APPLE II

GUIA DO USUÁRIO

APPLE II Plus e APPLE IIe

SEGUNDA EDIÇÃO

LON POOLE
MARTIN McNIFF
STEVEN COOK

McGraw Hill

APPLE II

APPLE II
Guia do Usuário

2ª Edição, revista e ampliada

LON POOLE
MARTIN McNIFF
STEVEN COOK

Tradução
Paulo Borelli
Engenheiro

Revisão Técnica e Adaptação
ROBERTO COSI CARDOSO JORGE
Engenheiro Eletrônico – Unitron Eletrônica Ltda.

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 • Johannesburg • Kuala Lumpur • London • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Do original:

Apple II, User's Guide

Editor:
Ricardo Reinprecht

Coordenadora de Revisão
Daisy Pereira Daniel

Supervisor de Produção:
Edson Sant'Anna

Capa

Criação: M. Assumpção
R. Reinprecht
Arte: Cyro Giordano

CIP-Brasil. Catalogação-na-Publicação
Câmara Brasileira do Livro, SP

Poole, Lon.
P862a Apple II : guia do usuário / Lon Poole, Martin McNiff, Steve Cook; tradução
2. ed. Paulo Borelli; revisão técnica e adaptação Roberto Cosi Cardoso Jorge. — 2. ed. —
São Paulo: McGraw-Hill do Brasil, 1985.

Bibliografia.

1. Apple II (Computador) I. McNiff, Martin, 1954- II. Cook, Steven, 1960- III. Título.

85-1426

17. CDD-651.8
18. -001.64

Índices para catálogo sistemático:

1. Apple II : Computadores eletrônicos : Processamento de dados
651.8 (17.) 001.64 (18.)

Não Fraudarás!

Seja Como For, é **Errado**

Pessoas que jamais entrariam em uma loja para furtar um produto de software, não hesitariam em fazer várias cópias do mesmo software. O resultado é o mesmo. A atitude é igualmente errada.

Muitas pessoas não percebem o dispendioso impacto causado ao criador do software e à comunidade de clientes. Em uma transação envolvendo software, existe entre o autor e o cliente uma relação de confiança mútua. O cliente confia em que o autor criou um produto que proporcionará o resultado desejado, funcionará de acordo com as especificações e está apropriadamente documentado e com suporte. O autor confia em que o cliente usará apenas as cópias pelas quais pagou uma licença, mesmo que seja relativamente fácil fazer cópias adicionais não autorizadas. A duplicação e uso não autorizado de software, além de imoral, viola nossa **Legislação de Direitos Autorais** (Lei nº 5.988, de 14 de dezembro de 1973) e constitui crime contra a propriedade intelectual (Código Penal, Artigo 184), além de privar injustamente os criadores de software do benefício que têm direito a receber por seu trabalho.

Quem Comete o Crime?

O **softfurto** é executado por indivíduos que fazem cópias ilegais para seu próprio uso, ou para um amigo; a **Pirataria de Software** ocorre quando organizações escolhem conscientemente encorajar, ou inconscientemente permitir que funcionários realizem e usem cópias ilegais de software.

Ambas as práticas violam a Lei Brasileira de Direitos Autorais, e expõem os indivíduos e empresas envolvidas às penalidades previstas em lei.

Cuidado! Alguém Pode Estar Expondo sua Organização

Embora acreditemos que a maioria das pessoas não encoraja a **pirataria de software** e o **softfurto**, sugerimos que verifique, por meio de uma **auditoria interna contínua**, a fim de assegurar-se de que ninguém os esteja praticando, evitando assim as ações previstas pela lei e, conseqüentemente, futuros embaraços para sua empresa.

Agindo dessa forma você estará colaborando para uma melhor conscientização do problema da violação da Legislação de Direitos Autorais no Brasil.

Sumário

Agradecimentos, V
Introdução, XI

1. APRESENTANDO O APPLE II, 1

Teclado e TV, 2. Dentro do Apple II, 3. Memória, 3. Gravador Cassette, 4. Drive de disco, 4. Programas, 5. Controladores de dispositivos externos, 6. Controle de jogos, 9. Impressora, 9. Tábua Gráfica, 11.

2. COMO O APPLE II OPERA, 12

Ligando na tomada, 12
O que você vê no televisor, 13. O caractere de entrada, 14.

O Teclado, 15
O Gravador Cassette, 18
Usando o Disk II, 20
O Sistema Operacional em disco, 20. Preparando novos disquettes, 25.

Carregando e Rodando um Programa, 26
Usando a Versão Correta do BASIC, 26. Carregando um Programa do Cassette, 27. Carregando um Programa do Disco, 28. Iniciando a Execução de um Programa, 28. Conectando TV Colorida, 28.

Componentes Misturados, 30
Copiando com Erros, 30
Mensagens de Erro, 30. Corrigindo Erros de Digitação, 31. Reset Acidental, 31.

3. PROGRAMANDO EM BASIC, 34

Iniciando com BASIC, 34

Modo Imediato e Programado, 35.

Imprimindo caracteres, 35. Imprimindo Cálculos, 36. Mensagens de Erro, 38. Espaços Extras em Branco, 38. Comandos, Linhas e Programas, 38. Modo Programado, 40. Guardando Programas em Cassette, 45.

Trocando de BASIC, 46

Técnicas Avançadas de Edição, 47

Eliminando Linhas de Programa, 48. Acrescentando Linhas de Programa, 48. Alterando Linhas de Programa, 49. Reexecutando em Modo Imediato, 53.

Linguagem de Programação, 54

Elementos de BASIC, 54

Repassando a Numeração de Linhas, 55. Espaços em Branco, 56. Dados, 56. Variáveis, 60. Arranjos, 63. Expressões, 65.

Comando do BASIC, 72

Comentários, 73. Declaração de Atribuição, 73. Declarando o Tamanho de Matrizes e Séries de Caracteres, 77. Comandos de Salto, 77. Loops, 80. Declarações de Sub-rotina, 84. Execução Condicional, 88. Declarações de Entrada e Saída, 89. Execução de Parada e Continuação de Programas, 94.

Funções, 96

Funções Numéricas, 97. Funções Série, 98. Funções de Sistema, 100. Funções Definidas pelo Usuário, 100. Funções Aninhadas, 101.

4. PROGRAMAÇÃO BASIC AVANÇADA, 102

Acesso Direto e Controle, 102

Memória e Endereçamento, 102

Usando Dispositivos Periféricos, 104

Saída de Programa e Entrada de Dados, 105

Mais Sobre a Declaração Print, 105. Funções de Formatação de Impressão, 114. Controle de Cursor e Efeitos Especiais de Vídeo, 117. Janelas de Texto, 119. A Função CHR$: Programando Caracteres em ASCII, 120. Programando Entrada de Dados, 121. Formas de Entrada de Dados, 134. Formatação de Saída, 139. Programando Impressoras, 146.

Guardando Dados em Cassette, 149

Otimização de Programas, 150

Programas Rápidos, 150. Programas Compactos, 151.

Depuração, 151

Restrições no Modo Imediato e Programado, 153

5. O DISK II, 155

Sobre Discos, 155. Como os Dados são Guardados nos Discos, 158. Localizando Trilhas e Setores, 160. Proteção Contra Escrita, 161.

O Sistema Operacional em Disco, 162

Versões do DOS, 162. Inicializando Discos, 162. Arquivos em Disco, 162. Diretório do Disquette, 162. Lista de Trilhas/Setores, 162. Panes em Discos (Crash), 163.

Carregamento do Sistema Operacional DISK II, 164
Como carregar o sistema, 164.

Iniciando Comandos do Disco, 165
CATALOG, 166. LOAD, 167. A Versão em Disco do Comando RUN, 168. Especificando o Número do Drive, 168. Especificação do Slot, 168. Especificação de Volume, 169.

Mais Comandos do DISK II, 170
INIT, 170. SAVE, 172. DELETE, 172. LOCK, 173. UNLOCK, 173. RENAME, 173. VERIFY, 174.

Usando Comandos DOS em Programas, 174

Usando Arquivos em Disco, 175
Usando Arquivos Seqüenciais, 175. Como Inserir Dados num Arquivo Seqüencial, 183. O Comando POSITION, 184. Usando Arquivos de Acesso Aleatório, 185. Um Exemplo Prático de Acesso Aleatório, 186. O Parâmetro Byte, 189.

Outros Comandos DOS, 189
EXEC, 189. MAXFILES, 192. Usando Instrumentos de Depuração do DOS, 193.

Linguagem de Máquina Arquivos de Disco (Imagem Binária), 194
BSAVE, 194. BLOAD, 195. BRUN, 195.

6. GRÁFICOS E SOM, 196

Gráficos de Baixa Resolução, 196
Montando a Página Gráfica, 197. Declarações de Programação de Gráficos, 198.

Gráficos de Alta Resolução, 202
Que Página Deve Ser Usada?, 202. Acessando o Display Gráfico, 203. Alternativas para o HGR e o HGR2, 204. Cores em Alta Resolução, 206. Desenhando Pontos e Linhas, 207.

Usando Formas em Alta Resolução, 209
Definindo Formas, 209. Montando a Tabela de Formas, 210. Entrando a Tabela de Formas, 217. Comandos de Desenhar as Formas, 220.

O Som do Apple II, 223
Operando o Alto-Falante, 224

7. MONITOR DE LINGUAGEM DE MÁQUINA, 229

Acessando o Monitor, 229. Deixando o Monitor, 230.

Funções do Monitor, 231.
Examinando a Memória, 231. Examinando os Registradores de Microprocessador, 234. Alterando Memória, 234. Alterando os Registradores do Microprocessador, 236. Guardando e Lendo Memória com os Periféricos do Apple II, 237. Movendo e Comparando Blocos de Memória, 240. O Comando GO, 244. Usando a Impressora, 245. Os Comandos de Teclado, 245. Indicando Modos de Display, 245. Aritmética Binária em Oito Bits, usando o Monitor, 246. Comandos do Monitor Definidos pelo Usuário, 246.

O Mini-Assembler, 247
Acessando o Mini-Assembler, 247. Comandos de Monitor no Mini-Assembler, 248. Deixando o Mini-Assembler, 248. Formatos de Instruções, 248. Usando o Mini-Assembler, 250. Lista-

gem de Desassembler, 251. Testando e Depurando Programas, 252. Integração do Programa com o BASIC, 257.

8. COMPÊNDIO DE DECLARAÇÕES DO BASIC, 258

Modos Imediato e Programado, 258. Versões do BASIC, 259. Convenções de Nomenclatura e Formato, 159.

Declarações (listadas em ordem alfabética), 260
Funções (listadas em ordem alfabética), 308

APÊNDICES

A — Funções Numéricas Derivadas, 319
B — Comandos de Edição, 323
C — Mensagens de Erro, 323
Mensagens de Erro do Integer BASIC, 323. Mensagens de Erro do Applesoft, 324. Mensagens de Erro do DOS, 325.

D — Sub-rotinas Intrínsecas, 328
E — Posições Úteis de PEEK e POKE, 335
F — Palavras Reservadas do BASIC, 341
Integer BASIC, 341. Applesoft, 341. DOS, 342.

G — Uso da Memória, 344.
Organização Geral da Memória, 344. Os Interpretadores da Linguagem BASIC, 344. Requisitos de Memória do DOS, 345. Uso de Memória do Interger BASIC, 346. Uso de Memória no Applesoft, 347.

H — Formato do DISK II, 349
Lista de Trilha/Setor, 349. O Diretório, 350.

I — Códigos de Caracteres ASCII e Códigos de Palavras Reservadas do Applesoft, 353.
J — Tabela de Conversão, 356
K — Bibliografia, 363
L — Formulários para Desenho na Tela, 365
M — Características específicas do Apple IIe

ÍNDICE ANALÍTICO 384

Introdução

Este livro é seu guia para o computador Apple II. Ele descreve o Apple II e cobre os periféricos mais comuns e acessórios, incluindo drives de disco e impressoras.

Assumimos que você tenha acesso a um sistema com o Apple II completamente instalado de acordo com as instruções do manual do proprietário que vem junto com o equipamento. Ele não explica como instalar o sistema, mas como usá-lo após a instalação.

O que é um Apple II? Como se faz para que ele funcione? Os dois primeiros capítulos respondem a esta questão. Você provavelmente já deve saber que o APPLE II é feito a partir de muitas partes colocadas juntas com fios e cabos. O primeiro capítulo diz o que é cada peça e para que serve. O segundo capítulo diz como operar cada parte do equipamento. Com esse conhecimento você estará pronto para utilizar qualquer um dos pacotes de software prontos-para-uso, extensamente disponíveis para processamento de palavra, análise financeira, contabilidade doméstica, instrução programada por computador e entretenimento.

Os quatro capítulos seguintes informam como executar seus próprios programas em BASIC no Apple II. O terceiro capítulo inicia com um arranjo tutorial de ambas as versões de BASIC disponíveis para o Apple II, Integer BASIC e Applesoft. O quarto capítulo continua o BASIC com abordagem de tópicos mais avançados de programação e outras propriedades do BASIC. Dois dos tópicos avançados, o drive de disco e os gráficos em vídeo são bastante importantes e a eles foram destinados capítulos especiais. O quinto capítulo explica como usar o drive de disco para guardar arquivos de dados e programas. O sexto capítulo ensina como programar gráficos usando o vídeo display pelas duas formas gráficas disponíveis no Apple II.

Os programas em BASIC operam no Apple II sob a supervisão de seu Monitor. O sétimo capítulo explica tanto o Monitor standard como o Autostart sob o ponto de vista da programação BASIC. O capítulo também fornece informação de como incorporar rotinas em linguagem assembler dentro do programa em BASIC.

O oitavo capítulo contém uma descrição completa de cada comando e função disponíveis em ambas as versões do BASIC, incluindo comandos de disco. Juntamente com os apêndices, ele funciona como uma referência imediata, uma vez que já se saiba programar em BASIC no Apple II.

1

Apresentando o Apple II

A Figura 1.1 mostra a configuração típica de um sistema de computador com o Apple II. Veja que o sistema é composto de várias partes separadas que compõem o equipamento.

Seu sistema particular pode não ser exatamente o que está mostrado na figura. Muitos componentes vêm de uma grande lista de dispositivos opcionais. Porém existem três elementos que todos os sistemas têm em comum: o próprio Apple II, o teclado conjugado e o televisor. Vamos dar uma olhada em cada um deles e nos dispositivos opcionais mais comuns. Não vamos ensinar como ligar esses componentes no Apple II. Para as instruções sobre instalação completa, veja os manuais de usuário fornecidos com cada componente.

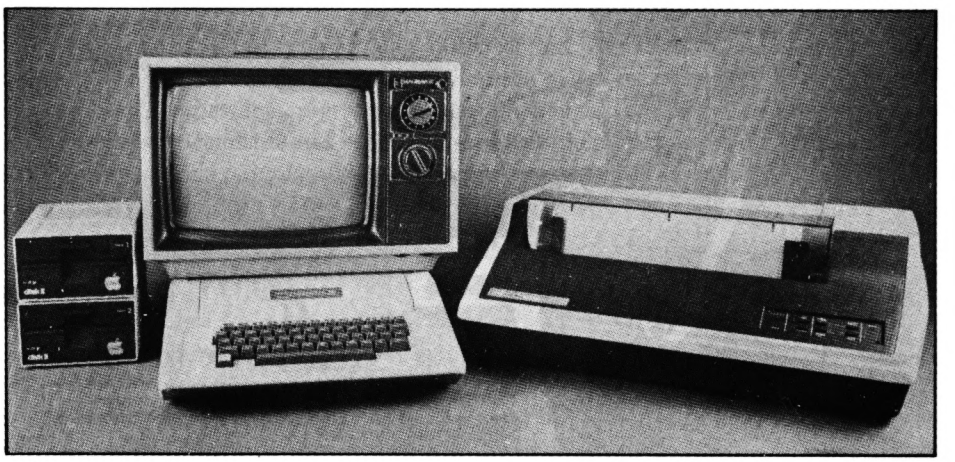


FIGURA 1.1. Um Sistema de Computação Típico do Apple II.

TECLADO E TV

O teclado e a tela de TV tornam a comunicação com o Apple II possível. Um teclado padrão do tipo de máquina de escrever vem com o Apple II. Ele transfere as instruções tecladas para dentro do computador.

A tela de TV pode ser tanto uma televisão colorida comum quanto um monitor em cores. Uma TV em preto e branco também funciona, mas é claro que a informação no vídeo se torna mais pobre do que em cores. A tela não apenas repete aquilo que é teclado de forma a se poder conferir, ela também mostra a reação do Apple II aos comandos teclados.

A tela de TV padrão tem três modos diferentes de operação. Um para caracteres de texto em branco e preto e os outros dois são dedicados a gráficos. No modo texto, a tela é dividida em 24 linhas com 40 caracteres cada uma. Os modos gráficos operam com linhas e pontos e não mais caracteres, e subdividem a tela em elementos menores (os gráficos são discutidos em detalhes no Capítulo 7).

Muitos proprietários de Apple II usam aparelhos de televisão para display, ou porque já possuíam um ou porque têm uma boa desculpa para comprar. Um monitor de TV produz imagens mais precisas do que um TV normal, mas não pode ser usado como televisor. A Figura 1.2 mostra a conexão com um televisor normal.

Como se pode ver, o TV não se conecta direto com o Apple II. Existe um dispositivo de conexão do sistema à antena do televisor. Existe uma chave que estando numa posição deixa o televisor operar de forma comum e em outra faz com que ele sirva de monitor para o Apple II. Um cabo vai do aparelho até um circuito colocado dentro do Apple II. O dispositivo em questão é chamado *modulador de RF* e serve para converter o sinal de vídeo gerado pelo Apple II em algo

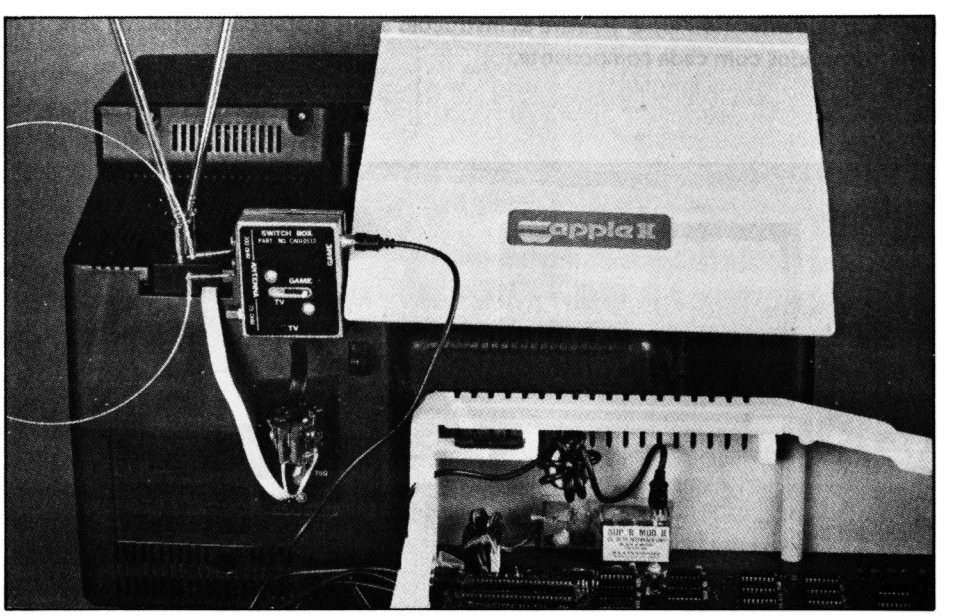


FIGURA 1.2. Conexão a um Televisor Comum.

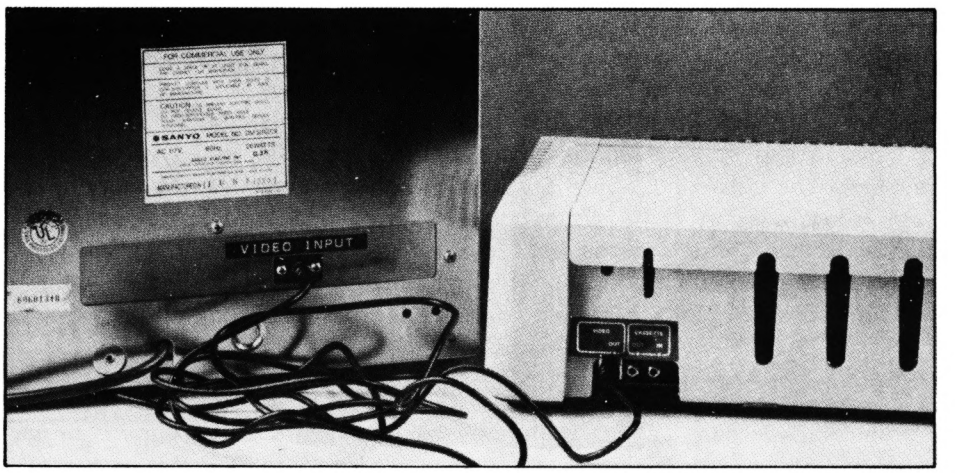


FIGURA 1.3. Conexão a um Monitor de Televisão.

que o televisor entende como imagem. Seu vendedor de Apple II pode lhe vender um modulador de RF e mostrar-lhe como fazer as conexões.

Um monitor de televisão não requer o modulador de RF; ele se conecta diretamente com o Apple II, como é mostrado na Figura 1.3.

DENTRO DO APPLE II

O Apple II guarda em si a parte que controla o resto do sistema — sob sua supervisão, naturalmente! Logo atrás do teclado está o banco principal de memórias, o microprocessador, os pontos de conexão de todos os componentes acessórios, e muito mais. A Figura 1.4 fornece a identificação de todos esses itens.

A parte interna de seu computador pode ser um pouco diferente do que é mostrado na Figura 1.4. O layout básico será o mesmo — a placa maior de circuito impresso com muitos *circuitos integrados*, as pastilhas pretas colocadas em fileiras ordenadas, e ainda algumas pequenas placas montadas verticalmente nos espaços disponíveis na parte traseira da placa maior. O número de circuitos integrados e o número de placas verticais pode variar de um sistema para outro.

MEMÓRIA

A memória do computador é medida em unidade chamada *byte.* Um byte de memória pode guardar um caractere ou uma quantidade semelhante de dados. Dependendo do número de circuitos integrados, seu Apple II pode ter de 4 096 a 65 536 bytes de memória. Isso é normalmente chamado de 4K a 64K, onde K representa 1 024 bytes. A quantidade de memória indica o que o computador pode fazer, como veremos mais tarde.

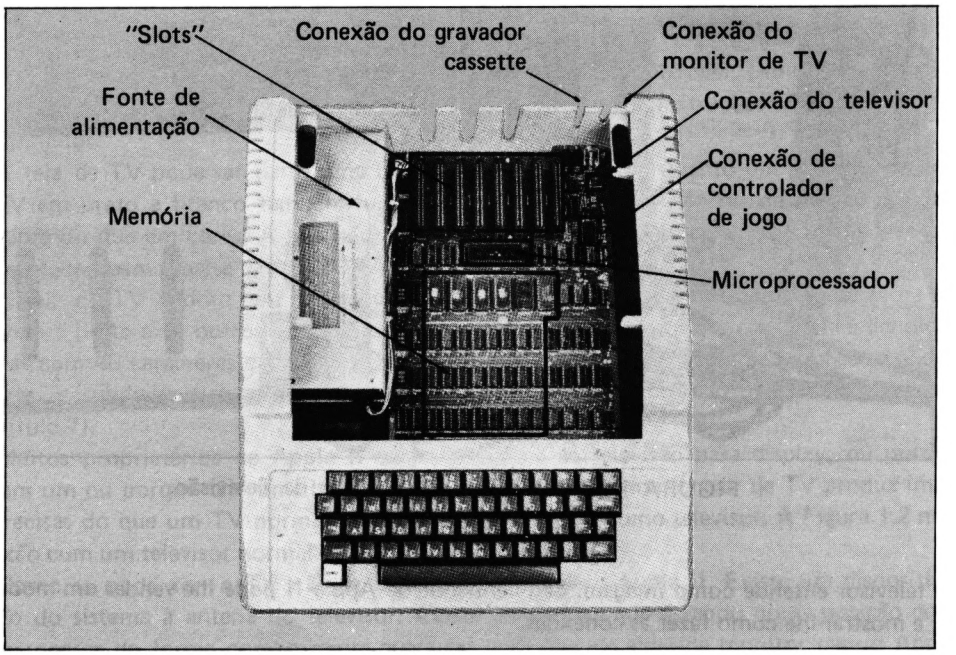


FIGURA 1.4. Dentro do Apple II.

O Apple II tem dois tipos de memória. Uma é a memória que só permite ser lida (*Read-Only Memory* ou simplesmente ROM); seu conteúdo nunca muda, mesmo se a tomada for desligada. A ROM contém os programas que dão ao Apple II identidade única e permitem que ele responda aos comandos que são digitados no teclado. O outro tipo de memória, de gravação e leitura (chamada de *Randon-Access Memory* ou simplesmente RAM) contém dados que podem ser alterados. Os programas em RAM determinam que tarefa específica seu computador está executando.

A memória de leitura e gravação mantém os dados enquanto a tomada estiver ligada. Quando se desliga o Apple II a informação contida na RAM é destruída.

GRAVADOR CASSETTE

Felizmente pode-se usar um gravador cassette convencional para guardar programas da RAM e lê-los depois, de forma que se tenha uma biblioteca de programas em fita cassette. A Figura 1.4 mostra uma instalação com gravador de fita cassette típica.

DRIVE DE DISCO

O drive de disco é muito melhor que o gravador para guardar dados e programas. Ele é mais confiável, tem maior capacidade de armazenamento e opera mais rapidamente. Além disso, o drive de

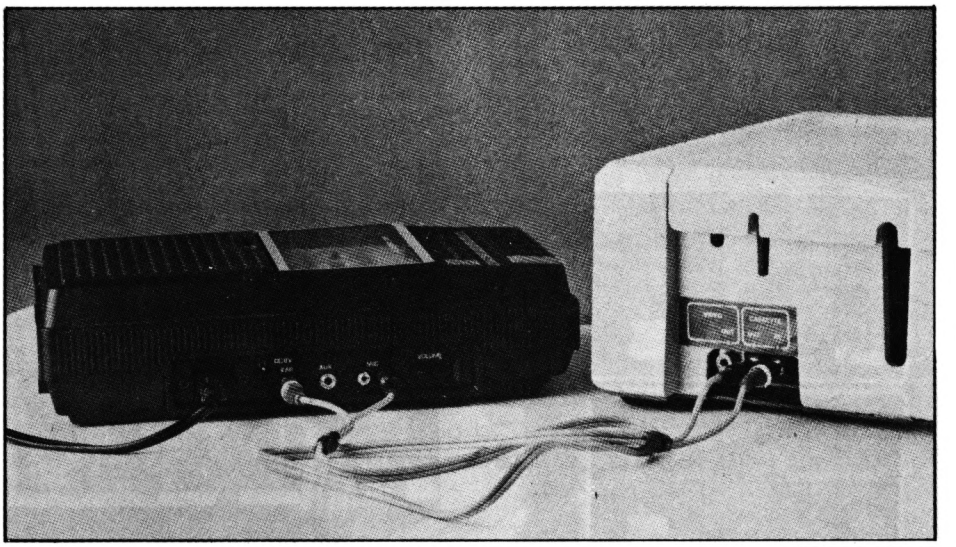


FIGURA 1.5. Conexão a um Gravador Cassette.

disco guarda fácil e rapidamente dados tais como nomes e endereços para uma relação de mala direta ou correspondência para um processador de palavra. Os drives de disco existem em vários tamanhos e formas; diferentes modelos permitem diferentes capacidades de armazenamento. A Figura 1.6 mostra duas unidades do Disk II (produto da Apple Computer Inc.).

PROGRAMAS

Deixemos um pouco de lado a apresentação do equipamento e vamos dar uma olhada nos diferentes tipos de programas que podem ser usados com o sistema. Não estamos falando dos diferentes tipos de coisas que se pode fazer com o Apple II, mas sobre as diferentes classes de programas que podem coexistir para que o Apple II execute as tarefas. Os programas que fazem coisas como jogos, escrever cartas etc., são chamados programas *aplicativos*. Eles sempre são colocados em memória de leitura e escrita (RAM), e não em ROM. Você os transfere para a RAM a partir do gravador ou do disco. Assim, quando você quer que seu Apple II seja um processador de palavra, deve usar um disco que tenha o programa aplicativo de processamento de palavra e transfira o programa para a RAM. O Capítulo 2 explica como fazer isso.

Muito freqüentemente, os programadores escrevem aplicativos numa linguagem de programação que lhes é mais fácil, porém a linguagem é muito avançada para que o Apple II entenda sem alguma ajuda. Um programa especial chamado *interpretador* faz exatamente o que o nome diz. Ele converte o programa aplicativo da linguagem em que ele foi escrito para a linguagem que o computador consegue entender. O Apple II tem um conjunto de diferentes interpretadores, que podem residir tanto em RAM quanto em ROM.

O interpretador utiliza, por sua vez, um outro programa que coordena os componentes do sistema. Este programa, chamado de *sistema operacional*, executa as funções fundamentais de

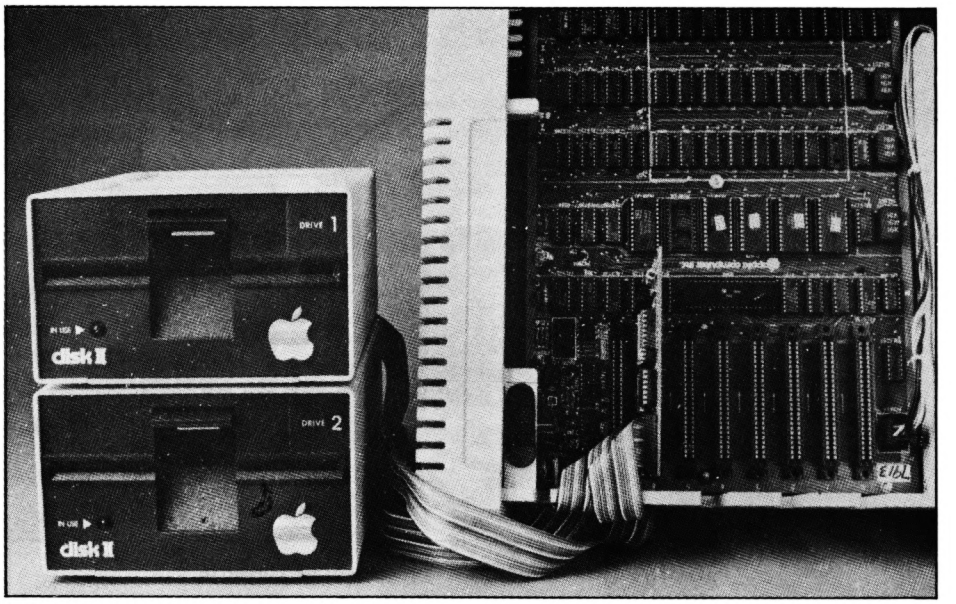


FIGURA 1.6. Ligação do Disk II.

operação do sistema, como transferir programas de cassette ou disco para a memória e mostrar na tela os dados teclados. O sistema operacional do Apple II é chamado de Monitor e sempre reside em ROM.

CONTROLADORES DE DISPOSITIVOS EXTERNOS

Veja as pequenas placas verticais de circuitos impressos colocadas na parte traseira da placa maior do Apple II, que são colocadas nos conectores vazios (slots). Os slots existem para acomodar os circuitos especiais. Tais circuitos, chamados *controladores* ou *cartões*, contêm eletrônica adicional para permitir ao Apple II operar dispositivos periféricos, tais como drives de disco.

A Apple Computer Inc. produz alguns cartões que são colocados nos slots: alguns deles são colocados em slots específicos, mas a maior parte pode ser colocada em qualquer slot. Na maior parte, os cartões necessitam de marcas, identificações ou outros sinais porque é difícil para os iniciantes distinguir um do outro. Cada cartão é marcado com seu nome, mas o nome nem sempre é visível quando o cartão está conectado. Você pode identificar o cartão pelo lugar onde ele está instalado e pelo que está conectado nele. Como mostrado na Figura 1.6, o cartão de controle do Disk II costumeiramente está no slot 6; um ou dois drives podem ser conectados a ele. Um segundo controlador do Disk II pode ocupar o slot 5 para um terceiro e quarto drive de disco, um terceiro controlador pode ocupar o slot 4 para os drives 5 e 6 etc.

Vamos ver alguns dos outros cartões que você pode ter no Apple II.

Cada um dos três cartões mostrados na Figura 1.7 permite operação com linguagem de programação no Apple II. Existem duas versões da linguagem BASIC disponíveis: BASIC integral e

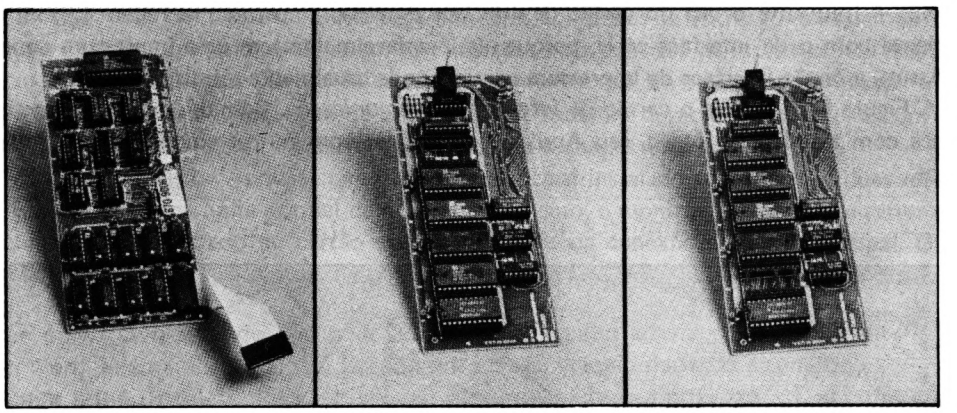


FIGURA 1.7. Esquerda para Direita: Cartões Language System, Applesoft Firmware e Integer BASIC.

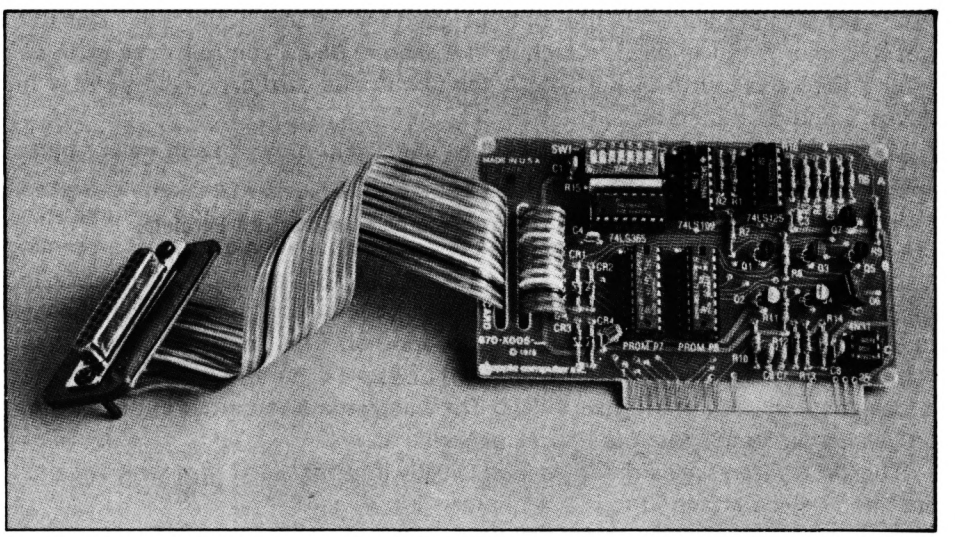


FIGURA 1.8. Cartão de Interface Serial.

Applesoft. Instalando o cartão apropriado no slot 0 pode-se passar de uma linguagem para outra. O *Language System* e o *Applesoft Firmware* operam em qualquer tipo de Apple II, mas o *BASIC integral* é específico para o Apple II Plus. O cartão Language System torna disponíveis outras linguagens, como Pascal, da mesma forma.

O cartão de *interface serial* (Figura 1.8) usualmente ocupa o slot 1. O dispositivo mais comum ligado a ele é a impressora.

Não é freqüente o uso do cartão de *interface paralela* (mostrado na Figura 1.9) ser usado juntamente com o de interface série, porque ele invariavelmente tem uma impressora conectada nele. Assim, o cartão interface de impressora paralela pode usualmente residir no slot 1.

A Figura 1.10 mostra o cartão de *interface de comunicação.* Com um modem conectando a interface com a linha telefônica, seu Apple II pode falar com outros computadores em lugares remotos.

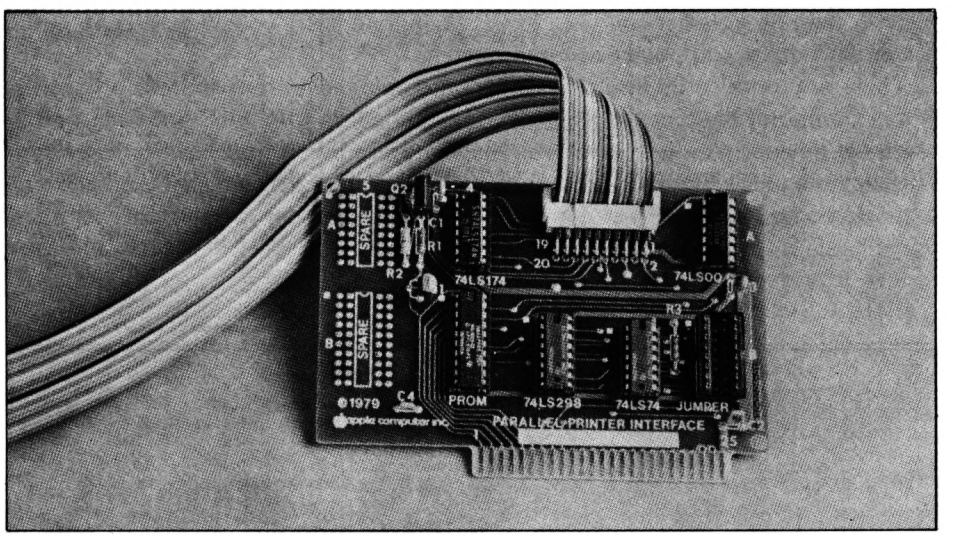


FIGURA 1.9. Cartão de Interface para Impressora Paralela.

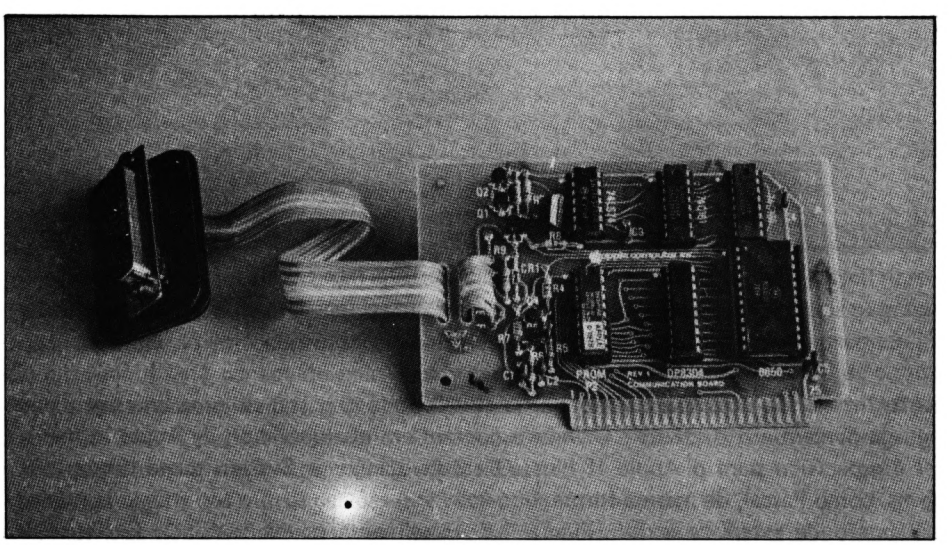


FIGURA 1.10. Cartão Interface de Comunicação.

Existem mais cartões disponíveis para inúmeras aplicações de outras fontes que não a Apple Computer Inc. Por exemplo, pode-se conectar cartões para controle de luzes e outros dispositivos elétricos. Outro cartão permite que se escrevam programas que sintetizam eletronicamente o som de instrumentos musicais. Existe ainda um cartão que funciona como extensor; ele usa um slot do Apple II e possui vários slots para cartões adicionais num chassis separado.

Alguns sistemas de Apple II têm um cartão especial instalado para permitir duas vezes mais caracteres do que uma linha normal de display. Nesse caso, o monitor de TV (um televisor normal não funciona) é conectado no cartão especial ao invés da conexão na placa principal. O cartão especial se conecta num dos slots da parte traseira da placa principal, normalmente slot 3 ou 4. A Figura 1.11 mostra um cartão típico.

CONTROLE DE JOGOS

Vamos concluir nossa *tournée* pelo interior do Apple II com uma vista sobre os dispositivos de controle de jogos que se podem ligar, como mostrado na Figura 1.12. Os controles de jogos são usados com mais freqüência em programas dedicados a lazer, mas podem ser aplicados, da mesma forma, em outras atividades. Estes dispositivos não são obrigatórios.

IMPRESSORA

Vamos observar agora a impressora do sistema (Figura 1.13).

A impressora se conecta ao sistema tanto pela interface serial quanto pela paralela (dependendo do tipo de impressora usada), geralmente pelo slot 1. Existem impressoras de todos os tama-

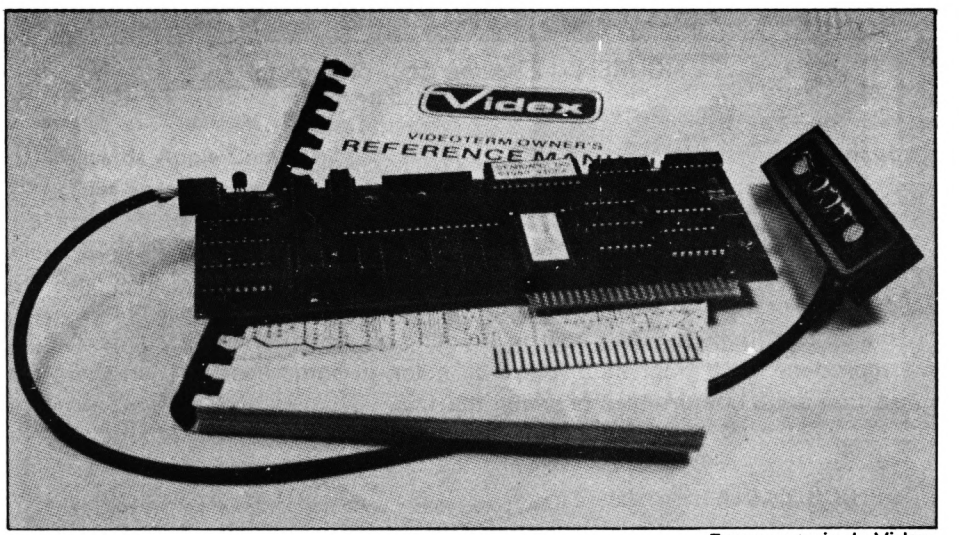


Foto: cortesia da Videx.

FIGURA 1.11. Dispositivo para Formatação Especial da Tela.

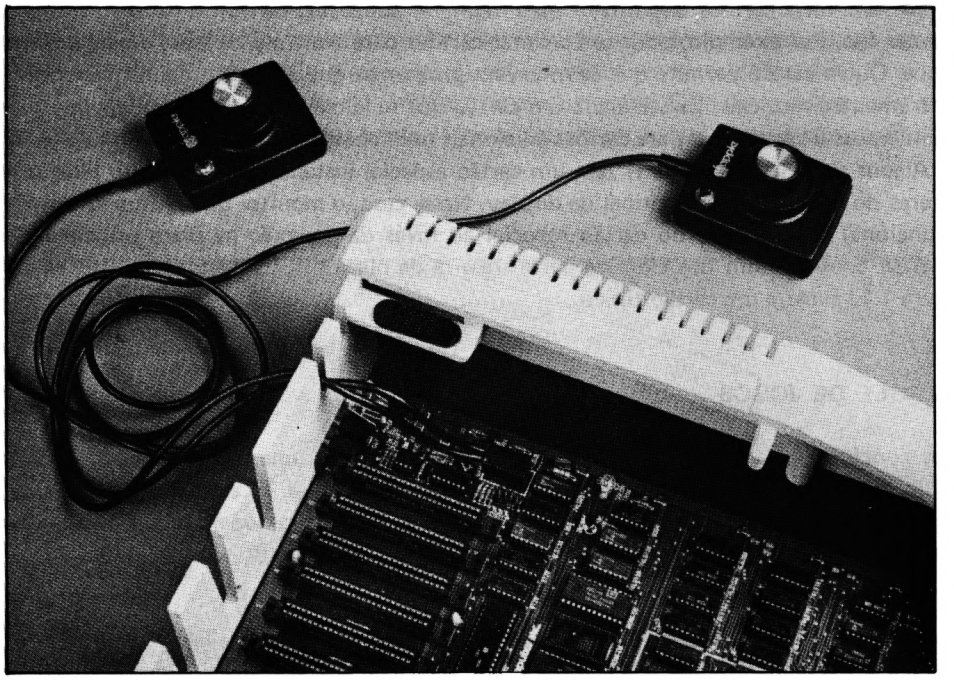


FIGURA 1.12. Dispositivos para Controle de Jogos.

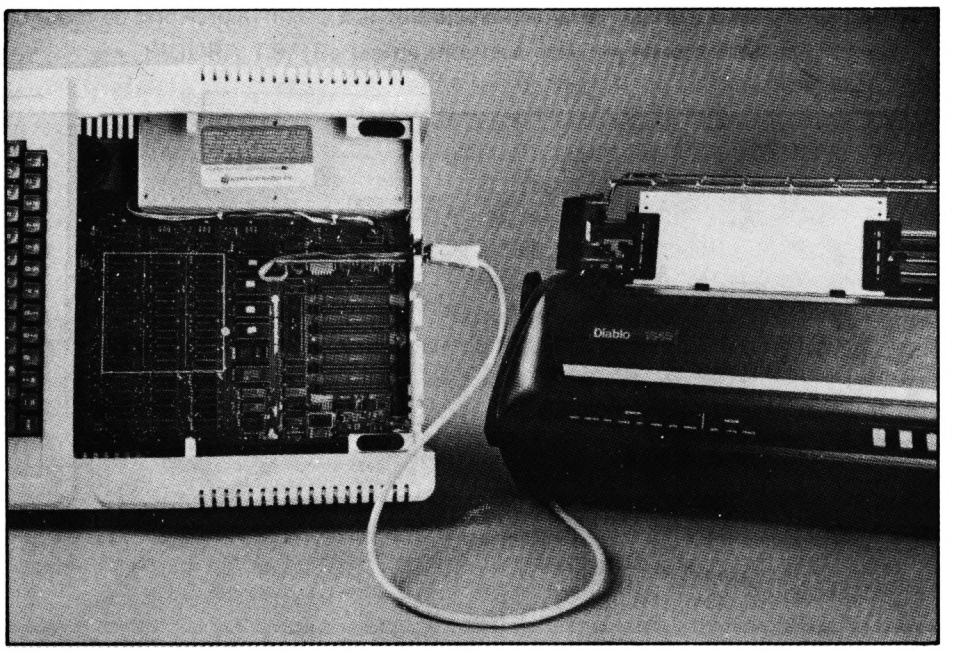


FIGURA 1.13. Conexão da Impressora.

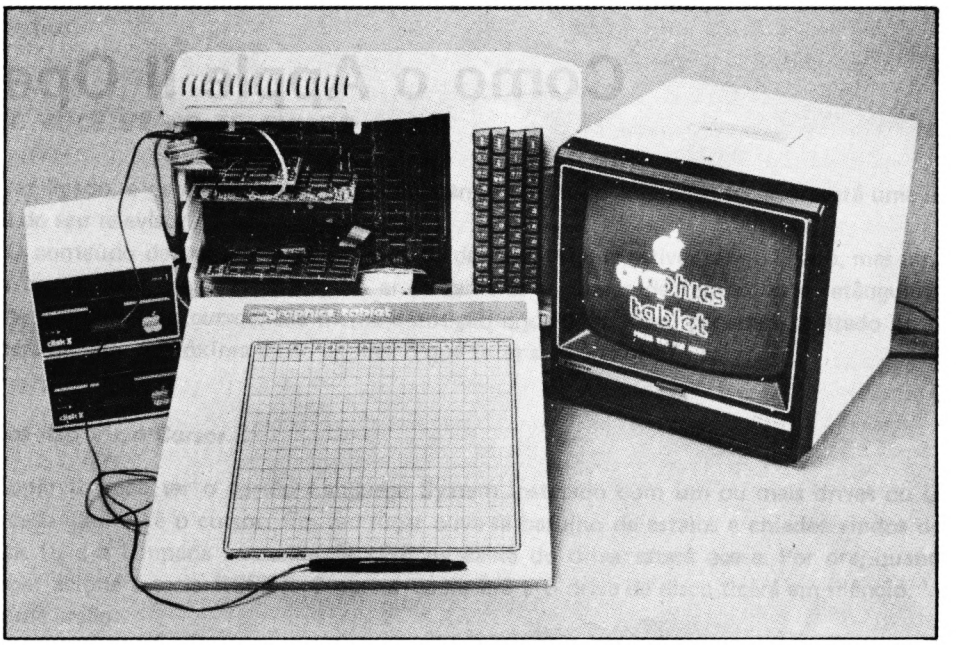


FIGURA 1.14. Conexão da Tábua Gráfica.

nhos, preços e características. Algumas têm qualidade de impressão idêntica à produzida por máquina de escrever. Outras podem produzir gráficos (até coloridos); as demais têm características de compromisso entre uma e outra.

TÁBUA GRÁFICA

A tábua gráfica da Apple Computer Inc., mostrada na Figura 1.14, é um dispositivo portátil que explora a capacidade gráfica do Apple II de forma especial. Com sua caneta, pode criar figuras, desenhos, mapas, gráficos e outros desenhos livres diretamente na tela, em cores. Sob sua direção, ele desenha linhas retas, formatos sólidos e pontos. Seu desenho pode ocupar a tela inteira ou apenas parte dela. Pode-se mover o desenho ao largo da tela, aumentá-lo, reduzi-lo e separar suas cores uma a uma. Em qualquer momento durante esse processo, pode-se guardar o desenho obtido em disco para chamá-lo depois. Pode-se ainda medir distâncias com este dispositivo.

2

Como o Apple II Opera

Qualquer sistema de computador intimida um pouco quando nos sentamos à sua frente, mesmo que esteja inteiramente instalado, como o seu Apple II deve estar antes de prosseguirmos. Este capítulo vai tornar mais confortável seu relacionamento com o Apple II, explicando como usá-lo, mas não como montá-lo. Os manuais de usuário que vêm com cada um dos equipamentos de fábrica trazem informação completa para ajudá-lo na instalação. Se for necessária mais assistência para certificar-se que tudo está correto, verifique com outra pessoa que também tenha um Apple II como o seu, ou com o distribuidor que lhe forneceu o equipamento.

LIGANDO NA TOMADA

Verifique que todos os componentes do sistema estejam ligados juntos, corretamente, e ligue o televisor. Deixe o volume no mínimo. Coloque a chave do dispositivo conectado à antena de TV na posição GAME ou COMPUTER. Sintonize o canal especificado nas instruções que acompanham o modulador de RF (normalmente canal 33). Se você não sabe qual canal usar, pergunte a alguém que tenha um sistema igual ou contate o seu fornecedor.

Procure a chave liga-desliga atrás do Apple II, perto de onde sai o cabo de força. Ligue a chave. Você deverá ouvir um sinal vindo de dentro do equipamento. O sinal indica que o computador está pronto. A lâmpada POWER do teclado deverá acender, a menos que esteja queimada.

Se o sinal não for ouvido, desligue a chave e ligue novamente. Se nada for ouvido, desligue o computador. A lâmpada acendeu? Se não, desligue o Apple II e ligue uma lâmpada ou rádio para ver se há força na tomada. Leia as instruções novamente e confira as conexões para certificar-se de que tudo está ligado corretamente. Se ainda assim o computador não emitir o sinal, DESLIGUE-O! Desconecte da tomada e peça auxílio a alguém que tenha mais experiência (seu distribuidor, por

exemplo). Você poderá causar muitos danos tentando mexer no computador ou desligando conexões de fios.

O QUE VOCÊ VÊ NO TELEVISOR

Uma vez ligado, e depois de soar o sinal indicando que tudo está correto, aparecerá uma imagem na tela do seu televisor.

O conteúdo da imagem depende do tipo de Apple II que estiver sendo ligado, mas uma coisa é bem distinguível, por ficar piscando em intervalos regulares de tempo; este retângulo branco piscante é o chamado *cursor*. Ele marca a posição onde o próximo caractere digitado no teclado aparecerá na tela. A próxima seção explica o que fazer se o cursor não aparecer.

Se Você não Vir o Cursor

Seu Apple II pode ter o cartão Language System instalado com um ou mais drives do Disk II. Neste caso não se vê o cursor. Em seu lugar ouve-se barulho de estalos e chiados vindos do drive do Disk II, e a lâmpada vermelha IN USE na caixa do drive estará acesa. Por ora, quando isso acontecer acione a tecla RESET. O cursor aparecerá e o drive de disco ficará em silêncio. Vá para a próxima seção.

Alguns sistemas como o Apple II são configurados para permitir mais de 40 caracteres em cada linha. Nestes casos, para ver o cursor deve-se digitar uma seqüência de comandos especiais como segue:

1. Acione a tecla CTRL e mantenha-a abaixada, acione a tecla B e em seguida solte ambas.
2. Para o próximo comando, você deve saber em qual slot o cartão especial (aquele ao qual o monitor de TV está ligado) está conectado, provavelmente slots 3 ou 4. Pode-se abrir o Apple II e olhar dentro — os slots são numerados de 0 a 7 da esquerda para a direita. A Figura 1.11 explica o que procurar. Peça ajuda a alguém (seu fornecedor) se estiver em dúvida.
3. Se o seu monitor está conectado no cartão do slot 3, digite PR#3 e acione a tecla RETURN. Se está no slot 4, use PR#4; para slot 2 digite PR#2 etc.
4. O cursor agora fica visível, embora ele possa não piscar (o que está correto). Continue como descrito abaixo.

TABELA 2.1. Caracteres de Entrada.

Caractere de Entrada	Linguagem
*	Monitor
>	Integer BASIC
]	Applesoft

O CARACTERE DE ENTRADA

À esquerda do cursor existe um outro caractere. Ele é o primeiro da linha. Pode ser um asterisco (*), um sinal de maior (>) ou um fecha-colchetes (]). O Apple II é um computador de multi-linguagens e o caractere de entrada indica qual a linguagem que está esperando instrução no momento. A Tabela 2.1 mostra os três caracteres de entrada e suas linguagens correspondentes.

*é o Monitor

Em muitas versões do Apple II o primeiro caractere que se vê ao se ligar o aparelho é o *. Se este não for o seu caso pule esta seção e a seguinte.

O asterisco é a entrada para o *Monitor de Linguagem Assembler* do Apple II. Em geral, apenas os usuários mais avançados têm a necessidade de se comunicar diretamente com o Monitor. Se você deseja experimentar, pode fazê-lo. Embora não haja condição de fazer nada por ora com o Monitor, você deverá desligar e religar a chave para anular a entrada do Monitor e cancelar os efeitos da experiência.

Quando o * aparece, deve-se dizer ao Apple II que pule para o BASIC.

CTRL-B em BASIC

Existe um comando no Monitor que instrui ao Apple II para pular para o BASIC. Para permitir este comando, acione e deixe acionada a tecla CTRL, enquanto aciona e solta a tecla B. Solte então a tecla CTRL e acione RETURN. Um caractere de entrada diferente aparecerá então sobrepujando o * . Dependendo do tipo de Apple II usado e de suas opções, aparece o caractere > ou].

A palavra BASIC vem de Beginner's All-purpose Symbolic Instruction Code, uma linguagem de computação muito usada, desenvolvida pelo Instituto Dartmouth. O Apple II suporta duas versões de BASIC: o Integer BASIC e o Applesoft. Ambas as versões contêm características do BASIC original de Dartmouth.

> é o Integer BASIC

O caractere > indica que o Apple II está pronto para receber instruções em Integer BASIC. As regras do Integer BASIC são diferentes daquelas do Applesoft, mas as duas têm muita coisa em comum. Por ora concentremo-nos nas instruções que são comuns às duas versões, de forma que não precisa haver preocupações com o tipo de BASIC usado.

] é o Applesoft

O caractere] informa que o Apple II está pronto para receber instruções do Applesoft. Não se preocupe por ora com o tipo de BASIC que está usando. Quando isso se tornar importante, abordaremos as diferenças entre ambos.

O TECLADO

O teclado do Apple II (veja Figura 2.1) se parece em muito com o teclado de uma máquina de escrever comum, mas ele tem cinco teclas que não existem nessas máquinas. Duas estão à esquerda, com as inscrições misteriosas ESC e CTRL. As outras três estão à direita, com RESET, ← e →. Existem mais duas teclas do lado direito desconhecidas: RETURN e REPT.

Vá em frente e digite no teclado. Tudo que for colocado no computador pode ser cancelado ligando e desligando a chave.

Durante a digitação, pode-se perceber que todas as letras digitadas são maiúsculas, como se vê na tela, a menos que se acione a tecla SHIFT antes de se teclar uma letra. O Apple II standard só sabe mostrar na tela as letras maiúsculas. Pode-se conectar um dispositivo que permite o uso de maiúsculas e minúsculas. Além disso, existem programas que sabem como mostrar na tela tanto maiúsculas quanto minúsculas.

A Tecla RESET

RESET é uma tecla muito especial no teclado do Apple II. Quando o RESET é acionado, pára tudo. Não importa o que o computador esteja fazendo no momento que se aciona o RESET, o controle do Apple II retorna ao teclado. Dependendo do Apple II, o RESET faz voltar ao Monitor, Integer BASIC ou Applesoft com o aparecimento do caractere de apresentação respectivo.

Às vezes o RESET causa um bocado de problemas, especialmente se o Disk II está sendo usado no momento em que se aciona a tecla. Assim, deve-se prestar muita atenção para não acionar a tecla RESET acidentalmente. Tome cuidado especial ao acionar a tecla RETURN, pois o dedo pode facilmente subir um pouco mais acionando o RESET sem querer. Algumas versões do Apple II dão uma proteção especial contra esta ação, exigindo o acionamento da tecla CTRL-RESET ao invés de simplesmente RESET (veja a discussão sobre a tecla CTRL a seguir).

FIGURA 2.1. O Teclado do Apple II.

A Tecla RETURN

À medida que se digita, os caracteres são mostrados na tela. Além disso, o Apple II guarda tudo na memória sem tentar interpretar o que foi teclado até que se acione a tecla RETURN. Esta tecla informa ao computador que a linha de comando que estava sendo digitada já terminou. Quando se aciona RETURN, o Apple II limpa qualquer caractere estranho à direita do cursor. Então faz a análise do comando digitado. Se a seqüência de caractere representar um comando que ele pode executar, ele executa a ação apropriadamente. Por outro lado, se for ouvido o sinal sonoro e aparecer uma das mensagens abaixo (ou eventualmente outra mensagem do mesmo tipo):

```
?SYNTAX ERROR
*** SYNTAX ERR
```

O Apple II está dizendo que não entendeu o comando dado antes da digitação da tecla RETURN. Você deve redigitar a linha (sem o erro detectado pelo Apple II).

A Tecla SHIFT

As letras são sempre maiúsculas no Apple II standard. Assim, a tecla SHIFT não muda as letras de minúsculas para maiúsculas. Ela faz com que uma mesma tecla permita dois caracteres diferentes. Um deles é obtido pelo acionamento apenas da tecla, sem a tecla SHIFT. O caractere que se obtém pelo acionamento da tecla em conjunto com SHIFT é mostrado na parte de cima da tecla.

Usa-se a notação SHIFT- para descrever o duplo acionamento envolvendo a tecla SHIFT. Por exemplo, SHIFT-3 produz o caractere #.

Algumas teclas de letras não imprimem letras quando se usa a tecla SHIFT. A tecla N, por exemplo, imprime ^, e um @ é apresentado quando se aciona SHIFT-P. A tecla G tem a palavra BELL escrita no alto, o SHIFT-G imprime a letra G, e não fornece o sinal sonoro.

A Tecla CTRL

CTRL é a contração de control. A tecla CTRL sempre é usada junto com outra tecla da mesma forma que o SHIFT. Deixa-se a tecla CTRL acionada enquanto se aperta e solta a outra tecla. Este livro se refere ao acionamento da tecla CTRL com outra tecla pela especificação CTRL-. Por exemplo, CTRL-B significa que as teclas CTRL e B devem ser acionadas em conjunto.

A tecla CTRL, como o SHIFT, permite a algumas teclas que tenham funções especiais. A tecla G é a única que tem informada no alto a sua função obtida com a tecla CTRL: BELL. A função do BELL fica óbvia quando se aciona CTRL-G e o Apple II responde com o *bip*. Ativou-se o sinal sonoro do computador.

Existem outras combinações de CTRL que causam várias reações ao Apple II. Ainda há pouco mencionamos o comando CTRL-B no Monitor que faz a passagem para o BASIC.

Outra combinação interessante é o CTRL-X. Ela informa ao Apple II que deve esquecer tudo o que foi digitado na linha de comando presente para que se recomece. Tente digitar qualquer coisa e em seguida acionar o CTRL-X. O cursor volta para o início da linha. Os caracteres antigos ainda permanecem na tela mas não têm mais significado para o computador. Ele entende como se eles nunca tivessem sido digitados.

A Tecla ESC

ESC vem de escape, que é um termo usado desde quando as *teletypes* eram os terminais mais usados. Embora o nome sugira, acionar a tecla ESC não provoca interrupções. A tecla ESC tem muitos usos, alguns dos quais aparecem neste capítulo e o resumo aparece no próximo.

Diferentemente de SHIFT e CTRL, a tecla ESC nunca é usada em conjunto com outra tecla. Ela é acionada e solta em separado, antes que outra seja acionada. Esta operação da tecla ESC com outra é chamada *seqüência de escape*.

Uma seqüência simples de escape é ESC-@. Aciona-se esta seqüência de escape, primeiro acionando a tecla ESC, a seguir acionando e mantendo acionada a tecla SHIFT, enquanto se aciona a tecla P (SHIFT-P = @). O resultado? Tudo o que está na tela é zerado, e o cursor se move para o canto esquerdo no alto da tela. (Em "computês" o canto esquerdo no alto da tela é chamado de *home*.)

As Teclas ← e →

As duas teclas de flechas são chamadas de flecha-à-esquerda e flecha-à-direita.

As teclas ← e → são muito úteis porque com elas pode-se corrigir erros de digitação que eventualmente tenham sido cometidos, bem como alterar dados que tenham sido colocados. A tecla ←, como a tecla backspace em máquinas de escrever, cada vez que é acionada, o caractere sob o cursor é apagado da memória do Apple II e o cursor volta uma posição para a esquerda. Tente agora. Digite uma palavra (tente PRINT). Acione ← várias vezes e veja como o cursor volta percorrendo a palavra digitada de trás para diante. Veja que os caracteres não são apagados da tela. Pode ter certeza que eles já foram esquecidos pelo Apple II. Tente voltar o cursor todo à esquerda da tela. Quando se chega ao limite e se aciona ← de novo, o cursor volta uma linha e o novo caractere de entrada aparece.

Como você deve estar suspeitando, a tecla → move o cursor para a direita ao longo da linha digitada. À medida que ele vai para a direita, todo o caractere pelo qual o cursor passa é copiado pela memória do Apple II exatamente como seria se ele fosse recém-digitado. Para ver a tecla → em ação, digite uma nova palavra e volte o cursor alguns caracteres usando ←. Agora acione → algumas poucas vezes. Cada vez que essa tecla é acionada, o caractere pelo qual o cursor passa é levado de volta para a memória do Apple II, como se ele estivesse sendo redigitado.

A Tecla REPT

REPT é abreviação de repetição. Se a tecla REPT fica acionada e outra tecla é tocada ao mesmo tempo, esta outra tecla será repetida até que ambas se soltem. Isso é bastante útil quando a outra tecla é de flecha e é necessário limpar (←) ou recopiar (→) muitos caracteres.

Para que a função de repetição funcione corretamente, deve-se primeiro acionar a tecla cuja função se deseja repetir, acionando-se então a REPT. Soltando a REPT a repetição cessa.

As Outras Teclas

As outras teclas do Apple II, sem dúvida são familiares. São elas as letras do alfabeto, os números de zero a nove, e um conjunto de símbolos padronizados.

Muitos datilógrafos não distinguem entre o número zero e a letra *O* ou entre a letra *I* e o número um. O Apple II não pode aceitar esta ambigüidade. Deve-se tomar bastante cuidado para que se digite um numeral quando se pede um numeral. Para ajudá-lo a lembrar, o teclado do Apple II mostra o zero com uma barra cruzada, e os zeros mostrados na tela também têm esta barra.

O GRAVADOR CASSETTE

Se o seu sistema possui um gravador cassette, ele pode receber programas através de fitas cassettes. Algumas delas vêm com o Apple II; pode-se comprar outras; e pode-se também montar um arquivo com fitas próprias. (Veja explicações no Capítulo 4.)

Manuseando Cassettes

Deve-se ter cuidado com o manuseio das fitas cassettes. Elas se danificam muito facilmente. Não importa o quanto sua pele seja limpa; o óleo natural estraga a fita. Certifique-se de que colocou as fitas em suas caixas corretas após usá-las. Não as guarde em lugares quentes, sob a luz direta do sol, ou sob campos magnéticos (tais como os encontrados perto de motores elétricos).

Rotule Todos os Cassettes

Deve-se rotular os cassettes com informação sobre os programas que eles contêm. Isso evita o aborrecimento de pesquisar todo um arquivo de cassettes, um a um, para se obter um programa desejado.

Protegendo os Cassettes Contra Escrita

Cada cassette tem dois entalhes na parte traseira. Os gravadores geralmente têm um dispositivo que sente a presença destes entalhes e não permitem nova gravação se eles não estão presentes. As fitas novas têm um pedaço de plástico cobrindo o entalhe, de forma que se pode gravar e ler à vontade. Os programas importantes gravados nos cassettes podem ser protegidos removendo a cobertura de plástico.

Para determinar qual dos dois entalhes é o correto, segure o cassette de forma que a parte exposta da fita fique exposta para a frente e o lado que se deseja proteger seja o de cima. Remova a parte plástica do lado direito para evitar gravações sobre a superfície superior (veja a Figura 2.2.). Cobrindo o entalhe com fita adesiva pode-se gravar dados numa fita que foi protegida.

Ajustando o Volume de Leitura

Deve-se determinar o volume de leitura do gravador cassette no nível próprio para o Apple II. Se o volume for muito alto ou baixo, a informação da fita será distorcida e o computador não será capaz de entendê-la.

FIGURA 2.2. Dispositivos de Proteção em Fitas Cassette.

A única forma de determinar o nível de volume correto para seu gravador é o método de tentativa e erro. Aqui está um procedimento geral. Primeiro, tente carregar um programa com o volume bem baixo. Se não funcionar, aumente um pouco e tente de novo. Permaneça ajustando para cima até que o programa seja carregado satisfatoriamente.

Pode-se usar uma das fitas que vêm com o Apple II. Se o caractere de entrada após o cursor for], ache o cassette chamado "Color Demosoft". Se o caractere de entrada for >, ache o cassette chamado "Color Graphics".

Coloque a fita no gravador. Certifique-se de que o nome do programa está em cima. Para cada posição do controle de volume:

1. Rebobine a fita completamente.
2. No teclado do Apple II, digite a palavra LOAD.
3. Aperte o botão PLAY do gravador para iniciar a leitura.
4. Acione a tecla RETURN.

Após acionar RETURN, o cursor desaparece. Depois de 15 ou 20 segundos, pode-se analisar o ocorrido.

Se apareceu a mensagem ?SYNTAX ERROR ou ***SYNTAX ERROR, não ajuste o volume, volte ao passo 1 e tente novamente. Se acontecer de novo, tente limpar a cabeça de leitura ou usar uma fita ou gravador diferente.

Se nada acontece, ou se é mostrado ERR ou ERRERR, acione RESET, coloque o volume um pouco mais alto, e tente novamente.

Se for ouvido o *bip*, ou nenhuma mensagem aparecer, as coisas vão indo bem. O Apple II achou o início do programa da fita e está carregando. Depois de mais uns 15 segundos (dependendo do tamanho do programa da fita), vai ser ouvido outro *bip* e o caractere de entrada, bem como o cursor, reaparecerão na tela. Deve-se então desligar o gravador. Anote o volume utilizado para

que não seja necessário repetir este processo novamente quando estiver usando o gravador longe do Apple II.

Quando for visto o caractere de entrada do BASIC e o cursor, o programa foi carregado com sucesso.

USANDO O DISK II

Se o sistema possui um ou mais drives do Disk II conectado ao Apple II, pode-se operar programas de disquettes ao invés de cassettes. O "System Master Diskette" fornecido junto com o Apple II tem a maioria dos programas que a Apple Computer Inc. fornece com os cassettes, mais alguns programas extras especialmente projetados para o Disk II.

Manuseando Diskettes

Deve-se ter extremo cuidado no manuseio de disquettes. Eles são muito mais delicados do que os cassettes. Nunca dobre um disquette. Nunca toque a superfície do disquette (a parte que aparece sob a abertura oval), e nunca force a entrada de um disquette no drive do Disk II. Sempre recoloque o disquette em sua capa protetora quando ele estiver fora do drive, e proteja-o contra o calor, a luz direta do sol e campos magnéticos (como os encontrados perto de motores elétricos). Tenha especial cuidado com o "System Master Diskette".

Como Colocar os Disquettes Dentro do Drive do Disk II

A forma apropriada de inserir o disquette no drive do Disk II é mostrada na Figura 2.3. Segure o disquette entre os dedos polegar e indicador de forma que o polegar cubra a etiqueta. Abra a porta do drive do Disk II e suavemente coloque o disquette totalmente dentro do drive. Não deve haver resistência. Uma vez que o disquette esteja dentro do drive, feche a porta delicadamente. Ela deve fechar-se facilmente. Se houver alguma resistência, solte a porta e empurre o disquette inteiramente para dentro do drive, tentando novamente. Se a porta for forçada, destruirá o disquette. Às vezes ajuda esperar que o disquette seja centrado, esperando que o drive comece a girar e então fechando a porta.

O SISTEMA OPERACIONAL EM DISCO

Antes que se possa usar qualquer programa do drive de disco, deve ser colocado na memória um programa especial chamado *Disk Operating System.* Este programa, também chamado *DOS* controla as atividades realizadas pelo drive de disco. O processo de colocar uma cópia do DOS na memória é chamado de *booting.* Em "computês" diz-se fazer o "boot do disco" ou "boot do DOS" ou simplesmente "boot DOS".

Deve-se executar o boot do DOS a cada vez que se desliga o equipamento e se liga novamente.

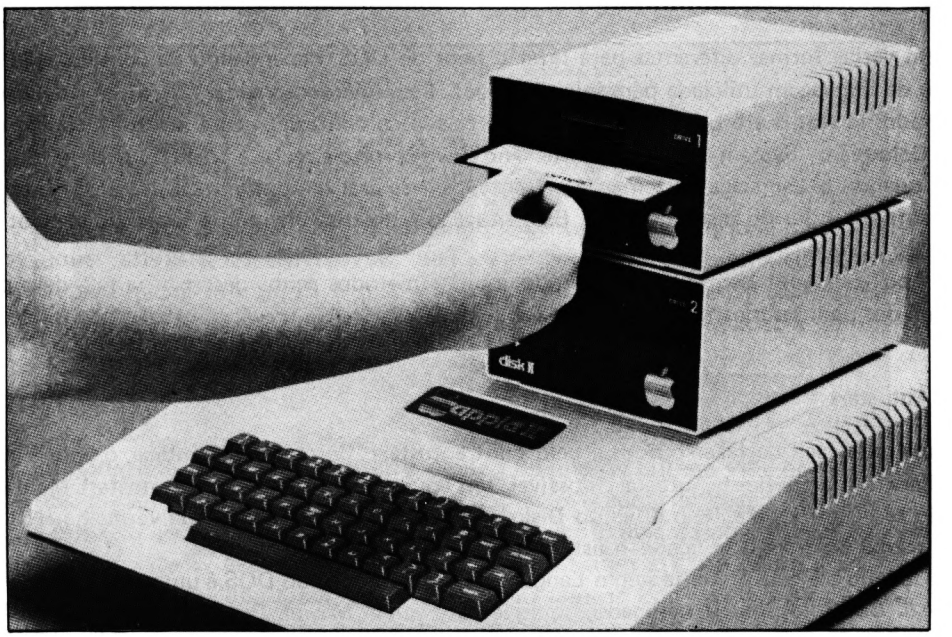


FIGURA 2.3. Inserindo o Disquette no Disk II.

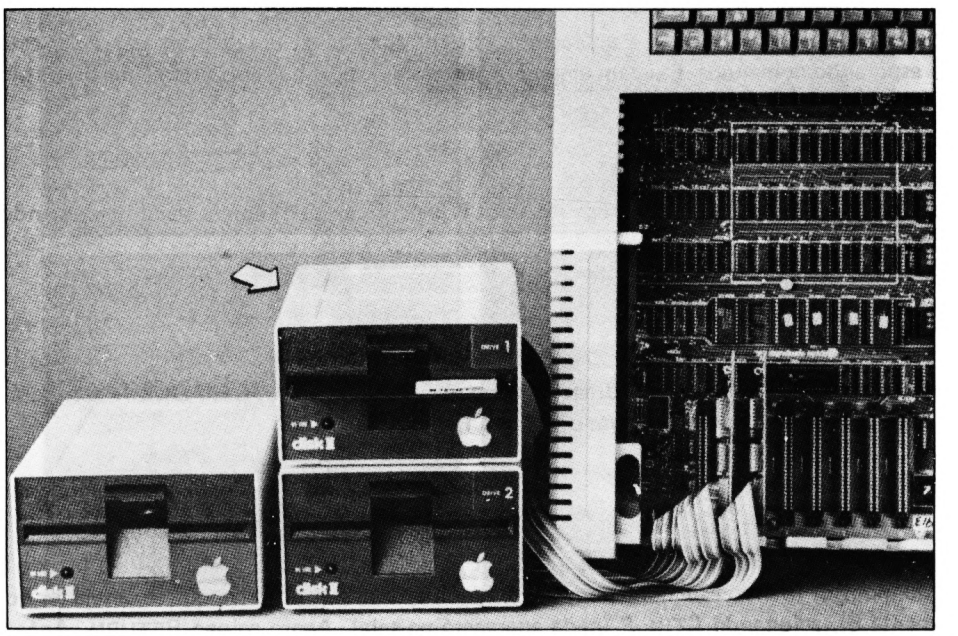


FIGURA 2.4. Drive de Disco Standard para o Boot do DOS.

Fazendo Boot do DOS

Existem várias formas diferentes para fazer o boot do DOS, dependendo da configuração do sistema e da linguagem utilizada para iniciar o boot. Cada método assume que o drive de disco está conectado no slot 6 e o drive 1 está ligado no cartão de controle. Isso é mostrado na Figura 2.4.

Coloque o "System Master Diskette" no drive correto e feche a porta do mesmo. Em alguns casos, quando o Language System está presente, deve-se usar um disquette especial, o "Integer and Applesoft II" antes do disco do DOS. Esta situação é descrita abaixo na seção sobre boot com o Language System.

Após um boot bem-sucedido usando um dos métodos acima descritos, a tela deve parecer com um dos padrões mostrados na Figura 2.5.

Boot Autostart

A forma mais fácil de fazer o boot é pelo boot autostart. Como o nome indica, o boot é automático. Para tornar o Boot Autostart possível, seu equipamento deve ter o Autostart Monitor. Se, quando se ligar o Apple II, o drive do Disk II ranger e estalar, e a lâmpada vermelha IN USE da parte frontal do drive ficar ligada, o sistema possui o Autostart Monitor.

Quando ele está presente sem o Language System, o boot do DOS é um procedimento de um só passo. Com o Apple II desligado, coloque o "System Master Diskette" no drive do Disk II apropriado e feche a porta. Ligue o aparelho. Após alguns segundos, o disco pára e a tela passa a mostrar um quadro semelhante a um dos mostrados na Figura 2.5.

Boot a Partir do Monitor

Quando o caractere de entrada * aparece na tela, o Monitor de Linguagem Assembler está acessado e esperando comandos. Existem algumas formas de fazer o boot do disco a partir do Monitor.

Salto do Monitor para o Boot

Pode-se fazer o boot do DOS do drive 1 no slot 6 (veja Figura 2.4) com o seguinte comando dado no Monitor:

```
*C600G
```

Lembre-se de acionar a tecla RETURN. A lâmpada vermelha IN USE do Disk II acenderá, e em alguns segundos a tela se parecerá com o visto na Figura 2.5.

Boot do Monitor com o CTRL-P

O outro comando do Monitor para gerar o boot do DOS é através do CTRL-P. Para fazer o boot a partir do Monitor usando este comando, digite o número do slot do drive que se deseja ler no disco (usualmente 6), acionando CTRL-P (nada aparecerá na tela). Agora acione RETURN. Após alguns segundos a tela se parecerá com algo da Figura 2.5.

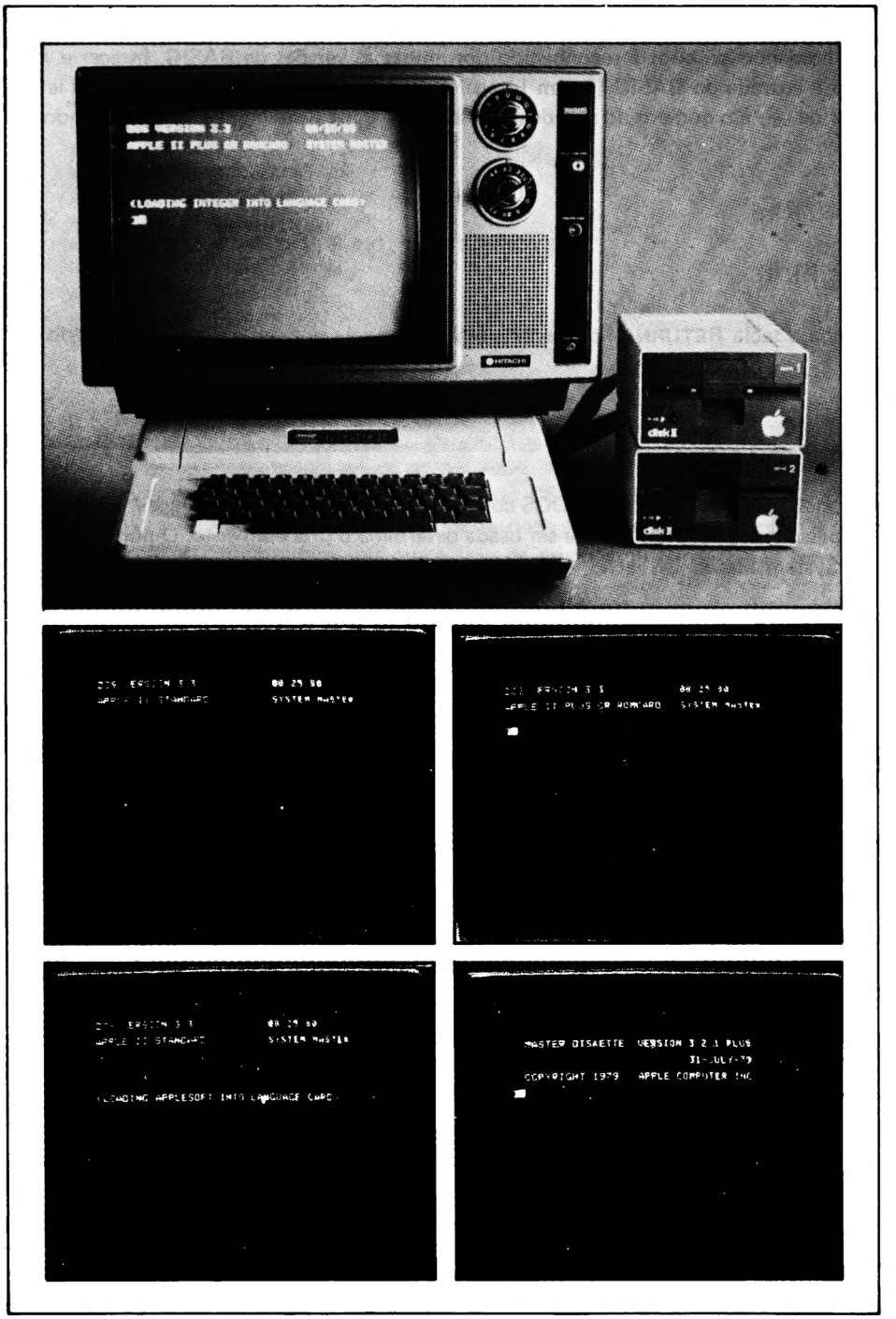


FIGURA 2.5. Boot Satisfatório do System Master Diskette.

Boot a Partir do Integer BASIC ou Applesoft

O mesmo comando de boot é reconhecido por ambas as versões de BASIC, Integer e Applesoft.

Após a entrada do BASIC (> em Integer BASIC ou] em Applesoft) digite as letras IN ou PR, o sinal de #, e o número do slot de onde se fará o boot (usualmente 6). O comando se parece com:

```
IN#6

PR#6
```

Agora acione a tecla RETURN. Após alguns segundos a tela se parecerá com o mostrado na Figura 2.5.

Boot a Partir do Language System

Sob tais condições, fazer o boot do DOS com a presença do System Language é um procedimento com dois passos. A versão do DOS a ser usada determina o procedimento. O número de versão do DOS no "System Master Diskette" está estampado no próprio disquette. É um número como 3.3, 3.2.1 ou 3.2.

Entrando com a versão 3.3 do DOS, não importa se existe ou não o Language System. Use o "System Master Diskette" com um dos procedimentos descritos acima. O DOS é carregado do disquette para a memória. Outro passo é executado automaticamente e faz com que tanto o Integer BASIC quanto o Applesoft fique disponível imediatamente, via Language System. Quando o Disk II cessa, o boot está terminado e a tela fica semelhante a uma daquelas mostradas na Figura 2.5.

Carregar as versões 3.2.1, 3.2 e menores do DOS requer dois disquettes. Primeiro é preciso colocar o disquette chamado "Integer and Applesoft II" no Disk II correto, normalmente aquele conectado no slot 6, e drive 1. Então executa-se uma das funções para boot descritas acima. O drive emite alguns sons e a lâmpada vermelha IN USE na frente da caixa do drive acende. Em poucos segundos, aparece a seguinte mensagem na tela:

```
INSERT BASIC DISK AND PRESS RETURN
```

Abra agora o drive e remova o disquette. Insira o "System Master Diskette" no drive e feche a porta. Acione a tecla RETURN do teclado. Após mais alguns segundos, a tela se parecerá com uma das telas da Figura 2.5. Isso indica que o boot foi executado corretamente.

Como Ver o Catálogo de Disquettes

Se o boot foi executado corretamente, pode-se ver que programas o disquette contém. No teclado do Apple II digite:

```
CATALOG
```

A tela mostrará algo como o que vem a seguir. (Você se lembrou de acionar a tecla RETURN?)

```
DISK VOLUME 254

*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

Como Fazer o Boot de Outros Disquettes

Descrevemos como realizar o boot do DOS a partir do "System Master Diskette". O procedimento é exatamente o mesmo para qualquer disquette onde foi colocado o Disk II. Os disquettes inicializados para outros computadores ou outro tipo de drives normalmente não funcionam no drive do Disk II.

PREPARANDO NOVOS DISQUETTES

De tempos em tempos precisa-se de novos disquettes para os programas que se rodam no Apple II. Antes de se usar um disquette com o Disk II pela primeira vez, deve-se inicializá-lo. Se o programa específico que se está usando inclui rotina especial para inicializar disquettes, use-a preferencialmente. Na sua ausência, pode-se usar o método geral a seguir para preparar disquettes extras.

O processo de inicialização torna o disquette pronto para uso subseqüente. Durante a inicialização, seja qual for o programa BASIC que está sendo usado, ele é salvo na memória, e se torna o programa de *apresentação* do disquette. O programa de apresentação é automaticamente carregado e rodado toda vez que se faz o boot do disquette. Se existe um programa de apresentação em sua mente, OK. Se não, pode usar a Figura 2.6 como modelo. Use o programa exatamente como é

```
NEW
10    REM INICIALIZADO EM data
20    REM MEMÓRIA DO SISTEMA bytes
30    REM VERSÃO DO DOS número
40    PRINT "nome do disco"
50    END
```

FIGURA 2.6. Modelo de Apresentação de um Programa.

mostrado exceto para os termos em letras minúsculas, que devem ser substituídos pelas informações de momento. Em *data*, coloque a data na qual o disquette está sendo inicializado. Em *bytes*, coloque o número de bytes da memória do sistema (32K, 36K etc.). O número é o número da versão do DOS que está presente no disquette mais recente (algo como 3.2, 3.2.1 ou 3.3). O nome do disquette é o nome que se deseja dar para aquele disquette especial. Abaixo vai uma ilustração de um programa de inicialização:

```
NEW

10 REM INICIALIZADO EM 1/1/81
20 REM MEMORIA DO SISTEMA 48K
30 REM VERSAO DO DOS 3.2.1
40 PRINT "MISC./VOL.3"
50 END
```

Assim, para inicializar um disquette, primeiro prepare o programa de apresentação, ponha o disquette no drive e digite o seguinte comando:

```
INIT HELLO
```

Certifique-se de que a porta do drive esteja fechada e acione RETURN. A lâmpada vermelha do drive acenderá, acompanhada dos estalos e ruídos característicos. O processo inteiro de inicialização leva aproximadamente dois minutos, de forma que deve haver paciência. Quando a lâmpada apaga, o processo estará completado.

Agora prepare uma etiqueta para o disquette. Remova-o do drive e aplique a etiqueta.

CARREGANDO E RODANDO UM PROGRAMA

Existem muitos programas já escritos para o Apple II. Alguns vêm em cassettes, outros em disquettes, ou em ambos. Alguns deles são escritos em Integer BASIC e outros em Applesoft. De forma geral, um programa escrito para o Integer BASIC não funciona no Applesoft e vice-versa.

USANDO A VERSÃO CORRETA DO BASIC

Embora muitas versões do Apple II venham com as duas versões da linguagem, nem todas têm essa possibilidade. O Apple II Plus não tem o Integer BASIC a menos que o cartão Language System ou o cartão Integer BASIC estejam presentes. Por outro lado, o Applesoft não é encontrado em muitas versões do Apple II a menos que se leia um interpretador de cassette ou disco.

Em um Apple II standard é fácil obter a entrada no Integer BASIC (>). Acione a tecla RESET e a seguir o CTRL-B (lembrando-se de terminar com RETURN).

É difícil de se obter o caractere de entrada do Applesoft em um Apple II standard, desde que ele reside num disco ou fita. Deve-se instruir o computador para carregar o interpretador do Applesoft na memória a partir do disco ou da fita.

Antes de se usar o drive de disco deve-se fazer o boot do DOS como descrito anteriormente neste capítulo. Uma vez que o DOS está carregado, pode-se carregar o interpretador Applesoft colocando-se o System Master Diskette no Disk II e digitando o seguinte comando:

```
]FP
```

O drive rangerá e estalará por alguns segundos e então o caractere de entrada do Applesoft (]) aparecerá na tela.

Pode-se também obter o interpretador do Applesoft a partir de fita cassette. Use o cassette chamado "Applesoft II" (da Apple Computer Inc.), coloque-o no gravador e rebobine-o todo. Digite o comando:

```
]LOAD
```

Antes de acionar o RETURN, acione a tecla PLAY para iniciar o movimento do gravador. Logo se ouvirá o *bip* do Apple II, o que indica que foi iniciado o carregamento do interpretador Applesoft. O tempo gasto vai de um minuto e meio a dois minutos. Uma vez carregado o programa, será dado um segundo *bip*. Desligue o gravador cassette. O caractere de entrada do Applesoft estará presente na tela.

A partir do Applesoft, pode-se passar para o Integer BASIC digitando o comando:

```
]INT
```

Desejando voltar para o Applesoft, deve-se recarregar o interpretador Applesoft do disco ou do cassette.

CARREGANDO UM PROGRAMA DO CASSETTE

Com a versão apropriada do BASIC carregada (ou o Apple II apto a fazer a seleção automaticamente), estes são os passos para carregar um programa a partir do cassette:

1. Posicione a fita no início do programa. Este será normalmente no início da fita, e nesse caso deve-se rebobinar a fita completamente. Se o programa desejado não está no início da fita, deve-se carregar todos os programas precedentes, em ordem, para posicionar a fita no lugar correto.
2. No teclado do Apple II digite o comando,

   ```
   LOAD
   ```

3. Aperte o botão PLAY no gravador cassette para iniciar.
4. Acione a tecla RETURN.

Após acionar RETURN, o cursor desaparece. Após alguns segundos, o Apple II gera o *bip* para indicar que iniciou a carga do programa. Mais tarde um outro *bip* indica que o processo já terminou. Acione então o botão STOP do gravador cassette para pará-lo.

O programa está carregado. Se nenhum bip for ouvido ou aparecer alguma mensagem de erro durante o processo de leitura, reveja o ajuste de volume de acordo com as recomendações dadas anteriormente, neste capítulo. Se o teclado não responder a tecla RESET deve ser acionada, ou deve-se desligar e ligar o computador. Se for necessário, recarregue o interpretador a partir

do cassette ou disquette. Se ainda aparecerem problemas, sua fita cassette provavelmente está inutilizada e deve ser trocada.

CARREGANDO UM PROGRAMA DO DISCO

Uma vez que foi feito o boot do DOS, pode-se ler programas a partir do disco, com o comando:

LOAD *nome do programa*

Nome do programa se refere ao programa que deve ser carregado. Naturalmente, o programa indicado deve estar presente no disco que vai ser colocado no drive.

INICIANDO A EXECUÇÃO DE UM PROGRAMA

Quando o programa desejado está carregado na memória, use o comando seguinte para iniciar sua execução:

RUN

O programa sai do controle do Apple II, incluindo teclado e vídeo. Para recuperar o controle, pode-se digitar CTRL-C, em muitos programas. Pode ser que seja necessário acionar a tecla RETURN. Se isso não funcionar, leia as instruções de operação do programa que está rodando. Numa emergência pode-se acionar a tecla RESET ou desligar o Apple II, porém em ambos os casos o programa pode ser perdido.

CONECTANDO TV COLORIDA

O Apple II suporta gráficos em cores. Se o programa que está sendo usado ou escrito usar esta característica, deve ser feito o ajuste das cores do televisor ou do monitor de TV para um balanço correto. A Apple Computer Inc. fornece um programa para auxiliar nessa tarefa. Dentro do programa Integer BASIC (>) use o cassette "Color Graphics". Com o Applesoft, (]) use o cassette "Color Demosoft" ou o programa COLOR DEMOSOFT em disquette. Em ambos os casos, carregue e rode o programa de acordo com as instruções precedentes neste capítulo.

A tela se parecerá com a mostrada na Figura 2.7.

O que se vê na Figura 2.7 é chamado *menu*. Um menu permite várias escolhas (quatro, no caso) e pede para que você faça a escolha. Para fixar a cor da TV, selecione o número 1 acionando a tecla 1 e o RETURN. Então a tela mostrará 16 barras verticais, como mostrado na fotografia da Figura 2.8, só que em cores.

Como se vê, as cores são colocadas na parte de baixo da tela. Da esquerda para a direita as cores são:

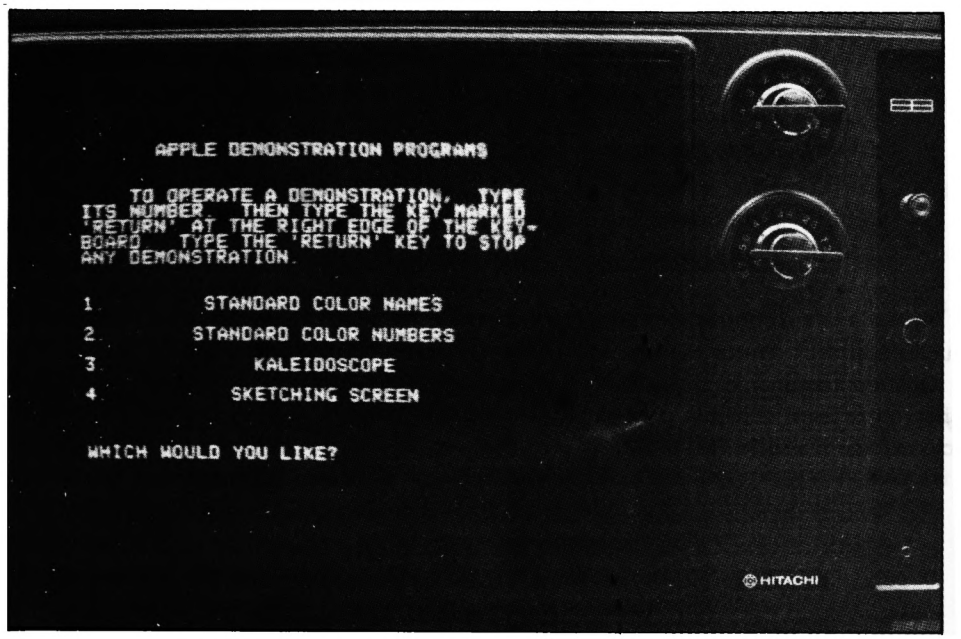


FIGURA 2.7. Menu do Programa COLOR DEMOSOFT.

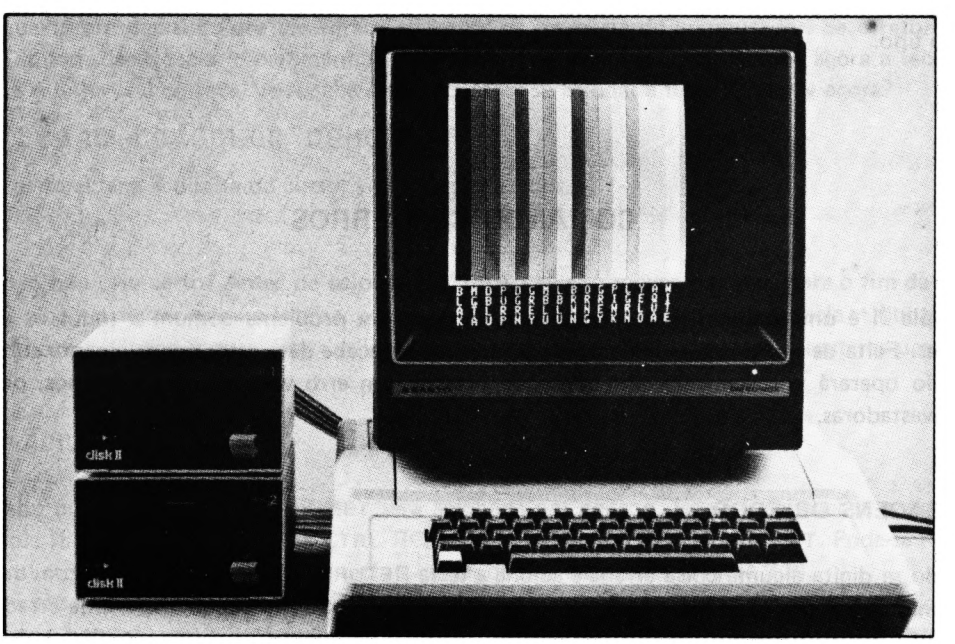


FIGURA 2.8. Ajustando as Cores da TV.

BLAK	Preto	BRWN	Marron
MGTA	Magenta	ORNG	Laranja
DBLU	Azul Escuro	GREY	Cinza
PURP	Púrpura	PINK	Rosa
DGRN	Verde Escuro	LGRN	Verde Claro
GREY	Cinza	YELO	Amarelo
MBLU	Azul Suave	AQUA	Água
LBLU	Azul Claro	WITE	Branco

Ajuste agora o contraste, brilho, cor e controles de tons da televisão, até que as cores tenham resolução aceitável. Preste especial atenção em Púrpura, Rosa, Amarelo e os três tons de Azul.

Quando terminar o ajuste das cores, acione a tecla RETURN e o menu reaparece. Pode-se escolher outro item qualquer para experimentar. Para terminar o programa acione CTRL-C (deve ser acionado com a tecla RETURN).

COMPONENTES MISTURADOS

Isto inclui as instruções especiais de operação para componentes do sistema. Para outros componentes do seu sistema, use o manual de operação específico para aquele componente. Se for uma impressora, por exemplo, então deve ser lido o manual da impressora, porque as instruções variam com o tipo.

COPIANDO COM ERROS

O Apple II é um aparelho maravilhoso, mas ele tem um problema comum a todos os computadores. Falta de imaginação. Toda a instrução que ele recebe deve estar rigorosamente correta ou ele não operará corretamente. As conseqüências de um erro variam entre extremos, podendo ser devastadoras.

MENSAGENS DE ERRO

Quando se digita alguma coisa errada e aciona a tecla RETURN, o Apple II responde com um bip e uma mensagem crítica. Freqüentemente a mensagem dá uma idéia do que houve de errado; mas pode não dar também. O remédio em geral é o mesmo em ambos os casos, repetir a linha. Uma completa lista de erros se encontra no Apêndice C.

CORRIGINDO ERROS DE DIGITAÇÃO

À medida que se digita no teclado do Apple II, pode-se cometer erros. Algumas das teclas descritas anteriormente podem auxiliar a corrigir erros antes que se acione a tecla RETURN para entrar a linha. São elas as teclas ←, →, REPT, CTRL-X e ESC-@:

- A tecla ← volta o cursor de uma posição e zera o caractere que passou. Os caracteres são zerados da linha de comando embora permaneçam na tela.
- A tecla → move o cursor para a frente, copiando as letras por que passa.
- A tecla REPT, usada em conjunto com ← e →, permite ir para a frente e para trás rapidamente.
- O comando CTRL-X cancela a linha que está sendo digitada.
- O comando ESC-@ limpa a tela e coloca o cursor na posição mais à esquerda em cima.

Vamos ver como se pode usar estas facilidades de edição. Suponha que se deseja digitar isso:

```
LOAD COLOR DEMOSOFT
```

No lugar disso, digita-se o seguinte:

```
LOAS COLOR DEMOSOFT
```

Existem duas chances. Ou aciona-se o CTRL-X para cancelar a linha de comando e reiniciar tudo ou se usa a tecla ← para voltar e corrigir o engano. Acione e mantenha a tecla ←. Acione então REPT. O cursor corre de volta para o início da linha. Retire o dedo da tecla REPT, quando o cursor atingir o caractere errado. Lembre-se que à medida que o cursor voltou, todos os caracteres que foram passados por ele foram apagados da memória do computador. Se o retorno foi muito rápido, use a tecla → para alinhar o cursor com a letra errada S. Acione agora a tecla D e pronto! A linha está correta. Você sabe o que acontece se acionar a tecla RETURN agora?

```
LOAD [] COLOR DEMOSOFT
```

Todos os caracteres à direita do cursor vão ser desprezados.

```
LOAD
```

Isso não está certo! Antes de acionar RETURN, deve-se mover o cursor para o fim da linha. Pode-se ainda redigitar a linha inteira, porém isso pode ocasionar outro erro. A tecla → (junto com REPT) move o cursor para o fim da linha, automaticamente reentrando os caracteres passados.

RESET ACIDENTAL

Mais cedo ou mais tarde, a tecla RESET será acionada por engano. Isso é inevitável, a menos que seu Apple II requeira a digitação de CTRL-RESET ao invés de simplesmente RESET. Pode-se reduzir o azar retirando a capa de plástico da tecla e deixando apenas a haste (Figura 2.9).

Pode-se tomar uma precaução a mais e tornar a tecla RESET mais difícil de ser acionada. Pegue uma borracha de 3/8 de polegada de diâmetro interno, 1/2 polegada de diâmetro externo e 1/8 de polegada de espessura. Remova a parte de cima da tecla RESET como mostrado na Figura

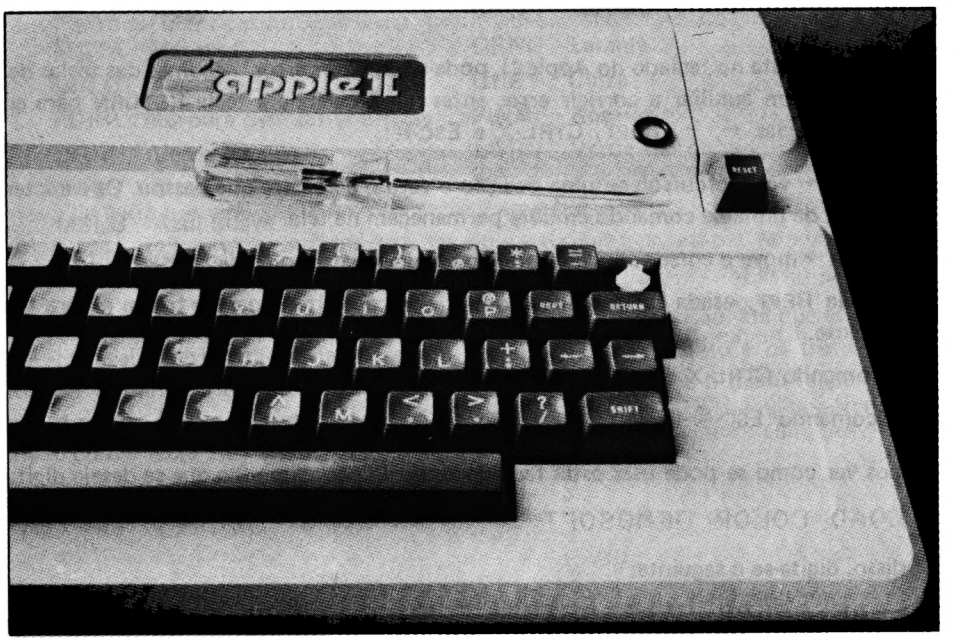


FIGURA 2.9. Proteção contra RESET Acidental: Fase I.

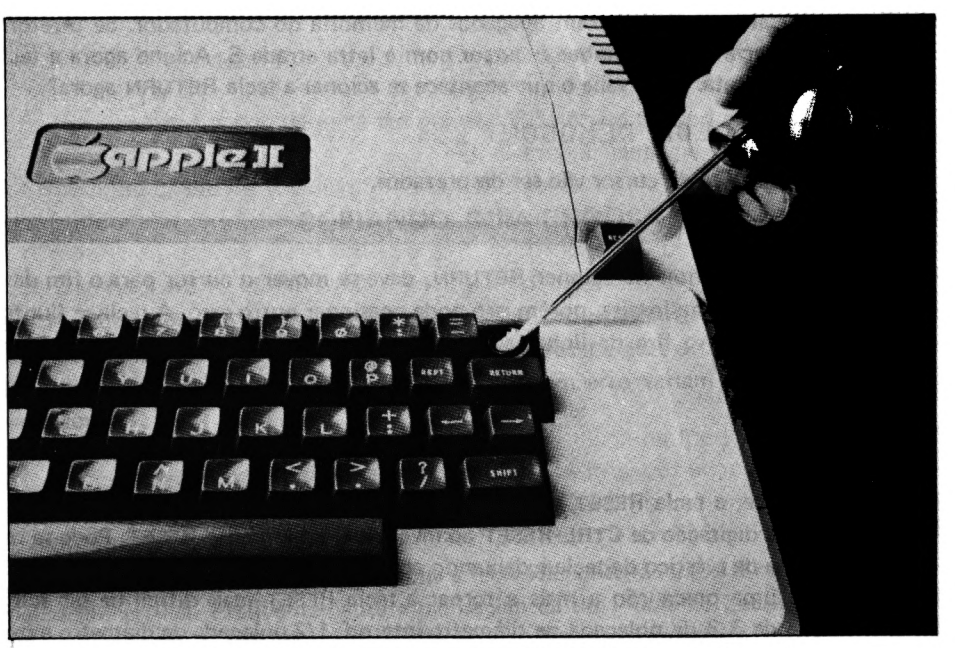


FIGURA 2.10. Proteção contra RESET Acidental: Fase II.

2.9. Coloque a borracha sobre a parte quadrada exposta da tecla como mostrado na Figura 2.10. Reponha a tampa da tecla.

Voltando de um RESET

O que se pode fazer após um RESET acidental depende do que se estava fazendo no instante em que o RESET ocorreu, e do tipo de Apple II que se está usando.

Qualquer programa usado, deve ter instruções específicas sobre o que fazer no caso de acionamento acidental do RESET durante a execução do programa. Certifique-se disso antes de iniciar a execução. Se a tecla RESET for acionada durante a execução do BASIC, pode-se reiniciar o programa desde o início. Este é um pequeno consolo durante algumas fases de programas aplicativos, desde que reiniciar pode dar muita dor de cabeça.

Quando se aciona o RESET, o Apple II pára tudo o que estava fazendo. O controle retorna para o teclado; aparece o caractere de entrada e o cursor piscando na parte de baixo da tela. Se o programa é o BASIC pode-se voltar à execução dando o comando RUN. Aquilo que foi processado antes do RUN provavelmente se perderá.

Se for no Monitor (*), pode-se voltar ao BASIC digitando-se CTRL-C *a menos* que se esteja usando Applesoft baseado em disco ou cassette. Para sair do Monitor para Applesoft em disco, digite o seguinte comando do Monitor:

```
*3D0G
```

Para pular do Monitor para o Applesoft em cassette, use o seguinte comando do Monitor:

```
*E000G
```

PRECAUÇÃO: Certifique-se de que usou o comando correto. Se não estiver certo sobre o que usar, pergunte a alguém mais experiente. Use o comando errado e estará perdido. O programa precisará ser recarregado, assim como o Applesoft e possivelmente até o DOS.

3

Programando em Basic

Este capítulo ensina como iniciar a execução de seus próprios programas em BASIC no Apple II.

O BASIC é uma linguagem de programação. Como outra linguagem do tipo, ela consiste num conjunto de comandos, que são combinados para criar programas. Um programa define a tarefa que se deseja que o computador execute.

Poderíamos ensiná-lo a programar em BASIC, primeiro forçando o aprendizado de todos os comandos, um a um. Porém isso não daria grande resultado, pois os comandos não possuem, individualmente, muito significado. O estudo de comandos individuais de BASIC rapidamente se degenera em estudo de regras arbitrárias que nada dizem ou significam, em termos de programação e prática em construção de programas. Assim, a definição rigorosa de todos os comandos da linguagem BASIC foi deixada para o Capítulo 8. Vá ao Capítulo 8 quando sentir necessidade, porém não o leia sem ter aprendido os conceitos deste capítulo.

INICIANDO COM BASIC

Existem duas versões diferentes de BASIC disponíveis para o Apple II. Alguns modelos possuem o Integer BASIC, outros têm o **Applesoft**, e outros ainda dispõem de ambos. Por ora, não se preocupe sobre que versão está usando, se Integer BASIC ou Applesoft. Mais tarde, quando as diferenças se tornarem significativas, entraremos em mais detalhes.

O Caractere de Entrada do BASIC

O Apple II é uma máquina multilingual. Se se deseja programar em BASIC, deve-se inserir instruções nessa linguagem. Existem instruções sobre como iniciar a operação com o BASIC no Capítulo 2. Vamos fazer uma rápida revisão.

No Apple II com o Monitor de início automático, precisa-se apenas ligar a máquina e acionar o RESET (CTRL-RESET em algumas versões do teclado).

Se o seu Apple II não tem início automático, ligue-o e acione CTRL-B e RETURN.

O caractere de entrada à margem esquerda e o cursor informam em que posição o BASIC se encontra. Se for Integer BASIC o caractere é ">", se Applesoft é "]".

MODO IMEDIATO E PROGRAMADO

Antes de nos preocuparmos em como pular do Integer BASIC para o Applesoft e vice-versa, vamos dar uma olhada em comandos simples de BASIC que podem ser usados em ambas as versões. Alguns dos exemplos deste capítulo usam Integer BASIC (>) enquanto outros usam o Applesoft (]). Pode-se treinar os exemplos em ambas as versões, não importando a sua versão específica, a menos que se especifique o contrário. Os exemplos em que não aparece o caractere de entrada, podem ser usados em ambas as versões.

IMPRIMINDO CARACTERES

Quando se inicia pela primeira vez com a operação do Apple II em BASIC, se está no *modo imediato*, também chamado modo "direto" ou "calculadora". Neste modo, o computador responde imediatamente às instruções que recebe. Tente escrevendo o exemplo:

```
PRINT "PROGRAMANDO EM MODO DIRETO"
```

Não se esqueça de acionar a tecla **RETURN**, após a última aspa. O Apple II imprime a seguinte mensagem:

```
PROGRAMANDO EM MODO DIRETO
```

Se é impressa a mensagem ?SYNTAX ERROR ou ***SYNTAX ERROR, isso está dizendo que o comando não foi entendido. Provavelmente houve erro de digitação da palavra PRINT. Se for impresso apenas o número 0 ao invés da mensagem, indica que se esqueceu de colocar a primeira aspa. Em ambos os casos pode-se redigitar o comando, porém com mais cuidado! Os computadores são muito meticulosos com sintaxe e pontuação. Um pequeno erro pode fazer o computador reclamar, ou pior ainda, fazer as coisas erradas.

Um comando como o acima faz com que o computador imprima tudo que estiver contido entre as aspas na tela do vídeo.

Existe um limite para o tamanho da mensagem que pode ser colocada entre as aspas. O limite é diferente para o Integer BASIC e o Applesoft, mas em ambos os casos ele excede o tamanho da tela. Isso quer dizer que o comando pode ocupar mais de uma linha da tela. Os comandos de tamanho grande automaticamente pulam para a linha de baixo da tela. Tente este exemplo:

```
PRINT "SOB CIRCUNSTANCIAS NORMAIS, O HO
MEN PODE SER CONSIDERADO LOUCO"
SOB CONDICOES NORMAIS O HOMEM PODE SER
CONSIDERADO LOUCO
```

O Integer BASIC permite até 120 caracteres. Se este limite for excedido, aparecerá a mensagem *** TOO LONG ERR após acionar o RETURN.

O Applesoft permite até 255 caracteres. Quando se chega ao limite, o Apple II dá um *bip*. Quando o limite é excedido, o Apple II imprime uma barra (\) e automaticamente despreza a entrada, como se tivesse sido acionado o CTRL-X.

IMPRIMINDO CÁLCULOS

Pode-se usar o Apple II em modo imediato como uma calculadora; ele responde diretamente com o resultado de cálculos aritméticos. Tente estes exemplos:

```
PRINT 4+6                    Adição
10
PRINT 500-437                Subtração
63
PRINT 100*23                 Multiplicação
2300
PRINT 96/12                  Divisão
8
PRINT 3^2                    Exponenciação
9
PRINT 3*4*10-800             Combinação
-680
```

A resposta correta se encontra na linha abaixo de cada um dos comandos. Veja que não são necessárias aspas nesses comandos. Sabe o que acontece se elas forem colocadas? Tente se não souber.

O Integer BASIC tem um limite máximo e mínimo para o tamanho dos números nos cálculos. Se o valor é maior que 32767, em qualquer ponto durante os cálculos, aparece a mensagem *** >32767. Se o valor é menor que −32767, aparece a mensagem − *** >32767. Alguns exemplos deste tipo de erro são mostrados abaixo. O último exemplo se refere a uma divisão por zero.

```
>PRINT -32767-2
-*** >32767 ERR
```

```
>PRINT 2^15-1

*** >32767 ERR

>PRINT 10/0

*** >32767 ERR
```

As frações decimais não são permitidas no Integer BASIC. Assim, se for dada uma instrução para divisão com resultado não inteiro, o resto será desprezado. Por exemplo, teste este cálculo:

```
>PRINT 9/2

4
```

O Applesoft permite operação com fracionários. Os valores numéricos podem ter até nove dígitos significativos, incluindo parte inteira e decimal. Isso quer dizer que números com mais de nove dígitos são arredondados para nove ou para o menor dígito diferente de zero. Estes exemplos ilustram como ele opera:

```
]PRINT 12.34567896          Arredondado para cima

12.345679

]PRINT 12.34567894          Arredondado para baixo

12.3456789
```

Se for tentado algum cálculo no modo imediato dentro do Applesoft, pode acontecer que o resultado venha em notação científica.

```
]PRINT 123456789123

1.23456789E+11
```

Se houver dificuldade em entender a notação científica, fique com os cálculos simples por enquanto. Abordaremos mais detalhes sobre notação científica e valores numéricos ainda neste capítulo.

Comando PRINT Abreviado

O Applesoft permite que se abrevie o comando PRINT com o ponto de interrogação (?). Aqui estão alguns exemplos que podem ser tentados:

```
]? "O TEMPO PASSA"

O TEMPO PASSA

]? 13-46*6

-263
```

MENSAGENS DE ERRO

Dissemos antes que o Apple II emite várias mensagens de erro quando aparecem situações em que ele não consegue operar. Ele gera o *bip* para chamar a atenção para o fato de que algo está errado, e tenta diagnosticar o problema. Sua capacidade de diagnóstico é limitada, de forma que não se deve esperar uma análise definitiva sobre os erros. Existem cerca de 35 mensagens que o Apple II pode usar em resposta ao universo de erros e suas combinações.

Ao longo deste capítulo e do resto do livro faremos menção a alguns dos erros que podem ocorrer em situações específicas, especialmente as mais comuns. Todas as mensagens de erro estão listadas em ordem alfabética no Apêndice C.

Formato da Mensagem de Erro

As mensagens de erro têm um formato ligeiramente diferente entre as duas versões do BASIC, como mostrado abaixo:

```
]PRNIT "A LAVA CORRE"
?SYNTAX ERROR
]
PRNIT "A LAVA CORRE"
*** SYNTAX ERROR
)
```

ESPAÇOS EXTRA EM BRANCO

Se existe preocupação quanto à questão de onde colocar espaços em branco, não se preocupe. O Apple II interpreta uma linha de comando BASIC pelos elementos nela colocados. Espaços, ou brancos, são irrelevantes. O único lugar em que deve ser colocado espaço é dentro das aspas, onde se deseja espaço como parte da mensagem.

COMANDOS, LINHAS E PROGRAMAS

Um programa consiste em um ou mais comandos que dão ao Apple II uma exata e completa definição da tarefa que deve ser executada. Se a tarefa é pequena e simples, o programa também será pequeno e simples. As instruções de modo imediato que abordamos há pouco são, de fato, pequenos programas. Cada um tem apenas uma instrução – um comando do Apple II. São casos triviais. Muitos programas têm 10, 100, 1000 ou até mais comandos. Considere os seguintes comandos:

```
PRINT "RATOS"

RATOS

PRINT "COBRAS"

COBRAS

PRINT " E LAGARTOS"

 E LAGARTOS
```

Cada um desses programas em modo imediato imprime uma linha na tela. Cada programa tem exatamente um comando e exatamente uma linha.

O Applesoft permite que se coloque mais de um comando em cada linha. Separam-se os comandos na mesma linha pelo sinal de dois pontos (:). Compare este programa em modo imediato com o exemplo acima:

```
]PRINT "RATOS":PRINT "COBRAS":PRINT " E
 LAGARTOS"
RATOS
COBRAS
 E LAGARTOS
```

Este programa de três comandos e uma linha imprime as mesmas três linhas de texto dos três programas independentes do exemplo anterior.

Não existe um limite específico para o número de comandos que se pode colocar numa linha do Applesoft. Lembre-se que uma linha não pode ter comprimento superior a 255 caracteres. Se se digita uma linha muito longa, o computador começa a emitir o *bip* a partir do caractere de número 248 informando que está chegando ao limite. Se o limite é atingido, a linha é automaticamente cancelada, como se fosse digitado CTRL-X, e deve-se recomeçar. Ele não executa nenhuma das instruções colocadas naquela linha. Desta forma vê-se que existe um limite claro do que se pode fazer em uma linha de comando no modo imediato.

Uma Linha de Comando no Applesoft

Pode-se colocar um bocado de programa em uma linha dentro do modo imediato, graças à habilidade que o Applesoft tem de operar com mais de um comando dentro de cada linha de 255 caracteres. Por exemplo, vejamos o exemplo abaixo:

```
]FOR I=1 TO 800:?"A";:NEXT:?"ACABOU!"
```

Sem saber o que este miniprograma faz exatamente, digite-o no teclado exatamente como

mostrado, finalizando com RETURN. Se a digitação for feita corretamente, vai aparecer a letra A em 20 linhas de 40 colunas cada, seguida pela palavra ACABOU!, na 21ª linha.

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ACABOU!
```

A linha de programa fica colocada no alto da tela do lado esquerdo. Isso porque o programa mostra um número de linhas suficiente para posicionar a linha no alto da tela, porém sem eliminá-la.

Quando o programa de uma linha descrito acima termina sua execução, o caractere de entrada do Applesoft é colocado com o cursor a seu lado na linha de baixo da tela.

MODO PROGRAMADO

A forma de programação apresentada acima é educacional, útil e interessante; porém tem limitações sérias, especialmente em Integer BASIC, com sua limitação de um comando por linha. Outra desvantagem do modo imediato é que se deve redigitar o programa a cada vez que se deseja rodá-lo. Existem algumas técnicas avançadas de edição que veremos de forma rápida e que permitem usar novamente o programa no momento em que ele aparece na tela. Mas, de qualquer forma, este fato ainda é uma desvantagem.

O que se precisa é uma forma de se entrar com várias linhas de programa e mantê-las disponíveis de forma que se possa escrever rotinas que são muito complexas para programas de apenas uma linha.

Existe uma forma de contornar estes problemas que se verificam no modo imediato, que é escrever programas no chamado *modo programado*, também chamado indireto. No modo programado, o computador aceita e guarda o programa em sua memória, mas não executa nenhuma das instruções mandadas até que se dê a ordem para executá-las. Pode-se colocar tantas linhas de programa quanto se desejar. Então, entrando com a ordem apropriada, o computador executa a operação especificada pelo programa entrado.

Execução de Programa

Dissemós que o computador *executa* ou *roda* um programa quando ele executa as operações especificadas por ele.

No modo imediato, o programa é executado no momento em que se aciona a tecla RETURN.

Em modo programado deve-se dar o comando RUN para executar o programa. Cada vez que se faz isso, o programa é executado novamente.

Limpando Programas Velhos

Devido ao Apple II guardar os programas na memória dentro do modo programado, deve-se colocar uma ordem especial para apagar os programas velhos antes de se executar a digitação de um novo. Para fazer isso deve-se dar o comando NEW. Se esta ordem for esquecida, um novo programa entrado vai se misturar com o velho.

Finalizando Programas Corretamente

O fim de um programa em modo imediato é óbvio. Isso não acontece no modo programado, como veremos. O comando END, dentro do programa diz ao BASIC para terminar a execução do programa e retornar ao modo imediato. Assim o comando END deve ser o último comando dentro de um programa a ser executado.

O Applesoft não requer o comando END. Ele finaliza o programa automaticamente após executar todas as instruções colocadas.

Números de Linha

Os *números de linha* tornam o modo programado possível. Um número de linha é simplesmente um número de um, dois, três, quatro ou cinco dígitos, digitado no início de uma linha de comando. Este número de linha é a única diferença que existe entre uma instrução no modo programado e imediato. Existem algumas instruções que só podem ser usadas no modo imediato e outras que só se usam no modo programado; isso será visto mais tarde em detalhes.

Vejamos um exemplo de programação em modo programado:

```
]NEW
]10 PRINT "TODO DIA, TODA HORA"
]20 END
]RUN
TODO DIA, TODA HORA
```

Cada número de linha deve ser único. Não pode haver duas linhas em um programa com o mesmo número. Se isso for feito, o computador guarda a última linha digitada. Para ver isso coloquemos o seguinte programa:

```
>NEW

>10 PRINT "PRIMEIRA LINHA 10"

>10 PRINT "SEGUNDA LINHA 10"

>20 END

>RUN

SEGUNDA LINHA 10
```

Os números de linha determinam seqüência das linhas de programa dentro do BASIC. A primeira linha deve ter o número menor, enquanto que a última deve ter o número maior. Mesmo digitando as linhas fora de ordem, o Apple II rearranja as linhas na seqüência pelos números das linhas. Vejamos um programa, com os números das linhas fora de ordem:

```
>NEW

>30 PRINT "TRINTA"

>10 PRINT "DEZ"
>20 PRINT "VINTE"
>40 PRINT "QUARENTA"
>50 END

>RUN

DEZ
VINTE
TRINTA
QUARENTA
```

Para provar que o Apple II não perdeu o programa colocado dentro do modo programado, limpe a tela com o comando Esc-@ e mande rodar o programa novamente.

```
> ← Acione Esc-@ e RETURN
>RUN
DEZ
VINTE
TRINTA
QUARENTA
```

Pode-se adicionar comandos a um programa que esteja dentro da memória do computador. Elas podem ser adicionadas no início, no fim ou em qualquer parte do meio do programa colocando o número de linha que vai posicionar a instrução onde ela é necessária. Vamos supor que se deseja colocar uma linha no início do programa acima. Como não foi dado o comando NEW, o

programa ainda está dentro da memória do Apple II. Como a menor linha de programa no exemplo é 10, qualquer comando com número de linha inferior a 10 será colocado no início do programa. Vejamos:

```
>5 PRINT "CINCO"

>RUN

CINCO
DEZ
VINTE
TRINTA
QUARENTA
```

É uma boa prática começar o programa original com o número de linha 10 ao invés de 0! Recomenda-se iniciar o programa com um número de linha um pouco alto e pular números entre uma linha e outra, de forma a deixar espaço para a digitação de outras linhas que forem necessárias, mais tarde.

Linhas de Programa Multiinstrução

Pode-se colocar mais de uma instrução dentro de uma linha de programa. O primeiro comando segue o número da linha. A segunda instrução segue a primeira com o caractere dois pontos (:) entre as duas. Este sinal (:) separa as instruções dentro de uma linha de mais de uma instrução.

O Integer BASIC permite que se coloque mais de uma instrução em cada linha dentro do modo programado, o que não acontece no modo imediato. O comprimento de cada linha é de aproximadamente 150 caracteres. O comprimento exato varia de acordo com o conteúdo da linha. Se é digitada uma linha muito extensa, o Apple II apresenta a mensagem *** TOO LONG ERR, e a linha deve ser redigitada.

No Applesoft são permitidas linhas de multiinstrução tanto no modo imediato quanto no programado. Em ambos o comprimento máximo é 255 caracteres, como já foi dito.

Listando as Linhas de Programa

Pode-se ver as linhas de programa que o computador guardou dentro da memória digitando-se o comando LIST. Tente agora. Se não foi digitado NEW depois do último exemplo, pode-se ver as linhas de comando do programa anterior mostradas na tela:

```
LIST

  5 PRINT "CINCO"
 10 PRINT "DEZ"
 20 PRINT "VINTE"
 30 PRINT "TRINTA"
 40 PRINT "QUARENTA"
 50 END
```

Isso se chama *listagem do programa*. Existem variações do comando LIST que permitem listar uma única linha do programa ou um conjunto delas. Esta última opção é útil quando se tem programas longos cujas listagens não cabem na tela de uma só vez. Com o exemplo de cima, tentemos o comando:

```
LIST 10
```

Ele causa o aparecimento de:

```
10 PRINT "DEZ"
```

Para listar várias linhas em seqüência, deve-se especificar as linhas inicial e final no comando, como neste exemplo:

```
LIST 20,40
   20 PRINT "VINTE"
   30 PRINT "TRINTA"
   40 PRINT "QUARENTA"
```

Em Applesoft pode-se listar todas as linhas de programa até uma especificada, inclusive. Pode-se também iniciar a listagem numa linha específica e ir até o fim do programa. Aqui estão exemplos das duas variações do comando LIST:

```
]LIST,10
 5 PRINT "CINCO"
10 PRINT "DEZ"

]LIST30,
30 PRINT "TRINTA"
40 PRINT "QUARENTA"
50 END
```

Interrompendo a Listagem

Pode-se parar a listagem antes que ela termine digitando-se CTRL-C. Isso é especialmente útil para interromper listagens intermináveis quando o programa é muito grande.

Se o Apple II tem o Autostart Monitor, pode-se suspender, ou parar temporariamente a execução da listagem digitando-se CTRL-S. Ela continuará a ser executada com o acionamento da barra de espaço. O CTRL-S permite que se veja um trecho de listagem do programa de cada vez.

Numeração Automática de Linha

O Apple II automaticamente renumera as linhas de programa do Integer BASIC. Usa-se o comando AUTO para realizar esta operação. Desta forma o computador vai fornecendo o próximo número de linha a cada vez que se aciona a tecla RETURN. Ele não avança para a próxima linha se encontra um erro na linha terminada, ou até que se acione a tecla RETURN.

Para sair da numeração automática de linhas, digita-se CTRL-X. Isso cancela a inserção de números de linha pelo computador. Em seguida, digita-se o comando MAN.

Para ilustrar como os comandos AUTO e MAN funcionam, vejamos o exemplo:

```
>AUTO 1000

>1000 PRINT "QUANTAS JARDAS TEM UMA MILHA?"
>1010     Acione apenas RETURN
>1010 PRIMT 5280/3

*** SYNTAX ERR

>1010 PRINT 5280/3
>1020 \  Acione CTRL-X
>MAN
>
```

Como se pode ver no exemplo, o AUTO exige que se informe a partir de que linha deve começar a numeração automática. Pode-se especificar também o incremento a ser dado entre cada linha. O exemplo abaixo ilustra essa opção.

```
>AUTO 1000,100
>1000 PRINT "SER OU NAO SER"
>1100 \    Acione CTRL-X
>MAN
```

Neste exemplo, os números das linhas têm incremento de 100. Se for omitido o incremento, o Apple II utiliza 10 como visto no exemplo anterior.

GUARDANDO PROGRAMAS EM CASSETTE

Um gravador cassette fornece meios de guardar um programa introduzido no modo programado fora do computador e mais tarde carregá-lo de volta na memória. Suponha que se tem o seguinte programa na memória:

```
10 PRINT "MUITA"
20 PRINT "BANANA"
30 PRINT "POR"
40 PRINT "UM"
50 PRINT "TOSTAO"
60 END
```

Para salvar este programa, coloque uma fita cassette no gravador. Rebobine até o início, acione os botões de RECORD e PLAY simultaneamente no gravador cassette e digite o comando seguinte no teclado:

```
SAVE
```

O Apple II vai dar um *bip* quando iniciar a gravação do programa em fita e outro quando terminar. Aperte o botão STOP no gravador após ouvir o segundo *bip*.

Nesse ponto digite NEW seguido de LIST para limpar o programa da memória principal e verificar que ele se foi. Para carregar o programa de volta da fita para a memória, primeiro rebobine a fita até o início. Acione então o botão PLAY do gravador cassette e digite o seguinte comando no teclado:

```
LOAD
```

O Apple II toca um *bip* no instante em que inicia a leitura do programa da fita para a memória e outro quando termina a leitura. Deve-se parar manualmente o gravador após ouvir o segundo *bip*. Use o comando LIST para verificar se o programa está dentro do computador.

No Capítulo 5 veremos como guardar e carregar programas no disco, que é mais conveniente que o cassette.

Guardando Vários Programas em Uma Única Fita

Como se pode ver executando o exemplo acima, não é necessário muita fita para se guardar um programa. Os programas maiores requerem mais fita, porém em geral existe espaço em um cassette para se guardar vários programas em BASIC. Pode-se guardar os programas em forma seqüencial numa fita: o segundo após o primeiro, o terceiro depois do segundo etc.

Carregar o segundo, terceiro, quarto e subseqüentes programas do cassette não é uma tarefa direta, como é o primeiro. Após rebobinar a fita até o início, deve-se passar o primeiro programa para acessar o segundo, passar o segundo para acessar o terceiro e assim por diante. Pode-se obter isso dando o comando LOAD repetidas vezes até que o programa desejado seja lido para a memória. Esse é um processo demorado, mas funciona.

Pode-se aumentar a velocidade consideravelmente se o gravador cassette tiver um contador de fita. Reinicie o contador para 0 quando rebobinar totalmente a fita. Após gravar o primeiro programa anote o número indicado no contador de fita. Essa será a posição inicial para leitura do segundo programa. Grave o segundo programa e anote a leitura do contador de fita no final da gravação (será a posição inicial de leitura do terceiro programa).

Agora, para carregar o segundo programa, rebobine a fita ao início e zere o contador. Dê avanço rápido na fita com o botão FAST FORWARD até posicionar o contador de fita no início do segundo programa. Pode-se usar o botão REWIND para voltar a fita se ela avançou demais. Agora basta usar o comando LOAD para ler o segundo programa.

TROCANDO DE BASIC

Em várias versões do Apple II, pode-se escolher entre programar em Integer BASIC ou Applesoft. As razões de se trocar um BASIC por outro ficarão claras no decorrer deste capítulo. Já vimos, por exemplo, que o Integer BASIC proporciona numeração automática de linhas, o que o Applesoft não faz. Por outro lado, o Applesoft permite trabalhar com valores numéricos fracionários, e o Integer BASIC não.

Muitas versões do Apple II operam ambos os BASIC's, porém nem todas. O Apple II standard, por exemplo, tem apenas o Applesoft. O processo para trocar de uma versão do BASIC

para outra varia de acordo com o modelo de Apple II utilizado, e dos opcionais que estão conectados nele.

Com o Language System da Apple, tem-se acesso imediato a ambas as versões do BASIC. Estando em Integer BASIC digite FP para passar para Applesoft. Dentro do Applesoft digite INT para acessar o Integer BASIC.

O cartão com o Applesoft em "firmware" também permite acesso imediato a ambas as versões do BASIC. Ele tem uma chave que se acessa através do slot atrás do Apple II e determina a versão do BASIC. Ligando-se o computador com a chave para baixo, acessa-se o Integer BASIC. Com a chave para cima, ao se ligar o Apple II acessa-se o Applesoft. Fora o posicionamento da chave, pode-se digitar FP para acessar o Applesoft e INT para acessar o Integer BASIC se o cartão estiver instalado.

No Apple II standard é fácil acessar o Integer BASIC. Acione a tecla RESET e digite CTRL-B. Nesta versão do Apple II é mais difícil se ter o Applesoft, uma vez que o programa vem em fita ou disco. Deve-se fornecer as instruções necessárias para ele carregar o Applesoft da fita ou do disco.

Antes de se usar o drive de disco deve-se dar o *boot* do Sistema Operacional em Disco (DOS) como foi descrito no Capítulo 2. Deve-se fazer isso a cada vez que se liga o computador. Para repassar rapidamente, estas são as instruções para dar o boot do DOS no Monitor e no Integer BASIC.

```
*6      Acione CTRL-P e RETURN

]PR#6
```

Uma vez dado o boot, pode-se colocar o Apple II no Applesoft a partir do Integer BASIC, colocando o disco com o Applesoft no drive de disco e digitando FP. Em poucos segundos o caractere de entrada do Applesoft ( ]) aparece na tela.

Pode-se também ter o Applesoft em fita. O processo completo para sua leitura é descrito no Capítulo 2. Em resumo, deve-se pôr o cassette com o Applesoft no gravador cassette, acionar a tecla PLAY e dar o comando LOAD dentro do Integer BASIC. Em 1 a 2 minutos aparece a nota de Copyright e o caractere de entrada na tela.

Dentro do Applesoft, pode-se voltar ao Integer BASIC digitando o comando INT. Caso se deseje voltar ao Applesoft, usa-se o comando FP com o drive de disco, ou o LOAD com a fita cassette.

TÉCNICAS AVANÇADAS DE EDIÇÃO

No Capítulo 2 vimos algumas formas de corrigir erros durante a digitação da linha de comando, antes do acionamento da tecla RETURN. Vamos, rapidamente, rever as técnicas apresentadas.

A tecla ← volta o cursor de um espaço e limpa o caractere passado. Os caracteres são eliminados da linha de programa, ainda que continuem aparecendo na tela.

A tecla → move o cursor para a frente, copiando (reentrando) o caractere passado.

A tecla REPT, usada em conjunto com as teclas ← e →, permite a ida para a frente e para trás mais rapidamente.

O comando CTRL-X cancela a linha que está sendo digitada.

O comando Esc-@ limpa a tela e coloca o cursor na linha superior à esquerda.

Vamos agora ver outras formas de editar linhas de programa. Estes novos métodos são particularmente úteis quando se deseja modificar as instruções no modo programado (ou seja, aquelas com número de linha).

ELIMINANDO LINHAS DE PROGRAMA

Para eliminar uma linha inteira, digite seu número seguido imediatamente pela tecla RETURN. Ao listar o programa, nota-se que tanto o número da linha quanto o seu conteúdo não mais fazem parte do programa. Aqui está um exemplo:

```
>NEW

>100 PRINT "MEU CORACAO"
>110 PRINT "NAO SEI PORQUE"
>120 PRINT "BATE FELIZ"
>130 PRINT "QUANDO TE VE"
>140 END
>110
>130

>LIST
   100 PRINT "MEU CORACAO"
   120 PRINT "BATE FELIZ"
   140 END
```

Pode-se usar o comando DEL para eliminar um bloco de linhas de programa. Por exemplo:

```
>DEL 110,140

>LIST

   100 PRINT "MEU CORACAO"
```

O comando DEL 110,140 eliminou todas as linhas de programa, iniciando na linha 110 e terminando na 140. Mesmo que a linha 110 não exista, todas as linhas entre 110 e 140 são retiradas.

ACRESCENTANDO LINHAS DE PROGRAMA

Pode-se acrescentar novas linhas de programa em qualquer ordem, a qualquer tempo. Seus números de linha vão determinar a posição que ocuparão no programa. O Apple II automaticamente ordena

as novas linhas e aquelas já existentes na memória. Adicionemos as linhas 110, 120 e 130 no exemplo acima.

```
>110 PRINT "NAO SEI PORQUE"
>120 PRINT "BATE FELIZ"
>130 END

>LIST

  100 PRINT "MEU CORACAO"
  110 PRINT "NAO SEI PORQUE"
  120 PRINT "BATE FELIZ"
  130 END
```

ALTERANDO LINHAS DE PROGRAMA

A forma mais simples de se alterar uma linha de programa é redigitá-la. Essa não é uma boa solução por vários motivos. Redigitar é uma tarefa que consome bastante tempo e as chances de se errar na digitação são grandes. Felizmente há uma maneira de se modificar as linhas de programa que estão na memória sem ter que redigitá-las. A forma pela qual se permite isso, existente tanto no Integer BASIC quanto no Applesoft, tem como base o fato de que tudo que é mostrado na tela está *vivo*. Pode-se editar qualquer coisa que esteja na tela. Usando a tecla Esc em conjunto com várias outras teclas, pode-se mover o cursor ao longo de todas as posições da tela. Isso faz com que se possa mover o cursor para o início de qualquer linha que esteja presente na tela. Pode-se então usar a tecla → para copiar as partes não modificadas do programa. Pode-se substituir, inserir ou eliminar caracteres em qualquer ponto da linha.

Aqui está como isso funciona. Primeiro usa-se o comando LIST para fazer aparecer na tela aquilo que se quer modificar. Perceba que o comando LIST coloca espaços extras, quando mostra as linhas de programa. (Para tornar o programa mais legível.) Esses espaços extras podem tornar mais difíceis de editar linhas de programa muito longas. Para livrar o comando LIST desses espaços extras que podem causar problemas na edição, limpe a tela (usando Esc-@) e digite o seguinte comando misterioso:

```
POKE 33,33
```

Além de suprimir os espaços extras em branco, este comando faz reduzir o tamanho da tela de 40 para 33 caracteres. Veremos o comando POKE em mais detalhes no Capítulo 4. Para fazer voltar a tela ao seu tamanho normal, digite:

```
POKE 33,40
```

Movendo o Cursor

Para mover o cursor pela tela, deve-se acionar duas teclas em seqüência. Primeiro acione a tecla Esc e depois A, B, C e D, toda vez que desejar colocar o cursor uma posição para a direita, esquerda, cima ou baixo. A Figura 3.1 ilustra como as quatro possibilidades afetam o movimento do cursor.

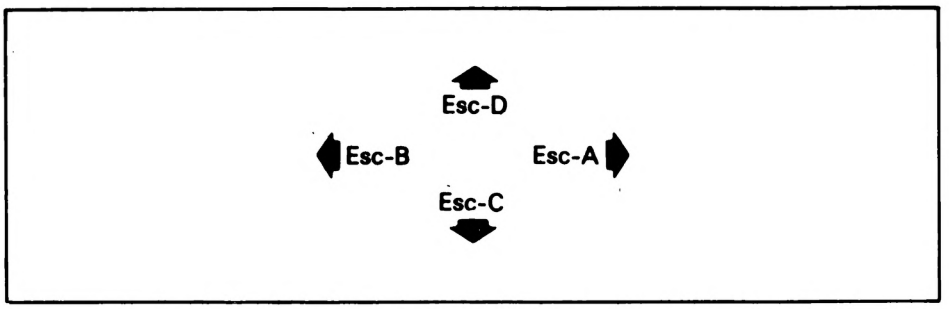


FIGURA 3.1. Movimento do Cursor.

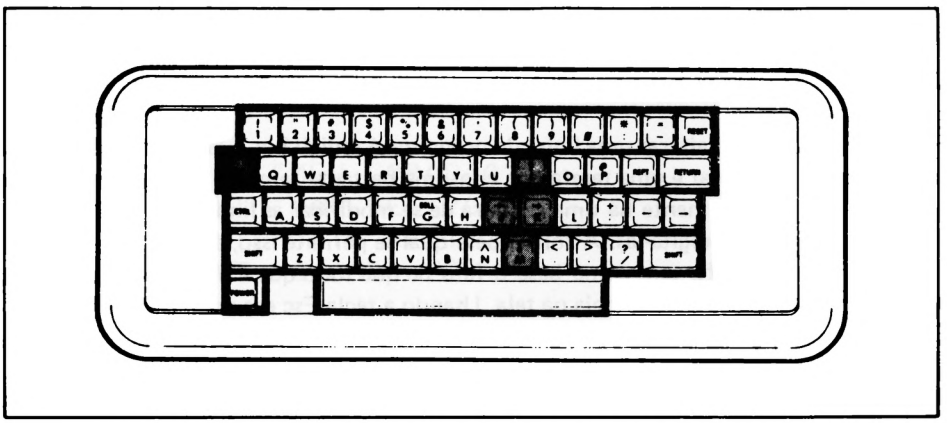


FIGURA 3.2. Movimento do Cursor (Versão Monitor Autostart).

Com o Autostart Monitor, pode-se usar ainda as teclas I, J, K e M em conjunto com Esc para mover o cursor. Pela forma como essas letras estão situadas no teclado, elas formam um painel de controle direcional, como mostrado na Figura 3.2.

Existe uma importante diferença na forma como a tecla Esc opera com I, J, K e M. Acione Esc e o Apple II vai para o *modo de edição*. Acione então I, J, K ou M para mover o cursor ao largo da tela — sem necessidade de acionar Esc toda vez. O Apple II fica no modo edição enquanto se acionar qualquer uma das teclas I, J, K, M ou REPT. Isso permite mover o cursor em mais de um espaço sem ter necessidade de digitar o Esc a cada vez. Para sair do modo de edição, aciona-se qualquer tecla que não seja I, J, K, M, REPT, CTRL ou SHIFT.

Pode-se usar a tecla REPT para fazer o cursor se mover longas distâncias na tela com pouca digitação.

Trocando Caracteres

Trocar um caractere por outro é simples. Posiciona-se o cursor sobre o caractere a ser trocado e digita-se o substituto sobre ele. Por exemplo, com o cursor posicionado no exemplo a seguir:

```
100 PRINT "TEMPO ESTIMADO DE CHEGADA"
```

pode-se digitar DESEMBARQUE, obtendo-se:

```
100 PRINT "TEMPO ESTIMADO DE DESEMBARQUE"
```

Adiciona-se RETURN para efetivar a modificação.

Eliminando Caracteres

Pode-se efetivamente eliminar caracteres individuais colocando-se espaços em branco sobre eles. Deve-se lembrar que no BASIC os espaços em branco não têm maior significado, a menos que estejam entre aspas, quando fazem parte de textos. Pode-se usar as teclas Esc e A (a tecla K no modo edição) para mover o cursor para a frente. Diferentemente da tecla →, as teclas Esc-A e Esc-K não recopiam os caracteres que são passados. Se os caracteres que se deseja eliminar estiverem dentro de aspas, será mais fácil usar as teclas Esc e A ou J no modo edição, para eliminar os caracteres indesejados.

Para limpar todos os caracteres a partir da posição do cursor até o fim da linha mostrada, aciona-se Esc seguido de E. Isso tem o mesmo efeito de se acionar a barra de espaço repetidamente até o fim da linha, porém o cursor não se move. Os caracteres da próxima linha não são limpos, mesmo se fizerem parte da mesma linha de programa. Por exemplo, se for acionado Esc e E tendo-se o cursor posicionado como abaixo:

```
100 PRINT "QUEM NUNCA COMEU MELADO, QU
ANDO COME SE LAMBUZA"
```

resulta em:

```
100 PRINT "QUEM NUNCA
ANDO COME SE LAMBUZA"
```

Cuidado! Se o RETURN for acionado agora, a linha 100 ficará terminada exatamente onde está o cursor. Use a tecla → para copiar o resto da linha.

Pode-se também limpar todo o texto da posição do cursor até o fim da tela. Para isso, deve-se acionar Esc e F.

Inserindo Caracteres

Inserir caracteres numa linha é simples. Pode parecer confuso de início porque o resultado final não é aparente. O Apple II não pode abrir espaço entre caracteres para gerar área de inserção. A Figura 3.3 mostra como o Apple II permite inserção de letras. Insere-se o texto sobre a linha com a ajuda dos movimentos do cursor (Esc etc.). O que se deve lembrar é que o que está mostrado na tela não é uma réplica do que existe na memória do computador.

Temos aqui um exemplo de edição que ilustra a inserção de caracteres. O exemplo usa o Integer BASIC (>) porém opera corretamente também no Applesoft. Vejamos a seguinte linha de programa:

FIGURA 3.3. Inserção de Caracteres

```
>NEW

>10 PRINT "VENDE-SE ESTE MOVEL"
>[]
```

Para inserir a palavra AUTO, na frente da palavra móvel, primeiro listemos o programa:

```
>LIST
   10 PRINT "VENDE-SE ESTE MOVEL"
>[]
```

Usando *apenas* as teclas de movimentação do cursor (Esc etc.), posiciona-se o cursor exatamente sobre o primeiro dígito do número da linha, como segue:

```
>LIST
   [1]0 PRINT "VENDE-SE ESTE MOVEL"
>
```

Usa-se agora a tecla → para copiar a primeira parte da linha, parando em M.

```
>LIST
   10 PRINT "VENDE-SE ESTE [M]OVEL"
>
```

Aciona-se a tecla Esc seguida por D para mover o cursor uma linha para cima. (Se houver caracteres do lado direito do cursor nessa linha, pode-se limpá-los com o acionamento das teclas Esc e E.)

```
>LIST                          [ ]
   10 PRINT "VENDE-SE ESTE MOVEL"
>
```

Digita-se a palavra AUTO

```
>LIST                          AUTO[]
  10 PRINT "VENDE-SE ESTE MOVEL"
>
```

Usando *apenas* os movimentos do cursor, posiciona-se o cursor sobre a primeira letra do texto a ser inserido. Não se deve usar a tecla ← para voltar o cursor. Embora pareça que ela faz a mesma coisa que Esc-B e Esc-J, a tecla ← realmente limpa os caracteres que passam; ela iria eliminar a inserção!

```
]LIST                              [A]UTO
   10 PRINT "VENDE-SE ESTE MOVEL"
]
```

Acione a tecla Esc seguida pela tecla C para retornar o cursor à linha de programa original.

```
]LIST                              AUTO
   10 PRINT "VENDE-SE ESTE [M]OVEL"
]
```

Finalmente use a tecla → para copiar o resto da linha. Então acione RETURN. Mostre a nova linha com o comando LIST.

```
]LIST                              AUTO
   10 PRINT "VENDE-SE ESTE MOVEL"

]LIST 10
   10 PRINT "VENDE-SE ESTE AUTOMOVEL"
][]
```

O Apêndice B contém uma tabela de referência para os comandos de edição.

REEXECUTANDO EM MODO IMEDIATO

O fato de que tudo que aparece na tela do Apple II está vivo permite que se execute qualquer comando colocado em modo imediato, desde que ainda seja visível na tela. Pode-se executar um comando em modo imediato da forma como está, ou ainda pode-se editá-lo.

Em ambos os casos, a primeira coisa a fazer é posicionar o cursor sobre o início da linha de modo imediato. Use as teclas Esc e A, B, C ou D alternativamente como mostrado acima (ou Esc seguida por I, J, K ou M se estiver com Autostart Monitor). Use então a tecla → para copiar a instrução de modo imediato. Pode-se fazer modificações na linha, usando as técnicas mostradas acima para trocar, retirar ou inserir caracteres na linha.

Para ver como isso funciona, observemos o programa em modo imediato abaixo, que calcula o espaço, em pés cúbicos, de uma área de 10 X 25 X 8 pés.

```
]PRINT "ESPACO = ";10*25*8
ESPACO = 2000
```

Pode-se mudar facilmente este programa para calcular o espaço de ambientes de diferentes tamanhos. Para mudar as dimensões para 10 X 25 X 14, por exemplo, primeiro deve-se posicionar o cursor no início da linha de programa em modo imediato. (Alternadamente, acione Esc e D três vezes). Acione e segure a tecla → acionando e mantendo a tecla REPT. O cursor irá para a frente ao longo da linha de programa. Solte ambas as teclas quando o cursor chegar ao dígito 8. Se ele for passado ou não tiver chegado, devido a se ter soltado as teclas antes ou depois, pode-se mover o cursor para a frente ou para trás um caractere por vez com as teclas ← e →. Pode-se também mover o cursor do início até o 8 acionando a tecla → repetidamente várias vezes, ao invés de usar o REPT e → em conjunto.

Com o cursor posicionado sobre o 8, digite a nova dimensão de 14 e acione RETURN.

```
>PRINT "ESPACO = ";10*25*14
ESPACO = 3500
```

LINGUAGEM DE PROGRAMAÇÃO

Uma linguagem de programação é o meio de comunicação pelo qual podemos falar com o computador. Existem muitas linguagens diferentes de programação. Algumas, como o BASIC, são de aplicação geral, outras são orientadas para tornar a execução de programas mais fácil dentro de áreas específicas, como área comercial, científica, gráficos, manipulação de textos, cálculos etc. As linguagens de programação são tão diferentes quanto as linguagens faladas. Além do BASIC, outras linguagens de programação comuns são Pascal, C, FORTRAN, COBOL, APL, PL/M, PL/1 e FORTH.

Os computadores Apple II podem usar várias linguagens de programação, dentre elas o BASIC e o Pascal. Este livro se concentra no uso do BASIC com o Apple II.

Não importando qual a linguagem de programação, todo o comando deve ser dado seguindo um conjunto de normas bem definido. Estas regras, no conjunto são chamadas de *sintaxe*. Cada linguagem diferente tem sua própria sintaxe.

As linguagens de programação, como linguagens faladas, têm *dialetos*. Os dialetos são variações menores da sintaxe. O Apple II tem dois dialetos do BASIC, o Integer BASIC e o Applesoft. Devido às diferenças entre os dois dialetos, um programa escrito para um dialeto pode não rodar no outro. Além disso, um programa escrito para o Apple II pode não rodar em outro computador, mesmo que esse outro computador também execute o BASIC. Entretanto, aprendendo a programar o computador Apple II em BASIC, qualquer que seja o dialeto, faz com que não se tenha grande dificuldade em se trabalhar com qualquer outro dialeto.

Algumas regras sintáticas de linguagens de programação são óbvias. Os exemplos de adição e subtração no início deste capítulo são exemplos de sintaxe óbvia. Não é necessário ser programador para entendê-las. Porém outras regras de sintaxe são arbitrárias e sem sentido, a menos que se conheça a sintaxe. Não se pode procurar racionalizar as regras de sintaxe; usualmente não há racionalidade. Por exemplo, por que usar * para representar multiplicação? Normalmente se usaria o X para multiplicação. Porém o computador não teria meios de diferenciar o X que representa multiplicação do que representa a letra X. Assim, mais recentemente todas as linguagens de programação têm optado pelo asterisco (*) para representar a multiplicação. A divisão é universalmente representada pelo símbolo /. Não existe razão real para esta opção; o sinal de divisão (÷) não está presente no teclado do computador, de forma que outros caracteres devem ser escolhidos.

ELEMENTOS DE BASIC

Muitas das regras básicas de sintaxe do BASIC se referem a comandos individuais. A sintaxe dos comandos BASIC distribuem-se separadamente pelos seus três grupos: números de linhas, instru-

ções ao computador e dados. Descreveremos cada uma a seu tempo. Existem ainda umas poucas regras que se referem ao programa como um todo, como a ordem dos comandos. Abordaremos essas regras nos pontos apropriados no decorrer deste capítulo.

REPASSANDO A NUMERAÇÃO DE LINHAS

Já falamos sobre a numeração das linhas com alguma profundidade. Após uma breve revisão, entraremos em mais detalhes. Em modo programado, toda a linha de programa em BASIC deve ter um único número de linha. Os números de linhas determinam a seqüência das instruções dentro do programa; o comando com o menor número será o primeiro do programa e o de maior número será o último.

O Integer BASIC permite números de 1 a 5 dígitos com números inteiros variando de 0 a 32767.

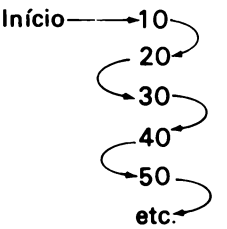
O Applesoft permite números de 1 a 5 dígitos com valores inteiros entre 0 e 63999.

Números de Linha como Endereço

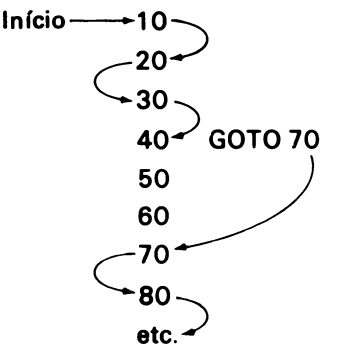
Em essência, os números de linha são uma maneira de endereçar as linhas de programa. Este é um conceito importante, desde que todo o programa tem dois tipos de comandos:

1. Comandos que criam ou modificam dados, e
2. Comandos que controlam a seqüência em que a operação é executada.

A idéia de que a operação especificada pelo programa deve ser executada numa seqüência bem definida é um conceito bastante simples. Normalmente, a execução do programa inicia e continua seqüencialmente, como ilustrado abaixo:



Logo descobriremos que muitos programas contêm seqüências de execução essencialmente não-seqüencial. Isso faz com que os números de linhas se tornem importantes, porque se pode usar o número da linha para identificar uma troca na seqüência de execução. Isso pode ser ilustrado como segue:



ESPAÇOS EM BRANCO

De forma geral, podem-se usar espaços extras em branco livremente para melhorar a legibilidade do programa. Porém, como cada espaço extra em branco ocupa espaço extra de memória, o Apple II coloca os espaços em branco não necessários para fora da linha de programa quando ela é entrada (e aciona-se RETURN). Assim, quando se listam linhas de programa com o comando LIST, o Apple II coloca espaços em branco nos lugares apropriados para aumentar a legibilidade, de acordo com determinações anteriores. Não se deve esquecer que é possível suprimir essa colocação de espaços com o comando TEXT ou digitando POKE 33,33 antes de dar o comando LIST. Um POKE 33,40 faz a tela voltar ao original.

Deve-se ter cuidado ao usar espaços extras dentro de aspas. Compare os dois comandos abaixo, por exemplo:

```
PRINT "COLOQUE OS DADOS INICIAIS"
COLOQUE OS DADOS INICIAIS

PR  INT "      COLOQUE OS DADOS I NICIAIS"
      COLOQUE OS DADOS I NICIAIS
```

DADOS

A principal atividade do computador é entrar, manipular e expelir dados. Por isso, a forma como a linguagem de programação manipula dados, sejam números ou texto, é um parâmetro muito importante. Descreveremos agora os tipos de dados que se podem encontrar em programas no Apple II.

Série de Caracteres

Uma *série* é uma seqüência de caracteres entre aspas. Usamos séries com o comando PRINT para as mensagens que aparecem na tela. Aqui estão mais alguns exemplos:

"DIGITE OS DADOS"
"CONTA NUMERO 105.443-4"
"PEDRO ALVARES CABRAL"
"LARGO DA CONCORDIA"
"SÃO PAULO, 10 DE DEZEMBRO DE 1954"

Com apenas algumas exceções, uma série de caracteres pode conter qualquer caractere que se possa entrar via teclado usando o alfabeto normal e as teclas numéricas, com ou sem SHIFT ou CTRL. As exceções são ←, →, RETURN, Esc, CTRL-H, CTRL-M, CTRL-U e CTRL-X. Essas exceções ou movem o cursor dentro da tela ou finalizam a entrada, ou ambos.

As séries podem ter comprimento de 0 a 255 caracteres. Uma série sem nenhum caractere é chamada de série nula.

Existem alguns caracteres invisíveis que podem ser produzidos acionando certas combinações de teclas. Por exemplo, acionando-se CTRL e G simultaneamente, o computador será o *bip*. Pode-se colocar essa combinação dentro da série. Tente usá-la dentro de um comando PRINT.

```
PRINT "  Acione CTRL-G várias vezes entre as aspas
```

Neste exemplo, pode-se ouvir o caractere embora não se possa vê-lo. Existem outros caracteres que são invisíveis e inaudíveis. Tais caracteres são usados para controlar funções da impressora, periféricos de comunicação e outros componentes que se podem conectar ao Apple II.

O Apêndice I lista o conjunto completo de caracteres do Apple II e informa que teclas geram qual caractere.

Em Applesoft existe uma maneira pela qual se pode incluir, dentro de uma série de caracteres, aqueles que não podem ser gerados pelo acionamento de teclas. Isso é feito com a função CHR$, descrita adiante no Capítulo 4.

Números

Existem dois tipos de números que podem ser guardados dentro do Apple II: *inteiros*, que são números sem parte decimal, e *números reais* (também chamados números em ponto flutuante), que têm parte fracionária. Como se pode suspeitar, o Integer BASIC só aceita números inteiros. O Applesoft aceita tanto inteiros como reais.

Devem-se expressar todos os números com vírgula. Por exemplo, deve-se usar 32000, e não 32.000.

Inteiros

Um inteiro é um número que não tem parte decimal ou ponto decimal. O número pode ser negativo (−) ou positivo (+). Um número sem sinal é tratado como positivo. Os números inteiros devem ter valores na faixa entre −32767 e 32767. A seguir, alguns exemplos de inteiros:

```
0
1
44
32699
−15
```

Números Reais

Um número real pode ser inteiro com parte fracionária ou fração. O número pode ser negativo (−) ou positivo (+). Se não tiver sinal assume-se como positivo. O menor (mais negativo) número real é:

−1000000000000000000000000000000000000

e o maior é:

100000000000000000000000000000000000

Aqui estão alguns exemplos de números reais:

5
−15
65000
161
0
0.5
0.0165432
1.6
24.0055
−64.2

Quando o valor fracionário de um número chegar próximo de zero em cerca de .000000000000000000000000000000000000003, será convertido para zero.

Notação Científica

Números reais muito grandes ou muito pequenos são representados em Applesoft usando *notação científica*. Qualquer número que tenha mais de nove dígitos à frente do ponto decimal será expresso em notação científica. Qualquer número fracionário mais próximo de zero que ±1 será expresso em notação científica.

Um número em notação científica tem a forma:

número E±*ee*

onde:

número é um inteiro, fracionário ou combinado, como ilustrado acima. A porção *número* contém os dígitos numéricos significantes, e chama-se *coeficiente.* Se não aparece o ponto decimal assume-se que ele está à direita do coeficiente.

E é sempre uma letra. Vem da palavra *expoente*.

± é o sinal opcional mais ou menos.

ee é o expoente com um ou dois dígitos. O expoente indica a magnitude do número, isto é, o número de posições à direita (expoente positivo) ou à esquerda (expoente negativo) de quanto o ponto decimal deve ser movido para fornecer a posição real do ponto decimal.

Aqui estão alguns exemplos de notação científica:

Notação Comum	Notação Científica
1000000000	1E + 09
.000000001	1E − 09
200	2E + 02
−123456789	−1.23456789E + 09
−.0000123456789	−1.23456789E − 06

Como se pode ver, a notação científica é uma forma conveniente de expressar números muito grandes ou muito pequenos. Os valores máximo e mínimo dos números reais, que foram expressados como um punhado de zeros, podem também ser expressados como 1E+38 e 1E−38, respectivamente (muito mais compactos). De forma idêntica, o número mais próximo de zero é 3E−38.

Arredondamento

Mencionamos anteriormente que os números reais podem ter até 9 dígitos de precisão. Para um número maior que 1 ou menor que −1, significa que apenas os nove dígitos mais à esquerda podem ser diferentes de zero. O Apple II arredonda e despreza qualquer dígito excedente a nove. Temos alguns exemplos (veja que números muito grandes são expressos em notação científica):

```
]PRINT 1234567891
1.23456789E+09
]?-123456789123456789
-1.23456789E+17
]?-150000475.75
-150000476
]?90000000.7558
90000000.8
```

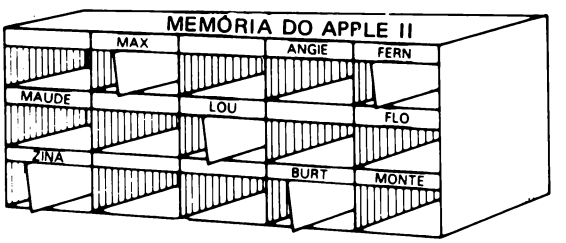
Números fracionários (entre −1 e 1) estão sujeitos às mesmas limitações. Neste caso, entretanto, os nove dígitos iniciam com o primeiro dígito diferente de zero à direita do ponto decimal. Aqui estão alguns exemplos:

```
]PRINT .1234567891
.123456789
]?-123456789123456789
-1.23456789E+17
]?-123456789 123456789
-1.23456789E+17
]?.000000000900000007558
9.00000008E-10
```

VARIÁVEIS

Até o presente, na discussão de dados só consideramos valores constantes. Às vezes é mais fácil referenciar um item pelo seu nome ao invés de seu valor. E aí entra o conceito de *variável.*

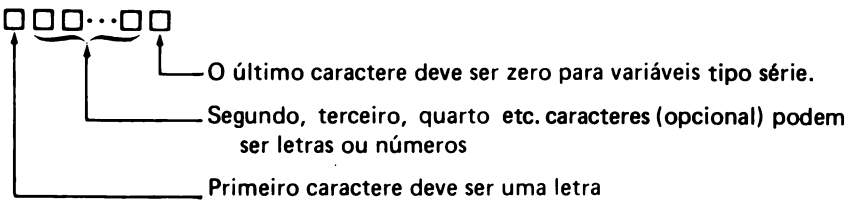
Para aqueles que já estudaram elementos de álgebra, os conceitos de variável e nome de variável são fáceis de serem entendidos. Para os que nunca estudaram álgebra, pode parecer que nomes de variáveis são como nome que se dão a caixas postais. Qualquer coisa colocada dentro da caixa passa a ser o valor associado com o nome da caixa, até que alguma coisa nova seja colocada na caixa. Em "computês", dizemos que um valor foi armazenado na variável.



Nem sempre uma variável se refere ao mesmo valor. Isso é muito importante — ela pode representar qualquer valor permitido. Pode-se alterar seu valor no curso do programa. O BASIC tem uma série de comandos para essa finalidade; eles serão vistos mais tarde.

Nomes de Variáveis em Integer BASIC

Os nomes de variáveis em Integer BASIC podem ter de 1 a 100 caracteres. Essas são as regras gerais para construir nomes de variáveis no Integer BASIC:



O último caractere do nome da variável diz ao BASIC que tipo de dado ela representa — série ou numérico.

As variáveis série referenciam uma seqüência de caracteres com tamanho de 0 a 255. Os caracteres em branco também contam dentro da série. Antes de se usar uma variável série, deve-se especificar seu tamanho máximo. Faz-se isso com o auxílio do comando DIM, que será descrito mais tarde. Se isso não for feito, será emitida a mensagem de erro *** STR OVFL ERR. Seguem adiante alguns exemplos de nomes de variáveis, legais e ilegais:

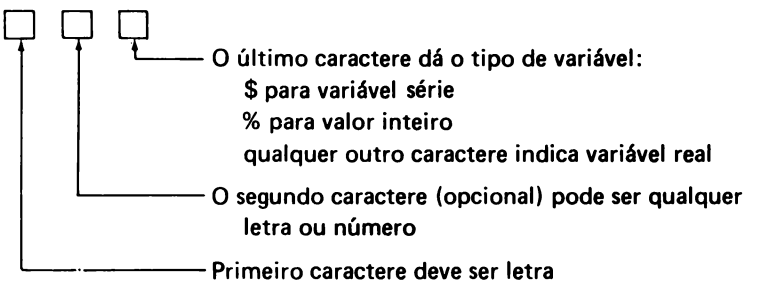
Legais	Ilegais
A$	$
NOMEADO$	9$
PARTE1$	NOME.DADO$
RESPOSTA$	
X8$	

Os valores numéricos no Integer BASIC devem ter valores contidos entre −32767 e +32767. Se esses limites forem ultrapassados, é dada a mensagem de erro *** > 32767 ERR. Abaixo seguem alguns exemplos de valores numéricos legais e ilegais:

Legais	Ilegais
A	APLICAÇÃO'BASE
CÓDIGOA	3X4Z
X0	$TOTAL

Nomes de Variáveis em Applesoft

Um nome de variável pode ter um, dois ou três caracteres no Applesoft. Aplicam-se as seguintes regras:



O último caractere diz ao Applesoft que tipo de dado a variável representa.

Uma *variável série* em Applesoft pode guardar um valor de tamanho entre 0 e 255 caracteres. Não se especifica o tamanho da série como no Integer BASIC; O Applesoft não necessita disso. Aqui estão alguns exemplos de nomes de variáveis, legais e ilegais:

Legais	Ilegais
A$	0?
MN$	M!$
F6$	77$

Variáveis inteiras referenciam números inteiros entre −32767 e 32767. Se esse limite for excedido, é dada a mensagem ?ILLEGAL QUANTITY ERROR. Se for tentado armazenar um valor real numa variável inteira, o Applesoft converterá o real para inteiro antes. Vamos ver as regras de conversão rapidamente. Aqui estão alguns exemplos de variáveis inteiras legais e ilegais do Applesoft:

Legais	Ilegais
A%	A$%
B%	31%
A1%	3D%
X4%	

Variáveis reais referenciam valores numéricos restritos na faixa de -10^{38} a $+10^{38}$, embora se possa trabalhar com valores de até 1.7×10^{38} (±) sob certas circunstâncias. Se for tentado (numa variável real) guardar um número maior, será dada a mensagem ?OVERFLOW ERROR.

Quando o valor de uma variável real chega mais próxima de zero do que $\pm 2.9388 \times 10^{-39}$ o Applesoft converte a zero.

Lembre-se: as variáveis reais podem ter valores inteiros, desde que o inteiro é uma real com a parte fracionária igual a zero. Aqui vão alguns exemplos de variáveis reais legais e ilegais:

Legais	Ilegais
A	0
B	7B
A1	A#
AA	
Z5	

Nomes Grandes de Variáveis em Applesoft

Os nomes de variáveis podem ter mais de dois caracteres alfanuméricos (mais o % ou $ para inteiros e séries), mas apenas os dois primeiros caracteres são considerados no Applesoft. Assim, PREÇO1 e PREÇO2 são o mesmo nome, porque ambos iniciam com PR. No entanto, PREÇO1 e PREÇO1% são diferentes, porque têm diferentes tipos de sufixo.

O Applesoft permite nomes de variáveis de até 255 caracteres.

Aqui estão alguns exemplos de nomes de variáveis com mais de dois caracteres:

Legais	Ilegais
CONTADOR%	ITEM#%
BALANÇOANUAL	2ATABELA
NOME$	ENDEREÇO.COMERCIAL%

Tenha os seguintes pontos em mente para uso de nomes de variáveis com mais de dois caracteres:

1. Apenas os primeiros dois caracteres e o sufixo do tipo da variável (% ou $) são significativos. Não se deve usar nomes com extensões do tipo CONTA1% e CONTA2%; eles referenciam a mesma variável: CO%.
2. Caracteres adicionais precisam espaço extra de memória, que podem ser necessários para os programas maiores. Porém a vantagem de se usar nomes de variáveis maiores é que eles tornam o programa mais fácil de ser lido. CODITEM, por exemplo, tem mais significado do que CO, para o nome da variável que fornece o código do item num programa de controle de estoque.

Palavras Reservadas

Todas as palavras que definem operações de comandos BASIC são chamadas *palavras reservadas*. O Apêndice F fornece todas as palavras reservadas tanto do Integer BASIC quanto do Applesoft. Muitas dessas palavras reservadas já foram vistas neste capítulo, e outras serão descritas em outras partes do livro.

Na execução de um programa BASIC, o Apple II varre todos os comandos BASIC, procurando séries que se constituam em palavras reservadas. A única exceção são as séries de texto entre aspas. Pode causar problema a inserção de uma palavra reservada como nome de variável. O Apple II não é esperto o suficiente para perceber um nome de variável apenas por sua localização no comando BASIC. Assim, *deve-se ter muito cuidado para não incluir palavras reservadas como nomes de variáveis*; isso é especialmente importante com as palavras reservadas menores que podem facilmente ser tomadas como nomes de variáveis.

MATRIZES

Matriz é apenas uma forma sistemática de se dar nomes a uma lista grande de variáveis. Elas são usadas freqüentemente em muitos tipos de programas de computador. Se for difícil entender o que significa matriz, ou como usá-la, leia esta seção; a informação que segue é muito importante em seu desenvolvimento para programação.

Conceitualmente, uma matriz é algo muito simples. Quando se tem dois ou mais itens, ao invés de se dar um nome a cada variável, dá-se um nome só ao conjunto inteiro. Esse conjunto é chamado *matriz*, e seu nome é o *nome da matriz.* Cada um dos elementos componentes é um *elemento da matriz.* Os elementos são numerados. Seleciona-se um elemento usando seu número de posição, que se chama *índice.*

Arranjos são uma forma compacta de descrever um amontoado de variáveis relacionadas. Veja, por exemplo, uma tabela de 200 números. Pense como seria penoso associar cada um dos elementos da tabela com um nome de variável. É muito mais simples dar-se um único nome à tabela inteira e identificar cada um dos números pela sua posição. Isso é precisamente o que faz a matriz.

Como exemplo de aplicação da matriz, pensemos num hotel com dez quartos com a necessidade de saber quem está em qual quarto.

PEDRO	MARIA	JOSÉ		PAULO	LUIZ	SÉRGIO	MÁRCIA	CELSO	
Q1$	Q2$	Q3$	Q4$	Q5$	Q6$	Q7$	Q8$	Q9$	Q10$

Ou se pode considerar a lista de clientes do hotel como uma matriz.

PEDRO	MARIA	JOSÉ		PAULO	LUIZ	SÉRGIO	MÁRCIA	CELSO	
Q$(1)	Q$(2)	Q$(3)	Q$(4)	Q$(5)	Q$(7)	Q$(7)	Q$(8)	Q$(9)	Q$(10)

Nesse exemplo, R$ é o nome da matriz. Ele tem dez elementos; cada elemento é o nome do ocupante de cada quarto. O índice (entre parênteses) segue cada nome da variável. Assim um dado específico (por exemplo, o ocupante do quarto 1) é identificado pelo nome da variável e um índice. O ocupante do terceiro quarto é o valor de Q$(3), que é JOSÉ.

O exemplo acima usa matriz de séries porque eles são mais intuitivos para se entender. Porém, o Integer BASIC só permite matrizes numéricas.

As matrizes do Applesoft podem representar variáveis inteiras, reais ou séries; no entanto, uma matriz simples só pode representar um tipo de dado. Em outras palavras, uma variável simples não pode misturar números inteiros e reais, à exceção de que uma variável real pode conter um número inteiro (não vice-versa). Cada tipo de matriz usa uma quantidade diferente de espaço de memória; veja o Apêndice G para detalhes.

Dimensões de Matrizes

Deve-se especificar o número de elementos de uma matriz antes de usá-la no Integer BASIC. Faz-se isso com o comando DIM. Os comandos DIM serão descritos adiante neste capítulo.

O Applesoft permite usar matrizes com até 10 elementos sem ter que defini-las antes.

As matrizes do Applesoft podem ter mais de uma dimensão, o que quer dizer que existirão mais de um índice para selecionar cada elemento. Uma matriz com uma só dimensão é equivalente a uma tabela com apenas uma linha de elementos. O índice identifica o elemento dentro da linha. Uma matriz com duas dimensões é como uma tabela de números, montados em forma de linhas e colunas; um índice identifica a linha e o outro a coluna. Pode-se visualizar uma matriz de três dimensões como um cubo de números, ou mesmo uma pilha de tabelas. Quatro ou mais dimensões na matriz é mais difícil de visualizar porém, sob o ponto de vista matemático, não é mais complicado que uma matriz simples de uma só dimensão.

Pode-se criar um exemplo de matriz de duas dimensões por extensão da anterior. Consideremos um hotel de oito andares, com dez quartos em cada andar. Existem quatro opções que se podem adotar para controlar os 80 possíveis clientes. Primeiro, cada quarto ter seu próprio nome de variável. Segundo, o hotel pode ter uma matriz de 80 elementos. Terceiro, cada andar pode ter sua própria matriz de 10 elementos. Quarto, o hotel pode ter uma matriz de duas dimensões. A opção final pode ser vista como segue:

H$(8,1)	H$(8,2)	H$(8,3)	H$(8,4)	H$(8,5)	H$(8,6)	H$(8,7)	H$(8,8)	H$(8,9)	H$(8,10)
H$(7,1)	H$(7,2)	H$(7,3)	H$(7,4)	H$(7,5)	H$(7,6)	H$(7,7)	H$(7,8)	H$(7,9)	H$(7,10)
H$(6,1)	H$(6,2)	H$(6,3)	H$(6,4)	H$(6,5)	H$(6,6)	H$(6,7)	H$(6,8)	H$(6,9)	H$(6,10)
H$(5,1)	H$(5,2)	H$(5,3)	H$(5,4)	H$(5,5)	H$(5,6)	H$(5,7)	H$(5,8)	H$(5,9)	H$(5,10)
H$(4,1)	H$(4,2)	H$(4,3)	H$(4,4)	H$(4,5)	H$(4,6)	H$(4,7)	H$(4,8)	H$(4,9)	H$(4,10)
H$(3,1)	H$(3,2)	H$(3,3)	H$(3,4)	H$(3,5)	H$(3,6)	H$(3,7)	H$(3,8)	H$(3,9)	H$(3,10)
H$(2,1)	H$(2,2)	H$(2,3)	H$(2,4)	H$(2,5)	H$(2,6)	H$(2,7)	H$(2,8)	H$(2,9)	H$(2,10)
H$(1,1)	H$(1,2)	H$(1,3)	H$(1,4)	H$(1,5)	H$(1,6)	H$(1,7)	H$(1,8)	H$(1,9)	H$(1,10)

Como se pode ver, o primeiro índice da matriz de duas dimensões indica o número do andar e o segundo índice é o número do quarto naquele andar. Assim, Q$(3,2) deve ser o nome do ocupante do segundo quarto no terceiro andar.

As matrizes do Applesoft podem ter até 88 dimensões. Não existe limite específico para o número de elementos em cada dimensão. O tamanho da memória disponível limita o número total de elementos, claro, desde que cada elemento necessita de um certo espaço de memória.

EXPRESSÕES

Na seção seguinte vamos explorar as formas pelas quais podem-se combinar os valores de variáveis e constantes usando *expressões.* Já usamos expressões para calcular o valor de problemas aritméticos simples em modo imediato. Lembre-se do comando

```
PRINT 4+6
10
```

que diz ao Apple II para somar 4 e 6, e mostrar a soma. O comando

```
PRINT A+B
0
```

diz ao Apple II para somar os valores de duas variáveis numéricas A e B, e mostrar a soma.

O sinal mais (+) especifica adição. Em "computês" esse sinal é um *operador.* O sinal mais é um *operador aritmético* porque especifica adição, que é uma operação aritmética.

Os operadores aritméticos são bastante simples de se entender; todos já aprendemos a somar, subtrair, multiplicar e dividir quando crianças. Porém, existem outros tipos de operadores: *operadores de cordão, operadores relacionais* e *operadores Booleanos.* Eles também são fáceis de se entender, mas requerem um pouco mais de explicação por usarem notação um pouco mais abstrata.

Cada categoria de operadores define um tipo de expressão. Existem expressões aritméticas, expressões de cordão, expressões relacionais e expressões Booleanas.

Precedência de Operadores em Expressões

As expressões podem provocar ocorrência de mais de uma operação. Por exemplo, o comando:

```
PRINT A+B/10
0
```

chama adição e divisão numa mesma expressão. Existe um esquema padrão para determinar em que ordem as expressões devem ser resolvidas. Vamos descrever essas regras de *precedência* para cada tipo de expressão que inicia com concatenação de séries, inteiros, reais, relacionais, Booleanos e expressões mistas, nessa ordem. Primeiro vamos ver uma forma de evitar as regras de precedência acima.

Contornando as Regras de Precedência

Pode-se trocar a ordem que o Apple II utiliza para resolver as expressões através do uso de parênteses. Qualquer operação dentro de parênteses é realizada primeiro. Quando mais de um conjunto de parênteses está presente, o Apple II calcula da esquerda para a direita.

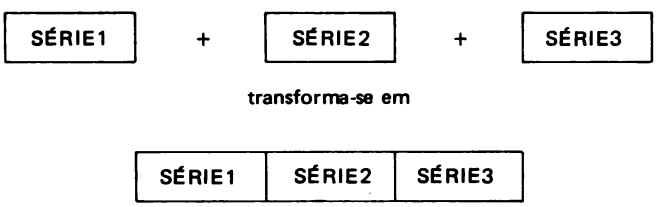
Quando um conjunto de parênteses está envolvido por outro, diz-se que ele está *aninhado.* Neste caso, o Apple II calcula primeiro o mais interno e depois os subseqüentes. Os parênteses podem ser aninhados em qualquer quantidade. Pode-se usá-los livremente para tornar clara a ordem de precedência dos cálculos na resolução de expressões.

Aqui estão alguns exemplos de cálculos aritméticos em modo imediato usando parênteses:

```
PRINT (2+10)*3
36
>
PRINT ((2+10)*3+31)*10
670
>
PRINT -(2^(3+8/4))
-32
```

Concatenação de Série de Caracteres

Pode-se juntar séries de caracteres uma no fim da outra para formar uma série maior. Isso se chama *concatenação.* Pode-se ver isso abaixo:



Com a concatenação, pode-se obter séries com até 255 caracteres de comprimento.

O Integer BASIC não possui operação de concatenação. Pode-se concatenar séries no Integer BASIC usando as técnicas mostradas no fim deste capítulo (mostrar agora seria prematuro).

O Applesoft usa o sinal mais (+) com operador de concatenação. Aqui estão alguns exemplos de concatenação no Applesoft:

"LOCO" + "MOTIVA"	torna-se	"LOCOMOTIVA"
"RELATÓRIO" + " " + "MENSAL"	torna-se	"RELATÓRIO MENSAL"
"CONTA" + A$	torna-se	"CONTA" seguido pelo valor de A$
A1$ + YA$ + C$(1)	torna-se	o valor de A1$ seguido pelo valor de YA$ seguido pelo valor de C$(1)

Expressões Inteiras

Expressões inteiras são expressões aritméticas que envolvem apenas variáveis e constantes inteiras. Mostraremos as expressões aritméticas que usam tanto valores inteiros quanto reais sob o título "Expressões de Tipo Misto".

Os operadores para expressões inteiras são adição (+), subtração (−), multiplicação (*), divisão (/) e exponenciação (∧). Pode-se também usar o sinal negativo (−) para indicar um valor numérico negativo. As operações são executadas na seguinte ordem: primeiro o negativo (−), depois a exponenciação, a multiplicação e a divisão, e por último a adição e a subtração. As operações com igual precedência são executadas na ordem da esquerda para a direita.

Aqui estão alguns exemplos de expressões inteiras no Integer BASIC:

100 − 30 * 2	resulta em	40
−9∧2	resulta em	81
A/B*C	resulta no	valor de A dividido pelo valor de B, e no valor inteiro da divisão multiplicado pelo valor de C
D + X*3	resulta em	3 vezes o valor de X somado ao valor de D
5/2*2	resulta em	4 (o quociente de 5/2 é convertido para o inteiro 2)

Aqui estão alguns exemplos de expressões inteiras no Applesoft:

−120/2 + 100	resulta em	40
2∧3*2	resulta em	16
N1%*N2%/N3%	resulta no	valor de N1% vezes o valor de N2% e no produto dividido pelo valor de N3%
AA%/AB%/AC%	resulta no	valor de AA% dividido pelo valor de AB% e no quociente dividido pelo valor de AC%
5/2*2	resulta em	5 (o quociente de 5/2 não é convertido para inteiro)

O Integer BASIC tem um operador a mais que pode ser usado em expressões inteiras. Ele retorna o resto de uma operação de divisão em que o dividendo não é divisível pelo divisor. O operador é o MOD, e tem a mesma precedência que a multiplicação e a divisão. Aqui estão alguns exemplos do MOD:

4 MOD 3	resulta em	1
3 * 5 MOD 4	resulta em	3
(41 + 2)/25 MOD A	resulta no	resto da divisão obtida dividindo 18 pelo valor de A
3 MOD 4	resulta em	3

Expressões Reais

O Applesoft tem outro tipo de expressões aritméticas; ele fornece números reais. Os seus operadores são os mesmos que os das operações inteiras do Applesoft: adição (+), subtração (−), multiplicação (*), divisão (/), exponenciação (∧) e negativo (−). A precedência dos operadores também é a mesma: primeiro negativo, depois exponenciação, multiplicação e divisão, e por fim adição e subtração. Aqui estão alguns exemplos de expressões reais:

87.5 − 4.25 * 2	resulta em	79
1.5 ∧ (3/2/2)	resulta em	1.35540301
AL * (PL − 3.1 * CB)	resulta no	valor de AL vezes o resultado da subtração do produto de 3.1 vezes o valor de CB do valor de PL
7.5 * 2/5	resulta em	3

Expressões Relacionais

Os operadores relacionais permitem que se compare dois valores para ver qual a relação entre um e outro. Pode-se comparar dois valores para ver se o primeiro é maior que, menor que, igual, diferente, maior ou igual ou menor ou igual que o segundo valor. Os valores comparados podem ser constantes, variáveis ou qualquer tipo de expressões. (Existem algumas restrições no Integer

TABELA 3.1. Operadores Relacionais.

Operadores do Integer BASIC	Operação	Operadores Applesoft
<	menor que*	<
>	maior que*	>
=	igual a	=
# ou <>	diferente	<> ou ><
>=	maior ou igual a*	>= ou =>
<=	menor ou igual a*	<= ou =<
* Não permitido com séries de caracteres em Integer BASIC.		

BASIC.) Se o valor de um lado do operador relacional é uma série, o valor do outro lado também deve ser tipo série. Fora isso, pode-se comparar um tipo de valor com outro usando operadores relacionais.

Se a relação é verdadeira, a expressão relacional tem o valor numérico de 1. Se ela é falsa a expressão relacional tem o valor numérico 0.

Os operadores relacionais para o Integer BASIC e o Applesoft são os mesmos, com exceções, como mostrado na Tabela 3.1.

Todos os operadores relacionais têm a mesma precedência; eles são calculados da esquerda para a direita.

Aqui estão alguns exemplos de expressões relacionais:

1 = 5 – 4	resulta em	1 (verdadeira)
14 > 66	resulta em	0 (falsa)
15 > = 15	resulta em	1 (verdadeira)
"AA" > "AA"	resulta em	0 (falsa)
"ABILIO" < "ANDRÉ"	resulta em	1 (verdadeira)
(A = B) = (A$ > B$)	depende	do valor das variáveis. Se o valor de A é igual ao valor de B e o valor de A$ é maior que o valor de B$, então o valor dessa expressão resulta em 1 (verdadeira)

O conceito que orienta os operadores relacionais é fácil de ser entendido. Os valores 0 e 1 que o BASIC determina arbitrariamente como condição falsa e verdadeira podem ser usados em expressões inteiras e reais. Isso não é tão fácil de se entender, porque é arbitrário. Por exemplo, qual o significado da expressão (1=1)*4? Fora de um programa em BASIC esta expressão não teria o menor sentido; mas dentro do BASIC é o mesmo que 1*4, o que resulta em 4. Pode-se incluir expressões relacionais dentro de outras expressões no BASIC. Aqui estão alguns exemplos:

25 + (14 > 66)	é o mesmo que	25 + 0
(A + (1 = 5 – 4)) * (15 > = 15)	é o mesmo que	(A + 1) * (1)

Comparação de Séries de Caracteres

Vamos aqui considerar quais as regras que o Apple II usa para comparar séries de texto. Existem duas considerações. A primeira é sobre o tamanho da série. Séries de diferentes tamanhos (lembre-se de que os espaços em branco ocupam espaços na série) não são iguais. Se uma série pequena é idêntica à primeira parte de uma maior, a série maior é tomada como a maior das duas (isso só se aplica ao Applesoft). A segunda consideração é sobre séries que têm o mesmo número de caracteres, na mesma ordem. No Integer BASIC, só se podem comparar duas séries com os operadores = e # ou < >. As séries são comparadas com um caractere por vez, iniciando com o colocado mais à esquerda – o primeiro caractere de uma série com o primeiro caractere da outra, o segundo com o da outra, o terceiro com o da outra e assim por diante até que as duas séries terminem ou ocorra uma diferença.

O Applesoft compara a ordem relativa dos caracteres um a um. Para propósitos de comparação, as letras do alfabeto têm a ordem A < B, B < C, C < D, 1 < 2, 2 < 3 etc. Outros caracteres

que aparecem na série, como +, −, $ etc. são colocados arbitrariamente na ordem mostrada no Apêndice I.

Expressões Booleanas

Os operadores Booleanos dão aos programas a habilidade de fazer decisões lógicas. Por isso eles são chamados também de *operadores lógicos.* Existem quatro operadores lógicos padrões: AND, OR, OR exclusivo e NOT. O BASIC do Apple II suporta três desses operadores: AND, OR e NOT.

Se existe dificuldade para se entender o conceito dos operadores Booleanos, uma analogia com compras em supermercado serve para ilustrar como são eles. Suponhamos que estamos num supermercado para comprar doces em companhia de duas crianças. O operador AND diz que você vai comprar o doce se as duas crianças quiserem. O operador OR diz que vamos comprar o doce se uma das crianças quiser. O operador NOT é o da contrariedade. Se a criança A insiste em discordar da criança B, sua decisão é o NOT dela.

Os computadores não operam com analogia; eles trabalham com números. Por isso a lógica Booleana reduz os valores das decisões a 1 ou 0 (verdadeiro ou falso). Como os operadores Booleanos trabalham com uns e zeros, eles são freqüentemente usados em expressões relacionais (lembre-se que as expressões relacionais sempre resultam em uns e zeros). Os operadores Booleanos podem trabalhar com outros tipos de operandos, como veremos rapidamente na próxima seção.

A Tabela 3.2 sumariza a forma pela qual as expressões Booleanas são calculadas. Essa tabela é referenciada como *tabela verdade.* Os operadores Booleanos têm igual precedência. Se aparece mais de um operador Booleano na mesma expressão, o cálculo é feito da esquerda para a direita. Aqui estão alguns exemplos de operadores Booleanos:

NOT ((3 + 4) > = 6)	resulta em	0 (falso)
("AA" = "AB") OR ((8*2) = 4^2	resulta em	1 (verdadeiro)
NOT ("APPLE" = "ORANGE") AND (A$ = B$)	resulta em	1 (verdadeiro) se A$ e B$ forem iguais e 0 (falso), se não forem.

TABELA 3.2. Tabela Verdade Booleana.

A operação AND resulta em 1 se ambos os valores forem 1.

1 AND 1 = 1	1 AND 0 = 0
0 AND 1 = 0	0 AND 0 = 0

A operação OR resulta em 1 se algum deles for 1.

1 OR 1 = 1	1 OR 0 = 1
0 OR 1 = 1	0 OR 0 = 0

A operação NOT complementa logicamente cada valor.

NOT 1 = 0
NOT 0 = 1

Expressões do Tipo Misto

Muitas vezes as expressões envolvem valores de mais de um tipo. Isso é especialmente freqüente no Applesoft, que trabalha com números inteiros e reais. Já foi colocada a idéia de expressões do tipo misto quando discutimos as expressões relacionais e Booleanas. Pode-se misturar os tipos de expressões de forma livre, à exceção de que as séries não podem fazer parte de expressões inteiras, reais ou Booleanas. As séries de caracteres só podem fazer parte de expressões delas mesmas e das relacionadas. Aqui estão alguns exemplos de expressões do tipo misto:

Legais	**Ilegais**
3.1416*(R∧2)	1600 + "AVENIDA SÃO JOÃO"
A% >= B/3	ST$ < A%
43 AND 137	A$ AND B$
1 OR 4E + 10	NOT (A$) = B$
(A$ = B$) AND −6.25	NOT (A = B) OR C$

TABELA 3.3. Operadores.

	Precedência	**Operador em Integer BASIC**	**Operador em Applesoft**	**Significado**
	9 (maior)	( )	( )	Parênteses dá a ordem de cálculo
Operadores Aritméticos	8	∧	∧	Exponenciação
	7	−	−	Negação
	6	*	*	Multiplicação
	6	/	/	Divisão
	6	MOD	não permite	Resto da divisão**
	5	+	+	Adição
	5	−	−	Subtração
Operadores Relacionais	4	=	=	Igual
	4	#	<> ou ><	Diferente
	4	<	<	Menor que
	4	>	>	Maior que
	4	<=	<= ou =<	Menor ou igual a
	4	>=	>= ou =>	Maior ou igual a
Operadores Lógicos	3	NOT	NOT	Complemento lógico
	2	AND	AND	E lógico
	1 (menor)	OR	OR	OU lógico

** Apenas no Integer BASIC.

O Apple II tem várias coisas a resolver quando está operando com operadores misturados. A primeira preocupação é com a precedência dos operadores. A Tabela 3.3 mostra os operadores de todos os tipos na ordem de precedência, da maior para a menor. Ela indica que tudo que estiver entre parênteses é calculado primeiro. Se existir mais de um nível de parênteses, o Apple II calcula o mais interno primeiro, depois o seguinte etc. (Relembre o conceito de parênteses *aninhados.*) Depois, são calculadas as expressões aritméticas. Finalmente as expressões relacionais e as expressões Booleanas.

Como foi visto antes, as expressões relacionais devolvem o valor 1 ou 0 dependendo se a relação testada for verdadeira ou falsa. Assim, uma expressão relacional pode ser parte de uma expressão inteira ou real.

Pode-se também misturar tipos em expressões Booleanas. Tudo, numa expressão Booleana, é convertido em 1 e 0, antes que a operação seja executada. Os valores numéricos são convertidos de acordo com a seguinte regra: se o valor é 0, ele fica 0; se é diferente de 0 é convertido para 1.

O BASIC não pode, automaticamente, converter séries para valores numéricos, de forma que as séries são ilegais em expressões inteiras, reais e Booleanas, exceto como parte de expressão relacional.

No Applesoft tanto valores inteiros quanto reais podem estar presentes em expressões reais inteiras e Booleanas.

Quando valores inteiros ocorrem em expressões reais, eles são convertidos temporariamente para real para o cálculo da expressão. O resultado final de tal expressão pode ser tanto um inteiro quanto um real, dependendo do contexto em que a expressão ocorra. O Applesoft converte o valor automaticamente de forma apropriada.

Os valores reais são convertidos para inteiros eliminando a parte fracionária e usando o menor número inteiro seguinte. Isso é chamado *truncar.* Por exemplo:

1.1	torna-se	1
1.9	torna-se	1
−1.1	torna-se	−2
−1.9	torna-se	−2

COMANDOS DO BASIC

Agora estamos prontos para descrever a parte do BASIC que especifica a operação que o comando executará. Especificam-se essas operações com instruções BASIC. É uma prática comum se usar os termos *comando* e *determinação* de forma idêntica e ambígua. Rigorosamente, um comando é uma instrução executável em modo imediato. A mesma instrução dentro do programa é uma determinação.

Cada instrução executa uma tarefa específica. Este capítulo introduz conceitos de programação, mostrando a forma como as determinações são usadas. Não as mostraremos em detalhes aqui. A descrição completa das determinações se encontra no Capítulo 8 que pode ser consultado

para se ter melhor idéia de como operam. Este capítulo apenas conduz ao entendimento de parte da capacidade das determinações.

Uma última observação antes de começarmos. Embora este capítulo introduza conceitos de programação, ele não divide programação em partes. Se for necessário ter mais conhecimentos sobre programação propriamente dita, consulte a lista de livros de iniciação constantes no Apêndice K.

COMENTÁRIOS

É interessante que toda a abordagem sobre as declarações do BASIC inicie com a descrição da única declaração BASIC que o computador ignora: o comentário. Se os primeiros três caracteres da declaração são REM, o computador ignora completamente a declaração. Então, por que existe tal declaração? A resposta é que o comentário torna o programa fácil de ser lido.

Se se escreve um programa pequeno com cinco ou dez declarações, provavelmente não existem maiores problemas quando, após algum tempo, se deseja lembrar o que aquele programa faz e como — a menos que o tempo decorrido seja muito grande. Porém se o programa escrito tem 100 ou 200 declarações, é muito provável que se esqueça algo muito importante sobre ele já na próxima vez que se vá usá-lo. Após escrever algumas dúzias de programas, é certo que não nos lembremos de cada um deles em detalhes. A solução para este problema de documentar programas é a inserção de comentários.

Os bons programadores têm seus programas repletos de comentários. Em todos os exemplos de programas deste capítulo incluiremos comentários para informar o que está acontecendo, para estimular a todos que os utilizem também.

Os comentários têm número de linha, como qualquer outra declaração. O número de linha da declaração de comentário pode ser usado como se usa qualquer outro número de linha.

DECLARAÇÃO DE ATRIBUIÇÃO

Declarações de atribuições fazem associar valores a variáveis. Encontram-se declarações de atribuições freqüentemente, em todos os tipos de programas BASIC.

```
 90 REM INICIALIZA A VARIAVEL X
100 LET X=3
```

Na declaração 100, à variável X é atribuído o valor 3; esta mesma declaração pode ser reescrita:

```
100 X=3
```

A palavra LET é opcional; normalmente é omitida.

Aqui está uma declaração de atribuição de variável do tipo série de caracteres:

```
215 A$="DOCE VIDA"
```

Atribui-se à variável série A$ as duas palavras "DOCE VIDA".

Seguem três declarações de atribuição que atribuem valores à variável tipo matriz R$( ), que foi encontrada antes quando descrevíamos as matrizes:

```
200 REM R$() E UMA LISTA DE CLIENTES DO HOTEL
210 R$(1)="JONES"
220 R$(2)="SMITH"
230 R$(3)="DOE"
```

Lembre-se de que podemos colocar mais de uma declaração por linha; assim as três declarações de R$ podem ser colocadas numa única linha, como segue:

```
200 REM R$() E UMA LISTA DE CLIENTES DO HOTEL
210 R$(1)="JONES"    :R$(2)="SMITH"        :R$(3)="DOE"
```

Lembre-se de que cada declaração na mesma linha deve ser separada da anterior por dois pontos (:).

As declarações de atribuição podem incluir quaisquer operadores aritméticos ou lógicos já descritos neste capítulo. Aqui estão alguns exemplos:

```
 90 REM ESTA E UMA FORMA BOBA DE ATRIBUIR UM VALOR
100 V=33+7/9
```

A declaração acima atribui o valor 33.77777778 para a variável real V; é equivalente a três declarações:

```
 90 REM X E Y SERAO USADAS MAIS TARDE
100 X=7
110 Y=9
120 V=33+X/Y
```

que poderiam ser escritas numa só linha:

```
100 X=7:Y=9:V=33+X/Y
```

Aqui está uma declaração de atribuição que executa operações Booleanas descritas anteriormente:

```
 90 REM EXEMPLOS DESCRITOS ANTERIORMENTE
100 A=NOT((3+4)>=6)
110 B=("AA"="AB") OR ((8*2)=(4^2))
```

O exemplo seguinte mostra como uma variável tipo série pode ter um valor atribuído usando a concatenação do Applesoft:

```
 90 REM R$(6) TERA O VALOR SEU JOAO
110 SEU$ = "SEU "
120 DONA$ = "DONA"
130 N$ = "JOAO"
200 R$(6) = SEU$ + N$
```

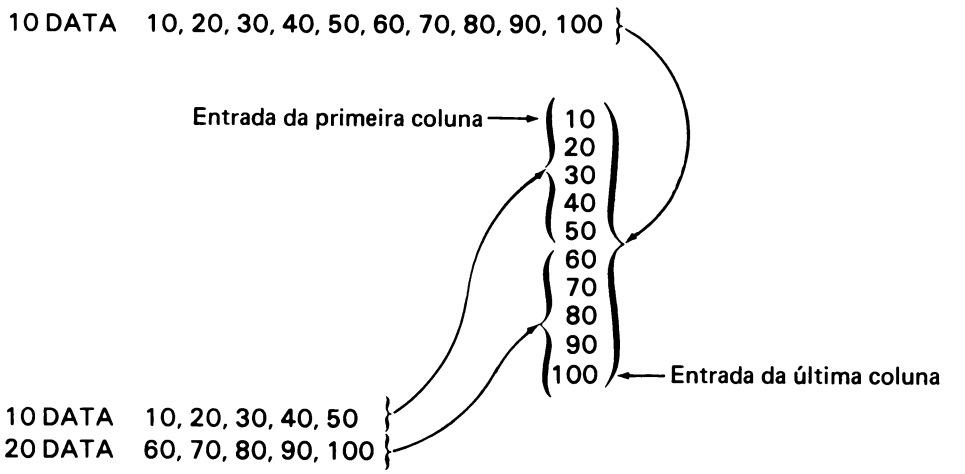
Declarações READ e DATA

Quando muitas variáveis precisam de atribuição num programa do Applesoft, pode-se usar as declarações READ e DATA ao invés do tipo de declaração de atribuição anterior. Veja os exemplos seguintes:

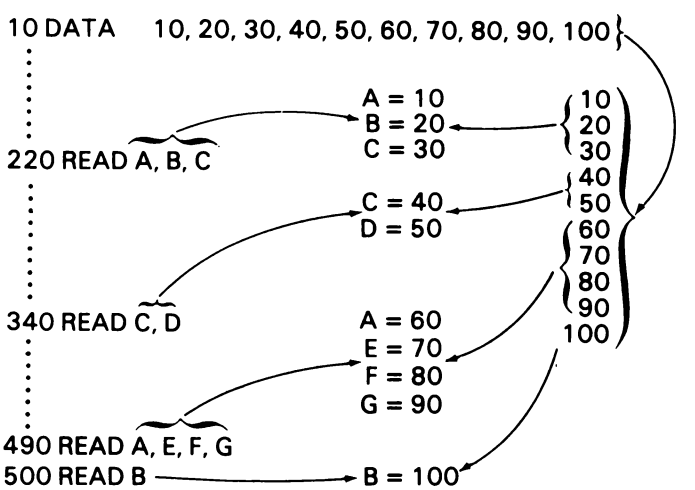
```
 5 REM INICIALIZA VARIAVEIS DO PROGRAMA
10 DATA 10,20,-4,300
20 READ A,B,C,D
```

A declaração da linha 10 especifica os valores numéricos das variáveis. Esses quatro valores são atribuídos para as variáveis pela declaração da linha 20. Depois que as declarações das linhas 10 e 20 forem executadas, A=10, B=20, C=−4 e D=300.

Se existe mais de uma declaração DATA no programa, pode-se visualizá-las como uma coluna de valores. Por exemplo, uma declaração DATA que contém uma lista de 10 valores constitui uma coluna de 10 entradas. Duas declarações DATA, cada uma contendo uma relação de 5 elementos, constitui exatamente a mesma coluna. Isso pode ser ilustrado como segue:



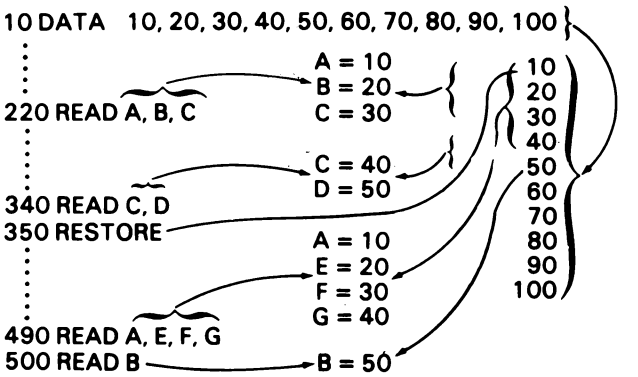
A primeira declaração READ no programa inicia na primeira coluna e pega os valores seqüencialmente, atribuindo-os às variáveis que aparecem na declaração READ. A segunda (e seguintes) declaração READ pega valores da coluna, iniciando no ponto onde o READ anterior parou. Isso pode ser ilustrado como segue:



A coluna DATA pode conter tanto valor numérico como série. Quando se associa um valor a uma variável usando a declaração READ, cada valor deve ser do mesmo tipo (série ou numérico) da variável à qual será atribuído.

Declaração RESTORE

Pode-se voltar o ponteiro ao início da coluna DATA executando a declaração RESTORE no Applesoft. Aqui está um exemplo de como o RESTORE opera:



Limpando Variáveis

Tanto o Integer BASIC quanto o Applesoft permitem colocar elementos numéricos e elementos de matrizes a zero, e toda a variável série e elementos da matriz nulos, tudo de uma só vez.

O comando CLR faz isso no Integer BASIC. Ele só pode ser usado em modo imediato. Aqui está um exemplo:

```
>X=37
>PRINT X
37

>CLR
>PRINT X
0
```

A declaração CLEAR faz isso no Applesoft. Ela também opera no DATA como faz o RESTORE. Aqui está um exemplo:

```
]10 REM INICIALIZA VARIAVEIS
]20 X=37
]30 A$="CAVALO DE ACO"
]40 PRINT A$
]50 CLEAR
]60 PRINT X
]RUN
CAVALO DE ACO
0
```

DECLARANDO O TAMANHO DE MATRIZES E SÉRIES

Se for necessário usar matrizes e séries no seu programa, devem ser declarados os tamanhos máximos (ou dimensões) numa declaração ou declarações DIM (chamada declaração de dimensionamento) no início do programa. Uma declaração de dimensionamento pode dar a dimensão de qualquer número de matrizes e séries desde que caiba numa linha padrão de programa.

Em Interger BASIC, isso é feito fornecendo o nome da matriz ou da variável série e a seguir o valor máximo, entre parênteses. São permitidas matrizes numéricas de apenas uma dimensão — não se pode usar matrizes de série e multidimensionadas. O exemplo que segue ilustra o dimensionamento de duas séries de 5 e 20 caracteres respectivamente, e uma matriz numérica com 13 elementos (0 até 12):

```
10 DIM S1$(5),S2$(25),NB(12)
```

O número que segue o nome da variável série na declaração DIM é o comprimento máximo que a série pode ter durante o programa. O número que segue a matriz numérica é o índice máximo que pode ser chamado dentro da matriz.

Na primeira vez que o Applesoft encontra uma matriz, verifica se ela foi dimensionada. Se não, ele automaticamente dimensiona uma matriz com índices que vão de 0 a 10 para cada dimensão usada. Assim, não é necessário dimensionar uma matriz, a menos que ela necessite de mais de 11 elementos em uma dimensão. O exemplo seguinte dimensiona a matriz de uma única dimensão R$. Ele também dimensiona uma matriz inteira de 21 elementos.

```
115 DIM R$(10),R%(20)
```

A lista de clientes do hotel do nosso exemplo anterior, com duas dimensões pode ser dimensionada assim:

```
115 DIM H$(8,10)
```

O número (ou números) que segue o nome da matriz na declaração DIM é igual ao maior valor de índice que acessa uma posição particular da matriz. Lembre-se de que os índices iniciam em 0. Assim R$(10) dimensiona a variável R$( ) para ter 11 elementos, não 10, porque os índices 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 são permitidos. H$(8,10), da mesma forma, especifica uma matriz de duas dimensões com 99 entradas, porque a primeira dimensão pode ter valores 0, 1, 2, 3 etc., enquanto o segundo pode ter os valores de 0 até 10.

Redimensionamento Matrizes

Uma vez dimensionada uma variável matriz, ela não pode ser redimensionada sem que o programa volte a ser rodado desde o início. As referências subseqüentes não podem usar índices maiores do que o declarado; cada índice deve ser um número compreendido entre 0 e o número usado no dimensionamento.

COMANDOS DE SALTO

As declarações dentro de um programa BASIC são executadas dentro de uma ordem ascendente de números de linhas. A seqüência de execução foi explicada anteriormente neste capítulo, quando foram descritos os números de linha. As declarações de salto alteram a seqüência de execução.

Declaração GOTO

O GOTO é uma declaração simples; ela permite especificar a declaração que será executada a seguir. Considere o seguinte exemplo:

```
20   A = 4
30   GOTO 100
40
50
60
70
80
90
100
110
etc.
```

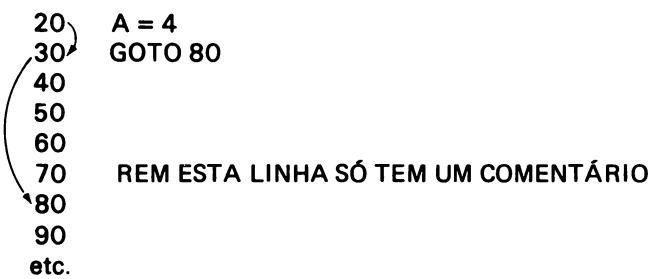
A declaração da linha 20 é de atribuição; ela atribui um valor à variável A. A próxima declaração é um GOTO; ela especifica que a execução do programa deve saltar para a linha 100. Assim a execução do programa seguirá a seqüência: linha 20, linha 30, e então linha 100.

Claramente, alguma outra declaração deve fazer o programa voltar para a linha 40, de outra forma a declaração dessa linha nunca seria executada, como mostrado acima.

Pode-se saltar para qualquer linha, mesmo que ela não tenha nada além de um comentário. Entretanto, o computador ignora o comentário e tudo se passa como se o salto tivesse sido para a linha seguinte. Por exemplo, veja o seguinte salto:

```
20  A = 4
30  GOTO 70
40
50
60
70   REM ESTA LINHA SÓ TEM COMENTÁRIO
80
90
etc.
```

A execução do programa pula para a linha 70; lá não existe nada além de um comentário, assim o computador vai para a linha 80; executando a declaração lá contida. Então, mesmo que se possa saltar para um comentário, é como pular para a linha seguinte. Isso pode ser ilustrado como segue:

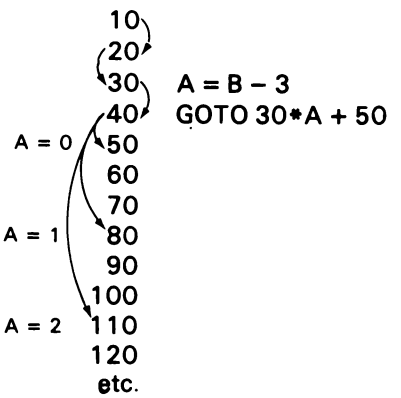


Tentar saltar para uma linha que não existe causa uma mensagem de erro.

Declaração GOTO Calculada

Existe mais um tipo de declaração GOTO que permite ao programa saltar logicamente para uma, duas ou mais linhas diferentes, dependendo do valor atual de uma expressão numérica.

Veja a seqüência de declarações do Integer BASIC:



A declaração da linha 40 é um GOTO *calculado.* Quando esta declaração é executada, a lógica do programa vai saltar para o número de declaração obtido calculando a expressão. Neste exemplo, pula para a declaração 50 se a variável A=0, para a declaração 80 se a variável A=1, enquanto que A=3 causa salto para a declaração 110. Se a linha calculada não existe o computador manda a mensagem *** BAD BRANCH ERR. Note que se atribui o valor à variável A na declaração 30. O valor de A depende do valor atual de B. O exemplo não mostra como B é obtido; entretanto, se B tiver os valores 3, 4 ou 5, a declaração da linha 40 causará um salto.

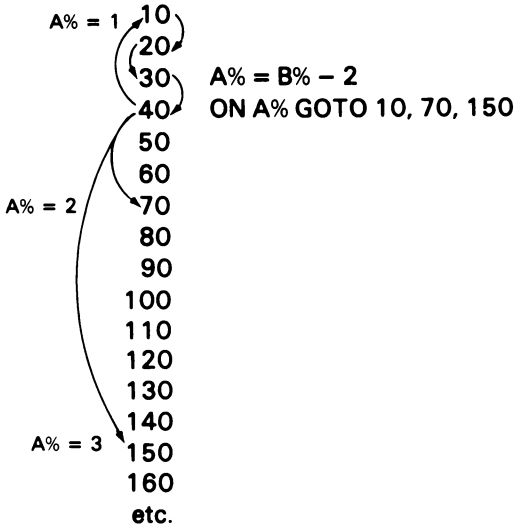
Para testar a declaração GOTO calculada no Integer BASIC, digite o seguinte programa:

```
>9    REM INICIALIZA VARIAVEL B
>10   B=4
>20   PRINT B
>30   A=B-3
>40   GOTO 30*A+50
>49   REM B=3
>50   END
>79   REM B=4
>80   PRINT B
>90   B=5
>100  GOTO 20
>109  REM B=5
>110  PRINT B
>120  B=3
>130  GOTO 20
```

Agora execute esse programa digitando RUN.

Pode-se contar a seqüência na qual os dígitos aparecem? Tente reescrever o programa de forma que cada número seja mostrado uma vez, na seqüência: 345345345...

O Applesoft tem uma versão um pouco diferente do comando GOTO calculado, como se vê abaixo:



A declaração na linha 40 é a forma de GOTO calculada do Applesoft. Quando esta declaração é executada, a lógica do programa tende para a declaração 10 se a variável A%=1, para a declaração 70 se A%=2, e A%=3 causa um salto para a declaração 110. Se A% tiver valores diferentes de 1, 2 ou 3, o programa continua na linha 50.

A expressão numa declaração GOTO calculado dentro do Applesoft é avaliada e seu valor determina para qual linha pular a partir da lista de números de linhas da declaração GOTO calculado. Se o valor é 1, usa-se o primeiro número de linha, se o valor for 2 usa-se o segundo número e assim por diante. Se o valor for 0 ou exceder a quantidade de números de linhas de lista, o programa vai para a declaração seguinte à declaração GOTO calculado.

O seguinte programa do Applesoft mostra como a declaração GOTO calculado opera:

```
10 B% = 4
20  PRINT B%
30 A% = B% - 2
40  ON A% GOTO 10,70,150
70  PRINT B%
80 B% = 5
90  GOTO 30
150  PRINT B%
160 B% = 3
170  GOTO 20
```

LOOPS

As declarações GOTO e GOTO calculado permitem criar qualquer seqüência de declarações que a lógica do programa precise. Mas suponha que se necessite executar uma instrução (ou grupo de

instruções) várias vezes. Por exemplo, suponha uma variável matriz A(I) com 100 elementos e que cada elemento necessite ter atribuição com valores entre 0 e 99. Escrever 100 declarações de atribuição seria tedioso. É mais fácil reexecutar uma mesma declaração de atribuição 100 vezes num loop.

Declarações FOR e NEXT

Pode-se criar um loop usando as declarações FOR e NEXT como segue:

```
10 DIM A(99)
20 FOR I= 0 TO 99 STEP 1
30 A(I)=I
40 NEXT I
```

A declaração ou declarações entre o FOR e o NEXT são executadas repetidamente. Neste caso uma declaração simples de atribuição aparece entre o FOR e o NEXT; essa declaração é executada repetidamente. Este tipo de estrutura de programa é chamado de *loop FOR-NEXT.*

Pode-se ver no exemplo abaixo um loop FOR-NEXT dentro do programa imprimindo os valores atribuídos à matriz A( ) dentro do loop.

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

Quando se digita RUN o programa mostra 100 números indo de 0 até 99.

As declarações entre o FOR e o NEXT são reexecutadas o número de vezes especificado pela *variável índice* que aparece diretamente após o FOR; no exemplo acima a variável índice é I. O I vai de 0 até 99 em passo (*step*) 1.

A variável I também aparece na declaração de atribuição em 30. Dessa forma, na primeira vez que a declaração de atribuição é executada, I é igual a 0 e a declaração é executada como:

```
30 A(0)=0
```

I é incrementada no tamanho do passo, que está especificado na linha 20, como 1; o I assim será 1 na segunda vez que a declaração da linha 30 for executada. A declaração de atribuição nessa segunda vez será equivalente a:

```
30 A(1)=1
```

A variável I continua a ser incrementada pelo passo até que o valor máximo (99) seja obtido (ou ultrapassado).

O passo não tem que ser 1; ele pode ser qualquer número inteiro. Troque o passo na linha 20 e reexecute o programa. Agora a declaração de atribuição é executada 20 vezes, desde que incrementando I 19 vezes em 5 chega a 95; a vigésima vez leva a 100, que é maior que 99. Mantendo o passo em 5, pode-se permitir que a declaração de atribuição seja executada 100 vezes aumentando o valor máximo de I para 500. Pode-se fazer essa modificação? (Lembre-se de modificar também o DIM.)

O tamanho do passo não precisa ser positivo. Mas se for negativo, o valor inicial de I deve ser maior que o valor final. Por exemplo, se o passo é −1 e desejamos inicializar 100 elementos de A(I) com valores de 0 a 99, pode-se reescrever a declaração na linha 20 como segue:

```
10 DIM A(99)
20 FOR I=99 TO 0 STEP -1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

Execute este programa para testar o passo negativo.

Se o passo é 1 (esse é o caso mais freqüente), não é necessário explicitar a definição do passo. Na ausência de definições, o BASIC assume que o passo é 1.

Pode-se especificar os valores inicial e final do índice e o passo usando-se as expressões que se desejar. Porém, deve-se fazer isso, para não complicar o programa desnecessariamente. Se for necessário calcular um desses valores, será mais interessante fazer isso separadamente antes de iniciar o loop.

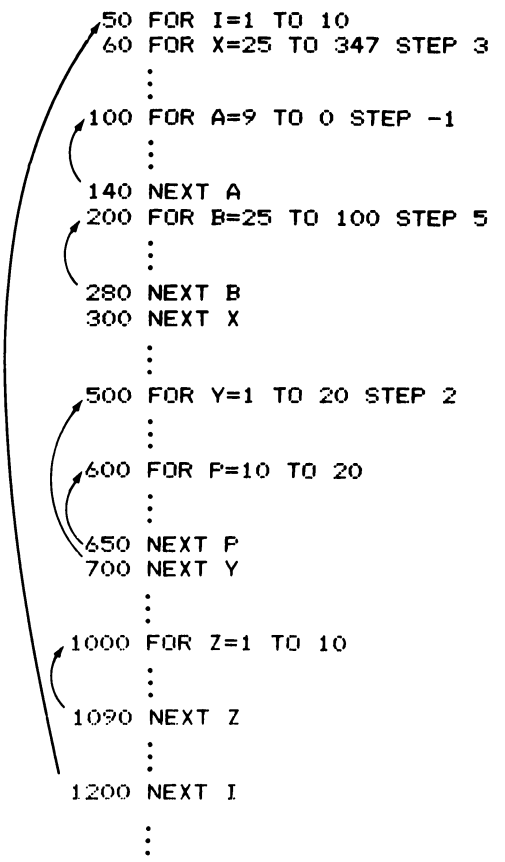
Pode-se usar valores reais para os valores inicial e final do índice e para o passo no Applesoft. Não é necessário explicitar o índice na declaração NEXT dentro do Applesoft. Porém, se isso for feito, tornará o programa mais fácil de ser entendido.

Loops Aninhados

A estrutura FOR-NEXT é chamada de *loop do programa*, porque a execução do programa vai do FOR ao NEXT, volta ao FOR, e fica em círculos. Esse tipo de estrutura é muito comum; quase todos os programas que se escrevem usando BASIC têm uma ou mais delas. Os loops são tão comuns que muitas vezes eles compõem estruturas *aninhadas*, uma dentro da outra, tal como são armazenados jogos de panelas, uma dentro da outra. Pode haver qualquer número de declarações entre um FOR e um NEXT. Freqüentemente existem dezenas e até centenas de declarações. E dentro dessas dezenas ou centenas, outros loops podem ocorrer. O exemplo abaixo mostra um nível simples de aninhamento:

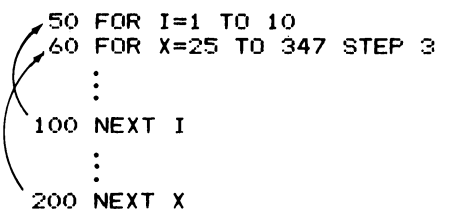
```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 REM MOSTRA TODOS OS VALORES DE A(I) JA DADOS
50 FOR J=0 TO I
60 PRINT A(J)
70 NEXT J
80 NEXT I
90 END
```

Freqüentemente aparecem estruturas complexas de loop, mesmo em programas relativamente pequenos. Aqui está um exemplo, mostrando as declarações FOR e NEXT, mas sem as declarações intermediárias:



O loop mais externo usa a variável índice I; ele contém três loops aninhados que usam os índices X, Y e Z. O loop X contém dois loops adicionais usando índice P. O loop Z não contém loops aninhados.

As estruturas de loop são muito fáceis de visualizar e usar. Existe apenas um erro muito comum que deve ser evitado; não terminar um loop externo antes de terminar um interno. Por exemplo, a seguinte estrutura de loop é ilegal:



Todo o programa deve ter algumas declarações FOR e NEXT, desde que todo loop inicia com a declaração FOR e termina com NEXT.

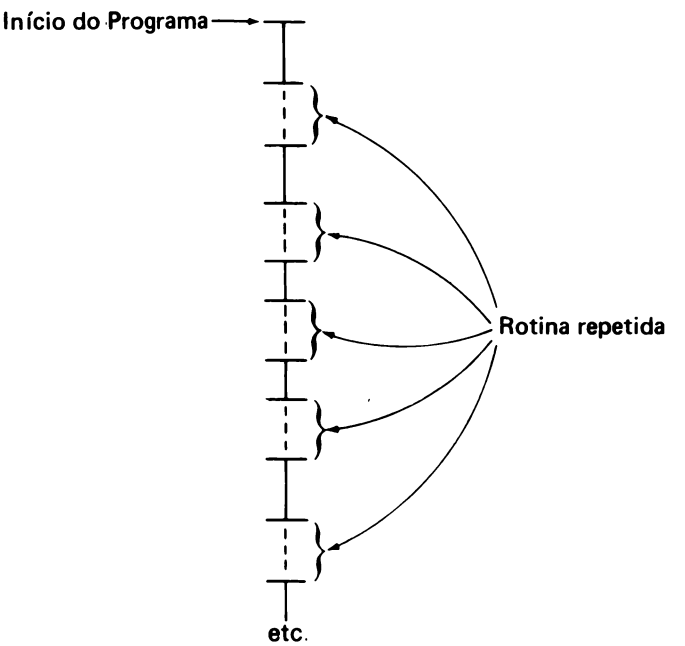
Por exemplo, suponha que exista uma declaração FOR e duas NEXT. O primeiro NEXT termina o FOR de forma que o loop opere corretamente. Porém o segundo NEXT não completa nenhum FOR, o que causa um erro.

Se não se inclui a variável índice na declaração NEXT em um programa no Applesoft, a lógica do programa automaticamente termina os loops corretamente, uma vez que só existe uma possibilidade de finalizar um loop a cada vez que um NEXT é encontrado. Caso haja dúvidas sobre isso, veja o exemplo complexo mostrado na página anterior. Faça estudos adicionais envolvendo estruturas mais complexas.

DECLARAÇÕES DE SUB-ROTINAS

Quando se começa a criar programas de tamanho maior, logo se percebe que existem pedaços de programa que são necessários mais de uma vez dentro da estrutura. Por exemplo, suponhamos uma variável matriz (A( )) que deve ser reinicializada em diferentes pontos do programa. Pode-se repetir as três instruções que compõem o loop várias vezes, porque são poucas instruções.

Suponhamos agora que o loop tenha dez ou doze instruções que processam dados de alguma forma antes de fazer a inicialização. Se tal loop precisar ser usado dentro do programa em vários pontos, redigitar todas as instruções toda vez que elas são necessárias pode exigir muito tempo, e mais importante, gastar memória. Este conceito é ilustrado no exemplo abaixo:

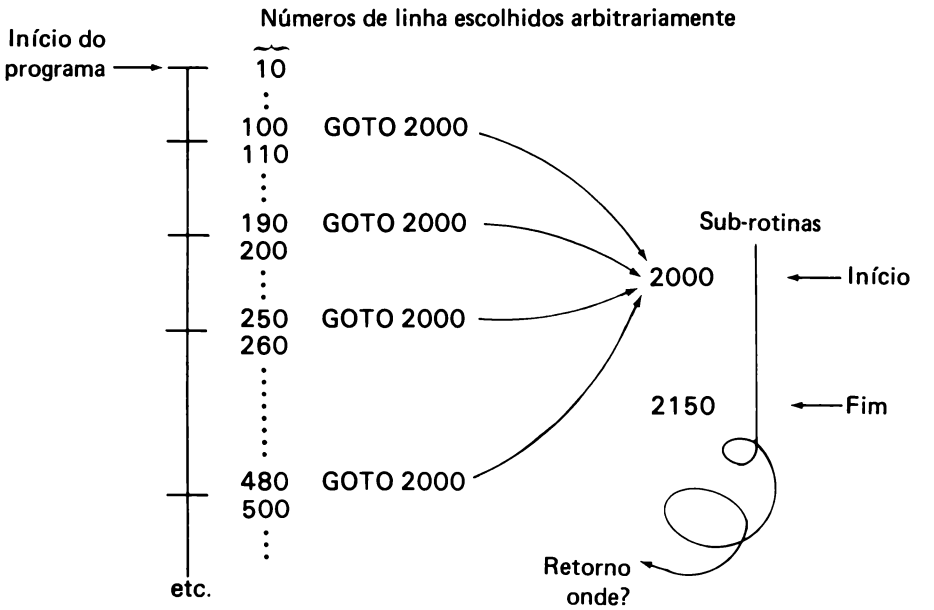


Que tal separarmos as determinações que se repetem e usá-las em separado?

Isso é precisamente o que é feito; o grupo de declarações em separado que é acessado dessa forma chama-se *sub-rotina*.

Mas aparece um problema. Pular do programa para a sub-rotina é fácil; ela tem um endereço inicial específico. Mas, no fim da sub-rotina, como se pode voltar?

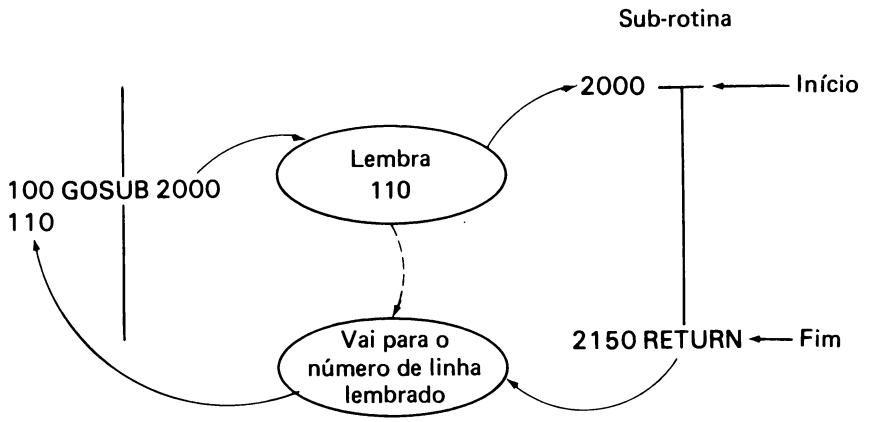
Pode-se executar um GOTO em qualquer ponto em que se deseja ir para a sub-rotina. Isso é ilustrado como segue:



Mas, no final da sub-rotina, como voltar? Se duas ou mais declarações GOTO vão para a mesma sub-rotina, existem dois ou mais lugares para voltar após concluí-la. A solução é se usar declarações especiais. No lugar de pular para a sub-rotina usando GOTO, usa-se uma declaração GOSUB.

A Declaração GOSUB

Esta declaração salta da mesma forma que o GOTO, porém além disso ela se lembra para onde deve voltar. Em "computês" dizemos que GOSUB *chama* uma sub-rotina. Isso pode ser ilustrado como segue:



A sub-rotina finaliza com uma declaração RETURN. Esta declaração faz um salto de volta para a declaração que vem depois do GOSUB. Se o GOSUB era a última declaração de uma linha, o programa retorna para a primeira declaração da linha seguinte.

O loop de três declarações que inicializa a matriz A( ), se convertido para sub-rotina fica assim:

```
  10 REM PROGRAMA  PRINCIPAL
  20 REM PODE-SE DIMENSIONAR AS VARIAVEIS
  30 REM DA SUBROTINA NO PROG.PRINCIPAL
  40 REM E UMA BOA IDEIA DIMENSIONAR TODAS
  50 REM AS VARIAVEIS NO INICIO DO PROGRAMA
  60 DIM A(99)
  70 GOSUB 2000
  80 REM MOSTRA ALGO PARA MOSTRAR FIM CERTO
  90 PRINT "VOLTOU"
 100 END
2000 REM SUBROTINA
2010 FOR I=1 TO 99
2020 A(I)=I
2030 PRINT A(I)
2040 NEXT I
2050 RETURN
```

Declaração POP

Sob certas circunstâncias, pode-se não desejar voltar de uma sub-rotina para o ponto de onde ela foi chamada. Nesse caso se poderia tentar usar um GOTO para retornar, porém isso pode causar problemas, porque o BASIC retém na memória o endereço em que deveria voltar. Nesses casos deve-se usar a declaração POP. De outra forma corre-se o risco de se obter um erro devido à acumulação de endereços de retorno não usados por declarações RETURN. Tudo o que o POP faz é mandar o BASIC esquecer o último endereço de retorno guardado. Assim pode-se usar um GOTO para ir para qualquer outro lugar no programa.

Contornar o RETURN constantemente, usando POP's em demasia para liberar saídas de sub-rotina através de GOTO não é uma prática muito aconselhável, uma vez que torna o programa bastante confuso.

Sub-rotinas Aninhadas

As sub-rotinas podem ser aninhadas. Quer dizer, uma sub-rotina pode, por sua vez, chamar outra sub-rotina etc. Não é necessário dar instruções especiais para que isso aconteça. Simplesmente é preciso dar-se a declaração GOSUB e finalizar a sub-rotina com RETURN. O BASIC se lembrará do número da linha correto para cada sub-rotina aninhada.

O programa seguinte ilustra sub-rotinas aninhadas:

```
 10 REM PROGRAMA PRINCIPAL
 20 REM PODE-SE DIMENSIONAR AS VARIAVEIS
 30 REM DA SUBROTINA NO PROG.PRINCIPAL
 40 REM E UMA BOA IDEIA DIMENSIONAR TODAS
 50 REM AS VARIAVEIS NO INICIO DO
 60 REM PROGRAMA PRINCIPAL
 70 DIM A(99)
 80 GOSUB 2000
 90 REM MOSTRA ALGO PARA INFORMAR RETORNO CORRETO
100 PRINT "ACABOU"
110 END
2000 REM SUBROTINA DE PRIMEIRO NIVEL
2010 FOR I=0 TO 99
2020 A(I)=I
2030 GOSUB 3000
2040 NEXT I
2050 RETURN
3000 REM SUBROTINA ANINHADA

3010 PRINT A(I)
3020 RETURN
```

Este programa move a declaração PRINT A(I) para fora da sub-rotina da linha 2000 e a coloca numa sub-rotina aninhada, na linha 3000. Nada mais mudou.

Enquanto seja aceitável e até desejável que uma sub-rotina possa chamar outra, ela não pode chamar a si própria. Nem uma sub-rotina pode chamar outra que por sua vez chame a primeira. Isso é chamado *recursividade* e não é permitida dentro do BASIC no Apple II.

Declaração GOSUB Calculada

As lógicas do GOTO e do GOSUB são muito semelhantes. Elas só diferem no fato de que o GOSUB guarda o endereço de retorno para voltar. Assim, não deve causar surpresa o fato de existir também uma declaração GOSUB que opere de forma similar ao GOTO calculado. O GOSUB calculado permite que se vá para uma entre duas ou mais sub-rotinas dependendo do valor de uma expressão numérica. O GOSUB calculado também se lembra para onde deve voltar. Não importa para qual sub-rotina ele vá. Ao final da sub-rotina, chegando na declaração RETURN, haverá o retorno para o ponto inicial de chamada.

Pode-se usar sub-rotinas aninhadas usando declarações GOSUB, da mesma maneira que se pode usar sub-rotinas aninhadas usando declarações GOSUB normais.

Vejamos a seguinte declaração do integer BASIC:

```
>100 GOSUB A*500+2000
>110 REM
```

A expressão na linha 100 é um GOSUB calculado. Quando essa declaração é executada, a lógica do programa pula para a linha cujo número é obtido calculando-se a expressão. Nesse

exemplo, pula-se para a linha número 2000 se A=0, para a linha 2500 se A=1 e assim por diante. Se a linha cujo número foi calculado não existe, aparece a mensagem *** BAD BRANCH ERR.

A versão Applesoft do GOSUB calculado opera de maneira semelhante à sua versão do GOTO calculado. Aqui está um exemplo:

```
]90
]100 ON A GOSUB 1000,500,5000,2300
]110 REM
```

Quando a declaração da linha 100 é executada, se A=1 a sub-rotina que inicia na linha 1000 é chamada. Se A=2, é chamada a sub-rotina que inicia na linha 500. Se A=3, é chamada a sub-rotina que inicia na linha 500 e se A=4, é chamada a sub-rotina que inicia na linha 2300. Qualquer outro valor de A diferente de 1, 2, 3 e 4 faz o programa ir para a linha 110, sem desviar para nenhuma sub-rotina.

EXECUÇÃO CONDICIONAL

As declarações GOTO e GOSUB condicionais são declarações condicionais. Isto é, a seqüência de execução do programa depende dos valores de uma ou mais variáveis que podem mudar durante a execução do mesmo. O fluxo do programa depende da condição de variáveis.

Declaração IF-THEN

Outra declaração condicional é a declaração IF-THEN. Ela tem a forma geral:

IF *expressão* THEN *declaração*

Se a *expressão* é verdadeira, então a *declaração* é executada. As expressões relacionais e Booleanas são muito comuns com declarações IF-THEN, embora expressões aritméticas possam ser usadas da mesma forma. Isso dá ao programa BASIC um poder real para tomar decisões. Aqui estão três exemplos simples de declarações IF-THEN:

```
10 IF A=B+5 THEN PRINT MSG$
40 IF CC$="M" THEN IN=0
50 IF Q<14 AND M<M1 THEN GOTO 66
```

A declaração na linha 10 causa a execução da declaração PRINT se o valor da variável A for igual ao valor da variável B mais 5. Em outro caso a declaração PRINT não é executada.

A declaração da linha 40 atribui o valor numérico 0 à variável IN se a variável série CC$ for a letra M.

A declaração da linha 50 causa o desvio do programa para a linha 66 se a variável Q for menor que 14 e a variável M for menor que M1. De outra forma, a execução do programa continua na linha seguinte.

Se houver dificuldade de entender a expressão que segue o IF, volte à discussão de expressões dadas antes neste capítulo.

Uma declaração IF-THEN pode ser seguida por outras declarações na mesma linha de programa. Qualquer declaração posterior na mesma linha de programa é sempre executada, não impor-

tando se a expressão do IF-THEN é verdadeira ou falsa. Podemos ver isso no exemplo:

```
10 IF V>1000 THEN PRINT "COMPLETO":GOSUB 2000
20 T=T+V:PRINT T
```

No exemplo acima, o programa imprime a expressão COMPLETO apenas quando a variável V é maior que 100. O programa chamará a sub-rotina na linha 2000, não importa qual seja o valor de V.

O Applesoft executa a declaração que segue o IF-THEN na mesma linha apenas quando a declaração for verdadeira. Se a expressão é falsa, a execução do programa pula para baixo para a próxima declaração da linha seguinte. No exemplo acima, o programa imprime a expressão COMPLETO e vai para a sub-rotina na linha 2000 apenas se V for maior que 100. Se V for menor ou igual a 100, o programa não imprimirá a mensagem nem irá para a sub-rotina, indo diretamente para a declaração da linha 20.

Existe uma forma especial da declaração IF-THEN no Applesoft. Sempre que a atitude a ser tomada condicionalmente for uma declaração GOTO, pode-se omitir a palavra THEN se desejado. As duas declarações seguintes são equivalentes:

```
110 IF MM$=DD$ THEN GOTO 100
```

é o mesmo que:

```
110 IF MM$=DD$ GOTO 100
```

DECLARAÇÕES DE ENTRADA E SAÍDA

Existe uma variedade de declarações no BASIC que controla a transferência de dados de e para o computador. A coleção delas é chamada de *declarações de entrada/saída.* As declarações de entrada/saída mais simples controlam a entrada de dados pelo teclado e a saída para a tela. Discutiremos essas declarações nos parágrafos que seguem. Porém, existem declarações de entrada/saída mais complexas que controlam a transferência de dados entre o computador e dispositivos periféricos, tal como o gravador cassette, o drive de disco e a impressora. Essas declarações mais complexas são tratadas nos Capítulos 4 e 5. O Capítulo 6 trata de declarações de saída na tela para gráficos.

Já vimos a declaração PRINT, que faz sair dados para a tela. Então, vamos ver esta declaração primeiro, antes das declarações de entrada.

Declaração PRINT

Por que usar PRINT no lugar de DISPLAY ou outra abreviação da palavra display? A resposta vem dos anos sessenta, quando a linguagem BASIC foi criada, os vídeos eram muito caros e em geral inacessíveis aos computadores de médio e baixo custo. Os terminais mais comuns para computadores tinham um teclado e uma impressora. A informação em geral era impressa ao invés de mostrada num vídeo, tal como nos dias de hoje. Assim a palavra PRINT, "imprima" em inglês, descreve a declaração que faz mostrar informação na tela do vídeo.

A declaração PRINT mostra texto e números. Por exemplo, a declaração seguinte vai mostrar na tela a palavra TEXTO:

```
10 PRINT "TEXTO"
```

Para mostrar números, coloca-se um número, ou o nome de uma variável após a palavra PRINT, como aqui:

```
>A=10
>PRINT 5,A
5          10
```

A declaração acima mostra o número e o número constante na variável A, que é 10, na mesma linha.

Pode-se mostrar uma mistura de textos e/ou números, listando a informação a ser mostrada após a declaração PRINT. Usa-se vírgulas para separar cada item. A declaração PRINT a seguir mostra as palavras UM, DOIS, TRÊS, QUATRO e CINCO, seguidas do numeral que representa cada número:

```
10 PRINT "UM",1,"DOIS",2,"TRES",3,"QUATRO",4,"CINCO",5
20 END
```

Pode-se separar as variáveis com vírgulas, como feito acima, de forma que o Apple II automaticamente aloca um número fixo de espaços para cada variável mostrada. Tente executar a declaração acima em modo imediato para ver por si próprio. Se for desejado eliminar os espaços em branco, basta separar as variáveis com o sinal ponto-e-vírgula (;), como abaixo:

```
10 PRINT "UM";1;"DOIS";2;"TRES";3;"QUATRO";4;"CINCO";5
20 END
```

Tente também esta variação para ver como o ponto-e-vírgula funciona.

Uma declaração PRINT automaticamente retorna o cursor para a margem esquerda e desce uma linha depois da última ação. Em "computês" diz-se que foi dado um *carriage return* (retorno do carro). Pode-se suprimir o retorno do carro colocando uma vírgula ou ponto-e-vírgula após o último item da lista do PRINT. Uma vírgula nessa posição move o cursor para a próxima localização onde um valor seria mostrado, caso existisse. Para ver isso, digite o programa seguinte e execute-o digitando RUN:

```
10 PRINT "UM",1
20 PRINT "DOIS",2
30 END
```

Agora coloque uma vírgula ao final da declaração na linha 10 e reexecute o programa com RUN. Vê-se então a concatenação da linha 20 com a linha 10 na mesma linha da tela.

Em seguida substitua a vírgula da linha 10 pelo ponto-e-vírgula e rode o programa com RUN. As duas linhas são concatenadas de novo, porém o espaço entre o numeral 1 e a palavra DOIS foi removido. Trocando outras vírgulas por pontos-e-vírgulas pode-se ir retirando os espaços em branco existentes entre os itens.

Mostramos os numerais colocando-os diretamente na declaração PRINT. Pode-se além disso mostrar numerais como conteúdo de variáveis. O programa seguinte fará o mesmo que o primeiro exemplo da declaração PRINT, porém usa o arranjo A( ) para criar os dígitos. Tente colocar o programa e rodá-lo.

```
 5 DIM A(5)
10 FOR I=1 TO 5
20 A(I)=I
30 NEXT I
40 PRINT "UM";A(1);"DOIS";A(2);"TRES";A(3);"QUATRO";A(4);
 "CINCO";A(5)
50 END
```

No Applesoft, podem-se colocar as palavras a serem mostradas numa variável série e colocar a declaração PRINT dentro de um loop FOR-NEXT, alterando o programa como mostrado abaixo:

```
10 DATA "UM","DOIS","TRES","QUATRO","CINCO"
20 FOR I=1 TO 5
40 READ N$
50 PRINT N$;I;
60 NEXT I
70 END
```

Declaração INPUT

Quando uma declaração é executada, o computador espera por entrada de informação pelo teclado; enquanto não for colocada a informação desejada, tudo fica parado.

Na sua forma mais simples, a declaração INPUT inicia com a palavra INPUT seguida do nome da variável. O dado entrado pelo teclado é atribuído àquela variável. O tipo do nome da variável determina o tipo de dado que deve ser colocado. Um nome de variável numérica é satisfeito com uma variável numérica. Para mostrar a entrada numérica, digite o programa seguinte e execute-o (tente colocar letras e veja o que acontece):

```
10 INPUT A
20 PRINT A
25 REM FIM DE PROGRAMA SE ACHAR 0
30 IF A=0 THEN END
40 GOTO 10
```

Na execução da declaração INPUT, o computador mostra um ponto de interrogação, e espera a entrada do dado. No programa acima, aparece cada tecla tal como foi digitada. Em "computês" a tela *ecoa* o teclado. Além disso, o número é mostrado duas vezes devido à declaração PRINT na linha 20. A primeira aparição do número ocorre na execução da declaração da linha 10, quando se faz uma digitação pelo teclado. A segunda aparição ocorre em respostas à declaração da linha 20, o PRINT.

A declaração INPUT pode entrar mais de uma variável de uma só vez. Para isso, liste todas as variáveis que deseja entrar depois da palavra INPUT. Separe as variáveis com vírgulas. Quando esse INPUT é executado, deve-se responder com um valor para cada variável. Certifique-se que o valor digitado seja do mesmo tipo da variável associada a ele.

Quando responder a uma declaração INPUT, não utilize vírgulas para pontuar números grandes; entre 1000 e *não* 1.000.

O exemplo seguinte entra dois valores numéricos mostrando em seguida o que foi entrado:

```
20 INPUT A,B
30 PRINT A,B
35 REM FIM DE EXECUCAO QUANDO ENTRA 0
40 IF A=0 OR B=0 THEN END
50 GOTO 20
```

Rode o programa acima e tente entrar um número seguido por vírgula, depois outro número, e em seguida a tecla RETURN. Agora tente algo um pouco diferente. Entre um número e digite a tecla RETURN. Como se pode ver, o Apple II informa que se deve digitar o outro número. Então coloque o outro número e acione RETURN. Vemos assim que quando a declaração INPUT pede mais de um valor numérico, existe a escolha de se digitar tudo dentro de uma mesma linha ou em linhas separadas.

A declaração INPUT opera de uma forma um pouco diferente quando se trata de variáveis tipo série no Integer BASIC. Antes de tudo, ele não coloca o ponto de interrogação; tente o exemplo:

```
10 DIM A$(19)
20 INPUT A$
30 PRINT A$
35 REM FIM DE PROGRAMA COM ENTRADA NULA
40 IF A$="" THEN END
50 GOTO 20
```

Na execução do programa acima, tente entrar uma série com mais de 19 caracteres. Aparecerá a mensagem *** STR OVFL ERR e o programa parará. O comprimento da série entrada não pode exceder o comprimento máximo da variável série usada como variável no INPUT.

O Integer BASIC força que se entre cada variável numa linha diferente. Se a declaração INPUT especifica uma lista de variáveis, e existem variáveis tipo série na lista, os respectivos valores tipo série devem ser entrados em linhas diferentes. Isso acontece porque o Integer BASIC permite que se coloquem vírgulas como parte da variável série. Veja isso, rodando o exemplo acima e colocando a variável PÃO, MANTEIGA. O exemplo seguinte mostra o que acontece quando uma variável série faz parte de uma declaração INPUT no Integer BASIC. Tente com este programa; tente entrar todos os quatro valores na mesma linha, separados por vírgulas. O que aconteceu? Tente colocar cada valor numa linha diferente. Veja o que acontece quando se tenta entrar um valor numérico ou vírgula como parte de um valor tipo série.

```
10 DIM A$(10),B$(10)
20 IMPUT A$,A,B$,B
30 PRINT A$,A,B$,B
35 REM FIM DE PROGRAMA COM ENTRADA NULA
40 IF A$=""THEN END
50 GOTO 20
```

Como foi discutido antes, qualquer variável real pode ter um valor inteiro no Applesoft. Assim pode-se entrar um número inteiro para uma variável real. Um valor real entrado para uma

variável inteira é convertido para valor inteiro de acordo com as regras de truncagem apresentadas na seção "Expressões do Tipo Misto" neste capítulo.

Caracteres da Declaração INPUT

A declaração INPUT é muito fastidiosa; sua sintaxe é pesada para qualquer operador humano normal. Imagine um operador novato de escritório que não entenda nada de programação; encontrando os tipos de mensagens de erro que aparecem se uma vírgula é colocada fora de lugar, certamente ele ficará desesperado. Em vista disso, pode-se perder muito tempo escrevendo programas "à prova de idiotices" na entrada de dados. Esses programas devem prever todos os tipos de enganos que se possam cometer na digitação de dados, e devem responder aos erros de forma simples e incisiva de maneira que o operador possa entender. No Capítulo 4 se descrevem essas técnicas de programação de entrada de dados em detalhes.

Uma forma interessante de orientar o operador na declaração INPUT é sua habilidade de colocar uma pequena mensagem que descreve a entrada desejada. Chamamos essa mensagem de *mensagem de entrada*. Ela aparece na declaração INPUT como uma série entre aspas, e é colocada à frente da entrada desejada. Essa mensagem permite colocar várias variáveis dentro de uma mesma declaração INPUT, auxiliando a memória sobre o que deve ser entrado.

No Integer BASIC, coloca-se a mensagem imediatamente após a palavra INPUT, e segue-se a lista de variáveis. Quando a lista contém mais de uma variável, a mensagem aparece apenas uma vez, na primeira linha de entrada. Se a primeira variável da lista é numérica, aparece um ponto de interrogação após a mensagem. Se a entrada for série, não aparece a interrogação. Aqui está um exemplo:

```
10 DIM A$(10)
20 INPUT "ENTRE SEU NOME E IDADE ",A$,A
30 PRINT A$;" TEM";A
35 REM "ENTRADA NULA=FIM DE PROGRAMA"
40 IF A$=0 THEN END
50 GOTO 20
```

No Applesoft, coloca-se a mensagem após a palavra INPUT. Ela é seguida por dois pontos e depois vem a lista de variáveis. A existência de dois pontos suprime o ponto de interrogação. A mensagem aparece apenas uma vez, mesmo que seja necessária mais de uma linha para entrar todas as variáveis.

Aqui está um exemplo em Applesoft:

```
20 INPUT "ENTRE SEU NOME E IDADE:";A$,A
30 PRINT A$;"TEM";A
35 REM "USE CTRL-C PARA FINALIZAR"
50 GOTO 20
```

A Declaração GET

A declaração GET, disponível apenas no Applesoft, entra um único caractere do teclado. Esse caractere não aparece na tela. Não é preciso digitar o RETURN após a digitação. Essa entrada é

tratada como uma variável série ou como um valor numérico, dependendo da variável que segue a palavra GET. Digite o seguinte programa e execute-o:

```
10 GET A$
20 PRINT A$
25 REM "SE A ENTRADA FOR E, FIM DE PROGRAMA"
30 IF A$="E" THEN END
40 GOTO10
```

Pode-se fazer a declaração GET esperar por um caractere específico, testando a entrada, como segue:

```
10 GET A$
20 IF A$<>"X" THEN GOTO 10
30 PRINT A$
40 END
```

Este programa espera pela digitação da letra X. Fora isso nada acontece.

Se a declaração GET especifica uma variável real ou inteira a entrada deve ser numérica. Em caso contrário ocorre uma mensagem ?SYNTAX ERROR e o programa pára. Devido a esse e a outros problemas que podem ocorrer com a declaração GET, em geral ela só é usada para receber caracteres tipo série.

Os programas usam a declaração GET mais freqüentemente quando dialogam com o operador. Por exemplo, um programa pode esperar a decisão de um operador aguardando dele (ou dela) o caractere "S" para "SIM". Aqui segue um exemplo dessa lógica:

```
10 PRINT "OPERADOR, VOCE ESTA AI ?"
15 PRINT "DIGITE SIM OU NAO"
20 GET A$
30 IF A$<>"Y" THEN GOTO 20
40 PRINT "OK, VAMOS CONTINUAR!"
50 END
```

Note que essa seqüência nunca mostra o caractere entrado no teclado. Tente reescrever o programa de forma que qualquer caractere digitado na declaração GET seja mostrado na tela.

EXECUÇÃO DE PARADA E CONTINUAÇÃO DE PROGRAMAS

Se um programa estiver rodando e se desejar pará-lo, basta digitar CTRL e C simultaneamente. Se o programa estiver esperando por entrada de teclado através de uma declaração INPUT, ao se acionar CTRL-C, deve-se seguir com o acionamento da tecla RETURN.

No Integer BASIC, aparece a mensagem STOPPED AT e o número da linha onde parou a execução do programa. Ele continua se for dado o comando CON.

No Applesoft, a mensagem é BREAK IN seguida pelo número da linha onde a execução parou ao se acionar CTRL-C. A execução continua com o comando CONT.

A Tecla RESET

É claro que se pode interromper o programa com a tecla RESET (ou CTRL-RESET em algumas versões do Apple II).

No Apple II Plus e no Apple II com o Language System instalado, o RESET tem o mesmo efeito que o CTRL-C.

Nas versões do Apple II sem o Autostart Monitor, o acionamento do RESET causa o aparecimento do caractere de entrada do monitor (*). Se se estiver usando o Integer BASIC ou o Firmware Applesoft, digite CTRL-C para retornar para a linguagem que estava usando. Se o sistema for o Applesoft baseado em cassette, digite OG para retornar ao Applesoft. Se o sistema for Applesoft baseado em disco, digite 3DOG para retornar ao Applesoft.

Caso se tente voltar incorretamente a partir de um RESET acidental, o programa em BASIC estará perdido.

A Declaração END

O programa pára sua execução quando encontra a declaração END, como foi descrito anteriormente, neste capítulo.

Não se pode continuar a executar o programa, se for encontrado em END no Integer BASIC.

A Declaração STOP

O Applesoft tem outro comando que pára a execução do programa quando encontrado: o comando STOP. Quando o Applesoft executa o comando STOP, aparece a mensagem BREAK IN seguida pelo número da linha onde o programa parou.

Pode-se continuar a execução digitando CONT.

A Declaração WAIT

O Applesoft tem uma declaração que permite especificar uma pausa na execução do programa. A declaração WAIT faz o programa parar até que a posição de memória especificada tenha um valor também especificado. Pode-se, por exemplo, parar o programa até que alguém aperte o botão no controlador de jogo 1. Aqui está como:

```
10 REM ESPERA POR BOTAO NO CONTROLADOR DE JOGO 1
20 PRINT "ACIONE O BOTAO NO CONTROLADOR DE JOGO 1
30 WAIT - 16286,128
40 PRINT "BANG!"
50 END
```

Consulte o Capítulo 8 para maiores detalhes sobre a declaração WAIT.

FUNÇÕES

Outro elemento do BASIC é a função, que de certa forma se parece com uma variável, mas pode se comportar mais como uma declaração BASIC.

Talvez a melhor forma de se entender o que a função faz seja através de um exemplo. Na declaração seguinte:

```
10 A=SQR(B)
```

a variável A é colocada como a raiz quadrada da variável B. O SQR especifica a função raiz quadrada. Veja a seguinte função em série de caracteres:

```
20 L=LEN(D$)
```

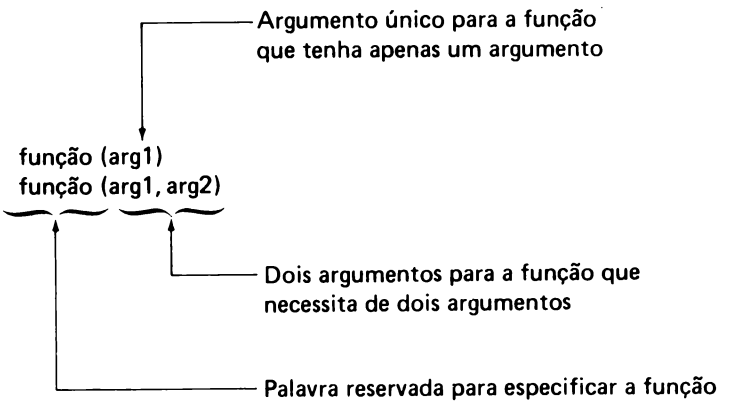
Nesse exemplo, a variável L é colocada como sendo o comprimento da variável D$.

As funções podem substituir variáveis ou constantes em qualquer lugar numa declaração BASIC, à exceção da esquerda do sinal de igual. Em outras palavras, pode-se dizer A=SQR(B), mas não é permitido dizer SQR(A)=B.

Em seguida veremos como funcionam as funções e como usá-las. Mostraremos aqui um sumário incompleto das funções permitidas no Integer BASIC e no Applesoft. Uma descrição completa das funções foi colocada no Capítulo 8. Muitas das funções não são permitidas no Integer BASIC. Isso será indicado no momento apropriado.

Especifica-se a função através de uma palavra reservada apropriada (como SQR para raiz quadrada), seguida por um argumento ou argumentos entre parênteses. No caso de A=SQR(B), SQR requer apenas um argumento, que no caso é o valor de onde deve ser extraída a raiz quadrada. Para L=LEN(D$), LEN especifica a função e o argumento D$, entre parênteses, que é a série da qual se obtém o comprimento.

Na forma geral, qualquer função terá um desses dois formatos:



Poucas funções necessitam de três argumentos. Cada argumento de função pode ser uma constante, uma variável ou uma expressão.

Todas e cada uma das funções numa declaração BASIC são reduzidas a um único valor numérico ou série antes que qualquer outra parte da declaração BASIC seja calculada. Primeiro o argumento da função é calculado de acordo com as regras mostradas antes. Uma vez reduzido a um valor numérico ou série, aplica-se a função ao argumento, obtendo-se outro valor numérico ou série. A expressão não é calculada até que todas as suas funções sejam calculadas. Por exemplo, veja o seguinte exemplo:

```
]10 B=24.7*(SQR(C)+5)-SIN(0.2+D)
```

As funções SQR e SIN são calculadas primeiro. Suponhamos SQR(C)=6.72 e SIN(0.2+D)=0.625. A expressão na linha 10 será primeiramente reduzida para:

```
24.7*(6.72+5)-0.625
```

e depois esta expressão simples é calculada.

FUNÇÕES NUMÉRICAS

Aqui está uma lista de funções numéricas que podem ser usadas tanto em Integer BASIC quanto em Applesoft:

- SGN Retorna o sinal de argumento: +1 se argumento positivo, −1 se argumento negativo, 0 para argumento zero.
- ABS Retorna o endereço absoluto de um argumento. Um argumento positivo não modifica; um argumento negativo é convertido para o positivo correspondente.
- RND Gera um número aleatório. Veja o Capítulo 8 para mais detalhes.

Aqui está uma lista de funções numéricas que só podem ser usadas com o Applesoft:

- INT Converte um argumento de ponto flutuante para seu inteiro equivalente.
- SQR Calcula a raiz quadrada de um argumento.
- EXP Eleva a constante "e" à potência que é o argumento.
- LOG Calcula o logaritmo natural de um argumento.
- SIN Retorna o seno trigonométrico do argumento, em radianos.
- COS Retorna o co-seno trigonométrico do argumento, em radianos.
- TAN Retorna a tangente trigonométrica do argumento, em radianos.
- ATN Retorna o arco-tangente trigonométrico do argumento, em radianos.

Usando Funções Numéricas

Deve-se usar as funções numéricas tão cedo quanto possível, mas não se preocupe com as funções que lhe são desconhecidas. Por exemplo, caso não exista o conhecimento sobre o comportamento das funções seno, co-seno e tangente, não é aconselhável colocá-las nos programas, posto que elas perdem o sentido.

Aqui estão alguns exemplos que usam funções numéricas:

```
10 A=-234
20 B=SGN (A)
30 PRINT B
40 END
```

Na execução deste programa, o resultado apresentado será −1, porque −234 é negativo. Como exercício, troque a declaração na linha 10 por INPUT. Troque a linha 40 para GOTO 10. Agora pode entrar uma variedade de valores de A e ver a função SGN funcionar.

Aqui está um exemplo mais complexo usando as funções numéricas do Applesoft BASIC:

```
10 INPUT A,B
20 IF LOG (A) < 0 THEN A=1/A
30 PRINT SQR (A) * EXP (B)
35 REM USE CTRL-C PARA TERMINAR O PROGRAMA
40 GOTO 10
```

Se houver domínio completo de sua parte sobre a função logaritmo, troque a declaração da linha 20, substituindo a função LOG por outra função numérica qualquer.

FUNÇÕES SÉRIE

As funções série permitem o manuseio de dados tipo série numa variedade de formas. Pode não ser necessário o uso de funções numéricas desconhecidas, mas é necessário um esforço para aprender todas as funções que operam com as séries.

Aqui vai uma lista de funções que podem ser usadas tanto no Integer BASIC quanto no Applesoft:

ASC	Converte uma série de caracteres no seu código padrão numérico (código ASCII) equivalente.
LEN	Retorna o número de caracteres contidos numa série de texto.

Aqui estão as funções de série que só podem ser usadas com o Applesoft:

STR$	Converte um valor numérico em uma série de caracteres de texto.
VAL	Converte uma série de texto em seu número equivalente (se tal conversão for possível).
CHR$	Converte um código numérico (ASCII) em seu equivalente caractere de texto.
LEFT$	Extrai a parte esquerda da série de texto. Os argumentos da função identificam a série de texto e a parte da esquerda desejada.
RIGHT$	Extrai a parte direita da série de texto. Os argumentos da função identificam a série e a parte da direita desejada.
MID$	Extrai uma parte do meio da série. Os argumentos da função identificam a série e o pedaço desejado.

As funções tipo série permitem determinar o comprimento de uma série, extrair parte dela e converter valores numéricos, códigos numéricos (ASCII), e caracteres da série. Essas funções usam um, dois ou três argumentos. Aqui estão alguns exemplos:

STR$(14)	**Converte 14 para "14".**
LEN("ABC")	Retorna o comprimento da série. O número 3 volta porque a série tem 3 caracteres.
LEN(A$+B$)	Retorna o comprimento da concatenação das duas séries.
LEFT$(ST$,1)	Retorna o caractere mais à esquerda da série ST$.

Subséries do Integer BASIC

Embora o Integer BASIC não tenha funções que permitam extrair partes de uma série, existe uma forma de fazê-lo. Especifica-se a posição inicial e o número de caracteres numa subsérie, como no exemplo que se segue:

```
10 DIM A$(20),B$(5)
20 B$=A$(1,4)
```

No exemplo acima, B$ é colocado igual aos quatro primeiros caracteres de A$. Pode parecer que atribuiu-se a B$ o valor de um dos elementos de uma matriz tipo série chamada A$, mas lembre-se de que o Integer BASIC não permite matrizes tipo série, muito menos tais matrizes com duas dimensões. Perceba bem que tal notação se refere a uma subsérie.

O primeiro valor entre parênteses é a posição inicial da subsérie e o segundo valor é o número de caracteres da subsérie.

Concatenação de Séries no Integer BASIC

A função LEN permite que se concatene séries no Integer BASIC. Aqui está um exemplo:

```
10 DIM A$(10),B$(10),C$(10)
20 A$="VENTO"
30 B$="INHA"
40 C$="CATA"
50 A$(LEN(A$)+1)=B$
60 PRINT A$
70 C$(LEN(C$)+1)=A$
80 PRINT B$
90 END

>RUN
VENTOINHA
CATAVENTOINHA
```

FUNÇÕES DO SISTEMA

Para efeito de complementação, as funções do sistema são listadas abaixo. Elas executam operações que não terão maior interesse até que se tenha experiência em programação.

Aqui está a lista de funções do sistema disponíveis:

PEEK	Pega o conteúdo de uma posição de memória.
FRE	Retorna o espaço livre disponível — o número de bytes de memória RAM. Não permitido no Integer BASIC.
USR	Transfere a execução para um programa em linguagem assembler. Não permitida no Integer BASIC.

FUNÇÕES DEFINIDAS PELO USUÁRIO

Além das várias funções que são disponíveis dentro do BASIC padrão, pode-se definir funções aritméticas próprias do usuário no Applesoft, uma vez que elas não são muito complicadas. Funções tipo série definidas pelo usuário não são permitidas. Uma declaração DEF FN define a função. Chama-se a função com declaração FN. Aqui está um pequeno programa que usa a declaração DEF FN:

```
10 DEF FN P(X) = 100*X
20 INPUT A
30 PRINT A,FN P(A)
35 REM USE CTRL-C PARA SAIR DO PROGRAMA
40 GOTO 20
```

O identificador da função segue a palavra reservada FN. As regras para batizar o identificador são as mesmas usadas para dar nomes a variáveis reais. No exemplo, usamos P, de forma que a função passou a se chamar FNP. Se o identificador fosse AB, o nome da função seria FNAB.

A expressão aritmética do lado direito do sinal de igual define a função. Quando se chama a função com a declaração FN, a expressão é calculada (usando os valores correntes de qualquer variável mencionada na expressão). O resultado numérico obtido é tratado da mesma forma que qualquer outro valor numérico no contexto onde a declaração FN aparece.

Na declaração DEF FN, uma única variável deve seguir o identificador da função, e deve ser colocada entre parênteses. O nome da variável é local, dentro da definição da função; ele não tem efeito fora da declaração DEF FN. Pode-se usar o mesmo nome de variável em qualquer outro local do programa sem afetar a função, e a variável na função não afetará outro nome de variável idêntico em outro local dentro do programa.

Em uso, a declaração FN deve ser seguida pelo identificador da função, que deve ser seguido por uma constante numérica, variável ou expressão entre parênteses. Quando o Applesoft encontra uma declaração FN, ele atribui os valores da constante, da variável ou da expressão às variáveis locais da declaração FN. (O valor de uma variável fora da declaração DEF FN que tenha o mesmo nome da variável local não é alterada.) Se a variável local aparece na expressão que define a função, o Applesoft usa seu valor mais recentemente atribuído para calcular a expressão.

FUNÇÕES ANINHADAS

O argumento de uma função pode ser uma expressão; a expressão pode conter funções. Em outras palavras, as funções podem ser aninhadas. Aqui está um exemplo:

```
10 INPUT A
20 PRINT SGN( ABS (A))
25 REM USE CTRL-C PARA FINALIZAR O PROGRAMA
30 GOTO 10
40 END
```

Realize experiências criando declarações PRINT em modo imediato para fazer uso mais complexo das funções numéricas e de cordão.

4

Programação Basic Avançada

Este capítulo continua as orientações do Capítulo 3 de como programar o Apple II em BASIC. Ele cobre outras declarações novas em BASIC e explora diferentes facetas das declarações já vistas. O Capítulo 3 mostrou como fazer seu Apple II realizar algumas funções interessantes; este capítulo mostra como torná-lo uma ferramenta realmente útil.

ACESSO DIRETO E CONTROLE

Existem algumas declarações que permitem acessar diretamente o Apple II. Existem muitas coisas que se podem fazer através delas — como operar controladores de jogos, operar os alto-falantes e fazer pleno uso dos dispositivos periféricos.

MEMÓRIA E ENDEREÇAMENTO

O Apple II pode ter até 65.536 locais de memória endereçáveis individualmente, sendo que cada um pode conter um número variando entre 0 e 255 (este limite um tanto estranho é, na verdade, 2 elevado ao expoente 8). Todos os programas e dados são convertidos em seqüências de números que são armazenados nessa forma.

Deve-se fornecer o endereço específico de memória para cada uma das seguintes declarações BASIC. Pode-se especificar o endereço com um número, uma variável ou uma expressão. Em qualquer caso, ele deve ser calculado e referenciar um endereço de memória real. Existem dois endereços válidos para cada endereço de memória. Um é positivo e inteiro entre 0 e 65.535. O outro é negativo e pode ser obtido subtraindo 65.536 do endereço positivo. Por exemplo, −32768 e

32768 são o endereço da mesma posição de memória. Essa é uma posição de memória que o Integer BASIC não pode acessar diretamente. Outra posição de memória é endereçada tanto por −1 quanto por 65535. Quando se recorda que o maior número permitido no Integer BASIC é 32767, pode-se ver a utilidade em usar números negativos para endereçar as posições de memória mais altas. Se especificarmos um endereço de memória no Applesoft com um valor real, ele será convertido a um valor inteiro para ser usado como endereço.

PEEK e POKE

A função PEEK permite ler o valor guardado em qualquer endereço da memória do Apple II. Considere a seguinte declaração:

```
10 A = PEEK (200)
```

Esta declaração atribui à variável A o valor da posição de memória 200.

A declaração POKE coloca um valor numa posição de memória. Por exemplo, a declaração:

```
20 POKE 8000,A
```

toma o valor da variável A e guarda na posição de memória 8000. O valor a ser escrito na memória pode ser um número, uma variável ou uma expressão com um valor final entre 0 e 255.

Pode-se dar a declaração PEEK em memória de leitura e escrita ou apenas de leitura (RAM ou ROM). Porém o POKE só pode ser dado em memória de leitura e escrita (RAM). Isso é evidente pois a característica básica da ROM é que ela somente pode ser lida, porém não se pode gravar nada nela.

Declaração CALL

Pode-se transferir o controle do BASIC para uma rotina em linguagem assembler com a instrução CALL. Veja este exemplo simples de declaração CALL:

```
100 CALL A1
```

O controle é transferido para a posição de memória cujo endereço está contido na variável A1.

A sub-rotina em linguagem assembler ou o programa pode ser um residente no Apple II (ROM) ou pode ser gerado pelo usuário. O Monitor tem uma sub-rotina intrínseca que limpa a tela, por exemplo. O Apêndice D tem uma lista completa de sub-rotinas intrínsecas que podem ser úteis. O Capítulo 7 também faz referência com mais detalhes ao monitor e à linguagem assembler.

HIMEM e LOMEM

A memória do Apple II é usada de formas diferentes. Parte dela é usada para guardar o programa BASIC, parte para as variáveis do programa, outra parte para os programas que mexem com drives de disco (se estão sendo usados) etc. Uma parte da memória é usada para gráficos, como se verá mais tarde no Capítulo 6.

As declarações HIMEM e LOMEM permitem reservar áreas de memória para programas em linguagem assembler e gráficos de alta resolução.

Aqui está um exemplo de cada uma destas declarações:

```
50 HIMEM: 38400
60 LOMEM: 12291
```

Os limites superior e inferior são dados automaticamente pelo Apple II pelas posições de memória mais alta e mais baixa disponíveis ao programa BASIC pelo equipamento Apple II particular. Deve-se redefinir estas posições no caso de se usar gráfico de alta resolução ou programas em linguagem assembler, o que será visto em detalhes nos Capítulos 6 e 7. Para mais informações sobre o uso da memória, veja o Apêndice G.

USANDO DISPOSITIVOS PERIFÉRICOS

Quando se liga o Apple II, ele espera entrada de dados pelo teclado e os apresenta na tela. O teclado é o dispositivo de entrada padrão e a tela de vídeo é o dispositivo padrão de saída. Todavia, como já é sabido, o Apple II pode se comunicar com outros dispositivos de entrada e saída. São eles:

- Gravador cassette para salvar e carregar programas (ou dados)
- Drives de disco para guardar dados e programas
- Impressora para cópia permanente de dados e programas
- Tábua Gráfica para entrada de dados gráficos desenhados à mão livre
- Comunicação com outros computadores.

O gravador cassette é conectado ao Apple II por intermédio de dois conectores especiais.

Todos os demais dispositivos periféricos são conectados através de circuitos eletrônicos adicionais que se conectam aos slots internos do Apple II. Essas placas de circuito são chamadas *controladores, cartões de interface, cartões* ou *interfaces*. Especifica-se que dispositivo se deseja usar através do número do slot ao qual será conectado o controlador. Existem oito slots, numerados de 0 a 7. A tela de vídeo é sempre o slot 0.

As Declarações PR# e IN#

Para selecionar uma saída diferente usa-se a declaração PR#. Para uma entrada diferente usa-se o IN#. O exemplo seguinte em modo imediato seleta o vídeo como saída:

```
PR#0
```

Se o Sistema Operacional em Disco (DOS) não está presente, usa-se o PR# e o IN# da mesma forma em modo imediato e programado. Usando o PR# e o IN# em modo programado com o DOS presente é um pouco mais complicado. Deve-se usar a declaração PRINT que coloca o caractere CTRL-D seguido imediatamente pelo comando PR# ou IN#. As declarações seguintes direcionam a declaração da saída PRINT para o slot 1. Através da interface deste slot, a saída irá para uma impressora ou qualquer outro dispositivo conectado ao cartão.

```
100 D$="":REM CTRL-D
110 PRINT D$;"PR#1":REM SELECIONA SLOT 1 (IMPRESSORA)
```

Sem o DOS, pode-se usar alternativamente a declaração seguinte:

```
110 PR#1 : REM SELECIONA SLOT 1 (IMPRESSORA)
```

Veja que tanto o PR# como o IN# têm um parâmetro singular cada um. Este deve ser um valor numérico entre 0 e 7. Qualquer outro valor causará resultados inesperados. Se não existe cartão de interface no slot selecionado por PR# ou IN#, o Apple II trava; o único recurso será acionar RESET.

SAÍDA DE PROGRAMA E ENTRADA DE DADOS

Os programadores menos experientes logo descobrem que a parte de entrada e saída de um programa é a parte mais delicada.

Quase todo o programa usa dados que devem ser acessados pelo teclado. Serão suficientes as poucas declarações INPUT? Na maioria dos casos a resposta é não. O que acontece se o operador aciona acidentalmente a tecla errada — após digitar dois ou três dados? Um bom programa deve saber que o operador é humano e é provável que cometa erros.

Os resultados, por isso, não podem ser simplesmente mostrados por intermédio de um punhado de declarações PRINT. Um ser humano deverá ler a informação mostrada. A menos que a mensagem e o dado sejam projetados com cuidado, poderá ser muito difícil ler e interpretar e, em conseqüência, pode haver mal-entendido ou confusão total.

Felizmente o BASIC do Apple II tem muitas características que permitem que se façam entradas e saídas corretas. Descreveremos algumas dessas características antes de abordarmos as práticas de programação específicas para boas entradas e saídas.

MAIS SOBRE A DECLARAÇÃO PRINT

Normalmente a execução de uma declaração PRINT termina retornando o cursor para o início da próxima linha (com o *carriage return*). Isso faz com que a próxima declaração PRINT inicie com o primeiro caractere colocado na primeira posição da próxima linha. Assim o programa seguinte apresenta uma coluna de 20 caracteres W como primeiro caractere em 20 linhas:

```
>NEW
>200 C$="W"
>210 FOR I=1 TO 20
>220 PRINT C$
>230 NEXT I
>240 PRINT "FIM"
>250 END
>RUN
W
W
```

```
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
FIM
```

Usando Ponto-e-Vírgula

Um ponto-e-vírgula (;) colocado depois de qualquer variável na lista de parâmetros da declaração PRINT causa o aparecimento do próximo caractere após o último caractere mostrado. Um ponto-e-vírgula seguindo a última (ou a única) variável na declaração PRINT suprime o retorno à próxima linha. Dessa forma o programa seguinte mostrará 800 caracteres em 20 linhas de 40 colunas.

```
>NEW
>200 C$="W"

>210 FOR I=1 TO 800
>220 PRINT C$;
>230 NEXT I
>240 PRINT "FIM"
>250 END

>RUN

WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

```
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
FIM
```

O índice do loop FOR-NEXT, a variável I, é usada como contador para indicar o número de W's a ser mostrado, no caso 800. Na primeira execução do PRINT, uma nova linha é iniciada para a impressão do W. O ponto-e-vírgula evita o retorno para a linha seguinte, de forma que o cursor fica na posição seguinte ao caractere mostrado. O segundo W é mostrado à frente do primeiro e o cursor vai uma posição à frente. A seqüência continua até o fim da primeira linha, quando então o cursor se move para o início da linha seguinte. Isso se processa durante 20 linhas (de 40 colunas cada).

O que imprime o FIM numa nova linha? Realmente nada; ele aparece no início da linha porque o último W mostrado foi colocado na última posição da linha anterior. Troque o loop FOR-NEXT de 800 para 780 vezes e veja que o FIM será mostrado adiante do último W. Isso pode ser visto adiante:

```
>200 C$="W"
>210 FOR I=1 TO 780
>220 PRINT C$;
>230 NEXT I
>240 PRINT "FIM"
>250 END

>RUN
```

```
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

```
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWFIM
```

O ponto-e-vírgula concatena séries de dados mostrando itens, um atrás do outro, sem espaços entre eles. Dados numéricos também são mostrados em formato de linha contínua, sem espaços entre valores contíguos.

Para ver isso, troque o valor da variável para um número de apenas um dígito. O programa será assim:

```
>200 C=5
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "FIM"
>250 END
```

O número 5 é mostrado como segue:

```
>RUN

5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
FIM
>
```

Agora troque o valor de C por −1:

```
>200 C=-1
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "FIM"
>250 END
>RUN

-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
```

```
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
FIM
```

O que aconteceu aqui? Quando C é −1, o programa mostra duas vezes mais linhas do que quando C=5. Usa-se uma posição extra na tela para mostrar o sinal de menos.

Números de múltiplos dígitos (incluindo negativos) vão estourar a capacidade da tela, a menos que o loop FOR-NEXT seja ajustado. Se o C for trocado por 2001, um campo de quatro dígitos será requerido, de forma que devemos ajustar o índice de 800 para 800/4=200:

```
>200 C=2001
>210 FOR I=1 TO 200
>220 PRINT C;
>230 NEXT I
>240 PRINT "FIM"
>250 END
>RUN

2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
FIM
```

Os números são cortados no fim da linha, se necessário. Isso é porque o ponto-e-vírgula (;) será contínua apresentação, e nada além do fim de linha pode causar um retorno. Tente o exemplo acima com C igual a 201 para ver isso.

Usando Vírgulas

O aparecimento de vírgulas entre variáveis ou no fim de uma declaração PRINT trata os dados mostrados como se eles estivessem tabelados em intervalos específicos. As regras de tabulação são diferentes no Integer BASIC e no Applesoft.

O Integer BASIC tem cinco paradas de tabulação na tela. São as colunas 1, 9, 17, 25 e 33; existem 8 espaços entre cada uma. Aqui está um exemplo:

```
]NEW
] 200 C=123
] 210 FOR I=1 TO 100
] 220 PRINT C,
] 230 NEXT I
] 240 PRINT "FIM"
] 250 END
]RUN

123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
123      123      123      123      123
FIM
```

Deve haver pelo menos um espaço em branco à frente da posição de tabulação (à exceção da primeira) ou a posição de tabulação ficará inativa. Assim, não deve haver nada colocado na coluna 8 para que a segunda posição de tabulação seja ativa, e assim por diante. O programa seguinte ilustra isso com séries de texto:

```
>10 REM DEVE-SE DIMENSIONAR AS VARIAVEIS CORDAO
>20 DIM C$(8)
>200 C$="12345678"
>210 FOR I=1 TO 20
>220 PRINT C$,
>230 NEXT I
>240 PRINT "FIM"
>250 END
>RUN

12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
12345678        12345678        12345678
        12345678        12345678
FIM
```

Se trocarmos o C$ por "1234567", todas as cinco posições de tabulação serão usadas.

Existem algumas condições especiais que governam a tabulação com vírgula no Applesoft, como mostrado na Figura 4.1.

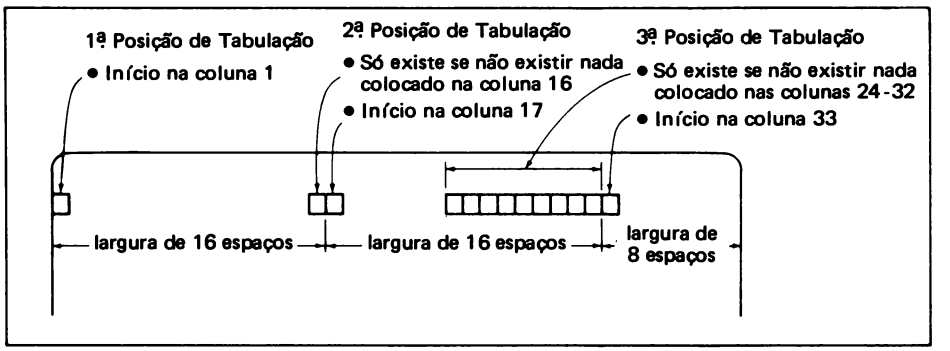


FIGURA 4.1. Paradas de Tabulação do PRINT no Applesoft.

Para ver como as vírgulas funcionam no Applesoft, usando nosso programa de exemplo, troque o ponto-e-vírgula da declaração PRINT por vírgula. Isso causa a colocação dos números em três colunas. Com três números por linha, o índice do loop FOR-NEXT será 3*20=60. O programa completo é mostrado a seguir.

```
]200 C = 2001
]210 FOR I = 1 TO 60
]220 PRINT C,
]230 NEXT I
```

```
]240 PRINT "FIM"
]250 END
RUN

2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
2001            2001            2001
FIM
```

As vírgulas também funcionam com séries. Como exemplo, coloque o seguinte programa para mostrar vinte linhas de caracteres alinhados:

```
]NEW
]100 A$="TERRA"
]110 B$="AGUA"
]120 C$="VIDA"
]210 FOR I=1 TO 20
]220 PRINT A$,B$,C$
]230 NEXT I
]240 PRINT "FIM"
]250 END
]RUN

TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
```

```
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
TERRA           AGUA            VIDA
FIM
```

Como exercício, tente colocar B$ como ATRIBUTOS no lugar de AGUA. Se for visto na Figura 4.1, não haverá surpresa ao se ver que a terceira posição de tabulação não será usada. A palavra ATRIBUTOS, iniciando na segunda posição de tabulação na coluna 17, mostra dados nas colunas de 24 até 32. Colocando qualquer coisa nas colunas 24 a 32 desativa a terceira posição de tabulação.

FUNÇÕES DE FORMATAÇÃO DE IMPRESSÃO

Usa-se a palavra *formatação* para descrever o processo de arranjar a informação na tela de vídeo de forma que ela seja mais fácil de ser lida e entendida, e mais agradável para a vista. Como se viu, os sinais de vírgula e ponto-e-vírgula ajudam. A formatação pode ainda ser uma tarefa desafiante. Por exemplo, suponha que se quer mostrar um cabeçalho no meio da primeira linha no alto da tela.

Aqui está um programa em Integer BASIC que usa vírgulas para tabular no centro da tela e imprimir o cabeçalho. Também é usada a declaração CALL, recentemente estudada, para posicionar o cursor no canto esquerdo do alto da tela (veja Apêndice D).

```
>NEW
>5 REM DEVE-SE DIMENSIONAR CORDOES
>10 DIM HD$(30)
>500 REM LIMPAR A TELA E
>510 REM MOVER O CURSOR PARA
>520 REM O CANTO SUPERIOR ESQUERDO
>530 CALL -936
>540 HD$="RELATORIO"
>550 REM TABULA NO CENTRO DA TELA
>555 REM E IMPRIME O CABECALHO
>560 PRINT "","",HD$
>570 END
>RUN

                    RELATORIO
```

O cabeçalho ficou aproximadamente centrado na tela. Se ele fosse mais extenso, por exemplo RELATÓRIO PARCIAL DE AGOSTO, poderia não ficar centrado. (Tente.) Este novo cabeçalho será aproximadamente centrado quando se remover uma das vírgulas e série nula correspondente, de forma que a série é colocada a partir da segunda posição de tabulação no lugar da terceira. terceira.

Pode-se usar a mesma técnica para mostrar cabeçalhos no Applesoft. Porém o Applesoft tem recursos de formatação da declaração PRINT mais fáceis de usar: as funções SPC, TAB e POS.

A Função SPC

A função SPC salta espaços. Inclui-se o SPC como um dos termos da declaração PRINT; após as letras SPC deve-se incluir (entre parênteses) o número de posições que se deseja pular. Por exemplo, pode-se centralizar o cabeçalho RELATÓRIO com o seguinte:

```
]100 REM LIMPA A TELA
]110 HOME
]120 REM PULA ESPACOS E DA MENSAGEM
]130 PRINT SPC(15);"RELATORIO"
]140 END
]RUN

                RELATORIO
```

Veja o ponto-e-vírgula após o SPC. Uma vírgula faria iniciar a apresentação do texto na próxima posição de tabulação seguinte ao número de espaços especificados pelo SPC.

Sempre que se inclui a função SPC numa declaração PRINT, simplesmente faz-se com que o próximo caractere a ser colocado no vídeo apareça deslocado do número de posições especificado pelo SPC, em relação ao local em que originariamente apareceria; nada mais acontece.

A Função TAB

A função TAB opera de forma muito semelhante ao que acontece com a tabulação de uma máquina de escrever.

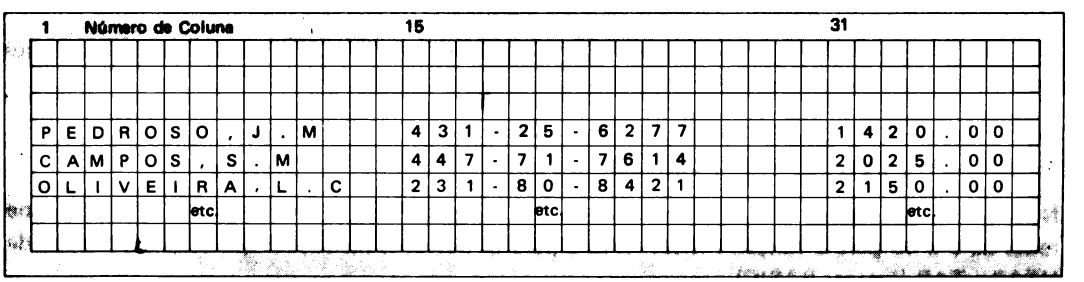


FIGURA 4.2. Determinando a Posição de Caracteres na Tela.

Suponha que se deseja colocar dados em colunas. Deve-se, em primeiro lugar, calcular a posição em que cada número deve começar. Um formulário como o mostrado no Apêndice L é útil para esse propósito. Na Figura 4.2, as colunas iniciam nas posições 1, 15 e 31. Agora, ao invés de se usar espaços em branco como se faz para ir de coluna a coluna, usa-se a função TAB após cada entrada na declaração PRINT.

Considere a linha mostrada na Figura 4.2. Contando posições de caracteres, poderíamos apresentar a linha sem paradas de tabulação, como segue:

```
]10 ?"PEDROSO,J.M        431-25-6277        1420.00"
```

Ao invés de se colocar espaços em branco, pode-se usar a função SPC e diminuir a declaração, como aqui:

```
]10 ?"PEDROSO,J.M;SPC(7);"431-25-6277";SP
C(5);"1420.00"
```

Mas tabular é mais fácil porque se tabula para um número de coluna conhecido ao invés de contar espaços:

```
]10 ?"PEDROSO,J.M";TAB(16);"431-25-6277";T
AB(32);"1420.00"
```

Determinando a Posição do Cursor (Horizontal)

POS é a última das funções de formatação do Applesoft. Ela mostra em que posição o cursor se encontra. Essa posição é expressa como um número, que é o número da coluna em que o cursor se encontra piscando. Sempre se inclui o argumento inútil (0) depois do POS, como segue: POS(0).

O seguinte comando demonstra a capacidade do POS:

```
]?"POSICAO DO CURSOR =";POS(0)
```

Execute este comando de modo imediato. Na tela aparecerá o seguinte:

```
]?"POSICAO DO CURSOR =";POS(0)
POSICAO DO CURSOR =19
```

O cursor estava na posição 19 após colocar POSIÇÃO DO CURSOR =. Se forem colocados alguns espaços após o "=", e antes do fechamento das aspas, troca-se o número 19 por algum número mais alto.

No Integer BASIC pode-se simular o POS(0) com um PEEK(36) para determinar em que coluna o cursor se encontra. Aqui está o exemplo anterior reformulado:

```
>PRINT "POSICAO DO CURSOR =";PEEK(36)
POSICAO DO CURSOR =19
```

Note que as colunas da tela são numeradas de 0 a 39 para POS, PEEK(36) e SPC. Elas são numeradas de 0 a 40 para a função PRINT TAB e para outras duas instruções que veremos mais tarde, HTAB e TAB (Integer BASIC).

Determinando a Posição do Cursor (Vertical)

Um PEEK(37) dá a posição da linha em que o cursor se encontra.

As linhas são numeradas de 0 a 23 para o PEEK(37), mas de 1 a 24 para o VTAB (visto na próxima seção).

CONTROLE DE CURSOR E EFEITOS ESPECIAIS DE VÍDEO

Existem algumas declarações em BASIC que aumentam a versatilidade do vídeo do Apple II. São elas as declarações FLASH, INVERSE, SPEED, NORMAL, HOME, VTAB e HTAB/TAB. A maior parte destas estão disponíveis apenas no Applesoft. Existem também algumas funções gráficas; são estudadas no Capítulo 6.

Posicionando o Cursor

Já vimos várias formas de controlar o cursor através do uso de vírgula e ponto-e-vírgula com a declaração PRINT. As funções de formatação SPC, TAB e POS do Applesoft também são úteis para posicionar o cursor.

Limpando a Tela do Vídeo

Pode-se limpar a tela e posicionar o cursor no alto à esquerda com a declaração CALL-936, ou com a declaração HOME no Applesoft. Tente digitá-los (um para o Integer BASIC, o outro para o Applesoft):

```
>CALL -936
]HOME
```

Posicionamento Horizontal e Vertical

Existem duas declarações que em conjunto permitem mover o cursor para qualquer posição na tela. VTAB move o cursor verticalmente e HTAB (TAB no Integer BASIC) move o cursor horizontalmente. Deve-se especificar o número da linha para VTAB e o número da coluna para HTAB. A linha mais alta da tela é 1 e a mais baixa 24. A coluna mais à esquerda é a de número 1 e a mais à direita é a 40.

O programa seguinte usa estas duas declarações (no Applesoft) para posicionar o cursor em mostrar um asterisco naquela posição. No caso do Integer BASIC use TAB ao invés do HTAB nas linhas 90 e 120.

```
90 HTAB 1:VTAB (1)
100 INPUT "LINHA?";L
110 INPUT "COLUNA?";C
120 VTAB L: HTAB C
130 PRINT "*";
140 GOTO 90
```

As Declarações INVERSE e NORMAL

Pode-se inverter as partes branca e preta do caractere na tela; a declaração faz isso. Uma vez que a declaração é executada, tudo que for colocado na tela pelo PRINT aparece no modo *vídeo reverso.* Os caracteres que são digitados no teclado, no entanto, são passados para o vídeo em modo normal.

O Apple II retorna ao modo normal quando for executado o comando NORMAL.

Para ver como estes dois comandos operam, tente isso (os caracteres sombreados são em vídeo reverso):

```
]INVERSE
]?"PRETO NO BRANCO"
PRETO NO BRANCO
]NORMAL
]?"BRANCO NO PRETO"
BRANCO NO PRETO
```

O INVERSO e o NORMAL não são aceitos pelo Integer BASIC.

A Declaração FLASH

Além de se poder reverter as partes branca e preta do caractere, pode-se ainda fazer com que ele fique intermitente entre os modos normal e inverso. Use a declaração FLASH para isso. De novo, o NORMAL faz o Apple II retornar ao modo normal.

Aqui está um exemplo (a área sombreada fica piscando):

```
]FLASH
]?"CARACTERES PISCANDO"
CARACTERES PISCANDO
]NORMAL
]?"CARACTERES NORMAIS"
CARACTERES NORMAIS
```

O FLASH não é permitido no Integer BASIC.

A Declaração SPEED

A velocidade com que os caracteres são mostrados no vídeo é variável. Pode-se diminuí-la da sua velocidade normal através da declaração SPEED.

O programa seguinte ilustra como o SPEED funciona:

```
]100 INPUT "VELOCIDADE";VL
]110 SPEED = VL
]120 FOR CT = 1 TO 3
]130 PRINT "VEL"
]140 NEXT
]150 PRINT "FIMVEL"
]160 SPEED = 255
]170 END
```

O valor da expressão (isto é, o valor de VL na linha 110) ajusta a velocidade de impressão; 0 é a menor e 255 é a maior. O SPEED também afeta a velocidade com que os caracteres são mandados para outros dispositivos, não apenas para a tela.

O SPEED não é permitido no Integer BASIC.

JANELAS DE TEXTO

Normalmente, o Apple II utiliza todas as 24 linhas e 40 colunas da tela no modo texto. Usando a declaração POKE, pode-se alterar o tamanho e a posição dessa *janela de texto*. Quatro posições de memória controlam o tamanho, a forma e a posição da janela de texto, como mostrado na Tabela 4.1.

TABELA 4.1. Posições de Memória da Janela de Texto.

Posição de Memória	Controle	Faixa Permitida
32	Margem Esquerda	0 a 39
33	Largura	1 a 40 menos Margem Esquerda
34	Linha de Cima	0 até Linha de Baixo
35	Linha de Baixo	Linha de Cima menos 24

Deve-se ter cuidado em observar o bom senso nos limites de faixa descritos na Tabela 4.1, quando se determina a janela de texto. Tentar determinar janelas fora desses limites dará resultados imprevisíveis.

O exemplo de programa abaixo será uma janela de texto de duas linhas no meio da tela para entrar um valor numérico. Para apreciar toda a utilidade da técnica, tente entrar algum valor não numérico e observe o que acontece com a mensagem de erro. Note também que em se determinando a janela de texto, não se apaga o resto da tela, nem se move o cursor para dentro da janela. Deve-se dar instruções especiais dentro do BASIC para fazer essas tarefas.

```
1000  REM  DETERMINA LINHA  DE  CIMA  DA
JANELA,MARGEM ESQUERDA E LINHA DE BAIXO
1010 T=10:W=20:LM=11:B=13
1020 REM LIMPA A TELA
1030 CALL -936
1040  REM  DETERMINA A JANELA  DE  TEXTO
PARA ENTRADA
1050 GOSUB 3200
1060  REM  LIMITA A JANELA DE TEXTO  COM
ASTERISCOS
1070 GOSUB 3000
```

```
1080  REM  MOVE CURSOR  PARA  DENTRO  DA
JANELA ; ENTRA
1090 VTAB T+2
1100 INPUT M1
1110 REM ELIMINA JANELA - VIDEO INTEIRO
1120 GOSUB 3300
1130 REM MOVE CURSOR.LINHA DE BAIXO
1140 VTAB 23
1150 END
2990   REM   LIMITANDO  A   JANELA   COM
ASTERISCOS
3000 VTAB T+1
3010 GOSUB 3100
3020 VTAB B+1
3030 GOSUB 3100
3040 RETURN
3090 REM COLOCA ASTERISCOS
3100 FOR I=1 TO W
3110 PRINT "*"
3120 NEXT I
3130 RETURN
3190 REM CRIA JANELA DE TEXTO
3200 POKE 32,T
3210 POKE 33,W
3220 POKE 34,LM
3230 POKE 35,B
3240 RETURN
3290 REM GERA JANELA INTEIRA
3300 POKE 32,0
3310 POKE 33,40
3320 POKE 34,0
3330 POKE 35,24
3340 RETURN
```

A FUNÇÃO CHR$: PROGRAMANDO CARACTERES EM ASCII

No capítulo anterior vimos como se pode gerar caracteres invisíveis como o CTRL-G, que dá o *bip*. Porém existem certos caracteres (tanto visíveis quanto invisíveis) que não podem ser digitados diretamente pelo teclado, incluindo "]" e "\". Podem-se gerar estes caracteres no Applesoft com a função CHR$.

Para entender como a função CHR$ opera, deve-se entender como o Apple II guarda os caracteres na memória. Na verdade é bem simples. A memória do computador pode armazenar números, mas não caracteres. Os caracteres devem ser convertidos para códigos numéricos. O Apple II usa o mesmo código que todos os outros microcomputadores, o ASCII (American Stan-

dard Code for Information Interchange). Por exemplo, o código ASCII para a letra A é 65, para B é 66, C é 67 e etc. Existe uma tabela completa do código ASCII no Apêndice I. Sempre que o Apple II estiver operando com série de caracteres, ele interpretará valores numéricos como códigos ASCII para caracteres.

No Applesoft, se não é possível acionar uma tecla para incluir um caractere dentro de uma série de texto, pode-se ainda selecionar o caractere usando seu código ASCII.

A função CHR$ traduz o código numérico em ASCII no caractere equivalente. Por exemplo, para criar o símbolo "$", primeiro ache o seu código ASCII no Apêndice I. Daí use o código com CHR$ como segue:

```
]PRINT CHR$(36)
$
```

Experimente em modo imediato usando qualquer código ASCII entre 0 e 255.

Pode-se usar a função CHR$ em conjunto com séries regulares na declaração PRINT, como segue:

```
]?CHR$(34);"ASPAS!";CHR$(34)
"ASPAS!"
```

A função CHR$ permite incluir caracteres que de outra forma não seriam permitidos, tal como retorno e aspas, como parte da série de texto.

PROGRAMANDO ENTRADA DE DADOS

O objetivo de qualquer programa é minimizar erros na entrada de dados e tornar fácil para o operador o ato de corrigir erros, caso ocorrerem. Existem formas de organizar os dados de entrada de forma a minimizar a ocorrência de erros.

Primeiro, entre *todos* os dados de um bloco e depois processe-os. Não é uma boa prática processar cada dado assim que ele é digitado pelo teclado.

Um programa de mala direta, por exemplo, requer nomes e endereços a serem digitados como dados. É interessante que o bloco nome-e-endereço seja tratado como um único bloco de dados. Em outras palavras, o programa deve pedir nome e endereço, permitindo ao operador entrar toda a informação, e ainda pode alterá-la. Quando o operador tiver certeza de que nome e endereço estão corretos, o programa poderá seguir adiante e processar essa informação. Após isso deverá parar e esperar por outro nome e endereço.

No caso do programa de mala direta seria uma prática ruim pedir o nome, processá-lo imediatamente, pedir cada linha de endereço, tratar cada pedaço do nome e do endereço como unidades separadas com funções distintas.

É sempre uma boa idéia organizar a entrada de dados de forma que eles permaneçam no vídeo por algum tempo após as suas digitações. Isso faz com que seja dada oportunidade ao operador para reparar erros e corrigi-los. Se os dados desaparecem tão logo são entrados, não há oportunidade para que o operador reveja a digitação e ocasionalmente detecte um erro. Naturalmente o operador deve ter alguma forma de corrigir digitações prévias.

Em alguns casos, não é bom deixar o operador digitar os dados em blocos funcionais e ter possibilidade de corrigir erros. O número de vezes em que isso acontece é bem grande. Ocorre quando um grande volume de dados deve ser digitado pelo operador. Por exemplo, suponha que

um operador de teclado deva digitar centenas de nomes e endereços durante o dia. A experiência tem mostrado que um volume de digitação mais correto é obtido quando o operador ignora os erros de digitação na primeira vez que digita. O programa de entrada de dados neste caso não permite que sejam feitas correções na primeira digitação, mesmo que o operador perceba o engano. O digitador deve ignorar o erro e seguir adiante colocando os dados o mais rapidamente possível. Sob este esquema, os dados são digitados duas vezes, de preferência por operadores diferentes. As chances de que os dois operadores cometam o mesmo erro é tão pequena que se pode contar todos os erros achados com as diferenças entre as duas digitações. Um programa subseqüente permite a reentrada dos dados colocados incorretamente.

Entrada de Dados Interativa

Um programa com entrada de dados interativa guia o operador com mensagens e indicações no vídeo. Para demonstrar a entrada de dados interativa vamos colocar um exemplo bem simples. Faremos uma modificação num programa já visto neste capítulo de forma que ele usará o processo interativo de entrada de dados, tornando assim o programa mais fácil de usar.

Aqui está o programa:

```
200  C$ = "W"
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "FIM"
250  END
```

Como foi mostrado, o programa coloca 800 caracteres W seguidos da palavra FIM. Ele opera da mesma forma tanto em Integer BASIC quanto em Applesoft.

Suponha que desejamos trocar o W por qualquer caractere.

Primeiro eliminamos a declaração de determinação do programa. Lembre-se de que para eliminar uma declaração do programa, digita-se o número da linha seguido pela tecla RETURN.

```
210  FOR I = 1 TO 800
220  PRINT C$
230  NEXT I
240  PRINT "FIM"
250  END
```

A linha 200 não está mais no programa.

Digite C$="X" em modo imediato, e execute o programa.

```
]C$="X"
]RUN
FIM
```

A palavra FIM foi colocada mas os X's não. Claramente o valor de C$ não foi transferido para o programa.

O comando RUN coloca todas as variáveis numéricas em 0 e todos os cordões em nulo antes

de iniciar a execução do programa. Então o C$ foi colocado em nulo, e o loop mostrou no vídeo o caractere nulo (um nulo é "nada", não transfere o caractere para o vídeo nem move o cursor).

Existe uma forma de transmitir um valor diferente de C$ a cada vez que se roda o programa? Ao invés do RUN que limpa as variáveis, use GOTO 210 (sendo 210 a primeira linha do programa). Isso não modifica nenhuma variável.

```
]C$="X"

]GOTO 210
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
FIM!
```

Existem apenas dois passos para rodar o programa, mas o processo é desajeitado. Deve-se digitar a linha (a declaração de atribuição), e se for digitado RUN ao invés de GOTO, tudo deve ser reiniciado. Porém, o programa poderia obter o caractere durante sua execução, usando a declaração INPUT.

Digite a linha seguinte:

```
]200 INPUT C$
```

Agora liste o programa para se certificar que tudo foi digitado corretamente:

```
]LIST
200  INPUT C$
210  FOR I = 1 TO 800
220  PRINT C$
230  NEXT I
240  PRINT "FIM"
250  END
```

Rode o programa. O cursor desce uma linha. (No Applesoft aparece um ponto de interrogação à frente do cursor também.) Digite qualquer caractere unitário (e acione RETURN). O caractere entrado é mostrado 800 vezes. Rode o programa novamente. Entre um caractere diferente. O mesmo acontece com o outro caractere. Veja que se for tentado colocar mais de um

caractere por vez no Integer BASIC, o programa pára com a mensagem *** STR OVFL ERR. Isso é porque o C$ não foi dimensionado.

Essa é uma melhoria significativa sobre o programa original. Entretanto atrapalha um pouco ter o cursor piscando na tela enquanto se espera uma digitação pelo teclado. Adicione uma linha no início do programa, pedindo a digitação de um caractere. Digite esta linha:

```
]190 PRINT "DIGITE UM CARACTERE"
```

Liste o programa e verifique se a linha foi digitada sem erros.

Agora o programa dá instruções ao operador. Rode-o várias vezes para mostrar caracteres diferentes e veja como a operação ficou muito mais fácil.

Existe uma importante alteração por fazer. Se for desejado que se repita várias vezes a execução do programa, use a declaração GOTO para voltar ao início do programa em vez de finalizar. Assim não é mais necessário redigitar RUN a cada vez que se deseje nova passagem. Coloque a linha abaixo:

```
]250 GOTO 190
```

Agora ficou ainda mais fácil de operar. Entre o RUN e siga as instruções.

Claramente, deve-se usar agora o CTRL-C para parar o programa. Isso pode ser eliminado programando-se uma tecla especial para parar a execução. Por exemplo, a tecla RETURN poderia ser programada para tal fim. Aqui está como:

```
190  PRINT "ENTRE UM CARACTERE"
200  INPUT C$
205  IF C$ = "" THEN END
210  FOR I = 1 TO 800
220  PRINT C$
230  NEXT I
240  PRINT "FIM"
250  GOTO 190
```

A linha 205 verifica se C$ tem valor nulo. Isso acontecerá quando nada for digitado em resposta à declaração INPUT da linha 200, sendo apenas acionada a tecla RETURN.

Como tarefa final, releia o programa e adicione comentários. Inclua uma observação no índice 800 do FOR-NEXT; pode-se opcionalmente colocar o comentário na mesma linha, usando os dois pontos para separar as declarações:

```
210  FOR I=1 TO 800:REM 800/40=20
```

Informe que uma entrada nula termina o programa:

```
203 REM FIM DE PROGRAMA COM ENTRADA NULA
```

Finalmente, adicione algumas linhas de comentário no início para descrever o programa, e lhe dê um nome. O formato final é mostrado na Figura 4.3.

```
 10 REM ******* REPETECO *******
 20 REM MOSTRA CONTINUAMENTE
 30 REM O CARACTERE
 40 REM DIGITADO NO TECLADO
 50 REM **********************
190 PRINT "DIGITE UM CARACTERE"
200 INPUT C$
203 REM FIM DE PROGRAMA COM NULO
205 IF C$ = " " THEN END
210 FOR I = 1 TO 800: REM 800/40 = 20
220 PRINT C$
230 NEXT I
240 PRINT "FIM"
250 GOTO 190
```

FIGURA 4.3. Programa REPETECO.

Mensagens de Orientação

Qualquer programa que requeira dados deve orientar o operador fazendo perguntas. Essas perguntas são geralmente apresentadas numa só linha e pedem respostas simples, como "sim" ou "não". Por exemplo, a seguinte mensagem pode aparecer:

VAI FAZER ALGUMA ALTERAÇÃO?

Um operador deve responder a essa mensagem com a palavra SIM ou com a palavra NÃO. Em geral, as letras S ou N são suficientes. Um outro exemplo como este é o que dá ao operador um leque de opções. A mensagem:

QUE ITEM DEVE SER ALTERADO?

pode permitir que o operador digite o código que identifica a entrada desejada.

Os programas que controlam este tipo de diálogo devem ser escritos como sub-rotinas independentes que não assumem nada, nem dependem de conhecimento do programa que as está chamando. Isso tem três aplicações:

1. Não se pode considerar que o local em que a mensagem será impressa esteja em branco. Se não estiver, a mensagem será colocada em cima da anterior. Porém se existirem caracteres não cobertos pela mensagem, eles ficarão lá. Isso pode atrapalhar o operador e levá-lo a erro. A sub-rotina seguinte limpa o número de espaços determinado pelo valor da variável ER:

```
5000 REM  LIMPA ESPACOS
5010 FOR I = 1 TO ER:PRINT " ":NEXT I:RETURN
```

2. Uma sub-rotina deve receber informação do programa que a chamou. Por exemplo, se a sub-rotina pede ao operador um número, então todo programa que a utilize deve especificar os valores máximo e mínimo aceitáveis na digitação.
3. A sub-rotina deve devolver a resposta do operador ao programa origem. Pode ser um caractere (como S ou N), uma palavra (como SIM ou NÃO) ou um número.

A lógica da sub-rotina não consegue deduzir em que parte da tela a mensagem aparecerá. O mais simples, no caso, é que o programa origem posicione o cursor antes de chamar a sub-rotina.

Veja agora a sub-rotina necessária para fazer uma pergunta que retorna um S para SIM e um N para NÃO. Usaremos a declaração PRINT para fazer a pergunta, seguida de INPUT para receber a resposta de um caractere. Limparemos parte da tela com a sub-rotina acima. Aqui está o programa e as sub-rotinas:

```
100 REM MOVE O CURSOR
140 VTAB 1
150 REM CHAMA SUBROTINA PARA PEDIR RESPOSTA
160 GOSUB 3020
170 END
3000 REM ++ PEGA A RESPOSTA S/N ++
3010 REM
3020 C=POS(0):REM GUARDA COLUNA DO CURSOR
3030 R=PEEK(37):REM GUARDA LINHA DO CURSOR
3040 REM LIMPA ESPACOS PARA A PERGUNTA
3050 ER=35:REM LIMPA 35 ESPACOS
3060 GOSUB 5010
3070 HTAB C+1:REM REPOSICAO DO CURSOR
3080 PRINT "QUER FAZER ALGUMA ALTERACAO"
3090 INPUT R$
3100 IF R$="S" OR R$="N" THEN RETURN
3110 REM RESPOSTA IMPROPRIA
3120 VTAB R+1:REM REPOSICIONA CURSOR
3130 GOTO 3050:REM TENTA NOVAMENTE
5000 REM ++ LIMPA ESPACOS ++
5010 FOR I=1 TO ER:PRINT " ":NEXT I:RETURN
```

Veja agora o diálogo que permite ao operador a entrada de um número. Vamos criar uma sub-rotina que aceita resposta não menor que o valor da variável MI e não maior que a variável MA. Aqui está o programa necessário:

```
100 REM DA LIMITES E POSICAO DO CURSOR
140 MI=1:MA=10
150 VTAB 1
160 REM CHAMA SUBROTINA PARA RESPOSTA
170 GOSUB 3500
180 END
3500 REM ++ PEGA RESPOSTA NUMERICA ++
```

```
3510 REM ++ MI<RESPOSTA<MA ++
3520 REM ++ NM E A RESPOSTA ++
3530 GOSUB 5110:REM ONDE ESTA O CURSOR
3540 REM LIMPA ESPACO PARA PERGUNTA
3550 ER=35:REM LIMPA 35 ESPACOS
3560 GOSUB 5010
3570 HTAB C+1:REM REPOSICIONA CURSOR
3580 PRINT "QUE CAMPO DEVE SER MUDADO ?";
3590 INPUT NM
3600 IF NM>MI AND NM<MA THEN RETURN
3610 REM RESPOSTA IMPROPRIA
3620 VTAB L+1:REM REPOSICIONA CURSOR
3630 GOTO 3550: REM TENTA DE NOVO
5000 REM ++ LIMPA ESPACOS ++
5010 FOR I=1 TO ER:;PRINT " ";NEXT I:RETURN
5100 REM ++GUARDA POS.ATUAL DO CURSOR ++
5110 C= PEEK(36):REM COLUNA
5120 L= PEEK(37):REM LINHA
5130 RETURN
```

Pode-se modificar a sub-rotina de forma que ela aceite entradas de dois caracteres? Tente modificar o programa por sua conta para fazer isso, então espere irmos um pouco mais adiante neste capítulo, onde se encontra a sub-rotina necessária no programa que controla entrada de dados.

Existe outra modificação simples em ambos os diálogos que mostramos. As mensagens de orientação podem ser fornecidas pelo programa que chamou a sub-rotina. Isso a torna de aplicação muito mais genérica. Tente fazer essas alterações no programa.

Detecção de Erros e Controle

Se for desejado fazer um programa de boa qualidade deve-se realizar um grande esforço para prever a possibilidade de ocorrência de erros que o usuário do programa possa fazer. O programa deve achar erros e forçar o operador a redigitar os valores que fariam o computador operar de forma anormal (estourar, na linguagem "computês").

É verdade que o Apple II acha alguns tipos de erros para nós. Ele não aceita entrada alfabética quando espera dados numéricos numa declaração do tipo INPUT A. Se for tentado colocar letras neste caso, é dada uma mensagem de erro e se pede para redigitar o valor.

A capacidade de detecção de erros embutida no sistema é limitada. É bem possível digitar um tipo de dado correto (ou seja, cordão ou numérico) com um valor inaceitável. Isto é, o valor pode causar um erro no programa num ponto posterior. Aqui está um pequeno programa que demonstra o problema:

```
100 INPUT X
200 PRINT 100/X
300 END
```

Se for digitado 0 em resposta à declaração INPUT, o programa estoura quando tentar dividir por 0 na declaração PRINT. É bastante fácil evitar isso. As linhas seguintes testam a entrada para ter ciência que o valor não é 0, e pedirão outro número se for.

```
110 IF X<>0 THEN 200
120 PRINT "NAO PERMITIDO,DIGITE DE NOVO"
130 GOTO 100
```

Estendendo os princípios mostrados neste exemplo, pode-se ver o quanto fácil é testar um valor entrado para manter os limites possíveis. Dependendo das circunstâncias, pode ser melhor testar os limites com o ON-GOTO do Applesoft ou o ON-GOSUB (GOTO e GOSUB calculados do Integer BASIC), ao invés da série de declarações IF-THEN. Existe um exemplo de detecção de erros mais elaborado na próxima seção.

O ONERR GOTO e a Declaração RESUME

O Applesoft tem uma declaração especial que permite contornar erros quando os encontra — antes que seja colocada uma mensagem ou pare a execução do programa. Aqui está um exemplo:

```
50    ONNER      GOTO 8000
```

Uma vez dada esta declaração, quando o Applesoft encontrar um erro, pulará para o endereço 8000. Também será mostrado um código numérico descrevendo o erro na posição de memória 222, que poderá ser consultada através de uma declaração PEEK. A Tabela C.1 no Apêndice C tem a relação de condições de erros detectáveis pelo ONNER GOTO.

A forma usual de tratar os erros com o ONNER GOTO é escrever uma rotina para onde o programa vá quando ocorra um erro. No final dessa rotina, a declaração RESUME causa o retorno para o início da declaração onde ocorreu o erro. Alternativamente, uma declaração GOTO mandará o programa para qualquer endereço. Escreva a rotina de erro de forma que ela tome atitudes diferentes dependendo da natureza do erro e do estado presente do programa, o que pode eventualmente ser verificado pela análise dos valores das variáveis-chave.

Para cancelar o ONNER GOTO e fazer o Apple II voltar à sua forma normal de tratamento de erro, use a declaração POKE 216,0.

O programa seguinte demonstra o uso do ONNER GOTO. Neste programa, quaisquer erros que não possam ocorrer como resultado de operação de entrada via teclado são tratados como fatais, com o aparecimento de uma mensagem de advertência. A entrada é desconsiderada e é requisitada nova digitação.

```
50  ONNER GOTO 8000
200  PRINT "ENTRE UM VALOR ALFANUMERICO"
210  INPUT X$
220  PRINT "ENTRE UM VALOR NUMERICO"
230  INPUT X
240  PRINT "ENTRE UM VALOR INTEIRO"
250  INPUT X%
260  GOTO 200
500  REM FIM DO PROGRAMA
```

```
510  PRINT "ULTIMA ENTRADA FOI ";X$,", ";X;" E ";X%
515  POKE 216,0:REM DESLIGUE POR ERRO
520  END
8000  REM ++ ROTINA DE ERRO ++
8010  E = PEEK (222):REM PEGA NUMERO DO ERRO
8020  IF E = 255 THEN GOTO 500:REM FIM DE PROGRAMA
      COM CTRL-C
8030  IF E = 53 OR E = 176 OR E = 254 THEN 8100
8035  INVERSE
8040  REM DETECTADO ERRO DE PROGRAMACAO
8050  PRINT "ARRGH! DETECTADO ERRO NUMERO ";E
8055  PRINT "DEIXE O NUMERO NA TELA"
8060  PRINT "E A DESCRICAO DO OCORRIDO."
8070  PRINT "CHAME UM PROGRAMADOR PARA AUXILIA-LO."
8080  PRINT "DEIXE O COMPUTADOR LIGADO."
8090  NORMAL : STOP
8100  REM DETECTADO ERRO DE ENTRADA DE DADO
8110  PRINT "";:REM CARFACTERE CTRL-G ENTRE ASPAS
8120  PRINT "ERRO, TENTE NOVAMENTE"
8130  RESUME
```

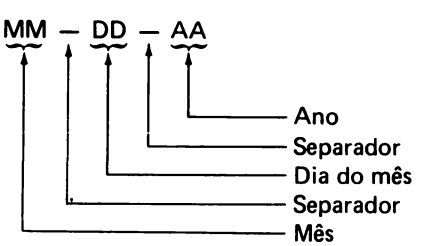
Entrando uma Data Válida

Nesta parte descreveremos um programa que usa várias técnicas apresentadas neste capítulo. São usadas algumas funções do BASIC e declarações disponíveis apenas no Applesoft. O programa pode ser escrito em Integer BASIC; pode-se fazer a conversão.

Muitos programas, em algum ponto de sua execução precisam de entrada de dados não tão simples quanto um sim ou não, porém não tão complexas quanto uma tela inteira. Considere uma data.

Deve-se ter mais cuidado com essa entrada de dados simples do que à primeira vista parece necessário. E muito provável que a data seja apenas um item dentro de uma seqüência. Tomando o devido cuidado com cada um dos itens numa digitação, pode-se evitar a digitação de muitos itens sempre que o operador se esqueça de um dado.

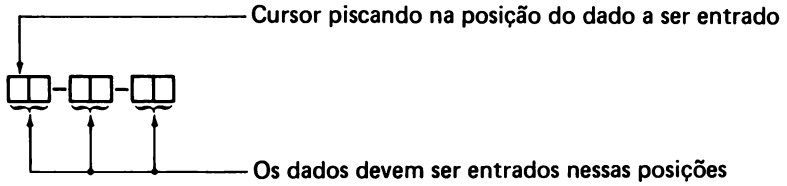
Assume-se que a data seja colocada na seguinte forma:



O mês, dia do mês e ano são, cada um, entrados como sendo dois dígitos numéricos, colocados sem a terminação RETURN.

O programa fornece os traços de separação entre os elementos. Dependendo de sua preferência, pode ser uma barra ou outro caractere desejado. Em muitas partes do mundo o dia precede o mês.

Programe a entrada de dados de forma que fique agradável ao operador. Ele deve reconhecer facilmente onde o dado deve ser colocado, que tipo de dado é e o que vem depois. Uma forma interessante de mostrar onde o dado deve ser colocado é mostrar o nome do campo em vídeo reverso. Por exemplo, o programa que pede a data pode ser criado da seguinte forma com vídeo reverso:



Pode-se criar o visual deste programa como segue:

```
10  HOME: VTAB 3: HTAB 20: REM POSICAO PARA ENTRADA
20  IW = 2: GOSUB 1100: REM CAMPO DE ENTRADA DE 2 DIGITOS
30  PRINT "-";
40  GOSUB 1100: REM CAMPO DE ENTRADA DE 2 CARACTERES
50  PRINT "-";
60  GOSUB 1100: REM CAMPO DE ENTRADA DE 2 CARACTERES
70  VTAB 3: HTAB 20: REM REPOSICAO PARA INICIO DO
    CAMPO DE ENTRADA
80  END
1090  REM ++ MOSTRA "IW" ESPACOS EM BRANCO ++
1100  INVERSE
1110  FOR I = 1 TO IW: PRINT " ";: NEXT I
1120  NORMAL
1130  RETURN
```

O programa acima inclui declarações que posicionam a digitação da data na coluna 21 e linha 3. Também é limpa a tela de forma que qualquer bobagem anterior não atrapalhe a introdução do dado. Após a apresentação do dado entrado, o cursor se move de volta para a posição do primeiro campo de entrada, embora isso não possa ser apreciado devido à declaração END.

Tente usar a declaração INPUT na linha 80 para receber o primeiro item da data: o mês. Isso pode ser feito como segue:

```
80  INPUT M$
90  END
```

Digite as declarações nas linhas 80 e 90 como acima e execute o programa. Como se vê, a declaração INPUT não opera. Fora o fato do ponto de interrogação deslocar o primeiro caractere

do campo de digitação, o acionamento do RETURN faz limpar todo o resto da linha de digitação. Esta é a ocasião de usar a declaração GET. Coloque as seguintes linhas adicionais de programa:

```
80  GOSUB 1200: MM$   C$: REM PEGA PRIMEIRO DIGITO
    DO MES
90  GOSUB 1200: MM$ = MM$ + C$: REM PEGA SEGUNDO
    DIGITO DO MES
1190  REM ++ ACEITA UM CARACTERE DIGITADO
1200  GET C$
1210  PRINT C$;: REM APRESENTA A DIGITACAO NO VIDEO
1220  RETURN
```

Essas declarações aceitam a digitação de dois caracteres. A entrada é mostrada no primeiro campo reverso da data. A entrada dos dois dígitos não pede RETURN ou outro terminador qualquer. O programa pára se pedir dados após a entrada do segundo caractere.

São necessárias três entradas de dois caracteres cada: para o mês, dia e ano. Ao invés de repetir as declarações das linhas 80 e 90, colocaremos estas declarações numa sub-rotina e iremos a ela três vezes, como segue:

```
80  GOSUB 1300: MM$ = CC$: REM PEGA MES
90  PRINT "-";: REM ESPACO PARA O PROXIMO CAMPO
180  GOSUB 1300: DD$ = CC$: REM PEGA DIA
190  PRINT "-";: REM ESPACO PARA O PROXIMO CAMPO
280  GOSUB 1300: AA$ = CC$: REM PEGA ANO
290  END
1290  REM ++ PEGA ENTRADA DE 2 CARACTERES ++
1300  GOSUB 1200: CC$ = CC$ + C$: REM PEGA SEGUNDO
      CARACTERE
1320  RETURN
```

As variáveis MM$, DD$ e AA$ têm o mês, dia e ano, respectivamente. Cada entrada tem dois caracteres.

Existem duas formas pelas quais se pode ajudar o operador a se recuperar de erros durante a digitação.

1. O programa pode testar automaticamente a validade do mês, dia e ano.
2. Pode ser dada ao operador a chance de redigitar estes itens.

O programa pode testar o mês entre 1 e 12. Os anos bissextos não precisam ser considerados, porém pode-se testar a existência de um dia num dado mês. Qualquer ano entre 00 e 99 será permitido. Qualquer digitação inválida fará o programa requisitar todos os itens novamente.

Se o operador acionar a tecla RETURN, toda a seqüência reinicia. Nossa versão final do programa de entrada de dados ficará assim:

```
10  HOME : VTAB 3: HTAB 20 : REM POSICIONA PARA DIGITACAO
15  RC$ = CHR$ (13): REM PERMITE REINICIO DE DIGITACAO
17  REM
18  REM MOSTRA 3 CAMPOS DE 2 CARACTERES CADA
19  REM
20  IW = 2: GOSUB 1100
30  PRINT "-"
40  GOSUB 1100
50  PRINT "-"
60  GOSUB 1100
65  PRINT "";: REM CTRL-G = BELL
67  REM
68  REM PEGA ENTRADA DE 3 CAMPOS
69  REM
70  VTAB 3: HTAB 3: REM POSICIONA NA INICIO DOS CAMPOS
    DE ENTRADA
79  REM
80  REM PEGA MES
81  REM
90  GOSUB 1300: IF C$ = RC$ THEN GOTO 10: REM TESTE PARA
    REINICIO DE ENTRADA
100  MM = VAL (CC$): REM MES EM NUMERICO
110  IF MM < 1 OR MM > 12 THEN 10: REM TESTE DE MES
     INEXISTENTE
120  DT$ = CC$: PRINT "-";: REM AVANCA PARA PROXIMO CAMPO
125  REM VE O MAIOR NUMERO DE DIAS DESTE MES
130  DM = 31: REM ASSUME 31 DIAS
135  REM A MENOS DE FEVEREIRO
140  IF MM = 2 THEN DM = 29
150  REM OU ABRIL, JUNHO, SETEMBRO OU NOVEMBRO
160  IF MM = 4 OU MM = 6 OR MM = 9 OR MM = 11 THEN DM = 30
169  REM
170  REM PEGA DIA
171  REM
180  GOSUB 1300: IF C$ = RC$ THEN GOTO 10 : REM TESTE PARA
     REINICIO DE DIGITACAO
190  DD = VAL (CC$): IF DD < 1 OR DD > DM THEN 10: REM
     REINICIO SE ENTRADA FOI INVALIDA
200  DT$ = DT$ + "-" + CC$: PRINT "-": REM AVANCA PARA O
     PROXIMO CAMPO
269  REM
270  REM PEGA ANO
271  REM
280  GOSUB 1300: IF C$ = RC$ THEN GOTO 10: REM TESTE PARA
     REINICIO DA DIGITACAO
```

```
290   YY = VAL (CC$): IF YY < 0 OR YY > 99 THEN 10: REM
      REINICIO PARA ENTRADA INVALIDA
300   DT$ = DT$ + "-" + CC$
389   REM
390   REM MOSTRA A ENTRADA
391   REM
400   VTAB (10): HTAB (18): PRINT "DATA DIGITADA:"
410   VTAB (11): HTAB (18): PRINT DT$
420   END
1089   REM
1090   REM ++ MOSTRA "IW" CARACTERES EM BRANCO REVERSO ++
1091   REM
1100   INVERSE
1110   FOR I = 1 TO IW: PRINT " ";: NEXT I
1120   NORMAL
1130   RETURN
1189   REM
1190   REM  ++ ACEITA ENTRADA DE 1 CARACTERE ++
1191   REM
1200   GET C$: IF C$ = "" THEN 1200
1210   IF C$ = RC$ THEN RETURN: REM TESTE PARA REINICIO
1220   IF C$ < "0" OR C$ > "9" THEN 1200
1230   PRINT C$;: REM SAI CARACTERE
1240   RETURN
1289   REM
1290   REM ++ PEGA ENTRADA DE 2 CARACTERES ++
1291   REM
1300   REM PEGA PRIMEIRO CARACTERE; TESTA REINICIO
1310   GOSUB 1200: IF C$ = RC$ THEN RETURN
1315   CC$ =C$
1320   REM PEGA SEGUNDO CARACTERE; TESTA REINICIO
1330   GOSUB 1200: IF C$ = RC$ THEN RETURN
1340   RETURN
```

Note que a data ficou presente num campo série de 8 caracteres, o DT$, após a digitação do mês, dia e ano.

São feitas três verificações nos dados no momento da digitação:

1. O caractere é *carriage return*?
2. Se não, é um caractere válido?
3. A combinação de 2 caracteres é um mês válido para a primeira digitação, um dia válido para a segunda e um ano válido para a terceira?

Selecionamos o *carriage return* para caractere de reinício. Trocando o CHR$(13) na linha 15, pode-se selecionar qualquer outro caractere que se queira. Quando o operador aciona a tecla esco-

lhida, toda a seqüência de entrada é reiniciada. Deve-se perguntar pelo caractere de reinício a cada um digitado, na rotina apropriada (na linha 1200) e depois na sub-rotina de digitação de dois caracteres (na linha 1300) desde que desejamos estar aptos a voltar ao início da digitação do primeiro ou segundo caractere. O programa principal também verifica se o caractere de reinício foi digitado para voltar para a linha 10 e reiniciar toda a entrada de dados. Poderíamos também pular diretamente da rotina de entrada de um caractere para a linha 10, eliminando assim outros testes. Porém não é uma boa prática sair de uma sub-rotina com um GOTO ao invés de um RETURN. Toda a sub-rotina deve ser tratada como um módulo lógico, com parâmetros específicos de entrada e retornos padronizados. Sair de uma sub-rotina de qualquer outra forma conduz facilmente a programas difíceis de entender e a erros de programação. (Lembre-se que antes de se sair de uma sub-rotina sem o RETURN, deve-se dar um POP para limpar o endereço de retorno que ficou guardado.)

A lógica do programa que testa os caracteres não numéricos ficou inteiramente localizada dentro da digitação de um caractere. Ignoramos dígitos não numéricos. A declaração que faz isso está na linha 1220.

A lógica para teste de mês, dia e ano válidos deve estar contida dentro do programa principal (fora da sub-rotina) desde que cada uma dessas entradas tenha um limite permitido diferente.

A declaração da linha 110 testa o mês válido.

As declarações das linhas 130, 140 e 160 verificam o máximo dia permitido para o mês entrado. A declaração da linha 190 testa um dia válido.

O teste de validade do ano é bem simples, e está na linha 290.

Leva-se mais tempo para escrever um programa com entrada de dados que mostrem informações de forma mais concisa e testa-se a validade dos parâmetros, permitindo ao operador a redigitação a qualquer momento. Será que esse tempo é bem gasto? Muito bem gasto. O programa será escrito uma só vez; já o operador processará a rotina centenas ou milhares de vezes. Dessa forma será gasto um tempo extra por uma vez, para que o operador não tenha centenas ou milhares de atrasos.

FORMAS DE ENTRADA DE DADOS

A seção seguinte descreve algumas técnicas de programação que são melhor implementadas no Applesoft. Muitos dos efeitos especiais usados neste exemplo de programa são difíceis, ou mesmo impossíveis, de se obter com o Integer BASIC. Se não houver outra alternativa além dele, ainda poderá ser interessante a leitura desta seção para o aproveitamento eventual de algumas técnicas com o Integer BASIC.

A melhor forma de se entrar muitos itens num programa é apresentar um formulário na tela, e preenchê-lo com os dados entrados. Considere um nome e endereço. Primeiro apresente um formulário como abaixo:

```
ENTRE NOME E ENDEREÇO ABAIXO
 1 NOME:
 2 ENDEREÇO:
 3 CIDADE:
 4 ESTADO:                          5 CEP:
```

Veja que a cada item está associado um número. O número do campo aparece em vídeo reverso neste formulário.

O operador entra os dados seqüencialmente, começando com o item 1 e terminando com o 5. O operador pode então alterar qualquer campo.

O programa seguinte limpa a tela e mostra o formulário inicial:

```
109  REM
110  REM MOSTRA O FORMULARIO DE ENTRADA DE DADOS
111  REM
120  CALL -936: VTAB 2: REM LIMPA A TELA E
     POSICIONA O CURSOR
125  PRINT "ENTRE O NOME E O ENDERECO ABAIXO"
130  REM PRIMEIRO MOSTRA OS NUMEROS DOS CAMPOS"
140  INVERSE
150  FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160  VTAB 6: HTAB 29: PRINT 5
170  NORMAL
180  REM MOSTRA OS NOMES DOS CAMPOS
190  VTAB 3: HTAB 6: PRINT "NOME:"
200  VTAB 4: PRINT "ENDERECO:"
210  HTAB 6: PRINT "CIDADE:"
220  HTAB 5: PRINT "ESTADO:"
230  HTAB 31: PRINT "CEP:"
```

Serão criados campos de vídeo reverso para informar onde os dados entrados devem ser colocados à medida que vão sendo digitados. O CTRL-X é usado para reiniciar a entrada de dados no campo corrente. A tecla RETURN termina a digitação do campo. A seqüência de instruções seguinte fornece a lógica de programa necessária:

```
100  RC$ = CHR$ (24): REM CTRL-X COMO CARACTERE DE REINICIO
200  REM
300  REM ENTRA TODOS OS 5 CAMPOS
301  REM
310  FOR I = 1 TO 5: GOSUB 1900: NEXT F
320  END
990  REM ++++++++ SUBROTINA 1000 ++++++++++++
991  REM ENTRA SERIE DE DADOS NO CAMPO COM LN CARACTERES
992  REM O CURSOR DEVE ESTAR NA PRIMEIRA POSICAO DO CAMPO
993  REM A TECLA RETURN TERMINA O CAMPO
994  REM A TECLA "FLEXA A ESQUERDA" REINICIA A ENTRADA
995  REM NAO TESTA A VALIDADE DO DADO ENTRADO
996  REM A SERIE ENTRADA VOLTA EM CC$
997  REM
1000  HT = POS (0) + 1: REM LEMBRA POSICAO DO INICIO DO CAMPO
1010  REM MARCARA COM VIDEO REVERSO
1020  INVERSE
1030  FOR I = 1 TO N: PRINT " ";: NEXT I
1040  NORMAL: HTAB (HT): REM REPOSICIONA NO INICIO DO CAMPO
1050  REM ENTRA DADO
```

```
1060  CC$ = "": REM INICIA ENTRADA COM NULO
1070  GET C$
1080  IF C$ = RC$ THEN HTAB (HT): GOTO 1020: REM REINICIA ENTRADA?
1090  IF C$ = CHR$ (13) THEN GOTO 1140: REM FIM DE ENTRADA?
1100  REM QUANDO ENTRADA TERMINA, ESPERA POR RETURN OU REINICIO
1110  IF LEN (CC$) = LN THEN GOTO 1070
1120  PRINT C$;: REM SAI CARACTERE TECLADO
1130  CC$ = CC$ + C$: GOTO 1070
1140  J = LEN (CC$)
1150  FOR I = J TO LN: CC$ = CC$ + " ": NEXT I
1160  REM RECOLOCA A ENTRADA
1170  HTAB (HT): PRINT CC$;: RETURN
1889  REM ++++++++++ SUBROTINA 1900 +++++++++++++++
1890  REM PULA PARA ROTINA DE ENTRADA PARA CAMPO NUMERO F
1891  REM
1900  ON F GOTO 2000,2100,2200,2300,2400:RETURN
1989  REM
1990  REM ENTRAM 20 CARACTERES -NOME-
1991  REM
2000  VTAB 3: HTAB 11
2010  LN = 20: GOSUB 1000: NA$ = CC$: RETURN
2089  REM
2090  REM ENTRAM 20  CARACTERES - ENDERECO -
2091  REM
2100  VTAB 4: HTAB 11
2110  LN = 20: GOSUB 1000:CI$= CC$: RETURN
2189  REM
2190  REM ENTRAM 20 CARACTERES - CIDADE -
2191  REM
2200  VTAB 5: HTAB 11
2210  LN = 20: GOSUB 1000: RETURN
2289  REM
2290  REM ENTRAM 18 CARACTERES - ESTADO -
2291  REM
2300  VTAB 6 : HTAB 11
2310  LN = 18: GOSUB 1000: ST$ = CC$: RETURN
2389  REM
2390  REM ENTRAM 5 CARACTERES - CEP -
2391  REM
```

Digite o programa inteiro da declaração 100 até 2410 e execute. Lembre-se de que ainda existem as declarações de 109 a 230 digitadas do exemplo anterior, de forma que não é necessário recolocar.

Se o programa não rodar direito, confira a listagem com cuidado. Em particular as declarações PRINT.

Na execução do programa, cada um dos cinco campos vai ficar destacado por sua vez. Assim que os caracteres forem digitados eles serão transferidos para o vídeo. Ao se acionar o RETURN, todo o campo será substituído pelo conteúdo final teclado. Tente o uso de CTRL-X para reiniciar a entrada de dados.

Analise com cuidado a lógica da sub-rotina de entrada de cordão, iniciando na linha 1000 e indo até a 1170. Antes de seguir adiante sugere-se que se entenda perfeitamente a lógica embutida nesta parte.

Veja como é fácil entender o que se está digitando, e como é fácil reiniciar a digitação para corrigir erros.

Depois da digitação completa de nome e endereço, o programa deveria perguntar ao operador se algum campo deve ser alterado; depois deveria pedir qual o campo. Uma rotina de pergunta SIM-NÃO já foi vista antes neste capítulo. Usaremos uma versão modificada dela, onde o programa que chama fornece a pergunta a ser feita ao operador. Aqui está a listagem completa do programa adicionando os comentários necessários.

```
9  REM ************************************************************
10  REM ESTE PROGRAMA MOSTRA UM FORMULARIO PARA ENTRAR NOME E ENDERECO
11  REM E PEDE EM SEGUIDA A ENTRADA DESSES DADOS
12  REM ************************************************************
13  REM
100  RC$ = CHR$ (24) : REM CTRL-X = CARACTERE DE REINICIO
109  REM
110  REM MOSTRA O FORMULARIO DE ENTRADA DE DADOS
111  REM
120  CALL -936: VTAB 2: REM LIMPA A TELA E POSICIONA O CURSOR
125  PRINT "ENTRE O NOME E O ENDERECO ABAIXO"
130  REM PRIMEIRO MOSTRA OS NUMEROS DOS CAMPOS"
140  INVERSE
150  FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160  VTAB 6: HTAB 29: PRINT 5
170  NORMAL
180  REM MOSTRA OS NOMES DOS CAMPOS
190  VTAB 3: HTAB 6: PRINT "NOME:"
200  VTAB 4: PRINT "ENDERECO:"
210  HTAB 6: PRINT "CIDADE:"
220  HTAB 5: PRINT "ESTADO:"
230  HTAB 31: PRINT "CEP:"
299  REM
300  REM ENTRA TODOS OS 5 CAMPOS
301  REM
310  FOR I = 1 TO 5: GOSUB 1900: NEXT F
320  END
990  REM ++++++++ SUBROTINA 1000 ++++++++++++
991  REM ENTRA SERIE DE DADOS NO CAMPO COM LN CARACTERES
992  REM O CURSOR DEVE ESTAR NA PRIMEIRA POSICAO DO CAMPO
993  REM A TECLA RETURN TERMINA O CAMPO
994  REM A TECLA "FLEXA A ESQUERDA" REINICIA A ENTRADA
995  REM NAO TESTA A VALIDADE DO DADO ENTRADO
996  REM A SERIE ENTRADA VOLTA EM CC$
997  REM
1000  HT = POS (0) + 1: REM LEMBRA POSICAO DO INICIO DO CAMPO
1010  REM MARCARA COM VIDEO REVERSO
1020  INVERSE
1030  FOR I = 1 TO N: PRINT " ";: NEXT I
1040  NORMAL: HTAB (HT): REM REPOSICIONA NO INICIO DO CAMPO
1050  REM ENTRA DADO
1060  CC$ = "": REM INICIA ENTRADA COM NULO
1070  GET C$
1080  IF C$ = RC$ THEN HTAB (HT): GOTO 1020: REM REINICIA ENTRADA?
1090  IF C$ = CHR$ (13) THEN GOTO 1140: REM FIM DE ENTRADA?
1100  REM QUANDO ENTRADA TERMINA, ESPERA POR RETURN OU REINICIO
1110  IF LEN (CC$) = LN THEN GOTO 1070
```

```
1120  PRINT C$;: REM SAI CARACTERE TECLADO
1130  CC$ = CC$ + C$: GOTO 1070
1140  J = LEN (CC$)
1150  FOR I = J TO LN: CC$ = CC$ + " ": NEXT I
1160  REM RECOLOCA A ENTRADA
1170  HTAB (HT): PRINT CC$;: RETURN
1189  REM ++++++++++++ SUBROTINA 1200 +++++++++++++++
1190  REM LIMPA A LINHA ONDE ESTA O CURSOR
1191  REM
1200  HTAB = 1: REM COMECA NO INICIO DA LINHA
1210  FOR I = 1 TO 39: PRINT " ";: NEXT I
1220  HTAB 1: REM DEIXA CURSOR NO INICIO DA LINHA
1230  RETURN
1289  REM ++++++++++ SUBROTINA 1300 ++++++++++++++++
1290  REM FAZ UMA PERGUNTA (QU$) E RETORNA RESPOSTA S OU N EM YN$
1291  REM
1300  GOSUB 1200: REM LIMPA TODA A LINHA
1310  PRINT QU$;: REM MOSTRA A PERGUNTA
1320  GET YN$: IF YN$< >"N" AND YN$ < >"Y" THEN GOTO 1320
1330  PRINT YN$;: REM MOSTRA RESPOSTA
1340  RETURN
1389  REM ++++++++++ SUBROTINA 1400 ++++++++++++++++
1390  REM PEDE ENTRADA NUMERICA (PERGUNTA QU$)
1391  REM RESPOSTA VEM EM NM
1392  REM NM DEVE SER <=HI E >=LO
1393  REM
1400  GOSUB 1200: REM LIMPA LINHA DE ENTRADA
1410  PRINT QU$;: REM MOSTRA PERGUNTA
1420  GET C$: NM = VAL (C$)
1425  REM VE SE ESTA DENTRO DA FAIXA
1430  IF NM < LO OR NM > HI THEN GOTO 1420
1440  PRINT C$;: REM SAI RESPOSTA
1450  RETURN
1889  REM ++++++++++++ SUBROTINA 1900 +++++++++++++++
1890  REM PULA PARA ROTINA DE ENTRADA PARA CAMPO NUMERO F
1891  REM
1900  ON F GOTO 2000,2100,2200,2300,2400:RETURN
1989  REM
1990  REM ENTRAM 20 CARACTERES -NOME-
1991  REM
2000  VTAB 3: HTAB 11
2010  LN = 20: GOSUB 1000: NA$ = CC$: RETURN
2089  REM
2090  REM ENTRAM 20  CARACTERES - ENDERECO -
2091  REM
2100  VTAB 4: HTAB 11
2110  LN = 20: GOSUB 1000:CI$= CC$: RETURN
2189  REM
2190  REM ENTRAM 20 CARACTERES - CIDADE -
2191  REM
2200  VTAB 5: HTAB 11.
2210  LN = 20: GOSUB 1000: RETURN
2289  REM
2290  REM ENTRAM 18 CARACTERES - ESTADO -
2291  REM
2300  VTAB 6: HTAB 11
2310  LN = 18: GOSUB 1000: ST$ = CC$: RETURN
2389  REM
2390  REM ENTRAM 5 CARACTERES - CEP -
2391  REM
2400  VTAB 6: HTAB 35
2410  LN = 5: GOSUB 1000: ZP = CC$: RETURN
```

Estude o programa acima cuidadosamente e entenda a forma como foi feito e as observações inseridas. São elas:

1. Numerar cada campo e justapor uma máscara de vídeo reverso no momento apropriado, isso deixa saliente o campo a ser digitado e o espaço disponível.
2. Quando o operador entra com o número do campo para ser modificado, a máscara também informa rapidamente se foi dado o número de campo correto.
3. O operador não precisa preencher todos os caracteres de cada campo; ao acionar a tecla RETURN, o resto do campo é preenchido com espaços.
4. A qualquer momento pode-se reiniciar a digitação com a tecla CTRL-X.
5. Quando são feitas perguntas, apenas são aceitas respostas com significado, tipo S ou N para "sim" ou "não" ou um número entre 1 e 5 para especificar um campo. Não é bom permitir-se a entrada de respostas sem significado. Por exemplo, aceitar a resposta S para "sim" e qualquer outra tecla para "não" poderia, por acidente, provocar a saída do programa de forma desastrosa. O contrário, aceitar N para "não" e qualquer tecla para "sim" poderia muitas vezes obrigar o operador a redigitar dados desnecessariamente, apenas por ter acionado uma tecla errada.

Abaixo vão outros aperfeiçoamentos que não foram colocados, mas poderiam sê-lo:

1. Testar o código de CEP para não haver entrada nula (existem códigos similares em muitos países para permitir entrada alfabética).
2. Os programadores mais cautelosos ainda fazem a pergunta VOCÊ TEM CERTEZA? Quando o operador responde NÃO à pergunta QUER FAZER ALGUMA MODIFICAÇÃO? Isso dá ao operador uma segunda chance de redigitar os dados no caso de ter digitado uma tecla errada.
3. Uma tecla que aborte a entrada atual de dado e devolva o valor anterior. Por exemplo, se o operador alterou o campo errado, o programa do exemplo faz com que ele seja forçado a redigitar o campo inteiro. Ele poderia facilmente reconhecer uma tecla que elimine a entrada e devolva o conteúdo anterior.
4. Permitir o acionamento da tecla ← como retorno de espaço. Cada vez que a tecla ← for acionada, o cursor volta um espaço e o último caractere entrado é substituído pela máscara em vídeo reverso na tela e por um caractere em branco na rotina de saída de CC$. Claro, não pode haver o retorno de espaço no primeiro caractere de esquerda do campo.

Tente modificar o programa exemplo para implementar as melhorias acima.

FORMATAÇÃO DE SAÍDA

Quando se liga o Apple II, automaticamente a saída é a tela. Existem declarações que permitem mandar dados para outros dispositivos tal como impressora e outros que aceitem dados de saída.

Existem algumas diferenças na programação de saída pela tela em comparação com impressora. Por exemplo, a impressora pode ser mais larga que a tela, de forma que uma linha de saída

caiba dentro de uma linha da impressora e pule para a linha de baixo na tela. Por outro lado, o HTAB e o VTAB (TAB no Integer BASIC) podem ser usados para mover o cursor pela tela, porém não podem ser usados para mover o cursor pelo espaço de uma folha de papel.

Existem também muitas similaridades nas técnicas de programação usadas para criar saídas na tela e na impressora. Muito do que será visto adiante se aplica a ambas. Aquilo que se aplicar apenas a vídeo será devidamente indicado. Se houver interesse em programar saídas por impressora, devem ser lidas as orientações específicas, no final deste capítulo.

Programar a saída de dados é mais simples que a entrada, desde que não é necessário se preocupar com a interação com o operador. Deve-se tornar a informação fácil de ser usada, e nada mais. Aqui estão algumas poucas regras para seguir:

1. Evite concentrar muita informação dentro de um espaço pequeno.
2. Se números ou letras forem listados em colunas, alinhe os dados de forma que o olho possa varrer a coluna com facilidade.
3. Use vídeo reverso para salientar informações chaves, cabeçalhos em cima e ao lado (apenas para vídeo).

Abaixo estão dicas que ajudarão a evitar erros desnecessários na programação de saída de dados:

1. Lembre-se de seguir cada item numa declaração PRINT com um ponto-e-vírgula, a menos que se deseja espaçamento em colunas. Este é o erro mais comum nas declarações de saída.
2. Antes de fazer qualquer coisa, projete sua tela ou relatório. Use uma folha de papel para gráficos ou um formulário feito especialmente para projetos de relatório. O Apêndice L tem uma tela que permite calcular linhas e colunas com precisão. A alternativa para isso é o método de tentativa e erro, que no fim faz perder muito mais tempo que se o formulário for projetado antecipadamente.
3. Procure subscrever arranjos que não se dividam em colunas pares. Por exemplo, suponha que se tem 25 itens num arranjo N$(I) que serão colocados em três colunas. Pode-se ser tentado a fazer algo como abaixo:

```
100 FOR I = 1 TO 25 STEP 3
200 REM PROCESSO DA COLUNA 1
 .
 .
 .
300 REM PROCESSO DA COLUNA 2
 .
 .
 .
400 REM PROCESSO DA COLUNA 3
 .
 .
 .
500 NEXT I
```

Mas na passagem final do loop FOR-NEXT, os índices 26 e 27 serão computados, embora não existam. Pode-se facilmente testar o fim da matriz num loop FOR-NEXT, como abaixo:

```
100 FOR I = 1 TO MX STEP ST
 .
 .
 .
350 I = I + 1
360 IF I > MX THEN 510
 .
 .
 .
500 NEXT
510 REM CONTINUA O PROGRAMA
```

A Tela Como Janela de Dados

Quando se lida com grande quantidade de dados, uma técnica muito comum é a de se usar a tela como uma janela de dados. Durante todo o tempo o vídeo mostra apenas uma parte dos dados existentes. Uma forma de fazer isso é agrupar os dados em páginas, cada uma cabendo na tela. Os programas que usam essa técnica devem ter rotinas especiais para mostrar o cabeçalho e os valores dos dados em cada página. Deve haver também alguma forma do programa mudar de uma página para a outra.

É muito comum as matrizes guardarem grandes quantidades de dados. Nesse caso, pode-se usar a tela como o visor de uma câmara. Imagine que a matriz de dados é escrita numa grande área e pode-se observar esta área através de uma câmara. A área é tão grande que não se pode observar todo o espaço pela câmara de uma só vez, porém pode-se ver partes pelo movimento da câmara para cima, baixo, direita e esquerda. A tela pode imitar isso.

Vamos ilustrar o uso da tela como janela de dados criando um exemplo no Applesoft da janela acima descrita. De início, vamos criar uma matriz inteira em duas dimensões. Como valor para cada elemento da matriz, associaremos números de 4 dígitos que identifica os índices, como segue:

$$X\%(i,j) = 0i0j$$

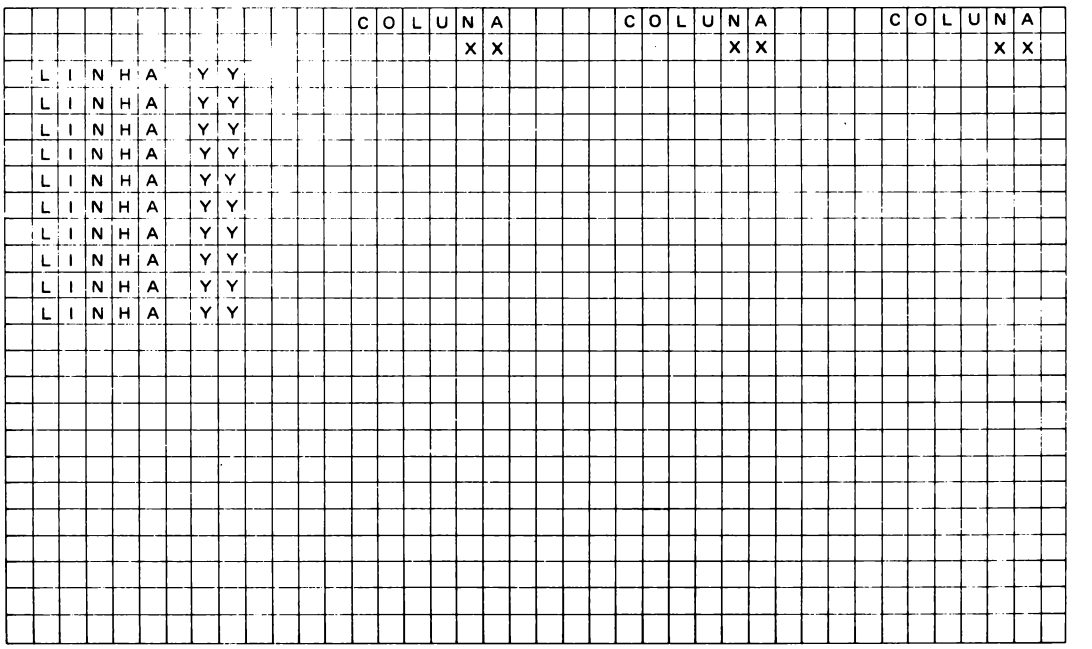
Por exemplo:

$$X\%(3,2) = 0302$$
$$X\%(19,8) = 1908$$
$$X\%(11,12) = 1112$$

Pode-se inicializar esta matriz inteira de forma simples, como segue:

```
10  DIM X%(14,50)
20  FOR I = 1 TO 14
30  FOR J = 1 TO 50
40  X%(I,J) = I * 100 + J
50  NEXT J
60  NEXT I
```

Agora vamos mostrar uma parte desta matriz. Usaremos as duas linhas superiores da tela para cabeçalho e os primeiros nove espaços de cada linha para cabeçalho lateral, como abaixo:



O número atual da coluna vai aparecer onde se vê o XX acima. O número atual da linha vai aparecer onde se vê YY. Aqui está o programa necessário para criar cabeçalhos de linhas e colunas em vídeo reverso:

```
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + I * 10: PRINT "COLUNA";
1040  NEXT I
1050  PRINT
1060  FOR I = 0 TO 2
1065  REM 1 ESPACO EXTRA ANTES DO PRIMEIRO DIGITO.
1070  S% = 0: IF C$ + I < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT SPC(S%);C% + I;
1090  NEXT I
```

```
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM 1 ESPACO EXTRA ANTES DO PRIMEIRO DIGITOS.
1120  S% = 0: IF I < 10 THEN S% = 1
1130  HTAB 2: PRINT "LINHA";: HTAB 8: PRINT SPC(S%);I
1140  NEXT I
1150  NORMAL: RETURN
```

Criamos deliberadamente uma janela que é menor do que a tela de forma a ficar melhor ilustrado o conceito de janela de dados. Não há nada que impeça que se crie uma janela que ocupe toda a tela; porém haverá ocasiões em que se deseje uma janela pequena de forma que outros dados apareçam concorrentemente no vídeo.

Adicionaremos agora instruções para pedir ao operador dois números representando a menor linha e a menor coluna da matriz. O elemento da matriz com esta linha e coluna aparecerá no alto à esquerda da tela. Os elementos adjacentes a este aparecerão de forma a preencher toda a janela. Aqui está o programa inteiro:

```
5   REM JANELA NUM PROGRAMA DE MOSTRAR TABELA
6   REM ************************************************
10   HOME: PRINT "ESPERE, PROCESSO DE INICIALIZACAO";
20   DIM X%(14,50)
30   FOR I = 1 TO 14
40   FOR J = 1 TO 50
50   X%(I,J) = I * 100 + J
60   NEXT J
70   NEXT I
75   HOME
80   HTAB 1: VTAB 20: INPUT "ENTRE COLUNA (1 A 12):";C%
90   IF C% < 1 OR C%   12 THEN GOTO 80
100   VTAB 21: INPUT "ENTRE LINHA (1 A 41):";R%
110   IF R% < 1 OR R% > 41 THEN GOTO 100
120   VTAB 1: HTAB 1: GOSUB 1000: REM MOSTRA CABECALHOS
130   REM PREENCHE VALORES DA JANELA
135   VTAB 3
140   FOR I=1 TO R% + 9
150   HTAB 10
160   FOR J = C% TO C% + 2
165   REM MOSTRA VALORES DA JANELA ALINHADOS A DIREITA
170   X$ = STR$(X%(J,I)): PRINT SPC(10-LEN(X$));X$;
180   NEXT J
190   PRINT: REM PROXIMA LINHA
200   NEXT I
210   VTAB 22: PRINT "CONTINUA? ENTRA S OU N:";
220   GET C$: IF C$ <> "S" AND C$ <> "N" THEN 220
230   IF C$ = "S" THEN GOTO 80
240   END
```

```
990  REM
991  REM ++++++++++++ SUBROTINA 1000 +++++++++++++++
992  REM SAI CABECALHOS DE LINHAS E COLUNAS
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + I * 10: PRINT "COLUNA";
1040  NEXT I
1050  PRINT
1060  FOR I = 0 TO 2
1065  REM 1 ESPACO EXTRA ANTES DOS NUMEROS DE 1 DIGITO
1070  S% = 0: IF C% + 1 < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT SPC(S%);C% + I;
1090  NEXT I
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM 1 ESPACO EXTRA ANTES DOS NUMEROS DE 1 DIGITO
1120  S% = 0: IF I < 10 THEN S% = 1
1130  HTAB 2: PRINT "LINHA";: HTAB 8: PRINT SPC(S%);I
1140  NEXT I
1150  NORMAL: RETURN
```

Coloque este programa no computador e execute-o. Se tudo for executado corretamente, a primeira coisa que acontecerá é o computador parar por instantes; ele estará rodando a rotina FOR-NEXT aninhada das linhas 30 até 70. São gastos de 5 a 10 segundos para preencher a matriz X% com números. O programa dá uma mensagem de advertência sobre isso. Sem tal mensagem, o operador poderia pensar que o computador não está operando. É sempre bom dar tais mensagens quando o sistema fica parado por um intervalo de tempo considerável.

Veja que são permitidas as colunas de número 1 até 12. Existem três colunas, embora qualquer coluna de número maior que 12 ficará dentro do arranjo de 14 colunas. Linhas de número 1 a 41 são permitidas, desde que o número de linhas iniciando em 41 atingiriam 50, que é a outra dimensão da matriz.

O valor inteiro da matriz X% é convertido para série na linha 170, antes de ser mostrado. Fizemos a conversão para simplificar a formatação de saída. Assim fica fácil de calcular o número de espaços entre colunas, como mostrado pela declaração PRINT na linha 170. Não é tão fácil alinhar valores numéricos corretamente quando são impressos diretamente. Para ver isso, troque o conteúdo da linha 170 por:

```
170 PRINT SPC (7); X% (J, I);
```

Os números serão alinhados não permitindo mostrar números de quatro dígitos — tornando o comando de impressão muito grande para uma tela de 40 caracteres.

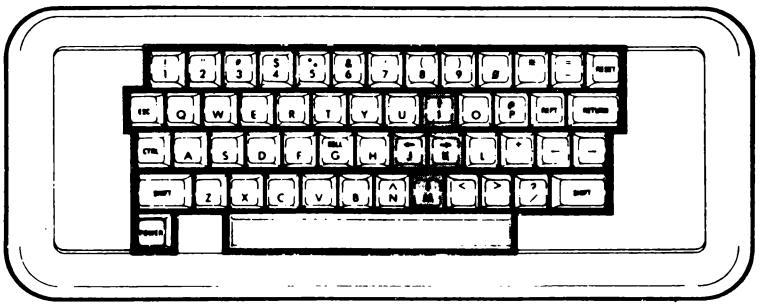
Nosso programa toma bastante cuidado para terminar a saída na 39ª coluna do vídeo, ao invés de na 40ª e última. Se forem colocados caracteres na 40ª coluna, aciona-se o mecanismo automático de salto de linha para continuar na próxima. Deve-se evitar o conflito entre os retornos devidos ao processo automático e os devido ao programa.

Como exercício, seria interessante modificar o programa de mostrar a tabela de forma a

fazer com que passe da 40ª coluna. Para fazer isso deve-se trocar a tabulação da linha 150 de 10 para 11, na linha 1030 de 4+I*10 para 5+I*10, na linha 1080 de 18+I*10 para 19+I*10 e finalmente na linha 1130 de 4 e 5 para 5 e 9. Tente rodar agora o programa; as colunas de números aumentaram, porém há muitos retornos e os números das colunas no cabeçalho do alto são cobertos pela primeira linha dos valores da matriz. Veja se é possível eliminar os retornos extras e gerar a tela corretamente. Essa não é uma tarefa fácil.

Veja que as declarações que pedem dados nas linhas 80, 100 e 220 são seguidas por passos de programa que impossibilitam digitação de dados inválidos. Mesmo num programa de demonstração perdemos tempo para criarmos rotinas de segurança na entrada de dados.

Um refinamento útil para o programa é a possibilidade de dar ao operador a chance de mover a janela para cima e para baixo, ou para a direita e esquerda. Isso é fácil. Usaremos as teclas I, J, K e M para direcionar o controle de forma semelhante àquela usada na edição (descrita no Capítulo 3). A tecla I move uma linha para cima. M desce uma linha, J move uma coluna para a esquerda e K uma coluna para a direita como mostrado a seguir:



Para fazer esta tarefa precisamos trocar as linhas 210 a 240 com as seguintes declarações:

```
210  VTAB 22: PRINT "CONTINUA?"
215  PRINT "ENTRE DIRECAO (I,J,K,M),S OU N ";
220  GET C$
225  REM ABAIXO UMA LINHA ?
230  IF R% > 1 THEN IF C$ = "M" THEN R% = R% - 1: GOTO 120
235  REM SOBE UMA LINHA ?
240  IF R% < 41 THEN IF C$ = "J" THEN C% = C% - 1: GOTO 120
245  REM ESQUERDA UMA COLUNA ?
250  IF C%   1 THEN IF C$ = "J" THEN C% = C% - 1: GOTO 120
255  REM DIREITA UMA COLUNA ?
260  IF C% < 12 THEN IF C% = "K" THEN C% = C% + 1: GOTO 120
270  IF C$ = "S" THEN GOTO 80: REM ENTRA NOVA LINHA E COLUNA
280  IF C$ = "N" THEN END
285  REM TOCA O SINO (BELL) E REJEITA OUTRA LETRA
290  PRINT CHR$(7);: GOTO 220
```

Veja como a lógica é direta, mesmo testando erros de digitação. Qualquer digitação diferente da permitida é rejeitada, e o caractere de controle direcional é rejeitado se mover a janela além dos limites da matriz.

PROGRAMANDO IMPRESSORAS

O Apple II trata a impressora como substituta da tela de vídeo. Para criar saída por impressora, devem-se dar declarações de programa que desviem a saída do vídeo para a impressora. Os comandos devem trazer de volta a saída para o vídeo, quando for terminada a impressão. Isso é feito usando a declaração PR#.

As impressoras se ligam ao computador por meio de uma interface serial ou paralela, dependendo do modelo usado.

Normalmente o cartão de interface serial é conectado ao slot 1 do Apple II enquanto a interface paralela é colocada no slot 2. Mas isso é mais convenção do que necessidade. De fato, os cartões de interface serial e paralela podem ser inseridos em qualquer um dos slots (exceto o slot 0).

Saindo Texto para a Impressora

Lembre-se que o PR# é considerado uma declaração do DOS sempre que o DOS está presente. Isso significa que ele deve ser impresso com um caractere pré-fixado, o CTRL-D (código 4 em ASCII). O programa abaixo imprime duas linhas de texto numa impressora conectada no slot 1 do Apple II com o DOS presente:

```
 10  REM MANDA DUAS LINHAS DE TEXTO PARA IMPRESSORA
 20  REM CRIA O CARACTERE CTRL-D
 30  D$ = ""
 40  REM SELECTA O PORT SERIAL DE E/S
 50  PRINT D$;"PR#1"
 60  PRINT "O APPLE ESCREVE PARA A TELA OU IMPRESSORA"
 70  PRINT "E EM CADA CASO SAI UM BYTE POR VEZ"
 80  REM DESCONECTA A IMPRESSORA
 90  PRINT D$;"PR#0"
100  END
```

A declaração na linha 30 cria o caractere de controle que converte o comando PR# na declaração BASIC. A declaração PR# aparece nas linhas 50 e 90. A declaração na linha 50 transfere a saída da tela para a impressora, enquanto que a da linha 90 transfere de volta para a tela. A declaração PRINT das linhas 60 e 70 manda duas linhas de texto para a impressora. Aqui está o resultado obtido quando se roda o programa:

```
O APPLE ESCREVE PARA A TELA OU IMPRESSORA
E EM CADA CASO SAI UM BYTE POR VEZ
```

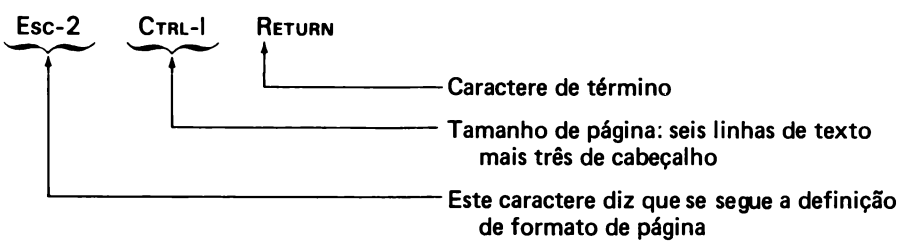
Aqui está o mesmo programa para o Apple II sem o DOS:

```
 10  REM SAIDA DE DUAS LINHAS DE TEXTO PARA IMPRESSORA
 40  REM SELECIONA O PORT DE E/S
 50  PRINT "PR#1"
 60  PRINT "O APPLE ESCREVE PARA A TELA OU IMPRESSORA"
 70  PRINT "E EM CADA CASO SAI UM BYTE POR VEZ"
 80  REM DESELECTA A IMPRESSORA
 90  PRINT "PR#0"
100  END
```

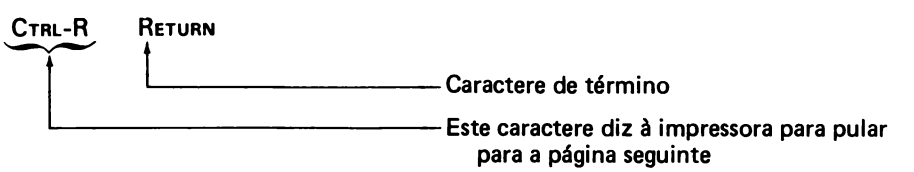
Impressoras Programáveis

Muitas impressoras permitem que a saída seja formatada sob controle do programa. Inserindo os caracteres de controle apropriados no texto que sai pela impressora, pode-se ajustar a largura de linha, o conjunto de caracteres, o tamanho da página e uma grande variedade de outras características.

Muitas impressoras populares são comumente usadas com o Apple II. Sob o ponto de vista de programação, entretanto, o Apple II trata todas as impressoras de forma idêntica. Por exemplo, suponha a impressora da Texas Instruments modelo 810 conectada a um computador Apple II por meio de uma interface serial conectada ao slot 1. O manual da Texas diz que podemos determinar o tamanho da página para seis linhas em cada uma com a seguinte seqüência de caracteres:



Além disso, pode-se forçar a impressora a pular para o início da página seguinte (esta operação é conhecida em computação como *line-feed*) com a seguinte seqüência de caracteres:



Modificando o nosso exemplo anterior, a listagem para o Applesoft abaixo faz a impressão das mesmas duas linhas 15 vezes, em blocos de 6 linhas. Este programa deve ser usado com a presença do DOS.

```
 10  REM SAIDA DE TEXTO PARA A IMPRESSORA
 11  REM USANDO 6 LINHAS POR PAGINA E 5 PAGINAS
 20  REM CRIAR O CARACTERE CTRL-D
 30  D$ = ""
 40  REM SELECIONA O PORT DE E/S SERIAL
 50  PRINT D$;"PR#1"
 51  REM DETERMINA 6 LINHAS POR PAGINA
 52  PRINT CHR$ (27);"2"; CHR$ (6)
 53  PRINT CHR$ (12);: REM SAI FORM FEED PARA POSICIONAR
     PARA O INICIO DA IMPRESSAO
 54  FOR I = 1 TO 15
 60  PRINT "O APPLE ESCREVE PARA A TELA OU IMPRESSORA"
 70  PRINT "E EM CADA CASO SAI UM BYTE POR VEZ"
 75  NEXT I
 80  REM DESCONECTA A IMPRESSORA
 90  PRINT D$;"PR#0"
100  END
```

No programa acima, a declaração da linha 52 seleciona seis linhas por página. A função CHR$(27) representa o caractere ESC. O CHR$(9) define a página como sendo seis linhas mais três de separação entre o fim de uma página e o início da outra. Os pontos-e-vírgulas concatenam todos os caracteres de forma a dar a seqüência requerida pela impressora.

A declaração da linha 53 executa o salto para o início de uma página. O CHR$(12) executa a função *form feed*. Um ponto-e-vírgula segue o caractere de *form feed* porque sem ele a declaração PRINT produziria um *carriage return*, resultando numa linha em branco. Sem o ponto-e-vírgula, entretanto, a primeira página teria uma linha em branco e cinco linhas impressas, enquanto as demais teriam seis linhas impressas por página.

Qualquer outra impressora programável pode ser conduzida transferindo-se para ela os *códigos de controle* da mesma forma.

Este programa não opera em Integer BASIC porque usa a função CHR$. No entanto, pode-se gerar qualquer caractere ou seqüência de caracteres acionando combinações de teclas. Pode-se trocar o CHR$ por um par de aspas fechando um caractere que não é impresso. Por exemplo, o valor gerado pela digitação (com três teclas) do "Esc" é o mesmo que CHR$(27). Veja o Apêndice I onde há uma listagem completa de tais equivalências.

Imprimindo Listagens de Programa

Digitando-se o comando LIST no teclado, qualquer programa do Apple II pode ser listado na tela do vídeo. Para *imprimir* a listagem, entretanto, deve-se dar os comandos PR# apropriados antes e depois de dar o comando LIST. Assumindo que a impressora está conectada a uma interface no slot 1, aqui estão os passos necessários:

1. Certifique-se qùe o programa a ser listado está na memória do Apple II.
2. Com o cursor numa linha em branco, selecione a impressora com o comando PR#1. O

cursor voltará para o primeiro caractere da linha corrente porém sem passar para a linha de baixo.

3. Digite LIST. Este comando não aparecerá na tela, porém será colocado na impressora onde o programa também deverá aparecer listado depois.
4. Quando o programa inteiro estiver listado, devolva a impressão para o vídeo com a digitação de PR#0. Este comando não aparecerá no vídeo; deverá aparecer na impressora.

GUARDANDO DADOS EM CASSETTE

Aprendemos no último capítulo como guardar e recuperar programas guardados numa fita cassette. No Applesoft, pode-se também guardar valores numéricos ou matrizes inteiras na fita.

A instrução STORE guarda valores de matrizes e a instrução RECALL os lê de volta. Nenhum dos dois controla os movimentos do gravador, nem apresenta diretivas dizendo ao operador quando operar os botões do gravador cassette. O programa abaixo demonstra o uso do STORE e RECALL. Ele atribui valores a uma matriz numérica, guarda numa fita cassette, coloca os valores da matriz em zero e, finalmente, relê os valores da fita. Os valores da matriz são mostrados em pontos-chave para documentar as mudanças que vão ocorrendo no decorrer do programa.

```
10  REM ESTE PROGRAMA DEMONSTRA O STORE E O RECALL
20  REM ************************************************
30  DIM A(10)
40  HOME
50  PRINT TAB (4);"GUARDOU; TAB (13);"LIMPOU"; TAB (22);
    "RELEU"
60  REM INICIALIZA VALORES DO ARRANJO
70  FOR I = 1 TO 10: A(I) = I: NEXT I
80  T = 8: GOSUB 1000: REM MOSTRA VALORES A GUARDAR
90  VTAB 20: HTAB 1
100  PRINT "COLOQUE O CASSETTE NO GRAVADOR.REBOBINE"
110  PRINT "ACIONE OS BOTOES 'RECORD' E 'PLAY'"
120  INPUT "E DIGITE 'VAI' ";C$
130  IF C$ < > "VAI" GOTO 90
140  STORE A
160  CLEAR: REM COLOCA VALORES DO ARRANJO A ZERO
170  VTAB 2: T = 18: GOSUB 1000: REM MOSTRA ARRANJO LIMPO
180  VTAB 20: HTAB 1
190  GOSUB 1100: REM LIMPA ULTIMAS INSTRUCOES
200  VTAB 20: HTAB 1
210  PRINT "REBOBINE A FITA. ACIONE 'PLAY',"
220  INPUT "E DIGITE 'VAI'";C$
230  IF C$ < > "VAI" GOTO 200
240  RECALL A
```

```
260  VTAB 2: T = 28: GOSUB 1000: REM MOSTRA VALORES LIDOS
270  GOSUB 1100: REM LIMPA ULTIMAS INSTRUCOES
280  VTAB 20: HTAB 1
290  PRINT "ACIONE 'STOP'."
300  END
990  REM ++++++++++++++ SUBROTINA 1000 ++++++++++++++++++
1000  FOR I = 1 TO 10: HTAB T: PRINT A(I): NEXT I: RETURN
1090  REM ++++++++++++++ SUBROTINA 1100 ++++++++++++++++++
1100  FOR I = 1 TO 119: PRINT " ";: NEXT I: RETURN
```

Pode-se guardar uma matriz sob o nome de uma variável e chamá-lo usando outro nome. Mas, de forma geral, a dimensão de uma matriz gravada deve ser a mesma da matriz usada na leitura. Existem alguns casos excepcionais mais complicados que são vistos no Capítulo 8. A menos que haja intenção de obter algum efeito especial, use o RECALL para os dados gravados no cassette com uma matriz das mesmas dimensões daquela utilizada no STORE.

OTIMIZAÇÃO DE PROGRAMAS

Tradicionalmente, o programa ótimo é aquele que, para uma dada tarefa, opera mais rapidamente e usa menos memória. Claramente estes dois objetivos devem ser balanceados de forma que o programa resultante seja eficiente, fácil de escrever, fácil de usar e fácil de modificar. Haverá mais benefícios ao longo do tempo em se direcionar a programação sobre esses aspectos do que ficar praticando malabarismos para otimizar a velocidade e a utilização da memória. Em tempo, se já houver o conhecimento de como otimizar a velocidade de programas e a utilização da memória, deve-se inicialmente escrever programas que sejam eficientes e que não precisem ajustes finos depois de estarem rodando. Neste espírito, apresentamos algumas formas de escrever programas que são mais rápidas e usam menos memória.

Algumas das técnicas para fazer um programa rodar mais rápido fazem-no consumir mais espaço, enquanto outras técnicas que economizam espaço em memória aumentam o tempo de execução. Deve-se decidir o que é mais importante em cada programa individualmente.

PROGRAMAS RÁPIDOS

Evite usar constantes (ou seja, 0, 100, "Y", "ENTRE"). No lugar delas, associe primeiramente em seu programa o valor da constante na variável. Use então a variável onde se usaria a constante. Isso é especialmente importante quando se usam constantes inteiras repetidamente em expressões reais. Leva muito tempo para se converter uma constante para um valor real porque é preciso varrer a tabela de variáveis. E quando tal conversão acontece dentro de um loop FOR-NEXT, uma sub-rotina muito usada ou uma função definida pelo usuário, as diferenças se tornam muito mais significativas. Esta técnica tem como benefício adicional o fato de tornar o programa fácil de ser alterado. Se for necessária sempre a troca de uma constante, será mais fácil modificar aquela dentro de uma declaração de atribuição, do que procurar e trocar cada ocorrência da constante ao longo do programa.

Use aquelas variáveis mais freqüentemente usadas no programa tão próximas da *execução* quanto for possível. A alocação de variáveis se dá de forma que a primeira variável que chega é alocada, primeiro, e, na medida em que elas chegam, vão sendo alocadas em seqüência. O BASIC encontra uma variável no início da lista mais rapidamente que uma no fim dela.

Quando o BASIC encontra uma declaração de salto para uma outra linha, ele começa a procurar o endereço a partir do início do programa e vai pesquisando até encontrá-lo. Claramente, quanto menor for o número da linha em relação às demais linhas do programa, tanto mais rápido o BASIC executa a instrução. Por isso, em seus programas, dê endereços baixos para as sub-rotinas de uso mais freqüente.

No Applesoft, não inclua índices na declaração NEXT. Dessa forma, evita-se a necessidade do programa conferir se o índice foi especificado corretamente ou não.

PROGRAMAS COMPACTOS

Use sub-rotinas para evitar duplicação de programa para a mesma lógica. Essa também será uma forma de aumentar a legibilidade, eficiência e facilidade de alteração de seu programa.

Use os elementos de índice 0 dos arranjos (ex. X(0), B(0)).

Existem menos caracteres num nome de variável pequena que numa constante de vários dígitos. Por isso atribua nomes de variáveis a constantes e use esses nomes ao invés de especificar os valores reais.

Ponha mais de uma declaração numa mesma linha. Cada nova linha de programa usa cinco bytes extras para seu número. Observe, entretanto, que as linhas de programa compostas de mais de uma declaração são mais difíceis de editar e também de ler e entender. Conceber como o programa deve operar pela primeira vez é difícil, pior ainda é ter que fazer isso muitas vezes, mais tarde.

Use declarações REM com critério; abrevie os comentários. Mas seja cuidadoso; quanto menos comentários seu programa tiver, mais difícil será destrinchá-lo mais tarde.

Seja econômico com o uso de variáveis. Cada uma delas requer uma certa quantidade de memória, mesmo se for usada apenas uma vez. Crie um sistema de criação de nomes de variáveis, que tenha algumas do tipo *rascunho*, que podem ser usadas dentro de loops FOR-NEXT, cálculos intermediários, e coisas do tipo. Porém não se exceda; o programa ficará mais fácil de ser entendido se os nomes forem escolhidos com critério e significado. Crie padrões de identificação para cada variável (por exemplo, NC$ para nome do cliente) e para tipos de variáveis (por exemplo, todas as variáveis de rascunho iniciam com a letra X).

Use declaração INPUT e arquivos de dados (se tiver um disco, veja o Capítulo 5) em vez de declarações de atribuição e declaração DATA.

No Applesoft, use matrizes inteiras no lugar de matrizes reais. Cada valor na matriz inteira usa dois bytes de memória, enquanto cada valor na matriz real usa cinco bytes. Use a função FRE periodicamente no seu programa para limpar áreas de armazenamento de séries na memória.

DEPURAÇÃO

Um programa novo nunca funciona segundo se espera que ele funcione. Mesmo se não houver erros de sintaxe BASIC, é comum o aparecimento de erros na lógica usada. Em "computês"

chama-se a qualquer um dos dois tipos de erro de *bug*. O processo para detecção e correção é conhecido como *debugging*. Existem várias formas de se depurar (ou fazer o *debugging*) do programa.

Existe uma forma de conduta sempre recomendável: tome seu tempo, planeje o que fazer, faça-o corretamente na primeira vez. Não se sente em frente ao teclado com um mero esboço do que deve ser feito, entrando com o programa "na raça". Se lhe faltar experiência em programação, complemente esta literatura com uma de diretivas do BASIC listadas no Apêndice K para ter algumas noções práticas de como se deve programar bem.

Se seu programa foi escrito e não está operando como foi imaginado que fizesse, existem algumas declarações do BASIC que podem ser usadas para ajudar na depuração.

A Declaração PRINT

Surpreendentemente, a velha e conhecida declaração PRINT é uma excelente ferramenta de depuração. Pode-se acrescentar algumas declarações PRINT de forma temporária em pontos estratégicos do programa para mostrar mensagens que informem se o programa atingiu determinados pontos sem falha, e informar valores intermediários de variáveis. Essa é uma forma de acompanhar o fluxo da execução do programa e verificar os resultados de cálculos de valores intermediários.

A Declaração TRACE

A declaração TRACE, como seu nome indica, traça o fluxo da execução do programa mostrando o número da linha de cada declaração que executa. Para ver como ela funciona, digite o programa abaixo, digitando em seguida o comando TRACE seguido por RUN.

```
100  PRINT "ENTRE UM NUMERO DE 1 A 5 (6 PARA FIM)":
110  INPUT N
120  IF N = 1 THEN PRINT "ONE"
130  IF N = 2 THEN PRINT "TWO"
140  IF N = 3 THEN PRINT "THREE"
150  IF N = 4 THEN PRINT "FOUR"
160  IF N = 5 THEN PRINT "FIVE"
170  IF N > 5 THEN GOTO 100
180  FOR I = 1 TO N
190  PRINT "*";: REM IMPRIME N ASTERISCOS
200  NEXT I
210  CALL -936: REM LIMPA A TELA
220  GOTO 100
```

Para cancelar o modo TRACE e retornar ao normal, execute o comando NO TRACE.

A Declaração DSP

Existe outra declaração de depuração útil disponível apenas no Integer BASIC: a declaração DSP. Aqui está um exemplo:

```
]10  DSP  CONTA
```

Uma vez que essa declaração DSP particular tenha sido executada, o Apple II informará cada vez que o valor da variável CONTA mudar, e em que linha ocorreu tal mudança. Como o comando RUN desliga as declarações DSP dadas previamente, deve-se dar GOTO para iniciar o programa, ou colocar o DSP como declaração dentro do programa.

Para se desativar o modo DSP para uma variável deve-se dar a declaração NODSP. Aqui está um exemplo:

```
]300  NODSP NOME$
```

Uma vez executada esta declaração, o Apple II não mais notifica cada alteração da variável NOME$.

Tente adicionar as linhas seguintes no exemplo usado para o TRACE para observar o efeito do DSP. Embora se possa usar ambos simultaneamente, tente rodar o exemplo sem o TRACE para ver mais claramente os efeitos do DSP.

```
10 DSPN
20 DSPI
100  PRINT "ENTRE UM NUMERO DE 1 A 5 (6 PARA FIM)";
110  INPUT N
120  IF N   1 THEN PRINT "ONE"
130  IF N = 2 THEN PRINT "TWO"
140  IF N   3 THEN PRINT "THREE"
150  IF N = 4 THEN PRINT "FOUR"
160  IF N = 5 THEN PRINT "FIVE"
170  IF N ) 5 THEN GOTO 100
180  FOR I = 1 TO N
190  PRINT "*";: REM IMPRIME N ASTERISCOS
200  NEXT I
215 NODSP
210  CALL -936: REM LIMPA A TELA
220  GOTO 100
```

RESTRIÇÕES NO MODO IMEDIATO E PROGRAMADO

Pode-se usar a maioria das declarações BASIC em ambos os modos. Porém há declarações que são legais apenas em modo programado, e outras apenas em modo imediato. A Tabela 4.2 lista as declarações restritas para o Integer BASIC. A Tabela 4.3 lista as declarações restritas para o Applesoft.

TABELA 4.2. Declarações Restritas a Modo Programado ou Imediato no Integer BASIC.

Modo Programado	Modo Imediato
END	AUTO
FOR	CLR
GOSUB	CON
INPUT	DEL
NEXT	HIMEM:
RETURN	LOAD
	LOMEM:
	MAN
	NEW
	RUN
	SAVE

TABELA 4.3. Declarações Restritas a Modo Programado no Applesoft.

Apenas em Modo Programado
DATA
DEF FN
GET
INPUT
ON ERR GOTO
RESUME

5

O Disk II

O drive de disco é um dos mais importantes componentes de um sistema de computação. Ele permite acesso instantâneo a qualquer item de uma longa lista de informações. O disco do Apple II pode guardar mais de 143.000 caracteres de dados num único disquette. Isso é quase três vezes o que se pode guardar em 48K de RAM e, quando o computador é desligado, toda a informação presente no disco permanece intacta.

SOBRE DISCOS

Os discos guardam informação magneticamente, da mesma forma que uma fita de gravador. A maior diferença é que o disco é redondo e roda como um disco de música. Dentro do drive existe uma *cabeça* que lê e grava informação. O computador pode mover a cabeça para qualquer parte da superfície do disco. Essa habilidade permite o *acesso aleatório.* Por isso, pode-se dizer que o disco é um *dispositivo de armazenamento de acesso aleatório.* Um programa especial, chamado *Sistema Operacional do Disco* (*Disk Operating System* ou simplesmente *DOS*), controla toda a operação do disco. Existem vários tipos diferentes de discos.

Discos Rígidos

Os discos rígidos, ou duros, são cobertos com uma substância magnética. Eles armazenam tipicamente de 5 a 10 megabytes de dados (um megabyte é equivalente a um milhão de bytes). Muitos discos rígidos são removíveis; isto é, o disco e o drive podem-se separar, de forma que se pode trocar de disco. Discos rígidos custam cerca de 150 dólares cada, os drives custam de 3 a 10 mil dólares. A Figura 5.1 mostra um sistema típico de disco rígido.

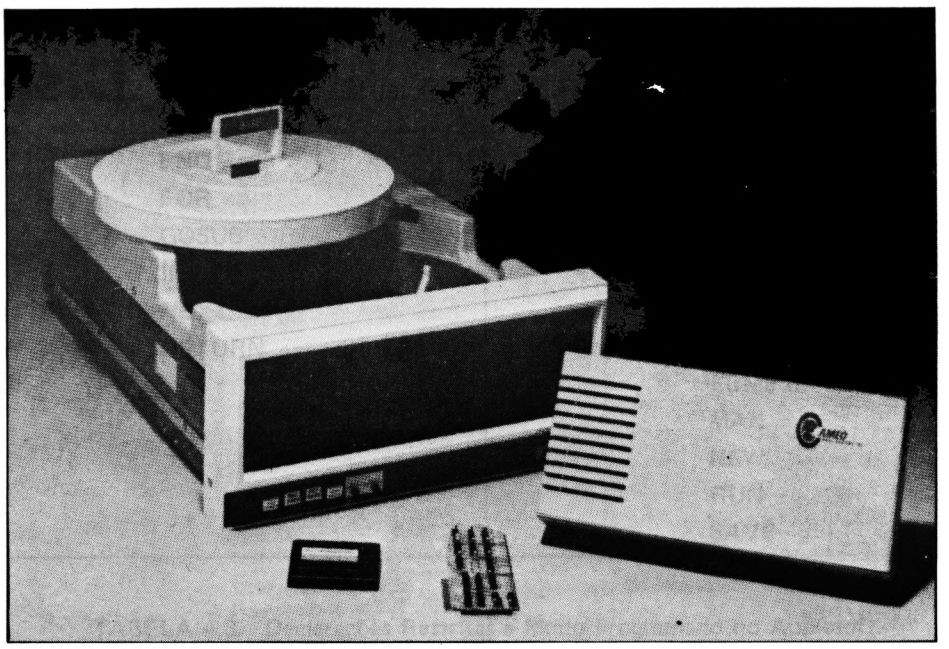


FIGURA 5.1. Sistema Típico com Disco Rígido. (Cortesia de Cameo Electronics.)

Discos Winchester

Os drives de disco Winchester, mostrados na Figura 5.2, usam uma tecnologia especial que permite de seis a dez vezes mais dados em cada disco. Os discos Winchester são extremamente susceptíveis a sujeira e pó — mesmo fumaça de cigarros. E por terem a necessidade de permanecerem sempre limpos, são selados dentro do drive e não podem ser tocados. Um sistema de disco Winchester custa de 2.500 a 8.000 dólares.

Disquettes

Os disquettes são o tipo mais popular de disco. Cada um deles consiste num disco de vinil circular fechado num envelope plástico. O envelope protege o disquette contra danos durante o manuseio normal. O disquette roda livre dentro do envelope. As aberturas do envelope possibilitam acesso da cabeça à superfície do disco e fornecem uma área onde o drive pode prender e rodar o disquette. Ele nunca deve ser retirado de seu envelope. A Figura 5.3 mostra como é o disquette fora de seu envelope.

Os disquettes, também conhecidos como *floppy discs*, são comuns em dois tamanhos: 8 e 5 ¼ de polegadas de diâmetro, ambos ilustrados na Figura 5.4. O Disk II do Apple usa disquettes de 5 ¼ de polegadas, que também são chamados minidiscos ou minidisquettes. O Disk II pode guardar 143.360 bytes em cada disquette.

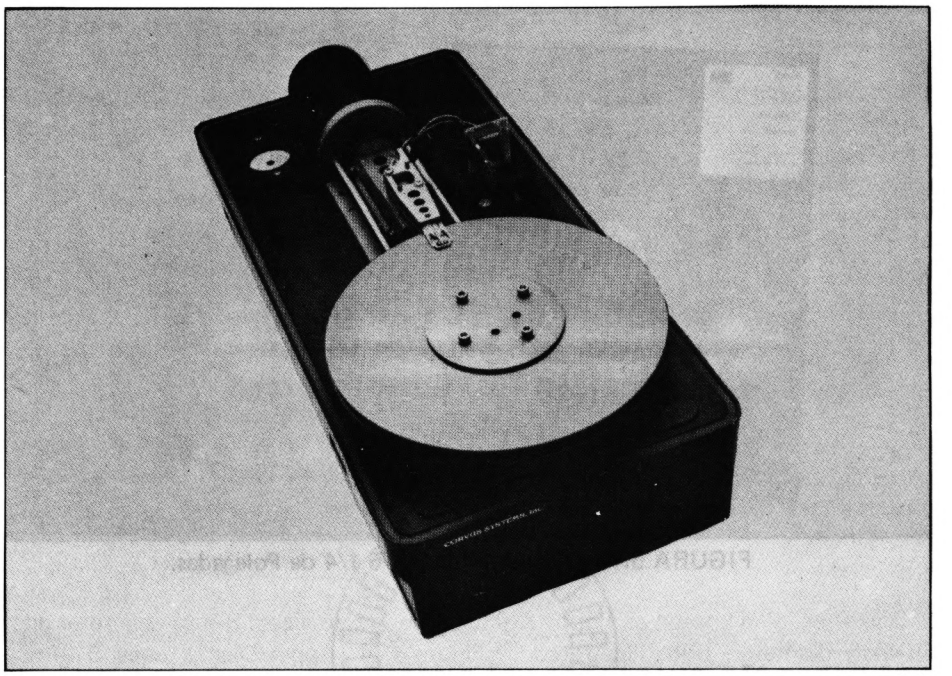


FIGURA 5.2. Drive de Disco Winchester. (Cortesia de Corvus Systems, Inc.)

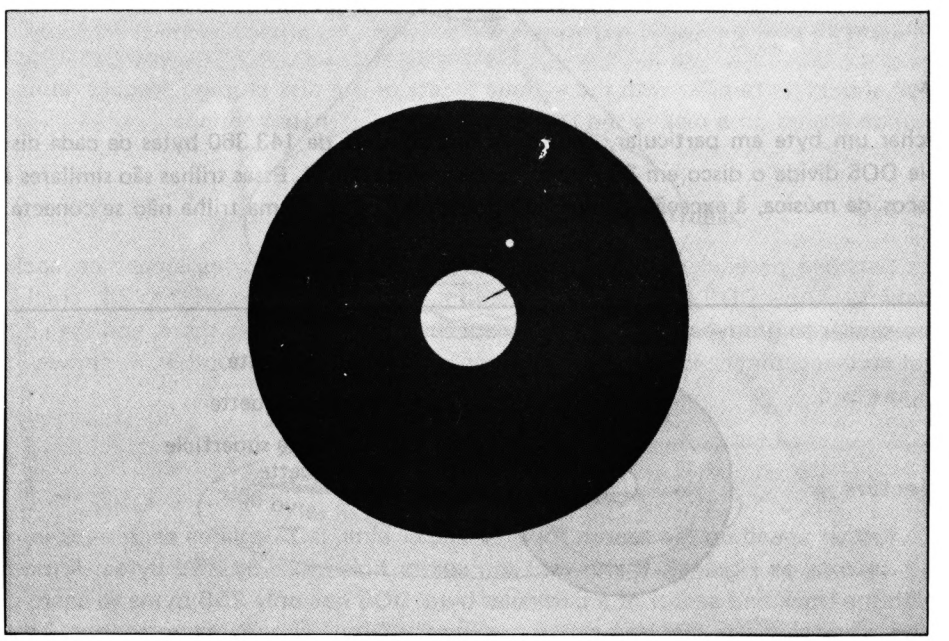


FIGURA 5.3. Um Disquette Sem a Proteção do Envelope.

FIGURA 5.4. Disquettes de 8 e 5 1/4 de Polegadas.

COMO OS DADOS SÃO GUARDADOS NOS DISCOS

É interessante familiarizar-se com as várias facetas do processo de armazenamento em disco. A informação armazenada num disco é o resultado de várias ações coordenadas.

Trilhas

Para achar um byte em particular, dentro de um universo de 143.360 bytes de cada disquette, o Apple DOS divide o disco em 35 trilhas, numeradas de 0 a 34. Essas trilhas são similares àquelas dos discos de música, à exceção de que não podem ser vistas e uma trilha não se conecta com a

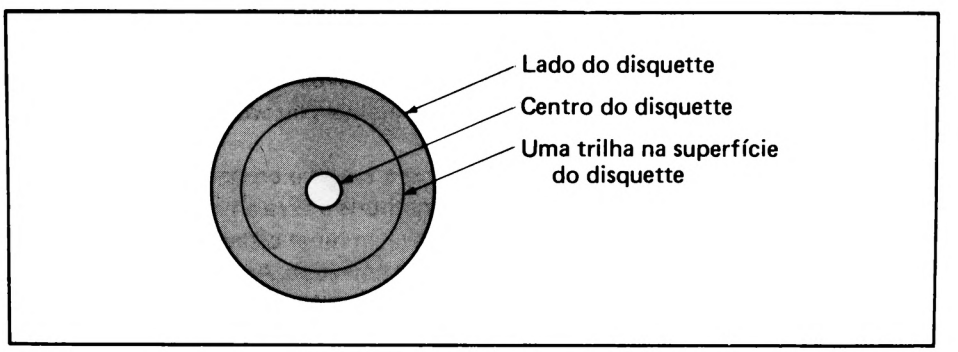


FIGURA 5.5. Trilhas do Disquette.

seguinte, mas sim consigo mesma; isto é, são círculos concêntricos, um dentro do outro, como aparece na Figura 5.5.

Setores

Num passo seguinte para achar um determinado byte, o DOS divide cada trilha em 16 setores, como mostra a Figura 5.6. Cada setor guarda exatamente 256 bytes. Informado sobre a trilha e o setor de um determinado byte, o DOS terá apenas 256 bytes para procurar. Agora, acessar um byte em particular, ou uma série deles, é quase instantâneo. É usado todo o poder do drive de disco.

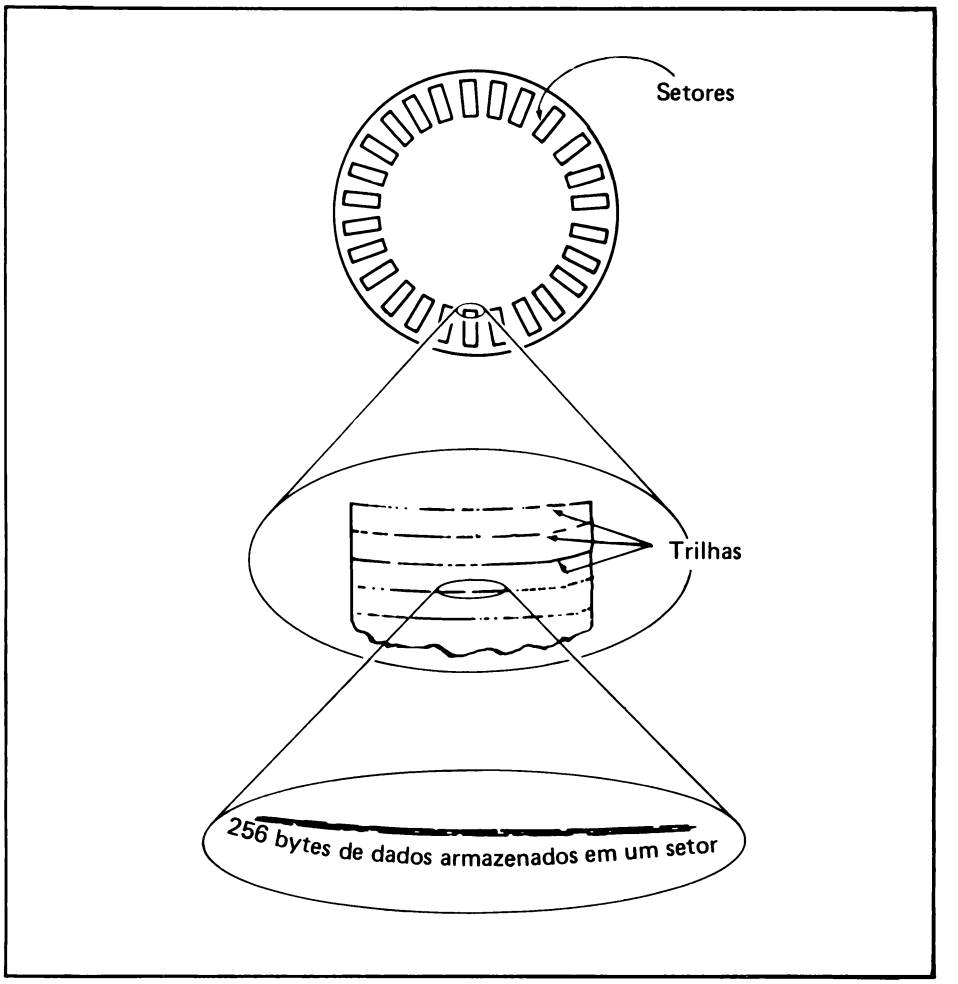


FIGURA 5.6. Superfície Gravada de um Disquette.

LOCALIZANDO TRILHAS E SETORES

Achar a trilha de um disquette é fácil: o drive move a cabeça até o ponto daquele disquette onde está localizada a trilha, da mesma forma que se faria para localizar um disco dentro de um álbum.

Achar o setor é um pouco mais difícil. Existem dois métodos usados para essa tarefa. Ambos usam um orifício de índice. Em muitos disquettes esse orifício é localizado perto do orifício central no centro do envelope. À medida que o disquette gira, o orifício (ou orifícios) de índice passa(m) pelo orifício correspondente do envelope. Uma fonte de luz dentro do drive passa luz para um sensor sempre que os orifícios estão alinhados. O computador sente os pulsos de luz e os setores são localizados tendo como base essa informação.

Existem dois métodos usados para achar os setores do disquette, chamados *setorização por hardware* e *setorização por software.*

Setorização por Hardware

Os disquettes deste tipo têm vários orifícios, como mostrado na Figura 5.7. Cada orifício indica a localização de um setor. Um orifício extra marca o local do primeiro setor. O computador acha os setores contando os orifícios que vêm depois do primeiro setor. O Disk II não usa este método.

Setorização por Software

Os disquettes deste tipo têm apenas um orifício de índice, como mostra a Figura 5.8. Ele marca o primeiro setor. A localização dos demais setores é feita baseada no tempo de rotação do floppy disc.

O Apple II ignora todos os orifícios de tempo. Sua temporização é totalmente eletrônica, e não física. Isso quer dizer que se pode usar tanto disquettes setorizados por hardware quanto por software com o Disk II.

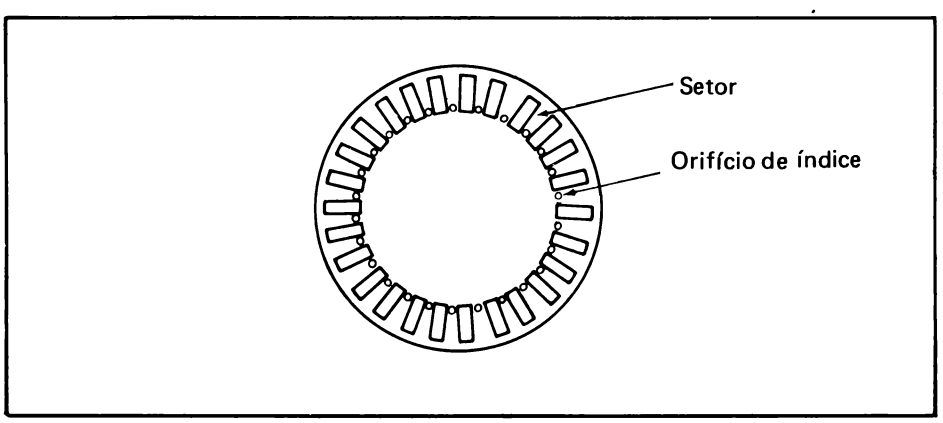


FIGURA 5.7. Disquette Setorizado por Hardware.

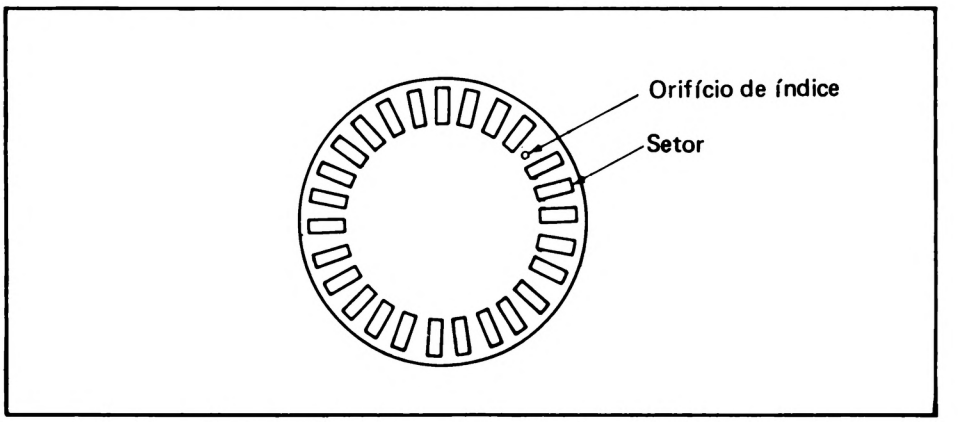


FIGURA 5.8. Disquette Setorizado por Software.

PROTEÇÃO CONTRA GRAVAÇÃO

Existe também um recorte do lado do envelope. Ele é usado para permitir ou evitar que seja gravada informação no disquette. Em disquettes de 8 polegadas este recorte é para proteger contra gravação, porque o computador não pode gravar num disquette se ele tiver o recorte do lado do envelope. Em disquettes de 5 ¼ polegadas o recorte *permite gravação* porque o computador não pode gravar num disquette, a menos que o recorte exista. Alguns disquettes, como os chamados "System Master Diskette", vendidos com o Apple II, são permanentemente protegidos. Eles vêm sem o recorte. Os disquettes de 5 ¼ polegadas podem ficar protegidos contra gravação, cobrindo-se o recorte com um pedaço de fita adesiva. A Figura 5.9 mostra esse processo.

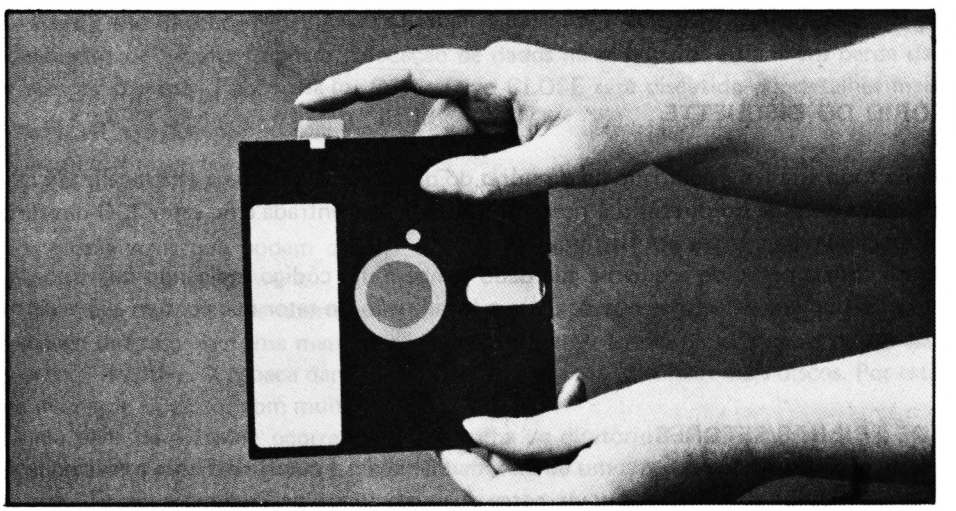


FIGURA 5.9. Protegendo um Disquette de 5 1/4" contra Gravação.

O SISTEMA OPERACIONAL EM DISCO

Todas as operações relacionadas com disco são controladas por um programa especial chamado *Disk Operating System*, ou DOS. O BASIC transmite pedidos ao DOS para qualquer operação onde o disco esteja envolvido. O DOS devolve o resultado ao BASIC.

VERSÕES DO DOS

Existem atualmente várias versões do DOS. O DOS 3.3 é a mais recente. Este capítulo a descreve. A principal diferença entre o DOS 3.3 e o DOS 3.2.1, a última mais recente, é o número de setores: no DOS eram 13 por trilha e no DOS 3.3 são 16. A Apple Computer Inc. tem um programa especial que converte discos do DOS 3.2.1 para DOS 3.3.

INICIALIZANDO DISCOS

Antes que o Apple II possa usar um disquette, ele deve ser *inicializado.* Inicializar um disquette é zerar todos os setores dele e colocar uma cópia do DOS nas trilhas 0, 1 e 2. As instruções de inicialização do Disk II serão vistas mais tarde.

ARQUIVOS EM DISCO

A informação é guardada nos disquettes, sob a forma de *arquivos*. Um arquivo pode ter qualquer tamanho desde que caiba num disquette. Todo arquivo tem um nome. Um arquivo pode guardar informação do tipo texto, um programa ou mesmo uma imagem de vídeo gráfico do Apple II. Os vários tipos de arquivo serão vistos em detalhes mais tarde.

DIRETÓRIO DO DISQUETTE

O nome de todo arquivo é guardado no *diretório* do disquette. O diretório está localizado na trilha 17. A primeira entrada do diretório é no setor 15. A última entrada é no setor 1. O diretório tem espaço suficiente para guardar até 104 arquivos.

Junto com o nome do arquivo é guardado também um código indicando que tipo de dado ele contém, o número de setores que ocupa e a localização do setor que contém a *lista de trilhas/setores* do arquivo.

LISTA DE TRILHAS/SETORES

A lista de trilhas/setores contém pares de bytes que especificam os endereços das trilhas e setores usados num arquivo. Cada par de bytes é chamado de *link.* O primeiro link numa lista de trilha/

setor fornece o endereço do próximo setor usado pela lista. A lista pode ocupar tantos setores quanto for necessário. O segundo link é o endereço do primeiro setor usado pelo arquivo propriamente dito, o terceiro link endereça o próximo setor usado pelo arquivo e assim por diante. Um link com zeros marca o fim da lista.

Sobre o Processo de Armazenamento em Disco

O DOS controla o fluxo de todos os dados de e para o disquette. Quando se grava um arquivo, várias coisas acontecem:

1. O DOS procura o diretório do disquette para achar o nome do arquivo. (O nome dado ao programa o identifica, como veremos mais tarde.)
2. Se o nome for achado, o DOS lê 256 bytes do setor apropriado e os guarda numa área de memória chamada *buffer.* Se o nome não é encontrado, ou se tal setor daquele arquivo jamais foi gravado, o DOS enche o buffer com zeros.
3. Até 256 bytes de dados gravados para o arquivo são copiados da memória para o buffer. Se forem copiados menos de 256 bytes, ficam os dados anteriores.
4. Se, e somente se, forem escritos 256 bytes no buffer, o seu conteúdo é gravado de volta no setor apropriado do disquette.
5. Este processo (passos 3 e 4) é repetido até que todos os dados gravados para o arquivo tenham sido copiados para o buffer e guardados no disquette.
6. Após a gravação de todos os dados no disquette, o DOS atualiza a lista de trilha/setor e o diretório.

Note que a menos que o número de bytes que se grava seja divisível por 256, o último bloco de bytes não caberá no buffer, e os passos 4 a 6 nunca acontecerão. Por isso, o comando CLOSE do Disk II força os dados do buffer a serem gravados no arquivo, após o que são atualizados a lista de trilha/setor e o diretório. Este processo é chamado *fechamento do arquivo*. Falhas no processo de fechamento de arquivos após a colocação de dados neles podem resultar em perda de dados. Feche sempre o arquivo após usá-lo. O comando CLOSE será discutido em detalhes mais tarde.

PANES EM DISCO (CRASH)

Um dos piores erros que podem ocorrer é a chamada "pane" no *disco.* Existem dois tipos de panes: de *hardware* e de *software.*

Uma pane de hardware ocorre quando a superfície do disco é danificada, ou tem um defeito físico, como um rasgo ou uma mancha de sujeira. Ela pode causar danos à cabeça de leitura e gravação dentro do drive. A cabeça danificada pode, por sua vez, danificar mais discos. Por esta razão, sempre manipule os discos com muito cuidado.

Uma pane de software ocorre quando a trilha de diretório é alterada com dados incorretos. O que ocorre com mais freqüência é o não fechamento de um arquivo após a gravação de algo nele, sendo que se coloca outro disco em seu lugar, e os arquivos do disco original são fechados nesse outro disco. Para apreciar o estrago que isso causa, faça-o por conta própria.

CARREGAMENTO DO SISTEMA OPERACIONAL DISK II

Para que o Apple II reconheça qualquer comando de disco, o programa especial DOS (Sistema Operacional do Disco) deve estar na memória. Se houvesse muito tempo disponível, poder-se-ia digitá-lo no teclado. Porém existe uma forma mais fácil: é o que se chama de fazer o *boot* do sistema; este processo consiste em ler uma cópia do DOS do disco e colocá-la na memória.

COMO CARREGAR O SISTEMA

Existem várias formas diferentes de carregar o DOS, dependendo da configuração do computador usada e da linguagem usada para iniciar o carregamento. Todo o método assume que o disco está colocado no DRIVE 1 e o controlador de disco está no slot 6.

Coloque o "System Master Diskette" no drive e feche a porta. O passo seguinte depende do sistema usado. No fim do processo, a tela de vídeo se parecerá com uma das formas mostradas na Figura 2.5.

Autostart

O autostart é a forma mais fácil de carregar o DOS. Como o nome diz, o carregamento é automático. Para isso, o computador deve ter o Autostart Monitor. Pode-se saber se o computador tem o Autostart Monitor instalado, ligando-o com o Disk II conectado. Se a lâmpada IN USE da cabine do disco acender e o disco rodar e gerar estalidos, ele terá o Autostart Monitor instalado.

Para carregar o sistema com o Autostart Monitor, simplesmente ligue a chave do Apple II.

Carregamento a Partir do Monitor

Quando o caractere de apresentação do monitor (*) aparecer na tela, o Monitor de Linguagem Assembler do Apple II estará esperando comandos. Existem várias formas de carregar o DOS a partir do Monitor.

Salto do Monitor para o "BOOT"

Digite a letra C, seguida pelo número do slot do drive que deve ser carregado (normalmente 6), dois zeros e a letra G. O comando inteiro é mostrado abaixo:

```
*C600G
```

C600 é o endereço do programa na memória que carrega a partir do drive do slot número 6. G é o comando que transfere o controle para o programa. Acione RETURN em seguida. A lâmpada do drive deve acender e o drive emitirá ruídos.

Boot do Monitor com CTRL-K e CTRL-P

Outros comandos no Monitor, que podem ser usados para carregar o DOS são CTRL-K e CTRL-P.

Para usar estes comandos, digite o número do slot (normalmente 6) seguido por CTRL-K ou CTRL-P. Tanto um quanto outro não aparecerão na tela.

Depois de digitar estes comandos acione a tecla RETURN.

Carregando a Partir do Integer BASIC ou Applesoft

Os mesmos comandos de carregamento são reconhecidos tanto pelo Integer BASIC quanto pelo Applesoft.

Carregando a Partir do BASIC Usando os Comandos PR# e IN#

Após o aparecimento do caractere de apresentação do BASIC (> para o Integer BASIC e] para o Applesoft), digite as palavras PR ou IN, o sinal de número (#), e finalmente o número do slot onde está o disco para o *boot.* O comando inteiro é como um dos que seguem:

```
PR#6
```

ou

```
IN#6
```

Agora digite a tecla RETURN.

Carregando a Partir do Apple Language System

Se este sistema estiver instalado no Apple II, os processos de carregamento acima não se aplicam. Ele usa dois discos para carregar o DOS versão 3.2.1, 3.2 e anteriores. Porém o DOS 3.3 não opera como foi descrito acima.

Para carregar o DOS 3.2.1, 3.2 e anteriores usando o Language System, coloque o disquette chamado "BASICS: Integer and Applesoft II" (fornecido com o Language System) no lugar do "System Master Diskette" descrito acima, procedendo então de uma das formas descritas antes (autostart, PR#6 etc.). Após o carregamento correto, a tela mostrará:

```
INSERT BASIC DISK AND PRESS RETURN
```

Isso não é bem correto. Realmente o disquette que deve ser colocado é o DOS (o "System Master Disquette" funciona, ou se pode usar qualquer outro que tenha sido inicializado). Após a colocação do segundo disquette e o acionamento da tecla RETURN, o carregamento será executado normalmente e a tela mostrará uma das formas da Figura 2.5.

INICIANDO COMANDOS DO DISCO

O Sistema Operacional do Disco do Apple II interpreta e executa comandos do disco. Muitos comandos permitem ou requerem parâmetros adicionais que definem mais especificamente a operação a ser executada. Alguns comandos elementares são descritos a seguir.

```
*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

FIGURA 5.10. Catálogo de Disco Típico.

CATALOG

O comando CATALOG lista todos os nomes dos arquivos do diretório do disquette para o dispositivo de saída corrente, normalmente a tela.

O CATALOG primeiro apresenta o número de volume do disquette (Número de volume serão discutidos mais adiante).

Para cada arquivo do disquette, o CATALOG lista o *tipo* de dado no arquivo, se o arquivo está *travado* ou não, o número de setores que são ocupados pelo arquivo e seu nome. Um CATALOG típico está mostrado na Figura 5.10.

Tipos de Arquivos

O tipo de dado no arquivo é representado por uma letra que aparece na coluna mais à esquerda do catálogo. Os códigos usados são listados na Tabela 5.1.

TABELA 5.1. Código de Tipo de Arquivo em Disco.

Código	Significado
A	Programas em Applesoft
B	Arquivos em imagem binária
I	Programas em Integer BASIC
T	Arquivos de texto
R	Binário relocável

Arquivos Travados

Não se pode gravar ou apagar um arquivo travado. O catálogo indica o travamento precedendo o tipo do arquivo com um asterisco (*). Se aparece um espaço em lugar do asterisco, o arquivo não está travado. Os arquivos travados serão estudados mais tarde, neste capítulo.

Número de Setores

O número de setores que o arquivo ocupa é mostrado como um número de três dígitos. O menor arquivo (vazio) tem o comprimento de um setor. Se um arquivo tiver mais que 255 setores, este número será zerado e reiniciado, o que não afeta o tamanho real do arquivo.

Nomes de Arquivos

O Apple II requer que se referencie cada arquivo pelo seu nome. Aqui estão as regras que governam os nomes que são dados aos arquivos:

1. Os nomes devem ter de 1 a 30 caracteres de comprimento. Caracteres em excesso são desprezados.
2. Devem começar por uma letra.
3. Qualquer caractere digitado no teclado pode fazer parte do nome do arquivo, com exceção de vírgulas.

Pode-se usar caracteres que não se imprimem (como aqueles criados com a tecla CTRL) como parte do nome de arquivo, porém eles não serão mostrados na lista do catálogo. Isso é útil quando se deseja que outros não saibam o nome dos arquivos com que se está trabalhando. (Não se esqueça de que caracteres foram colocados em seu arquivo.)

Usando o Comando CATALOG

Para usar o comando CATALOG, digite-o simplesmente (após o DOS ter sido carregado) como abaixo:

```
CATALOG
```

O resultado deve se parecer com algo mostrado na Figura 5.10.

Se o número de linhas do CATALOG exceder a 20, o computador mostrará as primeiras 21 linhas, esperando então o acionamento de qualquer tecla (com exceção de RESET, CTRL e SHIFT); para mostrar as próximas 21. Essa pausa dá tempo para que se leiam todos os nomes dos arquivos antes de qualquer um deles ser jogado para fora por excesso de nomes.

LOAD

O comando LOAD lê um arquivo de programa do disquette para a memória. Deve-se especificar o nome do arquivo a ser carregado. Este exemplo carrega o programa chamado COLOR DEMOSOFT:

```
LOAD COLOR DEMOSOFT
```

Se o nome de arquivo especificado não estiver no diretório do disquette, aparece a mensagem FILE NOT FOUND.

Se o arquivo está no disquette, o DOS verifica o seu tipo de dado. Se não for um arquivo de programa, aparece a mensagem FILE TYPE MISMATCH.

Considerando que tudo está certo, o LOAD zera o programa que está presentemente na memória de leitura/escrita e copia o programa do arquivo para a memória. Após a volta do caractere de apresentação pode-se listar, modificar ou rodar o novo programa carregado.

A VERSÃO EM DISCO DO COMANDO RUN

Freqüentemente após dar-se o comando LOAD, dá-se o comando RUN. Pode-se abreviar estes dois passos especificando o nome do arquivo com o comando RUN. O comando LOAD fica implícito, desde que o arquivo deve ser carregado antes que possa rodar. Aqui estão alguns exemplos:

```
RUN PROGRAMA2
RUN COLOR DEMOSOFT
RUN TESTE MEMORIA
```

ESPECIFICANDO O NÚMERO DO DRIVE

Muitos comandos do DOS permitem especificação do drive de disco. São usados dois parâmetros para especificar o drive de disquette: o parâmetro *drive* e o parâmetro *slot*.

Pode-se conectar dois drives de disquette a um controlador. Para selecionar o drive, ponha uma vírgula e D1 ou D2 no comando de disquette, como segue:

```
LOAD CIMA,D2
RUN ESTUDO,D1
CATALOG,D2
```

Depois que se usa o drive especificado uma vez, os comandos subseqüentes de disco serão processados sobre o mesmo drive, até que se especifique outro. O drive usado terá sempre sido especificado pelo comando anterior mais recente. Se não foi usado ainda nenhum drive, o sistema usa o drive 1.

ESPECIFICAÇÃO DO SLOT

Os cartões de controle do Disk II são conectados a slots dentro do Apple II. Existem oito slots disponíveis, porém o de número 0 não pode ser usado para controladores do Disk II. Restam sete slots para controladores do Disk II. Como cada controlador pode suportar de um a dois drives, será possível um máximo de 14 drives ao Apple II.

Se existirem mais de dois drives conectados, eles não poderão ser chamados de D3 ou D4 etc. Em vez disso deve-se usar um outro parâmetro para especificar o controlador apropriado. Este

parâmetro é chamado *slot* e se usa para indicar em qual slot o controlador de disco que se deseja utilizar está conectado.

Para usar o parâmetro slot, coloca-se uma vírgula, a letra S e o número do slot (1-7) dos comandos de disco, como a seguir:

```
CATALOG,S5
LOAD TRILHOS,S6
RUN ESTOQUE,S3
```

O slot usado quando se carrega o DOS torna-se o corrente. Ele será selecionado até que outro seja especificado num parâmetro slot.

Depois de usar o parâmetro slot uma vez, os comandos de disco subseqüentes se valerão daquele disquette selecionado no parâmetro até que se especifique outro.

Pode-se usar os parâmetros drive e slot conjuntamente, no mesmo comando. Especificando o número do slot de 1 a 7, e o número do drive entre 1 e 2, referencia-se qualquer drive do Disk II conectado ao sistema. Os parâmetros drive e slot podem aparecer em qualquer ordem. Os dois comandos seguintes são equivalentes:

```
CATALOG,D2,S5
CATALOG,S5,D2
```

Ambos os comandos produzirão o catálogo do disquette no drive número 2 do controlador instalado no slot 5.

Problemas com o Parâmetro Slot

Se selecionarmos um slot onde não haja um controlador conectado, o computador travará. Ele ficará esperando por um controlador que não existe para colocá-lo em uso. Para voltar com o programa intacto, acione RESET. Se aparecer um dos caracteres do BASIC (> ou]), tudo estará bem. Se aparecer o caractere do Monitor (*), digite 3DOG e a tecla RETURN. Se não funcionar, o programa foi perdido porque o DOS deverá ser recarregado.

ESPECIFICAÇÃO DE VOLUME

Volume é outro parâmetro que pode ser especificado com qualquer comando do DOS que permita especificação de slot e drive, exceto CATALOG. O volume permite certificar-se que foi colocado um disquette correto no drive que foi selecionado.

O comando CATALOG ignora a referência de volume. Quando ele lista o diretório do disquette, o número de volume do disco é o primeiro a ser listado.

Para usar o parâmetro volume, ponha uma vírgula e a letra V seguida pelo número de volume, como segue:

```
LOAD TOTAL,V191
```

Se o número do volume especificado não é o mesmo atribuído ao disquette quando ele foi inicializado, o DOS devolve a mensagem VOLUME MISMATCH.

O número de volume deve estar entre 1 e 254. Se não for especificado, ou se for selecionado o volume 0, o parâmetro volume é ignorado.

Pode-se usar o parâmetro volume em combinação com o de slot ou drive, ou ambos, em qualquer seqüência. Aqui estão alguns exemplos:

```
LOAD CARGA,D2,V24
RUN FOLHA,S3,V111
```

MAIS COMANDOS DO DISK II

Já sabemos consultar o que tem num disco usando o CATALOG, sabemos carregar e executar programas. Vamos ver como colocar nossos próprios programas em disco, usando os comandos INIT, SAVE, DELETE, LOCK, RENAME e VERIFY, que serão descritos abaixo.

INIT

Antes que se possa gravar algo no disquette, ele precisa ser *inicializado.* Quando isso acontece, qualquer coisa gravada anteriormente é zerada, por isso deve-se ter cuidado para não inicializar um disquette que contenha informações úteis.

O comando INIT coloca no disquette qualquer programa que esteja na memória no momento de sua execução. Esse programa se tornará de *saudação*, porque será executado automaticamente toda vez que se carregar o DOS a partir daquele disquette. Tais programas podem ser simples ou complexos, de acordo com o desejado. Por exemplo, suponha um disquette que contenha um sistema de mala direta. Poder-se-ia pensar em usar esse programa como o de saudação. Dessa forma, ao se colocar o disquette no drive e carregar o DOS, pronto! O programa de mala direta já sai rodando. Outro exemplo de programa de saudação que consiste numa declaração NEW seguida de uma END. Toda vez que o DOS é carregado, será dado o caractere de entrada do BASIC (> ou]).

Um programa de saudação bem escrito deve dizer algo sobre o disquette. Um exemplo típico pode ser visto abaixo:

```
100  TEXT
200  CALL -936
300  PRINT "ESTE E O MEU PRIMEIRO DISQUETTTE"
400  PRINT
500  PRINT "INICIALIZADO EM 25/3/83"
600  PRINT
700  PRINT "NUM SISTEMA DE 48K COM DOS 3.3"
800  END
```

O nome do arquivo do programa de apresentação deve ser especificado quando se dá o comando INIT. Será necessário manter sempre um programa com aquele nome no disquette. Se o

programa de saudação for eliminado (mostraremos como, adiante), aparecerá a mensagem de erro FILE NOT FOUND toda vez que se carregar o DOS daquele disquette. A única forma de parar com a mensagem é colocar um programa no disquette com o nome do programa de saudação. Isso pode ser uma tarefa difícil se esse nome não for lembrado, porque é difícil determinar o nome do programa de saudação depois de dado o INIT. A melhor solução é prevenir. *Sempre* especifique o mesmo nome de arquivo quando inicializar disquettes. O nome standard para o programa de saudação é HELLO.

Usando o Comando INIT

Um comando INIT típico parece-se com:

```
INIT HELLO,S6,D1,V36
```

Os números de slot, drive e volume são opcionais. Se o número de volume for incluído, o DOS *atribui* aquele número de volume ao disquette. O INIT é o único comando de atribuição do número de volume. Quando esse número é usado com outros comandos do DOS, o número especificado deve coincidir com o atribuído no momento do INIT. Se tal número for omitido, o INIT sempre atribuirá o número de volume 254.

Como sempre, a omissão de número de drive e slot fará com que sejam usados no comando do DOS anterior. Tenha certeza de qual drive o comando INIT selecionará. Se não estiver certo, especifique!

Para inicializar um disquette novo, primeiro remova o "System Master Diskette" do drive e troque-o por um novo em branco. Use o comando NEW para gerar espaço na memória para o programa de saudação. Digite então este programa de saudação mostrado acima, ou qualquer outro que desejar. É uma boa idéia rodar o programa de saudação antes de gravá-lo no disquette.

Vamos atribuir o número de volume para 123. Digite:

```
INIT HELLO,S6,D1,V123
```

Certifique-se que a porta do drive esteja fechada e acione RETURN. A lâmpada do drive se acenderá, acompanhada dos estalos e ruídos usuais. O processo inteiro leva cerca de dois minutos, portanto seja paciente. Após a lâmpada apagar, use o comando CATALOG para ver o que está gravado no disquette novo. O resultado deve parecer com:

```
DISK VOLUME 123
 I   002  HELLO
```

ou, usando o Applesoft, deve aparecer:

```
DISK VOLUME 123
 A   002  HELLO
```

A letra I indica que o programa de apresentação foi escrito em BASIC. A letra A diz que foi escrito em Applesoft. O 002 indica que o programa ocupa dois setores. Pode-se ver um número diferente, dependendo do comprimento do programa de saudação particular.

Prepare um rótulo para o disquette. Escreva o número de volume do rótulo, assim como qualquer informação que achar necessária sobre ele. Retire o disquette do drive, aplique o rótulo e coloque-o de volta no drive. Se for necessário recarregar usando o disquette recém-preparado, pode fazê-lo.

SAVE

Se foram seguidas as orientações deste capítulo, teremos agora um disquette recém-inicializado no drive 1, slot 6 do Apple II. Deve-se ter também uma cópia do programa de saudação na memória. Use o comando LIST e veja se está lá. Se não, digite:

```
LOAD HELLO
```

para ler uma cópia do disco.

O comando SAVE guarda programas em um disquette. Para gravar uma outra cópia de seu programa de saudação, digite o comando SAVE seguido pelo nome do arquivo. Neste exemplo, usaremos o arquivo chamado PROGRAMA SAUDAÇÃO. Pode-se usar qualquer nome de arquivo válido desejado.

```
SAVE PROGRAMA SAUDACAO
```

O disco deve rodar e emitir seus ruídos habituais. Quando o SAVE termina, reaparece o cursor e o caractere de apresentação do BASIC. A seguir use o comando CATALOG para verificar o conteúdo do disco. Aparece então algo assim:

```
DISK VOLUME 123
A 002  HELLO
A 002  PROGRAMA SAUDACAO
```

Pode-se salvar qualquer programa BASIC desejado desta forma.

Se já houver no disquette algum programa gravado com o mesmo nome que se der no comando SAVE, o novo programa gravado pelo SAVE ocupará o lugar do anterior. Assim o programa anterior será automaticamente eliminado. Isso funcionará desde que tanto o velho quanto o novo programa sejam da mesma versão do BASIC. Se não forem, o novo programa não será gravado, aparecendo a mensagem FILE TYPE MISMATCH.

DELETE

Em pouco tempo acumulam-se muitos programas num disquette que não têm mais propósito algum. O comando DELETE remove arquivos do disquette.

Para eliminar ("deletar") a cópia do programa de saudação gravado, podem ser usados qualquer dos comandos seguintes (a menos que tenha sido referenciado um outro drive no meio tempo).

```
DELETE PROGRAMA SAUDACAO,S6,D1,V12
DELETE PROGRAMA SAUDACAO,V123
DELETE PROGRAMA SAUDACAO
```

Lembre-se que se pode usar os parâmetros slot, drive e volume se quisermos, ou nenhum deles se o drive a ser acessado é o corrente.

LOCK

Alguns programas ou arquivos de dados em disquette devem ser mantidos permanentemente. Para isso, o DOS tem uma técnica de proteção chamada *locking* (travamento). Travar um arquivo evita que ele seja acidentalmente eliminado ou que algo seja escrito sobre ele. Para travar um arquivo, digite o comando LOCK, seguido pelo nome do arquivo e os parâmetros opcionais drive, slot e volume. O comando seguinte trava o programa saudação:

```
LOCK HELLO
```

É sempre bom travar o programa de saudação.

Qualquer tentativa subseqüente de eliminar ou gravar sobre um arquivo travado resultará na mensagem FILE LOCKED.

Se o arquivo travado for um programa em Integer BASIC ou Applesoft e for tentado salvar um programa com o mesmo nome porém escrito em outro BASIC, a mensagem mostrada será FILE TYPE MISMATCH.

Num catálogo de disquette, os arquivos travados são identificados por um asterisco (*) precedendo o tipo do arquivo.

UNLOCK

Quando se decide regravar ou eliminar um arquivo travado, remove-se o lock com o comando UNLOCK. A declaração seguinte destrava o programa saudação:

```
UNLOCK HELLO
```

Como sempre, as especificações de drive, slot e volume são opcionais e podem ocorrer em qualquer ordem.

RENAME

Pode-se trocar o nome de qualquer arquivo do disquette. Uma forma de trocar o nome de um programa seria lê-lo, eliminá-lo, e gravá-lo com um nome novo. Uma forma melhor é usar o comando RENAME. Ele funciona para qualquer arquivo do disco, independente do tipo de BASIC que está sendo usado. O RENAME também funciona com arquivos de texto e binários. Aqui está um exemplo do RENAME:

```
RENAME VELHONOME,NOVONOME
```

O DOS não verifica se o nome do novo arquivo já existe no disquette. Se existir, ficarão dois arquivos com o mesmo nome. Isso pode causar muita confusão e dificultará a recuperação.

Não faça RENAME com o programa saudação a menos que coloque um novo programa com o nome velho no disquette. O RENAME não muda nomes que o DOS travou quando o sistema do disquette é carregado.

Não se pode mudar o nome de um arquivo travado.

Pode-se especificar o drive, slot e volume (em qualquer ordem) se desejado.

VERIFY

Ocasionalmente, pode-se querer verificar se um arquivo está intacto. A melhor forma de fazer isso é usando o comando VERIFY. O comando seguinte testa o programa saudação:

```
VERIFY HELLO,V123
```

À medida que o DOS grava os setores de um arquivo, ele calcula um número, chamado *checksum.* O seu valor é baseado num valor numérico para cada caractere do setor. Ele também é gravado no setor. Quando se executa o comando VERIFY, o DOS recalcula o checksum para cada setor do arquivo e compara com o gravado. Se ele detectar qualquer diferença entre os valores, será devolvida a mensagem I/O ERROR.

Se todos os valores calculados coincidirem com os gravados, o DOS não manda nenhuma mensagem; reaparecem o cursor e o caractere do BASIC.

Assim como muitos outros comandos do DOS, pode-se especificar slot, drive e volume na ordem que se desejar.

USANDO COMANDOS DO DOS EM PROGRAMAS

Até agora todos os comandos do DOS usados foram digitados no teclado do Apple II. Porém os programas em BASIC que usam arquivos podem incluir comandos do DOS.

Para usar um comando do DOS dentro de um programa BASIC, coloca-se uma declaração PRINT pré-fixada com um código ASCII de 4 caracteres. Use o CTRL-D para criar este prefixo. O CTRL-D deve ser o primeiro caractere a sair na declaração PRINT. Certifique-se que o PRINT dado anteriormente não foi terminado com uma vírgula ou ponto-e-vírgula.

Desde que o CTRL-D produz um caractere que não é impresso, é recomendável que se documente cada ocorrência com uma declaração REM, como aqui:

```
100  PRINT  "RUN MENU": REM EXISTE UM CTRL-D
     ENTRE A PRIMEIRA ASPA E O R DE RUN
```

Pode-se simplificar as coisas definindo uma variável como CTRL-D, e mandando imprimir a variável seguida pelo comando do DOS. A variável D$ é comumente usada para isso, mas pode-se usar qualquer outro nome também. Usando a variável padronizada D$ em todos os programas eles se tornam compatíveis uns com os outros e com todos os de outras pessoas. Ponha uma linha como a seguinte no seu programa BASIC:

```
10  D$ = "": REM EXISTE UM CTRL-D INVISIVEL
    ENTRE AS ASPAS
```

ou, em Applesoft, pode-se usar:

```
10 D$ = CHR$ (4): REM CHR$ (4) = CTRL-D
```

Assim, sempre que o D$ for usado, será fácil ver que se trata do caractere CTRL-D.

Tente rodar o programa mostrado abaixo no seu Apple II:

```
10  D$ = "": REM CTRL-D
20  FOR I = 1 TO 10
30  PRINT D$;"CATALOG"
40  NEXT I
50  END
```

A menos que se tenha mais de 18 arquivos no disquette, deve-se ver com isso o catálogo mostrado 10 vezes, sem tocar qualquer tecla. Se existirem mais de 18 entradas no catálogo, deve-se acionar uma tecla cada vez que houver uma parada.

USANDO ARQUIVOS EM DISCO

O Apple II opera com dois tipos de arquivos em disco: arquivos *seqüenciais* e *aleatórios.* Ambos contêm blocos de dados, chamados campos. Um campo pode ter um ou vários caracteres de comprimento.

Arquivos Seqüenciais

Arquivos seqüenciais, como o nome indica, só podem ser acessados de maneira seqüencial. Para ler ou gravar o último campo, deve-se primeiro ler ou gravar todos os campos que o antecedem. Para algumas aplicações, os arquivos seqüenciais são interessantes.

Arquivos Aleatórios

Arquivos aleatórios são mais flexíveis que os seqüenciais. Pode-se ler ou gravar qualquer campo no arquivo não importando sua colocação. Para muitas aplicações, este tipo de arquivo é a melhor solução.

USANDO ARQUIVOS SEQÜENCIAIS

Para usar arquivos seqüenciais, é necessário aprender alguns comandos adicionais do DOS: OPEN, CLOSE, READ e WRITE.

Abrindo Arquivos Seqüenciais

Os arquivos em disco devem ser *abertos* antes de serem acessados. Abrir um disco faz com que o DOS obtenha informações sobre ele: se ele está no disco e, se estiver, sua posição. O OPEN também reserva uma área de memória para ser usada como buffer para o disco. Este buffer permite acessar partes pequenas do arquivo sem necessidade de ativar o drive de disco a cada vez que se acessa um campo, ganhando tempo. O comando OPEN parece-se com:

```
OPEN NOMEARQ,S6,D2,V99
```

Se o arquivo não existe como especificado, o DOS cria um novo no diretório do disco.

Dentro de um programa, o comando OPEN deve estar dentro de uma declaração PRINT e deve ser precedido pelo caractere CTRL-D.

O programa abaixo cria o arquivo seqüencial SEQÜÊNCIA no disco do drive corrente:

```
100 D$ = "": REM CTRL-D
200 PRINT D$;"OPEN SEQUENCIA"
300 END
```

Vamos chamá-lo *Programa 1.*

Pode-se incluir qualquer combinação de parâmetros slot, drive e volume com o comando OPEN:

```
100 D$ = "": REM CTRL-D
200 PRINT D$;"OPEN SEQUENCIA,S6,D1,V123"
300 END
```

ou

```
200 PRINT D$;"OPEN SEQUENCIA,V123,S6"
```

Note que todos os parâmetros são colocados *antes* do fechamento das aspas, e são separados por vírgulas.

Após rodar o programa, o arquivo SEQÜÊNCIA estará presente no catálogo. Confirme digitando CATALOG. Este deverá parecer-se com:

```
DISK VOLUME 123

*A 002 HELLO
 T 001 SEQUENCIA
```

Veja a letra T antes do número de setores de SEQÜÊNCIA. T é o código que indica que SEQÜÊNCIA é um arquivo de texto, em oposição a arquivos de programa em Applesoft ou Integer BASIC, ou arquivos binários. O asterisco (*) antes do tipo de HELLO indica que ele está travado.

Fechando Arquivos

Na verdade, o Programa 1 é bastante pobre, por uma boa razão: ele não fecha o arquivo depois de operá-lo. O comando CLOSE é muito importante. Não fechar os arquivos pode resultar em perda de dados, e possivelmente destruição de dados em outro disquette (veja a seção sobre panes de software neste capítulo). O comando CLOSE tem dois formatos. O primeiro é:

```
CLOSE
```

O comando CLOSE sem parâmetros fecha todos os arquivos, em todos os disquettes, independente do drive, slot e volume em que estiverem.

Às vezes é necessário fechar um ou mais arquivos específicos. Para fechar arquivos individuais coloca-se o nome do arquivo na frente do comando CLOSE, como:

```
CLOSE NOMEARQ
```

Não é necessário nem permitido que se coloque parâmetros de drive, slot ou volume no comando CLOSE. O DOS sabe onde o arquivo está desde que ele já foi aberto.

O Programa 1 pode ser corrigido adicionando as linhas:

```
290 PRINT D$;"CLOSE"
```

ou ainda:

```
290 PRINT D$;"CLOSE SEQUENCIA"
```

Gravando em Arquivos Seqüenciais

O Programa 1 é inútil ainda. Os disquettes são usados para armazenar e recuperar informações. Como não se pode recuperar nada que não tenha sido gravado no disco, vamos ver como gravar informações no disco.

Mandam-se os dados para o Disk II da mesma forma que para tela ou impressora: via declaração PRINT. Qualquer coisa que se imprime pode ser mandada para o disco. De fato, pode-se visualizar um arquivo seqüencial como uma tela de vídeo, ou melhor ainda, como papel numa impressora.

Quando se grava alguma coisa em um arquivo, o DOS atualiza um apontador interno, que aponta a próxima posição da superfície do disco para guardar o próximo dado, tal como a impressora pula para a próxima linha.

Um arquivo seqüencial só pode se mover para a frente. O comando OPEN move o apontador para o início do arquivo.

Antes de se gravar qualquer coisa no disco, deve-se antes dar o comando WRITE para informar ao DOS que as declarações PRINT serão para gravar no arquivo, ao invés da tela. Para arquivos seqüenciais, o comando WRITE é:

```
WRITE NOMEARQ
```

Após dar este comando, as saídas subseqüentes vão para o arquivo especificado. Note que são incluídas as mensagens de erro. Entretanto, após a mensagem ser gravada no arquivo, o comando WRITE é cancelado. Só é visto o cursor e o caractere do BASIC na tela.

O comando WRITE deve estar dentro de uma declaração PRINT, precedida por CTRL-D. Se for tentado dar o comando WRITE no modo imediato, aparece a mensagem de erro NOT DIRECT COMMAND. Ponha as seguintes linhas no Programa 1:

```
210 PRINT D$;"WRITE SEQUENCIA"
220 PRINT "ESTE TEXTO VAI PARA O ARQUIVO"
```

Este programa faz o seguinte:

1. Cria um arquivo, se necessário.
2. Abre o arquivo.
3. Guarda texto no arquivo.
4. Fecha o arquivo.

Pode-se inserir tantas declarações PRINT entre as linhas 210 e 290 quanto desejado. Pode-se gravar texto, números e variáveis, em qualquer combinação, desde que a sintaxe do PRINT esteja correta. Por exemplo:

```
220 FOR I = 1 TO 100
230 PRINT I
240 NEXT I
250 PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Veja que só se usa o prefixo CTRL-D com comandos do DOS e nunca quando se está gravando dados no arquivo.

Tome cuidado para não gravar dados no arquivo com as funções FLASH ou INVERSE atuantes, desde que esses caracteres não são tratados convenientemente pelo DOS.

Lembre-se que a cada vez que se roda este programa, o que estiver na declaração PRINT vai cobrir e apagar os dados que já estiverem no arquivo. Se forem colocados menos caracteres do que os que já estiverem lá, o restando anterior permanecerá, seguindo os novos dados.

Uma forma de contornar este problema é zerar o arquivo antes de gravar qualquer dado nele. O comando DELETE (dentro de uma declaração PRINT) pode ser incorporado ao programa antes do comando OPEN. Toda vez que o programa rodar, o arquivo será eliminado e depois recriado com o comando OPEN.

Entretanto, um problema ocorrerá ao se tentar rodar o programa sem a presença do arquivo SEQÜÊNCIA no disquette. Se for tentado, aparecerá a mensagem FILE NOT FOUND. Pode-se prevenir isso dando outro comando OPEN antes do DELETE. Acontecerá o seguinte:

1. O primeiro comando OPEN cria um arquivo que ainda não existe.
2. O comando DELETE limpa o arquivo, não importa quando foi criado.
3. O segundo comando OPEN cria um novo arquivo vazio.

Quando fizer estas modificações no Programa 1, lembre-se que uma vez especificado o slot, drive e volume, não será necessário especificá-los novamente, a menos que se deseje referenciar drive diferente. Com isso em mente, use o comando OPEN com os parâmetros slot, drive e volume como o primeiro comando do DOS, e referencia os comandos seguintes ao drive corrente.

Após as modificações no Programa 1, ele ficará assim:

```
100 D$ = "": REM CTRL-D
110 PRINT D$;"OPEN SEQUENCIA,V123,S6,D1"
120 PRINT D$;"DELETE SEQUENCIA"
200 PRINT D$;"OPEN SEQUENCIA"
210 PRINT D4;"WRITE SEQUENCIA"
220 PRINT "ESTE TEXTO VAI PARA O ARQUIVO"
290 PRINT D$;"CLOSE"
300 END
```

Sempre que se executar este programa, ele vai guardar o conteúdo das declarações PRINT entre as linhas 210 e 290 no arquivo SEQÜÊNCIA. Pode-se usar qualquer nome de arquivo que se desejar, mas se o nome do arquivo for mudado, certifique-se de mudar todas as referências ao arquivo.

Pode-se usar uma variável para nome do arquivo e fazer o programa pedir o nome para fazer o acesso. Com essa modificação, o programa fica:

```
100 INPUT "NOME DO ARQUIVO";F$
100 D$ = "": REM CTRL-D
110 PRINT D$;"OPEN";F$;"V123,S6,D1"
120 PRINT D$;"DELETE";F$
200 PRINT D$;"OPEN";F$
210 PRINT D$;"WRITE";F$
220 PRINT "ESTE TEXTO VAI PARA O ARQUIVO"
290 PRINT D$;"CLOSE"
300 END
```

Pode-se ir mais além e criar um programa que peça os números de slot, drive e volume com estas modificações:

```
10 INPUT "NOME DO ARQUIVO: ";F$
20 INPUT "NUMERO DO SLOT: ";S
30 INPUT "NUMERO DO DRIVE: ";D
40 INPUT "NUMERO DO VOLUME: ";V
100 D$ = "": REM CTRL-D
110 PRINT D$;"OPEN ";F$;",S";S;",D";D;",V;",V";V
```

Fazendo estas modificações, certifique-se de entrar da forma mostrada, não se esquecendo de incluir vírgulas entre os parâmetros S, D e V ou o DOS não será capaz de distinguir os parâmetros do nome do arquivo.

Para tornar o programa realmente útil, pode-se deixar entrar o texto a ser gravado no arquivo, ao invés de alterar o comando PRINT. Uma forma de fazer isso é:

```
150 INPUT "ENTRE TEXTO PARA GRAVAR: ";T$
.
.
.
220 PRINT T$
```

Veja que isso permite apenas a entrada e armazenamento de uma linha de texto. Pode-se alterar isso colocando mais declarações INPUT seguidas por declarações PRINT, mas uma vez mais fica determinada a quantidade de texto a ser entrada e guardada. Por que não colocar um teste após a declaração INPUT de forma a sinalizar o fim do texto? Ponha então um GOTO após o PRINT para entrar outra linha:

```
150 INPUT "ENTRE TEXTO PARA GRAVAR: ";T$
160 IF T$ = "FIM" THEN 290
210 PRINT D$;"WRITE ";F$
220 PRINT T$
230 GOTO 150
```

Agora, ao terminar de colocar o texto, pode digitar FIM e o arquivo será fechado. Porém resta um grande problema. Lembre-se de que o comando WRITE leva todas as saídas a serem dire-

cionadas para o arquivo. Desde que é entrada a declaração INPUT sai o texto ENTRE TEXTO PARA GRAVAR:, essa linha não aparecerá na tela após a execução do comando WRITE; ela será armazenada como parte do texto no arquivo em disquette.

O comando WRITE deve ser cancelado antes que a informação saia para o disquette. Qualquer comando do DOS cancelará o comando WRITE, mas o mais seguro é usar o *comando nulo,* que é CTRL-D sozinho. Ponha esta linha de programa:

```
225 PRINT D$
```

Com todas estas modificações, teremos um programa que permite gravar qualquer quantidade de texto desejada (com o tamanho máximo do disquette), sob qualquer nome de arquivo, em qualquer drive conectado ao Apple II.

Lendo Arquivos Seqüenciais

Assim como as saídas podem ser direcionadas para o disco, as entradas podem ser aceitas a partir de arquivos em disco. O comando READ identifica um arquivo em disco como fonte de dados de entrada. Para arquivos seqüenciais o comando READ é assim:

```
READ NOMEARQ
```

O comando READ deve estar dentro de uma declaração PRINT, precedido do caractere CTRL-D. Se o comando READ for dado em modo imediato será mostrado o erro NOT DIRECT COMMAND.

Depois do comando READ ser executado, as próximas declarações de entrada INPUT receberão dados do arquivo especificado até que outro comando do DOS, ou um erro, cancele o comando READ. O *Programa 2*, abaixo, demonstra o uso do comando READ:

```
100 D$ = "": REM CTRL-D
110 INPUT "NOME DO ARQUIVO PARA LER: ";F$
120 INPUT "NUMERO DO SLOT :";S
130 INPUT "NUMERO DO DRIVE :";D
140 INPUT "NUMERO DO VOLUME :";V
150 PRINT D$;"OPEN ";F$;",S";S;",D";D;",V";V
160 PRINT D$;"READ ";F$
170 INPUT A$
180 PRINT A$
190 GOTO 170
200 END
```

O Programa 2 mostra todo o texto do arquivo criado pelo Programa 1. Após todo o texto ter sido mostrado, aparecerá a mensagem END OF DATA e o programa parará com a mensagem BREAK IN 170.

Veja que o Programa 2 não tem o comando CLOSE. Explicamos que os arquivos devem ser fechados quando forem usados mais tarde, porém neste caso ele não será gravado, de forma que a informação de trilha/setor nunca precisa ser atualizada (assumindo que o nome do arquivo especificado realmente existe). O arquivo poderia ser fechado por questão de segurança.

Não é uma boa prática escrever programas que coloquem mensagens de erro e parem sob circunstâncias previsíveis, como o fim de dados no arquivo.

Contornando o Erro END OF DATA

A condição END OF DATA pode ser detectada antes de gerar o erro. O Programa 2 pode sair para o fechamento de arquivo ao detectar o fim dos dados. A maneira mais fácil de detectar o fim dos dados é usar o comando do Applesoft ONNER GOTO (visto no Capítulo 4), porém o ONNER GOTO não é permitido no Integer BASIC. (Os códigos de erro do BASIC são mostrados no Apêndice C.)

Uma outra forma de achar o fim do texto num arquivo escrito pelo Programa 1 é modificá-lo de forma que quando for digitada a palavra FIM, uma palavra ou caractere especial seja escrita antes de fechar o arquivo. Assim o Programa 2 procurará pelo caractere especial para fechar o arquivo de acordo. Este método funciona tanto no Applesoft quanto no Integer BASIC.

Diferenças entre o Applesoft e o Integer BASIC

O comando READ identifica um arquivo em disco como fonte de dados para as declarações INPUT subseqüentes. Porém essas declarações devem ter sua sintaxe de acordo com o BASIC utilizado para o programa (veja o Capítulo 8). De qualquer forma, os dados fornecidos por um arquivo dependem da forma como eles foram guardados.

Veja o programa:

```
100 D$ = "": REM CTRL-D
200 PRINT D$;"OPEN ARQ1"
300 PRINT D$;"WRITE ARQ1"
400 PRINT "ALÔ"," RODANDO PROGRAMA DE TESTE"
500 PRINT D$;"CLOSE"
600 END
```

Em Applesoft, rode este programa, e leia o texto de ARQ1 mudando as linhas 300 e 400:

```
300 PRINT D$;"READ ARQ1"
400 INPUT A$
```

O valor de A$ é ALÔ RODANDO PROGRAMA DE TESTE. Mude o original para gravar de novo:

```
300 PRINT D$;"WRITE ARQ1"
400 PRINT "ALÔ, RODANDO PROGRAMA DE TESTE"
```

e execute-o. Agora leia o valor guardado como antes. Agora apareceu a mensagem ?EXTRA IGNORED, e A$ é ALÔ.

Em Applesoft, as vírgulas separam valores múltiplos numa declaração PRINT única.

O Integer BASIC aceita ambas as linhas como uma, de forma que o valor de A$ guardado pelo primeiro exemplo será ALÔ RODANDO PROGRAMA DE TESTE, e o valor de A$ na segunda versão da linha 400 será ALÔ, RODANDO PROGRAMA DE TESTE.

Usando a Declaração GET do Applesoft para Ler Arquivos de Texto

Em Applesoft, a declaração GET pode ser usada para ler dados de arquivos em disco. A declaração GET difere da INPUT por aceitar um caractere de cada vez. Assim, se um arquivo contém o texto ESTE ARQUIVO CONTÉM TEXTO, e é executado um OPEN, um READ, e depois um GET, o primeiro GET devolverá a letra E; o próximo GET devolverá a letra S, e os GET subseqüentes devolverão T, E, espaço em branco, A, R, Q, U, I, V, O etc., até que todos os caracteres do texto tenham sido lidos. Se o GET devolver vírgulas ou *Carriage Return*, o programa poderá detectar e interpretar de acordo.

O *Programa 3*, mostrado abaixo, demonstra como um arquivo pode ser lido, usando a declaração GET do Applesoft para montar uma linha de texto, um caractere por vez.

```
100 D$ = CHR$ (4): REM  CTRL-D
200 INPUT "NOME DO ARQUIVO PARA LER: ";F$
300 INPUT "NUMERO DO SLOT: ";S
400 INPUT "NUMERO DO DRIVE: ";D
500 INPUT "NUMERO DO VOLUME: ";V
600 PRINT D$;"OPEN ";F$;",V";V;",D";D;",S";S
700 PRINT D$;"READ ";F$
800 B$ = ""
900 GET A$
1000 IF A$ = CHR$ (13) THEN 1300
1100 B$ = B$ + A$
1200 GOTO 900
1300 REM DEVOLVE CARACTERE ACHADO
1400 REM B$ ESTA COMPLETO
1500 PRINT B$
1600 GOTO 800
1700 END
```

Porém a declaração GET tem um problema quando usada com arquivos em disco. O primeiro caractere impresso após o GET ter sido executado é ignorado. Se o primeiro caractere é um comando do DOS, então o caractere CTRL-D será ignorado, o que significa que todo comando do DOS será impresso e não interpretado como comando do DOS.

Para resolver este problema é preciso imprimir um caractere sem significado primeiro, que será ignorado. Um bom caractere para usar é o CTRL-A (código 1 ASCII), por ele não ser imprimível e não ter significado especial (em oposição ao CTRL-D, por exemplo).

O Programa 3 pode ser corrigido, trocando a linha 1500 para ler:

```
1500 PRINT CHR$ (1);B$: REM CHR$ (1) = CTRL-A
```

Guardando Números em Arquivos

Pode-se usar o Programa 1 para guardar números em arquivos, tanto como parte de texto, como em:

```
220 PRINT "MEU ENDERECO E RUA DEZ NUMERO 1123"
```

quanto diretamente em valores numéricos:

```
230 PRINT 1,2,3,4,5
```

ou via variáveis numéricas:

```
240 PRINT A,B,C,D,E
```

Se forem guardados números diretamente, pode aparecer um resultado estranho ao se ler de volta no Programa 2.

Usando o Integer BASIC, números separados por vírgulas ou pontos-e-vírgulas formarão um número grande (12345 na linha 230 acima); para um total de uma linha de saída.

Usando Applesoft, isso se torna ainda mais confuso; os primeiros três números são concatenados (123) e vistos como uma saída de impressão. Os próximos dois valores (4 e 5) produzem uma linha de saída cada, dando o total de três linhas de saída.

Esses problemas vêm do formato usado para guardar dados em arquivos de disco. As vírgulas não são armazenadas entre os números se eles não forem para a tela. Em vez disso, na tela a vírgula causa a impressão do dado na próxima linha de tabulação. Quando a saída é direcionada para o arquivo, entretanto, o DOS despreza completamente as vírgulas. Não acontece nada equivalente à tabulação. Como resultado, os valores tornam-se concatenados até que o retorno (código 13) os separe. Ele, *Carriage Return*, é o único caractere que o DOS interpreta como separador de valores. O Integer BASIC manda um *Carriage Return* após a quinta parada de tabulação (vírgula), enquanto o Applesoft o manda após a terceira.

Para evitar problemas, certifique-se de que cada campo numérico guardado num arquivo em disco seja seguido pelo caractere *Carriage Return*. A forma mais fácil de fazer isso é mandar cada número com uma declaração PRINT separada:

```
230 PRINT 1: PRINT 2: PRINT 3: PRINT 4:
 PRINT 5
```

Outro método permitido aos usuários do Applesoft é incluir um caractere *Carriage Return* como separador na lista de parâmetros da declaração PRINT:

```
230 PRINT 1;CHR$(13);2;CHR$(13);3;CHR$(1
 3);4;CHR$(13);5
```

Pode ser mais cômodo definir uma variável como o caractere *Carriage Return*, e imprimi-la:

```
11 R$ = CHR$(13): REM CARACTERE RETURN
      .
      .
      .
230 PRINT 1;R$;2;R$;3;R$;4;R$;5
```

Isso torna o programa mais limpo.

COMO INSERIR DADOS NUM ARQUIVO SEQÜENCIAL

Um arquivo seqüencial deve ser fechado após se gravar dados nele. Quando se fecha um arquivo, perde-se o controle do local em que o último item foi gravado. Para adicionar dados ao fim do

arquivo, então, deve-se primeiro *achá-lo.* Pode-se ler cada item do arquivo até o último, porém isso consome muito tempo para arquivos longos. O comando APPEND faz isso para nós.

Este comando APPEND coloca o apontador de arquivo no primeiro caractere não usado após o fim do arquivo. Se for dado um READ após o APPEND, obtém-se a mensagem de erro END OF DATA. Se se grava após o APPEND, o novo dado será adicionado ao que já estava no arquivo.

O comando APPEND é usado no lugar do OPEN. Existem duas diferenças importantes entre os comandos APPEND e OPEN:

1. O APPEND requer que o arquivo já exista. Se não, o erro FILE NOT FOUND aparece. O APPEND não cria arquivo, ele assume que o arquivo já existe.
2. O APPEND coloca o ponteiro no *final* do arquivo. O OPEN coloca o ponteiro no *início* do arquivo.

O formato para o comando APPEND é o mesmo que o do comando OPEN. Aqui está um exemplo:

```
APPEND NOMEARQ,S6,D2,V99
```

Como sempre, os comandos slot, drive e volume são opcionais.

O COMANDO POSITION

Outro comando útil é o POSITION. Ele move o ponteiro *para a frente* (nunca para trás) do número especificado de campos em relação à posição atual do ponteiro. O comando se parece com:

```
POSITION NOMEARQ,R30
```

R indica campo relativo, o número após o R dá o número de campos para pular. Os campos são marcados pelo caractere *carriage return*, e o comando POSITION acima conta 30 *carriages returns* a partir da posição atual e move o ponteiro do arquivo para lá. Se for dado um RO, o ponteiro não se move.

Na verdade, o POSITION examina o arquivo caractere por caractere, iniciando pela posição corrente. Se não existirem campos suficientes, ou é encontrado um byte não usado, o erro END OF DATA é mostrado imediatamente. Não é necessário executar o INPUT ou o GET para dar a mensagem de erro.

Um arquivo deve ser aberto antes que possa ser referenciado pelo POSITION. Quando se abre o arquivo, seu ponteiro fica apontado para o início; se for dado o comando POSITION quando o apontador estiver no início do arquivo, ele irá efetivamente acessar o campo *absoluto* dentro do arquivo.

Lembre-se, como qualquer outro comando do DOS, o POSITION cancela os comandos READ e WRITE. Utilize o POSITION antes do READ e do WRITE e não depois.

USANDO ARQUIVOS DE ACESSO ALEATÓRIO

Os arquivos de acesso aleatório são estruturados em seções chamadas *registros*. Cada registro num arquivo particular suporta a mesma quantidade de informação, definida em número de bytes (caracteres), quando o arquivo é criado.

A quantidade de informação que pode ser guardada num registro é referida como *comprimento* do registro.

Os registros são identificados por um número indicando sua posição absoluta no arquivo. O primeiro registro de todo arquivo tem número 0, o seguinte número 1, seguido pelo 2 etc.

O menor arquivo de acesso aleatório tem um registro. A expansão é feita quando novos registros são adicionados, porém eles não encolhem. Para remover registros não usados de um arquivo de acesso aleatório e reduzir seu tamanho, deve-se copiar os registros que devem ser preservados em outro arquivo aleatório novo.

Os programas devem especificar qual registro de um arquivo aleatório deve ser selecionado e que parte dentro desse registro deve ser acessada.

Abrindo Arquivos de Acesso Aleatório

Para definir um arquivo como de acesso aleatório, deve-se incluir um parâmetro adicional quando o arquivo é aberto. O parâmetro (L) especifica o comprimento de cada registro, como aqui:

```
OPEN NOMEARQ,L10,S6,D1,V100
```

O parâmetro comprimento deve ter um valor entre 1 e 32767. Não deve ser o primeiro parâmetro na lista, mas deve estar presente se o arquivo for de acesso aleatório.

Os programas nunca devem gravar registros maiores que o tamanho especificado no parâmetro L, incluindo vírgulas e *carriages returns.* Se forem gravados caracteres em excesso num registro, o registro seguinte poderá ser superposto ou combinado, destruindo sua informação.

Fechando um Arquivo de Acesso Aleatório

O comando CLOSE é idêntico para os arquivos de acesso seqüencial e aleatório.

Leitura e Gravação em Acesso Aleatório

Os comandos READ e WRITE requerem o parâmetro *registro* para arquivos de acesso aleatório. O parâmetro registro move o apontador do arquivo para o início do registro. O exemplo seguinte usa o parâmetro registro (R):

```
READ NOMEARQ,R13
```

ou

```
WRITE NOMEARQ,R6
```

O parâmetro registro não precisa ser o único numa lista, pode-se também especificar drive, slot e volume. Os parâmetros podem aparecer em qualquer ordem. Se o parâmetro registro não está presente o ponteiro não se move.

UM EXEMPLO PRÁTICO DE ACESSO ALEATÓRIO

Os programas seguintes demonstram um uso prático de arquivos de acesso aleatório. Eles rodam tanto em Applesoft quanto em Integer BASIC. Para usar o segundo programa em Integer BASIC, adicione uma declaração DIM antes da linha 100, para a variável B$ e C$ (255 caracteres cada). O primeiro programa cria um arquivo de nome RANDOM.

```
10 REM  PROGRAMA DE CRIACAO DE ARQUIVO ALEATORIO
20 REM
30 REM  RODE ESTE PROGRAMA PRIMEIRO, POIS ELE
40 REM  GUARDA INFORMACAO NO REGISTRO ZERO, QUE
50 REM  DEVE EXISTIR PARA O PROXIMO PROGRAMA OPERAR.
60 REM
100 D$ = "": REM CTRL-D
200 PRINT D$;"OPEN RANDOM,S6,D1,L256" : REM ABRE ARQUIVO
300 PRINT D$;"WRITE RANDOM,R0" : REM GRAVA REGISTRO ZERO
400 PRINT 0 : REM GUARDA ZERO NO REGISTRO ZERO
500 PRINT D$;"CLOSE" : REM FECHA ARQUIVO
600 END
```

Uma vez que o arquivo tenha sido criado, o segundo programa permite ler, alterar, adicionar e listar os registros do arquivo. Cada registro conterá uma lista de informações digitadas no teclado.

```
10 REM DEMONSTRACAO DE ARQUIVO DE ACESSO ALEATORIO
20 REM
30 REM ESTE PROGRAMA MONTA UM ARQUIVO ALEATORIO
40 REM QUE CONSISTE EM REGISTROS DE UMA UNICA LINHA
50 REM
60 REM O REGISTRO ZERO CONTEM UM NUMERO INDICANDO
70 REM O ULTIMO NUMERO DE REGISTRO EM USO.
80 REM
85 REM NOTA: O ARQUIVO "RANDOM" DEVE SER CRIADO ANTES
90 REM       DE TENTAR USAR ESTE PROGRAMA
95 REM
100 D$ = "" : REM CTRL-D
200 PRINT D$;"OPEN RANDOM,S6,D1,L256"
225 PRINT D$;"READ RANDOM,R0" : REM LE REGISTRO ZERO
250 INPUT M : REM M = O ULTIMO REGISTRO EM USO
275 PRINT D$ : REM CANCELA O COMANDO READ
300 CALL -936 : REM LIMPA A TELA
400 PRINT "DEMONSTRACAO DE ACESSO ALEATORIO"
500 PRINT : PRINT : PRINT
600 PRINT "COMANDOS:" : PRINT
700 PRINT " 0 = PARA"
800 PRINT " 1 = LE UM REGISTRO"
```

```
900 PRINT " 2 = SOMA UM REGISTRO"
1000 PRINT " 3 = MUDA UM REGISTRO"
1100 PRINT " 4 = LISTA TODOS REGISTROS"
1200 PRINT : PRINT
1300 PRINT "QUAL";
1400 INPUT C
1500 IF C=0 THEN 8300 : REM PULA
1600 IF C=1 THEN 2200 : REM PARA
1700 IF C=2 THEN 3300 : REM A
1800 IF C=3 THEN 4600 : REM ROTINA
1900 IF C=4 THEN 6700 : REM SELECIONADA
2000 GOTO 300         : REM OU REFAZ O MENU
2050 REM
2100 REM * * * * LE UM REGISTRO * * * *
2150 REM
2200 CALL -936 : REM LIMPA A TELA
2300 PRINT : PRINT "LE UM REGISTRO" : PRINT
2400 PRINT "QUE NUMERO DE REGISTRO (0 PARA PARAR)";
2500 INPUT R
2600 IF R<1 THEN 300 : REM RETORNA AO MENU PPAL
2650 IF R>M THEN 2200 : REM REGISTRO NAO EXISTE
2700 PRINT D$;"READ RANDOM,R";R : REM PREPARA PARA LER
2800 INPUT B$ : REM LE OS DADOS
2900 PRINT D$ : REM CANCELA O COMANDO READ
3000 PRINT : PRINT B$ : PRINT : REM MOSTRA O DADO
3100 GOTO 2400 : REM PEDE OUTRO NUMERO DE REGISTRO
3150 REM
3200 REM * * * * SOMA UM REGISTRO * * * *
3250 REM
3300 CALL -936 : REM LIMPA TELA
3400 PRINT : PRINT "ADICIONA UM REGISTRO" : PRINT
3500 PRINT "PROXIMO NUMERO DE REGISTRO = ";M+1
3600 PRINT : PRINT "ENTRA DADO PARA O REGISTRO ";M+1
3625 PRINT "(DIGITE [RETURN] AGORA PARA PARAR AUMENTO)"
3700 INPUT B$ : REM PEGA RESPOSTA DO USUARIO
3750 IF B$ = "" THEN 300 : REM SAIR COM TECLA [RETURN]
3800 PRINT D$;"WRITE RANDOM,R";M+1 : REM PREPARA
 P/ESCREVER
3900 PRINT B$ : REM MANDA DADO PARA ARQUIVO
4000 M = M + 1 : REM INCREMENTA ULTIMO NUMERO DE REGISTRO
4100 PRINT D$;"WRITE RANDOM,RO" : REM PREPARA PARA
 ESCREVER
4200 PRINT M : REM GUARDA VALOR ATUAL
4300 PRINT D$ : REM CANCELA COMANDO WRITE
4400 GOTO 3500 : REM VOLTA PARA OUTRO REGISTRO
```

```
4450 REM
4500 REM * * * * ALTERA UM REGISTRO * * * *
4550 REM
4600 CALL -936 : REM LIMPA A TELA
4700 PRINT : PRINT "ALTERAR REGISTRO" : PRINT
4800 PRINT "NUMERO DO REGISTRO (0 PARA PARAR)";
4900 INPUT R
5000 IF R<1 THEN 300 : REM VOLTA AO MENU PRINCIPAL
5050 IF R>M THEN 4600 : REM TENTA DE NOVO SE NAO HA REG.
5100 PRINT D$;"READ RANDOM,R";R : REM PREPARA PARA LER
5200 INPUT B$ : REM LE O REGISTRO
5300 PRINT D$ : REM CANCELA O COMANDO READ
5400 PRINT : PRINT B$ : PRINT : REM MOSTRA O DADO
5500 PRINT "ENTRA O NOVO DADO"
5600 PRINT "(ACIONE [RETURN] PARA PARAR ALTERACAO)"
5700 PRINT
5800 INPUT C$ : REM PEGA RESPOSTA DO USUARIO
5900 IF C$>"" THEN 6200 : REM PULA SE DADO NOVO
6000 PRINT "REGISTRO ";R;" INTACTO ! ! ! ": REM VOLTA SE
6100 GOTO 4800 : REM NAO DESEJA ALTERAR
6200 PRINT D$;"WRITE RANDOM,R";R : REM PREPARA PARA GRAVAR
6300 PRINT C$ : REM GUARDA DADO ALTERADO
6400 PRINT D$ : REM CANCELA COMANDO WRITE
6500 GOTO 4800 : REM VOLTA PARA ALTERAR OUTRO REGISTRO
6550 REM
6600 REM * * * * LISTA TODOS OS REGISTROS * * * *
6650 REM
6700 CALL -936 : REM LIMPA TELA
6800 PRINT : PRINT "LISTA TODOS OS REGISTROS" : PRINT
6900 R = 0 : REM LIMPA O CONTADOR
7000 R = R + 1 : REM INCREMENTA CONTADOR
7100 IF R>M THEN 7700 : REM PARA APOS ULT.REGISTRO
7200 PRINT D$;"READ RANDOM,R";R : REM PREPARA PARA LER
7300 INPUT B$ : REM LE O DADO
7400 PRINT "NUMERO DO REGISTRO = ";R : REM NUM.DO REGISTRO
7500 PRINT B$ : PRINT : REM MOSTRA DADOS DO REGISTRO
7600 GOTO 7000 : REM LOOP PARA PROXIMO REGISTRO
7700 PRINT D$ : REM CANCELA COMANDO READ
7800 PRINT : PRINT "* * * * FIM-DE-ARQUIVO" : REM MOSTRA
 MENSAGEM EOF
7900 PRINT "ACIONE RETURN PARA CONTINUAR"; : REM PEDE A
 RESPOSTA
8000 INPUT B$ : REM PEGA RESPOSTA DO USUARIO
8100 GOTO 300 : REM RETORNA AO MENU PRINCIPAL
8150 REM
```

```
8200 REM * * * * PARA O PROGRAMA * * * *
8250 REM
8300 PRINT D$;"CLOSE" : REM FECHA ARQUIVO
8400 CALL -936 : REM LIMPA TELA
8500 FOR I = 1 TO 24 : PRINT : NEXT I : REM MOVE PARA
 ULTIMA LINHA
8600 PRINT "PROGRAMA COMPLETO."
8700 END
```

O PARÂMETRO BYTE

Byte é um outro parâmetro útil para arquivo aleatório. O parâmetro byte é usado com os comandos READ, WRITE e POSITION para mover o ponteiro a um byte específico (caractere) dentro de um registro selectado. Uma vírgula e a letra B podem ser adicionadas ao comando READ, WRITE e POSITION. O parâmetro registro deve estar presente com o parâmetro byte. Aqui está um exemplo:

```
READ NOMEARQ,R19,B3
```

Neste exemplo, a leitura iniciará no quarto byte do vigésimo registro. (Lembre-se, o primeiro byte tem número 0.) O parâmetro byte pode mover o ponteiro para a frente ou para trás dentro do registro.

Se for usado o parâmetro byte, os registros devem ter um formato exato de dados. Deve-se saber a posição exata dos bytes de cada campo dentro do registro, ou o resultado obtido não terá maior significado.

Quando estiver usando o parâmetro byte com o comando POSITION, lembre-se que ele cancela qualquer comando READ ou WRITE anterior. Pode-se executar um comando READ ou WRITE que inclua um parâmetro de registro, e então usar o POSITION para mover para a frente até o campo correto. Deve-se então executar outro READ ou WRITE, desde que o POSITION cancele os comandos READ ou WRITE anteriores.

O POSITION apenas move o ponteiro à frente para outro campo no registro corrente. Dentro do campo, o parâmetro byte permite referenciar subcampos, um caractere por vez.

OUTROS COMANDOS DO DOS

Existem alguns outros comandos que ainda não foram estudados. São eles o EXEC, MAXFILES, TRACE, MON e NOMON.

EXEC

O EXEC é um comando muito especial. Ele permite dar o controle do Apple II para um arquivo de texto. O comando EXEC se parece com:

```
EXEC NOMEARQ,R6,S5,D2,V23
```

O parâmetro R é similar ao parâmetro de registro do comando POSITION. Ele referencia o número de campo relativo ao arquivo onde o processamento inicia na execução do EXEC. Como o EXEC abre o arquivo e coloca o ponteiro no início do arquivo EXEC, o parâmetro R referencia um número de campo absoluto dentro do arquivo. R0 é a condição por ausência de informação, que inicia a execução no início do arquivo. R1 inicia a execução no segundo campo.

As especificações de registro relativo, slot, drive e volume são opcionais e podem ocorrer em qualquer ordem.

Quando é executado um EXEC, o arquivo especificado é aberto, e as declarações READ e INPUT implícitas são executadas. Se não houver nenhum parâmetro R presente, a primeira linha no arquivo é tomada. Se houver R, então a linha especificada pelo R é tomada.

A linha tomada é tratada como se ela fosse digitada no teclado em modo imediato. Se a linha não tiver significância, aparecerá a mensagem ?SYNTAX ERROR ou *** SINTAX ERR. Se for uma linha válida, como:

```
100 PRINT "ISTO É UM TESTE"
```

será guardada na memória, como se tivesse sido digitada. Se for um comando direto, como LIST ou RUN, será executada.

Após a execução do comando da primeira linha, a linha seguinte é lida e executada. Assim o processo continua até que o fim do arquivo seja encontrado, quando o controle do Apple II retorna ao teclado.

Suponha um arquivo de texto chamado BUBBA que contenha as seguintes linhas:

```
PRINT "EU TENHO CONTROLE SOBRE SEU APPLE ! ! !"
FP
100 GR
200 FOR I = 0 TO 39
300 FOR J = 0 TO 39
400 COLOR=RND(0)*15
500 PLOT I,J
600 NEXT J
700 NEXT I
800 FOR I = 1 TO 5000 : NEXT I
900 TEXT : CALL -936
999 END
LIST

FOR I = 1 TO 5000 : NEXT I
RUN

NEW
PRINT "TERMINOU.AQUI ESTA O CATALOGO DO DISCO:"
CATALOG
```

Quando for digitado o comando:

```
EXEC BUBBA
```

várias coisas vão acontecer. Primeiro a mensagem será mostrada. Depois o Applesoft será chamado. Depois, um pequeno programa será carregado na memória e listado. Após uma breve pausa o programa será rodado, e então tirado da memória. Outra mensagem aparece, seguida pelo catálogo do disco corrente.

Notas sobre o EXEC

O EXEC tem várias facetas interessantes que é bom conhecer.

Apenas um arquivo EXEC pode ser aberto de uma vez. Se um arquivo que está sendo acessado pelo EXEC contém um comando EXEC, o primeiro arquivo será fechado e o segundo pegará o controle. Depois de um arquivo EXEC executar um programa, o próximo campo (isto é, linha) é executado. Se o programa é abortado com CTRL-C, o EXEC não continuará.

Se um programa que está rodando sob um comando EXEC contém uma declaração INPUT, a entrada é feita com o campo seguinte do arquivo EXEC. Isso causa um problema, se o próximo campo é um comando do DOS em modo imediato (e não uma linha de programa). O comando será executado no lugar de ser interpretado como dado para a declaração INPUT.

Se um programa executa um comando CLOSE, ou se o arquivo EXEC contém um comando CLOSE em modo imediato, o arquivo EXEC não será fechado.

Usando o EXEC para Converter um Programa de um BASIC para Outro

O EXEC pode ser usado para converter programas entre o Integer BASIC e o Applesoft. Se a conversão for do Applesoft para o Integer BASIC, deve-se primeiro editar o programa Applesoft para adaptar a sintaxe do Integer BASIC. Deve-se então guardar o programa como arquivo de texto. Para isso, ponha algumas linhas no programa para criar o arquivo de texto, execute o comando WRITE e então liste o programa. A listagem vai para o arquivo de texto. Aqui estão as declarações necessárias:

```
 1 POKE 33,33: REM DESLIGA A FORMATACAO "LIST"
 2 D$ = "": PRINT D$"OPEN FILE": PRINT D$;"WRITE FILE"
 3 LIST 10,32767
 4 PRINT D$; "CLOSE": END
 5 POKE 33,40: REM REFAZ FORMATO "LIST"
10 REM SEM PROGRAMA INICIA AQUI
```

Quando se roda o programa, apenas as linhas 1, 2, 3, 4 e 5 são executadas. Elas abrem um arquivo e guardam o programa listado nele. Quando terminar, pode-se trocar para o outro BASIC (com FP ou INT, por exemplo). Digite então:

```
EXEC FILE
```

As linhas do programa velho são carregadas do disco para a memória. Se ainda não foi feito, deve-se editar o programa para corrigir as partes que ainda não estão de acordo com as regras da nova linguagem.

MAXFILES

O MAXFILES permite especificar o número máximo de arquivos que podem estar abertos a qualquer tempo. Cada arquivo tem 595 bytes de memória reservados para seu uso como buffer. Existem apenas duas seções de 256 bytes em cada buffer, uma para leitura e outra para gravação. Os 83 bytes restantes são usados para controle, como lista de trilha/setor.

Quando um arquivo é aberto, ou ocorre uma operação em disco tal como CATALOG ou LOCK, o DOS lê 256 bytes de informação do disco e a coloca num buffer. Se a informação é alterada e deve ser colocada de volta no disquette, 256 bytes são copiadas do buffer e transferidas para o disquette.

O exemplo seguinte especifica oito arquivos no máximo:

```
MAXFILES 8
```

O número de arquivos deve ser um inteiro entre 1 e 16. Quando o DOS é carregado, são alocados três buffers. O MAXFILES pode ser empregado se for necessário usar mais de três arquivos simultaneamente. E pode ser também usado para menos se os bytes de buffer precisarem ser usados para sua aplicação específica.

Um buffer é usado para executar qualquer um dos comandos do DOS seguinte:

APPEND	FP	POSITION
BLOAD	INIT	READ
BRUN	INT	RENAME
BSAVE	LOAD	RUN
CATALOG	LOCK	SAVE
CHAIN	MAXFILES	UNLOCK
CLOSE	MON	VERIFY
DELETE	NOMON	WRITE
EXEC	OPEN	

Dessa forma, se se abrem arquivos acima do limite e se dá um comando CATALOG, o erro NO BUFFER AVAILABLE é mostrado. Não é requerido buffer para comandos usados fora do contexto do DOS (por exemplo, comando LOAD de cassette).

Quando o MAXFILES é trocado, os programas no Integer BASIC são zerados e as séries de caracteres no Applesoft são destruídas. Por isso deve-se executar o MAXFILES antes de carregar e executar qualquer programa.

Pode-se executar o MAXFILES dentro de um programa no Applesoft se precedido por um caractere CTRL-D numa declaração PRINT. O MAXFILES faz as instruções GOTO, GOSUB e outras do tipo, funcionarem incorretamente, a menos que seja a primeira declaração num programa. Para evitar destruição de séries, use o MAXFILES como abaixo:

```
]11 REM PRIMEIRO ATRIBUICAO MAXFILES
]12 PRINT CHR$(4);"MAXFILES 9"
]13 REM PROGRAMA NORMAL COMECA AQUI
]14 D$=CHR$(4):REM CARACTERE CTRL-D
```

Só se pode usar MAXFILES em Integer BASIC no contexto de um arquivo EXEC. Por exemplo, o arquivo EXEC contém três declarações em modo pseudo-imediato:

```
MAXFILES 8
LOAD PROGRAM
RUN 10
```

Ele é chamado num programa como segue:

```
5  PRINT D$;"EXEC MXF": REM MANDA MAXFILES
   CONTINUA NA LINHA 10
10 REM PROGRAMA COMECA AQUI
```

USANDO INSTRUMENTOS DE DEPURAÇÃO DO DOS

Como os programas que acessam arquivos em disco são em geral de alguma complexidade, o DOS tem três comandos que ajudam a depurar programas: MON, NOMON e TRACE.

MON

MON é abreviação de monitor. Ele permite monitorar as informações que vão e vêm do disco. O MON usa três parâmetros. O exemplo seguinte mostra todos os três:

```
MON C,I,O
```

Os parâmetros do MON especificam o tipo de informação a ser mostrado. C causa a monitoração dos comandos de disco. I causa a mostra das entradas que vêm do disco. O causa a mostra das saídas para o disco.

Os parâmetros podem aparecer em qualquer ordem, e em qualquer combinação. No mínimo um parâmetro deve estar presente ou o comando é ignorado.

O MON permanece operando até que um NOMON, INIT ou FP seja executado, ou o DOS recarregado.

NOMON

O NOMON cancela o efeito do comando MON. Ele usa os mesmos três parâmetros que o MON usa, porém com o NOMON eles especificam quais dados não devem ser monitorados. Por exemplo, assumindo que foi dado o MON I, O, C anterior, o comando:

```
NOMON O
```

cancela a monitoração da saída para disco. A entrada do disco e os comandos do DOS continuarão sendo mostrados.

Usando o Comando TRACE

O comando TRACE (veja Capítulo 4) é usado na depuração de programas. Mas como ele mostra os números das linhas sem *carriage return*, os comandos do DOS ficam concatenados aos números das linhas, e ignorados. O remédio para isso é simples. Um *carriage return* deve ser colocado antes de cada comando do DOS. Para isso troque a linha:

```
100 D$ = CHR$ (4) : REM CTRL-D
```

para ler:

```
100 D$ = CHR$ (13) + CHR$ (4): REM RETURN + CTRL-D
```

Assim, sempre que o PRINT D$ ocorrer, o caractere CTRL-D será precedido por um *carriage return*, separando-o da execução do comando TRACE comum.

Entretanto, outro problema existe. Se um comando WRITE está em processo, todos aqueles números de linhas serão guardados no arquivo. (Desculpem, este problema não pode ser contornado.)

LINGUAGEM DE MÁQUINA
ARQUIVOS DE DISCO (IMAGEM BINÁRIA)

O Disk II suporta arquivo em linguagem de máquina e imagem binária (gráficos). Esses arquivos são mostrados com a letra B como código de tipo no catálogo do disco.

Podem ser guardadas imagens tanto de baixa quanto de alta resolução em disquette para posterior recuperação e uso.

Os programas em linguagem de máquina podem ser carregados e rodados diretamente, ou podem ser chamados por programas em BASIC usando a declaração CALL ou a função USR.

O DOS tem três comandos especialmente projetados para arquivos binários. São eles o BSAVE, BLOAD e BRUN. O efeito de cada um deles é o mesmo que o de seus equivalentes não binários (BSAVE=SAVE, BLOAD=LOAD, BRUN=RUN).

BSAVE

BSAVE como o nome indica, guarda a imagem binária no disco. Aqui vai um exemplo:

```
BSAVE NOMEARQ,A$78,L21,S6,D2,V2
```

Note que existem dois parâmetros não encontrados em outros comandos do DOS. Veja também que esses parâmetros *não* são opcionais; eles devem ser especificados. Os parâmetros slot, drive e volume são opcionais, como de costume.

O parâmetro A é o *endereço*; ele indica a posição de memória inicial da imagem binária a ser guardada. O endereço pode ser uma constante decimal ou hexadecimal. Os valores hexadecimais devem ser precedidos pelo sinal dólar ($). Os valores decimais devem estar entre 0 e 65535. Valores negativos são proibidos.

O parâmetro L especifica o *comprimento* da imagem binária a ser gravada. O comprimento é expresso em número de bytes. Pode ser um número decimal ou hexadecimal, com os valores hexadecimais precedidos pelo sinal de dólar ($). O comprimento vai de 1 a 32767 ou a mensagem SYNTAX ERROR será gerada. 32767 é o maior tamanho que o DOS pode guardar num único campo. Devem ser dados dois BSAVE para guardar mais de 32767 bytes.

Se as posições de memória na faixa especificada não existem fisicamente, nenhum erro é devolvido. Não é muito interessante especificar posições fora da faixa de memória instalada no seu Apple II (exemplo, 49151 ou $BFFF em um sistema de 48K bytes).

BLOAD

O BLOAD recupera o conteúdo de arquivos binários e os carrega na memória. O comando BLOAD tem a seguinte aparência:

```
BLOAD NOMEARQ,A378,S6,D2,V6
```

O parâmetro endereço é opcional no comando BLOAD. Se ausente, a imagem é carregada iniciando no endereço especificado quando a imagem foi gravada.

Os programas em linguagem de máquina podem não funcionar corretamente se forem carregados em endereços de memória errado.

Diferentemente do comando LOAD, o BLOAD não zera programas ou dados a menos que eles residam nas posições de memória onde a imagem será carregada. Apenas aquelas posições são afetadas pelo BLOAD; outras posições fora da faixa do BLOAD não são afetadas.

Não é mostrado nenhum tipo de erro se o endereço referenciado para o BLOAD é uma área de ROM. A área de ROM no caso não será afetada, é claro.

BRUN

BRUN é idêntico ao BLOAD exceto que após o arquivo ser carregado, o BRUN executa um salto em linguagem de máquina para o endereço inicial do programa. Se nenhum endereço é dado, o salto é para o endereço dado quando da gravação do programa. Aqui segue um exemplo do BRUN:

```
BRUN NOMEARQ,A378,S6,D2,V6
```

Nunca use o BRUN com imagem gráfica, pois o resultado é imprevisível.

6

Gráficos e Som

O Apple II tem capacidade de vídeo gráfico e geração de som. Juntas, essas características dão outra dimensão aos programas que se usam e que se escrevem. Este capítulo é dirigido aos iniciantes que estão aprendendo BASIC (talvez apenas lendo este livro), bem como a programadores em assembler. Os gráficos não são difíceis de serem montados, especialmente em linguagem de alto nível. O Monitor do Apple II, com suas sub-rotinas em linguagem assembler incorporadas, ajuda ainda mais os programadores em linguagem assembler a usarem gráficos e o alto-falante interno do equipamento. O intuito deste capítulo é que, ao terminá-lo, tenha-se sólidos conhecimentos no emprego destes elementos nos programas.

GRÁFICOS DE BAIXA RESOLUÇÃO

O Apple II tem duas áreas de memória separadas disponíveis para gráficos de baixa resolução. Estas duas áreas são chamadas *páginas.* Ambas as páginas podem aparecer na tela como gráfico de vídeo com 40 colunas por 48 linhas, como mostrado na Figura 6.1.

Cada coordenada (interseção de linha e coluna) aparece como um pequeno retângulo na tela. Cada página contém 1920 coordenadas (40 colunas vezes 48 linhas), e pode-se atribuir uma das 16 cores a cada coordenada em cada página. A Tabela 6.1 mostra os tons disponíveis. Não é necessário conhecer os fundamentos do funcionamento interno do Apple II para usar gráficos de baixa resolução; apenas o conhecimento de programação em BASIC (e um pouco de coordenadas gráficas) é suficiente para iniciar.

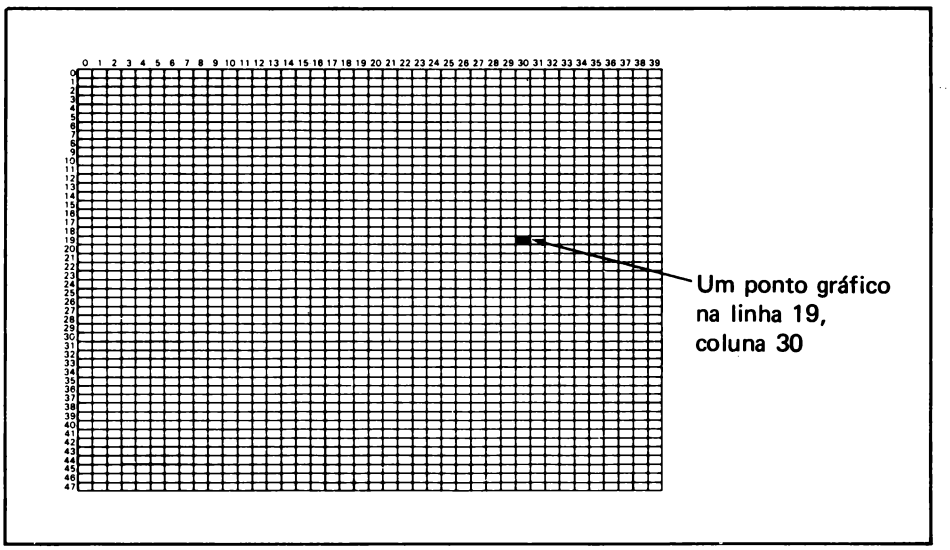


FIGURA 6.1. Tela de Gráfico de Baixa Resolução.

TABELA 6.1. Cores de Gráficos em Baixa Resolução.

Cor	Número	Cor	Número
Preto	0	Marrom	8
Magenta	1	Laranja	9
Azul Escuro	2	Cinza #2	10
Púrpura	3	Rosa	11
Verde Escuro	4	Verde Claro	12
Cinza #1	5	Amarelo	13
Azul Médio	6	Água	14
Azul Claro	7	Branco	15

MONTANDO A PÁGINA GRÁFICA

A página dedicada a gráficos de baixa resolução também dobra como a página de texto para o Apple II. Dentro do BASIC, muda-se para modo gráfico a partir do modo texto através da declaração:

```
GR
```

Uma vez executada a declaração, a tela fica preta à exceção das quatro linhas inferiores da tela que recebem texto. Essa área inferior da tela é chamada *janela de texto*. Com a janela de

texto na parte inferior da tela, existe espaço para 40 das 48 linhas disponíveis no modo gráfico de baixa resolução. Pode-se usar esta declaração mais de uma vez dentro de um programa, mesmo estando dentro do modo gráfico de baixa resolução, o que resulta na limpeza da tela.

Gráficos de Tela Inteira

Após executar a declaração GR, pode-se eliminar a janela de texto e permitir que as oito últimas linhas sejam usadas para gráfico, digitando:

```
POKE -16302,0
```

A janela de texto desaparece, substituída por área de gráfico.

Refazendo a Janela de Texto

Se o Apple II estiver no modo gráfico total, pode-se fazer voltar a janela de texto de duas formas. Para limpar a tela gráfica e voltar a janela de texto ao mesmo tempo, use a declaração GR. Para voltar a janela de texto sem alterar as primeiras 40 linhas de gráfico, entre a declaração:

```
POKE -16301,0
```

Uma vez executada, esta declaração repõe a janela de texto em baixo da tela.

Voltando ao Modo Texto em toda a Tela

Para deixar o modo gráfico de baixa resolução e voltar ao modo texto normal, use a declaração BASIC:

```
TEXT
```

isso faz voltar de gráfico para caracteres. Enquanto a declaração GR limpa a tela quando executada, o TEXT não limpa. Lembre-se que texto e gráfico usam a mesma área de memória. Quando esta declaração é executada, provavelmente a tela aparecerá cheia de caracteres ímpares; isso é porque o Apple II estará interpretando os caracteres gráficos como texto. Limpe a tela com a seqüência Esc-@, com a declaração CALL-936 ou, em Applesoft, com o comando HOME.

DECLARAÇÕES DE PROGRAMAÇÃO DE GRÁFICOS

Tanto o Integer BASIC quanto o Applesoft reconhecem comandos gráficos de baixa resolução para desenhar coordenadas na tela, trocar as cores usadas nos desenhos, e traçar linhas horizontais e verticais de vários tamanhos. Estes comandos operam apenas em gráficos de baixa resolução da página 1 (também chamada página primária). Se for usada a página 2, várias dessas declarações que poupam tempo serão eliminadas e deve-se recorrer às declarações não específicas, como PEEK, POKE e CALL para resolver os problemas.

A Declaração COLOR

Na Tabela 6.1, cada cor listada tem um número correspondente entre 0 e 15. Esse é o número usado na declaração COLOR para indicar a cor atual do gráfico de baixa resolução. Por exemplo:

```
COLOR=13
```

indica que a cor do desenho será amarela. Se não for indicada a cor a ser usada, o Apple II usa o preto, equivalente a COLOR=0, a cor adotada por ausência. Embora se possa especificar um número de cor entre 0 e 255 sem gerar erro de sintaxe, o COLOR toma apenas o *nibble* de menor valor selecionado. Dessa forma, se for dado COLOR=222 (igual a DE hexadecimal), será entendido COLOR=15 (igual a 0E hexadecimal).

A Declaração PLOT

Esta declaração BASIC coloca um único ponto gráfico — um pequeno retângulo — na tela do Apple II na coordenada que se especifica.

A declaração:

```
PLOT 23,18
```

ilumina um ponto na 24ª linha e 19ª coluna na cor indicada pela última declaração COLOR executada. O número da linha vai de 0 a 47 e o da coluna de 0 a 39. Excedendo estes limites na declaração PLOT, aparece uma mensagem de erro, e o programa pára. Como em qualquer outra declaração de gráfico em baixa resolução (exceto GR), podem-se trocar expressões literais por variáveis:

```
PLOT Y/2+12,X-4
```

Um Exemplo de "Plotagem"

O exemplo seguinte em Integer BASIC usa todas as declarações gráficas de baixa resolução vistas neste capítulo até aqui. O objetivo é desenhar uma linha diagonal a partir do canto superior esquerdo até o ponto direito mais baixo possível.

```
10 REM TRACA UMA LINHA DIAGONAL
11 REM PELA TELA EM BAIXA RESOLUCAO
20 GR
30 COLOR= RND (16)
40 FOR Y=0 TO 39
50 PLOT Y,Y
60 NEXT Y
70 GOTO 30
```

A tela se parecerá com a Figura 6.2, a menos da linha diagonal, que mudará de cor aleatoriamente.

Para mudar o programa para Applesoft, troque a linha 30 como visto abaixo.

```
30 COLOR= RND (16) * 16
```

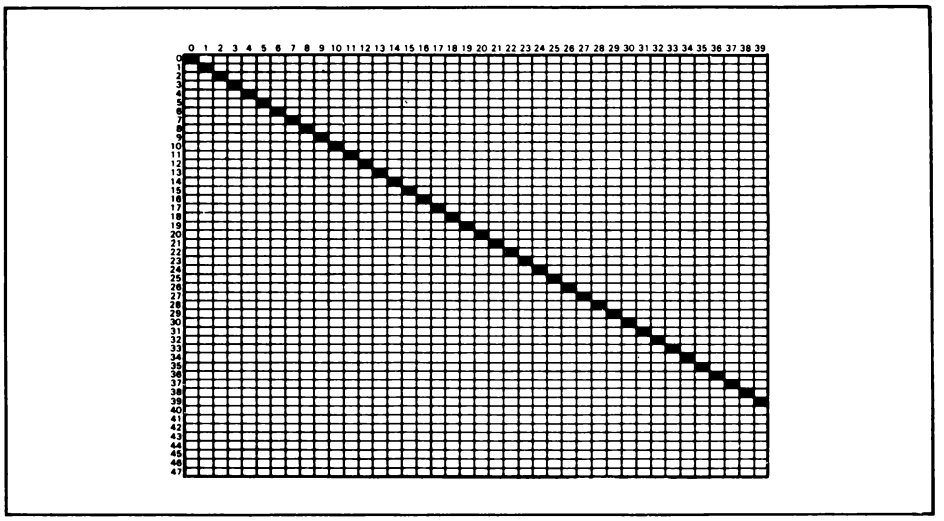


FIGURA 6.2. Exemplo de Desenho em Baixa Resolução.

Traçando Linhas Horizontais

O comando HLIN permite desenhar linhas de vários comprimentos da esquerda para a direita na página gráfica de baixa resolução. Esta declaração:

```
HLIN 0,39 AT 0
```

desenha uma linha horizontal na parte mais alta da tela, da margem esquerda até a margem direita. HLIN lembra Horizontal Linha. A forma geral desta declaração é colocar a coluna esquerda seguida por vírgula, seguida pela coluna mais à direita a ser ocupada pela linha (neste caso 39), a palavra AT, e a linha onde será desenhada a reta horizontal.

Os números de coluna da esquerda e direita colocados não podem ser negativos e devem ser menores que 256; o número da coluna da direita não pode ser menor que o da esquerda. O parâmetro à frente do AT não pode ser negativo nem maior que 48. Se algum desses parâmetros exceder o limite, aparece uma mensagem de erro, interrompendo a execução. Se for especificada coluna maior que 39 na declaração HLIN em Integer BASIC podem aparecer resultados imprevistos. O Applesoft fornece uma mensagem de erro. Por exemplo, execute a declaração:

```
10 GR
20 COLOR=12
30 HLIN 45,100 AT 0
```

Em Integer BASIC, isso causa o aparecimento de duas barras verdes. A primeira barra não está na linha 0 (a mais alta da tela) como deveria estar, e a barra continua por várias linhas tela abaixo. Em geral não é bom fazer esses valores excederem os limites de posição na tela em baixa resolução.

Traçando Linhas Verticais

A declaração VLIN (de Vertical Linha) desenha uma reta na cor selecionada, de uma linha até outra, numa coluna especificada.

Por exemplo:

```
VLIN 12,30 AT 33
```

desenha uma reta da linha 12 até a linha 30, na coluna 33. Os valores dos parâmetros do VLIN são idênticos aos de HLIN: a primeira e segunda expressões devem ser números inteiros entre 0 e 255 inclusive; o segundo parâmetro não pode ser menor que o primeiro, e o último parâmetro (referente ao número de coluna) deve ser inteiro entre 0 e 39 inclusive. Se qualquer desses parâmetros estiver fora da faixa permitida, aparece uma mensagem de erro na tela do Apple II.

Usando HLIN e VLIN num Programa

Um outro programa (listado abaixo) ajuda a compreender o uso de HLIN e VLIN. Aqui, são traçadas linhas aleatórias em cores aleatórias na tela. Para parar o programa, digite CTRL-C.

```
10 REM USO DE VLIN E HLIN
20 GR : USA PAGINA GRAFICA 1
30 POKE -16302,0: REM DET.PAGINA INTEIRA
40 CALL -1998: REM LIMPA TODAS AS 48 LINHAS
50 REM  INICIO DO PROGRAMA
60 COLOR= RND (16): REM SELEC.COR ALEATORIA
70 HLIN 0,RND (40) AT RND (48)
80 COLOR= RND (16)
90 VLIN 0, RND (48) AT RND (40)
100 GOTO 60
```

A Declaração SCRN

A declaração SCRN é um tanto mais sutil que as outras declarações em baixa resolução. Suponha que se deseje que o computador informe que cor está desenhada num certo ponto da tela. O SCRN faz isso. Esta declaração:

```
X=SCRN(12,24)
```

atribua à variável X o número da cor na coordenada entre parênteses (neste caso, linha 13 e coluna 25). A cor devolvida para a variável está entre 0 e 15; o número corresponde a uma das cores mostradas na Tabela 6.1. Por exemplo, se forem dadas as seguintes declarações em modo imediato:

```
GR
COLOR=14
PLOT 12,12
PRINT SCRN (12,12)
```

O Apple II responde:

14

Esta declaração pode ser muito útil ao se escrever programas avançados em baixa resolução.

GRÁFICOS DE ALTA RESOLUÇÃO

O aspecto mais atraente do Apple II é sua capacidade de fazer gráficos em alta resolução. Como no caso dos de baixa resolução, existem duas áreas ou páginas disponíveis para uso. Mas é apenas nisso que reside a similaridade. A resolução neste modo é de 280 linhas horizontais por 192 posições verticais, um incremento de 7 vezes na vertical e 4 na horizontal sobre o modo de baixa resolução. Entretanto, existe menor número de cores possível e pode-se desenhar linhas muito mais finas na tela.

As funções internas disponíveis para gráficos em alta resolução existem no Applesoft. O Integer BASIC não tem comandos específicos desse tipo. No entanto, não importando a linguagem usada no Apple II, existe a capacidade gráfica de alta resolução porque o Apple II usa parte da memória para guardar pontos, linhas e formas de alta resolução, e usa programas internos para interpretar e apresentar na tela esses dados. Certas declarações internas do Applesoft usam os programas internos automaticamente e a memória da tela; essa parte será vista em primeiro lugar. Mais adiante veremos como incorporar gráficos de alta resolução também aos programas em Integer BASIC. Como será visto adiante, existem algumas funções de alta resolução que só são possíveis em Integer BASIC.

QUE PÁGINA DEVE SER USADA?

Existe alguma dificuldade em se usar gráficos de alta resolução, e elas envolvem a quantidade de memória de que o sistema dispõe. Se estiver sendo usado o Applesoft *firmware* (isto é, em ROM no Apple II Language System), pode-se usar a página 1 de alta resolução *se* o computador possui mais de 16K, e pode-se usar a página 2 *se* o Apple II tem 24K ou mais. Ponha mais 12K se for usado o Disk Operating System (DOS) e o gráfico de alta resolução do Applesoft em conjunto.

Se for usada a versão do Applesoft em cassette ou disco, não se pode usar a página 1, por esse espaço ser empregado no interpretador do Applesoft. Ao se tentar usar a página gráfica 1 de alta resolução nesse caso, estará comprometida ou completamente perdida a possibilidade de programação em Applesoft bem como o programa BASIC presentemente na memória. *Pode-se* usar a página 2 se o Apple II tiver pelo menos 24K, ou 36K se o DOS estiver presente ao mesmo tempo.

Reservando Memória para Gráficos de Alta Resolução

O Apple II não reserva automaticamente espaço de memória para gráficos de alta resolução. À medida que o programa vai ficando maior, por aumento do número de declarações ou de variá-

veis, a chance de destruir páginas gráficas aumenta. A forma de resolver este problema é determinar dois ponteiros de memória, HIMEM: e LOMEM:, os valores que vão proteger a página ou páginas gráficas de alta resolução usadas. Esses ponteiros agem como limites que o programa não pode ultrapassar, permitindo assim a existência de até duas áreas de memória que podem ser usadas livremente.

O uso do HIMEM: e do LOMEM: difere entre o Applesoft e o Integer BASIC. Veja as diferenças no compêndio de BASIC, localizado no Capítulo 8. Veja também a Figura G.1 no Apêndice G que mostra o uso de memória em forma de desenho. Os três parágrafos que seguem ficarão mais fáceis de serem entendidos se forem observadas essas indicações.

Se for desejado usar página gráfica de alta resolução 1 (lembre-se de que é necessário ter-se o Applesoft em firmware para isso), tendo apenas um Apple II com 16K, ponha HIMEM: em 8191 e não mexa no LOMEM. Isso fará com que o programa em Applesoft se mantenha abaixo da posição de memória 8191, o que é uma severa porém necessária restrição, desde que se necessita 8K para memória gráfica de alta resolução. Se for o caso de se usar página 1 num Apple II com mais de 16K, pode-se também não mexer no HIMEM: e colocar o LOMEM: com 16384. Isso colocará o programa acima da página gráfica de alta resolução 1. CUIDADO: Não use este esquema em conjunto com o DOS a menos que o sistema tenha mais de 32K de memória. Caso contrário, use o primeiro esquema, que mantém o programa abaixo da página 1.

Se se deseja usar a página 2 (possível apenas com 24K ou mais), ponha LOMEM: em 24576, ou HIMEM: em 16383 deixando LOMEM: inalterado. Ajustando LOMEM: coloca-se o programa do Applesoft *acima* da página gráfica 2. Para usar o DOS ao mesmo tempo, é necessário no mínimo 48K de memória. Se for ajustado o HIMEM: o programa residirá *abaixo* da página gráfica 2. Neste último caso, em se usando o interpretador Applesoft não-firmware (a partir do cassette ou disco), o programa em Applesoft ficará no espaço acima do interpretador e abaixo da página gráfica 2, que é um espaço apertado.

No caso de usar ambas as páginas para gráfico de alta resolução (possível apenas com o Applesoft em firmware), ponha LOMEM: em 24576 e HIMEM: em 8191. Se o programa em Applesoft é colocado acima das páginas gráficas, ajustando o LOMEM: e deixando o HIMEM: então é necessário no mínimo 48K para usar o DOS e ainda ter uma quantidade de memória razoável para o programa em Applesoft.

ACESSANDO O DISPLAY GRÁFICO

Embora haja disponibilidade para duas páginas gráficas de alta resolução, o Applesoft em disco ou cassette usa parte da página 1 para armazenar a linguagem (o interpretador). Tendo o Apple II Plus ou o standard, ambos com o cartão Applesoft Firmware ou o Language System, pode-se usar a página 1 sem receio de alterar o BASIC. Para efeito de compatibilidade com outros computadores Apple II, pode-se usar apenas a página 2 em qualquer caso.

Em Applesoft, a declaração:

```
HGR
```

limpa a tela e mostra a página gráfica 1, com quatro linhas de texto na parte baixa. Pode-se ter gráfico e texto na tela com este modo usando declarações PRINT em Applesoft para mostrar texto na janela de texto. Entretanto, uma vez executado o HGR pelo Apple II, a tela apenas mostra 160 das 192 linhas gráficas horizontais de alta resolução disponíveis. Para mostrar a tela gráfica

inteira, execute POKE-16302,0 após o HGR. Isso eliminará a janela de texto trocando-a pelas 32 linhas gráficas restantes.

Para mostrar a página de alta resolução 2, use a declaração:

```
HGR2
```

A execução do HGR2 limpa a tela e mostra todas as 192 linhas da página gráfica de alta resolução 2, sem deixar janela gráfica em baixo da tela. Para abrir a janela de texto use POKE-16301,0 após o HGR2. A janela de texto é mais difícil de se usar na página gráfica 2. O texto que aparece na janela provém da página de texto secundária. O BASIC só pode acessar essa página através de declarações POKE (não via PRINT), o que é um limite. Entretanto, a segunda página de texto não é protegida contra escrita no BASIC como a primeira, de forma que é preciso colocar LOMEM: em 3071 ou mais.

ALTERNATIVAS PARA O HGR E HGR2

A principal desvantagem em se usar HGR e HGR2 é que em se executando qualquer dessas declarações limpa-se a página gráfica de alta resolução, quer se queira ou não. Além disso, essas declarações não estão disponíveis em Integer BASIC. Pode-se usar declarações PEEK, POKE e CALL para tornar as páginas gráficas mais flexíveis, e elas podem ser úteis não importando qual a linguagem usada no Apple II.

Outra Forma de Gerar Display Gráfico

É possível ir-se para o modo gráfico de alta resolução sem limpar a tela. Pode-se comparar este método ao acionamento de chaves. Teoricamente, o que se faz é exatamente isso, numa área de memória reservada, chamada *chave de software*, que se localizam de −16304 até −16297 ($C050 até $C057). A Figura E.1 do Apêndice E ilustra as chaves disponíveis. Para manter essas declarações compatíveis com Integer BASIC e Applesoft, usamos a representação inteira negativa dessas posições de memória, (ou seja, −16304 ao invés de 49231).

Para mostrar a página gráfica de alta resolução 1 sem limpar o conteúdo anterior, execute as seguintes declarações:

```
POKE -16304,0   Entra no modo gráfico
POKE -16297,0   Entra no modo alta resolução
POKE -16300,0   Seleciona página 1 de alta resolução
                  (Necessário apenas quando se troca da página 2 para a 1)
```

Tente essas declarações em modo imediato como experiência.

Para mostrar a página 2 de alta resolução, sem alterar o conteúdo anterior, entre as seguintes declarações:

```
POKE -16304,0   Necessário apenas se modo gráfico ainda não foi selecionado
POKE -16297,0   Necessário apenas se modo alta resolução ainda não foi selecionado
POKE -16299,0   Seleciona página gráfica de alta resolução 2
```

BASIC Normal após Gráfico em Alta Resolução

Em Integer BASIC, as declarações TEXT e GR podem ser insuficientes para limpar a tela. Em usando a página gráfica 2, deve-se re-selecionar explicitamente a página 1 com uma declaração POKE-16300,0. De outra forma o que se verá será a página de memória 2 de texto e gráfico em baixa resolução. Isso pode ser especialmente confuso em modo texto. O teclado parece não ter função pois tudo o que se digita vai para a página de memória 1, enquanto o que está sendo mostrado é a memória da página 2.

A declaração GR não mudou de alta para baixa resolução gráfica em Integer BASIC. Ela apenas seleciona modo gráfico com a janela de texto de quatro linhas (em oposição a modo texto inteiro). Deve-se selecionar explicitamente gráfico em baixa resolução com uma declaração POKE-16298,0.

Limpando as Páginas de Alta Resolução

Usando gráficos de alta resolução em Applesoft, HGR e HGR2 limparão a página sempre que a declaração é executada. Entretanto, sob Integer BASIC não há declaração simples que execute esta função. A sub-rotina seguinte usa uma função interna do Monitor para limpar a tela de alta resolução:

```
18990 REM ************************************
18991 REM * LIMPA TELA DE ALTA RESOLUCAO      *
18992 REM * (USA SUBROTINA "MOVE" DO MONITOR  *
18993 REM * EM $FE2C PARA MOVER DADOS RAPIDA- *
18994 REM * MENTE PELA AREA DE ALTA RESOLUCAO *
18995 REM * ------------------------------------ *
18996 REM * ENTRE EM PAGINA 1 OU 2; A ROTINA  *
18997 REM * FAZ O RESTO.                      *
18998 REM ************************************
19000 INICIO=32
19010 IF PAGINA=2 THEN INICIO=64
19020 POKE 60,0
19030 POKE 61,INICIO
19040 POKE 62,254
19050 POKE 63,INICIO+33
19060 POKE 66,1
19070 POKE 67,INICIO
19080 POKE -16304,0
19090 POKE -16297,0
19100 IF PAGINA=1 THEN POKE -16300,0
19110 IF PAGINA=1 THEN POKE -16299,0
19120 POKE INICIO*256,0
19130 CALL -468
19140 RETURN
```

Esta sub-rotina em BASIC move zeros para a página gráfica em alta resolução selecionada. Se a variável PÁGINA é posta em 1, a sub-rotina limpa a página 1; se posta em 2, a página 2 é

limpa. Esta rotina funciona apenas em Integer BASIC; entretanto, não considere isso uma desvantagem. O Applesoft executa esta função de forma muito mais eficiente com HGR e HGR2.

CORES EM ALTA RESOLUÇÃO

São disponíveis oito cores em modo gráfico de alta resolução, porém apenas quatro cores diferentes são permitidas, mais o preto e branco. A Tabela 6.2 mostra essas cores e seus números correspondentes (usados na seleção da cor).

TABELA 6.2. Cores Gráficas em Alta Resolução.

Cor	Número	Cor	Número
Preto	0	Preto	4
Verde	1	Laranja	5
Violeta	2	Azul	6
Branco	3	Branco	7

A Declaração HCOLOR

A declaração HCOLOR do Applesoft seleciona uma das oito cores disponíveis para uso em alta resolução. Diferente da declaração COLOR em gráficos de baixa resolução (que permite especificar números maiores que aqueles alocados para cores), o HCOLOR não aceita cor maior que 7. Se uma cor fora do limite é especificada, o programa pára abruptamente com a mensagem ?ILLEGAL QUANTITY ERROR. O HCOLOR não troca a cor de gráficos já presentes na tela, nem tem qualquer efeito em gráficos de baixa resolução.

Determinando a Cor de Fundo em Alta Resolução

Com uma pequena modificação, a sub-rotina apresentada antes de limpa a tela irá preenchê-la inteiramente com uma das cores disponíveis. A sub-rotina abaixo faz isso. Antes de chamá-la, determine a página gráfica através do uso da variável PÁGINA. Ainda, determine o número da cor pela variável HCOR. O programa que chama deve atuar nas chaves de software para modo gráfico de alta resolução. Esta sub-rotina opera apenas em Integer BASIC.

```
18990 REM **********************
18991 REM * GERA COR DE FUNDO  *
18992 REM * ------------------ *
18993 REM * DET.PAGINA=1 OU 2  *
18994 REM * E HCOR COM A COR   *
18995 REM * EQUIVALENTE   DO   *
18996 REM * APPLESOFT.         *
18997 REM **********************
```

```
19000 INICIO=32
19010 IF PAGINA=2 THEN INICIO=64
19020 POKE 60,0
19030 POKE 61,INICIO
19040 POKE 62,253
19050 POKE 63,INICIO+33
19060 POKE 66,2
19070 POKE 67,INICIO
19075 Y1=85: X1=42
19080 IF HCOR>0 AND HCOR<>4 THEN 19090:X1=0:Y1=0
19090 IF HCOR<>3 AND HCOR<>7 THEN 19100:X1=127:Y1=127
19100 IF HCOR=1 OR HCOR=5 THEN 19120
19110 X3=X1:X1=Y1:Y1=X3
19120 IF HCOR<4 THEN 19140
19130 Y1=Y1+128:X1=X1+128
19140 POKE INICIO*256,X1
19150 POKE INICIO*256+1,Y1
19160 IF PAGINA=1 THEN POKE -16300,0
19170 IF PAGINA=2 THEN POKE -16299,0
19180 CALL -468
19190 RETURN
```

DESENHANDO PONTOS E LINHAS

Uma vantagem poderosa dos gráficos de alta resolução em Applesoft é sua habilidade de desenhar linhas de qualquer ângulo bem como pontos isolados e linhas horizontais e verticais. A declaração HPLOT pode ser usada de três formas:

```
HPLOT 12,12
```

Isso desenha um ponto na página de alta resolução corrente na interseção da décima-terceira linha com a décima-terceira coluna, na cor de alta resolução selecionada.

O segundo uso do HPLOT é:

```
HPLOT 0,0 TO 279,191
```

Esta declaração desenha uma linha diagonal do ponto mais alto à direita até o mais baixo canto à esquerda na tela. Usando HPLOT com dois conjuntos de coordenadas como mostrado acima, pode-se desenhar uma linha de um ponto a outro da tela. O terceiro tipo é mais sofisticado:

```
HPLOT 0,0 TO 279,0 TO 279, 191 TO 0,191 TO 0,0
```

Esta versão do HPLOT só opera na versão firmware do Applesoft. As versões em disco e cassette não permitem esta forma de HPLOT. Aqui se pode definir múltiplas declarações do HPLOT facilmente. Todos os segmentos são desenhados na mesma cor. Isto pode ser útil ao se desenhar formas de vários lados.

Alternativas ao HPLOT

A sub-rotina listada abaixo permite programar gráficos de alta resolução em Integer BASIC, sem necessidade de comutar páginas. Esta rotina pode ser útil, embora seja lenta em vista de todos os cálculos serem feitos em BASIC. Para usá-la passe as coordenadas X e Y do ponto desejado. Certifique-se que PÁGINA contém a página desejada. Fica a seu cargo determinar os modos de alta resolução, gráfico e tela cheia. Isso permite flexibilidade adicional. Se é permitido usar esta rotina *sem* indicar antes o modo de alta resolução, a sub-rotina operará na memória de alta resolução, mas sem modificar a tela. Em seguida, depois do programa modificar a área de memória, ele poderá entrar em modo gráfico de alta resolução com uma série de declarações POKE (como foi descrito antes) e toda a operação de desenho leva a efeito fora de cena aparecerá subitamente. Isso permite ao programa apontar a página 2 enquanto mostra a página 1 e vice-versa.

```
20000 REM ************************************
20001 REM * DESENHO EM INTEGER ALTA RESOLUCAO *
20002 REM * ---------------------------------- *
20003 REM * INDIQUE X=COLUNA, Y=LINHA ,        *
20004 REM * PAGINA=1 OU 2; USA AS VARIAVEIS    *
20005 REM * Y1,X1 E X3                         *
20006 REM ************************************
20007 REM
20010 Y1=PAGINA*8192: REM DET. ENDERECO BASE
20020 Y1=Y1+(Y/64)*40+(Y MOD8)*1024
20030 Y1=Y1+(Y MOD 64/8)*128+X/7
20040 X1=PEEK (Y1): REM LE NO BYTE ALTA RESOL.
20050 X3=2^ (X MOD 7)
20060 REM 'OR' DO BIT EM X1 COM
20070 REM O VALOR DO BIT EM X3
20080 IF X1 MOD (X3 ^ 2)< X3 THEN X1=X1+X3
20090 POKE Y1,X3
20100 X1 = PEEK (Y1)
21010 X3 = 2 ^ (X MOD 7)
21020 IF X1 MOD (X3*2)<X3 THEN X1=X1+X3
21025 X3 = 2 ^ (X MOD 7)
21026 GOTO 21100
21030 POKE Y1,X3
21040 RETURN
21100 POKE Y1,X3
21130 RETURN
25000 GOSUB 19000
```

Um Exemplo de Alta Resolução em Integer BASIC

O programa seguinte usa duas das sub-rotinas recentemente introduzidas para desenhar pontos em telas de alta resolução. Controladores de jogos determinam a coordenada a desenhar. Rodando os controladores, obtêm-se linhas na tela.

```
 10 REM ESTE PROGRAMA USA ROTINAS
 20 REM ESPECIAIS PARA LIMPAR E DESENHAR
 30 REM EM GRAFICOS DE ALTA RESOLUCAO
 40 REM USE CTRL-C PARA FINALIZAR
 89 REM GERA MODO GRAFICO
 90 POKE -16304,0
 99 REM GERA GRAFICO DE ALTA RESOLUCAO
100 POKE -16297,0
109 REM DETERM.GRAFICO EM TODA A TELA
110 POKE -16302,0
200 PAGINA=2
204 REM LIMPA MEMORIA DE ALTA RESOLUCAO
205 GOSUB 19000
209 PEGA COORDENADAS DOS PONTOS
210 X= PDL (1)
220 Y= PDL (0)
229 REM DESENHA PONTO DE ALTA RESOLUCAO
230 GOSUB 20010
239 REM PEGA MAIS COORDENADAS
240 GOTO 210
250 END
```

Para uma variação interessante deste programa, tente usar a sub-rotina que preenche a tela de uma cor ao invés de limpar a tela.

Tente melhorar o programa verificando os botões do controlador de jogo com declarações POKE, e limpando a tela a cada vez que o botão é acionado.

USANDO FORMAS EM ALTA RESOLUÇÃO

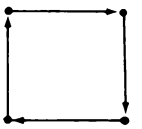
Juntamente com desenhos e traços, o Apple II permite definir desenhos e manipular formas de duas dimensões em modo gráfico de alta resolução. Esta seção ensina como criar, projetar e usar uma forma sob o Applesoft. Tanto quanto possível, esta seção apenas inicia a exploração das possibilidades de criação que são abertas.

Até agora vimos como gerar um programa que desenha gráficos em alta resolução que faça figuras geométricas. Quem operou com esses programas provavelmente encontrou alguma dificuldade em manipular as figuras na tela. Por exemplo, pode-se desejar rodar uma figura sobre um eixo, ou fazê-la aparecer maior ou menor na tela. As formas em alta resolução têm essas possibilidades.

DEFININDO FORMAS

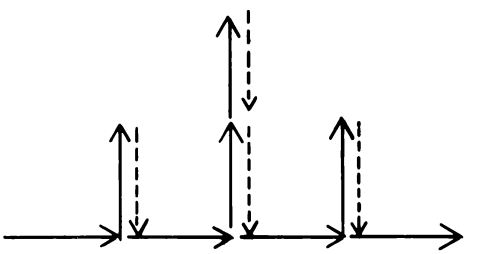
As formas em alta resolução requerem planejamento. Em essência, faz-se mais que dizer ao computador para desenhar uma linha do ponto A ao ponto B. Usando formas no Apple II, descreve-se

inteiramente a figura antes de orientar o computador para desenhá-la. Definem-se formas em alta resolução numa *tabela de formas*, chamada assim por conter as características codificadas da figura a ser desenhada. O primeiro passo para definir uma forma de alta resolução é desenhar a própria forma num papel. Pegue o exemplo de desenhar um quadrado, que consiste de quatro linhas de igual tamanho, cada uma fazendo um ângulo reto com a outra previamente desenhada.



A tabela de forma contém instruções codificadas para desenhar uma figura; essas instruções são chamadas *vetores de desenho.* Cada vetor descreve movimento de subida, descida, direita, esquerda, e ainda se o desenho vai para a tela ou não. Pode-se interpretar cada lado do quadrado ilustrado acima como uma direção a desenhar: um para cima, um para a direita, um para baixo e um para a esquerda. Essa é a forma pela qual as rotinas de manipulação de formas do Applesoft vê as figuras.

Figuras são mais difíceis de se desenhar, se contêm linhas diagonais ou curvas. Um triângulo, apesar de ter um lado a menos que o quadrado, envolve muito mais trabalho porque contém no mínimo uma linha diagonal. As linhas quebradas na ilustração abaixo indicam movimento sem desenho (vetores fantasmas):



Desde que se pode apenas definir uma forma com vetores que movem para cima, baixo e lados, algumas formas, como os círculos, podem não ser nem aproximados. Em tais casos pode ser mais fácil desenhar formas complicadas com o HPLOT ao invés de usar tabela de formas.

MONTANDO A TABELA DE FORMAS

A figura desenhada em papel deve ser convertida em vetores de desenho. Esta seção mostra como fazer a conversão. A seção seguinte mostra um programa em Applesoft que faz a conversão. Assim pode-se pular para a seção seguinte se não houver interesse em saber como as tabelas de formas funcionam internamente.

Os códigos dos vetores têm valor entre 0 e 7; cada bit numa definição de forma (parte da tabela de forma) pode ter até três vetores. A Tabela 6.3 mostra os códigos de vetores possíveis.

TABELA 6.3. Vetores de Desenho e seus Códigos Binários.

Símbolo	Ação	Código Binário	Código Decimal
↑	Move para cima sem desenhar	000	0
→	Move para direita sem desenhar	001	1
↓	Move para baixo sem desenhar	010	2
←	Move para esquerda sem desenhar	011	3
↑	Move para cima desenhando	100	4
→	Move para direita desenhando	101	5
↓	Move para baixo desenhando	110	6
←	Move para esquerda desenhando	111	7

Uma vez que a forma foi reduzida a um conjunto de vetores, eles podem ser colocados na memória, onde certos comandos em Applesoft os decodificam e desenham a forma.

Pegue um ponto inicial da forma. Faça uma lista de vetores de desenho necessários para construí-la, usando flechas (↑, ←, ↓, →). Liste os vetores na ordem em que aparecem na forma (em direção aos ponteiros do relógio ou ao contrário, tanto faz). Marque qualquer vetor que deva ser gerado mas não desenhado (vetores fantasmas). Iniciando no canto inferior esquerdo, nosso quadrado corresponde a estes vetores:

Direção	Desenho
↑	Sim
→	Sim
↓	Sim
←	Sim

Agora escreva o código binário na frente de cada vetor (use a Tabela 6.3 para converter). Obtém-se o seguinte:

Vetor	Código
↑	100
→	101
↓	110
←	111

Como mostrado na Tabela 6.4, cada byte da tabela de formas contém três partes, cada uma das quais pode conter um vetor de desenho. Veja que as partes 1 e 2 contêm três bits cada e a parte 3 contém apenas 2 bits.

TABELA 6.4. Byte da Tabela de Formas.

Bit	Parte 3		Parte 2			Parte 1		
	7	6	5	4	3	2	1	0
M = bit de movimento P = desenha ou não	M	M	P	M	M	P	M	M

Olhando a Tabela 6.3, pode-se ver que alguns dos códigos usados para desenhar são números de 3 bits. Isso é correto para as partes 1 e 2 de cada byte de definição de forma, cada uma das quais tem três bits. Na parte 3, por ter apenas 2 bits, ela só pode ter certos vetores de desenho. Os vetores permitidos na parte 3 são direita, esquerda, cima e baixo sem desenho. Não são reconhecidos outros vetores na parte 3.

Na maioria das vezes a parte 3 não é usada. Isso não significa que se possa esquecer de usá-la para compor o todo, mas na maioria das vezes pode-se fazê-lo. Se a parte 3 da definição de forma é deixada em zero, o Applesoft a ignora, indo para o byte seguinte e interpretando-o para compor o desenho.

O vetor de desenho como zero pode significar duas coisas. Na parte 3 de cada definição de forma, um vetor de desenho a zero sempre significa "sem movimento e sem desenho". Entretanto, na Tabela 6.3, um vetor zero significa "mover para cima sem desenhar". Essa ambigüidade pode causar problemas nas partes 1 e 2 em cada byte de definição de forma, porque sob certas circunstâncias o Applesoft ignora vetores de desenho em zero, e em outras ele executa movimento para cima sem desenhar. A regra aqui é nunca terminar um byte de definição de forma com um vetor de desenho em zero, se for desejado que esse vetor represente "movimento para cima sem desenhar". As rotinas de manipulação de formas do Applesoft assumem que se a porção mais significativa (parte 3) ou as duas mais significativas (2 e 3) são zero, não é executada nenhuma ação de desenho.

Se todas as três partes do byte de definição de forma estiverem em zero, o Applesoft interpreta como sinal de "fim da definição de forma". De fato, deve-se terminar cada definição de forma com um byte de terminação, igual a zero. De outra forma o Applesoft continua após o fim da definição da forma original, até encontrar um byte em 0.

Pode-se usar o vetor de "movimento para cima sem desenho" pois um vetor de desenho vem depois dele no mesmo byte. Por exemplo, a parte 2 pode ser posta em 0 (que significa "mover para cima sem desenhar) e se a parte 3 é deixada em 01, 10 ou 11 (binário), a parte 2 será interpretada como "mover acima sem desenhar". Se a parte 3 é deixada em 0 e a parte 2 também, não ocorre nenhum movimento e o Applesoft vem na parte 1 do byte seguinte para procurar o vetor de desenho válido seguinte.

Contando com esse conhecimento, pode-se agora montar os vetores de desenho em código binário para cada segmento da forma em grupos de dois ou três. Dessa forma, transformam-se os códigos dos vetores de desenho de três bits para oito bits que podem ser guardados em memória. Os vetores de desenho para o *mapa* do quadrado nas suas definições da forma são mostradas na Tabela 6.5.

Com a forma agora mapeada em bytes em codificação binária, pode-se facilmente converter

TABELA 6.5. Codificando a Tabela de Formas (Square Shape).

	Vetores			Códigos Binários			Códigos Hexadecimais
	Pte. 1	Pte. 2	Pte. 3	Pte. 1	Pte. 2	Pte. 3	
Byte 0	Nada	↑	→	00	100	101	25
Byte 1	Nada	↓	←	00	110	111	37
Byte 2	Nada	Nada	Nada	00	000	000	00
Byte 3	—	—	—	—	—	—	—

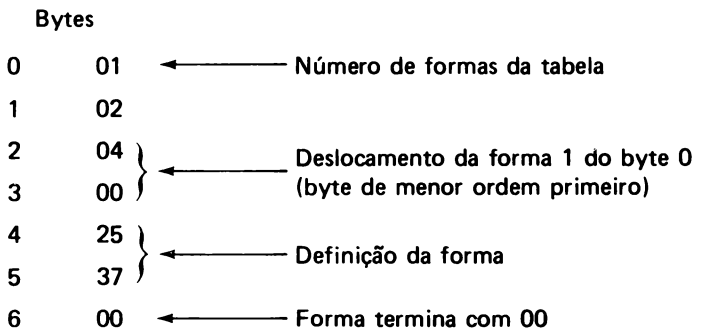
cada byte para notação hexadecimal. O Apêndice J contém uma tabela de conversão de binário para hexadecimal. A Tabela 6.5 mostra a codificação hexadecimal do quadrado.

A definição de forma agora está completa. O próximo passo é criar uma série de apontadores para essa forma (e outras – até 255 formas) que o Applesoft usa como diretório.

Montando o Diretório da Tabela de Formas

O diretório da tabela de formas é uma série de bytes que descrevem quantas formas existem na tabela, e também aponta cada definição de forma na tabela. O primeiro byte do diretório contém o número total de formas da tabela. Este número vai de 0 até 255 ($FF). O segundo byte não é usado e pode ser colocado em zero.

Os bytes restantes no diretório contêm apontadores para cada definição de forma existente na tabela. Cada apontador tem dois bytes de comprimento e contém o deslocamento (distância absoluta em bytes) da forma a partir do *início* do diretório. O byte de menor ordem do apontador precede o de maior ordem. Por exemplo, se o deslocamento em uma forma é 10 bytes, codifica-se em hexadecimal como 0A00. No caso do quadrado, existe apenas uma forma para listar no diretório, de forma que o deslocamento da forma 1 para o diretório é quatro bytes. Assim, 0400 vai para essa parte do diretório.



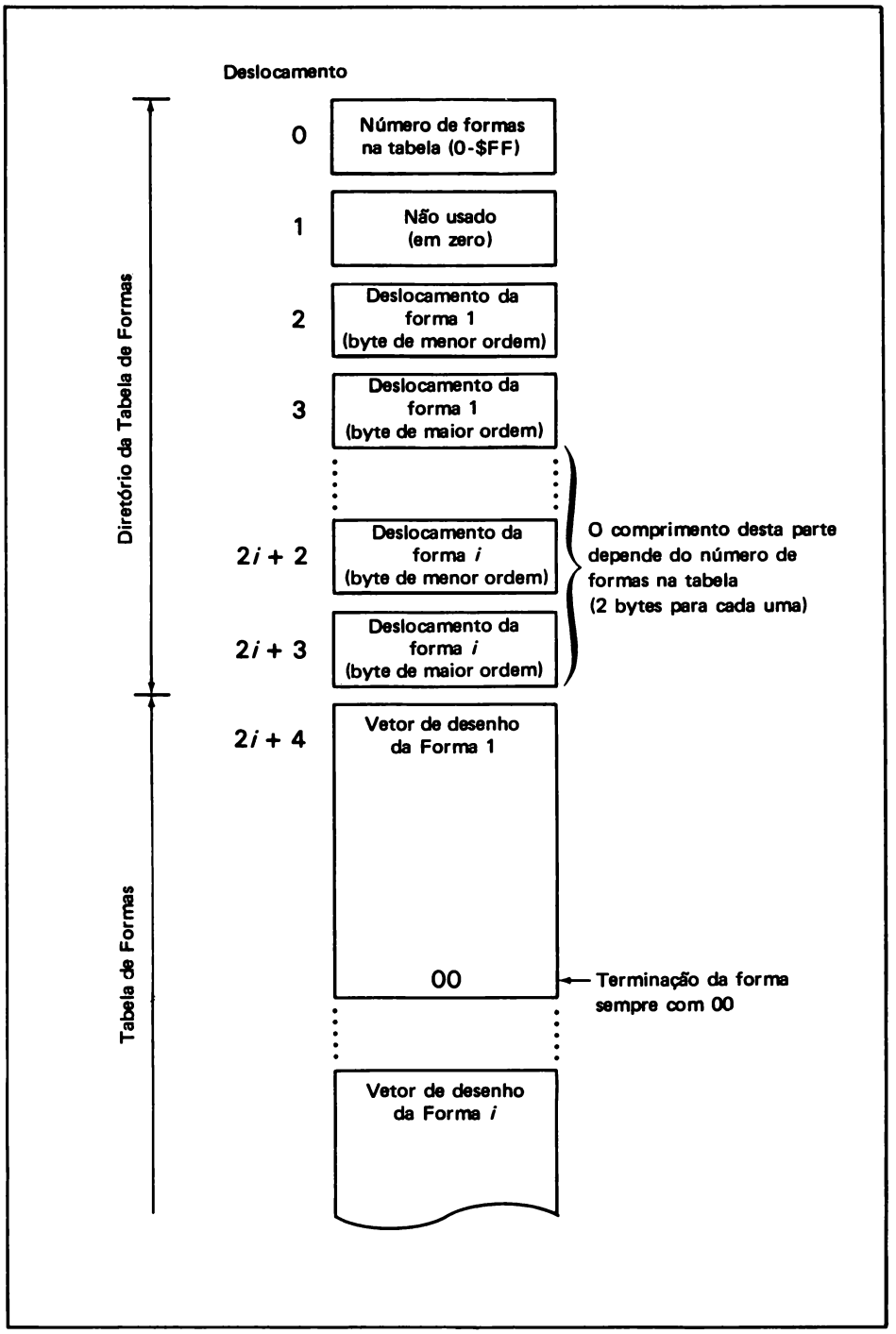


FIGURA 6.3. Organização da Tabela de Formas.

É uma boa prática deixar espaço extra em bytes no fim do diretório da tabela de forma para dar espaço para ponteiros de tabelas futuros. Se não existe espaço no final do diretório para permitir expansão, será necessário reorganizar a tabela de forma inteira quando houver um novo ponteiro de forma para inserir. Mesmo que só se necessite um diretório contendo dez formas, deve-se deixar espaço livre no final do diretório; 20 bytes extra permite mais dez ponteiros de formas para uso posterior. Quando for necessário colocar mais uma forma na tabela, coloque a nova definição após a última feita na tabela, calcule o deslocamento para a nova forma a partir do início do diretório, coloque o novo ponteiro imediatamente após o ponteiro da última forma do diretório, e some 1 ao byte 0 do diretório (que contém o número total de formas na tabela).

A Figura 6.3 ilustra o modo como as tabelas de formas e seus diretórios são organizados na memória.

Montando Vetores por Computador

O programa seguinte, em Applesoft, monta uma definição de forma. Ele pede que se digite cada vetor de desenho e se ele deve ou não ser desenhado. Após digitar o último vetor, digite "F" (de FIM) e acione RETURN. O programa pergunta se algum vetor deve ser alterado. Caso tenha havido algum engano durante a digitação de um vetor, digita-se novamente.

Quando não houver mais correções, digite 0 em resposta a VETOR PARA ALTERAR (0=FIM). Após alguns segundos, os vetores de desenho são apresentados em notação hexadecimal. Aqui está uma listagem do programa e um exemplo de sua atuação:

```
1    REM ********************************
2    REM * PROGRAMA DE CRIACAO DE FORMAS *
3    REM ********************************
10    DIM S1(100),V1(100)
20   I = 0
30    PRINT "CRIACAO DE VETORES DE FORMAS"
40    PRINT
41    REM DIGITACAO DAS ACOES DOS VETORES
50   V = I: GOSUB 270
58    REM DIGITA-SE ATE M$
59    REM SER IGUAL AO VALOR "F"
60    IF M$ <> "F" THEN S1(I) = M: I = I + 1: GOTO 50
70    PRINT
71    REM  PERMITE CORRECOES
80    INPUT "VETOR PARA MODIFICAR (0=FIM):";V
90    IF V > 0 THEN V = V - 1:GOSUB 270:S1(V) = M:GOTO 80
99    REM COLOCA VETORES NO ARRANJO V1()
100    FOR V = 0 TO I
110    IF B = 2 AND S1(V) > 0 AND S1(V) < 4 THEN 140
120    IF B < 2 AND (S1(V) > 0 OR S1(V) > 4) THEN 140
130   B = 0: Q = Q + 1
140   V1(Q) = V1(Q) + S1(V) * (8 ^ B)
```

```
150  B = B + 1
160   IF B > 2 THEN B = 0:Q = Q + 1
170   NEXT V
179   REM MOSTRA VETORES COMO NUMEROS HEXADECIMAIS
180   PRINT "BYTE","VETOR"
190   FOR V = 0 TO Q
200   H% = V1(V) / 16
210   L% = V1(V) - H% * 16
220   IF H% > 10 THEN H% = H% + 7
230   IF L% > 10 THEN L% = L% + 7
240   PRINT V, CHR$ (H% + 176); CHR$ (L% + 176)
250   NEXT V
260   END
269   REM SUBROTINA DE ENTRADA DO VETOR
270   PRINT "VETOR ";V + 1;":";
280   INPUT "MOVIMENTO: C/B/E/D? ";M$
290  M = 0
300   IF M$ = "D" THEN M = 1
310   IF M$ = "B" THEN M = 2
320   IF M$ = "E" THEN M = 3
330   IF M$ = "F" THEN RETURN
340   INPUT "DESENHO (S=SIM,N=NAO)? ";P$
350   IF P$ = "S" THEN M = M + 4: RETURN
360   IF P$ = "N" THEN RETURN
370   GOTO 340
]RUN

CRIACAO DE VETORES DE FORMAS
VETOR 1:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? N
VETOR 2:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? N
VETOR 3:MOVIMENTO: C/B/E/D? E
DESENHO (S=SIM,N=NAO)? S
VETOR 4:MOVIMENTO: C/B/E/D? E
DESENHO (S=SIM,N=NAO)? S
VETOR 5:MOVIMENTO: C/B/E/D? C
DESENHO (S=SIM,N=NAO)? N
VETOR 6:MOVIMENTO: C/B/E/D? C
DESENHO (S=SIM,N=NAO)? S
VETOR 7:MOVIMENTO: C/B/E/D? C
DESENHO (S=SIM,N=NAO)? S
VETOR 8:MOVIMENTO: C/B/E/D? C
DESENHO (S=SIM,N=NAO)? S
VETOR 9:MOVIMENTO: C/B/E/D? D
```

```
DESENHO (S=SIM,N=NAO)? N
VETOR 10:MOVIMENTO: C/B/E/D? D
DESENHO (S=SIM,N=NAO)? S
VETOR 11:MOVIMENTO: C/B/E/D? D
DESENHO (S=SIM,N=NAO)? S
VETOR 12:MOVIMENTO: C/B/E/D? D
DESENHO (S=SIM,N=NAO)? S
VETOR 13:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? N
VETOR 14:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? S
VETOR 15:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? S
VETOR 16:MOVIMENTO: C/B/E/D? B
DESENHO (S=SIM,N=NAO)? S
VETOR 17:MOVIMENTO: C/B/E/D? E
DESENHO (S=SIM,N=NAO)? N
VETOR 18:MOVIMENTO: C/B/E/D? E
DESENHO (S=SIM,N=NAO)? S
VETOR 19:MOVIMENTO: C/B/E/D? F

VETOR PARA MODIFICAR (0=FIM):0
BYTE            VETOR
0               12
1               3F
2               20
3               64
4               2D
5               15
6               36
7               1E
8               07
9               00

]
```

ENTRANDO A TABELA DE FORMAS

Antes de rodar as formas digitadas, deve-se colocá-las na memória do computador. Para isso é necessário determinar em que área de memória a tabela vai ficar. A maneira mais fácil de criar essa área reservada é colocar o HIMEM: em um valor imediatamente abaixo do DOS ou antes da página gráfica de alta resolução usada. Deve-se operar o HIMEM: *antes* de executar qualquer declaração em Applesoft que opere com séries. Em se usando Applesoft com disco, são necessários no mínimo 36K de memória (embora 48K seja melhor) para permitir o interpretador

Applesoft e o DOS. Os endereços 115 e 116 ($73 e $74) contêm o último HIMEM: determinado para o Applesoft, com o byte menor armazenado antes. Para calcular o novo valor do HIMEM: que libere área para a tabela de formas, use a seguinte declaração:

```
PRINT PEEK(116) * 256 + PEEK(115) - X
```

Esta declaração calcula o valor de HIMEM: a ser usado baseado no parâmetro *X*, que é o comprimento da tabela de forma, incluindo diretório. Usando esta declaração, troque *X* pelo comprimento da tabela. Coloque HIMEM: no valor calculado antes de colocar a tabela na memória. Isso a protegerá de ser suja pelo Applesoft.

Como alternativa, pode-se colocar a tabela na memória entre as posições 768 e 975 ($300 e $3CF) inclusive. Cuidado para não haver conflito com qualquer sub-rotina em linguagem de máquina colocada lá anteriormente.

Pode-se usar a declaração POKE para colocar a tabela de forma na memória. Por exemplo, a seguinte série de declarações POKE coloca a tabela de formas do nosso quadrado na memória iniciando na posição 768:

```
]POKE 768,01
]POKE 769,00
]POKE 770,04
]POKE 771,00
]POKE 772,37
]POKE 773,55
]POKE 774,00
```

Pode-se também entrar a tabela através do Monitor. Use a declaração CALL-151 para pular ao Monitor. Entre então o endereço "hexadecimal" da memória onde a tabela iniciará, seguido por dois pontos, digite o primeiro byte do diretório da tabela, um espaço em branco, o byte seguinte do diretório, outro espaço etc. Acione RETURN. Digite então outra vez dois pontos seguidos pelos bytes em hexadecimal da primeira forma da tabela (separando os bytes com espaços), e acione RETURN. Repita este último passo para cada forma do diretório. Pode-se rever o trabalho digitando o endereço inicial da memória, dois pontos, e o endereço final da tabela, acionando RETURN. Para maiores informações de como usar o Monitor, veja o Capítulo 7. A tabela para o quadrado é digitada assim:

```
]CALL  -151
*                                  Entrada do Monitor
*6000:01 00 04 00                  Entrando o diretório da tabela de formas
*:2C 3E 00                         Entrando a forma 1
*6000.6006                         Verificando o novo conteúdo da memória
6000- 01 00 04 00 2C 3E 00
*
```

A tabela de formas agora está na memória. A primeira entrada começa com o endereço inicial da tabela (neste caso $6000). Os dois pontos dizem ao Monitor para colocar uma série de dígitos hexadecimais na memória. Imediatamente após os dois pontos vem o diretório da tabela: 01 (o número de formas da tabela), 00 (o segundo byte não é utilizado), 04 e 00 (o deslocamento

da forma 1 a partir do início do diretório – o byte de menor ordem vem primeiro). A linha seguinte inicia com dois pontos; não é necessário nenhum endereço inicial se ele foi indicado na entrada anterior. O Monitor vai colocar a próxima série de dígitos hexadecimais imediatamente seguinte à primeira série entrada.

A última linha diz ao Monitor para mostrar os endereços de memória de $6000 a $6006. O formato deste comando é: endereço inicial, seguido por dois pontos (que diz ao Monitor para mostrar memória), seguido pelo endereço final. Verifique cuidadosamente o que foi entrado cada vez que usar o Monitor para digitar tabelas de formas.

Guardando a Tabela de Formas em Fita ou Disco

Após se gastar um bom tempo montando tabelas de forma, certamente é uma boa idéia guardar esse trabalho em algum meio magnético ao invés de perdê-lo quando se desligar o Apple II. Pode-se usar cassette ou disco para tal fim; o disco é preferível porque guarda e recupera os dados muito mais rapidamente que a fita.

O Applesoft tem um comando chamado SHLOAD que opera exclusivamente com fitas. Antes de jogar a tabela de formas na fita, deve-se calcular seu comprimento (em bytes, hexadecimal). Continuando com o nosso exemplo do quadrado, conte o número de bytes usados na tabela; o comprimento total é 7 bytes. Usando o Monitor, entre com o comprimento da tabela, iniciando no endereço 0:

```
*00:07 00
```

Como sempre acontece com quantidades de dois bytes entradas pelo Monitor, o byte de menor ordem vem primeiro. Rebobine a fita até o início. Ligue no modo RECORD. O Monitor será usado para gravar o comprimento da tabela e ela mesma. Isso é feito em duas operações de gravação, que podem ser comandadas na mesma linha:

```
*0.1W 6000.6006W
```

O primeiro comando de escrita coloca o comprimento da tabela (colocado nos endereços 0 e 1 da memória) na fita. A segunda operação coloca os sete bytes da tabela do endereço $6000 até $6006, na fita. As duas operações em conjunto demoram cerca de 20 segundos. O falante emite sons por duas vezes durante o processo de gravação. Após o segundo bip, pare o gravador. A tabela então estará gravada na fita para uso posterior. Use o tamanho, endereço inicial e endereço final da sua tabela ao invés destes do exemplo.

Ao usar o disco para guardar tabelas, use o Applesoft com o DOS (a partir do Monitor, com 3DOG). O comando usado é BSAVE. É necessário conhecer o endereço inicial e o comprimento da tabela a ser guardada, no nosso exemplo iniciando em $6000, o comando do DOS.

```
BSAVE QUADRADO, A$6000, L7
```

cria um arquivo em disco chamado QUADRADO com tipo B de binário. A tabela então fica gravada em disco, pronta para ser usada mais tarde. Use em cada caso particular o nome, endereço inicial e comprimento apropriados no lugar destes do exemplo.

Carregando Tabelas de Forma de Fita ou do Disco

Após guardar Tabelas de forma em fita, o Applesoft pode ler de volta para a memória. Primeiro, volte a fita até o início. Depois digite:

```
SHLOAD
```

Acione o botão PLAY do gravador. O alto-falante gera *bips* por duas vezes e então devolve o controle do computador para o teclado. Se existir algum problema na fita, pode aparecer a mensagem de erro ERR na tela. Nesse caso tente ler a fita novamente, ou, se o erro persistir, verifique o ajuste do volume, como foi descrito no Capítulo 2. O SHLOAD coloca automaticamente o HIMEM: no seu valor corrente menos o comprimento da tabela lida em bytes. Confira se o valor HIMEM: foi mesmo acertado.

Se a tabela foi gravada em disquette, acerte o HIMEM: antes de tentar lê-la para a memória. O comando seguinte lê a tabela do quadrado:

```
BLOAD QUADRADO
```

O DOS se lembra do endereço a partir de onde foi gravado ($6000 no nosso exemplo) e colocará a tabela de volta automaticamente. Se for desejado ler para outro endereço, por exemplo $3000, digite:

```
BLOAD QUADRADO, A$3000
```

De novo, use o nome e o endereço de sua tabela no lugar destes, do exemplo.

Após carregar a tabela com o BLOAD, será necessário colocar o endereço nas posições 232 e 233 ($E8 e $E9). O Applesoft usa essas posições para apontar a tabela de formas na memória (o SHLOAD acerta estes endereços automaticamente). Entre no Monitor e coloque o endereço ($6000 no exemplo):

```
E8:00 60
```

Alternativamente, pode-se usar a declaração POKE para colocar o endereço inicial da tabela nas posições 232 e 233. Lembre-se de que com o POKE deve-se colocar endereços em *decimal*.

Agora estamos prontos para usar as tabelas de forma nos programas em Applesoft.

COMANDOS DE DESENHAR AS FORMAS

O Applesoft tem quatro declarações para desenhar, apagar e mudar a orientação das formas:

DRAW, que desenha a forma na tela do Apple II.

XDRAW, que apaga a forma.

ROT, que roda a forma em torno de eixos X e Y.

SCALE, que altera o tamanho da forma desenhada.

Os comandos de manipulação de formas usam a página gráfica de alta resolução (usando HGR e HGR2) e a cor (usando COLOR), correntes.

O Comando SCALE

Como declaração em modo imediato ou programado, pode-se sempre usar o SCALE antes de desenhar uma forma pela primeira vez num programa

```
SCALE=1
```

manda manter a escala de um ponto para cada vetor de desenho. Se SCALE=5, o Apple II desenha 5 pontos para cada vetor de desenho. Pode-se comandar o SCALE até 255 (255 pontos para cada vetor). A máxima escala que se pode determinar é SCALE=0 que coloca 256 pontos para cada vetor de desenho.

O Comando DRAW

O DRAW desenha a forma (numerada de 1 a 255) a partir da tabela de formas, na última cor determinada, na escala e valor de rotação correntes. Esta declaração

```
DRAW 1 AT 140,96
```

coloca a primeira definição de forma iniciando na coluna 141 e linha 97 na página de alta resolução. O desenho começa nas coordenadas dadas nesta declaração. Uma segunda opção da declaração DRAW usa a posição inicial implícita.

```
DRAW 11
```

Esta declaração desenha a décima-primeira forma da tabela no último ponto usado pela declaração DRAW ou HPLOT anterior mais recente. No caso das coordenadas não terem sido referenciadas anteriormente, é adotado o ponto 0,0.

IMPORTANTE: O Applesoft assume que a tabela de forma está colocada corretamente na memória. Antes de executar uma declaração DRAW, certifique-se que a forma esteja realmente lá e os endereços 232 e 233 ($E8 e $E9) apontem o início da tabela. Se for indicado um número de forma maior que o número de formas existentes na tabela, ou se a declaração DRAW usar linhas ou colunas não válidas, o desenho não é executado; em seu lugar a mensagem de erro ?ILLEGAL QUANTITY ERROR é apresentada na tela.

O Comando XDRAW

Esta declaração permite zerar uma forma sem limpar formas gráficas anteriores. Aqui está um exemplo:

```
XDRAW 8 AT 90,96
```

Esta declaração é sintaticamente idêntica ao DRAW; as coordenadas podem ser explícitas como mostrado acima ou implícitas como mostrado na última declaração DRAW exemplificada. O XDRAW verifica a cor da coordenada desenhada e desenha a forma numa cor "complementar" àquela encontrada. No exemplo acima, o XDRAW ocorre na linha 91 e coluna 97 na tela. A Tabela 6.6 lista os complementos das cores de alta resolução.

Se as coordenadas, rotação e escala forem as mesmas da forma presente na tela, XDRAW limpa a forma, deixando todo o resto do gráfico intacto.

TABELA 6.6. Cores de Desenhos.

Se a Cor é	A Cor do XDRAW será
Preto	Branco
Branco	Preto
Violeta	Verde
Laranja	Azul
Verde	Violeta
Azul	Laranja

O Comando ROT

O comando ROT roda a forma em relação ao centro da tela (nos eixos X e Y). A declaração

```
ROT=16
```

coloca o ângulo de rotação da forma em 90 graus no sentido horário. O valor da ROT varia de 0 a 255, apesar de haver apenas 64 possibilidades de rotação, de 0 a 63. A Figura 6.4 mostra as mudanças nas orientações dos eixos a partir dos valores de ROT.

Quando o valor do SCALE é colocado em 1, o ROT apenas roda a forma em incrementos de 90 graus, o que significa que apenas quatro rotações válidas podem ser obtidas: 0=0 grau, 16=90 graus, 32=180 graus e 48=270 graus. O Applesoft arredonda o valor de rotação dado ao próximo

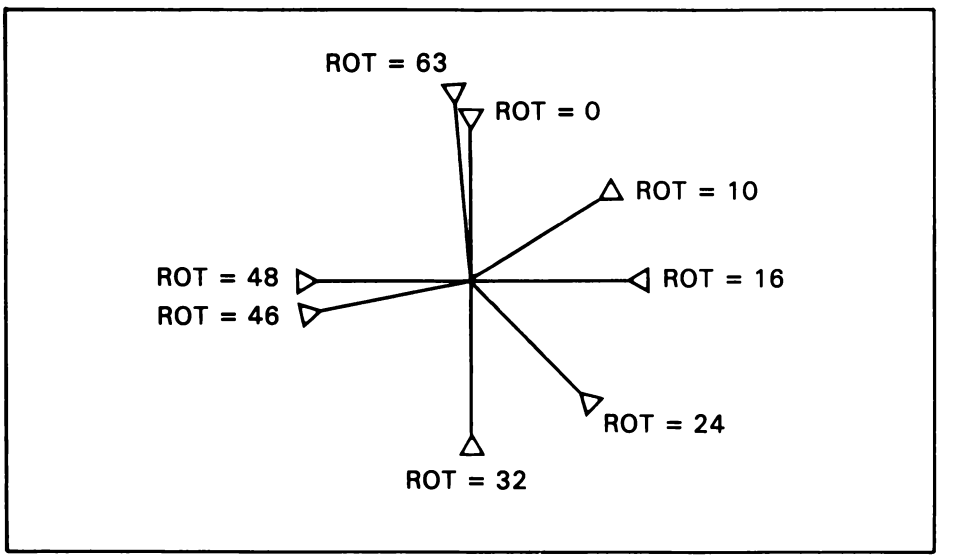


FIGURA 6.4. Rotação da Forma.

menor valor de incremento do ROT. Todas as 64 possibilidades de rotação ficam disponíveis ao se usar SCALE=5 ou maior.

Usando Formas em um Programa

O programa listado abaixo serve como um bom exemplo de aplicação de formas e como elas se aplicam a programas:

```
1  LOMEM: 24600
20 HGR2
50 REM INDICA ENDERECO DA FORMA
60 POKE 232,0
70 POKE 233,96
75 REM USA TODAS AS CORES ALTERNADAS
76 FOR I = 1 TO 7
79 REM INCREMENTA O FATOR DE ROTACAO
80 FOR H = 1 TO 80
82 HCOLOR = H
90 ROT = I
92 IF I<50 THEN SCALE=I
105 DRAW 1 AT 140,96
106 ROT = I + 32
110 DRAW 1 AT 140,96
130 NEXT I
140 NEXT H
150 GOTO 76
```

Usar formas gráficas de alta resolução é especialmente interessante em jogos. Pode-se montar uma biblioteca de formas sem ter que documentá-las como se faria usando declarações HPLOT. Elas diminuem em muito o consumo de memória. São necessários poucos cálculos para manipular o tamanho e a orientação dos objetos em alta resolução criados, e os cálculos são simples quando comparados com a série de declarações HPLOT necessária para desenhar uma forma. Se existir ainda alguma dificuldade sobre as formas de alta resolução, tente montá-las por conta própria. Isso é um pouco complicado devido à quantidade de passos necessários para fazer aparecer uma única forma gráfica na tela. Usando o programa anterior que codifica as formas automaticamente, o trabalho fica facilitado.

O SOM DO APPLE II

Após a última seção vista neste capítulo, pode ser um alívio ou um aborrecimento o estudo do sistema de alto-falante embutido no Apple II. Alívio porque a operação do alto-falante é realmente bem simples. Aborrecimento porque não se pode controlar esta parte com declarações em BASIC. Deve-se definir todos os sons que o Apple II emite em todos os detalhes. Essencialmente,

tudo o que se pode fazer na programação do alto-falante é fazê-lo emitir um único *click*. Para emitir sons diversos com o alto-falante varia-se a freqüência desses clicks, criando assim sons de diferentes tons. Esta seção mostra como operar o alto-falante, e inclui um programa que cria uma série de sons.

OPERANDO O ALTO-FALANTE

O Apple II usa a posição de memória −16336, também conhecida como 49200 e $C030, como uma chave liga-desliga, a exemplo das chaves gráficas vistas anteriormente neste capítulo. A qualquer momento em que se acessar esta posição, é emitido um click do falante do Apple II. Em BASIC, pode-se operar o falante usando a declaração

```
A=PEEK(49200)
```

ou

```
A=PEEK(-16336)
```

Usar um programa BASIC para comandar o falante (mostrado no exemplo acima) é simples, porém o falante só gera freqüências baixas. Isso porque o BASIC é relativamente lento. Em Integer BASIC, a maior freqüência possível é cerca de 256 Hz (ciclos por segundo), e em Applesoft, a maior freqüência é 72 Hz. A única forma de gerar sons de freqüência mais alta é usar uma sub-rotina em linguagem de máquina para operar o falante.

```
 9 REM DAR UM "CLICK" NO ALTO'FALANTE
10 A = PEEK (-16336)
20 GOTO 10
```

A Sub-rotina em Linguagem de Máquina para Som

Tanto o Applesoft quanto o Integer BASIC tem uma área de memória entre 768 e 975 ($300 e $3CF) para sub-rotinas em linguagem de máquina e tabelas de formas sem necessidade de alterar as atribuições de LOMEM: (o DOS usa esta área, eliminando todo o conteúdo anterior, quando carregado a partir do DOS master disquette em versões anteriores à 3.3).

Mesmo se não souber linguagem assembler e de máquina, incorpore a sub-rotina seguinte em programas escritos em Integer BASIC ou Applesoft, digitando uma série de números hexadecimais. Para colocar a sub-rotina na memória, entre no Monitor com o comando CALL-151 ou acionando RESET (*Importante:* com o DOS, certifique-se de tê-lo carregado antes de entrar no Monitor). Então, digite a sub-rotina em linguagem de máquina, como mostrado abaixo:

```
)CALL -151
*F666G

!302:LDY 301

0302-     AC 01 03        LDY    $0301
! LDX 301
```

O computador responde com as linhas escuras

```
0305-     AE 01 03        LDX    $0301
! LDA #4

0308-     A9 04           LDA    #$04
! JSR FCA8

030A-     20 A8 FC        JSR    $FCA8
! LDA C030

030D-     AD 30 C0        LDA    $C030
! INX

0310-     E8              INX
! BNE 310

0311-     D0 FD           BNE    $0310
! DEY

0313-     88              DEY
! BNE 305

0314-     D0 EF           BNE    $0305
! DEC 300

0316-     CE 00 03        DEC    $0300
! BNE 302

0319-     D0 E7           BNE    $0302
! RTS

031B-     60              RTS
```

Esta sub-rotina usa endereços $300 e $301 para guardar dados; os endereços $302 até $31B contêm a rotina. Uma vez digitada a sub-rotina, verifique se está correta, assim:

```
!$FF69G

*302L

0302-   AC 01 03    LDY   $0301
0305-   AE 01 03    LDX   $0301
0308-   A9 04       LDA   #$04
030A-   20 A8 FC    JSR   $FCA8
030D-   AD 30 C0    LDA   $C030
0310-   E8          INX
0311-   D0 FD       BNE   $0310
```

```
0313-   88          DEY
0314-   D0 EF       BNE   $0305
0316-   CE 00 03    DEC   $0300
0319-   D0 E7       BNE   $0302
031B-   60          RTS
031C-   20 20 70    JSR   $7020
031F-   08          PHP
0320-   18          CLC
0321-   D8          CLD
0322-   88          DEY
0323-   08          PHP
0324-   A0 A0       LDY   #$A0
0326-   10 38       BPL   $0360
* ← CTRL-B no BASIC aqui
```

Tendo o DOS no Apple II, pode-se guardar esta sub-rotina em disco com o comando BSAVE:

```
BSAVE SOM,A$302,L26
```

Isso cria um arquivo binário chamado SOM, que pode ser lido mais tarde. Tendo cassette, grave a rotina com o seguinte comando no Monitor:

```
*302.318W
```

A sub-rotina é gravada na fita para uso futuro.

A Interface do BASIC

Precisaremos uma rotina em BASIC que fale com a sub-rotina em linguagem de máquina. Digite a seguinte listagem em Applesoft ou Integer BASIC:

```
3200 REM OPERACAO DE ALTO-FALANTE
3210 POKE 768,D
3220 POKE 769,F
3230 CALL 770
3240 RETURN
```

Com esta rotina em operação conjunta com a rotina em linguagem de máquina, um programa BASIC pode gerar sons através de duas variáveis: F (para freqüências) com variação entre 1 e 255, com 255 sendo o maior tom, e D (para duração) também entre 1 e 255, com 255 sendo o maior intervalo de duração possível. Como exemplo, digite o programa BASIC seguinte após a sub-rotina anterior:

```
10 FOR I = 1 TO 254
20 F = 1
30 D = I
40 GOSUB 3200
50 NEXT I
60 END
```

Ao rodar o programa, ouça a duração de cada nota. Nos finais de escala em baixa e alta freqüência, os sons são menores que os obtidos em freqüências médias. Este problema é inevitável porque a sub-rotina em linguagem de máquina usa instruções ao invés de tempo para trocar a freqüência de cada nota. Desde que não existe relógio de tempo real no Apple II, este método é necessário. Gerando durações maiores para notas muito altas ou baixas pode-se obter a equalização.

Um Programa de Sons mais Elaborado

O programa listado abaixo usa as rotinas de geração de som mostradas antes para criar uma série de sons que podem ser ouvidos, alterados e finalmente impressos para uso em outros programas em BASIC. Ao rodar o programa, aparece a mensagem (D)IGITA, (E)SCUTA, (I)MPRIME? A primeira ação é para digitar tons; digite E e acione RETURN.

Agora a mensagem TON 0: FREQÜÊNCIA, DURAÇÃO? aparece. Entre dois números separados por vírgula. O primeiro é a freqüência e o segundo é a duração. Ambos os números devem estar entre 1 e 255. Ao acionar RETURN depois da digitação da freqüência e duração do tom, o falante do Apple II emite o tom. Este processo se repete para uma série de tons (até 100). Após o último tom entrado, digite 0 para tom e 0 para duração para finalizar.

Depois de digitar todos os tons, a escolha QUE NOTA DEVE ALTERAR? aparece. Se a intenção é trocar algum dos tons entrados, coloque o seu número; caso contrário digite 0.

A mensagem (D)IGITA, (E)SCUTA, (I)MPRIME? aparece de novo ao final das alterações; agora digite E e acione RETURN. O Apple II repete toda a série de tons digitada. Ao final da última nota, a mensagem aparece de novo. Coloque I para mostrar ou imprimir as freqüências e durações atribuídas a cada nota.

O programa listado abaixo pode facilmente ser modificado para permitir guardar os valores dos tons em cassette ou disco. Outros programas podem reler os tons gravados e, usando sub-rotinas de geração de som, tocar músicas e simplesmente emitir sons aleatórios.

```
10 REM PROGRAMA DE GERACAO DE SONS
19 REM A RAIZ GUARDA OS TONS
20 DIM A(100,2)
29 REM LIMPA A TELA
30 CALL -936
40 INPUT "(D)IGITA, (E)SCUTA, (I)MPRIME?";A$
50 IF A$ = "E" THEN 1000
60 IF A$ = "I" THEN 1200
80 IF A$ < > "D" THEN 30
81 REM ENTRANDO CADA TOM
90 PRINT
100 I = 0
105 M = I
110 GOSUB 3000
119 REM FIM DA DIGITACAO?
120 IF F = 0 AND D = 0 THEN I = I - 1: GOTO 200
129 REM NAO, GUARDA NOME
130 A(I,1) = F:A(I,2) = D
```

```
140 I = I + 1
150 GOTO 105
200 REM TROCA QUALQUER ITEM AQUI
205 PRINT "QUE NOTA DEVE ALTERAR? (0--";I;")";
206 INPUT E
208 IF E = 0 THEN 30
210 IF E < 1 OR E > I THEN 205
220 M = E: GOSUB 3000
230 A(E,1) = F:A(E,2) = D: GOTO 205
1000 REM OUVIR NOTAS ENTRADAS
1010 FOR K = 0 TO I
1020 F = A(K,1):D = A(K,2):GOSUB 3200
1030 NEXT K
1040 GOTO 30
1200 REM IMPRIME AS NOTAS
1210 PRINT "NOTA#","FREQ","DURACAO"
1220 FOR K = 1 TO I
1230 PRINT K,A(K,1),A(K,2)
1240 NEXT K
1250 PRINT
1255 PRINT "ACIONE RETURN PARA CONTINUAR":INPUT Z$
1260 GOTO 30
3000 PRINT "TOM ";M;
3010 INPUT " ENTRE FREQUENCIA, DURACAO";F,D
3015 IF F = 0 AND D = 0 THEN RETURN
3020 IF (F < 0 OR F > 255) OR (D < 1 OR D > 255)
 THEN 3010
3030 GOSUB 3200
3040 RETURN
3200 REM CONTROLE DO ALTO FALANTE
3210 POKE 768,D
3220 POKE 769,F
3230 CALL 770
3240 RETURN
```

7

Monitor de Linguagem de Máquina

Existe um programa de controle residente no Apple II em memória ROM (*ready-only memory*) chamado Monitor. Este capítulo descreve suas características e usos, mostrando como operá-lo em conjunto com os seus programas dedicados em BASIC. O Monitor é escrito em linguagem de máquina; serve como elo entre o BASIC (e as outras linguagens suportadas pelo Apple II) e as várias funções de baixo nível executadas pela máquina, como imprimir um caractere, desenhar uma linha e assim por diante.

Pode-se também usar o Monitor via comandos do teclado. Um dos usos é fazer a tabela de formas (descrita no Capítulo 6), para examinar memória e isolar problemas de hardware, ou programar em assembler. A maior parte das vezes não é necessário empregá-lo, porém ele tem algumas funções que podem ser úteis.

Após descrever o Monitor e suas facilidades, este capítulo mostra como usar o Mini-Assembler. Não nos propomos aqui a ensinar a programar em linguagem assembler; o Apêndice K de cada livro ensina assembler. Será mostrado como integrar seu programa em assembler com um programa BASIC, e como usar o Mini-Assembler para escrever, testar e depurar.

ACESSANDO O MONITOR

Existem duas versões do Monitor: a versão padrão e a versão Autostart. Se seu Apple II tem um Monitor padrão, simplesmente ligando a máquina, o controle estará no Monitor. Nesta situação, haverá a tela cheia de caracteres aleatórios, com um asterisco (*) no canto direito em baixo da tela e o cursor piscando à direita do asterisco. Este asterisco é o de entrada no Monitor.

Se o Apple II tem o Monitor Autostart (tanto como facilidade adicional ou como equipamento padrão no Apple II Plus), será necessário usar ou Integer BASIC ou Applesoft para acessar

o monitor. Quando aparecer o caractere de entrada do BASIC (ou > para o Integer BASIC ou] para o Applesoft), digite o seguinte comando:

```
CALL -151
```

Esta declaração é realmente uma chamada de sub-rotina para o endereço FF69, mas o BASIC não reconhece números hexadecimais, de forma que se deve usar o decimal equivalente ao FF69 neste comando. Uma vez digitada esta declaração, aparece o asterisco, seguido do cursor. Neste ponto já se está no Monitor.

DEIXANDO O MONITOR

Existem poucas formas de se deixar o Monitor e voltar ao BASIC; elas dependem fortemente do que se deseja fazer após deixá-lo. Para preservar as declarações de programa e as variáveis usadas no programa BASIC, deixe o Monitor acionando CTRL-C, e depois RETURN. Após fazer isso, volta-se ao BASIC. Neste ponto se pode imprimir o valor das variáveis e listar o programa (se algum deles estiver na memória nesse momento).

Para exemplificar, vamos supor que se colocou um valor numa posição de memória via declaração POKE e se deseja verificar o POKE. Claramente, se poderia usar PEEK para esse propósito, porém o Monitor permite acesso mais fácil à memória do Apple II do que PEEK e POKE. O exemplo em Integer BASIC dado abaixo mostra como usar o CTRL-C preservando o programa BASIC e as variáveis ao deixar o Monitor:

```
>10 A=123
>19 REM MOVE CURSOR PARA 13A. COLUNA
>20 POKE 36,12
>30 PRINT A
>40 END
>RUN
              123
>CALL -151
* ← Acione CTRL-C e depois RETURN


>PRINT A
123
>LIST
    10 A=123
    19 REM MOVE CURSOR PARA 13A.COLUNA
    20 POKE 36,12
    30 PRINT A
    40 END
>
```

Querendo deixar o Monitor e limpar o programa BASIC corrente e as variáveis guardadas na memória, acione CTRL-B e depois RETURN. Assim volta-se ao BASIC, porém qualquer declaração

de programa ou variável presente na memória serão zerados. Tente o exemplo acima, substituindo CTRL-C por CTRL-B. Na volta ao BASIC verifique que não haja mais variáveis nem programa algum.

O CTRL-C opera corretamente com o Integer BASIC e o Applesoft em *firmware*. Não funciona com Applesoft em disco ou cassette.

Com o Applesoft em disco ou cassette deve-se usar o CTRL-B para deixar o Monitor. Neste caso haverá o retorno para o Integer BASIC e as variáveis e programas em Applesoft serão perdidos. Existe uma forma de voltar ao Applesoft com programas e variáveis intactos.

Para voltar ao Applesoft em disco do Monitor, digite este comando:

```
*3D0G
```

Para voltar ao Applesoft em cassette do Monitor, digite este comando:

```
*0G
```

3D0G e 0G são dois comandos do Monitor (semelhantes ao GOTO do BASIC). Mais tarde serão vistos outros detalhes.

IMPORTANTE: Use o comando 0G *apenas* no Applesoft em cassette.

FUNÇÕES DO MONITOR

O Monitor executa um número limitado de tarefas, mas cada uma delas é relativamente poderosa, considerando o tamanho reduzido do programa. Pode-se examinar posições de memória ou os registradores internos do microprocessador, listar conteúdos de uma área de memória na tela, ou jogar para outro dispositivo de saída, como a impressora, e ainda trocar o conteúdo da memória ou dos registradores. Outras funções incluem mover blocos de dados de um endereço para outro e comparar dois blocos de posições de memória. Ainda existem algumas outras funções que aparecerão neste capítulo.

EXAMINANDO A MEMÓRIA

Existem três métodos no Monitor usados para análise do conteúdo da memória: modos de *endereçamento simples, palavra* ou *bloco*. Um endereço simples referencia uma posição de memória de um byte. Uma palavra é um segmento de memória de oito bytes, começando com um endereço divisível por oito. Um bloco é a forma mais conveniente de ver uma área de memória, iniciando com um endereço e terminando com outro mais alto.

Examinando Endereços Simples

Ao se ver o caractere do Monitor e desejando ver uma única posição de memória, digite o endereço hexadecimal dessa posição seguido pela tecla RETURN, como no exemplo abaixo:

```
*FF69
```

O Monitor responde com a apresentação do conteúdo daquela posição:

```
*FF69- A9
```

Quando se digita um endereço, o Monitor o retém para uso posterior, como um ponteiro. Assim, ao se digitar FF69 (ou outro endereço hexadecimal qualquer), o Monitor se lembrará daquela posição, usando-a como referência para exames posteriores de memória, até que ele seja mudado. Para trocar o ponteiro usado pelo Monitor, simplesmente digite um novo endereço. Por exemplo, digite:

```
*300F
```

e o Monitor põe esse novo endereço no ponteiro além de mostrar o conteúdo do endereço 300F.

Examinando Palavras de Memória

Suponhamos que foi examinado o endereço simples FF69 do exemplo acima, e agora se deseja ver os conteúdos dos próximos endereços superiores da memória. O Monitor vai continuar examinando memória se for digitado RETURN, como mostrado abaixo:

```
*FF69

FF69- A9

* ← Acione RETURN

 AA 85 33 20 67 FD

*

FF70- 20 C7 FF 20 A7 FF 94 34

* ← Acione RETURN novamente
```

A primeira vez que foi acionada a tecla RETURN no exemplo, foram mostrados seis bytes na tela. São eles os conteúdos das posições de FF6A até FF6F. A segunda vez que se acionou RETURN toda a palavra de oito bytes foi mostrada, com o endereço de memória inicial da frente (FF70) para lembrar qual foi a posição inicial. Toda a palavra inicia numa posição de memória divisível por oito, daí por que o endereço inicial não apareceu na primeira vez em que RETURN foi acionada. O primeiro RETURN simplesmente terminou o bloco iniciado anteriormente no exame de uma única posição.

Examinando Blocos de Memória

Pode-se examinar uma área maior de memória (normalmente mais de oito bytes) em modo blocado. Digite o endereço inicial em hexadecimal, seguido por um ponto, e finalmente o endereço final

também em hexadecimal. Por exemplo:

```
*F800,F83F
```

O Monitor responde com o conteúdo das posições de memória do bloco:

```
F800- 3A 21 4E 6F DA 76 98 12
F808- 21 00 98 FE D2 7F 44 98
F810- 33 5F DD 21 76 83 6F 1A
F818- A2 09 55 F0 09 3E 42 12
F820- C8 20 31 4F 4D 3C 59 2A
F828- 48 00 20 F8 68 C5 2D 27
F830- F5 67 1A AC 43 22 CB 6F
F838- 84 2D A0 3B 1E 3E 55 00
*
```

O endereço inicial deve ser menor ou igual ao endereço final para que se veja mais de um endereço. Se o endereço final for menor que o inicial, aparecerá apenas o conteúdo do endereço inicial.

Pode-se especificar uma área de endereços que exceda o tamanho da tela do Apple II. Neste caso os dados vão saindo tela acima para dar lugar aos outros que vão aparecendo. Não se pode cancelar este comando sem acionar RESET. Se o Apple II contiver o Monitor Autostart, pode-se parar temporariamente o processo acionando CTRL-S. Isso faz com que pare a saída pela tela, mas não cancela a saída para outro dispositivo qualquer, como a impressora, dando a chance de ler a tela. Acione a barra de espaço para continuar o processo.

O método blocado de examinar memória é conhecido como *dump*. Quando ele termina, o ponteiro que mantém o próximo endereço a ser examinado é atualizado de forma a apontar o próximo byte depois do endereço final examinado do bloco.

Uma forma mais compacta deste comando usa o ponteiro como endereço inicial do bloco. Digita-se o ponto seguido pelo endereço final do bloco. Por exemplo, se foi visto o endereço F800 até F83F, como no exemplo acima, pode-se continuar examinando um bloco iniciando em F840 e finalizando em F880 digitando-se:

```
*.F880
```

que resulta na seguinte saída:

```
F840- 20 28 F8 88 10 F6 60 48
F848- 3E 09 23 FE 54 1A DC 6F
F850- 2E 4F DD 3B 68 11 08 4A
F858- 08 00 12 34 4D 00 30 2A
F860- 6E 08 10 3B CA 1A 20 30
F868- 3A EE CF 08 00 34 87 00
F870- 54 AD 28 0C B3 1E F8 4F
F878- 1A 00 32 4D EC FA DE 01
F880- 0F
*
```

EXAMINANDO OS REGISTRADORES DO MICROPROCESSADOR

Às vezes é necessário examinar o conteúdo dos registradores internos do microprocessador. Isso é feito digitando-se CTRL-E, seguido pela tecla RETURN. O resultado se parece com:

```
A=CD  X=B1  Y=C3  P=B5  S=F0
```

Os valores mostrados são aqueles guardados no Acumulador (A), Registrador de Índice X (X), Registrador de Índice Y (Y), Registrador de Status do Processador (P) e Apontador de Pilha (S). Os valores à direita de cada sinal de igual são os últimos valores dos registradores. Entretanto, eles não são afetados pelo funcionamento do Monitor. Em outras palavras, o conteúdo dos registradores são salvos pelo Monitor e ficam intactos até que se execute um programa em assembler ou BASIC.

ALTERANDO MEMÓRIA

Alterar memória é mais completo que simplesmente examiná-la. Deve-se especificar qual o endereço a ser alterado bem como fornecer o novo dado que irá para o endereço. Pode-se alterar um endereço por vez (um byte por vez), ou pode-se modificar uma série de posições de memória consecutivas, entrando novos dados para várias posições de memória consecutivas num único comando.

Alterando Endereços Simples

O primeiro passo para trocar um endereço simples é indicar o ponteiro de endereço, que é o mesmo usado para examinar memória. Devido aos dois comandos terem o mesmo ponteiro, indica-se o endereço no ponteiro da mesma forma que no exame. Digite em hexadecimal o endereço a ser alterado, e acione RETURN. Por exemplo

```
*1200
```

coloca o ponteiro com 1200 hexadecimal.

O Monitor responde com o conteúdo da posição de memória:

```
1200- 73
```

Veja que esta entrada produziu a mesma resposta que o comando de examinar endereços simples. A resposta mostra que o endereço apontado é 1200, e também mostra o conteúdo anterior deste endereço (neste caso, 73).

O passo seguinte é colocar um novo valor neste endereço. Para trocar, primeiro digite dois pontos (:) seguido por um número hexadecimal de dois dígitos que irá ocupar aquele endereço. Por exemplo:

```
*:5F
```

Os dois pontos (:) indicam um comando de alteração para o Monitor. O 5F indica o novo dado a ser colocado na posição 1200. Pode-se alterar a memória usando uma linha de comando ao

invés de duas, como mostrado abaixo:

```
*1200:5F
```

Esta linha de comando tem o mesmo efeito na troca do endereço 1200 que as duas linhas separadas mostradas acima. O ponteiro de endereço é atualizado para 1200, e o Monitor coloca 5F naquele endereço. O ponteiro de endereço vai para a posição de memória superior seguinte. Se for desejado trocar o endereço 1201 para 7F, por exemplo, digite:

```
*:7F
```

e o Monitor automaticamente atualiza o conteúdo de 1201 para 7F. Aqui também o ponteiro incrementado de 1 e se pode então alterar o endereço 1202 para algum valor qualquer, continuando o processo de alteração de memória sem especificar o endereço.

Alterando mais Memória

O Monitor permite alterar mais de uma posição de memória por vez, colocando os endereços que vão sendo alterados em seqüência na memória (por exemplo, endereços de 1200 até 1207 inclusive). Este comando inicia da mesma forma que o método de alterar posições simples de memória. Primeiro acerta-se o ponteiro, se necessário. A seguir, usa-se dois pontos para indicar ao Monitor que este é um comando de alterar memória. Finalmente, digitam-se os dados que se desejam em cada posição sucessiva, separando cada um deles com um espaço.

Por exemplo, para colocar as quantidades de 00 a 07 nos endereços de 1200 a 1207, digite:

```
*1200:00 01 02 03 04 05 06 07
```

Pode-se na verdade alterar mais de oito endereços. Colocando-se o ponteiro no início deste comando, poder-se-ia alterar até 83 valores em uma única linha. Sem indicar o ponteiro, pode-se colocar até 84 valores. Em ambos os casos, a linha de comando pularia na tela ocupando várias de suas linhas. Isso não é prático porque quando uma linha de comando pula na tela, torna a verificação muito difícil. Além disso, a única forma de corrigir erros é dar *backspaces* até eles, corrigi-los, redigitando o resto da linha. Em caso de se colocar muitos dados numa linha de comando, isso fica mais difícil.

Verificando a Alteração da Memória

É uma boa prática verificar as alterações da memória se o produto final (ou uma tabela gráfica ou uma série de instruções em linguagem de máquina) deve ser preciso. Para isso deve-se usar uma das três formas de examinar o conteúdo da memória vistos antes neste capítulo. Para iniciar a verificação do que foi digitado, deve-se colocar o endereço inicial na primeira posição onde foi feita alteração.

Assumindo que foi colocado 00 até 07 nos endereços de 1200 a 1207, coloque o ponteiro em 1200:

```
*1200
```

O Monitor responde com:

```
*1200- 00
*
```

Pressionando RETURN pode-se ver os sete endereços restantes que foram modificados:

```
* ← Acione RETURN

 01 02 03 04 05 06 07
*
```

Se foram alteradas muitas posições, continue pressionando RETURN até aparecer a última alteração efetuada.

Pode-se também usar o método de examinar bloco para esta área:

```
*1200.1207
```

O Monitor responde com:

```
1200- 00 01 02 03 04 05 06 07

*
```

Corrigindo Erros

No caso de erros na digitação das alterações, pode-se corrigir cada um individualmente sem se redigitar todos os dados. A forma mais simples é anotar o endereço do dado errado e usar o método de alteração individual para corrigi-lo.

Por exemplo, se foi cometido um erro na colocação de 00 até 07 no exemplo anterior, e o erro apareceu em 1204, digite:

```
*1204:04
```

para corrigir o erro neste endereço. O passo seguinte é olhar de volta o endereço 1204 e qualquer outra alteração feita para certificar que agora tudo está correto. Finalmente verifique todas as alterações para se assegurar que tudo agora está correto.

ALTERANDO OS REGISTRADORES DO MICROPROCESSADOR

O processo de se alterar os registradores do microprocessador é um pouco diferente do de se alterar memória, porque os registradores não têm endereço. Para alterar seus conteúdos, primeiro deve-se examiná-los usando o comando CTRL-E. Imediatamente após o comando de examinar o registrador, troca-se seu conteúdo digitando-se dois pontos (o que significa uma operação de alteração no Monitor), seguido por de um a cinco números hexadecimais. Os números são separados por espaços.

O primeiro número hexadecimal será o novo valor do Acumulador, o segundo número será o novo valor do Registrador de Índice X, o terceiro será o novo valor do Registrador de Índice Y, o quarto será o novo valor do Registrador de Status do Processador e o quinto será o novo valor do Ponteiro de Stack. Deve-se colocar valores para todos os registradores até o último da série que se deseja trocar. Pode-se deixar intactos os valores dos registradores além dele.

Como exemplo, suponhamos que se deseja alterar o Registrador de Índice Y deixando os demais intactos. Primeiro, examinemos os registradores com CTRL-E.

```
*  ← Acione CTRL-E e RETURN
 A=CD  X=B1  Y=C3  P=B5  S=F0
*
```

(Note que os conteúdos dos registradores acima são apenas exemplos.)

Para trocar o Registrador de Índice Y para o valor hexadecimal 8A, sem trocar os outros registradores, digite os valores existentes do Acumulador e do Registrador de Índice X, seguidos pelo novo valor do Registrador de Índice Y.

```
* CD B1 8A
```

Para alterar qualquer registrador que não o Acumulador (A), deve-se redigitar o conteúdo de todos os registradores até e inclusive o que vai ser alterado.

Deve-se acionar CTRL-E e RETURN para examinar os registradores, ou o Monitor entende que se deseja alterar endereços de memória. O exame dos registradores diz ao Monitor para sair do ponteiro de endereços para indicar posições de memória e passar para alteração direta de registradores.

Para ilustrar outra alteração de registrador, vamos verificar o ponteiro de Stack (seu conteúdo segue o S quando se examina os registradores) para passar a ter o valor 4B. Primeiro, lembre-se de examinar os registradores:

```
*  ← Acione CTRL-E e RETURN

 A=FF  X=CD  Y=81  P=8A  S=F0

*
```

Agora digite os valores correntes, em ordem, para todos os registradores, e depois o novo valor do último:

```
*:FF CD 81 8A 4B
```

Confira a digitação com outro CTRL-E para examinar os registradores uma vez mais.

GUARDANDO E LENDO MEMÓRIA COM OS PERIFÉRICOS DO APPLE II

O Monitor permite que se use um gravador cassette para guardar o conteúdo de um bloco de memória em fita magnética. Isso é útil quando se criam formas gráficas de alta-resolução (veja o Capítulo 6), ou quando se faz programas em linguagem assembler que se deseja guardar. Com o Sistema Operacional em Disco do Apple II (DOS), pode-se guardar áreas de memória de forma ainda mais fácil e confiável, no disco. Para guardar memória em disco deve-se deixar o Monitor temporariamente, usando o DOS dentro do BASIC para gravar e ler memória.

Guardando a Memória em Fita Magnética

Para gravar em fita, use o comando de gravar fita do Monitor. Deve-se informar a ele os endereços inicial e final da área a ser gravada. O comando inicia com o endereço inicial, seguido por um ponto, o endereço final e finalmente a letra W.

Por exemplo, o comando:

```
*2200.2FFFW
```

diz ao Monitor para gravar conteúdo de memória, iniciando no endereço hexadecimal 2000 e finalizando no endereço 2FFF, no gravador cassette.

O comando de gravação de memória não verifica os dados que manda para a saída de cassette; ela também não testa (nem poderia) a existência de fita dentro do gravador conectado ao Apple II. Deve-se fazer todo o necessário para assegurar que a fita não tenha qualquer falha ou que exista qualquer problema inerente ao uso de fita cassette.

Quando der o comando para gravar em fita, não acione RETURN até colocar o gravador no modo RECORD e que a fita esteja visivelmente em movimento. Se o cassette usado estiver no início da fita, deixe-o correr pelo menor por cinco segundos antes de acionar RETURN, para permitir que passe a parte inicial não magnética pela cabeça.

Ao se acionar RETURN, o computador espera durante dez segundos antes de mandar dados. Isso permite ao gravador cassette zerar qualquer informação prévia (música, voz ou dados) na fita. O computador manda um tom de referência para a saída durante este tempo. O Monitor usa este tom mais tarde como um sinal de sincronismo para o comando de leitura (visto na seção seguinte).

Quando o comando de gravar memória termina, o alto-falante do Apple II dá um bip e o controle volta ao Monitor.

O comando de gravar permite que se guarde em fita desde um byte até 64 kilobytes (65.536 bytes). O Monitor manda dados através da saída para gravador na velocidade de 210 caracteres por segundo aproximadamente (considerando que 16.384 bytes são enviados em 77,5 segundos, após o tom de referência). Depois de transmitir o último byte de dado, o Monitor manda um byte de verificação ao gravador. O comando de leitura usa este byte para verificar a validade dos bytes na leitura.

Leitura de Dados da Fita Magnética

O comando de leitura da fita permite pegar de volta os dados gravados em cassette e jogá-los de volta na memória. Para executar este comando, digite o endereço inicial (a partir de onde os bytes da fita devem ser carregados na memória), seguido imediatamente por um ponto, o endereço final (onde o último byte lido será colocado), e por fim a letra R.

Por exemplo, o comando

```
*2000.20FFR
```

diz para ler dados do cassette para a memória, iniciando no endereço hexadecimal 2000 e terminando em 20FF.

Diferentemente do comando de gravar, este força o Monitor a esperar até que se acione a tecla PLAY do gravador cassette. O computador espera o tom do gravador cassette, e o Monitor bloqueia o computador até a chegada do sinal. Antes de se acionar o PLAY do gravador, posicione a fita onde o tom de referência começa. Pode-se perceber a diferença entre o tom gravado e o lido de forma auditiva. Remova o cabo do fone externo para ouvir a fita usando o alto-falante do gravador. O tom de referência é um zumbido de tom médio e constante. Já os dados se parecem com ruído aleatório ou estático.

Ajuste o volume do gravador antes de dar o comando de leitura. O procedimento para este ajuste é o mostrado no Capítulo 2.

O comando de leitura de memória espera pela leitura de exatamente a mesma quantidade de memória que foi gravada anteriormente. Se foram gravados 1024 bytes numa fita e se deseja ler apenas os primeiros 256 bytes, o Monitor vai transferir os dados mas provavelmente virá também uma mensagem de erro. O mesmo acontece no caso de se ler mais dados do que está gravado na fita; há uma forte possibilidade de aparecer uma mensagem de erro.

Condições de Erro no Comando de Leitura para a Memória

O Monitor ouve o gravador cassette pelo menos durante 3,5 minutos antes de esperar dados de entrada. Isso permite ao Monitor ficar em fase com o tom de referência. Se a fita contém menos de 3,5 segundos desse tom, o Monitor perde o início da transmissão de dados do cassette, resultando em erro de verificação. Além disso, não se fica sabendo em que ponto o Monitor começou a ler a fita, porque ele sempre fica tentando ler dados da fita para a memória, dados válidos ou não. Neste caso ocorre uma mensagem de erro (um *bip* no alto-falante seguido da mensagem ERR no vídeo e o retorno ao Monitor). Para corrigir o erro, volte a fita ao início. Acione PLAY com o fio de conexão entre o Apple II e o gravador desconectado e meça o tempo de duração do tom. Se for menos de 3,5 segundos até o início dos dados, será necessário gravar os dados novamente. Isso em geral acontece quando se esquece de pular manualmente o início não magnético da fita.

A leitura de mais ou menos dados da fita para a memória do que aquilo que foi originariamente gravado provavelmente causa uma mensagem de erro na tela do Apple II. O último byte enviado da memória para o gravador durante o processo de gravação é o byte conhecido como *checksum*, um byte obtido pela somatória de todos os bytes gravados. Logicamente, seu valor depende de quantos dados e do valor de cada um deles. Quando na leitura o que é lido não coincide com o original, o Monitor não pode detectar qual byte lido é o *checksum.* Ele assume que o último byte que consegue ler seja o *checksum.* Assume também que o último byte que lê do cassette para a memória (determinado pelo endereço final no comando de leitura) será seguido pelo *checksum.* Ele executa seu próprio cálculo do *checksum* com os dados entrados na operação de leitura e o compara com o que é lido do cassette. Se os dois bytes não coincidem, uma mensagem de erro é mostrada. É pouco provável, mas possível, que os dois bytes sejam iguais, por pura coincidência. Como regra geral, deve-se ler a mesma quantidade de memória que foi gravada para permitir ao Monitor executar totalmente o controle através do *checksum.* Mais tarde, neste capítulo será mostrado como verificar uma operação de leitura de memória através do Monitor.

Guardando Memória em Disco

Com o Sistema Operacional em Disco do Apple II, guardar conteúdo de memória é muito mais fácil e confiável que em fita cassette. As regras para isso são um pouco diferentes porque se usa BASIC ao invés do Monitor para executar esta função. Este é um comando de nível superior ao de manuseio de cassette incluído no Monitor, e por isso deve ser usado sempre que possível. Antes de ler esta parte, é recomendável que se tenha familiaridade com o DOS (no Capítulo 5). O DOS deve estar na memória antes do início (veja Capítulo 2).

Estando no Monitor, deve-se deixá-lo temporariamente, indo ao BASIC sob o DOS com os

comandos CTRL-C ou CTRL-B quando se tem o Monitor Autostart, ou o comando 3DOG no Monitor padrão.

Aqui está um exemplo do comando do DOS para guardar conteúdo de memória em disco:

```
BSAVE TABFORMA, A$3000, L256, S6, D1, V201
```

Este comando faz criar um arquivo DOS em disco chamado TABFORMA. O parâmetro seguinte, A$3000, jogará para o disco a partir da posição de memória 3000 hexadecimal. O A se refere ao endereço. A terceira parte L256 especifica o comprimento em bytes da memória a ser gravada, neste caso 256 bytes decimais. O maior comprimento é 32767 decimal ($7FFF hexadecimal). O BSAVE permite usar números decimais ou hexadecimais como endereço ou comprimento. Em se usando valores hexadecimais, deve-se colocar o sinal de dólar à frente.

Os últimos três parâmetros no exemplo (S6, D1, V201) são opcionais. Eles especificam o disco a usar. Use o parâmetro S (slot) apenas quando tiver mais de um cartão controlador de disco e o disco que deve receber a gravação não é o mesmo que foi operado quando do último acesso ao disco (não é o corrente). Usualmente o slot 6 é o padrão. O parâmetro D (número de drive) é útil, porém não necessário; ele especifica o número do drive apenas se ele difere do último drive acessado, o corrente. O parâmetro V (número de volume) é o número de volume guardado no diretório do disco onde se está guardando memória. Se o volume especificado na declaração BSAVE difere do gravado no disco, aparece uma mensagem de erro.

Lendo para a Memória a Partir do Disco

Como foi feito com o BSAVE, usa-se o DOS para ler do disco para a memória. O comando BLOAD faz isso.

Aqui está um exemplo do BLOAD:

```
BLOAD TABFORMA, A$3000
```

Este exemplo carrega o arquivo chamado TABFORMA do disco em uso corrente e coloca os dados diretamente na memória, iniciando no endereço 3000 hexadecimal. Esta declaração de comando também aceita endereço decimal. O parâmetro A (endereço inicial), não é necessário se o conteúdo do arquivo inicia na mesma posição a partir de onde foi gravado. Só é necessário especificar endereço se ele for diferente do usado na definição de endereço inicial do BSAVE. O parâmetro L (comprimento) é opcional mas não imprescindível. O DOS verifica o comprimento do arquivo em disco e termina o comando de leitura automaticamente. Os parâmetros de disco opcionais do BSAVE (S, D e V) também podem ser usados no BLOAD para especificar slot, drive e volume.

Cuidado para não usar este comando em áreas de memória que podem estar ocupadas com o DOS, o interpretador Applesoft, páginas de texto ou gráficas, ou variáveis BASIC. É possível que se escreva sobre essas partes, destruindo dados que poderão ser necessários mais tarde.

MOVENDO E COMPARANDO BLOCOS DE MEMÓRIA

O Monitor tem duas funções que podem ajudar, quando se lê ou grava memória em fita. A função de mover, que copia conteúdo de blocos de memória de uma área para outra e a função de com-

parar dois blocos de memória de endereços distintos e mostrar as diferenças entre eles. Usados em conjunto com os comandos de ler e gravar dados, estas funções fornecem uma maneira adicional de controlar se os arquivos gravados têm os dados corretos.

O Comando de Mover Memória

Para mover dados de um endereço de memória para outro, deve-se fornecer o endereço receptor inicial (para onde os dados devem ir), o endereço fonte inicial (de onde os dados partirão) e o endereço fonte final, o último endereço com dado a ser movido. O formato deste comando é o do endereço inicial receptor, seguido do sinal de menor ou igual (<), o endereço inicial fonte, um ponto, o endereço final fonte, e a letra M (de mover). Como outros comandos do Monitor, todos os endereços são em hexadecimal.

Por exemplo, o comando:

```
*1200<2000.2100M
```

move dados para o endereço 1200 hexadecimal (o endereço inicial do receptor), a partir do bloco que inicia em 2000 hexadecimal (o endereço fonte inicial) e finalizando em 2100 (o endereço fonte final). O Monitor copia o conteúdo da memória, do endereço 2000 até 2100 para os endereços entre 1200 até 1300, neste exemplo. Como os endereços são em hexadecimal, o comprimento do bloco a ser copiado é 257 bytes decimal, ou 101 bytes em hexadecimal. O conteúdo original dos endereços entre 2000 e 2100 ficam intactos.

Quando se especifica os endereços no comando de mover, o endereço fonte inicial deve ser menor ou igual ao do fonte final. Se o endereço final for menor que o inicial, o Monitor só moverá um byte, para o endereço receptor inicial, saindo da operação de mover.

Preenchendo Memória

O comando de mover memória também a *preenche*. Preencher memória é carregá-la com um ou mais bytes repetidamente em endereços consecutivos. Suponha que se queira colocar zeros numa área de memória, iniciando no endereço 1D00 e finalizando em 1DFF. Usando de forma criativa os comandos de alterar e mover memória, pode-se preencher áreas de memória com um valor pré-determinado, ou padrão.

Para início, deve-se colocar zeros no primeiro byte da memória:

```
*1D00:00
```

Este é o primeiro passo para preencher memória. O segundo usa o comando de mover memória para copiar o conteúdo de um (ou mais) bytes em um bloco adjacente de memória.

Especifique o endereço receptor inicial como um acima do último byte do padrão (neste exemplo, deve ser 1D01). Coloque o endereço inicial do fonte no início do padrão (neste caso 1D00), e coloque o endereço final do fonte como o último a ser preenchido com o padrão (1DFF) menos o comprimento do padrão que se quer colocar no bloco de memória (1DFE).

Continuando o exemplo, o comando

```
*1D01<1D00.1DFEM
```

enche as posições de 1D01 até 1DFF de zeros (ou mais precisamente, com o conteúdo da posição 1D00). Este processo só funciona quando o padrão existe no início do bloco que se deseja preen-

cher. Aqui está o que acontece. Como o endereço inicial do receptor é um byte adiante do endereço inicial fonte, o Monitor move primeiro o conteúdo de 1D00 para 1D01, colocando este último endereço em 00. Quando o segundo byte é movido, vai-se de 1D01 para 1D02. O processo continua até que o conteúdo do endereço 1DFE (colocado em zero na última transferência de byte) vai para 1DFF. Observando essas posições pode-se ver os endereços de 1D00 a 1DFF preenchidos com zeros.

Pode-se desejar preencher memória de 1D00 a 1DFF com um padrão de quatro bytes, sendo necessária a alteração dos primeiros quatro bytes do bloco:

```
*1D00:00 5E 7F FF
```

O padrão já está lá. Para preencher memória até 1DFF com este padrão, coloque o comando:

```
*1D04<1D00.1DFB
```

Note que o endereço inicial do receptor vem um byte depois do final do padrão, o endereço inicial do fonte aponta o início do padrão e o endereço final do fonte aponta o último endereço a ser preenchido menos o comprimento do padrão:

```
1DFF
 -04     (Aritmética hexadecimal)
1DFB
```

Uma vez mais, ao se executar este exemplo pode-se examinar o conteúdo de 1D00 até 1DFF para observar o padrão repetido no bloco.

O Comando para Verificar Memória

Este comando do Monitor compra dois blocos de memória entre si, detectando as diferenças entre os dados. Pode-se usá-lo em conjunto com os comandos de leitura e gravação de memória suportados pelo DOS. Ao se gravar dados da memória em qualquer dispositivo e querendo saber se o processo ocorreu de forma correta, usa-se o comando de verificar a memória.

O formato para este comando é parecido com o de mover bloco de memória. Digite o endereço inicial de destino (onde inicia a comparação de memória) seguido imediatamente pelo sinal de menor (<), o endereço inicial do fonte (onde inicia a comparação com o destino), um ponto, e o endereço final do fonte (onde fica o último byte a ser comparado neste comando). O último item é a letra V, de verificar.

Aqui está um exemplo:

```
*32D0<0.CV
```

Com isso o Monitor é instruído para iniciar a comparação no endereço 32D0 contra o endereço 0, e continuar a comparação até o endereço 000C. Veja que não são necessários zeros não significativos na indicação de endereços para o Monitor.

Se o Monitor encontra um byte no bloco fonte que não coincide com o destino, são mostrados o endereço fonte e seu valor, bem como o destino e seu valor.

Por exemplo, suponha que se move do endereço 0000 até 000C para os endereços 32D0 até 32DC:

```
*32D0<0.CM
```

e em seguida se manda mostrar os blocos fonte e destino:

```
*0.C

0000- 4C 3C D4 4C 3A DB 8C 8C
0008- FF FF 4C 99 E1
*32D0.32DC

32D0- 4C 3C D4 4C 3A DB 8C 8C
32D8- FF FF 4C 99 E1
*
```

pode-se verificar visualmente a operação de transferência de dados. Se o endereço 32D8 for alterado do seu valor atual (FF) para 5A

```
*32D8:5A
```

e em seguida for dado o comando de verificar memória

```
*32D0<0.CV
```

o Monitor vai comprar o bloco fonte com o destino byte a byte até encontrar 0008 contra 32D8. Como este último foi alterado, o Monitor informa a discrepância

```
0008-FF (5A)
*
```

significa que o valor do endereço 0008 no bloco fonte não coincide com o mesmo endereço relativo do bloco destino (32D8). Primeiro é mostrado o endereço fonte (0008, cujo conteúdo é FF) e depois, entre parênteses, é mostrado o conteúdo divergente do destino (5A).

O endereço do bloco destino não aparece; o Monitor assume que se pode somar facilmente números na base 16 para achar o endereço do destino. Uma forma de evitar o cálculo é refazer a operação trocando o fonte com o destino:

```
*0<32D0.32DCV
```

o que resulta em:

```
32D8-5A (FF)
```

Esta mensagem mostra que o endereço fonte 32D8 contém 5A hexadecimal, com seu correspondente (endereço 0008 como visto acima) contendo FF. Este método elimina um cálculo, mas coloca outro; isto é, deve-se calcular o novo endereço fonte final (32DC).

Verificando Memória Guardada em Periféricos do Apple II

O comando de verificar memória é especialmente útil quando se guardam dados da memória em fita ou disco. Gravando uma porção de memória e lendo para uma área diferente, pode-se verificar se os dados foram gravados corretamente. O exemplo a seguir mostra como executar este processo

para programas em linguagem assembler, tabelas de formas, e outras informações que se podem guardar nos periféricos do Apple II.

No caso de se usar gravador cassette para guardar memória em fita, o primeiro passo é dar o comando de gravar memória como mostrado neste exemplo:

```
*2000.20FFW
```

Este comando grava dados da memória, iniciando no endereço 2000 e terminando em 20FF, em fita cassette. Não se esqueça de acionar a tecla RECORD *antes* de acionar RETURN. Uma vez que o alto-falante do Apple II emite o *bip* indicando que o processo de gravação terminou, desligue o gravador e rebobine a fita até onde o tom de referência foi dado. Assumindo que existe uma área de memória disponível de 2100 a 21FF, use o comando de ler dados da fita para a memória a partir do endereço 2100, finalizando em 21FF:

```
*2100.21FFR
```

Não se esqueça de acionar no gravador a tecla PLAY. Quando o alto-falante do Apple II emitir o *bip*, a operação de leitura estará concluída. Agora então pode-se comparar o dado gravado com o lido da fita:

```
*2000<2100.21FFV
```

O comando compara memória iniciando em 2000 até 20FF, com o que está gravado e foi lido do cassette, a partir do endereço 2100. Se não aparecer nenhuma diferença, pode-se ter certeza de que os dados foram gravados corretamente.

Para verificar memória gravada em disco, o mesmo processo se aplica. Antes de tentar gravar ou ler memória no disco, o Sistema Operacional (DOS) deve também estar presente. Aqui, o primeiro passo é dar um comando BSAVE de bloco de memória para disco:

```
BSAVE MEMDADO, A$2000,L$FF
```

A memória guardada em disco pode ser lida de volta com BLOAD:

```
BLOAD MEMDADO, A$ 2100
```

Note que o parâmetro de endereço nesta declaração é 256 bytes (decimal) maior que o endereço inicial de gravação. Quando o arquivo MEMDADO é lido, o comando de verificação compara os dois blocos:

```
CALL -151

*2000<2100.21FFV
```

Se não ocorrer diferenças, pode-se estar certo de que os dados da memória foram gravados corretamente no disco; de outra forma, o Monitor vai apontar as diferenças entre os blocos de forma que se possa corrigir.

O COMANDO GO

O Monitor tem um comando que transfere o controle do Apple II para o programa no endereço que se especifica. No início deste capítulo viu-se como deixar o Monitor e voltar para o Applesoft

em disco. O comando:

```
*3D0G
```

instrui ao Monitor para pular para o endereço 3D0 e executar instruções em linguagem de máquina a partir de lá. A letra G no final da linha de comando vem de GO. Se esta instrução for dada com o DOS na memória, o endereço 3D0 terá a primeira parte da instrução em linguagem de máquina:

```
JMP  $9DBF
```

Quando o Monitor transfere o controle para o endereço 3D0, o computador pula para o endereço 9DBF, onde as rotinas do DOS iniciam.

A forma geral do comando GO é o endereço para onde o controle vai, seguido da letra G. O endereço é opcional, o Monitor usa o apontador de memória se ele não for fornecido.

USANDO A IMPRESSORA

Se o Apple II está conectado a uma impressora via Interface Serial ou Cartão de Comunicação, pode-se usá-la para saída de dados. Para transferir a saída do vídeo para a impressora, coloque o número do slot onde está localizado o cartão de interface, seguido de CTRL-P e RETURN. Uma vez dado este comando, toda a saída normalmente colocada no vídeo será transferida para a impressora. Para voltar a saída para a tela, use número de slot 0 com o comando CTRL-P.

Quando for usado este comando, verifique que exista mesmo o cartão de interface no slot especificado. Se for indicado um lugar sem cartão, o Apple II irá travar, e a única forma de sair será acionando o RESET.

Os comandos de impressora operam exatamente como o comando PR# (número de slot) do BASIC. Ambos operam a chave CSW de dois bytes (chave de saída de caractere) no endereço 54 ($36). Esses dois bytes contêm o endereço que aponta para a rotina de saída de caracteres corrente. Trocando o número do slot com CTRL-P, troca-se o endereço dessa chave.

OS COMANDOS DE TECLADO

Este comando direciona o Monitor para aceitar entrada via outro dispositivo que não o teclado do Apple II. Digite o número do slot e CTRL-K, acionando então RETURN. Para voltar para o teclado, digite o mesmo comando com número de slot 0.

Neste caso é atuada a chave KSW (chave de entrada via teclado) no endereço 56 ($38) sendo colocado o endereço do dispositivo de entrada que for colocado no teclado.

INDICANDO MODOS DE DISPLAY

Para que o Monitor se apresente em modo vídeo reverso, entre o comando de inverter, abreviação I. Isso fará com que tudo o que o Apple II mostra na tela seja colocado como letras brancas em fundo preto. Entretanto, todos os comandos do Monitor digitados ficarão em modo vídeo normal.

Para sair do modo inverso, digite o comando para vídeo normal, abreviação N. Nenhum destes comandos necessita de parâmetros adicionais, além das letras I ou N.

ARITMÉTICA BINÁRIA EM OITO BITS USANDO O MONITOR

O Monitor executa funções de adição e subtração em oito bits. O resultado obtido também se apresenta com oito bits. Para executar adição, digite a primeira parcela hexadecimal seguida pelo sinal (+), seguida pela segunda parcela. Se o resultado é maior que FF, o Monitor trunca o dígito mais significativo e mostra os oito bits menores do resultado, como mostrado neste exemplo:

```
*7F+8A
=09
```

Para executar subtração, digite o minuendo, seguido pelo sinal (—) e o subtraendo. Como na adição, ambos os números devem ser hexadecimais. Se o resultado for menor que zero, o Monitor mostrará o complemento do resultado em 1, como se vê abaixo:

```
*0A-2D
=DD
```

COMANDOS DO MONITOR DEFINIDOS PELO USUÁRIO

Digitando-se CTRL-Y no Monitor, pode-se chamar uma rotina especial definida por usuário. O Monitor pula automaticamente para o endereço hexadecimal 3F8 ao se entrar com CTRL-Y. Nesse local existe espaço para uma instrução de *jump* em linguagem de máquina. Se existir alguma rotina em linguagem de máquina para acessar dentro da máquina, o endereço inicial dessa rotina deve ser colocado em 3F8.

O exemplo abaixo mostra como usar o CTRL-Y para reiniciar o Applesoft em disco sem ter que redigitar o conhecido comando 3DOG.

Primeiro, deve-se conhecer o formato da instrução *jump* em linguagem de máquina. Ela ocupará três bytes. O primeiro é o código da instrução, 4C. Os dois bytes seguintes são o endereço para onde será dado o salto, com os bytes *trocados*. Deve-se indicar o byte de menor valor primeiro (parecido com informar o sobrenome primeiro).

Aqui está um comando de alteração de memória para orientar o salto para o endereço 3D0:

```
*3F8:4C D0 03
```

Tente agora o CTRL-Y. Isso é bem melhor que 3DOG!

Para outro exemplo, veja como se pode usar o CTRL-Y para ir ao Mini-Assembler, descrito na seção seguinte em detalhes. Digite o comando do Monitor:

```
*F666G
```

aparece o caractere de entrada do Mini-Assembler:

O endereço onde o Mini-Assembler inicia, F666, pode ser usado na instrução de salto em 3F8:

```
!3F8:JMP $F666
03F8-    4C 66 F6     JMP $F666 ← O Mini-Assembler
!                                  mostra esta linha
```

Embora o Mini-Assembler não tenha sido estudado até agora, a linha acima altera o endereço da instrução *jump* em 3F8 hexadecimal para $F666, que é o endereço onde inicia o programa Mini-Assembler. Para voltar ao Monitor, digite o comando:

```
!$FF69G
*
```

Aparece o caractere de apresentação do Monitor na parte inferior esquerda da tela. Em seguida, se for acionado CTRL-Y, aparecerá o caractere do Mini-Assembler. Operando como o comando definido por usuário para acessar o Mini-Assembler, elimina-se a digitação de endereços. Altere o comando definido pelo usuário colocando um endereço diferente na instrução de saldo no endereço 3F8.

O MINI-ASSEMBLER

No Apple II padrão (ou cartão Integer BASIC com Apple II Plus), existe um programa em ROM que permite ao programador em linguagem de máquina evitar a tortura de montar o programa. O Mini-Assembler é acoplado ao Integer BASIC em ROM. É chamado de Mini porque o programador precisa usar endereços literais, ao invés de rótulos mnemônicos, como operandos nas declarações em linguagem assembler. Além disso, cada linha de código digitado é automática e imediatamente montada em linguagem de máquina. O principal problema aqui é que não se pode inserir ou eliminar instruções, como se faz usando montadores assembler completos junto com editores de texto.

A principal vantagem do Mini-Assembler é sua habilidade de entrar instruções em linguagem de máquina diretamente no Apple II, embora mantendo a facilidade de se operar com os mnemônicos das instruções na linguagem assembler.

O resto deste capítulo descreve o Mini-Assembler e diz como usá-lo. Ele não ensina conceitos de programação em assembler, nem mostra o conjunto de instruções do 6502 que o Apple II usa.

Assim, se houver ainda dificuldade de entender coisas como assembler, operandos, mnemônicos etc., pare por aqui. Aprenda primeiro as técnicas de programação em assembler e as instruções do 6502. Depois termine de ler este capítulo.

ACESSANDO O MINI-ASSEMBLER

O ponto de entrada do programa mini-assembler é F666 hexadecimal. Para partir vindo do Monitor, digite o comando:

```
*F666G
```

Isso causa o salto do Monitor para o Mini-Assembler. Estando em Integer BASIC ou Applesoft (versões em disco ou cassette), entre o comando de modo imediato:

```
CALL -2458
```

Ao entrar no mini-assembler o alto-falante dá um *bip*. O caractere de entrada do Mini-Assembler é o ponto de exclamação (!).

Erros Entrados

O Mini-Assembler detecta erros na digitação de instruções em linguagem assembler. Ele mostra os erros com *bip* no alto-falante e mostrando a instrução errada com o sinal ( ) sob o primeiro caractere incorreto da instrução. O contador de posições não é incrementado; ele permanece intacto de forma que se pode redigitar a instrução corretamente.

COMANDOS DE MONITOR NO MINI-ASSEMBLER

A qualquer momento dentro do Mini-Assembler, pode-se executar comandos do Monitor. Imediatamente à apresentação da exclamação (!), digite o sinal dólar ($), seguido do comando do Monitor. O exemplo abaixo mostra como examinar conteúdo de memória a partir do Mini-Assembler:

```
!$1CFF
1CFF- E6
!
```

Isso elimina a perda de tempo em se ir e voltar do Mini-Assembler ao Monitor. Pode-se digitar qualquer comando do Monitor estando no Mini-Assembler, apenas entrando o sinal dólar como primeiro caractere do comando. De fato, esse procedimento também deve ser usado na saída do Mini-Assembler.

DEIXANDO O MINI-ASSEMBLER

Para deixar o Mini-Assembler, use um dos comandos do Monitor, com o sinal dólar como prefixo. Para voltar ao Monitor, pule para o endereço FF69 com o comando $FF69G.

$CTRL-B e $CTRL-C, mandarão para o BASIC, exceto se estiver sendo usado o Applesoft em disco ou cassette. Para Applesoft em disco use $3DOG, e para Applesoft em cassette $0G.

FORMATOS DA INSTRUÇÃO

Embora não seja o objetivo desta seção ensinar programação em linguagem assembler, existem certos aspectos do Mini-Assembler que devem ser bem conhecidos antes que se possa usá-lo. Primeiro,

ele mantém um apontador de instrução separado do apontador de memória do Monitor. Deve-se acertar este apontador antes de se entrar com instruções. Segundo, existem vários formatos de instruções usados na programação do microprocessador 6502. Estes formatos dependem em grande parte do esquema de endereçamento usado.

O microprocessador 6502 tem onze modos de endereçamento, mas apenas seis formatos de instruções separados. Eles são descritos abaixo.

O primeiro, modo absoluto e direto, requer apenas o endereço de memória do operando com um ou dois bytes. Por exemplo:

```
AND $303A
```

O Mini-Assembler não requer o sinal de dólar ($) antes de endereços em hexadecimal; ele assume que todos os endereços estão na base 16.

O segundo formato é o de endereçamento imediato, como no exemplo:

```
LDA #$04
```

Veja o sinal (#) como primeiro caractere do operando. Ele é um indicador explícito de que o valor 04 será carregado no Acumulador. Sem este sinal, o Mini-Assembler interpreta a instrução como "pegue o conteúdo da posição de memória 0004 e coloque-o no Acumulador", que é uma instrução que referencia endereço absoluto.

Perceba também o uso confuso do termo *imediato.* Não confunda endereçamento imediato da linguagem assembler com execução imediata de programa em BASIC. As operações são diferentes. O termo é o mesmo e é usado de forma ambígua.

O terceiro formato é o método indexado, que se parece com:

```
CMP $23,X
```

ou

```
AND $80,Y
```

Ambas as instruções são semelhantes na medida em que os registradores X e Y aparecem como segundo operando. Basicamente, este formato gera uma instrução em linguagem de máquina que adiciona o conteúdo do registrador X ou Y ao endereço colocado no primeiro operando, e usa a soma como um endereço a ser referenciado pela instrução.

Agora, o formato indireto pré-indexado, como neste exemplo

```
AND ($F0,X)
```

indica que a soma do endereço ($F0) e do conteúdo do registrador X aponta para um endereço dos primeiros 256 bytes da memória que, por sua vez, contêm outro endereço que aponta o dado a ser usado na operação da instrução.

O modo de endereçamento indireto pós-indexado tem a seguinte forma:

```
ORA ($22),Y
```

Este formato de instrução usa o primeiro operando como ponteiro de um endereço de dois bytes, localizado, neste caso, na posição $22. A instrução soma o conteúdo do registrador Y ao endereço encontrado em $22; o dado no endereço derivado é usado na instrução em linguagem de máquina. O Mini-Assembler reconhece endereçamento indireto pós-indexado quando o primeiro operando está entre parênteses, como no exemplo acima.

O endereçamento indireto é um pouco mais direto que o indexado. Aqui está um exemplo:

```
JMP  ($22FE)
```

Aqui, a instrução JMP não carrega o endereço $22FE no contador de programa. Em vez disso, o endereço de dois bytes da posição $22FE é carregado no contador de programa. Dessa forma, o operando no formato indireto é em verdade um apontador ao invés de um endereço literal.

USANDO O MINI-ASSEMBLER

Como visto na seção anterior, o Mini-Assembler mantém um contador de posição que é incrementado do tamanho de cada declaração em assembler entrada. Em outras palavras, uma vez digitada uma declaração em linguagem de máquina (toda vez que se digita uma instrução), o Mini-Assembler calcula seu comprimento (1, 2 ou 3 bytes) e incrementa o contador de posição para a linha seguinte.

O primeiro passo no uso do Mini-Assembler é acertar o contador de posição. Faça-o como parte da primeira declaração em linguagem de máquina digitada. Por exemplo:

```
!8DB0:LDA #$04
```

Imediatamente após o caractere de entrada do Mini-Assembler, digite o endereço base para os códigos em linguagem de máquina a serem entrados (neste caso, 8DB0), seguido de dois pontos (:) e a primeira declaração em linguagem assembler. Não será necessário digitar a posição para a declaração seguinte. O Mini-Assembler calcula o endereço seguinte, a menos que se vá colocar instruções em outro endereço fora da seqüência, como mostrado acima.

Uma vez posicionado o contador de posição, entre as instruções em linguagem assembler, uma por linha. Após a primeira linha, entre um espaço seguido pela declaração seguinte, como aqui:

```
! JSR FB1E
```

Isso leva o Mini-Assembler a calcular o valor seguinte do contador de posição.

Uma Seção de Exemplo

Esta seção mostra a operação do Mini-Assembler passo a passo e em detalhes. O seu objetivo é criar um pequeno programa que use as entradas de controle de jogo do Apple II e o alto-falante para criar sons. O procedimento para este programa é ler valores dos *paddles* 0 e 1, usando a subrotina PREAD (no endereço FB1E). O valor do *paddle* 0 dará o intervalo entre os sinais do falante (0=menor atraso, FF=maior atraso), e o valor do *paddle* 1 será outro intervalo, inversão ao *paddle* 0 (0=intervalo mais longo, FF=intervalo mais curto).

O programa inicia em 1D00 e usa o endereço 1CFF para guardar a leitura do *paddle* 0.

Quando se entra cada linha do programa em linguagem assembler, o Mini-Assembler sobrepõe a linha entrada com o valor do contador de posição corrente, código de operação e operando em forma de linguagem de máquina (também conhecida como código objeto), bem como o mnemônico da instrução digitada. Por exemplo:

```
1D00-     A2 00          LDX     #$00
```

O contador de posição é mostrado no início da linha seguido por um traço. Após esse campo, o código de operação (A2 para a instrução LDX) aparece, seguido pelo último byte da instrução. No caso de instruções de três bytes (aquelas que referenciam endereços de dois bytes), o byte de menor ordem aparece antes do de maior ordem. Por fim, é mostrado o código mnemônico da instrução.

O exemplo mencionado aparece abaixo. Veja que cada linha produzida pelo Mini-Assembler aparece aqui sob a linha entrada que a gerou.

```
!1D00:LDX #$00  ← Acerta o contador de posição e dá a primeira instrução
1D00-   A2 00       LDX  #$00
! JSR FB1E  ← Todos os números são hexadecimais (não é necessário o prefixo $)
1D02-   20 1E FB    JSR  $FB1E
! STY 1CFF
1D05-   8C FF 1C    STY  $1CFF
! INX
1D08-   E8          INX
! JSR FB1E
1D09-   20 1E FB    JSR  $FB1E
! LDA C030
1D0C-   AD 30 C0    LDA  $C030
! DEC 1CFF
1D0F-   CE FF 1C    DEC  $1CFF
! BNE 1D0C  ← O Mini-Assembler calcula o salto relativo (F8)
1D12-   D0 F8       BNE  $1D0C
! LDA C030
1D14-   AD 30 C0    LDA  $C030
! INY
1D17-   C8          INY
! BNE 1D14
1D18-   D0 FA       BNE  $1D14
! JMP 1D00
1D1A-   4C 00 1D    JMP  $1D00
```

Após digitar este programa, confira. A melhor forma de fazer isso é listá-lo na memória, de preferência no formato em linguagem assembler. Para isso será necessário usar o Monitor como se verá a seguir.

Como segurança adicional, pode-se guardar o programa em cassette (comando W no Monitor) ou disco (com a declaração BSAVE do BASIC).

Para rodar o programa, pule para o endereço 1D00. Use o comando G no Monitor, ou CALL 7424 no BASIC. Opere com os controladores de jogo e veja como eles afetam o alto-falante. Para finalizar o programa, acione RESET.

LISTAGEM DE DESASSEMBLER

O Monitor tem um comando que pode ser usado para listar instruções de máquina em formato de linguagem assembler, mesmo que o Apple II não tenha o Mini-Assembler em ROM. O comando

L, de Listar, faz desassembler de 20 instruções de linguagem de máquina para declarações em assembler mostrando-as na tela ou em outro dispositivo selecionado. O comando L usa o contador de posição como ponteiro para a próxima instrução a ser operada no desassembler. Dessa forma, digite L após entrar com o programa acima, e o processo se iniciará no endereço 1D1D, sem mostrar nada do que foi digitado.

É bom acertar o contador de posição ao usar o comando L. Aqui está a listagem do programa exemplo acima:

```
!$1D00L
1D00-   A2 00     LDX  #$00
1D02-   20 1E FB  JSR  $FB1E
1D05-   8C FF 1C  STY  $1CFF
1D08-   E8        INX
1D09-   20 1E FB  JSR  $FB1E
1D0C-   AD 30 C0  LDA  $C030
1D0F-   CE FF 1C  DEC  $1CFF
1D12-   D0 F8     BNE  $1D0C
1D14-   AD 30 C0  LDA  $C030
1D17-   C8        INY
1D18-   D0 FA     BNE  $1D14
1D1A-   4C 00 1D  JMP  $1D00
1D1D-   9F        ???
1D1E-   4E A5 12  LSR  $12A5
1D21-   A4 96     LDY  $96
1D23-   A3        ???
1D24-   D0 A4     BNE  $1CCA
1D26-   EF        ???
1D27-   A4 62     LDY  $62
1D29-   A2 70     LDX  #$70
```

Neste caso, as oito últimas instruções do desassembler são imateriais, porque o programa termina no endereço 1D1A.

Veja que o comando L é uma facilidade do Monitor, independente do Mini-Assembler, (portanto com o dólar como prefixo). Entrando com L (ou $L no Mini-Assembler) e acionando RETURN sem dar a posição inicial do contador, o Monitor lista o desassembler das próximas 20 instruções que encontrar depois das que foram listadas.

TESTANDO E DEPURANDO PROGRAMAS

Junto com o Mini-Assembler, o Monitor do Apple II padrão oferece ferramentas de depuração que ajudam muito quando se programa em linguagem assembler. Os programas em linguagem de baixo nível são provavelmente os mais difíceis de depurar e testar. Ao invés de simplesmente mostrar conteúdo de variáveis, deve-se inspecionar posições de memória, registradores e o programa em si. Dois comandos do Monitor, Step e Trace, são usados para esse propósito.

Tanto o Step quanto o Trace não são disponíveis na versão do Apple II com o Autostart Monitor.

O Comando Step

Embora seja bastante fácil testar um programa em linguagem assembler observando os sinais de execução de suas operações, este não é um método muito eficiente de isolar os erros. Se o programa é pequeno, fica possível rodá-lo passo a passo, verificando os resultados de cada instrução à linguagem de máquina após sua execução pelo Apple II. Isso é o que se faz através do comando Step.

Quando se executa um comando Step, o Monitor desassembliza e mostra a instrução apontada pelo contador de posições, executa-a, mostra o conteúdo dos registradores do microprocessador, e volta o controle para o Monitor do Apple II.

O formato do comando Step é um parâmetro de endereço opcional (para posicionar o contador de posição) seguido da letra S.

O comando Step abaixo executa a primeira instrução do nosso programa exemplo de geração de sons. O conteúdo do registrador X mostrado é 0. Os outros três registradores cujos conteúdos são mostrados não se alteram.

```
*1D00S
1D00-    A2 00        LDX   #$00
 A=FF  X=00  Y=8C  P=32  S=F8
*
```

Pode-se misturar o comando Step com outros do Monitor (tal como examinar memória). Para ver isso em ação, vá através do programa de sons até a instrução STY na posição de memória 1D05. Será necessário entrar com o comando S nove vezes, porque a instrução JSR na posição 1D02 chama uma sub-rotina na posição FB1E e deve-se percorrê-la passo a passo até que se volte para 1D05. Uma vez lá, use o comando de examinar memória do Monitor para ver a posição 1CFF. A instrução no endereço 1D05 guarda o que foi no *paddle* 0 no endereço 1CFF.

```
*S
1D05-    8C FF 1C     STY   $1CFF
 A=FF  X=00  Y=00  P=32  S=F8
*1CFF
1CFF-    00
*
```

Como se pode ver com o comando de examinar memória, o conteúdo do registrador Y (3F) agora está guardado no endereço 1CFF. Pode-se também misturar outros comandos do Monitor com o comando Step.

O Comando Trace

Muitas vezes o programa é muito longo para que se execute cada instrução passo a passo. Em vez disso, fica mais interessante que as instruções sejam executadas e só se interrompa a execução quando for necessário. O comando Trace do Monitor faz isso. Sua saída é semelhante à do Step, com exceção ao fato que o Trace elimina o trabalho de se colocar a instrução Step para cada instrução executada pelo Apple II. Para parar com o comando Trace, acione a tecla RESET ou entre a instrução assembler BRK no programa; quando o Monitor encontra esta instrução, ele devolve o controle para o teclado (via Monitor).

O formato do comando Trace é constituído de um endereço opcional seguido da letra T. Aqui está a primeira parte da execução do trace no programa de sons do exemplo:

```
*1D00T
1D00-   A2 00       LDX  #$00
 A=FF X=00 Y=8C P=32 S=F6
1D02-   20 1E FB    JSR  $FB1E
 A=FF X=00 Y=8C P=32 S=F6
FB1E-   AD 70 C0    LDA  $C070
A=00 X=00 Y=8C P=32 S=F4
FB21-   A0 00       LDY  #$00
A=00 X=00 Y=00 P=32 S=F4
FB23-   EA          NOP
A=00 X=00 Y=00 P=32 S=F4
FB24-   EA          NOP
A=00 X=00 Y=00 P=32 S=F4
FB25-   BD 64 C0    LDA  $C064,X
A=00 X=00 Y=00 P=32 S=F4
FB28-   10 04       BPL  $FB2E
A=00 X=00 Y=00 P=32 S=F4
FB2E-   60          RTS
A=00 X=00 Y=00 P=32 S=F4
1D05-   8C FF 1C    STY  $1CFF
A=00 X=00 Y=00 P=32 S=F6
1D08-   E8          INX
A=00 X=01 Y=00 P=32 S=F6
1D09-   20 1E FB    JSR  $FB1E
A=00 X=01 Y=00 P=30 S=F6
FB1E-   AD 70 C0    LDA  $C070
A=27 X=01 Y=00 P=30 S=F4
FB21-   A0 00       LDY  #$00
A=27 X=01 Y=00 P=30 S=F4
FB23-   EA          NOP
A=27 X=01 Y=00 P=30 S=F4
FB24-   EA          NOP
A=27 X=01 Y=00 P=30 S=F4
FB25-   BD 64 C0    LDA  $C064,X
A=27 X=01 Y=00 P=30 S=F4
FB28-   10 04       BPL  $FB2E
A=27 X=01 Y=00 P=30 S=F4
FB2E-   60          RTS
```

A desvantagem em se usar o comando Trace sobre o Step é que se tem menos controle sobre a execução de cada passo do programa. Quando se entra o programa pela primeira vez, deve-se colocar a instrução BRK em pontos-chave. Essas são as junções na lógica do programa. É lógico que

se pode trocar essas instruções por instruções NOP (código de não operação), porém isso não é interessante depois de se ter depurado o programa.

Adicionalmente, o comando Trace opera numa fração do tempo gasto pelo programa em linguagem assembler. Por exemplo, tente substituir no programa exemplo, a instrução do endereço 1D1A por um BRK. Digitando o comando 1D00G esta instrução leva uma fração de segundo para ser executada. Entretanto, digitando 1D00T, o programa leva de 60 a 70 segundos para ser executado. Portanto, tendo um programa grande para ser testado, seja moderado com o comando Trace se desejar alguma ajuda dele.

Mais sobre o Contador de Posições

Como foi mencionado antes na seção sobre o Comando Step, pode-se usar muitos comandos do Monitor alternados com o Trace e o Step. Entretanto, existem algumas exceções a essa regra. Os comandos List, Go e o definido pelo usuário CTRL-Y modificam o contador de posição quando são digitados. Isso altera o fluxo do programa que se está rastreando a menos que se devolva o contador de posição após a execução de qualquer um deles. O exemplo abaixo mostra como o valor do contador de posição é alterado usando um desses comandos:

```
*1D00S
 1D00-   A2 00           LDX  #$00
  A=62 X=00 Y=00 P=32 S=F8
*S
 1D02-   20 1E FB        JSR  $FB1E
  A=62 X=00 Y=00 P=32 S=F8
*L                                  ← Comando de listar aqui
FB1E-    AD 70 C0       LDA  $C070
FB21-    A0 00          LDY  #$00
FB23-    EA             NOP
FB24-    EA             NOP
FB25-    BD 64 C0       LDA  $C064,X
FB28-    10 04          BPL  $FB2E
FB2A-    C8             INY
FB2B-    D0 F8          BNE  $FB25
FB2D-    88             DEY
FB2E-    60             RTS
FB2F-    A9 00          LDA  #$00
FB31-    85 48          STA  $48
FB33-    AD 56 C0       LDA  $C056
FB36-    AD 54 C0       LDA  $C054
FB39-    AD 54 C0       LDA  $C051
FB3C-    A9 00          LDA  #$00
FB3E-    F0 0B          BEQ  $FB4B
FB40-    AD 50 C0       LDA  $C050
FB43-    AD 53 C0       LDA  $C053
```

```
FB46-     20 36 F8          JSR   $F836
*S                      ← Trocando o contador de posição, o próximo passo
                            será em $FB49 em lugar de $FB1E
FB49-     A9 14             LDA   #$14

 A=14 X=00 Y=00 P=30 S=F6
*S
FB4B-     85 22             STA   $22
 A=14 X=00 Y=00 P=30 S=F6
*
```

Sub-rotinas Úteis do Monitor

Em alguns casos, o BASIC não é suficientemente poderoso para executar todas as funções necessárias em um programa. Isso, claramente, é uma das razões pelas quais os programadores recorrem a sub-rotinas em assembler em seus programas BASIC. Esta seção mostra como referenciá-las dentro de um programa BASIC.

Pelo uso de programas em linguagem assembler dentro de BASIC, pode-se criar tantos problemas quanto os que se propõe resolver. Em que lugar da memória ficará o programa assembler? Lembre-se de que a memória do Apple II tem quatro grandes áreas reservadas (página de texto, de gráficos de baixa resolução e duas páginas de gráficos de alta resolução). O DOS e o interpretador do Applesoft também usam memória. Colocar um programa onde não cause problemas, depende do tamanho da memória e da versão do Apple II usada.

O Monitor é a melhor fonte para sub-rotinas em linguagem assembler por três razões: primeiro, sendo em ROM, não é necessário se preocupar com código relocável. Segundo, as rotinas do Monitor estão depuradas; e última, as rotinas intrínsecas não usam um byte adicional sequer. As sub-rotinas úteis do Monitor estão listadas no Apêndice D.

Incorporando a Sub-rotina

Decidindo usar uma sub-rotina do Monitor dentro de um programa em BASIC, primeiro certifique-se que não há nada em BASIC equivalente a ela. Isso pode evitar que um programa seja mais complicado que o necessário. Em seguida, verifique se a rotina precisa de parâmetros passados a partir do programa em BASIC. Se for necessário atribuir valores aos registradores do microprocessador antes de executar a sub-rotina, ou se o seu resultado vem dentro de registradores, será necessário usar instruções em assembler adicionais para interfacear com o BASIC. Muitas sub-rotinas do Monitor não precisam de parâmetros do BASIC; aquelas que precisam, em geral têm equivalentes no BASIC.

Uma vez sabido qual sub-rotina usar, é aconselhável documentá-la a fim de tornar mais clara a operação. Por exemplo, CALL-936 limpa a tela e coloca o cursor no alto à esquerda. Uma forma de tornar esse CALL mais descritivo é gerar uma variável no início do programa, como aqui:

```
10 LIMPTELA=-936
```

e para referência mais tarde no programa:

```
1510 CALL LIMPTELA
```

Isso torna a função mais clara a qualquer um que mais tarde venha a ler o programa, porém insere uma variável a mais. Esses elementos de estilo tornam os programas bem mais fáceis de serem lidos e entendidos.

Problemas a Evitar

Tendo um editor/assembler disponível para o Apple II, é fácil relocar programas reposicionando o ponto de origem e reassemblizando. Por outro lado, pode-se escrever uma sub-rotina em linguagem de máquina com o Mini-Assembler, usada com o BASIC. Esta alternativa pode gerar problemas de incompatibilidade, tendo-se muitas vezes necessidade de reescrever a sub-rotina para outra versão do Apple II com tamanho de memória diferente. Isso acontece quando se usam áreas de memória ocupadas com o DOS, as páginas gráficas ou o Applesoft em disco ou cassette. Use as sub-rotinas do Monitor sempre que possível.

Com programas em Applesoft, use sempre a função USR se tiver que passar parâmetros de ou para a sub-rotina, ao invés da declaração CALL. Os endereços 9D até A3 guardam valores passados pelo USR, e pode-se usar esta área para passar parâmetros de volta ao BASIC. Use a declaração POKE para colocar a instrução JUMP nas posições 10 até 12 (0A até 0C hexadecimal). Estas posições devem conter uma instrução JUMP para o início da sub-rotina em linguagem de máquina chamada pelo USR.

INTEGRAÇÃO DO PROGRAMA COM O BASIC

Em BASIC, as declarações LOMEM: e HIMEM: protegem o programa em assembler de ser "sujado" pelo BASIC. Existem algumas restrições na determinação de um programa em linguagem assembler se também tivermos o Applesoft em disco ou cassette ou ainda o DOS na memória ao mesmo tempo. A forma geral ao se usar sub-rotinas em assembler com programa em BASIC é a seguinte:

1. Carregue o DOS.
2. Carregue o interpretador Applesoft do disco ou cassette, se necessário.
3. Informe os valores para LOMEM: e HIMEM:.
4. Carregue o programa em linguagem assembler.
5. Carregue o programa BASIC do disco ou fita (ou digite-o).

O DOS e o interpretador Applesoft mexem com o HIMEM: após serem carregados. Deve-se reposicionar LOMEM: e HIMEM: para dar espaço ao programa em assembler, e em seguida carregá-lo em lugar seguro. Em seguida, carregar e rodar um programa BASIC afetará apenas a área entre o LOMEM: e o HIMEM:.

Para maior ajuda em como achar espaço para programas em linguagem assembler, veja o mapa de memória no Apêndice G. Veja também as referências ao LOMEM: e ao HIMEM: no Capítulo 8.

8

Compêndio de Declarações do Basic

Este capítulo descreve a sintaxe de todas as declarações e funções do BASIC do Apple II. As declarações são descritas em primeiro lugar, listadas em ordem alfabética. A seguir vêm as funções, descritas e listadas também em ordem alfabética.

Este capítulo serve como referência para todas as declarações e funções. O Capítulo 3 até o 7 descrevem conceitos de programação. São também dados exemplos de emprego de funções e declarações em programas.

MODOS IMEDIATO E PROGRAMADO

Muitas declarações podem ser executadas em modo imediato ou programado. A menos que se especifique o contrário, assuma que uma declaração pode ser usada em ambos os modos. As exceções são identificadas. Existem declarações que são restritas a um modo de operação, não funcionando no outro, outras existem que, embora operando nos dois módulos, somente em um deles são práticas.

Algumas declarações se reportam ao DOS (*Disk Operating System*). Elas podem ser empregadas em modo imediato, como foi mostrado. Em modo programado, entretanto, elas devem ser inseridas como parte de uma declaração PRINT, antecedidas pelo caractere CTRL-D (código ASCII 04). Assim, as duas declarações abaixo são equivalentes:

```
]CATALOG

]PRINT CHR$(4); "CATALOG"
```

Veja que em vez de se usar CHR$(4) como visto acima, pode-se digitar o sinal de aspas, seguido por CTRL-D, e outro sinal de aspas. Na tela só aparecerão as duas marcas de aspas, sem nada entre elas. O caractere CTRL-D estará lá, porém não poderá ser visto.

VERSÕES DO BASIC

Todas as declarações e funções são disponíveis tanto em Integer BASIC quanto em Applesoft a menos que seja especificado o contrário. Onde uma declaração ou função operar diferentemente para as duas versões, haverá a anotação.

CONVENÇÕES DE NOMENCLATURA E FORMATO

Usaremos um esquema padrão para apresentar a forma geral de cada declaração e função. Abaixo estão listadas pontuações e outras convenções mecânicas que usamos.

{ }	Chaves indicam uma escolha de itens. Um dos itens do grupo deve estar presente; as chaves não aparecem em uma declaração real.
[ ]	Colchetes indicam que o parâmetro interno é opcional; os colchetes não aparecem em uma declaração real.
	Indica que o procedimento anterior pode se repetir; não aparece em uma declaração real.
Número de linha	Um número de linha inicial é sempre aplicado para declarações em modo programado.
Outras pontuações	Todas as outras marcas de pontuação — vírgulas, dois pontos, aspas e parênteses — devem aparecer como mostrado.
MAIÚSCULAS	As palavras em maiúsculas devem aparecer exatamente como mostrado.
Itálicos	Itálicos são usados para termos genéricos. O programador informa a palavra ou valor exato, de acordo com a forma genérica descrita abaixo.

Os termos genéricos em itálico seguintes são usados em definição de declarações e funções. Qualquer termo itálico não listado aqui é particular àquela função em que aparece, e lá é descrito.

col	Número de coluna para gráfico de baixa resolução; uma expressão numérica que tem valor entre 0 e 39.
colh	Número de coluna de gráfico de alta resolução; uma expressão numérica que pode ter valor entre 0 e 279.
const	Qualquer constante numérica ou série.
D*n*	Um número de drive de disco deve ser especificado como D1 ou D2.
expr	Uma série numérica, constante relacional ou Booleana (apenas em Applesoft), variável ou expressão; qualquer combinação destas.
expr$	Qualquer constante série, variável ou expressão.
exprnm	Qualquer constante numérica, variável ou expressão.
nomearq	Qualquer nome de arquivo.
linha	Qualquer número de linha de programa BASIC.

linha;	Um dos vários números de linha do programa BASIC.
endmem	Uma expressão numérica, variável ou constante que se relaciona a um endereço de memória. Os endereços de memória variam de –32767 a 32767, ou em Applesoft, de 0 a 65535, onde –65535 é equivalente a 1, –65534 a 2 etc.
posmem	Uma posição de memória especificada por um inteiro constante entre 0 e 65535 (decimal) ou $0 a $FFF (hexadecimal). As constantes hexadecimais são identificadas pelo sinal de dólar ($) à frente.
mensagem	Qualquer série de texto entre aspas.
linha	Número de linha em gráfico de baixa resolução; uma expressão numérica com valor entre 0 e 47.
linhah	Número de linha em gráfico de alta resolução; uma expressão numérica com valor entre 0 e 191.
S*n*	Número do slot para entrada e saída; deve ser S0, S1, S2, S3, S4, S5, S6 ou S7.
var	Em Integer BASIC, uma variável numérica ou série. Em Applesoft, qualquer variável numérica, inteira ou série.
varnm	O nome de uma variável numérica.
var(sub)	Em Integer BASIC, o nome de uma variável numérica subscrita. Em Applesoft, qualquer variável inteira, numérica ou série.
V*n*	Qualquer número de volume para identificação de disco (entre V0 e V255).

DECLARAÇÕES

APPEND

Abre um arquivo (veja OPEN), e posiciona o ponteiro do arquivo no final.

Formato:

APPEND *nomearq* [, D*n*] [, S*n*] [, V*n*]

Uma área de buffer de 595 bytes é alocada para o arquivo de texto especificado. Ele deve ser seqüencial. O comando WRITE pode agora ser usado para guardar dados no disco, iniciando no primeiro byte não usado. Isso será imediatamente à frente do último caractere do arquivo, a menos que existam bytes não usados no meio.

Ocasionalmente, o APPEND pode não começar no primeiro byte não usado do arquivo (em geral no fim). Em vez disso ele começa no início do arquivo (Horror!). Para certificar-se de que

TABELA 8.1. Determinações em Linguagem de Máquina para o APPEND.

LINGUAGEM DE MÁQUINA		LINGUAGEM ASSEMBLER DO 6502	
Decimal	**Hexadecimal**	**Instrução**	**Comentário**
169	A9	LDA $0	A rotina do Monitor em $FDED põe o caractere do registrador A ($0 no caso) no dispositivo de saída corrente, o disco. Veja Apêndice D.
0	0		
76	4C	JMP $FDED	
237	ED		
253	FD		

isso não ocorreu, seu programa deve sempre gravar uma indicação de fim de arquivo antes de fechá-lo. Uma pequena rotina em linguagem de máquina, descrita na Tabela 8.1 faz isso. Coloque com POKE as instruções em qualquer lugar da memória onde existirem 5 bytes livres (posições de 768 até 772 são boas a menos que já estejam sendo usados para outra coisa). Chame então a sub-rotina (CALL) antes de dar a instrução de fechar o arquivo.

Se o arquivo não existe no drive D*n* do slot S*n*, aparece a mensagem FILE NOT FOUND. Se o disco no drive D*n* do slot S*n* não tem volume V*n*, aparece a mensagem de erro VOLUME MISMATCH. V0 serve para qualquer disco. Se o arquivo já está aberto, o APPEND o fecha e reabre (veja CLOSE).

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* é omitido, o drive ou slot referenciado por último é usado. V0 se aplica quando V*n* está ausente. Também, *n* pode estar ausente; serão usados D0, S0 ou V0.

O APPEND é um comando do DOS, requerendo PRINT e CTRL-D em modo programado.

Não pode ser usado em modo imediato.

AUTO

Gera numeração automática de linha em Integer BASIC.

Formato:

AUTO *linha* [, *incremento*]

Os números de linhas são mostrados automaticamente a cada vez que se digita a tecla RETURN, iniciando com *linha*, e incrementando cada vez de incremento, com a adoção de 10 se não for especificado. Digite CTRL-X para limpar um número de linha automático; a numeração automática continua a menos que se digite MAN na linha de texto (veja MAN).

Só pode ser usado em modo imediato.

Não é disponível em Applesoft.

BLOAD

Pega um arquivo binário do disco e guarda em um setor específico da memória.

Formato:

BLOAD *nomearq* [, A*posmem*] [, D*n*] [, S*n*] [, V*n*]

Se o parâmetro A está ausente, o arquivo especificado é colocado na memória a partir da posição de onde foi tirado na gravação (veja BSAVE). Se o parâmetro A está presente, o arquivo vai para a memória para *posmem.*

O BLOAD deve ser usado com cuidado. Aquilo que estiver na área de memória onde ele coloca o arquivo (como um programa, o Applesoft, DOS etc.) será destruído.

Se o arquivo não existe no drive D*n* no slot S*n*, aparece a mensagem FILE NOT FOUND. Se o disco do drive D*n* do slot S*n* não tiver o volume V*n*, resulta no erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* são omitidos, o drive ou slot referenciados por último são assumidos. Usa-se V0 se V*n* está ausente.

Este é um comando do DOS, requer PRINT e CTRL-D em modo programado.

BRUN

Pegue um arquivo binário (que poderia ser um programa em linguagem de máquina), guarde-o numa posição específica de memória, e execute um salto em linguagem de máquina (JMP no código assembler do 6502) para a posição inicial de memória.

Formato:

BRUN *nomearq* [, A*posmem*] [, D*n*] [, S*n*] [, V*n*]

Se o parâmetro A está ausente, o arquivo especificado é colocado na memória iniciando na posição de onde ele foi tirado para gravar (veja BSAVE). Se o parâmetro A estiver presente, o arquivo é colocado a partir da posição *posmem.*

Um programa em linguagem de máquina pode operar bem em qualquer posição de memória. Cuidado com as instruções que são dependentes de endereço antes de carregar o programa para uma área diferente de memória. O BRUN destrói tudo que estiver na área de memória onde ele coloca o arquivo; isso pode destruir o Applesoft ou o DOS no processo.

Se o arquivo não existe no drive D*n*, aparece a mensagem de erro FILE NOT FOUND. Se o disco no drive D*n* e slot S*n* não tem o volume V*n*, aparece a mensagem de erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* e S*n* são omitidos, o drive e slot referenciados por último são usados. Usa-se V0 quando V*n* está ausente.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

BSAVE

Cria um arquivo em disco e guarda uma área de memória do Apple II nele, em binário.

Formato:

BSAVE *nomearq*, A*posmem*, L*compr* [, D*n*] [,S*n*] [, V*n*]

O parâmetro A especifica o endereço inicial da área de memória a ser gravada. O parâmetro L dá o número de bytes a guardar. *compr* deve ser um inteiro entre 0 e 32767 (decimal). Também pode ser em hexadecimal, quando fica precedido pelo sinal dólar ($).

Se o disco no drive D*n* do slot S*n* não tem o volume V*n*, ocorre o erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* é omitido assume-se o último drive ou slot que foi acessado. Usa-se V0 se V*n* está ausente. Também o *n* pode estar ausente, e no caso é adotado D0, S0 e V0.

Este é um comando DOS, requer PRINT e CTRL-D quando usado em modo programado.

CALL

Pula para rotina em linguagem de máquina num endereço específico.

Formato:

CALL *endmem*

O CALL pode ser usado com sub-rotinas que se escrevem, ou mesmo com sub-rotinas intrínsecas tais como as listadas no Apêndice D.

CATALOG

Mostra uma lista com todos os arquivos de um disco especificado.

Formato:

CATALOG [, D*n*] [, S*n*]

O CATALOG primeiro imprime DISK VOLUME seguido pelo número de volume do disco. Se o parâmetro V*n* é incluído no comando CATALOG, é ignorado.

A lista de arquivos do disco é mostrada abaixo do número de volume. Para cada arquivo, o CATALOG coloca um código literal indicando o tipo de arquivo, o número de setores requeridos para guardar o arquivo, e seu nome. Um asterisco aparece à esquerda do tipo de arquivo se ele está bloqueado (veja LOCK). Os tipos de arquivo e seus códigos são:

I	Programa em Integer BASIC
A	Programa em Applesoft
T	Arquivo de texto
B	Arquivo binário (linguagem de máquina)

Se o tamanho do arquivo excede 255 setores, o comprimento é dado como módulo 255; ou seja, 0 para 256, 1 para 257 etc.

D*n* e S*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* são omitidos, o drive referenciado pela última vez é usado.

Este é um comando DOS, e requer PRINT e CTRL-D quando usado em modo programdo.

CHAIN

Carrega e roda um programa em Integer BASIC a partir do disco, sem limpar os valores de qualquer variável ou matriz.

Formato:

CHAIN *nomearq* [, D*n*] [, S*n*] [, V*n*]

O comando CHAIN só pode ser usado em Integer BASIC, e apenas para carregar programas em Integer BASIC.

Se o arquivo não existe no drive D*n* e slot S*n*, aparece a mensagem de erro FILE NOT FOUND. Se o disco no drive D*n* e slot S*n* tiver volume V*n*, aparece a mensagem de erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* são omitidos, o drive referenciado por último é usado. Usa-se V0 para V*n* ausente. Também *n* pode estar ausente, quando são usados D0, S0 e V0.

CLEAR

Esta declaração do Applesoft zera todas as variáveis numéricas e elementos de matrizes. Também atribui um valor nulo a variáveis séries e elementos de matriz tipo série.

Formato:

CLEAR

Executar esta declaração equivale a desligar o Apple II, ligá-lo de novo e carregar o programa na memória. Um programa existente na memória continuará rodando normalmente após os CLEAR pois ele não afeta a lógica de programas.

Em Integer BASIC, use CLR.

CLR

Este comando do Integer BASIC atribui valor 0 a variáveis e elementos de matrizes e valor nulo a variáveis tipo série.

Formato:

CLR

Ele também retira o dimensionamento de matrizes e séries. Pode-se imprimir valores de matrizes após a execução da declaração CLS, desde que não tenha sido atribuído valores a nenhuma variável nesse ínterim.

O CLR só pode ser usado em modo imediato.

Em Applesoft, use CLEAR.

CLOSE

Desaloca o buffer usado pelo arquivo em disco especificado, e se a última operação com disco foi de gravação, guarda a informação restante do buffer no disco.

Formato:

CLOSE [*nomearq*]

Deve-se FECHAR *todos* os arquivos onde se gravou qualquer informação por declaração WRITE para que se tenha os dados de forma integral. Se *nomearq* está presente, apenas aquele arquivo é fechado. Por ausência de *nomearq*, fecham-se todos os arquivos (à exceção do arquivo EXEC, que está controlando).

Com DOS 3.2.1 e mais recentes, um arquivo seqüencial irá ocasionalmente preencher um setor quando é fechado. Sob essas condições, um APPEND subseqüente irá apontar o arquivo em seu início ao invés de seu fim. Para contornar isso, chame a sub-rotina em linguagem de máquina da Tabela 8.1 antes do comando CLOSE. Pode-se colocar a sub-rotina em qualquer lugar da memória usando declarações POKE e ela usa cinco bytes (exemplo, posições de 768 a 772, a menos que já esteja usada).

Este é um comando do DOS, e requer PRINT e CTRL-D em modo programado.

COLOR=

Gera cor para gráficos de baixa resolução.

Formato:

COLOR= *exprnm*

Depois da última declaração COLOR, toda declaração PLOT, VLIN e HLIN ficará na cor especificada. Os códigos das cores são listados na Tabela 8.2. O valor de *exprnm* deve estar compreendido entre 0 e 255; valores reais são convertidos para inteiros. Valores maiores que 15 repetem as cores mostradas na Tabela 8.2 (0, 16, 32 etc., dão preto etc.).

Por ausência, fica adotado COLOR=0.

TABELA 8.2. Códigos de Cores em Baixa Resolução.

Código	Cor	Código	Cor	Código	Cor	Código	Cor
0	Preto	4	Verde Escuro	8	Marrom	12	Verde
1	Magenta	5	Cinza	9	Laranja	13	Amarelo
2	Azul Escuro	6	Azul Médio	10	Cinza	14	Água
3	Púrpura	7	Azul Claro	11	Rosa	15	Branco

A declaração COLOR não tem efeito quando se está usando gráfico em alta resolução. Dentro do modo texto, COLOR é um fator na determinação de que caractere é colocado na tela pela declaração PLOT. Para uma descrição detalhada dessa propriedade, veja PLOT.

CON

Este comando do Integer BASIC devolve a execução de um programa na instrução seguinte à parada.

Formato:

CON

O CON opera após a execução de uma parada com CTRL-C, e por vezes após o RESET. Se não houver interrupção do programa, o CON simplesmente trava o sistema. Um programa não pode continuar se foi interrompido por CTRL-C numa declaração de entrada tipo INPUT.

Se uma linha de programa foi inserida ou retirada, ou foi gerada uma mensagem de erro desde que a interrupção ocorreu, o CON poderá rodar normalmente, ou por vezes travar o sistema ou mesmo gerar mensagem de erro.

O CON só pode ser usado em modo imediato.

Para Applesoft, veja CONT.

CONT

Este comando do Applesoft devolve a execução na próxima instrução após a parada.

Formato:

CONT

O CONT opera após a parada do programa com STOP, END ou CTRL-C. Se uma declaração INPUT for interrompida com CTRL-C, o programa não pode ser continuado. Se não houver

interrupção de programa, ou houve colocação ou aumento de uma linha de programa, ou se foi gerada uma mensagem de erro depois do programa ter sido interrompido, o CONT produzirá a mensagem de erro ?CON'T CONTINUE ERROR.

Para Integer BASIC, veja CON.

DATA

Cria uma lista de valores a serem usados pela declaração READ no Applesoft.

Formato:

DATA *cont* [, *const*. . .]

A declaração DATA pode aparecer em qualquer lugar do programa, ela não precisa ser executada para ser acessada por uma declaração READ.

A declaração DATA especifica tanto valores numéricos quanto alfanuméricos (séries). Constantes de séries são geralmente colocadas entre colchetes; esses colchetes não são necessários a menos que a série tenha espaços, vírgulas ou dois pontos. Um sinal de colchete não pode ser usado como *const*; ele deve ser especificado usando a função CHR$(34).

Um ou mais parâmetros *const* podem ser nulos, ou seja, apenas espaços em branco. Uma constante nula é colocada em zero em valores numéricos, e uma série nula (" ") na variável série.

Não será dada mensagem de erro ao se tentar entrar DATA em modo imediato mas, porém os dados não serão lidos dentro do programa com READ.

Não disponível em Integer BASIC.

DEF FN

A declaração DEF FN permite funções de aplicação específica a ser definida e usada dentro de programas em Applesoft.

Formato:

DEF FN*nvar* (*arg*)=*exprnm*

A variável real *nvar* identifica a função, que é chamada posteriormente usando o nome FN*nvar*.

A declaração função é definida por *exprnm*. *arg* é um nome de variável fictício que pode estar (e em geral está presente em *exprnm*. Seu uso na declaração DEF FN não tem efeito em qualquer outra variável com o mesmo nome em qualquer outro lugar do programa.

Quando FN*nvar* é chamada, o valor da variável fictícia é especificado por uma expressão numérica, variável ou constante. Os valores de todas as variáveis em *exprnm* devem ser definidos antes de FN*nvar* ser chamada. Veja também FN na seção de funções deste capítulo.

A declaração DEF FN inteira deve aparecer numa única linha de programa. Entretanto, uma função definida anteriormente pode ser incluída em *exprnm*, de forma que podem ser desen-

volvidas funções da complexidade que se desejar. Entretanto, uma função definida por usuário não pode referenciar a si própria, direta ou indiretamente (chamando uma função que, por sua vez, a chame).

O nome de função *nvar* pode ser usado novamente, e também ser redefinido para outra declaração DEF FN que apareça depois, no mesmo programa.

Não disponível em Integer BASIC.

A declaração de definição DEF FN é ilegal em modo imediato. Entretanto, uma função que tenha sido definida após o último NEW, CLEAR ou LOAD pode ser referenciada em modo imediato.

DEL

Elimina as linhas de programa especificadas.

Formato:

DEL lin_1, lin_2

Toda a linha de programa maior ou igual a lin_1 e menor ou igual a lin_2 é removida do programa presente na memória. Se lin_1 não existe, a eliminação inicia na linha seguinte. Se lin_2 não existe, ela termina na linha anterior.

DEL deve ser seguido por dois números de linha separados por vírgula. Nenhum deles pode ser negativo, e o segundo deve ser maior ou igual ao primeiro. Se os números forem idênticos, uma linha (no máximo) será eliminada.

O DEL só pode ser usado em modo imediato no Integer BASIC.

Se DEL for usado em modo programado (possível apenas em Applesoft), a eliminação ocorre e o programa pára. CONT não continua a execução neste caso.

DELETE

Retira um arquivo do disco.

Formato:

DELETE *nomearq* [, D*n*] [, S*n*] [, V*n*]

O arquivo especificado é removido do disco.

Se ele não existir no drive D*n* do slot S*n*, aparecerá a mensagem de erro FILE NOT FOUND. Se o disco no drive D*n* no slot S*n* não tiver volume V*n*, resultará em erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se forem omitidos, o último drive referenciado será usado. V0 é usado quando V*n* estiver ausente. Também *n* pode ser omitido, sendo então usado D0, S0 e V0.

Este é um comando do DOS, e requer PRINT e CTRL-D em modo programado.

DIM

Reserva espaço na memória para uma matriz ou série.

Devido às grandes diferenças entre o Integer BASIC e o Applesoft, o DIM é estudado abaixo em cada uma das versões do Apple II.

Formato em Integer BASIC:

DIM *var* (*sub*) [, *var* (*sub*) . . .]

Em Integer BASIC só se pode definir matrizes de uma dimensão e variáveis tipo série simples.

Quando se dimensiona uma matriz, é reservado espaço na memória para o número de elementos *mais* 1. Eles são numerados de 0 até *sub*. O elemento 0 de uma matriz é igual à variável simples de mesmo nome (exemplo: A(0) = A).

A declaração DIM fornece o comprimento máximo de variáveis tipo série. Neste caso, *sub* é o comprimento da série.

Todo *sub* subscrito deve estar entre 1 e 255 na declaração DIM. Além disso, o maior dimensionamento permitido está vinculado ao espaço de memória existente.

Ao se referenciar uma matriz usando uma subscrição maior que aquela declarada em DIM, a mensagem *** RANGE ERR ocorre. Ao se tentar usar mais caracteres que o dimensionado em séries aparece a mensagem *** STRING ERR.

O DIM não atribui nenhum valor particular às variáveis da matriz quando é executado. Por isso, deve-se inicializar qualquer matriz (por exemplo, o zero) após dimensioná-la. Variáveis tipo série, por outro lado, sempre têm valor nulo após serem dimensionadas.

Formato em Applesoft:

DIM *var* (*sub* [, *sub* . . .]) [, *var* (*sub* [, *sub* . . .]) . . .]

A declaração DIM identifica matrizes com uma ou mais dimensões, como segue:

var (sub_i)	Matrizes de dimensão simples
var (sub_i,sub_j)	Matrizes de duas dimensões
var (sub_i,sub_j,sub_k . . .)	Matrizes de dimensões múltiplas

O Applesoft permite três tipos de matrizes: inteira, real e série. Cada elemento de uma matriz é de um tipo especificado pelo nome da variável usada para nome da matriz. O número de dimensões da matriz é dado pelo número de subscritos usados na declaração DIM. Quando uma matriz é referenciada, cada subscrito deve estar dentro da faixa de 0 até *sub*, onde *sub* é o subscrito correspondente à mesma variável na declaração DIM.

O número de dimensões em uma matriz é limitado pela quantidade de memória disponível. O número máximo de dimensões de uma matriz pode ser até 88, e isso só é possível quando muitos dos subscritos forem 0. Um DIM com 89 ou mais subscritos, ou um que faça exceder as limitações de memória produzirá a mensagem de erro ?OUT OF MEMORY ERROR.

Ao se tentar usar um elemento de matriz com subscrito fora da faixa ou com número de subscritos errado, a mensagem de erro ?BAD SUBSCRIPT ERROR aparece.

Se um elemento da matriz é referenciado antes que o DIM correspondente seja executado, o Applesoft atribui o valor de 10 para cada subscrito. A partir daí a matriz é tratada como se tivesse sido dimensionada em 10 para cada dimensão.

Uma matriz nunca pode ser dimensionada duas vezes, mesmo que uma delas tenha sido dimensionada por ausência. Ao se tentar dimensionar uma matriz já dimensionada, recebe-se a mensagem de erro ?REDIM'D ARRAY ERROR.

DRAW

Esta declaração do Applesoft desenha uma forma gráfica de alta resolução na tela.

Formato:

DRAW *exprnm* [AT *colh, linh*]

A forma identificada pelo valor inteiro de *exprnm* é desenhada na cor determinada pela declaração HCOLOR executada por último. A escala e rotação da forma deve ser indicada por comandos SCALE e ROT antes de se dar o DRAW.

O desenho inicia no local dado pelo valor inteiro da expressão numérica *colh* e *linh*. Se não for especificada a posição na declaração DRAW, a forma inicia no último ponto indicado pelo último comando DRAW, XDRAW ou HPLOT realizado.

O número da forma especificado (*exprnm*) deve estar entre 0 e o número de formas da tabela (que não pode exceder a 255), inclusive.

Evite usar DRAW se não houver tabela de formas na memória. O sistema pode travar, ou pode-se obter uma forma sem sentido na tela. Se o seu programa se estende até uma área de memória de alta resolução, ele também pode vir a ser destruído.

Não disponível em Integer BASIC.

DSP

Apresenta a variação de valores de variáveis na execução de um programa em Integer BASIC.

Formato:

DSP *var*

O valor da variável *var* e o número de linha corrente são mostrados sempre que o valor dessa variável é alterado. Essa mostragem pode interagir com os dados saídos de seu programa, tornando um ou ambos ininteligíveis. O RUN cancela todos os DSP dados. Use CON ou GOTO ao depurar programas com o DSP em modo imediato.

Para desligar o DSP, use NO DSP.

Não disponível em Applesoft.

END

Causa parada no programa.

Formato:

END

Não é colocada nenhuma mensagem. Em Integer BASIC, o END deve ser a última instrução executada ou a mensagem *** NO END ERR aparece. O END é opcional em Applesoft.

Não pode ser usado em modo imediato no Integer BASIC.

EXEC

Executa um arquivo de texto em disco como se cada caractere fosse digitado no teclado.

Formato:

EXEC *nomearq* [, R*n*] [, D*n*] [, S*n*] [, V*n*]

Um arquivo de texto a ser usado com EXEC deve conter combinação de comandos BASIC, comandos DOS e linhas de programa. Quando EXEC é executado, a primeira linha do arquivo especificado é lida do disco. Se for comando, ele será executado imediatamente; se for linha de programa, irá para a memória, exatamente como se tivesse sido entrado pelo teclado.

Um arquivo EXEC pode ser usado para entrar um programa inteiro, listá-lo, executá-lo, guardá-lo no disco, trazê-lo de volta, e qualquer outra coisa que se pode fazer a partir do teclado. Pode-se até usar um arquivo EXEC para criar e executar outro arquivo EXEC.

O parâmetro R, se presente, especifica qual campo do arquivo será executado primeiro. Quando usado com o EXEC, o parâmetro R sempre conta a partir do início do arquivo: o primeiro campo do arquivo é o campo 0, o segundo é o 1 etc. O número que segue o R deve ser uma constante inteira entre 0 e 32767. Se R*n* especificar o último campo após o fim de arquivo, nada acontece. Se ele especifica dois ou mais campos depois do fim de arquivo, aparece a mensagem END OF DATA.

Se uma declaração INPUT é executada enquanto o arquivo EXEC está aberto, a resposta é tirada do arquivo EXEC.

Quando a última linha de um arquivo foi executada, o arquivo EXEC fecha a si mesmo (veja CLOSE). Quando um comando EXEC é encontrado num arquivo de controle EXEC, o arquivo original é fechado e qualquer outro comando dele é ignorado; o novo arquivo EXEC é aberto e passa a operar normalmente.

Se o arquivo não existe no drive D*n* do slot S*n*, aparece a mensagem FILE NOT FOUND de erro. Se o disco no drive D*n* do slot S*n* não tem volume V*n*, aparece um erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n*, e S*n* são omitidos, o drive referenciado por último é adotado. V0 é usado por ausência de V*n*. Também *n* pode estar ausente, quando se adota D0, S0 e V0.

Este é um comando do DOS, requer PRINT e CTRL-D em modo programado.

FLASH

Esta declaração do Applesoft faz os caraçteres do vídeo piscarem.

Formato:

FLASH

Toda a saída por PRINT subseqüente ficará sendo mostrada alternativamente em branco-sobre-preto e preto-sobre-branco. As mensagens de erro também são dadas nessa forma. Entretanto, os caracteres mostrados na tela devidos a INPUT não são afetados, assim como todos os caracteres anteriores ao FLASH.

O FLASH opera alterando um pouco o código ASCII padrão. Por isso, todos os caracteres mandados para o disco em modo flash serão transferidos com códigos incorretos. Ao serem lidos de volta, aparecerão caracteres errados.

Não disponível em Integer BASIC.

FN

Mostrado na parte de funções neste capítulo. Veja também DEF FN.

FOR

Inicia um *loop* que repete uma série de instruções até que uma variável de incremento automático atinja um determinado valor.

Formato:

FOR *varnm*=*exprnm*$_1$ TO *exprnm*$_2$ [STEP *exprnm*$_3$]

Quando o FOR é executado pela primeira vez, é dado o valor *exprnm*$_1$ à variável *varnm.* As declarações seguintes ao FOR são executadas até que uma declaração NEXT apareça. O valor de *varnm* é então incrementado de *exprnm*$_3$ (ou 1 se STEP não está presente). Em seguida, o novo valor de *varnm* é comparado com o valor de *exprnm*$_3$. Se o sinal é positivo e o novo valor de *varnm* é menor ou igual a *exprnm*$_2$, a execução volta para a declaração seguinte ao FOR. A mesma coisa acontece se o sinal de *exprnm*$_3$ é negativo e o novo valor de *varnm* é maior ou igual a *exprnm*$_2$. Por outro lado, a execução continua com a declaração seguinte ao NEXT se *varnm* é maior que *exprnm*$_2$ (*exprnm*3 positivo) ou menor ou igual a *exprnm*$_2$ (*exprnm*$_3$ negativo). Devido à comparação ocorrer depois de incrementar *varnm*, as instruções entre o FOR e o NEXT são executadas em, pelo menos, uma vez.

Em Integer BASIC, *varnm* deve ser uma variável inteira. Em Applesoft, *varnm* deve ser uma variável real. Ela nunca pode ser do tipo série.

Os valores de início, fim e incremento são determinados por *exprnm*$_1$, *exprnm*$_2$ e *exprnm*$_3$ apenas uma vez, na primeira execução da declaração FOR. Se esses valores são trocados dentro do

loop, eles não afetarão o *loop*. Pode-se trocar o valor de *varnm* dentro do *loop*. Isso pode fazer com que se termine o FOR-NEXT antes que o valor final seja encontrado: coloque em *varnm* o valor final (de $exprnm_2$), e no caso seguinte do *loop* ele será terminado. Não inicie um *loop* dentro de uma sub-rotina finalizando em outra.

Os *loops* FOR-NEXT podem ser aninhados. Cada um dos *loops* no caso devem ter um *varnm* diferente. Cada *loop* aninhado deve estar totalmente contido no outro mais externo, os *loops* não podem terminar no mesmo ponto. O Integer BASIC permite 16 níveis de *loop* para o FOR-NEXT, o Applesoft permite apenas 10.

O FOR pode ser usado em modo imediato apenas em Applesoft. O *loop* inteiro deve ser entrado na mesma linha. Se o NEXT não estiver presente, o *loop* só será executado uma vez.

FP

Coloca o Apple II em Applesoft.

Formato:

FP [, D*n*] [, S*n*] [, V*n*]

A fonte do Applesoft depende de que tipo de Apple II se tem, e de que opcionais estão instalados:

1. Com Apple II Plus a linguagem está em ROM (memória só de leitura), não importa que tipos de opcionais existem.
2. Se tiver o cartão Applesoft firmware instalado, o FP obtém a linguagem a partir dele, não importando as chaves do cartão.
3. Com o Apple II Language System instalado, o FP pega o Applesoft dele.
4. Em qualquer outro Apple II, o FP procura o Applesoft no disco especificado (ou corrente). Se ele não existe lá, a mensagem LANGUAGE NOT AVAILABLE aparece.

O FP zera qualquer programa BASIC presente na memória.

Se o arquivo não existe no drive D*n* do slot S*n*, a mensagem de erro FILE NOT FOUND aparece. Se o disco do drive D*n*, slot S*n* não tem volume V*n*, aparece o erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem aparecer em qualquer ordem. Se D*n* e S*n* são omitidos, o drive referenciado por último é adotado. V0 é usado por ausência de V*n*. Também *n* pode estar ausente, e D0, S0 e V0 são usados.

Uso apenas em modo imediato.

GET

Esta declaração do Applesoft aceita um caractere único do teclado sem mostrar na tela.

Formato:

GET *var*

A execução pára até que uma tecla seja acionada. Quando *var* é uma variável série, o caractere é associado à variável. Se CTRL-@ é entrado, fica atribuída a série nula à variável.

O GET não tem uso freqüente com valores numéricos para variáveis. Quando é usado, permite a entrada de um dígito de 0 a 9, associando o valor à variável. Entrando-se o sinal mais, menos, vírgula, dois pontos, CTRL-E, espaço, E, ou ponto, o valor da variável fica sendo zero. Entrando qualquer caractere diferente desses acima, resulta a mensagem de erro ?SYNTAX ERROR, e o programa pára.

O GET não pode ser usado em modo imediato.

Não disponível em Integer BASIC.

GOSUB

Faz o programa pular para a linha indicada. Quando é executada uma instrução RETURN, o programa pula de volta para a instrução imediatamente seguinte ao GOSUB.

Formato Geral:

GOSUB *lin*

A declaração GOSUB chama uma sub-rotina. O ponto de entrada da sub-rotina deve ser na linha número *lin.* Esse ponto de entrada deve ser o início lógico de uma sub-rotina. Ou seja, uma linha que contenha uma declaração que é executada primeiro. O ponto de entrada da sub-rotina não precisa necessariamente ser o seu menor número de linha.

Após a completa execução da sub-rotina, o programa volta à linha seguinte ao GOSUB. A sub-rotina usa uma declaração RETURN para informar que o programa deve voltar.

Um GOSUB pode ocorrer em qualquer ponto do programa; portanto, uma sub-rotina pode ser chamada de qualquer ponto do programa.

Sub-rotinas podem ser aninhadas; isto é, podem ser chamadas de dentro de outras sub-rotinas. São permitidos vinte e cinco níveis de aninhamento no Applesoft, ou seja, podem ser executadas vinte e quatro declarações GOSUB antes da primeira declaração RETURN. O limite em Integer BASIC são 16 níveis.

Normalmente deve-se sair de uma sub-rotina com uma declaração RETURN, e não GOTO. Porém pode-se usar GOTO para sair da sub-rotina se antes for executada uma declaração POP.

Não pode ser usada em modo imediato no Integer BASIC.

Formato Adicional no Integer BASIC:

GOSUB *exprnm*

Em Integer BASIC, uma expressão numérica pode ser colocada no lugar do número de linha. Se o cálculo de *exprnm* não resultar num número de linha existente, aparece a mensagem de erro

*** BAD BRANCH ERR. Este formato do GOSUB permite simular a instrução ON GOSUB, que não é disponível em Integer BASIC.

GOTO

Salto incondicional do programa para a linha indicada.

Formato Geral:

GOTO *lin*

A execução do programa continua a partir da linha indicada. Se ela não existe, aparece a mensagem ?UNDEF'D STATEMENT ERROR no Applesoft, e a mensagem *** BAD BRANCH ERR no Integer BASIC.

Formato Adicional no Integer BASIC:

GOTO *exprnm*

Em Integer BASIC, é permitido colocar uma expressão numérica no lugar do número de linhas. Se o número de linhas calculado não existir, a mensagem *** BAD BRANCH ERR aparece. Essa forma do GOTO permite simular a declaração ON GOTO, não disponível em Integer BASIC.

GR

Converte a tela para gráfico de baixa resolução (40 X 40), deixando quatro linhas para texto na parte baixa da tela.

Formato:

GR

A porção gráfica da tela é colocada em preto, o cursor é posicionado na janela de texto, e COLOR é posto em 0 (preto).

Se executado enquanto HGR está em atividade, o GR se comporta normalmente. Entretanto, se HGR2 estiver em processo, o sistema dará acesso à página 2 de texto e gráficos em baixa resolução. Isso pode ser confuso, porque a tela ficará cheia de bobagem e nada que for digitado aparecerá na tela. Para voltar ao modo normal, digite TEXT. Use sempre TEXT nos programas, ao passar de HGR2 para GR.

Pode-se pular para a forma de tela cheia (40 X 48), em gráfico de baixa resolução, com a declaração POKE −16302,0 após executar GR. Qualquer coisa digitada após isso no modo imediato será colocada na tela como pontos coloridos nas quatro últimas linhas da tela, porém a execução será normal. POKE −16302,0, devolve a janela de texto.

HCOLOR=

Esta declaração do Applesoft fornece a cor para desenho em modo gráfico em alta resolução.

Formato:

HCOLOR= *exprnm*

Até a declaração HCOLOR seguinte, todas as declarações HPLOT e DRAW serão executadas na cor especificada. Os códigos de cores são listados na Tabela 8.3. O valor de *exprnm* deve estar na faixa entre 0 e 7. Valores fora dessa faixa produzem uma mensagem ?ILLEGAL QUANTITY ERROR. Um desenho executado antes da primeira determinação HCOLOR será feito numa cor indefinida.

O HCOLOR não afeta gráficos em baixa resolução. Uma declaração HCOLOR executada quando o Apple II não está em modo gráfico de alta resolução não afeta a cor do desenho seguinte em alta resolução.

Não disponível em Integer BASIC.

HGR

Esta declaração do Applesoft converte a tela em modo gráfico de alta resolução (260 X 160), com uma janela de texto de quatro linhas na parte de baixo da tela.

Formato:

HGR

É mostrada a página 1 de alta resolução. A memória da tela de baixa resolução (texto) não é afetada, porém apenas as quatro últimas linhas ficam visíveis. O cursor não se move para essa janela. Não se pode ver o cursor até que se tenha digitado várias linhas após executar o HGR. A porção gráfica da tela é colocada em preto. HCOLOR não é afetado com este comando.

TABELA 8.3. Códigos de Cores em Alta Resolução.

Código	Cor	Código	Cor
0	Preto	4	Preto
1	Verde	5	Laranja*
2	Violeta*	6	Azul*
3	Branco	7	Branco

* Depende dos controles da TV.

Pode-se ter toda a tela (280 X 192) em forma de alta resolução com a declaração POKE 16302,0, após executar HGR. Qualquer comando dado no modo imediato depois disso não será visto na tela, porém será executado de acordo. POKE −16301,0 devolve a janela de texto.

Em sistemas do Apple II com menos de 32K bytes de memória, não se pode usar HGR e o DOS ao mesmo tempo desde que eles ocuparão necessariamente a mesma área de memória.

Além disso, o interpretador Applesoft em disco ou cassette ocupa parte da página gráfica 1 de alta resolução. Por isso não se pode usar HGR com Applesoft em disco ou cassette.

Mesmo com Applesoft em firmware, se o programa é muito longo, ele pode entrar na área de página gráfica de alta resolução 1. Pode-se precaver sobre isso com o comando HIMEM: 8192 que mantém o programa fora da área gráfica 1.

Não disponível em Integer BASIC.

HGR2

Converte a tela inteira para modo gráfico de alta resolução (280 X 192). É mostrada a página 2 de gráfico em alta resolução.

Formato:

HGR2

A memória de tela em baixa resolução (texto) não é afetada. Entretanto não se consegue ver o que é digitado, apesar de ser executado normalmente. A tela colocada em preto. O HCOLOR não é afetado por este comando.

A página gráfica 2 não é disponível se o sistema tem menos de 24K de memória. Em sistemas com 24K, coloque HIMEM: em 16384 antes de usar HGR2 para proteger o programa e as variáveis dos gráficos, e vice-versa.

Não se pode usar HGR2 e o DOS de forma simultânea a menos que o sistema tenha 36K de memória ou mais. Isso quer dizer que são necessários pelo menos 36K para ter o HGR2 junto com o Applesoft em disco.

Não tente gerar uma janela de texto com POKE −16301,0. Isso vai mostrar a página gráfica 2 de baixa resolução enquanto os comandos digitados vão para a página 1 tornando-se invisíveis (embora sejam executados corretamente).

Não disponível em Integer BASIC.

HIMEM:

Gera o limite superior de memória disponível para programa em BASIC, incluindo área de variáveis.

Formato:

HIMEM: *exprnm*

HIMEM: determina a posição mais alta de memória de leitura e gravação (RAM) disponível para o programa BASIC. O sistema operacional em disco (DOS) sempre reside acima do HIMEM: se está

presente. Com a declaração HIMEM: pode-se reservar espaço para sub-rotinas em linguagem de máquina e tabelas de forma gráfica de alta resolução. Também se pode proteger a área de RAM reservada para tela gráfica de alta resolução.

HIMEM: primeiramente é preenchido com a posição de memória mais alta do Apple II (por exemplo, 49151 em sistemas de 48K). O DOS reside na parte mais alta da memória, de forma que ele ajusta o HIMEM: aproximadamente 10.800 bytes abaixo quando é carregado. Cada arquivo de buffer adicional que é carregado via MAXFILES abaixa o HIMEM: outros 595 bytes. Se o programa em Applesoft usa séries, seus valores são guardados iniciando na posição resultante de HIMEM:, trabalhando de cima para baixo. Consulte o mapa de memória no Apêndice G.

O valor de *exprnm* deve estar na faixa entre −65535 até 65535 (−32767 até 32767 em Integer BASIC) ou ocorre uma mensagem de erro.

Não se deve colocar o HIMEM: indicando posição acima da maior posição de memória disponível, porque nesse caso se corre o risco de ter armazenamento de variáveis em posições de memória não existente.

Pode-se ver o valor corrente de HIMEM: usando as instruções apropriadas, como listado abaixo:

```
PRINT PEEK(116) * 256 + PEEK(115)      para Applesoft
PRINT PEEK(77) * 256 X PEEK(76)        para Integer BASIC
```

HIMEM: não é afetado por NEW, RUN ou CLEAR.

Se for dado HIMEM: menor que LOMEM: ou não se deixar espaço suficiente para o programa, ocorre uma mensagem de erro.

Só pode ser usado em modo imediato no Integer BASIC.

HLIN

Desenha uma linha horizontal na tela dentro do modo gráfico em baixa resolução.

Formato:

HLIN col_1, col_2 AT *lin*

A linha é desenhada de col_1 até col_2 na linha especificada. A cor é determinada pela última declaração COLOR executada. Se a tela estiver em modo texto, ou se a janela de texto estiver presente, e *lin* for maior que 39, HLIN desenhará uma linha de caracteres na tela, no lugar referente à janela de texto, onde os pontos gráficos deveriam ser colocados. Os caracteres usados são determinados pela última declaração COLOR executada; veja a Tabela 8.5 (perto da declaração PLOT neste capítulo) para detalhes.

Em Integer BASIC, col_1 deve ser menor ou igual a col_2 ou a mensagem *** RANGE ERR aparecerá.

HOME

Esta declaração do Applesoft limpa a tela e posiciona o cursor no canto superior esquerdo da janela de texto.

Formato:

HOME

Em Integer BASIC, use CALL-936.

HPLOT

Esta declaração do Applesoft coloca um ponto ou desenha uma linha colorida na tela gráfica de alta resolução.

Formato:

HPLOT *colh, linh*

HPLO TO *colh, linh*

HPLOT $colh_1$, $linh_1$ TO $colh_2$, $linh_2$ [TO $colh_3$, $linh_3$ . . .]

A primeira forma do comando coloca um ponto colorido na tela e na posição especificada. A cor do ponto é determinada pela última declaração COLOR executada.

A segunda forma do comando desenha uma linha colorida a partir do último ponto colocado até as coordenadas *colh* e *linh.* Se não houve ainda nenhum desenho desde que foi dado o comando HGR ou HGR2, nada será desenhado. A cor da linha é dada pela última declaração HCOLOR executada.

A terceira forma do comando também desenha uma linha colorida. Com o Applesoft em firmware, a linha pode ter mais de um segmento. Primeiro é desenhado o segmento de $colh_1$ e $linh_1$ até col_2 e $linh_3$. A cor da linha (todos os segmentos) é dada pela última declaração COLOR executada.

Não são permitidas coordenadas adicionais em Applesoft em disco ou cassette. Quando presente em um programa Applesoft em firmware, o segundo segmento é desenhado a partir de col_2 e $linh_2$ até $colh_3$ e $linh_3$ etc. Pode haver qualquer número de pares de coordenadas desde que eles caibam em uma linha de programa.

Qualquer porção de uma linha ou pontos que caírem na janela de texto não serão visíveis. Entretanto, se for mudado para formato de gráfico total como o comando POKE −16302,0, qualquer linha ou ponto desenhado na janela de texto se tornará visível.

Deve-se sempre executar uma declaração HGR ou HGR2 antes de um HPLOT. De outra forma, pode-se destruir programas ou variáveis.

Não disponível em Integer BASIC.

HTAB

Esta declaração em Applesoft posiciona o cursor numa coluna especificada na linha corrente.

Formato:

HTAB *col*

O cursor se move para a direita ou esquerda para a coluna especificada pelo valor de *col*, sem limpar qualquer caractere na tela. As colunas são numeradas de 1 a 40 (esquerda para a direita).

Em Integer BASIC, usa a declaração TAB.

IF-THEN

Execução condicional de instrução ou instruções pelo programa. As regras para Integer BASIC e Applesoft são vistas em separado.

Formato em Integer BASIC:

IF *expressão* THEN *declaração*

IF *expressão* THEN [GOTO] *linha*

Na primeira forma da declaração IF-THEN, a expressão especifica uma condição que, se verdadeira, causa a execução da declaração seguinte ao THEN. Se a condição é falsa, é executada a declaração em seguida ao IF-THEN; a *declaração* seguinte ao THEN não é executada neste caso.

No segundo formato do IF-THEN (formato de salto condicional), a expressão especifica uma condição que, sendo verdadeira, causa o salto do programa para o número de linha indicado.

As expressões relacionais são as mais freqüentemente usadas em IF-THEN. Valores tipo série só podem ser comparados como iguais ou diferentes em Integer BASIC.

expr também pode ser uma expressão numérica. Nesse caso, *expr* é considerada verdadeira se for um valor diferente de zero.

A expressão IF-THEN não pode ser uma expressão série (ou seja, qualquer coisa que calcule uma variável série) em Integer BASIC.

Formato do Applesoft:

IF *expr* THEN *declaração* [: *declaração* . . .]

IF *expr* { THEN / GOTO / THEN GOTO } *linha*

No primeiro formato da declaração IF-THEN, a expressão especifica uma condição que, se verdadeira, causa toda a *declaração* que segue THEN na mesma linha de programa a ser executada. Se

a condição é falsa, o controle passa para a primeira *declaração* da linha de programa que segue e nenhuma declaração depois do THEN é executada.

No segundo formato (formato de salto condicional), o programa pula para a linha número *linha* se a condição é verdadeira. De outra forma a execução continua com a primeira declaração da primeira linha de programa que segue o IF-THEN.

Se um salto incondicional for uma das várias declarações que seguem o THEN, então ele deve ser a última declaração da linha, e deve ter o formato GOTO *linha.* Se um salto incondicional não for a última declaração da linha, as declarações que o seguem nunca serão executadas.

O tipo mais comum de expressão usado em IF-THEN é o relacional. Se expressões tipo série são comparadas usando operadores relacionais, os códigos ASCII (listados no Apêndice I) dos caracteres envolvidos determinam os valores relativos das séries. São comparados caractere por caractere até que uma diferença ocorra. Então a série com código ASCII maior naquela posição será considerada maior. Se não ocorrer diferença, a série mais longa será a maior.

A expressão também pode ser numérica. Se o valor da expressão não é zero, a condição é considerada verdadeira. Se o valor da expressão é zero (falso), a execução continua a partir da primeira linha de programa a seguir.

Em Applesoft, *expr* também pode ser uma expressão tipo série. Entretanto, a execução de duas ou três declarações no decorrer de um programa gera a mensagem ?FORMULA TOO COMPLEX ERROR.

O Applesoft tem problemas se o último caractere não-espaço que precede o THEN é a letra A. O A combinado com o T forma a palavra reservada AT. Para evitar esse problema pode-se fechar com parênteses toda a expressão, incluindo o A.

Se um loop FOR-NEXT vier em seguida ao THEN, ele deve estar inteiramente contido dentro da linha IF-THEN. Outros IF-THEN podem vir a seguir ao THEN, desde que estejam completamente contidos dentro da linha IF-THEN original. Entretanto, uma expressão Booleana ganha de IF-THEN aninhados no aspecto de clareza. Por exemplo, as duas declarações abaixo são equivalentes, mas a segunda é mais fácil de ser entendida.

```
10 IF A$ = "X" THEN IF B = 2 THEN IF C > D THEN 50
10 IF A$ = "X" AND B = 2 AND C > D THEN 50
```

IN#

Selecione o slot periférico de onde as entradas subseqüentes serão aceitas.

Formato:

IN# *slot*

Todas as declarações que seguirem procurarão dados no periférico do slot determinado. *slot* deve ser um valor inteiro entre 0 e 7. Note que slot 0 não é um dispositivo periférico; IN#0 especifica o teclado como dispositivo de entrada. Se não existir periférico acoplado ao slot especificado, o sistema ficará bloqueado até o acionamento de RESET.

Sempre que o DOS estiver presente na memória do Apple II, IN# será considerado um comando do DOS, requerendo PRINT e CTRL-D em modo programado.

INIT

Inicializa um disco.

Formato:

INIT *nomearq* [, D*n*] [, S*n*] [, V*n*]

O programa presente na memória é guardado no disco sob o nome de arquivo dado. Esse programa se torna o de saudação, e roda automaticamente sempre que o disco é carregado. Ao disco é associado o número de volume com que ele é inicializado; se não é informado nenhum número de volume, associa-se ao disco o número de volume 254.

Se o arquivo não existe no drive D*n* do slot S*n*, aparece a mensagem de erro FILE NOT FOUND.

D*n*, S*n* e V*n* podem ser dados em qualquer ordem. Se D*n* e S*n* são omitidos, o drive acessado por último é usado. Também *n* pode ser ausente; D0 ou S0 são usados.

O INIT só pode ser usado em modo imediato.

INPUT

Aceita caracteres entrados a partir do teclado ou outro dispositivo de entrada, calcula e associa o valor ou valores com a variável ou variáveis.

Formato em Integer BASIC:

INPUT [*"requisição"*,] *var* [, *var* . . .]

O INPUT no Integer BASIC aceita valores em qualquer combinação entre variáveis inteiras e séries. Se a primeira variável é um inteiro, então aparece um ponto de interrogação na posição atual do cursor como uma indicação para entrada. O Integer BASIC não coloca interrogação se a primeira variável a ser entrada é uma série.

A *requisição* opcional é uma constante tipo série. Se presente, é mostrada antes da primeira variável a ser entrada; ela não se repete a cada variável da lista. Aparece um ponto de interrogação à frente da *requisição* se a primeira variável da lista é um inteiro. Só é mostrada a *requisição* se a primeira variável a entrar é uma série. Veja que a *requisição* é seguida de vírgula na declaração INPUT. A *requisição* não pode ser uma variável série ou uma expressão tipo série.

Quando uma única declaração INPUT chama mais de uma variável de forma sucessiva, pode-se entrar cada uma delas numa linha separada; separando cada valor com a tecla RETURN. O Integer BASIC mostra um ponto de interrogação (?) a cada linha, pedindo outra entrada. Opcionalmente, pode-se entrar mais de um valor inteiro na mesma linha; separando os valores com vírgula.

Entradas numéricas devem conter apenas caracteres válidos como numéricos. Os dígitos de 0 a 9, espaço e sinais de mais e menos. Obtém-se uma mensagem de erro ao se acionar RETURN sem entrada de nenhum valor numérico.

Deve-se digitar cada variável tipo série numa linha separada. Todos os caracteres (exceto

CTRL-C, CTRL-M, CTRL-H, CTRL-U e CTRL-X) digitados antes de se acionar RETURN são aceitos e associados à variável. A série nula (" ") é associada à variável no caso de acionamento. simples da tecla RETURN quando uma variável cordão é esperada. Se forem digitados caracteres inaceitáveis (ou seja, letras em valores numéricos) a mensagem de advertência *** SYNTAX ERR e RETYPE LINE aparece. Deve-se reentrar todos os valores digitados na linha não aceita.

Não pode ser usado em modo imediato.

Formato no Applesoft:

INPUT [*requisição;*] *var* [, *var* . . .]

O INPUT pode requisitar qualquer combinação de valores numéricos e séries. Um ponto de interrogação é normalmente mostrado como indicação de início de digitação, no ponto onde está o cursor. O Applesoft suspende a interrogação se a *requisição* opcional estiver presente.

A *requisição* é uma constante tipo série. Se presente, ela é mostrada antes da variável ser digitada, não sendo repetida para as digitações das variáveis da lista. Não é colocado o ponto de interrogação após a *requisição.* Note como a *requisição* é seguida de ponto-e-vírgula na declaração INPUT.

De forma geral, quando uma declaração INPUT simples chama mais de um valor, pode-se entrar cada um em uma linha separada; terminando cada digitação com a tecla RETURN. O Applesoft mostra interrogação dupla (??) para cada valor requerido. Opcionalmente, pode-se entrar mais de um valor numa única linha; separando-os com vírgula.

Se forem digitados caracteres inaceitáveis (exemplo: letras em valores numéricos) aparece uma mensagem de advertência e deve-se redigitar toda a linha. O Applesoft manda a mensagem REENTER e reexecuta a declaração INPUT a partir de seu início. A interrogação ou a requisição é impressa e deve-se redigitar todos os valores do INPUT.

Entradas numéricas devem constituir apenas em caracteres numéricos. Se apenas for acionada a tecla RETURN, quando é esperado um valor numérico, aparece a mensagem de erro e a linha deve ser redigitada. Os dígitos de 0 a 9, espaço e os sinais de mais e menos são aceitos como valores numéricos. O Applesoft também aceita o ponto decimal, um sinal adicional de mais ou menos e a letra E para a entrada de números reais em notação científica.

Em Applesoft, se o primeiro caractere diferente de espaço numa entrada de série é aspas, todos os caracteres (incluindo vírgulas e dois pontos) até as próximas aspas ou RETURN são atribuídos à variável. Se a entrada não inicia com aspas, todos os caracteres (incluindo aspas) até a próxima vírgula, dois pontos ou RETURN são atribuídos à variável. Se duas ou mais séries são requisitadas pela declaração INPUT, elas devem ser separadas por vírgulas.

Se for acionada apenas a tecla RETURN quando é esperada uma série, a série nula (" ") é associada à variável.

Em Applesoft, todos os caracteres após dois pontos em INPUT são ignorados a menos que a digitação inicie com aspas.

INPUT não pode ser usado em modo imediato.

INT

Coloca o Apple II em Integer BASIC.

Formato:

INT

Qualquer programa presente na memória é zerado. Se o Integer BASIC não está presente (por exemplo, num Apple II Plus sem o Language Card), a mensagem LANGUAGE NOT AVAILABLE aparece.

Usado apenas em modo imediato.

INVERSE

Esta declaração do Applesoft liga o modo vídeo inverso (também chamado vídeo reverso).

Formato:

INVERSE

Todas as saídas subseqüentes, dadas com declaração PRINT aparecerão em caracteres preto-no-branco. As mensagens de erro também são afetadas. Entretanto, os caracteres mostrados na tela devido a declarações INPUT não são afetados, bem como os caracteres mostrados anteriormente.

O INVERSE opera alterando um pouco o código ASCII. Assim, qualquer caractere mandado para o disco em vídeo reverso será guardado em códigos incorretos. Ao serem lidos de volta, aparecerão caracteres errados.

Não disponível em Integer BASIC.

LET=

Uma declaração de atribuição, LET=, ou simplesmente =, atribui um valor a uma determinada variável.

Formato:

[LET] *var* = *expr*

A variável *var* passa a conter o valor obtido pelo cálculo de *expr.*

LIST

Mostra todo ou uma parte do programa corrente na memória. Existem dois formatos para o comando LIST. Um é reconhecido por Integer BASIC e Applesoft e o outro apenas pelo Applesoft. Eles são descritos em separado a seguir.

Formato Geral:

LIST *lin*$_1$ [, *lin*$_2$]

Qualquer parte do programa pode ser listada. Se não for colocada nenhuma linha a seguir dó LIST, todo o programa é listado. Se apenas *lin*$_1$ está presente, apenas aquela linha é listada, se existir. Se ambas as linhas estiverem presentes, o programa será listado a partir da *lin*$_1$ e continuará até a *lin*$_2$.

Se a *lin*$_1$ não existir, a listagem iniciará na próxima linha de número mais alto. Se *lin*$_2$ não existir, a listagem terminará na última linha de número mais baixo.

LIST não pode ser usado com variáveis ou expressões no lugar de números de linhas.

Quando o LIST mostra o programa, ele deixa espaço extra para variáveis e palavrás reservadas para tornar a listagem mais legível. Se isso causar problemas, pode-se eliminar esse efeito reduzindo a janela de texto para o tamanho 33 (ou menos) com o comando POKE 33,33. (POKE 33,40 ou TEXT devolve o tamanho normal para a janela de texto.)

O comprimento das linhas de programa é limitado, porém é calculado antes que o LIST forneça espaços extra. Dessa forma pode-se estender o comprimento aparente das linhas de programa deixando espaços ao se digitar as linhas. A desvantagem é que a linha será muito longa para copiar e editar após ser listada com muitos espaços incorporados.

Formato (expandido) do Applesoft:

$$\text{LIST} \begin{cases} \textit{lin}_1 \, [\{-,\}] \\ \textit{lin}_1 \,] \{-,\} \, \textit{lin}_2 \end{cases}$$

Em Applesoft, tanto a vírgula (,), quanto o hífen (-) podem separar dois números de linha.

Em Applesoft também se pode listar a partir do início do programa até uma determinada linha colocando uma vírgula ou hífen à frente da *lin*$_2$ (omitindo *lin*$_1$). Também se pode listar a partir de um número de linha específico até o fim do programa colocando a vírgula ou hífen após *lin*$_1$ (e omitindo *lin*$_2$).

LOAD

Carrega um, é de cassette ou disco.

Formato em Cassette:

LOAD

Carrega o programa seqüencial seguinte a partir do cassette, colocando no lugar do programa corrente na memória. Deve-se ter o gravador cassette rodando no modo PLAY quando o LOAD é executado; o Apple II não dá nenhuma informação sobre isso. Ele dá um *bip* quando inicia o carregamento e outro quando termina. O segundo *bip* informa que se deve desligar o gravador.

Só pode ser usado em modo imediato no Integer BASIC.

Formato em Disco:

LOAD *nomearq* [, D*n*] [, S*n*] [, V*n*]

O programa de nome *nomearq* é carregado a partir do disco. Se a operação for bem-sucedida, o programa anterior da memória será apagado.

Se o programa a ser carregado for em Applesoft e o Apple II estiver em Integer BASIC, ou vice-versa, ele troca as chaves para a linguagem apropriada. Isso pode requerer o carregamento da linguagem a partir de um disco específico. Se a linguagem não estiver disponível, a mensagem LANGUAGE NOT AVAILABLER aparece.

Se o arquivo não existir no drive D*n* do slot S*n*, a mensagem FILE NOT FOUND aparece. Se o disco no drive D*n* e slot S*n* não tiver o volume V*n*, o erro VOLUME MISMATCH é apresentado.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* e S*n* são omitidos, o último drive ou slot referenciado é usado. V0 será adotado quando V*n* estiver ausente. Também *n* pode estar ausente, sendo adotado D0, S0 e V0.

Este é um comando DOS, requerendo PRINT e CTRL-D em modo programado.

LOCK

Protege um arquivo em disco de eliminação acidental.

Formato:

LOCK *nomearq* [, D*n*] [, S*n*] [, V*n*]

Uma vez protegido, um arquivo não pode ser eliminado ou ter seu nome trocado até que seja desprotegido (veja UNLOCK). Nenhum programa pode ser guardado com o nome igual ao de um programa protegido. Um arquivo protegido é indicado no catálogo do disco por um asterisco à esquerda do tipo do arquivo.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* e S*n* são omitidos, o último drive ou slot referenciado é usado. V0 é adotado quando V*n* estiver ausente. Também *n* pode estar ausente, sendo adotado, D0, S0 e V0.

Este é um comando DOS, requerendo PRINT e CTRL-D em modo programado.

LOMEM:

Determina o limite inferior de memória disponível para variáveis de programas BASIC.

Formato:

LOMEM: *exprnm*

LOMEM: estabelece a menor posição de memória RAM disponível para as linhas de programa e variáveis no BASIC. O Monitor e o interpretador BASIC usam memória abaixo de LOMEM:

para apontadores, gráficos em baixa resolução e memória de texto da tela etc. Quando o interpretador Applesoft é carregado do disco ou cassette, ele sempre reside em RAM abaixo de LOMEM:. Pode-se adicionar ainda espaço para sub-rotinas em linguagem de máquina e tabelas de formas gráficas de alta resolução com a declaração LOMEM:.

Com Integer BASIC em Applesoft em firmware, LOMEM: é posicionado em 2048, imediatamente acima da área usada pelo sistema. Carregando o interpretador Applesoft do disco ou cassette faz-se o LOMEM: subir para 12291. A cada vez que se adiciona uma linha de programa ou se modifica uma linha existente, o LOMEM: é ajustado para cima ou para baixo. Limpando-se um programa em Applesoft (com NEW ou CTRL-B) também se altera LOMEM:. Assim, se se deseja reservar espaço abaixo do programa, deve-se fazer isso após limpar qualquer programa anterior e antes de carregar ou digitar o novo.

O valor de *exprnm* deve estar entre −65535 e 65535 (−32767 até 32767 em Integer BASIC), ou um erro ocorrerá.

Pode-se ver o valor corrente de LOMEM: com a instrução PRINT PEEK(106) * 256 + + PEEK(105).

Em Applesoft, se LOMEM: é colocado com um valor maior que HIMEM: menor que o valor já existente de LOMEM:, ou menor que a posição mais alta de memória usada para o sistema operacional corrente ou programa, a mensagem ?OUT OF MEMORY ERROR.

Só pode ser usado em modo imediato em Integer BASIC.

MAN

Termina numeração automática de linhas em Integer BASIC.

Formato:

MAN

A numeração automática de linhas inicia com AUTO.

Digite CTRL-X para parar temporariamente a geração de números de linhas, digitando então MAN.

Não disponível em Applesoft.

MAXFILES

Especifica o número máximo de arquivos em disco que podem ser ativados a qualquer tempo.

Formato:

MAXFILES *limite*

O sistema operacional (DOS) suporta um máximo de 16 arquivos abertos ao mesmo tempo. Quando executado, o MAXFILES reserva 595 bytes de memória (um *buffer* de arquivo) para cada arquivo. MAXFILES é colocado em 3 automaticamente ao se carregar o DOS.

Todos os comandos do DOS exceto o MAXFILES usam o *buffer* de arquivo quando são executados. Assim, o número máximo de arquivos que se pode ter aberto na prática é um a menos do limite de MAXFILES. Ao se tentar executar comandos DOS quando não há *buffers* disponíveis, a mensagem NO BUFFER AVAILABLE é gerada.

MAXFILES opera com o HIMEM: quando executado, o que pode limpar parte de programas, armazenamento de variáveis etc. Se possível, execute MAXFILES antes de carregar e executar programas.

Usando MAXFILES dentro de um programa Applesoft, use-o na primeira linha. Para usar MAXFILES em um programa Integer BASIC, deve-se criar um arquivo EXEC como visto no Capítulo 5 (veja também EXEC neste capítulo).

limite deve ser uma constante na faixa de 1 a 16 ou um erro RANGE ERROR aparece.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

MON

Causa o aparecimento, na tela, dos comandos de disco e/ou fluxo de dados.

Formato:

MON [C] [, I] [, O]

Os três parâmetros dizem o que será mostrado. Se é especificado C, todos os comandos do disco são mostrados na tela. Se é especificado I, todos os dados que o Apple II recebe vindos do disco são mostrados. Se é especificado O, toda a saída de dados do Apple II para o disco é mostrada na tela. Estes parâmetros podem ser usados em qualquer ordem. Se nenhum deles estiver presente, MON não terá efeito. O efeito de MON permanece até que seja executada uma declaração NOMON, FP ou INT, ou o sistema seja recarregado, ou em algumas máquinas, seja dado RESET.

Este é um comando DOS, e requer PRINT e CTRL-D quando usado em modo programado.

NEW

Elimina o programa corrente e todas as variáveis a ele associadas da memória.

Formato:

NEW

O NEW também reajusta o LOMEM:, porém HIMEM: COLOR e HCOLOR não são afetados. O NEW só pode ser usado no modo imediato em Integer BASIC.

NEXT

Termina o *loop* iniciado com a instrução FOR.

Formato Geral:

NEXT *varnm* [, *varnm*]

Quando NEXT é executado, a variável do *loop varnm* é incrementada do valor especificado na declaração FOR correspondente. O programa então, ou continua com a instrução que segue o FOR ou volta ao FOR correspondente, dependendo do estado dos parâmetros da declaração FOR. Veja o estudo da declaração FOR anteriormente elaborado, neste capítulo.

Se não houver nenhum FOR em processo que use a variável *varnm*, é acusado um erro. E mostrado ?NEXT WHITHOUT FOR ERROR no Applesoft, e *** BAD NEXT ERR no Integer BASIC.

Variáveis múltiplas seguintes ao NEXT devem ser listadas na ordem apropriada (o último *loop* iniciado deve ser terminado primeiro) ou um erro acontecerá.

O NEXT não pode ser usado em modo imediato no Integer BASIC. Em Applesoft, um NEXT em modo imediato causa salto para a declaração FOR que estava sendo executada no modo programado que estava ativo.

Formato Applesoft Adicional:

NEXT

Em Applesoft, pode-se usar NEXT sem identificação do nome da variável. A variável adotada por ausência é aquela do FOR que iniciou mais recentemente e ainda está em efeito. NEXT sem variável é executado mais rapidamente que o correspondente com variável.

NO DSP

Cancela o modo display para a variável especificada em Integer BASIC. (Veja DSP.)

Formato:

NO DSP *var*

Não disponível em Applesoft.

NOMOM

Finaliza a apresentação de comandos de disco e fluxo de dados iniciada com MON.

Formato:

NOMOM [C] [, I] [, O]

Cada parâmetro especificado cancela parte da apresentação iniciada com MON. Se C é especificado, os comandos do disco não são mostrados. Se I é especificado, os dados entrados do disco

para o Apple II não são mostrados. Se O é especificado, os dados saídos do Apple II para o disco não são mostrados. Esses parâmetros podem ser usados em qualquer combinação e em qualquer ordem. Se MON não estiver em efeito para o parâmetro ou parâmetros especificados, ou nenhum parâmetro for especificado, o NOMOM não terá efeito.

Este é um comando DOS, e requer PRINT e CTRL-D quando em modo programado.

NORMAL

Esta declaração do Applesoft desliga os modos de vídeo FLASH e INVERSE.

Formato:

NORMAL

Veja FLASH e INVERSE.

Não disponível em Integer BASIC.

NO TRACE

Cessa o acompanhamento da execução do programa iniciado com TRACE.

Formato:

NO TRACE

Se TRACE não estiver em efeito, NO TRACE também não terá efeito.

ONNER GOTO

Pula para a linha de número indicado quando acontecer erro num programa Applesoft.

Formato:

ONNER GOTO *linha*

Este comando liga um indicador que causa o salto do programa para a linha de número indicado quando um erro é detectado. ONNER GOTO deve ser dado antes que um erro ocorra.

Cada tipo de erro tem um código associado. O código do erro mais recente detectado fica guardado na posição de memória 222. PEEK(222) pega o código de erro. Os códigos de erro e suas mensagens são dados na Tabela C.1 (Apêndice C). Quando um erro ocorre dentro de um *loop* FOR-NEXT ou numa sub-rotina, os apontadores e as pilhas são alterados. A rotina de erro deve voltar para o início da declaração FOR ou GOSUB, reiniciando o *loop* ou a sub-rotina. Se a rotina de erro voltar para o NEXT ou o RETURN, outro erro poderá ocorrer.

ONNER GOTO não funciona corretamente em algumas circunstâncias. O Apple II trava se existem dois erros GET numa coluna e a rotina de manipulação de erros termina com RESUME, e não GOTO. Em programas que usam declarações PRINT (ou se o TRACE está em efeito), o 43º erro não vindo de uma declaração INPUT causa um salto para o MONITOR. Nesta situação, se o GOTO termina uma sub-rotina de manipulação de erro (em lugar de RESUME), o 87º erro de INPUT causa um salto para o Monitor.

Pode-se contornar todos esses problemas descritos se a rotina de tratamento de erro incluir uma chamada para o programa em linguagem de máquina descrito na Tabela 8.4.

ON-GOSUB

Possibilita chamada opcional para uma das várias sub-rotinas num programa em Applesoft, dependendo do valor atual de uma expressão.

Formato:

ON *exprnm* GOSUB *linha* [, *linha* . . .]

O programa salta para a primeira linha se o valor inteiro da expressão é 1, para o segundo se é 2 etc. O próximo RETURN encontrado faz o programa voltar para a declaração seguinte ao ON-GOSUB.

A expressão deve ter um valor na faixa entre 0 e 255 ou então ocorrerá a mensagem ?ILLEGAL QUANTITY ERROR.

TABELA 8.4. Programa em Linguagem de Máquina para ONNER GOTO.

LINGUAGEM DE MÁQUINA		LINGUAGEM ASSEMBLER DO 6502	
Decimal	**Hexadecimal**	**Instrução**	**Comentário**
104	68	PLA	Coloca o maior byte do stack no Acumulador
168	A8	TAY	Salva-o no Registrador Índice Y
104	68	PLA	Põe o próximo byte no Acumulador
166	A6	LDX $DF	Usa apontador do ONNER GOTO
223			
154	9A	TXS	como endereço de pilha
72	48	PHA	Joga o conteúdo salvo da pilha na
152	98	TYA	pilha "ONNER GOTO" (dois
72	48	PHA	bytes do Acumulador e do registrador Y)
96	60	RTS	Retorna ao Applesoft

ERROR ocorre. Se o cálculo da expressão der zero ou um valor maior que o número de linhas listado, a execução do programa continua a partir da linha seguinte ao ON-GOSUB, não disponível em Integer BASIC. (Mas veja o GOSUB, e a forma do GOSUB calculado para o Integer BASIC.)

ON-GOTO

Provoca um salto condicional para um dos vários pontos do programa Applesoft, dependendo do valor corrente de uma expressão.

Formato:

ON *exprnm* GOTO *linha* [, *linha* . . .]

O programa salta para o primeiro número de linha se o valor inteiro do cálculo da expressão é 1, para o segundo se é 2 etc.

A expressão deve ter um valor numérico entre 0 e 255, ou a mensagem ?ILLEGAL QUANTITY ERROR aparece. Se a expressão calculada der zero ou um valor acima do número de linhas listado, a execução do programa continua na primeira linha a seguir do ON-GOTO.

Não disponível em Integer BASIC. (Veja GOTO para Integer BASIC, na forma de GOTO calculado.)

OPEN

Prepara um arquivo de texto seqüencial ou aleatório para acesso.

Formato:

OPEN *nomearq* [, L*n*] [, D*n*] [, S*n*] [, V*n*]

Um *buffer* na memória de 595 bytes é alocado para o arquivo de texto especificado. Os comandos READ e WRITE podem agora ser usados como arquivo para obter e guardar informação. Se o arquivo especificado não está no disco, é criado um. Se ele já está aberto, é fechado e em seguida reaberto.

Se o parâmetro L não é especificado, o arquivo é aberto como seqüencial.

Se o parâmetro L é especificado, o arquivo é aberto como de acesso aleatório. O número *n* seguinte ao L é o tamanho do registro em bytes e deve ser uma constante inteira na faixa de 1 a 32767. Ele pode estar ausente; sendo usado L1. Cada vez que um arquivo particular de acesso aleatório é aberto, o tamanho do registro deve ser o mesmo.

Se o disco no drive D*n* do slot S*n* não tiver o volume V*n*, ocorre o erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* ou S*n* é omitido, o drive referenciado por último é usado. E usado V0 se V*n* está ausente. Também *n* pode estar ausente, quando é adotado D0, S0 e V0.

Este é um comando DOS e requer PRINT e CTRL-D quando em modo programado. OPEN não pode ser usado em modo imediato.

PDL

Listado na seção de funções deste capítulo.

PEEK

Listado na seção de funções deste capítulo.

PLOT

Mostra um ponto da tela gráfica de baixa resolução.

Formato:

PLOT *col, lin*

Em modo gráfico de baixa resolução, PLOT coloca um ponto colorido na tela. A cor do ponto é determinada pela última declaração COLOR executada. O número de coluna varia de 0 a 39. A coluna 0 é a mais à esquerda da tela; e a coluna 39 é a mais à direita. Os números de linha vão de 0 a 47. Linha 0 é a de cima da tela e 47 é a de baixo. Um ponto colocado nas linhas 40 até 47 cairá nas quatro linhas da janela de texto a menos que uma declaração POKE −16302,0 seja executada para eliminar a janela de texto.

No modo texto ou na janela de texto, o PLOT coloca um caractere colorido, em vez de um ponto na tela. Desde que um caractere ocupa o espaço de dois pontos verticais empilhados, existem dois conjuntos diferentes de coordenadas PLOT que causam o aparecimento de um dado caractere num mesmo ponto. Para colocar um caractere particular na tela, deve dar PLOT em ambas as metades da posição do caractere. O caractere que aparece é determinado pela declaração COLOR executada por último antes que cada metade seja desenhada.

A Tabela 8.5 mostra os caracteres gerados em cada combinação de cores nos pontos superior e inferior. Para gerar um caractere particular, ache-o na Tabela 8.5; veja para cima e para a esquerda para ver a cor que se deve usar no PLOT em cada metade do caractere. A metade superior do caractere é especificada por um PLOT em uma linha ímpar, e a metade de baixo por um PLOT em uma linha ímpar. Se apenas for desenhada a metade de um caractere, o caractere gerado é baseado na cor do PLOT e na cor já presente no outro ponto gráfico desenhado.

Note que cada caractere pode ser gerado por quatro cores diferentes para o ponto gráfico inferior. Entretanto eles não são idênticos. O caractere gera em vídeo reverso se o número da cor do ponto inferior for menor que 4, e estará piscando se o número estiver entre 4 e 7, sendo da forma normal na faixa entre 8 e 15.

TABELA 8.5. Caracteres Gerados por Declarações Gráficas em Modo Texto.

	MODO VÍDEO				COR DO PONTO GRÁFICO SUPERIOR															
	Para Vídeo Inverso	Para Caracteres Intermitentes	Para Caracteres Normais		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
COR DO PONTO GRÁFICO INFERIOR	0	4	8	12	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	1	5	9	13	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	2	6	10	14	¤	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
	3	7	11	15	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?

POKE

A declaração POKE guarda um byte de dado numa posição de memória especificada.

Formato:

POKE *endmem, byte*

Um valor entre 0 e 255, fornecido por byte, é escrito no endereço de memória *endmem.* Se a posição de memória especificada exceder o máximo tamanho de memória existente (exemplo, 16383 para sistema com 16K), ou acessar um dispositivo de saída que não está operando, o POKE não terá efeito.

Use com cuidado o POKE. Algumas posições de memória contêm informação essencial para a operação ininterrupta do Apple II. Mexer nessas posições de memória pode destruir o programa, travar o sistema ou alterar o BASIC.

POP

Faz com que o Apple II esqueça a posição de retorno para o mais recente GOSUB executado.

Formato:

POP

O POP efetivamente muda o GOSUB mais recentemente executado em uma declaração GOTO (a posteriori). A próxima declaração RETURN executada pulará para a instrução em seguida ao segundo mais recente GOSUB executado. Se o número total de declarações POP e RETURN

executadas num programa exceder o número de declarações GOSUB, ocorre uma mensagem de erro.

POSITION

Ao mover o apontador de arquivo do disco move-se um determinado número de posições para a frente.

Formato:

POSITION *nomearq* [, R*n*]

Se o arquivo não está aberto quando POSITION é executado, ele é aberto (veja OPEN). O parâmetro R especifica quantos campos o apontador de arquivo se move para a frente a partir de sua posição presente. O número a seguir ao R deve ser uma constante inteira num valor entre 0 e 32767. Se ausente, o parâmetro adotado é 0, ou seja, o apontador não se move. Se o arquivo é aberto por POSITION, os campos são contados a partir do seu início. Esse arquivo deve ser seqüencial.

Um campo consiste numa série de caracteres terminados com um *carriage return*. O POSITION vai ao longo do arquivo, caractere por caractere, contando tais caracteres; o número de *carriages returns* encontrado é o número de campos pulados. Se qualquer caractere não usado é encontrado, a mensagem END OF DATA aparece.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

POSITION não pode ser usado em modo imediato.

PR#

Seleciona o slot que irá receber as próximas saídas.

Formato:

PR# *slot*

Toda declaração PRINT a seguir vai mandar dados para o periférico do slot indicado. *slot* deve ser um inteiro entre 0 e 7. O slot 0 não é um dispositivo periférico. PR#0 especifica a tela de vídeo como saída. Se não houver periférico no slot especificado, o sistema travará até que seja acionado RESET.

Sempre que o DOS estiver presente na memória do Apple II, PR# será considerado um comando DOS, e vai requerer PRINT e CTRL-D em modo programado.

PRINT

Manda caracteres para a tela ou outro dispositivo de saída.

Formato:

PRINT [*expr*] [{;,}. . .[*expr*]]. . .]

Existem diversas variações aceitáveis da declaração PRINT. O PRINT sozinho manda um caractere *carriage return* e *line feed*. Quando é seguido por uma ou mais expressões, o valor dessas expressões é mostrado. A forma como os valores aparecem depende de sua natureza e do uso de ponto-e-vírgula ou vírgula entre eles.

Todos os valores numéricos em Integer BASIC e muitos em Applesoft são mostrados usando representação numérica padrão. Valores negativos são precedidos pelo sinal de menos; valores positivos não são precedidos por sinal ou espaço em branco. É usada notação científica em Applesoft para valores mais próximos de zero que ±0.01 e por qualquer valor com mais de nove dígitos antes do ponto decimal.

Os valores série aparecem da forma como são.

Vírgulas (,) e pontos-e-vírgulas (;) determinam o espaço entre cada valor mostrado. Um ponto-e-vírgula faz o valor seguinte ser impresso imediatamente após o último mostrado; eles ficam concatenados sem nenhum espaço intermediário. Uma vírgula faz o próximo valor ser colocado na posição de tabulação seguinte, alguns espaços à frente a partir do último valor.

Em Integer BASIC, as posições de tabulação são as colunas 1, 9, 17 etc. Se algum caractere diferente de espaço for colocado no espaço anterior ao de tabulação (ex. na coluna 16), aquela posição de tabulação ficará desativada.

O Applesoft coloca posições de tabulação em intervalos de 16, nas colunas 1, 17, 33 etc. As paradas de tabulação na tela serão inativadas de acordo com o esquema ilustrado na Tabela 4.1 (Capítulo 4). Para outros dispositivos, uma parada de tabulação será desativada se houver um caractere em sua coluna anterior (ex. coluna 32).

Se uma lista de expressões não termina com vírgula ou ponto-e-vírgula, é dado um *carriage return* e *line feed* depois do último item da lista. Se ela termina com ponto-e-vírgula, o primeiro caractere que aparecer na declaração NEXT seguinte aparecerá imediatamente após o último caractere mostrado pela declaração PRINT atual, sem espaços. Se a lista termina com vírgula, a próxima saída iniciará com o caractere na posição de tabulação seguinte.

Em Integer BASIC, todos os itens devem ser separados ou por vírgula, ou por ponto-e-vírgula. Em Applesoft, os itens podem ser listados sem vírgulas ou pontos-e-vírgulas intermediários. A saída para tais itens fica concatenada como se eles fossem separados por pontos-e-vírgulas.

O Applesoft reconhece o ponto de interrogação (?) com abreviação de PRINT. Neste caso a palavra PRINT será apresentada na listagem do programa.

READ (DECLARAÇÃO DE DISCO)

Especifica o arquivo em disco de onde os comandos GET e INPUT pegarão dados.

Formato:

READ *nomearq* [, R*n*] [, B*n*]

Se o arquivo ainda não está aberto, ele é aberto (veja OPEN). Toda declaração GET e INPUT

recebe caracteres do disco até que uma declaração de disco (ou um caractere CTRL-D (código ASCII 04) sozinho) RESET seja acionada, ou ocorra um erro. Se o arquivo não estiver no disco a mensagem FILE NOT FOUND aparece.

O arquivo será lido como seqüencial se o parâmetro R estiver ausente. Num arquivo seqüencial, o parâmetro B especifica em qual byte está (caractere) o READ inicial. Não tendo B, o READ inicia a partir do primeiro byte (byte 0). Se o byte a ser lido nunca foi gravado no disco por meio da declaração WRITE, aparece a mensagem END OF DATA ao se executar alguma instrução INPUT E GET subseqüente.

Se o parâmetro R está presente, o arquivo será acessado como de acesso aleatório. O parâmetro R especifica que registro será acessado com READ.

Num arquivo de acesso aleatório, o parâmetro B especifica em qual byte do registro especificado a leitura se inicia. Sem o parâmetro B, o READ inicia no primeiro byte do registro (byte 0). Se o byte a ser lido não foi gravado no disco antes com WRITE, a mensagem END OF DATA aparece na declaração INPUT ou GET executada em seguida.

Os números que seguem B e R devem ser constantes inteiras entre 0 e 32767. Se não especificadas, 0 é assumido.

Não use o CTRL-C para parar uma declaração READ no Applesoft. Isso causa uma série de mensagens ?REENTER. Use apenas o RESET para parar o programa.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

Não pode ser usado em modo imediato.

READ (DECLARAÇÃO DE ATRIBUIÇÃO)

Associa valores de DATA do Applesoft a variáveis.

Formato:

READ *var* [, *var* . . .]

Existe um apontador para a lista de dados DATA que determina qual valor deve ser associado à variável da declaração READ. No início do programa e após a declaração RESTORE, o apontador indica o primeiro valor de DATA. A cada declaração READ em que se atribui um valor para a variável, o apontador passa para o valor seguinte.

As variáveis podem ser de qualquer tipo, porém devem coincidir com o tipo de dado indicado pelo apontador do DATA. Um valor numérico atribuído a uma variável série causa problemas. Uma série associada a uma variável numérica provoca o aparecimento da mensagem de erro ?SINTAX ERROR. O número de linha do DATA envolvido também é mostrado com a mensagem.

Se o READ tenta ler mais variáveis que as existentes na declaração data, a mensagem ?OUT OF DATA ERROR é mostrada, com o número da declaração READ onde houve o erro.

O READ pode ser executado em modo imediato, desde que o programa na memória tenha valores em DATA correspondentes. De outra forma, a mensagem ?OUT OF DATA ERROR aparece. Se o DOS está presente, o READ em modo imediato é tratado como um comando DOS, e a mensagem NOT DIRECT COMMAND é apresentada.

Não disponível em Integer BASIC.

RECALL

Pega uma matriz numérica do Applesoft a partir da fita cassette.

Formato:

RECALL *varnm*

O Applesoft espera indefinidamente até que uma matriz seja encontrada na fita; nenhuma outra declaração pode ser executada nesse ínterim. O RECALL não controla o movimento da fita nem informa quando acionar o botão PLAY do gravador. Deve-se acrescentar informação através de PRINT antes de uma declaração RECALL, advertindo o início do processo. O Apple II também sinaliza com *bip* quando começa a carregar uma matriz e dá outro *bip* quando pára. A matriz deve ser dimensionada antes da declaração RECALL ser executada, ou a mensagem ?OUT OF DATA ERROR aparece (veja DIM).

Na declaração RECALL não é necessário usar os mesmos nomes de variáveis, usados na declaração STORE que gravou a matriz. Se a matriz da fita contiver mais elementos que os solicitados pelo RECALL, o erro ?OUT OF DATA ERROR aparece. Se a matriz chamada contém no mínimo tantos elementos quantos forem gravados, porém sem as mesmas dimensões, a mensagem ERR é gerada, porém a execução do programa continúa.

Se a matriz chamada tiver mais elementos que a gravada, os valores obtidos normalmente serão danificados. Existem duas opções. Pode-se chamar para uma matriz com o mesmo número de dimensões da gravada, onde cada dimensão, exceto a última, tenha o mesmo tamanho da dimensão correspondente da matriz gravada. A última dimensão pode ser mais larga na matriz chamada. Pode-se também chamar uma matriz com mais dimensões que a gravada, se as dimensões que estão na matriz coincidem com aquelas da matriz chamada (ou as exceda, no caso de última dimensão da matriz gravada).

Matrizes tipo série não podem ser usadas com RECALL. Porém, os valores numéricos podem ser convertidos para série, usando a função CHR$.

Não disponível em Integer BASIC.

REM

A declaração REM permite comentários a serem colocados no programa para efeito de documentação.

Formato:

REM *comentário*

comentário é qualquer seqüência que caiba na linha de programa.

As declarações REM são reproduzidas na listagem do programa, porém são ignoradas durante sua execução. A declaração REM pode ser colocada numa linha sozinha ou como uma das declarações de uma linha múltipla.

O REM não pode ser colocado à frente de outras declarações numa linha múltipla, porque todo o texto que segue é tratado como um comentário.

RENAME

Troca o nome de um arquivo em disco sem alterar seu conteúdo.

Formato:

RENAME *nomearq*$_1$, *nomearq*$_2$ [, D*n*] [, S*n*] [, V*n*]

O arquivo com nome *nomearq*$_1$ é achado no disco, e tem seu nome trocado por *nomearq*$_2$. Se o arquivo está aberto, ele é fechado (veja CLOSE). O arquivo não é afetado em seu conteúdo.

RENAME pode trocar o nome do arquivo por outro que já exista dentro do disco; e pode fazê-lo muitas vezes. Por isso deve-se ter cuidado para não escolher para um arquivo um nome que já exista no disco, antes de executar o RENAME.

Se *nomearq*$_1$ não existe no drive D*n* do slot S*n*, a mensagem de erro FILE NOT FOUND aparece. Se o disco no drive D*n* do slot S*n* não tem volume V*n*, a mensagem VOLUME MISMATCH aparece.

D*n*, S*n* e V*n* podem aparecer em qualquer ordem. Se D*n* e S*n* são omitidos, o drive referenciado por último é usado. V0 é usado se V*n* não for colocado. Também *n* pode estar ausente, quando D0, S0 e V0 são adotados.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

RESTORE

Coloca o apontador do DATA no início da lista.

Formato:

RESTORE

A declaração READ subseqüente vai operar a partir do primeiro valor da lista do DATA.

Não disponível em Integer BASIC.

RESUME

Faz o programa em Applesoft voltar a execução no início da instrução onde o erro ocorreu.

Formato:

RESUME

O RESUME não pode ser usado após uma declaração ONERR GOTO ter sido usada no tratamento do erro. Se o RESUME é executado sem ter ocorrido erro, o resultado é imprevisível, mas em geral trágico.

Não disponível em Integer BASIC.

Não pode ser usado em modo imediato.

RETURN

Faz o programa saltar para a declaração abaixo da última declaração GOSUB executada.

Formato:

RETURN

A declaração POP faz o programa esquecer o endereço de retorno do GOSUB mais recente. Uma declaração RETURN após um pop faz voltar para o endereço seguinte ao segundo mais recente GOSUB.

Se forem executadas mais declarações RETURN (ou POP) do que GOSUB, aparece uma mensagem de erro.

ROT=

Esta declaração do Applesoft muda a orientação do desenho da forma gráfica de alta resolução feito por DRAW ou XDRAW.

Formato:

ROT= *exprnm*

ROT=0 desenha a forma na orientação que foi definida. A forma é rodada em 90 graus em sentido horário para cada incremento de 16 no valor de *exprnm.* Assim, ROT=32 desenha uma forma de cabeça para baixo, e ROT=64 desenha-a em sua forma original. Valores de *exprnm* maiores que 64 não calculados no módulo 16.

Quando SCALE foi colocado em 1, só existem quatro formas possíveis de ROT=. São elas 0, 16, 32 e 48 (e os valores maiores que 63 equivalentes). Quando SCALE é igual a 2, ficam existindo oito valores possíveis para ROT=, quando SCALE=3, existem 16 valores etc. até o máximo de 64 valores diferentes. Um valor não reconhecível do ROT= será tratado como se fosse o valor reconhecível seguinte.

exprnm deve ter um valor calculado entre 0 e 255 ou a mensagem ?ILLEGAL QUANTITY ERROR é gerada quando o comando ROT= é executado.

ROT= não é reconhecido como uma palavra reservada, a menos que o caractere "=" seja o primeiro caractere diferente de espaço.

Não disponível em Integer BASIC.

RUN (DECLARAÇÃO DE DISCO)

Carrega e executa um programa do disco.

Formato:

RUN *nomearq* [, D*n*] [, S*n*] [, V*n*]

O programa chamado *nomearq* é carregado do disco e executado. Se o carregamento for correto, qualquer programa anterior da memória é zerado.

Se o programa a ser carregado for em Integer BASIC e o Apple II estiver em Applesoft, ou vice-versa, ele fará a mudança automaticamente para a linguagem adequada. Se necessário carregará o interpretador Applesoft a partir do disco especificado. Se a linguagem não estiver disponível, a mensagem LANGUAGE NOT AVAILABLE aparecerá.

Se o arquivo não existe no drive D*n* do slot S*n*, a mensagem FILE NOT FOUND aparecerá. Se o disco do drive D*n* do slot S*n* não tiver volume V*n* a mensagem de erro VOLUME MISMATCH aparecerá.

D*n*, S*n* e V*n* podem ser especificados em qualquer ordem. Se D*n* e S*n* são omitidos, o drive referenciado por último é adotado. V0 é usado por ausência de V*n*. Também *n* pode estar ausente, quando D0, S0 e V0 são adotados.

Este é um comando DOS e requer PRINT e CTRL-D em modo programado.

RUN (DECLARAÇÃO GERAL)

Executa o programa presente na memória, iniciando no número de linha especificado, se existir; de outra forma, é usado o endereço mais baixo do programa.

Formato Geral:

RUN [*linha*]

Se não existir linha de nome *linha*, a mensagem *** BAD BRANCH ERR em Integer BASIC e ?UNDEF'D STATEMENT ERROR em Applesoft aparecerá.

Formato Adicional do Integer BASIC:

RUN *exprnm*

Em Integer BASIC, o número de linha pode ser uma expressão numérica.

RUN só pode ser usado no modo imediato em Integer BASIC.

SAVE

Guarda o programa da memória em cassette ou disco.

Formato em Cassette:

SAVE

Guarda o programa presente na memória em fita cassette. Deve-se ter a fita em modo RECORD quando SAVE é executado. O Apple II não indica quando fazê-lo. Ele apita quando inicia a gravação do programa e apita de novo quando termina. O segundo apito é o sinal para parada manual do gravador.

Só pode ser usado em modo imediato em Integer BASIC.

Formato em Disco:

SAVE *nomearq* [, D*n*] [, S*n*] [, V*n*]

Se não existe arquivo com nome *nomearq* no disco, é criado um com aquele nome e a linguagem do programa atual. O programa é guardado. Se existe um arquivo chamado *nomearq* na mesma linguagem do programa corrente, o conteúdo daquele arquivo é zerado e o programa corrente é guardado em seu lugar. Se o programa *nomearq* existe, porém em linguagem diferente ou com tipo diferente, a mensagem FILE TYPE MISMATCH aparece.

Se o arquivo não existe no drive D*n* do slot S*n*, aparece a mensagem FILE NOT FOUND. Se o disco do drive D*n* do slot S*n* não tiver volume V*n* a mensagem de erro VOLUME MISMATCH aparecerá.

D*n*, S*n* e V*n* podem ser especificados em qualquer odem. Se D*n* e S*n* são omitidos, o drive referenciado por último é adotado. V0 é usado por ausência de V*n*. Também *n* pode estar ausente, quando D0, S0 e V0 são adotados.

Este é um comando DOS e requer PRINT e CTRL-D em modo programado.

SCALE

Esta declaração Applesoft dá o tamanho da forma gráfica de alta resolução desenhada por DRAW ou XDRAW.

Formato:

SCALA= *exprnm*

O tamanho da forma da tabela é multiplicado pelo valor de *exprnm.* Assim, se SCALE=1 a forma será desenhada como foi definida. Se SCALE=2 será desenhada com o dobro do tamanho. Se SCALE=0, a forma será desenhada 255 vezes maior que o tamanho original.

O valor de *exprnm* deve ser um inteiro entre 0 e 255 ou a mensagem ?ILLEGAL QUANTITY ERROR ocorre quando o comando SCALE é executado.

SCALE não é reconhecido como palavra reservada a menos que o caractere "=" seja o primeiro a seguir diferente de branco (espaço).

Não disponível em Integer BASIC.

SHLOAD

Esta declaração do Applesoft carrega uma forma gráfica de alta resolução da fita cassette.

Formato:

SHLOAD

A construção e o armazenamento de forma gráfica em cassette são estudados no Capítulo 6.

A forma gráfica é carregada imediatamente abaixo de HIMEM: e HIMEM. é colocado imediatamente abaixo da forma. Então o endereço inicial da forma é carregado nos endereços de memória 22 e 23.

Antes que se execute SHLOAD certifique-se de ter colocado HIMEM: de forma que a tabela não seja carregada sobre o programa ou variáveis e não vá ser zerada pelos gráficos. Veja o estudo sobre HIMEM: neste capítulo, mapas de memória no Apêndice G e no Capítulo 6 para maiores detalhes.

Não disponível em Integer BASIC.

SPEED

Esta declaração Applesoft troca a freqüência com que os caracteres são saídos.

Formato:

SPEED = *exprnm*

O valor de *exprnm* determina a freqüência com a qual os caracteres aparecerão na tela ou outro dispositivo de saída. A velocidade varia de 0 (mais lenta) até 255 (mais rápida).

Não disponível em Integer BASIC.

STOP

Faz o Applesoft parar a execução do programa.

Formato:

STOP

O Apple II retorna ao modo imediato. A mensagem BREAK IN linha aparece, onde linha é o número da linha na qual o programa parou.

Não disponível em Integer BASIC.

STORE

Guarda uma matriz específica do Applesoft em fita cassette.

Formato:

STORE *varnm*

O STORE não controla o movimento da fita nem adverte quando se deve acionar o botão RECORD do gravador. Deve-se ter o cassette rodando e pronto para gravar quando o STORE é executado. O programa em Applesoft deve mostrar indicações (via declaração PRINT). O Apple II sinaliza quando inicia a guarda de valores e sinaliza de novo quando termina.

Só se pode usar STORE para guardar valores numéricos; variáveis série devem ser convertidas para valores inteiros usando a função ASC antes de salvar (veja também RECALL).

Não disponível em Integer BASIC.

TAB

Esta declaração em Integer BASIC posiciona o cursor numa coluna específica na linha corrente.

Formato:

TAB *col*

O cursor se move para a direita ou para a esquerda de acordo com o valor de *col*, sem zerar os caracteres já mostrados. As colunas são numeradas de 1 a 40 (da esquerda para a direita).

Para Applesoft, use a declaração HTAB. Veja também a função TAB na seção sobre funções neste capítulo.

TEXT

Retorna a tela para o modo texto a partir de qualquer um dos modos gráficos.

Formato:

TEXT

O caractere de entrada e o cursor são movidos para a última linha da tela; se em modo texto este é o único efeito. Se a janela de texto foi colocado como algo diferente de tela cheia, o TEXT apaga a tela cheia.

O TEXT não apaga a tela, ou mais precisamente, não limpa a página 1 de memória de baixa resolução. Desde que o modo texto normal usa a mesma tela que o gráfico em baixa resolução, a execução de TEXT no modo gráfico de baixa resolução deixa as 20 linhas de cima da tela cheias de caracteres estranhos.

TRACE

Mostra o número da linha de cada declaração executada.

Formato:

TRACE

Esta ferramenta de depuração causa o aparecimento dos números das linhas entremeado com os caracteres saídos pelo programa, o que pode tornar um ou ambos ininteligíveis. O TRACE é desativado com o NO TRACE.

UNLOCK

Remove o estado de travado de um arquivo em disco, de forma que ele pode ser alterado ou eliminado.

Formato:

UNLOCK *nomearq* [, D*n*] [, S*n*] [, V*n*]

Se o arquivo especificado está travado, a trava é removida. Se não, nada acontece (veja LOCK).

Se o arquivo não existe no drive D*n* do slot S*n*, a mensagem FILE NOT FOUND aparece. Se o disco no drive D*n* do slot S*n* não tem volume V*n*, aparece o erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem aparecer em qualquer ordem. Se D*n* e S*n* são omitidos o drive referenciado por último é usado. Quando V*n* está ausente, V0 é adotado. Também *n* pode estar ausente, quando D0, S0 e V0 são adotados.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

USR

Listado na parte de funções deste capítulo.

VERIFY

Verifica um arquivo específico em disco para testar autoconsistência.

Formato:

VERIFY *nomearq* [, D*n*] [, S*n*] [, V*n*]

Quando um arquivo é guardado em disco com SAVE, BSAVE ou PRINT, é calculado um dígito de verificação para cada setor, que também é gravado no disco. O VERIFY recalcula este dígito e

o compara com o gravado. Se eles coincidirem, o arquivo está intacto e nada aparece na tela. Se ocorrerem erros em um ou mais setores, a mensagem I/O ERROR é gerada. Pode ser verificado qualquer tipo de arquivo.

Se o arquivo não existe no drive D*n* do slot S*n*, a mensagem FILE NOT FOUND aparece. Se o disco no drive D*n* do slot S*n* não tem volume V*n*, aparece o erro VOLUME MISMATCH.

D*n*, S*n* e V*n* podem aparecer em qualquer ordem. Se D*n* e S*n* são omitidos o drive referenciado por último é usado. Quando V*n* está ausente, V0 é adotado. Também *n* pode estar ausente, quando D0, S0 e V0 são adotados.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

VLIN

Desenha uma linha vertical na tela, na forma gráfica de baixa resolução.

Formato:

VLIN lin_1, lin_2, AT *col*

A linha é desenhada de lin_1 até lin_2 na coluna especificada por *col.* A cor é determinada pela declaração COLOR executada por último. Se a tela estiver em modo texto, ou se a janela de texto estiver presente com a linha maior que 39, parte ou a totalidade da linha aparecerá como caracteres ao invés de pontos. Os caracteres usados são determinados pela declaração COLOR anterior; veja a Tabela 8.5 (perto da declaração PLOT neste capítulo) para particularidades.

Em Integer BASIC, lin_1 deve ser menor ou igual à lin_2, ou a mensagem *** RANGE ERR aparece.

VTAB

Posiciona o cursor para uma linha especificada na coluna atual.

Formato:

VLIN *lin*

O cursor se move para cima ou para baixo até a linha especificada pelo valor de *lin*, sem zerar nenhum dos caracteres da tela. As linhas são numeradas de 1 a 24 (de cima para baixo).

WAIT

Para a execução de um programa Applesoft até que um endereço de memória tenha uma condição especificada.

Formato:

WAIT *endmem, exprnm*$_1$ [, *exprnm*$_2$]

O WAIT verifica parte ou o total dos oito bits da posição de memória *endmem*, para testar o padrão de uns e zeros especificado por *exprnm*$_1$. O valor binário de *exprnm*$_1$ determina quais bits da posição de memória devem ser considerados e quais devem ser ignorados. Se um bit de *exprnm*$_1$ é 1, então o bit correspondente da posição *endmem* é testado. O WAIT ignora aqueles bits de memória que são correspondentes a 0 no valor binário de *exprnm*$_2$.

Enquanto os bits significativos (determinados por *exprnm*$_1$) de *endmem* forem todos diferentes do valor correspondente de *exprnm*$_2$, a espera continua. No momento em que qualquer par for idêntico (ou 1 ou 0) a espera termina e o programa Applesoft continua.

Se *exprnm*$_2$ não está presente, é adotado 0.

O WAIT só pode ser interrompido por RESET (ou desligamento). O valor das expressões numéricas devem estar na faixa de 0 até 255 ou a mensagem ?ILLEGAL QUANTITY ERROR é gerada. Se a posição de memória especificada é maior que a posição de memória mais alta, (ex. 32767, com 32K de memória), ou acessar um dispositivo de saída que não está operando, o WAIT trava o sistema até o acionamento do RESET.

Não disponível em Integer BASIC.

WRITE

Especifica um arquivo em disco para onde as saídas subseqüentes serão mandadas.

Formato:

WRITE *nomearq* [, R*n*] [, B*n*]

Se o arquivo especificado não está aberto, ele o é (veja OPEN). As declarações PRINT subseqüentes mandarão dados no disco até uma declaração de disco (ou o caractere CTRL-D (código ASCII 4) sozinho) seja mandado. Se o arquivo não estiver no disco, a mensagem FILE NOT FOUND aparecerá.

O arquivo será tratado como seqüencial se o parâmetro R estiver ausente. Num arquivo seqüencial, o parâmetro B especifica em que byte (caractere) o WRITE inicia. Sem o parâmetro B, o WRITE inicia no primeiro byte do arquivo (byte 0).

Se o parâmetro R está presente, o arquivo será gravado como arquivo de acesso aleatório. O WRITE é para o registro especificado pelo parâmetro R.

Num arquivo de acesso aleatório, o parâmetro B especifica em que byte dentro de um registro especificado o WRITE inicia. Sem o parâmetro B, o WRITE inicia no primeiro byte (byte 0) do registro.

O parâmetro B pode ser usado para gravar iniciando num ponto além do último caractere do arquivo (ou registro). Este dado pode ser lido (com READ), porém qualquer tentativa de ler bytes intermediários não usados vai gerar a mensagem OUT OF DATA.

Os números à frente de R e B devem ser inteiros na faixa de 0 até 32767. Se não especificados, 0 é usado.

Enquanto WRITE estiver em uso, todo caractere que o Apple II mandaria normalmente para a tela será mandado para o disco. Incluindo pontos de interrogação gerados por INPUT e as mensagens de erro.

Este é um comando DOS, e requer PRINT e CTRL-D em modo programado.

Não pode ser usado em modo imediato.

XDRAW

Esta declaração Applesoft desenha uma forma gráfica de alta resolução na tela e, se usada uma segunda vez com os mesmos parâmetros, limpa a forma.

Formato:

XDRAW *exprnm* [AT *colh, linh*]

A forma número *exprnm* da tabela de formas é desenhada, com cada ponto na cor que é o complemento da cor da tela no ponto. Cores 0 e 3 são um par complementar, como são 1 e 2, 4 e 7, e 5 e 6 (veja Tabela 8.3). A escala e rotação da forma deve ser dada pelos comandos COLOR e SCALE antes do comando XDRAW ser executado.

Usa-se XDRAW ao invés de DRAW de forma que se pode facilmente limpar uma forma desenhada. Como XDRAW desenha na cor complementar à cor anterior do ponto, se executada duas vezes (ou quatro, ou seis), com os mesmos parâmetros, a declaração XDRAW, tudo o que estiver na tela não será alterado.

Se não se especifica uma posição na declaração XDRAW, a forma é desenhada iniciando na posição desenhada por último no comando DRAW, XDRAW ou PLOT anterior. Se não se especificar posição, o desenho inicia naquele ponto (*colh, linh*).

O número da forma, *exprnm*, deve ser um valor inteiro entre 0 e o número de formas da tabela (que não deve exceder 255), inclusive.

Não disponível em Integer BASIC.

FUNÇÕES

As funções do BASIC do Apple II são descritas abaixo em ordem alfabética. A nomenclatura e as abreviações são descritas no início deste capítulo.

Muitas das funções são disponíveis apenas em Applesoft. Estas funções são identificadas apropriadamente.

ABS

Retorna o valor absoluto de um número. Isto é, o valor sem sinal.

Formato:

ABS (*exprnm*)

ASC

Retorna o código ASCII de um caractere especificado.

Formato:

ASC (*expr$*)

Se a série tem mais de um caractere, o ASC devolve o código do primeiro caractere da série. O código devolvido não é necessariamente o menor código (na faixa 0-95) que aquele caractere. Os caracteres gerados pelo código ASCII entre 96 e 255 duplicam aqueles da faixa menor na tela. Entretanto, eles não são calculados como os mesmos caracteres pelos operadores relacionais como $<$, $>$ e $=$. Eles devem ser tratados diferentemente para impressoras e outros dispositivos, da mesma forma. Se o primeiro caractere da *expr$* é CTRL-@ (código ASCII 0), a mensagem ?SYNTAX ERROR aparece. Se *expr$* é uma série nula, a mensagem ?ILLEGAL QUANTITY ERROR aparece.

Os códigos ASCII são listados no Apêndice I.

ATN

Retorna o arco-tangente de um argumento.

Formato:

ATN (*exprnm*)

Calcula o arco-tangente, em radianos, de *exprnm*. O ângulo devolvido está na faixa entre $-\pi/2$ até $\pi/2$.

Não disponível em Integer BASIC.

CHR$

Retorna o valor série de um código ASCII especificado.

Formato:

CHR$ (*exprnm*)

Retorna o caractere representado pelo valor inteiro da expressão *exprnm*, interpretado como um código ASCII. Acha-se a tabela de códigos ASCII no Apêndice I. Use esta função para gerar carac-

teres que não se consegue obter a partir do teclado para controle de periféricos etc. O valor de *exprnm* deve estar entre 0 e 255 ou a mensagem ?ILLEGAL QUANTITY ERROR aparecerá.

Não disponível em Integer BASIC.

COS

Retorna o co-seno de um ângulo.

Formato:

COS (*exprnm*)

Calcula o co-seno de *exprnm* radianos.

Não disponível em Integer BASIC.

EXP

Retorna *e* elevado a uma potência.

Formato:

EXP (*exprnm*)

Calcula *e* (a base do logaritmo natural: 2,718281828) elevado à potência *exprnm.*

Não disponível em Integer BASIC.

FN

Chama uma função definida anteriormente pelo usuário.

Formato:

FN *varnm* (*exprnm*)

varnm é o nome da função. O valor da expressão é adotado sempre que a variável fictícia aparece na definição da função, e o resultado da expressão é calculado. Veja DEF FN na parte de declarações neste capítulo.

Uma função não pode ser recursiva, isto é, *exprnm* não pode referenciar FN *varnm* nem outra função que, por sua vez, referencie FN *varnm.*

Ao se tentar usar FN *varnm* antes da definição de função DEF FN *varnm* aparece a mensagem de erro ?UNDEF'D FUNCTION ERROR.

Não disponível em Integer BASIC.

FRE

Retorna o número de bytes da memória presentemente disponível para o programa BASIC em Applesoft.

Formato:

FRE (*exprnm*)

A memória disponível é aquela abaixo da área de armazenamento de séries e acima da área de armazenamento de matrizes. Existindo mais de 32767 bytes de memória livres, o FRE devolve um número negativo. Some 65536 ao número para achar o tamanho real da memória disponível.

O FRE também limpa séries não usadas da área de armazenamento. Quando uma série troca de valor durante um programa, o valor anterior é deixado na memória, e o novo valor é acrescentado à tabela. Eventualmente isso pode ser danoso, se for usado para outra coisa. Para evitar essa operação, use a declaração A = FRE(0) com alguma freqüência em programas que usam séries extensivamente.

O valor de *exprnm* não é usada por FRE, porém causa um erro se ilegal.

Não disponível em Integer BASIC.

INT

Retorna a parte inteira de um número.

Formato:

INT (*exprnm*)

Retorna o maior inteiro menor ou igual ao valor de *exprnm.*

Não disponível em Integer BASIC.

LEFT$

Retorna o caractere mais à esquerda de uma série.

Formato:

LEFT$ (*expr$, exprnm*)

Retorna o caractere mais à esquerda de *expr$. exprnm* deve ser um inteiro entre 1 e 255, e *expr$* não pode ter mais de 255 caracteres. Se *exprnm* for maior que o tamanho de *expr$*, toda a série é devolvida.

Não disponível em Integer BASIC.

LEN

Retorna o comprimento de uma série.

Formato:

LEN (*expr$*)

Conta o número de caracteres de *expr$*, incluindo todos os espaços e caracteres não imprimíveis. Se *expr$* tem mais de 255 caracteres (possível apenas se *expr$* é produto de concatenação de séries), a mensagem ?STRING TOO LONG ERROR aparece.

LOG

Retorna o logaritmo natural de um número.

Formato:

LOG (*exprnm*)

Calcula o logaritmo natural de *exprnm.* Retorna ?ILLEGAL QUANTITY ERROR se *exprnm* é zero ou negativo.

Não disponível em Integer BASIC.

MID$

Retorna a porção especificada de uma série.

Formato:

MID (*expr$, exprnm*$_1$ [, *exprnm*$_2$])

Retorna *exprnm*$_2$ caracteres de *expr$*, iniciando no caractere *exprnm*$_1$. Se *exprnm*$_2$ estiver ausente, MID$ retorna a porção de *expr$* a partir do caractere *exprnm*$_1$ até o último. Se o comprimento de *expr$* é menor que *exprnm*$_1$, a série nula é devolvida. Se existem menos que *exprnm*$_2$ caracteres em *expr$* após *exprnm*$_1$, o resultado é o mesmo que se *exprnm*$_2$ estiver ausente. *expr$* não deve exceder a 255 caracteres, e *exprnm*$_1$ e *exprnm*$_2$ ambos devem estar entre 1 e 255.

Não disponível em Integer BASIC.

PDL

Retorna o valor corrente do controlador de jogo (paddle) especificado.

Formato:

PDL (*exprnm*)

O valor retornado é um inteiro entre 0 e 255 baseado na rotação do *paddle* número *exprnm*, ou da resistência do dispositivo conectado ao soquete do controlador de jogo *exprnm.* Os controladores de jogo são numerados de 0 a 3. Se for indicado um controlador menor que 0 ou maior que 255 a mensagem ?ILLEGAL QUANTITY ERROR é gerada. Se o controlador for entre 4 e 255, o PDL retorna um valor imprevisível entre 0 e 255, podendo causar vários efeitos colaterais, tal como um sinal no alto-falante ou um súbito deslocamento no modo gráfico.

Se duas instruções PDL são executadas consecutivamente ou muito próximas, o segundo valor pode ser afetado pelo primeiro. Providencie para que várias instruções sejam executadas entre dois PDL (ou coloque um *loop* FOR-NEXT vazio).

PEEK

Retorna o valor de uma posição de memória.

Formato:

PEEK (*endmem*)

O valor retornado é o equivalente decimal aos oito bits da memória endereçada por *endmem.* O Apêndice E lista algumas posições de memória importantes.

POS

Retorna a coluna onde o cursor está posicionado.

Formato:

POS (*exprnm*)

A expressão é fictícia; não é usada, podendo ter qualquer valor legal.

O POS retorna um valor entre 0 e 39. As posições iniciam em 0 para o caractere mais à esquerda.

Não disponível em Integer BASIC.

RIGHT$

Retorna o caractere mais à direita de uma série.

Formato:

RIGHT$ (*expr$, exprnm*)

Retorna o caractere mais à direita da série *expr$*. O valor de *exprnm* deve ser um inteiro entre 0 e 255, e *expr$* não pode ter mais de 255 caracteres. Se *exprnm* é maior ou igual ao comprimento de *expr$*, a série inteira é retornada.

Não disponível em Integer BASIC.

RND

Retorna um número aleatório.

Formato:

RND (*exprnm*)

Retorna um número aleatório, na faixa que depende do valor de *exprnm* e da versão do BASIC.

Em Integer BASIC, o RND retorna um aleatório entre 0 e o valor de *exprnm*, exclusive *exprnm* mas inclusive 0. Assim, RND(1) sempre retorna 0, e RND(−2) produz um misto de 0 e −1. Ao tentar se usar RND(0) aparece a mensagem de erro *** > 32767 ERR.

Em Applesoft, RND sempre retorna um número real maior ou igual a 0 e menor que 1. O valor retornado pode ser de três tipos, dependendo do sinal de *exprnm*.

Se *exprnm* é positiva, RND retorna um valor diferente a cada vez que é usado, a menos que uma seqüência repetida tenha sido iniciada.

Uma seqüência repetida tem início quando *exprnm* é um número negativo. Qualquer valor particular sempre inicia a mesma seqüência; os argumentos positivos subseqüentes sempre vão retornar a mesma seqüência de números aleatórios. Uma seqüência repetida diferente vai retornar para cada valor negativo diferente de *exprnm*. Esta característica é útil para teste e depuração de programas que usam RND.

Se *exprnm* for 0 em Applesoft, RND retorna o número aleatório mais recentemente gerado (isso não é afetado por NEW ou CLEAR).

SCRN

Retorna o código da cor do ponto gráfico de baixa resolução com as coordenadas especificadas.

Formato:

SCRN (*col, lin*)

Se a tela está em modo texto, ou a janela de texto está presente e o ponto especificado está dentro dela, o SCRN devolve o código de cor de metade do caractere. O código da cor da metade de cima do caractere é devolvida se lin é par, e da metade inferior se lin é ímpar. O código ASCII do carac-

tere na posição (*a, b*) (com *a* na faixa de 0-39 e *b* na faixa de 0-23) é devolvido pela expressão SCRN(*a*, 2 * *b*) + 16 * SCRN(*a*, 2 * *b* + 1). Assim, o caractere é devolvido por CHR$(*n*), onde *n* é o valor obtido pela expressão acima.

Se *col* está na faixa de 0-39, SCRN retorna o código da cor do ponto gráfico (*col, lin*). Se *col* está na faixa 0-47 e *lin* na faixa 0-31, SCRN devolve o ponto o número da cor do ponto gráfico (*col* −40, *lin* +16). Se *col* está na faixa 40-47 e *lin* na faixa de 32-47, SCRN devolve um número sem relação com nada na tela.

Se SRCN é usado durante o modo gráfico de alta resolução, o número devolvido está relacionado com área de memória gráfica de baixa resolução em vez de alta resolução.

SRCN só é reconhecida como palavra reservada se o primeiro caractere diferente de espaço à sua frente for parênteses.

SGN

Determina se um número é positivo, negativo ou zero.

Formato:

SGN (*exprnm*)

A função SGN retorna +1 se *exprnm* é positiva, −1 se é negativa, e 0 se *exprnm* é 0.

SIN

Retorna o seno de um ângulo.

Formato:

SIN (*exprnm*)

Calcula o seno de *exprnm* radianos.

Não disponível em Integer BASIC.

SPC

Move o cursor para a direita de um número especificado de posições.

Formato:

SPC (*exprnm*)

A função SPC é usada nas declarações PRINT para imprimir *exprnm* espaços em branco. Assim, qualquer caractere passado pelo cursor é zerado.

A função SPC move o cursor para a direita a partir da coluna em que ele estiver no momento em que SPC for encontrado. Este é um contraste com a função TAB, que move o cursor para números de coluna fixos a partir da coluna mais à esquerda da tela.

Não disponível em Integer BASIC.

SQR

Retorna a raiz quadrada de um número positivo.

Formato:

SQR (*exprnm*)

Um valor negativo de *exprnm* causa a mensagem de erro ?ILLEGAL QUANTITY ERROR. SQR (*exprnm*) opera mais rapidamente que (*exprnm*) ∧ (.5).

Não disponível em Integer BASIC.

STR$

Converte um valor numérico em série

Formato:

STR$ (*exprnm*)

O valor de *exprnm* é convertido em série. A série de caracteres é o mesmo que se obteria com PRINT *exprnm*. Assim, STR$(2/3) = ".666666667" e STR$(2468013579) = "2.46801358E + 09". Se *exprnm* exceder o limite para números reais, a mensagem ?OVERFLOW ERROR aparece.

Não disponível em Integer BASIC.

TAB

TAB move o cursor para a direita de um número especificado de colunas.

Formato:

TAB (*exprnm*)

Use TAB com PRINT para mover o cursor para a coluna *exprnm*, se *exprnm* estiver à direita da posição atual do cursor. O cursor não se move se a posição de tabulação estiver à sua esquerda e em sua posição. TAB coloca caracteres em branco durante o movimento do cursor para a direita, de forma que zera o que havia previamente na tela.

Para TAB, as colunas são numeradas de 1 a 255. Se *exprnm* é maior que o comprimento do dispositivo de saída (40 caractere para o vídeo), ele move o cursor uma linha para baixo e continua contando a partir da margem esquerda. Se o valor de *exprnm* é 0, TAB move para a coluna 256. Um valor de *exprnm* fora da faixa de 0 a 255 causa a mensagem de erro ?ILLEGAL QUANTITY ERROR.

Veja também HTAB (Applesoft) e TAB (Integer BASIC) na parte de declarações neste capítulo.

Não disponível em Integer BASIC.

TAN

Retorna a tangente de um ângulo.

Formato:

TAN (*exprnm*)

Calcula a tangente de *exprnm* radianos.

Não disponível em Integer BASIC.

USR

Pula para uma sub-rotina em linguagem de máquina, passando os valores pelo Acumulador.

Formato:

USR *exprnm*

A sub-rotina inicia na posição 10 (0A hexadecimal). As posições 10 até 12 (0A até 0C em hexadecimal) devem conter uma instrução JMP em linguagem de máquina que salta para o endereço inicial da sub-rotina. Como USR é uma função, ela retorna um valor real. O que tiver no Acumulador quando for executada a instrução RTS em linguagem de máquina (para retornar ao Applesoft) será o valor retornado.

Existem muitas sub-rotinas úteis em linguagem de máquina presente no Monitor do Apple II. Elas estão listadas no Apêndice D.

Veja também a declaração CALL descrita na seção sobre declarações neste capítulo, que também é disponível em Integer BASIC.

Não disponível em Integer BASIC.

VAL

VAL converte uma série em um valor numérico.

Formato:

VAL (*expr$*)

Retorna o valor numérico representado por *expr$.* Se o primeiro caractere de *expr$* não representa um número, zero é devolvido. De outro modo, *expr$* é tomado, caractere por caractere, até que um caractere não aceitável seja acessado. Os caracteres aceitáveis são: os dígitos de 0 a 9, espaços, o ponto decimal, os sinais de mais e menos, um outro ponto decimal e a letra E.

Se *expr$* é uma expressão envolvendo concatenação e contém mais de 255 caracteres, a mensagem ?STRING TOO LONG ERROR ocorre. Se o valor numérico de *expr$* exceder os limites de números reais, a mensagem ?OVERFLOW ERROR ocorre.

Não disponível em Integer BASIC.

A

Funções Numéricas Derivadas

Mesmo que a relação de funções derivadas abaixo não seja completa, ela fornece as fórmulas de uso mais freqüente. Certos valores de x invalidam algumas funções (por exemplo, se COS(x) = 0 então SEC(x) não é real), de forma que o programa deve fazer essas verificações.

Nenhuma das funções derivadas operam em Integer BASIC.

ARCCOS(x) = −ATN(x/SQR(−x * x + 1)) + 1.5707633

Retorna o co-seno inverso de x (ABS(x) < 1).

ARCCOT(x) = −ATN(x) + 1.5707633

Retorna a co-tangente inversa de x.

ARCCOSH(x) = LOG(x + SQR(x * x − 1))

Retorna o co-seno hiperbólico inverso de x (x > = 1).

ARCCOTH(x) = LOG((x + 1)/(x − 1))/2

Retorna a co-tangente hiperbólica inversa de x (ABS(x > 1).

ARCCSC(x) = ATN(1/SQR(x * x − 1)) + (SGN(x) − 1) * 1.5707633

Retorna a co-secante inversa de x (ABS(x) > 1).

ARCCSCH(x) = LOG((SGN(x) * SQR(x * x + 1) + 1)/x)

Retorna a co-secante hiperbólica inversa de x (x > 0).

ARCSEC(x) = ATN(SQR(x * x − 1)) + (SGN(x) − 1) * 1.5707633

Retorna a secante inversa de x (ABS(x) > = 1).

ARCSECH(x) = LOG((SQR(−x * x + 1) + 1)/x)

Retorna a secante hiperbólica inversa de x (0 < x < = 1).

ARCSIN(x) = ATN(x/SQR(−x * x + 1))

Retorna o seno inverso de x (ABS(x) < 1).

ARCSINH(x) = LOG(x + SQR(x * x + 1))

Retorna o seno hiperbólico inverso de x.

ARCTANH(x) = LOG((1 + x)/(1 − x))/2

Retorna a tangente hiperbólica inversa de x (ABS(x) < 1).

COSH(x) = (EXP(x) + EXP(−x))/2

Retorna o co-seno hiperbólico de x.

COT(x) = 1/TAN(x)

Retorna a co-tangente de x (x <> 0).

COTH(x) = EXP(−x)/(EXP(x) − EXP(−x)) * 2 + 1

Retorna a co-tangente hiperbólica de x (x <> 0).

CSC(x) = 1/SIN(x)

Retorna a co-secante de x (x <> 0).

CSCH(x) = 2/(EXP(x) − EXP(−x))

Retorna a co-secante hiperbólica de x (x <> 0).

$LOG_a(x)$ = LOG(x)/LOG(a)

Retorna o logaritmo de x na base a (a > 0, x > 0).

$LOG_{10}(x)$ = LOG(x)/2.30258509

Retorna o logaritmo comum (decimal) de x (x > 0).

MODa(x) = INT((x/a − INT((x/a)) * a + .05) * SGN(x/a)

Retorna x módulo a: o resto da divisão de x por a (a <> 0).

SEC(x) = 1/COS(x)

Retorna a secante de x (x <> π/2).

SECH(x) = 2(EXP(x) + EXP(−x))

Retorna a secante hiperbólica de x.

SINH(x) = (EXP(x) − EXP(−x))/2

Retorna o seno hiperbólico de x.

TANH(x) = −EXP(−x)/EXP(x) + EXP(−x)) * 2 + 1

Retorna a tangente hiperbólica de x.

B

Comandos de Edição

Este apêndice sumariza os comandos de edição do teclado do Apple II.

→	Move o cursor para a frente ao longo da linha de edição. Cada caractere passado é copiado para a memória como se fosse digitado no teclado. Não altera a tela.
←	Move o cursor para trás ao longo da linha de edição, limpando os caracteres pelos quais passa na memória, porém deixando-os na tela.
REPT	Causa a repetição da tecla acionada por outra tecla enquanto ambas estão acionadas. A tecla REPT deve ser acionada após a outra tecla.
CTRL-X	O Apple II desconsidera a linha digitada e move o cursor para a próxima linha, colocando-o na margem esquerda da mesma.

Seqüências da Tecla ESC

Os sete comandos de edição que seguem são seqüências de duas teclas. Em cada instância, acione a tecla ESC, solte-a, e então acione a segunda tecla da seqüência.

ESC-A	Move o cursor uma posição para a direita. Não altera a tela nem a memória.
ESC-B	Move o cursor uma linha para a esquerda. Não altera a tela nem a memória.
ESC-C	Move o cursor uma linha para baixo. Não altera a tela nem a memória.
ESC-D	Move o cursor uma linha para cima. Não altera a tela nem a memória.
ESC-E	Limpa todos os caracteres a partir da posição do cursor até o fim da linha.
ESC-F	Limpa todos os caracteres da posição do cursor até o fim da tela.
ESC-@	Limpa a tela e move o cursor para o canto superior esquerdo.

Comandos em Modo de Edição

Os quatro comandos de edição seguintes requerem o Monitor Autostart. Eles são comandos que só operam no modo de edição. Entre no modo de edição acionando a tecla ESC e deixe o modo de edição acionando qualquer tecla que não seja I, J, K, M, REPT, CTRL ou SHIFT.

I Move o cursor uma linha para cima sem sair da edição.

J Move o cursor uma posição para a esquerda sem sair da edição.

K Move o cursor uma linha para a direita sem sair da edição.

M Move o cursor uma linha para baixo sem sair da edição.

C

Mensagens de Erro

As mensagens de erro são agrupadas em três categorias: Integer BASIC, Applesoft e DOS, e são listadas alfabeticamente dentro de cada categoria.

As mensagens de erro do DOS e muitas do Applesoft têm códigos de erro associados. Após um erro ter causado um desvio com ONNER GOTO, o código para o erro pode ser obtido na posição de memória 222. A Tabela C-1, no final deste apêndice, lista as mensagens de erro pelo número de código.

MENSAGENS DE ERRO DO INTEGER BASIC

*** > 255 ERR
O valor que deveria estar entre 0 e 255 está fora dessa faixa.

*** > 32767 ERR
Foi calculado ou digitado um número maior que 32767 ou menor que –32767.

*** 16 FORS ERR
Foram ativados mais que 16 *loops* FOR.

*** 16 GOSUBS ERR
Foram executadas dezessete declarações GOSUB sem ter sido dado uma declaração RETURN.

*** BAD BRANCH ERR
Tentou-se um salto para uma linha inexistente.

*** BAD NEXT ERR
Encontrou-se um NEXT sem ter sido dado um FOR anteriormente.

*** BAD RETURN ERR
Executaram-se mais declarações RETURN que declarações GOSUB.

***DIM ERR
A mesma matriz foi dimensionada mais de uma vez.

*** MEM FULL ERR
O programa precisa de mais memória do que existe disponível.

*** NO END ERR
A última instrução dentro de um programa não é END.

*** RANGE ERR
Foi referenciada uma matriz com subscrito menor que zero ou maior que o seu tamanho, ou um argumento numa instrução HLIN, VLIN, PLOT, TAB ou VTAB foi colocado fora da faixa prescrita.

RETYPE LINE
Provocou-se um erro em resposta a uma declaração INPUT. A mensagem de diagnóstico foi mostrada primeiro e depois esta diretiva.

*** STRING ERR
Foi detectada uma operação ilegal com série

*** STR OVFL ERR
Foram atribuídos mais caracteres a uma série do que ele pode aceitar.

*** SYNTAX ERR
Ocorreu um erro de digitação, pontuação ou seqüência ou outro erro qualquer que não pode ser detectado por outra mensagem.

*** TOO LONG ERR
Foram aninhados mais de 12 parênteses ou mais de 128 caracteres dentro de uma linha.

MENSAGENS DE ERRO DO APPLESOFT

? BAD SUBSCRIPT ERROR
Foi referenciada uma matriz com número errado de subscritos ou com um ou mais subscritos excedentes em dimensão. Código de erro 107.

? CAN'T CONTINUE ERROR
Foi tentado continuar (com o comando CONT) quando não existe o programa, ou quando ocorreu um erro fatal, ou foram feitas mudanças no programa.

? DIVISION BY ZERO ERROR
Foi feita uma tentativa de dividir uma expressão por zero. Código de erro 133.

? FÓRMULA TOO COMPLEX ERROR
Foram executadas mais de duas declarações na forma IF série THEN. Código de erro 191.

? ILLEGAL DIRECT ERROR
Foi digitado um comando INPUT, DEF FN ou GET em modo imediato.

? ILLEGAL QUANTITY ERROR

Um valor numérico está fora da faixa aceitável para uma função série, com valor numérico, declaração gráfica etc. Código de erro 53.

? NEXT WITHOUT FOR ERROR

Foi executada uma função NEXT sem que antes tivesse sido dado o FOR correspondente. Um NEXT sem variável só gera erro se não existir nenhum FOR em atividade.

? OUT OF DATA ERROR

Foi tentado ler mais dados com DATA do que existe disponível. Código de erro 42.

? OUT OF MEMORY ERROR

Pode ser causado por: programa muito comprido, muitas variáveis, mais de 10 FOR aninhados, mais de 24 níveis de sub-rotinas aninhadas, mais de 36 níveis de parênteses aninhados, LOMEM: muito alto, ou HIMEM: muito baixo. Código de erro 77.

? OVERFLOW ERROR

Foi calculado ou entrado um número muito grande ou muito pequeno. A faixa permitida é aproximadamente $-1.7E+38$ até $1.7E+38$. Código de erro 69.

? REDIM'D ARRAY ERROR

Foi achada uma declaração DIM para uma matriz já dimensionada. Ocorre mais freqüentemente quando uma matriz foi dimensionada por ausência. Código de erro 120.

? RETURN WITHOUT GOSUB ERROR

Foram executadas mais declarações RETURN que GOSUB. Código de erro 22.

? STRING TOO LONG ERROR

Tentou-se concatenar séries com total de caracteres excedendo 255. Código de erro 176.

? SYNTAX ERROR

Um erro de digitação, pontuação ou seqüência, ou outro erro qualquer onde não coube outra mensagem de erro. Código de erro 16.

? TYPE MISMATCH ERROR

Uma expressão numérica ou variável foi usada onde deveria ser usada uma série, ou vice-versa. Também ocorre quando os dois lados de uma igualdade não são do mesmo tipo. Código de erro 163.

? UNDEF'D FUNCTION ERROR

Foi referenciada uma função definida por usuário que não foi definida. Código de erro 224.

? UNDEF'D STATEMENT ERROR

Foi tentado um salto para uma linha de programa inexistente. Código de erro 90.

MENSAGENS DE ERRO DO DOS

DISK FULL

Foi feita uma tentativa de colocar mais informação num disco do que ele pode suportar. Num disco cheio, esta mensagem pode ocorrer no lugar da informação de uma mensagem mais apropriada (ex. FILE NOT FOUND). Código de erro 9.

TABELA C.1. Códigos da Erro.

PEEK (222)	Descrição do Erro	Linguagem
0	NEXT sem FOR	Applesoft
1	Linguagem não disponível	DOS
2 ou 3	Erro de faixa	DOS
4	Protegido contra escrita	DOS
5	Fim de dados	DOS
6	Arquivo não encontrado	DOS
7	Volume não coincidente	DOS
8	Erro de E/S	DOS
9	Disco cheio	DOS
10	Arquivo travado	DOS
11	Erro de sintaxe	DOS
12	Não há *buffers* disponíveis	DOS
13	Tipo de arquivo não coincidente	DOS
14	Programa muito comprido	DOS
15	Comando não direto	DOS
16	Erro de sintaxe	Applesoft
22	RETURN sem GOSUB	Applesoft
42	Fora de DATA	Applesoft
53	Quantidade ilegal	Applesoft
69	Estouro	Applesoft
77	Fora da memória	Applesoft
90	Declaração não definida	Applesoft
107	Subscrito errado	Applesoft
120	Arranjo redimensionado	Applesoft
133	Divisão por zero	Applesoft
163	Tipo não coincidente	Applesoft
176	Série muito longa	Applesoft
191	Fórmula muito complexa	Applesoft
224	Função não definida	Applesoft
254	Resposta errada a INPUT	Applesoft
255	CTRL-C foi acionado	Applesoft

END OF DATA

Foi feita uma tentativa de ler uma parte de um arquivo de texto que nunca foi gravado. Código de erro 5.

FILE LOCKED

Foi feita uma tentativa de usar SAVE, BSAVE, WRITE, DELETE ou RENAME num arquivo travado. Código de erro 10.

FILE NOT FOUND

Fez-se referência a um arquivo inexistente no disco. Este erro só ocorre se o comando DOS que referenciou o arquivo não o criou no momento em que ele não foi achado. Código de erro 6.

FILE TYPE MISMATCH

Um comando DOS referenciou um arquivo que não é do tipo requerido. Os comandos LOAD, RUN e SAVE só podem ser usados com arquivos de programas. O comando CHAIN só pode ser usado com arquivos de programa em Integer BASIC. Os comandos OPEN, READ, WRITE, APPEND, POSITION e EXEC só podem ser usados com arquivos de texto. Os comandos BLOAD, BSAVE e BRUN só podem ser usados com arquivos binários. Código de erro 13.

I/O ERROR

Tentou-se insatisfatoriamente guardar ou obter informação de um disco. Algumas causas podem ser: a porta do drive está aberta, o disco não foi inicializado, não há disco no drive, ou o disco está com defeito. Código de erro 8.

LANGUAGE NOT AVAILABLE

Tentou-se trocar de linguagem com FP ou INT sem a linguagem desejada estar presente no disco ou na ROM, ou foi feita uma tentativa de carregar ou rodar um programa quando a linguagem do programa não era disponível. Código de erro 1.

NO BUFFERS AVAILABLE

Foi requerido um outro *buffer* quando todos os *buffers* disponíveis já estavam em uso. Código de erro 12.

NOT DIRECT COMMAND

Os comandos DOS seguintes só podem ser usados com o uso de declaração PRINT em modo programado: APPEND, OPEN, POSITION, READ e WRITE. Código de erro 15.

PROGRAM TOO LARGE

Um comando DOS tentou ler um arquivo do disco na memória do Apple II, e não achou memória suficiente para colocá-lo. Código de erro 14.

RANGE ERROR

Um parâmetro usado num comando DOS está fora da sua faixa especificada; por exemplo, o parâmetro D (drive) deve ser 1 ou 2. Código de erro 2 ou 3.

SYNTAX ERROR

Um comando DOS tem um erro de digitação, pontuação ou seqüência. Código de erro 11.

VOLUME MISMATCH

O parâmetro V (volume) num comando DOS não coincidiu com o parâmetro volume do disco acessado. Código de erro 7.

WRITE PROTECTED

Tentou-se usar SAVE, BSAVE ou WRITE com um arquivo protegido do disco. Código de erro 4.

D

Sub-rotinas Intrínsecas

As duas tabelas seguintes listam algumas sub-rotinas em linguagem de máquina disponíveis no Apple II sub-rotina. Ache o ponto de entrada listado na Tabela D.1 na primeira coluna da Tabela D.2 para detalhes sobre registradores afetados etc.

A Tabela D.2 lista as sub-rotinas por ordem de ponto de entrada. A terceira coluna mostra os registradores (se existirem) que devem conter valores específicos antes que a sub-rotina seja acessada. A quarta coluna mostra quais registradores são afetados na execução da sub-rotina.

Muitas destas sub-rotinas têm equivalentes em comandos BASIC, ou podem ser acessadas a partir do BASIC com uma única instrução CALL. Essas equivalências aparecem na Tabela D.2. Alguns dos comandos BASIC listados são disponíveis apenas em Applesoft; eles são sinalizados com um *A*.

Algumas sub-rotinas, entretanto, não têm equivalências em nenhuma versão do BASIC, e não podem ser executadas por um CALL porque um ou mais registradores devem ser carregados com valores específicos antes de sua execução. São requeridas técnicas diferentes para lidar com esse problema, de acordo com a versão do BASIC.

O Integer BASIC fornece uma solução mais simples. Primeiro execute um CALL−182 para colocar os valores correntes dos registradores na memória RAM. Então dê um POKE nos valores desejados nas posições de memória 69 para o registrador A, 70 para o registrador X e 71 para o registrador Y. Execute um CALL−193 para colocar esses valores nos registradores, e dê um CALL para a rotina que será executada em primeiro lugar.

Esta técnica não funciona em Applesoft. Deve-se, então escrever e executar uma sub-rotina em linguagem de máquina que carregue os registradores com os valores desejados e então execute um salto para o ponto de entrada da rotina desejada, em linguagem de máquina.

TABELA D.1. Referência a Sub-rotinas Intrínsecas por Função.

	Função	Ponto de Entrada
Gráficos de Baixa Resolução	Desenha um ponto gráfico em baixa resolução	$F800
	Desenha uma linha horizontal em gráfico de baixa resolução	$F819
	Desenha uma linha vertical em baixa resolução	$F828
	Coloca todas as 48 linhas gráficas de baixa resolução em preto (se em modo texto, coloca "@" invertido)	$F832
	Coloca as 40 linhas de cima de gráfico de baixa resolução em preto (ou "@" inverso)	$F836
	Incrementa a cor do gráfico de baixa resolução atual em três	$F85F
	Determina a cor do gráfico em baixa resolução	$F864
	Lê a cor de um ponto gráfico de baixa resolução	$F871
	Entra no modo gráfico de baixa resolução, limpa a tela e deixa as quatro linhas de baixo para janela de texto	$FB40
Entrada	Espera por acionamento de tecla com o cursor piscando, e cria o gerador de números aleatórios nas posições $4E e $4F	$FD1B
	O mesmo que acima à exceção que os códigos de escape também são permitidos	$FD35
	Manda *carriage return* para a tela, e permite entrada de uma linha inteira de até 256 caracteres	$FD67
Saída	Manda três brancos para o dispositivo selecionado corrente	$F948
	Manda de 1 a 256 brancos para o dispositivo selecionado corrente	$F94A
	Manda um *carriage return* e um *line feed* para a tela do Apple II	$FC62
	Manda um caractere para o dispositivo selecionado corrente	$FDED
	Manda um caractere para a janela de texto	$FDF0
Saída de Sino	Manda o caractere de sino (código ASCII 07) para o dispositivo selecionado corrente	$FBD9
	Apita no falante interno por 1/10 segundo	$FBE4
	Manda a mensagem ERR e apita o falante interno	$FF2D
	Apita o falante interno	$FF3A

TABELA D.1. (continuação)

	Função	Ponto de Entrada
Janela de Texto	Põe a tela do Apple II com 24 linhas e 40 colunas	$FB2F
	Pula uma linha na janela de texto	$FC70
Controle de Cursor	Manda um caractere backspace para a tela, atualizando a posição do cursor	$FC10
	Move o cursor uma linha para cima. Se o cursor já estiver em cima, ele não se moverá	$FC1A
	Move o cursor uma linha abaixo sem trocar sua posição horizontal. Toda a tela salta uma linha se o cursor já estiver na de baixo	$FC66
Limpeza da Tela	Limpa a janela de texto a partir de sua posição atual até o canto inferior direito da tela	$FC42
	Limpa a janela de texto das coordenadas passadas no registrador até o canto inferior direito da tela	$FC46
	Limpa toda a tela e move o cursor para o canto superior esquerdo	$FC58
	Limpa a linha atual da posição do cursor até o fim da linha	$FC9C
Modo Vídeo	Coloca a tela em vídeo inverso (preto-sobre-branco)	$FE80
	Coloca a tela em vídeo normal (branco-sobre-preto)	$FE84
Impressão de Conteúdo de Registradores	Imprime o conteúdo dos registradores Y e X (no formato YYXX) no dispositivo selecionado corrente	$F940
	Imprime o conteúdo dos registradores A e X (no formato AAXX) no dispositivo selecionado corrente	$F941
	Imprime o conteúdo do registrador X no dispositivo selecionado corrente	$F944
	Imprime o conteúdo do registrador A no dispositivo selecionado corrente	$FDDA
Move Conteúdos de Registradores	Recoloca o conteúdo dos registradores (válido apenas se a rotina intrínseca em $FF4A foi executada previamente)	$FF3F
	Salva os conteúdos dos registradores nas posições reservadas da página Zero	$FF4A
Diversos	Lê o estado de um controlador de jogo (paddle)	$FB1E
	Executa um *loop* de atraso	$FCA8
	Retorna para o BASIC, eliminando o programa e as variáveis da memória	$FEB0
	Ponto de entrada do Monitor	$FF69

TABELA D.2. Sub-rotinas Intrínsecas por Ponto de Entrada

Ponto de Entrada	Uso	Registradores a Serem Carregados Antes da Chamada	Registradores Afetados	Equivalentes BASIC
$F800	Desenha um ponto na página gráfica de baixa resolução 1	Coloque linha em A e coluna em Y	Nenhum	PLOT
$F819	Desenha uma linha horizontal em baixa resolução	Linha em A, coluna em Y, coluna da direita na posição de memória 44	A, Y	HLIN
$F828	Desenha uma linha vertical em baixa resolução	Coluna em Y, linha em A, linha mais baixa na posição de memória 45	Nenhum	VLIN
$F832	Coloca todas as 48 linhas gráficas de baixa resolução em preto (se estiver no modo texto, enche de caracteres "@")	Nenhum	A, Y	CALL-1998
$F836	Limpa as linhas gráficas de baixa resolução, deixando a janela de texto intacta	Nenhum	A, Y	GR (veja $FB40)
$F85F	Incrementa a cor do gráfico de baixa resolução atual em três	Nenhum	A	CALL-1953
$F864	Determina a cor gráfica de baixa resolução	Numéro da cor em A	A	COLOR
$F871	Lê a cor de um ponto gráfico de baixa resolução	Linha em A, coluna em Y	A (contém número da cor)	SCRN
$F940	Imprime os conteúdos dos registradores Y e X (no formato YYXX) na tela ou em outro dispositivo selecionado	Nenhum	Nenhum	CALL-1728
$F941	Imprime os registradores (AAXX) como acima	Nenhum	Nenhum	CALL-1727
$F944	Imprime o conteúdo do registrador X	Nenhum	Nenhum	CALL-1724
$F948	Manda três brancos para o dispositivo selecionado corrente (determinado pelo conteúdo de CSW)	Nenhum	Nenhum	CALL-1720
$F94A	Manda de 1 a 256 caracteres brancos para o dispositivo selecionado corrente	Número de espaços em branco em A (carregando 0 imprime 256 caracteres)	Nenhum	SPC( )[A] CALL-1718
$FB1E	Lê o estado do *paddle* 0, 1, 2 ou 3	Número do *paddle* em X	0 a FF no registrador Y, e conteúdo de A destruído	PDL( )

TABELA D.2. (continuação)

Ponto de Entrada	Uso	Registradores a Serem Carregados Antes da Chamada	Registradores Afetados	Equivalentes BASIC
$FB2F	Formada a tela do Apple II para 40 linhas de 24 colunas	Nenhum	A	TEXT
$FB40	Formada a tela para gráfico de baixa resolução, limpa a tela e coloca 4 linhas para janela de texto	Nenhum	A, Y	GR
$FBD9	Manda o caractere BELL (código ASCII 07) para o dispositivo selecionado corrente	Nenhum	A, Y	CALL-1063
$FBE4	Apita o falante do Apple II por 1/10 segundo	Nenhum	A, Y	CALL-1052
$FC10	Manda um caractere backspace para a tela, atualizando a posição do cursor	Nenhum	A	CALL-1008
$FC1A	Move o cursor uma linha para cima. Se ele já estiver na linha superior, nada acontece	Nenhum	A	CALL-998
$FC42	Limpa a janela de texto da posição atual do cursor até o canto inferior direto da tela	Nenhum	A, Y	CALL-958
$FC46	Limpa a janela de texto a partir das coordenadas passadas nos registradores até o canto inferior direito	Coluna em Y, linha em A	A, Y	CALL-954
$FC58	Limpa toda a tela de texto e move o cursor para o canto superior esquerdo	Nenhum	A, Y	HOME[A] CALL-936
$FC62	Manda um *carriage return* e um *line feed* para a tela do Apple II	Nenhum		CALL-926
$FC66	Move o cursor uma linha para baixo sem alterar sua posição horizontal. Pula toda a tela uma linha para cima se estiver na linha de baixo	Nenhum	A, Y	CALL-922
$FC70	Pula toda a tela uma linha para cima	Nenhum	A, Y	CALL-912
$FC9C	Limpa o texto a partir da posição atual do cursor até o fim da linha. A posição do cursor permanece inalterada	Nenhum	A, Y	CALL-868

TABELA D.2. (continuação)

Ponto de Entrada	Uso	Registradores a Serem Carregados Antes da Chamada	Registradores Afetados	Equivalentes BASIC
$FCA8	Executa um *loop* de retardo que tem $0.5(5x^2 + 27x + 26)$ de duração	Valor do atraso *x* em A	A	CALL-856
$FDOC	Espera por acionamento de uma tecla. Gera semente de número aleatório nas posições de memória 78 e 79	Nenhum	Caractere retorna em A. X e Y	CALL-756
$FD35	Como $FDOC, exceto que o código escape não é permitido	Nenhum	Caractere retorna em A X e Y	CALL-715
$FD67	Manda *carriage return* para a tela; permite entrada de uma linha inteira de dados, até 256 caracteres	Caractere de entrada na posição 51	Y, A X contém o tamanho da entrada. Os dados entrados iniciam na posição $200	INPUT
$FDDA	Imprime o valor do Acumulador como dois dígitos hexadecimais	Dado em A	A	CALL-550
$FDED	Manda um caractere para o dispositivo selecionado corrente	Caractere em A	Nenhum	PRINT
$FDF0	Manda um caractere para a janela de texto do Apple II	Caractere em A	Nenhum	PRINT
$FE80	Coloca o vídeo em modo reverso (texto preto-sobre-branco)	Nenhum	Y	INVERSE[A] CALL-384
$FE84	Coloca o vídeo em modo normal (texto branco-sobre-preto)	Nenhum	Y	NORMAL[A] CALL-380
$FEB0	Retorna ao BASIC, eliminando o programa e as variáveis da memória	Nenhum	A, X, Y	CALL-336
$FF2D	Imprime a mensagem ERR e apita o falante interno	Nenhum	A	CALL-211
$FF3A	Apita o falante interno	Nenhum	A	CALL-198
$FF3F	Recupera o conteúdo dos registradores (só válido se a sub-rotina intrínseca de $FF4A foi executada previamente)	Nenhum	O conteúdo dos registradores é tirado das seguintes posições: registrador A: 69($45), registrador S: 72($48), registrador X: 70($46), Apontador de Pilha: 73($49), registrador Y: 71($47)	CALL-193

TABELA D.2. (continuação)

Ponto de Entrada	Uso	Registradores a Serem Carregados Antes da Chamada	Registradores Afetados	Equivalentes BASIC
$FF4A	Salva os conteúdos dos registradores nas posições da Página reservada 0: registrador A: 69($45), registrador S: 72($48), registrador X: 70($46), Ponteiro de Pilha: 73($49), registrador Y: 71($47)	Nenhum	Nenhum	CALL-182
$FF69	Ponto de entrada do Monitor	Nenhum	Nenhum	CALL-151

A denota comandos disponíveis apenas em Applesoft.

E

Posições Úteis de Peek e Poke

Cada uma das posições e memória listadas abaixo é expressa em termos de um número decimal menor que 32767 em módulo. As posições de memória acima de 32767 são expressas em termos de um número negativo. Existe um número positivo que se refere à mesma posição de memória. Some 65536 ao número listado para obter o positivo equivalente (exemplo, 65535 − 16384 = 49152).

Algumas das funções descritas abaixo são atuadas com o seu acesso. Quer dizer que a qualquer momento em que um PEEK acessar uma posição específica de memória, a operação especificada tem início. Um POKE para a posição específica de memória também dispara a ação, mas devido a características do microprocessador do Apple II, uma declaração POKE na verdade dispara a ação por duas vezes. Neste caso, POKE equivale a duas declarações PEEK. Normalmente isso não faz diferença, mas em casos como −16336 (apita o alto-falante) faz. O valor colocado na memória pelo POKE é irrelevante nessas atuações sobre endereços.

JANELAS DE TEXTO E POSIÇÕES DE CONTROLE DE CURSOR

32 Margem Esquerda da Janela de Texto

Especifica a coluna da margem esquerda da janela de texto. Um PEEK retorna um valor entre 0 e 39, como sendo o lado esquerdo da tela. A troca dessa posição não afeta a largura da janela; ambas as margens, esquerda e direita, se movem.

Usando um valor maior que 39 no POKE, ou se o valor da posição mais a largura da janela de texto exceder 40, alguns ou mesmo todos os dados mandados para a tela serão colocados fora de sua área de memória. Isso pode destruir o programa ou mesmo algum dado essencial.

33 Largura da Janela de Texto

Especifica a largura da janela de texto. O valor desta posição deve estar entre 1 e 40. A troca desse valor muda a margem direita para a coluna indicada, onde está especifi-

cado o número de caracteres apontado à frente da margem esquerda (posição de memória 32).

Um valor zero nesta posição (ou seja, largura zero) pode destruir o interpretador BASIC. Colocando-se um valor maior que 40 nesta posição com POKE, ou se o valor deste local mais a posição 32 (margem esquerda) exceder 40, algum ou todo o dado mandado para a tela cairá fora de sua área de memória. Isso poderá destruir o programa ou dados importantes.

34 Margem Superior da Janela de Texto

Especifica a margem superior da janela de texto. O valor desta posição deve estar na faixa de 0 até 23; 0 especifica a linha superior da tela, 23 a inferior. Colocando-se um valor maior que 23 nessa posição com POKE, algum ou todo o dado mandado para a tela irá para áreas de memória fora da área de tela, mexendo com outros dados que podem ser importantes. Não coloque a margem superior do texto abaixo da margem inferior.

35 Margem Inferior da Janela de Texto

Especifica a margem inferior da janela de texto. O valor desta posição deve estar na faixa entre 0 e 23; 0 especifica a linha superior da tela e 23 a inferior. Colocando um valor maior que 23 nessa posição, algum ou todo o dado mandado para a tela poderá cair fora da área de memória de vídeo, alterando dados que podem ser importantes. Não coloque a margem inferior da janela de texto acima da margem superior.

36 Posição Horizontal do Cursor

Especifica a posição horizontal corrente do cursor. Um PEEK retorna um valor entre 0 e 39 que especifica a posição do cursor em relação à margem esquerda da janela de texto (não necessariamente a margem esquerda da tela). Esta posição pode ser usada para posicionar além da margem direita (e depois imprimi-la com PRINT), mas o cursor fica lá apenas o necessário para imprimir um caractere. Não coloque um valor nesta posição que somado ao da margem esquerda da tela (posição 32) exceda 39.

Este PEEK é equivalente à função POS do Applesoft.

37 Posição Vertical do Cursor

Especifica a posição vertical corrente do cursor. Um PEEK retorna um valor entre 0 e 23, em relação à linha superior da tela (não em relação à linha superior da janela de texto). Não coloque um valor maior que 23 nesta posição.

POSIÇÕES DE TRATAMENTO DE ERRO

216 Indicação de Erro

Indica quando um ONNER GOTO está atuando. Se o bit 7 desta posição de memória é 1 (ou seja, se a posição tem um valor 128 ou mais), uma declaração ONNER GOTO foi encontrada, e o controle passará para a linha de número especificado no caso de erro. Coloque um valor menor que 128 na posição com POKE para inibir a operação de ONNER GOTO.

218 e 219 Número de Linha Acessada por Erro

Quando um erro provoca um salto de acordo com uma declaração ONNER GOTO, essas posições especificam o número da linha na qual o erro ocorreu. Este número de linha é PEEK(219) * 256 + PEEK(218).

222 Código de Tipo de Erro
Especifica que tipo de erro ocorreu. Os códigos de erro e suas especificações são dados no Apêndice C.

POSIÇÕES DE TECLADO

– 16384 Caractere do Teclado
Lê o teclado. Se o valor desta posição é maior que 127 (ou seja, se o bit 7 está em 1), uma tecla foi acionada. Determine o código ASCII equivalente subtraindo 128 desse valor.

– 13368 Indicação de Teclado
Desliga o inibidor de teclado (bit 7 da posição – 16384), colocando-o em zero de forma que o próximo caractere possa ser lido.

POSIÇÕES DE SAÍDA DE "CLICK"

– 16352 Click do cassette
Gera um click auditivo no conector do cassette.

– 16336 Click do falante
Gera um click no alto-falante interno.

CHAVES DE DISPLAY

As posições de memória listadas nesta seção ativam certas chaves que determinam as características de display. Elas não são chaves reais; apenas os comandos PEEK e POKE afetam essas posições.

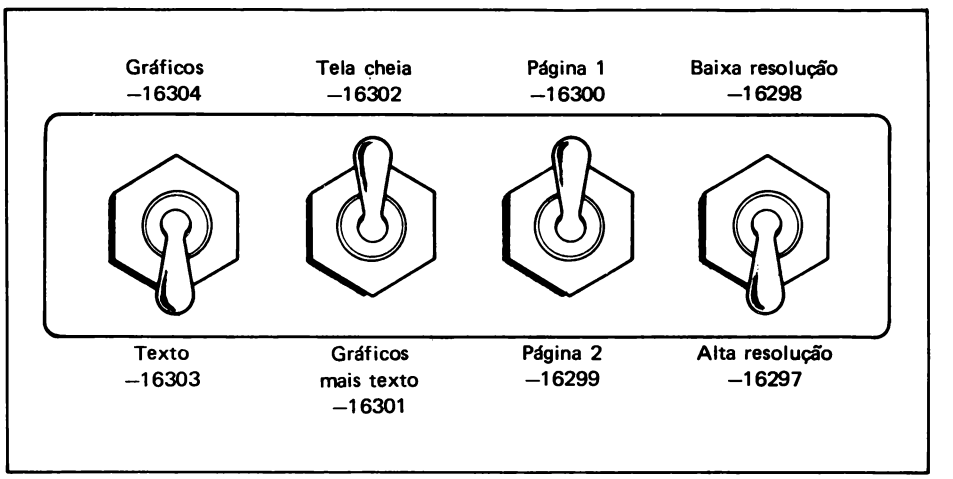


FIGURA E.1. Posições de Texto e Gráficos para PEEK e POKE.

Existem quatro chaves, e cada uma delas pode ser colocada em duas posições diferentes, como mostrado na Figura E.1. Como modo texto selecionado, a única chave que tem efeito é a de Página 1/Página 2.

— 16304 Seleção de Modo Gráfico
Seleciona o modo gráfico. A tela gráfica não é colocada em preto. O modo gráfico pode ser de alta ou baixa resolução, página 1 ou 2, gráfico de tela cheia ou misto de gráfico e texto. Essas características são determinadas por outras posições de memória.

— 16303 Seleção de Modo Texto
Seleciona modo texto. O texto pode ser na página 1 ou 2, o que é determinado por outras posições de memória.

— 16302 Seleciona Gráficos em Tela Cheia
Seleciona gráficos em tela cheia. Se a tela estiver em modo texto, isto não será visível até que a posição — 16304 seja acessada.

- 16301 Seleciona Gráficos mais Texto
Determina quatro linhas para janela de texto na parte de baixo da tela. Se a tela estiver no modo texto, isto não será visível até que a posição — 16304 seja acessada.

— 16300 Seleciona Página de Tela 1
Seleciona página 1 de gráficos ou texto.

— 16299 Seleciona Página de Tela 2
Seleciona página 2 de gráficos ou texto.

— 16298 Seleciona Gráficos em Baixa Resolução
Seleciona gráficos em baixa resolução. Se a tela estiver em modo texto, isto não será visível até que a posição — 16304 seja acessada.

— 16297 Seleciona Gráficos em Alta Resolução
Seleciona gráficos em alta resolução. Se a tela estiver em modo texto, isto não será visível até que acesse a posição — 16304.

POSIÇÕES DE CONTROLE DE JOGOS

As posições de memória nesta seção ligam e desligam os controles de saída de jogos, sentem se um botão foi acionado ou não, e atuam um liberador de saída. A Figura E.2 mostra como as saídas dos controladores de jogos são manipuladas.

Todas as entradas e saídas para estas declarações PEEK e POKE ligadas ao conector de controle de jogos são mostradas na Figura E.3.

— 16296 Anunciador 0 Desligado
Desliga a saída do controlador de jogo (anunciador) número 0. A tensão no pino 15 do conector do controlador é colocada em aproximadamente 0 volts (nível TTL alto).

— 16295 Anunciador 0 Ligado
Liga a saída do controlador de jogo (anunciador) número 0. A tensão do pino 15 do conector do controlador é colocada em aproximadamente +5 volts (nível TTL baixo).

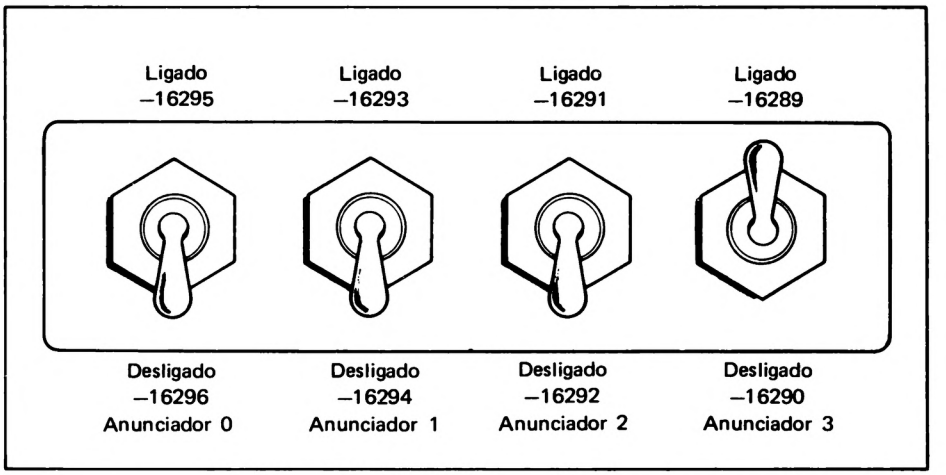


FIGURA E.2. Manipulação de Saídas de Controlador de Jogo (Anunciadores).

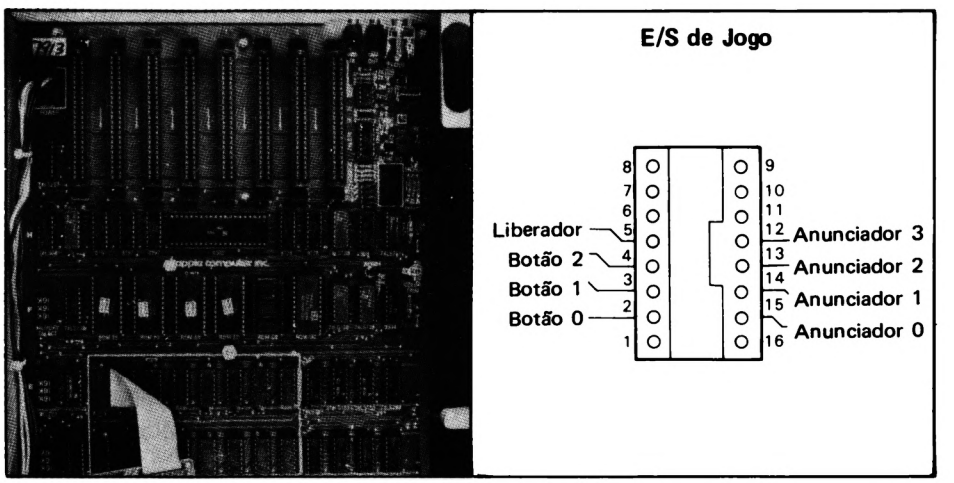


FIGURA E.3. Entradas e Saídas do Controlador de Jogo.

– 16294 Anunciador 1 Desligado

Desliga a saída do controlador de jogo (anunciador) número 1. A tensão no pino 15 do conector do controlador é colocada em aproximadamente 0 volts (nível TTL alto).

– 16293 Anunciador 1 Ligado

Liga a saída do controlador de jogo (anunciador) número 1. A tensão no pino 15 do conector do controlador é colocada em aproximadamente +5 volts (nível TTL baixo).

– 16292 Anunciador 2 Desligado

Desliga a saída do controlador de jogo (anunciador) número 2. A tensão no pino 15 do conector do controlador é colocada em aproximadamente 0 volts (nível TTL alto).

– 16291 Anunciador 2 Ligado

Liga a saída do controlador de jogo (anunciador) número 2. A tensão no pino 15 do conector do controlador é colocada em aproximadamente +5 volts (nível TTL baixo).

– 16290 Anunciador 3 Desligado

Desliga a saída do controlador de jogo (anunciador) número 3. A tensão no pino 15 do conector do controlador é colocada em aproximadamente 0 volts (nível TTL alto).

– 16289 Anunciador 3 Ligado

Liga a saída do controlador de jogo (anunciador) número 3. A tensão no pino 15 do conector do controlador é colocada em aproximadamente +5 volts (nível TTL baixo).

– 16287 Ler Botão 0

Quando o botão do controlador de jogo número 0 é acionado, o valor desta posição excede 127; quando não é acionado, o valor é 127 ou menos. O botão 0 é conectado ao pino 2 do conector do controlador de jogo.

– 16286 Ler Botão 1

Quando o botão do controlador de jogo número 1 é acionado, o valor desta posição excede 127; quando não é acionado, o valor é 127 ou menos. O botão 1 é conectado ao pino 3 do conector do controlador de jogo.

– 16285 Ler Botão 2

Quando o botão do controlador de jogo número 2 é acionado, o valor desta posição excede 127; quando não é acionado, o valor é 127 ou menos. O botão 2 é conectado ao pino 4 do conector do controlador de jogo.

– 16272 Liberador de Saída

Normalmente o pino 5 do conector do controlador de jogo está em +5 volts. Se for dado um PEEK na posição de memória – 16285, ele pula para 0 volts por um e meio microssegundo. Um POKE aciona o liberador por duas vezes.

F

Palavras Reservadas do Basic

O Apple II interpreta toda ocorrência das seguintes palavras reservadas como comandos, declarações ou funções do BASIC. A única exceção é quando elas são parte de uma série de texto entre aspas. Não use estas palavras reservadas como variáveis. Cuidado principalmente com as palavras reservadas pequenas.

Pode-se digitar as palavras intervaladas de espaços; o Apple II os despreza.

INTEGER BASIC

ABS	END	LET	PDL	SAVE
AND	FOR	LIST	PEEK	SCRN
ASC	GOSUB	LOAD	PLOT	SGN
AT	GOTO	LOMEM:	POKE	STEP
AUTO	GR	MAN	POP	TAB
CALL	HIMEM:	MOD	PRINT	TEXT
COLOR=	HLIN	NEW	PR#	THEN
CON	IF	NEXT	REM	TO
DEL	IN#	NOT	RETURN	TRACE
DIM	INPUT	NOTRACE	RND	VLIN
DSP	LEN	OR	RUN	VTAB

APPLESOFT

As palavras reservadas em Applesoft são codificadas; cada palavra usa apenas um byte de armazenamento do programa. Os códigos são listados em conjunto com suas palavras reservadas. Eles também são listados na ordem numérica no Apêndice I.

O Applesoft não reconhece de acordo a palavra reservada TO se:

1. O primeiro caractere diferente de branco antes do TO seja a letra A, e
2. Um ou mais brancos separem o T do O.

ABS	(212)	HTAB	(150)	REM	(178)
AND	(205)	IF	(173)	RESTORE	(174)
ASC	(230)	IN#	(139)	RESUME	(166)
AT	(197)	INPUT	(132)	RETURN	(177)
ATN	(225)	INT	(211)	RIGHT$	(233)
CALL	(140)	INVERSE	(158)	RND	(219)
CHR$	(231)	LEFT$	(232)	ROT=	(152)
CLEAR	(189)	LEN	(227)	RUN	(172)
COLOR=	(160)	LET	(170)	SAVE	(183)
CONT	(187)	LIST	(188)	SCALE=	(153)
COS	(222)	LOAD	(182)	SCRN(	(215)
DATA	(131)	LOG	(220)	SGN	(210)
DEF	(184)	LOMEM:	(164)	SHLOAD	(154)
DEL	(133)	MID$	(234)	SIN	(223)
DIM	(134)	NEW	(191)	SPC(	(195)
END	(128)	NEXT	(130)	SPEED=	(169)
EXP	(221)	NORMAL	(157)	SQR	(218)
FLASH	(159)	NOT	(198)	STEP	(199)
FN	(194)	NOTRACE	(156)	STOP	(179)
FOR	(129)	ON	(180)	STORE	(168)
FRE	(214)	ONERR	(165)	STR$	(228)
GET	(190)	OR	(206)	TAB(	(192)
GOSUB	(176)	PDL	(216)	TAN	(224)
GOTO	(171)	PEEK	(226)	TEXT	(137)
GR	(136)	PLOT	(141)	THEN	(196)
HCOLOR=	(146)	POKE	(185)	TO	(193)
HGR	(145)	POP	(161)	TRACE	(155)
HGR2	(144)	POS	(217)	USR	(213)
HIMEM:	(163)	PRINT	(186)	VAL	(229)
HLIN	(142)	PR#	(138)	VLIN	(143)
HOME	(151)	READ	(135)	VTAB	(162)
HPLOT	(147)	RECALL	(167)	WAIT	(181)
				XDRAW	(149)

DOS

Os comandos do DOS só são considerados como palavras reservadas se usados em modo imediato ou numa declaração PRINT que inicie com o caractere CTRL-D (código ASCII 04).

APPEND	CLOSE	LOCK	POSITION	UNLOCK
BLOAD	DELETE	MAXFILES	READ	VERIFY
BRUN	EXEC	MON	RENAME	WRITE
BSAVE	INIT	NOMON	RUN	
CHAIN	LOAD	OPEN	SAVE	

G

Uso da Memória

ORGANIZAÇÃO GERAL DA MEMÓRIA

A memória do Apple II é dividida em três categorias gerais: memória de leitura e gravação (também chamada de memória de acesso aleatório ou RAM), memória apenas de leitura (ROM) e posições de entrada/saída (E/S). As posições de memória de 0 a 49151 ($BFFF hexadecimal) são de RAM, as posições de 49152 ($C000) a 53247 ($CFFF) são em ROM. Seu sistema não precisa necessariamente ter todas essas posições. Por exemplo, tendo apenas 16K de RAM, as posições de 16384 ($4000) até 49151 ($BFFF) não são usadas.

A Tabela G.1 mostra como a memória é alocada no sistema do Apple II. Veja que existem dois blocos de posições de memória livres, entre as duas páginas gráficas de alta resolução. O apontador do sistema LOMEM: mantém o controle do limite inferior desta área livre, e o apontador do sistema HIMEM: marca o limite superior. A memória RAM pode ser usada para várias coisas. Entre elas está o interpretador Applesoft não firmware (do cassette ou disco), o Sistema Operacional em Disco (DOS), gráficos em alta resolução e seu programa e variáveis em BASIC.

OS INTERPRETADORES DE LINGUAGEM BASIC

Pode-se ver na Tabela G.1 que o interpretador Integer BASIC sempre reside em ROM. O interpretador Applesoft também reside em ROM se o sistema tem o cartão Applesoft Firmware ou o cartão Language System instalado. De outra forma, o interpretador Applesoft ocupa aproximadamente 10K bytes de memória iniciando em 2048 ($800).

TABELA G.1. Organização da Memória no BASIC.

Posição		Tipo de Memória	Uso
Decimal	Hex		
0-255	$0-$0FF	RAM	Programas do sistema
256-511	$100-$1FF	RAM	Pilha do sistema
512-767	$200-$2FF	RAM	Buffer de entrada do teclado
768-1023	$300-$3FF	RAM	Posições dos vetores do Monitor
1024-2047	$400-$7FF	RAM	Página 1 de texto e gráficos em baixa resolução
2048-3071	$800-$BFF	RAM	Página 2 de texto e gráficos em baixa resolução
3072-8191	$C00-$1FFF	RAM	Livre
8192-16383	$2000-$3FFF	RAM	Página gráfica de alta resolução 1
16384-24575	$4000-$5FFF	RAM	Página gráfica de alta resolução 2
24576-49151	$6000-$BFFF	RAM	Livre
49152-49279	$C000-$C07F	E/S	Posições internas especiais
49280-49407	$C080-$C0FF	E/S	Espaço dos cartões periféricos de E/S
49408-51199	$C100-$C7FF	E/S	Memória dos cartões periféricos
51200-53247	$C800-$CFFF	E/S	Expansão de memória dos cartões periféricos
53248-65535	$D000-$FFFF	ROM	Integer BASIC, Applesoft, o Monitor ou Autostart Monitor etc.

REQUISITOS DE MEMÓRIA DO DOS

Precisa-se no mínimo de 16K de memória para usar o DOS. Quando carregado, o DOS usa aproximadamente os 10K de cima da memória. HIMEM: é alterado para indicar imediatamente abaixo da memória usada pelo DOS. A Figura G.1 mostra como cada parte da memória é usada pelo

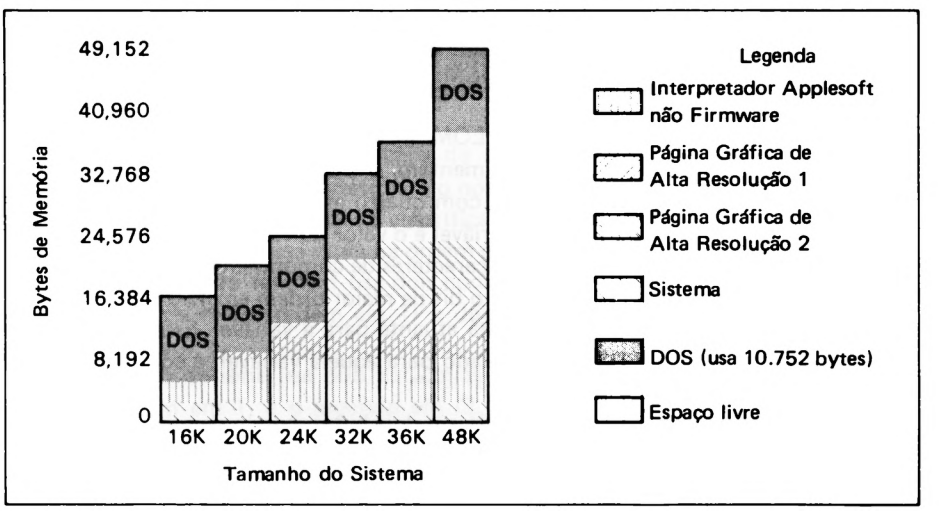


FIGURA G.1. Uso da Memória RAM.

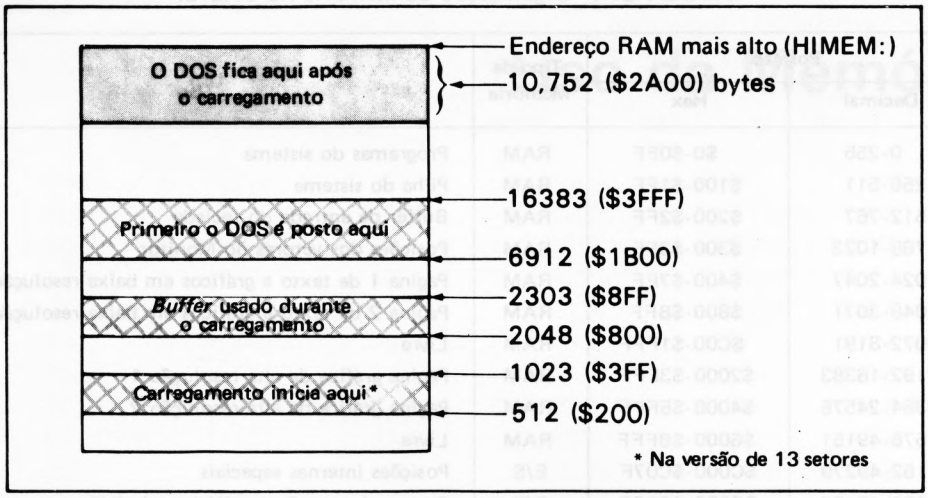


FIGURA G.2. Uso da Memória Durante o Carregamento.

DOS em vários tamanhos de sistemas. Veja também que se deve ter no mínimo 24K de RAM para suportar o DOS junto com Applesoft em disco (ou cassette), e no mínimo 32K para usar a página 1 de gráfico em alta resolução com o DOS. A Figura G.1 mostra claramente o conflito existente entre o Applesoft em disco ou cassette e a página gráfica de alta resolução 1.

O DOS usa várias partes adicionais da memória enquanto está sendo carregado (veja na Figura G.2). Qualquer coisa presente nessas áreas antes do carregamento estará perdida depois dele.

USO DE MEMÓRIA DO INTEGER BASIC

Os programas em Integer BASIC residem na parte mais alta da memória RAM, iniciando em HIMEM:. Como mostrado na Figura G.3, HIMEM: é ajustado automaticamente quando se adiciona, altera ou elimina linhas de programa.

As variáveis são guardadas iniciando em LOMEM: e locadas para cima. Na medida em que os requisitos de armazenamento de variáveis aumentam, o LOMEM: é ajustado automaticamente. Cada variável numérica é mapeada na memória com quatro atributos: nome, o byte DSP (ligado/desligado), a posição de memória da próxima variável e o valor ou valores da variável.

O nome de uma variável pode ter até 100 caracteres de comprimento. Cada caractere é representado na memória por um código ASCII, com o bit de maior ordem em 1.

O byte DSP indica se o comando DSP está atuante para a variável. Este byte, normalmente em 0, é colocado em 1 quando o DSP é executado para aquela variável, e colocado em 0 quando NO DSP é executado.

O endereço da próxima variável é guardado em dois bytes, com o de menor ordem primeiro.

Os dados são guardados em pares de bytes, com o byte de menor ordem primeiro. Se a variável não é uma matriz, existe apenas um desses pares. Se ela é matriz, existe um par por elemento, listados na ordem iniciando com elemento zero. Não existe distinção entre uma variável simples e uma matriz com o mesmo nome; a variável simples é o elemento zero de uma matriz.

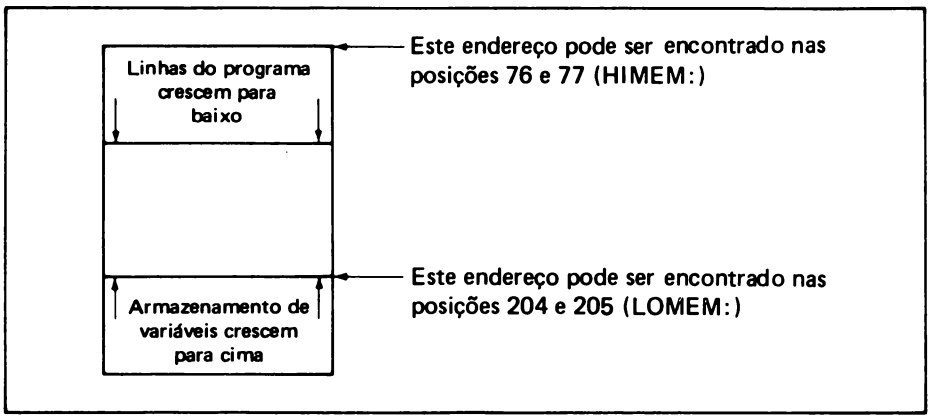


FIGURA G.3. Mapa de Memória de Programa em Integer BASIC.

As variáveis série são guardadas de forma semelhante. O nome da variável, o byte DSP, e o endereço da próxima variável são guardados da mesma forma que os valores numéricos. O código ASCII para cada caractere da série usa um byte, com o bit de maior ordem colocado em 1. O último caractere da série é seguido por um byte terminador de série, no qual o bit de maior ordem é 0.

USO DE MEMÓRIA NO APPLESOFT

As linhas de programa do Applesoft ocupam a parte inferior da memória RAM livre iniciando em LOMEM:, como mostrado na Figura G.4. Quando se adiciona, altera ou elimina as linhas de programa, LOMEM: é ajustado automaticamente. As variáveis numéricas simples e ponteiros de série são guardados diretamente acima das linhas de programa. Matrizes e ponteiros de matrizes tipo série são guardados acima das variáveis simples. Os valores de séries são guardados no alto da memória, iniciando em HIMEM:. Usando variáveis tipo série, HIMEM: ajusta-se automaticamente para baixo.

Cada variável numérica "e" ponteiro de série usa sete bytes de memória. Cada variável real usa dois códigos ASCII (dois bytes) para o nome da variável (ambos com o bit de maior ordem em 1). O valor é guardado em notação científica com um byte para o expoente e quatro para a mantissa. Esses bytes da mantissa estão na ordem do mais significativo para o menos.

Cada variável inteira também usa dois códigos ASCII (dois bytes) para o nome da variável (ambos com o bit de maior ordem em 1) e dois bytes para o valor da variável, com o byte de maior ordem primeiro.

Cada ponteiro de série usa dois caracteres ASCII (dois bytes) para o nome da variável (o bit de maior ordem em 1, e o segundo em 0), um byte para o comprimento da série, e dois bytes para o endereço do valor da série, byte de menor ordem primeiro. Os últimos três bytes de uma variável inteira e os dois últimos de um ponteiro de série não são usados.

Matrizes numéricas e matrizes de ponteiros de séries são guardadas imediatamente acima das variáveis. O nome da variável é guardado como um código ASCII nos dois primeiros

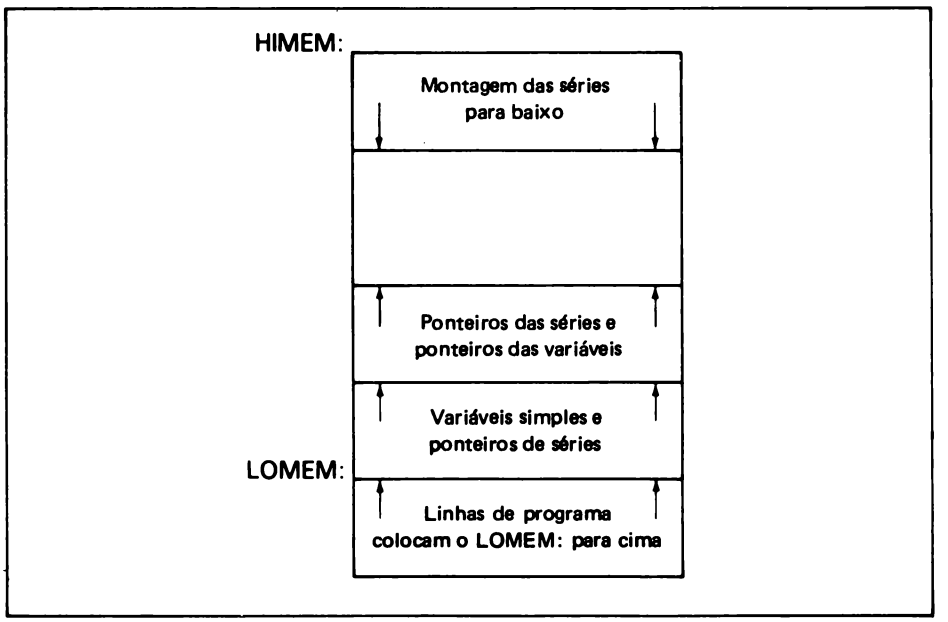


FIGURA G.4. Mapa de Memória para Programas em Applesoft.

bytes; ambos os bits de maior ordem para variáveis reais são 0, ambos são 1 para variáveis inteiras, e o primeiro é 1 e o segundo 0 para ponteiros de séries.

O nome da variável é seguido por dois bytes que indicam a posição da próxima variável. Isso é dado em relação ao primeiro byte do nome da variável, byte de menor ordem primeiro. A seguir vem um byte para o número de dimensões, e dois bytes para cada dimensão (maior ordem primeiro) indicando o tamanho de cada dimensão. Os tamanhos são listados em ordem inversa, ou seja, o tamanho da última dimensão é dado primeiro.

Cada elemento de uma matriz é então listado, do elemento (0, 0, ..., 0) ao elemento (N, N, ..., N). Os elementos são guardados na ordem, com o índice mais da esquerda incrementado primeiro. Cada elemento de uma matriz real usa cinco bytes, um para o expoente, e quatro (mais significativo primeiro) para a mantissa. Cada elemento inteiro usa dois bytes, maior ordem primeiro. Cada elemento ponteiro de série usa três bytes, um para o comprimento da série e dois (menor ordem primeiro) para o endereço da série.

Os valores das séries são guardados na parte mais alta da memória RAM. Eles requerem um byte de memória para cada caractere. As séries duplicadas são guardadas apenas uma vez; dois ou mais ponteiros de série podem apontar uma posição. À medida que novas séries vão sendo criadas, são colocadas nas próximas posições disponíveis (HIMEM: ajustado para baixo). As séries que não estão sendo usadas ficam na memória. A função FRE elimina a "coleção de detritos" eliminando todas as séries abandonadas e reajustando o HIMEM:.

H

Formato do Disk II

A informação é guardada no disquette em 35 bandas concêntricas chamadas trilhas. Essas trilhas são numeradas de 0 a 34 ($0 até $22). Cada trilha é dividida em 16 segmentos chamados *setores*, numerados de 0 até 15 ($0 até $F). Cada setor pode guardar até 256 bytes de dados. O DOS reserva três trilhas (48 setores) para ele mesmo, e uma trilha (16 setores) para o diretório do disquette. Isso deixa 31 trilhas (496 setores) para arquivos de dados e de programas.

O DOS transfere dados de e para o disco um setor por vez. Ele usa dois *buffers* de arquivo na memória, um para leitura e outro para gravação, para cada arquivo ativo.

Cada tipo de texto (texto, programa e binário) tem seu próprio formato no disco. Os textos são gravados em códigos ASCII, um byte por caractere. Um byte zero marca o fim do arquivo. Todos os bytes de um arquivo de texto são interpretados como texto.

Os primeiros dois bytes do primeiro setor de um arquivo de programa BASIC indica o comprimento do programa, com o byte de menor ordem primeiro. O resto do arquivo contém o programa em código ASCII. Num arquivo em Applesoft, as palavras reservadas são codificadas ao invés de guardadas inteiras (um código é um byte em código ASCII). Veja no Apêndice F a lista de códigos em ordem alfabética e no Apêndice I a lista em ordem numérica.

Os primeiros dois bytes do primeiro setor de um arquivo binário mostra o endereço inicial dos dados binários na memória RAM, com o byte de menor ordem primeiro. Os dois bytes seguintes dão o comprimento do arquivo, com o byte de menor ordem primeiro. O resto do arquivo contém os dados binários.

A LISTA DE TRILHA/SETOR

O DOS normalmente grava no disco onde quer que ele ache um setor livre. Isso quer dizer que num arquivo de muitos setores, o arquivo fica espalhado entre várias trilhas. O DOS faz uma lista de números de trilhas e setores usados para cada arquivo e a guarda num ou mais setores adicionais no disquette. Esta é chamada lista de trilha/setor.

Cada setor da lista contém um ponteiro para o próximo setor da lista (se existir) e aponta para até 122 setores que o arquivo pode conter.

Um setor de dados não usado não é gravado no disquette. Em seu lugar, o ponteiro para ele na lista de trilha/setor é 0, isso diz ao DOS para não procurar dados naquele setor. Além disso, o último setor da lista é aquele com o ponteiro para o último setor de dados. Assim, se o registro número 5000 é o primeiro e único registro num arquivo de acesso aleatório que tem o parâmetro L de 256 (um registro por setor), 42 setores serão usados. O dado no registro 5000 usa um setor. A lista de trilha/setor usa os outros 41 setores (122 ponteiros por setor vezes 41 setores é igual a 5002 apontadores de registro). Todos os ponteiros serão zero exceto o do registro 5000.

O byte 0 e os bytes de 3 a 12 do setor da lista de trilha/setor não são usados. Os bytes 1 e 2 contêm o número da trilha e os números dos setores, respectivamente, do próximo setor da lista. Se esses bytes forem ambos zero, esse será o último setor da lista.

O DIRETÓRIO

O DOS usa a trilha 17 ($11) para o diretório do disquette. Para cada arquivo, o diretório contém o nome, o tipo, o número de setores ocupados por ele (módulo 256) e a posição da lista de trilha/setor. Muitas dessas informações são mostradas com o comando CATALOG.

Cada setor do diretório contém informação de um a sete arquivos. O diretório inicia na trilha 17 setor 15. Quando este setor está cheio, o diretório continua no setor 14 etc, até o setor 1. O diretório pode conter entradas para até 84 arquivos.

O byte 0 e os bytes 3 até 10 de cada setor de diretório não são usados. Os bytes 1 e 2 contêm, respectivamente, os números da trilha e do setor seguinte do diretório. Se ambos são 0, este é o último setor do diretório. Os bytes de 11 até 255 contêm as entradas do diretório. Cada entrada usa 35 bytes; a primeira entrada vai do byte 11 até o 45, a segunda do 46 ao 80 etc.

As entradas de diretório são escritas no mesmo formato. A Tabela H.1 mostra o conteúdo de cada entrada. A Tabela H.2 mostra como o tipo de arquivo é codificado em cada entrada.

TABELA H.1. Formato do Diretório de Entrada.

Número do Byte Relativo	Conteúdo do Byte
0	Número da trilha na tabela de trilha/setor. Muda para 255 quando o arquivo é eliminado (conteúdo formador retiro no byte relativo 34)..
1	Número de setor da lista de trilha/setor.
2	Tipo de arquivo. Veja Tabela H.2.
3-32	Nome do arquivo, em ASCII.
33	Número de setores usados no arquivo, módulo 256.
34	Marca de fim. Normalmente 0, mas trocado para o conteúdo formador do byte relativo 0 quando o arquivo é eliminado.

O setor 0 da trilha 17 não tem entradas de diretório. Em seu lugar, ele tem informação de identificação de estado, descrição física e espaço disponível do disco. A Tabela H.3 mostra o conteúdo deste importante setor, chamado de Tabela de Volume do Conteúdo.

TABELA H.2. Codificação dos Tipos de Arquivos do Diretório do Disco.

Bit	Característica
0	Arquivo de programa em Integer BASIC se este bit é 1.
1	Arquivo de programa em Applesoft se este bit é 1.
2	Arquivo de programa binário se este bit é 1.
3-6	Reservado para futura expansão.
7	Arquivo travado se este bit é 1.
Se os bits de 0 até 6 forem todos 0, o arquivo é de texto.	

TABELA H.3. Tabela de Volume do Conteúdo (setor 0, trilha 17).

Byte	Descrição
0	Não usado.
1	Número de trilha do primeiro setor do diretório.
2	Número de setor do primeiro setor do diretório.
3	Número de distribuição do DOS.
4-5	Não usados.
6	Número de volume do disquette.
7-38	Não usados.
39	Número máximo de pares trilha/setor possíveis em cada setor da lista de trilha/setor.
40-47	Não usados.
48-51	Máscara para o mapa de setores disponíveis.
52	Número de trilhas por disquette.
53	Número de setores por disquette.
54-55	Número de bytes por setor: byte de menor ordem em 54, maior em 55.
56-59	Mapa de setores disponíveis, trilha 0.
60-63	Mapa de setores disponíveis, trilha 1.
64-195	Mapas de setores disponíveis, trilhas 2 até 195.
196-255	Não usados.

Cada grupo de quatro bytes do 56 até o 195 da Tabela de Volume do Conteúdo contém um mapa de disponibilidade para uma das trilha do disco. Cada mapa identifica quais setores da trilha associada está em uso e qual está disponível. Um bit tem o valor de 0 quando o setor correspondente está em uso, e tem o valor 1 quando o setor correspondente está livre. A Tabela H.4 indica quais bytes identificam quais setores.

TABELA H.4. Mapa de Disponibilidade de Setores.

Byte	Bit	Setor
Primeiro	7	15
	6	14
	5	13
	4	12
	3	11
	2	10
	1	9
	0	8
Segundo	7	7
	6	6
	5	5
	4	4
	3	3
	2	2
	1	1
	0	0
Terceiro	Todos	Não usado
Quarto	Todos	Não usado

I

Códigos de Caracteres Ascii e Códigos de Palavras Reservadas do Applesoft

A primeira tabela neste apêndice mostra os códigos ASCII de 1 a 96 e os caracteres que representam. Os códigos ASCII na faixa de 96 a 127 produzem os mesmos caracteres na tela do Apple II que os códigos de 32 até 63, embora com outros dispositivos de saída os códigos de 96 a 127 produzem letras minúsculas. Nenhuma tecla do teclado do Apple II produz códigos de 96 a 127.

Os códigos ASCII de 128 a 255 repetem os códigos de 0 a 127. Nenhuma tecla do Apple II produz estes códigos em Applesoft, embora eles possam ser gerados pelo Monitor e pelo Integer BASIC.

A segunda tabela lista os códigos de palavras reservadas do Applesoft. Cada palavra reservada usa apenas um byte na memória do programa. Cada palavra reservada é representada por um código, na faixa de 128 até 255. Este código substitui a palavra inteira na memória do Apple II e no disco. A lista está na ordem numérica por código. O Apêndice F contém uma lista de palavras reservadas em ordem alfabética.

Códigos de Caracteres ASCII.

Código ASCII	Caractere da Tela	Teclagem	Código ASCII	Caractere da Tela	Teclagem
0		CTRL-@	48	0	0
1		CTRL-A	49	1	1
2		CTRL-B	50	2	2
3		CTRL-C	51	3	3
4		CTRL-D	52	4	4
5		CTRL-E	53	5	5
6		CTRL-F	54	6	6
7	(bell)	CTRL-G	55	7	7
8	(backspace)	CTRL-H or ←	56	8	8
9		CTRL-I	57	9	9
10	(linefeed)	CTRL-J	58	:	:
11		CTRL-K	59	;	;
12		CTRL-L	60	<	<
13	(carriage return)	CTRL-M	61	=	=
14		CTRL-N	62	>	>
15		CTRL-O	63	?	?
16		CTRL-P	64	@	@
17		CTRL-Q	65	A	A
18		CTRL-R	66	B	B
19		CTRL-S	67	C	C
20		CTRL-T	68	D	D
21	(forward space)	CTRL-U or →	69	E	E
22		CTRL-V	70	F	F
23		CTRL-W	71	G	G
24	(cancela linha)	CTRL-X	72	H	H
25		CTRL-Y	73	I	I
26		CTRL-Z	74	J	J
27		Esc	75	K	K
28		n.d.	76	L	L
29		CTRL-SHIFT-M	77	M	M
30		CTRL- ^	78	N	N
31		n.d.	79	O	O
32	espaço	barra espaço	80	P	P
33	!	!	81	Q	Q
34	"	"	82	R	R
35	#	#	83	S	S
36	$	$	84	T	T
37	%	%	85	U	U
38	&	&	86	V	V
39	'	'	87	W	W
40	(	(	88	X	X
41	)	)	89	Y	Y
42	*	*	90	Z	Z
43	+	+	91	[	n.d.
44	,	,	92	\	n.d.
45	-	-	93	]	SHIFT-M
46	.	.	94	^	^
47	/	/	95		n.d.

Nota: Some 128 ao código ASCII se gerado por teclagem no Monitor ou no Integer BASIC.
n.d. = não disponível no teclado do Apple II.

Códigos de Palavras Reservadas do Applesoft.

Token	Reserved Word	Token	Reserved Word	Token	Reserved Word
128	END	164	LOMEM:	200	+
129	FOR	165	ONERR	201	-
130	NEXT	166	RESUME	202	*
131	DATA	167	RECALL	203	/
132	INPUT	168	STORE	204	^
133	DEL	169	SPEED=	205	AND
134	DIM	170	LET	206	OR
135	READ	171	GOTO	207	>
136	GR	172	RUN	208	=
137	TEXT	173	IF	209	<
138	PR#	174	RESTORE	210	SGN
139	IN#	175	&	211	INT
140	CALL	176	GOSUB	212	ABS
141	PLOT	177	RETURN	213	USR
142	HLIN	178	REM	214	FRE
143	VLIN	179	STOP	215	SCRN(
144	HGR2	180	ON	216	PDL
145	HGR	181	WAIT	217	POS
146	HCOLOR=	182	LOAD	218	SQR
147	HPLOT	183	SAVE	219	RND
148	DRAW	184	DEF	220	LOG
149	XDRAW	185	POKE	221	EXP
150	HTAB	186	PRINT	222	COS
151	HOME	187	CONT	223	SIN
152	ROT=	188	LIST	224	TAN
153	SCALE=	189	CLEAR	225	ATN
154	SHLOAD	190	GET	226	PEEK
155	TRACE	191	NEW	227	LEN
156	NOTRACE	192	TAB(	228	STR$
157	NORMAL	193	TO	229	VAL
158	INVERSE	194	FN	230	ASC
159	FLASH	195	SPC(	231	CHR$
160	COLOR=	196	THEN	232	LEFT$
161	POP	197	AT	233	RIGHT$
162	VTAB	198	NOT	234	MID$
163	HIMEM:	199	STEP		

J

Tabelas de Conversão

Este apêndice contém as seguintes tabelas de conversão:

- Números Hexadecimal – Binários
- Números Hexadecimal – Decimais Inteiros

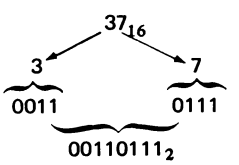
Tabela de Conversão Hexadecimal Binário

Use a tabela abaixo para conversões entre números hexadecimais na faixa de 0-0F e binários na faixa 0000-1111.

Para converter números maiores para hexadecimal, converte-se quatro dígitos binários por vez, operando da direita para a esquerda. Se existirem menos que quatro dígitos binários no grupo mais à esquerda, adicione zeros à direita. Aqui está um exemplo:

$$100101_2 = \underbrace{\underbrace{0010}_{2}\ \underbrace{0101}_{5}}_{25_{16}}{}_2$$

Converta números hexadecimais maiores que 0F para binário, um dígito por vez. Aqui está um exemplo:



Hexadecimal	Binário	Hexadecimal	Binário
00	0000	08	1000
01	0001	09	1001
02	0010	0A	1010
03	0011	0B	1011
04	0100	0C	1100
05	0101	0D	1101
06	0110	0E	1110
07	0111	0F	1111

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL INTEIRO

A tabela abaixo permite conversão direta entre inteiros hexadecimais de 0-FFF e decimais inteiros na faixa de 0-4096. Para conversão de números maiores, os valores da tabela podem ser adicionados aos seguintes:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

As frações hexadecimais podem ser convertidas para frações decimais como segue:

1. Expresse a fração decimal como um inteiro vezes 16^{-n}, onde n é o número de posições decimais significativas hexadecimais colocadas à direita do ponto hexadecimal

$$0.CA9BF3_{16} = CA9\,BF3_{16} \times 16^{-6}$$

2. Ache o equivalente decimal ao inteiro hexadecimal

$$CA9\,BF3_{16} = 13\,278\,195_{10}$$

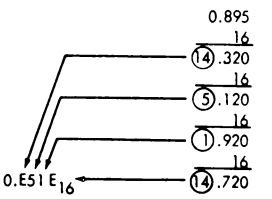
3. Multiplique o decimal equivalente por $16E^{-n}$

$$\begin{array}{r} 13\,278\,195 \\ \times\ 596\,046\,448 \times 10^{-16} \\ \hline 0.791\,442\,096_{10} \end{array}$$

As frações decimais podem ser convertidas para frações hexadecimais multiplicando sucessivamente a fração decimal por 16 na base 10.

Após cada multiplicação, a porção inteira é removida para formar a fração hexadecimal montada à direita do ponto hexadecimal. Entretanto, desde que é usada aritmética decimal para esta conversão, a porção inteira de cada produto deve ser convertida para números hexadecimais.

Exemplo: Converter 0.895 base 10 para seu equivalente hexadecimal:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1562	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

TABELA DE CONVERSÃO HEXADECIMAL-DECIMAL (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

K

Bibliografia

BASIC

Albrecht, Finkle, and Brown. *BASIC.* Peoples Computer Company, Menlo Park, California, 1967.

Coan, James S. *Advanced BASIC.* Hayden Book Co., Rochelle Park, New Jersey.

Coan, James S. *Basic BASIC.* Hayden Book Company, Rochelle Park, New Jersey.

Dwyer, T. *A Guided Tour of Computer Programming in BASIC.* Houghton Mifflin Company, 1973.

Kemeny, J., and Kurtz, T. *BASIC Programming.* Peoples Computer Company, Menlo Park, California, 1967.

Pegels, C. *BASIC: A Computer Programming Language.* Holden-Day, Inc., 1973.

Peoples Computer Company. *What to Do After You Hit Return.* Menlo Park, California.

Sack, J., and Meadows, J. *Entering BASIC.* Science Research Associates, 1973.

Programação em Linguagem Assembler

Leventhal, Lance A. *6502 Assembly Language Programming.* Osborne/McGraw-Hill, Berkeley, California, 1979.

Osborne, A. *An Introduction to Microcomputers: Volume 1 — Basic Concepts.* 2nd ed., Osborne/McGraw-Hill, Berkeley, California, 1980.

Scanlon, Leo J. *6502 Software Design.* Howard W. Sams, Indianapolis, Indiana.

Zaks, Rodnay. *6502 Applications Book.* Sybex, Berkeley, California.

Periódicos

Apple Orchard, P. O. Box 1493, Beaverton, Oregon 97075.

"Apple-Cart," *Creative Computing.* P.O. Box 789-M, Morristown, New Jersey 07960.

CALL A. P. P. L. E., 304 Main Ave. S., Suite 300, Renton, Washington 98055.

Compute!, P.O. Box 5406, Greensboro, North Carolina 27403.

Micro, P.O. Box 6502, Chelmsford, Massachusetts 01824.

Nibble. P.O. Box 325, Lincoln, Massachusetts 01773.

Personal Computing, P.O. Box 13916, Philadelphia, Pennsylvania 19101.

Purser's Magazine. P.O. Box 466, El Dorado, California 95623.

Softalk. 10432 Burbank Bl., N. Hollywood, California 91601.

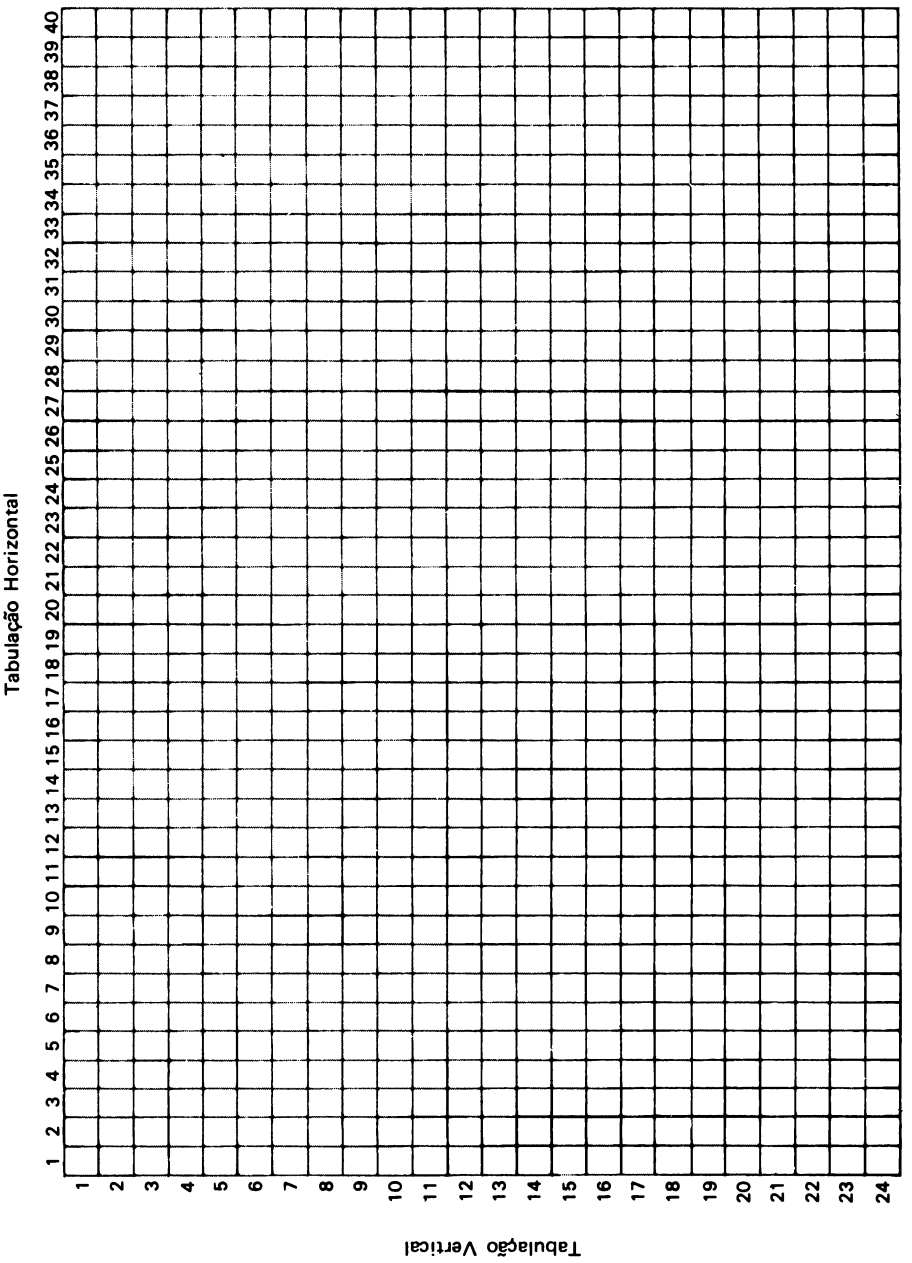
Publicações da Apple

Apple II/II Plus Reference Manual	A2L0001A
Parallel Printer Interface Manual	A2L0004
Apple II BASIC Programming Manual	A2L0005
Applesoft II Reference Manual	A2L0006
Communications Interface Manual	A2L0007
High-Speed Serial Interface Manual	A2L0008
Disk II Floppy Disk Manual (DOS 3.2.1)	A2L0012
Applesoft Tutorial Manual	A2L0018
Graphics Tablet Manual	A2L0033
Silentype Manual	A2L0034
The DOS Manual (DOS 3.3)	A2L0036

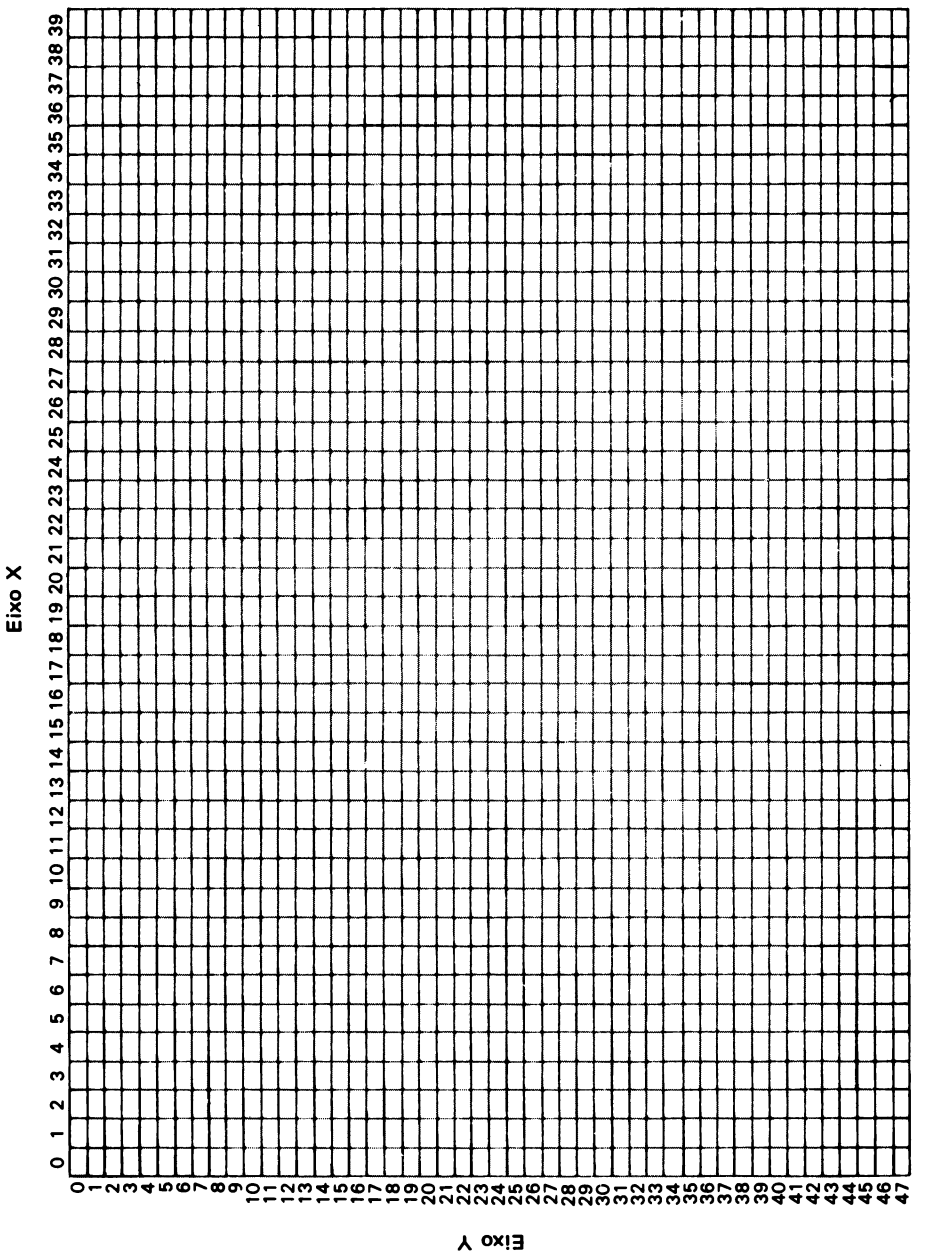
L

Formulários para Desenho na Tela

Use os formulários deste apêndice para projetar o formato da tela. No formulário de tela de texto, os números de linhas e colunas iniciam em 1, o que é apropriado para operação com texto. No formulário gráfico de baixa resolução, os números de linhas e colunas iniciam em 0, assim como os comandos gráficos de baixa resolução.



Tela de Texto



Tela Gráfica de Baixa Resolução

M

CARACTERÍSTICAS ESPECÍFICAS DO APPLE IIe

O Apple IIe teve sua produção iniciada em 1983 e é externamente semelhante ao modelo anterior, Apple II.* Possui um teclado aperfeiçoado, conectores para acessórios redesenhados no painel traseiro e algumas características que são vantajosas somente quando avaliadas juntamente com os demais acessórios.

De fato, o teclado do Apple IIe tem símbolos de pontuações em diferentes localizações dos modelos anteriores. Tem, também, algumas teclas que não são encontradas nas máquinas datilográficas normais. Estão incluídas as teclas ESC, TAB, CONTROL (ou CTRL), SHIFT LOCK, DELETE, RESET, RETURN, OPEN APPLE, SOLID APPLE, ←, →, ↑, e ↓.

* Consulte o catálogo de publicações da McGraw-Hill sobre suas últimas publicações — inclusive a versão do Apple IIe (e C) de 128 k. [Guia do Usuário do (Novo) Apple IIe — incluindo Apple IIc].

O Apple IIe tem, no mínimo, 64 K (65532) bytes de memória.

TABELA M.1. Interfaces/Expansões — Convenções de Slot

Interfaces	Slot (a)	Aplicações
Super Serial	1	Impressora
Super Serial	3	Comunicações
Interface Paralela	1 ou 2	Impressora; uso geral
80-colunas textos (b)	A	Mais caracteres por linha
80-colunas mais memória	A	Mais caracteres por linha; expansão de memória
Sistema de linguagem (c)	O	Expansão de memória e linguagem de programação
Linguagem Interger BASIC	O	Interpretador BASIC
Linguagem Applesoft	O	Interpretador BASIC
Controle de Disk-drives	6	Primeiro par de drives
Controle de Disk-drives	5	Segundo par de drives
Controle de Disk-drives	4	Terceiro par de drives
Controle de mesa gráfica	7	Mesa gráfica

(a) Os slots são numerados de 0 até 7, da esquerda para a direita. O slot A é o slot auxiliar do Apple IIe.

(b) Somente para o Apple IIe.

(c) Interno no Apple IIe.

TABELA M.2 Usos das Teclas Especiais

Tecla		Notação do Livro	Uso
IIe	Outros		
ESC	ESC	ESC	ESC abreviação de "escape". Usa-se ESC em conjunto com outra tecla (veja Tabela M.3), mas ela sempre é pressionada antes da outra. A isto chamamos uma seqüência "escape". Não se mantém pressionada ESC enquanto se pressiona outra tecla.
TAB	(não)	TAB	Avança o cursor até o próximo ponto de tabulação; atua somente em alguns programas. Outros programas ignoram a tecla TAB. Os pontos de tabulação padrão são a cada oito espaços.
CONTROL	CTRL	CONTROL	Sempre usa-se CONTROL juntamente com outra tecla: mantenha CONTROL pressionada enquanto pressiona e solta outra tecla. Tabela M. 3 lista algumas combinações de teclas.
SHIFT	SHIFT	SHIFT	Usa-se como a tecla de acionamento de maiúsculas de uma máquina de escrever para se obter letras maiúsculas ou acentos, pontuações e símbolos indicados no topo das teclas. Em alguns modelos anteriores ao Apple IIe, SHIFT não tem efeito sobre letras, pois elas são sempre maiúsculas.
CAPS LOCK	(não)	CAPS LOCK	Fixa o teclado para somente letras maiúsculas, sem a necessidade do uso da tecla SHIFT. Afeta somente as 26 letras do alfabeto.
(não)	REPT	REPT	REPT é a abreviação de repetir. Pressionando a tecla REPT junto com outra, ela será repetida até que se solte uma ou as duas teclas.

TABELA M.2 Usos das Teclas Especiais (cont.)

DELETE	(não)	DELETE	Em alguns programas, atua tal qual a tecla de retorno de posição em uma máquina de escrever, outras vezes apresenta no visor uma escala branca.
RETURN	RETURN	RETURN	Significa o fim de um comando ou fim da linha digitada. O Apple II não interpreta as teclas digitadas até que seja pressionada a tecla RETURN.
←	←	←	Usualmente trabalha como a tecla de retorno de posição em uma máquina de escrever. Tem outros usos; para edição de programas, veja o Capítulo 3.
→	→	→	Usualmente avança o cursor para a direita, copiando na memória do Apple II o caractere sobreposto, caso se deseje repeti-lo. Tem outros usos; para edição de programas, veja o Capítulo 3.
↑	(não)	↑	Vários usos; para edição de programas, veja o Capítulo 3.
↓	(não)	↓	Vários usos; para edição de programas, veja o Capítulo 3.
RESET	RESET	RESET	Interrompe o que o computador está fazendo e retorna o controle ao teclado. No Apple IIe e em alguns modelos anteriores, devemos pressionar a tecla CONTROL e RESET juntas.
OPEN APPLE 	(não)	OPEN APPLE	Vários usos. CONTROL-OPEN APPLE-RESET religa o Apple IIe como se tivesse sido desligado e ligado a chave geral.
SOLID APPLE 	(não)	SOLID APPLE	Vários usos. CONTROL-SOLID APPLE-RESET inicia a uma série de testes de diagnósticos que dura cerca de um minuto.

No Apple IIe podemos digitar letras minúsculas tão bem como letras maiúsculas. Em alguns programas (inclusive em BASIC) não se permitem letras minúsculas nos comandos. Portanto, deve-se pressionar a tecla CAPS LOCK antes de digitar o comando como descrito. Em processadores de textos e outros programas não há restrições para letras minúsculas, pode-se usar minúsculas livremente. Na dúvida, pressione a tecla CAPS LOCK.

O Apple IIe tem duas teclas do sinal gráfico apóstrofe. Em muitos casos, é comum comandos de programas requererem o caractere convencional, ' , direcionado para a direita. O caractere alternativo, ' , direcionado para a esquerda, não atuará.

Existe a tecla DELETE no Apple IIe, mas na maioria dos programas não retrocederá uma posição ao longo da linha. Neste caso, use a tecla ←, como descrito antes.

TABELA M.3 Seleção de Teclas Combinadas

Teclas combinadas *	Efeito
CONTROL-RESET	Interrompe o que estiver sendo executado pelo computador e retorna o controle ao teclado.
CONTROL-OPEN APPLE-RESET	Religa o Apple IIe como se tivesse sido desligado e ligado a chave geral.
CONTROL-SOLID APPLE-RESET	Inicia uma série de testes de diagnósticos que duram cerca de um minuto.
CONTROL-B	Atua somente quando o prompt do Monitor (*) estiver ativo. Quando se pressionar a tecla RETURN, em seguida, esta combinação retorna para o BASIC.
CONTROL-C	(Usualmente seguido do pressionamento da tecla RETURN.) Inicializa uma parada no que estiver sendo feito pelo computador. Em alguns modelos de Apple II, CONTROL-C chaveia do prompt do Monitor (*) para o BASIC.
CONTROL-S	Fixa a tela apresentada no visor; nenhuma nova tela será apresentada até que se pressione qualquer outra tecla. (Não Atua em todos os modelos Apple II.)
CONTROL-X	Cancela o comando anterior; usado após ter se pressionado a tecla RETURN.

* Em algumas combinações de teclas, todas elas devem ser pressionadas em conjunto; estas são as anotadas com hífen. Para outras combinações, elas devem ser pressionadas uma após a outra; estas são as anotadas com vírgulas.

TABELA M.3 Seleção de Teclas Combinadas (cont.)

REPT – qualquer tecla	Causa repetição de qualquer tecla. (Não necessário no Apple IIe).
ESC, @	Limpa a tela; o cursor move-se para o canto esquerdo do alto.
ESC, E	Limpa até o fim da presente linha.

Repetição Automática

Você deve ter verificado que quando mantém pressionada uma tecla no Apple IIe, ela se repete automaticamente. Isto acontece com todas as teclas, com exceção de ESC, CONTROL, CAPS LOCK, OPEN APPLE, SOLID APPLE e RESET.

TABELA M.4. Aspectos do Cursor Apple IIe

CURSOR		Interface de 80 Colunas	Número de Caracteres	Modo "escape"
Aspecto	Intermitente			
	Sim	Inativo	40	*
	Sim	Inativo	40	*
	Não	Ativo	80	Não
	Não	Ativo	40	Não
	Não	Ativo	80	Sim
	Não	Ativo	40	Sim

* Com a interface de 80 colunas, o cursor não indica o estado de modo "escape"

Movendo o Cursor

Existem três modos para mover o cursor usando o teclado. O método mais fácil atua somente no Apple IIe. Os métodos mais comuns atuam em todos os modelos Apple IIe.

No Apple IIe, podemos usar as teclas ó—, —ç, v, e ^ para mover o cursor por toda tela do visor. Mas primeiro devemos pressionar a tecla ESC para colocar o computador no modo "escape". Caso o Apple IIe esteja no modo de 40 colunas ou no modo de 80 colunas (interface de 80 colunas ativa) tecle ESC apresentando um cursor branco sólido. Este cursor avisa que o computador está no modo "escape". Caso a interface de 80 colunas esteja inativa (ou não instalada), pressionando-se a tecla ESC coloca-se o computador no modo "escape", apresentando um cursor quadrado axadrezado.

No Apple IIe, as teclas ←, →, ↓ e ↑ se repetirão caso você as mantenha pressionadas, movendo o cursor por longas distâncias, rapidamente.

Efeitos Especiais

O Apple IIe atualmente tem dois tipos de caracteres. O tipo normal é similar ao tipo de caracteres dos modelos anteriores ao Apple II. Eles podem ser letras maiúsculas intermitentes, normais, e inversas, mas não têm letras minúsculas intermitentes ou inversas. O tipo alternativo de caracteres, que atua somente quando com a interface de 80 colunas estiver ativa, tem letras intermitentes para qualquer tipo. Instantaneamente tem-se as letras maiúsculas como minúsculas, todos os dígitos e todos os caracteres especiais, tanto normais como inversos. Portanto, o comando FLASH não trabalha da mesma maneira no modo 80-ativo, bem como no modo 40-ativo.

Quando a tela estiver no modo de vídeo inverso e a interface de 80 colunas estiver ativa, o comando HOME limpa a tela, mudando a cor do fundo para branco e os caracteres para preto. Isso também ocorrerá quando a tela estiver no modo inverso no momento em que o comando PR#3 aciona a interface de 80 colunas.

Ativando a Tela de 80 Colunas

No Apple IIe equipado com uma interface de 80 colunas você pode colocar o computador no modo 80-ativo com um comando de Monitor. Tecle 3 após CONTROL-p, e pressione RETURN. Isto atuará como o comando PR#3.

TABELA M.5. Movimento do Cursor no Modo "Escape"

Teclado	O Cursor Move uma Posição	Atua no	
		Apple IIe	Apple II Plus'
—	Esquerda	Sim	Não
—	Direita	Sim	Não
↓	Para baixo	Sim	Não
↑	Para cima	Sim	Não
J	Esquerda	Sim ''	Sim
K	Direita	Sim ''	Sim
M	Para baixo	Sim ''	Sim
I	Para cima	Sim ''	Sim

* Para entrar no modo "escape", pressione a tecla ESC, para sair pressione-a novamente.

' Aplicável somente para os modelos Apple II mais recentes.

'' Não atua quando a interface de 80 colunas está ativa.

Apple IIe x outros modelos

Teclado

- Existem 63 teclas.
- Existem sete novas teclas de controle: OPEN APPLE, SOLID APPLE, ^, v, SHIFT LOCK, TAB, e DELETE (veja a Tabela M.2).
- A tecla de REPEAT não mais existe. Instantaneamente, todas as teclas repetem, automaticamente, quando mantidas pressionadas.
- Vários caracteres novos podem ser digitados: \ |, [, {, }, `, e ~.
- As teclas OPEN APPLE e SOLID APPLE são paralelas aos botões 0 e 1 do controlador de jogos.
- Vários caracteres são digitados agora em diferentes teclas: @, ^, &, (,), *, :, +, =, ", ', e].
- Para usar a tecla RESET, você deve pressionar a tecla CONTROL ao mesmo tempo.

Tela do Vídeo

- Todas as letras minúsculas podem ser usadas.
- Os tipos principais de caracteres têm letras maiúsculas ou minúsculas, pontuações e símbolos especiais no modo normal, formato branco e preto. Também têm letras maiúsculas, pontuações e símbolos especiais, intermitentes ou inversos.
- Os tipos alternativos de caracteres têm letras maiúsculas ou minúsculas, pontuações e símbolos especiais no modo normal ou inverse, mas nem todas são intermitentes.
- Telas com 80 colunas são possíveis com a instalação de uma interface.
- Funções adicionais podem ser programadas ou digitadas diretamente no teclado, usando-se seqüências de "escape".
- Existem três novos desenhos de cursores, descrevendo o modo atuante na tela (veja a Tabela M.4).

Dentro da CPU

- Existe um slot auxiliar no centro da placa principal para instalação da interface de 80 colunas.
- O slot 0 não mais existe e o cartão de linguagem está na própria placa principal, acrescentando 16k de memória de leitura e escrita adicionais.
- Uma memória auxiliar de leitura e escrita adicional é possível com uma interface de 80 colunas com extensão.

Painel Traseiro da CPU

- O painel traseiro de metal tem várias aberturas para dois tamanhos de conectores no padrão "D".
- Existe um soquete de nove pinos no padrão "D" (um DB9-P) para os controladores de mão.

Funções RESET

- CONTROL-RESET efetua um 'reset' normal.
- CONTROL-OPEN APPLE-RESET simula o desligar e ligar da fonte, preservando a chave da mesma.
- CONTROL-SOLID APPLE-RESET inicia um autoteste de um minuto.

BASIC

- Os comandos devem ser digitados em letras maiúsculas, embora nas variaveis possam ser incluídas letras minúsculas.
- Tabulações horizontais não atuam em mais de 40 colunas com o comando HTAB, ou com vírgulas ou com a função TAB no comando PRINT.

DOS

- As palavras de comando devem ser digitadas em letras maiúsculas.
- Usando-se PR#6 após ativar-se a interface de 80 colunas com PR#3, desorganiza-se a tela. Use PR#3 novamente para fixá-la.

Monitor

- O novo Monitor do Apple IIe usa os mesmos pontos para as sub-rotinas presentes, mas os códigos interiores devem ser diferentes.
- Algumas novas chaves e várias das antigas podem agora ser lidas.
- O Mini-Assembler só pode ser acessado através do Interger BASIC.
- O novo Monitor do Apple IIe pode ser identificado através da verificação do conteúdo de memória na localização $FBB3, que deverá ser 6.
- A partida automática do Monitor (normal no Apple II plus) é verificada através do Interger BASIC. Isto quando o conteúdo da memória na localização $FBB3 for $EA.

CÓDIGOS DOS CARACTERES ASCII

TABELA M.6 Caracteres, Teclas e Códigos ASCII

Códigos* ASCII	Caracteres na Tela		Teclado	
	Apple IIe	Outros	Apple IIe	Outros
0	Nenhum	Nenhum	CTRL-@	CTRL-@
1	Nenhum	Nenhum	CTRL-A	CTRL-A
2	Nenhum	Nenhum	CTRL-B	CTRL-B
3	Nenhum	Nenhum	CTRL-C	CTRL-C
4	Nenhum	Nenhum	CTRL-D	CTRL-D
5	Nenhum	Nenhum	CTRL-E	CTRL-E
6	Nenhum	Nenhum	CTRL-F	CTRL-F
7	Bell	Bell	CTRL-G	CTRL-G
8	Retrocesso	Retrocesso	CTRL-H	CTRL-H
9	Nenhum	Nenhum	CTRL-I	CTRL-I
10	Avanço de Linha	Avanço de Linha	CTRL-J	CTRL-J
11	Nenhum †	Nenhum	CTRL-K	CTRL-K
12	Nenhum †	Nenhum	CTRL-L	CTRL-L
13	Retorno	Retorno	CTRL-M	CTRL-M
14	Nenhum †	Nenhum	CTRL-N	CTRL-N
15	Nenhum †	Nenhum	CTRL-O	CTRL-O
16	Nenhum	Nenhum	CTRL-P	CTRL-P
17	Nenhum †	Nenhum	CTRL-Q	CTRL-Q
18	Nenhum †	Nenhum	CTRL-R	CTRL-R
19	Nenhum	Nenhum	CTRL-S	CTRL-S
20	Nenhum	Nenhum	CTRL-T	CTRL-T
21	Nenhum †	Espaço Avante	CTRL-U	CTRL-U
22	Nenhum †	Nenhum	CTRL-V	CTRL-V
23	Nenhum †	Nenhum	CTRL-W	CTRL-W
24	Nenhum	Linha de Cancel.	CTRL-X	CTRL-X
25	Nenhum †	Nenhum	CTRL-Y	CTRL-Y
26	Nenhum †	Nenhum	CTRL-Z	CTRL-Z
27	Nenhum	Nenhum	ESC	ESC
28	Nenhum †	Nenhum	CTRL-\	Nenhum
29	Nenhum †	Nenhum	CTRL-]	CTRL-SHIFT-M
30	Nenhum	Nenhum	Nenhum	CTRL-^
31	Nenhum	Nenhum	CTRL-SHIFT-_	Nenhum
32	Espaço	Espaço	Barra de espaço	Barra de espaço
33	!	!	SHIFT-1	SHIFT-1
34	"	"	SHIFT-'	SHIFT-2
35	#	#	SHIFT-3	SHIFT-3
36	$	$	SHIFT-4	SHIFT-4
37	%	%	SHIFT-5	SHIFT-5

* Acrescente 128 aos códigos listados se o caractere for gerado no Integer BASIC ou no Monitor.

† Atua como controle da tela quando a interface de 80 colunas estiver ativa.

Códigos* ASCII	Caracteres na Tela		Teclado	
	Apple IIe	Outros	Apple IIe	Outros
38	&	&	SHIFT-7	SHIFT-6
39	'	'		SHIFT-7
40	(	(	SHIFT-9	SHIFT-8
41	)	)	SHIFT-0	SHIFT-9
42	*	*	SHIFT-8	SHIFT-:
43	+	+	SHIFT-=	SHIFT-;
44	,	,	,	,
45	-	-	-	-
46	.	.	.	.
47	/	/	/	/
48	0	0	0	0
49	1	1	1	1
50	2	2	2	2
51	3	3	3	3
52	4	4	4	4
53	5	5	5	5
54	6	6	6	6
55	7	7	7	7
56	8	8	8	8
57	9	9	9	9
58	:	:	SHIFT-;	:
59	;	;	;	;
60	<	<	SHIFT-,	SHIFT-,
61	=	=	=	SHIFT-
62	>	>	SHIFT-.	SHIFT-.
63	?	?	SHIFT-/	SHIFT-/
64	@	@	SHIFT-2	SHIFT-P
65	A	A	SHIFT-A	A
66	B	B	SHIFT-B	B
67	C	C	SHIFT-C	C
68	D	D	SHIFT-D	D
69	E	E	SHIFT-E	E
70	F	F	SHIFT-F	F
71	G	G	SHIFT-G	G
72	H	H	SHIFT-H	H
73	I	I	SHIFT-I	I
74	J	J	SHIFT-J	J
75	K	K	SHIFT-K	K
76	L	L	SHIFT-L	L

* Acrescente 128 aos códigos listados se o caractere for gerado no Integer BASIC ou no Monitor.

Códigos* ASCII	Caracteres na Tela		Teclado	
	Apple IIe	Outros	Apple IIe	Outros
77	M	M	SHIFT-M	M
78	N	N	SHIFT-N	N
79	O	O	SHIFT-O	O
80	P	P	SHIFT-P	P
81	Q	Q	SHIFT-Q	Q
82	R	R	SHIFT-R	R
83	S	S	SHIFT-S	S
84	T	T	SHIFT-T	T
85	U	U	SHIFT-U	U
86	V	V	SHIFT-V	V
87	W	W	SHIFT-W	W
88	X	X	SHIFT-X	X
89	Y	Y	SHIFT-Y	Y
90	Z	Z	SHIFT-Z	Z
91	[	[	[	Nenhum
92	\	\	\	Nenhum
93	]	]	]	SHIFT-M
94	^	^	SHIFT-6	SHIFT-N
95	—	Nenhum	SHIFT-	Nenhum
96	`	Nenhum	`	Nenhum
97	a	A	A	Nenhum
98	b	B	B	Nenhum
99	c	C	C	Nenhum
100	d	D	D	Nenhum
101	e	E	E	Nenhum
102	f	F	F	Nenhum
103	g	G	G	Nenhum
104	h	H	H	Nenhum
105	i	I	I	Nenhum
106	j	J	J	Nenhum
107	k	K	K	Nenhum
108	l	L	L	Nenhum
109	m	M	M	Nenhum
110	n	N	N	Nenhum
111	o	O	O	Nenhum
112	p	P	P	Nenhum
113	q	Q	Q	Nenhum
114	r	R	R	Nenhum
115	s	S	S	Nenhum

* Acrescente 128 aos códigos listados se o caractere for gerado no Integer BASIC ou no Monitor.

Códigos* ASCII	Caracteres na Tela		Teclado	
	Apple IIe	Outros	Apple IIe	Outros
116	t	T	T	Nenhum
117	u	U	U	Nenhum
118	v	V	V	Nenhum
119	w	W	W	Nenhum
120	x	X	X	Nenhum
121	y	Y	Y	Nenhum
122	z	Z	Z	Nenhum
123	{	Nenhum	SHIFT - [	Nenhum
124	|	Nenhum	SHIFT - \	Nenhum
125	}	Nenhum	SHIFT -]	Nenhum
126	~	Nenhum	SHIFT-`	Nenhum
127	▒	Nenhum	DELETE	Nenhum

* Acrescente 128 aos códigos listados se o caractere for gerado no Integer BASIC ou no Monitor.

Chaves da Tela

O Apple IIe tem uma chave adicional, como listada na Tabela M.7.

– 16370 Seleciona os tipos principais de caracteres

No Apple IIe, seleciona os tipos principais de caracteres, que incluem letras maiúsculas ou minúsculas, pontuações e símbolos especiais em modo normal, formato branco no preto. Também são incluídos letras maiúsculas, pontuações e símbolos especiais, em modo intermitente ou inverso. Use somente POKE e não PEEK.

– 16369 Seleciona os tipos alternativos de caracteres

No Apple IIe, seleciona os tipos alternativos de caracteres, que incluem letras maiúsculas ou minúsculas, pontuações e símbolos especiais em modo normal, formato branco no preto ou inverso, formato preto no branco. Não inclui nenhum caractere intermitente. Use somente POKE, não PEEK.

TABELA M.7. Chaves do Apple IIe

Funções das chaves	Localização na memória	PEEK (localização) < 128	 > 127
Tipo de caractere	– 16354	Principal	Alternativo
Modo da tela	– 16358	Gráfico	Texto
Janela para texto	– 16357	Ausente	Presente
Memória para tela	– 16356	Página 1	Página 2
Modo gráfico	– 16355	Baixa-resolução	Alta-resolução

MEMÓRIA USADA

A memória do Apple II é dividida em três categorias gerais: memória de leitura e escrita ("read/write", também chamada de RAM "random-access memory"), memória só de leitura (ROM) e endereços de entrada e saída (I/O). Os endereços de memória de 0 até 49151 ($BFFF hexadecimal) são em RAM, e endereços de 49152 até 523257 ($C000 até $CFFF) são endereços de entrada e saída. Endereços de 53248 até 65535 ($DOOO até $FFFF) devem ser ROM ou RAM. Todo computador Apple IIe tem toda capacidade de memória, mas alguns modelos anteriores não têm a atual memória para todos os endereços. Por exemplo, se seu modelo Apple II tiver somente 16 K de RAM, endereços de memória de 16384 ($4000) até 49151 ($BFFF) não são usados.

Endereços dos Bancos de Memórias

Todo modelo de Apple II tem os endereços de ROM de 53248 até 65535 ($DOOO até $FFFF). O Apple IIe e modelos anteriores de Apple II têm o cartão de linguagem, bem como, os bancos de RAM nos mesmos endereços. Esta área é chamada banco de memória. Na programação em BASIC você chaveia entre os dois bancos quando usa os comandos FP ou INT para mudar a versão do BASIC.

Memória Auxiliar

Em adição aos 64k de memória descritos, o Apple IIe pode ter mais de 64k de RAM, chamada de memória auxiliar. Às vezes, o Apple IIe não pode acessar a memória auxiliar e a memória é que se pode selecionar quando ativa, a memória principal ou a auxiliar. Quando a interface de 80 colunas está ativa, o Apple IIe usa parte da sua memória auxiliar para guardar o texto extra de 40 colunas. O Monitor altera isto automaticamente após ter-se digitado o comando PR#3.

O Interpretador de Linguagem BASIC

No Apple IIe e nos demais modelos Apple II, tanto os interpretadores de Applesoft como de Integer BASIC residem no banco de memória entre os endereços 53248 e 63487 ($DOOO e $F800), o primeiro em ROM e o outro em RAM. A Tabela M.8 mostra em detalhe.

Se o seu sistema não tiver o banco-chaveado de memória por não ser um Apple IIe ou se não tiver cartão de linguagem, você pode carregar o interpretador de Applesoft de um disco ou cassette na área livre de memória entre os endereços 2048 e 12288 ($800 e $3000).

TABELA M.8. Distribuição das Áreas dos Bancos de Memórias

Modelo de Apple II	ROM		RAM	
	53248–63487 $D000–$F7FF	63488–65483 $F800–$FFFF	53248–63487 $D000–$F7FF	63488–65483 $F800–$FFFF
Apple IIe	Applesoft	Apple IIe Monitor	Integer BASIC	Autostar Monitor
Apple II plus com expansão	Applesoft	Autostar Monitor	Integer BASIC	Autostar Monitor
Apple II plus	Applesoft	Autostar Monitor	–	–

Índice Analítico

ABS, 308-09
Acumulador, 234, 236-37
Auto-falante, 102, 222
  posições de PEEK e POKE, 336
Anunciadores, controle de jogo, 338-39
APPEND, 184, 280. Veja também OPEN
Applesoft. Veja também Programas
  acesso a programas pelo Monitor, 231
  códigos de palavras reservadas, 343, 355
  comprimento de linhas de programa, 40
  nomes de variáveis, 61-62
  números de linhas, 55
  posições de tabulação em, 112-13
  reinício, 33
  reinício via CTRL-Y, 246
  troca para Integer BASIC, 46-47, 191
  uso de memória, 347-49
  valores numéricos, 37
  vírgulas em, 112-13
Applesoft Firmware, 7
Aritmética binária, 245-46
Arquivos
  acesso aleatório, 185-89
  buffers, 176, 192
  destravando, 173
  diferenças entre Integer BASIC/Applesoft, 181
  disco, 162
  eliminando do disco, 172
  END OF DATA, 181, 184
  EXEC, 189-90
  fechando, 163
  GET, 182
  guardando números em, 183
  linguagem de máquina, 194
  posicionando campos, 184, 189
  rebatizando, 173
  separação de campos, 183
  seqüenciais, 175-81
  setores usados, 167
  tipos, 167
  verificando, 174
Arquivo seqüencial
  abrindo, 176-77
  APPEND, 184
  fechando, 176
  gravando em, 177-82
Arquivos Imagem Binária. Veja Arquivos, linguagem de máquina
Arredondamento, 59
ASC, 309
ASCII,
  códigos, 354
  programação de caracteres em, 120-21
ATN, 309
Auto, 44, 154, 261

BASIC. Veja também Applesoft e Integer BASIC
  acessando, 26
  caractere de entrada, 35
  com linguagem assembler, 226-28
  iniciando, 27, 34-35
  reiniciando, 33
  versões do, 26, 191
BLOAD, 195, 220, 240-41, 262
  verificado, 245
BRUN, 195, 262

BSAVE, 194, 219, 262-63
verificando, 245

CALL, 103, 256, 263. Veja também USR
Caracteres de entrada, 14
applesoft, 14
integer BASIC, 14
monitor, 14
Carregamento de programa
cassette, 27-28
disco, 28
Carregando o DOS, 20-25, 46-47
applesoft, 165
CTRL-K, CTRL-P, carregando no Monitor, 165
início automático, 164
integer BASIC, 165
salto a partir do Monitor, 164
Carregando programas. Veja Carregamento de programas
Carriage return, 90
como parte da série, 121
Cartão controlador de disco, 7, 164
Cartão de interface de comunicação, 7
Cartão de interface de impressora paralela, 7, 146
Cartão integer BASIC, 7
Cartões. Veja Cartões de circuito
Cartões de circuito
cartão de interface de impressora paralela, 7
cartão de interface de impressora serial, 7
cartão integer BASIC, 7
Cartão Language System, 7, 13
Cartões Applesoft Firmware, 7
Cartões de controle, 6, 7, 104-105
cartões de interface de comunicação, 7
principal, 3
Cartões de interface, 104-05
Cassettes
com som, 226
guardando matrizes no, 149-150
guardando memória no, 237-38
guardando programas no, 45-46
lendo memória a partir do, 238-39
manuseio, 18
proteção contra gravação, 19
tons de referência, 238
CATALOG, 24-25, 166, 167, 263
CHAIN, 264
CHR$, 120-121, 148, 309-10
Chaves de software, 204, 337-38
CLEAR, 76, 264
CLOSE, 176-177, 181, 140, 265
CLR, 76, 154, 264-65
Códigos, 355
Coleção de lixo, 348
Color=, 265-66
Comando do DOS nulo, 180
Compondo linhas de programa, 151
Comprimento de linha, 147-148
ajustando impressão, 147-148
CON, 94, 154, 266
Concatenação, 66
Condicionais, 280
CONT, 94, 266-67
Constantes, 150-51
Contador de posição, 250-54, 257
Controladores de jogo, 9, 209, 348
operação, 102
posições de PEEK e POKE, 338-39
sensores, 102
Cor, 199
complemento, 221
Corte de permissão de escrita, 161
Corte de proibição de escrita, 161
COS, 310
CTRL, 16
CTRL-B, 14, 26, 47, 231
CTRL-C, 28, 33, 44, 95, 191, 230, 231
CTRL-E, 234, 236-37
CTRL-K, 164-65
CTRL-P, 164-65, 245
carregamentos no Monitor, 22
CTRL-S, 44, 233
CTRL-X, 16, 31, 48
CTRL-Y, 246
Cursor
controle, 117-18
definição, 13
determinação da posição horizontal, 116
determinação da posição vertical, 117
efeitos especiais de vídeo, 117-25
movimentação, 51-52
posicionamento, 117-18

Data (Dado), 56-58, 76, 154, 267
Declarações de atribuição, 75
Declarações PRINT, 89, 105, 152, 295-96
abreviada, 37
arquivos em disco, 177-80, 183
comandos DOS, 104-105
formatando, 114-16
numéricos como séries, 144
pontos e vírgulas em, 106-10
séries, 106-07, 112-13
SPC em, 115
TAB em, 115
vírgulas em, 111-13, 140
DEF FN. Veja também FN, 154, 267
DEL, 154, 268

DELETE, 172-73, 178, 268
Depuração de programas, 152
Dígito de verificação, 174, 238-39
DIM, 77, 269-70
Discos rígidos, 155
Discos. Veja também Disquettes
  buffer, 164
  carregamento, 165-66
  carregando imagem binário, 240-41
  catálogo, 24-25
  com som, 225
  diretório, 164, 166
  eliminando programas do, 172-73
  floppy, 156
  inicialização, 25-26, 162, 170-72
  lista de trilha / setor, 162-63
  LOAD, 168-69
  mini-, 156
  panes, 164
  processos de armazenamento, 164
  processos de armazenamento de dados, 158
  rígidos, 154
  RUN, 168
  setores, 159
  tabela de formas, 219
  trilhas, 159
  viagens, 170-72
  volume, 169-70
  winchester, 157
Disk II, 4-5, 20, 155
Dispositivos periféricos
  entrada pelos, 104-05
  saída para os, 104-05
Diskette System Master, 20, 22, 24-25, 27, 161, 166
Disquettes. Veja também Drives de disco, discos DOS
  buraco de índice, 160
  Diskette System Master, 20, 22, 24, 25, 27
  inserção, 20
  manuseio, 20
  proteção contra escrita, 161
  setorizados por software, 160
DOS
  carregando, 20-24, 46-47, 165-66
  modo programado, 104, 105, 146-47
  requisitos de memória, 345-46
  versões do, 24, 162-63, 166
DRAW, 221, 270
Drive de disco, 4-5, 155-64
  especificação dos comandos de disco, 167-68
  número de drive, 167-68
  número de slot, 167-68
  número máximo permitido, 168
Edição 17, 31
  adicionando linhas, 48-49
  eliminando caracteres, 51
  eliminando linhas, 48
  inserindo caracteres, 51-52
  limpando até o fim da linha, 52
  limpando até o fim da tela, 52
  modo edição, 49-50
  seqüências ESC, 49-50
  trocando caracteres, 50-51
  trocando linhas, 49-53
Efeitos de vídeo especiais, 117-20
END, 41, 95, 154, 271
Enganos. Veja Erros
Erros
  comando do Monitor, 236
  corrigindo, 31
  corrigindo durante entrada de dados, 1-38
ESC 17, 51, 52, 53
ESC @, 31, 48
Espaços em branco, 56
Estrada de dados, 105, 121-127
  backspaces, 139-40
  corrigindo erros, 139
  dados válidos, 129
  formulários, 134
  informações ao operador, 139-40
  interativa, 122-25
  marcados, 130, 134, 136, 139-40
  minimização de erros, 121
  organização, 121
  orientação de estilo, 125-27
  permitindo mudanças, 125-27
  procurando erros, 127-29, 131, 133-34
  projetando, 129-30
  verificação de faixa, 126-27
EXEC, 189-92, 271
EXP, 310
Exemplos de programa
  alta resolução em integer BASIC, 209
  colocação da janela de texto, 143-44
  definição de firma, 215-17
  entrada de janela de texto, 120
  formulário de entrada de dados, 134-35
  geração de som, 227-28
  linha diagonal em baixa resolução, 201
  marcador de fim de arquivo, 260-61
  programa de apresentação, 170
  repeteco, 125
  saída de som no falante, 226
  som em linguagem assembler, 250
  validação de entrada de dados, 131-34
Exemplo de sub-rotinas
  cabeçalho de impressão de linhas e colunas, 144

desenho de alta resolução em Integer BASIC, 208
determinação de cor em alta resolução, 207
entrada de dados numéricos, 126
entrada de dois caracteres, 131
entrando dados tipo série, 135-36
limpando página gráfica de alta resolução, 205
manuseio de erro, 129
máscara de entrada de dados, 130
perguntas sim/não, 126
Expressões, 65
aritméticas, 67, 68
booleanas, 70
reais, 68
relacionais, 68
Expressões aritméticas, 67
Expressões booleanas, 70
Expressões do tipo misto, 71
Expressões inteiras, 67
Expressões relacionais, 68

FN, 311. Veja também DEF FN
FOR, 82, 83, 118, 154, 178, 272-73
Formas
carregando tabela de forma, 220
definição de fim de forma, 212
desenhando, 221
guardando tabela de forma, 219
limpando, 221
montando tabelas de forma, 210
montando vetores por computador, 215
mudanças de tamanho, 221
reserva de memória para, 217
rotação, 220
Formatando, 114
FP, 26, 37, 273
FRE, 151, 311
Funções, 96-101
— aninhadas, 101
— séries, 99
— definidas por usuário, 100-101
— do sistema, 100
— numéricas, 94-95
Funções definidas por usuário, 100, 267

GET, 93-94, 131, 154, 273-74
ler arquivos de texto, 182
GOSUB, 87, 154, 274-75
GOSUB. Calculado, 87-88
GOTO, 78-82, 275
GOTO calculado, 79
GR, 197, 275
Gráficos
alta resolução, 202-23
baixa resolução, 196-202
janela de texto, 198
tela cheia, 198
Gráficos em alta resolução
arquivos em disco com, 194-95
cores, 206
desenhando linhas, 207-08
desenhando pontos, 207-08
dimensões de telas, 202
formas, 209-23
janela de texto, 204
integer BASIC, 204-09
limpando páginas em alta resolução, 205-06
mostrando na tela, 203-06
páginas, 202-03
requisitos de memórias, 202-03
reservando memórias, 202-03
voltando ao mono texto, 205
Gráficos em baixa resolução
cores, 196-97, 199
desenhando pontos, 199
determinando a cor da tela, 201
dimensões da tela, 196
gráficos de tela cheia, 198
janela de texto, 198
linhas horizontais, 200
linhas verticais, 200
páginas, 196-98
Gravador cassette, 18. Veja também cassettes
ajustando o volume, 18-20
conexão, 4
Gravador de fita. Veja gravador cassette

HCOLOR, 206, 276
HGR, 203-04, 276-77
alternativas para o, 204-05
HGR2, 204, 277
alternativas para o, 204-05
HIMEN:, 103, 154, 203, 217-18, 220, 257, 277-78, 346, 347-48
HLIN, 200, 278
HOME, 279
HPLOT, 210, 279
alternativas para o, 208
HTAB, 117, 140, 280

IF- THEN, 88, 280-81
Impressoras
ativando via Monitor, 246
CHR$ com, 148
códigos de controle, 148
conjunto de caracteres, 147-48
conectando, 9-11
formatando, 139
largura de linhas, 147-48

programáveis, 147-48
selecionando para saída, 146
tamanho de página, 147-48
Imprimindo cabeçalho, 114-15
Imprimindo listagem de programa, 148-49
IN#, 165, 281
modo programado, 104-05
INIT, 170-72, 281
INPUT, 91, 123-24, 130, 149, 154, 282-83
Integer BASIC. Veja também BASIC e Programas
acesso a partir do Monitor, 230-31
nomes de variáveis, 60
numeração automática de linhas, 44-45
números de linhas, 55
posições de tabulação no, 111
reiniciando, 33
trocando por Applesoft, 46-47
uso de memória, 345-47
valores numéricos, 36
Interface serial, 7, 146
INVERSE, 118, 178, 284

Janela de dados, 141-46
Janela de texto, 119-20
posições de PEEK e POKE, 336-37

Language System, 22
carregando DOS, 20-25, 165
cartão, 7, 13, 18
LEFT$, 311
LEN, 312
LET, 73, 284
Letras maiúsculas, 15-16
Letras minúsculas, 15-16
Ligando, 12
Linguagem assembler
com BASIC, 103, 256-57
depuração, 252-255
mini-assembler, 247-251
programas de listagem, 251-52
programas em disco, 184-85
relocação, 256-57
sub-rotinas internas, 255-57
Linguagens de programação, 54-55
dialetos, 54
sintaxe, 54
List, 43, 56, 284-85
Listando. Veja Listando programas
LOAD, 27, 154, 285-86
LOCK, 173, 286
LOG, 312
LOMEM:, 103, 154, 203, 257, 286-87, 346
LOOPS, 80-84, 272, 288
aninhados, 81-84

MAN, 44-45, 154, 287
Matrizes
armazenamento em cassete, 149-150
dimensionamento, 64-65
índices, 63
inicialização, 144, 152
nomes, 63
redimensionamento, 77
MAX FILES, 192-93, 287-88
Memória
acesso direto via BASIC, 102
alterando blocos de, 235
alterando endereços simples, 102, 234
apenas de leitura (ROM), 3-4
capacidade de, 3-4
comparando blocos, 243
de acesso aleatório (RAM), 3-4
determinando limites, 103, 104
endereçamento, 102
especificando em BASIC, 102-03
examinando palavras, 232
guardando via BASIC, 194-95, 240
guardando via Monitor, 237
leitura/escrita (RAM), 3-4
locação, 257
movimento de blocos, 195-242
mapas, 345-48
ponteiro de endereço, 232
preenchendo com um padrão, 240-41, 242
recuperando via Monitor, 238-39
verificando gravação, 244
Memória apenas de leitura. Veja Memória
Memória de leitura/gravação. Veja Memória
Mensagem de apresentação, 93-94, 125-27
Mensagens de erro, 30, 38, 323-327
MID$, 312
Mini- assembler, 246-51
acessando, 247
comandos do Monitor, 247
contador de posição, 248-49
formato de instrução, 148-49
saindo do, 248
seção de exemplos, 250-51
MOD, 68
Modo calculador. Veja Modo imediato
Modo definido. Veja Modo programado
Modo direto. Veja Modo imediato
Modo edição, 49-50
Modo imediato, 35, 154, 258
Modo indireto. Veja Modo programado
Modo programado, 35, 40, 258
comandos DOS, 105-06, 146-47, 174-75
restrições, 154

Modo texto, desenho em, 293
Modos de endereçamento, linguagem assembler, 248
Modulador de RF, 2, 12
MON, 193, 288
Monitor, 14, 229-48, 251-256
  acessando, 229
  alteração de registradores, 236-37
  Autostart Monitor, 22, 230
  com som, 224
  comando definido por usuário, 247
  comando de listar, 251-52
  comando de passo, 251-253, 255
  comando GO, 231, 245
  comando trace, 253
  comandos, 236
  contador de posição, 251-52
  definições, 5-6
  exame de registradores, 233
  guardando memória, 237
  ler memória, 238-39
  monitor padrão, 229
  mover memória, 240-41
  ponteiro de endereço, 233-35, 244
  procedimento para encher memória, 241
  processo de verificação de memória gravado, 244-45
  sub-rotinas internas, 255-257, 328
  saído do, 230
  seleção de dispositivo de entrada, 246
  seleção de dispositivo de saída, 246
  verificar memória, 241-44
  vídeo reverso, 246
Monitor Autostart,. Veja Monitor
Monitor de linguagem de máquina. Veja Monitor
Monitor em linguagem assembler. Veja Monitor

NEXT, 81, 154, 288-89
NEW, 41, 154, 288
NO DSP, 153, 289
NO TRACE, 152, 290
NOMON, 193-94, 289-90
NORMAL, 118, 290
Notação científica, 58-59
Numeração automática de linhas, 44-45
Número de slot, 3, 6, 7, 146, 168-69
  especificando em comandos de disco, 168
  selecionando, 105-06
Número de volume, 169-72
Número do drive, 168-69
Números, 57-59
  em notação científica, 58-59
  inteiro, 57
  reais, 58
Números aleatórios, 313
Números de linha, 41-42, 54, 151
  como endereços, 55
Números hexadecimais
  aritmética, 246-47
  conversão para decimal, 356-362

ON-GOSUB, 88, 291-92
ON-GOTO, 80, 292
ONNER GOTO, 154, 181, 290-91
  negando, 128
  rotina de manipulação de erros, 128
  rotina em linguagem de máquinas para, 291-92
OPEN. Veja também APPEND, 176, 185, 292
Operadores
  aritméticos, 65
  booleanos, 65
  precedência de, 65
  relacionais, 65
  séries, 65
Operadores lógicos, 70
Operadores relacionais, 68

Paddles. Veja Controle de jogo
Palavras reservadas, 63
Panes em disco rígido, 163
Panes em disco setorizados por software, 163
Parada de tubulação, 111-12
Parênteses, 66
PDL, 312-13
PEEK, 103,
  determinando coluna do cursor, 116
  determinando linha do cursor, 117
  posições úteis, 335
Plot, 199, 293
  em modo texto, 293
POKE, 103, 257, 294
  com formas, 218, 220
  com gráficos, 198, 204-05
  janela de texto, 119-20
  posições úteis, 335
Ponteiros de arquivos, 177-80
  movendo, 184, 189
Ponto e vírgula, 106-110, 140
  em arquivos de disco, 183
  separando variáveis, 90
POP, 86, 134, 294-95
POS, 116, 313
PR#, 146, 148-49, 165, 295
  modo programado, 104-05
Precedência, 65-66
  sobrepassando, 66
Programa de apresentação, 26, 170, 174
Programas

adicionando linhas, 43, 48-49
compactos, 150
declarações, 38
depuração, 151-52
destravando, 173
eliminando do disco, 172-73
eliminando linhas, 48
entrando espaços em branco, 56
execução, 41
finalizando, 41
guardando em cassettes, 45-46
largura de linhas, 36-43
linhas de múltiplas declarações, 43, 150
mais rápidos, 149-50
números de linhas, 41
otimização, 149-51
rebatizando, 173
seqüência de linhas, 41-43, 55-56
trocando linhas, 49-53
trovados, 173
verificando, 175
Programas em linguagem de máquina, 247-57
Veja também Linguagem Assembler

RAM. Veja Memória
READ, 74, 180-82, 184-85, 189
arquivo EXEC, 189-90
declaração de atribuição, 297
declaração de disco, 296-97
RECALL, 149-50
Recópia. Veja Tecla de flexa para a direita
Recursão, 86
Registrador Apontador de Pilha, 234, 236-37
Registrador índice X, 234, 236-37
Registrador índice Y, 234, 236-37
Registradores, 256
alterando, 236-37
examinando, 233
REM, 73, 151, 298-99
RENAME, 173-74, 299
Requisição, declaração INPUT, 93
REPT, 17, 31, 48, 53
RESET, 15, 16, 33, 95, 253
acidental, 16, 31-33
recuperação por, 33
RESTORE, 76, 299
RESUME, 128, 154, 299-300
RETURN, 154, 300
RIGHT$, 313-14
ROM . Veja Memória
ROT, 231, 300
RND, 314
RUN
declaração de disco, 301
declaração geral, 28, 123, 154, 301

Saídas
alinhando valores numéricos, 144
formatando, 140
matrizes, 141
Saídas de programa, 105-21, 139-49
Salto, 36, 145
Saltos
comandos do Monitor, 245
declarações BASIC, 78-80
SAVE, 154, 172, 301-02
SCALE, 221-22, 302
SCRN, 201, 314-15
Séries de caracteres
comparação, 69
concatenação, 66
série nula, 57
variáveis, 60, 61
Setores, 159
hard, 160
locação, 160
SGN, 315
SHIFT, 16
SHLOAD, 219-20, 302
SIN, 315
Sinal de aspas, com parte de série, 121
Sistema Operacional em Disco. Veja DOS
Som, 223-28
com linguagem de máquina, 224, 226
SPC, 115, 315-16
SPEED, 118, 303
SQR, 316
STOP, 95, 303
STORE, 149-50, 304
STR$, 316
Sub-rotinas
aninhadas, 86-87
chamando BASIC, 274, 292
chamando programa assembler, 263, 317
entrada de dados, 125-27
internas, 328-34
saída com GOTO, 134
Sub-rotinas intrínsecas, 328-34

TAB
declaração, 117, 140, 304
função PRINT, 115, 316-17
Tabela gráfica, 11
Tabela verdade, 70
Tabelas de forma, 210
apontando para, 220
carregando, 220

codificando bytes, 211-13
definição de fim de forma, 212
diretório, 213
entrando, 217-19
guardando, 219
tamanho, 213
vetor de desenho zero, 212
TAN, 317
Teclado, 1, 15
desativando via Monitor, 246
Tecla de flexa para a direita, 17, 31, 49, 51-53
Tecla Escape. Veja ESC
Tecla flexa para a esquerda, 17, 31, 53
Teclas de controle. Veja CTRL
Teclas de flexa, 17, 31
Técnicas de visualização, 142
Tela de TV, 1
Tela de vídeo
ajustando a largura, 49-50
ajuste de cor, 28-30
aparelho de televisão, 1, 2
chaves de software, 337-38
conexões, 2
determinando a cor da tela, 201
dimensões dá, 2
evitando a última coluna, 144
formas em alta resolução, 209-23
formatando, 139
gráficos em alta resolução, 202-09
gráficos em baixa resolução, 196-202
gráficos em tela cheia, 198
janela de dados, 141-46
limpando, 117
listando programas, 44
modo texto, 198
modo vídeo reverso, 118, 246
monitor de televisão, 1, 2
80 colunas, 9, 13
operando com, 36, 145
piscando, 124
projeto, 140
velocidade, 118
Teste de programa. Veja Depuração de Programa
TEXT, 304
TRACE, 152, 305
com DOS, 194
Trilhas, 158-59
locação, 160
Truncagem, 72
Tipos de programas
— aplicação, 5
— interpretadores, 5
— Monitor, 6
— saudação, 25, 170

UNLOCK, 173, 305
USR, 257, 321

VAL, 317-18
Valores numéricos, impressos como séries, 144
Variáveis, 150-51
atribuindo valores, 122-24
formato na memória, 345-46
inteiros, 61
nomes, 59, 63
numéricos, 61
rascunho, 151
reais, 62
séries, 60-61
Variáveis de índice, 80-81
Variáveis de rascunho, 151
Variáveis numéricas, 61
Variáveis reais, 62
Vetores
desenho, 210-13
fantasmas, 210
Vetores de desenho, 210
códigos, 211
montando vetores por computador, 215
Vetores fantasmas, 210
VERIFY, 174, 305-06
Vídeo reverso, 118, 246
Vírgulas, 111-15, 140
em arquivo de dados, 181-84
separando variáveis, 89
Vlin, 306
Vtab, 117, 140, 306

WAIT, 306-07
WRITE, 177-80, 184-85, 189, 307-08

XDRAW, 221, 308

Impressão e acabamento
(com filmes fornecidos):
EDITORA SANTUÁRIO
Fone (0125) 36-2140
APARECIDA - SP

OUTROS LIVROS NA ÁREA *

GUIAS DO USUÁRIO

Baras – Lotus 1-2-3 – Guia do Usuário
Castlewitz – VisiCalc – Guia do Usuário
Curtis – WordStar – IBM PC – Guia do Usuário
Ettlin – WordStar – Guia do Usuário (versão CP/M)
Hoffman – MS/DOS – Guia do Usuário
Hogan – CP/M – Guia do Usuário
Sachs – IBM PC – Guia do Usuário
Sanders – Manual do Apple – Macintosh
Townsend – dBase II – Guia do Usuário

GUIAS DO OPERADOR

Gifford – Apple II – Comandos Básicos
Ingrahan – CP/M – Comandos Básicos
Wilson – VisiCalc – Comandos Básicos
Wilson – IBM PC – Comandos Básicos

SÉRIE McGRAW-HILL/DATALÓGICA

Byers – dBase II: Aplicações Comerciais
Byers – dBase III: Banco de Dados para Todas as Aplicações
Fishback – Framework: Aplicações – Finanças/Administração/Negócios
Freedman – dBase II para Principiantes
Harrison – Framework para Principiantes

* Com referência a outros títulos, consulte nosso Catálogo de Informática.

0-07-450328-6

LON POOLE
MARTIN McNIFF
STEVEN COOK

APPLE II

GUIA DO USUÁRIO