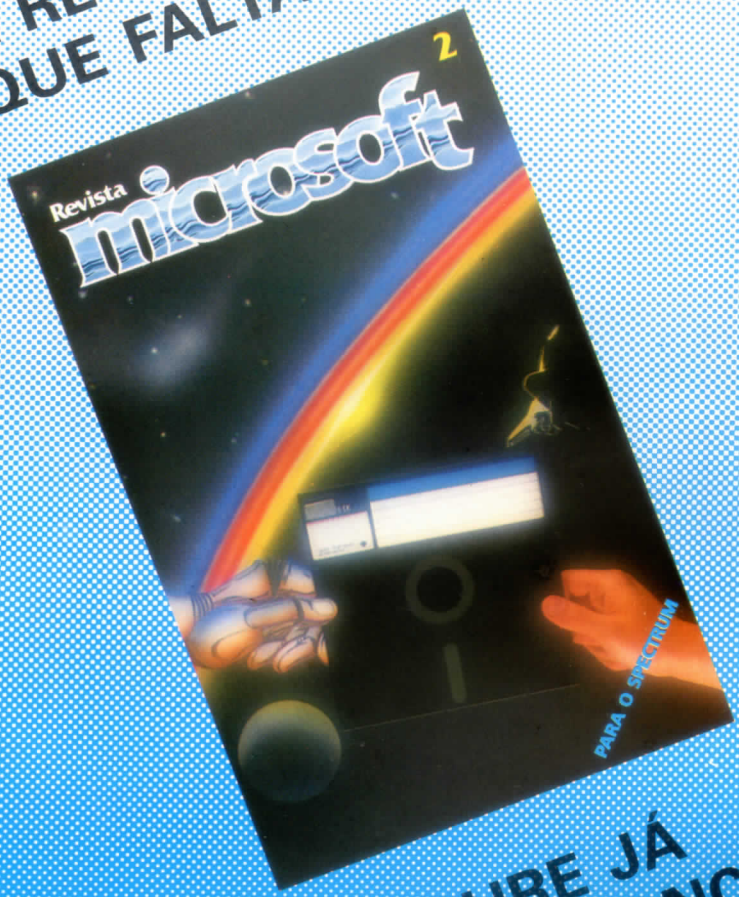
Ferramentas de programação

Videojogo n.º 19

19 Spectrum

16K/48K/PLUS

A REVISTA QUE FALTAVA!



PROCURE JÁ NA SUA BANCA

(PARA SPECTRUM)

VIDEO BASIC
Uma publicação de:
EDIÇÕES LATINAS-JACKSON
Director editor:
Manuel A. Lopes
Director editor da JACKSON ESPANHA:
Lorenzo Bertagnolio
Director de produção:
Vicente Robles
Autor:
Softidea
Redacção Software:
Francesco Franceschini
Stefano Cremonesi
Desenho gráfico:
Studio Nuovaidea
Ilustrações:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Edições Latinas, Lda.
Direcção, redacção e administração, números atrasados e assinaturas:
Av. Almirante Reis, 219, 3.º-Esq.
1000 Lisboa
Fotocomposição e impressão:
Antunes & Amílcar, Lda.

ISBN do tomo 1: 84-85831-12-8
ISBN do fascículo: 84-85831-11-X
ISBN da obra completa: 84-85831-10-1
Depósito Legal N.º: 11335/86
Plano geral da obra:
20 fascículos e 20 cassetes, de publicação quinzenal.
Edições Latinas-Jackson garante a publicação de todos os fascículos e cassetes que compõem esta obra e o fornecimento de qualquer número atrasado, até 1 ano, depois de terminada a sua publicação.
Está reservado ao editor, o direito de modificar o preço de venda dos fascículos durante a sua saída, se o mercado assim o exigir.

1.ª Quinzena – Agosto 1986

EDIÇÕES LATINAS
JACKSON

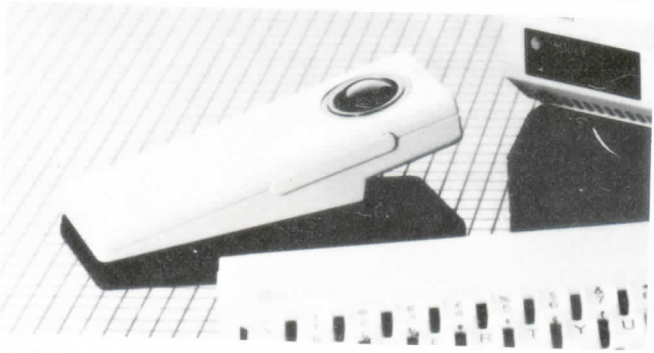
SUMÁRIO

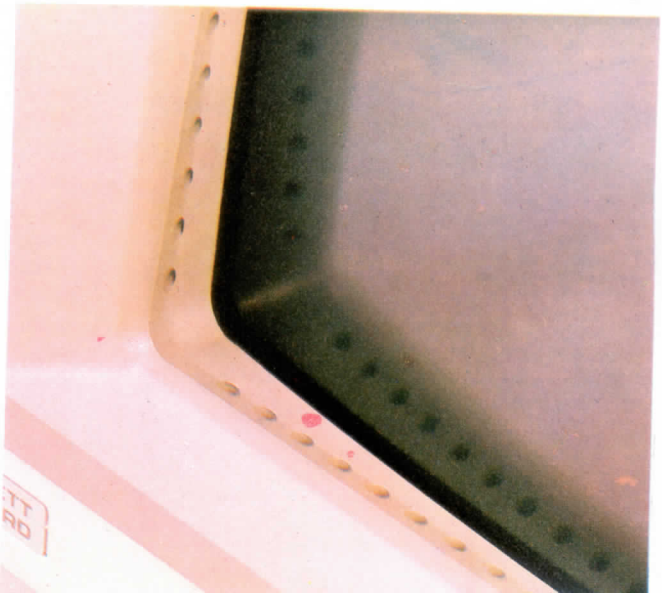
Introdução

Já se pode dizer que têm à vossa disposição todos os elementos necessários para se transformarem em bons programadores de BASIC. Mas, para conseguir resultados ainda mais surpreendentes do computador, fica apenas um caminho, o Código Máquina.
Felizmente não é preciso ter que programar tudo por nossa conta; muitas vezes é preferível, falando em termos de possiblidade reais, tempo e dinheiro: comprar directamente software standard.
Trataremos então dos processadores de textos, folhas de cálculo, bases de dados e demais "ferramentas" de utilidade geral.

Os modernos dispositivos de entrada

Como bem sabemos, o teclado constitui, para qualquer computador, um dispositivo de entrada extremamente importante, ou, melhor dizendo, fundamental. No entanto, existem muitos outros periféricos de entrada que permitem proporcionar ao computador, de uma maneira simples e imediata, todos aqueles dados que se desejem introduzir na sua memória Desde que o tratamento gráfico começou a ser parte do habitual campo de aplicações dos computadores pessoais, desenvolveram-se novos dispositivos para conseguir fazer mais fácil e imediato o colóquio entre homem e máquina. Os mais difundidos são o rato, o "trackball" e o ecran táctil.

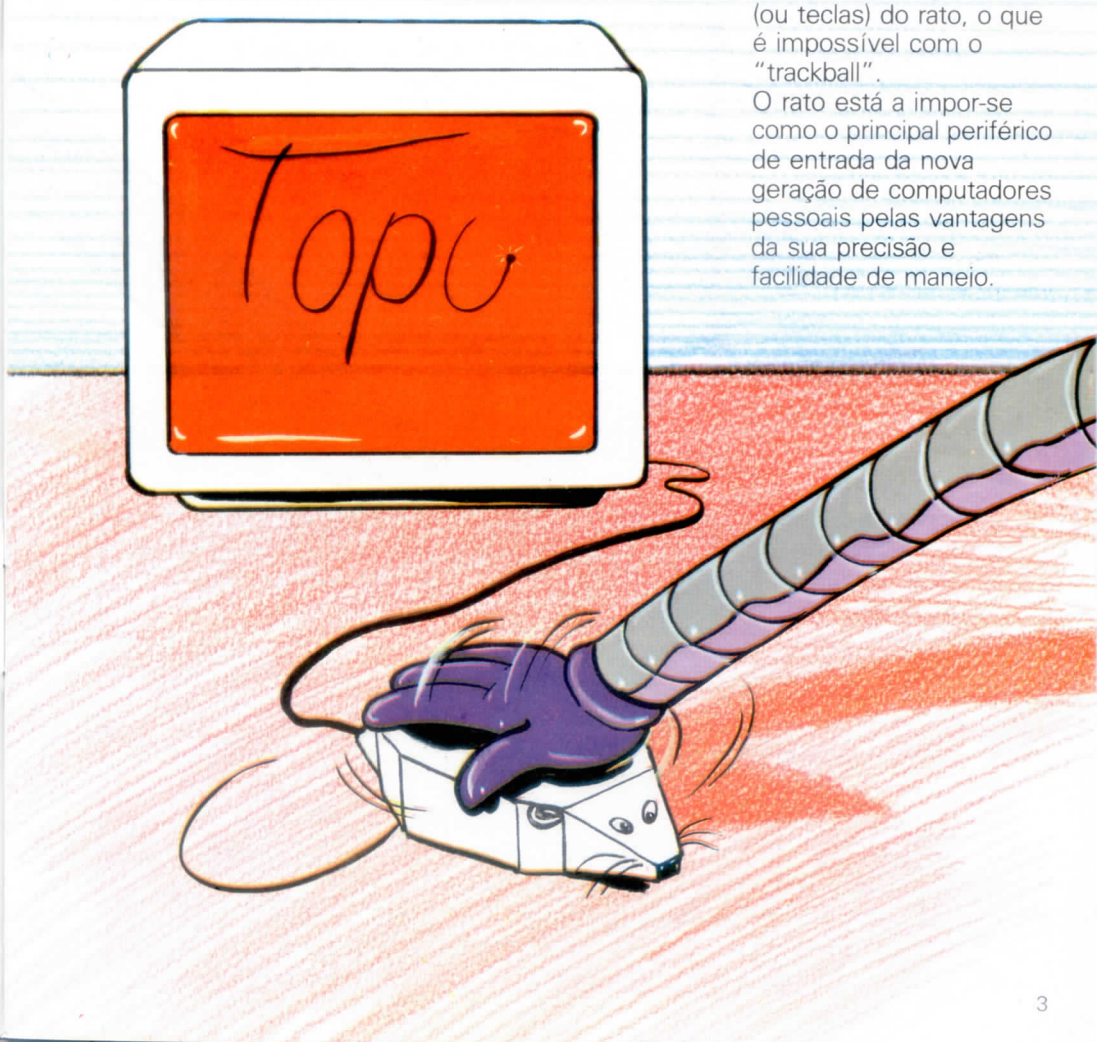




RATO

O rato ("mouse" em inglês) é um "trackball" virado ao contrário. Aqui faz-se rolar a esfera deslocando o dispositivo sobre um plano liso (por exemplo, uma mesa). O "mouse" precisa de um espaço maior, em relação ao "trackball", sendo necessário uma certa liberdade de movimentos para o poder manobrar; em compensação, a operação é muito mais intuitiva, dado que o deslocamento do cursor no ecran decalca com toda a fidelidade o que a mão vai realizando. Além do mais, é possível mover o cursor mantendo premida ao mesmo tempo a tecla (ou teclas) do rato, o que é impossível com o "trackball".

O rato está a impor-se como o principal periférico de entrada da nova geração de computadores pessoais pelas vantagens da sua precisão e facilidade de maneio.

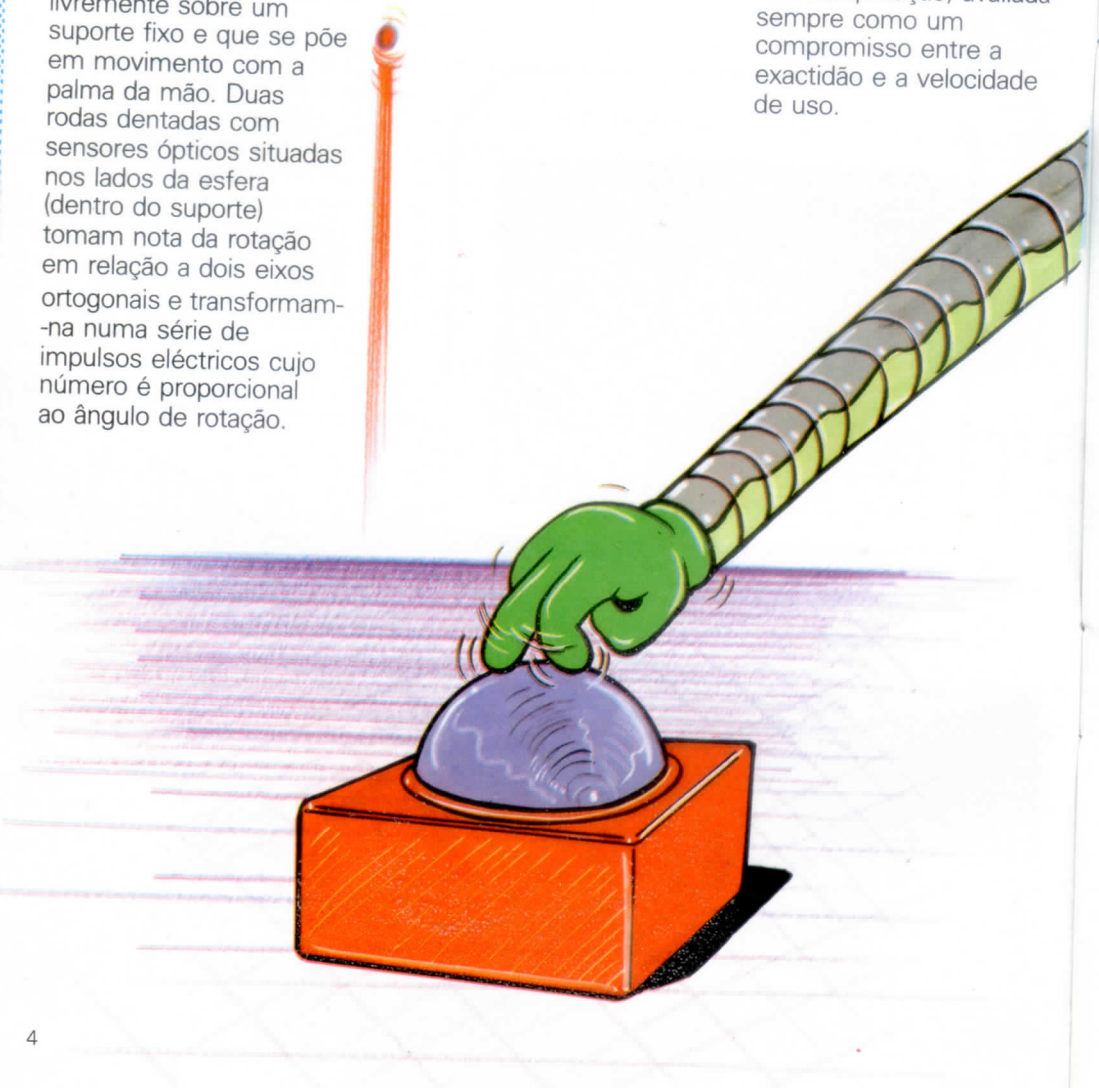


O "TRACKBALL"

O "trackball" (literalmente, "esfera traçadora") é um periférico constituído por uma bola completamente lisa que pode girar livremente sobre um suporte fixo e que se põe em movimento com a palma da mão. Duas rodas dentadas com sensores ópticos situadas nos lados da esfera (dentro do suporte) tomam nota da rotação em relação a dois eixos ortogonais e transformam-na numa série de impulsos eléctricos cujo número é proporcional ao ângulo de rotação.

Uma vez que, contrariamente ao que acontece com "joysticks" e "paddles" não existe uma extremidade, é possível obter toda a precisão que se deseje, embora, às vezes, se obtenha esta vantagem em detrimento da velocidade de manejo.

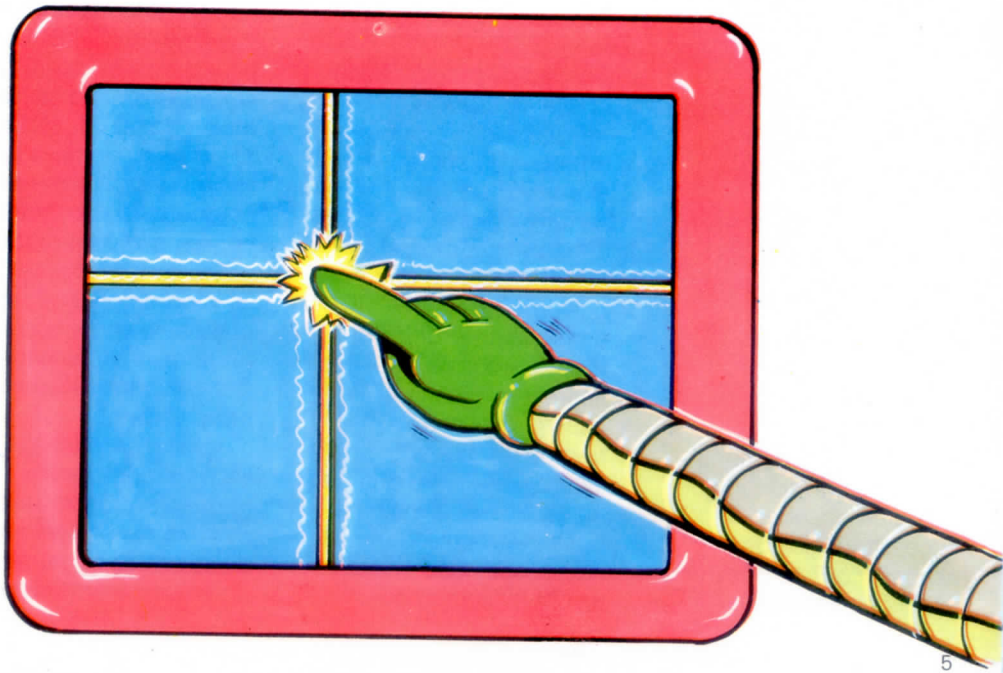
Para o software ficam as tarefas de "pilotagem" e a escolha da melhor relação de desmultiplicação, avaliada sempre como um compromisso entre a exactidão e a velocidade de uso.



ECRAN TÁCTIL

A ideia do ecran táctil é muito semelhante à do lápis óptico, o termo inglês "touch-screen" significa ecran sensível ao tacto.

Na realidade não é o ecran que é sensível ao toque do operador mas sim uma folha transparente que o cobre. Uma fina rede de sensores ópticos faz com que essa folha seja sensível aos dedos ou a uma caneta.
A maior desvantagem desta solução é a constituída pela pouquíssima resolução do dispositivo. Além do mais, e este é um inconveniente que não se pode desprezar, a película de gordura que existe sempre sobre a pele acaba por se acumular sobre o ecran, prejudicando a leitura.
No entanto, o uso deste periférico é extremamente natural e intuitivo, podendo, portantro, ser empregue por pessoas não muito experientes.



LINGUAGEM

Software auxiliar

Qualquer computador, por mais sofisticado e aperfeiçoado que seja, é incapaz de operar por si mesmo: é sempre necessário unir às características de hardware da máquina o software apropriado à resolução do problema ou problemas que se pretendem resolver.
No entanto, para conseguir este objectivo, o programador tem que realizar um conjunto de passos e operações que são constantemente variáveis. Portanto, é preciso analisar e avaliar o problema sob todos os seus aspectos, inclusivé os mais mínimos, examinando com atenção todas as dificuldades a superar. Uma vez estabelecido o planeamento do problema em termos gerais, as várias funções a realizar vão-se pondo sucessivamente em evidência mediante esquemas adequados (normalmente chamados – já o sabes – diagramas de fluxo), que permitem localizar a sucessão lógica de acções, com as possíveis situações alternativas e os eventuais momentos de decisão. Também nesta fase é preciso decidir se a totalidade do conjunto de funções a realizar se concentrará num único programa ou se se subdividirá em vários programas e os seus correspondentes subprogramas.
Uma vez estabelecidos os diagramas de fluxo, o programa passa finalmente à verdadeira fase de programação, realizada mediante a escrita das várias instruções numa linguagem que – como o BASIC – possa ser "compreendida" pela máquina.
Acabada a programação ocupar-nos-emos (após ter gravado previamente o programa sobre um suporte magnético seguro) de pôr em marcha a fase experimental.
Agora é quando chega o momento mais estimulante e difícil de todo o processo, quando aparecem todas os passos que nos demonstram as possíveis lacunas que uma análise apressada do problema possa ter deixado.
Nesta operação podem ajudar o programador aqueles conjuntos de programas normalmente chamados "software de ajuda", que lhe permitem inspeccionar, avaliar e corrigir os erros de uma maneira muito mais rápida e fácil.
Os programas de ajuda mais comuns ocupam-se de operações que, à primeira vista, parecem muito triviais; no entanto, no momento oportuno, são praticamente indispensáveis, especialmente quando se deseja aproveitar ao máximo as possibilidades do próprio computador.
Existem muitos programas deste tipo no mercado; também nas revistas especializadas costumam ser tidos em conta. Os mais completos TOOL-KIT ("tool" significa literalmente "ferramenta", porque estes programas podem ser considerados como as ferramentas do programador) permitem desenvolver numerosas operações, das quais destacaremos:

- **numerar** automaticamente as linhas do programa enquanto é escrito. Assim evita-se ter de escrever de cada vez os números de linha; naturalmente

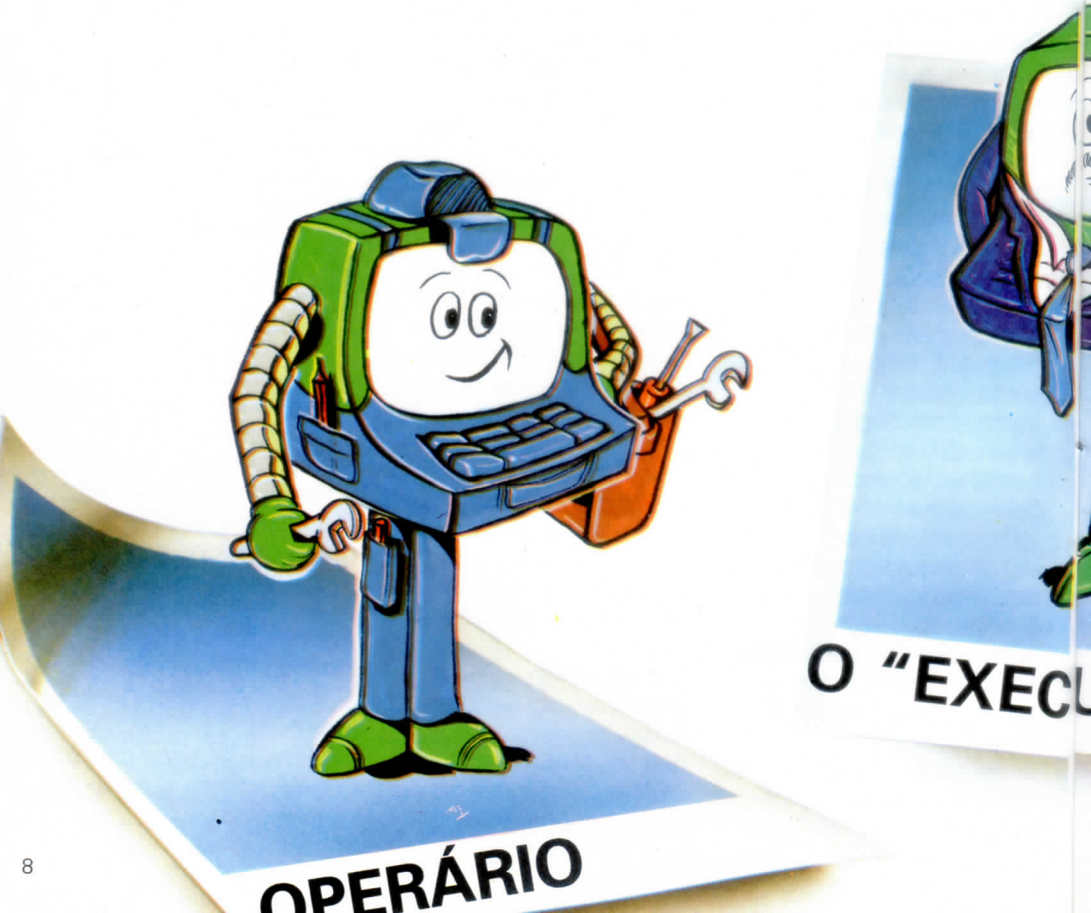
é possível especificar o passo, ou seja, o incremento que se deseja incluir entre uma linha e outra;
• **renumerar** um programa que tenha sofrido muitos ajustamentos. Esta é uma das possibilidades mais apreciadas pelos programadores, pois se não existissem estes programas, às vezes, já não poderiam inserir novas instruções.
É claro que a renumeração das linhas tem que implicar – pelo menos num programa profissional – também a renumeração correcta das várias instruções GOTO e GOSUB;
• **apagar** blocos de programa. Muitos computadores podem realizar esta acção: no entanto, com o sistema operativo do teu Spectrum só é possível apagar uma linha de cada vez. Com um programa de "delete" (apagamento) é possível indicar instruções do tipo "delete 10-130" (apaga todas as linha entre a 10 e a 130);
• **procurar** uma determinada cadeia de caracteres do programa. Desta maneira evita-se ter que procurar trabalhosamente a cadeia (por exemplo o nome de uma variável) ao longo de toda a listagem;
• **substituir** automaticamente uma cadeia de caracteres por outra;
• **visualizar** os números de linha do programa à medida que as instruções vão sendo executadas. Também esta possibilidade é de grande interesse e utilidade; é muito frequente que o programa funcione correctamente, ou seja, que proporcione resultados de saída, mas que estes não correspondam absolutamente aos que precisaríamos que aparecessem. Estes são os erros mais difíceis de localizar, uma vez que não afectam a sintaxe do programa mas sim a sua lógica. Com um programa de "trace" (traço) torna-se possível seguir, através da visualização progresssiva dos números de linha que vão sendo realizados, o desenvolvimento do programa em tempo real, isto é, no momento em que vai acontecendo cada coisa. Muitas vezes basta dar uma vista de olhos a estes números para sermos capazes de resolver um problema provocado por um salto a uma linha errada ou por uma sucessão de instruções não realizada.

Aplicações comerciais

Tal como acontece com a roupa, os programas também se podem fazer à medida ou ser pré-confeccionados. Como é natural, estes últimos custam menos.
Os programas projectados e escritos especificamente para a resolução de um determinado problema são aqueles que desempenham esta tarefa com a máxima velocidade, com o melhor aproveitamento das características do computador usado e com a máxima adequação às exigências de quem o tenha que usar.
As vantagens destes programas podem resumir-se em poucos, mas muito importantes, pontos:
– são realizados por profissionais e estão garantidos;

LINGUAGEM

– estão disponíveis em muito pouco tempo (isto é, não requerem longos prazos de espera);
– têm um preço bastante razoável, analisando também as possíveis desvantagens, a principal é que não podem satisfazer cem por cento as exigências do utilizador. Fica sempre uma certa margem de "insatisfação" devida ao facto de que um programa projectado para resolver um problema no seu conjunto não pode ter em conta todos os casos particulares possíveis. Por exemplo, um programa de contabilidade nunca poderá ser idêntico para um amador ou para uma empresa média; por esta razão muitos programas estão "abertos", ou seja, permitem uma margem mais ou menos larga de modificação para eventuais "personalizações". Em contrapartida, há exigências autenticamente comuns a um número muito grande de utilizadores, como a escrita e o arquivo de dados.

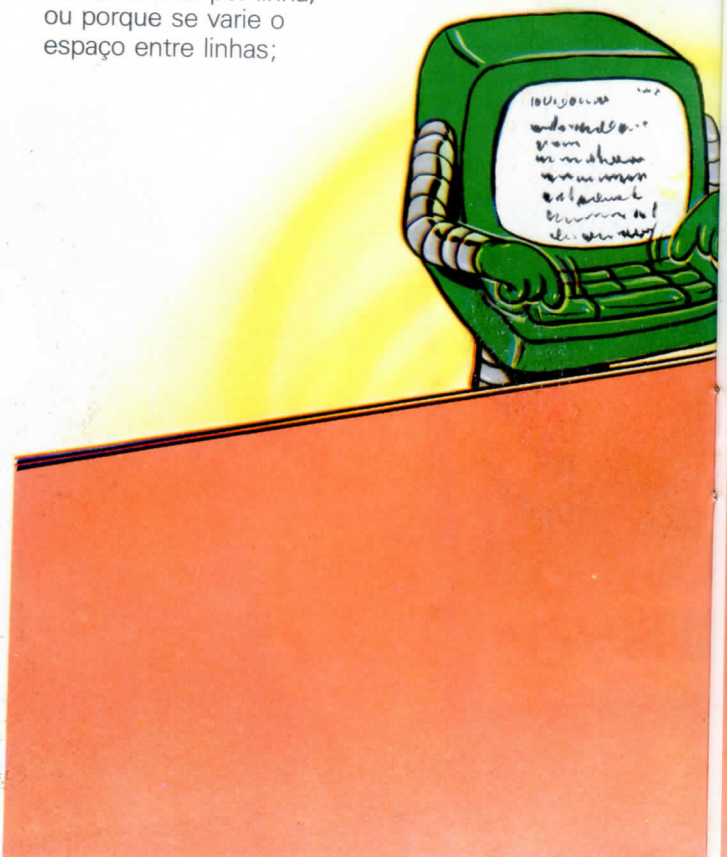


Assim, os mais importantes "pacotes de aplicações" disponíveis na actualidade são: os tratamentos de texto, as folhas de cálculo e as bases de dados.

Tratamento de textos

Um programa para o processamento de textos (do inglês "Word processor") é uma das possíveis aplicações que é capaz, por si só, de justificar a compra de um computador pessoal. A característica principal de um tratamento de textos é a de transformar o computador numa máquina de escrever muito especial. Permite escrever à primeira, e desordenadamente se se quiser, dado que existe a possibilidade de modificar qualquer parte do texto sem ter que voltar a escrevê-lo todo desde o princípio, para fazer uma cópia a limpo, e permite obter com a mesma

qualidade de impressão um número ilimitado de cópias.
O seu uso suprime de forma definitiva borrachas, materiais correctores, papel químico e outros inconvenientes.
Além do mais um sistema de tratamento de textos maneja automaticamente o alinhamento das palavras, tanto no meio das linhas como nas margens, que por sua vez se podem modificar à vontade com poucas e elementares intruções.
Mas a característica fundamental de um tratamento de dados é consentir a correcção "dinâmica" de tudo aquilo que foi escrito: pode "voltar-se atrás" e corrigir no ecran, deixando ao computador e à impressora o trabalho de voltar a escrever todo o texto no papel.
As funções mais importantes e necessárias de um sistema para o tratamento de textos são:
– retorno automático do carreto. Assim, não é preciso comprovar o fim de cada linha, visto que as palavras que excedem a sua capacidade são transferidas automaticamente para o princípio da seguinte;
– possibilidade de inserção de caracteres, palavras ou linhas, no texto já existente, para realizar correcções ou aumentos;
– apagamento de caracteres, palavras e frases com a eliminação automática do espaço novamente disponível em cada linha;
– possibilidade de colocação automática na página, seja porque se quer variar o número de caracteres por linha, ou porque se varie o espaço entre linhas;
– possibilidade de procura automática de uma palavra ou um grupo de palavras dentro do texto;
– possibilidade de copiar e transferir partes do texto para outro lugar do mesmo documento.
Em programas mais evoluídos existem outras



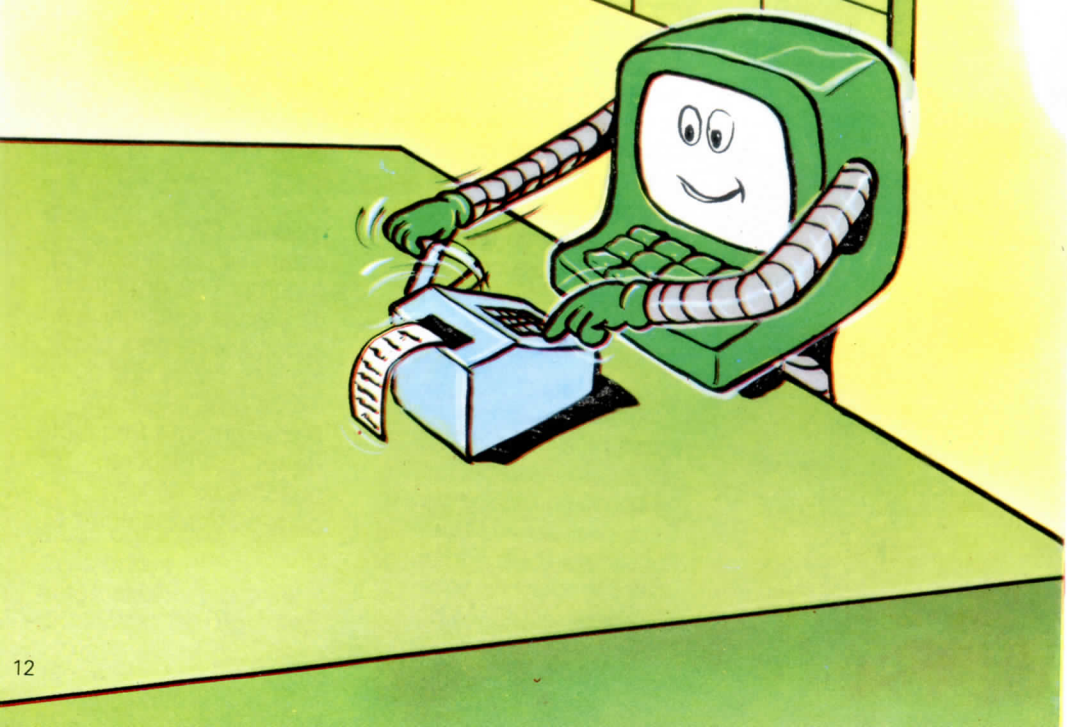
funções mais específicas que as indicadas, mas não de menor utilidade: o princípio geral de funcionamento continua a ser o mesmo. Como é natural, qualquer tratamento de textos permite guardar o texto num suporte magnético.

Folhas de cálculo

Uma das aplicações com maior êxito e difusão no sector dos computadores domésticos e pessoais, é, com certeza, o tratamento de folhas tipo mapa (aquelas que, isto para que nos possamos entender, estão organizadas em linhas e colunas) ou seja, a chamada folha de cálculo electrónico ou "worksheet" (e também "spreadsheet").
O uso prático destes sistemas é muito vasto e variado; desde as simulações de vendas em função dos níveis de produção, até à gestão de balanços pessoais ou familiares, ou o exame do orçamento dos custos, vendas e lucros de uma actividade comercial durante o decurso dos doze meses do ano.
Vejamos agora como funcionam.
Trata-se de um conjunto de linhas e colunas localizáveis mediante um código (por exemplo, letras para as colunas e números para as linhas, tal como na batalha naval). Cada coordenada pode conter um dado numérico, uma fórmula que ligue e elabore, algébrica ou logicamente, mais dados ou um conjunto qualquer de caracteres, como os componentes de um título. Entre as várias informações que podem aparecer em duas ou mais casas podem definir-se relações analíticas, algébricas ou de outro género (por exemplo, estatístico).
Feito isto, bastará introduzir mediante o teclado os vários valores das casas definidas como variáveis principais, para que todas as variáveis dependentes sejam automaticamente calculadas pelo programa e vão aparecendo nos quadros a que pertencem.
Desta característica deriva a capacidade das folhas de cálculo para serem usadas como potentes instrumentos de "simulação" para modelos de tipo estatístico ou financeiro.
A compra de uma folha de cálculo electrónica é aconselhável para todos aqueles casos nos quais exista uma exigência razoavelmente frequente de ter que elaborar quadros de dados correlacionados entre si.

LINGUAGEM

	JANEIRO	FEVEREIRO	MARÇO	ABRIL	MAIO	JUNHO
CASA						
GAS						
LUZ						
VIAGENS						
LETRAS						
TOTAL						



Base de dados

As bases de dados ou, em tradução livre, programas para a organização de dados, são programas de uso generalizado aptos a encontrar e memorizar dados e notícias, arquivar e classificar impressos ou conjuntos de informações, e assim sucessivamente.

Uma base de dados permite utilizar electronicamente qualquer tipo de arquivo de informações, disposto e estruturado conforme os desejos do utilizador. Um exemplo esclarecerá isto facilmente. Suponhamos que desejamos compor uma agenda telefónica electrónica, que contenham para além do nome, do apelido e do número telefónico de várias pessoas, também a morada e a cidade onde residem. Com uma base de dados, após ter introduzido os elementos, é possível ordenar a lista das pessoas, por exemplo, por ordem alfabética, por zonas de residência, pelo indicativo telefónico, ou por qualquer outra "chave".

Além do mais pode procurar-se imediatamente de uma maneira simples e automática "a pessoa que tem o indicativo X e que mora na cidade Y' Voltando aos aspectos gerais, as aplicações de uma base de dados são praticamente infinitas: tantas quantos os usos dos arquivos de informação. O importante é que estes programas permitem a utilização de qualquer tipo de informação, independentemente da raridade ou complicação, procurando-a e actualizando-a com a máxima rapidez e automatização.

Código máquina: os registos

O microprocessador que equipa o Spectrum (e, em geral, todas as CPU) dispõe de um determinado número de "localizações" especiais chamados registos, que para o programador de Código Máquina podem ser comparadas às variáveis do programador de BASIC.
No Z80 existem aproximadamente 10 registos, cada um dos quais é conhecido mediante uma letra do alfabeto; os principais são:

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

Cada registo pode ser considerado como uma localização especial da memória, que, consequentemente, pode conter um valor numérico compreendido entre 0 e 255. No entanto, existe (e essa é uma das vantagens do Z80) a possibilidade de agrupar em pares os registos HL, BC e DE, de maneira que se pode trabalhar com valores numéricos superiores a 255. Naturalmente, este facto não impede em nenhum caso o uso independente dos registos.
Examinaremos agora com maior atenção alguns dos registos do Z80, falando um pouco das funções envolvidas no seu uso, porque alguns deles, como veremos brevemente, estão especializados, e não é possível utilizá-los para qualquer circunstância.
– O registo A. Normalmente denominado "acumulador", é o registo mais importante do microprocessador. Ele é quem "acumula" a maior parte das tarefas de funcionamento da CPU. Em compensação aceita, ao contrário dos outros registos, todos os modos de direccionamento disponíveis (os modos de direccionamento ser-te-ão apresentados brevemente). Todas as instruções aritméticas, lógicas e de comparação usam-no como intermediário.
– O registo B. Este registo seria bastante simples se não fosse usado de uma maneira especial numa instrução de salto. Pode ser associado ao registo C para formar um registo duplo.
– Os registos C, D, E, H, L. São simples registos de 8 bits que podem ser usados aos pares: BC, DE, HL. Pode recorrer-se a eles muito rapidamente e geralmente são usados para conservar os dados. Cada um deles é privilegiado no que diz respeito a determinadas operações. Por exemplo, o registo C é usado nas operações de Entrada/Saída. Os registos D, E, H, L, usam-se, em contrapartida, para a transferência de dados.
– O registo F. Este registo é o "abandeirado" do microprocessador, no sentido em que cada um dos seus bits indica o tipo de resultado consequente a uma determinada operação. Os bits do registo de estado (numerados de 0 a 7, sete o da esquerda

e 0 o da direita) são especificados com nomes concretos que examinaremos brevemente.

- C, FLAG de carry, posição 0, contem o bit mais significativo da última operação realizada. Isto é, o bit de carry toma o resto gerado, no decurso de uma soma, da soma dos dois bits mais significativos.
- N, FLAG da subtração, posição 1, contém 1 se a última operação foi uma subtração ou diminuição.
- P/O, FLAG de Paridade/ Overflow, posição 2, tem um significado duplo. Nas operações aritméticas indica se existiu um transbordamento da capacidade (Overflow), invadindo a posição mais à esquerda do byte, usada para o sinal. Nas operações lógicas indica, pelo contrário, a paridade (tomando o valor 1 quando se verifica uma paridade par).
- AC, FLAG de carry auxiliar, posição 4, usa-se nas operações aritméticas executadas sobre números que se expressam mediante uma codificação especial. Trata-se de um "medio-carry", daí o seu nome. Dá as mesmas indicações do bit de carry, mas sobre um valor de só 4 bits.
- Z, FLAG de zero, posição 6, está em 1 se a última operação teve zero como resultado.
- S FLAG do sinal, posição 7, toma o valor do bit mais siginificativo do resultado da última operação lógica ou aritmética realizada pelo microprocessador.

Técnicas de direccionamento

O Z80 não dispõe de uma grande variedade de instruções (no conjunto são 67). Deste ponto de vista outras CPU são claramente superiores, chegando facilmente a ter um número de instruções quase duplo. Apesar disto, o Z80 é um dos microprocessadores mais difundidos. A razão é bem simples: deve-se à vasta possibilidade de técnicas de direccionamento disponíveis, o que aumenta o número efectivo de instruções para mais de 600. Vejamos em primeiro lugar o que entendemos pelo nome de "técnicas de direccionamento". Já comentámos que o Assembler é uma linguagem de baixo nível, pouco (ou melhor dizendo, nada) estruturado e dotado de formas lógicas verdadeiramente mínimas. Por outras palavras, o Assembler vale-se de instruções de uma extrema simplicidade operativa e executiva: tudo se baseia em números e localizações de memória. É

precisamente em virtude deste facto, e para poder trabalhar o melhor possível em qualquer parte da memória, que os construtores dos microprocessadores tentam introduzir a máxima flexibilidade nas operações executáveis, proporcionando à CPU a maior quantidade possível de técnicas de direccionamento.
O direccionamento faz referência ao modo como deve ser considerado modo operativo numa determinada instrução, o que influi no próprio resultado da operação.
Examinemos agora, um por um, os vários direccionamentos disponíveis, avaliando as suas possibilidades, os seus usos, as suas vantagens e as suas desvantagens.

Direccionamento directo sobre um registo

O operando é um registo. Assim:

```
LD A,B
```

significa "carrega em A o conteúdo de B".
As instruções de direccionamento sobre um registo ocupam únicamente um byte; consequentemente, não somente são breves, mas também permitem uma alta velocidade.
O tempo necessário para a sua execução limita-se a apenas 4 ciclos, que no Spectrum correspondem a menos de um milionésimo de segundo.
Nota: por ciclo entende-se o tempo que a CPU precisa para realizar uma operação elementar. Este tempo é medido inflexivelmente por um oscilador (uma espécie de relógio de quartzo em miniatura). Tem em conta, a título de informação, que o Z80 executa mais de 4 milhões de operações elementares num único segundo, e isto durante todo o tempo que o mantiveres ligado.

Direccionamento implícito

O código operativo "implica" a direcção do operando ou de um dos operandos: RRA serve para executar uma operação para a direita de um bit do registo A.
Bit 0,A examina, em contrapartida, o bit de posição 0 do registo A.

Direccionamento imediato

Esta forma de direccionamento utiliza-se para carregar o acumulador ou qualquer outro registo com um valor específico.
Portanto, todas as instruções no modo de direccionamento imediato tem um comprimento de dois bytes. O primeiro contém o código operativo e o segundo contém a constante numérica a carregar no registo ou utilizar para uma instrução aritmética ou lógica. Por exemplo, se nós quiséssemos carregar no acumulador o valor 160 (A0) hexadecimal em modo

imediato, poderíamos escrever:

```
LD A, A0H
```

Esta instrução diz: "carrega no acumulador o valor hexadecimal A0"

Direccionamento directo

O direccionamento directo é o modo no qual os dados são normalmente recuperados desde a memória. Está especificado por um código operativo, seguido por uma direcção de 16 bits encerrada entre parêntesis.
O direccionamento directo precisa, portanto, de três bytes. Um exemplo de direccionamento directo é:

```
LD A, (1234H)
```

Esta instrução específica que o conteúdo da localização de memória número 1234 deve ser memorizado no acumulador.
A desvantagem do direccionamento directo é que precisa instruções de três bytes: trata-se, portanto, de um modo bastante lento.

Direccionamento indirecto mediante registos

Para melhorar a eficácia do direccionamento directo pode utilizar-se outro modo, o modo indirecto mediante registo.
Com este direccionamento, um par de registos contém a direcção na qual se encontra o valor a elaborar, isto é, o operando; o par de registos indica-se entre parêntesis.
Na gíria informática diz-se que os registos "apontam" à localização de memória. Um exemplo deste modo é:

```
LD A, (HL)
```

que significa "carrega em A o conteúdo do byte cuja direcção se encontra em HL".
O direccionamento indirecto mediante registo é notavelmente mais veloz que o directo, uma vez que a CPU não deve ler a direcção de memória, que se encontra já nos registos.
Naturalmente, é preciso carregar no par de registos a direcção desejada, portanto, este método é vantajoso somente quando a direcção vai ser utilizada mais de uma vez.

Direccionamento relativo

Este método de direccionamento é bastante utilizado para as instruções de salto (isto é, para os comandos que transferem a execução de um ponto para outro do programa). Por definição, o direccionamento relativo utiliza dois bytes: o primeiro é uma instrução de salto, enquanto que o segundo especifica o deslocamento e o seu sinal.
O que significa isto? Por exemplo, suponhamos que queremos mandar a execução de um certo ponto do programa a outro diferente. Para fazer isto devemos especificar (para além, como é natural, das instruções de salto) o número de localizações que queremos saltar. Uma vez que a deslocação pode ser positiva ou negativa (isto é, pode acontecer tanto para a frente como para trás), a instrução de salto – que deve afectar no máximo 255 localizações – pode ser negativa, no máximo até 128, ou positiva, até 127 (salto para a frente de 127 localizações). Naturalmente, a maior desvantagem deste modo é que não permite saltos superiores a 128 localizações. No entanto, uma vez que a maior parte das instruções têm tendência a ser breves, este tipo de desvio pode ser utilizado quase sempre e ajuda a resolver o problema.

Direccionamento indexado

O direccionamento indexado é uma técnica especialmente útil para ter acesso aos dados contidos num grupo de localizações, um após outro. O princípio consiste em que a direcção desejada é calculada somando à direcção contida num dos registos índice (os registos índice IX e IV são outros dois pares de registos do Z80) o valor do operando. Assim

```
LD E, (IX+5)
```

carrega no registo E o conteúdo do byte que contenha a direcção formada somando o número 5 ao conteúdo do par de registos IX.
Todo os modos de direccionamento (para simplificar omitimos alguns, está bem?) requerem em qualquer caso, para ser adequadamente compreendidos e utilizados da melhor maneira, bastante tempo e (principalmente) muito trabalho de programação. Os exemplos que veremos daqui a pouco ilustrarão algumas possíveis aplicações passíveis dessas técnicas.

Incremento/ diminuição

A operação aritmética mais simples que o Z80 é capaz de executar é a operação algébrica de adição e subtração do valor 1 ao valor contido num registo. À primeira vista, esta possibilidade parece não abrir grandes horizontes: na realidade, a disponibilidade de uma operação deste tipo permite a construção de estruturas muito complexas como por exemplo as instruções FOR... NEXT do BASIC.
A importância das instruções de adição e subtração é tão grande que existem duas instruções específicas para levar estas missões a cabo.
Destas instruções existem duas maneiras distintas disponíveis com diferentes modos de direccionamento:

ADD	instrução de soma sem resto
ADC	instrução de soma com resto
SUB	instrução de diminuição sem resto
SBC	instrução de diminuição com resto

Todas estas instruções podem influenciar alguns bits do registo de estado, concretamente, o bit de sinal negativo e o bit de zero.
O bit de sinal coloca-se em 1 se o valor mais significativo do registo (isto é, o que ocupa a posição 7) se coloca a 1 em função do incremento ou da diminuição; de outra maneira, vale zero.
O FLAG de zero coloca-se em 1 somente se, após uma execução, o registo que foi usado cotém o valor zero.
Por exemplo, aumentando em 1 o valor FFH (255 hexadecimal) contido num determinado registo, elevar-se-á a 00H o valor do registo, colocando-se em 1 o FLAG de zero (Z=1) e anular-se-á o FLAG de sinal negativo.
Por outro lado, diminuindo um registo que contenha 00H, o valor do registo ficará FFH, pondo simultaneamente a 1 o FLAG de sinal e a zero o FLAG de zero.

LINGUAGEM

As instruções

As instruções são uma parte importantíssima – mais ainda, vital – em qualquer linguagem de programação: em Assembler ainda o são mais, uma vez que muitas instruções, que em linguagem de alto nível precisam de um único comando para a sua execução, estas precisam, pelo contrário, sequências e repetições mais ou menos longas em Código Máquina.
Antes de passar à demonstração prática do uso das instruções em Assembler é necessário, no entanto, apresentar dois outros grupos de instruções: as de comparação e as de salto.

As comparações

É possível comparar o valor contido no acumulador do Z80 com o conteúdo de uma localização da memória (ou com um valor numérico qualquer) mediante as instruções.

CP	compara o valor
CPI	compara o valor com incremento
CPD	compara o valor com diminuição

O resultado da comparação altera o FLAG de zero (Z), de sinal (N) e de resto ou carry (C) do registo de estado.
De que modo? A resposta está nesta pequena tabela:

ACUMULADOR	CARRY	ZERO	SINAL
menor que o dado	1	0	1
igual ao dado	0	1	1
maior que o dado	0	0	1

Comparando, por exemplo, o valor do acumulador (suponhamos que seja 5AH) com o número B3H, obtemos através da tabela que o valor das várias FLAG será:

C=1 Z=0 N=1

Com base nestes resultados poderemos então tomar as decisões mais apropriadas.

Os saltos

Uma vez efectuada a operação de comparação pode ser necessário executar também uma operação de salto (assim como em BASIC escrevemos, por exemplo, IF A>B THEN GO TO 300).
Podemos distinguir três grupos de saltos:
– os saltos propriamente ditos
– as chamadas aos subprogramas
– o retorno desde os subprogramas.
Também há duas modalidades de saltos:
– absolutos
– relativos.
Uma instrução deste tipo:

```
JP ADR
```

chama-se salto incondicional; provoca o salto sistemático à direcção ADR (indicando com ADR uma direcção genérica da memória). É o equivalente, em Código Máquina, ao GOTO que é usado em BASIC. Estas outras instruções são, em contrapartida, exemplos de saltos condicionais:

```
JP C,ADR        JP NC,ADR
JP Z,ADR        JP NZ,ADR
JP M,ADR        JP P,ADR
JP PE,ADR       JP PO,ADR
```

"Salto condicional" significa que o salto deve ser efectuado somente se a condição requerida foi satisfeita.
Naturalmente, as condições de salto condicional podem apresentar-se num programa somente atrás de instruções que posicionem de qualquer maneira as FLAG.
A condição de salto é comprovada com base nos valores assumidos por qualquer destas FLAG.

```
....
CP 8            ; compara A com 8
JP Z, ADR1      ; salta se A=8
JP C, ADR2      ; salta se A<8
....            ; então A>8
```

Neste exemplo, a instrução CP posiciona

os flag Z e C. O salto condicional JPZ comprova a presença do flag Z e provoca o salto à direcção ADR1, se Z está a 1, assinalando desta maneira que o conteúdo do acumulador é igual a 8. Caso contrário o programa prossegue e encontra a instrução JPC, que comprova o estado do flag C. Se este está a 1 efectua-se um salto à direcção ADR 2; desta forma podemos estar certos de que o conteúdo de A é maior que 8, uma vez que não satisfaz o teste precedente e o programa continua sequencialmente.

Até agora vimos como operando da instrução de salto a instrução JP. No entanto, é possível efectuar um salto não somente para uma direcção dada mas também para uma direcção calculada, contida num dos registos HL, IX ou IY.

```
JP (HL)
JP (IX)
JP (IY)
```

Esta modalidade de salto torna-se muito útil em muitas ocasiões. Os comandos de SALTO RELATIVO distinguem-se dos outros por uma característica especial: especificam uma deslocação relativa, ou seja, para a frente ou para trás de +127 localizações ou de −128 bytes, em relação à posição ocupada nesse momento. As instruções de salto são:

```
JR      VAL
JR C,   VAL     JR NC, VAL
JR Z,   VAL     JR NZ, VAL
```

VAL representa um valor numérico que indica — em modo relativo — a direcção para a qual saltar.
por exemplo:

```
JR Z, −14
```

significa: "se o resultado de comparação (que já se terá realizado) pôs o flag Z a 1, então excuta um salto para trás de 14 bytes".

Além das instruções propriamente ditas de salto, existe uma última instrução de salto: CALL. O CALL do Assembler é equivalente ao GOSUB do BASIC. Esta instrução efectua um salto ao subprograma cuja direcção está especificada no operando, não sem ter efectuado antes uma gravação na área do stack da direcção de volta. Este último ponto é o que deferencia CALL de uma simples JP, apesar da semelhança entre ambas ser notável.

Para poder voltar dos subprogramas em Código Máquina, tal como no BASIC é necessário recorrer ao RETURN, é preciso, em contrapartida uma instrução RET. Neste caso o microprocessador extrai

um valor da área de stack e devolve a execução do programa à direcção correspondente ao valor tomado.
Terminada, finalmente, a parte teórica, passaremos agora à prática, isto é, às aplicações dos conceitos que vimos até agora.
Começamos com um exemplo muito fácil...: suponhamos que queremos escrever um caracter (por exemplo, um "A") no ângulo superior esquerdo do ecran. Sabemos já como se faz em BASIC: basta um PRINT AT e o assunto está resolvido.
Mas agora devemos esquecer-nos destas comodidades e tentar fazer tudo com os nossos meios. Primeiro que tudo é preciso recordar o mecanismo através do qual os caracteres podem aparecer no ecran: cada posição do ecran é composta por uma "rede" de 8x8 quadradinhos. Cada grupo de 8 quadradinhos dispostos sobre a mesma fila correspondem a um byte; cada caracter escreve-se com 8 bytes. Para visualizar no ecran um determinado caracter bastará, portanto, memorizar nas 8 localizações correspondentes à posição escolhida, os 8 valores que o identificam. Dado que a memória do ecran começa na localização 16384, para visualizar o nosso A em BASIC poderemos fazer

```
POKE 16384,0
POKE 16640,60
POKE 16896,66
POKE 17152,66
POKE 17408,126
POKE 17664,66
POKE 17920,66
POKE 18176,0
```

Executando estas instruções em modo imediato, veremos aparecer, um bocadinho da cada vez, o nosso caracter A. Examinaremos agora como fazer em Código Máquina.
O procedimento não mudará muito: será necessário carregar em memória os vários valores:

```
LD A,0
LD (4000H),A
LD A,60
LD (4100H)A
LD A,66
LD (4200H),A
LD (4300H),A
LD A,126
LD (4400H),A
LD A,66
LD (4500H),A
LD (4600H),A
LD A,0
LD (4700H),A
```

Carregamos o Código Máquina usando o habitual programa BASIC:

```
10 CLS:RESTORE:CLEAR 32500
20 LET PRINCIPIO=32500:LET FIM=33000
30 FOR X=PRINCIPIO TO FIM
40 READ A
50 IF A=999 THEN GO TO 80
60 POKE X,A
70 NEXT X
80 RANDOMIZE USR 32500: STOP
90 DATA... aqui escrevem-se os códigos...
```

Para inserir os códigos no programa em Código

Máquina na linha DATA devemos executar a transformação das instruções menemónicas nos correspondentes valores numéricos:

ASSEMBLER	CÓDIGOS HEXADECIMAIS	CÓDIGOS DECIMAIS
LD A,0H	3E,0	62,0
LD (4000H),A	32,00,40	50,0,64
LD A,3CH	3E,3C	62,60
LD (4100H)	32,00,41	50,0,65
LD A,42H	3E,42	62,66
LD (4200H),A	32,00,42	50,0,66
LD (4300H),A	32,00,43	50,0,67
LD A,7EH	3E,7E	62,126
LD (4400H),A	32,00,44	50,0,68
LD A,42H	3E42	62,66
LD (4500H),A	32,00,45	50,0,69
LD (4600H),A	32,00,46	50,0,70
LD A,0H	3E,0	62,0
LD (4700H),A	32,00,47	50,0,71

Recordando além do mais, que o nosso programa BASIC, para não se ver interrompido por uma mensagem de erro, precisa que o último valor seja o número 999. Executando o programa, verás aparecer no ecran, como antes, a letra "A", na posição correspondente às posições afectadas pela rotina em Código Máquina.
O exemplo que acabamos de ver é bastante interessante mas não é muito prático: o comprimento da rotina

em Assembler parece excessivo para um trabalho assim tão simples.
Para quê especificar a forma do caracter "A", dado que na própria memória ROM o Spectrum já sabe todas as suas características? Tentaremos então recorrer a uma segunda solução. Para imprimir o caracter no ecran devemos primeiro saber duas coisas:

1) a direcção onde estão memorizados os 8 números próprios do caracter que desejamos imprimir
2) a direcção (ou antes, as direcções) que correspondem a uma determinada posição do ecran.
No caso da letra A, e da primeira posição no canto superior esquerdo, estas direcções são, respectivamente, a 15880 e a 16384.

Para imprimir o caracter bastará simplesmente tomar 8 vezes um byte da "memória de caracteres" e depositá-lo numa determinada localização da "memória de ecran".
Eis aqui o aspecto de uma possível solução:

```
LD HL,3E08H
LD BC,4000H
LD A,8H
PUSH AF
LD A,(HL)
LD (BC),A
INC HL
INC B
POP AF
SUB 01
JR NZ, –10
RET
```

O programa começa por memorizar nos dois pares de registos HL e BC as direcções da memória de caracteres e da memória de ecran. Além do mais o acumulador é carregado com o valor 8 (LD A,8A). Aqui é onde começa a instrução propriamente dito.

1) PUSH AF serve para memorizar numa zona segura do microprocesador (chamada área do STACK) o valor do acumulador.
2) Carrega-se o registo A com o valor contido na direcção HL. A instrução LD,A(HL) é um exemplo de direccionamento indirecto mediante registos.
3) Memoriza-se A na localização para a qual apontam os registos BC.

```
LD (BC),A
```

4) Incrementam-se as direcções para as quais apontam HL e BC.
5) Recuperam-se (POP) da área do Stack o valor do acumulador anteriormente memorizado.
6) Se subtraindo 1 ao acumulador ainda não se chegou a 0, então volta-se a começar o ciclo.
A instrução JR NZ,–10 é um salto relativo condicional para terminar o ciclo. O desvio efectua-se quando os FLAG N e Z estão em 1.
Como deves ter reparado esta rotina é muito mais breve que a anterior e, em suma, bastante mais elegante.
Os códigos numéricos obtidos com a transformação das instruções em código menemónico são:

```
33, 8, 62, 1, 0, 64, 62, 8, 245, 126, 2, 35, 4, 241, 214, 1, 32, 246, 201.
```

PROGRAMAÇÃO

Ferramentas de programação

Dado que cada vez nos introduzimos mais no interior do Spectrum podemos começar a examinar alguns "Programas de utilidades", truques e breves rotinas, que sempre ajudarão em muitas e várias circunstâncias.
Vejamos em primeiro lugar, o que é preciso fazer para impossibilitar o apagamento da primeira linha de um programa; este truque, sem nenhuma utilidade prática aparente, será muito útil a todos aqueles que querem que na primeira linha dos seus programas apareça uma instrução REM que contenha o nome do autor ou qualquer outro tipo de aviso.
O segredo é bem simples: bastará, depois de ter estabelecido a linha, comunicar a seguinte instrução em modo imediato:

```
POKE 1+PEEK23635+256*PEEK 23626,0
```

Esta técnica atribuirá à linha o número 0, de maneira que já não haverá meio de a mudar ou apagar. O progresso que permite esta operação vem do facto que, nas localizações 23635 e 23636, é guardada a direcção do programa BASIC.
Com um mínimo de prática (ou, melhor dizendo, de experiência) descobrirás que trocando o 0 de POKE por outro valor, se podem conseguir outros efeitos, diferentes do que acabamos de indicar, mas igualmente interessantes.
Vejamos agora um programa útil, que permite transformar um número qualquer, escrito numa determinada base (A), no mesmo número com outra base (B).
Devido à maneira como o programa foi planeado, a máxima base que se pode manipular é 36.
Os números superiores a 9 estão representados por letras do alfabeto (tanto em maiúsculas como em minúsculas, dado que o programa não as diferencia); assim, por exemplo: os doze algarismos de um sistema de base 12 serão:

0 1 2 3 4 5 6 7 8 9 A B.

Eis aqui a listagem:

```
 10 REM TRANSFORMAÇÃO DE BASES
 20 INPUT "DESDE QUE BASE?"; A
 30 INPUT "PARA QUE BASE?"; B
 40 PRINT "DESDE A BASE:";A,"PARA A BASE:";B
 50 INPUT "QUE NÚMERO DESEJAS
    TRANSFORMAR?" LINE A$
200 REM DE BASE E PARA BASE 10
210 LET S=0
220 FOR C=LEN (A$) TO 1 STEP -1
230 LET D=CODE(A$(C))-48-(7 AND A$ (C)>
    ="A")-(32 AND A$(C)>="a")
240 LET S=S+D*A  (LEN A$-C)
250 NEXT C
300 REM DE BASE 10 PARA BASE B
310 LET B$="":PRINT
320 LET R=INT(S-B*INT(S/B)+0.5):LET S=INT(S/
B)
330 LET B$=CHR$(R+48+(7 AND R>9))+B$
340 IF S<>0 THEN GOTO 320
350 PRINT A$;TAB 12;CHR$(61);TAB 16;B$;TAB 9
360 GO TO 10
```

O coração do programa está nas linhas 230-240 e 320-330. As expressões matemáticas que podes ler nestas instruções derivam da maneira como o Spectrum codifica os caracteres (de facto, o número a transformar é lido como uma cadeia de caracteres, não como um número). Finalmente, poderás observar que a passagem da base A para a base B se efectua mediante a transformação intermédia do número da base A para a base 10.

Examinaremos agora outro programa, de género completamente diferente do anterior, mas não por isso menos útil: trata-se de uma rotina de renumeração. Como já deves saber, um programa de renumeração é de grande utilidade para a programação, para poder reordenar automaticamente todos os números de linha no momento em que nos comecem a oferecer problemas por causa da sua confusão.
A versão que propomos é do tipo "amador", no sentido em que a reordenação é efectuada excluindo os GOTO e os GOSUB (que, portanto, será preciso modificar manualmente). Por outro lado se tivéssemos

incluído a renumeração dos saltos, o programa ter-se-ia tornado muito mais longo e complicado.

```
9000 REM RENUMBER
9010 STOP
9020 INPUT "INDICA O NÚMERO DA PRIMEIRA
     LINHA A RENUMERAR",NIR
9030 INPUT "INDICA O NOVO NÚMERO
     DE LINHA INICIAL",NLI
9040 INPUT "INDICA O INTERVALO
     ENTRE LINHAS",ITL
9050 LET IM=PEEK(23635)+256*PEEK(23636)-1
9060 LET NX=PEEK(IM+1)*256+PEEK(IM+2)
9070 IF NX<NIR THEN GOTO 9120
9080 IF NX=9000 THEN GOTO 9140
9090 POKE (IM+1),INT(NLI/256)
9100 POKE (IM+2),NLI-256*INT(NLI/256)
9110 LET NLI=NLI+ITL
9120 LET IM=IM+4+PEEK(IM+3)+256*
     PEEK(IM+4)
9130 GO TO 9060
9140 LIST
```

O programa tem que ser posto em andamento com RUN 9020 ou com GOTO 9020. Aparecerão as legendas da entrada, que pedem a indicação do número de linha desde o qual tem que começar a renumeração (por exemplo, 5) depois do novo número de linha inicial (por exemplo, 100) e finalmente da distância, isto é, o intervalo entre as linhas. Naturalmente, também é possível efectuar mais do que uma renumeração, por exemplo, para ter as primeiras 20 linhas renumeradas como 1000, 1050, 1100, etc.

Por comodidade, o programa pode ser memorizado em cassete com a instrução SAVE "RENUMBER" e pode ser carregado em memória, quando te seja necessário, com MERGE "RENUMBER", o que permite inserir a rotina na memória sem que se apague o programa já existente.

Para usar o RENUMBER o programa a renumerar não precisa usar as linhas da 9000 à 9140; além do mais é preciso reservar para o funcionamento da rotina as variáveis IM, NIR, NLI, ITL, NX.

A título de curiosidade (mas não só) mostraremos agora uma versão de RENUMBER escrita em Código Máquina; esta desempenha também o mesmo trabalho que o programa em BASIC que acabamos de ver, ou seja, ocupa-se simplesmente de renumerar as linhas sem se ocupar da possível presença de GOTO, GOSUB e RESTORE.

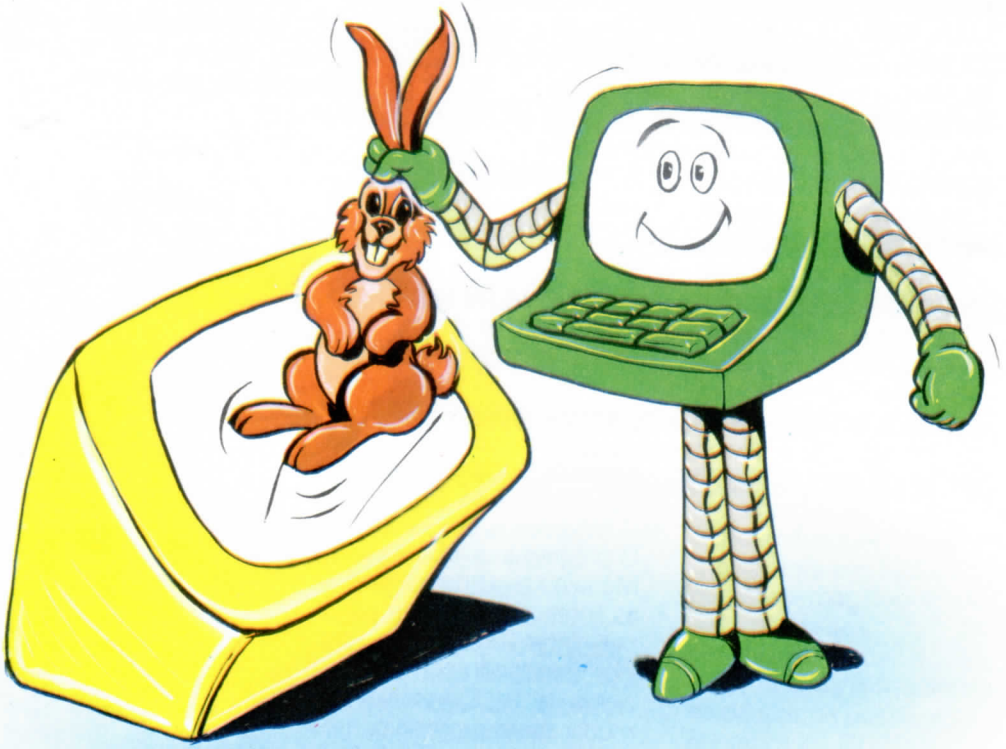
No entanto, é espectacular a extrema velocidade com a qual leva a cabo a tarefa; além do mais a versão em código é muito mais curta do que a correspondente em BASIC.
Eis aqui a listagem em Assembler:

```
LD DE, 5CCAH    ; princípio do BASIC
LD HL,90        ; número princípio-ciclo
INC DE
LD A, (DE)      ; o que é?
CP 28H          ; fim de linha?
RET NC          ; se não, terminado
LD BC, 10       ; dimensão do ciclo
AAD HL, BC      ; número nova linha
EX DE, HL       ; substituição momentânea
LD (HL), D      ; insere o novo número de linha
INC HL
LD (HL), E
INC HL          ; toma o comprimento da linha
LD C, (HL)      ; coloca-a em BC
INC HL
LD B, HL
ADD HL, BC      ; posição de fim de linha
EX DE, HL       ; fim substituição
JR -21          ; faz a próxima linha
```

O programa em Código Máquina renumera todas as linhas, começando pela linha 100 e incrementando com ciclos de 10. É possível mudar tanto o número do princípio, modificando o argumento da segunda linha (actualmente 90 = 100-10), como o incremento, substituindo com outro número o valor 10 da instrução
LD BC, 10.
Para carregar a rotina em memória propomos-te agora um pequeno programa BASIC, que se encarrega de pôr as instruções em Código Máquina numa área onde a rotina não poderá ser sobrescrita.

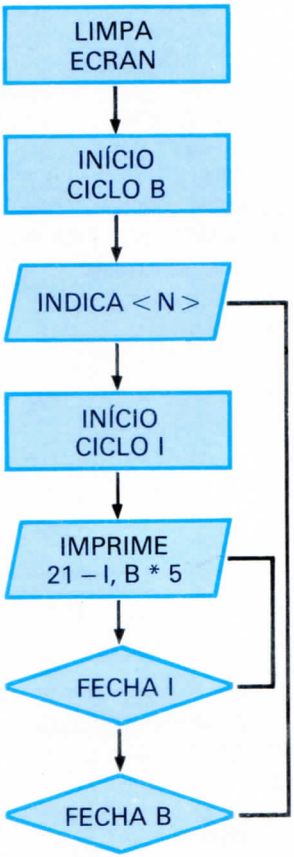
```
100 CLEAR 32499: LET A=32500
110 READ N: IF N=999 THEN STOP
120 POKE A,N
130 LET A=A+1
140 GO TO 110
150 DATA 17,202,92,33,90,0,19,26,254,40,208,1
160 DATA 10,0,9,235,114,35,115,35,78,35,70,9
170 DATA 235,24,235,999
```

Após ter introduzido o programa em memória bastará executá-lo para ter a rotina em memória. Para fazer executar a renumeração podes escrever em modo imediato uma instrução USR, como PRINT USR 32500 ou LET B=USR 32500.
Lembra-te, os GOTO e os GOSUB não se renumeram!



Representação gráfica

Eis aqui um programa para gerar um diagrama de barras. O ecran visualiza até 5 elementos verticais capazes de representar valores compreendidos entre 0 e 21. O caracter utilizado para imprimir as barras obtém-se premindo CAPS SHIFT e 8 em modo gráfico (cursor **G**). Observa o estabelecimento das instruções automáticas, que se podem observar facilmente no diagrama de fluxo.

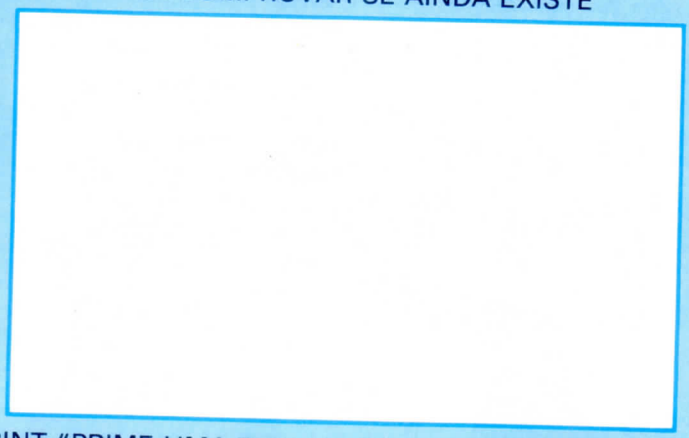



```
10 CLS
20 FOR B=1 TO 5
30 INPUT "N=";N
40 FOR I=0 TO N
50 PRINT AT 21-I, B*5; "■"
60 NEXT I
70 NEXT B
```

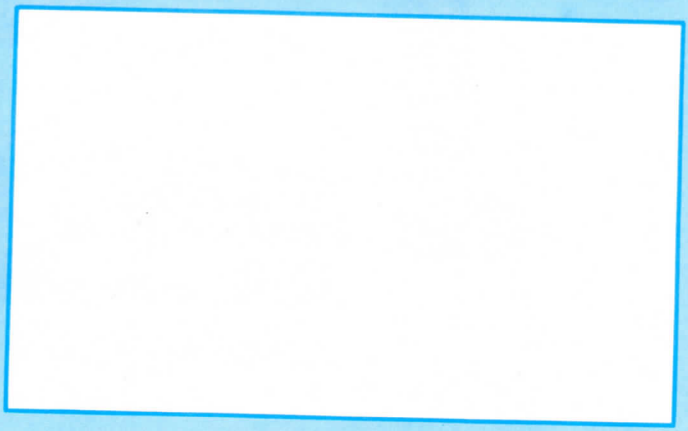
EXERCÍCIOS

Introduz as seguintes listagens tendo em atenção o que se diz nas instruções REM.

```
10 PRINT "ADIVINHA O QUE FAZ A ROTINA DA ROM SITUADA
   NA DIRECÇÃO 4757"
20 PRINT: PRINT "PARA O SABERES PRIME UMA TECLA"
30 PAUSE 1: PAUSE 0: CLS
40 RANDOMIZE USR 4757
50 > REM NO FIM PRIME LIST PARA COMPROVAR SE AINDA EXISTE
   A LISTAGEM
```



```
10 CLS: BEEP 0.5,30: PRINT "PRIME UMA TECLA": PAUSE 1: PAUSE 0
20 REM OUTRA MANEIRA DE EFECTUAR UMA INSTRUÇÃO
30 REM PARA SABER QUAL SÓ TENS QUE EXPERIMENTAR
40 REM BASTA UM RUN
50 REM NO FIM PRIME UMA TECLA
60 PRINT USR 6137
70 PAUSE 1: PAUSE 0
```



BOLETIM DE ASSINATURA

Envie a **EDIÇÕES LATINAS – VIDEOBASIC**
Av. Almirante Reis, 219, 3.º-Esq. • 1000 LISBOA

Desejo subscrever o curso completo de VIDEOBASIC por 7.500$00.

Junto envio cheque n.º_____ Banco_____

Nome

Morada

Cód. Postal Cidade

(Em vez de cortar o cupão tire uma fotocópia)

TC 2048/TC 2068
DOIS COMPUTADORES DISTINTOS
A MESMA TECNOLOGIA AO SERVIÇO
DA MICRO-INFORMÁTICA

- **OFERTA NATAL**
  (1 RELÓGIO PULSO + 1 RELÓGIO PAREDE)

TIMEX COMPUTER 2048

- Compatível SPECTRUM
- Teclado circuito impresso
- Porto para Joystick tipo KEMPSTONE incorporado
- Saída para Monitor Vídeo composto
- Interruptor ON/OFF
- LED de sinalização
- Barra de espaços
- Ficha de expansão para periféricos com sinal R.G.B.
- Dupla resolução gráfica
- Sistema de disco — (pode suportar, controlando até 4 Drives de 3", 140 K/face (formatado), com porto RS 232 para comunicação/série)
- Armazenamento de informação:
  - Gravador de cassettes
  - Microdrive
  - Sistema de Floppy Disk

TIMEX COMPUTER 2068

Na linha do TC 2048 e compatível com ele, apresenta-se mais potente graças à adição de novas funções, sintetizador de Som e Porto para Cartridges. Este permite-lhe carregar instantaneamente programas em ROM, sem necessidade de recorrer ao tradicional gravador de cassetes.

- Cartridges disponíveis:
  - ANDROIDS • CRAZY BUGS • BUDGETER
  - FLIGHT SIMULATOR • CASINO

  brevemente.
  - PROCESSADOR DE TEXTO/GESTOR DE LEITOR DE CÓDIGO DE BARRAS
- Compatível SPECTRUM através de Cartridge emuladora
- Novas funções: STICK/ON ERR/FREE/DELETE/SOUND
- 3 Canais de som independentes e programáveis

- **OFERTA NATAL**
  (1 RELÓGIO PULSO + 1 RELÓGIO PAREDE)

TABELA COMPARATIVA

	SPECTRUM +	TC 2048	TC 2068
Microprocessador	Z80-A	Z80-A	Z80-A
RAM física	48K	48K	48K
RAM utilizável	41472 Bytes	41472 Bytes	38652 Bytes Modo Spectrum 41772 Bytes
ROM	16K	16K	24K
Écran	24×32 ou 24×64	24×32, 24×64, 24×80	24×32, 24×64, 24×80
Resolução (Pixel)	256×192	256×192, 512×192	256×192, 512×192
Som	Beep	Beep	Beep e gerad. programável
Joysticks	Não	1 Joystick	2 Joystick
Solid State Software	Não	Não	Sim
Gravador	Vulgar	Vulgar	Vulgar
Disk Drive	Pode suportar	Pode suportar	Pode suportar
TV	UHF 36	UHF 36	UHF 36
Vídeo composto	Não	Sim	Sim
Interruptor ON/OFF	Reset	Sim	Sim
Cores	8 + Bright = 15	8 + Bright = 15	8 + Bright = 16
Teclado circuito impresso	Não	Sim	Sim
Circuitos integrados	26	15	15
Manual em português	Não	Sim	Sim
Garantia	6 meses	1 ano	1 ano

MONITOR VIDEO COMPOST MONOCROMÁTICO (FÓSFORO VERDE) NEPTUN 156

- Impedância de entrada: 75 OHM
- Resolução: 520 pontos/linhas
- Alimentação: ~ 220 V ou 12 VDC
- Écran: 31 cm (12")

TIMEX PRINTER 2080 IMPRESSORA MATRICIAL

- Ligação a qualquer computador com saída RS 232 C
- Papel A4 ou banda contínua
- Cópias múltiplas
- Capacidades gráficas
- Velocidade de impressão: 100 caracteres/s
- 7 categorias impressão gráfica
- 130 caracteres

TIMEX