sinclair
ZX81

MANUAL DE PROGRAMAÇÃO BASIC

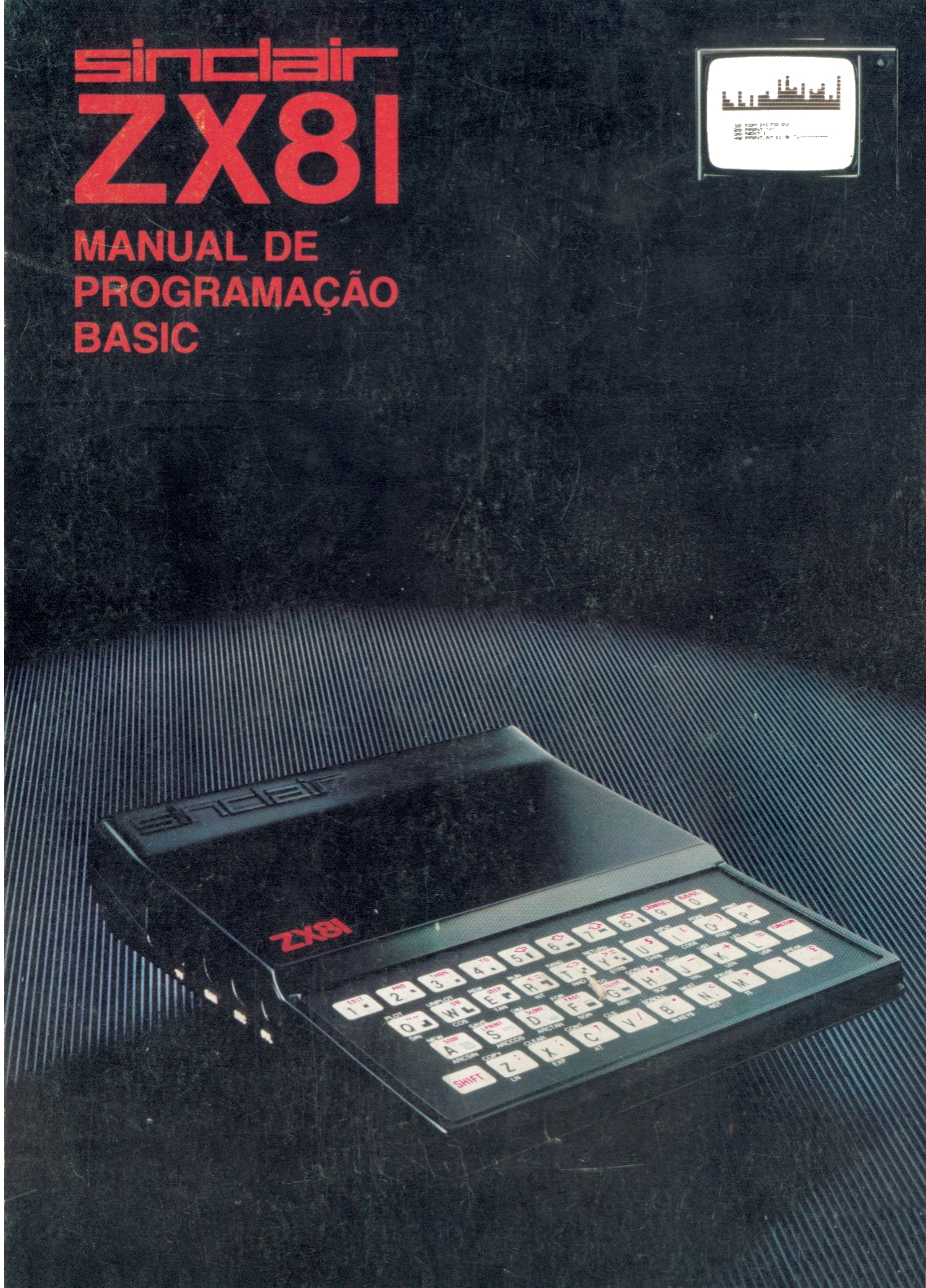
SINCLAIR RESEARCH LIMITED
6 KING'S PARADE
CAMBRIDGE CB2 1SN
ENGLAND

Representante em Portugal
LANDRY ENGENHEIROS CONSULTORES, LDA.
Rua Coelho da Rocha, 57-3.º Dto.
1300 Lisboa

1981

sinclair ZX81

MANUAL DE PROGRAMAÇÃO BASIC



sinclair

ZX81

PERSONAL COMPUTER

MANUAL DE PROGRAMAÇÃO BASIC

1.ª Edição portuguesa
traduzido da 1.ª edição
do "ZX81 BASIC PROGRAMMING"
DE STEVEN VICKERS.
Por LANDRY ENGENHEIROS
CONSULTORES, LDA.

Novembro de 1981

10 000 EXEMPLARES

PREFÁCIO À 1.ª EDIÇÃO PORTUGUESA

O SINCLAIR ZX81 foi, sem dúvida, o produto que, vindo da Investigação, teve a mais rápida divulgação em todo o Mundo.

A tradução do Manual BASIC não foi um empreendimento tão fácil como vai ser para si aprender a trabalhar com o ZX81.

Tentou-se traduzir o Inglês para um Português acessível, utilizando vocábulos da língua Portuguesa em substituição de palavras consagradas em Inglês o que, muitas vezes, se tornou impossível e, desse modo, foi obrigatória a sua utilização na língua original.

Queremos frisar que não estamos alheios ao "perigo" que é o de traduzir uma obra destas para Português, mas era imprescindível pois é praticamente apenas uma élite da população que domina o Inglês à altura de poder estudar nessa língua.

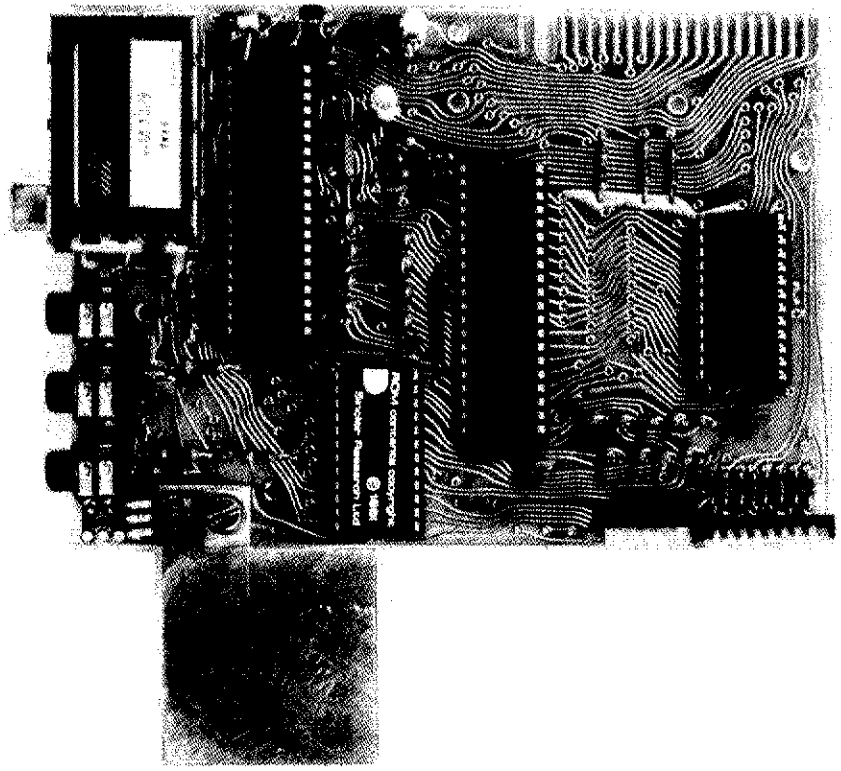
Esperamos que o Vosso Computador ZX81 lhe vá agradar inteiramente, e tanto mais quanto melhor o souber utilizar.

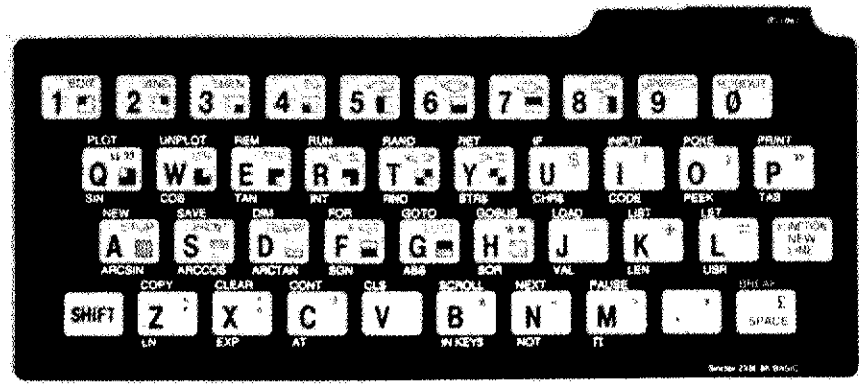
Queremos também salientar que tivémos a ousadia de mandar imprimir 10 000 exemplares, porque estamos convictos que este computador vai alterar em muito os hábitos dos Portugueses, contribuindo assim para o seu desenvolvimento e progresso.

LANDRY — ENGENHEIROS CONSULTORES, LDA.

O Sócio Gerente

CARLOS MANUEL MAIA NOGUEIRA





Capítulo 1

Sintonizar o ZX81

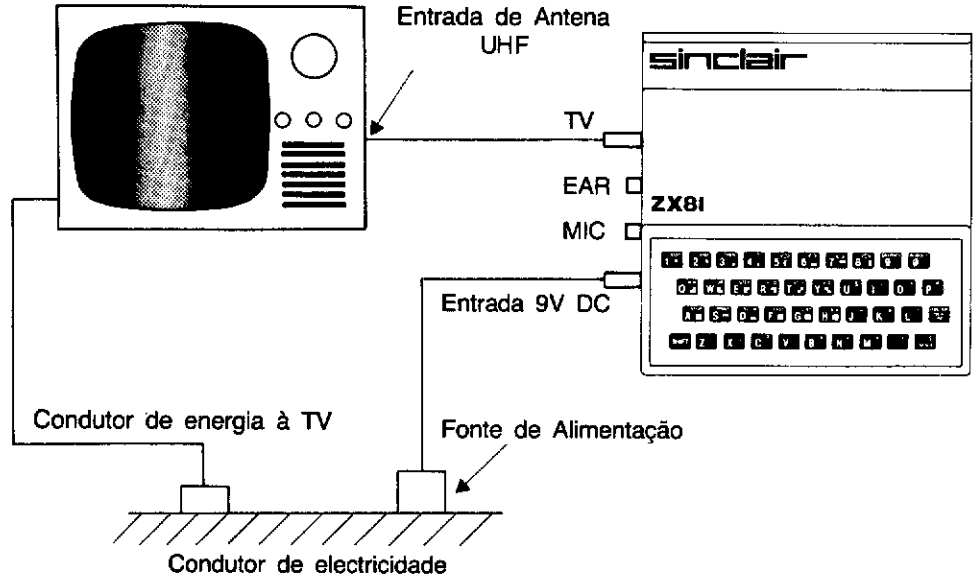
Ao desembrulhar o ZX81, encontrará:

1. Este Manual.

2. O Computador, que é composto por: 3 entradas de jack (marcadas 9V DC IN, EAR e MIC), 1 entrada para o jack da antena e uma parte que sobressai do quadro do circuito onde se pode ligar o equipamento extra. Não tem botões — para o pôr a funcionar, basta ligá-lo à corrente.

3. Fonte de Alimentação. (Opcional)
   Este transformador converte a electricidade na forma usada pelo ZX81. Se, por engano, o meter na entrada errada do computador, não haverá qualquer problema. Se quiser utilizar a sua própria fonte de alimentação, esta deverá fornecer 9 volts DC a 700 mA sem ser regulada, e acabar num jack de 3,5 mm com extremidade positiva.

4. Um cabo de antena com 120 cm de comprimento, que liga o computador à televisão.



5. Um par de cabos com cerca de 30 cm de comprimento, com jacks de 3,5 mm em ambas as pontas. Estes ligam o computador a um gravador.

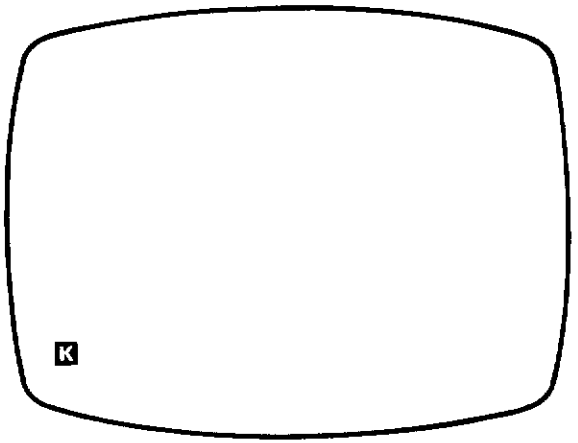
Necessitará também de um aparelho de TV — o ZX81 poderá trabalhar sem ele, mas não será possível ver o que o ZX81 está a fazer! O TV deve ter UHF — se não estiver preparado para receber a RTP-2, não serve.

Precisará, mais tarde, de um gravador de cassettes. Este é necessário porque, quando se desliga o ZX81, perdem-se para sempre todas as informações nele armazenadas. A única forma de as poder utilizar mais tarde será gravá-las numa cassette — poderá saber como fazê-lo no Capítulo 16. Pode também comprar cassettes já programadas.

Quando tiver tudo (excepto o gravador), ligue como mostra a página anterior.

Se o TV tiver duas entradas de antena, marcadas UHF e VHF, então use o UHF.

Ligue o aparelho e o TV. Precisa agora de regular o aparelho de TV. O ZX81 funciona no canal 36 UHF e quando é ligado pela primeira vez e devidamente regulado, dá uma imagem como esta:



Ao utilizar o computador, desejará certamente baixar o volume.

Se o seu TV tiver um controle de regulação, então terá que o regular até conseguir esta imagem. Há já muitos TVs que têm um botão individual que se carrega para cada estação. Escolha um que não costume utilizar e carregue-o.

Se ficar confuso com o computador, lembre-se que o poderá sempre acertar e voltar a esta imagem desligando a ficha da entrada 9 DC e voltando a ligá-la. Isto em último recurso, porque irá perder todas as informações que deu ao computador.

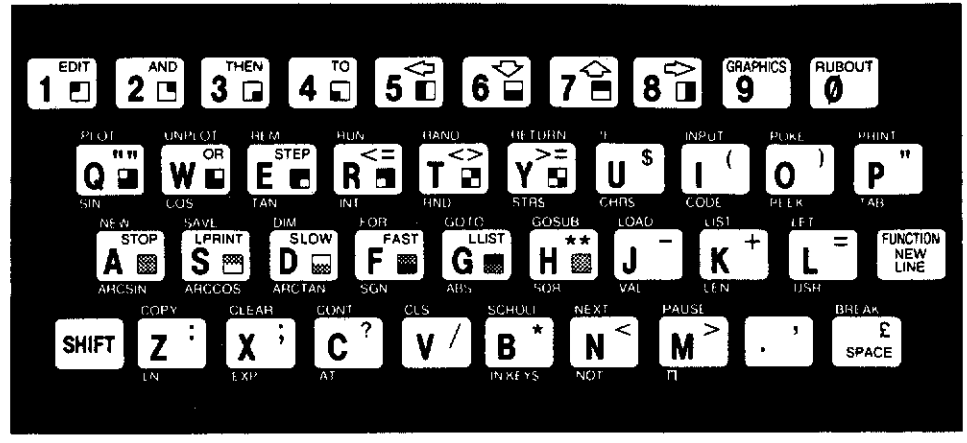
Nota: Esta descrição do TV aplica-se a Portugal, Inglaterra e na maior parte dos Países da Europa Ocidental (excepto na França), onde há um sistema de UHF que usa 625 linhas com 50 imagens por segundo. Os E.U.A. utilizam o VHF e 525 linhas com 60 imagens por segundo.

Agora que pôs o computador a funcionar, poderá passar ao seu funcionamento propriamente dito. Se já conhece a linguagem de computadores BASIC, leia o apêndice C e utilize o resto do Manual apenas na clarificação de pontos obscuros.

Se é um principiante, então a parte principal do Manual foi escrita para si. Não ignore os exercícios; muitos deles levantam importantes questões que não são debatidas no texto. Analise-os e faça o que mais lhe aprouver, ou que lhe pareça entrar num campo que não conheça bem.

Faça o que fizer, continue a utilizar o computador. Se puser uma questão "O que é que ele faz se lhe ordenar isto ou aquilo?" Então a resposta é fácil: ordene e veja. Sempre que o Manual lhe diz para escrever algo, interrogue-se sempre, "O que é que poderia fazer em vez disto?", e tente as suas respostas. Quanto mais dados pessoais escrever, mais ficará a compreender o ZX81 (é o que se chama de conhecimento não programado). Mesmo que não preste atenção àquilo que escreve, nunca pode danificar o computador.

TRADUÇÃO DO TECLADO DO



EDIT — Editar

AND — E

THEN — Então

TO — Até

GRAPHICS — Gráficos

RUBOUT — Apagar

PLOT — Pôr um ponto em X, Y

SIN — Seno

UNPLOT — Tirar um ponto de X, Y

COS — Coseno

REM — Comentário

TAN — Tangente

RUN — Executar

INT — Inteiro; anula a parte decimal de um número

RAND — Inicialização da rotina RND

RETURN — Instrução de fim e retorno de subrotinas

STR$ — "String"; transforma um valor numérico em alfanumérico

IF — Se

CHR$ — Manda imprimir um caracter através de um número.

INPUT — Entre

CODE — Código; número decimal do caracter

POKE — Dar um valor (hexadecimal) a um Byte determinado

PEEK — Mostra o valor de um determinado Byte

PRINT — Escreva

TAB — Tablação

NEW — Novo programa

ARCSIN — Arco de Seno

SAVE — Guardar

ARCCOS — Arco de Coseno

DIM — Dimensionar

FOR — Para

SGN — Sinal do argumento de um número

GOTO — Vá para a linha...

ABS — Valor absoluto; torna o argumento positivo, (módulo).

GOSUB — Vá para a subrotina que começa na linha...

SQR — Raiz quadrada

LOAD — Chamar o programa

VAL — Valor da variável alfanumérica

LIST — Listar

LEN — Número de caracteres da variável alfanumérica

LET — Faça

USR — Chama a subsrotina linguagem máquina

FUNCTION — Função

NEWLINE — Nova linha

STOP — Pára o programa

OR — Ou

STEP — Passo (incremento)

LPRINT — Escrever na impressora

SLOW — Lento

FAST — Rápido

LLIST — Listar na impressora

** — Elevado a...

SHIFT

COPY — Copie o écran

LN — Logaritmo natural

CLEAR — Limpe as memórias

EXP — Expoente

CONT — Continue

AT — Em

CLS — Limpe o écran

SCROLL — Mova todo o écran uma linha para cima

INKEY$ — Leitura do teclado

NEXT — Fim de ciclo FOR

NOT — Não

PAUSE — Pausa

π — Pi

BREAK — Suspensão do programa

SPACE — Espaço

RND — Número casual ou aleatório

Vocábulos

AGULHA — Tal como a agulha de um comboio, só pode tomar dois valores: 0 (baixa ou reset), 1 (alta ou set), servindo algumas vezes como indicadora do estado parcial de um processamento.

ARGUMENTO — 1. Variável ou expressão a que uma função é aplicada. 2. A parte de vírgula fixa dum número representado em vírgula flutuante.

"ARRAY" — Conjunto de elementos (registos, números, etc.) organizados de acordo com uma configuração unidimensional ou multidimensional.

ASSEMBLAGEM (ASSEMBLY) — A operação de tradução que é efectuada sobre uma linguagem simbólica de programação, para produzir um programa equivalente traduzido em linguagem máquina.

ATRIBUIÇÃO — De um determinado valor z a um determinado conjunto de funções (nomeadamente variáveis).

BIT — Abreviatura de BInary DigiT (dígito binário), isto é, cada um dos dígitos (0 e 1) usados na notação binária (8 bits constituem 1 byte).

BYTE — (Posição alfanumérica; octeto) — Um conjunto de dígitos binários operados como uma unidade, em regra um conjunto de 8 bits e que comporta, quer a armazenagem de um caracter alfanumérico, quer dois dígitos. Dum modo genérico, utiliza-se para especificar uma posição de armazenagem de 6, 7 ou 8 bits. Neste último caso designa-se por octeto.

CPU — Abreviatura de Central Processing Unit (Unidade Central de Processamento). Num microcomputador, é a parte que contém a unidade aritmética e lógica (ALU), a unidade de controlo e grupos de registos. Realiza operações aritméticas e lógicas, controla o processamento das instruções, armazena temporariamente a informação e fornece os sinais de temporização e outros ao resto do sistema.

DADOS (DATA) — Informação processada ou a processar por um CPU.

ENDEREÇO — 1. A parte de uma instrução que especifica a localização real de um registo de memória onde é possível registar ou recuperar dados ou informação. 2. Uma etiqueta, nome, designação ou número, que se atribui à localização de: palavra, registo, posição da memória, campo ou byte de acordo com a organização interna do computador em causa.

EXPOENTE — A potência à qual uma quantidade é elevada, ou seja, a indicação do número de vezes que a base é repetida ou actua como factor. Na aritmética de representação em vírgula flutuante, o expoente é uma das partes essenciais da representação de qualquer número. Ver: Número em Vírgula Flutuante.

Ex.: $3{,}943 \times 10^{9} < = > 3.943\ E9$

INTRODUÇÃO (INPUT) — Instrução que permite a entrada de dados.

ITEM — Unidade de informação, artigo, elemento; os campos de um dado tipo (ou conjunto de campos), que se relacionam entre si e dizem respeito ao mesmo facto, objecto, acontecimento ou operação.

MANTISSA — A parte fraccionária positiva de um logaritmo, embora se utilize para referir a parte decimal de qualquer número em vírgula flutuante. Ver: Expoente.

MATRIZ — Uma rede de números, em regra sob forma rectangular, que podem ser operados de forma a envolver operações matemáticas ou lógicas de tipo corrente, tais como adição, multiplicação, inversão, etc. Duma forma lata, o termo aplica-se a qualquer tabela de elementos (numéricos ou alfanuméricos).

NÚMERO EM VÍRGULA FLUTUANTE — Notação numérica em que é variável a posição da vírgula decimal. A representação dos números neste sistema é feita através de dois números, x e y, tais que $a = x.b^{y}$ e em que x é a mantissa, y o expoente e b a base de representação do número (binário, octal, decimal, etc.). Este sistema é muito utilizado porque a armazenagem dos números é mais económica, é maior a precisão dos cálculos a efectuar, etc.

PILHA (STACK) — Uma sequência de registos e/ou posições de memória, usada no modo "Last In/First Out" (a última informação a entrar é a primeira a sair), frequentemente utilizada para salvaguardar o conteúdo de registos antes da execução de uma subrotina.

POINTER — Registo do microprocessador que contém o endereço de uma posição de memória.

PROJECÇÃO (DISPLAY) — A operação que consiste em enviar a informação, em geral através de mensagens adequadas, para um equipamento de saída, para inspecção visual pelo operador ou utilizador. A projecção é feita no tubo cinescópio de raios catódicos, no ZX81 é normalmente a TV.

PROM — Sigla de Programmable Read Only Memories (Memórias programadas só para leitura.

PULSO (FRAME) — Conjunto de elementos que constituem uma informação completa, que dão informação suficiente para se construir a imagem na televisão. Nomeadamente em Portugal, a televisão utiliza 50 desses pulsos por segundo.

RAM — Sigla de Random Access Memory. Memória a que se tem acesso tanto para ler como para programar (memória viva).

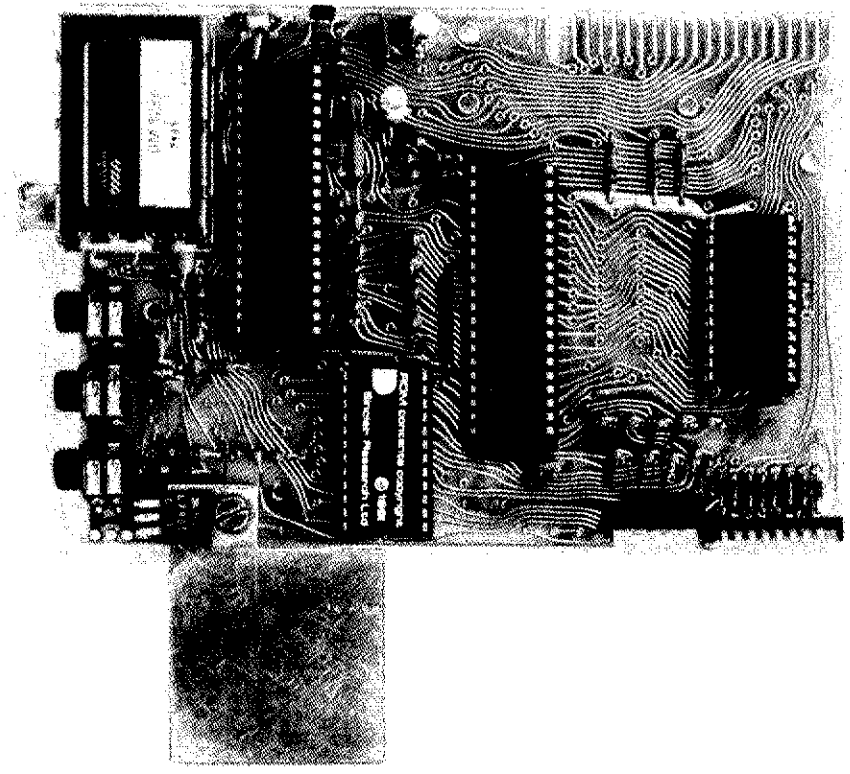
RAMTOP — Limite possível de ser estabelecido (pelo conjunto de instruções POC) numa zona de memória RAM, declarando que o compilador BASIC não pode escrever abaixo dele.

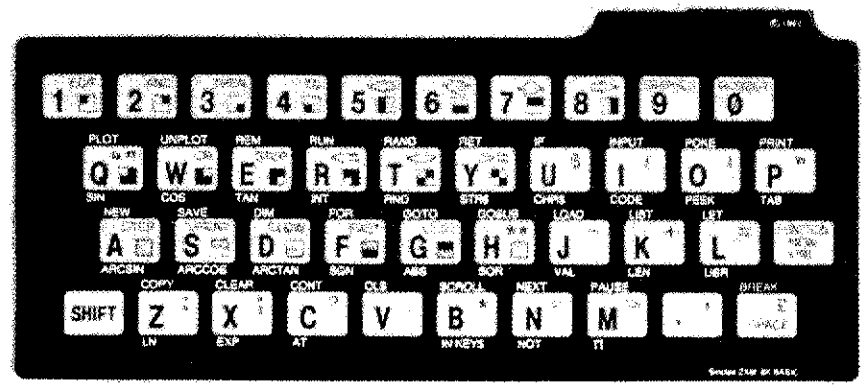
ROM — Sigla de Read Only Memory. Memória pré-programada. Apenas é permitido lê-la ou executá-la. Tipo de memória cujo conteúdo é fixo, em geral já vindo de fábrica e que não é susceptível de alteração.

ROTINA — Um procedimento "Software", representado por um conjunto de instruções básicas, destinadas à realização de uma função específica e bem definida.

"STRING" — Conjunto ordenado de elementos alfanuméricos, onde existe a possibilidade de tratar cada um deles individualmente ou em conjunto, tal como uma matriz.

SUBROTINA — As subrotinas são programas concebidos de tal forma que se bastem a si próprios e de modo a resolver um dado problema específico. Uma subrotina é constituída de forma tal que, quando é executada, existe uma instrução de retorno (RETURN) que permite o regresso à instrução que se segue imediatamente à chamada é a subrotina.




1 2 3 4 5 6 7 8 9 0
PLOT UNPLOT REM RUN RAND RET IF INPUT POKE PRINT
Q W E R T Y U I O P
SIN COS TAN INT RND STR$ CHR$ CODE PEEK TAB
NEW SAVE DIM FOR GOTO GOSUB LOAD LIST LET
A S D F G H J K L
ARCSIN ARCCOS ARCTAN SGN ABS SQR VAL LEN LEN
COPY CLEAR CONT DEF SCROLL NEXT PAUSE
SHIFT Z X C V B N M
LN EXP AT INKEY$ NOT PI
BREAK
SPACE

Capítulo 2

Programar o computador

Ligue o computador (metendo-lhe a ficha) e obtenha o écran branco com o caracter inverso vídeo K, como na imagem do Capítulo 1. Para que algo lhe apareça, tem que escrever uma mensagem que ele compreenda; como, por exemplo, a mensagem

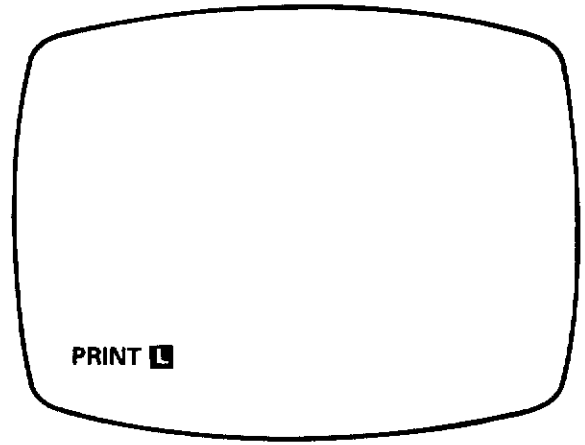
PRINT 2+2

ordena-lhe que resolva a soma 2+2 e escreva a resposta no écran de TV.

Uma mensagem como esta, dizendo ao computador para fazer algo de concreto, chama-se uma ordem; esta é uma ordem **PRINT** (escreva), mas é também uma instrução **PRINT.** Ao chamar-lhe uma instrução **PRINT,** estamos apenas a especificar a sua forma sem mencionarmos como o computador a vai utilizar. Assim, toda a ordem toma a forma de instrução, mas também faz outras coisas — programa as linhas, como veremos no Capítulo 8.

Para escrever esta ordem,

1. Primeiro escreva **PRINT.** Mas, como pode ver, o teclado tem uma tecla para cada ordem, não se soletra a palavra P, R, I, N, T. Logo que prima o P, aparecer-lhe-á no écran a palavra completa, juntamente com um espaço para dar melhor aspecto, e o écran aparecerá assim:



A explicação para isto é que, no princípio de cada ordem, o computador espera uma palavra-chave — uma palavra que especifica qual o género de ordem. As palavras-chave estão escritas por cima das teclas, e pode verificar-se que **'PRINT'** aparece por cima da tecla P de modo a que, para obtermos **'PRINT'** , prime-se P.

O computador faz-lhe saber que espera uma palavra-chave pela letra K com que tem de começar. Há quase sempre uma letra inverso vídeo, seja K ou L (ou, como

veremos adiante, F ou G), chamada cursor. A letra K significa "seja qual for a tecla que primir, interpretá-la-ei como palavra-chave". Como se verificou, depois de ter primido P para **PRINT**, a letra K passou para L.

Este sistema de primir só uma tecla para nos dar mais que um símbolo é muito utilizado no ZX81. No resto do Manual, as palavras com tecla própria são impressas em Negrito (**BOLD TYPE**).

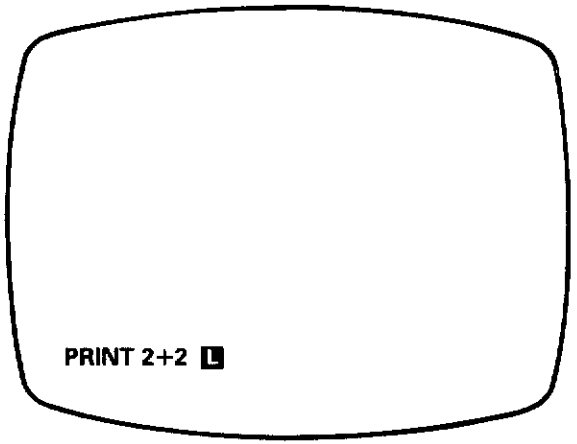
Deve-se ter sempre em conta que não adianta tentar soletrar estas palavras por completo porque o computador não as compreenderá.

2. Agora escreva 2. Não haverá qualquer problema. O 2 deverá aparecer novamente no écran e o L anda um espaço.

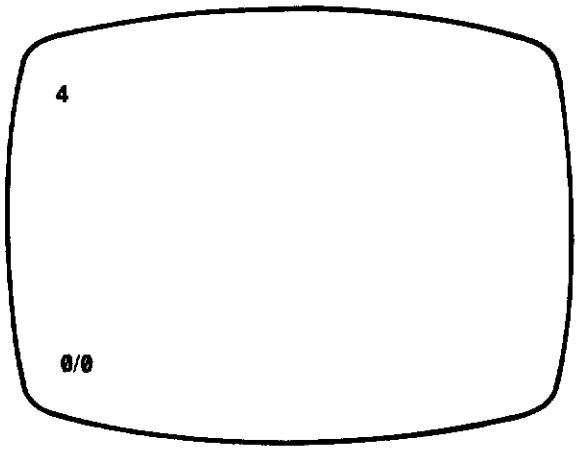
Note-se também a forma como aparece automaticamente um espaço entre **PRINT** e 2 para sobressair do resto da instrução. Isto é feito tantas vezes quanto possível, de modo a que raramente se tenha que dar um espaço. Se der um espaço, este aparecerá no écran mas não afectará de modo algum o significado da mensagem.

3. Agora, prima o sinal +. Para ter acesso a esta instrução, necessita carregar na tecla **SHIFT**, enquanto prime a tecla [K+] (LIST / LEN) Do mesmo modo, poderá ter acesso a todas as informações a vermelho dos cantos superiores direitos de cada tecla da mesma cor que o [SHIFT].

4. Prima novamente 2. O écran aparecerá assim:



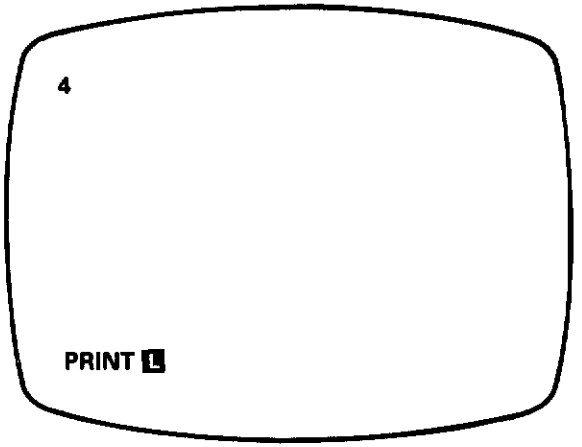
5. Agora — deve ter isto sempre em conta — prima **NEWLINE**, a tecla [FUNCTION / NEW LINE]. Isto significa "mensagem completa" ou "O.K., computador, vejamos o que fazes". O computador lerá agora a mensagem, trabalha o que tem de ser feito, e resolve-a. Neste caso, o écran mudará para



A resposta é 4 — mas é claro que não necessita comprar um computador para chegar a esta conclusão.

0/0 (Note-se como o zero está escrito com uma barra para se distinguir do O maiúsculo. Isto é vulgar nos circuitos de programação.) É um relatório através do qual o computador lhe diz as dificuldades que teve. O primeiro 0 significa "O.K., não há problemas". (No apêndice B há uma lista dos outros códigos de relatório que podem aparecer quando, por exemplo, há algo de errado). O segundo 0 signifca "A última coisa que fiz, foi a linha 0". Poder-se-á verificar mais tarde — quando começar a escrever programas — que se pode dar um número a uma instrução e armazená-la para a voltar a executar mais tarde: isto é uma linha de programa. As ordens não têm números mas, para fins de informação, o computador pretende que sejam a linha 0.

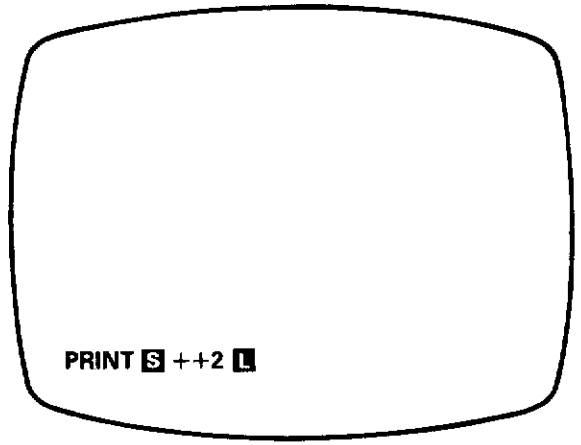
Deve imaginar que este relatório em código oculta o cursor K — se primir P para **PRINT**, nesse momento a informação desaparecerá e o écran mudará para



O cursor também pode ser utilizado para corrigir erros: prima ++2, para obter

PRINT ++2 L

na última linha. São caracteres um tanto ou quanto incompreensíveis, e quando prime **NEWLINE** obtém



O S é o marcador do erro de sintaxe (a sintaxe é a gramática das mensagens, referindo as que são permitidas e as que o não são), e mostra que o computador chegou até '**PRINT**', mas depois disso decidiu que a mensagem não estava correcta.

O que se pretende fazer é apagar o primeiro +, e substituí-lo por, digamos, 3. Primeiro, tem que se mudar o cursor até este estar à direita do primeiro +; há duas teclas, ◊ e ◊ (SHIFT 5 e SHIFT 8), que deslocam o cursor para a direita e para a esquerda. Continuando a primir **SHIFT**, prima a tecla ◊ duas vezes. Isto desloca o cursor para a esquerda duas casas, o que lhe dá

PRINT + L +2

Prima agora a tecla **RUBOUT** (SHIFT Ø), e obterá

PRINT L +2

RUBOUT apaga o caracter (ou palavra-chave) à esquerda do cursor.

Se escrever agora 3, introduzirá '3', novamente à esquerda do cursor, dando

PRINT 3 L +2

e primindo **NEWLINE** dá a resposta (5).

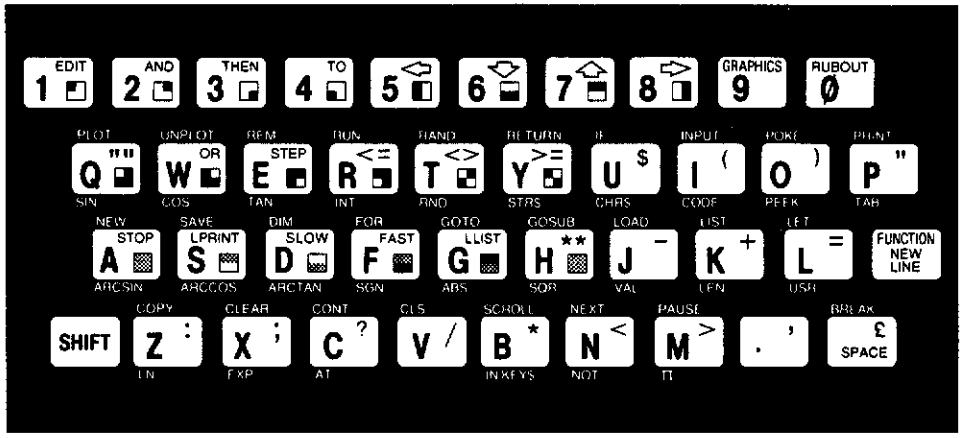
A tecla ◊ (primindo simultaneamente SHIFT e 8) trabalha como a tecla ◊, com a excepção de que move o cursor para a direita em vez de ser para a esquerda.

Sumário

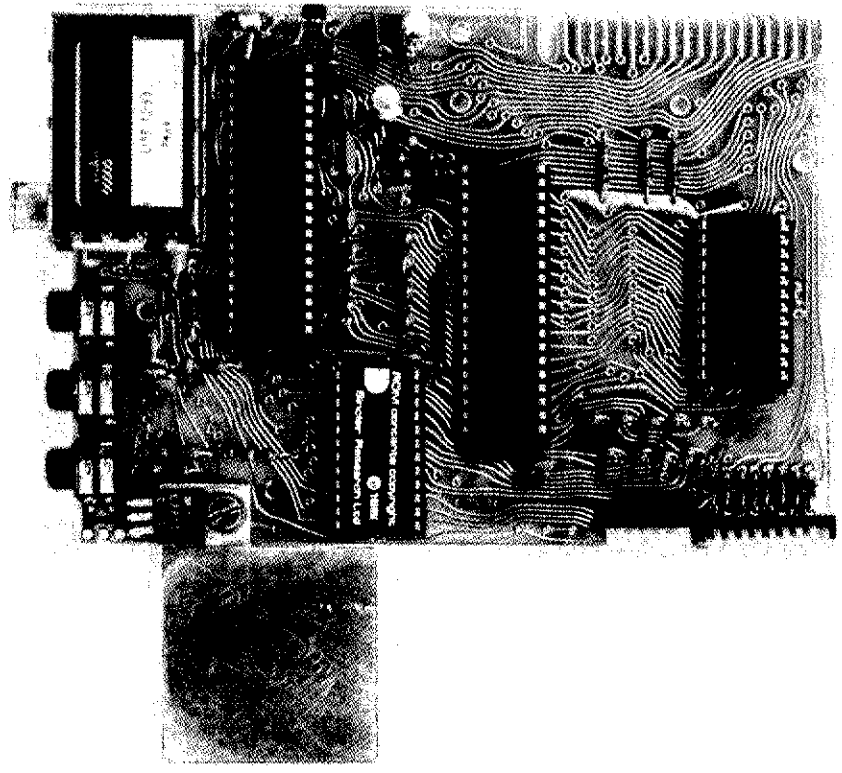
Este capítulo debruçou-se sobre a forma de escrever mensagens no ZX81, explicando o sistema de tecla-única para as palavras-chave,
os cursores K e L,
relatórios
o marcador de erros de sintaxe,
e como corrigir os erros utilizando ◊, ◊ e **RUBOUT.**

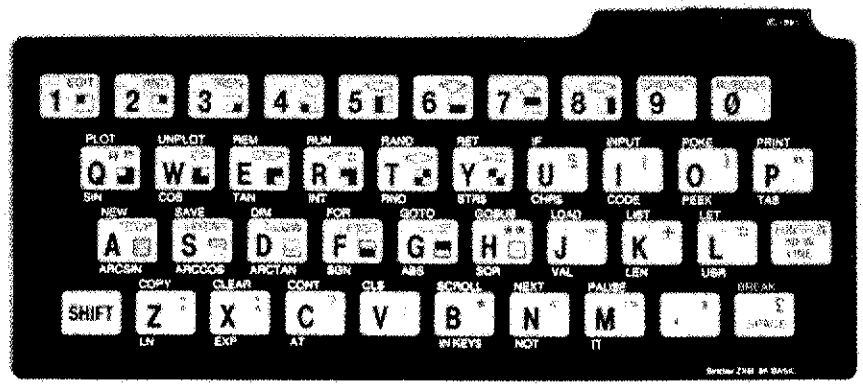
O teclado

Mostramos a seguir uma figra do teclado.



Note-se que para utilizar **SHIFT**, tem que primi-lo ao mesmo tempo que prime outra tecla. Não confundir o dígito Ø com a letra O.





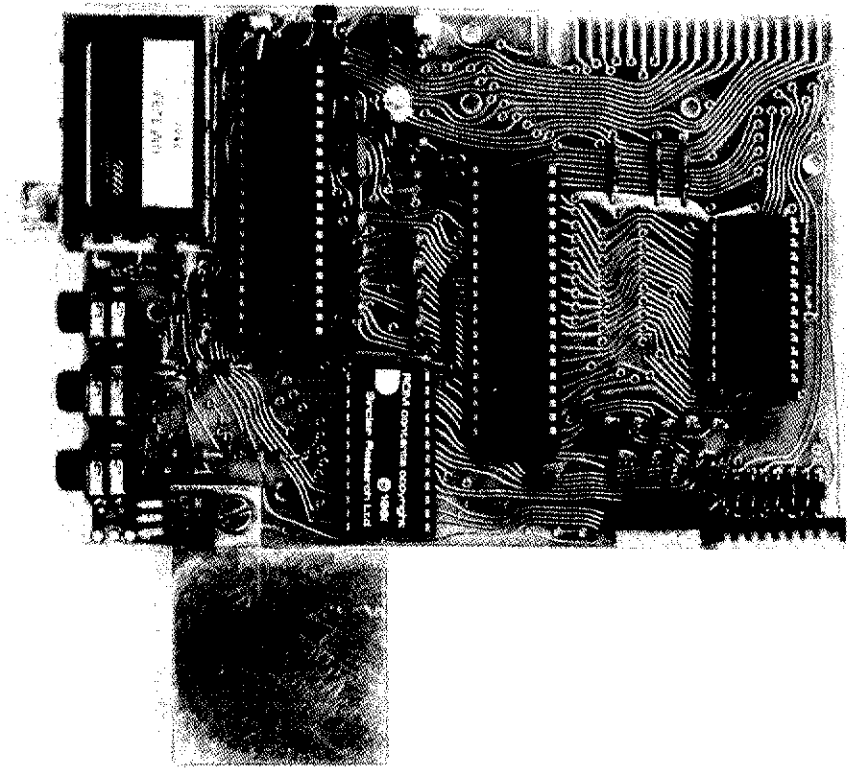
Capítulo 3

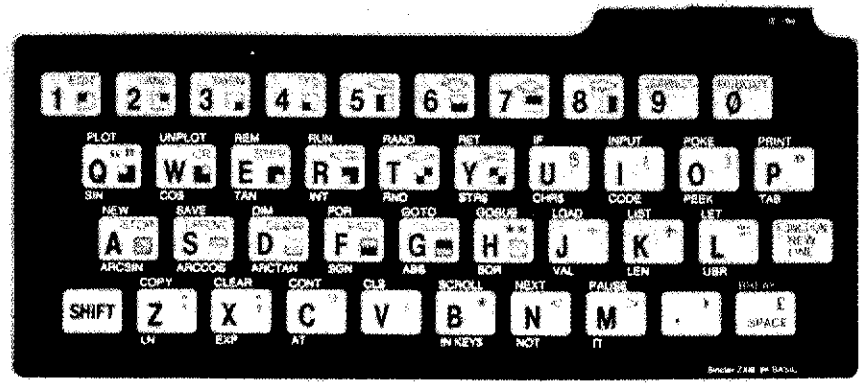
Uma lição de História

As mensagens que dá ao computador denominam-se, na linguagem de computadores, BASIC (que significa Código de Instrução Simbólica para todos os objectivos dos Principiantes). Exige do computador um esforço enorme a decifrar as mensagens BASIC para as suas operações rudimentares mas, de facto, é o que se exige deles. As mensagens BASIC contêm bastantes palavras inglesas (como **PRINT**) para se tornar mais fácil a sua apreensão para uma pessoa que saiba Inglês.

A linguagem BASIC foi concebida no Dartmouth College em New Hampshire, E.U.A., em 1964, e desde então tem sido a linguagem de computadores mais utilizada por principiantes e amadores. E isto porque se adapta muito bem à utilização "on-line" em que o utente escreve algo e o computador responde imediatamente. Há outras linguagens — tais como o ALGOL (que é, de facto, uma família de ALGOLS) e PASCAL — com uma estrutura mais clara e maior poder que o BASIC, mas só alguns — tais como os relativamente pouco conhecidos APL e POP-2 — são igualmente fáceis de utilizar "on-line". Ainda podemos mencionar outras como FORTRAN, PL 1 e COBOL.

Muitas revistas de computadores pessoais publicam já programas em BASIC, e que merecem ser lidas como fontes de ideias. Certamente que as terão de adaptar ligeiramente porque qualquer computador que utilize a linguagem BASIC tem o seu dialecto próprio, diferente de todos os outros.




1 2 3 4 5 6 7 8 9 0
PLOT UNPLOT REM RUN RAND RET IF INPUT POKE PRINT
Q W E R T Y U I O P
SIN COS TAN INT RND STR$ CHR$ CODE PEEK TAB
NEW SAVE DIM FOR GOTO GOSUB LOAD LIST LET
A S D F G H J K L
ARCSIN ARCCOS ARCTAN SGN ABS SQR VAL LEN USR
COPY CLEAR CONT CLS SCROLL NEXT PAUSE
SHIFT Z X C V B N M . SPACE
LN EXP AT IN KEYS NOT IT

Capítulo 4

O Sinclair ZX81 como calculadora

Ligue o computador. Pode agora utilizá-lo como calculadora, como está descrito no Capítulo 2; escreva **PRINT**, depois o que quer que deseje trabalhar, e depois **NEWLINE**. (Não nos incomodaremos mais a dizer-lhe para fazer **NEWLINE**).

Como era de esperar, o ZX81 pode não só somar mas também subtrair, multiplicar (usando o asterisco * em vez do sinal de vezes — isto é muito comum em computadores) e dividir (utilizando / em vez de ÷). Tente fazer algumas contas.

+, —, * e / são operações, e os números com que operam são os operandos.

O computador pode também elevar um número à potência de outro utilizando a operação ** (shift H. Não escreva * — primindo simultaneamente shift e a tecla B — duas vezes): escreva

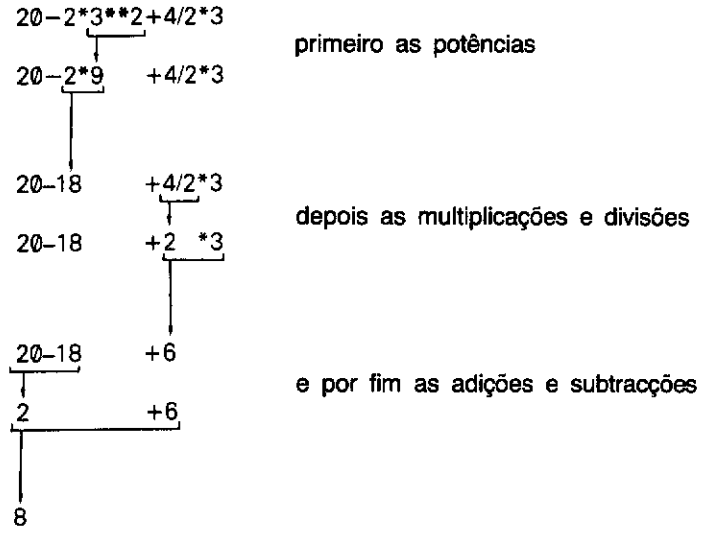
PRINT 2**3 (lembre-se de fazer **NEWLINE**)

e terá a resposta 8 (2 elevado à potência 3, ou 2^3, ou 2 ao cubo).

O ZX81 resolverá também combinações das operações. Por exemplo.

PRINT 20—2*3**2+4/2*3

dá-nos a resposta 8. Percorre toda a instrução até chegar a esta, porque resolve primeiro todas as potências (**) da esquerda para a direita, e então todas as multiplicações e divisões (* e /), de novo da esquerda para a direita, e depois as adições e subtracções (+ e —), novamente da esquerda para a direita. O nosso exemplo é assim resolvido nos níveis seguintes:



Formalizamos isto dando a cada operação uma prioridade, um número entre 1 e 16. As operações com maior prioridade são avaliadas em primeiro lugar, e as operações com igual prioridade são avaliadas da esquerda para a direita.

** tem prioridade 10
* e / têm prioridade 8
+ e — têm prioridade 6

Quando – é utilizado para ter acesso aos números negativos, como quando escreve –1, então tem prioridade de 9. (É uma operação unitária, oposta da subtracção 3–1: uma operação unitária tem um operando, enquanto uma operação binária tem dois. É de notar que no ZX81 não pode utilizar o sinal + como operação unitária).

Esta ordem é absolutamente rígida, mas pode-se ultrapassar utilizando parêntesis: qualquer coisa entre parêntesis é avaliada em primeiro lugar e é depois tratada como um só número de modo que

PRINT 3*2+2

dá a resposta 6+2=8, mas

PRINT 3*(2+2)

dá a resposta 3*4=12.

Uma combinação como esta chama-se uma expressão — neste caso, uma expressão aritmética ou numérica, porque a resposta é um número. Regra geral, sempre que o computador espera que lhe dê um número, pode dar-lhe uma expressão e ele obterá a resposta.

Pode escrever números com pontos decimais (usando o ponto final), e pode também utilizar-se a notação cientifica — como é vulgar nas calculadoras de bolso. Neste caso, depois de um número comum (com ou sem ponto decimal), pode-se escrever uma parte exponencial que consiste na letra E, depois pode ser + ou — um número não decimal. Neste caso, o E significa *10** (dez elevado à potência de), de modo a

```
2.34E0    = 2.34*10**0    = 2.34
2.34E3    = 2.34*10**3    = 2340
2.34E—2   = 2.34*10**—2   = 0.0234        etc.
```

(Tente escrever isto no ZX81).

A melhor forma de pensar nisto é imaginar a parte exponencial deslocando o ponto decimal para a direita (para um expoente positivo) ou para a esquerda (para um expoente negativo).

Pode escrever mais que uma coisa ao mesmo tempo, podendo separá-las com vírgulas (,) ou ponto e vírgula (; prima shift simultaneamente que X). Se utilizar uma

vírgula, então o próximo número aparecerá no écran começando ou na margem esquerda, ou no meio da linha (16.ª coluna). Se utilizar um ponto e vírgula, então o número seguinte aparecerá no écran imediatamente a seguir ao último. Tente

PRINT 1;2;3;4;5;6;7;8;9;10

e

PRINT 1,2,3,4,5,6,7,8,9,10

para ver as diferenças. Pode misturar vírgulas com pontos e vírgula dentro de apenas uma instrução **PRINT** se assim o desejar.

Sumário

Instruções: **PRINT**, com vírgulas ou ponto e vírgula
Operações: +, —, *, /, **
Expressões, notação científica

Exercícios

1. Tente

PRINT 2.34E0
PRINT 2.34E1
PRINT 2.34E2

e assim consecutivamente até

PRINT 2.34E15

Verá que depois de algum tempo o ZX81 começa também a utilizar notação científica. É isto porque ele nunca escreve um número com mais de 14 digitos. Da mesma forma, experimente

PRINT 2.34E—1
PRINT 2.34E—2

etc.

2. Tente

PRINT 1,,2,,3,,,4,,,,5

Uma vírgula faz sempre avançar um pouco o número seguinte. Agora tente

PRINT 1;;2;;3;;;4;;;;5

Porque é que uma série de pontos e vírgulas não é diferente de apenas uma?

3. **PRINT** dá apenas 8 digitos significativos. Tente

PRINT 4294967295, 4294967295 —429E7

Isto prova que o computador pode trabalhar todos os digitos de 4294967295, embora não esteja preparado para os fazer aparecer no écran todos ao mesmo tempo.

4. Se tem tábua de logaritmos, faça o teste desta regra:

Elevar 10 à potência de um número é o mesmo que ter o antilogaritmo desse mesmo número. Escreva por exemplo

PRINT 10**0.3010

e procure o antilogaritmo de 0.3010. Porque será que as respostas não são exactamente iguais?

5. O ZX81 utiliza a aritmética em vírgula flutuante, o que significa que mantém separados os dígitos de um número em mantissa e em expoente. Isto nem sempre é exacto, mesmo para números inteiros, escreva

PRINT 1E10+1—1E10—1E10+1

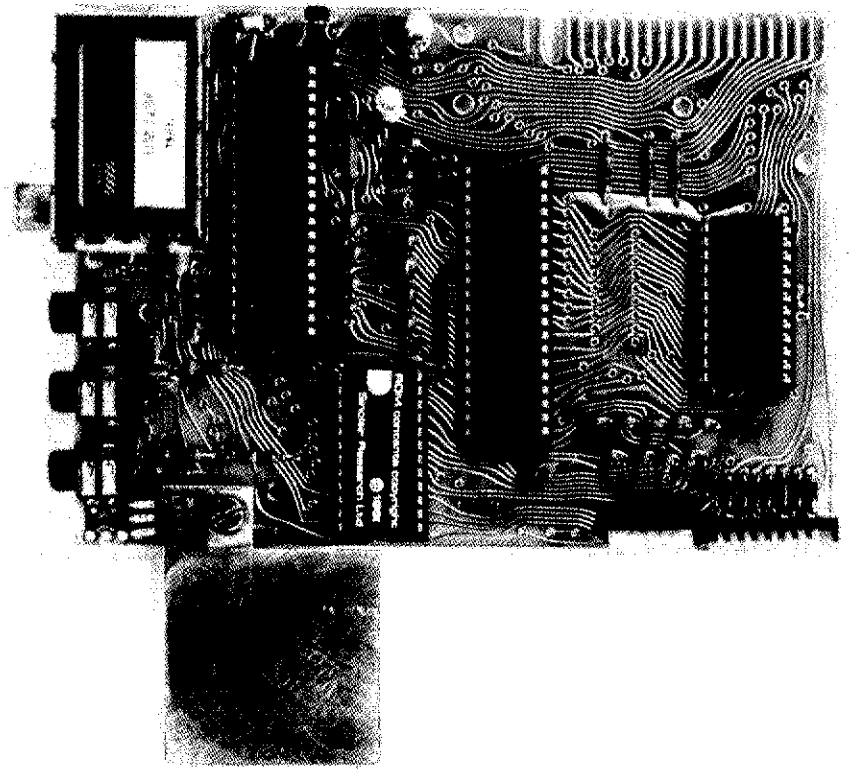
Os números são mantidos até à precisão de 9 1/2 dígitos, e assim 1E10 é muito grande para ser mantido exactamente certo. A não precisão é maior que 1 (na verdade cerca de 2) e assim os números 1E10 e 1E10+1 parecem iguais ao computador.

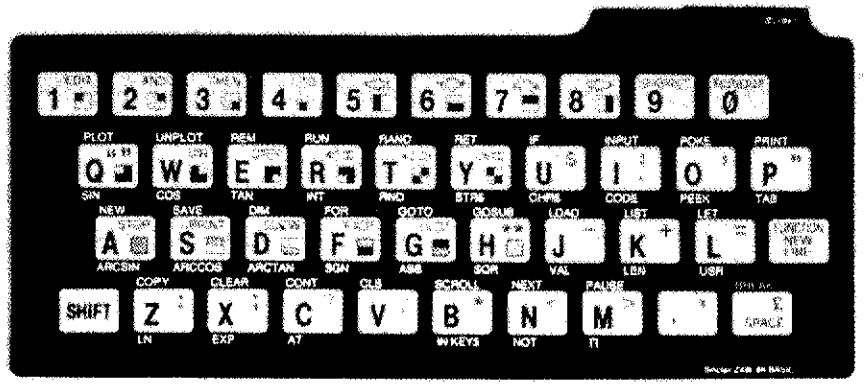
Para um exemplo ainda mais peculiar, escreva

PRINT 5E9+1—5E9

Aqui a inexactidão em 5E9 é apenas de 1 e o 1 ao ser adicionado é arredondado e dá 2. Aqui, os números 5E9+1 e 5E9+2 parecem iguais ao computador.

O maior número inteiro que pode ser mantido completamente exacto é $2^{32}-1$ (4,294,967,295).





Capítulo 5

Funções

Em termos matemáticos, uma função é uma regra para dar um número (o resultado) em troca de outro (o argumento ou operando) e é realmente uma operação unitária. O ZX81 tem alguns deles já programados e os seus nomes são as palavras escritas por baixo das teclas. **SQR**, por exemplo, é a função tão conhecida, raiz quadrada, e

PRINT SQR 9

dá 3, a raiz quadrada de 9. (Para se obter **SQR**, tem que se primir em primeiro lugar a tecla **FUNCTION** — carregando simultaneamente nas teclas **SHIFT** e **NEWLINE.** Isto irá mudar o cursor para **F** . Prima agora **SQR** tecla (H): **SQR** aparecerá no écran e o cursor retrocederá para **L**. Este mesmo método funciona para todas as palavras que estão escritas por baixo das teclas, que são nomes de funções.

Tente

PRINT SQR 2

Pode testar a exactidão da resosta por

PRINT SQR 2***SQR** 2

que deveria dar 2. Note-se que ambas as **SQR**s são resolvidas antes de *e, de facto, todas as funções (excepto uma — **NOT**) são resolvidas antes das cinco operações +, —, *, / e **.

Pode-se novamente lograr esta regra utilizando os parêntesis

PRINT SQR (2*2)

que dá 2.

Eis mais algumas funções (há uma lista completa no apêndice C). Se os seus conhecimentos de matemática não lhe permitem compreender algumas delas, não tem importância — poderá mesmo assim, continuar a utilizar o computador.

SGN A função sinal (por vezes denominado signum para evitar a confusão com **SIN**). O resultado é −1, 0 ou +1 segundo o argumento é negativo, zero ou positivo.

ABS O valor absoluto, ou módulo do valor. O resultado é a transformação do argumento em positivo, tal que

ABS—3.2 = 3.2

SIN
COS
TAN
ASN arco do seno
ACS arco de cosmo
ATN arco de tangente

As funções trigonométricas trabalham em radianos, não em graus.

LN logaritmo natural (base 2.718281828459045... alias e)
EXP função exponencial
SQR raiz quadrada
INT parte inteira do número. Esta é sempre arredondada "por defeito" assim **INT** 3.9 = 3 e **INT** −3.9 = −4. (Um número inteiro é um número natural, possivelmente negativo).
PI π = 3.14159265358979..., o perímetro em centímetros de um círculo com um centímetro de diâmetro. **PI** não tem argumento. (Apenas dez dígitos do mesmo são realmente acumulados no computador, e só oito aparecerão no écran).
RND Também não tem argumento. Dá-nos um número ao acaso entre Ø (cujo valor pode obter) e 1 (que não pode obter).

Utilizando o calão do último capítulo, todas estas operações, excepto **PI** e **RND** são operações unitárias com prioridade 11. (**PI** e **RND** são operações nulas, porque não têm operandos).

As funções trigonométricas, **EXP, LN** e **SQR,** são geralmente calculadas até 8 dígitos de exactidão. **RND** e **RAND:** estão ambas na mesma tecla, mas enquanto **RND** é uma função, **RAND** é uma palavra-chave, como **PRINT. RAND** utiliza-se para controlar o grau de casualidade de **RND.**

RND não é verdadeiramente aliatório, segue sim uma sequência fixa de 65536, números que estão misturados de modo a parecerem casuais. (**RND** é um pseudo-aliatório). Pode utilizar-se **RAND** para iniciar o **RND** num lugar definido nesta sequência escrevendo **RAND,** e depois um número entre **1** e 65535, e depois **NEWLINE.** Não é muito importante saber-se quando um número inicia o **RND,** pois o mesmo número depois de **RAND** iniciará sempre o **RND** no mesmo lugar. Escreve por exemplo

```
RAND 1          (e NEWLINE)
```

e depois

```
PRINT RND
```

e escreva-as ambas, uma de cada vez, diversas vezes. (Não esqueça de usar **FUNCTION** para obter **RND**). A resposta de RND será sempre **Ø.ØØ22735596,** uma sequência pouco casual

```
RAND Ø
```

(ou pode omitir o Ø) funciona de modo ligeiramente diferente: ele decide como deve

iniciar o **RND** em relação ao tempo em que a TV está ligada, e isto dá uma verdadeira casualidade.

Note: Em alguns dialectos do BASIC, deve-se sempre pôr o argumento de uma função entre parêntesis. Não é este o caso do ZX81 8K BASIC.

Sumário

Instrução: **RAND**
Funções: **SGN, ABS, SIN, COS, TAN, ASN, ACS, ATN, LN, EXP, SQR, INT, PI, RND.**
FUNCTION (FUNÇÃO)

Exercícios

1. Para se obterem os logaritmos normais (base **1Ø**), que são os que se procuram nas tábuas de logaritmos, divide-se o logaritmo natural por **LN 1Ø**. Por exemplo, para encontrar o logaritmo 2,

```
PRINT LN2/LN 1Ø
```

que dá a resposta Ø.3Ø1Ø3.

Tente multiplicar e dividir usando logaritmos, utilizando para isso o ZX81 como um conjunto de tábuas de logaritmos (para antilogaritmos, veja Capítulo 2, exercício 3). Confirme as respostas utilizando * e /− que são mais fáceis, mais rápidos e mais eficazes.

2. **EXP** e **LN** são funções mutuamente inversas pelo facto de que se aplicar uma e seguidamente a outra, volta a obter o primeiro número. Por exemplo,

```
LN EXP 2 = EXP LN2 = 2
```

O mesmo acontece com **SIN** e **ASN, COS** e **ACS,** e com **TAN** e **ATN.** pode-se utilizar isto para testar a precisão do computador a trabalhar com estas funções.

3. π radianos são 180° e assim, para converter os graus em radianos divide-se por 180 e multiplica-se por π: assim

```
PRINT TAN (45/180*PI)
```

dá um valor TAN 45° (1).

Para se passar de radianos a graus, divide-se por π e multiplica-se por 180.

4. Tente

```
PRINT RND
```

algumas vezes para ver como a resposta varia. Consegue detectar alguma sequência periódica? (Pouco provável).

Como se poderia utilizar **RND** e **INT** para se obter um número inteiro casual entre 1 e 6, para representar o lançamento de um dado? (Resposta: **INT** (**RND***6)+1.)

5. Teste esta regra:

Suponha que escolheu um número entre 1 e 872 e escreva

RAND e depois o seu número (e **NEWLINE**)

Então o valor seguinte de **RND** será:

(75*(o seu número+1) —1)/65536

6. (Só para matemáticos.)
Suponha que p é um (grande) número primo, e que a é uma raiz primitiva módulo p. Então se b; é o resto de a^i módulo P ($1 < b_i < p—1$), a sucessão

$$\frac{b_i—1}{p—1}$$

será uma sucessão cíclica de p — 1 números distintos na gama Ø até 1 (excluindo 1). Se escolher adequadamente os parâmetros, pode parecer bastante casual.

65537 é um número primo "Mersenne", $2^{16}—1$. Utilize este e a lei da reciprocidade do 2º grau de Gauss, para demonstrar que 75 é uma raiz primitiva módulo 65537.

O ZX81 utiliza p = 65537 e a = 75, e armazena alguns b_i-1 na memória. A função **RND** provoca a substituição de b_i-1 na memória por $b_{i+1}-1$ e dá o resultado $(b_{i+1}-1)/(p-1)$. **RAND** n (com $1 < n < 65535$) torna b_i-1 na memória por $b_{i+1}-1$ e dá o resultado.

7. **INT** arredonda sempre para o número inferior. Para se arredondar para o número inteiro superior mais próximo, soma-se primeiro Ø.5. Por exemplo

INT (2.9+Ø.5) = 3
INT (—2.9+Ø.5) = —3

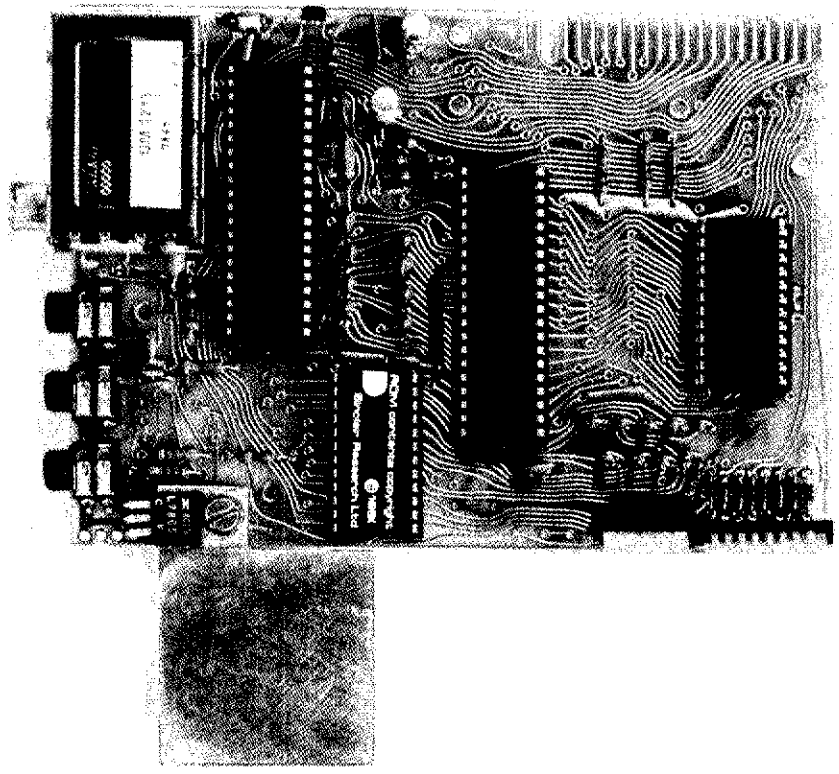
INT (2.4+Ø.5) = 2
INT (—2.4+Ø.5) = —2

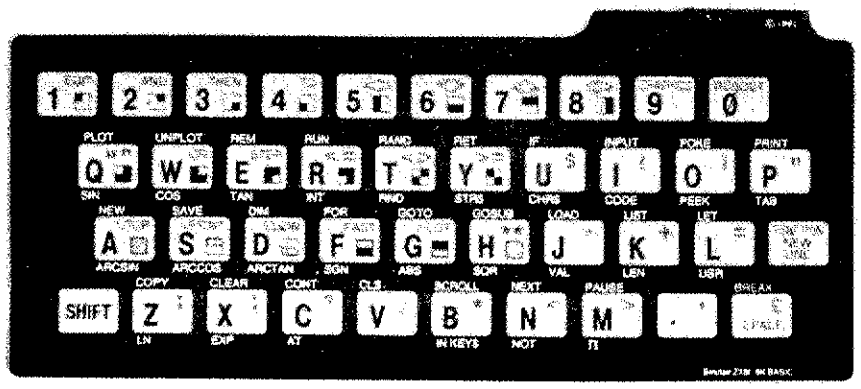
Compare estes resultados com os que obtém se não somar Ø.5.

8. Tente

PRINT PI, **PI** —3, **PI** —3.1, **PI** —3.14, **PI**—3.141

Isto mostra a precisão do computador no armazenamento de π.





Capítulo 6

Variáveis

"A minha calculadora de bolso", dirá, "pode acumular um número e voltar a lembrá-lo mais tarde. O ZX81 pode fazer isso?".

Sim. De facto, ele pode acumular centenas deles, utilizando a instrução **LET.** Suponha que os ovos custam 50$00 a dúzia, e V. quer lembrar-se disso. Escreva

LET OVOS=50 (e **NEWLINE,** claro)

O computador, em primeiro lugar, reservou nele um espaço em que pode acumular um número e, em segundo lugar, deu a esse espaço o nome de "OVOS", de modo a que nos possamos referir a ele mais tarde. Esta combinação de acumulação de espaço e nome chama-se variável. Em terceiro lugar, acumulou o número 50 no espaço: dizemos que ele atribuiu o valor 50 à variável (cujo nome é) OVOS. OVOS é uma variável numérica, porque o seu valor é um número.

Quer saber quanto custam os ovos? Escreva

PRINT OVOS

Se quiser saber o preço de meia dúzia de ovos, então escreva

PRINT OVOS/2

Na realidade, se quiser saber o quadrado do coseno do preço de um ovo, pode escrever

PRINT COS (OVOS/12)**2

"Como é fácil", deve pensar, e interrogar-se-á o que fazer em seguida, quando a pessoa que governa a casa disser, preocupada, "Meu Deus", os ovos acabam de subir para 60$00 a dúzia".

Bem, não há tempo a perder. Escreva

LET OVOS=60

Isto não reserva nenhum espaço de acumulação extra, mas substitui o valor anterior de 50$00 por 60$00.

Agora pode escrever

PRINT OVOS

com a certeza de receber o preço mais recente.

Agora escreva

PRINT LEITE

Obterá uma informação 2/Ø, e se verificar no apêndice B alínea 2 constatará que significa "não encontrou variável" — o computador não faz a menor ideia de quanto custa o leite porque nunca lho disse. Escreva

LET LEITE=15

e tudo se resolverá.

(Escreva

PRINT LEITE

novamente).

Não é necessário dar nomes de artigos de mercearia às variáveis — pode utilizar letras ou dígitos desde que a primeira seja uma letra. Podem-se dar espaços de modo a tornar mais fácil a leitura, mas estes não contarão como fazendo parte do nome.

Eis, por exemplo, alguns nomes que se podem dar às variáveis:

DOIS QUILOS DE MAÇÃS REINETAS
RÁDIO 3
RÁDIO 33
X
K9P

mas estes não:

3 URSOS (começa com um dígito)
AGORA? (? não é letra nem dígito)
BRANCO NO PRETO (os caracteres inverso vídeo não são permitidos)
DÁ-ME (- não é letra nem dígito)

Agora escreva

CLEAR

e

PRINT OVOS

Receberá novamente a informação 2 (variável não encontrada). O efeito de **CLEAR** é de desmemorizar todo o espaço acumulado que foi reservado para variáveis — é como se as variáveis nunca tivessem sido definidas. Ao desligar e ligar o computador isso também acontecerá — mas então ele não se lembrará de nada quando o voltar a ligar.

As expressões podem conter o nome de uma variável onde quer que elas possam ter um número.

Nota: Nalgumas versões da linguagem BASIC pode-se omitir **LET** e escrever, por exemplo

OVOS=50

Isto não se pode fazer no ZX81. De qualquer modo, ser-lhe-ia bastante difícil escrevê-lo.

Também nalgumas versões, só são verificados os dois primeiros caracteres de um nome, pelo que RADIO 3 e RADIO 33 são o mesmo nome; e noutros um nome de uma variável deve ser uma letra seguida de um dígito. Nenhuma destas restrições se aplica ao ZX81.

Contudo, nalgumas versões BASIC, se uma variável ainda não apareceu no lado esquerdo de uma instrução **LET**, presume-se que ela tenha o valor Ø. Como foi visto acima com **PRINT** LEITE, isto não acontece no ZX81.

Sumário

Variáveis
Instruções: **LET**, **CLEAR**

Exercícios

1. Porque é que os nomes das variáveis têm que começar com uma letra?

2. Se não está habituado a elevar números a potências (**, primindo SHIFT e H), então faça este exercício.

No seu grau mais elementar, 'A**B' significa 'A multiplicado por si próprio B vezes', mas é óbvio que isto só faz sentido se B for um número positivo inteiro. Para achar uma definição que trabalhe com outros valores de B, consideramos a regra

A ** (B+C) = A**B * A**C

Não é preciso muito para chegarmos à conclusão que isto resulta quando B e C são números inteiros positivos, mas se decidirmos que queremos que resultem mesmo quando o não são, então vemo-nos obrigados a aceitar que

A ** Ø = 1
A ** (−B) = 1 / A**B
A ** (1/B) = raiz B de A

e

A ** (B*C) = (A**B) ** C

Se não conhece ainda estas expressões, não tente lembrar-se logo delas; lembre-se só que

A ** −1 = 1/A

e

A ** (1/2) = raiz quadrada de A

e talvez quando se começar a familiarizar com isto e todo o resto comece a fazer mais sentido.

Experimente todas estas expressões dizendo ao computador para escrever várias expressões que contenham **, por exemplo:

```
PRINT 3**(2+0),3**2*3**0
PRINT 4**-1,1/4
```

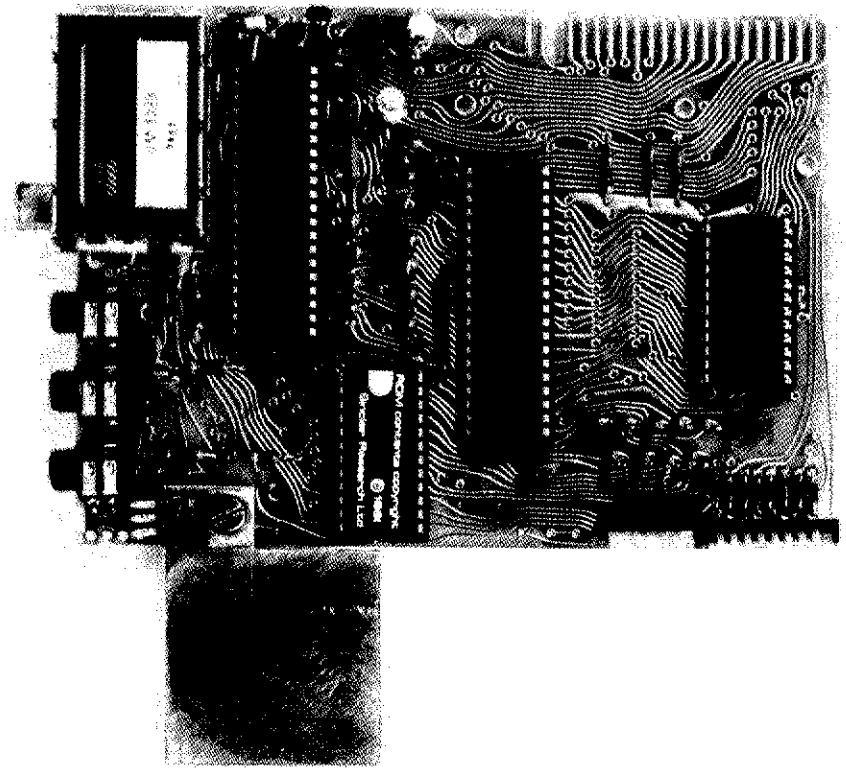
3. Escreva

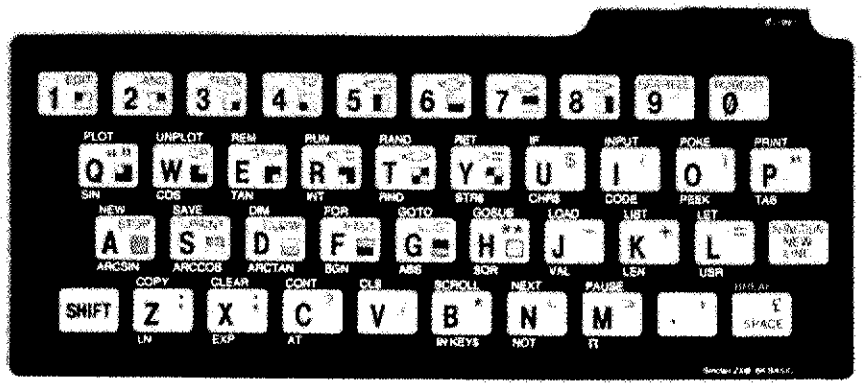
```
LET E=EXP 1
```

Agora E tem o valor 2.718281828. . ., a base dos logaritmos naturais. Teste a regra

```
EXP um número = E** o número
```

para vários números.





Capítulo 7

Strings (variáveis alfanuméricas)

Uma coisa que o ZX81 pode fazer e as calculadoras de bolso não podem é trabalhar com texto. Escreva

PRINT "OLÁ. SOU O SEU ZX81" (obtém-se ", primindo SHIFT e P)

A fria saudação entre aspas chama-se "string" (que significa uma série de caracteres), e pode conter qualquer caracter que se quiser excepto as aspas da "string", ". (Mas pode-se usar a chamada imagem entre aspas, "" (primindo SHIFT e Q), e aparecerá escrito como " por uma instrução **PRINT**.)

Um erro comum de escrita com as aspas é omitir uma delas — o que lhe dará o marcador **S**.

Se está a escrever números, pode utilizar as aspas para explicar o que significam os números. Por exemplo, escreva

LET OVOS=60

e depois

PRINT "O PREÇO DOS OVOS É";OVOS;"ESCUDOS A DÚZIA."

(Não se preocupa se isto ultrapassar o fim da linha.)

Esta instrução faz aparecer três coisas (elementos **PRINT**), nomeadamente a "string" "O PREÇO DOS OVOS É", o número 60 (o valor da variável OVOS), e depois a "string" "ESCUDOS A DÚZIA". De facto, pode imprimir qualquer número de elementos que quiser, e qualquer mistura de "strings" e números (ou expressões), note-se como os espaços numa "string" fazem parte dela como as letras. Eles não são ignorados nem no fim.

Há muitas coisas que se podem fazer com "strings".

1. Pode-se imputá-las a variáveis. Contudo, o nome da variável deve ser especial para mostrar que o seu valor é uma "string" e não um número: deve ser uma só letra seguida de $ (primindo SHIFT e U). Por exemplo, escreva

LET A$="COMPUTADOR PESSOAL"

e

PRINT A$

2. Pode juntá-las. É o que normalmente se chama de concatenação, que significa "encadeadas", e é exactamente o que ele faz. Tente

PRINT "COM"+"PUTADOR"

Não pode subtrair, multiplicar ou dividir "strings", ou elevá-las às potências.

3. Pode aplicar algumas funções a "strings" para obter números, e vice-versa.

LEN é aplicado a uma "string", e o resultado é o seu comprimento. Por exemplo **LEN** "QUEIJO"=6.

VAL é aplicado a uma "string", e o resultado é o que essa "string" dá quando calculada como expressão aritmética. Por exemplo (se A=9), **VAL** "1/2+**SQR**A"=3.5. Se a "string" à qual **VAL** é aplicada contém variáveis, deve-se obedecer a duas regras

a) Se a função **VAL** é parte de uma expressão maior, deve ser o primeiro elemento; por exemplo **10 LET** X = 7+**VAL** "Y" deve modificar-se para **10 LET** X =**VAL** "Y"+7.
b) **VAL** só pode aparecer na primeira coordenada de um **PRINT AT**, **PLOT** ou instrução **UNPLOT** (ver Capítulos 17 e 18), por exemplo

```
    10 PLOT 5, VAL "X" deve modificar-se para
10 LET Y=VAL "X"
15 PLOT 5, Y
```

STR$ Aplica-se a um número, e o resultado é o que aparece no écran se o número for mostrado por uma instrução **PRINT**. Por exemplo **STR$**3.5="3.5".

4. Tal como para os números, pode combiná-las de modo a fazer expressões "string", como

```
VAL (STR$ LEN "123456"+"-4")
```

que é avaliado como

```
VAL (STR$ LEN "123456"+"-4")
VAL (STR$ 6           +"-4")
VAL ("6"              +"-4")
VAL ("6-4"                  )
VAL "6-4"
6-4
2
```

Sumário

"Strings"
Operação: + (para "strings")
Funções: **LEN**, **VAL**, **STR$**.

Exercícios

1. Escreva

```
LET A$="2+2"
```

e depois

```
PRINT A$;"=", VAL A$
```

Tente mudar A para coisas mais complicadas e fazendo o mesmo, por exemplo:

```
LET A$="ATN1*4"
```

(Aqui, a resposta deveria ser π).

2. A "string" sem caracteres denomina-se "string" vazia ou nula. É só a "string" cuja dimensão seja 0. Lembre-se que os espaços são significativos e uma "string" vazia não equivale a uma que contenha espaços.

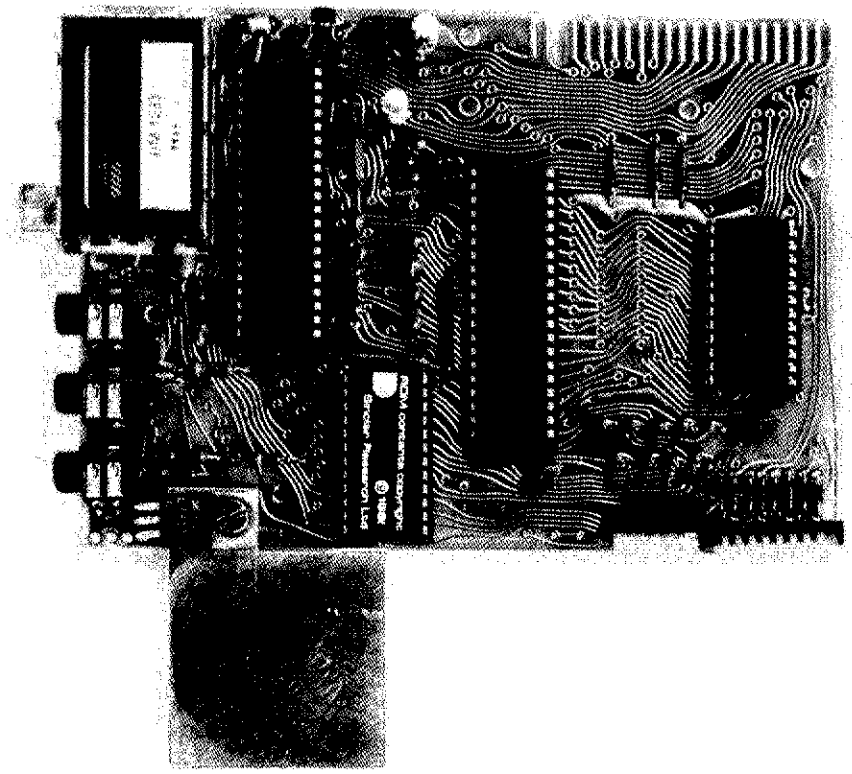
Não a confunda com a imagem entre aspas, "" (só um sinal, primindo SHIFT e Q). Este é um meio especial de se vencer o facto de que não se pode escrever uma aspa no meio de uma "string" (porque não?). Quando a imagem aspas aparece numa "string" que tem as suas aspas no fim (por exemplo, na listagem de um programa), aparecem dois símbolos de aspas, para o distinguir das aspas simples; mas quando é activada por uma instrução **PRINT**, aparece apenas um símbolo de aspas.

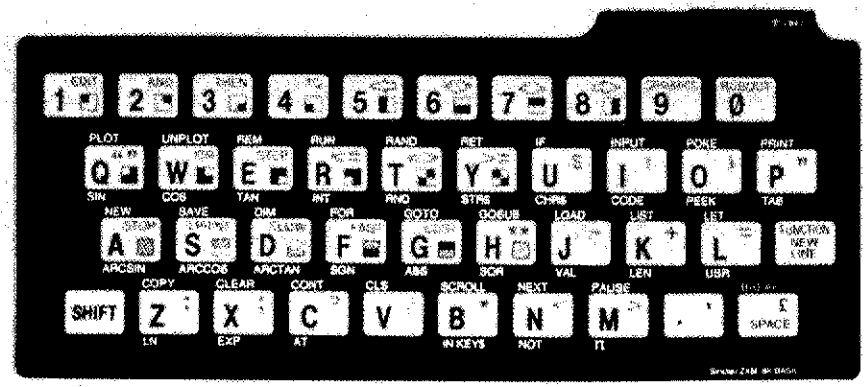
Tente

```
PRINT "X";"";"X","""";";"";""""
```

3. Se gostar de humilhar os computadores, escreva

```
PRINT "2+2=";2+1
```





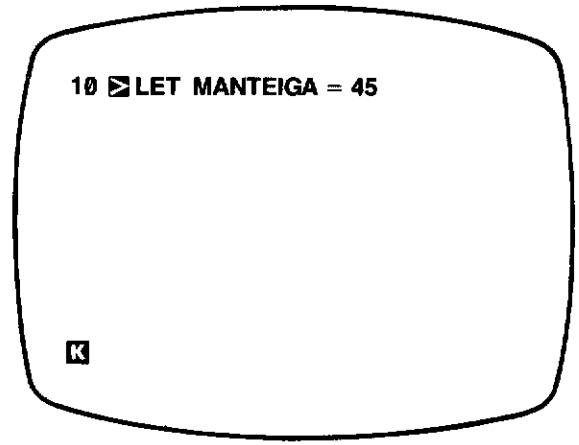
Capítulo 8

Programação do Computador

Agora, por fim, vai escrever um programa de computador. Desligue e ligue o computador, só para se certificar de que está tudo bem. Agora escreva

10 LET MANTEIGA = 45 (e **NEWLINE**)

e o écran aparecerá assim



É diferente do que aparecia com OVOS no Capítulo 6; se escrever

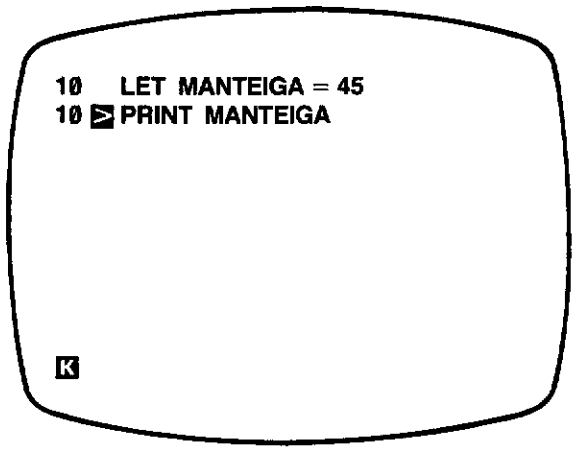
PRINT MANTEIGA

verá (da informação 2) que a variável MANTEIGA não foi estabelecida (volte a primir novamente **NEWLINE** e o écran voltará a aparecer como na figura).

Devido à instrução **LET** ter um número, **10**, atrás dela, o computador não a executou logo de seguida, mas acumulou-a para ser executada mais tarde. **10** é o seu número de linha, e utiliza-se para nos referirmos a ela da mesma forma como os nomes são utilizados para se referirem às variáveis. A um conjunto destas instruções acumuladas chama-se programa. Agora escreva

20 PRINT MANTEIGA

e o écran deverá aparecer-lhe assim



Isto é uma listagem do seu programa. Para realizar (executar ou correr) o programa, escreva

RUN (não esqueça **NEWLINE**)

e a resposta 45 aparecerá no canto superior esquerdo. Ao fundo do canto esquerdo aparecerá a informação Ø/2Ø. Ø, como se sabe, significa "O.K., não há problemas", e 2Ø é o número da linha onde acabou. Prima **NEWLINE**, e a listagem voltará.

Note que as instruções foram executadas pela ordem dos seus números de linha.

Suponha agora que se lembrou repentinamente que também necessitava guardar o preço do tabaco.

Escreva

15 **LET** TABACO = 50

e já está lá dentro. Tudo teria sido mais difícil se as primeiras duas linhas tivessem sido numeradas 1 e 2 em vez de 1Ø e 2Ø (os números de linha devem ser números inteiros entre 1 e 9999), razão pela qual, quando se começa a escrever um programa, é boa prática habituar-se a deixar lacunas nos números de linha.

Agora precisa modificar a linha 2Ø para

2Ø **PRINT** MANTEIGA,TABACO

Pode fazer a substituição escrevendo a linha por completo, mas há um modo de utilizar o que já lá está. Já reparou no pequeno na linha 15? É o cursor do

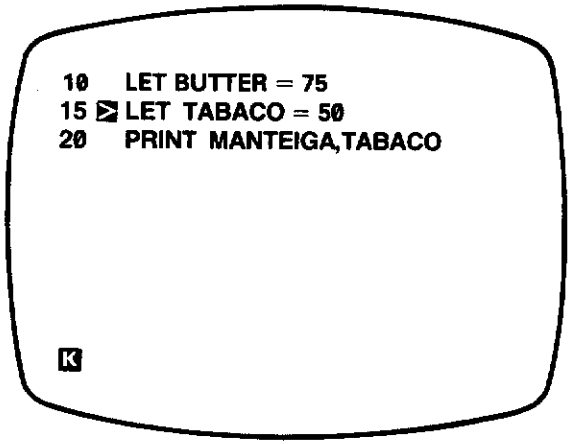
programa, e a linha que aponta é a linha corrente. Prima a tecla ↔ (primindo SHIFT e 6), e ela baixará para a linha 2Ø (↔ levantá-la-á novamente). Agora prima a tecla **EDIT** (primindo SHIFT e 1), e aparecer-lhe-á no écran uma cópia da linha 2Ø, na parte inferior do écran. Prima a tecla ◊ 7 vezes, de modo a que o cursor se mova para o fim da linha, e depois escreva

TABACO (sem **NEWLINE**)

Na linha do fundo deve ver

2Ø PRINT MANTEIGA,TABACO

Prima **NEWLINE** e ela substituirá a velha linha 2Ø. O écran agora apresentar-se-á assim:



Faça executar o programa por intermédio da instrução **RUN**, e aparecerão no écran ambos os preços.

(Um truque útil que se faz com **EDIT**, que é utilizado quando se quer apagar a linha de baixo do écran toda ao mesmo tempo. Prima **EDIT**, e a linha corrente virá do programa, subbstituindo o que V. queria apagar. Se primir **NEWLINE**, a linha voltará novamente ao programa, não alterando absolutamente nada, e a parte de baixo do écran ficará limpa, ficando apenas o cursor).

Escreva agora, distraidamente,

12 **LET** TABACO = 5Ø

Isto entrará em programa e você aperceber-se-á do seu erro. Para apagar esta linha, que já não é necessária, escreva

12 (com **NEWLINE**, claro)

Surpreender-se-á ao verificar que o cursor da linha corrente do programa desapareceu. Pode imaginar que ele está oculto entre as linhas 10 e 15 de modo que, se fizer ⌃ ele subirá para a linha 10, enquanto que se fizer ⌄ , descerá para a linha 15.

Por fim, escreva

LIST 15

No écran está

15 ▶ LET TABACO = 50
20 PRINT MANTEIGA,TABACO

A linha 10 desapareceu do écran, mas ainda está no programa — o que pode confirmar escrevendo novamente **NEWLINE**. A única coisa que acontece quando escreve **LIST** 15 é o aparecimento de uma listagem que começa na linha 15 e o facto de o cursor lhe aparecer na linha 15.

LIST

só por si, faz a listagem desde o início do programa.

Sumário

Programas
Edição de programas utilizando ⌃ , ⌄ e **EDIT**.
Instruções: **RUN**, **LIST**

Exercícios

1. Modifique o programa de modo a que lhe apareça no écran não só os dois preços, mas também as mensagens para a identificação deles.

2. Use a tecla **EDIT** para mudar o preço da manteiga.

3. Faça correr o programa e depois escreva

PRINT MANTEIGA,TABACO

As variáveis ainda lá estão, mesmo que o programa tenha acabado.

4. Escreva

12 (e **NEWLINE**)

O cursor está novamente oculto entre as linhas 10 e 15. Escreva **EDIT**, e a linha 15 passará para baixo: quando o cursor está oculto entre duas linhas, **EDIT** copiará a

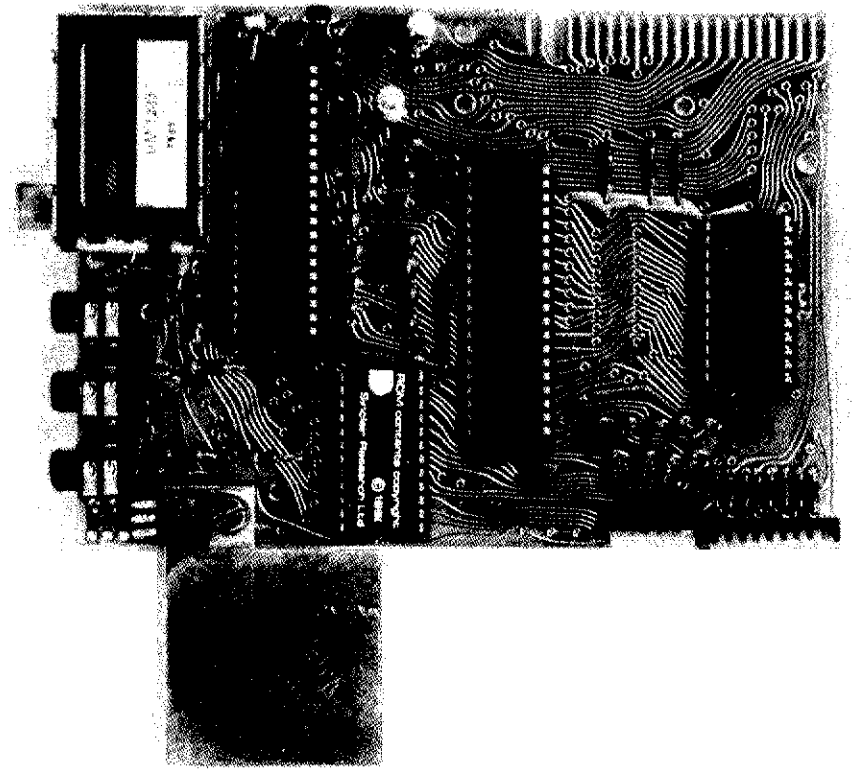
linha de número imediatamente superior. Escreva **NEWLINE**para apagar a parte de baixo do écran.

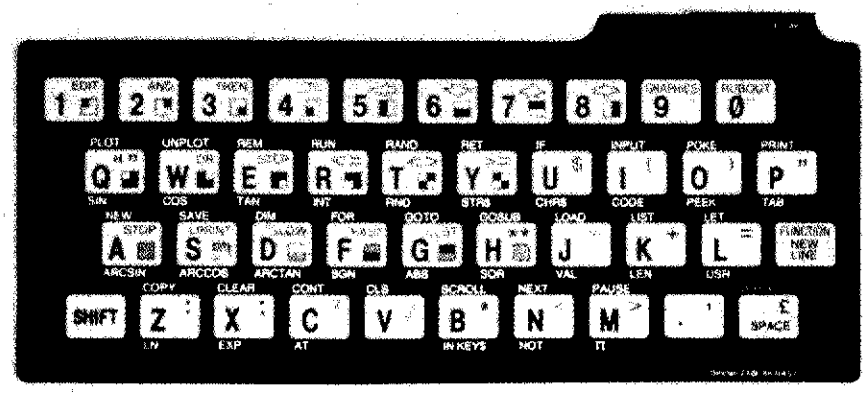
Agora escreva

30

Desta vez, o cursor está oculto depois do fim do programa e, se escrever **EDIT**, a linha 20 será copiada para o fundo do écran.

5. Ponha uma instrução **LIST** no programa de modo a que quando o correr, ele faça a listagem sozinho.





Capítulo 9

Mais programação

Escreva

NEW

Esta instrução limpa-lhe todos os programas e variáveis existentes no ZX81. (É parecida com a instrução **CLEAR**, embora **CLEAR** só limpe as variáveis).

Escreva com cuidado este programa:

```
10 REM ESTE PROGRAMA TIRA RAÍZES QUADRADAS
20 INPUT A
30 PRINT A,SQR A
40 GOTO 20
```

(Na linha 10, tem que ser você a dar quase todos os espaços).

Agora execute este programa (primindo RUN). Aparentemente, o écran fica vazio, e nada acontece, mas se reparar no cursor no canto inferior esquerdo, onde deveria aparecer um K , aparece um L — o aparelho começou a funcionar no modo de introdução de dados. Esta é a consequência da instrução **INPUT** na linha 20. O aparelho espera que você escreva um número (ou mesmo uma equação), e não continuará enquanto você não o fizer. Depois disso, é como se fosse

20 **LET** A = . . . o que você escreveu.

O que acontece então à linha 10? É como se o computador a tivesse ignorado completamente. E foi o que de facto aconteceu. **REM**, na linha 10, funciona como um comentário, ou aviso, e está lá apenas para lhe lembrar o que faz o programa. Uma instrução **REM** consiste em **REM** seguido daquilo que se quiser, e o computador ignora-a.

Bem, estávamos no modo de entrada de dados na linha 20. Escreva um número, por exemplo 4, e depois **NEWLINE**. Aparecer-lhe-á no écran o número 4 e a sua raiz quadrada, mas não acabou. Parece pedir outro número. Devido à linha 40 — **GOTO** 20 — que significa exactamente o que diz (vai para 20). Em vez de correr o programa e parar, o computador volta à linha 20 e começa novamente. Escreva agora outro número (por ex., 2: é melhor que seja um número positivo).

Depois de escrever alguns, ficará a pensar se o aparelho nunca se cansará deste jogo o que, de facto, não acontecerá. Quando ele lhe pedir novo número, escreva **STOP** (SHIFT+A) em vez de um número. Ele aceitará a sugestão. O computador responderá com o relatório D/20 — procure D na lista de relatórios (apêndice B). A linha 20 é onde ele estava à espera quando lhe ordenou que parasse.

Se entretanto, se lembrou de mais números dos quais queria a raiz quadrada, então escreva

CONT

(abreviatura de continuar) e o computador limpará o écran e pedir-lhe-á outro número.

Com a instrução **CONT**, o computador lembra-se do número da linha do último relatório que deu e que tinha um código sem ser Ø, e passa para essa linha. Como o último relatório era D/2Ø (e D não é Ø), neste caso **CONT** é igual a **GOTO** 2Ø.

Seguidamente, escreva números até o écran estar cheio. Quando estiver cheio, o computador pára no relatório 5/3Ø. O número 5 significa écran cheio, e o número 3Ø é o número da declaração **PRINT** que ele estava a tentar executar quando descobriu que não tinha espaço no écran. Escreva novamente

CONT

que limpará o écran e continue — desta vez, **CONT** significa **GOTO** 3Ø.

Note-se que o écran foi limpo, não devido à instrução **CONT**, mas porque é uma ordem, e todas as ordens, (excepto **COPY**, que aparece no Capítulo 2Ø), limpam primeiro o écran.

Quando não quiser mais, pare o programa escrevendo **STOP** e obtenha a listagem com **NEWLINE**.

Veja a instrução **PRINT** na linha 3Ø. De cada vez que se escreve o par de números A e **SQR** A, faz-se numa nova linha, e isto porque a instrução **PRINT** não acaba numa vírgula ou num ponto e vírgula. Neste caso todas as instruções **PRINT** começam a escrever numa nova linha. (Assim, para obter uma linha em branco, utilize uma instrução **PRINT** onde não haja nada para escrever — escreva só **PRINT**).

Contudo, uma instrução **PRINT** pode acabar em vírgula ou ponto e vírgula, pelo que a próxima instrução **PRINT** continua a escrever como se as duas estivessem sido uma longa instrução.

Por exemplo, utilizando as vírgulas, substitua 3Ø por

3Ø **PRINT** A

e corra o programa para ver como as sucessivas instruções **PRINT** podem escrever na mesma linha mas espalhadas em duas colunas.

Por outro lado, com pontos e vírgulas, com

3Ø **PRINT** A;

fica tudo misturado.

Tente também

3Ø **PRINT** A

Escreva agora estas linhas extras.

```
1ØØ REM ESTE PROGRAMA MEDE STRINGS
11Ø INPUT A$
12Ø PRINT A$,LEN A$
13Ø GOTO 11Ø
```


Este programa é independente do outro, mas pode mantê-los ao mesmo tempo. Para fazer correr o novo programa escreva

RUN 1ØØ

Este programa pede a entrada de uma "strings" em vez de um número, e escreve a própria "strings" e o seu comprimento. Escreva

CAT (e **NEWLINE**; como sempre)

Visto que o computador está à espera de uma "string", escreve duas aspas de seguida — isto é um aviso, que lhe poupa também trabalho. Mas não tem que limitar às "strings" constantes (strings explícitas com aspas a abrir e a fechar com todos os caracteres entre elas); o computador avaliará qualquer expressão string, como por exemplo uma com variáveis string. Neste caso, teria que apagar as aspas que o computador escreveu. Tente fazer isso. Apague-as (com ◊ e **RUBOUT** duas vezes) e escreva

A$

Uma vez que A$ ainda tem o valor "CAT", a resposta é novamente CAT 3.

Agora dê novamente a instrução

A$

mas desta vez sem apagar as aspas. A$ tem agora o valor "A$", e a resposta é 2.

Se quiser utilizar **STOP** para uma instrução "string", deve primeiramente retroceder o cursor para o princípio da linha primindo ◊ .

Voltemos agora ao **RUN** 1ØØ que tínhamos visto. Salta para a linha 1ØØ. Será que deveríamos ter escrito **GOTO** 1ØØ? Neste caso a resposta é sim, mas há uma diferença. Antes de mais, **RUN** 1ØØ limpa as variáveis (como **CLEAR** no Capítulo 6), e depois funciona só como **GOTO** 1ØØ. **GOTO** 1ØØ não limpa nada. Pode acontecer-lhe querer um programa sem querer limpar as variáveis. Nesse caso, é necessário **GOTO**, mas **RUN** seria desastroso, pelo que é melhor não se habituar a escrever automaticamente **RUN** para executar um programa.

Outra diferença é que se pode escrever **RUN** sem um número de linha, pois ele começa na primeira linha do programa. **GOTO** tem que ter sempre um número de linha.

Os dois programas pararam porque escreveu **STOP** na linha **INPUT**, mas por vezes por engano escreve-se um programa que não se pode parar, e ele também não para só por si. Escreva

```
2ØØ GOTO 2ØØ
RUN 2ØØ
```

Isto parece nunca mais acabar, a menos que desligue o aparelho; mas há uma solução menos drástica. Prima a tecla **SPACE**, que tem a palavra "BREAK" por cima. O programa parará, aparecendo D/200.

No fim de cada linha de programa, o computador verifica se esta tecla foi primida, se foi, ele pára. A tecla **BREAK** pode também ser utilizada enquanto se está a utilizar o gravador ou a Printer.

Vimos agora as instruções **PRINT**, **LET**, **INPUT**, **RUN**, **LIST**, **GOTO**, **CONT**, **CLEAR**, **NEW** e **REM**, e a maior parte delas podem ser utilizadas como ordens ou como linhas de programa — o que acontece com quase todas as instruções — em BASIC. A única excepção é **INPUT**, que não pode ser utilizada como ordem (se tentar, obtém o relatório 8; a razão é que a mesma área dentro do computador é utilizada para ambos: as ordens e para a introdução de dados; e uma ordem **INPUT** seria uma confusão). **RUN**, **LIST**, **CONT**, **CLEAR** e **NEW** não se utilizam muito num programa, mas podem-se utilizar.

Sumário

Instruções: **GOTO**, **CONT**, **INPUT**, **NEW**, **REM**, **PRINT**
STOP em introdução de dados
BREAK

Exercícios

1. No programa da raiz quadrada, tente substituir a linha 40 por **GOTO** 5, **GOTO** 10 ou **GOTO** 15 — não deverá haver nenhuma diferença perceptível na passagem do programa. Se número de linha numa instrução **GOTO** se referir a uma linha que não exista, então passa à linha seguinte. Faz-se o mesmo para **RUN**; de facto **RUN** só por si significa **RUN** 0.

2. Execute o programa do comprimento das string, e quando aparecer o pedido de entrada de uma string, escreva

```
X$        (depois de apagar as aspas)
```

Claro que X$ é uma variável indefinida e obtém-se o relatório 2/110.
Se escrever agora

```
LET X$ = "ALGO DEFINIDO"
```

(que tem o seu próprio relatório de 0/0) e

```
CONT
```

descobrirá que pode utilizar X$ como um dado que pode ser introduzido sem qualquer problema.

A questão, neste exercício, é que **CONT** tem o efeito de **GOTO** 110. Ele não liga

ao relatório 0/0 porque este tinha o código 0, e utiliza a linha do relatório anterior, 2/110. Isto tem a sua utilidade. Se um programa para num erro, pode fazer-se tudo o que se quiser para o resolver, e **CONT** continuará depois disso.

3. Tente este programa:

```
10 INPUT A$
20 PRINT A$;" = ";VAL A$
30 GOTO 10
```

(confira no Capítulo 7, exercício 1).

Ponha elementos print extra de modo que o computador lhe diga o que vai fazer, e lhe peça que introduza dados "string" com delicadeza.

4. Escreva um programa que para preços introduzidos dê o respectivo I.T. (a 15%). Escreva novamente a instrução **PRINT** para que ele mostre o que faz. Modifique o programa de modo a poder introduzir também a percentagem I.T. (para permitir futuros orçamentos).

5. Escreva um programa para imprimir o total de números que introduziu. (Sugestão: arranje duas variáveis TOTAL — estabelecida como 0 no início — e ITEM. Faça entrar ITEM, adicione-o ao TOTAL, escreva-os a ambos, e volte novamente ao princípio).

6. As listagens automáticas (as que não resultam de uma instrução **LIST** podem tê-lo confundido. Se fizer um programa com 50 linhas, todas com declarações **REM**,

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

então poderá tentar compreender o que se passa.

A primeira coisa a tomar em conta é que a linha corrente (com ▶) aparecerá sempre no écran, e de preferência, próxima do centro.

Escreva

```
LIST      (e NEWLINE, claro)
```

e prima novamente **NEWLINE**. Deverá obter linhas de 1 a 22 no écran. Agora escreva

```
23 REM
```

e deverá obter linhas de 2 a 23 no écran; prima

28 **REM**

e obterá linhas de 27 a 48. (Em ambos os casos, ao escrever nova linha o cursor moveu-se de modo que se teve que fazer nova listagem).

Isto poderá parecer-lhe um pouco arbitrário, mas ele está apenas a tentar dar-lhe o que V. quer. Contudo, devido à não predictibilidade do pensamento humano, nem sempre adivinha.

O computador pode manter a gravação não só da linha corrente, a que tem de aparecer no écran, mas também a primeira linha do écran. Quando tenta fazer uma listagem, a primeira coisa que o computador faz é comparar a primeira linha com a linha corrente.

Se a primeira linha vem depois, não há maneira de poder começar por aí, pelo que ele utiliza a linha corrente para nova primeira linha e faz a sua listagem.

Por outro lado, ele começa por tentar fazer a listagem aparecer a partir da primeira linha. Se a linha corrente aparecer no écran, tudo corre bem; se a linha corrente estiver fora do fundo do écran, ele modifica a primeira linha para que a linha corrente apareça.

Tente, movendo a linha corrente, escrevendo

número de linha **REM**

LIST faz andar a linha corrente mas não faz andar a primeira linha, pelo que as listagens subsequentes serão diferentes. Por exemplo escreva

LIST

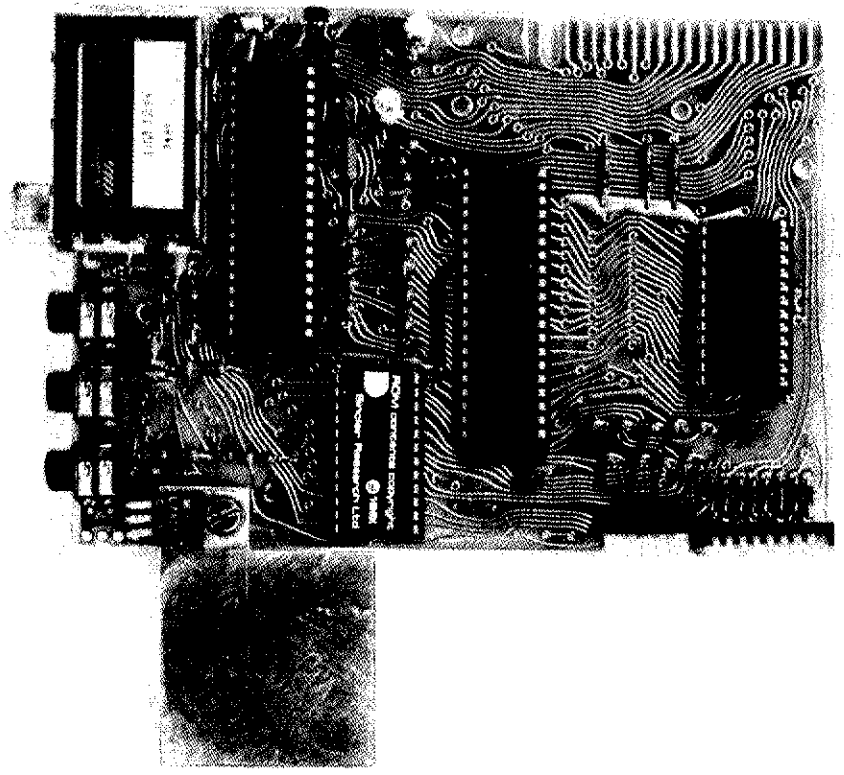
para obter a listagem de **LIST**, e depois prima novamente **NEWLINE** de modo a que a primeira linha seja 0. Aparecerão no écran linhas 1 a 22. Escreva

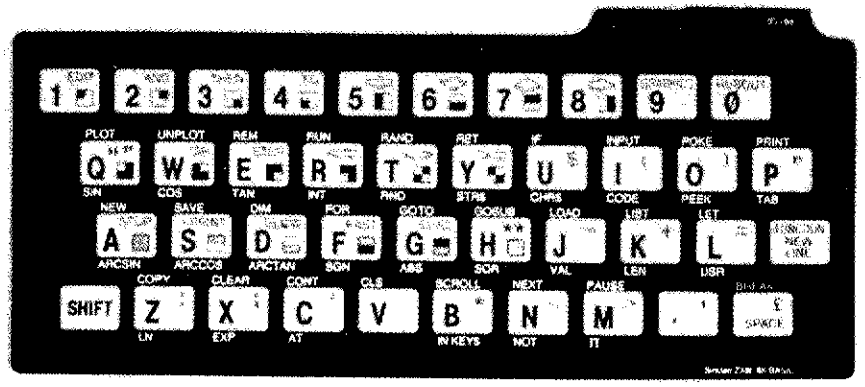
LIST 22

que lhe dará as linhas 22 a 43; quando prime novamente **NEWLINE**, volta às linhas 1 a 22.

Isto é mais útil para pequenos programas do que para programas longos.

7. Que fariam **CONT**, **CLEAR** e **NEW** num programa? Quais serão os casos em que estas instruções se poderão aplicar?





Capítulo 10

Se . . .

Todos os programas que vimos até agora têm sido bastante prognosticáveis — limitaram-se a cumprir todas as instruções, e depois voltaram talvez novamente ao princípio. Isto não é, de facto, muito útil. Na prática, esperava-se que o computador tome decisões e aja de acordo com elas; ele fará isto se utilizarmos a instrução **IF.**

Limpe o computador (usando **NEW**) e escreva, fazendo executar este divertido pequeno programa:

```
10 PRINT "POSSO CONTAR-LHE UMA PIADA?"
20 INPUT A$
25 CLS
30 A$="NÃO" THEN GOTO 100
40 PRINT,,"QUAL É A PERNA DIREITA DO CAVALO DE D. JOSÉ?"
       (DIREITA/ESQUERDA)
50 INPUT A$
60 IF A$="ESQUERDA" THEN GOTO 120
70 PRINT,,"NÃO. ESSA É A TORTA. A ESQUERDA É QUE ESTÁ
     DIREITA."
80 PRINT,,"POSSO CONTAR OUTRA VEZ?"
90 GOTO 20
100 PRINT "ENTÃO NÃO CONTO."
110 STOP
120 PRINT,,"NÃO. JÁ NÃO SABE QUAL É O LADO DIREITO?"
130 GOTO 80
```

Antes de falarmos na instrução **IF**, vamos primeiro olhar para a instrução **STOP** na linha 110, uma instrução **STOP** para a execução do programa, dando o relatório 9.

Agora, como se pode verificar, uma instrução **IF** toma a forma de

IF condição **THEN** instrução

(se) (então)

As instruções são, neste caso, instruções **GOTO**, mas poderiam ser qualquer outra coisa, mesmo até mais instruções **IF**. A condição é algo que vai ser trabalhado para se saber se é verdadeira ou falsa; se aparecer como verdadeira, a instrução depois de **THEN** é executada, de outro modo, passar-se-á por cima dela.

As condições mais úteis comparam dois números ou duas "strings" (séries de caracteres alfanuméricos): pode verificar se dois números são iguais ou se um é maior que o outro; e podem também verificar se duas "strings" são iguais ou se uma segue a outra por ordem alfanumérica. Elas usam as relações =, <, >, <=, >=, e <>.

Este sinal de = que utilizámos por duas vezes no programa uma vez para números e outra para "strings", não é o mesmo que = numa instrução **LET**.

< significa "menor que" e assim

1 < 2
—2 < —1
e —3 < 1

são todas verdadeiras, mas

1 < 0
e 0 < —2

são falsas.

Para verificarmos como isto funciona, vamos escrever um programa para introduzir números e ele escreve o maior deles.

```
10 PRINT "NÚMERO", "MAIOR ATÉ AGORA"
20 INPUT A
30 LET MAIOR = A
40 PRINT A, MAIOR
50 INPUT A
60 IF MAIOR < A THEN LET MAIOR = A
70 GOTO 40
```

A parte mais importante é a linha 60, que actualiza MAIOR, se o seu valor anterior for menor que o novo número introduzido A.

> (SHIFT M) significa "maior que", e é como <, mas ao contrário. Pode lembrar qual é que é, porque a ponta da seta está apontada para o número que se supõe ser menor.

< = (SHIFT R — não escreva < seguido de =) significa "menor ou igual a", pelo que é igual a < à excepção de que é também verdadeira mesmo que os dois números sejam iguais: assim 2 < = 2 é verdadeiro, mas 2 < 2 é falsa.

> = (SHIFT Y) significa "maior ou igual a" e é semelhante a >.

< > (SHIFT T) significa "diferente de" é o oposto de =.

Todas estas operações de relação tem prioridade de 5.

Os matemáticos costumam escrever < =, > =, < > como ≤, ≥ e ≠. Escrevem também coisas como '2 < 3< 4' que significa '2< 3 e 3< 4', mas isto não se pode fazer na linguagem BASIC.

Estas relações podem ser combinadas utilizando as operações lógicas **AND**, **OR** e **NOT**.

uma relação **AND** outra relação

(e)

é verdadeira quando as ambas relações são verdadeiras.

uma relação **OR** outra

(ou)

é verdade sempre que uma ou ambas das duas relações forem verdadeiras.

NOT relação

é verdade sempre que a relação seja falsa e é falsa sempre que a relação seja verdadeira.

Podem-se fazer expressões lógicas com relações e **AND**, **OR** e **NOT** tal como se podem fazer expressões numéricas com números e +, − etc.; pode mesmo pô-las entre parêntesis se for necessário. **NOT** tem prioridade 4, **AND** 3 e **OR** 2.

Uma vez que (ao contrário das outras funções) **NOT** tem uma prioridade relativamente baixa, o seu argumento não necessita de parêntesis, a menos que contenha **AND** ou **OR**; assim, **NOT** A = B significa **NOT** (A = B) (que é o mesmo que A < > B).

Para ilustrar isto, limpe o computador e tente fazer este programa.

```
10 INPUT F$
20 INPUT IDADE
30 IF F$ = "A" AND IDADE < 18 OR F$ = "B" AND IDADE < 13
THEN PRINT
"NÃO VÁ PORQUE AINDA NÃO TEM IDADE."
40 PRINT "PODE IR"
50 GOTO 10
```

Suponhamos que, neste caso F$ é a categoria de um filme — A para maiores de 18 anos, B para maiores de 13 e C para todos, e o programa resolve se uma pessoa de determinada idade pode ver o filme.

Por fim, podemos não só comparar números, mas também valores alfanuméricos "string". Já vimos '=' utilizado em 'F$ = "A"', e pode mesmo utilizar os outros cinco, < etc.

Assim, que significa "menor que" para as strigs? Há uma coisa que não pode significar, que é "mais pequeno que", por isso não cometa esse erro. Fica definido que uma string tem valor menor que outra se vier primeiro na ordem alfabética: então

"ALBERTA"	< "ALBERTO"
"CARLOS"	> "CARLA"
"CURTO"	< "CURTO-CIRCUITO"
"BILIÃO"	< "MILHÃO"
"TCHAIKOVSKY"	< "WAGNER"
"ESCUDO"	< "LIBRA"

são todas verdadeiras. < = significa "menor que ou igual a", etc., tal como para os números.

Nota: Nalgumas versões do BASIC — não no ZX81 —, a instrução **IF** pode tomar a forma **IF** condição **THEN** número da linha, mas significa o mesmo que

IF condição THEN GOTO número da linha

E agora o ZX81 já compreende.

Sumário

Instruções: **IF**, **STOP**

Operações: =, < , > , < =, > =, < > , **AND**, **OR**

Função: **NOT**

Exercícios

1. < > e = são opostos no sentido de que **NOT** A = B é o mesmo que A< > B

e

NOT A< > B é o mesmo que A = B

Convença-se que > = é oposto de < , e < = é oposto de > de modo que pode livrar-se sempre de **NOT** à frente de uma relação modificando a relação para o seu oposto.

Também

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

é o mesmo que

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

e

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

é o mesmo que

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

Utilizando isto, pode trabalhar vários **NOT** através de parêntesis até de facto estarem todos aplicados a relações, e depois pode desembaraçar-se deles. Assim logicamente, **NOT** é desnecessário. Poderá, contudo continuar a achar que, se o utilizar, o programa fica mais claro.

2. A linguagem BASIC pode por vezes trabalhar ao longo de linhas diferentes de Português. Considere, por exemplo, a frase "Se A diferente de B ou C". Como escreveria isto em BASIC? (A resposta não é

IF A< > B **OR** C' nem 'IF A< > B **OR** A< > C')

Não se preocupe se não perceber os exercícios 3, 4 e 5, pois os pontos que ele refere são bastante "refinados".

3. (Para peritos).
Tente

PRINT 1 = 2, 1 < > 2

que espera que dê um erro de sintaxe. De facto, no que diz respeito ao computador, não há tal coisa como valor lógico.

(i) =, < , > , < =, > = e < > são operações binárias com prioridade 5. O resultado é 1 se a relação é verdadeira, e **Ø** se for falsa.

(ii) Em

IF condição **THEN** instrução

a condição pode, de facto, ser uma expressão numérica. Se o seu valor for **Ø**, conta como falsa, e qualquer outro valor conta como verdadeiro. Assim, a declaração **IF** significa exactamente o mesmo que

IF condição < > **Ø** declaração **THEN**

(iii) **AND, OR** e **NOT** são operações cujos resultados são números.

X **AND** Y tem valor
- X se Y não for zero (conta como verdadeiro)
- **Ø** se Y é zero (conta como falsa)

X **OR** Y tem valor
- 1 se Y não for zero
- X se Y é zero

e

NOT X tem valor
- **Ø** se X não for zero
- 1 se X é zero

Leia novamente o capítulo, à luz desta descoberta, certificando-se de que tudo funciona.

Nas expressões X **AND** Y, X **OR** Y e **NOT** X, X e Y tomarão cada uma o valor **Ø** ou 1, para verdadeira ou falsa. Resolva as dez diferentes combinações e certifique-se se elas fazem o que se espera que **AND**, **OR** e **NOT** façam.

4. Escreva este programa:

```
10 INPUT A
20 INPUT B
30 PRINT (A AND A>=B)+(B AND A<B)
40 GOTO 10
```

Ele escreve de cada vez o maior dos dois números A e B — porquê? Convença-se de que pode pensar em

X **AND** Y

significando

(x se y < > **0** ; senão o resultado é **0**)

Uma expressão que use **AND** e **OR** como esta faz, denomina-se expressão condicional. Vamos dar um exemplo utilizando **OR**:

```
LET PREÇO REVENDA = PREÇO MENOS I.T.* (1.15 OR V$ = "TAXA ZERO")
```

Repare como **AND** tem tendência a adicionar (porque a sua tendência é ser **0**), e **OR** tende a multiplicar (porque a sua tendência é ser 1).

5. Pode-se também fazer expressões condicionadas de string, utilizando apenas **AND**.

X$ **AND** Y$ tem valor { X$ se Y não for zero
                        se Y é zero

o que significa que 'X$ se Y (senão a string vazia)'.

Tente este programa, que instrui duas strings e põe-nas por ordem alfabética.

```
10 INPUT A$
20 INPUT B$
30 IF A$<=B$ THEN GOTO 70
40 LET C$=A$
50 LET A$=B$
60 LET B$=C$
70 PRINT A$;" ";("<" AND A$<B$)+("=" AND A$=B$);" ";B$
80 GOTO 10
```

6. Tente este programa:

```
10 PRINT "X"
20 STOP
30 PRINT "Y"
```

Quando o passar ele escreverá "X" e pára como relatório 9/20. Agora escreva

```
CONT
```


Espera que se comporte como '**GOTO** 20', de modo que o computador pararia novamente sem escrever "Y"; mas isto não seria muito útil, e assim tudo está feito de modo a que os relatórios com código 9 (declaração **STOP**), o número da linha aumenta para 1 para uma instrução **CONT**. Assim, no nosso exemplo, '**CONT**' comporta-se como '**GOTO** 21' (o qual, visto que não há linhas entre **20** e **30** comporta-se como '**GOTO** 30').

7. Muitas versões do BASIC (mas não no caso do BASIC ZX81), têm uma instrução ON, que toma a forma de

ON expressão numérica GOTO número de linha, número de linha, número de linha

Aqui, a expressão numérica é avaliada; supondo que o seu valor é n então dá-nos que

GOTO o número de linha n

Por exemplo

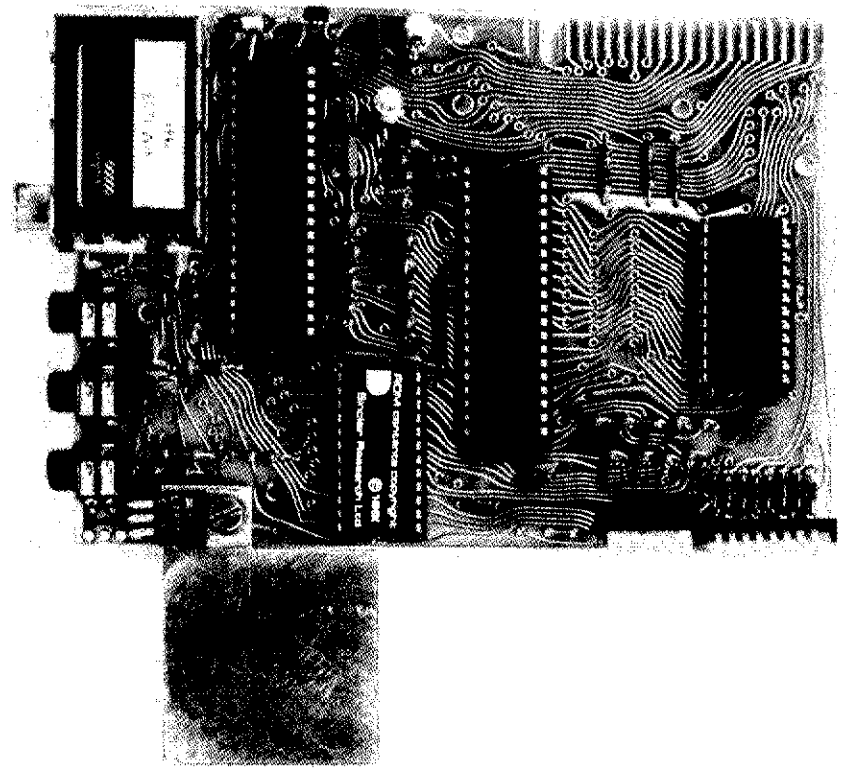
```
ON A GOTO 100, 200, 300, 400, 500
```

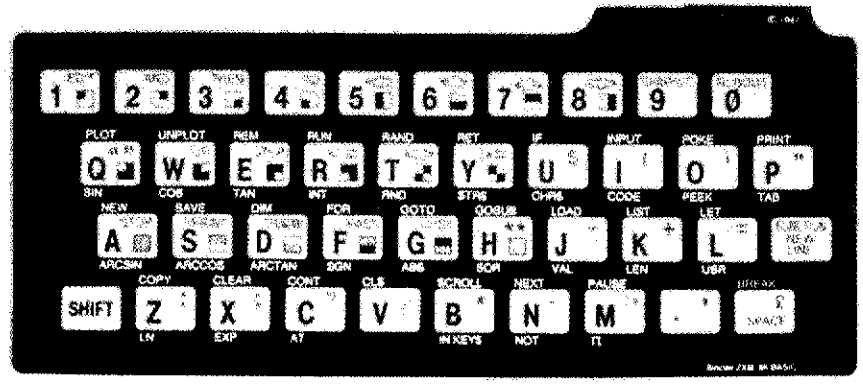
Neste caso, se A tiver valor 2, então 'GOTO 200' é executado. No BASIC ZX81, isto pode ser substituído por

```
GOTO 100*A
```

No caso dos números da linha não aumentarem às centenas como neste caso, veja a forma de utilizar, em vez disto,

GOTO uma expressão condicional




1 2 3 4 5 6 7 8 9 0
PLOT UNPLOT REM RUN RAND RET IF INPUT POKE PRINT
Q W E R T Y U I O P
SIN COS TAN INT PIND STR$ CHR$ CODE PEEK TAB
NEW SAVE DIM FOR GOTO GOSUB LOAD LIST LET
A S D F G H J K L
ARCSIN ARCCOS ARCTAN SGN ABS SQR VAL LEN USR
SHIFT COPY CLEAR CONT CLS SCROLL NEXT PAUSE
Z X C V B N M
LN EXP AT IN KEYS NOT TI
SPACE

Capítulo 11

Os caracteres

As letras, dígitos, pontuação, etc., que aparecem em strings são chamados caracteres e formam o alfabeto, ou conjunto de caracteres, que o ZX81 utiliza. A maior parte destes caracteres são símbolos simples, mas há outros, denominados sinais, que representam palavras completas, tais como **PRINT, STOP,** **, etc.

Ao todo existem 256 caracteres, e cada um tem um código entre 0 e 255. Há uma lista completa destes caracteres no Apêndice A. Para fazer a conversação entre códigos e caracteres, existem duas funções. **CODE** e **CHR$.**

CODE aplica-se a uma string, e dá o código do primeiro caracter da "string" (ou 0 se a "string" for vazia).

CHR$ aplica-se a um número, e dá o caracter simples, cujo código é esse número.

Este programa escreve todos os 256 caracteres.

```
10 LET A=0
20 PRINT CHR$ A;
30 LET A=A+1
40 IF A<256 THEN GOTO 20
```

Na primeira linha, podem-se ver os símbolos ", ▛, $, etc., até dois, que aparecem todos no teclado e que podem ser obtidos quando o cursor estiver ▙. Podem-se ainda ver os mesmos caracteres em inverso vídeo (branco no preto); os quais se podem também obter a partir do teclado. Se escrever **GRAPHICS** (SHIFT 9), o cursor aparecerá como ▛ : que significa modo/gráfico. Se primir um símbolo, ele aparecerá sob a forma de inverso vídeo, o que continuará a fazer até primirmos novamente **GRAPHICS** ou **NEWLINE. RUBOT** terá o significado normal. Tenha cuidado e não deixe perder o cursor ▛ entre outros caracteres inversos que vai acabar de escrever.

Depois de experimentar um pouco verá que o conjunto de caracteres continua no topo do écran; se não estiverem, terá que se passar novamente o programa. Logo no princípio está um espaço e dez padrões em preto, branco e cinzento; mais à frente estão mais onze. Estes denominam-se símbolos gráficos e utilizam-se para construção de desenhos. Pode metê-las no programa através do teclado, usando novamente o modo gráfico (excepto para o espaço, que é um símbolo vulgar que se obtém deslocando o cursor ▙ ; o quadrado preto é um espaço inverso). Utilizam-se as 20 teclas que têm símbolos gráficos escritos. Suponhamos, por exemplo, que quer o símbolo ▚, que está na tecla T. Prima **GRAPHICS** para obter o cursor ▛, e depois prima SHIFT T. A partir da discrição anterior do modo gráfico, dever-se-ia obter um símbolo inverso vídeo; mas SHIFT T é normalmente < >, uma comparação e estes símbolos não têm inversos, pelo o que em vez de inverso vídeo, obtém-se o símbolo gráfico ▚.

Temos aqui os 22 símbolos gráficos

Símbolo	*Código*	*Como se obteve*
	0	K ou L **SPACE**
	1	G shift 1
	2	G shift 2
	3	G shift 7
	4	G shift 4
	5	G shfit 5
	6	G shift T
	7	G shift E
	8	G shift A
	9	G shift D
	10	G shift S

Símbolo	*Código*	*Como se obteve*
	128	G **SPACE**
	129	G shift Q
	130	G shift W
	131	G shift 6
	132	G shift R
	133	G shift 8
	134	G shift Y
	135	G shift 3
	136	G shift H
	137	G shift G
	138	G shift F

Vejamos novamente o conjunto de caracteres. Os sinais aparecem claramente divididos em dois blocos: um pequeno grupo de três (**RND, INKEY$** e **PI**) depois de 2. e um grupo maior (começando na imagem de aspas a seguir a Z, e continuando de AT até **COPY**) todos os restantes caracteres parecem ser "?". Esta é a forma como eles aparecem escritos — o verdadeiro ponto de interrogação está entre : e (. Os que estão situados acima, alguns são caracteres de control como ◊, **EDIT** e **NEWLINE**, e os restantes são para caracteres, que não tem absolutamente nenhum significado especial para o ZX81.

Sumário

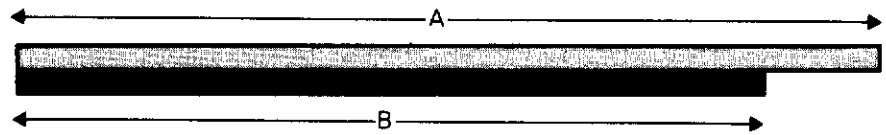
Funções: **CODE, CHR$**

Exercícios

1. Imagine que o espaço para um símbolo gráfico está dividido em quatro quartos:  . Se cada quarto puder ser preto ou branco, existem 2*2*2*2=16 possibilidades. Encontre todos no conjunto de caracteres.

2. Suponha que o espaço para um símbolo está dividido em duas partes horizontais:  . Se cada metade puder ser preta, branca ou cinzenta, existem 3*3=9 possibilidades. Encontre-as.

3. Os caracteres no exercício 2 estão concebidos para serem utilizados em Gráficos de barras horizontais, com duas cores: cinzento e preto. Escreva um programa que utilize dois números A e B (ambos entre 0 e 32) e desenhe para eles um gráfico de barras.



Achará necessário começar por escrever "▩", e mude para "▨" ou "▬", consoante A for maior ou menor que B.

Que fará o seu programa se A e B não forem números inteiros? Ou se não estiverem na escala de 0 a 32? Um bom — "sensível amigo utilizador" é o termo adequado — programa já fará algo sensível e útil.

4. Existem dois caracteres diferentes completamente cinzentos no teclado, em A e H. Se reparar neles de perto, verá que o caracter H é uma miniatura de um tabuleiro de xadrez, enquanto que A é um tabuleiro de xadres visto de lado. Escreva-os um ao lado de outro e verá que se ajustam apropriadamente. O de A é utilizado para se ajustar com ▨ e ▬ (em S e D), enquanto o H se ajusta nitidamente com ▩ e ▩ (em F e G).

5. Passe este programa

```
10 INPUT A
20 PRINT CHR$ A;
30 GOTO 0
```

Se o passar, verificará que para CRH$, A é arredondado ao número inteiro mais próximo; se A não estiver na escala de 0 a 255, o programa pára com o relatório B.

6. Utilizando os códigos para os caracteres, podemos estender o conceito de "ordenação alfabética" para abranger "strings" com qualquer caracter que não sejam só letras. Se, em vez de utilizar o alfabeto normal de 26 letras utilizaremos o alfabeto de 256 caracteres, pela ordem dos seus códigos; o princípio é exactamente o mesmo. Estes "strings", por exemplo, estão pela ordem alfabética do ZX81:

```
"  ZACHARY"
"■"
"(ASIDE)"
"123 TAXI SERVICE"
"AASVOGEL"
"AA RDVARK"
"ZACHARY"
"AA RDVARK"
```

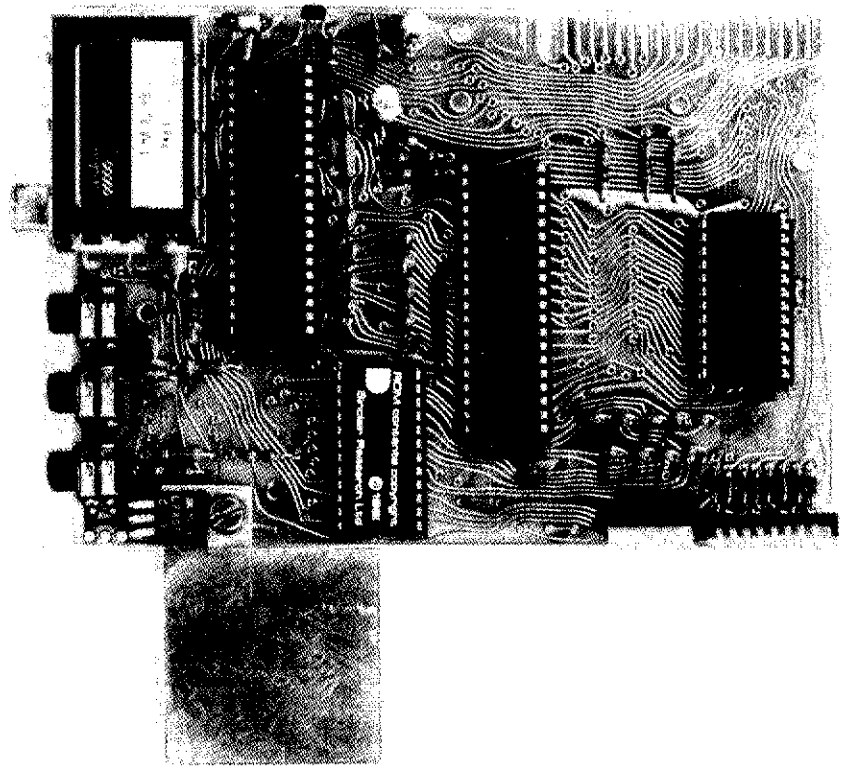
A regra é a seguinte. Primeiro, compare os primeiros caracteres de duas "strings". Se estes forem diferentes, uma delas tem o código menor que o outro, e a "string" da qual é o primeiro caracter é a primeira (menor) das duas "strings". Se forem iguais, continue a comparar os caracteres seguintes. Se, neste processo, uma das "strings" sair primeiro que a outra, essa é a primeira string; se não, são iguais.

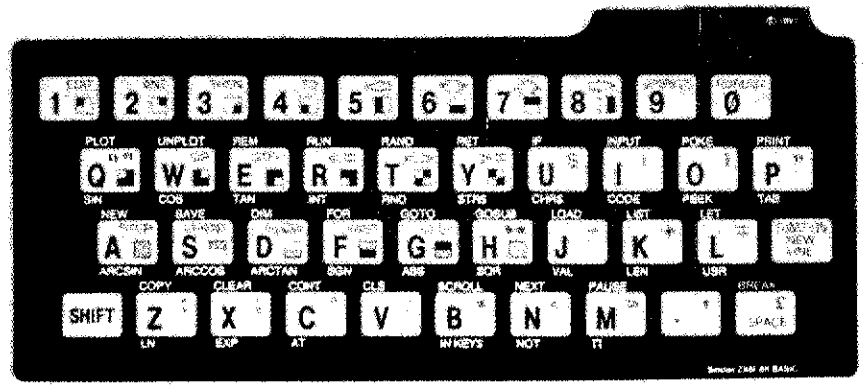
Escreva novamente o programa do exercício 4, Capítulo 10 (a que dá entrada de duas "strings" e as põe por ordem), e experimente-o.

7. Este programa enche um écran de caracteres gráficos pretos e brancos ao acaso:

```
10 LET A=INT (16*RND)
20 IF A>=8 THEN LET A=A+120
30 PRINT CHR$ A;
40 GOTO 10
```

(Como é que funciona?)





Capítulo 12

Ciclos repetitivos

Suponha que quer fazer entrar cinco números e adicioná-los. Uma maneira de o fazer (só escreva isto se não acreditar) é 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 não é muito prático. Isto pode ser feito para cinco números, mas seria bastante cansativo utilizar um programa destes para adicionar dez números, e adicionar cem seria impossível.

O melhor é definir uma variável para contar até cinco e depois parar o programa, como por exemplo (este deve-se escrever):

```
10 LET TOTAL=0
20 LET COUNT=1
30 INPUT A
40 REM COUNT = NUMBER OF TIMES
FAR
50 LET TOTAL=TOTAL+A
60 LET COUNT=COUNT+1
70 IF COUNT<= 5 THEN GOTO 30
80 PRINT TOTAL
```

Como pode ver é extremamente fácil modificar a linha 70 para que este programa some dez números, ou mesmo uma centena.

Esta forma de contar é tão útil que existem duas instruções especiais que a tornam mais fácil: a instrução **FOR,** e a instrução **NEXT.** Usam-se sempre juntas. O programa que acabou de introduzir faz precisamente o mesmo que este que utiliza estas 2 instruções

```
10 LET TOTAL = 0
20 FOR C = 1 TO 5
30 INPUT A
40 REM C = NÚMERO DE VEZES
50 LET TOTAL = TOTAL+A
60 NEXT C
80 PRINT TOTAL
```

(Para se obter este programa a partir do anterior, tem que transformar as linhas 20, 40, 60 e 70. **TO** é SHIFT 4).

Note que alteramos COUNT para C. A variável que conta — ou variável de controle — de um ciclo **FOR-NEXT** deve ter um nome com uma só letra.

O efeito deste programa é que C varia entre os valores 1 (valor inicial), 2, 3, 4 e 5 (o limite), e as linhas 30, 40 e 50 são executadas para cada um deles. Quando C tiver passado pelos seus cinco valores, a linha 80 será executada.

Há ainda um outro requinte extra que é o facto de a variável de controle não ter necessariamente que ser incrementada de 1: pode-se alterar este 1 para qualquer outra coisa que queira, utilizando **STEP** na instrução **FOR.** A forma mais geral de uma instrução **FOR** é

FOR VARIÁVEL DE CONTROLE = VALOR INICIAL **TO** LIMITE
STEP INCREMENTO

em que a variável de controle é uma só letra, e o valor inicial, limite e passo são qualquer expressão numérica. Assim, se substituir a linha 20 no programa por

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

então C varia entre os valores 1, 2.5 e 4. Repare-se que não necessitamos limitar-nos aos números inteiros, e que o valor de controle não tem que atingir exactamente o limite superior — continuando no ciclo enquanto for menor ou igual ao limite (veja exercício 4).

Deve-se ter cuidado quando se tem dois ciclos **FOR-NEXT,** um dentro do outro. Tente fazer este programa que descreve um dominó completo.

```
10 FOR M=0 TO 6
20 FOR N=0 TO M
30 PRINT M;":";N;" ";   } ciclo N
40 NEXT N                } ciclo M
50 PRINT
60 NEXT M
```


Como se pode verificar, o ciclo N está completamente dentro do ciclo M, como mostram as chavetas. O que se deve evitar são os ciclos **FOR-NEXT** que se sobrepõem sem estarem completamente dentro do outro, como:

ERRADO

```
10 FOR M=0 TO 6
20 FOR N=0 TO M          } ciclo M
30 PRINT M;":";N;" ";    }
40 NEXT M                } ciclo N
50 PRINT                 }
60 NEXT N                }
```

Dois ciclos **FOR-NEXT** devem estar ou metidos um no outro ou completamente separados.

Outra coisa a evitar é saltar de fora para dentro de um ciclo **FOR-NEXT.** A variável de controle só é estabelecida quando a sua instrução **FOR** for executada, e se **NEXT** não for executada, baralha o computador. Com um pouco de sorte dever-se-á obter o relatório de erro 1 ou 2 (que significa que uma instrução **NEXT** não contém uma variável de controle conhecida).

Sumário

Funções: **FOR, NEXT**
TO, STEP

Exercícios

1. Escreva novamente o programa do Capítulo 11 que mostra o conjunto de caracteres, utilizando um ciclo **FOR-NEXT** (resposta no Capítulo 13).

2. Uma variável de controle não tem apenas um nome e um valor, como uma variável comum, mas tem também um limite, um passo, e um número de linha para continuar o ciclo (a linha depois da instrução **STEP** em que ela foi estabelecida). Em primeiro lugar, tem de perceber que toda a informação está disponível quando **FOR** for executada (usando o valor inicial como primeiro valor a tomar) e, em segundo lugar, (dando como exemplo o nosso 2.° e 3.° programa), que esta informação é suficiente para transformar a linha

```
NEXT C
```

nas duas linhas

```
LET C = C+1
IF C < =THEN GOTO 30
```

(De facto, alterámos isto um pouco: deveria ser **GOTO** 21 em vez de **GOTO** 30. O efeito, no nosso programa, é o mesmo).

3. Passe o terceiro programa, e depois escreva

```
PRINT C
```

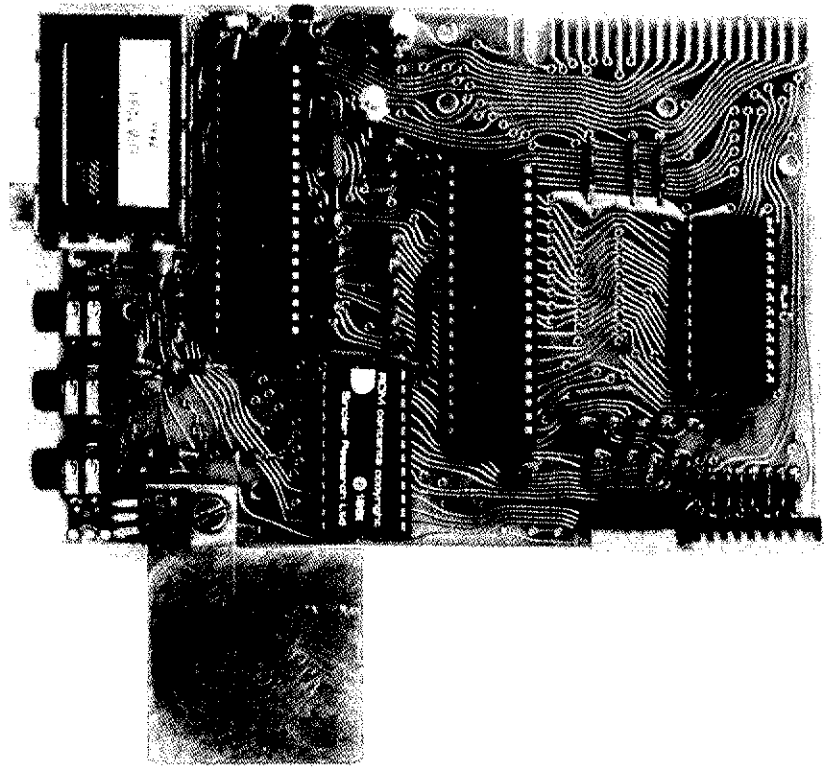
(Porque é que a resposta é 6, e não 5?)
Resposta: a instrução **NEXT** na linha 60 é executada 5 vezes e, de cada uma das vezes soma-se 1 a C.
O que acontece se puser **STEP** 2 na linha 20?

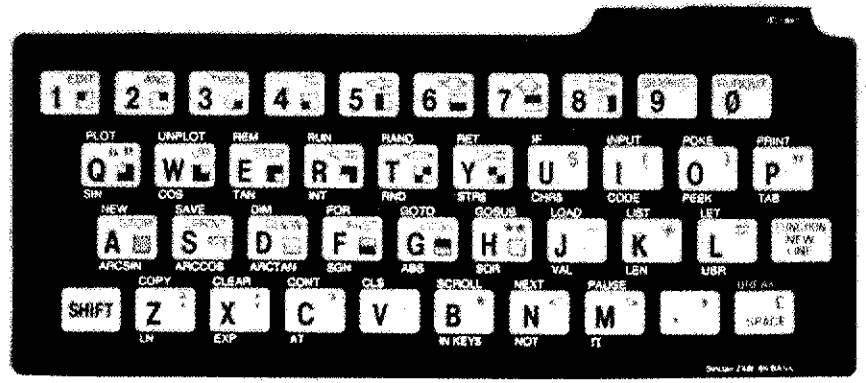
4. Altere o terceiro programa de modo a que em vez de somar automaticamente cinco números, ele pede que lhe pergunte quantos números é que quer somar. Quando passar este programa, o que acontece se escrever 0 (o que significa que não quer adicionar números)? Porque espera que isto cause problemas ao computador, mesmo que seja perfeitamente claro o que se quer? O computador tem que procurar a instrução **NEXT** C, o que normalmente não é necessário. A verdade é que tudo isto está previsto.

5. Tente fazer este programa, para escrever números de 1 a 10 na ordem inversa.

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

Converta isto num programa que não utilize ciclos **FOR-NEXT** da mesma forma que converteria o programa 3 no programa 2 (veja exercício 2). Porque é que o passo negativo o modifica ligeiramente?





Capítulo 13

Lento e rápido

O ZX81 pode trabalhar em duas velocidades — SLOW (LENTO) e FAST (RÁPIDO). Quando se liga pela primeira vez, o computador trabalha no modo SLOW podendo-se programar e dar informações simultaneamente no écran. Este modo é ideal para desenhos animados.

Contudo, pode trabalhar numa velocidade quatro vezes maior, em que o computador se esquece da imagem excepto quando não tem mais nada que fazer. Para ver como isto funciona, escreva

```
FAST
```

Agora sempre que prime uma tecla, o écran cintilará — porque o computador deixa de dar a imagem enquanto resolve a tecla que se primiu.

Escreva, por exemplo, este programa

```
10 FOR N=0 TO 255
20 PRINT CHR$ N;
30 NEXT N
```

Quando o passar, todo o écran ficará numa espécie de cinzento até ao fim do programa, quando o resultado da instrução **PRINT** aparecer no écran.

O écran aparece também durante uma instrução **INPUT**, enquanto o computador espera que introduza o valor para **INPUT**. Tente fazer este programa:

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

Para se voltar ao modo normal (computação e imagem), escreva

```
SLOW
```

É apenas uma questão de gosto, o facto de se querer o modo computação e imagem para obter nitidez, ou o modo rápido para velocidade, mas normalmente usa-se o modo rápido quando

(a) O seu programa contém uma série de cálculos numéricos, em especial se não tem muitas imagens, e no caso de as haver estas não precisam de demorar muito tempo ou

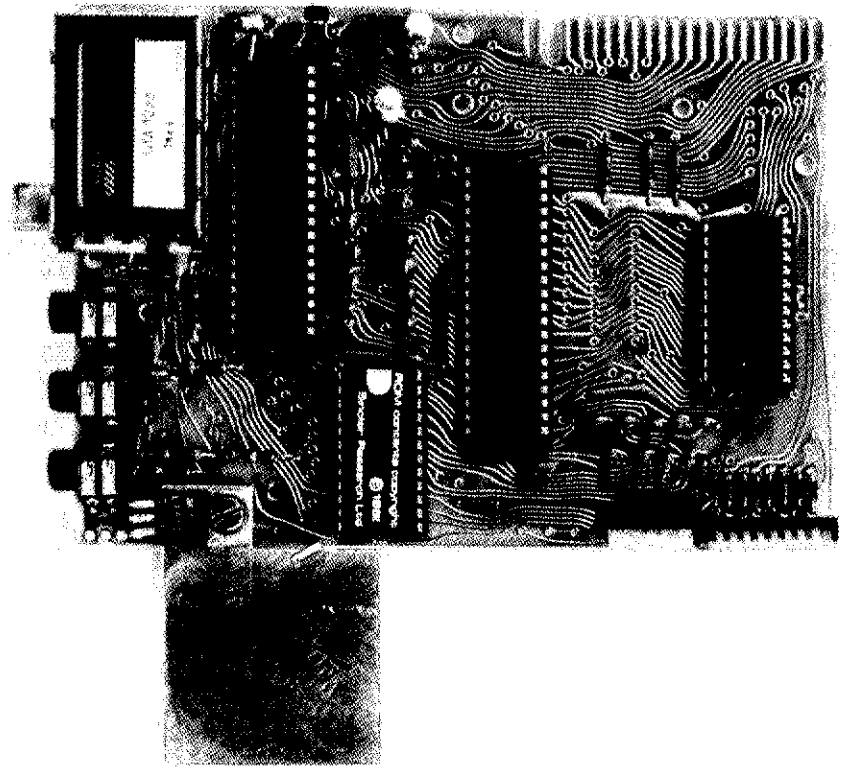
(b) Estiver a fazer um programa longo. Já deve ter reparado que a listagem volta a ser feita sempre que se introduz uma nova linha, o que se torna maçador.

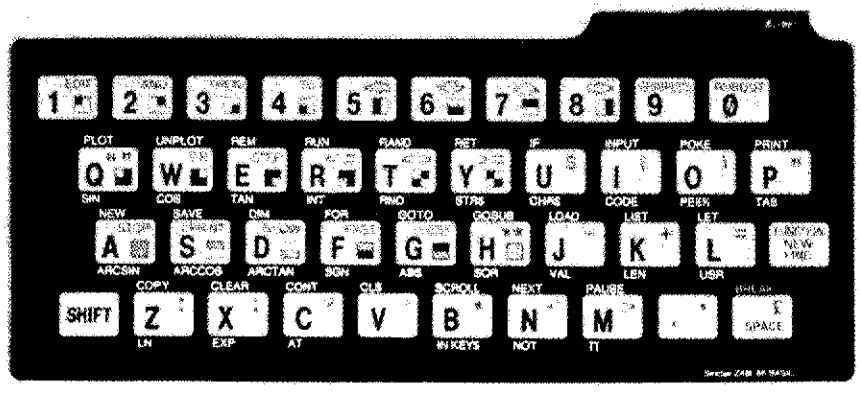
Pode utilizar as instruções **SLOW** e **FAST** nos programas sem qualquer problema
Por exemplo

```
10 SLOW
20 FOR N=1 TO 64
30 PRINT "A";
40 IF N = 32 THEN FAST
50 NEXT N
60 GOTO 10
```

Sumário

Instruções: **FAST**. **SLOW**





Capítulo 14

Subrotinas

Há por vezes, partes diferente do programa que têm funções semelhantes a desempenhar, e poderá acontecer que se esteja a escrever as mesmas linhas duas ou três vezes; contudo, isto não é necessário. Pode escrever as linhas uma vez, sob designação de subrotina e depois usá-las ou chamá-las ao programador, em qualquer sítio sem ter que tornar a escrevê-las.

Para isso utilize as instruções **GOSUB** (GO (vá para) SUB rotina) e **RETURN.**

GOSUB n

onde n é o número da 1.ª linha da subrotina, e como **GOTO** n à excepção de que o computador se lembra do número da linha da instrução **GOSUB** de modo a que possa voltar a ela depois de se fazer a subrotina. Ele lembra-o pondo o número da linha (a direcção de retorno) no cimo de uma série de endereços (o stack de GOSUB rotinas).

RETURN

toma o primeiro da pilha do stack **GOSUB,** e vai para a linha seguinte

Como primeiro exemplo,

```
10 PRINT "ESTE É O PROGRAMA PRINCIPAL",
20 GOSUB 1000
30 PRINT "NOVAMENTE";
40 GOSUB 1000
50 PRINT "É TUDO"
60 STOP
1000 REM A SUBROTINA COMEÇA AQUI
1010 PRINT "ESTA É A SUBROTINA,"
1020 RETURN
```

A instrução **STOP** na linha 60 é muito importante pois, de outro modo o programa passaria para dentro da subrotina e daria o erro 7 quando se alcançasse a instrução **RETURN.**

Dando um exemplo mais simples: suponhamos que queríamos escrever um programa de computador para se tratar com libras, xelins e dinheiros. (Quem tiver boa memória, lembrar-se-á que, antes de 1971, uma libra estava dividida em vinte xelins — 1 xelim eram 5 dinheiros — e que um xelim se subdividia em onze dinheiros antigos; d era a abreviatura de um dinheiro antigo). Terá três variáveis L, S e D (e talvez outras — L1, S1, D1, etc.) e a aritmética é muito fácil. Primeiro trabalha-se as libras separadamente dos xelins e dinheiros — por um exemplo, para somar duas quantias, soma-se os dinheiros, os xelins e as libras; para duplicar uma quantia duplica-se os dinheiros, os xelins e as libras, etc. Quando tudo estiver feito, acerte-os à forma correcta para que os dinheiros fiquem entre 0 e 11, e os xelins entre 0 e 19. Esta última fase é comum em todas as operações pelo que se pode transformá-la numa subrotina.

Pondo de parte a noção de subrotinas durante alguns instantes, é preferível tentar fazer o programa sozinho. Dados os números arbitrários L, S e D, como convertê-los

devidamente para libras, xelins e dinheiros? O problema é que se começará a imaginar problemas cada vez mais complexos e excêntricos.

O que nos vem logo à ideia será algo como £1..25s..17d, que terá que converter em £2..6s..5d. O que não é muito difícil. Mas suponha que tem números negativos? Um défice de £1..25s..17d, ou £−1..−25s..−17d, pode converter-se em £−3..13s..7d, que é uma maneira pouco comum de o exprimir (pois as pessoas só emprestam libras). E quanto a fracções? Se dividir £1.25s..17d por dois, obtém £ .5..12.5s..8.5d, apesar de ter os dinheiros 8.5, entre 0 e 11; os xelins, 12.5, entre 0 e 19, não ficam tão bem como £1..3s..2.5d. Tente obter uma resposta para tudo isto — e utilize-as num programa de computador — antes de continuar a ler.

Eis uma solução:

```
1000 REM SUBROTINA PARA REGULAR L, S, D PARA A FORMA
     USUAL PARA LIBRAS, XELINS E DINHEIROS
1010 LET D = 240*L+12*S+D
1020 REM AGORA ESTÁ TUDO EM DINHEIROS
1030 LET E = SGN D
1040 LET D = ABS D
1050 REM TRABALHAMOS COM D POSITIVO, MANTENDO O SEU
     SINAL EM E
1060 LET S = INT (D/12)
1070 LET D (D—12*S)*E
1080 LET L = INT (S/20)*E
1090 LET S = S*E—20*L
1100 RETURN
```

Só por si, isto não é muito útil porque não há programa para estabelecer previamente L, S e D, nem para trabalhar com eles posteriormente. Escreva este programa principal, bem como a subrotina para escrever L, S e D.

```
  10 INPUT L
  20 INPUT S
  30 INPUT D
  40 GOSUB 2000
  45 REM ESCREVE OS VALORES
  50 PRINT
  60 PRINT " □□□ = ";
  70 GOSUB 1000
  75 REM O ACERTO
  80 GOSUB 2000
  85 REM ESCREVE OS VALORES
  90 PRINT
 100 GOTO 10
2000 REM SUBROTINA PARA ESCREVER L, S E D
2010 PRINT "  ";L;"..";S;S..";D;"D";
2020 RETURN
```


(Lembre-se do Capítulo 9 em que se viu que, a instrução **PRINT** vazia na linha 50 escreve uma linha em branco).

O que fizemos foi poupar linhas de programa servindo-nos da subrolina escrita em 2000 o que, só por si, é uma vantagem das subrotinas: encurtar os programas. Contudo as subrotinas tornam um programa mais moroso; assim, o comprimento do programa não é o único considerando. Utilizadas devidamente, as subrotinas podem tornar os programas mais compreensíveis a quem interessa — os seres humanos.

O programa principal é simplificado através de declarações mais fortes: cada **GOSUB** representa algumas coisas complicadas do BASIC, mas apenas se deve ter em conta o resultado final. Daí, que seja mais fácil apreender a estrutura principal do programa.

Por outro lado, as subrotinas são simples por uma razão muito diferente, nomeadamente pelo facto de serem mais curtas. Usam ainda as mais velhas e persistentes instruções **LET** e **PRINT**, mas têm apenas que fazer uma parte do trabalho e, são consequentemente mais fáceis.

A habilidade está na escolha do nível — ou níveis — onde se vão escrever as subrotinas. Devem ser suficientemente grandes para terem um impacto significativo no programa principal, embora tenham que ser suficientemente pequenas para serem bastante mais fáceis de escrever do que um programa completo sem subrotinas. Estes exemplos (não recomendados) são ilustrativos:

```
  10 GOSUB 1000
  20 GOTO 10
1000 INPUT L
1010 INPUT S
1020 INPUT D
1030 PRINT " ";L;"..";S;"S..";D;"D";TAB 8;"=";
1040 LET D=240*L+12*S+D
   :     :
   :     :
2000 RETURN
```

E segundo

```
  10 GOSUB 1010
  20 GOSUB 1020
  30 GOSUB 1030
  40 GOSUB 1040
  50 GOSUB 1050
   :     :
   :     :
```

```
  80 GOTO 10
1010 INPUT L
1015 RETURN
1020 INPUT S
1025 RETURN
1030 INPUT D
1035 RETURN
1040 PRINT " ";L;"..";S;"S..";D;"D";TAB 8; "= ";
1045 RETURN
1050 LET D=240*L+12*S+D
1055 RETURN
   :    :
   :    :
```

O primeiro, com a sua subrotina simples e forte, e o segundo, com as suas trivialidades, demonstram extremos opostos, mas com igual futilidade.

Felizmente, uma subrotina pode chamar outra, ou a ela própria (uma subrotina que se chama a si mesma é dita recursiva), por isso não receie ter vários níveis.

Sumário

Instruções: **GOSUB**, **RETURN**

Exercícios

1. O programa exemplificativo é quase um calculador universal LSD. Como utilizá-lo:
   a) para converter libras e dinheiros novos em libras, xelins e dinheiros?
   b) para converter guinéus* em libras e xelins? (1 guinéu = 1..1s).
   c) para achar as fracções de uma libra? (i.g. um terço de uma libra, é 6s..8d.
   Escreva uma linha para arredondar o dinheiro para a sua quarta parte (1/4d).
2. Junte duas declarações ao programa:

```
4 LET ACERTO = 1000
7 LET LSDPRINT = 2000
```

e altere

GOSUB 1000 para **GOSUB** ACERTO
GOSUB 2000 para **GOSUB** LSDPRINT

Isto funciona precisamente como esperava. De facto, o número de linha numa instrução **GOSUB** (ou **GOTO** ou **RUN**) pode ser qualquer expressão numérica. (Não espere que isto funcione em computadores diferentes do ZX81, porque isto não é BASIC standardizado).

Isto é realmente vantajoso e pode melhorar grandemente a clareza dos seus programas.

3. Volte a escrever o programa principal do exemplo de modo a fazer algo diferente, continuando a utilizar as mesmas subrotinas.

4.
```
... GOSUB n
... RETURN
```

em linhas consecutivas pode ser substituído por

```
... GOTO n
```

Porquê?

5. Uma subrotina pode ter vários pontos de entrada. Por exemplo, devido à forma como o nosso programa principal as utiliza, com **GOSUB 1000** seguido imediatamente de **GOSUB 2000**, pode-se substituir as nossas duas subrotinas por uma grande que faça o acerto de L, S e D, e os escreva depois. Têm dois pontos de entrada: um no princípio para a subrotina completa e outro mais adiante só para a parte impressora.
   Reajuste, conforme necessário.

6. Passe este programa

```
10 GOSUB 20
20 GOSUB 10
```

Os endereços de retrocesso são guardados no stack **GOSUB** sucessivamente, mas nunca conseguem sair e possivelmente não haverá espaço para mais no computador. Nesse caso, o programa pára com erro 4 (ver apêndice B).

Poderá ter dificuldade em voltar a limpar o stack sem perder tudo, mas poderá fazê-lo.

a) Apaque as duas declarações **GOSUB**
b) Metas duas linhas novas

```
11 RETURN
21 RETURN
```

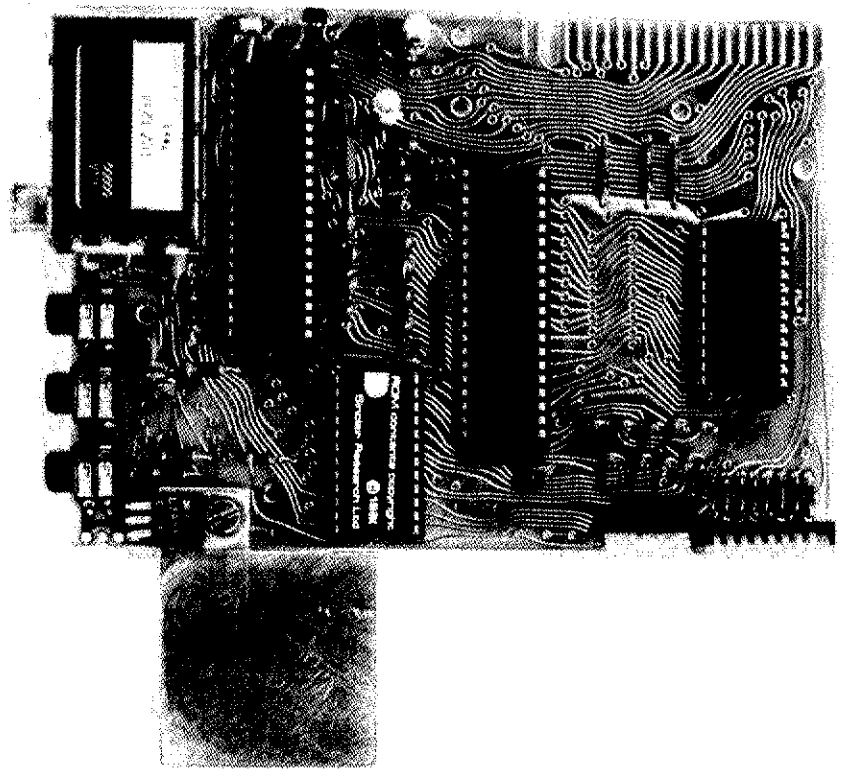
c) Escreva

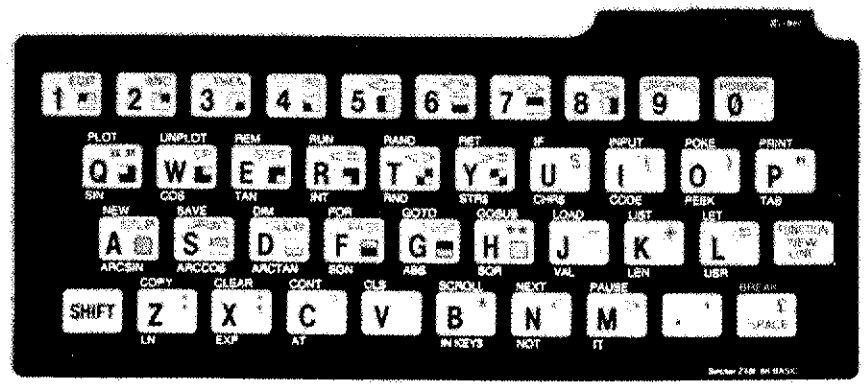
```
RETURN
```

Os endereços de retorno podem ser tirados até obter o erro 7.
d) Altere o seu programa para que não lhe volte a acontecer a mesma coisa.

Conseguiu?

NOTA: (* antiga moeda de ouro britânica, precisamente com o valor de 21 xelins).





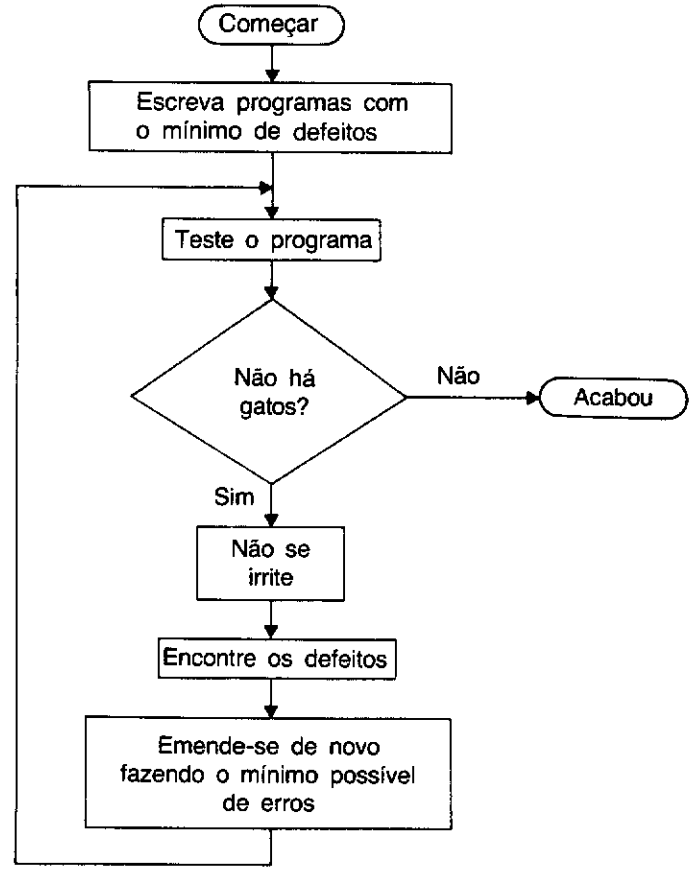
Capítulo 15

Como funcionam os programas

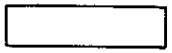
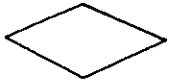
Isto é mais propriamente a arte de programar computadores do que saber o que faz cada instrução. Certamente já descobriu que a maior parte dos seus programas têm aquilo que tecnicamente se denomina de gatos da primeira vez que os escreve: mesmo ao escrever erros, ou enganos naquilo que poderia pensar que o programa deveria fazer. Poder-se-ia atribuir isto à inexperiência, mas estaria a enganar-se a si próprio.

QUALQUER PROGRAMA COMEÇA COM DEFEITOS

Muitos programas também acabam com erros. Há duas soluções para isto, primeiro, deve testar correctamente os seus programas; segundo, não adianta perder a calma sempre que eles não funcionam. Podemos ilustrar o plano geral com um fluxograma:

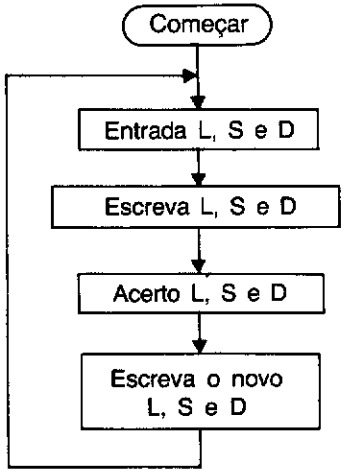


A ideia é seguir as setas de caixa para caixa, fazendo o que cada uma delas indica. Utilizámos caixas diferentes para instruções diferentes:

Redonda		começar ou acabar
Rectangular		instrução executável
Losango		pede-lhe que tome uma decisão antes de continuar

(Estas formas são largamente usadas, mas nada de importante depende delas).

É evidente que estes fluxogramas não são a melhor forma de descrever as actividades humanas; não é bom para a criatividade e flexibilidade pensar dentro destas linhas fixas. Contudo, para os computadores, é o que eles fazem realmente. São óptimos para descrever a estrutura geral dos programas como se cada caixa fosse uma subrotina, de modo que um fluxograma para o nosso exemplo do último capítulo seria.



Por conseguinte os rectângulos correspondem às **GOSUB**; apesar de, na prática, algumas caixas, como a caixa acima que dá entrada de L, S e D, sejam traduzidas directamente para instruções BASIC, sem serem subrotinas.

Tudo o que torne o programa mais claro — como os fluxogramas, subrotinas, como as declarações **REM** — tornam mais fácil a sua compreensão; e então são menos prováveis os erros. Mas as subrotinas também o ajudam a sair dos erros que se fizeram, tornando o programa mais fácil de testar. Verá que é mais fácil testar individualmente as subrotinas e assegurando-se de que elas se ajustam adequadamente, do que testar um programa completo não estruturado. Nesse caso, as subrotinas ajudam-no na caixa "encontre os defeitos", e esta é a caixa em que V. precisa de toda a ajuda que encontrar, pois é muitas vezes a mais exasperante. Outras sugestões para encontrar defeitos são:

a) Verifique se não há erros de escrita. Faça sempre isto.

b) Tente calcular o que cada variável deveria ser em cada fase — e, se possível, explicar isto em declarações **REM**. Podem-se verificar as variáveis num dado ponto do programa introduzindo aí uma declaração **PRINT**.

c) Se o objectivo do programa é fazê-lo parar com um relatório de erro, então utilize esta informação tanto quanto puder. Repare no código de erro e verifique porque é que ele parou naquela linha. Se necessário escreva os valores das variáveis.

d) É capaz de ter de percorrer o programa, linha por linha, escrevendo as suas linhas como ordens.

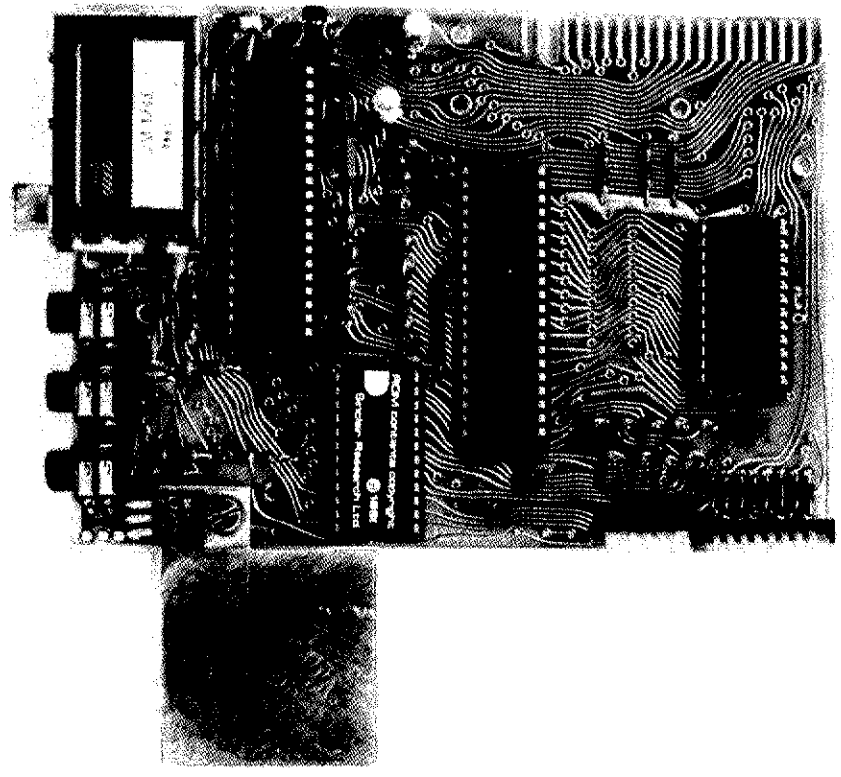
e) Suponha que é o computador: passe o programa sozinho, servindo-se do lápis e papel para apontar os valores das variáveis.

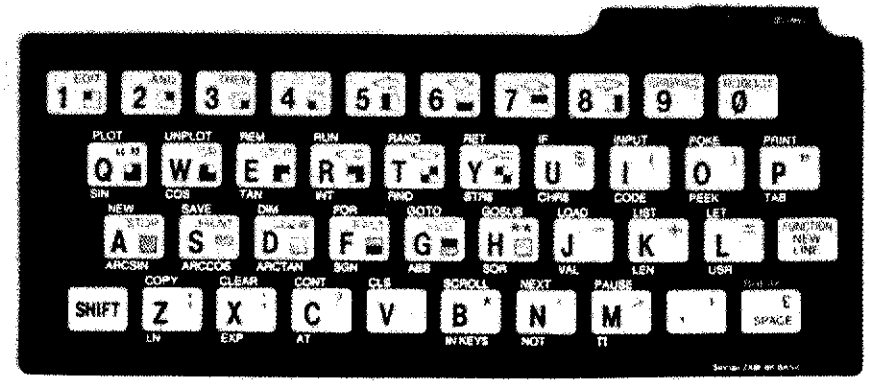
Uma vez encontrados emende-os, tendo o mesmo cuidado que ao escrever o programa original e teste-os novamente. É frequente emendar um defeito introduzindo outro.

Exercícios

1. O fluxograma para o calculador LSD não tem caixa de "fim". Faz diferença? Se quisesse, onde poderia pôr um?

2. Faça fluxogramas para os programas dos ciclos no Capítulo 12.




1 EDIT
2 CAPS
3 TRUE
4 $
5 %
6 &
7 '
8 (
9)
0
PLOT UNPLOT REM RUN RAND RET IF INPUT POKE PRINT
Q W E R T Y U I O P
SIN COS TAN INT RND STR$ CHR$ CODE PEEK TAB
NEW SAVE DIM FOR GOTO GOSUB LOAD LIST LET
A S D F G H J K L
ARCSIN ARCCOS ARCTAN SGN ABS SQR VAL LEN USR
COPY CLEAR CONT CLS SCROLL NEXT PAUSE
SHIFT Z X C V B N M SPACE
LN EXP AT INKEY$ NOT PI

Capítulo 16

Armazenamento em fita magnética

Como referimos no Capítulo 1, e que certamente já descobriu através da experiência, quando se desliga o ZX81, perdem-se todos os programas e variáveis que estavam memorizados. A única forma de os preservar, é gravá-los com um gravador. Mais tarde, poderá voltar a passá-los, pelo que o computador voltará a estar como estava quando fez a gravação.

O ZX81 trouxe um par de cabos (artigo 5 no Capítulo 1), que liga o ZX81 a um gravador de cassetes. Deverá escolher o seu próprio gravador/leitor, pois uns trabalham melhor que outros.

Em primeiro lugar e no que diz respeito ao ZX81, os gravadores portáteis são tão bons como os stereofónicos, e dão menos maçada. O contador de voltas será muito útil.

Em segundo lugar, o gravador deve ter uma entrada de microfone e uma saída amplificada (se não tiver tente a saída do altifalante), porque outras saídas não dão o sinal suficientemente forte para o ZX81. De preferência os jacks devem ter 3,5 mm (i.e., para os jacks fornecidos servirem).

Qualquer cassete serve, embora sejam melhores as cassetes de baixo ruído (low noise).

Agora que temos o respectivo gravador, vamos ligá-lo ao computador: um cabo liga a entrada do micro no gravador à entrada que diz "MIC" do lado esquerdo do ZX81, e o outro liga a saída do gravador à entrada "EAR" do ZX81. Certifique-se de que não os cruza (embora, se isso acontecer o ZX81 não se estraga).

Escreva um programa qualquer no computador, por exemplo, o programa do conjunto de caracteres no Capítulo 11. Quando gravar o programa, terá que lhe dar um nome e será melhor escrever o nome dentro do programa de modo a aparecer nas listagens — a melhor forma de o fazer é com uma declaração REM. Assim escreva

```
5 REM "CARACTERES"
```

Agora — isto é apenas um programa vazio, para se poder ver o que acontece — escreva

```
SAVE "CARACTERES"
```

e olhe para o écran. Ficará cinzento durante cinco segundos; depois, por seis segundos, poder-se-á distinguir uma mudança de padrão listas estreitas a preto e branco, e depois o écran ficará branco com o relatório 0/0. O computador esteve a emitir um sinal para a saída "MIC"; mas estava também a emitir o mesmo sinal para a televisão, dando a imagem que viu. A parte cinzenta era uma zona silenciosa, e a parte listrada era o programa.

O que se pretende é apanhar esse sinal no gravador, que é o que vamos agora fazer.

Gravar um programa

1. Ponha a fita num sítio em branco ou no qual se possa gravar.

2. Com um micro, grave falando "caracteres". Não é imprescindível, mas assim será mais fácil encontrar depois o programa. Volte a ligar o computador ao gravador.

3. Escreva

SAVE "CARACTERES" (sem **NEWLINE**)

4. Ponha o gravador em posição de gravar.

5. Prima **NEWLINE**.

6. Continue a olhar a televisão. Quando acabar (com o relatório Ø/Ø), pare de gravar.

Para se certificar de que isto funcionou, deve-se ouvir a gravação através do altifalante do gravador. (Terá que desligar o condutor da saída do gravador). Retroceda a fita para onde começou, e volte a passar.

Primeiro, ouvirá a sua voz dizendo "caracteres".

Depois, ouve-se um ruído suave. Isto não faz parte da gravação, é apenas o fim do sinal dado à televisão (antes de primir **NEWLINE**), que também foi dado ao gravador.

Há depois cinco segundos de silêncio, que são o princípio do devido sinal de gravação. Isto corresponde ao período em que o écran ficou cinzento.

Depois são seis segundos de um ruído desagradável, que seria insuportável se o volume estivesse no máximo. Isto é a gravação do programa, que corresponde ao padrão branco e preto do écran.

Finalmente, volta ao ruído suave.

Se não ouviu todos estes ruídos que descrevemos, verifique se tinha o computador e o gravador devidamente ligados. Há gravadores em que o jack não faz contacto se estiver todo metido. Tente puxá-lo para fora um pouco — por vezes sente-se ele voltar à devida posição.

Suponhamos agora que os sons de gravação parecem estar bons ao ouvido e que quer voltar a gravá-lo no computador.

Chamar um programa com um nome

1. Retroceda a fita para onde começou.

2. Certifique-se de que a saída "EAR" do computador está ligada à saída do gravador.

3. Ponha o volume do gravador em 3/4 do volume máximo; se tem controlador de tom, acerte-o de modo a que os agudos fiquem mais altos que os graves (para ficar com som sibilante).

4. Escreva

LOAD "CARACTERES" (novamente sem **NEWLINE**)

5. Ponha o gravador a funcionar.

6. Prima **NEWLINE**.

Poderá novamente ver no écran figuras daquilo que gravou, mas desta vez são diferentes, estando tudo numa configuração preto e branco. As duas partes, a silenciosa e o programa serão mais difíceis de distinguir, mas poder-se-á ver que a parte do programa tem linhas mais claras e definidas. (Tente o exercício 1 durante algum tempo).

Ao fim de quinze segundos deverá ter gravado e parado no relatório Ø/Ø. Se não o fizer, prima a tecla **BREAK** (espaço), que deverá resolver a situação.

O mais provável é que o volume não esteja correcto. Deveria estar:

a) bastante alto para a parte do programa de modo a que o computador o captasse.

b) não demasiado alto para não distorcer a parte do programa (o que é muito raro).

e c) suficiente baixe para que a parte silenciosa seja reconhecida pelo computador como silenciosa.

O melhor acerto é aumentar o volume tanto quanto puder sem que a parte silenciosa fique muito alta; pode fazer isto enquanto ouve a gravação através do altifalante; se a parte silenciosa não se conseguir acertar, deverá haver outros problemas.

Alguns gravadores estabelecem "feed back" com o ZX81. Isto só acontece quando os condutores EAR e MIC estiverem ligados ao mesmo tempo, pelo que a solução é gravar com o condutor de saída do gravador, desligado.

Há gravadores que podem gravar um zunido de fundo. A solução é pô-lo a funcionar a pilhas.

Há gravadores que são naturalmente barulhentos especialmente se forem velhos e usados. Isto soluciona-se usando um gravador de melhor qualidade, embora não fosse necessário.

Limpe a cabeça do gravador, caso esteja sujo.

Finalmente, pode haver o mesmo problema da ficha estar muito metida, como a do microfone à pouco.

Se tem um programa gravado e não se lembra do nome, pode ainda chamá-lo. (Tente fazê-lo com o programa "CARACTERES" que usou antes).

Um programa sem nome

1. Ajuste a fita para parte silenciosa.

2. Verifique tudo e acerte os controles como tinha feito antes. Terá que ser mais cuidadoso com o volume do que foi quando sabia o nome.

3. Escreva

LOAD "" (sem **NEWLINE**)

4. Ponha o gravador a funcionar.

5. Prima **NEWLINE**.
6. O resto é como anteriormente.

A questão é que se o nome do programa que pediu para chamar for uma "string" vazia, o computador grava o primeiro programa que lhe aparecer. Repare que quando tentar gravar um programa, não poderá fazer se o seu nome for uma string vazia — se tentar, obterá o erro F.

LOAD e **SAVE** também podem ser utilizados em programação. Com **SAVE**, o programa garvar-se-á sozinho de tal modo que, depois de gravado, começará imediatamente a executar a partir da linha depois da declaração **SAVE.**

Escreva, por exemplo,

```
  5 REM "INÚTIL"
 10 PRINT "É TUDO O QUE FAZ"
 20 STOP
100 SAVE "INÚTIL"
110 GOTO 10
```

Ligue o gravador e escreva

RUN 100 (sem **NEWLINE**)

comece a gravar e prima **NEWLINE**. Quando o programa se tiver gravado, continuará a passar. Verificará então que o último L em INÚTIL na linha 100 mudou para inverso vídeo, mas não se preocupe

Para gravar o programa, retroceda a fita para antes do programa, e escreva

LOAD "INÚTIL" (sem **NEWLINE**)

comece a passar novamente a fita para o computador e prima **NEWLINE**. Quando estiver gravado, continuará com a linha 110 e auto-executa-se sem que precise fazer nada.

Repare que o facto de pôr a instrução **SAVE** no fim do programa não significa que para o passar tenha que fazer **SAVE**, tem apenas que escrever **RUN** — não tem de saltar a instrução **SAVE.**

Não escreva **SAVE** dentro de uma **GOSUB** rotina — ela não funciona correctamente.

Não ponha caracteres inverso vídeo no nome de um programa. Há uma parte do nome depois do caracter inverso vídeo que se perde.

O nome não deve ter mais de 127 caracteres.

O nome, em **LOAD** ou **SAVE**, não tem que ser uma string constante, pode ser qualquer expressão com valor string, como A$ ou CHR$ 100.

Sumário

Gravar um programa

Gravar um programa que quando chamado, se auto executa
Chamar um programa pelo nome
Gravar o primeiro programa que apareça na fita
Instruções: **SAVE, LOAD**

Nota

Não pode chamar programas que foram gravados noutra espécie de computador, ou do ZX80 com o seu próprio BASIC. Os programas que gravou não podem ser passados noutros tipos de computador ou no ZX80 com o seu BASIC. Por outro lado, o ZX80 com o BASIC do ZX81 é compatível com o ZX81; os programas gravados num, podem ser passados para o outro. Depois de passar um programa do ZX80, no ZX81 será executado no modo rápido.

Exercícios

1. Grave uma fita com muitos pequenos programas e comece a passá-los para o computador, escrevendo

LOAD "SEM NOME DE PROGRAMA"

Poderá facilmente ver no écran a diferença entre as partes vazias da fita (com uma configuração preta e branca não estruturada) e os programas (com linhas mais definidas). Ambas as configurações são diferentes das que vê quando grava. Se baixar o volume enquanto está a ser passado pode ver a figura a mudar para a configuração do espaço vazio quando o sinal está demasiado baixo para ser um programa.

2. Grave uma fita em que o primeiro programa, quando for chamado, escreva uma lista (a dos outros programas da fita) e lhe peça que escolha um programa, e depois o chame (LOAD).

3. Chame de novo o programa "CARACTERES" e depois faça

LET X = 7

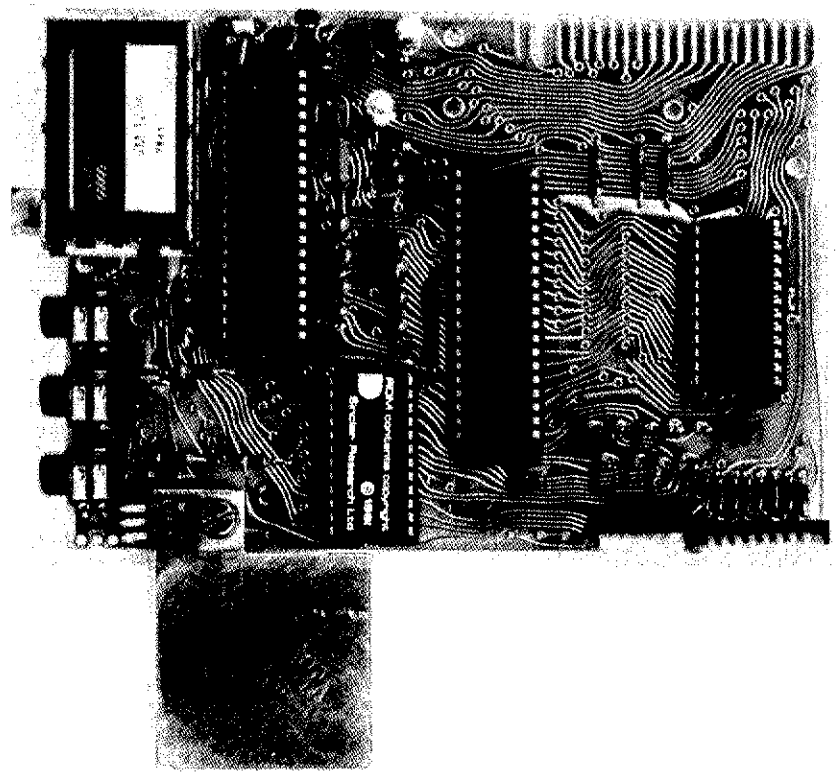
de modo a que o computador tenha uma variável X com valor 7 — embora não apareça no programa. Grave agora o programa, e depois desligue e ligue o computador (para se certificar que não há engano), e chame novamente o programa. Escreva

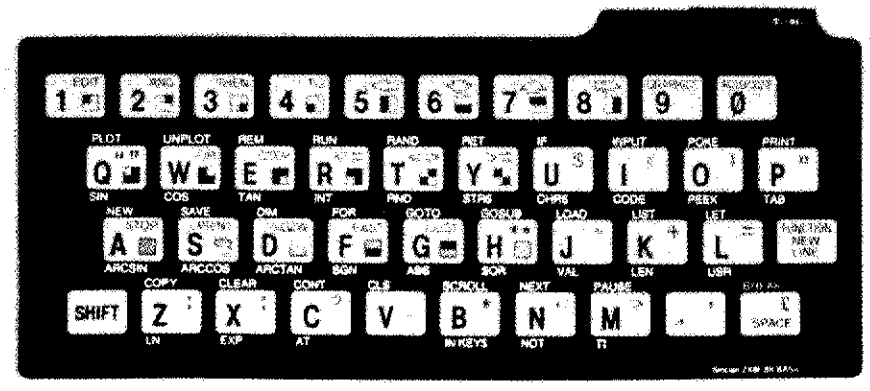
PRINT X

e terá a resposta 7. A declaração **SAVE** gravou não só o programa como também as variáveis — incluindo X.

Se quiser manter estas variáveis quando executar o programa tem de lembrar-se de usar **GOTO** e não **RUN** (como referimos no Capítulo 9). Pode evitar ter que lembrar-se disto se fizer com que o programa se auto-execute (usando **SAVE** como linha de programa).

4. Escreva um programa muito grande, depois desligue por um momento a fonte de alimentação. Estas coisas acontecem sem se esperar; não é um defeito. Não há nada a fazer. Se isso acontece mais vezes do que as que pode suportar, há algo que não está bem; habitue-se a gravar por partes os programas muito longos, conforme os vai construindo.





Capítulo 17

Imprimindo com formatos

Decerto que se recorda que uma instrução **PRINT** tem uma lista de ítems sendo cada um uma expressão (ou mesmo nada), e que estão separadas por vírgulas ou pontos e vírgulas. Há mais duas espécies de item **PRINT** que servem para dizer ao computador, não o que imprimir, mas onde imprimir. Por exemplo, **PRINT AT** 11, 16;"*" escreve um asterisco no meio do écran.

AT linha, coluna

move a posição **PRINT** (a posição onde o próximo item vai ser escrito) para a linha e a coluna especificadas. As linhas estão numeradas de **Ø** (parte de cima) a 21, e as colunas de **Ø** (à esquerda) a 31.

TAB coluna

move a posição **PRINT** para a coluna especificada. Fica na mesma linha ou, se isso implicasse retrocesso de espaço, movê-la-ia para a seguinte. Repare que o computador reduz o número da coluna a módulo de 32 (divide por 32 e fica com o resto); assim, **TAB** 33 significa o mesmo que **TAB** 1.

Por exemplo

```
PRINT TAB 30;1;TAB 12;"(CONTEÚDO)";AT 3,1;"CAPÍTULO";TAB
24;"PÁGINA"
```

(Isto seria a forma de escrever o cabeçalho da página dum livro).

Mas precisemos um pouco mais:

a) Estes novos items devem ser terminados com pontos e vírgulas, como fizemos acima. Podem-se utilizar vírgulas (ou nada, no fim da instrução), mas isto significa que, depois de ter estabelecido cuidadosamente a posição PRINT, a vai imediatamente voltar a mover — o que nem sempre é muito útil.

b) Embora **AT** e **TAB** não sejam funções, tem que primir a tecla função (shift newline) para os obter.

c) Não pode escrever nas duas linhas do fundo do écran (22 e 23) porque estas estão reservadas para os comandos, entradas **INPUT,** relatórios, etc. Quando nos referimos à "linha do fundo" normalmente significa linha 21.

d) Pode-se utilizar **AT** para pôr a posição **PRINT** onde já está algo escrito e o texto será escrito por cima do antigo.

Há ainda duas outras instruções relacionadas com o **PRINT**, nomeadamente **CLS** e **SCROLL**.

CLS limpa o écran (e nada mais).

SCROLL passa tudo o que está escrito para uma linha acima (perdendo-se a linha de cima) e move a posição **PRINT** para o princípio da linha do fundo do écran.

Para ver como funciona, passe este programa

```
10 SCROLL
20 INPUT A$
30 PRINT A$
40 GOTO 10
```

Sumário

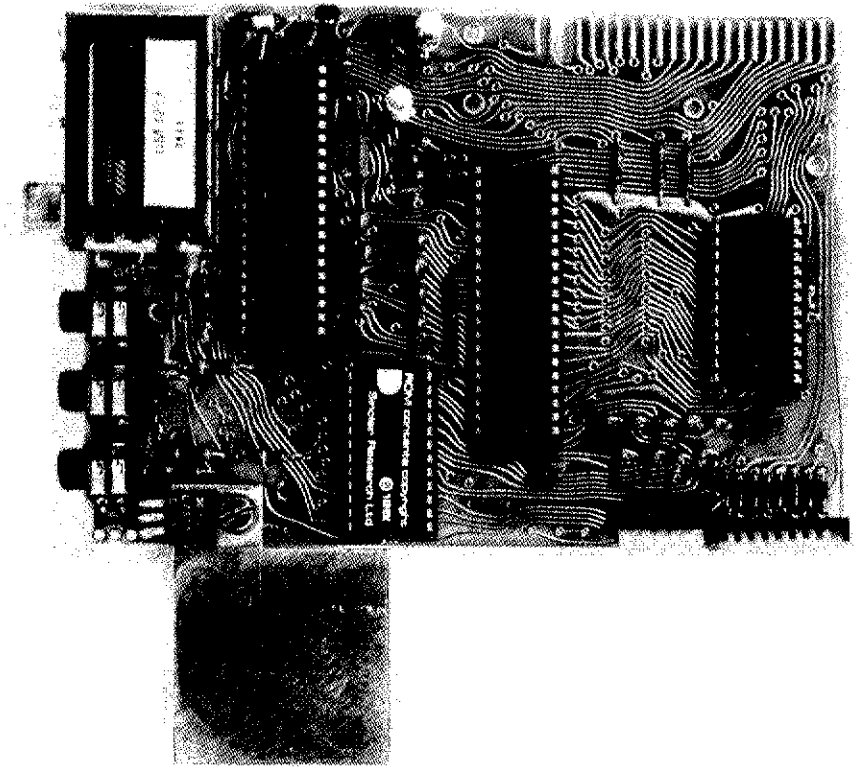
Elementos PRINT: AT, TAB
Instruções: **CLS, SCROLL**

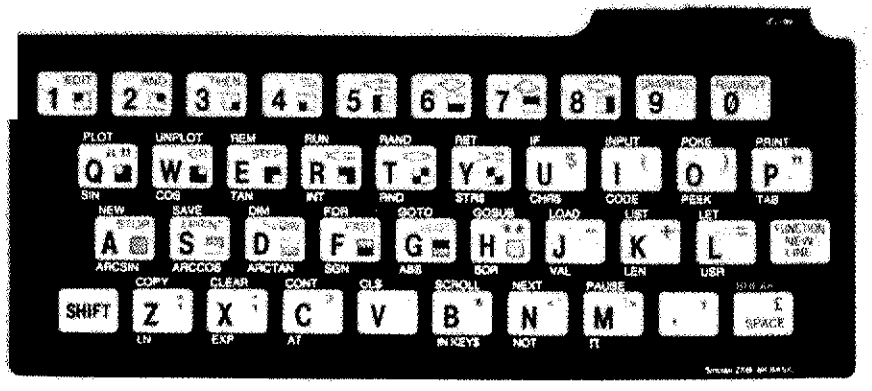
Exercícios

1. Tente executar isto:

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

Isto mostra o que significa dizer-se que o número **TAB** é reduzido a módulos de 32. Para um exemplo mais elegante, altere o 8, na linha 20, para 6.





Capítulo 18

Gráficos

Eis aqui algumas das características mais elegantes do ZX81; utilizam aquilo a que se denomina pixels (elementos de figura). O écran onde se escreve tem 22 linhas e 32 colunas, o que dá 22*32 = 704 posições de caracteres, cada qual contendo 4 pixels, divididos como fatias de bolo.

Um pixel é especificado por dois números, as suas coordenadas. A primeira, a coordenada-x, diz-nos a que distância está da coluna do lado esquerdo (recorde X é na horizontal), e a segunda a coordenada Y, diz-nos a que distância está do fundo do écran. Estas coordenadas estão normalmente escritas como um par entre parêntesis, de modo que (0,0), (63,0), (0,43) e (63,43) são os cantos inferior-esquerdo, inferior--direito, superior-esquerdo e superior-direito.

A instrução **PLOT**

PLOT coordenada-X, coordenada-Y

torna o pixel com estas coordenadas preto, enquanto que a instrução

UNPLOT coordenada-X, coordenada-Y

volta a pô-lo branco

Tente escrever este maléfico programa:

```
10 PLOT INT (RND*64),INT (RND*44)
20 INPUT A$
30 GOTO 10
```

Isto desenha um ponto numa posição de cada vez que se prime newline.

Eis um programa mais útil. Faz um gráfico da função **SIN** (uma onda seno) para valores entre 0 e 2π.

```
10 FOR N=0 TO 63
20 PLOT N, 22+20*SIN (N/32*PI)
30 NEXT N
```

O seguinte faz um gráfico de **SQR** (parte de uma parábola) entre 0 e 4:

```
10 FOR N=0 TO 63
20 PLOT N,20*SQR (N/16)
30 NEXT N
```

Repare que as coordenadas do pixel são bastante diferentes da linha e coluna num **AT**. No fim deste capítulo encontrará um diagrama que será útil para trabalhar com coordenadas pixel e números de linhas e colunas.

Exercícios

1. Há duas diferenças entre os números de **AT** e as coordenadas pixel; quais são?
Suponha que uma posição **PRINT** corresponde a **AT** L, C (por linha e coluna). Certifique-se de que os quatro pixels nessa posição têm coordenadas-X 2*C ou 2*C+1, e coordenadas-Y 2* (21-L) ou 2*(21-L)+1. Veja o diagrama.

2. Faça um queijo Gruiére fazendo um programa de modo a que o écran seja primeiro preenchido a preto (o quadrado preto é um espaço em inverso vídeo), e depois faça **UNPLOT.** Se tem apenas 1 K de memória — isto é, o aparelho central sem memória extra — terá falta de memória, pelo que deverá regular o programa de modo a utilizar apenas parte do écran.

3. Modifique o programa do gráfico SIN (com plot) de modo a que, antes de traçar o gráfico, escreva uma linha horizontal de "—" para um eixo-X, e uma linha vertical de "/"s para um eixo-Y.

4. Escreva programas para fazer o plot de mais funções por e.g. **COS, EXP, LN, ATN, INT,** etc. Em cada um deles deve verificar se o gráfico cabe no écran, e para isso deverá considerar os valores.
a) entre os quais vai tomar as funções (correspondentes aos valores de 0 a 2 π para o gráfico SIN).
b) onde pôr, no écran, o eixo-X (que corresponde ao 22 na linha 20 no programa do gráfico SIN.
c) como escalonar eixo-Y do gráfico (que corresponde ao 20 na linha 20 do programa do gráfico SIN).
Verá que **COS** é o mais fácil — é tal e qual o SIN.

5. Execute isto:

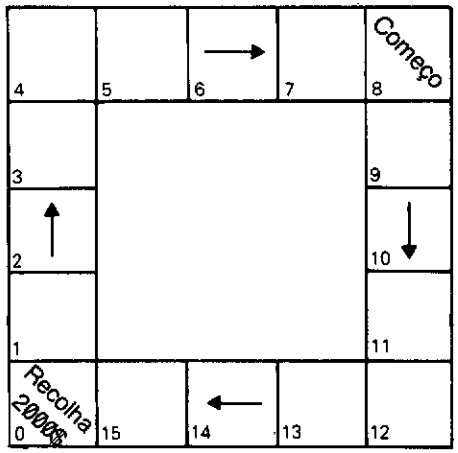
```
10 PLOT 21,21
20 PRINT "COTAÇÃO ALTA"
30 PLOT 46,21
```

PLOT muda a posição **PRINT** (**UNPLOT** também).

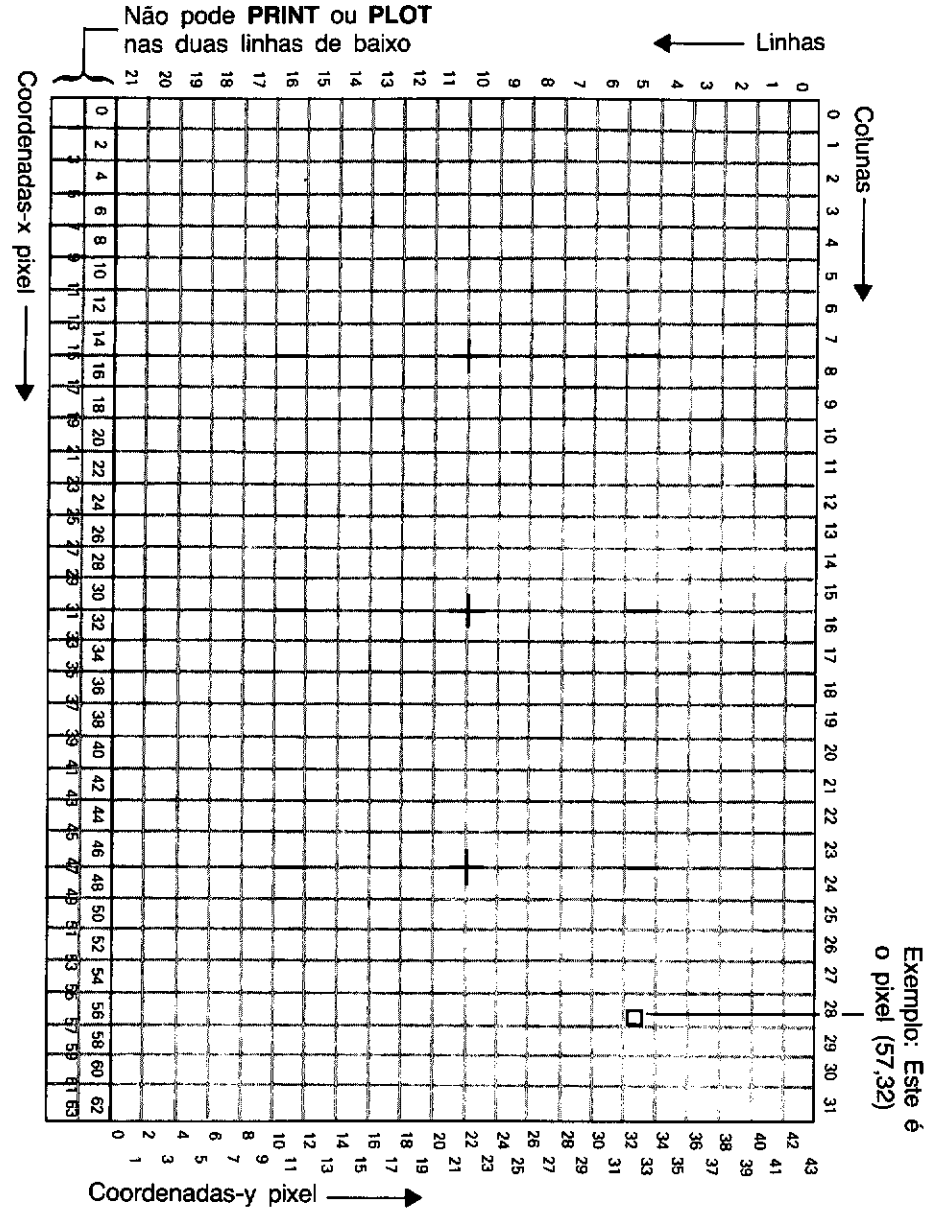
6. Esta subrotina desenha aproximadamente uma linha recta desde o pixel (A,B) ao pixel (C,D). Utilize isto como parte do programa principal que fornece os valores de A, B, C e D.
(Se não tiver uma expansão de memória, terá que omitir as **REM**).

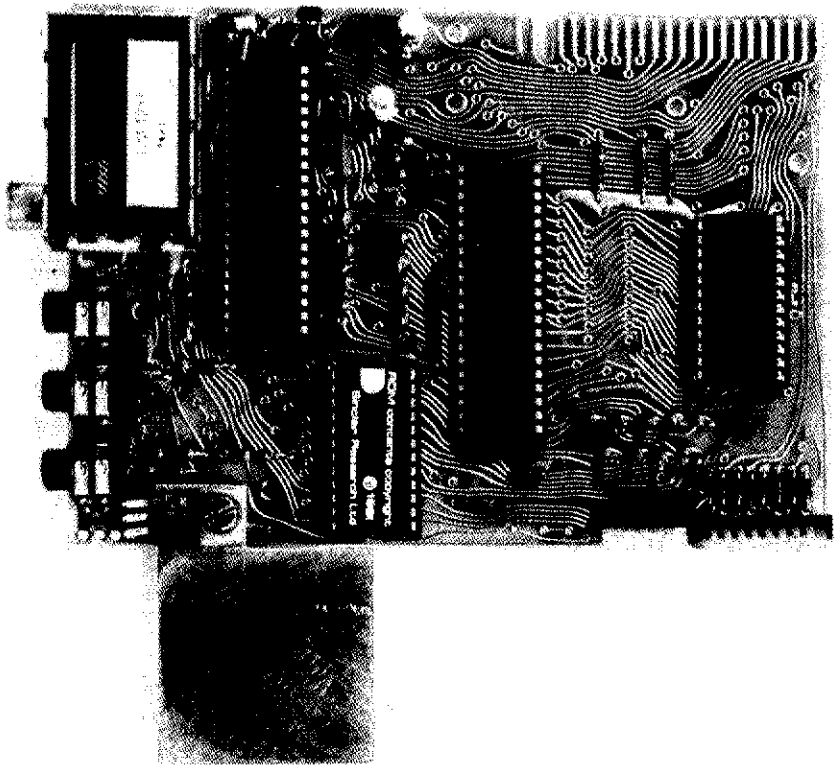

```
1000 LET U=C—A
1005 REM U INDICA QUANTOS INCREMENTOS NECESSITAMOS TER
1010 LET V=D—B
1015 REM V INDICA QUANTOS INCREMENTOS PARA CIMA
1020 LET D1X=SGN U
1030 LET D1Y=SGN V
1035 REM (D1X, D1Y) É UM SÓ INCREMENTO NUMA DIAGONAL
1040 LET D2X=SGN U
1050 LET D2Y=0
1055 REM (D2X, D2Y) É UM INCREMENTO PARA A ESQUERDA
OU PARA A DIREITA
1060 LET M=ABS U
1070 LET N=ABS V
1080 IF M> N THEN GOTO 1130
1090 LET D2X=0
1100 LET D2Y=SGN V
1105 REM AGORA (D2X, D2Y) É UM INCREMENTO PARA CIMA
OU PARA BAIXO
1110 LET M=ABS V
1120 LET N=ABS U
1130 REM M É O MAIOR DE ABS U e ABS V, N É O MAIS PEQUENO
1140 LET S=INT (M/2)
1145 REM QUEREMOS PASSAR DE (A,B) PARA (C,D) EM M
INCREMENTOS USANDO N PASSOS D2 PARA CIMA E PARA BAIXO
OU PARA A ESQUERDA E PARA DIREITA E M-N PASSOS D1
DIAGONAIS, DISTRIBUÍDOS O MAIS IGUALMENTE POSSÍVEL
1150 FOR I=0 TO M
1160 PLOT A,B
1170 LET S=S+N
1180 IF S< M THEN GOTO 1230
1190 LET S=S—M
1200 LET A=A+D1X
1210 LET B=B+D1Y
1215 REM A É INCREMENTO DIAGONAL
1220 GOTO 1250
1230 LET A=A+D2X
1240 LET B=B+D2Y
1245 REM UM ESPAÇO PARA CIMA OU PARA BAIXO, PARA
A ESQUERDA OU PARA A DIREITA
1250 NEXT I
1260 RETURN
```

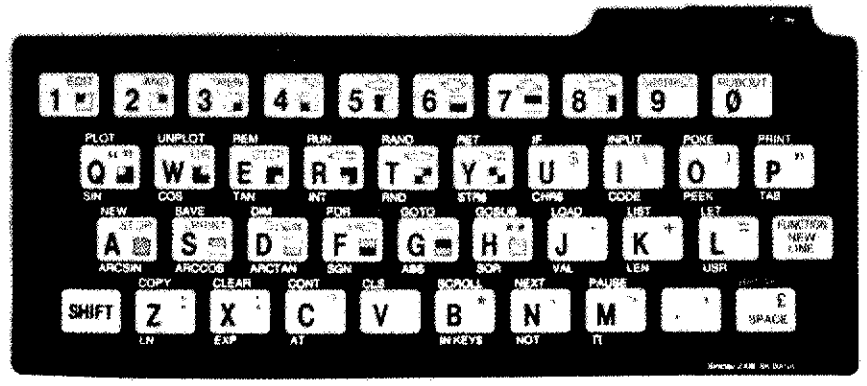
A última parte (linhas 115Ø em diante) mistura os M—N incrementos D1 homogeneamente com os N incrementos D2.
Imagine um tabuleiro de Monopólio com M quadrados de perímetro, numerados de Ø a M—1.
O quadrado em que se encontra em qualquer momento é o número S, começando a contar a partir do canto oposto ao começo. Cada jogada leva-o N quadrados para a frente, e pode levá-lo tanto da esquerda como de cima para baixo ou ainda em diagonal.
Como o jogo envolve M—N passos à volta do tabuleiro, ou N voltas; tem de passar pela casa de recolhas N vezes igualmente espaçados em relação aos M passos e destes N são da esquerda para a direita ou de baixo para cima.



Ajuste o programa de modo a que, se outro parâmetro, E, for I, então a linha é desenhada a preto (como aqui), e no caso de ser Ø, a desenhe a branco (usando **UNPLOT**). Pode então apagar a recta que acabou de desenhar, por desenhá-la a branco por cima.






1 EDIT
2 AND
3 OR
4
5
6
7
8
9
0 RUBOUT
PLOT UNPLOT REM RUN RAND NEXT IF INPUT POKE PRINT
Q SIN W COS E TAN R INT T RND Y STR$ U CHR$ I CODE O PEEK P TAB
NEW SAVE DIM FOR GOTO GOSUB LOAD LIST LET
A ARCSIN S ARCCOS D ARCTAN F SGN G ABS H SQR J VAL K LEN L USR
FUNCTION NEW LINE
COPY CLEAR CONT CLS SCROLL NEXT PAUSE
SHIFT Z LN X EXP C AT V B INKEY$ N NOT M PI
£ SPACE

Capítulo 19

Tempo e movimento

Frequentemente, quererá que o programa dure um determinado tempo, e para isso é útil a instrução **PAUSE** (em especial no modo FAST):

PAUSE n

as interrupções da Imagem de Televisão faz aparecer a figura n vezes (50 pulsos por segundo, ou 60 nos E.U.A.). n pode ir até 32767, o que lhe dá aproximadamente 11 minutos; se n for maior, significa '**PAUSE** para sempre'.

Uma pausa pode ser sempre abreviada primindo uma tecla (note que um espaço ou £, causarão também uma interrupção). Tem que primir a tecla depois da pausa começar.

No fim da pausa, o écran piscará.

Se utilizarmos uma instrução **PAUSE** num programa a passar em modo FAST (rápido), ou no ZX80 com o novo 8K ROM, a instrução **PAUSE** deve ser seguida de **POKE** 16437,255. A **PAUSE** trabalhará aparentemente sem fazermos isto, mas o resultado poderá ser a destruição do seu programa.

Este programa trabalha como o ponteiro dos segundos de um relógio (só com um ponto na ponta).

```
  5 REM PRIMEIRO DESENHAMOS O MOSTRADOR DO RELÓGIO
 10 FOR N=1 TO 12
 20 PRINT AT 10-10*COS (N/6*PI),10+10*SIN (N/6*PI);N
 30 NEXT N
 35 REM AGORA POMOS O RELÓGIO A FUNCIONAR
 40 FOR T=0 TO 10000
 45 REM T É O TEMPO EM SEGUNDOS
 50 LET A=T/30*PI
 60 LET SX=21+18*SIN A
 70 LET SY=22+18*COS A
200 PLOT SX,SY
300 PAUSE 42
310 POKE 16437,255
320 UNPLOT SX,SY
400 NEXT T
```

(Não ponha as instruções **REM**, a menos que tenha expansão de memória).

O relógio parará depois de 2h.45m devido à linha 40, mas poderá facilmente fazê-lo trabalhar mais tempo. Note-se como a regulação é controlada pela linha 300. **PAUSE** 50 deveria fazer um "tic" por segundo, mas a computação também demora um pouco e tem que ser optimizada. A melhor forma de fazer isto é através da tentativa erro, regulando o relógio do computador por um verdadeiro, e acertando a linha

300 até coincidirem. (Pode-se fazê-lo com muita precisão; um acerto de um pulso por segundo é 2% ou meia hora por dia).

As funções **INKEY$** (que não tem argumento *) lê o teclado. Se estiver a primir exactamente nessa altura uma tecla (ou SHIFT e uma outra), o resultado é o que essa tecla dá no modo L; de outro modo, o resultado é a string vazia. Os caracteres de controle não têm o efeito habitual, dão resultados como **CHR$** 118 para newline — são escritos como "?".

Tente escrever este programa, que funciona como uma máquina de escrever.

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

Neste caso a linha 10 espera que V. levante o dedo do teclado e a linha 20 espera que prima uma nova tecla.

Lembre-se que **INKEY$** não esperam por si como **INPUT** faz. Deste modo, não necessita de escrever newline, mas por outro lado, se não escrever nada, perde a sua oportunidade.

Exercícios

1. Que acontece se falhar a linha 10 no programa da máquina de escrever?
2. Porque é que não se pode dar espaço ou £ no programa da máquina de escrever?

Eis um programa, modificado, que dá um espaço se primir o cursor para a direita (SHIFT 8).

```
10 IF INKEY$<>"" THEN GOTO 10
20 IF INKEY$="" THEN GOTO 20
30 LET A$=INKEY$
40 IF A$=CHR$ 115 THEN GOTO 110
90 PRINT A$;
100 GOTO 10
110 PRINT " ";
120 GOTO 10
```

Repare como demos o valor **INKEY$** a A na linha 30. Em vez de fazermos isto, poderíamos substituir A$ por **INKEY$** nas linhas 40 e 90, mas **INKEY$** poderia variar entretanto.

Junte mais algumas linhas ao programa para que ao escrever **NEWLINE** (**CHR$** 118) obter uma nova linha.

NOTA: ARGUMENTO* — um número ou uma variável sobre o qual actua uma função.

3. Outra maneira de usar **INKEY$**, é em conjunto com **PAUSE**, como neste programa alternativo da máquina de escrever.

```
10 PAUSE 40000
20 POKE 16437,255
30 PRINT INKEY$;
40 GOTO 10
```

Para que isto funcione, porque é que é essencial que a pausa não acabe se, quando começar, V. ainda estiver a primir uma tecla?

A desvantagem deste método é que o écran pisca, mas a única maneira de o fazer é em modo rápido, e repare que o computador aproveita a oportunidade de uma pausa para projectar a imagem da TV.

4. Este dá-lhe volta à cabeça. O computador dá um número, que V. (ou uma vítima inocente), terá de voltar a escrever. Para começar, tem um segundo para o introduzir, mas se se enganar, terá mais tempo no número seguinte, enquanto que se acertar, tem ainda menos tempo no seguinte. A ideia é andar o mais depressa possível, e depois prima Q para ver a sua pontuação — quanto mais alta melhor.

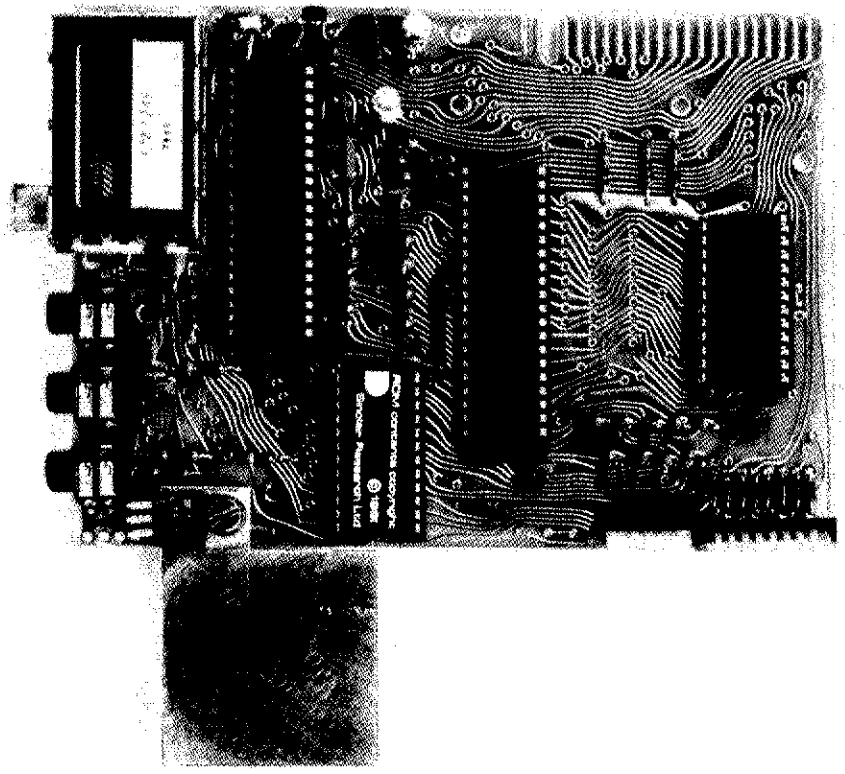
```
10 LET T=50
15 REM T= NÚMERO DE PULSOS POR JOGADA — PRIMEIRO 50
       POR SEGUNDO
20 SCROLL
30 LET A$=CHR$ INT (RND*10+CODE "0")
35 REM A$ É UM DÍGITO ALEATÓRIO
40 PRINT A$
45 PAUSE T
50 POKE 16437,255
60 LET B$=INKEY$
70 IF B$="Q" THEN GOTO 200
80 IF A$=B$ THEN GOTO 150
90 PRINT "NÃO SERVE"
100 LET T=T*1.1
110 GOTO 20
150 PRINT "OK"
160 LET T=T*0.9
170 GOTO 20
200 SCROLL
210 PRINT "A SUA PONTUAÇÃO É"; INT (500/T)
```

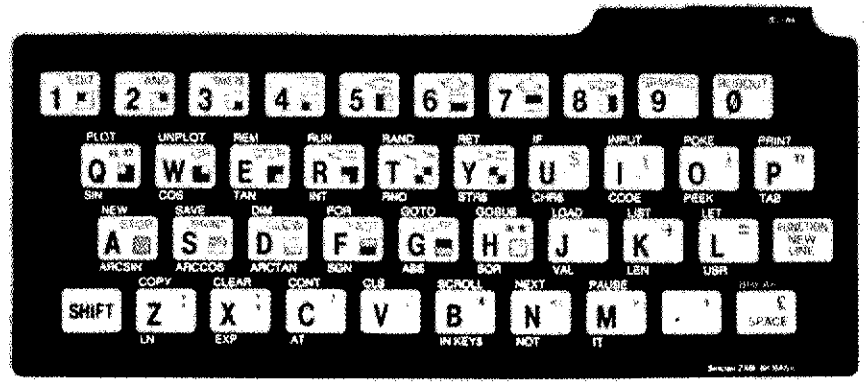
5. (Só para aqueles que têm RAM extra). Utilizando a rotina da linha recta no Capí-

tulo 15, mude o programa do relógio de modo a mostrar os ponteiros dos minutos e das horas, mudando-o a cada instante (encurte o ponteiro das horas). Se for ambicioso, programe-o para exibir algo de quarto em quarto de hora.

6. (Para brincar). Tente este:

```
10 IF INKEY$ = "" THEN GOTO 10
20 PRINT AT 11,14;"AI AI"
30 IF INKEY$<>"" THEN GOTO 30
40 PRINT AT 11,14; "      "
50 GOTO 10
```



Capítulo 20

O printer ZX81

Se tem um printer ZX81, encontrará nele algumas instruções. Este capítulo engloba as instruções necessárias para o pôr a funcionar.

As primeiras instruções, **LPRINT** e **LLIST**, são como **PRINT** e **LIST**, mas, em vez de utilizarem a TV, utilizam o printer. (O L é um acidente histórico. quando se inventou o BASIC, ele usava uma máquina de escrever eléctrica em vez da TV, pelo que **PRINT** significava mesmo imprimir. Se quisesse saídas em massa utilizaria um printer de linhas rápida ligado ao computador e, nesse caso, uma instrução **PRINT** significaria "Linha de impressão **PRINT**").

```
10 LPRINT "ESTE PROGRAMA:",,,
20 LLIST
30 LPRINT,,"ESCREVE O CONJUNTO DE CARACTERES",,,
40 FOR N=0 TO 255
50 LPRINT CHR$ N;
60 NEXT N
```

A terceira instrução, **COPY**, imprime uma cópia do écran de TV. Obtenha no écran, por exemplo, uma listagem do programa acima e escreva

```
COPY
```

Pode parar sempre o printer, menos quando está a escrever, primindo a tecla **BREAK** (espaço).

Se executar estas instruções sem ligar o printer, perderá toda a instrução, passando à instrução seguinte. Contudo, pode acontecer que ele encrave, pelo que a tecla **BREAK** o irá desencravar.

Sumário

Instruções: **LPRINT, LLIST, COPY**

Nota: Nenhuma destas instruções está em BASIC standardizado, embora **LPRINT** seja utilizada noutros computadores.

Exercícios:

1. Tente

```
10 FOR N=31 TO 0 STEP -1
20 PRINT AT 31-N,N;CHR$ (CODE "0" +N);
30 NEXT N
```

Verá que há um modelo de letras que aparecem na diagonal, desde o canto

superior esquerdo até atingir a parte de baixo do écran, quando o programa pára com o relatório de erro 5.

Agora altere **AT** 31 — N,N na linha 2Ø para **TAB** N. O programa terá precisamente o mesmo efeito.

Altere agora **PRINT**, na linha 2Ø, para **LPRINT**. Desta vez não haverá erro 5, o que não deve acontecer com o Printer, e o padrão executará dez linhas extra com os dígitos.

Agora altere **TAB** N para **AT** 21 — N,N continuando a usar **LPRINT.** Desta vez obterá uma só linha de símbolos. A razão desta diferença é que a saída de **LPRINT** não é impressa directamente, mas encontrou, numa memória tampão, uma figura do comprimento de uma linha do que o computador escreverá quando lá chegar. Começa a impressão

a) quando o tampão está cheio,

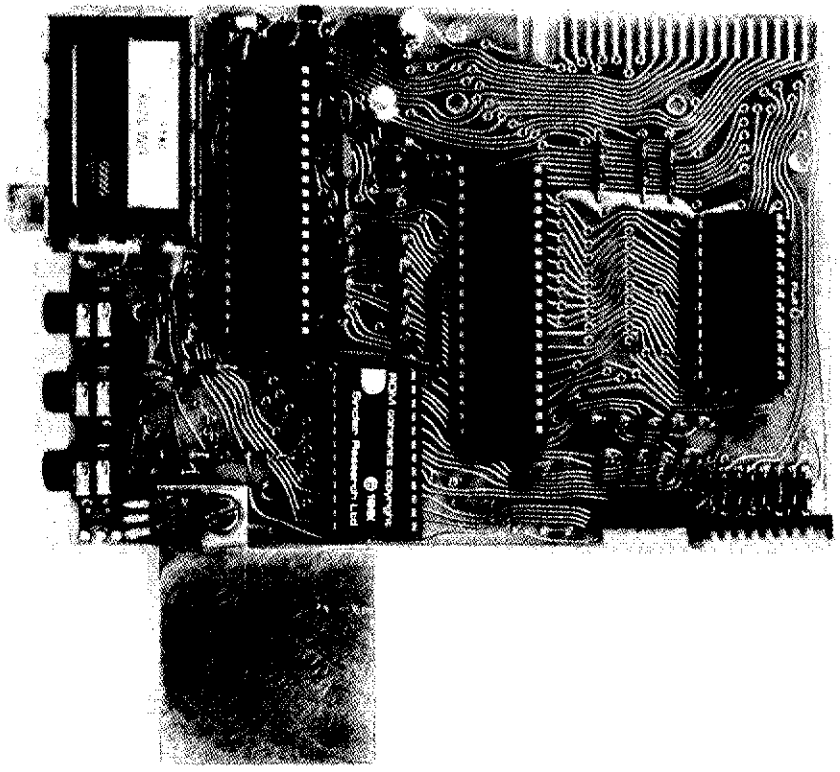
b) depois de uma instrução **LPRINT** que não acabe em vírgula ou ponto e vírgula,

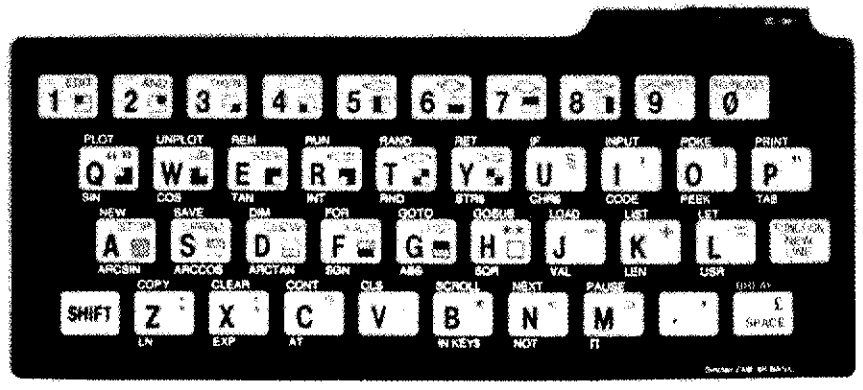
c) quando uma vírgula ou elemento **TAB** requerer nova linha, ou

d) no fim do programa, se faltar imprimir alguma coisa,

e) diz-lhe porque é que o seu programa trabalha assim com **TAB.** Em relação ao **AT**, ignora o número de linha e a posição **LPRINT** (como a posição **PRINT**, mas em relação à Printer e não à TV) é alterada para o número da coluna. Um elemento **AT** não pode mandar uma linha para o printer. (De facto, o número de linha depois de **AT** não é completamente ignorado; tem que ser entre —21 e +21 ou dará erro. Por isso é melhor especificar a linha Ø. O elemento **AT** 21 — N,N na versão final do nosso programa ficaria melhor, contudo menos ilustrativo, se o substituíssemos por **AT** Ø,N).

2. Imprima um gráfico de SIN passando o programa do Capítulo 18 e usando depois **COPY.**





Capítulo 21

Substrings

Dada uma "string", a sua "subtring" consiste nalguns dos seus caracteres consecutivos, tirados em sequência. Assim, "STRING" é uma "subtring" da "MAIOR STRING", mas "B STRING" e "GRANDE REG" não são.
Existe uma notação denominada divisora para descrever substrings, o mesmo se podendo aplicar para expressões arbitrárias string. A forma geral é

expressão string (começo **TO** fim)

ficando, por exemplo,

"ABCDEF" (2 **TO** 5) = "BCDE"

Se omitir o começo, presume-se que é 1; se omitir o fim, presume-se o comprimento da string. Assim

"ABCDEF" (**TO** 5) = "ABCDEF" (1 **TO** 5) = "ABCDE"
"ABCDEF" (2 **TO**) = "ABCDEF" (2 **TO** 6) = "BCDEF"

e

"ABCDEF" (**TO**) = "ABCDEF" (1 **TO** 6) = "ABCDEF"

(pode-se também escrever esta última como "ABCDEF" (), se for necessário).
Há uma forma ligeiramente diferente que omite **TO** e tem apenas um número:

"ABCDEF" (3) = "ABCDEF" (3 **TO** 3) = "C"

Apesar de ambas as partes, começo e fim, deverem referir-se a partes existentes da string, esta regra é ultrapassada por outra: se o começo é mais que o fim, o resultado é a string vazia. Assim

"ABCDEF" (5 **TO** 7)

dá o erro 3 (erro subscrito) porque a string só contém 6 caracteres, e 7 é demais, mas

"ABCDEF" (8 **TO** 7) = ""

e

"ABCDEF" (1 **TO** 0) = ""

O começo e o fim não devem ser negativos, ou obterá o erro B.
O próximo programa dá B$ igual a A$, omitindo os espaços finais.

```
10 INPUT A$
20 FOR N=LEN A$ TO 1 STEP -1
30 IF A$(N)<>" " THEN GOTO 50
40 NEXT N
50 LET B$=A$( TO N)
60 PRINT """";A$;"""","""";B$;""""
70 GOTO 10
```

Note-se que se A$ fosse inteiramente espaçado, na linha 50 teríamos N = 0 e A$ (TO N) = A$ (1 TO 0) = " ".

Para variáveis string, podemos não só extrair substrings, mas também atribuí-las. Escreva, por exemplo

```
LET A$ = "ANA AMA UM PATO"
```

e depois

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

e

```
PRINT A$
```

Repare como a substring A$ (5 TO 8) tem apenas quatro caracteres só as quatro primeiras estrelas foram usadas. Isto é uma característica da atribuição de substring: a substring tem que ter depois exactamente o mesmo comprimento que tinha anteriormente.

Para se certificar se isto acontece, a string que se atribuiu é cortada à direita se for demasiadamente longa, ou preenchida com espaços se for demasiadamente curta — chama-se descrição ou atribuição Procrustean segundo o estalajadeiro Procrustes que, para se certificar que os seus hóspedes cabiam na cama, esticava-os ou cortava-lhes os pés.

Agora tente

```
LET A$() = "COR BLIMEY"
```

e

```
PRINT A$;" "
```

verá que voltou a acontecer o mesmo (desta vez com espaços entre eles) porque A$() conta como substring.

```
LET A$ = "COR BLIMEY"
```

servirá perfeitamente.

A divisora pode ser considerada como tendo prioridade 2, o que dá por exemplo,

```
LEN "ABCDEF" (2 TO 5) = LEN ("ABCDEF" (2 TO 5)) =4
```


As expressões string complicadas precisam de ser postas entre parêntesis ou serão divididas. Por exemplo,

```
"ABC"+"DEF" (1 TO 2)="ABCDE"
("ABC"+"DEF") (1 TO 2)="AB"
```

Sumário

Dividindo, com **TO.** Repare que esta notação não é standardizada).

Exercícios

1. Alguns BASICs (não o BASIC ZX81) tem funções denominadas LEFT$, RIGHT$, MID$ e TL$.

LEFT$ (A$,N) dá a substring de A$, que consiste nos primeiros N caracteres.

RIGHT$ (A$,N dá a substring de A$ que consiste nos caracteres a partir de N.

MID$ (A$,N1,N2) dá a substring de A$ que consiste nos N2 caracteres que começam em N1.

TL$ (A$) dá a substring de A$ que consiste em todos os seus caracteres excepto o primeiro.

Como os escreveria em BASIC ZX81? As respostas funcionariam com strings de comprimento 0 ou 1?

2. Tente esta sequência de comandos:

```
LET A$="X"+"Y"
LET A$(2)=CHR$ 11
LET A$(4)=CHR$ 11     (o caracter de aspa string)
PRINT A$
```

A$ é agora uma string com aspas string! Nada o poderá impedir de fazer isto se for bastante persistente, mas se V. tivesse escrito

```
LET A$="X"+"Y"
```

o lado direito do sinal de igual teria sido tratado como expressão, dando a A$ o valor "XY".

Agora escreva

```
LET B$="X""+""Y"
```

Verificará que, embora depois de escritas A$ e B$ pareçam o mesmo, não são iguais — faça

```
PRINT A$=B$
```

Enquanto B$ tem apenas caracteres metidos entre imagens aspas (com código 192), A$ tem os verdadeiros caracteres de aspas string (com código 11).

3. Passe este programa:

```
10 LET A$="LEN ""ABCD"""
100 PRINT A$;" = ";VAL A$
```

Isto não funciona porque **VAL** não trabalha a imagem aspas **" "** como aspas string.

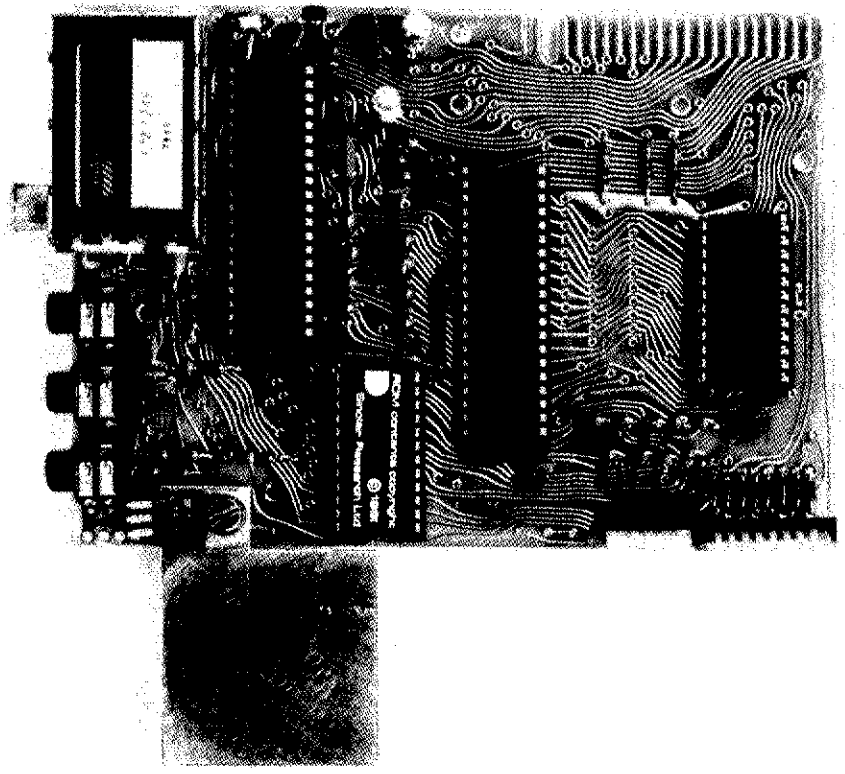
Meta algumas linhas extras entre 10 e 100 para substituir as imagens aspas em A$ por aspas string (a que se deve chamar **CHR$** 11), e tente novamente.

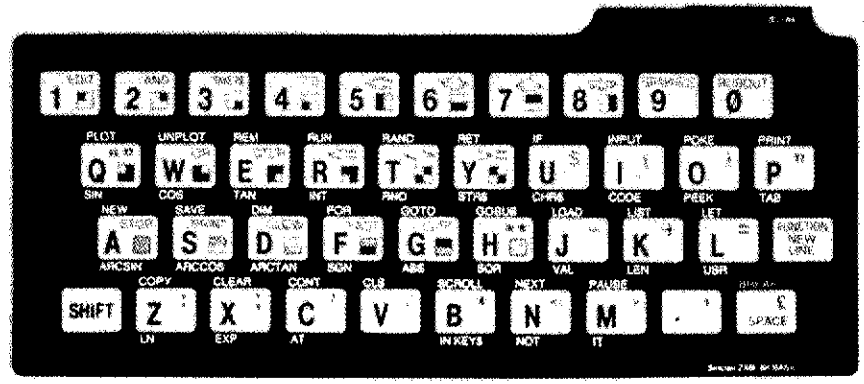
Faça a mesma modificação para o programa do Capítulo 9, exercício 3, e tente fazê-lo com isto.

4. Esta subrotina apaga tudo o que se passa com a string "CARTHAGO" de A$.

```
1000 FOR N=1 TO LEN A$-7
1020 IF A$(N TO N+7)="CARTHAGO" THEN LET
A$(N TO N+7)="********"
1030 NEXT N
1040 RETURN
```

Escreva um programa que dê vários A$ (p.e., "DELENDA EST CARTHAGO.") e aplique a subrotina.





Capítulo 22

Arrays (Matrizes)

Suponha que tem uma lista de números, por exemplo, o número de cobradores de impostos de rendimento que morreram por mês durante o corrente ano financeiro. Para os armazenar no computador, poderia estabelecer só uma variável para cada mês, mas tornar-se-ia antiquado. Poderia decidir chamar as variáveis, LÁ VAI 1, LÁ VÃO 2, e assim sucessivamente até LÁ VÃO 12, mas o programa ficaria muito longo e chato de escrever.

Seria bom que pudesse escrever:

```
 5 REM ESTE PROGRAMA NÃO FUNCIONARÁ
10 FOR N = 1 TO 12
20 PRINT LÁ VÃO N
30 NEXT N
```

Mas não pode

Contudo, há um mecanismo a que pode aplicar esta ideia, utilizando arrays (matrizes). Uma matriz é um conjunto de variáveis ou elementos, todos com o mesmo nome, diferençáveis por um só número (o subscrito) escrito entre parêntesis depois do nome. No nosso exemplo, o nome seria L (como as variáveis de controle dos ciclos **FOR--NEXT,** o nome de um array deve ser uma única letra), e as doze variáveis seriam L(1), L(2), etc., até L(12).

Chamamos variáveis subscritas aos elementos de um array, ao contrário das variáveis simples que já conhece.

Antes de usar um array, deve reservar espaço para ele no computador, o que se faz usando uma instrução **DIM** (para dimensão)

```
DIM L(12)
```

estabelece um array denominado L com dimensão 12 (i.e., há duas variáveis subscritas L(1), . . ., L(12)) e, inicializa os 12 valores a 0. Também apaga todos os arrays L que existiram anteriormente. (Mas não uma variável simples. Um array e uma variável numérica simples podem coexistir com o mesmo nome, e não se confundirão porque a variável do array tem um subscrito).

O subscrito pode ser uma expressão numérica arbitrária, pelo que agora pode escrever

```
10 FOR N = 1 TO 12
20 PRINT A(N)
30 NEXT N
```

Pode também estabelecer arrays com mais de uma dimensão. Num array bi-dimensional precisa de dois números para especificar um dos elementos — tal como os números de linha e coluna para especificar a posição de um caracter no écran de TV —

pelo que faz a forma de uma tabela. Alternativamente, se imaginar os números de linha e coluna (duas dimensões) como se se referissem a uma página impressa, pode ter uma dimensão extra para os números das páginas. É evidente que estamos a referir-nos a arrays numéricos; assim, os elementos seriam números e não caracteres impressos como num livro. Imagine que os elementos de um array tri-dimensional C são especificados por C (número de página, número de linha, número de coluna).

Para se estabelecer, por ex., um array B bi-dimensional com as dimensões 3 e 6, usa-se uma instrução **DIM**

DIM B(3,6)

O que lhe dá 3*6 = 18 variáveis subscritas

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

O mesmo princípio rege qualquer número de dimensões.

Embora se possa ter um número e um array com o mesmo nome, não se podem ter dois arrays com o mesmo nome, embora tenham diferentes números de dimensões.

Há também arrays string. As strings num array diferem das strings simples porque têm o comprimento fixo e a sua atribuição é sempre Procrusteana — outra maneira de as imaginar é como arrays (com uma dimensão extra) de caracteres únicos. O nome de um array string é uma letra seguida de $, e um array string e uma variável string simples não podem ter o mesmo nome (ao contrário do caso dos números).

Suponha que queria um array A$ de cinco strings. Tem que estabelecer o tamanho das strings — supondo que bastam ter 10 caracteres cada. Então

DIM A$ (5,10) (introduza isto)

Isto estabelece um array de caracteres 5*10, mas cada linha pode também ser uma string:

A$(1) =	A$(1,1)	A$(1,2)	. . .	A$(1,10)
A$(2) =	A$(2,1)	A$(2,2)	. . .	A$(2,10)
:	:	:	:	:
A$(5) =	A$(5,1)	A$(5,2)	. . .	A$(5,10)

Se der o mesmo número de subscritos (dois, neste caso), tantas quantas as dimensões na instrução **DIM**, obtém um só caracter; mas se omitir o último, obtém uma string de comprimento fixo. Assim, por exemplo, A$(2,7) é o 7.º caracter da string A$(2); utilizando a notação divisora, poderia também escrever isto como A$(2)(7). Agora escreva

LET A$(2) = "1234567890"

e

PRINT A$(2),A$(2,7)

obtém

1234567890 7

O último subscrito (o que pode omitir), pode também dividi-lo,

A$(2,4 **TO** 8) = A$ (2) (4 **TO** 8) = *"45678"*

Note:

Num array string, todas as strings têm o mesmo comprimento fixo.

A instrução **DIM** tem um número extra (o último) para especificar este comprimento.

Quando escreve uma variável subscrita para um array string, pode-lhe acrescentar um número extra, ou divisor, para corresponder ao número extra da instrução **DIM.**

Sumário

Arrays (o modo como o ZX81 trabalha arrays string não é completamente standardizados).

Instruções: **DIM**

Exercícios

1. Estabeleça um array M$ de doze strings em que M$(N) seja o nome do mês N. (Sugestão: a instrução **DIM** será DIM M$(12,9). Experimente isto escrevendo todo M$(N) (use um ciclo). Escreva

PRINT "AGORA É O MÊS DE";M$(5);"EM QUE OS RAPAZES BRINCAM FELIZES"

O que pode fazer com todos os espaços?

2. Pode ter arrays string sem dimensões. Escreva

DIM A$(10)

e verá que A$ age apenas como uma variável string, com a única diferença que tem comprimento 10, e a sua atribuição é sempre Proscrusteana.

3. READ, DATA, e RESTORE; são necessários?

A maior parte das linguagens BASIC (embora não BASIC ZX81) têm três instruções denominadas READ, DATA e RESTORE.

Uma instrução DATA é uma lista de expressões, e todas as instruções DATA dum programa consistem numa longa lista de expressões, a lista DATA.

As instruções READ usam-se para atribuir essas expressões, uma a uma, a variáveis:

```
READ X
```

por exemplo, atribui a expressão corrente na lista DATA à variável X, e faz avançar a expressão seguinte para a instrução READ a seguir.

(RESTORE, faz retroceder para o princípio da lista DATA).

Teoricamente, podem-se sempre substituir instruções READ e DATA por instruções **LET**: contudo, uma das suas grandes utilizações é na inicialização de arrays, como neste programa:

```
 5 REM ESTE PROGRAMA NÃO FUNCIONA NO BASIC ZX81
10 DIM M$(12,3)
20 FOR N = 1 TO 12
30 READ M$(N)
40 NEXT N
50 DATA "JAN","FEV","MAR","ABR"
60 DATA "MAIO","JUN","JUL","AGO"
70 DATA "SET","OUT","NOVE","DEZ"
```

Se quisesse passar este programa apenas uma vez, deverá substituir a linha 30 por uma instrução **INPUT**, assim:

```
10 DIM M$(12,3)
20 FOR N = 1 TO 12
30 INPUT M$(N)
40 NEXT N
```

e não escreverá mais nada. Contudo, se quiser gravar o programa, decerto não desejaria escrever os meses sempre que o passa.

Sugerimos este método:

a) Inicialize o array usando um programa como o acima.

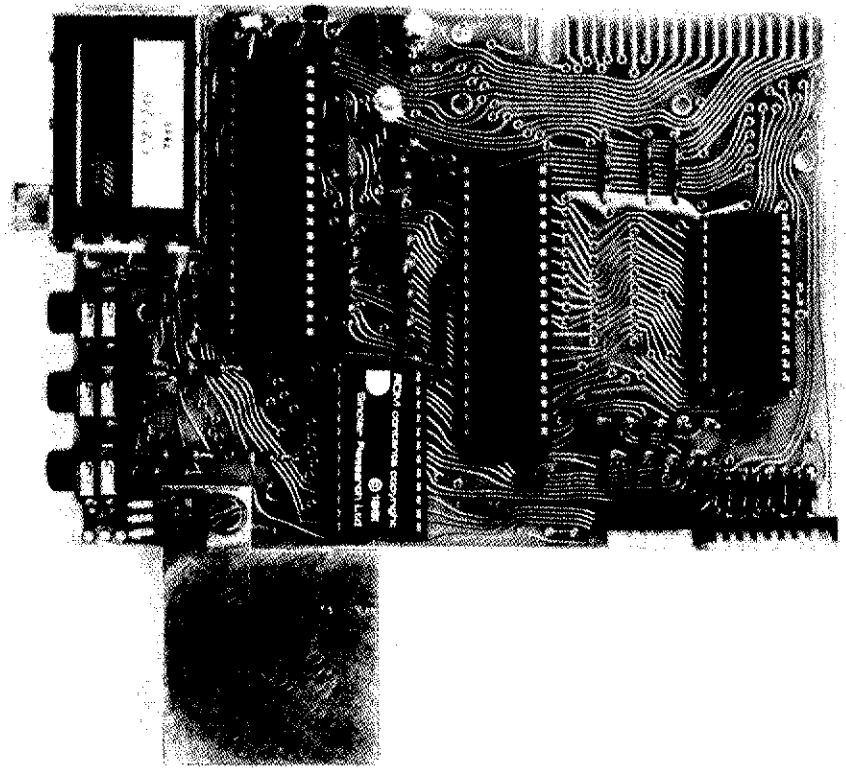
b) Elimine este programa de inicialização. (Não utilize **NEW**, porque quer manter o array).

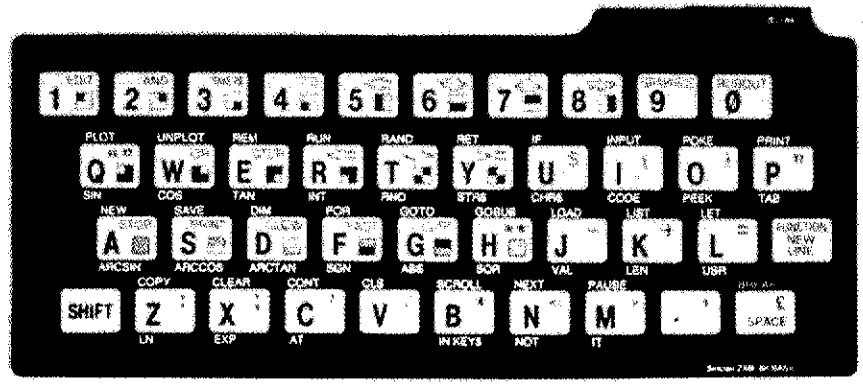
c) Escreva o resto do programa e grave-o. Isto também gravará as variáveis, bem como o array).

d) Quando voltar a passar o programa, passará também o array.

e) Quando executar o programa, não use **RUN**, que apaga as variáveis. Use antes **GOTO** com um número de linha.

Em alternativa, poderá usar **LOAD** e executar a técnica do Capítulo 16, exercício 3. Na alínea c) acima usará uma instrução **SAVE** no programa, e a alínea e) será completamente omitida.





Capítulo 23

Quando o computador está cheio

O ZX81 tem uma quantidade limitada de armazenagem, e não é difícil de esgotá-la. Quando isso acontece, aparece um relatório de erro 4, mas também podem acontecer outras coisas, algumas bastante estranhas. O comportamento exacto depende do facto de termos um quadro de expansão de memória incorporado; portanto, suponhamos que não o tem. Se o tiver tire-o (desligando primeiro o computador).

O ficheiro de projecção, que é a área dentro do computador onde ele armazena a figura TV, foi concebida engenhosamente de modo a guardar espaço só para o que foi escrito até aí: uma linha da projecção consiste em até 32 caracteres e depois um caracter **NEWLINE.** Isto significa que pode ficar sem memória ao escrever qualquer coisa, e o sítio mais óbvio é quando se faz uma listagem. Escreva

```
NEW
DIM A (15Ø)
1Ø FOR N = 1 TO 15
2Ø PRINT N
```

Aqui aparece a primeira surpresa: a linha 1Ø desaparece da listagem. A listagem é obrigada a incluir a linha corrente, a 2Ø, pelo que não haverá espaço para ambas as linhas.

Agora escreva

```
3Ø NEXT N
```

Há novamente apenas espaço para a linha 3Ø na listagem. Escreva

```
4Ø REM X        (sem NEWLINE)
```

e verá que a linha 3Ø desaparece e a linha 4Ø passa para o princípio do écran. Não foi introduzida no programa — tem ainda o cursor **L** e pode movê-lo. O que vimos foi um mecanismo obscuro que dá à metade de baixo das 24 linhas do écran prioridade sobre a metade de cima. Ora escreva:

```
XXXXXX        (sem NEWLINE)
```

e o cursor desaparecerá, pois não há espaço para o projectar. Escreva outro X, sem **NEWLINE**, e desaparecerá um dos Xs. Prima **NEWLINE**. Desaparecerá tudo, mas o programa continua no computador, o que pode confirmar apagando a linha 1Ø e usando ◇ e △ . Escreva

```
1Ø FOR N = 1 TO 15
```

novamente — saltará para o princípio do écran, como aconteceu com a linha 4Ø. Mas quando prime **NEWLINE**, não será introduzido, embora não haja nenhuma mensagem

de erro ou indicador **S** a indicar-lhe que algo está errado. Este é o resultado de não haver espaço para verificar a sintaxe de uma linha o que normalmente acontece nas linhas que têm números (diferentes do número de linha no princípio).

A única solução é arranjar espaço, mas primeiro apague a linha **1Ø** que não quer entrar. Prima **EDIT**: o écran ficará vazio, pois não há espaço para baixar a linha.

Quando **EDIT** não funciona, pode-se por vezes arranjar espaço escrevendo um número de espaços até o cursor aparecer no écran.

Prima **NEWLINE**, e voltará a obter parte da listagem. Apaque agora a linha **4Ø** (que não queria), escrevendo

4Ø (e **NEWLINE**)

Tente agora voltar a escrever na linha **1Ø** — e continua a não funcionar. Apague-a novamente. Precisa ainda de encontrar mais espaço. Note que a razão porque a linha **1Ø** foi rejeitada foi possivelmente porque não havia espaço para verificar a sintaxe dos dois números, 1 e 15: assim, se apagar a linha **2Ø** do programa, poderá ter espaço para introduzir a linha **1Ø**, e ainda para voltar depois a instroduzir a linha **2Ø** (que não tem nenhum número). Tente fazer isso. Escreva

```
2Ø
1Ø FOR N = 1 TO 15
2Ø PRINT N
```

e o programa será devidamente introduzido.

Escreva

GOTO 1Ø

e verá mais uma vez que esta linha é rejeitada porque a sua sintaxe não pode ser verificada; contudo, se a apagar e escrever

RUN

funcionará. (**RUN** apaga o array, dando muito espaço).

Agora escreva o mesmo desde **NEW** até à linha 3Ø, e depois

4Ø REM XXXXXXXXXXXX

(12 Xs), que acabará parecendo **4Ø** RE. Quando prime **NEWLINE**, a listagem consistirá apenas na linha 3Ø, pelo que a linha **4Ø** se perdeu completamente. E isto apenas porque era demasiado longa para caber no programa. O efeito é um pouco pior quando a linha é uma versão prolongada de uma linha que já estava no programa, pelo que ambas as linhas desaparecerão do mesmo, tanto a que lá estava bem como a nova linha que iria substituir.

A melhor solução é comprar um módulo RAM, que se acupula na parte de trás do computador. O módulo Sinclair 16K RAM aumenta de dezasseis vezes a memória central do computador.

Se tiver o módulo 3K RAM do ZX80, não funcionará no ZX81.

O comportamento do módulo RAM é bastante diferente, pois o arquivo de projecção é preenchido com espaços para dar linhas de 32 caracteres (repare que **SCROLL** perturba isto — ver Capítulo 27). Agora o computador não ficará sem memória ao imprimir ou fazer listagens, e as listagens não serão tão curtas nem saltarão de linha para linha; mas verá que as linhas ficam coladas ou perdem-se, e a única solução será novamente procurar espaço que esteja mal aproveitado.

Se tiver um quadro de expansão de memória, ligue-o e comece a ler este capítulo escrevendo

DIM A(3Ø69)

para substituir **DIM** A(15Ø).

Resumindo: a moral da história é evitar que se fique com um computador encravado. Contudo, as coisas nem sempre são tão más como parecem

1. Se a listagem começar a encurtar ou se as coisas começarem a saltar de um lado para o outro, é porque já há pouca memória disponível.

2. Se **NEWLINE** parece não surtir efeito no fim de uma linha, é porque possivelmente não há espaço para tratar um número. Apague a linha usando **EDIT-NEWLINE** ou **RUBOUT.**

3. **NEWLINE** pode fazer perder uma linha completa.

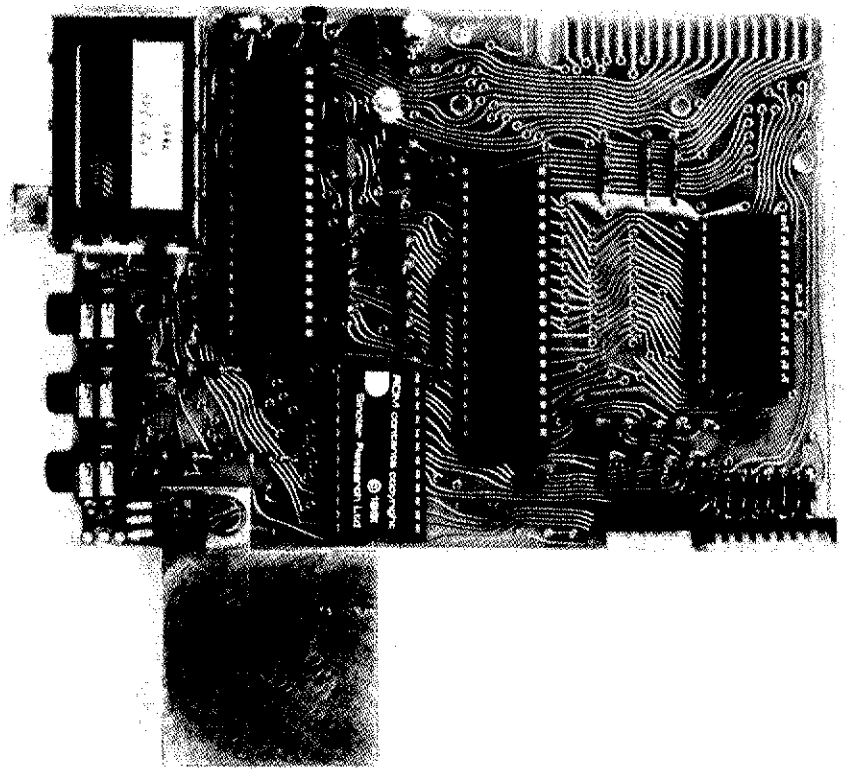
A solução é igual para todos estes "caprichos". Não entre em pânico, procure espaços desnecessários.

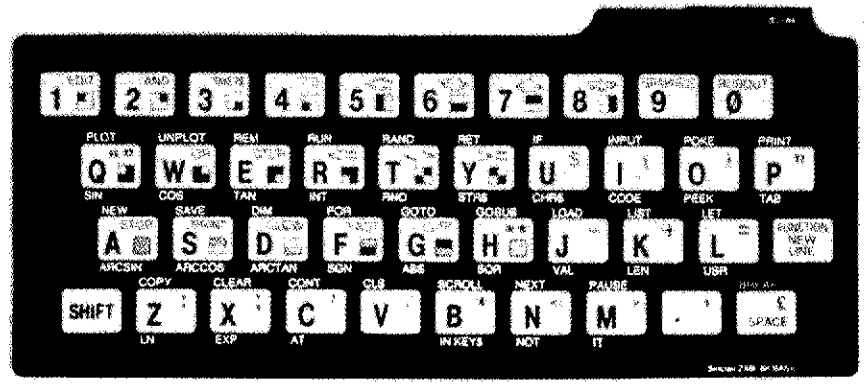
A primeira coisa a considerar é **CLEAR**. Se tem algumas variáveis e não se importa de as perder, é o que há a fazer.

Se isto falhar, procure instruções que não sejam necessárias no programa, tais como as instruções **REM**, e apague-as.

Sumário

Quando a memória está completamente cheia, podem acontecer coisas bastante excêntricas, mas geralmente todas têm solução.





Capítulo 24

Contando pelos dedos

O próximo capítulo explora o interior do computador, mas antes de entrarmos nele seria melhor descrevermos o modo de contar dos computadores: eles fazem-no usando o sistema binário, o que significa que apenas têm polegares.

A maior parte das línguas europeias contam usando um padrão de dez mais ou menos regular — o português por exemplo, apesar de começar irregularmente, acaba por estabelecer-se em grupos regulares:

vinte,	vinte e um,	vinte e dois,	..., vinte e nove
trinta,	trinta e um,	trinta e dois,	..., trinta e nove
quarenta,	quarenta e um,	quarenta e dois,	..., quarenta e nove

e assim sucessivamente. A numeração é ainda mais sistemática. Contudo, a única razão para utilizarmos dez, é pelo simples facto de termos dez dedos.

Suponha que os Marcianos têm mais três dedos que nós em cada mão (até que ponto lhe podemos chamar dedos). Assim, em vez de usarem o sistema decimal, de base dez, usam um sistema hexadecimal (ou hexa, abreviando), baseado em dezasseis. Eles precisam de seis dígitos extra, somando aos dez que utilizamos e escrevem-nos como A,B,C,D,E e F. E a seguir a F? Tal como nós, com dez dedos, escrevemos 10 para dez, eles com dezasseis dedos, escrevem 10 para dezasseis. O seu sistema numérico começa em

Hexa	*Português*
0	zero
1	um
2	dois
:	:
:	:
9	nove

tal como o nosso sistema, mas depois continua

A	dez
B	onze
C	doze
D	treze
E	catorze
F	quinze
10	dezasseis
11	dezassete
:	:
:	:
19	vinte e cinco
1A	vinte e seis
1B	vinte e sete
:	:

Hexa	*Português*
1F	trinta e um
20	trinta e dois
21	trinta e três
:	:
:	:
9E	cento e cinquenta e oito
9F	cento e cinquenta e nove
A0	cento e sessenta
A1	cento e sessenta e um
:	:
:	:
FE	duzentos e cinquenta e quatro
FF	duzentos e cinquenta e cinco
100	duzentos e cinquenta e seis

Se estiver a usar a notação hexadecimal e quiser explicá-lo, escreva "h" no fim do número, que significa hexa. Por exemplo, para cento e cinquenta e oito, escreva '9Eh', que significa 'nove e hexa'.

Poder-se-á interrogar o que é que isto tem a haver com computadores. De facto, os computadores comportam-se como se tivessem apenas dois dígitos, representados por uma baixa voltagem, ou (0), e por alta voltagem (1). É aquilo a que se chama o sistema binário, e os dois dígitos binários denominam-se bits: assim, um bit é 0 ou 1.

Nos vários sistemas, a contagem começa em

Português	*Decimal*	*Hexadecimal*	*Binário*
zero	0	0	0 ou 0000
um	1	1	1 ou 0001
dois	2	2	10 ou 0010
três	3	3	11 ou 0011
quatro	4	4	100 ou 0100
cinco	5	5	101 ou 0101
seis	6	6	110 ou 0110
sete	7	7	111 ou 0111
oito	8	8	1000
nove	9	9	1001
dez	10	A	1010
onze	11	B	1011
doze	12	C	1100
treze	13	D	1101
catorze	14	E	1110
quinze	15	F	1111
dezasseis	16	10	10000

O que é importante é que dezasseis é igual a dois elevado à potência quatro, o que torna fácil a conversão do sistema hexadecimal no binário.

Para converter do sistema hexadecimal ao sistema binário, muda-se cada dígito hexa em quatro bits, usando a tabela acima.

Para converter do sistema binário ao sistema hexadecimal, divide-se o número binário em grupos de quatro bits, começando pela direita, mudando depois cada grupo para o dígito hexa correspondente.

É por esta razão que, apesar de regra geral os computadores usarem estritamente o sistema binário, os seres humanos escrevem normalmente os números armazenados no computador, usando a notação hexadecimal.

Os bits armazenados no computador estão agrupados grande parte, em conjuntos de oito, ou bytes. Um só byte pode representar qualquer número de zero a duzentos e cinquenta e cinco (11111111 em binário ou FF em hexadecimal), ou alternativamente qualquer caracter o conjunto de caracteres do ZX81. O seu valor pode ser escrito com dois dígitos hexa.

Podem-se agrupar dois bytes, formando aquilo a que tecnicamente se denomina uma palavra. Esta pode ser escrita utilizando dezasseis bites ou quatro dígitos hexa, e representa um número de 0 a (em decimal) $2^{16}-1 = 65535$.

Um byte é sempre oito bites, mas as palavras variam de um computador para outro.

Sumário

Sistemas decimal, hexadecimal e binário.

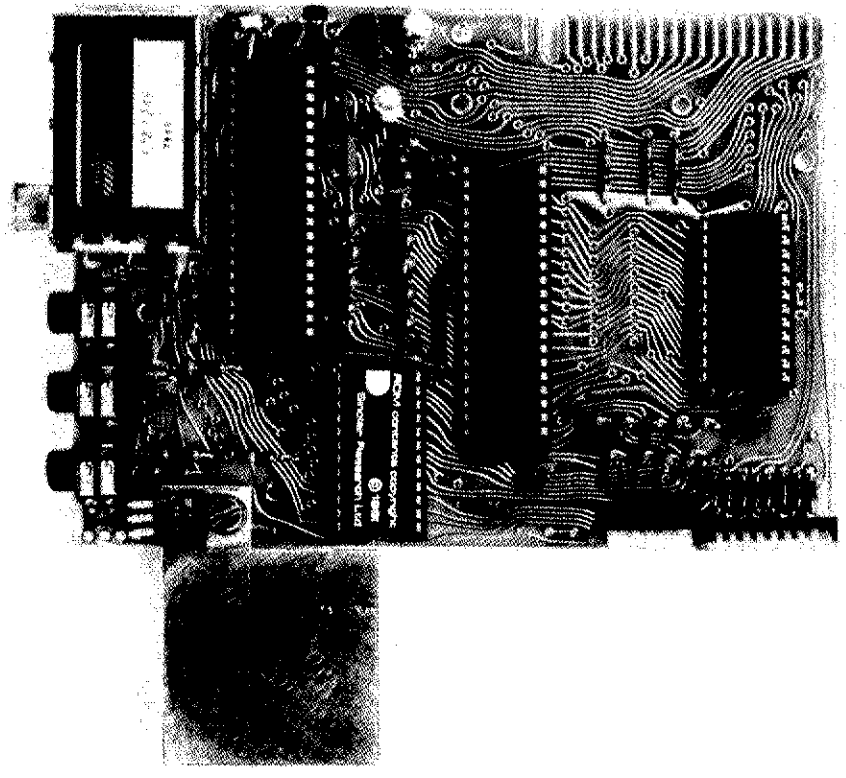
Bites, bytes (não os confunda) e palavras.

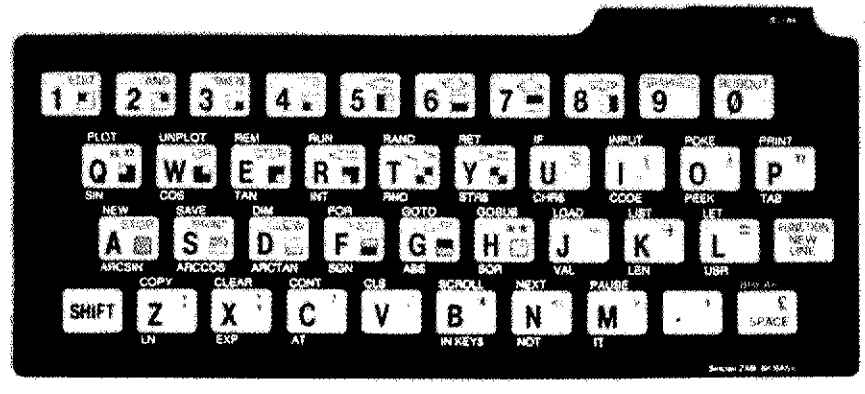
Exercícios

1. A unidade corrente de Marte é a libra, que está dividida em dezasseis onças. Como poderíamos converter libras e onças e voltar novamente para libras e onças.
   a) quando todos os números estiverem escritos em decimal?
   b) quando todos os números estiverem escritos em hexadecimal?

2. Como converter um número decimal em hexadecimal? (Sugestão: exercício 1).
   Escreva programas no ZX81, de modo a converter valores numéricos em strings dando-lhe a representação hexadecimal, e vice versa. (É o que **STR$** e **VAL** fazem com representações decimais).

3. Suponha que os habitantes de Vénus têm um total de oito dedos, não têm polegares; como se poderia adaptar esta contagem octagonal (base oito) aos computadores?





Capítulo 25

Como funciona o computador

A ilustração na página seguinte mostra o interior do ZX81 (mas não o desmonte, pois pode ser difícil voltar a montá-lo).

Como pode verificar, tudo tem uma abreviatura de três letras.

As peças de plástico com muitas pernas são os maravilhosos circuitos integrados de silício, que nos trouxeram não só os relógios digitais, mas também o ZX81. Dentro de cada peça de plástico existe uma peça de silício com aproximadamente a mesma dimensão do cursor K, ligados através de fios às pernas de metal.

O "cérebro" que está por detrás da operação é o circuito integrado processador, geralmente denominados CPU (Central Processing Unit = Unidade Central de Processamento). Esta última é um processador Z80 (ou melhor, um Z80A, que é mais rápido).

O processador controla todo o computador, faz a aritmética, considera a tecla que primiu, decide que resultado vai dar e coordena a imagem do televisor. Contudo, com toda a sua inteligência, não pode fazer tudo sozinho. Separadamente nada sabe de BASIC, aritmética em vírgula flutuante ou televisores, e tem que obter todas as instruções dum outro circuito, o ROM (Read Only Memory = Memória Pré-programada). O ROM é apenas uma longa lista de instruções que prefazem um programa de computador completo, dizendo ao processador o que fazer em todas as circunstâncias previsíveis. Este programa não é escrito em BASIC, mas naquilo a que se chama código máquina Z80, e toma a forma de uma longa sequência de bytes. (Lembre-se que um byte é apenas um número entre 0 e 255). Cada byte tem um endereço que mostra em que parte do ROM está; o primeiro tem o endereço 0, o segundo tem o endereço 1, e assim sucessivamente até 8191. Tudo junto, prefaz 8192 = 8*1024 bytes, que é a razão porque o BASIC ZX81 também se chama por vezes BASIC 8K. 1K é 1024, ou 2^{10}.

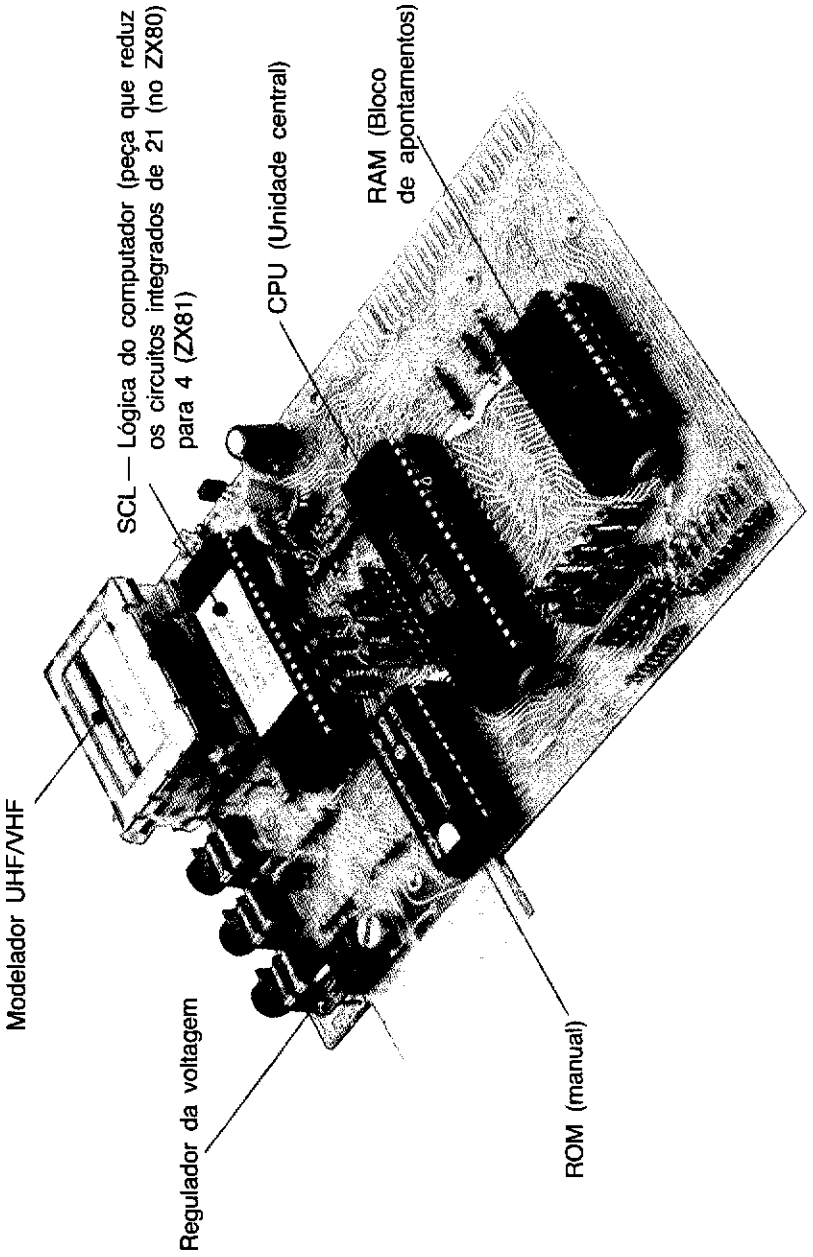
Apesar de haverem peças semelhantes em muitas máquinas diferentes, esta sequência especial de instruções é única do ZX81 e foi escrita especialmente para ele por uma pequena firma de matemáticos de Cambridge.

Pode verificar qual o byte que está em determinado endereço usando a função **PEEK**. Este programa, por exemplo, escreve os primeiros 21 bytes do ROM (e seus endereços).

```
10 PRINT "ENDEREÇO";TAB 8;"BYTE"
20 FOR A = 0 TO 20
30 PRINT A;TAB 8;PEEK A
40 NEXT A
```

O circuito seguinte a considerar é a memória, ou circuito RAM (Random Access Memory = Memória Viva). É onde o processador armazena a informação que quer guardar — o seu programa BASIC, variáveis, a imagem do TV, e diversas notas (o sistema de variáveis) para saber em que estado está o computador.

Como o ROM, a memória está ordenada em bytes, cada um com um endereço:



os endereços ordenam-se de 16384 a 17407 (ou possivelmente até 32767 se tiver um módulo 16K RAM). Tal como para o ROM, pode achar os valores destes bytes usando **PEEK**, mas a grande diferença deste em relação ao ROM é que os pode sempre modificar. (Claro que o ROM é fixo e inalterável).

Escreva

POKE 17300,57

Isto faz com que o byte no endereço 17300 tenha o valor de 57. Se escrever agora

PRINT PEEK 17300

obterá novamente o número 57. (Tente alterar o registo para outros valores, para provar que não há batota).

Repare que o endereço tem de estar entre 0 e 65535; e a maior parte destes números referem-se a bytes no ROM ou a nada, e neste último caso não surtem qualquer efeito. O valor tem de ser entre −255 e +255, e se for negativo dará o resultado somado de 256.

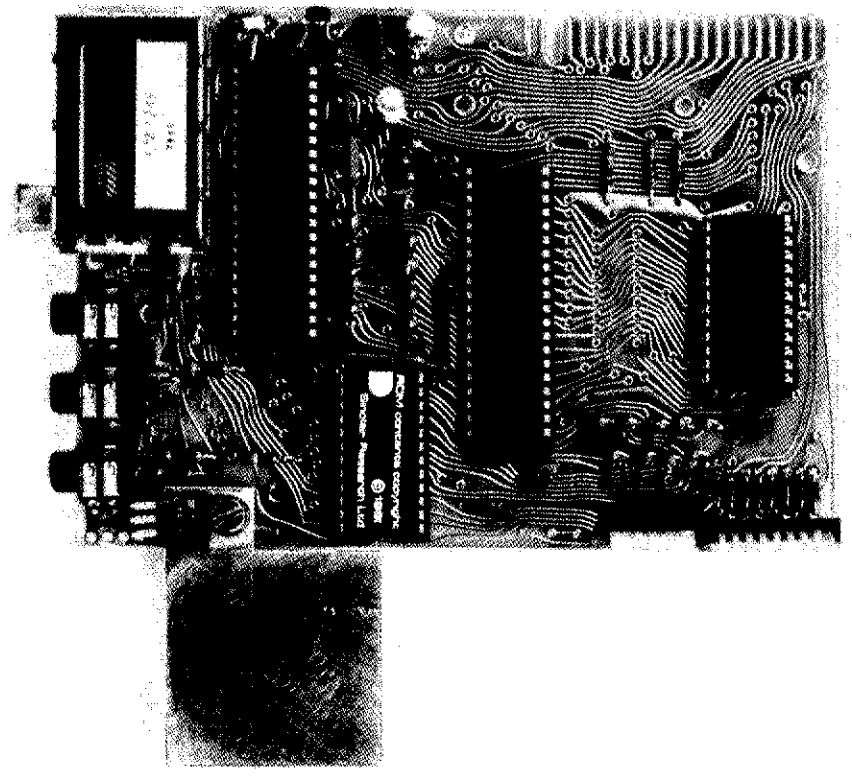
A capacidade de alterar os registos (com POKE) dá-lhe imensos poderes sobre o computador se souber como utilizá-la; contudo, os conhecimentos necessários para isso são muito mais extensos do que aqueles que se poderiam dar num manual deste tipo.

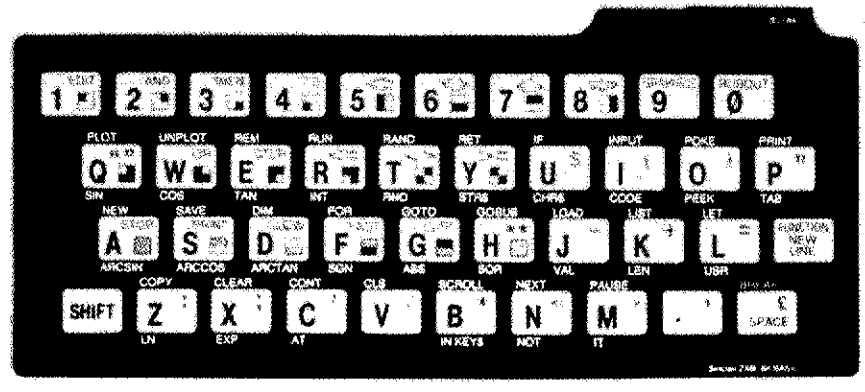
O último grande circuito é o circuito lógico, ou circuito SCL (Sinclair Computer Logic = Lógica do Computador Sinclair). Este foi também especialmente concebido e fabricado para o ZX81, e liga todas as outras peças de forma inteligente, para que eles façam mais do que fariam normalmente.

O modulador transforma a saída da TV do computador numa forma que se adapta ao TV e o regulador transforma os 9 volts da fonte de alimentação, que é pulsada em 5 volts regulados.

Sumário

Circuitos integrados
Instruções: **POKE**
Funções: **PEEK**





Capítulo 26

Linguagem máquina

Este capítulo foi escrito para quem percebe o código (ou linguagem) máquina, o conjunto de instruções utilizado pelo circuito processador Z80. Se não souber mas quiser saber, há livros que tratam este assunto: dois deles, introdutórios, são "Programming the Z8Ø" (Programando o Z8Ø), de Rodnay Zaks, publicado pela Sybex e "Mastering Machine Code on your ZX81" (Dominando a Linguagem Máquina no seu ZX81), de Tony Baker, publicado por Database Consultancy (em breve à venda em Portugal).

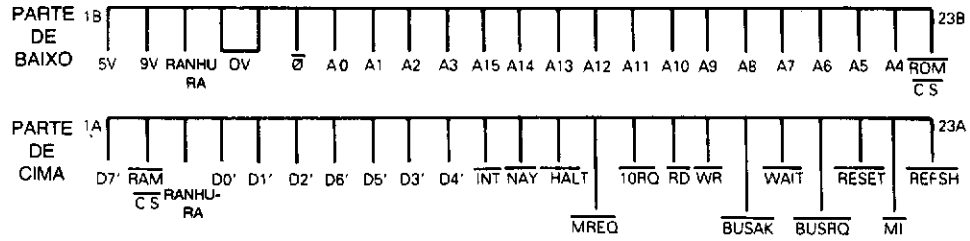
As rotinas em linguagem máquina podem ser executadas a partir de um programa BASIC, usando a função **USR.** O argumento de **USR** é o endereço inicial da rotina, e o seu resultado é um número inteiro de dois bytes, o conteúdo do registo de retorno bc. O endereço de retorno ao BASIC está empilhado da forma usual, pelo que o retorno se faz através de uma instrução RET do Z8Ø.

Há algumas restrições nas rotinas **USR;**

a) - No retorno os registos iy e i têm que ter os valores 4ØØØh e 1Eh.

b) A rotina de "display" usa os registos a', f', ix, iy, e r, pelo que uma rotina **URS** não os deve utilizar se estivéssemos a programar e a projectar. (Nem é seguro ler o par af').

As linhas de alimentação, de controle, dados e endereços estão todas expostas na parte de trás do ZX81, pelo que pode fazer com o ZX81 tudo o que faz com o Z8Ø. Contudo, o hardware do ZX81 pode por vezes enveredar por um caminho próprio, especialmente na programação e projecção. Eis um diagrama das ligações expostas na parte de trás.



Um pedaço de linguagem máquina no meio da memória corre o risco de voltar a ser escrita pelo sistema BASIC. Os lugares mais seguros são:

a) Numa declaração **REM**: escreva uma declaração **REM** com caracteres suficientes para suportarem a linguagem máquina, que depois irá alterar com POKE. Faça isto na primeira linha do programa, ou poderá mover-se. Evite as instruções de paragem, uma vez que estas seriam interpretadas como o fim da declaração **REM.**

b) Numa string: elabore uma string suficientemente longa, e atribua um byte em linguagem máquina a cada caracter. as strings estão sempre sujeitas a moverem-se na memória.

No Apêndice A, o conjunto dos caracteres, encontrará os caracteres e instruções Z8Ø escritas lado a lado e por ordem, o que lhe poderá ser útil quando intruduzir a linguagem.

c) No topo da memória. Quando o ZX81 é ligado, testa a memória que existe e põe a pilha da máquina no topo, de modo a não deixar espaço para as rotinas **USR.** Armazena o endereço do primeiro byte não existente (i.e., 17K, ou 17408, se tiver 1K de memória), numa variável de sistema conhecida como RAMTOP (ver vocábulos), nos dois bytes com endereços 16388 e 16389. **NEW**, por outro lado, não faz um teste completo da memória, mas verifica-a até antes do endereço RAMTOP. Assim se alterar o endereço de um byte existente e o incluir em RAMTOP, para **NEW** toda a memória desse byte interno está fora do sistema BASIC e não lhes toca. Por exemplo, suponha que tem 1K de memória e que acabou de ligar o computador.

PRINT PEEK 16388+256*PEEK 16389

diz-lhe o endereço (17408) do primeiro byte não existente. Suponha agora que tem uma rotina **USR** com 20 bytes de comprimento. Quer mudar RAMTOP para 17388= =236+256*67 (como resolveria isto no computador?), escreva

POKE 16388,236
POKE 16389,67

e depois **NEW**. Os vinte bytes de memória do endereço 17388 até 17407 estão agora à sua disposição para fazer o que quizer. Se voltar a escrever **NEW**, não afectará esses 20 bytes.

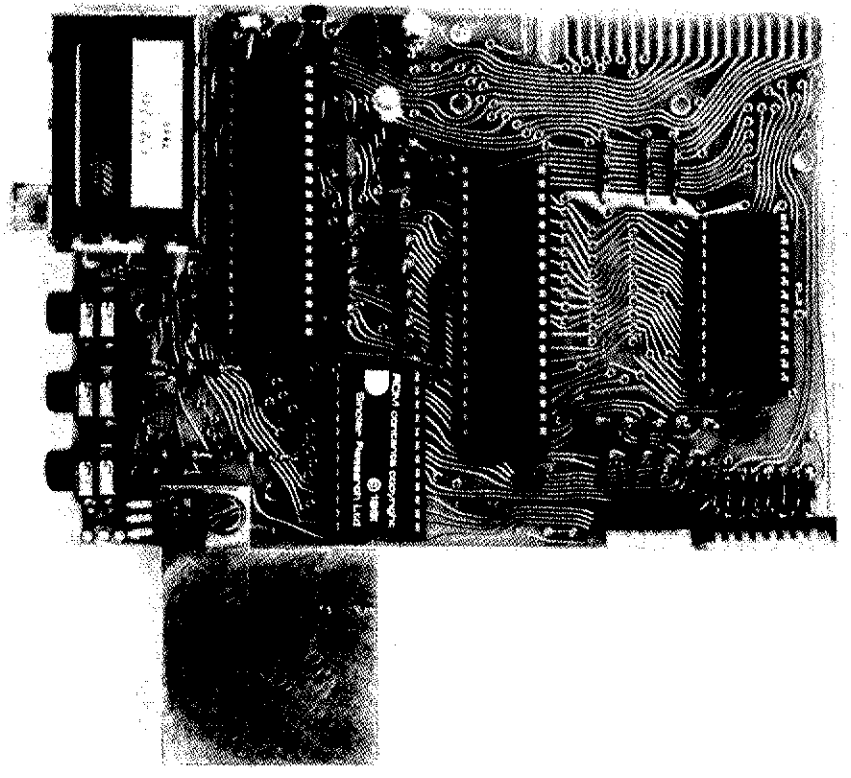
O cimo da memória é um bom sítio para rotinas **USR**, seguro (mesmo com **NEW**) e imóvel. A sua única desvantagem é não ser gravado por **SAVE**.

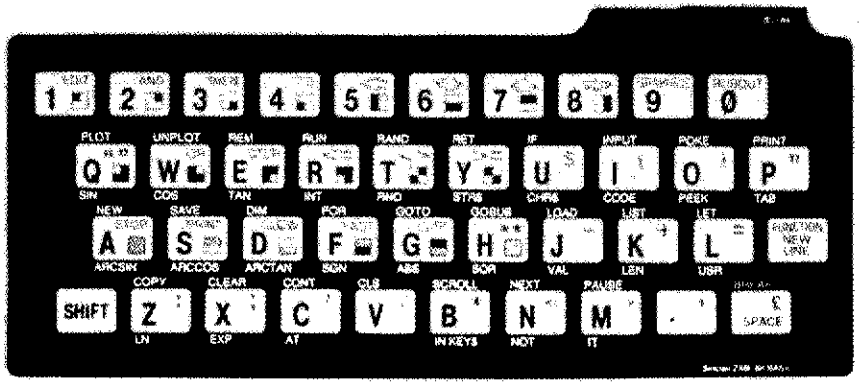
Sumário

Funções: **USR**
Instruções: **NEW**

Exercícios

1. Faça RAMTOP igual a 16700 e depois execute **NEW**. Ficará com uma ideia do que acontece quando a memória se enche.

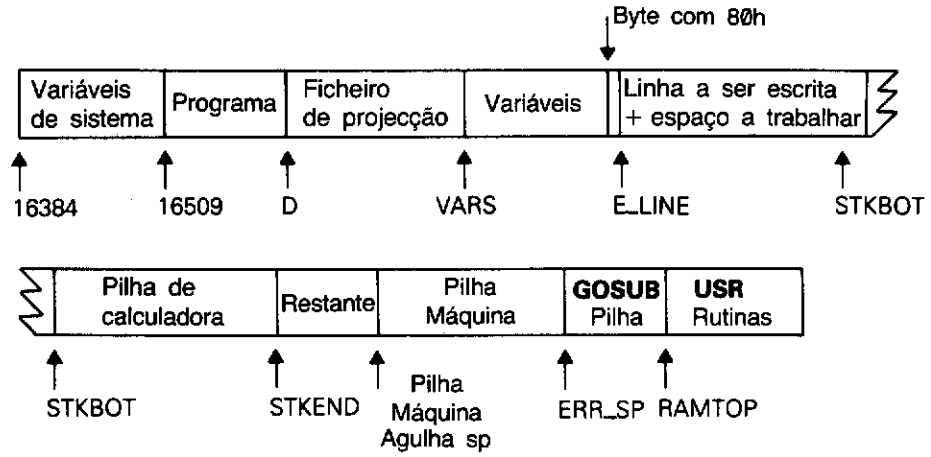




Capítulo 27

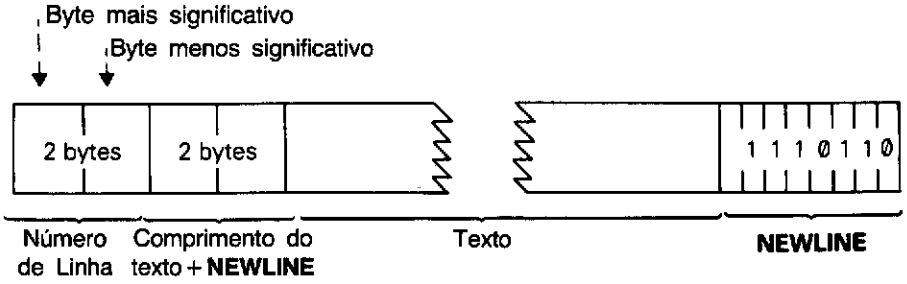
Organização da memória

A memória está dividida em zonas diferentes para armazenar diferentes tipos de informação. As zonas são apenas suficientes para a informação que realmente contém, e se lhe introduzir mais num determinado ponto (por exemplo, juntando uma linha ou variável no programa), é arranjado espaço deslocando tudo, acima desse ponto. Pelo contrário, se apagar a informação, tudo o que está acima dela passa para baixo.



As variáveis de sistema contém vários excertos de informação que dizem ao computador em que estado ele se encontra. Elas estão completamente listadas no próximo capítulo, mas por agora repare que há algumas (denominadas D FILE, VARS, E LINE, etc.) que contêm os endereços dos limites entre as várias zonas de memória. Estas não são variáveis BASIC, e os seus nomes não serão identificados pelo computador.

Cada linha do programa é armazenada como:



Repare que, contrariamente aos outros casos de números de dois bytes no Z80, o número de linha neste caso (e também numa variável de controle **FOR-NEXT**) é arma-

zenado pondo em primeiro lugar o seu byte mais significativo: isto é, pela ordem em que os iria escrever.

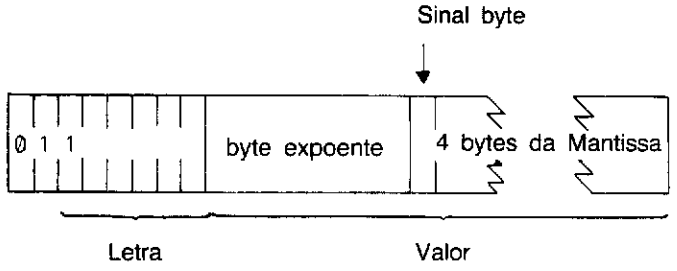
No programa uma constante numérica é seguida da sua forma binária, usando o caracter **CHR$** 126 seguido de cinco bytes para o próprio número.

O ficheiro de projecção é a cópia da memória da imagem do televisor. Começa com um caracter **NEWLINE**, e depois as vinte e quatro linhas do texto, cada uma delas acabando com uma **NEWLINE**. O sistema está concebido de tal forma que uma linha de texto não necessita de ter todos os trinta e dois caracteres: os espaços finais podem ser omitidos. Isto utiliza-se para poupar espaço quando a memória é pequena.

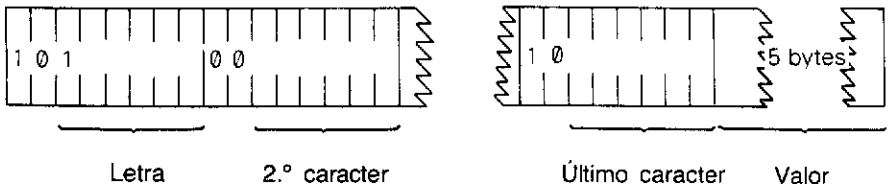
Quando o total da memória (segundo a variável de sistema RAMTOP) é menor que 3 1/4K, então um écran limpo — como se estabeleceu no princípio ou por **CLS** — consiste em apenas vinte e cinco **NEWLINE**s. Quando a memória é maior, então o écran fica limpo, mas com 24*32 espaços e fica com o seu tamanho total; contudo, **SCROLL**, e certas situações em que a parte de baixo do écran tem mais de duas linhas, pode contrariar isto introduzindo pequenas linhas no fim.

As variáveis têm diferentes formatos, segundo as suas diferentes naturezas.

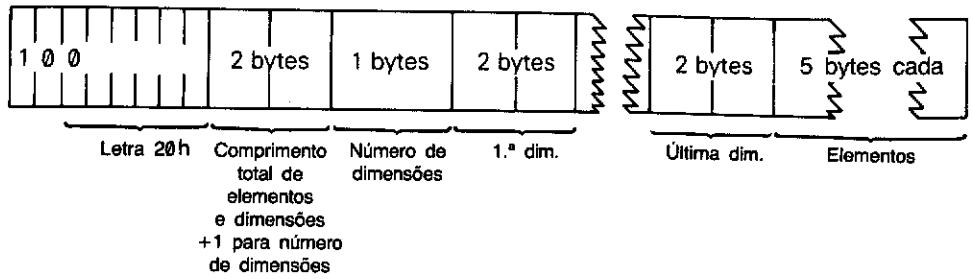
Número cujo nome é apenas uma letra:



Número cujo nome seja maior que uma letra:



Array de números:



A ordem dos elementos é:

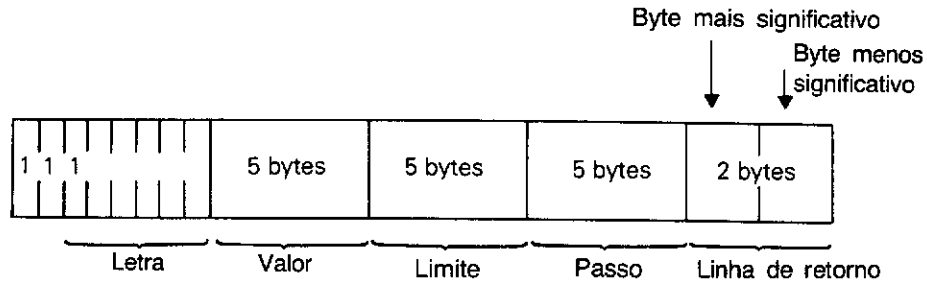
primeiro, os elementos para os quais o primeiro subscrito é 1
a seguir, os elementos para os quais o primeiro subscrito é 2
depois, os elementos para os quais o primeiro subscrito é 3

e assim por diante para todos os possíveis valores do primeiro subscrito.

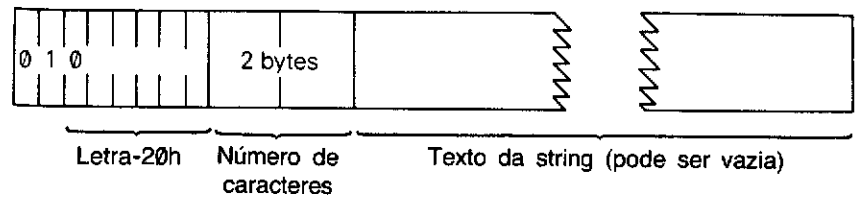
Os elementos com um determinado primeiro subscrito são ordenados da mesma forma usando o segundo subscrito, e assim sucessivamente até ao último.

Como exemplo, os elementos do array B(3 x 6) no Capítulo 22 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).

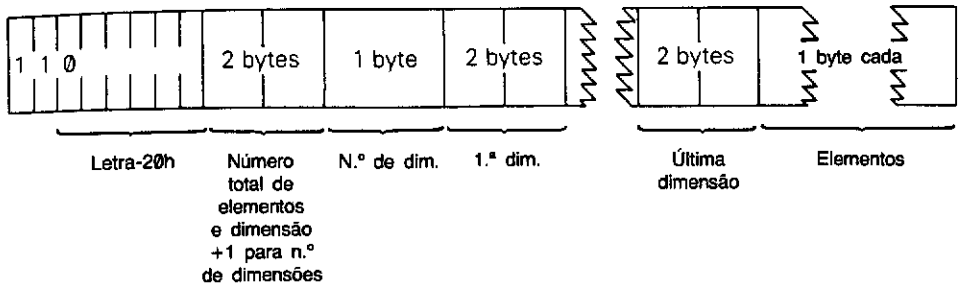
Variável de control para ciclo **FOR-NEXT**:



String:



Array de caracteres:



A parte que começa em E LINE contém a linha que está a ser escrita (como programa, linha de programa ou dados **INPUT**) e também algum espaço de trabalho.

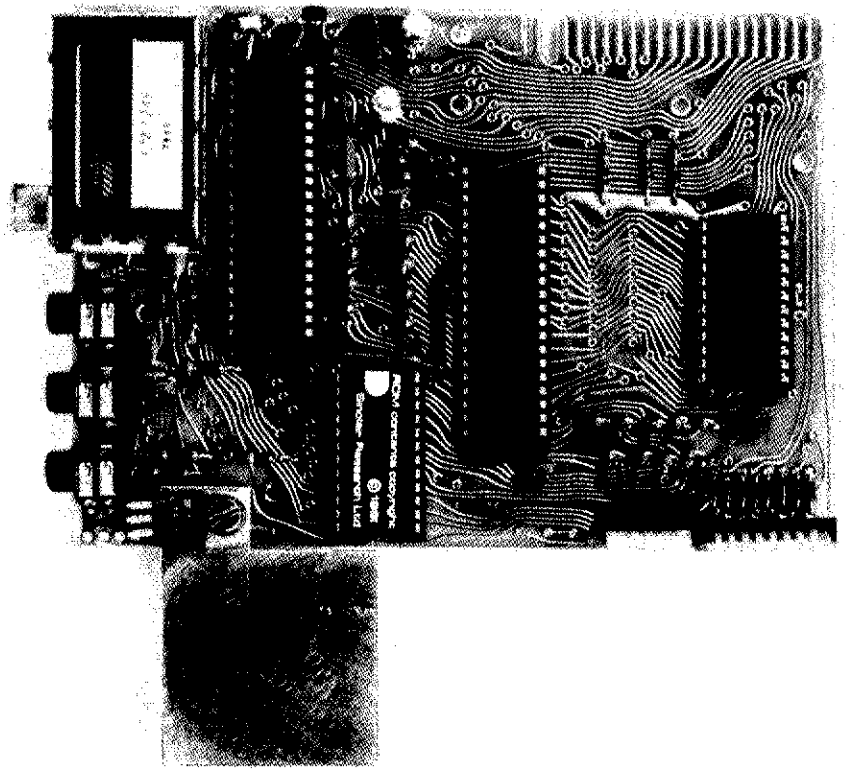
A calculadora é a parte do sistema BASIC que trabalha com a aritmética, e os números com que está a trabalhar são mantidos na pilha de cálculo.

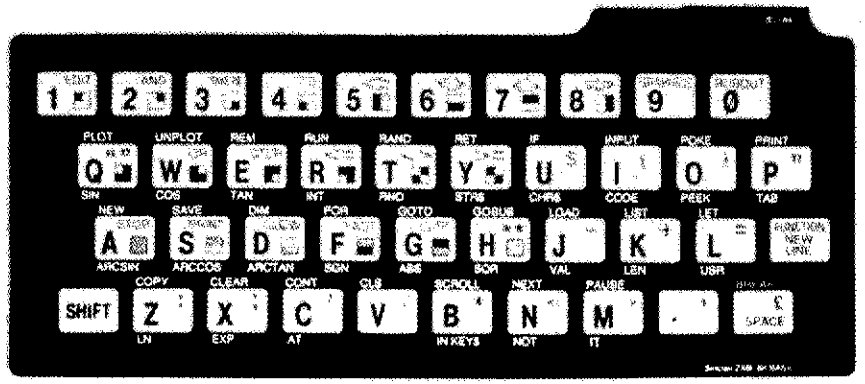
A parte restante contém o espaço não utilizado até aí.

A pilha máquina é a pilha utilizada pelo circuito Z8Ø para manter os endereços de retorno, etc.

A pilha **GOSUB** foi mencionada no Capítulo 14.

O espaço para as rotinas **USR** tem que ser estabelecido por si, usando **NEW** como descrevemos no capítulo anterior.





Capítulo 28

As variáveis de sistema

Os bytes em memória de 16384 a 16508 são estabelecidos pelo sistema para utilizações específicas. Pode utilizá-los para descobrir inúmeras coisas acerca do sistema, e alguns dele podem ser utilmente alterados. Estão aqui listados com os seus usos.

Estes bytes denominam-se variáveis de sistema, e tem nomes, mas não as confunda com as variáveis utilizadas pelo BASIC. O computador não identificará os nomes como variáveis de sistema, e são dados apenas como mnemónicas * para os seres humanos.

As abreviaturas na coluna 1 têm os seguintes significados:

X A variável não deve ser alterada ou pode destruir o programa.
N Se a variável for alterada não terá um efeito duradouro.
S A variável é guardada por **SAVE.**

O número na coluna 1 é o número de bytes da variável. Quando tem dois bytes, o primeiro é menos significativo — o inverso do que esperava. Assim, para colocar o valor v numa variável de dois bytes no endereço n, faça

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

e para ler o valor do registo, use a expressão

```
PEEK n+256*PEEK (n+1)
```

Notas	*Endereço*	*Nome*	*Conteúdo*
1	16384	ERR_NR	Menos 1 que o código de relatório. Começa em 255 (para −1), assim **PEEK** 16384, se funcionar dá 255. **POKE** 16384, n pode-se utilizar para obrigar a parar com um erro: 0 ≤ n ≤ 14 dá um dos relatórios usuais, 15 ≤ n ≤ 34 ou 99 ≤ n ≤ 127 dá um relatório não-standard, e 35 ≤ n ≤ 98 é capaz de alterar o ficheiro de projecção.
X1	16385	BANDEIRAS	Diversas bandeiras para controlar o sistema BASIC.
X2	16386	ERR_SP	Endereço do primeiro elemento (item) da pilha máquina depois dos retornos **GOSUB**.)
2	16388	RAMTOP	Endereço do primeiro byte acima da zona do sistema BASIC. Pode alterar o registo para reservar espaço (através de **NEW**) acima dessa zona (ver Capítulo 26) ou para lograr

MNEMÓNICA — Um tipo de notação, representação ou etiqueta que normalmente, é constituído por uma abreviatura de um dado elemento, ou seja, aquele que se representa.

Notas	Endereço	Nome	Conteúdo
			CLS estabelecendo um ficheiro de projecção mínimo (Capítulo 27). Alterar o RAMTOP não surte efeito até um deles ser executado.
N1	16390	MODO	Cursores especificados K, L, F ou G.
N2	16391	PPC	Número de linha de instrução que está a ser executado. Alterar o registo não tem efeito duradouro, excepto na última linha do programa.
S1	16393	VERSN	Ø identifica ZX81 BASIC nos programas guardados.
S2	16394	E_PPC	Número da linha corrente (com o cursor do programa).
SX2	16396	D_FILE	Ver Capítulo 27.
S2	16398	DF_CC	Endereço da posição **PRINT** no ficheiro de projecção. Pode alterá-lo de modo a que a saída **PRINT** seja enviada para outro sítio.
SX2	16400	VARS	Ver Capítulo 27.
SN2	16402	DEST	Endereço da variável na atribuição.
SX2	16404	E_LINE	Ver Capítulo 27.
SX2	16406	CH_ADD	Endereço do novo caracter a ser interpretado: o caracter depois do argumento de **PEEK**, ou a **NEWLINE** no fim de uma instrução **POKE**.
S2	16408	X_PTR	Endereço do caracter que precede o marcador **S**.
SX2	16410	STKBOT	
SX2	16412	STKEND	Ver Capítulo 27.
SN1	16414	BERG	Registo da calculadora.
SN2	16415	MEM	Endereço da zona usada para a memória da calculadora (que normalmente é MEMBOT, mas nem sempre o é).
S1	16417	não é usado	
SX1	16418	DF_SZ	O número de linhas (incluindo uma linha em branco) na parte de baixo do écran.
S2	16419	S_TOP	O número da linha no cimo do programa nas listagens automáticas.
SN2	16421	LAST_K	Mostra que tecla se primiu.
SN1	16423		Denuncia a situação do teclado.
SN1	16424	MARGIN	Número de linhas em branco abaixo ou acima da figura: 55 em Portugal e Inglaterra, 31 na América.
SX2	16425	NXTLIN	Endereço da próxima linha do programa a ser executada.
S2	16427	OLDPPC	Número da linha donde salta **CONT**.
SN1	16429	FLAGX	Diversas bandeiras (flags).

Notas	Endereço	Nome	Conteúdo
SN2	16430	STRLEN	Comprimento do destino do tipo string na atribuição.
SN2	16432	T_ADDR	Endereço do ìtem seguinte no quadro de sintaxe (pouca probabilidade de utilização).
S2	16434	SEED	A origem de **RND**. Esta é a variável estabelecida por **RAND**.
S2	16436	FRAMES	Conta os pulsos (Frames) projectados no televisor. O bite 15 é 1. Os bites de Ø a 14 são decrementadas (referenciando a diminuição da grandeza das variáveis) em cada pulso transmitido à TV. Isto pode ser utilizado para calcular a duração, embora **PAUSE** também o utilize. **PAUSE** anula para Ø o bite 15, e põe em bites de Ø a 14 o comprimento da pausa. Quando estes tiverem sido contados decrescentemente até Ø, a pausa pára. Se a pausa parar devido a uma depressão da tecla, o bite 15 volta novamente a 1.
S1	16438	COORDS	A coordenada-x do último ponto é assinalada com **PLOT**.
S1	16439		A coordenada-y do último ponto é assinalada com **PLOT**.
S1		PR_CC	O byte menos significativo do endereço da posição seguinte para **LPRINT** imprimir (em PRBUFF).
	16440		
SX1	16441	S_POSN	Número da coluna para a posição **PRINT**.
SX1	16442		Número de linha para a posição **PRINT**.
S1	16443	CDFLAG	Diversas bandeiras. Bite 7 está em (1) durante o modo de programação.
S33	16444	PRBUFF	Tampão da impressora (buffer) (33.° caracter é **NEWLINE**).
SN30	16477	MEMBOT	A zona de memória da calculadora; é usada para armazenar números que não podem ser postos convenientemente na pilha da calculadora.
S2	16507	não é usado	

Exercícios

1. Tente escrever este programa

```
10 FOR N=0 TO 21
20 PRINT PEEK (PEEK 16400+256*PEEK 16401+N)
30 NEXT N
```

Isto diz-lhe os primeiros 22 bytes da zona das variáveis: experimente a variável de control N de acordo com o Capítulo 27.

2. No programa acima, mude a linha 20 para

```
20 PRINT PEEK (16509+N)
```

Isto dá-lhe os primeiros 22 bytes da zona do programa. Experimente isto com o próprio programa.

Os caracteres

Este é o conjunto completo de caracteres do ZX80, com códigos em decimal e hexadecimal. Se imaginarmos que os códigos são as instruções da linguagem máquina Z80, a coluna direita dá a correspondente linguagem de assemblagem mneumónica. Como já deve saber, certas instruções Z80 são compostos que começam com CBh ou EDh; as duas colunas da direita ilustram isto.

Código	*Caracter*	*Hex A*	*Assembler Z80*	*— Depois de CBh*
0	espaço	00	nop	rlc b
1	▘	01	ld bc,NN	rlc c
2	▝	02	ld (bc),a	rlc d
3	▀	03	inc bc	rlc e
4	▖	04	inc b	rlc h
5	▌	05	dec b	rlc l
6	▞	06	ld b,N	rlc (hl)
7	▛	07	rlca	rlc a
8	▗	08	ex af,af'	rrc b
9	▚	09	add hl,bc	rrc c
10	▐	0A	ld a,(bc)	rrc d
11	"	0B	dec bc	rrc e
12	£	0C	inc c	rrc h
13	$	0D	dec c	rrc l
14	:	0E	ld, c,N	rrc (hl)
15	?	0F	rrca	rrc a
16	(	10	djnz DIS	rl b
17	)	11	ld de,NN	rl c
18	>	12	ld (de),a	rl d
19	<	13	inc de	rl e
20	=	14	inc d	rl h
21	+	15	dec d	rl l
22	−	16	ld d,N	rl (hl)
23	*	17	rla	rl a
24	/	18	jr DIS	rr b
25	;	19	add hl,de	rr c
26	,	1A	ld a,(de)	rr d
27	.	1B	dec de	rr e
28	0	1C	inc e	rr h
29	1	1D	dec e	rr l
30	2	1E	ld e,N	rr (hl)
31	3	1F	rra	rr a
32	4	20	jr nz,DIS	sla b
33	5	21	ld hl,NN	sla c
34	6	22	ld (NN),hl	sla d

Código	Caracter	Hex A	Assembler Z80	– depois de CBh	– Depois de EDh
35	7	23	inc hl	sla e	
36	8	24	inc h	sla h	
37	9	25	dec h	sla l	
38	A	26	ld, h,N	sla (hl)	
39	B	27	daa	sla a	
40	C	28	jr z,DIS	sra b	
41	D	29	add hl,hl	sra c	
42	E	2A	ld hl,(NN)	sra d	
43	F	2B	dec hl	sra e	
44	G	2C	inc l	sra h	
45	H	2D	dec l	sra l	
46	I	2E	ld,l,N	sra (hl)	
47	J	2F	cpl	sra a	
48	K	30	jr nc,DIS		
49	L	31	ld sp,NN		
50	M	32	ld (NN),a		
51	N	33	inc sp		
52	O	34	inc (hl)		
53	P	35	dec (hl)		
54	Q	36	ld (hl),N		
55	R	37	scf		
56	S	38	jr c,DIS	srl b	
57	T	39	add hl,sp	srl c	
58	U	3A	ld a,(NN)	srl d	
59	V	3B	dec sp	srl e	
60	W	3C	inc a	srl h	
61	X	3D	dec a	srl l	
62	Y	3E	ld a,N	srl (hl)	
63	Z	3F	ccf	srl a	
64	**RND**	40	ld b,b	bit 0,b	in b,(c)
65	**INKEY$**	41	ld b,c	bit o,c	out (c),b
66	**PI**	42	ld b,d	bit 0,d	sbc hl,bc
67		43	ld b,e	bit 0,e	ld (NN),bc
68		44	ld b,h	bit 0,h	neg
69		45	ld b,l	bit 0,l	retn
70		46	ld b,(hl)	bit 0,(hl)	im 0
71		47	ld b,a	bit 0,a	ld i,a
72	not used	48	ld c,b	bit 1,b	in c,(c)
73		49	ld c,c	bit 1,c	out (c),c
74		4A	ld c,d	bit 1,d	adc hl,bc
75		4B	ld c,e	bit 1,e	ld bc,(NN)
76		4C	ld c,h	bit 1,h	

Código	Caracter	Hex A	Assembler Z80	– depois de CBh	– Depois de EDh
77		4D	ld c,l	bit 1,l	reti
78		4E	ld c,(hl)	bit 1,(hl)	
79		4F	ld c,a	bit 1,a	ld r, a
80		50	ld d,b	bit 2,b	in d,(c)
81		51	ld d,c	bit 2,c	out (c),d
82		52	ld d,d	bit 2,d	sbc hl,de
83		53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	ld a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl,de
91		5B	ld e,e	bit 3,e	ld de,(NN)
92		5C	ld e,h	bit 3,h	
93		5D	ld e,l	bit 3,l	
94	not used	5E	ld e,(hl)	bit 3,(hl)	im 2
95		5F	ld e,a	bit 3,a	ld a, r
96		60	ld h,b	bit 4,b	in h,(c)
97		61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103		67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	cursor acima ◇	70	ld (hl),b	bit 6,b	
113	cursor abaixo ◇	71	ld (hl),c	bit 6,c	
114	cursor esquerda ◊	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor direita ◊	73	ld (hl),e	bit 6,e	ld (NN),sp
116	**GRAPHICS**	74	ld (hl),h	bit 6,h	
117	**EDIT**	75	ld (hl),l	bit 6,l	
118	**NEWLINE**	76	halt	bit 6,(hl)	

Código	Caracter	Hex A	Assembler Z80	– depois de CBh	– Depois de EDh
119	**RUBOUT**	77	ld (hl),a	bit 6,a	
120	◧ / ◨ mode	78	ld a,b	bit 7,b	in a,(c)
121	**FUNCTION**	79	ld a,c	bit 7,c	out (c),a
122	não é usado	7A	ld a,d	bit 7,d	adc hl,sp
123	não é usado	7B	ld a,e	bit 7,e	ld sp,(NN)
124	não é usado	7C	ld a,h	bit 7,h	
125	não é usado	7D	ld a,l	bit 7,l	
126	número	7E	ld a,(hl)	bit 7,(hl)	
127	cursor	7F	ld a,a	bit 7,a	
128	▉	80	add a,b	res 0,b	
129	▊	81	add a,c	res 0,c	
130	▋	82	add a,d	res 0,d	
131	▌	83	add a,e	res 0,e	
132	▍	84	add a,h	res 0,h	
133	▎	85	add a,l	res 0,l	
134	▏	86	add a,(hl)	res 0,(hl)	
135	▐	87	add a,a	res 0,a	
136	▒	88	adc a,b	res 1,b	
137	▓	89	adc a,c	res 1,c	
138	▒	8A	adc a,d	res 1,d	
139	inverso "	8B	adc a,e	res 1,e	
140	inverso £	8C	adc a,h	res 1,h	
141	inverso $	8D	adc a,l	res 1,l	
142	inverso :	8E	adc a,(hl)	res 1,(hl)	
143	inverso ?	8F	adc a,a	res 1,a	
144	inverso (	90	sub b	res 2,b	
145	inverso)	91	sub c	res 2,c	
146	inverso >	92	sub d	res 2,d	
147	inverso <	93	sub e	res 2,e	
148	inverso =	94	sub h	res 2,h	
149	inverso +	95	sub l	res 2,1	
150	inverso –	96	sub (hl)	res 2,(hl)	
151	inverso *	97	sub a	res 2,a	
152	inverso /	98	sbc a,b	res 3,b	
153	inverso ;	99	sbc a,c	res 3,c	
154	inverso ,	9A	sbc a,d	res 3,d	
155	inverso .	9B	sbc a,e	res 3,e	
156	inverso 0	9C	sbc a,h	res 3,h	
157	inverso 1	9D	sbc a,l	res 3,l	
158	inverso 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverso 3	9F	sbc a,a	res 3,a	
160	inverso 4	A0	and b	res 4,b	ldi

Código	Caracter	Hex A	Assembler Z80	– depois de CBh	– Depois de EDh
161	inverso 5	A1	and c	res 4,c	cpi
162	inverso 6	A2	and d	res 4,d	ini
163	inverso 7	A3	and e	res 4,e	outi
164	inverso 8	A4	and h	res 4,h	
165	inverso 9	A5	and l	res 4,1	
166	inverso A	A6	and (hl)	res 4,(hl)	
167	inverso B	A7	and a	res 4,a	
168	inverso C	A8	xor b	res 5,b	ldd
169	inverso D	A9	xor c	res 5,c	cpd
170	inverso E	AA	xor d	res 5,d	ind
171	inverso F	AB	xor e	res 5,e	outd
172	inverso G	AC	xor h	res 5,h	
173	inverso H	AD	xor l	res 5,l	
174	inverso I	AE	xor (hl)	res 5,(hl)	
175	inverso J	AF	xor a	res 5,a	
176	inverso K	B0	or b	res 6,b	ldir
177	inverso L	B1	or c	res 6,c	cpir
178	inverso M	B2	or d	res 6,d	inir
179	inverso N	B3	or e	res 6,e	otir
180	inverso O	B4	or h	res 6,h	
181	inverso P	B5	or l	res 6,l	
182	inverso Q	B6	or (hl)	res 6,(hl)	
183	inverso R	B7	or a	res 6,a	
184	inverso S	B8	cp b	res 7,b	lddr
185	inverso T	B9	cp c	res 7,c	cpdr
186	inverso U	BA	cp d	res 7,d	indr
187	inverso V	BB	cp e	res 7,e	otdr
188	inverso W	BC	cp h	res 7,h	
189	inverso X	BD	cp l	res 7,l	
190	inverso Y	BE	cp (hl)	res 7,(hl)	
191	inverso Z	BF	cp a	res 7,a	
192	""	C0	ret nz	set 0,b	
193	**AT**	C1	pop bc	set 0,c	
194	**TAB**	C2	jp nz,NN	set 0,d	
195	não é usado	C3	jp NN	set 0,e	
196	**CODE**	C4	call nz,NN	set 0,h	
197	**VAL**	C5	push bc	set 0,l	
198	**LEN**	C6	add a,N	set 0,(hl)	
199	**SIN**	C7	rst 0	set 0,a	
200	**COS**	C8	ret z	set 1,b	
201	**TAN**	C9	ret	set 1,c	
202	**ASN**	CA	jp z,NN	set l,d	

Código	*Caracter*	*Hex A*	*Assembler Z80*	*— depois de CBh*
203	**ACS**	CB		set l,e
204	**ATN**	CC	call z,NN	set 1,h
205	**LN**	CD	call NN	set 1,l
206	**EXP**	CE	adc a,N	set 1,(hl)
207	**INT**	CF	rst 8	set 1,a
208	**SQR**	D0	ret nc	set 2,b
209	**SGN**	D1	pop de	set 2,c
210	**ABS**	D2	jp nc,NN	set 2,d
211	**PEEK**	D3	out N,a	set 2,e
212	**USR**	D4	call nc,NN	set 2,h
213	**STR$**	D5	push de	set 2,l
214	**CHR$**	D6	sub N	set 2,(hl)
215	**NOT**	D7	rst 16	set 2,a
216	**	D8	ret c	set 3,b
217	**OR**	D9	exx	set 3,c
218	**AND**	DA	jp c,NN	set 3,d
219	<=	DB	in a,N	set 3,e
220	>=	DC	call c,NN	set 3,h
221	<>	DD	prefixa instruções usando ix	set 3,1
222	**THEN**	DE	sbc a,N	set 3,(hl)
223	**TO**	DF	rst 24	set 3,a
224	**STEP**	E0	ret po	set 4,b
225	**LPRINT**	E1	pop hl	set 4,c
226	**LLIST**	E2	jp po,NN	set 4,d
227	**STOP**	E3	ex (sp),hl	set 4,e
228	**SLOW**	E4	call po,NN	set 4,h
229	**FAST**	E5	push hl	set 4,l
230	**NEW**	E6	and N	set 4,(hl)
231	**SCROLL**	E7	rst 32	set 4,a
232	**CONT**	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	**GOTO**	EC	call pe,NN	set 5,h
237	**GOSUB**	ED		set 5,l
238	**INPUT**	EE	xor N	set 5,(hl)
239	**LOAD**	EF	rst 40	set 5,a
240	**LIST**	F0	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

Código	*Caracter*	*Hex A*	*Assembler Z80*	*— depois de CBh*
244	**POKE**	F4	call p,NN	set 6,h
245	**PRINT**	F5	push af	set 6,l
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	**RAND**	F9	ld sp,hl	set 7,c
250	**IF**	FA	jp m,NN	set 7,d
251	**CLS**	FB	ei	set 7,e
252	**UNPLOT**	FC	call m,NN	set 7,h
253	**CLEAR**	FD	prefixa instruções usando iy	set 7,l
254	**RETURN**	FE	cp N	set 7,(hl)
255	**COPY**	FF	rst 56	set 7,a

Lista de códigos

Esta tabela dá cada código de relatório com uma descrição geral e uma lista das situações em que pode ocorrer. No Apêndice C, há uma descrição mais detalhada do significado dos relatórios de erro sob cada instrução ou função.

Código	*Significado*	*Situações*
0	Realização ou salto feliz para o número de linha maior que existe. Um relatório com código 0 não altera o número de linha usado por **CONT**.	Todas
1	A variável de controle não existe (não foi estabelecida por uma instrução **FOR**) mas há uma variável vulgar com o mesmo nome.	**NEXT**
2	Utilizou-se uma variável indefinida. Para uma variável simples, isto acontecerá se a variável for utilizada antes de ser atribuída a uma instrução **LET.** Para uma variável subscrita, acontecerá se a variável for utilizada antes de ser dimencionada numa instrução **DIM.** Para uma variável de controle, acontecerá se a variável for utilizada antes de ser estabelecida como variável de controle numa instrução **FOR**, quando não houver nenhuma variável vulgar simples com o mesmo nome.	Todas
3	Subscrito fora de escala. Se o subscrito está de facto fora da escala (negativa, ou maior que 65535), o erro B aparecerá como resultado.	Variáveis subscritas
4	Não há espaço suficiente na memória. Repare que o número de linha no relatório (depois de /) pode aparecer incompleto no écran, devido à memória ser curta: por exemplo, 4/20 pode aparecer como 4/2. Ver Capítulo 23. Para **GOSUB**, ver exercício 6 no Capítulo 15.	**LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB**. Por vezes durante a avaliação da função.
5	Não há mais espaço no écran. **CONT** arranjará espaço, limpando o écran.	**PRINT, LIST**.
6	Overflow na aritmética: os cálculos levaram a um número a mais de 10^{38}.	Toda a aritmética
7	Não há **GOSUB** correspondente para uma instru-	**RETURN**

Código	*Significado*	*Situações*
	ção **RETURN.**	
8	Tentou-se **INPUT** como ordem (não permitido).	**INPUT**
9	Instrução **STOP** executada. **CONT** não tentará executar novamente a instrução **STOP.**	**STOP**
A	Argumento inválido para certas funções.	**SQR, LN, ASN, ACS**
B	Número inteiro fora de escala. Quando se quer um número inteiro, o argumento em vírgula flutuante é arredondado para o número inteiro mais próximo. Se estiver fora da escala adequada, dá como resultado o erro B.	**RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, LLIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR**
	Para acesso array, ver também relatório 3.	Acesso array
C	O texto do argumento (string) de **VAL** Não forma uma expressão numérica válida.	**VAL**
D	a) Programa interrompido por **BREAK.**	No fim de qualquer instrução ou em **LOAD, SAVE, LPRINT, LLIST,** ou **COPY.**
	b) A linha **INPUT** começa com **STOP.**	
E	Não utilizado.	
F	O nome do programa dado é a string vazia.	**SAVE**

O ZX81 para quem sabe BASIC

Geral

Se já conhece o BASIC não terá problemas em usar o ZX81; mas tem uma ou duas idiossincrasias.

a) As palavras não são soletradas, mas têm chaves próprias — isto é abordado no Capítulo 2 (para palavras-chave e teclas com SHIFT) e no Capítulo 5 (para nomes de funções). No texto, essas palavras são impressas em **BOLD TYPE** (negrito).

b) O BASIC ZX81 não tem READ (ler), DATA (dados) e RESTORE (restaurar) (mas veja o exercício 3 do Capítulo 22), funções definidas (FN e DEF; mas **VAL** pode, por vezes, ser utilizado), e linhas de multi-instruções.

c) As facilidades de manuseamento da string são compreensíveis mas não standardizadas — ver Capítulo 21 e Capítulo 22 (arrays string).

d) Os caracteres do ZX81 são próprios.

e) A projecção no televisor não está no mapa geral da memória.

f) Se costuma utilizar **PEEK** e **POKE** numa máquina diferente, lembre-se que todos os endereços serão diferentes no ZX81.

Velocidade

O aparelho trabalha em duas velocidades, os modos programação e projecção e rápido.

Em programação e projecção, a imagem de TV é gerada continuamente e a programação é feita durante as partes em branco no princípio e no fim.

No modo rápido, a imagem TV é desligada durante a programação, e só é projectada no fim de um programa, enquanto espera pelos dados **INPUT,** ou durante uma pausa (ver **PAUSE**).

O modo rápido é quatro vezes mais rápido, e deve ser usado para programas com muita computação e poucas saídas ou quando se estão a escrever grandes programas.

A mudança de modos de velocidade faz-se através das instruções **FAST** (rápido) e **SLOW** (lento).

O teclado

Os caracteres do ZX81 compreendem não só símbolos (letras, dígitos, etc.), mas também os sinais compostos (palavras-chave, nomes de funções, etc., que estão aqui escritos em **NEGRITO (BOLD TYPE)**), e todos eles são introduzidos através do teclado e não soletrados. Para isso, há teclas que têm até cinco significados diferentes, que em parte são introduzidos primindo simultaneamente **SHIFT** e em parte pondo o aparelho em modos diferentes.

O modo é indicado pelo cursor, uma letra em inverso vídeo que mostra onde vai ser integrado o caracter seguinte vindo do teclado.

O modo K (para palavras-chave) aparece automaticamente sempre que o aparelho espera uma ordem ou linha de programa (sem serem dados **INPUT**) e, através da sua posiçãona linha, sabe que deve esperar um número de linha ou uma palavra-chave. Isto é: No início da linha, ou logo a seguir a alguns dígitos do princípio da

linha, ou depois de **THEN.**

Se não primirmos **SHIFT** simultaneamente, a tecla seguinte será interpretada ou como palavra-chave (que estão normalmente escritas por cima das teclas), ou como um dígito.

O modo **L** (para letras) aparece normalmente nas outras situações. Se não primirmos simultaneamente **SHIFT,** a tecla seguinte será interpretada como símbolo principal dessa tecla.

Em ambos os modos, **K** e **L**, as teclas SHIFT serão interpretadas como o caracter vermelho no canto superior direito da tecla.

O modo **F** (para funções) aparece depois de primirmos **FUNCTION** (com **NEWLINE**), e dura apenas o tempo de carregarmos apenas uma tecla.

Essa tecla será interpretada como o nome de uma função, que aparece por baixo das teclas.

O modo **G** para gráficos aparece depois de primirmos **GRAPHICS** (com SHIFT 9), e dura até voltar a ser primida. Uma tecla sem fazer SHIFT dá o inverso video do seu modo de interpretação **L**; uma tecla com shift também o fará, se for um símbolo, mas se uma tecla shift dá normalmente um sinal, no modo gráfico dá o símbolo gráfico que aparece no canto inferior direito da tecla.

O écran

O écran tem 24 linhas, cada uma com o comprimento de 32 caracteres, e está dividido em duas partes. A parte de cima é, quando muito, 22 linhas, e projecta ou uma listagem ou as saídas de um programa. A parte de baixo tem, pelo menos, duas linhas, e é usado para introduzir ordens, linhas de programa ou dados **INPUT**, bem como para dar relatórios

Entradas do teclado: aparecem na metade de baixo do écran e escrevem-se, introduzindo cada caracter (símbolo simples ou um sinal composto) imediatamente antes do cursor. O cursor pode mover-se para a esquerda com ◊ (primindo **SHIFT** e 5) ou para a direita com ◊ (primindo **SHIFT** e 8). O caracter antes do cursor pode ser apagado com **RUBOUT** (com **SHIFT** e Ø). (Nota: Pode-se apagar toda a linha primindo **EDIT** (com **SHIFT** e 1) seguido de **NEWLINE**; ou, se forem dados **INPUT**, basta primir **EDIT.**

Quando se escreve **NEWLINE**, a linha é executada, introduzida no programa ou usada devidamente como dados **INPUT**, a menos que contenha um erro de sintaxe. Neste caso, o símbolo **S** aparece logo antes do erro.

À medida que se introduzem as linhas de programa, é projectada, na metade superior do écran, uma listagem. A forma como é feita é bastante complicada, mas está convenientemente explicada no Capítulo 9, exercício 6.

A última linha a ser introduzida denomina-se linha corrente e é indicada pelo símbolo **>**; pode-se modificar isto usando as teclas ◊ (com **SHIFT** e 6) e ◊ (com **SHIFT** e 7). Se primir **EDIT** (com **SHIFT** e 1), a linha corrente desce para a parte inferior do écran e pode ser corrigida parcial ou totalmente.

Quando se executa uma ordem ou se passa um programa, limpa-se primeiro o écran e projecta-se depois a saída na metade de cima do écran, ficando aí até que se introduza uma linha de programa, ou se prima **NEWLINE** com uma linha vazia, ou se prima ◊ ou ◊ . Na parte de baixo aparece um relatório da forma m/n em que m é um código que mostra o que aconteceu (ver Apêndice B), e n é o número da última linha executada — ou Ø para uma ordem. O relatório permanece até se primir uma tecla (e indica o modo **K**).

Em determinadas situações, a tecla **SPACE** (espaço) age como uma tecla **BREAK** (travão), parando o computador com o relatório D. Isto acontece:

a) no fim de uma instrução, enquanto está a passar um programa;

b) enquanto o computador procura um programa na fita;

ou c) enquanto o computador está a utilizar a impressora (ou por engano, tentando utilizá-la quando ela não está lá).

BASIC

Os números são armazenados a uma precisão de 9 ou 1Ø dígitos. O maior número que se pode obter é cerca de $1Ø^{38}$, e o menor (positivo) é cerca de $4 * 1Ø^{-39}$.

Armazena-se um número no ZX81 em binário em vírgula flutuante com um byte de exponencial e ($1 \leq e \leq 255$), e quatro bytes de mantissa m ($1/2 \leq m < 1$). Isto representa o número $m * 2^{e-128}$.

Sendo $1/2 \leq m < 1$, o bite mais significativo da mantissa m é sempre 1. Portanto podemos efectivamente substituí-lo por um bite para mostrar o sinal: Ø para números positivos, 1 para negativos.

O zero tem uma representação especial na qual todos os 5 bytes são Ø.

As variáveis numéricas têm nomes de comprimento variável, começando com uma letra e continuando com letras e dígitos. Todas elas são significativas pelo que, por exemplo, LONGNAME (longo nome) e LONGNAMETOO (também é longo nome) são nomes distintos. Os espaços são ignorados.

As variáveis de controle de ciclos **FOR-NEXT** têm nomes com uma só letra.

Os arrays numéricos têm nomes com o comprimento de uma só letra, que pode ser o mesmo que o nome de uma variável simples. Podem, arbitrariamente, ter várias dimensões de extensão arbitrária. Os subscritos começam em 1.

As strings têm comprimento flexível. O nome de uma string consiste numa simples letra seguida de $.

Os arrays string podem ter arbitrariamente muitas dimensões de extensão arbitrária. O nome é apenas uma letra seguida de $ e pode não ser o mesmo que o de nome de uma string. Todas as strings no mesmo array têm o mesmo comprimento fixo, que é especificado como extra, dimensão final na instrução **DIM**. Os subscriptos começam em 1.

Dividir: Podem-se especificar substrings e strings usando divisores.
Um divisor pode ser

a) vazio

ou

b) expressão numérica

ou

c) expressão numérica de opção **TO**, expressão numérica opcional

e, define-se uma substring ou por,

a) expressão string (divisor)

ou por,

b) variável array string (subscrito,..., subscrito, divisor).

que significa o mesmo que

variável array string (subscrito,..., subscrito) (divisor).

Em a) suponha que a expressão string tem o valor s$.

Se o divisor for vazio, o resultado é s$ considerado como o substring de si mesmo.

Se o divisor for uma expressão numérica com valor m, o resultado é o caracter m de s$ (uma substring de comprimento 1).

Se o divisor tiver a forma c), suponha que a primeira expressão numérica tem valor m (o valor normal é 1), e a segunda, n (o valor normal é o comprimento de s$).

Se $1 \leqslant m \leqslant n \leqslant$ o comprimento de s$, então o resultado é a substring de s$ começando com o caracter m e acabando com o n.

Se $0 \leq n < m$, o resultado é a string vazia.

De outro modo, resultará o erro 3.

Faz a divisão antes das funções ou operações serem avaliadas, a menos que os parêntesis indiquem outro modo.

As substrings podem ser atribuídas (ver **LET**).

O argumento de uma função não necessita de parêntesis se for uma variável ou uma constante (possivelmente subscrita ou dividida).

Função	Tipo de Operando (X)	Resultado
ABS	número	Grandeza absoluta
ACS	número	Arco de seno em radianos. Erro A se X não estiver entre −1 e + 1.
AND	operação binária, o operando direito é sempre um número. Operando numérico esquerdo:	A **AND** B = { A if B ≠ 0 0 if B = 0
	Operando com string à esquerda:	A$ **AND** B = { A$ if B ≠ 0 "" if B = 0
ASN	número	Arco de seno em radianos. Erro A se X não estiver entre −1 e + 1.
ATN	número	Arco de tangente em radianos.
CHR$	número	O caracter cujo código é X, arredondado

Função	Tipo de operando	Resultado
		ao número inteiro mais próximo. Erro B se X não estiver entre 0 e 255
CODE	string	O código do primeiro caracter em X (ou 0 se X for a string vazia).
COS	número (em radianos)	Coseno
EXP	número	e^x
INKEY$	nenhum	Lê o teclado. O resultado é o caracter que representa (em modo L) a tecla primida se houver mesmo uma, além da string vazia.
INT	número	Parte inteira (sempre arredondada por defeito).
LEN	*string*	*comprimento*
LN	número	Logaritmo natural (base e). Erro A se X < 0.
NOT	número	0 se X ≠ 0, 1 se X = 0. **NOT** tem prioridade 4.
OR	operação binária, ambos os operandos são números	A **OR** B = { 1 if B ≠ 0 A if B = 0 **OR** tem prioridade 2.
PEEK	número	O valor do byte em memória cujo endereço é X (arredondado ao número inteiro mais próximo). Erro B se X não estiver de 0 a 65535.
PI	nenhum	π (3.14159265..)
RND	nenhum	O número seguinte, pseudo aleatório y duma sequência gerado ao tirar as potências de 75 módulo 65537, subtraindo 1 e dividindo por 65536. 0 ≤ Y < 1.
SGN	número	Signum: o sinal (−1,0 ou +1) de x.
SIN	número (em radianos)	Seno
SQR	número	Raiz quadrada Erro B se X < 0
STR$	número	A string de caracteres que seria projectada se X fosse primido.
TAN	número (em radianos)	Tangente

Função	Tipo de operando	Resultado
USR	número	Chama a subrotina em linguagem máquina cujo endereço inicial é X (arredondado ao número inteiro mais próximo). No retorno, o resultado é o conteúdo do par registo bc. Erro B se X não estiver entre Ø e 65535.
VAL	string	Avalia X (sem as aspas limite) como uma expressão numérica. Erro C se X tiver um erro de sintaxe, ou dá um valor string. Há outros erros possíveis, dependendo da expressão.
—	número	Negação

As seguintes operações são binárias:

+	Adição (nos números), ou concatenação (nas strings).	
−	Subtracção	
*	Multiplicação	
/	Divisão	
**	Elevado a uma potência. Erro B se o operando for negativo.	
=	Igual	Ambos os operandos devem ser do mesmo tipo. O resultado é um número, 1, se a comparação fôr verdadeira, e Ø se não o fôr.
>	Maior que	
<	Menor que	
<=	Menor ou igual a	
>=	Maior ou igual a	
<>	Diferente de	

As funções e as operações têm as seguintes prioridades:

Operação	Prioridade
Subscrição e divisão de string's	12
Todas as funções excepto **NOT** e a negação unitária	11
**	1Ø
Negação unitária	9
*, /	8
+, — (binária —)	6
=, >, <, < =, > =, < >	5
NOT	4
AND	3
OR	2

Instruções

Nesta lista,

α	Representa uma só letra
v	Representa uma variável
x,y,z	Representam expressões numéricas
m,n	Representam expressões numéricas que são arredondadas ao número inteiro mais próximo
e	Representa uma expressão
f	Representa uma expressão avaliada em string
s	Representa uma instrução

Repare que são permitidas espressões arbitrárias em qualquer lugar (excepto para o número de linha no princípio de uma instrução).

Todas as instruções excepto **INPUT**, podem ser utilizadas ou como ordens ou em programas (apesar de poderem ser mais sensíveis num ou noutro caso).

CLEAR	Apaga todas as variáveis, libertando o espaço que ocupavam.
CLS	(Clear screen = limpa o écran) limpa o ficheiro de projecção. Ver Capítulo 27, relativo ao ficheiro de projecção.
CONT	Suponha que p/q foi o último relatório com um não-zero. Então **CONT** tem o efeito **GOTO** q se p = 9 **GOTO** q+1 se p = 9 (instrução **STOP**)
COPY	Manda uma cópia da imagem para a impressora, se estiver ligada; de outro modo, não faz nada. Ao contrário de todas as outras ordens, uma ordem **COPY** não limpa primeiro o écran. Não podem haver espaços antes de **COPY.** Relatório D se se prime **BREAK.**
DIMα $(n_1, \ldots, n_k)$	Apaga qualquer array com o nome α, e estabelece uma array α de números com K dimensões $n_1, \ldots, n_k$. Inicializa todos os valores a Ø. Erro 4 aparece se não houver espaço para meter a array. Uma array é indefinida até ser dimensionado numa instrução **DIM.**
DIM$\alpha\$(n_1, \ldots, n_k)$	Apaga qualquer array ou string com o nome $\alpha\$$, e estabelece um array $\alpha\$$ de caracteres com K dimensões $n_1, \ldots, n_k$. Inicializa todos os valores para " ". Pode ser considerada como array de string de comprimento fixo n_k, com k−1 dimensões $n_1, \ldots, n_{k-1}$.

Erro 4 aparece se não houver espaço para introduzir a array . Uma array é indefinida até ser dimensionada numa instrução **DIM**.

FAST — Começa o modo rápido, em que o ficheiro de projecção é projectado apenas no fim do programa, ou enquanto os dados **INPUT** estão a ser introduzidos, ou durante uma pausa.

FORα = x **TO** y — **FOR**α = x **TO** y **STEP** 1

FORα = x **TO** y **STEP** z — Apaga qualquer variável simples α, e estabelece uma variável de control com o valor x, limite y, passo (step)z, e endereço de ciclo com mais 1 do que o número de linha da instrução **FOR** (−1 se é uma ordem). Verifica se o valor inicial é maior (se passo $\geq$ Ø) ou menor (se passo $<$ Ø) que o limite, e se assim fôr, salta para a instrução **NEXT** α no princípio de uma linha.
Ver **NEXT** α.
Erro 4 aparece se não há espaço para a variável de control.

GOSUBn — Empurra o número de linha da instrução **GOSUB** para uma pilha; depois é como **GOTO**n.
Erro quatro pode aparecer se não há espaço suficiente para os **RETURN**s.

GOTOn — Salta para a linha n (ou, se não houver nenhuma, para a primeira linha depois dessa).

IF x **THEN** s — Se x for verdadeiro (não-zero), então s é executado. A forma '**IF** x **THEN** número de linha' não é permitido.

INPUT v — Pára (sem prazo especial) e espera que se introduza uma expressão; o seu valor é a atribuição a v. No modo rápido, o ficheiro de projecção é projectado. **INPUT** não pode ser usado como ordem; se tentar dá-se o erro 8.
Se o primeiro caracter na linha **INPUT** for **STOP**, o programa pára com o relatório D.

LET v = e — Atribui o valor de e à variável v.
Uma variável simples é indefinida até ser atribuida a uma instrução **LET** ou **INPUT**.
Se v for uma variável string subscrita, ou uma variável string dividida (substring), a atribuição é Procrusteana: o valor string de e é, ou cortado ou preenchido com espaços no lado direito, para ficar com o mesmo comprimento da variável v.

LIST — **LIST** Ø

LIST n — Faz a listagem do programa na televisão, começando na linha n, e faz de n a linha corrente.
Erro 4 ou 5 se a listagem é demasiado grande para o écran; **CONT** fará novamente o mesmo.

LLIST — **LLIST** Ø

LLIST n — Tal como **LIST**, mas usando a impressora em vez do televisor.
Não fará nada se o printer não estiver ligado.
Pára com o relatório D se se primir **BREAK**.

LOAD f — Procura na fita um programa denominado f, e carrega-o, bem como as suas variáveis.
Se f="", carrega o primeiro programa disponível.
Se se primir **BREAK** ou se detectar um erro de gravação, então
a) se não tiver lido nenhuma parte do programa da gravação, pára com o relatório D;
b) Se já se tiver lido uma parte do programa, faz-se **NEW**.

LPRINT... — Tal como **PRINT**, mas usando o impressor em vez da televisão. Envia uma linha de texto para a impressora.
a) quando a impressão passar de uma linha para a outra,
b) depois de uma instrução **LPRINT** que não acabe em vírgula ou ponto e vírgula,
c) quando uma vírgula ou elemento **TAB** pedir uma nova linha ou
d) no fim do programa, se faltar imprimir alguma coisa.
Num elemento **AT**, só o número da coluna tem efeito; o número da linha é ignorado (excepto que as mesmas condições de erro aparecem como para **PRINT** se estiver fora de escala). Um elemento **AT** nunca manda uma linha de texto para o impressor.
Não acontecerá nada se o impressor não estiver ligado.
Pára com o relatório D se se primir **BREAK**.

NEW — Inicia novamente o sistema BASIC, apagando o programa e as variáveis, e usando — sem incluir — a memória até ao byte cujo endereço está na variável RAMTOP (bytes 16388 e 16389).

NEXT α — a) Acha a variável de control α.
b) Adiciona o seu passo ao seu valor.
c) Se o passo $>$ Ø e o valor $>$ que o limite; ou se o

passo < Ø e o valor < que o limite, salta para a linha de início do ciclo.
Dá erro 1 se houver uma variável simples α.
Dá erro 2 se não houver nenhuma variável α simples nem de control.

PAUSE n

Pára a programação e projecta o ficheiro de projecção durante n pulsos (a 50 pulsos/segundo) ou até se primir uma tecla.
$Ø \leq n \leq 65535$, ou erro B. Se $n \geq 32767$, então a pausa não é medida, mas dura até se primir uma tecla.

PLOTm,n

Enegrece o pixel (|m|,|n|; move a posição **PRINT** para logo depois desse pixel.
$|m| \leq 63, |n| \leq 43$, ou erro B.

POKEm,n

Escreve o valor de n para o byte em memória com endereço m.
$Ø \leq m \leq 65535$, $-255 \leq n \leq 255$, ou erro B.

PRINT...

'...' é uma sequência de elementos **PRINT**, separados por vírgulas ou ponto e vírgula, e são escritos para o ficheiro de projecção exibir na televisão. A posição (linha e coluna) onde vai ser imprimido o caracter seguinte denomina-se posição **PRINT**.
Um elemento **PRINT** pode ser:
a) vazio, i.e. nada
b) uma expressão numérica.
Em primeiro lugar, um sinal negativo é imprimido se o valor for negativo. Imagine que X é o módulo do valor.
Se $x \leq 10^{-5}$ ou $x \geq 10^{13}$, então é imprimido usando a notação científica. A parte da mantissa tem até oito dígitos (sem zeros atrás), e o ponto decimal (que não existe se houver apenas um dígito) está depois do primeiro. A parte exponencial é E, seguida de + ou −, seguida de um ou dois dígitos. Senão, x é escrito em notação decimal simples com até 8 dígitos significativos, se sem zeros depois do ponto decimal. Um ponto decimal logo no princípio é sempre seguido de um zero; assim, por exemplo: .Ø3 e Ø.3 escrevem-se assim.
Ø é escrito como um só dígito Ø,
c) Uma expressão string.
Os sinais de string expandem-se, possivelmente com um espaço antes ou depois.
O caracter imagem de aspas aparece como ".

Os caracteres não habituais e os caracteres de control escrevem-se como?
d) AT m,n
altera-se a posição **PRINT** para a linha |m| (contando do princípio), coluna n (contando da esquerda).
$m < Ø$ ou $m > 21$ dá erro 5 se $|m| = 22$ ou 23, ou então dá erro B.
$|n| \leq 31$, ou erro B.
e) **TAB**n
n é reduzido ao módulo 32. Depois move-se a posição **PRINT** para a coluna n, ficando na mesma linha, a menos que se tivesse que espaçar à rectaguarda, e nesse caso move-se para a linha seguinte.
$|n| \leq 255$, ou erro B.
Um ponto e vírgula entre dois elementos não altera a posição **PRINT**, e assim e segundo elemento segue imediatamente o primeiro. Por outro lado, uma vírgula move a posição **PRINT** pelo menos um espaço e, senão apesar de serem necessárias muitas para o deixar na coluna Ø ou 16, põe-se uma nova linha se necessário.
No fim da instrução **PRINT**, se não acabar em ponto e vírgula, lança-se uma nova linha.
O erro 4 (falta de memória) pode aparecer com 3K ou menos de memória.
O erro 5 significa que o écran está cheio.
Em ambos os casos, a solução é **CONT**, tecla que limpa o écran e possibilita a continuação.

RAND

RAND Ø

RAND n

Estabelece a variável de sistema (denominada SEED = = origem) usada para gerar o valor seguinte de **RND**. Em $n \neq Ø$, dá-se um valor n a SEED; se n = Ø então é dado o valor de outra variável do sistema (denominada FRAMES = pulsos) que conta os pulsos projectados até ali no televisor, pelo que deve ser praticamente aleatório. O erro B aparece se n não está entre Ø e 65535.

REM...

Não surte efeito. '...' pode ser qualquer sequência de caracteres excepto **NEWLINE** (ver utilizações num livro de linguagem máquina).

RETURN

Tira um número de linha da pilha **GOSUB**, e salta para a linha que se segue.
O erro 7 aparece quando não há número de linha na pilha.Há um erro no programa; se os **GOSUB** não são devidamente considerados pelos **RETURN**.

RUN	**RUN Ø**
RUN n **SAVE** f	**CLEAR**, e depois **GOTO** n. Grava o programa e variáveis na fita, e chama-lhes f. **SAVE** não deve ser usado dentro de uma rotina **GOSUB.** O erro F aparece se f for a string vazia, o que não é permitido.
SCROLL	Move todo o écran uma linha para cima, perdendo a primeira linha e ficando com uma linha vazia no fim. NB a nova linha está completamente vazia, apenas com um caracter **NEWLINE** e sem espaços. (Ver Capítulo 27).
SLOW	Põe o computador em modo de programação e projecção, no qual o ficheiro de projecção é exibido continuamente, e a programação é feita durante os espaços no princípio e no fim da imagem.
STOP	Pára o programa com o relatório 9. **CONT** retoma-o na linha seguinte.
UNPLOTm,n	Tal como **PLOT**, mas pondo o pixel em branco em vez de o enegrecer.

Cassettes — Programas-Teste

PROGRAMA-TESTE: **MATEMÁTICA**
Para o ZX81 (e ZX80 com 8K ROM BASIC)

Testa a aritmética (adição, subtracção, multiplicação e divisão) com três níveis de dificuldade. O computador faz-lhe 10 perguntas, e classifica de "certo" ou "errado" as suas respostas. Ao fim de 10 perguntas, dá-lhe a pontuação, e a possibilidade de tentar novamente. Registe a sua pontuação no papel para poder ver o progresso que fez.

```
 10 RAND
 20 SLOW
 30 LET F = 0
 50 CLS
 60 PRINT "FUNÇÃO 1 = +; 2 = −; 3 = *; 4 = /"
 70 INPUT A
 80 PRINT„ "NÍVEL 1−3"
 90 INPUT B
100 FOR N = 1 TO 10
110 CLS
120 PRINT "PERGUNTA";N,F;"RESPOSTAS CERTAS"
130 LET C = INT (10**B*RND)
140 LET D = INT (10**B*RND)
150 IF A > 2 THEN LET D = INT (D/(10** (B−1)))+1
160 LET B$ = STR$ C+""A$(A)+""+STR$ D
170 PRINT „ B$;"=";
180 INPUT D
190 PRINT D
200 IF ABS (VAL B$−D) > 0.01 THEN GOTO 240
210 PRINT „"CERTO;PRIMA N/L"
220 LET F = F+1
230 GOTO 250
240 PRINT „"ERRADO PRIMA N/L"
250 INPUT D$
260 NEXT N
270 PRINT „"PONTUAÇÃO";F;"CERTAS, EM 10"
280 INPUT D$
290 RUN
```

PROGRAMA-TESTE: **SNIPER**
Para o ZX81 (e ZX80 com 8K ROM BASIC)

Testa a capacidade de pensar sob tensão! Rodeiam-no 40 inimigos que, a intervalos

regulares, aparecem em 10 posições diferentes. Durante o pequeno espaço de tempo em que eles se mostram, tem que calcular que número de tecla corresponde a essa posição e primi-la. Se acertar, o indivíduo atingido pelo seu tiro abre os braços e as pernas, caindo. O espaço de tempo em que eles estão à vista vai diminuindo progressivamente.

As teclas de tiro são: 1,2,3,4,5,6,7,8,9,0 (da esquerda para a direita). No ZX81, passe este programa no modo **FAST** (RÁPIDO).

```
 40 CLS
 50 LET S = 0
 60 LET C = 0
 70 LET X = INT (RND * 9.9)
 80 LET T = (X+1) * SGN (9-X)
 90 LET X = X * 3
100 PRINT AT 4,X; "   O"
110 PRINT AT 5,X; " ▄▀▄ "
120 PRINT AT 6,X; "  ▒  "
130 PRINT AT 7,X; "▄   ▄"
140 PAUSE 90-C
150 POKE 16437, 255
160 IF INKEY$ < > STR$ T THEN GOTO 250
170 PRINT AT 4,X; "▄   ▄"
180 PRINT AT 5,X; "  ▀  "
190 PRINT AT 6,X; "  ▀  "
200 PRINT AT 7,X; "▀   ▀"
210 LET S = S+1
220 PRINT AT 15, 0; "PONTUAÇÃO"; S
230 PAUSE 60
240 POKE 16437, 255
250 CLS
260 IF C = 40 THEN GOTO 40
270 PAUSE INT (RND * 120)+30
280 POKE 16437, 255
290 LET C = C+1
300 GOTO 70
```

PROGRAMA-TESTE: **TESTE DO TECLADO**
Para o ZX81

Testa a velocidade de reacção. Verá que aparecerá um caracter no écran, a intervalos diferentes. Tem que escrever o mesmo caracter o mais rápido possível.

Ao fim de 10 vezes, aparece a pontuação.

```
 5 CLS
10 LET N = 0
20 FOR M = 1 TO 10
30 FOR B = 0 TO RND * 100+50
40 NEXT B
50 LET B$ = CHR$ (28+35*RND)
60 PRINT B$
70 LET C$ = ""
80 FOR B = 0 TO 30
90 NEXT B
100 LET C$ = INKEY$
110 IF C$ = B$ THEN GOSUB 200
120 CLS
130 NEXT M
140 PRINT "PONTUAÇÃO="; N
150 PRINT ,,,,"PRIMA N/L"
160 INPUT I$
170 RUN
200 LET N=N+1
210 PRINT "CERTO"
220 FOR B=0 TO 20
230 NEXT B
240 RETURN
```

PROGRAMA-TESTE: **CONVERSÃO DE TEMPERATURA**
Para o ZX81 (e ZX80 com 8K ROM BASIC)

Este programa converte a temperatura de graus Fahrenheit para Centígrados, e de Centígrados ou Kelvin (ou absoluta) para Fahrenheit.

```
 10 REM CONVERSOR DE TEMPERATURA
 20 PRINT "INTRODUZA A TEMPERATURA"
 30 INPUT T
 40 PRINT T
 50 PRINT "C,F OU K?";
 60 INPUT Q$
 70 IF Q$ < > "F" AND Q$ < > "C" AND Q$ < > "K" THEN GOTO 60
 80 PRINT Q$
 90 IF Q$ = "F" THEN GOSUB 200
100 IF Q$ = "C" OR Q$ = "K" THEN GOSUB 300
110 CLS
120 PRINT T; "  "; Q$ " =";N; "  ";R$
130 INPUT A$
140CLS
```

```
150 GOTO 10
200 LET N = (T-="")*5/9
210 LET R$ = "C"
220 RETURN
300 LET C = T
310 IF Q$ = "K" THEN LET C = T-273
320 LET N = C*9/5+32
330 LET R$ = "F"
340 RETURN
```


Índice

Este Índice inclui as teclas do teclado e como as obter (o modo — **K**, **L**, **F** ou **G** — e se precisam de ser feitas com **SHIFT** ou não), bem como os seus códigos.

D

E

F

G

TIPOGRAFIA CORREIA
Rua do 4 de Infantaria, 4-B — 1300 LISBOA