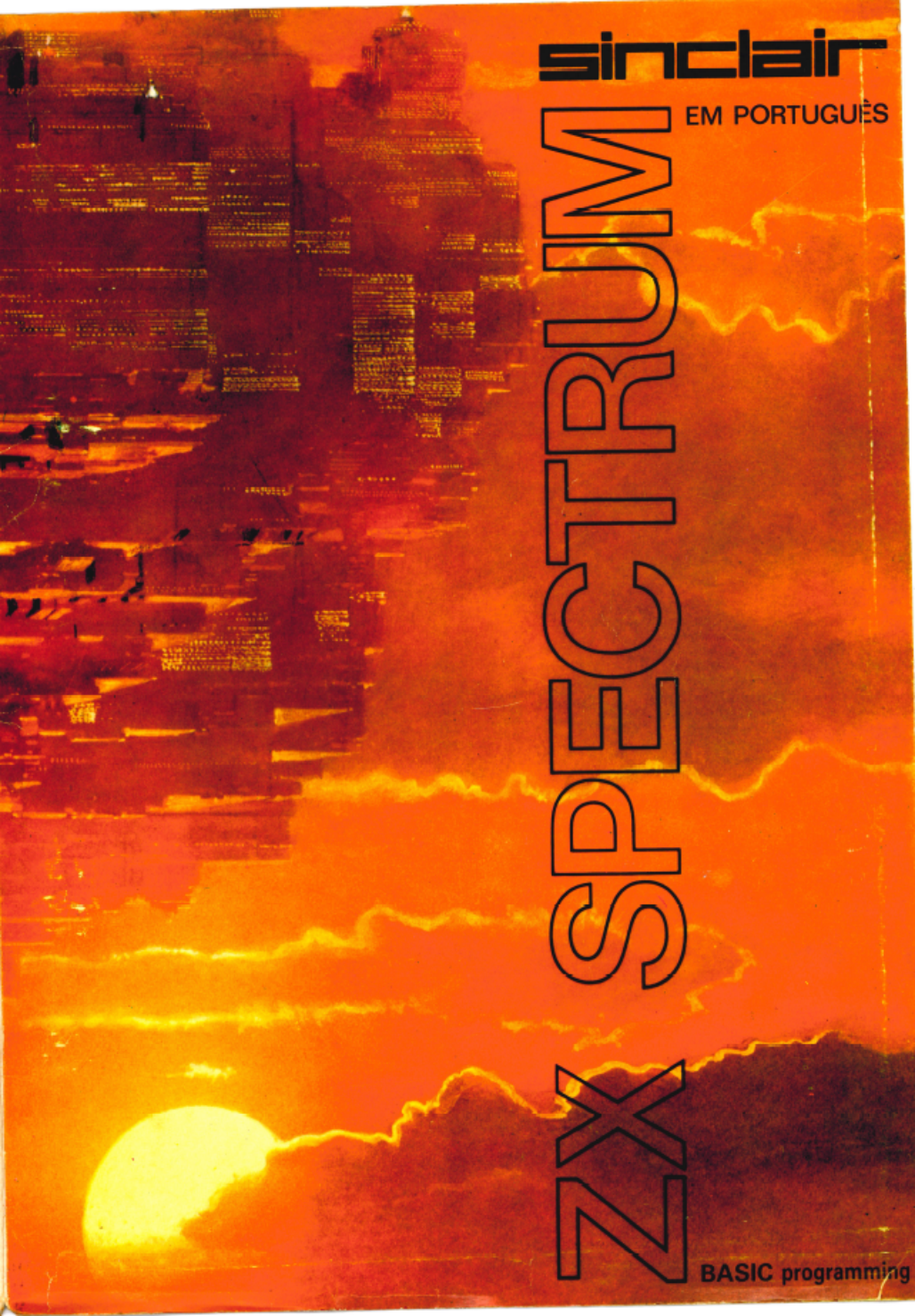

sinclair
EM PORTUGUÊS
ZX SPECTRUM
BASIC programming

sinclair

ZX SPECTRUM

INTRODUCTION

by Steven Vickers
and Robin Bradbeer

First Edition 1982

PREFÁCIO

Mais uma vez a Landry empenhou os seus esforços para que todo o Português tenha acesso à informática.

A Landry, como pioneira dos microcomputadores em Portugal, oferece este manual juntamente com o Computador, na tentativa de também nas escolas se começar a ensinar microinformática, abrindo assim as portas do "futuro", que já é presente, à juventude.

A Gerência

Carlos M. Nogueira

1. O COMPUTADOR E A SUA INSTALAÇÃO

Este pequeno livro foi escrito para dois tipos de pessoas. Primeiro para aqueles que não sabem nada, ou quase nada, acerca de computadores, e, em segundo lugar, para aqueles que estão familiarizados com sistemas à base de computadores mas que gostam de ler os livros de instruções antes de ligar qualquer coisa.
Há um segundo livro, mais grosso, que é o manual de programação em BASIC. Este não deve ser lido pelos novos utentes do computador sem que este pequeno livro tenha sido lido e compreendido.

Quando abrir a caixa do ZX Spectrum encontrará:

1. Este livro indrodutório e o manual de programação em BASIC.

2. O computador. Este tem três entradas de jack (marcadas com 9V DC IN, EAR, e MIC), uma entrada de TV, e uma ligação de bordo na parte de trás onde se pode ligar equipamento extra. Não tem interruptores - para o pôr a funcionar basta ligá-lo à fonte de alimentação.

3. Uma fonte de alimentação. Esta converte a electricidade da rede para uma forma que o ZX Spectrum usa. Se quiser usar a sua própria fonte de alimentação, ela deve fornecer 9 volts DC a 1,4A sem regulação.

4. Um cabo para antena com cerca de 2 metros de comprido e que liga o computador à televisão.

5. Um par de cabos com cerca de 75 cm de comprimento com fichas jack de 3,5 mm em cada ponta. Estes cabos ligam o computador a um gravador de cassetes.

Também vai precisar de uma televisão - o ZX Spectrum pode funcionar sem ela, mas não poderá ver o que se passa ! Tem de ser uma televisão com UHF (em Portugal); se ela não fôr capaz de receber a RTP-2, então não serve. Como o nome indica, o ZX Spectrum emite um sinal a cores que, se a sua televisão fôr a cores, produzirá uma imagem a cores. Se só tiver uma televisão a preto e branco, então a côr apa-

.../

recerá como negro, branco e seis tonalidades diferentes de cinzento; mas além disso uma televisão a preto e branco funcionará tão bem como uma televisão a cores.

Os componentes do sistema devem agora ser ligados da seguinte forma:

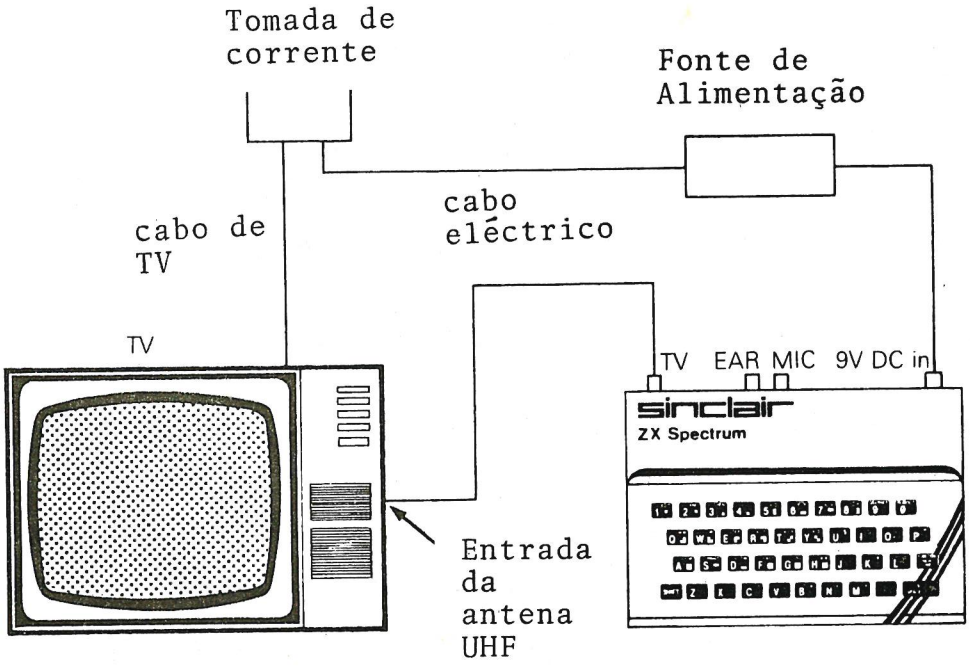


Figura 1

Se a sua televisão tiver duas entradas para antena, uma marcada para UHF e outra para VHF, então deve usar a de UHF.

Ligue à corrente e ligue a televisão. Agora precisa sintonizar a televisão. O ZX Spectrum opera no canal 36 de UHF, e quando é acabado de ligar e está correctamente sintonizado a imagem ficará assim:

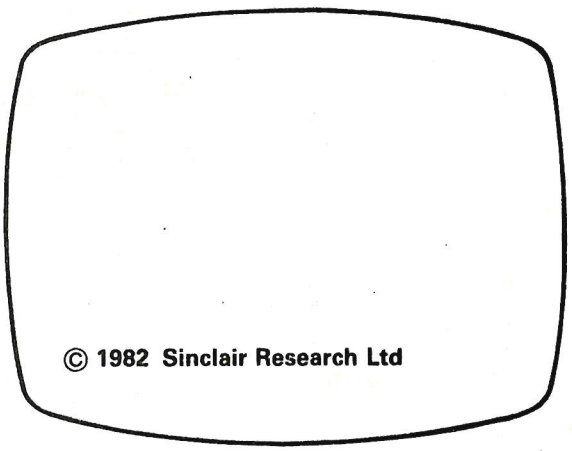


Figura 2

.../

Quando estiver a usar o computador, provavelmente quererá desligar completamente o volume de som no televisor.
Se a sua televisão tiver um controle de sintonização continuo, terá de o ajustar por forma a obter a imagem que se mostra na figura 2. Muitas televisões agora têm canais individuais para cada estação. Escolha um que ainda não esteja usado e sintonize-o.
Para ser usado em países que têm um sistema de TV diferente do da Inglaterra uma versão do ZX Spectrum especialmente desenhada para esse sistema é necessária. A inglaterra utiliza um sistema UHF com 625 linhas e 50 imagens por segundo. Também usa um sistema de codificação de cores denominado PAL. A maior parte dos países da Europa Ocidental (excepto a França) usam um sistema semelhante e nesses países o computador poderá funcionar sem qualquer modificação. Os Estados Unidos, o Canadá e o Japão, por exemplo, usam um sistema de TV completamente diferente e uma versão diferente do computador também terá de ser utilizada.
Quando desligar o ZX Spectrum toda a informação nela contida desaparecerá. Um modo de a guardar para ser usada mais tarde é gravá-la num gravador de cassetes. Também pode comprar cassetes que outras pessoas tenham preparado e assim correr programas que elas fizeram. O cabo com duas fichas jack em cada ponta pode ser usado para ligar um gravador de cassetes normal ao ZX Spectrum. O capitulo 8 deste folheto fornece mais explicações sobre este processo.
Agora que já instalou o computador, vai querer usá-lo. O resto deste folheto explicar-lhe-á como o fazer; mas com a sua impaciência provavelmente já começou a premir teclas do teclado, e já deve ter descoberto que fazê-lo apaga a mensagem de copyright do ecran. Isto é bom; não vai avariar o computador assim. Seja ousado. Experimente. Se ficar encravado lembre-se que pode sempre levar o computador à imagem inicial com a mensagem de copyright retirando a ficha de '9V DC IN' e voltando a ligá-la. Não faça isto senão como última oportunidade porque perderá toda a informação no computador.

ATENÇÃO. Não tente utilizar o modulo ZX 16K RAM com o ZX Spectrum. Não funciona.

.../

2. O TECLADO

O teclado do Spectrum é muito semelhante ao de uma máquina de escrever vulgar. As teclas de letras e de números estão no mesmo lugar; no entanto cada tecla pode executar mais do que uma função. Numa máquina de escrever normal as le - tras aparecem como pequenas, e quando usadas conjuntamente com a tecla de mai- usculas, aparecem como letras grandes. No teclado do Spectrum passa-se o mesmo. Para o ajudar a reconhecer em que modo é que o teclado está a trabalhar, apa - rece no écran uma letra invertida (branco sobre negro), que indica a posição do próximo caracter que aparecerá quando se tocar uma tecla. A letra pisca para a distinguir de qualquer outro caracter já no écran. Chamar-lhe-emos cursor. Quando se põe em funcionamento o Spectrum aparece no ecran a mensagem de copy- -right. Tocar em qualquer tecla faz com que apareça a palavra que está inscrita na tecla por baixo da letra (chamada de palavra chave). Isto acontece porque o computador está à espera que lhe seja dado um comando que lhe diga o que deve fazer e todos os comandos têm de começar com uma palavra chave.
Ao contrário do que acontece com a maior parte dos outros computadores o Spectrum permite-lhe introduzir as palavras chaves carregando somente numa tecla. Por exemplo, se carregar na tecla P imediatamente depois de ligar, a palavra chave PRINT aparece no écran. O simbolo " também está marcado na tecla P. Para o obter tem de carregar simulteneamente em duas teclas; carregue na tecla SYMBOL SHIFT, que está ao fundo do lado direito do teclado, e enquanto o faz pressione também a tecla P.

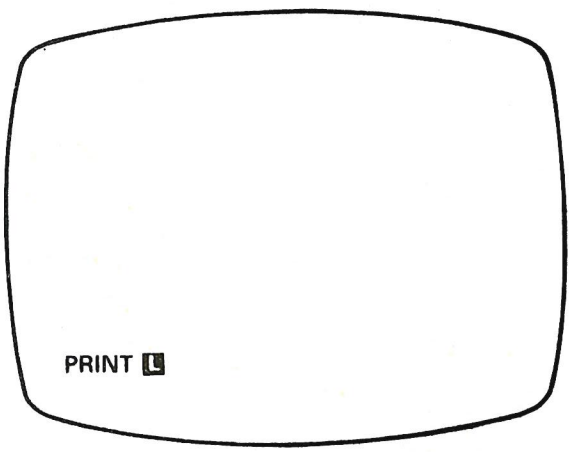


Figura 3

.../

O cursor agora muda para um **L** , porque agora é uma letra que o computador está à espera. Introduza as letras "Ola". Se já houver qualquer outro texto, por exemplo, no écran, desligue o computador (retire a ficha 9V) e recomece Use a tecla **CAPS SHIFT** para obter um O grande. Em geral qualquer coisa que esteja impresso a branco acima das teclas precisa da tecla **CAPS SHIFT** para poder ser escrita, e qualquer coisa que esteja escrita a vermelho na tecla requer a tecla **SYMBOL SHIFT**

Um comando que comece com **PRINT** diz ao computador para escrever as letras que estão entre aspas no écran. Para este comando ser executado pelo computador tem de ser usada também a tecla **ENTER** Quando tiver feito tudo isto o écran deve mostrar a palavra

Ola

e alguns outros caracteres. (Um ponto de interrogação a piscar indica um erro algures. Se isto acontecer recomece e repita o exercício). A mensagem na parte de baixo é na verdade o computador a indicar que tudo está bem ('Ok'). A mensagem é importante quando se estão a correr programas mas pode ser ignorada de momento.

Repare noutra coisa: A letra O e o número Ø são representados por caracteres diferentes. É importante que se lembre disto. O numero Ø tem sempre um traço a cortá-lo. O computador interpretará sempre a letra O como uma letra, por isso não se engane na tecla a usar. Também o número 1 e a letra pequena L são diferentes e, ao contrário do que acontece em algumas máquinas de escrever, não podem ser trocados.

Como é extremamente importante compreender-se em que modo de utilização está o teclado, vamos resumir novamente o que acontece.

O caracter **L** a piscar é chamado cursor. Mostra em que posição do ecran é que o computador vai colocar a proxima coisa que fôr introduzida. Não é sempre um **L** . Se se desligar o computador e se voltar a ligar e depois se carregar em **ENTER** a mensagem de copyright transformar-se-á num cursor **K** . A letra que usa indica como é que o computador vai interpretar a proxima coisa que fôr escrita. No princípio da linha aparecerá sempre um **K** a piscar que representa as palavras chave (keyword). (a mensagem de copyright, bem como mensagens do compu

.../

tador também contam como se fossem um [K] . Uma palavra chave é uma das palavras especiais do computador, aparecendo no princípio de uma ordem para dar ao computador uma ideia geral sobre o que o comando lhe vai indicar para fazer. Uma vez que o computador está à espera de uma palavra chave no começo de cada linha, quando carregar - por exemplo - na tecla P, o computador decide-se a não interpretar isso como um P, mas sim como um **PRINT** ; e avisa-o de que vai fazer isso marcando o cursor com um [K] . Quando já tem a primeira palavra chave, já não está à espera de mais nenhuma, por isso o que fôr introduzido a seguir será interpretado como letras. Para mostrar isto o computador muda o cursor para um [L] - representando 'letra',

Estes estados diferentes são muitas vezes chamados de modos - daqui em diante falaremos do modo de palavra chave (ou K) e do modo de letra (ou L).

Se quiser escrever uma série de letras grandes sem estar a segurar o **CAPS SHIFT** , pode fazer com que todas as letras saiam grandes carregando primeiro na tecla **CAPS LOCK** **CAPS SHIFT** com 2). Para indicar que isto está a acontecer o cursor [L] será substituído por um [C] a piscar (representado letras grandes - capitais). Para obter novamente letras pequenas e o cursor voltar para [L] , carregar uma segunda vez em **CAPS LOCK**.

(Se carregar em **CAPS LOCK** enquanto está em modo de palavra chave, não notará de imediato nenhuma diferença, mas poderá ver o efeito depois de introduzir a palavra chave e o computador aparecer em modo C em vez de modo L).

Além de palavras chave, letras, numeros e várias expressões de programação e científicas, o teclado também tem oito caracteres gráficos. Estes aparecem com as teclas dos numeros de 1 a 8 e podem ser impressas no écran de forma semelhante às letras e aos números. Para fazer isto o teclado tem de ser modificado para o modo gráfico. Isto é feito carregando na tecla **CAPS SHIFT** e na tecla 9. Repare que o cursor mudou para [G] . Carregar de novo na tecla 9 volta a passar para o modo L .

Há ainda um último modo em que se pode colocar o teclado. O modo de extensão, indicado por um cursor [E] e que se obtem carregando simultaneamente em **CAPS SHIFT** e **SYMBOL SHIFT** . Isto permite que se utilizem maior parte das funções científicas e de programação. Carregar de novo nas duas teclas **SHIFT** fará com que o teclado volte para o modo L.

Mesmo que seja um dactilógrafo, ou programador, muito eficiente, vão aparecer sempre teclas erradas. Até agora o único modo que tinha era retirar a ficha! ainda, que possa ser conveniente se se tiver introduzido somente uma ordem para o computador, não é certamente nada cómodo se já se tiver introduzido uma grande quantidade de informação.

.../

Felizmente podemos usar a tecla **DELETE** para apagar os erros. Por exemplo, nada deve correr mal quando se introduz um comando simples como:

PRINT "Ola"

ou será que corre?

Vamos presumir que não foi utilizada a tecla de **SYMBOL SHIFT** para escrever as aspas iniciais. No ecran teriamos então

PRINT POla"

O computador não teria reconhecido o que vinha depois do **PRINT** uma vez que a falta de aspas indica ao ZX Spectrum que se está à espera de um número - e em vez disso encontrou uma letra. O computador mostra a sua confusão com um ■ a piscar no fim da linha.

Felizmente não tem de escrever tudo outra vez. Na fila de cima do teclado estão quatro setas apontando em direcções diferentes e a palavra **DELETE**. Para utilizar estas teclas, tem de se usar **CAPS SHIFT** , quando se carrega nelas. As setas voltadas para os lados movem o cursor para a esquerda ou para a direita, **DELETE** apaga o caracter imediatamente antes do cursor.

Para corrigir a sua linha errada, carregue em ◄(**CAPS SHIFT** e 5 ao mesmo tempo) até que o cursor esteja imediatamente a seguir ao P que introduziu por engano - se continuar a carregar durante um ou dois segundos então as teclas começarão a funcionar continuamente, emitando um estalido abafado. De facto, se carregar continuamente em qualquer tecla por mais do que cerca de três segundos ela repete-se automaticamente. Carregue em **DELETE** **CAPS SHIFT** e Ø) para apagar o P errado, e então escreva " **SYMBOL SHIFT** e P) para inserir o que lá devia estar - repare que foi inserido sem apagar nada. Experimente a tecla que desloca o cursor para a direita também, só para se habituar. Se cometer alguns erros de introdução a sério, corrija-os desta maneira, lembrando-se sempre que não pode escrever por cima dos erros; tem de os apagar e inserir as correcções.

Agora, quando carregar em **ENTER** o computador escreverá a sua mensagem no cimo do écran - ou por baixo da que tinha escrito da primeira vez, se ainda lá estiver.

Uma descrição completa do teclado pode ser encontrada no Capitulo 1 do manual de Programação em BASIC.

.../

3. NUMEROS, LETRAS, E O COMPUTADOR COMO MÁQUINA DE CALCULAR

Já vimos como podemos dizer ao computador para escrever letras e caracteres gráficos no écran usando **PRINT**. Também vimos que **ENTER** tem de ser usado para indicar ao computador que deve executar a ordem que acabou de ser introduzida. De agora em diante não vamos indicar o **ENTER** no manual cada vez que fôr usado um comando, mas vamos pressupor que ele será introduzido automaticamente sempre que se chegar ao fim da linha.
Os números podem ser tratados pelo computador mais facilmente do que as letras. Nos capítulos anteriores demos algumas pistas sobre isto ao explicar que o computador espera que exista um número depois de **PRINT** se não forem usadas aspas.

Assim, se introduzir

```
PRINT 2
```

o número 2 aparecerá no ecran.

É possível misturar letras com números:

```
PRINT 2, "ABC"
```

Repare que há um intervalo no ecran entre o 2 e ABC.

Agora escreva

```
PRINT 2; "ABC"
```

e depois

```
PRINT 2 "ABC"
```

Usando uma vírgula entre os items depois de **PRINT**, separa-os de 16 colunas, usando um ponto e vírgula não deixa qualquer espaço e se não se usar nada teremos um erro.
A instrução **PRINT** também pode ser usada com as funções matemáticas do teclado. De facto o ZX Spectrum pode ser usado como uma máquina de calcular electrónica.

.../

Por exemplo:

```
PRINT 2+2
```

A resposta aparece no cimo do écran. Compare como:

```
PRINT "2+2"
```

É possível combinar estas duas para dar algo mais útil. Tente

```
PRINT "2+2=";2+2
```

Experimente também alguns outros tipos de aritmética:

```
PRINT 3-2
PRINT 4/5
PRINT 12*2
```

O símbolo * é usado como sinal de multiplicação em vez de x para evitar que se confunda com a letra x; e / é usado como sinal de divisão em vez de : .
Experimente com muitos cálculos diferentes. Se quiser pode usar números negativos ou números com vírgula.
Se fizer cálculos suficientes para utilizar as 22 linhas da parte de cima do écran verá que acontece algo muito interessante; mover-se-á tudo para cima uma linha e a linha de cima perder-se-á. Chama-se a isto scrolling.
=========

Os cálculos nem sempre são executados pela ordem que se espera. Por exemplo experimente

```
PRINT 2+3*5
```

Podia-se esperar que esta instrução somasse dois com três, dando cinco e depois multiplicasse o resultado por cinco, obtendo-se 25; no entanto não é isto que se passa. As multiplicações, - bem como as divisões - são executadas antes das adições e das subtrações, de modo que a expressão '2 + 3 * 5' significa 'multiplique-se 3 por 5, dando 15; e depois somar isto com 2, dando 17'. 17 deve ser a resposta que aparece no écran.

.../

Uma vez que as multiplicações e divisões são feitas em primeiro lugar dizemos que elas têm uma prioridade mais elevada do que a adição e a subtracção. Relativamente uma à outra a multiplicaçao e a divisão têm a mesma prioridade , o que significa que as multiplicações e divisões são feitas ordenadamente da esquerda para a direita.Quando se tiverem acabado estas, trataremos então das adições e substracções - as quais também têm a mesma prioridade uma que a outra, por isso as fazemos ordenadamente da esquerda para a direita.

Vamos ver como o computador calcularia:

PRINT 20-2*9+4/2*3

i. 2Ø - 2 * 9 + 4/2 * 3
ii. 2Ø - 18 + 4/2 * 3
iii. 2Ø - 18 + 2 * 3

Primeiro executam-se as multiplicações e divisões ordenadamente da esquerda para a direita

iv. 2Ø - 18 + 6
v. 2 + 6
vi. 8

e depois as adições e sobtracções

Ainda que a única coisa que seja preciso saber é se uma operação tem uma prioridade mais alta ou mais baixa, o computador faz isto tendo um número entre 1 e 16 para representar a prioridade de cada operação: * e / têm prioridade 8 e + e - têm prioridade 6.
Esta ordem de cálculo é absolutamente rígida, mas pode-se passar por cima dela utilizando parentesis; o que estiver entre parentesis é calculado primeiro e depois tratado como um único numero, de forma que

PRINT 3*2+2

dá como resultado 6 + 2 = 8 mas

PRINT 3*(2+2)

dá como resposta 3 * 4 = 12

É por vezes útil dar ao computador expressões como esta porque sempre que o computador está à espera que lhe demos um número, podemos dar-lhe antes uma ex

.../

pressão que ele calculará o resultado. As excepções a esta regra são tão poucas que serão formuladas explicitamente em cada caso.
Podemos escrever números com vírgula (usando o ponto decimal), e também se pode usar notação científica - como é usual em calculadoras de bolso. Neste caso, depois de um número vulgar (com ou sem vírgula) escrevemos a parte do expoente consistindo na letra e e depois talvez um -, e então ainda um número. A parte do expoente muda a vírgula para a direita (ou para a esquerda se o expoente fôr negativo), multiplicando-se (ou dividindo-se) assim o número inicial por 1Ø várias vezes. Por exemplo:

```
2.34 e Ø = 2.34
2.34 e 3 = 234 Ø
2.34 e - 2 = Ø.Ø 234    e assim por diante
```

(experimente fazer sair isto no computador). Este é um dos poucos casos em que não se pode substituir um número por uma expressão: por exemplo, não se pode escrever

```
(1.34 + 1) e (6/2).
```

Também se podem ter expressões cujos resultados não são números, mas cadeias (strings) de letras. Já vimos várias vezes a forma mais simples disto, a cadeia de letras que é escrita entre aspas. Esta forma é inteiramente análoga à forma mais simples de expressão numérica, que é um número escrito por si só. O que ainda não vimos é que também se pode usar o sinal + com cadeias (mas nunca -, * ou / por isso aqui não vamos ter problemas de prioridade). Adicionar cadeias é somente juntá-las uma a seguir à outra: experimente

```
PRINT "hei" + "vaca"
```

Podem-se adicionar quantas cadeias quisermos numa só expressão, e se quisermos até podemos usar parentesis.

.../

4. ALGUMAS INSTRUÇÕES SIMPLES

A memória do computador pode ser usada para guardar todo o genero de coisas. Já vimos, até aqui, que a instrução PRINT permite escrever letras, números e os resultados de calculos usando tanto números como letras no écran.
Se quisermos dizer ao computador para se lembrar de um número, ou de uma cadeia de letras, então temos de guardar algumas memórias para esse fim.
A maior parte das máquinas de calcular de bolso têm uma tecla chamada 'memória' que é usada para mais tarde se relembrar numeros. O seu computador pode fazer muito melhor do que isso: pode tantas caixinhas destas quantas quisermos, e podemos escrever um nome em cada uma delas.
Como exemplo, suponhamos que nos queremos recordar da nossa idade! Usaremos a instrução LET (LET é a palavra chave na tecla L) : digamos que são 34 anos

```
LET idade = 34
```

O que acontece quando a instrução LET é utilizada, é que uma certa zona de memória é designada por 'idade' e o número 34 é lá guardado. Para fazer sair esta informação escrevemos

```
PRINT idade
```

e teremos o número 34 de volta. É muito fácil modificar o conteúdo da 'caixa' chamada 'idade'. Introduza:

```
LET idade = 56
```

e depois

```
PRINT idade
```

e deve aparecer 56 no ecran. 'idade' é um exemplo de um variável, assim chamada porque o seu valor pode variar. É possível combinar a impressão de uma mensagem directamente para o écran, juntamente com uma variável. Introduza

```
PRINT "A sua idade é"; idade
```

.../

No entanto o computador é muito mais útil. Não se lembra somente de números com nomes a eles associados. Também se pode recordar de cadeias de letras. Para diferenciar entre as variáveis numéricas e as variáveis de cadeia - como lhes chamaremos - utiliza-se o simbolo dolar - $ - no fim do nome da variável.
Por exemplo: se quisermos guardar a cadeia de letras

"A sua idade e"

chamar-lhe-iamos

a $

(variáveis de cadeia não podem ter nomes com mais do que uma letra, seguida de $).
Introduza então

LET a$ = "A sua idade e"

se agora escrever

PRINT a $

a cadeia de letras voltará a aparecer no ecran.
Se o computador não tiver sido lesligado desde o início deste capítulo escreva

PRINT a $; idade

e veja o que acontece.
Há outros modos de colocar a informação na memória do computador sem usar a instrução **LET**.
Por exemplo a instrução **INPUT**, na sua forma mais simples, diz ao computador que ele deve esperar informações vindas do teclado. Em vez de escrever **LET** etc, cada vez que precisar, escreva

INPUT idade

.../

Assim que tiver carregado na tecla **ENTER** aparecerá no écran um cursor ■ a piscar. Isto significa que o computador espera de si alguma informação. Escreva então a sua idade e carregue de novo em ENTER. Ainda que nada pareça acontecer a variável recebeu o valor que você acabou de lhe dar. Se escrever

```
PRINT idade
```

poderá verificar isso.

Vamos combinar todo isto numa serie de instruções.
Escreva

```
LET b$ = "Qual e a sua idade?"
LET a$ = "A sua idade e"
INPUT (b$); idade: PRINT a$; idade
```

Repare que a última linha consiste em duas instruções separadas por dois pontos.

```
INPUT (b$); idade
```

é outra forma de escrever

```
INPUT "Qual é a sua idade?; idade
```

.../

5. PROGRAMAÇÃO SIMPLES

Até aqui temos dito directamente a partir do teclado ao computador o que ele devia fazer. Ainda que seja possível combinar numa só várias instruções, somente aplicações muito limitadas seriam possíveis usando esse método.
O melhor dos computadores é que eles são programáveis. Isto significa que podemos dar-lhes uma serie de instruções para os levar a fazer coisas em sequência.
Cada computador tem a sua própria linguagem que nos permite comunicar com ele.
Algumas linguagens são muito simples - de modo que o computador as consegue compreender facilmente. Infelizmente as linguagens que são simples para o computador compreender são difíceis para os seres humanos. De certo modo o contrário também é verdade - linguagens suficientemente simples para a nossa compreensão são relativamente complicadas para o computador - e até têm de ser traduzidas ou interpretadas.
O ZX Spectrum utiliza uma linguagem de alto nível chamada BASIC.
BASIC significa Beginners All-purpose Symbolic Instruction Code (Código de Instruções Simbólicas de Aplicabilidade Geral para Principiantes) e esta linguagem foi desenvolvida na Universidade de Dartmouth em New Hampshire, Estados Unidos, em 1964. É largamente utilizada em computadores pessoais, e ainda que seja essencialmente semelhante em todos eles, há diferenças subtis. É por isso que este manual é escrito especificamente para o ZX Spectrum. Mas o BASIC do ZX Spectrum não é muito diferente de um (inexistente) BASIC geral e por isso não deve ter muito trabalho para adaptar qualquer programa em BASIC para o ZX Spectrum. Ao contrário de outros BASICs, o BASIC do ZX Spectrum não permite que a instrução **LET** seja omitida quando se atribuem valores a variáveis.
Há um limite para o número de instruções que pode ser armazenado no computador. O ZX Spectrum indica este limite emitindo um zumbido.
Quando se programa em BASIC ,e necessário dar a conhecer a computador em que ordem é que as instruções devem ser executadas. Assim, cada linha da sequência de instruções tem um número no princípio. É normal começar em 1Ø e depois acrescentar sempre 1Ø a cada linha. Isto permite que se insiram outras linhas se tiverem sido omitidas, ou se se quiser modificar o programa.
Vamos ver um programa simples. Considere-se a serie de instruções no fim do último capítulo. Se quisessemos repetir aquela série de comandos seria necessário introduzi-las de cada uma das vezes. Um programa supera essa necessidade.
Escreva o seguinte com **ENTER** no fim de cada linha.

.../

```
1Ø LET b$ = "Qual e a sua idade?"
2Ø LET a$ = "A sua idade e"
3Ø INPUT (b$); idade
4Ø PRINT a$; idade
```

Repare que não é necessário introduzir os espaços, excepto entre aspas.
Nada acontecerá até que digamos ao computador para começar a trabalhar no programa. Isto é feito utilizando **RUN** (a palavra chave em R)

Introduza esta instrução e veja o que acontece

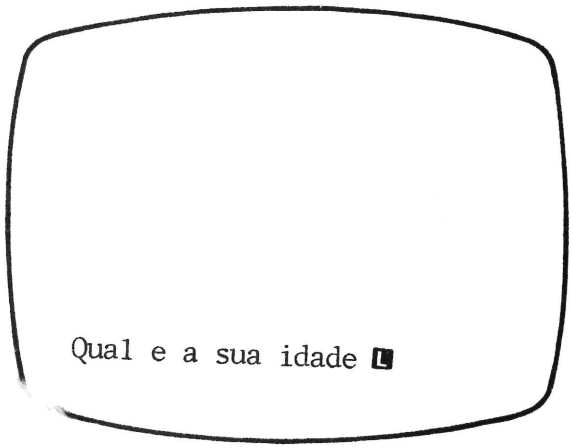


Figura 4

Também deve ter reparado uma seta voltada para a direita cada vez que se introduzia uma linha. Esta indica a ultima linha que foi introduzida. Se quiser ver o programa outra vez carregue em **ENTER** de novo, (ou **LIST**). Pode usar **RUN** para executar o programa quantas vezes quiser. Quando já não precisar deste programa pode apagá-lo usando a instrução **NEW.**
Introduza **NEW** e depois **LIST** e veja o que acontece.

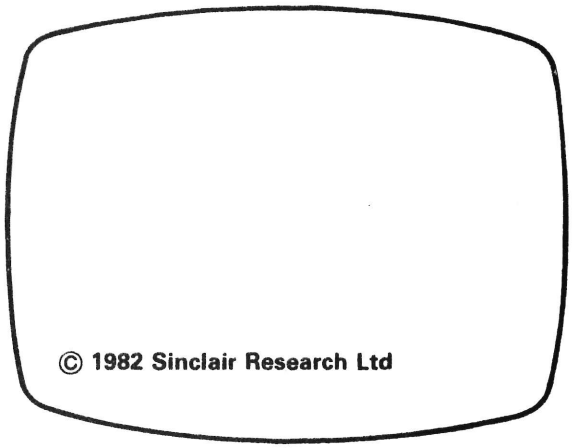


Figura 5

.../

Recapitulando:
Quando se introduz uma instrução precedida de um número, isto diz ao computador que não é somente uma ordem, mas uma linha de programa. O computador não a executa, mas guarda-a para mais tarde.
Para auxiliar o ZX Spectrum escreve no écran (ou lista) todas as linhas de programa que se introduziram com um ▶ contra a última linha introduzida.
O computador não executará nenhuma destas linhas imediatamente, mas guardá-las-á dentro dele.
Para que o computador execute estas linhas tem de usar a instrução **RUN**.
Se carregar somente em **ENTER** voltará a ter a listagem.
Vamos considerar outro programa simples. Este será um pouco mais virado para a matemática e imprimirá os quadrados de todos os números entre 1 e 1Ø (o quadrado de um número é somente esse numero multiplicado por ele próprio).
Para gerar números de 1 a 1Ø introduzimos outro conceito na programação em BASIC. Este é o método que utilizamos para que o computador conte. Anteriormente já vimos que os números podem ser guardados na memória do computador pegando-os a um nome - ou tecnicamente atribuindo um valor a uma variável. Vamos então fazer com que a variável x comece com 1 e vá subindo de 1 em 1 até 1Ø. Isto é feito utilizando a instrução **FOR ... TO ... STEP .**
Assim, para introduzir este programa carregue em **NEW** para se livrar do anterior e escreva o seguinte:

```
10 FOR x=1 TO 10 STEP 1
```

(Normalmente a parte **STEP 1** pode ser omitida se a contagem fôr crescente e de 1 em 1).
A linha seguinte deve dizer ao computador o que deve fazer com x qualquer que seja o valor que ele tenha, por isso escreva:

```
20 PRINT x, x*x
```

Finalmente precisamos de uma linha que diga ao computador para ir para o valor seguinte de x, assim introduza

```
30 NEXT x
```

.../

Quando chegar a esta instrução o computador volta a linha 1Ø e repete a sequência. Quando x fôr maior que 1Ø o computador volta para a linha seguinte do programa, isto é, a linha 4Ø.
O programa deve agora ter o seguinte aspecto no écran:

```
10 FOR x=1 TO 10 STEP 1
20 PRINT x, x*x
30 NEXT x
```

Para que ele esteja completo temos na verdade de ter outra linha a dizer ao computador que o programa acabou quando x = 1Ø por isso escreva

```
40 STOP
```

Se carregarmos agora em **RUN** devem aparecer duas colunas, a primeira com valores de x, a segunda com valores de x * x ou seja de x ao quadrado. É possível identificar estas colunas acrescentando outra linha, assim

```
5 PRINT "x","x*x"
```

Repare que ainda que esta tenha sido introduzida depois de todas as outras linhas, dado que o seu número é mais baixo, o computador coloca-a automaticamente no sítio certo.
Tente escrever programas usando outras funções matemáticas. Se tiver algumas dúvidas sobre o seu uso procure nas páginas aproriadas no manual de programação BASIC.

6. UTILIZAÇÃO DE UM GRAVADOR DE CASSETES

É bastante fastidioso ter de introduzir os programas pelo teclado do computador cada vez que os quisermos usar. O ZX Spectrum tem um sistema para gravar programas em fitas magnéticas com um gravador de cassetes normal e doméstico. Se tiver um programa em memória tente guardá-lo de acordo com o procedimento seguinte.

Se guardar programas em cassete pode carregá-los novamente quando quiser.

A maior parte dos gravadores de cassete servem: no que diz respeito ao computador os gravadores de cassetes mono-fónicos portáteis e baratos são pelo menos tão bons como os estereofónicos mais dispendiosos, e também dão menos maçada. Verá que um contador de voltas é muito útil.

O gravador de cassetes tem de ter uma entrada para usar com microfones e uma saída para usar com auscultadores (se não houver uma experimente a saída externa para o altifalante). Devem ter fichas de 3,5 mm (isto é, adequados para as fichas de ligação que acompanham o computador), porque outra espécie de saída pode não dar um sinal suficientemente forte para o computador.

Qualquer cassete serve, ainda que, fitas de baixo ruído sejam mais adequadas.

Depois de ter adquirido um gravador de cassetes adequado, ligue-o ao computador usando os cabos fornecidos com o ZX Spectrum: um cabo deve ligar a entrada marcada 'MIC' na parte de trás do computador com a entrada de microfone do gravador e o outro deve ligar a saída para auscultadores do gravador com a entrada marcada com 'EAR'. (não acontecerá nenhum mal ao ZX Spectrum se ligar os cabos ao contrário).

Quando usar a instrução **SAVE** para guardar o programa em cassete, tem de se certificar que uma das fichas do cabo que liga as entradas marcadas com 'EAR' está desligada - qualquer delas serve. Se se esquecer de fazer isto não obterá nada no gravador senão uma nota constante, o que não lhe serve de nada. O motivo para isto, é que, quando o gravador está a gravar amplifica o sinal que entra pela ficha do microfone e coloca-a na entrada 'EAR'. Se este volta a entrar no computador formará um ciclo, que oscilará, homogeneizando o sinal que estamos a tentar gravar.

Introduza um programa no computador, por exemplo o programa dos quadrados do capítulo anterior, e depois escreva:

SAVE "Quadrados"

.../

Quadrados é somente o nome que se usa para chamar o programa guardado na fita. Tem direito a usar até dez caracteres no nome que tem de consistir somente de letras e numeros.
O computador ter-lhe-á enviado a mensagem Start tape then press any key (Li - gue o gravador para gravar e depois carregue numa tecla qualquer). Vamos primeiro fazer um ensaio em seco para poder ver o que acontece: não ligue o gravador de cassetes, mas carregue numa tecla do ZX Spectrum e olhe para a parte exterior do ecran de TV. Verá padrões de riscas horizontais coloridas.
5 segundos de tiras vermelhas e azul claro, com cerca de 1 cm de largura e movendo-se lentamente para cima.
Um intervalo muito pequeno de tiras azuis e amarelas.
1 segundo com tudo normal.
2 segundos de padrão vermelho e azul claro de novo.
cerca de 1 segundo do padrão azul e amarelo outra vez.
Experimente outra vez até ser capaz de reconhecer tudo isto.
A informação é guardada em dois blocos e ambos os blocos levam uma etiqueta correspondendo ao padrão vermelho e azul, e a informação em si correspondendo ao padrão azul e amarelo. O primeiro bloco é preliminar e contém o nome e vários outros bits de informação acerca do programa, e o segundo é o programa em si juntamente com quaisquer variáveis presentes. A secção branca entre eles é somente um intervalo vazio.
Agora vamos capturar efectivamente o sinal na fita da cassete.

1. Colocar a fita numa zona ou que esteja em branco ou que não se importe de desgravar.
2. Escreva.

SAVE "Quadrados" (e **ENTER**)

3. Ligue o gravador para gravar.
4. Carregue em qualquer tecla do ZX Spectrum.
5. Observe o écran da televisão como antes. Quando o computador tiver acabado (com a mensagem Ø OK) desligue o gravador.

Para se certificar que isto funcionou, pode verificar o sinal na fita comparativamente com o programa no computador usando a instrução **VERIFY**.

.../

1. Ponha o volume do gravador de cassete em mais ou menos metade e volte a ligar a ficha na entrada 'EAR'.
2. Volte com a fita atrás até um ponto anterior aquele em que começou a gravar.
3. Escreva.

VERIFY "Quadrados"

(VERIFY encontra-se em modo de extensão, e depois R com SYMBOL SHIFT).

4. Ligue o gravador para reprodução.

A parte exterior do ecran alternará entre vermelho e azul pálido até que a fita atinja a gravação que se fez; depois verá o mesmo padrão que quando guardou o seu programa. No segundo de intervalo aparecerá escrito no écran Program Quadrados - quando o computador está à procura de alguma coisa na fita imprime o nome de tudo o que encontra. Se vir todo o padrão outra vez e depois o computador parar com a mensagem Ø OK o seu programa está guardado em segurança na fita e pode passar por cima dos parágrafos seguintes. Senão, algo está errado. Percorra o texto seguinte para ver o que é.

CERTIFIQUE-SE QUE O SEU PROGRAMA ESTÁ GUARDADO

O nome apareceu ?

Se não, ou o programa não foi guardado adequadamente logo ao princípio, ou foi, mas não foi lido adequadamente. Tem de descobrir qual das causas foi. Para ver se foi guardado bem, volte atrás com a fita para onde começou a gravar e reproduza-a no altifalante do gravador (provavelmente ter. de desligar a ficha de saída dos auscultadores do gravador). A etiqueta vermelha e azul pálido dá uma nota muito clara e alta, e a informação azul e amarela dá um som algo menos agradável, como uma mensagem em código morse transmitida em alta velocidade. Ambos os sons são bastante altos e com o volume no máximo facilmente impedem a conversação.

Se não ouvir estes ruidos então o programa provavelmente não ficou guardado. Verifique se todas as ligações estão feitas nos sítios adequados. Certifique-se que ambas as ligações marcadas com 'MIC' estão ligadas e de que as marcadas com 'EAR' não estão. Acontece que em alguns gravadores a ficha não faz contacto quando está completamente introduzida. Tente puxar um pouquinho para fora -

.../

- por vezes sente-se que ela fica numa posição mais natural. Verifique também se não está a tentar gravar na fita de plástico no princípio da cassete. Quando tiver verificado tudo isso, tente de novo.
Se conseguir ouvir os sons descritos então o **SAVE** correu provavelmente bem e o problema estava na leitura.
Verifique as ligações de novo, e verifique também o nível do som. Se fôr muito baixo o computador não poderá ouvir o sinal adequadamente, e não se conseguirá ver os padrões certos no écran; se estiver muito alto o sinal ficará distorcido - deve ser capaz de ouvir o som a sair pelo próprio altifalante do computador. Há uma grande gama de valores aceitáveis entre esses extremos, mas pode experimentar.
O caso seguinte é quando o computador encontra o nome e o escreve mas ainda assim algo corre mal. Algumas possibilidades são:
Enganou-se no nome, ou uo **SAVE** (quando o computador escreve o nome errado no écran) ou quando deu a instrução **VERIFY** o computador ignorará o programa e continuará com a cintilação azul e vermelha.
Pode haver um erro genuino na fita: o computador dará uma mensagem com **R Tape loading error** (Erro na leitura da fita), o que significa neste caso que ele não conseguiu verificar o programa. Guarde-o outra vez.
É possível que o volume no gravador não esteja muito certo; mas não deve estar muito longe porque o computador conseguiu ler o primeiro bloco.

Agora vamos supor que tinha conseguido com sucesso guardar o programa e que a verificação tinha corrido bem. Voltar a carregá-lo é exactamente como fazer a verificação só que a instrução é

LOAD "Quadrados"

em vez de

VERIFY "Quadrados"

LOAD está na tecla J. Uma vez que a verificação tinha corrido bem não deve haver problema na leitura.
LOAD apaga o programa anterior (e variáveis) no computador antes de carregar o novo a partir da fita.

.../

Uma vez o programa carregado, a instrução **RUN** permitirá executá-lo.
É possível comprar programas pré-gravados em cassete. Devem ser escritos especialmente para o ZX Spectrum:tipos diferentes de computadores têm modos diferentes de gravar programas, por isso não podem usar as fitas uns dos outros. Se a sua cassete tem mais do que um programa gravado no mesmo lado, então cada um deles terá um nome. Pode escolher qual o programa a carregar com a instrução **LOAD**: por exemplo, se quiser um chamado 'helicopter' pode escrever.

LOAD "helicopter"

(a instrução (**LOAD ""** significa carregar o primeiro programa que o computador encontrar, o que pode ser muito útil se não se conseguir lembrar do nome do programa).

.../

7. CORES

Um dos motivos que leva as pessoas a comprar um ZX Spectrum é a sua capacidade para usar um écran de TV a cores. O écran está dividido em duas zonas. A zona exterior é referida como **BORDER**, e o Zona central como **PAPER**. É possível mudar as cores destas duas zonas à vontade, tanto directamente do teclado como por programa.
O ZX Spectrum tem oito cores para usar, e a estas são dados números entre Ø e 7. Ainda que as cores pareçam estar desordenadas, de facto elas dão tonalidades de cinzento decrescentes numa televisão a preto e branco.
Aqui está uma lista delas para referência; também estão escritas por cima das teclas respectivas:

Ø negro
1 azul
2 vermelho
3 purpura ou magenta
4 verde
5 azul pálido ou cian
6 amarelo
7 branco

Quando o computador é ligado o sistema trabalha a preto e branco. Assim o valor normal para **BORDER** e **PAPER** é 7, isto é, branco. A côr de qualquer caracter que aparece no écran é definido pela instrução **INK**. Esta é normalmente Ø, isto é, negro. Inicialmente os três comandos que controlam as cores do écran são colocados pelo computador.
No entanto pode-se alterar estes valores. Por exemplo escreva

BORDER 2

Se se lembrou de carregar depois de **ENTER**, a zona exterior deve ter mudado de branco para vermelho. Isto inclui o zona de baixo onde se escrevem numeros e veja como as cores mudam.
Agora tente mudar o centro do écran introduzindo

PAPER 5

.../

O comando **PAPER** encontra-se no modo de extensão, como foi dito antes. Obtem-se carregando simultaneamente em **CAPS SHIFT** e em **SYMBOLS SHIFT**. **PAPER** é então a tecla C com SYMBOL SHIFT. Quando a tecla **ENTER** é pressionada duas vezes o centro do écran deve mudar para azul. O primeiro **ENTER** cancela a instrução **PAPER** que já existia no computador, mas só quando o segundo **ENTER** é premido (fazendo com que o computador **LIST** e qualquer programa e assim reconstrua a informação do écran) é que a nova côr para a zona **PAPER** é usada. Se estiver a usar uma televisão a cores e ela não tiver mudado de côr, tente ajustar os controles de côr da televisão, e talvez alterar a sintonização.

A instrução **INK** é semelhante à instrução **PAPER** e controla a côr dos caracteres que aparecem sobre a parte do écran que se denomina **PAPER**. Obviamente se a côr que figura na instrução **INK** e a côr que figura na instrução **PAPER** forem a mesma, nada aparecerá no écran!

As instruções **BORDER**, **PAPER**, e **INK** podem ser usadas em programas. Aqui está um programa simples que mostra a gama de cores disponível:

```
10 FOR x=0 TO 7
20 BORDER x
30 PAPER 7-x: CLS
40 PAUSE 50
50 NEXT x
```

Este programa, quando executado (**RUN**), passa pelas oito cores, contrastando as cores da zona **PAPER** com as da zona do **BORDER**. A instrução **CLS** depois da instrução PAPER obriga o computador a reconstituir a imagem do écran e a usar a nova côr para a zona **PAPER**. A instrução **PAUSE** diz ao programa para parar durante 1 segundo para que nós possamos ver o que se está a passar (Tente correr o programa sem a instrução de **PAUSE**). Para mostrar como funciona a instrução **INK** introduza o seguinte programa, depois de ter executado uma instrução **NEW**.

```
10 BORDER 7
20 PAPER 1
30 INK 4
40 PRINT "Caracteres Verdes em fundo azul"
```

.../

Existem outras instruções associadas com as capacidades de controle de côr do ZX Spectrum, e estas são descritas em detalhe no manual de programação em BASIC.

8. SOM

O ZX Spectrum pode fazer sons de uma variadede infinita. A frequência da nota e a sua duração estão sob controle do utilizador. A instrução **BEEP** é usada para dizer ao computador para fazer um som. O **BEEP** é uma instrução do modo de extensão e obtido usando a tecla Z.
A frequência 'central' para o comando **BEEP** é um Dó médio. Este pode ser variado com a instrução **BEEP** e qualquer nota pode ser obtida se fôr expressa como semitons ou partes de semitons acima e abaixo desta frequência central. Se a instrução

BEEP 2,0

fôr introduzida no computador será emitido o som de um dó médio durante dois segundos.
Os dois números comandam o tipo de nota que é emitida, o primeiro dando a extenção á nota em segundos, e o segundo a frequência da nota em semitons acima do dó médio. Assim o código de frequência para o dó médio é Ø, o código para um dó sustenido é 1, para um ré é 2, e assim por diante até ao dó seguinte que é 12, porque 12 semitons fazem uma oitava. Pode-se passar para 13 e ainda além, se quisermos, de modo que quanto maior o número, mais alto será o som.

Experimente isto:

BEEP 1,4: BEEP 1,2: BEEP 2,0

Ouvirá precisamente o primeiro compasso de uma melodia inglesa intitulada 'Three Blind Mice'. Uma vez que pode juntar vários **BEEP** com dois pontos como acima, pode, se tiver paciência, produzir uma melodia completa. Pode gostar de experimentar usar mais notas do que três.
(Os dois pontos não se limitam a juntar instruções **BEEP**; pode usá-los para construir instruções compostas a partir de quaisquer instruções elementares).

.../

Um exemplo mais complicado: pode-se fazer uma instrução de camaleão cantor misturando instruções **BEEP** e **BORDER** :

```
BORDER 1: BEEP 1,14: BORDER 3: BEEP 1,16: BORDER 4: BEEP
1,12: BORDER 6: BEEP 1,0: BORDER 5: BEEP 4,7: BORDER 1
```

(Não se preocupe com o facto de que isto se estende por mais que uma linha de texto: o computador não repara nisso).
Um pequeno programa para tocar uma série completa de notas seria da seguinte forma:

```
10 FOR x=0 TO 24
20 BEEP 2, x
30 NEXT x
```

Há muito mais coisas que se podem fazer com esta instrução - veja o manual de programação BASIC para mais ideias.
Para notas abaixo do dó médio, o número de semitons é indicado por um número negativo.

.../

9. O QUE ESTÁ DENTRO DA CAIXA ?

A figura mostra como é o interior de um ZX Spectrum.

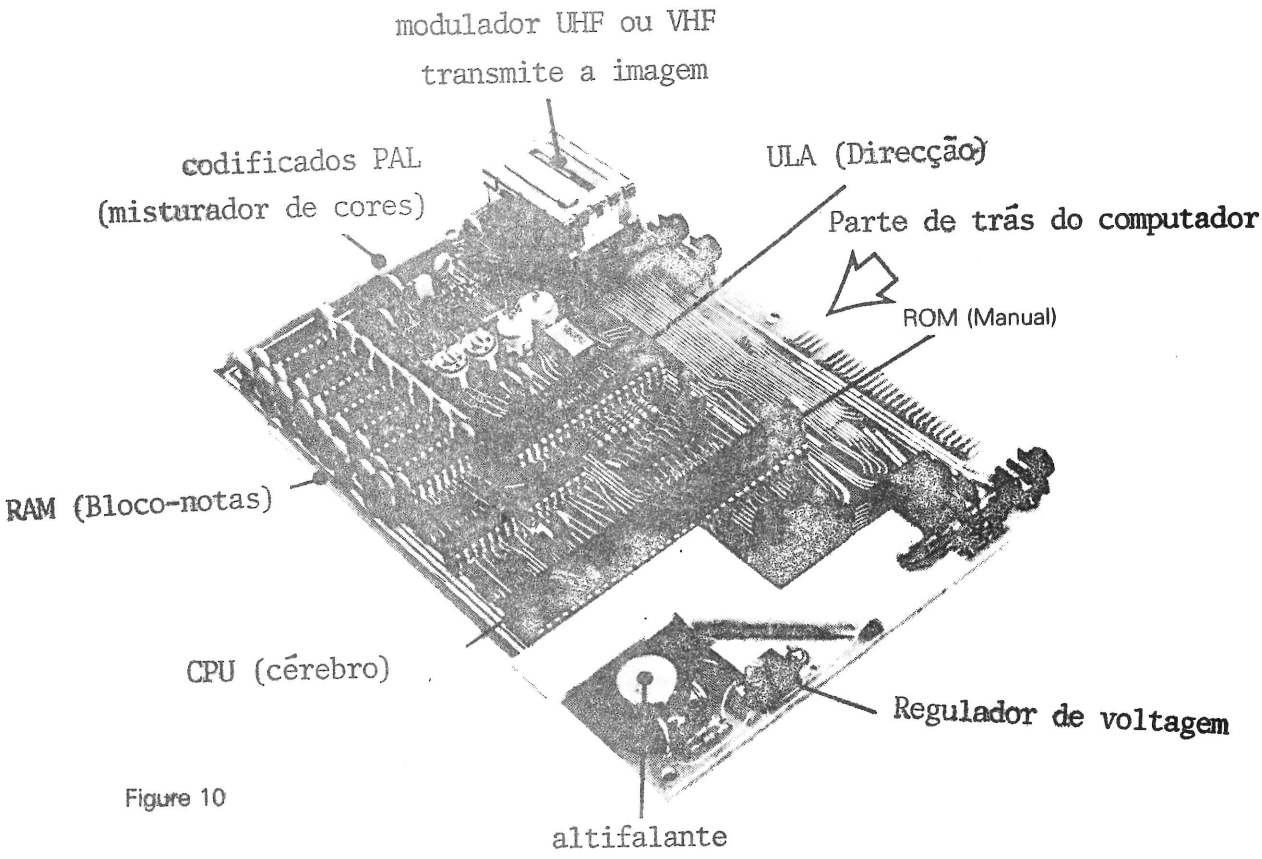


Figure 10

Figura 10

Como pode ver todos os nomes estão dados por abreviaturas de três letras. As peças rectangulares negras de plástico com muitas pernas de metal são circuitos integrados que na verdade fazem o trabalho todo. Dentro de cada um há um quadrado de silício de 1/4" x 1/4" ligados por fios ás pernas de metal. Nessa placa de silício há centenas de transistores que compõem os circuitos electrónicos que são o computador.

O cérebro responsável pela operação é a placa do processador, muitas vezes chamado de CPU (Central Processor Unit - Unidade Central de Processamento). Este particular chama-se z8ØA, que é uma versão mais rapida do já popular Z8Ø.

O processador controla o computador, faz a aritmética, verifica que teclas foram premidas, decide o que deve fazer em consequência disso, e em geral

.../

decide tudo o que o computador deve fazer. No entanto, apesar de toda a sua inteligência, não podia fazer isto tudo sozinho. Não percebe nada de BASIC ou de aritmética com vírgula decimal, por exemplo, e tem de obter todas as suas instruções de outra placa, a ROM (Read Only Memory - Memoria só para Leitura). A ROM contém uma longa lista de instruções que constituem um programa de computador, dizendo ao processador o que deve fazer em todas as circunstâncias previsíveis. Este programa está escrito numa linguagem diferente do BASIC, chamada a linguagem máquina do Z8Ø, e toma a forma de uma longa sequência de números. Há ao todo 16384 (16 * 1Ø24) destas instruções, motivo porque o BASIC do ZX Spectrum é muitas vezes chamado um BASIC de 16K - dado que 1Ø24 constituem 1K.

Ainda que haja placas semelhantes em outros computadores, esta sequência particular de instruções é única para o ZX Spectrum e foi escrita especialmente para ele.

As oito placas próximas dele são para a memória. Esta é a RAM (Random Access Memory - Memoria de Acesso Aleatório) e há duas outras placas que trabalham em conjunto com elas. A RAM é onde o processador guarda informação que quer guardar, qualquer programa em BASIC, as variáveis, a imagem do écran de televisão e vários outros items que mantêm a informação do estado do computador.

A placa grande é a SCL (Sinclair Computer Logic - Lógica de Computadór Sinclair). Actua na verdade como um centro de comunicações, certificando-se que tudo o que o processador quer se efectua realmente. Também lê a memória para ver em que consiste a imagem de televisão e envia o sinal apropriado para a interface da TV.

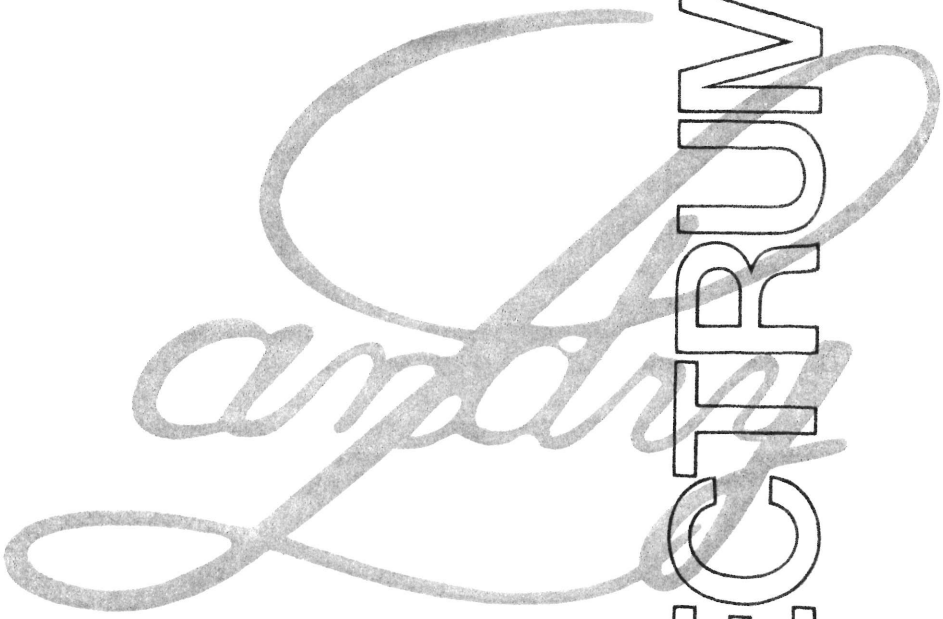
O codificador PAL é um conjunto completo de componentes que converte a saída de televisão da placa de lógica para uma forma mais adequada para televisões a cores.

O regulador converte a voltagem algo errática da fonte de alimentação para cinco volts absolutamente constantes.

Aqui se conclui este Folheto Introdutório. Se acha que o compreendeu bem, sugerimos que tente agora ler o manual de programação BASIC.

.../

sinclair



SPECTRUM

ZX

BASIC programming

by Steven Vickers
edited by Robin Bradbeer

First Edition 1982

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

1

CAPÍTULO 1

Introdução

Quer tenha lido o livro introdutório primeiro ou tenha vindo directamente para este, deve saber que comandos são obedecidos imediatamente e que as instruções começam com um numero de linha e são guardadas para mais tarde. Também deve saber os comandos: **PRINT**, **LET**, e **INPUT** (que podem ser usadas em todas as máquinas que usam BASIC). **BORDER**, **PAPER**, e **BEEP** (que são usados no Spectrum). Este manual de BASIC começa por repetir algumas das coisas que foram apresentadas no livreto introdutório, mas com muito mais detalhe, dizendo-lhe exactamente o que pode e não pode fazer. Também irá encontrar alguns exercícios no fim de cada capítulo. Não os ignore: muitos deles ilustram pontos que são somente entrevistos no texto. Passa uma vista de olhos por eles e faça os que lhe interessarem mais, ou que pareçam cobrir algum campo que não compreenda completamente.

O que quer que seja que tem para fazer, continue a usar o computador. Se tiver alguma pergunta do género "o que é que ele faz se eu lhe der uma instrução assim? então a resposta é fácil, escreva-a e veja. Sempre que o manual lhe diga para introduzir qualquer coisa, pergunte sempre a si próprio "o que é que eu poderia escrever em vez disto?", e experimente as respostas. Quanto mais programas seus escrever, melhor compreenderá o computador.

No fim deste manual de programação estão alguns apêndices. Estes incluem secções sobre o modo como a memória está organizada, como o computador manipula números, e uma série de programas exemplo que ilustram o poder do ZX Spectrum.

O Teclado

Os caracteres do ZX Spectrum compreendem não somente símbolos (letras, dígitos, etc), mas também termos compostos (palavras chave, nomes de funções,etc) e todos estes termos entram directamente pelas teclas em vez de serem escritos letra a letra. Para obter todas estas funções e instruções, algumas teclas têm cinco ou mais significados diferentes, dados parcialmente pelas teclas de alteração (**SHIFT**), (isto é, ao premir ou **CAPS SHIFT** ou **SYMBOL SHIFT** ao mesmo tempo que a tecla requerida) e parcialmente por ter a máquina em modos diferentes.

.../

O modo é indicado pelo cursor, uma letra a piscar que indica o local onde será colocado o próximo caracter introduzido.
O modo K (para palavras-chave-keywords) substitui automaticamente o modo L quando a máquina está à espera de uma ordem ou de uma linha de programa (em vez de dados para a instrução **INPUT**), e da sua posição na linha sabe se deve esperar um número de linha ou uma palavra-chave. Isto acontece sempre no princípio de uma linha, ou exactamente depois de um **THEN**, ou depois de : (excepto quando este se encontre numa cadeia de letras). Se não se tocar em nenhuma tecla de alteração (**SHIFT**), a próxima tecla será interpretada ou como uma palavra-chave (impressa sobre as teclas) ou um digito.
O modo L (para letras) ocorre normalmente em todas as outras ocasiões. Se não se premir nenhuma tecla de alteração a próxima tecla será interpretada como o símbolo principal impresso na tecla, em letras pequenas para o caso de uma letra.
Tanto no modo K como no L carregar em **SYMBOL SHIFT** e noutra tecla será interpretado como o caracter a vermelho inscrito na tecla e carregar em **CAPS SHIFT** com uma tecla numérica será interpretado como uma função de controle escrita a branco acima da tecla. Carregar em **CAPS SHIFT** não afecta as outras teclas no modo K, e no modo L converte uma letra pequena em letra grande.
O modo C (para letras grandes) é uma variante do modo L em que todas as letras aparecem como letras grandes. A tecla **CAPS LOCK** transforma o modo L em modo C e vice-versa.
O modo E (de extensão) é obtido para usar ainda mais caracteres, essencialmente palavras. Ocorre depois de se ter carregado simultaneamente em ambas as teclas de alteração, e só se mantem até se ter premido uma tecla. Neste modo as teclas de letras dão um caracter ou palavra (impresso a verde por cima dela) se não se tocar mais nenhuma tecla de alteração, e outro (impresso a vermelho por baixo) se se permir uma tecla de alteração simultaneamente. Uma tecla numérica dá um simbolo se fôr permida com **SYMBOL SHIFT**; de outra forma dá uma sequência de controle de côr.
O modo G (para gráficos) ocorre depois de se ter carregado em **GRAPHICS** (**CAPS SHIFT** e 9) e dura até que seja novamente carregada a mesma tecla ou um 9. Uma tecla numérica dá um mosaico gráfico, exceptuando as que têm **GRAPHICS** e **DELETE**, e cada tecla de letra exceptuando V, W, X, Y, Z, dará um caracter gráfico definido pelo utilizador.
Se alguma das teclas fôr permida durante mais do que três segundos ela repete-se a si própria automaticamente.

.../

Elementos introduzidos pelo teclado aparecem na metade inferior do écran à medida que são escritos, sendo cada um dos caracteres (símbolo único ou composto) inserido imediatamente antes do cursor. O cursor pode ser movido para a esquerda utilizando CAPS SHIFT e 5 ou para a direita utilizando CAPS SHIFT e 8. O caracter que se encontra imediatamente antes do cursor pode ser apagado com DELETE (CAPS SHIFT e Ø). (Nota: a linha inteira pode ser apagada premindo EDIT (CAPS SHIFT e 1) seguido de ENTER).

Quando se introduz um ENTER a linha é executada, introduzida no programa ou usada como dados de entrada (INPUT) conforme o caso, a não ser que ela contenha um erro de sintaxe. Neste caso um ? aparece imediatamente após o erro, piscando.

Á medida que as linhas do programa vão entrando, uma listagem aparece na metade superior do écran. A última linha entrada é chamada a linha corrente e é indicada pelo simbolo > ; este simbolo pode ser movido usando as teclas ↓ (CAPS SHIFT e 6) e ↑ (CAPS SHIFT e 7). Se se pressionar EDIT (CAPS SHIFT e 1), a linha corrente é puxada para a parte de baixo do écran e pode ser modificada.

Quando um comando é executado ou um programa é corrido, as saídas aparecem na metade superior do écran e aí ficam, até que se introduza uma linha de programa, ou se carregue em ENTER com uma linha em branco, ou ↑ ou ↓ sejam pressionadas. Na parte inferior aparece uma mensagem que dá um código (letra ou número) referido no Apendice B. A mensagem permanece no écran até que uma tecla seja pressionada (e indica o modo K).

Em algumas circunstâncias (CAPS SHIFT) com um espaço (SPACE) pode funcionar como uma paragem (BREAK) do computador com uma mensagem D ou L. Isto indica:

(I) parou no fim de uma instrução enquanto um programa era executado ou
(II) enquanto o computador está a usar o gravador de cassetes ou a impressora.

O écran de televisão

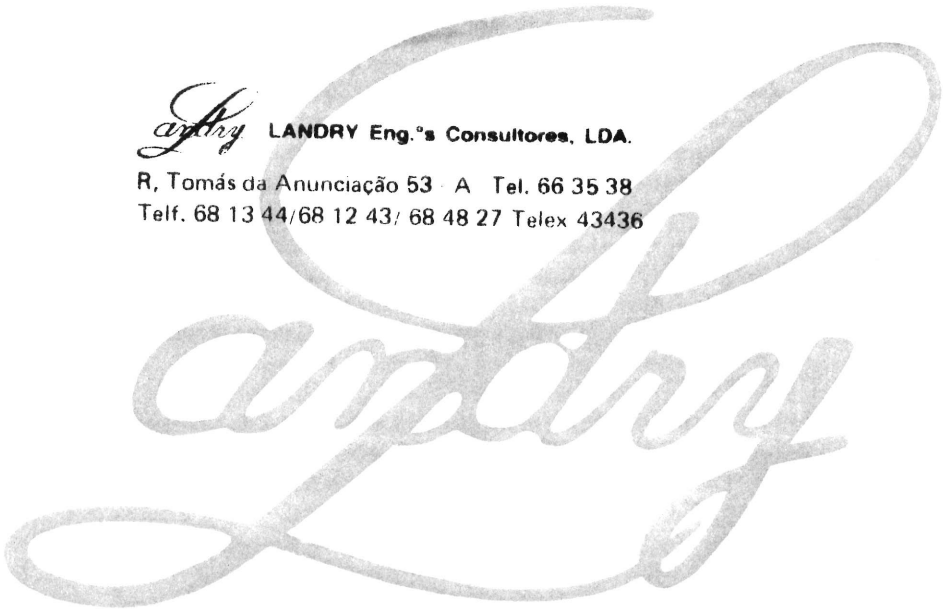
O écran tem 24 linhas, cada uma com 32 caracteres e está dividido em duas partes. A parte de cima tem no máximo 22 linhas e, ou mostra uma listagem, ou uma saída do programa. Quando a impressão na parte de cima atinge o fundo do écran, sobe tudo uma linha; se isto envolver perder uma linha que ainda não tinha tido oportunidade de ver, então o computador pára com a mensagem

.../

scroll?. Carregar nas teclas N SPACE ou STOP faz com que o programa seja interrompido com uma mensagem D BREAK - CONT' repeats; qualquer outra tecla permitirá que a impressão continue. A parte de baixo é utilizada para entrar comandos, linhas de programa, e entrada de dados (INPUT), e também para impressão de mensagens. A parte inferior começa com duas linhas (a de cima em branco), mas expande-se para se adaptar ao que está a ser escrito. Quando chegar a uma zona em que já está impresso algo na metade de cima, mais entradas farão com que a metade de cima vá subindo.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436



2

CAPÍTULO 2

Conceitos de Programação Básicos

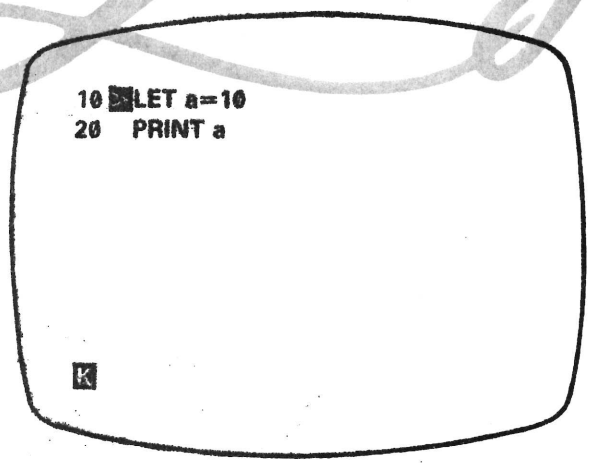
Sumário

Programas
Números de Linha
Alteração de programas usando ←, →, e EDIT
RUN, LIST
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
STOP em dados de entrada (INPUT)
BREAK

Introduza estas duas linhas de um programa de computador para imprimir a soma de dois números:

```
20 PRINT a
10 LET a=10
```

O écran aparecerá assim:



Como já sabe, como estas linhas começam com números, não são executadas imediatamente mas antes guardadas, como linhas de programa. Também deve ter reparado aqui que o número de linha governa a ordem pela qual as linhas se ordenam no programa: isto é muito importante quando o programa é executado, mas também se reflete na ordem das linhas na listagem que pode ver agora no ecran.

.../

Até agora só se introduziu um número, por isso escreva:

```
15 LET b=15
```

e introduza. Seria impossível inserir esta linha entre as duas primeiras se elas tivessem sido numeradas com 1 e 2 em vez de 1Ø e 2Ø (os números de linha têm de ser números inteiros entre 1 e 9999), e é por isso que quando se introduz um programa pela primeira vez é aconselhável deixar espaços entre os números de linha.
Agora é preciso mudar a linha 2Ø para

```
20 PRINT a+b
```

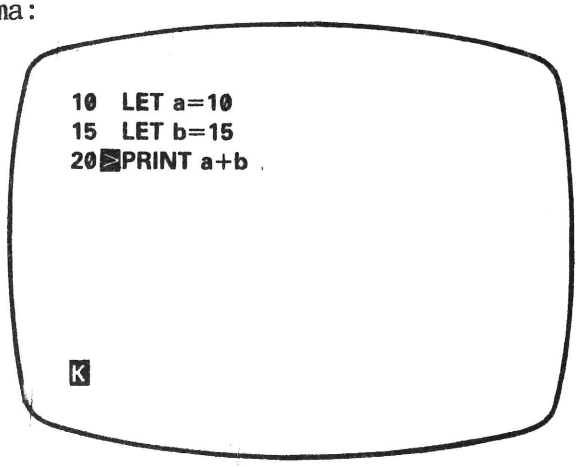
Poder-se-ia escrever a linha substituta por extenso, mas é mais cómodo usar a facilidade de edição (alteração-EDIT) que foi descrita no livro introdutório. O sinal ▶ na linha 15 é chamado o cursor do programa, e a linha que ele aponta é a linha corrente. Esta é normalmente a última linha que foi introduzida mas pode usar as teclas ↑ ou ↓ para mover o cursor do programa para cima ou para baixo. (Experimente, deixando eventualmente o cursor do programa na linha 2Ø).
Quando premir a tecla **EDIT**, uma cópia da linha corrente aparecerá no fundo do écran - no seu caso uma cópia da linha 2Ø. Carregue na tecla → até que o cursor ▮ se desloque para o fim da linha, depois escreva

```
+b          (sem carregar em ENTER)
```

a linha do fundo deve aparecer agora como

```
20 PRINT a+b + b
```

Carregue em **ENTER** e esta substituirá a antiga linha 2Ø, de forma que o écran agora terá a forma:



.../

Corra este programa usando **RUN** e **ENTER** e a soma aparecerá no écran.
Corra o programa outra vez e depois introduza

PRINT a, b

As variáveis ainda lá estão, ainda que o programa tenha acabado.
Há um modo muito útil de utilizar a tecla **EDIT** para nos livrarmos da parte de baixo do écran. Escreva uma quantidade de disparates (sem carregar em **ENTER**) e depois decida-se que afinal não quer essa linha. Um modo de a apagar é carregar na tecla **DELETE** até que a linha desapareça; mas outro modo é o seguinte. Se carregar em **EDIT**, os disparates no fundo do écran serão substituídos por uma cópia da linha corrente. Se agora carregar em **ENTER**, a linha corrente voltará a ser colocada no programa sem ser alterada, deixando a zona de baixo do écran vazia.
Se introduziu uma linha por engano, por exemplo

12 LET b=8 = 8

ela irá parar ao programa e você então notará que se enganou. Para apagar esta linha desnecessária escreva

12 (carregando em **ENTER** claro)

Observará com surpresa que o cursor do programa desapareceu.
Deve imaginar que ele está escondido entre as linhas 1Ø e 15, por isso se carregar em ↑ ele deslocar-se-á para cima, para a linha 1Ø, ao passo que se se carregar em ↓ deslocar-se-á para baixo para a linha 15.
Escreva

12 (e **ENTER**)

De novo o cursor de programa estará escondido entre as linhas 1Ø e 15.
Agora carregue em **EDIT** e a linha 15 aparecerá de novo em baixo: quando o cursor de programa está escondido entre duas linhas, a tecla **EDIT** transporta para baixo a linha seguinte ao novo número de linha. Carregue em **ENTER** para limpar a parte de baixo do écran.
Agora escreva

3Ø (e **ENTER**)

.../

Desta vez o cursor do programa está escondido depois do fim do programa: e se carregar em EDIT, então a linha 2Ø será transportada para baixo.
Por fim escreva

```
LIST 15
```

Verá agora no écran

```
15 LET b=5
20 PRINT a+b
```

A linha 1Ø desapareceu do écran, mas ainda está no seu programa - o que pode ser demonstrada carregando em ENTER. O único efeito de LIST 15 é produzir uma listagem que começa na linha 15, e colocar o cursor do programa na linha 15. Se tiver um programa muito longo, então o LIST provavelmente será um modo mais útil para mover o cursor do programa do que ▲ e ▼ .
Isto ilustra outra utilidade dos números de linha: actuam como nomes para as linhas do programa de modo a que nos possamos referir a eles, mais ou menos da mesma forma que os nomes para as variáveis.
O comando LIST sòzinho faz com que a listagem comece no princípio do programa.
Outro comando que foi visto no livro introdutório é:

```
NEW
```

Este comando apaga os velhos programas e variáveis no computador. Agora cuidadosamente introduza este programa que transforma graus Fahrenheit para graus centígrados.

```
10 REM Conversão de temperatura
20 PRINT "grau F", "grau C"
30 PRINT
40 INPUT "Entre grau F", F
50 PRINT F, (F-32) * 5/9
60 GO TO 40
```

As palavras que aparecem na linha 1Ø terão de ser introduzidas letra a le-

.../

tra. Também ainda que o comando **GO TO** tenha um espaço de permeio, é na verdade sómente uma palavra chave (na tecla **G**).

Agora execute-o. Verá o cabeçalho escrito no écran pela linha 2Ø, mas o que acontece com a linha 1Ø? Aparentemente o computador ignorou-a completamente. Bem, foi isso mesmo. O comando REM na linha 1Ø é um **comentário** (remark), ou um auxiliar de memória , e só lá está para o lembrar do que o programa faz. Uma instrução **REM** é constituida é constituida por um **REM** seguido por qualquer coisa que lá se quiera pôr e o computador ignorará tudo até ao fim da linha.

Agora o computador já chegou á instrução de **INPUT** na linha 4Ø e está á espera que você introduza o valor para a variavél F-pode-se ver isso porque onde se poderia estar á espera de um cursor **K** e aparece um cursor **L**. Introduza um número ; lembre-se de introduzir **ENTER** . Agora já o computador mostrou o resultado e está á espera de outro número . Isto porque a linha 6Ø, **GO TO 40** ,significa exactamente o que diz (vá para 4Ø). Em vez de correr o programa e parar, o computador salta de novo para a linha 4Ø e recomeça. Por isso introduza outra temperatura .

Depois de mais algumas passagens destas, poder-se-á perguntar se a máquina nunca se irá cansar de fazer isto, na realidade nunca se cansará ! Da próxima vez que ela lhe pedir outro número carregue **STOP** .O computador emite a mensagem **H STOP in INPUT in line 40:1**
que lhe diz porque é que ele parou e onde (na primeira instrução da linha 3Ø).

Se quiser continuar com o programa carregue em

CONTINUE

e o computador pedir-lhe-á outro número.

Quando se usa a instrução **CONTINUE** o computador lembra-se do número de linha na última mensagem que emitiu, desde que não fosse um **0 OK** , e volta para essa linha: no seu caso isso envolve saltar para a linha 4Ø, a instrução de **INPUT**.

Substitua a linha 6Ø por **GO TO 31** - não fará qualquer alteração notória no programa a ser executado. Se o número de linha de uma instrução **GO TO** se referir a uma linha inexistente, então o salto é para a linha seguinte ao número dado. O mesmo acontece com um **RUN**; de facto a instrução **RUN** só por si significa sòmente **RUN 0**.

Agora introduza números até que o écran comece a ficar cheio. Quando estiver cheio o computador irá mover a parte de cima do écran uma linha para cima para abrir espaço, perdendo a primeira linha do topo. A isto chama-se scrolling.

.../

Quando estiver cansado disto pare o programa usando um **STOP** e obtenha a listagem carregando em **ENTER.**

Olhe para a instrução **PRINT** na linha 5Ø. A pontuação nela - a vírgula (,) - - é muito importante e deve lembrar-se que segue regras muito mais restritas do que a pontuação em Português.

As vírgulas são usadas para fazer com que a impressão se faça ou na margem esquerda ou no meio do écran, dependendo do que vem a seguir. Assim, na linha 5Ø, a vírgula faz com que a temperatura centígrada seja impressa no meio da linha. Com um ponto-e-vírgula, por outro lado, o número ou cadeia de letras que se segue é impresso imediatamente após o anterior. Pode-se ver isto na linha 5Ø, se a vírgula fôr substituída por um ponto-e-vírgula.

Outro sinal de pontuação que se podo usar assim na instrução **PRINT** é o apóstrofo ('). Isto faz com que o que fôr impresso a seguir a ele, ou seja na linha seguinte, mas isto acontece de qualquer maneira no fim de cada instrução **PRINT**, por isso não precisará muito do apostrofo. É por isso que a instruçao **PRINT** na linha 5Ø começa sempre a impressão numa nova linha, e é também por isso que a instrução **PRINT** na linha 3Ø produz uma linha em branco.

Se quiser evitar isto, de modo que a seguir a uma instrução **PRINT** a próxima continue na mesma linha, pode colocar uma vírgula ou um ponto-e-vírgula no fim da primeira. Para ver como isto funciona, substitua a linha 5Ø consecutivamente pelas seguintes:

```
50 PRINT F,
50 PRINT F;
```

```
50 PRINT F
```

e execute cada uma das versões - para se habituar tente também

```
50 PRINT F'
```

Aquele que tem a vírgula distribui tudo em duas colunas, aquele que tem o ponto-e-vírgula acomula tudo numa só, aquele que não tem neuhuma das coisas dá uma linha para cada número, bem como a que tem o apóstrofo - o apóstrofo dá uma nova linha só por si, mas inibe a que é dada automaticamente.

.../

Lembre-se da diferença entre vírgula e ponto-e-vírgula na instrução **PRINT**; mas não os confunda com os dois pontos (:) que são usados para separar as instruções numa mesma linha.
Agora introduza estas novas linhas:

```
100 REM este programa bem educado lembra-se do seu
    nome
110 INPUT n$
120 PRINT
130 GO TO 110 "Olá " n$ "
```

Este programa é separado do anterior, mas pode manter ambos no computador ao mesmo tempo. Para executar o novo programa introduza:

```
RUN 100
```

Dado que este programa pede como entrada uma cadeia de letras em vez de um número, também imprime as aspas - isto é para que você se lembre e geral - mente também lhe evita algum trabalho de teclas. Experimente uma vez com qualquer nome pressuposto que queira inventar para si mesmo.
Da próxima vez também voltará a usar duas aspas, mas não precisa de as usar se não quiser. Experimente isto por exemplo. Apague-as (com ◗ e **DELETE** duas vezes) e escreva:

```
n$
```

Uma vez que não há aspas, o computador sabe que tem de fazer alguns cálculos: o cálculo neste caso é saber o valor da variável literal chamada n$, que é o nome que você introduziu da última vez. Claro que a instrução de **INPUT** actua como **LET n$=n$** , de modo que o valor de n$ não é modificado.
Para a próxima vez, para comparar, introduza:

```
n$
```

de novo, mas desta vez sem retirar as aspas. Agora, só para o confundir a variável n$ tem o valor "n$".
Se quiser usar o **STOP** para uma entrada não numerica, tem primeiro de deslocar o cursor para o princípio da linha usando ◖ .

.../

Agora volte a olhar para a instrução **RUN 100** que tinhamos antes. Esta instrução só faz saltar para a linha 1ØØ, e então, não poderíamos ter dito antes GO TO 1ØØ? Neste caso acontece que a resposta é sim, mas há uma diferença. **RUN 100** primeiro limpa todas as variáveis e o écran, e depois disso funciona exactamente como **GO TO 100**. A instrução **GO TO 100** não limpa nada. Pode haver ocasiões em que queira correr um programa sem apagar as variáveis; aqui a instrução **GO TO** seria necessária e **RUN** era desastroso, por isso é melhor não se habituar a carregar em **RUN** cada vez que quiser correr um programa.

Outra diferença é que pode-se introduzir **RUN** sem o fazer seguir por um número de linha e ele começa a execução na primeira linha do programa. A instrução **GO TO** tem sempre de ter um número de linha.

Ambos estes programas pararam porque se introduziu um **STOP** na linha de entrada; por vezes - por engano - escreve-se um programa que não se pode parar e que não pára sozinho. Escreva:

```
200 GO TO 200
RUN 200
```

Parece que o computador está disposto a continuar sempre a não ser que se tire a ficha; mas há um remédio menos drástico. Pressione **CAPS SHIFT** juntamente com a tecla **SPACE**, que tem **BREAK** escrito por cima. O programa parará, dizendo **L BREAK** into program.

No fim de cada instrução o programa pára para ver se alguma dessas teclas foi pressionada, e se tiverem sido pára. A tecla **BREAK** também pode ser usada quando se estiver no meio da utilização do gravador de cassetes ou da impressora, ou de outras peças de equipamento que possa ligar ao seu computador - no caso de o computador estar à espera que eles façam alguma coisa e eles não o façam.

Nestes casos a mensagem é diferente, **D BREAK - CONT** repeats. Neste caso a instrução **CONTINUE** (e na verdade na maior parte dos outros casos também) repete a instrução onde o programa foi parado; mas depois de uma mensagem **L BREAK** into program, um **CONTINUE** continua para a instrução seguinte àquela em que se parou, tendo em conta qualquer salto que possa ocorrer.

Corra o programa do nome e quando ele lhe pedir os dados escreva:

n$ (depois de retirar as aspas)

.../

n$ é uma variável indefinida e receber-se-á uma mensagem de erro 2 Variable not found (Variável não encontrada). Se agora escrever

```
LET n$= "algo definido"
```

(que tem a sua própir mensagem de Ø OK, Ø:1) e

```
CONTINUE
```

vera que pode usar n$ como dado de entrada sem qualquer problema.
Neste caso a instrução **CONTINUE** salta para a instrução **INPUT** na linha 11Ø. Não considera a mensagem da instrução **LET** porque estava tudo bem, e salta para a instrução referida na mensagem anterior, a primeira instrução da linha 11Ø. Espera-se que isto seja útil. Se um programa parar com um erro pode-se fazer tudo o que se possa imaginar para reparar a instrução **CONTINUE** continuará a funcionar depois.
Como dissemos antes a mensagem **L BREAK** into program é especial, porque depois dela a instrução **CONTINUE** não repete a instrução onde o programa perou.
As listagens automáticas (aquelas que não são resultado de uma instrução LIST mas que ocorrem sempre que se introduz uma nova linha) devem tê-lo confundido. Se introduzir um programa com 5Ø linhas, todas com instrução **REM**

```
1 REM
2 REM
3 REM
: :
: :
49 REM
50 REM
```

então poderá experimentar.

A primeira coisa a lembrar-se é que a linha corrente (a que está marcada com ▶) aparecerá sempre no écran e geralmente perto do centro.
Carregue em

LIST (e **ENTER** claro)

e quando o computador perguntar scroll? (porque encheu o écran) carregue no n para não dizer não. O computador dará a mensagem **D BREAK - CONT** repeats como se tivesse carregado em **BREAK**. Noutra altura poderá ver o que acontece se carregar no v em vez de no n; n, **SPACE** e **STOP** contam como não, e tudo o resto como sim.
Agora carregue em **ENTER** de novo para obter a listagem automatica e verá aparecerem no écran as linhas de 1 a 22. Agora escreva:

23 REM

e fica com as linhas entre 2 e 23 no écran; escreva

28 REM

e fica com as linhas entre 27 e 48. (em ambos os casos, escrevendo uma nova linha, moveu o cursor do programa produzindo uma nova listagem).
Talvez isto pareça um pouco arbitrário para si. No entanto está precisamente a tentar dar-lhe o que você quer, mas sendo os seres humanos criaturas pouco previsíveis, nem sempre advinha certo.
O computador mantem registada não só a linha corrente, aquela que tem de aparecer no écran, mas também a linha de cima do écran. Quando tenta fazer uma listagem, a primeira coisa que faz é comparar a linha que está no cimo com a linha corrente.
Se a linha de cima vier depois, então não vale a pena começar ali, por isso usa a linha corrente como nova linha de cima e faz a listagem.
Se isso não acontecer, o método dele é começar a fazer a listagem a partir da linha de cima, e continuar até que tenha aparecido a linha corrente, puxando para cima conforme necessário. No entanto primeiro faz um cálculo grosseiro para ver quanto tempo isso vai demorar, e se a resposta fôr um tempo demasiado grande, então move a linha de cima para baixo de forma a ficar muito mais perto da linha corrente. Agora, depois de ter determinado a sua linha de cima, começa a listagem a partir daí. Se, quando chega ao fim do programa, ou ao fundo do écran, a linha corrente já foi listada,

.../

pára. De outra forma continua a fazer scroll até que a linha corrente apareça no écran, e por cada linha a mais que ele escreve desce a linha de cima de forma que a linha de cima se desloca em direcção à vizinhança da linha corrente.
Experimente modificar a linha corrente escrevendo

número de linha **REM**

A instrução **LIST** move a linha corrente mas não a linha de cima, de modo que listagens consecutivas podem ser diferentes. Por exemplo, carregue em

LIST

para obter uma listagem através da instrução **LIST** e depois carregue em **ENTER** de novo para tornar a linha Ø a linha de cima. Deve ter as linhas de 1 a 22 no écran. Escreva:

LIST 22

o que lhe dá as linhas de 22 a 43; quando carregar novamente em **ENTER**, volta a ter as linhas de 1 a 22. Isto é mais útil para programas curtos do que para programas longos.

Usando o programa cheio de instruções **REM** que foi dado acima, escreva

LIST

e depois um n quando ele lhe perguntar scroll?. Agora escreva:

CONTINUE

A instrução **CONTINUE** é um pouco escranha aqui, porque a parte de baixo do écran apaga-se; mas pode restaurar a normalidade com um **BREAK**. O motivo foi porque a instrução **LIST** foi a primeira na linha, de modo que Continue repete esta instrução. Infelizmente o primeiro comando da linha é agora **CONTINUE** de modo que o computador não pára de fazer **CONTINUE** até que nós o obriguemos a parar.
Pode arranjar isso substituindo a instrução **LIST** por

: LIST

para o qual a instrução **CONTINUE** dá a mensagem **0 OK** (porque a instrução **CONTINUE** salta para a segunda instrução da linha, que é o fim) ou

:: **LIST**

ao que a instrução **CONTINUE** dá como resposta N Statement lost (porque o **CONTINUE** salta para a terceira instrução na linha, que já não existe). Acabámos de ver as instruções **PRINT**, **LET**, **INPUT**, **RUN**, **LIST**, **GO TO**, **CONTINUE**, **NEW** e **REM** e todas podem ser usadas como comandos directos ou em linhas de programa - isto é verdade para quase todas as instruções no BASIC do ZX Spectrum. **RUN**, **LIST**, **CONTINUE** e **NEW** não são geralmente muito utilizados em programas, mas podem ser usados.

Exercícios:

1. Ponha uma instrução **LIST** num programa, de modo a que quando o tentar executar ele se liste a ele próprio.
2. Escreva um programa que tenha como entrada preços e que escreva taxas de desconto (por exemplo 15 por cento). Coloque instruções **PRINT** para que o computador anuncie o que vai fazer, e peça os preços com uma delicadeza extravagante. Modifique o programa de modo a que também possa introduzir a taxa de juro (para permitir taxas nulas ou posteriores alterações).
3. Escreva um programa para imprimir um conjunto de números que você lhe dê, adicionados (Sugestão: escolha duas variáveis, uma chamada total - - que começa em Ø - e item. Introduza o item e adicdone-o ao total, imprima ambos e volte ao princípio).
4. O que é que as instruções **CONTINUE** e **NEW** fariam num programa? Consegue imaginar alguma utilidade para essas instruções?

.../

LANDRY Eng.ºs Consultores, LDA.
R. Tomás da Anunciação 53 A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

3

CAPÍTULO 3

Decisões

Sumário

IF, STOP

=, <, >, <=, >=, <>

Todos os programas vistos até aqui têm sido bastante previsiveis - executam todas as instruções de seguida e voltam de novo ao princípio. Isto não é no entanto muito útil. Na realidade, do computador, seria de esperar que tomasse decisões e que agisse de acordo com elas. A instrução usada tem a forma ... **IF**, (se), algo é verdadeiro, ou não, **THEN** (então) faça-se outra coisa.

Por exemplo, utilize **NEW** para apagar o programa anterior que esteja metido no computador, e escreva o seguinte programa (este é feito para se jogar a dois).

```
10 REM Adivinhe o número
20 INPUT a: CLS
30 INPUT "adivinhe o número", b
40 IF b=a THEN PRINT     "Está correcto": STOP
50 IF b<a THEN PRINT       "É demasiado pequeno,
                           tente outra vez
60 IF b>a THEN PRINT       "É demasiado grande,
                           tente outra vez"
70 GO TO 30
```

Pode ver que uma instrução **IF** toma a forma:

IF condição **THEN** ...

onde as '...' significam uma sequência de comandosx separados por dois pontos da forma usual. A condição é algo que vai ser calculada por forma a dar verdadeiro ou falso: se fôr verdadeira então as instruções no resto da linha, que se seguem ao **THEN**, são executadas, de outra forma serão passadas por cima e a execução do programa continuará na linha seguinte.

.../

A condição mais simples é comprar dois números ou duas cadeias de letras: pode-se testar se dois números são iguais, ou se um é maior que o outro, e pode-se testar se duas cadeias de letras são iguais, ou se (aproximadamente) uma vem antes da outra na ordem alfabética. Usam-se as relações

=, <, >, <=, >= e <> .

= significa 'igual'. Ainda que seja o mesmo símbolo que na instrução **LET** é usado de um modo muito diferente.

< (**SYMBOL SHIFT** com um **R**) significa 'menor do que', de forma que

```
1<2
-2<-1
-3<1
```

são proposições todas verdadeiras, mas

```
1<Ø
Ø<-2
```

são falsas.

A linha 4Ø compara **a** com **b** . Se forem iguais o programa é parado pela instrução **STOP**. A mensagem no fundo do écran é 9 **STOP**, statement, 4Ø:3 mostra que a terceira instrução, ou comando, na linha 4Ø provocou uma paragem do programa, isto é o **STOP**. A linha 5Ø verifica se **b** é menor do que **a**, e a linha 6Ø se **b** é maior do que **a**. Se uma destas condições é verdade o comentário é emitido, e o programa avança para a linha 7Ø que diz ao computador para voltar à linha 3Ø e recomeçar.

A instrução **CLS** na linha 2Ø era para impedir que a outra pessoa visse o numero introduzido.

Assim > (**SYMBOL SHIFT** com **T**) significa 'maior que', e é exactamente como < mas ao contrário. Para se lembrar de qual é qual repare que a ponta mais fina aponta para o numero que se espera que seja menor.

<= (**SYMBOL SHIFT** com **Q** - não escreva um < seguido de =) significa 'menor ou igual a', de forma que é igual ao < excepto que é verdade mesmo quando os números são iguais: assim 2 <= 2 é verdade, mas 2 < 2 é falso.

>= (**SYMBOL SHIFT** com **E**) significa 'maior ou igual a' e é semelhante a > .

<> (**SYMBOL SHIFT** com **W**) significa 'é diferente de', o significado oposto de =.

Os matemáticos geralmente escrevem <=, >= = e <> na forma ≤, ≥ e ≠.

..../

Também escrevem coisas como '2 < 3 < 4' o que significa '2 < 3 e 3 < 4', mas isto não é possível em BASIC.
Nota: em algumas versões de BASIC - mas não no ZX Spectrum - a instrução **IF** pode ter a forma

IF condição THEN número de linha

isto tem o mesmo significado que

IF condição **THEN GO TO** número de linha

Exercícios

1. Experimente este programa:

```
10 PRINT "x": STOP: PRINT "y"
```

Quando o correr o programa imprimirá o **x** e parará com a mensagem **9 STOP** statement, **10: 2**. Agora escreva

```
CONTINUE
```

Podia esperar que ele saltasse de novo para a instrução **STOP** - a instrução **CONTINUE** normalmente repete a instrução que é referida na mensagem. No entanto, aqui isso não seria muito útil porque o computador pararia outra vez sem imprimir o **y**. Assim as coisas estão organizadas por forma a que depois da mensagem 9 a instrução **CONTINUE** salte para a instrução depois da instrução **STOP** - por isso no nosso exemplo, depois de **CONTINUE** o computador imprime **y** e chega ao fim do programa.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

4

CAPÍTULO 4

Ciclos

Sumário

FOR, NEXT
TO. STEP

Suponha que quer dar cinco números e adicioná-los. Um modo de o fazer seria (não introduza este programa a não ser que sinta dúvidas) escrever.

```
10 LET total=0
20 INPUT a
30 LET total=total+a
40 INPUT a
50 LET total=total+a
60 INPUT a
70 LET total=total+a
80 INPUT a
90 LET total=total+a
100 INPUT a
110 LET total=total+a
120 PRINT total
```

Este método de programação não é muito razoável. Pode ser perfeitamente controlável para cinco números, mas pode imaginar como seria cansativo um programa destes para adicionar dez números, e para somar cem seria praticamente impossível.
Muito melhor será arranjar uma variável que conte até cinco e depois pare o programa, assim (este programa deve ser introduzido):

```
10 LET total=0
20 LET contador =1
30 INPUT a
40 REM o contador tem o número de vezes que já
foi introduzido um número
50 LET total=total+a
60 LET contador = contador +1
70 IF contador <=5 THEN GO TO 30
80 PRINT total
```

.../

Note como é fácil mudar a linha 7Ø para que este programa somasse dez números, ou mesmo cem.
Este género de contagem é tão útil que até há dois comandos especiais para o tornar mais fácil: o comando **FOR** e o comando **NEXT**. São sempre usados juntos.
Usando estes comandos, o programa que acabou de introduzir faria exactamente o mesmo se fosse assim:

```
10 LET total=0
20 FOR c=1 TO 5
30 INPUT a
40 REM c = número de entradas já feitas
50 LET total=total+a
60 NEXT c
80 PRINT total
```

(Para obter este programa a partir do anterior, só tem de alterar as linhas 2Ø, 4Ø, 6Ø e 7Ø. TO obtém-se com **SYMBOL SHIFT** e **F**).
Repare que modificamos o contador para **c**. A variável de contagem de um ciclo **FOR NEXT** (também chamada variável de controle)tem de ter uma única letra como nome.
Este programa faz com que c passe pelos valores 1 (valor inicial), 2, 3, 4 e 5 (o limite), e para cada uma delas execute as linhas 3Ø, 4Ø e 5Ø. Depois, quando o c acabou de percorrer os seus cinco valores, a linha 8Ø é executada.
Uma subtileza suplementar a esta instrução é que a variável de controle não tem de subir de 1 em 1: pode-se alterar esse 1 para outra coisa qualquer usando um **STEP** (passo) na instrução **FOR**. A forma mais geral para a instrução **FOR** é

FOR variável de controle = valor inicial **TO** limite **STEP** incremento

onde o variável de controle é uma única letra, e o valor inicial, o limite e o incremento são tudo coisas que o computador possa calcular como números - como números em si, ou somas, ou nomes de variáveis numéricas. Por isso, se substituir a linha 2Ø do programa por

```
20 FOR c=1 TO 5 STEP 3/2
```

.../

o e percorrerá os valores 1, 2, 5, 4. Repare que não precisa de restringir a números inteiros, e também que a variável de controle não precisa de passar exactamente pelo valor do limite - o ciclo continua enquanto ela fôr menor ou igual ao limite.

Experimente este programa, para imprimir números de 1 a 1Ø por ordem inversa.

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Dissemos antes que o programa continuaria o ciclo enquanto a variável de controle fosse menor ou igual ao limite. Se fizer isso neste exemplo verá que isso é um disparate. A regra normal tem de ser modificada: quando o incremento é negativo o programa continua o ciclo enquanto a variável de controle fôr maior ou igual ao limite.
Tem de ter cuidado se estiver a fazer dois ciclos FOR - NEXT ao mesmo tempo, um dentro do outro. Experimente este programa, que imprime os números de um jogo completo de dominó:

```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT n
50 PRINT
60 NEXT m
```

ciclo n — ciclo m

Pode verificar que o ciclo n está completamente dentro do ciclo m - estão adequadamente enquadrados. O que tem de ser evitado é ter dois ciclo FOR - - NEXT que se sobrepõem sem um deles estar completamente dentro do outro, assim,

```
5 REM este programa está errado

10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT m
50 PRINT
60 NEXT n
```

ciclo m — ciclo n

.../

Dois ciclos **FOR - NEXT** devem ou estar um dentro do outro ou completamente separados.
Outra coisa que se deve evitar é saltar para o meio de um ciclo **FOR - NEXT** a partir do exterior. A variável de controle só é colocada na posição apropriada quando a instrução **FOR** é executada, e se esta faltar a instrução **NEXT** irá confundir o computador. Provavelmente obterá um erro do tipo **NEXT** without **FOR** (**NEXT** sem **FOR**) ou variable not found (variável não encontrada).
Não há nada que o impeça de usar um **FOR** e **NEXT** num comando directo. Experimente por exemplo:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

Pode por vezes usar isto como um modo (algo artificial) de circundar o problema de não poder ter instruções **GO TO** num comando directo - porque um comando não tem número de linha.
Por exemplo:

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

O incremento nulo aqui faz com que o comando se repita indefenidamente. Este género de coisas na verdade não é recomendável porque se lhe surge um erro perdeu a instrução e terá de a escrever toda outra vez - e um **CONTINUE** não dará resultado.

Exercícios

1. Uma variável de controle não tem somente um nome e um valor, como uma variável vulgar, mas também tem um limite e um incremento, e uma referência à instrução seguinte à correspondente instrução **FOR**. Convença-se que quando é executada uma instrução **FOR** toda esta informação está disponível (usando o valor inicial como o primeiro valor que a variável toma), e também que esta informação é suficiente para que a instrução **NEXT** saiba de quanto deve aumentar o valor, se deve voltar para trás, e, se fôr esse o caso, para onde saltar.

2. Corra o terceiro programa que lhe foi apresentado neste capítulo e depois escreva:

.../

```
PRINT c
```

Porque é que a resposta é 6 e não 5?

(Resposta: a instrução **NEXT** na linha 6Ø é executada cinco vezes, e de cada vez soma-se 1 a **c**. Na última vez **c** torna-se 6; e depois a instrução **NEXT** decide não voltar para trás, mas sim continuar uma vez que **c** já passou o limite).
O que acontece se colocar **STEP 2** na linha 2Ø?

3. Modifique o terceiro programa para que em vez de adicionar automaticamente cinco números ele lhe pergunte quantos números quer somar. Quando correr esse programa, o que acontece se introduzir Ø, significando que não quer somar mais números? Porque acha que isto pode causar problemas ao computador apesar de ser claro o que você quer dizer? (O computador tem de procurar o comando **NEXT c**, o que geralmente não é necessário. De facto tudo isto foi tratado.

4. Na linha 1Ø do quarto programa acima, modifique **10** para **100** e execute o programa. Este programa vai imprimir os números entre 1ØØ e 89 no écran e depois pergunta scroll? no fim. Isto é para lhe dar oportunidade de ver os números que vão ser apagados em cima. Se carregar em **n**, **STOP** ou em **BREAK** o programa será interrompido com a mensagem **D BREAK - CONT** repeats. Se premir qualquer outra tecla ele escreverá outras 22 linhas e perguntará de novo.

5. Apague a linha 3Ø do quarto programa. Quando executar o novo programa reduzido, ele vai escrever o primeiro número com a mensagem **Ø OK**. Se escrever:

```
NEXT n
```

o programa irá dar nova volta ao ciclo, escrevendo o número seguinte.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

5

CAPITULO 5

Subrotinas

Sumário

GO SUB RETURN

Por vezes partes diferentes de um programa terão funções muito semelhantes a executar, e verá que tem de introduzir as mesmas linhas duas ou mais vezes; no entanto isto não é necessário. Pode escrever as linhas uma só vez, numa forma conhecida como subrotina, e depois usar, ou chamar, essas li - nhas de qualquer parte do programa sem ter de as escrever outra vez.
Para fazer isto usam-se as instruções **GO SUB** (ir para a subrotina) e **RETURN** (voltar).
Isto toma a forma

GO SUB n

onde n é um número de linha, da primeira linha da subrotina. É exactamente como **GO TO** n excepto que o computador se lembra onde estava a instrução **GO SUB** para voltar para lá depois de executar a subrotina. Faz isto colando o número da linha e o número da instrução dentro da linha (juntos estes dois números constituem o endereço de retorno) em cima de uma pilha de endereços (a pilha - stack - **GO SUB**).

RETURN

toma o endereço de cima da pilha do **GO SUB** e vai para a instrução que se segue a ele.
Como exemplo vamos voltar ao programa de advinhação de números. Volte a escrevê-lo da maneira seguinte:

```
10 REM "Um jogo de advinha re-arranjado"
20 INPUT a: CLS
30 INPUT "Advinhe o numero", b
40 IF a=b THEN PRINT "Esta correcto": STOP
50 IF a<b THEN GO SUB 100
60 IF a>b THEN GO SUB100
```

.../

```
70 GO TO 30
100 PRINT "Experimente outra vez"
110 RETURN
```

A instrução **GO TO** na linha 7Ø é muito importante porque de outra forma o programa entraria na subrotina e causaria um erro (**7 RETURN** without **GO SUB**) - **RETURN** sem **GO SUB**) quando a instrução RETURN fôr atingida.
Aqui está outro programa bastante elementar ilustrando o uso de **GO SUB.**

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 TO x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Quando este programa fôr executado, veja se percebe o que se está a passar. A subrotina começa na linha 5ØØ.
Uma subrotina pode muito bem chamar outra, ou mesmo ele própria (uma sub rotina que se chama a si própria diz-se de recorrência), por isso não tenha medo de usar várias camadas de subrotinas.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

6

CAPÍTULO 6

READ, DATA, RESTORE

Sumário

READ, DATA, RESTORE

Em alguns dos programas apresentados anteriormente vimos que a informação, ou os dados, podem ser introduzidos directamente no computador utilizando a instrução **INPUT**. Por vezes isto pode ser muito cansativo, especialmente se houver muitos dados que têm de ser repetidos cada vez que o programa é executado. Pode poupar muito tempo usando as instruções **READ**, **DATA** e **RESTORE**. Por exemplo:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 10,20,30
40 STOP
```

Uma instrução **READ** (leitura) consiste de um **READ** seguido de uma lista de variáveis, separadas por vírgulas. Funciona mais ou menos como uma instrução de **INPUT**, só que em vez de esperar que você introduza os valores a dar às variáveis, o computador procura esses valores na instrução **DATA**. Cada instrução **DATA** é uma lista de expressões - expressões numéricas ou literais - separadas por vírgulas. Pode colocá-las em qualquer ponto do programa, porque o computador as ignora excepto quando está a fazer um **READ**. Tem de imaginar que as expressões de todas as instruções **DATA** no programa são colocadas juntamente numa longa lista de expressões, a lista de dados (**DATA**). A primeira vez que o computador vai a uma instrução **READ**, retira a primeira expressão da lista de dados; da vez seguinte retira a seguinte; e assim, à medida que vai encontrando instruções **READ** sucessivas, vai avançando por dentro da lista de dados das instruções **DATA**. (Se se tentar passar para além do fim da lista **DATA** teremos um erro).
Repare que é perda de tempo colocar instruções **DATA** num comando directo porque o **READ** nunca os encontrará. As instruções **DATA** têm de entrar num programa.
Vamos ver como é que isto se enquadra no programa que acabou de introduzir. A linha 1Ø diz ao computador para ler três elementos de informação e

.../

atribui-los às variáveis **a, b** e **c**. A linha 2Ø diz então que esses valores devem ser impressos (**PRINT**). A instrução **DATA** na linha 3Ø dá os valores para **a, b** e **c**. A linha 4Ø interrompe o programa. Para ver a ordem pela qual as coisas funcionam modifique a linha 2Ø para:

```
20 PRINT b,c,a
```

A informação na instrução **DATA** pode fazer parte de um ciclo **FOR.... NEXT.** Escreva:

```
10 FOR n=1 TO 6
20 READ D
30 DATA 2,4,6,8,10,12
40 PRINT D
50 NEXT n
60 STOP
```

Quando este programa fôr executado pode ver a instrução **READ** a varrer a lista de dados da instrução **DATA**. As instruções **DATA** também podem conter variáveis de cadeia de letras. Por exemplo:

```
10 READ d$
20 PRINT "A data de hoje é", d$
30 DATA "1 de Julho de 1982"
40 STOP
```

Esta é a forma mais simples de ir buscar expressões à lista de dados **DATA**: começar no princípio e avançar até chegar ao fim. No entanto pode-se fazer o computador saltar pela lista **DATA**, usando a instrução **RESTORE**. Esta instrução é constituída por **RESTORE** seguido por um número de linha, e faz com que as instruções **READ** que se seguem comecem a obter os seus dados na primeira instrução **DATA** em ou para a frente do número da linha dado. (Pode-se omitir o número de linha, caso em que é como se se tivesse escrito o número da primeira linha do programa).
Experimente este programa:

.../

```
10 READ a,b
20 PRINT a,b
30 RESTORE 10
40 READ x,y,z
50 PRINT x,y,z
60 DATA 1,2,3
70 STOP
```

Neste programa os dados requeridos pela linha 1Ø fez com que **a**=1 e **b**=2. A instrução **RESTORE** 1Ø repõe as variáveis e permite que **x** , **y** e **z** sejam lidos a partir do primeiro número da instrução **DATA**. Volte a correr este programa sem a linha 3Ø e veja o que acontece.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

7

CAPÍTULO 7

Expressões

Sumário

Operações: +, −, *, /
Expressões, notação científica, nomes de variáveis.

Já vimos alguns dos modos que o ZX Spectrum usa para trabalhar com números. Pode executar as quatro operações aritméticas +, −, * e / (lembre-se que * é usado para a multiplicação, e / é usada para a divisão), e pode encontrar o valor de uma variável se lhe dermos o nome.
O exemplo:

```
LET TAX    = soma *15/100
```

dá-nos um indício do facto importante de que os cálculos pode ser combinados. Uma tal combinação, como soma *15/100 , é chamada uma expressão: assim umr expressão é um modo rápido de dizer ao computador para fazer vá - rios cálculos, um após o outro. No nosso exemplo, a expressão soma *15/100 significa 'procure o valor da variável chamada "soma", multiplique--o por 15 e divida-o por 1ØØ'.
Se ainda não o fez recomendamos-lhe que passe uma vista de olhos pelo livro introdutório para ver como o ZX Spectrum trata os números, e a ordem pela qual calcula as expressões metemáticas.
Recapitulando:
As multiplicações e divisões são feitas em primeiro lugar. Têm uma prioridade mais elevada do que a adição e a subtração. Relativamente uma à outra a multiplicação e divisão têm a mesma prioridade, o que significa que as multiplicações e divisões são feitas pela ordem em que aparecem da esquerda para a direita. Quando tiverem acabado, seguem-se as adições e subtrações - estas também têm a mesma prioridade uma que a outra, por isso executá-las-emos em ordem, da esquerda para a direita.
Ainda que tudo o que você precise de saber seja se a operação tem uma prioridade mais elevada ou mais baixa do que outra, o computador faz isto atribuindo um número entre 1 e 16 que representa a prioridade de cada opera - ção: * e / têm prioridade 8, e + e - têm prioridade 6.

.../

Esta ordem de cálculo é absolutamente rígida mas pode-se obviar pela utilização da parentesis: tudo o que estiver entre parentesis é calculado em primeiro lugar e depois é tratado como um único número.
As expressões são úteis porque sempre que o computador está à espera de um número, lhe pode ser dada uma expressão e ele calcula a resposta. As excep - ções a esta regra são tão poucas que serão explicitadas em cada caso.
Podem-se adicionar quantas cadeias se quiserem (ou variáveis literais) numa única expressão, bem como usar parentesis.
Na verdade deviamos dizer-lhe o que é que pode ou não ser usado como nomes de variáveis. Como já dissemos o nome de uma variável literal tem de ser uma única letra seguida por um **$**; e o nome de uma variável de controle para um ciclo **FOR - NEXT** tem de ser uma letra única; mas os nomes de variáveis numéricas ordinárias são muito mais livres. Podem-se usar quaisquer letras ou digitos desde que a primeira seja uma letra. Também se podem colocar espaços pelo meio para a tornar mais fácil de ler, mas esses não contarão como parte do nome.
Não faz qualquer diferença que o nome seja escrito com letras maiusculas ou minúsculas.
Aqui estão alguns exemplos de nome de variáveis que são permitidos:

x
t42
este nome é tão comprido que nunca serei capaz de o reescrever sem cometer algum erro
AGORA SOMOS SEIS (estes dois últimos nomes são considerados o mesmo e refe
aGORASomoSeiS rem-se à mesma variável)

Os seguintes não podem ser nomes de variáveis:

2001	(começa com um digito)
3 ursinhos	(começa com um digito)
M*A*S*H	(* não é letra nem digito)
Fotherington-Thomas	(- não é letra nem digito)

Expressões numéricas podem ser representadas por um número e um expoente: de novo reveja o livro introdutório. Experimente os seguintes para comprovar:

```
PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
```

.../

e assim por diante até

```
PRINT 2.34e15
```

Verá que a partir de certa altura o computador também começará a usar notação científica. Isto é porque não se podem usar mais do que catorze caracteres para escrever um número. À semelhança experimente

```
PRINT 2.34e-1
PRINT 2.34e-2
```

e assim por diante.
A instrução **PRINT** só dá oito digitos significativos de um número. Experimente

```
PRINT 4294967295, 4294967295-429e7
```

Isto prova que o computador é capaz de guardar todos os digitos de 4294967295, ainda que não apareçam no écran todos ao mesmo tempo.
O ZX Spectrum usa aritmética de ponto flutuante, o que significa que ele guarda separadamente os digitos do número (a sua mantissa) e a posição da virgula ou ponto decimal (o expoente). Isto não é sempre certo, mesmo para números inteiros. Escreva

```
PRINT 1e10+1-1e10,1e10-1e10+1
```

Os números são guardados com cerca de nove e meio digitos significativos, de modo de 1 e 1Ø é grande demais para se poder guardar exactamente. A imprecisão (na verdade cerca de 2) é maior do que 1, de modo que os números 1 e 1Ø e 1 e 1Ø + 1 parecem iguais ao computador.
Para ver um exemplo ainda mais peculiar escreva

```
PRINT 5e9+1-5e9
```

Aqui a imprecisão é de cerca de 1 e o 1 que se adiciona acaba por ser arre - dondado para 2. Os números 5 e 9 + 1 e 5 e 9 + 2 parecem ser iguais para o computador.

.../

O maior inteiro que pode ser guardado com precisão completa é um abaixo de 32 2's multiplicados (ou seja 4 294 967 295).
A cadeia "" sem qualquer caracter é chamada a cadeia vazia ou nula. Lembre-se que os espaços contam e que uma cadeia vazia não é o mesmo que uma que só contenha espaços.
Experimente

PRINT "Já acabou de ler o **"Finnegans Wake" ?** "

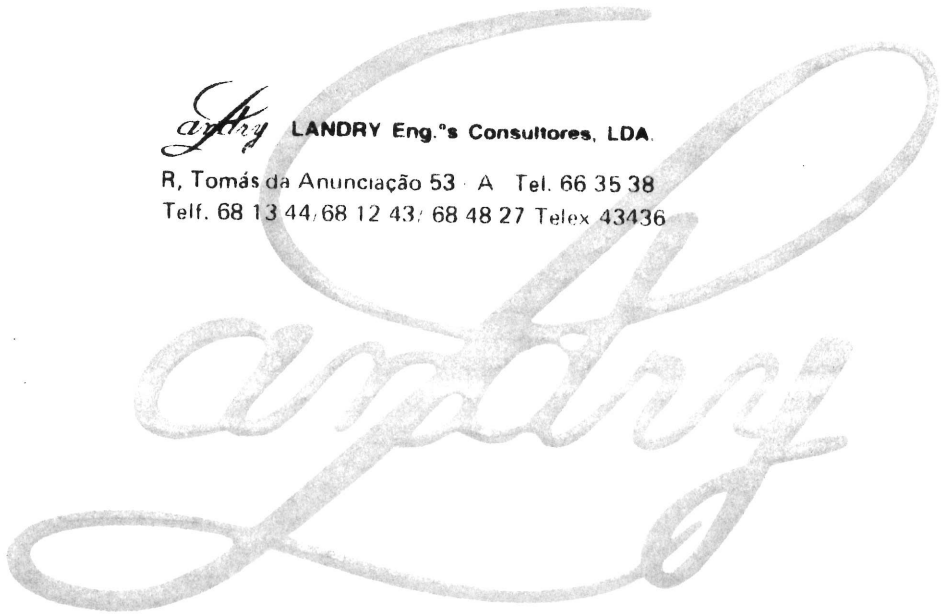
Quando carregar em **ENTER**, obterá um ponto de interrogação a piscar que indicará que há um erro algures na linha. Quando o computador encontrar as aspas no princípio de "Finnegans Wake", imagina que estas marcam o fim da caeia "Já acabou de ler o", e depois não consegue perceber o que é que 'Finnegans Wake' quer dizer.
Há um modo especial para superar isto: quando se quer escrever numa cadeia umas aspas, tem de se escrever a dobrar, assim:

PRINT "Já acabou de ler""**Finnegans Wake**""?"

Como pode ver pelo que aparece no écran, cada sinal de aspas a dobrar só lá aparece uma vez; tem de se introduzir a dobrar para que o computador as reconheça.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436



8

CAPÍTULO 8

Cadeia de Letras

Sumário

Corte, utilização de **TO**. Repare que esta notação não é uniforme em BASIC.

Dada uma cadeia, uma sub-cadeia consistirá num conjunto de caracteres consecutivos retirados da primeira sequência. Assim o literal "cadeia" é uma sub-cadeia de "cadeia maior", mas "cadeia a" e "ma ca" não são.
Há uma notação que chamaremos de corte (slicing) para descrever sub-cadeias, e isto pode ser aplicado a expressões literais arbitrárias. A forma geral é

Expressão literal (começo **TO** fim)

de modo que, por exemplo,

```
"abcdef"(2 TO 5)="bcde"
```

se se omitir o começo então presume-se que é 1; se se omitir o fim então assume-se que é o comprimento total da cadeia. Assim

```
"abcdef"( TO 5)="abcdef"(1 TO 5)="abcde"
"abcdef"(2 TO )="abcdef"(2 TO 6)="bcdef"
"abcdef"( TO )="abcdef"(1 TO 6)="abcdef"
```

(também se pode escrever esta última como `"abcdef"()` , embora não valha a pena).
Uma forma ligeiramente diferente omite o **TO** e só tem um número:

```
"abcdef"(3)="abcdef"(3 TO 3)="c"
```

Ainda que normalmente tanto o princípio como o fim se devam referir a zonas existentes do literal, esta regra é superada por outra: se o começo é maior que o fim, então o resultado é uma cadeia vazia. Assim

```
"abcdef"(5 TO 7)
```

.../

dá o erro **3 subscript wrong** (índice errado) porque a cadeia só tem 6 caracteres e 7 é demais, mas

```
"abcdef"(8 TO 7)=""
```

e

```
"abcdef"(1 TO 0)=""
```

Tanto o início como o fim não devem ser negativos, ou irá obter **B integer out of range** (número inteiro fora da gama permitida). O programa seguinte é simples para ilustrar algumas destas regras.

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Carregue em **NEW** quando tiver corrido este programa e introduza o programa seguinte:

```
10 LET a$="FUI CAPAZ"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10),a$((10-n) TO 10)
40 NEXT n
50 STOP
```

Para variáveis literais, não só lhes podemos extrair pedaços, mas também atribui-los a elas. Por exemplo escreva

```
LET a$="Eu sou o ZX Spectrum"
```

e depois

```
LET a$(5 TO 8)="******"
```

e

```
PRINT a$
```

.../

Repare como, uma vez que a sub-cadeia **a$(5 TO 8)** só tem 4 caracteres de comprimento, sòmente as quatro primeiras cruzes foram usadas. Esta é uma característica da atribuição a sub-cadeias: a sub-cadeia tem de ter exactamente o mesmo comprimento antes e depois da atribuição. Para nos certificarmos que isto acontece a cadeia que lhe vai ser atribuída é cortada à direita se fôr muito comprida, ou cheia com espaços se fôr muito curta - a isto chama-se atribuição Procrusteana, nome que vem de um estalajadeiro chamado Procrustes que costumava certificar-se que os seus locatários cabiam na cama ou esticando-os numa tábua ou cortando-lhes os pés.
Se agora experimentarmos

```
LET a$()=      "Olá"
```

e

```
PRINT a$;"."
```

verá que o mesmo aconteceu outra vez (desta vez foram colocados espaços) porque **a$()** conta como uma sub-cadeia.

```
LET a$=     "Olá"
```

fará o efeito normal.
Expressões literais complicadas necessitarão de parentesis antes de poderem ser cortadas. Por exemplo:

```
"abc"+"def"(1 TO 2)="abcde"
("abc"+"def")(1 TO 2)="ab"
```

Exercício

1. Experimente escrever um programa que escreva o dia da semana usando o corte de cadeias Sugestão: faça uma cadeia com SegTerQuartQuintSexSabDom.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

9

CAPÍTULO 9

Funções

Sumário

DEF

LEN, STR$, VAL, SGN, ABS, INT, SQR

FN

Pense numa máquina de fazer salsichas. Coloca-se a carne em massa numa ponta, roda-se a manivela e na outra ponta sai uma salsicha. Um pedaço de porco dá uma salsicha de porco, um pedaço de peixe dá uma salsicha de peixe, e um pedaço de carne de vaca dá uma salsicha de carne de vaca.
As funções são praticamente indescerniveis das maquinas de fazer salsichas, mas há uma diferença: trabalham com números e letras em vez de carne. Fornece-se um valor (chamado o argumento), trabalha-se um pouco através de alguns cálculos, e eventualmente obtem-se outro valor, o resultado.

```
        Entra a carne -- Maquina de Salsichas -- sai salsicha
    Entra o argumento --        função        -- sai resultado
```

Argumentos diferentes dão resultados diferentes, e se o argumento fôr completamente inadequado para a função esta pára e dá uma mensagem de erro.
Tal como se pode ter máquinas diferentes para fazer produtos diferentes - uma para salsichas, outra para panos de louça, e uma terceira para filetes, e assim por diante, diferentes funções executam cálculos diferentes. Cada uma delas terá os seus valores para a distinguir das outras.
Usa-se uma função em expressões escrevendo o seu nome seguido pelo argumento, e quando a expressão é calculada o resultado da função será determinado.
Como exemplo, há uma função chamada **LEN**, que calcula o comprimento de uma cadeia. O seu argumento é uma cadeia cujo comprimento queiramos determinar, e o resultado é o comprimento, de modo que se escrevermos

```
PRINT LEN "Sinclair"
```

o computador responde 8, o número de letras que existe em "Sinclair". (Para obter a função **LEN** como a maior parte dos nomes de funções é preciso usar o modo de extensão: carregar em **CAPS SHIFT** e em **SYMBOL SHIFT** ao mesmo tempo

.../

para mudar o cursor de L de E , e depois carregar na tecla L). Se se misturarem funções e operações numa mesma expressão, então as funções terão de ser calculadas antes das operações.
De novo, no entanto, pode superar essa regra utilizando parentesis. Por exemplo, aqui estão duas expressões que diferem somente nos parentesis, e no entanto os calculos são executados numa ordem completamente diferente, num e noutro caso (ainda que, por acaso, o resultado final seja o mesmo).

```
                                    LEN  ("João"  +  "Manuel")
LEN "Ze"+ LEN "Manuel"              LEN  ("JoãoManuel")
     4 + LEN  "Manuel"              LEN  "JoãoManuel"
     4 + 6                               1Ø
       1Ø
```

Aqui vão mais algumas funções.

STR$ converte numeros em cadeias: o seu argumento é um número, e o resultado é uma cadeia tal como a que o número provocaria se aparecesse no ecran depois de uma instrução **PRINT**. Repare como o seu nome acaba com o sinal **$** para mostrar que o seu resultado é uma cadeia. Por exemplo, podia dizer-se

```
LET a$=STR$ 1e2
```

que teria exactamente o mesmo efeito que

```
LET a$="100"
```

ou poderia dizer-se

```
PRINT LEN STR$ 100.0000
```

e obter-se-ia a resposta 3, porque **STR$ 100.0000="100"**

VAL é como **STR$** ao contrário: converte cadeias em números. Por exemplo

```
VAL "3.5"=3.5
```

.../

De certa forma VAL é o inverso de STR$, porque se se tomar qualquer número, e se aplicar STR$ e depois VAL ao resultado, volta-se a ter o número com que se começou.
Se se tomar uma cadeia, depois aplicarmos VAL e em seguida lhe aplicarmos STR$, nem sempre se obtém a cadeia original.
VAL é uma função muito poderosa, porque a cadeia que lhe serve de argumento não se restringe a ter a aparência de um número vulgar - pode ser uma expressão numérica. Assim, por exemplo

```
VAL "2*3"= 6
```

ou mesmo

```
VAL ("2"+"*3") = 6
```

Há dois processos em jogo aqui. No primeiro, o argumento de VAL é calculado como uma cadeia: a expressão literal "2"+"*3" é calculada por forma a dar "2*3". Depois a cadeia vê-lhe ser retiradas as aspas, e o que sobra é calculado como um número: assim 2*3 é calculado para dar o número 6.
Isto pode ser muito confuso se não se tiver cuidado; por exemplo

```
PRINT VAL "VAL""VAL""""2""""""""
```

(Lembre-se de que dentro de uma cadeia as aspas têm de ser escritas a dobrar. Se se entrar em maiores profundezas de cadeias, então verá que as aspas terão de ser quadruplicadas ou mesmo multiplicadas por oito).
Há outra função, muito semelhante a VAL, ainda que provavelmente menos útil, chamada VAL$.O seu argumento ainda é uma cadeia, mas o resultado também é uma cadeia. Para mostrar como ela funciona lembremos que VAL percorre dois passos: primeiro o argumento é calculado enquanto cadeia, depois retiram-se-lhe as aspas, e o que fica é calculado como um número. Com VAL$, o primeiro passo é o mesmo, mas depois de se lhe ter retirado as aspas no segundo passo, o que sobra é ainda encarado como uma cadeia diferente. Assim

```
VAL$ """"Ponche de frutas"""" = "Ponche de Frutas"
```

(Repare novamente como abundam as aspas). Escreva

```
LET a$="99"
```

.../

e mande escrever todos os valores seguintes: **VAL a$**, **VAL "a$"**, **VAL """a$"""**, **VAL$ a$** ,**VAL$ "a$"** , **VAL$ """a$"""**. Algumas destas fornecerão um resultado e outras não; tente explicar porquê em todos os casos. (Mantenha um pensamento claro).

SGN é a função sinal (por vezes também chamada de signum). É a primeira função que vemos que não tem nada a ver com cadeias de letras, porque tanto o argumento como o resultado são números. O resultado é +1 se o argumento fôr positivo, Ø se o argumento fôr zero e -1 se o argumento fôr negativo.

ABS é outra função cujo argumento e resultado são números. Converte o argumento num número positivo (que é o resultado),esquecendo-se do sinal, de modo que, por exemplo

ABS −3.2 = ABS 3.2 = 3.2

INT significa 'parte inteira' - um inteiro pode ser negativo. Esta função converte um número fraccionário num número inteiro desfazendo-se da parte fracionária, de modo que, por exemplo

INT 3.9 = 3

Cuidado quando o aplicar a números negativos, porque o arredondamento é sempre para baixo, como se vê

INT −3.9 = −4

SQR calcula a raiz quadrada de um número - o resultado é um número que, multiplicado por si mesmo, dá o argumento. Por exemplo

SQR 4 = **2** porque $2 * 2 = 4$

SQR 0.25 = **0.5** porque $Ø.5 * Ø.5 = Ø.25$

SQR 2 = **1.4142136**36 (aproximadamente)

porque $1.4142136 * 1.4142136 = 2.ØØØØØØ1$

Se se multiplicar qualquer número (mesmo um número negativo) por ele próprio a resposta é sempre positiva. Isto significa que os números negativos não têm raiz quadrada, de modo que se aplicarmos **SQR** a um número negativo teremos uma mensagem que diz An invalid argument (argumento inválido).

.../

Também se podem definir funções à vontade. Nomes possíveis para essas funções são **FN** seguido por uma letra (se o resultado fôr um número) ou **FN** seguido por uma letra seguida por **$** (se o resultado fôr uma cadeia de letras). Estas funções são muito mais rigorosas a respeito dos parentesis; o argumento tem sempre de vir entre parentesis.
Define-se uma função colocando uma instrução **DEF** algures no programa. Por exemplo, aqui está uma definição da função **FN s** cujo resultado é o quadrado do argumento:

```
10 DEF FN s(x)=x*x: REM          o quadrado de x
```

A instrução **DEF** obtem-se no modo de extensão, usando **SYMBOL SHIFT** e **1**. Quando se escreve isto o computador dá-nos o **FN** automaticamente, porque uma instrução **DEF** tem sempre de ter o **DEF** seguido por um **FN**. Depois disso o **s** completa o nome **FN s** da função.
O **x** entre parentesis é o nome pelo qual nos desejamos referir ao argumento da função. Pode-se usar uma única letra sòmente (ou, se o argumento fôr uma cadeia, uma única letra seguida por **$**).
Depois do sinal de = vem a verdadeira definição da função. Esta definição pode ser uma expressão qualquer, e pode-se referir ao argumento desde que use o mesmo nome que já lhe foi dado (**x** neste caso), como se fosse uma variável vulgar.
Quando se acabou de introduzir esta linha, já podemos chamar a função como se fosse uma das funções próprias do computador, escrevendo o seu nome, **FN s**, seguido do argumento. Lembre-se de que quando a função foi definida pelo utilizador o argumento tem sempre de vir entre parentesis. Experimente algumas vezes:

```
PRINT FN s(2)
PRINT FN s(3+4)
PRINT 1+INT FN s (LEN "galinha"/2+3)
```

Assim que tiver colocado a instrução **DEF** correspondente no programa, pode usar as suas funções em expressões de uma forma tão livre como as do próprio computador.
Nota: em alguns dialectos de BASIC também é obrigatório ter o argumento de uma função do próprio computador entre parentesis. Este não é o caso do BASIC do ZX Spectrum.

.../

INT arredonda sempre para baixo. Para arredondar para o inteiro mais próximo basta primeiro adicionar Ø.5 - pode escrever a sua própria função para fazer isto.

```
20 DEF FN r(x)=INT (x+0.5): REM
```

dá **x** arredondado ao inteiro mais próximo.

Obterá então, por exemplo

```
FN r(2.9) = 3
FN r(-2.9) = -3
```

```
FN r(2.4) = 2
FN r(-2.4) = -2
```

Compare estes valores com os que obtem quando usa directamente a função **INT** em vez de **FN r**. Escreva o seguinte e depois execute:

```
10 LET x=0: LET y=0: LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3),FN q()
```

Há muitos pontos subtis neste programa.

Primeiro, uma função não está só restrita a um argumento: pode ter mais, ou até nenhum - mas tem de se ter sempre os parentesis.

Segundo, não importa a localização no programa da instrução **DEF**. Depois do computador ter executado a linha 1Ø, passa pura e simplesmente por cima das linhas 2Ø e 3Ø para chegar à 4Ø. No entanto as definições têm de estar algures no programa. Não podem ser dadas num comando directo.

Terceiro, **x** e **y** são tanto o nome das variáveis no programa no seu todo como o nome dos argumentos para a função **FN p**. **FN p** esquece-se temporariamente das variáveis chamadas **x** e **y**, mas uma vez que não tem nenhum argumento chamado **a**, ainda se lembra da variável **a**. Assim, quando **FN p(2,3)** está a ser calculado, **a** tem o valor 1Ø porque é uma variável, **x** toma o valor 2 porque é o primeiro argumento e **y** o valor 3 porque é o segundo argumento. O resultado é então, 1Ø + 2 * 3 = 16. Quando a função **FN q()** está a ser calculada, por outro lado, não há argumentos, de modo que **a**, **x** e **y** ainda se referem a variáveis e têm respectivamente os valores 1Ø, Ø e Ø. A resposta, neste caso é 1Ø+Ø*Ø= =1Ø.

.../

Agora modifique a linha 2Ø para

```
20 DEF FN p(x,y)=FN q()
```

Desta vez **FN p(2,3)** terá o valor 1Ø porque **FN q** ainda irá buscar os valores das variáveis x e y em vez de usar os argumentos da função **FN p**.
Alguns dialectos de BASIC (não do BASIC do ZX Spectrum) têm funções chamadas LEFT$, RIGHT$, MID$ e TL$.
LEFT$ (a$,n) dá a sub cadeia de a$ que consiste nos primeiros n caracteres.
RIGHT$ (a$,n) dá a sub-cadeia de a$ que consiste nos caracteres do n-ésimo para a frente.
MID$ (a$,n1,n2) dá uma sub-cadeia de a$ que consiste em n2 caracteres começando no n1-ésimo.
TL$ (a$) dá a sub-cadeia de a$ que consiste em todos os seus caracteres excepto o primeiro.
Pode-se escrever funções definidas pelo utilizador para fazer o mesmo: por exemplo

```
10 DEF FN t$(a$)=a$(2 TO ): REM TL$
20 DEF FN l$(a$,n)=a$( TO n): REM LEFT$
```

Veja se estas funções funcionam com cadeias de comprimento Ø ou 1.
Note que a nossa função **FN l$** tem dois argumentos, um e, um número e o outro uma cadeia. Uma função pode ter até 26 argumentos numéricos (porquê 26?) e ao mesmo tempo até 26 argumentos alfanuméricos (de cadeia).

Exercício

Use a função **FN s(x)=x*x** para testar **SQR**: deve descobrir que

```
FN s(SQR x)=x
```

se substituir x por qualquer número positivo e

```
SQR FN s(x)=ABS x
```

quer o x seja positivo ou negativo (Porquê o **ABS**?).

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

10

CAPITULO 10

Funções Matemáticas

Sumário

↑
PI, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

Este capítulo trata da matemática que pode ser efectuada no ZX Spectrum. Provavelmente nunca terá de a utilizar, por isso se achar que é difícil demais não tenha problemas em saltar por cima deste capítulo. Trata da operação ↑ (levantar a uma potencia), as funções **EXP** e **LN** e as funções trignometricas **SIN, COS, TAN** e sua inversas **ASN, ACS**, e **ATN.**

↑ e **EXP**

Pode-se elevar um número a uma potência de outro - isto significa "multiplicar o primeiro número por ele próprio tantas vezes quanto o segundo número'. Isto normalmente é indicado escrevendo o segundo número exactamente acima e à direita do primeiro; mas obviamente isto seria difícil num computador, por isso usamos o símbolo ↑. Por exemplo, algumas potências de dois são

2↑1=2
2↑2=2*2=4 (2 ao quadrado, normalmente escrito 2^2)
2↑3=2*2*2=8 (2 ao cubo, normalmente escrito 2^3)
2↑4=2*2*2*2=16 (2 à quarta potência, normalmente 2^4)

Assim, no seu nível mais elementar 'a↑b'significa "a multiplicado por ele próprio b vezes', mas obviamente isto só faz sentido se o b fôr um número inteiro e positivo. Para encontrar uma definição que faça sentido para outros valores de b vamos considerar a regra

a↑(b+c)=a↑b*a↑c

(Repare que damos a ↑ uma prioridade mais elevada do que a * e /, de modo que quando há várias operações numa expressão, as 's são calculadas primeiro e depois as *'s e /'s. Não deve ser preciso convencê-lo de que isto está certo se tanto b como c são números inteiros e positivos; mas se decidirmos que queremos que isso funcione mesmo quando não são, então teremos que aceitar o seguinte:

.../

a ↑ Ø = 1
a ↑ (-b) = 1/a ↑ b
a ↑ (1/b) = a raiz b de a, ou seja, o número que tem de ser multiplicado b vezes por si mesmo para dar a
e
a ↑ (b * c) = (a ↑ b) ↑ c

Se nunca tinha visto isto antes, não tente lembrar-se imediatamente de tudo: lembre-se só que

a ↑ (-1) = 1/a
e
a ↑ (1/2) = **SQR** a

e talvez quando já estiver familiarizado com estas coisas o resto comece a fazer sentido.
Experimente tudo isto com o seguinte programa:

```
10 INPUT a,b,c
20 PRINT a ↑ (b+c)=a ↑ b*a ↑ c
30 GO TO 10
```

Claro que se a regra que demos anteriormente fôr verdadeira em todas as vezes os dois números que o computador imprimir serão iguais. (Nota - devido ao modo como o computador calcula ↑ o número do esquerda - neste caso **a** - - nunca pode ser negativo).
Um exemplo típico do que se pode fazer com esta função é o juro composto. Supomos que queremos investir algum dinheiro numa sociedade de construção e que eles pagam 15% de juros por ano. Então ao fim de um ano não terá sòmente os 1ØØ% que de qualquer forma já tinha, mas também os 15% de juros que a sociedade de construção lhe deu, fazendo ao todo 115% do que tinha originalmente. Explicando de forma diferente multiplicou a sua quantidade de dinheiro por 1.15, e isto é verdade para qualquer quantia que lá tivesse colocado. Depois de outro ano, o mesmo voltará a acontecer, de modo que terá então 1.15*1.15= =1.15 ↑ 2=1.3225 vezes a quantia original que lá tinha colocado. Em geral depois de **y** anos terá **1.15 ↑ y** vezes o que lá tinha colocado inicialmente.
Se experimentar esta instrução

.../

```
FOR y=0 TO 100: PRINT n,10*1.15 ↑ y: NEXT y
```

e verá que mesmo começando sòmente com 1Ø escudos, subirá rapidamente e mais que se vai tornando uma subida cada vez mais rapida (ainda que possa verificar que é capaz de não superar a inflação).
Esta espécie de comportamento, onde após um intervalo fixo de tempo uma dada quantidade se multiplica a si própria por uma proporção fixa, é chamado de crescimento exponencial,e é calculado levantando um número fixo a uma potência do tempo.
Suponhamos que faz isto:

```
1Ø DEF FN a(x)=a ↑ x
```

Aqui o a é mais ou menos fixo, por instruções **LET**: o seu valor irá corresponder à taxa de juro, que se modifica só muito esporàdicamente.
Há um certo valor de a que faz com que a função **FN a** apareça especialmente bela para o olhar experimentado de um matemático: e a esse valor chama-se e.
O ZX Spectrum tem uma função chamada **EXP** definida por

```
EXP x = e ↑ x
```

Infelizmente o e em si não é um número particularmente bonito: é um número fracionário de décima não periódica. Pode ver quais são as primeiras casas decimais fazendo

```
PRINT EXP 1
```

porque **EXP 1 = e ↑ 1 = e**. Claro que isto é sòmente uma aproximação. Nunca se poderá escrever exactamente o **e**.

LN

A função inversa de uma exponencial é a função logarítmica: o logaritmo (numa base a) de um número x é a potência a que é preciso levantar a para obter de novo o valor de x, e escreve-se $\log_a x$. Assim, por definição $a \uparrow \log_a x = x$; e também é verdade que $\log_a(a \uparrow x) = x$.
Talvez já saiba como pode usar logaritmos na base 1Ø para fazer multiplicações; esses logaritmos chamam-se de comuns. O ZX Spectrum tem uma função **LN**

.../

que calcula os logaritmos na base e; a estes chamam-se logaritmos naturais. Para calcular logaritmos em qualquer outra base, tem de se dividir o loga - ritmo nacural do número pelo logaritmo natural da nova base:

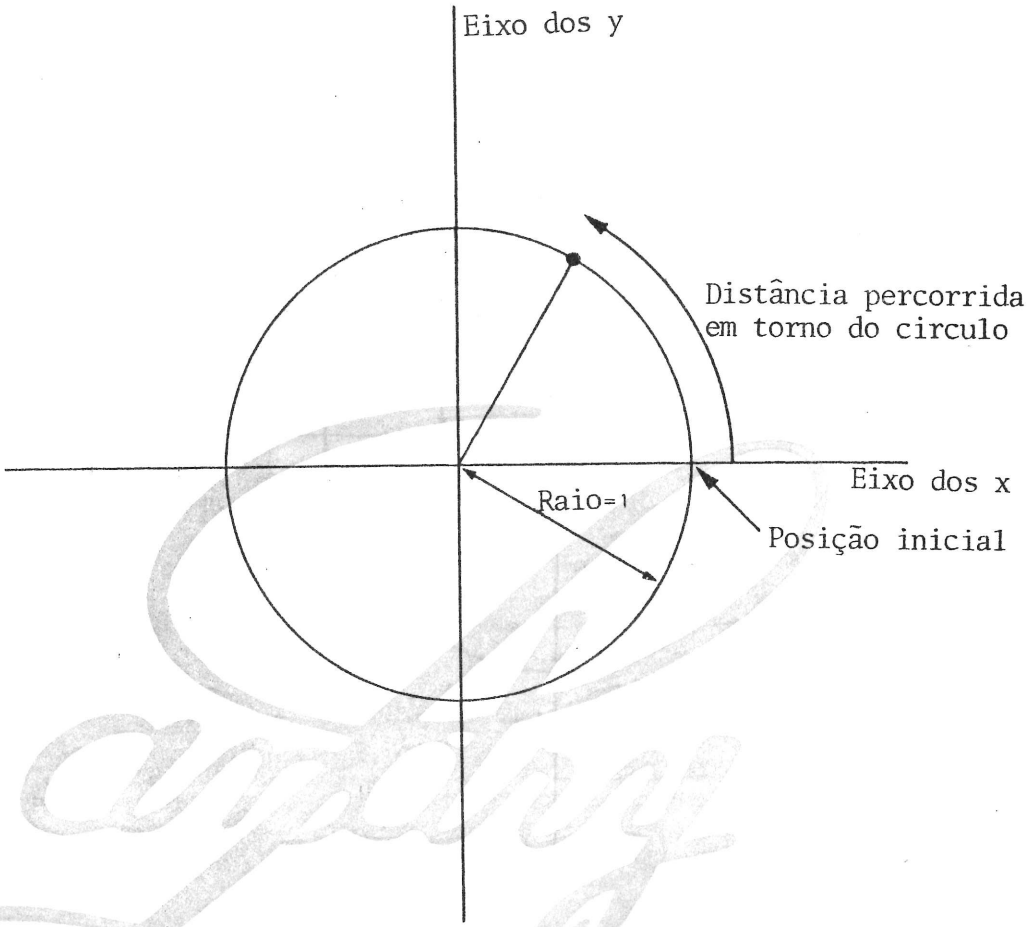
$$\log_a x = \text{LN } x / \text{LN } a$$

PI

Dado um círculo qualquer, pode-se sempre achar o seu perímetro (a distância da sua envolvente; muitas vezes chamada de circunferência) multiplicando o seu diametro (largura) por um número chamado π . (π é um p Grego, e é usado porque representa perímetro. O seu nome é **pi**).
Tal como e o π é uma dízima infinita não periódica; começa com 3.141592653589... A palavra **PI** no Spectrum (modo de extensão, depois M) representa este número - experimente **PRINT PI**.

SIN, COS e **TAN; ASN, ACS** e **ATN**

As funções trignométricas medem o que se passa quando um ponto se move em torno de um círculo. Aqui está um círculo de raio 1 (1 o quê? não importa desde que mantenhamos a mesma unidade durante todos os cálculos. Não há nada que o impeça de inventar uma unidade própria para cada círculo que lhe interessar estudar) e um ponto que se move em torno dele. O ponto começou na posição das 3 horas, depois moveu-se no sentido anti-horário.

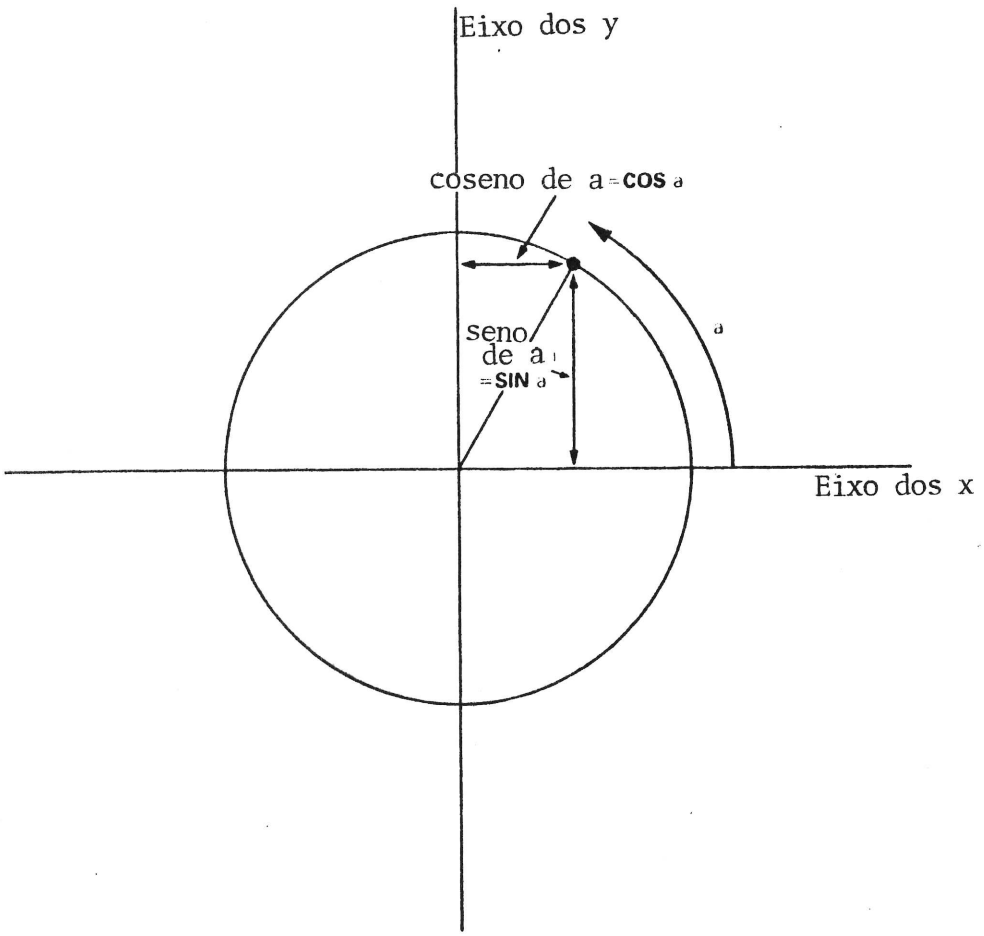


Também desenhámos duas linhas chamadas eixos passando pelo centro da circunferência. O que passa pelas 3 e pelas 9 horas chama-se o eixo dos x, e aquele que passa pelas 6 horas e pelas 12 horas chama-se o eixo dos y.

Para especificarmos onde o ponto de encontra dizemos quanto é que ele andou a partir da sua posição inicial às 3 horas. chamemos a a distância. Sabemos que o perímetro do círculo é 2 π (porque o seu raio é unitário e assim o diâmetro é 2); assim, quando ele tiver dado um quarto de volta em torno do círculo teremos que a = π /2; quando acabar a meia volta, a = π ; e quando a volta estiver completa teremos a = 2 π .

Dada a distância curva em torno da circunferência, a, duas outras distâncias que gostaríamos de saber são a distância a que está o ponto do eixo dos y, e a distância a que está do eixo dos x. A estas distâncias chamam-se respectivamente o coseno e o seno de a. A função **COS**, bem como a função **SIN** do computador calculam estas duas quantidades.

.../



Repare que se o ponto passar para a esquerda do eixo dos y, então o coseno torna-se negativo; e se o ponto passar abaixo do eixo dos x, será o seno que se torna negativo.

Outra propriedade é que assim que a passar por 2 π , o ponto volta ao ponto de partida e o seno e o coseno voltam a tomar os valores por que já ha - viam passado:

```
SIN (a+2*PI) = SIN a
COS (a+2*PI) = COS a
```

A tangente de a é definida como sendo o seno dividido pelo coseno; a função correspondente no computador chama-se **TAN.**

Algumas vezes temos de trabalhar com estas funções ao contrário, encontrando o valor de a que dá o seno, o coseno ou a tangente. As funções que fazem isto são chamadas arcoseno (**ASN** no computador), arccoseno (**ACS**) e arctangente (**ATN**).

.../

No diagrama do ponto movendo-se em torno do círculo, olhe para o raio que liga o centro ao ponto. Deve ver que a distância a que chamámos a, a distância que o ponto percorreu em torno do circulo é um modo de medir o ângulo que o raio varreu a partir do eixo dos x. Quando a = π /2, o ângulo é de 9Ø graus; quando a = π o ângulo é de 18Ø graus; e assim por diante até que a = 2 π e o ângulo seja de 36Ø graus. Já agora podemo-nos esquecer dos graus e medir o ângulo sòmente em termos de a: dizemos que estamos a medir o ângulo em radianos . Assim π /2 radianos = 9Ø graus e assim por diante. Também tem de se lembrar que no ZX Spectrum as funções **SIN**, **COS** e assim por diante só podem usar radianos e nunca graus. Para converter graus em radianos divide-se por 18Ø e multiplica-se por π , para converter radianos em graus divide-se por π e multiplica-se por 18Ø.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 · A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

11

CAPÍTULO 11

Números Aleatórios

Sumário

RANDOMIZE
RND

Este capítulo trata da função **RND** e da palavra chava **RANDOMIZE.** Ambas são usadas relativamente a números aleatórios, de modo que deve ter cuidado para não as confundir. São colocadas com a mesma tecla (**T**); **RANDOMIZE** teve de ser abreviada para um **RAND.**
De certa forma **RND** é como uma função: faz os cálculos e produz um resultado. É invulgar sòmente pelo facto de que não tem argumentos.
Cada vez que se usar, o resultado é um novo número entre Ø e 1. (por vezes pode tomar o valor Ø mas nunca 1).
Experimente

```
10 PRINT RND
20 GO TO 10
```

Para ver como é que a resposta varia. Consegue detectar alguma regularidade? Não deve ser; 'aleatório' significa que não existe qualquer regularidade.
Na verdade a função **RND** não é verdadeiramente aleatória, porque segue uma sequência fixa de 65536 números. No entanto estão tão cuidadosamente misturados que não existe pelo menos nenhum padrão óbvio e dizemos que **RND** é pseudo-aleatória.
RND dá números aleatórios entre Ø e 1, mas pode-se facilmente obter números aleatórios noutras gamas. Por exemplo **5*RND** produz números entre Ø e 5, e **1.3+0.7*RND** está entre 1.3 e 2. Para obter números inteiros use **INT** (lembrando-se que a função **INT** arredonda sempre para baixo) tal como em **1+INT(RND*6)** que nós vamos usar num programa para simular o lançamento de um dado. **RND*6** está na gama entre Ø e 6, mas como nunca chega a 6, a expressão **INT(RND*6)** só toma os valores Ø, 1, 2, 3, 4 e 5.
Aqui está o programa:

.../

```
10 REM programa de lançamento de dados
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+INT (RND*6);" ";
50 NEXT n
60 INPUT a$: GO TO 20
```

Carregue em **ENTER** cada vez que quiser lançar o dado.
A instrução **RANDOMIZE** é usada para fazer com que o **RND** comece num sítio definido na sequência de números, como se pode ver neste programa:

```
10 RANDOMIZE 1 !
20 FOR n=1 TO 5: PRINT RND ,: NEXT n
30 PRINT : GO TO 10
```

Depois de cada execução de **RANDOMIZE 1** a sequência **RND** recomeça em Ø.ØØ22735596. Pode-se usar outros números entre 1 e 65535 na instrução **RANDOMIZE** para começar a sequência **RND** em pontos diferentes.
Se tiver um programa com **RND** e não conseguir encontrar os erros que esse programa possa ter, talvez ajude se usar **RANDOMIZE** para que o programa se comporte sempre da mesmo forma.
RANDOMIZE sòzinho (e **RANDOMIZE Ø** faz a mesma coisa) é diferente porque, na verdade, torna RND aleatorio - pode ver isso no programa seguinte.

```
10 RANDOMIZE
20 PRINT RND : GO TO 10
```

A sequência que se obtêm aqui não é muito aleatória, porque a instrução **RANDOMIZE** usa o tempo desde que o computador foi ligado. Uma vez que este sobe mais ou menos a mesma coisa sempre que **RANDOMIZE** é executado, a proxima instrução **RND** faz mais ou menos o mesmo que a anterior. Obter-se-ia uma variação mais aleatória se se substituisse **GO TO 10** por **GO TO 20** .
Nota: a maior parte dos dialectos de BASIC utiliza **RND** e **RANDOMIZE** para produzir números aleatórios, mas nem todos da mesma forma.
Aqui está um programa para atirar moedas e contar o número de caras e coroas.

```
10 LET caras = Ø: LET    coroas = Ø
```

.../

```
20 LET  MOEDA =INT (RND*2)
30 IF   moeda =0 THEN LET     caras = caras +1
40 IF   moeda =1 THEN LET     coroas = coroas +1
50 PRINT  caras; ", "; coroas,
60 IF  coroas <>0 THEN PRINT      caras/coroas;
70 PRINTT : GO TO 20
```

A razão entre as caras e as coroas deve tornar-se aproximadamente 1 se se continuar durante muito tempo, porque a longo prazo espera-se que saiam o mesmo número de caras que de coroas.

Exercícios

1. Experimente a regra:

   Suponha que escolhe um número entre 1 e 872 e escreve

   RANDOMIZE o seu número

   Então o próximo valor de **RND** será

   (75 * (o seu número + 1) - 1)/65536

2. (Só para matemáticos)

   Seja p um número primo (grande), e seja a uma raiz primitiva modulo p. Então se b_i é o resíduo de a^i modulo p ($1 \leq b_i \leq p-1$), a sequência

$$\frac{b_i - 1}{p-1}$$

é uma sequência cíclica de p-1 números distintos na gama entre Ø e 1 (excluindo 1). Escolhendo a cuidadosamente podemos fazer com que esta sequência pareça bastante aleatória.
65537 é um número primo de FERMAT, $2^{16} + 1$. Dado que o grupo multiplicativo de resíduos não nulos modulo 65537 tem uma potência de 2 como ordem, um resíduo é uma raiz primitiva se e só, não fôr um resíduo quadrático. Usar a lei de Gauss para a reciprocidade quadrática, para mostrar que 75 é uma raiz primitiva modulo 65537.

.../

O ZX Spectrum usa p=65537 e a=75, e guarda alguns b_i-1 em memória. **RND** substitui b_i-1 em memória por b_{i+1}-1 e dá o resultado (b_{i+1}-1) (p-1). A instrução **RANDOMIZE** n (com $1 \leq n \leq 65535$) torna b_i igual a n+1.

RND é aproximadamente uma distribuição uniforme na gama Ø a 1.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

12

CAPÍTULO 12

Matrizes

Sumário

Matrizes (o modo como o ZX Spectrum manipula matrizes de cadeias de letras é algo invulgar).

DIM...

Suponhamos que temos uma lista de números, por exemplo as notas de dez pessoas de uma turma. Para os guardar no computador seria necessária uma variável para cada pessoa, mas isso seria defícil de manejar. Podia chamar às variáveis NOTA 1, NOTA 2 e assim por diante até NOTA 1Ø, mas o programa para introduzir estes números seria bastante comprido e maçador de introduzir. Como seria muito mais fácil se se pudesse escrever assim:

```
5 REM este programa não funciona

10 FOR n=1 TO 10
20 READ  NOTA n
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

pois é, mas não se pode.
No entanto há um mecanismo pelo qual pode aplicar esta ideia, e usa matrizes. Uma matriz é um conjunto de variáveis, os seus elementos, todos com o mesmo nome, e distintos uns dos outros sòmente por um número (o índice) escrito entre parentesis depois do nome. No nosso exemplo o nome pode ser b (tal como em variáveis de controle em ciclos **FOR - NEXT**, o nome de uma matriz tem de ser uma única letra), e as dez variáveis seriam então b(1), b(2), etc até b(1Ø).
Os elementos de uma matriz dizem-se variáveis indexadas, em oposição às variáveis simples que já são nossas conhecidas.
Antes de podermos usar uma matriz, tem de se reservar espaço para ela dentro do computador, e faz-se isso utilizando a instrução **DIM** (de dimensionamento).

```
DIM b(10)
```

.../

cria uma matriz chamada b com uma dimensão 1Ø (isto é há 1Ø variáveis indexadas b(1),...b(1Ø) e inicializa os 1Ø valores a Ø. Também apaga qualquer matriz chamada b que existisse anteriormente (mas não uma variável simples. Uma matriz e uma variável numérica simples podem coexistir com o mesmo nome e não há confusão entre elas porque a variável da matriz tem sempre um índice).
O índice pode ser uma expressão numérica arbitrária, de modo que podemos escrever

```
10 FOR n=1 TO 10
20 READ b(n)
30 NEXT n
40 DATA 10,2.5,19,16,3,11,1,0,6
```

Também se podem arranjar matrizes com mais que uma dimensão. Numa matriz bi-dimensional são necessários dois números para especificar cada um dos elementos - tal como o número de linha e de coluna para especificar a posição de um caracter no écran de televisão - por isso tem a forma de uma tabela. Alternativamente se se imaginar o número de linha e de coluna (duas dimensões) como referência e uma página impressa, podemos acrescentar outra dimensão para o número de página. Claro que estamos a falar de matrizes numéricas; de modo que os elementos não seriam caracteres impressos num livro mas sim números. Pense nos elementos de uma matriz tri-dimensional v especificados por v (número de página, número de linha, número de coluna).
Por exemplo, para organizar uma tabela bi-dimensional c com as dimensões 3 e 6, usamos a instrução **DIM**.

```
DIM c(3,6)
```

Isto então dá-nos 3 * 6 = 18 variáveis indexadas

c(1,1),c(1,2),...,c(1,6)
c(2,1),c(2,2),...,c(2,6)
c(3,1),c(3,2),...,c(3,6)

O mesmo princípio funciona para qualquer número de dimensões.
Ainda que se possa ter um número e uma matriz com o mesmo nome, não se pode ter duas matrizes com o mesmo nome, ainda que tenham um número de dimensões diferentes.

.../

Também há matrizes de cadeias. As cadeias de uma matriz diferem das cadeias simples pelo facto de terem comprimento fixo e a atribuição que lhes é feita é sempre Procusteana - cortada à medida ou cheia com espaços. Outro modo de pensar nelas é como matrizes (com uma dimensão extra) de caracteres simples. Onome de uma matriz de cadeia é uma letra única seguida por **$**, e uma matriz de cadeia simples não podem partilhar o mesmo nome, ao contrário do que acontece com os números.

Suponha então que quer uma matriz **a$** com cinco cadeias. Tem de se decidir quanto ao tamanho que estas cadeias podem ter - por exemplo, vamos supôr que 1Ø caracteres de comprimento é suficiente. Então diz.

```
DIM a$(5,10)
```

(escreva isto)

Isto cria uma matriz com 5 * 1Ø caracteres, mas também pode pensar em cada fila como sendo uma cadeia:

```
a$(1) = a$(1,1) a$(1,2)···a$(1,1Ø)
a$(2) = a$(2,1) a$(2,2)...a$(2,1Ø)
  .       .       .       .
  .       .       .       .
a$(5) = a$(5,1) a$(5,2)...a$(5,1Ø)
```

Se se der o mesmo número de índices (dois neste caso) quantas forem as dimensões na instrução **DIM** então obtém-se um único caracter; mas se se esquecer do último, então obtém uma cadeia de comprimento fixo. Assim, por exemplo, A$(2,7), é o sétimo caracter na cadeia A$(2); usando a notação de corte também se podia escrever isto como A$(2)(7). Agora escreva

```
LET a$(2)="1234567890"
```

e

```
PRINT a$(2),a$(2,7)
```

obtém-se

```
1234567890     7
```

Para o último índice (aquele que pode ser omitido) também se pode ter um cor-

.../

te, por exemplo

```
a$(2,4 TO 8)=a$(2)(4 TO 8)="45678"
```

Lembre-se:

Numa matriz de cadeia, todas as cadeias têm o mesmo comprimento fixo.
A instrução **DIM** tem um número extra (o último) para especificar este comprimento.
Quando se escreve uma variável indexada para uma matriz de cadeia, pode-se colocar um número, ou um corte, que corresponda ao número extra na instrução **DIM**.
Pode ter matrizes de cadeia sem dimensões. Introduza

```
DIM a$(10)
```

e verá que **a$** se comporta como uma variável de cadeia, excepto que ela tem um comprimento fixo de 1Ø, e a atribuição será sempre procrusteana.

Exercícios

1. Usar a instrução **READ** e **DATA** para organizar uma matriz **m$** de doze cadeias na qual o elemento **m$(n)** é o nome do **n**-ésimo mês.
(sugestão: a instrução **DIM** poderá ser **DIM m$(12,9)** .Experimente imprimir tudo o que está em **m$(n)**, usando um ciclo).
Escreva

```
PRINT "Estamos no mês de "; m$(5) ;" em que os rapazes se
divertem"
```

O que poderá fazer quanto aos espaços que aparecem?

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

13

CAPÍTULO 13

Condições

Sumário

AND, OR

NOT

Vimos no capítulo 3 como a instrução **IF** se escreve

IF condição **THEN** ...

As condições eram as relações (=, <=, <, >, <=, >= = e <>) que compararam dois números ou duas cadeias. Também se podem combinar várias datas destas, usando as operações lógicas **AND** (e), **OR** (ou) e **NOT** (não).
Uma relação **AND** (e) outra é verdade quando ambas as relações são verdade, de modo que se podia ter uma linha como

```
IF a$="sim" AND x<>0 THEN PRINT x
```

em que o x só é impresso se a$ = "sim" e x> Ø. O BASIC neste caso assemelha-se muito a uma linguagem corrente de modo que nem vale muito a pena explicar os detalhes. Tal como em português se pode juntar uma série de relações com e, e depois o conjunto só é verdade se cada uma das relações o fôr, aqui faz-se o mesmo usando **AND**.
Uma relação **OR** (ou) outra é verdade sempre que pelo menos uma das duas relações fôr verdade. (lembre-se que ainda é verdade se ambas as relações forem verdade; isto é algo que a linguagem corrente nem sempre admite).
A relação **NOT** (não) vira as coisas de pernas para o ar. Se tivermos **NOT** relação, isto só será verdade se a relação fôr falsa, e é falso se a relação fôr verdade!
As expressões lógicas podem-se fazer com **AND**, **OR** e **NOT** tal como as expressões numéricas podem ser feitas com números e +, - etc; até as pode colocar entre parentesis quando isso fôr necessário. Têm prioridades do mesmo modo que as operações +, -, *, / e ↑ : **OR** tem a mais baixa prioridade, e a seguir **AND** e depois **NOT**, depois as relações e por fim as operações usuais.

.../

O **NOT** na verdade é uma função, com um argumento e um resultado, mas a sua prioridade é muito mais baixa do que a das outras funções. Por isso o seu argumento não precisa de parentesis a não ser que contenha **AND** ou **OR** (ou ambos). **NOT (a=b)** tem o mesmo significado que **NOT a=b** e, é claro, o mesmo que **a<>b** .

<> é a negação de = no sentido em que só é verdade se = fôr falso. Por outras palavras

a<>b é o mesmo que **NOT a=b**

e também

NOT a<>b é o mesmo que **a=b**

Convença-se que > = e < = são respectivamente as negações de < e > ; por isso pode ver-se livre do **NOT** em frente de relações, alterando a relação.

Também

NOT (primeira expressão lógica **AND** segunda)

é o mesmo que

NOT (a primeira) **OR NOT** (a segunda)

e

NOT (primeira expressão lógica **OR** segunda)

é o mesmo que

NOT (a primeira) **AND NOT** (a segunda).

Usando estas relações pode-se ir fazendo os **NOT**'s passar pelos vários níveis da parentesis, até que eventualmente aplicando-os só a relações, já poderá ver-se livre deles. Logicamente falando o **NOT** não é necessário, ainda que por vezes se verifique que a sua utilização torna os programas mais claros.

A secção seguinte é algo complicada e pode ser omitida pelos mais fracos! Experimente

.../

PRINT 1=2,1<>2

podia esperar que isto desse um erro de sintaxe. De facto no que diz respeito ao computador não há nada que se chame um valor lógico: em vez disso ele usa números vulgares, sujeitos a algumas regras.

(i) =, <, >, <=, >=, e <> dão sempre resultados numéricos. 1 para verdade e Ø para falso. Assim a instrução **PRINT** escrita acima iria indicar Ø para '1=2' que é falso e 1 para '1 <> 2' que é verdade.

(ii) Em

IF condição **THEN**...

a condição na verdade pode ser qualquer expressão numérica. Se o seu valor fôr Ø, conta como falsa, e qualquer outro valor (incluindo o de 1 que daria uma verdadeira relação) conta como verdade. Assim a instrução **IF** significa exactamente o mesmo que

IF condição <>Ø **THEN**...

(iii) **AND**, **OR** e **NOT** também são operações com valores numéricos

x **AND** y tem o valor { x, se y fôr verdade (diferente de zero) ; Ø (falso) se y fôr falso (zero)

x **OR** y tem o valor { 1 (verdade) se y fôr verdade (diferente de zero) ; x se y fôr falso (zero)

NOT x tem o valor { Ø (falso) se x é verdade (diferente de zero) ; 1 (verdade) se x é falso (zero)

(repare que 'verdade' quer dizer 'diferente de zero' quando estamos a verificar um certo valor, mas significa '1' quando estamos a produzir um valor novo).

Volte a ler este capítulo à luz desta revelação, certificando-se que funciona tudo.

Nas expressões **x AND y**, **x OR y** e **NOT x**, tanto o **x** como o y normalmente toma-

.../

rão os valores Ø e 1 para falso e verdade. Escreva as dez combinações diferentes possíveis (quatro para **AND**, quatro para **OR**, e duas para **NOT**) verifique que elas fazem o que este capítulo lhe fez esperar que fizesse.
Experimente este programa:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>=b)+(b AND a<b)
40 GO TO 10
```

De cada vez imprime o maior dos dois números **a** e **b**.
Convença-se que consegue pensar em

x AND y

como sendo

x se y (de outra forma o resultado é Ø)

e de

x OR y

como

x a não ser que y (caso em que o resultado é 1)

Uma expressão usando **AND** ou **OR** como estas chama-se uma expressão condicional. Um exemplo usando **OR** podia ser

```
LET preço total = preço sem taxa * (1.15 OR v$ = "taxa zero")
```

Repare como o **AND** tem tendência a aparecer juntamente com adições (porque no caso de indefinição o valor é Ø), e o **OR** tem tendência a acompanhar multiplicações (porque o valor de indefinição é 1).
Também se podem fazer expressões condicionais com valores de cadeia, mas sòmente usando **AND**.

.../

x$ AND y tem o valor { x$ se y é diferente de zero
"" se y é zero

significando assim **x$** se **y** (em caso contrário uma cadeia vazia).
Experimente este programa, que introduz duas cadeias e as coloca por ordem alfabética.

```
10 INPUT "introduza as duas cadeias"'a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";("<" AND a$<b$)+("=" AND a$=b$);" ";b$
40 GO TO 10
```

Exercício

1. O BASIC por vezes pode trabalhar de formas diferentes da linguagem corrente, Considere, por exemplo, a frase 'se a não fôr igual a b ou c'. Como poderia escrever isto em BASIC? A resposta não é

```
IF A<>B OR C
```

nem

```
IF A<>B OR A<>C
```

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

14

CAPÍTULO 14

O Conjunto de Caracteres

Sumário

CODE, CHR$
POKE, PEEK
USR
BIN

As letras, dígitos, sinais de pontuação e assim por diante, que podem aparecer em cadeias, são chamados de caracteres, e em conjunto constituem o alfabeto, ou conjunto de caracteres, que o ZX Spectrum usa. A maior parte destes caracteres são símbolos simples, mas há mais alguns, chamados simbolos compostos que representam palavras completas, tais como **PRINT**, **STOP**, <> e assim por diante.

Há 256 caracteres, e cada um deles tem um código numérico entre 0 e 255. Há uma lista completa deles no Apêndice A. Para interconverter códigos e caracteres há duas funções, **CODE** (código) e **CHR$**.

A instrução **CODE** é aplicada a uma cadeia, e dá o código do primeiro caracter da cadeia (ou Ø se a cadeia fôr vazia).

CHR$ é aplicado a um número e dá um caracter cujo código é aquele número.

Este programa imprime o conjunto completo de caracteres

```
10 FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

No princípio pode ver-se um espaço, 15 símbolos e sinais de pontuação, os dez dígitos, mais sete símbolos, as letras maiúsculas, mais seis símbolos, as letras minúsculas e mais cinco símbolos. Estes caracteres (exceptuando £ e ©) são todos tirados de um conjunto de caracteres largamente usado e que é conhecido por ASCII (que significa American Standard Codes for Information Interchange); O ASCII também atribui códigos numéricos a estes caracteres, e são estes códigos que o ZX Spectrum usa.

O resto dos caracteres não faz parte do ASCII e são particulares para o ZX Spectrum. Os primeiros entre eles são um espaço e 15 padrões de sinais a preto e branco. A estes últimos chama-se símbolos gráficos e podem ser usados

.../

para desenhar figuras. Se carregar em **GRAPHICS** (**CAPS SHIFT** e **9**) então o cursor mudará para [G] . Agora as teclas para os dígitos de 1 a 8 escreverão os símbolos gráficos: só por si obtêm-se os símbolos que estão desenhados nas teclas; e com uma tecla de shift o mesmo símbolo mas invertido, isto é, o negro torna-se branco e vice-versa.
Independentemente das teclas de shift, o dígito 9 volta para o modo normal (L) e o dígito zero apaga (**DELETE**).
Aqui estão os dezasseis caracteres gráficos:

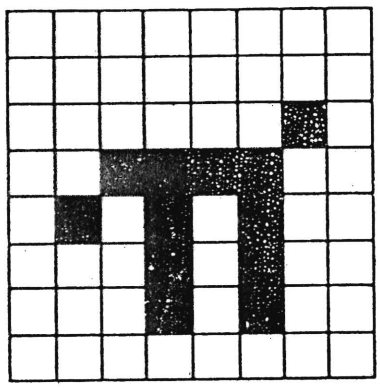
Símbolo	Código	Como se obtem	Símbolo	Código	Como se obtem
	128	[G] 8		143	[G] shift e 8
	129	[G] 1		142	[G] shift e 1
	130	[G] 2		141	[G] shift e 2
	131	[G] 3		140	[G] shift e 3
	132	[G] 4		139	[G] shift e 4
	133	[G] 5		138	[G] shift e 5
	134	[G] 6		137	[G] shift e 6
	135	[G] 7		136	[G] shift e 7

Depois dos símbolos gráficos verá aparecerem o que parece ser outra cópia do alfabeto de A a U. Estes são todos caracteres que você pode re-definir como quiser, ainda que quando a máquina fôr ligada eles formem um conjunto de letras - são chamadas caracteres gráficos definidos pelo utilizador. Pode introduzir estes caracteres pelo teclado entrando no modo gráfico e depois usando as teclas das letras de A a U.

.../

Para definir um novo caracter à sua vontade siga esta receita - que define um caracter para mostrar um π .

(i) veja qual o aspecto do caracter. Cada caracter tem uma matriz de 8 x 8 pontos, cada um deles podendo aparecer ou da cor do papel, ou da cor da tinta (ink) (ver o livro introdutório). Devia desenhar um diagrama parecido com este, com quadrados negro para a cor da tinta:



Deixámos uma margem de 1 quadrado em toda a volta porque as outras letras também têm uma (exceptuando as letras minúsculas com caudas, em que a cauda vai até ao fundo).

(ii) Decida qual dos caracteres gráficos definidos pelo utilizador vai mostrar o π - suponhamos que é o que corresponde ao P, de modo que, se carregar na tecla O no modo gráfico, lhe aparece um π .

(iii) Guarde o novo padrão. Cada gráfico definido pelo utilizador tem o seu padrão guardado em oito números, um por cada fila. Pode escrever cada um destes números como um BIN (binário) seguido por oito Ø's ou 1's - Ø para cor do papel (paper), 1 para cor da tinta (ink) - de modo que os oito números para o nosso caracter π são

BIN ØØØØØØØØ
BIN ØØØØØØØØ
BIN ØØØØØØ1Ø
BIN ØØ1111ØØ
BIN Ø1Ø1Ø1ØØ
BIN ØØØ1Ø1ØØ
BIN ØØØ1Ø1ØØ
BIN ØØØØØØØØ

.../

(Se souber alguma coisa sobre números binários, deve ajudar saber que a instrução BIN é usada para escrever um número em binário em vez de um décimal).
Estes oito números são guardados em memória, em oito locais, cada um deles com um endereço. O endereço do primeiro byte, ou primeiro grupo de oito dígitos, é USR"P" (P porque foi esse que escolhemos em (II), o do segundo é USR"P"+1 , e assim por diante até ao oitava, cujo endereço será USR"P"+7.
USR é a função que converte um argumento de cadeia no endereço do primeiro byte na memória que corresponde ao caracter gráfico definido pelo utilizador. O argumento de cadeia tem de ser um único caracter que pode ser ou o caracter gráfico definido pelo utilizador em si ou a letra correspondente (em letras maiúsculas ou minúsculas). Há outra utilidade para USR, quando o seu argumento é um número, mas mais tarde trataremos disso.
Ainda que não compreenda isso, o seguinte fará isso por si:

```
10 FOR n=0 TO 7
20 INPUT row: POKE USR "P"+n,row
30 NEXT n
```

O programa pára para entrada de dados oito vezes, para permitir a introdução dos oito números binários que foram dados acima - escreva-os por ordem, começando com a fila de cima.
A instrução POKE guarda o número directamente na memória, passando por cima do mecanismo normalmente usado pelo BASIC. O oposto do POKE é o PEEK, e isto permite-nos ver o que está guardado numa localização de memória sem que lhe alteremos o conteúdo. Serão tratados na devida altura no Capitulo 18.
Depois dos caracteres gráficos definidos pelo utilizador vêm os símbolos compostos.
Deve ter reparado que não imprimimos os primeiros 32 caracteres, com os códigos de Ø a 31. Estes são caracteres de controle. Não produzem nada que se possa imprimir, mas têm um efeito menos tangível no écran de televisão, ou então são usados para controlar outras coisas que não a televisão, e a televisão imprime ? para mostrar que não os compreende. São descritos mais completamente no Apêndice A.
Três que o televisor usa são os que têm os códigos 6, 8 e 13; no fim de contas o único que poderá ter interesse é o CHR$ 8.
O CHR$ 6 imprime um espaço exactamente como uma vírgula numa instrução PRINT. Por exemplo

```
PRINT 1; CHR$ 6;2
```

.../

faz o mesmo que

```
PRINT 1,2
```

Obviamente esta não é uma maneira muito clara de o usar. Um modo mais subtil seria dizer

```
LET a$="1"+CHR$ 6+"2"
PRINT a$
```

O **CHR$ 8** é um espaço para trás: anda a posição do cursor um caracter para trás - experimente

```
PRINT "1234"; CHR$ 8;"5"
```

que escreve **1235**.

CHR$ 13 é o 'salto de linha': move a posição do cursor para o começo da linha seguinte.
A televisão também usa os caracteres que têm os códigos 16 a 23; estes são explicados nos capítulos 15 e 16. Todos os caracteres de controle estão listados no Apêndice A.
Usando os códigos para os caracteres podemos estender o conceito de 'ordenação alfabética' para cobrir cadeias com caracteres arbitrários, e não sòmente letras. Se em vez de pensarmos em termos do vulgar alfabeto de 26 letras usarmos o alfabeto estendido de 256 caracteres, na mesma ordem que os seus códigos, então o princípio é exactamente o mesmo. Por exemplo, estas cadeias estão na ordem alfabética do ZX Spectrum. (repare na característica estranha de que as letras minúsculas vêm depois de todas as letras maiúsculas de modo que um "a", vem depois do "Z", e também os espaços contam).

```
CHR$ 3+   "JARDIM ZOOLOGICO"
CHR$ 8+   "CAÇA AO AARDVARK"
"  AAAARGH!"
"(Nota entre parentesis)"
"100"
"129.95 c/ IT"
"AASVOGEL"
```

.../

```
"Aardvark" "
"PRINT"
"Zoologico"
 "[interpolação]"
"aardvark"
"aasvogel"
"zoologico"
```

Aqui está uma regra para saber em que ordem vêm duas cadeias. Primeiro comparar o primeiro caracter. Se forem diferentes, então um deles tem um código menor do que o outro, e a cadeia em que ele figura é a primeira (menor) das duas. Se forem iguais, então comparar os caracteres seguintes. Se neste processo uma das cadeias acabar antes da outra, então essa é a primeira cadeia; de outra forma terão de ser iguais.
As relações =, <, >, <=, >= e <> são usadas para cadeias de forma idêntica à dos números: < significa 'vem antes de' e > significa 'vem depois', de modo que

```
"AA homem"<"AARDVARK"
   "AARDVARK">"AA homem"
```

são ambas propqsições verdadeiras.

<= e >= = funcionam do mesmo modo que com números, de modo que

```
"A cadeia" <  ="A cadeia"
```

é verdade, mas

```
"A mesma cadeia"  <  "A mesma cadeia"
```

é falso.
Experimente tudo isto usando o programa que aqui vem, que introduz duas cadeias e as coloca por ordem.

```
10 INPUT  "Introduza as duas cadeias:" a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
```

.../

```
30 PRINT a$;" ";
40 IF a$<b$ THEN PRINT "<"; GO TO 60
50 PRINT "="
60 PRINT " ";b$
70 GO TO 10
```

Note como tivemos de introduzir **c$** na linha 2Ø para trocar **a$** com **b$**.

```
LET a$=b$: LET b$=a$
```

não teria o efeito desejado.

Este programa cria caracteres gráficos definidos pelo utilizador para peças de xadrez:

P para o Peão
T para a Torre
C para o Cavalo
B para o Bispo
R para o Rei
Q para a Rainha

Peças de xadrez

```
 5 LET b=BIN 01111100: LET c=BIN
   00111000: LET d=BIN 00010000
10 FOR n=1 TO 6: READ p$: REM 6      peças
20 FOR f=0 TO 7: REM    ler a peça para os 8 bytes
30 READ a: POKE USR p$+f,a
40 NEXT f
50 NEXT n
100 REM bispo
110 DATA "b",0,d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM king
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM rook
170 DATA "r",0, BIN 01010100,b,c
```

.../

```
180 DATA c,b,b,0
190 REM rainha
200 DATA "q",0, BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM peão
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM cavalo
260 DATA "n",0,d,c, BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Note que pode-se usar Ø em vez de **BIN 00000000**.
Quando correr o programa, veja como são as peças entrando no modo gráfico.

Exercícios

1. Imagene que o espaço para o simbolo é dividido em quatro partes como um bolo. Depois se cada quarto puder ser branco ou preto, há 2x2x2x2=16 possibilidades. Encontre-as todas no conjunto dos caracteres.

2. Corra o programa:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Se fizer experiências com ele, verá que **CHR$ a** é arredondado para o número inteiro mais próximo; e se **a** não estiver entre Ø e 255 então o programa pára e dá uma mensagem de erro **B integer out of range** .

3. Qual destes dois simbolos é menor?

"MAU"
"mau"

4. Veja como pode modificar o programa para estabelecer os caracteres gráficos definidos pelo utilizador, para que ele use **READ** e **DATA** em vez da instrução **INPUT** .

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

15

CAPÍTULO 15

Mais sobre as instruções PRINT e INPUT

Sumário

CLS

Elementos de **PRINT**: nada
Expressões (de tipo numérico ou de cadeia): **TAB** expressão numérica, **AT** expressão numérica, expressão numérica
Separadores de **PRINT**: , ; '
Elementos de **INPUT**: variáveis (de tipo numérico ou de cadeia)
LINE variável de cadeia
Elementos de **PRINT** que não começam com letras. (Símbolos compostos não se consideram como começando com uma letra).
Scrolling

Já vimos a instrução **PRINT** usada em muitas condições, por isso já deve ter uma ideia do modo como é usada. Expressões cujos valores são impressos, são chamados elementos de **PRINT**, e são separados por vírgulas, ponto-e-vírgula, que são chamados os separadores de **PRINT**. Um elemento de **PRINT** também pode consistir em nada, o que é um modo de explicar o que acontece quando se utilizam duas vírgulas de seguida.
Há mais duas espécies de elementos de **PRINT**, que são usados para dizer ao computador não o que deve imprimir mas sim onde. Por exemplo **PRINT AT 11,16;"*"** imprime um asterístico no meio do écran.

AT linha, coluna

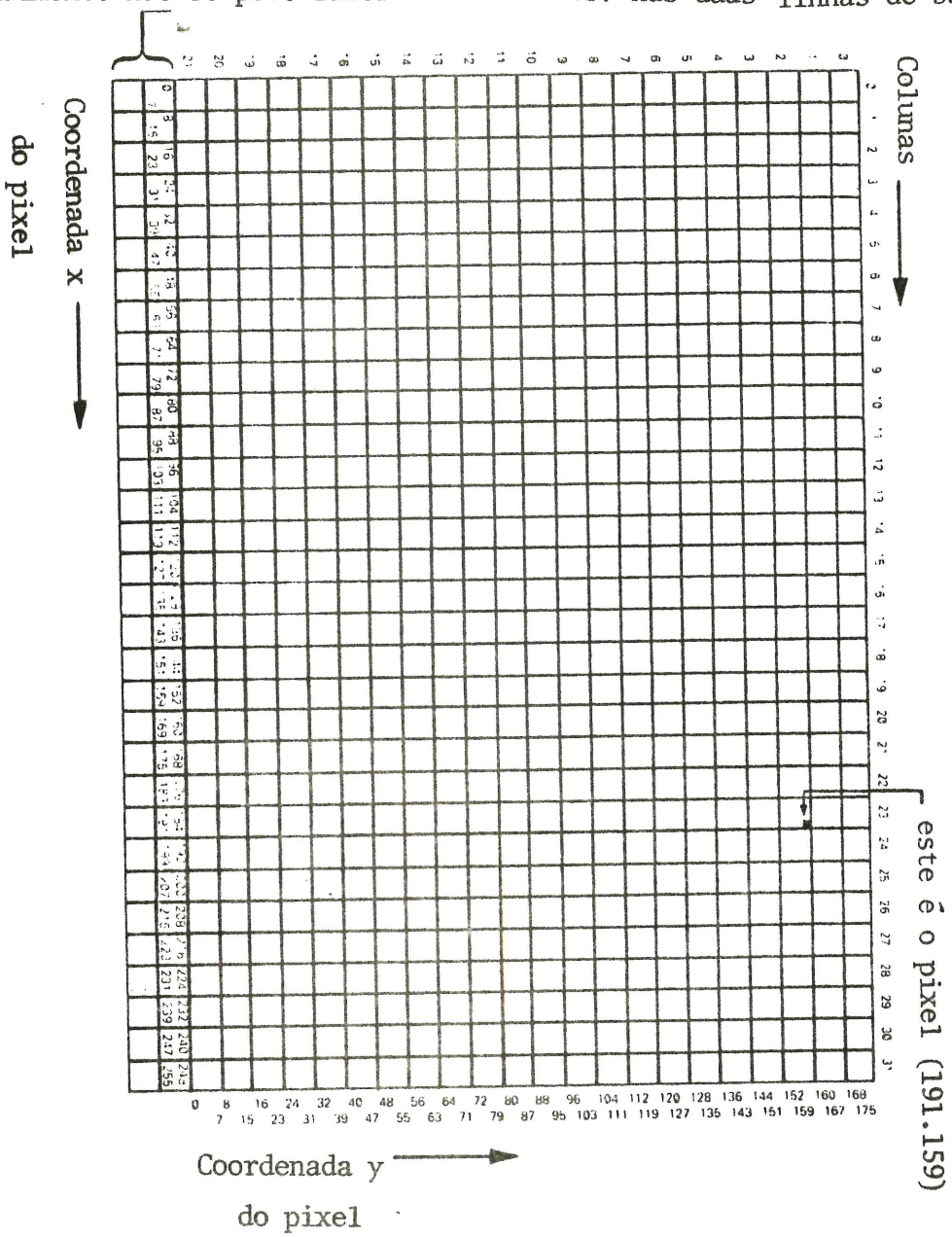
move a posição de **PRINT** (o local onde o próximo elemento deve ser impresso) para a linha e coluna especificadas. As linhas estão numeradas de Ø (no cimo do écran) até 21, e as colunas de Ø (à esquerda) até 31.

TAB coluna

imprime um número de espaços suficiente para mover a posição de **PRINT** para a coluna especificada. Fica sempre na mesma linha a não ser que a instru-

.../

Normalmente não se pode fazer **PRINT** nem **PLOT** nas duas linhas de baixo



Um exemplo:

.../

ção envolvesse um recuo, caso em que passa para a linha seguinte. Repare que o computador reduz o número de coluna por 'módulo 32' (divide por 32 e fica somente com o resto); assim TAB 33 significa o mesmo que TAB 1.
Por exemplo

```
PRIN PRINT TAB 30;1;TAB 12;"conteúdo"; AAT 3,1; "Capítulo";
 TAB 24; "pagina"
```

seria o modo pelo qual se podia fazer um índice de páginas na primeira página de um livro.
Emperimente correr este programa:

```
10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Isto mostra o que significa que os números na instrução TAB sejam reduzidos a módulo 32.
Para um exemplo mais elegante, modifique o 8 na linha 2Ø para 6.
Algumas pequenas indicações:
(i) Estes novos elementos são normalmente melhor utilizados quando terminam com um ponto-e-vírgula, como fizemos acima. Também pode usar vírgulas (ou nada, se fôr no fim da instrução), mas isto significa que depois de ter cuidadosamente escolhido a posição de impressão, ela vai ser imediatamente alterada, o que não parece ter grande utilidade.
(ii) Não se pode imprimir nada nas duas últimas linhas (22 e 23) porque estão reservadas para instruções, dados para INPUT, mensagens, etc. Referências à'última linha' normalmente quer dizer linha 21.
(iii) Pode-se usar a instrução AT para colocar a posição de impressão num sítio onde já esteja alguma coisa escrita; as impressões anteriores são apagadas quando se escreve mais.
Outra instrução que está relacionada com PRINT é CLS. Este instrução limpa todo o écran, uma coisa que também é feita por CLEAR e RUN.
Quando a impressão atinge o fim do écran, começa a puxar tudo para cima, tal como numa máquina de escrever. Pode ver isto se fizer

```
CLS: FOR n=1 TO 22: PRINT n: NEXT n
```

.../

e depois fazer

PRINT 99

várias vezes.
Se o computador estiver a imprimir montes de coisas, então ele toma grande cuidado para se certificar que nada sai fora do écran até que você tenha visto tudo. Pode ver isto acontecer se escrever

CLS: FOR n=1 TO 100: PRINT n: NEXT n

Quando acabou de imprimir um écran cheio ele pára escrevendo scroll? no fundo do écran. Agora pode examinar os primeiros 22 números com calma. Quando acabar de os examinar carregue em **y** (yes-sim) e o computador encherá outro écran com números. Na verdade qualquer tecla fará o computador continuar exceptuando **n** (que quer dizer n), **STOP** (**SYMBOL SHIFT** e **a**), ou **SPACE** (a tecla de **BREAK**). Estas três teclas farão o computador parar com a mensagem D **D BREAK - CONT** repeats.
A instrução de **INPUT** pode fazer muito mais coisas do que as que já lhe dissemos. Já viu instruções de **INPUT** como

INPUT "qual é a sua idade?, idade

nas quais o computador imprime o literal qual é a sua idade? no fundo do écran e depois lhe pede para escrever a sua idade.
De facto uma instrução de **INPUT** é feita de elementos e separadores exactamente da mesma forma que uma instrução **PRINT**, de modo que qual é a sua idade e idade são elementos de **INPUT**. Os elementos de **INPUT** são geralmente os mesmos que os elementos de **PRINT**, mas há algumas diferenças importantes.
Primeiramente temos um elemento de **INPUT** que não existia antes, e que é a variável cujo valor você deve introduzir - idade no exemplo que démos acima. A regra é que se um elemento de **INPUT** começa com uma letra, deve ser uma variável cujo valor deve ser introduzido.
Segundo, isto podia fazer parecer que não se pode imprimir o valor de variáveis como parte dos literais; no entanto pode-se obviar a esse problema colocando parentesis em torno da variável. Qualquer expressão que comece com uma letra tem de ser encerrada entre parentesis para ser impressa numa instrução

.../

de INPUT.

Qualquer espécie de elemento de PRINT que não seja afectado por estas regras também é um elemento de INPUT. Aqui está um exemplo para vermos o que se passa:

```
LET minha idade = INT (RND * 100): INPUT ("eu tenho"; minha
    idade; "anos."); "qual é a sua idade?". sua idade
```

A variável minha idade está entre parentesis, de modo que o seu valor é impresso. Sua idade não está entre parentesis, de modo que você terá de introduzir o seu valor.

Tudo o que numa instrução de INPUT escreve tem de ir para a parte de baixo do écran, que de certa forma é independente da parte de cima. Em particular, as suas linhas estão numeradas relativamente à linha de cima da metade de baixo, mesmo que a sua utilização tenha empurrado para cima a parte que constitui verdadeiramente o écran de televisão (e isto acontece se se escrever muitas coisas na instrução de INPUT).

Para ver como a instrução AT funciona conjuntamente com INPUT tente correr este programa:

```
10 INPUT "Esta é a linha 1.", a$; AT 0,0; "Esta é a linha 0.",a$;
        AT 2,0; "Esta é a linha 2.", a$; AT 1,0; "Esta ainda
    é a linha 1.",a$
```

(carregue sòmente em ENTER cada vez que o programa parar). Quando fôr escrito Esta é a linha 2, a parte de baixo do écran movo-se para cima para abrir espaço pera ela; mas a numeração também sobe, de modo que as linhas de texto mantêm a mesma numeração.

Experimente agora:

```
10 FOR n=0 TO 19: PRINT AT n,0;n;: NEXT n
20 INPUT AT 0,0;a$; AT 1,0;a$; AT 2,0;a$; AT 3,0;a$; AT 4,0;a$;
   AT 5,0;a$;
```

à medida que a parte de baixo do écran sobe, a parte de cima mantem-se imperturbada até que a parte de baixo ameaça escrever na mesma linha em que se encontra a posição de PRINT. Então a parte de cima começa a subir também (scrolling) para evitar isto.

.../

Outro refinamento da instrução de INPUT que ainda não vimos é chamado de entrada LINE (em linha), que é uma forma diferente de fazer a entrada de variáveis de cadeia. Se escrevermos LINE antes do nome de uma variável de cadeia que deve ser introduzida, como em

```
INPUT LINE a$
```

então o computador não lhe vai dar as aspas de cadeia que normalmente apresenta nos casos de variáveis deste tipo, ainda que para ele tudo se passe como se ele tivesse escrito. Como se se escrever

gato

como dado para a instrução de INPUT, a$ ficará com o valor gato. Dado que as aspas não aparecem em cadeia, não se pode apagá-las para escrever um tipo diferente de expressão de cadeia, para dado, para a instrução de INPUT. Lembre-se que não pode usar a instrução LINE para entrada de variáveis numéricas. Os caracteres de controle CHR$ 22 e CHR$ 23 têm mais ou menos o mesmo efeito que AT e TAB.. São muito estranhos como caracteres de controle, porque quando um é enviado para a televisão para ser impresso, ele deve ser seguido por mais dois carecteres que não irão ter o seu efeito habitual: são tratados como números (os seus códigos) para especificar a linha e a coluna (para a instrução AT) ou a posição do tabulador (para TAB). Verá que é sempre mais fácil usar as instruções AT e TAB na sua forma usual em vez de usar os caracteres de controle, mas podem ser úteis em algumas circunstâncias. O caracter de controle para a instrução AT é o CHR$ 22 . O primeiro caracter que se lhe segue especifica o número de linha e o segundo de coluna, de modo que

```
PRINT CHR$ 22+CHR$ 1+CHR$ c;
```

tem exactamente o mesmo efeito que

```
PRINT AT 1,c;
```

Isto é verdade mesmo que o CHR$ 1 ou CHR$ c tivessem normalmente outro significado (por exemplo se c=13 ; o CHR$ 22 antes deles passa por cima desse significado.

.../

O caracter de controle para a instrução **TAB** é o **CHR$ 23** e os dois caracteres depois dele são usados para dar um número entre Ø e 65535 que especifica o número que normalmente seria colocado no elemento **TAB**:

```
PRINT CHR$ 23+CHR$ a+CHR$ b;
```

terá o mesmo efeito que

```
PRINT TAB a+256*b;
```

Pode usar a instrução **POKE** para evitar que o computador lhe faça a pergunta scroll? fazendo

```
POKE 23692,255
```

de vez em quando. Depois disto ele subirá 255 linhas antes de parar para perguntar scroll?. Por exemplo experimente

```
10 FOR n=0 TO 10000
20 PRINT n: POKE 23692,255
30 NEXT n
```

e veja como tudo passa a correr no écran!

Exercícios

1. Experimente este programa com algumas crianças, para testar a tabuada.

```
10 LET m$=""
20 LET a=INT (RND*12)+1: LET b=INT (RND*12)+1
30 INPUT (m$)  ""quanto é"; (a);" * ";(b);"?";c
100 IF c=a*b THEN LET m$        = "Certo.": GO TO 20
110 LET m$  = "Errado. Experimente outra vez." GO TO 30
```

Se forem de compreensão rápida, talvez consigam perceber que não têm eles de fazer os cálculos. Por exemplo, se o computador lhes perguntar quanto é 2 * 3, tudo o que eles têm que fazer é introduzir **2*3**.

.../

Um modo de rodear este problema é de os fazer introduzir variáveis de cadeia em vez de números. Substituia c na linha 3Ø por c$, e na linha 1ØØ por VAL c$, e introduza a linha

```
40 IF c$<> STR$ VAL c$ THEN LET m$ = "Escreva como deve ser,
como um número." : GO TØ 3Ø
```

Isso já os vai enganar. Depois de mais alguns dias, no entanto, um deles é capaz de descobrir que pode superar isto apagando as aspas e escrevendo STR$ (2*3) . Para evitar esta trafulhice pode substituir o c$ na linha 3Ø por LINE c$.

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

16

CAPITULO 16

Cores

Sumário

INK, PAPER, FLASH, BRIGHT, INVERSE, OVER
BORDER

Execute este programa :

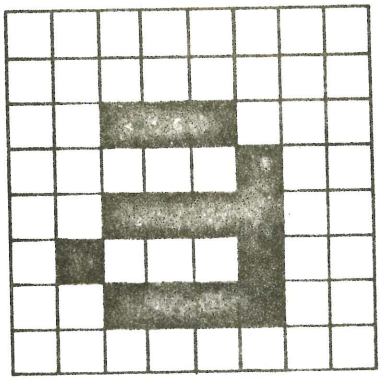
```
10 FOR m=0 TO 1: BRIGHT m
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT "     ";: REM 4 espaços coloridos
50 NEXT c: NEXT n: NEXT m
60 FOR m=0 TO 1: BRIGHT m: PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c;"  ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"  ";
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Este programa mostra as oito cores (incluindo branco e negro) e os dois niveis de brilho que o ZX Spectrum pode produzir numa televisão a cores . (Se a sua televisão fôr a preto e branco, então só conseguirá ver várias tonalidades de cinzento.) Aqui está uma lista das oito cores, para referência; também estão escritas acima das respectivas teclas numéricas.

0 - negro
1 - azul
2 - vermelho
3 - púrpura ou magenta
4 - verde
5 - azum pálido, técnicamente chamado cian
6 - amarelo
7 - branco

Numa televisão a preto e branco estes números dão a ordem de brilho .

Para se usar estas cores adequadamente, é necessário compreender um pouco da forma como a imagem está organizada.
A imagem está dividida em 768 posições (24 linhas de 34 posições) onde se podem imprimir carcteres, e cada caracter é impresso como uma matriz quadrada de 8 x 8 pontos. Como por exemplo para o a. Isto deve fazê-lo lembrar-se dos caracteres gráficos definidos pelo utilizador no 16, onde tinhamos Ø's pa



ra os pontos brancos e 1's para os pontos negros.
A posição respeitante a um caracter também tem associada com ela duas cores: a ink (tinta), ou cor de cima, que é a cor que vão assumir os pontos negros no nosso quadrado, e o paper (papel), ou cor do fundo, que é usada para os pontos brancos. Para começar todas as posições têm tinta negra e papel branco de modo que a escrita faz-se a preto e branco.
A posição do caracter também tem brilho (normal ou acentuado) e algo que diga se deve piscar ou não - para fazer um caracter piscar troca-se a tinta e o papel. Isto pode ser tudo codificado em números, de modo que uma posição tem:
(i) uma qradrado de 8 x 8 de Øs de 1's para definir a forma do caracter, representando um Ø a cor do papel e 1 a cor da tinta,
(ii) cor de papel e de tinta (respectivamente PAPER e INK), cada uma delas codificada num número entre Ø e 7,
(iii) brilho - Ø para o normal e 1 para o acentuado e
(iv) um codigo para piscar - Ø estacionário, 1 a piscar.
Note que uma vez que as cores do papel e da tinta se referem a uma posição de caracter completo, não se pode ter mais do que duas cores por cada bloco de 64 pontos. O mesmo diz respeito ao brilho e ao código de piscar; referem-se a uma posição de caracter completa, não a pontos individuais. As cores, brilho e o código de piscar numa dada posição são chamados os atributos.

.../

Quando se escreve alguma coisa no écran, modifica-se o padrão de pontos nessa posição; é menos evidente, mas também verdade, que se pode modificar os atributos nessa posição. Para começar não se nota isto porque está tudo a ser escrito a preto e branco (e com brilho normal e sem piscar), mas pode-se modificar isto com as instruções **INK**, **PAPER**, **BRIGHT** e **FLASH**. Experimente

```
PAPER 5
```

e depois escreva algumas coisas: vão aparecer sobre um papel cor de cian, porque à medida que elas são escritas as posições que são ocupadas vêm a sua cor de papel alterada para cian (que corresponde ao código 5).
As outras instruções funcionam do mesmo modo, de modo que de

PAPER número entre Ø e 7
INK número entre Ø e 7
BRIGHT Ø ou 1
ou
FLASH Ø ou 1

} Pense em Ø como desligado e 1 como ligado

qualquer impressão depois destas instruções colocará os atributos correspondentes de todas as posições que usar nestes valores. Experimente algumas. Agora já deve ver como é que o programa que lhe demos no princípio funcionava (lembre-se que um espaço é um caracter que tem a tinta (**INK**) e o papel (**PAPER**) da mesma cor).
Há outros números que podem ser usados nestas instruções e que têm um efeito menos directo.
O 8 pode ser usado em todas as quatro instruções e significa 'transparente' no sentido que os atributos antigos devem transparecer. Suponhamos, por exemplo que se faz

```
PAPER 8
```

Nenhuma posição de impressão terá a sua cor passada para 8 porque essa cor não existe: o que acontece é que quando se imprime nessa posição, a sua cor de papel é deixada como estava. **INK 8**, **BRIGHT 8** e **FLASH 8** trabalham exactamente do mesmo modo para os outros atributos.
O 9 pode ser usado sòmente com as instruções **PAPER** e **INK**, e significa 'contraste'. A cor (tinta ou papel) que se usar com ela vai ser obrigada a con-

.../

trastar com a outra, ficando branca se a outra cor fôr escura (negro, azul, vermelho ou magenta), e negra se a outra cor fôr clara (verde, cian, amarelo ou branco).
Experimente isto fazendo

```
INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c
```

Uma exibição mais impressionante do seu poder é correr o programa do princípio para fazer tiras de cor e depois escrever

```
INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000: PRINT n;:
NEXT n
```

A cor da tinta nesta instrução é sempre obrigada a contrastar com a anterior cor do papel em cada posição.
As televisões a cores fiam-se no facto algo curioso que o olho humano na verdade só consegue distinguir três cores - as cores primárias, azul, vermelho e verde. As outras cores são misturas destas. Por exemplo, o magenta é feito misturando azul com vermelho - motivo porque o seu código é 3, sendo a soma dos códigos para o azul e o vermelho.
Para ver com as oito cores se encaixam, imagine três focos de luz rectangulares, um azul, outro vermelho e o terceiro verde, incidindo em papel branco no escuro mas não exactamente no mesmo local. Onde elas se sobrepõem ver-se-á misturas de cores, como é demonstrado neste programa (repare que os espaços com a cor da tinta são obtidos usando a tecla **8** com **SHIFT** no modo G):

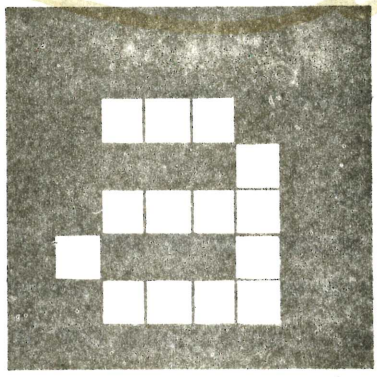
```
10 BORDER 0: PAPER 0: INK 7: CLS
20 FOR a=1 TO 6
30 PRINT TAB 6; INK 2;"[18 quadrados da cor da tinta]": REM
   18 quadrados da cor da tinta
   squares
40 NEXT a
50 LET   linha de dados = 200

60 GO SUB 1000
70 LET[ linha de dados = 210
80 GO SUB 1000
90 STOP
200 DATA 2,3,7,5,4
210 DATA 2,2,6,4,4
1000 FOR a=1 TO 6
```

.../

```
1010 RESTORE linha de dados
1020 FOR b=1 TO 5
1030 READ c: PRINT INK c;"      ":: REM 6 6 quadrados
1040 NEXT b: PRINT : NEXT a
1050 RETURN
```

Há uma função chamada **ATTR** que vê quais são os atributos de uma dada posição do écran. É uma função bastante complicada, de modo que será relegada para o fim deste capítulo.

Há mais duas instruções, **INVERSE** e **OVER**, que controlam, não os atributos, mas o padrão de pontos que é impresso no écran. Usam os números Ø para desligado e 1 para ligado, tal como as instruções **FLASH** e **BRIGHT**, mas são somente essas as únicas possibilidades. Se se fizer **INVERSE 1**, então o padrão de pontos que é impresso será o inverso da sua forma vulgar: os pontos da cor do papel serão substituídos por pontos da cor da tinta e vice-versa. Assim um a seria impresso como



Se (na altura em que o ligarmos) tivermos tinta negra e papel branco, então o a aparecerá escrito em branco sobre negro - mas a tinta continuará a ser negra e o papel branco nessa posição.

Os pontos é que são trocados.

A instrução

.../

OVER 1

faz entrar em acção um estilo particular de impressão sobreposta. Normal - mente quando algo é escrito numa posição de caracter apaga completamente o que lá estava antes; mas agora o novo caracter será simplesmente adicionado sobre o outro (mas veja o exercício 3): Isto pode ser particularmente útil quando se escrevem caracteres compostos, como letras com acentos, como neste programa que escreve letras Alemãs - um 'o' com um trema sobre ele. (Faça primeiro **NEW**)

```
10 OVER 1
20 FOR n=1 TO 32
30 PRINT "o"; CHR$ 8;"""";
40 NEXT n
```

(repare que foi usado o **CHR$ 8** que recua um espaço).
Há outro modo de usar as nustruções **INK**, **PAPER** e assim por diante, e que você poderá achar mais útil do que tê-las como instruções. Pode colocá-las como elementos numa instrução (**PRINT** seguidas de ;) e podem fazer exactamente o mesmo que teriam feito se fossem usadas como instruções isoladas, excepto que o seu efeito é só temporário: dura sòmente até se chegar ao fim da instrução **PRINT** que as contém. Assim, se escrevermos

```
PRINT PAPER 6;"x";: PRINT "y"
```

então sòmente o x será a amarelo.
Tanto a instrução **INK** como as outras quando usadas como instruções não afectam a cor da parte de baixo do écran, onde os comandos e os dados para **INPUT** são introduzidos. A parte de baixo do écran a cor do bordo para sua cor de papel e o código 9 para contraste com a sua cor de tinta, não pisca e tem tudo com brilho normal. Pode-se mudar a cor do bordo para qualquer das oito cores (exceptuando 8 ou 9) usando a instrução

BORDER cor

Quando se introduzem dados para **INPUT**, estes seguem a regra do contraste da tinta sobre a cor do papel da borda, mas pode-se alterar a côr dos comentários escritos pelo computador usando os elementos **INK** e **PAPER** (e assim por

.../

diante) na instrução de **INPUT** tal como se faria para uma instrução **PRINT**. O seu efeito dura até ao fim da instrução, ou até que seja introduzido algum dado, dependendo do que acontecer primeiro. Experimente.

```
INPUT FLASH 1; INK 1; "Qual é o seu número?"; n
```

Há ainda outro modo de modificar as cores utilizando caracteres de contro - le - mais ou menos como os caracteres de controle para o **AT** e o **TAB** no capí tulo 15.

CHR$ 16 corresponde a **INK**
CHR$ 17 corresponde a **PAPER**
CHR$ 18 corresponde a **FLASH**
CHR$ 19 corresponde a **BRIGHT**
CHR$ 20 corresponde a **INVERSE**
CHR$ 21 corresponde a **OVER**

Cada um destes caracteres de controle é seguido por um caracter que indica uma cor com o seu código: assim (por exemplo):

```
PRINT CHR$ 16+CHR$ 9; ....
```

tem o mesmo efeito que

```
PRINT INK 9; ...
```

Regra geral não deve ser muito vantajoso usar os caracteres de controle porque é mais fácil utilizar os elementos de cor. No entanto uma coisa muito útil que se pode fazer com eles é colocá-los em programas: isto faz com que diversas partes sejam listadas com cores diferentes, as afaste ou torne a sua aparência mais bonita. Têm de ser colocadas logo a seguir ao número de linha ou perder-se-ão.

Para colocar estes caracteres num programa tem de os colocar a partir do teclado, essencialmente usando o modo de extensão e dígitos.

Os dígitos de Ø a 7 preparam a cor correspondente - tinta se também se tiver carregado em **CAPS SHIFT**, papel no caso contrário. Mais precisamente, se se estiver no modo E e se carregar num número (digamos 6, para dar amarelo; de qualquer forma tem de estar entre Ø e 7 - nunca 8 nem 9) então serão in

.../

seridos dois caracteres: primeiro **CHR$ 17** para **PAPER** e **CHR$ 6** que representa 'amarelo'. Se se estivesse a carregar também em **CAPS SHIFT** quando se introduz um dígito, obter-se-ia **CHR$ 16** que significa 'cor da tinta' em vez do **CHR$ 17**.

Uma vez que com estes dois caracteres se podem obter efeitos estranhos quando se apagam - é necessário carregar duas vezes em **DELETE** e depois da primeira vez verá que frequentemente lhe aparece um ponto de interrogação ou mesmo coisas mais estranhas. Não se preocupe, carregue outra vez no **DELETE**.

◂ e ▸ também se podem comportar estranhamente enquanto o cursor estiver a passar pelo controle dos caracteres.

Ainda no modo de extensão

8 dá **CHR$ 19** e **CHR$ 0** para brilho normal
9 dá **CHR$ 12** e **CHR$ 1** para brilho acentuado
CAPS SHIFT com 8 dá **CHR$ 18** e **CHR$ 0** para não piscar
CAPS SHIFT com 9 dá **CHR$ 19** e **CHR$ 1** para piscar

Também há alguns no modo vulgar (L):

CAPS SHIFT com 3 dá **CHR$ 20** e **CHR$ 0** para caracteres normais
CAPS SHIFT com 4 dá **CHR$ 20** e **CHR$ 1** para caracteres inversos

Resumindo, aqui está uma descrição completa da fila de cima do teclado:

MODE	SHIFT										
E	SYMBOL	**DEF FN**	**FN**	**LINE**	**OPEN #**	**CLOSE #**	**MOVE**	**ERASE**	**POINT**	**CAT**	**FORMAT**
	CAPS	Ink blue	Ink red	Ink magenta	Ink green	Ink cyan	Ink yellow	Ink white	Flash off	Flash on	Ink black
	NONE	Paper blue	Paper red	Paper magenta	Paper green	Paper cyan	Paper yellow	Paper white	Normal brightness	Extra bright	Paper black
G	EITHER	▙	▛	▬	▗	▌	▚	▖	■	EXIT GRAPHICS	**DELETE**
	NONE	▟	▜	▬	▗	▐	▚	▖	□	EXIT GRAPHICS	**DELETE**
K,L or C	CAPS	**EDIT**	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO	⇦	⇩	⇧	⇨	GRAPHICS MODE	**DELETE**
	SYMBOL	!	@	#	$	%	&	'	(	)	_
	NONE	1	2	3	4	5	6	7	8	9	0

.../

A função **ATTR** tem a forma

ATTR (linha, coluna)

Os seus dois argumentos são os números de linha e de coluna que se usaria num elemento **AT**, e o resultado é um número que mostra as cores e assim por diante que corresponde à posição do caracter no écran de televisão. Pode-se usar este valor com perfeita liberdade em expressões, tal como outra função qualquer.

O número que é originado como resultado é a soma de quatro outros números:

128 se a posição estiver a piscar, e Ø se não estiver
64 se a posição fôr brilhante acentuado, e Ø se fôr normal
8*vezes o código de cor do papel
o código de cor da tinta

Por exemplo, se a posição do caracter estiver a piscar, com brilho normal e papel amarelo e tinta azul, então os quatro números que temos de adicionar são 128, Ø, 8 * 6 = 48 e 1, fazendo um total de 177. Experimente com

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1;" "; ATTR (0,0)
```

Exercícios

1. Experimente

```
PRINT "B"; CHR$ 8; OVER 1;"/";
```

Onde a / cortou o B deixou um ponto branco. É assim que a impressão sobreposta funciona no ZX Spectrum: dois pontos de papel ou dois de tinta dá um ponto de papel, um de cada dá um de tinta. Esta é uma propriedade interessante; se se imprimir duas vezes a mesma coisa voltamos ao ponto de partida. Se agora escrever

```
PRINT CHR$ 8; OVER 1;"/"
```

porque é que se recupera um **B** intacto?

.../

2. Escreva

```
PAPER 0: INK 0
```

- não é bom que esta instrução não afecte a parte de baixo do écran?
Agora escreva

```
BORDER 0
```

e veja como o computador toma conta de si !

3. Corra este programa:

```
10 POKE 22527+RND*704, RND*127
20 GO TO 10
```

Não se preocupe com o modo como isto funciona; está a mudar as cores dos quadrados do écran de televisão e os **RND**'s asseguram-lhe que isto acontece aleatòricamente. As listas diagonais que eventualmente vai observar são as manifestações do padrão escondido em **RND** - o padrão que o torna pseudo-aleatório em vez que verdadeiramente aleatório.

4. Introduza pelo teclado um faça um **LOAD** dos caracteres para as peças de xadrez do capítulo 14, e dpois escreva este programa que desenha um diagrama de uma posição de xadrez usando-os.

```
5 REM desenhar o tabuleiro vazio
  10 LET bb=1: LET bw=2: REM tabuleiro azul e vermelho
  15 PAPER bw: INK bb: CLS
  20 PLOT 79,128: REM bordo
  30 DRAW 65,0: DRAW 0,-65
  40 DRAW -65,0: DRAW 0,65
  50 PAPER bb
60 REM tabuleiro
 70 FOR n=0 TO 3: FOR m=0 TO 3
```

.../

```
 80 PRINT AT 6+2*n, 11+2*m;" "
 90 PRINT AT 7+2*n, 10+2*m;" "
100 NEXT m: NEXT n
110 PAPER 8
120 LET pw=6: LET pb=5: REM      cores das peças brancas
e negras
  200 DIM b$(8,8): REM posição das peças
205 REM posições iniciais
210 LET b$(1)="rnbqkbnr"
220 LET b$(2)="pppppppp"
230 LET b$(7)="PPPPPPPP"
240 LET b$(8)="RNBQKBNR
300 REM saída do tabuleiro
310 FOR n=1 TO 8: FOR m=1 TO 8
320 LET bc=CODE b$(n,m): INK pw
  325 IF bc=CODE" " THEN GO TO 350: REM espaço
  330 IF bc>CODE "Z" THEN INK pb: LET bc=bc-32: REM
    letra minuscula para as negras
340 LET bc=bc+79: REM converter em caracteres gráficos
350 PRINT AT 5+n, 9+m; CHR$ bc
360 NEXT m: NEXT n
400 PAPER 7: INK 0
```

.../

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

17

CAPÍTULO 17

Gráficos

Sumário

PLOT, DRAW, CIRCLE
POINT

pixels

Neste capítulo vamos ver como se podem desenhar imagens no ZX Spectrum. A parte do écran que você pode usar tem 22 linhas e 32 colunas, fazendo ao todo 22*32=7Ø4 posições de caracteres. Como talvez se lembre do Capítulo 16 cada uma destas posições de caracter é feita de um quadrado de 8 por 8 pontos, e a estes chamaremos pixels (elementos de imagem).

Um pixel é especificado por dois números, as suas coordenadas. O primeiro, a sua coordenada x, indica a que distância ele está da coluna da esquerda. (lembre-se que x é na horizontal), a segunda, a sua coordenada y, diz a que distância está do fundo (contando para cima). Estas coordenadas são geralmente escritas como um par entre parêntesis, de modo que (Ø,Ø), (255,Ø), (Ø,175) e (255,175) são, respectivamente, os cantos inferior esquerdo, inferior direito, superior esquerdo e superior direito.

A instrução

PLOT coordenada x, coordenada y

pinta um pixel nestas coordenadas, de modo que este programa

```
10 PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 10
```

imprime um ponto aleatório cada vez que se carregar em **ENTER**.

Aqui está um programa bastante mais interessante. Desenha um gráfico da função **SIN** (uma onda sinusoidal) para valores entre Ø e 2π.

```
10 FOR n=0 TO 255
20 PLOT n,88+80*SIN (n/128*PI)
30 NEXT n
```

Este próximo programa desenha um gráfico de **SQR** (um bocado de parábola) entre Ø e 4:

```
10 FOR n=0 TO 255
20 PLOT n,80*SQR (n/64)
30 NEXT n
```

Repare que as coordenadas dos pixel são algo diferentes das linhas e colunas no elemento **AT**. Talvez ache que o diagrama do Capítulo 15 seja útil quando quiser converter coordenadas de pixel para números de linha e coluna.

Para o ajudar com as suas imagens, o computador poder traçar linhas rectas , círculos e partes de círculo, usando as instruções **DRAW** e **CIRCLE**.

A instrução **DRAW** desenha uma linha recta e toma a forma

```
DRAW x,y
```

O ponto de partida da linha é o pixel onde a última instrução **PLOT**, **DRAW** ou **CIRCLE** parou (a esta posição chama-se a posição **PLOT**, as instruções **RUN**, **CLEAR**, **CLS** e **NEW** repõem-na no canto inferior esquerdo, em (0,0), e o ponto final está a x pixels para a direita e y pixels para cima. A instrução **DRAW** sozinha determina o comprimento e direcção da linha, mas não o seu ponto de partida.

Experimente com algumas instruções **PLOT** e **DRAW**, por exemplo

```
PLOT 0,100: DRAW 80,-35
PLOT 90,150: DRAW 80,-35
```

Repare que os números numa instrução **DRAW** podem ser negativos, ainda que numa instrução **PLOT** não possam ser.

Também pode fazer plot e draw a cores, mas tem de ter em mente que as cores cobrem sempre uma posição de caracter completa e não podem ser especificadas para pixels individuais. Quando um pixel é desenhado é colocado com a cor da tinta, e todo o caracter em que esse ponto está fica com essa cor de tinta atribuida. Este programa demonstra isto:

```
10 BORDER 0: PAPER 0: INK 7: CLS : REM écran negro
20 LET x1=0: LET y1=0: REM principio da linha
30 LET c=1: REM para a cor da tinta, começa em azul
```

```
40 LET x2=INT (RND*256): LET y2=INT (RND*176): REM
   fim da linha é aleatório
50 DRAW INK c;x2-x1,y2-y1
      60 LET x1=x2: LET y1=y2: REM a proxima linha começa
   onde a ultima acabou
         70 LET c=c+1: IF c=8 THEN LET c=1: REM nova cor
         80 GO TO 40
```

As linhas parecem ficar cada vez mais largas à medida que o programa avança, e isto deve-se a que uma linha muda as cores de todos os pontos numa dada posição de caracter por onde passe.

Note que pode incluir elementos **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** e **OVER** numa instrução **PLOT** ou **DRAW** tal como nas instruções **PRINT** ou **INPUT**. Estes têm de entrar entre a palavra-chave e as coordenadas, e são terminadas ou com ponto-e-vírgula ou vírgula.

Uma extravagância extra com a instrução **DRAW** é que se pode traçar partes de circulo em vez de linhas rectas, usando um número extra para especificar um ângulo a ser rodado: a forma é

```
DRAW x,y,a
```

x e **y** são usados para especificar o ponto final da linha, tal como antes, e **a** é o número de radianos que deve girar enquanto avança - se **a** é positivo roda para a esquerda, enquanto que se **a** fôr negativo roda para a direita. Outro modo de ver o significado de **a** é mostrar a fracção de círculo que deve ser traçada: um círculo completo é de 2π radiano, de modo que se $a=\pi$ será desenhado um semi-círculo, se $a=0.5*\pi$ será um quarto de círculo, e assim por diante.

Por exemplo, suponhamos que $a=\pi$. Então quaisquer que sejam os valores de **x** e **y** será desenhado um semicírculo. Corra

```
10 PLOT 100,100: DRAW 50,50, PI
```

que desenhará isto:

Fim em (15Ø,15Ø)

começo em (1ØØ,1ØØ)

O desenho começa numa direcção sudeste, mas quando pára já vai na direcção noroeste: pelo meio rodou de 180 graus, ou π radianos (o valor de a).

Corra o programa várias vezes, com PI substituido por várias expressões - por exemplo -PI, PI/2, 3*PI/2, PI/4, 1,0

A última instrução deste capítulo é a instrução **CIRCLE**, que desenha uma circunferência completa. Especifica-se as coordenadas do centro e o raio do circulo usando

CIRCLE coordenada x, coordenada y, raio

Tal como com as instruções **PLOT** e **DRAW**, podem-se colocar as diversas cores no começo de uma instrução **CIRCLE**.

A função **POINT** diz-nos se um pixel é da cor do papel ou da tinta. Tem dois argumentos, as coordenadas do pixel (e têm de estar entre parêntesis), e o seu resultado é 0 se o pixel fôr da cor do papel e 1 se fôr da cor da tinta. Experimente

```
CLS : PRINT POINT (0,0): PLOT 0,0: PRINT POINT (0,0)
```

Escreva

```
PAPER 7: INK 0
```

e vamos investigar como é que as instruções **INVERSE** e **OVER** funcionam dentro de uma instrução **PLOT**. Estas duas afectam somente o pixel em questão, e não o resto da posição de caracter. Normalmente estão desligadas numa instrução **PLOT** (isto é são 0), por isso só precisa de as mencionar para as ligar (1).

Aqui está uma lista das possibilidades para referência:

PLOT - esta é a forma usual. Imprime um ponto a tinta, isto é faz com que o pixel exiba a cor da tinta.

PLOT INVERSE 1 - esta instrução imprime um apagador de tinta, isto é, faz com que o pixel mostre a cor do papel.

PLOT OVER 1 - muda o pixel para o contrário do que estava antes: se estava com a cor da tinta passa a papel e vice-versa.

PLOT INVERSE 1; **OVER** 1; - Deixa o pixel exactamente como estava antes; mas note que também muda a posiçaõ do **PLOT**, por isso pode usá-la precisamente para isso.

Como outro exemplo de utilização da instrução **OVER** encha primeiro o écran com escrita a preto e branco e depois escreva

```
PLOT 0,0: DRAW OVER 1;275,175
```

Isto vai desenhar uma linha razoavelmente recta, ainda que apresente falhas sempre que atingir algo que já esteja escrito. Agora repita exactamente a mesma instrução. A linha irá desaparecer sem deixar quaisquer traços. Esta é a grande vantagem de **OVER 1**. Se tiver desenhado uma linha usando

```
PLOT 0,0: DRAW 255,175
```

e a apagar usando

```
PLOT 0,0: DRAW INVERSE 1;255,175
```

também irá apagar alguma escrita.

Agora experimente

```
PLOT 0,0: DRAW OVER 1;250,175
```

e experimente apagá-la com

```
DRAW OVER 1;-250,-175
```

Isto não funciona muito bem porque os pixels que são usados no caminho de regresso não são exactamente os mesmos que os que são usados para cima. Tem de apagar cada linha exactamente pelo mesmo processo que usou para a desenhar.

Um modo de obter cores invulgares é de dispersar duas normais num único quadrado, usando um caracter gráfico definido pelo utilizador. Corra este programa:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n, BIN 01010101: POKE USR "a"+n+1,
     BIN 10101010
1020 NEXT n
```

que dá um caracter gráfico definido pelo utilizador correspondente a um

padrão em xadrez. Se imprimir este caracter (modo gráfico e depois a) em tinta vermelha e papel amarelo, verá que dá um laranja muito aceitavel.

Exercícios

1. Brinque um pouco com os elementos PAPER, INK, FLASH e BRIGHT numa instrução PLOT. Estes são os elementos que afectam toda a posição de caracter que contêm o pixel. Normalmente é como se a instrução PLOT começasse com

```
PLOT PAPER 8; FLASH 8; BRIGHT 8; ...
```

e somente a cor da tinta da posição de caracter fosse alterada quando algo é impresso ali, mas pode modificar isto se quiser.

Tenha um cuidado especial quando usar cores com INVERSE 1 porque esta instrução coloca o pixel de forma a mostrar a cor do papel, mas modifica a cor da tinta e isto pode não ser o que você estava à espera.

2. Experimente desenhar círculos usando as funções SIN e COS (se tiver lido o Capítulo 9 tente descobrir como isso se faz). Corra este programa

```
10 FOR n=0 TO 2*PI STEP PI/100
20 PLOT 100+80*COS n,87+80*SIN n
30 NEXT n
40 CIRCLE 150,87,80
```

Pode ver que a instrução CIRCLE é muito mais rápida, ainda que não seja tão precisa.

3. Experimente

```
CIRCLE 100,87,80: DRAW 50,50
```

Pode ver por aqui que a instrução CIRCLE deixa a posição de PLOT num local algo indeterminado - é sempre algures perto do meio do lado direito da circunferência. Geralmente precisará de fazer a instrução CIRCLE ser seguida de um PLOT antes de fazer algum DRAW.

4. Aqui está um programa para desenhar o gráfico de qualquer função. Primeiro pergunta-lhe um número **n**; o gráfico será traçado entre **-n** e **+n** Depois pergunta-lhe qual é a função, entrada como uma cadeia. A cadeia deve ser uma expressão usando **x** por argumento.

```
10 PLOT 0,87: DRAW 255,0
20 PLOT 127,0: DRAW 0,175
30 INPUT s,e$
35 LET t=0
40 FOR f=0 TO 255
50 LET x=(f-128)*s/128: LET y=VAL e$
60 IF ABS y>87 THEN LET t=0: GO TO 100
70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
80 DRAW 1,y-old y
100 LET old y=INT (y+.5)
110 NEXT f
```

Corra-o e, como exemplo, introduza **10** para **n** e **10*TAN x** para a função. Irá aparecer um gráfico da tangente de x quando o argumento varia de -1Ø a 1Ø.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

CAPÍTULO 18

Movimento

Sumário

PAUSE, INKEY$, PEEK

Muitas vezes vai querer que um programa seja executado durante um intervalo de tempo especificado, e para isto vai ver que a instrução **PAUSE** é extremamente útil.

PAUSE n

pára o computador e envia a imagem para a televisão durante n imagens (a 5Ø imagens por segundo na Europa e 6Ø na América).
O n pode ser um número inteiro até 65535, o que dá cerca de 22 minutos; se n=Ø isto significa **PAUSE** para sempre.

Uma pausa pode sempre ser interrompida carregando numa tecla (repare que um espaço simultaneamente com **CAPS SHIFT** também interrompe o programa). Tem de carregar na tecla depois da pausa ter começado.

Este progama move o segundo braço de um relógio:

```
10 REM primeiro desenhamos o mostrador
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 REM Agora ligamos o relógio
   60 FOR t=0 TO 200000: REM t é o tempo em segundos
   70 LET a=t/30*PI : REM a é o angulo do segundo braço
  em radianos
   80 LET sx=80*SIN a: LET sy=80*COS a
  200 PLOT 128,88: DRAW OVER 1;sx,sy: REM desenhar o se-
  gundo braço
  210 PAUSE 42
      220 PLOT 128,88: DRAW OVER 1;sx,sy: REM apagar o se-
  gundo braço
  400 NEXT t
```

Este relógio vai andar durante cerca de 55.5 horas devido à linha 6Ø, mas facilmente o pode fazer durar mais tempo. Repare como o tempo é controlado pela linha 21Ø. Podia esperar que se fizesse **PAUSE 5Ø** para

o fazer andar 1 segundo, mas o cálculo leva algum tempo também e tem de se ter a isso em conta. Isto pode ser feito por tentativas, verificando o relógio do computador com um real, e ajustando a linha 21Ø de acordo com isso. (não se pode fazer isto com muita precisão; um ajuste de uma imagem num segundo é 2% ou cerca de meia hora por dia).

Há um modo muito mais exacto de medir o tempo. Isto usa o conteúdo de certas posições de memória. Os dados armazenados são retirados usando um **PEEK**. O capítulo 25 explica o que estamos a dizer com detalhe. A expressão usada é

```
(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50
```

Isto dá o número de segundos desde que o computador foi ligado (até cerca de 3 dias e 21 horas, altura em que volta a Ø).

Aqui está um programa de relógio para usar isto:

```
10 REM primeiro desenhamos o mostrador
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 DEF FN t()=INT ((65536*PEEK 23674+256*PEEK 23673+
      PEEK 23672)/50): REM número de segundos desde o
   começo
100 REM agora ligamos o relógio
110 LET t1=FN t()
    120 LET a=t1/30*PI : REM a é o angulo do segundo braço
   em radianos
130 LET sx=72*SIN a: LET sy=72*COS a
      140 PLOT 131,91: DRAW OVER 1;sx,sy: REM desenhar o
   braço
    200 LET t=FN t()
      210 IF t<=t1 THEN GO TO 200: REM esperar a altura de
   mover o braço
      220 PLOT 131,91: DRAW OVER 1;sx,sy: REM apagar o braço
   anterior
    230 LET t1=t: GO TO 120
```

O relógio interno que este metodo utiliza deve ter uma precisão de cerca de Ø.1% desde que o computador esteja somente a correr o seu programa, ou seja 1Ø segundos por dia; mas pára temporariamente sempre que utilizar

a instrução **BEEP** ou execute uma operação com o gravador de cassetes, ou utilize a impressora, ou ainda qualquer outra peça de equipamento extra que possa ser utilizada pelo computador. Todas estas operações o farão perder o controle do tempo.

Os números **PEEK 23674** , **PEEK 23673** e **PEEK 23672** estão guardados dentro do computador e são usados para contar em unidades de 1/50 de segundo. Cada um deles está entre Ø e 255 e vão aumentando gradualmente passando por todos os números entre Ø e 255; depois de atingirem 255, eles voltam a Ø.

Aquele que é incrementado mais vezes é o **PEEK 23672**. Por cada 1/5Ø de segundo que passa, é aumentado de 1. Quando este chega a 255, o próximo incremento leva-o a zero e ao mesmo tempo empurra o **PEEK 23673**, 1 para cima. Quando (uma vez em cada 256/5Ø segundos) o **PEEK 23673** é empurrado de 255 para Ø, por sua vez empurra o **PEEK 23674** um para cima. Isto deve ser suficiente para explicar como é que a expressão acima mencionada funciona.

Agora pense cuidadosamente: suponha que os nossos três números são zero (para **PEEK 23674**), 255 (para **PEEK 23673**) e 255 (para o **PEEK 23672**). Isto significa que passaram cerca de 21 minutos desde que se ligou o computador - a nossa expressão deve dar

(65536*Ø+256*255+255)/5Ø=131Ø.7

Mas existe aqui um perigo escondido. Da próxima vez que houver uma contagem de 1/5Ø do segundo, os três números mudarão para 1, Ø e Ø. Muitas vezes isto irá acontecer enquanto ainda estivermos a calcular a expressão: o computador calcularia **PEEK 23674** como Ø, mas a seguir mudaria os outros dois para zero antes de poder ver qual o seu conteúdo. A resposta seria então

(65536*Ø+256*Ø+Ø)/5Ø=Ø

o que está irremediavelmente errado.

Uma regra simples para evitar este problema é calcular a expressão duas vezes seguidas e aproveitar o maior valor.

Se olhar cuidadosamente para o programa acima descrito pode ver que isto é feito implicitamente.

Aqui está um truque para aplicar a regra. Defina as funções:

```
10 DEF FN m(x,y)=(x+y+ABS (x-y))/2: REM o maior entre x e y
20 DEF FN u()=(65536*PEEK*23674+256*PEEK 23673+PEEK
23672)/50: REM tempo, que pode estar errado
30 DEF FN t()=FN m(FN u()): REM tempo, certo
```

Pode-se modificar os três números de contagem por forma a que eles dêm o tempo real em vez do tempo desde que o computador foi ligado. Por exemplo, para acertar o relógio pelas 1Ø.ØØ am, terá de calcular que este corresponde a 1Ø*6Ø*6Ø*5Ø=18ØØØØØ em unidades de 1/5Ø do segundo e que

18ØØØØØ=65536*27+256*119+64

Para colocar os três números em 27, 119 e 64 tem de fazer

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

Em países que tenham frequências de 6Ø HERTZ estes programas têm que ser corrigidos, passando de '5Ø' para '6Ø' nos sítios apropriados.

A função INKEY$, a qual não tem argumento, lê o teclado. Se estiver a carregar numa tecla (ou numa tecla de SHIFT e outra qualquer) então o resultado é o caracter que a tecla daria no modo L; de outra forma o resultado é uma cadeia vazia.

Tente este programa que funciona como uma máquina de escrever.

```
10 IF INKEY$ <>"" THEN GO TO 10
20 IF INKEY$ ="" THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 10
```

Aqui a linha 1Ø espera que levante o dedo do teclado e a linha 2Ø espera que carregue numa nova tecla.

Lembre-se que ao contrário do INPUT, a instrução INKEY$ não espera por si. Por isso não tem que carregar em ENTER, mas por outro lado se não carregar em nenhuma tecla terá perdido a oportunidade.

Exercícios

1. O que acontecerá se faltar a linha 1Ø no programa da máquina de escrever?

2. Outro modo de usar a instrução **INKEY$** é em conjunção com a instrução **PAUSE**, como neste programa alternativo para a máquina de escrever:

```
10 PAUSE 0
20 PRINT INKEY$;
30 GO TO 10
```

Para fazer com que este programa funcione porque é que é essencial que a pausa não termine se o encontrar já a carregar numa tecla quando começar?

3. Adaptar o programa do relógio de modo a que ele mostre também o ponteiro dos minutos e das horas, desenhando-os de minuto a minuto. Se se sente ambicioso, faça as coisas de modo a que de quarto em quarto de hora ele execute uma espécie de demonstração - pode produzir um sino com instruções **BEEP**. (ver capítulo seguinte).

4. (Para sádicos).Exeperimente este:

```
10 IF INKEY$ ="" THEN GO TO 10
20 PRINT AT 11,14;"OUCH!"
30 IF INKEY$ <>"" THEN GO TO 30
40 PRINT AT 11,14"  "
50 GO TO 10
```

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

19

CAPÍTULO 19

BEEP

Sumário

BEEP

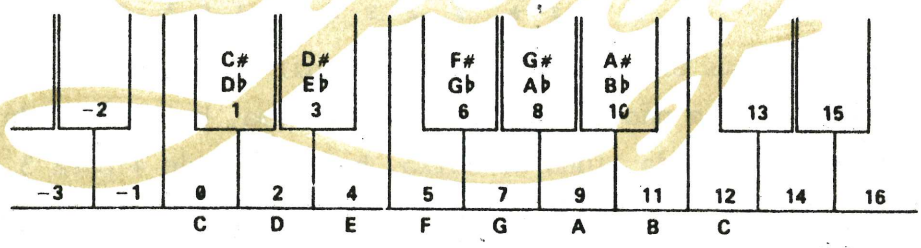
Se ainda não descobriu que o ZX Spectrum tem um altifalante incorporado é melhor ler o livro introdutório antes de continuar.

O altifalante é accionado usando a instrução **BEEP**.

BEEP duração, frequência

onde, como de costume, 'duração' e 'frequência' representam qualquer expressão numérica. A duração é dada em segundos e a frequência é dada em semi-tons acima do Dó medio, usando números negativos para notas abaixo do Dó médio.

Aqui está um diagrama que mostra os valores de frequência para todas as notas contidas numa oitava de piano:



Para obter notas mais elevadas ou mais baixas tem de se adicionar ou subtrair 12 a cada oitava conforme se quer subir ou descer.

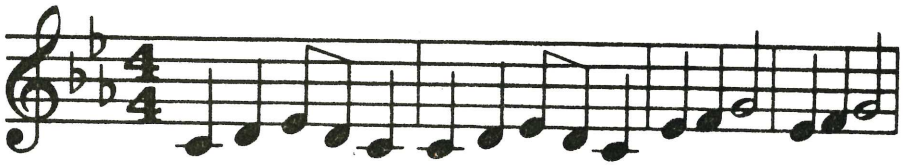
Se tiver um piano à sua frente quando estiver a programar uma melodia, este diagrama será provavelmente suficiente para obter os valores de frequência. Se, no entanto, estiver a transcrever directamente de uma pauta, então sugerimos que desenhe um diagrama de cada compasso com o valor da frequência escrito sobre cada linha e espaço, tomando em conta a tecla.

Por exemplo, escreva:

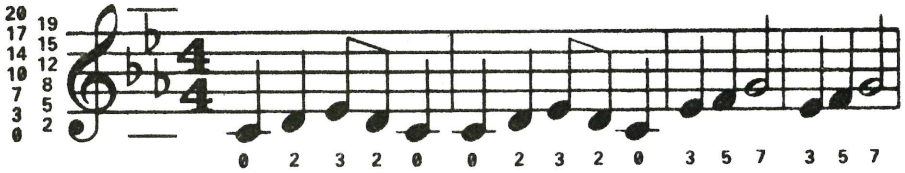
```
10 PRINT "Frere Gustav"
20 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
30 BEEP 1,0: BEEP 1,2: BEEP .5,3: BEEP .5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
70 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
80 BEEP 1,0: BEEP 1,-5: BEEP 2,0
90 BEEP 1,0: BEEP 1,-5: BEEP 2,0
```

Quando correr este programa deve ouvir a marcha fúnebre da 1a. sinfonia de Mahler, a parte onde os duendes enterram o cavaleiro americano.

Suponha, por exemplo, que a sua melodia começa na clave de Dó menor, como a melodia de Mahler que apresentámos acima. O começo está como se indica:



e pode escrever os valores da frequências assim:



Colocámos duas linhas extra, para podermos medir bem. Reparem como o Mi bemol na clave afecta não só o Mi na zona de cima, baixando-o de 16 para 15, mas também o MI da linha de baixo, baixando-o de 4 para 3. Agora deve ser extremamente fácil calcular o valor da frequência de cada nota na pauta.

Se quiser modificar a clave do trecho, o melhor que tem de fazer é criar uma variável chamada nota e inserir nota+ antes de cada valor de frequência: assim a segunda linha transforma-se em

```
20 BEEP 1,nota+0: BEEP 1, nota+2: BEEP .5, nota+3: BEEP .5,nota+2:
   BEEP 1,nota+0
```

Antes de correr o programa tem de dar à variável nota o valor adequado - 0 para Dó menor, 2 para Ré menor, 12 para Dó menor uma oitava acima e assim por diante. Pode afinar o computador com outro instrumento ajustando o valor de nota, usando valores fraccionários.

Também tem de descobrir a duração de todas as notas. Uma vez que esta melodia era muito lenta, demos um segundo para uma semínima e baseamo--nos nisso para as outras, metade de segundo para uma colcheia e assim por diante.

Mais flexivel será usar uma variável chamada semínima para guardar a duração de uma semínima e especificar a duração das outras em termos desta. Então a linha 2Ø ficaria

```
20 BEEP semínima,nota+0: BEEP semínima,nota+2:
   BEEP semínima/2,nota+3: BEEP semínima/2, nota+2:
   BEEP semínima, nota+0
```

(provavelmente vai querer dar nomes mais curtos a semínima e a nota).

Dando a semínima valores apropriados, podemos facilmente variar a velocidade do trecho.

Lembre-se que dado que só há um altifalante no computador não pode tocar mais do que uma nota de cada vez, por isso está restringido a músicas sem harmonia. Se quiser mais terá que cantar você.

Experimente programar músicas você mesmo - comece com coisas simples como 'Os Três Ratos Cegos'. Se não tem um piano nem música escrita, agarre-se a um instrumento simples como uma flauta ou uma harmónica e invente músicas com isso. Pode fazer um gráfico com os valores das frequências que pode obter com esse instrumento.
Escreva:

```
FOR n=0 TO 1000: BEEP .5,n: NEXT n
```

Este programa toca as notas mais altas que pode, e depois pára com a mensagem de erro B integer out of range. Pode pedir para ver o n para descobrir até onde é que se pode ir.

Experimente a mesma coisa, mas indo em direcção às notas baixas. As notas mais baixas parecem estalidos; de facto as notas mais altas também são feitas de estalidos, mas mais depressa, de modo que o ouvido não os distingue.

Somente a gama média de notas consegue tocar-se decentmente; as notas baixas parecem-se muito com estalidos e as notas altas são muito esganiçadas e oscilam um pouco.

Introduza esta linha de programa:

```
10 BEEP .5,0: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7:
   BEEP .7,9: BEEP .5,11: BEEP .5,12: STOP
```

Este programa toca a escala de Dó maior, que usa todas as notas brancas do piano desde o Dó médio até ao Dó seguinte. O modo como esta escala é afinada é exactamente como a do piano, e é chamado de temperamento certo porque o intervalo de frequência de um semitom é constante até ao fim da escala. Um violinista, no entanto tocaria a escala de forma algo diferente, ajustando todas as notas por forma a que elas soassem mais agradáveis ao ouvido. Ele consegue fazer isto movendo o dedo muito levemente para cima e para baixo na corda, de uma forma que o pianista não consegue.

A escala natural, que é a que o violinista toca, sai assim:

```
20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.86: BEEP .5,4.98:
   BEEP .5,7.02: BEEP .5,8.84: BEEP .5,10.88: BEEP .5,12: STOP
```

Pode ser ou não capaz de detectar qualquer diferença entre estas duas; algumas pessoas conseguem. A primeira dieferença notável é que a terceira nota é ligeiramente mais baixa na escala natural. Se fôr verdadeiramente um perfeccionista pode gostar de programar as suas melodias para usar esta escala natural em vez da de temperamento certo. A desvantagem é que, apesar de ela servir muito bem na clave de Dó, nas outras claves funciona menos bem - todas as claves têm a sua própria escala natural - e em algumas claves funciona mesmo muito mal. A escala de temperamento certo é só ligeiramente diferente e funciona igualmente bem em todas as claves.

Isto é um problema menor no computador, claro, porque pode-se usar o truque de adicionar a variável nota.

Alguns tipos de música - nomeadamente a música indiana - usam intervalos de frequência menores do que um semitom. Pode programar isto usando a instrução **BEEP** sem qualquer problema; por exemplo um quarto de tom acima do Dó medio tem o valor de .5.

Pode fazer com que o teclado emita uma nota em vez de dar um estalido se fizer.

POKE 23609,255:55

O segundo número nesta instrução determina a duração do som (experimente vários valores entre Ø e 255). Quando é Ø, o som é tão curto que parece um suave estalido.

Se estiver interessado em fazer mais com o som do Spectrum, como ouvir o som que o **BEEP** faz noutra coisa que não seja o altifalante interno, vai ver que o sinal está presente tanto na saída marcada com 'MIC' como na marcada com 'EAR'. Na saída 'EAR' estará mais alto mas de resto são absolutamente iguais. Pode usar isto para ligar uns auscultadores ou um cabo ao amplificador. Isto não desliga o altifalante interno. Se quiser que o volume de som saia muito alto o melhor é ligar a um amplificador - a saída 'MIC' deve dar um volume adequado - ou pode gravar o som em fita e fazer com que o Spectrum toque consigo mesmo.

Não fará qualquer mal ao Spectrum ainda que faça um curto-circuito nas saídas de 'MIC' e 'EAR', por isso experimente para ver qual é que dá a melhor saída para o que quer fazer.

Exercício

1. Re-escreva o programa de Mahler no Capítulo 8 de modo a usar um ciclo **FOR** para repetir os compassos.

Programe o computador para que toque não só a marcha fúnebre mas também o resto da sinfonia de Mahler.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

20

CAPÍTULO 20

Armazenamento em fita

Sumário

LOAD, SAVE, VERIFY, MERGE

Os métodos básicos para usar um gravador de cassetes são as instruções para guardar (**SAVE**), carregar (**LOAD**) ou verificar (**VERIFY**) um programa e já foram vistas no livro introdutório. Esta secção deve ser lida e os procedimentos experimentados antes de prosseguir a leitura aqui.

Vimos que a instrução **LOAD** apaga o programa antigo bem como as variáveis do computador antes de carregar o novo da fita; há uma outra instrução, a instrução **MERGE**, que o não faz. A instrução **MERGE** só apaga uma linha ou uma variável de um programa antigo se tiver de o fazer porque o novo tem uma linha com o mesmo numero ou uma variável com o mesmo nome. Introduza de novo o programa de 'dados' do capítulo 11 e guarde-o em fita com o nome 'dados'.
Agora introduza e execute o seguinte:

```
1 PRINT 1
2 PRINT 2
10 PRINT 10
20 LET x=20
```

e depois continue como se fosse fazer uma verificação mas substituindo **VERIFY** "dados" por

MERGE "dados"

Se listar o programa pode ver que as linhas 1 e 2 sobreviveram mas que as linhas 1Ø e 2Ø foram substituidas pelas do programa dados. x também sobreviveu (experimente com **PRINT** x).

Agora já vimos formas simples das quatro instruções que são usadas com gravadores de cassetes:

SAVE - grava o programa e as variáveis em cassete.

VERIFY - verifica o programa e as variáveis na cassete comparando-as com as que estão no computador.

LOAD - limpa o computador de todos os seus programas e veriáveis e substitui-os pelos novos que são lidos da cassete.

MERGE - como **LOAD** mas não apaga o programa antigo nem as variáveis

a menos que haja um número de linha ou uma variável igual no programa novo.

Em cada um destes casos a palavra-chave é seguida por uma cadeia: para a instrução **SAVE** essa cadeia fornece o nome que o programa vai ter na fita, enquanto que para as outras três diz ao computador qual o programa que deve procurar. Enquanto está à procura imprime o nome de todos os programas que encontra. Ainda há alguns truques nisto tudo.

Para as instruções **VERIFY**, **LOAD** e **MERGE** pode-se fornecer uma cadeia vazia como nome de procura: então o computador não se incomoda com o nome, mas aceita o primeiro programa que encontra.

Uma variante da instrução **SAVE** toma a forma

SAVE cadeia **LINE** número

Um programa que é guardado usando esta variante é gravado de forma que quando é lido por uma instrução **LOAD** (mas não por um **MERGE**) salta imediatamente para a linha com o número que é dado, executando-se assim a si próprio.

Até aqui os únicos tipos de informação que guardámos em cassete foram os programas juntamente com as suas variáveis. Há ainda dois outros tipos, chamados matrizes e bytes.

As matrizes são tratadas de forma algo diferente:

Podem-se guardar matrizes em fita usando a instrução **DATA** juntamente com a instrução **SAVE** da seguinte forma

SAVE cadeia **DATA** nome de matriz ()

Cadeia é o nome que a informação vai ter na fita e funciona exactamente do mesmo modo que para se guardar um programa ou simplesmente alguns bytes.

O nome de matriz especifica qual a matriz que se quer guardar, de modo que é somente uma letra ou uma letra seguida de $. Lembre-se que tem de colocar os parentesis depois; pode pensar que logicamente não seriam necessários, mas mesmo assim tem de os colocar para facilitar a tarefa do computador.

Seja claro quando aos papéis separados de cadeia e do nome de matriz. Se disser (por exemplo)

SAVE "Fitas" **DATA** b()

então a instrução **SAVE** agarra na matriz b que está no computador e guarda-a na fita sob o nome de "Fitas". Quando escrever

VERIFY "Fitas" **DATA** b()

o computador vai procurar uma matriz numérica guardada em fita sob o nome "Fitas" (quando a encontrar escreve 'Number array:Fitas') e vai compará-la com a matriz b que está no computador.

A instrução

LOAD "Fitas" **DATA** b()

encontra a matriz em fita e depois - se houver espaço no computador - - apaga qualquer matriz que já exista com o nome de b e carrega a nova matriz a partir da fita, chamando-a b.

Não pode usar a instrução **MERGE** com matrizes em fita.

Pode guardar matrizes de cadeias (ou caracteres) exactamente do mesmo modo. Quando o computador estiver à procura e encontrar um destes escreve 'Character array:' seguido do nome. Quando se carrega uma matriz de caracteres, apaga não só qualquer matriz de caracteres com o mesmo nome, mas também qualquer cadeia simples com o mesmo nome.

O armazenamento de bytes é usado para bocados de informação sem qualquer referência a que tipo de informação é usada - pode ser uma imagem de televisão, ou caracteres gráficos definidos pelo utilizador, ou algo que você mesmo fez. Mostra-se isso usando a palavra **CODE** como em

SAVE "imagem" **CODE** 16384,6912

A unidade de armazenamento em memória é o byte (um número entre Ø e 255), e cada byte tem um endereço (que é um número entre Ø e 65535). O primeiro número que se segue à palavra **CODE** é o endereço do primeiro byte que deve ser guardado em fita, e o segundo é o número de bytes que devem ser guardados. No nosso caso, 16384 é o endereço do primeiro byte do ficheiro de imagem (que contém a imagem de televisão) e 6912 é o número de bytes que lá existem, de modo que estamos a guardar uma cópia do écran de televisão - experimente. O nome "imagem" funciona exactamente como os nomes para programas.

Para o voltar a carregar usa-se

LOAD "imagem" **CODE**

Podem-se colocar números depois de **CODE** na forma

LOAD nome **CODE** começo, comprimento

Aqui comprimento é só uma medida de segurança: quando o computador encontrar os bytes na fita com o nome certo, recusar-se-á a carregá--los se encontrar mais do que o que está especificado em comprimento -- uma vez que há obviamente mais dados do que os que eram esperados, facilmente se podia apagar algo que não se pretendia perder. Dá uma mensagem de erro R Tape loading error. Também se pode omitir o comprimento, e então o computador lê os bytes sem se importar com quantos são.

O começo indica o endereço onde o primeiro byte deve ser armazenado - este pode ser diferente do endereço de onde fora retirado, ainda que se forem iguais se possa omitir o começo na instrução **LOAD**.

CODE 16384,6912 é tão importante para guardar e recuperar a imagem que se pode substituir por **SCREEN$** - por exemplo,

SAVE "imagem" **SCREEN$**
LOAD "imagem" **SCREEN$**

Este é um dos raros casos em que a função **VERIFY** não funciona - o **VERIFY** escreve os nomes do que vai encontrando na fita, de modo que quando chegar à verificação a imagem já se terá modificado e a verificação falha. Em todos os outros casos deve ser usado o **VERIFY** sempre que se fizer um **SAVE**.

Abaixo está um sumário completo das quatro instruções que são usadas neste capítulo.

Nome representa qualquer expressão de cadeia, e refere-se ao nome sob o qual a informação é guardada em cassete. Deve consistir em caracteres de impressão ASCII, dos quais somente os primeiros 1Ø são usados.

Há quatro tipos de informação que pode ser guardada em fita: programas e variáveis (conjuntamente), matrizes numéricas, matrizes de caracteres e bytes sozinhos.

Quando as instruções **VERIFY LOAD** e **MERGE** estão a pesquisar a fita em busca de informação com um dado nome e de uma dada espécie, elas imprimem no écran o tipo e o nome de toda a informação que encontram. A especie é mostrada por 'Program:', 'Number array:', 'Character array:' ou 'Bytes:' (respectivamente para programas, matrizes numéricas, matrizes de caracteres e bytes). Se o nome fôr um cadeia vazia estas instruções tomam o primeiro grupo de informação da espécie certa, independentemente do nome

SAVE

Guarda informação em fita sob um dado nome. O Erro F ocorre quando o nome é vazio ou tem mais de 10 caracteres.

A instrução **SAVE** emite sempre uma mensagem Start tape, then press any key (Ligue o gravador e depois carregue em qualquer tecla), e espera que uma tecla seja pressionada antes de guardar alguma coisa.

1. Programa e variáveis:

SAVE nome

é a forma normal.

SAVE nome **LINE** número de linha

guarda o programa de forma que a instrução **LOAD** é imediatamente seguida por

GO TO número de linha

2. Bytes:

SAVE nome **CODE** começo, comprimento

guarda tantos bytes quantos os especificados em comprimento começando

no endereço indicado em começo.

```
SAVE nome SCREEN$
```

é equivalente a

```
SAVE nome CODE 16384,6912
```

e guarda a imagem de televisão.

3. Matrizes:

```
SAVE nome DATA letra ()
```

ou

```
SAVE nome DATA letra$
```

guarda a matriz cujo nome é letra ou letra $ (esta não precisa de ter qualquer relação com o nome).

VERIFY

Verifica a informação que se encontra no gravador contra aquela que já está em memória. Um erro de verificação dá a seguinte mensagem R Tape loading error (erro de leitura da fita).

1. Programa e variáveis:

```
VERIFY nome
```

2. Bytes:

```
VERIFY nome CODE começo, comprimento
```

Se os bytes gravados na fita sob o nome são mais em número que os especificados em comprimento, então obtem-se um erro R. De outra forma faz a verificação dos bytes em memória, começando no endereço indicado em começo, contra os da fita.

```
VERIFY nome CODE começo
```

verifica os bytes com o nome na fita contra os que tem em memória, come

çando no endereço indicado em começo.

VERIFY nome **CODE**

verifica os bytes com o nome na fita contra os que estão na memória começando no endereço a partir do qual os dados da fita foram gravados.

VERIFY nome **SCREEN$**

é equivalente a

VERIFY nome **CODE 16384,6912**

mas é quase certo que dá erro de verificação.

3. Matrizes:

VERIFY nome **DATA** letra

ou

VERIFY nome **DATA** letra **$**

verifica a matriz que tem o nome na fita contra a matriz cujo nome é a letra ou letra$ na memória

LOAD

Carrega a informação a partir da cassete, apagando a informação anterior que se encontrava na memória.

1. Programa e variáveis:

LOAD nome

apaga o programa anterior, bem como as variáveis, e carrega o programa e as variáveis que se encontram sob o nome na cassete; se o programa foi guardado usando **SAVE** nome **LINE** passa automaticamente à execução na linha indicada.

O erro **4** Out of memory (memória esgotada) ocorre se não houver espaço para o novo programa e suas variáveis. Neste caso o programa antigo e respectivas variáveis não é apagado.

2. Bytes:

LOAD nome **CODE** começo, comprimento

Se o número de bytes guardado em fita com o nome é mais do que o indicado em comprimento obtem-se um erro R. De outra forma carrega-os para a memória começando com o endereço indicado em começo, e apagando tudo que lá estava anteriormente.

LOAD nome **CODE** começo

carrega os bytes com o nome indicado a partir da fita para a memória, começando com o endereço começo, e apagando o que lá estava anteriormente.

LOAD nome **CODE**

carrega os bytes que têm o nome indicado na fita para a memória, começando no endereço a partir dos quais eles tinham sido retirados para a fita e apagando os bytes que estavam nessa localização de memória antes.

3. Matrizes:

LOAD nome **DATA** letra ()

ou

LOAD nome **DATA** letra **$** ()

apaga qualquer matriz que já tenha o nome indicado pelo letra ou por letra$ (conforme o caso) e forma uma nova a partir dos dados armazenados em cassete.

O erro **4** Out of memory (memória esgotada) ocorre se não houver espaço para a nova matriz. Nesse caso as matrizes anteriores não são apagadas.

MERGE

Carrega nova informação a partir da cassete sem apagar a anterior informação da memoria.

1. Programa e variáveis:

MERGE nome

junta o programa com o nome indicado com aquele que já está em memória, apagando qualquer linha de programa ou variável do velho programa cujo número de linha ou o nome entre em conflito com as do novo programa.

Error **4** Out of memory ocorre se não houver espaço suficiente em memória para todo o programa antigo e respectivas variáveis bem como para o novo programa e variáveis, que estão a ser carregados da fita.

2. Bytes

Não é possível

3. Martizes:

Não é possível

Exercícios

1. Faça uma cassete na qual o primeiro programa, quando carregado, imprima uma ementa (uma lista dos programas que estão na cassete), lhe peça para escolher um programa e depois o carregue.

2. Vá buscar os caracteres gráficos do capítulo 14 e depois escreva **NEW**: os caracteres sobrevivem a isto. No entanto não sobrevivem se se apagar o computador: se os quiser manter tem de os guardar na fita usando um **SAVE** com **CODE**. O modo mais fácil de guardar todos os caracteres gráficos definidos pelo utilizados é

SAVE "xadrez" **CODE USR** "a",21*8

seguido de

VERIFY "xadrez" **CODE**

Este é o sistema de guardar bytes que foi usado para guardar a imagem de televisão. O endereço do primeiro byte a ser guardado é **USR** "a", o endereço do primeiro dos oito bytes que determina o padrão do primeiro caracter gráfico definido pelo utilizados e o número de bytes a ser guardado é 21*8 - oito bytes por cada caracter gráfico.

Para voltar a carregar normalmente utilizaria

LOAD "xadrez" **CODE**

No entanto, se estiver a carregar para um Spectrum com uma quantidade de memória diferente, ou se deslocou os caracteres gráficos definidos pelo utilizador para um endereço diferente (tem de fazer isto deliberadamente usando tecnicas mais avançadas), tem de se ter mais cuidado e usar

LOAD "xadrez" **CODE USR "a"**

A instrução **USR** dá conta do facto de que os caracteres gráficos foram carregados num endereço diferente.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

CAPÍTULO 21

A impressora ZX

Sumário

LPRINT. LLIST. COPY

Nota: nenhulma destas instruções é BASIC normalizado, ainda que a instrução **LPRINT** seja usada em alguns outros computadores.

Se tem uma impressora ZX, também a verá acompanhada de instruções. Este capítulo cobre as instruções BASIC que são necessárias para a fazer funcionar.

As duas primeiras, **LPRINT** e **LLIST**, são exactamente como o **PRINT** e o **LIST**, excepto que são usadas na impressora em vez da televisão. (O L é um acidente histórico. Quando o BASIC foi inventado usava normalmente uma máquina de escrever eléctrica em vez de uma televisão, de modo que **PRINT** queria dizer mesmo imprimir. Se quisesse grande quantidade de saídas usava-se uma impressora de linhas de alta velocidade ligada ao computador, e uma instrução **LPRINT** que queria dizer 'Impressão numa impressora de linhas')**PRINT**.

Experimente este programa, por exemplo

```
10 LPRINT "Este programa".
20 LLIST
30 LPRINT '"imprime o conjunto de caracteres."'
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

A terceira instrução, **COPY**, imprime uma cópia do écran de televisão. Por exemplo escreva **LIST** para obter uma listagem no écran do programa acima, e escreva

```
COPY
```

Repare que a instrução **COPY** não funciona com uma listagem que o computador faça sair automaticamente, porque essa é apagada quando uma ordem é executada. Tem de usar **LIST** primeiro, ou usar **LLIST** e esquecer-se do **COPY**.

Pode sempre parar a impressora quando ela está a escrever carregando na tecla **BREAK** (**CAPS SHIFT** e **SPACE**).

Se executar estas instruções sem a impressora ligada, perde-se toda a informação de saída e o computador continua para a instrução seguinte.

Experimente isto:

```
10 FOR n=31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$ (CODE "0"+n);
30 NEXT n
```

Verá um padrão de caracteres a descer diagonalmente a partir do canto superior direito até que chega ao fim do écran, altura em que o programa lhe pergunta se quer que ele faça scroll.

Agora mude **AT** 31-**n**,**n** na linha 2Ø para **TAB n**. O programa terá exactamente o mesmo efeito que antes.

Agora modifique o **PRINT** na linha 2Ø para **LPRINT**. Desta vez vai obter somente uma linha direita de símbolos. O motivo da diferença é que a saída para **LPRINT** não é impressa de imediato, mas organiza numa memória tampão uma imagem com o comprimento de uma linha do que o computador vai mandar escrever quando acabar. A impressão tem lugar quando

(i) a memória tampão está cheia

(ii) depois de uma instrução **LPRINT** que não acaba com vírgula ou ponto-e-vírgula

(iii) quando a vírgula, o apóstrofo ou o elemento **TAB** requerem uma nova linha ou

(iv) quando acaba o programa, se houver ainda alguma coisa por escrever.

(iii) explica porque é que o nosso programa com o **TAB** funciona assim. Quanto ao elemento **AT**, o número de linha é ignorado e a posição de impressão na impressora (tal como a posição de impressão, mas na impressora em vez de televisão) é mudada para o número de coluna. Um elemento **AT** nunca pode obrigar uma linha a ser enviada para a impressora.

Exercício

1. Faça um gráfico na impressora da função **SIN** correndo o programa do Capítulo 19 e depois usando a instrução **COPY**.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

CAPÍTULO 22

Outros Equipamentos

Há outros tipos de equipamento que poderá ligar ao Spectrum.

O ZX Microdrive é um aparelho de armazenamento de dados massivo, e é muito mais flexivel no modo como pode ser usado do que um gravador de cassetes. Vai trabalhar não só com as instruções **SAVE**, **VERIFY** **LOAD** e **MERGE**, mas também como **PRINT**, **LIST**, **INPUT** e **INKEY$**.

A rede que se pode criar vai permitir ligar vários Spectrum de forma a que eles possam falar uns com os outros - uma das utilidades disto é que só vai precisa de uma Microdrive para servir vários computadores.

A interface RS232 é uma ligação padrão que lhe permite ligar um Spectrum com teclados, impressoras, computadores e várias outras máquinas ainda que não tenham sido desenhadas especialmente para o Spectrum.

Tudo isto irá usar ainda algumas palavras-chave extra que já estão no teclado mas não podem ser usadas sem os acessórios: são **OPEN#**, **CLOSE#**, **MOVE**, **ERASE**, **CAT** e **FORMAT**.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

CAPÍTULO 23

IN e OUT

Sumário

OUT
IN

O processador pode ler e (pelo menos com a RAM) escrever na memória usando as instruções **PEEK** e **POKE**. O processador em si, na verdade, não se preocupa com que a memória seja ROM ou RAM, ou outra coisa qualquer; só sabe que há 65536 endereços de memória, e que ele pode ler um byte de cada um desses endereços (mesmo que seja um disparate), e escrever um byte em cada um deles (mesmo que se perca). De forma completamente análoga há 65536 objectos a que se chamam I/O ports (que representa Input/Output ports - Portas de Entrada/Saída). Estas são usados pelo processador para comunicar com coisas como o teclado ou a impressora, e podem ser controladas a partir das instruções de BASIC usando a função **IN** e a instrução **OUT**.

IN é uma função como **PEEK**.

IN endereço

Tem um argumento, o endereço da port, e o seu resultado é um byte que é lido desse port. **OUT** é uma instrução como **POKE**

OUT endereço, valor

escreve o valor dado para o port com o endereço dado. Como é que o endereço é interpretado depende muito do resto do computador; muitas vezes endereços diferentes significam o mesmo. No Spectrum o melhor é imaginar que o endereço está escrito em binário, porque os bits individuais tendem a trabalhar independentemente.
Há 16 bits, que vamos chamar de (Usando E para endereço)

E15,E14,E13,E12,E11,E1Ø...,E2,E1,EØ

Aqui EØ é o primeiro bit, E1 o segundo (bit dos dois), E2 é o terceiro (bit dos 4) e assim por diante. Os bits EØ, E1, E2, E3 e E4 são os mais importantes. São normalmente 1, mas se algum deles fôr Ø, isto diz ao computador algo específico. O computador não pode tratar

mais que uma coisa de cada vez, de modo que nunca deve ser zero mais do um destes cinco bits. Os bits E5, E6 e E7 são ignorados, de modo que se fôr um génio da electrónica pode usá-los como quiser. Os melhores endereços para usar são aqueles que são 1 menos que um múltiplo de 32, de modo que EØ...E4 são todos 1. Os bits E8 e E9 são por vezes usados para dar informação extra.

O byte que é escrito ou lido tem 8 bits, e estes são muitas vezes referidos como sendo (usando D para dados) D7, D6, D5... D1, DØ. Aqui está uma lista dos endereços de port que são usados.

Há um conjunto de endereços de entrada que lêm o telcado e também a saída 'EAR'. O teclado está dividido em 8 meias filas de 5 teclas cada uma.

IN 65278 lê a meia fila de CAPS SHIFT a V
IN 65022 lê a meia fila de A a G
IN 64510 lê a meia fila de Q a T
IN 63486 lê a meia fila de 1 a 5
IN 61438 lê a meia fila de 0 a 6
IN 57342 lê a meia fila de P a 7
IN 49150 lê a meia fila de ENTER a H
IN 32766 lê a meia fila de SPACE a B

(Estes endereços são 254+256*(255-2↑n) quando n vai de Ø a 7.)

No byte que é lido os bits DØ e D4 representam as cinco teclas numa dada meia fila - DØ para a tecla de fora, D4 para a que está mais próxima do centro. O bit é Ø se a tecla foi premiada, 1 se não foi. D6 é o valor na entrada 'EAR'.

O endereço de port 254, quando funcionando como saída, controla o altifalante (D4) e a saída de MIC (D3), e também escolhe as cores dos bordos do écran (D2, D1 e DØ).

O endereço de port 251 opera a impressora, tanto na leitura como na escrita: a leitura detecta se a impressora já está pronta para mais, e a escrita envia o que deve ser impresso.

Os endereços de port 254, 247 e 239 são usados para os equipamentos extra mencionados no Capítulo 22.

Execute este programa

```
10 FOR n=0 TO 7 REM número de meia fila
```

```
20 LET a=254-256*(255-2 * n)
30 PRINT AT 0,0, IN a: GO TO 30
```

e pode brincar carregando em teclas. Quando se fartar de cada meia fila, carregue em BREAK e depois escreva

```
NEXT n
```

LANDRY Eng.°s Consultores, LDA.
R. Tomás da Anunciação 53 A Tel. 66 35 38
Telf. 68 13 44/68 12 43, 68 48 27 Telex 43436

CAPÍTULO 24

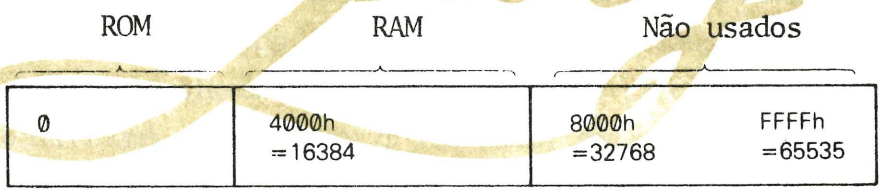
A memória

Sumário

CLEAR

No fundo do computador tudo está guardado em bytes, isto é, números entre Ø e 255. Pode-se pensar que se guardou o preço da lã ou a morada do seu fornecedor de fertilizantes, mas tudo isso foi convertido num conjunto de bytes, e bytes é a única coisa que o computador vê.

Cada local onde se pode armazenar um byte tem um endereço, que é um número entre Ø e FFFFh (de modo que um endereço pode ser guardado em dois bytes), de forma que se pode pensar na memória como uma longa fila de caixas numeradas, cada uma delas contendo um byte. Nem todas as caixas são iguais. Na máquina de 16 K RAM as caixas entre 8ØØØh e FFFFh pura e simplesmente não estão lá. As caixas entre 4ØØØh e 7FFFh são caixas RAM, o que quer dizer que podemos abrir a tampa e alterar-lhes o conteúdo, e aquelas entre Ø e 3FFFh são caixas ROM, que têm uma parede de vidro que não pode ser aberta. Tem de se ler o que lá foi colocado quando computador foi feito.



Para inspeccionar o conteúdo de cada caixa, usamos a função **PEEK**: o seu argumento é um endereço de uma caixa, e o resultado é o seu conteúdo. Por exemplo, este programa imprime os primeiros 21 bytes da ROM (e os seus endereços):

```
10 PRINT "Endereço"; TAB 8; "Byte"
20 FOR a=0 TO 20Ø
30 PRINT a; TAB 8; PEEK a
40 NEXT a
```

Todos estes bytes provavelmente não lhe dirão nada, mas a placa do processador compreende-os como sendo instruções que lhe dizem o que deve fazer.

Para modificar o conteúdo de uma caixa (se fôr RAM), usamos a instrução **POKE**. Tem a forma

POKE endereço, novo conteúdo

onde o 'endereço' e 'novo conteúdo' representam expressões numéricas. Por exemplo, se se disser

```
POKE 31000,57
```

o byte que tem como endereço 31ØØØ receberá o valor 57. Escreva

```
PRINT PEEK 31000
```

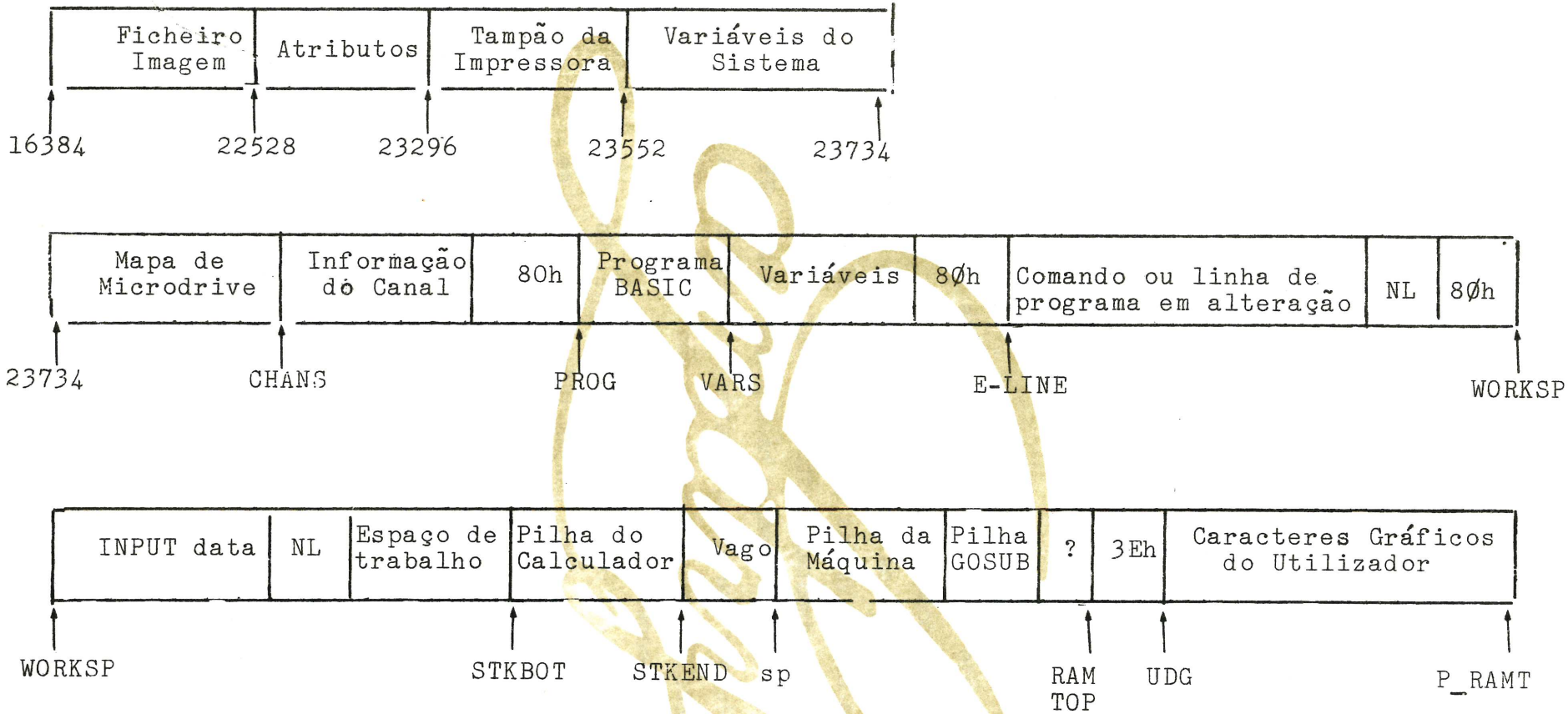
para provar que isto aconteceu. (Experimente introduzir outros valores para ver que não há batota.) o novo valor tem de estar compreendido entre -255 e +255, e se fôr negativo então ser-lhe-á adicionado 256.

A possibilidade de introduzir novos valores na memória dá-lhe um poder enorme sobre o computador se souber como o usar; e imensas capacidades de destruição se não souber. É muito fácil, introduzindo um valor errado na morada errada, perder grandes programas que lhe levaram muito tempo a introduzir. Felizmente não afectará o seu computador permanentemente.

Agora vamos dar uma visão mais detalhada de como a RAM é usada, mas não se incomede a ler isto a menos que esteja interessado.

A memória está dividida em áreas diferentes (mostrada no diagrama grande) para armazenar espécies diferentes de informação. As áreas só têm o tamanho necessário para a informação que vão conter, e se inserir mais alguma coisa num dado ponto (por exemplo se adicionar uma linha de programa ou uma variável) cria-se espaço empurrando tudo o que está acima desse ponto. Inversamente se se apagar informação, então tudo é puxado para baixo.

O ficheiro de imagem guarda a imagem da televisão. O seu arranjo é muito peculiar de modo que provavelmente vai querer ver o que lá está dentro e alterar-lhe o conteúdo. Cada posição de caracter no écran tem um quadrado de 8 por 8 pontos e cada ponto pode ser ou Ø (papel) ou 1 (tinta): e usando uma notação binária podemos guardar esse padrão em 8 bytes, um para cada fila. No entanto esses 8 bytes não estão juntos. As filas correspondentes nos 32 caracteres de uma única linha são guardadas juntas como um conjunto de 32 bytes, porque


Ficheiro Imagem
Atributos
Tampão da Impressora
Variáveis do Sistema
16384
22528
23296
23552
23734
Mapa de Microdrive
Informação do Canal
80h
Programa BASIC
Variáveis
8Øh
Comando ou linha de programa em alteração
NL
8Øh
23734
CHANS
PROG
VARS
E-LINE
WORKSP
INPUT data
NL
Espaço de trabalho
Pilha do Calculador
Vago
Pilha da Máquina
Pilha GOSUB
?
3Eh
Caracteres Gráficos do Utilizador
WORKSP
STKBOT
STKEND
sp
RAM TOP
UDG
P_RAMT

isto é o que o feixe de electrões na televisão precisa quando varre do lado esquerdo do écran para o outro. Uma vez que a imagem completa tem 24 linhas de 8 filas cada uma, pode-se esperar que haja um total de 172 filas guardadas por ordem, uma após a outra; mas engana-se se pensar isto. Primeiro vêm a fila de cima, das linhas Ø a 7, depois a fila seguinte das linhas de Ø a 7 e assim por diante até à última fila das linhas de Ø a 7; depois o mesmo para as linhas de 8 a 15; e depois o mesmo para as linhas de 16 a 23. O resultado de tudo isto é que se estava habituado a um computador que usasse **PEEK** e **POKE** para alterar a imagem do écran, terá de começar a usar **SCREEN$** e **PRINT AT** em vez disso, ou **PLOT** e **POINT**.

Os atributos, são as cores e outras coisas assim para cada posição de caracter, usando o formato do **ATTR**. Estes atributos são guardados linha a linha na ordem que se pode esperar.

O tampão da impressora armazena os caracteres destinados à impressora.

As variáveis do sistema contêm várias informações que dizem ao computador qual o estado em que o computador está. São listadas exaustivamente no capítulo seguinte, mas de momento há algumas(chamadas CHANS, PROG, VARS, E LINE e assim por diante) que contêm os endereços das fronteiras das várias áreas de memória. Estas não são variáveis BASIC e os seus nomes não serão reconhecidos pelo computador.

Os mapas para Microdrive só são usados com a Microdrive. Normalmente não está lá nada. A informação do canal contem informação sobre os equipamentos de entrada e saída, nomeadamente o teclado (com a parte de baixo do écran), a parte superior do écran e a impressora.

Cada linha de um programa em BASIC tem a forma:

Byte mais significativo

Byte menos significativo

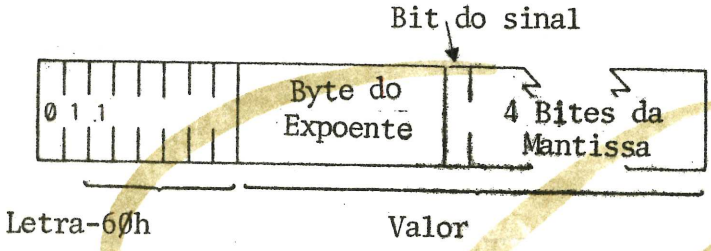
Número de linha do texto | cumprimento | Text | Enter

Repare que, ao contrário de todos os outros casos de números de dois bytes no z8Ø , o número de linha é guardado com o seu byte mais significativo primeiro: isto quer dizer, na ordem pela qual se escrevem.

Uma constante numérica no programa é seguida pela sua forma binária, usando o caracter **CHR$ 14**, seguido de cinco bytes para o número em si.

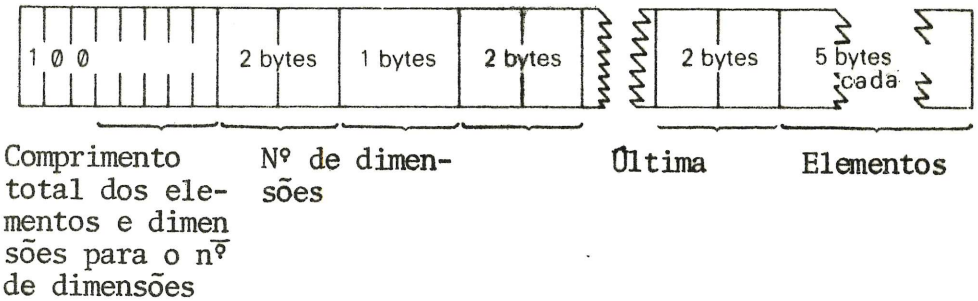
As variáveis têm formatos diferentes de acordo com as suas naturezas diferentes. As letras nos nomes devem ser imaginadas como sempre em letra minúscula.

Números cujo nome seja somente uma letra:



Números cujo nome tenha mais que uma letra:

Matrizes de números:

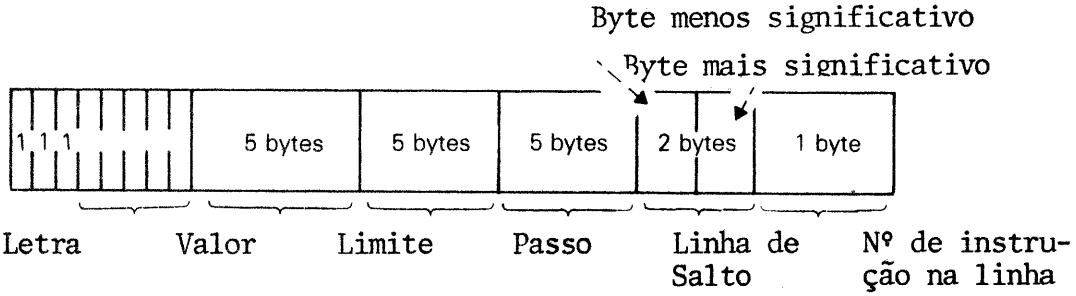


A ordem dos elementos é:
primeiro, os elementos para os quais o primeiro indice é 1
a seguir, os elementos para os quais o primeiro indice é 2
a seguir, os elementos para os quais o primeiro indice é 3
e assim por diante para todos os valores possíveis do indice.

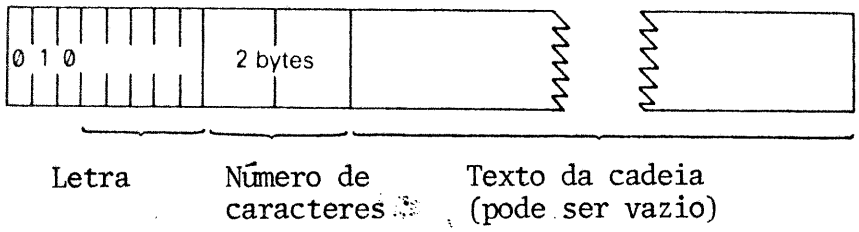
Os elementos que tenham o mesmo primeiro indice são ordenados da mesma forma usando o segundo, e assim por diante até ao último.

Como exemplo, os elementos da matriz de 3 por 6 chamada b no Capítulo 13 são armazenados na ordem b(1,1) b(1,2) b(1,3) b(1,4) b(1,5) b(1,6) b(2,1) b(2,2)... b(2,6) b(3,1) b(3,2)... b(3,6).

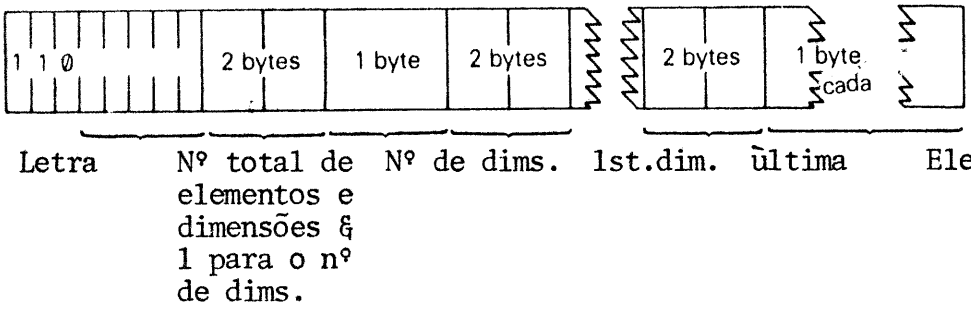
Variável de controle para um ciclo **FOR-NEXT**:



Cadeia:



Matriz de caracteres:



O calculador é a parte do sistema de BASIC que trata da aritmética, e os números com os quais ele opera são mantidos na pilha da calculadora.

A parte sobresselente contém espaço até agora sem utilidade.

A pilha da máquina é a pilha usada pelo processador Z8Ø para guardar endereços de retorno e assim por diante.

A pilha de GOSUB foi mencionada no Capítulo 5.

O byte intitulado de RAMTOP tem o endereço mais elevado usado pelo sistema BASIC. Mesmo o **NEW** que limpa a RAM, só o faz até ali, por isso não altera em nada os caracteres gráficos definidos pelo utilizador. Pode-se alterar o endereço da RAMTOP colocando um número numa instrução

de limpeza:

CLEAR novo RAMTOP

Isto

(i) limpa todas as variáveis
(ii) limpa o ficheiro de imagem (como o **CLS**)
(iii) repõe a posição de **PLOT** no canto inferior esquerdo
(iv) executa um **RESTORE**
(v) limpa a pilha de **GOSUB** e coloca-a no novo RAMTOP - assegurando-se de que esta fica entre a pilha do calculador e o fim físico da RAM; de outra forma deixa o RAMTOP como estava.

A instrução **RUN** também executa um **CLEAR**, ainda que nunca modifique o RAMTOP.

Usar um **CLEAR** desta forma pode mover a RAMTOP para cima para arranjar mais espaço para o BASIC apagando os caracteres gráficos definidos pelo utilizador, ou pode movê-la para baixo para preservar mais RAM das instruções **NEW**.

Escreva **NEW** e depois **CLEAR 23800** para ter uma ideia do que acontece quando a máquina se enche.

Uma das primeiras coisas que vai reparar se começar a introduzir um programa é que depois de algum tempo o computador deixa de aceitar mais linhas e começa a zumbir. Isto significa que o computador está completamente cheio e você vai ter de o limpar um pouco. Também há duas mensagens de erro mais ou menos com o mesmo sentido, **4** Memory full (memória cheia) e **G No** room for line (não há espaço para a linha).

O zumbido também ocorre quando introduz uma linha com mais de 23 linhas - então o que você está a escrever não está a ser ignorado, ainda que não o possa ver; mas o zumbido soa para o desencorajar de continuar.

Pode ajustar o intervalo do zumbido colocando um número no endereço 23608 O intervalo normal tem o número 64.

Qualquer número (excepto Ø) pode ser escrito univocamente como

$$\pm m \times 2^e$$

onde $\pm$ é o sinal,
m é a mantissa, e está entre 1/2 e 1 (sem chegar a 1),
e e é o expoente, um número inteiro (que pode ser negativo).

Suponha que escreve m na escala binária. Dado que é uma fracção, terá um ponto binário (como o ponto decimal-ou vírgula na decimal): de modo que em binário
metade se escreve como .1
um quarto é .Ø1
três quatros se escreve .11
um décimo é escrito como ØØØ11ØØ11ØØ11ØØ11ØØ11... e assim por diante. Com o nosso número m, dado que ele é menor que 1, não há nenhum bit antes do ponto binário, e dado que é no minimo 1/2, o bit imediatamente depois do ponto binário é 1.

Para guardar o número no computador usamos cinco bytes, da seguinte forma:

(i) escrevemos os primeiros oito bits da mantissa no segundo byte (sabemos que o primeiro bit é 1), os segundos oito bits no terceiro byte, e os terceiros oito bits no quarto byte e os quartos oito bits no quinto byte.
(ii) substituimos o primeiro bit do segundo byte - que sabemos que é 1 - pelo sinal: Ø para mais e 1 para menos,
(iii) escrevemos o expoente +128 no primeiro byte. Por exemplo, suponhamos que o nosso número é $^{1}/_{1Ø}$

$$^{1}/_{1Ø} = 4/5 \times 2^{-3}$$

Assim a mantissa m é .11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ11ØØ em binário (uma vez que o trigésimo terceiro bit é 1, arredondamos o trigésimo segundo de Ø para 1), e o expoente é -3. Aplicando as nossas três regras temos os cinco bytes

Zero para mostrar sinal +

Ø111 1101	Ø100 1100	1100 1100	1100 1100	1100 1101

-3+128 a mantissa E 4/5 excepto. o primeiro byte que era 1

Há um modo alternativo de guardar números inteiros entre -65535 e +65535:

(i) o primeiro byte é Ø
(ii) o segundo byte é Ø para um número positivo e FFh para um negativo
(iii) o terceiro e quarto bytes são o menos e mais significativo byte do número (ou o número +131Ø72 se fôr negativo),
(iv) o quinto byte é zero.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

25

CAPÍTULO 25

As variáveis do sistema

Os bytes de memória entre 23552 e 23733 são guardados para uso específico do sistema. Pode-se espreitar para dentro deles para descobrir várias coisas sobre o sistema, e alguns deles podem ter grande utilidade quando se lhes altera o valor.

A estas variáveis chamam-se variáveis do sistema, e têm nomes, que não devem ser confundidos com as variáveis usadas em BASIC. O computador não reconhece os nomes como referidos às variáveis do sistema, e estes nomes só são dados como mnemónicas para nós humanos.

As abreviaturas na primeira coluna têm o seguinte significado:

X As variáveis não devem ser sujeitas a **POKE** porque o sistema pode ficar inoperacional.

N Fazer um **POKE** a esta variável não terá efeito durador.

O número na primeira coluna é o número de bytes dessa variável. Se forem dois bytes o primeiro é o menos significativo - o contrário do que se podia esperar. Por isso para introdupir um valor v numa variável de dois bytes com endereço n faça

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

e para ver o conteúdo use a expressão

```
PEEK n+256*PEEK (n+1)
```

Nota	Endereço	Nome	Conteúdo
N8	23552	KSTATE	Usado na leitura do teclado
N1	2356Ø	LAST K	Guarda a última tecla premida
1	23561	REPDEL	Tempo (em $^1/_{50}$ do segundo ou $^1/_{60}$ do segundo na América do Norte) que uma tecla tem de ser premida para começar a repetir. Começa com 35 mas pode-se alterar.
1	23562	REPPER	Intervalo (em $^1/_{50}$ do segundo ou $^1/_{60}$ na América do Norte) entre

Notas	Endereço	Nome	Conteúdo
			duas repetições sucessivas de uma tecla continuamente premida: inicialmente 5.
N2	23563	DEFADD	Endereço dos argumentos de uma função definida pelo utilizador se se estiver a calcular alguma; senão Ø.
N1	23565	K DATA	Guarda o segundo byte do controle de cor que entra pelo teclado.
N2	23566	TVDATA	Guarda bytes de cor e controles **AT** e **TAB** que vão para a televisão.
X38	23568	STRMS	Endereços dos canais ligados ao sistema.
2	236Ø6	CHARS	256 menos do que o endereço do conjunto de caracteres (que começa num espaço e continua até ao símbolo de copyright). Normalmente em RAM e fazer o CHARS apontar para ele.
1	236Ø8	RASP	Intervalo do zumbido
1	236Ø9	PIP	Intervalo do estalido do teclado
1	2361Ø	ERR NR	1 a menos que o código da mensagem. Começa em 255 (para -1) de modo que **PEEK 23610** dá 255.
X1	23611	FLAGS	Diversos 'flags' de controle do sistema BASIC.
X1	23612	TV FLAG	'flags' associados à televisão
X2	23613	ERR SP	Endereço do elemento da pilha da máquina que deve ser usado como regresso do erro.
N2	23615	LIST SP	Endereço do endereço de retorno da listagem automática.
N1	23617	MODE	Especifica cursor tipo K,L,C,E ou G
2	23618	NEWPPC	Linha para que se deve saltar.
1	2362Ø	NSPPC	Número da instrução na linha para a qual se deve saltar. Fazendo um **POKE** primeiro para NEWPPC e depois NSPPC força um salto para uma instrução específica de uma linha.

Notas	Endereço	Nome	Conteúdo
2	23621	PPC	Número de linha da instrução que está a ser executada
1	23623	SUBPPC	Número dentro da linha da instrução que está a ser executada.
1	23624	BORDCR	Cor do BORDER vezes 8; também contem os atributos para a parte de baixo do écran.
2	23625	E PPC	Número da linha corrente (com cursor de programa).
X2	23627	VARS	Endereço das variáveis.
N2	23629	DEST	Endereço da variável a ser modificada.
X2	23631	CHANS	Endereço dos dados do canal.
X2	23633	CURCHL	Endereço da informação que está a ser usada para entrada/saída.
X2	23635	PROG	Endereço do programa BASIC.
X2	23637	NXTLIN	Endereço da próxima linha de programa.
X2	23639	DATADD	Endereço do indicador do último elemento DATA.
X2	23641	E LINE	Endereço da instrução a ser introduzida pelo teclado.
2	23643	K CUR	Endereço do cursor.
X2	23645	CH ADD	Endereço do próximo caracter a ser interpretado: o caracter depois do argumento em **PEEK**, on **NEWLINE** no fim de uma instrução **POKE**.
2	23647	X PTR	Endereço do caracter depois do ?.
X2	23649	WORKSP	Endereço do espaço de trabalho temporário..
X2	23651	STKBOL	Endereço do fim da pilha do calculador.
X2	23653	STKEND	Endereço do principio do espaço livre.
N1	23655	BREG	Registo b do calculador.
N2	23656	MEM	Endereço da área usada para memória do calculador (geralmente MEMBOT, mas nem sempre).
1	23658	FLAGS2	mais 'flags'.
X1	23659	DF SZ	O número de linhas (incluindo uma em branco) na parte de baixo do écran.
2	23666Ø	S TOP	O número de linhas no início de um programa para listagem automática.

Notas	Endereço	Nome	Conteúdo
2	23662	OLDPPC	Número de linha para a qual o **CONTINUE** deve saltar.
1	23664	OSPPC	Número da instrução dentro da linha para a qual o **CONTINUE** salta.
N1	23665	FLAGX	Várias 'flags'.
N2	23666	STRELEN	Comprimento da variável de cadeia que está a ser modificada.
N2	23668	T ADDR	Endereço do próximo elemento da tabela de sintaxe (de pouca utilidade para si).
2	2367Ø	SEED	O primeiro número para a instrução **RND**. Esta variável é inicializada por **RANDOMIZE**.
3	23672	FRAMES	3 bytes, o menos significativo em primeiro lugar), contador de imagens. Incrementado todos os 20 ms. Ver Capítulo 18.
2	23675	UDG	Endereço do primeiro caracter gráfico definido pelo utilizador. Pode modificar isto, por exemplo, para arranjar espaço tendo menos caracteres definidos.
1	23677	COORDS	Coordenada x do último ponto de um **PLOT**.
1	23678		Coordenada y do último ponto de um **PLOT**.
1	23679	P POSN	Número até 33 para a coluna da posição da impressora.
1	23668Ø	PR CC	Byte menos significativo do endereço da próxima posição de **LPRINT** em que se deve escrever (na memória tampão da impressora).
1	23681		Não usado.
2	23682	ECHO E	Número de coluna (até 33) e número linha (até 24) (na parte de baixo) da memória tampão de entrada.
2	23684	DF CC	Endereço no ficheiro de imagem da posição de impr **PRINT**.
2	23686	DFCCL	Como DF CC para a parte de baixo do écran.

Notas	Endereço	Nome	Conteúdo
X1	23688	S POSN	Número de coluna (até 33) para a posição de **PRINT**.
X1	23689		Número de linha (até 24) da posição de **PRINT**.
X2	2369Ø	SPOSNL	Como S POSN para parte de baixo.
1	23692	SCR CT	Contador para scroll: é sempre 1 a menos do que o número de scrolls que serão feitos antes de parar para perguntar scroll?. Se introduzir aqui continuamente um número maior do que 1 (por exemplo 255), o écran continuará a fazer scroll sem lhe perguntar.
1	23693	ATTR P	Cores corrente permanentes, etc (estabelecidas pelas instruções de cores).
1	23694	MASK P	Usado para cores transparentes, etc. Qualquer bit que seja 1 mostra que o atributo correspondente é tomado não do ATTR P, mas do que já está no écran.
N1	23695	ATTR T	Cores correntes temporárias, etc. (estabelecidas pelos elementos de cores).
N1	23696	MASK T	Como MASK P, mas temporário.
1	23697	P FLAG	Mais 'flags'.
N3Ø	23698	MEMBOT	Área de memória do calculador; usada para guardar números que não podem ser colocadas convenientemente na pilha do calculador.
2	23728		Não usado
2	2373Ø	RAMTOP	Endereço do último byte da área do sistema BASIC.
2	23732	P-RAMT	Endereço do último byte da RAM física.

Este programa diz-lhe os 22 primeiro bytes da área de variáveis:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT n
```

Tente comparar a variável de controle n com as descrições dadas acima.

Agora modifique a linha 2Ø para

```
20 PRINT PEEK (23755+n)
```

Isto diz-lhe quais os 22 primeiros bytes da área de programa. Compare-os com o programa em si.

LANDRY Eng.°s Consultores, LDA.
R, Tomás da Anunciação 53 - A Tel. 66 35 38
Telf. 68 13 44/68 12 43/ 68 48 27 Telex 43436

26

CAPÍTULO 26

Usar código de máquina

Sumário

USR com argumento numérico.

Este capítulo é escrito para aqueles que compreendem o código de máquina Z8Ø, o conjunto de instruções que o processador Z8Ø usa. Se não compreender, mas pretender vir a compreendê-lo, há muitos livros sobre ele. Vai querer arranjar um com um nome parecido com 'Z8Ø Machine code (ou assembley language) for the absolute beginner', e se falar no Spectrum ainda melhor.

Estes programas estão normalmente escritos em linguagem assembly, a qual, ainda que muito simbólica, não é difícil de compreender com um pouco de prática. (Pode ver as instruções da linguagem assembley no Apêndice A.) No entanto para as executar no seu computador precisa de codificar o programa numa sequência de bytes - nesta forma é conhecida por código de máquina.A tradução é normalmente feita pelo próprio computador, usando um programa chamado assembler. O Spectrum não tem qualquer assembler, mas pode obter um em cassete. Se não conseguir terá de traduzir você mesmo, desde que o programa não seja muito comprido.

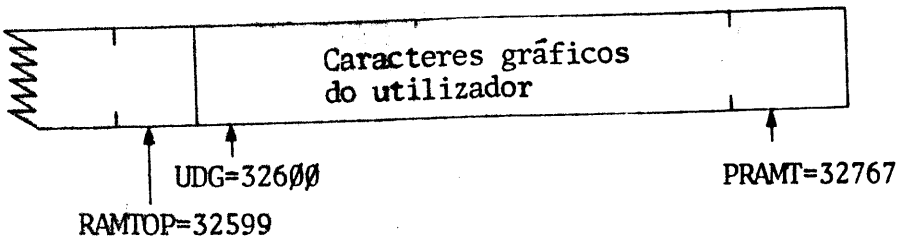
Vamos ver, por exemplo, o programa

```
Ld bc,99
ret
```

que introduz 99 no par de registos bc. Isto traduz-se para os quatro bytes em código de máquina em 1, 99, Ø (para o Ld bc,99) e 2Ø1 (para o ret). (Se procurar o 1 e o 2Ø1 no Apendice A, vai ver que correspondem a Ld bc, NN - onde NN é um número de dois bytes - e ret).

Quando tiver o seu programa em código máquina, o próximo passo é metê-lo no computador. (Um assembler provavelmente faria isto automaticamente). Precisa de se decidir em que localização de memória o vai colocar, e o melhor é arranjar um espaço extra para ele entre a zona de BASIC e os caracteres gráficos definidos pelo utilizador.

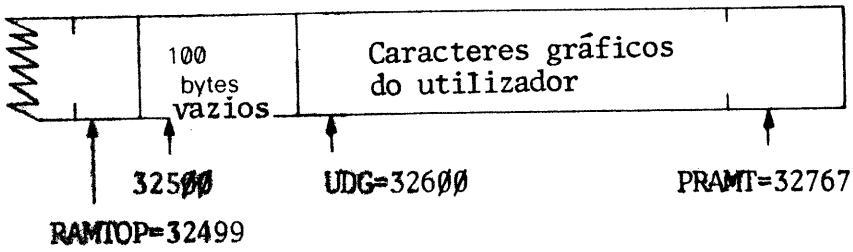
Suponha, por exemplo, que tem um Spectrum de 16K. Para começar a parte final da RAM tem



Se escrever

```
CLEAR 32499
```

isto dar-lhe-à um espaço de 1ØØ (para uma boa medida) bytes começando no endereço 325ØØ.



Para introduzir o programa em código máquina, tem de executar um programa em BASIC que será algo como

```
10 LET a=32500
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(Este programa vai parar com a mensagem **E** Out of **DATA**- acabaram-se os dados - quando tiver acabado de preencher os quatro bytes que você especificou).

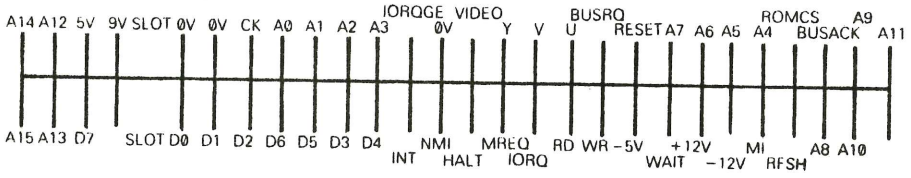
Para executar o programa em código máquina, deve usar a função **USR** - mas desta vez com um argumento numérico, o endereço inicial. O resultado é o valor do registo bc quando voltar do programa em código máquina, por isso se fizer

```
PRINT USR 32500
```

obterá a resposta 99.

O endereço de regresso ao BASIC está numa pilha de forma convencional, por isso o regresso é feito pela instrução ret do Z8Ø. Não deve usar os registos iy ou i numa rotina em linguagem máquina.

As saídas de controle, de dados e de endereçamento estão todas expostas na parte de trás do Spectrum, de modo que pode fazer com ele quase tudo o que faria com um Z8Ø. No entanto, por vezes, a máquina que constitui o Spectrum pode intrometer-se. Aqui está um diagrama das ligações expostas na parte de trás:



Pode guardar o seu programa em código máquina facilmente com

SAVE "um nome qualquer" **CODE 32500,4**

Vendo bem não é possível guardá-lo de forma a que quando se carregar ele se corra automaticamente, mas pode-se obviar a isso usando um programa em BASIC.

```
10 LOAD "" CODE 32500,4
20 PRINT USR 32500
```

Primeiro faça

SAVE "um nome qualquer" **LINE**

e depois

SAVE "xxxx" **CODE 32500,4**
LOAD "um nome qualquer"

que carregará automaticamente e correrá o programa em BASIC que por sua vez vai carregar e correr o programa em código máquina.

APENDICE A

O conjunto de caracteres

Este é o conjunto completo dos caracteres do Spectrum, com os códigos em decimal e hexa. Se se imaginar os códigos como sendo as instruções máquina do Z8Ø, então as colunas da direita dão as correspondentes memonicas da linguagem assembly. Como já deve ter percebido se compreender destas coisas, algumas instruções Z8Ø são compostas começando com CBh ou EDh; as duas colunas da direita dão estas instruções.

Código	Caracter	Hexa	Assembler Z8Ø	depois de CB	depois de ED
Ø	Não usados	ØØ	nop	ric b	
1		Ø1	Id bc,NN	ric c	
2		Ø2	Id (bc),a	ric d	
3		Ø3	inc bc	ric e	
4		Ø4	inc b	ric h	
5		Ø5	dec b	ric 1	
6	**PRINT** vírgula	Ø6	Id,b,N	ric (hl)	
7	**EDIT**	Ø7	rica	ric a	
8	cursor para Esquerda	Ø8	ex af,af'	rrc b	
9	cursor para direita	Ø9	add hl,bc	rrc c	
1Ø	cursor em baixo	ØA	Id a,(bc)	rrc d	
11	cursor em cima	ØB	dec bc	rrc e	
12	**DELETE**	ØC	inc c	rrc h	
13	**ENTER**	ØD	dec c	rrc 1	
14	Número	ØE	Id c,N	rrc (hl)	
15	Não usado	ØF	rrca	rrc a	
16	**INK** control	1Ø	djnz DIS	ri b	
17	**PAPER** control	11	Id de,NN	ri c	
18	**FLASH** control	12	Id (de),a	ri d	
19	**BRIGHT** control	13	inc de	ri e	
2Ø	**INVERSE** control	14	inc d	ri h	
21	**OVER** control	15	dec d	ri 1	
22	**AT** control	16	Id d,N	ri (hl)	
23	**TAB** control	17	ria	ri a	
24	Não usado	18	jr DIS	rr b	
25		19	add hl,de	rr c	
26		1A	ld a,(de)	rr d	
27		1B	dec de	rr e	

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
28	Não usados	1C	inc e	rr h	
29		1D	dec e	rr 1	
3Ø		1E	Id e,N	rr (hl)	
31		1F	rra	rr a	
32	Espaço	2Ø	jr nz,DIS	sla b	
33	!	21	Id hl,NN	sla c	
34	"	22	Id (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla 1	
38	&	26	Id h,N	sla (hl)	
39	'	27	daa	sla a	
4Ø	(	28	Jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	Id hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc 1	sra h	
45	-	2D	dec 1	sra 1	
46	.	2E	Id 1,N	sra (hl)	
47	/	2F	cpl	sra a	
48	Ø	3Ø	jr nc,DIS		
49	1	31	Id sp,NN		
5Ø	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	Id (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	Id a,(NN)	srl d	
59	;	3B	dec sp	srl e	
6Ø	<	3C	inc a	srl h	
61	=	3D	dec a	srl 1	
62	>	3E	Id a,N	srl (hl)	
63	?	3F	ccf	srl a	

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
64	@	4Ø	Id b,b	bit Ø,b	in b, (c)
65	A	41	Id b,c	bit Ø,c	out (c),b
66	B	42	Id b,d	bit Ø,d	sbc hl,bc
67	C	43	Id b,e	bit Ø,e	Id (NN),bc
68	D	44	Id b,h	bit Ø,h	neg
69	E	45	Id b,1	bit Ø,1	retn
7Ø	F	46	Id b,(hl)	bit Ø,(hl)	im Ø
71	G	47	Id b,a	bit Ø,a	Id i,a
72	H	48	Id c,b	bit 1,b	in c,(c)
73	I	49	Id c,c	bit 1,c	out (c),c
74	J	4A	Id c,d	bit 1,d	adc hl,bc
75	K	4B	Id c,e	bit 1,e	Id bc,(NN)
76	L	4C	Id c,h	bit 1,h	
77	M	4D	Id c,1	bit 1,1	reti
78	N	4E	Id c,(hl)	bit 1,(hl)	
79	O	4F	Id c,a	bit 1,a	Id r,a
8Ø	P	5Ø	Id d,b	bit 2,b	in d,(c)
81	Q	51	Id d,c	bit 2,c	out (c),d
82	R	52	Id d,d	bit 2,d	sbc hl,de
83	S	53	Id d,e	bit 2,e	Id (NN),de
84	T	54	Id d,h	bit 2,h	
85	U	55	Id d,1	bit 2,1	
86	V	56	Id d,(hl)	bit 2,(hl)	im 1
87	W	57	Id d,a	bit 2,a	Id a,i
88	X	58	Id e,b	bit 3,b	in e,(c)
89	Y	59	Id e,c	bit 3,c	in e,(c)
9Ø	Z	5A	Id e,d	bit 3,d	adc hl,de
91	[	5B	Id e,e	bit 3,e	Id de,(NN)
92	/	5C	Id e,h	bit 3,h	
93	]	5D	Id e,1	bit 3,1	
94	↑	5E	Id e,(hl)	bit 3,(hl)	im 2
95	_	5F	Id e,a	bit 3,a	Id a,r
96	£	6Ø	Id h,h	bit 4,b	in h,(c)
97	a	61	Id h,c	bit 4,c	out (c),h
98	b	62	Id h,d	bit 4,d	sbc hl,hl
99	c	63	Id h,e	bit 4,e	Id (NN),hl
1ØØ	d	63	Id h,h	bit 4,h	

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
1Ø1	e	65	Id h,1	bit 4,1	
1Ø2	f	66	Id h,(hl)	bit 4,(hl)	
1Ø3	g	67	Id h,a	bit 4,a	rrd
1Ø4	h	68	Id 1,b	bit 5,b	in 1,(c)
1Ø5	i	69	Id 1,c	bit 5,c	out (c),1
1Ø6	j	6A	Id 1,d	bit 5,d	adc hl,hl
1Ø7	k	6B	Id 1,e	bit 5,e	Id hl,(NN)
1Ø8	l	6C	Id 1,h	bit 5,h	
1Ø9	m	6D	Id 1,1	bit 5,1	
11Ø	n	6E	Id 1,(hl)	bit 5,(hl)	
111	o	6F	Id 1,a	bit 5,a	rld
112	p	7Ø	Id (hl),b	bit 6,b	in f,(c)
113	q	71	Id (hl),c	bit 6,c	
114	r	72	Id (hl),d	bit 6,d	sbc hl,sp
115	s	73	Id (hl),e	bit 6,e	Id (NN),sp
116	t	74	Id (hl),h	bit 6,h	
117	u	75	Id (hl),1	bit 6,1	
118	v	76	halt	bit 6,(hl)	
119	w	77	Id (hl),a	bit 6,a	
12Ø	x	78	Id a,b	bit 7,b	in a,(c)
121	y	79	Id a,c	bit 7,c	out (c),a
122	z	7A	Id a,d	bit 7,d	adc hl,sp
123	{	7B	Id a,e	bit 7,e	Id sp,(NN)
124	|	7C	Id a,h	bit 7,h	
125	}	7D	Id a,1	bit 7,1	
126	-	7E	Id a,(hl)	bit 7,(hl)	
127	©	7F	Id a,a	bit 7,a	
128	□	8Ø	add a,b	res Ø,b	
129	◳	81	add a,c	res Ø,c	
13Ø	◱	82	add a,d	res Ø,d	
131	▬	83	add a,e	res Ø,e	
132	◪	84	add a,h	res Ø,h	
133	◨	85	add a,1	res Ø,1	
134	▚	86	add a,(hl)	res Ø,(hl)	
135	▛	87	add a,a	res Ø,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	

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
139	▗	8B	adc a,e	res 1,e	
14Ø	▄	8C	adc a,h	res 1,h	
141	▖	8D	adc a,1	res 1,1	
142	▙	8E	adc a,(hl)	res 1,(hl)	
143	■	8F	adc a,a	res 1,a	
144	(a)	9Ø	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub 1	res 2,1	
15Ø	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j)	99	sbc a,c	res 3,c	
154	(k) Usar gráficos	9A	sbc a,d	res 3,d	
155	(l)	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,1	res 3,1	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
16Ø	(q)	AØ	and b	res 4,b	Idi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	(t)	A3	and e	res 4,e	outi
164	(u)	A4	and h	res 4,h	
165	RND	A5	and 1	res 4,1	
166	INKEY$	A6	and (hl)	res 4,(hl)	
167	PI	A7	and a	res 4,a	
168	FN	A8	xor b	res 5,b	Idd
169	POINT	A9	xor c	res 5,c	cpd
17Ø	SCREEN$	AA	xor d	res 5,d	ind
171	ATTR	AB	xor e	res 5,e	outd
172	AT	AC	xor h	res 5,h	
173	TAB	AD	xor 1	res 5,1	
174	VAL$	AE	xor (hl)	res 5,(hl)	
175	CODE	AF	xor a	res 5,a	
176	VAL	BØ	or b	res 6,b	Idir

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
177	LEN	B1	or c	res 6,c	cpir
178	SIN	B2	or d	res 6,d	inir
179	COS	B3	or e	res 6,e	otir
18Ø	TAN	B4	or h	res 6,h	
181	ASN	B5	or 1	res 6,1	
182	ACS	B6	or (hl)	res 6,(hl)	
183	ATN	B7	or a	res 6,a	
184	LN	B8	cp b	res 7,b	Iddr
185	EXP	B9	cp c	res 7,c	cpdr
186	INT	BA	cp d	res 7,d	indr
187	SQR	BB	cp e	res 7,e	otdr
188	SGN	BC	cp h	res 7,h	
189	ABS	BD	cp 1	res 7,1	
19Ø	PEEK	BE	cp (hl)	res 7,(hl)	
191	IN	BF	cp a	res 7,a	
192	USR	CØ	ret nz	set Ø,b	
193	STR$	C1	pop bc	set Ø,c	
194	CHR$	C2	jp nz,NN	set Ø,d	
195	NOT	C3	jp NN	set Ø,e	
196	BIN	C4	call nz,NN	set Ø,h	
197	OR	C5	push bc	set Ø,1	
198	AND	C6	add a,N	set Ø,(hl)	
199	<=	C7	rst Ø	set Ø,a	
2ØØ	>=	C8	ret z	set 1,b	
2Ø1	<>	C9	ret	set 1,c	
2Ø2	LINE	CA	jp z,NN	set 1,d	
2Ø3	THEN	CB		set 1,e	
2Ø4	TO	CC	call z,NN	set 1,h	
2Ø5	STEP	CD	call NN	set 1,1	
2Ø6	DEF FN	CE	adc a,N	set 1,(hl)	
2Ø7	CAT	CF	rst 8	set 1,a	
2Ø8	FORMAT	DØ	ret nc	set 2,b	
2Ø9	MOVE	D1	pop de	set 2,c	
21Ø	ERASE	D2	jp nc,NN	set 2,d	
211	OPEN #	D3	out (N),a	set 2,e	
212	CLOSE #	D4	call nc,NN	set 2,h	
213	MERGE	D5	push de	set 2,1	
214	VERIFY	D6	sub N	set 2,(hl)	

Código	Caracter	Hexa	Assembler Z8Ø	Depois de CB	Depois de ED
215	BEEP	D7	rst 16	set 2,a	
216	CIRCLE	D8	ret c	set 3,b	
217	INK	D9	exx	set 3,c	
218	PAPER	DA	jp c,NN	set 3,d	
219	FLASH	DB	in a,(N)	set 3,e	
22Ø	BRIGHT	DC	call c,NN	set 3,h	
221	INVERSE	DD	prefixes instructions c/ ix	set 3,1	
222	OVER	DE	sbc a,N	set 3,(hl)	
223	OUT	DF	rst 24	set 3,a	
224	LPRINT	DØ	ret po	set 4,b	
225	LLIST	E1	pop hl	set 4,c	
226	STOP	E2	jp po,NN	set 4,d	
227	READ	E3	ex(sp),hl	set 4,e	
228	DATA	E4	call po,NN	set 4,h	
229	RESTORE	E5	push hl	set 4,1	
23Ø	NEW	E6	and N	set 4,(hl)	
231	BORDER	E7	rst 32	set 4,a	
232	CONTINUE	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,C	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GO TO	EC	call pe,NN	set 5,h	
237	GO SUB	ED		set 5,1	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 4Ø	set 5,a	
24Ø	LIST	FØ	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,1	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RANDOMIZE	F9	id sp,hl	set 7,c	

Código	Caracter	Hexa	Assembles Z8Ø	Depois de CB	Depois de ED
25Ø	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	DRAW	FC	call m,NN	set 7,h	
253	CLEAR	FD	prefixes instructions c/ iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	

APENDICE B

Mensagens

As mensagens aparecem no fundo do écran quando o computador pára de executar em BASIC, e explicam porque é que ele parou, quer tenha sido um motivo natural quer tenha ocorrido um erro.

A mensagem tem um número ou letra de código de modo que se pode referenciar aqui na tabela, uma mensagem curta explicando o que aconteceu e o número de linha e número de instrução dentro da linha onde a paragem ocorreu. (Uma ordem directa é mostrada como linha Ø. Dentro de uma linha a instrução 1 é no principio, a instrução 2 vem depois dos primeiros dois pontos ou **THEN**, e assim por diante).

O comportamento do comando **CONTINUE** depende muito destas mensagens. Normalmente o **CONTINUE** vai para a linha e instrução especificadas na última mensagem, mas há excepções com as mensagens Ø, 9 e D (ver também o Apêndice C).

Aqui está uma tabela mostrando todas as mensagens. Também lhe diz em que circunstâncias a mensagem pode ocorrer, e isto envia-o para o Apêndice C. Por exemplo um erro **A** Invalid argument pode ocorrer com **SQR**, **IN**., **ACS** e **ASN** e as entradas para o Apendice C dizem-lhe exactamente os argumentos inválidos.

Código	Significado	Situações
Ø	Tudo bem Final bem sucedido, ou salto para uma linha maior do que as existentes. Esta mensagem não modifica a linha e a instrução para a qual saltou o **CONTINUE**.	Qualquer
1	NEXT sem FOR A variável de controle não existe(não foi iniciada por uma instrução **FOR**), mas há uma variável vulgar com o mesmo nome.	**NEXT**
2	Variável não encontrada Para uma variável simples isto ocorre se a variável fôr usada antes de lhe ser atribuido um valor numa instrução **LET**, **READ** ou **INPUT** ou carregada de cassete ou establecida por uma ins-	Qualquer

Código	Significado	Situações
	trução **FOR**. Para uma variável indexada acontecerá se ela fôr usada antes de ter sido dimensionada numa instrução **DIM** ou lida de uma cassete.	
3	Indice errado Um índice está para além da dimensão da matriz, ou há o número errado de índices. Se o indice fôr negativo ou maior do que 65535 então ocorrerá um erro B.	Variáveis indexadas Sub-cadeias
4	Memória esgotada Não há espaço suficiente no computador para o que está a tentar fazer. Se o computador parecer mesmo preso nesta situação pode ter de apagar a linha de comando usando **DELETE** e depois apagar uma ou duas linhas de programa (com a intenção de as voltar a introduzir) para lhe dar espaço de manobra com - por exemplo - **CLEAR**.	**LET, INPUT, FOR, DIM, GO SUB, LOAD, , MERGE**. Por vezes durante o cálculo de uma expressão
5	Écran cheio Uma instrução de INPUT tentou gerar mais do que 23 linhas na parte de baixo do écran. Também ocorre com **PRINT AT** 22...	**INPUT, PRINT AT**
6	Numero grande demais Os cálculos levaram a números maiores do que $1Ø^{38}$.	Aritmética
7	RETURN sem GO SUB Houve um **RETURN** mais do que o número de **GO SUB**s.	**RETURN**
8	Fim de ficheiro	operações com Microdrive, etc.

Código	Significado	Situações
9	Instrução STOP Depois disto a instrução **CONTINUE** não irá repetir o **STOP**, mas continua com a instrução seguinte.	**STOP**
A	Argumento inválido O argumento de uma função não servia por qualquer motivo	**SQR, LN, ASN, ACS, USR** (com argumento de cadeia)
B	Número inteiro fora da gama permitida Quando um número inteiro é exigido, o número em ponto flutuante é arredondado ao inteiro mais próximo. Se estes estiver fora da gama permitida, teremos um erro B. Para acesso a matrizes ver o erro 3.	**RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$ PEEK, USR** (com argumento numérico) Acesso a matrizes
C	Disparate em BASIC O texto do argumento (de cadeia) não forma uma expressão válida.	**VAL, VAL$**
D	BREAK - CONT repeats Foi premiada a telca **BREAK** durante uma operação periférica. O comportamento do **CONTINUE** depois desta mensagem é normal dado que repete a instrução. Comparar com a mensagem L:	**LOAD, SAVE, VERIFY, MERGE, LPRINT, LLIST, COPY**. Também quando o computador pergunta scroll? e se carrega em **N, SPACE** ou **STOP**.
E	Fim dos dados (DATA) Tentou utilizar a instrução **READ** depois do fim da linha de **DATA**.	**READ**
F	Nome de ficheiro inválido **SAVE** com o nome vazio ou maior do que 1Ø caracteres.	**SAVE**
G	Não há espaço para a linha Não há espaço suficiente na memória para receber uma nova linha de programa.	Ao introduzir uma linha de programa
H	STOP num INPUT Um dado para **INPUT** começou com um **STOP**, ou este foi carregado para um **INPUT LINE**. Ao contrário da mensagem 9, a men-	**INPUT**

Código	Significado	Situações
	sagem H faz com que o **CONTINUE** se comporte normalmente, repetindo a instrução de **INPUT**.	
I	FOR sem NEXT Houve um ciclo **FOR** que não foi executado vez nenhuma (por exemplo **FOR** n=1 **TO** Ø) e o **NEXT** correspondente não foi encontrado.	**FOR**
J	Equipamento de 1/0 inválido	Operações com Microdrive, etc
K	Cor inválida O número especificado não é um valor válido	**INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER** também depois de um dos caracteres de controle correspondentes.
L	BREAK no programa Foi carregado o **BREAK**, e isto foi detectado entre duas instruções. A linha e o número da instrução na mensagem referem-se à instrução antes do **BREAK** ter sido carregado, mas o **CONTINUE** vai para a instrução seguinte (permitindo que qualquer salto se efectue), por isso não se repetem quaisquer instruções.	Qualquer
M	RAMTOP não é válido O número especificado para o RAMTOP ou é muito pequeno ou muito grande.	**CLEAR**; possível com **RUN**
N	Instrução perdida Salto para uma instrução que já não existe.	**RETURN, NEXT, CONTINUE**
O	Corrente não válida	Operação com Microdrive, etc.
P	FN sem DEF Função definida pelo utilizador	**FN**
Q	Erro nos parâmetros Número de argumentos errado, ou um deles é do tipo errado (cadeia em vez de número, ou vice-versa).	**FN**

Código	Significado	Situações
R	Erro na leitura da fita Um ficheiro em fita pode ser lido por qualquer motivo, ou não se consegue verificar	**VERIFY LOAD** ou **MERGE**

APENDICE C

Uma descrição do XZ SPECTRUM

A primeira secção deste apêndice é uma repetição de parte da introdução no que respeita ao teclado e ao écran.

O teclado

Os caracteres do ZX Spectrum compreendem não somente símbolos (letras, digitos, etc.), mas também termos compostos (palavras-chave, nomes de funções, etc) e todos estes termos entram directamente pelas teclas em vez de serem escritos letra a letra. Para obter todas estas funções e instruções, algumas teclas têm cinco ou mais significados diferentes, dados parcialmente pelas teclas de alteração (SHIFT) (isto é, ao poremir ou **CAPS SHIFT** ou **SYMBOL SHIFT** ao mesmo tempo que a tecla requerida) e parcialmente por ter a máquina em modos diferentes.

O modo é indicado pelo cursor, uma letra a piscar que indica o local onde será colocado o próximo caracter introduzido.

O modo K (para palavra-chave - keyword) substitui automaticamente o modo L quando a máquina está à espera de uma ordem ou de uma linha de programa (em vez de dados para a instrução **INPUT**) e da sua posição na linha sabe se deve esperar um número de linha ou uma palavra-chave. Isto acontece sempre no principio de uma linha, ou exactamente depois de um **THEN**, ou depois de : (excepto quando este se encontre numa cadeia de letras). Se não se tocar em nenhuma telca de alteração (SHIFT), a próxima tecla será interpretada ou como uma palavra-chave (impressa sobre as teclas) ou um dígito.

O modo L (para letras) ocorre normalmente em todas as outras ocasiões. Se não se premir nenhuma tecla de alteração a próxima tecla será interpretada como o símbolo principal impresso na tecla, em letras pequenas para o caso de uma letra.

Tanto no modo K como no L carregar em **SYMBOL SHIFT** e noutra tecla será interpretado como o caracter a vermelho inscrito na tecla e carregar em **CAPS SHIFT** com uma tecla numérica será interpretado como uma função de controle escrita a branco acima da tecla. Carregar em **CAPS SHIFT** não afecta as outras teclas no modo K, e no modo L converte uma letra pequena em letra grande.

O modo C (para letras grandes) é uma variante do modo L em que todas as letras aparecem como letras grandes. A tecla **CAPS LOCK** transforma o modo L em C e vice-versa.

O modo E (de extensão) é obtido para usar ainda mais caracteres, essencialmente palavras. Ocorre depois de se ter carregado simultaneamente em ambas as teclas de alteração, e só se mantem até se ter premido uma tecla. Neste modo as teclas de letras dão um caracter ou uma palavra (impresso a verde por cima dela) se não se tocar mais nenhuma tecla de alteração, e outro (impresso a vermelho por baixo) se se premir uma tecla de alteração simultaneamente. Uma tecla numérica dá um simbolo se fôr premiada com **SYMBOL SHIFT**; de outra forma dá uma sequência de controle de cor.

O modo G (para gráficos) ocorre depois de se ter carregado em **GRAPHICS** (**CAPS SHIFT** e **9**)e dura até que seja novamente carregada a mesma tecla ou um 9. Uma tecla numérica dá um mosaico gráfico, exceptuando as que têm **GRAPHICS** e **DELETE**, e cada tecla de letra exceptuando V,W,X,Y,Z dará um caracter gráfico definido pelo utilizador.

Se alguma das teclas fôr premiada durante mais do que três segundos ela repete-se a si própria automaticamente.

Elementos introduzidos pelo teclado aparecem na metade inferior do écran à medida que são escritos, sendo cada um dos caracteres (símbolo único ou composto) inserido imediatamente antes do cursor. O cursor pode ser movido para a esquerda utilizando **CAPS SHIFT** e **5**, ou para a direita utilizando **CAPS SHIFT** e **8**. O caracter que se encontra imediatamente antes do cursor pode ser apagado com **DELETE** (**CAPS SHIFT** e **0**). (Nota: a linha inteira pode ser apagado premindo **EDIT** (**CAPS SHIFT** e **1**) seguido de **ENTER**).

Quando se introduz **ENTER** a linha é executada, introduzida no programa ou usada como dados de entrada (**INPUT**) conforme o caso, a não ser que ela contenha após o erro, piscando ▒

À medida que as linhas do programa vão entrando, uma listagem aparece na metade superior do écran. O modo como a listagem é produzida é bastante complicado e é explicado mais detalhadamente no Capítulo 2. A última linha entrada é chamada a linha corrente e é indicada pelo símbolo ▒ ; este símbolo pode ser movido usando as teclas ← (**CAPS SHIFT** e **6**) e ↑ (**CAPS SHIFT** e **7**). Se se pressionar **EDIT** (**CAPS SHIFT** e **1**), a linha corrente é puxada para a parte de baixo do écran e pode ser modificada.

Quando um comando é executado ou um programa é corrido, as saídas aparecem na metade superior do écran e aí ficam, até que se introduza uma linha de programa, ou se carregue em **ENTER** com uma linha em branco, ou ↑ ou ↓ sejam pressionadas. Na parte inferior aparece uma mensagem que dá um código (letra ou número) referido no Apendice B. A mensagem permanece no écran até que uma tecla seja pressionada (e indica o modo K).

Em algumas circunstâncias **CAPS SHIFT** com um espaço (**SPACE**) pode funcionar como uma paragem (**BREAK**) do computador com uma mensagem **D** ou **L**. Isto indica

(i) parou no fim de uma instrução enquanto um programa era executado ou
(ii) enquanto o computador está a usar o gravador de cassetes ou a impressora.

O écran de televisão

O écran tem 24 linhas, cada uma com 32 colunas e está divido em duas partes. A parte de cima tem no máximo 22 linhas e, ou mostra uma listagem, ou uma saída de programa. Quando a impressão na parte de cima atinge o fundo do écran, sobe tudo uma linha; se isto envolver perder uma linha que ainda não tenha tido oportunidade de ver, então o computador pára com a mensagem scroll? Carregar nas teclas **N**, **SPACE** ou **STOP** faz com que o programa seja interrompido com uma mensagem **D BREAK - CONT** repeats; qualquer outra tecla permitirá que a impressão continue. A parte de baixo é utilizada para entrar comandos, linhas de programa e dados (**INPUT**) e também para impressão de mensagens. A parte inferior começa com duas linhas (a de cima em branco), mas expande-se para se adaptar ao que está a ser escrito. Quando chegar a uma zona em que já está impresso algo na metade de cima, mais entradas farão com que a metade de cima vá subindo.

Cada posição de caracter tem atributos especificando as suas cores de papel (fundo) e tinta, um brilho de dois níveis e se está a piscar ou não. As cores disponíveis são o negro, azul, vermelho, magenta, verde, amarelo e branco.

A borda do écran pode tomar qualquer das cores usando a instrução **BORDER**.

Uma posição de caracter é dividida em 8x8 pixels e podem-se obter gráficos de alta resolução especificando individualmente se os pixels devem mostrar a cor da tinta ou do papel dessa posição de caracter.

Os atributos de uma dada posição de caracter são ajustados sempre que lá se escreva um caracter ou um pixel. O modo exacto como o ajuste é feito é determinado pelos parâmetros de impressão. dos quais há dois conjuntos (chamados permanentes e temporários) de seis: os parâmetros **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** e **OVER**. Os parâmetros permanentes para a parte de cima são colocados com as instruções **PAPER**, **INK**, etc.,e permanecem até outra instrução em contrário. (inicialmente estão em tinta negra e papel branco. Com brilho normal, sem piscar, video normal e sem impressão sobreposta). Parâmetros permanentes para a parte de baixo usam a cor do bordo como cor de papel, com a cor da tinta a preto ou branco para contrastar, brilho normal, sem piscar, video normal e sem impressão sobreposta.

Os parâmetros temporários são colocados com os elementos **PAPER**, **INK**, etc. que estão incluidos em instruções **PRINT**, **LPRINT**, **INPUT**, **PLOT**, **DRAW** e **CIRCLE**, e também pelos caracteres de controle **PAPER**, **INK**, etc. quando são impressos na televisão - são seguidos por um byte extra para especificar o valor do parâmetro. Parâmetros temporários só duram até ao fim da instrução **PRINT** (ou outra), ou, em instruções de **INPUT**, até ter entrado os dados de **INPUT** necessários no teclado, altura em que são substituidos pelos parâmetros permanentes.

Os parâmetros **PAPER** e **INK** estão numa gama de Ø a 9. Os parâmetros de Ø a 7 são as cores usadas na impressão:

Ø negro
1 azul
2 vermelho
3 magenta
4 verde
5 cian
6 amarelo
7 branco

O parâmetro 8 ('transparente') especifica que a cor no écran não deve ser alterada quando um caracter é impresso.

FLASH e **BRIGHT** são parâmetros que só podem ser Ø, 1 ou 8: 1 significa que o brilho ou o piscar são ligados, Ø que são desligados e 8 ('transparente') que são deixados como estão.

Os parâmetros **OVER** e **INVERSE** só podem ser Ø ou 1.

OVER Ø	caracter novo apaga o anterior
OVER 1	os padrões de bit dos caracteres novos e antigos são combinados usando uma operação de ou exclusivo
INVERSE Ø	os novos caracteres são impressos com a cor da tinta sobre a cor do papel (video normal)
INVERSE 1	os novos caracteres são impressos com a cor do papel sobre a cor da tinta (video inverso)

Quando é recebido um caracter de controle **TAB** na televisão, esperam-se mais dois bytes para especificar o ponto de paragem do tab (primeiro o byte menos significativo). Isto é reduzido a modulo 32 para n_o (por exemplo) e então são deixados espaços em branco suficientes para deslocar a posição de impressão para a coluna n_o.

Quando se recebe um caracter de controle vírgula, então deixam-se espaços suficientes (pelo menos um) para deslocar a posição de impressão para a coluna Ø ou 16.

Quando se encontra um caracter de controle **ENTER**, a posição de impressão é deslocada para a próximo linha.

A impressora

A saída para a impressora ZX é feita via um tampão em linha de 32 caracteres de comprimento e uma linha é enviada para a impressora.

(i) quando a impressão salta de uma linha para a outra
(ii) Quando se recebe um caracter **ENTER**
(iii) no fim do programa, se ainda houver soisas por escrever
(iv) Quando se encontra um controle **TAB** ou uma vírgula que desloca a posição de impressão para nova linha.

Os controles **TAB** e os controles de vírgula fazem sair espaços da mesma forma que para a televisão.

O controle **AT** modifica a posição de impressão usando um número de coluna e ignorando o número de linha.

A impressora é afectada pelos controles **INVERSE** e **OVER** (e também pelas instruções) da mesma forma que o écran, mas não pelas inscruções **PAPER**, **INK**, **FLASH** ou **BRIGHT** .

A impressora pára com um erro B se se carregar em **BREAK**.

Se a impressora estiver ausente a saída perde-se.

O BASIC

Os números são guardados com uma precisão entre 9 e 1Ø digitos. O maior número que se pode guardar é cerca de $1Ø^{38}$, e o mais pequeno número (positivo) é cerca de $4*1Ø^{-39}$.

Um número é guardado no ZX Spectrum em binário de ponto flutuante com um byte para o expoente e (1 < =e < =255), e quatro bytes para a mantissa m ($^1/2$ < =m < 1). Isto representa o número $m*2^{e-128}$.

Uma vez que $^1/2$ < =m < 1, o bit mais significativo da mantissa m é sempre 1. Assim na verdade podemos substitui-lo com um bit para mostrar o sinal - Ø para números positivos, 1 para negativos.

Inteiros pequenos têm uma representação especial na qual o primeiro byte é Ø, o segundo é um byte de sinal (Ø ou FFh) e o terceiro e o quarto são o inteiro na forma binária, sendo o byte menos significativo o primeiro.

Variáveis numéricas têm nomes de comprimento arbitrário, começando com uma letra e continuando com letras e dígitos. Espaços e controles de cor são ignorados e todas as letras são convertidas para letras minúsculas.

Variáveis de controle para ciclos **FOR-NEXT** têm como nome uma única letra.

Matrizes numéricas têm nomes com uma única letra, podendo ser igual ao nome de uma variável simples. Podem ter um número de dimensões arbitrário. Os índices começam em 1.

As cadeias são completamente flexiveis no comprimento. O nome de uma cadeia consiste numa única letra seguida por $.

Matrizes de cadeias podem ter um número de dimensões arbitrário, de comprimento arbitrário também. O nome é uma única letra seguida por $ e não pode ser igual ao nome de uma cadeia simples. Todas as cadeias numa dada matriz têm o mesmo comprimento fixado, que é especificado por uma dimensão extra, no fim da lista da instrução **DIM**. Os indices começam em 1.

Corte: Sub-cadeias de cadeias podem ser especificadas por instruções de corte. Um corte pode ser

(i) vazio

ou

(ii) uma expressão numérica

ou

(iii) expressão numérica de opção **TO** expressão numérica de opção

e é usado para exprimir uma sub-cadeia por

(a) expressão de cadeia (corte)
(b) matriz de cadeia (indice,..., indice, corte)

que significa o mesmo que

variável de uma matriz de cadeia (indice,..., indice) (corte)

Em (a) suponhamos que a expressão de cadeia tem o valor s$.

Se o corte fôr vazio, o resultado é s$ considerado ele mesmo como uma sub-cadeia.

Se o corte fôr uma expressão numérica com valor m, então o resultado é o m-ésimo caracter de s$ (uma sub-cadeia de comprimento 1).

Se o corte tiver a forma (iii), então suponha que a primeira expressão numérica tem o valor m (se faltar considera-se 1), e o segundo n (se faltar considera-se o comprimento de s$).

Se 1 < =m < =n < = comprimento de s$ então o resultado é a sub-cadeia de s$ que começa com o m-ésimo caracter e acaba no n-ésimo.

Se ∅ < =n < m então o resultado é uma cadeia vazia.

Noutro caso resulta um erro 3.

O corte é executado antes que as funções ou expressões sejam avaliadas, a não ser que se usem parêntesis para indicar outra forma.

Sub-cadeias podem receber directamente valores (ver LET).

Se umas aspas tiverem de aparecer num literal de cadeia, então devem aparecer a dobrar.

Funções

O argumento de uma função não precisa de parêntesis se fôr uma constante ou uma variável (possivelmente com indice ou corte).

Função	tipo de argumento.	Resultado
ABS	número (x)	valor absoluto
ACS	número.	Arcocoseno em radianos Erro A se x não estiver entre -1 e 1
AND	operação binária, operando da direita sempre um número.	

Função	Tipo de argumento	Resultado
	operando numérico da esquerda: operando literal da esquerda:	A **AND** B = { A se B <> Ø ; Ø se B=Ø } A$ **AND** B= { A$ se B <> Ø ; "" se B=Ø }
ASN	número	Arcoseno em radianos. Erro A se x não estiver entre -1 e 1
ATN	número	Arcotangente em radianos.
ATTR	dois argumentos, x e y, ambos números; entre parêntesis	Um número cuja forma binária codifica os atributos da linha x, coluna y da televisão. O bit 7 (o mais significativo) é 1 se o campo piscar e Ø se não. O Bit 6 é 1 para brilho acentuado e Ø para normal. Os Bits de 5 a 3 são a cor do papel. Os Bits de 2 a Ø a cor da tinta. Erro B se Ø <=x <=23 e Ø <=y <=31.
BIN		Não é verdadeiramente uma função, mas uma notação alternativa para números: **BIN** seguido por uma sequência de Øs e 1s é o número com tal representação em binário.
CHR$	número	O caracter cujo código é x, arredondado ao inteiro mais próximo.
CODE	cadeia	O código do primeiro caracter em x (ou Ø se x fôr uma cadeia vazia).
COS	número(em radianos)	Coseno de x
EXP	número	e^x
FN		**FN** seguido por uma letra chama a função definida pelo utilizador (ver **DEF**). Os argumentos têm de estar entre parêntesis; ainda que não existam quaisquer argumentos, os parêntesis têm de aparecer
IN	número	O resultado de uma entrada ao nivel do processador da porta x (Ø <=x <=FFFFh) (carrega o registo bc com x e executa a instrução da linguagem assembly in a (c)).
INKEY$	nenhum	Lê o teclado. O resultado é o caracter representando (no modo **L** ou **C**) a tecla premida se fôr exactamente uma, de outra forma será uma cadeia vazia.

Função	Tipo de argumento	Resultado
INT	número	Parte inteira(arredondada sempre para baixo).
LEN	cadeia	Comprimento
LN	número	Logaritmo natural (na base e) Erro A se x < =Ø
NOT	número	Ø se x <>Ø, 1 se x=Ø. **NOT** tem prioridade 4
OR	operação binária, ambos os operandos são números	a **OR** b = { 1 se b <>Ø ; Ø se b=Ø } **OR** tem prioridade 2
PEEK	número	O valor do byte em memória cujo endereço é x arredondado ao inteiro mais próximo Erro B se x não estiver na gama Ø - 65535
PI	nenhum	π (3.14159265...)
POINT	Dois argumentos, x e y, ambos números; entre parêntesis	1 se o pixel em (x,y) tiver a cor da tinta. Ø se tiver a cor do papel. Erro B a não ser que Ø < =x < =255 e Ø < =y < =175
RND	nenhum	O próximo número pseudo-aleatório na sequência gerada tomando as potências de 75 modulo 65537, subtraindo 1 e dividindo por 65536. Ø≤y<1 Resultado entre Ø e 1 (sempre diferente de 1)
SCREEN$	Dois argumentos, x e y, ambos números; entre parentesis	O caracter que aparece, quer normal quer invertido, na televisão, na linha x e coluna y. Dá uma cadeia vazia, se o caracter não é reconhecido. Erro B se x não estiver entre Ø e 23 e y entre Ø e 31
SGN	número	Sinal: o sinal (-1 se fôr negativo, Ø se fôr zero ou +1 para positivo) de x
SIN	número(em radianos)	Seno de x
SQR	número	Raiz quadrada Erro A se x <Ø
STR$	número	A cadeia de caracteres que seria gerada se o x fosse impresso.
TAN	número(em radianos)	Tangente
USR	número	Chama a subrotina em linguagem máquina cujo primeiro endereço é x. Ao voltar o resultado é o conteúdo do registo bc.

Função	Tipo de argumento	Resultado
USR	cadeia	O endereço do padrão de bit para o caracter gráfico definido pelo utilizador correspondente a x. Erro A se x não fôr uma única letra entre a e u, ou um caracter gráfico definido pelo utilizador.
VAL	cadeia	Calcula x (sem as suas aspas) como uma expressão numérica. Erro C se x contiver um erro de sintaxe ou der um valor de cadeia. Outros erros possiveis dependendo da expressão.
VAL$	cadeia	Calcula x (sem as suas aspas) como uma expressão literal. Erro C se x contiver um erro de sintaxe ou der um valor numérico. Outros erros possiveis como para **VAL**.
–	número	Negação

As seguintes são operações binárias:

+	Adição (em números), ou colagem (em cadeias)
–	Subtração
*	Multiplicação
/	Divisão
↑	Elevar a uma potencia. Erro B se o operando da esquerda fôr negativo
=	Igual
>	Maior do que
<	Menor que
<=	Menor ou igual a
>=	Maior ou igual a
<>	Diferente

Ambos operandos têm de ser do mesmo tipo. O resultado é um número, 1 se a comparação fôr verdadeira e Ø se não fôr.

As funções e operações têm as seguintes prioridades:

Operação	Prioridade
corte e sub-cadeias	12
Todas as funções excepto **NOT** e menos unário	11
↑	1Ø

Operação	Prioridade
Menos unário (isto é menos usado para negar)	9
*, /	8
+, – (menos usado para subtração)	6
= >, <, < =, > =, <>	5
NOT	4
AND	3
OR	2

Instruções
Nesta lista

x	representa uma única letra
v	representa uma variável
x,y,z	representa uma expressão numérica
m,n	representam expressões numéricas que são arredondadas ao inteiro mais próximo
e	representa uma expressão
f	representa uma expressão cujo valor é uma cadeia
s	representa uma sequência de instruções separadas por dois pontos
c	representa uma sequência de elementos de cor, cada um terminando por vírgulas ou ponto-e-vírgula. Um elemento de cor tem a forma de uma instrução **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** ou **OVER**.

Repare que expressões arbitrárias são permitidas em qualquer parte (exceptuando nos números de linha no começo de uma instrução).

Todas as instruções, exceptuando **INPUT**, **DEF** e **DATA** podem ser usadas como comandos ou em programas (ainda que seja mais razoável no segundo caso do que no primeiro). Um comando ou uma linha de programa podem ter várias instruções separadas por dois pontos. Não há restrições quanto à posição de qualquer instrução dentro de uma linha -- no entanto veja **IF** e **REM**.

BEEP x,y — Emite uma nota pelo altifalante durante x segundos e com uma frequência y semi-tons acima do Dó medio (ou abaixo se y fôr negativo)

BORDER m — Especifica a cor da borda do écran e também a cor do papel para a parte de baixo do écran.
Erro K se m não estiver na gama Ø a 7

BRIGHT n — Epecifica o brilho dos caracteres que serão impressos a seguir. n=Ø para brilho normal, 1 para brilho acentuado, 8 para transparência.

Erro K se n não fôr Ø, 1 ou 8

CAT — Não funciona sem um Microdrive, etc.

CIRCLE x,y,z — Desenha um arco de circunferência, centro (x,y) raio z.

CLEAR — Apaga todas as variáveis, libertando o espaço que eles ocupavam.

Faz um **RESTORE** e **CLS**, repõe a posição de **PLOT** no canto inferior esquerdo e apaga a pilha **GO SUB**.

CLEAR n — Como **CLEAR**, mas se fôr possivel modifica o a variável do sistema chamada **RAMTOP** para o valor n e coloca lá a nova pilha **GO SUB**.

CLOSE # — Não funciona sem Microdrive, etc.

CLS — Limpa o ficheiro de imagem.

CONTINUE — Continua o programa, começando onde parou da última vez com uma mensagem diferente de Ø. Se a mensagem fosse 9 ou L, então continuava com a instrução seguinte (tendo em conta possiveis saltos); de resto repete aquela onde o erro ocorreu.

Se a última mensagem foi numa linha de comando, então o **CONTINUE** tentará continuar a linha de comando e ou entrará num círculo se o erro fosse em Ø:1, ou dará uma mensagem Ø se fosse em Ø:2, ou dá um erro N se fosse em Ø:3 ou maior.

CONTINUE aparece como **CONT** no teclado.

COPY — envia uma cópia das primeiras 22 linhas do écran para a impressora se ela estiver ligada; de outra forma não faz nada. Repare que o **COPY** não pode ser usado para imprimir as listagens automaticas que aparecem no écran.

Dá mensagem D se se carregar em **BREAK**.

DATA $e_1,e_2,\ldots$ — Parte de uma lista **DATA**. Tem de estar em programa

DEF FN $a.(a_1,\ldots,a_k)=e$ — Definição de uma função definida pelo utilizador; tem de estar num programa. Cada um dos a,a_1 até a_k ou é uma letra única ou uma letra seguida de '$' para um argumento de cadeia ou um resultado de cadeia.

Toma a forma **DEF FN** a ()=e se não tiver argumentos

DELETE f — Não funciona sem uma Microdrive, etc.

DIM α (n$_1$,...,n$_k$) — Apaga qualquer matriz com o nome α e organiza uma matriz α de números, com k dimensões n$_1$,...n$_k$. Inicializa todos os valores a Ø.

DIM α$ (n1,...n$_k$) — Apaga qualquer matriz ou cadeia com o nome α$, e organiza uma matriz de caracteres com k dimensões n$_1$,...,n$_k$. Inicializa todos os valores a " ". Isto pode ser considerado uma matriz de cadeias de comprimento fixo n$_k$, com k-1 dimensões n$_1$,...,n$_{k-1}$.

Erro 4 pode ocorrer se não houver espaço para a matriz. Uma matriz está indefinida até que seja dimensionada numa instrução **DIM**.

DRAW x,y — **DRAW** x,y,Ø

DRAW x,y,z — Traça uma linha da posição de PLOT corrente movendo x pixels horizontalmente e y verticalmente relativamente ao ponto inicial e rodando um ângulo z.

Erro B se sair do écran

ERASE — Não funciona sem Microdrive, etc.

FLASH — Define se um caracter deve piscar ou não. n=Ø para não piscar, n=1 para piscar, n=8 para não mudar

FOR α=x **TO** y — **FOR** α=x **TO** y **STEP** 1

FOR α=x **TO** y **STEP** z — Apaga qualquer variável simples chamada α e cria uma variável de controle com o valor x, limite y e incremento z, e um endereço de ciclo que se refere à instrução depois de **FOR**. Verifica se o valor inicial é superior (se o passo fôr maior ou igual a Ø) ou menor (se o passo fôr negativo) do que o limite, e se fôr salta para a instrução **NEXT** α, dando erro 1 se esta não existir. Ver **NEXT**.

Erro 4 ocorre se não houver espaço para a variável de controle.

FORMAT f — Não funciona sem Microdrive, etc.

GOSUB n — Leva o número de linha da instrução **GOSUB** para uma pilha; depois faz como o **GO TO** n.

Erro 4 ocorre se não houver **RETURN**s suficientes.

GO TO n — Salta para a linha n (ou, se não existir, para a primeira linha depois dessa).

IF x **THEN** s — Se x é verdade (diferente de zero) então s é executado. Repare que s compreende todas as instruções até ao fim da linha. A forma '**IF** x **THEN** número de linha' não é permitida.

INK n

Estabelece a cor da tinta dos caracteres a ser escritos a seguir. n tem de estar entre Ø e 7 para uma cor, n=8 para transparência e 9 para contraste. Veja O écran de televisão - Apendice C(1).

Erro K se n não estiver entre Ø e 9.

INPUT

O '...' é uma sequência de elementos de **INPUT**, separados como numa instrução **PRINT** por vírgulas, ponto-e-vírgula ou apóstrofos. Um elemento de **INPUT** pode ser

(i) Qualquer elemento de **PRINT** que não comece com uma letra

(ii) o nome de uma variável, ou

(iii) **LINE** e depois o nome de uma variável do tipo de cadeia.

Os elementos de **PRINT** bem como os separadores comentados em (i) são tratados exactamente em **PRINT**; exceptuando que tudo é impresso na parte de baixo do écran.

Quando encontra um elemento do tipo (ii) o computador pára para admitir uma expressão do teclado; o valor desta expressão é atribuido à variável. A entrada é tratada da forma usual e os erros de sintaxe dão um ? a piscar. Para expressões do tipo de cadeia, o tampão de entrada é inicialisado para conter duas aspas (que podem ser apagadas em caso de necessidade). Se o primeiro caracter de entrada é **STOP**, o programa pára com um erro H. (iii) é como (ii) excepto que a entrada é tratada como uma cadeia sem as aspas, e o mecanismo **STOP** não funciona; para parar é necessário introduzir ↵ em vez disso.

INVERSE n

Controla a inversão de caracteres que são impressos a seguir. Se n=Ø, os caracteres é impresso em video normal, com a cor da tinta sobre a cor do papel.

Se n=m os caracteres são impressos em video inverso isto é a cor do papel sobre a cor da tinta. Veja O écran de televisão - Apendice C(1).

Erro K se n não é nem Ø nem 1.

LET v=e — Atribui o valor de e à variável v. A palavra LET não pode ser omitida. Uma variável simples é indefenida até que lhe seja atribuido um valor numa instrução LET, READ ou INPUT. Se v fôr uma variável de cadeia indixada, ou uma variável de cadeia cortada (su-cadeia), então a atribuição é Procrusteana (comprimento fixo): o valor de cadeia de e é truncado ou cheio de espaços à direita para ficar com o mesmo comprimento que a variável v.

LIST — LIST Ø

LIST n — Lista o programa na parte superior do écran começando com a primeira linha cujo número seja maior ou igual a n e torna n a linha corrente.

LLIST — LLIST Ø

LLIST n — Como LIST mas usando a impressora

LOAD f — Carrega um programa e variáveis.

LOAD f DATA () — Carrega uma matriz numérica.

LOAD f DATA $() — Carrega uma matriz de caracteres

LOAD f CODE m,n — Carrega no maximo n bytes começando no endereço m

LOAD f CODE m — Carrega bytes começando no endereço m

LOAD f CODE — Carrega os bytes de novo para o endereço de onde sairam

LOAD f SCREEN$ — LOAD f CODE 16384,6912.

Procura o ficheiro do tipo certo na cassete e carrega-o, apagando as anteriores versões em memória. Ver o capítulo 2Ø.

LPRINT — Como PRINT mas usando a impressora.

MERGE f — Como LOAD f, mas não apaga o programa nem as variáveis anteriores, excepto para arranjar espaço para os novos que tenham nome ou número igual.

MOVE f_1,f_2 — Não funciona sem um Microdrive, etc.

NEW — Faz o sistema BASIC começar do principio, apagando o programa e as variáveis e usando a memória até e incluindo o byte cujo endereço estiver na variável de sistema denominada RAMBOT e guarda as variáveis do sistema UDG, P RAMT, BASP e PIP.

NEXT α — (i) Encontra a variável de controle α
(ii) Adiciona o seu incremento ao seu valor
(iii) Se o incremento fôr maior ou igual a zero e o valor maior que o limite; ou se o incremento fôr me-

nor que zero e o valor menor que o limite, então salta para fora do ciclo.

Erro 2 se não existir a variável a.

Erro 1 se houver mas não fôr de controle a.

OPEN # Não funciona sem um Microdrive, etc.

OUT m,n Faz sair o byte n no porto m ao nivel do processador. (Carrega o par de registos bc com m, o registo a com n e executa a instrução de linguagem assembly out (c),a.)

Ø <=m <=65535, -255 <=n <=255, ou então Erro B.

OVER n Controla a impressão sobreposta dos caracteres que forem escritos depois da instrução.

Se n=Ø, os caracteres apagam os anteriores nessa posição.

Se n=1, os novos caracteres são misturados com os antigos dando cor da tinta onde qualquer dos pontos a tivesse (mas não os dois simultaneamente) e cor do papel dos locais onde ambos tivessem ou a cor da tinta ou a cor do papel. Veja O écran da televisão - Apendice C(1).

Erro K se n não fôr Ø ou 1

PAPER n Como **INK** mas controlando a cor do papel (fundo).

PAUSE n Pára de calcular e emite o ficheiro de imagem durante n imagens (a 5Ø imagens por segundo ou 6Ø se fôr na América do norte) ou até que uma tecla seja premida.

n tem de estar entre Ø e 65535 ou aparece um erro B. Se n=Ø então a pausa não tem tempo defenido e dura até que seja premida uma tecla.

PLOT c;m,n Imprime um ponto de tinta (sujeito a **OVER** e **INVERSE**) no pixel (/m/,/n/); desloca a posição de **PLOT**.

A menos que o elemento de cor c especifique outra coisa, a cor da tinta na posição do caracter contendo o pixel é mudado para a cor corrente e permanente da tinta e os outros atributos (cor do papel, piscar e brilho) são deixados na mesma.

Ø <=m <=255, Ø <=n <=175, ou então erro B.

POKE m,n Escreve o valor n no byte que na memória tem endereço m.

Ø <=n <=255, Ø <=m <=65535 ou então Erro B.

PRINT... Os '...' são uma sequência de elementos de **PRINT**, separados por virgulas , , ponto-e-virgula ; , ou plicas ' e são enviados para o ficheiro de imagem para serem colocados na televisão.

Um ponto-e-virgula entre dois elementos não tem qualquer efeito, é somente para separar os elementos. Uma virgula envia o caracter de controle virgula, e uma plica envia o caracter **ENTER**.

Um fim da instrução **PRINT**, se não acabar com um ponto-e-virgula, uma virgula ou uma plica, é enviado também um caracter **ENTER**.

Um elemento de **PRINT** pode ser

(i) vazio, isto é, nada.

(ii) uma expressão numerica

Primeiro é impresso um sinal menos se o número fôr negativo. Agora seja x o modulo do valor Se $x < =10^{-5} >$ ou $x) =1\emptyset^{13}$, então é impresso usando notação cientifica. A parte da mantissa tem até oito digitos (sem os zeros à esquerda), e o ponto decimal, ausente no caso de ser só um digito, está depois do primeiro. A parte do expoente é E seguido de + ou - seguido de um ou dois digitos.

De outra forma o x é impresso na notação decimal corrente com até oito digitos significativos, e sem zeros à direita do ponto decimal. Um ponto decimal mesmo no principio do número é sempre acompanhado de um zero, de modo que .Ø3 e Ø.3 são escritos desta forma.

Ø é impresso como um único digito Ø.

(iii) uma expressão de cadeia

Os simbolos compostos na cadeia são expandidos, possivelmente com um espaço antes ou depois.

Os caracteres de controle têm o seu efeito de controle.

Caracteres que não são reconhecidos aparecem como ?.

(iv) **AT** m,n

Faz sair um caracter de controle **AT** seguido

por um byte para m (o número de linha) e um byte para n (número de coluna).

(v) **TAB** n

Emite um caracter de controle **TAB** seguido por dois bytes para n (o byte menos significativo primeiro), e o fim do **TAB**.

(vi) Um elemento de cor que toma a forma de uma instrução **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** ou **OVER**.

RANDOMIZE **RANDOMIZE** Ø

RANDOMIZE n Inicia a variável do sistema (chamada SEED) usada para gerar o prxima valor de **RND**. Se n fôr diferente de Ø, a variável SEED recebe o valor de n; se n=Ø então recebe o valor de outra variável do sistema chamada FRAMES que conta as imagens que foram até ali emitidas para a televisão, e por isso deve ser bastante aleatório.

RANDOMIZE aparece como **RAND** no teclado.

Ocorre um erro B se n não estiver compreendido entre Ø e 65535.

READ $v_1,...v_k$ Dá valores às variáveis usando valores sucessivos da lista **DATA**.

Erro C se a expressão fôr do tipo errado.

Erro E se houver ainda variáveis por ler quando a lista **DATA** se esgotar.

REM ... Não tem qualquer efeito. '...' pode ser qualquer sequência de caracteres, exceptuando **ENTER**. Estes podem incluir :, de modo que depois de uma instrução **REM** não são possiveis mais instruções na mesma linha

RESTORE **RESTORE** Ø

RESTORE n Repõe o ponteiro da lista **DATA** para a primeira instrução **DATA** na linha com o número maior ou igual a n: a proxima instrução **READ** começará a ler ali.

RETURN Toma como referência uma instrução retirada da pilha **GO SUB** e salta para a linha que lá fôr indicada.

Ocorre um erro 7 quando não há nenhuma instrução referenciada na pilha. Deve haver um erro qualquer no seu programa; o número de **GO SUB**s não está correctamente equilibrado com o número de **RETURN**s.

RUN **RUN** Ø

RUN n — **CLEAR** seguido de **GO TO** n

SAVE f — Guarda em cassete o programa e variáveis.

SAVE f **LINE** — Guarda o programa e variáveis de modo que quando estes forem carregados há um salto automatico para o principio do programa.

SAVE f **LINE** m — Guarda o programa e variáveis de modo que se estes forem carregados há um salto automatico para a linha m.

SAVE f **DATA** () — Guarda a matriz numérica.

SAVE f **DATA** $ () — Guarda a matriz de caracteres $.

SAVE f **CODE** m,n — Guarda n bytes começando no endereço m.

SAVE f **SCREEN$** — **SAVE** f **CODE** 16384,6912.

Guarda informação na cassete dando-lhe o nome f. Errr F se f fôr uma cadeia vazia ou tiver um comprimento maior do que 1Ø. Ver o Capítulo 2Ø.

STOP — Pára o programa com uma mensagem 9. A instrução **CONTINUE** retoma a execução na instrução seguinte.

VERIFY — O mesmo que a instrução **LOAD** excepto que os dados não são carregados para a RAM , mas comparados com os que já lá existem.

Erro R se uma das comparações mostrar diferenças.

APENDICE D

Programas de exemplo

Este apendice contem alguns programas exemplificativos para demonstrar as capacidades do ZX Spectrum.

O primeiro destes programas pede que lhe seja dada uma data e diz o dia da semana que corresponde a esta data.

```
10 REM converter data em dia
20 DIM d$(7,6): REM dias da semana
30 FOR n=1 TO 7: READ d$(n): NEXT n
40 DIM m(12): REM comprimento dos meses
50 FOR n=1 TO 12: READ m(n): NEXT n
100 REM dados de entrada
110 INPUT "dia?"; dia
120 INPUT "mês?"; mês
130 INPUT "ano (só para o seculo XX)?; ano
140 IF ano<1901 THEN PRINT "O seculo XX começa em 1901":
GO TO 100
150 IF ano>2000 THEN PRINT "O seculo XX acaba em 2000":
GO TO 100
160 IF mes<1 THEN GO TO 210
170 IF mes >12 THEN GO TO 210
180 IF ano /4-INT(ano/4)=0 THEN LET m(2)=29: REM ano
bisexto
190 IF dia >m(mes) THEN PRINT "Este mes só tem":
m(mes); "dias.": GO TO 500
200 IF dia>0 THEN GO TO 300
210 PRINT "Cambada de disparates. De-me uma data real".
220 GO TO 500
300 REM converter a data para número de dias desde o
começo do século
310 LET y=ano-1901
320 LET l=365*y+INT (y/4): REM número de dias até ao
começo do ano
330 FOR n=1 TO mes-1: REM somar os meses anteriores
340 LET l=l+m(n) NEXT n
350 LET l=l+ dia
400 REM converter para dia da semana
```

```
410 LET l=l-7* INT (l/7)+1
    420 PRINT di ; "/";mes;"/"; ano
            430 FOR n=6 TO 3 STEP -1: REM retirar os espaços
440 IF d$(l,n) <> " " THEN GO TO 460
450 NEXT n
460 LET e$=d$(l, TO n)
    470 PRINT "é uma"; d1$; dia
        500 LET m(2)=28: REM repor Fevereiro
    510 INPUT "outra vez?", a$
    520 IF a$="n" THEN GO TO 540
530 IF a$ <> "N" THEN GO TO 100
1000 REM dias da semana
1010 DATA "Segunda", "Terca", "Quarta"
1020 DATA "Quinta", "Sexta", "Sabado", "Domingo"
1100 REM comprimento dos meses
1110 DATA 31, 28, 31, 30, 31, 30
1120 DATA 31, 31, 30, 31, 30, 31
```

Este programa trata jardas, pés e polegadas.

```
10 INPUT "jardas?", yd, "pés?", ft, "polegadas", in
  40 GO SUB 2000: REM imprimir os valores
50 PRINT " = ";
  70 GO SUB 1000: REM o ajuste
  80 GO SUB 2000: REM imprimir os valores ajustados
90 PRINT
100 GO TO 10
1000 REM subrotina para ajustar yd., ft, in para a for-
    ma normal em jardas, pés e polegadas
    1010 LET in=36*yd+12*ft+in: REM tudo para polegadas
    1030 LET s=SGN in: LET in=ABS in: REM trabalhamos com
    in positivo, guardando o sinal em s
            1060 LET ft=INT (in/12): LET in=(in-12*ft)*s: REM agora
    in já esta bem
1080 LET yd=INT (ft3)*s: LET ft=ft*s-3*yd: RETURN
2000 REM subrotina para imprimir yd., ft e in
2010 PRINT yd;"yd";ft;"ft";in;"in";: RETURN
```

Aqui está um programa para atirar moedas para o I CHING (infelizmente produz os padrões de pernas para o ar, mas não tem que se preocupar com isto. No capítulo 17 vai aprender para que serve o AT, que lhe permite imprimir de baixo para cima).

```
 5 RANDOMIZE
10 FOR m=1 TO 6: REM 6 lançamentos
20 LET c=0: REM inicializar o total de moedas a 0
30 FOR n=1 TO 3: REM 3 moedas
40 LET c=c+2+INT (2*RND)
50 NEXT n
60 PRINT "    ";
70 FOR n=1 TO 2: REM primeiro para o hexagrama de
   lançamentos, depois para as alterações
80 PRINT "---";
90 IF c=7 THEN PRINT "-";
100 IF c=8 THEN PRINT " ";
110 IF c=6 THEN PRINT "X";: LET c=7
120 IF c=9 THEN PRINT "0";: LET c=8
130 PRINT "---   ";
140 NEXT n
150 PRINT
160 INPUT a$
170 NEXT m: NEW
```

Para usar este programa introduza-o e execute-o, e depois carregue em **ENTER** cinco vezes para obter os dois hexagramas. Procure-os num exemplar do Livro Chinês das Alterações. O texto descrever-lhe-à a situação e as medidas a tomar, e tem de pensar maduramente para descobrir os paralelos entre isso e a sua vida. Carregue em **ENTER** pela sexta vez e o programa apaga-se a si próprio - isto é para o desencorajar de o usar frivolamente.

Muitas pessoas acham que os textos são mais certos do que seria de esperar pelas probabilidades; este pode ser ou não o caso com o seu ZX Spectrum. Em geral os computadores são criaturas sem Deus.

Aqui está um programa para jogar à adivinha de nomes de animais. Você pensa num animal e o computador tenta adivinhar que animal é, fazendo-lhe perguntas que podem ser respondidas com 'sim' ou 'não'. Se ele nunca tiver ouvido falar desse animal, pede-lhe para lhe dar uma pergunta para usar da próxima vez que alguém pense nesse animal.

```
 5 REM adivinha
10 LET nq=100: REM número de perguntas e animais
```

```
 15 DIM q$(nq,50): DIM a(nq,2): DIM r$(1)
 20 LET qf=8
 30 FOR n=1 TO qf/2-1
 40 READ q$(n): READ a(n,1): READ a(n,2)
 50 NEXT n
 60 FOR n=n TO qf-1
 70 READ q$(n): NEXT n
100 REM começo do jogo
110 PRINT "Pense num animal.", "Carregue numa tecla
    qualquer para continuar"
120 PAUSE 0
  130 LET c=1: REM começar pela primeira pergunta
140 IF a(c,1)=0 THEN GO TO 300
150 LET p$=q$(c): GO SUB 910
160 PRINT "?": GO SUB 1000
170 LET in=1: IF r$="y" THEN GO TO 210
180 IF r$="Y" THEN GO TO 210
190 LET in=2: IF r$="n" THEN GO TO 210
200 IF r$<>"N" THEN GO TO 150
210 LET c=a(c,in): GO TO 140
300 REM animal
 310 PRINT "Está a pensar em";
320 LET p$=q$(c): GO SUB 900 PRINT "?"
330 GO SUB 1000
340 IF r$="y" THEN GO TO 400
350 IF r$="Y" THEN GO TO 400
360 IF r$="n" THEN GO TO 500
370 IF r$="N" THEN GO TO 500
 380 PRINT "Responda-me como deve ser quando","falo
    consigo.": GO TO 300
400 REM adivinhou
 410 PRINT "Logo vi.": GO TO 800
500 REM animal novo
      510 IF qf>nq-1 THEN PRINT "Tenho a certeza que o seu
    animal","é muito interessante, mas eu não tenho",
    "espaço para ele de momento.": GO TO 800
      520 LET q$(qf)=q$(c): REM mover animal antigo
 530 PRINT "Então qual é ele?": INPUT q$(qf+1)
 540 PRINT "De-me uma pergunta que o distin-","ga en-
    tre"
```

```
550 LET p$=q$(qf): GO SUB 900: PRINT "e "
560 LET p$=q$(qf+1): GO SUB 900: PRINT
570 INPUT s$: LET l=LEN s$
580 IF s$(l)="?"" THEN LET l=l-1
590 LET q$(c)=s$(TO l): REM inserir pergunta
600 PRINT "Qual é a resposta para"
610 LET p$=q$(qf+1): GO SUB 900: PRINT "?"
620 GO SUB 1000/
630 LET in=1: LET io=2: REM respostas para os animais
    novos e antigos
640 IF r$="y" THEN GO TO 700
650 IF r$="Y" THEN GO TO 700
660 LET in=2: LET io=1
670 IF r$="n" THEN GO TO 700
680 IF r$="N" THEN GO TO 700
690 PRINT "Essa não serve": GO TO 600
700 REM actualizar as respostas
710 LET a(c,in)=qf+1: LET a(c,io)=qf
720 LET qf=qf+2: REM proximo espaço livre para animal
730 PRINT "Esse enganou-me"
800 REM outra vez?
810 PRINT "Quer jogar outra vez?": GO SUB 1000
820 IF r$="y" THEN GO TO 100
830 IF r$="Y" THEN GO TO 100
840 STOP
900 REM escrever sem espaços em branco a seguir
905 PRINT " ";
910 FOR n=50 TO 1 STEP -1
920 IF p$(n)<>" " THEN GO TO 940
930 NEXT n
940 PRINT p$(TO n);: RETURN
1000 REM obter resposta
1010 INPUT r$: IF r$="" THEN RETURN
1020 LET r$=r$(1): RETURN
2000 REM animais iniciais
2010 DATA "Vive no mar",4,2
2020 DATA "Tem pele escamosa",3,5
2030 DATA "Papa formigas",6,7
2040 DATA "uma baleia","um lagarto","um papa-formigas",
    "uma formiga"
```

Aqui está um programa para desenhar a bandeira da Grã-Bretanha.

```
  5 REM bandeira da gra-bretanha
 10 LET r=2: LET w=7: LET b=1
 20 BORDER 0: PAPER b: INK w: CLS
 30 REM negro no fundo do ecran
 40 INVERSE 1
 50 FOR n=40 TO 0 STEP -8
 60 PLOT PAPER 0;7,n: DRAW PAPER 0;241,0
 70 NEXT n: INVERSE 0
100 REM desenhar nas partes brancas
105 REM São Jorge
110 FOR n=0 TO 7
120 PLOT 104+n,175: DRAW 0,-35
130 PLOT 151-n,175: DRAW 0,-35
140 PLOT 151-n,48: DRAW 0,35
150 PLOT 104+n,48: DRAW 0,35
160 NEXT n
200 FOR n=0 TO 11
210 PLOT 0,139-n: DRAW 111,0
220 PLOT 255,139-n: DRAW -111,0
230 PLOT 255,84+n: DRAW -111,0
240 PLOT 0,84+n: DRAW 111,0
250 NEXT n
300 REM São Andre
310 FOR n=0 TO 35
320 PLOT 1+2*n,175-n: DRAW 32,0
330 PLOT 224-2*n,175-n: DRAW 16,0
340 PLOT 254-2*n,48+n: DRAW -32,0
350 PLOT 17+2*n,48+n: DRAW 16,0
360 NEXT n
370 FOR n=0 TO 19
380 PLOT 185+2*n,140+n: DRAW 32,0
390 PLOT 200+2*n,83-n: DRAW 16,0
400 PLOT 39-2*n,83-n: DRAW 32,0
410 PLOT 54-2*n,140+n: DRAW -16,0
420 NEXT n
425 REM encher o resto dos bits
430 FOR n=0 TO 15
440 PLOT 255,160+n: DRAW 2*n-30,0
```

```
450 PLOT 0,63-n: DRAW 31-2*n,0
460 NEXT n
470 FOR n=0 TO 7
480 PLOT 0,160+n: DRAW 14-2*n,0
485 PLOT 255,63-n: DRAW 2*n-15,0
490 NEXT n
500 REM  tiras vermelhas
510 INVERSE 1
520 REM São Jorge
530 FOR n=96 TO 120 STEP 8
540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
550 NEXT n
560 FOR n=112 TO 136 STEP 8
570 PLOT PAPER r;n,168; DRAW PAPER r;0,-113
580 NEXT n
600 REM São Patricio
610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
690 INVERSE 0: PAPER 0: INK 7
```

Tente agora desenhar a bandeira Portuguesa. As bandeiras tricolores são as mais faceis, ainda que algumas cores - como a cor de laranja para a bandeira Irlandesa - possam apresentar dificuldade. Se fôr Americano pode introduzir também o caracter*.

Aqui está um programa para jogar ao enforcado. Um jogador introduz a palavra e outro adivinha.

```
  5 REM Enforcado
 10 REM Inicializar o ecran
 20 INK 0: PAPER 7: CLS
    30 LET x=240: GO SUB 1000: REM desenhar o homem
     40 PLOT 238,128: DRAW 4,0: REM boca
100 REM colocar a palavra
 110 INPUT w$: REM palavra a adivinhar
```

```
120 LET l=LEN w$: LET v$=" "
130 FOR n=2 TO l: LET v$=v$+" "
140 NEXT n: REM v$=palavra que já foi adivinhada
150 LET c=0: LET d=0: REM contadores de adivinhas
e erros
160 FOR n=0 TO l-1
170 PRINT AT 20,n;"-";
180 NEXT n: REM escrever -'s em vez de letras
200 INPUT "adivinha uma letra: ";g$
210 IF g$="" THEN GO TO 200
220 LET g$=g$(1): REM só a primeira letra
230 PRINT AT 0,c;g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO l: REM corrigir a apalvra adivinhada
260 IF w$(n)=g$ THEN LET v$(n)=g$
270 NEXT n
280 PRINT AT 19,0;v$
290 IF v$=w$ THEN GO TO 500: REM palavra adivinhada
300 IF v$<>u$ THEN GO TO 200: REM adivinha certa
400 REM desenhar parte seguinte da forca
410 IF d=8 THEN GO TO 600: REM enforcado
420 LET d=d+1
430 READ x0,y0,x,y
440 PLOT x0,y0: DRAW x,y
450 GO TO 200
500 REM homem livre
510 OVER 1: REM apagar o homem
520 LET x=240: GO SUB 1000
530 PLOT 238,128: DRAW 4,0: REM boca
540 OVER 0: REM redesenhar o homem
550 LET x=146: GO SUB 1000
560 PLOT 143,129: DRAW 6,0, PI/2: REM sorriso
570 GO TO 800
600 REM enforcar o homem
610 OVER 1: REM apagar o chão
620 PLOT 255,65: DRAW -48,0
630 DRAW 0,-48: REM abrir o alçapão
640 PLOT 238,128: DRAW 4,0: REM apagar a boca
650 REM mover os membros
655 REM braços
```

```
660 PLOT 255,117: DRAW -15,-15: DRAW -15,15
670 OVER 0
680 PLOT 236,81: DRAW 4,21: DRAW 4,-21
690 OVER 1: REM pernas
700 PLOT 255,66: DRAW -15,15: DRAW -15,-15
710 OVER 0
720 PLOT 236,60: DRAW 4,21: DRAW 4,-21
730 PLOT 237,127: DRAW 6,0, -PI/2: REM franzir
740 PRINT AT 19,0;w$
800 INPUT "outra vez? "; a$
810 IF a$="" THEN GO TO 850
820 LET a$=a$(1)
830 IF a$="n" THEN STOP
840 IF a$(1)="N" THEN STOP
850 RESTORE : GO TO 5
1000 REM desenhar o homem na coluna x
1010 REM cabeça
1020 CIRCLE x,132,8
1030 PLOT x+4,134: PLOT x-4,134: PLOT x,131
1040 REM corpo
1050 PLOT x,123: DRAW 0,-20
1055 PLOT x,101: DRAW 0,-19
1060 REM pernas
1070 PLOT x-15,66: DRAW 15,15: DRAW 15,-15
1080 REM braços
1090 PLOT x-15,117: DRAW 15,-15: DRAW 15,15
1100 RETURN
2000 DATA 120,65,135,0,184,65,0,91
2010 DATA 168,65,16,16,184,81,16,-16
2020 DATA 184,156,68,0,184,140,16,16
2030 DATA 204,156,-20,-20,240,156,0,-16
```

APENDICE E

Binário e hexadecimal

Este apendice descreve como é que os computadores contam usando o sistema binário.

A maior parte das línguas Europeias contam usando um padrão mais ou menos regular relativamente às dezenas - em Português, por exemplo, ainda que comece um pouco erraticamente, em breve o padrão se torna regular:

> vinte, vinte e um, vinte e dois... vinte e nove
> trinta, trinta e um, trinta e dois... trinta e nove
> quarenta, quarenta e um, quarente e dois... quarenta e nove

e assim por diante, e isto ainda se torna mais sistemático com a numeração Arabe que usamos. No entanto o único motivo para usar dez é porque acontece que temos dez dedos nas mãos.

Em vez de usar o sistema decimal, que tem dez por base, os computadores usam uma forma de binário chamada de hexadecimal (ou hexa, como abreviatura), baseada em dezasseis. Tal como há dez digitos disponiveis no nosso sistema de numeração precisamos de dezasseis para fazer a contagem em hexadecimal. Assim, alem dos dez já existentes, usamos A,B,C,D,E e F. e o que é que vem depois do F? Tal como nós, com dez dedos escrevemos 1Ø para dez, os computadores escrevem 1Ø para dezasseis. Os seus números começam:

Hexa	Português
Ø	zero
1	um
2	dois
.	.
.	.
9	nove

tal como o nosso, mas depois continua

A	dez
B	onze
C	doze
D	treze

E	catorze
F	quinze
1Ø	dezasseis
11	dezassete
.	.
.	.
19	vinte e cinco
1A	vinte e seis
1B	vinte e sete
.	.
.	.
1F	trinta e um
2Ø	trinta e dois
21	trinta e três
.	.
.	.
9E	cento e cinquenta e oito
9F	cento e cinquenta e nove
AØ	cento e sessenta
A1	cento e sessenta e um
.	.
.	.
B4	cento e oitenta
.	.
.	.
FE	duzentos e cinquenta e quatro
FF	duzentos e cinquenta e cinco
1ØØ	duzentos e cinquenta e seis

Se estiver a usar notação hexadecimal e quiser tornar o facto evidente, então escreva 'h' no fim do número, e diga 'hexa'. Por exemplo, para cento e cinquenta e oito escreva '9Eh' e diga 'nove E hexa'.

Deve estar a perguntar-se o que é que isto tem a ver com computadores. De facto os computadores comportam-se como se só tivessem dois digitos, representados por uma baixa voltagem, ou desligado (Ø), e uma alta voltagem, ou ligado (1). A este chama-se o sistema binário, e os dois digitos binários chamam-se bits: de modo que um bit ou é Ø ou é 1.

A contagem começa assim nos vários sistemas:

Português	Decimal	Hexadecimal	Binário
Zero	Ø	Ø	Ø ou ØØØØ
um	1	1	1 ou ØØØ1
dois	2	2	1Ø ou ØØ1Ø
três	3	3	11 ou ØØ11
quatro	4	4	1ØØ ou Ø1ØØ
cinco	5	5	1Ø1 ou Ø1Ø1
seis	6	6	11Ø ou Ø11Ø
sete	7	7	111 ou Ø111
oito	8	8	1ØØØ
nove	9	9	1ØØ1
dez	1Ø	A	1Ø1Ø
onze	11	B	1Ø11
doze	12	C	11ØØ
treze	13	D	11Ø1
catorze	14	E	111Ø
quinze	15	F	1111
dezasseis	16	1Ø	1ØØØØ

O importante é que dezasseis é igual a dois à quarta, e isto torna a conversão entre hexa e binário muito fácil.

Para converter hexa para binário, troca-se cada digito hexa por quatro bits, usando a tabela acima.

Para converter binário para hexa, divide-se o número binário em grupos de quatro bits, começando pela direita, e depois troca-se cada grupo pelo correspondente digito hexa.

Por este motivo, ainda que estritameute falando os computadores sejam sistemas puramente binário, os humanos muitas vezes escrevem os números que estão guardados no computador em notação hexa.

Os bits dentro do computador estão essencialmente agrupados em grupos de oito, ou bytes. Um único byte pode representar um número entre zero e duzentos e cinquenta e cinco (o binário 11111111 ou FF hexa), ou alternativamente um caracter qualquer no conjunto de caracteres do Spectrum. O seu valor pode ser escrito com dois digitos hexa.

Dois bytes podem ser agrupados para fazer o que se chama em linguagem tecnica uma palavra. Uma palavra pode ser escrita usando dezasseis bits ou quatro digitos hexa, e representa um número de Ø até (em décimal) $2^{16}-1=65535$

Um byte são sempre oito bits, mas as palavras variam de comprimento de computador para computador.

A notação BIN no Capítulo 14 fornece um meio de escrever números em binário no ZX Spectrum: 'BIN Ø' representa zero, 'BIN 1' representa um, 'BIN 1Ø' representa dois, e assim por diante.

Só se pode usar Ø's e 1's para isto, de modo que o número tem de ser inteiro e positivo; por exemplo, não se pode escrever 'BIN -11' para representar -3 - em vez disso tem de se escrever '-BIN11'. O número também não pode ser maior que o decimal 65535 - isto é, não pode ter mais do que dezasseis bits.

ATTR é na verdade binário. Se converter o resultado dado por ATTR para binário, pode escreve-lo em oito bits.

O primeiro é 1 para um campo a piscar, Ø se não estiver.

O segundo é 1 para brilho acentuado, Ø para normal.

Os três seguintes são o código para a cor do papel, em binário.

Os últimos três são o código para a tinta, em binário.

Os códigos de cor tambem usam binário: cada código escrito em binário pode ser escrito em três bits, o primeiro para verde, o segundo para vermelho e o terceiro para o azúl.

O negro não tem luz, de modo que todos os bits são Ø (desligado). Assim o código para negro é ØØØ, em binário, ou zero.

As cores puras, verde, vermelho e azul, só têm um bit 1 (ligado) nos três. Os seus códigos são 1ØØ, Ø1Ø, e ØØ1, em binário, ou seja quatro, dois e um.

As outras cores são misturas destas, de modo que os seus códigos em binário têm dois ou mais bits 1.

INDICE:

- Prefacio: O computador e a sua instalação Pag. 1
- Capitulo 1: Introdução Pag. 31
- Capitulo 2: Conceitos de programação basicos Pag. 35
- Capitulo 3: Decisões (IF, STOP, etc) Pag. 47
- Capitulo 4: Ciclos (FOR NEXT, TO, STEP) Pag. 50
- Capitulo 5: Subrotinas (GO SUB, RETURN) Pag. 55
- Capitulo 6: READ, DATA, RESTORE Pag. 57
- Capitulo 7: Expressões (Operações, notação cientifica, etc) Pag. 69
- Capitulo 8: Cadeias de letras (STRINGS) Pag. 64
- Capitulo 9: Funções (DEF, LEN, STR$, YAL, SGN, INT, SQR, FN)Pag. 67
- Capitulo 10: Funções Matemáticas (^, PI, EXP,LN,SIN,COS, Etc)Pag. 74
- Capitulo 11: Números Aleatórios (RANDOMIZE,RND) Pag. 81
- Capitulo 12: Matrizes (DIM) Pag. 85
- Capitulo 13: Condições (AND,OR,NOT) Pag. 89
- Capitulo 14: Conjunto de caracteres (CODE,CHR$,POKE,PEEK,USR)Pag. 94
- Capitulo 15: Mais sobre PRINT e INPUT (TAB,AT,LINE,CLS) Pag. 102
- Capitulo 16: Cores (INK,PAPER,BORDER,FLASH,BRIGHT,INVERSE) Pag. 110
- Capitulo 17: Gráficos (PLOT,DRAW,CIRCLE,POINT) Pag. 121
- Capitulo 18: Movimento (PAUSE,INKEY$,PEEK) Pag. 128
- Capitulo 19: BEEP-Som no SPECTRUM Pag. 133
- Capitulo 20: Armazenamento em fita (LOAD,SAVE,VERIFY,MERGE) Pag. 138
- Capitulo 21: Impressora ZX (LLIST,LPRINT,COPY) Pag. 148
- Capitulo 22: Outros Equipamentos Pag. 150
- Capitulo 23: IN,OUT Pag. 151
- Capitulo 24: A Memória (CLEAR) Pag. 154
- Capitulo 25: As variáveis do sistema Pag. 162
- Capitulo 26: Usar código de máquina (USR) Pag. 168
- APENDICES:
    - ★ A-O Conjunto de ceracteres Pag. 171
    - ★ B-Mensagens Pag. 179
    - ★ C-Uma descrição do ZX SPECTRUM Pag. 184
      -O BASIC Pag. 190
    - ★ D-Programas de exemplo Pag. 204
    - ★ E-Binario e Hexadecimal Pag. 213

ZX SPECTRUM

LANDRY-Engenheiros Consultores, Lda.
Rua Tomás da Anunciação, 53-A
Telefones: 68 13 43 - 68 13 44 - 68 68 37
1300 LISBOA Tlx. 43436 COMPUT P

Sinclair Research Limited
6 King's Parade,
Cambridge CB2 1SN
England