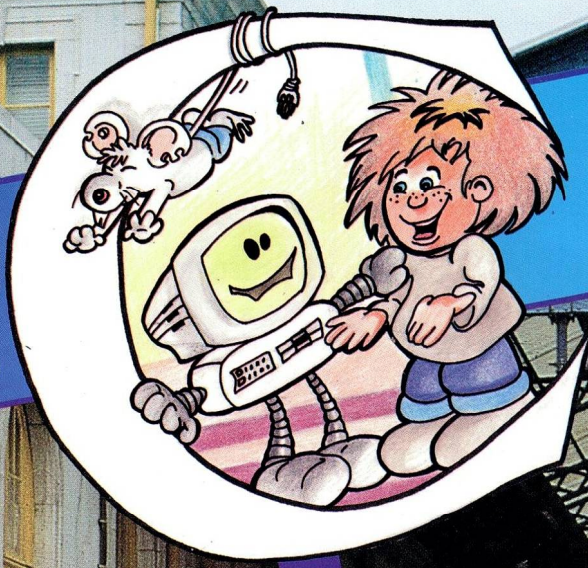
Spooler

MAGAZINE DOS UTILIZADORES PC

Nº 17 ABRIL DE 1992 PREÇO: 500$00

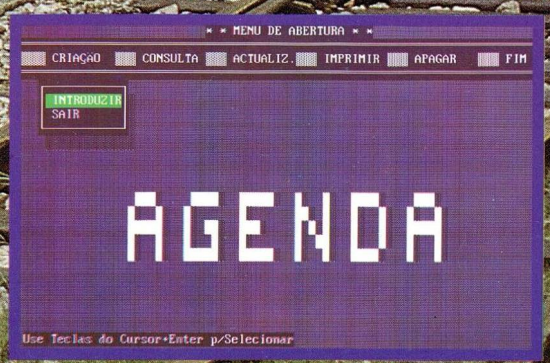
PROGRAMAR EM



ASSEMBLY

CLIPPER

- OS ROEDORES INFORMÁTICOS (III)
- A UTILIZAÇÃO DA MEMÓRIA EXTRA
- UM MICROPROCESSADOR POR SOFTWARE
- A DINASTIA INTEL (I)
- D. O. S. (VI)



AGENDA

A disqueta gratuita que acompanha cada exemplar da revista, é parte integrante da mesma e não pode ser comercializada separadamente.

NOVA SÉRIE INVES PC



PROMOÇÃO DE LANÇAMENTO

420 000$00 + IVA



INVES PC-325 S (Minitower)

- Processador 80386SX-25 MHz
- 1 a 8 Mb RAM

INVES NB-320S (Notebook)

- Processador 80386SX-20 MHz
- 2 a 4 Mb RAM
- 2 Portas série, 1 Paralela, 1 Game
- Adapt. Modem/Network externo
- Porta 8 pinos para Handy Scanner
- Drive 3 ½" – 1,44 Mb
- Disco rígido 2½" – 80 Mb
- Ecrã cristais líquidos contra-iluminados VGA 640x480, 32 tonalidades de cinzento
- Socket para 80387SX
- Teclado 81 teclas, 12 funções
- Dimensões: 290x220x53 mm



INVES PC-340D (Supertower)

- Processador 80386DX-40 MHz
- 2 a 64 Mb RAM

INVES PC-433D (Supertower)

- Processador 80486DX-33 MHz
- 4 a 64 Mb RAM

OUTRAS CONFIGURAÇÕES POSSÍVEIS (todos os modelos):

- Drives de 3½" - 1,44 Mb e 5¼" - 1,2 Mb
- Discos de diversas capacidades
- Placas vídeo VGA de 256 Kb a 1 Mb RAM
- Monitores VGA mono ou SuperVGA (1024x768)

GARANTIA 2 ANOS

CONDIÇÕES ESPECIAIS DE PAGAMENTO

SAT
SERVIÇO ASSISTÊNCIA TÉCNICA
TODAS AS MARCAS
Tel. 778 73 24
Fax 778 53 01

inves

REPRESENTANTE OFICIAL:

INVESMÁTICA – Investigação e Informática, Lda.
Rua Abel Botelho, 8-A – 1500 LISBOA – PORTUGAL
Telefs.: 351-1-7740880/7787324/7787326 – Fax: 351-1-7785301

Distribuidores:

RVF – LISBOA – MALVEIRA – Telef.: 986 21 59
JOTINFOR – SEIXAL – PAIVAS – Telef.: 225 06 90
JOÃO F. CLEMENTE – CALDAS DA RAÍNHA – Telef.: 062/34 203
SODEFE – ABRANTES – Telef.: 041/21 184
A. ANGELO – LISBOA – LUMIAR – Telef.: 758 92 02
ORLANDO G. FREIRE – SINES – S. ANDRÉ – 069/73 734

Graficria pub

Nº 17 – Abril de 1992



Spooler
MAGAZINE DOS UTILIZADORES PC

Já tardava...

Claro que a **Spooler** não possui nenhuma imunidade contra «inflações», «IVA's» e outras moléstias periódicas que afligem todos e tudo o que se publica nesta santa terrinha. E, como não podia deixar de ser, apanhou-as e não tem outra alternativa senão «contaminar» os pobres leitores pois, que saibamos, ainda não foi descoberto nenhum «anti-vírus» contra tais maleitas. Falamos, como é evidente, do aumento do preço de capa que entrará em vigor a partir do próximo número. Não ultrapassará os dez por cento, o que representa cerca de metade do aumento verificado em Fevereiro do ano passado. Estamos certos que os nossos leitores serão receptivos a esta medida «impopular», tanto mais que será acompanhada por uma correcção que se impunha pôr em prática. Trata-se do custo das assinaturas, o qual passará a ter um diferencial de apenas 500$00 entre os dois tipos de disquetas. Tal torna-se possível pela diminuição do preço das disquetas de 3 1/2" que se tem vindo a verificar nos últimos meses, tudo apontando para uma cada vez maior aproximação entre os dois tipos. Para que os actuais assinantes na modalidade de 3 1/2" não se sintam prejudicados em relação aos que venham a fazer a assinatura a partir do número 18, a **Spooler** oferecer-lhes-á um *bonus* de **três** números! Esperamos, deste modo, satisfazer quantos têm reclamado contra uma certa «discriminação» por via das disquetas...

ESCRITORES, APRESENTEM-SE!

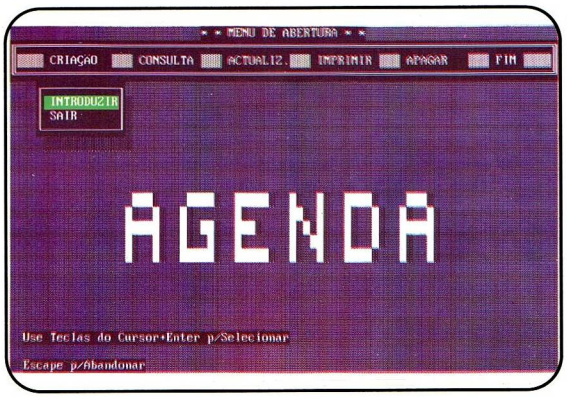
A GRAFICRIA, editora e proprietária da **Spooler**, incumbiu-nos de apreciar e propor a publicação de livros portugueses na área da informática. É nossa convicção existirem muitos jovens (e não só...) com capacidade para se lançarem numa actividade tão pouco dinamizada no nosso país. Não vos prometemos lucros chorudos – é sobejamente reconhecido que os livros não são fonte de grandes compensações materiais – mas é sempre gratificante vermos o nosso nome na montra de uma livraria e sabermos que o fruto do nosso esforço poderá contribuir para a divulgação e o ensino da informática. Revelem-se! Cá os esperamos.

Renato Casquilho

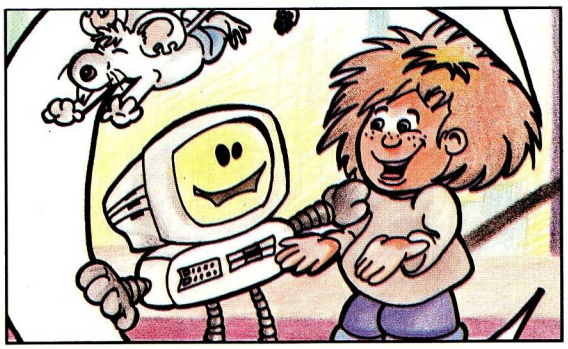
Spooler

- **Propriedade:** Graficria, Publicidade e Artes Gráficas, Lda.
- **Director de Edição:** Franco Gomes ■ **Director**: Renato Casquilho
- **Redacção:** Franco Gomes, Paulo Bernardo e Renato Casquilho.
- **Colaboradores:** Carlos Albuquerque, Carlos Ladeira, Cunha Brandão, José Orlando Pereira, José Páscoa, Vítor Monteiro e Vítor Guerreiro.
- **Ilustrações:** Leonor Gomes
- **Relações Públicas e Publicidade:** Fátima Ribeiro
- **Assinaturas e encomendas:** Paula Martinez
- **Redacção, Publicidade e Serviços Administrativos**: Rua Duarte Pacheco Pereira, 69-A – Damaia – 2700 AMADORA – Telfs.: 475 22 89/474 60 11/41
- **Produção Gráfica:** Graficria, Publicidade e Artes Gráficas, Lda.
- **Impressão:** Empresa Litográfica do Sul, S.A. ■ **Distribuição:** Electroliber
- **Revista bimestral** ■ **Preço:** 500$00 ■ **SRIP N.º** 213900
- **Tiragem:** 12500 exemplares ■ **Dep. Legal N.º:** 28587/89

2 Informática em Notícia
4 O Conteúdo da Disqueta
6 Correio dos Leitores
10 História e Programação de Roedores Informáticos (Parte III)
12 Programar em Assembly (Parte I)
16 A Utilização da Memória Extra
21 Programar em Clipper (Parte I)
24 Um Microprocessador por Software
28 A Dinastia Intel (Parte I)



30 Agenda
31 Astúcias



32 Programar em C (Parte I)
38 D. O. S. – (Parte 6)
41 Montra Spooler

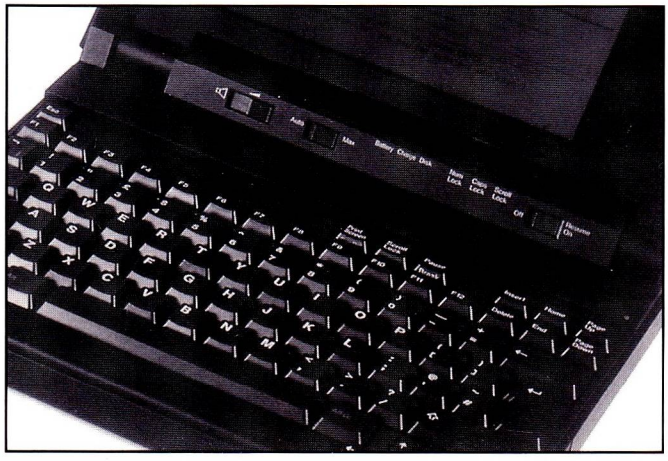
INFORMÁTICA EM NOTÍCIA



FRANQUEADORA ELECTRÓNICA FRANCOTYP POSTALIA T 1000

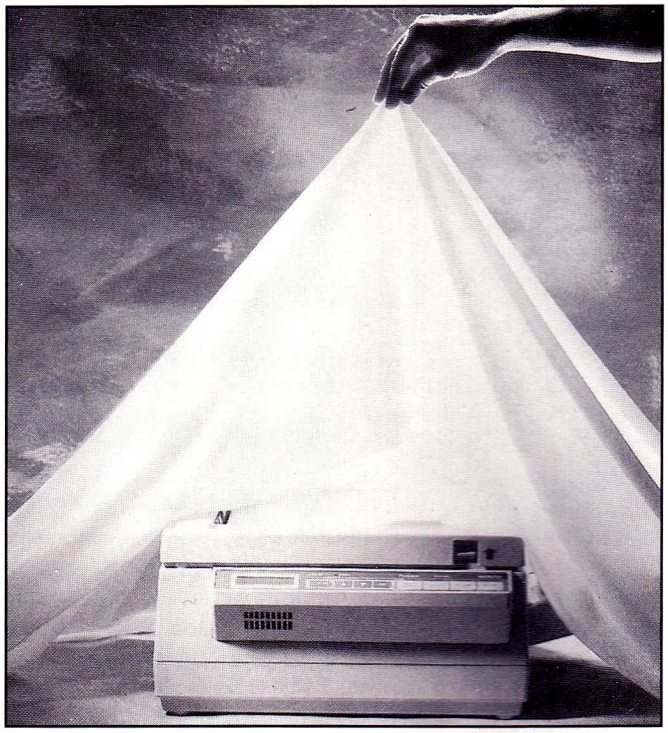
Vocacionada para os pequenos e médios utilizadores, a franqueadora Francotyp Postalia T 1000 foi recentemente apresentada no nosso mercado pela Equisistemas, Lda. Inteiramente electónica, a T 1000 permite o controlo por centros de custo e não usa os cunhos tradicionais, podendo ter até seis «cunhos» de correspondência memorizados na *eprom*. Relativamente aos Correios, a T 1000 apresenta as hipóteses de pré-pagamento ou carregamento telefónico.

IBM PS/2 NOTE



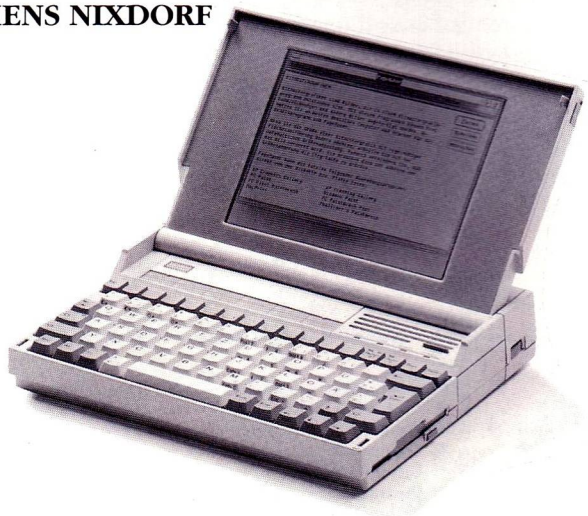
Integrados na gama de portáteis PS/2, a IBM anunciou em fins do passado mês de Março dois novos modelos: PS/2 N33 SX e PS/2 N51 SX.
No que refere ao PS/2 N33 SX serão comercializados dois modelos AT-Bus, o A33 e o A35, com discos rígidos de 40 Mb e 80 Mb, respectivamente.
Processador i386 SX a 16 NHz, 2 Mb de RAM, ecrã de cristais líquidos VGA monocromático com 16 tonalidades de cinzentos, unidade externa de disquetas de 1,44 Mb, porta VGA para ligação a monitor externo, portas série e paralela, porta para «rato» ou teclado numérico externo e porta para ligação de placa de comunicações opcional.
Por seu turno, o PS/2 N51 SX – primeiro *notebook* com arquitectura Micro Channel, está também baseado no i386 SX a 16 MHz e será comercializado o modelo A33 com 2 Mb de RAM exp. a 10 Mb, ecrã de cristais líquidos VGA mono de 9,5" retro-iluminado, comportando até 32 tonalidades de cinzentos, unidade interna de disquetas de 1,4 Mb e conjunto de portas idêntico ao PS/2 N33 SX.
Apresentados no formato compacto A/4, pesam respectivamente 2,5 Kg e 2,8 Kg incluindo o conjunto de baterias.

IMPRESSORA LASER MICROTEK MTP-306



Aliando um preço aliciante (390 contos, IVA não incluido), esta impressora Laser utiliza a nova linguagem de descrição de página TrueImage da Microsoft, que inclui TrueType (a mesma tecnologia de fontes que será incorporada no Apple Macintosh System 7.0 e no Microsoft Windows 3.1) A resolução é de 300x300 e a velocidade de 6 páginas por minuto.
Equipada com o microprocessador RISC de 32 bit AMD 29000, com uma velocidade de 16 MIPS (milhões de instruções por segundo), a MTP-306 dispõe de interfaces Apple Talk, Centronics e RS-232C, podendo trabalhar simultaneamente com plataformas PC e Macintosh.
Um sistema de «auto-switch» permite a diversos utilizadores o acesso à mesma impressora. Este produto é representado no nosso país por H. S. C. Ibérica.

NOTEBOOK 386 A 20 MHZ SIEMENS NIXDORF



Equipado com um processador Intel 80386SX a 20 Mhz, este novo modelo – «irmão mais velho» do PCD-3NSX/16 – vem alargar a gama de *notebooks* da Siemens Nixdorf.
De tamanho reduzido, extremamente leve (3,2 Kg) e com autonomia de 3 horas, este modelo está especialmente indicado para postos móveis em redes locais e aplicações verticais no sector de vendas, banca e seguros.
Monitor VGA LCD de 9", com 32 tonalidades de cinzentos, 2 Mb de RAM e disco rígido de 60 Mb complementam os atributos deste elegante *notebook*.

CURSOS de Informática

Em colaboração com **INVESMÁTICA – Investigação e Informática, Lda,** a **Spooler** vai levar a efeito mais uma série de cursos a iniciar em **Junho** próximo.

A CURSO GERAL DE INFORMÁTICA

- Início do curso: 2 de Junho de 1992
- Fim do curso: 2 de Julho de 1992
- Duração: 27 horas
- Requisitos: Nenhuns
- Horário: Terças e Quintas das 18.30 às 21.30 horas
- Custo: 30 000$00 (IVA incluído)

B PROGRAMAÇÃO EM C

- Início do curso: 1 de Junho de 1992
- Fim do curso: 3 de Julho de 1992
- Duração: 27 horas
- Requisitos: Conhecimentos médios de MS-DOS
- Horário: Segundas, Quartas e Sextas, das 18.30 às 21.30 horas
- Custo: 40 000$00 (IVA incluído)

C PROGRAMAÇÃO EM CLIPPER

- Início do curso: 1 de Junho de 1992
- Fim do curso: 1 de Julho de 1992
- Duração: 42 horas
- Requisitos: Conhecimentos médios de MS-DOS
- Horário: De Segunda a Sexta, das 10.30 às 12.30 horas
- Custo: 40 000$00 (IVA incluído)

D PROGRAMAÇÃO EM ASSEMBLY 8086/88

- Início do curso: 2 de Junho de 1992
- Fim de curso: 2 de Julho de 1992
- Duração: 27 horas
- Requisitos: Conhecimentos médios de MS-DOS e de, pelo menos, uma linguagem de programação
- Horário: Terças e Quintas, das 15.00 às 18.00 horas
- Custo: 35 000$00 (IVA incluído)

E HARDWARE DO IBM-PC E PERIFÉRICOS

- Início do curso: 2 de Junho de 1992
- Fim de curso: 2 de Julho de 1992
- Duração: 27 horas
- Requisitos: Conhecimentos médios de MS-DOS e de, pelo menos, uma linguagem de programação
- Horário: Segundas e Quartas, das 15.00 às 18.00 horas
- Custo: 35 000$00 (IVA incluído)

F CICLO DE *WINDOWS* E APLICAÇÕES

F1 – Introdução ao ambiente *Windows*

- Início do curso: 6 de Julho de 1992
- Fim do curso: 7 de Julho de 1992
- Duração: 14 horas
- Requisitos: Nenhuns
- Horário: Manhã das 9.30 às 13.00; tarde das 14.30 às 18.00 horas
- Custo: 17 000$00 (IVA incluído)

F2 – Programa de desenho *Corel Draw 2.0*

- Início do curso: 8 de Julho de 1992
- Fim do curso: 10 de Julho de 1992
- Duração: 21 horas
- Requisitos: Conhecimentos médios de ambiente *Windows*
- Horário: Manhã das 9.30 às 13.00; tarde das 14.30 às 18.00 horas
- Custo: 31 000$00 (IVA incluído)

F3 – Processador de texto *Word for Windows 2.0*

- Início do curso: 13 de Julho de 1992
- Fim do curso: 16 de Julho de 1992
- Duração: 28 horas
- Requisitos: Conhecimentos médios de ambiente *Windows*
- Horário: Manhã das 9.30 às 13.00; tarde das 14.30 às 18.00 horas
- Custo: 42 000$00 (IVA incluído)

F4 – Folha de cálculo *Excel 3.0*

- Início do curso: 20 de Julho de 1992
- Fim do curso: 22 de Julho de 1992
- Duração: 21 horas
- Requisitos: Conhecimentos médios de ambiente *Windows*
- Horário: Manhã das 9.30 às 13.00; tarde das 14.30 às 18.00 horas
- Custo: 31 000$00 (IVA incluído)

F5 – Gestor de base de dados *dBFAST*

- Início do curso: 23 de Julho de 1992
- Fim do curso: 24 de Julho de 1992
- Duração: 14 horas
- Requisitos: Conhecimentos médios de ambiente *Windows* e de dBASE
- Horário: Manhã das 9.30 às 13.00; tarde das 14.30 às 18.00 horas
- Custo: 21 000$00 (IVA incluído)

- Turmas com máximo de 8 participantes
- 1 computador (marca **INVES**) por participante
- Material didáctico incluído no preço do curso:
  - manual
  - disqueta de trabalho
  - dossier, bloco de apontamentos, caneta, lápis, borracha
- Certificado de Frequência

Os interessados devem remeter-nos o cupão ao lado (ou fotocópia), devidamente preenchido e acompanhado do pagamento de 30% relativos ao custo total do curso. O restante poderá ser pago em 2 prestações, no início e a meio do curso.

LOCAL DO CURSO

INVESMÁTICA – Investigação e Informática, Lda.
Rua Abel Botelho, 8 c/v – 1500 LISBOA (Benfica)

Desejo frequentar o curso que assinalo com um ☒

A	B	C	D	E
F1	F2	F3	F4	F5

NOME .....

ENDEREÇO .....

C. POSTAL ..... LOCALIDADE ..... TELEF .....

Junto envio a importância de _____ $__ referente a 30% do custo total do curso em cheque ou vale de correio.

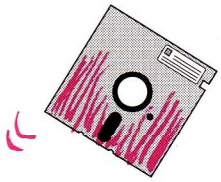
_____
Assinatura

SPOOLER MAGAZINE

Rua Duarte Pacheco Pereira, 69 / A – Damaia
2700 AMADORA – Telfs.: 475 22 89/474 60 11/41



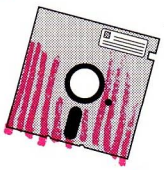
O CONTEÚDO DA DISQUETA



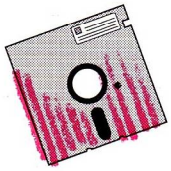
A Spooler adquire, em cada número, novos leitores. É a esses que dedicamos esta pequena introdução. O material contido na disqueta encontra-se comprimido sob a forma de ficheiros autodescompactáveis. Quer isto dizer que os nossos leitores deverão colocar-se no directório escolhido e copiar para um subdirectório do disco rígido ou para outra disqueta, o ficheiro com extensão .EXE. Depois basta digitar o nome desse ficheiro, premir ENTER e este descompactar-se-á automaticamente, dando origem aos diferentes ficheiros que constituem o respectivo programa ou utilitário. As excepções, se existirem, encontram-se devidamente assinaladas nesta rubrica.

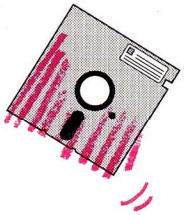


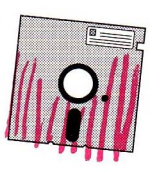












Desta vez, a disqueta **Spooler** traz uma novidade. O facto de, neste número, iniciarmos três cursos de programação, criou-nos um problema de espaço para alguns artigos. Mas nem por isso queremos privar os leitores das necessárias explicações acerca dos programas, nem os autores da devida consideração pelos seus trabalhos. Desta forma, temos dois utilitários cujos artigos acompanham os ficheiros compactados no respectivo directório da disqueta. Têm a mesma designação dos programas, mas com a extensão .DOC. Basta corrê-los em qualquer editor ASCII e lê-los no vosso ecrã.

DIRECTÓRIO AGENDA

Contém o ficheiro AGENPRG.EXE o qual, depois de descompactado, dará origem a um conjunto de ficheiros destinados a correr dentro do dBASE III Plus. O autor enviou-nos também um executável compilado com o Clipper Summer 87, mas o espaço disponível na disqueta não permitiu a sua inclusão. Mas leia o artigo do Guilherme Santos antes de correr o programa.

DIRECTÓRIO BIBTECA

Conforme foi anunciado no número anterior, motivos de ordem técnica impediram-nos a inclusão do programa na disqueta. Ele aqui está, em conjunto com as «fontes». Mais uma vez as nossas desculpas ao autor e aos nossos leitores.

DIRECTÓRIO BOXATTR

Este directório contém o ficheiro BOXATTR.EXE. Trata-se da fonte (compactada) em *Assembly* de uma rotina que, por lapso, não foi incluída conjuntamente com as «fontes» do programa GENIO. Para mais pormenores sobre este assunto, queiram ler a rubrica «Correio dos Leitores».

DIRECTÓRIO C_1

A primeira parte do curso «Programar em C» contempla a versão inicial de um programa demonstrativo da matéria apresentada. O leitor encontrará neste directório o ficheiro CALCULO.EXE o qual, depois de descompactado, dará origem aos executáveis e às «fontes» em C.

DIRECTÓRIO DOS

DOSEXE.EXE é a designação do ficheiro contido neste directório. Após descompactação dará origem ao programa objecto do 3º Prémio do passatempo «Programas Didáticos», da autoria de João Carlos Rodrigues Pereira.

DIRECTÓRIO EMS

A adequada utilização da memória dos nossos computadores é matéria para um excelente artigo do nosso colaborador Eng. Paulo Bernardo. As suas explicações são demonstradas pelo programa e «fontes» .PAS contidas neste directório.

DIRECTÓRIO MICROPC

«Um Microprocessador por *Software*» é o título de um interessante artigo da autoria de Jorge Anastácio, para o qual chamamos a atenção dos nossos leitores. Descompacte-se o ficheiro MICROPC.EXE e simule-se o funcionamento de um processador.

DIRECTÓRIO PCIMP

Configurar as nossas impressoras é tarefa fácil com o utilitário IMPRIME, tal como nos é sugerido pelo Filipe Costa. O tal condicionalismo de espaço, tanto na revista como na disqueta, impede-nos de publicar as «fontes» e obriga-nos a oferecer aos nossos leitores o artigo no formato .DOC. O descompactácel contido neste directório dá pelo nome de PCIMPEXE.EXE.

DIRECTÓRIO PUDISCO

Francisco Garcia meteu mãos à obra e ofereceu-nos um simpático utilitário destinado a gerir os nossos discos e os nossos ficheiros. Fazendo nossas as palavras do orador anterior, resta acrescentar que o descompactável se chama PUDEXE.EXE e o artigo – não deixem de o «verler» – está no formato .DOC.

Graficria pub

SENSACIONAL CAMPANHA

MOUSE ABC

3 Teclas, com tapete: Software de instalação e adaptador de 9/25 pinos
4 resoluções 100/200/400/800 Dot

de 5 a 10 unid.	+10 unid.
3 800$00	3 500$00

DISKETES

		de 10 a 50 Cx.(Cx. de 10)	+ 50 Cx.
BUDGET 3,5"	DD	900$00	850$00
BUDGET 3,5"	HD	1 850$00	1 750$00
BASF 5,25"	DD	880$00	800$00
BASF 5,25"	HD	1 300$00	1 200$00
BASF 3,5"	DD	1 300$00	1 200$00
BASF 3,5"	HD	2 300$00	2 200$00

COMPUTADORES da marca ABC

MINITOWER 386SX – 25 Mhz

- 2 Mb RAM
- 16 Kb Cache memória
- Drive 3,5" a 1.44" Mb
- Hard DisK 80 Mb a 15 ms
- 2 Portas Série RS232
- 1 Porta Paralela
- Teclado português
- Sistema Operativo MS. DOS 5
- Monitor Super VGA Policromático (1024×768)

PREÇO: 259 000$00

BIGTOWER 486 – 33 MHz

- 4 Mb RAM
- 256 Kb Cache memória
- Drive 3,5" a 1.44" Mb
- Hard Disk 211 Mb a 15 ms
- 2 Portas Série RS232
- 1 Porta Paralela
- Teclado português
- Sistema Operativo MS. DOS 5
- Monitor Super VGA Policromático (1024× 768)

PREÇO: 535 000$00

PORTÁTIL TOSHIBA 1200XE

80286 – 1 Mb RAM – 40 MGHD

OPORTUNIDADE ÚNICA
170 000$00
(+16% I.V.A.)

COMPUTADORES PORTÁTEIS

TOSHIBA	ASI
Mod. 2200 SX	Mod. Premium Exec
Processador 386 SX	Processador 386 SX
Velocidade 20 MHz	Velocidade 25 Mhz
Memória RAM 2 Mb	Memória RAM 4 Mb
Disco 60 Mb	Disco 63 Mb
Ecran VGA	Ecran LCD Super VGA
Bateria Aut. 2,5 H	Bateria Aut. 3 H
Resolução (800×600)	Resolução (1024×768)
MS. DOS 4.01	MS. DOS 5.01
PREÇO: 580 000$00	**PREÇO: 595 000$00**

AGENTES CASA VIOLA

BRAGA:
Av. Central, 85 – 1º
Telefs.: 72 798-74 369
S: JOÃO ESTORIL:
Av. Florinda Leal, Lote 1-A
Telef.: 467 07 33
VISEU:
Rua Direita, 79 – 1º
Telef.: 27 664
PORTIMÃO:
Rua D. Carlos I
Telefs.: 83 653-85 670
SETÚBAL:
Largo da Misericórdia, 28
Telef.: 31 432

abc A VERDADE EM SOLUÇÕES INFORMÁTICAS

Rua dos Sapateiros, 160 – 2º — 1000 LISBOA
Telef.: (01) 342 44 48/347 80 94/347 78 96 – Telex: ABCINF 13348 P — Fax: 346 22 40/342 67 22

Correio dos Leitores



Por Renato Casquilho

...na última revista da Spooler foi publicado um artigo sobre o vencedor do passatempo, que foi ganho pelo programa MESTRE. Na disquete que acompanha a revista vem a listagem do mesmo, que depois de compilado dá erro na instrução _BOXATTR. Isto acontece porque a rotina está escrita em Assembler e talvez o autor do referido programa não a possua. E porque será que o autor não a possui? Talvez porque não foi escrita por ele.

José Calado – Coimbra

A resposta de Victor Monteiro esclarece este assunto:
«...Com efeito, o programa MESTRE tem uma função escrita em Assembler, que tem como objectivo desenhar as sombras das caixas. Função essa que não foi realmente escrita por mim (aliás na conversa que tivemos nas instalações da Spooler, eu confessei que não conhecia ainda a linguagem Assembler). A listagem que tenho desta função (e que enviei na primeira versão do MESTRE), não tinha qualquer texto de identificação do autor. Sendo assim, achei que era uma função escrita para livre utilização por parte dos programadores interessados. Como agora apareceu o autor da função, considero que se deveria enviar na disquete o procedimento _BOXATTR.ASM (que, por lapso, não acompanhou as restantes fontes do MESTRE), devidamente identificada pelo nome do autor. Acho importante assinalar que esta função não afecta o funcionamento do programa, apenas tem como missão desenhar as sombras das caixas dos *menus pull-down*. Na disquete que envio junto, segue a listagem desta função, já devidamente identificada.»
A fonte em *assembly* desta função encontra-se na disqueta deste número, fazendo justiça ao seu autor.

...gostava que me informassem onde posso arranjar a disqueta dBASE III Plus para chamar o programa dBASE e arrancar com o FILATECA...
...diversas...

Luis Aguilar – Covilhã

O dBASE III Plus da Ashton Tate é um *package* comercial protegido por *copyright*, pelo que poderá ser adquirido em qualquer loja de *software* ou na «Montra Spooler».
O nosso colaborador Victor Monteiro responde a «diversas»:
«Antes de mais, é necessário explicar que o MS-DOS apenas executa programas que tenham como extensão EXE, COM ou BAT. Os programas escritos em dBASE têm como extensão PRG, o que significa que é necessário correr primeiro o dBase e só então é que se pode chamar o programa. A tradução dos programas escritos em linguagens de alto nível («programas fonte») em linguagem máquina («programas objecto») é efectuada através de programas especiais que se designam por compiladores. Algumas linguagens de alto nível (como por exemplo o BASIC), permitem que os programas possam ser imediatamente executados mediante a intervenção de programas chamados interpretadores. No momento da execução do programa, o interpretador traduz cada uma das instruções de alto nível em instruções máquina inteligíveis para o CPU. É evidente que um programa interpretado se torna muito mais lento na sua execução do que um programa compilado. O interpretador está constantemente a servir de «intermediário» entre as instruções de alto nível e o computador. É isso que torna os programas interpretados muito mais lentos na execução.
As linguagem Clipper, Pascal, *Assembly* e C, necessitam de ter um compilador que transforme o respectivo código fonte em linguagem máquina (a única que o computador entende).
O EXE2BIN do MS-DOS, tem como objectivo transformar os ficheiros executáveis (de extensão EXE) em ficheiros binários, e não serve para transformar códigos fonte em ficheiros executáveis.»

*No número 15 da **Spooler**, o leitor Artur Manuel Da Costa – Porto, colocava uma questão acerca do teste do estado da impressora num programa escrito em Cobol. Recebemos duas respostas ao nosso apelo. Uma, do leitor Luis Miguel Palma Gomes,, outra do leitor José Avelino Moreira Victor. Publicamos a segunda por ser mais completa e didática. Aqui vai a resposta do seu conterrâneo:*

Na sequência do desafio por vós lançado no anterior número, pedindo a validação do estado da impressora em COBOL, resolvi empenhar-me um pouco e tentar descobrir a solução a esse problema.
Chamo-me José Avelino Moreira Victor e, trabalhando já há algum tempo com a linguagem COBOL consegui, após uma consulta aos manuais de que disponho, achar a solução para o problema. A abordagem do leitor que lançou o problema estava a seguir não seria a mais correcta, visto que o valor contido na variável de FILE-STATUS seria um erro do ponto de vista de ficheiro e não de impressora, uma vez que o COBOL trata a impressora como ficheiro. Para ultrapassar o problema recorri a uma pequena rotina em *Assembly* que, utilizando a interrupção 17h do ROM-BIOS, permite obter o estado da impressora. A partir daí o problema resume-se à passagem desse estado para o programa de COBOL, que se resolve com a instrução CALL, usando 3 parâmetros :

PARM1 – destina-se a enviar o valor 02 para AX.
PARM2 – destina-se a enviar o valor 00 para DX.
PARM3 – parâmetro destinado a receber o valor do estado da impressora.

Os valores de retorno (contidos na variável PARM3), podem ser os seguintes :

00 – Impressora Desligada.
44 – Impressora Pronta.
24 – Impressora com ON-LINE Desligado.
56 – Impressora sem Papel.

...a sugestão que queria fazer era, se possível, incluir nas vossas páginas um suplemento mais completo a manuais de linguagens de programação...

Ângelo Manuel da Silva Ferreira/ Tondela

Pensamos que a sua sugestão será (já está a ser) satisfeita neste mesmo número da nossa revista, através das novas rubricas sobre «C», «*Assembly*» e «Clipper». Proximamente contamos incluir o COBOL nos nossos «minicursos» para programadores.

O nosso leitor Artur Moreira, do Porto, tem dificuldades em correr o programa GENIO. A extensão da sua carta, aliás muito amável, não permite a sua reprodução. Mas pensamos que o seu problema pode ser resolvido como nos diz o Victor Manuel Monteiro:

«O problema com que este nosso leitor se depara é muito frequente em programas escritos em Clipper e em aplicações de dBASE. Para poder trabalhar com estes programas deverá ser inserido no CONFIG.SYS a instrução FILES=20 (ou 30). Quanto à segunda questão sobre a resposta dada a um leitor de Braga, em que se dizia que 'o programa GENIO está preparado para eliminar registos existentes', já ficou esclarecido que se tinha tratado de uma confusão com o programa MESTRE».

...Congratulo-me por terem começado a abordar o DEBUG. Apenas lamento que não o façam de forma detalhada, porque muitos leitores como eu não percebem ainda grande coisa de Assembly.
...Poderiam indicar-me alguns livros ...

Sérgio Alcides Fernandes/Ponte de Lima

A pessoa indicada para lhe responder é o Eng. José Páscoa:
«O DEBUG é um programa que tem acompanhado todas as versões do MS-DOS desde o seu início. Uma das suas principais utilidades consiste em converter para código máquina as instruções *Assembly* que o utilizador introduz, quer directamente pelo teclado, quer redireccionando um ficheiro cujo conteúdo corresponde exactamente ao que se digitaria no DEBUG para a sua introdução. A descricção do seu funcionamento consta, como sabe, dos manuais do MS-DOS. O que esses manuais de facto não ensinam é linguagem *Assembly*. E o DEBUG não é também a ferramenta mais adequada para realizar essa aprendizagem. Para essa finalidade o leitor deverá começar por adquirir um Assemblador. Recomendamos um bastante bom que dá pelo nome de A86 e que é distribuído em regime *Shareware*. Fazem parte dele vários ficheiros texto com uma boa introdução á linguagem. Como pode verificar ele consta da Montra Spooler da nossa revista. A indicação de um bom livro para iniciar o estudo da linguagem *Assembly* é uma pergunta muito mais dificil. As melhores obras existem em inglês. Nacionais infelizmente não conheço nenhuma. Mas se o leitor prometer tomar as devidas precauções com a terminologia talvez acabe mesmo por lhe recomendar uma obra brasileira (não é uma tradução) que achei bastante interessante e se chama: "Programando em Assembler 8086/8088" da editora McGraw-Hill. E naturalmente não deixe de continuar a seguir a série de artigos sobre programação em *Assembly* que a nossa revista iniciou neste mesmo número.»

Há alguma maneira de o computador

BBSpooler

☎ (01) 475 22 89

Dias úteis: das 18.30 às 9.00 horas
Feriados e fins-de-semana: 24 horas

ao ser ligado, ler a unidade B antes da A ?

José Leça/Esmoriz

O Eng. José Páscoa dá a resposta:
«Estritamente falando não há. O computador ao ser ligado corre um programa residente em ROM que após inicializar o sistema passa o controle a uma rotina de nome *bootstrap loader* a qual tenta sempre o arranque pela *drive* A. Se houver disquete nessa *drive*, é carregado em memória e executado um pequeno programa existente no primeiro sector da disquete. Esse pequeno programa pode servir para carregar em memória o sistema operativo (IO.SYS e MSDOS.SYS se for MS-DOS), para carregar um programa autoexecutável como o Flight Simulator, ou pode não servir nem para uma coisa nem para outra e então o controle é devolvido ao *bootstrap loader* que nos comunicará a conhecida mensagem de erro de que a disquete não serve para o arranque. No caso de não haver disquete na *drive* A mas existir disco rígido então o computador tenta a execução de um programa semelhante existente no sector *boot* da partição de arranque desse disco. Todo este procedimento é dirigido pelo programa residente em ROM do computador e por conseguinte não pode ser. Mas nem tudo está perdido, pois existe uma fuga, que é quando o *bootstrap loader* faz executar o pequeno programa existente no primeiro sector da disquete existente na *drive* A. Se esse programa for substituido por um outro que faça arrancar a *drive* B então temos o problema resolvido na maior parte dos casos. Conhecemos um programa de nome BOOT_B escrito por Scott A. Numbers (e existente em algumas BBS portuguesas) que faz precisamente isso. Mas advertimos que não é uma solução universal para todas as situações».

...sempre que tento correr o FIND a partir da prompt o computador envia para o ecrã «format not correct»; no entanto se correr o utilitário a partir do XTGOLD ele é executado! Qual a explicação?
...O HDIR embora esconda os directórios no TREE do MS-DOS esles são visualizados pelo XTGOLD...
..não consigo imprimir as letras acentuadas portuguesas no Windows utilizando as fontes originais da impressora, excepto por sobreposição de caracteres...
...Como posso acelerar a impressão de texto a partir do Windows...o meu computador tem 2 MB de RAM.

João Madeira/Almada

Quanto às duas primeiras perguntas, passamos «a bola» para o autor dos programas, o João Carlos Rodrigues Pereira. As outras serão respondidas pelo nosso colaborador Eng. José Páscoa.
«De facto se a sua impressora não possui *eprom* com caracteres portugueses (*code page* 860) só a impressão em modo gráfico com as *soft fonts* que acompanham o Windows constituirá uma alternativa.
Sobre a aceleração da impressão. Todos sabemos que o Windows é muito lento em tudo, mas existem alguns procedimentos que o espevitam um pouco:
- Desactive o Print Manager no Control Panel.
- Se puder instale mais 2 MB de RAM e reserve 1 para uma *cache* de disco (SMARTDRV.SYS por exemplo) e outro para um disco virtual como o RAMDRIVE.SYS. Faça apontar a variável de environment TEMP para esse disco virtual.
- E não trabalhe em modo Windows enhanced se o modo standard for suficiente para as suas necessidades».

...Qual a quantidade de memória a reservar para uma cache de disco e para uma drive virtual num PC com 2 MB de RAM, 4 MB de RAM, etc.
...Um PC com 2 MB deveriam ser 640 KB + 1408 KB extended, mas no computador em que trabalho são 640 KB + 1024 KB.
...Eu tentei recuperar os dados de um computador com a FAT danificada. Utilizei o Norton Disk Doctor e apareceram muitos ficheiros como File001, File002. Como posso reconstituir o original?
...Como é possível recuperar um disco a que por engano se tenha desactivado a partição activa?
...O computador dá a seguinte mensagem "on board parity error at adress (Hex) 000400"

Fernando José Abrantes/Parede

Mais uma vez, a resposta do Eng. José Páscoa:
«Em relação á primeira questão do nosso leitor não existe infelizmente uma resposta única. Tudo depende das aplicações que pretendamos correr no computador. Repare que a *drive* virtual e a *cache* de disco são apenas duas utilizações para a memória extendida ou expandida. Folhas de cálculo electrónicas, processadores de texto e bases de dados podem fazer também uso interno dessas categorias de memória, caso a tenham á disposição.
Para a segunda questão começo por dizer que a RAM não desapareceu, ela de facto está lá. O que se passa é que muitos PC com processador 386 (e alguns 286) passam durante a inicialização do sistema a zona do ROM (compreendida entre os 640 KB e os 1024 KB e que soma 384 KB) para RAM. Designa-se essa operação por passagem do ROM para Shadow RAM por *hardware*. Essa operação tem vantagens pois o ROM só pode ser acedido 8 bits de cada vez enquanto a RAM de um 386 é-o a 32 bits o que trás ganhos nítidos em termos de velocidade. Também parte dessa memória pode ser aproveitada para colocação de programas TSR (residentes após a passagem de controle ao DOS). Contudo o utilizador pode querer eliminar esse *shadowing* e aqui é que pode surgir o problema. Enquanto a generalidade dos PC permite que o *shadowing* seja desligado, nem todos libertam essa memória para memória de extensão. E o seu caso pode ser um desses. Á sua terceira questão só lhe posso dizer que a recuperação é como reconstituir um *puzzle*. Tente recuperar o mais importante e esqueça o resto.
Para a quarta questão: Se não alterou a tabela de partições mas só desactivou a partição activa, corra o FDISK e active-a.
Para a quinta questão: Erros como o que indica especialmente quando sistemáticos tem normalmente a ver com *chips* de memória avariados».

...Tendo já trabalhado com o TURBO PASCAL bem como RM-COBOL 85 e sabendo que o primeiro não faz uma gestão de ficheiros, como é feita no segundo, gostaria de saber se a linguagem C me faz a gestão dos ficheiros .

Carlos Nunes/Coimbra

Pedimos ao nosso colaborador Carlos Ladeira que preparasse uma resposta a este nosso leitor:
«O COBOL é uma linguagem de programação orientada para um tipo de programação completamente diferente do C e do PASCAL. Enquanto o COBOL (RM-COBOL 85 no caso do nosso leitor) foi criado para manipular ficheiros de uma forma relativamente fácil, o C e o PASCAL têm outros objéctivos. Mesmo assim é possivel, tanto em C como em PASCAL, gerir ficheiros como no COBOL, embora para isso o programador tenha de implementar, ou adquirir, um gestor de bases de dados. No caso do nosso leitor, como disse na sua carta que estava a aprender C, penso que o melhor que tem a fazer é adquirir um gestor de bases de dados ou então continuar a fazer programas em COBOL».
Permitimo-nos acrescentar que este mesmo número da **Spooler** inicia um curso de «C» e que contamos, brevemente, alargar os nossos «mini-cursos» ao COBOL.

...queria muito ter os programas Clipper e C, mas não sei onde os arranjar...
...adorei montes de programas que vinham nas vossas disquetes...mas nem o ZAP nem o SOFGEST porque aparecía-me no primeiro a prompt, no último «Error in EXE file»...

Miguel Nuno da Costa

Os programas Clipper e C são protegidos por *Copyright*, pelo que deverão ser adquiridos nas casas da especialidade.
Os elementos que nos dá são insuficientes para podermos imaginar a solução para esses problemas. Já leu atentamente os respectivos artigos?

...Tenho um PC200 da Sinclair...comprei uma drive externa de 5 1/4 DD...liguei os cabos e não sei como se liga a drive além dos cabos...

André Manuel Gomes/Porto

Gostaríamos de o ajudar, mas acontece não conhecermos o seu computador. Aliás, e isto destina-se a todos os leitores que nos escrevem ou nos telefonam sobre situações semelhantes, a **Spooler** não deve «meter foice em seara alheia», sobretudo no que respeita a equipamentos e *hardware*.. Para este tipo de problemas só existe uma boa solução: pedir as necessárias informações à casa que vendeu o computador ou os acessórios.

...no meu PC 286-16MHz tenho vários menus gerenciadores em BATCH que saiem a preto e branco. Como será possível que saiam a cores?

Alvaro da Costa Miranda/Penacova

No número 9 da **Spooler** publicámos um artigo intitulado «ANSI.SYS» que contém tudo o que é necessário para pôr os *BATCH* a dar cores. Que tal uma leitura desse artigo?

...o vosso colaborador Carlos Azevedo diz no artigo...«para além dos modos vídeo ainda é possível especificar nas placas EGA ou VGA, se a memória deve ou não ser limpa» Mais adiante...para somarmos 128 ao modo de vídeo...pergunto: este valor é hexadecimal ou decimal? A linha a ser executada então deve ser «MOV AL, 12h+128?
...Segundo - se o anterior não der certo eu pergunto se será possível utilizar ficheiros de GIF's ou no Pictor ou no meu programa...

Eduardo Reis/Trofa

Eis a resposta de Carlos Azevedo:
«Relativamente à primeira pergunta a resposta é *decimal*. Por norma, todos os valores que não são terminados por «h» (no caso do *Assembly*) ou indicados por «0x» (no caso do C ou C++), são valores decimais. A linha a executar pode então ser MOV AL,12h+128 ou, simplificando, MOV AL,92h.
Quanto à segunda pergunta a resposta é afirmativa. A única regra a respeitar prende-se com o número de cores e a resolução da imagem; convém, para uma maior simplicidade, que a imagem a carregar tenha sido criada já com o tamanho necessário e no modo de vídeo pretendido, pois caso contrário torna-se necessário proceder a

Correio dos Leitores

conversões. No caso de GIF's aconselho-o a consultar o Nº 5 da **Spooler** onde é descrita a estrutura deste tipo de ficheiros.»

...às vezes quando estou a trabalhar num programa ou a jogar um jogo, o PC desliga-se aparecendo a seguinte mensagem: ***«ON BOARD PARITY ERROR, ADDR (HEX) = (2000:FC 04), SYSTEM HALTED****...gostaria de saber o que se passa...*

Luis Ferreira/Barreiro

O seu problema assemelha-se ao do leitor Fernando José Abrantes, pelo que o remetemos para a resposta dada a esse leitor pelo Eng. José Páscoa.

O nosso leitor Fábio Luis tinha colocado uma questão sobre programas residentes. Aqui vai a resposta do nosso colaborador José Orlando Pereira:

«A desinstalação de um programa residente pode, à primeira vista, causar duas espécies de problemas:
i) O mais grave é quando um programa ao instalar-se captura as mesmas interrupções do nosso programa. Ao repormos as interrupções iniciais deixamo-lo desactivado, o que poderá levar a todo o tipo de incómodos. Porém é o que tem solução mais simples, bastando comparar os vectores de interrupção actuais com o endereço da nossa rotina; se forem iguais, tudo bem; caso contrário deve recusar-se a desinstalação.
ii) O segundo problema prende-se com a fragmentação da memória livre, que a torna inútil. Para o impedir é necessário que o programa a retirar da memória seja o último a ser carregado, ou seja, o que se encontra no topo da memória. A solução para este caso é um pouco mais complicada, pois é preciso recorrer a funções do DOS não documentadas para percorrer a lista de blocos de memória ocupados. Pode ver como isto é feito examinando a fonte do programa DOSMEM publicado na Spooler, não esquecendo que o DOS 5.0 e os programas carregados acima dos 640Kb viciam as regras do jogo. Não tem de preocupar-se com a desinstalação no meio de secções críticas, tal como o acesso ao disco pois, nessa altura, o código da ROM inibe as interrupções e, consequentemente, não é interrompido por TSR's. No entanto existe outro problema: como para libertar a memória ocupada pelo TSR tem de utilizar um serviço da INT 21h, há que tomar uma precaução: como o «tic» do relógio pode aparecer numa altura em que já está a ser executado código dentro do DOS, e este não é reentrante, isto é, não aceita uma chamada antes de terminar a anterior, tem de certificar-se que está a ser executado código fora do DOS antes de invocar qualquer serviço. Para isso tem de recorrer a outra função não documentada, a 34h di INT 21h (*Get INDOS Pointer*), que devolve em ES:BX um apontador para uma *flag* de um *byte*, o qual, se for diferente de 0, indica que está a ser executado código do DOS. Deve-se ainda verificar que o *byte* imediatamente antes desse (isto é, ES:[BX-1]), também é nulo, para autorizar a chamada ao DOS. Caso contrário deve recusar-se a desinstalação do programa residente. Ainda mais uma sugestão: pode tentar encontrar forma de esperar que o código do DOS termine a sua execução, vigiando o INT 21h, o que lhe permitirá desinstalar o TSR noutra oportunidade. Se estiver interessado em explorar o DOS não documentado, sugiro o ficheiro INTERxxx.ZIP (xxx é a versão), de Ralph Brown que poderá encontrar nas BBS's ou o livro «Undocumented DOS» de Andrew Schulman, publicado pela Adison-Wesley.»

...qual a melhor linguagem para fazer jogos do tipo Golden Axe, Out Run, Formula 1, etc...onde aprender essa linguagem na área do Porto?

Fernando João Lira/Rio Tinto

Não temos dúvida de que a linguagem mais aconselhada para esse tipo de jogos, onde a rapidez e o grafismo imperam, será o *Assembly*. Lamentamos, mas não conhecemos as empresas do Porto onde poderá aprender linguagens de programação.

...mas sempre que entro no dBASE aparece uma imagem que não consigo evitar...

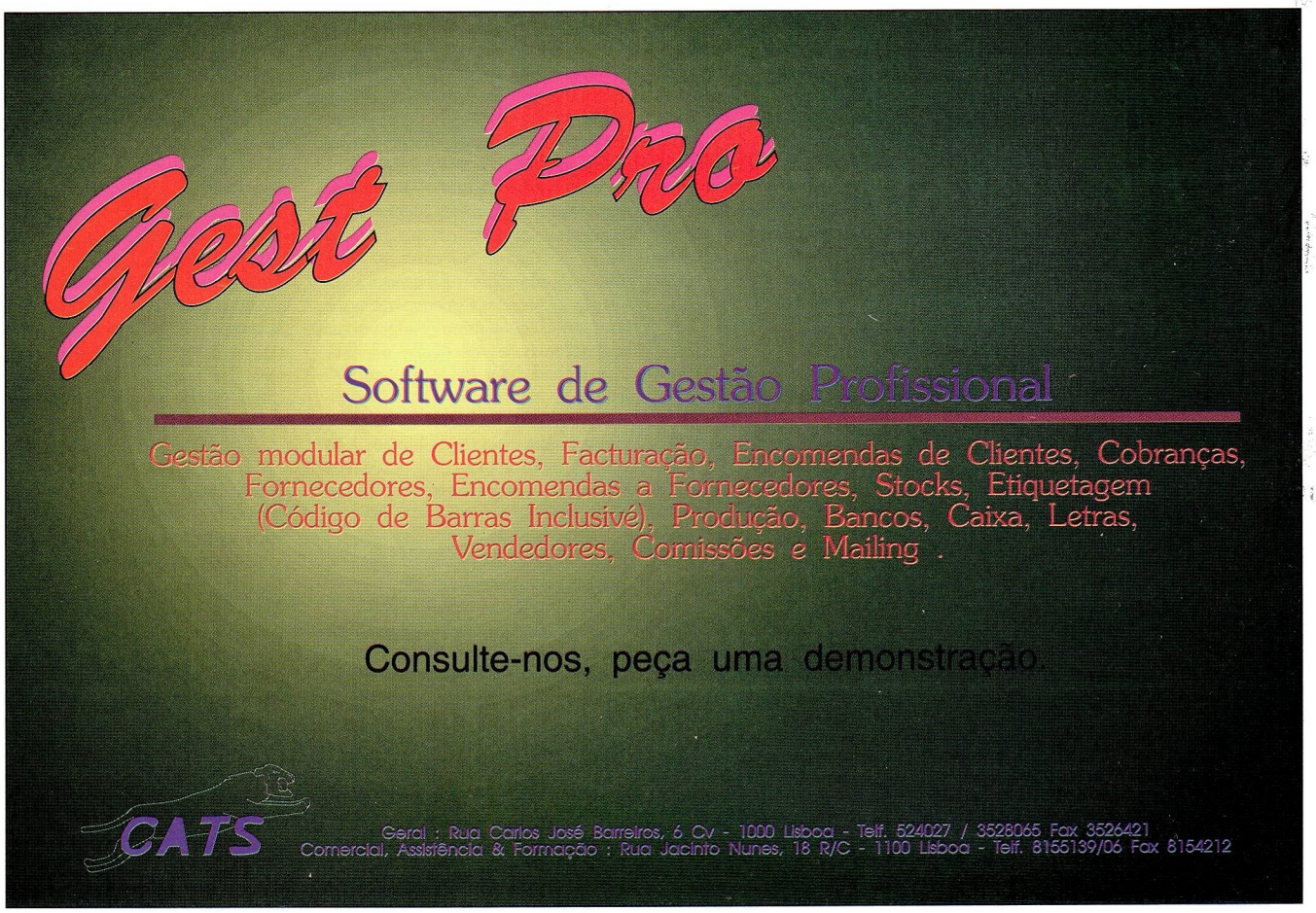
Ademar da Cruz C. Tavares/ Moncorvo

A imagem a que se refere é o ecrã de *Copyright* da Ashton Tate, empresa autora e proprietária do programa. Não se deve evitar, pois isso seria ignorar os termos contratuais aos quais demos o nosso acordo quando adquirimos o programa.

...na «Montra Spooler» estão à venda Joysticks com e sem placa. Estes podem funcionar sem placa ou esta é indispensável?

João P.C. Tomás/Vagos

Existem computadores que já vêm equipados com placa para jogos, normalmente compatíveis com a maioria dos Joysticks. Para os que não possuem essa placa, é necessário adquiri-la, de preferência em conjunto com o Joystick. A nossa resposta é a habitual nesses casos: consulte o vendedor do seu equipamento.



Spooler 386/33



A **Spooler** escolheu, importou e preparou para si um computador «topo de gama» na categoria 386.

Com componentes de primeira qualidade, devidamente testados, montámos uma poderosa máquina apta a satisfazer o utilizador mais exigente.

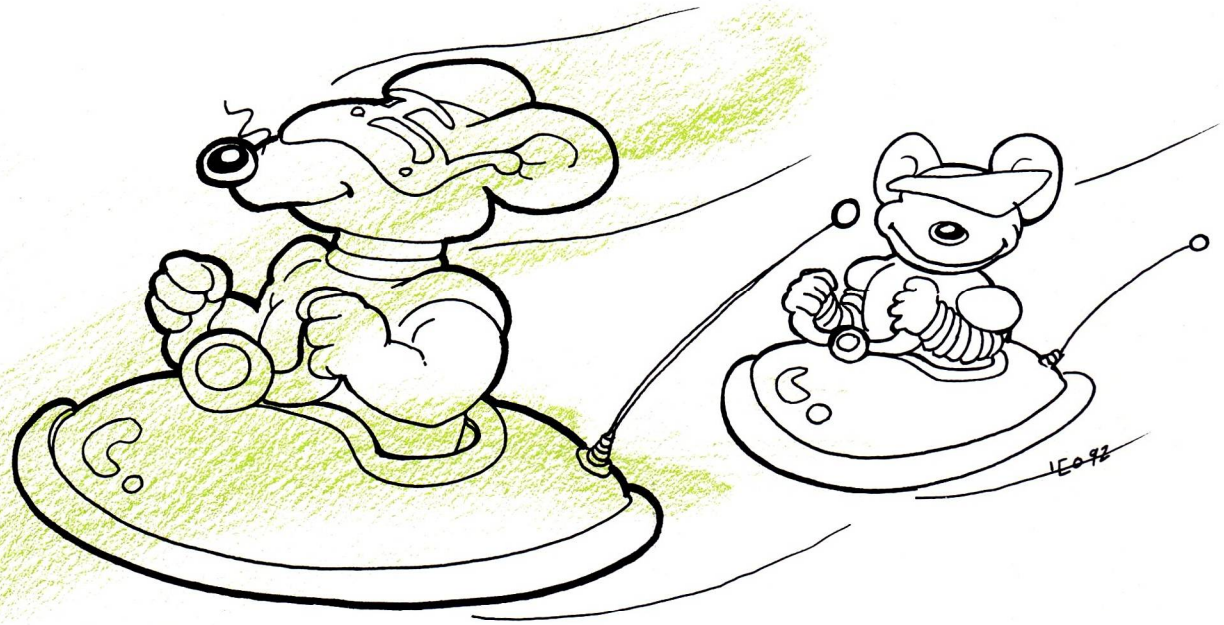
- Processador INTEL 30386 a 33 Mhz
- 4 Mb de RAM expansíveis até 64 Mb
- Memória «Cache» de 64 Kb
- Disco rígido Quantum de 100 Mb a 17 ms
- Drive de disquetas de 3,5" – 1,44 Mb
- Placa SuperVGA de 1 Mb – Monitor SVGA policromático
- Teclado em português
- DOS 5.0 – Versão integral em português com manual
- 2 anos de garantia
- Preço: 487 688$00 (16% IVA incluído)

OFERTA: «Rato» Primax e TODO o software «freeware e shareware da «Montra Spooler»

Reserve já o seu computador **Spooler 386/33**

Rua Duarte Pacheco Pereira, 69/A – Damaia-de-Baixo – 2700 AMADORA
Telefs.: 474 60 11/41 Fax: 475 22 89

HISTÓRIA E PROGRAMAÇÃO DE ROEDORES INFORMÁTICOS (PARTE III)



Serviço 1Dh – *Set CRT Page Number*

Escolhe a página da memória de video onde o cursor do **rato** deve ser escrito.

Chamada:
```
MOV   AX,1Dh
MOV   BX,página
INT   33h
```

Retorna: Nada

Nota : O número de páginas válidas depende do modo de video e da placa de video.

Serviço 1Eh – *Get CRT Page Number*

Retorna o número da página da memória de video onde o cursor do **rato** está a ser escrito.

Chamada:
```
MOV   AX,1Eh
INT   33h
```

Retorna: **BX** = página

Serviço 1Fh – *Disable Mouse Driver*

Desactiva o*driver* do rato e devolve o vector da interrupção **33h** antigo.

Chamada:
```
MOV AX,1Fh
INT 33h
```

Retorna: Se o serviço for bem sucedido
AX = 1Fh
ES:BX = vector da interrupção 33h
Se o serviço for mal sucedido
AX = FFFFh

Nota : Quando este serviço é usado, o *driver* do **rato** restaura quaisquer outros vectores que possa ter capturado, excluíndo o **33h** (pode incluir o **10h**, o **71h** e/ou o **74h**). Para restaurar o vector **33h** deve usar-se o endereço retornado, dado que este serviço destina-se a reinstalar o antigo vector na interrupção **33h**. Só então o processo de desinstalação estará completo.

Serviço 20h – *Enable Mouse Driver*

Activa o *driver* do **rato** e as interrupções provocadas pelo **rato**.

Chamada:
```
MOV AX,20h
INT 33h
```

Retorna: Nada

Serviço 21h – *Software Reset*

Inicializa o *driver* do rato e devolve o estado do *driver* . A única diferença entre este serviço e o **00h** é de que este serviço não inicializa o **rato** (a parte do *hardware*).

Chamada: **MOV AX,21h**
INT 33h

Retorna: Se o **rato** estiver operacional
AX = FFFFh
BX = Número de botões do **rato**
Se o **rato** não estiver operacional
AX = 21h

Nota : Para uma lista das acções deste serviço veja o serviço **00h.**

Serviço 22h – *Set Languages for Messages*

Escolhe a língua que será usada para as mensagens e erros do *driver* do **rato**.

Chamada: **MOV AX,22h**
MOV BX,número da língua

0 = Inglês
1 = Francês
2 = Holandês
3 = Alemão
4 = Sueco
5 = Finlandês
6 = Espanhol
7 = Português
8 = taliano

INT 33h

Retorna: Nada

Nota : Este serviço só está disponível nas versões internacionais do *driver* do **rato** Microsoft Mouse.

Serviço 23h – *Get Language Number*

Devolve o número da língua que é usada pelo *driver* do **rato** para as mensagens e os erros.

Chamada: **MOV AX,23h**
INT 33h

Retorna: **BX** = nº da língua (ver serviço **22h)**

Nota : Este serviço só está disponível nas versões internacionais do *driver* do **rato** Microsoft Mouse.

Serviço 24h – *Get Driver Version, Mouse Type and IRQ* Number

Devolve a versão do *driver* do **rato**, o tipo do **rato** e o número da linha de interrupção usada pelo **rato** (número **IRQ**).

Chamada: **MOV AX,24h**
INT 33h

Retorna: **BH =** parte inteira do nº da versão
BL = parte fraccionária do número da versão
CH = tipo de **rato**

1 = **rato** cominterface debus
2 = **rato** com interface série
3 = **rato** com interface InPort
4 = **rato** PS/2
5 = **rato** HP (Hewlett Packard)

CL = número **IRQ**

0 = PS/2
2,3,4,5 ou 7 = número **IRQ**

Conforme os leitores mais experientes já terão reparado, as descrições dos processos de chamada dos serviços são constituidas por instruções de *Assembly* 80x86. Para os leitores que queiram experimentar estes serviços com a linguagem C, podem utilizar o seguinte grupo de instruções (este é um dos possíveis métodos) :

```
struct REGPACK registo;
registo.r_(nome do registo a usar, que pode
ser dos seguintes:
AX,BX,CX,DX,DI,SI,BP,ES,DS
e FLAGS) = valor;
.....
intr(0x33,&registo);
```

Como exemplo vou usar o serviço **1Ah**, que pode ser chamado, usando a linguagem C, com as seguintes instruções :

```
struct REGPACK registo;
registo.r_ax = 0x1A;
registo.r_bx = <deslocamento horizontal>;
registo.r_cx = <deslocamento vertical>;
registo.r_dx = <limite de velocidade dupla>;
intr(0x33,&registo);
```

Os valores retornados podem ser obtidos lendo os elementos da estrutura «registo», pois cada elemento da estrutura corresponde a um registo do CPU. A estrutura do tipo REGPACK é definida no ficheiro «dos.h» do Turbo C ou Turbo C++, podendo ser necessário usar outros métodos com outros compiladores.

O FUTURO

Muitos dos **ratos** de fabricação mais recente possuem uma capacidade denominada «aceleração variável» (em inglês também é usado o termo *ballistic tracking*), que se traduz na capacidade do **rato** medir a aceleração a que é sujeito. Quando deslocamos o **rato** lentamente, a sua resolução será a normal. Quando deslocamos o **rato** rapidamente, a aceleração é detectada e medida, o que provoca um aumento na resolução do **rato**, deslocando-se o cursor muito mais rapidamente (quanto mais alta for a resolução, mais pontos existem por polegada, logo, tendo uma relação *mickeys*/pixeis fixa, quantos mais *mickeys* (pontos por polegada) forem detectados, mais o cursor se desloca no ecrã).

A vantagem desta capacidade é de evitar (pelo menos em grande parte) os grandes deslocamentos do **rato** que a todos nos enfurece (ninguém gosta de ter de gesticular como um doido para fazer deslocar um minúsculo cursor dum canto do ecrã para outro).

Quanto a evoluções, não me parece que iremos assistir a grandes alterações no desenho e funcionamento dos **ratos** do futuro, pois como se costuma dizer «em equipa que vence não se mexe» e não há dúvidas que o **rato** realmente venceu a sua luta pela aceitação.

Carlos Azevedo

“Muitos dos ratos de fabricação mais recente possuem uma capacidade denominada «aceleração variável»”

PROGRAMAR EM ASSEMBLY (PARTE I)

Muitos utilizadores e a totalidade dos programadores deparará,tarde ou cedo, com a necessidade de resolver um problema específico de codificação para o qual as linguagens de alto nível não oferecem qualquer solução ou, se oferecem, esta não é satisfatória.

PORQUÊ PROGRAMAR EM *ASSEMBLY* ?

Muitos utilizadores de computadores pessoais conhecem linguagens de programação como o BASIC, o PASCAL ou o C e dão-se por muito satisfeitos. Na realidade, para muitos objectivos de programação, ferramentas como essas podem ser, e são normalmente, mais do que suficientes, ou dito de outro modo, são efectivamente as mais adequadas.

Essas linguagens de programação são designadas por linguagens de «**alto nível**», o que significa que quem as utiliza programa a um elevado nível de distanciamento do equipamento. Outra forma de as designar é por linguagens «**procedurais**» pois permitem-nos descrever as tarefas a serem realizadas de forma orientada directamente para a resolução do problema.

Para elas não há qualquer preocupação com instruções aceites a nível do processador, registos internos e tudo que tenha a ver com a arquitectura particular de um computador.

Na verdade é possível programar em linguagem de alto nível e ter um conhecimento não muito pormenorizado do computador com que se está a trabalhar.

As linguagens de alto nível têm ainda a vantagem de ser relativamente «portáveis» entre computadores de arquitecturas diferentes, se é que se pode falar de portabilidade satisfatória nos nossos dias em que os programas tendem a ser grandes e complexos procurando tirar o maior partido dos computadores em que irão ser executados. Mas, como o leitor já adivinhou, nem tudo são rosas com as linguagens de alto nível. Se assim fosse, não só este artigo não faria sentido, como não se venderiam mundialmente milhares de obras sobre a outra linguagem, a linguagem de **baixo nível.**

Muitos utilizadores e a totalidade dos programadores deparará,tarde ou cedo, com a necessidade de resolver um problema específico de codificação para o qual as linguagens de alto nível não oferecem qualquer solução ou, se oferecem, esta não é satisfatória. A frequência de aparecimento desses problemas não é tão invulgar como à primeira vista possa parecer. Para um programador de aplicações de gestão podem aparecer ocasionalmente mas, para um programador de sistemas operativos, aparecerão a todo o momento. E a sua resolução passará normalmente por possuir um elevado conhecimento dos «meandros» de como o equipamento funciona e de como actuar sobre ele. A melhor via para realizar essa aprendizagem é começar a estudar a única linguagem que o computador entende.

Ora bem, mas todos sabemos que os computadores internamente – os seus componentes físicos e circuitos electrónicos, todo esse conjunto que se designa por *hardware*, só conhecem uma linguagem que é constituída por zeros e uns, a **linguagem binária ou código máquina.**

Felizmente, não nos devemos alarmar, pois longe vão os dias em que se programava directamente em binário. Nos nossos dias já ninguém o faz!

Um programa binário é praticamente inintelígivel para nós humanos, mesmo que contenha apenas 2 ou 3 instruções. Mas nos primeiros tempos da era dos computadores, nos anos 40 e 50, houve quem o fizesse, naqueles grandes computadores de válvulas que já passaram à História.

Tentou-se em alternativa programar em hexadecimal, que é uma base de numeração em que se começa a contar no zero e se termina no «F» (que significa 15 em numeração decimal). Base esta que tem uma relação simples com a base binária, pois por cada grupo de 4 dígitos binários temos 1 em hexadecimal. Porém, a programação em hexadecimal também era extraordinariamente complicada e demorada.

Finalmente encontrou-se a solução, o *Assembly*.

O *Assembly* é uma linguagem de símbolos, designados **mnemónicas**.

Cada mnemónica tem a sua correspondência em um comando elementar inteligível pelo computador. Cada mnemónica é uma **instrução** *Assembly*.

Normalmente todos os comandos que podem ser dados em binário podem ser dados em *Assembly* com a mesma eficiência. Mas talvez seja melhor, neste momento, dar uma pequena espreitadela ao quadro seguinte, sem qualquer preocupação especial de compreender:

Endereço	Hexadecimal	Instrução em *Assembly*
0600:0100	B4	MOV AH,20
0600:0101	20	
0600:0102	B7	MOV BH,30
0600:0103	30	
0600:0104	00	ADD AH,BH
0600:0105	FC	
0600:0106	80	CMP AH,B0
0600:0107	FC	
0600:0108	B0	
0600:0109	75	NZ 104
0600:010A	F9	
0600:010B	B4	MOV AH,4C
0600:010C	4C	
0600:010D	CD	INT 21
0600:010E	21	

Certamente que, se o leitor nada sabia de *Assembly*, continou sem saber após a visulização do quadro, mas não fique preocupado por causa disso, pois «Roma e Pavia não se fizeram num dia» e, no seguimento deste artigo e

nos seguintes, tudo começará a fazer sentido. O quadro que acabou de ver apresenta-nos um extracto de um programa muito simples.

O formato do quadro é algo parecido com o que nos apresenta o programa DEBUG que acompanha o sistema operativa MS-DOS quando carregamos na tecla 'U'. Mas vamos explicar um pouco o quadro:

Os valores da coluna de titulo **Hexadecimal** são directamente intelígiveis, ou melhor, o seu equivalente na base binária é directamente inteligível por um computador da família IBM PC. As mnemónicas *Assembly* da coluna **Instrução em *Assembly*** tem o mesmo objectivo, mas são inteligíveis por nós humanos, e apenas por nós. O programa acima executa como se segue:

1) Mover (note o **MOV** na coluna de instruções em *Assembly*) para um **registo** designado por **AH** o valor 20 (hexadecimal). E aqui o leitor pensará, e com razão, que não só não sabe o que é um registo como **AH** não lhe diz nada. O assunto será desenvolvido a seu tempo mas, por hoje, fique com a ideia que a UCP (Unidade Central de Processamento) do computador é dividida em pequenas unidades de armazenagem designadas por registos. Os registos são de 16 bits (2 bytes) nos processadores 8086 e 80286. Todos têm um nome.

Um deles chama-se Accumulator e recebe a abreviatura **AX**.

O registo **AX** pode ser controlado por inteiro ou por metades. Se forem os primeiros 8 bits designa-se registo **AL** (*Accumulator Low*), se forem os últimos designa-se por **AH** (*Accumulator High*). Repare que não se utilizou o termo subregisto.

Outro registo designa-se por **BX** (*Base Register*) e o que se disse para o **AX** aplica-se a ele, isto é, pode ser considerado dividido em dois registos de 8 bits, o **BH** e o **BL.**

Mas continuando a nossa análise:

2) Mover também, mas agora o valor 30 (hexadecimal) para o registo **BH**.

3) Somar (Diz-se **ADD** em inglês) o conteudo do registo **AH** com o conteúdo do registo **BH** e deixar o resultado em **AH.**

Naturalmente que na primeira iteração **AH** ficará com o valor 50 (hexadecimal).

4) Agora comparamos (Note **CMP** de *Compare*) se o conteúdo do registo **AH** é igual ou não a **B0** (corresponde a saber se é igual a 176 em decimal).

A comparação é feita subtraindo **AH** de **B0** sem devolver o resultado, mas apenas sinalizando para um registo especial de nome ***flags*** o que se passou. O registo ***flags*** é também de 16 bits e cada um desses bits ora fica a **0** ora a **1** consoante o que se passou na instrução que acabou de ser executada. No nosso caso se houver igualdade de **AH** com **B0** o sétimo bit do registo ***flags*** e que é designado por ***zero flag*** assume o valor **1**.

5) Chegamos agora a um ponto de decisão. Se a comparação não der igual saltamos de novo para a instrução existente no endereço 104 (e que já vimos qual era) para ser executada de novo. Se a comparação der igual seguimos em frente e executamos as duas instruções seguintes que terminam o programa e nos trazem de novo para o *prompt* do DOS.

A instrução **JNZ** (*JUMP IF NOT ZERO*) actuou, por conseguinte,verificando no registo ***flags*** se o resultado da instrução anterior foi zero, isto é se o conteudo de **AH** é igual a **B0**.

Experimente transcrever as mnemónicas acima para o DEBUG teclando 'A' e escrevendo-as uma a uma. Não se preocupe com os valores constantes das colunas de endereços ou de símbolos hexadecimais, mas apenas com as mnemónicas. Execute o programa em «câmara lenta» teclando sucesivamente 'P' e verifique que os valores de alguns registos se alteram após cada instrução.

Pode gravar o programa para uso posterior ou para executá-lo do *prompt* do DOS, e fazer muitas outras coisas com o DEBUG, pelo que vale bem a pena ler o manual do MS-DOS para ver como trabalha esta pequena pérola.

É possivel que, no decorrer das suas experiências com o DEBUG, o computador ocasionalmente «pendure» e tenha de carregar na tecla de *Reset.*

Normalmente isso é inconsequente, pois sei de pessoas que destruiram os dados do disco com outros programas que não o DEBUG (não conheço nenhuma mas é natural que existam).

Tenha contudo em atenção que alguns comandos do DEBUG são extremamente perigosos e não deverão ser utilizados sem uma perfeita compreensão da sua acção e do que se pretende atingir. Estou a lembrar-me de um, o que permite a escrita em sectores absolutas do disco – se possível nunca o use.

A aprendizagem das instruções de *Assembly* é fácil pois as mnemónicas estão organizadas por classes de instruções de código máquina de um modo que faz sentido para nós humanos. Por exemplo, todas as acções que implicam um movimento de bytes de um local de armazenamento para outro representam-se pela mnemónica **MOV** seguida de qualquer outra coisa.

O **MOV** pode ser de transferência entre registos do processador, de transferência da memória para os registos ou vice-versa, de carregamento de um valor constante para um registo, etc. Deste modo, com o conhecimento de umas poucas dezenas de mnemónicas, podemos realizar quase qualquer programa em *Assembly*.

E fique desde já consciente que a mnemómica mais usada é precisamente o **MOV** e que alguns programas chegam a ter mais de 50% de **MOV**'s.

Outra mnemónica muito usada é **ADD** e que já vimos também.

A mnemónica **CMP** é fundamental para a instrução que se lhe segue e que é normalmente uma instrução de «salto condicional para».

As instruções de salto condicional actuam verificando o que se passa nas ***flags*** . No nosso exemplo, e como se viu, foi o bit ***zero flag*** do registo ***flags*** que foi testado.

Uma ideia muito difundida, mas largamente errada, é a de que a programação em *Assembly* é algo que se assemelhará muito a construir um edificio, tendo que cuidar pessoalmente da colocação de cada tijolo, de cada prego, de cada janela, de cada porta, enfim, de cada pormenor.

Nada mais errado: a programação em *Assembly* é bastante produtiva, como irá reconhecer se continuar a seguir estes artigos.

Nesta família de computadores, o que torna o *Assembly* tão produtivo é que nenhuma outra linguagem consegue fazer um uso tão eficiente das centenas de rotinas já feitas e prontas a ser usadas e que se encontram à espera de serem invocadas dentro do nosso computador.

Um grupo dessas rotinas faz parte do BIOS (*Basic Input Output System*) e reside fisicamente no ROM do computador. O outro grupo vem com o sistema operativo MS-DOS. Tanto umas como outras estão amplamente documentadas em muitos livros.

Ainda em relação ao nosso mini-programa acima note o modo airoso como este é terminado:

“A aprendizagem das instruções de *Assembly* é fácil pois as mnemónicas estão organizadas por classes de instruções de código máquina de um modo que faz sentido para nós humanos.”

```
MOV AH,4C
INT 21
```

São 4 bytes e o DOS encarrega-se de tudo quanto é necessário para arrumar a casa antes de libertar a memória e lhe apresentar de novo o *prompt*. Podiamos ter terminado o programa de outros modos, por exemplo apenas com 2 bytes:

```
INT 20
```

O que no nosso caso serviria bem, pois não havia grandes arrumações a fazer, designadamente fechar outros programas abertos de dentro do nosso.

Tanto num caso como noutro invocamos rotinas do MS-DOS para executar o trabalho.

Contudo note que o *Assembly* não se substitui às linguagens de alto nível nem estas se substituem ao *Assembly*: cada qual tem o seu âmbito próprio e vamos ver qual é.

“As linguagens de alto nível não foram concebidas com o objectivo de dar resposta a toda e qualquer necessidade.”

QUANDO, ONDE E O QUE PROGRAMAR EM *ASSEMBLY* ?

De um modo geral, quanto mais se souber do computador com o qual se está a trabalhar, mais eficiente será o nosso trabalho de produzir programas.

As linguagens de alto nível não foram concebidas com o objectivo de dar resposta a toda e qualquer necessidade. Quando estivermos em dúvida devemos sempre seguir a velha regra que diz que, para qualquer problema específico, os meios ideais são os globalmente mais vantajosos.

Se conseguirmos resolver o problema a contento com uma linguagem como o BASIC, devemos fazê-lo sem qualquer hesitação. Ao fim e ao cabo é rápido escrever um programa em BASIC.

Uma instrução em BASIC corresponde normalmente a dezenas ou centenas de instruções em *Assembly*. Existem programas-ferramenta chamados «compiladores» que transformam as instruções que escrevemos em BASIC (referimo-nos naturalmente a versões de BASIC que suportam compilação), e que constituem o **código fonte**, em instruções de binário directamente inteligíveis pelo computador e ainda acrescentam tudo o que é necessário para o programa poder ser carregado em memória e executado. Porém, existem programas que não devem ser escritos, pelo menos integralmente em BASIC, e quem diz BASIC, diz PASCAL, ‘C’ ou qualquer linguagem de alto nível, designadamente:

1) Se existir necessidade de limitar o tamanho dos programas.

Isto, porque os compiladores nunca produzem código muito eficiente, existe muita redundância, muito código que nunca chega a ser executado.

Por exemplo, os programas destinados a ficar residentes em memória após passarem o controle para o DOS, e designados por TSR (do inglês *Terminate and Stay Resident)*, deverão ser o mais curtos possíveis.

Numa linguagem de alto nível altamente compacta como o ´C´, é praticamente impossível construir um TSR, mesmo muito rudimentar, que ocupe para si um espaço de memória inferior a 15 KB, e este valor crescerá muito rapidamente se falarmos de PASCAL ou BASIC.

Em *Assembly* o mesmo TSR pode ocupar pouco mais de 256 bytes e não ocupa menos porque o MS-DOS reserva para si exactamente 256 bytes no ínicio de cada programa numa zona designada por **PSP** (*Program Segment Prefix*). Poderia eventualmente ocupar até menos de 256 bytes, mas isso obrigaria a entrar no terreno do **PSP,** pelo que o leitor fica desde já alertado que não é pratica recomendável.

2) Se for necessário que o programa seja rápido e atender a uma exigente temporização da sua execução.

Os programas de comunicações e todo o tipo de aplicações em que exista necessidade de apertado controle em tempo real.

3) Se houver necessidade de controlar o *hardware*, de dialogar directamente com os *chips*, de espreitar continuamente para dentro da memória e alterar os seus valores, de fazer uso directo das muitas rotinas que o DOS e o ROM-BIOS põem á disposição do programador.

E mesmo que para tal existam soluções nas linguagens de alto nível, deve-se ter em atenção que, se o programa que estiver a desenvolver se destinar a ser comercializado e a concorrência apresentar um programa curto e rápido que faça o mesmo que o seu longo e lento programa, isso pode causar-lhe várias dificuldades. E aqui surje a necessidade da programação em *Assembly*.

O *ASSEMBLY* É UTILIZADO EM DUAS VERTENTES:

1) Programas integralmente realizados nessa linguagem.

Normalmente são programas pequenos, raramente com mais de 10 KB.

Existem contudo excepções a essa regra, inclusivé programas com centenas de kilobytes.

2) Criação de rotinas ou módulos que podem ser ligadas com programas em linguagem de alto nível.

Normalmente essas rotinas executam acções críticas cuja adequada condução não pode ser realizada pelo reportório de instruções da linguagem de alto nível.

E, de facto, a importância da linguagem *Assembly*, ao contrário do que seria de esperar, é cada vez maior nos nossos dias. A maioria dos compiladores modernos possuem *interface* para ligação de módulos escritos em linguagem *Assembly* ,quer directamente quer após a sua passagem a **código objecto** (ver mais adiante a referência a **OBJ**).

E os compiladores mais recentes vão um passo mais adiante e permitem a incorporação directa de instruções em *Assembly* no seio de programas fonte escritos na linguagem de alto nível. Incluem-se neste grupo as recentes versões do Turbo Pascal e Turbo ‘C’.

E antes de continuar uma pequena nota:

Alguns autores classificam algumas linguagens, incluindo a linguagem ‘C’, por linguagem de «médio nível». Sem discutir a correcção do termo, com o qual pessoalmente concordamos, confirmamos que na realidade as linguagens de médio nível substituem em muitas situações o *Assembly*.

O sistema operativo MS-DOS e ambientes de lançamento de programas como o Windows foram, em larga escala, escritos em ‘C’. Contudo, essas linguagens de médio nível não substituiram integralmente a linguagem *Assembly*, e as instruções que possuem para execução a baixo nível são, quase sempre, mais complicadas que as mnemónicas de *Assembly*, e, curiosamente, não dispensam o seu conhecimento.

AS FERRAMENTAS PARA PROGRAMAR EM *ASSEMBLY*

Se a sua decisão neste momento for de que vale a pena aprender *Assembly*, então deverá ter ao seu dispor um

certo conjunto de ferramentas.

Nada de muito complicado nem dispendioso.

1) A ferramenta base designa-se por **Assemblador**, ou *Assembly* em inglês (os brasileiros utilizam o termo «montador»).

Mais do que entrar em análises semânticas, o importante aqui é saber que o Assemblador é para o *Assembly* o que o Compilador é para uma linguagem de alto nível.

Muita gente sabe que o MS-DOS é distribuído com um pequeno programa designado por DEBUG, mas poucos aprendem a trabalhar com ele, ou sabem sequer para que serve. Bem, o DEBUG é o programa mais poderoso dos utilitários que acompanham o MS-DOS, um autêntico prodígio de poder.

Já atrás falámos do DEBUG e vimos que contém um Assemblador, mas ficamos por aqui. O DEBUG embora valente, não passa de um «canivete suiço», uma ferramenta multiuso mas primitiva.

No campo dos verdadeiros Assembladores começamos por citar um que é de *shareware* e de nome A86 (que significa, segundo o seu autor, «Assemblador para a família de processadores 80x86 da Intel»).

É extremamente fácil aprender a trabalhar com o A86, pois ele dispensa quase totalmente o conhecimento prévio daquilo que se designa por «pseudo-operadores». Estes não são instruções *Assembly*, mas directivas para os Assembladores e são a causa número um de desânimo para muitos que tentam iniciar-se no estudo do *Assembly*.

Exemplo de pseudo-operadores são os célebres ASSUME, os PROC, os ENDP, etc.

E como o A86 demonstra muito bem, é possível construir programas algo sofisticados sem utilizar qualquer desses quebra-cabeças. Se os quiser utilizar muito bem, o A86 conhece-os também, senão, confie no bom senso do A86 que ele procede como se os pseudo-operadores lá estivessem e normalmente acerta.

O A86 tem muitos pontos a seu favor e, naturalmente,algumas pequenas limitações. Mas, em nossa opinião, um principiante terá toda vantagem em conhecer este programa-ferramenta, pois permitir-lhe-á um evoluir rápido para um conhecimento bastante elevado de linguagem *Assembly*. E, após esse conhecimento ser obtido, o tema dos pseudo-operadores torna-se muito mais fácil, quase intuitivo.

No âmbito dos Assembladores ditos profissionais devemos citar os mais conhecidos: O MASM e o TASM. O MASM é produto da Microsoft e considerado implicitamente a referência que qualquer dos outros Assembladores não pode perder de vista. Eles podem superar aqui e ali o MASM mas, acima de tudo, têm de poder afirmar que são 100% (ou quase) compatíveis com o MASM. Mesmo que não gostemos do MASM, devemos conhecê-lo, pois ele constitui a linguagem comum que todos temos de ter em qualquer área do conhecimento e o *Assembly* não é excepção.

O MASM foi, durante muitos anos, praticamente o único Assemblador profissional disponível no mercado.

O TASM surgiu em 1988, é um produto da Borland Internacional e, em nossa opinião, o melhor existente actualmente. Curiosamente, tanto o TASM como o MASM, nas versões mais recentes, têm aliviado bastante as suas exigências em matéria de directivas requeridas pelo Assemblador e aceitam uma forma simplificada dessas directivas baseada nos «modelos de memória» das linguagens de alto nível, forma essa que pode ser usada na grande maioria dos programas.

Os Assembladores actuam sobre o código *Assembly*, isto é, mnemónicas *Assembly* como as que vimos atrás e que escrevemos com um simples editor de texto como o EDIT do MS DOS 5.0, ou qualquer outro que produza texto não formatado, ou ASCII puro. Designa-se esse código por **programa fonte** (do inglês *source code*). Quando o MASM ou o TASM operam sobre esse programa, o resultado não é de imediato um programa executável com extensão COM ou EXE, mas sim um programa intermédio designado por **programa objecto** e que tem normalmente a extensão OBJ.

O A86, por seu lado, pode produzir também OBJ, mas por omissão construirá de imediato um executável COM, ou em alternativa um BIN, que é um programa binário puro que é carregado num endereço de memória de deslocamento zero. Devido à existência dos OBJ, é necessário termos outra ferramenta que é o ***linker***, ou ligador (ou «linkeditor» se o leitor for brasileiro).

2) O linker.

O MASM tem o seu *linker* de nome LINK, e o TASM tem o TLINK. Se tiver um OBJ produzido pelo A86, então peça «emprestado» um ou outro para **ligar** o programa. A existência de programas OBJ é uma «fatalidade» que tem a ver com:

– O programa pode ter referenciado rotinas ou variáveis de outros programas-módulo, ou funções que existem em bibliotecas de rotinas (*libraries* em inglês) e é necessário ligar tudo isso num único programa para que este possa funcionar.

– A estrutura de memória dos computadores baseados no 80x86 permite que os programas sejam carregados em qualquer ponto dessa memória para serem executados. Assim, outra função do *linker* é fabricar um cabeçalho (ou *Header* em inglês) que oriente o DOS no carregamento e relocação das várias partes do programa em memória. Deste modo, o resultado do trabalho do *linker* acaba por ser um programa com a extensão EXE, que congrega tudo o que é necessário para poder ser executado.

Mas alguns programas EXE têm a característica de não terem exigências particulares em matéria de relocação, pois a sua imagem em memória pode ser idêntica à sua imagem em ficheiro (*file* ou ficha são termos comuns, mas usaremos ficheiro por estar mais divulgado).

Num caso desses poderemos fazer uso de outra das ferramentas que é o:

3) EXE2BIN.

Este programa transforma (quando é possível) os ficheiros EXE em COM (ou noutras extensões mais arrevezadas como SYS, mas não compliquemos desnecessárimente nesta fase). Se dispuser do TLINK, o EXE2BIN não é necessário, pois basta usar o *switch* /t do TLINK para que o ficheiro venha com a extensão COM. Se o nome EX2BIN não lhe é estranho e nunca utilizou *Assembly*, talvez isso se deva a que algumas versões do MS-DOS trazem esse programa incluido. Porquê só o EXE2BIN acompanhar o MS-DOS, porquê às vezes o EXE2BIN e o LINK e outras vezes nem um nem outro, é um mistério não revelado pela Microsoft.

Que outras ferramentas existem ? Algumas outras, mas estas chegam para começar.

No artigo de hoje cobrimos um largo território para quem à partida muito pouco ou nada soubesse de *Assembly*. Mas toda a panorâmica que demos irá ajudá-lo muito nos próximos números, nos quais falaremos de arquitectura dos PC, segmentação de memória , interrupções, serviços do DOS e do BIOS e, simultâneamente, iremos aprofundando os nossos conhecimentos de *Assembly* sempre devidamente ilustrados com programas exemplo.

José Páscoa

“Mais do que entrar em análises semânticas, o importante aqui é saber que o Assemblador é para o *Assembly* o que o Compilador é para uma linguagem de alto nível.”

A Utilização da Memória Extra do Computador

“Independentemente da quantidade de memória que esteja instalada no computador, o DOS apenas reconhece e utiliza 640K.”

Olá leitores.

Eis que regressamos ao tema da utilização da memória do computador mas desta vez abordando o tema sob o ponto de vista do programador e demonstrando como se pode, através de um programa, aceder à expansão de memória. O acesso à *expanded memory* obedece a uma estrutura com regras bem definidas, denominada LIM\EMS.

A especificação LIM/EMS (Lotus/Intel/Microsoft *Expanded Memory Specification*) foi desenvolvida para permitir aos processadores da família 8086/88 (e 286/386 operando em modo real) aceder a uma quantidade de memória superior à físicamente endereçável (ver caixa).

Apesar de correrrmos o risco de uma certa repetição – este assunto já foi abordado na **Spooler** número 9 – não queremos deixar de enunciar aqui alguns conceitos essenciais para os nossos leitores que não tiveram acesso ao referido número da nossa revista.

Como se sabe, independentemente da quantidade de memória que esteja instalada no computador, o DOS apenas reconhece e utiliza 640K – é a chamada memória convencional (também referida em alguma literatura pelas designações *conventional memory* ou *base memory*). Quando apareceram os primeiros computadores IBM PC, com a sua arquitectura baseada no CPU 8086 da Intel, os projectistas dividiram o espaço de endereçamento do CPU (1Mb) em duas fatias: uma de 640Kb para memória convencional do DOS – onde seriam carregados e executados os programas – e uma segunda fatia, compreendida entre 640Kb e 1Mb e designada por *high*-DOS, totalmente reservada para acesso a *cards* vídeo, controladores de disco, *cards* de rede, etc.

Entretanto, as arquitecturas evoluiram: a Intel criou o CPU 80286, com capacidade de endereçamento de 16Mb (1Mb convencional e 15Mb *extended memory*) e a IBM lançou a gama de computadores AT baseada neste microprocessador. O CPU 286 possui dois modos de funcionamento: o modo real, em que é 100% compatível com o 8086, endereçando apenas 1Mb de RAM e o modo protegido, em que endereça directamente aos 16Mb. É, no entanto, em termos de sistema operativo que surgem os problemas: enquanto, por exemplo, o OS/2 e o UNIX correm em modo protegido, beneficiando dos 16Mb de memória disponível, o DOS (por questões de compatibilidade de *software*) corre apenas em modo real. Para o DOS, a *extended memory* dos AT não é endereçável, podendo apenas ser usada por um restrito número de programas, como sejam o VDISK (que implementa um RAM-disk) e alguns *caches* (como, por exemplo, o PC-CACHE da Central Point).

Com o aparecimento dos 80386 e 80486, as capacidades de endereçamento subiram para 4Gb mas as limitações do DOS permaneceram no mesmo ponto: manteve-se fiel aos seus 1024Kb!

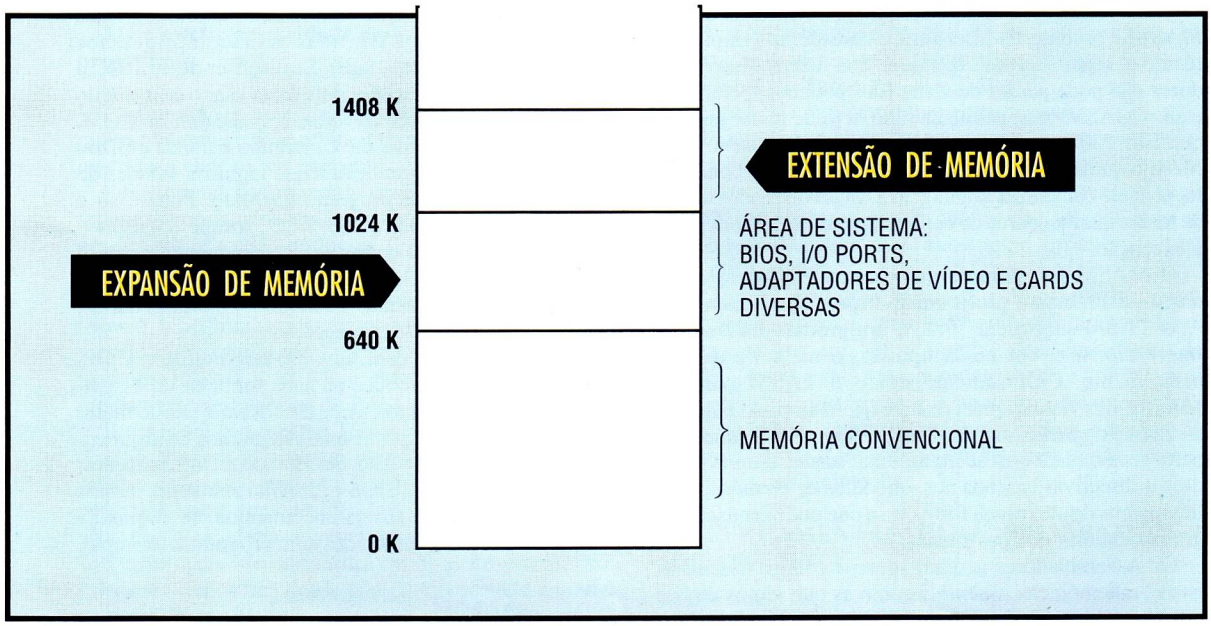


Figura 1

É neste cenário que surge a extensão de memória. A EMS foi implementada para proporcionar aos CPU's operando em modo real o acesso a uma determinada quantidade de memória adicional. Esta especificação requer *hardware* externo (uma *card* instalada no PC onde está colocada a RAM adicional) e um programa (*device driver*) para gerir os recursos da placa e comunicar com os programas que requisitam memória. Refira-se que a expansão de memória, por si só não traz qualquer benefício aos programas em geral, mas sómente aos que foram escritos específicamente para tirarem partido da sua existência.

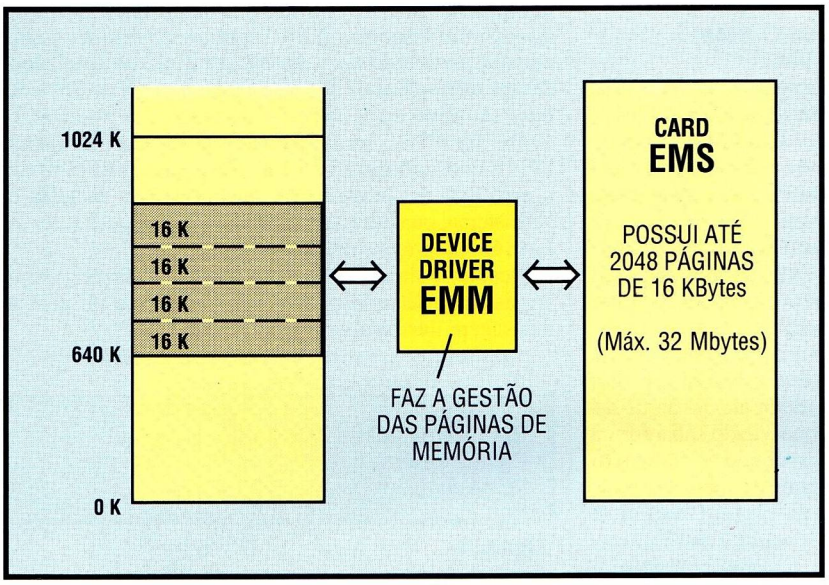


Figura 2

Inicialmente designada por LIM/EMS, como já referimos, esta especificação disponibilizava até 8Mb de RAM adicionais. Actualmente, a especificação é designada por LIM/EEMS (para *Enhanced Expanded Memory Specification*), permitindo aceder até 32 Mb adicionais.

Cabe aqui um parêntese para esclarecer uma dúvida que possivelmente poderá já ter surgido: então se este artigo se refere à forma de utilizar a expansão de memória, tudo o que aqui for tratado não vai ser passível de aplicação em computadores que tenham apenas a vulgar *expanded memory*...

Esta questão é pertinente e foi exactamente como resposta a ela que decidimos abordar também nestas linhas a utilização de um programa que emula uma expansão de memória à custa da extensão de memória!

Quer isto dizer que, mesmo sem ter a tal placa, mas com o *device driver* EMS40.SYS instalado, o DOS «vê» a placa e tem pleno acesso à memória nela «instalada». Assim sendo, pensamos que este artigo vai interessar a todos os leitores: os felizes possuidores da placa de expansão de memória vão ficar a saber como lhe podem aceder através dos seus programas; quem não tem a dita placa mas possui um 286 (ou superior) com mais de 640Kb de RAM vai aprender a instalar um *device driver* que usa a memória acima dos 640Kb para simular a existência da placa, podendo usá-la exactamente da mesma forma. Refira-se ainda que o DOS 5.0 fornece já um *device driver* para os computadores 386 e superiores, designado por EMM386.EXE, que implementa exactamente esta emulação e que pode ser acedido exactamente da mesma forma que o EMS40.SYS.

VAMOS SABER MAIS SOBRE A EXPANSÃO DE MEMÓRIA

A expansão de memória é um subsistema constituído por um módulo de *hardware* (uma placa onde residem os *chips* da RAM adicional) e um módulo de *software* designado por EMM (o EMM – *Expanded Memory Manager* é um programa gestor do acesso à memória disponível na placa que é carregado pelo DOS durante a operação de *boot*, sob a forma de um *device-driver*). A instalação física da expansão é a coisa mais simples do mundo: coloca-se a *card* no computador, numa *slot* livre, e copia-se para o disco de *boot* o *device-driver*, alterando o CONFIG.SYS para que o EMM passe a ser instalado quando o sistema «arranca». Se as coisas correrem bem, ao fazer *boot* obter-se-á uma mensagem tranquilizadora do EMM dizendo que está instalado correctamente. Se as coisas correrem mal, há que tentar saber porquê (recorrendo ao manual da *card* de expansão) e tentar resolver o problema. A fonte de problemas mais trivial é um conflito de endereços entre a *card* de expansão e alguma outra *card* já instalada no computador. Este contratempo é normalmente resolvido com a alteração de um ou mais *jumpers* na *card* de expansão, alterando o endereço de acesso à *card* para uma zona de memória livre.

> **“A expansão de memória é um subsistema constituído por um módulo de *hardware* (uma placa onde residem os *chips* da RAM adicional) e um módulo de *software* designado por EMM .”**

Quanto ao funcionamento da engenhoca própriamente dita, a ideia é muito simples, como vão ter oportunidade de perceber. Pelo facto de o CPU só poder endereçar até 1Mb, o acesso à expansão de memória é feito através de uma artimanha. Ao instalar-se, o EMM reserva na zona *high*-DOS (entre os 640Kb e os 1024Kb) um espaço contíguo de 64K; esta área, designada por janela, está dividida em 4 páginas de 16Kb. Cada uma destas páginas pode ser endereçada individualmente e colocada em qualquer local da *expanded memory*. Para ler dados da memória, os programas lêem as páginas através desta janela; para guardar informações na memória, os programas escrevem na janela, em blocos com um máximo de 64K (no máximo, 4 páginas de cada vez).

Podemos imaginar esta janela como o elevador para um grande edifício: temos acesso à porta do elevador mas não podemos entrar e subir nele! A capacidade do elevador é de 64Kb e o espaço disponível no edifício é de 32Mb. Como é agora evidente, no caso de se pretender usar mais de 64Kb, tem de se pedir ao elevador que faça as viagens necessárias para disponibilizar a quantidade de memória pretendida. Mas atenção: os blocos de memória vêm sempre no mesmo elevador, o que quer dizer que um novo bloco pedido vai sempre sobrepor-se ao que já lá estava! É da responsabilidade do programa que acede à expansão de memória usar correctamente os recursos disponíveis para cada finalidade específica.

Vamos então assentar ideias quanto a números:

- a expansão de memória permite aceder até um máximo de 32Mb;
- a unidade de memória da expansão são páginas de 16Kb, o que significa que existirão no subsistema 2048 páginas, designadas por páginas lógicas;

- a janela de acesso está baseada num endereço acima dos 640Kb, ou seja, um endereço superior a 000A:0000h;
- a janela de acesso tem 64Kb de largura, permitindo manipular até 4 páginas em cada chamada ao subsistema – estas são as páginas físicas;
- a área de endereçamento ocupada por esta janela não pode coincidir, nem num único *byte* que seja, com a área usada por qualquer outra *card* (video, rede, fax,modem, *scanner*, etc) já existente; no caso de tal suceder, a *card* de expansão tem de ser programada para outro endereço base, por acção sobre os seus*j umpers*.

VAMOS SABER MAIS SOBRE O EMM

O EMM vai servir de interlocutor entre os requisitos dos programas que pretendem aceder à memória e a placa instalada no computador. As funções do EMM são as de operário especializado, inicializando a placa de expansão ao começar uma sessão de trabalho e verificando a sua funcionalidade, «mapeando» as páginas lógicas em páginas físicas e vice-versa, garantindo que as páginas de memória são correctamente atribuídas ao programa seu proprietário e libertando a memória ocupada por um programa, sempre que este termina.

“O EMM vai servir de interlocutor entre os requisitos dos programas que pretendem aceder à memória e a placa instalada no computador.”

Do ponto de vista do programador, trabalhar com a expansão de memória vai ser como trabalhar com um ficheiro DOS: os programas pedem um *handle* ao EMM (um género de código de acesso, que é dado sob a forma de um número inteiro); uma vez obtido o código (o *handle*), os programas pedem blocos de memória onde escrevem dados e guardam-nos na placa. Como cada programa pode possuir um ou mais *handles* mas é garantido que o EMM nunca atribui *handles* repetidos, a expansão de memória pode ser usada em ambiente de multitarefa, onde vários programas em execução simultânea podem aceder à placa sem qualquer tipo de conflitos. Os serviços do EMM vão desde o simples fornecimento da versão e endereço base da janela até às mais complicadas tarefas da gestão de memória.

As capacidades do EMM estão perfeitamente definidas e o acesso é feito através de um *interrupt* (**int 67h**), de forma idêntica aos acessos ao **int 21h** do DOS: preenche-se um conjunto de registos com os valores que são bons e invoca-se o *interrupt;* o EMM faz o que lhe mandaram fazer e, no fim, devolve um resultado indicando se tudo correu bem ou se, pelo contrário, tudo correu mal.

Um exemplo simplificado de utilização da expansão de memória poderia ser ilustrado pelo seguinte diálogo (um pouco simplificado!) entre um programa e o EMM:

Programa –> «EMM, qual é a tua versão?»
EMM –> «É a versão 4.0.»
Programa –> «Ah! E então se eu quiser aceder aos teus blocos de memória, em que endereço é que tu colocas a janela?»
EMM –> «No endereço XXXX!»
Programa –> «Bom, sendo assim, dá cá um *handle* ...»
EMM –> «Está disponível o 5.»
Programa –> «Então, para o *handle* 5, dá-me 16K de memória.»
EMM –> «Aqui estão eles!»
Narrador-> Sabendo que o bloco pedido está no endereço XXXX, o programa pode guardar lá a informação que entender necessária. O tipo de informação, o formato como é guardada e tudo o resto é da inteira responsabilidade do programa. O EMM apenas se encarrega de guardar ou ir buscar o bloco à placa, em obediência às ordens do programa...
Programa –> «EMM, já fiz o que tinha a fazer com os tais 16K, podes guardá-los na placa.»
EMM –> «OK!»
...

Figura 3

VAMOS SABER MAIS SOBRE O NOSSO EMULADOR DE EXPANSÃO DE MEMÓRIA

O nosso famoso emulador chega-nos sob a forma de *shareware*, tendo sido escrito em Julho 1989 por um senhor que atende pelo nome de Douglas Boling. Este programa foi originalmente publicado na revista PC-MAGAZINE. O pacote EMS40.ZIP é composto pelos ficheiros EMS40.SYS (o *device driver* que se encarrega do trabalho, devidamente compilado), EMS40.ASM (a *source* do programa) e EMS40.DOC (as instruções de utilização). Conforme vem referido nestas instruções, o EMS40.SYS é um *device-driver* que emula, por *software*, um subsistema LIM/EMS 4.0, permitindo que a *extended memory* das máquinas 286 (e superiores) possa ser usada como *expanded memory*. Para a instalação do emulador, deverá ser copiado o EMS40.SYS para a *root* do *drive* de *boot* (consideremos que é o disco C:) e alterado o CONFIG.SYS, para passar a incluir a linha DEVICE=C:\EMS40.SYS 100. Neste caso, o EMS40 vai usar 100K de extensão de memória para instalar o subsistema; quando não é especificado qualquer valor, o EMS40 reserva por defeito os 384K.

Depois de alterado o CONFIG.SYS, é só fazer *re-boot* ao sistema. Se tudo correu bem, obteve-se no ecrã a mensagem ilustrada na figura 4:

EMS 4.0 Simulator, Ver 1.1 (C) 1989 Ziff Comunications
PC Magazine * Douglas Boling

Figura 4

Se tudo correu mal, foi (quase de certeza) porque a *extended memory* já estava ocupada com outros *drivers*, tais como *cache* de disco, disco virtual, etc. Como a memória não cresce, a resolução deste problema passa pelo «emagrecimento» dos *drivers* já instalados, para que seja possível arranjar espaço para o EMS40.SYS.

UTILIZAÇÃO DAS FUNÇÕES MAIS IMPORTANTES DO EMM

Como é bastante exaustiva a lista de TODAS as funções do EMM, dispusémo-nos neste número da **Spooler** a elaborar um programa de teste das funções consideradas mais importantes. Assim, seleccionámos 11 funções e elaborámos num pequeno programa utilitário (EMSSPO.EXE) que implementa um *menu* de opções, onde cada opção invoca uma das funções a testar. O código da função a executar é sempre carregado no registo **AH**. No caso da função se executar correctamente, o registo **AH** regressa com o valor **00H**; caso possua um valor diferente de zero, ocorreu um erro e o valor em **AH** é o código do erro. Alguns códigos de erro são comuns a todas as funções:

- AH = 80H – erro interno no *software* do EMM;
- 81H – erro de funcionamento no hardware do EMM;
- 84H – a função invocada não está definida. No caso das funções em Pascal implementadas no programa EMSSPO.PAS, não foi elaborada uma rotina de tratamento de erros: ou tudo corre bem ou é devolvida uma mensagem de ERRO genérica, sem especificar o problema.

Função EMS nº 2 (02H):

Descrição -> Determina o endereço da janela de acesso à expansão de memória (interes sa apenas o segmento, já que o *offset* a usar é 0000H).
Entrada -> AH=41h
Saída -> AH=00h (sucesso)
BX=Segmento da janela
Exemplo ->

```
mov ah,41h      ; invoca a função 41h
int 67h         ; por intermédio do interrupt 67h
or  ah,ah       ; testa se AH=00H
jnz ERRO        ; não é zero - ocorreu erro!
mov PG_SEG,bx   ; tudo bem!
...
ERRO:           ; em função do valor de ah, decide
...             ; qual a mensagem de erro a enviar para
                ; o utilizador
PG_SEG: dw 0
```

Nota -> Em Pascal, o mesmo exemplo pode ser implementado da seguinte forma:

```
function EMSgetFrameSeg(var p:pagPtr):boolean;
  var r:registers;
  begin
    r.ah:=$41;
    Intr($67,r); { Invoca o EMM }
    if r.ah=0 then
      begin
        (* em BX está o resultado *)
        (* Coloca em P a resposta *)
        EMSgetFrameSeg:=true
      end
    else
      EMSgetFrameSeg:=false;
  end;
```

Função EMS nº 3 (03H):

Descrição -> Determina o número de páginas lógicas presentes no sistema e o número de páginas ainda disponíveis. Estes valores dependem da quantidade de memória instalada no subsistema EMS.
Entrada -> AH=42h
Saída -> AH=00h (sucesso)
BX=Páginas disponíveis
DX=Páginas no sistema

Função EMS nº 4 (04H):

Descrição -> Indica ao EMM que um programa pretende usar a expansão de memória, obtém um handle para atribuir a esse programa e reserva um conjunto de páginas lógicas para esse *handle*. A primeira página lógica atribuída tem o índice zero.
Entrada -> AH=43h
BX=Número de páginas pretendido
Saída -> AH=00h (sucesso)
DX=Handle
Erros -> ...
85H - *Handles* esgotados
87H - Foram pedidas mais páginas que o total existente no sistema (não foram atribuídas páginas)
88H - Foram pedidas mais páginas que o total momentaneamente disponível no sistema (não fo ram atribuídas páginas)
89H - Foram pedidas 0 páginas

Função EMS nº 5 (05H):

Descrição -> Mapeia uma página lógica numa das 4 páginas físicas disponíveis, tornando-a acessível por endereçamento recto. Esta função pressupõe que já foi usada a função 04H; o parâmetro BX terá um valor entre zero e (número_de_páginas_pedido-1).
Entrada -> AH=44h
AL=Página física (0-3)
BX=Página lógica (0..num_pags-1)
DX=*Handle*
Saída -> AH=00h (sucesso)
Erros -> ...
83H - *Handle* inválido
8AH - A página lógica requisitada para mapeamento não está atribuída a este *handle*
8BH - A página física requisitada para mapeamento não estána gama 0..3

Função EMS nº 6 (06H):

Descrição -> Indica ao EMM que um programa pretende libertar um *handle* e as respectivas páginas associadas.
Entrada -> AH=45h
DX=*Handle*
Saída -> AH=00h (sucesso)
Erros -> ...
83H - *Handle* inválido
86H - Erro ao salvar/restaurar o contexto

Função EMS nº 7 (07H):

Descrição -> Determina a versão do EMM.
Entrada -> AH=46h
Saída -> AH=00h (sucesso)
AL=Versão do EMM, em BCD

Função EMS nº 8 (08H):

Descrição -> Salva os registos do EMM e o conteúdo das páginas físicas na EMS, associando-as a um *handle* específico.
Entrada -> AH=47h
DX=Handle
Saída -> AH=00h (sucesso)
Erros -> ...
83H - Handle inválido

Função EMS nº 9 (09H):

Descrição -> Recupera os registos do EMM e repõe o conteúdo das páginas físicas.

Figura 5 (continua na pág. seguinte)

```
                Esta função pressupõe que foi invocada
                préviamente a função 08H.
   Entrada  ->  AH=48h
                DX=Handle
     Saída  ->  AH=00h (sucesso)
     Erros  ->  ...
                83H - Handle inválido

Função EMS nº 12 (0CH):

  Descrição ->  Determina o número de handles activos.
    Entrada ->  AH=4Bh
      Saída ->  AH=00h (sucesso)
                BX=Número de handles activos
      Erros ->  ...
                83H - Handle inválido

Função EMS nº 13 (0DH):

  Descrição ->  Determina quantas páginas lógicas
                estão atribuídas a um determinado handle.
    Entrada ->  AH=4Ch
                DX=Handle
      Saída ->  AH=00h (sucesso)
                BX=Número de páginas
      Erros ->  ...
                83H - Handle inválido

Função EMS nº 14 (0EH):

  Descrição ->  Determina o número de handles activos
                e preenche um array com informação
                do número do handle respectivo número
                de páginas associado.
    Entrada ->  AH=4Dh
                ES:DI=Segmento:Offset do array
                a preencher
      Saída ->  AH=00h (sucesso)
                BX=Número de handles activos
```

Figura 5 (continuação da pág. anterior)

Outro aspecto do EMS-SPO.PAS que nos parece importante realçar é a forma como foi feita no programa a detecção da presença do EMM.

O PROGRAMA EMS-SPO.EXE: ALGUMAS QUESTÕES TÉCNICAS

Como já referimos, este programa vai permitir testar individualmente cada função do EMM. Para este fim implementámos em Pascal um conjunto de funções de acesso ao EMM. Nas chamadas ao EMM foi utilizada a função INTR($NUMERO,R) do Turbo Pascal, em que $NUMERO é o número do interrupt a invocar (no nosso caso, **$INT67**) e **R** é um *record*, do tipo pré-definido REGISTERS que existe para suportar o acesso aos registos do CPU através do Turbo Pascal – para usar cada registo individualmente acede-se aos campos **R.AL, R.AX, R.BX,** etc.

Para ilustrar a técnica utilizada vamos exemplificar para a função que determina o número de páginas do sistema: **EMSgetNumPages**.

Esta função é booleana, devolvendo FALSE ou TRUE conforme ocorra ou não erro na chamada ao EMM; os resultados da chamada ao EMM são devolvidos nos parâmetros passados à função por variável, onde **NP** será o número total de páginas e **ND** o número de páginas disponíveis.

```
function EMSgetNumPages(var np,nd:integer):boolean;
   var r:registers;
      begin
         r.ah:=$42;        { AH recebe o número da função a executar }
         Intr($67,r);      { Invoca o EMM }
          if r.ah=0 then   { Se AH=0, não houve erro }
            begin
               np:=r.dx; { DX tem o número total de páginas }
               nd:=r.bx; { BX tem o número de páginas disponíveis }
               EMSgetNumPages:=true
            end
         else
            EMSgetNumPages:=false;
      end;
```

Figura 6

Outro aspecto do EMS-SPO.PAS que nos parece importante realçar é a forma como foi feita no programa a detecção da presença do EMM. Para este fim implementámos a função **EMStest**, que devolve TRUE se o EMM estiver presente. Como é evidente, no caso de se concluir que não está instalado o EMM, não fará sentido tentar usar os serviços do *interrupt* **67h**.

Vejamos então qual a estratégia utulizada nesta detecção. Como já referimos, o EMM é instalado no DOS sob a forma de um *device driver*. Por definição, a especificação LIM/EMS impõe que este *device driver* tenha o nome "EMSXXXX0" – note-se que, apesar do ficheiro se chamar «EMS40.SYS», a designação pela qual ele vai ser conhecido no sistema é diferente ("EMSXXXX0"), sendo isto determinado pelo programador ao escrever o código do próprio *device driver* – os leitores podem confirmar isto mesmo se inspeccionarem o programa EMS40.ASM.

Portanto, o que esta função EMStest vai determinar é se está instalado o *device driver* de nome "EMSXXXX0". Para este fim, usámos a função **3Dh** do *interrupt* **21h** do DOS (*Open File*) para tentar abrir o ficheiro "EMSXXXX0" – como é sabido, um *device driver* comporta-se exactamente como um ficheiro, podendo ser aberto e fechado com as funções que o DOS fornece para a manipulação de ficheiros.

Se o ficheiro não for aberto, sabemos que não está instalado o EMM (assunto encerrado); pelo contrário, se for aberto, ficamos a saber que existe um ficheiro chamado "EMSXXXX0". Mas isto ainda não nos diz tudo: falta descobrir se é o *device driver* do EMM ou, por azar, um verdadeiro ficheiro DOS com este nome, gravado no *drive* de defeito. Para determinar a diferença, usa-se a função **44h** (*Device driver control*) que fornece a informação de que necessitamos – ao ser invocada com **AL=0** (*get device information*) esta função indica no **bit 7** do par de registos **DX** o valor "**0**" para ficheiro e "**1**" para *device-driver*. Finalmente, ao obter "**1**", ficamos a saber que está instalado o EMM!

Esta explicação é, sem dúvida, bastante técnica mas, em nosso entender, importante para os leitores que pretendam perceber como funciona o nosso program EMS-SPO.PAS.

Vamos atalhar esta dissertação que já vai longa. A ideia fundamental que presidiu à execução do EMS-SPO foi a de fornecer aos programadores interessados em trabalhar com a *expanded memory* uma ferramenta de teste das funções mais interessantes do EMM, bem como a *source* de um programa devidamente testado. No nosso *menu* de opções de "**0**" a "**D**", a opção "**C**-Escrever uma página na EMS" permite encher uma ou mais páginas de 16K com um carácter à nossa escolha e guardá-lo na EMS para que, posteriormente, se possam ler através da opção "**D**-Ler uma página da EMS".

Isto não tem, obviamente, qualquer interesse que não seja o de demonstrar que o sistema funciona – pode, no entanto, ser o ponto de partida para algumas aplicações interessantes... Gostaríamos de receber algum *feed-back* da vossa parte em relação a este assunto e, eventualmente, conhecer alguns programas que venham a escrever em que recorram a estas técnicas.

Paulo Bernardo

PROGRAMAR EM CLIPPER (PARTE I)

O crescente interesse dos utilizadores portugueses por bases de dados, levou a Spooler **a iniciar uma série de artigos sobre este tema, nomeadamente sobre os conhecidos dBase III Plus e Clipper.**
Começaremos pelo dBase III Plus, mostrando os seus diversos tipos de ficheiros, comandos e operadores. Seguidamente (no 3º artigo) passaremos para o Clipper, explorando as diversas potencialidades deste compilador.
Para simplificar a tarefa, vamos supor que o leitor é legitimo possuidor do dBase III Plus e do Clipper, encontrando-se ambos instalados no disco C: nas directorias DBASE e CLIPPER, respectivamente.

INTRODUÇÃO

A utilidade de um **Sistema Gestor de Bases de Dados** (SGBD) é muito grande, na medida em que pode controlar e organizar o armazenamento de qualquer tipo de informação de um modo rápido e fácil.

O dBase III Plus é o mais conhecido gestor de bases de dados para microcomputadores. Wayne Ratliff, um programador de *software* da Jut Propulsion Laboratory em Pasadena, Califórnia, impressionado pela capacidade e características de um gestor de base de dados de grande porte, resolveu desenvolver um sistema semelhante para o seu computador pessoal. Ratliff baptizou-o de «Vulcan» (em homenagem ao planeta de Mr. Spock, da série «Start Treck»).

O programa foi testado por George Tate, um distribuidor de *software* e, ficou tão impressionado com este, que assinou um contrato de direitos de autor com Ratliff. Para aumentar as vendas do Vulcan, Tate utilizou várias estratégias de *marketing*. O primeiro passo foi mudar o nome para «dBase II» (nunca existiu o dBase I, mas II significava uma versão mais actualizada). Tate criou uma empresa com Hal Lasblee, a **Ashton-Tate**, para comercializar o produto (não havia Ashton, mas Tate achou que os dois nomes ficavam bem juntos). Após alguns anos, concorrentes lançaram produtos com maiores vantagens. Como resposta, a Ashton-Tate desenvolveu o dBase III, que foi escrito em "C". Depois, devido às necessidades dos computadores pessoais desenvolverem trabalhos em rede, foi criado o dBase III Plus. Esta nova versão oferece muitas melhorias em relação às suas antecessoras.

PRIMEIRAS NOÇÕES

Campos e Registos

Vamos agora tentar uma aproximação ao tema, por analogia. Imaginemos que temos numa gaveta da nossa secretária um *dossier* onde estão guardadas informações sobre os nossos leitores. Essas informações são: nome, morada, código postal, telefone e data de nascimento. Imaginemos ainda que cada leitor tem uma ficha individual, onde constam estas informações. Básicamente, é este o princípio do dBase (e de qualquer sistema gestor de bases de dados).

Passando este exemplo para a informática, a «secretária» é o nosso computador pessoal, a «gaveta» uma directoria do disco duro e o «dossier» um ficheiro de dBase. Este ficheiro, com extensão DBF, contém as informações dos leitores. Assim, o nome, a morada, o código postal, o telefone e a data de nascimento designam-se por CAMPOS e ao conjunto dos campos que contêm as informações relativas a cada leitor, chamamos REGISTO (ver figura 1). Um ficheiro não será mais que um conjunto de registos.



Figura 1: Campos e Registos

Tipos de Ficheiros

O dBase grava informações no disco, em ficheiros. Cada ficheiro tem um nome e extensão. Existem treze formas especiais de ficheiros, as quais são distintas pela extensão:

.dbf – Ficheiros de dados (onde são guardados os campos)
.dbt – Ficheiros de memo (notas dos campos memo)
.ndx – Ficheiros de índice (utilizado para ordenação)
.prg – Ficheiros de programas (listagem de comandos)
.lbl – Ficheiros de etiquetas
.mem – Ficheiros de variáveis de memória
.fmt – Ficheiros de formatação
.txt – Ficheiros de texto
.frm – Ficheiros de relatório
.qry – Ficheiros de query
.vue – Ficheiros de view
.scr – Ficheiros de screen
.cat – Ficheiros de catalog

CONFIG.SYS E CONFIG.DB

Para poder abrir vários ficheiros as mesmo tempo, necessita de ter a seguinte linha no seu CONFIG.SYS (este ficheiro deve obrigatoriamente encontrar-se na raiz do seu disco): FILES=15 (ou superior). Este exemplo permite-lhe manter abertos 10 ficheiros ao mesmo tempo (o MS-DOS guarda para si 5 ficheiros). Para mais informações, veja o seu manual do Sistema Operativo ou consulte a rúbrica D.O.S.– Do Nascimento à Maioridade.

Quando se chama o dBase, este procura o ficheiro CONFIG.DB e caso o encontre, é lido e os seus comandos executados. Este ficheiro é composto por uma série de linhas e tem o seguinte formato:

(PALAVRA-CHAVE) = (parâmetro)

Por palavra-chave entendemos comandos SET ou outro comando permitido e parâmetro é o valor que lhe desejamos atribuir (ex.: ON ou OFF).

“Os comandos existentes na linguagem do dBase permitem várias formas de visualizar, ordenar e imprimir as informações.”

Criação de um CONFIG.DB:

```
C:\DBASE>copy con config.db

WP       =  WS (caso o seu processador de
            texto seja o WordStar)
TEDIT    =  WS
COLOR    =  bg/n,w/b
STATUS   =  on
BELL     =  off
COMMAND  =  do inicio

^Z (F6 e ENTER)
```

Comandos SET, definições de cores, teclas de função e o nome do seu processador de texto (necessário para escrever programas, devido às limitações do processador do dBase), poderão ser guardados no CONFIG.DB e executados sempre que chamar o dBase.

A última linha deste exemplo, chama um ficheiro de programa (INICIO.PRG) que pode conter qualquer tipo de instruções, como veremos no próximo número. Se esta linha fosse substituida por:

```
COMMAND=ASSIST
```

quando iniciássemos uma sessão de trabalho no dBase, estaríamos a optar por um dos modos de trabalho disponíveis no dBase III Plus: o modo assistido.

Comandos

Um comando consiste de um verbo (em inglês, é claro!) como DISPLAY e uma qualificação adicional que o torna mais especifíco. Exemplo:

```
DISPLAY                    && mostra o
                           conteúdo do
                           registo actual
DISPLAY NEXT 10            && lista os
                           próximos 10
                           registos
DISPLAY FOR nome="MARIA"   && lista todos os
                           registos cujos
                           nomes comecem
                           por MARIA
```

Os comandos existentes na linguagem do dBase permitem várias formas de visualizar, ordenar e imprimir as informações. Estes comandos podem ser utilizados conjuntamente, quer digitando-os na *prompt* do dBase, quer escrevendo-os num programa. Por exemplo, para criar e utilizar uma agenda, será necessário:

1º Criar a estrutura da base de dados;
2º Digitar a informação nos diversos campos (inserir registos);
3º Listar a informação.

A maior parte dos comandos podem ser executados através dos menus interactivos do dBase (Assist), mas como o nosso objectivo é iniciar os leitores na linguagem e programação do dBase, iremos pôr de parte o sistema de menus. No entanto, caso queira apenas executar um ou dois comandos, recomendamos a sua utilização.

Tipos de Comandos

As figuras 2,3 e 4 mostram os diferentes comandos que o dBase aceita. Estes comandos serão explicados à medida que forem utilizados em aplicações.

No entanto, o dBase dispõe de menus de ajuda que lhe permitirão conhecer, desde já, a função de cada comando. Para aceder ao nível de ajuda, prima a tecla F1, ou digite na *prompt* o comando HELP seguido do comando para o qual pretende uma explicação. Ex.: .HELP APPEND

1 – ?/??	17 – ERASE	33 – REPLACE
2 – APPEND	18 – EXPORT	34 – RENAME
3 – AVERAGE	19 – FIND	35 – REPORT
4 – BROWSE	20 – GO/GOTO	36 – SCREEN
5 – CHANGE	21 – IMPORT	37 – SEEK
6 – CLEAR	22 – INDEX	38 – SET
7 – CONTINUE	23 – LABEL	39 – SKIP
8 – COPY	24 – LIST	40 – SORT
9 – COUNT	25 – LOCATE	41 – STORE
10 – CREATE	26 – LOGOUT	42 – SUM
11 – DELETE	27 – MODIFY	43 – TOTAL
12 – DELETE FILE	28 – PACK	44 – TYPE
13 – DIR	29 – QUERY	45 – UNLOCK
14 – DISPLAY	30 – QUIT	46 – USE
15 – DO	31 – RECALL	
16 – EDIT	32 – RELEASE	

Figura 2: Comandos Básicos

1 – @	21 – ON
2 – ACCEPT	22 – PARAMETERS
3 – CANCEL	23 – PRIVATE
4 – CALL	24 – PROCEDURE
5 – CLOSE	25 – PUBLIC
6 – COPY FILE	26 – READ
7 – DISPLAY	27 – REINDEX
8 – DO CASE	28 – RESTORE
9 – DO WHILE	29 – RESUME
10 – EJECT	30 – RETRY
11 – EXIT	31 – RETURN
12 – IF	32 – RUN/!
13 – INPUT	33 – SAVE
14 – INSERT	34 – SELECT
15 – JOIN	35 – SUSPEND
16 – LIST	36 – TEXT
17 – LOOP	37 – UPDATE
18 – MACRO/&	38 – VIEW
19 – MODIFY	39 – WAIT
20 – NOTE/*/&&	40 – ZAP

Figura 3: Comandos Avançados

1	SET ALTERNATE	13	SET HISTORY
2	SET CATALOG	14	SET INDEX
3	SET COLOR	15	SET MARGIN
4	SET DATE	16	SET MEMOWIDTH
5	SET DECIMALS	17	SET MESSAGE
6	SET DEFAUT	18	SET ORDER
7	SET DELIMITERS	19	SET PATH
8	SET DEVICE	20	SET PRINT
9	SET FIELDS	21	SET PROCEDURE
10	SET FILTER	22	SET RELATION
11	SET FUNCTION	23	SET TYPEAHEAD
12	SET FORMAT	24	SET VIEW

Fiura 4: Comandos SET TO

Tipos de Operadores

Os operadores, que são representados por símbolos, funcionam como valores relacionados para criar um único valor. Existem quatro tipos de operadores: **matemático, relacional, lógico** e de **string** (carácter).

OPERADORES MATEMÁTICOS

Estes operadores são utilizados para obter resultados numéricos. As operações disponíveis são: adição (+), subtração (-), multiplicação (*), divisão (/) e exponenciação (** ou ^).

Ex.: ((40 + 54) / 10) * (3 *(50 – 65))3**

OPERADORES RELACIONAIS

Estes operadores são utilizados para comparar **strings** de caracteres e números, e devolvem um valor que pode ser verdadeiro (.**T**.) ou falso (.**F**.). Os operadores são:

< menor que
> maior que
= igual a
<> ou # diferente de
<= menor ou igual a
>= maior ou igual a

Ex.: 23 < 46 && verdadeiro (.T.)
23 > 46 && falso (.F.)

OPERADORES LÓGICOS

Estes operadores são: .AND. , .OR. e .NOT. .

Ex.: 123 = 123 .AND. "GUIDA" = "ANA"
&& falso (.F.)
123 = 123 .OR. "MARIA" = "MANUEL"
&& verdadeiro (.T.)
123 = 123 .AND. .NOT. "SILVIA" = "MARIO"
&& verdadeiro (.T.)

OPERADORES DE STRINGS

O único operador de **strings** do dBase é o sinal de adição (+), e permite concatenar duas **strings** de caracteres.

Ex.: "Spooler" + " Magazine" && "Spooler Magazine"

Os operadores, que são representados por símbolos, funcionam como valores relacionados para criar um único valor.

NO PRÓXIMO NÚMERO...

Manutenção de uma Base de Dados: criar, inserir, alterar e remover informação de uma base de dados.

Listagem e Organização: ordenar e listar a base de dados.

Vitor Manuel Monteiro

SECTOR DE INFORMÁTICA

ESTUDO 1 FORNECEDORES DE HARDWARE EM PORTUGAL

ESTUDO 2 SERVIÇOS INFORMÁTICOS E SOFTWARE

Conteúdo dos estudos:
a) Caracterização do sector
b) Análise por empresa
c) Ficha por empresa

ESTUDO 3 PARQUE INFORMÁTICO PORTUGUÊS

- TOMO I – Estado e Entidades Públicas
- TOMO II – Bancos, Seguros e Financeiras
- TOMO III – Empresas Industriais, Comerciais e de Serviços

Conteúdo do estudo:
a) Classificação Quantitativa das Empresas do Sector
b) Ficha por Entidade com indicação dos Equipamentos instalados
c) Análise Global por Marcas

Para mais informações contactar:

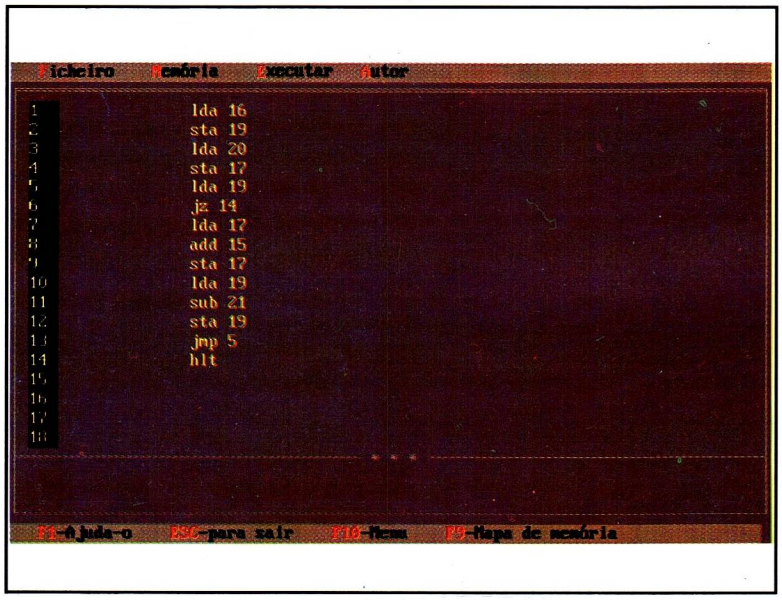
ifquatro

Processamento de Informações, LDA.

Empreendimento Amoreiras, Torre 2, 3º Andar, Sala 9 – 1000 LISBOA – Telef.: 686730/687642 – Fax: 657726

COLABORAÇÃO DO LEITOR

UM MICROPROCESSADOR POR SOFTWARE



Eis-nos perante uma curiosa abordagem aos «mistérios» dos processadores que equipam as nossas máquinas. O autor criou um programa que simula alguns dos processamentos mais simples, mas não menos complexos. Vejamos o que o Jorge Anastácio tem para nos oferecer:

> **Qualquer computador, sem ser previamente programado, não serve para nada.**

Desde os primórdios da electrónica, tem sido preocupação dos projectistas a criação de dispositivos universais de processamento (circuitos electrónicos), que devidamente programados sejam capazes de executar vários tipos de algoritmos. Pretendendo-se que estes dispositivos (sistemas digitais) deixem de ter qualquer ligação com os algoritmos por eles executados, por mais complexos que sejam. E, numa perspectiva de redução de espaço, surgiram os microprocessadores e com eles, os microcomputadores.

Em qualquer processador, toda a informação necessária a um cálculo, será introduzida antecipadamente em memória. Após toda a informação introduzida, todo o processamento do cálculo, até à obtenção do resultado pretendido será o mais possivel independente da intervenção humana.

Qualquer computador, sem ser previamente programado, não serve para nada. Existem portanto, uma grande quantidade de programas que permitem fazer do computador uma máquina excepcional capaz de superar o homem na execução de determinadas tarefas. Assim, quando liga o seu PC, após «carregar» o sistema operativo, fica aquele pronto a receber instruções.

Mas, o que é o sistema operativo? O sistema operativo é um programa geralmente escrito em código máquina (*assembly*) que permite ao computador interpretar um certo numero de instruções, e fazer a gestão da memória RAM, assim como o controlo do restante *hardware*.Contudo, este artigo não pretende ensinar-lhe a construir um sistema operativo. Pretende sim, dar-lhe uma ideia do que é o código máquina a fim de que possa posteriormente desenvolver os seus conhecimentos neste campo, possuindo desde já uma base de formação. Além disto, e como complemento a este artigo,encontra também um programa que eu designei por MICRO.EXE, programa este que executa os algoritmos por si construidos e implementados com as instruções indicadas. Pretendo, assim, que verifique na prática como se processa a execução das instruções em código máquina.

Este programa, constitui um simulador de um microprocessador muito básico que não existe. No entanto, o modo de funcionamento e a filosofia inerentes ao mesmo, são precisamente idênticas aos processadores reais. Fica desde já advertido de que não ficará a conhecer um prosessador do tipo 8086. Contudo, ficará familiarizado com conceitos muito básicos necessários ao entendimento duma «filosofia» mais complexa.

Consideremos aqui, que um conjunto mínimo (funcional) de um microprocessador necessita de ter:

– Memória onde possa armazenar dados e variáveis durante a execução dum programa, programa esse que residirá também nessa mesma memória. Pelo facto de ter que armazenar variáveis em tempo de excução, a memória será do tipo RAM (memória de leitura e escrita).

– O microprocessador que trata (processa) os dados contidos na memória e controla todas as operações. Dentro do microprocessador, existe a ALU (unidade aritmética e lógica) que, como o seu próprio nome indica, é um dispositivo que se destina a efectuar operações lógicas (somas, sub-trações, etc).

– DMA, acesso directo à memória. Dispositivo que permite inserir (aceder) directamente à memória para gravar os dados necessários assim como o programa a ser executado pelo processador.

No caso presente, este dispositivo será o próprio editor (ficheiro MICRO.EXE)

Para simplificar, definimos que o nosso processador apenas teria que executar 8 instruções. Acho que estas operações são suficientes para alcançar o objectivo pretendido, uma vez que permitem realizar as operações essenciais à manipulação da memória, assim como diversos algoritmos.

As abreviaturas das instruções utilizadas são as descritas a seguir:

```
1   LDA <ENDER>
2   STA <ENDER>
3   ADD <ENDER>      <ENDER> Significa
                     endereço de memória.
4   SUB <ENDER>
5   JP  <ENDER>
6   JZ  <ENDER>
7   JMP <ENDER>
8   HLT
```

Pretendemos codificar estas instruções, atribuir um número a cada uma numa base binária, (apenas dispõe dos números 0 ou 1 por cada posição), isto porque é nesta base que todos os computadores funcionam. Para identificar e codificar 8 instrucções, apenas necessitamos de 3 bits (o equivalente a três casas decimais) note que 2^3 = 8. O código binário que irá corresponder às instrucções vai de 0 a 7.

O peso dos números aumenta numa exponencial de base 2 (binária), ou 10 (se considerar uma base decimal). Assim, numa base binária terá os seguintes pesos:

```
... 16   8   4   2   1
     |   |   |   |   |
    2^4 2^3 2^2 2^1 2^0
```

Por exemplo: **o número 1111 (bin.) = 8+4+2+1 = 15 (dec.)**
o número 1011 (bin.) = 8+(0*4)+2+1 = 11 (dec.)

Assim, e duma maneira totalmente arbitrária, definimos aqui que cada instrução terá o código que a seguir se indica:

```
LDA  —>  instrucção 0   (000) bin.
STA  —>  instrucção 1   (001) bin.
ADD  —>  instrucção 2   (010) bin.
SUB  —>  instrucção 3   (011) bin.
JP   —>  instrucção 4   (100) bin.
JZ   —>  instrucção 5   (101) bin.
JMP  —>  instrucção 6   (110) bin.
HLT  —>  instrucção 7   (111) bin.
```

Saberá assim que a instrução 001 será "STA".

A partir de agora, sempre que encontrar o termo «máquina», esse refere-se ao microprocessador.

A nossa máquina apenas possui um registo interno de uso geral que o utilizador pode aceder: o **acumulador**. É sobre este registo que todas as operações são realizadas. Um **registo** é um circuito capaz de reter (guardar) informação durante todo o tempo em que tiver alimentação (energia eléctrica). O acumulador é um registo de 8 bits (possui 8 posições) que tem associado a si uma ALU (unidade aritmética e lógica) para efectuar as operações. No presente caso, a ALU apenas tem que fazer somas e subtracções. A seguir, refere-se o que é que cada instrucção faz:

LDA <ENDER> **Acc <— (n)**
Load acumulador. Transfere (copia) o conteúdo da posição de memória indicada em <ENDER> para o acumulador.

STA <ENDER> **(n) <— Acc**
Store acumulador.Ttransfere (copia) o conteúdo do acumulador ara a posição de memória indicada em <ENDER>.

ADD <ENDER> **Acc <— Acc + (n)**
Adiciona o conteúdo do acumulador com o conteúdo da posição de memória indicada em <ENDER>. O resultado fica no acumulador.

SUB <ENDER> **Acc <— Acc - (n)**
Subtrai o conteúdo do acumulador com o conteúdo da posição de memória indicada em <ENDER>. O resultado fica no acumulador.

JP <ENDER> ***Jump if positive***
Saltar (passar) para a posição de memória indicada em <ENDER> apenas se o conteúdo do acumulador for >= 0.

JZ <ENDER> ***Jump if zero***
Saltar (passar) para a posição de memória indicada em <ENDER> apenas se o conteúdo do acumulador for = 0.

JMP <ENDER> ***Jump***
Saltar (passar) para a posição de memória indicada em <ENDER> onde fica situada a próxima instrucção, incondicionalmente, isto é, mesmo se não se verificarem outras situações.

HLT **Halt**
Parar a actividade do processador.

Associado a qualquer processador, temos a RAM (memória do computador).

É na RAM que se guardam as instrucções de todo e qualquer programa, assim como os dados referentes à execução dos programas e outros dados necessários ao funcionamento do processador. No presente caso, a nossa RAM será um bloco de memória que nada tem a ver com a RAM real do seu computador.

No presente processador, a RAM apenas possui 32 posições de 8 bit cada. Assim, para endereçar (localizar) qualquer destas posições, apenas necessitamos de 5 bits (2^5 = 32). Pretendemos dizer com isto que com 5 bits conseguimos obter 32 configurações diferentes com os numeros 0 e 1.

Verificámos anteriormente que o código das instrucções necessitava de 3 bits. Podemos, então, juntar a instrucção com o seu endereço associado numa única posição de memória. Admitir que os três primeiros bits são relativos às instruções, os cinco restantes referem-se aos endereços:

3 bits da instrucção
+ 5 bits para endereço
8 bits de posição de memória

O processador, vai, sempre que necessita, buscar as instruções à RAM. Existem para isso dois registos internos ao microprocessador associados à RAM que permitem as operações seguintes:

Existe um *program counter* (contador de programa), que não é mais do que um simples contador de 5 bits (contagem até 32). Este contador aumenta uma posição sempre que o processador for buscar uma instrução.

É por isso designado de contador de programa, uma vez que conta as instruções que se forem executando.

"É na RAM que se guardam as instrucções de todo e qualquer programa, assim como os dados referentes à execução dos programas e outros dados necessários ao funcionamento do processador."

Além disto, e como as instruções são colocadas na RAM por ordem de execução, o *program counter* (PC) fornecerá o endereço da RAM em que se encontra sempre a próxima instrução. Quando se liga o processador, é feito um *reset*, isto é, o contador de programa é colocado a zero, ficando a indicar a primeira posição de memória. No entanto, os endereços fornecidos pelo PC não acedem directamente à RAM, passam antes disto pelo MAR. O MAR (*memory address register*) é um registo de 5 bits que fornece directamente os endereços à RAM. Poderiamos ser levados a pensar que o MAR seria dispensável, uma vez que para endereçar a RAM seria suficiente o PC. Verá mais tarde, que o MAR é absolutamente necessário.

Temos assim a RAM endereçada (pelo PC), isto é, temos no barramento de dados o conteúdo da posição de memória que o PC está a enderecar (localizar). Um outro registo é o MBR (*memory buffer register*). É este um registo de 8 bits por onde passam todos os dados que entram ou saem de e para a memória. Será lógico entender o facto deste registo ter 8 bits, isto deve-se ao facto de cada posição da RAM também ter o mesmo número de bits. Este é um registo auxiliar onde ficam guardados os dados antes de irem para o acumulador ou para o IR.

Um último registo existente, é o IR (*instruction register*), registo de instruções. Quando o PC endereça a 1ª instrução (1ª posição da RAM), o conteúdo dessa posição passa para o MBR, e em seguida para o IR. É no IR que se faz o reconhecimento da instrução. Para saber o que é, por exemplo, um «LDA 20» e não um «STA 20». Este é um registo de 8 bits que possui um dispositivo que, consoante a instrução reconhecida, efectua diversas operações (dependendo da instrução).

Temos portanto no IR, um dado que contém o código de instrução nos 3 bits mais altos (maior peso), e temos nos 5 bits mais baixos o endereço associado à instrução. Este endereço é colocado no MAR. O conteúdo do PC mantém-se no entanto inalterável, a apontar a proxima instrução. É esta a outra função do MAR atrás referida: endereçar a RAM consoante o endereço indicado na instrução independentemente do valor do PC.

Considere o seguinte exemplo:

Admita que ligou o computador, e que o PC dirigindo-se à 1ª posição de memória, fez com que passasse o seu conteúdo para o MBR, e do MBR, para o IR. Admita que o IR identificou a seguinte instrução: «LDA <ender>». Então, o processador sabe que tem que passar o conteúdo da posição de memória especificada em <ender> para o acumulador. Necessita então de saber qual a posição onde se encontra o valor pretendido. Essa informação, encontra-se já no MAR (passou, dos 5 bits referentes ao endereco, do IR para o MAR). Suponha agora que a instrução era «LDA 20», então, no MAR encontra-se o valor 20. Para continuar (executar) a instrução, o processador dirige-se então à memória (posição 20) para buscar o valor. E sabe que agora esse valor já não vai passar do MBR para o IR, mas sim do MBR para o acumulador. Isto após ter passado da memória para o MBR.

Após isto, é incrementado o PC, e colocado o seu valor no MAR a fim de buscar a próxima instrução. E assim sucessivamente até encontrar a instrução HLT, altura em que pára a execução do programa.

Numa instrução de *jump* (ex: JP <ENDER>), depois de buscar a instrução, o que se faz será apenas colocar no PC, o valor especificado em <ENDER>.

Constata-se que existem duas fases inerentes a cada instrução: a Fase *fetch* (preparação) e a fase *execute* (execução). A fase *fetch* é comum a todas as instruções:

- passar o valor do PC para o MAR.
- passar o conteúdo da memória indicada pelo MAR para o MBR, ao mesmo tempo que incrementa o PC.
- passar o conteúdo do MBR para o IR.
- passar os 5 bits low do IR para o MAR.

Esta fase destina-se a reconhecer a instrução e preparar o processador para as acções seguintes dependendo da instrução.

Temos de seguida a fase *execute* que depende do tipo de instrução identificada:

- **LDA**
  passar o conteúdo da memória para o MBR.
  passar o conteúdo do MBR para o acumulador.

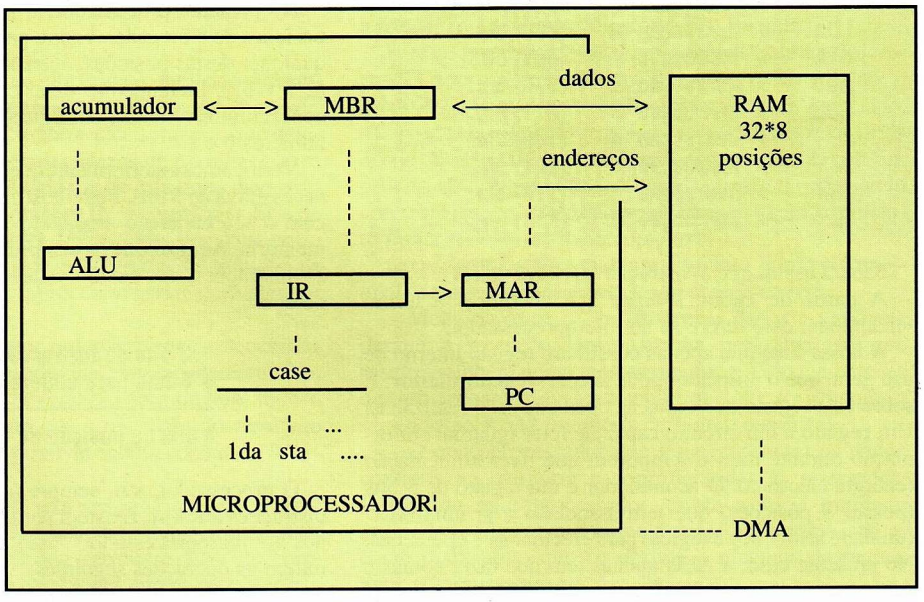
- **STA**
  passar o conteúdo do acumulador para o MBR.
  passar o conteúdo do MBR para a memória .

- **etc..**

Nesta fase, após preparação das acções necessárias à execução da instrução, esta será então executada.

O IR, possui um dispositivo que implementa a instrução «CASE» (um IF múltiplo) que executa uma série de acções se for uma instrução, e executa uma série diferente de acções se a instrução for outra.

A figura que se segue pretende dar uma ideia da teoria atrás referida:



Os rectângulos constituem registos.

Vamos de seguida, realizar um programa que multiplique o conteúdo das memórias de endereço 15 e 16 e colocar o resultado na memória de endereço 17.

Existem duas fases inerentes a cada instrução: a Fase *fetch* (preparação) e a fase *execute* (execução).

(15) * (16) = (17)

O primeiro passo a efectuar será escrever o algoritmo de resolução do problema em Pseudo-Pascal. De seguida, codificar o algoritmo em linguagem *assembly* com as instruções disponiveis. Por fim, testar o programa.

A nossa máquina, não multiplica números. Vimos que as instruções que dispomos que executam operações aritméticas, são apenas soma e subtração. O algoritmo utilizado é o seguinte:

R = M * m

em que **R** é o resultado da multiplicação que se pretende e **M** e **m** são os operandos.

então: **R = M+M+.....+M (m vezes)**
é uma soma de termos M repetida m vezes.

Vem que:

```
c: =  m;
R: =  0;
enquanto c <> 0  faça     //será este o algoritmo
                          a implementar
inicio
        R : =  R + M;
        c : =    c - 1;
 fim;
 (17) : = R;
```

Vamos pôr o contador (c) numa posição de memória, e subtrair-lhe sempre 1 unidade. O contador ficará na posição 19. Começamos por ir buscar o valor do 2º operando (m) que será o contador do número de vezes que se soma o outro operando com ele próprio.

Vem assim: LDA 16 (passa **m** (posição 16) para o acumulador). De seguida colocamo-lo na posição de memória que reservámos ao contador —> posição 19.

Vem assim: STA 19 (passa conteúdo do acumulador para posição 19)

Para iniciar o resultado a 0, teremos que colocar o valor 0 numa posição pré-definida e ir buscá-lo para a posição do resultado. Suponha que tem 0 na posição 20:

LDA 20	fomos buscar o conteúdo (0) da posição 20.
STA 17	colocámolo na posição 17 (resultado).

Aqui, começa o ciclo **enquanto:** e teremos que verificar se o contador chegou a 0:

LDA 19	Fomos buscar o valor do contador
JZ 14	**se for = 0** salta para a posição 14 onde está a instrução HLT (acabar o programa)

se não for 0 não faz nada e continua o programa.

R := R + M

LDA 17	Vai buscar o resultado
ADD 15	soma-o com o 1º operando
TA 17	coloca a soma na posição do resultado

c := c - 1

LDA 19	buscar o valor do contador
SUB 21	subtrair-lhe 1 unidade (supor que na posição 21, tinha colocado o valor 01.
STA 19	colocar o valor da subtração na posição 19 (sítio do contador)
JMP 5	saltar para o inicio do enquanto (repete o ciclo, donde só sairá quando o contador chegar a 0)

A maneira como deve introduzir o programa no editor é a seguinte:

```
LDA   16
STA   19
LDA   20
STA   17
LDA   19
JZ    14
LDA   17
ADD   15
STA   17
LDA   19
SUB   21
STA   19
JMP   5
HLT
```

Pode também, se quiser, executar a operação de ler dados dum ficheiro chamando o ficheiro «a:\exemplo.jma», onde se encontra já este programa.

Posteriormente, dirija-se ao menu e entre na janela memória, escolhendo a opção 1 (introduzir dados na memória). Assim, deverá introduzir os dados referentes a este programa, da seguinte maneira:

posição:	dados:
15	(número entre 0 e 255) 1º operando
16	(número entre 0 e 255) 2º operando
17	(resultado)
18	0
19	0 sitio do contador
20	0
21	1

Após a introdução dos dados escolha a opção 3 (situação anterior). Nesta altura, o seu programa encontra-se pronto para ser executado. Volte ao menu e escolha a opção executar. Premir ENTER. O programa será executado até encontrar a instrução HLT. Quando terminar a execução do programa, dirija-se de novo ao menu, entrando na opção Memória, e escolha a opção 2 (visualizar dados da memória) escolha a posição 17. Nessa posição terá o resultado da execução do programa. Assim, se tiver introduzido 7 na posição 15 e 3 na posição 16, o resultado obtido será de 21. Se obtiver resultados diferentes dos esperados, verifique se o produto dos operandos que colocou não excede 255 (8bits).

Ao construir os seus programas, não se esqueça de que só dispõe de 32 posições (de memória) para introdução das instruções e dos dados. Assim, também este editor só aceita 32 instruções (dados). Todos os restantes dados que quiser introduzir, serão ignorados pelo programa.

“A nossa máquina, não multiplica números. Vimos que as instruções que dispomos que executam operações aritméticas, são apenas soma e subtração.”



Jorge Anastácio

A DINASTIA INTEL

(PARTE I)

Apesar da polémica que surgiu há alguns meses, a invenção do microprocessador é ainda atribuída aos engenheiros da INTEL, pelo fabrico do 4004 em 1971, o primeiro a ser comercializado – um *chip* que hoje já nos faz sorrir pela sua simplicidade. Era o princípio da dinastia.
Depois de a IBM ter utilizado um processador 8088 a 4,77MHz no primeiro PC «azul», os engenheiros da INTEL não descansaram, antes muito pelo contrário, pelo que temos hoje em dia à nossa disposição, nos múltiplos compatíveis PC, uma panóplia de processadores compatíveis com o 8088 correndo a variadas velocidades de relógio.
Outras companhias que, inicialmente, receberam licenças da INTEL para fabricar processadores compatíveis com os seus, foram mais longe e introduziram no mercado versões mais rápidas, consumindo menos energia, ou mesmo com características genuinamente inovadoras.
Esta variedade que faz as delícias dos aficcionados da Informática, pela liberdade de escolha que lhes proporciona, confunde certamente o «comum mortal» que, ao comprar um PC, é bombardeado pelo revendedor com toneladas de termos obscuros, que o levarão a investir o seu dinheiro numa máquina que mais tarde ou mais cedo se revelará inadequada.
Grande parte das pessoas não sabe ainda quais são as verdadeiras capacidades dos seus PCs, nem as diferenças efectivas entre os vários modelos INTEL, apesar destes processadores darem vida à (esmagadora) maioria das máquinas de secretária actualmente em utilização.

16 BITS, 20 MHZ?

Antes de examinar a linha INTEL 80x86, convém saber alguns conceitos básicos sobre o funcionamento de um processador, e alguma da terminologia associada, para percebermos a evolução desta tão popular arquitectura.

Um processador reconhece um conjunto bem definido de operações que vai recolhendo da memória e executando. O tempo que cada instrução demora a ser descodificada e posta em prática é medido em ciclos do relógio do sistema. Este é um oscilador que produz um sinal a uma frequência bem definida e constante. Esta frequência mede-se em mega Hertz, ou **milhões de ciclos por segundo**. É óbvio, portanto, que quanto maior for a frequência do relógio, menor duração têm os ciclos e mais rapidamente é executado o código.

Esta é uma boa base de comparação de processadores, mas atenção, só se eles forem idênticos! Um procesador i486 é aproximadamente 3 vezes mais rápido que um i386 correndo à mesma velocidade de relógio! Como explicar tal? É muito simples: cada uma das instruções do 486 é executada em menos ciclos que a instrução correspondente num 386, devido a ser uma evolução deste. (É claro que a diferença não se resume somente a esta, pois outras evoluções contribuem para o efeito final.)

Um dos componentes essenciais deste tipo de processadores são os registos que, de uma forma simplista, se podem descrever como as variáveis de uma linguagem como o BASIC. São usados para conter dados temporários do programa em execução ou informações relacionadas mais intimamente com o processador. Existe geralmente um registo que é tratado não como um todo mas *bit* a *bit*.. São as *flags* que, ou controlam alguma função específica do processador, ou assinalam a ocorrência de algum evento especial.

Outra das palavras chave é o tamanho da «palavra» do processador. É aqui que entram os 8 *bits*, 16 *bits* ou 32 *bits* que conhecemos dos folhetos publicitários, os quais representam o número de dígitos binários (0 ou 1) que possuem o maior número que é possivel tratar pelo processador sem recorrer a programação. Este número é dado pela formula $2n-1$, onde n é o numero de *bits*. Num processador de 16 *bits* este é 65.535, enquanto que num de 32 é a respeitável quantia de 4.294.967.295! Logo, num processador de 16 *bits*, qualquer operação que possa produzir um número maior que 65.535 tem que ser tratada de um modo especial, pelo que será inevitavelmente mais lenta do que se for realizada num processador de 32 *bits*. Mas a vantagem mais significativa de uma maior «largura» da palavra encontra–se na movimentação de dados de uma área de memória para outra. Quanto maior, menor número de operações elementares serão necessárias e, consequentemante, mais rápida será a transferência.

Os registos de um processador são (com raras excepções) do tamanho da palavra. Pode–se, na maior parte das vezes, tratar dois deles como um todo ou um dividir-se em vários, para lidar com tipos de dados de tamanhos diferentes.

Outro componente essencial é o «barramento» ou *bus*, dividido em barramento de dados e endereços. O barramento é uma «auto–estrada» de várias vias por onde a informação circula. A rapidez do «trânsito» está directamente relacionada com a largura (número de faixas) da via e com a velocidade a que a informação a percorre, esta ditada pelo relógio do sistema. Pelo barramento de dados circulam as instruções e dados para o processador e regressam os resultados para a memória. Através do *bus* de endereços o processador controla a posição de memória a ser acedida.

É de frisar que as larguras do barramento de dados e de endereços estão relacionadas com o tamanho da palavra, embora nem sempre sejam equivalentes, por considerações quer técnicas, quer comerciais, como se verá mais à frente.

Como o tempo de acesso das memórias DRAM (*Dynamic Random Acess Memory*) não conseguiu acompanhar o aumento espectacular da velocidade dos processadores, estas começaram a ser o «lastro» que impedia os computadores de voar mais alto. As SRAM (*Static Random Acess Memory*) são rápidas quanto baste, mas são muito mais caras e não atingem níveis de integração suficientemente grandes para fornecer os Mb a que já nos habituámos. A solução de compromisso é a interposição de um *cache* entre o processador e a memória central.

Um *cache* é uma pequena quantidade de memória rápida (SRAM) que contém os dados mais frequentemente usados pelo processador. Elaborados algoritmos determinam quais as zonas de memória central que merecem o tratamento priveligiado, de modo que o processador (em mais de 90% das vezes nas máquinas actuais), encontre os dados que precisa directamente no *cache* e, como tal, não ter que esperar pela vagarosa DRAM.

Como comentário final, os processadores da linha INTEL 80x86 são todos CISC (de *Complex Instruction Set Computer*), ao contrário de RISC (de *Reduced Instruction Set Computer*), que é um tipo de arquitectura mais utilizada nas estações de trabalho de alto desempenho. CISC significa que o processador possui um conjunto mais vasto de instruções, que são mais especializadas e complexas. Um processador RISC tem, pelo contrário, apenas o mínimo indispensável. Esta simplicidade permite–lhe que todas as instruções sejam executadas num único ciclo de relógio, pelo que atingem desempenhos consideráveis.

OS PRIMÓRDIOS: 8086,8088

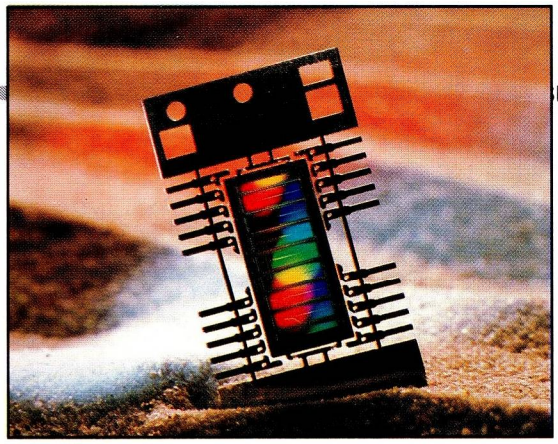
Em 1978, a INTEL lançou no mercado a sua terceira geração de microprocessadores, na forma do 8086. Como novidade apresentava o facto de ser um processador de 16 bits, ao contrário do seu predecessor, o 8088 de apenas 8 bits. Com a capacidade de endereçar 1Mb de memória, graças ao barramento de endereços de 20 bits, era um grande passo em frente em relação aos 64Kb anteriormente disponíveis.

Porém, para aceder a esse 1Mb de memória foi preciso recorrer a alguma «magia negra» sob a forma que é conhecida como «memória segmentada». Para aceder a qualquer endereço, o 8086 tem, antes de mais, que carregar no registo de segmento a zona da memória que quer aceder e depois tem que carregar num registo de deslocamento a posição dentro dessa zona de 64Kb da célula a ser realmente acedida. Os problemas começam quando se quer utilizar uma estrutura de dados de tamanho superior a 64Kb, pois as operações necessárias à manipulação simultânea dos dois registos desaceleram significativamente o processamento.

Além disto, as raizes deste processador dos anos 70 pesavam sob a forma de uma total ausência de facilidades para implementar um sistema operativo multitarefa, das quais a mais importante é a ausência de protecção da memória. Isto significa que qualquer programa que corra neste processador tem autorização para alterar o que lhe apetecer e aceder directamnte aos periféricos que bem entender. Se o programa é, efectivamente, o único, tudo bem, caso contrário está aberto o caminho aos «estoiros» (*crash*) que os utilizadores do DOS tão bem conhecem.

Uma das características do 8086 era o seu flexível tratamento das interrupções, que podiam ser accionadas por *software*. As interrupções são o reflexo do facto de o processador,por vezes, ter de abandonar a tarefa que estava a desempenhar para tratar algum acontecimento urgente. O primeiro Kb do espaço de endereço do 8086 está reservado para a tabela de interrupções, que é um conjunto de 256 endereços que podem corresponder a outras tantas rotinas invocadas quando a respectiva interrupção é activada. Algumas são reservadas pelo processador para assinalar excepções, tal como a divisão por 0. No PC, outras são activadas pelo *hardware*. Por exemplo, ao movermos o rato, despoletamos uma interrupção que invoca uma rotina encarregada de mover o cursor, de acordo com os dados recebidos.

Podem, como já se disse, ser activadas por *software*. É esse o caso dos serviços do DOS, acessíveis através da famosa interrupção 21h (33 decimal). O conhecimento do mecanismo das interrupções do 8086 é indispensável para quem se quiser dedicar à programação deste processador em Assembly, e sob DOS. Este sistema operativo tem *interface* de programação inteiramente constituída por interrupções, que desempenham também um importante papel nas comunicações interiores ao DOS.



Chip de 4 Mbits da Siemens.

É o seguinte o conjunto de registos do 8086 :

AX,BX,CX,DX	– De uso geral (Acumulador, Base, Contador e Dados).
CS,DS,SS,ES	– De segmento (Código, Dados, *Stack* e Extra).
SI,DI	– Índices de memória ou *offsets* (Origem e Destino).
BP	– Apontador para o *stack* (Base).
IP	– Instrução seguinte.
SP	– Topo do *stack*.
FLAGS	– Um conjunto de *flags* de estado e controlo.

Os registos de uso geral são, como todos os outros, de 16 bits; no entanto podem usar–se como se cada um deles fosse dois de 8 bits. Assim, AX está dividido em AH e AL, respectivamente os bytes mais e menos significativos. O mesmo se passa com os restantes: BX,CX e DX. As facilidades de *debugging* incorporadas no 8086 são um modo de execução «passo a passo» (*trace*) activada por uma das *flags* e uma instrução de paragem (*break–point*). O 8086 é, como já se disse, um processador de 16 bits que, como seria de esperar, tem um barramento de dados de 16 bits. No entanto, no princípio dos anos 80, 16 bits eram um luxo, dado que os componentes adicionais necessários para fazer um computador eram substancialmente mais caros em versão 16 bits do que em 8 bits; assim, a INTEL construiu uma versão do 8086 que se apresentava ao exterior como sendo de 8 bits: era o 8088. Como a largura do seu barramento era inferior à do 8086, este apresentava um desempenho ligeiramente inferior, dado que todas as transferências de e para a memória demoravam o dobro do tempo. Como os acessos à memória não ocupam 100% do tempo, esta perda de velocidade era apenas de cerca de 30%, em média.

As velocidades de relógio a que o 8088 corria iam desde os 4,77MHz originais, até aos 10MHz, velocidade a que eram mais ou menos equivalentes a um 8086 a 8MHz, que foi, sem dúvida, o mais divulgado. Estes processadores foram largamente produzidos por outras companhias e é muito provável que, se o seu PC é 8088, o *chip* que tem no «coração» não seja INTEL.

Os 8086 estão agora praticamente obsoletos, vendendo–se apenas pelo seu baixo custo, ou nas suas versões de baixo consumo para os portáteis autónomos. Hoje em dia, um PC baseado nestes microprocessadores já não é aconselhado a ninguém, a não ser que o seu reduzido custo seja um imperativo e talvez nem mesmo isso justifique a aquisição.

Acrescente–se que estes são os processadores da linha IBM PC original e da linha PC/XT, há muito abandonadas pela *Big Blue*.

José Orlando Pereira

“Os 8086 estão agora praticamente obsoletos, vendendo–se apenas pelo seu baixo custo, ou nas suas versões de baixo consumo para os portáteis autónomos.”

COLABORAÇÃO DO LEITOR

AGENDA



O nosso leitor Guilherme Santos oferece-nos uma AGENDA. À primeira vista, diríamos que se trata de mais uma «Agenda» para números de telefone e moradas, igual a tantas outras que encontramos a funcionar na maioria dos computadores. Na realidade é isso, mas com algo mais que pensamos justificar a sua inclusão nesta rubrica — uma boa demonstração da «arte» de programar em dBASE. E duas opções: uma executável a partir do DOS (compilada com Clipper Summer 87) e outra para correr no ambiente dBASE. Aqui reproduzimos a mensagem do Guilherme Santos a propósito do seu trabalho:

Uma agenda é sempre útil a toda a gente. Uma agenda facilmente actualizável ainda mais. Esta Agenda, para quem o computador já faz parte do seu dia-a-dia, é uma ferramenta simples mas eficaz. Nela podemos anotar os endereços que mais necessitamos, os números de telefone, os aniversários que não convém esquecer e ainda um pequeno espaço para observações. Posto isto vamos às explicações necessárias para a instalação e utilização do programa:

Para a versão compilada basta criar uma directoria «Agenda» numa disquete ou no disco duro e copiar para lá os seguintes ficheiros:

Agenda.exe, Agenda.dbf, Nome.ndx e Apel.ndx.

Na versão dBase devem ser copiados os seguintes ficheiros para a directoria do dBase III:

Agenda.prg, Intro.prg, Consu.prg, Alt.prg, Apa.prg, Imp.prg, Reindex.prg, Agenda.dbf, Nome.ndx e Apel.ndx.

Se possuir o dBase IV proceda da mesma forma. Ao fazer correr o utilitário serão criados ficheiros com a extensão .DBO que vão substituir os ficheiros .PRG. Estes novos ficheiros pseudo-compilados correm mais depressa e quando todos os PRG tiveram corrido podem ser retirados e guardados para qualquer eventualidade. O programa foi feito em dBase III Plus e mais tarde alterado e melhorado em dBase IV. Para correr o utilitário basta escrever DO AGENDA na *prompt* do dBASE ou apenas AGENDA na versão compilada.

A base de dados contem os seguintes campos:

NOME	**(36 caracteres)**
APELIDO	**(24 caracteres)**
MORADA	**(40 caracteres)**
LOCALIDADE	**(25 caracteres)**
CODIGO POSTAL	**(26 caracteres)**
TELEFONE	**(20 caracteres)**

ANIVER (campo data)
OBSERVAÇÕES (100 caracteres)

Permite as seguintes opções:

Introdução de dados

Ao introduzir novos dados será necessário confirmá-los, de seguida decidir se quer introduzir mais. Se em qualquer altura desejarmos abandonar a introdução de dados basta premir **PgDn**, não confirmar e sair.

Consultar dados

Permite fazer consulta por nome, apelido e a todos os registos. Basta teclar o primeiro nome ou apelido ou apenas a primeira letra do nome ou apelido para iniciar uma consulta. Se existirem nomes iguais aparece o primeiro de acordo com a indexação. Por exemplo: JOSÉ MANUEL aparece antes de JOSÉ VICENTE, etc., mas com **PgDn** podemos passar aos seguintes até encontrar o desejado. É evidente que se usarmos nome e sobrenome encontramos logo o que se deseja. Por cada registo editado é-nos dada a hipótese de sair ou continuar.

Alterar dados

Permite alterar qualquer ficheiro com pesquisa por nome e funciona nos mesmos moldes da consulta.

Apagar dados

Permite apagar qualquer registo depois de o editar no ecrã e de confirmarmos esse desejo. Caso contrário podemos abandonar esta opção.

Imprimir listagens

Nem sempre dá jeito fazer a consulta no computador. Por isso é possível tirar listagens por «telefone + nome», «listagem de endereços» e «listagem de Aniversários». Para imprimir ter em atenção as mensagens para cada caso quanto ao tipo de caracteres. As mensagens referem-se aos da minha impressora. No caso de 20 cpi, noutras será 12 cpi condensado.

Reindexar a base de dados

Para tornar mais rápida a saída da «Introdução de dados», decidi criar esta opção, que deve ser usada sempre que necessário para refazer as indexações. Se foram introduzidos dados ou alterados os existentes deve-se reindexar para que as listagens e as consultas funcionem perfeitamente.

As vantagens desta alteração só são visíveis quando a base de dados já tem mais de 100 registos.

Para quem utiliza o programa CATALOGO publicado na **Spooler** Nº 9 será conveniente criar uma subdirectoria na directoria dBASE e deixar apenas o ficheiro AGENDA.PRG na directoria dBASE. Editar todos os ficheiros com extensão .PRG e alterar os caminhos. Por exemplo, se criar uma subdirectoria com o nome AGENDA, onde está «USE AGENDA» passará a estar «USE AGENDA\AGENDA»; onde existe «DO AGENDA» deve ser mudado para «DO AGENDA\AGENDA» etc. É importante não esquecer nada para que tudo trabalhe bem.

Guilherme Santos

ASTÚCIAS

ENCRIPTAÇÃO/ DESCRIPTAÇÃO DE STRINGS

Certamente que todos já sentimos o desejo de que todos os nossos dados fossem só e exclusivamente acedidos por nós próprios, uns por intermédio de *passwords*, outros pela incriptação desse mesmos dados.

O que aqui vos proponho é um simples algoritmo exemplo de encriptação de *strings*. Este algoritmo é composto por 3 procedimentos, **InverteString**, **Encripta**, e **Desencripta**. O funcionamento do programa é o seguinte:

Em primeiro lugar é lida do teclado uma mensagem, que vai ser encriptada.

O modo de encriptação que eu escolhi e que pode variar com a imaginação e necessidades de cada um, foi o de adicionar a cada carácter dessa mensagem o valor a que corresponde a posição desse mesmo carácter na *string* e depois invertê-la. O resultado será uma verdadeira «sopa de letras» ou de caracteres que tornará a decifragem da informação uma grande dor de cabeça a quem não possuir este algoritmo. Para reconstruir a mensagem tal como originalmente ela se apresentava deve-se utilizar o procedimento **Desencripta**, que efectuará o processo oposto ao do procedimento **Encripta**. Este algoritmo requer a utlização de uma variável do tipo **str** (já declarada como uma *string* de 255 posições) que irá conter a mensagem. Essa variável vai ser usada como nos procedimentos **Encripta** e **Desencripta**:

Paulo Franco

PROGRAM ENCRIPTAGEM (INPUT, OUTPUT);

```
type
        str = string [255];

procedure InverteString (var s : str); {Inverte uma string.}
var
        a, Sum : byte;
        IS : string [255];
begin
        Sum := 0;
        IS [0] := chr (length(s));
        for a := length(s) downto 1 do
        begin
                Sum := Sum + 1;
                IS[Sum] := s[a];
        end; {for}
        s := IS;
end; {InverteString.}

procedure Encripta (var s : str); {Encripta uma string.}
var
        a : bute;
        Sum : word;
begin
        for a : = 1 to length(s) do
        begin
                Sum : = ord(s[a]) + a;
                If Sum > 255 then
                        Sum:= Sum - 255;
                S[a] := chr (Sum);
        end; {for}
        InverteString(s);
end; {Encripta.}

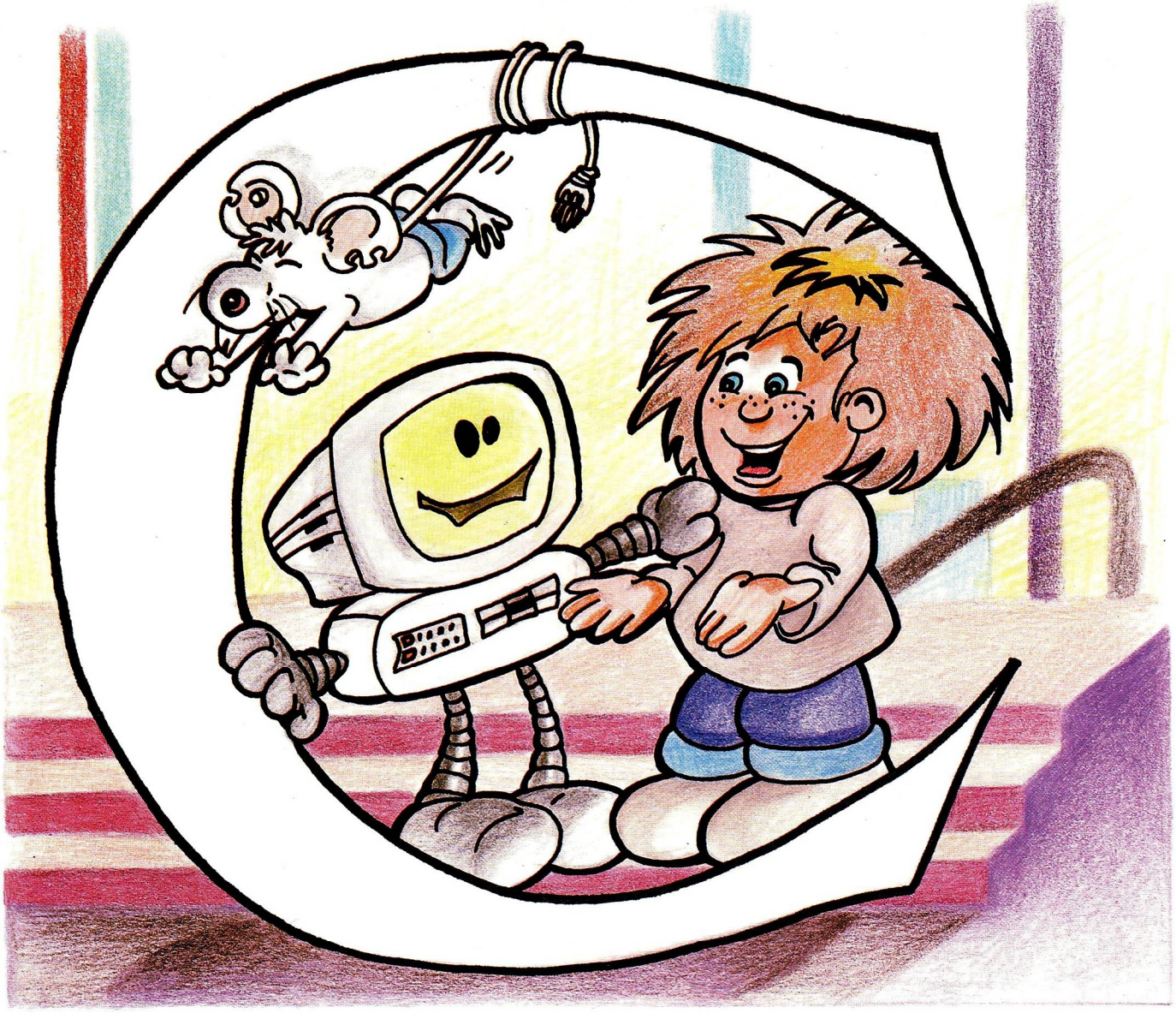
procedure Desencripta (var s : str); {Desencripta uma string.}
var
        a : byte;
        Sum : integer;
begin
        InverteString(s);
        for a := 1 to length(s) do
        begin
                Sum := Odr(s[a]) - a;
                if Sum < 1 then
                        Sum := 255 + Sum;
                s[a] := chr (Sum);
        end; {for}
end; {Desencripta.}

var
        Mensagem : str;

begin
        readln (Mensagem);
        Encripta (Mensagem);
        writeln (Mensagem);
        Desencripta (Mensagem);
        writeln (Mensagem);
end.
```

PROGRAMAR EM C

(PARTE I)



O ESQUEMA DESTE CURSO

Este artigo é o primeiro de uma série de cinco sobre a linguagem C. Nesta sequência de artigos vão focar-se os aspectos mais importantes desta linguagem, partindo das bases (para permitir que, mesmo quem desconhece o C, aprenda a «mexer» nesta excepcional linguagem de programação) até chegarmos aos aspectos mais avançados. Todos artigos serão acompanhados (na disquete) dos programas-exemplo focados ao longo mesmo. Será também desenvolvida uma calculadora, a qual irá ser aperfeiçoada ao longo dos artigos. No final do nosso curso será feito um teste aos conhecimentos do leitor, com uma classificação para efeitos de atribuição de um prémio.

BREVE INTRODUÇÃO AO C

A linguagem C é, talvez, a mais utilizada hoje em dia e tem qualidades que a tornam uma das linguagens de programação mais eficientes. De entre essas qualidades, destaca-se: flexibilidade, portabilidade e estrutura. Pode-se considerar uma linguagem de **médio-nível**, pois combina a eficiência do *Assembly* (baixo-nível) com a estrutura de linguagens como o Pascal (alto-nível). O C foi pela primeira vez implementado no sistema operativo UNIX num DEC PDP-11 (em 1978), por Dennis Ritchie (seu criador). Grande parte das ideias do C partiram de uma outra linguagem, a BCPL, desenvolvida por Martin Richards. A influencia da BCPL no C não foi directa, pois esta evolução fez-se através de uma outra linguagem,

«B», desenvolvida por Ken Thompson. Com o crescimento da popularidade do C, ao longo dos anos, foram surgindo alterações na sua versão original. Estas novas versões, na maioria dos casos, perderam uma das qualidades mais importantes do C, a sua portabilidadede. Sendo esta uma das características mais importantes da linguagem, tornava-se necessário fazer qualquer coisa para impedir esta invasão de versões da linguagem. Foi assim que,em 1983, a *American National Standards Institute* (ANSI) nomeou uma comissão que teve como objectivo a criação do «C Standard». Hoje em dia a maioria dos compiladores de C respeitam o chamado «ANSI C». Qualquer programador que escreva um programa em C respeitando o ANSI C, poderá ter a certeza de que o seu programa poderá ser compilado com sucesso em qualquer outro computador, desde que sobre um compilador de ANSI C. O C é, actualmente, uma das linguagens mais utilizadas pelos programadores profissionais. Algumas das aplicações mais comuns do C destinam-se à realização de:

- Sistemas operativos
- Interpretadores / compiladores
- Editores
- Gestores de bases de dados
- Utilitários

No que diz respeito ao compilador que o leitor deverá utilizar, serve qualquer um que respeite o ANSI C. No entanto, eu aconselho o Turbo C 2.0 da Borland pois, para além de respeitar o ANSI C, inclui ainda muitas outras funções de grande utilidade. A última das cinco lições deste curso vai ser totalmente dedicada a este compilador.

ESTRUTURA GERAL DE UM PROGRAMAR EM C

Antes de mais, convém salientar alguns aspectos carácteristicos de um programa em C. Para começar, o compilador de C diferencia as letras maiúsculas das minúsculas. Por exemplo: "printf" é diferente de "PRINTF" ou ainda "PrInTf". Todo o texto que estiver entre "/* " e "*/" é ignorado, sendo-lhe dado o nome de «comentário» e deve ser utilizado para salientar aspectos importantes relativos ao programa. Um programa em C é constituído por variáveis e por funções que se «chamam» umas às outras. Por **função** entende-se um bloco de código que pode ser «mandado» executar a partir de qualquer ponto do programa. Quando a função chega ao fim da sua execução, volta para o local do programa de onde foi chamada, retornando opcionalmente um determinado valor. As funções serão descritas posteriormente de uma forma pormenorizada. De entre as várias funções que um programa pode ter, existe uma que tem de estar sempre presente, a função "main", sendo nesta função que é iniciada a execução do programa, e sendo também nela que termina a execução do mesmo. De seguida vamos ver a estrutura básica de um programa escrito em C:

DECLARAÇÕES GLOBAIS

```
tipo-de-retorno main(lista de parametros)
{
  declarações locais
  sequência de instruções
}
declaração das funções definidas pelo programador
```

De seguida o leitor vai ver um programa em C. Este exemplo simplesmente imprime a palavra "OLÁ!" no monitor:

```
#include <stdio.h>
/* Inclui informação referente à função printf, */
/* que faz parte da Standard Library */

/* A função main é declarada do tipo void. */
/* void informa que a main não retorna nenhum
valor */
/* A função main não tem parametros de entrada */
void main()
{           /* é aqui que começa o programa */
  printf("OLÁ!"); /* Escreve no monitor "OLÁ!" */
}           /* Termina a execução do programa */
```

Como o leitor deve ter reparado, o início da função **main** é determinado com o carácter "{", e o fim com o carácter "}". Para o C, estes dois caracteres representam, respectivamente, o início e o fim de um bloco de instruções. Para quem sabe Pascal, estes dois caracteres são comparáveis às instruções Begin e End.

IDENTIFICADORES

Em C, nomes de variáveis, *label's*, funções e outros objectos definidos pelo programador (**#define xxx** , por exemplo) são chamados identificadores. Devem começar sempre por uma letra ou pelo carácter "_"; os caracteres que se seguem poderão ser letras, números, ou o carácter "_". O número de caracteres de um identificador poderá ser no máximo 32. Num programa em C, não poderá existir mais que um identificador com o mesmo nome. No entanto, podem existir no mesmo programa os dois identificadores que se seguem: «total» e «Total». De seguida vamos ver alguns exemplos de identificadores, uns correctos e outros incorrectos:

Correctos	Incorrectos
abc_do_C	OLÁ
__123_ab	1vez
OkToGo	ze&manel

> **Em C, nomes de variáveis, *label's*, funções e outros objectos definidos pelo programador (#define xxx , por exemplo) são chamados identificadores.**

DECLARAÇÃO DE VARIÁVEIS USANDO OS TIPOS BÁSICOS DO C

As variávei são locais da memória onde podem ser guardados valores. O conteúdo de uma variável poderá ser alterado, sempre que necessário. Existem dois tipos principais de variáveis: as **globais** e as **locais**. As variáveis globais são «visíveis» para todo o programa. No caso das locais, estas são definidas dentro de funções. São criadas quando a função é chamada e destruídas quando a função termina a sua execução. Só são «visíveis» dentro da função em que são declaradas. As variáveis podem ainda subdividir-se em vários tipos quanto ao seu conteúdo. Nas implementações mais comuns do C, os tipos básicos são:

Tipo	Tamanho (bytes)	Gama de valores
char	1	-128 a 127
int	2	-32768 a 32767
float	4	3.4E-38 a 3.4E+38
double	8	1.7E-308 a 1.7E+308

Existem no entanto os modificadores de tipo que têm como função «alargar» o número de tipos básicos do C. Eles são: **signed, unsigned, long** e **short.** Os tipos Básicos depois de alterados com os modificadores, podem assumir valores diferentes:

Tipo	Tamanho (bytes)	Gama de valores
unsigned char	1	0 a 255
signed char	1	-128 a 127
unsigned int	2	0 a 65535
signed int	2	-32768 a 32767
short int	2	-32768 a 32767
unsigned short int	2	0 a 65535
signed short int	2	-32768 a 32767
long int	4	-2147483648 a 2147483647
signed long int	4	-2147483648 a 2147483647
unsigned long int	4	0 a 4294967295

Existe ainda um outro tipo que serve, tanto para indicar funções sem valor de retorno, como definir *pointers* de um tipo genérico. Este tipo é o **void**. Agora só nos falta saber como declarar uma variável.. A sintaxe é a seguinte:

```
tipo nome [ = valor_inicial] [, nome [ =
valor_inicial] ...] ;
```

Seguem-se alguns exemplos:

```
char carácter;      /* define um char */
unsigned int a=8;   /* define um inteiro sem sinal */
float x, y=3.5, z;  /* define três float */
void *ptr;          /* define um pointer de tipo
                       genérico */
```

No que diz respeito aos locais da declaração, se forem globais são declaradas em qualquer parte do programa, desde que fora de funções. Se forem locais, são declaradas no início do código da função.

```
int b=1000;   /* variável global */
void main()
{
  float f;    /* variável local */
  .
  .
  .
}
```

INICIALIZAÇÃO E AFECTAÇÃO DE VARIÁVEIS

Em C a inicialização e a afectação de variáveis é muito simples. Qualquer variável global que não seja inicializada pelo programador, é, por defeito, inicializada a zero pelo compilador. O bloco de código que se segue exemplifica bem estas duas situações:

```
int a=25, b=50, c, d; /* inicializa "a" a 25 e "b" a
                         50 */
float f;              /* a variável "f" é
                         inicializada a 0 pelo
                         compilador */
char chr='A';         /* inicializa "chr" com o
                         valor ASCII de "A" que é 65 */
.
.
.
a=-34;      /* coloca -34 em "a" */
b=c=0       /* coloca 0 em "b" e "c" */
d=c         /* coloca o valor de "c" em "d" */
f=56.56;    /* coloca 56.56 em "f" */
```

EXPRESSÕES E OPERADORES

Básicamente, uma expressão poder-se-á comparar a uma fórmula. Esta é avaliada e produz um determinado resultado. Quando a expressão a avaliar é do tipo lógico ou relacional, o resultado é sempre do tipo verdadeiro ou falso. Para o C, falso significa o valor 0 e verdadeiro o valor 1; no entanto, qualquer valor diferente de zero é avaliado como sendo verdadeiro. As expressões são constituídas por «operadores» e «operandos». Os operandos são as variáveis e constantes intervenientes na expressão. Quanto aos operadores, são eles que indicam as operações a efectuar. Em C existem os seguintes tipos de operadores: **aritméticos**, **relacionais**, **lógicos**, **bit-a-bit**, **afectação** e **pointer**.

> **“Os operandos são as variáveis e constantes intervenientes na expressão.”**

OPERADORES ARITMÉTICOS

operador	acção
-	subtracção, menos unário
+	adição
*	multiplicação
/	divisão
%	resto da divisão
--	decremento
++	incremento

OPERADORES RELACIONAIS:

operador	significado
>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
==	igual
!=	diferente

OPERADORES LÓGICOS:

operador	significado
&&	AND
||	OR
!	NOT

OPERADORES BIT-A-BIT:

operador	significado
&	AND
|	OR
^	XOR
~	complemento a um
>>	deslocamento à direita
<<	deslocamento à esquerda

De todos estes operadores vamos estudar mais detalhadamente só alguns, pois os outros tornam-se evidentes, começando pelos operadores (++) e (—). São normalmente usados para simplificar o programa e a sua codificação é da seguinte forma: “++variável” ou “variável++”. A diferença entre estas duas formas, é que na primeira, a variável é incrementada antes da expressão ser avaliada, enquanto que na segunda a incrementação só é feita depois da expressão ser avaliada. Por exemplo:

```
a=10;
b=a++-3; /* coloca 10-3 em b e depois incrementa
            a */
```

```
c=++b;    /* incrementa b e coloca o resultado em
             c (11) */
```

Quanto aos operadores relacionais e lógicos vamos ver só alguns exemplos:

```
(45==46)              falso (0)
(34 < 50)             verdadeiro (1)
((34!=43) && (!56))   falso (0)
((45 || 0) == (4 < 6))  verdadeiro (1)
```

Os operadores **bit-a-bit** merecem um pouco mais de insistência. Suponha que tem uma variável do tipo **unsigned char**, "a" com o valor inicial de 7. Se fizermos um **XOR** de "a" com 9, obtemos o resultado de 14:

```
a=a^9;

   00000111   (7)
^  00001001   (9)
  ----------
   00001110   (14)
```

de seguida vamos fazer um **AND** de "a" com 2, obtemos 2:

```
a=a&2;

   00001110   (14)
&  00000010   (2)
  ----------
   00000010   (2)
```

vamos agora ver o complemento a um, obtendo o resultado 253:

```
a=~a;

~  00000010   (2)
  ----------
   11111101   (253)
```

só nos falta ver como funcionam os operadores de deslocamento dos bits dentro de um **char** ou **int**. Continuando com a variável **a** com o valor de 253, vamos primeiro deslocar os seus bits duas posições para a esquerda:

```
a=a<<2;

antes   11111101   (253)
depois  11110100   (244)
```

IMPORTANTE: os bits introduzidos (neste caso 2 à direita) têm sempre o valor 0.

agora deslocamos uma posição para a direita

```
a=a>>1;

antes   11110100   (244)
depois  01111010   (122)
```

Para além destes operadores, existem ainda os de **afectação**. Em C, o operador de afectação é um simples sinal de igual. No entanto, o C permite simplificar ainda mais a afectação, quando ela é do tipo:

```
variável_1 = variável_1 operador expressão;
```

Por exemplo:

```
a=a+b;
x=x/2;
a=a>>2;
```

afectações deste tipo poderão ser simplificadas da seguinte forma:

```
variável_1 operador= expressão;
```

No caso dos exemplos anteriores:

```
a+=b;
x/=2;
a>>=2;
```

Esta simplificação poderá ser feita com todos os operadores binários do C.

Existe ainda um outro operador de grande importância no C. É a ",". Normalmente é usada para separar os parâmetros de uma função, mas também poderá ser utilizado na separação de expressões. Por exemplo:

```
x=10;
y=(x*=2, x/2);   /* multiplica x por 2, e coloca o
                    resultado de x/2 em y (10) */
                    * a ultima expressão da lista é que
                    é utilizada na afectação */
```

Para terminar falta dizer que os operadores de **pointer** serão estudados na próxima lição, quando analisarmos os **pointers.**

LABEL'S

Quem não conhece a instrução GOTO, do BÁSIC ?. Em C existe também a instrucção **goto** (em minúsculas). Esta instrução tem como finalidade desviar a execução do programa para um determinado **label**. A sua sintaxe é:

```
goto nome_do_label;
```

Um pequeno exemplo vai clarificar o assunto:

```
#include <stdio.h>
/* Inclui informação referente à função printf */

/* A função main é do tipo void sem parametros */
void main()
{
 printf("Olá")            /* Imprime no ecrã "Olá"
                             */
 goto fim;                /* Salta para o label fim */
 printf(", tudo fixe!!!"); /* Nunca executa esta
                             instrução */
 fim:                     /* local do label fim */
 printf("\nadeus");       /* muda de linha e escreve
                             "adeus" */
}
```

Resta salientar que esta instrução deve ser evitada tanto quanto possível, pois o seu uso torna o programa confuso e difícil de ler. Além disso, o C é rico em estruturas de controlo, sendo a sua utilização injustificado. No entanto, como o C foi criado para ser usado em substituição da linguagem-máquina, a sua inclusão é justificada

CONTROLE DO FLUXO (IF'S E SWITCH'S)

O C suporta dois tipos de controle do fluxo: **if** e **switch**. Existe ainda o operador **?**, que é uma alternativa em alguns casos ao **if**. Vamos começar pelo **if**. A sua forma geral é:

```
if (expressão)
  statement;     /* se a expressão for verdadeira */
else
  statement;     /* se a expressão for falsa */
```

De acordo com a sintaxe do C, **statement** poderá ser: uma instrução, um bloco de instruções ou então nada (só com ";"). A cláusula **else** é opcional. No entanto, é necessário ter cuidado com a sucessão de «if's». Observe com atenção os seguintes casos:

```
1   if (a==b)                    2   if (a==b) {
      if (b==c) flag=0;                if (b==c) flag=0;
    else x=1;                          }
                                     else x=1;
```

> **Pode parecer estranho, mas nenhuma das palavras chave do ANSI C tem como função comunicar com utilizador.**

No primeiro caso o **else** pertence ao **if (b==c)**; para se fazer com que o **else** pertença ao **if (a==b)**, dever-se-á fazer como no segundo caso. Quando um **if** é utilizado só com expressões, poderá ser substituído pelo operador **?**, com maior eficiência. Por exemplo:

```
if (c==d) x=20*c;
else x=40+d;
```

deverá ser substituído por:

```
x=(c==d) ? 20*c : 40+d;
```

Resta-nos o **switch**. Este é utilizado quando existem várias opções de escolha. A sua sintaxe geral é:

```
switch (expressão) {
    case caso_1:
       sequência de instruções
       break;
    case caso_2:
       sequência de instruções
       break;
    .
    .
    .
    default:
       sequência de instruções
    }
```

O **switch** vai comparando de uma forma sequêncial o resultado da expressão com a lista de constantes inteiras ou do tipo **char**. Quando é encontrada uma que condiz, é executada a sequência de instruções que lhe está associada. Se no fim desta sequência de instruções não existir a palavra-chave **break** a execução continuará na sequência de instruções do caso seguinte até que seja encontrado um **break** ou o fim do **switch**. Caso exista o **break**, a execução continua na primeira instrução a seguir ao bloco do **switch**. A sequência de instruções associada ao caso **default** é executada se a expressão não corresponder a nenhum dos casos listados. Este caso é opcional. Segue-se um pequeno exemplo:

```
.
.
.
switch (tecla) {  /* contem uma tecla pressionada
                     pelo utilizador */
    case 'A':     /* se a tecla for A */
       A();       /* chama a função A() */
       break;     /* sai do switch */
    case 'q':     /* se a tecla for q */
                  /* sequência nula, continua no
                     próximo caso */
    case 'Q':     /* se a tecla for Q */
       sai();     /* se a tecla for q ou Q chama
                     função sai() */
       break;     /* sai do switch */
    default:      /* se não for nenhum dos casos
                     acima assinalados */
       printf("Tecla sem significádo");
    }
.
.
.
```

COMUNICAÇÃO COM O UTILIZADOR (SCANF E PRINTF)

Pode parecer estranho, mas nenhuma das palavras chave do ANSI C tem como função comunicar com utilizador. Para isto torna-se necessário recorrer à *Standard Library* (estudada detalhadamente mais à frente). É lá que estão definidas as funções que nos permitem executar este tipo de operações. De momento só nos interessam duas funções da *Standard* **Library**, **printf** e **scanf**. A função **printf** permite-nos enviar caracteres e números para o ecrã. Imagine que pretende escrever "Olá , tudo bem?". Para o fazer, poderia escrever a seguinte linha:

```
printf("Olá, tudo bem?");
```

Mas se, por exemplo, a sua necessidade fosse a de mostrar o valor de uma variável do tipo inteiro, escreveria:

```
printf("O valor da variável é %d .\n", variável);
```

A sequência **%d** será substituída pelo valor da variável. Quanto à sequência **\n**, faz o cursor mudar para o inicio da linha que se segue. No entanto, não são só inteiros que pretendemos para mostrar; segue-se uma tabela que indica os tipos de dados que poderemos enviar para o ecrã.

código	apresentação
%c	carácter simples
%i, %d	decimal
%e	notação científica
%f	vírgula flutuante
%g	utiliza o mais curto (% e ou %f)
%o	octal
%s	string (termina com '\0')
%u	decimal sem sinal
%x	hexadecimal
%p	pointer
%%	o carácter %

Segue-se um pequeno exemplo:

```
#include <stdio.h>
float f=3.1415;
int i=-100;
unsigned int u=35;
void main()
{
     printf("\nf=%f\ni=%i\nu=%u", f, i, u);
}
```

E agora vamos ver como poderemos ler dados do teclado. Esta operação poderá ser feita através da função **scanf**. Esta função, em parte, é parecida com a função **printf**, com a diferença que os argumentos, em vez de passarmos o seu valor, vamos passar o seu endereço de

memória. O próximo exemplo lê do teclado um inteiro para a variável "a".

```
scanf("%d", &a);
```

O operador **&** (não é um **AND**, pois só tem um operando) será explicado na próxima lição. Por agora só interessa saber que ele nos dá o endereço da variável que se lhe segue. Outro exemplo do uso desta função seria:

```
int a, b;
char c;
.
.
.
scanf("%d%c%d", &a, &c, &b);
```

No entanto, existem algumas diferenças nos códigos que determinam os argumentos em relação à função **prinf**. Segue-se a tabela:

código	significado
%c	lê um carácter simples
%i	lê um inteiro decimal, octal ou hexadecimal
%d	lê um inteiro decimal
%e, %f	lê um número de vírgula flutuante
%o	lê um octal
%s	lê uma string
%x	hexadecimal

A sequência **%i** lê um inteiro octal se o número for precedido de um **0**, um inteiro hexadecimal se for precedido de **0x** ou **0X**, caso contrário lê um inteiro decimal. Segue-se um exemplo,

```
#include <stdio.h>
int a;
void main()
{
    printf("Escreva o número 33\n—
    >");
    scanf("%d", &a);
     if (a==33) printf("Ok!\n");
     else printf("BURRO!!! Era 33 e
    não %d.\n", a);
}
```

Para terminar, a utilização de qualquer uma destas funções (**scanf** ou **printf**), deverá ser incluida a linha **#include <stdio.h>** no início do programa. A directiva **#include** será estudada mais tarde.

PROGRAMA EXEMPLO (VERSÃO 1)

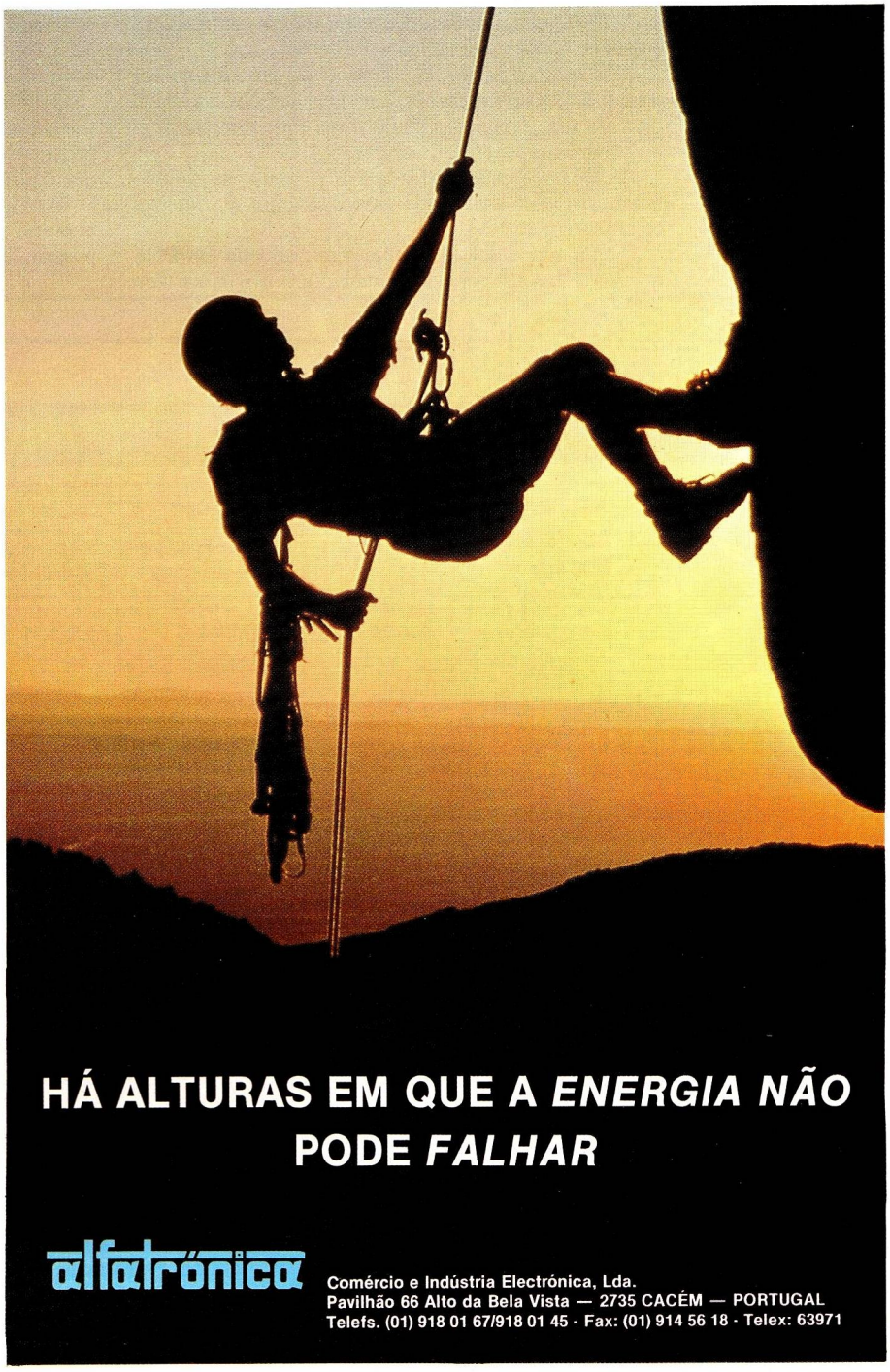
Esta primeira versão da nossa calculadora é muito simples, interpreta expressões do tipo «constante operador constante», em que constante representa um número decimal, octal ou hexadecimal. O operador poderá ser um dos quatro (+, –, *, /). Depois de executada uma operação, mostra o resultado nas três bases (DECimal, OCTal e HEXadecimal), e pergunta se o utilizador pretende resolver outra expressão. A fonte do programa poderá ser encontrada no ficheiro CALC0.C.

A PRÓXIMA LIÇÃO

A próxima lição irá focar os tópicos:

- Constantes
- Arrays
- Pointers
- Strings
- Structs
- Unions
- Loops (for's e while's)
- Nova versão da nossa calculadora

Carlos Ladeira



D. O. S.

– DO NASCIMENTO À MAIORIDADE

(6ª Parte)

Continuamos neste número a analisar mais alguns comandos do sistema operativo DOS, procurando contribuir para que um número crescente de utilizadores possam manipular os comandos que estão à sua disposição com facilidade e correcção, sem necessidade de se "perderem" em manuais mais ou menos esotéricos, e a maior parte das vezes em lingua estrangeira.

Apresentamos desta vez dois comandos de configuração e dois comandos de manipulação de ficheiros.

Gostariamos de chamar a atenção para os utilizadores do DR 5.0, para a notável capacidade do comando REN para efectuar a mudança de ficheiros entre directorios e/ou subdirectórios

É claro que esta mudança não é «real» no sentido fisico, visto que o comando apenas se limita a actualizar os endereços na FAT.

Face à metodologia que temos que adoptar em DOS quando temos necessidade de mudar ficheiros de um lado para o outro, ou seja, utilizar o comando COPY ou XCOPY para a 1ª fase e o comando DEL ou ERASE para a 2ª fase, torna-se evidente a vantagem da utilização deste comando que efectua as duas fases de uma só vez.

O comando PROMPT apresenta algumas virtualidades, nomeadamente na possibilidade que nos dá de podermos ter na linha de comando algumas informações importantes, como sejam a localização do ponto em que nos encontramos na árvore de directórios e subdirectórios ou outras de acordo com o critério do utilizador.

O comando CHCP faz parte de um conjunto de ficheiros (mencionados na ficha respectiva) e que analisaremos em próximos artigos, que permitem configurar os teclados que é possivel utilizar.

Esperamos com estas fichas contribuir para esclarecer algumas dúvidas que utilizadores menos experientes possam ter sobre estas questões.

D. O. S. — RD (RMDIR) — FICHA Nº. 10

COMANDO	RD (RMDIR)
1ª. VERSÃO	DOS 3.21
ÚLTIMA VERSÃO	DOS 2.0 DR 3.21
NÚMERO DE ALTERAÇÕES	0

SINTAXE

RD [DRIVE] [CAMINHO/PATH]

GRUPO	Interno
FUNÇÃO	Directórios
INTERRUPTORES (DOS)	
INTERRUPTORES (DR 5.0)	

DESCRIÇÃO

Este comando permite apagar directórios e subdirectórios, desde que estejam vazios de ficheiros.

EXEMPLOS

RD\TESTE
Apaga o directório\teste do drive corrente

RD A:\CARTAS\ANTIGAS
Apaga o subdirectório\antigas do directório\cartas situadas no drive A:

DICAS

1 – Antes de apagar um directório tem de remover todos os subdirectórios dele dependentes e respectivos ficheiros.

2 – Antes de apagar um subdirectório, tem de apagar todos os ficheiros nele contidos.

3 – Antes de apagar um subdirectório, certifique-se de que ele não está afectado pelos comandos:
– SUBST
– JOIN

4 – Na versão do DR 5.0 se ao pretender apagar um directório ou subdirectório aparentemente vazio e receber a mensagem:
"Directory not empty or in use"
Utilize o comando **xdir** que lhe revelará se existem alguns ficheiros escondidos ou protegidos por password. Se assim acontecer tem de os apagar primeiro e só depois poderá apagar o seu directório ou subdirectório.

D. O. S. — REN (RENAME) — FICHA Nº. 11

COMANDO REN (RENAME)

1ª. VERSÃO DOS 3.21 DR

ÚLTIMA VERSÃO DOS 3.21 DR

NÚMERO DE ALTERAÇÕES 0

SINTAXE

REN [drive:][path]fich1 fich2

fich1 – ficheiro a que se quer alterar o nome
fich2 – ficheiro com a nova denominação.

GRUPO Externo

FUNÇÃO Ficheiros

INTERRUPTORES (DOS)

INTERRUPTORES (DR 5.0)

DESCRIÇÃO

EM DOS 3.3 A 5.0

Este comando permite:
1 – Mudar o nome e/ou extensão a um ou mais ficheiros.

EM DR 5.0

Este comando permite:
1 – Mudar o nome e/ou extensão a um ou mais ficheiros
2 – Mudar um ou mais ficheiros de um directório para outro,ou subdirectorio com ou sem alteração do nome e/ou extensão desde que dentro do mesmo drive, eliminando os ficheiros no PATH de origem

EXEMPLOS

DOS

REN *.txt *.doc

Muda as extensões .TXT de todos os ficheiros que as possuam em extensões .DOC, sem alterar os nomes de cada ficheiro.

REN carta.tx cart1.txt

Muda o nome do ficheiro carta.txt para carta1.txt, SEM ALTERAR O CONTEúDO.

DR 5.0

REN carta.txt c:\pf\wp\docs

Desloca o ficheiro carta.txt da directoria onde nos encontramos para a subdirectoria \pt\wp\docs

REN carta.txt carta2.txt

Muda o nome do ficheiro carta.txt para carta2.txt, SEM ALTERAR O CONTEúDO

REN *.txt c:\pf\wp\docs*.doc

Muda todos os ficheiros com a extensão .txt do directório corrente para o subdirectório c:\pt\wp\docs\ e muda as extensões para .doc

DICAS

1 – Em ambas as versões as mudanças quer de nomes, quer de tensões só podem ser realizadas, dentro do mesmo drive, isto é, não pode renomear um ficheiro que se encontra no drive C: e indicar um drive diferente para o ficheiro com o novo nome.
2 – Na versão DR 5.0 a função de deslocação de um ou mais ficheiros, equivale a um comando que há muito tem vindo a ser solicitado pela comunidade de utilizadores DOS, ou seja o comando MOVE, que até aqui só através de utilitários como o Pctools/Pcshell, era possivel executar.
3 – Na versão DR 5.0, para este comando funcionar em ficheiros com Password, este deve ser fornecido à entrada, e manter-se-á inalterado nos ficheiros com novo nome e/ou extensão, ou nos ficheiros que foram movidos de localização.
4 – Na versão DR 5.0 o comando não lhe permite renomear ficheiro(s) para nomes que já existam, dando-lhe nessa situação uma mensagem de erro.

"File already exists"

D. O. S. — CHCP — FICHA Nº. 13

COMANDO CHCP

1ª. VERSÃO DOS 5.0 DR

ÚLTIMA VERSÃO DOS 5.0 DR

NÚMERO DE ALTERAÇÕES 0

SINTAXE CHCP nnn

GRUPO Interno

FUNÇÃO Configuração

INTERRUPTORES (DOS) Nenhum

INTERRUPTORES (DR 5.0) Nenhum

DESCRIÇÃO

Este comando permite selecionar um código de página, (cujos simbolos podem ser verificados nos quadros de caracteres que acompanham normalmente os manuais), para todos os periféricos (impressora,ecran,etc...) que a partir do carregamento deste comando, passam a utilizar o código e simbolos constantes na tabela escolhida.

EXEMPLOS

chcp 860

carrega o código da tabela dos simbolos portugueses

chcp 437

carrega o código da tabela dos simbolos anglo-saxónicos (versão americana)

DICAS

1 – Este comando pertence ao ficheiro AUTOEXEC.BAT

2 – Se digitar CHCP na linha de comando, obterá a informação de qual o código de página com que o seu computador está a trabalhar.

3 – ANTES da linha onde coloca o comando CHCP deve ser colocado o comando KEYB, MODE seguido do comando NLSFUNC e só depois o comando CHCP.

4 – Para este comando trabalhar correctamente, deve ser colocado no CONFIG.SYS, os ficheiros COUNTRY.SYS, PRINTER.SYS e DISPLAY.SYS

D. O. S. — PROMPT — FICHA Nº. 12

COMANDO PROMPT

1ª. VERSÃO DOS 2.0 DR 3.21

ÚLTIMA VERSÃO DOS 2.0 DR 3.21

NÚMERO DE ALTERAÇÕES 0

SINTAXE **PROMPT [Strings]**

STRINGS

$ = Mostra o caracter $
t = Mostra a HORA
d = Mostra a DATA
p = Mostra o caminho (PATH) do directório corrente
v = Mostra a versão do DOS
n = Mostra o Drive onde se encontra
g = Mostra o caracter >
l = Mostra o caracter <
b = Mostra o caracter |
q = Mostra o caracter =
h = Apaga o caracter anterior, e pode ser repetido as vezes que for necessário
e = Caracter ESCAPE
_ = Introduz a sequencia CR LF (Carriage-return and Line Feed) e posiciona o cursor na linha imediatamente inferior

GRUPO Interno

FUNÇÃO Configuração

INTERRUPTORES (DOS)

INTERRUPTORES (DR 5.0)

DESCRIÇÃO

Este comando permite configurar a linha de comando e/ou a disposição de informações (Ex:Hora,Data) visiveis no ecran.

EXEMPLOS

PROMPT PG – Vizualiza constatemente o caminho (PATH) do directório onde nos encontramos

C:\DOS>
C:\PT\WSTAR>

PROMPT HORA=t_DATA=$d $_$p$g - Vizualiza a Hora com os minutos segundos e decimos de segundos seguida na linha de baixo da Data e na 3ª linha o Path onde se encontra.

PROMPT HORA=t hhhhhh $_DATA=$d $_$p$g - Vizualiza a Hora com os minutos e SEM os segundos e decimos de segundos seguido dos mesmos elementos da versão anterior

DICAS

1 – Em qualquer altura na linha de comando, o utilizador pode modificar o PROMPT que carregou através do seu Autoexec.bat, digitando novo PROMPT.

2 – Se tiver carregado o ficheiro ANSI.SYS no seu CONFIG.SYS, poderá utilizar as sequencias de escape deste driver de periferico (a analizar em artigo posterior dada a sua extensão), o que lhe possibilitará configurar o seu ecran com diversas variáveis.

3 – O PROMPT que inclui a Hora, só mostra a nova hora após carregar em ENTER.

D. O. S. — TYPE — FICHA Nº. 14

COMANDO TYPE

1ª. VERSÃO DOS 1.0 DR 3.21

ÚLTIMA VERSÃO DOS 5.0 DR 5.0

NÚMERO DE ALTERAÇÕES 1 DOS 5.0 1 DR 5.0

SINTAXE DOS

TYPE [drive:] ficheiro [|MORE]

DR 5.0

TYPE [drive:] ficheiro [PASSWORD] [/P]

GRUPO DOS até 4.01 e DR 5.0

FUNÇÃO Interno

INTERRUPTORES (DOS) | MORE
Em conjunto com este comando, permite visualizar o conteúdo de um ficheiro ecran a ecran

INTERRUPTORES (DR 5.0) /P
Permite a vizualição do conteúdo de um ficheiro ecran a ecran

DESCRIÇÃO

Este comando permite:

1 – visualização do conteúdo de um ficheiro de texto no ecran
2 – impressão sem formatação de texto.

EXEMPLOS

DOS

TYPE carta.txt |more
visualiza o ficheiro carta.txt no ecran parando ao fim de 25 linhas e solicita do operador que carregue numa tecla para apresentar a nova página.
TYPE carta.txt >prn
Imprime o conteúdo de carta.txt na impressora

DR 5.0

TYPE carta.txt /p
visualiza o ficheiro carta.txt no ecran parando ao fim de 25 linhas e solicita do operador que carregue numa tecla para apresentar a nova página.
TYPE carta.txt;teste
Visualiza o conteúdo do ficheiro carta.txt que foi protegido com o password TESTE

DICAS

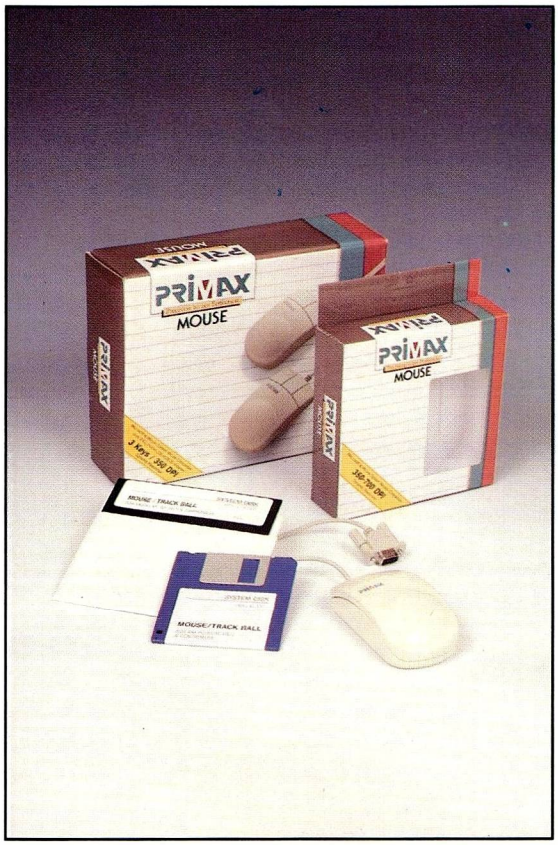
MONTRA Spooler

10% de desconto
para assinantes em todos os produtos da Montra

TODOS OS PREÇOS APRESENTADOS JÁ INCLUEM O IVA.

Mais produtos, criteriosamente seleccionados, aos melhores preços, encontram-se à disposição do leitor. Sem querermos aumentar demasiado o espaço dedicado à «Montra Spooler», pretendemos expor aqui os produtos mais procurados pelo leitor. Contamos, como sempre, com as suas sugestões.
Já sabe que para encomendar os produtos constantes deste catálogo, basta preencher o cupão publicado para o efeito, e remetê-lo para os nossos serviços. O leitor NÃO deve ENVIAR qualquer forma de PAGAMENTO. Todos os produtos serão enviados à cobrança e, aos preços indicados, serão acrescidos os portes e taxas de cobrança. Devido ao facto de alguns dos produtos aqui apresentados esgotarem rapidamente, pensamos ser este o melhor processo para evitar contratempos como seja encomendar produtos que já se encontram esgotados, etc. etc..

RATOS PRIMAX

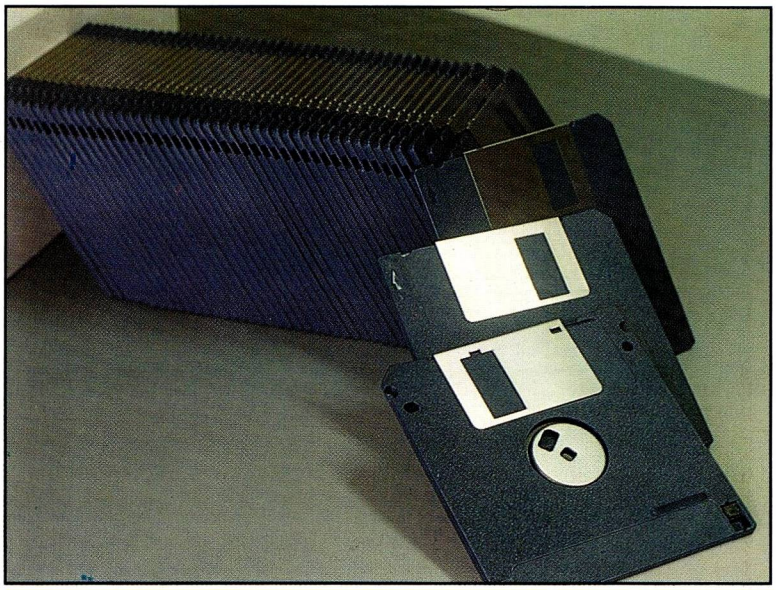


«RATO» COMPATÍVEL MICROSOFT E MOUSE SYSTEM 350-700 DPI – DYNAMIC RESOLUTION – 3 TECLAS

Embalagem com o «rato» e software de instalação
Refª PR01 **Preço: 5 052$00**

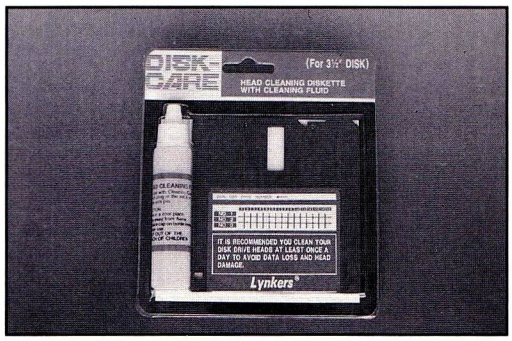
Embalagem com o «rato», software, tapete, caixa e ficha adaptadora 9/25 pinos
Refª PR02 **Preço: 6 485$00**

DISQUETAS BULK (sem marca)



50 DISQUETAS 3,5" DS/DD	**Refª. D040**	**Preço: 5 429$00**
50 DISQUETAS 3,5" DS/HD	**Refª. D041**	**Preço: 10 179$00**

KITS PARA LIMPEZA DE DRIVES

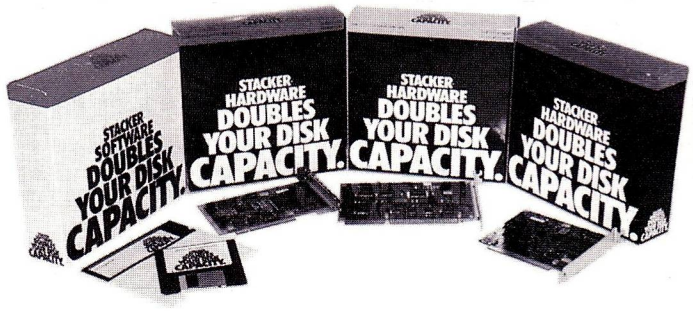


Ref.ª G001
Disqueta 3 ½
602$00

Ref.ª G002
Disqueta 5 ¼
602$00

Kits de limpeza de drives incluindo disquete e frasco com o líquido de limpeza. Qualidade comprovada.

STACKER



STACKER – PARA DUPLICAR A CAPACIDADE DO SEU DISCO E INCREMENTAR O DESEMPENHO DO SISTEMA

O que obtém com o Stacker:
- O dobro da capacidade do seu disco (de qualquer tipo)
- Rápida compressão em tempo real
- «Cache» que aumenta consideravelmente o desempenho do sistema
- 100% de compatibilidade
- Instalação rápida e fácil (basta abrir a caixa do seu PC, colocar a placa **Stacker** numa *slot* livre e «correr» o software de instalação com instruções em português)
- 1 ano de garantia da placa

Conteúdo da embalagem:
- Placa **Stacker**
- Sofware **Stacker** (em disquetas de 3,5" e 5,25")
- Certificado de garantia
- Manual de instruções (em inglês)

Refª ST01	**Só software**	**26 100$00**
Refª ST02	**Placa + Software para processadores XT**	**43 848$00**
Refª ST03	**Placa + Software processadores AT (286/386/486)**	**49 300$00**

CAPA PROTECTORA PARA PCs E IMPRESSORAS



Capa protectora para computadores em plástico especial reforçado para a completa protecção do seu equipamento contra fumos, poeiras, humidade e sujidades de toda a espécie.
Adaptável a qualquer tipo de PC e impressoras.

Ref.: G006 – Capa para PC	**1 674$00**
Ref.: G007 – Capa p/ imp. 80 c.	**913$00**
Ref.: G008 – Capa p/ imp. 132 c.	**913$00**

REEDIÇÃO DOS NÚMEROS ESGOTADOS DA SPOOLER (4, 5 e 6)

Depois de esgotada a reedição dos números 1, 2 e 3 da Spooler e para satisfazer a procura manifestada por inúmeros leitores vamos tornar a imprimir os números 4, 5 e 6, agrupados num só volume.

Esta edição tem uma tiragem reduzida e não será distribuida.

O único modo de possuir um exemplar é reservá-lo, desde já, preenchendo o cupão anexo (ou fotocópia) e remete-lo para os nossos serviços:

SPOOLER MAGAZINE
Rua Duarte Pacheco Pereira,
69-A – Damaia-de Baixo
2700 AMADORA

CUPÃO DE RESERVA DA EDIÇÃO ESPECIAL SPOOLER – NÚMEROS 4, 5 E 6

NOME .....

ENDEREÇO .....

.....

C. POSTAL ..... LOCALIDADE ..... TELEF. .....

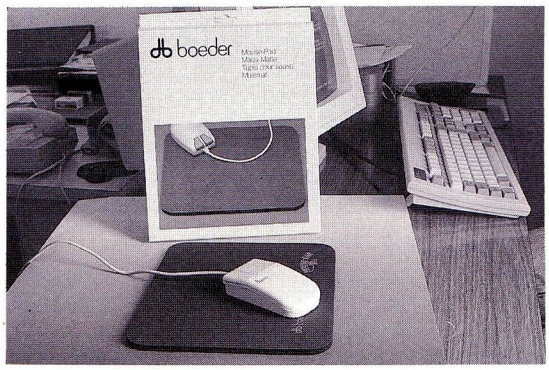
DESEJO RECEBER A EDIÇÃO ESPECIAL «SPOOLER 4 - 5 - 6»

☐ **COM 3 DISQUETAS DE 5 1/4" ..... 1 200$00**
☐ **COM 2 DISQUETAS DE 3 1/2" ..... 1 200$00**

Não mande qualquer pagamento agora. A edição especial ser-lhe-á enviada à cobrança.
Aos preços indicados serão acrescidos os portes de envio e taxa de cobrança dos C.T.T.

Assinatura

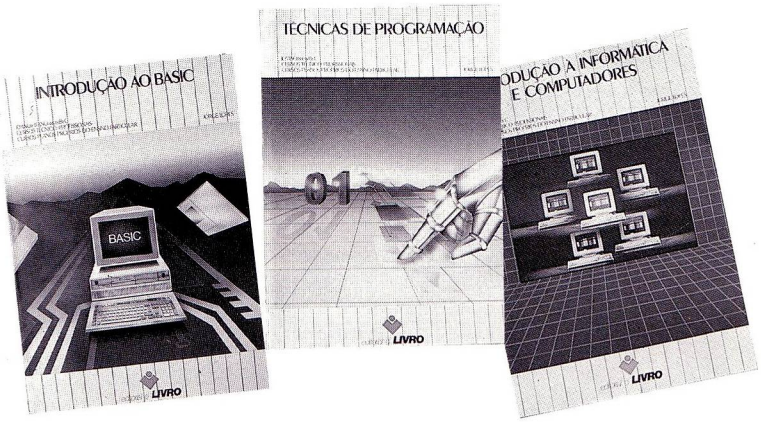
TAPETE PARA MOUSE



Tapete para Mouse de elevada qualidade. Anti-estático. Permite-lhe uma maior precisão e facilidade no seu trabalho.

Ref.ª G005 **PREÇO: 871$00**

LIVROS EDITORIAL O LIVRO (Em português)

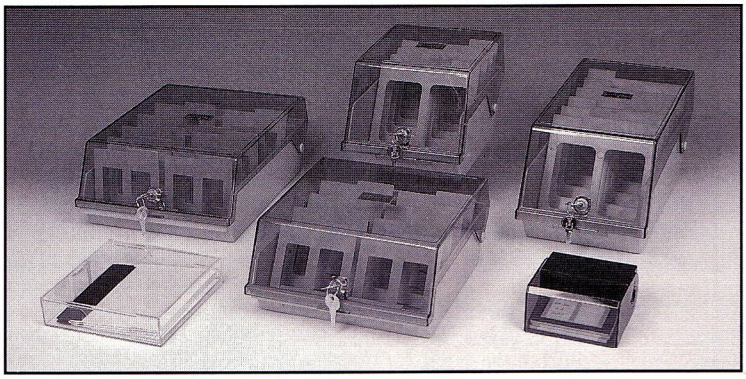


– INTRODUÇÃO À INFORMÁTICA E COMPUTADORES
REF.: L001 **PREÇO: 1 806$00**

– TÉCNICAS DE PROGRAMAÇÃO
REF.: L002 **PREÇO: 1 575$00**

– INTRODUÇÃO AO BASIC
REF.: L003 **PREÇO: 2 352$00**

CAIXAS ARQUIVADORAS PARA DISQUETAS



– 10 disquetas 3,5"	Ref.: H004	Preço:	272$00
– 10 disquetas 5,25"	Ref.: H005	Preço:	344$00
– 50 disquetas 5,25"	Ref.: H006	Preço:	1 131$00
– 100 disquetas 5,25"	Ref.: H007	Preço:	1 328$00
– 50 disquetas. 3,5"	Ref.: H008	Preço:	1 192$00
– 80 disquetas 3,5"	Ref.: H009	Preço:	1 356$00
– 100 disquetas. 3,5"	Ref.: H010	Preço:	1 500$00



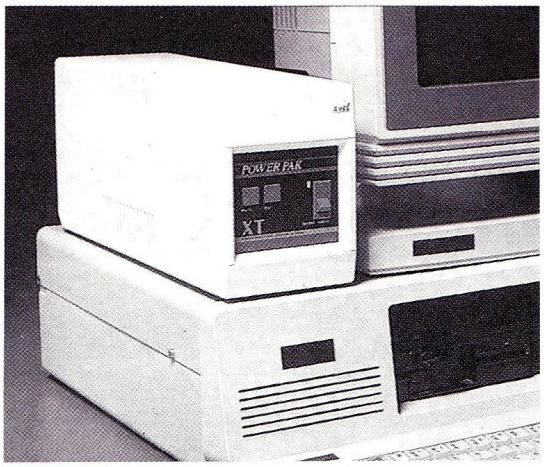
DBase III Plus
PROGRAMAÇÃO AVANÇADA

Refª. GR01 COM DISQUETA DE 5,25" 1 890$00

Refª. GR02 COM DISQUETA DE 3,5" 1 995$00

INVISUAIS: Quando solicitado forneceremos em disqueta suplementar o texto integral do livro.

POWER PAK – AT



Unidade de Estabilização e Manutenção de energia eléctrica, indicada para protecção da configuração completa de PC AT.
Bateria selada, sem manutenção.
Potência máxima: 550 VA – Autonomia: 25 minutos.

GARANTIA DE 1 ANO

Refª. PK01 **PREÇO: 52 003$00**

PAPEL CONTÍNUO

Embalagem de 500 folhas de papel contínuo (zebra americana) 12x9,5"

Refª. P020 **Preço: 785$00**

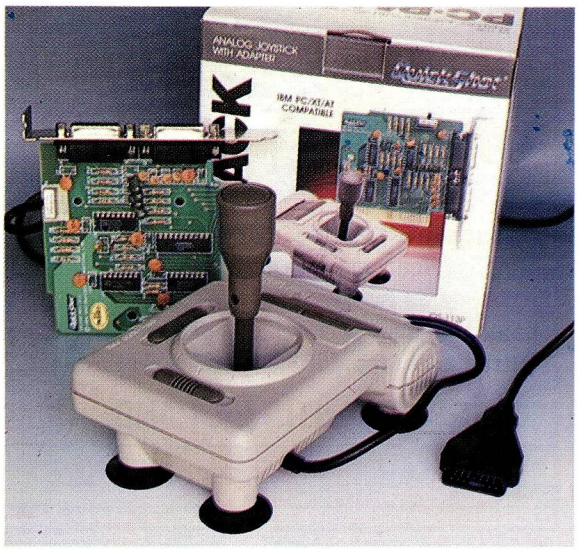
QUICKJOY M-5



Compatível para PC/XT/AT este joystick de *design* ergonómico é o ideal para os seus jogos de acção. A placa pode ser adquirida em separado.

Refª G004 **Preço: 4 289$00**

QUICK SHOT – JOYSTICK + PLACA



Joystick para compatíveis PC/XT/AT e placa. Óptimo desempenho e *design* atractivo. Um novo dinamismo nos seus jogos predilectos. Um preço incrível!

Refª G003 **Preço: 6 825$00**

DISQUETAS PLATINUM



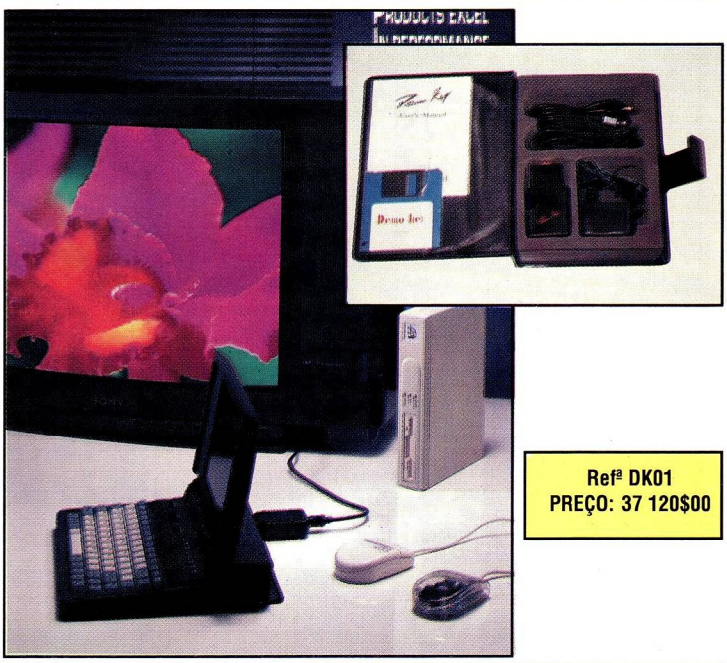
Caixas com 10 disquetes:
– 5,25" DS/DD **Ref.: PL01** **Preço: 905$00**
– 3,5" DS/DD **Ref.: PL02** **Preço: 1 463$00**

PLACA QS-120 PARA JOYSTICK

Placa para joystick

Refª G04A **Preço: 3 861$00**

DEMOKEY (VGA TO VIDEO)



Refª DK01
PREÇO: 37 120$00

Dispositivo que permite transformar em sinal vídeo qualquer saída VGA de computador.
Ideal para os possuidores de portáteis ou *notebook*, ou de monitores VGA monocromáticos, que assim podem visualizar os seus programas confortavelmente no monitor colorido do seu receptor de televisão.
Também indicado para demonstrações e exibições, uma vez que o sinal VGA do computador pode ser «desdobrado» e visualizado em receptores de televisão montados em cadeia. Outra possibilidade é a de permitir registar as imagens do seu computador no seu videogravador.

- Resoluções até 640x480, 256 cores
- Saída vídeo NTSC ou PAL
- Portas de saída AV Vídeo e S-VHS Vídeo

A prática embalagem do Demokey contém:

- O Demokey
- Transformador de alimentação de 9 V
- Cabos de ligação para AV Vídeo e S-VHS
- Manual e disqueta de instalação (em inglês)

COMPUTADOR SPOOLER 386/33

VEJA NA PÁGINA 9
A MÁQUINA
QUE A SPOOLER
ESCOLHEU E PREPAROU
PARA SI

WINDOWS 3.0 (EM PORTUGUÊS)

Não vale a pena possuir cópias piratas. Além da ilegalidade o utilizador está sempre privado dos «upgrades» que vão surgindo e das vantagens que um Manual (neste caso em português) lhe oferece para optimização do seu sistema.

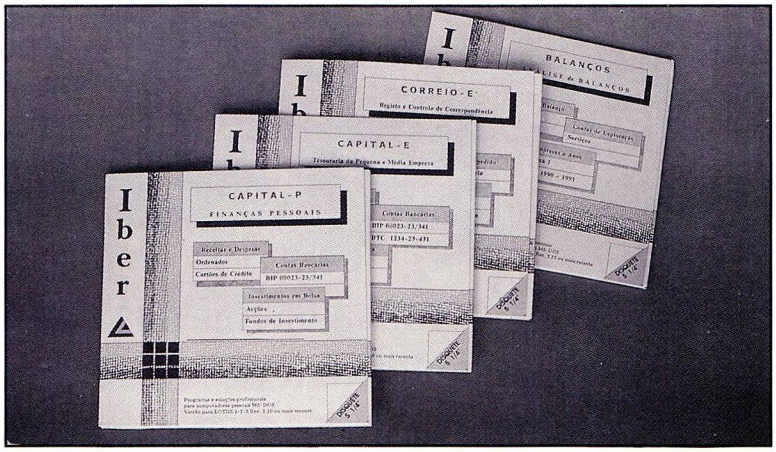
Refª. W001	COM DISQUETAS DE 5,25"	39 324$00
Refª. W002	COM DISQUETAS DE 3,5"	39 324$00

DOS 5.0 (EM PORTUGUÊS)

Conheça as novas e poderosas potencialidades do sistema operativo MS-DOS 5.0. Tudo em português.

Refª. MS01	COM DISQUETAS DE 5,25"	20 300$00
Refª. MS02	COM DISQUETAS DE 3,5"	20 300$00

SOFTWARE IBER



Programas e soluções profissionais para compatíveis MS-DOS. Não requerem conhecimentos de Lotus 1-2-3 ou de MS-DOS.

Refª IB01* CAPITAL-P – FINANÇAS PESSOAIS.

Refª IB02* CAPITAL-E – TESOURARIA DE PEQUENA E MÉDIA EMPRESA.

Refª IB03* CORREIO-E – REGISTO E CONTROLO DE CORRESPONDÊNCIA..

Refª IB04* BALANÇOS – ANÁLISE DE BALANÇOS.

Refª IB05 BIBLOS – GESTÃO DE LIVROS, VIDEOS E DISCOS.

Em disquetas de 3,5" (720 Kb) ou 5,25" (360 Kb)
PREÇO: 3 620$00 (cada)

***Versões para Lotus 1-2-3 – Rev 2.10 ou mais recente**

CUPÃO MONTRA SPOOLER

Recorte ou fotocopie o cupão e remeta-o para:

SPOOLER MAGAZINE
Rua Duarte Pacheco Pereira, 69-A – Damaia
2700 AMADORA

ESTIMADO LEITOR:
Tentamos entregar as suas encomendas o mais rápido possível.
Quando um número da **Spooler** sai ao público, somos «bombardeados» com muitos milhares de pedidos.
Não desespere se não receber a sua encomenda no dia seguinte e **por favor não a devolva.** Os preços por nós praticados (quase isentos de lucro) não permitem o encargo adicional de ter que pagar os portes do retorno da encomenda. Comunique-nos se pretender, à *posteriori*, desistir da encomenda.

CUPÃO DE ENCOMENDA – MONTRA SPOOLER

Cliente nº.:
A PREENCHER PELOS NOSSOS SERVIÇOS

Desejo receber à cobrança (com os **custos acrescidos de portes e taxas**) os seguintes produtos:

Quant.	Refª.	Designação	Preço
		TOTAL	

NOME .....
MORADA .....
C. POSTAL ..... LOCALIDADE .....
TELEFONE ..... ❑ NÃO SOU ASSINANTE ❑ SOU ASSINANTE Nº .....

Spooler 17

ABERTOS AOS SÁBADOS ATÉ ÀS 13 H.

Loja Spooler

Venha visitar-nos. Na Loja Spooler, vai encontrar um pouco de tudo a preços imbatíveis

Só vendemos produtos de qualidade comprovada

Computadores • Impressoras • Power Packs • Modems • Disquetas • Caixas arquivadoras para disquetas • Ratos • Jogos • Joysticks • Papel contínuo • Tapetes para rato, etc., etc., etc.

Rua Duarte Pacheco Pereira, 69-A – Damaia-de-Baixo – 2700 AMADORA
Telefs.: 475 22 89/474 60 11/474 60 41

DESCONTO 10% – ESPECIAL ASSINANTES

ASSINATURAS Spooler

Recorte ou fotocopie o cupão, preencha-o com letra bem legível e remeta-o, acompanhado do respectivo pagamento em cheque ou vale de correio, para:

SPOOLER MAGAZINE
R. Duarte Pacheco Pereira, 69-A
Damaia – 2700 AMADORA

*Os números anteriores da **Spooler** podem ser encomendados à cobrança utilizando o cupão da **Montra Spooler**. Ao preço de capa serão acrescidos os custos dos portes e taxas de cobrança.*
Se preferir disquetas de 3,5", isto custar-lhe-á mais 140$00 por exemplar.

NOTA: Os números 1 a 6 e o 10 e 11 da **Spooler** encontram-se esgotados. A reedição especial dos números 1, 2 e 3, também!

Spooler 17

CUPÃO DE ASSINATURA

Assinante nº:
A PREENCHER PELOS NOSSOS SERVIÇOS

NOME .....
DATA DE NASCIMENTO* _____ / ____ / ____
PROFISSAO* .....
TRABALHA COM COMPUTADORES?* SIM ☐ NÃO ☐
SE SIM ESPECIFIQUE A MARCA E MODELO* .....
ENDEREÇO .....
.....
C. POSTAL ..... LOCALIDADE ..... TELEF(S) .....

DESEJO ASSINAR A SPOOLER A PARTIR DO Nº. 18 (inclusive) E DURANTE 12 NÚMEROS.

☐ **MODALIDADE COM DISQ. DE 5 ¼" ..... 5 500$00**
☐ **MODALIDADE COM DISQ. DE 3 ½" ..... 6 000$00**

PARA PAGAMENTO DA ASSINATURA ENVIO :
☐ O CHEQUE Nº. ..... S/ O BANCO .....
☐ O VALE DE CORREIO Nº. .....

** A resposta aos itens assinalados com um (*) é opcional.*

O valor indicado não paga os direitos destes programas, mas sim, apenas as próprias disquetas e o trabalho de cópia e distribuição. Será o utilizador dos programas tipo «shareware» a ter a responsabilidade de efectuar o respectivo pagamento, directamente aos autores, nos termos e condições por eles propostos. A *Spooler* limita-se a distribuir esses programas, devidamente testados e na sua versão integral.

A86 – Assemblador e desassemblador – *Eric Issacson* (86-87-88)
Documentação em francês e inglês
Refª. SH01 – 1 disqueta de 5,25" ..... 500$00
Refª. SH02 – 1 disqueta de 3,5" ..... 600$00

QModem SST – Versão 4.2E – Utilitário de comunicações
– *The Forbis Project Inc.* – Doc. em inglês
Refª. SH03 – 3 disquetas de 5,25" ..... 1 500$00
Refª. SH04 – 2 disquetas de 3,5" ..... 1 200$00

Thedraw – Versão 3.40 – Utilitário de desenho, animação e tratamento da imagem em modo texto – Doc. em inglês – *The Soft Prog. Serv.*
Refª. SH05 – 1 disqueta de 5,25" ..... 500$00
Refª. SH06 – 1 disqueta de 3,5" ..... 600$00

Pkzip – Versão 1.10 – Compactador e descompactador de ficheiros
– *Pkware Inc.* – Doc. em inglês.
Lha – Versão 2.11 – Compactador e descompactador de ficheiros
– *Haruyasu Yoshizaki* – Doc. em inglês.
Refª. SH07 – 1 disqueta de 5,25" ..... 500$00
Refª. SH08 – 1 disqueta de 3,5" ..... 600$00

List – Versão 7.5 E– O famoso *List* com respectivos utilitários
– *Vernon D. Buerg* – Doc. em inglês.
VPIC – Versão 4.2 – Leitor de Gifs – *Bob Montgomery* – Doc. em inglês
Picem – Versão 2.1 – Leitor de Gifs – *John Bridges* – Doc. em inglês
Refª. SH09 – 1 disqueta de 5,25" ..... 500$00
Refª. SH10 – 1 disqueta de 3,5" ..... 600$00

Boxer – Versão 3.03 – Editor de texto muito completo e versátil
– *David R. Hamel* – Doc. em inglês.
Refª. SH11 – 1 disqueta de 5,25" ..... 500$00
Refª. SH12 – 1 disqueta de 3,5" ..... 600$00

Utilitários para DOS – Selecção de diversos utilitários para DOS.
(Manipulação ecrã, rotinas para impressora, caracteres medievais para VGA, utilitários para *Batchs*, gestores de memória, etc...)
Diversos autores – *Doc. em inglês.*
Refª. SH13 – 1 disqueta de 5,25" ..... 500$00
Refª. SH14 – 1 disqueta de 3,5" ..... 600$00

Utilitários para DBase – Selecção de diversos utilitários para DBase.
(drivers para impressora, etiquetas, exemplos de base de dados relacional, etc....) – *Diversos autores* – Doc. em inglês.
Refª. SH15 – 1 disqueta de 5,25" ..... 500$00
Refª. SH16 – 1 disqueta de 3,5" ..... 600$00

Utilitários para Clipper – Selecção de diversos utilitários para Clipper
(Janelas para Clipper, rotinas para o rato, escudos por extenso, dicas e truques, editor de campo memo, animação, etc...)
Diversos autores – Doc. em inglês.
Refª. SH17 – 1 disqueta de 5,25" ..... 500$00
Refª. SH18 – 1 disqueta de 3,5" ..... 600$00

Utilitários para Windows 3.0 – Diversos utilitários para Windows 3.0
(Almanaque, Ascii, Biorritmo, File Manager (aceita zips), etc...)
Diversos autores – Doc. em inglês.
Refª. SH19 – 1 disqueta de 5,25" ..... 500$00
Refª. SH20 – 1 disqueta de 3,5" ..... 600$00

CD-MAN – Jogo estilo «Pac-man» com um grafismo fora de série.
– *Creative Dimensions* – Doc. em inglês – EGA / VGA
Refª. SH21 – 1 disqueta de 5,25" ..... 500$00
Refª. SH22 – 1 disqueta de 3,5" ..... 600$00

Dark_ages – Jogo de arcada/aventuras. *Apogée Software Productions*
– Doc. em inglês. – EGA / VGA.
Refª. SH23 – 1 disqueta de 5,25" ..... 500$00
Refª. SH24 – 1 disqueta de 3,5" ..... 600$00

Vampyr – Jogo de aventuras. Género medieval/fantástico. .....
Brian Weston/Victor Shao – Doc. em inglês – EGA / VGA.
Refª. SH25 – 1 disqueta de 5,25" ..... 500$00
Refª. SH26 – 1 disqueta de 3,5" ..... 600$00

GIFS – Imagens digitalizadas
(Para visionar estas imagens necessita de um editor de GIFs. Disponível nas referências SH09 e SH10).

– Imagens de desenhos animados
Refª. SH27 – 1 disqueta de 5,25" ..... 500$00
Refª. SH28 – 1 disqueta de 3,5" ..... 600$00

– Imagens do mundo animal
Refª. SH29 – 1 disqueta de 5,25" ..... 500$00
Refª. SH30 – 1 disqueta de 3,5" ..... 600$00

– Garotas. Belas imagens
Refª. SH31 – 1 disqueta de 5,25" ..... 500$00
Refª. SH32 – 1 disqueta de 3,5" ..... 600$00

– Paisagens
Refª. SH33 – 1 disqueta de 5,25" ..... 500$00
Refª. SH34 – 1 disqueta de 3,5" ..... 600$00

– Imagens de filmes
Refª. SH35 – 1 disqueta de 5,25" ..... 500$00
Refª. SH36 – 1 disqueta de 3,5" ..... 600$00

– Diversos. Raquel Welch, cão punk, carro antigo, etc.
Refª. SH37 – 1 disqueta de 5,25" ..... 500$00
Refª. SH38 – 1 disqueta de 3,5" ..... 600$00

– Paisagens diversas, um quadro «à Van Gog», S.Francisco, Chicago, a Guerra das Estrelas, uma águia e o pato Dodger.
Refª. SH55 – 1 disqueta de 5,25" ..... 500$00
Refª. SH56 – 1 disqueta de 3,5" ..... 600$00

Utilitários para DOS – COPYIT: um copy muito melhorado, com pedido de mais disquetas, se necessário. **LO**: temporizador de teclado com envio de dados para o ecrã. **XEQ**: agrupa executáveis num único ficheiro, podendo chamar-se qualquer um isoladamente. **CHANGE**: muda atributos, data e hora. **YAY**: utilitário com editor, calendário, tabela ASCII, pesquisa de palavras e SHELL. **RUN**: corre programas a partir de outra drive e/ou directório. **1PASS**: diskcopy melhorado.
Refª. SH39 – 1 disqueta de 5,25" ..... 500$00
Refª. SH40 – 1 disqueta de 3,5" ..... 600$00

Utilitários para Clipper – CLIPCALC: simulação de uma calculadora. **COPYPROT**: proteção contra cópias. **NANFORLIB**: diversas bibliotecas para Clipper. **SB**: utilitário para execução de menus, caixas, textos, colorir, etc.
Refª. SH41 – 1 disqueta de 5,25" ..... 500$00
Refª. SH42 – 1 disqueta de 3,5" ..... 600$00

Aplicações para Windows – DBFBROWSER: visualizaçãoe tratat. de Bases de Dados. **DICE**: jogo de dados. **ICONDRAW**: para desenhar *icons*. **FIREDRAW**: simulação de fogo de artifício, **FISH3**: simulação de aquário, **HUNTER**: pesquisa de ficheiros em qualquer drive. **WINEZ**: cria nova janela de comando para execução de aplicações.
Refª. SH43 – 1 disqueta de 5,25" ..... 500$00
Refª. SH44 – 1 disqueta de 3,5" ..... 600$00

Utilitários para C – CBD: ferramentas para desenvolvimento de bases de dados para programadores de C, compatível com Microsoft C e Turbo C. **CLOCK**: rotina para inclusão de um relógio digital TSR. **DBLOOK**: visualização de .DBF's do dBASE. **EXEC**: fontes para a função «EXEC» que permite chamar programas externos, opcionalmente trocando a memória de imagem para EMS, XMS ou para um ficheiro. Compatível com Turbo C (V. 1.x, 2.x e C++ 1.0), Borland C V. 2.0, Microsoft (V. 5.1 e 6.0), Watcom (V. 8.0), Turbo Pascal (V. 4.x e 5.x). **MOUSEC**: rotinas para rato em Microsoft C ou Turbo C. **PROTOC**: execução de protótipos de funções. **TRAVDIRS**: executa um comando específico em qualquer directório.
Refª. SH45 – 1 disqueta de 5,25" ..... 500$00
Refª. SH46 – 1 disqueta de 3,5" ..... 600$00

Turbo Toolkit V.5.0 - Fontes e Manual. Construção de janelas, menus, entradas do utilizador, formação de cadeias de caracteres, listagem de directórios, etc. Compatível com Turbo Pascal V. 4.0 e 5.0.
Refª. SH47 – 1 disqueta de 5,25" ..... 500$00
Refª. SH48 – 1 disqueta de 3,5" ..... 600$00

Keen - Jogo para EGA e VGA.
Refª. SH49 – 1 disqueta de 5,25" ..... 500$00
Refª. SH50 – 1 disqueta de 3,5" ..... 600$00

Kroz e Quest - Jogos (todas as placas)
Refª. SH51 – 1 disqueta de 5,25" ..... 500$00
Refª. SH52 – 1 disqueta de 3,5" ..... 600$00

Paga1 - Jogo (todas as placas)
Refª. SH53 – 1 disqueta de 5,25" ..... 500$00
Refª. SH54 – 1 disqueta de 3,5" ..... 600$00

Utilitários Diversos: Diversos autores. Documentação em inglês.
QFIND: Pesquisa de qualquer ficheiro no sistema. Inclui utilitário para Windows 3.0
CLEAN84 e **SCANV84**: As versões V 7.9V84 do famoso «Viruscan» da McAfee.
VCOPY82: Um DOS COPY com detecção automática de vírus V 0.7V82 da McAfee.
KEYLOCK: Um eficiente cadeado para o teclado.
PUTPASS: Um *password* para aceder a ficheiros .COM.
PULLDWN: Um cómodo e eficiente «menu» para gerir aplicações
Refª. SH57 – 1 disqueta de 5,25" ..... 500$00
Refª. SH58 – 1 disqueta de 3,5" ..... 600$00

Utilitários para Windows 3.0
PCASE – CAD para Windows: Versão de demonstração.
UNICOM: Programa de comunicações.
WUNZIP: «Zipar» e «Dezipar» dentro do Windows.
ZIPAPER: «Wall-paper» comprimido pronto a utilizar.
Refª. SH59 – 1 disqueta de 5,25" ..... 500$00
Refª. SH60 – 1 disqueta de 3,5" ..... 600$00

Utilitários e jogos para Windows 3.0
CUBIC: O famoso jogo dos cubos.
NEWICONS: Biblioteca de ícones.
WINFRA20: Visualização de fractais.
XVTDRAW: Utilitário para desenho de precisão.
Refª. SH61 – 1 disqueta de 5,25" ..... 500$00
Refª. SH62 – 1 disqueta de 3,5" ..... 600$00

Utilitários diversos
EMSTOOL2: O famoso «ToolKit» da Intel para programadores de C, *Assembly* e Pascal.
HRAM1: Um gestor para memória alta.
DOSMAP2: Apresenta o mapa da memória.
Refª. SH63 – 1 disqueta de 5,25" ..... 500$00
Refª. SH64 – 1 disqueta de 3,5" ..... 600$00

Utilitários diversos
VGACOPY: Copiador de disquetas em VGA
GIFDOT: Imprime GIF's em vários formatos
CODEP: Converte codepage 850 em 860 e vice-versa
DRVINST: Instalador de Device-drivers
HELPSPEC: Conjunto residente de caracteres especiais
PCCALC: Calculadora TSR
BREAKIUP: Parte ficheiros grandes em vários pequenos
SILENCE: Silencia o altifalante
VGADISK: Instala um disco virtual a partir da memória do VGA
Refª. SH65 – 1 disqueta de 5,25" ..... 500$00
Refª. SH66 – 1 disqueta de 3,5" ..... 600$00

V860V321
O famoso *package* de José Páscoa, destinado à adequada instalação do *code page* português, poupando precioso espaço na memória convencional e substituindo as obsoletas instruções do MS-DOS. Inclui ainda as seguintes capacidades:
– Acesso a todas as teclas do teclado AT extendido a nível do DOS
– Clique do teclado configurável
– Redefinição do *typematic rate* e *delay rate* do teclado
– Possibilidade de desinstalação do programa da memória
– Alarme musical configurável

Tem mais os seguintes utilitários:
MODO.EXE: Altera o modo gráfico utilizando os serviços do DOS
GEN.EXE: Gera e captura fontes para utilização com FONTES.EXE
PLACA.EXE: Identifica a placa gráfica
Refª. SH67 – 1 disqueta de 5,25" ..... 500$00
Refª. SH68 – 1 diqueta de 3,5" ..... 600$00

A/B – HIPERTEXTO
BlackMagic: Processador em «Hypertext», versão 1.2. O sistema chamado de «hipertexto» permite expandir textos ligando palavras-chave e blocos de informação a outros textos.
Refª. SH69 – 1 disqueta de 5,25" ..... 500$00
Refª. SH70 – 1 disqueta de 3,5" ..... 600$00

Utilitários
SHEZ: Versão 6.3 do famoso «Shell» altamente especializado para manipulação de compressões. Corre em Windows 3.0 e suporta o rato.
COLORIT: Utilitário para colorir textos ASCII.
PKLITE: Compressor de ficheiros .EXE e .COM mantendo-os executáveis.
HDM4: Gestor do disco rígido com *password*.
Refª. SH71 – 1 disqueta de 5,25" ..... 500$00
Refª. SH72 – 1 disqueta de 3,5" ..... 600$00

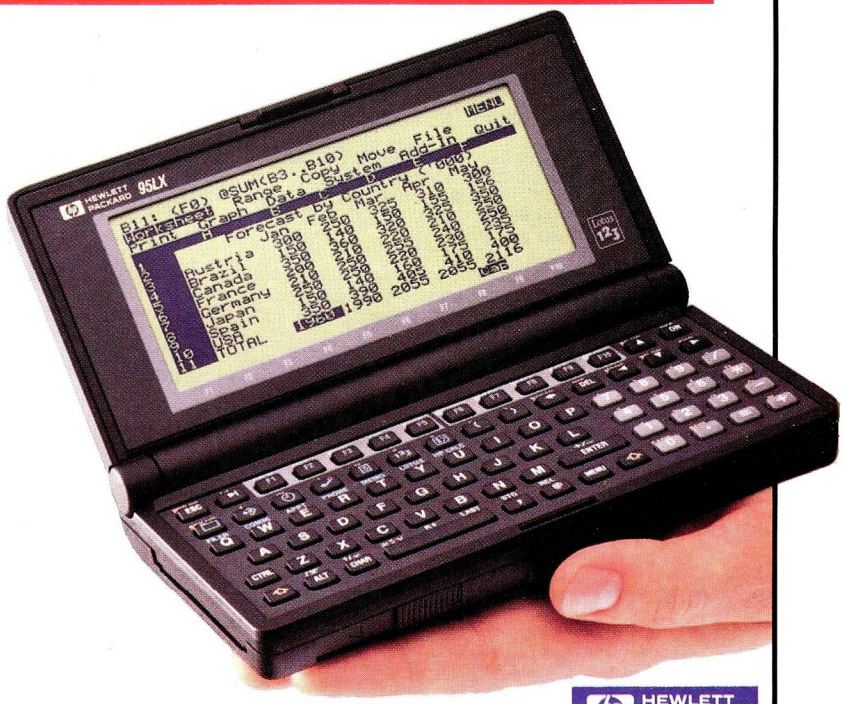
Utilitários e jogos para Windows 3.0
DISKLOOK: Informação permanente sobre os seus discos.
KLZ209: O Tetris para Windows 3.0
MAGIC110: Um *screen-saver* com *password* e diversas configurações
BG06: O jogo do Gamão para Windows 3.0
PUZZLE: Um *puzzle* para Windows 3.0
WINWEEL: Roda da Sorte para Windows 3.0
IEDIT: Editor de ícones para Windows 3.0
WIN3CLOC: Um relógio configurável para Windows 3.0
CHANGCUR: Altere o cursor do rato em Windows 3.0
Refª. SH73 – 1 disqueta de 5,25" ..... 500$00
Refª. SH74 – 1 disqueta de 3,5" ..... 600$00

Utilitários para DOS
MATCH201: Compara documentos linha a linha. Útil para programadores.
AGENDAP: Calendário residente desde 1901 até 2089.
CNFMT16: Formata disquetas enquanto se executa outra tarefa.
ARJ221: A versão V2.21 do célebre compressor.
SHERIFF: Um administrador de segurança para o sistema.
EEMRAM21: Aumenta a memória disponível até 736K com recurso à LIM.
Refª. SH75 – 1 disqueta de 5,25" ..... 500$00
Refª. SH76 – 1 disqueta de 3,5" ..... 600$00

O SEU PC DE BOLSO HP 95LX

Gráfcria pub

– 1 Mb de ROM
– *Software*:
  – MS DOS®3.22
  – Lotus® 1-2-3®, versão 2.2
  – Comunicação de dados
  – Editor de texto
  – Gestão de ficheiros
  – Calculadora H. P.
  – Agenda
  – Lista telefónica
– 512 Kb de RAM expansíveis até 1 Mb
– Ligação a PC, impressora, *modem*, etc.



Contacte:

DIMECRO
Representações Comerciais e Técnicas, Lda.
Rua Joaquim António de Aguiar, 41 - 6º Esqº – 1000 LISBOA
Telefones: 315 76 12/3/4/5 – Fax: 315 76 12

HEWLETT PACKARD
Distribuidor

THE COMPLETE COLOUR SOLUTION

73 500$ + IVA

«The Complete Colour Solution» proporciona-lhe tirar todo o partido da sua placa VGA!
Você pode capturar imagens a cores de alta resolução a partir do seu vídeogravador caseiro, camâra ou qualquer outra fonte de sinal vídeo.
As imagens (não retocadas) que ilustram esta página foram digitalizadas a 256 cores. São um pequeno exemplo do que você pode obter em apenas alguns segundos.

DÊ LARGAS À SUA IMAGINAÇÃO E CRIATIVIDADE



CONTEÚDO DA EMBALAGEM
- Vidi-PC (Carta para a captura e digitalização de imagens)
- VidiChrome-PC (software para tratamento da cor em alta resolução)
- Vidi-RGB (separador electrónico de cores)
- PC Paintbrush IV Plus (provavelmente o melhor software do mundo para arte e grafismos informáticos)
- Picture Box (Conjunto de demonstração)

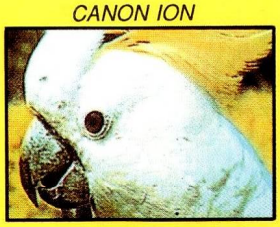
SOFTWARE:
Disponível em disquetes de 3,5" ou 5,25"

CARACTERÍSTICAS
Resoluções de imagem possíveis:
- 320x200/256 cores
- 640x400/256 cores
- 640x480/256 cores
- 800x600/256 cores
- Suporta todas as resoluções de 256 cores até 800x600 (EVGA)
- Captura até 4096 cores em menos de 1 segundo
- Avançado algoritmo para selecção de paletes
- Zoom interactivo
- Converte imagens negativas em positivas
- Imprime em todas as impressoras *standadrd* mono ou policromáticas
- Grava as imagens nos formatos Gem, PCX ou Tiff.

APLICAÇÕES:
- Apresentação de produtos ou serviços
- Demonstração e vendas
- DTP a cores (edição computorizada)
- Ensino e formação profissional
- Animação
- Concepção de catálogos e literatura diversa

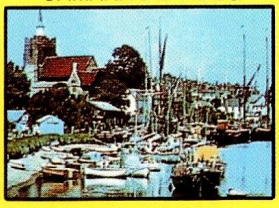


CANON ION





CAMÂRA DE VÍDEO



VIDEOGRAVADOR



Para obter estes resultados, apenas necessita possuir os seguintes equipamentos:
- Computador compatível AT ou XT
- 640 Kb de RAM
- Monitor VGA policromático
- Disco rígido
- Rato
- Uma fonte de vídeo (câmara, videogravador, etc.)

Game® informática

DISTRIBUÍDOR EM PORTUGAL DA ROMBO Limited

Loja 1 – R. Elias Garcia, 362-D – C. C. Babilónia, Loja 70 – 2700 AMADORA – Tel. 492 14 32
Loja 2 – Departamento Profissional – Rua D. Pedro IV, Lote 13, Loja C – 2745 QUELUZ
Telef.: 436 15 38 – Tel./Fax: 436 10 08

Grafiteria pub adapt.

Graficria PUB

Consuldata
SERVIÇOS DE INFORMATICA, LDA.

UM MUNDO EM CONSUMÍVEIS PARA INFORMÁTICA

AS MELHORES MARCAS
O MELHOR ATENDIMENTO
A ENTREGA MAIS RÁPIDA
OS MELHORES PREÇOS!!!

DISQUETES DE 3", 3,5", 5,25" E 8"

DATA CARTRIDGES (STREAMTAPES)
DE 10, 20, 45, 60 E 150 MEGABYTES

DISK CARTRIDGES E DISK PACKS
PARA GRANDES SISTEMAS

PASTAS PARA ARQUIVO DE FORMULÁRIOS
DE 9,5x12, 12x9,5, 12x15,5 E 15,5x12
RESPECTIVOS SEPARADORES

FILTROS ANTI-MAGNÉTICOS E ANTI-ESTÁTICOS
PARA TODOS OS TIPOS DE ECRÃS DE COMPUTADOR

MOBILIÁRIO PRÓPRIO
PARA CENTROS DE INFORMÁTICA

TAPE CARTRIDGES PARA IBM 34800
INCLUINDO CARROS DE TRANSPORTE
E MÓVEIS ESPECIAIS DE ARQUIVO

E MUITO MAIS!
PARA MAIS INFORMAÇÕES CONTACTE-NOS
OU SOLICITE O CATÁLOGO CONSULDATA

Consuldata Serviços de Informática, Lda.
Rua Quinta do Almargem, 7 A/B - 1300 LISBOA
Telef.: 364 98 00 Fax: 364 91 08