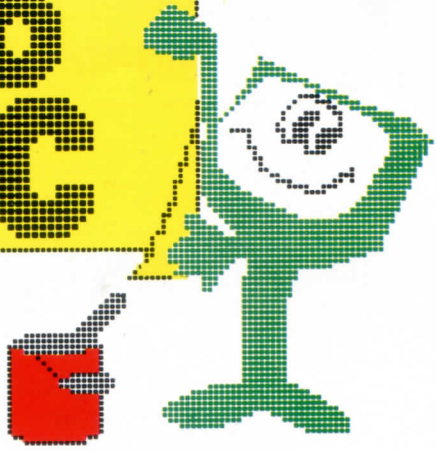
VIDEO BASIC



20 LIÇÕES DE BASIC
PARA APRENDER COM O SPECTRUM

EDIÇÕES LATINAS

JACKSON

- A CPU do SPECTRUM
- BUS, BIT e BYTES
- Sistema binário e hexadecimal
- Técnicas de correcção
- REM – GOTO – IF THEN – CLS
- Verdadeiro ou falso?
- ...A decisão do Spectrum
- Exercícios de vídeo
- Vídeojogo n.º 3

3

Spectrum

16K/48K/PLUS

TIMEX COMPUTER 2048

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

ISBN do tomo 1: 84-85831-12-8
ISBN do fascículo: 84-85831-11-X
ISBN da obra completa: 84-85831-10-1
Depósito Legal N.º: 9960/85
Plano geral da obra:
20 fascículos e 20 cassetes, de publicação quinzenal.
Edições Latinas-Jackson garante a publicação de todos os fascículos e cassetes que compõem esta obra e o fornecimento de qualquer número atrasado, até 1 ano, depois de terminada a sua publicação. Está reservado ao editor, o direito de modificar o preço de venda dos fascículos durante a sua saída, se o mercado assim o exigir.

1.ª Quinzena – Dez. 1985

EDIÇÕES LATINAS
JACKSON

SUMÁRIO

Introdução

Como irás ver, na primeira parte, falaremos da unidade fundamental de qualquer computador, a CPU, explicando qual é a sua função, e dos códigos binário e hexadecimal. Abordaremos, depois, a maneira de escrever correctamente os programas, referindo o EDITOR, o processo de detecção e correcção de erros (DEBUG) e introduzir as instruções REM e CLS. Por fim, depois de termos aprendido o funcionamento da instrução de transferência condicional IF THEM GOTO, poderemos pôr o nosso computador a executar tarefas de uma certa importância, fazendo-o operar como um ser "quase inteligente", graças à sua capacidade de decisão.

Um director principal: a CPU

Principiamos pela análise das diversas unidades que formam o sistema de um computador, partindo do "cérebro" do nosso sistema: a CPU.
CPU significa *Central Processing Unit,* que, traduzido para português, significa «Unidade Central de Processamento».
Esta ocupa-se, essencialmente, em mandar executar todas as instruções necessárias ao funcionamento do computador e do programa. Podemos avaliar a importância desta função comparando-a com um concerto sinfónico. O conjunto da orquestra corresponde ao conjunto de circuitos do computador e o director de orquestra, à CPU. O director (a CPU) lê a partitura (o programa) e transmite ordens à orquestra, de modo a que os instrumentos (as diversas partes do computador) intervenham unicamente quando são solicitados e parem caso seja necessário.
Em seguida, o director encarrega-se de que tudo isto se processe harmonicamente, ou seja, de que o programa se processe segundo esquemas rigorosos.
A CPU «dirige» o computador. Apesar da pesada tarefa que lhe cabe, a CPU não é muito diferente, no seu aspecto físico, dos outros componentes do interior de um computador, de modo que passa quase despercebida. Existem, no mercado, diversos tipos de CPU, de maneira a satisfazerem a maioria das exigências dos seus utentes: algumas são muito rápidas, outras mais potentes, outras ainda mais flexíveis, mas todas elas se assemelham a aranhas pretas com muitas patas e são, à primeira vista, dificilmente diferenciáveis umas das outras.

Felizmente têm um sinal de identificação: Z80A a CPU do Spectrum: 6510 a do C64; 6502, a do VIC 20. À semelhança do que se passa com as interpretações dos directores da orquestra, os diversos modelos de CPU não são equivalentes entre si e requerem instruções e circuitos distintos. Uma vez escolhida a CPU em que se vai operar, projecta-se o computador à sua volta, ficando assim aproximadamente determinadas as tarefas do computador.

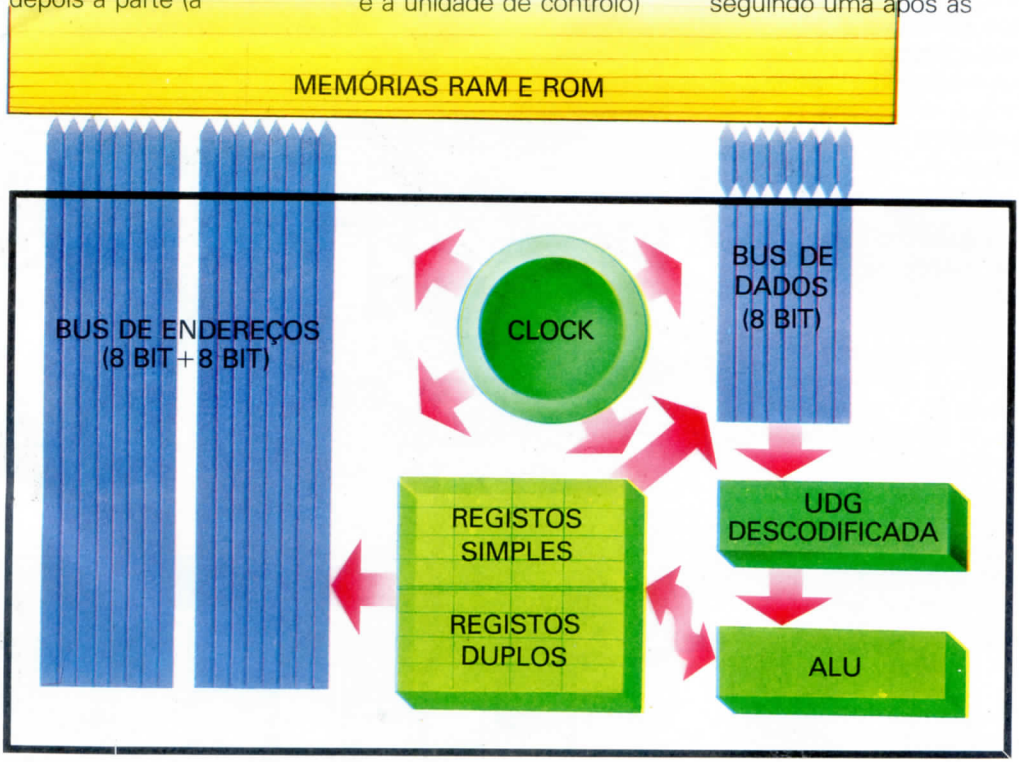
De qualquer maneira, todas as CPU têm que resolver um problema comum: como executar um conjunto de instruções. Esclareçamos melhor o problema. Todas as instruções têm algo em comum: soma, subtrai, acende, apaga, podem ser considerados casos diferentes de uma mesma tarefa.

O que equivale a dizer que estas instruções são consideradas iguais e analisadas do mesmo modo. A diferença reside na execução; com efeito, a compreensão da instrução é algo bem diverso da sua realização. Em particular, uma CPU

considera igualmente todas as ordens quando «traduz» o programa para as suas instruções elementares e as põe em correspondência com diferentes acções.
É como traduzir inglês para português: existem regrais gerais gramaticais e sintáticas bem definidas e pode fazer-se uma tradução sem ter em conta o significado concreto de cada vocábulo. A tradução destes últimos realiza-se depois à parte (a execução da ordem concreta). Em síntese, toda a máquina que «faz contas» (a CPU, portanto) tem de ser capaz de:

1 – representar, registar e manipular números;
2 – identificar uma instrução;
3 – executá-la;
4 – escolher a seguinte.

Para isso são exigidas a algumas partes da CPU (a ALU, o relógio, os registos, a memória exclusiva de leitura e a unidade de controlo) funções bem definidas. Antes, porém, de analisarmos pormenorizadamente estas partes, é melhor ver como funciona realmente uma CPU. Quando ligas o computador, «dás vida» à CPU, que começa a ler um programa existente no seu interior. Este último ordena à CPU que execute o programa que começa no endereço 0 (visto que ela considera o 0 como primeiro número), seguindo uma após as

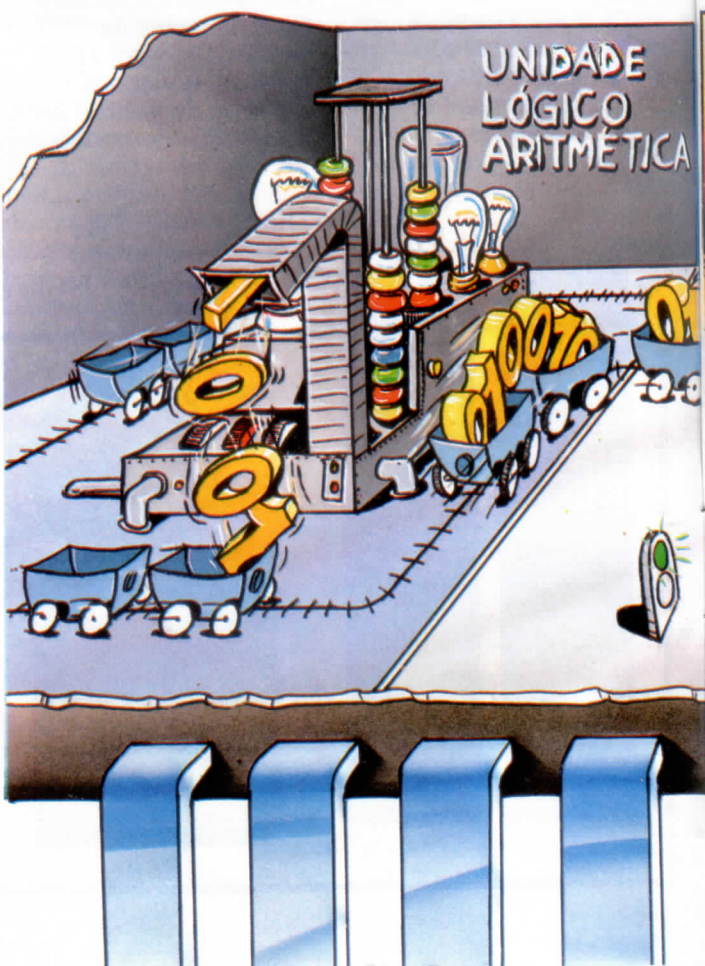


outras, passo a passo, todas as instruções. A CPU lê a instrução no endereço correspondente, compara-a com uma das listas que pode executar, existente no seu interior e encontra a «acção» correspondente a essa ordem. Em última análise, cada instrução é executada através de uma série de operações elementares, realizadas pelos circuitos adequados (internos da CPU), os quais, tal como no exemplo da orquestra, são activados e desactivados com base no programa. Para fazer isso é necessária a presença das unidades anteriormente citadas que são:

1 – A ALU ou Unidade Aritmética-Lógica, que realiza todas as operações matemáticas e lógicas (como a comparação entre dois números);
2 – A ROM, ou memória exclusiva de leitura (interna da CPU) onde estão permanentemente contidas as informações das instruções elementares próprias da CPU;

3 – Os registos, que são memórias temporárias onde a CPU pode armazenar números sem os perder;

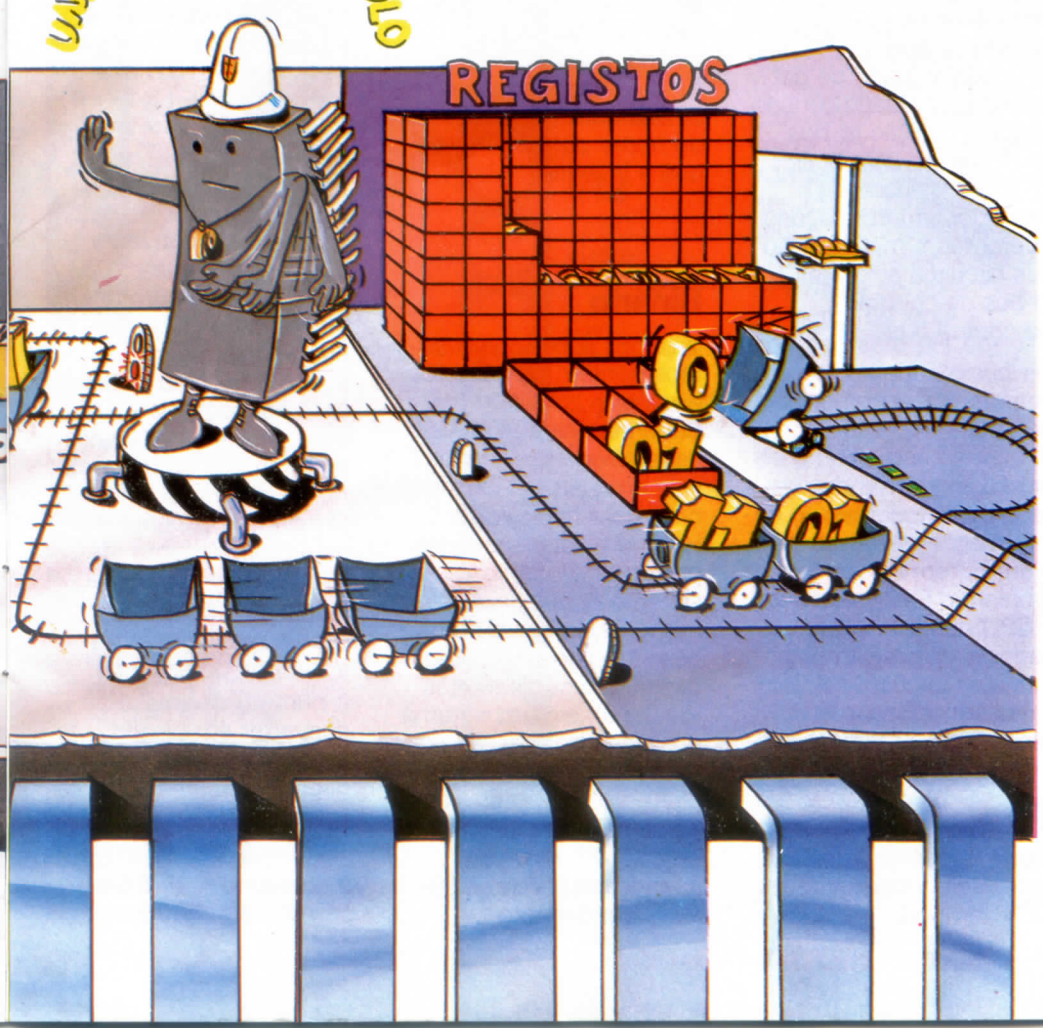
4 – O relógio (*clock*) que marca o tempo a todas as partes do micro-ordenador exactamente da mesma maneira que o metrónomo marca o ritmo ao intérprete. A ROM interna e o relógio (considerados conjuntamente) são usualmente designados unidade de controlo para sublinhar a sua importância na gestão do «microssistema».



O BUS

Para comunicar com os músicos, um director de orquestra recorre à batuta. Qual é, porém, a «batuta» usada pelo computador?

Por outras palavras, até agora dissemos o que é e como actua a CPU, mas não como é capaz de pôr à disposição dos demais circuitos os dados que elabora. Em todas as CPU existem estruturas destinadas a comunicar os dados e a dar a conhecer a outros circuitos as indicações da CPU. No calão do ramo estes circuitos são chamados *Bus*, livremente traduzível como «meio de comunicação» entre



a CPU e o resto do computador. Em cada CPU existe mais de um, para poder acudir a todas as exigências. Exigências que podem ser:

a) a necessidade de comunicar qual a posição de memória que a CPU está a analisar (é o chamado *bus* de endereços ou *adress bus*);
b) a necessidade de dar a conhecer o dado contido naquela posição (o *bus* de dados ou *data bus*);
c) a necessidade de conhecer as informações referentes à manipulação dos circuitos auxiliares (o *bus* de controlo ou *control bus*).

Os *bus* não devem entender-se como uma parte dos circuitos da CPU, mas antes como um canal através do qual se enviam todas as informações relativas a um determinado problema. É o *Bus* e não a CPU que põe estes dados à disposição de todos os circuitos: a CPU limita-se a colocá-los no canal.

Bit

Agora já começamos a entrever a «lógica» da CPU: embora conhecendo unicamente dois estados (ligado ou desligado), activa ou desactiva um certo número de circuitos. A CPU adopta, portanto, uma lógica binária (de dois estados). Para abreviar, estes são indicados por meio de 1 e 0. Cada um destes chama-se *bit* (de *Binary digit*, ou seja, dígito binário) e sobre eles se fundamenta a álgebra binária ou de Boole, do nome do matemático que a inventou.

Binário

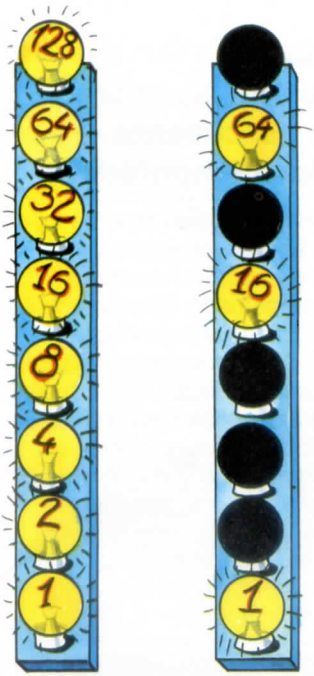
Vejamos agora como se representam os números em lógica binária. Suponhamos que desejamos saber quanto vale em sistema binário o número décimal 39. Há que estabelecer uma premissa. O nosso sistema de escrita dos números processa-se na base 10, isto é, emprega dez símbolos diferentes (de 0 a 9), e é um sistema posicional: o mesmo número toma um valor diferente de acordo com a sua posição (10^0, 10^1, 10^2, ...10^n). Ou seja, quando escrevemos 39, subentendemos a seguinte relação: $39 = 3 \times 10^1 + 9 \times 10^0 = 3 \times 10 + 9 \times 1$. Recordemos que qualquer número elevado a 0 é igual a 1. Em binário vimos que os estados possíveis são 0 e 1 e, por isso, dizemos que é um sistema de base dois (emprega dois símbolos).

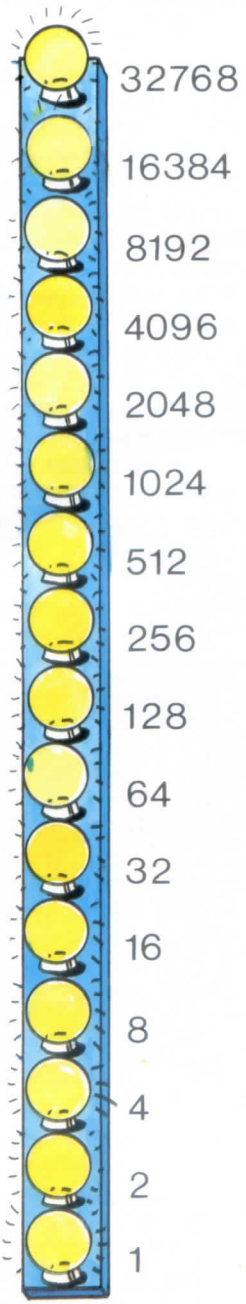
Aqui podemos escrever, tal como para os números décimais: $00100111 = 0 \times \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times \times 2^1 + 1 \times 2^0$

Voltemos, após esta brevíssima nota, ao nosso problema de converter em binário o número 39. Toma-se o número que se pretende converter em binário, regista-se qual é a máxima potência de 2 nele contida e determina-se a diferença entre ambos os números; faz-se o mesmo com o resultado e continua-se este procedimento até se obter um zero como resultado. No nosso caso o procedimento é: a máxima potência de o número décimal a converter em binário, anotanto, a partir da direita, os restos:

39 : 2 = 19 resto 1
19 : 2 = 9 resto 1

2 contida em 39 é 5: $2^5 = 32$, $39 - 32 = 7$;





a máxima potência contida em 7 é 2: $2^2=4$, $7-4=3$; a máxima potência contida em 3 é 1: $2^1=2$, $3-2=1$; a máxima potência contida em 1 é 0: $2^0=1$, $1-1=0$; FIM. Onde não houver uma potência de 2 coloca-se um 0. Obteremos portanto: 100111. Outro método consiste em dividir sucessivamente por dois

$9:2=$ 4 resto 1
$4:2=$ 2 resto 0
$2:2=$ 1 resto 0
$1:2=$ 0 resto 1

Portanto 39 em binário corresponde a 100111. Também podemos representar o número 39 por:

$39=0\times2^7+0\times2^6+1\times$
$\times2^5+0\times2^4+0\times2^3+$
$+1\times2^2+1\times2^1+1\times2^0=$
$=00100111$.

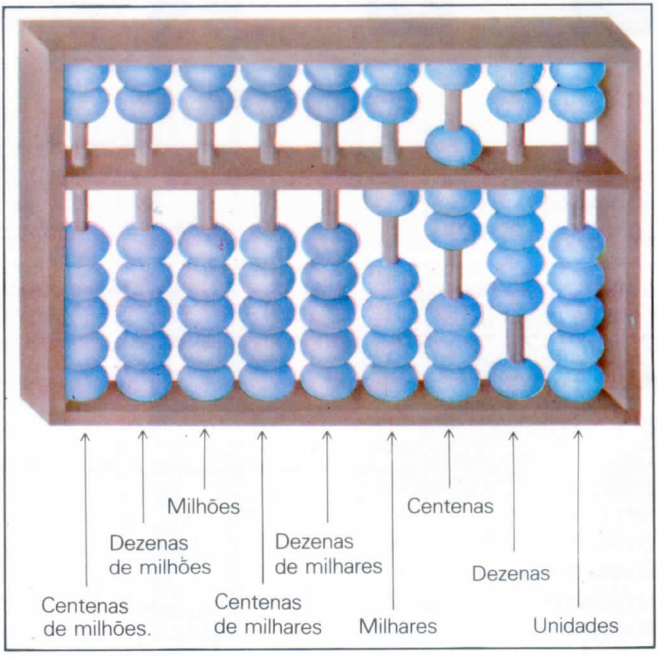
Utilizámos 8 dígitos binários em vez de 6 (00100111). É como escrever 039. Isto, que não parece fazer grande sentido, é, na realidade, muito importante para um computador, visto que utiliza sempre números de igual comprimento por óbvias exigências práticas. No caso do teu *Spectrum*, o comprimento padrão é de *8 bits*. Como a linguagem do computador se baseia em números, pensou-se chamar a estes agrupamentos *palavra* ou *byte*. O número mais alto representável por um *byte* é 11111111 = 255. Mas se quisermos representar um número mais elevado? Basta considerar dois *bytes* consecutivos na memória como um único número para se poder chegar a 65535.

Hexadecimal

Pelo que acabamos de ver, para se representar um número, empregam-se dois *bytes* consecutivos, considerando cada informação como sendo composta por 16 *bits*. Como se torna, porém, pouco prático escrever 16 *bits* em fila para representar um número, introduziu-se o sistema de numeração de base 16 (hexadecimal), análogo ao nosso sistema décimal com a única diferença de que além dos 10 símbolos usuais se empregam outros seis: A=10 B=11 C=12 D=13 E=14 F=15. Portanto, o número 20 escreve-se 14 em hexadecimal ou então 14 (decimal) escreve-se simplesmente E.
O sistema hexadecimal é cómodo: é certamente mais fácil ler D3 (dê –3) que 11010011 (um--um-zero-um-zero-zero--um-um). É necessário evitar ler os números hexadecimais como se fossem decimais. Por isso, 20 (hex) lê-se dois-zero e não 20, uma vez que efectivamente vale 32.

Notas históricas sobre o nascimento dos computadores

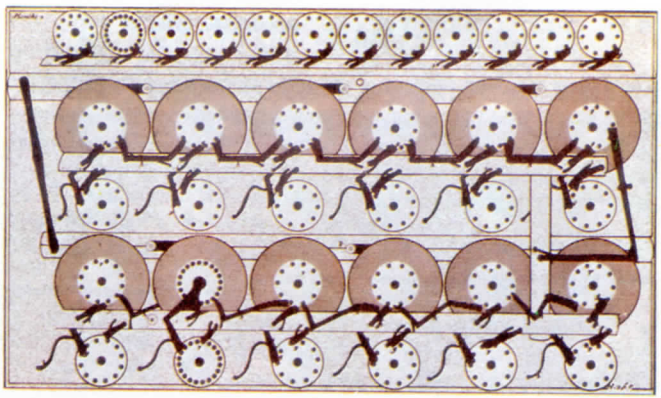
Analisámos sucintamente o funcionamento e o papel da CPU num computador. Fizemo-lo de maneira a justificar as suas singulares características (a aritmética binária, os *bus*...), mas não explicámos como é que se chegou a semelhante estrutura. E menos ainda esclarecemos o papel da lógica. Com efeito, historicamente, a lógica



O ábaco pode ser considerado a primeira máquina de calcular.

matemática codificou os princípios do cálculo matemático muito antes de o processo científico e tecnológico terem tornado possível a construção de computadores. A «ideia» de computador já era conhecida muito antes de estes, na acepção usual do termo, terem sido construídos. Vamos recuar no tempo.

O homem sentiu desde sempre a necessidade de efectuar cálculos, de maneira exacta e rápida: no princípio, para prever os fenómenos celestes, depois para determinar as rotas marítimas, em seguida para efectuar as complicadas operações ligadas ao incremento do tráfico comercial. Pensa-se que a primeira máquina de calcular inventada pelo homem foi o ábaco: as famosas bolinhas enfiadas em paus, ainda hoje usadas nalguns países. A ideia de base consiste em levar além do número 10 as possibilidades de cálculo proporcionadas pelos dedos. Contudo foi necessário esperar pelos primeiros anos do século XVII para se chegar à primeira autêntica calculadora mecânica, obra do matemático francês Pascal. Embora a Pascalina (era este o seu nome) apenas fosse capaz de efectuar adições e subtrações, constituíu para todos os efeitos o primeiro modelo de um computador: fornecendo-se uma informação de entrada, obtinha-se, mediante complicadas rotações das engrenagens, um resultado de saída. E embora se estivesse ainda muito longe do actual conceito de computador era, sem dúvida, a primeira resposta dada pelo homem à necessidade de automatizar os cálculos. Faltava ainda o conceito de sequência de ordens e, consequentemente, de controlo. O passo em frente decisivo foi obra do lógico inglês Charles Babagge que, em 1834, inventou a Máquina Analítica. Na máquina analítica, baseada num programa realizado sobre uma fita de papel perfurado (concebido por Jacquard para a programação dos teares), podemos reconhecer já as partes fundamentais dos modernos computadores.

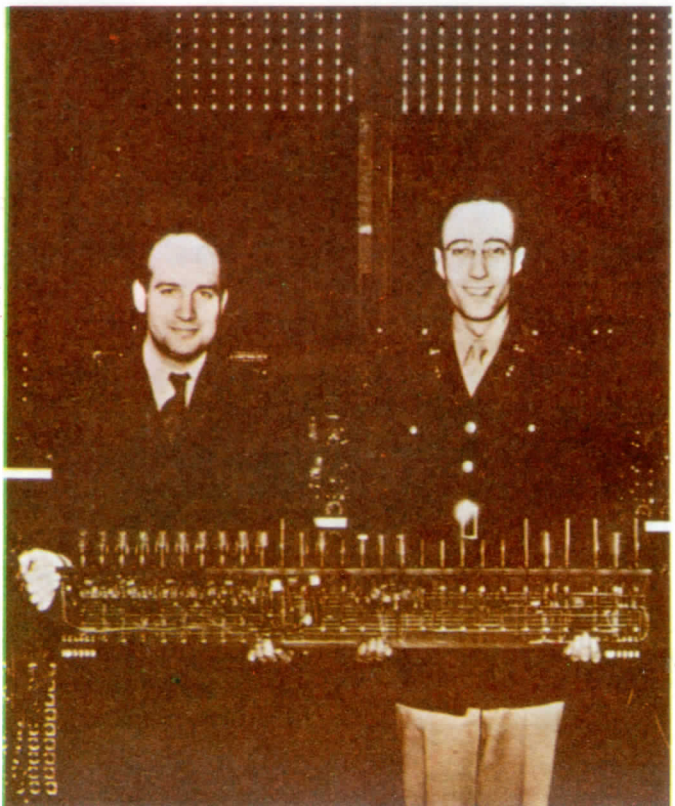


A Pascalina automatizou pela primeira vez a operação de achar o resto.

Existia a unidade de cálculo, chamada fábrica ou moinho, onde se realizavam os cálculos, e existia uma memória central, chamada armazém. Lady Ada Lavelace, amiga de Babagge e sua colaboradora, afirmou a esse respeito: «podemos dizer que a Máquina Analítica tece desenhos algébricos, tal como o tear de Jacquard tece flores e folhas.»
Devido à sua actividade, Lady Lavelace é considerada a primeira programadora da História.

No entanto, Babagge foi vencido pela incipiente tecnologia de então e apenas conseguiu construir parcialmente a sua Máquina Analítica. Da fita perfurada passou-se com Holerith (o fundador da IBM) aos cartões perfurados. Mas para se obter o primeiro verdadeiro computador, foi preciso esperar até 1946, ano em que nasceu o ENIAC: Calculadora e Integradora Numérica Electrónica. Pela primeira vez a electrónica com válvulas entrava no computador e a partir de então as suas histórias passariam a caminhar juntas. Apesar dos milhares de válvulas, que precisavam de ser substituídas de seis em seis horas sob pena de se queimarem (aqueciam ao ponto de serem necessários sistemas de refrigeração), não se conseguiam

Goldstine e Eckert com uma unidade de válvulas do ENIAC.



velocidades de cálculo elevadas. (Imagina que o teu *Spectrum* é muito mais potente, veloz e versátil do que o ENIAC, embora este último ocupasse toda a área de um ginásio). Em 1948, J. Bardeen, W. Brattain e W. Schockley anunciam a descoberta de um componente fundamental: o transístor. O transístor desempenha todas as funções de uma válvula, mas com numerosas vantagens: ocupa menos espaço, dissipa pouco calor, é muito rápido e quase eterno. A sua aplicação aos computadores era inevitável, dando origem à chamada "nova geração". Em seguida, graças ao seu menor custo, os computadores sofreram uma grande difusão e principiou então a batalha dos serviços, uma luta incruenta pela melhoria das características.
Depois do transístor, a electrónica concentrou todos os seus esforços na integração: miniaturização e compactação dos componentes básicos (entre eles o citado transístor) num espaço cada vez mais reduzido. Reduzir os componentes a dimensões apenas visíveis ao microscópio não é coisa de pouca monta e desde 1958 (ano em que foi construído o primeiro circuito integrado) até hoje o percurso está assinalado por sucessivos melhoramentos:

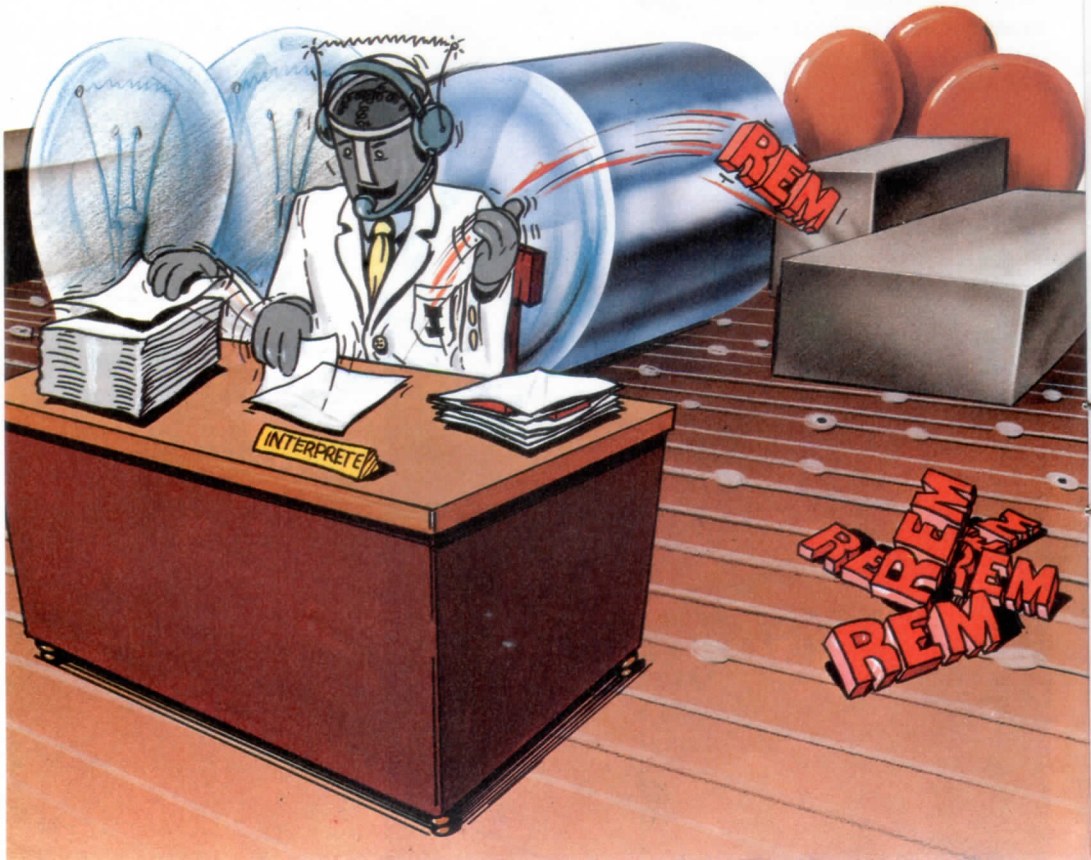
- anos 60, SSI (integração em Pequena Escala), 12 portas lógicas por circuíto integrado;
- final dos anos 60, MSI (Integração em Escala Média), 100 portas lógicas por circuito integrado;
- anos 70, LSI (Integração em Grande Escala), 1000 portas lógicas por circuito integrado;
- fim dos anos 70 VLSI (Integração em Muito Grande Escala), para lá de 50.000 portas lógicas por circuito integrado.

Actualmente numa área equivalente a um cm^2 podem colocar-se cem mil transístores.
A história do computador e da electrónica está, no entanto, muito longe de se esgotar; de facto, calculou-se que o progresso no campo da integração ainda não atingiu o seu nível máximo e há que esperar uma nova «revolução» neste campo.
O teu *Spectrum* é filho desta tecnologia e sem ela não seria capaz de realizar, num espaço tão reduzido e a baixos custos, todas as funções que irás aprender.

LINGUAGEM

REM

A REM (abreviatura do termo inglês *Remark*, «comentário»), é uma instrução que... parece não servir para nada! Efectivamente, quando o teu *Spectrum* encontra uma REM, ignora-a por completo e passa a executar a linha seguinte do programa. Na realidade, a REM é uma instrução importante: permite-te incluir comentários, títulos e descrições sem que eles alterem o processamento normal do programa. O seu argumento pode ser uma sequência qualquer de caracteres ou de símbolos gráficos e matemáticos e até uma linha vazia. Podes introduzir uma REM em qualquer parte do programa ou no final de cada linha de instruções, bastando respeitar as regras sintáticas do teu *Spectrum*. A seguir à REM não podes introduzir nenhuma declaração ou ordem, pois seria tudo considerado como um comentário.



Exemplos:

```
10 REM "GEOMETRIA 1"
```

Esta linha dá ao teu programa o título «GEOMETRIA 1».

Quando, ao fim de um certo tempo, voltares a usá-lo, não precisarás de analisar as instruções para saberes se o programa em questão é uma contabilidade doméstica ou um arquivo de discos. Tens a resposta na primeira linha do programa.

```
40 ...
50 ...
60 ...
70 INPUT B
80 REM CÁLCULO DA MÉDIA
```

Comunica-te que as linhas que se seguem à REM servem para calcular a média.

```
 90 LET M=(B+C+F+G)/4
100 ...
110 ...
120 ...
```

Quando necessitares de calcular a média de vários números para outro programa, saberás onde encontrar as instruções necessárias sem precisares de as escrever outra vez. Ou então, se, em vez da média, precisares de achar a raiz quadrada desses números, poderás, muito simplesmente, substituir as linhas que se seguem à REM, sem teres, para isso, de voltar a analisar e modificar todo o programa.

```
30 REM "GEOMETRIA":PRINT GEOMETRIA
```

O teu *Spectrum* interpreta todo o argumento de REM ("GEOMETRIA": PRINT GEOMETRIA) como um comentário e não executará nunca

a instrução contida (PRINT).

Sintaxe da instrução

REM qualquer sequência de caracteres

GOTO

Quando falámos do algoritmo, deves ter visto que este contém todas as informações necessárias para levar a cabo uma elaboração. Porém o algoritmo não precisa forçosamente de se desenvolver sequencialmente (uma instrução após a outra), podendo dar «saltos», para poder modificar, segundo as necessidades, a ordem das instruções a executar.

LINGUAGEM

O BASIC possui uma instrução que efectua esta tarefa: GOTO.
A instrução GOTO (passa para...) é sempre seguida por um número em forma explícita (ou sob a forma de uma variável ou de uma expressão). Na prática, quando o teu *Spectrum* encontra uma instrução GOTO, em vez de continuar com a instrução seguinte, salta para a indicada, continuando a partir dela como se nada se tivesse passado. (Mas atenção, porque se esta última não existir, o intérprete de BASIC enviar-te-á a mensagem de erro <UNDEF STATEMENT>, precisamente para te avisar de que essa linha não existe.)
Em resumo, GOTO permite-te modificares a ordem de execução de acordo com as tuas necessidades. Um exemplo claro é o cálculo da tabela dos quadrados.

```
 10 REM CALCULA OS QUADRADOS DOS NÚMEROS
 20 REM A PARTIR DO 1
 30 REM ATÉ QUE SE INTERROMPA
 40 LET NÚMERO = 0
 50 REM ACRESCENTA 1 EM CADA ETAPA
 60 LET NÚMERO = NÚMERO + 1
 70 REM IMPRIME O NÚMERO E O SEU QUADRADO
 80 PRINT NÚMERO, NÚMERO ↑ 2
 90 REM REPETE INCREMENTADO
100 GOTO 50
```

GOTO é também chamada instrução de transferência incondicional, uma vez que ordena ao teu *Spectrum* que salte, seja qual for o caso, para linha indicada. Por exemplo, observa o programa que se segue:

```
10 GOTO 40
20 ...
30 ...
40 REM
```

O programa principia sempre na linha 40, saltando-se as linhas 20

e 30. Esta é a sua aplicação mais comum, mas existem alguns casos especiais. Um deles é:

```
90 GOTO 90
```

ou, em geral:

```
90 uma instrução qualquer: GOTO 90
```

A primeira é usada em casos particulares para interromper o fluxo do programa sem que, na realidade, ele pare: a calculadora efectua um número infinito de saltos na linha 90. A segunda é equivalente a 90 GOTO 90, com a única diferença de que o teu *Spectrum* executa um número infinito de vezes as ordens especificadas antes da instrução GOTO. Em qualquer dos casos, ambas servem para criar programas que nunca se interrompem a não ser através de uma ordem exterior. Para te convenceres faz a experiência com este programa:

```
10 REM PROGRAMA QUE SE INTERROMPE
20 REM SEM PARAR
30 PRINT "ESTOU NA LINHA 30"
40 PRINT "ESTOU NA LINHA 40":GOTO 40
```

Exemplos

```
30 ...
40 ...
50 ...
60 GOTO 100+(ÂNGULO * 4)
```

Como já se disse, o argumento de GOTO pode ser uma expressão qualquer que dê um número como resultado. Neste caso, se ÂNGULO = 100, quando o programa chegar à linha 60, saltará para a 100 + (100 * 4) = 500.

```
30 ...
40 ...
50 ...
60 GOTO A
```

O argumento de GOTO também pode ser uma variável numérica. Quando A vale 100, GOTO terá que saltar a execução do programa até à linha 100. Se esta não existir, até à linha que se lhe segue imediatamente.

```
 70 LET A=100
 80 GOTO A
 90 REM
105 PRINT A
```

Uma vez que falta a linha 100, a execução prossegue a partir da que se lhe segue imediatamente: no exemplo, a linha 105.

```
10 REM É UM QUADRADO OU UM LOSANGULO?
20 INPUT "NÚMERO DE LADOS IGUAIS"; L
30 INPUT "NÚMERO DE ÂNGULOS IGUAIS"; A
40 GOTO 20
50 IF L=A THEN PRINT "É UM QUADRADO"
60 PRINT "NÃO É UM QUADRADO"
```

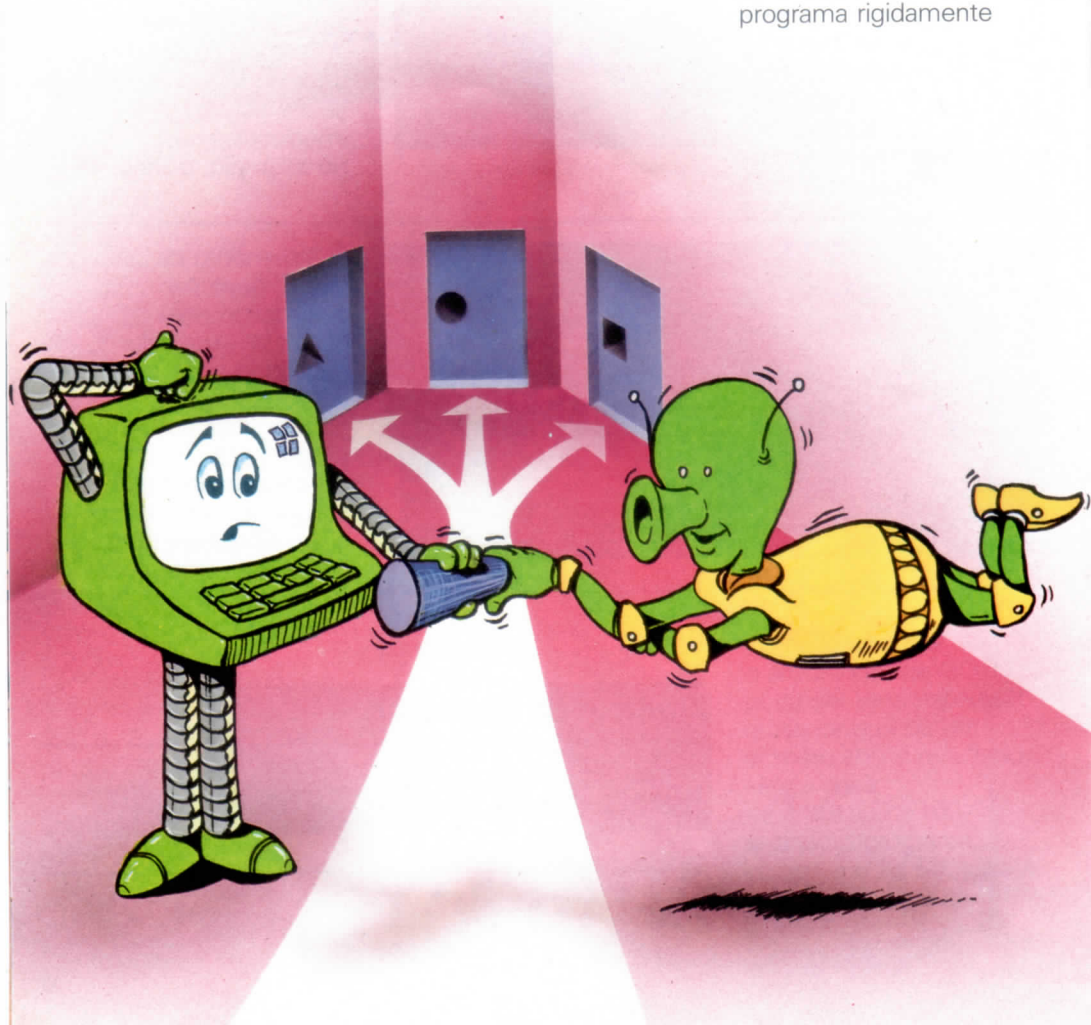
O argumento da instrução GOTO deve ser cuidadosamente escolhido: um valor enganado pode ter consequências catastróficas no teu programa.
Se considerares este programa, verás que o teu *Spectrum* nunca chegará às instruções das linhas 50 e 60.

Sintaxe da instrução

GOTO número / expressão numérica / variável

IF THEN

Como iremos ter ocasião de particularizar mais adiante (quando retomarmos o tema do Diagrama de fluxo), uma parte fundamental da programação ocupa-se das decisões. É graças a elas que conseguimos adaptar uma solução geral (o programa) à situação particular (o problema a resolver). Sem a existência de alguma coisa que permita ao programa modificar o seu curso, com base no contexto, muitas das aplicações do computador não seriam possíveis: teríamos de escrever um programa rigidamente



sequencial. Por outro lado, as decisões desempenham um papel fundamental na nossa vida: acontece, porém, que subvalorizamos a sua importância porque desde sempre estamos habituados a tomá-las.
O computador, continua a ser importante sublinhá-lo, é um executante formidável ao qual é necessário dizer tudo o que tem de fazer (até as coisas mais óbvias), visto que carece de raciocínio e de iniciativa próprios.
Por isso, a decisão é uma das pedras angulares de programação. O que é que um computador pode decidir?
Quando falámos da CPU, dissemos que o computador apenas analisa números e sabe dizer se dois números são iguais ou diferentes.
Para reduzir qualquer decisão a uma forma «compreensível» para o computador é, pois, necessário reduzir o problema a números.
Se número 1 = número 2, então escreverá «os dois números são iguais».
Esta é a forma computável e, generalizando, em BASIC escreve-se:

```
IF CONDIÇÃO LÓGICA THEN INSTRUÇÃO
```

Supõe que atribuíste à variável X um número secreto e que pretendes que um amigo o adivinhe inserindo na variável T a tentativa de resolução.
Necessitas de uma instrução que ordene ao teu *Spectrum* que assinale quando o número secreto tiver sido descoberto, ou seja, quando a variável T for igual à variável X.

```
SE T=X ENTÃO IMPRIME "OK"
```

Bem, não deverias ter muita dificuldade em escrever uma instrução deste estilo, pois já o fizeste antes...

Basta traduzires para inglês a instrução, tal como a pensaste e terás:

```
IF T=X THEN PRINT "OK"
```

É evidente a importância de IF... THEN...: serve para o computador efectuar autênticas escolhas, baseado nas quais o programa possa executar, em cada caso, a função mais adequada a uma circunstância particular. É um pouco como dotar o teu *Spectrum* com a capacidade de discernir, de o tornar quase um «cérebro». Naturalmente, em lugar de PRINT, podes escrever qualquer outra instrução, assim como podes igualmente indicar comparações (maior ou menor) de valores numéricos ou alfanuméricos. No caso da condição não ser certa, a continuação da linha é completamente ignorada e o controlo passa para a seguinte. Não esqueças, pois, que tudo o que se encontra a seguir ao THEN está sujeito a comparação. Esta possibilidade particular proporciona notáveis vantagens: permite-te escrever grupos completos de instruções controlados por um único IF, poupando GOTO supérfluos e obtendo programas compactos, velozes e muito legíveis. Mesmo assim, acontecer-te-á com frequência encontrares IF... THEN GOTO, uma vez que a combinação destas duas instruções gera autênticas ramificações no programa, muito semelhantes a ramais ferroviários, os quais conduzem para um ou outro caminho, conforme a sua colocação. Vejamos uma aplicação simples mas útil de IF THEM GOTO, retomando um dos exemplos já analisados (90 GOTO 90). Introduzamos-lhe uma pequena modificação:

```
80 PAUSA=0
90 PAUSA=PAUSA+1: IF PAUSA<1000 THEN GOTO 90
```

Obtemos um *timer* (temporizador): o teu computador conta até 1000 (neste caso) antes de continuar com a instrução seguinte.

Este pode ser um recurso válido para quando precisares de retardar a execução de algumas instruções. No caso de teres de comparar valores alfanuméricos, o computadr atribuirá a cada letra do alfabeto, símbolo gráfico, etc., um determinado número de acordo com um padrão definido. Este código é proporcional à posição alfabética do carácter: por exemplo, A corresponde a 65, B a 66, C a 67... e assim sucessivamente. Agora a comparação é simplicíssima, reduzindo-se ao confronto dos códigos que constituem as duas cadeias. A seguir a uma declaração IF THEN podes omitir a palavra reservada GOTO, indicando unicamente o número de linha a saltar. O resultado será o mesmo: o salto para a linha indicada.

Exemplos:

```
10 PRINT "QUANTO DINHEIRO TENS NO
   BOLSO?"
20 INPUT DINHEIRO
30 IF DINHEIRO≥3000 THEN PRINT
   "ÉS RICO!"
40 IF DINHEIRO<3000 THEN PRINT
   "NÃO TENS MUITO!"
50 IF DINHEIRO=O THEN PRINT
   "NÃO TENS NADA!"
60 $TOP
```

Analisemos o programa anexo: se uma das três expressões for verdadeira, o programa imprime o comentário apropriado. Se tivesses 500 escudos, o resultado seria: "NÃO TENS MUITO!", uma vez que a segunda instrução IF é verdadeira.

```
70 IF TV=30 THEN LET RÁDIO=24
```

Se a variável TV vale 30, então a variável RÁDIO toma o valor 24.

```
70 IF NÚMERO=88 THEN IF OUTRO NÚMERO
   =66 THEN GOTO 90
```

A ordem IF THEN tem algumas características interessantes como a possibilidade de juntar sequencialmente mais de uma instrução. Este processo chama-se concatenação, visto reagrupar (concatenar) instruções distintas como se fossem parte de uma única instrução.

No exemplo, se o número for 88, o computador verifica se o outro é 66 antes de passar à linha 90. Virtualmente não existe limite para a concatenação de IF THEN; no entanto, convém não abusar para evitar problemas de correcção (DEBUGGING).

```
IF RODAS=4 AND VEÍCULO=1 THEN PRINT
"É UM CARRO"
```

IF verifica se uma operação lógica é verdadeira: podes então introduzir operadores lógicos que te permitam ampliar o controlo de IF a mais de uma expressão, tal como se fez no exemplo. A expressão RODAS=4 AND VEÍCULO=1, é verdadeira unicamente quando ambas as igualdades se verificam e apenas neste caso o computador imprime o comentário apropriado.

```
40 IF DINHEIRO<100 THEN PRINT
   "NÃO TENS MUITO!"
```

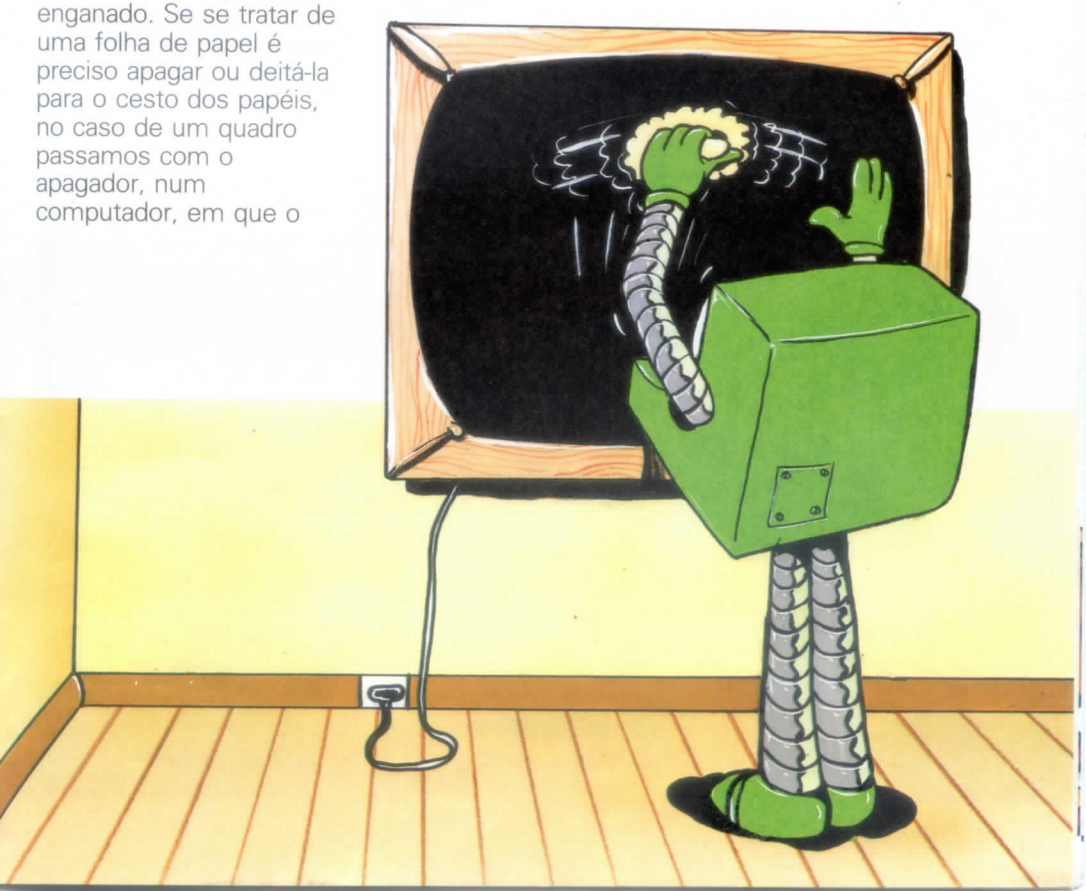
Deves dar atenção à selecção do argumento de IF. Se no exemplo "QUANTO DINHEIRO TENS NO BOLSO?", pões na linha 40 e dizes que tens 2500, o computador, como poderás verificar, não te responderá nada!

Sintaxe da instrução

IF expressão lógica THEN instrução exequível

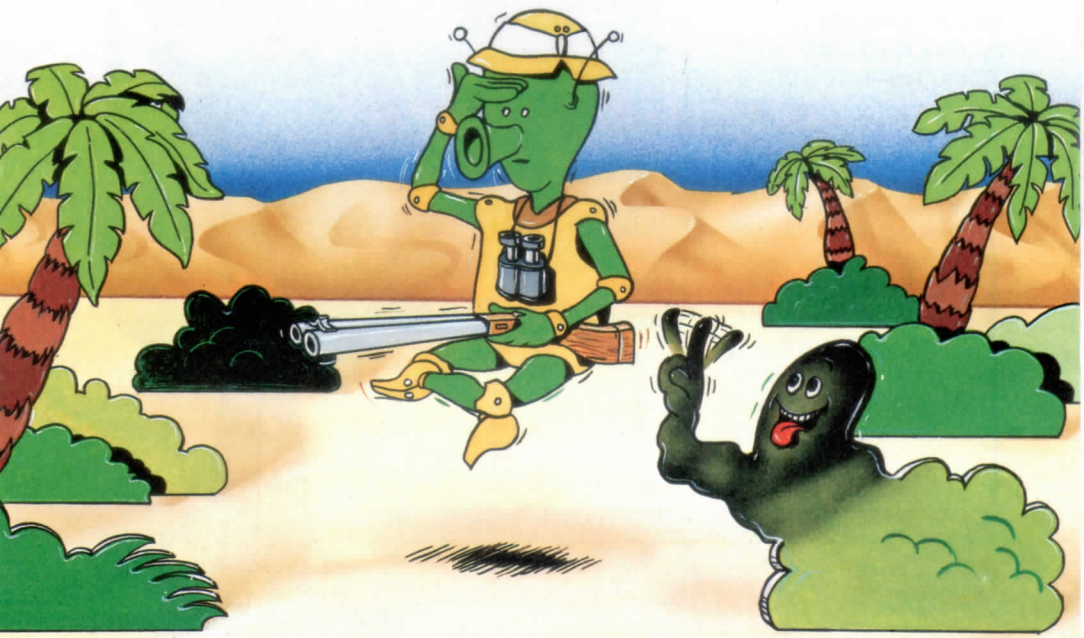
CLS

Acontece frequentemente termos de escrever o que dizemos quer seja para nos lembrarmos de alguma coisa quer seja por precisarmos de a ensinar a outros. Com a mesma frequência vemo-nos na necessidade de apagar o que escrevemos, porque já não tem utilidade ou porque está enganado. Se se tratar de uma folha de papel é preciso apagar ou deitá-la para o cesto dos papéis, no caso de um quadro passamos com o apagador, num computador, em que o ecrã de TV funciona como uma «folha electrónica», basta indicar a ordem adequada. No nosso caso CLS, do inglês *Clear Screen* ("Limpa o ecrã"). Com estas três letras apenas podes esvaziar a teu gosto o ecrã de tudo o que lá está. O teu *Spectrum* considera o ecrã como um conjunto de inúmeros pequenos pontos: ser-te-á fácil vê-los aproximando-te do ecrã depois de teres escrito alguma coisa. Quando o computador encontra a instrução CLS, "pinta" esses pontos com a mesma cor do fundo (na prática apaga-os) e coloca o cursor no canto superior esquerdo. Se um programa te exigir que visualizes muitas coisas, não poupes os CLS: os teus olhos agradecer-to-ão.



Como escrever programas: técnicas de clareza e de correcção

Já vimos alguns programas simples, porém quais são os requisitos necessários para escrever um bom programa? Normalmente a um programa «ideal», pedimos que execute todas as operações necessárias no menor lapso de tempo possível, que seja facilmente modificável e corrigível e que ocupe a menor quantidade possível de memória. Porém, excepto em casos particulares, não podemos variar grandemente a velocidade de cálculo e, com efeito, o único parâmetro sobre o qual podemos influir significativamente continua a ser a depuração da listagem. De facto, a tarefa mais ingrata na redacção de um programa é normalmente a de detectar e corrigir os erros cometidos, operação a que no calão do ramo se chama *debugging*, que, traduzindo, significa literalmente «expurgar». O problema torna-se mais grave quando a linguagem proporciona muitas (demasiadas) possibilidades de abordagem do problema, como acontece no BASIC. Muitas vezes, ignorando-se em que zona do programa se esconde o erro, é-se obrigado a analisar a listagem do



princípio até ao fim, com grande gasto de tempo e trabalho, precisamente porque as variáveis e as instruções se encontram desordenadamente colocadas nela. O primeiro passo para melhorar a redacção dos programas consiste em estabelecer uma ordem interna: criar uma estrutura. Deste modo, será uma boa norma incluir em primeiro lugar as variáveis, de maneira a estarem imediatamente disponíveis para a sua verificação e/ou correcção. Procedendo assim, entre outras coisas, aumentaremos ligeiramente a velocidade de execução do programa, uma vez que no seu processamento o computador apenas lerá instruções «activas», que consistem em operações que conduzem à elaboração dos dados em exame, memorizando-se todas as variáveis no início. Outra dificuldade que se encontra durante a redacção de um programa é a sua abstracção. Quase todas as linguagens existentes estão ligadas, mais ou menos pronunciadamente, à maneira como a máquina elabora as informações que lhe proporcionamos. E esta «lógica» está muito distante da nossa maneira de actuar e de pensar, obrigando-nos a «traduzir» a nossa lógica para a do computador, trabalho que não é fácil de todo.

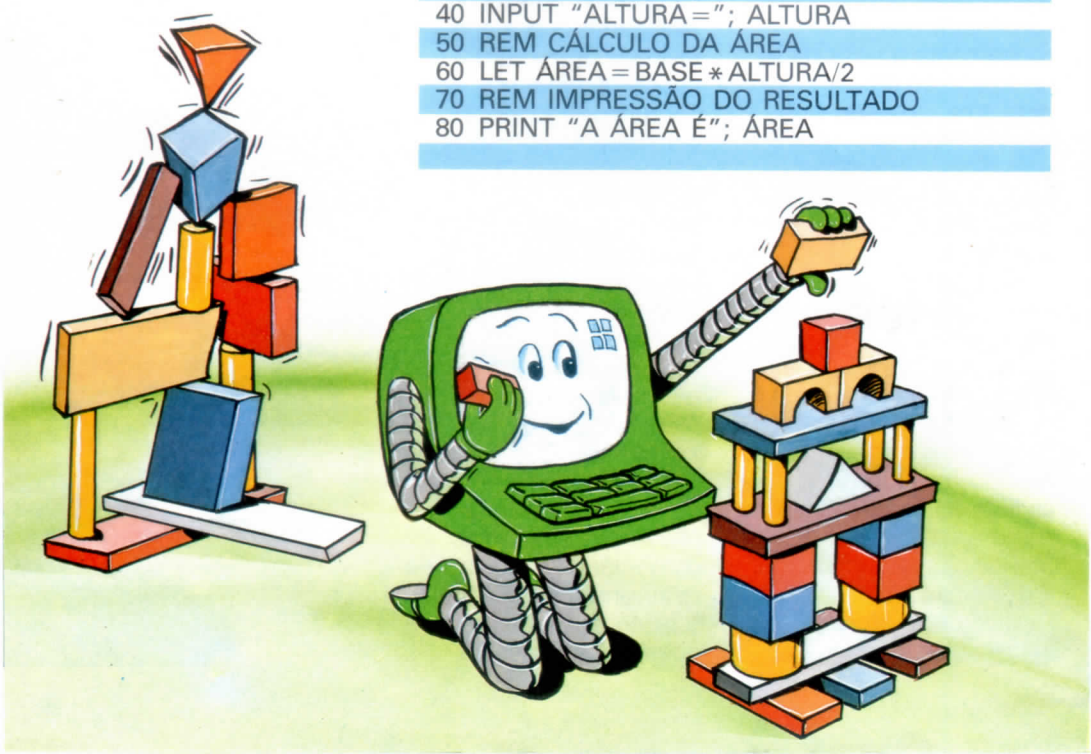
Por isso é aconselhável adoptar algumas precauções a fim de tornar os programas menos abstractos e formais e consequentemente mais acessíveis aos seus utilizadores. Uma primeira técnica consiste em empregar na definição de uma variável nomes compostos que recordem facilmente qual a sua utilidade. Por exemplo, para programar o Teorema de Pitágoras, poderias usar como variáveis CATETO 1, CATETO 2, HIPOTENUSA; a, b, c também seriam nomes correctos. Embora nem todos os computadores prevejam este tipo de variáveis, o teu *Spectrum*, felizmente, é capaz de reconhecer variáveis compostas. Outro tipo de estratégia a adoptar é a de incluir nos pontos-chave dos programas comentários e indicações relativamente áquilo que o bloco de instruções em questão se destina a realizar. Desta forma, mesmo passado tempo, bastar-te-á um relance para identificares as diferentes partes do programa, ganhando em clareza e possibilidades de modificação.

Em BASIC esta possibilidade é-te proporcionada pelo comando REM, que permite escrever qualquer texto sem que o programa o tenha em consideração. Outro

requisito essencial para a clareza de um programa é a sua linearidade, ou seja, que o programa se processe sequencialmente, uma linha a seguir à outra, com o menor número possível de «saltos». Na linguagem BASIC, a instrução «salto para uma linha» é GOTO e deverás procurar usá-la o menos possível, para facilitares ao máximo o *debugging*.

Se o programa escrito e concebido por ti previr diferentes possibilidades de cálculo, segundo as necessidades de quem o use, será conveniente dividir, em seguida, as suas diversas possibilidades, separando-as em módulos distintos no interior da listagem. E, por exigências de clareza, é sempre conveniente que as linhas do programa não sejam demasiado longas, para evitar complicações de leitura. Resumindo todos os conceitos expostos, podemos agora escrever um programa BASIC que se aproxime mais das nossas exigências. Retomemos, como exemplo, o programa da área de um triângulo. Começaremos por pôr como primeira linha da listagem uma REM que nos explique a finalidade do programa. Neste ponto obteremos:

```
10 REM A ÁREA DE UM TRIÂNGULO
20 REM INTRODUÇÃO DOS DADOS
30 INPUT "BASE="; BASE
40 INPUT "ALTURA="; ALTURA
50 REM CÁLCULO DA ÁREA
60 LET ÁREA=BASE*ALTURA/2
70 REM IMPRESSÃO DO RESULTADO
80 PRINT "A ÁREA É"; ÁREA
```



Como deves ter notado, as variáveis usadas têm designações extensas que declaram explicitamente o seu conteúdo. Uma leitura do programa poderia ser:

```
10 CABEÇALHO
20 ÷ 40 MÓDULO DE ENTRADA DE DADOS
50 ÷ 60 MÓDULO DE ELABORAÇÃO
60 ÷ 80 MÓDULO DE SAÍDA DE DADOS
```

Nas primeiras vezes parecer-te-á artificioso programar por estruturas, mas ao fim de pouco tempo apreciarás as vantagens, a clareza das listagens, a economia de tempo e de cansaço durante e após a programação. O conjunto de regras que permitem modificar um programa acrescentando e/ou corrigindo as suas partes, chama-se EDITOR (algo semelhante a «corrector» ou «supervisor»). Cada computador – e o teu *Spectrum* também – tem um programa editor que permite efectuar estas modificações: poderás informar-te acerca das suas possibilidades lendo o manual de instruções. Em seguida será útil apropriares-te destes instrumentos de correcção a fim de economizar esforços inúteis durante e após a correcção.

As decisões

Agora tens de aprender a comunicar com o computador, ainda que seja apenas para realizar algumas simples operações aritméticas. O seu valor e as suas capacidades ficariam, porém, limitadas se se limitassem exclusivamente a reproduzir modelos já pré-estabelecidos. Efectivamente, em seguida poderemos falar de «inteligências artificiais» precisamente porque os computadores «decidem» o que fazer (relativamente à realização das tarefas que lhes encomendamos através dos programas) com base em critérios lógicos e de valor – isso sim! – anteriormente estabelecidos. Em BASIC, as decisões executam-se empregando a instrução IF... THEN, que permite ao teu *Spectrum* proporcionar-te o resultado de um TESTE e executar imediatamente uma instrução em vez de

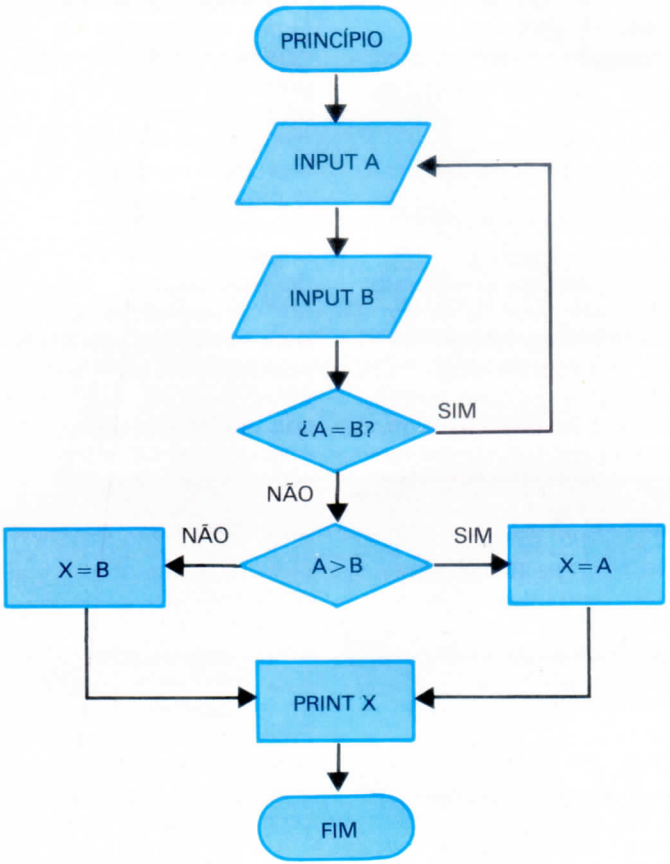
outra. Apresentamos agora um primeiro e simples problema para ver como podes implementar um programa no qual o computadr realiza uma «escolha» com base nos «conhecimentos» que guarda na memória.

Objectivo

Queremos, por exemplo, que o nosso computador determine qual de dois números quaisquer, que designamos A e B, é o maior. Pretendemos portanto que o computador atribua a uma incógnita, a que chamamos X, o valor mais alto.

Procedimento

Para resolver este problema, primeiro é preciso comparar A e B. Se A>B, então põe-se o valor A em X, de outra maneira, no X por-se-ia o valor de B. Este modo de proceder na resolução pode ser representado graças ao fluxograma que contém dois «losangos» nos quais se encontra a comparação entre A e B, e dois «rectângulos», que correspondem às



Pede o primeiro número

Pede o segundo número

Se são iguais volta atrás, se não...

Verifica qual dos dois é o maior

...e imprime-o

«atribuições ou acções». A listagem que se segue corresponde à sua sequência em BASIC.

Outro exemplo de problema resolúvel mediante um programa pode ser o seguinte:

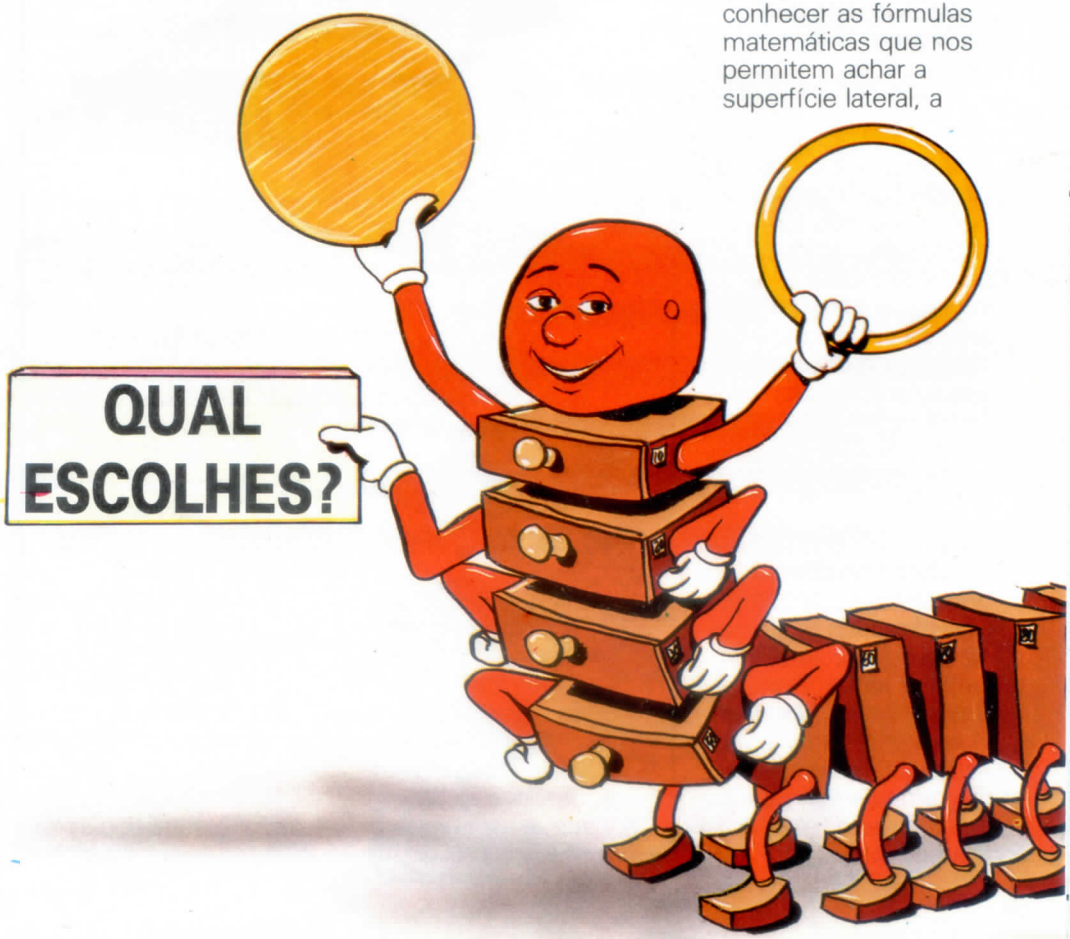
```
10 REM O NÚMERO MAIOR ENTRE 2
20 INPUT "PRIMEIRO NÚMERO"=; A
30 INPUT "SEGUNDO NÚMERO="; B
40 IF A=B THEN GOTO 20
50 IF A>B THEN LET X=A
60 IF A<B THEN LET X=B
70 PRINT "O NÚMERO MAIOR É"; X
```

Objectivo

Queremos determinar a superfície lateral, a superfície total e o volume de um cilindro de qualquer dimensão.

Procedimento

Para podermos resolver este problema é preciso conhecer as fórmulas matemáticas que nos permitem achar a superfície lateral, a



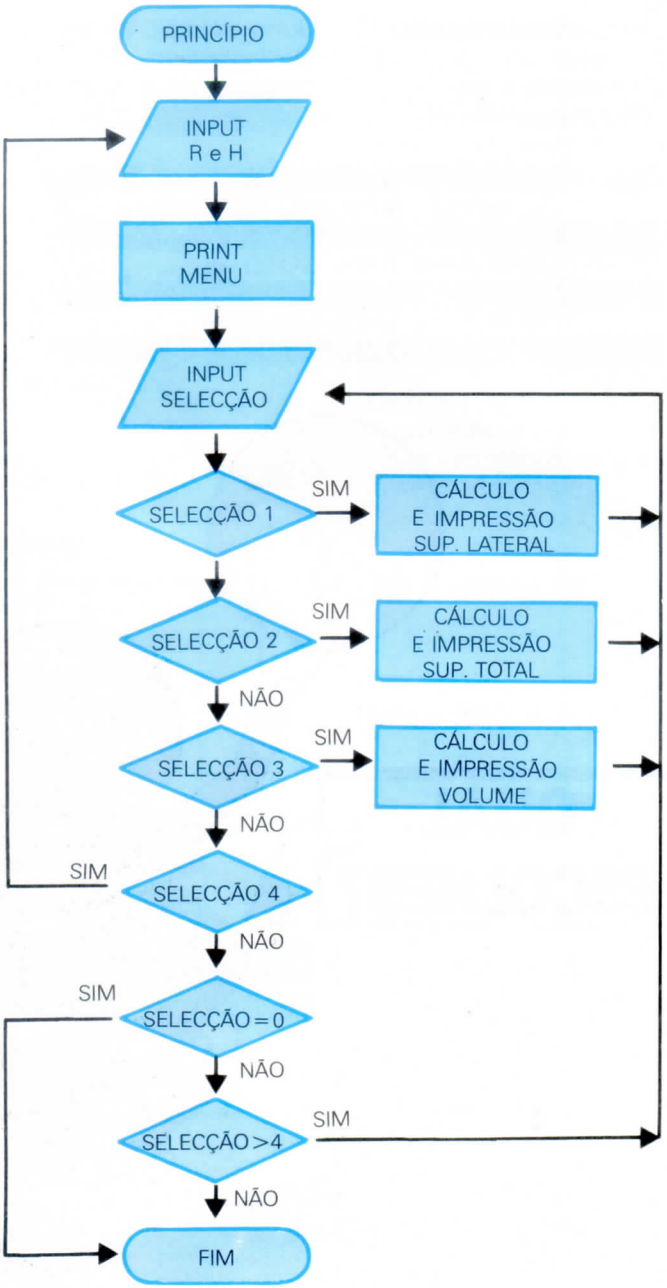
superfície total e o volume de um cilindro. Vejamo-las juntos.

a) A superfície lateral do cilindro (SLAT) obtém-se multiplicando a altura pela circunferência da base, recordando que esta última é o resultado da seguinte operação: raio X 2 X 3, 14. Expressão esta que, na linguagem do teu *Spectrum* é: R * 2 * PI.

b) A superfície total do cilindro (SOT) obtém-se adicionando as superfícies das duas bases circulares à superfície lateral. A área de um círculo obtém-se mediante a seguinte operação: raio X raio X 3, 14. Expressão que corresponde a R ↑ 2 * PI.

c) O volume do cilindro (VOLUME) obtém-se multiplicando a área do círculo da base pela altura = R * R * PI * H.

É bastante simples passar ao diagrama de fluxo e deste à listagem BASIC correspondente.



```
 10 REM SUPERFÍCIE E VOLUME DE UM CILINDRO
 20 REM INTRODUÇÃO DADOS DO CILINDRO
 30 INPUT "RAIO="; R
 40 INPUT "ALTURA="; H
 50 REM IMPRIME O MENU
 60 PRINT PRIME A TECLA
 70 PRINT "1 PARA SUPERFÍCIE LATERAL"
 80 PRINT "2 PARA SUPERFÍCIE TOTAL"
 90 PRINT "3 PARA VOLUME"
100 PRINT "4 PARA NOVOS DADOS"
110 PRINT "O PARA TERMINAR"
120 REM SELECÇÃO
130 INPUT "SELECCIONO"; SELECÇÃO
140 IF SELECÇÃO=1 THEN GOTO 200
150 IF SELECÇÃO=2 THEN GOTO 300
160 IF SELECÇÃO=3 THEN GOTO 400
170 IF SELECÇÃO=4 THEN GOTO 20
180 IF SELECÇÃO=0 THEN GOTO 500
190 IF SELECÇÃO>4 THEN GOTO 120
200 REM CÁLCULO E IMPRESSÃO SUPERFÍCIAL LATERAL
210 LET SLAT=R*2*PI*H
220 PRINT "SUPERFÍCIE LATERAL="; SLAT
230 GOTO 120
300 REM CÁLCULO E IMPRESSÃO SUPERFÍCIE TOTAL
310 LET SOT=R*2*PI*H+2*R*R*PI
320 PRINT "SUPERFÍCIE TOTAL="; SOT
330 GOTO 120
400 REM CÁLCULO E IMPRESSÃO VOLUME
410 LET VOLUME=R*R*PI*H
420 PRINT "VOLUME="; VOLUME
430 GOTO 120
500 REM FIM
```

Destes dois simples exemplos depreendem-se já algumas conclusões fundamentais: um computador, graças à sua memória e aos programas que lhe proporcionas, consegue processar as tarefas que lhe atribuíste com muito maior precisão e, sobretudo, maior velocidade do que qualquer pessoa. Um computador pode, além disso, escolher, por exemplo, qual de dois números é o maior, graças ao conjunto de conhecimentos que lhe proporcionas. Finalmente, um computador é capaz de voltar a executar um problema tantas vezes quantas forem necessárias, podendo escolher de novo os dados e informações que deve empregar.

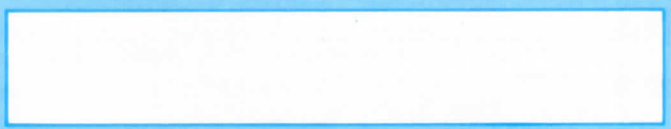
EXERCÍCIOS

Anota nos espaços em branco (a lápis, para poderes apagar) o resultado que prevês para cada exercício proposto e confronta-o depois com as soluções do teu *Spectrum*. Se tiveres cometido um só erro que seja, revê a lição.

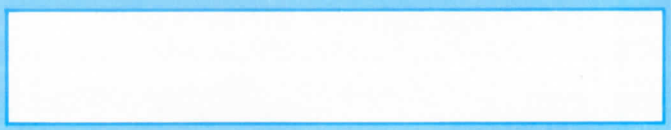
```
10 REM: LET N$="PEDRO"
20 PRINT N$
```

```
10 PRINT "NÃO ACREDITO"
20 PRINT "QUE TU CONSIGAS"
30 CLS
40 PRINT "VER ALGUMA COISA"
```

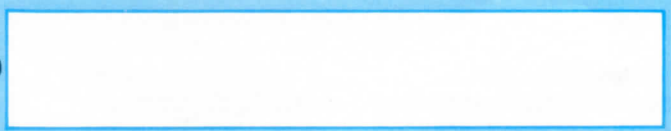
```
10 CLS
20 PRINT "AGORA SIM"
```



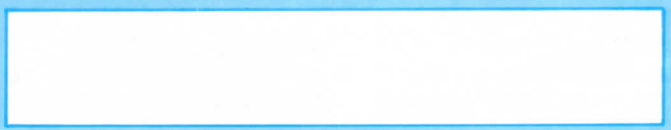
```
10 CLS
20 REM:GOTO 40
30 PRINT "VIDEOBASIC"
40 PRINT "JACKSON"
```



```
10 LEP P=0:LET S=1
20 IF P=1 THEN LET S=0
30 PRINT S, P
```



```
10 LET A=129:LET C=131
20 IF A>C THEN LET C=A
30 IF C>A THEN LET A=C
40 PRINT A, C
```



Regulamento do Concurso de VIDEOBASIC

Para participar neste concurso, deverá enviar um "slogan" criativo que defina o Curso VIDEOBASIC. O "slogan" deverá ser o mais pequeno possível e escrito num postal onde colocará os 5 selos dos fascículos n.os 1 a 5, (os selos recortam-se da contra-capa do VIDEOBASIC de 1 a 5). Serão considerados participantes todos os "slogans" enviados até ao dia 31 de Janeiro de 1986. Pode participar com quantos "slogans" quiser, separadamente, desde que mande cada "slogan" com 5 selos. Todos os "slogans", classificados ou não, passarão a pertencer às Edições Latinas.

Os funcionários das Edições Latinas, não poderão participar no concurso.

Os trabalhos de selecção e classificação serão realizados por uma comissão julgadora, constituída por elementos a designar pelas Edições Latinas, cuja decisão será soberana e irrevogável. Os "slogans" serão seleccionados pela sua criatividade, clareza e adequação à publicação. Serão seleccionados e classificados os 5 melhores "slogans", cabendo a cada criador do "slogan" seleccionado um **TIMEX COMPUTER 2048 PERSONAL COLOR.**

- Escrita automática de instruções de comando
- Verificação imediata de erros
- Memória de 48K (RAM) e 16K (ROM)
- Linguagem integrada de programação BASIC
- Ficha de expansão para periféricos
- Joystick Interface Kempstone
- Saída Monitor Video composto
- Compatível Spectrum

A apuração será efectuada no dia 1 de Março de 1986, na sede das Edições Latinas, Lda., na Av. Almirante Reis, 219, 3.º-Esq. – 1000 Lisboa, onde os prémios se encontram.

O resultado será divulgado nos fascículos do VIDEOBASIC, e os contemplados serão notificados pelas Edições Latinas, Lda.

Enviar o postal a:

EDIÇÕES LATINAS, LDA.
Av. Almirante Reis, 219, 3.º-Esq.
1000 Lisboa

BOLETIM DE ASSINATURA

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

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

Junto envio cheque n.º_____ Banco _____

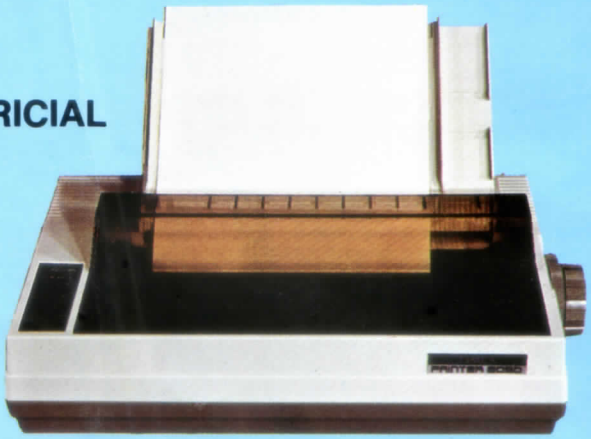
Nome

Morada

Cód. Postal Cidade

(Em vez de cortar o cupão tire uma fotocópia)

TIMEX PRINTER 2080 IMPRESSORA MATRICIAL



- **Ligação a qualquer computador com saída RS 232C.**
- **Papel A-4 ou banda contínua.**
- **Cópias múltiplas (original mais duas cópias).**
- **Capacidades gráficas.**

CARACTERÍSTICAS:

1. ESPECIFICAÇÕES

• Dimensão	390×119×266 mm
• Peso	4.9 Kg
• Alimentação	~ 220 VAC ± 10%, 50 HZ ± 3%
• Consumo	30 W (Auto-teste), 15 W (Repouso)

2. ESPECIFICAÇÕES DE IMPRESSÃO

• Método de impressão	Impacto por matriz de pontos (Bidireccional)
• Cabeça de impressão	9 pontos
• Caracteres	130
• Impressão gráfica	7 categorias (480, 576, 640, 720, 960, 1920 pontos/linha)
• Modos de impressão	Standard Pica 10 CPI Standard Elite 12 CPI Standard Condensed 17 CPI High Qualite Pica 10 CPI High Qualite Elite 12 CPI 7 categorias impressão gráfica

Possível misturar qualquer dos modos descritos na mesma linha.
Ainda possivel os seguintes tipos: Bold, Dupla Passagem, Dupla Largura, Superscript, Subscript, Proportional, Itálico.

• Tipo de papel	Banda contínua Folhas soltas A-4
• Espaçamento entre linhas	Mínimo 1/216''
• Formatos do papel	Largura 112 mm (4'')·até 254 mm (10'') Espessura 52 g/m2 - 76 g/m2 0.07 mm - 0.1 mm
• Cópias múltiplas	Original mais 2 cópias, 40 g/m2 Total da espessura superior 0.2 mm
• Fita	Tipo cassette, cor única (Preto)

3. LIGAÇÕES

Pode ser ligada a qualquer computador com saída RS 232 através de cabo apropriado (i.e. QL, TIMEX FDD, INTERFACE I).

TIMEX printer 2080