OSBORNE / McGRAW-HILL

SEGUNDA EDIÇÃO REVISADA

MSX

GUIA DO USUÁRIO

Adaptação

Luis Sérgio Young Moreira
Oscar Julio Burd

McGraw-Hill

Paul Hoffman

MSX
GUIA DO USUÁRIO

paul hoffman

MSX
GUIA DO USUÁRIO
paul hoffman

Tradução
Lars Gustav Erik Unonius
Engenheiro Eletrônico

Revisão Técnica
Oscar Júlio Burd
Luiz Sérgio Young Moreira

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires • Guatemala • Madrid • México • New York • Panamá • San Juan • Santiago

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Do original

The MSX Book

Coordenadora de Revisão: Daisy Pereira Daniel
Capa: Layout: Cyro Giordano
Arte-final: Jaime Marques

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Hoffman, Paul.
H647m MSX : guia do usuário / Paul Hoffman ; tradução Lars Gustav Unonius ; revisão técnica
2ª ed. Oscar Júlio Burd, Luiz Sérgio Young Moreira. – São Paulo : McGraw-Hill, 1987.

1. MSX (Computadores) I. Título.

86-0356 CDD-001.64

Índices para catálogo sistemático:

1. MSX : Computadores : Processamento de dados
001.64

AGRADECIMENTOS

Gostaria de agradecer a muitas pessoas pela ajuda em fazer deste livro um sucesso: Phil Moon da Yamaha emprestou-me um computador MSX durante a elaboração deste livro. John Sabol e Chris Larson da Microsoft Corporation também emprestaram-me partes do computador e forneceram informações valiosas de mercado. Yuuji Kitamura ajudou-me na aquisição de peças do MSX no Japão e explicou como os japoneses vendem e utilizam os computadores domésticos. Tamara Nicoloff revisou muitos dos capítulos referentes ao MSX-BASIC e forneceu sugestões valiosas de como explicar programação aos usuários iniciantes em computadores. David Richie forneceu ajuda e idéias para muitos dos exemplos do livro, apesar de não gostar de BASIC. Denise Penrose da OSBORNE/McGRAW-HILL também forneceu suporte razoável durante a elaboração do livro.

Paul Hoffman

SUMÁRIO

INTRODUÇÃO

Este livro será de grande utilidade para todos aqueles que possuam um computador MSX, ou que estejam pensando em comprar um. Contém informação de como utilizar o computador, de como programar o computador em MSX-BASIC, e, caso você tenha um acionador de disco, como utilizar o MSX-DOS.

Este livro responde às perguntas feitas por muitos dos usuários do MSX: O que posso fazer além de jogar? É fácil programar em MSX-BASIC? Para que serve o MSX-DOS?

Apesar de o livro ser escrito com a finalidade de tornar compreensível o uso do MSX, principalmente pelo usuário primário, ele também contém muita informação útil ao usuário adiantado. Caso você tenha ficado confuso com seu manual, este livro o ajudará a compreender muitos dos conceitos apresentados.

O MSX é um padrão para computadores, e a maioria dos computadores MSX se parecem e operam da mesma maneira. Alguns fabricantes adicionaram características aos seus MSX para torná-los mais interessantes, mas todos aderiram ao padrão de modo que a *informação deste livro se aplica a todos os computadores MSX.*

Por que Você Deve Ler este Livro

Seja você um usuário novo do MSX ou um que o tenha utilizado durante muitos anos, poderá não estar familiarizado com todas as funções do MSX-BASIC e MSX-DOS. Este livro fornecerá uma introdução branda, porém direta, a estas funções. O livro tem ainda diversas outras características importantes, incluindo:

– *Introdução aos computadores para os iniciantes.* No início do livro existem descrições não técnicas do que é um computador e como começar a utilizá-lo. Você verá que quanto mais souber a respeito da operação de seu computador MSX, maior utilidade ele terá para você. Os Capítulos 1 e 4 são dedicados a estes itens.

– *Uma visão do mercado do MSX.* Existem centenas de programas úteis que podem ser processados pelo seu computador MSX, e existem diversos componentes que podem ajudar seu computador MSX a controlar dispositivos em sua casa. Estes tópicos são discutidos nos Capítulos 2 e 3.

– *Uma lista completa de todos os comandos do MSX-BASIC.* Estes comandos são discutidos por grupos funcionais, de maneira que seja possível determinar com rapidez a relação entre comandos. São fornecidos programas que exemplificam o alcance destes comandos. O MSX-BASIC é abordado do Capítulo 5 ao 16.

– *Uma introdução em profundidade ao MSX-DOS.* Os iniciantes normalmente ficam confusos sobre o que são discos e arquivos, de modo que existe um capítulo bastante extenso que abarca todos os conceitos existentes no MSX-DOS. (Lembre-se de que este material tem utilidade apenas para aqueles que possuem um acionador de disco.) O MSX-DOS é discutido do Capítulo 17 ao 19.

– *Tabelas para referência-rápida.* Os apêndices têm diagramas de todos os comandos MSX-BASIC e MSX-DOS, bem como uma tabela dos caracteres que o computador MSX pode exibir na tela do vídeo. Existe ainda uma discussão a respeito dos aspectos técnicos do padrão MSX.

A Origem do Material deste Livro

Apesar de o padrão MSX ser relativamente novo, é baseado em padrões estabelecidos há muitos anos. O MSX é licenciado aos fabricantes de computadores pela Microsoft Corporation, uma das maiores empresas de software para microcomputadores dos Estados Unidos. As partes do MSX estão baseadas em padrões de outros produtos também produzidos pela Microsoft.

O MSX-BASIC é estruturado da mesma forma que o MBASIC e do BASIC do IBM PC; o MSX-DOS é estruturado da mesma que o MS-DOS. Assim, grande parte do material deste livro é adaptado diretamente de um livro também publicado pela Osborne/McGraw-Hill:

– *The Osborne/McGraw-Hill MS-DOS User's Guide*, de Paul Hoffman e Tamara Nikoloff (1984).

Outro livro da Osborne/McGraw-Hill que contém maiores detalhes sobre programação em BASIC do que este livro, e poderá ser útil caso queira continuar o aprendizado a respeito do MSX-BASIC é *The MBASIC Handbook*, de Walter A. Ettlin e Gregory Solberg (1983). Todos os programas deste livro foram rigorosamente testados em um computador MSX.

É claro que qualquer informação que seja diferente, para o MSX, daquela apresentada pelo livro original, é completamente revisada aqui. Por exemplo, o material dos Capítulos 1 ao 4 é completamente novo, e refere-se apenas aos computadores MSX.

PARTE

1

Seu Computador MSX

CAPÍTULO

1

DESCRIÇÃO DE UM COMPUTADOR MSX

Mesmo que você não tenha conhecimento algum sobre o que existe dentro do computador ou como ele opera, é fácil jogar e realizar diversas tarefas com o computador MSX. Entretanto, você verá que é muito mais fácil fazer o que você quer se entender um pouco sobre o funcionamento dos computadores MSX.

Neste capítulo são descritas algumas das partes essenciais de seu computador MSX, sem entrar em detalhes técnicos. Ao terminar este capítulo, você terá uma boa noção sobre o que existe dentro do computador e como ele processa um programa.

Uma vez que o computador MSX tem características similares às de diversos outros computadores, domésticos e comerciais, este capítulo lhe dará informações sobre computadores em geral, utilizando o computador MSX como um exemplo específico. Como já fora mencionado na introdução deste livro, todos os computadores MSX têm muito em comum, mas apresentam também diferenças nas características e na aparência. Este livro descreve os computadores MSX em geral, e não uma determinada marca ou modelo.

Partes de um Computador

Um computador tem dois componentes básicos: *hardware* e *software*. Simplificadamente, hardware é o nome dado à parte física do computador, enquanto software é o nome dado ao conjunto de programas processados pelo computador

HARDWARE

Muitas pessoas têm receio de nunca entender os computadores, porque não entendem eletrônica. Este temor não se justifica e equipara-se àquele de não saber cozinhar por nada entender sobre química orgânica. Na realidade, um conhecimento sobre eletrônica em nada ajuda quando você aprende os princípios básicos dos computadores MSX.

Quando você retira o computador da caixa pela primeira vez, observará que muitas partes se assemelham a outros objetos com os quais já está familiarizado. A Figura 1.1 mostra um computador MSX típico (neste caso, um Yamaha CX5M). A parte principal da frente do computador, *o teclado*, se assemelha muito aos teclados da maioria das máquinas de escrever eletrônicas.

Próximo ao teclado existem quatro teclas grandes com setas apontando para cima, para baixo, para a esquerda e para a direita. São conhecidas por *teclas de controle do cursor*, já que são utilizadas para movimentar o cursor pela tela. (O *cursor* identifica o local, na tela, em que irá aparecer o próximo caractere que for teclado. É descrito no Capítulo 6.) Existem ainda teclas que apresentam identificação do tipo F1 ou F2, denominadas *teclas de função*, assim como teclas com identificação do tipo HOME ou INS. Estas teclas são utilizadas ao ser processado um programa.

Ao lado ou atrás do computador, existem diversos conectores para a ligação de cabos. Parecem-se com alguns dos plugues encontrados na parte traseira dos equipamentos de som ou televisores. Nos computadores, estes plugues são muitas vezes chamados *portas*. Provavelmente existe uma chave liga/desliga, que sem dúvida liga e desliga o computador. Muitos computadores MSX são acompanhados dos cabos que se ligam às portas do computador.

Figura 1.1 Um computador MSX típico: o computador musical CX5M da Yamaha.

Alguns computadores MSX apresentam *luzes de indicação* próximas ao teclado, normalmente perto das teclas CAPSLOCK ou da chave liga/desliga. Estas luzes são semelhantes aos indicadores de um estéreo: informam quando alguma coisa está "ligada".

Algumas das outras partes que você vê podem não ser familiares, mas operam de um modo fácil de se entender. Em todo computador MSX existe um local para inserção de um *cartucho*, normalmente na parte superior do computador acima do teclado. Consiste em um retângulo de 19 cm × 1,5 cm, que pode estar coberto por uma tampa com dobradiça. Como o nome já está

dizendo, é o local em que são inseridos os cartuchos. Muitos computadores MSX têm local para colocação de dois cartuchos. Poderão vir junto com o computador um ou mais cartuchos contendo programas, ou você poderá comprar cartuchos para aumentar a utilidade de seu computador MSX. O cartucho é inserido da mesma forma que uma fita cassete é colocada em um toca-fitas. Você "processa" os programas residentes no cartucho da mesma forma que você toca música gravada em um cassete.

Seu computador MSX poderá vir acompanhado de um *monitor*, que se parece com uma tela de televisão. Se você não recebê-lo com seu sistema, será necessário ligar o computador em um monitor ou um aparelho normal de tevê para que você possa ser aquilo que estiver teclando ou a "saída" dos programas à medida em que são processados. O procedimento para esta montagem é descrito no manual do usuário e é discutido também no Capítulo 4 deste livro.

Seu computador poderá vir acompanhado por outros componentes de hardware que você liga ao computador. São denominados *periféricos* e poderão estar ligados ao computador. Alguns periféricos comuns são manipuladores para jogos, impressoras e acionadores de discos. Estes e outros periféricos são descritos no Capítulo 3, e os periféricos específicos de seu computador são descritos no manual que o acompanha.

O termo *hardware* também engloba a eletrônica dentro de seu computador. Mais adiante, neste capítulo, você aprenderá mais sobre o tipo de hardware que existe dentro do MSX e um pouco do seu funcionamento (repetindo, não haverá necessidade de conhecimento sobre eletrônica). Porém, você nunca terá de se preocupar com o hardware dentro de seu computador, uma vez que nunca o verá nem o tocará. Toda a interação com o hardware de seu computador MSX é feita através de componentes externos, especialmente o teclado.

SOFTWARE

A descrição do software é muito mais difícil do que a descrição do hardware, já que você não pode tocar ou ver estes programas em seu computador. Um programa é o conjunto de instruções que dizem ao computador o que fazer. Os termos "programas" e "software" são utilizados, com freqüência, indistintamente, sendo que algumas vezes software refere-se a um conjunto de programas. Também existem diversos tipos de software: aquele que controla seu computador é bastante complexo, mas alguns outros são tão simples que você pode aprender a escrevê-los em alguns minutos. O nível de dificuldade depende da dificuldade da tarefa que estiver sendo programada.

Algum software pode ser "aprendido" pela leitura de programas escritos em uma linguagem de programação. Por exemplo, muitas listagens de programação em MSX-BASIC aparecem na parte 2 deste livro. Você pode teclar estes programas no computador, mas assim que o software estiver dentro do computador transforma-se em pequenos pulsos elétricos que não podem ser vistos, mas são compreendidos apenas pelo computador.

Seu computador MSX vem com dois elementos de software dentro dele: o sistema operacional e o MSX-BASIC. O sistema operacional controla o hardware de seu computador e torna simples a tarefa de processar outros programas no computador. Não se preocupe em entender o sistema operacional do MSX, já que os únicos que precisam conhecê-lo são aqueles que estão escrevendo programas complexos.

O sistema operacional e o MSX-BASIC estão armazenados dentro do computador em circuitos integrados, denominados ROM. ROM significa "Ready Only Memory" (Memória Apenas de Leitura), que representa uma maneira estranha de informar que os programas nos integrados são permanentes e não podem ser alterados. Se você comprar software em cartuchos, como os jogos ou um software comercial, verá que estes também são armazenados em integrados ROM.

À medida que você aprende o BASIC-MSX, fará seus próprios programas. É claro que você poderá guardar os programas que escreve de modo que não seja necessário reescrevê-los cada vez que ligar o computador. Mais adiante verá como armazenar programas em fita (utilizando um gravador cassete) ou em disco (utilizando o acionador de disco).

Alguns fornecedores incluem software adicional com seus computadores. Poderá ser software para jogos, para atividades comerciais, educacionais, ou outro tipo de software específico àquela máquina em particular. Este software freqüentemente vem em um cartucho, mas às vezes está incluído dentro do computador em uma ROM ou em fita cassete.

Dentro de seu Computador

Agora que você conhece um pouco sobre hardware e software, está pronto para conhecer a parte interna de seu computador MSX. Como já foi dito, todos os computadores MSX têm características similares, e com freqüência são muito semelhantes. A discussão que segue diz respeito aos integrados internos comuns a todo computador MSX.

Você já viu que a unidade principal inclui o teclado, local para inserção de cartucho(s) e diversos plugues para outras partes de seu computador. Apesar de não ser possível abrir o computador, você deve saber que existem diversas coisas dentro da unidade principal que afetam os programas que são processados e outros itens de hardware que são comprados.

Como deve ser de seu conhecimento, os computadores são formados por "integrados" interligados de modo a permitir a passagem de um sinal eletrônico entre eles. A maior parte destes integrados existentes no MSX não são interessantes, mas alguns são bem importantes. Felizmente, os que são importantes são fáceis de serem descritos. Todo computador MSX tem o seguinte hardware em seu interior:

- Uma UPC Z-80A (unidade de processamento central). Todo computador tem uma UPC, que é seu "cérebro". O "Z-80A" é o tipo de UPC que existe no computador MSX; outros tipos, comuns em outros computadores, são "6502", "8088" e "68000". A UPC do MSX pode processar 3.58 milhões de instruções por segundo, mais rápido que muitos outros computadores pessoais.
- Memória RAM. Anteriormente você leu a respeito da ROM, que armazena o sistema operacional do MSX e o MSX-BASIC. Não é possível escrever na ROM; não é possível alterar o conteúdo da ROM, de tal modo que ela não tem utilidade nos programas que podem ter suas informações ou instruções alteradas ou acrescentadas. Por outro lado, a RAM é uma memória de uso geral, uma vez que ela pode ser lida (processar o programa) e escrita (modificada). Porém, qualquer informação que estiver armazenada em uma RAM é "esquecida" quando você desliga o computador.

Quanto mais RAM tiver, maior poderá ser seu programa. Seu computador provavelmente tem "16K RAM", "32K RAM", ou "64K RAM". O "K" representa 1024 unidades de memória, de modo que quanto mais "K" você tiver, mais memória terá.

- Integrados especiais para gráficos. As imagens geradas pelo seu computador são controladas por um integrado especial. Isto permite que a UPC ocupe uma maior parte do tempo "pensando" a respeito de seu programa. Com um integrado específico para a geração de gráficos o seu programa pode ser processado mais rapidamente.
- Integrado especial para geração de sons. Assim como o integrado para geração de gráficos, o integrado para geração de sons também é bem avançado; permite ao seu computador gerar facilmente sons complexos.

A maioria dos demais integrados existente na unidade principal do computador apenas ajuda aqueles listados, na execução de suas tarefas.

MSX-DOS

Alguns computadores MSX vêm acompanhados com os acionadores de disco, enquanto outros têm acionadores que podem ser comprados separadamente. As vantagens de possuir um acionador de disco são: maior rapidez no armazenamento dos programas; maior espaço para o armazenamento dos programas, e um acesso mais rápido dos dados, para programas comerciais ou educacionais.

Para que seja possível utilizar um acionador de disco, você deve ter também o MSX-DOS. Este normalmente é fornecido com os acionadores de disco projetados para os computadores MSX. O programa MSX-DOS vem em disco e é colocado no acionador de disco. O MSX-DOS é discutido dos Capítulos 17 ao 19.

Para que seja possível utilizar o MSX-DOS, você deverá ter 64K de RAM. Isto se deve ao fato de que o MSX-DOS, como outros programas, exige uma certa quantidade de memória para seu funcionamento. Você não pode utilizar o MSX-DOS em um computador que tenha menos do que 64K de RAM. Se o computador não tiver a quantidade suficiente de RAM, você poderá expandir a quantidade de RAM através da aquisição de cartuchos adicionais de RAM (veja no Capítulo 3).

Para utilizar o MSX-DOS, você deverá ainda interligar o acionador de disco ao seu computador. (Você bem que pode imaginar a dificuldade de sua utilização caso não estivesse conectado.) Entretanto, isto pode apresentar um problema já que a maioria dos acionadores de disco é conectada pelo conector de inserção do cartucho. Se você tiver apenas um conector de inserção, não será possível ligar um acionador de disco e um outro cartucho. Muitos computadores MSX têm mais do que um conector ou têm caixas para expansão que permitem a expansão do número de conectores.

CAPÍTULO

2

O QUE É POSSÍVEL REALIZAR COM UM COMPUTADOR MSX

Este capítulo é um guia para os tipos de programas que você pode comprar para o seu computador MSX. Descreve, de um modo generalizado, o alcance do software disponível e o que pode ser esperado de alguns dos programas. Uma vez que programas novos são escritos diariamente e os antigos estão constantemente sendo melhorados, listamos aqui apenas alguns programas específicos. Em revistas sobre computadores domésticos você poderá encontrar listas sobre os programas disponíveis. Algumas das grandes empresas distribuidoras de software, como Lifeboat, têm muitos produtos de software para o MSX.

Uma Visão do Software do MSX

A maior parte do software do MSX vem em cartuchos, uma vez que todo computador MSX dispõe de espaço para pelo menos um cartucho. Os cartuchos são uma maneira bem conveniente de distribuir o software uma vez que são difíceis de danificar e muito difíceis de serem copiados por piratas. Alguns fabricantes de software distribuem programas em cassete ou em discos flexíveis.

Ao comprar um software esteja seguro de que você possa utilizá-lo. Por exemplo, se você não tiver um gravador cassete associado ao seu computador MSX, não compre um programa armazenado em cassete. Caso tenha um acionador de disco, poderá comprar software em disquete. Assegure-se de que o software é compatível com seu acionador de disco; que os disquetes podem ser utilizados com seu modelo e tamanho de acionador. Quando existir dúvida, pergunte ao vendedor de software.

O MSX é de utilização tanto doméstica como educacional e comercial, e o software disponível mostra esta distinção. Por exemplo, existem, disponíveis, diversos programas para processamento de palavras; alguns são apropriados para utilização caseira – datilografia de tarefas domésticas, correspondência em geral, ou projetos especiais – enquanto outros incluem características bastante poderosas e necessárias em um escritório ou escola. É claro que você pode

utilizar o programa onde desejar, mas antes de fazer a compra convém perguntar sobre a finalidade a que se destina.

Uma vez que o computador MSX é popular no Japão e na Europa, a mais tempo do que está disponível nos Estados Unidos, uma parte do software de qualidade do MSX foi convertido ao mercado americano. Entretanto, o sistema operacional do MSX é similar ao sistema operacional CP/M, e muitos programas americanos em CP/M foram convertidos para serem processados no MSX. À medida que os computadores MSX se tornam mais populares nos Estados Unidos, veremos aumentar o software ali escrito para o MSX.

Jogos

A utilização doméstica mais popular do computador MSX são os jogos. Existem dezenas (ou centenas) de jogos disponíveis para os computadores MSX, a maioria em cartuchos.

A maioria dos jogos do MSX pode ser classificada na categoria dos jogos arcádicos (de fliperama) ou de raciocínio. Muitas pessoas não gostam dos jogos tipo fliperama, em virtude de sua violência e natureza repetitiva. Entretanto, é fácil evitar aqueles jogos de "atire-neles" (ou "coma-os"); se você não se sentir atraído pelos jogos arcádicos, pode ser que ainda encontre interesse em algum jogo de raciocínio.

Caso você não tenha conhecimento sobre os jogos de ação de computadores, leve em conta as seguintes considerações:

- Não julge um jogo pela embalagem. A arte executada na embalagem pode ser bonita, mas lembre-se de que é mais barato fazer uma embalagem bonita do que um jogo interessante. É mais caro conseguir um bom programador do que conseguir um bom desenhista.
- Não espere mais do que já viu nos fliperamas (de fato, provavelmente deverá esperar bem menos). Caso não goste de nenhum dos jogos de fliperama, certamente não irá gostar dos jogos arcádicos disponíveis para os computadores MSX. Muitos dos jogos arcádicos do MSX são cópias conceituais dos jogos de fliperama.
- Pense em acrescentar um joystick ao seu computador. O joystick é um periférico que controla determinadas ações da tela. Normalmente, apertamos os botões do joystick para movimentar objetos ou executar uma ação. Muitos dos jogos arcádicos que requerem agilidade ou velocidade são extremamente difíceis de se jogar sem um controlador de jogos. A movimentação de objetos pela tela utilizando as teclas de controle do cursor em vez de um controlador pode-se tornar muito difícil nestes jogos.
- Experimente o jogo antes de comprá-lo. Lembramos, porém, que na maioria das lojas de computadores você não terá permissão para jogar, na loja.

JOGOS ARCÁDICOS (FLIPERAMA)

A maioria dos jogos arcádicos processados no MSX não é muito interesante, mas é popular. Alguns jogos do tipo arcádico para o MSX são "Mouser" (animais perspicazes e escadas), "Car Jamboree" (uma corrida de automóvel com muitos elementos de demolição), "Pitfall!" (cor-

rida na selva), "Keystone Kapers" (policiais e ladrões), "Hyper Olympics" (simulação esportiva), e "Sparkie" (um jogo alegre em um labirinto).

Muitos jogos populares nos fliperamas foram convertidos para utilização doméstica, com muitos títulos disponíveis aos computadores MSX. Esportes (como baseball, golfe e tênis) são outros tópicos muito populares para os jogos do MSX, mas, raramente os jogos esportivos de vídeo são tão interessantes como os jogos esportivos em si.

Alguns jogos de habilidade são também ferramentas de aprendizado muito bons. A simulação de vôo faz você monitorar os diversos discos e luzes típicos de uma pequena nave e responder utilizando controles similares aos de um avião real. Você não marca pontos, como normalmente ocorre com um jogo; em vez disto o objeto tem de decolar e aterrar sem que haja um acidente.

JOGOS DE RACIOCÍNIO

É difícil comparar os jogos de raciocínio com alguma coisa familiar. Alguns são descritos como "ficção interativa", enquanto outros são "Jogos de aprendizado". Parecem-se com histórias que você ajuda a criar, porque você descreve para onde deve ir o personagem principal e o que ele deve fazer.

Os jogos não-arcádicos mais interessantes são os de aventura, onde você resolve mistérios e encontra tesouros pela utilização de palavras que permitem a sua movimentação por um certo ambiente. Por exemplo, se o programa informa que você está nas proximidades de uma caverna e, através do teclado, você lhe diz que quer entrar na caverna, o programa lhe informa o que encontrará lá dentro. Você poderá fazer diversas coisas (achar tesouros, matar dragões etc.), mas a finalidade do jogo é solucionar um mistério ou alcançar um objetivo.

No mundo dos jogos de aventura, os jogos da Infocom são considerados os mais interessantes (incluem a série "Zork", "Planetfall" e "The Hitchhiker's Guide to the Galaxy"). Os cenários variam desde a fantasia do dragão e calabouço, ficção científica e histórias detetivescas comuns. Os jogos Infocom são conhecidos pelo humor e pela intriga. Sua interação com o jogo é bastante realista uma vez que lhe é permitido fazer perguntas e fornecer listas de ordens, em inglês.

Os jogos Infocom fornecem um texto na tela, descrevendo o ambiente em que você se encontra. Existem outros jogos de aventura que apresentam uma imagem para cada ambiente. Por exemplo, se você estiver sentado em uma sala vitoriana, veria uma imagem desta sala; e se um morcego entrasse voando pela janela da sala, o morcego apareceria na tela.

Outros jogos de raciocínio disponíveis para o MSX incluem conversões de jogos populares de tabuleiro. Por exemplo, existem versões MSX de Othello, gamão e mah-jongg. Estão disponíveis ainda programas de jogos de cartas e de xadrez.

Processamento de Palavras

O software de processamento de palavras permite que você escreva e armazene qualquer tipo de texto – memorandos, cartas, relatórios e livros – e imprima o texto com uma impressora. A maioria das pessoas acha que o processamento de palavras é o programa aplicativo mais útil

uma vez que torna o ato de escrever mais simples. Depois de você utilizá-lo, verá que o processamento de palavras é uma ferramenta muito valiosa. Mesmo que o computador tenha sido adquirido com outra finalidade deve ter um bom processador de palavras.

À medida que os programas de processamento de palavra se tornam mais comuns, maior número de pessoas está sentindo a diferença entre utilizar um processador de palavras e uma máquina de escrever. A revisão de texto, com um processador de palavras, é muito simples; a reestruturação de uma carta ou nota é rápida, e guardar o texto no disco ou cassete do computador significa que você nunca precisará refazer a datilografia.

MANUAL DO PROCESSADOR DE PALAVRAS

Você deve ter a facilidade para imaginar um programa que mostra na tela os caracteres digitados à medida que estiverem sendo teclados. Com exceção da tela, o programa se comporta como uma máquina de escrever comum. Você pode imaginar também que à proporção que você tecla os caracteres, o programa armazena-os em disco ou fita; posteriormente pode ser dado um comando para imprimir estes caracteres com uma impressora. Até aqui, esta descrição coincide com uma máquina de escrever de memória: uma vez teclado, é sempre possível imprimir.

A semelhança termina quando você quiser alterar o que foi teclado. Com um processador de palavras é simples incluir ou excluir palavras, sentenças e parágrafos. É possível movimentar sentenças e parágrafos pelo texto que foi criado. Ao fazer estas alterações, o programa permite que você faça a edição do texto armazenado. Outra característica do processador de palavras é que ele permite que se dê um determinado formato ao documento (numeração automática de página, parágrafos, itálicos) quando estiver sendo impresso em papel.

Você fornece ao programa de processamento de palavras comandos que o informam o que fazer e quando. Com a maioria dos programas, você deve utilizar uma combinação de teclas para dar estes comandos (como movimentação para frente ou para trás no texto, para deletar, para inserir, e assim por diante). Os computadores MSX têm teclas especiais no teclado que tornam mais fácil o processamento de palavras.

Ao utilizar um processador de palavras você começa por informar ao programa qual é o texto existente no disco, ou no disquete, que deve ser editado, ou se deseja iniciar um novo texto. As cartas, capítulos ou artigos são guardados em arquivos, descritos com detalhes no Capítulo 18. Se estiver utilizando um arquivo novo, informe ao programa que deseja escrever um texto e comece a digitar. Se estiver editando um texto já escrito, informe ao programa o nome do arquivo que deve ser editado e onde está o texto (como, a primeira linha ou a segunda sentença no quarto parágrafo). Assim que o programa o posicionar, poderá fazer suas alterações. De fato, você se movimenta pelo arquivo todo desta maneira; alterando ou acrescentando texto. Uma das características interessantes da edição é que você pode movimentar-se para frente e para trás pelo arquivo; não é obrigado a ir sempre do começo ao fim.

Quando terminar de editar, ou criar um arquivo, você informará ao programa para gravar o arquivo novo (com todas as alterações) em um disco ou cassete. Se quiser, poderá, também, imprimir este arquivo novo. Após ter utilizado o programa de processamento de palavras para fazer edição de texto, provavelmente nunca voltará a utilizar uma máquina de escrever.

Geralmente, quanto mais potente é o processador de palavras, mais você consegue fazer com seus comandos de formatação. Existem diversos tipos de *comandos de formatação*. Alguns permitem que se faça coisas simples, típicas (como a colocação das margens esquerda e direita, numeração de páginas, ou marcação de um parágrafo), e há outros mais sofisticados que podem, por exemplo, fazer tarefas como ajustar automaticamente listas, centralizar títulos de capítulos, colocar um cabeçalho no topo de cada página, ou sublinhar automaticamente um grupo de palavras. A Figura 2.1 ilustra diversas destas características do processamento de palavras.

O QUE OBSERVAR EM UM PROCESSADOR DE PALAVRAS

Os programas de processamento de palavras estão disponíveis em uma faixa muito grande de preços. Por mais estranho que possa parecer, não podemos assumir que um pacote de processamento de palavras mais caro seja mais fácil de utilizar e tenha mais facilidades do que uma versão mais barata. Certos processadores de palavras mais baratos têm mais opções que alguns mais caros. Os critérios mais importantes são, em primeiro lugar, se um pacote é fácil de aprender a utilizar, e em segundo lugar, se o número de facilidades é suficiente para as suas necessidades.

O que segue são algumas considerações que devem ser feitas quando você começa a procurar pelo melhor programa:

- Qual é a facilidade de dar entrada, alterar, formatar e imprimir o texto? Você provavelmente vai querer evitar programas que exijam a memorização de seqüências complicadas para a teclagem dos comandos de edição. Esta verdade é mais acentuada quando você utiliza processamento de palavras apenas em casa.

- O processador de palavras pode operar com sua impressora? Algumas impressoras podem alterar o estilo de impressão (denominado fonte, utilizar espaçamento proporcional ou criar sobrescritos e subscritos. Entretanto, você poderá utilizar estas características apenas se seu processador de palavras foi configurado para o tipo de impressora que você tem. Em caso afirmativo, isto significa que a impressora tem condição de executar os efeitos solicitados pelo programa do processador de palavras. Fazemos este lembrete porque a maioria dos processadores de palavras é configurada apenas para algumas impressoras; se aquele que você comprar não for configurável para sua impressora, não será possível utilizar as características especiais do programa.

- O programa inclui a característica de elaboração de uma listagem para endereçamento postal? Muitos programas permitem que você junte um arquivo com endereçamentos postais a uma carta, formando cartas-padrões pela inserção de nome e endereço no topo de cada carta.

- O programa tem características para usuários antigos? Muitas pessoas que utilizam o processador de palavras regularmente tornam-se hábeis com o programa. Muitos pacotes têm características avançadas que, apesar de serem complicadas para iniciantes, fornecem mais flexibilidade aos veteranos. Por exemplo, alguns programas permitem que você solicite a execução de diversos comandos diferentes um após o outro, sem solicitação adicional de sua parte.

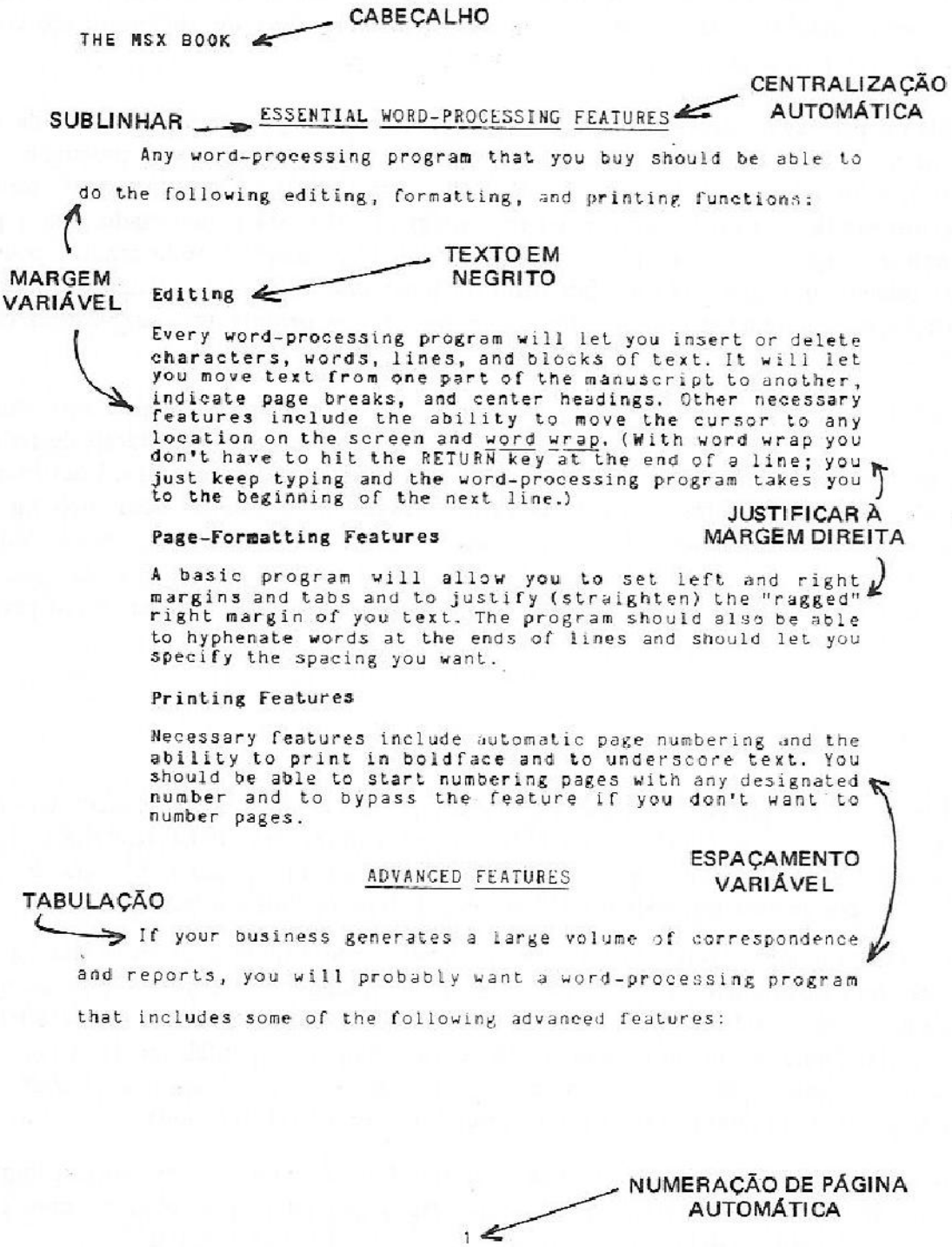


Figura 2.1 Algumas características do processamento de palavras.

- Qual é a facilidade de aprendizagem? Alguns programas vêm acompanhados por uma documentação farta, menus de auxílio na tela, gabaritos ou cartões que o auxiliem a aprender rapidamente o sistema. Outros programas têm documentação deficiente, teclagens estranhas para os comandos e outras características que impedem seu conforto com o programa.

Alguns pacotes de processamento de palavras incluem programas de verificação de escrita. Um verificador de escrita examina cada palavra em seu arquivo de texto, procura-a em seu dicionário, e lhe informa qual é que não reconhece. Em seguida permite que você decida se a palavra foi escrita de maneira errada, ou se o programa não está familiarizado com a palavra (por exemplo, siglas ou nomes próprios). Como você pode imaginar, todo arquivo poderá ter diversas palavras que não estão no dicionário do programa; assim, é especialmente importante a obtenção de um programa de verificação de escrita que permita uma atualização fácil do dicionário.

Grande parte dos pacotes de processamento de palavras do MSX é para uso comercial, existindo porém, pacotes para uso doméstico. Estes são normalmente mais fáceis de ser usados já que contêm menor número de características do que os pacotes comerciais. Com freqüência você pode aprender a utilizar um sistema de processamento de palavras doméstico em menos de uma hora. O sistema, usualmente, utiliza-se das possibilidades gráficas e sonoras do MSX com maior vantagem que os sistemas de processamento de palavras comerciais. Por exemplo, o programa Bank Street Writer, que acompanha o computador MSX da Toshiba, é um pacote de processamento de palavras popular e fácil de usar.

Telecomunicações

Um acréscimo popular de hardware aos computadores MSX é um *modem*. Um modem permite que você se comunique com outros computadores pela linha telefônica. Também permite que você utilize um grande número de serviços de informação (como grandes bancos de dados e grupos de usuários) disponíveis nos computadores de outras pessoas.

Como um modem opera? O procedimento é realmente simples. Com os modens, as linhas telefônicas se transformam em cabos longos entre computadores. Da mesma maneira que seu computador envia caracteres a sua impressora por um cabo, um modem envia caracteres por uma linha telefônica a um outro modem em outro computador. Infelizmente, a maioria dos modens não permite que você se comunique com rapidez. Com freqüência, transmitem 30 caracteres por segundo (300 baud) ou 120 caracteres por segundo (1200 baud).

Será necessário mais do que o modem para transferência de dados com outro computador; é indispensável também um software de comunicação que informe ao modem como enviar a informação. Estas são as tarefas que o software de comunicação pode executar:

- O programa de comunicação liga seu terminal diretamente ao outro computador. A isto denominamos emulação do terminal. Por exemplo, se você se interligar com um computador tipo "mainframe", o software de comunicação enviará todo caractere que você digitar a outro computador e imprimirá todo caractere que o outro computador

colocar em sua tela. Existem diversas empresas que permitem que você acesse informações existente em seu "mainframe", se você tiver um modem e um programa de comunicação.

- O programa de comunicação permite que você envie arquivos de texto entre computadores. Por exemplo, você pode enviar um relatório para alguém que tenha um computador e um modem; seu programa de comunicação envia o arquivo pela linha telefônica. Enviar um arquivo de texto com um programa de comunicação é mais rápido e mais seguro do que redatilografar o arquivo e enviá-lo pelo correio.

- Alguns programas permitem que você envie programas pela linha telefônica. Para que isto seja possível, os dois elementos têm de processar o mesmo programa de comunicação ou ter dois programas que utilizam o mesmo método para transmitir e receber arquivos de programas. Uma vez que os programas são como arquivos que têm caracteres especiais, a maioria dos programas de comunicação, que permite que você envie e receba arquivos do texto, permite que você envie e receba programas.

- Alguns modens discam para você. A discagem automática necessita de um software para fornecer os comandos corretos. Muitos dos programas de comunicação permitem um armazenamento interno de números telefônicos de maneira que o programa faz a discagem. Com isto a utilização do modem fica mais fácil; basta ordenar ao programa de comunicação que disque para um determinado computador.

Quase todos os programas de comunicação podem executar a emulação de um terminal e transferência de arquivos de texto, e grande parte pode executar transferência de programas e discagem automática.

As outras características que você deve levar em conta em um software de comunicação são a facilidade de utilização (a maioria é bem simples) e a compatibilidade com outros programas. Infelizmente, a última é difícil de ser determinada. Você deverá saber qual é o protocolo de comunicação (o método que os computadores utilizam para ter certeza de que os dados foram enviados corretamente) utilizado pelos computadores com que você se comunica, porque seu programa de comunicação deverá utilizar o mesmo protocolo. Você não precisa saber o que é que o protocolo faz, mas apenas seu nome. Dois dos protocolos mais comuns são o XON/XOFF (pronuncia-se ex-on-ex-off), também conhecido como "Control-Q/Control-S", e o MODEM7, um protocolo utilizado quase que exclusivamente por microcomputadores.

Planilhas

Os programas de planilhas levam grande parte do crédito pela popularidade dos micros em pequenos negócios. Antes do lançamento do VisiCalc pela VisiCorp, os computadores, pessoais eram utilizados pelas empresas para processar software de processamento de palavras e de contabilização. O microcomputador tornou-se uma ferramenta muito mais viável para as empresas quando foi desenvolvido um programa que modelava o crescimento da empresa: analisava a projeção de vendas *versus* o registro de despesas. Isto fez o computador executar funções normalmente feitas por meio de gráficos pela contabilidade.

	A	B	C	D	E	F
1		1981	1982	1983	1984	1985
2						estim.
3						
4	Hardware	110201	124933	121640	145283	148910
5	vendas					
6						
7	Software	48270	65391	86922	94280	97640
8	vendas					
9						
10	Renda	0	0	15900	32740	46000
11						
12						
13	Rendim.					
14	total	158471	190324	224462	272303	292550

Figura 2.2 Uma tabela, mostrando números e etiquetas.

Após a introdução do VisiCalc, muitas pessoas encontraram utilização para as tabelas no âmbito doméstico. Há um grande número de livros que mostram como as tabelas podem ajudar no planejamento do imposto de renda e gastos domésticos. Existem muitos programas de tabelas para o MSX, como o Multiplan da Microsoft.

A idéia básica que se encontra em um pacote de planilhas é bem simples: a tela é dividida em celas retangulares, similares às repartições existentes na folha do contador. A Figura 2.2 mostra o exemplo de uma tabela deste tipo. As colunas são marcadas com letras e as linhas são marcadas com números. As celas são denominadas pela coluna e linha em que estão localizadas. Por exemplo, o número "110201" está na cela B4.

Cada cela poderá conter uma destas três informações: números, equações ou texto.

- *Números.* Poderá ser o número de cruzeiros ganhos por um departamento durante o ano, o número de uma peça em um inventário, o número de horas despendidas em um projeto etc. Estes números são fixos; isto é, são como os dados que você utiliza como entrada. Por exemplo, na Figura 2.2, o número na cela B4 representa o número de cruzeiros ganhos em 1981 na venda de hardware.
- *Equações.* As equações relacionam as celas entre si. Por exemplo, a cela B13 na Figura 2.2 contém a equação B4 + B7 + B10. Pela adição do conteúdo destas três celas, tem-se o rendimento total de 1981. Da mesma maneira, as celas na coluna F contêm equações que projetam as vendas para 1985, baseadas no crescimento de vendas do ano anterior. As equações não aparecem na folha, apesar de que você as vê quando movimenta o cursor para a cela em questão. De outro modo, são indicados apenas os resultados das equações.

As equações podem utilizar ainda outras funções matemáticas, como multiplicação e divisão. Por exemplo, uma cela pode conter uma equação que determina qual foi a porcentagem de um determinado departamento para o rendimento de uma empresa.

– *Texto*. O texto é utilizado para identificar partes da tabela, lembrando o que os números representam; um exemplo é a palavra "Renda" na cela A10.

A vantagem nos programas de tabela é que permitem alterar com facilidade os números e equações e em seguida ver o resultado. (Estes cálculos são denominados "se-então".) Por exemplo, você quer saber o que aconteceria se o padrão de vendas, mostrado na Figura 2.2, fosse deslocado de maneira a indicar um aumento da renda. Basta mudar os números da coluna 1984 e verificar como isso afetaria a coluna 1985. Ou poderia alterar as equações utilizadas na coluna 1985 e ver como se alteram os ganhos projetados.

Os programas de tabelas têm numerosas aplicações como ferramentas de uso geral para modelagem com números. Além de fornecerem uma maneira mais fácil de adivinhar o futuro, fornecem também um método sofisticado para rastrear o fluxo de caixa de uma empresa ou uma conta bancária individual. Após observar as diversas maneiras de como o dinheiro entra e sai de uma empresa, um administrador poderá analisar diversas estratégias para maximizar o lucro. Poderá utilizar tabelas para solucionar problemas de engenharia ou para qualquer tarefa que necessite de equações.

Sistema de Gerenciamento de Banco de Dados

Um sistema de gerenciamento de banco de dados (DBMS) é um programa que organiza a informação, extrai informação selecionada, permite o acréscimo de nova informação, e atualiza a informação já existente no computador. Nos negócios, um DBMS o ajuda a controlar o inventário, o livro de contas, nome e endereço de clientes e assim por diante. Em casa, pode auxiliar na organização de contas, diversões e outros interesses pessoais.

Alguns programas de DBMS do MSX são destinados à utilização doméstica e são portanto muito simples. Outros são utilizados em empreendimentos comerciais, e as características potentes que têm poderão tornar estes programas bem complexos. A seguir alguns dos fundamentos dos programas DBMS.

Um banco de dados é uma coletânea de informações armazenadas em seu computador. Todo banco de dados tem uma estrutura, que é o padrão ou formato em que suas informações serão armazenadas pelo DBMS. A estrutura define o tipo de informação que você pode armazenar (como nomes, endereços, código de peças etc.) e se a informação é texto ou um valor numérico. Por exemplo, se você mantém uma base de dados de clientes, a estrutura da base de dados poderá assemelhar-se à Tabela 2.1 (este exemplo é utilizado constantemente nesta seção).

A informação, na estrutura, está contida em campos e registros. Campos são pedaços de informação com finalidades específicas no conjunto de dados. Por exemplo, a entrada de um nome é um campo, um endereço é um campo etc. A estrutura define as características de cada campo.

Tabela 2.1 Estrutura de Banco de Dados de um Cliente

Campo	Tipo	Notas
ID Cliente	Número	Sempre 4 dígitos
Empresa	Texto	Máximo de 20 caracteres
Nome do Contato	Texto	Máximo de 25 caracteres
Endereço	Texto	Máximo de 25 caracteres
Cidade	Texto	Máximo de 15 caracteres
Estado	Texto	Máximo de 2 caracteres
CEP	Número	Sempre 5 dígitos
Telefone	Texto	Sempre na forma ###-##-##
Nome representante vendas	Texto	Máximo de 25 caracteres
Balanço	Número	Indicado no relatório em cruzeiros

Um conjunto de campos inter-relacionados é denominado registro, o qual possui um conjunto de valores para cada campo. Uma analogia seria um catálogo de cartões de uma biblioteca. O catálogo em si é o conjunto de dados, cada cartão um registro e cada pedaço de informação (como o nome do autor ou o título do livro) é um campo.

A Figura 2.3 mostra um registro de um banco de dados conforme definido na Tabela 2.1. Cada campo aparece no registro exatamente uma vez. Você não precisa preencher todos os campos de um registro. A Figura 2.4 mostra uma vista esquemática de um banco de dados. Como você pode ver, cada registro confere com a organização geral do banco de dados.

ID Cliente: 4120
Empresa: CEBI
Nome do Contato: Oscar Burd
Endereço: Av. Corifeu de Azevedo Marques, 1.304
Cidade: São Paulo
Estado: SP
CEP: 05539
Telefone: 815-5669
Nome representante vendas: Luís
Balanço: Cr$ 700.000.000

Figura 2.3 Um registro de um banco de dados de cliente.

O DBMS armazena seu banco de dados em um arquivo no seu disco ou cassete. O DBMS é utilizado para alterar registros ou adicionar novos registros ao banco de dados. Por exemplo, suponhamos que o CEBI mudou o número telefônico. Você utilizaria seu DBMS para encontrar o registro apropriado para em seguida alterar o número telefônico listado nos arquivos.

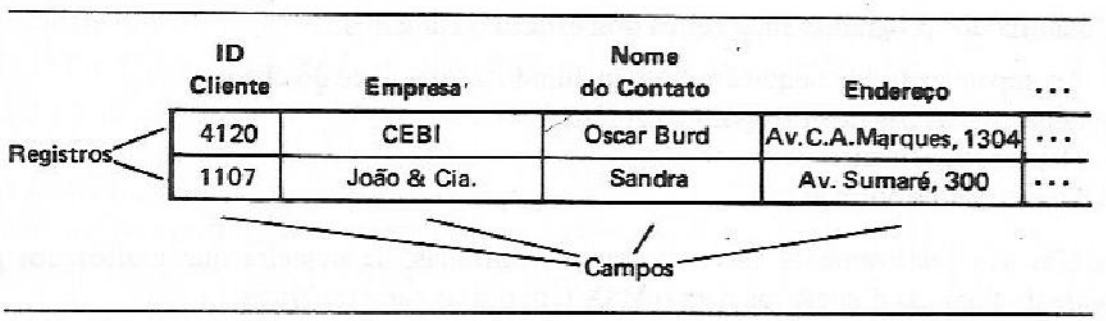


Figura 2.4 Vista esquemática de um banco de dados.

É simples alterar e acrescentar registros, porém a razão principal de você utilizar o DBMS é a pesquisa da informação de seu banco de dados. Um bom DBMS permite que você veja a informação baseado em qualquer critério desejado. Algumas das solicitações que você pode fazer para seu DBMS são:

- Listar todas as empresas que tem CEP iniciando com 902;
- Fazer etiquetas de endereçamento postal na impressora para todos os clientes do CEBI;
- Listar todos os clientes devedores, classificando pelo valor devido e demonstrando o total devedor;
- Listar todos os clientes acrescentados ao banco de dados durante o último mês.

Um DBMS é bem útil para fazer análises imediatas de informação, como o fornecimento de totais para quantidades numéricas. Outra tarefa que faz com perfeição é localizar a informação. Por exemplo, você pode querer olhar o registro de uma empresa, mas tudo o que lembra a respeito dela é que o seu nome inicia com a letra "P" e que está localizada em São Paulo. Seu DBMS poderá encontrar os registros que atendam aos dois critérios.

Você pode imaginar que leve muito tempo para que um DBMS percorra um arquivo de dados muito grande, à procura de informação específica. Para acelerar a procura, quase todo DBMS existente no mercado utiliza campos com chave. Ao definir uma estrutura, você informa ao DBMS qual é o campo que julga utilizar com maior freqüência para fazer procuras (no exemplo, nome da empresa e o local são boas conjecturas). Assim, o DBMS utiliza um campo especial de chaves para encontrar, rapidamente, os registros selecionados.

Agora, você já deve ter imaginado diversas maneiras em que possa utilizar o DBMS. Um DBMS avançado, mas complexo, pode levar horas para ser compreendido, ao passo que um simples pode levar apenas alguns minutos. Se comprar um DBMS, esteja certo de estar adquirindo um que atenda às suas necessidades (e, é claro, seu bolso). Dois sistemas populares de arquivamento são FYI File da FYI, Inc., e Aackobase da Aacksoft International.

Finanças Domésticas

Cuidar de seu talão de cheques particular é bem mais fácil do que tomar conta de um negócio completo. Um programa de finanças domésticas é, portanto, muito mais simples do que o programa empregado em um empreendimento comercial.

A maioria dos programas financeiros domésticos o ajudam a:

- Acompanhar todo cheque emitido, incluindo a finalidade do cheque;
- Separar seus pagamentos por categorias;
- Mapear suas finanças; e
- Prever contas normais.

Estas tarefas são relativamente fáceis de serem realizadas, de maneira que muitos dos pacotes de software de finanças domésticas para o MSX têm outras características.

Se você tiver coragem, poderá escrever seu próprio pacote em MSX-BASIC pois, apesar de ser um programa bem longo, não é difícil de ser feito. De fato, muitos livros de BASIC têm programas que podem ser colocados em seu computador tão bons como muitos dos pacotes que podem ser comprados.

Auto-Aperfeiçoamento

Uma das principais razões porque as pessoas compram computadores domésticos é para melhorar suas vidas. A utilização de um processador de palavras em lugar de uma máquina de escrever (ou lápis e papel) é normalmente um progresso tão grande que satisfaz totalmente suas necessidades. Porém, existem muitos outros programas que permitem um aprendizado sobre você mesmo, uma melhoria em suas habilidades e o estudo de assuntos novos.

Muitas pessoas não sabem escrever a máquina e não querem perder tempo indo à escola para aprender. Os programas que ensinam datilografia, disponíveis para o MSX, são maneiras excelentes de aprender a datilografar com velocidade própria. Muitos destes programas não ensinam apenas o básico sobre o funcionamento do teclado, mas também como aumentar a velocidade. Em alguns, existem jogos que transformam o aprendizado de datilografia numa brincadeira.

Existem ainda muitos cursos acadêmicos disponíveis para todos, desde crianças em idade pré-escolar até adultos. Os educadores discordam entre si da utilização do computador para ensinar às crianças, coisas elementares como aprender a escrever e matemática: enquanto alguns acham excelente, outros o condenam por tornar o aprendizado um exercício de repetição. Independentemente do debate, existem dúzias de pacotes educacionais disponíveis aos computadores MSX.

Existem ainda jogos educacionais que ensinam às crianças e os adultos tarefas não-tradicionais como cooperação e taquigrafia. Apesar de estes pacotes serem com freqüência listados como jogos, são em verdade programas que auxiliam no aprendizado de tarefas que tornam a convivência com outras pessoas mais fácil. Por exemplo, um programa muito peculiar é "In Search of the Most Amazing Thing", de Spinnaker Software. Este jogo ajuda crianças e jovens adolescentes a fazer anotações, ler um mapa, barganhar e conseguir ajuda na tomada de decisões. Este jogo é muito mais complexo do que os outros, apesar de cada parte ser simples para a maioria das crianças de dez anos.

Aprendendo Mais sobre Computadores

Uma das razões mais comuns que levam as pessoas a comprar um computador MSX é que "querem aprender mais sobre computadores". Existe um sentimento generalizado na socie-

dade brasileira que diz que se você não "conhece sobre computadores", está destinado a um emprego pobre e uma desgraça social. Este receio é completamente infundado.

O material constante nos Capítulos 1 ao 4 deste livro, será provavelmente mais do que suficiente para uma compreensão básica a respeito de computadores – pelo menos suficiente para que não tenha medo deles se os encontrar em seu trabalho. Certamente não precisará saber como programar um computador, ou como opera a parte eletrônica dentro do computador, para que você possa sobreviver em uma sociedade moderna.

Existe, sem dúvida, muito mais a aprender sobre seu computador MSX. Você poderá saber mais a respeito de software, como programar em MSX-BASIC, ou a respeito do hardware dentro de seu computador MSX. Na maioria dos casos, qualquer conceito aprendido sobre seu computador MSX será largamente aplicável em outros computadores, independentemente se são domésticos ou de grande porte. Assim você pode utilizar o seu computador MSX como um degrau para entender mais sobre outros computadores. Novamente, é importante enfatizar aqui que a maior parte deste entendimento tem utilidade apenas para aquelas pessoas que praticamente só trabalham com computadores.

O que a maioria das pessoas faz com os computadores MSX é aprender MSX-BASIC. O aprendizado de uma linguagem de computador do tipo MSX-BASIC poderá beneficiá-lo em diversas maneiras:

- Você entenderá melhor como seu computador executa as tarefas;
- Você começará a entender como é complicado programar um pacote grande como um processador de palavras; e
- Você entenderá melhor o relacionamento entre os diferentes componentes de seu computador (como o teclado e a UCP).

Para os computadores MSX existem ainda disponíveis outras linguagens como C e Logo, que poderão interessá-lo. A C é uma linguagem avançada de programação; o pacote BDS-C está disponível pela Lifeboat Associates. A linguagem Logo é muito popular no ensino de programação para crianças; uma versão excelente da Logo está disponível pela The Lisp Company.

É importante lembrar que para utilizar o computador você nada precisa aprender sobre linguagens de computador. Grande parte das pessoas da área de computação defende o aprendizado de uma linguagem de computador, apesar de muitos outros acharem que é contraproducente para a maioria dos usuários primários. Não se preocupe se você acha que o MSX-BASIC o intimida; é assim para muitas pessoas.

Se quiser aprender mais sobre hardware, existem muitos livros no mercado que explicam a parte interna dos computadores. Entretanto, é extremamente difícil descrever o hardware de um computador sem se tornar muito técnico, e será necessário bastante conhecimento de eletrônica para entender como os diversos integrados de seu MSX interagem.

CAPÍTULO

3

EXPANDINDO O SEU COMPUTADOR MSX

No capítulo anterior foram discutidos os diversos tipos de programas que você pode comprar para o seu MSX; este discorrerá sobre o hardware que você pode adicionar ao seu computador MSX. Em alguns computadores, parte dos produtos mencionados aqui são padrões, enquanto outros estão disponíveis apenas como opções.

Assim como o conector padronizado, para colocação de um cartucho, facilita a compra dos programas para o MSX, facilita também a compra do hardware para o MSX. Os fabricantes de periféricos de hardware (hardware que você acrescenta ao seu computador) não precisam se preocupar com o tipo de plugue que existe no computador MSX; basta fazer um cartucho como interface para o hardware.

Uma vez que os computadores MSX são relativamente baratos, você pode indagar por que deve comprar outro hardware, em alguns casos mais caro do que o computador. Alguns periféricos são muito úteis em casa ou no escritório e permitem que seu computador MSX execute mais funções (como processamento de palavras). Você pode querer comprar outros periféricos apenas porque parecem interessantes. Como no caso dos programas, existe mais hardware disponível do que o necessário para um único usuário.

Este capítulo discute a maior parte dos periféricos disponíveis para os computadores MSX. À medida que o MSX se tornar mais popular, serão desenvolvidos novos periféricos. Uma vez que o preço dos periféricos muda com a sua disponibilidade e popularidade, neste capítulo o preço é mencionado apenas ocasionalmente. Para preços atuais de periféricos, verifique os anúncios nas revistas sobre computadores domésticos.

Expansor de Cartucho

Se o seu computador MSX tiver lugar para apenas um cartucho, a primeira aquisição eventualmente feita será um expansor de cartucho. Este dispositivo ocupa o espaço do cartucho, mas lhe dá dois ou mais espaços novos. Deste modo é possível utilizar mais do que um cartucho por vez.

Alguns computadores MSX incluem um conector que não é um padrão do MSX. Os fabricantes destes computadores com freqüência vendem expansores que podem ser ligados a estes conectores, fornecendo deste modo espaço para mais cartuchos sem perder nenhum. Para utilizar este conector, entretanto, você deve comprar o expansor do mesmo fornecedor que fez seu computador.

Antes de comprar periféricos de hardware que requerem um lugar de encaixe, verifique se existem locais suficientes para estes periféricos poderem ser ligados. Por exemplo, se seu computador tiver espaço apenas para um cartucho e você quiser utilizar um programa de processamento de palavras que está em cartucho bem como uma impressora que utiliza o mesmo ponto de conexão, não será possível utilizar os dois ao mesmo tempo a menos que se adquira um expansor. Uma alternativa seria a aquisição de uma impressora que pode ser ligada ao conector de impressão existente na maioria (mas não todos) dos computadores MSX.

RAM adicional

Se o seu computador MSX tiver 16K ou 32K de RAM, poderá haver necessidade de expandir a memória. O MSX-BASIC poderá utilizar até 32K de RAM, enquanto o MSX-DOS exige 64K. Caso você seja um programador experiente necessitará de mais memória para escrever programas em BASIC ou outras linguagens que utilizam mais do que 32K de RAM.

Existem cartuchos disponíveis pela maioria dos fabricantes que permitem a expansão da memória de seu computador MSX. Antes de comprar um cartucho deste tipo, você deve verificar com o distribuidor, certificando-se de que a memória adicional irá operar com seu computador e que seu software pode utilizar a memória adicional. Por exemplo, se você estiver utilizando apenas o BASIC, a expansão RAM a um computador de 32K não fornecerá possibilidades adicionais de programação. Assim, esteja certo de poder utilizar a memória adicional antes de comprá-la.

Controles para Jogos

A maioria dos jogos disponíveis para os computadores MSX pode ser controlado por manipuladores, muito semelhantes àqueles existentes nos aparelhos de vídeo jogos. Estes manipuladores permitem que se tome rapidamente as direções para cima, para baixo, esquerda e direita; possui também um gatilho utilizado quando se deseja, por exemplo, dar tiros em uma nave espacial inimiga.

Todos os computadores MSX têm pelo menos um conector para a ligação de um controlador, podendo ter mais um para outro manipulador. Qualquer computador MSX pode utilizar manipuladores fabricados para outros computadores MSX, bem como aqueles fabricados para os computadores domésticos tipo Atari ou para os vídeo jogos. A maioria dos usuários compra manipuladores baratos, existindo porém os modelos mais caros para os jogadores "experientes".

Armazenamento em Cassete

Todos os computadores MSX têm um plugue onde pode ser conectado um gravador de fita cassete. Podemos utilizar os cassetes para armazenar e recuperar programas escritos em MSX-BASIC; pode-se também armazenar dados em cassete.

Alguns gravadores cassete, caseiros, podem ser utilizados com seu computador MSX. Caso você já tenha um gravador cassete é possível utilizá-lo, em vez de comprar outro para ser utilizado no computador. As principais exigências para estes gravadores é que tenham três soquetes: microfone, fone de ouvido (ou alto-falante) e "remoto". Verifique se o tamanho dos plugues fornecidos com o seu computador MSX é compatível com os soquetes de seu gravador. Se for, provavelmente será possível utilizá-lo.

Se você não tiver um gravador cassete compatível, poderá comprar um relativamente barato. São vendidos em lojas de computação e de som. (Não esqueça de verificar se ele é adequado ao seu computador.) (Os gravadores para armazenamento de dados normalmente custam menos do que 15 ORTN.)

O cassete que você utiliza para armazenar informação, é o mesmo utilizado para gravar música. Alguns fabricantes vendem cassetes "especiais", que, normalmente, são idênticos aos de música de alta qualidade, apesar de serem, geralmente, mais caros.

Acionadores de Disco

Caso você utilize bastante o seu computador MSX – especialmente se você o está utilizando em seu trabalho – pode ser que precise comprar um acionador de disco para ele. Um acionador de disco armazena dados e programas com muito mais velocidade e facilidade do que um gravador cassete.

A principal desvantagem de se comprar um acionador de disco, é seu preço. É comum o acionador de disco custar o mesmo que o computador MSX. Para muitos usuários, este custo é proibitivo. Entretanto, certos programas, para serem utilizados, exigem que se tenha um acionador de disco. (Estes programas por sua vez são caros, tornando a compra do acionador ainda mais difícil.) Cada modelo de acionador tem uma quantidade diferente de dados que podem ser armazenados, de modo que é importante que se faça uma boa pesquisa antes de fazer uma compra.

Quando você salva os dados em um disco ou em um cassete, a informação é armazenada em arquivos. O armazenamento de informação em arquivos é parecido com organizar dados em pastas suspensas em um gabinete de aço. Um arquivo é uma coleção de informações identificadas por um único nome, fornecido por você. O armazenamento de informações em arquivos é fundamental para utilização do computador; sem os arquivos o seu trabalho se perderia com o desligamento do computador.

A informação em um arquivo pode ser: texto (como um memorando), dados (uma lista de endereçamento postal), ou um programa (como processamento de palavras). O arquivo pode ter qualquer extensão, sendo limitado apenas pelo espaço disponível no disco ou cassete em que será armazenado. Quando você quiser que um programa opere com um arquivo (ou para ler informações do arquivo ou acrescentar informações ao arquivo), basta utilizar o comando de acesso ao arquivo e informar ao programa o seu nome.

Como foi mencionado anteriormente, é muito mais conveniente armazenar arquivos em discos do que em cassetes. O disco é uma peça redonda, de material rígido, coberto por um material magnético; um disquete (ou floppy) é a versão flexível do disco. Os dois tamanhos mais comuns para os disquetes são 5 1/4" e 3 1/2". O que acontece quando o MSX recupera arquivos

armazenados em disco? O disco gira em alta velocidade enquanto a cabeça do disco se movimenta para dentro e para fora (a cabeça do disco é semelhante às cabeças de gravador de fita comum). O movimento é idêntico à seleção de uma música em um disco. Quando você pede para ver um determinado arquivo, a cabeça se dirige primeiro ao diretório, uma região especial do disco que tem informações sobre cada arquivo. Encontra o local do arquivo desejado e em seguida se desloca para o local em que o arquivo se encontra.

Você pode estar se perguntando como a cabeça do disco encontra o arquivo. Existem entradas no diretório formadas por dois números para cada arquivo: a trilha e o setor. O MSX-DOS utiliza esta informação para determinar o local de cada arquivo no disco. O número de bytes por setor é constante em um determinado disco, mas o número de bytes por setor poderá variar para os diferentes acionadores que podem ser comprados para o computador MSX. Assim pode variar também o número de setores por trilha. Felizmente, o MSX-DOS mantém este controle para você.

A Figura 3.1 ilustra as trilhas e os setores em um disquete. O número da trilha é basicamente a medida da distância do arquivo à borda do disco. As trilhas são círculos concêntricos no disco e como podem comportar uma grande quantidade de informação, são divididas em setores. Dando

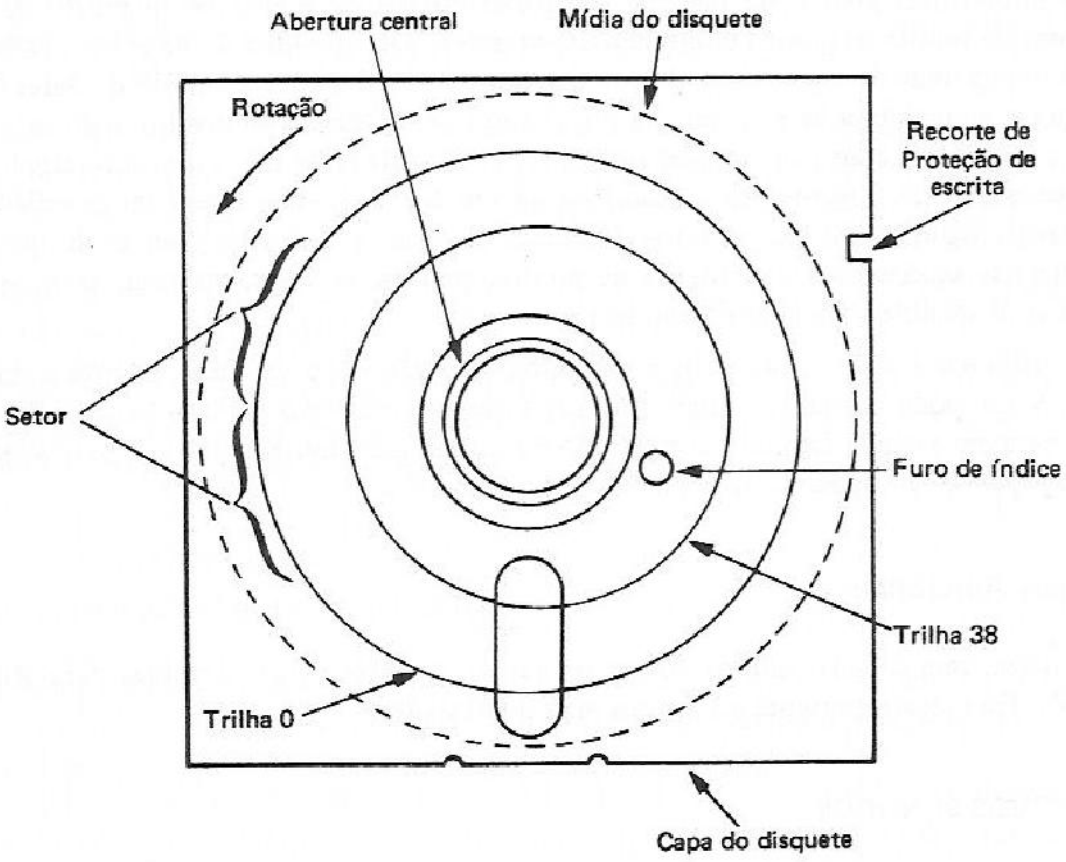


Figura 3.1 Trilhas e setores em um disco.

continuidade à analogia com o disco de som, as trilhas são como ranhuras, e os setores como partes de uma ranhura. Os setores são medidos a partir de um ponto fixo, no sentido anti-horário, dando volta no disco do começo ao fim.

Assim, o diretório informa à cabeça do disco a que distância do centro deve procurar e quanto tempo deve aguardar durante uma rotação antes de iniciar a leitura de informação. Encontrar uma determinada trilha é como encontrar uma música em um disco fonográfico. A seleção do setor é como procurar um compasso dentro da música.

Impressoras

Após ver a informação na tela de seu computador MSX, provavelmente vai querer imprimi-la em papel para referência futura ou para mostrá-la a outras pessoas. Para imprimir a informação no papel é necessário utilizar uma impressora.

Quase todo usuário de computador comercial tem uma impressora. As impressoras variam muito nas características e no preço. Uma impressora comum é ligada ao seu computador MSX por um conector de impressão ou por meio de um cartucho. Podemos instruir a impressora para imprimir caracteres através do MSX-BASIC ou do MSX-DOS. Algumas impressoras ainda podem imprimir gráficos (figuras) com o processamento de um programa especial.

As impressoras podem ser de duas categorias: matriz de pontos ou qualidade de letra. As impressoras de matriz de pontos imprimem um caractere pela combinação de pontos para formar a letra. As impressoras do tipo qualidade de letra, por outro lado, imprimem os caracteres da mesma forma que uma máquina de escrever. As impressoras de matriz de pontos normalmente são mais baratas, e imprimem com mais rapidez do que as de qualidade de letra, mas o resultado destas é de melhor qualidade. Algumas impressoras do tipo matriz de pontos dizem ter qualidade de correspondência, significando que as letras formadas parecem exatamente com as da qualidade de letra. Algumas impressoras tipo matriz de pontos, podem, às vezes, imprimir gráficos. Mas, as impressoras de qualidade de letra não podem.

As utilizações mais comuns para as impressoras são os programas de processamento de palavras. Você pode digitar e editar uma carta ou um relatório utilizando o processador de palavras para em seguida imprimir o resultado em sua impressora. Sempre que necessitar de uma cópia em papel de seu trabalho, você precisa de uma impressora.

Periféricos Adicionais

Existem, sem dúvida, muitos outros periféricos que você pode comprar para seu computador MSX. Esta seção apresenta alguns dos mais interessantes.

DISPOSITIVOS SONOROS

Uma das utilizações mais populares dos computadores domésticos é o aprendizado e criação de música. As possibilidades sonoras dos computadores MSX são melhores do que as de muitos computadores domésticos, e é sempre possível fazer adições a estas capacidades.

Quase todo mundo já escutou música tocada por um sintetizador. Você pode comprar cartuchos para o seu computador MSX que o transformam em um sintetizador; na verdade, no computador CX5M da Yamaha, está incluído, dentro da unidade principal, um sintetizador e uma interface para instrumento musical. O sintetizador pode ser controlado via MSX-BASIC ou via teclado.

Uma vez que é possível criar música e efeitos sonoros de alta qualidade com o seu computador MSX, poderá haver interesse em amplificar o resultado em seu sistema de alta-fidelidade. Vários computadores MSX têm plugues que permitem a ligação do áudio em amplificadores externos.

GRÁFICOS

Na indústria dos computadores o campo de maior crescimento é o dos gráficos. Os preços dos periféricos gráficos estão caindo rapidamente, de modo que muitos periféricos que eram comprados apenas pelas grandes empresas, podem agora ser adquiridos pelo usuário doméstico.

Um periférico que se está tornando bem comum é a caneta ótica. Esta ferramenta em forma de lápis permite "desenhar" quando é encostada na tela. Você pode utilizar a caneta ótica para fazer desenhos, para dar respostas rápidas a questões existentes na tela, bem como para muitas outras tarefas. Podemos utilizar da mesma forma que uma caneta ótica, uma prancha digitalizadora. Esta prancha é colocada sobre a mesa. Com o dedo você faz traços sobre a tabuleta, que se repetem na tela.

Apesar de ser possível desenhar com uma caneta ótica ou com uma prancha digitalizadora no seu computador MSX, pode haver interesse em ler fotografias ou outro tipo de informação pictórica, em vez de desenhar. Para conseguir isto, será necessário um digitalizador de imagem, que nada mais é do que uma câmara eletrônica que pode ser ligada ao computador. A imagem será "digitalizada" para uma forma que o computador possa ler, armazenar e imprimir. Será necessário também o programa que permita ao MSX-BASIC ler a imagem digitalizada.

Alguns periféricos permitem que você misture as imagens de seu televisor com aquelas que você armazenou em seu computador MSX. Estes dispositivos podem ainda armazenar as imagens da televisão e permitir que sejam modificadas, pela alteração da cor ou eliminação de parte da figura.

CONTROLE DAS APLICAÇÕES DOMÉSTICAS

Os computadores MSX podem controlar diversos dispositivos eletrônicos por intermédio de uma central de controle doméstico. Aplicações simples da central de controle doméstico consistem em ligar e desligar a luz em instantes determinados ou ligar o bule de café toda manhã. Pode-se adquirir harware mais sofisticado, que permita ao computador MSX controlar um sistema de segurança, impedindo assaltos ou alertando a polícia e o corpo de bombeiros em caso de emergência pessoal.

O computador MSX poderá ainda ser utilizado para controlar o sistema de lazer doméstico. Por exemplo, você pode comprar o hardware que controla o sistema de disco laser, de modo que

seja possível mudar para qualquer quadro do disco laser, baseado em informações armazenadas no computador. Isto pode ser útil para enciclopédias armazenadas em disco laser ou para jogos que armazenam imagens em disco laser. À medida que a tecnologia do disco laser japonês progredir, haverá maior número de aplicações que vão ligar os computadores MSX aos discos laser.

ROBÔS

Apesar das previsões dos últimos cinqüenta anos, robôs que fazem a limpeza da casa e preparam comida não são comuns ainda. Os robôs que você pode comprar têm utilização muito limitada e geralmente não são muito rápidos. Entretanto, o estudo dos robôs aumentou na última década, existindo muitos robôs pequenos à disposição caso esteja interessado em conhecê-los.

Alguns robôs podem ser programados pelo seu computador MSX. Por exemplo, um robô barato poderá ter um lugar para inserção de um cartucho e um módulo de memória. Você encaixa o módulo de memória em seu computador MSX e armazena ali as instruções do robô. Ao plugar o módulo no robô, este passa a obedecer as instruções.

CAPÍTULO

4

INSTALANDO O SEU SISTEMA

Os computadores não saem de suas caixas prontos para serem utilizados. É necessário realizar algumas tarefas para fazê-los operar. Felizmente, porém a maioria dos computadores MSX é fácil de ser instalada necessitando apenas alguns minutos e nenhuma ferramenta.

Como cada tipo de computador MSX é instalado de maneira diferente, a informação deste capítulo é geral e não específica para um determinado computador. A seção que trata de cartuchos e cuidados gerais se aplica a todos os computadores MSX.

Como Fazer o seu Computador MSX Operar

Para instalar o seu computador MSX existem basicamente quatro passos:

1. Retire o computador e todos os componentes da(s) caixa(s).
2. Leia as instruções que acompanham o seu computador.
3. Conecte o computador ao seu televisor ou monitor.
4. Conecte o computador à rede e ligue-o.

Todos estes passos são bem simples, mas a maioria das pessoas pula o mais importante: a leitura das instruções. A importância deste passo não pode ser desprezada. Você deve ler estas instruções mesmo que as ache muito complicadas ou maçantes. Após ter retirado tudo da caixa (ou das caixas), guarde a(s) caixa(s) e o material de embalagem. O material de embalagem terá importância caso você mude o computador, e também será necessário caso você o devolva.

Guarde todo o material impresso que veio junto com o seu computador em um mesmo local (uma pasta pode ser conveniente). Novamente, você deve ler a documentação — ou pelo menos aquelas partes que discutem a instalação e manutenção de seu computador; pode deixar as partes do MSX-BASIC para depois.

Dependendo da marca de seu computador MSX, poderá encontrar um hardware adicional, além da unidade principal. Alguns computadores MSX vêm com periféricos mencionados no Capítulo 3. A maneira de ligar qualquer hardware adicional é descrita no manual do usuário.

Se você estiver utilizando o televisor como tela de seu computador, o seu computador MSX poderá vir acompanhado com um modulador RF. É um nome estranho para a caixa que liga o computador ao aparelho de televisão; em certas ocasiões é chamado "caixa de comutação de tevê". Você liga os terminais do modulador RF ao conector de antena, atrás do aparelho de televisão, como indicado na Figura 4.1, e liga o computador ao outro lado do modulador RF. Uma vez que os moduladores RF variam de um fabricante para outro, é importante ler as instruções para fazer a instalação. Se estiver utilizando um monitor com seu computador, o cabo para este monitor deve acompanhar o computador.

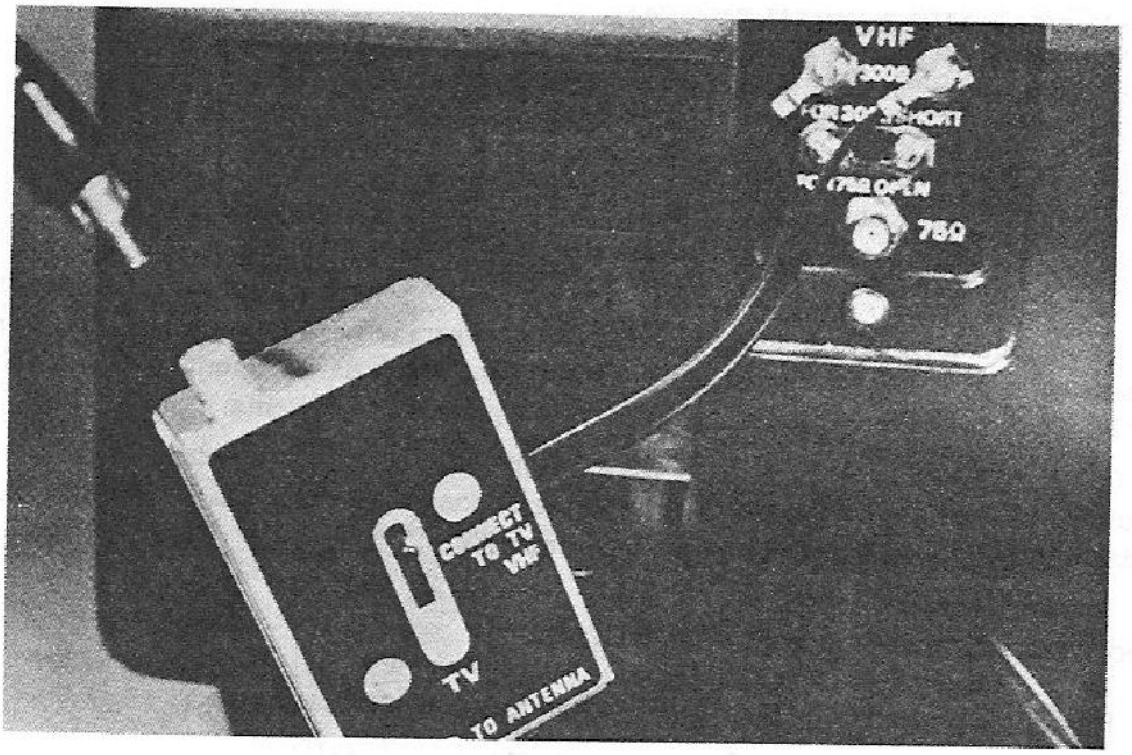


Figura 4.1 Modulador RF e ligações.

Uma vez que o seu computador esteja ligado ao monitor ou televisor (e tenha sido conectado qualquer outro hardware descrito em seu manual), você está quase pronto para conectá-lo e ligá-lo. Antes de fazê-lo leia a página que está no início do manual e se intitula "Informação FCC" (ou algo parecido). Não se assuste com os avisos, mas esteja ciente de que a utilização de seu computador poderá degradar a recepção do rádio ou do aparelho de televisão de seu vizinho.

Siga as instruções do fabricante para a conexão do seu computador. Pode ser que você tenha de utilizar uma caixa de conversão, que tenha um plugue normal de parede de um lado e um plugue diferente no outro; em alguns computadores isto é interno e tem apenas os plugues normais. Se você estiver utilizando um aparelho de televisão, siga as instruções do manual quanto ao canal que deve ser utilizado.

Finalmente, ligue o seu televisor (pode ser que queira abaixar o som); em seguida ligue o computador. Se você seguiu as instruções corretamente, a sua tela mostrará uma mensagem com os direitos autorais do MSX; em seguida verá a tela inicial do MSX-BASIC, similar à Figura 4.2 (não se preocupe se a informação na tela for ligeiramente diferente).

Não entre em pânico, caso não apareça a tela inicial do MSX-BASIC. Seu computador pode estar programado para iniciar um programa diferente do MSX-BASIC. Caso não veja nada na tela pode ser que a conexão entre o computador e o monitor (ou televisor) não está correta.

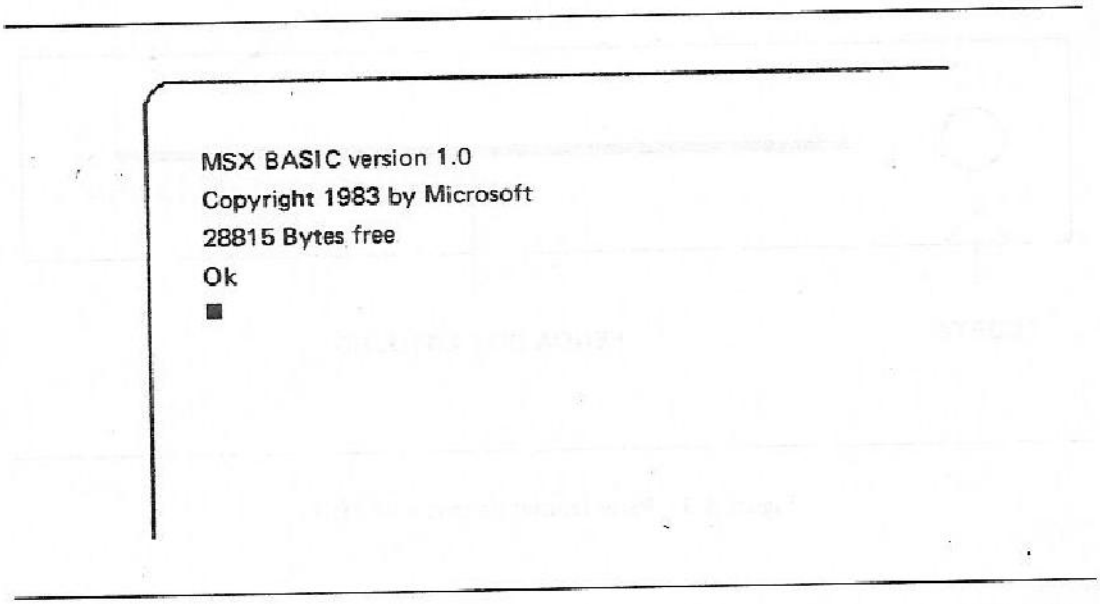


Figura 4.2 Tela inicial do MSX-BASIC.

Utilizando Cartuchos

A utilização de cartuchos na MSX é bastante simples, existindo porém algumas regras de como instalá-los. Uma das regras mais importantes a serem seguidas é a de não colocar ou retirar um cartucho enquanto o computador estiver ligado. Se tal acontecer, quando você inserir ou remover o cartucho, isso poderá danificar o programa do cartucho; é até possível danificar o seu computador. Portanto, esteja certo de que o computador está desligado antes de mexer com o cartucho.

Uma vez que o cartucho do MSX é retangular, existem oito modos de colocá-lo, mas apenas um está correto. A Figura 4.3 mostra a vista posterior de um cartucho. A maneira correta de inserir o cartucho é com a etiqueta para cima e a fenda do cartucho deve encaixar no conector.

Fazendo a Manutenção de Seu Sistema

O seu computador MSX não precisa de manutenção, no sentido daquela que um carro precisa. Existem, entretanto, algumas diretrizes bem simples que devem ser seguidas para manter seu sistema livre de erros:

- Trate a unidade principal e os periféricos com cuidado. Não coma, nem beba, enquanto estiver utilizando o computador, uma vez que migalhas e derrames acidentais danificam o teclado e a eletrônica interna;

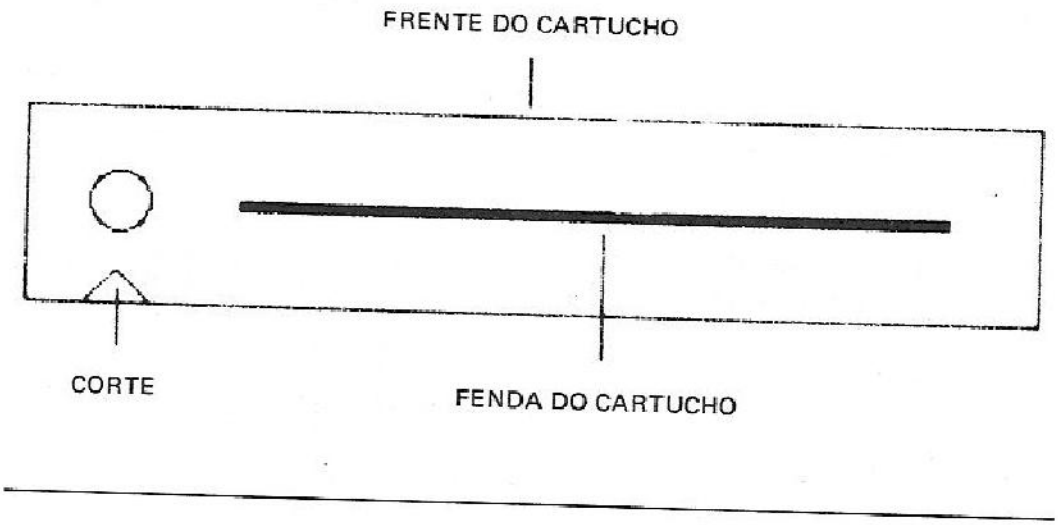


Figura 4.3 Parte frontal do cartucho MSX.

- Mantenha os animais longe do computador. Saliva e pêlo são tão prejudiciais como bebida e comida;
- Cubra o teclado quando não estiver em uso. Tenha cuidado especial caso haja um excesso de pólem ou poeira no ar. Utilize uma cobertura de tecido ou plástico antiestático; e
- Não lave seu computador. Você pode limpá-lo com um tecido úmido, tomando cuidado para que não entre água nele. Não utilize produtos químicos na limpeza (é pouco provável que seu computador fique tão sujo a ponto de precisar de química).

Sem dúvida, siga qualquer outro cuidado listado nos manuais que acompanham o computador.

PARTE

2

APRENDENDO MSX-BASIC

CAPÍTULO

5

INTRODUÇÃO AO MSX-BASIC

É uma infelicidade a existência de tantas pessoas que acreditam ser necessário saber programação para utilizar um computador. Isto pode ter sido verdade há vinte anos, mas hoje, com certeza, não é assim. Programas comerciais disponíveis podem executar quase qualquer função desejada. Por exemplo, você não precisa saber nada sobre programação para poder utilizar o programa de processamento de palavras.

Você pode achar que pelo fato de o seu computador MSX ter vindo acompanhado pela linguagem de programação MSX-BASIC tenha de aprender a utilizá-lo. Fique tranqüilo, não há nenhuma obrigação para aprender a programar em MSX-BASIC. Por outro lado, para entender melhor os computadores, pode ser que você queira aprender a programar.

Aprender a programar é como aprender a consertar seu carro: não é necessário, mas muitas pessoas gostam de "mexer". Com freqüência você adquire um respeito maior pelo seu carro quando aprende a consertá-lo. Enquanto o aprendizado do MSX-BASIC não amplia significativamente seu entendimento de computadores, você pode apreciar a visão obtida no processo. A diferença que existe entre consertar um carro e fazer uma programação é de que com o carro você pode economizar milhares de cruzeiros, ao passo que dificilmente fará economia escrevendo um programa. Se você tiver conhecimento sobre mecânica de automóvel, poderá discutir misturas de ar-combustível e razões de compressão durante um coquetel; quando conhecer programação de computadores poderá discutir rotinas recursivas e variáveis com ponteiros indiretos.

Existem dúzias de linguagens de programação que você poderia aprender, mas existem diversas razões para que o MSX-BASIC seja um bom ponto de partida:

- *Conveniência*. Uma vez que o MSX-BASIC vem incluído em seu computador, você não precisa comprar nada. Basta ligar o seu computador MSX e processar o MSX-BASIC.
- *Popularidade*. Quase todo computador pessoal fabricado nos últimos dez anos processa alguma versão de BASIC. Já que existem mais pessoas que sabem programar em BASIC do que qualquer outra linguagem, existem centenas de livros que ensinam programação BASIC.

– *Fácil de aprender.* BASIC é uma das linguagens de computador mais simples, o que a torna uma das mais fáceis de ser aprendida. Apesar de existir mais de 100 comandos e funções no MSX-BASIC, os conceitos que regem seu uso são fáceis de entender.

Muitos expertos em computação descartam BASIC porque os programas que são escritos nela não são tão elegantes como aqueles escritos em outras linguagens. Alegam que se você aprende BASIC como sua primeira linguagem de computador não conseguirá apreciar a beleza de algumas das outras linguagens disponíveis. De qualquer modo, permanece uma das linguagens de computação mais popular.

O que é Programação

A finalidade de escrever um programa, em qualquer linguagem, é a de fornecer ao computador um conjunto de instruções a serem executadas. Por exemplo, se você quiser que o computador adicione uma lista de números e encontre a média, deve escrever um programa que especifique cada passo necessário para atingir a meta desejada.

Apesar de os computadores não "entenderem" as instruções como o faz uma pessoa, o resultado em se fornecer instruções ao computador será o mesmo que dar informações para uma pessoa dócil: ou o computador executa as instruções, ou lhe informará que não está entendendo o que foi solicitado. Evidente é que, ao "falar" com um computador, ou vice-versa, não se escuta som algum: as instruções são digitadas e o computador responde através da tela.

DEFINIÇÃO DE UM PROGRAMA

Um programa, portanto, é simplesmente um conjunto de instruções escritas em linguagem compreensível pelo computador. A este conjunto de instruções pode-se dar um nome que o descreve. Por exemplo, se você pedir a uma criança para tirar o lixo, poderá dar-lhe as seguintes instruções: "Pegue o lixo que está em baixo da pia da cozinha e coloque-o na lata de lixo. Se for segunda-feira à tarde, você coloca a lata na calçada; caso contrário, basta verificar se a tampa está bem fechada." Da próxima vez bastará que lhe diga: "Retire o lixo", e a criança saberá, com esta descrição resumida, executar o conjunto de instruções que você descreveu antes.

Programar um computador é como ensinar uma criança. O programa que segue (escrito em português) ensina o computador como tirar o lixo:

```
Tirar o lixo significa:
    Primeiro, tirar o lixo que está embaixo
        da pia da cozinha, em seguida
        colocá-lo na lata de lixo.
    Segundo, verificar se hoje é segunda-feira.
        Se for segunda-feira, coloque a lata na
        calçada, caso contrário verifique se
        a tampa está bem fechada. Se a tampa
        não estiver bem fechada, movimente-a
        de modo que fique bem fechada.
```

Este programa mostra muitas das características encontradas em programas típicos de um computador:

- *Organização.* Os programas sempre informam ao computador em que ordem ele deve executar as tarefas. Um computador começa do início do programa lendo-o até o fim, executa as instruções à medida que as lê.
- *Comandos diretos.* São comandos incondicionais que o computador sempre deve fazer (como "tirar o lixo que está embaixo da pia da cozinha").
- *Informações variáveis.* Neste programa, a informação variável é o tempo (segunda-feira) e a condição da tampa. Outro exemplo seria de que o computador deve colocar a lata de lixo na calçada "a não ser que estivesse vazia"; neste caso, o computador verificaria também o peso da lata.
- *Ação condicional.* O computador toma uma decisão e executa uma determinada ação se uma dada condição for verdadeira, mas executa uma ação diferente se a condição for falsa. Neste caso, a condição é "hoje é segunda-feira"; se for, o computador retira a lata de lixo, se não for, verifica a tampa. Os programas de MSX-BASIC com freqüência executam ações condicionais.

Como você pode ver, dizer ao computador como executar uma tarefa é como dizer a uma pessoa como executá-la. Ao escrever um programa, é da maior importância ser explícito sobre a ordem dos comandos e as ações condicionais. (Por exemplo, você não descreveria para alguém como trocar os pneus dando as etapas na ordem errada.) É claro que, se não precisasse lembrar de mais nada, programar seria uma barbada.

COMUNICAÇÃO COM O SEU COMPUTADOR MSX

Em um determinado aspecto, escrever um programa para um computador é diferente de explicar um conjunto de ações para uma pessoa: é necessário aprender uma nova linguagem para se comunicar. Apesar do aprendizado não ser muito difícil, dar instruções naquela linguagem poderá apresentar dificuldades até ter-se acostumado com ela.

Linguagem de programação é a linguagem utilizada quando você informa ao computador o que fazer. Não é possível fazer a programação em inglês já que os computadores não entendem suficientemente o inglês (apesar de você, certamente, ficar surpreso com a quantidade de inglês que eles entendem). Já que o seu computador MSX entende o MSX-BASIC é esta a linguagem que você deve aprender.

Uma linguagem de programação é a linguagem que você utiliza para falar com o computador, não sendo aquela utilizada em conversação. A diferença é muito importante. O seu computador MSX não falará com você em MSX-BASIC. Normalmente responderá em inglês, mas não sempre.

Um dos aspectos mais frustrantes da programação de computador é a de ser quase sempre uma comunicação unidirecional: o que você realiza em uma hora de teclagem, poderá resultar em uma resposta de apenas cinco palavras do computador.

Normalmente, o computador não responde nada a não ser que você peça. É como dizer a uma criança como retirar o lixo; normalmente a única resposta dada é um ocasional "uh-huh". Se você diz ao computador para tirar o lixo, ele somente dirá alguma coisa se não entender a instrução. Estas "falhas" nas instruções indicarão o aparecimento de uma *mensagem de erro* na tela. No MSX-BASIC, estas mensagens são indicadas em inglês. As mensagens de erro ajudarão na determinação exata do que foi feito de errado.

Não pense que você não pode conversar com o computador; pode, desde que lhe dê as instruções corretas. Por exemplo, as instruções poderão informá-lo como deve fazer uma pergunta a você e como deve responder, se você der uma determinada resposta. Sem dúvida, estas instruções são fornecidas por meio de uma linguagem de programação.

Para ser um bom programador, você deve entender a linguagem que estiver programando. Toda linguagem de programação tem um vocabulário bem definido, bem como um conjunto de regras gramaticais — como as linguagens humanas. O vocabulário é como um vocabulário humano limitado, com verbos, substantivos e adjetivos. Quando você comete um erro ao informar o computador o que fazer, isto normalmente se deve à utilização de uma palavra que não faz parte do vocabulário estabelecido ou porque foi utilizada uma construção gramatical inválida.

O vocabulário do MSX-BASIC é bem extenso e relativamente fácil de ser aprendido porque os verbos são muito parecidos com os do inglês. Por outro lado, existem poucos substantivos e adjetivos para serem aprendidos. A gramática do MSX-BASIC também é bem direta.

Pense em Escrever seus Próprios Programas

Agora que você conhece um pouco mais sobre programação, deve estar pensando porque existem aquelas pessoas que passam por dificuldades para escrever um programa. Algumas pessoas acham bem atraente o salário recebido por programadores profissionais. Outros, sentem prazer em programar; consideram isto uma extensão de seus cérebros.

Muitos se iludem pensando que podem escrever todos os programas com a linguagem de programação que acompanha o computador MSX. Parece bem atraente, principalmente quando você descobre que alguns dos programas comercialmente disponíveis são bem caros. A triste verdade é que escrever um programa complicado poderá levar semanas ou meses, mesmo para o melhor programador. Freqüentemente é mais econômico comprar um programa do que escrevê-lo.

Isto não deve desencorajá-lo a aprender a programar. Como foi mencionado no Capítulo 2, existem muitas razões boas para aprender o MSX-BASIC. Entretanto, não comece a aprender a linguagem pensando que vai utilizá-la para criar todos os programas que for precisar.

O MSX-BASIC é muito bom para escrever programas de jogos. Já que existem muitos livros que contêm listagens em MSX-BASIC de programas para jogos, você pode economizar muito dinheiro digitando os programas em vez de comprar o cartucho. À medida que você digita os programas, poderá ter uma noção a respeito da linha de raciocínio do programador ao fazer o programa. Após ter jogado, poderá fazer pequenas modificações no programa, que atenda a seus interesses particulares ou capacidades.

Lembre-se de que o melhor motivo para que se escreva um programa é aprender mais sobre o funcionamento do computador MSX – e computadores em geral. Enquanto você escreve programas e os processa, verá que aumentará seu conhecimento a respeito da lógica, e da ordem, exigidas pelo computador. Terá um melhor conhecimento a respeito das limitações do computador e das limitações inerentes do MSX-BASIC.

Após ter dominado o MSX-BASIC e ter escrito alguns de seus próprios programas, pode ser que você queira aprender outra linguagem. Existe muito em comum entre o BASIC e outras linguagens, de modo que aprender uma segunda linguagem se torna muito mais fácil do que aprender a primeira.

Independentemente da linguagem escolhida, escrever programas ajudará você a aprender mais sobre seu computador. Se você se envolver muito com programação, descobrirá, também, novas idéias de como organizar dados e procedimentos. Assim, aprender a programar ajudará você a entender mais a respeito de informação e de lógica.

CAPÍTULO

6

O SEU PRIMEIRO PROGRAMA MSX-BASIC

A maneira mais fácil de aprender o MSX-BASIC é colocar um programa no computador e analisá-lo. Este capítulo ensina como colocar um programa no computador, armazená-lo em fita ou disco e processá-lo. Você também aprenderá a utilizar algumas técnicas usadas para teclar um programa e fazer pequenas alterações nele. O Capítulo 7 discutirá as diferentes partes de um programa. Os Capítulos 8 até 16 falarão mais a respeito dos comandos e funções específicas do MSX-BASIC.

Um manual de referência do MSX-BASIC sem dúvida acompanhará seu computador. Em vez de ensinar, o manual descreve cada parte da linguagem. À medida que você ler estes capítulos, poderá ser interessante acompanhar pelo manual alguns detalhes sobre os comandos específicos. Este livro cobre todas as partes do MSX-BASIC, não entrando nos detalhes que são encontrados no manual, uma vez que isto normalmente é considerado um fator desfavorável ao aprendizado de uma linguagem. Utilize este livro como ferramenta para aprender e o manual como fonte de referência.

Iniciando o MSX-BASIC

Para que seja possível programar em MSX-BASIC é indispensável que você esteja processando o MSX-BASIC. Se você ligar o seu computador MSX sem inserir nenhum cartucho, ele começará a processar o MSX-BASIC. (Na verdade, quase todos os computadores MSX se comportam deste modo. A um procedimento diverso consulte o manual para ver como deve ser iniciado o MSX-BASIC.) Se você tiver um acionador de disco, com o disco colocado, siga as instruções para dar início ao MSX-BASIC fornecidas no início do Capítulo 18.

Caso você tenha um acionador de disco com seu computador MSX, haverá um programa denominado MSX-DISK BASIC, que é uma versão extensível do MSX-BASIC, significando que é quase o mesmo mas com algumas características a mais. Quando você dá início ao MSX-BASIC com o acionador de disco, também estará dando partida ao MSX-DISK BASIC. Todas estas

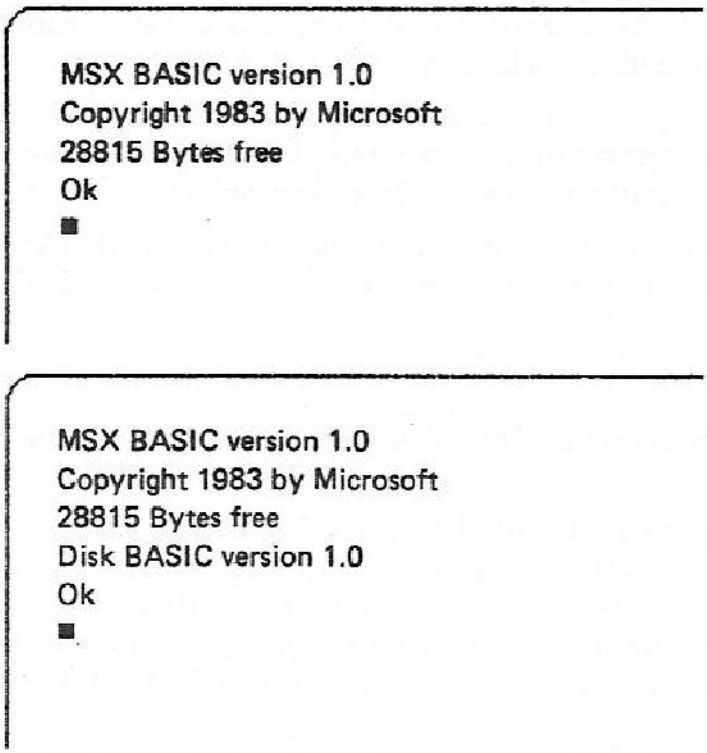


Figura 6.1 Telas iniciais do MSX-BASIC.

características estão relacionadas à utilização do acionador de disco, de modo que quem não possui um acionador de disco não perde nada por ter o MSX-DISK BASIC. Os comandos do MSX-DISK BASIC são mencionados com os comandos MSX-BASIC neste livro; se você não tiver um acionador de disco, ignore-os.

Entrando com Comandos em MSX-BASIC

Uma vez iniciado o processamento do MSX-BASIC, a sua tela deve apresentar-se com uma das apresentações da Figura 6.1. As palavras na parte superior da tela aparecem apenas quando você dá início ao MSX-BASIC; a linha que diz "Ok" significa que ele está pronto para receber instruções. Existe também uma linha com palavras na parte inferior da tela, indicada na Figura 6.2.

color auto goto list run

Figura 6.2 Linha na parte inferior da tela.

O retângulo brilhante que aparece embaixo do pronto "Ok" é denominado cursor. O cursor é importante no MSX-BASIC porque indica aonde irá aparecer a próxima letra teclada. Para verificar, tecle algumas letras, e verifique que elas aparecem onde o cursor se encontra e que o cursor se movimenta para a direita cada vez que uma letra é teclada.

Para apagar aquilo que foi teclado, aperte a tecla denominada BS ou BACKSPACE. A tecla BACKSPACE opera da mesma maneira que a tecla de retrocesso em uma máquina de escrever, exceto que, ao deslocar o cursor para a esquerda, apaga o caractere.

Agora você está pronto para teclar seu primeiro comando MSX-BASIC. Assegure-se de que o cursor está na margem esquerda (caso não esteja utilize a tecla BACKSPACE), e tecle:

```
LIST
```

(Não importa se você tecla com maiúsculas ou minúsculas, ou ambos; este livro mostrará sempre maiúsculas.)

Em seguida pressione a tecla RETURN; esta é marcada com uma flecha apontando para baixo e para esquerda ou com a palavra RETURN. Você encontra esta tecla próximo ao local em que está localizada a tecla RETURN em uma máquina de escrever eletrônica. Pressionar a tecla RETURN, em MSX-BASIC, significa que você quer que o MSX-BASIC execute o comando que foi teclado nesta linha. Neste caso, você quer que o MSX-BASIC execute o comando LIST.

Ao pressionar a tecla RETURN, o MSX-BASIC responde "Ok". Esta, na verdade, não é uma resposta muito interessante, uma vez que o MSX-BASIC lhe informará "Ok" para quase tudo que você fizer. Entretanto, já que o MSX-BASIC não deu indicação de erro, significa que ele recebeu um comando válido. Se o MSX-BASIC tivesse imprimido a mensagem de erro

```
Syntax error
```

antes de imprimir o "Ok", significaria que você lhe deu um comando que ele não entendeu. Se você neste exemplo tivesse recebido uma mensagem de erro, provavelmente teria digitado LIST incorretamente.

Como brincadeira, faça que o MSX-BASIC lhe dê um erro (no futuro isso poderá acontecer com tanta freqüência que você deve ir se acostumando com as mensagens de erro). Tecle

```
LISR
```

e pressione RETURN. Você verá

```
Syntax error
Ok
```

Não importa o que você faça, o MSX-BASIC sempre dirá "Ok" quando estiver pronto para receber mais comandos. "Syntax error" é a maneira de o MSX-BASIC dizer que você lhe deu um comando que ele não conhece, isto é, um que não está em seu vocabulário. Como você pode ver, o MSX-BASIC conhece o comando LIST, mas não a palavra LISR.

CORRIGINDO ERROS DE DIGITAÇÃO NOS COMANDOS

Pela utilização das teclas de controle do cursor o MSX-BASIC permite que sejam corrigidos os seus erros de digitação. Como você deve lembrar-se, as teclas de controle do cursor são aquelas com as setas, normalmente no lado direito do teclado.

Pressione três vezes a tecla com a seta para cima; observe que o cursor se movimenta para cima na tela e pára sobre o "L" em "LISR". Pressione três vezes a tecla com a seta para a direita, e o cursor se encontrará sobre a letra "R" em "LISR" (sem dúvida você será capaz de adivinhar o que vai acontecer se pressionar as teclas com seta para esquerda e seta para direita).

Uma vez que você quer alterar "LISR" para "LIST", deverá substituir o "R" pelo "T". Com o MSX-BASIC isto é bem simples: basta teclar "T" com o cursor sobre o "R". Observe que o comando agora é "LIST". Teclar um caractere sobre o outro resulta em uma substituição (o "R" é esquecido como se nunca tivesse existido). Isto é denominado de teclagem sobreposta. Ao pressionar RETURN, a tela ficará como a Figura 6.3.

```
MSX BASIC version 1.0
Copyright 1983 by Microsoft
28815 Bytes free
Ok
LIST
Ok
LIST
Okntax error

Ok
```

Figura 6.3 Após o segundo comando LIST.

Sem dúvida, "Okntax Error", parece um pouco estranho. Mas, se você pensar um pouco sobre o que foi feito pelo MSX-BASIC verá que faz sentido. Uma vez que "LISR" foi substituído por "LIST", que é um comando válido, a resposta ao comando "LIST" foi exatamente igual à da vez anterior: imprimiu "Ok" na linha seguinte e colocou o cursor na linha que segue ao "Ok". Ele ignorou o que havia na tela antes da alteração do comando. Assim, "Okntax error" é "Syntax error" com o "Ok" sobreposto; o "Ok" que apresenta o cursor sobre o "O" estava na tela desde o momento em que foi teclado "LISR".

O MSX-BASIC permite que você corrija um erro de diversas maneiras; a sobreposição é a mais fácil. Para verificar as outras, sobreponha ao "Ok", que se encontra na mesma linha do cursor, a palavra LIRST, e pressione RETURN. Mais uma vez receberá a mensagem "Syntax error". Utilize as teclas de controle do cursor para movimentar o cursor até o "R" em "LIRST" e pressione a tecla marcada com DEL ou DELETE. Observe que a letra que estava embaixo do cursor (o "R") desapareceu quando foi utilizada a tecla DEL. Pressione RETURN, e o MSX-BASIC executará o comando LIST.

Uma outra tecla prática de se conhecer é INS, que em certos casos é designada como INSERT (INSERIR). Para saber como opera a tecla INS, dê o comando LT (você adivinhou: "Syntax error"!). Coloque agora o cursor sobre o "T" em "LT" e pressione a tecla INS. Observe que o cursor se transforma em uma barra na parte inferior da letra – não no tamanho total da letra. Agora, tecle I, e observe que a letra é inserida à esquerda do cursor, e que a letra embaixo do cursor e aquelas à esquerda deslocam-se para a direita, abrindo um espaço para a nova letra. Tecle S e pressione RETURN.

A lição que se tira daqui é que o MSX-BASIC permite corrigir erros de digitação, mesmo após a teclagem do RETURN. Basta mover o cursor para a linha em que o erro foi cometido e editar a linha por meio de sobreposição, cancelamento ou inserção de caracteres. Ao ser pressionado RETURN, o MSX-BASIC aceita a nova linha como se fosse recém-digitada.

Digitando o seu Primeiro Programa

Agora que você já sabe como inserir os comandos em MSX-BASIC e como corrigir os erros eventualmente ocorridos, está em condições de digitar o seu primeiro programa. Sem dúvida, uma vez que você não conhece muito sobre programação, o texto não terá muito significado. Não se preocupe muito com isto, uma vez que em breve você verá muito mais sobre programação (imagine que esteja assistindo a sua primeira aula de grego e o professor esteja explicando como perguntar em grego onde fica a biblioteca).

O programa que segue é um programa típico em MSX-BASIC. Observe que toda linha inicia com um número; estes são denominados números de linha. Observe que a numeração aumenta à proporção que se avança no programa. O MSX-BASIC utiliza estes números para determinar em que ordem devem ser executadas as instruções.

Por outro lado, você deve ter percebido que a palavra "Comando" tem sido utilizada desde o início do capítulo. Na verdade tudo digitado até o presente foram declarações. Existe apenas uma ligeira diferença entre as duas palavras: antes de um comando existe um número de linha, e no caso da declaração não. Uma declaração é executada imediatamente pelo MSX-BASIC; um comando é memorizado para ser executado posteriormente. Todo comando aprendido neste livro pode ser utilizado, também, como uma declaração e vice-versa.

Digite o programa com muito cuidado, verificando cada linha após completá-la. Os erros de sintaxe serão apontados apenas mais tarde pelo MSX-BASIC, de modo que é melhor evitá-los por meio de uma digitação cuidadosa. Caso você perceba a existência de um erro, faça a correção utilizando os métodos aprendidos na última seção.

```
10  REM Meu primeiro programa MSX-BASIC
20  CLS
30  PRINT "Aqui vem alguns gráficos ..."
40  REM Acerte o relógio e espere 2 seg.
50  TIME = 0
60  IF  TIME < 100 THEN GOTO 60
70  REM Vá ao modo gráfico
80  SCREEN 2
90  REM Faça alguns círculos concêntricos
100 FOR I = 10 TO 50 STEP 5
110 CIRCLE (126,86), I
120 NEXT I
130 REM Espere 2 segundos novamente
140 TIME = 0
150 IF TIME < 100 THEN GOTO 150
160 REM Faça algumas caixas aleatórias
170 FOR J = 1 TO 50
180 LINE (RND(1)*255,RND(1)*195)
    — STEP (RND(1)*50,RND(1)*50),
    RND(1)*15,BF
190 NEXT J
200 REM Espere 2 segundos novamente
210 TIME = 0
220 IF TIME < 100 THEN GOTO 220
230 SCREEN 0
240 PRINT "Tudo executado!"
```

Tenha certeza de que o comando iniciado com "180" é digitado em apenas uma linha, e não em três linhas como é visto aqui. Ao atingir o limite direito da tela, continue teclando (não acione a tecla RETURN); a linha de comando terá continuidade na próxima linha da tela. Neste livro, todas as linhas existentes nas listagens e que tenham mais do que 40 colunas (a largura da tela) continuarão em linhas novas. Porém, ao digitar as linhas, sempre tecle-as como uma linha longa; caso contrário o MSX-BASIC ficará confuso e dará mensagens de erro.

Processando o seu Programa

À medida que você digitava seu primeiro programa, deve ter visto elementos de programação que pode compreender. Uma vez que uma parte das linhas digitadas saiu da tela (também conhecido por scrolling), você eventualmente vai querer rever o programa através do comando LIST. É ainda recomendável rever o programa quanto a erros de digitação.

Entretanto, existe um interesse maior em processar o programa para ver o que ele faz. Assim, está na hora de você ver o segundo comando MSX-BASIC; o comando RUN. Tecle RUN e pressione RETURN. Como você deve ter percebido é necessário digitar RETURN após todos comandos. Daqui em diante, sempre que for dado um comando, este deve ser seguido por um RETURN. Caso não tenha havido erros de digitação, ocorrerá o seguinte:

1. A tela fica fazia;
2. A mensagem "aqui vem alguns gráficos ..." aparece durante dois segundos;
3. A tela fica novamente vazia, e você verá nela oito círculos concêntricos;
4. Dois segundos depois, 50 retângulos de diversos tamanhos e cores aparecerão espalhados na tela; e
5. Dois segundos após os retângulos aparecerem na tela, a tela fica vazia e aparecerá a mensagem "Tudo executado!" assim como o "Ok".

Se tiver havido um erro de digitação e o MSX-BASIC o tenha detectado, limpará a tela imprimindo uma mensagem de erro na linha superior e o "Ok" na linha seguinte. O MSX-BASIC pára de processar o programa assim que ocorrer o erro. Não se preocupe com isto, pois você aprenderá a corrigir estes erros no Capítulo 7.

Dando uma Olhada no seu Programa

Por ora vamos assumir que o programa operou como era de se esperar, e ao processar o programa você viu o que acabamos de descrever. Observe que a tela está limpa, e que não se vê os comandos do programa digitado. Para que o MSX-BASIC liste todo o programa na tela utilize o primeiro comando que aprendeu, o comando LIST. Dê o comando LIST e em seguida RETURN; o programa será listado na tela.

Observe que parte do programa desaparece na parte superior da tela, já que havia mais linhas do que a tela comporta. No Capítulo 7 você verá como listar apenas algumas linhas.

Se houver uma impressora ligada ao seu computador MSX, você poderá utilizar o comando LLIST para imprimir pela impressora em lugar de imprimir na tela. Assegure-se de que a impressora está corretamente ligada e que tenha papel. Dê o comando LLIST seguido de RETURN para imprimir a listagem em papel.

Armazenando o seu Programa em Fita ou Disco

Lembre-se do que foi dito no Capítulo 1, quando o computador é desligado toda informação armazenada em RAM (onde o BASIC armazena os programas digitados) é perdida. Para evitar que seja necessário reteclar tudo novamente, devemos fazer o armazenamento em fita ou disco. Felizmente, isto é relativamente fácil. Mesmo que haja erros no programa, que não tenham sido corrigidos, ele deve ser armazenado em fita ou disco.

Se você tiver um gravador cassete, siga as instruções do manual do computador para ligar o gravador ao computador. Se tiver um acionador de disco, siga as instruções do manual do acionador de disco para ligá-lo ao seu sistema. As duas seções que seguem explicam como preparar o gravador cassete ou o acionador de disco para salvar o programa. Leia a seção referente ao dispositivo que você tiver. A terceira seção explica como utilizar comandos para salvar o programa em fita ou disco, e como recolocar o programa de volta na RAM uma vez que ele foi armazenado.

PREPARANDO O SEU GRAVADOR CASSETE

O MSX-BASIC conhece dois tipos de gravadores cassete: aqueles que têm controle remoto (liga e desliga) do motor, e aqueles que não têm. Se o seu gravador tiver controle remoto, o MSX-

BASIC movimentará a fita para você. Caso o seu gravador de fita cassete não tenha controle remoto, será necessário acionar o botão PLAY do gravador, cada vez que seja preciso movimentar a fita.

Uma vez ligado o cabo entre seu gravador e computador, é necessário inserir um cassete no gravador. A fita deve ser reenrolada até seu início. Se o seu gravador tiver a característica remota, nada irá acontecer quando for pressionado o botão *rewind*; primeiro você terá de desligar o cabo ligado no plugue do remoto.

O MSX-BASIC armazena programas em seu gravador da mesma forma que o seu *tape deck* grava música: do começo ao fim. Se com o seu estéreo, você grava "Parabéns a você" no começo da fita, e reposiciona o gravador, gravando a "Nona Sinfonia", será cancelada a gravação de "Parabéns a você." Da mesma forma, se você gravar um programa no começo da fita faz o rebobinamento e gravar um outro programa, o primeiro programa será perdido para sempre.

Portanto, você deverá tomar cuidado quanto ao local em que será feita a gravação na fita. Você tem duas alternativas: gravar apenas um programa em cada lado da fita, ou utilizar o contador do gravador (se houver) para determinar o local seguro para gravar um novo programa. A utilização de um lado da fita por programa é o método mais seguro, mas o mais caro, uma vez que um programa ocupa apenas alguns segundos da fita. Anotar o número do contador para cada programa gravado e deslocar-se para uma área livre da fita é muito mais econômico, mas leva a uma maior possibilidade de se perder um programa. Lembre-se de que os contadores na maioria dos gravadores mais econômicos não são muito precisos.

Independentemente do método escolhido, você deverá colocar o gravador no modo de "gravação" antes de salvar um programa em fita (normalmente acionando os botões RECORD e PLAY simultaneamente). Para carregar o programa na memória RAM, você deverá colocar o gravador no modo "play".

PREPARANDO O SEU ACIONADOR DE DISCO

Após ligar o acionador de disco ao computador, ele estará pronto para armazenar e recuperar programas. Porém, para este propósito você necessitará de um disquete. Em nenhuma circunstância deverá ser utilizado o disquete de carregamento do MSX-DOS para armazenar programas. Será necessário preparar outro disco.

A operação de preparação de um disco para receber programas é denominado formatação, e será discutida no Capítulo 18. Se você não estiver familiarizado com o conceito de formatação de um disco, deve ler aquele capítulo antes de continuar com este.

Coloque um disquete novo no acionador de disco (se não souber como fazê-lo, leia o manual do acionador). Observe que formatar um disco apaga tudo o que existir nele; assim, você deverá tomar um cuidado todo especial para não formatar o disco do MSX-DOS que acompanha o seu acionador.

É possível formatar um disco enquanto este estiver processando MSX-DOS ou MSX-BASIC. Para formatar um disco enquanto este estiver processando MSX-BASIC, dê o comando CALL FORMAT. Como cada fornecedor adota um procedimento diferente para formatar um disco, é impossível explicar o que ocorre em seguida. É provável que apareça um conjunto de opções na

tela; afortunadamente, estas opções são explicadas no manual do computador ou no manual do acionador de disco. Caso contrário, leia as opções e utilize a mais apropriada.

Após a formatação do disco, ele está pronto para o armazenamento de programas. É uma boa idéia colocar uma etiqueta na capa do disco que descreva o seu conteúdo (como "Meu primeiro programa em MSX-BASIC").

ARMAZENANDO O SEU PRIMEIRO PROGRAMA MSX-BASIC

O programa que você digitou será armazenado pelo MSX-BASIC caso você siga as instruções anteriores para utilização do acionador de disco ou do gravador cassete. Quando você estiver pronto para armazenar o programa, dê um dos seguintes comandos:

Para cassete:	Para disco:
CSAVE "PRIMEI"	SAVE "PRIMEI"

Quando for dado o comando, o disco fará barulho ou a fita irá se movimentar.

Como você deve ter adivinhado, o "C" em CSAVE representa cassete. O nome que aparece após o comando é o nome de seu programa. Quando você for recuperar o programa, deverá utilizar o mesmo nome. Desta maneira você pode armazenar mais do que um programa no mesmo cassete ou disco, uma vez que cada programa terá um nome diferente. O nome não deve ter mais do que seis caracteres para a fita nem mais do que oito para o disco; as aspas não são levadas em conta. No caso acima, "PRIMEI" tem cinco caracteres.

Uma vez que o programa está armazenado, podemos desligar o computador sem receio. Para verificar se o programa foi armazenado corretamente, você deve desligar o computador agora. (Em vez disso você pode dar entrada ao comando NEW; isto elimina o programa da memória sem desligar o computador.)

CARREGANDO O SEU PROGRAMA NA MEMÓRIA

Ligue novamente o computador e dê o comando LIST assegurando-se de que não existe programa algum na memória. Caso esteja utilizando fita, rebobine-a e coloque o gravador no modo "play". Agora, dê um dos seguintes comandos:

Para cassete:	Para disco:
CLOAD "PRIMEI"	LOAD "PRIMEI"

Observe como CLOAD combina com CSAVE e LOAD com SAVE. Dê o comando LIST para ver se o programa foi carregado.

À medida que você escrever outros programas, lembre-se de dar a eles nomes diferentes na hora do armazenamento. Por enquanto, tudo o que foi feito foi a digitação de um programa que você não precisava necessariamente entender; listou-o; e salvou-o em fita ou disco. No próximo capítulo serão discutidos outros aspectos do MSX-BASIC, como é sua estruturação, e como compreender a linguagem.

CAPÍTULO

7

ENTENDENDO UM PROGRAMA MSX-BASIC

Uma vez que você já tem um programa que foi digitado e salvo, deve estar curioso sobre o seu conteúdo e porque a tela apresentou o comportamento visto, durante o processamento do programa. Para que isto seja compreensível, você deverá conhecer um pouco mais a respeito da estrutura de um programa MSX-BASIC. Felizmente, o programa denominado "PRIMEI" é um exemplo adequado de muitos dos elementos do MSX-BASIC.

Você deve lembrar-se de que no Capítulo 5 dizíamos que existem dois elementos básicos em uma linguagem como o MSX-BASIC: gramática e vocabulário. Você já aprendeu um pouco do vocabulário, como LIST, RUN e SAVE. Também aprendeu um pouco de gramática. Como em muitas línguas, a gramática é mais complicada do que o vocabulário. As regras gramaticais que você aprendeu até agora a respeito do MSX-BASIC são:

- Todas as declarações são fornecidas digitando algo do vocabulário do MSX-BASIC e pressionando RETURN;
- Quando você digita uma declaração, em resposta, normalmente alguma coisa ocorre;
- Os comandos são declarações que iniciam com um número de linha;
- Um programa é um conjunto de comandos. À medida que você digita um programa, parece que o MSX-BASIC nada faz a respeito daquilo que foi digitado;
- Para que o MSX-BASIC execute o programa, deve ser dado o comando RUN;
- Alguns comandos aparecem sozinhos na linha (como RUN), enquanto outros aparecem seguidos de outras informações (como em SAVE e LOAD); e
- Você pode dar nomes aos programas.

Editando Linhas de um Programa

É uma boa idéia, neste momento, verificar se o programa "PRIMEI" está correto. Se você tiver passado pela primeira vez sem erros, parabéns; caso contrário, deverá descobrir o que está errado. Felizmente, com o MSX-BASIC isto é mais fácil do que percorrer todo o programa à procura de algo que pareça "estar errado".

Processe o programa novamente até que ocorra uma eventual mensagem de erro. Observe que, agora, a mensagem apresenta algo que o nosso "Erro de sintaxe" não apresentava: um número. Por exemplo, se tiver ocorrido um erro de datilografia na linha 80, como:

```
80 SCREEM 2
```

teria aparecido a mensagem de erro:

```
Syntax error in 80
```

Assim, "in 80" indica que o erro de sintaxe aconteceu na linha 80. Esta linha pode ser corrigida agora pela utilização dos comandos de edição aprendidos no capítulo anterior.

O MSX-BASIC consegue identificar muitos erros, mas não todos. Por exemplo, altere a linha 60 para:

```
60 IF TIME < 10 THEN GOTO 60
```

Obviamente isto não corresponde àquilo que você pensava em digitar. Entretanto, ao processar o programa MSX-BASIC não vai perceber o "erro"; ele vai aguardar durante 0.2 segundos em vez de 2 (no Capítulo 8 você verá por quê).

Mesmo que a mensagem de erro do MSX-BASIC apresente um número de linha, esta não será necessariamente aquela que você tenha cometido o erro de entrada – é aquela em que o MSX-BASIC ficou confuso. À medida que você continuar a programar, aparecerão mais mensagens de erro indicando que existe alguma coisa de errado em uma linha que parece perfeita. Neste caso, dê uma olhada em algumas linhas anteriores para ver se há algo que possa ter causado o problema.

Caso você descubra que existe um erro em uma linha anterior à linha 50, verá que tem um outro problema: toda vez que digitar o comando LIST, aquelas linhas rolam da tela, e você não consegue editá-las. Existe uma solução simples: utilize o comando LIST apenas com o número de linha. Por exemplo, se a linha fosse 20

```
20 CLD
```

e você quisesse alterá-la, dê o comando:

```
LIST 20
```

Neste caso apenas a linha 20 será listada, ficando fácil a sua edição. Você pode dar um comando LIST com uma faixa de números, e serão impressas apenas as linhas desta faixa. Por exemplo, se quiséssemos ver apenas da linha 20 à 80, daríamos o comando

```
LIST 20-80
```

Você acabou de ver um exemplo da seguinte regra gramatical:

– Alguns comandos podem ser dados sozinhos ou com outras informações.

Como será visto adiante, muitos são os comandos que seguem esta regra.

Reveja seu programa de maneira que ao processá-lo ele execute exatamente o que deve. O resultado correto deve ser armazenado em disco ou fita.

As Partes de uma Linha

Como já foi dito, um programa em MSX-BASIC é composto por linhas. A maneira mais fácil de explorar a gramática do MSX-BASIC é entender, primeiro, a gramática de uma única linha de programa. Após ter feito isto, o restante da gramática (e a maior parte do vocabulário) fica bem simples.

Existem três partes em uma linha de um programa MSX-BASIC:

- O número da linha;
- O comando MSX-BASIC;
- A informação pertinente ao comando.

A ordem dos elementos na linha é fixa: primeiro é colocado o número da linha, em seguida vem o comando (com um espaço entre eles), e a última informação vem após o comando.

Existem algumas restrições no MSX-BASIC para as linhas. Uma linha deve ter menos do que 255 caracteres; isto não deve ser problemático a não ser que você tenha cordões muito longos ou deseja colocar vários comandos de uma mesma linha. O número da linha deve, ainda estar entre 1 e 65529.

As palavras ou os números que vêm após o comando são denominados argumentos do comando. Muitos comandos têm argumentos, mas não todos. Imagine o MSX-BASIC sendo a língua portuguesa, com os comandos agindo como verbos e os argumentos como substantivos e advérbios.

À medida que você varre o programa, poderá descobrir outras regras gramaticais:

- Alguns comandos têm mais do que um argumento;
- Alguns argumentos são números, alguns são texto, e outros são ambos; e
- Alguns argumentos são símbolos matemáticos como "= " e "<".

Você pode ainda adicionar uma série de comandos ao seu vocabulário: REM, CLS, PRINT, IF THEN, SCREEN, FOR, CIRCLE, NEXT, LINE. Você conhecerá alguns destes comandos neste capítulo, e os outros nos capítulos que se relacionam com as funções que eles executam.

O leitor mais astuto deve ter reparado que na lista anterior não estava incluída a palavra "TIME" que aparece nas linhas 50, 140 e 210. Assim é porque "TIME" não é um comando. Estas linhas utilizaram um pouco de taquigrafia, utilizado por todo programador BASIC: o comando LET não foi utilizado. Assim, a linha 50 poderia ser:

```
50 LET TIME = 0
```

O único comando que você pode deixar de lado ao programar é o comando LET; por uma questão de otimização de espaço.

O MSX-BASIC permite que você coloque mais do que um comando em uma mesma linha, desde que os comandos sejam separados por dois pontos e o primeiro comando não seja o REM. Por exemplo, as duas linhas

```
230 PRINT "BOM"
240 PRINT "LAGO"
```

podem ser escritos em uma linha

```
230 PRINT "BOM": PRINT "LAGO"
```

Observe que não foi utilizado um segundo número de linha: os dois comandos são considerados partes da mesma linha. A colocação de mais de um comando em uma mesma linha torna o programa mais difícil de ser lido. Porém, este método é útil com o comando IF THEN, como você verá no Capítulo 9.

Agora que você já conhece os componentes de uma linha, é útil a compreensão do relacionamento entre os comandos e seu argumentos. A Tabela 7.1 relaciona o que você já conhece a respeito do vocabulário existente. Na tabela, o X indica que você já viu um exemplo do comando com o número especificado de argumentos; um asterisco (*) indica que o número especificado de argumentos é possível, mas você ainda não o viu.

Na Tabela 7.1, REM e LET estão indicados como tendo apenas um argumento, apesar de que você pode pensar, ao olhar o programa, que eles podem ter diversos argumentos. No comando REM, tudo aquilo que segue o comando é ignorado pelo MSX-BASIC, de modo que se trata apenas de um argumento, independentemente do número de palavras. Todo comando LET é seguido por uma equação "alguma coisa é igual a alguma coisa", como em "TIME = 0", que

Tabela 7.1 Comandos e seus argumentos.

Comando	Sem Argumento	1 Argumento	Vários Argumentos
LIST	X	X	*
RUN	X	*	*
CSAVE		X	*
SAVE		X	*
NEW	X		
CLOAD		X	*
LOAD		X	*
REM	*	X	
CLS	X		
PRINT	*	X	*
SCREEN		X	*
CIRCLE			X
NEXT		X	
LINE			X
LET		X	

é considerado um argumento. IF e FOR não são incluídos na tabela porque se trata de comandos com diversas partes, de difícil classificação.

Observe que os comandos PRINT do programa têm apenas um argumento cada: tudo o que estiver dentro das aspas é um único argumento. Dizemos que as aspas delimitam o argumento, significando que indicam ao MSX-BASIC onde o argumento começa e onde termina. As aspas também delimitam o argumento "nome-do-programa" em CSAVE, SAVE, CLOAD e LOAD.

Um aspecto da tabela que chama a atenção é como muitos dos comandos aceitam mais do que um argumento. Você verificará que a maioria dos comandos do MSX-BASIC pode ter diversos argumentos que executam tarefas diferentes.

Exemplos de Comandos do MSX-BASIC

Agora que você já conhece as partes de uma linha de programa, pode começar a aprender os comandos específicos que apareceram em seu primeiro programa. É útil ver agora os comandos que são mencionados neste capítulo, porque eles aparecem em quase todos os programas. Alguns destes comandos serão abordados com maiores detalhes em outros capítulos.

A primeira linha do programa é um comando REM. REM é a representação de *remark* (observação), que é o propósito do comando REM. Você utiliza o comando REM pelo programa todo para informar o que as diversas partes do programa fazem. Por exemplo, a linha 10 em seu programa "PRIMEI" lhe informa a finalidade do programa como um todo, a linha 40 informa o que fazem as linhas 50 e 60, a linha 70 indica a linha 80, e assim por diante.

Poderá parecer estranha a necessidade de fazer tantas observações a respeito do que está desejando ao escrever o programa. À proporção que seu programa se complica, você verá que vai utilizar maior número de "truques" de programação. Quanto maior o programa, mais difícil será lembrar o truque ou tarefa executado por um determinado trecho do programa. É neste momento que você valoriza as observações feitas com o comando REM.

Se você deseja alterar a atuação de um programa ou corrigir algo que não está funcionando direito, levará muito tempo procurando a área que deve ser alterada. Se tivessem sido utilizados vários comandos REM para explicar a finalidade de cada seção e fossem explicados os truques, seria muito mais fácil ler o seu programa. A utilização do REM desta maneira é denominada documentação do programa.

Observando o processamento do programa você poderá ter adivinhado o propósito do comando PRINT, que imprime seus argumentos na tela. Como já foi visto, o argumento do comando PRINT é delimitado pelas aspas. Na linha 30 temos um comando PRINT típico; tudo o que estiver entre aspas (sem incluí-las) aparecerá na tela quando o MSX-BASIC executar o comando PRINT.

Como foi anteriormente visto, a linha 50 na verdade tem um comando LET, apesar de ter sido omitido o termo LET, como em quase todos os programas em BASIC. O argumento que vem depois do número da linha é formado pelo nome da variável, um sinal de igual (=), e um valor. As variáveis representam os valores numéricos, da mesma forma que na álgebra elementar, e são descritas em maiores detalhes no Capítulo 8. Por enquanto, o que é necessário compreender é que o comando LET altera os valores da variável.

Organização do Programa

Como você já viu, o MSX-BASIC executa as linhas de um programa na ordem crescente dos números das linhas. Uma vez que você deseja que o MSX-BASIC execute certas instruções antes de outras, o número dado para cada linha é muito importante.

Um padrão adotado pelos programadores BASIC é o de numerar as linhas em intervalos de 10, em lugar de 1, 2, 3, 4 e assim por diante. A razão principal para este procedimento é a necessidade posterior que pode haver em acrescentar comandos ao meio do programa, e esta é a maneira de deixar espaço reservado para uma eventualidade deste tipo. Por exemplo, se você quiser acrescentar uma linha entre as linhas 20 e 30, esta poderá receber o número 25. A linha 25 pode ser digitada em qualquer lugar vazio da tela; ao dar o comando LIST da próxima vez, você a verá entre as linhas 20 e 30.

A maioria dos programas, entretanto, não é processada, do começo ao fim, linha por linha. É comum que os programas sofram desvios, significando que algumas linhas são puladas de acordo com determinadas condições. Por exemplo, você pode escrever um programa que faça uma pergunta sim ou não. Se a resposta for sim deverão ser executadas certas linhas do programa, se a resposta for não, serão executadas outras linhas do programa.

Para tomar estas decisões, devem ser utilizados os comandos IF e GOTO. Eles são explicados em maiores detalhes no Capítulo 9, mas será interessante ver um exemplo aqui. Limpe a memória dando o comando NEW. Digite o seguinte programa:

```
10 REM Exemplo da utilizacao de desvio
20 LINE INPUT "Entre $ ou N: "; QUESTAO$
30 IF RESPOSTA$ = "$"  THEN GOTO 70
40 IF RESPOSTA$ = "N"  THEN GOTO 90
50 PRINT "Voce nao inseriu $ ou N"
60 GOTO 100
70 PRINT "Voce digitou $"
80 GOTO 100
90 PRINT "Voce digitou N"
100 END
```

Este programa deve ser salvo em fita ou disco com o nome "SOUN" (representando "SIM OU NÃO").

Processe o programa e insira a letra S quando o programa pergunta "Insira S ou N: " em seguida pressione RETURN (certifique-se de que esteja digitando S e não s). O programa informará que você digitou "S".

Você já sabe o que a linha 10 faz: nada. O comando LINE INPUT da linha 20 imprime a mensagem delimitada pelas aspas e, em seguida, aguarda que sejam teclados alguns caracteres seguidos de RETURN. Em seguida dá à variável QUESTÃO$ o valor do caractere teclado (você verá mais a este respeito no próximo capítulo); em relação ao LINE INPUT o Capítulo 10 tratará disso, por ora, assuma que RESPOSTA$ contém a letra que você teclou.

Nas linhas 30 e 40 existe o comando IF THEN (é mais apropriado denominar estes comandos de "IF THEN" do que de "IF", porque todo comando IF tem um THEN). O comando IF

THEN é denominado comando de desvio condicional, já que o MSX-BASIC desvia para outra linha caso uma determinada condição seja atendida. A linha 30 informa ao MSX-BASIC: "Se a pessoa deu entrada ao S, pule todas as linhas até a linha 70." O comando GOTO informa ao MSX-BASIC em que ponto deve ser iniciada a execução das linhas se a condição entre o IF e o THEN for verdadeira.

A lógica seguida pelo programa é indicada na Figura 7.1. Resumidamente, a linha 30 verifica se você entrou com S e o envia à linha 70 em caso positivo; a linha 40 verifica se você entrou com N e o envia à linha 90 em caso positivo; caso as condições das linhas 30 e 40 não sejam verdadeiras, você acaba caindo na linha 50.

Como pode ver pelas linhas 60 e 80, é possível utilizar um comando GOTO sem um comando IF. Como você provavelmente já adivinhou, um comando GOTO sem o IF é denominado desvio incondicional. A razão de se utilizar o GOTO na linha 60 é para impedir a execução da linha 70 após a linha 50. Após a impressão de "Você não entrou com S ou N", o programa já terminou e não há necessidade de nova impressão. Assim, você deve evitar as linhas que imprimem "Você digitou S" e "Você digitou N"; o desvio incondicional contorna estas linhas e vai para a linha 100.

A linha 100 tem o comando END, que você ainda não viu. A única coisa que o comando END faz é o de ordenar ao MSX-BASIC que pare de processar o programa. Você deve lembrar que o seu programa "PRIMEI" não terminou com um comando END. Isto é correto, uma vez que o MSX-BASIC sabe que deve parar quando não tem mais comandos para executar. Neste programa, entretanto, havia a necessidade de enviar o programa para algum lugar após a impressão de uma das três mensagens. O comando END é um fim adequado para qualquer problema, mas não é obrigatório ...

Este programa é um exemplo simples da utilização dos comandos IF THEN e GOTO em seus programas. Como será visto no Capítulo 9, o comando GOTO pode enviá-lo para linhas anteriores. Programas mais longos conterão dezenas de GOTO que o enviarão para o programa todo.

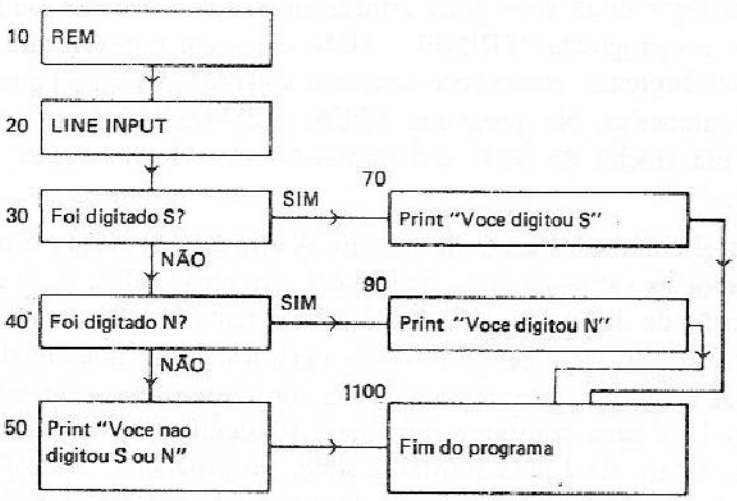


Figura 7.1 Diagrama esquemático do programa SOUN.

CAPÍTULO

8

VARIÁVEIS, CONSTANTES, FUNÇÕES E ARITMÉTICA

Como podemos observar pelo título deste capítulo, ele cobre diversos tópicos ao fim dos quais, você estará sabendo quase tudo sobre a gramática do MSX-BASIC, e saberá bem mais sobre o vocabulário. Os Capítulos 10 a 16 são organizados com base nos comandos descritos neste capítulo.

Variáveis e constantes são os dois últimos conceitos de importância que você deve aprender no MSX-BASIC. São os argumentos utilizados com maior freqüência nos comandos. As funções são similares aos comandos, mas não aparecem no começo da linha como os comandos. Como você já deve ter adivinhado, aritmética é exatamente igual àquela que você aprendeu no primário: adição, subtração etc.

Variáveis e Constantes

Variável é um lugar onde você pode armazenar dados. Os dados podem ser números ou letras. Por exemplo, no programa "PRIMEI", TIME e I eram variáveis que continham números (TIME é uma variável diferente, como você verá mais adiante). A variável que contém um número é chamada variável numérica. No programa SOUN, RESPOSTA$ é uma variável que contém texto. Em BASIC um trecho de texto é denominado *série* de caracteres; assim, RESPOSTA$ é uma variável *série*.

Pense na variável como uma caixa de correio. A etiqueta da caixa é o nome dado à variável. Podemos colocar dados na caixa, os quais podem ser retirados e lidos. Você pode alterar os dados da caixa pela colocação de dados novos. Se você estiver familiarizado com a álgebra elementar, as variáveis do MSX-BASIC são idênticas às variáveis algébricas. Você pode recordar as duas variáveis mais comuns em suas aulas de álgebra: X e Y. Uma constante é algo cujo valor não se altera. Por exemplo, o número 15 é uma constante numérica: 15 significa sempre 15. Quando você explicitamente define um texto, ele é uma constante *série*. No programa "SOUN", a *série* da linha 90 que era "Você digitou N" é uma constante. As constantes *série* são sempre delimitadas por aspas. Constantes não têm nomes – apenas valores.

O MSX-BASIC permite que você dê nomes para as suas variáveis. O nome de uma variável pode ter qualquer comprimento, mas o MSX-BASIC olha apenas os dois primeiros caracteres. O primeiro caractere deve ser uma letra, e os demais podem ser letras ou números (excluindo-se os sinais de pontuação).

TIPOS DE VARIÁVEIS E CONSTANTES

No MSX-BASIC existem quatro tipos de variáveis e constantes: inteiros, números reais de precisão simples, números reais de precisão dupla e *séries*. (Caso você tenha esquecido, um número real é aquele que tem um ponto decimal.) Cada tipo é diferente; entretanto, a única diferença entre precisão simples e precisão dupla nos números reais é a precisão do número (precisão de 6 dígitos ou precisão de 14 dígitos) e a quantidade de memória que ocupam (2 bytes ou 4 bytes).

Existe um limite imposto pelo MSX-BASIC à faixa que um número pode tomar. Os inteiros são números entre -32768 e 32767. Números reais devem estar entre 10^{-64} e 10^{63}.

As *séries* são armazenadas na memória, um caractere por byte. Como a memória dos computadores é inerentemente numérica, e não em forma de texto, cada caractere é associado a um número. Os números variam de 0 a 255. O padrão que inter-relaciona uma letra a um número é denominado código ASCII (esta é uma sigla arcaica que não tem sentido hoje). O Apêndice C mostra os códigos ASCII para os caracteres que seu computador MSX pode gerar.

NOMES E VALORES DE VARIÁVEIS

À medida que o MSX-BASIC executa o seu programa ele pode saber que tipo de variável está sendo utilizada, bastando olhar o símbolo que se encontra no fim do nome da variável.

Tabela 8.1 Símbolos para os tipos de variáveis.

Tipo de Variável	Símbolo	Exemplos
Inteiro	%	Tela%, FAIXA%
Precisão simples real	!	GG1!, GENTE!
Precisão dupla real	# ou vazio	ESTAÇÃO#, BOLA
Série	$	RESPOSTA$, ANUNCIO$

A Tabela 8.1 mostra os últimos caracteres para cada tipo de variável. Observe que, se não for colocado um símbolo no fim do nome da variável, o MSX-BASIC assume que você está utilizando uma variável de dupla precisão.

Existe outra regra que deve ser lembrada ao se escrever um programa: as variáveis não podem ter o mesmo nome que as palavras reservadas do MSX-BASIC que é composto por tudo que está no vocabulário do MSX-BASIC. Esta regra impede que o MSX-BASIC confunda algo que você esteja utilizando como uma variável, com um comando, uma função etc. De fato, o nome de suas variáveis não pode conter palavra alguma reservada do MSX-BASIC, nem no meio deste nome. A Figura 8.1 relaciona as palavras reservadas.

ABS
AND
ASC
ATN
ATTR$
AUTO
BASE
BEEP
BIN$
BLOAD
BSAVE
CALL
CDBL
CHR$
CINT
CIRCLE
CLEAR
CLOAD
CLOSE
CLS
CMD
COLOR
CONT
COPY
COS
CSAVE
CSNG
CSRLIN
CVD
CVI
CVS
DATA
DEF
DEFDBL
DEFINIT
DEFSNG
DEFSTR
DELETE
DIM
DRAW
DSKF$
DSKI$
DSKO$
ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FN
FOR
FPOS
FRE
GET
GO
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP
INPUT
INSTR
INT
IPL
KEY
KILL
LEFT$
LEN
LET
LFILES
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MAX
MERGE
MID$
MKD$
MKI$
MKS$
MOD
MOTOR
NAME
NEW
NEXT
NOT
OCT$
OFF
ON
OPEN
OR
OUT
PAD
PAINT
PDL
PEEK
PLAY
POINT
POKE
POS
PRESET
PRINT
PSET
PUT
READ
REM
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SCREEN
SET
SGN
SIN
SOUND
SPACE$
SPC
SPRITE
SQR
STEP
STICK
STOP
STR$
STRIG
STRING$
SWAP
TAB
TAN
THEN
TIME
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VDOP
VPEEK
VPOKE
WAIT
WIDTH
XOR

Figura 8.1 Palavras reservadas do MSX-BASIC.

Veja novamente o programa "SOUN":

```
10 REM Exemplo da utilizacao de desvio
20 LINE INPUT "Digite S ou N:" ; RESPOSTA$
30 IF RESPOSTA$ = "S" THEN GOTO 70
40 IF RESPOSTA$ = "N" THEN GOTO 90
50 PRINT "Voce nao digitou S ou N"
60 GOTO 100
70 PRINT "Voce digitou S"
80 GOTO 100
90 PRINT "Voce digitou N"
100 END
```

Na linha 20, "Digite S ou N:" é uma constante e RESPOSTA$ é uma variável *série* (agora você vê porque o sinal $ foi utilizado neste programa). Quando o comando LINE INPUT é executado, o MSX-BASIC lê o teclado até que seja pressionada a tecla RETURN para em seguida, colocar os caracteres teclados na variável *série* que você nomeou. Você é obrigado a utilizar uma variável *série* com o comando LINE INPUT. Na linha 30, RESPOSTA$ é, evidentemente, ainda uma variável e S uma constante. O comando IF THEN compara o conteúdo da variável RESPOSTA$ com a constante S.

A maioria dos comandos que você viu com constantes pode aceitar variáveis. Por exemplo o programa que segue age exatamente da mesma forma que o programa "SOUN":

```
10 REM Exemplo da utilizacao de variaveis
20 REM em lugar de costantes no
30 REM   programa SOUN
40 S RESPOSTA$ = "S"
50 N RESPOSTA$ = "N"
60 NAOSOUN$ = "Voce nao digitou S ou N"
70 SMGS$ = "Voce digitou S"
80 NMSG$ = "Voce digitou N"
90 LINE INPUT "Digite S ou N: "; RESPOSTA$
100 IF RESPOSTA$ = SRESPOSTA$ THEN GOTO 140
110 IF RESPOSTA$ = NRESPOSTA$ THEN GOTO 160
120 PRINT NAOSOUN$
130 GOTO 170
140 PRINT SMSG$
150 GOTO 170
160 PRINT NMSG$
170 END
```

Observe que as constantes SRESPOSTA$, NRESPOSTA$, NAOSOUN$, SMSG$ e NMSG$ são definidas das linhas 40 até 80 e utilizadas no resto do programa. Este não é um método padronizado de programação mas um bom exemplo de como é possível utilizar variáveis em muitos locais de seu programa.

O erro mais comum que se comete ao escrever um programa MSX-BASIC é esquecer de que o MSX-BASIC apenas olha para as duas primeiras letras de uma variável. Esta regra deve ser lembrada ao nomear uma variável nova. Por exemplo, observe o seguinte programa, bastante simples:

```
10 REM Exemplo do nome de uma variavel comum
20 REM  fornecida por engano
30 VAR1 = 5
40 VAR2 = 6
50 PRINT VAR1
60 PRINT VAR2
```

À primeira vista você pode achar que o programa vai imprimir "5" e depois "6". Em lugar disto, imprime "6" e depois "6" já que o MSX-BASIC reconhece apenas a parte "VA" do nome das variáveis.

Quando o programa é processado pelo MSX-BASIC, tudo se passa como se você tivesse escrito:

```
10 REM Exemplo do nome de uma variavel comum
20 REM fornecida por engano
30 VA = 5
40 VA = 6
50 PRINT VA
60 PRINT VA
```

Na linha 50, VA é igual a 6.

Se a utilização do caractere no fim do nome das variáveis se tornar monótona, o comando DEF pode ser utilizado para informar ao MSX-BASIC que todas as variáveis que iniciam com uma determinada letra representam um determinado tipo de variável. Os comandos são DEF DBL, DEF INT, DEF SNG e DEF STR para reais de precisão dupla, inteiros, reais de precisão simples, e *séries* de caracteres. Você dá o comando com uma letra ou um grupo de letras. Por exemplo,

```
10 DEF SNG C-E
20 DEF INT I
```

informa ao MSX-BASIC que toda variável que tem como primeira letra C, D ou E será um número real de precisão simples, e toda a variável em que a primeira letra seja I é inteira.

Existem variáveis especiais que o MSX-BASIC define para você. Elas serão mencionadas no decorrer do próximo capítulo, embora você já conheça uma delas: TIME. TIME é uma variável numérica a qual se soma 1 a cada 1/50 de segundo. Você pode usar TIME em qualquer ponto de seu programa.

Isto explica a utilização de TIME no programa "PRIMEI": ela foi colocada em 0, esperando até que ficasse igual a 100. Já que ela é atualizada a cada 1/50 de segundo, será 2 segundos. Nas linhas 50 e 60 deste programa tem-se:

```
50 TIME = 0
60 IF TIME < 100 THEN GOTO 60
```

Quando o MSX-BASIC chega à linha 60, verifica-se que o valor de TIME é menor do que 100. Uma vez que seu computador MSX é muito rápido, a primeira vez que o MSX-BASIC chega a 60, TIME provavelmente estará em 0. Assim, o MSX-BASIC vai até a linha 60, reexecutando a linha. Até que o valor de TIME chegue a 100, 0 MSX-BASIC continuará reexecutando a linha 60. Quando TIME atingir 100, o MSX-BASIC observa que "TIME < 100" é falso e continua até a linha 70.

Se você estiver confuso sobre o procedimento TIME, processe o seguinte programa:

```
10 REM Experimentando TIME
20 PRINT "Aqui vem TIME"
30 TIME = 0
40 PRINT TIME
50 REM Passaram 5 segundos?
60 IF TIME < 250 GOTO 40
70 PRINT "Passaram os seus 5 segundos"
```

MATRIZES

Em certas situações é conveniente termos um conjunto de variáveis que possam ser manipuladas com um único nome. Uma matriz tem esta característica. Podemos ter matrizes de *séries* de caracteres, números reais de precisão simples, números reais de precisão dupla ou inteiros. Cada elemento da matriz tem um valor próprio e pode ser tratado como uma variável separada.

A matriz é definida com o comando DIM. A estrutura mais simples deste comando é

```
DIM nomevar(n)
```

Nesta estrutura, nomevar é o nome dado para as variáveis de sua matriz. O n representa o elemento de maior ordem da matriz. Toda matriz inicia com a numeração em 0, de maneira que, na verdade, é criado um conjunto de "n+1" elementos, com o comando DIM. Por exemplo, para definir uma matriz de 15 elementos denominada LL, deve ser dado o comando

```
10 DIM LL(14)
```

É possível dar comandos em seu programa MSX-BASIC que pode utilizar estas variáveis, como em

```
50 LL(5) = 23.5
60 N = 7
70 LL(N) = 9.1
```

A linha 50 assume a variável LL(5), enquanto a linha 70 assume a variável LL(7).

Por ora vimos apenas matrizes com um conjunto de elementos, conhecidas também como unidimensional. É possível termos matrizes com mais do que uma dimensão, bastando acrescentá-las ao comando DIM. Por exemplo, para criar uma matriz denominada QN com 12 linhas e 17 colunas, dê o comando

```
120 DIM QN(12, 17)
```

Para obter uma determinada variável desta matriz, devem ser fornecidas as duas variáveis da matriz.

```
180 QN(7,5) = 3: PRINT QN(7,5)
```

UTILIZANDO OUTROS SISTEMAS NUMÉRICOS

O MSX-BASIC pode manipular constantes com diversos tipos de notações. Apesar de a maioria das pessoas ter mais familiaridade com a notação decimal, muitos conhecem a notação científica, algumas vezes chamada de forma exponencial. Um número representado na notação científica é formado por um inteiro ou número real à esquerda (chamado mantissa), seguido por um "E" e em seguida por um inteiro (o expoente). O MSX-BASIC interpreta este número como 10 elevado ao valor do expoente e em seguida multiplicado pela mantissa.

Por exemplo, o número "4.732E5" é o mesmo que 47,320. Um número em notação científica com um "E" é interpretado pelo MSX-BASIC como sendo um número de precisão simples. Para especificar um número de precisão dupla, utilize um "D" no lugar do "E" (como "4.732D5").

Uma vez que o computador pensa na forma binária, às vezes é conveniente exprimir os números na notação binária, octal ou hexadecimal. Se estas palavras não têm significado para você, sinta-se à vontade para passar à próxima seção.

Se você desejar expressar constantes na forma binária, poderá antepor um "&B" ao número. Para especificar um octal, utilize "&O", e para especificar um hexadecimal utilize "&H". As linhas que seguem estabelecem para a variável VG o número 431 (decimal):

```
VG = &B110101111    (Notação Binária)
VG = &0657          (Notação Octal)
VG = &H1AF          (Notação Hexadecimal)
```

Funções

A função é uma operação que processa um número ou uma *série* de caracteres. Podemos considerar as funções como subprogramas; são semelhantes às funções matemáticas. As funções têm argumentos, assim como os comandos, mas os argumentos de uma função são colocados entre parênteses após o nome da função. Algumas funções têm apenas um argumento, outras têm mais de um.

Por exemplo, você deve estar lembrando da raiz quadrada em suas aulas de matemática. A raiz quadrada de um número tem como resultado um outro número; ao multiplicar o segundo número por ele próprio, teremos o primeiro número. Em MSX-BASIC, utilizamos a função SQR para encontrar a raiz quadrada de um número. A estrutura da função SQR é

```
SQR(n)
```

O número do qual queremos encontrar a raiz quadrada é (n). Por exemplo:

```
30 B = SQR(A)
```

Neste caso, B assume o valor da raiz quadrada de A.

Uma função pode ser utilizada em todos os locais em que se pode utilizar uma constante, uma vez que uma função sempre tem um valor, da mesma maneira que uma constante. De fato, podemos utilizar uma função dentro de outra função. Por exemplo, se desejar que J1 seja o logaritmo de seno de J2, utilize o comando LET:

```
50 LET J1 = LOG(SIN(J2))
```

Ao encontrar esta expressão o MSX-BASIC resolve primeiro "SIN(J2)" para em seguida calcular o logaritmo do resultado.

No MSX-BASIC existem funções matemáticas para quase todas as suas necessidades. Elas estão relacionadas na Tabela 8.2. Existem ainda muitas funções de *série*, relacionadas na Tabela 8.3. A Tabela 8.4 relaciona as funções que ajudaram na conversão de números para a *séries* e vice-versa.

Tabela 8.2 Funções matemáticas do MSX-BASIC.

Função	Significado
ABS(NÚMERO)	Valor absoluto do NÚMERO
ATN(NÚMERO)	Arcotangente, em radianos.
COS(RADNÚMERO)	Coseno, em radianos.
EXP(NÚMERO)	Expoente (e elevado à potência de NÚMERO).
FIX(NÚMERO)	Retorna o NÚMERO retirando toda sua parte fracionária.
INT(NÚMERO)	Faz o arredondamento para o primeiro inteiro menor que NÚMERO.
LOG(NÚMERO)	Logaritmo natural (base e) do NÚMERO.
SGN(NÚMERO)	Retorna +1 caso NÚMERO seja positivo, -1 caso NÚMERO seja negativo e 0 quando NÚMERO for 0.
SIN(RADNÚMERO)	Seno, em radianos.
SQR(NÚMERO)	Raiz quadrada do NÚMERO.
TAN(RADNÚMERO)	Tangente, em radianos.

Tabela 8.3 Funções *série* do MSX-BASIC.

Função	Significado
INSTR(INÍCIO, *SÉRIE1*$, *SÉRIE2*$)	Encontra a posição da primeira aparição de *série* 2$ em *série* 1$, partindo da posição INÍCIO (o argumento INÍCIO é opcional). No caso de não se encontrar *série* 2$ em *série* 1$ o resultado é 0.
LEFT$(*SÉRIE*$, COMPRIMENTO)	Cria uma *série* de COMPRIMENTO caracteres de comprimento com os caracteres mais à esquerda de *SÉRIE*$.
LEN(*SÉRIE*$)	Fornece o comprimento de uma *SÉRIE*$.
MID$(*SÉRIE*$, INÍCIO, COMPRIMENTO)	Retorna uma *série* com COMPRIMENTO caracteres de comprimento, iniciando na posição INÍCIO; COMPRIMENTO é um argumento opcional.
RIGHT$(*SÉRIE*$, COMPRIMENTO)	Cria uma *série* de COMPRIMENTO caracteres de comprimento com os caracteres à direita de *SÉRIE*$.
SPACE$(COMPRIMENTO)	Retorna uma *série* COMPRIMENTO caractere de espaço.
STRING$(COMPRIMENTO, CARACTERE$)	Gera uma *série* com COMPRIMENTO repetições do caractere CARACTERE$.

Tabela 8.4 Funções de conversão do MSX-BASIC.

Função	Significado
ASC(*SÉRIE*$)	Retorna o código ASCII do primeiro caractere de *SÉRIE*$.
BIN$(NÚMERO)	Retorna uma *série* que representa NÚMERO em binário.
CDBL(NÚMERO)	Um número real de dupla precisão.
CHR$(NÚMERO)	Retorna uma *série* de comprimento 1 do caractere cujo código ASCII é NÚMERO.
CINT(NÚMERO)	Retorna um inteiro.
CSNG(NÚMERO)	Retorna um número de simples precisão.
HEX$(NÚMERO)	Retorna uma *série* que representa NÚMERO em hexadecimal.
OCT$(NÚMERO)	Retorna uma *série* que representa NÚMERO em octal.
STR$(NÚMERO)	Retorna uma *série* que representa NÚMERO como um número real.
VAL(*SÉRIE*$)	Retorna um número real de dupla precisão que representa o número armazenado em *SÉRIE*$.

Apesar de muitas desta funções serem auto-explicativas, uma exemplificação ajuda a esclarecer sua utilização.

```
10 REM Amostra das funcoes MSX-BASIC
20 REM Primeiro, as funcoes numericas
30 A = 5
40 B = 10
50 C = -3.7
60 D = .8245
70 PRINT "O valor absoluto de " ; C; "e" ; ABS(C)
80 PRINT "O arco-tangente de "; A; "e"; ATN(A)
90 PRINT "O co-seno de "; D; "e" ; COS(D)
100 PRINT "O exponencial de "; A; "e"; EXP(A)
110 PRINT "O FIX de "; C: "e" ; FIX(C)
120 PRINT "O FIX de "; B; "e"; FIX(B)
130 PRINT "O INT de "; B; "e"; INT(B)
140 PRINT "O INT de "; C; "e"; INT(C)
150 PRINT "O logaritmo de "; A; "e"; LOG(A)
160 PRINT "O sinal de "; A; "e"; SGN(A)
170 PRINT "O sinal de "; C; "e"; SGN(C)
180 PRINT "O seno de "; D; "e"; SIN(D)
190 PRINT "A raiz quadrada de "; A; "e"; SQR(A)
200 PRINT "A tangente de "; D; "e"; TAN(D)
210 REM em seguida, as funcoes série
220 X$ = "rainha de copas"
230 COMPRIMENTO = 6
240 INICIO = 4
250 SÉRIE1$ = "como ela brilha"
260 SÉRIE2$ = "ela"
270 CARACTERE$ = "g"
280 PRINT "INSTR exemplo resulta ***";
    INSTR(INICIO, SÉRIE1$, SÉRIE2$); "***"
290 PRINT "LEFT$ exemplo resulta ***";
    LEFT$(X$,COMPRIMENTO); "***"
300 PRINT "LEN exemplo resulta ***";
    LEN(X$); "***"
310 PRINT "MID$ exemplo resulta ***";
    MID$(X$,INICIO,COMPRIMENTO); "***"
320 PRINT "RIGHT$ exemplo resulta ***";
    RIGHT$(X$,COMPRIMENTO); "***"
330 PRINT "SPACE$ exemplo resulta ***";
    SPACE$(COMPRIMENTO); "***"
340 PRINT "STRING$ exemplo resulta ***";
    STRING$(COMPRIMENTO,CARACTERE$);
350 REM por fim, as funcoes de conversao
```

```
360 X$ = "d"
370 PRINT "O valor em ASCII de "; X$; "e"; ASC(X$)
380 NUMERO = 100
390 PRINT "O valor binario de"; NUMERO; "e"; BIN$(NUMERO)
400 PRINT "CHR$ exemplo resulta ***";
    CH$(NUMERO); "***"
410 PRINT "O valor hexadecimal de ";
    NUMERO; "e"; HEX$(NUMERO)
420 PRINT "O valor octal de"; NUMERO:
    "e"; OCT$(NUMEROC)
430 PRINT "O valor decimal de";
    NUMERO; "e"; STR$(NUMERO)
440 X$ = "-34.99"
450 PRINT "VAL exemplo resulta"; VAL(X$)
```

A impressão deste programa resulta no seguinte (certifique-se de que está entendendo):

```
O valor absoluto de -3.7 é 3.7
O arcotangente de 5 é 1.3734
O coseno de .8245 é .678924
O exponencial de 5 é 148.413
O FIX de -3.7 é -3
O FIX de 10 é 10
O INT de 10 é 10
O INT de -3.7 é -4
O logaritmo de 5 é 1.60944
O sinal de 5 é 1
O sinal de -3.7 é -1
O seno de .8245 é .734208
A raiz quadrada de 5 é 2.23607
A tangente de .8245 é 1.08143
INSTR exemplo resulta ***6 ***
LEFT$ exemplo resulta ***rainha ***
LEN exemplo resulta ***15 ***
MID$ exemplo resulta ***ha da ***
RIGHT$ exemplo resulta ***copas***
SPACE$ exemplo resulta ***  ***
STRING$ exemplo resulta ***ggggggg ***
O valor em ASCII de d é 100
O valor binário de 100 é 1100100
CHR$ exemplo resulta ***d ***
O valor hexadecimal de 100 é 64
O valor octal de 100 é 144
O valor decimal de 100 é 100
VAL exemplo resulta -34.99
```

Existem mais duas funções que não apresentam a mesma facilidade para serem descritas que aquelas que acabamos de ver. As duas funções são MID$ = e RND. Apesar de terem muito em comum, a função MID$ = é diferente da função MID$. O nome da função MID$ = , demonstra que a função MID$ é utilizada do lado esquerdo de um comando LET. De fato, a função MID$ = é a única função que pode aparecer do lado esquerdo de um comando LET. Lembre-se de que o comando LET iguala a variável que está do lado esquerdo do sinal de igual ao valor da direita. Se a função MID$ for utilizada do lado esquerdo do sinal de igual, o MSX-BASIC é informado que deve alterar o valor de uma parte da *série*.

O programa que segue mostra como se utiliza a função MID$ = ;

```
10 REM Exemplo de MID$ =
20 ASÉRIE$ = "aaaaaaaaaa"
30 BSÉRIE$ = "bbbbbbbbbb"
40 MID$(ASÉRIE$ 3, 4) = BSÉRIE$
50 PRINT ASÉRIE$
```

O programa imprime:

```
aabbbbaaaa
```

A função RND gera um número randômico entre 0 e 1. Isto será útil quando você quiser simular um evento randômico, como jogar um dad. Existem três tipos diferentes de argumentos numéricos que podem ser fornecidos para a função RND:

- Qualquer número positivo fará com que seja gerado um número randômico de precisão dupla, entre 0 e 1.
- Qualquer número negativo fará com que o gerador de números randômicos inicie uma nova lista de números randômicos; isto é denominado semeadura. Se você não semear o gerador de números randômicos cada vez que você processa um programa, os números randômicos gerados serão os mesmos (o que, sem dúvida, contradiz o propósito dos números randômicos).
- Caso você utilize o argumento 0, o gerador lhe dará o último número randômico anteriormente recebido do gerador.

O programa abaixo mostra como utilizar os números randômicos:

```
10 REM Exemplo de numero randomico
20 FOR I = 1 TO 10
30 PRINT RND(%)
40 NEXT I
50 REM  Faca nova semeadura no gerador
60 X = RND(-TIME)
70 FOR I = 1 TO 10
80 PRINT RND(5)
90 NEXT I
```

Uma vez que a variável TIME se altera a cada 1/50 segundo, semear o gerador de números randômicos com RND(-TIME) é uma boa maneira de gerar números randômicos.

Aritmética do MSX-BASIC

Já que o MSX-BASIC é utilizado com freqüência para executar operações de matemática, ele possui um número grande de operações matemáticas. Estas operações se enquadram em duas categorias, numéricas e lógicas. Operações numéricas são aquelas que você aprendeu no primário e no ginásio (adição, subtração etc.). Operações lógicas são aquelas utilizadas para comparar dois elementos e determinar se a comparação é verdadeira ou falsa.

A Tabela 8.5 relaciona as operações numéricas. Como pode ver, quase todas são diretas e provavelmente familiares para você. Os operadores menos familiares para a maioria das pessoas são: divisão inteira, resto inteiro e exponenciação.

Tabela 8.5 Operadores numéricos do MSX-BASIC.

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
\	Divisão com resultado inteiro
MOD	Resto inteiro de uma divisão
^	Exponenciação

A utilização do operador de divisão com resultado inteiro sempre resulta em um inteiro. O resultado desta divisão é o mesmo que o da divisão real, com a parte fracionária eliminada. Resto de um inteiro (MOD) é o resto da divisão entre dois inteiros. Quando colocamos o operador de exponenciação entre dois números, o resultado será o primeiro número elevado ao segundo número.

As operações numéricas são demonstradas no programa que segue:

```
 10 REM Exemplos de operacoes
 20 X = 3.4
 30 Y = 6.6
 40 PRINT X + Y
 50 PRINT X - Y
 60 PRINT X * Y
 70 PRINT X/Y
 80 Z1 = 64
 90 Z2 = 15
100 PRINT Z1 \ Z2
110 PRINT Z1 MOD Z2
120 PRINT X ^ Y
```

O programa imprime:

```
 10
-3.2
 22.44
 .515152
 4
 4
 3219.29
```

Podemos utilizar o parêntese para agrupar partes de declarações matemáticas. O parêntese tem o mesmo significado para o MSX-BASIC que para a matemática usual: as operações dentro do parêntese são executadas como entidades separadas. No exemplo que segue, valor tem como resultado 45:

```
240 VALOR = 5*((1 + 2) ^ 2)
```

Como se vê, o MSX-BASIC permite que se tenha um parêntese dentro de outro parêntese.

Se o parêntese não é utilizado, o MSX-BASIC utiliza o critério da precedência para avaliar os operadores. Isso significa que certas operações serão executadas antes que outras; por exemplo a multiplicação é executada antes da soma. Observe a dificuldade que existe para lembrar a precedência de uma linha como a seguinte:

```
40 QUAL =93.4 + 73/72.8 * 8-3
```

Os bons programadores utilizam sempre o parêntese, sendo este um hábito que você deve adquirir.

Operadores relacionais são utilizados com maior freqüência nas declarações IF THEN para comparar dois números ou duas *séries*. Até aqui, já vimos dois operadores relacionais: "<" foi utilizado na linha 60 de "PRIMEI", e "=" foi utilizado na linha 30 de "SOUN". Quando o MSX-BASIC avalia uma relação, o resultado é 0 se a relação for verdadeira, e um número diferente de zero (normalmente 1) se a relação for falsa. A Tabela 8.6 mostra todos os operadores relacionais.

Tabela 8.6 Operadores relacionais do MSX-BASIC.

Operador	Condição a ser Verificada
=	Igualdade
<>	Desigualdade
<	Menor do que
<=	Menor ou igual a
>	Maior do que
>=	Maior ou igual a

No MSX-BASIC existem ainda operadores lógicos, similares aos operadores relacionais. Os operadores lógicos são utilizados entre dois valores lógicos (isto é, entre dois valores que são verdadeiros ou falsos). Os operadores lógicos seguem as definições padronizadas para comparação, indicadas na Figura 8.2.

O operador NOT é um operador unário: opera apenas no argumento que o segue. Como é de se esperar NOT fornece como resultado o oposto do argumento (falso fica verdadeiro e verdadeiro fica falso). O programa que segue utiliza os operadores relacionais e lógicos. É utilizada uma forma nova da declaração IF THEN: IF THEN ELSE. Caso a condição entre o IF e o THEN seja verdadeira, a declaração que segue o THEN é executada; se a condição é falsa, a declaração após o ELSE é executada.

X	Y	X AND Y	X OR Y	X EQV Y	X IMP Y	X XOR Y
1	1	1	1	1	1	0
1	0	0	1	0	0	1
0	1	0	1	0	1	1
0	0	0	0	1	1	0

Figura 8.2 Operadores lógicos do MSX-BASIC.

```
10 REM Exemplo de operadores
20 REM logicos e relacionais
30 P = 10
40 Q = 20
50 PRINT "P = Q  e";
60 IF P = Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso"
70 PRINT "P <> Q e";
80 IF P <> Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso '
90 PRINT "P < Q e";
100 IF P < Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso"
110 PRINT "P < = Q E
120 IF P < = Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso"
130 PRINT "P > Q e";
140 IF P > Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso"
150 PRINT "P > = Q e";
160 IF P > = Q THEN PRINT "verdadeiro" ELSE
   PRINT "falso"
```

```
170 REM Posicione TR em verdade, e FA em falso
180 TR = 0
190 FA = 1
200 PRINT "Verdadeiro AND falso e";
210 IF TR AND FA THEN PRINT "verdadeiro" ELSE
    PRINT "falso"
220 PRINT "Verdadeiro'XOR Falso e;
230 IF TR XOR FA THEN PRINT "verdadeiro" ELSE
    PRINT "falso"
```

Podemos colocar outros valores neste programa para testar outras relações lógicas.

MANIPULANDO *SÉRIES* DE CARACTERES

As mesmas operações de adição e comparação que se fazem com números podem ser executadas com *séries* de caracteres. Quando adicionamos duas *séries* (chamadas de concatenação), o MSX-BASIC cria uma *série* nova com as duas. Por exemplo,

```
10 REM Concatenacao de séries
20 K$ = "Liberdade"
30 L$ = "de escolha"
40 M$ = K$ + L$
50 PRINT M$
```

resulta em

```
Liberdade de escolha
```

Os operadores relacionais que utilizamos com números podem ser utilizados, também, com *séries*. O MSX-BASIC compara duas *séries* comparando um caractere por vez, a partir da esquerda. Todas as comparações são baseadas nos valores ASCII dos caracteres. Assim a comparação

```
"Abc" < "a"
```

seria verdadeira, uma vez que "A" tem um valor menor (em ASCII) que "a" e o MSX-BASIC ignora o comprimento da *série*. Se duas *séries* têm os mesmos caracteres, mas um é mais curto, ele é considerado menor que o mais longo.

Agora que já foram vistas todas as maneiras em que podem ser utilizadas as variáveis e fazer contas de matemática com o MSX-BASIC, estamos prontos para começar a escrever um programa. Os capítulos que se seguem discutem comandos e funções que o ajudarão a escrever programas não usuais.

CAPÍTULO

9

CONTROLE DO PROGRAMA

No Capítulo 7 você aprendeu um pouco sobre o controle do fluxo de um programa; este vai explicar muitas outras maneiras de controlar um programa. Os comandos utilizados até o presente, que impedem o MSX-BASIC de simplesmente ler e executar as linhas em seqüência, são IF THEN, IF THEN ELSE e GOTO. No programa "PRIMEI" você conheceu o comando FOR, mas não aprendeu como ele opera. Este capítulo discute todos estes, e alguns comandos de controle usados em programas mais avançados.

Os Comandos IF THEN e IF THEN ELSE

Quanto maior for a quantidade de programas que você escrever, mais utilizará os comandos IF THEN e IF THEN ELSE. A razão é bem simples: todos os programas úteis terão de tomar decisões baseadas em valores que podem ser alterados.

O comando IF THEN ELSE é um caso especial do comando IF THEN. Em um comando IF THEN, o MSX-BASIC executa a próxima linha caso a condição que foi avaliada seja falsa; no comando IF THEN ELSE, o MSX-BASIC executa o comando que segue ao ELSE se a condição for falsa. Já que os dois comandos são muito semelhantes, serão discutidos simultaneamente ao comando IF THEN.

Lembre-se de que a condição existente entre IF e THEN pode ser bastante complexa, apesar de ser permitida apenas uma condição. Quando utilizamos uma condição muito complexa, devemos utilizar o parêntese para impedir que o MSX-BASIC fique confuso. Por exemplo, a linha abaixo verifica cinco variáveis:

```
130 IF (((DIA$ = "12") AND (MES$ = "Outubro")
    AND (ANO% = 1985)) XOR (NOT(V1 < = V2)))
    THEN GOTO 190
```

Existem duas maneiras principais de utilizar o comando IF THEN:

```
IF condição THEN ação
IF condição THEN GOTO número de linha
```

Utilizamos o primeiro tipo de comando IF THEN quando se quer realizar uma única coisa e a condição é verdadeira, e utilizamos o segundo tipo de IF THEN para a maioria das outras finalidades.

Uma utilização comum do comando IF THEN com um parâmetro de ação é a mudança de uma variável baseada em seu valor. Por exemplo, caso seja solicitada uma inserção em letras maiúsculas, embora o usuário tenha digitado letras minúsculas, pode haver interesse em alterar a variável que contém a resposta para as maiúsculas. O que segue é uma variação do programa SOUN que entende "N", "n", "S" ou "s":

```
10 REM Um novo exemplo para utilizacao de desvio
20 LINE INPUT "Entre S ou N: "; RESPOSTA$
22 IF RESPOSTA$ = "s" THEN RESPOSTA$ = "S"
24 IF RESPOSTA$ = "n" THEN RESPOSTA$ = "N"
30 IF RESPOSTA$ = "S" THEN GOTO 70
40 IF RESPOSTA$ = "N" THEN GOTO 90
50 PRINT "Voce nao entrou com S ou N"
60 GOTO 100
70 PRINT "Voce digitou S"
80 GOTO 100
90 PRINT "Voce digitou N"
100 END
```

As linhas 22 e 24 foram acrescentadas para lidar com as letras minúsculas.

Se você seguir o fluxo de controle deste programa verá que não é muito eficiente. Por exemplo, se o usuário digitar "s", o MSX-BASIC verá isto só na linha 22; é um desperdício executar as linhas 24 e 30 antes de passar para a linha 70. Neste caso, provavelmente seria uma boa idéia colocar outro comando após a declaração IF THEN das linhas 22 e 24. Lembre-se de que a maneira de fazer isto é a colocação de dois pontos entre as duas declarações.

Se, após a declaração IF THEN, aparecer um segundo comando, ele será executado após o argumento de ação, quando a condição for verdadeira:

```
10 REM  Um novo exemplo para utilizacao de desvio
20 LINE INPUT "Entre S ou N: " ; RESPOSTA$
22 IF RESPOSTA$ = "s" THEN RESPOSTA$ = "S"; GOTO 70
24 IF RESPOSTA$ = "n" THEN RESPOSTA$ = "N"; GOTO 90
30 IF RESPOSTA$ = "S" THEN GOTO 70
40 IF RESPOSTA$ = "N" THEN GOTO 90
50 PRINT "Voce nao entrou com S ou N"
60 GOTO 100
```

```
 70 PRINT "Voce digitou S"
 80 GOTO 100
 90 PRINT "Voce digitou N"
100 END
```

Agora, a linha 22 informa ao MSX-BASIC que, se a RESPOSTA$ é "s", deve substituir o valor de RESPOSTA$ por "S", e seguir para a linha 70. Neste programa, não há necessidade de substituir o valor de RESPOSTA$ antes de seguir para a linha que imprime a mensagem, mas isto seria necessário se outras linhas dependessem de RESPOSTA$. Estes acréscimos farão seu programa mais elegante, bem como mais rápido.

Existe outra maneira de otimizar este programa. A linha 40 pode ser alterada para um comando IF THEN ELSE com um argumento de ação e outro comando; deste modo é possível eliminar as linhas 50 e 60. O resultado é

```
 10 REM "Um novo exemplo para utilizacao de desvio
 20 LINE INPUT "Entre S ou N: "; RESPOSTA$
 22 IF RESPOSTA$ = 's" THEN RESPOSTA$ = "S" : GOTO 70
 24 IF RESPOSTA$ = "n" THEN RESPOSTA$ = "N" : GOTO 90
 30 IF RESPOSTA$ = "S" THEN GOTO 70
 40 IF RESPOSTA$ = "N" THEN GOTO 90 ELSE
    PRINT "Voce nao entrou com S ou N": GOTO 100
 70 PRINT "Voce digitou S"
 80 GOTO 100
 90 PRINT "Voce digitou N"
100 END
```

Este programa, ao ser processado, se comporta exatamente como o programa "SOUN".

MAIS A RESPEITO DE MODIFICAÇÕES DE PROGRAMAS

A discussão que se segue lhe dará uma visão maior de como modificar um programa. O momento é propício, uma vez que você acabou de alterar o programa "SOUN" adicionando e eliminando linhas.

Como você viu, a adição de uma linha é simples: basta dar-lhe um número que esteja entre os números das linhas onde vai ser inserida a nova linha. Ao ser dado o comando LIST, a nova linha aparece entre as outras. Para eliminar linhas, utilize o comando DELETE. O argumento do comando DELETE é semelhante ao do comando LIST, isto é, uma faixa de números. Se, após a modificação da linha 40, quisermos eliminar as linhas 50 até a 60 basta digitar:

```
DELETE 50-60
```

A última versão deste programa apresenta um comando MSX-BASIC bastante interessante, muito útil nas modificações de programas. Observe que a sua numeração de linhas, tão ordenada, foi corrompida pela adição de algumas linhas e eliminação de outras. Se houver interesse em que

a numeração de suas linhas volte a ser feita em incrementos de 10, sem espaços entre elas, utilize o comando RENUM. Dê o comando RENUM e em seguida o comando LIST: RENUM: LIST

```
10 REM Um novo exemplo para utilizacao de desvio
20 LINE INPUT "Entre S ou N:"; RESPOSTA$
30 IF RESPOSTA$ = "s" THEN RESPOSTA$ = "S": GOTO 70
40 IF RESPOSTA$ = "n" THEN RESPOSTA$ = "N": GOTO 90
50 IF RESPOSTA$ = "S" THEN GOTO 70
60 IF RESPOSTA$ = "N" THEN GOTO 90 ELSE
   PRINT "Voce nao entrou com S ou N":
   GOTO 100
70 PRINT "Voce digitou S"
80 GOTO 100
90 PRINT "Voce digitou N"
100 END
```

Sua numeração volta a ser ordenada de 10 em 10.

O comando RENUM tem uma característica muito potente que não foi vista no último programa. Acrescente a linha que segue ao programa:

```
55 REM A proxima linha e longa
```

Dê, agora, os comandos RENUM e LIST; o resultado é

```
10 REM Um novo exemplo para utilizacao de desvio
20 LINE INPUT "Entre S ou N: "; RESPOSTA$
30 IF RESPOSTA$ = "s" THEN RESPOSTA$ = "S": GOTO 80
40 IF RESPOSTA$ = "n" THEN RESPOSTA$ = "N": GOTO 100
50 IF RESPOSTA$ = "S" THEN GOTO 80
60 REM A proxima linha e longa
70 IF RESPOSTA$ = "N" THEN GOTO 100 ELSE
   PRINT "Voce nao entrou com S ou N":
   GOTO 110
80 PRINT "Voce digitou S"
90 GOTO 110
100 PRINT "Voce digitou N"
110 END
```

Observe que a linha que era 60, agora é 70, e a linha que era 70, agora é 80, e assim por diante. À primeira vista você pode entrar em pânico pensando que todas as declarações GOTO estão erradas, mas se comparar esta versão com a anterior, verá que o comando RENUM mudou todos os comandos GOTO para você. Assim, sempre que você desejar, pode utilizar o comando RENUM, e estar confiante de que todos os números de linhas, estejam eles dentro de um comando, ou no começo de uma linha, terão a correspondência correta.

Os Comandos FOR e NEXT

Em muitas situações precisamos executar uma tarefa repetidas vezes. Por exemplo, você pode se lembrar da ocasião em que foi obrigado a escrever 100 vezes no quadro-negro "Eu não vou dormir!". Os comandos FOR e NEXT tornam este tipo de tarefa muito mais fácil. Processe o seguinte programa:

```
10 REM Terceira serie
20 FOR CONT = 1 TO 100
30 PRINT "Eu nao vou dormir"
40 NEXT CONT
50 PRINT "Posso ir para casa agora?"
```

As linhas rolam rapidamente pela tela, mas o MSX-BASIC realmente escreve a linha 100 vezes.

Ao comando FOR sempre está associado um comando NEXT: esta é a razão pela qual as linhas intermediárias são denominadas loop FOR NEXT. As linhas 20 até 40 do programa anterior são um loop FOR-NEXT. A estrutura do loop FOR-NEXT é

```
FOR varloop = numin TO numfin
.
.
.
outros comandos . . .
.
.
.
NEXT varloop
```

O "varloop" utilizado nos comandos FOR e NEXT é uma variável numérica, e para "numin" e "numfin" podemos utilizar qualquer número (ou variável) desde que "numin" seja maior do que "numfin".

Quando o MSX-BASIC encontra o comando FOR, dá ao "varloop" o valor de "numin". Em seguida processa normalmente os demais comandos. Quando encontra o comando NEXT, volta ao comando FOR, adicionando 1 ao "varloop", e verificando se a variável é maior do que "numfin". Se não é maior, repete novamente o loop; se é maior, passa ao comando que segue NEXT.

O que segue é um exemplo simples de um loop FOR-NEXT. Observe que a variável H1 é impressa dentro do loop, o que é perfeitamente aceitável, uma vez que ela é como outra variável qualquer.

```
10 REM Um FOR-NEXT loop simples
20 PRINT "Iniciando"
30 FOR H1 = 1 TO 5
40 PRINT H1
50 NEXT H1
60 PRINT "Terminado"
```

Ao processar este programa será feita a seguite impressão:

```
Iniciando
1
2
3
4
5
Terminando
```

Apesar de ser permitido alterar o valor de uma variável de loop dentro do loop, isso não é uma boa idéia uma vez que tal operação poderá ser esquecida ao ser feita outra alteração posterior.

O MSX-BASIC permite que você coloque qualquer comando dentro de um loop FOR-NEXT. De fato, é comum termos um loop FOR-NEXT dentro de outro; a isto chamamos de aninhar loops FOR-NEXT. Quando você aninha loops, deve estar bem seguro de que o loop interno está completamente dentro do loop externo, já que o MSX-BASIC dará uma mensagem de erro se encontrar uma variável errada em um comando NEXT.

O programa que segue forma quadrados de 10 × 10 pontos com os números de 100 até 199. Isto pode parecer bem difícil de ser feito, se uma programação complicada; como você vê, é bem fácil fazê-lo com loops FOR-NEXT aninhados.

```
10 REM Faca um quadrado de 10 × 10 pontos
20 FOR X = 10 TO 19
30 FOR Y = 0 TO 9
40 PRINT (10*X) + Y;
50 NEXT Y
60 PRINT
70 NEXT X
```

Você verá no próximo capítulo porque é que o ponto e vírgula (;), após o comando PRINT, faz os números aparecerem na mesma linha.

O comando FOR poderá ter o argumento STEP, que informa ao MSX-BASIC o quanto deve ser somado à variável cada vez que passa pelo NEXT. Caso seja utilizado o argumento STEP, o formato do loop FOR-NEXT é:

```
FOR varloop = numin TO numfin STEP passval
.
.
.
outros comandos . . .
.
.
.
NEXT varloop
```

Em vez de somar 1, o "passval" é somado ao "varloop" pelo MSX-BASIC a cada passagem feita pelo loop. O programa anterior pode ser reescrito:

```
10 REM Faca um quadrado de 10 × 10
20 FOR X = 100 TO 190 STEP 10
30 FOR Y = 0 TO 9
40 PRINT X + Y;
50 NEXT Y
60 PRINT
70 NEXT X
```

Se for necessário poderá ser utilizado um valor negativo para "passval", o que tem grande utilidade em contagens regressivas.

```
10 REM Contagem regressiva
20 FOR Q = 37 TO 28 STEP -1
30 PRINT Q
40 NEXT Q
```

Que causará a seguinte impressão:

```
37
36
35
34
33
32
31
30
29
28
```

Os Comandos GOSUB e RETURN

Uma característica de grande parte das linguagens modernas de programação são as sub-rotinas. A sub-rotina é um programa menor dentro de um maior. Ela é útil quando existe um conjunto de passos que devem ser executados repetidas vezes, em diferentes locais do programa. Em vez de repetir sempre os mesmos passos, criamos uma sub-rotina, fazemos um desvio até ela quando necessário e depois voltamos à parte principal do programa.

O MSX-BASIC tem uma capacidade limitada de fazer sub-rotinas. O comando GOSUB o envia para uma sub-rotina (GOSUB significa "vá até a sub-rotina"). Toda sub-rotina termina com o comando RETURN, que instrui o MSX-BASIC que volte para a linha seguinte ao comando GOSUB. Ao entrar em uma sub-rotina, sempre é necessária a utilização do comando GOSUB, pois assim o comando RETURN saberá para onde retornar.

O exemplo adequado de uma sub-rotina é um programa que imprime as palavras de uma música qualquer. Em vez de ter muitas linhas que imprimam "Falalalalalalalala", podemos ter uma sub-rotina para esta impressão.

```
10 REM Uma estranha maneira de imprimir uma
20 REM cancao
30 PRINT "Enfeite o salao com luzes prateadas,"
40 GOSUB 200
50 PRINT "E epoca de estar feliz".
60 GOSUB 200
70 PRINT "Senhor conhecemos nossa missao,"
80 GOSUB 200
90 PRINT "Cantarolemos a velha cancao."
100 GOSUB 200
110 END
200 REM Esta e a sub-rotina do Falala
210 PRINT "Fa";
220 FOR X = 1 TO 8
230 PRINT "la";
240 NEXT X
250 PRINT
260 RETURN
```

Ao encontrar o comando GOSUB, o MSX-BASIC sai da linha 40, vai para a linha 200 e memoriza que da próxima vez que encontrar o comando RETURN, deverá retornar à linha 50. Na sub-rotina, o MSX-BASIC imprime "Fa" seguido de oito "la"s (o loop FOR-NEXT é sem dúvida fútil, mas deve servir de exemplo para a importância de um loop). Na linha 250 termina a linha "Falalalalalalalala" da tela e a 260 informa ao MSX-BASIC que é o fim da sub-rotina.

Observe a linha 110: o "fim" do programa está no meio. Se você pensar sobre a execução de um programa, verá que este é realmente o melhor local para colocar a declaração END. Se a linha 110 não existisse, o MSX-BASIC executaria a linha 200 após retornar da última chamada da sub-rotina. Ao executar a linha 260, teríamos a seguinte mensagem de erro:

```
RETURN WITHOUT GOSUB IN 260
```

Isto ocorre porque o MSX-BASIC quer retornar ao programa, mas não sabe para onde, já que a sub-rotina não foi executada com o auxílio de um comando GOSUB. Esta é a razão pela qual os programadores BASIC normalmente colocam a declaração END antes do início de uma sub-rotina. O MSX-BASIC permite que se tenha tantas declarações END quantas se desejar em um programa.

Comandos ON GOTO e ON GOSUB

A utilidade do comando IF THEN é a de verificar uma única condição e escolher um entre dois caminhos, dependendo do resultado. Existem, entretanto, locais nos programas onde uma decisão poderá ter mais do que duas respostas possíveis. Por exemplo, você poderá pedir ao usuário que digite A, B, C ou D. A utilização de quatro comandos IF THEN é um desperdício, de modo que o MSX-BASIC possui os comandos ON GOTO e ON GOSUB.

A estrutura do comando ON GOTO é

```
ON numérico GOTO linha1,linha2,...
```

O comando ON GOSUB apresenta uma estrutura similar. O termo "numérico" é uma expressão que rètorna um número entre 0 e 255. Caso a variável seja avaliada em 1, 0 MSX-BASIC vai para o número de linha linha1; se for avaliada em 2, o MSX-BASIC vai até o número de linha linha2, e assim por diante. Caso a variável seja avaliada em 0 ou um número que é maior do que o comprimento da lista, o MSX-BASIC assume que você quer ir até a linha que segue o comando ON GOTO.

Como se vê, utilizar o ON GOTO para escolhas conhecidas é muito fácil. No exemplo mencionado acima, o usuário digita A, B, C ou D e pressiona a tecla RETURN. O programa a seguir mostra como utilizar o ON GOTO para isto:

```
10 REM Exemplo de ON GOTO
20 LINE INPUT "Digite A, B, C ou D: "; LETRA$
30 ON (ASC(LETRA$)-64) GOTO 60,80,100,120
40 PRINT "Nao era A, B,C ou D!"
50 END
60 PRINT "Voce digitou A"
70 END
80 PRINT "Voce digitou B"
90 END
100 PRINT "Voce digitou C"
110 END
120 PRINT "Voce digitou D"
130 END
```

A expressão no comando ON GOTO utiliza o operador ASC para encontrar o código ASCII da letra digitada. Uma vez que o código ASCII de A é 65, ASC(LETRA$)-64 resultará 1 para A, 2 para B e assim por diante. Tente digitar uma letra minúscula para ver o que ocorre com números maiores que a lista de números de linhas.

Interrompendo e Parando um Programa

Este capítulo mostrou como você pode controlar a execução das partes de um programa. Entretanto, o usuário também tem algum controle sobre o que ocorre em um programa.

Enquanto um programa estiver sendo processado, o usuário pode interromper a execução pressionando a tecla STOP. O MSX-BASIC apenas estará sendo informado para nada fazer até que o usuário volte a acionar novamente a tecla STOP. Assim, o usuário aciona a tecla STOP para interromper temporariamente o programa e pressiona-a novamente para que continue.

Pressionar simultaneamente a tecla CTRL e STOP devolve você ao MSX-BASIC. Isso é útil quando você está experimentando um programa e quer testar diversas partes. De maneira geral, a combinação CTRL-STOP não é um bom método para interromper programas, mas em uma emergência é muito prática.

CAPÍTULO

10

INSERÇÃO VIA TECLADO E EXIBIÇÃO DE TEXTO

A maioria dos programas é controlada digitando-se as instruções no teclado e vendo os resultados na tela. Este capítulo apresenta os comandos que executam estas tarefas de inserção e saída. Por exemplo, você já viu como o comando LINE INPUT obtém o texto do usuário, e o comando PRINT apresenta o texto na tela. Estes comandos, e aqueles a eles relacionados, serão descritos em detalhes neste capítulo.

Armazenando Inserções pelo Teclado

Ao escrever um programa que exige a digitação de um determinado texto pelo usuário, normalmente apresentamos uma mensagem na tela e aguardamos a inserção. O MSX-BASIC armazena-a em uma variável (normalmente uma variável *série* de caracteres), que você poderá utilizar pelo resto do programa.

O COMANDO INPUT

O comando INPUT é a maneira mais simples de se conseguir caracteres do teclado. Seu formato é

```
INPUT "mensagem"; variável1,variável2,...
```

A mensagem é opcional. Quando o MSX-BASIC se depara com um comando INPUT, imprime um ponto de interrogação e um espaço, para em seguida, aguardar uma resposta do usuário terminada com a tecla RETURN. Se você inclui uma *série* de orientação, ela será impressa pelo MSX-BASIC antes do ponto de interrogação. Ao fornecer uma resposta, o usuário pode utilizar as teclas INS, DEL e BACKSPACE.

O caso mais simples de um comando INPUT é aquele com apenas uma variável *série*:

```
450 INPUT JX$
```

Neste caso, o MSX-BASIC lê a linha teclada pelo usuário armazenando-a na *série* JX$, exceto nas seguintes circunstâncias:

- O usuário não deve incluir uma vírgula no texto. Quando o MSX-BASIC vê uma vírgula, assume que duas *séries* foram digitadas em vez de uma e imprime a mensagem

  ```
  ?Extra ignored
  ```

  Apenas o texto antes da vírgula será colocado na variável.

- O primeiro caractere não deve ser um ponto de interrogação, a menos que o usuário não pretenda que o mesmo faça parte da *série*. Quando o MSX-BASIC vê o ponto de interrogação no começo da linha assume que o usuário está teclando uma *série* delimitada por um ponto de interrogação. Assim, não a coloca na variável.

- Para incluir uma vírgula no texto, este deve começar e terminar com aspas.

Incidentalmente, o caractere RETURN não é colocado na *série*.

Se houver interesse em se aceitar duas *séries*, elas deverão ser separadas por uma vírgula. Por exemplo:

```
450 INPUT JX$,KX$
```

O usuário deve digitar a primeira *série*, uma vírgula e a segunda *série*. Caso não seja fornecida a segunda *série*, o MSX-BASIC informa

```
??
```

que talvez seja a mensagem de erro mais confusa que você poderá obter do MSX-BASIC. Ela indica que o MSX-BASIC está aguardando mais inserções.

Se for utilizado o comando INPUT com uma variável numérica, o usuário deve teclar um número. Por exemplo, se o comando INPUT é

```
230 INPUT X
```

e o usuário teclar "xxx", será dada a seguinte mensagem pelo MSX-BASIC:

```
?Redo from start
?
```

Esta talvez seja a segunda mensagem de erro mais confusa que você pode receber do MSX-BASIC. É a maneira de o MSX-BASIC informar ao usuário que o que foi digitado não era um número.

Caso seu comando INPUT tenha muitas variáveis, e pelo menos uma delas é numérica, e seja feita uma teclagem incorreta, surgirá a mensagem de erro anterior e o usuário terá de repetir toda a lista novamente. Por exemplo,

```
170 INPUT VW$,U,M
```

O COMANDO LINE INPUT

Como foi visto na discussão anterior, o comando INPUT não é o melhor modo de se conseguir informação do usuário. O comando LINE INPUT é muito mais limpo, sendo esta a razão de ter sido utilizada nos capítulos anteriores.

A estrutura do comando LINE INPUT é similar à do comando INPUT

```
LINE INPUT "mensagem";variável$
```

A mensagem é opcional. Observe que o comando LINE INPUT aceita apenas uma variável, que deve ser uma *série* de caracteres. Assim fica tudo muito mais simples para o MSX-BASIC, uma vez que não precisa procurar vírgulas ou números de variáveis. Ainda mais, o comando LINE INPUT não imprime o ponto de interrogação.

O programa abaixo permite que o usuário veja o tipo de texto que pode ser inserido usando o comando LINE INPUT. Processe-o, tentando diversas *séries* diferentes, incluindo caracteres que poderiam confundir o comando INPUT, como as vírgulas, pontos de interrogação etc. Ao terminar, pressione a tecla RETURN sozinha (isto é, insira uma *série* que não contenha nada).

```
10 REM Experimentando com LINE INPUT
20 LINE INPUT "Tecle alguma coisa → "; V$
30 IF V$ = " " THEN END
40 PRINT V$
50 GOTO 20
```

Deste modo você deve convencer-se de que o comando LINE INPUT aceita tudo o que você tecla.

A FUNÇÃO INPUT$ E A VARIÁVEL INKEY$

Há instantes em que você quer receber dados do teclado, mas não deseja forçar o usuário a pressionar a tecla RETURN. Caso você saiba a quantidade exata de caracteres que o usuário deve teclar, pode ser utilizada a função INPUT$. O argumento de INPUT$ é o número de caracteres que você está aguardando.

Por exemplo, a primeira versão do programa "SOUN" poderá ser escrita de modo a aceitar um caractere sem obrigar ao usuário operar a tecla RETURN. A linha 20 pode ser convertida de

```
20 LINE INPUT "Digite S ou N:"; RESPOSTA$
```

para

```
20 PRINT "Digite S ou N: ";
25 RESPOSTA$ = INPUT$(1)
```

Ao processar isto, observe que a letra digitada não apareceu na tela: esta é outra diferença entre a função INPUT$ e o comando LINE INPUT. Podemos imprimir o caractere pela adição de

```
27 PRINT RESPOSTA$;
```

O último método para a introdução de texto é a variável INKEY$. Esta variável é semelhante à função INPUT$, exceto que a variável INKEY$ retém o caractere que está sendo pressionado pelo usuário no momento em que o MSX-BASIC lê a linha com INKEY$. Se neste momento o usuário não está pressionando tecla alguma INKEY$ resulta em uma *série* vazia. Por outro lado a variável INKEY$ aceita apenas um caractere.

A utilização da variável INKEY$ não é fácil, já que o usuário não está pressionando a tecla com freqüência. Entretanto, se você verificar continuamente se INKEY$ está vazio (normalmente denominado nulo), é possível identificar com facilidade a teclagem do usuário. Por exemplo, podemos utilizar INKEY$ no programa "SOUN". Pode-se converter a linha 20 de

```
20 LINE INPUT "Digite S ou N: "; RESPOSTA$
```

para

```
20 PRINT "Digite S ou N: ";
25 IF INKEY$ = " " THEN GOTO 25 ELSE RESPOSTA$ = INKEY$
27 PRINT RESPOSTA$;
```

A diferença fundamental entre a utilização da variável INKEY$ e a função INPUT$ é que o cursor não aparece na tela ao processar o loop INKEY$ na linha 25. Isto poderá confundir alguns usuários que esperam ver o cursor; entretanto, se você não quiser o cursor na tela utilize INKEY$.

INSERINDO CARACTERES ESPECIAIS

Até o presente, você apenas aprendeu como inserir caracteres que são vistos no teclado, como números, letras e símbolos de pontuação. O computador pode mostrar muitos outros caracteres, e todos podem ser digitados. No Apêndice C são mostrados todos os caracteres que podem ser impressos. Por exemplo, para imprimir uma letra "A", maiúscula, com um trema, deve ser dado o comando

```
PRINT CHR$(142)
```

Pode haver interesse em inserir estes caracteres a partir do teclado com um dos comandos de inserção do teclado. Para que isto seja possível, você deve pressionar uma tecla especial no seu teclado, enquanto pressiona uma tecla regular.

As teclas especiais são as teclas SHIFT, CODE e GRAPH. Você já está familiarizado com a tecla SHIFT; para gerar uma letra maiúscula, você deve manter pressionada a tecla SHIFT enquanto pressiona a letra. As teclas CODE e GRAPH operam da mesma maneira que a tecla SHIFT, no sentido em que elas são mantidas pressionadas enquanto você pressiona outra tecla.

Ao entrar com caracteres especiais, você mantém retida uma das teclas especiais. Por exemplo, se você estiver digitando um texto e desejar incluir uma letra "A" maiúscula com trema, basta pressionar as teclas SHIFT e GRAPH e em seguida a tecla A. O Apêndice E mostra as diversas teclas que devem ser pressionadas para os diferentes tipos de caracteres gráficos.

O Comando PRINT

Existem muitas maneiras de apresentar um texto na tela de seu computador MSX, e todas envolvem o comando PRINT, que é o mais flexível do MSX-BASIC e pode ser utilizado em muitos tipos de apresentações de texto na tela. Você já viu a utilização do comando PRINT em muitos dos programas anteriores.

Toda a discussão que se faz a seguir com o comando PRINT também se aplica ao comando LPRINT. Da mesma forma que o comando LLIST imprime a saída do comando LIST em uma impressora, o comando LPRINT imprime a saída do comando PRINT na impressora.

IMPRESSÃO SIMPLES

A forma mais simples do comando PRINT contém apenas uma *série*:

PRINT *série*.

A *série* pode ser tanto uma constante do tipo "Você digitou S", ou uma variável como RESPOSTA$. Neste caso, a *série* é impressa e o MSX-BASIC desloca o cursor para o começo da próxima linha.

```
10 REM Um PRINT simples
20 PRINT "Isto e a linha 1"
30 PRINT "Isto e a linha 2"
40 PRINT
50 PRINT "Isto e a linha 4"
```

O comando PRINT pode ser dado sem argumento se você simplesmente quer descer com o cursor uma linha.

Você pode imprimir mais do que uma variável em uma mesma linha, com o comando PRINT. A maneira mais fácil de fazer isto é separando as variáveis com um (;). Por exemplo, o programa abaixo cerca uma *série* de asteriscos:

```
10 REM PRINT com ;
20 PRINT "Digite 2 caracteres : " ;
30 MM$ = INPUT$(2)
40 PRINT "******"; MM$; "******"
```

Neste caso o ponto e vírgula é utilizado na linha 20 (impedindo que o cursor se desloque para a linha seguinte) e na linha 40 (para colocar as três *séries* na mesma linha). Observe que o MSX-BASIC não coloca espaço entre as *séries*. Caso você queira colocar um espaço de cada lado da *série*, deve colocá-lo assim:

```
40 PRINT "****** "; MM$; "******"
```

É possível imprimir números da mesma maneira:

```
10 REM PRINT com ;
20 PRINT "Digite um numero de dois digitos: " ;
30 I1  = VAL(INPUT$(2))
40 PRINT "******"; I1 ; "******"
```

Observe que o MSX-BASIC coloca espaços em volta do número; esta é uma limitação frustrante do comando PRINT. Tente processar o programa novamente com um número negativo. Como você pode ver, o espaço antes do número é reservado ao sinal negativo. Entretanto, sempre haverá um espaço depois do número.

Se você utilizar vírgulas no lugar dos pontos e vírgulas entre as variáveis, o MSX-BASIC colocará espaços entre eles, de modo que cada variável fique alinhada nas colunas 1 e 14. Porém, isto não é uma regra fixa e depende do tipo de dados utilizados, de modo que se recomenda a utilização do ponto e vírgula em vez da vírgula.

Como foi observado na linha 20 do programa anterior, você pode terminar um comando PRINT com um ponto e vírgula e o MSX-BASIC não irá para a linha seguinte. Assim, as linhas

```
150 PRINT "—→";
160 PRINT "←—"
```

imprimirão

```
—→←—
```

Existem duas funções que podem ser utilizadas apenas como argumentos do comando PRINT; não poderão ser utilizados em nenhum outro local. A primeira é a função SPC, que opera da mesma forma que a função SPACE$, bastando adicionar o número desejado de espaços à linha. Por exemplo,

```
30 PRINT "Aqui "; SPC(10); "e ali"
```

imprime

```
Aqui          e ali
```

A função TAB é muito mais útil: desloca o cursor para uma determinada coluna. A primeira coluna da tela é a coluna 0, a próxima é a 1, e assim por diante. A função TAB facilita a montagem de tabelas quando não se conhece o comprimento de uma *série* ou de um número ao imprimi-lo.

As linhas abaixo imprimem uma lista de nomes com os primeiros alinhados na coluna 13 e as idades na coluna 20.

```
320 PRINT "Ultimo"; TAB(14); "Primeiro"; TAB(21); "Idade"
330 PRINT L1$; TAB(14); F1$; TAB(21); A1
340 PRINT L2$; TAB(14); F2$; TAB(21); A2
350 PRINT L3$; TAB(14); F3$; TAB(21); A3
```

Basta que cada nome fique na coluna certa e todos ficam alinhados se fosse dado um argumento à função TAB que movesse o cursor para frente e para trás, o MSX-BASIC o ignoraria. Assim, caso L2$ fosse maior que 14 caracteres, F2$ seria impresso em cima dele. Para impedir isto podemos utilizar a função LEFT$.

```
320 PRINT "Ultimo"; TAB(14); "Primeiro"; TAB(21); "Idade"
330 PRINT LEFT$(L1$, 12); TAB(14);
    LEFT$(F1$,6); TAB(21); A1
340 PRINT LEFT$(L2$, 12); TAB(14);
    LEFT$(F2$, 6); TAB(21); A2
350 PRINT LEFT$(L3$, 12); TAB(14);
    LEFT$(F3$, 6); TAB(21); A3
```

FORMATAÇÃO AVANÇADA DE IMPRESSÃO

Apesar da impressão com o comando PRINT e as diversas funções da *série* permitirem grande flexibilidade, existe uma outra forma de uso do comando PRINT que o torna ainda mais poderoso. O comando PRINT USING permite que o formato da linha de saída seja especificado com caracteres especiais em uma *série*. A estrutura do comando é

```
PRINT USING série formato; argumento1,argumento2,...
```

*Série*formato é uma constante ou variável *série*, e os argumentos podem ser qualquer constante ou variável. Infelizmente, preparar uma *série* formatada geralmente é difícil. Caso você tenha formatação simples para a linha é melhor utilizar funções e uma declaração PRINT regulares.

Os caracteres que podem ser utilizados em uma *série* formatada são indicados nas Tabelas 10-1 e 10-2. Qualquer caractere em uma *série* formatada que não apareça nesta lista é tratado como um caractere normal e é impresso exatamente como aparece na *série* formatada.

Se tudo isto parecer um pouco confuso, não desista. O primeiro exemplo, a impressão de dígitos, tornará as coisas um pouco mais claras. Você utilizará o sinal de número (#) em uma *série* formatada para indicar o lugar em que deve ficar um dígito. Processe o seguinte programa:

```
10 REM Exemplo de PRINT USING
20 N1 = 1234
30 PRINT USING "ooo####ooo"; N1
```

A saída deste programa será

```
0001234000
```

Os "o"s estão lá para mostrar-lhe que o MSX-BASIC imprime qualquer caractere não formatado na *série* formatada diretamente. Uma vez que você indicou quatro sinais de "número", o MSX-BASIC imprimiu apenas quatro dígitos do número.

Tabela 10.1 Caracteres para *série* de formatação numérica.

Caractere	Utilização
#	Espaço para um dígito
.	Inclui um ponto decimal
+	Indica + ou –; pode ser utilizado antes ou depois do número
-	Indica apenas –; pode ser utilizado após o número
$$	Coloca um $ à esquerda do número
**	Desloca *'s à esquerda do número para toda a largura do campo
**$	Desloca um $ precedido pelos *'s para toda a largura do campo
^ ^ ^ ^	Apresenta o número em notação científica.

Agora, altere ligeiramente o programa:

```
10 REM Exemplo de PRINT USING
20 N1 = 1234
30 PRINT USING "ooo####ooo"; N1
40 PRINT USING "ooo#######ooo"; N1
```

Tabela 10.2 Caracteres para *série* de formatação de texto.

Caractere	Utilização
\\	Espaço para caractere
!	Espaço para 1 caractere
&	Espaçamento variável
_	Próximo caractere é literal
outro	Coloca caracteres na saída

Quando existem sete sinais de "número", apesar de 1234 ser de apenas quatro dígitos, o resultado é

```
ooo1234ooo
ooo    1234ooo
```

Ao ser utilizado o sinal de "número", o MSX-BASIC sempre deixa o número de espaços especificados. Se o número impresso ocupar um número menor de espaços que aqueles especificados, o MSX-BASIC preencherá os espaços à esquerda com vazios.

O preenchimento de espaços à esquerda com vazios é denominado justificação à esquerda. Isto é bastante útil quando existe uma coluna de números que se deseja adicionar visualmente. Compare a saída das duas partes do programa abaixo:

```
 10 REM Somando colunas a moda antiga
 20 X1 = 6391
 30 X2 = 934
 40 X3 = 5179
 50 PRINT X1
 60 PRINT X2
 70 PRINT X3
 80 PRINT "------"
 90 PRINT X1+X2+X3
 92 PRINT : PRINT
100 REM Somando colunas pelo novo metodo
110 PRINT USING "#####"; X1
120 PRINT USING "#####"; X2
130 PRINT USING "#####"; X3
140 PRINT "------"
150 PRINT USING "#####"; X1+X2+X3
```

O resultado é

```
 6391
 934
 5179
 ------
 12504

 6391
  934
 5179
------
12504
```

Observe a dificuldade em adicionar o primeiro conjunto de números mentalmente, uma vez que não estão alinhados na forma usual. O segundo conjunto de números é muito mais fácil de ser somado visualmente.

Se você estiver utilizando números reais, poderá colocar o ponto (.) na *série* de formatação para especificar onde deseja que apareça o ponto decimal. O MSX-BASIC arredondará as casas decimais em excesso ou colocará zeros se houver poucas casas decimais:

```
10 REM PRINT USING com .
20 PRINT USING "###.###"; 123.456
30 PRINT USING "###.###"; 1.23456
40 PRINT USING "###.###"; 1
50 PRINT USING "###.###"; 1.1
```

A saída é

```
123.456
  1.235
  1.000
  1.100
```

Observe que 1.23456 foi corretamente arredondado para 1.235 e não truncado para 1.234. O MSX-BASIC utiliza regras padronizadas de arredondamento.

Em um programa podemos utilizar o sinal de adição antes ou depois do número fazendo o sinal do número aparecer antes dele na tela. Independentemente do tamanho do número, o MSX-BASIC deslocará o sinal para a coluna à esquerda do número. O sinal de subtração também pode ser utilizado para apresentar um "-" após o número caso o mesmo seja negativo.

```
10 REM PRINT USING com + e -
20 PRINT USING "+####"; 721
30 PRINT USING "+#### "; -721
40 PRINT USING "####+"; 93
50 PRINT USING "####+ "; -93
60 PRINT USING "####-"; 128
70 PRINT USING "####-"; -128
```

O resultado é

```
+721
-721
 93+
 93-
128
128-
```

Note que os números impressos sempre ocupam o número de espaços indicados pelos sinais "#" mais um para o "+" ou "-".

A colocação de dois sinais de cifrão à esquerda dos sinais de # faz o MSX-BASIC deslocar um sinal de cifrão para a esquerda da saída. Por exemplo

```
210 PRINT USING "$$####.##"; 77.32
220 PRINT USING "$$####.##"; 8120.31
```

imprimiria

```
   $77.32
$8120.31
```

Dependendo do fabricante do seu MSX, o mesmo poderá ser feito com dois sinais de libra ou iene na *série* de formatação.

Você deve ter visto cheques escritos com asteriscos precedendo os números; isto ajuda na prevenção de falsificações. O comando PRINT USING permite que você coloque antes de um número diversos asteriscos, colocando, para tal, dois asteriscos na *série* de formatação. Por exemplo:

```
210 PRINT USING "**##.##";77.32
```

imprimiria

```
****77.32
```

Se você desejar tanto o sinal de cifrão como o asterisco, utilize o "**$" na *série* de formatação:

```
210 PRINT USING "**$####.##";77.32
```

imprimiria

```
**** $77.32
```

Algumas pessoas gostam de ver as vírgulas nos números impressos. Se você incluir uma vírgula à esquerda do ponto, na *série* de formatação, a vírgula será acrescentada na saída. A vírgula insere um espaço na *série* de saída, porém cada vírgula impressa ocupa o espaço de um dígito. Por exemplo:

```
80 PRINT USING "########,.##";6328900.99
```

imprimirá

```
6,328,900.99
```

Você pode utilizar quatro circunflexos (^^^^) para indicar que o número deve ser impresso em notação científica. Utilizaremos os sinais dos números da mesma forma que antes para indicar os dígitos à esquerda e à direita do decimal. Por exemplo,

```
320 PRINT USING "#.####^^^^";35.7182
```

imprimiria

```
0.3572E+02
```

Também é possível formatar *séries* com o comando PRINT USING. Para especificar quantas posições de uma *série* devem ser impressas, utilize dois sinais "\" e espaços. Por exemplo, para fazer uma saída que deixe cinco espaços, utilize "\,espaço,espaço, espaço,\", como no programa abaixo:

```
10 REM PRINT USING \
20 S1$ = "mnopqrst"
30 S2$ = "xx"
40 PRINT USING "yyy\      \yyy"; S1$
50 PRINT USING "yyy\      \yyy"; S2$
```

O resultado é

```
yyymnopqyyy
yyyxx     yyy
```

Se a *série* for maior do que o número de espaços disponíveis, como em S1$, serão utilizados só os caracteres da esquerda. Se a *série* for muito curta, como em S2$, serão colocados vazios à sua direita.

O ponto de exclamação (!) indica que apenas o primeiro caractere da *série* deverá ser impresso. O programa

```
10 REM PRINT USING !
20 S1$ = "mnopqrst"
30 S2$ = "xx"
40 PRINT USING "yyy!yyy"; S1$
50 PRINT USING "yyy!yyy"; S2$
```

imprime

```
yyymyyy
yyyxyyy
```

O símbolo "&" indica que toda a *série* deverá ser impressa. Entretanto, uma vez que isto impede o alinhamento de outro texto após a *série* na linha, é pouco utilizado.

Você pode estar pensando no que fazer para que um dos caracteres especiais seja impresso. Por exemplo, você poderá desejar cercar um número com pontos. Para tal, utilize o sinal "_" antes do caractere especial, pois isto faz com que ele deixe de ser considerado especial. Por exemplo,

```
40 PRINT USING "_._._. ### _._._."; 590
```

imprimirá

```
...590...
```

UTILIZANDO TODA A TELA

O comando PRINT inicia a impressão onde quer que se encontre o cursor ao ser dado este comando. Se você terminar um comando PRINT com um ponto e vírgula, o cursor permanecerá na posição após a impressão do último caractere. Pode haver o interesse em mover o cursor para qualquer posição da tela. O MSX-BASIC lhe dá a flexibilidade de movimentar o cursor em dois sentidos.

No modo texto, que tem sido utilizado, a tela tem 24 linhas horizontais e 39 colunas verticais. O MSX-BASIC utiliza coordenadas cartesianas para representar a posição da tela (talvez seja bom consultar seus livros de Geometria). Cada posição da tela tem uma coordenada x e y, indicada como

```
(x,y)
```

A coordenada x varia de 0 (à esquerda) a 38 (à direita) e a coordenada y varia de 0 (em cima) até 23 (em baixo), assumindo que você tenha desligado a linha das funções como veremos a seguir). Assim, o canto superior esquerdo é denominado (0,0) e o canto inferior direito (38,23). A Figura 10.1 mostra uma representação das posições da tela.

A primeira coisa que você deve fazer é desligar a linha das teclas de funções no fim da tela e em seguida limpar a tela. O comando KEY OFF desliga a linha das teclas de funções. O comando CLS é útil quando acionado com o comando PRINT, já que ele elimina o texto da tela. Após o comando CLS, o cursor é colocado no canto superior esquerdo da tela, ou seja, em (0,0).

```
10 REM Posicionamento de texto
20 KEY OFF
30 CLS
```

O comando LOCATE desloca o cursor para uma posição específica. Sua estrutura é

```
LOCATE x,y
```

Processe o seguinte programa:

```
10 REM Posicionamento de texto
20 KEY OFF
30 CLS
40 LOCATE 20,12
50 PRINT "*";
60 TIME = 0
70 IF TIME < 100 THEN GOTO 70
```

Assim, será colocado um asterisco no centro da tela e haverá uma pausa de dois segundos. O programa seguinte traça uma linha de asteriscos do canto inferior esquerdo ao centro superior da tela:

```
10 REM Trace uma diagonal
20 KEY OFF
30 CLS
40 FOR I = 23 TO 0 STEP -1
50 LOCATE (23-I), I
60 PRINT "*";
70 NEXT I
```

O MSX-BASIC permite a exclusão dos valores do comando LOCATE, desde que se inclua a vírgula. Se um dos argumentos for excluído o MSX-BASIC assumirá que você deseja que o cursor permaneça no mesmo local em que estava. Por exemplo, o programa abaixo imprime cinco asteriscos em uma linha vertical iniciando no topo da coluna em que se encontra o cursor:

```
160 FOR N = 0 TO 4
170 LOCATE,N
180 PRINT "*"
190 NEXT N
```

Enquanto você estiver escrevendo um programa poderá não saber com certeza onde se encontra o cursor na tela. A função POS lhe dará o valor da coordenada x. O argumento da função POS poderá ser qualquer número, já que o MSX-BASIC ignora o argumento. A variável CSRLIN retorna o valor da coordenada y e não é uma função. (Isto explica porque muitos programadores reclamam sobre a inconsistência do BASIC em geral.)

Por exemplo, as linhas que seguem armazenam a posição atual do cursor nas variáveis X9 e Y9:

```
410 X9 = POS(1)
420 Y9 = CSRLIN
```

Caso você esteja utilizando o comando LPRINT para dar saída a uma impressora, pode utilizar a função LPOS da mesma maneira que a função POS. Não há equivalente ao CSRLIN para impressora.

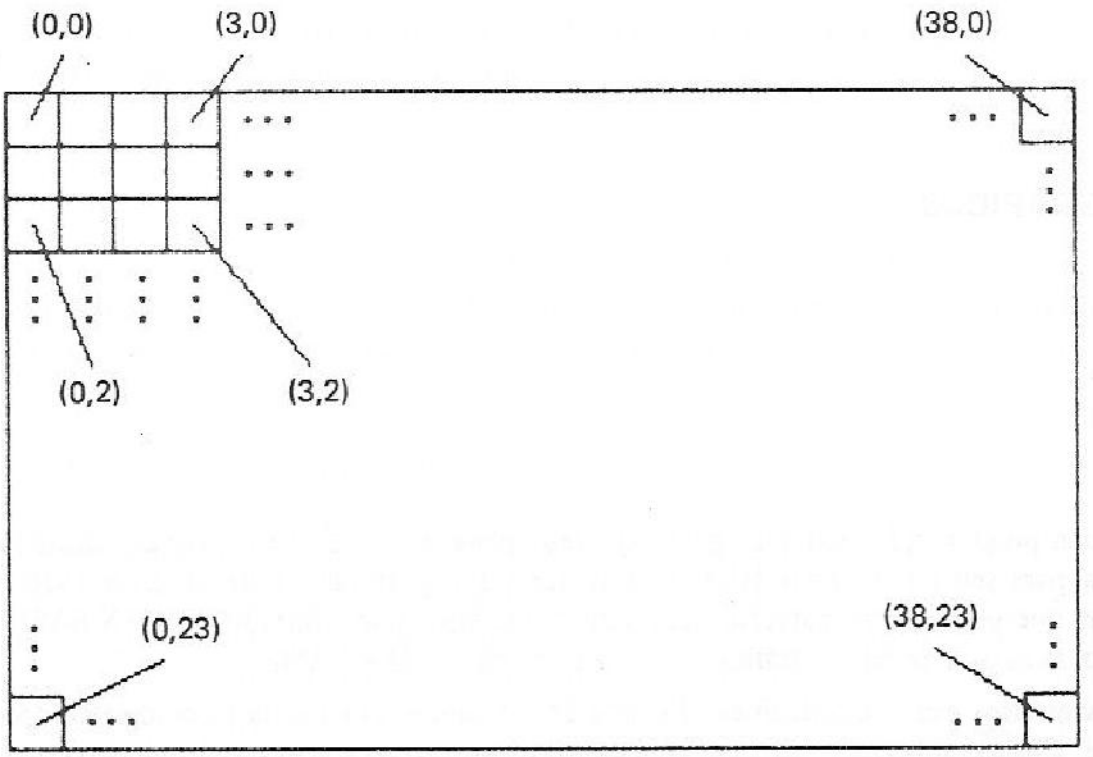


Figura 10.1 Posições na tela.

CAPÍTULO

11

GRÁFICOS

Toda programação feita até agora envolveu apenas texto. Se você tiver adquirido cartuchos de jogos para seu computador MSX, já deve ter tido oportunidade de observar todas as cores e formas que em nada se parecem com aquilo que você tem utilizado no MSX-BASIC. Entretanto, todas as características gráficas estão disponíveis no MSX-BASIC.

Os gráficos em computadores são uma área muito excitante da tecnologia computacional moderna. Observamos gráficos feitos por computadores nas propagandas da televisão e em muitos filmes. Uma imagem gerada por um computador é formada por três elementos básicos: cor, movimento (animação) e forma.

Os gráficos que podem ser criados com o MSX-BASIC não serão tão detalhados ou excitantes como aqueles da televisão, mas fornecerão uma compreensão básica de como os gráficos por computador, com finalidades comerciais, são criados. O MSX-BASIC oferece a você uma grande flexibilidade no desenvolvimento de imagens. Por exemplo, você pode escolher entre 15 cores diferentes, podendo ainda desenhar *sprites*, que são pequenas imagens móveis as quais podem ser utilizadas em jogos.

O seu computador MSX tem quatro modos de tela:

- Modo 0, texto-padrão. É o modo com o qual você operou até agora. Não é possível fazer gráficos nele, mas o texto é mostrado em uma tela de 40 × 24 pontos; com uma cor para os caracteres (denominada cor do primeiro plano) e outra para o fundo.
- Modo 1, texto multicolorido. Como no modo 0, este permite apresentar principalmente texto, mas cada conjunto de oito letras poderá ter suas cores próprias, tanto no primeiro plano como no segundo plano, podendo utilizar-se sprites (mas não gráficos). O texto é apresentado em uma tela de 32 × 40 pontos.
- Modo 2, gráficos de alta resolução. A maioria dos jogos é feita neste modo. Você pode utilizar sprites e desenhar pontos (chamados retículas) em qualquer lugar da tela que tem 256 × 192 pontos. Cada conjunto de oito retículas tem sua própria cor de primeiro e de segundo plano. A maioria dos exemplos deste capítulo será feita neste modo.

– **Modo 3, gráficos de baixa resolução. A tela deste modo apresenta apenas 64 × 48 pontos, o que o toma inútil para muitas aplicações. Porém, cada retícula tem sua própria cor e podem ser utilizados sprites à vontade.**

No programa "PRIMEI" você teve uma noção do que pode ser feito no modo 2. Este capítulo cobrirá todos os aspectos de gráficos disponíveis no MSX-BASIC.

Cor

Você poderá estar cansado das letras brancas em fundo azul vistas até agora. Apesar de o mundo ser "uma aquarela colorida", o MSX-BASIC ainda não demonstrou isto. A parte mais simples na programação de um gráfico é o controle da cor através do comando COLOR. A estrutura do comando COLOR é

```
COLOR primeiroplano,segundoplano,contorno
```

Cada um dos argumentos pode variar de 0 a 15.

A Tabela 11.1 apresenta as cores disponíveis. As cores estão listadas para um aparelho de televisão regulado para uma apresentação normal. Este, normalmente, é mais brilhante do que se deseja para uma imagem computadorizada, de modo que é bastante usual diminuir o brilho e aumentar o contraste quando se utiliza um aparelho de televisão como terminal de computador. Ao fazer isto a matiz e a saturação da cor vão-se alterar bastante. Em outras palavras, pode ser necessário que você construa sua própria tabela de cores.

Tabela 11.1 Cores do MSX-BASIC.

Número	Cor
1	Preto
2	Verde-médio
3	Verde-claro
4	Azul-escuro
5	Azul-médio
6	Vermelho-escuro
7	Azul-claro
8	Vermelho-médio
9	Vermelho-claro
10	Amarelo-escuro
11	Amarelo-claro
12	Verde-escuro
13	Roxo
14	Cinza
15	Branco
0	Transparente (o primeiro plano terá a cor do segundo plano)

No modo 0, o texto que você programa não tem cor de contorno. Neste modo você pode formar 256 pares de cores de primeiro/segundo plano. Muitos destes pares não são interessantes porque as cores são muito semelhantes, tornando o texto ilegível, ou porque as cores contrastam muito. O programa que segue fornece alguns exemplos de pares de cores.

```
 10 REM  Experiencia com pares de cores
 20 CLS
 30 PRINT "Este texto e padrao: (15,4)"
 40 GOSUB 1000
 50 COLOR 10,6
 60 PRINT "Muitas pessoas acham isto agradavel (10,6)"
 70 GOSUB 1000
 80 COLOR 1,15
 90 PRINT "Preto sobre branco (1,15), mas as letras borram"
100 GOSUB 1000
110 COLOR 4,15
120 PRINT "Azul-escuro sobre branco (4,15), uma boa escolha"
130 GOSUB 1000
140 COLOR 12,6
150 PRINT "Pessima combinacao (12,6)"
160 GOSUB 1000
170 COLOR 10,6
180 END
1000 REM Aguarde 3 segundos
1010 TIME = 0
1020 IF TIME < 150 THEN GOTO 1020
1030 RETURN
```

Em seguida temos um programa que percorre todas as combinações de cores. Observe que quando as cores de primeiro e segundo plano forem as mesmas não é possível ver o texto.

```
 10 REM  Selecao de cores
 20 KEY OFF
 30 CLS
 40 FOR N = 1 TO 23
 50 PRINT STRING$(39,"X");
 60 NEXT N
 70 REM B = segundo plano, F = primeiro plano
 80 LOCATE 0,0
 90 PRINT "F = "
100 PRINT "B = "
110 REM Percorra todos F's para todo B
120 FOR B = 1 TO 15
130 LOCATE 2,1
140 PRINT USING "##"; B
```

```
150 FOR F = 1 TO 15
160 LOCATE 2,0
170 PRINT USING "##"; F
180 COLOR F, B
190 TIME = 0
200 IF TIME < 75 THEN GOTO 200
210 NEXT F
220 NEXT B
230 REM  Tudo completado
240 CLS
250 COLOR 15,4
```

GRÁFICOS NO MODO 2

Uma vez vistas as cores que o MSX-BASIC coloca a sua disposição, devemos iniciar a utilização de gráficos no modo 2. O primeiro comando necessário é o comando SCREEN, que muda de um para outro modo de tela. Sua estrutura é

```
SCREEN modo
```

Neste caso "modo" é o número do modo desejado. Assim fica explicada a linha 80 do programa "PRIMEI"; ela coloca a tela no modo 2.

No modo 2 o comando COLOR seleciona os valores das cores do primeiro e do segundo plano. É possível a existência de várias cores no primeiro plano da tela ao mesmo tempo, como foi visto nos retângulos coloridos do programa "PRIMEI".

Muitos dos comandos que desenham na tela, em modo 2, podem determinar a cor do primeiro plano, enquanto durar a execução do comando, sendo que isto não a transforma na cor fundamental do primeiro plano. Caso você não especifique uma cor, a cor atual do primeiro plano é a utilizada.

Após a utilização do comando COLOR para estabelecer a cor do segundo plano no modo 2, deve ser utilizado o comando CLS para aparecer cor na tela. Por este motivo é que, nos programas, encontramos linhas do tipo

```
210 COLOR ,1: CLS
```

No modo 2 não há equivalentes para a função POS e a variável CSRLIN: cuidado, não se esqueça onde está o cursor, não existe nenhum comando ou função que o auxilie. O MSX-BASIC fornece a função POINT que informa a cor de uma determinada retícula. A estrutura da função POINT é

```
POINT (x,y)
```

Como já é de seu conhecimento, seu valor varia de 0 a 15, dependendo da cor no ponto (x,y).

Ao programar, lembre-se de que cada conjunto de oito retículas horizontais pode ter apenas uma cor de primeiro plano. Infelizmente, esta limitação restringe um pouco sua liberdade já que a alteração da cor do primeiro plano somente poderá ser feita a cada oito retículas. Entretanto, com a utilização cuidadosa das cores de primeiro e segundo plano é possível a criação de imagens vivas e multicoloridas.

Considerando que o modo 3 tem uma resolução muito baixa, este capítulo enfoca apenas os gráficos em modo 2. Porém, a maioria dos comandos deste capítulo opera da mesma maneira no modo 3.

TRAÇANDO PONTOS

Lembre-se de que o modo 2 tem uma tela de 256 × 192 retículas. Elas são definidas por coordenadas cartesianas iniciando com (0,0) no canto superior esquerdo. A Figura 11.1 mostra como se organiza a tela.

É importante observar que muitos aparelhos de televisão utilizados como terminais de computador não mostram todos as retículas da margem esquerda ou direita, dependendo da regulagem do aparelho de televisão. Muitos aparelhos não apresentam as retículas das colunas 0 à 8.

Em vez de utilizar o comando LOCATE do MSX-BASIC, para movimentar o cursor na tela de texto, será utilizado o comando PSET para a movimentação do cursor gráfico no modo 2.

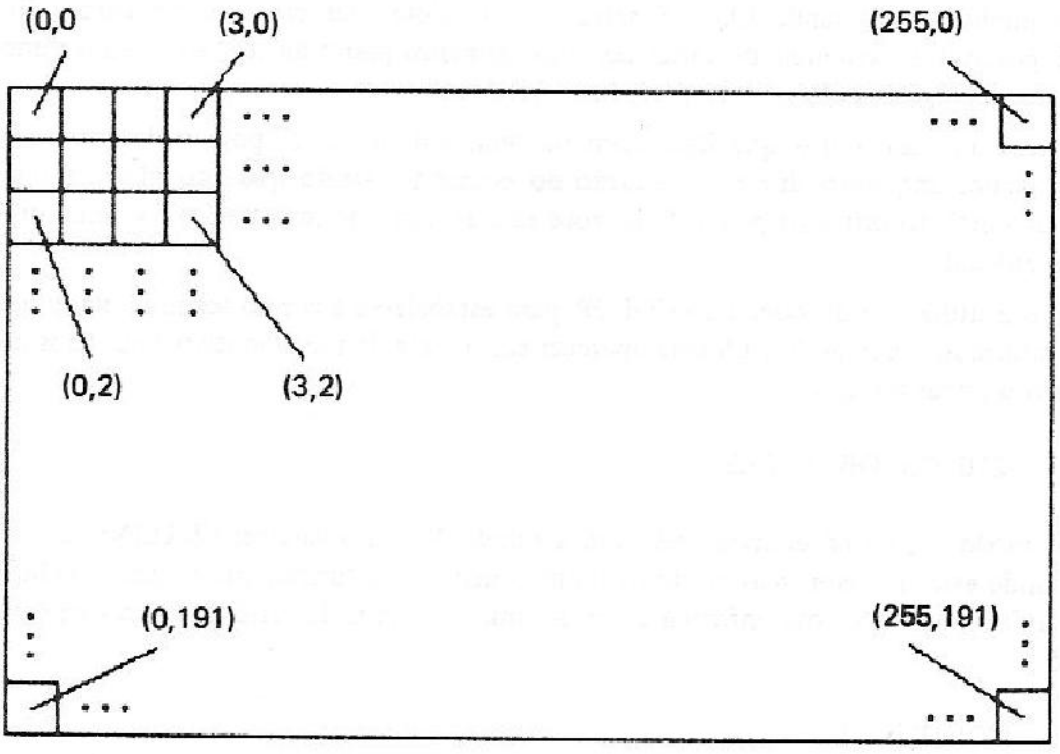


Figura 11.1 Posições na tela gráfica.

Observe, porém, que todos os comandos que fazem desenhos na tela em modo 2 geralmente alteram a posição do cursor gráfico. O comando PSET além de atualizar a posição do cursor, traça uma retícula nesta posição. A estrutura do comando PSET é

```
PSET STEP(x,y), cor
```

Os argumentos STPE e cor são opcionais.

O exemplo que segue define o vermelho-escuro (cor 6) como sendo a cor de segundo plano e traça um quadrado azul com suas arestas em (100,100), (100,150), (150,150) e (150,100).

```
10 REM utilizando PSET
20 SCREEN 2
30 COLOR ,6: CLS
40 FOR Y = 100 TO 150
50 PSET(100,Y), 5
60 NEXT Y
70 FOR X = 100 TO 150
80 PSET(X, 150), 5
90 NEXT X
100 FOR Y = 150 TO 100 STEP -1
110 PSET (150,Y), 5
120 NEXT Y
130 FOR X = 150 TO 100 STEP -1
140 PSET(X,100), 5
150 NEXT X
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

O argumento STEP do comando PSET (assim como os de outros comandos que serão vistos) facilita muito o ato de desenhar. Por enquanto foi especificado o local exato de cada retícula. Com o argumento STEP os valores de x e y não indicam mais a posição da retícula, mas a relativa à posição do cursor gráfico. Por exemplo, a colocação de uma retícula a 30 retículas à direita e 15 retículas abaixo do cursor gráfico, independentemente de sua posição atual, exige o comando

```
PSET STEP(30,15)
```

Para colocar uma retícula à esquerda e duas acima do cursor gráfico, você deve dar o comando

```
PSET STEP (-1, -2)
```

Observe como isto facilita o traçado do nosso quadrado:

```
10 REM utilizando PSET e STEP
20 SCREEN 2
30 COLOR ,6: CLS
```

```
 35 PSET(100,100), 5
 40 FOR I = 1 TO 50
 50 PSET STEP(0,1), 5
 60 NEXT I
 70 FOR I = 1 TO 50
 80 PSET STEP(1,0), 5
 90 NEXT I
100 FOR I = 1 TO 50
110 PSET STEP(0, -1), 5
120 NEXT I
130 FOR I = 1 TO 50
140 PSET STEP(-1,0), 5
150 NEXT I
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Basta acrescentar a linha 35 para fornecer um ponto de partida e em seguida utilizar o loop FOR-NEXT para passar de um ponto ao outro.

A vantagem da utilização do argumento STEP é a de que podemos movimentar toda a figura apenas alterando a posição da primeira retícula. Se você decidir traçar o quadrado iniciando em (37,91) basta alterar a linha 35 para

```
35 PSET(37,91), 5
```

Lembre-se de que é possível mudar a cor atual do primeiro plano com o comando COLOR. O programa pode sofrer nova simplificação colocando-se a cor do desenho na linha 30, sem mencioná-la em outro lugar:

```
 10 REM  Utilizando PSET e STEP
 20 SCREEN 2
 30 COLOR 5,6: CLS
 35 PSET(100,100)
 40 FOR I = 1 TO 50
 50 PSET STEP (0,1)
 60 NEXT I
 70 FOR I = 1 TO 50
 80 PSET STEP (1,0)
 90 NEXT I
100 FOR I = 1 TO 50
110 PSET STEP (0, -1)
120 NEXT I
130 FOR I = 1  TO  50
140 PSET STEP (-1,0)
150 NEXT I
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

O comando PRESET é similar ao comando PSET, e sua estrutura é a mesma:

```
PRESET STEP(x,y), cor
```

Os argumentos STEP e cor permanecem opcionais. A diferença entre os dois comandos é que, se não for fornecido o argumento cor, o PRESET fará o traçado na cor do segundo plano em vez de fazê-lo na cor do primeiro plano. Isto, em essência, equivale a apagar aquilo que existe no ponto.

Se você utilizar a opção da cor, o comando PRESET agirá da mesma forma que o comando PSET. Podemos observar que o comando PRESET é similar ao comando PSET com a opção da cor, igual à cor do segundo plano. O comando PRESET apenas tem utilidade quando não se conhece a cor do segundo plano.

TRAÇANDO LINHAS E CAIXAS

O traçado de linhas é um dos procedimentos mais comuns quando você escreve programas gráficos. Pode-se tornar monótona a utilização do loop FOR-NEXT cada vez que se deseja traçar uma linha. Adicionalmente, as linhas traçadas até agora são apenas paralelas à margem da tela. Imagine como seria o loop FOR-NEXT para traçar uma linha entre (100,100) e (141,107). A matemática se tornaria bem complicada.

Para facilitar este traçado o MSX-BASIC possui o comando LINE. A estrutura do comando LINE é complexa, mas pode ser explicada passo a passo. De maneira similar ao comando PSET, a estrutura é

```
LINE (x1,y1) - (x2,y2), cor
```

Novamente o argumento cor é opcional. Como você percebe este comando traçará uma linha de (x1,y1) até (x2,y2). Ao terminar a execução do comando, o cursor gráfico estará em (x2,y2).

O comando LINE simplifica o programa do quadrado:

```
10 REM Utilizando LINE
20 SCREEN 2
30 COLOR 5,6: CLS
40 LINE(100,100) - (100,150)
50 LINE(100,150) - (150,150)
60 LINE(150,150) - (150,100)
70 LINE(150,100) - (100,100)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Como se vê, este programa é bem menor do que o original.

Entretanto, como é comum traçar muitas linhas unidas, o MSX-BASIC fornece uma versão mais compacta do comando LINE

```
LINE - (x2,y2), cor
```

Este comando inicia a linha no local do cursor gráfico. Seu programa pode ser simplificado para

```
10 REM Utilizando LINE
20 SCREEN 2
30 COLOR 5,6: CLS
40 LINE(100,100) - (100,150)
50 LINE - (150,150)
60 LINE - (150,100)
70 LINE - (100,100)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Isto, sem dúvida, é mais simples do que lembrar onde se encontra o cursor gráfico.

Com o argumento STEP a estrutura fica ainda mais elegante:

```
LINE - STEP(x2,y2), cor
```

Assim como o argumento STEP do comando PSET, este também torna o seu programa mais generalizado. O programa traçado do quadrado ficaria assim:

```
10 REM Utilizando LINE e STEP
20 SCREEN 2
30 COLOR 5,6: CLS
35 PSET(100,100)
40 LINE - STEP(0,50)
50 LINE - STEP(50,0)
60 LINE - STEP(0,-50)
70 LINE - STEP(-50,0)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Agora, para mudar a posição do quadrado, basta alterar a linha 35.

Uma forma menos utilizada do comando LINE é

```
LINE STEP(x1,x2) - STEP(x2,y2), cor
```

Além do traçado de uma linha, outra atividade comum é o traçado de quadrados (que pode estar cansando-o!). Se traçarmos um quadrado que seja paralelo aos lados da tela (como até agora), isto pode ser ainda mais fácil com o argumento B no comando LINE. O argumento B

aparece depois de cor no comando LINE e faz o MSX-BASIC traçar um quadrado cujos vértices opostos sejam (x1,y1) e (x2,y2).

O argumento B reduz o nosso programa para

```
 10 REM Utilizando LINE e B
 20 SCREEN 2
 30 COLOR 5,6: CLS
 40 LINE(100,100) - (150,150), ,B
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Não havendo especificação da cor, devem ser incluídas as vírgulas de modo que o MSX-BASIC não tente colorir a linha com o valor de uma variável denominada "B". O argumento B pode ser utilizado também com o argumento STEP:

```
 10 REM Utilizando LINE e B e STEP
 20 SCREEN 2
 30 COLOR 5,6: CLS
 35 PSET(100,100)
 40 LINE - STEP(50,50), ,B
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

A linha 40 é um sumário daquilo que você tem feito até agora: traçar um quadrado de 50 × 50.

Como foi mencionado anteriormente, o comando LINE pode ser utilizado para traçar linhas entre dois pontos quaisquer, e não apenas retas paralelas às beiradas da tela. Para traçar um paralelogramo basta alterar o programa para

```
 10 REM paralelogramo
 20 SCREEN 2
 30 COLOR 5,6: CLS
 35 PSET(100,100)
 40 LINE - STEP(0,50)
 50 LINE - STEP(20,30)
 60 LINE - STEP(0, -50)
 70 LINE - STEP(-20, -30)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

É claro que podemos traçar qualquer forma desejada. O programa que segue traça 100 linhas aleatórias pela tela:

```
 10 REM Tracador de linha randemica
 20 SCREEN 2
 30 COLOR 15,1: CLS
```

```
 40 A = RND(-TIME)
 50 PSET(0,0)
 60 FOR I = 1 TO 100
 70 LINE - ((RND(1)*255), (RND(1)*191))
 80 NEXT I
 90 TIME = 0
100 IF TIME < 250 THEN GOTO 100
```

DESENHANDO CÍRCULOS E CURVAS

É interessante traçar pontos e linhas com o MSX-BASIC, mas o traçado deve refletir melhor as imagens que você quer criar. A capacidade de traçar curvas fará suas imagens ficarem mais reais. Para tanto, o MSX-BASIC fornece o comando CIRCLE.

Apesar do nome, este comando permite que se tracem diversas formas: circunferências, elipses, partes de circunferências (denominadas arcos) e arcos elípticos. Assim como o comando LINE, a estrutura do comando CIRCLE é complexa. A forma mais simples do comando CIRCLE é aquela que traça uma circunferência:

```
CIRCLE STEP(x,y),raio,cor
```

Como sempre, os argumentos STEP e cor são opcionais, sendo que o MSX-BASIC traça um círculo com o centro no ponto (x,y) e o raio especificado. Caso o argumento STEP seja utilizado, o centro será deslocado relativamente à posição atual do cursor gráfico. Após a execução do comando CIRCLE, o cursor gráfico é posicionado no centro.

O programa abaixo traça 25 círculos de raio 10, na tela:

```
 10 REM 25 Circulos
 20 SCREEN 2
 30 COLOR 1,6: CLS
 40 PRESET(68,46)
 50 FOR A = 1 TO 5
 60 FOR B = 1 TO 5
 70 CIRCLE STEP(20,0), 10
 80 NEXT B
 90 PRESET STEP(-100,20)
100 NEXT A
110 TIME = 0
120 IF TIME < 500 THEN GOTO 120
```

Algumas coisas devem ser observadas a respeito deste programa:

- Os círculos são traçados com muita rapidez. O MSX-BASIC utiliza uma fórmula especial que torna muito eficiente o traçado dos círculos nos computadores MSX;

- Os círculos não são exatamente redondos: as bordas são grosseiras. Isto se deve ao fato de as retículas da tela terem um tamanho finito. O traçado de curvas com retículas sempre provocará as irregularidades vistas na circunferência;
- Quando as circunferências se tocam, as irregularidades ficam mais aparentes. Se estiver utilizando um aparelho de televisão, observe que os pontos em que as circunferências se encontram não são iguais; à medida que se deslocam pela tela, as junções parecem diferentes. Isto se deve ao fato de o aparelho de televisão não apresentar da mesma forma diferentes pontos coloridos. Procure alterar as cores da linha 30 e observe como estes pontos de encontro se alteram;
- O MSX-BASIC não traça o ponto central porque o programa utiliza PRESET em lugar de PSET. O comando PRESET é utilizado apenas para movimentar o cursor gráfico, e não para fazer traçados.

Pela adição de dois argumentos após cor, o comando CIRCLE permite o traçado de arcos:

```
CIRCLE STEP(x,y),raio,cor,angulinic,angulfin
```

Angulinic e angulfin são medidos em radianos. O que segue é uma lição de trigonometria, caso você nunca tenha aprendido (ou tenha esquecido) como medir ângulos em radianos.

Provavelmente você se lembra de que a circunferência está dividida em 360 graus. Como contagem, o grau é o ponto do círculo logo à direita do centro. Assim, 90 graus está para cima, 180 graus está à esquerda e 270 está para baixo. Existem 2*pi radianos em um círculo (pi ←= 3.14159).

Isto significa que 360 graus equivalem a 6.28318 radianos, ou:

```
1 radiano =57.2958 graus
1 grau    =0.01745 radianos
```

A Figura 11.2 mostra um círculo com ambos, radianos e graus.

Para traçar um arco, você deve conhecer o começo e o fim do arco em radianos. Já que a maioria das pessoas pensa em termos de graus e não de radianos, é prático ter no programa um fator de conversão. Por exemplo, para traçar um arco a partir de 45 (1:30 no relógio) até 135 graus (10:30), com o centro em (80,60) e com um raio de 25, seria utilizado o seguinte comando:

```
RC = 0.01745
CIRCLE(80,60), 25, ,45*RC,135*RC
```

Isto é indicado na Figura 11.3.

A ordem dos ângulos de início e fim é importante. A Figura 11.4 mostra o resultado caso tenham sido invertidos os ângulos inicial e o final:

```
CIRCLE(80,60), 25, ,135*RC,45*RC
```

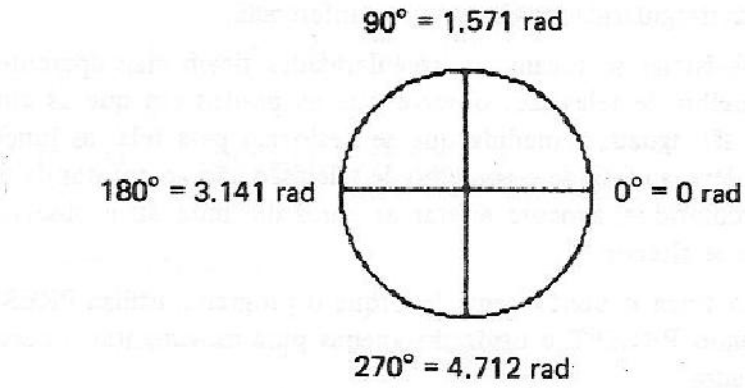


Figura 11.2 Medindo ângulos em radianos.

A elipse é um círculo alongado em uma direção. Imagine um quadrado que envolva uma circunferência e toque esta circunferência em cada lado. Se o quadrado fosse esticado em forma de um retângulo, a circunferência se transformaria em uma elipse. O comando CIRCLE permite que você desenhe uma elipse, especificando a relação dos lados do retângulo chamada relação de aspecto.

A estrutura para esta forma de comando do CIRCLE é

```
CIRCLE STEP(x,y),raio,cor,anginic,angfin,aspecto
```

Se forem incluídos os dois ângulos o MSX-BASIC fará um arco elíptico. No comando CIRCLE a relação de aspecto é a relação entre altura e largura. Se a relação de aspecto for maior do que 1 a elipse será vertical e estreita; se menor do que 1, a elipse será horizontal e larga.

Figura 11.3 Arco de 45° a 135°.

O raio será sempre a medida mais longa da elipse, significando que a curva traçada com o comando CIRCLE sempre caberá dentro do círculo que seria traçado se o argumento "aspecto" fosse excluído (veja Figura 11.5).

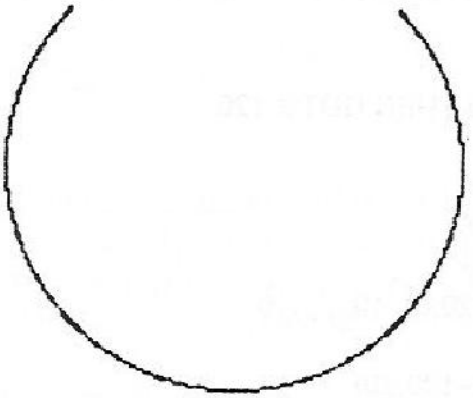


Figura 11.4 Arco de 135° a 45°.

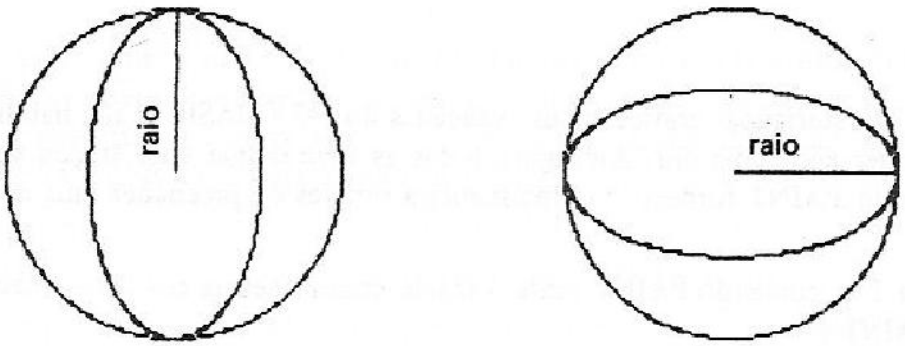


Figura 11.5 Relações de aspectos.

O programa que segue é uma modificação do programa anterior que traçou 25 circunferências Este programa desenha 25 elipses verticais e 25 horizontais.

```
10 REM  25 Elipses verticais e depois horizontais
20 SCREEN 2
30 COLOR 1,6: CLS
40 PRESET(68,46)
50 FOR A = 1 TO 5
60 FOR B = 1 TO 5
```

```
 70 CIRCLE STEP(20,0), 10, , , , 2
 80 NEXT B
 90 PRESET STEP(-100,20)
100 NEXT A
110 TIME = 0
120 IF TIME < 500 THEN GOTO 120
130 CLS
140 PRESET(68,46)
150 FOR A = 1 TO 5
160 FOR B = 1 TO 5
170 CIRCLE STEP(20,0), 10, , , , .5
180 NEXT B
190 PRESET STEP(-100,20)
200 NEXT A
210 TIME = 0
220 IF TIME < 500 THEN GOTO 220
```

Observe que as três vírgulas entre o raio e a razão de aspecto contêm os locais da cor e dos dois ângulos.

PINTANDO

Uma das características gráficas mais avançadas do MSX-BASIC é sua habilidade em pintar uma área qualquer com uma cor. Até agora todas as figuras que você traçou foram feitas com linhas; o comando PAINT fornece-lhe uma maneira simples de preencher uma área fechada com uma cor.

No modo 2 o comando PAINT pinta a região com a mesma cor do contorno. A estrutura do comando PAINT é

```
PAINT STEP(x,y),cor
```

Os argumentos STEP e cor são opcionais. O ponto (x,y) é onde deve ser iniciada a pintura. No comando PAINT "cor" é a mesma do contorno sendo somente necessária se não for a mesma cor que a do primeiro plano.

O programa abaixo traça um círculo de centro em (100,100) com raio 60 preenchendo-o no final.

```
10 REM Pintando
20 SCREEN 2
30 COLOR 15, 6: CLS
40 CIRCLE(100,100), 60
50 PAINT STEP(0,0)
60 TIME = 0
70 IF TIME < 250 THEN GOTO 70
```

O comando PAINT da linha 50 se aproveita do fato de o comando CIRCLE deixar o cursor gráfico no centro da circunferência. O comando PAINT pode ser iniciado em qualquer parte dentro da região a ser pintada. Por exemplo, a linha 50 poderia ser alterada para

```
50 PAINT(120,80)
```

e o resultado seria o mesmo.

A única exigência com respeito ao comando PAINT é a de que a região que estiver sendo pintada deve ser completamente fechada. Isto é, não deve existir abertura alguma nem mesmo uma única retícula — ou o comando PAINT derramará pintura por toda tela. Por exemplo, altere o programa anterior de modo a acrescentar uma falha no contorno e observe o que ocorre:

```
10 REM Pintando com uma falha
20 SCREEN 2
30 COLOR 15, 6: CLS
40 CIRCLE(100,100),60,0,1.5*3.14159
45 PRESET(100,160)
50 PRINT(100,100)
60 TIME = 0
70 IF TIME < 250 THEN GOTO 70
```

O MSX-BASIC sempre encontra a falha, não importando o seu tamanho. Entretanto, a falha é definida como um espaço de retículas com a mesma cor do segundo plano. Mesmo que as retículas sejam quadradas, a diagonal entre duas delas não é considerada uma falha. A Figura 11.6 mostra o que o MSX-BASIC considera ou não uma falha.

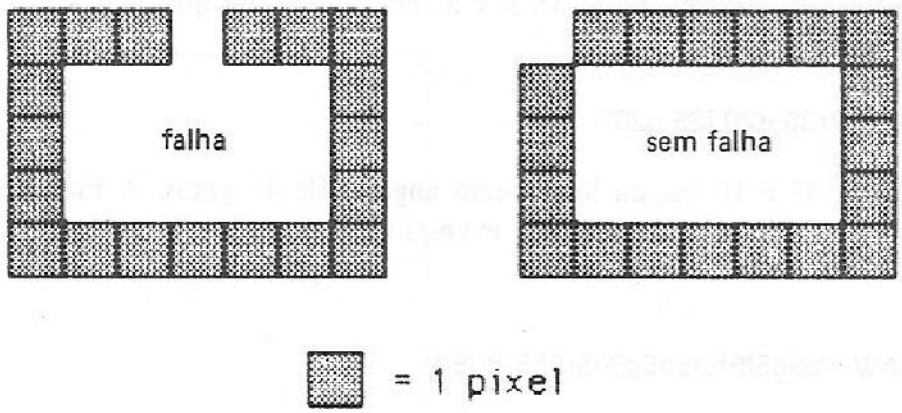


Figura 11.6 Pintura com e sem falhas.

Se você estiver traçando retângulos e preenchendo-os com cores, não será preciso utilizar o comando PAINT, pois existe um argumento do comando LINE que preenche os retângulos. Da mesma forma que o argumento B cria um quadrado, o argumento BF desenha um quadrado cheio. Assim, o comando:

```
290 LINE-STEP(20,50), 13, BF
```

traça um retângulo cheio, de 20 × 50 pontos na cor roxo, iniciando-o na posição atual do cursor gráfico.

FAZENDO DESENHOS

O comando DRAW do MSX-BASIC torna o traçado de desenhos mais fácil do que com comandos múltiplos PSET e LINE. Em vez de utilizar os comandos do MSX-BASIC, o comando DRAW tem sua linguagem própria, denominada "Graphics Macro Language" ou GML. Os comandos GML são colocados em uma *série* em vez de linhas separadas, e o comando DRAW traça a imagem.

O comando DRAW é muito simples:

```
DRAW série
```

Existem muitos comandos GML que podem ser colocados em uma *série*. Existe ainda a possibilidade de se misturar os comandos DRAW com outros comandos gráficos do MSX-BASIC. Na Tabela 11.2 estão resumidos os comandos GML que são discutidos em detalhes neste capítulo.

Um ponto e vírgula pode separar cada comando dentro de uma *série* apesar disso não ser necessário na maioria dos casos. A utilização do ponto e vírgula é sempre uma garantia. Os comandos podem ser digitados tanto em maiúsculas como em minúsculas (nos exemplos são utilizadas letras minúsculas) e, se quiser, você pode utilizar um espaço entre eles. Após cada comando o cursor gráfico é reposicionado.

Os comandos mais simples são U, D, L e R. Para traçar um quadrado de 35 × 20 retículas, seria dado o comando

```
DRAW "r35 d20 l35 u20"
```

Os comandos E, F, G e H traçam linhas com ângulos de 45 graus. A Figura 11.7 mostra os ângulos que estes comandos utilizam. Por exemplo, o comando abaixo traçará um catavento:

```
DRAW "e5d5l5f5l5u5g5u5r5h5r5d5"
```

O comando M do GML é muito parecido ao comando PSET do MSX-BASIC. Pela colocação de duas coordenadas, após o M, indicamos um movimento absoluto:

```
DRAW "m100,35; u10"
```

Tabela 11.2 Sumário de comandos GML.

Comando	Significado
Un	Suba n retículas
Dn	Desça n retículas
Ln	Vá para esquerda n retículas
Rn	Vá para direita n retículas
En	Suba e vá para direita n retículas
Fn	Desça e vá para direita n retículas
Gn	Desça e vá para esquerda n retículas
Hn	Suba e vá para esquerda n retículas
Bcomando	Faça o próximo movimento sem desenhar
Ncomando	Volte para a origem após o próximo comando
Mx,y	Vá até (x,y)
An	Gire n*90 graus
Sn	Faça a escala n/4
Cn	Mude a cor para n
X*série*	Execute macro em *série*

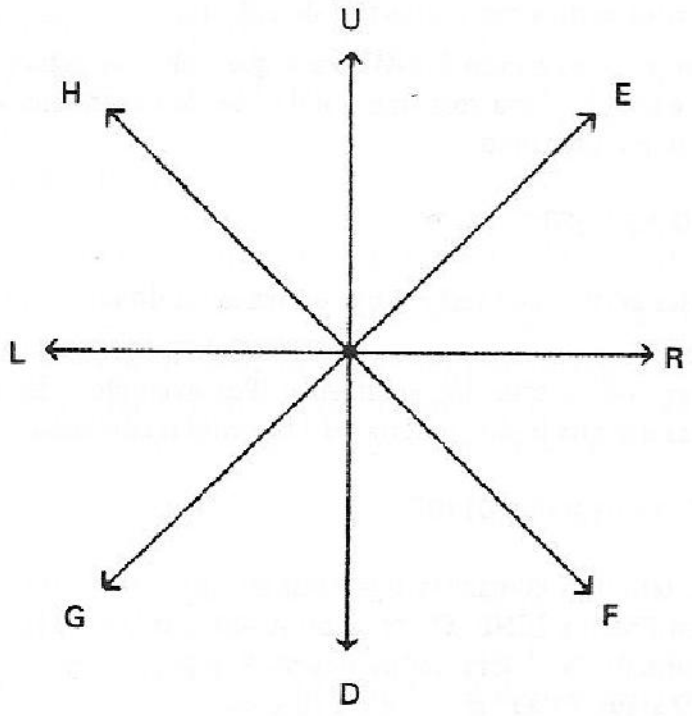


Figura 11.7 Movimentos do comando DRAW.

Pela colocação de um "+" ou de um "-" antes da coordenada x indicamos movimentos relativos. Para mover 3 pontos para a esquerda e 10 para baixo, dê o comando

```
DRAW "2m-3,10"
```

Para movimentar 50 pontos para a direita e 20 para cima, dê o comando

```
DRAW "m+50,-20"
```

Uma vez que o argumento ao comando DRAW é uma *série*, é fácil colocar esta *série* em uma variável e utilizar a variável no comando DRAW. Já que as variáveis *séries* podem ser concatenadas, muitos traçados podem ser feitos com facilidade. Por exemplo:

```
20 PW$ = "e5d5l5f5l5u5g5u5r5h5r5d5"
...
250 DRAW PW$ + "m-20,-20"+PW$
```

Os comandos B e N, em verdade, são prefixos para outros comandos GML. O prefixo B informa ao comando DRAW para que não faça traçados enquanto estiver executando o comando seguinte. Por exemplo, altere a linha 250 do exemplo anterior para:

```
250 DRAW PW$ + "bm-20,-20"+PW$
```

Isto impede que a linha entre os dois cataventos seja desenhada.

O prefixo N informa ao comando DRAW para que volte ao começo do desenho após o próximo comando. Por exemplo, uma maneira simples de desenhar uma seta cuja extremidade está no cursor gráfico, é com o comando

```
DRAW "nl20nh20ng20"
```

Isto também deixa o cursor gráfico no local em que estava antes do início do comando.

O comando de cor, C, permite a colocação da cor do primeiro plano. É como utilizar o comando COLOR apenas com o primeiro argumento. Por exemplo, colocar a cor do primeiro plano em preto e desenhar um quadrado, poderia ser feito com o comando

```
280 DRAW "c1 u10r10d10l10"
```

A linguagem GML tem dois comandos interessantes que torna o traçado de desenhos muito mais flexível do que com PSET e LINE. O comando A altera o ângulo em que o desenho é feito. "a0" é a orientação normal, "a1" gira todos desenhos seguintes em 90 graus no sentido do relógio, "a2" gira em 180 graus, e "a3" gira-os em 270 graus.

Para sentir a utilidade do comando A, assuma que gastou muito tempo em um desenho GML de um carro cujo chassi esteja paralelo à parte inferior da tela. Mais adiante em seu programa, decidiu que quer desenhá-lo de lado. Em vez de reescrever todo GML do carro, basta adicionar o comando "a1" ao início:

```
100 CAR$ = "...(algumas séries complexas)..."
...
500 REM Desenhe o carro normalmente
510 DRAW CAR$
...
800 REM Desenhe o carro de lado
810 DRAW "a1"+CAR$
```

O comando S também dá grande flexibilidade. Permite alterar a escala de um desenho. Após um comando S, o desenho foi reduzido para uma escala de um quarto do argumento do comando S. Assim, "S8" indica ao comando DRAW para aumentar o desenho em duas vezes; "S2" indica que o desenho deve ser diminuído pela metade.

Continuando com o exemplo do carro, imagine que CAR$ desenhou um carro com o tamanho da tela. Mais adiante no programa há necessidade de uma imagem do carro no canto inferior direito da tela, com um quarto do tamanho. Imagine que todos os comandos M na *série* CAR$ utilizaram movimentos relativos, basta, então dar o comando:

```
1270 DRAW "m128,86; s1; "+CAR$
```

O comando X não tem muita serventia, uma vez que apenas permite a utilização de outra *série* no comando DRAW. A facilidade de concatenar a *série* é a mesma mostrada nos exemplos anteriores.

Em qualquer comando GML que aceita valores, podemos dar nome de variáveis substituindo o valor com um sinal de igual, seguido pelo nome da variável e um ponto e vírgula. Por exemplo, se desejasse que uma *série* genérica fizesse um quadrado, poderia utilizar:

```
"r = X0; d = Y0; 1 = X0; u = X0; "
```

Ao ver esta *série*, DRAW substitui as variáveis do MSX-BASIC X0 e Y0 na *série*.

Como podemos observar, o comando DRAW fornece uma grande flexibilidade aos seus comandos de desenho. A melhor maneira de aproveitar esta vantagem é pela colocação de todos os comandos em variáveis e utilização apenas de movimentos relativos. Você verá que o seu tempo de programação com o comando DRAW será bem menor do que aquele com os comandos PSET e LINE.

Apresentando Texto em Modo 2

Você deve ter observado que não foi feita nenhuma menção de como colocar texto na tela. Infelizmente, o MSX-BASIC não tem uma maneira elegante de colocar texto na tela. O método descrito aqui se apóia em comandos que nada têm a ver com gráficos. Entretanto, como é bem provável que você queira texto na tela ao programar com gráficos, verá que a informação seguinte será útil.

Para imprimir texto na tela de gráfico, você terá de aprender um pouco sobre os dispositivos do MSX-BASIC. "Dispositivo" é a palavra para um item de hardware que pode ser acessado pelo MSX-BASIC. As principais características dos dispositivos são:

- Um dispositivo é inicializado com o comando OPEN. Este comando informa ao MSX-BASIC que você quer acessar um dispositivo. Após utilizar o dispositivo, pode-se dar o comando CLOSE, informando ao MSX-BASIC que não se quer mais acessá-lo.
- Alguns dispositivos podem receber informação, outros enviam informação, e alguns fazem ambos. Se um dispositivo pode receber informação, você abre para saída (significando que o MSX-BASIC enviará dados para o dispositivo); caso ele apenas envie informação, você o abre para inserção de dados; e se ele pode fazer ambos, será aberto para acesso randômico.
- Muitos comandos do MSX-BASIC podem enviar informação para (ou receber informação de) um dispositivo. O modo mais primitivo de enviar informação para um dispositivo é utilizando o comando PRINT, e o mais primitivo de receber informação é utilizando os comandos INPUT ou LINE INPUT.
- Todo dispositivo tem uma denominação. Por exemplo, a tela de gráficos é denominada "GRP: ", e o cassete é denominado "CAS: ".
- Ao inicializar um dispositivo, você lhe fornece um número, que é utilizado em todos os comandos do MSX-BASIC que se comunicam com o dispositivo.

Isto poderá parecer mais complicado que os comandos normais MSX-BASIC, mas ficará mais claro com os exemplos posteriores.

O dispositivo que nos interessa no momento é a tela de gráficos. Se você deseja escrever nesta tela, o MSX-BASIC saberá que qualquer texto enviado ao dispositivo "GRP:" deverá ser visto na tela em modo 2.

Você pode abrir a tela "GRP:" somente para saída, uma vez que serão enviadas apenas informações para ela. A estrutura do comando OPEN é:

```
OPEN nomedes FOR acesso AS # numdes
```

Neste caso "nomedes" é "GRP:", e acesso é "OUTPUT". O "numdes" é o número designado ao dispositivo que será utilizado nos comandos MSX-BASIC. Cada dispositivo deve ter um número diferente, sendo que o 0 é reservado ao MSX-BASIC. Já que o comando OPEN ainda não foi utilizado por você, utilize 1 como "numdes". Assim, o seu comando OPEN fica:

```
OPEN "GRP:" FOR OUTPUT AS#1
```

O único comando que você precisa conhecer para imprimir informação na tela de gráfico em modo 2 é o PRINT. A única diferença entre este PRINT e qualquer um dos outros que você aprendeu no Capítulo 10 é que foi incluído o número do dispositivo seguido por uma vírgula, logo após a palavra PRINT. Por exemplo, se você utilizou a linha abaixo em modo 0 para apresentar texto:

```
260 PRINT "Pontos:"
```

em modo gráfico 2, basta colocar "#1," após a palavra PRINT:

```
260 PRINT#1,"Pontos: "
```

Agora é possível imprimir texto na tela de gráfico. O canto superior esquerdo do primeiro caractere impresso aparecerá no cursor gráfico. Por exemplo, para adicionar uma mensagem na parte inferior do exemplo dos 25 círculos, acrescente o comando OPEN:

```
15 OPEN "GRP:" FOR OUTPUT AS #1
```

e as linhas:

```
103 PRESET(90,150)
106 PRINT#1,"25 circulos"
```

Se desejar, poderá alterar a cor do texto, como:

```
104 COLOR 12
```

Existem algumas diferenças para imprimir neste modo na tela. A idéia principal é a de que qualquer imagem embaixo dos caracteres será simplesmente sobreposta e não apagada. Caso você faça uma impressão em um local em que já exista o texto, o texto antigo não será eliminado. Também a impressão de espaços não eliminará outros desenhos.

Sprites

Os sprites são figuras que tornarão bem mais simples elaborar jogos e animação com o MSX-BASIC. São imagens especiais que você pode manipular, podendo ser movimentados pela tela sem interferir nas outras imagens. Por exemplo, você pode mover um sprite pela tela, e a imagem desenhada não será apagada pelo movimento do sprite.

A utilização dos sprites em toda sua riqueza requer um conhecimento da operação interna de seu computador MSX que está além do contido neste livro. Os comandos relacionados aos sprites, bem como alguns exemplos de sua utilização, são mostrados neste capítulo para aguçar seu apetite para experiências posteriores. Caso haja interesse, com certeza o manual de seu MSX-BASIC lhe fornecerá maiores informações sobre como programar com sprites. Vamos começar.

O comando SCREEN tem um segundo argumento que controla os tamanhos dos sprites. Sua estrutura é:

```
SCREEN modo,tamanhosprite
```

A variável tamanhosprite terá de ser 0, 1, 2 ou 3; a Tabela 11.3 descreve cada um deles. Na discussão que se segue, estamos assumindo como padrão o valor 0. A cada comando SCREEN, toda informação sobre os sprites é destruída para que haja a certeza de que o sprite solicitado com o comando SCREEN seja montado corretamente.

Você pode fornecer um número para cada sprite. Podem existir até 64 (ou 256) sprites, numerados de 0 a 63 (ou 255). Ao acessar um sprite com os comandos neste capítulo, sempre se utiliza o número do sprite.

A imagem assumida por um sprite é mantida em uma área de memória, reservada unicamente para sprites. Não é possível utilizar os comandos gráficos para definir a forma do sprite; é obrigatório o uso de dados em *série* e a função SPRITE$. Cada sprite tem oito caracteres de comprimento; na montagem de um sprite, o MSX-BASIC verifica os bits no valor ASCII de cada caractere. Veja a Figura 11.8.

Tabela 11.3 Argumento do tamanho do sprite no comando SCREEN.

Valor	Significado
0	Cada sprite tem 8 × 8 pontos, com 8 bytes de dados.
1	Cada sprite tem 8 × 8 pontos com 8 bytes de dados, sendo que o sprite é aumentado em 2x na tela.
2	Cada sprite tem 16 × 16 pontos, com 32 bytes de dados.
3	Cada sprite tem 16 × 16 pontos, com 32 bytes de dados, sendo que o sprite é aumentado em 2x na tela.

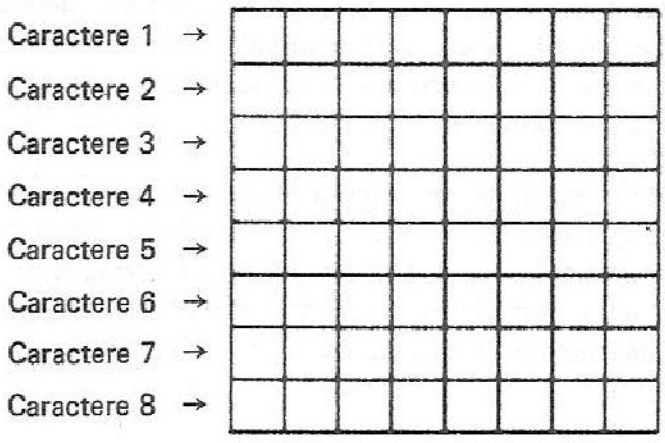


Figura 11.8 Definição do sprite.

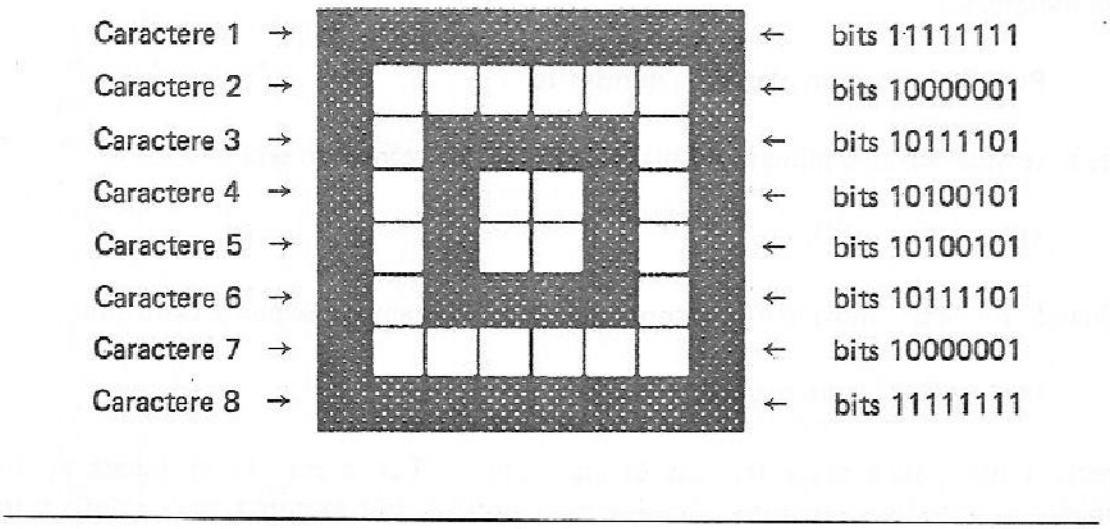


Figura 11.9 Esboço do exemplo de sprite.

Ao definirmos a forma do sprite, cada bit com o valor 1 se apresenta como um ponto ligado; cada bit que é um 0 é transparente, significando que, ao movimentar o sprite pela imagem, a imagem pode ser vista através do sprite nos locais em que existir um bit 0. Para desenhar uma figura com 8 × 8 ponto, é mais fácil esboçá-la em uma grade, utilizando em seguida o operador binário "&B" para melhor visualização.

Por exemplo, a Figura 11.9 mostra um sprite consistindo em dois quadrados. O programa que segue monta a variável S1$ com um sprite.

```
100 B1$ = CHR$(&B11111111)
110 B2$ = CHR$(&B10000001)
120 B3$ = CHR$(&B10111101)
130 B4$ = CHR$(&B10100101)
140 S1$ = B1$+B2$+B3$+B4$+B4$+B$3+B$2+B$1
```

Uma vez definido S1$, podemos utilizar a função SPRITE$ para armazenar o valor do sprite. A estrutura da função sprite é:

```
SPRITE$(numimag) = série
```

Vamos assumir que você quer armazenar isto no sprite de número 5. Seria então dado o comando:

```
150 SPRITE$(5) = S1$
```

O comando PUT SPRITE permite que se coloque e movimente o sprite na tela. Ainda é possível alterar sua cor, assim como o número de plano em que está. A primeira utilização para

o comando PUT SPRITE é fornecer ao número do sprite um número de plano. Sua estrutura para esta indicação é:

```
PUT SPRITE num plano , , , numsprite
```

Para este programa, adote o número do plano 1. Assim, seu comando será:

```
160 PUT SPRITE 1, , , 5
```

Quando queremos movimentar o sprite pela tela, utilizamos a seguinte estrutura:

```
PUT SPRITE num plano,STEP(x,y),cor
```

Pode parecer um pouco maçante, mas os argumentos STEP e cor são opcionais. Se for dado o argumento cor, todos os pontos com 1 serão coloridos. Por exemplo, para colorir o sprite de cinza e colocá-lo no centro da tela, dê o comando:

```
170 PUT SPRITE1,(128,96),14
```

Agora estamos prontos para movimentar o sprite pela tela com o comando PUT SPRITE. Isto poderá ser observado em cores, no programa abaixo, a primeira parte do programa desenha um fundo complexo, a segunda parte (das linhas 100 até 170) são as linhas do exemplo anterior, e a terceira parte permite que o sprite seja movimentado com as teclas de controle do cursor. Para movimentar o sprite, acione uma das teclas com uma seta; para parar o programa acione a tecla RETURN.

```
 10 REM  Exemplo de sprite
 20 SCREEN 2,0
 30 COLOR 11,8: CLS
 40 REM  Coloque alguns graficos na tela
 50 GOSUB 500
 60 GOSUB 600
100 B1$ = CHR$(&B11111111)
110 B2$ = CHR$(&B10000001)
120 B3$ = CHR$(&B10111101)
130 B4$ = CHR$(&B10100101)
140 S1$ = B1$+B2$+B3$+B4+B4$+B3$+B2$+B1$
150 SPRITE$(5) = S1$
160 PUT SPRITE 1, , , 5
170 PUT SPRITE 1,(128,96),14
200 REM  Rotina de entrada de teclado
210 K$ = INPUT$(1)
220 REM  Verifique o RETURN
230 IF K$ = CHR$(13) THEN GOTO 390
240 ON(ASC(K$)-27) GOSUB 270,300,330,360
```

```
250 PUT SPRITE1, STEP(X,Y)
260 GOTO 200
270 REM operado para a direita
280 X = 2: Y = 0
290 RETURN
300 REM Pressionado para esquerda
310 X = -2: Y = 0
320 RETURN
330 REM Pressionado para cima
340 X = 0: Y = -2
350 RETURN
360 REM Pressionado para baixo
370 X = 0: Y = 2
380 RETURN
390 REM Saia do programa
400 COLOR 15,4
410 END
500 REM Muitos quadrados coloridos
510 A = RND(-TIME)
520 FOR I = 1 TO 45
530 LINE(RND(1)*255,RND(1)*192) - STEP
    (20,40),(I MOD 15), BF
540 NEXT I
550 RETURN
600 REM Acrescente um texto
610 REM Elimine uma barra na parte superior
620 LINE(0,0) - (255,8),1, BF
630 OPEN "GRP:" FOR OUTPUT AS # 1
640 PSET(5,0),1
650 PRINT#1,"Utilize setas para movimentar"
660 RETURN
```

Caso queira experimentar com outras figuras, procure alterar as linhas 100 até 150. Por exemplo, as linhas que seguem fazem um quadrado sombreado.

```
100 B1$ = CHR$(&B10101010)
110 B2$ = CHR$(&B01010101)
140 S1$ = B1$+B2$+B1$+B2$B1$+B2$+B1$+B2$
```

Se você estiver interessado em algo estranho, o que segue poderia ser utilizado em um jogo:

```
100 B1$ = CHR$(&B00011000): B5$ = B1$
110 B2$ = CHR$(&B00100100): B4$ = B2$
120 B3$ = CHR$(&B01011010)
125 B6$ = CHR$(&B00100100)
```

```
130 B7$ = CHR$(&B11000011)
135 B8$ = CHR$(&B01000010)
140 S1$ = B1$+B2$+B3$+B4$+B5$+B6$+B7$+B8$
```

Para os programadores experientes que forem utilizar os sprites, existem mais dois comandos que podem ser usados. O comando SPRITE ON permite que uma função especial do MSX-BASIC seja ligada e desligada, pulando para uma sub-rotina quando dois sprites se tocam. A linha para o qual o MSX-BASIC pula é ativada pelo comando ON SPRITE GOSUB. Caso você esteja interessado em continuar neste nível de programação, existem muitas coisas interessantes que podem ser realizadas com sprites.

CAPÍTULO

12

SOM

Uma das primeiras coisas que chamam a atenção quando entramos em um fliperama – mesmo antes de ver os jogos – é o barulho. Escutamos uma cacofonia de extraterrenos morrendo, monstros mastigando e as canções-tema dos diversos jogos. Com comandos simples do MSX-BASIC é possível descrever programas que produzem os mesmos tipos de sons de alta qualidade.

Alguns computadores MSX, como o CX5M da Yamaha, apresentam características musicais que excedem aqueles discutidos neste capítulo. Utilize o manual que acompanha o computador para saber como utilizar-se deles. Mesmo que você não tenha um destes aparelhos musicais avançados, verá que a música (e o barulho) que pode ser criado com os comandos do MSX-BASIC é de excelente qualidade.

São três as características do sistema musical do MSX que o tornam tão poderoso:

1. A maioria dos computadores domésticos tem apenas uma "voz", significando que podem produzir apenas um som por vez. Os computadores MSX têm três vozes, e assim é possível fazer música mais sofisticada.
2. O MSX-BASIC toca música enquanto o programa estiver sendo processado. Significa que você pode escrever um fundo musical, que toca durante o processamento do programa, sem que haja atraso no processamento.
3. O computador MSX pode ainda variar as características principais do som que produz. É possível adaptar o som criado de modó que aparente diversos instrumentos, ou sons complicados para jogos.

Antes de iniciar as experiências com os programas deste capítulo, deve-se acrescentar um alto-falante ao seu sistema: o computador MSX comum não dispõe de tal ampliador próprio. É necessário ou um aparelho de televisão, ou um monitor com alto-falante ou ainda um conector para "saída de áudio" no computador, que permita sua ligação a um equipamento de som. Caso utilize o aparelho de televisão, certifique-se de que o volume está suficientemente alto, permitindo que você escute a música que está produzindo.

Apenas um Beep

Antes de escutar todos os sons maravilhosos que existem disponíveis no MSX-BASIC, será interessante observar como é produzido o som mais simples. O comando BEEP faz exatamente aquilo que está implícito em seu nome: emite um pequeno apito no alto-falante. Tem utilidade quando se deseja emitir um aviso de alarma ou enfatizar uma mensagem.

A estrutura do comando BEEP é:

```
BEEP
```

Por exemplo:

```
400 PRINT "Nao faca isto!"
410 BEEP
```

É possível criar um apito imprimindo o caractere 7 em ASCII. Por exemplo:

```
400 PRINT "Nao faca isto!"; CHR$(7)
```

Tocando Música

A maneira mais simples de escrever música no MSX-BASIC é pela utilização do comando PLAY. O comando PLAY é como o comando DRAW; utiliza *séries* (como argumento) que são escritos em linguagem própria, que é denominada Linguagem Musical ou Macro, ou LMM. Se estiver familiarizado com escalas e os rudimentos musicais, terá muita facilidade na criação de música com o comando PLAY.

A estrutura do comando PLAY é bem simples:

```
PLAY série1,série2,série3
```

As *séries* são os comandos LMM para cada uma das três vozes. Caso estejam sendo controladas vozes diferentes da voz 1, é necessário colocar as vírgulas no comando PLAY para manter os lugares. Por exemplo, para tocar uma música na voz 3, dê o comando

```
PLAY , , "CDEF"
```

As regras que envolvem o LMM são muito parecidas com aquelas do GML. Os comandos podem ser separados com espaços ou ponto e vírgula, e podem ser utilizadas variáveis MSX-BASIC em vez de argumentos numéricos pela colocação de um sinal de igual antes do nome e um ponto e vírgula após o nome. Os comandos podem ser dados em letras maiúsculas ou minúsculas. Os comandos LMM estão resumidos na Tabela 12.1.

Tabela 12.1 Sumário de comandos LMM.

Comando	Significado
Cn	Nota musical,1/n de duração
Dn	Nota musical,1/n de duração
En	Nota musical,1/n de duração
Fn	Nota musical,1/n de duração
Gn	Nota musical,1/n de duração
An	Nota musical,1/n de duração
Bn	Nota musical,1/n de duração
Rn	Pausa,1/n de duração
#	Sustenido
+	Sustenido
-	Bemol
.	Aumento da duração em 50%
On	Oitava(1-8,padrão é 4)
Ln	Comprimento de nota(1-64, padrão é 4)
Tn	Tempo em quartos de nota por minuto (32-255,padrão é 120)
Vn	Volume(0-15,padrão é 8)
Nn	Nota determinada(1-96)
Mn	Período da envoltória(1-65535,padrão é 255)
Sn	Forma de onda(1-15,padrão é 1)
X*série*	Executa conteúdo da *série*.

NOTAS SIMPLES

Como pode ser visto pela tabela, tocar notas comuns é bem fácil: basta dar o nome da nota. Por exemplo, para tocar as primeiras notas da canção infantil, "Maria tinha um carneirinho" na voz 1, forneça o comando:

```
PLAY "edcdeee"
```

Ao executar este comando, escutamos estas notas no alto-falante.

Os sustenidos e bemóis são tão fáceis como as notas básicas. Para elevar a melodia anterior de uma nota, dê o comando:

```
PLAY "fd#c#d#fff"
```

ou:

```
PLAY "fd+c+d+fff"
```

O valor-padrão para a duração da nota é 4, significando que esta é tocada como uma nota de um quarto. O valor 1 corresponde a uma nota inteira, 2 corresponde à meia nota etc. Para se tocar a melodia original em terços de nota, poderíamos escrever:

```
PLAY "e3d3c3d3e3e3e3"
```

Podemos especificar comprimentos de 1 a 64. O comando R representa uma pausa e pode ser utilizado exatamente como as outras notas.

Você deve ter observado que o MSX-BASIC não colocou uma pausa curta entre as três últimas notas da frase porque eram a mesma nota. Escutamos pausas apenas entre notas diferentes. Esta limitação pode ser vencida com algum esforço. Posteriormente, veremos como o MSX-BASIC coloca pequenas pausas entre repetições da mesma nota.

TOCANDO MÚSICA MAIS AVANÇADA

Cada oitava é distribuída de C (nota dó) até B (nota si). As oitavas são numeradas de 1 até 8, sendo 4 a oitava-padrão. Esta quebra da oitava poderá atrapalhar; por exemplo, se a melodia original fosse abaixada em duas notas, teríamos:

```
PLAY "dca#cddd"
```

Ao ser tocado, isto soa estranho, já que A# é um pouco elevado demais para as outras notas. Esta situação poderá ser alterada facilmente pela adição de um deslocamento de oitava ao redor da nota A#:

```
PLAY "dco3a# o4cddd"
```

É tão fácil utilizar duas vozes como utilizar uma. Por exemplo, as notas iniciais de uma outra melodia podem ser escritas como:

```
PLAY "fedc","ggbo5co4"
```

Com o "o4" você volta para a oitava original. É fácil esquecer-se de repor coisas deste tipo, mas o MSX-BASIC lembra a última colocação de uma voz após o término da declaração PLAY. Para confirmar isto, retire o "o4" da voz da segunda *série* e escreva o comando duas vezes. Observe que ao processá-lo pela segunda vez, a segunda voz inicia uma oitava mais alto.

O comando L fornece a velocidade para as notas sem comprimento; o padrão é 4. Pode-se utilizar este comando caso se deseje aumentar a velocidade. Por exemplo, para tocar tudo no dobro do tempo, dê o comando:

```
PLAY "L8"
```

Lembre-se de dar o comando "l4" para repô-lo.

O comando T também altera a velocidade. O padrão é 120, significando 120 quartos de nota por minuto. A velocidade pode ser colocada em qualquer ponto entre 32 (lento) e 255 (rápido). O volume pode ser alterado com o comando V. Inicialmente é colocado em 8, mas pode ser colocado em qualquer ponto entre 0 (silêncio) e 15 (alto) .

Se quiser especificar notas como números e não como letras, utilize o comando N. N0 é a nota mais baixa e N96 a mais alta; o Dó central, representando por O4C, é N36. Isto somente tem serventia quando utilizamos variáveis MSX-BASIC. Por exemplo, para tocar notas (incluindo sustenidos) na oitava iniciando em O4C, em dezesseis avos, devemos dar o comando:

```
420 FOR I = 36 TO 48
430 PLAY "n = i: 16"
440 NEXT I
```

Os comandos M e S (descritos na próxima seção) alteram a onda sonora produzida pelo MSX-BASIC; somente têm utilidade se você tiver um bom conhecimento de como um som musical é gerado. Você deve observar que o comando S se sobrepõe ao comando V, de tal modo que não é possível colocar o volume após a utilização do comando S. O comando X permite que se utilize uma *série* no comando PLAY.

É possível também verificar se o MSX está tocando com a função PLAY. Sua estrutura é:

```
PLAY(numvoz)
```

e retorna -1 (verdadeiro) se houver música na voz "numvoz". Por exemplo:

```
230 IF PLAY(1) THEN GOTO 230
240 PRINT "Nao ha nada tocando em 1"
```

Utilize 0 como "numvoz" para verificar se existe alguma voz tocando.

COMO OPERA O COMANDO PLAY

Para cada comando PLAY, o MSX-BASIC coloca em fila, em um buffer, as notas. Uma vez que a música foi colocada no buffer, o MSX-BASIC continuará processando o programa enquanto toca a música Na verdade, caso o programa tenha terminado e o MSX-BASIC já deu o "Ok", o buffer musical continuará tocando até ficar vazio.

Caso você dê um segundo comando PLAY, o MSX-BASIC coloca as notas no buffer. Presuma, por exemplo, que U1$ e U2$ tenham trechos extensos de música. Se você executar os comandos:

```
140 PLAY U1$
150 PRINT "Favor manter..."
160 PLAY U2$
170 PRINT "Continue mantendo..."
```

o primeiro comando PRINT será executado enquanto a primeira e a segunda notas de U1$ estiverem tocando, e será impresso também o segundo comando PRINT. A música de U2$ é iniciada logo após o término de U1$.

Mesmo que U2$ inicie imediatamente, é possível perceber um pequeno intervalo na música após U1$. Esta interrupção poderá ser utilizada em seu favor; supera a impossibilidade de diferenciar entre duas notas repetidas. Processe o seguinte programa, observando a diferença em duas linhas:

```
10 REM Provocando pequenas interrupcoes
20 REM Sem interrupcao:
30 PLAY "edcdeeer1"
40 REM Com interrupcoes:
50 PLAY "edcde": PLAY "e": PLAY "e"
```

Este foi o truque sugerido anteriormente neste capítulo. Pode ser um método simples, mas dá resultado.

Quando um programa que esteja tocando música for interrompido pela operação de CTRL+STOP ou com um comando STOP em seu programa; ou pela existência de um erro no programa, o MSX-BASIC executa um "bip" e desliga o som. Desta maneira, todas as alterações feitas na música são repostas: O, L, T, V, M e S são todos recolocados com seus valores-padrões. Um comando BEEP no programa possui o mesmo efeito.

Fazendo Barulho

O comando SOUND permite que programadores experientes tenham acesso a todas as partes do gerador de som do MSX. Esta seção presume que você esteja familiarizado com a formação dos sons e a preparação dos registradores eletrônicos. Com SOUND é possível criar diversos ruídos e efeitos sonoros.

Com o comando SOUND é possível determinar os valores do circuito integrado responsável pela geração de som em seu computador MSX. Sua estrutura é:

```
SOUND numreg,valor
```

O "numreg" é o número do registro, de 0 até 13. O "valor" é o número que deve ser armazenado no registro de um byte. A Tabela 12.2 é um resumo dos registros. (Observe que as vozes são denominadas A, B e C em vez de 1, 2 e 3 do comando PLAY.)

Os registros 0 e 1 formam um registro de 12 bits (os quatro bits superiores do registro 1 não são utilizados), que é o tom da voz A; os registros 2 e 3 formam o par da voz B, e os registros 4 e 5 são o par da voz C. Carregue-os com o tom desejado. Para determinar o tom utilize a seguinte equação:

```
tom = 124797/freqüência
```

Tabela 12.2 Resumo dos registros SOUND.

Número	Finalidade
0	Tom da voz A, Byte inferior
1	Tom da voz A, Byte superior
2	Tom da voz B, Byte inferior
3	Tom da voz B, Byte superior
4	Tom da voz C, Byte inferior
5	Tom da voz C, Byte superior
6	Controle do gerador de ruídos (0–31)
7	Controle do liga/desliga voz e ruído
8	Controle de amplitude da voz A
9	Controle de amplitude da voz B
10	Controle de amplitude da voz C
11	Período da envoltória, byte inferior
12	Período da envoltória, byte superior
13	Forma da envoltória

Assim, para gerar um som de 440 Hz, o tom será 284. Carregue os registros:

```
SOUND 0,29
SOUND 1,1
```

O registro 6 estabelece o período do ruído, criando aspectos diferentes que dependem da situação de todos os registros; para efeitos sonoros, principalmente aqueles envolvendo trinados, experimente este registro. O registro 7 é utilizado para ligar/desligar o ruído e as vozes. A Figura 12.1 mostra como este registro é formado. Para ativar uma voz (ou um ruído nesta voz) coloque o bit correspondente em 0. A colocação deste bit em 1 desliga esta voz. Para desligar o gerador de ruído e ativar as vozes A e B, dê o comando:

```
SOUND 7,&B00111100
```

	Ruído C	Ruído B	Ruído A	Som C	Som B	Som A
Bit	6	5	4	3	2	1

Figura 12.1 Disposição do registro 7.

Os registros 8 até 10 controlam a necessidade de utilização do controle de volume ou controle da envoltória/forma de onda. A Figura 12.2 indica como utilizar estes registros para cada um dos casos. Os três bits superiores são ignorados. Para controle de volume, os bits 1 a 4 variam o volume de 0 (baixo) até 15 (alto).

Os registros 11 e 12 estabelecem o período da envoltória, e o registro 13 a forma de onda. Para determinar a freqüência da envoltória, você pode utilizar a equação:

freqüência = 7799.8/período

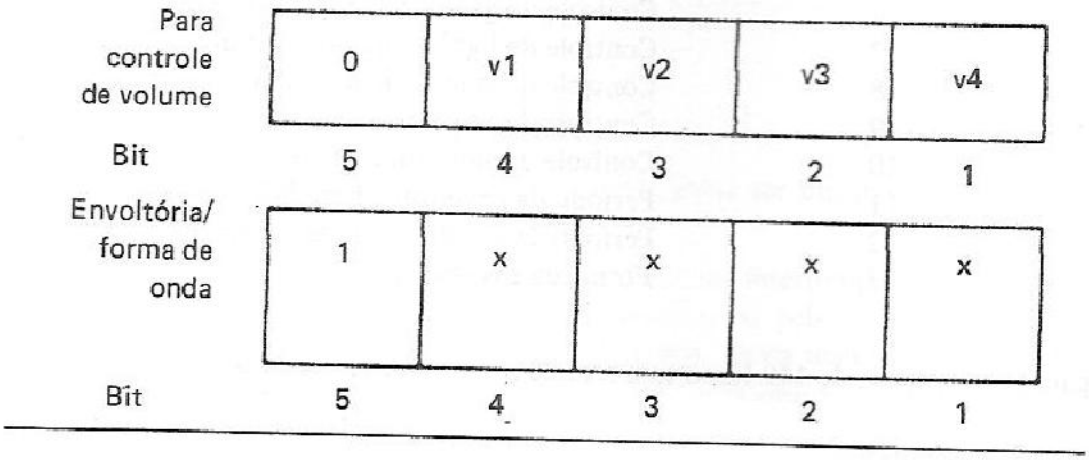
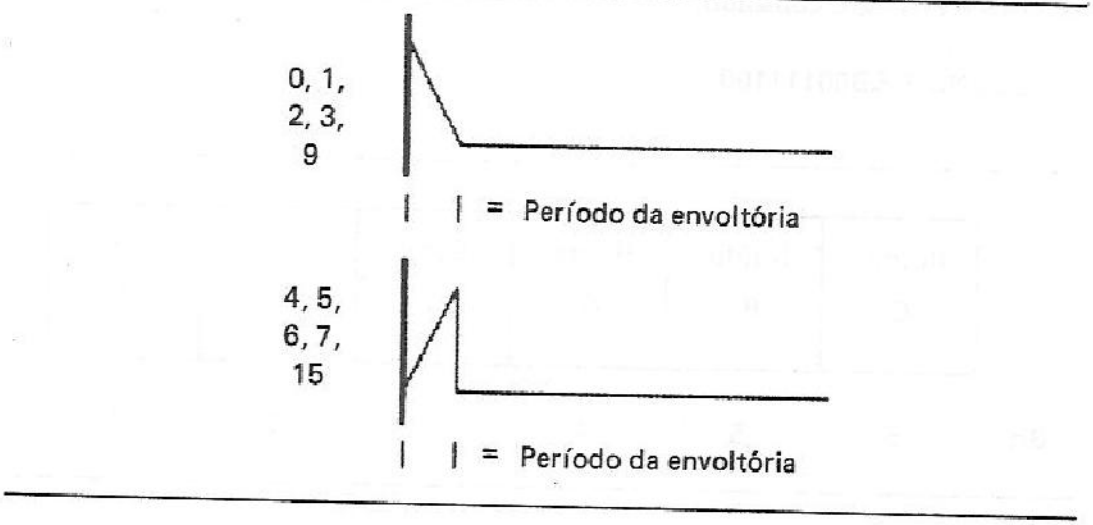


Figura 12.2 Disposição dos registros de 8 até 10.

A Figura 12.3 indica-nos as formas de onda que podem ser utilizadas. Observe que são utilizados apenas os quatro bits inferiores do registro 13.



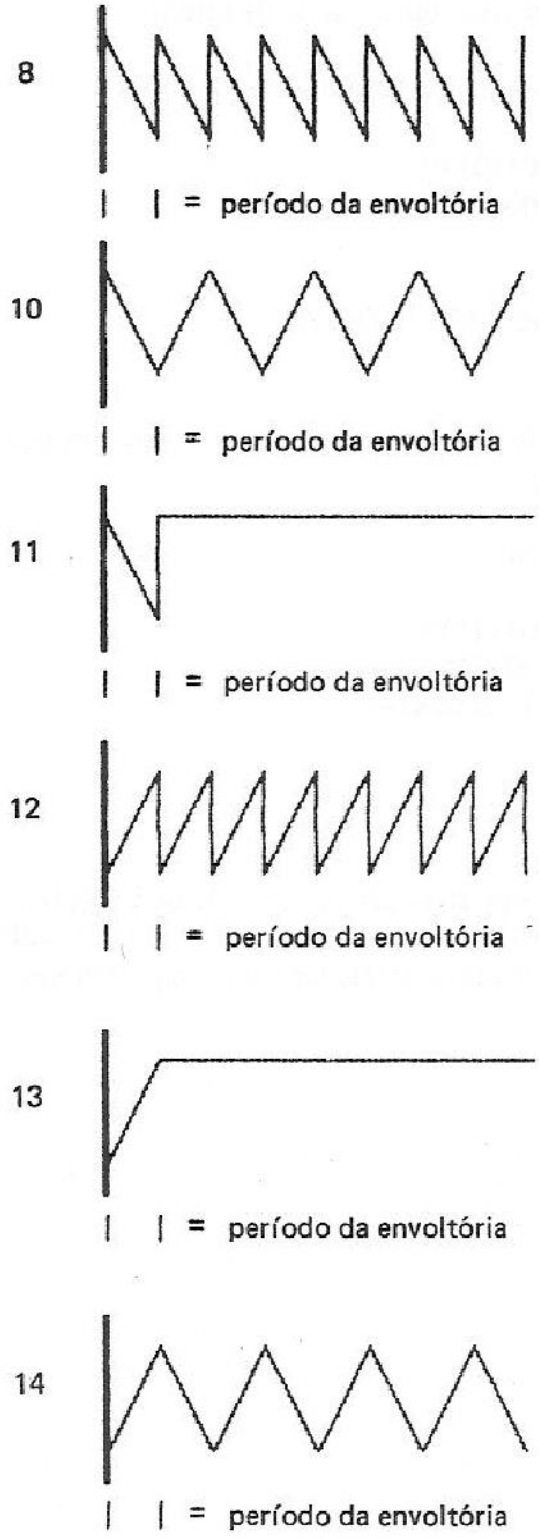


Figura 12.3 Formas de onda do registro 13 (cont).

Existem muitos sons estranhos e interessantes que podem ser gerados com o comando SOUND. Tente a experiência abaixo com a função de ruído:

```
10 REM Dirigindo um jipe (exemplo de ruido)
20 SOUND 6,10
30 SOUND7,&B00110110
40 SOUND8,&B00000100
50 SOUND1,5
60 TIME = 0
70 IF TIME < 100 THEN GOTO 70
80 BEEP
```

Com a utilização do comando SOUND, temos uma melhor variação nos tons do que com o comando PLAY. Por exemplo:

```
10 REM Tons puros
20 SOUND1,0
30 SOUND7,&B00111110
40 SOUND8,&B00000100
50 FOR X = 255 TO 0 STEP-1
60 SOUND0,X
70 NEXT X
80 BEEP
```

À medida que você continuar operando com o comando SOUND, verá que existe uma faixa muito grande de ruídos interessantes. Apesar de ser necessário muitas tentativas ao se operar com os efeitos de muitas vozes, os resultados poderão ser bem compensadores.

CAPÍTULO

13

OUTRAS INTERFACES

Os três últimos capítulos nos mostraram que seu computador MSX pode fazer mais do que simplesmente processar números. Você viu como ele apresenta e recupera informação, cria gráficos, toca música e faz ruídos. Este capítulo abrange processos de entrada e saída do MSX-BASIC que aumentam ainda mais a capacidade do seu computador.

Entre os periféricos existentes, dois dos mais populares para qualquer computador doméstico são os acionadores de disco e os *joysticks*. (O *joystick* é um acréscimo muito importante para qualquer sistema que é utilizado em jogos.) Uma vez que o MSX-BASIC trata dos arquivos em disco como dispositivos de entrada/saída, estes comandos são apresentados neste capítulo.

Joysticks

No seu computador MSX existe pelo menos um conector compatível com um joystick do tipo Atari; a maioria dos MSX permite conectar dois. Um joystick tem oito posições (para cima, para baixo, esquerda, direita e diagonais), e um gatilho ou botão que pode ser pressionado. Duas funções do MSX-BASIC lêem os botões do joystick e do gatilho.

Mesmo que você não tenha um joystick, é possível sua simulação com as teclas de controle do cursor. Elas não são tão divertidas nem tão precisas como o joystick para, por exemplo, dirigir espaçonaves. Apesar disto são úteis. Além das direções fundamentais fornecidas pelo controle do cursor, é possível indicar movimentos diagonais pressionando suas teclas ao mesmo tempo. A barra de espaço substitui o gatilho.

A função STICK lê um joystick e fornece um valor numérico de acordo com a direção em que foi empurrado. Sua estrutura é:

```
STICK(n)
```

onde n é o número do joystick que está sendo lido. A Tabela 13.1 fornece os valores de n, e a Figura 13.1 mostra os valores possíveis da função STICK.

Tabela 13.1 Conectores do joystick.

Número	Nome
0	Teclas de controle do cursor
1	Conector 1 do joystick
2	Conector 2 do joystick

A função STRIG tem um valor -1 se o gatilho foi empurrado ou 0 quando não foi. Sua estrutura é:

```
STRIG(n)
```

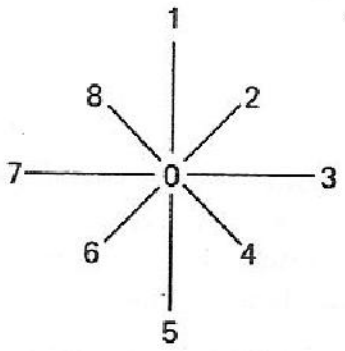


Figura 13.1 Valores da função STICK.

onde n é o número do joystick sendo lido (da mesma maneira que na função STICK).

Já que tanto a função STICK como a função STRIG têm o valor 0 quando nada estiver acontecendo, isto poderá ser utilizado em expressões lógicas sem se fazer comparações. Por exemplo, a expressão abaixo informa se o gatilho do joystick 1 está sendo operado:

```
IF STRIG(1) THEN PRINT "Bang bang!"
```

Para ver como estas funções operam em um programa, as linhas abaixo substituem as linhas 200 até 410 no programa de movimentação de sprite do Capítulo 11. O valor da variável JN deve ser 0 para a tecla de controle do cursor, ou 1 ou 2 para o joystick. Pressionando-se o gatilho, terminamos o programa.

```
200 REM Entrada de joystick ou tecla do cursor
210 IF STRIG(JN) THEN GOTO 310
220 D = STICK(JN)
230 IF D = 0 THEN GOTO 220
240 IF ((D >= 2) AND (D <= 4)) THEN X = 2
250 IF ((D >= 4) AND (D <= 6)) THEN Y = 2
260 IF ((D >= 6) AND (D <= 8)) THEN X = -2
270 IF ((D = 8) OR (D = 1) OR (D = 2)) THEN Y = -1
```

```
290 PUT SPRITE1, STEP(X,Y)
295 X = 0: Y = 0
300 GOTO 200
310 REM Termine o programa
320 COLOR15,4
330 END
```

O MSX-BASIC estabelece uma característica interessante para os programadores de jogos que utilizam o joystick. Os comandos ON STRIG GOSUB e STRIG ON permitem ao MSX-BASIC descobrir quando um gatilho foi pressionado, independentemente de onde você está no programa. Caso seu programa utilize estes comandos e o usuário pressione um gatilho, o MSX-BASIC interrompe o que estiver fazendo e passa para a sub-rotina especificada (procedimentos deste tipo são denominados "procedimentos de interrupção"). Isto significa que seu programa poderá estar executando outros processos do MSX-BASIC sem estar sempre verificando a função STRIG.

Para programar o procedimento de interrupção, utilize o comando ON STRIG GOSUB. Sua estrutura é:

```
ON STRIG GOSUB subr0,subr1,subr2
```

Este comando informa ao MSX-BASIC qual sub-rotina deve ser executada quando o gatilho 0, 1 ou 2 for pressionado. Cada argumento é o número da linha onde se inicia uma sub-rotina.

O comando STRIG ON também deve ser utilizado para ligar/desligar a manipulação da interrupção; o comando ON STRIG GOSUB apenas informa ao MSX-BASIC onde ir mas não para procurar interrupções. As estruturas do comando STRIG ON são as seguintes:

```
STRIG(n)ON
STRIG(n)OFF
STRIG(n)STOP
```

O argumento ON informa ao MSX-BASIC para procurar interrupções do gatilho. Para parar de procurar interrupções, utilize o argumento OFF. O argumento STOP informa ao MSX-BASIC que continue procurando uma interrupção, mas que nada faça ao encontrar uma. Ao ser dado um comando STRIG ON, o MSX-BASIC pula imediatamente para a sub-rotina especificada.

Por exemplo, o programa abaixo prepara as sub-rotinas para a barra de espaço e o gatilho 1:

```
520 ON STRIG GOSUB 800,900
530 STRIG(0) ON
540 STRIG(1) ON
 ...
800 PRINT "Gatilho do teclado acionado"
...
860 RETURN
900 PRINT "Gatilho#1 acionado"
...
960 RETURN
```

Arquivos em Disco

Um acionador de disco pode ser utilizado para muito mais do que simplesmente armazenamento e recuperação de informação de programas; você pode escrever e ler arquivos que contenham informações. Mesmo que você não tenha um acionador de disco, esta seção pode ser interessante, já que existem muitas vantagens na utilização de arquivos. Pode ser que, sabendo como funciona um acionador de disco, você se decida pela aquisição de um.

ABRINDO E FECHANDO ARQUIVOS

Já que o MSX-BASIC trata um arquivo como se este fosse um dispositivo, a discussão no Capítulo 11 a respeito dos dispositivos se aplica aos arquivos. Assim, para iniciar uma operação com um arquivo deve-se dar o comando OPEN. O nome do dispositivo no comando OPEN é o nome do arquivo, com uma designação do acionador de disco (opcional) antes do nome. (Se você não estiver familiarizado com designação de acionadores e nome de arquivos, veja o Capítulo 18.) O exemplo de um comando OPEN para um arquivo é:

```
30 OPEN "PROG1.DAT" FOR INPUT AS #2
```

Com o MSX-BASIC é possível ter 15 dispositivos abertos simultaneamente, sendo necessário utilizar a pseudovariável MAXFILES=. Sua estrutura é:

```
MAXFILES=n
```

Desta maneira o MSX-BASIC é informado que deve reservar espaço de memória para cuidar de n dispositivos (o valor-padrão é 1). Observe que a utilização do MAXFILES= limpa a memória de todas as variáveis, colocando todos os valores em 0. Portanto, MAXFILES= deve ser utilizado apenas no início do seu programa.

Ao utilizar o comando OPEN com arquivos, o argumento FOR pode assumir quatro valores. Estes valores são muito importantes, uma vez que a utilização do valor errado poderá apagar acidentalmente um arquivo. A Tabela 13.2 resume as escolhas. Note que se você abrir um arquivo com FOR OUTPUT e este arquivo já existir no disco, ele será apagado!

O comando CLOSE fecha um arquivo e libera o número do dispositivo. Sua estrutura é:

```
CLOSE dev1,dev2...
```

Não será necessário que você especifique o número do dispositivo; quando é dado o comando CLOSE sem argumento, todos os dispositivos serão fechados. Você deve ter certeza de que ao final do programa todos os arquivos estarão fechados, já que o MSX-BASIC às vezes não escreve a última informação no arquivo antes que este seja fechado. O comando END também fecha todos os arquivos abertos (de modo que terminar um programa com END é equivalente a dar o comando CLOSE).

Tabela 13.2 Escolhas para o argumento FOR no comando OPEN.

Escolha	Significado
INPUT	Ler o arquivo.
OUTPUT	Escrever no arquivo; apaga o arquivo se já existir.
APPEND	Escreve no fim do arquivo; não apaga um arquivo existente.
Não fornecido	Acesso aleatório; pode ler e escrever registros de comprimento fixo.

LENDO ARQUIVOS SEQÜENCIAIS

Para que seja possível ler a informação de um arquivo seqüencial, é necessário que ele tenha sido aberto com o argumento FOR INPUT. Para ler um arquivo, deve ser utilizado o comando INPUT ou LINE INPUT. Utilize estes comandos com #n, após a palavra INPUT, como em:

```
INPUT #2, GL$
```

A função INPUT$ também pode ser utilizada; neste caso, o número do arquivo vem após o número do string a ser lido. Por exemplo:

```
GL$ = INPUT$(5,2)
```

lê cinco caracteres do dispositivo 2.

Estes comandos operam da mesma forma que aqueles que você aprendeu no Capítulo 10, com a diferença de não imprimirem nada na tela. Por exemplo, assuma que a primeira linha no arquivo de dados FIRSTFIL.DAT é:

```
12,17, Jose Silva
```

e a primeira linha no arquivo de dados SCNDFIL.DATA é:

```
19,2, Robson dos Santos
```

Insira o seguinte programa:

```
10 REM  Exemplo de ler um arquivo
20 MAXFILES= 2
30 OPEN "FIRSTFIL.DAT" FOR INPUT AS #1
40 OPEN "SCNDFIL.DAT" FOR INPUT AS #2
50 PRINT "A primeira linha do primeiro arquivo e:"
60 LINE INPUT #1,A$
70 PRINT A$
80 PRINT "O primeiro numero no segundo arquivo e:";
90 INPUT #2, B
100 PRINT B
110 CLOSE
```

O programa imprime:

```
A primeira linha do primeiro arquivo e:
12,17,Jose Silva
O primeiro numero do segundo arquivo e: 19
```

Se você tentar ler, além do fim do arquivo, o MSX-BASIC dará uma mensagem de erro "Input past end". Para evitar que isto aconteça, utilize a função EOF. Esta função será -1 (verdadeiro) caso você esteja no fim do arquivo ou 0 quando não estiver (EOF significa – fim de arquivo – na linguagem de computação). Sua estrutura é:

```
EOF(n)
```

Neste caso n representa o número do dispositivo que está sendo verificado. O programa abaixo lê um programa de texto e o imprime na tela.

```
10 REM Imprima um arquivo de texto
20 OPEN "SAMPLE.TXT" FOR INPUT AS #1
30 IF EOF(1) THEN GOTO 70
40 LINE INPUT #1, V$
50 PRINT V$
60 GOTO 30
70 REM Nao ha mais linhas
80 END
```

ESCREVENDO EM ARQUIVOS SEQÜENCIAIS

Para escrever informações em um arquivo, este deverá ser aberto com FOR OUTPUT ou FOR APPEND. Como já foi dito, com a utilização de FOR OUTPUT qualquer arquivo com o mesmo nome será apagado antes de ser aberto. O argumento FOR APPEND abre o arquivo para leitura, adicionando informação apenas no fim do arquivo sem alterar a informação já existente. Utilize os comandos PRINT do Capítulo 11:

```
10 REM Exemplo de escrever um arquivo
20 MAXFILES= 2
30 OPEN "OUT1.FIL" FOR OUTPUT AS #1
40 OPEN "OUT2.FIL" FOR OUTPUT AS #2
50 PRINT #1, "Testando, a parte 1"
60 PRINT #2, "Testando, a parte 1"
70 CLOSE
80 REM Observe as proximas duas linhas
90 OPEN "OUT1.FIL" FOR OUTPUT AS #1
100 OPEN "OUT2.FIL" FOR APPEND AS #2
110 PRINT #1, "Testando, a parte 2"
120 PRINT #2, "Testando, a parte 2"
130 CLOSE
```

Após o processamento deste programa, OUT1.FIL conterá apenas:

```
Testando, a parte 2
```

porque na linha 90 foi utilizado FOR OUTPUT, mas OUT2.FIL conterá:

```
Testando, a parte 1
Testando, a parte 2
```

já que ele foi aberto com FOR APPEND na linha 100.

Para descobrir quantos caracteres foram escritos ou lidos de um arquivo, utiliza-se a função LOF. Sua estrutura é:

```
LOF(n).
```

É útil quando se deseja saber o espaço disponível do arquivo.

Antes de abrir um arquivo para gravação, poderá ser necessário saber se o disco tem espaço suficiente para armazenar os dados. A função DSKF informa quantos espaços estão disponíveis em seu disco (o manual do disco informa o número de kbytes possíveis em cada espaço). A estrutura da função DSKF é:

```
DSKF(numdriv)
```

O valor de "numdriv" é 0 para o disco em uso, 1 para "A:", ou 2 para "B:". Por exemplo, poderá haver interesse em incluir as seguintes linhas em um programa:

```
...
480 IF DSKF(0) < 3 THEN GOTO 2000
490 OPEN "DATA.OUT" FOR OUTPUT AS #3
...
2000 REM Aqui caso o disco esteja completamente cheio
...
```

ARQUIVOS DE ACESSO ALEATÓRIO

Um arquivo de acesso aleatório é aquele que pode ser lido e escrito onde se desejar. Pode parecer bem melhor do que um arquivo seqüencial, mas existem muitas limitações que dificultam a utilização dos arquivos de acesso aleatório. A principal dificuldade é que devem ser utilizados registros de comprimento fixo (os registros foram descritos no Capítulo 2 na discussão sobre sistemas de banco de dados).

Em resumo, para utilizar este tipo de arquivo, você terá de decidir, antes de utilizá-lo, como será cada registro. Cada registro é formado por campos, como os indicados na Figura 13.2. Não é apenas o tamanho do registro que é estabelecido, cada campo também tem o seu comprimento determinado. Haverá um desperdício muito grande de espaço no disco quando os registros

contiverem *séries*, uma vez que o comprimento do campo deve ter o comprimento de acordo com a maior *série* possível. Por exemplo, é possível estabelecer o campo para o nome da cidade com 20 caracteres, apesar de sabermos que a maioria das cidades tem nomes com menos de 10 caracteres.

Quando acessamos dados do arquivo, o MSX-BASIC lê um registro completo por vez e armazena a informação em um "armazém de apoio" (buffer), formado por variáveis alfanuméricas. Para acrescentar ou alterar dados, são utilizadas funções especiais para armazenar os dados no buffer que podem alterar ou acrescentar um registro inteiro por vez no arquivo.

Para abrir um arquivo de acesso aleatório, damos o comando OPEN sem o argumento FOR, mas com o argumento LEN= no fim. Sua estrutura é:

```
OPEN nomearq AS#n LEN= comreg
```

O argumento "comreg" é o comprimento, em caracteres, de cada registro e deve estar entre 1 e 256. Todo campo é armazenado como uma *série*, de modo que a soma do comprimento dos campos é igual ao comprimento do registro. Se estivermos armazenando dados numéricos, estes podem ser convertidos em *séries* para diminuir o espaço utilizado no arquivo. Os comprimentos dos tipos numéricos são:

- Os inteiros ocupam dois caracteres
- Números reais de simples precisão ocupam quatro caracteres.
- Números reais de dupla precisão ocupam oito caracteres.

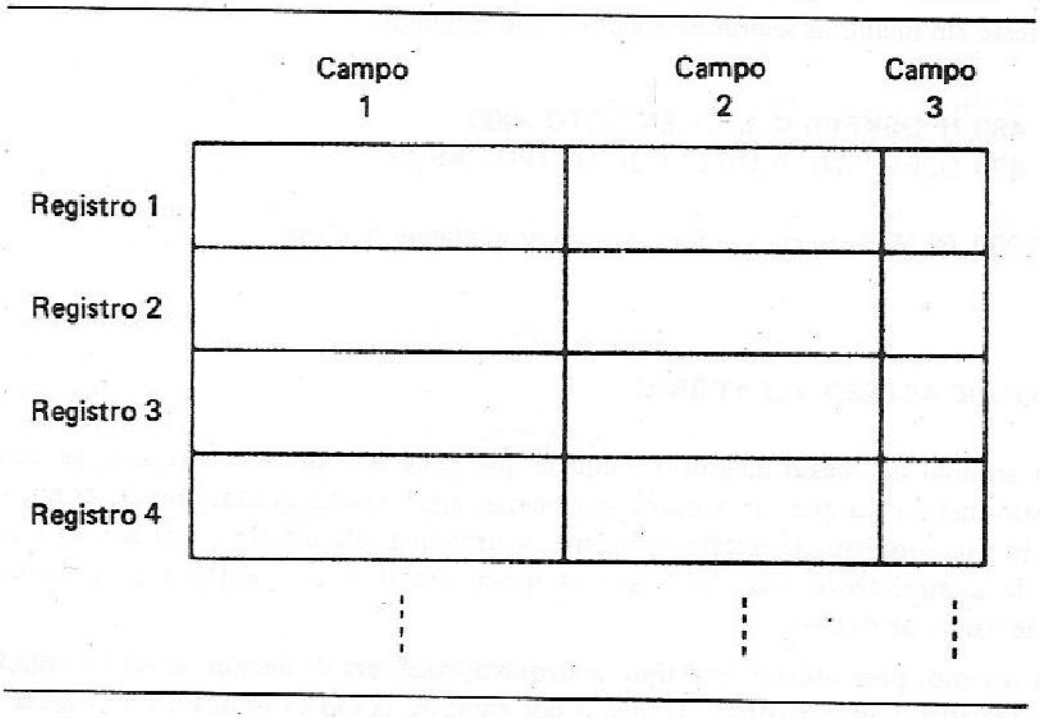


Figura 13.2 Registros e campos em um arquivo de acesso aleatório.

Após o comando OPEN, deverá ser dado o comando FIELD para informar ao MSX-BASIC a formação dos campos. Sua estrutura é:

```
FIELD #n,comp1 AS var1$,comp2 AS var2$, ...
```

Onde n representa o número do dispositivo, comp1 o número de caracteres no primeiro campo e var1$ o nome da variável do buffer que vai armazenar o primeiro campo. Deverá existir tantos "comp As var$" quantos forem os campos de registro, e os comprimentos devem estar de acordo com o comprimento total do registro especificado no comando OPEN.

Nos exemplos desta seção, os registros terão três campos: um para o sobrenome (comprimento 25), um para o nome (comprimento 15), e um para os pontos (números reais de simples precisão). Os comandos OPEN e FIELD são:

```
10 REM Acesso aleatorio
20 OPEN "TEST.DAT" AS #1 LEN= 44
30 FIELD #1, 25 AS LN$,15 AS FN$,4 AS SC$
```

Para ler um registro utilizamos o comando GET. Sua estrutura é:

```
GET #n,numeroreg
```

O "numeroreg" é o número do registro desejado. Assim o registro é lido no buffer. Por exemplo, para ler o quarto registro, deve ser dado o comando:

```
240 GET #1,4
```

Você pode utilizar uma variável para "numeroreg", como em:

```
180 H7 = H8 + 5
190 GET #1, H7
```

As variáveis *série* do buffer poderão ser utilizadas em seu programa MSX-BASIC, mas nunca no lado esquerdo de um comando LET (os resultados seriam desastrosos). Já que os números são armazenados de uma forma compacta no arquivo, deverão ser utilizadas funções de conversão para que possam ser transformadas em números.

As funções que você utiliza para converter um dado (que foi lido com um comando GET) para as variáveis do MSX-BASIC são CVI, CVS e CVD para inteiros, variáveis de simples precisão e variáveis de dupla precisão, respectivamente. As estruturas são:

```
CVI(buffervar$)
CVS(buffervar$)
CVD(buffervar$)
```

Por exemplo, para imprimir o conteúdo do buffer no exemplo dado, poderia utilizar-se o comando:

```
300 GET #1,RN
310 PRINT LN$;", ";FN$;", ";CVS(SC$)
```

A função CVS transforma a *série* de dois caracteres em um número de precisão simples. As funções CVI, CVS e CVD podem ser utilizadas também para cálculos:

```
720 TS = CVS(SC$)*5.3
```

Quando você escreve em um registro tudo aquilo que anteriormente estava armazenado nele é apagado. A criação de novos registros e a atualização de velhos registros possuem o mesmo efeito. Você pode utilizar o comando PUT para gravar um registro em um arquivo randômico. Sua estrutura é:

```
PUT #n,numreg
```

É o contrário do comando GET; lê o conteúdo do buffer e grava-o no disco.

O comando LET não pode ser utilizado para estabelecer valores às variáveis do buffer. Em vez disto, para colocar as *séries* no buffer, utilize os comandos LSET e RSET; sua utilização é idêntica ao comando LET. O comando LSET coloca uma *série* na variável do buffer com espaços em branco à esquerda, ao passo que o comando RSET coloca a *série* com espaços em branco à direita; quase sempre utilizamos o LSET.

Existe a possibilidade de converter números em *séries* para o buffer. Para isto utilize as funções MKI$, MKS$ e MKD$ que são análogas às funções CVI, CVS e CVD. As suas estruturas são:

```
MKI$(inteiro)
MKS$(real precisão simples)
MKD$(real precisão dupla)
```

Por exemplo:

```
100 LINE INPUT "Novo sobrenome:"; KA$
110 LINE INPUT "Novo nome:"; KB$
120 LINE INPUT "Pontos novos:"; KC$
130 SO!= VAL(KC$)
140 LSET LN$ = KA$
150 LSET FN$ = KB$
160 LSET SC$ = MKS$(SO!)
180 PUT #1,RN
```

Em caso de esquecimento do número do último registro lido ou escrito, utilize a função LOC. Sua estrutura é:

```
LOC(n)
```

Esta função raramente é utilizada, já que a maioria dos programas mantém um controle dos registros dos comandos PUT e GET, em uma variável.

Outras E/S

Apesar de os joysticks e arquivos em disco serem os dispositivos de entrada e saída mais comuns do MSX-BASIC, existem outros disponíveis. Por exemplo, com os comandos MOTOR ON e MOTOR OFF é possível ligar e desligar o motor de seu gravador cassete. Esta possibilidade pode parecer inútil à primeira vista, mas você pode usar outros periféricos e ligá-los na saída do cassete, fazendo o seu controle com os comandos do MSX-BASIC.

Alguns fabricantes de periféricos vendem "mesas digitais". Você aponta um local da mesa com um lápis especial e o MSX-BASIC lê esta posição. Este dispositivo é ligado aos conectores do joystick do seu computador MSX.

A função PAD é utilizada para ler os dados da "mesa". Sua estrutura é:

```
PAD(n)
```

O n, neste caso, tem um significado bem diferente do que para as funções STICK e STRIG. Na Tabela 13.3 estão resumidos os valores de n.

Tabela 13.3 Resumo dos argumentos PAD.

Número	Significado
0	Leia pad da porta 1; -1 significa tocado
1	Coordenada X já lida para porta 1
2	Coordenada Y já lida para porta 1
3	Situação do botão para porta 1; -1 significa tocado
4	Leia pad da porta 2; -1 significa tocado
5	Coordenada X já lida para porta 2
6	Coordenada Y já lida para porta 2
7	Situação do botão para porta 2; -1 significa tocado

Antes de utilizar PAD(1) ou PAD(2), você deve executar um comando com PAD(0), que verifica se o lápis especial encostou na "mesa"; caso tenha encostado, PAD(0) será -1 (verdadeiro). Após a confirmação de PAD(0), PAD(1) contém o valor da coordenada x e PAD(2) contém o valor da coordenada y. PAD(3) verifica a situação do botão no pad; será -1 (verdadeiro) caso o botão esteja pressionado e 0 (falso) em caso contrário. Por exemplo:

```
130 IF (NOT PAD(0)) THEN GOTO 130
140 PSET (PAD(1),PAD(2))
```

Com o seu computador MSX é possível utilizar também os controladores tipo "paddle". São relíquias dos primeiros tempos dos jogos de vídeo domésticos, e muitas pessoas ainda os têm. São encaixados no conector do joystick e podem ser lidos com a função PDL. Sua estrutura é:

```
PDL(n)
```

Da mesma forma que com STICK, n representa o número do conector do joystick. A função terá um valor entre 0 e 255. A função STRIG pode ser utilizada para ler o botão do "paddle".

CAPÍTULO

14

PROGRAMANDO EM LINGUAGEM ASSEMBLY

Quando você programa em MSX-BASIC, um outro programa (denominado interpretador BASIC) pega os seus comandos MSX-BASIC e os transforma em linguagem assembly do Z80, que pode ser "compreendida" pela UCP. (Uma vez que a UCP do computador MSX é um Z80, ela compreende a linguagem assembly do Z80.) A maioria das pessoas não se preocupa em entender a linguagem assembly; não é fácil e não tem muita utilidade para a maioria dos proprietários de computadores.

Entretanto, você poderá ficar fascinado com a perspectiva de operar com uma linguagem que pode executar 3,5 milhões de instruções por segundo em seu computador MSX (corresponde a aproximadamente 100 a 300 vezes mais rápida do que o MSX-BASIC). É possível escrever programas em linguagem assembly, inseri-los nos seus programas MSX-BASIC e processá-los junto com o MSX-BASIC. É possível programar tarefas complexas com a linguagem assembly do Z80 e tarefas simples com o MSX-BASIC.

A compreensão dos conceitos básicos da linguagem assembly poderá ser-lhe útil. Por exemplo, podemos utilizá-la para ler e alterar algumas posições da RAM que não são normalmente acessíveis com os comandos e funções do MSX-BASIC. Apesar de não ser tudo o que pode ser feito com a programação em linguagem assembly, a compreensão sobre o ambiente desta linguagem permite que você realize tarefas com o MSX-BASIC normalmente reservadas aos programadores de linguagem assembly.

Acesso à RAM

Existem muitas posições da RAM cujos valores podem ser importantes para seu programa. Por exemplo, ao passo que muitos valores podem ser fornecidos em MSX-BASIC, não há comando algum MSX-BASIC que pode verificar posteriormente os valores do programa. Em substituição deve ser utilizada a função PEEK para verificar o que existe nas posições de memória.

A estrutura da função PEEK é:

```
PEEK(end)
```

O argumento "end" representa o endereço ou posição do byte da RAM que desejamos verificar; deve variar de 0 até 65535. Um endereço indica um número de bytes do começo da RAM. Com o comando PEEK é possível verificar qualquer byte da RAM, seja ele parte do programa MSX-BAIC, do interpretador BASIC, ou parte da RAM que contém os valores determinados pelo MSX-BASIC para seu próprio uso.

Observe que na verdade os endereços de 0 até 32767 estão em ROM e não em RAM. Isto se deve ao modo com que o seu computador MSX utiliza dados e programas. Entretanto, estes endereços podem ser lidos com a função PEEK.

Por exemplo, enquanto você se encontra no modo 2, pode ser que queira encontrar as coordenadas x e y do cursor gráfico. Como já sabemos, não existe comando ou função do MSX-BASIC que faça isto. Entretanto, estes valores são mantidos em RAM para serem utilizados pelo MSX-BASIC nos endereços &HFCB3 e &HFCB5 (os programadores de linguagem assembly normalmente fornecem os endereços em notação hexadecimal). Assim, os valores poderiam ser encontrados com os comandos:

```
GX = PEEK(&HFCB3)

GY = PEEK(&HFCB5)
```

Não é possível causar danos à memória com a utilização da memória PEEK.

Caso tenha mais coragem, poderá alterar qualquer parte da RAM com o comando POKE. Tenha, porém, muito cuidado, porque a alteração de áreas desconhecidas da RAM poderá ter resultados inesperados, como a perda de um programa MSX-BASIC da memória. Um uso inadvertido do comando POKE poderá parar seu computador até que seja desligado e novamente ligado.

A estrutura do comando POKE é:

```
POKE end,valor
```

O "end" é o mesmo da função PEEK; "valor" é um número entre 0 e 255 para colocar no local. Observe que a utilização do comando POKE em qualquer endereço inferior a 32768 não tem utilidade, já que esta área está em ROM (que é impenetrável ao comando POKE).

Não existem muitas outras utilizações para o comando POKE, a não ser carregar rotinas em linguagem assembly na RAM. Porém, para ver como opera o comando POKE, altere a largura no modo 0 sem limpar a tela. O endereço &HF3B0 contém o comprimento da linha; para alterá-la para 34, dê o comando:

```
POKE &HF3B0,34
```

Isto altera a largura da tela para 34 caracteres.

O Apêndice E tem uma listagem dos locais da RAM que poderão ter interesse em serem utilizados por você para a função PEEK e o comando POKE. Observe que muitos destes locais são apenas de leitura; a alteração do valor deste local com o comando POKE poderá ou não alterar o modo de agir do MSX-BASIC.

Processando Programas em Linguagem Assembly

Caso você saiba como programar na linguagem assembly do Z80 (repetindo, isto apenas tem interesse ou utilidade para cerca de 5% dos proprietários de computadores MSX), você poderá incluir programas em linguagem assembly nos seus programas MSX-BASIC. Para tanto, adote o seguinte procedimento:

- Carregue o programa na memória ou em variáveis MSX-BASIC.
- Informe ao MSX-BASIC aonde está o programa.
- Informe ao MSX-BASIC para processar o programa da RAM.

Enquanto estes passos são bem diretos, é necessário, primeiramente decidir como armazenar o programa na memória. O método mais usual é armazenar as etapas em uma matriz, que em geral é preenchida com números armazenados em declarações DATA (veja no Capítulo 16 a discussão dos comandos DATA e READ). Por exemplo:

```
200 DIM PG%(1000)
210 FOR I = 0 TO 1000 STEP 2
220 READ A,B
230 PG%(I) = 256*A+B
240 NEXT I
1000 DATA 12,210,48
...
```

Podemos ainda utilizar os comandos BLOAD e BSAVE para armazenar e recuperar dados da memória; são também discutidos no Capítulo 16. Independentemente de como é carregado o programa, a última instrução deve ser um RET.

Depois que o programa estiver na memória, deve-se indicar onde ele começa, utilizando o comando DEF USR. Sua estrutura é:

```
DEF USRdigito=end
```

O argumento "dígito" vai de 0 até 9. É possível definir até 10 programas na memória. Se o dígito não é declarado, o MSX-BASIC presume que você esteja se referindo ao USR0. O argumento "end" é o endereço do início do programa. Ao utilizarmos BLOAD, "end" é o começo do código.

A utilização do comando DEF USR com uma matriz é mais complexa. É necessário utilizar a função VARPTR para determinar o endereço do início da matriz. A estrutura da função VARPTR é:

```
VARPTR(variavel)
```

Desta maneira é encontrado o endereço da variável. Para que isto seja utilizado com o exemplo anterior, deve ser dado o comando:

```
250 DEF US$0= VARPTR(PG%(0))
```

Para iniciar o funcionamento de um programa em linguagem assembly, utilize a função USR. Sua estrutura é:

```
var1 = USRdigito(var2)
```

Da mesma forma que no comando DEF USR, o "dígito" é assumido como 0 quando não é colocado e pode variar de 0 a 9. A variável "var1" é o resultado que retorna do programa de linguagem assembly (se houver), e a variável "var2" é a variável levada ao programa. Estas variáveis devem sempre estar presentes, podendo ser variáveis fictícias. (Significa que podem ter qualquer valor que se deseje.) Seu programa em linguagem assembly poderá verificar o registro A para o tipo de variável, caso o argumento "var1" seja passado para a função USR. Se for utilizada uma variável *série* (o registro A será 3), o registro DE aponta para o ponteiro *série* de três bytes (comprimento e endereço inicial de dados). Caso você utilize uma variável numérica, o registro HL aponta para o número atual. Para recolocar um valor à "var2", monte os registros como foi descrito anteriormente.

Para processar os exemplos anteriores de rotinas em linguagem assembly, seria utilizado o comando:

```
360 DM = US$0(DM)
```

Se o seu programa em linguagem assembly for armazenado em uma variável, chame o programa assim que tiver determinado o endereço da variável, de maneira que o MSX-BASIC não altere o endereço caso venha a embaralhar a memória.

O Apêndice E lista alguns ponteiros que existem na ROM do MSX-BASIC, que podem ser chamados com a função USR. Estes locais não precisam de argumento. Por exemplo, a única maneira de repor rapidamente todos os registradores sonoros com seus valores-padrões no MSX-BASIC é através do comando BEEP. Caso você queira repor todos os registros sem este comando, utilize os comandos abaixo:

```
DEF USR =&H0090
    DM =USR(DM)
```

Porta de Entrada e Saída

Se quisermos executar entradas e saídas diretamente no Z80, poderemos utilizar a função INP ou o comando OUT do MSX-BASIC. Suas estruturas são:

```
INP(port)
OUT port,dado
```

Onde, "port" é o endereço da porta, e "dado" deve estar entre 0 e 255.

O comando WAIT espera que os dados de uma determinada porta atendam a certos requisitos (dificilmente é utilizado). Sua estrutura é:

```
WAIT port,andexp,xorexp
```

O MSX-BASIC vai ler os dados em "port"; quando o MSX-BASIC encontra os dados, compara-os com "xorexp" usando uma instrução XOR, em seguida compara o resultado com "andexp" usando uma instrução AND. Se o resultado é 0, o MSX-BASIC continua a esperar. O "xorexp" é opcional.

CAPÍTULO

15

AUXÍLIOS À PROGRAMAÇÃO

Este capítulo fornece diversos comandos e funções que vão ajudá-lo a escrever os programas, acelerando a localização de erros e permitindo uma utilização eficiente do MSX-BASIC.

Carregando o Programa

Para editar um programa, você já aprendeu como utilizar as teclas de controle do cursor e as teclas INS e DEL. Existem ainda cinco outras teclas que poderão ter utilidade na programação; elas estão relacionadas na Tabela 15.1.

Tabela 15.1 Teclas adicionais para edição.

Tecla	Função
CTRL+F	Deslocar o cursor para o início da próxima palavra.
CTRL+B	Deslocar o cursor para o início da palavra anterior.
CTRL+N	Deslocar o cursor até o fim da linha.
CTRL+E	Elimine até o fim da linha.
CTRL+U	Elimine toda linha.

A digitação do número das linhas poderá tornar-se cansativa, e para evitar isto o MSX-BASIC fornece a declaração AUTO. Com esta declaração o MSX-BASIC é instruído para iniciar cada linha com um novo número. A estrutura da declaração AUTO é:

```
AUTO inicio,incremento
```

Os dois argumentos "início" e "incremento" são opcionais. O argumento "início" informa ao MSX-BASIC em que linha iniciar a numeração, enquanto "incremento" indica os intervalos entre a numeração das linhas. O padrão dos dois argumentos é 10.

Para sair da numeração automática de linhas, acione CTRL+C ou CTRL+STOP. Caso já exista este número de linha visualizado na tela, o MSX-BASIC coloca um asterisco após o número. Para armazenar a linha existente e não permitir que ela seja sobreposta com outros caracteres basta pressionar a tecla RETURN. Quando você quiser escrever sobre a linha existente, basta digitar a nova linha com as instruções desejadas.

Manipulando Erros

O MSX-BASIC oferece diversas maneiras de tratar seus erros de programação. Em certos casos, você pode ter um interesse proposital na geração de erros para verificar o que está acontecendo no programa. Até o presente, ao ocorrer uma mensagem de erro, o programa MSX-BASIC simplesmente parava. O MSX-BASIC pode ser instruído a não parar ao ocorrer um erro, mas que vá para outra parte do programa.

O comando ON ERROR GOTO permite que ocorram erros no programa que o usuário nunca vai ver. Sua estrutura é:

```
ON ERROR GOTO linha
```

Onde "linha" é o número da linha em que se inicia a rotina de manipulação de erros. Uma vez executado este comando, todos os erros farão o MSX-BASIC ir até a linha especificada. Para eliminar este comando, utilize:

```
ON ERROR GOTO 0
```

A rotina de manipulação de erro poderá determinar o que houve de errado com as variáveis ERR e ERL. ERR contém o número do erro gerado, e ERL contém o número da linha em que ocorreu o erro. Para cada tipo de erro é associado um número; estes estão relacionados no fim deste capítulo.

A rotina de manipulação de erro deve terminar com o comando RESUME; opera da mesma forma que o comando RETURN. A estrutura do comando RESUME é:

```
RESUME NEXT
RESUME linha
RESUME 0
```

Com RESUME NEXT o programa continua a ser executado na linha que segue àquela em que ocorreu o erro (assim o comando ON ERROR GOTO opera da mesma forma que o comando GOSUB). RESUME com o argumento "linha" instrui o MSX-BASIC para que vá até aquela linha. RESUME 0 instrui o MSX-BASIC para que processe novamente a linha que causou o erro: isto pode ser perigoso, a não ser que você tenha certeza de que fez algo para corrigir o erro.

Por exemplo, se seu "Syntax error" favorito é o erro número 2, processe o seguinte programa:

```
10 REM  Procurando erros
20 ON ERROR GOTO 100
30 PRINT "Quase achando um erro"
40 SCREEM 2
50 PRINT "Este foi um grande erro"
60 END
100 REM  Rotina de procura de erro
110 PRINT "Voce achou um erro"; ERR; "na linha numero"; ERL
120 RESUME NEXT
```

Neste caso o usuário é informado de que havia um erro no programa que ainda está sendo processado. Freqüentemente não é o que se deseja, pois com toda certeza havia algo importante na linha que não foi executado.

Rotinas de erro eficientes podem tratar os erros com criatividade. Por exemplo, pode ser que o usuário tenha de digitar o nome do arquivo que deseja abrir. Se este arquivo não existir no disco, o MSX-BASIC dará o número de erro 53, "File not found". Em vez de parar o programa, poderá ser dado ao usuário outra oportunidade para fornecer um nome de arquivo, válido. O programa baixo imprime um arquivo na tela:

```
10 REM  Exemplo de RESUME 0
20 ON ERROR GOTO 100
30 LINE INPUT "Forneca o nome do arquivo que deseja ver"; A$
40 OPEN A$ FOR INPUT AS #1
50 IF EOF(1) THEN END
60 LINE INPUT #1, B$
70 PRINT B$
80 GOTO 50
100 REM Rotina de manipulacao de erro
110 LINE INPUT "Aquele arquivo nao foi encontrado; tente outro nome: "; C$
120 A$ = C$
130 RESUME 0
```

Quando o usuário digita o nome de um arquivo que não existe no disco, a linha 110 pede outro nome, e fixa A$ para este nome. Quando a linha 40 é executada novamente, A$ terá o novo nome.

O exemplo anterior apresenta um defeito: presume-se que o único erro será "file not found". Uma rotina de manipulação de erro deve ser mais poderosa, especialmente quando os dados de inserção são fornecidos pelo usuário. Uma vez que as rotinas de manipulação de erro não podem prever todos os casos, deve ser utilizada uma rotina generalizada que trata todos os erros que são conhecidos. Substitua as seguintes linhas nas linhas 100 até 130:

```
100 REM  Rotina de manipulacao de erro
110 IF ERR = 53 THEN GOTO 170
120 IF ERR = 56 THEN GOTO 200
130 REM  Nao reconheci este erro
140 PRINT "O MSX-BASIC tem o numero do erro"; ERR; "na linha" ERL
150 PRINT "Saindo para MSX-BASIC"
160 END
170 LINE INPUT "Aquele arquivo nao foi encontrado; tente outro nome: "; C$
180 A$ = C$
190 RESUME 0
200 LINE INPUT "Aquele nome de arquivo nao era correto; tente outro nome:"; C$
210 A$ = C$
220 RESUME 0
```

O MSX-BASIC permite que você gere suas próprias mensagens de erro de forma a acionar a função de localização de erro. O comando ERROR tem a seguinte estrutura:

```
ERROR n
```

Onde n pode ser qualquer número entre 0 e 255; observe que todos os erros de 72 até 225 apresentam a mensagem "Unprintable error", a não ser que você os prenda com o comando ON ERROR GOTO.

Como foi visto, o comando END encerra um programa. O comando STOP é exatamente como o comando END, exceto que permite ao MSX-BASIC imprimir a mensagem:

```
Break in nn
```

onde "nn" é o número da linha do comando STOP. O comando STOP não precisa de argumentos, de maneira que sua estrutura é simplesmente:

```
STOP
```

O comando STOP é utilizado na localização de problemas nos programas. Um programa que parou com um comando STOP, pode ser reiniciado digitando a declaração CONT. O programa será iniciado novamente a partir da linha que segue o STOP.

Quando houver necessidade de um método mais elaborado para o rastreio de problemas, podem ser utilizadas as declarações TRON e TROFF. A declaração TRON liga o rastreio em passos, ao passo que TROF o desliga. Enquanto o rastreio em passos estiver ativo, cada linha terá seu número impresso pelo MSX-BASIC à medida que ela é processada. Os números das linhas são impressos envoltos com colchetes:

```
[nnn]
```

Caso seu programa faça outras impressões na tela, provavelmente irá se perder com os números de linha. As declarações TRON e TROFF são úteis para seguir elos complexos do tipo FOR-

NEXT, bem como programas que têm muitos comandos GOTO e GOSUB. As declarações TRON e TROFF não precisam de argumentos, de modo que suas estruturas são:

```
TRON
TROFF
```

Por exemplo, insira o seguinte programa:

```
10 REM Exemplo TRON
20 PRINT "Como vai?"
30 FOR I = 1 TO 3
40 PRINT I
50 NEXT I
```

Sem o rastreio da linha, a saída é:

```
Como vai?
1
2
3
```

Dada a declaração TRON, e a saída será:

```
[10] [20] Como vai?
[30] [40] 1
[50] [40] 2
[50] [40] 3
[50]
```

Se aparecer o número de erro 7 ("Out of memory"), seu programa pode ser muito grande ou podem estar sendo solicitadas *séries*, ou matrizes muito grandes, em excesso (a capacidade de memória não está sendo suficiente para suas exigências). Para verificar a quantidade de memória livre em um programa, podemos utilizar a função FRE. Sua estrutura é:

```
FRE(numero)
```

O argumento "número" pode ser qualquer número. O valor da função FRE representa o número de bytes que sobram na memória. Se, em um programa extenso, este número é menor do que 1000, você pode ter certeza de que vai faltar memória. Pode ser necessário utilizar a seguinte rotina nos programas que têm problemas de memória:

```
...
50 ON ERROR GOTO 1000
...
400 REM Verificar memoria
410 IF FRE(0) < 2000 THEN ERROR 100
```

```
...
520 REM Verificar memoria
530 IF FRE(0) < 2000 THEN ERROR 100
...
1000 REM Tratamento de erros
1010 IF ERR = 100 GOTO 2000
...
2000 PRINT "Existe apenas"; FRE(0); "K sobrando na linha"; ERL
2010 RESUME NEXT
...
```

Impedindo a Parada do Usuário

Em certas situações deve-se impedir que o usuário possa interromper o programa com CTRL+STOP. Por exemplo, um programa abre um arquivo que posteriormente vai receber dados. Se o usuário interrompe o programa, o arquivo aberto poderá ser estragado. Para evitar isto, utilize os comandos ON STOP GOSUB e STOP ON para controlar a utilização da tecla CTRL+STOP.

O comando ON STOP GOSUB informa ao MSX-BASIC qual será a sub-rotina que ele deve executar ao ser pressionada a tecla CTRL+STOP. O comando STOP ON tem a estrutura:

```
STOP ON
STOP OFF
STOP STOP
```

O comando STOP ON informa ao MSX-BASIC para que inicie o bloqueio das operações da tecla CTRL+STOP; o comando STOP OFF informa ao MSX-BASIC para que retire o bloqueio. O comando STOP STOP temporariamente impede o bloqueio, mas mantém um registro se a tecla foi pressionada, de modo que o próximo comando STOP ON vai automaticamente à sub-rotina especificada.

As Principais Mensagens de Erro do MSX-BASIC

A lista que segue sobre as possíveis mensagens de erro do MSX-BASIC inclui uma pequena descrição de cada mensagem. O número do erro pode ser lido na variável ERR.

Número	Mensagem
1	**NEXT without FOR** Uma variável em uma declaração NEXT não corresponde a nenhuma variável de uma declaração FOR, sem correspondente, previamente executado.
2	**Syntax error** Foi encontrada uma linha que contém uma seqüência incorreta de caracteres (como um parênteses sem correspondência, um comando ou declaração errada ou pontuação incorreta).

3 **RETURN without GOSUB**
Foi encontrada uma declaração RETURN sem que tenha havido anteriormente a declaração GOSUB correspondente.

4 **Out of DATA**
Foi executada uma declaração READ quando não há nenhuma declaração DATA com os dados disponíveis no programa.

5 **Illegal function call**
Foi colocado para uma função *série* ou matemática, um parâmetro incorreto. Este erro pode ocorrer também como o resultado de um subscrito negativo ou muito grande, um argumento negativo ou nulo com a função LOG, um argumento negativo para a função SQR, uma mantissa negativa com um expoente não-inteiro, a chamada para uma função USR sem que tenha sido fornecido o endereço inicial, ou um argumento inadequado para um comando ou função.

6 **Overflow**
O resultado de um cálculo é grande demais para que possa ser representado no formato numérico do MSX-BASIC.

7 **Out of memory**
O programa é muito grande, tem muitos elos FOR ou GOSUB, ou tem muitas variáveis ou expressões complicadas.

8 **Undefined line number**
É feita referência para uma linha inexistente em uma declaração GOTO, GOSUB, IF/THEN/ELSE, ou DELETE.

9 **Subscript out of range**
O elemento de uma matriz foi referenciado com um índice que está fora da dimensão da matriz ou com o índice errado.

10 **Redimensioned array**
Foram dadas duas declarações DIM para a mesma matriz, ou foi dada uma declaração DIM para uma matriz após ter sido estabelecida para ela a dimensão 10 (padrão).

11 **Division by zero**
Em uma expressão é encontrada uma divisão por zero, ou um zero é elevado a uma potência negativa. Como resultado para a divisão é fornecida a representação infinita com o sinal do numerador, e a execução continua normalmente.

12 **Illegal direct**
Uma declaração ilegal em modo direto foi colocada como um comando em modo direto.

13 **Type mismatch**
Foi dado um valor numérico ao nome de uma variável *série* ou vice-versa, ou uma função que aguardava um argumento numérico recebeu um argumento *série* ou vice-versa.

14 **Out of string space**
As variáveis *série* excederam ao espaço disponível alocado. Utilize o comando CLEAR para alocar mais espaço para *série* ou então diminua o tamanho e número de *séries*.

15 **String to long**
Foi feita uma tentativa de criar uma *série* com mais de 255 caracteres.

16 **String formula too complex**
Uma expressão *série* é muito longa ou muito complexa. A expressão deve ser dividida em expressões menores.

17 **Cant'continue**
É feita uma tentativa de continuar um programa que foi interrompido devido a um erro, e que foi modificado durante a interrupção do processamento; ou que não existe.

18 **Undefined user function**
Foi solicitada uma função FN antes de ter sido dada a definição da função.

19 **Device I/O error**
Houve um erro de E/S em cassete, impressora ou em uma operação de vídeo. É um erro fatal, significando que o MSX-BASIC não consegue recuperar-se do erro.

20 **Verify error**
O programa atual é diferente daquele armazenado no cassete.

21 **No RESUME**
Não existe a declaração RESUME em uma rotina de erro.

22 **RESUME without error**
Foi encontrada uma declaração RESUME antes do fornecimento de uma rotina de erro.

23 **Unprintable error**
Não existe uma mensagem de erro para a condição de erro encontrada. Isto normalmente é causado por um ERROR com um código de erro indefinido.

24 **Missing operand**
Uma expressão contém um operador sem operando.

25 **Line buffer overflow**
Foi feita uma tentativa de entrar com uma linha com mais do que 255 caracteres.

50 **FIELD overflow**
Uma declaração FIELD está tentando alocar um número de bytes maior do que foi especificado para o comprimento do registro de um arquivo randômico.

51 **Internal error**
Houve um erro de funcionamento interno do MSX-BASIC.

52 **Bad file number**
Um comando ou uma declaração fazem referência a um arquivo com um número de arquivo que não está aberto ou que está fora da faixa dos números de arquivo especificado com o comando MAXFILES=.

53 **File not found**
Uma declaração LOAD, KILL ou OPEN fizeram referência a um arquivo que não existe no disco em questão.

54 **File already open**
Foi solicitado um comando OPEN para saída seqüencial, para um arquivo que já está aberto, ou foi dado o comando KILL para um arquivo aberto.

55 **Input past end**
É dada uma declaração INPUT para um arquivo nulo (vazio), ou é executado após a entrada de todos os dados do arquivo. Para evitar este erro, utilize a função EOF para detectar o fim do arquivo.

56 **Bad file name**
É utilizada uma forma ilegal para o nome do arquivo com LOAD, SAVE, KILL ou OPEN (por exemplo um nome do arquivo com excesso de caracteres).

57 **Direct statment in file**
Foi encontrada, com o comando LOAD, uma declaração direta em um arquivo com formatação ASCII. O comando LOAD terminou.

58 **Sequential I/O only**
Uma declaração PUT ou GET é utilizada para um arquivo seqüencial.

59 **File not OPEN**
O arquivo especificado em um comando de entrada ou saída de arquivo (como em PRINT# e INPUT#) não foi aberto.

60-255 **Unprintable error**
Estes códigos não têm mensagens.

FONTE: MSX Technical Reference Documentation 1984. Bellevue, Wash.: Microsoft Corporation, 1984.

CAPÍTULO

16

OUTROS COMANDOS E FUNÇÕES

Este capítulo explora os comandos e funções do MSX-BASIC não utilizados nos capítulos anteriores. Estão organizados por ordem de importância dos tópicos discutidos.

Dados em seus Programas

Quando utilizamos matrizes para guardar os dados gerados por um programa, poderá ser conveniente armazenar todos os dados que serão colocados em uma matriz em uma parte do programa. Os comandos READ e DATA freqüentemente são utilizados com o elo FOR-NEXT, para armazenar informação em uma matriz.

O comando DATA contém a informação que será lida pelo comando READ. A estrutura do comando DATA é:

DATA *elemento1, elemento2, ...*

Os elementos poderão ser constantes numéricas ou alfanuméricas. As *séries* deverão ou estar entre aspas ou então não deverão conter vírgulas, pontos de interrogação internamente, ou dois pontos. Os comandos de dados poderão conter um número variado de elementos, dependendo de suas referências.

O comando READ lê o próximo dado de uma declaração DATA e o armazena na variável que foi declarada. Sua estrutura é

READ *var1,var2, ...*

O argumento "var" pode ser qualquer tipo de variável, e a lista pode ter o comprimento necessário. Quando um comando READ é lido pelo MSX-BASIC, este procura o próximo dado no comando DATA, lê seu valor e o coloca na variável. Assim, o comando DATA é ignorado até o momento da execução de um comando READ.

Para entender melhor como os comandos READ e DATA operam, imagine que se deseja armazenar uma matriz de duas dimensões com cinco elementos, denominada BP. O programa pode ser como o descrito a seguir:

```
...
 30 DIM BP(4,1)
...
 80 FOR J = 0 TO 4
 90 READ BP(J,0)
100 READ BP(J,1)
110 NEXT J
...
430 DATA 12,43,7,90,16
440 DATA 29,3
450 DATA 67,19,22
...
```

Observe que os comandos DATA têm quantidades variáveis de dados. O MSX-BASIC simplesmente procura por um dado não lido para cada comando READ, de maneira que o comprimento dos comandos DATA é irrelevante.

O programa que segue executa a mesma função do anterior:

```
...
 30 DIM BP(4,1)
...
 80 BP(0,0) = 12
 90 BP(0,1) = 43
100 BP(1,0) = 7
110 BP(1,1) = 90
120 BP(2,0) = 16
130 BP(2,1) = 29
140 BP(3,0) = 3
150 BP(3,1) = 67
160 BP(4,0) = 19
170 BP(4,1) = 22
...
```

Observe a conveniência em se utilizar dos comandos READ e DATA.

O comando RESTORE permite ao MSX-BASIC recolocar o ponteiro no próximo dado. A estrutura do comando RESTORE é:

```
RESTORE linha
```

Onde *linha* é o número da linha em que o MSX-BASIC deve iniciar a leitura de dados. Se *linha* for omitido, o MSX-BASIC inicia a leitura a partir do início do programa. O comando RESTORE raramente é utilizado.

Organizando o Disco

Existem comandos fornecidos pelo MSX-BASIC que ajudam a organizar os arquivos no disco. Estes comandos, aqui resumidos, são similares aos comandos do MSX-DOS descritos no Capítulo 19. Se você não estiver familiarizado com o conceito de arquivos e discos, leia o Capítulo 18.

Para formatar um disco dê o comando CALL FORMAT. Você será instruído a colocar o disco no seu acionador e pode ser solicitado para fazer algumas escolhas. Uma vez que cada fabricante tem um método diferente para formatar o disco, leia a descrição do comando CALL FORMAT existente no manual que acompanha o seu acionador.

Para a obtenção de uma lista dos arquivos que existem no disco utilize o comando FILES. Sua estrutura é:

```
FILES nomearq
```

Lembre-se de que o MSX-BASIC lista todos os arquivos existentes no seu disco, se não for utilizado "nomearq", que poderá ser um determinado arquivo ou um arquivo com caracteres-chave (estes são descritos no Capítulo 18). O comando LFILES imprime, e uma impressora, a lista fornecida por FILES.

Um arquivo pode facilmente ter seu nome alterado com o comando NAME. Sua estrutura é:

```
NAME nomeant AS nomenovo
```

Onde "nomeant" é o nome atual do arquivo, e "nomenovo" é o que se deseja para o arquivo.

Conseguir uma cópia de um arquivo em disco também é fácil. A estrutura do comando COPY é:

```
COPY arq1 TO arq2
```

Onde "arq1" é o nome do arquivo que está sendo copiado e "arq2" é o nome do arquivo copiado.

Para eliminar um arquivo utilize o comando KILL. Sua estrutura é:

```
KILL nomearq
```

No "nomearq" podemos utilizar caracteres-chave. Esteja certo de não ter aberto um arquivo que esteja sendo cancelado, uma vez que neste caso o MSX-BASIC indica uma condição de erro.

É possível retornar ao MSX-DOS do MSX-BASIC com o comando CALL SYSTEM quando você tiver MSX-DOS em seu disco. Ao fazer isto, o programa existente na memória estará perdido, a menos que tenha sido salvo em disco ou em cassete; é o mesmo que desligar o computador MSX.

Fixar Parâmetros do MSX-BASIC

Existem dois comandos que fixam alguns parâmetros do MSX-BASIC: SCREEN e WIDTH. Já foram vistos alguns argumentos do comando SCREEN, como modo e tamanho do sprite. Os outros argumentos fixam parâmetros que nada têm a ver com a tela.

A estrutura completa do comando SCREEN é:

```
SCREEN modo,tamsprite,clictec,velfita,tipoimpr
```

Os argumentos são:

- A opção "clictec" liga ou desliga o som que se escuta no aparelho de televisão ou no monitor. A situação-padrão (clictec = 1) permite que seja escutado quando pressionam-se as teclas: para desligá-las coloque 0 como valor de clictec.
- A opção "velfita" informa ao MSX-BASIC se deve utilizar 1200 ou 2400 baud para leitura ou gravação no gravador cassete; o padrão (velfita = 1) é 1200 baud. Para ler e gravar com 2400 baud, coloque "velfita" em 2.
- A opção "tipoimp" informa ao MSX-BASIC se a impressora tem condição de imprimir todos os caracteres gráficos do MSX (poucas o fazem). O padrão (tipoimp = 1) informa que a impressora não imprime gráficos; se a sua impressora conseguir imprimir os gráficos MSX, coloque "tipoimp" em 0.

O comando WIDTH informa ao MSX-BASIC quantos caracteres devem ser impressos em uma linha da tela. Este comando não tem muita utilidade, uma vez que com o valor-padrão, o MSX-BASIC tenta imprimir a maior quantidade possível de caracteres. Diminuindo este valor significa apenas que podem ser impressos menos caracteres. Sua estrutura é:

```
WIDTH tamanho
```

Onde "tamanho" deve estar entre 1 e 40.

Teclas de Função

As teclas de função (F1 até F10) de seu computador MSX podem ser programadas para facilitar suas vidas. Em alguns computadores MSX, as teclas F5 até F10 não estão incluídas no teclado. Em seu lugar são utilizadas as teclas F1 até F5, com a tecla SHIFT para a obtenção das restantes. A Microsoft programou as teclas de função com comandos MSX-BASIC que podem ter utilidade para os programadores, mas você pode reprogramar estas teclas para qualquer função desejada.

Cada tecla poderá ser programada para conter um texto de até 16 caracteres. Para cada operação de uma tecla de função, o MSX-BASIC reage como se estes caracteres tivssem sido teclados por você. Assim, estas teclas poderão ser utilizadas na resposta aos comandos INPUT ou durante a escrita de um programa.

O comando KEY lista as definições atuais das teclas e define novas *séries* para teclas individuais. Sua estrutura é:

```
KEY LIST
KEY numtec,séries.
```

O comando KEY LIST lista as definições das dez teclas, na tela. Em KEY numtec,*série*, o "numtec" é o número da chave de função (de 1 até 10) e a *série* é a seqüência de caracteres que se deseja colocar na tecla.

Por exemplo, podemos ter um programa em que entramos com valores em cruzeiros. Caso "$0,00" seja um valor comum, F1 pode assumir esta *série*:

```
KEY 1,"$0,00"
```

O MSX-BASIC permite que seja utilizado qualquer caractere na *série*; por exemplo, havendo intenção de entrar com "$0,00", pressione a tecla RETURN. A tecla F1 pode ser alterada para:

```
KEY 1, "$0,00"+CHR$(13)
```

Independentemente do comprimento da *série* estabelecida para uma tecla, apenas os sete caracteres iniciais aparecerão na linha de indicação. Lembre-se de que esta linha pode ser desligada no final da tela com o comando KEY OFF e ligada novamente com o comando KEY ON.

O MSX-BASIC pode ser informado para executar uma determinada sub-rotina quando uma das teclas de função for operada pelo usuário. Esta condição será conseguida com os comandos ON KEY GOSUB e KEY n ON. A estrutura do comando ON KEY GOSUB é:

```
ON KEY GOSUB linha1,linha2,...
```

Onde "linha" são os números das linhas da sub-rotina que o MSX-BASIC irá executar ao se pressionar uma tecla. Para permitir qualquer enlace utilize KEY n ON:

```
KEY n ON
```

Onde n é a tecla de função que se quer utilizar.

Por exemplo, podemos adotar a tecla F1 como uma tecla universal de "auxílio" em um programa; cada vez que o usuário pressionar a tecla F1, uma mensagem informará sobre o que está ocorrendo. Podemos adotar a tecla F2 como uma tecla universal de "saída". O seu programa pode ficar tal como:

```
...
130 ON KEY GOSUB 1000,2000
140 KEY 1 ON
150 KEY 2 ON
...
990 END
```

```
1000 REM Sub-rotina de auxilio
1010 CLS: PRINT "Aqui esta um pouco de ajuda"
  ...
1160 REM Fim da sub-rotina
1170 RETURN
2000 REM Sub-rotina de saida
2010 CLS: PRINT "Passe muito bem..."
2020 CLOSE
2030 END
2040 REM Nao precisa de RETURN ja que
2050 REM o programa terminou.
```

Tempo de Enlace

Você já aprendeu como "prender" diversos acontecimentos, como a operação do gatilho de um joystick ou da tecla CTRL+STOP. Para que o MSX-BASIC vá até uma sub-rotina após um determinado tempo, utilize os comandos ON INTERVAL = GOSUB e INTERVAL ON. Estes operam da mesma forma que outros comandos de enlace. A estrutura do comando ON INTERVAL = GOSUB é:

```
ON INTERVAL = nGOSUB linha
```

Onde n é o número de intervalos de 1/60 do segundo, como na variável TIME, entre chamadas da sub-rotina da "linha". Após o comando ON INTERVAL GOSUB, dê o comando INTERVAL ON para ativar o enlace (ele pode ser desligado com INTERVAL OFF).

Por exemplo, pode existir uma indicação na parte superior da tela informando o tempo de jogo que resta a você. Para atualizar esta indicação a cada 10 segundos, poderia ser incluída a seguinte rotina em seu jogo:

```
  ...
300 REM T representa os segundos restantes
310 T = 100
320 ON INTERVAL = 600 GOSUB 1500
330 INTERVAL ON
  ...
1500 REM Sub-rotina para indicar tempo restante
1500 T = T - 10
1520 X0 = POS(0)
1530 Y0 = CSRLIN
1540 LOCATE 0,0
1550 PRINT "Tempo restante: "; T; "segundos"
1560 LOCATE X0,Y0
1570 RETURN
  ...
```

Já que é possível que o MSX-BASIC esteja imprimindo na tela, no momento da ocorrência deste evento, é importante que se retorne o cursor para a sua posição original ao terminar a sub-rotina.

Mudando Valores de Variáveis

Ocasionalmente poderá haver necessidade de comutar os valores de duas variáveis. Para fazê-lo, você deve utilizar o comando SWAP. Sua estrutura é:

```
SWAP var1,var2
```

Desta maneira var1 se iguala ao valor antigo de var2, e var2 se iguala ao valor anterior de var1.

Definindo suas Próprias Funções

O MSX-BASIC fornece uma facilidade que permite a você escrever suas próprias funções e utilizá-las em seus programas. Caso seja possível escrever sua função em uma única expressão matemática, poderá utilizar o comando DEF FN para criar uma função que se iguala com esta expressão.

A estrutura do comando DEF FN é:

```
DEF FNvar(p1,p2,...) = expressao
```

Onde o argumento "var" é o nome da função. Os eventuais parâmetros existentes são incluídos como p1, p2 etc. Já que o MSX-BASIC apenas lembra os tipos de parâmetros, e não os nomes, estes não são importantes. A "expressão" é qualquer expressão MSX-BASIC válida.

Por exemplo, podemos definir uma função que determina o seno hiperbólico de um número (caso você se lembrasse desta equação, seria motivo de orgulho de seu professor de matemática). Para sua definição, você poderia utilizar o comando:

```
30 DEF FNSH(X) = (EXP(X) - EXP(-X))/2
```

O X não tem importância; a única lembrança do MSX-BASIC é a de que FNSH é uma função que tem argumento real de dupla precisão. Podemos agora utilizar a função FNSH da mesma forma que outras do MSX-BASIC:

```
90 G = FNSH(3.1)
```

A principal limitação do comando DEF FN é a de que a função deve ser definida como uma única função. Para funções simples isto é ótimo, mas normalmente isto impede que sejam definidas funções complexas.

O Comando CALL

No caso de seu computador MSX possuir programas especiais escritos em ROM, estes provavelmente serão acessados com o comando CALL. (Estes programas serão documentados no manual que acompanhar o computador.) A estrutura do comando CALL é:

```
CALL rotina(param1,param2,...)
```

Onde "rotina" é o nome do programa na ROM; quando os parâmetros forem necessários deverão ser especificados no parênteses. Em vez da palavra "CALL" poderá ser utilizado o sinal "_":

```
_rotina(param1,param2,...)
```

Por exemplo, o CX5M da Yamaha contém um sintetizador de música que pode ser controlado pelo programa MUSIC. Para que este programa seja processado a partir do MSX-BASIC, dê o comando:

```
CALL MUSIC
```

ou

```
_MUSIC
```

Misturando Programas

É pouco provável que você irá unir dois programas, uma vez que a numeração da linhas possivelmente entrará em choque. O comando MERGE tem entretanto esta função. Sua estrutura é:

```
MERGE nomearq
```

Desta maneira nomearq é colocado na memória junto com o programa atual.

Limpando a Memória

Durante a execução de um programa complicado em MSX-BASIC, poderá haver falta de memória. O MSX-BASIC fornece duas maneiras de manipular a memória disponível: os comandos CLEAR e ERASE, que raramente são utilizados.

O comando CLEAR elimina todas as variáveis da memória e permite que o espaço seja utilizado para *séries*. Isto, sem dúvida, somente terá utilidade no início do programa, antes da existência de variáveis. A estrutura é:

```
CLEAR tamsérie
```

O valor-padrão de "tam*série*" (que representa a quantidade de RAM disponível para variáveis *série*) é 200 bytes. O comando CLEAR também fecha todos os arquivos. O comando ERASE libera o espaço utilizado por uma matriz e faz o MSX-BASIC comprimir a memória livre. A estrutura do comando ERASE é:

```
ERASE matriz1,matriz2,...
```

Cada "matriz" é o nome da matriz que se quer eliminar da memória. Após a utilização de ERASE, a matriz poderá ser novamente dimensionada com o comando DIM.

Manipulação Avançada de Vídeo

Quando estiver familiarizado com o processador de vídeo (VDP) TI9918A, você poderá utilizar os comandos VDP, VPEEK, VPOKE e BASE para alterar a RAM utilizada pelo integrado.

A função VDP permite que você olhe e altere os registradores do integrado VDP. Ao estabelecer um registro, sua estrutura é:

```
VDP(reg) = valor
```

Onde "reg" é o número do registrador, que varia de 0 até 7; o "valor" pode ser de 0 até 255. Para ler o conteúdo do registrador utilize a função:

```
VDP(reg)
```

Já que cada bit do registrador é significativo, tenha certeza de seu significado antes de alterá-los.

A função VPEEK e o comando VPOKE são muito parecidos com PEEK e POKE, exceto que operam sobre os 16K de RAM do vídeo. Esta é a única maneira de acessar o RAM do vídeo. Esta RAM mantém, além dos valores das retículas e caracteres da tela, a informação dos sprites e cores utilizadas. Os endereços dos VRAM vão de 0 até 16383.

A função BASE permite que sejam alterados os endereços básicos utilizados pelo integrado VDP ao colocar texto e gráficos na tela. Somente tem utilidade para programadores experientes que queiram armazenar diversas imagens simultaneamente na VRAM.

Para estabelecer um registro, utilize a estrutura:

```
BASE(entrtab) = endvram
```

Os valores de "entrtab" estão indicados na Tabela 16.1. O "endvram" representa o endereço da VRAM que você deseja alterar. Para ler o valor atual da tabela, utilize:

```
BASE(entrtab)
```

Tabela 16.1 Parâmetros para a função BASE.

Parâmetro	Significado
0	Base da tabela de nome
1	Base da tabela de cor
2	Base da tabela de gerador de figuras
3	Base da tabela do atributo sprite
4	Base da tabela do padrão sprite

Para você poder utilizar a função BASE, deverá determinar o modo de tela e seu número, multiplicá-lo por 5, e somar o resultado ao parâmetro da Tabela 16.1. Assim, para utilizar C2 como o valor do endereço básico da tabela do atributo sprite para modo 2, utilize o comando:

```
C2 = BASE((2*5)+3)
```

Armazenando Imagens Binárias

É possível carregar ou salvar uma cópia do RAM em disco ou cassete, com os comandos BLOAD e BSAVE. Além de salvar um programa em linguagem assembly ou uma cópia de parte da BRAM, existem poucas utilidades para estes comandos.

A estrutura do comando BSAVE é:

```
BSAVEnomarq,endinic,endfin
```

Onde "nomarq" é o nome do arquivo em que se deseja armazenar a imagem binária. O "endenic" e "endfin" correspondem aos endereços iniciais e finais da RAM que se deseja salvar. Estes dois endereços são armazenados com o arquivo.

A estrutura do comando BLOAD é:

```
BLOADnomarq,,desloc
```

Quando é incluído um endereço "desloc", este é acrescentado ao endereço inicial e final armazenado no arquivo; desta maneira é possível carregar um bloco de memória para uma região diferente da memória que aquela de onde fora retirada para ser salva no cassete (ou disco).

Caso você deseje salvar uma imagem da VRAM, a opção S deve ser utilizada. As estruturas neste caso ficam:

```
BSAVEnomarq,endinic,endfin,S
BLOADnomarq,S,desloc
```

PARTE

3

Aprendendo MSX-DOS

CAPÍTULO

17

DESCRIÇÃO DO MSX-DOS

Se você tiver um acionador de disco e 64K de RAM, você pode usar o MSX-DOS, que nada mais é do que o sistema operacional para disco de seu computador MSX. O MSX-DOS se torna necessário quando se deseja processar programas aplicativos externos ao MSX-BASIC, sendo desnecessário se a finalidade for a de operar o disco.

Grande parte dos programas aplicativos que são adquiridos para o computador MSX presume que você conheça os comandos do MSX-DOS, de modo que não fornece as instruções necessárias para operar o programa. Portanto, você poderá descobrir que algumas das tarefas que quiser realizar (como a formatação de um disco novo) não são executadas pelo seu programa aplicativo. Poderão ser feitas com MSX-DOS ou, em alguns casos, com os comandos de disco do MSX-BASIC.

O que Faz o MSX-DOS

Podemos imaginar que um sistema operacional seja similar à cabina de comando de uma aeronave. Sem ela, o potencial é grande, mas não poderá voar com o avião. O sistema operacional permite que você (o piloto) controle o computador informando-o para onde ir e o que fazer. O sistema operacional coordena as partes do computador fornecendo-lhe uma maneira simples de controlá-lo. Neste capítulo você vai começar a aprender como o MSX-DOS executa seu papel.

O MSX-DOS é uma extensão do sistema operacional do seu MSX; permite ao sistema operacional do MSX controlar os acionadores de disco. O MSX-DOS, entretanto, faz muito mais do que isto; fornece uma maneira de informar ao computador qual é o programa ou comando que você está querendo executar, onde este programa ou comando serão encontrados e o que fazer com ele.

A função operacional principal do MSX-DOS é utilitária. Nesta situação o MSX-DOS executa comandos, que permitem sua interação direta com o computador. As funções executadas por estes comandos são: nomeare, arquivos do disco ou copiar arquivos de um disco para outro, entre outros.

O MSX-DOS trata seus próprios comandos como programas aplicativos. Estes programas, porém, são mais limitados que a maioria dos programas aplicativos. Não executam tarefas do tipo processamento de palavras ou contabilidade; em seu lugar preenchem as funções de manutenção do computador. Todo comando tem um nome facilmente lembrado. Por exemplo, para copiar as informações de um disco para outro, é utilizado o comando COPY. Todos estes comandos são discutidos no Capítulo 19.

Olhando a Parte Interna do MSX-DOS

A maioria das pessoas opera com os computadores MSX-DOS sem nada saber sobre o que o MSX-DOS está fazendo. Conhecer um pouco a maneira como ele trabalha poderá ajudá-lo na utilização mais efetiva de seu sistema operacional. Poderá ajudá-lo, ainda, no estabelecimento dos limites sobre o que esperar do MSX-DOS.

Se fosse possível olhar para dentro do MSX-DOS, o que veríamos seria um conjunto muito complicado de instruções de computador. Estas instruções estão descritas em linguagem assembly. Felizmente não será necessário conhecê-la para utilizar o MSX-DOS. Nem é preciso saber como o MSX-DOS executa seu trabalho. Entretanto, o processo pelo qual o MSX-DOS processa seus programas não é difícil de entender, e é útil conhecer algo a seu respeito, especialmente quando os comandos são dados diretamente ao MSX-DOS.

COMO O MSX-DOS PROCESSA OS COMANDOS

Os processos discutidos nesta seção relacionam-se com a função utilitária do MSX-DOS; como estes utilitários operam você vai aprender nos dois próximos capítulos. O MSX-DOS é como um programa que está sempre operando. Quando você liga o computador o MSX-DOS é passado do disco para a memória RAM e começa a operar (uma descrição completa deste procedimento é fornecida no Capítulo 18). Quando o MSX-DOS estiver pronto para receber o primeiro comando ou processar um programa, coloca uma mensagem na tela e aguarda suas instruções.

Esta indicação geralmente é "A>", informando que o interpretador de comandos do MSX-DOS está pronto para a próxima instrução. Este interpretador lê seu comando, encontra o programa certo, e inicia seu processamento. (No restante desta seção será utilizada a palavra "comando"; este processo também se aplica para todos os programas processados pelo MSX-DOS.)

Para processar um comando basta digitar seu nome (e argumentos) pelo teclado. O MSX-DOS mostra os caracteres na tela, à medida que são digitados. Em seguida pressione a tecla RETURN da mesma forma que no MSX-BASIC.

Após informar ao MSX-DOS o nome do comando, o sistema operacional deverá encontrar o programa do comando. Deve olhar em dois locais: um comando poderá estar armazenado interna ou externamente. Os comandos internos são parte integrante do MSX-DOS. Significa que eles não precisam ser procurados pelo MSX-DOS em um disco, já que foram carregados na RAM com o restante do MSX-DOS. Os programas aplicativos são externos. Sempre que estes programas são processados, deverão ser lidos do disco pelo MSX-DOS antes que possam ser

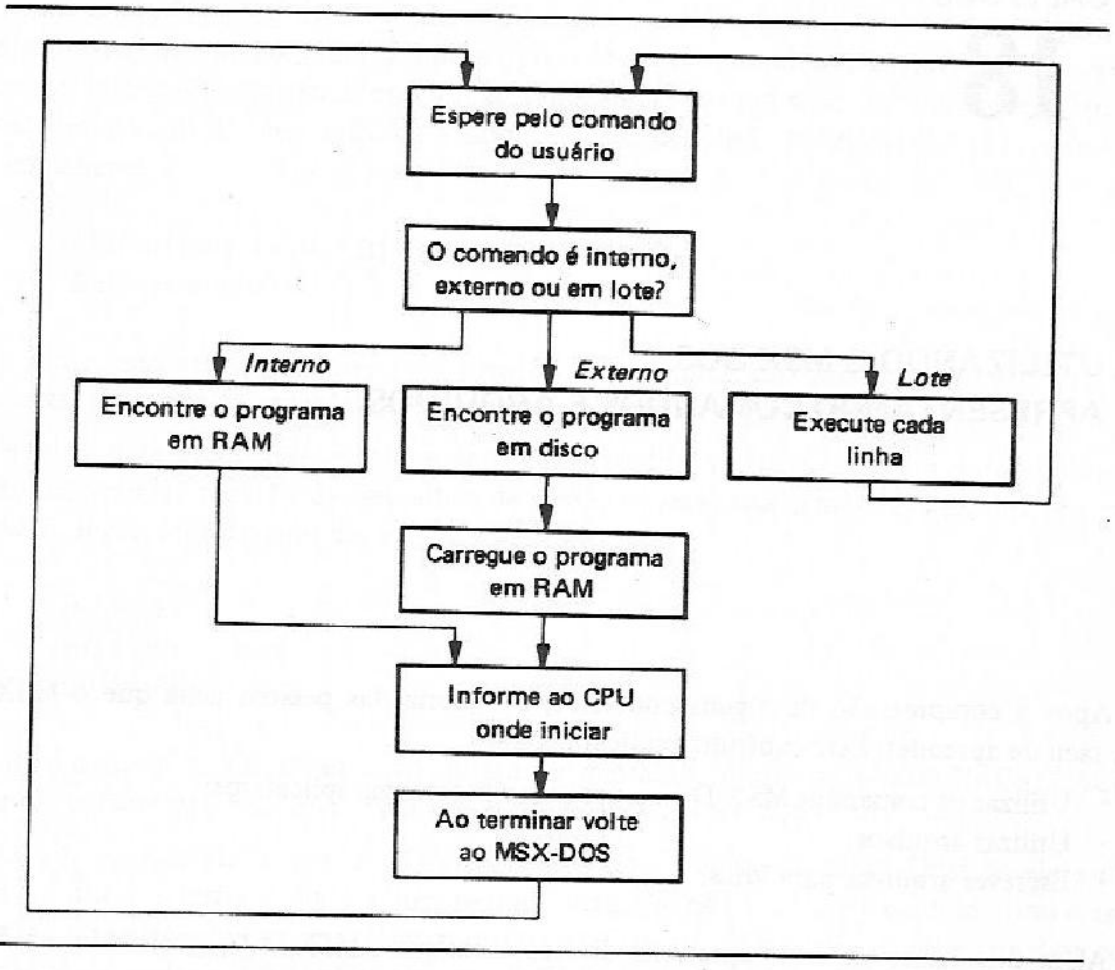


Figura 17.1 Como o MSX-DOS processa um programa.

executados. Por exemplo, caso você tenha um programa de processamento de palavras que não esteja em ROM, trata-se de um programa externo.

Arquivo em lotes é um tipo especial de comando externo formado por um conjunto de comandos MSX-DOS (discutidos em detalhe no Capítulo 18). A Figura 17.1 mostra os passos seguidos pelo sistema operacional quando você o instrui para processar um comando.

Como não há diferença na maneira com que você informa ao MSX-DOS para processar comandos externos ou internos, não é necessário (de modo geral) que você saiba que tipo de comando está pedindo ao MSX-DOS para executar. Por enquanto podemos ignorar as diferenças, porém mais adiante você verá porque é necessário saber quando um comando é externo ou interno.

Quando processamos um comando externo, deve existir a certeza de que o disco que contém o programa do comando está colocado no acionador. Caso não esteja no acionador (ou caso tenha erro na digitação do nome de comando), o MSX-DOS emitirá uma mensagem de erro "Bad comand or file name" (Erro de comando ou no nome do arquivo).

O Capítulo 18 descreve como utilizar o MSX-DOS, e o Capítulo 19 descreve todos os comandos MSX-DOS.

CAPÍTULO

18

UTILIZANDO O MSX-DOS APRESENTANDO COMANDOS E ARQUIVOS

Após a compreensão de alguns conceitos, a maioria das pessoas acha que o MSX-DOS é muito fácil de aprender. Este capítulo explicará como:

- Utilizar os comandos MSX-DOS e processar programas aplicativos;
- Utilizar arquivos;
- Escrever arquivos para lotes.

Além disto, este capítulo apresenta diversos comandos MSX-DOS mostrando, ainda, como utilizá-los para a realização de tarefas simples. Não procura apresentar todos os comandos do MSX-DOS nem todas as utilizações dos comandos discutidos aqui. Para este tipo de informação, veja o Capítulo 19, "Os comandos MSX-DOS", onde são discutidos todos os comandos do MSX-DOS, em profundidade, incluindo tanto sua utilização comum como sua utilização não-comum. Poderá ser utilizado como referência para dúvidas a respeito de qualquer comando.

A maioria dos comandos apresentados neste capítulo vem acompanhada de exemplos. Experimente os comandos dos exemplos em seu computador MSX. Os mais simples serão discutidos primeiro. Já que a maioria dos comandos do MSX-DOS tem nomes relacionados com a ação que realizam deve ser fácil lembrar a utilização de cada um deles. Se houver necessidade de uma referência posterior a eles, lembre-se de que são explicados em detalhe no Capítulo 19.

Inicializando o MSX-DOS

Siga as instruções que acompanham o acionador de disco, para carregar o MSX-DOS. Certifique-se de que o disco esteja firmemente colocado em seu lugar (a menos que no seu computador MSX ele faça parte integrante do aparelho). Quando seu acionador apresentar uma chave liga/desliga, ligue primeiro esta chave e depois o computador MSX.

Assim que o MSX-DOS estiver carregado, será apresentada na tela a indicação com o número de versão do MSX-DOS e a declaração de direitos autorais da Microsoft, seguida pela solicitação

"Enter new date: "Isto acontece a não ser que seu computador MSX tenha um relógio interno ou que seu disco de sistema contenha um arquivo especial denominado arquivo AUTOEXEC.BAT que, ao ser carregado o sistema, executa automaticamente uma série de comandos escritos nele. Ao final deste capítulo este arquivo é discutido em detalhes. A orientação para colocação da data é semelhante a:

```
Current date is Sun 1-01-1984
Enter new date: ■
```

O bloco que aparece à direita da orientação é denominado cursor, como no MSX-BASIC. Observe que o cursor está aguardando após os dois pontos na segunda linha.

Digite a data como três números, separados por hífen (-) ou barra (/). Em seguida indicaremos formas válidas para 14 de Setembro de 1985. Observe que o mês deve sempre ser colocado em primeiro lugar, seguido pelo dia e depois pelo ano.

```
9-14-85
9/14-/85
09/14/85
```

Na eventualidade de cometer um erro durante a digitação, pressione a tecla BACKSPACE para retornar aos caracteres errados. Ao terminar pressione a tecla RETURN.

Qual é a necessidade que o MSX-DOS tem de conhecer a data? Quando ele armazena arquivos no disco, registra o dia e a hora permitindo assim que você saiba quando criou o arquivo. Entretanto, o registro da data e do horário não é obrigatório. Se não quiser colocar a data correta, basta pressionar a tecla RETURN para que o MSX-DOS presuma que a data seja Janeiro, dia 1 de 1984 (ou qualquer que seja a data indicada no vídeo). Após ter dado entrada à data aparece na tela o sinal A>.

```
A>■
```

A indicação-padrão, A>, será vista após ter informado ao MSX-DOS a data e a hora. A> é a maneira com que o MSX-DOS solicita que você lhe dé um comando. Ao ver o A>, saberá que não há outro programa em processamento.

O que fazer ao ver o Indicador A>

Uma vez que o sinal A> aparece em seu vídeo, o próximo passo é fornecer um comando ao MSX-DOS. Normalmente, para dar um comando ou processar um programa, deve-se digitar o nome do comando ou do programa e pressionar a tecla RETURN. (Existem outras informações que podem ser digitadas após o nome de um comando; estas são descritas mais adiante neste capítulo.) Pode ser que o comando DIR seja o primeiro comando que você queira dar: fornece uma listagem de todos os arquivos que estão em seu disco. O exemplo abaixo mostra como é que isto deve ser feito. (O modo de ilustrar os exemplos, em forma de quadros, será utilizado para apresentar os comandos ou procedimentos discutidos no texto.)

Ao ver o sinal A>, digite DIR. A entrada que é dada deve ser algo semelhante a:

```
A> DIR
```

Observe que aquilo que você digita é sublinhado na ilustração. Em todo o capítulo, o que você digita será sublinhado de modo que seja facilmente distinguido das mensagens geradas pelos MSX-DOS.

Em seguida pressione a tecla RETURN. O comando DIR percorre e digita uma lista de todos os arquivos no acionador A. Você acabou de dar seu primeiro comando MSX-DOS.

Enquanto você digita os comandos, poderá utilizar algumas das teclas especiais que existem no teclado, para ajudá-lo. Poderá utilizar a tecla BACKSPACE para correção de erros (como foi feito no MSX-BASIC). Para eliminar uma linha inteira, pressione a tecla denominada ESC ou ESCAPE.

Na eventualidade de ocorrer uma mensagem de erro do tipo "Bad command or file name" durante a tentativa de processar um comando, não se preocupe. O erro mais comum é a digitação errada do nome do programa.

O ACIONADOR "ASSUMIDO" E COMANDOS EXTERNOS

Como você já viu, a indicação do MSX-DOS lhe diz que o sistema operacional está aguardando um comando. A indicação diz algo mais. Diz para qual acionador o MSX-DOS se dirige automaticamente ao procurar comandos e arquivos. Este acionador é chamado "acionador assumido". Significa alguma coisa presumida caso não haja especificação em contrário.

Como a maioria dos computadores tem apenas um acionador de disco, não precisa se preocupar como utilizar o segundo. Mesmo que você tenha apenas um acionador de disco, será demonstrado mais adiante como fazer para que o MSX-DOS pense que existem dois.

Quando o MSX-DOS lhe dá a indicação A>, o acionador assumido é A:. Quando indica B> ou C>, informando-lhe que o acionador assumido é B: ou C: (as letras dos discos são normalmente seguidas por dois pontos para distingui-las do nome de arquivos).

Você poderá querer saber porque as letras A: e B:. Seu computador poderá ter vários acionadores de disco ou apenas um. Os discos são nomeados com letras, iniciando com A: representando o primeiro disco do computador, B: representando o segundo etc. O manual de seu computador deve explicar como os discos são identificados. Caso você tenha um disco rígido, ele poderá ser tratado pelo MSX-DOS como se estivesse dividido em discos menores, cada um com uma letra própria.

Não é importante o conhecimento de qual acionador contém o disco de sistema do MSX-DOS, quando se digita um comando do tipo DIR. Entretanto, faz diferença quando é dado um comando externo, uma vez que o MSX-DOS deve ler o comando externo do disco de sistema antes de executá-lo. Normalmente o MSX-DOS olha para o acionador assumido, na tentativa de ler um comando.

Por exemplo, ao listar o diretório de seu disco de sistema, você deve ter visto um arquivo denominado MBASIC.COM. – comando externo que permite que sejam feitos programas em um dialeto denominado MBASIC (dificilmente será utilizado já que o MSX-BASIC é muito mais poderoso). Para executar um comando externo, basta digitá-lo da mesma forma que se digita um comando interno.

Por exemplo, para dar início ao MBASIC, dê o comando:

```
A> MBASIC
```

Podemos escutar o zumbido do acionador enquanto lê o programa MBASIC do disco. O MSX-DOS carrega o programa MBASIC e inicia seu processamento..

Para sair do programa MBASIC, digite o comando SYSTEM e pressione a tecla RETURN. Novamente aparece A>.

Quando você informa ao MSX-DOS para processar um comando externo, este olha apenas para o disco do acionador assumido, a não ser que receba uma informação diferente. Caso os comandos que devam ser processados estejam no disco B:, pode haver interesse em alterar o acionador assumido para B:. Esta alteração pode ser feita pela digitação de outra letra seguida por dois pontos e depois pressionando a tecla RETURN.

Por exemplo, para alterar de A: para B: o acionador assumido, dê o comando:

```
A> B:
B> ■
```

A indicação B> mostra que o acionador assumido foi alterado para B:.

Se você tiver apenas um acionador, o MSX-DOS chama-o de A:. Mesmo que você se refira a B: isto não confundirá o MSX-DOS. Ele sabe que você quer pensar nele como se fosse B:. Quando você se refere a uma letra de disco diferente daquela que o MSX-DOS acredita estar no acionador, o MSX-DOS pede que um novo disco seja colocado no acionador. Por exemplo, caso o MSX-DOS pense que o acionador atual é A:, e você solicita um arquivo de B:, dará a seguinte mensagem:

```
Insert diskette for drive B: and strike
any key when ready
■
```

Retire o disco que está no acionador, insira aquele que você quer acessar e pressione qualquer tecla. Agora o MSX-DOS pensa no acionador como tendo a letra que foi especificada.

Caso o programa não esteja no acionador assumido e você não especificou o disco que deve ser procurado, será impresso na tela (pelo MSX-DOS) a mensagem "Bad command or file name".

Uma maneira de processar um comando de um disco diferente é alterando o acionador assumido (o que se pode tornar um transtorno se feito com freqüência). Felizmente, existe uma maneira mais fácil de informar ao MSX-DOS onde procurar. Basta dar o nome do acionador de disco antes do programa.

Por exemplo, caso o acionador assumido seja B:, e o MBASIC.COM está em A:, dê o comando:

```
B>A:MBASIC
```

Assim o MSX-DOS é informado de que não deve olhar no acionador assumido, mas apenas o A:. É mais fácil do que alterar o acionador assumido como no exemplo seguinte:

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

Fornecendo Argumentos aos Comandos

Os comandos do MSX-DOS utilizados até agora foram bem simples. É possível fornecer maiores informações ao MSX-DOS quando são dados comandos que permitam ao MSX-DOS executar tarefas mais complexas.

Um maior número de informações pode ser fornecido ao MSX-DOS pela utilização de argumentos – da mesma forma que os argumentos do MSX-BASIC. Quase todos os comandos do MSX-DOS têm argumentos necessários ou opcionais. Você terá um computador mais eficiente se souber utilizar os argumentos.

Por exemplo, você já viu como o comando DIR imprimiu uma lista de arquivos existentes no disco:

```
A>DIR
```

O comando DIR, sem argumentos, lista os arquivos existentes no acionador assumido, que no caso é A:. É possível fornecer ao comando um argumento opcional que define de qual acionador desejamos ver o diretório. Por exemplo, para ver os arquivos no acionador B:, digite:

```
A>DIR B:
```

Neste caso B: é um argumento da instrução DIR que instrui o comando a mostrar os arquivos de B:, e não aqueles do acionador assumido.

O comando DIR tem um argumento opcional uma vez que pode operar com ou sem ele. Alguns comandos, entretanto, têm argumentos solicitados porque precisam saber no que operar; é o mesmo conceito aprendido nos argumentos do MSX-BASIC.

Por exemplo, o comando TYPE do MSX-DOS, que apresenta o conteúdo de um arquivo na tela, exige que você dê o nome do arquivo, cujo conteúdo queremos ver. Vamos supor que exista interesse em ver o texto de um arquivo chamado ACCNTS.RPT. Deveria dar o comando:

```
A> TYPE ACCNTS.RPT
```

O conteúdo de ACCNTS.PRT será impresso na tela. Se tivesse sido digitado apenas TYPE sem argumento, o MSX-DOS teria respondido com uma mensagem de erro "Invalid number of parameters".

Nem todos os arquivos podem ser vistos de forma legível pela utilização do comando TYPE. Caso você faça a tentativa de digitar um arquivo que não esteja na forma ASCII, seu computador apresenta uma série de caracteres ilegíveis na tela.

Existem diversos tipos de argumentos que podem ser fornecidos aos comandos; os mais comuns são nomes de acionadores de disco ou arquivos. Nos dois exemplos anteriores, o comando DIR levou o nome de um arquivo B:, enquanto o comando TYPE levou o nome de um arquivo (ACCNTS.RPT). Outros tipos de argumentos são explicados no Capítulo 19; a seção "Como ler a sintaxe de comando" explica a apresentação dos argumentos.

Alguns Comandos Simples: FORMAT e COPY

Agora que você viu como iniciar o MSX-DOS e dar-lhe comandos, já está em condição de utilizar o seu computador de maneira prática. A primeira tarefa que deve ser aprendida é como preparar e copiar seus discos. Os comandos MSX-DOS serão utilizados um pouco mais nesta seção.

PREPARANDO O DISCO COM O COMANDO FORMAT

A maioria dos discos quando é comprada não está pronta para ser utilizada. Deve-se inicialmente "formatá-los". O processo de formatação significa colocar marcas magnéticas no disco, de maneira que o MSX-DOS saiba como encontrar o local certo para a colocação dos dados no disco.

Formatar um disco é semelhante à pintura das faixas brancas em um estacionamento. O padrão de formatação no disco informa onde o MSX-DOS deve colocar os dados. Quando o estacionamento é novo, não haverá linhas anteriores: entretanto, se houverem linhas antigas, todo o estacionamento terá, primeiro, de ser pintado de preto, de modo que as linhas novas se destaquem. Pintando o estacionamento de preto, ou formatando o disco, toda a "informação" anterior se perde.

Portanto, quando um disco já utilizado é formatado, toda informação existente é destruída. Ao utilizar o comando FORMAT, deve-se tomar o cuidado de não utilizar um disco que tenha informações que serão usadas no futuro. Antes de formatar um disco utilizado anteriormente verifique o seu conteúdo com o comando DIR. Os discos rígidos normalmente já estão formatados; é pouco provável que você algum dia precise formatar um disco rígido.

O comando FORMAT leva como argumento o nome do acionador que vai ser formatado. É idêntico ao comando CALL FORMAT do MSX-BASIC.

Por exemplo, para formatar um disquete no acionador B:, dê o comando:

```
A> FORMAT B:
```

Quando o MSX-DOS solicita o acionador que deve ser formatado, responda B:. Em seguida ele vai pedir que você coloque um disquete no acionador B:; coloque o disquete e pressione qualquer tecla. O disco no acionador B: será formatado.

Copiando Arquivos com o Comando COPY

O comando COPY permite que um arquivo seja copiado de um disquete para outro. O comando COPY tem no mínimo dois argumentos: o nome do arquivo a ser copiado e o do acionador em que este arquivo será copiado. (Na verdade, o comando COPY pode fazer muito mais, mas suas outras funções são descritas em detalhes no Capítulo 19.) O comando COPY do MSX-DOS é semelhante ao comando COPY do MSX-BASIC.

Por exemplo, se é dado o comando:

```
A> COPY MBASIC.COM B:
```

O MSX-DOS vai copiar o arquivo MBASIC.COM do disco A: para o disco B:.

UTILIZANDO FORMAT E COPY PARA FAZER UMA RÉPLICA DE SEU DISQUETE DE SISTEMA

Agora que conhecemos os comandos FORMAT e COPY, devemos utilizá-los de forma adequada. Nesta seção faremos uma cópia duplicada do disco de sistema do MSX-DOS (o disquete utilizado para dar início ao MSX-DOS).

Por que fazer uma cópia do disquete de sistema? Não é aconselhável confiar em um único disquete utilizado diariamente. Muitos acidentes podem acontecer: uma eliminação acidental, café derramado, ou mesmo a falta de energia elétrica durante uma leitura ou uma gravação no disco poderá danificar o disquete e/ou os arquivos que ele contém.

Para evitar uma catástrofe, deveremos fazer uma cópia deste disquete e utilizar a cópia em vez do original. Caso aconteça algo com a cópia, basta fazer outra. Os comandos FORMAT e COPY permitem que você faça cópias de seu disquete de sistema MSX-DOS.

Uma vez que um novo disquete foi formatado, o próximo passo é copiar todos os arquivos do disquete de sistema.

Por exemplo, caso você tenha um computador com dois discos, inicialmente coloque o disquete do sistema no acionador A:. Em seguida dê o comando

```
A> FORMAT
```

e responda à solicitação. O disquete ainda não tem como conteúdo os arquivos MSX-DOS indicados na listagem do diretório. Para copiar estes arquivos, dê o comando:

```
A> COPY A: * : * B:
```

Se tiver apenas um acionador, terá de mudar os disquetes enquanto o MSX-DOS copia os arquivos. O disquete no acionador B: agora está em condição que permite iniciar o computador. O *.* utilizado no comando COPY é explicado posteriormente. Informa ao MSX-DOS para copiar todos os arquivos que estão em A:, independentemente de seus nomes.

PROTEGENDO O DISQUETE DE SISTEMA CONTRA GRAVAÇÃO

Agora que foi feita uma duplicação do disquete de sistema, devemos protegê-lo contra rasuras acidentais ou alterações indesejáveis. Isto pode ser feito protegendo-o contra gravações.

Se seu computador utiliza disquetes de 5 1/4", você já deve ter observado o entalhe existente em seu lado direito, a aproximadamente 2,5 cm de sua parte superior. É o que se denomina "entalhe de proteção contra gravação escrita" (indicado na Figura 18.1). Quando este entalhe é coberto, o MSX-DOS protege o disco, não permitindo que escrevamos nele. No entanto, o MSX-DOS continua permitindo que ele seja lido.

Quando seu computador utiliza disquetes de 3 1/2", o pequeno furo no canto superior direito é utilizado para proteger o disco contra gravação. O método utilizado para proteger este disco depende do tipo de disco adquirido. Alguns têm uma peça de plástico que é deslocada para revelar o furo, enquanto outros vêm com uma aba que deve ser quebrada para revelar o furo.

É de boa prática proteger os disquetes, tanto os discos que acompanham o computador como as cópias do disco de sistema. É claro que se pode retirar a proteção e escrever no disco caso seja necessário. A principal razão de se proteger um disco contra gravação é a de diminuir a chance de destruir acidentalmente informações importantes.

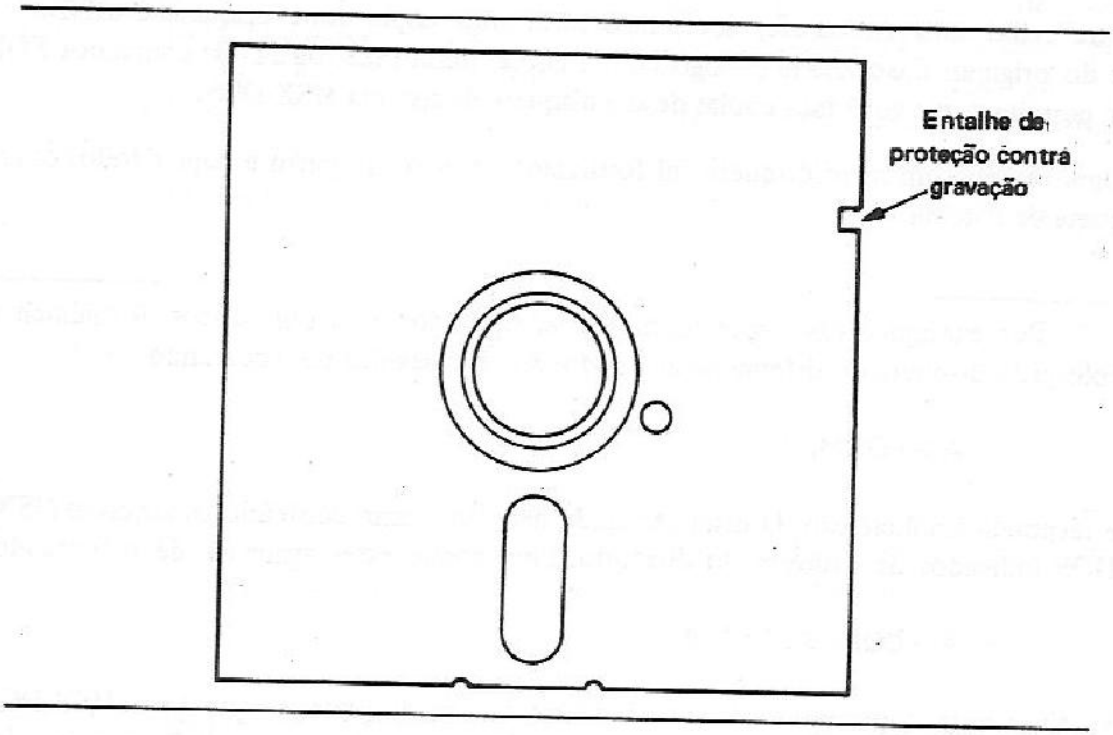


Figura 18.1 Entalhe para proteção contra gravação em um disquete de 5 1/4".

A Utilização de Arquivos em Comandos e Programas

A utilização mais comum dos comandos do MSX-DOS provavelmente refere-se à criação, manutenção e movimentação de arquivos no computador. Existe, sem dúvida, muito mais a ser aprendido a respeito de comandos. Entretanto, para utilizar o MSX-DOS com eficiência, deve-se apenas saber como utilizar alguns comandos comuns. A descrição que segue sucintamente apresenta alguns deles. Como esta seção trata de arquivos, não são apresentadas todas as utilizações dos comandos.

CRIANDO ARQUIVOS

Os arquivos são sempre criados por meio de programas. Você pode instruir um programa para que crie um arquivo (como um texto criado por um programa de processamento de palavras), ou um programa pode criar arquivos para uso próprio. Nesta seção vamos descrever os métodos gerais de criação de arquivos sem entrar nos detalhes de cada comando.

Uma maneira de criar arquivos é com um pacote de processamento de palavras. Outra é a de criar arquivos com o MSX-BASIC. A maioria de seus programas aplicativos também criará arquivos. Por exemplo, um programa contábil cria os arquivos para guardar os números que utiliza, e um programa de tabelas cria os arquivos que guardam os modelos financeiros. Adicionalmente, se você estiver utilizando uma linguagem de programação diferente do MSX-BASIC, esta permite que seus programas sejam guardados em disco.

DANDO NOME AOS ARQUIVOS

Para que um arquivo possa ser armazenado em disco, ele deve receber um nome. Cada arquivo tem uma referência única chamada "especificador de arquivo". Este especificador é desdobrado em duas partes: o nome do arquivo e a extensão. Todo arquivo deve ter um nome, mas a extensão é opcional. O nome do arquivo e a extensão são utilizados para descrever o conteúdo do arquivo. A combinação do nome do arquivo e a extensão deve ser única no disco; isto é, um disco não pode ter dois arquivos com o mesmo especificador de arquivo. A regra que deve ser memorizada ao nomear um arquivo é a de que o nome do arquivo pode ter até oito caracteres e a extensão de até três caracteres.

O nome do arquivo e a extensão são sempre separados por um ponto. Por exemplo, TELS.DAT é um especificador de arquivo: onde TELS é o nome do arquivo e DAT a extensão. Outros exemplos de especificadores de arquivo são CONTA.BAS, DENTAL12.COM e TIMEPRNT (podemos ter um nome de arquivo sem extensão como no último caso). Quando se pede ao MSX-DOS para ler ou escrever um arquivo, é obrigatório dar o nome do arquivo e a extensão, quando existir.

Nos manuais de computadores é normal utilizar "nome de arquivo" em vez de "especificador de arquivo". Caso você encontre "nome de arquivo", lembre-se de que esta referência será sempre do especificador completo, incluindo a extensão. A extensão do arquivo muitas vezes é denominada "tipo de arquivo".

Existe um grau de liberdade bem grande para dar nome aos arquivos, porém lembre-se das seguintes regras ao escolher os nomes dos arquivos e suas extensões:

PERMITIDO

Todas as letras do alfabeto (maiúsculas):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Os numerais:

0 1 2 3 4 5 6 7 8 9

Muitos símbolos de pontuação, especificamente:

! @ # $ % ^ & ( ) { } - _ ' ` ~

NÃO PERMITIDO

Os seguintes caracteres não podem ser utilizados:

, . / \ | ? * " : ; [ ] + =

Figura 18.2 Nome de caracteres legais e ilegais.

- Nem todos os caracteres existentes no teclado são permitidos em nome de arquivos. Muitos são utilizados em outras partes do MSX-DOS, podendo criar confusão se fossem utilizados em nomes de arquivo. Os caracteres permitidos e não permitidos estão indicados na Figura 18.2.
- Selecione um nome de arquivo pertinente com o conteúdo do arquivo. Por exemplo, uma carta dirigida para alguém chamado Carlos poderia ser denominada CARLOS.CRT, ou o seu imposto com data-base de 1984 poderia ser denominado IMP-84.DAD. Um nome apropriado para os arquivos facilita quando desejamos saber o conteúdo do arquivo, não sendo necessário, portanto, observar o conteúdo na tela.
- O MSX-DOS não permite nomes de arquivos iguais aos dispositivos que ele utiliza. (Nomes de dispositivos são nomes de componentes específicos de hardware, como a impressora.) Os nomes que não podem ser utilizados são AUX, COM1, COM2, CON, LST, LST1, LST2, LST3, PRN e NUL. Depois que você conhecer um pouco mais a respeito dos dispositivos, verá porque o MSX-DOS ficaria confuso se estes nomes fossem utilizados nos arquivos.
- Coloque uma extensão em seu especificador de arquivo, mesmo que não seja requerido. Um nome mais detalhado lhe dará maiores informações ao ler uma listagem de arquivos. Além do que a extensão acrescenta informação ao nome do arquivo que poderá ter importância mais tarde. Por exemplo, ao escrever uma carta para Armazém Geral, em vez de chamar o arquivo simplesmente de ARMAZÉM, poderia chamá-lo de ARMAZÉM. CRT. Quando, mais tarde, é gerado um arquivo com cálculos desta empresa, pode chamá-lo de ARMAZÉM.DAD. A extensão permite uma identificação bem simples com o outro arquivo ARMAZÉM.
- Os arquivos que tratam de um mesmo assunto devem ter nomes semelhantes. Desta maneira facilitamos a procura no disco de um arquivo ou grupo de arquivos. Assim, os memorandos para Sra. Wong poderão ser chamados de WONG1.MEM, WONG2.MEM, e assim em diante.
- Quando possível, utilize extensões padronizadas. A Tabela 18.1 mostra algumas extensões convencionais na indústria de microcomputação. Evidentemente, você pode ignorar estas convenções e chamar seus arquivos como bem desejar. Porém, alguns programas presumem que os arquivos têm certas extensões, não os encontrando se assim não for.
- Evite a utilização de extensões que tenham um significado especial para o MSX-DOS (como COM e EXE e BAT). Caso estas letras sejam utilizadas, o MSX-DOS pode pensar que estes arquivos são comandos e tentará executá-los. As conseqüências poderão ser desastrosas. (Por outro lado, é possível dizer quais arquivos existentes no disco são comandos, programas e arquivos em lote, bastando olhar as extensões. As extensões dos comandos são COM ou EXE, e os arquivos em lote têm como extensão BAT.)
- Apesar de ser permitida a utilização de grande parte dos símbolos de pontuação, a maioria utiliza apenas alguns deles. De uma maneira geral, a utilização do cifrão ($), do hífen (-) e do sublinhado (_) fornece um número suficiente de caracteres para a definição de um nome. A Figura 18.3 mostra alguns especificadores de arquivo válidos e não válidos.

Tabela 18.1 Extensões comuns de arquivos.

Extensão	Significado
ASM	Programa-fonte da linguagem assembly
BAK	Cópia duplicada de outro arquivo
BAS	Programa BASIC
BAT	Arquivo em lote
BIN	Arquivo binário utilizado por um programa
C	Programa-fonte C
COM	Programa
DAT	Arquivo de dados
EXE	Programa
OBJ	Arquivo objeto de um compilador
OVR	Arquivo adicional de um programa aplicativo
TEX	Arquivo de texto
TXT	Arquivo de texto

Válido:	Válido, porém não recomendável
MODEMS.DOC	XX"^ ^XX
CHAP-3A.TXT	!!!!!!!! @@@
PHONES.DAT	
10IDEAS	

Não-válido:	Motivo:
COMPUTERS.DOC	Nome de arquivo com mais de 8 caracteres
CHAP-3A.TEXT	Extensão com mais de 3 caracteres
PHONES.NAMES.DATA	Duas extensões
10IDEAS?	Caractere não permitido (?)

Figura 18.3 Exemplo de especificadores permitidos e não permitidos.

UTILIZANDO CARACTERES-CHAVE PARA GRUPOS DE ARQUIVOS

Alguns comandos do MSX-DOS poderão ser utilizados para a operação com grupos de arquivos com nomes ou extensões semelhantes. Podemos economizar muita digitação no agrupamento de arquivos durante a utilização de um comando. A utilização de um nome comum para um grupo de arquivos é como utilizar o sobrenome para um grupo de pessoas. Em vez de dizer "eu quero encontrar Roberto Moraes, Maria Moraes, Pedro Moraes e Antonia Moraes" poderia dizer "Eu quero encontrar a família Moraes". Igualmente, poderia utilizar o primeiro nome para encontrar um grupo de pessoas: "Quero encontrar todo mundo de nome Sabrina."

A utilização de um nome para um grupo de arquivos poderá economizar tempo e esforço. Por exemplo, pode haver necessidade em se transferir uma cópia dos 40 arquivos de seu programa BASIC de A: para B:. Seria necessário dar o comando COPY 40 vezes:

```
A> COPY PROG1.BAS B:
A> COPY PROG2.BAS B:
```

e assim em diante.

Para evitar este trabalho, no MSX-DOS existem dois caracteres especiais que poderão ser utilizados no nome dos arquivos quando você quiser especificar mais do que um arquivo: o ponto de interrogação (?) e o asterisco (*). Estes caracteres são utilizados nos especificadores de arquivo quando desejamos que o MSX-DOS atue sobre um grupo de arquivos e não sobre um único arquivo. Em vez de casar os nomes dos arquivos letra por letra, estes caracteres fazem o MSX-DOS procurar arquivos que tenham qualquer caractere naquele ponto do arquivo em que se encontra o ? ou *. Da mesma forma que um caractere-chave, em um jogo de cartas, pode ser utilizado para representar qualquer carta, estes dois caracteres podem ser utilizados para representar qualquer caractere do teclado. São, corretamente, denominados *caracteres-chave.*

O ? é utilizado para combinar com qualquer caractere na posição em que aparece no nome do arquivo.

Por exemplo, caso você queira copiar todos os arquivos que tenham cinco letras de comprimento, sendo que os quatro primeiros são PROG e tenham como extensão BAS, poderia dar o comando:

```
A> COPY PROG?.BAS B:
```

O comando vai copiar, do acionador A: para o disquete do acionador B:, os arquivos PROG1.BAS, PROG2.BAS, PROGA.BAS, PROGB.BAS e assim em diante. Ele não copiará nenhum dos programas PROG10.BAS ou PROGBM.BAS, uma vez que estes nomes têm mais letras do que PROG?.BAS e o ponto de interrogação representa apenas um caractere. Outro exemplo será:

```
A> COPY TAX?83.DAT B:
```

Este comando copiaria os seguintes arquivos: TAXA83.DAT, TAX183.DAT, TAX-83.DAT e assim em diante; não copia TAXAB83.DAT ou TAXA.DAT.

Outro caractere-chave, o caractere *, combina com qualquer número de caracteres na posição em que se encontra no nome do arquivo. Significa que o número de letras combinadas não é importante, e que podem ser combinados arquivos com números diferentes de letras no local especificado com o caractere *.

Por exemplo, para copiar qualquer arquivo que tem o nome iniciando com PROGR e cuja extensão é BAS, dê o comando:

```
A> COPY PROG*. BAS B:
```

Confere com PROG*1*.BAS, PROG*1A*.BAS, PROG*9999*.BAS e PROG*2B2B*.BAS.

Comparando este exemplo com o anterior, utilizando o asterisco (algumas vezes denominado estrela) sempre serão combinados pelo menos o mesmo número de arquivos do que com o ponto de interrogação.

Poderá ser utilizado mais do que um caractere-chave em uma especificação de arquivo. Por exemplo, B???E.* combina nomes de arquivos como BLARE.COM, B000E.123, B-D-E.TXT e BRAKE.

Com a utilização do caractere-chave, o problema de como copiar todos os programas BASIC de A: para B: fica muito mais fácil de ser resolvido. Para copiar qualquer arquivo que tenha a extensão BAS, independentemente do número de letras no nome, o nome do arquivo deve ser substituído por um asterisco depois do comando COPY:

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

Quando, em lugar disto, você quer copiar todo arquivo para B:, independentemente de seu nome ou extensão, dê o comando:

```
A> COPY *. * B:
```

É o mesmo procedimento utilizado anteriormente nesta seção para fazer uma cópia do disquete de sistema.

UTILIZANDO OS NOMES DE DISPOSITIVOS NOS COMANDOS

Como foi visto, o comando COPY pode ser utilizado tanto para copiar arquivos de (ou para) um disco como também para movimentar informações entre dispositivos. O MSX-DOS fornece uma maneira muito simples de comunicação com dispositivos, como uma impressora e portas de comunicação.

É mais do que provável que será necessário utilizar nomes de dispositivos com o comando COPY, porque a maioria dos programas que utiliza dispositivo tem comandos internos que imediatamente se comunicam com os dispositivos. Os nomes estão relacionados e definidos na Tabela 18.2. Todos os nomes de dispositivos têm três caracteres (às vezes seguidos por um número). Alguns dispositivos são sinônimos, como AUX e COM1, ou LPT1 e PRN. É muito comum que os estreantes em MSX-DOS confundam nomes de arquivos com nomes de dispositivos.

Em vez do nome do arquivo no comando COPY, utilize o nome do dispositivo. Da mesma forma que um arquivo, qualquer dispositivo pode ser utilizado para inserção (leitura) ou saída

(gravação). Na verdade, como a comunicação com estes dispositivos se processa de forma similar ao processo de copiar dados de e para arquivos, veremos que o aprendizado dos nomes dos dispositivos é uma continuação do aprendizado do nome de arquivos. O único comando com o qual se utiliza o nome dos dispositivos é o COPY.

Tabela 18.2 Nomes de dispositivos utilizados pelo MSX-DOS.

Nome	Dispositivo
CON	Console. É na verdade a combinação do teclado com a tela. A entrada é feita pelo teclado e a saída aparece na tela.
AUX e COM1	A primeira porta de comunicação (serial). Também é referida como Porta de Adaptação para Comunicação Assíncrona ou Porta RS-232. Nem todos os computadores têm esta porta. As portas de comunicação dois e três (se existirem em seu computador) são denominados COM2 e COM3.
LPT1 ou PRN	Porta da impressora paralela. Esta pode ser chamada ainda de Porta Centronics ou Porta Paralela. Nem todos os computadores têm esta porta. Como muitas impressoras utilizam uma porta serial, poderá ser necessário se fazer um redirecionamento de E/S para esta porta com o comando MODE ou outro comando similar fornecido pelo fabricante. As portas paralelas dois e três (caso existam) são denominadas LPT2 e LPT3.
NUL	Um dispositivo inexistente, utilizado apenas para teste de programas aplicativos.

Por exemplo, poderá haver necessidade de enviar um arquivo para a impressora; para tanto, dê o comando:

```
A> COPY ARQUIVO.TXT PRN
```

Desta maneira o arquivo ARQUIVO.TXT, existente no disquete do acionador A: , será copiado para a impressora.

Utilizando Arquivos-em-Lote para Combinar Comandos MSX-DOS

Agora que já aprendemos uma série de comandos MSX-DOS e estamos familiarizados com o procedimento geral para sua utilização, estamos em condição de executar um grupo deles, um após o outro. Existindo diversos comandos que devem ser executados, uma maneira de ganharmos tempo é pela sua combinação em um arquivo de texto denominado "arquivo-em-lote". Ao pro-

cessar um arquivo-em-lote, os comandos serão processados como se estivessem sendo digitados no momento da execução.

Escrever um arquivo-em-lote é como escrever um programa muito simples que fornece ao MSX-DOS uma lista de comandos a serem executados. Porém, é bem mais simples do que programar. Quando os comandos estão prontos para ser executados, basta fornecer ao MSX-DOS o nome do arquivo-em-lote, em lugar do nome de cada um dos comandos.

Os comandos ou programas que podem ser utilizados em um arquivo-em-lote são os mesmos que para a solicitação A>. Observe como este tipo de combinação de comandos evita um excesso de digitação.

Por exemplo, vamos supor que você esteja utilizando um programa de contabilização denominado ACCT, que gera um arquivo denominado SALES.DAT. Este programa é processado todo dia, sendo que toda vez em que ACCT é processado, é utilizado também o comando COPY para copiar o arquivo SALES.DAT no arquivo SALESNEW.DAT. Adicionalmente, o programa ACCT exige que se dê o nome dos dois outros arquivos com quem opera, MAINACCT.DAT e NEWREPT.TXT, na linha de comando. Sem a utilização do arquivo-em-lote, seriam dados os comandos:

```
A> ACCT SALES.DAT MAINACCT.DAT NEWREPT.TXT
A> COPY SALES.DAT SALSNEW.DAT
A> ■
```

Uma vez que sempre são digitados os nomes dos três arquivos e o segundo comando, o tempo de digitação poderá ser bastante reduzido pela utilização de arquivo-em-lote. Para gerar um arquivo-em-lote utilizamos um editor de texto. Suponhamos que foi criado o arquivo-em-lote DAILY.BAT:

DAILY.BAT

```
ACCT  SALES.DAT  MAINACCT.DAT  NEWREPT.TXT
COPY  SALES.DAT  SALESNEW.DAT
```

Agora para processar o arquivo-em-lote, basta que seja fornecida um único comando, simples, DAILY, para o MSX-DOS (não há necessidade de digitar a extensão

BAT). Assim que o MSX-DOS encontra o arquivo-em-lote denominado DAILY.BAT inicia a execução dos comandos ali existentes:

```
A> DAILY
A> ACCT SALES.DAT MAINACCT.DAT NEWREPT.TXT
A> COPY SALES.DAT SALESNEW.DAT
A> ■
```

Note que o MSX-DOS primeiro encontrou o arquivo-em-lote (DAILY.BAT) para em seguida executar cada uma de suas linhas. À medida que cada uma das linhas é executada, ela será digitada na tela pelo MSX-DOS.

REGRAS PARA NOMEAR OS ARQUIVOS-EM-LOTE

Existem algumas regras que devem ser seguidas durante a nomeação dos arquivos-em-lote:

- Um nome de arquivo-em-lote deverá incluir a extensão BAT de modo que ao ser dado o comando, o MSX-DOS o reconheça como um arquivo-em-lote.
- Não dê ao arquivo-em-lote o mesmo nome que o de um comando. Caso isto ocorra, o MSX-DOS vai processar o comando, e não o arquivo-em-lote. Quando o MSX-DOS recebe um comando, primeiro verifica se este é um comando interno; não sendo, procura um nome com a extensão COM, em seguida uma extensão EXE e finalmente a extensão BAT.
- Uma vez que os arquivos-em-lote são idênticos aos outros arquivos, aplicam-se as mesmas regras quanto aos caracteres que podem ou não ser utilizados.

Em um arquivo-em-lote podemos colocar qualquer comando MSX-DOS, e podemos dar para este tipo de arquivo qualquer comprimento. Cada um dos comandos devem estar em linhas separadas, porque o MSX-DOS lê os comandos do arquivo-em-lote como se tivessem sido digitados em resposta a uma solicitação do MSX-DOS.

Para tornar o processamento em lotes ainda mais fácil, o MSX-DOS fornece comandos específicos para este fim. Eles permitem que seja feita mudança de discos entre comandos, verificam a existência de um determinado arquivo, bem como executam outras tarefas úteis. Estes comandos são descritos no Capítulo 19, na seção "Comandos de arquivos-em-lote".

UTILIZANDO ARGUMENTOS NOS ARQUIVOS-EM-LOTE

Nos exemplos anteriores foi visto como utilizar os arquivos-em-lote quando são processados sempre os mesmos comandos com os mesmos argumentos. Mas como fazer, caso queiramos dar a um dos comandos do arquivo-em-lote um argumento diferente cada vez que ele é executado? O MSX-DOS permite que seja utilizado "substituição de argumento" para mudar o argumento ao processar o arquivo-em-lote. Os novos argumentos são então utilizados nos comandos existen-

tes no arquivo-em-lote. Felizmente, o MSX-DOS não se incomoda com o tipo de argumento dado; poderá ser o nome de um arquivo, o nome de um acionador ou qualquer outro argumento de sua vontade.

Para substituir um argumento na linha de comando de um arquivo-em-lote, é utilizado o sinal percentual e o número do argumento no arquivo-em-lote. (Por exemplo, o primeiro argumento seria denominado %1.) O sinal percentual informa ao MSX-DOS que deve substituir %1 no arquivo-em-lote com o primeiro argumento da linha de comando, para substituir %2 do arquivo-em-lote com o segundo argumento da linha de comando, e assim em diante. Em seguida é apresentado um exemplo mostrando quando uma substituição de argumento pode ser útil.

Vamos supor que haja necessidade de copiar um grupo de arquivos-relatório do disco A: para o disco B: ou C:. Poderíamos fazer dois arquivos-em-lote diferentes – um que copia os arquivos em B: e o outro que os copia em C:. Porém, para economizar ainda mais o tempo, podemos escrever um arquivo-em-lote que utiliza substituição de argumento. O arquivo-em-lote RPTCOPY.BAT poderia ser:

RPTCOPY.BAT

```
COPY * RPT %1
```

Observe que neste arquivo-em-lote, o segundo argumento após o comando COPY é %1. Sempre que este arquivo-em-lote é utilizado, deve-se fornecer como argumento o nome do disco em que se deseja copiar os arquivos. Assim o MSX-DOS substitui no comando COPY de seu arquivo-em-lote o nome do dispositivo dado.

Para processar o arquivo-em-lote RPTCOPY tendo C: como argumento, digite:

```
A> RPTCOPY C:

A> COPY *. RPT C:
        5 File(s) copied

A> ■
```

Assim que o MSX-DOS encontra %1 em seu arquivo-em-lote RPTCOPY, substitui-o por C:, porque este é o primeiro argumento na linha de comando. O mesmo arquivo-em-lote poderá ser utilizado para copiar no disco B:

```
A> RPTCOPY B:

A> COPY *. RPT B:
        5 File(s) copied

A> ■
```

Caso existam diversos comandos que utilizem o mesmo argumento cada vez em que é processado o arquivo-em-lote, o argumento percentual pode ser utilizado mais do que uma vez.

Por exemplo, suponhamos a existência de dois comandos CALC-BEG e CALC-END, que sempre são processados em conjunto. Cada comando utiliza o mesmo argumento, o nome de um arquivo de dados. Podemos escrever um arquivo CALCBOTH.BAT para processar tanto o programa CALC-BEG como o programa CALC-END com o mesmo argumento:

CALCBOTH.BAT

```
CALC-BEG %1
CALC-END %1
```

Ao processar este arquivo com o argumento FY1982.DAT, veremos na tela:

```
A> CALCBOTH FY1982.DAT

A> CALC-BEG FY1982.DAT

A> CALC-END FY1982.DAT

A> [ ]
```

Se houver necessidade de substituir diversos argumentos em um comando, podemos utilizar mais símbolos percentuais: %2, %3, %4 e assim em diante. De fato, o MSX-DOS permite que sejam fornecidos até nove argumentos ao seu arquivo-em-lote. Caso seja utilizado um segundo argumento, o MSX-DOS substitui cada um dos %2 existentes em seu arquivo-em-lote com o segundo argumento; substituirá cada %3 com o terceiro argumento e assim em diante.

A capacidade do MSX-DOS de incluir diversos argumentos em seu arquivo-em-lote é uma ferramenta muito útil. Deste modo é possível criar arquivos-em-lote que realizam tarefas complexas.

Vamos supor que haja necessidade de fornecer um único nome ao segundo e terceiro arquivos no programa de contabilização do arquivo DAILY.BAT, do exemplo anterior. Iniciaríamos reescrevendo o arquivo de maneira que sejam incluídos dois argumentos para serem substituídos:

DAILY.BAT

```
ACCT  SALES.DAT %1  %2
COPY  SALES.DAT SALESNEW.DAT
```

A seguir será dado o comando DAILY e os dois nomes de arquivo:

```
A> DAILY RECEIVE.DAT RCVREPT.TXT

A> ACCT SALES.DAT  RECEIVE.DAT  RCVREPT.TXT

A> COPY SALES.DAT  SALESNEW.DAT

A> ■
```

Agora que estamos familiarizados com os argumentos de substituição dos arquivos-em-lotes, já sabemos porque devemos evitar a utilização do sinal percentual nos nomes dos arquivos. Caso exista um especificador de arquivo, em seu programa de lote, que tem o sinal percentual (como N%X.DAT), será necessário lembrar de dar-lhe outro sinal percentual (como N%%X.DAT). O segundo sinal percentual informa ao MSX-DOS que o primeiro sinal é parte do nome do arquivo.

O ARQUIVO AUTOEXEC.BAT

Existe um tipo de arquivo-em-lote que ainda não foi comentado. Este arquivo inicia um processamento automático ("auto-execução") cada vez que se inicia o computador. É denominado arquivo AUTOEXEC.BAT, e pode ser elaborado de modo a executar certos programas automaticamente, ou imprimir os resultados dos programas processados no dia anterior, assim que você iniciar o computador.

Para escrever um programa AUTOEXEC.BAT, basta criar um arquivo-em-lote (como aqueles já vistos) e dar-lhe o nome de AUTOEXEC.BAT. Podemos escrever um arquivo AUTOEXEC.BAT com qualquer comando desejado. Cada vez que o MSX-DOS é iniciado, ele vai procurar o arquivo de nome AUTOEXEC.BAT. Encontrando-o, automaticamente executa os comandos. Casualmente este é o arquivo especial mencionado no início deste capítulo que impede o MSX-DOS de pedir-lhe a data e a hora.

Os comandos que estabelecem os sistemas (como data e hora) são normalmente colocados no arquivo AUTOEXEC.BAT. Podem ser incluídos pacotes de software, que devem ser carregados na memória assim que o sistema é ligado. Um arquivo AUTOEXEC.BAT típico é apresentado no final da seção "Comandos de arquivos-em-lote" do Capítulo 19.

Um Modelo de uma Sessão com MSX-DOS

Agora você já sabe quase tudo a respeito do MSX-DOS que lhe permite iniciar sua utilização como um profissional. Nesta seção vamos apresentar o exemplo de uma sessão típica com um computador, utilizando o MSX-DOS.

Suponhamos que a tarefa do dia seja a de processar um pacote, que tenha como resultado um relatório, que é parte de seu sistema de controle de inventário, e ainda escrever um memorando sobre o resultado do relatório. Você ainda deve imprimir o memorando e fazer uma cópia, em disquete, do memorando e do relatório. Estas tarefas serão apresentadas passo a passo.

PASSO 1:
UTILIZANDO UM ARQUIVO-EM-LOTE PARA PROCESSAR O PROGRAMA

Primeiro você vai processar o programa de relatório do inventário. O processamento do programa é simples porque você já escreveu o arquivo-em-lote (chamado INVRPT) que vai tratar os três diferentes comandos que normalmente são dados. O arquivo INVRPT.BAT contém:

```
A> INVRPT 8/1/84   8/15/84

A> INVMAKE 8/1/84 8/15/84  INVKEY.TXT

A> INVFILE INVKEY.TXT
```

INVRPT.BAT

```
INVMAKE %1  %2  INVKEY.TXT
INVFILE  INVKEY.TXT
INVOUT  %1  %2  INVKEY.TXT
```

Será necessário, apenas, formar ao arquivo-em-lote quais são as datas iniciais e finais do relatório:

```
A> INVOUT 8/1/84  8/1/84  INVKEY.TXT
O relatório está arquivado no arquivo INVRPT93.TXT

A> ■
```

A última mensagem ("o relatório está arquivado...) é impressa pelo programa INVOUT. O programa hipotético lhe informa o nome do arquivo gerado e salvo no disco (neste caso, INVRPT93.TXT). Quando o programa terminou, a solicitação A> voltou à tela.

PASSO 2:
ANALISANDO O RELATÓRIO

O passo seguinte é a análise do relatório gerado. Pode vê-lo na tela ou imprimi-lo pela impressora. Para mostrar o relatório na tela utilize o comando TYPE:

```
A> TYPE INVRPT93.TXT

          Relatório de inventário entre 8/1/84 e 8/15/84

Itens que devem ser requisitados em 8/20/84:

      Item          Existente       Requisitar       Preço
-------------------------------------------------------------
    CF3499              8               12            14.50
    CF5000             52               60             1.19
      .                 .                .              .
      .                 .                .              .
      .                 .                .              .
```

Caso o relatório fique maior do que se esperava, ele pode ser impresso com o comando COPY e o dispositivo PRN:

```
A> COPY INVRPT93.TXT PRN

A> ■
```

PASSO 3:
ESCREVENDO O MEMORANDO

Após avaliar o relatório, utilize o programa de processamento de palavras para descrever um memorando detalhando os resultados. Uma vez que cada processador de palavras é diferente do outro, neste livro não vamos explicar como escrever o memorando.

Vamos supor que você esteja processando um programa de processamento de palavras chamado TEXTP e deseja editar o arquivo INVRPT93.TXT. Será dado o comando:

```
A> TEXT  INVRPT93.TXT
```

Você acrescenta um texto ao relatório e rearranja os resultados. Após terminar o memo, vamos imprimi-lo pela impressora. Muitos são os processadores de palavras que, ao serem solicitados, imprimem o memorando; muitos não o fazem.

PASSO 4:
COPIANDO O MEMORANDO E OS ARQUIVOS DO RELATÓRIO

Lembre-se de que para este trabalho é necessário ter uma cópia do relatório e do memo em disco. Para fazer isto deve ser colocado no segundo acionador (B:) um disco já formatado, com a certeza de que está vazio pela utilização do comando DIR.

```
A> DIR B:
        File not found
A> ■
```

A mensagem antes de "File not found" poderá variar de acordo com a versão do MSX-DOS utilizado. Isto, entretanto, não importa porque encontramos o que queríamos saber: nada há no disco.

Agora podemos copiar o relatório e o memo:

```
A> COPY INVRPT93.TXT B:
                1 File(s) copied
A> COPY NEWINV.MEM B:
                1 File(s) copied
A> ■
```

Lembre-se de que o comando COPY exige dois argumentos: a origem dos dados e o destino dos dados. No caso em questão a fonte é o arquivo A:, o acionador assumido, e o destino é o acionador B:. Para termos certeza de que tudo foi copiado de acordo, basta verificar o conteúdo de B: com o comando DIR:

```
A> DIR B:
INVRPT93.TXT    8110    8-20-84    10:42p
NEWINV  MEM      613    8-20-84    11:20p
       2 File(s)      151515 bytes free

A> ■
```

O comando DIR fornece-lhe o nome do arquivo, extensão do arquivo, o número de bytes no arquivo, a data e hora de sua geração. Informa ainda quantos arquivos existem no disco e a quantidade de espaço disponível ("bytes free") antes que o disco fique cheio.

Você chegou ao fim de sua tarefa e deve sentir-se satisfeito em poder dar os comandos MSX-DOS e em responder às perguntas feitas pelo programa. Agora você está em condição de poder aprender muitos outros comandos MSX-DOS descritos no próximo capítulo.

CAPÍTULO

19

COMANDOS MSX-DOS

Este capítulo descreve os comandos MSX-DOS que acompanham seu sistema e informa como utilizá-los. A maioria das descrições dos comandos inclui exemplos que mostram como podem ser aplicados. Os exemplos são hipotéticos e contêm nomes de arquivos que não existem em seus discos. Devem ser utilizados como protótipos genéricos de utilização de um determinado comando para uma determinada finalidade.

Seu computador poderá ter vindo acompanhado tanto do MSX-DOS como de programas aplicativos do tipo para processamento de palavras ou planilhas eletrônicas. Uma vez que o MSX-DOS não inclui programas aplicativos, estes não são explicados aqui. Estes programas devem vir acompanhados pelo manual de instruções. Caso você tenha discos de programas, sem a documentação (isto é, um manual que explique como utilizar o programa), peça um ao seu distribuidor.

Como este Capítulo está Organizado

Neste capítulo, todos os comandos MSX-DOS são apresentados em grupos funcionais genéricos. Por exemplo, você verá que todos os comandos utilizados para a manutenção de arquivos (como copiar e apagar) estão agrupados sob o título "Comandos para Manutenção de Arquivos". A Figura 19.1 mostra estes grupos funcionais e os comandos em cada grupo.

Cada descrição de comando inclui a seguinte informação:

- Utilização comum. Esta seção mostra como utilizar o comando e seus argumentos, de diversas maneiras. Aqui apresentamos a maioria dos exemplos de cada comando. Alguns comandos apresentam, ainda, uma seção denominada "Outras utilizações", em que são descritas utilização menos práticas dos comandos.
- Regras. Alguns comandos têm regras bem definidas sobre quando ou não podem ser utilizados. Estas regras são descritas nesta seção.

Comandos MSX-DOS	Função
Comandos para Manutenção de Arquivos	
ERASE	Remove arquivos do disco
RENAME	Dá nomes novos aos arquivos
COPY	Copia arquivos
Comando para Saída de Arquivo	
TYPE	Apresenta um arquivo na tela
Comandos para Manutenção do Disco	
DIR	Apresenta uma lista dos arquivos existentes no disco
FORMAT	Prepara um disco para MSX-DOS
Identificação do Sistema	
DATE	Coloca a data
TIME	Coloca o horário
MODE	Altera os parâmetros de comunicação
Comandos de Arquivos-em-Lote	
PAUSE	Aguarda a operação de uma tecla
REM	Coloca uma observação em um arquivo-em-lote
Outros Comandos	
BASIC	Processa MSX-BASIC

Figura 19.1 Agrupamentos de comandos deste capítulo.

- Avisos e erros comuns. Esta seção relaciona os problemas que normalmente são encontrados com determinados comandos; ainda fornece avisos que podem ajudar a evitar problemas maiores.
- Sintaxe. A sintaxe de um comando é um arranjo ordenado dos argumentos do comando, que são colocados na linha do comando. Esta seção relaciona a sintaxe de cada comando e apresenta um volume de informação muito maior do que o necessário para a utilização do comando. (Esta informação ajuda na compreensão exata de como dar o comando.) Uma sintaxe generalizada é fornecida na seção de "Utilização comum".

Como Ler uma Sintaxe de Comando

Você deve aprender a escrever um pouco da linguagem MSX-DOS, para poder dar um comando. Para que o MSX-DOS possa ler o comando, o nome do comando e seus argumentos devem ser organizados de acordo com a sintaxe do comando. A sintaxe pode ser curta e simples ou longa e complexa. Especifica tudo o que o comando necessita para executar seu trabalho.

Pela leitura cuidadosa da sintaxe e pelo fornecimento de todas as informações nos argumentos, pode estar certo de que o comando será processado sem problema.

Para entender como interpretar as notações da sintaxe de comando neste livro, os três conceitos que seguem são muito importantes. Observe que a sintaxe dos comandos MSX-DOS é diferente daquela mostrada para o MSX-BASIC.

- Símbolos especiais. Este livro utiliza símbolos especiais – relacionados na Tabela 19.1 – para indicar tipos diferentes de argumentos. Qualquer argumento que não se apresenta envolto por um símbolo especial é exigido para aquele comando. Como será visto a maioria dos argumentos é opcional. Você deve familiarizar-se com estes símbolos, já que eles são utilizados neste livro e também em outros livros sobre computação.

Tabela 19.1

Símbolo	Descrição
[ ]	Este símbolo indica que tudo que estiver dentro dele é opcional.
{ }	Este símbolo indica que você deve escolher um dos elementos dentro dele. Cada escolha é separada por uma barra vertical (|).
. . .	Este símbolo indica que o último argumento pode ser repetido novamente.

Tabela 19.2 Argumentos gerais nas sintaxes de comandos.

Argumento geral	Descrição
d:	Representação de disco, como A: e B:
nomearq	O nome de um arquivo (ou grupo de arquivos), com exclusão da extensão de arquivo. Podem ser utilizados nomes de arquivos-chave a não ser que a descrição do comando diga o contrário.
ext	Extensão de um arquivo ou grupo de arquivos. Podem ser utilizados nomes de arquivos-chave, a não ser que a descrição do comando diga o contrário.
esparq	Um especificador de arquivo, com pelo menos o nome de um arquivo. Assim, a sintaxe de esparq é: [d:] nome arq [.ext]
n	Um número. Por exemplo, COMn pode significar COM1, COM2 e assim por diante.

Argumentos itálicos. Estes argumentos, também conhecidos como "argumentos generalizados", são utilizados para mostrar que o argumento pode assumir diversos valores. Por exemplo, na sintaxe TYPEesparq, "esparq" mostra que pode ser utilizado qualquer nome de arquivo ou especificação de arquivo. O argumento "esparq" é uma palavra inventada para mostrar onde deve ser colocado o nome de um arquivo. Alguns argumentos generalizados, utilizados neste capítulo, estão descritos na Tabela 19.2; outros argumentos, específicos para um determinado comando, são descritos à medida que aparecem no comando em questão.

Argumentos com uma letra. Alguns comandos têm argumentos de uma única letra, precedidos por uma linha inclinada (por exemplo, ,/V). É importante entender que estes argumentos têm significados diferentes em comandos diferentes. A utilização de uma mesma letra com significados diferentes poderá criar confusão. É sempre necessário verificar se este tipo de argumento está sendo utilizado corretamente no comando apropriado. (Alguns manuais chamam estes argumentos de "opções".) Ao digitar estes argumentos, certifique-se de que a linha inclinada utilizada é (/), e não (\).

Nos exemplos seguintes, você poderá verificar como é importante compreender a sintaxe de um comando.

O primeiro exemplo é:

```
DIR [d:]
```

Com esta sintaxe, o comando DIR pode aparecer só ou com um descritor de disco como B:. Outro exemplo será:

```
TYPE esparq
```

"esparq", normalmente apenas o nome do arquivo é necessário neste comando. Lembre-se de que esparq pode incluir também o nome do disco.

Interrompendo um Comando

Pode ser que você queira parar um comando – pois você notou que ele está imprimindo algo que você não quer –, após iniciar seu processamento. Isto normalmente pode ser feito pressionando a tecla CONTROL e ao mesmo tempo a tecla STOP. Assim o MSX-DOS é informado de que você quer parar um comando. Após a operação de CONTROL-STOP, a tela deverá apresentar A>. Normalmente a operação de CONTROL-STOP funciona, mas nem sempre.

Quando você quer interromper um comando e a operação descrita não dá resultado, pode desligar o computador como um último recurso. Desligar o computador interrompe o que ele está fazendo, independentemente das conseqüências.

O desligamento do computador é uma maneira efetiva de parar um comando, mas não é recomendável. Uma informação que está sendo escrita no momento de desligamento da chave pode ser seriamente danificada.

Para interromper temporariamente um programa sem que apareça o sinal A>, pressiona-se a tecla STOP. Uma nova operação da tecla STOP normalmente continuará o processamento do comando.

Comandos para Manutenção de Arquivos

Nesta seção serão abordados os comandos ERASE, RENAME e COPY.

Comando ERASE

O comando ERASE remove um arquivo, ou grupo de arquivos do diretório do disco. O comando DEL é idêntico ao ERASE.

UTILIZAÇÕES COMUNS

Existem diversos motivos para se utilizar o comando ERASE. Se um de seus programas criou um arquivo inútil, ou quando há um arquivo que não é mais necessário, eles podem ser removidos do disco com o comando ERASE. Uma vez que existe um espaço limitado para armazenamento, é importante a eliminação de arquivos velhos, liberando espaço para outros arquivos.

A sintaxe do comando ERASE é

```
ERASE esparq
```

O processo para o cancelamento de um arquivo é simples, bastando citar o nome do arquivo a ser eliminado.

Por exemplo, para cancelar um arquivo denominado BOSTON88.TST no acionador assumido, dê o comando:

```
A> ERASE BOSTON88.TST
```

ou

```
A> DEL BOSTON88.TST
```

O arquivo será removido do disco e do diretório.

Com o comando ERASE podem ser utilizados caracteres-chave. Desta maneira, é possível eliminar mais do que um arquivo por vez.

Por exemplo, para cancelar todos os arquivos com a extensão BAK (utilizado para arquivos duplicados), dê o comando:

```
A> ERASE *.BAK
```

Caso solicitemos que todos os arquivos do disco sejam cancelados, antes de fazê-lo o comando ERASE faz uma confirmação.

```
A>  ERASE *. *
Are you sure (Y/N)?  Y
```

Sempre que você vê esta mensagem, pense sobre o conteúdo dos arquivos com cuidado. Uma vez cancelado, não existe nenhum método simples de se recuperar um arquivo; pode, inclusive, ser irrecuperável.

REGRAS

O comando ERASE deve ser informado qual é o arquivo que deve ser eliminado. Quando o nome do arquivo é esquecido, o MSX-DOS emite a mensagem "Invalid number of parameters" (Número de parâmetros incorreto).

Quando o disco é especificado mas o nome do arquivo é esquecido, o MSX-DOS informa "File not found" (Arquivo não encontrado). Tal procedimento é como uma precaução do comando ERASE, que exige que, quando é especificado um disco, o nome do arquivo também o seja.

AVISOS E ERROS COMUNS

Evidentemente, deve haver cuidado para não eliminar arquivos importantes. Caso outras pessoas utilizem seu computador, tenha o cuidado antes de cancelar arquivos. Apesar de você pensar que um arquivo não tem importância, poderá ser valioso para uma aplicação, feita por outra pessoa, que não seja de seu conhecimento.

SINTAXE

A sintaxe do comando ERASE é:

```
ERASE esparq
```

É obrigatório fornecer "esparq" com o comando ERASE. "Esparq" pode conter caracteres-chave.

Comando RENAME

O comando RENAME muda o nome de um arquivo sem alterar o seu conteúdo. O comando REN é semelhante ao comando RENAME.

UTILIZAÇÕES COMUNS

Existem algumas razões pelas quais você pode querer mudar o nome de um arquivo. O conteúdo de um arquivo pode ter sido alterado e o seu nome não é mais apropriado, ou você pode ter dado nomes semelhantes para dois arquivos. Um dos arquivos pode ter o nome alterado para evitar confusão.

A sintaxe comum deste comando é:

```
RENAMEesparq1 esparq2
```

onde esparq1 refere-se ao nome original do arquivo, e esparq2 refere-se ao nome novo.

Por exemplo, para alterar o nome de SALES.DAT para US-SALES-DAT, dê o comando:

```
A> RENAME SALES.DAT  US-SALES.DAT
```

ou

```
A> REN  SALES.DAT  US-SALES.DAT
```

Podemos utilizar o comando RENAME juntamente com o comando ERASE para nos livrarmos da versão atual de um arquivo e substituí-la por uma versão anterior. Isto poderá ser necessário quando, ao revisarmos um arquivo, decidirmos que a versão original era mais apropriada.

Por exemplo, vamos assumir que haja interesse em alterar o nome do programa de reserva de um arquivo de cobrança, denominado BILLPROG.COB. Você está utilizando um editor de texto que guarda um arquivo de reserva com o mesmo nome do arquivo original, mas com uma extensão BAK. Seriam dados os comandos:

```
A> ERASE BILLPROG.COB

A> RENAME BILLPROG.BAK  BILPROG.COB
```

Se você apenas quer comutar os nomes de dois arquivos, será necessário utilizar um nome temporário, já que o comando RENAME não permite que os dois arquivos tenham os nomes alterados ao mesmo tempo. Esta operação poderá ser necessária, quando, após a revisão de um arquivo decidimos utilizar a versão anterior, mas queremos manter uma cópia das revisões para utilização futura.

Por exemplo, para comutar os nomes dos arquivos SALEPROG.COB e SALEPROG.BAK, digite os seguintes comandos:

```
A> RENAME SALEPROG.COB   TEMPFILE.COB

A> RENAME SALEPROG.BAK   SALEPROG.COB

A> RENAME TEMPFILE.COB   SALEPROG.BAK
```

Neste exemplo, TEMPFILE.COB é o nome de um arquivo temporário.

Quando houver dois ou mais arquivos em um disco que compartilham o mesmo nome ou a mesma extensão, podemos utilizar os caracteres-chave para alterar o nome destes arquivos, como um grupo.

O primeiro exemplo altera todos os arquivos de nome ACCOUNTS para ACC.

```
A> RENAME ACCOUNTS.*ACC.*
```

O segundo exemplo altera todos os arquivos com extensão TXT para extensão TEX.

```
A> RENAME *.TXT *.TEX
```

REGRAS

Não podemos utilizar RENAME para alterar os nomes de um grupo de arquivos que estão em discos diferentes; primeiro devemos utilizar o comando COPY para colocar os arquivos no mesmo disco.

Não é possível alterar o nome de um arquivo para um nome já existente no disco. Caso seja feita a tentativa, o MSX-DOS envia a seguinte mensagem "Duplicate file name or file name not found" (Duplicação do nome do arquivo ou arquivo não encontrado).

AVISOS E ERROS COMUNS

O erro mais comum é esquecermos a ordem em que devem ser colocados os nomes dos dois arquivos. É obrigatória a colocação do nome antigo antes do nome novo; exigência feita pelo comando RENAME. (Aqueles familiarizados com o sistema operacional CP/M reconhecem que é o oposto às exigências do CP/M).

Assegure-se de que, quando existirem várias pessoas que se utilizam de seu computador, ao alterar o nome de um arquivo, todos os usuários tenham conhecimento do novo nome. A sua não informação, ou esquecimento, poderá causar confusão desnecessária.

Não altere os nomes de arquivos especiais, utilizados por programas aplicativos. Por exemplo, o programa WordStar exige que o arquivo WSOVLY1.OVR esteja no disco. Se você alterar o nome deste arquivo não haverá meios de o programa saber o novo nome.

SINTAXE

A sintaxe do comando RENAME pode ser:

```
RENAME esparq1 esparq2
```

ou

```
RENAME esparq1 nomearq2 [.ext2]
```

Na primeira forma, esparq1 é o nome antigo do arquivo e esparq2 é o novo nome do arquivo. Na segunda forma, nomearq2.ext2 é o nome novo do arquivo (caso você não inclua a a extensão, o MSX-DOS utilizará a mesma que havia no nome do arquivo original). Observe que o nome antigo é dado primeiro..

Comando COPY

O comando COPY permite que sejam copiados registros de um arquivo para outro. Permite também que você copie um arquivo no mesmo disco (porém com um nome diferente), copie informação do (e para) os dispositivos de hardware do sistema, bem como combine diversos arquivos formando um único.

UTILIZAÇÕES COMUNS

O comando COPY é utilizado com freqüência para copiar um arquivo de um disco para outro.

A sintaxe mais comum deste comando é:

```
COPY fonte destino
```

Neste caso, "fonte" é o nome do arquivo do qual é feito copia e "destino" é para onde você quer copiar o arquivo.

Por exemplo, caso haja interesse em copiar o arquivo OPENACCT.DAT de um disco no acionador A: para um disco no acionador B:, deve ser dado o comando:

```
A> COPY OPENACCT.DAT B:
        1 File(s) copied
```

O arquivo OPENACCT.DAT será copiado para o acionador B: com o mesmo nome.

O comando COPY sempre informa quantos foram os arquivos copiados. Observe que não há necessidade de fornecer o nome do arquivo no disco de destino, apenas o nome do acionador. Isto se deve ao fato de o comando COPY presumir que você quer que o arquivo novo tenha o mesmo nome do antigo.

Havendo interesse em alterar o nome do arquivo enquanto o MSX-DOS faz a cópia no disco de destino, basta acrescentar o novo nome ao comando COPY.

Por exemplo:

```
A> COPY OPENACCT.DAT B: NEWACCT.DAT
        1 File(s) copied
```

Neste caso, além de copiar OPENACCT.DAT de A: para B:, deu ao arquivo um novo nome (NEWACCT.DAT) no disco B:.

Lembre-se de que é possível utilizar caracteres-chave no nome do arquivo para copiar vários arquivos.

Por exemplo,

```
A> COPY *.BAT B:
```

Este comando copia todos os arquivos com a extensão BAT do disco A: para B:.

Podemos ainda utilizar o comando COPY para duplicar um arquivo no mesmo disco. Entretanto, a cópia deve ter outro nome uma vez que o MSX-DOS não permite a existência de dois arquivos de mesmo nome em um mesmo disco.

Para fazer uma cópia do arquivo JUDGES.TXT:

```
A> COPY JUDGES.TXT JUDGES.NEW
     1 File(s) copied
```

Como vemos, o disco de destino, não sendo especificado significa que a cópia permanece no mesmo disco. Agora temos uma nova cópia do arquivo JUDGES.TXT (denominado JUDGES.NEW) no mesmo disco.

Quando copiamos de um disco para o disco assumido, sem alterar o nome do arquivo, não é necessário incluir o nome do disco de destino.

Por exemplo, caso A: seja o acionador assumido e desejamos copiar o arquivo CASHFLOW.RPT de B: para A:, damos o comando:

```
A> COPY B: CASHFLOW.RPT
```

O comando COPY duplica o arquivo no acionador assumido.

O comando COPY pode ainda ser utilizado quando queremos que o MSX-DOS leia ou escreva nos dispositivos de hardware. É bastante comum quando se deseja imprimir um arquivo. Para tanto, utiliza-se o nome do dispositivo em lugar do arquivo, e o comando COPY enxerga o dispositivo como se fosse um arquivo. Os nomes de dispositivos foram discutidos no Capítulo 18. Normalmente não utilizamos dispositivos com o comando COPY.

Querendo imprimir o arquivo OPERWX.FOR na impressora, dê o comando:

```
A> COPY OPERWX.FOR PRN
```

Quando um dispositivo é definido como fonte, o comando COPY continua a lê-lo até que seja digitado CONTROL+Z acompanhado por um RETURN (o CONTROL+Z é denominado "marcação de fim-de-arquivo). Este método de determinar o fim de um arquivo pode parecer arbitrário; entretanto, quando o comando COPY é utilizado com um dispositivo, ele continuará a ler informação do dispositivo até que encontre um "fim-de-arquivo".

Querendo digitar texto diretamente no arquivo SHORT-MEM, tem-se de utilizar o seguinte comando:

```
A> COPY CON SHORT.MEM
```

O dispositivo chamado CON representa o teclado e a tela. Agora tudo que é digitado será colocado diretamente no arquivo. Observe que a marca de solicitação do MSX-DOS não reaparece. Para terminar, após digitar algumas linhas, opere o CONTROL+Z seguido pela tecla RETURN.

Utilizando COPY para combinar arquivos. A segunda forma geral da sintaxe do comando COPY é:

```
COPY fonte 1 + fonte2 destino
```

Utilize esta forma para combinar arquivos de ponta a ponta ou concatená-los. Podemos concatenar qualquer número de arquivos em um único arquivo de destino.

Por exemplo, desejando fazer um arquivo chamado STAFF.TXT, que seja uma cópia de JANE.TXT seguido por uma cópia de SYLVIA.TXT, dê o seguinte comando:

```
A> COPY JANE.TXT + SYLVIA.TXT STAFF.TXT
JANE.TXT
SYLVIA.TXT
        1 File(s) copied
```

À medida que o comando COPY adiciona um arquivo ao de destino ele lista este arquivo. (Neste caso o destino está no mesmo disco.)

Utilizando o argumento /V com COPY. Poderá haver conseqüências desastrosas, se o MSX-DOS copiar incorretamente os dados. Mesmo que o MSX-DOS dificilmente erre ao copiar, grande parte dos usuários gosta de se sentir seguro a este respeito. Por isto, sempre deve ser utilizado o argumento /V depois de um comando COPY.

Tal procedimento torna o comando mais lento, mas assegura que as cópias feitas estão certas. A certeza de que seu arquivo foi copiado corretamente compensa o tempo que levaria para o MSX-DOS reler aquilo que foi escrito. O argumento /V é colocado no fim do comando COPY.

Por exemplo:

```
A> COPY PURCH.ACT B: /V
        1 File(s) copied
```

Assim fica garantido que a cópia do arquivo PURCH.ACT, feita pelo MSX-DOS para o disco B:, está correta.

OUTRAS UTILIZAÇÕES

Dois outros argumentos, /A e /B, raramente são utilizados com o comando COPY porque são confusos. São utilizados para distinguir entre arquivos ASCII e binários (por isto o A e B). Enquanto um arquivo ASCII contém apenas texto normal, como uma carta ou um memorando, um arquivo binário contém caracteres especiais. Os programas e bancos de dados, normalmente são arquivos binários. Os argumentos /A e /B são utilizados apenas quando copiamos arquivos que não se enquadram facilmente nestas definições. Leia o mánual de referência do MSX-DOS, para maiores detalhes a respeito destes argumentos.

REGRAS

Não é possível sobrepor um arquivo com ele mesmo. Caso você tente copiar um arquivo no mesmo disco, sem alterar seu nome, o MSX-DOS emite a mensagem "File cannot be copied onto itself".

AVISOS E ERROS COMUNS

O erro mais comum na utilização do comando COPY é a reversão entre os arquivos de origem e destino. A possibilidade de isto ocorrer é maior caso você esteja familiarizado com o sistema operacional CP/M, que exige que o destino apareça antes da fonte na linha de comando.

Quando você tentar copiar um arquivo que não existe, o MSX-DOS emite uma mensagem de erro.

Por exemplo, caso você tente copiar o arquivo TETHER. PRG e este não existe:

```
A> COPY TETHER.PRG B:
TETHER.PRG File not found
        0 File(s) copied
```

MSX-DOS emite a mensagem de erro "File not found".

Lembre-se da opção /V, ou colocar VERIFY em ON, ao copiar arquivos. Assim evitamos erros que poderiam ocorrer quando os arquivos são copiados incorretamente.

Esteja certo de que o dispositivo está posicionado corretamente, antes de nomeá-lo como fonte em um comando COPY. É especialmente importante quando a fonte é a porta de comunicação. Lembre-se de que o comando COPY espera indefinidamente pelos dados que devem ser enviados pelo dispositivo. Uma vez iniciada a recepção de dados, o comando COPY continua a leitura até encontrar um sinal de fim-de-arquivo. Caso seja fornecido, acidentalmente, o nome errado do dispositivo no comando e o dispositivo não responder, o computador deverá ser inicializado.

SINTAXE

À primeira vista a sintaxe do comando COPY poderá parecer estranha; ela é porém muito fácil de ser utilizada já que os argumentos /A e /B raramente são utilizados. São fornecidas em diversas linhas as variações sintáticas do comando COPY, que (como a maioria dos comandos do MSX-DOS) sempre é dado em apenas uma linha.

Copiando um arquivo normalmente:

```
COPY [{/A}]esparq [{/A|/B}]
{d:|esparq2} [{/A|/B}] [/V]
```

Concatenando um arquivo:

```
COPY [{/A|/B}]esparq1a [{/A|/B}]
[+esparq1b] [{/A|/B}]... [/V]
{d:|esparq2} [{/A|/B}] [/V]
```

Copiando de ou para um dispositivo:

```
COPY {esparq1 | dispositivo1} {esparq2 | dispositivo2}
```

Em cada uma destas variações sintáticas, esparq1 é a fonte de cópia e esparq2 o destino da cópia. O comando COPY sempre copia do primeiro argumento para o segundo. Podemos utilizar também os caracteres-chave no nome dos arquivos para copiar vários arquivos.

{/A|/B} indica que podem ser utilizados /A ou /B com qualquer especificador de arquivo. O significado de /A ou /B depende se você os denomina como arquivos de origem ou destino. O /A e /B podem ser colocados antes ou depois do fonte "esparq".

/V faz o comando COPY verificar se os dados copiados no disco de destino estão iguais aos do disco de origem.

O argumento "dispositivo" representa o nome de qualquer um dos dispositivos do MSX-DOS, como a impressora LPT1, COM1 da porta de comunicação, e assim em diante. Estes dispositivos e os nomes dos dispositivos MSX-DOS foram discutidos no Capítulo 18.

Comandos de Saída de Arquivo

Esta seção trata do comando TYPE.

Comando TYPE

O comando TYPE apresenta um arquivo de texto na tela.

UTILIZAÇÕES COMUNS

O comando é muito fácil de ser utilizado. Basta dar o nome do comando seguido pelo nome do arquivo:

```
TYPE esparq
```

O MSX-DOS vai "imprimir" o arquivo na tela.

Por exemplo, para ver o arquivo LEADSRPT.TXT na tela, dê o comando:

```
A> TYPE LEADSRPT.TXT

                    Relatório de Vendas
                      José Antonio

Geração de novas sugestões:
```

Estes são comandos seqüenciais. Significa que ao utilizar um destes comandos, um arquivo pode ser visto, seqüencialmente, apenas do começo ao fim; não é possível retroceder e reler uma parte do arquivo. Neste sentido os comandos são restritos, já que as pessoas gostam de percorrer os arquivos durante a leitura e reexaminar a informação.

Um modo de você conseguir maior flexibilidade é com a utilização de um editor de texto ou processador de palavras que permitem apresentar o conteúdo de um arquivo na tela. Praticamente todo editor de texto permite que se veja um arquivo de texto e que haja movimentação para frente e para trás no arquivo. Assim, para ler um arquivo devemos utilizar o programa de processamento de palavras, exceto quando os arquivos de interesse são curtos ou se deseja uma cópia impressa deles.

Com os comandos discutidos nesta seção não é possível apresentar programas ou outros arquivos binários. Estes arquivos têm normalmente a extensão COM, EXE, OVR e BIN. Os arquivos não apresentáveis podem ter até 256 tipos de caracteres; podem aparecer na tela como caracteres gráficos ou deixar de aparecer. De qualquer modo, as informações nestes arquivos não podem ser utilizadas simplesmente pela sua leitura.

O comando TYPE, continuamente "rola" o arquivo (isto é, indica linha por linha) na tela. Portanto, se o arquivo tiver mais do que as 24 ou 25 linhas que a tela pode apresentar, o texto estará se movendo com tamanha velocidade que se torna impossível sua leitura. Para ler o arquivo, devemos operar CONTROL+S para estacionar o texto na tela e operar CONTROL+Q para reiniciar o movimento. Para interromper o comando TYPE basta pressionar as teclas CONTROL+C para voltar ao MSX-DOS.

Quando o arquivo que está sendo digitado tem caracteres tab (criados pela operação da tecla TAB), o comando TYPE vai-se deslocar para a coluna seguinte que seja múltiplo de 8 (isto é, coluna 8, 16, 24 e assim em diante) e imprimir o texto que segue o caractere tab. Assim, na impressão do último exemplo, o primeiro caractere impresso aparece na coluna 24 porque havia quatro caracteres tab no início da linha.

AVISOS E ERROS COMUNS

O comando TYPE não impede que digitemos programas de arquivos e arquivos binários na tela. Entretanto, caso o comando TYPE seja utilizado nestes arquivos provavelmente teremos anarquia.

SINTAXE

A sintaxe do comando TYPE é

```
TYPE esparq
```

O argumento "esparq" representa o arquivo a ser digitado. Os caracteres-chave não podem ser utilizados nesta especificação de arquivo.

Comandos para Manutenção de Discos

Esta seção discute os comandos DIR e FORMAT.

Comando DIR

O comando DIR lista os arquivos de um disco fornecendo informações sobre seu tamanho e a data da última atualização.

UTILIZAÇÕES COMUNS

A utilização mais comum do comando DIR é a listagem dos arquivos que existem no disco assumido. A sintaxe geral é

```
DIR [d:]
```

Por exemplo, para entrar com o comando DIR, digite

```
A> DIR
CALC        EXE    25600    5-20-84
DRIVER      EXE    32420    1-27-84
SALES84     RPT     7168    8-03-84
MAKEFILE    BAT     1664    4-01-84
TOTSALES    DAT     5888    3-08-84
MARTY       DAT     1280    3-08-84
RDBMS       COM    39936    4-27-84
RDBMS       OVL    33120    4-27-84
        9 File(s)  11264 bytes free
```

O MXS-DOS apresenta uma listagem dos arquivos como está mostrada aqui.

A listagem DIR contém diversos elementos que você pode não reconhecer:

- As primeiras colunas da listagem DIR apresentam o nome e a extensão do arquivo. Nesta listagem, estas duas partes do especificador de um arquivo não estão separadas com um ponto;
- A terceira coluna contém o tamanho do arquivo, em bytes;
- As duas últimas colunas fornecem a data e hora em que o arquivo foi atualizado pela última vez. Se o arquivo nunca foi atualizado, estas colunas fornecem a data e hora de sua criação. Alguns computadores não apresentam quando o registro foi criado. Caso o seu computador não tenha um calendário em tempo real e você não coloca a data e hora como um procedimento rotineiro ao inicializar o computador, esta data e horário poderão estar incorretos;
- A última linha mostra quantos arquivos existem no disco e quantos bytes estão disponíveis para outros registros.

Pela especificação de um outro disco é possível ver seu diretório.

Por exemplo:

```
A> DIR B:
```

Com este comando o MSX-DOS apresenta o diretório do disco B: na tela.

Utilizando DIR em arquivos individuais. O comando DIR permite que visualizemos arquivos individuais. Podemos conseguir também a listagem de um grupo de arquivos pela utilização de caracteres-chave nos nomes dos arquivos.

Por exemplo, vamos supor que desejássemos ver o arquivo MARTY.DAT. Daríamos o comando:

```
A> DIR MARTY.DAT
MARTY   DAT  1280  3-08-84
        1 File(s)  11264  bytes free
```

Havendo interesse em ver todos os arquivos com extensão DAT, devemos dar o comando

```
A> DIR *.DAT
TOTSALES   DAT   5888   3-08-84
MARTY      DAT   1280   3-08-84
        2 File(s)    11264  bytes free
```

A utilização do caractere-chave (*) para o nome ou extensão do arquivo é tão comum que o comando DIR permite que seja excluído.

Por exemplo, o comando anterior (DIR *.DAT) poderia ser dado como:

```
A> DIR .DAT
```

O caractere-chave é desnecessário. Entretanto, o ponto antes da extensão é necessário, porque informa ao MSX-DOS que o nome é uma extensão.

Modificando a listagem DIR com argumentos. O comando DIR pode ser utilizado com dois argumentos, /W e /P. Caso o diretório tenha um grande número de arquivos e você não está preocupado com o tamanho ou deseja atualizar a informação, o argumento /W pode ser utilizado para simplificar a lista.

Por exemplo, observe a alteração nos dois próximos diretórios:

```
A> DIR
CALC          EXE    25600     5-20-84
DRIVER        EXE    32420     1-27-84
SALES84       RPT     7168     8-03-84
MAKEFILE      BAT     1664     4-01-84
TOTSALES      DAT     5888     3-08-84
MARTY         DAT     1280     3-08-84
RDBMS         COM    39936     4-27-84
RDBMS         OVL    33120     4-27-84
          9 File(s)    11264   bytes free
```

```
A> DIR /W
CALC        EXE     DRIVER      EXE
SALES84     RPT     MAKEFILE    BAT
TOTSALES    DAT     MARTY       DAT
RDBMS       COM     RDBMS       OVL
          9 File(s)  11264  bytes free
```

O argumento /P fará a listagem do diretório parar ao fim de cada página. A mensagem "Strike a key when ready..." aparece no fim da tela. O comando DIR aguarda até que seja pressionada uma tecla antes de continuar.

SINTAXE

A sintaxe do comando DIR é

```
DIR [d: | esparq] [/P] [/W]
```

Quando o comando DIR é dado sem argumentos, é feita a listagem do disco assumido. Quando é especificado d:, será apresentado o diretório deste disco; quando é fornecido "esparq" (que pode conter caracteres-chave) somente serão listados os arquivos que fizerem parte desta "esparq".

O argumento /W apresenta os arquivos em cinco colunas na tela, mas não mostra o tamanho do arquivo ou a última atualização. O argumento /P faz a listagem parar ao final de cada página.

Comando FORMAT

O comando FORMAT prepara o disco (rígido ou flexível) para ser utilizado com o MSX-DOS. Alguns fornecedores de discos rígidos para computador exigem que se utilize comandos diferentes para formatar seus discos.

UTILIZAÇÕES COMUNS

Os disquetes, quando adquiridos normalmente, não estão formatados; o comando FORMAT deve ser utilizado antes que sejam feitas tentativas de se escrever arquivos no disco. (No Capítulo 17 foram descritos alguns dos conceitos na formatação de discos.) O comando deve ser utilizado com cuidado, uma vez que a formatação do disco apaga toda informação armazenada nele. Utilize o comando FORMAT na preparação dos discos para o MSX-DOS.

A sintaxe geral do comando FORMAT é:

```
FORMAT
```

AVISOS E ERROS COMUNS

Novamente alertamos: lembre-se de que o comando FORMAT apaga toda a informação do disco destinatário. Por este motivo, ao formatar um disco que já contenha informação, tenha certeza de que esta informação não será mais necessária.

SINTAXE

A sintaxe do comando de formatação é:

FORMAT

Preparação do Sistema

Esta seção apresenta os comandos DATE, TIME e MODE. Estes comandos são utilizados para colocar diversas "identificações de sistema" no MSX-DOS. Consiste em informações internas, mantidas pelo MSX-DOS, que o informa como agir. Apesar de estes comandos não serem de conhecimento obrigatório, são fáceis de ser aprendidos.

Toda vez que o MSX-DOS é inicializado, o sistema assume uma situação preestabelecida. Quando esta situação não for de seu agrado pode alterá-la com os comandos desta seção. Uma vez modificada, os comandos poderão ser incluídos em um arquivo AUTOEXEC.BAT que altera a situação automaticamente. Por exemplo, poderia alterar o sinal preestabelecido A> para uma cadeia de caracteres.

Comando DATE

O comando DATE coloca a data que o MSX-DOS mantém na RAM e utiliza na atualização dos arquivos. Age como a data que se vê ao inicializar o MSX-DOS.

UTILIZAÇÕES COMUNS

Você pode fornecer a data na linha de comando ou fazer o comando DATE solicitá-la de você.

A sintaxe geral é:

DATE [nova-data]

Note que este comando coloca a data interna que o MSX-DOS mantém na RAM, e não a data de um arquivo colocada quando o arquivo é escrito no disco).

Por exemplo, para fornecer uma nova data na linha de comando, insira:

```
A> DATE 4-12-85
```

Para que se tenha a solicitação do comando DATE, insira:

```
A> DATE
Current date is Fri 2-11-85
Enter new date:
```

Podemos ainda deixar de fornecer a data, pressionando a tecla RETURN, e mantendo a data existente antes do comando.

Alguns computadores têm relógios internos. Em vez de colocar a data e o horário manualmente, o fabricante poderá ter incluído um programa que pode ser processado para ler o relógio interno e colocar automaticamente a data e o horário.

REGRAS

A data deve ser digitada como três números separados por (-) ou (/). A regra completa para a formatação da data foi dada no Capítulo 18 em "Inicializando o MSX-DOS".

AVISOS E ERROS COMUNS

Alguns programas em BASIC alteram a data. Caso você perceba, após o processamento de um programa, que a data foi colocada com erro, pode restaurá-la com o comando DATE.

SINTAXE

A sintaxe para o comando DATE é:

```
DATE [mm-dd-yy]
```

onde mm-dd-yy é o mês, dia e ano. Estes parâmetros poderão ser dados tanto com (-) ou (/) entre os números. Se não for fornecida uma data na linha de comando, o comando DATE pedirá uma.

Comando TIME

O comando TIME coloca o horário que o MSX-DOS mantém na RAM e utiliza na atualização dos arquivos. Age como o horário que se vê ao inicializar o MSX-DOS. O comando TIME não opera na maioria dos computadores MSX.

UTILIZAÇÕES COMUNS

A sintaxe geral do comando TIME é:

```
TIME [novo-horário]
```

O horário tanto pode ser dado na linha de comando como pode ser solicitado pelo comando TIME.

Por exemplo, para fornecer um novo horário na linha de comando, insira:

```
A> TIME 9: 41
```

Para que se tenha a solicitação do comando TIME, insira:

```
A> TIME
Current time is 9:41:03
Enter new time:
```

Não havendo interesse em fornecer o tempo basta operar RETURN, mantendo o mesmo horário de antes do comando.

Alguns computadores têm relógios internos. Em vez de colocar a data e o horário manualmente, o fabricante poderá ter incluído um programa que pode ser processado para ler o relógio interno e colocar automaticamente a data e o horário.

REGRAS

A única forma aceitável para o horário é em horas, minutos e segundos separados por dois-pontos (:). As regras para a formatação do horário foram dadas na seção denominada "Inicializando o MSX-DOS", no Capítulo 18.

AVISOS E ERROS COMUNS

Alguns programas em BASIC alteram o relógio para temporizar acontecimentos. Quando isto é feito normalmente é colocado em meia-noite (00:00:00). Assim, observando que há erro após processar um programa, o horário pode ser acertado com o comando TIME.

SINTAXE

A sintaxe do comando TIME é:

```
TIME [hh:mm:ss.xx]
```

onde hh:mm:ss.xx representam as horas, minutos, segundos e centésimos de segundo. O horário deve ser fornecido com vírgulas (e ponto decimal, quando houver muita precisão). Os segundos e centésimos de segundos são opcionais.

Comando MODE

O comando MODE estabelece o número de caracteres em cada linha da tela, da mesma forma que o comando WIDTH no MSX-BASIC.

UTILIZAÇÕES COMUNS

É pouco provável que você queira alterar a largura da tela, uma vez que há interesse em ter o maior número de colunas possível. A sintaxe geral é:

```
MODE largura
```

onde "largura" varia de 1 até 40.

Comandos de Arquivos-em-Lote

Nesta seção são discutidos os comandos PAUSE e REM. Estes comandos são denominados "comandos de arquivos-em-lote", porque são utilizados apenas dentro deste tipo de registro.

No Capítulo 18 você aprendeu como escrever arquivos-em-lote. Em resumo podemos dizer que arquivos-em-lote são arquivos que contêm uma lista de comandos que normalmente seriam fornecidos ao MSX-DOS. Os comandos nestes arquivos podem incluir comandos MSX-DOS, os nomes de programas aplicativos ou o nome de outro arquivo-em-lote. Você também tomou conhecimento sobre o arquivo AUTOEXEC.BAT, um arquivo-em-lote que é automaticamente processado quando o sistema é inicializado.

Agora, que você já leu a respeito da maioria dos comandos do MSX-DOS, deve ter descoberto alguns deles que deseja incluir no arquivo AUTOEXEC.BAT.

Comando PAUSE

O comando PAUSE apresenta a mensagem "Strike a key when ready" (pressione uma tecla quando estiver pronto), e aguarda a operação da tecla para poder continuar.

UTILIZAÇÕES COMUNS

O comando PAUSE tem muita utilidade quando as operações que estão para serem executadas dependem de algum preparo mecânico.

Caso você prepare seu arquivo-em-lote para imprimir o arquivo PAYROLL.RPT na impressora, pode haver interesse que o usuário verifique se a impressora está ligada. Se o apontador estiver no arquivo PR1.DAT, o arquivo-em-lote conteria:

PRNPAY.BAT

```
TYPE PR1.DAT
PAUSE
COPY PAYROLL.RPT PRN
```

Este arquivo-em-lote permite que verifiquemos se a impressora está pronta, antes de enviar arquivos para ela. Enquanto estiver utilizando o comando PAUSE, você pode fazer o que desejar exceto utilizar o teclado. Para continuar o arquivo-em-lote, basta acionar uma tecla qualquer.

Outra utilização comum para o comando PAUSE é permitir que o usuário coloque um disquete novo no acionador.

SINTAXE

O comando PAUSE é colocado da seguinte forma:

```
PAUSE
```

Faz parte de um arquivo-em-lote.

Comando REM

O comando REM é utilizado para colocar comentários em um arquivo-em-lote. Não fornece resultado algum. É idêntico ao comando REM do MSX-BASIC.

UTILIZAÇÕES COMUNS

O comando REM é utilizado para colocar anotações em um arquivo-em-lote. Este comando pode ser utilizado para descrever o conteúdo de um arquivo-em-lote. Ao escrever um arquivo-em-lote, pode haver interesse em deixar anotações sobre a finalidade do arquivo.

É sempre uma boa idéia colocar ao menos um comando REM em um arquivo-em-lote, lembrando porque escreveu o arquivo.

SINTAXE

A sintaxe do comando REM é:

```
REM comentário
```

O argumento "comentário" é qualquer texto. O comando REM pode ser colocado em qualquer lugar do arquivo-em-lote.

Comando BASIC

O único comando MSX-DOS que falta é o comando BASIC, que permite o processamento do MSX-BASIC. Sua sintaxe é simples:

```
BASIC
```

Lembre-se de que o seu computador pode vir acompanhado de outros programas aplicativos. Poderão ser vistos com a extensão de arquivo .COM ou .EXE. Processe-os dando o nome do arquivo da aplicação.

PARTE

4

APÊNDICES

APÊNDICE

A

REFERÊNCIA AO MSX-BASIC

Na Figura A-1 estão relacionados os comandos MSX-BASIC e na Figura A-2 estão relacionadas as funções do MSX-BASIC.

AUTO *partida,incremento*
BEEP
BLOAD *nomarq,opçãoprocessamento,compensação*
BSAVE *nomarq,endinicial,endfinal,endprocessamento*
CALL (ou _) *nomeprograma*
CALL FORMAT
CALL SYSTEM
CIRCLE STEP(x,y), *raio,cor,anguloinicial,angulofinal,aspecto*
CLEAR *tamanhosérie,espaçolivre*
CLOAD *nomarq*
CLOSE *numeroarquivo*
CLS
COLOR *primeiroplano,segundoplano,contorno*
CONT
COPY *nomearq1* TO *nomearq2*
CSAVE *nomearq,velocidade*
DATA *valores*
DEF FN *var(args) = expressão*
DEF *valtipo faixa*
DEF USR *dígito = n*
DELETE *faixa*
DIM *variáveis*

Figura A-1 Estruturas dos comandos MSX-BASIC.

DRAW *gm1série*
END
ERASE *matrizes*
ERROR *n*
FIELD #*n,larg1* AS *var1*$,...
FILES *nomarq*
FOR *var* = *valinic1* TO *valfina1* STEP *incremento*
GET #*n,numerorec*
GOSUB *linha*
GOTO *linha*
IF *condição* THEN *declaração*
IF *condição* THEN *declaração1* ELSE *declaração2*
INPUT *apontador, vars*
INTERVAL OFF/ON/STOP
KEY LIST
KEY *n,série*
KEY(n) OFF/ON/STOP
KILL *nomearq*
LET *var* = *expressão*
LFILES *nomarq*
LINE INPUT *"ponteiro";var*$
LINE INPUT #*n;var*$
LINE STEP *(x1,y1)* – STEP *(x2,y2),cor,B*
LINE STEP *(x1,y1)* – STEP *(x2,y2),cor,BF*
LIST *faixa*
LLIST *faixa*
LOAD *nomarq,R*
LOCATE *x,y,cursor*
LPRINT *vars*
MAXFILES = *n*
MERGE *nomearq*
MOTOR OFF
MOTOR ON
NAME *nomearq1* AS *nomearq2*
NEW
NEXT *var*
ON ERROR GOTO *linha*
ON INTERVAL = *n* GOSUB *linha*
ON KEY GOSUB *linhas*
ON *linha* GOSUB *linhas*
ON *linha* GOTO *linhas*
ON SPRITE GOSUB *linha*
ON STOP GOSUB *linha*

ON STRIG GOSUB *linhas*
OPEN *dispositivo* FOR *modo* AS #*n*
OUT *end,valor*
PAINT STEP *(x,y),cor,contorno*
PLAY *mm1série*
POKE *endereço,valor*
PRESET STEP *(x,y),cor*
PRINT # n, USING *formatsérie,vars*
PSET STEP *(x,y),cor*
PUT #*n,recnúmero*
PUT SPRITE *n*, STEP *(x,y),cor,imagem*
READ *vars*
REM *texto*
RENUM *novalin,antigalin,incremento*
RESTORE *linha*
RESUME *modo*
RETURN
RUN *nomearq*
SAVE *nomearq,A*
SCREEN *modo,tamanhosprite,clic,velfita,tipoimp*
SOUND *registro,valor*
SPRITE OFF/ON/STOP
STOP
STOP OFF/ON/STOP
STRIG OFF/ON/STOP
SWAP *var1,var2*
TROFF
TRON
VPOKE *end,valor*
WAIT *end,endexp,xorexp*
WIDTH *n*

ABS
ASC
ATN
BASE
BIN$
CDBL
CHR$
CINT
COS
CSNG
CVD
CVI
CVS
DSKF
EOF
EXP
FIX
FRE
HEX$
INP
INPUT$
INSTR
INT
LEFT$
LEN
LOF
LOG
LPOS
LSET$
MID$
MKD$
MKI$
MKS$
OCT$
PAD
PDL
PEEK
PLAY
POINT
POS
RIGHT$
RND
RSET$
SGN
SIN
SPACE$
SPC
SPRITE$
SQR
STICK
STR$
STRIG
STRING$
TAB
TAN
USR
VAL
VPEEK

Figura A-2 Funções do MSX-BASIC.

APÊNDICE

B

REFERÊNCIAS AO MSX-DOS

A Figura B-1 mostra os comandos do MSX-DOS.

BASIC *nomearq*
COPY *nomearq nomearq2* /V
DATE
DEL *nomearq*
DIR *nomearq* /P /W
FORMAT
MODE *largura*
PAUSE
REM *texto*
REN *nomearq1 nomearq2*
TIME
TYPE *nomearq*

Figura B-1 Estrutura dos comandos MSX-DOS.

APÊNDICE

C

TABELA ASCII

A Tabela C-1 mostra os caracteres que correspondem a cada número na seqüência ASCII. Observe que alguns dos caracteres gráficos são malformados em uma tela normal: isto decorre do fato de sua apresentação deve ser feita em uma tela de 32 colunas. Caso seja dado o comando:

```
SCREEN 1
```

a aparência dos caracteres corresponde ao desta tabela. Para a impressão da maioria dos caracteres, pode ser utilizado o comando:

```
PRINT CHR$(n);
```

onde n corresponde ao número do caractere ASCII. Por exemplo, para imprimir o caractere grego alfa (ASCII 224), dê o comando:

```
PRINT CHR$(224);
```

É difícil imprimir os caracteres que têm um valor ASCII entre 1 e 31. Para tanto, devemos primeiro imprimir o caractere com o valor ASCII de 1 para em seguida imprimir o caractere desejado com um acréscimo de 64 ao seu valor. É melhor explicado com um exemplo. Para imprimir outro, que tem um valor ASCII de 3, dê o comando:

```
PRINT CHR$(1); CHR$(3+64);
```

Observe que isto é verdade apenas para os caracteres com valores de ASCII entre 1 e 31.

O caractere ASCII de número 0 nada imprime, e o caractere ASCII de número 255 imprime o que porventura está embaixo do cursor na sua apresentação na tela.

Tabela C-1 A Tabela ASCII.

Número	Caractere	Número	Caractere
1	☺	24	┌
2	☻	25	┐
3	♥	26	└
4	♦	27	┘
5	♣	28	╳
6	♠	29	╱
7	•	30	╲
8	◘	31	┼
9	○	32	
10	◙	33	!
11	♂	34	"
12	♀	35	#
13	♪	36	$
14	♫	37	%
15	☼	38	&
16	┼	39	'
17	┴	40	(
18	┬	41	)
19	┤	42	*
20	├	43	+
21	┼	44	,
22	│	45	E
23	─	46	

Tabela C-1 A Tabela ASCII. (Cont.)

Número	Caractere	Número	Caractere
47		70	F
48	0	71	G
49	1	72	H
50	2	73	I
51	3	74	J
52	4	75	K
53	5	76	L
54	6	77	M
55	7	78	N
56	8	79	O
57	9	80	P
58	:	81	Q
59	;	82	R
60	<	83	S
61	=	84	T
62	>	85	U
63	?	86	V
64	@	87	W
65	A	88	X
66	B	89	Y
67	C	90	Z
68	D	91	[
69	E	92	\

Tabela C-1 A Tabela ASCII. (Cont.)

Número	Caractere	Número	Caractere
93	]	116	t
94	^	117	u
95	_	118	v
96	`	119	w
97	a	120	x
98	b	121	y
99	c	122	z
100	d	123	{
101	e	124	|
102	f	125	}
103	g	126	~
104	h	127	
105	i	128	Ç
106	j	129	ü
107	k	130	é
108	l	131	â
109	m	132	ä
110	n	133	à
111	o	134	å
112	p	135	ç
113	q	136	ê
114	r	137	ë
115	s	138	è

Tabela C-1 A Tabela ASCII. (Cont.)

Número	Caractere	Número	Caractere
139	ï	162	ó
140	î	163	ú
141	ì	164	ñ
142	Ä	165	Ñ
143	Å	166	ª
144	É	167	º
145	æ	168	¿
146	Æ	169	⌐
147	ô	170	¬
148	ö	171	½
149	ò	172	¼
150	û	173	¡
151	ù	174	<<
152	ÿ	175	>>
153	Ö	176	Ã
154	Ü	177	ã
155	¢	178	Ĩ
156	£	179	ĩ
157	¥	180	Õ
158	Pt	181	õ
159	ƒ	182	Ũ
160	á	183	ũ
161	í	184	Ỽ

Tabela C-1 A Tabela ASCII. (Cont.)

Número	Caractere	Número	Caractere
185	ij	208	
186	¾	209	
187	~	210	
188	◊	211	
189	‰	212	
190	¶	213	
191	§	214	
192		215	
193		216	^
194		217	‡
195		218	ω
196		219	
197		220	
198		221	
199		222	
200		223	
201		224	α
202		225	β
203		226	Γ
204		227	Π
205		228	Σ
206		229	σ
207		230	μ

Tabela C-1 A Tabela ASCII. (Cont.)

Número	Caractere	Número	Caractere
231	γ	244	⌠
232	Φ	245	⌡
233	θ	246	÷
234	Ω	247	≈
235	δ	248	°
236	∞	249	∙
237	φ	250	·
238	ε	251	√
239	η	252	ⁿ
240	≡	253	²
241	±	254	■
242	≥	255	
243	≤		

APÊNDICE

D

O PADRÃO MSX

As tabelas e figuras que seguem são de interesse principalmente para os leitores com conhecimento técnico. Cobrem algumas das partes internas do computador MSX.

A Figura D-1 mostra o mapa de memória do MSX-DOS. A Figura D-2 mostra o mapa de memória do MSX-BASIC. A Figura D-3 mostra o mapa de memória do MSX-DISK BASIC. A Tabela D-1 mostra posições úteis de RAM. A Tabela D-2 mostra rotinas úteis de ROM que não exigem parâmetros. A Tabela D-3 mostra os sinais no barramento do cartucho e seu sentido.

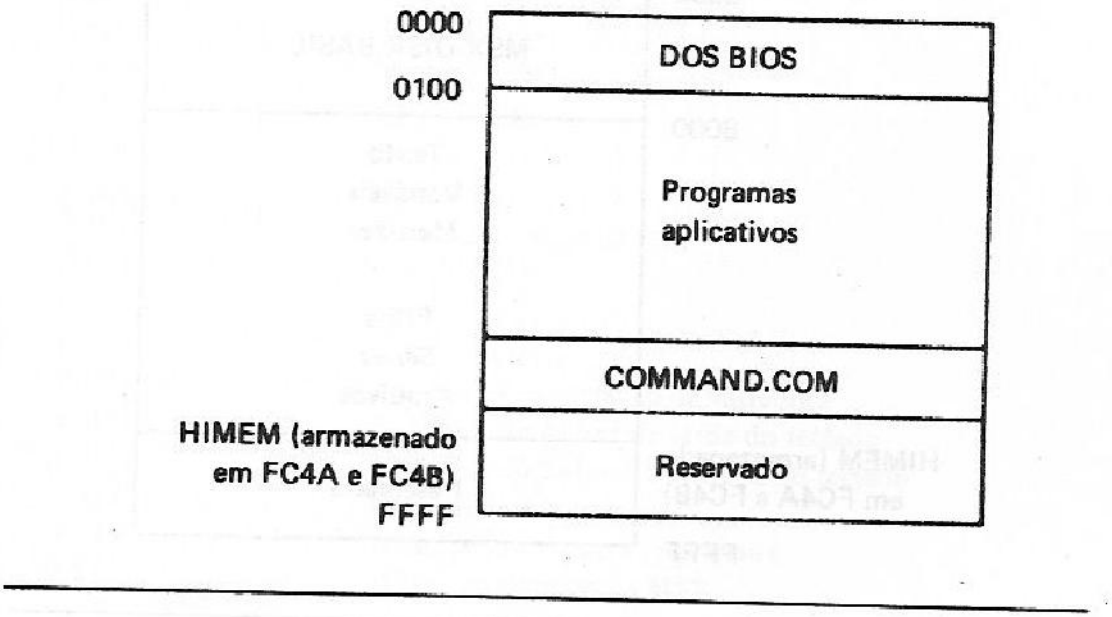


Figura D-1 Mapa de memória MSX-DOS.

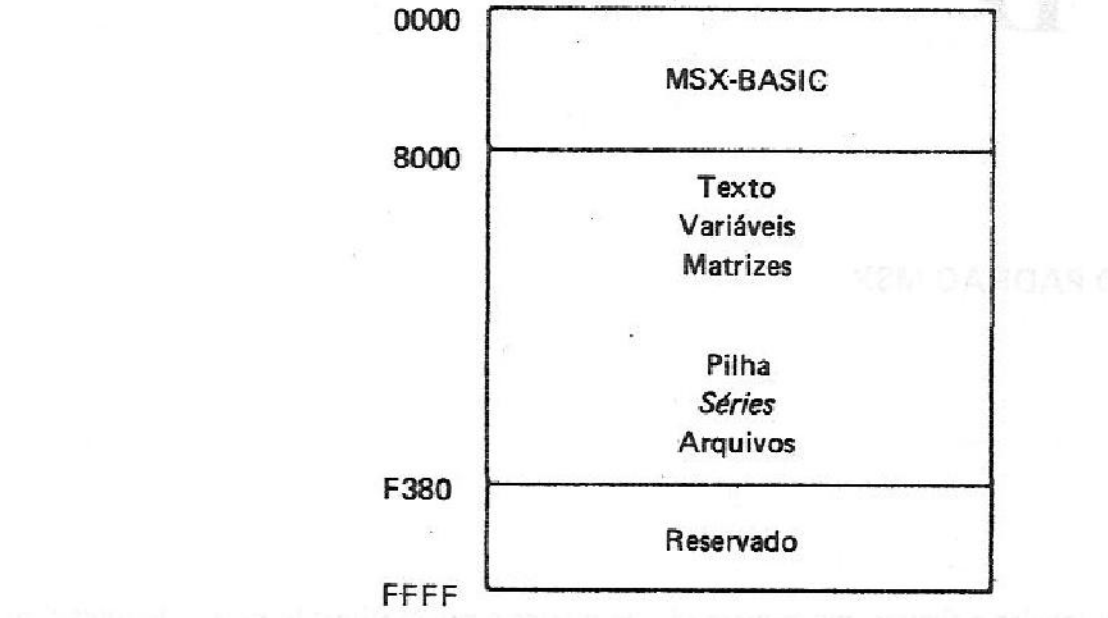


Figura D-2 Mapa de memória MSX-BASIC.

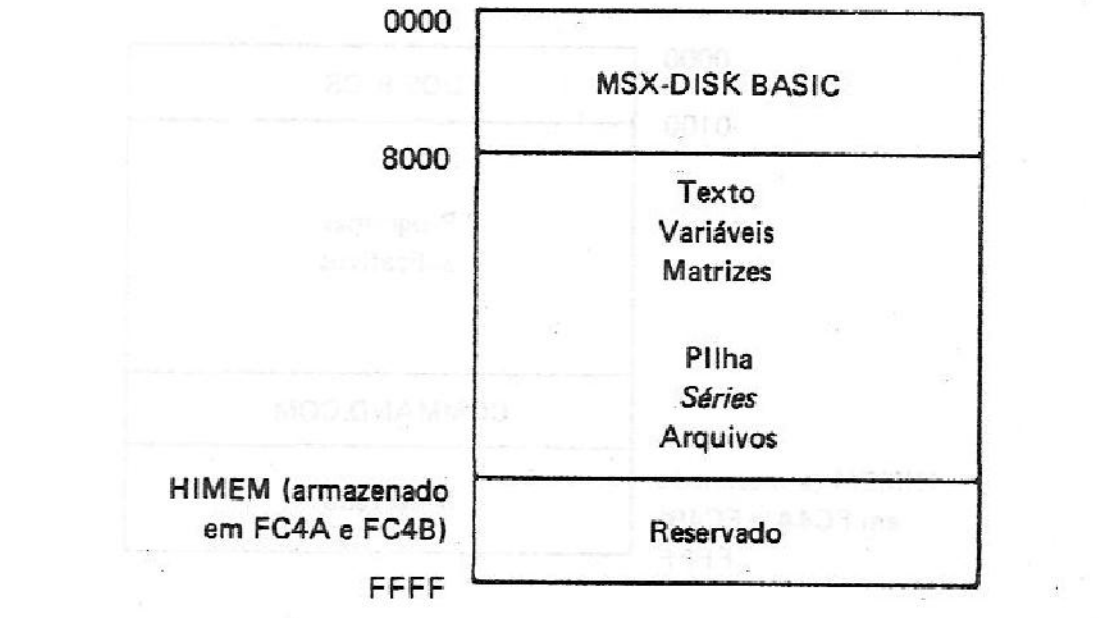


Figura D-3 Mapa de memória MSX-DISK BASIC.

Tabela D-1 Posições úteis da RAM.

Endereço	Conteúdo
F380	Rotina de leitura do conector primário
F385	Rotina de escrita no conector primário
F39A	Endereço do USR0
F39C	Endereço de USR1...
F3AC	Endereço de USR9
F3B0	Comprimento da linha
F3B3	Modo 0, tabela de nome
F3B5	Modo 0, tabela de cor
F3B7	Modo 0, padrão de caractere
F3B9	Modo 0, tabela de atributo
F3BB	Modo 0, tabela de sprite
F3BD	Modo 1, tabela de nome
F3BF	Modo 1, tabela de cor
F3C1	Modo 1, padrão de caractere
F3C3	Modo 1, tabela de atributo
F3C5	Modo 1, tabela de sprite
F3C7	Modo 2, tabela de nome
F3C9	Modo 2, tabela de cor
F3CB	Modo 2, padrão de caractere
F3CD	Modo 2, tabela de atributo
F3CF	Modo 2, tabela de sprite
F3D1	Modo 3, tabela de nome
F3D3	Modo 3, tabela de cor
F3D5	Modo 3, padrão de caractere
F3D7	Modo 3, tabela de atributo
F3D9	Modo 3, tabela de sprite
F3DB	Clic de tecla
F3DC	Posição do cursor Y
F3DD	Posição do cursor X
F3DE	Teclas de função da tela
F3E9	Cor de frente
F3EA	Cor de fundo
F3EB	Cor de borda
F3F6	Tecla de temporização de varredura
F3F8	Endereço do buffer de saída do teclado
F3FA	Endereço do buffer de entrada do teclado
F414	Número de erro
F416	"Flag" de saída para impressora
F417	"Flag" da impressora MSX

Tabela D-1 Posições úteis da RAM. (Cont.)

Endereço	Função
F418	"Flag" para impressão
F55E	Buffer de entrada do teclado
F663	Tipo de variável atual
F672	Topo da memória
F676	Topo do espaço de string
F6A3	Apontador de DATA
F6AA	"Flag" para o modo interno de AUTO
F6AB	Número de linha atual de AUTO
F6B3	Número da linha com erro
F6B5	Número atual da linha
F6B9	Número de linha do ON ERROR GO TO
F6BB	"Flag" de ON ERROR GOTO
F6C0	Próxima declaração
F6C2	Topo da área de variável
F6C4	Topo da tabela de matriz
F6C6	Fim de armazenamento
F6C8	Topo do armazenamento de DATA
F6CA	Tipo de variável iniciando com "A"
F6CB	Tipo de variável iniciando com "B" ...
F6E5	Tipo de variável iniciando com "Z"
F7BA	Número de funções utilitárias em uso
F7C4	"Flag" TRON/TROFF
F922	Base da tabela de nome atual
F924	Base da tabela de caractere atual
F926	Base da tabela de padrão-de-sprite atual
F928	Base da tabela de atributo atual
F931	Razão de aspecto para o último CIRCLE (2 bytes)
F93B	Número de pontos no último CIRCLE (2 bytes)
F975	Fila musical voz 1
F9F5	Fila musical voz 2
FA75	Fila musical voz 3
FC4A	Posição mais alta de memória disponível
FCA8	"Flag" para modo de inserção
FCA9	"Flag" para apresentação do cursor
FCAA	Estilo do cursor
FCAB	Status principal
FCAF	Modo atual de tela
FCB3	Posição X do cursor gráfico
FCB5	Posição Y do cursor gráfico

Tabela D-2 Rotinas da ROM sem parâmetros.

Endereço	Função
003B	Inicializar todos os dispositivos
003E	Inicializar todas as teclas de função
0041	Desativar mostrador de tela
0044	Ativar mostrador de tela
005F	Altera cor de tela baseado nos valores RAM
0066	Execute as funções NMI
0069	Inicializar todo os sprites
006C	Inicializar tela para modo 0
006F	Inicializar tela para modo 1
0072	Inicializar tela para modo 2
0075	Inicializar tela para modo 3
0078	Colocar a tela em modo 0
007B	Colocar a tela em modo 1
007E	Colocar a tela em modo 2
0081	Colocar a tela em modo 3
008A	Retorna ao tamanho atual de sprite
0090	Inicializa som
0099	Inicia a tarefa de fundo para PLAY
009F	Espera a digitação de caractere
00A8	Retorna status da impressora
00AE	Aceita linha do teclado
00B4	Imprime "?" e depois aceita a linha do teclado
00BA	Verifica o status da tecla SHIFT+STOP
00C0	Mesmo que BEEP (MSX-BASIC)
00C3	Limpa tela
00CC	Elimina a apresentação da tecla de função
00CF	Apresenta teclas de função
00D2	Vá ao próximo modo
00FC	Mova uma retícula para a direita
00FF	Mova uma retícula para a esquerda
0102	Mova uma retícula para cima
0108	Mova uma retícula para baixo

Tabela D-3 Sinais no "bus" do cartucho.

Pino	Nome	E/S	Descrição
1	CS1	S	Seleção do RAM 4000-7FFF
2	CS2	S	Seleção do ROM 8000-FFFF
3	CS12	S	Seleção do ROM 4000-BFFF
4	SLTSL	S	Seleção de dispositivo
5	res		Reservado
6	RFSH	S	Refrescamento
7	WAIT	E	Aguardando CPU
8	INT	E	Sinal de interrupção
9	M1	S	Tomar o sinal de ciclo da CPU
10	BUSDIR	E	Direção do bus de dados
11	IORQ	S	Solicitação de E/S
12	MERQ	S	Solicitação de memória
13	WR	S	Escrever
14	RD	S	Ler
15	RESET	S	Reposição do sistema
16	res		Reservado
17	Ag	S	Bus de endereço
18	A15	S	"
19	A11	S	"
20	A10	S	"
21	A7	S	"
22	A6	S	"
23	A12	S	"
24	A8	S	"
25	A14	S	"

Tabela D-3 Sinais no "bus" do cartucho. (Cont.)

Pino	Nome	E/S	Descrição
26	A13	S	"
27	A1	S	"
28	A0	S	"
29	A3	S	"
30	A2	S	"
31	A5	S	"
32	A4	S	"
33	D1	E/S	Bus de dados
34	D0	E/S	"
35	D3	E/S	"
36	D2	E/S	"
37	D5	E/S	"
38	D4	E/S	"
39	D7	E/S	"
40	D6	E/S	"
41	GND		Terra
42	CLOCK	S	Clock da CPU (3.58 MHz)
43	GND		Terra
44	SW1		Detector de inserção de cartucho
45	+5V		Alimentação
46	SW2		Detector de inserção de cartucho
47	+5V		Alimentação
48	+12V		Alimentação
49	SOUNDIN	E	Entrada de som
50	-12V		Alimentação

APÊNDICE

E

APRESENTAÇÃO DO TECLADO

As figuras abaixo mostram os teclados quando são pressionadas as teclas RGRAPH e LGRAPH. Na Figura E-1 temos o teclado para a tecla RGRAPH. A Figura E-2 mostra o teclado para as teclas RGRAPH e SHIFT. A Figura E-3 mostra o teclado para a tecla LGRAPH. A Figura E-4 mostra o teclado para as teclas LGRAPH e SHIFT.

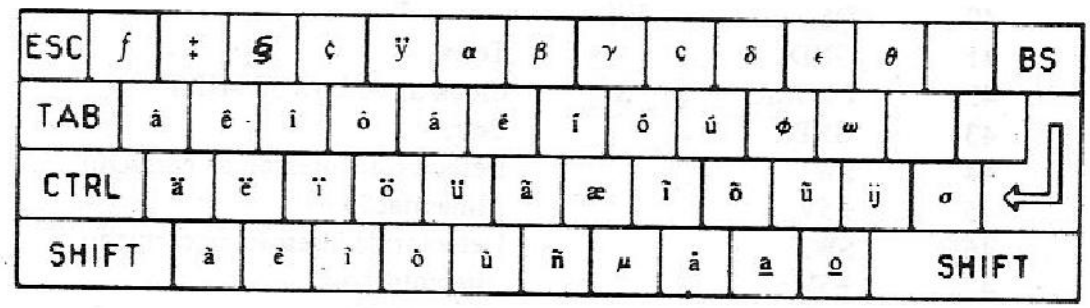


Figura E-1 Teclado para a tecla RGRAPH.

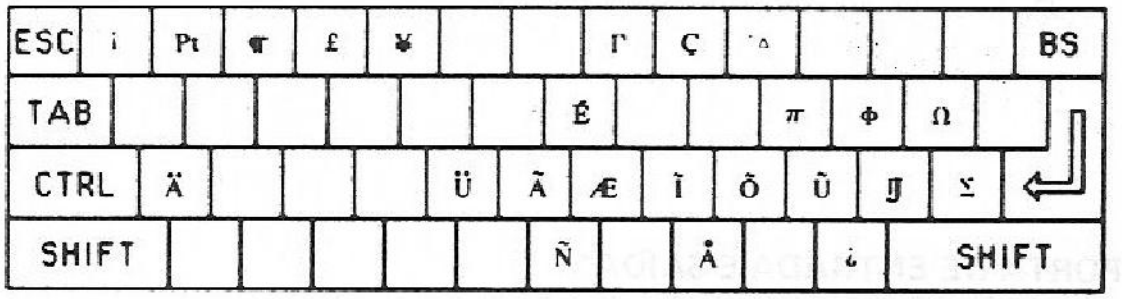


Figura E-2 Teclado para as teclas RGRAPH e SHIFT.

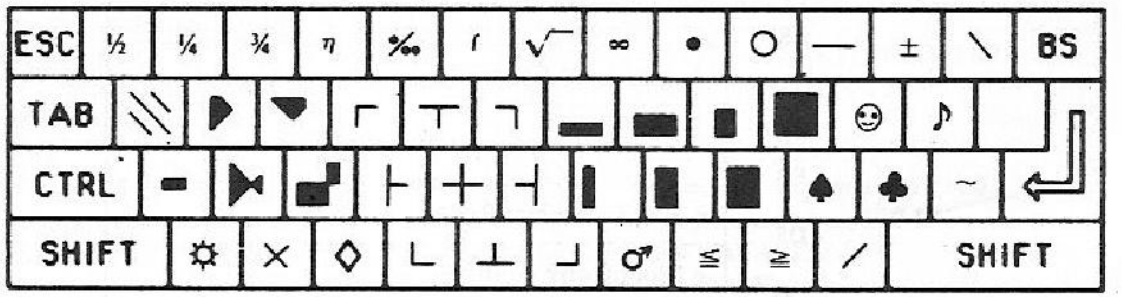


Figura E-3 Teclado para a tecla LGRAPH.

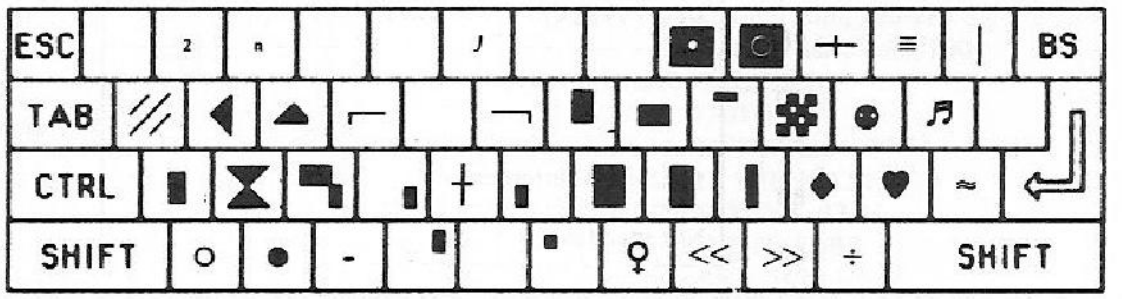
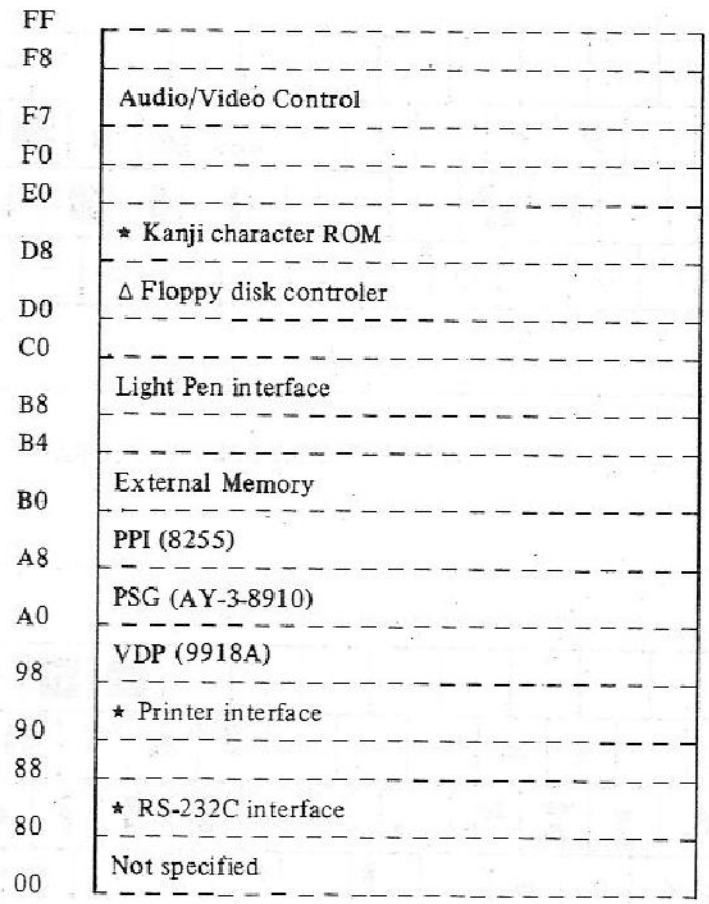


Figura E-4 Teclado para as teclas LGRAPH e SHIFT.

APÊNDICE

F

PORTA DE ENTRADA E SAÍDA*



* Preparado por: Oscar Burd e Luiz Sérgio Young

ÍNDICE ANALÍTICO

Impressão e Acabamento
GRÁFICA E EDITORA FCA
com filmes fornecidos pelo editor.
AV. HUMBERTO DE ALENCAR CASTELO BRANCO, 3972 - TEL.: 419-0200
SÃO BERNARDO DO CAMPO - CEP 09700 - SP

OUTROS LIVROS NA ÁREA

GUIAS DO USUÁRIO

Castlewitz – VisiCalc – Guia do Usuário
Curtis – WordStar – Guia do Usuário – IBM/PC e seus Compatíveis
Ettlin – WordStar – Guia do Usuário – Versão 8 bits – CP/M
Forth – Guia do Usuário – Aprendendo e Programando a Linguagem Forth
Hogan – CP/M – Guia do Usuário
Poole – Apple II – Guia do Usuário – II Plus e IIe
Townsend – dBase II – Guia do Usuário

GUIAS DO OPERADOR

Burd e Moreira – MSX – Comandos Básicos
Gifford – Apple II – Comandos Básicos
Ingrahan – CP/M – Comandos Básicos
Wilson – IBM PC – Comandos Básicos

HARDWARE/SOFTWARE

Botelho – Basic Prático – Conceitos Fundamentais e Avançados
Fox/Fox – Iniciação ao Basic
Gottfried – Programação com Basic (SC)
Hehl – Fortran IV
Newcomer – Cobol Estruturado (SC)
Orilia – Processamento de Dados nas Empresas (SC)
Osborne – Introdução aos Microcomputadores – vol. 0
Peckham – Manual de Basic para o Apple II
Poole – Programas Práticos em Basic
Poole – Programas Usuais em Basic – Apple II
Siragusa – Basic Estruturado
Tremblay – Ciência dos Computadores
Verzello – Processamento de Dados
Hardware – volume I
Software – volume II

0-07-450203-4