Muitos são os microcomputadores que actualmente usam o sistema operativo CP/M. Este guia foi concebido para principiantes, mas é tão completo quanto possível. Está dividido em secções que cobrem todos os comandos CP/M (DIR, STAT, etc.) e os utilitários como ED, ASM e DDT. A exposição é acompanhada de exemplos que demonstram a aplicação prática. Um capítulo é dedicado ao MBASIC, a linguagem mais comum para uso em sistemas CP/M. Outro capítulo tem por tema o uso de compiladores em CP/M. Os últimos capítulos descrevem as diferentes versões do CP/M. Em apêndices finais são dados uma útil tabela dos códigos ASCII, os comandos de edição de consola CP/M e também os *packages* CP/M mais comuns.

COLECÇÃO SISTEMAS

1. A INFORMÁTICA NA ESCOLA, Manual de Utilização do ZX Spectrum (e tc 2068), *Luís de Campos*
2. GUIA DOS MICROPROCESSADORES, *F. A. Pan*
3. INICIAÇÃO À BASE DE DADOS, *François Fargette*
4. PROGRAMAÇÃO DE COMPUTADORES EM PASCAL, *David Lightfoot*
5. OS SISTEMAS OPERATIVOS, *A. M. Lister*
6. O SISTEMA OPERATIVO DO SPECTRUM ROM DISASSEMBLY, *Ian Logan e Frank O'Hara*
7. PROGRAMAÇÃO DE COMPUTADORES EM COBOL, *Melinda Fisher*
8. PROGRAMAÇÃO DE COMPUTADORES EM BASIC, *L. R. Carter e E. Huzan*
9. PROGRAMAÇÃO EM LINGUAGEM C, *R. E. Berry e B. A. E. Meekings*
10. PROGRAMAÇÃO EM BASIC MSX, *L. R. Carter e E. Huzan*
11. PROGRAMAÇÃO EM dBASE II, *Irene Cohen*
12. PROGRAMAÇÃO EM FORTH, *Steve Oakey*
13. GRAFISMOS A 3 DIMENSÕES, *Michel Rousselet*
14. PROGRAMAÇÃO AVANÇADA EM BASIC, *Augustus J. Quillinan*
15. PROGRAMAÇÃO EM dBASE III, *Carlos Reis*
16. GUIA PRÁTICO DO UNIX, *D. Budgen*
17. PROGRAMAÇÃO EM LOGO, *Martin Lesser*
18. OS MICROPROCESSADORES DE 16 BIT, *Trevor Raven*
19. LINGUAGENS DE MICROCOMPUTADOR, Basic, Pascal, Logo, Comal, Prolog, Forth, *Organização de Mike Duck*
20. COMPILADORES, Sua Concepção e Programação em Pascal, *Robin Hunter*
21. O SISTEMA OPERATIVO CP/M, *Peter Gosling*

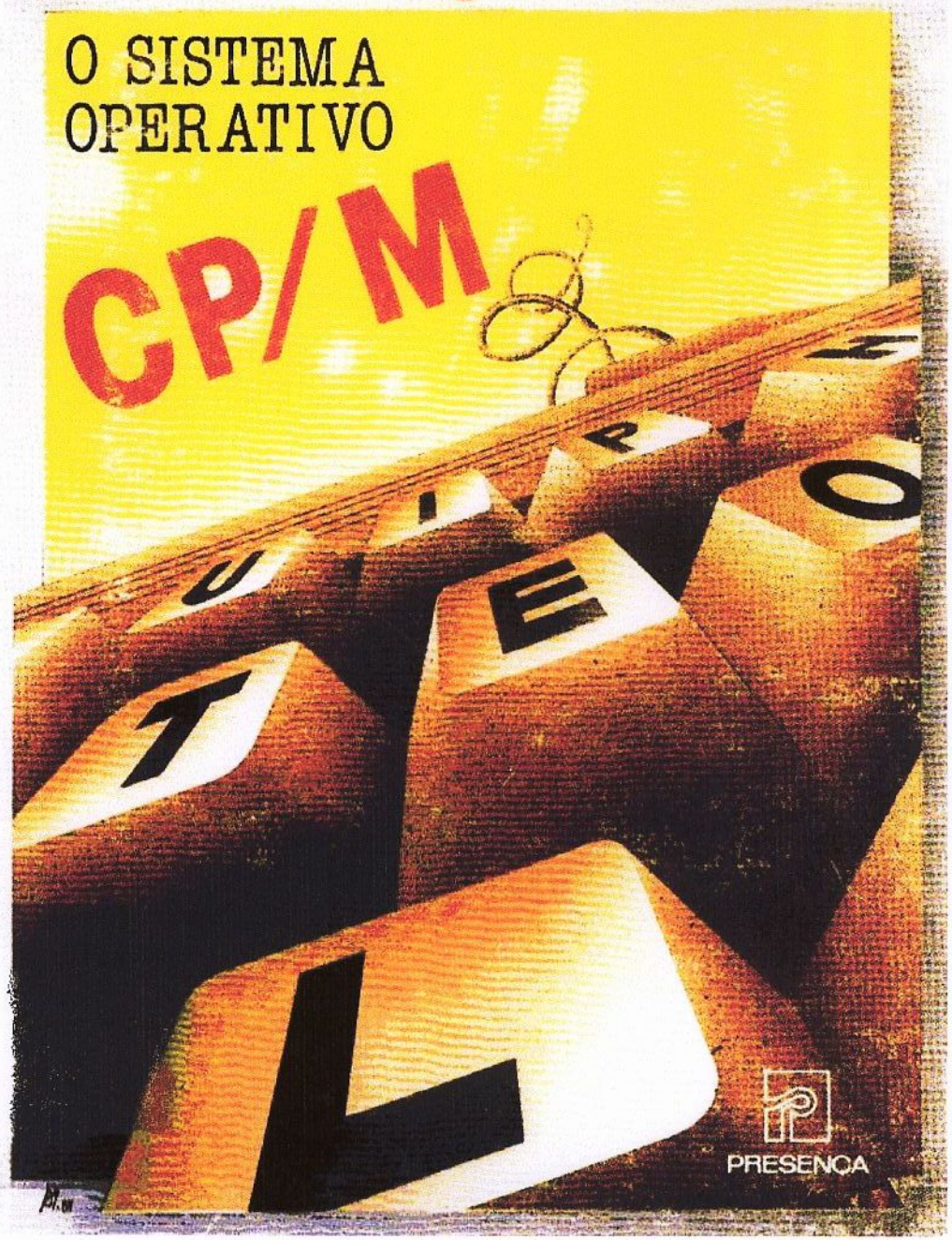
EDITORIAL PRESENÇA

21
Peter Gosling
O SISTEMA OPERATIVO CP/M

Peter Gosling

COLECÇÃO SISTEMAS

O SISTEMA OPERATIVO CP/M



PRESENÇA

COLECÇÃO SISTEMAS

O SISTEMA OPERATIVO CP/M

Peter Gosling

O SISTEMA OPERATIVO CP/M

EDITORIAL PRESENÇA

FICHA TÉCNICA

Título original: *Using CP/M*
Autor: *Peter Gosling*

Publicado originalmente por MacMillan Publishers, Ltd., England
Tradução © *Editorial Presença, Lda., Lisboa, 1987*
Tradução de: *Eduardo Nogueira*
Capa: *Ilustração de António Marques*
Fotocomposição: *Textype — Artes Gráficas, Lda.*
Montagem, Impressão e Acabamento: *Rolo & Filhos - Artes Gráficas, Lda. - MAFRA*
Tiragem: *3000 exemplares*
1.ª edição, Lisboa, 1987
Depósito Legal n.º 16922/87

PREFÁCIO

Este livro foi pensado para dar aos principiantes no sistema operativo CP/M uma ideia das suas capacidades e da forma de o utilizar. São tantos os microcomputadores que actualmente usam este sistema operativo que se torna necessário um guia simples, mas tão completo quanto possível, dos muitos comandos e funções nele contidos. Os manuais fornecidos com os sistemas computadores são notoriamente difíceis de seguir, mesmo por especialistas e muito mais pelos principiantes. Este livro pretende ser tão geral e não específico quanto possível, dado que existem muitos sistemas microcomputadores no mercado que consistem em extras acrescentados ao núcleo básico do CP/M. Foi escrito para tratar precisamente este núcleo comum.

O autor gostaria de agradecer em particular à Digital Research (UK) Ltd. a grande quantidade de úteis informações por ela fornecidas, assim como a autorização para a reprodução de certos diagramas no texto. Recebemos também o auxílio de Midland Micro Services e de Burghley Computers de Stamford, e do Stamford College for Further Education.

Peter Gosling

1

O QUE É O CP/M?

CP/M significa «Programa de Controlo para Microcomputadores» («Control Program for Microcomputers») ou «Programa de Controlo e Monitor» («Control Program and Monitor»); ninguém hoje parece saber exactamente qual a versão correcta. Trata-se de um sistema operativo para a gama de microcomputadores que utilizam um Zilog Z80 — ou outro processador semelhante — como unidade de processamento central.

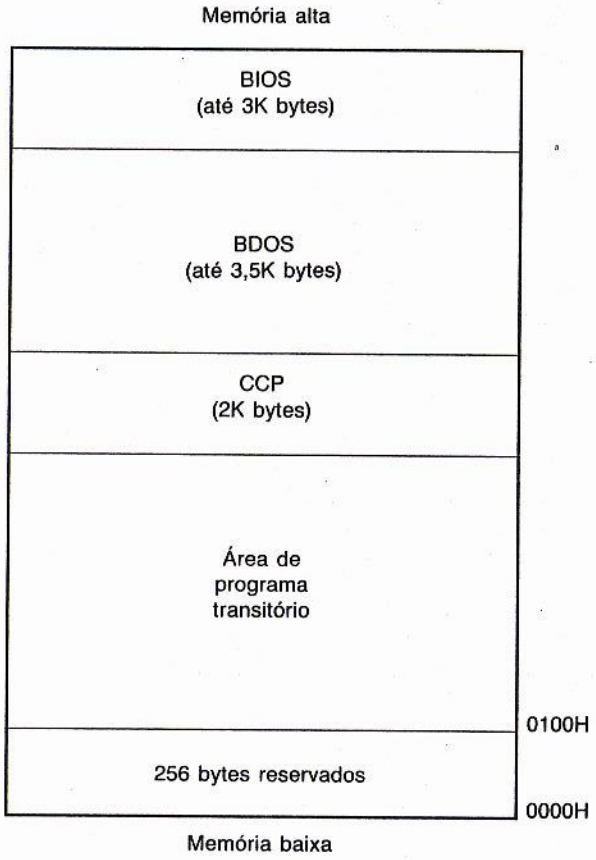
O primeiro sistema operativo CP/M viu a luz do dia em inícios da década de 1970 e foi escrito pelo Dr. Gary Kildall, um dos cientistas ligados ao desenvolvimento inicial de software para o microprocessador Intel 8080.

À medida que surgiu no mercado uma multiplicidade de unidades de disco (todos de diferentes dimensões e usando formatos também diferentes), tornou-se rapidamente evidente que não haveria qualquer futuro em conceber um sistema operativo diferente para cada conjunto de periféricos ligados ao computador. Isto deu origem ao conceito que dá ao CP/M a sua grande flexibilidade e portabilidade. O sistema operativo consiste em três partes, designadas por BIOS («Sistema Básico de Entradas/Saídas»), o BDOS («Sistema Operativo Básico de Disco») e o CCP «Processador de Comando de Consola».

O BIOS é a única parte do CP/M que depende do hardware utilizado. É a parte não coberta legalmente do sistema operativo, e pode ser codificado pelo fabricante de hardware de modo a ter em conta o arranjo especial de drives de disco, terminais e impressoras que comercializa. A Digital Research Inc. fornece uma listagem-fonte de código BIOS.

O BDOS é uma parte fixa, invariante e protegida legalmente do sistema operativo, que controla todos os drives de disco. É independente do hardware que está a ser usado.

O CCP é o software, também invariante e protegido legalmente, que traduz os comandos dados pelo utilizador através do teclado da consola para a codificação que o microprocessador pode executar.



Quando se carrega o CP/M em memória, este é guardado em segmentos de tal modo que o BIOS ocupa a parte superior. O BDOS fica imediatamente abaixo e o CCP na parte inferior. O resto da memória é deixado para os programas de aplicação e é designado por TPA, «Transit Program Area» ou área de programa transitória. É a parte da memória que pode ser endereçada pelo programa em execução. A quantidade de memória usada pelo BIOS depende da configuração de hardware e será tipicamente de cerca de 3K bytes. O BDOS, que tem sempre uma dimensão constante, ocupa 3,5K bytes. Isto deve-se ao facto de, qualquer que seja o formato de disco em uso, o CP/M tratar sempre a leitura e escrita do disco em blocos de 128 bytes até dezasseis drives de disco. A transferência real de dados entre disco e RAM é tratada pelo BIOS.

Por baixo do BIOS e do BDOS encontra-se o CCP e o TPA, e exactamente na parte mais baixa da memória encontra-se um bloco de 256 bytes que contém os parâmetros do sistema. É através destes que o programa em execução comunica com os utilitários CP/M residentes. Isto significa que quando é iniciada uma operação de escrita em disco pelo programa de aplicação, esta actividade é de facto entregue ao BDOS e por este ao BIOS, o qual executará a transferência real dos dados. A propósito, como 256 correspnde a 100 em hexadecimal, é este o endereço inferior do TPA e portanto o endereço inicial de muitos programas. Mostramos na figura um típico mapa de memória de CP/M.

Parte do CP/M, como se pode ver no mapa de memória, está sempre «residente» na memória. Existem outras partes «transitórias» do sistema operativo que são carregadas de disco em função das necessidades. Esses comandos serão descritos no capítulo 3.

Quando ligamos o microcomputador, a primeira operação a realizar é a carga do CP/M para a memória da máquina. Chama-se a isto «bootstrapping» ou «booting» do sistema. Por vezes isto é iniciado carregando em certas teclas da consola, outras vezes ocorre automaticamente assim que se insere o disco de sistema e se fecha a portinhola do drive. Sabemos que a carga terminou quando aparece no visor algum tipo de «anúncio» do fabricante, seguido pelo familiar pedido de entradas do sistema CP/M:

```
A>
```

Este símbolo indica-nos que o CCP está em funcionamento, preparado para responder a uma linha de comando.

Se por alguma razão se torna necessário reiniciar o sistema CP/M de pois de ter ligado o sistema, pode-se executar um «warm boot», uma reentrada «a quente» no sistema. Isto é muitas vezes necessário depois de ter ocorrido um erro e de este erro ter sido indicado por uma das mensagens produzidas pelo CP/M. O arranque a quente é geralmente efectuado carregando em CONTROL-C (ou seja, nas teclas CONTROL e C simultaneamente). O BDOS e o CCP são assim recarregados em memória, passando o computador ao modo de aceitação de comandos.

Uma mensagem de erro muito comum em CP/M é

```
Drive=nn,Track=nn,Sector=nn:Error=nn
Bdos error on X:bad sector
```

onde além de se indicar o número do drive, da pista, do sector e do erro se indica um problema de hardware de algum tipo. CONTROL-C realiza um novo «boot» do sistema, permitindo ao utilizador repetir a operação antes efectuada.

Se carregamos em RETURN, o erro é ignorado e voltamos ao CCP. Tanto o número do drive como os de pista, sector e erro são indicados em hexadecimal.

Todos os códigos de erro prefixados por 1 indicam que não é possível encontrar um sector. Os códigos prefixados por 2 indicam que não se conseguem lê-lo. O prefixo 3 diz-nos que há um erro na verificação do que se encontra no sector. Finalmente, um código prefixado por 4 indica um erro de formatação.

Uma outra mensagem é:

```
Bdos err on X:R/O
```

que nos diz que o drive especificado está configurado apenas para leitura, não aceitando qualque escrita. O CONTROL-C permitir-nos-á igualmente voltar ao CCP.

2

FORMATAÇÃO DE UM DISCO

Antes de usar qualquer disco novo num sistema CP/M, ou aliás em qualquer outro sistema computador, é necessário «formatá-lo». A formatação dos discos é a operação que os prepara para receber dados, sendo bastante semelhante à representação de linhas numa folha de papel para separar colunas de valores num livro de contabilidade, por exemplo. Como todos os sistemas computadores parecem ter modos diferentes de preparar os discos que usam, não podem ser fornecidos já formatados pelos fabricantes. Vale talvez a pena, nesta fase, antes de discutirmos a forma como os discos são formatados, descrever os discos usados tipicamente pelos sistemas computadores.

Actualmente existem três tipos de disco no mercado: os discos «floppy» com um diâmetro de 5 a 1/4 polegadas, os discos «floppy» com um diâmetro de 8 polegadas e os discos rígidos ou tipo «Winchester».

Cada um destes tipos de disco armazena os dados magneticamente numa série de pistas concêntricas. Os discos menores podem guardar dados num ou em ambos os lados, e em 40 ou 80 pistas. Os maiores discos «floppy» podem também ser apenas de um lado ou de dois lados, e guardam geralmente os dados em 77 pistas concêntricas. Para aumentar um pouco mais a confusão, o armazenamento dos dados pode ralizar-se com uma densidade simples ou dupla; isto refere-se à quantidade de dados que podem ser «arrumados» em cada pista, e é decidido pelo fabricante dos discos. Por outras palavras, trata-se de uma medida da «magnetizabilidade» de substância magnética que reveste o suporte plástico de que o disco é feito. Os drives Winchester consistem em vários discos de alumínio revestidos

de material magnético, montados num mesmo eixo comum e encontram-se fechados hermeticamente numa caixa.

Cada pista de um disco é formada por uma série de «sectores» que podem armazenar um número específico de bytes, ou caracteres. Em alguns casos o número de bytes por sector e, aliás, o número de sectores por pista podem ser determinados pelo utilizador, se bem que em geral sejam decididos pelo fabricante do sistema computador. Se o número de sectores por disco pode ser decidido antes da formatação, diz-se que os discos são «soft-sectored», ou que os seus sectores são definidos por programa. Estes discos podem ser identificados pelo facto de existir apenas um orifício no disco, imediatamente fora do orifício central. Uma luz estroboscópica passando através deste orifício assegura que o disco roda a uma velocidade constante. Os fabricantes podem fornecer discos «hard-sectored» para certas unidades, possuindo vários orifícios dispostos em torno do orifício central. Não se podem substituir estes dois tipos de discos entre si. No caso das diskettes ou discos «floppy» menores, um formato típico poderá ser 80 pistas de cada lado, estando cada pista dividida em cinco sectores. Cada sector conterá 1024 bytes (1 K em linguagem de computadores), e portanto cada disco pode armazenar 5 × 80 × 2 × 1 K bytes, ou seja, 800K bytes. Um outro disco num outro sistema pode ser de um só lado, admitindo 80 pistas cada uma com 64 sectores, e uma capacidade total de 574K bytes.

Uma outra complicação é que alguns sistemas computadores que usam discos de dois lados podem ler ambos simultaneamente, tendo gravado os dados naquilo a que se chama «cilindros» do mesmo modo que os computadores grandes e médios organizam os seus discos. Isto significa que, em termos simples, metade de cada byte é guardada numa superfície, e a outra metade na segunda superfície. Reduz-se assim consideravelmente o tempo de acesso aos dados presentes em disco. Alguns discos Winchester usados em microcomputadores armazenam os seus dados deste modo. Alguns sistemas de disco «floppy», no entanto, podem tratar cada um dos lados do disco separadamente, como se se tratasse de discos diferentes, pelo que um sistema de dois discos pode de facto utilizar quatro dri-

ves «lógicos», designados por A, B, C e D e sendo cada um deles formatado separadamente. Os drives A e C correspondem a um mesmo disco, e os B e D ao outro.

Recentemente, uma revista apresentou uma lista de microcomputadores que usam CP/M; esta lista contém cerca de 130 máquinas diferentes. É mais do que provável que apenas alguns destes utilizem um formato de discos que permita a leitura numa destas máquinas de um disco formatado noutra. Isto provoca uma sucessão de problemas de compatibilidade, como é fácil de imaginar.

A acção de formatação de um disco, de modo a poder trabalhar num qualquer sistema, consiste, portanto, em definir o número apropriado de pistas e de sectores por pista, de forma a permitir ao disco receber os dados a registar.

Todos os sistemas CP/M possuem um programa de formatação, que é invocado, escrevendo simplesmente a instrução

```
A>FORMAT
```

Segue-se geralmente um diálogo em que o computador pergunta ao utilizador o drive onde este deseja fazer a formatação, por exemplo:

```
Diskette FORMAT Utility - Version 2.7
Format drive? (A or B; press return key to end)
Format drive B. Press space bar when ready
Format drive B
Format drive B complete
Format drive? (A or B; press return key to end)
A>
```

Esta sequência dá-nos a oportunidade de colocar um disco virgem não formatado no drive B, por exemplo, e de indicar este drive à máquina antes de a formatação se realizar. Se se colocar no drive B um disco que já possui dados registados, a acção de formatação destruirá tudo o que nele se encontra. Portanto, se se possui apenas um drive, o A, é necessário retirar o disco que contém o programa de formatação (FORMAT.COM ou FORMAT.CMD) e substituí-lo pelo disco virgem antes de executar a formatação.

Neste momento o leitor terá um disco já pronto a receber todos os ficheiros que quiser. No entanto, trata-se apenas de um disco «escravo»; deve encontrar-se sempre no drive B, acompanhado por um disco do sistema CP/M no drive A, na medida em que todos os utilitários de sistema que se pretende usar deve provir do disco presente no drive principal — a menos que se transfira o privilégio de que o drive A goza para o drive B, usando a instrução:

```
A>B:
B>
```

No entanto, se desligarmos o sistema e quisermos reiniciá-lo, o disco de sistema deve encontrar-se sempre no drive A. Por outras palavras, se quisermos criar um novo disco de sistema teremos de executar uma operação que transfigura os programas CP/M e os respectivos utilitários não apenas para o novo disco mas ainda para pistas específicas do novo disco. Tudo isto é realizado por um utilitário designado COPYSYS nos sistemas CP/M de versão 2.2 em diante, e SYSGEN nos de versão anterior à 2.2. No caso da versão do CP/M-86 fornecido com o Sirius ACT, o comando equivalente é chamado BOOTCOPY. Qualquer que seja o nome do utilitário, o efeito é, no entanto, sempre o mesmo: a cópia do sistema CP/M para pistas específicas do novo disco, de tal modo que, se este for em seguida colocado no drive A e se reinicializar o sistema, o CP/M arrancará automaticamente a partir dele como se se tratasse do disco original. COPYSYS, SYSGEN e BOOTCOPY podem ser colocados num disco sem afectar quaisquer ficheiros que já nele se encontrem.

No caso dos sistemas CP/M anteriores à versão 2.2 existe ainda um utilitário MOVCPM; este permite «relocatar» o CP/M em memória, isto é, alterar a sua localização dentro do computador, dependendo da quantidade de memória disponível no sistema que se usa. O modo como este programa funciona está fora do âmbito deste livro; basta dizer que é usado antes de SYSGEN a fim de relocatar o CP/M para aproveitar, do melhor modo, a quantidade de memória disponível. Por exemplo, se tivermos uma versão do sistema operativo que funcione em 32 K e a máquina dispuser de 64 K de RAM, a primeira coisa a fazer

é deslocar o CP/M de forma a aproveitar a memória extra. Isto será feito escrevendo:

```
A>MOVCPM64 (RET)
```

Em seguida pode-se usar SYSGEN, e gravar o novo sistema operativo.

Quando se usa um dos utilitários para copiar o sistema operativo para outro disco, ocorre o seguinte diálogo entre o utilizador e a máquina:

```
A>COPYSYS
Source drive name (or RETURN for default drive) (RET)
Source on A then type RETURN (RET)
Function complete              (isto indica-nos que o CP/M se
                                encontra já carregado em
                                memória)
Destination drive (RETURN to reboot) B
                               (isto permite-nos mudar de
                                opinião e abandonar a ope-
                                ração)
Destination on B then type RETURN (RET)
Function complete
```

A sequência consiste portanto em formatar um novo disco que será usado para guardar o sistema adaptado às capacidades da máquina; depois de ter realizado a formatação, coloca-se uma cópia do sistema CP/M naquilo a que se chama «pistas de sistema», de maneira que o novo disco possa actuar como um disco «master». Isto não significa que tenham sido copiados para o novo disco os utilitários que acompanham o sistema, como o PIP, o STAT, o ASM ou qualquer outro. Terão de ser copiados para o novo disco, se assim se quiser, usando PIP.

3

COMANDOS RESIDENTES DE CP/M

Além dos comandos propriamente ditos de CP/M, identificados pelos respectivos nomes seguidos de extensão .COM ou .CMD, existem vários outros que fazem parte dos utilitários incorporados no sistema. Estes comandos são DIR, ERA, REN e TYPE.

O comando DIR

Todos os ficheiros gravados no disco são indicados num índice guardado num directório que se encontra nas pistas reservadas ao sistema dentro do mesmo disco. Este directório guarda o nome de cada ficheiro juntamente com os parâmetros importantes relacionados com o ficheiro: o seu endereço inicial e respectivo comprimento. Se se escreve o comando

```
A>DIR
```

são listados no visor todos os nomes de ficheiros presentes no directório; por exemplo:

```
A>DIR
A:PIP  COM:MBASIC  COM:STAT     COM:PROG1  BAS:PROG2  BAS
A:WS   COM:WSMSGS  OVR:WSOVLYL  OVR:TEXT   TXT:TEXT   BAK
A>
```

Se quiséssemos poderíamos verificar a existência de um dado ficheiro escrevendo

```
A>DIR WS.COM
```


Se o ficheiro estiver no disco será impresso o seu nome. Se não estiver, será impressa a mensagem NO FILE.

Os nomes de ficheiro em CP/M podem conter até oito caracteres, mas nunca qualquer dos seguintes:

```
<>.,:;=/*[]
```

Pode-se escrever o nome de um ficheiro tanto em maísculos como em minúsculas; o CP/M guarda automaticamente todos os nomes em maiúsculas.

Os nomes de ficheiro podem ser prefixados pelo nome do drive seguido de dois pontos (:), de tal modo que os ficheiros A:FILE.DAT e B:FILE.DAT correspondem a ficheiros diferentes, encontrando-se um no drive A e outro no drive B.

Podem-se incluir caracteres «wild card» em certos comandos como DIR. Estes caracteres especiais têm um valor genérico, não tendo valor em si próprios mas designando um «tipo» de caracteres. Por exemplo, se quisermos obter uma lista de todos os ficheiros cujos nomes apresentam características comuns, por exemplo todos os ficheiros BASIC, caracterizados pela extensão .BAS, poderemos escrever

```
A>DIR *.BAS
```

o asterisco (*) empregue no comando designa de facto qualquer nome; nestas condições, o sistema imprimirá todos os ficheiros, independentemente do seu nome, que estejam gravados com a extensão .BAS.

Um outro caracter nas mesmas condições é o «?»; este caracter designa qualquer caracter, mas apenas um caracter e não um número indeterminado destes ao contrário do que acontece com o asterisco. Indica portanto um qualquer existe, ou não exista, na posição em que o escrevemos. Se escrevermos

```
A>DIR CPM???.*
```

obteremos uma lista de todos os ficheiros que começam pelas letras «CPM» seguidas ou não de outras, e que dispõem de uma qualquer extensão:

```
A:CPMCOM  :CPMCOMBAK  :CPM31BAK  :CPMASM
```

Mas não podemos escrever

```
A>DIR *31.*
```

Para listarmos todos os ficheiros com os algarismos 31 no final, necessitaríamos de escrever

```
A>DIR ??????31.*
```

Para listar todos os ficheiros presentes num disco diferente daquele que se encontra no drive principal, basta citar o nome do drive:

```
A>DIR B:
```

o que dará uma lista de todos os ficheiros no drive B.

Na versão 3.1 de CP/M, as potencialidades de DIR foram consideravelmente ampliadas e, nessa versão do sistema operativo, DIR é agora um comando por mérito próprio (ver capítulo 10).

O comando ERA

O comando ERA apaga, ou seja, elimina, um ficheiro de disco. Para sermos rigorosos, o ficheiro propriamente dito não é de facto eliminado, sendo-o apenas a sua indicação no directório. Isto significa que um ficheiro «eliminado» pode ser recuperado se pudermos repor a sua identificação no directório, no caso de o espaço não ter sido reescrito por outro ficheiro. Quando se apaga um ficheiro, a sua entrada no directório é eliminada e o espaço por ele ocupado em disco fica livre para uso nouro ficheiro. Para apagar um ficheiro, escreve-se ERA seguido de um nome, que deve ser completo e explicitar a extensão se esta existir:

```
A>ERA PROG1.BAS
```

Se, por qualquer razão, o ficheiro especificado não existir no disco, será impressa a mensagem NO FILE.

A fim de eliminar grupos de ficheiros de uma só vez, podem-se usar novamente caracteres «wild card»; por exemplo:

```
A>ERA *.BAS
```

provocará a eliminação de todos os ficheiros com a extensão. BAS. De facto pode-se ir mais longe e escrever

```
A>ERA *.*
```

eliminando todos os ficheiros presentes no disco — drástico, mas por vezes necessário. Para evitar desastres nesta operação, a máquina pergunta

```
ALL(Y/N)?
```

a fim de saber se queremos verdadeiramente apagar todos os ficheiros; temos então de responder Y («sim») ou N («não»).

O comando REN

O comando REN permite-nos alterar o nome de um ficheiro; uma vez mais, só será efectivamente alterada a sua entrada no directório. Basta-nos agora escrever

```
A>REN novonome=antigonome
A>
```

Assim, se quisermos alterar o nome, por exemplo, do ficheiro TEXT1.BAK no drive B para TEST1 no mesmo drive, escreveremos:

```
A>REN B:TEST1=A:TEST1
A>
```

Não se podem usar caracteres «wild card» com o comando REN, e o novo nome dado ao ficheiro não deve existir ainda no disco. Por outro lado não se podem alterar nomes de um disco para outro, pelo que

```
A>REN B:TEST1=A:TEST1
```

é ilegal e CP/M exigirá a alteração do comando dado.

Qualquer nome de ficheiro pode ser alterado, pelo que no caso de nos aborrecer a escrita de CBAS86 podemos modificar este comando para BAS escrevendo

```
A>REN BAS.CMD=CBAS86.CMD
A>
```

Depois, sempre que escrevermos o comando

```
A>BAS
```

será executado o programa CBAS86.

O comando TYPE

TYPE provocará sempre a impressão no visor do conteúdo do ficheiro nomeado. Assim,

```
A>TYPE TEXT1.TXT
```

listará todo o conteúdo do ficheiro no visor, uma página de cada vez. No final de cada página o utilizador é convidado a carregar em RETURN a fim de obter a página seguinte. É assim possível ler cada página antes de ser substituída pela seguinte.

Se escrevermos

```
A>TYPE B:TEXT2.TXT
```

imprimiremos do mesmo modo o ficheiro designado por TEXT2.TXT, que se encontra guardado no disco presente no drive B.

É necessário ter o cuidado de indicar apenas os ficheiros que podem ser lidos. Esta afirmação não é tão estúpida como parece porque nem todos os ficheiros são guardados sob a forma de caracteres ASCII. Um programa BASIC, por exemplo, que foi gravado usando MBASIC será guardado numa forma comprimida, usando aquilo a que se chama «tokens» em vez das palavras reservadas da linguagem BASIC. Qualquer tentativa de

TYPE num programa provocará, portanto, a impressão de um conjunto de símbolos ilegíveis. Se o mesmo programa tiver sido gravado sob a forma ASCII, usando

```
SAVE «PROG1»,A
```

é perfeitamente possível escrever TYPE para o imprimir no visor. Do mesmo modo, não vale a pena escrever

```
A>TYPE PIP.COM
```

dado que PIP, e todos os outros comandos, se encontra guardado em disco no formato de código-máquina, o que além de produzir uma impressão incompreensível pode até provocar um «crash» do sistema.

4

PIP, «PERIPHERAL INTERCHANGE PROGRAM»

Tal como acontece com o editor de linha, o PIP vem desde os primeiros tempos dos minicomputadores (por volta de 1968), e da série de máquinas PDP-8 da Digital Equipment Corporation. Nessa época, este programa era usado para ligar ficheiros em disco a um dispositivo de fita de papel, tanto leitores como perfuradores. Usando PIP, era então possível copiar um ficheiro guardado em fita perfurada directamente para disco. Do mesmo modo, podia-se copiar um ficheiro em disco para fita de papel perfurado. Daí o nome de «Peripheral Interchange Program», dado que o programa permitia trocas de ficheiros entre periféricos.

No início o programa apenas fazia isto: uma troca de dados entre dois equipamentos periféricos, a saber, um drive de disco e uma leitora ou perfuradora de fita de papel. A versão do PIP usada sob CP/M é hoje muito mais versátil, permitindo-nos deslocar dados entre muitos pares de dispositivos periféricos, e mesmo internamente a um mesmo dispositivo. Como PIP é um ficheiro de comando (possui a extensão .COM ou .CMD) basta citar o seu nome para o invocar:

```
A>PIP
*
```

O pedido «*» indica-nos que o programa está a esperar as nossas instruções.

A forma geral de um comando PIP é

```
Ficheiro de destino=ficheiro de fonte
```

de tal modo que se escrevermos

```
A>PIP
*B:FILE1=A:FILE2
```


o ficheiro chamado FILE2 no drive A (o A: antes do nome do ficheiro indica ao CP/M o drive em que o ficheiro se encontra actualmente guardado) será copiado para o drive B sob o nome de FILE1. Ao carregar na tecla RETURN ouviremos o ruído dos drives e veremos as luzes indicadoras, enquanto o ficheiro é copiado. Quando a transferência é terminada, reaparece o pedido «*», e poderemos realizar outra cópia ou abandonar o programa. Esta última operação realiza-se carregando na tecla RETURN. O diálogo entre o utilizador e a máquina será portanto:

```
A>PIP
*B:FILE1=A:FILE2
* (RET)
A>
```

Mas há mais alguma coisa a dizer sobre PIP. De facto, podemos reduzir o número de instruções. Se quisermos apenas transferir um ficheiro, basta-nos escrever:

```
A>PIP B:FILE1=A:FILE2
A>
```

Se não for apresentado qualquer especificador de ficheiro, PIP considera sempre que o utilizador se refere ao drive A. O nosso comando seria também válido se escrevêssemos

```
A>PIP B:FILE1=FILE2
A>
```

Se quisermos copiar o ficheiro de um disco para outro mantendo o mesmo nome, de modo a termos ficheiros idênticos com os mesmos nomes em ambos os discos, necessitamos apenas de

```
A>PIP B:=FILE2
A>
```

ou

```
A>PIP B: FILE2=A:
A>
```

em vez de

```
A>PIP B:FILE2=A:FILE2
A>
```

Poderíamos evidentemente produzir uma cópia do ficheiro no mesmo disco, sob um novo nome, escrevendo

```
A>NOVOFICHEIRO=ANTIGOFICHEIRO
A>
```

Nestas condições obteríamos duas cópias do mesmo ficheiro no drive A, sendo uma designada por ANTIGOFICHEIRO e outra por NOVOFICHEIRO. Poderá não haver grande interesse em fazer isto, mas por vezes é útil e PIP permite-nos fazê-lo.

PIP permite-nos igualmente realizar a operação de concatenação de ficheiros. Esta operação junta dois ficheiros num único. Isto é muito útil no caso de ficheiros de texto, capítulos ou parágrafos separados, criados separadamente e juntos depois num ficheiro mais extensos. Por exemplo

```
A> B:NOVOTEXTO.TXT=A:TEXT1.TXT,A:TEXT2.TXT
A>
```

produzirá um ficheiro chamado NOVOTEXTO.TXT no drive B, fomado a partir dos ficheiros no drive A, chamados TEXT1.TXT e TEXT2.TXT, ligados num ficheiro único.

Outros ficheiros podem ser manipulados usando PIP através da inclusão de caracteres «wild-card», de tal modo que se podem transferir grupos de ficheiros com certas características comuns nos seus nomes. Assim, se escrevermos

```
A>PIP B:=*.BAS
```

não só serão transferidos para o drive B todos os ficheiros com uma extensão .BAS que se encontram no drive principal, o A, como ainda serão listados à medida que forem transferidos.

```
A>PIP B:=*.BAS

COPYING-
PROG1.BAS
PROG2.BAS
PROG4.BAS
PROG5.BAS
PROG8.BAS
A>
```


Este método é simultaneamente um modo simples e rápido de produzir cópias de «back-up» num disco diferente, e de facto é uma boa forma de copiar todos os ficheiros de um disco para outro escrevendo

```
A>PIP B:=*.*
```

Mas tenha cuidado, deverá assegurar que há espaço suficiente no segundo disco para receber todos os ficheiros. A cópia de um disco inteiro usando este método pode ser demorada, e seria aborrecido que tudo ficasse perdido ao transferir o último ficheiro devido a falta de espaço. Já aconteceu muitas vezes...

O PIP pode ser igualmente usado para transferir ficheiros de disco, por exemplo, para outro dispositivo. De facto, foi esse o objectivo original do programa. Mas agora o dispositivo em causa pode ser qualquer dos periféricos ligados ao seu computador. Por exemplo, pode-se definir o destino da transferência como sendo a impressora, escrevendo:

```
A>PIP LST:=B:TEXT.TXT
A>
```

e o ficheiro será impresso no periférico especificado no sistema, normalmente a impressora. De facto, a fim de garantir que o texto não passa sobre as perfurações no caso do papel contínuo, podemos forçar a impressora a fazer uma mudança de página em cada 60 linhas, incluindo um dado extra no comando:

```
A>PIP LST:=B:TEXT.TXT[P60]
A>
```

Pode-se igualmente transferir um ficheiro de um dispositivo para um ficheiro em disco, ou para outro periférico, especificando o dispositivo de entrada como, por exemplo, RDR:, e o de saída como CON: (terminal de consola). Se escrevermos

```
A>PIP CON:=RDR:
A>
```

imprimirá o ficheiro no visor do terminal; e se escrevermos

```
A>PIP FILE1=RDR:
A>
```

transferirá o ficheiro para um disco.

O que se disse pode ser usado para transferir ficheiros de um computador para outro, ligando um dispositivo de saída de uma máquina ao dispositivo de entrada de outra.

Um outro acrescendo importante ao comando é aquele que diz a PIP que incide a transferência numa determinada cadeia de caracteres. Por exemplo:

```
A>PIP LST:=B:TEXT.TXT[SREM***^Z]
A>
```

Note-se que ^Z é CONTROL-Z, ou seja, o resultado de se carregar simultaneamente nas teclas CONTROL e Z. Isto forçará a impressão a iniciar-se na cadeia «REM***» no ficheiro designado por TEXT.TXT do drive B. Do mesmo modo, poderemos indicar que a transferência deve ser terminada quando se atinge uma cadeia em particular. Assim,

```
A>PIP B:TEXT1.TXT[QEND^Z]
A>
```

transferirá todo o ficheiro do drive principal designado por TEXT1.TXT até à cadeia «END» inclusive.

Podem ser usados muitos outros acrescentos, ou parâmetros, neste comando, mas estes podem ser experimentados depois de o utilizador se ter familiarizado com o uso de PIP. Damos em seguida uma lista completa destes possíveis parâmetros:

Efeito	Parâmetro
B	Transfere em bloco, permitindo a passagem de dados de, por exemplo, uma cassette para um ficheiro em disco

Dn	Elimina todos os caracteres para além da coluna n
E	Ecoa a saída para a consola
F	Elimina as mudanças de página do ficheiro que está a ser transferido
Gn	Obtém o ficheiro da área **n** do utilizador. Permite-nos copiar de uma área do utilizador para outra
H	Transfere apenas dados hexadecimais válidos
I	Ignora quaisquer registos «:00» e transfere apenas caracteres hexadecimais
L	Converte todos os caracteres maiúsculos para minúsculos
N	Numera cada linha, começando em 1 e incrementando de 1
N2	Tal como com o parâmetro N, mas coloca zeros antes do número de linha e um caracter «tab» depois dele
O	Transfere ficheiros-objecto não ASCII e ignora os separadores normais de final de ficheiro
Pn	Insere um caracter de mudança de página depois de cada n linhas de texto. Se n = 1 ou for omitido, as mudanças de página são incluídas depois de cada 60 linhas. Usado em conjunto com o parâmetro F
Qcadeia^Z	Abandona a cópia quando encontra a cadeia especificada

R	Transfere ficheiros de sistema de outro modo não acessíveis a PIP
Scadeia^Z	Copia um ficheiro a partir da primeira ocorrência da cadeia no ficheiro fonte
Tn	Coloca um caracter «tab» em cada enésima coluna
U	Traduz todos os caracteres maiúsculos para minúsculos
V	Verifica se os dados foram correctamente transferidos
W	Escreve ficheiros que foram protegidos como R/O (leitura apenas)
X	Copia ficheiros que não são cadeias de caracteres ASCII
Z	Passa a zero o bit de paridade de todos os caracteres ASCII existentes no ficheiro que está a ser transferido

Mensagens de erro do PIP

DESTINATION IS R/O, DELETE (Y/N)?	O ficheiro de destino já existe e é um ficheiro apenas de leitura. Pretende eliminá-lo antes de executar a transferência?
DISK READ ERROR: nomeficheiro	Não pode ler o ficheiro de entrada
DISK WRITE ERROR: nomeficheiro	Não pode abrir ficheiro para escrita
NO FILE:-nomeficheiro	O ficheiro de origem não existe
CANNOT WRITE: nome dispositivo	O dispositivo destinatário não existe ou é ilegal. Talvez o utilizador tenha tentado escrever para CDR

INVALID FORMAT: entrada	A sua instrução não se encontra no formato PIP válido
CANNOT READ: dispositivo/nomeficheiro	Origem ilegal. Por exemplo, leitura de LPT
QUIT NOT FOUND: =origem	O argumento de cadeia especificado num parâmetro Q não é encontrado
START NOT FOUND: =origem	O argumento de cadeia especificado num parâmetro S não é encontrado
ABORTED=nomeficheiro	PIP abortado por se ter carregado numa tecla
VERIFY ERROR: nomeficheiro	Ao copiar usando a opção V PIP encontrou uma diferença entre os dados lidos e os dados escritos

5

O COMANDO STAT

STAT é usado para determinar o estado dos ficheiros armazenados num disco ou do próprio disco, além de fornecer algumas outras informações úteis, como se descreverá adiante.

Na sua forma mais simples o comando STAT diz-nos a quantidade de espaço livre de que dispomos nos discos e que no momento se encontram nos drives. Por exemplo, poderemos ver o que se segue no visor:

```
A>STAT
A:R/W,Space:200K
B:R/O,Space:16K
A>
```

o que nos indica que dispomos no drive A de 200K bytes de espaço livre e que podemos ler e escrever nos ficheiros contidos nesse disco. O disco B, no entanto, encontra-se no modo de leitura apenas e dispõe somente de 16K bytes livres, onde poderemos colocar outros ficheiros. Note-se que se quisermos escrever seja o que for neste espaço de 16K bytes seremos forçados a alterar o estado do disco, dado que neste momento apenas o podemos utilizar em leituras.

Se quisermos definir o estado do disco num único drive podemos escrever

```
A>STAT B:
```

e obteremos o estado do disco que se encontra no drive B; mas nestas condições será apenas impresso o espaço livre, sem incluir quaisquer informações quanto ao estado do disco em termos de leitura e escrita.

A fim de obtermos mais informação sobre os ficheiros de um disco, para além dos seus nomes (fornecidos por DIR), podemos escrever, por exemplo,

```
A>STAT B:*.BAS
```

o que nos permite obter informações sobre todos os ficheiros do drive B com uma extensão .BAS. O resultado poderá ser

```
 Recs    Bytes     Ext Acc
    3      2k      1 R/W   B:BASE16.BAS
   16      2k      1 R/W   B:CREDIT.BAS
    2      2k      1 R/W   B:GREAT.BAS
    3      2k      1 R/W   B:FILEBM.BAS
   57      8k      1 R/W   B:PINSTALL.BAS
    2      2k      1 R/W   B:PRINT.BAS
   14      2k      1 R/W   B:SLOT.BAS
    6      2k      1 R/W   B:SORTCL.BAS

Bytes Remaining on B: 28k
```

A informação que nos é apresentada nestas tabelas diz-nos muitas coisas sobre cada um destes ficheiros, todos eles programas BASIC no caso citado, e indirectamente sobre o próprio CP/M. O leitor notará que a informação está dividida em «Recs», que são o número de registos de 128 bytes que se encontram guardados em cada «extent» do ficheiro, sendo que cada «Ext» vale 16K bytes. O número de bytes atribuído ao ficheiro é sempre um múltiplo de 2K, mesmo apesar de o conteúdo real do ficheiro ser bastante reduzido. Por exemplo, na tabela apresentada, o programa SLOT tem um comprimento de um pouco mais de cem linhas, enquanto que o programa BASE16 tem apenas um quinto desse comprimento. Ambos recebem o mesmo espaço em disco, porque o CP/M só pode funcionar em pedaços de 2K bytes. Antes do nome do ficheiro, precedido pelo nome do drive, encontra-se o acesso ao ficheiro; neste caso são todos os ficheiros que aí podem ser lidos ou escritos.

Se se escreve $S após o nome do ficheiro indicado num comando STAT, obtém-se uma nova coluna de informações. Esta apresenta o título «Size». Se o ficheiro especificado é um ficheiro serial, a entrada sob «Size» será igual à existente sob

«Recs». Isto deve-se ao facto de a entrada «Size» ser o número de registos actualmente atribuídos ao ficheiro. No entanto, se o ficheiro for aleatório, caso em que foram atribuídos vários registos mas nem todos foram preenchidos, «Size» será menos do que «Recs» como se mostra abaixo:

```
Size     Recs     Bytes     Ext Acc
500         8        2k       1 R/W A:RANDOM.DAT
```

Podemos usar igualmente STAT para tornar certos ficheiros «transparentes» relativamente aos comandos DIR e PIP transformando-os em ficheiros de «sistema». Isto é feito colocando uma extensão $SYS num comando STAT

```
A>STAT PIP.CMD $SYS
```

A resposta do CP/M é

```
PIP.CMD set to SYS
A>
```

Isto garante que, quando o directório é listado, PIP não aparece nessa lista e, no caso de uma listagem STAT dos ficheiros, por exemplo STAT *.CMD, os ficheiros SYS são listados com os respectivos nomes entre parêntesis.

Será assim impresso o seguinte:

```
A>DIR PIP.CMD
A>                          (Sem resposta, PIP está «escondido»)

A>STAT *.CMD
Recs     Bytes     Ext Acc
227        30k     2 R/W A:BASIC86.CMD
 43         6k     1 R/W A:DCOPY.CMD
 64         8k     1 R/W A:(PIP.CMD)

Bytes remaining on A: 486k
```

Para devolver o ficheiro ao seu estado anterior usamos $DIR em vez de $SYS.

Podemos igualmente usar STAT para passar um drive inteiro ao modo de leitura apenas, ou a leitura/escrita. Por exemplo

```
A>STAT A:R/O
```


passará o disco no drive A para leitura apenas, de tal modo que não é possível alterar ou apagar qualquer ficheiro nele existente. No entanto, isto só é eficaz enquanto o disco se encontra nesse drive e até o sistema ser novamente inicializado. Se o disco for inserido de novo depois de o sistema ser inicializado, todos os ficheiros voltam ao seu estado de leitura/escrita. Se quisermos passar qualquer ficheiro permanentemente para o estado de leitura apenas, o comando será

```
STAT A:PROG1.DAT $R/O
```

a única maneira de alterar este estado consiste em passá-lo explicitamente para leitura/escrita com

```
STAT A:PROG1.DAT $R/W
```

Note-se que o sinal $ é usado quando se referem ficheiros num disco, não sendo requerido esse sinal quando nos referimos ao conjunto do disco.

Em seguida trataremos de um aspecto muito importante do comando STAT: o que trata do estado dos dispositivos, tanto lógicos como físicos, que o sistema reconhece. Um dispositivo lógico é aquele que é conhecido pelo sistema operativo, e consiste de:

- `CON:` O dispositivo combinado de entrada e saída que constitui o principal equipamento para comunicação com o processador; geralmente um conjunto monitor-teclado
- `LST:` Um dispositivo de saída para onde são enviadas todas as listagens; por outras palavras, um dispositivo de impressão ou de cópia «dura»
- `RDR:` Um dispositivo de entrada, tradicionalmente uma leitora de fita de papel perfurada
- `PUN:` Um dispositivo de saída, também tradicionalmente uma perfuradora de fita de papel

Um dispositivo físico será o equipamento real que é equiparado ao dispositivo lógico. Normalmente os ligadores encontram-se nas costas do computador. Na maior parte das vezes exis-

tem pelo menos dois; um será um ligador conhecido pela designação «Centronics», geralmente usado para ligação e uma impressora paralela. A outra «tomada», mais pequena, designada por ligador RS 232, é usada para ligação a uma impressora serial ou a outro dispositivo — por exemplo, outro computador. Uma impressora paralela é aquela que obtém os seus dados por oito condutores simultaneamente (em paralelo), e uma impressora serial é aquela que obtém os seus dados numa cadeia de oito bits, um a seguir ao outro, por um único condutor. Estas são obviamente descrições muito simples das diferenças existentes entre os dois métodos de comunicação, mas bastarão por agora. Os nomes possíveis para os dispositivos físicos reconhecidos pelo CP/M são

`TTY:` Um dispositivo do género teletipo que funciona a baixa velocidade e recebe os seus dados serialmente. Pode ser um dispositivo de entrada e saída dado que as teletipos são uma combinação de impressora e teclado

`CRT:` Um dispositivo combinado de entrada e saída de velocidade muito alta, geralmente um dispositivo tipo monitor, e um teclado

`LPT:` Um dispositivo de impressão que funciona através do porto paralelo — isto é, é um dispositivo de tipo Centronics

`PTP:` Dispositivo de perfuração de fita de papel de alta velocidade

`PTR:` Dispositivo de leitura de fita de papel de alta velocidade

`BAT:` Processamento em modo «batch»; o dispositivo de entrada passa a RDR:, e o de saída a LST:. Isto permite ao computador funcionar em «batch»

`UP1:` e `UP2:` perfuradoras definidas pelo utilizador; ou seja, dispositivos de saída numerados 1 e 2

`UR1:` e `UR2:` são leitoras definidas pelo utilizador; ou seja, dispositivos de entrada numerados 1 e 2

`UL1:` Um segundo dispositivo de listagem; isto é, uma outra impressora serial

`UC1:` Uma segunda consola, definida pelo utilizador

A fim de descobrir qual a configuração actual do seu computador, deve escrever:

```
A>STAT DEV:
```

e a resposta pode ser

```
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is LPT:
```

Estas indicações dizem-nos que o dispositivo de consola actual é um CRT — por outras palavras, o monitor e o seu teclado. O dispositivo de listagem actual é uma impressora de tipo Centronics, e tanto RDR: e PTP: são capazes de comunicar com um dispositivo de tipo teletipo, um dispositivo de entrada e saída seriais de baixa velocidade.

```
A>STAT VAL:
```

e obteremos

```
Temp R/O Disk: d:R/O
Set Indicator: d:filename:typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
USER Status  : USR:
Iobyte Assign:
CON:= TTY: CRT: BAT: UC1:
RDR:= TTY: PTR: UR1: UR2:
PUN:= TTY: PTP: UP1: UP2:
LST:= TTY: CRT: LPT: UL1:
```

A tabela acima indica-nos todas as reatribuições que podem ser feitas usando o comando STAT. As primeiras duas linhas já foram de facto tratadas anteriormente neste mesmo capítulo. As quatro últimas linhas apresentam-nos todas as configurações possíveis envolvendo os quatro periféricos. Por exemplo, em relação ao dispositivo de listagem LST: pode ser uma impressora serial (TTY:), o monitor (CRT:), uma impressora paralela Centronics (LPT:) ou o outro porto serial (UL1:). Realizamos esta atribuição escrevendo

```
A>STAT LST:=CRT:
```

de tal modo que quando escrevemos

```
A>STAT DEV:
```

obtemos

```
CON: is CRT:
RDR: is TTY:
PUN: is TTY:
LST: is CRT:
```

o que significa que sempre que queremos uma listagem, ou damos o comando ^P, que geralmente dirige todas as saídas seguintes para a impressora, estas ocorrerão de facto no visor.

Podem-se ligar comandos STAT DEV:, pelo que podemos escrever

```
A>STAT DEV:LST:=LPT:,CON:=TTY:
```

Antes de continuarmos convém notar que tudo o que o comando STAT faz é emparelhar cada dispositivo lógico com um dispositivo físico. Se pensarmos no microprocessador como tendo quatro «canais» através dos quais pode comunicar com o mundo exterior — e através dos quais o mundo exterior pode comunicar com ele — teremos uma ideia destes quatro dispositivos lógicos. Podemos então começar a compreender que existem muitos periféricos físicos diferentes que podem encontrar-se ligados aos quatro canais; impressoras de vários tipos e modelos, leitores de cartões, leitoras de códigos de barras, e até outros computadores. É através destes portos de entrada e saída que os computadores podem «falar» entre si através de linhas telefónicas, através de um dispositivo chamado «modem». O modem (palavra que significa **mo**dulador/**des**modulador) é ligado a um porto de entrada/saída, e este converte a cadeia de dados do computador para uma forma que permite o seu transporte pela rede telefónica pública. Na outra extremidade da linha telefónica encontra-se outro modem, ligado através do seu porto de entrada a outro computador que descodifica os dados e os coloca à disposição.

Os periféricos seriais, em particular, não têm todos as mesmas características operativas. É necessário fazer mais alguma coisa, além de afirmar que se trata de um dispositivo do «género teletipo», para levar o computador a «conversar» com eles de modo inteligível.

Por exemplo, se indicarmos ao CP/M que vamos utilizar uma impressora serial como dispositivo de listagem, teremos igualmente de dizer ao CP/M qual a velocidade a que os dados lhe vão ser transmitidos, assim como outras informações. A velocidade é expressa em «baud», uma unidade equivalente ao número de bits que serão transmitidos por segundo. A propósito, vale a pena notar que nem todos os bits são bits de dados; alguns deles contêm informações que permitem a sincronização da transferência com a velocidade a que o dispositivo receptor pode tratar a cadeia de dados. Por exemplo, não vale a pena transmitir dados a 9600 bits por segundo para um periférico que apenas pode tratar 110 bits por segundo. Um outro programa usado nas versões do CP/M até à 2.2 é chamado CONFIG, e permite-nos adaptar a forma como os dados são produzidos na saída dos portos à forma em que o dispositivo receptor deles necessita. A versão 3 do CP/M permite-nos realizar esta operação de adaptação de uma forma muito mais simples, e que trataremos no capítulo 10.

Finalmente, podemos utilizar STAT para obtermos informação relativa às características dos drives de disco. Escrevemos:

```
A>STAT DSK:
```

e poderemos obter uma resposta como

```
   A: Drive Characteristics
4512: 128 Byte Record Capacity
 564: Kilobyte drive Capacity
 128: 32 Byte Directory Entries
 128: Checked Directory Entries
 128: Records/Extent
  16: Records/Block
  64: Sectors/Track
   0: Reserved Tracks
```

Isto indica-nos várias coisas interessantes, incluindo a quantidade de entradas em directório que poderemos usar. Neste caso serão 128. Isto significa que por muito pequenos que sejam os ficheiros, e não esqueçamos que o seu menor tamanho possível é 2K, o número máximo de ficheiros guardados no disco é 128. Dado que o número máximo de bytes de informação que podem ser guardados no disco é 564K, temos uma situação em que podemos dispor de 128 ficheiros pequenos, ocupando 128 × 2K bytes, num total de 256K, tornando-se impossível gravar qualquer outro ficheiro no disco porque o directório já se encontra cheio. O resultado disto é que mais de metade do disco não será usado. Convirá portanto ter o cuidado de não guardar muitos ficheiros pequenos num disco.

Se por exemplo uma empresa utiliza um programa baseado em CP/M, como o WORDSTAR, para produzir descrições «standard», não será possível manter num disco pormenores relativos a mais de 128 linhas de produtos.

6

ED, EDITOR DE LINHA DE CP/M

O editor «standard» fornecido em todas as versões do CP/M tem uma linhagem extensa e honrosa. Apresenta grandes semelhanças com um certo número de editores desenvolvidos inicialmente para os minicomputadores na década de 1960, em particular nas máquinas de DEC e nas linhas Nova e Eclipse da Data General. Todos os que usam um microcomputador com CP/M sentem a facilidade de uso de ED se já tiverem tido alguma experiência ao nível de sistema operativo num minicomputador.

ED é um editor de **linha**, e a sua principal utilidade consiste na criação de código-fonte para programas escritos em linguagens diferentes da BASIC; de facto, para as linguagens que necessitam de ser «assembladas», usando os vários assembladores disponíveis em CP/M, ou compiladas usando os compiladores de linguagens como a COBOL, a FORTRAN e a CBASIC. O autor considerou sempre bastante estranho que, apesar de o principal uso de ED ser o acima descrito, os manuais e muitos outros textos insistem sistematicamente na explicação do seu uso recorrendo a poemas e citações de Shakespeare! Este livro tentará tratar ED a partir do que este é na realidade.

Em primeiro lugar, o leitor deve compreender que o editor guarda texto em memória, numa área de «buffer», sob a forma de uma série de cadeias de caracteres separadas por caracteres de «retorno de linha». Cada uma destas cadeias de caracteres é identificada por um número de linha único que lhe é atribuído pelo editor, razão por que este é definido como um editor «de linha» por oposto àqueles que tratam «blocos» de texto, como acontece num programa de tratamento de texto. Os números

de linha são fornecidos por ED a fim de actuarem simplesmente como pontos de referência, e não fazem parte do ficheiro que está a ser editado. Um indicador de caracteres (designado por CP) desloca-se através do buffer ao serem introduzidos comandos pelo teclado.

Para começar, vejamos o que acontece quando pretendemos criar um ficheiro contendo caracteres que vão encontrar-se no formato de um programa CBASIC, apresentando portanto a extensão .BAS após o nome do ficheiro. CBASIC é uma versão da bem conhecida linguagem BASIC, tendo a característica de ser compilada, como veremos no capítulo 10. Caracteriza-se pela falta dos habituais números de linha e o uso de «etiquetas» numéricas a fim de marcar as linhas de programa para onde serão realizados saltos. Se escrevermos

```
ED DEMO.BAS
```

ED responderá com o pedido

```
NEW FILE
     : *
```

e ficará à espera de novas instruções.

Reconhecendo que não existe ainda qualquer ficheiro designado DEMO.BAS, ED diz-nos que foi criado um ficheiro novo e que lhe foi atribuída uma área de buffer vazia, na qual as «linhas» do nosso programa serão guardadas à medida que forem escritas. A nossa primeira tarefa consiste em indicar que queremos inserir texto no buffer, e ED responderá apresentando-nos uma linha vazia com o número 1, como se mostra:

```
ED DEMO.BAS

NEW FILE
     : *I
    1:
```

Estamos agora preparados para escrever a primeira linha do programa, e quando carregarmos na tecla RETURN esta será

colocada no buffer, sendo-nos «aberta» a linha seguinte. Isto é traduzido pelo seguinte:

```
ED DEMO.BAS

NEW FILE
     : *I
    1: REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
    2:
```

É um pouco como introduzir um programa MBASIC normal usando a facilidade AUTO. Introduzimos cada linha do nosso programa, e acabamos por obter algo com a seguinte aparência:

```
ED DEMO.BAS

NEW FILE
     : *I
    1: REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
    2: OPEN ''DEMFILE'' AS 1
    3: INPUT ''QUANTOS REGISTOS'';N
    4: FOR I = 1 TO N
    5: INPUT ''ESCREVA UMA FRASE'';A$
    6: PRINT#1;A$
    7: NEXT I
    8: CLOSE 1
    9: END
    10:
```

Depois de terminarmos a introdução do texto, terminamos escrevendo ^Z, o que nos conduz de novo ao pedido de entradas *:

```
ED DEMO.BAS

NEW FILE
     : *1
    1: REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
    2: OPEN ''DEMFILE'' AS 1
    3: INPUT ''QUANTOS REGISTOS''; N
    4: FOR I = 1 TO N
    5: INPUT ''ESCREVA UMA FRASE'';A$
    6: PRINT#1; A$
    7: NEXT I
```

```
 8: CLOSE 1
 9: END
10: ^Z (não impresso no visor)
  : *
```

Se escrevermos E — indicando que queremos terminar a sessão — o ficheiro será automaticametne gravado em disco, sob o nome DEMO.BAS.

Neste ponto, podemos tentar executar o programa (do modo indicado no capítulo 8), e descobriremos que foi cometido um erro no programa CBASIC. O erro deve ser corrigido, para o que teremos de inserir uma nova linha. Voltamos, portanto, a chamar o ficheiro original, o que será feito em duas fases. Antes do mais

```
ED DEMOS.BAS
```

produz

```
: *
```

Temos de indicar ao editor que queremos chamar o ficheiro em causa para a área de buffer, e portanto escrevemos 9A dado que havia nove linhas no programa original. Um método mais simples consistirá em escrever «#A» para copiar todo o ficheiro para o buffer. 9A copia as primeiras nove linhas do ficheiro, pelo que o visor fica com a seguinte aparência:

```
ED DEMO.BAS
    1: *9A
    1: *
```

Sabemos assim que o ficheiro se encontra já carregado em memória e que o CP (indicador de caracteres) se encontra na primeira linha do ficheiro. Neste ponto, poderemos pedir que sejam impressas algumas linhas; por exemplo, 3T imprimirá as três linhas seguintes, e assim por diante. «#T» imprime todo o conteúdo do buffer, como é óbvio. O número de linhas impressas é contado a partir da posição actual do indicador. Para deslocarmos o indicador escrevemos um número seguido da letra L, a fim de instruir o editor no sentido de se deslocar para diante

de um certo número de linhas; se escrevermos antes do número o prefixo «-», o CP deslocar-se-á para trás o número de linhas indicado. Poderemos ter, por exemplo:

```
ED DEMO.BAS
    : *9A
   1: *3T
   1: REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
   2: OPEN ''DEMFILE'' AS 1
   3: INPUT ''QUANTOS REGISTOS'';N
   1: * (notar que o número de linha volta a 1,
      indicando que CP não se deslocou)
```

ou poderemos ter

```
ED DEMO.BAS
    : *9A
   1: *2L
   3: *
```

colocando o CP no início da terceira linha.

Podemos deslocar o indicador de um certo número de caracteres de cada vez se necessário — não esquecendo que o «retorno de linha-mudança de linha» no final de cada linha conta como dois caracteres. Teríamos portanto

```
ED DEMO.BAS
    : *9A
   1: *2L   (Passar duas linhas)
   3: *8C   (Mover CP 8 caracteres)
   3: *2T   (Escrever duas linhas)
UANTOS CARACTERES '';N
   3: FOR I = 1 TO N
   3: *
```

Isto deve-se ao facto de termos deslocado o indicador ao longo da linha 3 de oito caracteres, iniciando-se a impressão na posição actual do indicador; obtemos portanto a parte restante da linha 3 e toda a linha 4.

Para tentarmos corrigir o nosso erro devemos inserir uma nova linha antes da linha 3; por outras palavras, criamos uma nova linha 2. Para tal, movemos o indicador para a linha em que pretendemos fazer a inserção, escrevendo depois a linha. Pas-

samos o indicador para o início da linha 2 usando 1L. Notemos que 1L nos conduz da linha 1 para o início da linha 2. 2L levar-nos-a do início da linha 1 para o início da linha 3; 10L levar-nos-ia para o início da linha 11. Escrevemos I, indicando que deseja introduzir caracteres, e carregamos no «retorno de linha»; depois podemos escrever a nossa linha extra, terminando-a por um «retorno de linha» que nos conduzirá para a linha seguinte.

Se assim quisermos, podemos escrever uma outra linha, e outra ainda, até termos inserido tantas linhas quantas quisermos. Depois de escrevermos todas as linhas, terminamos a inserção destas usando ^Z, como se mostra adiante:

```
ED DEMO.BAS
    : *9A
   1: *1L
   2: *I
   2:  CREATE ''DEMFILE'' AS 1
   3: ^Z
   3: *E
```

O comando **E**nd terminará a sessão, voltando o sistema ao modo de entrada normal:

```
A>
```

Dispomos agora da última versão do ficheiro designado DEMO.BAS. A versão original é retida e designada DEMO.BAK (uma cópia de back-up para o caso de se verificarem acidentes).

De facto o nosso programa ainda não está correcto, como veremos, e necessita de novas emendas:

```
ED DEMO.BAS
    : *9A
   1: *1L
   2: *I
   2:  IF END#1 THEN 5
   3: ^Z (Não impresso)
   3: *1K (O comando K elimina - ''kill'' - linhas)
   3: *1T
   3:  OPEN ''DEMFILE'' AS 1
   3: *L
   4: *I
```



```
   4: GOTO 10  (Estas duas linhas são inseridas
   5:  5        na linha 4)
   6: ^Z
   6: *B  (O comando B reenvia-nos para o início
   1: *6T  do ficheiro)
   1:  REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
   2:  IF END#1 THEN 5
   3:  OPEN ''DEMFILE'' AS 1
   4:  GOTO 10
   5:  5
   6:  INPUT ''QUANTOS REGISTOS '';N
   1: *5L
   6: *I
   6:  CREATE ''DEMFILE'' AS 1  (Mais duas linhas
   7: 10                        inseridas na linha 6)
   8: ^Z
   8: *B
   1: *8T
   1:  REM***PROGRAMA CBASIC DE DEMONSTRAÇÃO***
   2:  IF END#1 THEN 5
   3:  OPEN ''DEMFILE'' AS 1
   4:  GOTO 10
   5:  5
   6:  CREATE ''DEMFILE'' AS 1
   7:  10
   8:  INPUT ''QUANTOS REGISTOS '';N
   1: *E
```

Se o leitor seguir cuidadosamente a anterior sequência de comandos verificará que o ficheiro de instruções CBASIC foi emendado de tal modo que agora é executado correctamente, como se mostra no capítulo 8.

Mas como podemos fazer uma alteração de uma linha a fim de conseguirmos, no nosso exemplo, uma melhor apresentação quando o programa executa? De facto, podemos fazê-lo de várias maneiras, e a mais simples consiste em dar um comando que diga «descobrir o texto incorrecto e substituí-lo por outro; fazer isto um certo número de vezes»:

```
ED DEMO. BAS
    : *14A  (O ficheiro contém agora 14 linhas
   1: *      de programa)
```

escrevemos SREGISTOS^ZREGISTOS ?:

O «S» significa «procurar e substituir» a cadeia indicada em seguida, cujo final é marcado por ^Z; a cadeia de substituição é indicada a seguir. Ao carregarmos em «return» a operação será executada. Substituímos assim «REGISTOS» por «REGISTOS?:».

O CP encontra-se agora na linha 8, imediatamente após o último caracter substituído, de tal forma que uma instrução que imprima a linha imprimirá o resto desta, a saber, '';N. Para verificarmos se a emenda foi introduzida correctamente, podemos escrever -1L2T, o que indica ao editor que deve recuar uma linha e escrever depois as duas linhas que se seguem; obteremos:

```
ED DEMO.BAS
     : *14A
    1: *SREGISTOS^REGISTOS ?:
    8: *T
'';N
    8: *-1L 2T
    7: 10
    8: INPUT ''QUANTOS REGISTOS ?:'';N
    7: *
```

Em seguida alteraremos o «INPUT» da linha 10, transformando-o num «PRINT», e inseriremos uma linha de INPUT imediatamente após a 10. Deslocamos o indicador para um ponto anterior às linhas a modificar:

```
 7: *2L
 9: *SINPUT^ZPRINT
10: *
```

Passamos depois FRASE '';A$ para FRASE #:'';I; incluindo depois uma nova linha contendo a instrução INPUT A$, como se segue:

```
10: *SFRASE '';A$^ZCADEIA #:'';I;
10: *1L
11: *I
11:  INPUT A$
12: ^Z
12: *
```


Vejamos a totalidade do ficheiro:

```
12: *B
 1: *#T
 1:  REM***PROGRAMA DE DEMONSTRAÇÃO CBASIC***
 2:  IF END#1 THEN 5
 3:  OPEN ''DEMFILE'' AS 1
 4:  GOTO 10
 5:  5
 6:  CREATE ''DEMFILE'' AS 1
 7:  10
 8:  INPUT ''QUANTOS REGISTOS ?: '';N
 9:  FOR I = 1 TO N
10:  PRINT ''ESCREVA UMA FRASE #:'';I;
11:  INPUT A$
12:  PRINT#1;A$
13:  NEXT I
14:  CLOSE 1
15:  END
 1: *
```

Depois de a edição estar terminada, pode-se gravar a nova versão do texto escrevendo E em resposta ao pedido de entradas «*». A última versão do ficheiro é gravada sob o actual nome, sendo retida um ficheiro de back-up do original, não editado, com a extensão .BAK.

Existem evidentemente muitos outros comandos para controlo do editor. Por exemplo, podemos eliminar linhas a partir da posição actual do CP usando o comando

```
nK
```

onde «n» é o número de linhas a eliminar.

Do mesmo modo

```
nD
```

elimina um número específico de caracteres antes (n negativo) ou depois (n positivo) de CP.

Podem-se localizar cadeias específicas de caracteres usando «F» («Find»), de tal modo que, por exemplo:

```
3FCOUNTER^Z
```

encontrará a terceira ocorrência da palavra «COUNTER» num ficheiro e posicionará o CP depois do último caracter da última cadeia encontrada.

O comando «E» terminará a edição e fechará todos os ficheiros, escrevendo a última versão do ficheiro no disco. O comando «H» («Home») gravará o ficheiro no estado em que se encontra e voltará ao modo de edição. Isto é particularmente útil quando o ficheiro em edição é extenso e necessita de muitas alterações. A gravação de um ficheiro a intervalos regulares é uma boa garantia contra a possibilidade de todo o ficheiro ser perdido devido a falta de energia ou deficiência do equipamento.

Os ficheiros podem ser removidos para o buffer de edição recorrendo a

```
Rnomeficheiro
```

que lerá um ficheiro para o buffer de edição à frente da posição actual do CP. A única coisa a verificar é se o ficheiro a ler tem a extensão .LIB, apesar de esta poder ser omitida do comando.

Do mesmo modo, o comando

```
nX
```

transferirá n linhas de texto a partir de CP para fora do buffer de edição, passando-as para um outro ficheiro que receberá o nome

```
X$$$$$$$.LIB
```

Este ficheiro fica assim à disposição do utilizador, que poderá alterar o seu nome conforme desejar.

Se o leitor dispõe da versão 3.1 de CP/M, poderá recorrer a uma outra tarefa útil de ED: a criação de um ficheiro PROFILE.SUB, que é o ficheiro procurado quando se inicializa o CP/M. Se este ficheiro não for encontrado, é impresso o habitual pedido de entradas

```
A>
```


No entanto, se existir um ficheiro PROFILE.SUB, o CP/M executa-o e o utilizador pode passar imediatamente a qualquer rotina por si criada. Por exemplo, se definirmos um PROFILE.SUB do seguinte tipo:

```
ED PROFILE.SUB

NEW FILE
     : *I
    1:  MBASIC DEMPROG
    2: ^Z
     : *E
A>
```

O sistema carregará imediatamente MBASIC ao ser inicializado. Em seguida, será carregado e executado o programa designado «DEMOPROG». Isto é apenas um exemplo: PROFILE.SUB pode consistir em várias linhas de programa se assim se desejar.

Do mesmo modo, podemos utilizar ED para criar ficheiros .SUB que contenham um conjunto de instruções usados constantemente. Por exemplo, é muitas vezes necessário construir uma sequência complicada de instruções; os comandos em causa podem ser colocados num ficheiro com a extensão .SUB, e em seguida o comando SUBMIT, seguido do nome do ficheiro, que provocará a execução de todas as instruções contidas nesse ficheiro.

Um exemplo simples disto é o arranque do «package» WORDSTAR no conhecido microcomputador SIRIUS. Esta máquina possui uma série de teclas programáveis, muito úteis quando se usa WORDSTAR. Estas são inicializadas por um programa chamado WS.KEY. Pode-se, portanto, criar um ficheiro designado por EDIT.SUB que contém as duas instruções SET WS.KEY e WS usando o editor:

```
ED EDIT.SUB

NEW FILE
     : *I
    1:  SET WS.KEY
```

```
2:  WS
3:  ^Z
 : *E
```

Sempre que for dada a instrução

```
A>SUBMIT EDIT
```

o computador passa a constituir um tratamento de texto dedicado, desde que evidentemente os ficheiros chamados WS.KEY, WS.CMD e SUBMIT.CMD estejam presentes no disco em uso.

Se estivermos a utilizar a versão 3.1 de CP/M, podemos renomear EDIT.SUB como PROFILE.SUB, de tal modo que WORDSTAR seja chamado assim que o sistema é inicializado, ficando pronto a ser usado sem qualquer intervenção do operador.

Vejamos agora o conjunto completo de comandos ED:

nA	Coloca n linhas do ficheiro de texto original a seguir ao último caracter no buffer do editor
nW	Escreve n linhas do buffer do editor num ficheiro temporário
E	Termina ED. O conteúdo do buffer do editor é copiado para um ficheiro temporário, recebendo os ficheiros novos nomes
H	Grava o ficheiro tal como se encontra. O buffer do editor é esvaziado, e o ficheiro temporário passa a ser o novo ficheiro-fonte. É criado um novo ficheiro temporário
O	Reinicia a edição, usando o ficheiro original
Q	Abandona o editor — sai para CP/M
rnomeficheiro	Lê o ficheiro <nomeficheiro> para o buffer do editor
nX	Transfere n linhas de texto do buffer do editor para o ficheiro temporário chamado X$$$$$$$.LIB

nL	Desloca o indicador n linhas para trás ou para a frente. Se n for zero, o indicador é passado para o início da linha actual
nT	Imprime n linhas de texto — a linha actual mais (n-1) linhas. Imprime a linha anterior se n for negativo. Se n for zero imprime a linha actual a partir da posição do indicador até ao final da linha
OTT	Imprime a linha actual, qualquer que seja a posição do indicador no interior da linha
B	Desloca o indicador para o início do buffer do editor. Se for prefixado do sinal «-», desloca o indicador para o final do buffer
nC	Desloca o indicador n caracteres para a frente ou para trás a partir da posição actual do indicador
n:	Desloca o indicador, tal como o comando L, e imprime a linha actual
:n	Define a extensão de texto entre a linha actual e a linha de número n
nD	Elimina n caracteres a seguir (n positivo) ou antes (n negativo) do indicador
nK	Elimina n linhas a seguir (n positivo) ou antes (n negativo) do indicador
nfcadeia	Descobre a enésima ocorrência da cadeia no texto. O indicador é deixado após o último caracter da cadeia encontrada
nscadeiaA^ZcadeiaB	Substitui a cadeiaA pela cadeiaB n vezes
icadeia	Insere a cadeia à frente da posição actual do indicador
jcadeiaA^ZcadeiaB^ZcadeiaC	Substitui o texto entre a cadeiaA e a cadeiaC pela cadeiaB
nmcadeia	Onde cadeia é uma sequência de comandos e não de texto. Se n for zero ou 1 o comando em cadeia é executado repetidamente até ser atingido o final do

	buffer. Se n for maior do que 1, o comando será executado n vezes
nncadeia	Procuram a cadeia especificado n vezes. Preenche continuamente o buffer a partir do ficheiro fonte até terem sido encontradas as n cadeias ou todo o ficheiro ter sido investigado

Enquanto se executa a edição do texto, o ficheiro temporário recebe a extensão .$$$. Depois de a adição ter sido terminada é retido o ficheiro de texto original, não editado, mas com a extensão .BAK; o ficheiro editado passa a ser indicado pelo nome original do ficheiro. Isto significa que no caso de o ficheiro-fonte ser designado TESTE.BAS, durante a execução passará a chamar-se TESTE.$$$. Depois de a sessão de edição ter sido terminada, o novo ficheiro, já editado, designar-se-á TESTE.BAS e o ficheiro original terá a designação TESTE.BAK.

7

USO DE MBASIC

A versão mais vulgar da popular linguagem BASIC para uso em sistemas CP/M é a designada por MBASIC; trata-se de um compilador de que a Microsoft Inc. posui os direitos. A versão usada sob CP/M-86 é conhecida por BASIC-86, idêntica à MBASIC. Trata-se de uma linguagem de programação interactiva, desenvolvida inicialmente para ensino dos princípios da programação em finais dos anos 60. Desde então tem sido objecto de muitas variações, tendo cada fabricante de computadores produzido versões próprias da linguagem.

Felizmente, dispomos na versão da Microsoft de algo que se aproxima razoavelmente de uma norma desta linguagem. A um nível simples a BASIC da Microsoft é fácil de usar e, no seu estado de desenvolvimento actual, constitui um instrumento de programação bastante potente e sofisticado.

Qualquer pessoa que encontra a MBASIC depois de utilizar uma versão da BASIC num minicomputador ou num computador de grandes dimensões verifica que aquela versão dispõe de um certo número de facilidades que não existem nas máquinas maiores. Uma característica que lhe falta, no entanto, é o conjunto de comandos que manipulam matrizes. Um extra que não existe por outro lado nas versões da BASIC para minicomputador é o facto de os «arrays» poderem ser numéricos ou alfanuméricos, de tal modo que podemos declarar:

```
10 DIM A(10,20),A$(10,20)
```

ambas as declarações tendo significados semelhantes; teremos primeiro uma tabela de números designada por «A» com 10 linhas e 20 colunas, e depois uma tabela de cadeias de caracteres (por exemplo nomes) com 10 linhas e 20 colunas. Isto

resolve uma incoerência que existe nas versões mais vulgares da BASIC para minicomputadores.

Enquanto estamos a tratar o tópico das cadeias alfanuméricas, convirá notar ainda que dispomos de uma forma mais simples de referenciar subcadeias usando as instruções LEFT$, RIGHT$ e MID$. Estas foram de facto usadas pela primeira vez numa versão da BASIC fornecida no computador DEC 10 vários anos antes de os microprocessadores terem aparecido no mercado.

```
LEFT$(A$,K) - especifica os primeiros K caracteres da
              cadeia A$

RIGHT$(A$,K) - especifica os últimos K caracteres da cadeia
               A$

MID$(A$,K,L) - especifica a subcadeia de L caracteres come-
               çando no caracter de ordem K da cadeia A$
```

As variáveis MBASIC podem ter nomes contendo até 40 caracteres, e podem ser definidas como tendo um rigor simples, duplo, ou ser de tipo inteiro. Esta especificação faz-se indicando um «!», «#» ou «%» respectivamente, como último caracter do nome.

Se tivermos um ficheiro de comando MBASIC, MBASIC.COM ou BASIC86.CMD, é apenas necessário escrever:

```
A>MBASIC
```

ou

```
A>BASIC86
```

e o CP/M carregará o interpretador MBASIC em memória, respondendo com uma mensagem como

```
BASIC-86 Version 5.22
[Versão CP/M-86]
Copyright 1977-1982 (C) by Microsoft
Created: 5-Mar-82
52294 Bytes free
Ok
```


Neste ponto, podemos introduzir na máquina um programa BASIC através do teclado. A verificação sintática não será feita enquanto o programa não for executado. Assim, se escrevermos um programa contendo muitos descobrimentos de um qualquer algoritmo, devemos ter o cuidado de os testar a todos; senão acabaremos por percorrer caminho desconhecido e sermos vítimas de desastres!

A MBASIC permite-nos incorporar alguma estruturação nos nossos programas através do recurso a ciclos do tipo WHILE, além dos FOR...NEXT. Por outro lado, permite-nos usar IF...THEN...ELSE e várias instruções em cada linha.

Para ilustrarmos a forma como a MBASIC utiliza o ciclo FOR...NEXT, podemos executar o programa seguinte:

```
10 INPUT I,F,S
20 FOR J=I TO F STEP S
30 PRINT J
40 NEXT J
50 PRINT J
```

Se os valores inicial, final e de passo da variável de controlo de ciclo foram 1, 10, 1, depois 1, 1, 1 e finalmente 1, 0, 1, verificaremos que não só a variável (J neste caso) sairá do ciclo com um valor superior numa unidade ao da variável F nos dois primeiros casos, como ainda terminará com um valor unitário no terceiro caso. Este último facto indica-nos que o ciclo não foi sequer executado uma vez. Vale a pena experimentar este pequeno programa em diversos microsistemas, observando as variações nos resultados obtidos.

A instrução WHILE pode ser facilmente explicado usando um programa deste tipo:

```
10 INPUT NÚMERO
20 WHILE NÚMERO<>0
30 TOTAL=TOTAL + NÚMERO
40 INPUT NÚMERO
50 WEND
60 PRINT TOTAL
```

Este programa percorre continuamente o ciclo entre as linhas 20 e 50 enquanto a variável designada NÚMERO não for igual

a zero. Quando se introduz um zero, o programa termina na linha 60. Se omitirmos a primeira linha do programa obteremos

```
20 WHILE NÚMERO<>0
30 TOTAL=TOTAL + NÚMERO
40 INPUT NÚMERO
50 WEND
60 PRINT TOTAL
```

e verificaremos que o ciclo não é executado porque logo de início a MBASIC atribui a todas as variáveis o valor zero. Demonstra-se assim que, a menos que o valor da variável de controlo não seja nulo, o ciclo não é executado.

Apresentamos em seguida, um programa muito simples que soma uma série de números, terminando as entradas com um valor nulo, e que imprime o total e o valor médio. Usámos no programa instruções GOTO convencionais.

```
10 SUM=0:NUMBER%=0
20 INPUT ''ESCREVA UM NÚMERO'';NUMBER
30 IF NUMBER=0 THEN 70
40 SUM=SUM+NUMBER
50 NUMBER%=NUMBER%+1
60 GOTO 20
70 AVERAGE=SUM/NUMBER%
80 PRINT ''O TOTAL É '';SUM
90 PRINT ''A MÉDIA É ''; AVERAGE
100 END
```

Note-se do uso da variável inteira NUMBER% para contar a quantidade de números introduzidos, que será certamente inteira. O mesmo não acontece com a variável NUMBER, o valor que é somado.

Usando as instruções WHILE e WEND este programa transforma-se no seguinte:

```
10 SUM=0:NUMBER%=0
20 INPUT ''ESCREVA UM NÚMERO'';NUMBER
30 WHILE NUMBER<>0
40   SUM=SUM+NUMBER
50   NUMBER%=NUMBER%+1
60   INPUT ''OUTRO NÚMERO ''; NUMBER
```



```
70 WEND
80 AVERAGE=SUM/NUMBER%
90 PRINT ''O TOTAL É '';SUM
100 PINT ''A MÉDIA É ''; AVERAGE
110 END
```

Note-se que é bastante mais fácil seguir a lógica da segunda versão do programa.

A instrução de tipo IF...THEN...ELSE é muito útil porque ajuda a reduzir a quantidade de programação tipo «spaghetti» que a BASIC tende a encorajar. Esta instrução, juntamente com as WHILE e WEND, ajuda a estruturar melhor os programas. Podemos escrever programas de maneira bastante económica, por exemplo

```
10 INPUT X
20 IF X>100 THEN PRINT ''GRANDE'' ELSE PRINT ''PEQUENO''
30 END
```

A possibilidade de escrever mais de uma instrução em cada linha resulta numa programação económica dado que se escrevermos

```
10 INPUT X
20 IF X<100 THEN PRINT ''MUITO PEQUENO'':GOTO 10
30 PRINT X
40 END
```

isto significa que, no caso de o valor de X ser inferior a 100 não só será impressa a mensagem "MUITO PEQUENO" como ainda se voltará à linha 10 para pedir nova entrada. Se o valor de X for igual ou maior do que 100, é executada a linha 30.

Uma característica muito útil da MBASIC é que é possível a alteração de linhas. Por exemplo, se uma linha requer alteração, passamos ao modo EDIT escrevendo

```
EDIT 200
```

o que imprime o número da linha. Podemos então realizar a alteração. Se a linha 200 for

```
200 INPUT ''ESCREVA UM NÚMERO'',N
```

e quisermos passar a vírgula a ponto e vírgula, escrevemos «S,» obrigando o editor a procurar a vírgula:

```
A>EDIT 200
200 (Escrevemos ''S,'') INPUT ''ESCREVA UM NÚMERO''
```

e o cursor será colocado junto ao caracter a alterar. Escrevendo D será apagado o caracter imediatamente a seguir ao cursor, e veremos

```
200 INPUT ''ESCREVA UM NÚMERO''\,\
```

onde o traço oblíquo delimita o caracter eliminado. Escrevemos em seguida I, para introduzir o novo caracter, escrevemos este (o «;»), e carregamos finalmente em RETURN. Obteremos:

```
200 INPUT ''ESCREVA UM NÚMERO''\,\;N
```

Note-se que nem o D nem o I são impressos. Se escrevermos em seguida

```
A>LIST 200
```

obteremos

```
200 INPUT ''ESCREVA UM NÚMERO'';N
```

que é exactamente o que queríamos.

Se escrevermos um número antes do comando D, eliminaremos esse número de caracteres a partir da posição actual do cursor. Depois do comando I podemos escrever tantos caracteres quantos quisermos; todos eles serão inseridos na posição actual do cursor. Carregando na tecla RETURN introduzimos as alterações e saímos do editor. Vejamos alguns outros comandos:

H elimina todos os caracteres que se encontram à direita do cursor. Note-se que deslocamos o cursor para a direita carregando na barra de espaços e para a esquerda carregando na tecla de RUBOUT.

K elimina todos os caracteres até uma posição especificada. Isto significa que no caso de termos:

```
200 INPUT (obtemos isto escrevendo EDIT 200
          e carregando 5 vezes na tecla RETURN)
```

a instrução 3KN eliminará tudo até à vírgula e colocará o cursor na terceira ocorrência da letra N, como se mostra:

```
A>EDIT 200
200 INPUT (escrevemos ''3KN'')\''ESCREVA UM NÚMERO,''\
```

Carregando então na tecla RETURN obtemos uma nova linha 200 com a seguinte aparência

```
200 INPUT N
```

C altera o caracter que se segue ao cursor, substituindo-o pelo caracter indicado imediatamente a C. Para repetir o comando para vários caracteres prefixamos o C do número de caracteres a alterar.

L lista o que resta da linha e reenvia para o início desta, deixando-nos no modo EDIT. Isto é particularmente útil quando se pretendem fazer várias alterações numa mesma linha.

A permite-nos abandonar as alterações feitas até ao momento e voltar ao princípio. É muito útil quando pretendemos recomeçar a edição da linha.

Q abandona o trabalho sem manter qualquer das alterações feitas entretanto.

Finalmente, convirá notar uma característica muito útil e nem sempre bem documentada do editor. O leitor talvez tenha notado que EDIT imprime sempre o número de linha, esperando depois que a alteremos. Mas suponhamos que perdemos muito tempo na escrita de uma linha complicada, e que depois verificamos que ela deve ser incluída noutro ponto do programa e não onde se encontra. EDIT permite-nos copiar a linha para outro número de linha. Suponhamos que a linha é:

```
40 X=SQR(SIN(LOG(I))^2)+EXP(COS(I))^.4
```

e que queremos esta mesma instrução na linha 100. Começamos por escrever

```
EDIT 40
```

e obteremos

```
40
```

como seria de esperar. Em seguida, carregamos em E seguido de CONTROL e A (simultaneamente), e obteremos um pedido de entrada indicado pelo caracter !. Escrevemos I, indicando que queremos introduzir texto, e o novo número de linha. Passamos assim a dispor de duas linhas contendo exactamente as mesmas instruções mas com diferentes números de linha. A sequência será a seguinte:

```
A>EDIT 40
40    (Escrevemos ''E'', seguido de ''RETURN'')
      (Carregamos em CONTROL-A)
! 100 (Escrevemos ''I'' seguido de ''100'' e ''RETURN'')
100 X=SQR(SIN(LOG(I))^2)+EXP(COS(I))^.4
```

Se escrevermos a instrução «LIST» verificamos que as instruções são agora:

```
40  X=SQR(SIN(LOG(I))^2)+EXP(COS(I))^.4
100 X=SQR(SIN(LOG(I))^2)+EXP(COS(I))^.4
```

Se um programa requer uma paragem por qualquer razão, por exemplo, ter entrado num ciclo infinito, bastará carregar em CONTROL e C simultaneamente para o parar. Se quisermos que o programa continue, o comando CONT permitirá continuar a execução a partir do ponto onde foi interrompida. Se alterarmos o programa de algum modo depois de o pararmos, deixará de ser possível continuá-lo, e obteremos a mensagem «Can't Continue» no visor. Se quisermos continuar mesmo nesse momento podemos, no entanto, escrever

```
GOTO nnnn
```

onde nnnn é um número de linha. Recomeçamos assim a execução a partir do número de linha que nos parece mais aconselhável. Não existe aqui qualquer equivalente da instrução RUN nnnn disponível noutras versões da BASIC.

O tratamento de ficheiros em MBASIC deve ser bem estudado, especialmente no caso de se estar habituado ao uso de ficheiros noutras versões da linguagem. São suportados tanto os ficheiros de acesso directo como os de acesso sequencial, mas são tratados de modos bastante diferentes. Antes do mais, é necessário abrir os ficheiros seriais em modo Input (leitura) ou Output (escrita). É preciso atribuir um número de canal em ficheiro que é aberto, o qual deve ser claramente especificado. Usaremos, portanto, um comando do seguinte tipo:

```
10 OPEN ''O'',1,''FICHEIRO''
```

quando queremos abrir um ficheiro em escrita. Este comando abrirá o ficheiro designado por «FICHEIRO» em modo Output e, se já existir um ficheiro com este nome, elimina-o automaticamente. Para escrevermos algo num ficheiro aberto em modo output, utilizamos a instrução PRINT, como se verá em seguida.

Se escrevermos

```
10 OPEN ''I'',1,''FICHEIRO''
```

o ficheiro chamado "FICHEIRO", que deve obrigatoriamente existir, é aberto de modo a permitir-nos ler os dados que contém. Estaremos de facto a substituir o teclado, que é o dispositivo habitual de comunicação com o microprocessador, pelo ficheiro; e usamos para tal as mesmas instruções INPUT e PRINT. A única diferença é que o canal deve ser especificado depois de INPUT ou PRINT. Isto significa que se escrevermos

```
 5 INPUT A$
10 OPEN ''O'',1,''FICHEIRO''
20 PRINT#1,A$ (imprimir no ficheiro, não no visor)
```

imprimiremos os caracteres guardados na variável A$ no ficheiro associado ao canal 1, designado por «FICHEIRO».

Se escrevermos:

```
10 OPEN ''I'',1,''FICHEIRO''
20 INPUT#1,A$ (obter dados do ficheiro e não do teclado)
30 PRINT A$
```

leremos os caracteres contidos no ficheiro para a variável A$, e imprimimo-los em seguida no visor. Temos de começar agora a ter algum cuidado.

A escrita dos dados contidos numa variável para um ficheiro e a sua leitura a partir deste é bastante fácil e não tende a produzir erros inesperados. No entanto, quando se trata de escrever no ficheiro os valores contidos em diversas variáveis ou de ler para estas os valores já existentes em ficheiro, podemos enfrentar problemas. Consideremos o seguinte programa, que parece bastante inocente:

```
10 OPEN ''O'',1,''DEMOFILE''
15 FOR I=1 TO 3
20 INPUT A$,B$,C$
30 PRINT#1,A$,B$,C$
40 NEXT I
50 CLOSE#1
60 OPEN ''I'',1,''DEMOFILE''
70 FOR I=1 TO 3
80 INPUT#1,A$,B$,C$
90 PRINT A$,B$,C$
100 NEXT I
110 CLOSE
```

Se executarmos o programa acima, a saída não será exactamente a esperada. De facto, obtemos os dados introduzidos, mas seguidos da mensagem

```
Input past end in 80
```

Se investigarmos melhor o que se passa, escrevendo

```
PRINT A$
```

verificaremos que esta variável contém não uma cadeia de caracteres, mas três! De facto, A$ contém três cadeias, e o mesmo acontece a B$ e a C$. O programa tenta, portanto, ler os dados para além de um ponto onde já não existe qualquer valor no ficheiro. O que aconteceu foi que as instruções INPUT e PRINT, quando referidas a ficheiros são equivalentes às usadas para o teclado ou o visor. Nestas condições PRINT# imprime dados no ficheiro exactamente do mesmo modo que o faria no visor.

Isto significa que no caso de os dados impressos estarem separados por vírgulas, como acontece no nosso programa, são impressos com espaços separando as cadeias. Quando lemos os dados do ficheiro, o sistema procura um delimitador, que pode ser uma vírgula, um ponto e vírgula, ou um retorno de linha. Encontra o retorno de linha que usámos para cada conjunto de entradas originalmente feitas no teclado, e portanto atribui à primeira variável tudo o que existe até esse delimitador. Isto tem como consequência a atribuição do triplo de informação às variáveis A$, B$ e C$.

Os dados do ficheiro teriam a seguinte aparência:

```
TOM          DICK          HARRY
FRED         GEORGE        JIM
RON          TIM           MARY
```

sem quaisquer vírgulas.

Este problema pode ser resolvido escrevendo os delimitadores, neste caso vírgulas, à medida que introduzimos os dados. Isto significa que o nosso progrma terá de ser corrigido do seguinte modo:

```
10 OPEN ''O'',1,''DEMOFILE''
15 FOR I=1 TO 3
20 INPUT A$,B$,C$
30 PRINT#1,A$;'','';B$;'','';C$
40 NEXT I
50 CLOSE#1
60 OPEN ''I'',1,''DEMOFILE''
70 FOR I=1 TO 3
80 INPUT#1,A$,B$,C$
90 PRINT A$,B$,C$
100 NEXT I
110 CLOSE
```

Note-se a pequena, mas muito importante, alteração feita na linha 30. Os dados em ficheiro terão agora a aparência

```
RON,TOM,DAN
FRED,MEG,HARRY
DICK,BILL,NORMAN
```

Notem-se as vírgulas.

Se escrevermos dados numéricos, ao contrário de caracteres, o programa será o seguinte:

```
10 OPEN ''O'',1,''DEMOFILE''
15 FOR I=1 TO 3
20 INPUT A,B,C
30 PRINT#1,A,B,C
40 NEXT I
50 CLOSE#1
60 OPEN ''I'',1,''DEMOFILE''
70 FOR I=1 TO 3
80 INPUT#1,A,B,C
90 PRINT A,B,C
100 NEXT I
110 CLOSE
```

Note-se que na linha 30 os campos são limitados por vírgulas, e portanto os dados serão guardados no ficheiro sob a forma:

```
 34          45          32
 32          21          11
 67          68          69
```

tal como seriam impressos no visor. Se substituirmos as vírgulas da linha 30 por ponto e vírgula, os dados serão guardados assim:

```
44  55  66
66  77  88
77  88  99
```

Nestas condições, a segunda versão ocupa menos espaço do que a primeira. O leitor notou provavelmente que não há qualquer necessidade de escrever especificamente vírgulas entre os nomes das variáveis no caso dos dados numéricos, se bem que possamos incluí-los. Mas o sistema necessita de saber onde começa e termina uma cadeia de caracteres, dado que estas tendem a ter comprimentos diferentes; felizmente, os números são guardados sempre no mesmo formato, não necessitando portanto de delimitadores.

O conteúdo destes ficheiros foi impresso usando o comando TYPE de CP/M, não do BASIC como é evidente, mas depois de o comando ter voltado ao sistema operativo.

Se estivermos a ler um ficheiro sequencial e não tivermos nenhuma ideia do número de registos que contém, será necessário dispormos de um comando que pare a leitura quando for encontrado o indicador de final de ficheiro. É isto o habitual em todas as versões da BASIC. Mas é necessário cuidado com o ponto onde utilizamos a instrução. É habitual ordenar os comandos de tal modo que primeiro se abre o ficheiro, se lê um registo, e no caso de este conter o indicador de final de ficheiro se pára a leitura. Se não encontrar este indicador, volta atrás e lê o registo seguinte, e assim por diante. No entanto, a BASIC Microsoft não funciona deste modo. Um programa que leia registos até ao final do ficheiro terá a seguinte aparência:

```
10 OPEN ''I'',1,''DEMOFILE''
20 IF EOF(1) THEN 60
30 INPUT#1,A,B,C
40 PRINT A,B,C
50 GOTO 20
60 CLOSE
```

Notemos que o teste do final de ficheiro é realizado na linha 20, sendo a esta que voltamos de cada vez que o ciclo é percorrido. Por outras palavras, estamos a dizer: «Se o próximo caracter lido do ficheiro for o indicador de final, então...»

A BASIC Microsoft permite-nos criar e usar ficheiros de acesso directo. Um ficheiro de acesso directo, ou aleatório, é criado por um comando semelhante ao que já utilizámos para a abertura de um ficheiro sequencial. Nestas condições

```
10 OPEN ''R'',1,''RFILE'',48
```

abrirá um ficheiro chamado RFILE em acesso directo, com um comprimento de registo de 48 bytes. Se este comprimento não for especificado, o sistema considerará automaticamente um comprimento de 128 bytes. Não esqueçamos que um ficheiro aleatório é aberto tanto para leitura como para escrita, dado que podemos aceder qualquer registo directamente e actuar de qualquer modo sobre o espaço nele reservado aos dados, escrevendo ou lendo. Todas as leituras e escritas num ficheiro alea-

tório são executadas através de um buffer, com o comprimento total de um registo, que é definido por uma instrução como

```
20 FIELD#1,38 AS A$,2 AS B$,8 AS C$
```

onde é especificado o número de bytes atribuído a cada variável de tal modo que no total dêem o comprimento do registo tal como foi indicado na instrução OPEN.

Todos os dados escritos num ficheiro aleatório devem encontrar-se em formato alfanumérico. Quando guardados em disco, os dados são de facto convertidos do formato de cadeia para um formato binário compactado, a fim de poupar espaço.

Antes de um registo ser escrito no ficheiro, será necessário colocar o conteúdo dos seus campos no buffer respectivo. Isto é feito usando os comandos LSET e RSET. Ambos justificam a cadeia de caracteres, o primeiro à esquerda e o segundo à direita, preenchendo o espaço livre com espaços. As variáveis numéricas são convertidas para cadeias usando MKI$, MKS$ e MKD$, conforme devem ser guardadas como inteiros, números de rigor simples ou rigor duplo. Os inteiros são guardados sob a forma de uma cadeia de dois bytes. Um número de rigor simples é guardado numa cadeia de quatro bytes, e um número de rigor duplo numa cadeia de oito bytes. Podemos portanto ter um programa do seguinte tipo:

```
10 OPEN ''R'',1,''RFILE'',48
20 FIELD#1,38 AS A$,2 AS B$,8 AS C$
30 INPUT X$,Y,Z
40 LSETA$=X$
50 LSETB$=MKI$(Y)
60 LSETC$=MKD$(Z)
```

A linha 40 justifica à esquerda a variável, atribuindo-lhe os 38 bytes postos à disposição de A$. A linha 50 converte Y num inteiro e justifica à esquerda o seu equivalente em cadeia, atribuindo-lhe os 2 bytes de B$. Finalmente, Z é convertido num número de precisão dupla e o seu equivalente de cadeia é colocado nos 8 bytes atribuídos à variável C$.

O conteúdo do buffer é depois transferido para um determinado registo do ficheiro, usando

```
70 PUT#1,10
```

que coloca a cadeia existente em buffer no registo 10, neste caso.

Para recuperar dados de um determinado registo usamos o inverso de PUT, que é GET, e isto transfere o registo no seu conjunto para o buffer. Teremos em seguida de descodificar os dados numéricos usando CVI, CVS e CVD para converter de novo do formato de cadeia para o de número inteiro, de precisão simples ou dupla. Podemos portanto escrever

```
80 GET#1,10
90 Y=CVI(B$)
100 Z=CVD(C$)
110 PRINT A$,Y,Z
```

Note-se que não temos de descodificar o conteúdo da cadeia A$ porque foi guardada como cadeia. Isto não é verdadeiro para as outras variáveis usadas neste exemplo.

O uso de ficheiros em BASIC Microsoft não é difícil mas é necessário pensar exactamente na forma como os registos se encontram estruturados no interior dos ficheiros e como é usado o espaço em disco. Isto significa, no entanto, que a MBASIC pode ser usada para escrever programas de tipo comercial de boa qualidade.

Estas novas não pretendem ser uma descrição exaustiva da MBASIC; esta tarefa é deixada ao manual da linguagem. São no entanto concebidas para fornecer alguns indicadores a todos aqueles que já conheçam a BASIC e queiram evitar alguns problemas sem terem de ler todo o manual. Um último aspecto é formado pelos úteis comandos que se podem fornecer à MBASIC a partir do CP/M. Trata-se de operações que são invocadas quando a MBASIC é carregada. Por exemplo, se escrevermos

```
A>MBASIC TESTPROG
```

então não só carregaremos em memória o interpretador MBASIC como também o programa em BASIC designado TESTPROG, que será carregado e executado. Podemos igualmente usar comandos mais complicados como

```
A>MBASIC FILEPROG/S:256
```

que não só carrega e executa o programa chamado FILEPROG como ainda altera a dimensão do registo de quaisquer ficheiros aleatórios usados para 256 bytes. Normalmente, a dimensão máxima atribuída automaticamente aos registos é de 128 bytes. O comando acima pode alterar estas dimensões.

Finalmente, enquanto tratamos de ficheiros, vejamos como podemos gravar ficheiros em disco de diferentes maneiras. Se escrevermos simplesmente

```
SAVE ''PROG1''
```

será gravado um ficheiro de nome «PROG1.BAS», em formato binário comprimido, no drive actualmente em uso.

Escreva

```
SAVE ''B:PROG1''
```

e o programa será gravado no drive B.

Grave o programa com

```
SAVE ''PROG1'',A
```

e este será guardado em caracteres ASCII de tal modo que possa ser editado, se necessário, usando um processador de texto como o WORDSTAR ou o próprio ED de CP/M.

Grave o programa com

```
SAVE ''PROG1'',P
```

e este será guardado numa forma codificada, como ficheiro binário, de tal modo que qualquer tentativa de o listar ou alterar não terá êxito. O programa só poderá ser executado. Um programa protegido não pode ser desprotegido. Note-se, a propósito, que

a MBASIC não diferencia entre instruções em maiúsculas ou em minúsculas: a linha

```
10 INPUT X
```

é equivalente a

```
10 input x
```

Mas deve-se ter cuidado com tudo o que estiver entre aspas, incluindo nomes de programas. Se escrevermos a ordem

```
load ''prog1''
```

obteremos a mensagem «program not found» (programa não encontrado), no caso de termos gravado o programa sob o nome «PROG1».

Para passar de MBASIC para CP/M é sempre melhor escrever

```
RESET
```

o que actualiza o directório do disco e fecha todos os ficheiros que possam ter sido deixados abertos. Depois podemos escrever

```
SYSTEM
```

que nos reenvia para CP/M.

8

USO DE UM COMPILADOR

O sistema operativo CP/M permite-nos usar muitas linguagens diferentes para além da MBASIC, que é provavelmente a mais comum. A MBASIC foi concebida, e é muito usada, como linguagem interactiva para permitir aos programadores aprenderem as técnicas de programação. Os comandos e as instruções de programa são executadas quer num modo «imediato», em que o computador é levado a comportar-se como se fosse uma máquina de calcular, ou depois de ter sido dado o comando RUN. Isto é tratado em maior detalhe no capítulo anterior. Uma linguagem compilada, por outro lado, terá os seus programas escritos inicialmente sob a forma de uma cadeia de caracteres, podendo portanto ser criadas por um editor que funciona sob CP/M, por exemplo o seu próprio utilitário ED, o WORDSTAR, o T/MAKER ou qualquer outro editor de texto que produza um grupo de caracteres ASCII. No exemplo apresentado no capítulo 6, usámos ED para criar e modificar subsequentemente um programa CBASIC.

A CBASIC é uma versão da MBASIC capaz de produzir programas que correm mais depressa e de forma mais eficaz do que os mesmos programas escritos em MBASIC. Esta última versão, interactiva, da linguagem BASIC guarda o programa naquilo que se designa pela sua forma «fonte» e de cada vez que o programa é executado torna-se necessário realizar uma conversão do programa da sua forma «fonte» para uma sequência de instruções que possam ser compreendidas e utilizadas pelo processador. Para uma melhor explicação deste processo, o leitor poderá consultar a obra «Mastering Computer Programming» do mesmo autor (MacMillan, 1982), no capítulo 1.

A interpretação de um programa numa forma que a máquina entenda é um processo bastante lento e ocupa muito tempo ao processador. No entanto, se um programa pode ser guardado de tal modo que o seu processo de conversão seja eliminado, pode-se poupar bastante tempo e utilizar de forma mais eficaz o tempo do processador. O compilador CBASIC é um programa que aceita linhas de texto na entrada; analisa em seguida estas linhas em função das regras da CBASIC, que são apresentadas no respectivo manual, e produz um ficheiro que se encontra muito mais próximo do código-máquina compreendido pelo computador. Se o programa executa bem, pode-se eliminar o programa-fonte usando apenas este ficheiro intermédio criado pelo compilador.

A CBASIC consiste em dois programas, um designado CBAS2, ou CBAS86, e um outro chamado CRUN2, ou CRUN86. Dá-se preferência a um ou a outro conforme se está a utilizar um CP/M de oito bits ou de dezasseis bits. O ficheiro de entrada de CBAS2 deve possuir uma extensão .BAS, e depois da compilação é criado um novo ficheiro, designado pelo mesmo nome, mas tendo uma extensão .INT (indicando o facto de se tratar de um ficheiro intermédio). CRUN2 será então utilizado para executar este ficheiro. O ficheiro com a extensão .BAS pode ser guardada noutro disco, passando a trabalhar-se apenas com o ficheiro .INT. Se se pretender introduzir qualquer alteração no programa, será necessário voltar ao ficheiro .BAS. Este será recompilado, passando a nova versão a substituir o anterior ficheiro .INT.

Para podermos ter uma ideia da diferença entre a CBASIC e a BASIC, escreveremos um simples programa de três linhas; primeiramente em BASIC:

```
10 FOR I%=1 TO 50
20 PRINT I%;I%^2
30 NEXT I%
```

Este programa corre evidentemente sem qualquer dificuldade e dando os resultados esperados. Depois gravamos o programa tal como se encontra, mas em formato ASCII — o que significa

que é guardado sob a forma de uma cadeia de caracteres, tal como é escrito no teclado. Conseguimos isto escrevendo

```
SAVE ''PROG'',A
```

Procedendo deste modo teremos de facto gravado o programa com uma extensão .BAS, automática, pelo que se usarmos uma instrução DIR veremos que o sistema chamou ao nosso programa PROG.BAS.

Em seguida apresentamos o programa ao compilador CBASIC, escrevendo:

```
CBAS86 PROG  (Não é necessário empregar aqui a extensão
              .BAS)
```

e obteremos o seguinte:

```
CBASIC/86 COMPILER VER 1.00
COPYRIGHT COMPILER SYSTEMS INC.
    1* 10 FOR I%=1 TO 50
    2* 20 PRINT I%;I%^2
    3* 30 NEXT I%
    4: END
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:          38
DATA STMT AREA:      0
VARIABLE AREA:       8
A>
```

Note-se que o compilador acrescentou uma instrução END depois da última instrução de programa, pelo que devemos ter o cuidado de não colocar a mesma instrução no meio de um programa. END é uma palavra compreendida pelo compilador como significando que foi atingido o final do programa. Como não são detectados quaisquer erros no nosso uso da linguagem, podemos executar o programa compilado escrevendo

```
CRUN86 PROG
```

o programa será executado do mesmo modo que a versão interpretada.

No entanto, podemos modificar o programa retirando os números de linha, usando o editor (ver o capítulo 6), e obteremos um programa com a aparência seguinte:

```
FOR I%=1 TO 50
PRINT I%;I%^2
NEXT I%
```

Isto pode ser compilado:

```
CBASIC86 PROG

CBASIC/86 COMPILER VER 1.00
COPYRIGHT COMPILER SYSTEMS INC.
    1: FOR I%=1 TO 50
    2: PRINT I%;I%^2
    3: NEXT I%
    4: END
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:          38
DATA STMT AREA:      0
VARIABLE AREA:       8
A>
```

e o programa executará tal como anteriormente. Note-se, a propósito, que o compilador fornece os números de linha, que não faziam parte desta versão do nosso programa. Servem apenas para referência.

Como novo exemplo, vejamos uma simples rotina matemática para cálculo de raízes quadradas. Em MBASIC tem a seguinte aparência:

```
10 FOR I=1 TO 4
20 READ X
30 X1=1
40 Y=(X1+X/X1)/2
50 IF X1=Y THEN 100
60 X1=Y
70 GOTO 40
100 PRINT Y
110 NEXT I
120 STOP
130 DATA 10,100,1000,10000
```

este programa produz as seguintes saídas:

```
3.162278
10
31.62278
100
```

Gravamos o programa na sua forma ASCII sob o nome de ROOTS, e saímos em seguida da MBASIC. Voltamos a CP/M, e compilamos o nosso programa satisfatoriamente usando CBAS. Mas quando queremos executar o programa não acontece aparentemente nada. Nenhum resultado é impresso no visor. Torna-se óbvio ao fim de algum tempo que o programa entrou num ciclo infinito, e a habitual combinação CONTROL--C não pára o programa. É necessário reinicializar o computador antes de editar o programa e tentar de novo.

É necessário uma notificação na quinta linha do programa, como se mostra em seguida:

```
CBAS86 ROOTS

CBASIC/86 COMPILER VER 1.00
COPYRIGHT COMPILER SYSTEMS INC.
    1* 10 FOR I=1 TO 4
    2* 20 READ X
    3* 30 X1=1
    4* 40 Y=(X1+X/X1)/2
    5* 50 IF ABS(X1=Y)<.000001 THEN 100
    6* 60 X1=Y
    7* 70 GOTO 40
    8: 100 PRINT Y
    9* 110 NEXT I
   10* 120 STOP
   11* 130 DATA 10,100,1000,10000
   12: END
NO ERRORS DETECTED
CONSTANT AREA:      16
CODE SIZE:          86
DATA STMT AREA:     20
VARIABLE AREA:      32
A>
```

Note-se que as linhas de programa que possuem números de linha redundantes são indicados por um «*».

Se executarmos este programa obteremos:

```
CRUN86 ROOTS
CRUN86 VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
 3.16227766017
 10
 31.6227766017
 100
A>
```

Estamos agora em posição de escrever um simples programa CBASIC desde início, recordando que não existe qualquer forma «directa» de escrever o programa que se assemelhe à usada no caso de um interpretador BASIC. Temos de introduzir as linhas de programa usando o editor, gravar o programa como ficheiro de texto e apresentá-lo em seguida ao compilador CBAS.

Se a passagem pelo compilador não revelar quaisquer erros, podemos executar a versão compilada usando o programa CRUN. Mas antes de o fazermos sejamos um pouco mais aventureiros e escrevamos um programa ligeiramente mais complexo, apenas para termos uma ideia das possibilidades da CBASIC. O programa será o seguinte, em BASIC normal:

```
10 FOR I=1 TO 50 STEP 0.25
20 PRINT I;TAB(8);I^2;TAB(20);SQR(I)
30 NEXT I
```

o que funciona sem quaisquer problemas. Gravamos portanto este programa em formato ASCII, e apresentamo-lo ao compilador CBAS. Obteremos

```
CBAS86 PROG1

CBAS/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1* 10 FOR I=1 TO 50 STEP 0.25
ERROR FE IN LINE    1 AT POSITION      8
    2* 20 PRINT I;TAB(8);I^2;TAB(20);SQR(I)
    3* 30 NEXT I
    4: END
1 ERROR DETECTED
```

```
CONSTANT AREA:      16
CODE SIZE:          66
DATA STMT AREA:      0
VARIABLE AREA:       8
A>
```

Há portanto algo de errado. Se observarmos a lista de mensagens de erro do manual da CBASIC, verificaremos que se trata de um erro «FOR». Isto significa que temos aquilo a que se chama, em CBASIC, uma expressão em «modo misto» — algo com que não temos, geralmente, de nos preocupar em BASIC «normal».

O que acontece de facto é que temos uma instrução na primeira linha do programa — posição 8 refere-se ao facto de haver algo de errado quanto à variável I — um pouco incoerente. A cura consiste em obrigar todas as variáveis a serem do mesmo tipo, porque estamos a exigir que I seja incrementada em passos de 0.25, e não de 1 como anteriormente. I é uma variável de contagem que deve contar não em inteiros mas em números de vírgula flutuante. Temos, portanto, de resolver esta ambiguidade exprimindo as variáveis como decimais. Tudo passa assim a estar certo. Se quiser estudar mais detalhadamente o tópico do armazenamento de variáveis o leitor deverá consultar a obra de Gosling e Laarhoven «Codes for Computers and Microprocessors» (MacMillan, 1980). O resultado desta alteração e da eliminação dos números de linha será:

```
CBAS86 PROG1

CBAS/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1: FOR I=1.00 TO 50.00 STEP 0.25
    2: PRINT I;TAB(8);I^2;TAB(20);SQR(I)
    3: NEXT I
    4: END
CONSTANT AREA:      32
CODE SIZE:          67
DATA STMT AREA:      0
VARIABLE AREA:       8
A>
```


Observemos agora o programa CBASIC criado no capítulo 6 através de ED. Foi gravado sob o nome de DEMO.BAS, e quando o apresentamos a CBAS86 convém recordar que não é necessário preocuparmo-nos com a extensão .BAS. CBAS trata do problema sem o nosso auxílio.

A CBASIC é bastante semelhante, mas não exactamente igual, à MBASIC; é portanto normal supor-se que se pode escrever um programa mais ou menos da mesma maneira em ambas as versões, apenas evitando no primeiro caso os número de linha. Podemos de facto proceder deste modo, desde que tenhamos presentes alguns pequenos problemas relativos ao armazenamento de números.

Tendo já errado uma vez, observamos o manual e verificamos que a instrução que abre um ficheiro é também ligeiramente diferente, o mesmo acontecendo com a instrução que imprime para um ficheiro na medida em que utiliza um ponto e vírgula em vez de uma vírgula depois do número do ficheiro. Criamos portanto o nosso ficheiro de instruções de programa e, tendo-o gravado, apresentamo-lo ao CBAS86 a fim de verificar o que acontece

```
A>CBAS86 DEMO

CBASIC/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1: REM***PROGRAMA DE DEMONSTRAÇÃO CBASIC***
    2: OPEN ''DEMFILE'' AS 1
    3: INPUT ''QUANTOS REGISTOS '';N
    4: FOR I = 1 TO N
    5: INPUT ''ESCREVA UMA FRASE '';A$
    6: PRINT#1;A$
    7: CLOSE 1
    8: END
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:         108
DATA STMT AREA:      0
VARIABLE AREA:      24
A>
```

Tudo parece bem; dir-se-ia que o nosso programa ficou bom logo à primeira vez. Tentamos portanto executá-lo:

```
A>CRUN86 DEMO   (Uma vez mais, CRUN86 considera automatica-
                 mente que o ficheiro tem uma extensão .INT)

CRUN86 VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.

ERROR OE

A>
```

Algo parece ter corrido mal. O programa compilou correctamente, pelo que devemos ter usado instruções correctas, mas não executou devido a um qualquer erro de lógica. Teremos de voltar ao manual e descobrir o que significa ERROR OE.

No manual descobrimos que o erro OE é um erro de abertura de ficheiro, e que significa que tentámos abrir um ficheiro que não existe. CBASIC insiste em que antes de escrevermos num ficheiro novo devemos, antes do mais, criá-lo. A MBASIC é mais liberal neste aspecto, e se o ficheiro não existir cria-o automaticamente. A CBASIC não funciona do mesmo modo — e por outro lado as suas mensagens de erro não são tão claras. Somos sempre forçados a consultar o manual.

Tendo descoberto que necessitamos de criar o ficheiro antes de o tentarmos abrir para saída, voltamos ao editor e inserimos uma nova linha. Depois experimentamos de novo:

```
A>CBAS86 DEMO

CBASIC/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1: REM***PROGRAMA DE DEMONSTRAÇÃO CBASIC***
    2: CREATE ''DEMFILE'' AS 1
    3: OPEN ''DEMFILE'' AS 1
    4: INPUT ''QUANTOS REGISTOS '';N
    5: FOR I = 1 TO N
    6: INPUT ''ESCREVA UMA FRASE '';A$
    7: PRINT#1;A$
    8: CLOSE 1
    9: END
```



```
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:         126
DATA STMT AREA:      0
VARIABLE AREA:      24
A>
```

A forma como utilizámos a linguagem encontra-se novamente correcta, pois no caso contrário o compilador imprimiria uma qualquer mensagem de erro sintático. Voltamos portanto a experimentar o nosso programa:

```
A>CRUN86 DEMO

CRUN/86 VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.

ERROR DF

A>
```

Desta vez, o erro DF indica-nos que tentámos OPEN ou CREATE um ficheiro com um número já activo. Isto significa que para abrirmos um ficheiro é necessário que este já exista, mas que no caso de criarmos o ficheiro deixa de ser necessário abri-lo. É portanto necessário escrever uma parte de programa que verifique se um dado ficheiro já existe ou não. Se não, é criado; se existe, é aberto. Alterámos as primeiras seis linhas do programa a fim de realizarmos estas operações.

Eis a nova tentativa:

```
A>CBAS86 DEMO

CBASIC/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1: REM***PROGRAMA DE DEMONSTRAÇÃO CBASIC***
    2: IF END#1 THEN 5
    3: OPEN ''DEMFILE'' AS 1
    4: GOTO 10
    5: 5
    6: CREATE ''DEMFILE'' AS 1
    7: 10
    8: OPEN ''DEMFILE'' AS 1
```

```
    9: INPUT ''QUANTOS REGISTOS '';N
   10: FOR I = 1 TO N
   11: INPUT ''ESCREVA UMA FRASE '';A$
   12: PRINT#1;A$
   13: CLOSE 1
   14: END
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:         135
DATA STMT AREA:      0
VARIABLE AREA:      24
A>CRUN DEMO

CRUN86 VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
QUANTOS REGISTOS  4

ESCREVA UMA FRASE FRED

ESCREVA UMA FRASE GEORGE
ESCREVA UMA FRASE HENRY

ESCREVA UMA FRASE JOANNA

A>
```

O programa funciona finalmente. Notemos que de cada vez que realizamos uma alteração temos de voltar ao editor a fim de alterar o ficheiro que possui a extensão .BAS. O ficheiro é compilado, sendo depois executado o resultante ficheiro .INT. O esforço vale de facto a pena. Agora resta-nos dar uma melhor aparência ao nosso programa. Voltemos uma última vez ao editor:

```
A>CBAS86 DEMO

CBASIC/86 COMPILER VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
    1: REM***PROGRAMA DE DEMONSTRAÇÃO CBASIC***
    2: IF END#1 THEN 5
    3: OPEN ''DEMFILE'' AS 1
    4: GOTO 10
    5: 5
    6: CREATE ''DEMFILE'' AS 1
```



```
    7: 10
    8: OPEN ''DEMFILE'' AS 1
    9: INPUT ''QUANTOS REGISTOS ?: '';N
   10: FOR I = 1 TO N
   11: PRINT ''ESCREVA UMA FRASE #:'';I;
   12: INPUT A$
   13: PRINT#1;A$
   14: CLOSE 1
   15: END
NO ERRORS DETECTED
CONSTANT AREA:       8
CODE SIZE:         147
DATA STMT AREA:      0
VARIABLE AREA:      24
A>CRUN86 DEMO

CRUN86 VER 1.00
COPYRIGHT 1981 COMPILER SYSTEMS INC.
QUANTOS REGISTOS ?:  5

ESCREVA UMA FRASE #: 1 ? TOMMY

ESCREVA UMA FRASE #: 2 ? HARRY

ESCREVA UMA FRASE #: 3 ? KENNETH

ESCREVA UMA FRASE #: 4 ? JOSEPHINE

ESCREVA UMA FRASE #: 5 ? LARRY

A>
```

Eis-nos finalmente chegados ao final desta sessão de trabalho. Este método de criação, compilação e execução de programas pode ser utilizado para uma grande variedade de linguagens de alto nível, como a FORTRAN, a COBOL, a PL/1 e outras. É apenas necessário que o leitor compre o software adequado a cada uma delas.

9

ASM, O ASSEMBLADOR CP/M, E DDT

Se o leitor quiser verdadeiramente atingir o «coração» do sistema computador terá necessidade de aprender a falar com a máquina, numa linguagem que esta seja capaz de entender.

O uso de linguagens de alto nível como a MBASIC equivale a falar com o computador através de um intérprete, e portanto todas as instruções que usamos num programa demoram muito mais tempo a ser executadas. É por isso que usamos um compilador; ver o que se disse já sobre o assunto no capítulo 8.

Mas mesmo quando usamos uma linguagem de alto nível compilada, não conseguimos controlar alguns dos aspectos do funcionamento do computador, ao nível que podemos designar «de bit». Este é o nível mais baixo de comunicação com o microprocessador, e no passado, quando os computadores estavam ainda na sua infância, era esta de facto a única maneira de os programar. Os programas eram construídos directamente numa codificação binária, ou seja em sequências de zeros e uns. Este facto não é surpreendente dado que é este o único tipo de código que o processador entende. As linguagens de alto nível tiraram muito trabalho aos programadores, mas pelo caminho perdeu-se a possibilidade de controlar muitos pormenores do funcionamento da máquina.

Só usando a programação em linguagem «assembly» podemos, quando necessário, aceder a tudo o que nos interessa no computador.

Um código «assembler» utiliza referências a posições reais no interior da memória do computador, em vez dos nomes de «variáveis» usados, por exemplo, em BASIC. Refere igualmente um certo número de «registos» que o microprocessador utiliza

nas operações aritméticas. As tarefas envolvidas, por exemplo, na execução de simples operações aritméticas devem ser divididas em operações ainda mais elementares. Através de uma programação cuidadosa, podemos assim obter um uso muito mais eficaz da memória disponível. Para uma completa explicação do código assembler do microprocessador Z80 recomenda-se ao leitor que leia o livro de Roger Hutty «Programming in Z80 Assembly Language» (Macmillan, 1984). O conteúdo deste capítulo será apenas uma breve explicação da forma como o CP/M trata o código assembler e de como se escreve e executa um programa em código-máquina sob o sistema CP/M.

A sequência de acontecimentos que conduz à execução de um programa nesta linguagem é a seguinte. Antes do mais, introduz-se na máquina um programa em código-fonte usando um package processador de texto, como o WORDSTAR, ou o próprio utilitário ED do CP/M exactamente da mesma maneira que um programa de linguagem de alto nível concebido para ser compilado.

O programa em causa deve possuir uma extensão .ASM, e é em seguida passado para aquilo que é designado por «código-máquina» usando um programa assemblador, conhecido por ASM.COM. Este produz dois outros ficheiros, ambos com o mesmo nome do original, mas usando as extensões .PRN e .HEX. O primeiro destes é uma cópia do ficheiro-fonte, o ficheiro .ASM original, mas possuindo números de linha e indicando as posições atribuídas às várias instruções .HEX contém as instruções do programa, mas desta vez expressas em forma hexadecimal. É este o ficheiro que deve ser apresentado ao programa LOAD.COM a fim de produzir um programa executável, desta vez com a extensão .COM. É este programa, que se encontra em formato binário, e portanto mais próximo da linguagem compreendida pelo microprocessador, que vem depois a ser executado pelo CP/M.

A fim de compreendermos o que se passa, é necessário descrever ainda um pouco da forma como o microprocessador é construído. O microprocessador Zilog Z80 consiste de nove registos de oito bits e quatro de 16 bits.

Os registos de oito bits são os seguintes:

```
Registo A - o Acumulador.
Registo F - o registo de Flags.
Registos B,C,D,E,H,L - registos secundários ou temporários.
Registo I - registo de vector de interrupção.
```

Os registos B e C, D e E, H e L podem ser combinados em três registos de 16 bits se assim se quiser.

Os registos de 16 bits são:

```
Registo SP - registo indicador de stack
Registo PC - registo contador de programa
Registo IX - registo de indexação X
Registo IY - registo de indexação Y
```

Cada posição de memória utilizado pelo Z80 possui um comprimento de 8 bits (o que equivale a 1 byte). Na programação em assembler usamos a convenção de encerrar um endereço entre parêntesis quando queremos dizer «conteúdo do byte cujo endereço é». Assim, (10) significa: «conteúdo do byte cujo endereço é 10».

Um programa escrito em Assembler do Z80 assume a forma:

```
(1) uma etiqueta seguida de:
(2) um ''código de operação'' seguido de um ou mais operandos
(3) ; seguido de um comentário que é ignorado pelo assem-
    blador.
```

As etiquetas e os comentários são opcionais, mas cada campo deve ser separado do seguinte por um espaço. Uma linha pode consistir simplesmente num comentário. Uma linha típica de código poderá ser, por exemplo,

```
LABEL1:     ADD  A,A          ; A x 2
```

A etiqueta é LABEL1, a instrução obriga a somar o conteúdo do registo A, o Acumulador, ao conteúdo de A, e o comentário é que o efeito disto consiste na multiplicação do conteúdo do Acumulador por 2. Por outras palavras, o conteúdo do Acumulador é somado a si mesmo, sendo o resultado guardado nele novamente.

Apresentamos em seguida um programa simples que servirá para ilustrar como se usa o assemblador; o programa foi tirado do livro de Roger Hutty, página 12, programa 3.1. Introduzimos o programa usando um editor, e gravamo-lo sob o nome de DEMO.ASM. O programa é

```
            ORG 100H
; O PROGRAMA 3.1 MULTIPLICA 2 NÚMEROS POR 4
;
            LD  A,(N1)  ;início do programa principal
            CALL QUAD   ;calcular N1 x 4
            LD  (R1),A  ;R1=N1 x 4
;
            LD  A,(N2)
            CALL QUAD
            LD  (R2),A  ;R2=N2 x 4
            HALT
N1:         DEFB 31
N2:         DEFB 25
R1:         DEFB 0
R2:         DEFB 0
;
; Sub-rotina - multiplica o Acumulador por 4
;
QUAD:       ADD A,A     ; A x 2
            ADD A,A     ; (A x 2) x 2 = A x 4
            RET
```

Note-se como o programa se torna bastante mais legível quando se usam linhas de comentário em branco. É agora requerida alguma explicação sobre o que se está a passar. A primeira instrução — ORG 100H — indica ao programa assemblador que a primeira instrução deve ser guardada na posição 100H, início do «TPA» — ver capítulo 1. O H no fim do número indica ao assemblador que este se encontra escrito em notação hexadecimal. A letra B a seguir a um número indica que este se encontra em forma binária. O ou Q indica um número em octal, e D, ou a ausência de qualquer letra, indica um número decimal.

A linha que diz «LD A,(N1)» é a instrução que manda carregar o conteúdo da posição N1 para o Acumulador. O «DEFB 31» com a etiqueta N1 é uma indicação de que a posição conhe-

cida por N1, o seu endereço exacto, será atribuído mais tarde pelo assemblador e deverá conter o número 31. DEFB significa de facto «DEFinir o valor de um Byte». Os dois números são chamados N1 e N2, e os resultados da multiplicação por quatro são guardados em posições, inicialmente passadas a zero, chamadas R1 e R2. A subrotina etiquetada QUAD é chamada para realizar a multiplicação no Acumulador, onde se realizam todas as operações aritméticas, e em seguida o conteúdo do Acumulador é guardado nos locais apropriados ao terminar a chamada QUAD. RET é a instrução que devolve o controlo ao programa principal a partir da subrotina, e HALT é a instrução que obriga à paragem do programa.

Podemos ver qual é o conteúdo da memória quando DEMO.ASM reside nesta usando o utilitário DUMP. Este indica-nos exactamente os caracteres que se encontram guardados em cada posição específica da memória:

```
A>DUMP DEMO.ASM
0000 20 20 20 20 20 20 20 20 20 20 20 4F 52 47 20 31
0010 30 30 48 0D 0A 3B 20 50 52 4F 47 52 41 4D 20 33
0020 2E 31 20 4D 55 4C 54 49 50 4C 49 45 53 20 54 57
0030 4F 20 4E 55 4D 42 45 52 53 20 42 59 20 34 0D 0A
0040 3B 0D 0A 20 20 20 20 20 20 20 20 20 20 4C 44 20
0050 41 2C 28 4E 31 29 20 3B 53 54 41 52 54 20 4F 46
0060 ........
```

Se traduzirmos a porção anterior do ficheiro DEMO.ASM da forma hexadecimal para caracteres ASCII veremos que o texto que constitui o ficheiro se encontra guardado em memória a partir da posição 0000. O caracter mais comum é 20H, que corresponde a um espaço; 4FH é «O», 52H é «R», 47H é «G», 31H é «1», 30H é «0», e assim por diante. ODH é o caracter de retorno de linha, e OAH é o de mudança de linha, pelo que encontramos de facto na memória tudo aquilo que escrevemos anteriormente. Um «dump» deste tipo é muitas vezes o último recurso do programador cujo programa não funciona, e existem alguns profissionais que são capazes de o ler exactamente da mesma maneira que o leitor e eu lemos um texto normal...

Apresentemos agora este ficheiro do texto, dado que é isso mesmo, ao ASM.COM do seguinte modo:

```
A>ASM DEMO    Não é necessária a extensão .ASM
ASM Z80 ASSEMBLER V 3.0 A

7E42 bytes free
No errors
A>
```

Não tendo o processo da assemblagem encontrado qualquer erro (como indica a expressão «No errors»), passamos a dispor de um ficheiro chamado DEMO.PRN e de um outro designado DEMO.HEX. Podemos descobrir o que contêm pedindo ao CP/M que os imprima:

```
A>TYPE DEMO.PRN
0100                    ORG 100H
                ; O PROGRAMA 3.1 MULTIPLICA 2 NÚMEROS POR 4
                ;
0100 3A1301             LD  A,(N1)     ;início do prog. princ.
0103 CD1701             CALL QUAD      ;cálculo N1 x 4
0106 321501             LD  (R1),A     ;R1=N1 x 4
                ;
0109 3A1401             LD  A,(N2)
010C CD1701             CALL QUAD
010F 321601             LD  (R2),A     ;R2=N2 x 4
0112 76                 HALT
0113 1F     N1:         DEFB 31
0114 19     N2:         DEFB 25
0115 00     R1:         DEFB 0
0116 00     R2:         DEFB 0
                ;
                ; Sub-rotina - multiplica o acumulador por 4
                ;
0117 87       QUAD:     ADD A,A ; A x 2
0118 87                 ADD A,A ; (A x 2) x 2 = A x 4
0119 C9                 RET


0113 N1    0114 N2      0117 QUAD     0115 R1   0116 R2
```

A primeira das duas colunas do lado esquerdo é uma lista de endereços, começando pelo endereço 100 (em hexadecimal).

A segunda coluna lista o conteúdo dos endereços. Isto significa que o endereço 100 contém o caracter «3A», o endereço 101 contém «13», e 102 contém «01». O leitor poderá notar que 101 e 102 contêm o endereço do número a carregar no acumulador; mas, como sempre acontece, este número está escrito de trás para a frente. O byte de valor mais significativo (13) encontra-se antes do de valor menos significativo (01), dando o endereço 0113. E vemos que a posição 113 contém o número hexadecimal 1F, que é 31 em decimal, enquanto a posição 115 contém zero, ou seja exactamente o que desejamos.

A etiqueta QUAD está guardada na posição 117, de tal modo que quando o programa salta para a subrotina conhece a posição da primeira instrução desta. Isto acontece porque as posições 103 e 10C contêm o código CD seguido do endereço para onde deve ser executado o salto, codificado sob a forma 1701 — posição 0117, onde se encontra a etiqueta QUAD.

A última linha impressa é a tabela de símbolos que nos dá os endereços de cada uma das etiquetas usadas no programa.

Se pedirmos o conteúdo de DEMO.HEX obteremos

```
A>TYPE DEMO.HEX
:100100003A1301CD17013215013A1401CD1701320E
:0A0110001601761F1900008787C949
:0000000000
```

o que, como o leitor poderá verificar, corresponde ao total da informação sobre o programa: o seu endereço inicial e o conteúdo codificado de todas as posições.

Finalmente, o ficheiro .HEX deve ser convertido num ficheiro binário com uma extensão .COM, o que é feito recorrendo ao comando LOAD como se segue:

```
A>LOAD DEMO  Note-se que não é necessária a extensão.
             LOAD procura um ficheiro com a extensão
             .HEX
```

e obteremos uma resposta como

```
FIRST ADDRESS 0100
LAST ADDRESS  0119
```



```
BYTES READ    001A
RECORDS WRITTEN 01
A>
```

Podemos agora executar o programar em CP/M escrevendo muito simplesmente o seu nome, dado que possuímos um ficheiro chamado DEMO.COM. Mas como se trata de um programa que não produz qualquer saída no visor, nada saberemos escrevendo

```
A>DEMO
```

dado que aparentemente o computador limitar-se-á a ficar parado. Poderemos no entanto saber o que se passa se utilizarmos o potente utilitário chamado DDT (Dynamic Debugging Tool). Este programa permite-nos observar outro a verificar o que acontece em memória, assim como os diversos registos usados pelo microprocessador Z80. Basta-nos escrever

```
A>DDT DEMO.COM
```

e obteremos a resposta

```
DDT VERS 2.2
NEXT PC
0180 0100
-
```

O número sob PC indica-nos o valor inicial do contador de programa, neste caso 0100 que é o endereço inicial do programa que neste momento se encontra em memória. Por outras palavras, diz-nos que a primeira instrução do programa se encontra guardada na posição 0100. Sob a palavra NEXT encontra-se o endereço 0180. Trata-se da primeira posição livre a seguir ao programa em causa. O CP/M indica-nos sempre o espaço livre a seguir a um programa carregado, como veremos dentro em pouco.

O sinal — é um pedido de entradas do DDT. Este utilitário encontra-se portanto à espera de um comando. O comando L

levar-nos-á a obter uma listagem das instruções de programa a partir da que se encontra na posição 0100. Obteremos portanto

```
0100 LDA  0113   (LDA significa ''carregar no
0103 CALL 0117       acumulador'')
0106 STA  0115   (STA indica ''guardar o conteúdo do
0109 LDA  0114       acumulador'')
010C CALL 0117
010F STA  0116
0112 HLT
0113 RAR
0114 DAD D
0115 NOP     (Indica ''nenhuma operação'', um zero
0116 NOP      nesta posição)
```

Se carregarmos novamente em L, obteremos as instruções presentes nas dez posições seguintes:

```
-L
0117 ADD A
0118 ADD A
0119 RET
011A NOP
011B NOP
011C NOP
011D NOP
011E NOP
011F NOP
0120 NOP
```

São estas as instruções restantes, seguidas de NOP's, como se vê. Os NOP's nas posições 115 e 116 são zeros porque pedimos ao programa que colocasse aí dois zeros. De facto, são as posições etiquetadas R1 e R2.

O que a instrução L faz é «desassemblar» o código hexadecimal que se encontra guardado em memória. No entanto, talvez o leitor note que a codificação que foi dada ao programa não se encontra na forma das mnemónicas Z80 com que começámos. O significado disto será compreendido dentro em pouco.

Podemos agora observar a memória pedindo um «dump». Esta lista os endereços de memória do lado esquerdo, depois

o conteúdo destes em código hexadecimal, e finalmente, do lado direito, o equivalente ASCII destes códigos, se puderem ser traduzidos.

```
-D
0100 3A 16 01 CD 13 01 32 76 00 3A 17 01 CD 13 01 32 :.....2v.:.....2
0110 78 00 76 87 87 C9 1F 19 00 00 00 00 00 00 00 00 x.v.............
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1080 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i..........
1090 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2_..!f.p+q*e....
10A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2_..........2
10B0 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
```

Esta listagem diz-nos que a posição 100 contém o código hexadecimal 3A, 0101 contém 13, e assim por diante. Se observarmos a tabela de códigos ASCII e os seus equivalentes em hexadecimal fornecida no apêndice A, verificaremos que 3AH é o sinal de dois pontos, 32H é o algarismo 2 e 76H é «v». Se o código não possui qualquer equivalente ASCII, DDT tenta traduzi-lo e, se não consegue, imprime um ponto. De facto, se observarmos qualquer ficheiro .COM em CP/M encontramos geralmente um texto citando um «copyright»:

```
0740 43 4F 50 59 52 49 47 48 54 20 31 39 38 31 20 43 COPYRIGHT 1981 C
0750 4F 4D 50 49 4C 45 52 20 53 59 53 54 45 4D 53 2C OMPILER SYSTEMS,
0760 20 49 4E 43 0D 0A 45 4E 44 0D 0A 0D 18 23 0C 28  INC..END....#.(
```

Mas voltando ao nosso exemplo, podemos usar DDT para verificar o que acontece à medida que o programa prossegue. Para tal, utilizamos a instrução Trace e dizemos a DDT que deve executar um certo número de instruções. No nosso exemplo existem no total doze instruções a seguir à primeira. Escrevemos portanto

```
-T12
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LDA  0113
COZOMOEOIO A=IF B=0000 D=0000 H=0000 S=0100 P=0103 CALL 0117
```

```
COZOMOEOIO A=IF B=0000 D=0000 H=0000 S=00FE P=0117 ADD  A
COZOMOEOI1 A=3E B=0000 D=0000 H=0000 S=00FE P=0118 ADD  A
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=00FE P=0119 RET
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=0100 P=0106 STA  0115
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=0100 P=0109 LDA  0114
COZOMOEOI1 A=19 B=0000 D=0000 H=0000 S=0100 P=010C CALL 0117
COZOMOEOI1 A=19 B=0000 D=0000 H=0000 S=00FE P=0117 ADD  A
COZOMOEOI1 A=32 B=0000 D=0000 H=0000 S=00FE P=0118 ADD  A
COZOMOEOI0 A=64 B=0000 D=0000 H=0000 S=00FE P=0119 RET
COZOMOEOI0 A=64 B=0000 D=0000 H=0000 S=0100 P=010F STA  0116
COZOMOEOI0 A=64 B=0000 D=0000 H=0000 S=0100 P=0112 HLT
```

O leitor notará que o conteúdo do acumulador, A, passa de 1FH (31D) para 3EH (62D) e 7CH (124D), e daí para 19H (25D), 32H (50D) e finalmente 64H (100D). O contador de programa, P, indica ordenadamente as posições onde as instruções estão guardadas, a saber 100, 103, 117, 118, 119, 106, 109, 10C, 117, 118, 119, 10F e finalmente 112, onde se encontra guardada a instrução HALT.

Os dez caracteres no início de cada linha indicam-nos o estado de C (a flag «carry»), Z (a flag «zero»), M (a flag «minus»), E (a flag de «paridade») e I (a flag «carry interalgarismos»). Como se pode ver, estas flags possuem o valor zero ou um.

O comando T permite-nos percorrer manualmente as instruções do programa, uma de cada vez. Neste caso obtemos uma saída semelhante à cada pelo comando «global» T12 que executa doze instruções. A única diferença é que encontramos o endereço da instrução a executar em seguida no final da linha. Vejamos:

```
-T
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LDA  0113*0103
```

Uma nova ordem T produzirá o seguinte:

```
-T
COZOMOEOIO A=1F B=0000 D=0000 H=0000 S=0100 P=0103 CALL 0117*0117
```

e assim por diante.

Para abandonarmos DDT e voltar ao CP/M carregamos em GO.

É muito possível que o leitor queira experimentar este pequeno programa, ou qualquer outro. Introduza portanto a listagem no editor, e em seguida tente assemblá-lo:

```
CP/M ASSEMBLER - VER 2.0
100                     ORG 100H
S                       LD  A,(N1)      ;início do prog. princ.
P0100 CD0601            CALL QUAD       ;cálculo N1 × 4
P                       LD  (R1),A      ;R1=N1 × 4
S                       LD  A,(N2)
P0103 CD0601            CALL QUAD
S                       LD  (R2),A      R2=N2 × 4
L               N1:     DEFB 31
L               N2:     DEFB 25
L               R1:     DEFB 0
L               R2:     DEFB 0
S0106 87        QUAD:   ADD A,A         ;A × 2
S0107 87                ADD A,A         ;(A × 2) × 2 = A × 4
0109
000H USE FACTOR
END OF ASSEMBLY
```

Segundo parece, não terá ocorrido qualquer problema. Se examinarmos o ficheiro .HEX que é produzido, usando DDT, verificaremos que as únicas instruções traduzidas são as que residem nas posições indicadas:

```
0100 CALL 0106
0103 CALL 0106
0106 ADD  A
0107 ADD  A
0108 RET
```

Tudo isto parece muito confuso até compreendermos que os sistemas CP/M não utilizam apenas o processador Z80 mas também o Intel 8080. Este, apesar de terminar com o mesmo código nativo, deve ser alimentado em código 8080 e não Z80. O produto final pode ser o mesmo, mas o código-fonte é diferente. De facto o assemblador converte de uma linguagem assembler ou outra para o mesmo código-alvo. Teremos portanto

de reescrever o programa usando mnemónicas 8080, como se segue:

```
;programa assembler 8080 semelhante ao programa 3.1
                ORG 100H
                LDA N1
                CALL QUAD
                STA 0118
                LDA N2
                CALL QUAD
                STA 0120
                HLT
QUAD;           ADD A
                ADD A
                RET
N1:             DB 31
N2:             DB 25
                END
```

A instrução STA, guardar o conteúdo de A, indica-nos a posição para onde o número deve ir. Mas convirá ter cuidado porque, como acontece neste caso, se não existir H para significar que o número se encontra em hexadecimal será convertido de decimal para hexadecimal. Portanto, a posição 0118 passa a 0076H e 0120 a 0078H. Se apresentarmos este programa revisto ao assemblador, obtemos

```
A>ASM DEMO
CP/M ASSEMBLER - VER 2.0
0018
000H
000H USE FACTOR
END OF ASSEMBLY
```

Se imprimirmos o ficheiro .PRN obteremos o seguinte:

```
A>TYPE DEMO.PRN
     ;Um programa assemblador 8080 semelhante ao programa 3.1
0100                        ORG 100H
0100 3A1601                 LDA N1
0103 CD1301                 CALL QUAD
0106 327600                 STA 0118
0109 3A1701                 LDA N2
```



```
010C CD1301                 CALL QUAD
010F 327800                 STA 0120
0112 76                     HLT
0113 87         QUAD:       ADD A
0114 87                     ADD A
0115 C9                     RET
0116 1F         N1:         DB 31
0117 19         N2:         DB 25
0118                        END
```

Em seguida, se examinarmos o ficheiro .HEX obteremos

```
A>TYPE DEMO.HEX
:100100003A1601CD13013276003A1701CD130132B0
:08011000780076787C91F19EA
:0000000000
```

Um dump, usando o comando DUMP DEMO.HEX, dá-nos

```
0000 3A 31 30 30 31 30 30 30 30 33 41 31 36 30 31 43
0010 44 31 33 30 31 33 32 37 36 30 30 33 41 31 37 30
0020 31 43 44 31 33 30 31 33 32 42 30 0D 0A 3A 30 38
0030 30 31 31 30 30 30 37 38 30 30 37 36 38 37 38 37
0040 43 39 31 46 31 39 45 41 0D 0A 3A 30 30 30 30 30
0050 30 30 30 30 30 0D 0A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0060 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
0070 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A
```

Não esqueçamos que neste ponto o código-fonte já foi assemblado mas ainda não carregado, dado que se especificou o endereço inicial como sendo 0100H, não 0000H. Carregamo-lo portanto, tal como anteriormente, e obtemos:

```
A>LOAD DEMO

FIRST ADDRESS 0100
LAST  ADDRESS 0117
BYTES ADDRESS 0118
RECORDS WRITTEN 01
```

Em seguida examinamos a codificação final recorrendo a DDT:

```
A>DDT DEMO.COM
DDT VERS 2.2
```

```
NEXT  PC
0180 0100
-L                (Lista o ''código nativo'')
0100  LDA  0116
0103  CALL 0113
0106  STA  0076
0109  LDA  0117
010C  CALL 0113
010F  STA  0078
0112  HLT
0113  ADD A
0114  ADD A
0115  RET
0116  RAR
-L
0117  DAD D
0118  NOP
0119  NOP
011A  NOP
011B  NOP
011C  NOP
011D  NOP
011E  NOP
011F  NOP
0120  NOP
0121  NOP
```

Um «dump» do programa mostrar-nos-á o que se encontra guardado nas posições a partir de 0100H:

```
-D
0100 3A 13 01 CD 17 01 32 15 01 3A 14 01 CD 17 01 32 :.....s..:.....2
0110 16 01 76 1F 19 00 00 87 87 C9 00 00 00 00 00 00 ..v.............
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i..........
0190 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2_..!f.p+q*e....
10A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2_...........2
10B0 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
```

E se agora fizermos um «dump» do conteúdo de memória a partir do endereço 0000H, poderemos ver que o conteúdo das

posições 0076H e 0078H se alterou, tal como esperávamos que acontecesse:

```
-D0000
0000 C3 03 F2 81 00 C3 00 D4 FF 00 FF 11 FF 00 FF 00 ................
0010 F5 00 FF 12 B7 00 FF 02 DF 00 FF 00 6F 00 FF 90 ............o...
0020 FF 00 FF 13 FD 00 FF 10 EF 00 FF 10 EF 00 FF 42 ...............B
0030 BF 00 FF 48 EF 00 FF 18 C3 86 DA 12 EF 00 6B BE ...H..........k.
0040 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 ................
0050 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 00 44 45 4D .............DEM
0060 4F 20 20 20 20 43 4F 4D 00 00 80 01 CD 13 01 32 O    COM....w...
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FB A5 28 ...............(
0080 3A 00 76 87 87 C9 1F 19 00 00 00 00 00 00 00 00 :.....2v:......2
0090 78 00 76 87 87 C9 1F 19 00 00 00 00 00 00 00 00 x.v.............
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Se agora seguirmos a execução do programa dizendo a DDT que execute as doze primeiras instruções deste, veremos o movimento dos dados entre os registos da mesma maneira que anteriormente:

```
-T12
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LDA  0116
COZOMOEOIO A=1F B=0000 D=0000 H=0000 S=0100 P=0103 CALL 0113
COZOMOEOIO A=IF B=0000 D=0000 H=0000 S=00FE P=0113 ADD  A
COZOMOEOI1 A=3E B=0000 D=0000 H=0000 S=00FE P=0114 ADD  A
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=00FE P=0115 RET
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=0100 P=0106 STA  0077
COZOMOEOI1 A=7C B=0000 D=0000 H=0000 S=0100 P=0109 LDA  0113
COZOMOEOI1 A=19 B=0000 D=0000 H=0000 S=0100 P=010C CALL 011
COZOMOEOI1 A=19 B=0000 D=0000 H=0000 S=00FE P=0113 ADD  A
COZOMOEOI1 A=32 B=0000 D=0000 H=0000 S=00FE P=0114 ADD  A
COZOMOEOIO A=64 B=0000 D=0000 H=0000 S=00FE P=0115 RET
COZOMOEOIO A=64 B=0000 D=0000 H=0000 S=0100 P=010F STA  0078
```

E se agora fizermos um «dump» do conteúdo da memória a partir do endereço 0000H, poderemos ver que o conteúdo das posições 0076H e 0078H se alterou, tal como esperávamos que acontecesse:

```
-D0000
0000 C3 03 F2 81 00 C3 00 D4 FF 00 FF 11 FF 00 FF 00 ................
0010 F5 00 FF 12 B7 00 FF 02 DF 00 FF 00 6F 00 FF 90 ............o...
0020 FF 00 FF 13 FD 00 FF 10 EF 00 FF 10 EF 00 FF 42 ...............B
0030 BF 00 FF 48 EF 00 FF 18 C3 86 DA 12 EF 00 6B BE ...H..........k.
0040 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 ................
```

```
0050 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 00 44 45 4D .............DEM
0060 4F 20 20 20 20 43 4F 4D 00 00 80 01 CD 13 01 32 O    COM....w...
0070 00 00 00 00 00 00 7C 00 64 00 00 00 01 FB A5 28 ......|.d.......(
0080 3A 00 76 87 87 C9 1F 19 00 00 00 00 00 00 00 00 :.....2v:.......2
0090 78 00 76 87 87 C9 1F 19 00 00 00 00 00 00 00 00 x.v.............
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Os outros comandos de DDT são os seguintes:

A<endereço>
Introduzir código assembler (assemblar).

F<endereço inicial>,<endereço final>,<constante>
Colocar uma constante especificada em todos os endereços de memória entre duas posições especificadas.

G<endereço inicial>
Passar ao endereço inicial e começar a execução a partir deste.

G,<ponto de paragem1>
Executar até ao endereço onde se encontra o ponto de paragem indicado.

G,<endereço inicial>,<ponto de paragem1>,<ponto de paragem2>
Executar desde o endereço inicial até serem encontrados dois pontos de paragem.

GO
Sair para CP/M.

H,a,b
Realizar aritmética binária sobre um par de números.
Produz a+b e a−b em hexadecimal.

I<nome de ficheiro>
Introduzir um nome de ficheiro no Bloco de Controlo de Ficheiros antes de ler este último (entrada).

M<endereço inicial original>,<endereço final original>,<novo endereço inicial>
Mover um bloco.

R
Ler o ficheiro, indicado no comando I, para a memória.

S<endereço inicial>
Alterar o conteúdo de um endereço especificado.
Executa as instruções, listando os registos após a última instrução. O comando «trace» lista os registos após cada instrução.

X
Imprimir o conteúdo de todos os registos.

X<r>
Examinar o conteúdo de um registo específico.

10

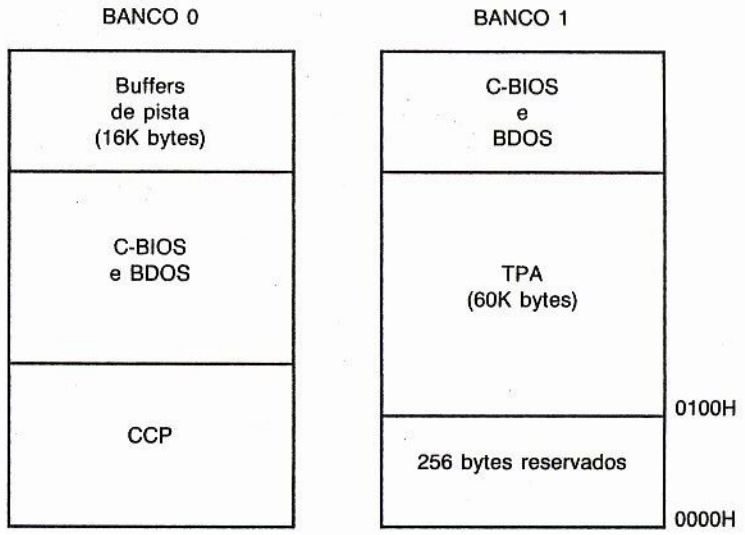
CP/M PLUS (CP/M VERSÃO 3.1)

Dado o baixo preço actual das memórias e a sua disponibilidade para uso em microcomputadores, nota-se que, de ano para ano, em dimensão de memória instalada aumenta de um factor de dois. Actualmente é norma usar 64 K de RAM em micros de 8 bits, e a passagem para 128 k produziu um problema. Um processador de 8 bits não pode endereçar, pelo menos directamente, mais de 64 K. Isto deve-se ao facto de o maior inteiro que pode ser guardado num registo do processador Z80 com 16 bits ser 65535 em notação decimal, ou FFFF em hexadecimal. 65535 é de facto 64 vezes 1024 menos 1; não esqueçamos que 1 K é de facto igual a 2 elevado à potência 10 (1024) e não a 1000.

No entanto, recorrendo a uma técnica designada por «comutação de memórias» é hoje possível levar o processador de 8 bits a utilizar mais de 64 K de RAM. Como é evidente, no caso de um processador de 16 bits não encontramos este problema, sendo possível a estes dispositivos endereçar directamente vários megabytes de memória. Mas neste campo aplica-se a versão do CP/M conhecida por CP/M-86, que será tratada no próximo capítulo.

Na figura que se segue apresentamos um mapa de memória de um sistema CP/M Plus típico, que utiliza 128 K de memória.

O BIOS é ampliado, passando a ser adaptado ao hardware de cada fabricante. O BDOS normal, dado que não se altera, é empregue nas mesmas condições, acompanhado de uma TPA (Área de programa transitória) com 60K bytes. Os dois grupos de memórias têm 64K cada.



Devido à quantidade extra de memória disponível em CP/M Plus esta versão apresenta certas facilidades que não existiam nas versões anteriores. Por exemplo, existe um HELP, um comando DIR mais extenso, gravação de data e hora nos ficheiros em disco e ainda a possibilidade de proteger diversas partições independentes de um mesmo disco recorrendo a uma «password». Por outro lado, usa um assemblador mais sofisticado designado por MAC, e um assemblador «relocatável» chamado RMAC; mas falaremos melhor disto adiante, ao tratarmos da nova versão de DDT que é designado por SID.

O CP/M Plus pode controlar uma mistura de tipos de disquettes pequenas, floppys de 8 polegadas e contém ainda um controlador de disco Winchester.

A especificação dos ficheiros é agora ampliada, de tal modo que o ficheiro CP/M pode ser identificado usando até um máximo de quatro características. Estas são o nome do drive, o nome do ficheiro, a sua extensão e uma password. Nestas condições, um nome de ficheiro pode ter a forma:

```
drive:nomeficheiro.extensão;password
```

1. HELP

Sabe-se que os manuais de computador são difíceis de seguir, especialmente para os principiantes em informática. O CP/M Plus permite-nos pedir informações sobre um vasto número das suas características, evitando a tarefa aborrecida de consulta ao extenso manual fornecido pela Digital Research. Basta-nos escrever

```
A>HELP
```

para surgir imediatamente no visor uma lista das características que o sistema nos pode explicar. Escrevemos em seguida o nome da característica que nos interessa, sendo imediatamente impresso no visor o texto explicativo.

Por exemplo, podemos obter:

```
A>HELP ERASE
                    ERASE
Syntax:

     ERASE {filespec} {[CONFIRM]}
```

seguido de uma explicação deste comando e da sua sintaxe: «O comando ERASE remove um ou mais ficheiros do directório de um disco. São aceites 'wildcards' na especificação do ficheiro. É imediatamente reservado espaço no directório para os dados usados mais tarde por um ficheiro. O comando ERASE pode ser abreviado para ERA.»

A máquina imprime em seguida esclarecimentos sobre as opções disponíveis neste comando:

OPÇÃO

[CONFIRM] Esta opção informa o sistema de que deve exigir confirmação antes de apagar um ficheiro cujo nome concorde com a especificação fornecida. «Confirm» pode ser abreviado para «C».

Finalmente são impressos alguns exemplos:

EXEMPLOS

```
A>ERASE X.PAS
```

Elimina o ficheiro X.PAS do disco no drive A.

```
A>ERA *.PRN
Confirm (Y/N)?Y
```

Todos os ficheiros com o tipo PRN serão eliminados do disco que se encontra no drive A.

```
B>ERA A:MY*.* [CONFIRM]
```

Cada ficheiro do drive A com um nome que se inicia por MEU será impresso ordenadamente, acompanhado de um ponto de interrogação pedindo confirmação. Escreva Y para eliminar o ficheiro indicado, ou N para não o eliminar.

```
A>ERA B:*.*
Confirm (Y/N)?Y
```

Neste caso, todos os ficheiros do drive B serão apagados após confirmação.

Este comando HELP diz-nos não só a forma de utilizar muitas das instruções de CP/M como ainda, se escrevermos

```
A>HELP HELP
```

explica-nos como poderemos usar a própria facilidade HELP. Mais ainda, é possível acrescentar outras explicações ao ficheiro HELP, designado HELP.HLP.

2. DIR

Nas anteriores versões de CP/M, o comando DIR reside no próprio sistema. Isto continua a acontecer, mas existe agora um novo ficheiro DIR.COM que nos fornece uma gama de informa-

ções muito mais vasta sobre os ficheiros presentes no disco. Por exemplo, se escrevermos:

```
A>DIR[FULL]
```

obteremos os ficheiros listados por ordem alfabética, com todos os seus atributos e muita informação sobre o espaço que lhes foi atribuído no disco. Por outro lado, poderemos ver qual o espaço ocupado no directório. A saída deste comando será algo do seguinte tipo:

```
Scanning Directory...
Sorting Directory...

    Name      Bytes  Recs  Attributes    Name      Bytes  Recs  Attributes
-------------------------------------------------------------------------
AFFIXTAB SPL    2k     6    DIR RO     ALIGN   TMK   6k    33    Dir RW
ARRANGE  TMK    2k     9    Dir RW     BAR     TMK   14k   106   Dir RW
CCP      COM    4k    25    Dir RW     CDATA   BAS   2k     3    Dir RW
COMBINE  TMK   12k    85    Dir RW     COMPUTE TMK   14k   110   Dir RW
CPM3     SYS   28k   214    Dir RW     CUSTOMER DAT  2k     4    Dir RW
CUSTOMER KEY    2k     1    Dir RW     DEMO          8k    64    Dir RW
DEMO     1      8k    59    Dir RW     DEMO    2     6k    37    Dir RW
DEMO     3      2k    16    Dir RW     DEMO    4     6k    33    Dir RW
DICTNARY SPL   30k   236    Dir RW     DIR     COM   16k   114   Dir RW
EDITOR   TMK   16k   116    Dir RW     INCOME  BAS   6k    41    Dir RW
INCOME1  BAS    4k    30    Dir RW     KEYS    BAS   2k     2    Dir RW
LABELS   BAS    2k     2    Dir RW     LOAD    TMK   6k    37    Dir RW
MAIL     BAS   12k    93    Dir RW     MAIL1   BAS   12k   94    Dir RW
MAILLIST MLL    2k     4    Dir RW     MAILLOOK BAS  2k     3    Dir RW
MATCH    TMK    2k    15    Dir RW     MBASIC  COM   24k   190   Dir RW
MCOUNTER FIL    2k     1    Dir RW     PARAMTER FIL  2k     1    Dir RW
PIP      COM   10k    68    Dir RW     PRINT   TMK   12k   90    Dir RW
Press RETURN to continue
Directory For Drive A: User 0

    Name      Bytes  Recs  Attributes    Name      Bytes  Recs  Attributes
-------------------------------------------------------------------------
PS       COM   14k   104    Dir RW     SETUP   BAS   2k     8    Dir RW
SORT     TMK    4k    27    Dir RW     TALLY   TMK   6k    42    Dir RW
TERMINAL UTL    6k    38    Dir RW     TMAKER  COM   8k    53    Dir RW
TMAKER   UTL    4k    26    Dir RW     TMAKER2 COM   10k   66    Dir RW
TMODIFY  COM    6k    42    Dir RW     TMODIFY UTL   12k   95    Dir RW
TRANS    BAS    2k     2    Dir RW     TRANSACT DAT  2k     2    Dir RW
UNLOAD   TMK    4k    23    Dir RW

Total Bytes     =   360k  Total Records =  2470   Files found =   47
Total 1k Blocks =   333   Used/Max Dir Entries For Drive A:   50/ 128
```


As opções disponíveis no utilitário DIR (não confundir com o comando DIR incorporado), são consideráveis:

OPÇÃO	EFEITO
ATT	Fornece os atributos do ficheiro
DATE	Indica a data e hora gravados em cada ficheiro
DIR	Indica apenas os ficheiros com o atributo DIR
DRIVE=(A,B,..)	Indica os ficheiros presentes nos drives listados
DRIVE=ALL	Como anteriormente, mas para todos os drives
EXCLUDE	Indica todos os ficheiros que não correspondem à especificação
FF	Envia um caracter de mudança de página para a impressora - se foi inicializada com o caracter CONTROL-P
FULL	Apresentado no exemplo anterior
LENGTH=n	Imprime n linhas na impressora antes de imprimir um cabeçalho de tabela (5<n<65536)
MESSAGE	Imprime os nomes dos drives e os números com utilizador que estão a ser procurados por DIR
RO	Apresenta os ficheiros apenas de leitura
RW	Apresenta apenas os ficheiros usados só para escrita
SYS	Apresenta apenas os ficheiros que possuem o atributo SYS
USER=(0,..15)	Apresenta apenas os ficheiros correspondentes aos utilizadores especificados
USER=ALL	Apresenta os ficheiros do drive correspondentes a todos os utilizadores

Uma outra extensão de DIR é DIRS, um outro utilitário nele incorporado. Permite a impressão de ficheiros com atributos de sistemas. Poderemos escrever

```
A>DIRS
```

e imprimir todos os ficheiros de sistema do utilizador 0 o que se encontram no drive A. Se escrevermos

```
4B>DIRS *.COM
```

veremos os nomes de todos os ficheiros de sistema correspondentes ao utilizador 4 no drive B que possuem a extensão .COM.

3. USER

O leitor poderá ter notado que o cabeçalho da listagem de directório menciona um «número do utilizador». Isto deve-se ao facto de esta versão de CP/M permitir a divisão do espaço em disco entre um número de utilizadores até dezasseis, cada um deles com um directório próprio, ou melhor dizendo um subdirectório próprio.

O directório de cada utilizador é mantido separado do de qualquer outro, e cada utilizador pode dispor de uma password que proteja o seu directório. Para definir uma nova partição para um utilizador usa-se o comando

```
A>USER 3
```

por exemplo, a fim de utilizar uma destas partições. O CP/M responde apresentando um pedido de entradas que contém não só o nome do drive mas também o número do utilizador

```
3A>
```

No entanto, este directório não possui quaisquer ficheiros, e não é sequer possível, pelo menos nesta fase, deslocar ficheiros de uma partição para outra. Que fazemos então? É como se tivéssemos um directório que não podemos usar! No entanto, a versão CP/M Plus do PIP permite-nos deslocar um ficheiro de um directório para outro.

Para fazermos isto partimos do exterior, voltando ao directório principal do disco, e carregamos PIP para a memória do computador. Em seguida entramos na área destinada ao novo utilizador. Usamos então o comando SAVE para gravar uma cópia de PIP nesse directório. A sequência será a seguinte:

```
A>USER 1          Isto coloca-nos na partição 1
1A>DIR            Pedir uma listagem do directório
No File           Não existe qualquer ficheiro
1A>USER
Enter User #: 0   Voltamos ao directório principal
A>PIP
CP/M 3 PIP VERSION 3.0
*                 Neste ponto carregamos em RETURN

A>USER 1 SAVE 30 PIP.COM
1A>PIP PIP.COM [G1]=A:PIP.COM
        O [G1] indica ao sistema que deve usar o PIP na área
        do utilizador 1. Note-se que não existe qualquer
        espaço entre o nome do ficheiro e o número da área do
        utilizador que constitui o seu destino.

1A>DIR
A:PIP.COM
1A>
```

Neste momento é já possível, por exemplo, copiar o MBASIC para a área do novo utilizador.

```
1A>PIP BASIC.COM[G1]=A:MBASIC.COM
1A>DIR
A:  PIP      COM: BASIC     COM
```

Pode-se então transferir outros ficheiros usando PIP, gravando-os na área do novo utilizador. Podemos portanto fazer agora uma listagem deste directório, obtendo:

```
Scanning Directory...

Sorting Directory...

Directory for drive A: User 1

    Name      Bytes  Recs  Attributes    Name      Bytes  Recs  Attributes
-------------------------------------------------------------------------
BASIC    COM   24k    190    Dir RW      DIR      COM   16k    114    Dir RW
TEST     BAS    2k      2    Dir RW      PIP      COM   10k     68    Dir RW
DEMO     BAS    2k      6    Dir RW

Total Bytes     =    54k  Total Records =   380   Files Found =    5
Total 1k Blocks =    50   Used/Max Dir Entries For Drive A:  113/ 128
```

Note-se que dividindo o disco em várias áreas de utilizador não se obtém mais espaço de directório. Acontece apenas que as 128 entradas do directório são simplesmente divididas entre os utilizadores.

4. SHOW

Em CP/M Plus, o comando STAT foi substituído por SHOW, sendo igualmente ampliada a informação por este produzida. Por exemplo, depois de obtermos a listagem apresentada acima, poderíamos escrever o seguinte:

```
1A>SHOW [USERS]
```

e obteríamos

```
A: Active User :    1
A: Active Files:    0      1
A: # of files  :   98      5

A: Number of free directory entries:        15
```

Se escrevermos

```
A>SHOW [SPACE]
```

obteremos uma indicação do modo de acesso e da quantidade de espaço disponível nos drives que se encontram em comunicação com o sistema.

```
A>SHOW [DRIVE]
```

dar-nos-á as características do drive A, e

```
A>SHOW B: [DIR]
```

imprimirá o número de entradas de directório livres no drive B. Finalmente,

```
A>SHOW [LABEL]
```

dar-nos-á a informação de etiquetas do drive A.

5. INITDIR

Antes de podermos gravar datas e horas nos ficheiros de um directório, é necessário inicializar este usando o programa INITDIR. Quando o fazemos são impressas as seguintes frases no visor:

```
A>INITDIR D:

INITDIR ACTIVARÁ A GRAVAÇÃO DE DATAS NO DRIVE ESPECIFICADO


Deseja reformatar o directório de D: (Y/N)?
```

Podemos usar em seguida SET para datar os ficheiros e utilizar as passwords apropriadas.

6. SET

A fim de protegermos os ficheiros existentes em áreas do utilizadores recorrendo a uma password utilizamos o comando SET. Fazemos isto em duas fases. Primeiro temos de activar a protecção por password escrevendo:

```
A>SET [PROTECT=ON]
```

Em seguida podemos atribuir passwords aos ficheiros escrevendo

```
A>SET FILE.DAT [PASSWORD=MUFFIN, PROTECT=WRITE]
```

o que produzirá a resposta

```
A:FILE.DAT Protection = Write, Password = MUFFIN
```

e isto protegerá o ficheiro designado FILE.DAT de qualquer actualização não autorizada, permitindo-nos escrever nele apenas no caso de o nome do ficheiro conter a password MUFFIN. Para usarmos o ficheiro teremos agora de incluir a password juntamente com o seu nome, como se segue:

```
FILE.DAT;MUFFIN
```

Podemos proteger, por exemplo, o comando DIR do mesmo modo

```
A>SET DIR. COM [PASSWORD=MUFFIN, PROTECT=READ]

A:DIR.COM Protection = Read, Password = MUFFIN
```

de tal modo que para usarmos o comando DIR teremos de escrever

```
A>DIR: MUFFIN [FULL]
```

Podemos igualmente definir a protecção como necessária para «DELETE», o que significa que só nos será exigida se quisermos eliminar o ficheiro ou dar-lhe um novo nome. Se definirmos a protecção para NONE («nada»), libertaremos o ficheiro de qualquer protecção.

SET permite-nos ainda colocar uma etiqueta num disco e proteger essa etiqueta com password se assim o desejarmos. Primeiro teremos de activar a protecção:

```
A>SET [PROTECT=ON]
```

depois etiquetamos o disco:

```
A> [NAME=MYDISK]
```

e damos a este disco etiquetado uma password:

```
A>SET [PASSWORD=FRED]
```

Para abandonarmos a password existente escrevemos

```
A>SET [PASSWORD=        Carregar em RETURN
```

Podemos acrescentar mais informação ao directório gravando datas e horas nos ficheiros. Podemos definir a data escrevendo

```
A>DATE SET
```


e a resposta será:

`Enter today's date (MM/DD/YY):`	Respondemos indicando a data do dia no formato MM/DD/AA, mês/dia/ano.
`Enter the time (HH:MM:SS)`	Respondemos indicando as horas no formato hora/minutos/segundos.
`Press any key to set time:`	Ao carregar numa tecla, é contado o tempo a partir dos valores indicados.

Basta-nos agora escrever o comando

```
A>DATE
```

para obtermos, por exemplo,

```
Tue 01/10/84 16:27:00
```

o que nos indica a data e a hora actuais, acrescentando o dia da semana em inglês.

Existe uma outra característica que podemos definir com SET. Se escrevermos

```
A>SET [CREATE=ON]
```

colocaremos a data e a hora de criação do ficheiro no directório. Se escrevermos

```
A>SET [ACCESS=ON]
```

colocamos a data e a hora do último acesso no directório. No entanto, não podemos criar e aceder «CREATE» e «ACCESS» simultaneamente. Apenas podemos fazer uma das duas coisas. Podemos no entanto escrever

```
A>SET [UPDATE=ON]
```

e registar a data e a hora da última actualização dos ficheiros. Se depararmos então uma listagem do directório obteremos informação adicional relativamente às datas de criação e última actualização, e sobre o facto de o ficheiro ter uma protecção de leitura, escrita ou eliminação.

7. DEVICE

Não só se substituiu o comando STAT por SHOW como ainda se introduziu o comando DEVICE, que passa a sê-lo por direito próprio. Em CP/M 3.1 podemos agora alterar todos os atributos dos vários dispositivos físicos e lógicos reconhecidos pelo CP/M em vez de passarmos pela rotina SYSGEN necessária nas versões anteriores.

Se escrevermos

```
A>DEVICE
```

obteremos os nomes dos dispositivos físicos e das suas atribuições a dispositivos lógicos, enquanto que

```
A>DEVICE NAMES
```

listará todos os dispositivos físicos e respectivas características, e

```
A>DEVICE VALUES
```

listará todas as actuais atribuições de dispositivos lógicos.

Podemos igualmente descobrir o estado actual de quaisquer dispositivos físicos escrevendo

```
A>DEVICE LPT
```

e poderemos alterar as características de um dispositivo escrevendo o seu nome seguido das novas características

```
A>DEVICE LPT [XON,2400]
A>
```

que nos permite usar o periférico de impressão a 2400 baud e aceitando o protocolo XON/XOFF. Este protocolo tem a ver com o facto de o CP/M poder enviar dados para um dispositivo quando tal lhe é pedido (isto é, quando a impressora está disponível) ou sem esperar que tal aconteça. Ao primeiro chama-se protocolo XON/XOFF, e ao último NOXON.

Tanto a velocidade de baud como o protocolo são característicos do dispositivo. Depois de estas características terem sido

definidas passam a existir como parte do sistema operativo guardado em memória. Quando a máquina é desligada são perdidas, só podendo ser tornadas permanentes executando um GENCPM. Os detalhes quanto à forma como isto é feito são complicados, e encontram-se tratados em maior profundidade no manual CP/M Plus. Encontram-se fora do âmbito deste livro.

No entanto, se quisermos será fácil aproveitar um ficheiro PROFILE.SUB, capaz de executar qualquer comando CP/M assim que o sistema é inicializado. É portanto possível definir dispositivos a partir de um ficheiro PROFILE.SUB, assim como definir a data e a hora, introduzindo um espaço específico do utilizador. O uso de ED permite-nos criar um ficheiro como

```
A>ED PROFILE.SUB

NEW FILE
     : *I
    1: DEVICE LPT [XON,9600]
    2: USER 1
    3: DATE SET
    4: MBASIC
    5: ^Z
     : *E
```

e quando o sistema é inicializado começa por definir o dispositivo de impressão físico como tendo o protocolo XON/XOFF e funcionando a 9600 baud. Depois coloca o sistema na área do utilizador 1, pede a data e a hora, e carrega o MBASIC pronto a usar.

8. PIP

Existem ainda dois outros parâmetros disponíveis para o comando PIP nesta versão de CP/M. Trata-se de

Comando	Efeito
A	''Archive''. Copiará apenas os ficheiros que foram actualizados desde a última cópia
C	''Confirm''. PIP pede em seguida uma confirmação de que a cópia deve realizar-

```
                -se. É útil quando se usam ''wild-cards''
                nos nomes de ficheiros.
Gn              Permite que um ficheiro seja colocado na
                área de outro utilizador.
```

O parâmetro B, leitura de blocos, está já disponível na versão 3.1.

9. GENCOM

O GENCOM permite-nos criar um novo ficheiro .COM a partir de uma série de ficheiros RSX (Resident System Extension, extensões de sistemas residentes).

Trata-se de ficheiros que devem encontrar-se em memória durante a execução do ficheiro .COM, mas podem ser eliminados depois a fim de libertar essa porção de memória. Diz-se que estes ficheiros estão «ligados» a uma área de CP/M. Estes ficheiros RSX podem ser módulos que tratam gráficos ou pilotam um periférico particular. GENCOM colocará um bloco de cabeçalho no início do ficheiro .COM. Se quisermos incluir certos ficheiros RSX temporariamente num ficheiro .COM existente, poderemos escrever

```
A>GENCOM PIP DRIVER1 DRIVER2
```

onde PIP.COM terá os ficheiros DRIVER1 e DRIVER2 temporariamente ligados. Se escrevermos subsequentemente

```
A>GENCOM PIP
```

os ficheiros adicionais serão desligados e PIP voltará à sua identidade normal.

10. COPYSYS

Depois de o disco ter sido formatado, é necessário colocar a informação essencial relativa às pistas de sistema no novo disco se se pretende utilizá-lo depois para inicializar a máquina. COPYSYS copiará a informação CP/M essencial quando se dá o comando:

```
A>COPYSYS
```


com o disco «master» no drive A, funcionando como fonte e o novo disco no drive B, funcionando como destino. A mensagem «função terminada» indicará que a cópia teve êxito.

Pode-se usar COPYSYS num disco que já possua ficheiros gravados. O que se encontra no disco não é afectado pelo programa COPYSYS. Isto deve-se ao facto de as pistas de sistema serem reservadas a uma única utilização, não sendo escritos por directórios ou ficheiros.

11. MAC, RMAC E SID

Em vez de um único assemblador, o CP/M dispõe de dois. O primeiro destes é conhecido por MAC. Este programa aceita o mesmo código fonte do anterior programa ASM, mas além disso aceita igualmente comandos macro. Por exemplo, existe uma função «escrever caracter», designada «wchar», que pode ser usada num programa assembler, evitando o aborrecimento de ter de escrever uma rotina de saída separada. Os convencionais ficheiros .PRN, .HEX e .SYM são produzidos do mesmo modo que anteriormente, e o ficheiro assemblado é carregado pelo mesmo ficheiro LOAD.COM que se usa nas outras versões de CP/M.

O CP/M Plus substituiu DDT pelo DID — «Symbolic Instruction Debugger». Este trabalha do mesmo modo que o DDT e utiliza os mesmos comandos. Por outro lado, o SID fornece um par de utilitários TRACE e HIST; no caso do primeiro, podemos voltar atrás relativamente a um ponto de passagem e descobrir as instruções que conduziram a esse ponto. TRACE recolhe os endereços de um máximo de 256 instruções. HIST desenha um gráfico de barras mostrando a frequência relativa de partes do código. Isto significa que podemos descobrir quais as passagens do programa que são executadas com maior frequência.

Os comandos SID são um superconjunto dos comandos DDT descritos anteriormente, incluindo:

```
C     Chamar uma sub-rotina do seu endereço inicial
E     Carregar um programa e tabela de símbolos para
      execução
E*    Empregar uma tabela de símbolos
```

P Regista o número de vezes que um programa passa por um determinado endereço

V Indica o valor actual dos parâmetros SID. São NEXT, a posição que se segue em memória, MSIZE, a posição em memória que se segue ao maior ficheiro lido, PC que corresponde ao valor actual do contador de programa, e END que é o endereço do final da memória disponível

W Escrever o conteúdo de um segmento de memória específico para um dado ficheiro

RMAC é um instrumento mais útil do que MAC quando se desenvolvem extensos programas em código assembler, dado que se trata de um macro-assemblador relocatável. Isto significa que não é necessário especificar um endereço inicial no código-fonte, ao contrário do que é habitual noutros programas ASM. Os ficheiros de saída de RMAC são .SYM, .PRN e .REL.

Qualquer número de ficheiros que tenham sido produzidos por RMAC podem ser ligados entre si, recorrendo ao comando LINK, e sendo produzido em endereço inicial. O resultado é que podemos criar, deste modo, um ficheiro .COM pronto a ser usado. Por exemplo, se existirem quatro módulos produzidos por RMAC, todos eles terão extensões .REL. Podem ser combinados como tidas as referências externas e transformados num único ficheiro executável escrevendo

```
A>LINK MAIN=PROG1,PROG2,PROG3,PROG4
```

produzindo assim um ficheiro designado MAIN.COM por ligação («linkage») dos quatro ficheiros PROG1.REL, PROG2.REL, PROG3.REL e PROG4.REL. Esta operação será já familiar dos utilizadores de minicomputadores que, por exemplo, compilam um programa FORTRAN e todas as suas subrotinas, ligando-as à biblioteca de execução FORTRAN, de tal modo que todas as macros referenciadas nos módulos originais FORTRAN são ligadas ao conteúdo apropriado da biblioteca de execução. É então possível executar o módulo completo.

11

CP/M-86

Quando surgiram no mercado os processadores 8088 e 8086 da Intel, de 16 bits, permitindo o desenvolvimento de uma nova geração de microcomputadores, o CP/M foi rapidamente alterado de modo a poder ser executado neles. A CP/M-86 é uma versão do CP/M 2.2, contendo todos os comandos normais deste sistema. Devido às maiores capacidades de endereçamento dos novos processadores de 16 bits, os programas de aplicações escritos para o CP/M-86 podem endereçar até 1 megabyte (1 048 576 bytes) de RAM. Todos os ficheiros escritos por CP/M-86 são compatíveis com as versões de CP/M de 8 bits, o que permite uma fácil comunicação de dados entre as máquinas de 8 bits e as de 16 bits.

Para além do que se disse, a nova versão dispõe de uma maior flexibilidade de controlo dos drives de disco. O CP/M-86 permite até 16 drives lógicos de até 8 megabytes cada, e portanto um máximo de 128 megabytes de memória periférica em linha.

Todos os comandos de CP/M residentes, como DIR, ERA, TYPE e REN são exactamente iguais aos de CP/M 2.2. Os ficheiros com a extensão .COM das versões anteriores do sistema imperativo passaram a ser conhecidos por ficheiros .CMD, e o PIP.CMD, o ED.CMD, o FORMAT.CMD, o STAT.CMD e o SUBMIT.CMD funcionam da mesma maneira que anteriormente. O assemblador passou a ser conhecido por ASM86.CMD, o DDT por DDT86.CMD, e o MBASIC é executado a partir de um ficheiro de comandos chamado BASIC86.CMD.

Quando é assemblado código escrito em linguagem 8086, é necessário substituir o comando LOAD por GENCMD. Os programas em código Assembler devem possuir um nome com uma

extensão .A86, e os ficheiros em codificação hexadecimal que resultam da execução do ASM86 terão a extensão .H86. A assemblagem produz ainda um ficheiro .LST e uma tabela de símbolos num ficheiro .SYM. DDT86 utiliza todos os comandos DDT «standard», exceptuando GO, que geralmente nos reenvia para o nível de aceitação de comandos do CP/M. A forma de sair do DDT86 consiste em carregar em CONTROL-C. O ASM-86 permite igualmente o uso de macros, e pode ser fornecido numa versão capaz de correr sob CP/M num sistema de 8 bits.

Vejamos um programa escrito em código 8086, mostrando o uso do ASM86 e do DDT86:

```
A>TYPE DEMO1.A86

;UM PROGRAMA 8086 SEMELHANTE AO PROGRAMA 3.1
;
                              ORG 100H
N1                            EQU 31
N2                            EQU 25
                              MOV AX,(N1)
                              CALL QUAD
                              MOV BX,AX
                              MOV AX,(N2)
                              CALL QUAD
                              MOV CX,AX
STOP:                         JMP STOP
;SUBROUTINE STARTS
QUAD:                         ADD AX,AX
                              ADD AX,AX
                              RET
                              END
```

Note que os registos possuem nomes ligeiramente diferentes dos usados no assemblador de 8 bits.

Em seguida assemblamos o programa:

```
A>ASM86 DEMO1

CP/M 8086 ASSEMBLER VER 1.0
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY. NUMBER OF ERRORS:     0
```


Podemos agora observar os ficheiros que resultam da assemblagem:

```
A>TYPE DEMO1.LST

ASM86 VER 1.0       SOURCE: DEMO1.A86
        ;UM PROGRAMA 8086 SEMELHANTE AO PROGRAMA 3.1
        ;
                                      ORG 100H
  001F              N1                EQU 31
  0019              N2                EQU 25
  0100 B8 1F 00                       MOV AX,(N1)
  0103 E8 0D 00                       CALL QUAD
  0106 8B D8                          MOV BX,AX
  0108 B8 19 00                       MOV AX,(N2)
  010B E8 05 00                       CALL QUAD
  010E 8B C8                          MOV CX,AX
  0100 E9 FD FF     STOP:             JMP STOP
        ;INÍCIO SUBROTINA
        QUAD:                         ADD AX,AX
  0113 03 C0                          ADD AX,AX
  0115 03 C0                          RET
  0117 C3                             END

  END OF ASSEMBLY. NUMBER OF ERRORS:       0
```

Podemos então examinar a tabela de símbolos:

```
A>TYPE DEMO1.SYM
0000 VARIABLES
0000 NUMBERS
0019 N2          001F N1
0000 LABELS
0110 STOP        0113 QUAD
```

O ficheiro hexadecimal pode ser igualmente impresso:

```
A>TYPE DEMO1.H86
:04000003000001 00F8
:18010081BB1F00E80D008BD8B81900E805008BC8E9FDFF03C003C0C3F8
:00000001FF
```

Podemos em seguida carregar o ficheiro assemblado usando GENCMD

```
A>GENCMD DEMO1

BYTES READ     001C
RECORDS WRITTEN 04
```

Finalmente podemos examinar os restantes ficheiros .CMD usando DDT86

```
A>DDT86 DEMO1.CMD
DDT86 X.0
START     END
CS 14CD:0000 14CD:011F
-L
14CD:0100 MOV   AX,001F
14CD:0103 CALL  0113
14CD:0106 MOV   BX,AX
14CD:0108 MOV   AX,0019
14CD:010B CALL  0113
14CD:010E MOV   CX,AX
14CD:0110 JMP   0100
14CD:0113 ADD   AX,AX
14CD:0115 ADD   AX,AX
14CD:0117 RET
14CD:0118 ADD   [BX+SI],AL
14CD:011A ADD   [BX+SI],AL
-D100
```

Note-se que os endereços são dados sob a forma de um deslocamento relativamente a um endereço-base, dado que não podemos guardar qualquer endereço superior a FFFF em dois bytes.

```
14CD:0100 B8 1F 00 E8 0D 00 8B D8 B8 19 00 E8 05 00 8B C8 ................
14CD:0110 E9 FD FF 03 C0 03 C0 C3 00 00 00 00 00 00 00 00 ................
14CD:0120 7F 37 00 DF 14 01 7F 37 00 00 00 00 00 00 00 00 .7.....7........
14CD;0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
14CD:0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
14CD:0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
14CD:0160 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20 20 .............
14CD:0180 20 20 20 20 20 20 20 20 00 00 00 00 00 20 20 20          .....
14CD:0190 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00          ........
14CD:01A0 00 00 44 45 4D 4F 31 2E 43 4D 44 00 00 00 00 00 ..DEMO1.CMD.....
14CD:01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-T13
          AX   BX   CX   DX   SP   BP   SI   DI   IP
--I------ 0000 0000 0000 0000 119E 0000 0000 0000 0100 MOV   AX,001F
--I------ 001F 0000 0000 0000 119E 0000 0000 0000 0103 CALL  0113
```



```
--I------ 001F 0000 0000 0000 119C 0000 0000 0000 0113 ADD   AX,AX
--I---A-- 003E 0000 0000 0000 119C 0000 0000 0000 0115 ADD   AX,AX
--I----A- 007C 0000 0000 0000 119C 0000 0000 0000 0117 RET
--I----A- 007C 0000 0000 0000 119E 0000 0000 0000 0106 MOV   BX,AX
--I----A- 007C 0000 0000 0000 119E 0000 0000 0000 0108 MOV   AX,0019
--I----A- 0019 007C 0000 0000 119E 0000 0000 0000 010B CALL  0113
--I----A- 0019 007C 0000 0000 119C 0000 0000 0000 0113 ADD   AX,AX
--I----A- 0032 007C 0000 0000 119C 0000 0000 0000 0115 ADD   AX,AX
--I------ 0064 007C 0000 0000 119C 0000 0000 0000 0117 RET
--I------ 0064 007C 0000 0000 119E 0000 0000 0000 010E MOV   CX,AX
--I------ 0064 007C 0064 0000 119E 0000 0000 0000 0110 JMP   0110
--I------ 0064 007C 0064 0000 119E 0000 0000 0000 0110 JMP   0110
--I------ 0064 007C 0064 0000 119E 0000 0000 0000 0110 JMP   0110
--I------ 0064 007C 0064 0000 119E 0000 0000 0000 0110 JMP   0110
^C
```

O comando CONTROL-C terminará a execução do programa sob DDT86 e reenviará o utilizador para CP/M.

12

MP/M

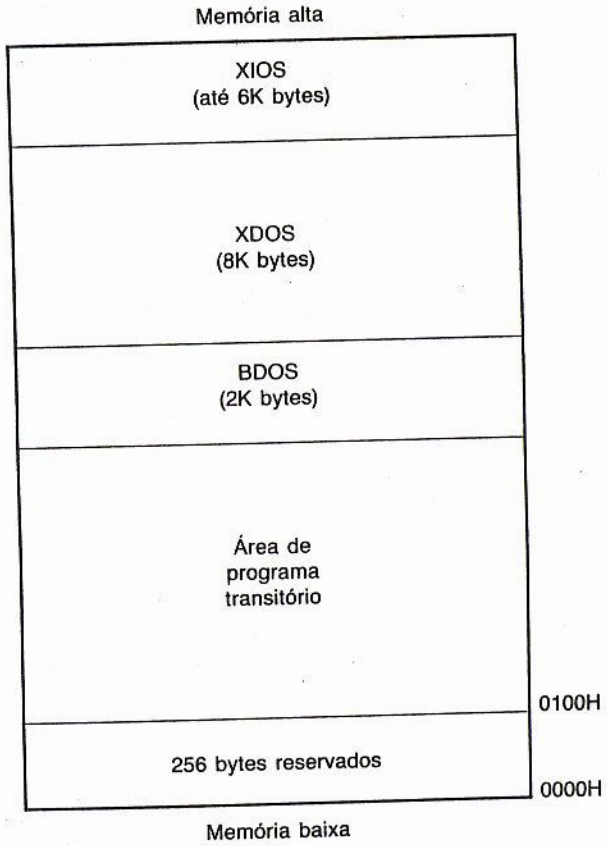
Dada a constante popularidade do sistema operativo CP/M para a execução de programas de aplicações comerciais — folhas de cálculo, tratamentos de texto, bases de dados e sistemas de contabilidade — era lógico colocar os ficheiros e programas escritos sob este sistema à disposição de mais de um utilizador de cada vez. Dado o preço dos microcomputadores actuais e a sofisticação da tecnologia de comunicações, não é surpreendente que tenha surgido no mercado uma versão de CP/M para vários utilizadores. A MP/M existe em versões de 8 a 16 bits: MP/M e MP/M-86.

O MP/M, cuja versão actual é genericamente conhecida por MP/M II, é um sistema operativo multitasking e multiutilizador. A um computador funcionando em MP/M estarão ligados vários terminais, impressoras e outros dispositivos periféricos. O MP/M pode suportar até dezasseis drives lógicos com uma capacidade até 512 megabytes cada. 254 dispositivos lógicos, como impressoras, terminais e outros dispositivos de entrada/saída, podem ser suportados pelo MP/M. Pode-se executar um job separado em cada terminal de modo independente, de tal modo que é como se estivesse a ser usado o CP/M normal. O MP/M funciona de uma maneira semelhante a um sistema minicomputador em partilha de tempo mas, convirá dizê-lo, a velocidade não se compara à deste.

A fim de satisfazer as exigências extra de vários utilizadores, o MP/M usa versões ampliadas do BIOS e do BDOS, conhecidas por XIOS (Extended Input/Output System) e XDOS (Extended Disk Operating System). O XDOS (e note-se que o BDOS fixo continua a existir) realiza o controlo do «multitasking», dos terminais e filas de espera de uso dos periféri-

cos. O interpretador da linha de comandos está contido no XDOS.

O arranjo do CP/M em memória, no caso do sistema MP/M, é o apresentado na figura seguinte.



O MP/M II foi concebido para executar a maior parte dos programas CP/M nos terminais que se encontram ligados ao processador central. Cada utilizador tem acesso aos ficheiros guardados nos discos de sistema e a uma ou mais impressoras a ele ligadas. A fim de garantir que não ocorrem choques quando mais de uma pessoa deseja modificar o mesmo ficheiro, existe um sistema de reserva de registos. Se um ficheiro se encontra reservado, só um utilizador o poderá abrir num dado momento.

Se um ficheiro não é reservado, e mesmo que sejam feitos vários pedidos de abertura de ficheiro, o utilizador pode reservar temporariamente grupos de registos.

Os ficheiros podem ainda ser obrigatoriamente usados apenas para leitura, de tal modo que a informação neles guardada possa ser consultada mas não modificada.

Os comandos CP/M, tal como o PIP, o STAT, o DIR, o TYPE, o REN e o ERA são exactamente iguais aos do CP/M convencional. De facto, a única diferença óbvia para o utilizador é o pedido de entradas normal:

```
A>
```

se apresentar prefixado pelo número do utilizador:

```
0A>
```

o que é bastante semelhante à partição de um disco em CP/M Plus, como já se referiu. Os utilizadores são ligados ao sistema pelo comando USER, exactamente da mesma maneira. De facto, existem várias características semelhantes entre o MP/M II e o CP/M Plus. Uma outra, é o modo como o directório do utilizador é impresso pelo comando DIR. A semelhança não é muito surpreendente na medida em que ambos os sistemas operativos necessitam de um microcomputador com «bancos» de memória.

O MP/M II é concebido para ser usado num sistema de disco rígido, se bem que se possam utilizar floppy disks, e é capaz de usar um disco «virtual». Este é, na realidade, uma secção da RAM do computador que é semelhante a um drive de disco, geralmente com uma dimensão de 256 K, mas tendo tempos de acesso muito inferiores (cerca de um décimo) aos de um sistema de disco. Isto torna extremamente rápido o acesso aos dados guardados (se bem que temporariamente) neste disco virtual, o que o torna muito útil num sistema multiutilizador. No entanto, é necessário carregar um ficheiro no disco virtual antes de se poder usá-lo, o que se faz recorrendo ao PIP. O ficheiro é evidentemente perdido quando se desliga o sistema, dado que

toda a informação guardada em RAM é volátil. Um disco virtual pode ser obtido usando o comando

```
0A>VDSK:=ON
```

e o seu «nome de drive» é atribuído usando ASSIGN

```
0A>ASSIGN D:VDSK
```

A partir daqui, o «drive» D é tratado como se fosse verdadeiramente um novo disco.

Vejamos alguns outros comandos especializados do MP/M II:

ABORT	Pára a execução de um programa, em qualquer terminal, desde que se especifique o número de consola de onde o programa foi posto em execução
ATTACH	Liga um programa a uma consola
CONSOLE	Cada terminal do sistema MP/M possui um número de consola, que não é igual ao número do utilizador. MPMSTAT lista a sua informação por número de consola, em vez de o fazer por número do utilizador
CONTROL-D	''Desliga'' o programa da consola e executa-o enquanto o utilizador executa outro programa ligado com ATTACH, o que permite portanto executar dois programas simultaneamente
DSKRESET	Reinicializa o mapa de atribuição de um drive de disco depois de o disco ter sido alterado
ERAQ	Tal como ERA, em CP/M, mas pergunta se queremos apagar ou não. Por outras palavras, podemos utilizar ERAQ juntamente com um nome geral de ficheiro, incluindo ''wild cards'', sendo cada nome específico impresso juntamente com uma pegunta sim/não. Semelhante ao ERA de CP/M Plus, com a opção C (confirmar)
MPMSTAT	Lista todas as actividades em progresso no momento em que o comando é dado
PRINTER	Este comando atribui um dispositivo de impressão para uma consola particular. Vários utilizadores podem ter acesso a uma impressora, mas só um a pode usar de cada vez
SCHED	Permitirá a carga de um programa e a sua execução num momento futuro especificado

SDIR	Imprime e lista todo o directório MP/M, incluindo ficheiros de sistema
SPOOL	Envia um ficheiro para a fila de ''spool'' do dispositivo de ficheiro
STOPSPLR	Termina o spooling e esvazia a fila de spool
TOD	Imprime ou define a data e a hora – como em CP/M Plus

13

CONCURRENT CP/M

A maior parte dos processadores centrais gastam o seu tempo fazendo muito pouco. Isto deve-se ao facto de muitos dos jobs realizados pelo computador, grande ou pequeno, mainframe ou micro, serem atrasados pela lentidão de entrada e saída de dados. O dispositivo de entrada é geralmente o teclado, e mesmo o mais rápido dactilógrafo não consegue fornecer dados a uma velocidade que se compare àquela com que estes podem ser processados. Do mesmo modo, a velocidade das impressoras é bastante mais reduzida do que aquela a que o computador lhe entrega os dados — e daí a necessidade de buffers de impressora. Trata-se de um problema que já será conhecido pelos utilizadores de mainframes e minicomputadores; foi ultrapassado recorrendo a uma técnica dita de «multiprogramação».

Esta técnica permite ao computador tratar mais de um programa de cada vez. O resultado significa que, por exemplo, enquanto alguém está a editar um programa, o processador central pode estar a realizar alguma outra tarefa, por exemplo compilar um programa em FORTRAN, nos intervalos entre a recepção de dados do teclado.

A Concurrent CP/M e a Concurrent CP/M-86 permitem, portanto, a execução de várias tarefas simultaneamente usando uma técnica de multiprogramação. Isto é feito permitindo que um único microcomputador que execute Concurrent CP/M tenha até quatro consolas «virtuais», cada uma delas executando um programa diferente, e todas funcionando ao mesmo tempo. Em teoria podem ser usadas até dezasseis consolas virtuais, mas nota-se uma considerável degradação da resposta se cada tarefa tiver dimensões razoáveis.

Um bom exemplo do uso prático da Concurrent CP/M é aquele em que um package de tratamento de texto, um package de transacções em carteira e um package de modelação financeira estão a actualizar três das consolas virtuais, enquanto a quarta está ligada à linha telefónica e daí a um serviço de distribuição.

A Concurrent CP/M permite a inicialização de um job numa destas consolas virtuais e, quando este está em execução, o arranque de um outro job numa outra consola; e assim por diante até serem executados simultaneamente quatro tarefas. É muito simples comutar janelas a fim de observar qualquer consola virtual. Isto é feito pelo operador carregando na tecla CONTROL juntamente com a tecla 0, 1, 2 ou 3. CONTROL e 0 dá-nos a consola virtual 0, CONTROL e 3 permite-nos ver o que se passa na consola virtual 3, e assim por diante.

As últimas versões da Concurrent CP/M permitem o uso de janelas, um passatempo muito comum nos microcomputadores actuais. Isto significa que é possível dividir o visor num máximo de quatro visores mais pequenos, de tal modo que cada um deles mostre um dos programas em execução.

Não esqueçamos que tudo isto é feito num único microcomputador que possui pelo menos uma RAM de 128 K. A Concurrent CP/M pode suportar até dezasseis drives lógicos, de até 512 megabytes cada.

Tal como em MP/M, usa-se a reserva de registos de tal modo que um programa que esteja a ser executado de uma consola virtual não pode corromper os ficheiros que estão a ser usados por outro, dado que o Concurrent CP/M é um sistema operativo multitasking monoutilizador. A execução de vários programas é feita dividindo o tempo do processador central («time--slicing») do mesmo modo utilizado nos minicomputadores multiutilizador.

Os utilizadores de Concurrent CP/M podem identificar-se por um número de utilizador, que fica assim ligado a um directório específico. Os ficheiros podem incluir data e hora, e ser protegidos por password.

Quando se usa o microcomputador sob Concurrent CP/M, a primeira coisa que se nota é que existe uma linha na parte

inferior do visor que nos indica qual a consola a que estamos ligados, o nome do programa em execução nessa consola e a impressora usada pelo programa. Vejamos um exemplo:

```
CONSOLE=1 DYNAMIC SC PRINTER=1
```

que nos indica que estamos a usar a consola 1, executando o programa SuperCalc e que a impressora ligada à consola é a impressora 1. O modo dinâmico é um modo de funcionamento tal, que todas as saídas que iriam normalmente para o visor são enviadas para uma área de RAM, prontas para serem apresentadas quando a consola passar a activa. No entanto, se estivermos em modo dinâmico, esta área de RAM contém apenas a última imagem criada. Se estivermos em modo BUFFERED então, à medida que cada imagem é preenchida, o funcionamento do programa é suspenso até o conteúdo dela ser de facto impresso. Assim, se executarmos um programa em modo DYNAMIC, há a possibilidade de perder certas imagens para sempre — por exemplo as mensagens de erro numa compilação.

A Concurrent CP/M utiliza todos os habituais comandos CP/M como PIP, STAT, DIR, etc., juntamente com alguns outros que lhe são específicos. Por exemplo:

```
ABORT   Permite a paragem de um determinado programa em
        fundo. Torna-se assim possível, ao utilizador numa
        consola virtual, parar um programa executado
        noutro. Equivale à instrução MP/M com o mesmo nome
SYSTAT  Mostra o estado das quatro consolas, além do
        estado dos processos nelas executados
VCMODE  Define o modo de uma consola virtual - como
        DINAMIC ou BUFFERED - e a dimensão máxima de
        ficheiro para o modo BUFFERED
```

A versão 3.1 de Concurrent CP/M permite a execução de programas escritos para o IBM PC, para execução sob PC-DOS 1.1. Por enquanto, só executa os programas correspondentes à versão 1.1 daquele sistema, mas espera-se dentro em pouco uma emulação da versão 2.0. Por outro lado, esta versão do sistema operativo pode ser usada também em multiutilizador, com até quatro consolas «reais» ligadas ao mesmo microprocessador e

executando cada uma delas até quatro consolas «virtuais». No entanto, como já se disse anteriormente, se o sistema se tornar muito pesado haverá uma redução considerável nos tempos de resposta.

14

CP/NET

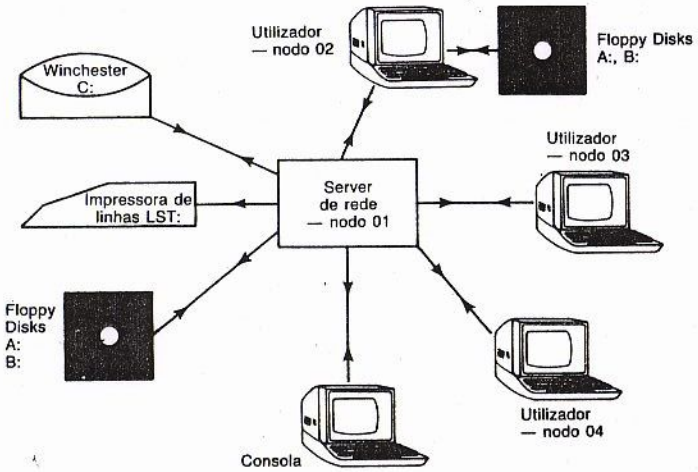
Agora que a ligação de microcomputadores em redes se tornou bastante comum, não é de admirar que o sistema operativo CP/M e os programas executados sob este passem a estar disponíveis em rede.

À primeira vista, dir-se-ia que um sistema MP/M e um CP/M em rede não deveriam ser muito diferentes. No entanto, não é este o caso. Um sistema MP/M requer um único computador que disponha de um certo número de terminais «escravos», cada um deles partilhando os recursos do principal. Numa rede, são ligados entre si vários computadores CP/M independentes, talvez tendo fabricos diferentes, de tal modo que partilham não só recursos comuns como ainda os recursos específicos de cada um deles. As máquinas separadas podem funcionar isoladamente sob os seus próprios sistemas CP/M, mas o uso de software de CP/NET permite-lhes comunicarem com os outros computadores ligados em rede.

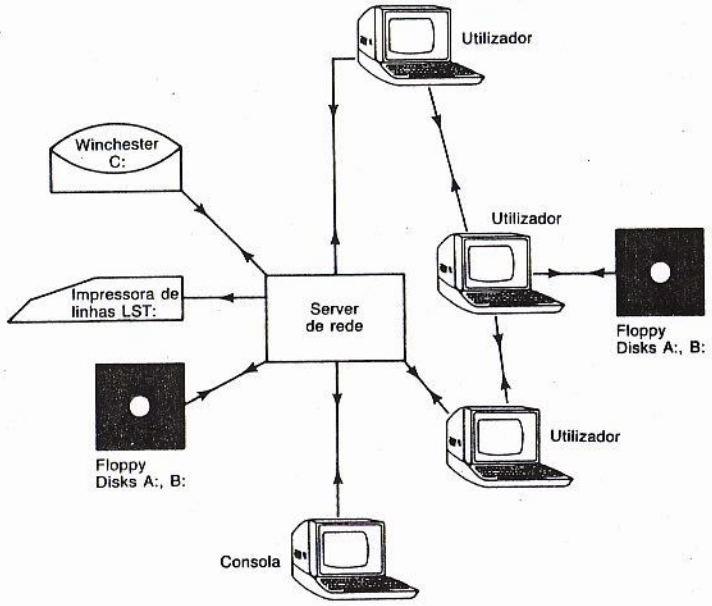
A CP/NET pode ser ligada a um certo número de sistemas de rede diferentes, entre os quais a Ethernet e a OMNINET. A configuração dos computadores numa rede é geralmente uma das três apresentadas em seguida:

Em cada uma destas redes devem existir pelo menos um «Server» executando MP/M II e que trata o conjunto do sistema. Existe, além disso, um conjunto de nodos que corresponderão a microcomputadores CP/M ou a terminais inteligentes. Um único «server» de rede MP/M II pode suportar até 16 nodos. Os micros disporão do sistema CP/M normal, mas conterão além deste o software de CP/NET que lhes permitirá comunicarem com o resto da rede. Os terminais inteligentes disporão de uma pequena quantidade de RAM e do seu microprocessador pró-

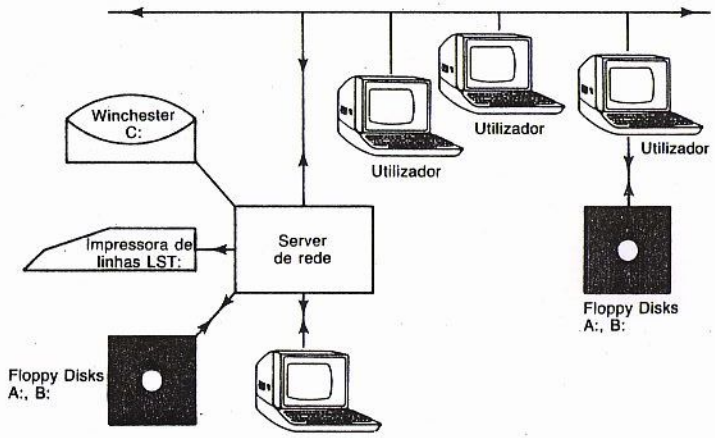
1) Estrela



2) Anel



3) «Multidrop»



prio, mas não de discos. Necessitarão, no entanto, de uma ROM de 4K contendo CP/NOS que é uma versão reduzida da CP/NET e contém apenas o software requerido ao terminal inteligente para ligar à rede. Daqui em diante todo o software requerido nesse nodo da rede é carregado através desta pelo «server».

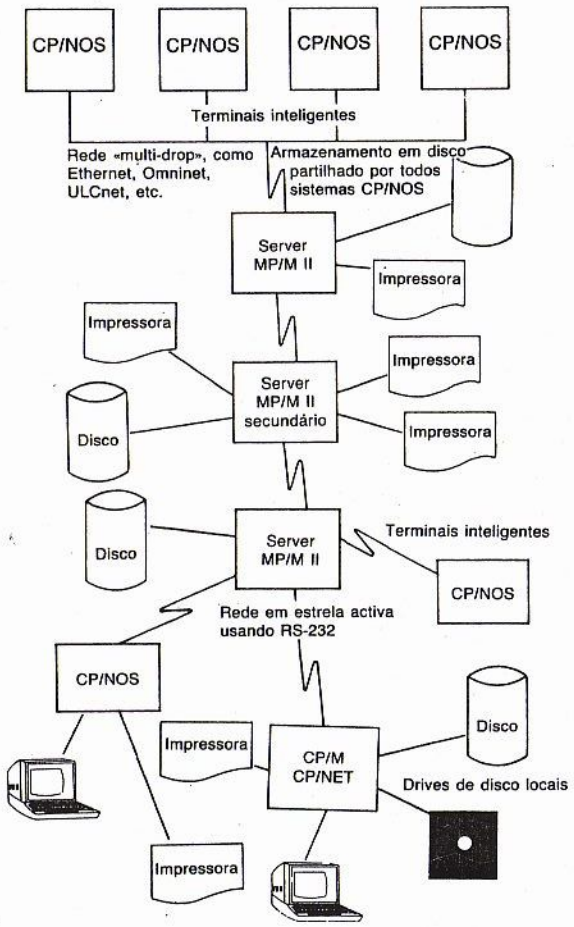
O resultado de tudo isto é que o processamento é realizado localmente, e só quando é necessária uma facilidade que se encontra noutro ponto da rede ocorre uma comunicação entre os processadores. Cada utilizador parece assim dispor de um potente sistema CP/M dedicado. No diagrama seguinte apresenta-se um complexo sistema CP/NET.

Para além dos habituais comandos CP/M, existem vários específicos de redes. São os seguintes:

```
LOGIN   Liga o utilizador ao sistema
LOGOFF  Desliga o utilizador do sistema
NETWORK Permite ao utilizador alterar o ambiente de rede.
        Isto é feito atribuindo nomes lógicos a dispositivos
        físicos noutro ponto da rede. Por exemplo:

        A>NETWORK H:=C:(01)
```

atribuirá o drive C: no server da rede, nodo 01, ao drive lógico H:. Isto significa que um utilizador no nodo onde este comando



foi dado, pode referir-se ao dispositivo H: e aceder de facto drive C: no server. Isto pode ser feito para qualquer dispositivo, quer se trate de drives de disco ou impressoras.

```
LOCAL     Passa o utilizador para o seu ambiente local
CPNETSTS  Imprime o estado da rede e as atribuições
          actuais no nodo do utilizador. Por exemplo:

          A>CPNETSTS

          CP/NET 1.2 Status
          ================
          Requester ID = 02H
```



```
Network Status Byte = 14H
Disk Device Status
Drive A: = LOCAL
Drive B: = LOCAL
Drive C: = Drive A: on Network Server ID = 01H
Drive D: = LOCAL
Drive E: = Drive B: on Network Server ID = 01H
Drive F: = LOCAL
Drive G: = LOCAL
Drive H: = Drive C: on Network Server ID = 01H
Drive I: = LOCAL
Drive J: = LOCAL
Drive K: = LOCAL
Drive L: = LOCAL
Drive M: = LOCAL
Drive N: = LOCAL
Drive O: = LOCAL
Drive P: = LOCAL
Console Device = LOCAL
List Device = List #0 on Network Server ID = 01H
```

Esta tabela indica-nos que quando nos referimos aos drives A, B, D, F, G, I, J, K, L, M, N, O ou P, estes são drives locais — pertencentes ao sistema do utilizador. No entanto, quando nos referimos aos drives C, E ou H, estes são de facto os drives A, B e C no server da rede. O dispositivo de impressão é de facto a impressora #0 no server da rede e não uma impressora local. Para o utilizador, todos estes dispositivos são endereçados como se fossem dispositivos locais. Isto significa que no caso de um utilizador, por exemplo no nodo 02, apresentar o comando

```
A>H:DBASE
```

o sistema carregará o package DBASE que se encontra no drive C: do server. Isto deve-se ao facto de o seu drive H: estar designado como drive C: do server. Por outro lado, todas as saídas do seu programa de base de dados serão enviadas para a impressora #0 do server.

```
MAIL  Permite a transmissão de mensagens através da rede
      entre nodos.
```

APÊNDICE A

CÓDIGOS DE CARÁCTER ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
000	00H	NUL	043	2BH	+	086	56H	V
001	01H	SOH	044	2CH	,	087	57H	W
002	02H	STX	045	2DH	-	088	58H	X
003	03H	ETX	046	2EH	.	089	59H	Y
004	04H	EOT	047	2FH	/	090	5AH	Z
005	05H	ENQ	048	30H	0	091	5BH	[
006	06H	ACK	049	31H	1	092	5CH	\
007	07H	BEL	050	32H	2	093	5DH	]
008	08H	BS	051	33H	3	094	5EH	^
009	09H	HT	052	34H	4	095	5FH	_
010	0AH	LF	053	35H	5	096	60H	`
011	0BH	VT	054	36H	6	097	61H	a
012	0CH	FF	055	37H	7	098	62H	b
013	0DH	CR	056	38H	8	099	63H	c
014	0EH	SO	057	39H	9	100	64H	d
015	0FH	SI	058	3AH	:	101	65H	e
016	10H	DLE	059	3BH	;	102	66H	f
017	11H	DC1	060	3CH	<	103	67H	g
018	12H	DC2	061	3DH	=	104	68H	h
019	13H	DC3	062	3EH	>	105	69H	i
020	14H	DC4	063	3FH	?	106	6AH	j
021	15H	NAK	064	40H	@	107	6BH	k
022	16H	SYN	065	41H	A	108	6CH	l
023	17H	ETB	066	41H	B	109	6DH	m
024	18H	CAN	067	42H	C	110	6EH	n
025	19H	EM	068	43H	D	111	6FH	o
026	1AH	SUB	069	44H	E	112	70H	p
027	1BH	ESC	070	45H	F	113	71H	q
028	1CH	FS	071	46H	G	114	72H	r
029	1DH	GS	072	47H	H	115	73H	s
030	1EH	RS	073	48H	I	116	74H	t
031	1FH	US	074	49H	J	117	75H	u
032	20H	SPC	075	4AH	K	118	76H	v

033	21H	!	076	4BH	L	119	77H	w
034	22H	"	077	4CH	M	120	78H	x
035	23H	#	078	4DH	N	121	79H	y
036	24H	$	079	4EH	O	122	7AH	z
037	25H	%	080	4FH	P	123	7BH	{
038	26H	&	081	50H	Q	124	7CH	|
039	27H	'	082	51H	R	125	7DH	}
040	28H	(	083	52H	S	126	7EH	~
041	29H	)	084	53H	T	127	7FH	DEL
042	2AH	*	085	54H	U			

Nota: Os primeiros 32 códigos ASCII podem ser gerados no teclado como «códigos de controlo», isto é, como códigos produzidos carregando na tecla «CONTROL» em combinação com outras teclas. O código 000 é «CONTROL», o código 001 é «CONTROL-A», o código 002 é «CONTROL-B» e assim por diante. Os códigos 028 a 032 são «CONTROL-SHIFT», e K, L, M, N, O respectivamente.

APÊNDICE B

COMANDOS DE EDIÇÃO DE CONSOLA CP/M

Estes comandos podem ser usados na maioria dos micro-computadores para editar comandos CP/M. Os seguidos de (3) estão geralmente apenas disponíveis em CP/M Plus. O leitor poderá experimentar se funcionam na máquina que usa e para a versão de CP/M que utiliza.

Comando	Efeito
CTRL-A	Desloca o cursor um caracter para a esquerda (apenas sistemas com bancos de memória) (3)
CTRL-B	Desloca o cursor através da linha de comandos (apenas sistemas com bancos de memória) (3)
CTRL-C	Pára a execução do programa. Devolve o utilizador ao nível de comando
CTRL-E	Força um retorno de linha sem enviar o comando para CP/M
CTRL-F	Desloca o cursor um caracter para a direita (apenas sistemas com bancos de memória) (3)
CTRL-G	Elimina o caracter na posição do cursor se não for final de linha (apenas sistemas com bancos de memória) (3)
CTR1-H	Elimina o caracter para a esquerda do cursor
CTRL-I	Tabula para a direita (3)
CTRL-J	Mudança de linha
CTRL-K	Elimina o caracter sob o cursor e todos os caracteres para a sua direita (3)
CTRL-M	Retorno de linha
CTRL-P	Accionar/parar impressora
CTRL-Q	Reinicia o ''scrolling'' do visor depois de CTRL-S (3)
CTRL-R	Reescreve os caracteres à esquerda do cursor numa linha nova

CTRL-S	Pára o ''scrolling'' do visor (no caso das versões 2.2 ou anteriores o ''scrolling'' recomeça ao carregar em qualquer tecla)
CTRL-U	Elimina a linha actual. Actualiza o buffer da linha de comandos de modo a conter os caracteres à esquerda do cursor
CTRL-W	Invoca a linha de comandos anterior se a linha actual estiver vazia (apenas sistemas com bancos de memória) (3)
CTRL-X	Elimina todos os caracteres à esquerda do cursor

APÊNDICE C

PACKAGES CP/M

O número de packages de software disponíveis para os sistemas CP/M é muito grande e aumenta cada dia que passa. Segue-se uma lista de alguns dos packages mais populares, juntamente com uma breve descrição daquilo que fazem. Apresentamos as nossas desculpas a qualquer «software house» cujos produtos não tenham sido incluídos e se sinta lesada. Tentámos apresentar uma lista actualizada.

WORDSTAR	Package de tratamento de texto. Potente e, depois de algum treino, fácil de usar
MAIL-MERGE	Package de desenvolvimento do WORDSTAR. Lista de correio e personalização de cartas
SPELLSTAR	Acrescento ao WORDSTAR. Verifica a correcção da escrita
STARINDEX	Acrescento ao WORDSTAR. Cria índices, tabelas de conteúdo e documentos em formatos normalizados
DATASTAR	Entrada e recuperação de dados pelo visor
REPORTSTAR	Gerador de relatórios usado juntamente com DATASTAR
INFOSTAR	Sistema de gestão de base de dados usado juntamente com REPORTSTAR e DATASTAR
CALCSTAR	Package de folha de cálculo
PROSTAR	Package de treino para os produtos já indicados
SUPERSORT	Package de ordenação e mistura de ficheiros
Dbase II	Sistema de gestão de bases de dados. Inclui relatórios. Apropriado para facturações, controlo de stocks, sistemas de salários
QUICKCODE	Fácil criação de ficheiros de comandos Dbase II

dGRAPH	Extra dados de registos Dbase II em forma gráfica
DMS	Sistema de gestão de base de dados
THE LAST ONE	Gerador de código para MBASIC
CARDBOX	Sistema de recuperação de informação
MULTIPLAN	Folha de cálculo electrónica
SUPERCALC	Folha de cálculo electrónica
MICROMODELLER	Package de planeamento financeiro
BASIC-80 COMPILER	Compilador BASIC para execução mais rápida
FORTRAN-80	Compilador ANSI FORTRAN 66
COBOL-80	Compilador ANSI COBOL 74
CIS-COBOL	Compilador ANSI 74. Suporta ficheiros indexados sequenciais, funcionamento interactivo
LEVEL II COBOL	Utilizado com FORMS2, FILESHARE e ANIMATOR
PASCAL/MT+	Compilador Pascal
CBASIC	Package compilador e de suporte à execução para linguagem BASIC
PL1/80	Subconjunto G do compilador
''C'' COMPILER	Compilador de duas passagens
BSTAM	Permite a troca de ficheiros entre sistemas CP/M
XLT-86	Converte Assembler 8080 para código 8086
PERFECT WRITER	Package de processamento de texto
PERFECT SPELLER	Verifica erros de escrita em ficheiros PERFECT WRITER
PERFECT CALC	Folha de cálculo electrónica
PERFECT FILER	Sistema de recuperação de dados
T/Maker	Package combinado de tratamento de texto, folha de cálculo e manutenção de ficheiros
SAGE	Sistema de contabilidade integrada
Pulsar	Package financeiro incluindo capacidades de modelação

ÍNDICE