CP-500
Microcomputador
Operação e
Linguagem Basic

EDITELE

CP-500 Microcomputador Operação e Linguagem Basic

CP 500
Microcomputador Operação e Linguagem Basic

5ª Edição
2ª Impressão
1984

EDITELE
Editora Técnica Eletrônica Ltda.
São Paulo - Brasil

Editele - Editora Técnica Eletrônica Ltda.
Rua Casa do Ator, 1060.
04546 - São Paulo - SP - Brasil.
Caixa Postal 30141

5ª Edição
2ª Impressão

Impresso no Brasil - Printed in Brazil

Advertência

Antes de ligar o seu CP 500, ou qualquer dos periféricos, verifique se as tensões indicadas nos equipamentos correspondem à tensão da rede local.

Leia com atenção, no capítulo 3, as instruções sobre o procedimento correto para operar o computador e seus periféricos. Caso não sejam seguidas as instruções do fabricante quanto a instalação do computador, este poderá causar interferência em receptores de rádio ou televisão.

Isto se deve ao fato de os componentes do CP 500 operarem em velocidade elevada, gerando sinais elétricos de alta freqüência.

Pode-se tentar as seguintes medidas para eliminar, ou pelo menos minimizar, a interferência no seu receptor:

- Mudar a posição da antena do receptor;
- Mudar a posição do computador em relação ao receptor;
- Afastar o computador do receptor;
- Conectar o computador a uma tomada que não esteja na mesma ramificação em que se encontra o receptor.

Apresentação

O CP 500 é uma poderosa ferramenta para o trabalho, para a casa ou para o lazer. Vinte anos atrás, a mesma capacidade custaria centenas de vezes mais do que o seu CP 500 custa hoje, e ocuparia toda uma sala.

Apesar de sua capacidade e complexidade interna, o CP 500 é de extrema simplicidade de operação. De fato, você pode determinar exatamente que configuração deve ter a sua máquina.

No nível mais simples de operação, você poderá utilizar os programas em fita cassete ou disquete pré-programados. Tudo que você precisará saber, nesse caso, é como carregar o programa na memória e como executá-lo; bem como seu modo de operação. Se é por aí que você pretende começar, veja nos capítulos 1 a 6, Seção de Operação, como proceder. No capítulo 16 você encontrará informações sobre os comandos CLOAD e SYSTEM, necessários para o carregamento desses programas na memória.

Num nível mais elevado, você poderá criar seus próprios programas e armazená-los na memória do CP 500. Para isso, caso você ainda seja um principiante em programação, estude os seis primeiros capítulos deste livro e então passe para a seção de Linguagem Basic (capítulos 15 a 26). Este livro, juntamente com outras obras da Editele especializadas em programação, fará de você um programador em linguagem Basic. Mas, se você já possui os conhecimentos necessários e especialmente se já tiver experiência em programação, poderá então ler toda a Seção de Operação, sem problemas.

O CP 500 possui características próprias e algumas importantes diferenças dos demais computadores. Por isso, alguns minutos gastos com a leitura deste livro, antes de pressionar a tecla ENTER, poderão lhe poupar muito trabalho.

Sobre este Livro

Este livro contém instruções sobre a operação do CP 500 e uma descrição da linguagem Basic que este utiliza. Ele foi estruturado para permitir uma localização rápida dos assuntos, quer você procure informações básicas ou técnicas.

Se você é um principiante não se preocupe com as partes técnicas da Seção de Operação. Cada capítulo começa com uma explicação básica sobre o assunto para, depois, entrar no aspecto técnico. Dessa forma, você poderá passar para o capítulo seguinte ao deparar com alguma dificuldade. Mais tarde, após assimilar novos conhecimentos, retorne àquele ponto e tente acompanhar as explicações.

Se você já tem alguma noção, pode ir diretamente às partes técnicas, ou consultar o apêndice A (resumo).

Atenção

Antes de fazer qualquer conexão no seu CP 500 leia atentamente os capítulos 2 e 3, não importando o quanto você pensa que sabe. Estes capítulos se referem ao CP 500 em sua configuração básica (sem *drives*).

Quando tudo mais falhar, lembre-se: **leia as instruções.**

Sumário

Seção Operacional

Seção de Linguagem Basic

Seção de Apêndices

Operacional

Capítulo 1

Uma Rápida Descrição do Computador

O CP 500 é um computador à base de EPROM que incorpora numa mesma estrutura os seguintes componentes:

Vídeo

A tela do computador pode mostrar todos os 96 caracteres normais do nosso alfabeto, com maiúsculas e minúsculas, e ainda 64 símbolos gráficos e 160 caracteres especiais próprios do CP 500. Além desses, ainda existem conjuntos de caracteres de controle e de compressão de espaço, que conferem maior poder aos seus programas em Basic.

Teclado

O teclado do CP 500 permite a introdução de todo o texto normal e dos caracteres de controle. A digitação de valores numéricos é agilizada pela existência de um teclado numérico reduzido, à esquerda do teclado de texto. No teclado você pode selecionar letras maiúsculas ou minúsculas.

Você controla o computador através do teclado. Toda vez que quiser interromper qualquer operação do computador, quer seja a execução de um programa, ou uma gravação em fita, ou ainda uma impressão, você poderá usar a tecla BREAK, a qual devolverá o controle do CP 500 para você.

Uma outra característica do teclado é que toda tecla tem função de auto-repetição, ou seja, quando uma tecla qualquer for mantida pressionada por mais de um segundo (aproximadamente), o caractere dessa tecla será repetido enquanto a tecla assim for mantida.

Microprocessador Z 80

Trata-se da *unidade central de processamento* (UCP ou CPU — *central processing unit*), componente eletrônico onde todo o "pensamento" do computador é realizado, a uma velocidade acima de dois milhões de ciclos por segundo.

Memória Apenas de Leitura (EPROM)

Aqui são armazenados os programas internos do computador, incluindo o *"interpretador de linguagem Basic"*. Toda vez que o computador é ligado, a memória EPROM

comanda o microprocessador, habilitando você a operá-lo com instruções simples da linguagem Basic.

Tecnicamente, dizemos que a memória EPROM do CP 500 armazena 16 "kB" (lê-se *quilobait*). A unidade de medida *kB* equivale a 1024 *bytes*, que são posições dentro da memória onde se pode armazenar informação. Em linhas gerais, cada caractere ocupa um *byte*. Portanto, o CP 500 armazena, em EPROM, 16 * 1024 = 16384 caracteres, ou *bytes*, de programação permanente.

Memória de Acesso Aleatório (RAM)

Esta memória armazena seus programas e resultados, assim como os dados a eles referentes. Você pode não apenas gravar seus programas, como também alterar, corrigir ou atualizar programas e dados, modificar a configuração do CP 500, e várias outras possibilidades que você terá a chance de aprender com este livro. O CP 500 possui 48 kB de memória RAM que armazenam as informações enquanto o computador estiver ligado, ou seja, assim que você desligar o CP 500, toda a informação na RAM se perderá.

Periféricos

Periféricos são equipamentos que você pode acoplar ao seu computador para aumentar sua versatilidade em programação e armazenamento de dados. O CP 500 contém as "interfaces" necessárias para simplificar a conexão de muitos periféricos.

FITA CASSETE

Para guardar seus programas e/ou dados para referências futuras, você só precisa dispor de uma fita cassete comum. O CP 500 permite que um gravador cassete comum lhe seja acoplado diretamente. Isso permite que a gravação de programas e dados da memória para a fita, ou da fita para o computador, seja realizada sem complicações, como a gravação de suas músicas preferidas. O CP 500 ainda oferece a opção de se gravar em alta ou baixa velocidade de transferência, mas isso fica para ser abordado mais a frente.

IMPRESSORA

A maioria das impressoras com interface serial ou paralela pode ser conectada ao CP 500. Com a impressora você poderá emitir documentos, correspondência, listas de programas para arquivo, convites e tantas outras coisas que dependem de uma cópia impressa e de baixo custo.

Outras Características

O CP 500 possui, opcionalmente, dois *drives* para disquetes de 5¼", mas pode ainda acomodar duas outras unidades externas, perfazendo um total de quatro unidades de discos flexíveis. Com os disquetes você poderá gravar e carregar programas e dados com grande rapidez e confiabilidade, substituindo com vantagens a fita cassete. Ao usar os disquetes, seu computador estará sob controle de um programa, que vem gravado no

disquete, mais poderoso que o próprio interpretador Basic, residente em EPROM. Esse programa é chamado *"Sistema Operacional de Disco"* ou, simplesmente, DOS 500.

Você poderá, ainda, usar a interface serial RS 232 C do CP 500, que permite sua comunicação com outros computadores que estejam equipados com o mesmo padrão de interface serial, com impressoras seriais ou com qualquer outro equipamento serial.

Capítulo 2

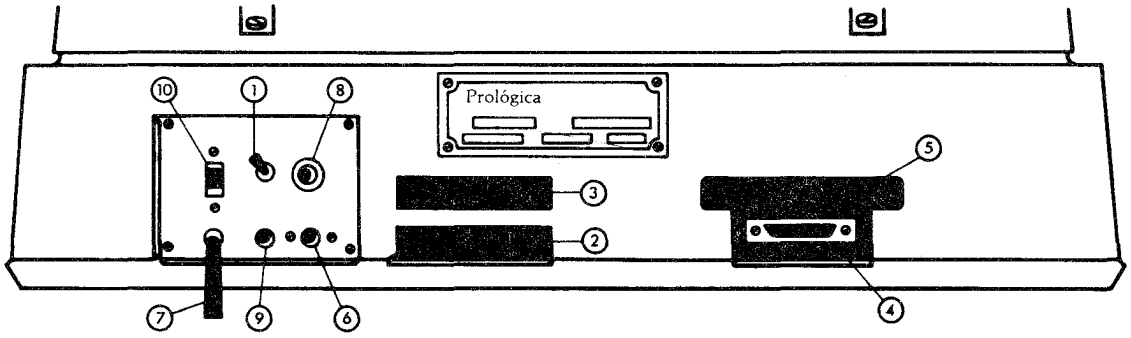
Instalação

Muito cuidado ao desembalar seu computador, pois ele é composto por componentes eletrônicos integrados que podem ser danificados por um impacto mais forte. Ao abrir a embalagem, remova todos os calços e guarde-os, para o caso de precisar transportar o CP 500 novamente. Certifique-se de guardar todos os cabos, calços, papéis e outros objetos que possam ser úteis na reembalagem.

Coloque o computador no local onde será utilizado (mesa, bancada, etc.). Ele deve ser conectado diretamente à tomada de força, sendo **desaconselhado o uso de extensões.** Não conecte o computador à tomada, ainda.

Conexão dos Periféricos

Antes de conectar qualquer periférico (o gravador, por exemplo), certique-se de que tanto o periférico quanto o computador não estejam conectados às tomadas. Só então, leia no manual apropriado o modo correto de proceder a essa conexão. A figura a seguir mostra a distribuição dos conectores na parte de trás do CP 500.



1. Interruptor (Liga/Desliga).
2. Conector para expansão de *drives.*
3. Conector para impressora com interface paralela.
4. Conector RS 232 C.
5. Conector do *I/O Bus* (barramento de entrada/saída de dados), para expansões futuras.
6. Tomada para gravador cassete. Gravador e cabo acompanham o computador. O plugue com fio preto é conectado à tomada *MONITOR* do gravador; o com fio cinza, à tomada *MIC* e; o com fio cinza pequeno ao *MIC REMOTE.*
7. Cabo de força.
8. Ajuste de brilho do vídeo.
9. Fusível.
10. Chave seletora de tensão (110/220V).

Conexão com Gravador Cassete

As instruções a seguir se aplicam ao gravador que acompanha o computador. Se você preferir usar outro gravador, lembre-se que tanto a conexão quanto a operação poderão ser diferentes.

1. Conecte o cabo (plugue DIN de um lado e três pinos do outro) na tomada atrás do computador (veja figura na página) e certifique-se de que o plugue está corretamente encaixado.
2. Conecte o plugue preto na tomada EAR ou MONITOR na lateral do gravador. Essa conexão fornece o sinal de saída do gravador para o computador (para carregar programas da fita na memória do CP 500).
3. Conecte o plugue preto na tomada MIC. Essa ligação fornece o sinal de gravação do computador para o gravador.
   Use esta conexão mesmo quando carregando programas na fita.
4. Conecte o plugue cinza pequeno na tomada *REMOTE* (furo menor). Isso permitirá que o CP 500 controle automaticamente o motor do gravador, parando ou movimentando a fita de acordo com os seus comandos pelo teclado ou por programa.

Nota: Você não precisa conectar o gravador a menos que pretenda gravar ou carregar programas.

Nota: Ao utilizar um gravador, não se esqueça de desligar microfones internos, para que não haja sobreposição de gravações de áudio com as de programas.

Capítulo 3

Operação

Como ligar

A conexão dos equipamentos às tomadas deve ser feita com os mesmos desligados.

As especificações elétricas são encontradas nos próprios equipamentos e nos manuais de operação que os acompanham.

As instruções a seguir se referem a operação do CP 500 sem disquete. Se o seu CP 500 está equipado com *drives* e você pretende usar o DOS 500, então deve seguir as instruções do livro *DOS 500 Sistema de Operação de Disco*, que acompanha o equipamento. Se você não quiser usar o DOS 500, deve proceder do modo explicado no quadro da página 23.

O computador, bem como os periféricos, deve estar desligado. Os periféricos devem ser ligados antes do computador (se você tiver todo o sistema conectado a um interruptor comum, acione-o, simplesmente).

Logo após, aparecerá no vídeo a seguinte mensagem:

```
Basic (S ou N)?
```

O significado desta mensagem será explicado mais adiante.

Se a mensagem não aparecer, poderá ter ocorrido um dos seguintes problemas:

a) A tela pode estar precisando de ajuste de brilho. Você pode ajustá-lo através do botão cuja localização está indicada na figura da página 19

b) Se com o ajuste a mensagem ainda não aparecer, então desligue todo o sistema e verifique as conexões. Veja no capítulo *"Detecção de Falhas e Manutenção"*.
Não ligue ou desligue nenhum periférico enquanto o computador estiver em uso, pois isso poderá causar problemas (o computador pode "travar" ou perder todos os dados armazenados).

Botão RESET

É o botão que se encontra na cavidade à direita do teclado. Sua função é reiniciar a operação do computador após um corte na energia, ou outra situação imprevista qualquer. Nesses casos, basta apenas pressionar esse botão, ao invés de desligar e ligar o computador no interruptor.

Neste livro identificaremos esse botão como RESET

Nota: A palavra inglesa *reset* é um termo técnico que significa inicializar, ou seja, recolocar um sistema, no caso o CP 500, em sua condição inicial de operação.

Quando acionada essa tecla, o computador não apaga os conteúdos da RAM, entretanto, o interpretador de linguagem Basic recomeçará suas operações sem levar em consideração qualquer programa ou dado que você tiver colocado na memória.

Para interromper um programa em execução sem perder o programa ou os dados, pressione a tecla BREAK.

Como Desligar

Desligue o computador *sempre* antes dos periféricos.

Se você desligar o computador por qualquer razão, deixe-o desligado por, pelo menos, quinze segundos antes de acioná-lo novamente. A fonte de alimentação interna precisa desse tempo para descarregar completamente a energia armazenada.

Os programas contidos na memória sempre serão apagados quando se desligar o computador. Portanto você não deve se esquecer de guardar as informações (em fita, por exemplo) antes de desligá-lo.

Diálogo Inicial

Quando você liga o computador, ou pressiona RESET, o computador começa suas operações com uma série de perguntas, denominada *"diálogo inicial"*. A primeira pergunta é:

```
Basic (S ou N)?
```

Ela oferece a você a opção de escolher entre o Interpretador Basic e o Monitor-residente. Como resposta você deve digitar S para operar o computador em Basic, ou N para usar o Monitor-residente.

O Monitor-residente é um programa de apoio para depuração de programas em linguagem de máquina, gravar ou carregar trechos de memória em fita ou, ainda, confeccionar pequenas rotinas ou programas em linguagem de máquina do Z 80. O apêndice G contém os comandos e o modo de operação do Monitor-residente.

Respondendo com S à primeira pergunta, o CP 500 perguntará:

```
Cass?
```

Essa pergunta se refere à velocidade de transferência dos dados do computador para o gravador e vice-versa. Você pode escolher entre velocidade alta (1500 bauds), pressionando A, ou baixa (500 bauds), usando B.

Pode-se, ainda, pressionar a tecla ENTER ao invés dessas, e o CP 500 assumirá que foi escolhida a velocidde alta. Para maiores detalhes, veja o capítulo "Usando a Interface para Gravador Cassete".

A terceira e última pergunta diz respeito à quantidade de memória que deve ser usada pelo computador, e aparece no vídeo da seguinte forma:

```
Mem. usada?
```

A resposta a essa pergunta estabelecerá o limite superior de endereçamento da memória RAM que o Interpretador Basic poderá utilizar. Será dentro desses limites que os seus programas e dados serão armazenados.

Responda simplesmente com ENTER a essa pergunta. O CP 500 entenderá com isso que você deseja usar toda a capacidade de armazenamento da RAM. Programadores experientes podem querer reservar parte da memória para programas e rotinas em linguagem de máquina. Informações sobre como conseguir isso são dadas no capítulo "Informações Técnicas".

Terminado o diálogo inicial o computador indicará que está pronto para uso com a mensagem:

```
PROLOGICA
READY
>
```

Como Acionar Sistemas com Disquetes sem Carregar o DOS 500.

Se o seu computador possui *drives* para disquetes e você quer operá-lo sem fazer uso do DOS 500, mantenha a tecla BREAK pressionada quando ligar ou inicializar (RESET) o computador. Se usar RESET lembre-se de soltá-la antes da tecla BREAK . Ao começar suas operações e constatar essa tecla pressionada, o computador entende que não se quer usar o sistema operacional de disco.

Modos de Operação

O interpretador Basic possui quatro modos distintos de operação:

Imediato: Para a introdução de programas e linhas imediatas.

Execução: O computador está nesse modo durante a execução de um programa ou de uma linha imediata.

Edição: Esse modo é utilizado para a correção de linhas de programa.

SYSTEM: Usado para carregar programas em linguagem de máquina gravados em fita e para o controle de transferência desses mesmos programas.

MODO IMEDIATO

Toda vez que o CP 500 passar para o modo imediato aparecerão no vídeo um cabeçalho e uma combinação de símbolos da seguinte forma:

```
READY  (cabeçalho)
>      (sinal de introdução seguido do cursor intermitente)
```

Enquanto o computador estiver no modo imediato, só será reconhecido qualquer comando após digitar-se a tecla ENTER . Chamaremos isso de *"introdução de linha"*.

O interpretador de linguagem Basic ignora espaços em branco dentro de uma linha. Sempre que encontra um espaço ele passa para a próxima letra, ou melhor, para o próximo caractere diferente de espaço.

Se o primeiro caractere de uma linha for um número, o interpretador a tratará como uma *linha de programa* e a armazenará na memória de programas. Caso contrário o interpretador considera a linha como *linha imediata* e a executa.

Por exemplo:

```
PRINT "LINHA IMEDIATA"
```

Esta linha será considerada imediata pelo interpretador. Mas, se você escrever assim:

```
10 PRINT "LINHA DE PROGRAMA"
```

Ela será armazenada na memória como linha de programa.

Linha Imediata: Consiste de uma ou mais instruções separadas entre si por dois pontos. Sua execução ocorre tão logo seja pressionada a tecla ENTER daí ser chamada imediata.

Por exemplo:

```
CLS: PRINT "A RAIZ QUADRADA DE 4 E'";SQR(4)
```

Quando você introduzir essa linha, pressionando ENTER, ela será executada, aparecendo na tela a seguinte linha:

```
A RAIZ QUADRADA DE 4 E' 2
```

Linha de Programa: O número que as inicia deve ser um inteiro entre 0 e 65529, inclusive. O restante da linha segue as mesmas regras que uma linha imediata.

Quando uma linha de programa é introduzida (ENTER), seu conteúdo é armazenado na área da memória RAM reservada para textos de programas, juntamente com outras linhas que você já tiver introduzido. O programa armazenado não será executado até que introduza um comando de execução, que será discutido mais adiante.

Por exemplo:

```
100 CLS: PRINT "A RAIZ QUADRADA DE 4 E'"; SQR(4)
```

Essa linha, por começar por um número dentro da faixa válida, é uma linha de programa e será armazenada, tão logo se pressione ENTER, na memória de programa, obedecendo à ordem de numeração da linha.

Para executá-la, digite R U N e pressione ENTER. Na tela aparecerá a linha:

```
A RAIZ QUADRADA DE 4 E' 2
```

Teclas Especiais no Modo Imediato

Tecla	*Função*
?	O ponto de interrogação pode substituir a instrução PRINT. Por exemplo, a linha imediata: `? "OLA'!"` Essa abreviação pode ser usada tanto em programas quanto em linhas imediatas. É importante salientar que L? não substitue LPRINT.

. O ponto pode substituir o número da linha atual de programa, ou seja, a última linha introduzida, alterada ou executada. Por exemplo:

```
LIST .
```

Essa linha faz com que a linha atual seja mostrada na tela (em programação, diz-se "listada").

.' O apóstrofo substitue a instrução REM. Sua função é fazer com que o computador não interprete o restante da linha. Isso permite que se coloque comentários em linhas de programas, para facilitar sua interpretação. Quando usado em linhas de várias instruções não precisa ser precedido por dois pontos. Por exemplo:

```
20 PRINT 2+2 ' IMPRIME 4
```

Quando executada essa linha, aparecerá na tela o número 4, como resultado da operação 2+2. A frase após o apóstrofo serve como descrição da função da linha, e é ignorada pelo interpretador.
Essa abreviação é geralmente usada em programas, porém, sintaticamente, não há problemas em usá-la em linhas imediatas.

SHIFT **1** ***** Essa combinação de teclas faz com que o computador imprima o conteúdo da tela em uma impressora a ele acoplada. A impressão pode ser interrompida a qualquer momento pressionando-se a tecla **BREAK**.
Essa função também opera em outros modos.

MODO DE EXECUÇÃO

Sempre que o computador estiver executando instruções (linhas imediatas ou programas), ele estará no modo de execução. Nesse modo o conteúdo da tela fica sob controle do programa.

Teclas Especiais no Modo de Execução

Tecla	*Função*
SHIFT **@**	Essa combinação força uma pausa no processamento. Para prosseguir basta pressionar qualquer tecla, com excessão de **BREAK** e **RESET**.
BREAK	Interrompe a execução de qualquer operação do computador, seja programa, linha imediata, impressão ou gravação. Após **BREAK** o controle do computador está com você.

Nota: Deve ser usada a tecla **SHIFT** da esquerda. Em alguns equipamentos a combinação **SHIFT** **1** ***** é substituida pela combinação das teclas **S** e **P**, pressionadas ao mesmo tempo.

MODO DE EDIÇÃO

Além do interpretador e do Monitor-residente, o CP 500 incorpora um *editor de linha* para auxiliar o programador a alterar linhas de programas. Para ter essa função operando, introduz-se o seguinte comando:

EDIT *número de linha*

onde *número de linha* especifica qual linha será alterada (editada).

Durante a edição de uma linha, seu número é mantido na tela, e não pode ser alterado. A introdução é realizada por caractere digitado, sendo dispensado o uso de ENTER. Maiores informações são fornecidas no capítulo "Edição", na seção de Linguagem Basic.

MODO MONITOR (SYSTEM)

Nesse modo você pode carregar e executar qualquer programa em linguagem de máquina. Por linguagem de máquina entendemos que seja o conjunto de instruções do microprocessador Z 80 do seu computador. Neste manual nos referimos à linguagem de máquina como *programação Z 80,* para distingüir de *programação Basic.*

Você não precisa entender de programação Z 80 para desfrutar das vantagens que ela oferece. Boa parte dos programas pré-gravados estão escritos nessa linguagem ao invés de em Basic. Para carregar o computador com esses programas (os em fita), você precisa do modo System.

Apesar de ter importantes vantagens sobre o Basic, como uma velocidade de processamento, a linguagem de máquina é muito mais trabalhosa e requer um bom conhecimento do circuito interno do computador. O capítulo "Informações Técnicas" é dedicado àqueles que já possuem esses conhecimentos. Maiores detalhes podem ser encontrados nos capítulos "Usando a Interface para Gravador Cassete" e "Comandos".

Exemplo de Utilização do CP 500

Mostraremos a seguir um exemplo passo a passo de como operar seu computador, desde o diálogo inicial até a correção de uma linha e execução do programa.

Se você ainda não sabe usar o teclado, leia o capítulo "Usando o Teclado", antes de começar com este exemplo.

NOTAÇÕES USADAS NESSE EXEMPLO

`Pontilhado`	Mensagens colocadas pelo computador; não é preciso digitar.
Itálico	Mensagens digitadas por você.
ENTER	Significa que deve ser introduzida uma linha.
SHIFT 0	Indica que devem ser usadas as letras minúsculas.
SHIFT →	Indica o uso da tabulação.

PROCEDIMENTO

Ligue o computador (se já estiver, pressione RESET), para poder acompanhar o exemplo.

```
Basic (S ou N)? S
Cass? B (para baixa, A ou ENTER para alta)
Mem. usada? ENTER
PROLOGICA
READY
>
```

O sinal intermitente é o cursor. Sua função é indicar onde aparecerá o próximo caractere digitado. Nesse ponto, o computador está pronto (em inglês, *ready*) para receber suas instruções. Então, continue.

```
> NEW ENTER
> READY
> —
> 10 CLS
> 20 PRINT" OLA'! > EU SOU O SEU COMPUTADOR PESSOAL CP 500 !"
> 30 PRINT > SHIFT 0 o que me faz ser esperto?!" SHIFT 0
> 40 PRINT" SHIFT 0 milhões desses:" SHIFT 0
> 50 PRINT CHR$(21)
> 60 FOR I=1 TO 256
> 70 PRINT CHR$(253);CHR$(254);
> 80 NEXT 1
> 90 PRINT CHR$(21)
> 100 END
```

O programa agora já está na memória. Para vê-lo, digite:

LIST ENTER

Na tela surgirá uma listagem do programa como a seguinte:

```
10 CLS
20 PRINT"OLA'!EU SOU O SEU COMPUTADOR PESSOAL CP 500 !"
30 PRINT"o que me faz ser esperto?!"
40 PRINT"milhoes desses:"
50 PRINT CHR$(21)
60 FOR I=1 TO 256
70 PRINT CHR$(253);CHR$(254);
80 NEXT I
90 PRINT CHR$(21)
100 END
READY
>
```

Verifique linha por linha para ver se não houve enganos na digitação. Não se preocupe com os espaços. Caso uma das linhas esteja incorreta, digite-a novamente. Por exemplo, digamos que haja um erro na linha 90 do tipo:

```
90 PRINT CHR$(201)
```

Para corrigi-la, você deve digitar:

90 PRINT CHR$(21) ENTER

Quando tudo estiver correto, o programa poderá ser executado através do comando:

RUN ENTER

Capítulo 4

Usando o Teclado

A finalidade do teclado é permitir a introdução de todo o texto normal de um programa, bem como os dados e os caracteres de controle. Como no teclado das máquinas de escrever comuns, certas teclas possuem dois símbolos. Para introduzir o símbolo inferior basta pressionar a tecla, enquanto que para o símbolo superior deve ser pressionada a tecla desejada juntamente com SHIFT. O ponto de interrogação, por exemplo, é introduzido pressionando-se SHIFT e a tecla 1.

Maiúsculas e Minúsculas (SHIFT 0)

As teclas de A a Z podem produzir letras maiúsculas e minúsculas, como as das máquinas de escrever. Porém o CP 500 possui dois modos distintos de digitação: só maiúsculas e maiúsculas e minúsculas. O computador começa suas operações aceitando apenas maiúsculas, o que facilita a digitação de programas. Quando você quiser usar maiúsculas e minúsculas, para digitar mensagens, por exemplo, deve pressionar SHIFT 0. Dessa forma as teclas passam a introduzir minúsculas diretamente e maiúsculas quando usadas juntamente com SHIFT. Para voltar ao modo inicial basta pressionar SHIFT 0, novamente.

Teclas Especiais

Certas teclas têm funções especiais em Basic e, ao invés de aceitá-las como caracteres, o computador executa as funções a elas atribuídas.

Tecla	*Função*
←	Retrocede e apaga o último caractere digitado.
→	Passa o cursor para o início do próximo grupo de oito colunas, na mesma linha (tabulação).
SHIFT ←	Retorna ao começo da linha.
SHIFT →	Muda o vídeo para 32 caracteres por linha.
SHIFT @	Força uma pausa na execução de um programa. Pressione qualquer tecla para retomar o processamento.
ENTER	Introduz uma linha. O interpretador não reconhece nenhuma digitação até que esta tecla seja usada.
CLEAR	Cancela a linha que estiver sendo digitada, apaga toda a tela, converte

para 64 caracteres por linha e posiciona o cursor no canto superior esquerdo.

BREAK — Interrompe qualquer operação do computador e devolve o controle para o teclado.

SHIFT ↓ * — Ativa a função de impressão do conteúdo da tela na impressora. A tecla BREAK interromperá essa função, colocando o computador no modo imediato.
ou
S P

Outras Características

Toda tecla tem função de auto-repetição, ou seja, quando você mantém qualquer tecla pressionada por mais que um segundo, o caractere correspondente passa a ser repetido na tela, enquanto a tecla assim permanecer.

À direita do teclado de texto há um teclado menor, com doze teclas, que facilita a introdução de valores numéricos. Ele opera da mesma forma que o teclado maior.

Códigos de Controle

Se você não está familiarizado com o conceito de códigos de controle, consulte o apêndice C, "Códigos de Caractere".

Existem, no código ASCII, 32 caracteres de controle (códigos 0 a 31). Alguns deles são introduzidos diretamente por teclas especiais (ENTER e BREAK, por exemplo). Outros são introduzidos por combinação das teclas SHIFT e ↓ com letras.

No apêndice C existe uma lista completa dos códigos de caracteres de controle.

Nota: Deve ser utilizada a tecla SHIFT da esquerda.

Capítulo 5

Usando a Tela de Vídeo

Dimensão do Caractere

A tela do CP 500 está dividida em 16 linhas que podem conter caracteres de duas largùras diferentes: a normal, que permite 64 caracteres por linha (64 cpl); e a largura dupla, que permite apenas 32 caracteres por linha (32 cpl).

Quando acionado o computador opera com 64 cpl até que essa condição seja alterada pelo acionamento simultâneo das teclas SHIFT e →, ou pela seguinte instrução:

```
PRINT CHR$(23) ENTER
```

Para retornar à condição inicial deve-se pressionar CLEAR ou executar o comando abaixo:

```
CLS ENTER
```

Cursor

O cursor indica a posição do próximo caractere a ser colocado na tela. Inicialmente ele aparece como um pequeno retângulo intermitente. Porém, tanto o caractere quanto a condição de intermitência podem ser alterados.

A posição 16412 controla a intermitência do cursor. Quando seu conteúdo é nulo, o cursor é intermitente. Quando for diferente de zero, não haverá intermitência. Para ilustrar essa explicação, veja o seguinte exemplo:

```
POKE 16412,1 ENTER
```

Essa linha fixa o cursor na tela.

```
POKE 16412,0 ENTER
```

Essa outra retorna à condição inicial.

A posição 16419 contém o código ASCII do caractere correspondente ao cursor. Inicialmente seu conteúdo é 176 (veja no apêndice C, na tabela de caracteres gráficos). A mudança desse valor é realizada pela instrução POKE, como no caso anterior, da seguinte forma:

```
POKE 16419,63 ENTER
```

Essa linha troca o cursor normal por um ponto de interrogação. Para repor o cursor normal, digite:

```
POKE 16419,176 ENTER
```

Ao acompanhar o exemplo do capítulo 3 você deve ter notado que o cursor não aparece durante a execução de um programa. Essa característica também pode ser mudada. Introduza a seguinte linha:

```
PRINT CHR$(14) ENTER
```

Para voltar à condição inicial, introduza:

```
PRINT CHR$(15) ENTER
```

Proteção contra Deslocamento

O deslocamento do conteúdo da tela ocorrerá quando a tela estiver repleta e se forçar o computador a escrever uma nova linha. Com o deslocamento a primeira linha da tela será substituída pela segunda, e essa pela terceira, e assim sucessivamente, até que a última seja substituída pela nova.

Essa característica permite que uma lista maior que a tela passe continuamente pelo vídeo, permitindo uma verificação "dinâmica". Uma "listagem" de programa, por exemplo, se desloca continuamente, de baixo para cima.

Já em uma tabela, por exemplo, o cabeçalho será deslocado juntamente com o restante, o que é inconveniente para a identificação das colunas. Devemos, então, proteger o cabeçalho contra o deslocamento, mantendo-o no alto da tela.

Você pode proteger, no CP 500, um cabeçalho de até sete linhas, deixando que apenas as nove restantes sejam deslocadas.

O endereço 16916 na memória define o número de linhas protegidas. Portanto, um zero nessa posição indica que nenhuma linha está protegida; o número um indica que uma linha não será deslocada; e assim por diante. A linha abaixo, por exemplo, protege as quatro primeiras linhas da tela:

```
POKE 16916,4 ENTER
```

E a seguinte recoloca na condição inicial (sem proteção):

```
POKE 16916,0 ENTER
```

Os valores maiores que sete são divididos por oito e apenas o resto da divisão é considerado. Como exemplo, considere que o valor 26 seja colocado (POKE 16916,26) nesse endereço: o computador divide esse número por oito, o que resulta em três e restam dois. O computador considera, então, o número 26 como 2, simplesmente. Esse método é chamado de "módulo oito".

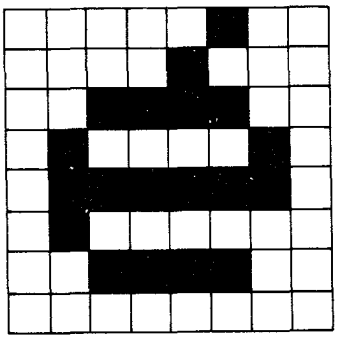
O programa a seguir demonstra a característica de proteção:

```
10 CLS:POKE 16916,3       ' PROTEGE AS 3 LINHAS DO TOPO
20 PRINT "AS 3 PRIMEIRAS LINHAS NAO SERAO DESLOCADAS"
30 PRINT "MAS AS DEMAIS SERAO"
40 PRINT "----------------------------------------"
50 FOR I=1 TO 100
60 PRINT "ESTAS  LINHAS NAO ESTAO EM  AREA PROTEGIDA"
70 PRINT "PORTANTO SERAO DESLOCADAS"
80 NEXT I
90 POKE 16916,0           ' REMOVE A PROTECAO
```

Caracteres de Texto

A tela do CP 500 pode reproduzir todos os caracteres normais do código ASCII, incluindo letras maiúsculas e minúsculas.

A formação dos caracteres na tela é feita pela seleção de pontos dentro de uma matriz de oito colunas por oito linhas, como a mostrada abaixo, o que permite uma boa definição das letras.



O programa em Basic, abaixo, coloca na tela todos os 96 caracteres de texto, com os respectivos códigos.

```
10 CLS
20 FOR I=32 TO 127
30 PRINT @ (I-32)*8,I;CHR$(I)
40 NEXT I
```

A maioria desses caracteres podem ser introduzidos diretamente pelo teclado.

Caracteres Gráficos

O CP 500 possui também 64 caracteres gráficos que podem ser usados na formação de desenhos ou tabelas na tela. Esses caracteres são formados por todas as combinações possíveis de uma matriz de duas colunas por três linhas (seis pontos), como é mostrado a seguir.

Nota: A tecla ↑ aparece na tela como um colchete ([), ao invés de uma seta para cima. Isto se deve ao fato de o CP 500 produzir na sua tela caracteres segundo o padrão ASCII. Entretanto, se o seu programa necessitar colocar em uma mensagem esta seta, o circunflexo (^) dará um melhor resultado.

Os códigos de 128 a 191 correspondem aos caracteres gráficos. No apêndice C existe uma tabela completa com os caracteres gráficos e o programa abaixo os coloca na tela de seu computador.

```
10 CLS
20 FOR I=128 TO 191
30 PRINT @ (I-128)*8,I;CHR$(I)
40 NEXT I
```

Caracteres de Compressão de Espaço

Na codificação do CP 500 os códigos de 192 a 255 são reconhecidos como códigos de compressão de espaço. O código 255 representa 63 espaços em branco; o código 254, 62 espaços; e assim sucessivamente até chegarmos ao código 193, que representa um único espaço. O código 192 representa zero espaços, ou seja, um "caractere nulo".

Esses códigos são muito úteis quando se quer economizar memória, substituindo uma seqüência de espaços dentro de uma linha por um único caractere; ou quando se quer ter flexibilidade na tabulação de vários ítens de comprimento variável.

Podemos tomar como exemplo uma linha de 64 caracteres usada para cabeçalho de uma agenda onde constam: nome, endereço e telefone. Esse cabeçalho deve ficar da seguinte forma:

```
                21 espaços          18 espaços
NOME                    ENDEREÇO                FONE
```

Nesse exemplo existem duas séries de espaços que podem ser substituídas por dois caracteres de compressão. Se usarmos espaços reais no lugar de dois caracteres, serão necessários 39 espaços. Portanto, nesse caso, economiza-se 37 caracteres e, conseqüentemente, área de memória.

O programa abaixo ilustra esse exemplo:

```
5 CLS
10 POKE 16526,105          'BYTE MENOS SIGNIFICATIVO DE
ENTRADA DO ENDERECO $INITIO
20 POKE 16527,0            'BYTE MAIS SIGNIFICATIVO
30 X=USR(0)                'CHAMADA DE $INITIO (CALL)
40 CLEAR 100
50 A$="NOME"+CHR$(192+21)+"ENDERECO"+CHR$(192+18)+"FONE"
60 PRINT "O COMPRIMENTO DA STRING E'";LEN(A$)
70 PRINT "AQUI ESTA' ELA:"
80 PRINT A$
```

Característica Especiais

Além desses caracteres, o CP 500 possui mais um conjunto de 96 caracteres especiais, englobando desde sinais matemáticos como o da raiz, além de letras acentuadas de nossa língua, até os alfabetos grego e japonês (katakana).

Os 32 caracteres de códigos mais baixos (0 a 31), devem ser colocados por uma instrução POKE diretamente na memória de vídeo (endereços de 15360 a 16383). Os 64 restantes (192 a 255), que normalmente são códigos de compressão, podem ser impressos por uma instrução do tipo PRINT CHR$, após terem sido ativados por caracteres de controle.

O caractere de controle que aciona os caracteres especiais é o 21. Por isso, a instrução:

```
PRINT CHR$(21) ENTER
```

muda os caracteres de 192 a 255 de códigos de compressão para caracteres especiais e vice-versa.

O caractere de controle 22 alterna os caracteres especiais com o alfabeto katakana japonês. Dessa forma, a instrução:

```
PRINT CHR$(22) ENTER
```

aciona o alfabeto katakana, que será impresso apenas se os caracteres especiais estiverem ativados (código 21). A mesma instrução desativa essa condição.

O programa a seguir ativa caracteres especiais e, em seqüência, o alfabeto japonês:

```
5 CLS
10 POKE 16526,105          'BYTE MENOS SIG. DE ENTRADA DO
ENDERECO $INITIO
20 POKE 16527,0            'BYTE MAIS SIG.
30 X=USR(0)                'CHAMADA DE $INITIO
40 PRINT CHR$(21)          'SELECIONA CARACTERES
ESPECIAIS
50 INPUT "PRESSIONE <ENTER> PARA VER OS CARACTERES
ESPECIAIS"; X
60 FOR I=192 TO 255
70 PRINT CHR$(I);
80 NEXT I
90 PRINT
100 INPUT "USE <ENTER> PARA VER O ALFABETO JAPONES";X
110 PRINT CHR$(22)          'SELECIONA KATAKANA
120 INPUT "USE <ENTER> PARA RETORNAR AO MODO NORMAL";X
130 PRINT CHR$(22);CHR$(21)
```

Capítulo 6

Usando a Interface para Gravador Cassete

A capacidade do CP 500 de ser conectado diretamente a um gravador cassete comum permite que as fitas magnéticas sejam utilizadas como expansão da memória de seu computador. Dessa forma, é possível guardar todos aqueles programas que você utiliza normalmente, eliminando-se a necessidade de redigitações cansativas. Recomendamos fitas C-45.

Conecte o gravador seguindo as instruções do capítulo 3.

Velocidade de Transferência

Como já foi explicado, você pode escolher entre velocidade alta ou baixa de transferência de dados durante o diálogo inicial. Se você estiver usando fitas cassete comuns, recomendamos baixa velocidade (500 bauds). Nessa velocidade são transferidos, aproximadamente, 63 caracteres por segundo.

Na velocidade alta (1500 bauds) são transferidos, aproximadamente, 190 caracteres por segundo. Em programas curtos essa diferença de velocidade não é sensível devido ao fato de ser gravado um "prefixo", com duração fixa, antes de qualquer dado na fita. A diferença torna-se relevante quando se transfere grande quantidade de informação, como programas extensos e listas de dados.

A mudança de velocidade pode ser realizada sem a necessidade de voltar ao diálogo inicial, pois é determinada pelo conteúdo do endereço 16913.

Quando esse endereço (posição) contiver um valor diferente de zero, a velocidade de transferência será "alta". Caso contrário (nulo), será "baixa". A linha abaixo seleciona a velocidade alta:

```
POKE 16913,1 ENTER
```

Erros de Carregamento

Durante a operação de carregamento da memória com programas em fita podem surgir três mensagens diferentes no canto superior direito da tela. Essas mensagens indicam que a operação não foi bem sucedida e deve ser repetida. A tabela abaixo explica o significado de cada uma.

Mensagem	*Significado*
C *	Ocorre quando há um erro de verificação (*checksum error*) durante o carregamento de programas em linguagem Z 80.
D *	Ocorre quando há um erro de dados (*data error*) durante o carregamento de um programa em Basic.
BK	Indica que a tecla BREAK foi pressionada cancelando a operação.

Os dois primeiros podem ser causados por conexão incorreta do gravador ou volume inadequado do sinal. Verifique essas condições e tente novamente. Se ainda persistir o problema, pode ser que haja sujeira na cabeça de gravação. Limpe-a conforme está explicado no manual do gravador e tente mais uma vez. Se mesmo assim a operação não se realizar, então, provavelmente a gravação na fita foi danificada por eletricidade estática, campo magnético intenso ou alguma outra causa.

Gravando um Programa Basic em Fita

Você pode gravar em fita um programa que esteja na memória usando o comando CSAVE. Com isso você passa a dispor de uma cópia permanente que poderá ser usada a qualquer momento, sem a necessidade de ser redigitada.

Não se esqueça de escolher a velocidade de transferência mais adequada à fita que você for usar. Para fitas comuns recomendamos a velocidade baixa (500 bauds).

Após todas essas precauções, comece a gravação seguindo as seguintes instruções:

Etapa 1

Coloque a fita, de preferência virgem, no gravador.

Etapa 2

Prepare o gravador e verifique os controles de volume e tonalidade (este deve estar no máximo).

Etapa 3

Coloque o gravador para gravar, pressionando as teclas PLAY e RECORD (ou equivalente).

Etapa 4

Digite CSAVE "P" e pressione ENTER .

O computador, então, gravará (alguns usam o termo *"salvar"*) o programa da memória na fita identificando-o com a letra P. Essa identificação é chamada *"nome de programa"* e pode ser usada qualquer letra como nome de arquivos em fita.

Quando a gravação terminar aparecerá na tela a mensagem:

```
READY
>
```

É uma boa idéia gravar o mesmo programa em duas fitas distintas, quando possível, ou, pelo menos, duas vezes na mesma fita. Essa precaução evita perder definitivamente cópias de programas cuja gravação seja danificada por qualquer motivo.

Copiando um Programa Basic (Carregamento)

Para copiar na memória (carregar) um programa Basic gravado em fita siga rigorosamente as seguintes instruções:

Etapa 1

Verifique se a velocidade de transferência com a qual o programa foi gravado coincide com a velocidade atual do computador. Se for o caso, mude o conteúdo do endereço 16913, conforme já foi explicado nesse capítulo.

Etapa 2

Prepare o gravador para tocar a fita, ajustando o volume para os níveis indicados na tabela a seguir.

Suas fitas	Fitas pré-gravadas
Volume entre 5 e 7	Volume entre 4 e 6

Etapa 3

Digite CLOAD e pressione ENTER . O computador carregará (copiará) o primeiro programa que encontrar na fita. Enquanto o programa estiver sendo carregado, no canto superior direito da tela aparecerão dois asteriscos. O da esquerda "piscará" a cada 64 caracteres (*bytes*) transferidos. Quando terminar o carregamento, o computador mostrará a mensagem:

```
READY
>
```

Etapa 4

Digite LIST e pressione ENTER para listar na tela o programa carregado.

Etapa 5

Execute o programa usando para isso o comando RUN ENTER

Como Procurar um Programa

Se no mesmo lado da fita foram gravados vários programas, você deve fazer com que o computador procure, entre eles, aquele que lhe interessa. Para isso você deve usar o nome do programa. Por exemplo: se o programa foi gravado com o nome "P", você deve

usar o comando:

```
CLOAD "P" ENTER
```

Enquanto o computador estiver procurando pelo programa, ele colocará na tela o nome dos programas (um caractere) que for encontrando. Quando encontrar o programa especificado, no lugar do nome ficará um asterisco.

Carregando um Programa em Linguagem de Máquina

Como já foi mencionado nesse livro, as fitas podem conter programas gravados em linguagem de máquina (Z 80), e estes podem ser carregados na memória do computador. É o caso de algumas fitas pré-gravadas.

A primeira providência é verificar a compatibilidade da velocidade de transferência da gravação com a do computador. Siga as seguintes instruções:

Etapa 1

Prepare seu gravador para tocar a fita, ajustando o volume da mesma forma que para fitas com programas Basic.

Etapa 2

Digite SYSTEM e pressione ENTER . Em resposta o computador colocará na tela o sinal indicativo do Monitor-residente:

```
*?
```

Etapa 3

Digite o nome do programa. Por exemplo: se o programa tem o nome EDTASM, você deve digitar EDTASM e pressionar ENTER .

O computador carregará o programa e, enquanto isso, aparecerão na tela dois asteriscos no canto superior direito da tela. O da esquerda "piscará" a cada 64 caracteres (*bytes*) transferidos.

Etapa 4

Quando o computador terminar de copiar o programa, ele colocará na tela o sinal do Monitor-residente, novamente:

```
*?
```

Nota: Se o programa não estiver na fita, o computador continuará esperando por ele, mesmo após ter terminada a fita. Nesse caso, pressione a tecla BREAK para aparecer a mensagem:

```
READY
>
```

Instruções mais específicas são fornecidas com as próprias fitas. Podemos citar como exemplo uma instrução que faça o computador executar o programa Z 80 a partir do endereço 32000. Para tanto digite:

*?/32000 ENTER

Um outro caso seria iniciar a execução do programa a partir do endereço especificado na própria fita. Nesse caso, digite, simplesmente:

*? ENTER

Nota: a) Se você quiser copiar outro programa, prepare a fita e repita a etapa 3.

b) Pára retornar ao interpretador Basic basta apenas pressionar a tecla BREAK.

c)Caso queira executar o programa carregado, digite uma barra (/) seguida do *endereço de partida* e, então, pressione ENTER.

Capítulo 7

Usando uma Impressora

Qualquer impressora que disponha de uma interface paralela pode ser conectada ao seu CP 500. Também pode-se conectar uma impressora com interface serial, desde que se use o padrão RS 232 C.

É importante notar que, devido à grande variedade de modelos existentes, algumas funções diferem significativamente de impressora para impressora. Por isso, aconselhamos consultar o manual de cada modelo utilizado para poder explorar ao máximo suas possibilidades.

Impressora × Saída de Vídeo

A saída para a impressora é similar à de vídeo. De fato, as duas operações principais de saída para a tela têm suas equivalentes para a impressora, como é mostrado no quadro abaixo:

Vídeo	Impressora
PRINT LIST	LPRINT LLIST

Essas operações estão descritas na Seção de Linguagem Basic deste manual.

Ao interpretar uma instrução de impressão, o computador primeiramente "verifica" se a impressora está pronta para receber dados. Se não estiver, o computador simplesmente fica "a espera" de que isso aconteça. Durante essa "espera" não é possível realizar nenhuma outra operação com seu computador. Para reestabelecer o controle pelo teclado será necessário pressionar a tecla BREAK até que o computador mostre a mensagem:

```
READY
>
```

Algumas das características do vídeo do CP 500 não são possíveis na maioria das impressoras. Os caracteres gráficos, por exemplo, só são impressos pela P 500. No entanto, algumas impressoras dispõem de seu próprio conjunto de caracteres gráficos e especiais. As instruções CLS e PRINT @ não têm equivalente para impressão. Como já aconselhamos antes, consulte o manual da impressora para conhecer suas características.

Características de Controle da Impressora

O uso de uma impressora envolve o uso das seguintes variáveis:

— Largura máxima de cada linha (colunas por página);
— Comprimento da página (quantas linhas por página);
— Condição *(status)* da impressora (se ela está conectada e pronta para imprimir).

A seguir mostraremos como definir essas variáveis no CP 500.

FIXANDO A LARGURA MÁXIMA DA LINHA

O CP 500 permite especificar a largura máxima de linha da impressora. Quando uma linha exceder a essa especificação, o computador automaticamente vai inserir um final de linha (retorno de carro), para que o restante da linha seja impresso na próxima linha do formulário.

Existem fatores importantes que implicam a necessidade de uma escolha correta da largura da página, que é definida pela largura da linha.

O primeiro fator é a diferença que existe entre a largura da tela (64 ou 32 colunas) e a da impressora, que varia conforme o papel (geralmente, um formulário contínuo). No computador, qualquer texto que exceda a largura da tela continua normalmente na linha seguinte. Nas impressoras, o tratamento desses casos varia de modelo para modelo. Alguns modelos passam para a próxima linha automaticamente, outros podem perder o restante da linha e alguns podem até ter uma reação imprevista como, por exemplo, "travar" a impressão.

Outro fator é a própria largura do papel usado. Suponha que a sua impressora permita uma linha de 132 colunas e que você esteja usando um formulário com apenas 80 colunas. Se você quiser imprimir uma linha com mais de 80 caracteres, o final da linha será impresso sobre o cilindro da impressora, fora do papel!

O valor no endereço 16427 equivale à largura máxima menos dois. No caso de se querer a mesma largura que a tela, 64 colunas, este endereço deve conter o valor 62. Para colocar esse valor, você deve usar a instrução POKE da seguinte forma:

```
POKE 16427,62
```

Quando o valor do endereço é igual a 255, o controle da largura de linha é desativado. O computador deixa de inserir um retorno de carro, qualquer que seja o comprimento da linha a ser impressa. Lembre-se de que algumas impressoras fazem isso automaticamente.

O valor inicial desse endereço é 255.

CONTROLE DE PÁGINA

Em muitas aplicações de uma impressora é necessário controlar o número de linhas

---

Nota: Observe que omitimos propositamente a tecla ENTER após a instrução pois a essa altura você já deve saber que qualquer ordem ao computador só será executada após seu acionamento. Portanto, daqui para frente, não se esqueça de pressioná-la toda vez que introduzir uma linha.

impressas em uma página. Um exemplo típico é a impressão de relatórios, onde, após um certo número de linhas, se queira mudar de página. O CP500 mantém as seguintes informações sobre a impressão em andamento:

Informação	Endereço
Número de linhas da página mais um. Inicialmente contém o valor 67 (66+1).	16424
Número de linhas impressas na página atual, mais um. Inicia com o valor 1.	16425

A maioria das impressoras imprime seis linhas por polegada. O comprimento padrão de uma página (formulários contínuos) é de onze polegadas (279 mm), o que permite 66 linhas por página. Esse é o comprimento inicialmente adotado pelo CP 500. Ao usar papel ou impressora com características diferentes destas, você deve calcular o novo comprimento e realizar a alteração do endereço 16424. Se o papel comportar 88 linhas, execute a seguinte instrução:

```
POKE 16424,89
```

Antes de iniciar a impressão, ajuste o papel para que esta comece no início da página. Deste modo as variáveis de controle estarão corretas. A cada linha impressa o valor do endereço 16425 é incrementado de um, atualizando a contagem de linhas.

Para passar para a primeira linha da próxima página, a partir de qualquer ponto da página atual, imprima o código de controle 12, que executa essa função, da seguinte forma:

```
LPRINT CHR$(12)
```

Com esse comando, o papel avançará o número de linhas definido pela fórmula:

Avanço = *comprimento de página - contador de linhas*

Após executar esse comando o computador coloca o contador de linhas em sua condição inicial.

Às vezes é necessário mudar o contador de linhas (endereço 16425) como no caso de se avançar o papel manualmente. Para fazer isso, coloque o valor correspondente da seguinte forma:

```
POKE 16425,1 (começo de página)
```

VERIFICANDO A CONDIÇÃO *(STATUS)* DA IMPRESSORA

Ao contrário da tela, a impressora não está sempre a disposição do computador.

Nota: Se a largura de linha da impressora for menor que a especificada, a impressora poderá mudar automaticamente de linha (se possuir essa característica) sem que o computador envie esse comando. Nesse caso, a contagem de linha no computador estará errada. Para evitar que isso ocorra, certifique-se de que a largura fixada (endereço 16427) não ultrapasse a capacidade da impressora, que deve constar em seu manual de operação.

Ela pode estar desligada, desconectada do computador, sem papel e uma série de outras condições que impedem sua operação. Quando a impressora não pode receber dados, o computador suspende suas operações e fica à espera de que a impressora receba os dados. Nessa condição, seria de grande valia se o computador indicasse que não pode imprimir nada e parasse ou passasse para uma outra tarefa qualquer.

Para se conseguir esse resultado, basta verificar-se as condições da impressora antes de mandar imprimir. As condições são armazenadas na forma "binária" no endereço 14312. Como apenas alguns *bits* desse endereço são significativos (veja "Informações Técnicas"), o valor resultante deve ser comparado logicamente com um valor de referência. Esse valor de referência, para o CP 500, é o número 240, que, comparado com o conteúdo do endereço 14312 por uma instrução lógica AND, deve resultar em 48. O trecho de programa a seguir ilustra essa operação:

```
100 ST%=PEEK(14312) AND 240
110 IF ST%<>48 THEN PRINT "IMPRESSORA FORA DE CONDICAO"
:STOP
120 PRINT "IMPRESSORA EM CONDICAO"
```

Função de Reprodução da Tela

O CP 500 permite que seja tirado um "instantâneo" da tela, que é copiada na impressora. Essa função pode operar nos modos imediato, de execução, de edição e *system*. Não opera durante o acesso a periféricos como gravador, disquetes ou a própria impressora.

Quando você quiser copiar o conteúdo da tela, simplesmente pressione as teclas SHIFT ↓ *, simultaneamente. Em alguns equipamentos, essa função é realizada pela combinação das teclas S e P. O computador interromperá o que estiver fazendo e imprimirá todo o conteúdo da tela, inclusive espaços em branco. Caso você queira interromper a impressão, pressione a tecla BREAK no momento que a parte que lhe interessa tiver sido copiada.

Os caracteres especiais e gráficos serão substituídos por pontos. Como na impressão normal, o computador vai esperar que a impressora esteja pronta.

Nota: Essa função pode, ainda, ser ativada por uma chamada direta da sub-rotina gravada na EPROM pela função USR do Basic. Veja $PRSCRN, em "Informações Técnicas", para maiores detalhes.

Capítulo 8

Usando a Interface Serial RS 232 C

O que é uma Interface

É um meio de comunicação entre o seu computador e um outro equipamento, que fornece as convenções necessárias a respeito da identificação dos dados, da velocidade de transferência, das seqüências de envio-recepção, das técnicas de verificação de erro, etc.

A interface apenas "acopla", dentro de um determinado padrão, dois equipamentos distintos. Ela não incorpora a programação necessária para enviar e/ou receber informações.

Por exemplo, ter a interface instalada não permite, automaticamente, que você envie programas em Basic de um CP 500 para outro ou controle uma impressora via interface. Essas aplicações requerem *programas de controle* os quais devem ser criados de acordo com a necessidade do equipamento que se pretende usar.

A interface RS 232 C do CP 500 satisfaz as exigências das normas EIA (*Electronics Industries Association*). No entanto, não podemos garantir que opere normalmente com todos os equipamentos auto-denominados *compatíveis com RS 232 C*, nem tampouco nos comprometemos a fornecer suporte técnico, ou mesmo programas, para essas aplicações, ou situações de uso específico. Podemos garantir, entretanto, que essa interface funciona corretamente com todos os equipamentos da Prológica que usam esse mesmo padrão.

O termo RS 232 C se refere a uma norma específica da EIA, a qual descreve um método amplamente aceito de interligação de equipamentos terminais de dados com equipamentos de comunicação de dados. A interface RS 232 C é, sem dúvida, o padrão mundialmente mais difundido para comunicação entre equipamentos de processamento de dados. A maioria dos terminais de vídeo, modens, leitoras de cartão, impressoras seriais, mini e microcomputadores, etc., utilizam esse padrão para intercâmbio de informação.

Com a interface RS 232 C, seu CP 500 abre um mundo inteiramente novo de possibilidades, no qual você poderá se comunicar à distância com outros computadores, enviar informações, consultar bancos de dados, e muitas outras coisas que dependem da comunicação entre equipamentos.

Usando o CP 500 como Terminal

Provavelmente o uso mais comum da interface RS 232 C seja o que permite usar seu computador como terminal de outro computador. Nessa aplicação, tudo que você di-

---

Nota: As informações a seguir se aplicam apenas a computadores equipados com a interface RS 232 C.

gitar será enviado, pela RS 232 C, ao computador "central", e tudo que ele enviar será mostrado na tela do seu "terminal".

Antes de prosseguir com detalhes de operação da RS 232 C, mostraremos um programa Basic que permite uma operação **simplificada** do CP 500 como terminal. Para executar esse programa, é imprescindível a existência de um computador central, e as seguintes etapas devem ser cumpridas:

Etapa 1

Certifique-se de que as características de sua interface coincidem com as do computador central. Se for o caso, altere-as como é explicado mais adiante, nesse mesmo capítulo.

Etapa 2

Conecte o CP 500 ao central via RS 232 C. Será necessário um modem telefônico, ou outro equipamento, dependendo do meio de comunicação utilizado.

Etapa 3

Digite e rode (execute) o programa Basic a seguir. Não é necessário digitar os comentários (explicações após os apóstrofos). O computador mostrará os caracteres recebidos e enviará os que forem digitados. Esse programa é demonstrativo apenas, não servindo como monitor de um terminal prático.

Notar que na linha 160 não existe espaço algum entre aspas.

```
5 DEFINT A-Z                    'DEFINE INTEIROS P/ AGILIZAR
10 POKE 16890,0                 'NAO ESPERA POR E/S SERIAL
15 POKE 16888,(2*16)+2          'VELOCIDADE DE 110 BAUDS
17 U1=16526                     'BYTE MENOS SIGNIFICATIVO (LSB)
18 U2=16527                     'BYTE MAIS  SIGNIFICATIVO (MSB)
20 POKE U1,90                   'FIXA LSB P/ CHAMADA USR
30 POKE U2,0                    'FIXA MSB P/ CHAMADA USR
40 X=USR(0)                     'CHAMADA DE $RSINIT
50 RCV=80                       'MSB DE $RSRCV
60 TX=85                        'LSB DE $RSTX
70 CI=16872                     'ENDERECO DO BUFFER DE ENTRADA
80 CO=16880                     'ENDERECO DO BUFFER DE SAIDA
100 POKE U1,RCV                 'PREPARA CHAMADA DE $RSRCV
110 X=USR(0)                    'CHAMADA DE $RSRCV
120 C$=CHR$(PEEK(CI))           'VERIFICA O BUFFER DE ENTRADA
130 PRINT C$;                   'SE C$="" NADA ACONTECE
150 C$=INKEY$
160 IF C$="" THEN 100           'SEM TECLA, VOLTA 'A SERIAL
165 PRINT C$;                   'MOSTRA CARACTERE DIGITADO
170 POKE CO,ASC(C$)             'POE CARACTERE BUFFER DE SAIDA
180 POKE U1,TX                  'PREPARA CHAMADA DE $RSTX
190 X=USR(0)                    'CHAMADA DE $RSTX
200 GOTO 100                    'VOLTA PARA ENTRADA SERIAL
```

Nota: Para esse programa Basic deve ser usada a velocidade de 110 baud. Um programa Z 80 equivalente poderá usar qualquer outra velocidade.

Programando a Interface RS 232 C

Trataremos agora da RS 232 C como qualquer outro dispositivo de entrada/saída de dados e mostraremos como seu programa em Basic pode usá-la.

Em "Informações Técnicas" mostramos como usá-la em um programa Z 80, bem como detalhes sobre conversão de sinais e teoria de operação dessa interface.

SELECIONANDO AS CARACTERÍSTICAS DA RS 232 C

Antes de usar a interface para se comunicar com outro equipamento, certifique-se de que ela está compatível com as exigências deste. Assim, comece por obter as informações, relacionadas a seguir, sobre o equipamento com o qual deseja se comunicar. Na coluna da direita estão os valores mais utilizados.

Característica	*Valores típicos*
Velocidade de transferência (bauds)	110, 150, 300, 600, 1200, 2400, 4800, 9600
Comprimento da palavra (bits)	5, 6, 7, 8
Paridade	par, ímpar, nenhuma
Bits de parada	1, 2

Quando você ligar o computador, ou usar RESET , este selecionará as seguintes características iniciais:

Característica	*Valor inicial*
Velocidade de transferência	300 bauds
Comprimento da palavra	8 *bits*
Paridade	nenhuma
Bits de parada	1

Além das características já expostas, a RS 232 C possui outra muito importante. Antes de retornar o comando para o programa, ou para o teclado, ela espera pelo término das operações de entrada ou de saída em andamento. Isto é, o computador aguarda que um caractere seja recebido durante uma operação de entrada, ou que o equipamento a ele ligado esteja preparado para receber os dados numa operação de saída.

Durante a espera o controle pelo teclado pode ser retomado usando-se a tecla BREAK . Essa deve ser mantida pressionada até que a mensagem READY apareça na tela.

COMO MUDAR AS CARACTERÍSTICAS DA RS 232 C

Se as características inicialmente selecionadas pelo CP 500 não coincidirem com as do equipamento a ele interligado, essas poderão, total ou parcialmente, ser alteradas. Para tanto existe uma sub-rotina em EPROM de inicialização da RS 232 C, denominada $RSINIT. Essa sub-rotina utiliza parâmetros armazenados em posições definidas da RAM. Mudando-se os conteúdos dessas posições e "chamando-se" essa sub-rotina, altera-se as características da interface serial.

Os endereços e respectivos conteúdos são os seguintes:

Endereço	*Conteúdo*
16888	Código de velocidade de transferência (envio/recepção).
16889	Código de paridade/comprimento da palavra/*bits* de parada.
16890	Chave de espera.

Código de Velocidade de Transmissão/Recepção: O CP 500 pode receber e transmitir dentro de uma ampla gama de velocidade. Para a maior parte das aplicações as velocidades de recepção e de transmissão são as mesmas. Para cada velocidade existe um código que deve ser armazenado no endereço 16888.

As velocidades possíveis, com respectivos códigos e erros percentuais, estão na tabela a seguir. Escolha os códigos apropriados de transmissão e de recepção e, então, coloque-o na posição 16888 da memória obedecendo a seguinte fórmula:

Código transmissão/recepção = *(código de transmissão* * 16*)* + *código de recepção*

Tomemos um exemplo prático. Suponhamos que se queira acoplar o CP 500 à uma rede de processamento de dados que opere a uma velocidade de 110 bauds, tanto para transmissão quanto para recepção.

Vemos na tabela que o código para essa velocidade é dois (2). Substituindo os códigos da fórmula acima por esse valor, teremos:

Código de transmissão/recepção = (2 * 16) + 2 = 32 + 2 = 34

Em termos técnicos, estamos colocando o código de transmissão nos quatro *bits* mais significativos do *byte* no endereço 16888, e o código de recepção, nos quatro *bits* menos significativos desse mesmo *byte.* Esses dois conjuntos de quatro *bits* nos quais se pode dividir um *byte* são muitas vezes chamados de *nibbles.*

Códigos de Velocidade de Transmissão/Recepção

Velocidade desejada (baud rate)	*Erro (%)*	*Código de velocidade*
50	0	0
75	0	1
110	0	2
134.5	0.016	3
150	0	4
300	0	5
600	0	6
1200	0	7
1800	0	8
2000	0.253	9

Velocidade desejada (baud rate)	*Erro (%)*	*Código de velocidade*
2400	0	10
3600	0	11
4800	0	12
7200	0	13
9600	0	14
19200	3.125	15

Código de Paridade/Comprimento da palavra/*Bits* de parada: Para colocar todas essas informações em um único *byte* deve ser utilizada a seguinte fórmula:

código = (*par**128) + (*pal**32) + (*bits**16) + (*hab**8) + (*trans**4) + (*DTR**2) + *RTS*

onde as variáveis podem assumir os seguintes valores:

par (paridade)
0 para ímpar
1 para par

pal (palavra)
0 para palavras de 5 *bits*
1 para palavras de 6 *bits*
2 para palavras de 7 *bits*
3 para palavras de 8 *bits*

bits
0 para um *bit* de parada
1 para dois *bits* de parada

hab
(habilitação de paridade)
0 para habilitar
1 para desabilitar

trans
(transmissão)
1 para ativar transmissão
0 para desativar transmissão

DTR
0 para desativar o sinal indicativo de terminal de dados pronto (*DTR - data terminal ready*), colocando-o em nível baixo (0)
1 para ativar o sinal DTR, colocando-o em nível alto

RTS
0 para desativar o sinal que indica um pedido de transmissão (RTS - *request to send*), colocando-o em nível baixo (0)
1 para ativar o sinal RTS, colocando-o em nível alto (1)

Como exemplo, podemos supor que sejam as seguintes as características do equipamento ao qual você quer interligar seu CP 500:

paridade par

palavra de sete *bits*
dois *bits* de parada
transmissão ativa
DTR e RTS ativos (nível 1)

Para calcular o valor a ser armazenado, substitua os códigos relativos a cada característica na fórmula, que deve ficar da seguinte forma:

código = (1*128) + (2*32) + (1*16) + (0*8) + (1*4) + (1*2) + (1*1) = 215

Maiores informações sobre como determinar as características adequadas podem ser obtidas em "Informações Técnicas".

Chave de Espera: O CP 500 pode realizar as operações de entrada/saída de dados com ou sem espera. Isso quer dizer que o computador, dependendo do modo de operação, pode ou não esperar pela conclusão da operação determinada. Por exemplo, se for chamada a rotina de recepção de dados e o modo for de espera, o computador só retornará ao programa Basic após receber algum dado. Se o modo for invertido o computador retornará imediatamente, tenha ou não recebido algo.

A tecla BREAK interrompe a espera, retornando o controle ao teclado.

O conteúdo do endereço 16890 determina qual o modo de operação, funcionando como uma "chave de espera". A chave é interpretada da seguinte forma:

Conteúdo	*Modo de operação*
nulo	sem espera
não-nulo	com espera

ENTRADA/SAÍDA PELA INTERFACE RS 232 C

Se as características iniciais estiverem corretas, você poderá proceder à entrada ou saída serial dos dados.

Caso sejam necessárias alterações nas características, consulte, neste capítulo, o tópico "Como Mudar as Características da RS 232 C".

Existem duas sub-rotinas gravadas na EPROM que realizam, especificamente, as operações de entrada e de saída serial. Ambas foram utilizadas no programa de exemplo e são as seguintes:

$RSTX	Envia um caractere
$RSRCV	Recebe um caractere

Essas sub-rotinas podem ser utilizadas em um programa Basic graças à função USR (abreviação de USer Routine, rotina do usuário), usada no programa de exemplo. A função USR permite que rotinas Z 80 sejam integradas a programas em Basic. Isso permite aproveitar as facilidades de programação em Basic com a velocidade de processamento da linguagem Z 80. Essa função é discutida em detalhes nos capítulos "Características Especiais" e "Informações Técnicas".

Nota: Para mudar as características da interface, você deve realizar uma chamada da sub-rotina $RSINIT após preparar os conteúdos dos endereços 16888 a 16890.

Transmissão: Para transmitir um caractere, observe a seqüência abaixo.

Etapa 1

O computador deve estar conectado ao equipamento através da interface serial.

Etapa 2

Defina uma chamada USR para $RSTX (endereço 85) executando as seguintes instruções:

```
POKE 16526,85
POKE 16527,0
```

Etapa 3

Envie o caractere colocando seu código ASCII na posição de memória 16880. Suponha, por exemplo, que A$ contenha o caractere. A instrução POKE 16880,ASC(A$) executa essa tarefa.

Etapa 4

Faça a chamada USR usando uma variável e um argumento auxiliares. Esses parâmetros apenas satisfazem a sintaxe da função USR, nesse caso. A chamada pode ficar da seguinte forma:

`X=USR(0)` (X e 0 apenas completam a sintaxe)

O computador deve esperar que você pressione uma tecla. E para que isso ocorra, é necessária uma ordem específica. Se essa ordem não for dada, o controle retornará ao interpretador, mesmo que não tenha sido transmitido nenhum caractere.

Etapa 5

Repita os itens 3 e 4 até que todos os caracteres da informação a ser transmitida tenham sido enviados.

Recepção: A seqüência descrita a seguir apresenta o modo correto de se receber um caractere através da interface serial.

Etapa 1

O computador deve estar ligado ao equipamento serial que irá transmitir os dados.

Etapa 2

Defina uma chamada USR para $RSRCV (endereço 80) executando as seguintes instruções:

```
POKE 16526,80
POKE 16527,0
```

Etapa 3

Receba o caractere executando a chamada USR. Nesse caso também deve-se usar parâmetros auxiliares para variável e argumento da chamada. Por exemplo:

```
X=USR(0)
```

Essa sub-rotina coloca o código ASCII do caractere recebido na posição 16872 da memória. O valor zero indica que não foi recebido caractere algum. O retorno dessa sub-rotina se processa da mesma maneira que para o envio. Portanto, para interromper uma espera pressione BREAK .

Etapa 4

Para utilizar esse caractere, coloque-o em uma variável com uma instrução do tipo:

```
A$=CHR$(PEEK(16872))
```

Lembre-se de que nenhum caractere foi recebido se A$ for igual a CHR$(0).

Etapa 5

Repita os itens 3 e 4 até que todos os caracteres referentes à informação a ser recebida tenham sido transmitidos. A recepção pode ainda ser interrompida quando um caractere de fim de transmissão (que pode ser convencionado pelos operadores dos equipamentos envolvidos) for recebido. Maiores detalhes serão discutidos oportunamente.

CHAMANDO $RSINIT DE UM PROGRAMA BASIC

Coloque com a instrução POKE os valores desejados nos endereços de controle da RS 232 C (16888-16890). Se as características estiverem em parte corretas, deixe os endereços correspondentes inalterados. Se for necessário mudar o código de paridade, veja como proceder consultando $RSINIT em "Informações Técnicas".

Uma vez tendo calculado e armazenado os códigos de forma apropriada, você estará pronto para realizar a chamada da sub-rotina $RSINIT.

Para definir uma chamada USR para $RSINIT, execute as seguintes instruções:

```
POKE 16526,90
POKE 16527,0
X=USR(0)
```

Quando estas instruções forem executadas, a interface RS 232 C será inicializada de acordo com os códigos armazenados nos endereços de 16888 a 16890.

Capítulo 9

Desviando Entradas e Saídas

O CP 500 permite que o endereçamento de seus periféricos seja desviado, fazendo, por exemplo, com que uma instrução endereçada à tela seja enviada para a impressora. Essa característica faz com que seus programas se tornem mais flexíveis, eliminando a necessidade de rotinas específicas para cada periférico.

Um caso típico é a depuração de programas de emissão de listagens. Pode-se testar o programa utilizando a tela, sem gastar formulário, realizando as alterações necessárias para que a listagem saia na forma desejada. Quando o programa estiver correto, basta desviar a saída da tela para a impressora e o programa passará a emitir a listagem nesse periférico.

Isso é feito sem alterar o programa, utilizando-se a capacidade de desvio de entradas e saídas (E/S) do CP 500. Após o desvio, o periférico original (no exemplo, a tela), será equivalente ao periférico especificado (a impressora).

Essa condição se manterá até que a sub-rotina de inicialização dos periféricos ($INITIO), recoloque todas as (E/S) em suas condições originais.

A seguir listamos os periféricos (e abreviaturas) que podem ser alterados.

Periférico	*Abreviatura*
teclado	KI
tela	DO
impressora	PR
RS 232 C	
transmissão	RO
recepção	RI

Como Desviar uma Entrada ou Saída

Para facilitar a explicação, utilizaremos o exemplo de desviar o endereçamento da tela para a impressora.

Etapa 1

Armazene a abreviatura do periférico original (DO para a tela), nas posições 16930/16931 da memória, da seguinte forma:

```
POKE 16930,ASC("D")
POKE 16931,ASC("O")
```

Nota: Para acompanhar este tópico, é necessário que uma impressora esteja conectada ao computador.

Etapa 2

Armazene a abreviatura do periférico desejado nas posições 16928/16929 da memória, da seguinte forma:

```
POKE 16928,ASC("P")
POKE 16929,ASC("R")
```

Etapa 3

Prepare uma chamada USR para $ROUTE (endereço 108), usando as instruções seguintes:

```
POKE 16526,108
POKE 16527,0
```

Etapa 4

Execute a chamada USR, completando a sintaxe dessa função com parâmetros auxiliares como, por exemplo:

```
X=USR(0)
```

Com a realização da etapa 4 o desvio estará concluído. A partir daí, tudo que for endereçado à tela será desviado para a impressora.

Desviando Várias Entradas e Saídas

Para desviar o endereçamento de várias entradas e saídas, as etapas 1 a 4 devem ser repetidas para cada desvio individual. Entretanto, deve ser obedecida uma ordem coerente para que o resultado seja o esperado. Se um periférico A tem seu endereçamento desviado para um periférico B, um desvio posterior para A será desviado para B. Para explicar melhor, vamos enumerar essa seqüência:

Passo 1

Desviar de A para B.

Passo 2

Desviar de C para A.

Essa seqüência faz com que as operações em C sejam realizadas em B, e não em A, como o segundo desvio propõe. É o mesmo que fazer o seguinte:

Passo 1

Desviar de A para B.

Passo 2

Desviar de C para B.

Para que A passe a ser considerado como B, e C como A, a seqüência correta é:

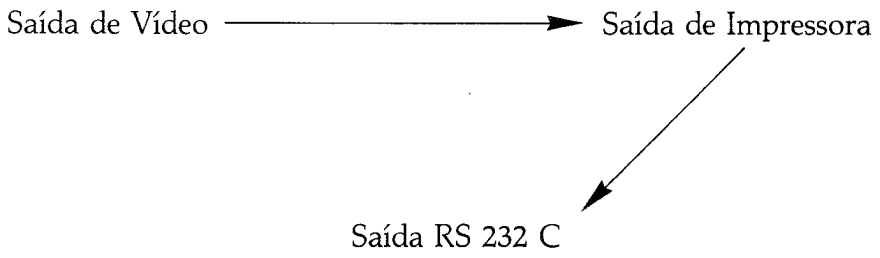
Passo 1

Desviar de C para A.

Passo 2

Desviar de A para B.

Como exemplo, suponha que você queira usar as instruções de vídeo para a impressora, e as da impressora para a saída serial (RS 232 C). O diagrama do que deve ser feito é o seguinte:



Por esse diagrama vemos que a saída de vídeo vai para a impressora e a da impressora, para a interface RS 232 C. As demais saídas e entradas não são afetadas. Esses desvios devem ser realizados na seguinte ordem:

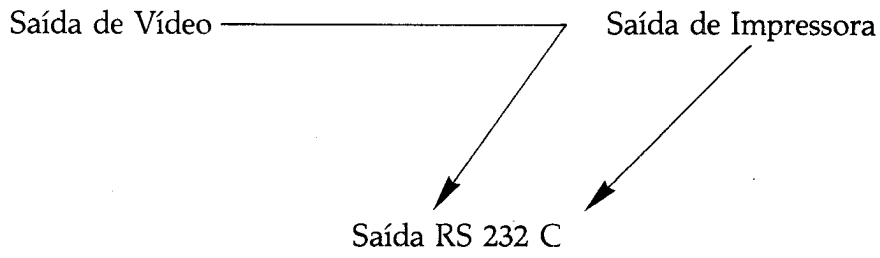
Passo 1

Desvie DO para PR.

Passo 2

Desvie PR para RO.

Note que a saída de vídeo não deve ser desviada da impressora para a interface. Isso acontecerá se a ordem de desvios for alterada. O resultado será, então:



Nesse caso, a saída de vídeo realmente será transferida da impressora para a saída serial, o que vale dizer que todos dados enviados para a tela sairão pela interface serial, assim como os dados enviados para a impressora.

Capítulo 10

Relógio Interno

O CP 500 incorpora um relógio interno capaz de fornecer as horas e a data. Ele está quase sempre em funcionamento, parando apenas durante as operações de acesso ao gravador ou ao disquete, e em outros casos específicos.

O relógio mantém as seguintes informações na memória:

Dado	*Abreviatura*	*Faixa*	*Posição na memória*
Mês	MM	01-12	16924
Dia	DD	01-31	16923
Ano	AA	00-99	16922
Hora	HH	00-23	16921
Minuto	MN	00-59	16920
Segundo	SS	00-59	16919

Possui também lógica para meses de 28, 30 e 31 dias, mas não reconhece anos bissextos.

Ao ligar o computador, os endereços com as informações do relógio são preenchidos por zeros deixando a hora e a data da seguinte forma:

Data inicial 00/00/00
Hora inicial 00:00:00

Como Acertar a Hora e a Data

Simplesmente armazene os dados nos respectivos endereços. O programa a seguir realiza essa operação:

```
10 DEFINT A-Z
20 DIM TM(5)
30 CL=16924
40 PRINT "DIGITE OS 6 VALORES:MM,DD,AA,HR,MN,SS"
50 INPUT TM(0),TM(1),TM(2),TM(3),TM(4),TM(5)
60 FOR I=0 TO 5
70 POKE CL-I,TM(I)
80 NEXT I
90 PRINT "O RELOGIO ESTA CORRETO"
100 END
```

Como Ler a Hora e a Data

O CP 500 possui uma função em Basic (TIME$) que fornece a data e a hora em uma *string* de dezessete *bytes*. Para saber as horas, basta colocar essa *string* na tela com uma instrução PRINT como esta:

```
PRINT TIME$
```

Como Fixar a Hora na Tela

Você pode fazer com que a hora permaneça na tela. Nesse caso, apenas a hora certa (sem a data) aparecerá, ocupando as colunas de 57 a 64 da linha superior, sempre que o relógio estiver operando.

A sub-rotina $CLKON (endereço 664) coloca a hora na tela, enquanto $CLKOFF retira, mas sem desativar o relógio. O programa abaixo controla o aparecimento da hora.

```
10 DEFINT A-Z
20 AT=152:DS=161                    'LSB DE $CLKON E $CLKOFF
30 PRINT "<C> COLOCA HORA NA TELA"
40 PRINT "<R> RETIRA HORA DA TELA"
50 INPUT A$
60 IF A$="C" THEN SW=AT:GOTO 100
70 IF A$="R" THEN SW=DS:GOTO 100
80 GOTO 30
100 POKE 16526,SW
110 POKE 16527,2                    'MSB E' O MESMO
120 X=USR(0)                        'CHAMADA USR
130 END
```

Consulte "Informações Técnicas" para maiores informações sobre essas sub-rotinas do CP 500.

Nota: O endereço de chamada da sub-rotina deve ser armazenado em dois *bytes*. Portanto o valor decimal do endereço deve ser transformado em dois valores, na faixa de 0 a 235, que possam ser armazenados na memória.

A transformação obedece às seguintes fórmulas:

MSB = parte inteira de (endereço/256)
LSB = endereço - (MSB*256)

onde:

MSB	é o *byte* mais significativo do endereço (MSB são as iniciais de *Most Significant Byte*)
LSB	é o *byte* menos significativo do endereço *(Least Significant Byte)*
endereço	é o endereço, em decimal, do início da sub-rotina na EPROM.

Vamos usar como exemplo o endereço da sub-rotina $CLKON, que é 664.

MSB = parte inteira de (664/256) = 2
LSB = 664 - (2*256) = 152

Capítulo 11

Inicialização de Entradas e Saídas

Sempre que o computador for ligado, ou for pressionada RESET , as sub-rotinas de entrada/saída (*"I/O drives"*) retomam suas condições iniciais, como está explicado nos capítulos seguintes. Na condição inicial da tela, por exemplo, o cursor é intermitente.

Como foi descrito nos capítulos anteriores, existem maneiras de se alterar essas condições iniciais através de programas Basic ou Z 80. Por isso é importante haver um modo de recolocar todas as E/S em suas condições iniciais.

O CP 500 possui uma sub-rotina na EPROM que realiza essa função. Essa sub-rotina é chamada $INITIO e já apareceu em alguns programas Basic dos capítulos anteriores. O programa abaixo demonstra como utilizá-la.

```
10 POKE 16526,105          'LSB DA CHAMADA DE $INITIO
20 POKE 16527,0            'MSB DA CHAMADA DE $INITIO
30 X=USR(0)                'CHAMADA DE $INITIO
```

Capítulo 12

Informações Técnicas

Este capítulo é destinado aos programadores Basic e Z 80 que estejam familiarizados com aritmética binária e hexadecimal e com noções de *hardware*, como *bit* e *byte*. As informações aqui apresentadas têm o propósito de permitir ao programador controlar todas as vantagens que o computador oferece.

Limitação da Memória RAM

Em muitas aplicações é necessário interligar programas Basic com outros em linguagem de máquina. Quando isso ocorrer uma parte da memória deverá ser separada para acomodar o programa Z 80. De outra forma o interpretador Basic poderá ocupar toda a memória para o armazenamento de programas Basic e de dados.

Durante o "diálogo inicial" você tem a opção de realizar essa separação através da pergunta "Mem usada?". Pressionando apenas a tecla ENTER em resposta a essa pergunta, o computador entende que deve usar toda a memória.

Para separar uma área para programas Z 80, digite o *endereço limite*, na forma decimal, que o interpretador Basic poderá usar, seguido de ENTER. O *endereço limite* se refere ao último *byte* disponível na memória para armazenar programas e dados relacionados com a linguagem Basic. Digitando 32667 ENTER, por exemplo, o interpretador Basic não poderá utilizar nenhum *byte* acima do endereço 32667. Dessa forma, a área acima desse endereço fica reservada (protegida) para os programas em linguagem de máquina que forem carregados.

Sub-rotinas da EPROM

O CP 500 contém muitas sub-rotinas gravadas na EPROM, que podem ser chamadas de dentro de um programa em Basic, através da função USR.

A listagem a seguir dispõe as sub-rotinas da EPROM de acordo com a função. Nas descrições, mais à frente, elas estão em ordem alfabética.

Controle do Sistema

Sub-rotina	*Função*
$CLKON	coloca a hora na tela
$CLKOFF	retira a hora da tela
$DATE	fornece a data
$DELAY	retarda por um tempo definido

$INITIO	inicializa os periféricos (E/S)
$READY	coloca na condição READY
$RESET	inicializa o computador
$ROUTE	desvia o endereçamento de E/S
$SETCAS	especifica a velocidade de transferência
$TIME	fornece a hora certa

Gravador

Sub-rotina	*Função*
$CSHIN	liga o gravador, procura pelo "prefixo" e pelo *byte de sincronismo*
$CSIN	entra um *byte*
$CSOFF	desliga o gravador
$CSHWR	liga o gravador, grava o prefixo e o *byte de sincronismo*
$CSOUT	grava um *byte*

Teclado

$KBCHAR	quando possível, "pega" um caractere
$KBWAIT	espera por um caractere
$KBLINE	espera por uma linha
$KBBRK	verifica apenas a tecla *BREAK*

Vídeo

$VDCHAR	coloca um caractere na tela
$VDCLS	limpa a tela
$VDLINE	coloca uma linha na tela

Impressora

$PRCHAR	imprime um caractere
$PRSCN	imprime o conteúdo da tela

Interface Serial RS 232 C

$RSINIT	inicializa
$RSRCV	recebe um caractere
$RSTX	transmite um caractere

A seguir descreveremos todas as sub-rotinas em ordem alfabética. Para facilitar as consultas, todas as sub-rotinas são descritas conforme exposto na nota da página 66 .

Os programas Z 80 demonstrativos são todos parte de um único programa em Assembly, como se poderá notar pela numeração das linhas (terceira coluna). O primeiro trecho, listado a seguir, define os endereços de entradas das várias sub-rotinas utilizadas.

End. *Instr.* *Linha* *Label* *Opcode* *Argumentos* *Comentário*

```
                00001
                00002 ;                        **C P  5 0 0 **
                00003 ;
                00004 ;PROGRAMA DEMONSTRATIVO DE CHAMADAS DE SUB-ROTINAS DA EPROM
                00005 ;
                00006 ;COMO DEMONSTRACAO, VOCE DEVE EXECUTAR UM SALTO PARA O
                00007 ;PONTO DE ENTRADA APROPRIADO. CADA DEMONSTRACAO TERMINA COM
                00008 ;UM SALTO PARA A CONDICAO DE "READY"
                00009 ;
                00010 ;
0000            00011 RESET  EQU  0000H
002B            00012 KBCHAR EQU  002BH
0033            00013 VDCHAR EQU  0033H
003B            00014 PRCHAR EQU  003BH
0040            00015 KBLINE EQU  0040H
0049            00016 KBWAIT EQU  0049H
0050            00017 RSRCV  EQU  0050H
0055            00018 RSTX   EQU  0055H
005A            00019 RSINIT EQU  005AH
0060            00020 DELAY  EQU  0060H
0069            00021 INITIO EQU  0069H
006C            00022 ROUTE  EQU  006CH
01C9            00023 VDCLS  EQU  01C9H
01D9            00024 PRSCN  EQU  01D9H
01F8            00025 CSOFF  EQU  01F8H
021B            00026 VDLINE EQU  021BH
0235            00027 CSIN   EQU  0235H
0264            00028 CSOUT  EQU  0264H
0287            00029 CSHWR  EQU  0287H
028D            00030 KBBRK  EQU  028DH
0296            00031 CSHIN  EQU  0296H
0298            00032 CLKON  EQU  0298H
02A1            00033 CLKOFF EQU  02A1H
1A19            00034 READY  EQU  1A19H
3033            00035 DATE   EQU  3033H
3036            00036 TIME   EQU  3036H
3042            00037 SETCAS EQU  3042H
37E8            00038 PRSTAT EQU  37E8H
                00039 ;-------------------------------------------------------------
8000            00040        ORG  8000H
                00041 ;
```

$CLKOFF - 673D/02A1H

Retira a hora da tela.

Condições de Entrada: Nenhuma.

Condições de Saída: A é alterado e os demais registradores não sofrem alteração.

Programa Z 80 de exemplo:

```
                00042 ;RETIRA A HORA DA TELA
8000  CDA102      00043        CALL CLKOFF
8003  C3191A      00044        JP   READY
```

Nota: Este programa demonstrativo não está completo, podendo conter mais pontos de entrada para chamada das demais sub-rotinas da EPROM.

Programa Basic de exemplo:

```
100 POKE 16526,161:POKE 16527,2     'BYTES - E + SIGNIF.
110 X=USR(0)                        'ARGUMENTO AUXILIAR
```

$CLKON - 664D/0298H

Coloca a hora na tela.

Condições de Entrada: Nenhuma.

Condições de Saída: O registrador A é alterado. Os demais não são afetados.

Programa Z 80 de exemplo:

```
                00045 ;COLOCA A HORA NA TELA
8006  CD9802    00046         CALL CLKON
8009  C3191A    00047         JP   READY
```

Programa Basic de exemplo:

```
100 POKE 16526,152:POKE 16527,2    'BYTES - E + SIGNIF.
110 X=USR(0)                       'ARGUMENTO AUXILIAR
```

$CSHIN - 662D/0296H

Procura por um prefixo na fita e por um *byte de sincronismo*. Cada gravação na fita começa com um prefixo *("heading")*, constituído por uma seqüência-guia e é seguido de um *byte de sincronismo*. Essa sub-rotina liga o motor do gravador e começa a procurar por essa informação inicial. Ela retorna para o programa de chamada assim que o *byte de sincronismo* tenha sido encontrado.

Condições de Entrada: Nenhuma.

Condições de Saída: O registrador A é alterado. Os demais não são afetados.

Nota: As descrições das sub-rotinas da EPROM são compostas pelos seguintes itens:

$NOME - endereço de entrada

O nome *($NOME)* da rotina serve apenas como referência.
Ele não é reconhecido pelo computador.
O *endereço de entrada* é fornecido em decimal (00000D) e em hexadecimal (0000H), separados por uma barra.
Abaixo do nome aparece uma exposição de como a sub-rotina realiza seu trabalho.

Condições de Entrada e de Saída

As condições de entrada e saída são fornecidas de acordo com o efeito de cada sub-rotina sobre o registradores do microprocessador. Quando não forem mencionadas, significa que a sub-rotina não os altera.

Programa-exemplo

Programas-exemplo são fragmentos ilustrativos, em geral em linguagem *Assembly* e, quando apropriado, em Basic.

Programa Z 80 de exemplo: O programa abaixo lê a gravação criada pelo programa de exemplo da sub-rotina $CSHWR.

```
                00048 ;LE UMA MENSAGEM NA FITA E PARA EM UM RETORNO DE CARRO
800C  CDC901    00049         CALL VDCLS          LIMPA A TELA
800F  3E0D      00050         LD   A,0DH
8011  CD3300    00051         CALL VDCHAR         SALTA UMA LINHA
8014  CD4230    00052         CALL SETCAS         SELECIONA VEL. DE TRANSF.
8017  213B80    00053         LD   HL,MSG0        (HL)= INICIO DA MENSAGEM MSG0
801A  CD1B02    00054         CALL VDLINE
801D  CD4900    00055         CALL KBWAIT         ESPERA POR UMA TECLA
8020  216280    00056         LD   HL,TXT         (HL)=INICIO DO BUFFER
8023  CD9602    00057         CALL CSHIN          BUSCA INICIO DA GRAVACAO
8026  CD3502    00058 LOOP    CALL CSIN           CARREGA UM BYTE
8029  77        00059         LD   (HL),A         ARMAZENA-O NO BUFFER
802A  23        00060         INC  HL             INDICA A PROXIMA POSICAO
802B  FE0D      00061         CP   0DH            COMPARA COM RETORNA DE CARRO:
802D  20F7      00062         JR   NZ,LOOP          SE NAO FOR, PEGA O PROXIMO
802F  CDF801    00063         CALL CSOFF            SE FOR, DESLIGA O GRAVADOR.
8032  216280    00064         LD   HL,TXT         COLOCA A MENSAGEM NA TELA
8035  CD1B02    00065         CALL VDLINE
8038  C3191A    00066         JP   READY          E TERMINA
803B  50        00067 MSG0    DEFM 'PREPARE A FITA E DIGITE QUALQUER TECLA'
8061  0D        00068         DEFB 0DH
8062            00069 TXT     DEFS 256            BUFFER DE 256 BYTES PARA
                                                  ;RECEBER MENSAGEM DA FITA
```

$CSIN - 565D/0235H

Carrega um *byte* da fita no gravador. Após concluída $CSHIN, utilize $CSIN para carregar (copiar na memória) os dados gravados, um *byte* por vez.

Condições de Entrada: Nenhuma.

Condições de Saída: O conteúdo de A será o *byte* carregado.

Programa Z 80 de exemplo: Veja $CSHIN.

$CSHWR - 647D/0287H

Grava o prefixo e o *byte de sincronismo*. Cada gravação na fita começa com um prefixo que consiste de uma seqüência-guia seguida de um *byte de sincronismo*. A sub-rotina $CSHWR aciona a fita e grava esse prefixo.

Condições de Entrada: Nenhuma.

Condições de Saída: Apenas o registrador A é alterado.

Programa Z 80 de exemplo:

Ver programa da página a seguir.

Nota: O seu programa deve chamar essa sub-rotina quantas vezes forem necessárias para se manter a velocidade de transferência (500 ou 1500 bauds).

```
                00070 'INTRODUZ UMA MENSAGEM PELO TECLADO E A GRAVA
8162  CDC901    00071         CALL VDCLS
8165  3E0D      00072 LOOP1   LD   A,0DH          O RETORNO DO CARRO FAZ
8167  CD3300    00073         CALL VDCHAR         SALTAR PARA A PROXIMA LINHA.
816A  21A081    00074         LD   HL,MSG1         A MENSAGEM DE INTRODUCAO E'
816D  CD1B02    00075         CALL VDLINE          COLOCADA NA TELA.
8170  21EA81    00076         LD   HL,TXT1        BUFFER DE ENTRADA.
8173  06FF      00077         LD   B,255          MAXIMO DE 255 CARACTERES.
8175  CD4000    00078         CALL KBLINE         PEGA UMA LINHA DO TECLADO.
8178  38EB      00079         JR   C,LOOP1        VAI P/ LOOP1 SE HOUVE BREAK.
817A  3E0D      00080         LD   A,0DH
817C  CD3300    00081         CALL VDCHAR         SALTA UMA LINHA.
817F  CD4230    00082         CALL SETCAS         SELECIONA A VEL. DE TRANSF.
8182  21B381    00083         LD   HL,MSG2        MENSAGEM FINAL.
8185  CD1B02    00084         CALL VDLINE
8188  CD4900    00085         CALL KBWAIT         ESPERA POR UMA TECLA.
818B  CD8702    00086         CALL CSHWR          GRAVA O PREFIXO.
818E  21EA81    00087         LD   HL,TXT1        (HL)=MENSAGEM
8191  7E        00088 LOOP2   LD   A,(HL)         A=BYTE ASCII DO CARACTERE.
8192  23        00089         INC  HL             INDICA PROXIMO CARACTERE.
8193  CD6402    00090         CALL CSOUT          GRAVA O BYTE ATUAL NA FITA.
8196  FE0D      00091         CP   0DH            E'UM RETORNO DO CARRO ?
8198  20F7      00092         JR   NZ,LOOP2        NAO. ENTAO PEGA O PROXIMO BYTE
819A  CDF801    00093         CALL CSOFF           SIM. DESLIGA O GRAVADOR.
819D  C3191A    00094         JP   READY
81A0  49        00095 MSG1    DEFM 'INTRODUZA MENSAGEM'
81B2  0D        00096         DEFB 0DH
81B3  51        00097 MSG2    DEFM 'QUANDO PRONTO PARA GRAVAR, PRESSIONE QUALQUER TECLA'
81E9  0D        00098         DEFB 0DH            FIM DA LINHA (RETORNO DE CARRO)
81EA            00099 TXT1    DEFS 256
```

$CSOFF - 504D/01F8H

Desliga o gravador. Após gravar os dados na fita, chame esta sub-rotina para desligar o motor do gravador.

Condições de Entrada: Nenhuma.

Condições de Saída: Nenhuma.

Programa Z 80 de exemplo: Veja $CSHWR.

$CSOUT - 612D/0264H

Grava um *byte* de dado na fita cassete.

Condições de Entrada: Em A deve estar o *byte* de dado.

Condições de Saída: Nenhuma.

Programa Z 80 de exemplo: Veja $CSHWR.

$DATE - 12339D/3033H

Fornece a data atual.

Condições de Entrada: O par HL endereça o *buffer* de saída de oito *bytes.*

Condições de Saída: O par HL endereça a data, que estará no formato "MM/DD/AA". Todos os outros registradores são alterados.

Nota: Deve ser usada após a sub-rotina $CSHWR.

Programa Z 80 de exemplo:

```
82EA  210883  00101          LD    HL,TXT2          BUFFER DE 8 BYTES
82ED  CD3330  00102          CALL  DATE             BUFFER DE 8 BYTES
82F0  21FF82  00103          LD    HL,TXT3          (HL)=HORA/DATA MAIS SIGNIF.
82F3  CD3630  00104          CALL  TIME
82F6  21FF82  00105          LD    HL,TXT3
82F9  CD1B02  00106          CALL  VDLINE           COLOCA HORA OU DATA NA TELA
82FC  C3191A  00107          JP    READY
82FF          00108 TXT3     DEFS  8                A HORA FICA AQUI
8307  20      00109          DEFB  20H              CODIGO ASCII DE ESPACO
8308          00110 TXT2     DEFS  8                A DATA FICA AQUI
8310  0D      00111          DEFB  0DH              FIM DA LINHA (RETORNO DE CARRO)
```

$DELAY - 96D/0060H

Suspende o processamento durante um tempo pré-definido. Esta é uma rotina de uso geral, que pode ser utilizada sempre que você quiser dar uma pausa antes de continuar com o programa.

Condições de Entrada: O fator de atraso deve ser armazenado em BC. O atraso real será dado por:

$$\text{atraso} = 2.46 + (14.8*BC) \text{ microssegundos}$$

Quando o conteúdo de BC for nulo (BC=0000H), será usado o valor 65536, que corresponde ao tempo máximo (perto de um segundo).

Condições de Saída: São alterados apenas os registradores BC e A.

Programa Z 80 de exemplo:

```
              00112 ;COLOCA, COMPASSADAMENTE, TODOS OS CARACTERES NA TELA.
3E20          00113 CENTER EQU  3E20H              CENTRO DA TELA (L=8,C=32)
8311  CD6900  00114          CALL INITIO            INICIALIZA E/S
8314  CDC901  00115          CALL VDCLS             LIMPA A TELA
8317  3E00    00116          LD   A,0H
8319  01FF7F  00117          LD   BC,7FFFH          AJUSTA ATRASO EM 1/2 SEG.
831C  32203E  00118 LOOP3    LD   (CENTER),A        COLOCA O CARACTERE NA TELA
831F  F5      00119          PUSH AF                SALVA O ULTIMO CODIGO
8320  C5      00120          PUSH BC                E O FATOR DE ATRASO
8321  CD6000  00121          CALL DELAY
8324  C1      00122          POP  BC
8325  F1      00123          POP  AF
8326  3C      00124          INC  A                 PROXIMO CODIGO ASCII
8327  20F3    00125          JR   NZ,LOOP3          SE NAO FOR ZERO, POE NA TELA
8329  C3191A  00126          JP   READY             SE FOR, TERMINA O PROGRAMA
```

$INITIO - 105D/0069H

Inicializa todos os *drives* de E/S. Chame essa sub-rotina para recolocar todas as entradas e saídas do CP 500 em suas condições iniciais, incluindo-se aí, os desvios realizados.

Condições de Entrada: Nenhuma.

Condições de Saída: Todos os registradores são alterados.

Programa Z 80 de exemplo: Veja $DELAY.

Programa Basic de exemplo:

```
10 POKE 16526,105:POKE 16527,0      'BYTES - E + SIGNIF.
20 X=USR(0)                         'ARGUMENTO AUXILIAR
```

$KBCHAR - 43D/002BH

Essa sub-rotina fornece um caractere do teclado, quando algum estiver sendo digitado. Ela verifica o teclado para um caractere apenas, e esse não aparecerá na tela.

Condições de Entrada: Nenhuma.

Condições de Saída: O registrador armazena o código ASCII. Se seu valor for zero, não haverá caractere disponível. O par DE também é alterado.

Programa Z 80 de exemplo: Veja $RSINIT.

$KBLINE - 64D/0040H

Espera por uma linha de dados vinda do teclado. Uma linha é terminada por um retorno de carro (código ASCII 0DH) ou por um BREAK (código ASCII 01H). Os caracteres digitados serão colocados na tela.

Condições de Entrada: O conteúdo de B define o comprimento máximo da linha. Quando este comprimento for atingido, nada mais será permitido exceto BREAK ou ENTER. O par HL endereça o *buffer* de armazenamento. O comprimento desse *buffer* deve ser B+1, para armazenar a linha mais o caractere de finalização.

Condições de Saída: A condição de *carry* (registrador F) indica que BREAK finalizou a linha. O número de caracteres digitados estará armazenado em B. O par HL endereça a linha introduzida pelo teclado seguida do caractere de finalização. O par DE também é alterado.

Programa Z 80 de exemplo: Veja $CSHWR.

$KBWAIT - 73D/0049H

Espera por um caractere do teclado. Esta sub-rotina varre o teclado até que uma tecla seja pressionada, gerando um código de caractere que será armazenado no registrador A. O caractere não será colocado na tela.

Condições de Entrada: Nenhuma.

Condições de Saída: Em A estará o caractere digitado. O para DE é alterado.

Programa Z 80 de exemplo: Veja $CSHWR.

$KBBRK - 653D/028DH

Verifica apenas a tecla BREAK. Esta sub-rotina tem como função uma rápida verificação da tecla BREAK. Use-a quando quiser minimizar o tempo de varredura do teclado sem, no entanto, bloqueá-lo totalmente.

Condições de Entrada: Nenhuma.

Condições de Saída: A condição NZ (registrador F) indica que BREAK foi pressionada (NZ=diferente de zero). O registrador A é alterado.

Programa Z 80 de exemplo: Veja $RSINIT.

$PRCHAR - 59D/003BH

Envia um caractere para a impressora. $PRCHAR espera até que a impressora esteja pronta para receber o caractere, ou então, que a tecla BREAK seja pressionada. Se BREAK for pressionada, a sub-rotina interromperá a espera, retornando ao programa que a chamou.

Condições de Entrada: O código do caractere deve ser armazenado em A.

Condições de Saída: O par DE é alterado.

Programa Z 80 de exemplo:

```
                00148 ; DEMONSTRACAO DA IMPRESSORA
8356  216583    00149         LD    HL,TXT4          (HL)= TEXTO DE EXEMPLO
8359  7E        00150 LOOP5   LD    A,(HL)           POE O CARACTERE EM A
835A  23        00151         INC   HL               INDICA O PROXIMO CARACTERE
835B  CD3B00    00152         CALL  PRCHAR           O CARACTERE EM A E' IMPRESSO
835E  FE0D      00153         CP    0DH              E' UM RETORNO DE CARRO ?
8360  20F7      00154         JR    NZ,LOOP5         SE NAO FOR, PEGA O PROXIMO
8362  C3191A    00155         JP    READY            SE FOR, TERMINA O PROGRAMA
8365  49        00156 TXT4    DEFM 'ISTO E' UM TESTE DE IMPRESSAO'
```

$PRSCN - 473D/01D9H

Imprime todo o conteúdo da tela na impressora. Esta sub-rotina envia para a impressora todos os 1024 caracteres da tela, obedecendo o comprimento da linha na tela. Se for pressionada BREAK durante sua execução, o processamento retorna ao programa principal (aquele onde ocorreu a chamada).

Condições de Entrada: Nenhuma.

Condições de Saída: Todos os registros são alterados.

Programa Z 80 de exemplo: Pode ser chamada do mesmo modo que SVDCLS (veja o programa de exemplo de $CSHIN).

$READY - 6681D/1A19H

Recoloca o computador na condição "READY". Para retornar ao interpretador Basic, saindo de uma rotina Z 80, deve ser executado um desvio *(jump)* para esta sub-rotina. Não se deve fazer uma chamada, mas um salto para o ponto de entrada de $READY.

Condições de Entrada: Nenhuma.

Condições de Saída: Nenhuma.

Programa Z 80 de exemplo: Todos os programas de exemplo aqui apresentados usam a instrução JP READY.

$RESET - 0D/0000H

O mesmo que pressionar RESET . Desvia a programação para esse endereço, inicializando o sistema, o que faz voltar ao "diálogo inicial". Se o CP 500 possuir *drives*, o computador tentará carregar o DOS 500. Para evitar que isso ocorra, o operador deve pressionar BREAK antes que esse desvio seja executado.

Condições de Entrada: Nenhuma.

Condições de Saída: Nenhuma.

Programa Z 80 de exemplo: Opera como $READY.

$ROUTE - 108D/006CH

Desvia o endereçamento dos periféricos.

Condições de Entrada: A abreviatura ASCII (dois *bytes*) do periférico original deve ser armazenada nos *bytes* 4222H e 4223H, enquanto que a do periférico desejado, nos endereços 4220H e 4221H.

As abreviaturas válidas são:

KI teclado
DO vídeo
RI entrada serial
RO saída serial
PR impressora

Condições de Saída: Apenas o par DE é alterado.

Programa de exemplo: Veja no capítulo "Desviando Entradas e Saídas".

$RSINIT - 90D/005AH

Inicializa a interface RS 232 C. Quando você liga o computador, a interface é inicializada com as características:

Velocidade de transferência	300 bauds
Comprimento da palavra	8 *bits*
Paridade	nenhuma
Bits de parada	1 *bit*

Espera pela complementação da operação de transmissão/recepção.

Para mudar qualquer dessas características você deve alterar as condições de entrada e chamar $RSINIT.

Condições de Entrada: Os três *bytes* a seguir armazenam os códigos de inicialização da interface serial.

(16888) = código de velocidade de transmissão/recepção:

4 *bits* mais signficativos = velocidade de transmissão
4 *bits* menos significativos = velocidade de recepção

Veja no capítulo "Usando a Interface Serial RS 232 C" os códigos de transmissão/recepção.

(16889) = chave de características da RS 232 C:

Bits	*Significado*
7	Paridade: 1 = par 0 = ímpar
6,5	comprimento da palavra: 00 = 5 *bits* 01 = 6 *bits* 10 = 7 *bits* 11 = 8 *bits*
5	*bits* de parada: 0 = 1 *bit* 1 = 2 *bits*
4	controle de paridade: 0 = com paridade 1 = sem paridade
3	transmissão: 0 = desativada 1 = ativada
2	terminal de dados (DTR): 0 = não preparado 1 = preparado
1	pede para enviar (RTS): 0 = não 1 = sim

(16890) = chave de espera de transmissão/recepção:

0 = não espera
≠ 0 espera

Condições de Saída: Apenas o par DE é alterado.

Programa Z 80 de exemplo:

```
                00127 ;PROGRAMA DE DEMONSTRACAO DAS SUB-ROTINAS:
                00128 ;$RSINIT, $RSTX, $RSRCV, $KBCHAR E $VDCHAR
                00129 ;ASSUME OS CONTEUDOS DE 16888 E 16889 COMO VALORES DE
                00130 ;INICIALIZACAO DA INTERFACE SERIAL
832C  AF        00131         XOR   A                 ZERA REGISTRADOR A PARA
832D  32FA41    00132         LD    (16890),A         SELECIONAR "NAO ESPERA"
8330  CD5A00    00133         CALL  RSINIT
8333  CDC901    00134         CALL  VDCLS
8336  CD2B00    00135 KEYIN   CALL  KBCHAR            VERIFICA TECLADO
8339  FE00      00136         CP    0
833B  2806      00137         JR    Z,RSIN            CHECA RS SE NAO HA' TECLA
833D  CD3300    00138         CALL  VDCHAR            ECO DO TECLADO
8340  CD5500    00139         CALL  RSTX              ENVIA INTRODUCAO PELA RS
8343  21E814    00140 RSIN    LD    HL,16872          (HL)= BUFFER DE ENTRADA
8346  CD5000    00141         CALL  RSRCV             CHECA ENTRADA SERIAL
8349  7E        00142         LD    A,(HL)            PEGA O CONTEUDO DO BUFFER
834A  FE00      00143         CP    0
834C  28E8      00144         JR    Z,KEYIN           SE NAO HA NADA, CHECA TECLADO
834E  CD3300    00145         CALL  VDCHAR            CASO CONTRARIO, POE NA TELA
8351  18E3      00146         JR    KEYIN             CHECA TECLADO
8353  C3191A    00147         JP    READY             RETORNE AO INTERPRETADOR BASIC
```

$RSRCV - 80D/0050H

Recebe um caractere através da interface RS 232 C. Se a interface estiver no modo de espera, esta rotina retém o processamento até que seja recebida uma informação através da interface, ou até que BREAK seja pressionada.

Caso o modo seja de não espera, o processamento retornará ao ponto de chamada mesmo que não seja recebido nenhum caractere.

Condições de Entrada: Nenhuma.

Condições de Saída: O caractere recebido é armazenado no endereço 16872. Um valor nulo indica que não foi recebido caractere algum. O par DE é alterado.

Programa Z 80 de exemplo: Veja $RSINIT.

$RSTX - 85D/0055H

Transmite um caractere através da interface RS 232 C. Se o modo for de espera, essa sub-rotina aguardará até que um caractere seja transmitido, ou então, até que se pressione BREAK.

Se o modo for de não espera, a sub-rotina retornará tendo ou não transmitido um caractere.

Condições de Entrada: O caractere deve estar em A ou no endereço 16880.

Condições de Saída: Se a condição NZ (registrador F) for satisfeita, ou o conteúdo de 16880 for zero, então não foi enviado nenhum caractere. O par DE é alterado.

Programa Z 80 de exemplo: Veja $RSINIT.

$SETCAS - 12354D/3042H

Permite ao operador especificar a velocidade de transferência para o gravador cassete. A primeira questão do diálogo inicial é repetida, aparecendo na tela a frase "Cass?".

Essa frase é colocada na linha imediatamente abaixo da posição do cursor. O computador espera, então, que seja digitado "A" (1500 bauds), "B" (500 bauds) ou, simplesmente, ENTER (1500 bauds, por omissão).

Ao retornar da sub-rotina, a velocidade estará de acordo com o especificado.

Condições de Entrada: Nenhuma.

Condições de Saída: Todos os registradores são alterados.

Programa Z 80 de exemplo: Veja $CSHWR.

$TIME - 12342D/3036H

Essa sub-rotina fornece a hora certa.

Condições de Entrada: O par HL indica o *buffer* de saída com oito *bytes.*

Condições de Saída: O par HL endereça a hora certa, que estará armazenada no formato "HR:MN:SS". Todos os outros registradores são alterados.

Programa Z 80 de exemplo: Veja $DATE.

$VDCHAR - 51D/0033H

Esta sub-rotina coloca um caractere na posição atual do cursor.

Condições de Entrada: O caractere ASCII deve ser armazenado em A.

Condições de Saída: O par DE é alterado.

Programa Z 80 de exemplo: Veja $RSINIT.

$VDCLS - 457D/01C9H

Limpa a tela.

Condições de Entrada: Nenhuma.

Condições de Saída: Todos os registradores são alterados.

Programa Z 80 de exemplo: Veja $CSHWR.

$VDLINE - 539D/021BH

Esta sub-rotina coloca uma linha na tela. O último caractere da linha pode ser um ETX (03H) ou um retorno de carro (0DH). Se for um retorno de carro, a linha será impressa na tela. Caso seja um ETX, a linha não será impressa. Isto permite que a $VDLINE posicione o cursor no começo da próxima linha de texto, ou imediatamente após o último caractere do texto impresso.

Condições de Entrada: O par HL endereça o início do texto a ser impresso, que deve terminar em 03H ou 0DH.

Condições de Saída: O par HL estará indicando o primeiro caractere após o término do texto. O par DE é alterado.

Programa Z 80 de exemplo: Veja $CSHWR.

PROCESSAMENTO DE BREAK

A tecla BREAK é reconhecida durante a operação de varredura do teclado. O computador transfere o controle para um vetor de desvio de três *bytes* na memória RAM (valores hexa: C3 LSB MSB). Para aplicações especiais, mude o endereço do vetor de desvio de forma a permitir que seus programas manipulem a tecla BREAK.

O vetor de desvio da varredura BREAK está localizado em 16396 (400CH).

Condições de Entrada: O par DE é alterado pelo computador. O par SP (*stack pointer*) deve conter o endereço de retorno para o programa interrompido. Isso quer dizer que uma instrução Z 80 RET fará com que o controle seja transferido para o ponto onde o processamento foi interrompido.

Programa Basic de exemplo: Execute este programa Basic para desativar BREAK:

```
10 POKE 16396,175          '175 = CODIGO Z 80 P/ "XOR A"
20 POKE 16397,201          '201 = CODIGO Z 80 P/ "RET"
```

E este para reativar a tecla BREAK:

```
10 POKE 16396,201          'CODIGO Z 80 PARA "RET"
```

Mapa de Memória

A seguir mostramos um resumo da distribuição da memória de seu CP 500. A primeira tabela mostra a organização da memória como um todo. As demais mostram os principais endereços, tanto da memória RAM quanto da EPROM.

Organização da Memória		
Endereço decimal	*Conteúdo*	*Endereço hexadecimal*
0	Interpretador Basic 12 k de EPROM 12 k de EPROM	0
12288	2 k de EPROM para uso do sistema	3000
14336	Matriz do teclado	3800
15360	Memória de vídeo Canto superior esquerdo: 15360+0 Canto superior direito: 15360+1023	3C00
16384	Reservado para uso do sistema	4000
17385	Memória do usuário para programas e dados	43E9
32767	Final de 16 k de RAM	7FFF
49151	Final de 32 k de RAM	BFFF
65535	Final de 48 k de RAM	FFFF

Os endereços da EPROM devem ser utilizados em chamadas USR, sendo armazenados em dois *bytes*, nos endereços 16526 e 16527.

Sumário dos Endereços Importantes da EPROM			
Endereço		*Conteúdo*	*Função*
Dec.	*Hex.*		
0	0000	$RESET	inicialização do sistema.
43	002B	$KBCHAR	verifica caractere no teclado.
51	0033	$VDCHAR	coloca um caractere na tela.
59	003B	$PRCHAR	imprime um caractere na impressora.
64	0040	$KBLINE	espera por uma linha do teclado.
73	0049	$KBWAIT	espera por um caractere do teclado.
80	0050	$RSRCV	recebe um caractere pela RS 232 C.
85	0055	$RSTX	transmite um caractere pela RS 232 C.
90	005A	$RSINIT	inicializa a RS 232 C.
96	0060	$DELAY	suspende o processamento por um tempo determinado.
105	0069	$INITIO	inicialização de E/S.
108	006C	$ROUTE	desvia endereçamento de E/S.
457	01C9	$VDCLS	limpa a tela.
473	01D9	$PRSCN	imprime o conteúdo da tela.
504	01F8	$CSOFF	desliga o gravador.
539	021B	$VDLINE	coloca uma linha na tela.
565	0235	$CSIN	carrega um *byte* da fita na memória.
612	0264	$CSOUT	grava um *byte* na fita.
647	0287	$CSHWR	grava um prefixo na fita.
653	028D	$KBBRK	verifica apenas a tecla BREAK.
662	0296	$CSHIN	lê o prefixo na fita.
664	0298	$CLKON	coloca a hora na tela.
673	02A1	$CLKOFF	retira a hora da tela.
6681	1A19	$READY	desvia para a condição *READY*.
12339	3033	$DATE	fornece a data.
12342	3036	$TIME	fornece a hora.
12354	3042	$SETCAS	ajusta a velocidade de transferência da fita.

14312	37E8	$PRSTAT	condições da impressora (apenas lê), prosseguir apenas se:
			bit 7 = 0 (desocupada);
			bit 6 = 0 (papel no lugar);
			bit 5 = 1 (disp. selecionado);
			bit 4 = 1 (impressão s/ erro);
			os *bits* 3, 2, 1 e 0 não são usados.

Sumário dos Endereços Importantes da RAM			
Endereço		*Conteúdo*	*Conteúdo Inicial*
Dec.	*Hex.*		
16396	400C	vetor de desvio *BREAK* operações de varredura do teclado (3 *bytes*)	C9 xx xx
16409	4019	chave de maiúsculas: 0 = "minúsculas e maiúsculas" = 0 "maiúsculas"	maiúsculas
16412	401C	chave do cursor: 0 = "intermitente" = 0 "constante"	intermitente
16416	4020	endereço do cursor: 2 *bytes:* LSB, MSB	não é usado
16419	4023	caractere do cursor: código ASCII de 32 a 255	176
16424	4028	número máximo de linhas/página mais um	67
16425	4029	número de linhas impressas mais um	1
16427	402B	comprimento máximo da linha impressa, menos dois: 255 é o comprimento máximo	255
16872	41E8	$RSRCV, *buffer de entrada* (1 *byte*)	1
16880	41F0	*buffer de saída* do $RSTX (1 *byte*)	0
16888	41F8	código de transmissão/recepção para $RSINIT parte mais sign = código TX parte menos sign = código RCV	85
16889	41F9	código de paridade/comprimento da palavra/*bits* de parada para $RSINIT	108
16890	41FA	chave de espera do $RSINIT: 0 = "não espera" = 0 "espera"	espera
16913	4211	chave de velocidade de transferência 0 = 500 baud = 0, 1500 baud	não usa
16916	4214	proteção contra deslocamento (0 a 7). Valores maiores que sete são interpretados em módulo oito.	0

Sumário dos Endereços Importantes da RAM			
Endereço		*Conteúdo*	*Conteúdo Inicial*
Dec.	*Hex.*		
16919	4217	data e hora: 6 *bytes* binários SS MN HR AA DD MM	00:00:00 00/00/00
16928	4220	determinador de E/S de 2 *bytes* ($ROUTE). Dispositivo original.	não usa
16930	4222	determinador de E/S de 2 *bytes* ($ROUTE). dispositivo destino.	não usa

Capítulo 13

Detecção de Falhas e Manutenção

Tabela de Sintoma/Providência

Este capítulo apresenta uma relação de sintomas e possíveis causas que o impossibilitarão de operar normalmente seu CP 500. Mas não se esqueça; uma instrução que não tenha sido seguida corretamente também representa um problema na operação do equipamento.

Ao perceber qualquer anormalidade, tente, em primeiro lugar, as providências enumeradas a seguir. Se o seu problema ainda persistir, leve a unidade até o representante local da Prológica. A solução será imediata.

Sintoma	*Causa provável/providências*
O computador é ligado e a mensagem Cass? não aparece na tela.	1 - Não há energia. Verifique as conexões do cabo. 2 - Seqüência incorreta ao ligar o computador. 3 - Algum periférico não está devidamente conectado. Reveja as ligações. 4 - Em sistemas com drives para disquete, mantenha pressionada BREAK enquanto liga ou pressiona RESET. 5 - A tela pode estar precisando de um ajuste de brilho. Verifique-a.
O computador não carrega um programa da fita.	1 - Conexão incorreta do gravador. Verifique as instruções de conexão no capítulo "Usando a Interface para Gravador Cassete". 2 - A velocidade de carregamento do computador não é a mesma que foi gravado o programa. 3 - Ajuste incorreto de volume. Tente outro nível. 4 - As informações na fita foram danificadas por campo magnético ou mesmo pela própria deterioração da fita. Se você tiver uma outra cópia, tente usá-la.

Sintoma	*Causa provável/providências*
Durante a operação normal, o computador trava, sendo necessário desligá-lo e ligá-lo novamente, ou usar RESET	1 - Isso pode ocorrer quando há flutuação de energia. Veja adiante em "Alimentação CA". 2 - Conector defeituoso ou mal instalado. Verifique se todos os cabos de conexão estão bem colocados e se não estão danificados. 3 - A programação pode estar irregular. Verifique-a. Provavelmente o computador entrou em *looping*.

Alimentação CA

Os computadores são sensíveis à flutuação da rede elétrica. Isto raramente constitui um problema, a menos que você esteja operando próximo a equipamentos elétricos pesados. A alimentação pode também ser instável se, nas proximidades, algum eletrodoméstico ou equipamento de escritório estiver com o interruptor defeituoso. Isso provocará um faiscamento ao se acionar o equipamento.

Para suportar as condições que mencionamos é que o computador foi projetado com um filtro de linha CA interno. Ele deverá eliminar os efeitos de flutuações normais. Entretanto, se as flutuações forem severas será necessário tomar algumas das seguintes providências abaixo:

a) Instale circuitos de isolação ou *"bypass"* nos aparelhos que causam interferência;
b) Substitua os interruptores defeituosos;
c) Instale uma linha individual para o computador;
d) Instale um filtro especial de linha projetado para computadores e outros equipamentos sensíveis.

Os problemas com a rede elétrica são raros e, muitas vezes, podem ser evitados pela escolha correta do local de instalação de seu computador. Quanto mais complexo for o sistema, mais séria será a aplicação e maior a importância que deverá ser dada à rede de alimentação do computador.

Manutenção

Em termos de manutenção, o computador é bem econômico. É bom mantê-lo sempre limpo e livre de poeira, especialmente o teclado.

Quando você precisar limpar o gabinete do computador, utilize uma flanela limpa e seca. Os periféricos (gravador, impressora, etc.) requerem maiores cuidados. Cada periférico tem um manual específico; verifique o sistema de manutenção de cada um.

Capítulo 14

Especificações Técnicas

Alimentação

Estas especificações se aplicam a sistemas sem unidades de disquetes. Para sistemas com disquetes, veja o livro "Sistema Operacional de Disco DOS 500".

Tensão de alimentação: 105 a 130 V_{CA}, 60 Hz

Consumo de corrente: 0,83 A_{RMS}

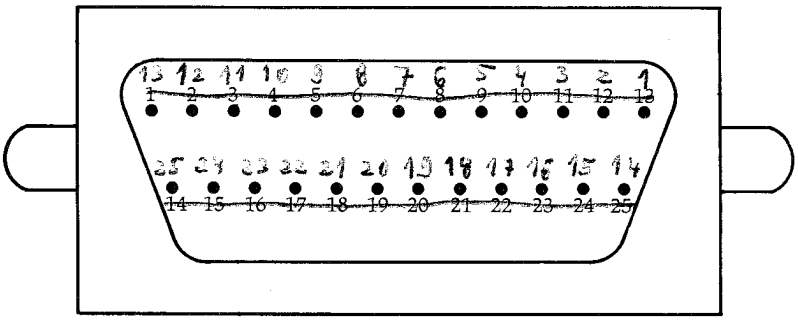
Microprocessador

Tipo: Z 80

Freqüência: 2,02752 MHz

Interface Serial RS 232 C

A localização do conector dessa interface pode ser vista na figura do capítulo 2. A função de cada pino, bem como a posição deste no conector, é mostrada na tabela abaixo. Na figura, o conector é visto pelo lado de fora do computador.



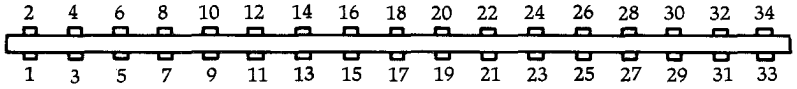
Sinais	*Padrão RS 232 C*	*Pino*
PG	*Protective Ground* — Terra de proteção	1
TD	*Transmit Data* — Transmite dados	2
RD	*Receive Data* — Recebe dados	3
RTS	*Request To Send* — Pede para enviar	4
CTS	*Clear To Send* — Limpa para enviar	5
DSR	*Data Set Ready* — Conjunto de dados pronto	6
SG	*Signal Ground* — Terra de sinal	7
CD	*Carrier Detect* — Detecta *carrier*	8
DTR	*Data Terminal Ready* — Terminal de dados pronto	20

Sinais	*Padrão RS 232C*	*Pino*
RI	*Ring Indicator* — Indicador de chamada	22
STD*	*Secondary Transmit Data* — "Transmite dados" secundário	14
SUN*	*Secondary UNassigned* — Secundário impedido	18
SRTS	*Secondary Request To Send* — "Pede para enviar" secundário	19

*Nota: Estes sinais não são utilizados em funções secundárias, mas reservados para uso futuro.

Interface Paralela para Impressora

A tabela a seguir identifica e localiza os sinais do conector da interface paralela. A figura mostra o conector pelo lado de fora do computador.



Sinais	*Função*	*Pino*
STROBE*	Pulso de 1,5 μs para controle do dado do processador para a impressora.	1
DADO0	*Bit* 0 (menos sign.) do *byte* do dado de saída.	3
DADO1	*Bit* 1 do *byte* de saída de dados	5
DADO2	*Bit* 2 do *byte* de saída de dados	7
DADO3	*Bit* 3 do *byte* de saída de dados	9
DADO4	*Bit* 4 do *byte* de saída de dados	11
DADO5	*Bit* 5 do *byte* de saída de dados	13
DADO6	*Bit* 6 do *byte* de saída de dados	15
DADO7	*Bit* 7 do *byte* de saída de dados	17
BUSY	Sinal da impressora para o computador. Se estiver em nível alto, significa que ela está ocupada.	21
PAPER EMPTY	Sinal da impressora para o computador. Indica a ausência do papel quando em nível alto. Se a impressora não dispuser de sensor de papel, este sinal estará sempre em nível baixo.	23
SELECT	Sinal da impressora para o computador, nível alto indica que o dispositivo foi selecionado.	25
FAULT*	Indica uma falha para o computador (falta papel, não selecionado, etc.)	28
GROUND	Terra comum	2,4,6,8, 10, 12, 14, 16, 18, 20, 22, 24, 27, 34,
NC	Sem conexão	26,31,33
AUX	Auxiliares (não usados)	19, 29, 30, 32

*Nota: Estes sinais são ativos em nível baixo *(active-low).*

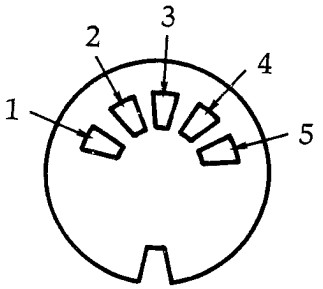
Interface para Gravador Cassete

Nível de entrada recomendado: 1 a 5 V (pico-a-pico), a uma impedância mínima de 220 Ohms.

Nível de saída típico do computador: 800 mV (pico-a-pico), a 1 kOhm.

Capacidade de interrupção do REMOTE: 0,5 A máximo a 6 V_{CC}.

O esquema abaixo mostra o conector visto pelo lado de fora.



1. Controle REMOTE;
2. Entrada do *jack* EAR do gravador;
3. Terra de sinal;
4. Saída para o gravador (*jack* AUX ou MIC);
5. Controle REMOTE.

Basic

Capítulo 15

Elementos de Programa

Nos próximos quatro capítulos, apresentamos os conceitos básicos que permitem utilizar a capacidade total da linguagem Basic do seu computador. Desta forma, nossa finalidade é permitir aos programadores a elaboração de programas eficientes e confiáveis.

Caso você não sinta necessidade poderá passar para os capítulos seguintes, mas tendo sempre em mente que as informações a seguir estão à sua disposição.

Um programa consiste em um conjunto de instruções, as quais serão explicadas detalhadamente nos capítulos seguintes.

Um programa é constituído de uma ou mais linhas. Cada linha consiste em uma ou mais instruções. O Basic admite numeração de linhas entre 0 e 65529, inclusive.

Em cada linha você pode introduzir até 240 caracteres, incluindo o número da linha. Caso você utilize em uma linha o modo de edição, você poderá introduzir 15 caracteres extras.

Você poderá armazenar em uma linha duas ou mais instruções, sendo que estas devem sempre ser separadas por dois pontos (:).

Programa exemplo:

Número de linha	*Instrução Basic*	*Caractere (:) entre as instruções*	*Instrução Basic*
110	CLS	:	PRINT"ALFABETO"
120	PRINT		
130	FOR I=1 TO 1000:		NEXT I
140	PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"		
150	END		

Quando um programa é executado, as instruções são manipuladas individualmente, iniciando pela primeira e prosseguindo até a última.

Algumas instruções, que serão abordadas mais a frente, tais como: GOTO, ON...GOTO, GOSUB e ON...GOSUB, modificam essa seqüência.

Instruções

Uma instrução determina quais operações específicas devem ser executadas pelo computador, através da linguagem utilizada.

Exemplo de uma instrução:

```
END
```

Esta instrução faz com que o computador realize a operação de término da execução do programa.

Expressões

Uma expressão é um conjunto de dados (informações) fornecidos ao computador. Existem quatro tipos de expressões:

EXPRESSÕES NUMÉRICAS

São aquelas compostas por dados numéricos.
Exemplos:

```
(1+5.2)/3
SIN(3+E)
D
5*8
3.7682
ABS(X)+RND(0)
```

EXPRESSÕES STRING

São constituídas por dados na forma alfanumérica.
Exemplos:

```
A$
"A"
"A"+"B"
M$+"DADO"
MID$(A$,2,5)+MID$("HOMEM",1,2)
M$+A$+B$
```

EXPRESSÕES RELACIONAIS

Estas expressões testam a relação lógica entre duas ou mais expressões.
Exemplos:

```
A$="SIM" AND B$="NAO"
C>5 OR M<B OR 0>2
578 AND 452
```

Funções

Funções são sub-rotinas automáticas e grande parte delas executam cálculos com dados.

Deve-se utilizar as funções do mesmo modo que usamos qualquer outra informação, isto é, como parte de uma instrução.

Exemplos de funções:

```
INT
ABS
STRING$
```

Capítulo 16

Tratamento dos dados

O Basic residente oferece vários métodos diferentes de tratar seus dados. Utilizando-os adequadamente, pode-se melhorar bastante a eficiência de um programa.

Neste capítulo, discutiremos como os dados são representados e armazenados; como são classificados os dados e as variáveis; e ainda como ocorre a conversão de dados numéricos.

Como o Basic Representa os Dados

Os dados podem ser representados como constantes ou como variáveis.

CONSTANTES

São todas as informações introduzidas num programa que não estão sujeitas a alterações. Por exemplo, na instrução:

```
PRINT"1 MAIS 1 E' IGUAL A";2
1 MAIS 1 E' IGUAL A - é uma constante string.
2                   - é uma constante numérica.
```

VARIÁVEIS

Uma variável é um lugar na memória onde os dados são armazenados. Estes dados estão constantemente sujeitos a alterações, permitindo, assim, que o programador trabalhe com quantias variáveis. Por exemplo, na instrução:

```
A$="ATIVIDADE"
```

a variável A$ contém a informação "ATIVIDADE". No entanto, se posteriormente for executada uma instrução como:

```
A$="NEGOCIOS"
```

a variável A$ não mais conterá a informação "ATIVIDADE" mas, sim, "NEGOCIOS".

Nomes de variáveis: Em Basic, as variáveis são representadas por nomes, sendo que estes devem começar com uma letra de A a Z. Esta letra pode ser seguida por um número ou por outra letra. Por exemplo:

```
AM    A1    AB
A     B1    XR
```

Os exemplos acima são todos válidos e de variáveis distintas. Outra característica é a possibilidade dos nomes de variáveis possuirem mais que dois caracteres. Todavia, somente os dois primeiros serão significativos em Basic. Por exemplo:

```
SUPER
SUJEITO
SUB
```

são manipulados como sendo a mesma variável.

Palavras Reservadas: Certas combinações de letras são interpretadas como palavras chave em Basic, e não podem ser usadas como nomes de variáveis. Por exemplo:

```
FORA
MANDAR
LENHA
CIFRA
```

Estas palavras não podem ser utilizadas como nomes de variáveis, pois contêm as seguintes palavras reservadas, na ordem: FOR, AND, LEN e IF.

Variáveis Simples e Subscritas: Todas as variáveis mencionadas acima são variáveis simples, portanto devem referir-se somente a um dado. As variáveis podem também ser subscritas, permitindo assim que uma lista inteira de dados seja armazenada sob um mesmo nome. Este método de armazenamento de dados é chamado **matriz.**

Por exemplo, uma matriz chamada A pode conter os seguintes elementos:

```
A(0)      A(1)      A(2)      A(3) ...
```

Cada um destes elementos é chamado variável subscrita. Pode-se, com isso, armazenar cada dado individualmente, da seguinte forma:

```
A(0)=5.3        A(3)=3.2
A(1)=7.2        A(4)=9.4
A(2)=8.3        A(5)=7.8
```

Neste exemplo a matriz é unidimensional, pois cada elemento contém apenas um subscrito que o identifica. Com a utilização do Basic, você pode dimensionar a sua matriz com até 255 subscritos (dimensões). Por exemplo, uma matriz bidimensional (cada elemento é identificado por dois subscritos) chamada "X", poderia conter os seguintes elementos:

```
X(0,0)=8.6        X(0,1)=3.5
X(0,1)=7.3        X(1,1)=32.6
```

Uma matriz tridimensional chamada "L" poderia conter estes elementos:

```
L(0,0,0)=32233      L(0,1,0)=96522
L(0,0,1)=52000      L(0,1,1)=10255
L(1,0,0)=33333      L(1,1,0)=96253
L(1,0,1)=53853      L(1,1,1)=79654
```

O interpretador Basic parte da premissa de que todas as matrizes contém onze elementos em cada dimensão. Caso você deseje acrescentar mais elementos, utilize, no início do seu programa, a instrução DIM, da qual existem explicações mais detalhadas no capítulo sobre matrizes.

Como o Basic Armazena os Dados

O espaço de memória que os dados ocuparão, assim como o tempo necessário para processá-los, depende do modo como forem armazenados.

DADOS NUMÉRICOS

Os dados numéricos podem ser classificados como números inteiros, de precisão simples e de precisão dupla.

Modalidade Inteiros: Para ser armazenado como inteiro, um número não pode ser fracionário e deve estar na faixa de −32768 a +32767. O inteiro requer somente dois *bytes* de memória para seu armazenamento e, porisso, as operações aritméticas são mais rápidas quando ambos os operandos são inteiros. Por exemplo:

```
1     32000     -2     500   -12345
```

Todos esses números podem ser armazenados como valores inteiros.

Modalidade Precisão Simples: Números de precisão simples possuem até sete algarismos significativos dos quais apenas seis são impressos, o sétimo serve como algarismo de arredondamento. Pode-se representar valores em notação científica com expoentes de até ±38.
Isto significa valores na faixa de:
$[-1x10^{38}, -1x10^{-38}][1x10^{38}, 1x10^{-38}]$

Um valor de precisão simples requer quatro *bytes* de memória para armazenagem. Quando não for especificado o nível de precisão de um número, o interpretador Basic classificará o mesmo como de precisão simples.

Notação científica é a representação de um número com apenas um algarismo significativo à esquerda do ponto decimal. Por exemplo, o número 12.3 é representado em notação científica como $1.23x10^{1}$. Em Basic, ele é representado como 1.23E1.

Nota: Quando um dos símbolos, E ou D, estiver acompanhando um número decimal, estará indicando a representação em notação científica deste valor. Isto equivale a dizer que o valor antes do símbolo (E para precisão simples e D para dupla) deve ser multiplicado por dez elevado ao valor após o símbolo. Portanto, 6.024E-23 representa o valor de precisão simples $6.024x10^{-23}$
E, 8.00100708D12 representa o valor de precisão dupla $8.00100708x10^{12}$. Exemplos:

```
1010234578            3.14159265358979 32
-8.7777651010         8.00100708D12
```

Estes números serão armazenados como de precisão dupla.

Exemplos de números de precisão simples:

```
54321.345        -200034
10.001           1.774E6
```

Esses números podem ser armazenados como números de precisão simples.

Modalidade Precisão Dupla: Os números de precisão dupla envolvem até dezessete algarismos significativos (dezesseis serão impressos) e podem representar valores na mesma faixa abrangida pelos números de precisão simples.

Um valor de precisão dupla requer oito *bytes* de memória para seu armazenamento. Porisso, as operações aritméticas com pelo menos um operando desta modalidade, são mais demoradas do que as que envolvem apenas números inteiros ou de precisão simples.

DADOS STRING

Se você quiser armazenar nomes, endereços, textos e informações em geral, deverá armazená-los como *strings.*

Uma *string* é uma seqüência de caracteres, que podem ser letras, números, ou quaisquer outros símbolos que o seu computador puder gerar. Essa seqüência não terá significado para o computador, sendo armazenada integralmente. Na memória, cada caractere de uma *string* é representado por um código ASCII.

Por exemplo:

A constante de dados *JOSE SILVA IDADE 38* pode ser armazenada numa *string* de 19 caracteres, utilizando-se o código ASCII, com um *byte* para cada caractere.

O computador armazenaria esta informação como mostra o quadro abaixo.

Código Hexa	4A	6F	53	45	20	53	49	4C	56	41	20	49	44	41	44	45	20	33	38
Caracteres ASCII	J	o	s	e		S	i	l	v	a		I	d	a	d	e		3	8

Uma *string* pode ter até 255 caracteres (*bytes*), e aquelas cujo comprimento é zero são chamadas *strings nulas* ou *vazias.*

Como o Basic Classifica as Constantes

Quando o Basic encontra uma constante de dados em uma instrução, esta pode ser uma *string*, um número de precisão simples, dupla ou um inteiro.

Primeiramente mostraremos as regras do Basic para classificação de constantes. Depois será mostrado como se pode anular essas regras nos casos onde é necessária uma classificação especial.

REGRA 1

Se o valor for escrito entre aspas será armazenado como uma *string*. Por exemplo:

```
"SIM"
"AV BRASIL 1234"
"1234567890"
```

REGRA 2

Se o valor não estiver entre aspas, ele será um número (uma exceção a essa regra se verifica durante uma instrução INPUT e nas linhas de dados - instrução DATA). Por exemplo:

```
123001
1
-7.3214E6
```

Estes valores são todos numéricos.

REGRA 3

Números não-fracionários, na faixa de −32768 a +32767, são inteiros. Por exemplo:

```
12350
-12
10012
```

São todos constantes inteiras.

REGRA 4

Se uma constante numérica não for um inteiro, e contiver até sete algarismos significativos, será considerada como de precisão simples. Por exemplo:

```
1234567
-1.23
1.3321
```

São todos números de precisão simples.

REGRA 5

Se um número contiver de 7 a 17 algarismos significativos, será de precisão dupla. Por exemplo:

```
12345678901234 56
-10000000000000.1
2.777000321
```

São todos de precisão dupla.

DECLARAÇÃO DE TIPOS

Pode-se mudar os critérios de classificação normal do Basic adicionando os seguintes símbolos no final de uma constante numérica:

% Transforma em um número inteiro. Por exemplo:

```
A=12.3%
```

A constante é armazenada como inteiro e abreviada para dois dígitos.

! Transforma em um número de precisão simples * . Por exemplo:

```
A=12.345678901234!
```

Esta constante é classificada como de precisão simples e abreviada para sete dígitos: 12.34567 (na tela ou na impressora será mostrado 12.3457).

Transforma em um número de precisão dupla.** Por exemplo, na instrução:

```
PRINT 3#/7
```

A primeira constante é classificada como de precisão dupla, antes que a divisão seja realizada.

Como o Basic Classifica as Variáveis

Quando o Basic encontra um nome de variável no programa, ele pode classificá-lo como *string*, inteiro, precisão simples ou dupla.

Em primeiro lugar, o Basic classifica todos os nomes de variáveis como de precisão simples. Por exemplo:

```
AB
CAPITAL
XY
L
```

Estes são todos de precisão simples. Se essa for a primeira linha de seu programa:

```
LP=1.2
```

o Basic classificará LP como uma variável de precisão simples. Entretanto, você pode determinar diferentes atributos às variáveis utilizando instruções de definição no início de seu programa. Tais instruções são:

DEFINT	Define variáveis como "inteiras";
DEFSNG ***	Define variáveis como "precisão simples";
DEFDBL	Define variáveis como "precisão dupla";
DEFSTR	Define variáveis como *"string"*.

Notas: * Quando o símbolo E estiver acompanhando um número decimal, estará representando um número de precisão simples em notação científica. O valor deve ser lido como: *o número vezes dez elevado a...*
Portanto, 6.024E −23 representa o valor de precisão simples 6.024×10^{-23}.

** Quando o símbolo D acompanhar um número decimal, estará representando um número de precisão dupla em notação científica. O valor deve ser lido como: *o número vezes dez elevado a...*
Portanto, 8.00100708D12 representa o valor $8.00100708 \times 10^{12}$.

Exemplo de definição:

```
DEFSTR L
```

Esta instrução faz com que o Basic passe a classificar todas as variáveis que comecem por L como variáveis *string*. Dessa forma, as variáveis L, LP e LAPIS, por exemplo, somente poderão armazenar valores *string*.

DECLARAÇÃO DE TIPOS

Assim como no caso das constantes, o tipo do nome de variável pode ser modificado acrescentando-se um símbolo de declaração no final. Existem quatro símbolos de declaração para variáveis, que são:

%	inteiros;
!	precisão simples;
#	precisão dupla;
$	*string*.

Por exemplo, I%, FT%, NUM% e PARCIAL% são todas variáveis inteiras, não importando o que tenha sido definido para as iniciais I, F, N e C.

Da mesma forma, os nomes T!, RY!, QUAN! e PERCET! identificam variáveis de precisão simples a despeito do que tenha sido definido para as iniciais T, R, Q e P.

O mesmo raciocínio se aplica aos símbolos # e $, para variáveis de precisão dupla e *strings*, respectivamente.

Nesse ponto, é importante notar que todo nome de variável pode se referir a quatro variáveis distintas, usando, para identificá-las, o declarador de tipo. Por exemplo:

```
A5%  A5!  A5#  A5$
```

Todos esses exemplos são nomes válidos de **variáveis distintas.**

Após ser executada uma instrução de definição, todos os nomes de variáveis por ela definidas não precisarão ser seguidas pelo símbolo de declaração de tipo. Depois da instrução DEFSTR C, por exemplo, a variável C1$ pode ser identificada por C1, simplesmente.

Como o Basic Converte os Dados Numéricos

Às vezes, em seu programa, pode ser necessário armazenar um determinado tipo de constante em um tipo diferente de variável. Veja essa instrução, por exemplo:

```
A%=2.34
```

Nesse caso, é necessário converter a constante de precisão simples 2.34 em um inteiro, antes de armazená-la na variável inteira A%. Pode-se, também, transferir o conteú-

---

Nota: *** Levando-se em consideração que o Basic classifica inicialmente todas as variáveis como de precisão simples, a instrução DEFSNG só precisa ser utilizada nos casos em que uma outra instrução de definição tenha sido executada anteriormente.

do de uma variável para outra, de tipo diferente. O computador se encarrega de converter o conteúdo de forma a compatibilizá-lo com a nova variável. Por exemplo:

```
A#=A%        A!=A#        A!=A%
```

CONVERSÃO DE PRECISÃO SIMPLES OU DUPLA EM INTEIRO

O Basic considera o maior inteiro não superior ao valor original*. Exemplos:

`A%= -10.5`	armazena em A% o valor -11;
`A%= 32767.9`	armazena em A% o valor 32767;
`A%= -2.5D3`	armazena em A% o valor 2500;
`A%= -123.456789012345678`	armazena em A% o valor -124;
`A%= -32768.1`	causa um erro de *overflow* (fora da faixa dos inteiros).

CONVERSÃO DE INTEIROS EM PRECISÃO SIMPLES OU DUPLA

O valor convertido assemelha-se ao valor original, apenas acrescido de zeros à direita do ponto decimal. Por exemplo:

`A#= 32767`	armazena 32767.000000000000 em A#;
`A!=-1234`	armazena–1234.000 em A!.

CONVERSÃO DE PRECISÃO DUPLA EM SIMPLES

Essa conversão envolve a transformação de um número de até 17 dígitos em um equivalente com, no máximo, sete. O Basic elimina os números menos significativos (os mais à direita), para conseguir um número de sete dígitos. Ao imprimir tal número, o Basic arredonda o valor para seis dígitos (arredondamento 4/5). Por exemplo:

`A!=1.2345679012345`	armazena 1.234567 em A!.
`PRINT A!`	mostra 1.23457 na tela.

CONVERSÃO DE PRECISÃO SIMPLES EM DUPLA

Para se efetuar essa conversão, o Basic simplesmente acrescenta uma seqüência de zeros ao número de precisão simples. Se o valor original tiver representação binária exata em formato de precisão simples, a conversão será precisa. Por exemplo:

`A#= 1.5` armazena 1.5000000000000000 em A#, desde que 1.5 realmente possue uma representação binária exata. Entretanto, os números que não preenchem esse requisito têm sua conversão para precisão dupla prejudicada, resultando em uma aproximação do número original. Por exemplo:

`A# = 1.3` armazena 1.299999952316284 em A#.

Nota: O valor original deve ser maior ou igual a–32768, e menor que 32768.

A maior parte dos números fracionários não possui representação binária exata, por isso, evite esse tipo de conversão.

Para contornar esse problema, você pode, por exemplo, forçar o valor que quer armazenar em uma variável de precisão dupla a ser classificado como sendo desse tipo usando um declarador. Exemplificando:

`A# = 1.3#` ou `A# = 1.3D` armazenam o valor exato 1.3 em A # (o símbolo # transforma o número em precisão dupla, e o D identifica esse tipo de número em notação científica).

Existe, ainda, uma técnica especial para conversão de precisão simples em dupla, sem incorrer em erros de precisão. Essa técnica é especialmente útil quando o valor a ser convertido está armazenado em uma variável. A técnica é a seguinte:

— Tome uma variável de precisão simples e converta-a em *string* com a instrução STR$;
— Então, converta a *string* resultante novamente em um número, usando VAL;

Esquematizando, teremos uma instrução do tipo:

VAL (STR$ (*variável de precisão simples*))

Execute o seguinte programa:

```
10 A!=1.3
20 A#=A!
30 PRINT A#
```

Este programa imprime o valor de 1.299999952316284.

Compare agora com este programa:

```
10 A!=1.3
20 A#=VAL(STR$(A!))
30 PRINT A#
```

Este imprime o valor 1.3. A conversão na linha 20 fez com que o valor exato em A! fosse armazenado em A #.

CONVERSÕES ILEGAIS

O Basic não converte valores numéricos diretamente em *string*, ou *strings* em números. Por exemplo, as instruções: A$=1234 e A% ="1234" não são realizadas. Para essas conversões, utilize STR$ e VAL.

Capítulo 17

Processamento dos Dados

Em Basic existem muitos métodos rápidos que podem ser utilizados para cálculo, classificação, teste e armazenamento dos dados. Esses métodos podem ser sub-divididos em duas categorias, como segue:

Operadores;

numéricos;
string;
relacionais;
lógicos.

Funções.

Operadores

Um operador é o símbolo ou palavra chave que indica qual a ação a ser tomada, em um ou dois valores especificados. Estes valores são chamados *operandos.* Em geral, um operando é usado da seguinte forma:

operando$_1$	*operador*	*operando*$_2$
6	+	2

Algumas operações envolvem apenas um operando e obedecem o seguinte formato:

operador	*operando*
−	5

Nesse caso, o operador negação (−) age no operando 5, tornando-o negativo.

Os exemplos citados acima devem ser utilizados dentro de instruções para serem significativos em Basic. Por exemplo:

```
A=-5
PRINT 6+2
```

Existem quatro categorias para operadores, que são: numéricos; *strings;* relacionais e; lógicos.

NUMÉRICOS

Os operadores numéricos são exclusivos para expressões numéricas. Seus operan-

---

Nota: Os termos *operando*$_1$ e *operando*$_2$ podem ser expressões.

dos devem ser sempre numéricos, assim como o resultado.

Nas descrições abaixo usamos operandos inteiros, de precisão simples e de dupla.

Operações com inteiros envolvem operandos de dois *bytes*, enquanto as de precisão simples, operandos de quatro *bytes* e as de precisão dupla, de oito *bytes*. Quanto maior o número de *bytes* envolvido, mais morosa será a operação.

Existem cinco operadores numéricos distintos, que são:

+ adição
* multiplicação
[potenciação*
− subtração
/ divisão

Adição: A adição é realizada com a precisão do operando mais exato (o outro é convertido). Por exemplo, quando um operando é inteiro e o outro é de precisão simples, o inteiro é convertido em precisão simples e a operação se opera em quatro *bytes*.

Quando um operando é de precisão simples e o outro, de precisão dupla, o primeiro é convertido em precisão dupla e a operação é realizada em oito *bytes*. Por exemplo:

```
PRINT 2+3                    adição de inteiros;
PRINT 3.1+3                  adição de precisão simples;
PRINT 1+1.23456789012345 67  adição de precisão dupla.
```

Subtração: Assim como na adição, na subtração o número menos exato é convertido e a operação é realizada com a precisão do mais exato. Por exemplo:

```
PRINT 33-11                    subtração de inteiros;
PRINT 33-11.1                  subtração de precisão simples;
PRINT 12.34567890123456 7-11   subtração de precisão dupla.
```

Potenciação: Converte ambos os operandos em precisão simples e o resultado é calculado também em precisão simples.

Multiplicação: Mais uma vez, a operação é realizada com a precisão do mais exato. O outro é convertido. Por exemplo:

```
PRINT 33*11                  multiplicação de inteiros;
PRINT 33*11.1                multiplicação de precisão simples;
PRINT 2*12.34567890123456 7  multiplicação de precisão dupla.
```

Divisão: Ambos os operandos são convertidos em precisão simples, ou dupla, dependendo da precisão original, obedecendo às seguintes regras:

— Se pelo menos um dos operandos for de precisão dupla, então o outro é convertido e a divisão é processada em precisão dupla (oito *bytes*);
— Se nenhum dos operandos for de precisão dupla, ambos os operandos são convertidos em precisão simples e a operação é efetuada em precisão simples (quatro *bytes*).

Nota: Para obter o operador [, pressione ↑.

Exemplos:

```
PRINT 3/4                    divisão de precisão simples;
PRINT 3.8/4                  divisão de precisão simples;
PRINT 3/12345678901234567    divisão de precisão dupla.
```

STRING

Em Basic dispõe-se de um único operador *string* (+), o qual permite unir duas ou mais *strings*. Esse operador deve ser usado como parte de uma expressão *string*, na qual tanto os operandos como o resultado serão também *strings*. Essa operação é chamada **concatenação.**

A concatenação de *strings* mantém a ordem na qual as *strings* são colocadas na formulação da operação. A *string* à esquerda do operador ficará à esquerda da segunda *string* na resultante. Por exemplo:

```
PRINT "GATOS"+"AMAM"+"RATOS"   imprime:
GATOSAMAMRATOS
```

Como o Basic não permite que a *string* tenha mais que 255 caracteres, você poderá incorrer em erro se a sua *string* resultante for demasiadamente longa.

RELACIONAIS

Operadores relacionais comparam duas expressões numéricas ou *strings* formando uma expressão relacional. A veracidade dessa relação é verificada e, se confirmada, a expressão relacional resultará em −1, caso contrário, em 0.

Essa relação pode ser numérica ou *string*. A seguir descrevemos cada tipo, separadamente.

Relações numéricas: Os operadores relacionais numéricos são os seguintes:

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

Exemplos de relações numéricas verdadeiras:

```
1<2      2<=5      5>2
2<>5     2>=2      7=7
```

Relações *string:* Os operadores relacionais para *expressões string* são os mesmos que para as numéricas. Porém, possuem significados ligeiramente diferentes. Ao invés de comparar magnitudes numéricas, será verificada a seqüência ASCII das expressões. Isso permite que se confronte dados *string*, da seguinte forma:

<	precede (maior prioridade que);
>	segue (menor prioridade que);
=	tem a mesma prioridade que;
>< ou <>	não tem a mesma prioridade que;

<=	prioridade igual ou maior que;
>=	prioridade igual ou menor que.

As expressões *string* são comparadas caractere por caractere. Quando são encontrados caracteres não-coincidentes, na mesma posição em ambas as expressões, aquela que possuir o caractere com o menor código ASCII terá maior prioridade. Isso é muito útil para se ordenar uma lista de nomes, por exemplo, em ordem alfabética.

Exemplos de relações *string* verdadeiras:

"A"<"B"	o código ASCII de A é 65, e o de B, 66;
"MATE"<"MATO"	o código ASCII de E é 69, e o de O, 79;'

Se, ao efetuar as comparações, o computador chegar ao fim de uma das *strings* sem encontrar caracteres não-coincidentes, a *string* mais curta será considerada como de maior prioridade. Por exemplo:

```
"TRILHA"<"TRILHAR"
"Z 80"<"Z 80A"
```

As margens e os espaços também são considerados como pertencentes às *strings.* Por exemplo:

" A"<"A"	o código ASCII para o espaço é 32.

Utilização das expressões relacionais: Normalmente, expressões relacionais são usadas como teste em uma instrução IF/THEN. Veja este exemplo:

```
IF A=1    THEN PRINT "CORRETO"
```

O Basic verifica se o valor de A é 1. Se essa condição for verdadeira, o Basic mostrará a mensagem "CORRETO", na tela.

```
IF A$<B$  THEN 50
```

Se a *string* armazenada em A$ preceder alfabeticamente a contida em B$, a execução do programa passará para a linha de número 50.

```
IF R$="SIM"    THEN PRINT A$
```

Se a mensagem em R$ for "SIM", então o conteúdo de A$ será mostrado na tela.

No entanto, você pode também utilizar expressões relacionais simplesmente para confirmar a veracidade de uma comparação. Por exemplo:

```
PRINT 7=7
```

Esta instrução imprimirá na tela o valor −1, já que a expressão 7=7 é verdadeira. Já a instrução abaixo imprimirá o valor 0, pois a expressão 7=9 é falsa:

```
PRINT 7=9
```

OPERADORES LÓGICOS

Esses operadores fazem comparações lógicas entre duas ou mais expressões relacio-

---

Nota: Nos apêndices você encontrará uma lista dos caracteres do CP 500 com os respectivos códigos ASCII.

nais, formando as expressões lógicas. Essas expressões são normalmente usadas em instruções IF/THEN. Por exemplo:

```
IF A=1 OR C=2 THEN PRINT X
```

No caso desse exemplo, podemos ler a instrução da seguinte forma: *se* (IF) A é igual a 1 *ou* (OR) C é igual a 2, *então* (THEN) *imprima* (PRINT) o valor de X.

As expressões lógicas podem ser formadas diretamente pela comparação de duas expressões numéricas. Nesse caso, será feita a comparação *bit* a *bit* entre os resultados das expressões numéricas, de acordo com regras pré-estabelecidas para o operador lógico usado.

A tabela a seguir resume a ação dos operadores booleanos na manipulação de *bits:**

Operador	*Significado*	*Operando*$_1$	*Operando*$_2$	*Resultado*
AND	quando ambos os *bits* forem 1, o resultado será 1. Em qualquer outro caso, o resultado será 0;	1 1 0 0	1 0 1 0	1 0 0 0
OR	O resultado será 1 a menos que ambos os *bits* sejam 0;	1 1 0 0	1 0 1 0	1 1 0
NOT	Troca o valor do *bit.*	1 0	—	0 1

HIERARQUIA DOS OPERADORES

Quando as expressões têm vários operadores, a execução das operações por eles definidas segue uma hierarquia bem definida. Dessa forma, o resultado é sempre previsível.

Parênteses: Quando uma expressão completa incluir parênteses, o Basic sempre dará prioridade às operações matemáticas dentro deles. Por exemplo, a expressão:

8 − (3−2)

é calculada na seguinte ordem:

passo 1 3−2=1

passo 2 8−1=7

Notas: Os operandos (expressões, variáveis ou constantes numéricas) são convertidos em inteiros, sendo armazenados como números de 16 *bits* complementar de dois.

Você precisa conservar isso em mente para compreender os resultados das operações *bit* a *bit.*

Os operadores lógicos são definidos de acordo com as regras de Álgebra de Boole (Boole, matemático inglês).

Quando houver vários níveis de parênteses, o Basic começará a calcular pelo mais interno. Por exemplo:

4*(2−(3−4))

é calculada assim:

passo 1 3−4=−1

passo 2 2−(−1)=3

passo 3 4*3=12

Ordem das operações: Quando se efetuar uma seqüência de operações no mesmo nível de parênteses, o Basic usará a hierarquia, mostrada nas duas tabelas a seguir, para definir em que ordem as operações devem ser efetuadas.

Nas tabelas os operadores estão colocados em ordem decrescente de prioridade. Aqueles na mesma linha têm a mesma prioridade e são executados na ordem em que aparecem na expressão, da esquerda para a direita.

Operações numéricas:	
[ou ↑	potenciação;
+ −	sinais-operandos (definem se o valor depois deles é positivo ou negativo);
* /	multiplicação e divisão;
+ −	adição e subtração;
< > = <= >= <>	comparações;
NOT AND OR	operações lógicas.

Operações *string:*	
+	concatenação;
< > = <= >= <>	comparações.

Como ilustração, mostramos um esquema da ordem de operação da expressão abaixo:

X = 2 * 2 + 5[3

↓ ↓

4 125 calcula 2*2 e 5[3

↓

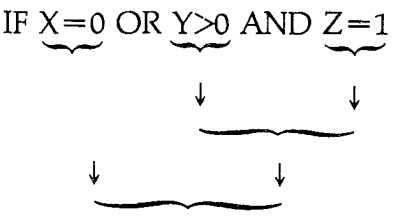
129 calcula 4+125

Se você quiser que as operações sejam executadas em ordem diferente da indicada, acrescente os parênteses. Por exemplo:

2 * (2 + 5[3) = 254

2 * (2 + 5)[3 = 686

Veja agora esse exemplo:



Se estes exemplos lhe parecerem confusos, pela dificuldade em gravar qual a prioridade correta, você pode simplesmente usar parênteses para tornar a seqüência mais clara. Por exemplo:

```
IF X=0  OR  ((Y>0) AND (Z=1)) THEN 255
```

Funções

Uma função é uma seqüência de operações que o Basic executa baseado em dados. Ela é, na verdade, uma sub-rotina que geralmente calcula um valor. Uma função torna desnecessário escrever uma rotina específica em Basic, além de ser mais rápida. Uma palavra chave identifica cada função do seu computador e, geralmente, é seguida por parâmetros que você especifica. Esses parâmetros devem estar sempre entre parênteses e são separados entre si por vírgulas.

Os parâmetros podem ser quaisquer expressões, variáveis ou constantes das já descritas nesse capítulo. Por exemplo:

```
SQR(A+6)
```

Essa função calcula a raiz quadrada do resultado da expressão entre parênteses.

```
MID$(A$,3,2)
```

Essa outra identifica o trecho *(substring)* da *string* em A$ com dois caracteres de comprimento, partindo do terceiro.

As funções não podem existir isoladamente em um programa Basic, isto é, devem ser utilizadas dentro de instruções. Por exemplo:

```
A=SQR(7)
```

Essa instrução faz com que a raiz quadrada de 7 seja armazenada em A.

```
PRINT MID$(A$,3,2)
```

Coloca na tela o terceiro e o quarto caracteres de A$.

Pode-se utilizar funções dentro de expressões, desde que o resultado fornecido por elas sejam compatíveis com os da expressão. Isso quer dizer que funções que resultam em valores numéricos (funções numéricas) só podem ser utilizadas em expressões numéricas, o mesmo valendo para as funções *string*.

Capítulo 18

Como Elaborar uma Expressão em Linguagem Basic

É muito importante saber como elaborar instruções corretas e completas. Neste capítulo discutiremos os dois tipos de expressões que se pode elaborar (numéricas e *strings*) e também como utilizar uma função.

Uma expressão é, na realidade, um conjunto de dados composto por:

constantes;
variáveis;
operadores;
funções.

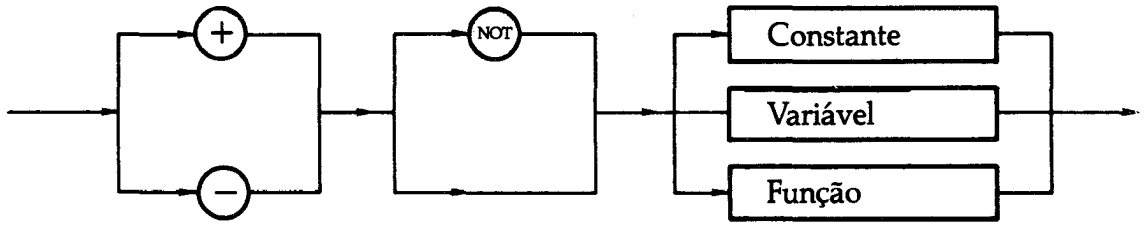
Uma expressão simples consiste em um único termo (constante, variável ou função). No caso de ser um termo numérico, poderá ser precedido por um sinal-operando (+ ou −). Exemplos de expressões numéricas:

```
SQR(8)
+A
3.3
-5
```

Exemplos de expressões *string:*

```
A$
"PALAVRA"
STRING$(20,A$)
"M"
```

Podemos representar a elaboração de uma expressão simples com o seguinte diagrama:



Uma expressão complexa consiste em dois ou mais termos (expressões simples) combinados por operadores. Podemos citar as seguintes expressões complexas como exemplo:

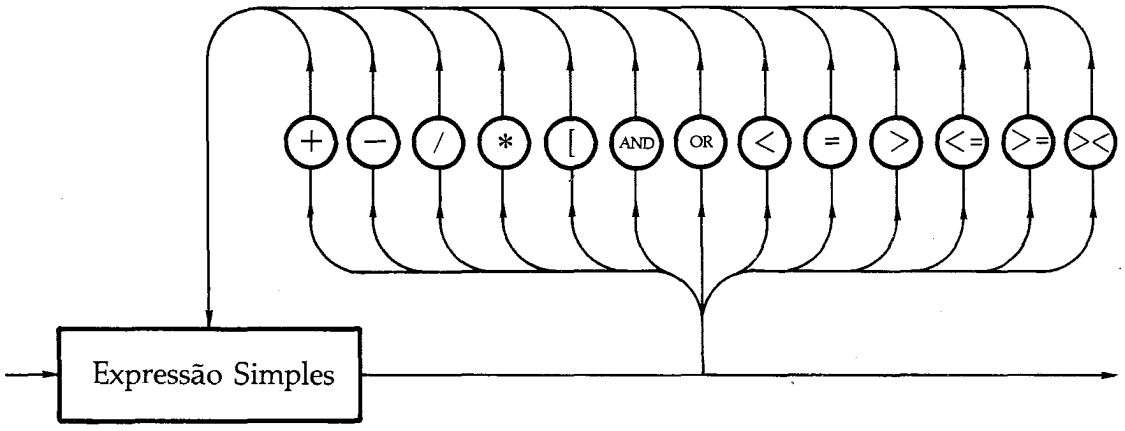
```
A-1
X+3.2-Y
ABS(B)+LOG(2)
A AND B
1=1
```

Observe que pode-se classificar expressões lógicas (A AND B) e relações (1=1) como expressões numéricas complexas, desde que estas resultem em valores numéricos.

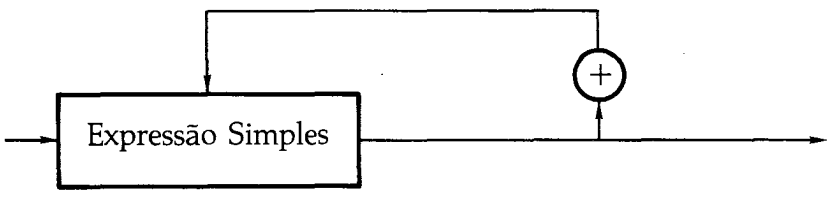
A elaboração de uma expressão *string* complexa fica limitada ao operador de concatenação (+). Veja os seguintes exemplos:

```
A$+B$
"Z"+Z$
STRING$(10,"A")+"M"
```

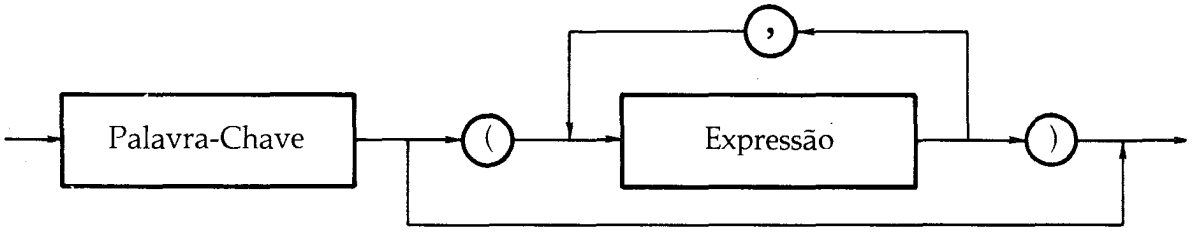
Dessa forma, a elaboração de expressões numéricas complexas difere da de expressões *string*. Para as numéricas, podemos adotar a seguinte representação:



E para as expressões *strings*, esta outra:



As funções, com exceção daquelas que calculam informações internas do sistema, exigem que você introduza pelo menos uma expressão numérica ou *string*. No caso de mais de uma expressão, deve-se manter a compatibilidade dos dados (numéricos ou *strings*). Assim, a elaboração das funções pode ser representada da seguinte maneira:



Se a informação resultante é um número, a função só poderá ser utilizada dentro de uma expressão numérica. O mesmo valendo para funções que resultam em *strings*.

Capítulo 19

Edição

As características de edição do CP 500 facilitam muito o trabalho de correção ou alteração de linhas de seu programa. Abaixo, enumeramos os comandos, subcomandos e funções especiais descritos nesse capítulo:

EDIT

L	SHIFT .	H	Q
ENTER	*n* D	I	E
n ESPAÇO	*n* C	X	*n* K *c*
n ←	*n* S *c*	A	

EDIT número de linha

Este comando ativa o modo de edição. O *número de linha* não pode ser omitido, pois a edição é sempre realizada sobre uma linha, já existente, por vez. Podemos, então, utilizar esse comando de uma das duas formas seguintes:

EDIT. Permite a correção da linha atual do programa (última linha processada, introduzida ou mesmo editada);

EDIT *n* Permite a correção da linha especificada pelo número decimal *n*. Se este número não constar no programa, ocorrerá um erro FC.

Por exemplo:

```
100 FOR I=1 TO 10 STEP .5:PRINT I,I□2,I□3:NEXT
```

Introduza esta linha e digite EDIT 100 ENTER. Aparecerá na tela:

```
100
```

Você agora está no modo de edição e poderá começar a corrigir (ou alterar) o conteúdo da linha 100. Para isso pressione a barra de espaço (ESPAÇO) até que o cursor (_) fique sobre o caractere a ser alterado. As opções de correção são descritas detalhadamente neste capítulo.

Nota: O acionamento do modo de edição apaga todas as variáveis e cancela as operações pendentes de *loop* (FOR/NEXT) ou sub-rotinas (GOSUB).

Se for encontrado um erro de sintaxe em alguma linha durante a execução de um programa, o computador automaticamente ativa o modo de edição para correção desta linha. Não será possível retomar o processamento do ponto onde o erro foi encontrado, sendo necessário reiniciar o programa.

L

Lista o restante da linha que está sendo editada. Não pode ser utilizada quando um dos subcomandos de edição estiver ativado. Ao se pressionar **L**, o cursor passará para a próxima linha da tela, onde estará repetido o número da linha em edição. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione **L** e surgirá:

```
100 FOR I=1 TO 10 STEP .5:PRINT I,I□2,I□3:NEXT
100
```

ENTER

Pressionando-se **ENTER** no modo de edição, serão efetuadas na memória todas as alterações feitas na linha editada e o computador retornará ao modo imediato. As alterações só são efetivamente realizadas após pressionada essa tecla.

n **ESPAÇO**

No modo de edição, ao se pressionar a barra de espaço o cursor é deslocado uma posição à direita mostrando o caractere da posição anterior. Usando ainda a linha 100, acompanhe o exemplo abaixo:

```
EDIT 100 ENTER
100
```

Agora pressione a barra de espaço e o cursor se moverá um espaço, mostrando o primeiro caractere da linha. Pressione a barra até que um caractere diferente de espaço em branco seja mostrado:

```
100 F
```

Para um deslocamento mais rápido pela linha, pode-se digitar o número de caracteres que se quer percorrer antes de pressionar a barra. Continuando com a linha 100, do exemplo anterior:

```
100 F
```

Digite 8 e pressione a barra de espaço, aparecerá:

```
100 FOR I=1 T
```

n **←**

Desloca o cursor para a esquerda por *n* caracteres. Se *n* for omitido, o cursor retornará apenas um caractere. À medida que o cursor se desloca, os caracteres são retirados da tela, permanencendo, contudo, armazenados na memória. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 1 4 ESPAÇO:

```
100 FOR I=1 TO 10
```

Digite 8 e pressione .*:

```
100 FOR I=
```

SHIFT ↑

Pressionando-se essas teclas simultaneamente, os sub-comandos I, X e H serão interrompidos, bloqueando assim a introdução de novas informações.

n D

Estando no modo de edição, esse comando permite eliminar os caracteres à direita do cursor (a começar pelo caractere na posição do cursor). Os caracteres eliminados serão mantidos entre pontos de exclamação, para que se saiba exatamente o que foi eliminado. Por exemplo:

```
EDIT 100 ENTER
100
```

Digite 14 e pressione a barra de espaço:

```
100 FOR I=1 TO 10
```

Agora digite as teclas 7 e D :

```
100 FOR I=1 TO 10 !STEP .5!
```

Nesse exemplo, o trecho STEP .5 será eliminado da linha assim que se pressionar ENTER.

n C

No modo de edição, este comando possibilita a modificação de *n* caracteres da linha, a partir da posição do cursor. Se for omitido *n*, apenas um caractere será modificado. Após esse comando, os próximos *n* caracteres digitados serão considerados como parte do novo texto da linha, e não será preciso usar ENTER após digitá-los. O computador volta automaticamente ao modo de edição. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione L :

```
100 FOR I=1 TO 10:PRINT I,I□2,I□3:NEXT
```

Digite 11 e a barra de espaço:

```
100 FOR I=1 TO
```

Nota: O uso de *n* ← em conjunto com os comandos D, K ou J, pode resultar em visualização não confiável da linha editada.

Pressione, 2 C para trocar dois caracteres e, então, 35:

```
100 FOR I=1 TO 35
```

n S c

Estando no modo de edição, esse comando possibilita a procura da *enésima* ocorrência do caractere *c*. Se o caractere *c* não for encontrado, o cursor irá para o final da linha.

Exemplo:

```
EDIT 100 ENTER
100
```

Pressione 2 S I:

```
100 FOR I=1 TO 35:PRINT
```

Nesse ponto, pode-se utilizar qualquer um dos subcomandos de edição.

H

No modo de edição, o subcomando H permite que seja eliminado o restante da linha, a partir da posição do cursor, e seja introduzido um novo texto. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 3 S I. O computador vai procurar a terceira ocorrência da letra I:

```
100 FOR I=1 TO 35:PRINT I,
```

Pressione H e digite:

```
:NEXT:END ENTER
```

A linha deverá ficar como a seguinte:

```
100 FOR I=1 TO 35:PRINT I,:NEXT:END
```

I

No modo de edição, o subcomando I possibilita a introdução de novas informações na linha. A inserção é realizada na posição do cursor. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione S , (o computador irá procurar pela primeira vírgula que aparecer na linha):

```
100 FOR I=1 TO 35:PRINT I
```

Nota: A tecla de retrocesso (←) não funciona durante o sub-comando de substituição (*n* C). Se você cometer um erro de digitação, terá que reeditar a linha, sem utilizar ← , como normalmente faria em Basic.

Nota: Este subcomando de edição sempre procura o caractere à direita da posição atual do cursor.

Pressione a barra de espaço e digite:

```
IC2,IC3 ENTER
```

A linha editada deve ficar como esta:

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:END
```

X

No modo de edição, o subcomando X lista na tela toda a linha que está sendo editada, posiciona o cursor no final da mesma e, nesse ponto, permite a introdução de novas informações. É muito utilizado para incrementar linhas de programas, adicionando novas funções a essas. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 3 S I (o computador irá procurar pelo terceiro sinal de dois pontos):

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT
```

Pressione, agora, 4 D ENTER (elimina quatro caracteres e armazena a alteração):

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT!:END!
```

Edite novamente a linha, acrescentando o seguinte texto:

```
:PRINT "FIM":END
```

Para isso, siga essas etapas:

```
EDIT 100 ENTER
100
```

Pressione X:

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT
```

E introduza o texto adicional:

```
:PRINT "FIM":END ENTER
```

A linha editada deve ficar:

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM":END
```

A

No modo de edição, o subcomando A cancela as modificações realizadas na linha. O cursor passa para a próxima linha, onde também será repetido o número da linha editada. Esse subcomando só cancela alterações que não tenham sido confirmadas com a tecla ENTER. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 4 S : (para posicionar o cursor sobre o quarto sinal de dois pontos):

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM"
```

Agora, pressione I e digite:

```
:PRINT "DA LISTA"
```

Para sair do subcomando de inserção, pressione SHIFT e 1. Para verificar o novo conteúdo da linha, pressione L :

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM":
PRINT "DA LISTA"
```

Uma vez fora do subcomando de inserção, você pode confirmar a alteração, usando ENTER ou cancelá-la. Para cancelar a alteração introduzida, simplesmente pressione A:

```
100
```

Para verificar se a alteração de fato não se realizou, pressione L :

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM":END
```

E

No modo de edição, este comando é interpretado como fim da edição da linha. As alterações são armazenadas e a edição é terminada. Este comando não é interpretado quando algum outro comando estiver sendo usado. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 4 S : , para posicionar o cursor sobre o último ":":

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM"
```

Pressione H e introduza:

```
:GOTO 100
```

A linha deve ficar assim:

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM":
GOTO 100
```

Pressione SHIFT 1 para sair do comando de inserção e E para indicar o fim da edição:

```
100 FOR I=1 TO 35:PRINT I,IC2,IC3:NEXT:PRINT "FIM":
GOTO 100
```

Q

No modo de edição, este comando cancela as modificações feitas na linha e sai do modo de edição. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 4 S : . Aparecerá:

```
100 FOR I=1 TO 35:PRINT I,I□2,I□3:NEXT:PRINT "FIM"
```

Insira a seguinte instrução, nesse ponto:

```
:CLS
```

Pressionando L você poderá verificar como a linha ficou após essa inserção:

```
100 FOR I=1 TO 35:PRINT I,I□2,I□3:NEXT:PRINT "FIM":CLS:
GOTO 100
```

No entanto, você pode abandonar a edição dessa linha sem que essa alteração seja registrada. Para tanto, pressione Q .

Para ver se a linha 100 não foi realmente alterada, use a instrução LIST 100 ENTER .

n K *c*

No modo de edição, esse comando elimina todo o texto até a *enésima* ocorrência do caractere *c*, e coloca o cursor nessa posição. Por exemplo:

```
EDIT 100 ENTER
100
```

Pressione 3 K P (todo o conteúdo da linha que estiver antes da terceira letra P será eliminada):

```
100 !FOR I=1 TO 35:PRINT I,I□2,I□3:NEXT:!PRINT "FIM":
GOTO 100
```

Se agora você pressionar ENTER ou E , a linha 100 será armazenada com o seguinte conteúdo:

```
100 PRINT "FIM":GOTO 100
```

Capítulo 20

Comandos Basic

Quando o símbolo > aparece na tela de seu computador, este estará operando no modo imediato. Nessa condição, é possível dar ordens diretas ao computador, através de comandos da linguagem Basic. Esses comandos, seguidos de ENTER, serão executados imediatamente. Os comandos permitem várias operações no seu computador como, por exemplo:

1 - Controlar o computador com o intuito de modificar os modos de operação;
2 - Iniciar os processos de entrada e saída de dados;
3 - Alterar a memória de programa, etc.

Todos os comandos, aqui descritos, exceto CONT, poderão ser usados em seu programa como instruções. Isso pode ser útil em alguns casos, onde é necessária uma aplicação específica.

Os comandos do Basic do seu computador são os seguintes:

AUTO	CSAVE	DELETE	LLIST	SYSTEM
CLEAR	CLOAD?	EDIT	NEW	TROFF
CLOAD	CONT	LIST	RUN	TRON

AUTO *número de linha, incremento*

Este comando liga a função automática de numeração de linhas de programa, que agiliza a introdução de programas. Você pode especificar o *número de linha* inicial e o incremento usado entre os números de linhas consecutivas ou, simplesmente, digitar AUTO e pressionar ENTER. Nesse caso, a numeração automática será feita de dez em dez, começando pela linha dez. Toda vez que você terminar uma linha, pressionando ENTER, o número da próxima linha será calculado somando-se o incremento ao número da linha introduzida. O número da nova linha será mostrado no início da linha seguinte, seguido de um espaço.

Exemplos:

Comando	*Numeração*
AUTO	10,20,30...
AUTO 5,5	5,10,15...
AUTO 100	100,110,120...
AUTO 100,25	100,125,150...

Para desativar a função AUTO, pressione a tecla ENTER. Se a faixa de numeração especificada já contiver linhas de programa, todos os números de linhas já existentes serão acompanhados de um asterisco. Caso você não queira modificar a linha, pressione BREAK, e a função será desativada.

CLEAR *n*

Zera todas as variáveis numéricas e anula todas as *strings*, reservando um espaço para essas cujo comprimento é especificado por *n*. Quando omitido o comprimento (*n*), é adotado o valor 50*.

CLOAD *"nome de arquivo"*

Carrega na memória do computador um programa gravado em fita cassete. Para carregar o programa você deve preparar o gravador adequadamente.

Coloque o gravador no modo PLAY e certifique-se de que foram feitas todas as conexões necessárias e de que a fita está na posição correta, ou seja, num ponto em que não seja necessário esperar demais para alcançar a gravação, nem após esta, evidentemente. Não posicione a fita sobre a gravação já iniciada, pois esta poderá não ser reconhecida pelo computador.

Se introduzirmos o comando CLOAD, ou seja, digitarmos CLOAD ENTER, o computador ligará o gravador e carregará o primeiro programa que encontrar na fita.

Podemos, porém, querer que seja carregado um determinado programa, dentre vários na fita. Para isso é possível acrescentar ao comando um *nome de arquivo*. Nesse caso, o computador ligará o gravador e irá "procurar" o programa desejado. À medida que procura, os nomes dos arquivos (gravações) encontrados serão colocados no canto superior direito, juntamente com um asterisco intermitente.

CSAVE *"nome de arquivo"*

Grava o programa armazenado na memória em fita cassete. O *nome do arquivo* pode ser qualquer caractere alfa-numérico, exceto aspas, pois esta delimita o nome. Para essa operação deve-se conectar o gravador corretamente, acionar a gravação (teclas PLAY e RECORD, na maioria dos modelos) e digitar: CSAVE *"nome de arquivo"* ENTER.

CLOAD? *"nome de arquivo"*

Permite a comparação de um programa gravado em fita com o armazenado na memória. Isto é muito útil para se verificar se a transferência de um programa foi bem sucedida, tanto da fita para a memória (CLOAD), quanto ao sentido inverso (CSAVE).

Durante a execução do comando CLOAD? o computador compara os programas (na memória e em fita) *byte* por *byte*. Qualquer *byte* que não coincidente terminará a ve-

---

Nota: Quando se inicializa o computador (com RESET ou ao ligar) é executado automaticamente um CLEAR 50, que reserva 50 *bytes* de memória para as informações *string*.

rificação, aparecendo a mensagem "MAL" na tela. Nesse caso você deverá repetir o comando de transferência (CSAVE ou CLOAD), fazendo os ajustes que julgar necessários no gravador*.

CONT

Esse comando é extremamente útil na depuração de programas.

Durante a execução de um programa, pode-se interromper o processamento com uma instrução STOP, dentro do mesmo, ou pressionando a tecla BREAK. A interrupção geralmente é usada para se verificar o conteúdo das variáveis durante a execução do programa.

A retomada do processamento é realizada pelo comando CONT, seguido de ENTER**.

DELETE *linha inicial-linha final*

Este comando tem como função apagar as linhas de um programa armazenado na memória. Pode-se especificar uma linha individual ou um trecho do programa. Veja os exemplos:

DELETE *linha*	apaga a *linha* especificada;
DELETE *linha inicial - linha final*	apaga todas as linhas de programa dentro da faixa especifi-da, desde a *linha inicial* até a *linha final;*
DELETE -linha final	apaga todas as linhas iniciais do programa, até a especificada.

Se você acabou de introduzir, ou editar, uma linha de programa e deseja eliminá-la por qualquer motivo, use simplesmente "DELETE." (o ponto substitui o número da linha atual).

EDIT *número de linha*

Ativa o modo de Edição do seu computador para que sejam realizadas correções em linha já introduzidas. Quanto maior e mais complexo o seu programa, mais importante será esse comando. O modo de Edição possui seu próprio conjunto de sub-comandos que são detalhadamente descritos no capítulo 19.

LIST *linha inicial-linha final*

Este comando mostra as linhas de programa armazenadas na memória. Se forem omitidos os números de linhas, todo o programa armazenado será listado na tela. Ao

---

Nota: Não se esqueça do ponto de interrogação. Sem ele o computador carregará na memória o programa que encontrar na fita, com o *nome de arquivo,* e esse novo carregamento também deverá ser checado.

Nota: Se for efetuada qualquer alteração no texto do programa através do comando EDIT (capítulo anterior), durante a interrupção, a retomada não será possível, e o computador responderá com uma mensagem de erro.

preencher toda a tela, as linhas já listadas são deslocadas para cima, abrindo espaço para novas linhas na parte inferior da tela. Dessa forma, a listagem de um longo programa passa pela tela continuamente, até que todas as linhas tenham sido listadas. Para paralizar esse "deslocamento", fazendo com que uma parte do programa seja fixada na tela, pressione as teclas SHIFT e @, simultaneamente. Essas teclas forçarão uma pausa na listagem, que continuará ao se pressionar qualquer tecla.

Pode-se listar uma linha em particular, como também um trecho de programa. Veja os exemplos:

LIST	lista na tela todo o programa armazenado;
LIST *linha*	lista a *linha* especificada pelo argumento;
LIST *linha-*	lista as últimas linhas do programa, começando pela *linha* inicial;
LIST *inicial- final*	lista o trecho do programa que começa na linha *inicial* e termina na linha *final;*
LIST *-linha*	lista as primeiras linhas do programa, até a *linha* especificada;
LIST .	lista e linha atual (a última linha introduzida ou corrigida).

LLIST *linha inicial - linha final*

Funciona da mesma forma que LIST, com a única diferença de que a listagem é realizada na impressora e não na tela.

Exemplos:

`LLIST`	lista todo o programa na impressora;
`LLIST .`	lista a linha atual na impressora;
`LLIST 50`	lista a linha 50 na impressora;
`LLIST 100-`	lista as últimas linhas do programa, começando pela de número 100, inclusive;
`LLIST 100-200`	lista todas linhas com números entre 100 e 200, inclusive;
`LLIST -200`	lista todas as linhas iniciais do programa, até a de número 200, inclusive;

NEW

Este comando tem como função apagar um programa na memória, zerar todas as variáveis numéricas e anular as *strings* (torná-las nulas). Ele não modifica o espaço reservado para as *strings* por uma instrução CLEAR *n*.

Pode-se utilizar esse comando tanto diretamente pelo teclado quanto dentro de um programa. Nesse último caso, pode-se criar uma proteção por senha do programa, como é feito no exemplo a seguir.

```
10 INPUT A$:IF A$<>"E" THEN 65520
20 REM
30 REM
40 REM (RESTANTE DO SEU PROGRAMA)
65519 END
65520 NEW
```

Este programa não pode ser executado sem que se introduza a senha correta, no caso, a letra "E".

RUN *número de linha*

Este comando é usado para fazer com que o computador execute um programa armazenado na memória. Existem dois modos para se utilizar esse comando:

RUN ENTER — a execução é iniciada a partir da primeira linha do programa (a de menor número);

RUN *linha* ENTER — a execução começa pela linha especificada pelo argumento.

SYSTEM

Este comando permite que seja carregado um arquivo-objeto (rotinas em linguagem de máquina) na memória de seu computador.

Para carregar esses arquivos (gravados em fita cassete) digite SYSTEM e pressione ENTER. Aparecerão na tela os dois símbolos "*?". Você deverá, então, digitar o nome do programa que quer carregar, seguido de ENTER. A fita começará a ser lida enquanto que, no canto superior direito da tela, surgirão dois asteriscos, um fixo e o outro intermitente.

Assim que o programa estiver na memória voltarão os símbolos "*?" e você poderá executar o programa carregado de um dos seguintes modos:

a) Digitando uma barra seguida do endereço decimal de onde deve partir a execução e ENTER, (/ *nnnnn* ENTER);

b) Digitando apenas / ENTER. Nesse caso, a execução partirá do endereço especificado pelo arquivo-objeto.

Existindo dúvidas a respeito deste comando, retorne ao capítulo "Utilizando a Interface para Gravador".

TRON

Este comando ativa a função TRACE, que permite acompanhar o fluxo de um programa em execução, para análise do algorítmo, localização e correção de erros. Cada

---

Nota: O comando RUN pode ser usado como instrução dentro de um programa.

Nota: Se ocorrer um erro no carregamento do programa, os dois asteriscos no canto superior da tela serão substituídos pelos caracteres "C*". Quando isso ocorrer, recarregue o arquivo-objeto.

vez que a execução avança para uma nova linha, o número da linha executada é colocado na tela entre os caracteres "<" e ">". Veja o seguinte programa:

```
10 PRINT "LINHA 10"
20 INPUT "PRESSIONE <ENTER> PARA INICIAR O LOOP";X
30 PRINT "AQUI VAMOS NOS..."
40 GOTO 30
```

Se for introduzido o comando TRON no computador e depois se executar o programa, assim ficará a tela:

```
<10> LINHA 10
<20> PRESSIONE <ENTER> PARA INICIAR O LOOP?
<30> AQUI VAMOS NOS...
<40><30> AQUI VAMOS NOS...
<40><30> AQUI VAMOS NOS...
  .
  .
  .
```

Se quiser forçar uma pausa na execução do programa, deixando mais tempo para a análise do fluxo, use SHIFT e @ , simultaneamente. Para prosseguir com o programa basta pressionar qualquer tecla.

TROFF

Este comando desativa a função TRACE descrita no comando anterior.

Capítulo 21

Instruções de Entrada e Saída

As instruções descritas nesse capítulo têm como finalidade controlar o fluxo de dados no seu computador, tanto de entrada (teclado, interface serial, gravador, etc.), quanto de saída (tela, impressora, interface serial, etc.).

As seguintes instruções são abordadas nesse capítulo:

PRINT	LPRINT	RESTORE
PRINT @	INPUT	PRINT #−1
PRINT TAB	READ	INPUT #−1
PRINT USING	DATA	

PRINT *lista de itens*

Esta instrução permite a impressão de um item ou uma série de itens na tela. Os itens podem ser constantes, variáveis ou expressões numéricas ou *strings*. A separação entre itens de uma lista é feita por vírgula ou ponto e vírgula.

Quando o interpretador Basic encontrar uma vírgula separando dois itens de uma lista, ele imprimirá o segundo item na próxima *zona de impressão* da tela (a tela é dividida em quatro zonas de impressão, cada uma com 16 colunas). Dessa forma, a vírgula opera como um sinal de tabulação automática muito útil para montar tabelas.

Quando o separador encontrado for o ponto e vírgula, os itens serão impressos lado a lado, sem separação.

a)
```
30 X=5
40 PRINT 25;"E' IGUAL A";X^2
50 END
```

b)
```
80 A$="STRING"
90 PRINT A$;A$,A$;" ";A$
100 END
```

c)
```
130 X=25
140 PRIT 25"E' IGUAL A"X
150 END
```

d)
```
180 A=5:B=10:C=3
190 PRINT ABC
200 END
```

Nota: A pontuação pode ser omitida nos casos em que não houver ambigüidade.

Os dados numéricos são impressos sempre com um espaço em branco adicional, separando-os para maior clareza de leitura. Já aos os dados *string* nada é acrescentado.

No exemplo d, o valor a ser impresso pela instrução da linha 190 será zero. Isso se deve ao fato de que, para o computador, ABC é um **nome de variável**, e não uma multiplicação de três fatores (A*B*C). Como nada foi armazenado na variável ABC, seu conteúdo permaneceu igual a zero.

PRINT @ *posição, lista de itens*

Esta instrução especifica onde os itens devem ser impressos. A *posição* indica o ponto da tela onde a impressão terá início e deve ser um número entre 0 e 1023. Um *layout* da tela, com todas as posições numeradas, pode ser encontrado no apêndice C.

Uma atenção especial deve ser dada à impressão de dados na última linha da tela (posições de 960 a 1023); sempre que um dado é impresso nessa linha, o avanço automático do cursor para a próxima faz com que ocorra um deslocamento para cima de todo o conteúdo da tela. Para evitar esse problema, use um ponto e vírgula no final da lista de itens. Por exemplo:

```
100 PRINT @ 1000,"MENSAGEM";
```

PRINT TAB *(expressão), lista de itens*

Esta instrução permite tabular a informação a ser impressa. A tabulação é realizada dentro da linha onde está o cursor, não podendo ser usada para imprimir dados em outras linhas da tela.

Pode-se tabular individualmente cada item da lista, usando vários TABs dentro da linha de instrução. Por exemplo:

```
10 PRINT TAB(5),"TABULA 5";TAB(25);"TABULA 25"
```

Uma expressão numérica pode ser utilizada para definir a tabulação, como mostra o exemplo a seguir:

```
350 X=3
360 PRINT TAB(X);X;TAB(X*2);X*2;TAB(X*3);X*3
370 END
```

A tabulação é sempre tomada em módulo 64, ou seja, os números maiores que 63 são divididos por 64, e é considerado apenas o resto dessa divisão. Por exemplo, se a *expressão* dentro dos parênteses de TAB resultar em 74, a tabulação usará o valor 10, pois 74÷64=1, com 10 de resto.

Não são aceitos argumentos maiores que 255 para a tabulação.

PRINT USING *formato; lista de itens*

Esta instrução permite definir um formato fixo para impressão dos dados, tanto nu-

---

Nota: O modificador TAB não pode ser usado para deslocar o cursor à esquerda. Se o ponto definido estiver à esquerda do cursor, a tabulação é simplesmente ignorada.

méricos quanto *string*. Seu campo de aplicação é bem amplo, abrangendo impressão de cabeçalhos em relatórios, formatação de tabelas extensas, normalização da apresentação de dados e em outras condições onde seja requerido um formato específico para os dados.

Nessa instrução, a *lista de itens* é impressa de acordo com o formato especificado pela *string formato*. A *string formato* pode ser uma constante, variável ou expressão *string* e os caracteres que contiver são chamados de **especificadores de campo.** Os seguintes especificadores de campo são reconhecidos pelo interpretador Basic de seu CP 500:

Caractere	*Função*
#	Este caractere determina a posição de cada dígito de um valor numérico. Uma seqüência desses sinais define um campo numérico. Se o campo numérico for maior que o número a ser impresso, os dígitos em excesso à esquerda do número, serão preenchidos com espaços e os à direita do ponto decimal, com zeros; se o campo for menor, o número será impresso precedido pelo sinal "%";
.	O ponto decimal pode ser colocado em qualquer lugar do campo numérico, descrito acima. Sua posição define a separação decimal do valor. Quando os dígitos decimais do número a ser impresso superarem os reservados no campo, o número será arredondado para coincidir com o formato do campo;
,	A vírgula, quando encontrada em qualquer ponto do campo numérico à esquerda do ponto decimal, fará com que o número seja impresso com separação de milhares*. Cada vírgula colocada no campo estabelece uma posição a mais;
**	Dois asteriscos colocados no começo do campo farão com que toda posição vaga, dentro dos limites do campo e à esquerda do número, sejam preenchidas por asteriscos. Os dois asteriscos representam dois caracteres a mais no campo numérico;
$	Esse caractere, colocado no início do campo numérico, faz com que seja impresso um cifrão à esquerda do valor. A posição do cifrão é fixa, ficando em branco as posições que não forem preenchidas pelo valor numérico;

Nota: * A notação decimal usada em linguagem Basic é a mesma que em inglês. Por isso é usado o ponto decimal no lugar da vírgula para separar os inteiros das casas decimais. Pelo mesmo motivo, a separação de classes (cada três casas dos inteiros), é realizada pela vírgula. Dessa forma, o número dois milhões, por exemplo, será representado, no computador, da seguinte forma:

2,000,000.00 (dois milhões ou 2000000,00)

Caracter	*Função*
$$	Estes caracteres devem ser colocados no começo do campo e atuam como um cifrão móvel, o qual será impresso imediatamente à esquerda do valor numérico;
**$	Essa combinação de caracteres, quando no início do campo, fazem com que um cifrão móvel seja impresso da maneira descrita acima e, nas posições do campo que não forem preenchidas pelo valor numérico, sejam colocados asteriscos;
↑↑↑↑ ou [[[[	Esses caracteres devem estar colocados no início do campo e fazem com que o número seja impresso em notação científica. Correspondem a quatro posições a mais no campo numérico;
+ ou −	Esses caracteres, quando colocados no início do campo, forçam o número a ser impresso com o respectivo sinal; O sinal de menos, colocado no final do campo numérico, faz com que valores negativos sejam impressos seguidos pelo devido sinal, e os positivos, por um espaço em branco;
!	Esse caractere define um campo *string* de apenas uma posição. É usado para fazer com que apenas o primeiro caractere de uma *string* seja impresso;
%espaços%	Essa combinação de espaços e caracteres define um campo *string* para colocação de mensagens, ou outras informações quaisquer. O comprimento do campo será dado pelo número de espaços mais dois, ou seja, o comprimento total da combinação;

Todos os caracteres que compõem a *string formato*, e que não forem interpretados como *especificadores de campo* (inclui-se aqui tanto os demais caracteres ASCII quanto os próprios especificadores que não satisfaçam as condições acima), serão tratados como uma *string* comum.

Exemplo:

Instrução	*Resultado*
`PRINT USING"##.#";12.12`	`12.1`
`PRINT USING"##.#";1.34`	`1.3`
`PRINT USING"###.#";1000.33`	`%1000.3`
`PRINT USING"##.##";12.127`	`12.13`
`PRINT USING"##.##";12.123`	`12.12`
`PRINT USING"+##.##";12.12`	`+12.12`

continua

```
PRINT USING"+##.##";-12.12                    -12.12
PRINT USING"##.##+";12.12                     12.12+
PRINT USING"##.##-";12.12                     12.12
PRINT USING"##.##-";-12.12                    12.12-
PRINT USING"**##";12.12                       **12
PRINT USING" $###.#";12.12                    $ 12.1
PRINT USING"$$##.##";12.12                    $12.12
PRINT USING"** $##.##";12.12                  ** $12.12
PRINT USING"#,###,###.##";1234567,89          1,234,567,89
PRINT USING"!";"ABCDE"                        A
PRINT USING"%%";"ABCDE"                       AB
PRINT USING"% %";"ABCDE"                      ABC
```

Na utilização do especificador "!", pode-se obter resultados interessantes para múltiplas *strings*. Esse caractere, como foi explicado, faz com que seja impresso apenas o primeiro caractere da *string* e, por isso, pode ser usado para se conseguir abreviaturas ou siglas dentro de programas. Veja o exemplo nesta instrução:

```
PRINT USING"!";"EDYLENE";"SANDRA";"HUMBERTO"
```

Ao pressionar **ENTER**, será impresso o seguinte:

```
ESH
```

Pode-se definir múltiplos campos dentro de um mesmo formato. Para isso é necessário usar vários especificadores. Nesse caso, cada informação a ser impressa terá seu formato definido pelo campo correspondente.

Se, ao terminar os campos de um formato, ainda existirem informações a ser impressas, o computador repetirá o formato, até que não haja mais informações pendentes. Foi isso que aconteceu no exemplo acima; o computador imprimiu o primeiro nome usando o formato "!", como havia ainda mais dois nomes para imprimir, esse formato foi repetido.

Um exemplo de vários campos dentro de um mesmo formato seria o seguinte:

```
PRINT USING"! ! !";"EDYLENE";"SANDRA";"HUMBERTO"
```

Essa instrução imediata tem como resultado na tela o seguinte:

```
E S H
```

Nesse caso, os espaços dentro do formato foram incluidos na informação impressa.

O programa a seguir ilustra o uso da instrução PRINT USING:

```
510 CLS
520 A$=" * * $##########.## cruzeiros"
530 INPUT"DIGITE O SEU PRIMEIRO NOME";P$
540 INPUT"E O SEU SEGUNDO (P/ PULAR, DIGITE <ENTER>)";N$
550 INPUT"DIGITE O SEU SOBRENOME";S$
560 INPUT"DIGITE A QUANTIA A SER PAGA";M
570 PRINT:PRINT"Pague a quantia de` ";
580 PRINT USING A$;M
590 PRINT USING"a ! ! %                    % ou a sua ordem";
P$;N$,S$: REM          % 16 ESPACOS     %
600 END
```

Ao executar esse programa, você deve introduzir os dados que achar convenientes e observar o resultado, principalmente com relação à forma de apresentação. Usamos abaixo um exemplo da utilização desse programa:

```
DIGITE O SEU PRIMEIRO NOME?PAULO
E O SEU SEGUNDO (P/ PULAR, DIGITE <ENTER>)SOARES
DIGITE O SEU SOBRENOME?LIMA
DIGITE A QUANTIA A SER PAGA?1234567

Pague a quantia de * * $ 1234567.00 cruzeiros
a P S LIMA                    ou a sua ordem
```

Se você quiser adaptar o programa para usar importância sem arredondamento, ou mesmo usar notação científica, acrescente o declarador de tipo de precisão dupla (#) após a variável M, nas linhas 560 e 580. Com essa medida você poderá usar valores com até 16 algarismos significativos.

LPRINT

Esta instrução é similar à PRINT com exceção de dois detalhes: o primeiro é que a impressão é realizada em uma impressora, no lugar da tela; e o segundo ponto é que LPRINT não pode ser usada com o modificador @. Por exemplo:

```
LPRINT TAB(5)"NOME"TAB(30)"ENDERECO"STRING$(63,32)
"BALANCO"
```

Essa instrução imprime "NOME" a partir da coluna 5, "ENDEREÇO" a partir da coluna 30 e "BALANÇO", a partir da centésima coluna.

INPUT *lista de itens*

Esta instrução tem como finalidade suspender a execução de um programa para que dados sejam introduzidos através do teclado. A instrução INPUT pode especificar uma lista de variáveis *string* ou numéricas a serem introduzidas, ou uma combinação de ambas.

Os itens da *lista* devem ser separados por vírgulas, e os valores podem ser introduzidos individualmente ou em grupo. Para se introduzir os valores de uma só vez, basta separá-los por vírgulas. Se os dados forem introduzidos separadamente, ao se introduzir o primeiro aparecerão dois pontos de interrogação indicando que mais dados devem ser in-

troduzidos. Os pontos de interrogação continuarão aparecendo após cada introdução até que todas as informações pedidas pela instrução INPUT tenham sido introduzidas. Acompanhe o seguinte exemplo:

```
100 INPUT X$,X1,Z$,Z1
```

Ao ser executada essa linha de programa, o computador vai esperar pela introdução de uma *string*, seguida de um dado numérico, outra *string* e outro número, nessa ordem. Para proceder a introdução, você precisa apenas digitar a *string*, seguida dos demais dados, separando-os por vírgulas, na mesma ordem em que aparecem na instrução. As *strings* não precisam estar entre aspas (a menos que você queira introduzir uma mensagem que comece com espaços em branco ou possua caracteres especiais, como vírgulas ou dois pontos).

Resumindo: para que a instrução acima seja executada corretamente, você deve digitar os dados num dos seguintes formatos:

a) $item_1$, $item_2$, $item_3$, $item_4$ ENTER

b) $item_1$ ENTER
$item_2$ ENTER
$item_3$ ENTER
$item_4$ ENTER

Não se esqueça de que os itens introduzidos devem ser compatíveis com as variáveis referenciadas na instrução INPUT. Se você tentar introduzir um valor *string* em uma variável numérica, o computador indicará a incoerência mostrando a mensagem:

```
?REDO
?
```

Essa mensagem pede que a introdução seja refeita (*redo* é uma palavra em inglês que significa "refazer"). O ponto de interrogação abaixo da mensagem indica que o computador espera por uma nova introdução.

No caso de se introduzir um valor numérico em uma variável *string*, a incoerência não é percebida pelo computador. Isso é natural, pois o computador, ao pedir uma *string*, armazenará qualquer seqüência de caracteres que for digitada, e isso inclui números.

Outro caso que deve ser mencionado é quando você introduzir uma linha que contenha mais informação do que foi pedido. Nesse caso, o computador mostrará a mensagem:

```
EXTRA IGNORADO
```

Essa mensagem indica que as informações excedentes foram simplesmente ignoradas pelo computador, e a execução do programa prossegue normalmente.

Quando se introduz dados pressionando ENTER sem ter digitado mais nada, a variável correspondente não tem seu conteúdo alterado. Se nada tiver sido armazenado anteriormente, seu valor será nulo.

Nota: Não se pode introduzir uma expressão em uma variável numérica com a instrução INPUT. Deve ser introduzida uma constante numérica simples.

Você pode melhorar a introdução de dados durante um programa usando instruções PRINT, antes de INPUT, que coloquem mensagens (lembretes) que façam referência às variáveis a serem introduzidas. Isso é uma prática normal em linguagem Basic padrão.

No entanto, a linguagem Basic do CP 500 permite que esses "lembretes" sejam incluídos dentro da própria instrução INPUT. A mensagem deve ser colocada entre aspas logo após a palavra INPUT, e deve ser seguida por ponto e vírgula, como na seguinte linha de programa:

```
10 INPUT"DIGITE SEU NOME";N$
```

Ao executar essa linha, o computador mostrará a mensagem "DIGITE SEU NOME" seguida de um ponto de interrogação.

DATA *lista de itens*

Permite o armazenamento de dados dentro do próprio programa. Esses dados são acessados no decorrer do programa pela instrução READ, descrita mais adiante.

Um mesmo programa pode conter várias *linhas de dados* (é como são chamadas as linhas de programa que usam a instrução DATA), formando uma única *lista de itens.*

Os itens são lidos seqüencialmente, ou seja, não se pode "ler" (*read* em inglês) um determinado item no meio da *lista* sem que o precedente tenha sido acessado. Portanto, a primeira vez que se executar a instrução READ, o primeiro item da primeira linha de dados será lido; na segunda vez, o segundo item; e assim por diante, até que o último item da última linha de dados tenha sido acessado.

Os itens, ou dados, podem ser constantes *strings* ou numéricas e devem ser separados entre si por vírgulas. As *strings* não precisam estar entre aspas, a menos que comecem com espaços em branco ou contenham vírgulas ou ponto e vírgula.

As linhas de dados podem ser colocadas em qualquer ponto do programa, e não necessitam ser consecutivas. Em geral, são colocadas todas juntas no início ou no final do programa; por uma questão de documentação, simplesmente.

O programa abaixo ilustra a forma geral de uma instrução DATA:

```
10 READ N1$,N2$,N3$,N4$
20 DATA ESTE E' O ITEM 1,ESTE E' O 2,"ESTE, O 3", O 4
30 PRINT N1$,N2$,N3$,N4$
```

Para compreender melhor sua aplicação, veja também READ e RESTORE.

READ *lista de variáveis*

Tem como função instruir o computador a ler valores na *lista de dados* (instrução DATA) e armazená-los nas variáveis correspondentes, na ordem em que aparecem na *lista de variáveis.* O tipo de cada dado lido deve coincidir com o tipo da variável no qual será armazenado, caso contrário ocorrerá um erro e o programa será interrompido.

A leitura da lista de dados é seqüencial, o que quer dizer que os dados devem ser lidos na ordem em que aparecem na *lista de dados.* Não se pode ler uma informação que es-

teja no meio da lista sem ter lido a informação anterior. Portanto, a primeira vez que a instrução READ for executada, serão lidos os primeiros *n* dados da *lista* (*n* é o número de variáveis que compõem a *lista* da instrução READ), que começa na primeira instrução DATA. Da segunda vez, serão lidos os *n* seguintes, e assim sucessivamente, até o último dado da última instrução DATA ser lido.

Se for tentado ler mais dados do que existem na lista de dados do programa, ocorrerá um erro de "falta de dados", e a execução será interrompida. O programa a seguir ilustra uma aplicação típica das instruções READ e DATA:

```
700 PRINT"NOME","IDADE"
710 READ N$
720 IF N$="FIM" THEN PRINT"FIM DA LISTA":END
730 READ IDADE
740 IF IDADE<18 THEN PRINT N$,IDADE
750 GOTO 710
760 DATA "LIMA,JOAO",30,"BARBOSA,A.M.",20
770 DATA "SILVA,JOAO",15,"APARECIDA,MARIA",21
780 DATA "TERRA,RENATO",17,FIM
```

Este programa localiza e imprime todos os nomes de pessoas cuja idade seja inferior a 18 anos. Observe o uso da *string* "FIM" para marcar o final da lista de dados. Esse recurso é muito útil em lista cujo número de itens seja desconhecido.

RESTORE

É usada em conjunto com DATA e READ. Faz com que a leitura da lista de dados retorne ao primeiro item. Sua aplicação abrange os casos em que é necessário ler os dados mais de uma vez. No exemplo abaixo, o primeiro dado da lista é lido duas vezes e armazenado em variáveis distintas, devido ao uso da instrução RESTORE.

```
810 READ X
820 RESTORE
830 READ Y
840 PRINT X,Y
850 DATA 50,60
860 END
```

PRINT # -1, *lista de itens*

Tem como função gravar em fita os dados relacionados na *lista de itens*. Para tanto o gravador deve estar devidamente preparado (conectado e com a função RECORD acionada) quando esta instrução for executada. Por exemplo:

```
900 PRINT #-1,A1,B$,"E' TUDO"
910 END
```

Depois de executado esse programa, ficarão gravados em fita os conteúdos das variáveis A1 e B$, além da constante *string* "E' TUDO".

INPUT # -1, *lista de itens*

Realiza a função inversa de PRINT # -1, carregando na memória valores gravados por esta em fita cassete. A instrução abaixo, por exemplo, carregará na memória os valo-

res gravados em fita pelo programa de exemplo de PRINT #-1 (linhas 890 a 910), mostrado anteriormente.

```
INPUT #-1,X,P$,T$
```

Note que os nomes das variáveis não são os mesmos, mas que os tipos coincidem: uma variável numérica e duas *strings*. Não se esqueça que o gravador deve ser apropriadamente preparado para "enviar" essas informações para o computador.

As informações recebidas são armazenadas nas variáveis mencionadas na lista seguindo a ordem de entrada.

Se for carregada uma *string* quando o computador estiver aguardando um valor numérico, o processamento será interrompido e uma mensagem de erro será colocada na tela.

Outro erro que pode ocorrer é o de falta de dados. Nesse caso, não foram encontrados dados em número suficiente na fita para armazenar em todas as variáveis da lista de itens.

Resumindo, a *lista de itens* da instrução INPUT #-1 deve ter o mesmo número de variáveis, e na mesma disposição, que a instrução PRINT #-1 que gerou a gravação.

Como exemplo, utilize o programa apresentado na descrição da instrução PRINT #-1 para gravar dados na fita e, depois, carregue-os de volta com o programa abaixo.

Não se esqueça de rebobinar a fita para colocar na posição inicial da gravação antes de tentar carregar os dados de volta. Depois de convenientemente acertada a fita (e o gravador conectado ao computador), coloque o gravador na condição de tocar a fita (PLAY) e execute o programa:

```
10 INPUT #-1,A1,B$,L$
20 PRINT A1,B$,L$
30 IF L$="E' TUDO" THEN END
40 REM O PROGRAMA PODERA' VOLTAR A LINHA 10
```

Nota: Os valores apresentados na lista de itens não devem exceder a 248 *bytes* no total, caso contrário toda informação além desse limite não será gravada. Por exemplo, na instrução a seguir, o conteúdo de A$ não poderá ter mais que 75 caracteres, sob pena de não serem gravados os caracteres finais *dessa string:*

```
PRINT #-1,A#,B#,C#,D#,E#,F#,G#,H#,I#,J#,A$
```

Aconselha-se a dividir as listas de itens muito grandes em várias instruções PRINT -1 distintas.

Capítulo 22

Instruções de Programação

A linguagem Basic impõe vários pré-requisitos com relação ao modo de execução de seu programa. Os principais são:

— Qualquer variável utilizada em seu programa será considerada uma variável de precisão simples, a menos que seja usado um declarador de tipo ou uma instrução de definição;

— A execução é seqüencial, começando pela linha de menor número e terminando na de maior, caso não haja desvios.

As instruções descritas neste capítulo permitem anular os pré-requisitos da linguagem, fornecendo maior versatilidade e potência aos seus programas.

Definição de tipos:

DEFINT
DEFSNG
DEFDBL
DEFSTR

Atribuição e Alocação:

CLEAR *n*
DIM
LET

Seqüência de execução:

END	ON...GOTO	ON ERROR GOTO
STOP	ON...GOSUB	RESUME
GOTO	FOR/NEXT/STEP	REM
GOSUB	ERROR	

Testes (Instruções condicionais):

IF...THEN...ELSE

DEFINT *faixa*

Faz com que toda variável cujo nome comece por qualquer letra dentro da *faixa* especificada seja armazenada e manipulada como variável inteira, a menos que o nome contenha um caractere de declaração de tipo.

Essa instrução economiza memória, uma vez que os valores inteiros ocupam menos espaço de memória do que os demais, além de agilizarem o processamento. A grande restrição feita aos inteiros é a relativamente limitada faixa de trabalho, que se situa entre −32768 e 32767, inclusive. Para compreender melhor o uso dessa instrução, observe a seguinte linha de programa:

```
10 DEFINT A,I,N
```

Depois de ter executado esta linha, o computador tratará todas as variáveis cujos nomes comecem por A, I ou N como variáveis inteiras. Dessa forma, os seguintes nomes seriam considerados como de variáveis inteiras:

```
A1, AA, I3, N, etc...
```

No entanto, se nesse mesmo programa fossem encontrados nomes de variáveis dentro dessa faixa, com outros tipos declarados como, por exemplo, A1 # ,AA # ou I3# , estas seriam tratadas como de precisão dupla, respeitando-se os caracteres de declaração. Conclui-se, então, que **os caracteres de declaração têm prioridade sobre as instruções de definição (DEF).**

Além da maneira mostrada no exemplo anterior, pode-se definir o tipo das variáveis dentro de uma faixa de letras, como na seguinte instrução:

```
10 DEFINT I-N
```

Esta faz com que as variáveis cujos nomes comecem pelas letras I, J, K, L, M e N sejam manipuladas como inteiras.

A instrução DEFINT pode ser colocada em qualquer ponto do programa, porém, só tem efeito sobre os nomes de variáveis que forem processados após sua execução.

Portanto, para manter a uniformidade do programa, esta instrução deve ser executada antes de qualquer referência às variáveis.

DEFSNG *faixa*

Faz com que as variáveis cujos nomes comecem por letras dentro da *faixa* especificada sejam tratadas como de precisão simples, desde que estes nomes não contenham caracteres de declaração de tipo. Nesse caso, o tipo declarado no nome prevalece. Constantes e variáveis de precisão simples são armazenadas com sete algarismos significativos e impressas (na impressora ou na tela) com seis.

Como, inicialmente, o computador assume que todas as variáveis sem declaradores de tipo são de precisão simples, essa instrução só precisa ser utilizada para redefinir faixas que tenham sido afetadas pelas demais instruções de definição (DEFINT,DEFDBL ou DEFSTR). O programa abaixo ilustra esse caso:

```
10 DEFINT A-Z
20 DEFSNG N
```

Nesse programa, todas as variáveis (de A a Z) são definidas como inteiras ao ser executada a linha 10. Depois disso, a instrução na linha 20 faz com que as variáveis com nomes que comecem com N voltem a ser consideradas como de precisão simples. Entre-

tanto, nomes como N% e N# continuam definindo variáveis inteira e de dupla precisão, respectivamente, pois os caracteres de declaração têm precedência sobre as instruções de definição.

DEFDBL *faixa*

De utilização similar a DEFINT e DEFSNG, esta instrução tem como função definir faixas de nomes de variáveis que serão tratadas como de precisão dupla. Os nomes que tiverem iniciais dentro da *faixa* especificada, e não tenham declaradores de outro tipo, identificarão variáveis de precisão dupla. Por exemplo:

```
10 DEFDBL D,P-T
```

Essa linha de programa faz com que variáveis com nomes que comecem por D, P, Q, R, S ou T, sejam classificadas como de precisão dupla.

Os valores de precisão dupla são armazenados com 17 algarismos significativos, sendo impressos com 16. Isso confere maior precisão aos cálculos ocupando, porém, mais espaço na memória e levando mais tempo no processamento.

DEFSTR *faixa*

Assim como as três anteriores, essa instrução define qual a classificação das variáveis cujos nomes não possuem declaradores de tipo. Após sua execução, os nomes de variáveis sem declaradores e iniciados por letras dentro da *faixa* especificada identificarão variáveis *string*. Por exemplo:

```
10 DEFSTR L-Z
```

Após executar essa linha, o computador tratará todas as variáveis com nomes que comecem por letras de L a Z, e que não possuam declaradores de tipo, como *strings*. Portanto, as *strings* poderão ser referidas nesse programa da seguinte forma:

```
20 L="MENSAGEM":MES="AGOSTO"
```

Note que não é necessário acrescentar o cifrão ($) ao nome da variável.

REM *comentário*

Possibilita a inserção de comentários dentro de seu programa. O conteúdo da linha após REM não é interpretado pelo computador, tendo significado apenas para quem examinar a listagem do programa. Como exemplo, tomemos o caso de um trecho de programa que calcule a área de um triângulo. Podemos escrever esse trecho da seguinte forma:

```
690 REM ** AS LINHAS COM REM AJUDAM A **
700 REM ** ENTENDER O PROGRAMA **
710 REM
720 REM ** ROTINA DE CALCULO DA AREA **
730 A=B*H/2
740 REM ** A=AREA  B=BASE  H=ALTURA  **
```

Qualquer caractere alfa-numérico pode ser usado dentro de um *comentário*, e este pode ter até 255 caracteres, no total.

CLEAR *n*

Reserva *n bytes* de memória para armazenamento de *string*. O argumento *n* pode ser uma constante ou variável numérica. Além dessa função, CLEAR *n* também zera todas as variáveis numéricas e anula as *strings*.

Ao ligar o CP 500 é executado automaticamente uma instrução CLEAR 50.

O espaço reservado para armazenamento de *strings* deve comportar todos os caracteres necessários para o processamento desses valores. É importante salientar que, para realizar operações com *strings*, o computador utiliza variáveis auxiliares, as quais não são mencionadas no programa. Portanto, pode faltar espaço para armazenamento durante a execução de um programa se esse aspecto não for considerado na escolha do espaço necessário. Por exemplo:

```
10 CLEAR 1000
```

A instrução acima reserva na memória 1000 *bytes* para acomodar os valores *string* que serão utilizados pelo programa.

Um programa que não utilize *strings* pode conter uma instrução CLEAR 0, o que aumenta a área de memória para os dados numéricos e para o próprio programa.

O argumento *n* pode ser uma constante, variável ou expressão numérica e, evidentemente, não pode ser negativo.

DIM *nome* ($dim_1, dim_2, \ldots, dim_k$)

Possibilita definir o número de elementos por dimensão de uma matriz, ou de uma lista de matrizes. As matrizes utilizadas em um programa que não tiverem sido previamente dimensionadas (usando-se DIM) terão 11 elementos por dimensão, numerados de 0 a 10. Por exemplo:

```
DIM A(5),B(2,3),C$(20)
```

Esta instrução dimensiona a matriz A() com uma dimensão de seis elementos (0 a 5); a matriz B() com duas dimensões, a primeira com três elementos (0 a 2) e a segunda com quatro (0 a 3); e a matriz *string* C$ com uma dimensão de 21 elementos (0 a 20). Se não houver definição anterior a essa instrução que modifique a classificação das variáveis, as matrizes A e B serão numéricas de precisão simples.

As instruções DIM podem ser colocadas em qualquer ponto do programa e os argumentos ($dim_1, \ldots dim_k$) devem ser constantes ou expressões numéricas. Por exemplo:

```
40 INPUT"NUMERO DE NOMES";N
50 DIM N$(N,2)
```

Nota: O apóstrofo (') pode ser usado no lugar da palavra chave REM. Exemplo:

```
750 ' O APOSTROFO SUBSTITUI "REM"
```

Para poder redimensionar uma matriz, mudando o numero ou tamanho de suas dimensões, use a instrução CLEAR (com ou sem argumento) antes de usar DIM. Sem esse procedimento o redimensionamento não é permitido e provoca um erro. Por exemplo:

```
40 AA(4)=11.5
50 DIM AA(7)
```

Ao executar esse programa, o computador mostrará a mensagem:

```
?DD na 50
READY
>
```

LET *variável=expressão*

Esta instrução faz com que o resultado da *expressão* seja armazenado na *variável.* Para o CP 500 a palavra chave (LET) é opcional. Por exemplo:

```
100 LET A$="UMA VARIAVEL STRING"
110 LET B2=1.23E-4
120 B1=1.23
```

Note que as linhas 110 e 120 se equivalem.

END

Essa instrução finaliza a execução do programa. Em muitas versões da linguagem Basic, esta deve ser a última instrução executada pelo computador. No CP 500 isso não acontece, podendo ser omitida a instrução END se a última linha na memória for a última a ser executada. No CP 500, a principal utilização dessa instrução é a de finalizar o programa em qualquer ponto que não seja a última linha. Por exemplo:

```
10 INPUT S1,S2
20 GOSUB 100
30 REM          MAIS LINHAS DE PROGRAMA AQUI
99 END:   REM  BLOCO PROTEGIDO POR END
100 H=SQR(S1*%1+S2*S2)
110 RETURN
```

STOP

Esta instrução interrompe a execução do programa e mostra a mensagem "Break na", seguida do número da linha onde ocorreu a interrupção.

Durante a interrupção, você pode examinar ou modificar os valores das variáveis. Por exemplo:

```
10 X=RND(10)
20 STOP
30 GOSUB 1000
99 END
1000 REM
1010 RETURN
```

Suponhamos que precisemos saber que valor de X será transferido para a sub-rotina da linha 1000. Durante a interrupção da linha 20, pode-se ver qual o valor de X com uma linha imediata como PRINT X*.

GOTO *número de linha*

Transfere o controle do programa para a linha de número especificado pelo argumento. Por exemplo:

```
200 GOTO 10
```

Esta linha fará com que a próxima linha a ser executada seja a de número 10.

Esta instrução pode ser utilizada no modo imediato em substituição a RUN, para se executar um programa sem perder as variáveis.

GOSUB *número de linha*

Transfere a execução do programa para a sub-rotina que inicia na linha especificada. Além disso, o endereço onde a instrução GOSUB se encontra é armazenado para que, após a execução da sub-rotina, o processamento seja retomado nesse ponto. A área da memória onde são armazenados os vários endereços de retorno é chamada de *stack*.

A finalização da sub-rotina é identificada pela instrução RETURN. Ao encontrar a instrução RETURN, o computador "pega" o último endereço armazenado do *stack* e transfere a execução para esse ponto. O endereço é apagado do *stack*.

Se não for usada a instrução RETURN, cada nova instrução GOSUB fará com que um endereço seja acrescentado no *stack*, o que pode sobrecarregar essa área da memória, ficando o computador em uma condição de erro *(Overflow)*.

Assim sendo, para cada instrução GOSUB deve existir uma instrução RETURN correspondente. Por exemplo:

```
100 GOSUB 200
110 PRINT "VOLTA DA SUB-ROTINA":END
200 PRINT "EXECUTANDO A SUB-ROTINA"
210 RETURN
```

Ao executar esse programa, notamos que a mensagem na linha 200 é colocada na tela antes da mensagem na linha 110. O que ocorreu foi um desvio na seqüência de execução da linha 100 para a linha 200. Após executar a linha 200, o computador passou para a linha seguinte (210), onde a instrução RETURN fez com que o processamento retornasse ao ponto imediatamente após a instrução GOSUB, no caso, a linha 110.

RETURN

Veja GOSUB.

Nota: Para continuar o processamento, use o comando CONT.

ON *expressão* GOTO *1.º número de linha,...*

Esta instrução permite múltiplos desvios controlados por uma expressão ou variável numérica.

O valor da *expressão* (ou o conteúdo da variável) deve estar entre 0 e 255, inclusive. Fora dessa faixa ocorrerá um erro e o processamento será interrompido.

Quando ON...GOTO for executada, será considerada a parte inteira desse valor. Esse número inteiro define a posição, na lista de números, do número de linha para onde será desviado o processamento.

Se essa posição não existir, ou seja, se o valor for zero ou for maior que o número de linhas na lista, o computador passa para a próxima instrução do programa. Por exemplo:

```
100 ON MI GOTO 150,160,170,150,180
```

Nessa instrução, em primeiro lugar será tomada a parte inteira do conteúdo de MI (1.99, por exemplo, é considerado 1). Se o inteiro for zero ou maior que cinco (na lista, cinco linhas são mencionadas), o computador passará para a linha seguinte. Qualquer número de um a cinco indicará qual a posição na lista do número da próxima linha a ser executada. Por exemplo, o número um fará a execução passar para a linha 150, pois esse é o primeiro número da lista. O número dois passa a execução para a linha 160, e assim por diante, até o número cinco, o qual fará a linha 180 ser acionada.

ON *expressão* GOSUB *1.º número de linha,...*

Opera da mesma forma que ON...GOTO, porém ativando uma sub-rotina, ao invés de provocar apenas um desvio. Após terminada a execução da sub-rotina, o processamento retorna à instrução seguinte à ON...GOSUB. Por exemplo:

```
100 INPUT"ESCOLHA 1,2 OU 3";I
110 ON I GOSUB 200,300,400
120 PRINT "FIM DO PROGRAMA":END
200 PRINT "SUB-ROTINA #1":RETURN
300 PRINT "SUB-ROTINA #2":RETURN
400 PRINT "SUB-ROTINA #3":RETURN
```

As mesmas restrições se aplicam a ON...GOSUB e ON...GOTO.

FOR/NEXT

Essa combinação de instruções é capaz de controlar o número de vezes que se quer repetir um dado trecho do programa. No início do trecho é colocada a instrução FOR, obedecendo-se o seguinte formato:

FOR *variável=valor inicial* TO *valor final* STEP *incremento*

Nessa instrução, os valores *inicial* e *final*, assim como o *incremento*, podem ser constantes, variáveis ou expressões numéricas. Quando essa instrução for executada, esses três valores serão avaliados e armazenados. Depois disso, as variáveis utilizadas para definir qualquer desses três valores poderão sofrer alteração sem influir na contagem das

repetições. Modificações no conteúdo da *variável* influenciam diretamente na contagem do *loop* (trecho a ser repetido).

O trecho termina na instrução NEXT *variável.*

A contagem das repetições é realizada da seguinte forma:

1º Ao ser executada a instrução FOR/STEP, o *valor inicial* é armazenado na *variável;*

2º O processamento prossegue até atingir a instrução NEXT, onde é somado ao conteúdo da *variável* o valor do *incremento;*

3º Nesse ponto, o conteúdo da *variável* é comparado com o *valor final.* Se tiver ultrapassado esse valor, estarão terminadas as repetições e o programa continua na instrução seguinte à NEXT. Se ainda não tiver sido ultrapassado o *valor limite,* o processamento é retomado na instrução seguinte à FOR.

Exemplos de *loops:*

a)
```
10 FOR I=10 TO 2 STEP -1
20 PRINT I;
30 NEXT I
```

Se você introduzir e executar o programa acima obterá a seguinte resposta na tela:

```
10  9  8  7  6  5  4  3  2
```

b)
```
10 FOR K=0 TO 1 STEP .3
20 PRINT K;
30 NEXT
```

Se você executar o programa acima, obterá a seguinte resposta:

```
0  .3  .6  .9
```

c)
```
10 J=3:K=8:L=2
20 FOR I=J TO K+L STEP L
30 J=0:K=0:L=0
40 PRINT I;
50 NEXT I
```

Se você executar o programa acima, obterá a seguinte resposta:

```
3  5  7  9
```

As variáveis e expressões na linha 20 são avaliadas uma vez. A partir daí, estes valores tornam-se constantes para o loop FOR/NEXT/STEP. Uma modificação posterior de valores de variáveis não surtirá efeito sobre o *loop.*

Nota: Se o *incremento* for negativo, ocorrerá na realidade um decremento do valor da *variável,* e o *loop* será executado normalmente, até que a *variável* fique menor que o *valor final.*

A utilização de STEP é opcional; sua omissão implica em um *incremento* unitário (igual a um).

Os *loops* FOR/NEXT podem ser entrelaçados. Por exemplo:

```
10 FOR I=1 TO 3
20 PRINT "LOOP EXTERNO"
30 FOR J=1 TO 2
40 PRINT,"LOOP INTERNO"
50 NEXT J
60 NEXT I
```

Se você executar o programa acima, obterá a seguinte resposta:

```
LOOP EXTERNO
     LOOP INTERNO
     LOOP INTERNO
LOOP EXTERNO
     LOOP INTERNO
     LOOP INTERNO
LOOP EXTERNO
     LOOP INTERNO
     LOOP INTERNO
```

Observe que cada instrução NEXT especifica uma variável apropriada, no entanto, esta é somente uma conveniência do programador para auxiliar e manter a seqüência da ordem inserida. A variável contadora pode ser omitida das instruções NEXT. Mas se você usar as variáveis contadoras ordene-as apropriadamente, isto é: a variável para o *loop* mais interno deve vir primeiro.

Outra opção com instruções NEXT "entrelaçadas" é o uso de uma só linha de variáveis.

Elimine a linha 50 do programa acima e modifique a linha 60 para: `60 NEXT J,I`.

Os *loops* podem atingir vários níveis de profundidade. O único limite é a quantidade de memória disponível.

ERROR

Esta instrução possibilita a simulação de um determinado erro durante a execução do programa. Sua principal utilização é no teste de uma rotina ON ERROR GOTO que será vista a seguir.

Consulte o apêndice B na listagem de código de erro e os significados que possuem. Por exemplo:

```
100 ERROR 1
```

Se você executar esta instrução, obterá a seguinte resposta:

```
? NF Erro na 100
```

Este código é correspondente ao erro de tentativa de executar instruções NEXT sem FOR compatibilizante.

ON ERROR GOTO *número de linha*

Normalmente quando é encontrado qualquer tipo de erro em seu programa, a execução é normalmente interrompida e é impressa uma mensagem especificando qual o tipo de erro ocorrido. Utilizando a instrução ON ERROR GOTO, você poderá utilizar uma rotina de detecção de erros, que possibilita a execução de seu programa, sem nenhuma interrupção.

Você deverá ter um tipo de erro específico a detectar ao utilizar esta instrução. Por exemplo:

Suponha que seu programa execute algumas operações de divisão e não foi prevista a possibilidade de divisão por zero. Você pode no entanto preparar esta rotina que manipule tal erro. A instrução ON ERROR GOTO deve ser usada para esta rotina se ocorrer este erro.

RESUME *número de linha*

Esta instrução finaliza uma rotina de manipulação de erros, especificando para onde a execução deve retornar. A instrução RESUME sem número de linha, ou RESUME 0 faz a execução voltar para a linha onde o erro ocorreu. Esta instrução seguida por um número de linha faz com que ocorra um desvio para um determinado número de linha.

A instrução RESUME seguida de NEXT faz com que ocorra um desvio para a instrução posterior à do ponto onde ocorreu o erro. Por exemplo:

```
 10 ON ERROR GOTO 100
 20 A=1/0
 90 END
100 PRINT "ERRO 3"; ERR/2 + 1
110 RESUME 90
```

IF *exp* THEN *ação* ELSE *alternativa*

Esta instrução verifica se uma expressão é verdadeira ou falsa. Se a expressão (*exp*) for verdadeira, o controle passará automaticamente para a instrução de *ação*. Se a instrução for falsa, o controle passará para a instrução *alternativa* ou, na falta desta, para a próxima linha do programa. Por exemplo:

```
100 IF X>127 THEN PRINT "FORA DA FAIXA": END
```

Se X for maior que 127 o controle passará para instrução PRINT e depois para a instrução END, caso contrário passará para a próxima linha do programa, omitindo as instruções PRINT e END.

Nota: A instrução ON ERROR GOTO deve ser executada antes do aparecimento do erro; ou então não surtirá efeito.
Esta instrução pode ser desativada executando-se ON ERROR GOTO 0 na sua rotina de detecção de erros.

Nota: A palavra THEN é opcional, exceto quando se requer a eliminação de uma ambigüidade, como em IF A < 0 100.

O exemplo a seguir utiliza a palavra ELSE introduzindo na linha uma instrução alternativa, para o caso da expressão ser falsa:

```
100 INPUT A$:IF A$="SIM" THEN 300 ELSE END
```

Nesta linha, se A$ for igual a "SIM" então a execução será desviada para a linha 300. Caso contrário será executada a instrução contida na instrução ELSE que, neste caso, finaliza a execução. Por exemplo:

```
200 IF A<B THEN PRINT "A<B" ELSE PRINT "B<A".
```

Se A for menor que B será impresso A < B e a execução continua na próxima linha do programa. Se A não for menor que B, será impressa a mensagem "B < A". Outro exemplo:

```
200 IF A >.001 THEN B=1/A:A=A/5: ELSE 260
```

Se A for maior que 0.001, serão executadas as duas instruções seguintes atribuindo novos valores a A e B. Então a execução prosseguirá na próxima linha de instrução sendo ignorada a instrução ELSE. Mas se A < = .001, será executada a instrução ELSE que desvia para a linha 260.

Nota: A instrução GOTO não é necessária após o ELSE, e as instruções IF/THEN/ELSE podem ser entrelaçadas desde que sejam compatibilizados os IFs e ELSEs. Por exemplo:

```
810 INPUT "ENTRE COM 2 NÚMEROS"; A, B
820 IF A<=B THEN IF A<B PRINT A;: ELSE PRINT "NENHUM ";:
ELSE PRINT B
830 PRINT "E MENOR"
840 END
```

Execute este programa introduzindo vários pares de números. Este programa escolhe e imprime o menor dos dois números introduzidos.

Capítulo 23

Strings

"Sem a capacidade de manipular strings,
o computador se torna apenas uma calculadora superpotente".

Apesar do exagero desta afirmação, ela não está totalmente incorreta. Quanto mais se usa a capacidade *string* do Basic do CP 500, mais verdadeira esta afirmativa será. No Basic do CP 500 qualquer nome de variável válido, poderá ser usado para conter valores *string* usando-se a instrução DEFSTR ou acrescentando a ele um caractere de declaração de tipo ($). Além disso, cada *string* pode conter até 255 caracteres. Pode-se comparar strings para alterá-las alfabeticamente, ou então separá-las e uni-las novamente.

Espaço String

Cinqüenta bytes de memória são reservados automaticamente para armazenar *strings*. Se este espaço reservado acabar, um erro do tipo OS ERRO ocorrerá, podendo ser corrigido utilizando-se o comando CLEARn.

Para o cálculo do espaço necessário, multiplique o espaço que cada variável ocupa pelo número de variáveis *string* utilizadas, inclusive em variáveis temporárias, as quais são criadas durante o cálculo de funções *string*. Portanto, mesmo que você tenha somente algumas variáveis *string* curtas determinadas em seu programa, ainda assim pode ocorrer falta de espaço caso você as una, concatenando-as várias vezes.

Este capítulo abrange:

ASC	LEN	STRING$
CHR$	LEFT$	STR$
FRE	MID$	VAL
INKEY$	RIGHT$	TIME$

ASC (*string*)

Fornece o código ASCII para o primeiro caractere da *string* especificada. O argumento *string* deve ser colocado entre parênteses. Um argumento de *string* nulo causará um erro. Por exemplo:

```
100 PRINT ASC ("A")
110 T$ = "AB" : PRINT ASC (T$)
```

As linhas 100 e 110 imprimirão o mesmo número. O argumento pode ser uma expressão que envolva funções e operadores *strings*.

CHR$ (*expressão*)

Realiza o inverso da função ASC, isto é, fornece o caractere cujo código é especificado pelo valor da *expressão*. Caractere este, que tenha um código gráfico, de controle ou ASCII. O argumento pode ser qualquer número de 0 a 255, ou qualquer expressão variável com um valor nesta faixa, e deve estar entre parênteses.

Ao se utilizar a instrução CHR$, pode-se inclusive colocar aspas dentro de *strings*. Por exemplo:

```
410 A$=CHR$(34)
420 PRINT "ELE DISSE, ";A$;"EU PROMETO...";A$
```

A função CHR$ pode também ser usada para mostrar na tela qualquer caractere gráfico ou especial. Por exemplo:

```
460 CLS
470 FOR I = 129 TO 191
480 PRINT I ; CHR$ (I),
490 NEXT
500 END
```

Execute o programa para observar os vários caracteres gráficos.

Os códigos de controle de vídeo vão de 0 a 31. Quando se imprime um caractere de controle, a função é executada. Por exemplo: 23 é o código de controle do formato de 32 caracteres por linha, sendo assim, o comando PRINT CHR$ (23) converte o formato de vídeo em 32 caracteres por linha. Para retornar ao de 64 caracteres/linha, pressione CLEAR, execute CLS ou PRINT CHR$ (28).

FRE (*string*)

Esta função, quando utilizando uma variável ou constante *string* como argumento, calcula o espaço de armazenamento *string* disponível no momento. O argumento deve estar entre parênteses.

Internamente, faz com que o BASIC comece a procurar, na memória, um espaço *string* não utilizado. Se o seu programa tem efetuado muitos processamentos com *string*, você levará algum tempo para a recuperação da memória do tipo "rascunho".

FRE (*número*)

Esta função calcula o espaço total de memória disponível (o mesmo que MEM).

INKEY$

Esta função tem por objetivo fornecer uma *string* de um caractere determinado pela última tecla pressionada, antes de uma leitura do teclado. Se nenhuma tecla for pressionada, retorna uma *string* nula, isto é, de comprimento igual a zero. Esta é uma função muito importante, pois possibilita a introdução de valores enquanto o computador se en-

---

Nota: Os códigos ASCII estão no apêndice C.

contra em execução, sem a utilização da tecla ENTER. Os caracteres digitados para uma função INKEY$ não são visualizados na tela. A função INKEY$ é geralmente colocada dentro de um *loop*, sendo assim, o teclado é varrido repetidas vezes. Por exemplo:

```
540 CLS
550 PRINT @ 540, INKEY$ : GOTO 550
```

Execute o programa. Observe que a tela permanece vazia, enquanto uma tecla não for pressionada pela primeira vez, a partir daí a última impressão permanece na tela até que se pressione outra tecla, pois um novo valor sempre prevalecerá sobre o anterior.

Esta função pode ser usada em seqüências de *loop*, para permitir ao usuário a elaboração de uma *string* maior. Por exemplo:

```
590 PRINT "DIGITE 3 CARACTERES"
600 A$=INKEY$:IF A$=""THEN 600 ELSE PRINT A$;
6L0 B$=INKEY$:IF B$=""THEN 610 ELSE PRINT B$;
620 C$=INKEY$:IF C$=""THEN 620 ELSE PRINT C$;
630 D$=A$+B$+C$
```

Assim, uma *string* de três caracteres D$ pode ser introduzida via teclado, sem utilização da tecla ENTER.

LEN (*string*)

Esta função fornece o comprimento de caracteres de uma determinada *string*. A constante, expressão ou variável *string* deve estar entre parênteses. Por exemplo:

```
730 A$ = ""
740 B$ = "BANG"
750 PRINT A$, B$, B$ + B$
760 PRINT LEN (A$), LEN (B$), LEN (B$ + B$)
```

LEFT$ (*string,n*)

Esta função fornece os primeiros *n* caracteres da *string*. A *string* pode ser uma constante ou expressão e *n* uma expressão numérica, e estes argumentos devem estar entre parênteses. Por exemplo:

```
10 A$="GILBERTO"
20 B$=LEFT$(A$,3)
30 PRINT B$;" E' O APELIDO DE ";A$
```

MID$ (*string,p,c*)

Esta função fornece uma *sub-string*, cujo comprimento é *c* e posição inicial *p*. O nome da *string*, o comprimento e a posição inicial são os argumentos, e estes devem estar entre parênteses.

A *string* mencionada pode ser uma constante ou expressão *string*, *c* e *p* podem ser expressões ou constantes numéricas. Se não for determinado o argumento, a *string* que

---

Nota: A instrução IF A$ = "" compara A$ com uma *string* nula. Não há espaços entre as aspas.

inicia na posição *p* será selecionada. Por exemplo:

```
MID$ (L$, 3, 1)
```

Este exemplo refere-se a uma *string* de um caractere que começa no terceiro caractere da *string* original.

Um outro exemplo seria examinar o nome de uma pessoa, que está armazenado em uma variável *string*. Por exemplo:

```
110 A$="HUMBERTO NUNES GARCIA"
120 PRINT MID$(A$,10,5)
```

RIGHT$ (*string,n*)

Esta função fornece os últimos *n* caracteres da *string*. A *string* pode ser uma constante ou expressão e *n* uma expressão numérica, e estes argumentos devem estar entre parênteses. Se o comprimento da *string* for menor ou igual a *n*, então toda a *string* será selecionada. Por exemplo:

```
10 PRINT "DIGITE UMA PALAVRA" ; M$
20 IF LEN (M$) = 0 THEN 10
30 PRINT " A ULTIMA LETRA FOI: " ; RIGHT$ (M$, 1)
40 GOTO 10
```

STRING$ (*n, "caractere"* ou *número*)

Esta função fornece uma *string* composta de *n* caracteres. Tanto o argumento *n* como o argumento "caractere" podem ser um número de 0 a 255, contudo o argumento "caractere" será manipulado como um código gráfico, de controle ou ASCII. Os dois programas a seguir usam essa função para provocar efeitos especiais na tela.

```
170 CLEAR 200
180 FOR I=32 TO 191
190 A$=STRING$(64,I)              'A$ CONTEM 64 CARACTERES
200 PRINT A$;:PRINT:PRINT:        'IMPRIME A$:SALTA 2 LINHAS
210 FOR J=1 TO 200
220 NEXT J
230 NEXT I
```

```
10 CLEAR 100
20 A$=STRING$(10,10)
30 PRINT A$;"OK"
40 GOTO 10
50 'SALTA 10 LINHAS NA
60 'TELA E IMPRIME "OK"
```

STR$ (*expressão*)

Esta função converte uma expressão ou constante numérica em *string*. A expressão ou constante numérica deve estar entre parênteses. Por exemplo:

Ver programa da página a seguir.

```
860 A=58.5:B=-58.5
870 PRINT STR$(A)
880 PRINT STR$(B)
890 PRINT STR$(A+B)
900 PRINT STR$(A)+STR$(B)
```

No exemplo acima, STR$ (A) será igual à *string* " 58.5" (Observe o espaço dianteiro). Enquanto as operações aritméticas podem ser realizadas com A, somente as funções ou operações *string* podem ser realizadas na *string* " 58.5".

A instrução PRINT STR$ (A) imprime A com o espaço dianteiro.

VAL (*string*)

Esta função realiza o inverso da função STR$, fornecendo assim, o número representado pelos caracteres em um argumento *string*. O tipo numérico do resultado pode ser inteiro, precisão simples ou precisão dupla. Por exemplo:

```
10 A$="12"
20 B$="34"
30 C=VAL(A$+"."+B$)
40 PRINT "C=";C
50 D=VAL(A$+"E"+B$)
60 PRINT "D=";D
```

Neste exemplo, obteremos C = 12.34 e D = 12E34. VAL opera de maneira um pouco diferente em *strings* mistas, cujos valores consistem de um número seguido por caracteres não numéricos. Em tais casos, somente o número dianteiro será utilizado para determinação do VAL e o restante não numérico será ignorado. Por exemplo:

```
20 X$ = "100 CRUZEIROS"
30 PRINT VAL (X$)
```

Neste exemplo, teremos como resultado o número 100.

TIME$

Esta função fornece a data atual e a hora. O CP 500 tem um relógio e para usá-lo, deve-se ajustá-lo na data e hora corretas. Para isso, pode-se digitar e executar este programa:

```
10 DEFINT A-Z
20 DIM TM(5)
30 CL=16924
40 PRINT "DIGITE 6 VALORES : MM, DD, AA, HR, MN, SS"
50 INPUT TM(0),TM(1),TM(2),TM(4),TM(5)
60 FOR I=0 TO 5
70 POKE CL-I,TM(I)
80 NEXT I
90 PRINT "RELOGIO AJUSTADO"
100 END
```

Após o ajuste, você poderá utilizar o relógio a qualquer momento. Por exemplo:

```
110 PRINT TIME$
```

A instrução acima, faz com que seja impresso no vídeo a data e a hora. Se não houver um ajuste adequado, isto não impedirá o relógio de funcionar, no entanto, a data e a hora serão ajustadas em zero, quando você ligar o computador ou reajustá-lo.

Nota: O relógio estará desligado durante operações com cassete, portanto precisará ser corrigido periodicamente.

Capítulo 24

Matrizes

Uma matriz é uma lista ordenada de valores. Os valores podem ser tanto numéricos, quanto strings, o que os diferencia é a definição dada à matriz.

As matrizes proporcionam um método fácil e organizado de manipular grandes quantidades de dados.

Como ilustração utilizaremos uma matriz para armazenar informações referentes a talões de cheque, números, datas e quantias de cada cheque.

Considere a seguinte tabela de informações sobre relações de cheques:

CHEQUE #	*DATA*	*QUANTIA*
025	01.01.81	100,00
026	01.11.81	50,00
027	02.02.81	328,55
028	23.07.81	10.500,00
029	15.10.81	500,00
030	28.12.81	800,00

Observe que cada item da tabela pode ser especificado por duas coordenadas: uma de linha e outra de coluna. Por exemplo:

O elemento comum à linha 3 e à coluna 3 da tabela acima, corresponde à quantia de 328,55. Sendo assim, dá-se o nome de endereço ao par (3,3) correspondente ao valor 328,55 da tabela.

Elaboramos para ilustração, uma matriz chamada CK para identificar a tabela de informação sobre a relação de cheques. Como a tabela contém seis linhas e três colunas, a matriz CK necessitará de dois índices: um para o número de coluna e outro para o de linha.

Logo, exemplificando melhor temos:

```
CK(1,1)=025    CK(1,2)=01.01.81    CK(1,3)=100.00
CK(6,1)=030    CK(6,2)=28.12.81    CK(6,3)=800.00
```

Nota: Nestes capítulos serão omitidos os elementos de índice zero da matriz, mas eles estão disponíveis e deverão ser utilizados para maior eficiência do uso da memória. Por exemplo:

Após a instrução DIM A (4), a matriz A conterá cinco elementos que são: A(0), A(1), A(2), A(3), A(4). No caso da não utilização da instrução DIM para dimensionar matrizes, fica determinado 11 elementos subscritos para cada dimensão.

Observe que as datas são gravadas na forma: DD.MM.AA, onde DD = número correspondente ao dia, MM = número correspondente ao mês, AA = número correspondente aos dois últimos dígitos do ano.

Utilizaremos agora um programa exemplo para leitura de valores na matriz CK.

```
50 FOR LIN=1 TO 6
60 FOR COL=1 TO 3
70 READ CK(LIN, COL)
80 NEXT COL,LIN
90 DATA  025, 01.0181,     100.00
100 DATA 026, 01.1181,      50.00
110 DATA 027, 02.0281,     328.55
120 DATA 028, 07.2381, 10.500.00
130 DATA 029, 10.1581,     500.00
140 DATA 030, 12.2881,     800.00
```

Suponha agora que se queira somar todos os cheques registrados. Para isso deverão ser acrescentadas as seguintes linhas ao programa acima:

```
150 FOR LIN=1 TO 6
160 SOM=SOM+CK(LIN, 3)
170 NEXT
180 PRINT "VALOR TOTAL DOS CHEQUES EMITIDOS";
190 PRINT USING "$$#######.##" ; SOM
```

Suponha agora que se queira imprimir todos os cheques que foram registrados num determinado dia. Para isto deverão ser acrescentadas as seguintes linhas ao programa acima:

```
200 PRINT "CHEQUES EMITIDOS NA DATA (DD.MMAA)";
210 INPUT DT
220 PRINT:PRINT "O(s) CHEQUE(s) ESTA(o) A SEGUIR:"
230 PRINT "CHEQUE #","VALOR":PRINT
240 FOR LIN=1 TO 6
250 IF CK(LIN,2)=DT THEN PRINT CK(LIN,1),CK(LIN,3)
260 NEXT
```

Tipos de Matrizes

No CP 500, o número de dimensões que uma matriz pode ter é limitado apenas pela quantidade de memória disponível. Matrizes *string* também podem ser utilizadas. Por exemplo:

Uma matriz definida como C$ seria imediatamente interpretada como uma matriz string. Outra forma seria utilizar a instrução DEFSTR C. Nesse caso qualquer matriz cujo nome inicie com a letra C será interpretada como sendo uma matriz string.

Uma aplicação interessante para uma matriz *string* seria o armazenamento de tex-

---

Nota: Como CK é uma matriz numérica, não é permitido o armazenamento de caracteres alfa-numéricos como elementos desta matriz, tais como travessões.

tos. Por exemplo:

```
10 CLEAR 1200
20 DIM TXT$(10)
```

As instruções acima formam uma matriz *string* capaz de armazenar 10 linhas de texto. Para isto são reservados 1200 *bytes* para possibilitar a introdução de 10 linhas de 60 caracteres, além de 600 *bytes* extras para manipulação de *string* com outras variáveis *string*.

Sub-rotinas de Manipulação de Matrizes

Para ser utilizada a sub-rotina que será descrita abaixo, o seu programa deverá fornecer os valores para duas variáveis N1 e N2, onde N1 representa o número de linhas e N2 o número de colunas. E, ainda, dimensionar a matriz de acordo com N1 e N2.

Esta sub-rotina exemplifica como atribuir valores a elementos numa matriz, linha por linha, respondendo à instrução INPUT.

```
10 FOR LIN=1 TO N1
20 FOR COL=1 TO N2
30 PRINT "ENTRE COM OS DADOS PARA (";LIN;":";COL;")";
40 INPUT A(LIN, COL)
50 NEXT COL
60 NEXT LIN
70 RETURN
```

Para ser utilizada a sub-rotina que será descrita abaixo, o seu programa deverá fornecer os valores para três variáveis que são respectivamente N1, N2 e N3, além de dimensionar a matriz.

Dentro desta sub-rotina, você pode atribuir valores a cada elemento da matriz utilizando instruções READ e DATA*.

```
400 REM UTILIZA "READ" E "DATA"
410 FOR K=1 TO N3
420 FOR I=1 TO N1
430 FOR J=1 TO N2
440 READ A(I,J,K)
450 NEXT J,I,K
460 RETURN
```

A sub-rotina descrita abaixo imprime a matriz da sub-rotina anterior.

```
560 FOR K=1 TO N3
570 FOR I=1 TO N1
580 FOR J=1 TO N2
590 PRINT A(I,J,K);
600 NEXT J,I,K:PRINT
630 RETURN
```

Nota: Para utilizar esta sub-rotina, você deverá introduzir em seu programa a instrução DATA para atribuir os valores aos elementos da matriz.

A sub-rotina descrita abaixo atribui a cada elemento da matriz um valor correspondente, utilizando para isto a instrução INPUT.

```
660 FOR K=1 TO N3
670 PRINT "PAGINA";K
680 FOR I=1 TO N1
690 PRINT "ENTRE COM A LINHA";I
700 FOR J=1 TO N2
710 INPUT A(I,J,K)
720 NEXT J
730 NEXT I
740 PRINT;NEXT K
750 RETURN
```

A sub-rotina abaixo faz a multiplicação de todos os elementos da matriz por X, onde X deve ter seu valor atribuído no programa.

```
780 FOR K=1 TO N3
790 FOR J=1 TO N2
800 FOR I=1 TO N1
810 B (I,J,K)=A(I,J,K)*X
820 NEXT I
830 NEXT J
840 NEXT K
850 RETURN
```

A sub-rotina abaixo faz a transposição de uma matriz de duas dimensões.

```
880 FOR I=1 TO N1
890 FOR J=1 TO N2
900 B (J,I)=A(I,J)
910 NEXT J
920 NEXT I
930 RETURN
```

A sub-rotina abaixo faz a adição de duas matrizes de três dimensões.

```
960 FOR K=1 TO N3
970 FOR J=1 TO N2
980 FOR I=1 TO N1
990 C (I,J,K)=A(I,J,K)+B(I,J,K)
1000 NEXT I
1010 NEXT J
1020 NEXT K
1030 RETURN
```

A sub-rotina abaixo faz a multiplicação de duas matrizes de três dimensões termo a termo.

```
1060 FOR K=1 TO N3
1070 FOR J=1 TO N2
1080 FOR I=1 TO N1
1090 C(I,J,K)=A(I,J,K)*B(I,J,K)
1100 NEXT I
1110 NEXT J
1120 NEXT K
1130 RETURN
```

Capítulo 25

Funções Matemáticas

O CP 500 oferece uma ampla variedade de funções intrínsecas, para a realização de funções matemáticas e especiais. As funções de operações especiais são descritas no próximo capítulo. Todas as funções matemáticas comuns, descritas neste capítulo, fornecem valores de precisão simples com seis casas decimais de precisão.

As funções ABS, FIX e INT fornecem valores cuja precisão depende de uma função especial. As funções trigonométricas utilizam ou fornecem o resultado em radianos e não em graus. Uma conversão radiano-graus é dada para cada uma das funções. Para todas as funções, o argumento deve permanecer entre parênteses.

Funções descritas neste capítulo:

CDBL	LOG	SIN	SQR
CINT	INT	COS	RND
CSNG	FIX	TAN	RANDOM
EXP	ABS	ATN	SGN

CDBL (*x*)

Esta função fornece uma representação de precisão dupla do argumento. O valor resultante contará com 17 dígitos, mas somente aqueles contidos no argumento serão significativos.

CDBL pode ser útil, quando se quer forçar uma operação a ser efetuada em precisão dupla, mesmo que os operandos sejam de precisão simples ou inteiros. Por exemplo:

```
100 FOR I% = 1 TO 25 : PRINT 1/CDBL (I%), :NEXT
```

CINT (*x*)

Esta função fornece o maior inteiro menor que o argumento. Por exemplo:

`CINT (1.5)` resulta em 1
`CINT (-1.5)` resulta em −2.

Para a função CINT o argumento deve estar na faixa de −32768 a +32767 e o resultado é armazenado como um inteiro, ocupando assim 2 bytes.

CINT pode ser utilizada para acelerar a operação envolvendo operandos de precisão simples e dupla, sem perder a exatidão dos mesmos. Por exemplo:

```
90 INPUT X#,Y#
100 K%=CINT(X#)+CINT(Y#)
110 PRINT K%
```

Note que os valores em precisão dupla, assim como o resultado de sua soma, devem estar dentro da faixa dos inteiros.

CSNG (*x*)

Esta função fornece uma representação de precisão simples do argumento. Quando o argumento for um valor de precisão dupla, esta função fornecerá um valor com seis dígitos significativos e um arredondamento de 4/5 no dígito menos significativo. Por exemplo:

```
CSNG(0.6666666666666667)
```
resulta em 0.666667.

```
CSNG(0.3333333333333333)
```
resulta em 0.333333

EXP (*x*)

Esta função fornece o exponencial natural de *x*, que é e . Por exemplo:

```
50 INPUT X
60 PRINT EXP(-X)
```

EXP é a função inversa da logarítmica, portanto, X = EXP(LOG(X)).

LOG (*x*)

Esta função fornece o logaritmo natural do argumento, isto é, log (argumento). Esta é a função inversa da exponencial, sendo assim, X=LOG(EXP(X)).

Para mudança de base do logaritmo utilize a seguinte fórmula.

$$\mathrm{Log}_b x = \mathrm{Log}_e x \,/\, \mathrm{Log}_e b$$

Por exemplo:

```
LOG(32767)/LOG(2)
```
resulta em Log_2 32767

INT (*x*)

Esta função fornece uma representação inteira do argumento, utilizando o maior número inteiro menor que o argumento. O argumento não é limitado à faixa de −32768 a +32767. O resultado é armazenado internamente como um número de precisão simples, sem casas decimais. Por exemplo:

```
90 INPUT A
100 Z=INT(A*100+0.5)/100
110 PRINT Z
```

FIX (*x*)

Essa função fornece uma representação truncada do argumento. Todos os dígitos à direita do ponto decimal são simplesmente cortados e o valor resultante será um inteiro. Para um valor X não negativo, FIX(X) = INT(X), e para um valor X negativo, FIX(X) = INT (X)+1. Por exemplo:

```
FIX(2.2)
```
resulta em 2

```
FIX(-2.2)
```
resulta em −2

ABS (*x*)

Esta função fornece o valor absoluto do argumento.

ABS (X)=X , para todo X maior ou igual a zero e,
ABS (X)=-X , para todo X menor que zero.

Por exemplo:

```
70 INPUT X
80 IF ABS(X)<2 THEN PRINT "MUITO PEQUENO"
```

SIN (*x*)

Esta função fornece o seno do argumento, sendo que este deve estar em radianos. Para obter-se o seno de X em graus, use SIN(X*0.01745329). Por exemplo:

```
200 INPUT A,B
210 PRINT SIN (A*B-B)
```

COS (*x*)

Esta função tem por finalidade fornecer o cosseno do argumento, sendo que este deve estar em radianos. Para obter o cosseno se X em graus, use COS (X * 0.01745329). Por exemplo:

```
150 INPUT X
160 Y = COS (X+3.3)
170 PRINT Y
```

TAN (*x*)

Esta função fornece a tangente do argumento e, este deve estar em radianos. Para obter-se a tangente em graus, usa-se TAN (X*0.01745329). Por exemplo:

```
310 INPUT A
320 Z = TAN (2*A)
330 PRINT Z
```

ATN (*x*)

Esta função fornece o arco-tangente em radianos, isto é, o ângulo cuja tangente é X. Para obter o resultado em graus, basta multiplicar ATN(X) por 57.29578. Por exemplo:

```
30 INPUT B,C
40 Y = ATN (B/C)
50 PRINT Y
```

SQR (*x*)

Esta função fornece a raiz quadrada do argumento. SQR (X) é o mesmo que X [(1/2), porém o resultado é fornecido mais rapidamente. Por exemplo:

```
510 INPUT X,H
520 Y = SQR (X[2-H[2)
530 PRINT Y
```

RND (*x*)

Esta função gera um número pseudo-aleatório, utilizando um número base corrente, também pseudo-aleatório, gerado internamente e inacessível ao usuário. RND pode ser utilizada para a produção de números aleatórios entre 0 e 1, ou inteiros aleatórios maiores que 0, dependendo do argumento.

RND (0) resulta em um valor de precisão simples entre 0 e 1.

RND (*inteiro*) resulta num inteiro entre 1 e o próprio argumento, inclusive devendo este argumento ser positivo e menor que 32768. Por exemplo:

`RND(55)` fornece um inteiro pseudo-aleatório maior que zero e menor que 56.

`RND(55.55)` fornece um número na mesma faixa, pois RND utiliza o valor inteiro do argumento.

```
100 X=RND(2)
110 PRINT X
120 GOTO 100
```

RANDOM

RANDOM na realidade é uma instrução completa e não uma simples função. Ela ativa um gerador de números aleatórios interno. De preferência, coloque a instrução RANDOM no início do programa; pois isto garantirá a obtenção de uma seqüência imprevisível de números pseudo-aleatórios, sempre que se ligar o computador, carregar este programa e executá-lo.

```
10 RANDOM
20 PRINT RND(0)
30 GOTO 20
```

SGN (*x*)

A função "signal", ou "signum", fornece −1 para X negativo, 0 para X nulo e +1 para X positivo. Por exemplo:

```
320 INPUT X
330 ON SGN(X)+2 GOTO 400,500,600
400 PRINT "NUMERO NEGATIVO":GOTO 320
500 PRINT "ZERO":GOTO 320
600 PRINT "NUMERO POSITIVO":GOTO 320
```

A sub-rotina abaixo faz a multiplicação de duas matrizes de duas dimensões.

```
1160 FOR I=1 TO N1
1170 FOR J=1 TO N2
1180 C(I,J)= 0
1190 FOR K= 1 TO N3
1200 C(I,J)=C(I,J)+A(I,K)*B(K,J)
1210 NEXT K,J,I
1220 RETURN
```

Nota: Esta sub-rotina deve ser utilizada por um programa que forneça os valores para os elementos das matrizes.

Capítulo 26

Características Especiais

Existem algumas funções e operações que são altamente específicas e tornam-se mais significativas, quando se adquire maior experiência em programação e em rotinas em linguagem de máquina.

Mas existem outras funções que são menos específicas e que serão utilizadas freqüentemente.

Estão relacionadas abaixo todas as funções e instruções descritas neste capítulo.

Funções Gráficas:

SET
RESET
CLS
POINT

Funções de Rotina de Erro:

ERL
ERR

Outras Funções e Instruções:

INP
MEM
OUT
PEEK
POKE
POS
USR
VARPTR

SET (x,y)

A função SET ativa um ponto gráfico, na posição determinada pelas coordenadas x e y.

A tela é dividida em uma grade de 128 (horizontal) por 48 (vertical). As abcissas são numeradas da esquerda para a direita de 0 a 127, e as ordenadas de cima para baixo de 0 a 47, portanto o ponto (0,0) se encontra no extremo superior esquerdo da tela, enquanto que o (127,47) no extremo oposto.

Veja no apêndice o *Lay-out* do vídeo. Os argumentos x e y podem ser constantes,

variáveis ou expressões numéricas, e não é necessário que sejam valores inteiros, visto que SET (x, y) utiliza a parte inteira de x e y. Esta função é válida para todo x e y, no intervalo dado abaixo.

$0<=x<128$ e $0<=y<48$

Por exemplo:

```
100 SET(RND(128) -1, RND(48)- 1) 'ATIVA UM PONTO NA TELA
```

RESET (x,y)

Esta função desativa o ponto gráfico especificado pelas coordenadas x e y. Esta função tem os mesmos limites e parâmetros da função SET (x,y). Por exemplo:

```
90 INPUT X
100 SET (X,3)
110 FOR I=1 TO 200:NEXT I
120 RESET (X,3)
```

CLS

Esta função "limpa" a tela, ou seja, desativa todos os pontos gráficos da tela e move o cursor para o canto superior esquerdo.

POINT (x,y)

Essa função verifica se um ponto está ativado ou não. Se o ponto estiver ativado é porque foi utilizada a função SET e, a função POINT fornecerá o código binário -1, caso contrário esta função fornecerá o código binário 0. Por exemplo:

```
100 SET (50,28)
110 IF POINT(50,28) THEN PRINT "ON" ELSE PRINT "OFF"
```

Se este programa for executado será sempre impressa a mensagem "ON", pois a função SET foi ativada na linha 100.

ERL

Esta função fornece o número de linha onde ocorreu um erro, e é usada principalmente dentro de uma rotina de manipulação de erros, na qual existe a instrução ON ERROR GOTO.

Se ao ativar a função ERL nenhum erro foi encontrado será fornecida como resposta a linha 0. Mas se for cometido um erro a resposta será a linha em que o erro ocorreu. Por exemplo:

```
10 CLEAR 10
20 ON ERROR GOTO 1000
30 INPUT "DIGITE A SUA MENSAGEM";M$
40 INPUT "AGORA, DIGITE UM NUMERO";N%
```

continua

```
50 Z%=1/N%
60 PRINT "DIGITE VALORES E TENTE NOVAMENTE CAUSAR UM ERRO"
70 GOTO 30
1000 IF ERL=30 AND (ERR/2+1=14) THEN 1040
1010 IF ERL=40 AND (ERR/2+1=6) THEN 1050
1020 IF ERL=50 AND (ERR/2+1=11) THEN 1060
1030 ON ERROR GOTO 0 : RESUME
1040 PRINT"MENSAGEM MUITO LONGA":FOR I=1 TO 200:RESUME 10
1050 PRINT"NÚMERO MUITO GRANDE":FOR I=1 TO 200:RESUME
1060 PRINT"DIVISÃO POR ZERO - ENTRE COM OUTRO NÚMERO"
1070 PRINT"DIVISÃO POR ZERO - ENTRE COM OUTRO NÚMERO"
1070 RESUME 40
```

Se você executar este programa e tentar introduzir uma mensagem longa, e digitar zero quando for pedido um número, observará que ERL é usada na linha 1000 para determinar onde ocorreu o erro.

ERR/2+1

Esta função é similar a ERL, fornece o valor relativo ao código de erro, ao invés da linha onde o erro ocorreu. Ela é comumente, usada em uma rotina de manipulação de erros, na qual exista a instrução ON ERROR GOTO.

Você poderá conhecer todos os códigos de erros consultando o apêndice B deste manual.

INP (*porta*)

Esta função fornece o valor do *byte* introduzido pela porta especificada. Existem 256 portas numeradas de 0 a 255. Por exemplo:

```
100 PRINT INP (50)
```

A instrução acima introduz um *byte* pela porta 50 e imprime seu valor decimal. Não há necessidade de acesso pelas portas Z 80 para otimizar o uso do CP 500.

MEM

Esta função fornece o número de *bytes* em desuso e desprotegidas na memória.

Esta função pode ser usada no modo imediato para verificar qual o espaço ocupado por um programa residente em memória.

Ela pode também ser usada em programas para se evitar erros do tipo OM, através da colocação de menos espaços *string*, ou do dimensionamento de matrizes menores etc. Por exemplo:

```
100 IF MEM<80 THEN 900
110 PRINT"MEMORIA SUFICIENTE":STOP
900 PRINT"MEMORIA INSUFICIENTE"
```

Nota: Para descobrir a quantidade de memória não utilizada para armazenagem de programas, variáveis, *strings, stack,* ou sendo reservada para arquivos-objeto deverá ser utilizado o comando PRINT MEM no modo direto.

OUT *porta, valor*

Esta função envia o valor do *byte* a uma porta especificada.

Para a execução desta função são necessários dois argumentos separados por vírgula e sem parênteses. Estes argumentos são o destino e o valor do *byte* a ser enviado respectivamente. Tanto a porta quanto o valor estão na faixa de 0 a 255.

PEEK *(endereço)*

O endereço deve estar na forma decimal e esta função fornece o valor armazenado no endereço de *byte* especificado.

Você deverá estar familiarizado com o mapa de memória e com a tabela de códigos gráficos e funções ASCII, para utilização desta função e seria também conveniente a utilização do manual do conjunto de instruções para microprocessadores Z 80, quando esta função for utilizada para examinar arquivos-objeto.

Uma rotina em linguagem de máquina pode armazenar informações numa certa posição de memória. Esta função pode ser usada em seu programa para reaver a informação. Por exemplo:

```
100 A=PEEK(17999)
```

Neste exemplo é atribuido à variável A o valor armazenado na posição 17999.

A função PEEK pode também reaver informações armazenadas pela instrução POKE. A utilização das funções PEEK e POKE possibilita a elaboração de sistemas compactados e de armazenagem orientada por *byte.*

POKE *endereço, valor*

Esta função introduz um valor em uma posição específica da memória. Para isso há necessidade de dois argumentos: um endereço do *byte* na forma decimal, e um valor entre 0 e 255, inclusive.

Se você necessitar de um endereço acima de 32767 para as funções PEEK e POKE utilize a seguinte fórmula:

$$-1 * (65536 - \text{endereço desejado}) = \text{endereço PEEK ou POKE}$$

Por exemplo:

Para a introdução do valor 255 no endereço 32769, use a instrução POKE - 32767, 255.

Esta instrução pode ainda ser utilizada para colocar caracteres diretamente na memória de vídeo (endereços 15360 a 16383). O programa a seguir demonstra essa característica.

```
10 CLS
20 FOR M = 15360 TO 16383
30 POKE M, 191
40 NEXT M
50 GOTO 50
```

Se utilizarmos a função POKE fora da faixa poderemos armazenar *bytes* em posições críticas, como por exemplo em nosso programa ou na memória STACK. O uso indiscriminado desta função acarreta sérias conseqüências, e seremos obrigados a usar RESET, ou mesmo desligar o equipamento e recomeçar.

POS *(x)*

Esta função fornece um número de 0 a 63 indicando a posição corrente do cursor na tela. Por exemplo:

```
100 PRINT TAB(40);POS(0)
```

USR *(x)*

Esta função possibilita a chamada de uma sub-rotina em linguagem de máquina.

As sub-rotinas em linguagem de máquina, são úteis para aplicações especiais, isto é, aquelas que não podem ser executadas através do BASIC, devido a própria velocidade do BASIC.

É necessário um conhecimento da linguagem assembly e do conjunto de instruções Z 80 para se escrever tais sub-rotinas.

COMO CARREGAR UMA ROTINA ASSEMBLY

Reserve uma área no final da memória, onde a rotina será posicionada. Este posicionamento ocorre logo após a ativação do sistema, ao se responder a questão MEM usada?, — juntamente com o endereço anterior ao endereço inicial da sua rotina USR. Por exemplo:

Se a sua rotina inicia em 32700, então digite 32699, em resposta à questão acima.

Se ela for armazenada na fita no formato SYSTEM, você deverá carregá-la através do comando SYSTEM.

Após a gravação da fita pressione a tecla **BREAK** para retornar ao modo imediato do BASIC.

Se a rotina for curta você poderá simplesmente introduzi-la no final da memória, informando onde inicia a rotina USR.

Antes da chamada da USR, deverá ser informado o endereço de acesso à rotina. Para isso introduza o endereço de 2 *bytes* com as posições da memória.

Antes de fazer uma chamada USR, deverá ser informado o endereço de partida da rotina, através do comando POKE. As posições de memória que conterão o endereço da rotina são: 16526 e 16527. Deverá ser colocado na posição 16526 o *byte* de menor valor, que compõe o endereço de chamada da rotina, e na posição 16527 o *byte* mais significativo. Por exemplo:

Ponto de Acesso = *32700*
32700 (decimal) = *7FBC (hexadecimal)*
Byte menos significativo = *BC (hexadecimal)* = *188 (decimal)*
Byte mais significativo = *7F (hexadecimal)* = *127 (decimal)*

Para carregar os endereços de chamada da rotina utilizamos:

```
POKE 16526,188    (byte menos significativo)
POKE 16527,127    (byte mais significativo)
```

Neste momento a rotina já poderá ser executada usando-se apenas a instrução:

```
X = USR (N)
```

onde N pode ser uma expressão e deve ter valores entre −32768 e +32767.

Para a instrução dada ocorrerá o seguinte:

```
(X = USR (N))
```

Será chamada a sub-rotina indicada nos endereços pré-fixados anteriormente (*bytes* 16526 e 16527), sendo que o argumento N da instrução, servirá como parâmetro (se o usuário desejar) para a rotina. A variável X deverá conter, após o retorno de sub-rotina, o valor do par de registradores "HL" (opcionalmente; se o usuário desejar). Por exemplo:

Deverá ser chamada a sub-rotina (através do USR) que se encontra no endereço hexa FF00 da memória e deverá ser passado o parâmetro "2" para esta sub-rotina. A sub-rotina, por sua vez, deve retornar com um valor para o programa principal, valor este que estará armazenado em "X", quando o programa retornar. Exemplificando, teremos:

Programa principal (Basic)

```
10 INPUT "      ";A$
20  .
25  .
30  .
40 POKE 16526,0       'CARREGANDO O
50 POKE 16527,255     'ENDEREÇO DA SUB ROTINA
60 X = USR (2)        'CHAMANDO A SUB ROTINA
70                    'PASSANDO PARAMETRO "2"
80                    'PEDINDO RETORNO EM "X"
90  .
100 .
110 .
```

Sub-rotina *(Assembly)*

```
FF00  CALL DA7FH
FF01  .
FF02  .
FF03  .
FF04  JP DA9AH
```

Nota: Ao ser ativada a rotina *Assembly*, o parâmetro 2 será armazenado no par de registradores "HL". Ao terminar, o conteúdo de "HL" estará disponível em "X" e o processamento retornará ao programa principal.

VARPTR *(nome de variável)*

Fornece um valor-endereço que o auxiliará a localizar onde o *nome de variável* e seu valor estão armazenados na memória. Se não foi determinado um valor à variável que você especificar, um erro FC ocorrerá quando esta função for chamada.

Se VARPTR (variável inteira) fornece o endereço K, este conterá o *byte* menos significativo (LSB) do inteiro de 2 *bytes*.

O endereço K+1 conterá o *byte* mais significativo (MSB) do inteiro. Pode-se mostrar estes *bytes* (representação decimal complementar de 2) na tela, executando-se uma instrução PRINT PEEK (K) e uma PRINT PEEK (K+1).

Se VARPTR (variável de precisão simples) fornece o endereço K, então:

(K)* = LSB (*byte* menos significativo) do valor
(K+1) = próximo *byte* mais significativo (NEXT MSB).
(K+2) = MSB (*byte* mais significativo) com o primeiro *bit* implícito. O *bit* mais significativo é o sinal do número.
(K+3) = o expoente em excesso de 128 (128 é acrescentado ao expoente).

Se VARPTR (variável de dupla precisão) fornece K:

(K) = LSB (*byte* menos significativo) do valor.
(K+1) = NEXT MSB (*byte* mais significativo)
(K+...) = NEXT MSB (*byte* mais significativo)
(K+6) = MSB com o segundo *byte* implícito. O *bit* mais significativo é o sinal do número.
(K+7) = expoente em excesso de 128(128 é acrescentado ao expoente).

Para valores de precisão simples e dupla, o número é armazenado na forma exponencial normalizada. 128 é acrescentado ao expoente. Além disso, o *bit* mais significativo do MSB é usado como sinal. Ele será 0 se o número for positivo ou 1 para números negativos. Veja exemplo no final deste capítulo.

Pode-se mostrar estes *bytes* na tela executando o PRINT PEEK (X) apropriado, onde X é o endereço que você quer que seja mostrado. Lembre-se que o resultado será uma representação decimal de *byte*, com o *bit* 7 (MSB) sendo usado como sinal. O número será na forma exponencial normalizada com o decimal adotado antes de MSB (*byte* mais significativo). 128 é acrescentado ao expoente. Se VARPTR (variável *string*) resultar em K:

(K) = Comprimento do *string*
(K+1) = LSB (*byte* menos significativo) de endereço inicial de valor *string*.
(K+2) = MSB (*byte* mais significativo) de endereço inicial de valor *string*.

O endereço estará provavelmente no final da RAM, onde o espaço de armazenagem *string* foi reservado. Mas, se for uma constante *string*, então será indicada a área da memória, onde a linha do programa com a constante está armazenada.

As instruções do programa como A$ = "ALO!" não utilizam o espaço de armazenagem *string*.

Nota: *(K) significa "CONTEÚDO DO ENDEREÇO K".

Para todas as variáveis acima, os endereços (K-1) e (K-2) armazenarão o código de caracteres do CP 500 para o nome da variável. O endereço (K-3) conterá um código de representação, que diz ao computador qual é o tipo de variável. O inteiro é 02, precisão simples é 04, precisão dupla é 08 e *string* é 03.

VARPTR (variável matriz) retornará o endereço para o primeiro *byte* daquele elemento na matriz. O elemento consistirá de 2 *bytes* se for uma matriz de inteiros, 3 *bytes* se for uma matriz de *strings*, 4 *bytes* se for uma matriz de precisão simples e 8 *bytes* se for de precisão dupla.

O primeiro elemento na matriz é precedido de:

1 - Um código de tipo de 1 *byte* (02 = inteiro, 03 = *string*, 04 = precisão simples, 08 = precisão dupla);

2 - Um par de *bytes* contendo o nome de matriz codificada em ASCII;

3 - Um par de *bytes* indicando o número total de elementos na matriz;

4 - Um *byte* simples indicando o número total de dimensões na matriz;

5 - Uma seqüência de 2 *bytes* por dimensão, cada par indicando a "profundidade" de cada respectiva dimensão.

Os elementos de matriz são armazenados seqüencialmente variando as suas posições na primeira dimensão, depois na segunda e assim por diante. Exemplos:

A! = 2 será armazenado assim:

2 = 10 binário, representado assim: 1E2 = $0{,}1 \times 2^2$. Então o expoente de A é 128 + 2 = 130 (chamado excesso).

MSB (*byte* mais significativo) de A é 10000000, entretanto o *bit* mais significativo é modificado para zero, desde que o valor seja positivo (chamado *bit* implícito ou oculto).

Sendo A! armazenado como:

Expoente (K+3)	MSB (K+2)	Próximo MSB (K+1)	LSB (K)
130	0	0	0

A! = −5 será armazenado como:

Expoente (K+3)	MSB (K+2)	Próximo MSB (K+1)	LSB (K)
128	128	0	0

A! = 7, assim:

Expoente (K+3)	MSB (K+2)	Próximo MSB (K+1)	LSB (K)
131	96	0	0

A! = −7:

Expoente (K+3)	MSB (K+2)	Próximo MSB (K+1)	LSB (K)
131	224	0	0

O zero é simplesmente armazenado com um expoente zero. Os outros *bytes* são irrelevantes.

Apêndices

Apêndice A

Sumário do CP 500

Neste apêndice estão resumidas as características do CP 500. Este sumário destina-se àqueles que já têm algum conhecimento da linguagem Basic e, principalmente, das características particulares de seu computador.

As informações aqui apresentadas foram dispostas de forma a permitir consultas rápidas e objetivas. Quando forem necessárias maiores informações deve-se recorrer aos capítulos específicos.

Abreviaturas e Caracteres Especiais

MODO IMEDIATO

Tecla	*Função*
ENTER	Introduz a linha digitada e interpreta o comando.
←	Retrocede o cursor e apaga o último caractere digitado.
SHIFT ←	Posiciona o cursor no começo da linha e a apaga.
↓	Muda para a próxima linha.
:	Delimitador de instrução, usado entre instruções na mesma linha lógica.
→	Move o cursor para a próxima parada de tabulação. As tabulações estão nas posições 0, 8, 16, 24, 32, 48 e 56.
SHIFT →	Converte a tela para 32 caracteres por linha.
CLEAR	Limpa a tela e converte para 64 caracteres por linha.
?	Usado no lugar de PRINT.
'	Usado no lugar de REM.
.	"linha atual", usado no lugar do número de linha com LIST, EDIT, etc.
SHIFT ↓	Para dar saída de um caractere de controle, pressione SHIFT e depois ↓. Enquanto se retêm as duas teclas, pressione a tecla do caractere de controle desejado. Por exemplo, para digitar o controle-Z, pressione SHIFT ↓ Z.

Nota: Você deve usar a tecla SHIFT da esquerda.

MODO DE EXECUÇÃO

Tecla	*Função*
SHIFT @	Pausa na execução; "congela" a tecla durante o comando LIST.
BREAK	Suspende a execução do programa e retorna o comando ao teclado.

CARACTERES ESPECIAIS

Caracteres de Declaração de Tipo		
Caractere	*Tipo*	*Exemplos*
$	*String*	A$,ZZ$
%	Inteiro	A1%,SOM%
!	Precisão simples	B!,N!
#	Precisão dupla	A#,1/3#
D	Precisão dupla (notação científica)	1.234567890D-12
E	Precisão simples (notação científica)	1.234567E-12

Operadores Matemáticos		
Caractere	*Operação*	*Exemplos*
+	Adição	2+2,A#+Z!
—	Subtração	2-N,-F
*	Multiplicação	2*A3,B*B
/	Divisão	N#/-.3,A/.3
[	Potenciação*	2[3,GL[-1

Operadores Relacionais		
Caractere	*Significado*	
	em expressões numéricas	**em expressões** *string*
<	menor que	precede (tem prioridade sobre)
>	maior que	segue (tem menor prioridade que)
=	igual a	tem mesma prioridade que
<= ou =<	menor ou igual a	tem prioridade menor ou igual a
>= ou =>	maior ou igual a	tem prioridade maior ou igual a
<> ou ><	diferente de	não tem a mesma prioridade que

*Nota: Use ↑ para obter este operador.

Operadores Lógicos	
Palavra-chave	*Operação*
NOT	Inversão lógica
AND	Multiplicação lógica (E)
OR	Soma lógica (OU)

Operador String		
Caractere	*Operação*	*Exemplos*
+	Concatenação	"2"+"2"="22"

Ordem das Operações	
Operador	*Observação*
Funções	As funções têm prioridade sobre todas as outras operações
[	Potenciação
+,—	sinais (—4, por exemplo)
*,/	Multiplicação e Divisão
+,—	Adição e Subtração
<,>,=,<=,>=,<>	
NOT	
AND	
OR	

Nos casos de operadores do mesmo nível de prioridade (mesma linha, na tabela acima), as operações são realizadas à medida em que aparecem na expressão, que é lida pelo computador da esquerda para a direita.

Comandos

Comando *Função* *Exemplos*

AUTO *mm, nn*

Ativa a numeração automática de linha, iniciando por *mm* e utilizando incrementos *nn*.

```
AUTO .,10
AUTO
```

CLEAR

Zera as variáveis e anula as *strings*.

```
CLEAR
```

Sintaxe	*Descrição*	*Exemplos*
CLEAR *n*		
	O mesmo que CLEAR , mas também separa *n bytes* para *strings*.	`CLEAR 500` `CLEAR MEM/4`
CLOAD		
	Copia um programa Basic da fita.	`CLOAD "A"`
CLOAD?		
	Compara um programa Basic na fita com um contido na memória.	`CLOAD? "A"`
CONT		
	Continua a execução após um BREAK ou STOP.	`CONT`
CSAVE		
	Armazena um programa Basic na fita.	`CSAVE "A"`
DELETE *mm-nn*		
	Apaga as linhas de programa da linha *mm* à linha *nn*.	`DELETE 100` `DELETE 10-50`
EDIT *mm*		
	Introduz o modo de edição para a linha *mm*. Consulte o capítulo 19 sobre os subcomandos do modo de edição.	`EDIT 100` `EDIT .`
LIST *mm-nn*		
	Lista todas as linhas do programa de *mm* a *nn*, inclusive.	`LIST` `LIST 30-60`
LLIST *mm-nn*		

	Lista todas as linhas do programa que tiverem numeração entre *mm* e *nn* na impressora, inclusive.	`LLIST 30-60`
		`LLIST 30-`
		`LLIST -60`

NEW		
	Apaga o programa inteiro e inicializa todas as variàveis, contadores internos, indicadores, etc.	`NEW`

Sintaxe *Descrição*	*Exemplos*
RUN *mm*	
Executa o programa iniciando pela linha *mm*, ou pela primeira, se *mm* não for especificado.	`RUN` `RUN 999`
SYSTEM	
Aciona o Monitor-residente (Apêndice G).	`SYSTEM`
TROFF	
Desativa a marcação das linhas executadas.	`TROFF`
TRON	
Ativa a marcação das linhas executadas (Capítulo 20).	`TRON`

Instruções

DEFINIÇÃO DE TIPO

DEFDBL *lista* ou *faixa de letras*

Define como precisão dupla todas as variáveis começando por letras dentro da faixa de letras especificada.

```
DEFDBL J
DEFDBL X,Y,Z
DEFDBL A-E, J
```

DEFINT *lista* ou *faixa de letras*

Define como sendo inteiras todas as variáveis começando por letras dentro da faixa de letras especificada.

```
DEFINT A
DEFINT C,E,G
DEFINT A-K
```

DEFSNG *lista* ou *faixa de letras*

Define como sendo de precisão simples todas as variáveis começadas por letras dentro da faixa de letras especificada.

```
DEFSNG L
DEFSNG A-L,Z
DEFSNG P,R,A-K
```

DEFSTR *lista* ou *faixa de letras*

Define como sendo *string* todas as variáveis começando por letra ou faixa de letras especificada.

```
DEFSTR A-Z
DEFSTR M,S-U
DEFSTR A,B,C
```

ATRIBUIÇÃO OU ALOCAÇÃO

Sintaxe *Descrição* *Exemplos*

CLEAR *n*

Separa um número especificado de n *bytes* para armazenagem de *string*. Limpa valores e tipos de todas as variáveis.

```
CLEAR 750
CLEAR MEM/10
CLEAR
```

DIM *matriz* (dim_1, dim_2 ...dim_k)

Define uma matriz de *k* dimensões com tamanho determinado por dimensão:
dim_1, dim_2 ...., etc...
Pode ser seguida por uma lista de variáveis separadas por vírgulas.

```
DIM A (2,3)
DIM A1(15),A2(15)
DIM A$(A!,B(X))
DIM B(X+2),C(J,K)
DIM T(3,3,5)
```

LET *variável* = *expressão*

Atribui o valor da *expressão* à *variável*.
A palavra-chave LET é opcional.

```
LET A$="CHARLIE"
A%=B+C
```

SEQÜÊNCIA DE EXECUÇÃO

END

Finaliza a execução, retorna para o modo imediato.

```
END
```

STOP

Pára a execução, imprime a mensagem "Break na" com o número da linha onde o programa foi interrompido. A execução pode ser retomada com a instrução CONT.

```
STOP
```

GOTO *n*

Desvia a execução do programa para a linha de número especificado (*n*).

```
GOTO 200
```

GOSUB *n*

Desvia a execução para a sub-rotina que começa pela linha de número especificado (*n*).

```
GOSUB 300
```

RETURN

Passa a execução do programa para a linha subseqüente à que contém a última instrução GOSUB executada.

```
2000 RETURN
```

Sintaxe *Descrição*	*Exemplos*
ON *exp* GOTO $linha_1$, $linha_2$, ..., $linha_k$ Avalia *exp;* se INT (*exp*) for igual a um dos números de 1 a *k*, a execução passará para o número de linha correspondente. De outra forma, avançará para a próxima linha de programa.	`ON K GOTO 100,200`
ON *exp* GOSUB Opera de forma similar a ON...GOTO, exceto no fato de que a execução é destinada para uma sub-rotina.	`ON J GOSUB 20,190`
FOR *var* = *exp* TO *exp* STEP *exp* Abre um *loop* FOR/NEXT, STEP e opcional. Caso não seja usado STEP, o incremento será igual a 1.	`FOR I=1TO9 STEP 2` `FOR M%=J% TO K%-1`
NEXT *variável* Fecha um loop FOR-NEXT. A *variável,* ou *lista de variáveis,* pode ser omitida.	`NEXT ITEM` `NEXT A,B,COD,D`
ON ERROR GOTO *n* Define o início de uma rotina de manipulação de erro; em caso de ocorrência de erro, a execução será desviada para a linha de número especificado (*n*).	`ONERRORGOTO 64000` `ON ERROR GOTO 0`
ERROR (*código*) Simula um erro especificado pelo *código* (Apêndice B).	`ERROR(14)` `ERROR(X!)`

RESUME *n*
Retorna da rotina de erro para a linha especificada por *n*. Se *n* for zero ou não for especificado, retornará à instrução contendo erro. Se *n* for NEXT, retornará para a instrução seguinte a instrução de erro.
	`RESUME`
	`RESUME 0`
	`RESUME 100`
	`RESUME NEXT`

Sintaxe	*Descrição*	*Exemplos*
RANDOM		
	Alimenta o gerador de números aleatórios.	`RANDOM`
REM		
	Indicador de comentário (REMark); ignora o restante da linha.	`REM A=AREA`

INSTRUÇÕES CONDICIONAIS

IF *exp* THEN *ação* ELSE *alternativa*

Testa exp: se verdadeira, executa a instrução *ação* e então salta para a próxima linha de programa (exceto se a *ação* for GOTO).
Se *exp* for falsa, saltará diretamente para instrução *alternativa* após ELSE e executará as instruções subseqüentes.

```
IF A=0THEN?"ZERO"
  ELSE ?"UM"
```

INSTRUÇÕES GRÁFICAS

CLS

Limpeza da tela.

```
CLS
```

RESET (*x,y*)

Desativa uma posição (ponto da tela) especificada pela coordenada horizontal x e pela vertical y, sendo que:
$0 <= x < 128$ e $0 <= y < 48$

```
RESET (24-X,11)
RESET (X,Y)
```

SET (*x,y*)

Ativa a posição (ponto da tela) especificada pelas coordenadas *x* e *y*. Mesma limitação de argumentos do RESET.

```
SET (X,Y)
SET (RND(47),Y)
```

INSTRUÇÕES ESPECIAIS

POKE *local, valor*

Carrega o *valor* em um certo *local* da memória (ambos os argumentos sob a forma decimal):
$0 < valor <= 255$

```
POKE 15635,34
POKE 17770,A+N
```

Sintaxe *Descrição*	*Exemplos*
OUT *porta, valor*	
Envia *valor* à *porta* (ambos os argumentos entre 0 e 255, inclusive).	`OUT 255,10` `OUT 55,A`

INSTRUÇÕES DE ENTRADA E SAÍDA

PRINT *exp*

Coloca na tela o valor da expressão *exp*, que pode ser uma constante, como pode também ser tanto numérica quanto *string*, ou, ainda, uma combinação destas.

```
PRINT A$
PRINT X+3
PRINT "D="D
```

A vírgula funciona como um modificador de impressão. Faz o cursor avançar para a próxima zona de impressão.

```
PRINT 1,2,3,4
PRINT  "1", "2"
PRINT 1,2
```

O ponto e vírgula funciona como um modificador de impressão. Insere um espaço após um item numérico na listagem PRINT. Não insere nenhum espaço após um item *string*. No final da listagem PRINT, suprime o retorno automático de carro.

```
PRINT X;Y;Z
PRINT"LADO=";L
```

PRINT @ *n*

Modificador de impressão; inicia a impressão na posição *n* especificada na tela.

```
PRINT@540,"*"
```

PRINT TAB (*n*)

Modificador de impressão; move o cursor para a posição *n* (expressão) especificada na tela.

```
PRINTTAB(10);A$
```

PRINT USING *string; exp*

Especificador de formato de impressão; coloca *exp* no formato especificado pela *string* (veja a seguir).

```
PRINT USING A$;X
?USING "#.#";Y+X
```

INPUT *"mensagem"; variável*

Imprime a *mensagem* (se houver) e espera uma entrada pelo teclado, a qual será armazenada na *variável.*

```
INPUT"NOME:",A$
INPUT "VALOR";X
INPUT A,B,C,D$
```

Sintaxe	*Descrição*	*Exemplos*
LPRINT		

	Opera da mesma forma que PRINT, imprimindo na impressora ao invés de na tela.	`LPRINTUSING A$;A!`
		`LPRINTA$;B#,C$,D!`
		`LPRINT A$`

PRINT #—1		
	Grava dados em fita cassete.	`PRINT#-1,A,B,C,D$`
INPUT #—1		
	Carrega, na memória, dados provenientes de um gravador cassete.	`INPUT#-1,A,B,C,D$`
DATA *lista de itens*		
	Retém os dados para acesso pela instrução READ.	`DATA 22,23,11,1.1`
READ *lista de variáveis*		
	Atribui valores às variáveis especificadas.	`READ A,A1,A2,A3`
RESTORE		
	Reajusta o indicador de dados ao primeiro item na primeira instrução DATA.	`RESTORE`

ESPECIFICADORES DE CAMPO PARA INSTRUÇÕES PRINT USING

Especificador	*Função*
#	
	Campo numérico (um dígito por #).
.	
	Posição do ponto decimal.
+ ou —	
	Imprime sinais dianteiros e traseiros. + para números positivos e — para negativos.
—	
	Imprime sinal traseiro apenas se o valor a imprimir é negativo.

*** ***

Preenche espaços em branco com asteriscos.

$$

Coloca o cifrão imediatamente a esquerda do primeiro dígito.

*** *$**

Coloca o cifrão à esquerda do primeiro dígito e preenche os espaços em branco anteriores com asteriscos.

[[[[ou ↑↑↑↑

Formato exponencial com um dígito significativo à esquerda do decimal. Pressione ■ para introduzir este caractere.

Imprime números com vírgulas, como em 1,356,000.

!

Caractere único.

%espaço%

String com comprimento igual a 2 mais o número de espaços entre os símbolos de porcentagem.

Funções

FUNÇÕES NUMÉRICAS *

Sintaxe	*Operação*	*Exemplos*
ABS *(exp)*		
	Fornece o valor absoluto do valor de *exp*.	`ABS(L*.7)`
ATN *(exp)*		
	Fornece o arco-tangente em radianos.	`ATN(2.7)`
CDBL *(exp)*		
	Fornece representação do valor de *exp* em precisão dupla.	`CDBL(A)` `CDBL(A+1/3#)`

Nota: *exp* é qualquer expressão ou constante numérica. Exceto onde especificado, $-1.7E+38 <= exp <= 1.7E+38$.

Sintaxe	*Descrição*	*Exemplos*
CINT (*exp*)		
	Fornece o maior inteiro não maior do que o valor de *exp*. Limite: —32768 < = *exp* < + 32768.	`CINT(A#+B)`
COS (*exp*)		
	Fornece o cosseno de *exp*, assumindo *exp* em radianos.	`COS(2*A)` `COS(A/57.29578)`
CSNG (*exp*)		
	Fornece representação de precisão simples, arredondamento 5/4 do decimal menos significativo, quando *exp* tem precisão dupla.	`CSNG(33*B#)` `CSNG(A#)`
EXP (*exp*)		
	Fornece o exponencial natural, e^{exp} = EXP (*exp*)	`EXP(34.5)` `EXP(A*B*C-1)`
FIX (*exp*)		
	Fornece o inteiro equivalente a *exp* truncada (a parte fracionária de *exp* é eliminada).	`FIX(A-B)`
INT (*exp*)		
	Fornece o maior inteiro não maior que *exp*.	`INT(A+B*C)`
LOG (*exp*)		
	Fornece o logaritmo natural (base e) de *exp*. Limites: *exp* deve ser positiva.	`LOG(12.33)` `LOG(A*B+B)`
RND (0)		
	Fornece um número pseudo-aleatório entre 0.000001 e 0.999999, inclusive.	`RND(0)`
RND (*exp*)		
	Fornece um número pseudo-aleatório entre 1 e INT (*exp*), inclusive. Limites: 1 < = *exp* < 32768	`RND(40)` `RND(A+B)`
SGN (*exp*)		
	Fornece — 1 se *exp* for negativo, 0 para *exp* = 0 e + 1 para *exp* positivo.	`SGN(A*B+3)` `SGN(COS(X))`

Sintaxe	*Descrição*	*Exemplos*
SIN (*exp*)	Fornece o seno de *exp*, sendo *exp* em radianos.	`SIN(A/B)` `SIN(90/57.29578)`
SQR (*exp*)	Fornece a raiz quadrada de *exp*. Limites: *exp* não deve ser negativa.	`SQR(A*A-B*B)`
TAN (*exp*)	Fornece a tangente de *exp*, sendo *exp* em radianos.	`TAN(X)` `TAN(X*.01745329)`

FUNÇÕES STRING

ASC (*string**)	*Fornece o código ASCII do primeiro caractere da string.*	`ASC(B$)` `ASC("H")`

CHR$ (*código*)	Fornece uma *string* de um caractere, definida pelo *código*. Se o *código* especifica uma função de controle, aquela função é ativada.	`CHR$(A)`
		`CHR$(1)`
		`CHR$(3+4)`

FRE (*string*)	Fornece o espaço disponível de memória para armazenagem de *string*. O argumento (variável ou constante) apenas satisfaz a sintaxe da função.	`FRE(A$)`
INKEY$	Varre o teclado e fornece uma *string* com o caractere correspondente à tecla pressionada durante a varredura do teclado (*string* nula se nenhuma tecla for pressionada).	`INKEY$`
LEFT$ (*string, n*)	Fornece os *n* primeiros caracteres de uma *string*.	`LEFT$(A$,1)` `LEFT$(A$,M+L)`

*Nota: *string* pode ser uma variável, expressão ou constante *string*.

Sintaxe	*Descrição*	*Exemplos*
LEN (*string*)		
	Fornece o comprimento da *string* (zero para *string* nula).	`LEN(A$+B$)` `LEN("HORA")`
MID$ (*string, n, p*)		
	Fornece, a partir do enésimo caractere da *string* original, uma *substring* de comprimento *P*.	`MID$(M$+B$,P,L-1)` `MID$(M$,5,2)`
RIGHT$ (*string, n*)		
	Fornece os *n* últimos caracteres da *string*.	`RIGHT$(NA$,7)` `RIGHT$(AB$,M2)`
STR$ (*exp*)		
	Fornece a representação *string* do valor de *exp* (expressão numérica).	`STR$(A+B*2)`
STRING$ (*n, carac*)		
	Fornece uma seqüência de *n* caracteres definidos pelo código *carac*.	`STRING$(30,".")` `STRING$(5,C$)`
TIME$		
	Fornece data e horas.	`PRINT TIME$`
VAL (*string*)		
	Fornece um valor numérico correspondente a uma *string* dotada de valor numérico.	`VAL("1"+A$)` `VAL(G1$)`

FUNÇÕES ESPECIAIS

Função	*Operação e Limites*	*Exemplos*
ERL		
	Fornece o número da linha do último erro ocorrido.	`ERL`
ERR		
	Fornece um valor relacionado ao código do erro corrente (caso haja algum erro). ERR = (código de erro—1)*2 ou: (ERR/2)+1 = código de erro	`ERR/2+1`

Função	*Operações e Limites*	*Exemplos*
INP (*porta*)	Introduz e fornece o valor corrente da porta especificada. Tanto o resultado como o argumento encontram-se na faixa de 0 a 255, inclusive.	`INP(55)`
MEM	Fornece o número total de *bytes* desprotegidos e não utilizados da memória. Não inclui o espaço não utilizado para armazenagem de *strings*.	`MEM`
PEEK (*local*)	Fornece o valor guardado no *byte* especificado de memória. O local deve ser um endereço válido de memória, sob a forma decimal (veja o Mapa de Memória, no apêndice C).	`PEEK(15730)`
POINT (*x,y*)	Verifica o ponto da tela especificado pelas coordenadas *x* e *y*. Se o ponto estiver "ativo", fornece um nível verdadeiro (—1); senão, fornece um nível falso (0). Limites: 0 < =x < 128;0 < = y < 48	`POINT(0,47)`
POS (0)	Fornece um número indicando a posição corrente do cursor. O argumento "0" apenas satisfaz a síntaxe da função.	`? POS(0)`
USR (*n*)	Desvia para uma sub-rotina em linguagem de máquina. Para maiores detalhes, consulte o capítulo 12.	`X=USR(0)`
VARPTR (*var*)	Fornece o endereço onde estão armazenados o nome, o valor e o indicador de uma variável; *var* deve ser um nome válido de variável.	`VARPTR(A$)`

Palavras Reservadas para o Basic *

@	ELSE	LLIST	RENAME
ABS	END	LPRINT	RESET
AND	EOF	LOAD	RESTORE
ASC	ERL	LOC	RESUME
ATN	ERR	LOF	RETURN
AUTO	ERROR	LOG	RIGHT$
CDBL	EXP	MEM	RND
CHR$	FIELD	MERGE	RSET
CINT	FIX	MID$	RUN
CLEAR	FN	MKD$	SAVE
CLOCK	FOR	MKI$	SET
CLOSE	FORMAT	MKS$	SGN
CLS	FRE	NAME	SIN
CMD	FREE	NEW	SQR
CONT	GET	NEXT	STEP
COS	GOSUB	NOT	STOP
CSNG	GOTO	ON	STRING$
CVD	IF	OPEN	STR$
CVI	INKEY$	OR	SYSTEM
CVS	INP	OUT	TAB
DATA	INPUT	PEEK	TAN
DEFDBL	INSTR	POINT	THEN
DEFFN	INT	POKE	TIME$
DEFINT	KILL	POS	TO
DEFSNG	LEFT$	POSN	TROFF
DEFUSR	LET	PRINT	TRON
DEFSTR	LSET	PUT	USING
DELETE	LEN	RANDOM	USR
DIM	LINE	READ	VAL
EDIT	LIST	REM	VARPTR
			VERIFY

Limites de Memória para Uso Dinâmico

FAIXAS ABRANGIDAS

Inteiros: de −32768 a + 32767, inclusive.

Precisão simples: de −1.701411E±38 a 1.701411E±38, inclusive.

---

*Nota: Algumas destas palavras não têm função no BASIC do CP 500; elas estão reservadas para uso do BASIC com disco. Nenhuma delas pode ser utilizada como nome de variável; você incorrerá em um erro de sintaxe, caso tente usar alguma delas com esse objetivo.

Precisão dupla: de −1.701411834544556D±38 a 1.701411834544556D±38, inclusive.

Faixa de *strings:* até 255 caracteres.

Numeração permitida de linha: de 0 a 65529, inclusive.

Extensão de linha de programa: até 255 caracteres (entrada 240, edição 255).

REQUISITOS DE MEMÓRIA

As linhas de programa requerem no mínimo 5 *bytes*, da seguinte forma:

Número de linha: 2 *bytes.*

Indicador de linha: 2 *bytes.*

Retorno da linha: 1 *byte.*

Além disso, cada palavra rèservada, operador, nome de variável, caractere especial e caractere constante exige 1 *byte.*

ALOCAÇÃO DINÂMICA DE MEMÓRIA (EM OPERAÇÃO)

Variáveis inteiras: 5 *bytes* cada (2 para o valor, 3 para o nome).

Variáveis de precisão simples: 7 *bytes* cada (4 para o valor, 3 para o nome).

Variáveis de precisão dupla: 11 *bytes* cada (8 para o valor e 3 para o nome).

Variáveis *string:* 6 *bytes* no mínimo, distribuídos da seguinte forma:

3 para o nome
3 para o indicador de *stack* e variável
1 para cada caractere.

Variáveis matriz: 12 *bytes* no mínimo, da seguinte forma:

3 para o nome
2 para o tamanho total
1 para o número de dimensões
2 para o tamanho de cada dimensão
2, 3, 4, ou 8 para cada elemento da matriz (dependendo do tipo da mesma)

Loop **FOR/NEXT:** 16 *bytes.*

Parênteses: 4 *bytes* para cada nível, mais 12 *bytes* para cada valor temporário.

FÓRMULA GERAL PARA COMPUTAR OS REQUISITOS DE MEMÓRIA PARA MATRIZES

A matriz G (N_1, N_2, ...,N_k) requer o seguinte espaço de memória:

$$14 + (k*2) + T* \{(N_1+1)*(N_2+1)*...*(N_k+1)\}$$

onde:

k é o número de dimensões da matriz;

T é um fator que depende do tipo de matriz:

TIPO	T =
inteiro	2
precisão simples	4
precisão dupla	8
*string**	3

Precisão dos Cálculos

Cálculos de precisão simples envolvendo as quatro operações básicas (+, —, * e /) têm precisão de até seis dígitos significativos.

Cálculos de precisão dupla, envolvendo as mesmas operações, têm precisão de até 16 dígitos significativos.

O operador de exponenciação ↑ (exibido como "["), tem precisão simples, confiável em até quatro algarismos significativos.

As funções logarítmicas e trigonométricas têm precisão simples; outras funções têm a precisão em função do argumento de entrada e da função.

Assim, por exemplo, CDBL fornece um valor de precisão dupla, e ABS fornece um valor com a mesma precisão do argumento de entrada.

Ao converter de precisão simples para dupla, utilize a seguinte técnica para evitar a introdução de valores incorretos nos dígitos adicionais de precisão:

variável de precisão dupla = VAL (STR$ (*variável de precisão simples*))

*Nota: Ao computar os requisitos de memória para matriz em *string*, você deve acrescentar a extensão do texto de cada elemento da matriz. Quando a matriz é dimensionada pela primeira vez, todos os elementos têm extesão 0. O texto da *string* será armazenado no espaço reservado para strings (reservado pela instrução CLEAR *n*).

Apêndice B

Códigos de Erros

Códigos	*Abreviação*	*Erro*
01	NF	NEXT sem o FOR
02	SN	Erro de sintaxe
03	RG	RETURN sem GOSUB
04	OD	Insuficiência de dados
05	FC	Chamada ilegal da função
06	OV	Overflow
07	OM	Insuficiência de memória
08	UL	Linha indefinida
09	BS	Subscrito fora da faixa
10	DD	Matriz dimensionada
11	/0	Divisão por zero
12	ID	Imediato ilegal
13	TM	Introdução ilegal
14	OS	Falta de espaço
15	LS	*String* muito extensa
16	ST	Fórmula de *string* muito complexa
17	CN	Impossibilidade de continuar
18	NR	Falta RESUME
19	RW	RESUME sem Erro
20	UE	Erro não imprimível
21	MO	Falta de operando
22	FD	Dados defeituosos de arquivo
23	L3	Somente em BASIC-DISCO

Explicação das mensagens de erro

NF

NEXT sem FOR: NEXT utilizado sem uma instrução FOR correspondente. Esse erro também pode ocorrer se a instrução NEXT variável for invertida num *loop* entrelaçado.

SN

Erro de sintaxe: é o resultado, normalmente, de pontuação incorreta; parênteses abertos, um caractere ilegal ou um comando errado.

RG

RETURN sem GOSUB: uma instrução RETURN foi encontrada antes que um GOSUB correspondente fosse executado.

OD

Insuficiência de Dados: uma instrução READ ou INPUT foi executada com dados insuficientes. A instrução DATA pode ter sido deixada de fora, ou todos os dados podem ter sido lidos, da fita ou DATA.

FC

Chamada ilegal de função: tentativa de exercer uma operação empregando um parâmetro ilegal. Exemplos: raiz quadrada de um argumento negativo, dimensão negativa de matriz, argumentos negativos ou nulos em logaritmos, etc. Ou então, chamada de USR sem que o ponto de entrada tenha passado por um POKE.

OV

Overflow (sobrecarga): a magnitude do número introduzido, ou derivado, é muito grande para o computador *

Os números entre ±1.701411E-38 (precisão simples) e ±1.7014118345445-56D-38 (precisão dupla) são arredondados para 0. Veja /0 a seguir.

OM

Insuficiência de memória: toda a memória disponível foi utilizada ou reservada. Pode ocorrer com grandes dimensões de matrizes, ramificações como o GOTO, GOSUB e FOR/NEXT.

UL

Linha indefinida: tentativa de referir ou desviar uma linha inexistente.

BS

Subscrito fora de faixa: tentativa de estipular um elemento de matriz com um subscrito fora da faixa dimensionada.

DD

Matriz redimensionada: tentativa de dimensionar uma matriz já previamente dimensionada por uma instrução DIM. É bom colocar todas as instruções de dimensionamento no início de cada programa.

Nota: Matematicamente falando, não existe erro de sobrecarga. O problema todo se resume na capacidade do computador representar valores dentro de uma faixa limitada.

/0

Divisão por zero: tentativa de utilizar o valor zero no denominador*.

ID

Imediato ilegal: utilização do comando INPUT como sendo imediato.

TM

Introdução ilegal: tentativa de atribuir uma variável simples, a um *string* ou vice-versa.

OS

Falta de espaço para *string*: o espaço reservado para *string* foi excedido.

LS

String muito extensa: uma variável *string* recebeu um valor *string* que excedeu 255 caracteres.

ST

Fórmula de *string* muito complexa: operação *string* muito complexa para o computador. A operação deve ser dividida em etapas.

CN

Impossibilidade de continuar um programa: O CONT foi executado em uma situação onde é impossível dar continuidade ao programa. Por exemplo, após um END ou EDIT.

NR

Falta RESUME: o processamento termina (por END ou simplesmente por acabarem-se as linhas) durante a execução de uma rotina de manipulação de erros.

RW

Resume sem erro: uma instrução RESUME foi encontrada antes que um ON ERROR GOTO fosse executado.

UE

Erro não imprimível: tentativa de gerar um erro, usando uma instrução ERROR, com um código inválido.

Nota: Se for impossível localizar uma divisão por zero que seja óbvia, verifique divisões por números inferiores às faixas permitidas. Veja também erro OV, acima, e as faixas válidas no capítulo anterior.

MO

Falta de operando: tentativa de operação sem o fornecimento de um dos operandos necessários.

FD

Dados defeituosos em arquivo: entrada de dados de uma fonte externa (fita por exemplo) de forma incorreta ou em uma seqüência imprópria.

L3

Somente o BASIC-DISCO: tentativa de utilizar uma instrução, função ou comando, somente disponível no BASIC-DISCO (DOS 500).

Apêndice C

Códigos de Caracteres do CP 500

Os códigos de caracteres são necessários quando se quer representar dentro da memória do computador. Como o computador pode representar apenas números, associamos um número a cada caractere que queremos representar. Por exemplo: a letra A é representada pelo código 65; as funções de controle e os gráficos também são representados por códigos; e os códigos de caractere assumem valores de zero a 255.

Os códigos de zero a 31 normalmente representam funções de controle. Por exemplo: o código 13 representa um retorno de carro ou "final de linha".

Entretanto, no CP 500, estes mesmos códigos também representam 32 caracteres especiais na tela. Para esta aplicação, eles devem ser colocados (POKE) na memória e não serem impressos (PRINT).

Os códigos de 32 a 127 representam todos os caracteres normalmente utilizados em texto: as letras, números e outros. Os caracteres de texto do CP 500 obedecem ao padrão ASCII (*American National Standard Code for Information Interchange*).

Os códigos 128 a 191, quando endereçados à tela do vídeo, representam os 64 caracteres gráficos.

Os códigos 192 a 255, quando endereçados à tela do vídeo, representam códigos de compressão de espaços ou caracteres especiais, conforme determinado por *software*.

Muitos dos códigos podem ser introduzidos pelo teclado; todos eles podem ser armazenados em uma *string* e endereçadas para qualquer periférico.

Por exemplo, para enviar um código 31 para a tela use uma instrução como:

```
PRINT CHR$(31)
```

Para maiores detalhes, veja "Usando a Tela do Vídeo" neste manual.

Na tabela a seguir resumimos os caracteres de controle de vídeo e do teclado.

Nota: Na Tabela a seguir, *end* se refere ao endereço decimal da memória RAM da tela (endereços 15360 a 16383), e *cód.* ao código decimal do caractere (primeira coluna).

Código		*Teclado*	*Efeito na Tela*	POKE*end,cód.*
Dec.	Hex.		PRINTCHR$ (cód.)	
0	00		Sem efeito	
1	01	BREAK	Sem efeito	
		SHIFT ↓ A		
2	02	SHIFT ↓ B	Sem efeito	
3	03	SHIFT ↓ C	Sem efeito	
4	04	SHIFT ↓ D	Sem efeito	
5	05	SHIFT ↓ E	Sem efeito	
6	06	SHIFT ↓ F	Sem efeito	
7	07	SHIFT ↓ G	Sem efeito	
8	08	←	Retrocesso e limpeza	
		SHIFT ↓ H		
9	09	→	Tabulação (0,8,16,24,...)	
		SHIFT ↓ I		
10	OA	↓	Move o cursor para o começo da	
		SHIFT ↓ J	próxima linha e apaga a linha	
11	OB	SHIFT ↓ K	Sem efeito	
12	OC	SHIFT ↓ L	Sem efeito	
13	OD	ENTER	Move o cursor para o	
		SHIFT ↓ M	começo da próxima linha e apaga a linha	
14	OE	SHIFT ↓ N	Ativa o cursor	*Ver caracteres especiais de 0 a 31 na Página 155
15	OF	SHIFT ↓ O	Desativa o cursor	
16	10	SHIFT ↓ P	Sem efeito	
17	11	SHIFT ↓ Q	Sem efeito	
18	12	SHIFT ↓ R	Sem efeito	
19	13	SHIFT ↓ S	Sem efeito	
20	14	SHIFT ↓ T	Sem efeito	
21	15	SHIFT ↓ U	Controla compressão de espaços/ caracteres especiais	
22	16	SHIFT ↓ V	Controla caracteres especiais/ alternativos	
23	17	SHIFT ↓ W	Caracteres tamanho duplo (32 cpl)	
24	18	SHIFT ←	Retrocesso sem limpeza	
		SHIFT ↓ X		
25	19	SHIFT ↓ Y	Avança o cursor	
26	1A	SHIFT ↓ Z	Desce o cursor	
27	1B	SHIFT ↑	Sobe o cursor	
28	1C	SHIFT ↓ ,	Move o cursor para o canto superior esquerdo	

Código		*Teclado*	*Efeito na Tela*	POKE*end, cód.*
Dec.	Hex.		PRINTCHR$ (cód.)	
29	1D	SHIFT ↓ 9	Apaga a linha e recomeça	
30	1E	SHIFT ↓ .	Apaga o final da linha	
31	1F	CLEAR SHIFT ↓ /	Apaga toda a tela	
32	20	ESPAÇO	½	½
33	21	!	!	!
34	22	"	"	"
35	23	#	#	#
36	24	$	$	$
37	25	%	%	%
38	26	&	&	&
39	27	'	'	'
40	28	(	(	(
41	29	)	)	)
42	2A	*	*	*
43	2B	+	+	+
44	2C	,	,	,
45	2D	-	-	-
46	2E	.	.	.
47	2F	/	/	/
48	30	0	0	0
49	31	1	1	1
50	32	2	2	2
51	33	3	3	3
52	34	4	4	4
53	35	5	5	5
54	36	6	6	6
55	37	7	7	7
56	38	8	8	8
57	39	9	9	9
58	3A	:	:	:
59	3B	;	;	;
60	3C	<	<	<
61	3D	=	=	=
62	3E	>	>	>
63	3F	?	?	?
64	40	@	@	@
65	41	A	A	A
66	42	B	B	B

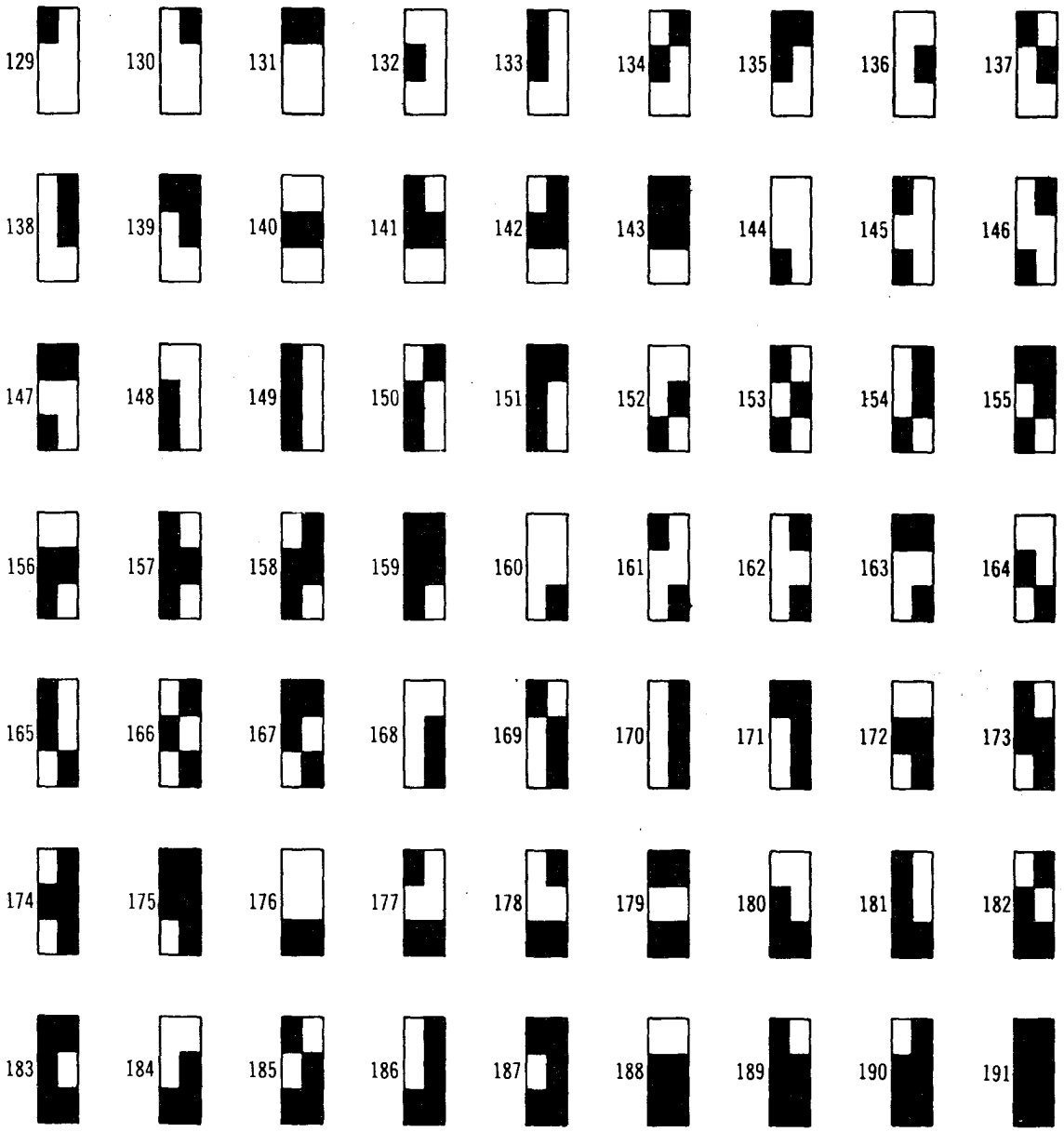
Código		*Teclado*	*Efeito na Tela*	POKE*end, cód.*
Dec.	Hex.		PRINTCHR$ (cód.)	
67	43	C	C	C
68	44	D	D	D
69	45	E	E	E
70	46	F	F	F
71	47	G	G	G
72	48	H	H	H
73	49	I	I	I
74	4A	J	J	J
75	4B	K	K	K
76	4C	L	L	L
77	4D	M	M	M
78	4E	N	N	N
79	4F	O	O	O
80	50	P	P	P
81	51	Q	Q	Q
82	52	R	R	R
83	53	S	S	S
84	54	T	T	T
85	55	U	U	U
86	56	V	V	V
87	57	W	W	W
88	58	X	X	X
89	59	Y	Y	Y
90	5A	U	Z	Z
91	5B		[	[
92	5C		\	\
93	5D		]	]
94	5E		^	^
95	5F			
96	60	SHIFT		
97	61	A	a	a
98	62	B	b	b
99	63	C	c	c
100	64	D	d	d
101	65	E	e	e
102	66	F	f	f
103	67	G	g	g
104	68	H	h	h
105	69	I	i	i

Código		*Teclado*	*Efeito na Tela*	POKE*end, cód.*
Dec.	Hex.		PRINTCHR$ (cód.)	
106	6A	J	j	j
107	6B	K	k	k
108	6C	L	l	l
109	6D	M	m	m
110	6E	N	n	n
111	6F	O	o	o
112	70	P	p	p
113	71	Q	q	q
114	72	R	r	r
115	73	S	s	s
116	74	T	t	t
117	75	U	u	u
118	76	V	v	v
119	77	W	w	w
120	78	X	x	x
121	79	Y	y	y
122	7A	Z	z	z
123	7B		{	{
124	7C		:	:
125	7D		}	}
126	7E		~	~
127	7F		±	±
128	80	Os códigos 128 a 191 são caracteres gráficos. Veja a tabela de gráficos neste Apêndice.		
192	C0	Os códigos 192 a 255 podem ser códigos de compressão de espaço em caracteres especiais, quando usados com PRINT CHR$ (*cód*).		
255	FF	Quando você usar POKE, *end, cód* sempre será considerado o caractere especial correspondente. Veja a tabela de Caracteres Especiais.		

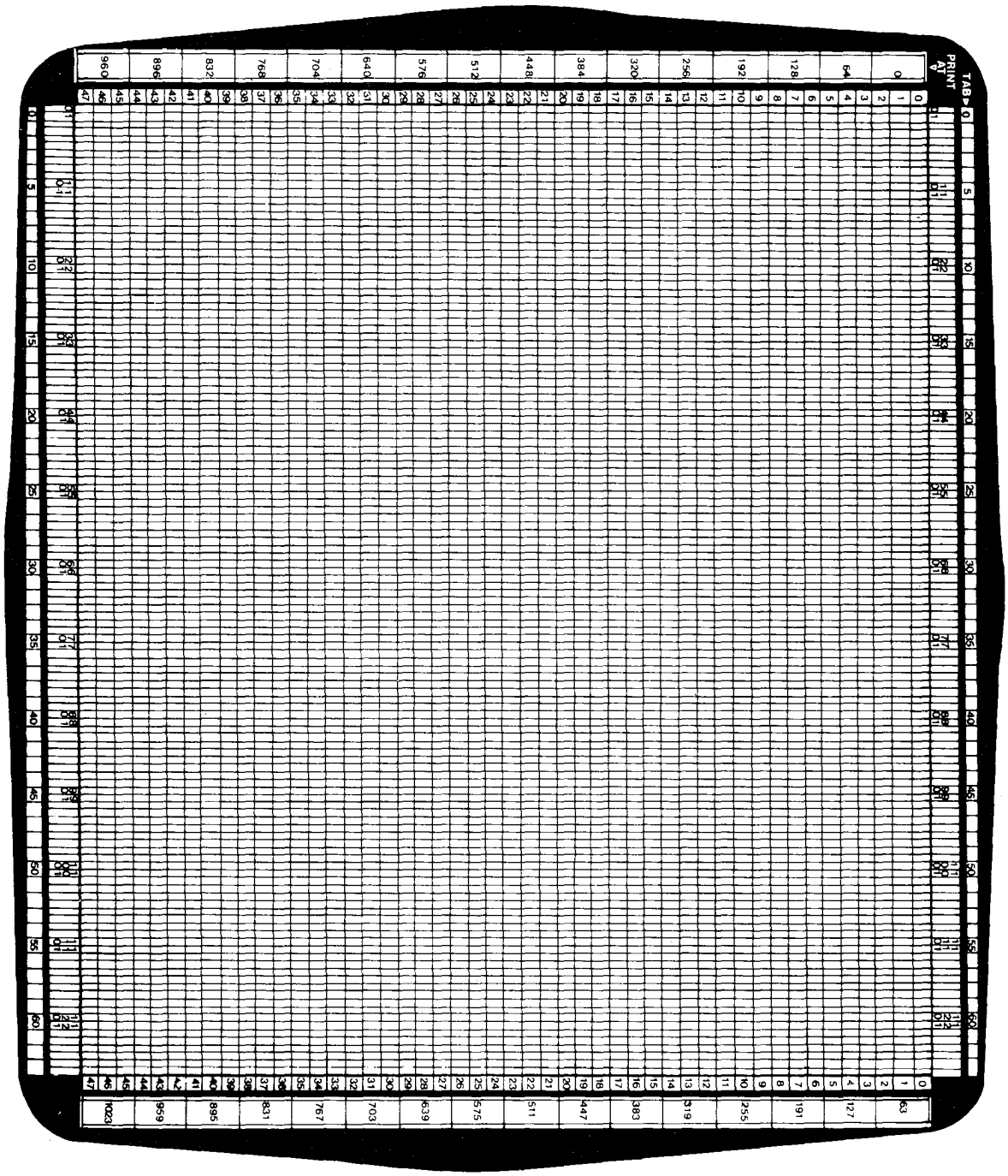
Alguns desses caracteres do teclado só podem ser introduzidos com a função INKEY$.

Caracteres Especiais (de 0 a 31 e de 192 a 255)

Caracteres Gráficos (de 128 a 191)



Lay-out da Tela (Endereços 15360 a 16383)



Apêndice D

Códigos Internos das Palavras Chaves em Basic

São os seguintes os códigos que o computador usa para armazenar as palavras-chave em BASIC.

Se você pegar (instrução PEEK) no *buffer* de programa (começa no endereço 17129 em decimal) você encontrará seu programa armazenado com estes códigos:

Código Decimal	*Palavra-Chave BASIC*	*Código Decimal*	*Palavra-Chave BASIC*
129	FOR	153	DEFINT
130	RESET	154	DEFSNG
131	SET	155	DEFDBL
132	CLS	156	LINE
133	CMD	157	EDIT
134	RANDOM	158	ERROR
135	NEXT	159	RESUME
136	DATA	160	OUT
137	INPUT	161	ON
138	DIM	162	OPEN
139	READ	163	FIELD
140	LET	164	GET
141	GOTO	165	PUT
142	RUN	166	CLOSE
143	IF	167	LOAD
144	RESTORE	168	MERGE
145	GOSUB	169	NAME
146	RETURN	170	KILL
147	REM	171	LSET
148	STOP	172	RSET
149	ELSE	173	SAVE
150	TRON	174	SYSTEM
151	TROFF	175	LPRINT
152	DEFSTR	176	DEF

Código Decimal	*Palavra-Chave BASIC*	*Código Decimal*	*Palavra-Chave BASIC*
177	POKE	214	<
178	PRINT	215	SGN
179	CONT	216	INT
180	LIST	217	ABS
181	LLIST	218	FRE
182	DELETE	219	INP
183	AUTO	220	POS
184	CLEAR	221	SQR
185	CLOAD	222	RND
186	CSAVE	223	LOG
187	NEW	224	EXP
188	TAB	225	COS
189	TO	226	SIN
190	FN	227	TAN
191	USING	228	ATN
192	VARPTR	229	PEEK
193	USR	230	CVI
194	ERL	231	CVS
195	ERR	232	CVD
196	STRING$	233	EOF
197	INSTR	234	LOC
198	POINT	235	LOF
199	TIME$	236	MKI$
200	MEM	237	MKS$
201	INKEY$	238	MKD$
202	THEN	239	CINT
203	NOT	240	CSNG
204	STEP	241	CDBL
205	+	242	FIX
206	−	243	LEN
207	*	244	STR$
208	/	245	VAL
209		246	ASC
210	AND	247	CHR$
211	OR	248	LEFT$
212	>	249	RIGHT$
213	=	250	MID$

Apêndice E

Funções Derivadas

Algumas vezes, as funções matemáticas de que o Basic dispõe são insuficientes para uma determinada tarefa. Nesses casos, torna-se necessário recorrer a funções que não podem ser calculadas diretamente em Basic.

Para resolver esse problema, apresentamos a seguir as fórmulas que relacionam as principais funções trigonométricas e hiperbólicas com as funções disponíveis no Basic.

Secante:

```
SEC(X) = 1/COS(X)
```

Cossecante:

```
CSC(X) = 1/SIN(X)
```

Cotangente:

```
COT(X) = 1/TAN(X)
```

Arco-seno:

```
ARCSIN(X) = ATN(X/SQR(-X*X+1))
```

Arco-cosseno:

```
ARCCOS(X) = -ATN(X/SQR(-X*X=1)) + 1.5708
```

Arco-secante:

```
ARCSEC(X) = ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
```

Arco-cossecante:

```
ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
```

Arco-cotangente:

```
ARCOT(X) = -ATN(X)+1.5708
```

Seno hiperbólico:

```
SINH(X) = (EXP(X)-EXP(-X))/2
```

Cosseno hiperbólico:

```
COSH(X) = (EXP(X)+EXP(-X))/2
```

Tangente hiperbólica:

```
TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
```

Secante hiperbólica:

```
SECH(X) = 2/(EXP(X)+EXP(-X))
```

Cossecante hiperbólica:

```
CSCH(X) = 2/(EXP(X) -EXP(-X))
```

Cotangente hiperbólica:

COTH(X) = EXP(−X)EXP(X)−EXP(−X))*2+1

Arco-seno hiperbólico:

ARGSINH(X) = LOG(X+SQR(X*X+1))

Arco-cosseno hiperbólico:

ARGCOSH(X) = LOG(X+SQR(X*X−1))

Arco-tangente hiperbólico:

ARGTANH(X) = LOG((1+X)/(1−X))/2

Arco-secante hiperbólico:

ARGSECH(X) = LOG((SQR(−X*X+1)+1)/X)

Arco-cossecante hiperbólico:

ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)

Arco-cotangente hiperbólico:

ARGCOTH(X) = LOG((X+1)/(X−1)/2)

Faixas Válidas de Entrada

Existem restrições que devem ser observadas quando se estiver trabalhando com as funções acima. O argumento (X) para cada caso deve estar dentro de faixas aceitáveis pela função calculada. As restrições são as seguintes:

Arco-seno	−1<X<1
Arco-cosseno	−1<X<1
Arco-secante	X<−1,X>1
Arco-cossecante	X<−1,X>1
Arco-cosseno hiperbólico	X>1
Arco-tangente hiperbólico	X*X<1
Arco-secante hiperbólico	0<X<1
Arco-cossecante hiperbólico	X<>0
Arco-cotangente hiperbólico	X*X<1

Certos valores são matematicamente indefinidos, porém, nossas funções podem calcular valores inválidos. É o caso, por exemplo, da tangente de 90 graus (1.5708 radianos) que, como sabemos, não pode ser calculada.

Se pedirmos a função TAN(90*0.01745329), o computador responderá com um erro de "divisão por zero".

Entretanto, a expressão TAN(1.5708) é calculada sem problemas (1.5708 é o mesmo que 90*0.01745329), devido ao arredondamento do valor de precisão simples (internamente, o computador opera com sete dígitos).

Esse mesmo problema se verifica com a tangente de 270 graus, com as secantes de 90 e 270 graus e com as cotangentes e cossecantes de 0 e 180 graus.

Devemos destacar, então, algumas relações úteis:

ARCSIN(−1) = −PI/2 *
ARCSIN (1) = PI/2
ARCCOS (−1) = PI
ARCCOS (1) = 0
ARCSEC (−1) = −PI
ARCSEC (1) = 0
ARCCSC (−1) = −PI/2
ARCCSC (1) = PI/2

Note que as informações acima podem estar incompletas.

Nota: *PI = 3.141592

Apêndice F

Conversão de Base

DEC.	*HEX.*	*BINÁRIO*
0	00	00000000
1	01	00000001
2	02	00000010
3	03	00000011
4	04	00000100
5	05	00000101
6	06	00000110
7	07	00000111
8	08	00001000
9	09	00001001
10	0A	00001010
11	0B	00001011
12	0C	00001100
13	0D	00001101
14	0E	00001110
15	0F	00001111
16	10	00010000
17	11	00010001
18	12	00010010
19	13	00010011
20	14	00010100
21	15	00010101
22	16	00010110
23	17	00010111
24	18	00011000
25	19	00011001
26	1A	00011010
27	1B	00011011
28	1C	00011100
29	1D	00011101
30	1E	00011110
31	1F	00011111
32	20	00100000

DEC.	*HEX.*	*BINÁRIO*
33	21	00100001
34	22	00100010
35	23	00100011
36	24	00100100
37	25	00100101
38	26	00100110
39	27	00100111
40	28	00101000
41	29	00101001
42	2A	00101010
43	2B	00101011
44	2C	00101100
45	2D	00101101
46	2E	00101110
47	2F	00101111
48	30	00110000
49	31	00110001
50	32	00110010
51	33	00110011
52	34	00110100
53	35	00110101
54	36	00110110
55	37	00110111
56	38	00111000
57	39	00111001
58	3A	00111010
59	3B	00111011
60	3C	00111100
61	3D	00111101
62	3E	00111110
63	3F	00111111
64	40	01000000
65	41	01000001

DEC.	HEX.	BINÁRIO
66	42	01000010
67	43	01000011
68	44	01000100
69	45	01000101
70	46	01000110
71	47	01000111
72	48	01001000
73	49	01001001
74	4A	01001010
75	4B	01001011
76	4C	01001100
77	4D	01001101
78	4E	01001110
79	4F	01001111
80	50	01010000
81	51	01010001
82	52	01010010
83	53	01010011
84	54	01010100
85	55	01010101
86	56	01010110
87	57	01010111
88	58	01011000
89	59	01011001
90	5A	01011010
91	5B	01011011
92	5C	01011100
93	5D	01011101
94	5E	01011110
95	5F	01011111
96	60	01100000
97	61	01100001
98	62	01100010
99	63	01100011
100	64	01100100
101	65	01100101
102	66	01100110
103	67	01100111
104	68	01101000
105	69	01101001
106	6A	01101010

DEC.	HEX.	BINÁRIO
107	6B	01101011
108	6C	01101100
109	6D	01101101
110	6E	01101110
111	6F	01101111
112	70	01110000
113	71	01110001
114	72	01110010
115	73	01110011
116	74	01110100
117	75	01110101
118	76	01110110
119	77	01110111
120	78	01111000
121	79	01111001
122	7A	01111010
123	7B	01111011
124	7C	01111100
125	7D	01111101
126	7E	01111110
127	7F	01111111
128	80	10000000
129	81	10000001
130	82	10000010
131	83	10000011
132	84	10000100
133	85	10000101
134	86	10000110
135	87	10000111
136	88	10001000
137	89	10001001
138	8A	10001010
139	8B	10001011
140	8C	10001100
141	8D	10001101
142	8E	10001110
143	8F	10001111
144	90	10010000
145	91	10010001
146	92	10010010
147	93	10010011

DEC.	HEX.	BINÁRIO
148	94	10010100
149	95	10010101
150	96	10010110
151	97	10010111
152	98	10011000
153	99	10011001
154	9A	10011010
155	9B	10011011
156	9C	10011100
157	9D	10011101
158	9E	10011110
159	9F	10011111
160	A0	10100000
161	A1	10100001
162	A2	10100010
163	A3	10100011
164	A4	10100100
165	A5	10100101
166	A6	10100110
167	A7	10100111
168	A8	10101000
169	A9	10101001
170	AA	10101010
171	AB	10101011
172	AC	10101100
173	AD	10101101
174	AE	10101110
175	AF	10101111
176	B0	10110000
177	B1	10110001
178	B2	10110010
179	B3	10110011
180	B4	10110100
181	B5	10110101
182	B6	10110110
183	B7	10110111
184	B8	10111000
185	B9	10111001
186	BA	10111010
187	BB	10111011
188	BC	10111100

DEC	HEX	BINÁRIO
189	BD	10111101
190	BE	10111110
191	BF	10111111
192	C0	11000000
193	C1	11000001
194	C2	11000010
195	C3	11000011
196	C4	11000100
197	C5	11000101
198	C6	11000110
199	C7	11000111
200	C8	11001000
201	C9	11001001
202	CA	11001010
203	CB	11001011
204	CC	11001100
205	CD	11001101
206	CE	11001110
207	CF	11001111
208	D0	11010000
209	D1	11010001
210	D2	11010010
211	D3	11010011
212	D4	11010100
213	D5	11010101
214	D6	11010110
215	D7	11010111
216	D8	11011000
217	D9	11011001
218	DA	11011010
219	DB	11011011
220	DC	11011100
221	DD	11011101
222	DE	11011110
223	DF	11011111
224	E0	11100000
225	E1	11100001
226	E2	11100010
227	E3	11100011
228	E4	11100100
229	E5	11100101

DEC	*HEX*	*BINÁRIO*
230	E6	11100110
231	E7	11100111
232	E8	11101000
233	E9	11101001
234	EA	11101010
235	EB	11101011
236	EC	11101100
237	ED	11101101
238	EE	11101110
239	EF	11101111
240	F0	11110000
241	F1	11110001
242	F2	11110010

DEC	*HEX*	*BINÁRIO*
243	F3	11110011
244	F4	11110100
245	F5	11110101
246	F6	11110110
247	F7	11110111
248	F8	11111000
249	F9	11111001
250	FA	11111010
251	FB	11111011
252	FC	11111100
253	FD	11111101
254	FE	11111110
255	FF	11111111

Apêndice G

Monitor Residente

O CP 500 além da linguagem BASIC, possui a capacidade de operar, através de seu Monitor-residente, em linguagem de máquina (Z 80).

Com o Monitor-residente pode-se verificar as condições reais do microprocessador, verificando seu conteúdo, conferindo os programas na memória. Para tanto, proceda da forma descrita a seguir.

PROCEDIMENTO

CP 500 Sem Disquete: Ao ligar o equipamento pressione o botão RESET. Aparecerá na tela a seguinte mensagem:

```
BASIC (S ou N)?
```

Para acionar o monitor digite N e o sistema passará o controle para o Monitor.

CP 500 Com Disquete: Devem ser pressionadas as teclas BREAK e RESET ao mesmo tempo. Aparecerá então a seguinte mensagem:

```
BASIC (S ou N)?
```

Para acionar o monitor digite N e o sistema passará o controle para o monitor. Quando o controle é passado ao monitor aparecerá a seguinte mensagem:

```
MONITOR VERSAO 1.1 1982
```

Comandos do Monitor

D

Lista memória (Hexa e ASCII).

Sintaxe:

D *hexa,bytes* ENTER

onde *hexa* é o endereço inicial da parte da memória a ser listada e *bytes* indica (em hexa) quantos *bytes* deverão ser mostrados.

Nota: O endereço inicial deve ser qualquer endereço válido:
Para 16k temos de 0 até 7FFF;
Para 32k temos de 0 até BFFF;
Para 48k temos de 0 até FFFF.

Exemplo:

D6C20,B0 ENTER

Efeito:

Lista 176 (B0) *bytes* a partir do endereço 6CD0 da memória. Portanto, lista do endereço 6CD0 até o 6D80.

S

Substitui o conteúdo do endereço da memória fornecido.

Sintaxe:

S *hexa* ENTER

onde *hexa* indica o endereço do valor a ser alterado.

Após esse comando será apresentado o seguinte:

endereço conteúdo subcomando

Nessa situação o computador está aguardando um dos seguintes subcomandos:

ENTER	Sai do comando sem alterar o conteúdo;
/	Passa para o endereço seguinte;
–	Volta ao endereço anterior;
xx	Um novo conteúdo. Esse deve ser formado por dois dígitos, em hexadecimal.

Exemplos:

S64D3 ENTER

Endereço	*Conteúdo*	*Sub-comando*	*Efeito*
64D3	FC	/	Passa para o endereço seguinte;
64D4	7F	4 D	Alteração do conteúdo;
64D5	3C	–	Retorna ao endereço anterior;
64D4	4D	ENTER	Saída do subcomando.

R

Altera os registradores do Z 80.

Sintaxe:

R

Ao ser dado este comando, serão visualizados os registradores e seus respectivos conteúdos, da seguinte forma:

```
 I  AF   BC   DE   HL   AF'  BC'  DE'  HL'  IY   IX   SP   PC
xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
```

onde xx representa o conteúdo em hexadecimal de cada registrador (xxxx indica um par de registradores).

Para alterar um dos registradores deve ser dado a sigla do mesmo seguida de um espaço. Será então apresentado o seguinte:

Sigla do registrador **Conteúdo** **Subcomando**

O computador aceita um dos seguintes subcomandos:

Novo conteúdo ESPAÇO

Altera o conteúdo do registrador.

CLEAR

Salta para o próximo registrador.

ENTER

Sai do comando.

Exemplos:

```
R
 I  AF   BC   DE   HL   AF'  BC'  DE'  HL'  IY   IX   SP   PC
3D 4D53 327A D8F0 3CFC 54D9 A9DF 5F3C 7524 6328 632A 403B 62B7
```

Registrador	*Conteúdo*	*Subcomando*	*Efeito*
A	4D	ESPAÇO	Salta para o próximo registrador;
F	32	3 2 ESPAÇO	Altera conteúdo do registrador F e salta para o próximo;
BC	32	CLEAR	Salta para o próximo registrador;
CD	7A	D 8 ESPAÇO	
		6 D ENTER	Altera o conteúdo de CD e sai do comando.

J

Salta para o endereço especificado e inicia a execução.

Sintaxe:

J *início, fim* ENTER

onde *início* é o endereço hexadecimal do ponto de partida da sub-rotina, e *fim*, o endereço hexadecimal do *break point*.

Assim que for pressionada a tecla ENTER haverá uma execução a partir do endereço de início até encontrar o *break point*. Cuidado ao definir o *break point*, pois se colocado em uma área de dados ou em um ponto que não seja executável a operação não passará por ele.

Nota: Registradores de 8 *bits* são apresentados separadamente e não aos pares. Seu conteúdo é dois dígitos hexadecimais.
Os registradores de 16 *bits* são apresentados em siglas de dois caracteres. Seu conteúdo é quatro dígitos hexadecimais.

Exemplo:

J8000 ENTER

Efeito:

Inicia a execução em 8000 sem haver um *break point.*

J8000,803B ENTER

Efeito:

Inicia a execução em 8000 e finaliza ao encontrar o *break point* no endereço 803B.

G

Continua a execução a partir do endereço do último *break point.*

Sintaxe:

G *break* ENTER

onde *break* é o endereço do novo *break point.*

Exemplos:

G813D ENTER

Efeito:

Continua a execução que foi interrompida em 803B (do exemplo anterior) até o endereço 813D que é o novo *break point.*

G ENTER

Retoma a execução que foi interrompida no *break point* 813D, sem parar em outro *break point.*

T

Transfere o conteúdo da RAM para a fita cassete (veja exemplo).

Sintaxe:

T *Código Baud Rate*
Nome do Programa ENTER
Endereço inicial (hexa) ENTER
Número de bytes ENTER

Nota: Sempre que o cassete for usado, a mensagem "Cass?" aparecerá devendo então ser informada a taxa de transferência, usando uma das seguintes teclas:
A para uma *baud rate* de 1500 baud
B para uma *baud rate* de 500 baud

Exemplo:

```
T
Cass? ENTER
NOME END.INICIO QTOS BYTES
MON ENTER
8000 ENTER
400 ENTER
```

Salvará o conteúdo da RAM do endereço 8000 até 83FF (1k) no arquivo de nome MON na fita.

C

Carrega um programa da fita para a memória.

Sintaxe:

C
Código de baud rate
Nome do arquivo ENTER

O programa será procurado na fita cassete e quando for encontrado será carregado na memória. Durante a carga será visualizado os dois asteriscos no canto superior direito, um dos quais é intermitente. Após a carga do programa será visualizado na tela o endereço de início do programa.

A operação poderá ser abortada (interrompida) pela tecla BREAK.

Exemplo:

```
C
Cass? ENTER
MON ENTER
8000 ENTER
```

Notas: A - Programas do tipo SYSTEM são incompatíveis com o Monitor, isto é, fitas SYSTEM não são carregadas pelo Monitor e vice-versa.

B - A tecla RESET não apaga a memória, desde que:
1 - Não se pressione RESET por muito tempo;
2 - Ao sair do monitor com RESET se entre novamente no monitor, sem *"boot"* do DOS 500 e sem nenhuma entrada no BASIC-RESIDENTE. (Responda sempre não para BASIC(S/N)?);

C - A tecla ←, apesar de ativa, não é interpretada pelo monitor. Em caso de erro digite os *bytes* na seqüência desejada. Por exemplo, ao examinar a memória de 6250 até 6260, pode-se digitar o seguinte comando:

```
D6246250,0010 ENTER
```

Serão considerados apenas os últimos *bytes*, sendo desprezados os excedentes à esquerda.

D - Nomes de arquivos podem ter tantos caracteres quantos se desejar (não há limite), sendo que para a carga do arquivo será exigido o mesmo nome.

E - Se alguma rotina criada ou carregada pelo monitor for utilizada pelo Basic, pressione RESET pa-

ra sair do monitor (mantenha essa tecla pressionada se o sistema tiver *drive* de disquete), chame Basic (BASIC S/N? S) e proteja a área de memória onde se encontra a rotina para que o Basic não a sobreponha.
Mem. usada? (máximo 17686 (decimal)).
A rotina criada pelo monitor deverá estar após o endereço 45FF (hexa) para não ser encoberta pelo Basic.

F - Se algum engano for cometido em qualquer comando (exceto, J,G ou T), poderá ser interrompido pressionando-se a tecla BREAK, voltando o monitor ao seu início sem alteração dos registradores ou da memória.

G - O retorno de um programa ao monitor é feito executando-se um *jump* para o endereço 4300H (C30043).

H - O registrador SP *(stack pointer)* não é iniciado pelo monitor devendo o usuário iniciá-lo para uma área de RAM reservada para o *stack* antes de qualquer comando, toda vez que entrar no monitor.

Apêndice H

Glossário

A

Acesso - maneira pela qual o computador obtém ou atinge um conjunto de dados.

Acumulador - registrador para armazenamento de dados durante a operação do computatador. Endereços e informações podem ser guardados temporariamente nesse dispositivo. Certos tipos são colocados à disposição do usuário, enquanto outros ficam rservados para uso interno do processador.

Aleatorizar - dispor dados de forma que não respeitem nenhuma seqüência ou relação pré-determinada. Dispor dados de forma aleatória.

Alfanumérico - qualquer um dos caracteres usados nas linguagens de computação. Pode ser uma letra, um número ou um símbolo padronizado.

ALGOL - abreviação de *ALGOrithmic Language* (linguagem algorítmica). Linguagem de aplicação geral, mas com ênfase nas aplicações de cunho numérico. Linguagem algorítmica de orientação científica.

Algoritmo - processo ou conjunto de regras fixas que, numa seqüência passo a passo, leva a um resultado determinado.

Alimentar - suprir o computador com os dados que deve processar. Pode significar, ainda, o ato de fornecer ao equipamento energia elétrica.

Alocação - preencher áreas específicas de memória com blocos de dados determinados. Reservar áreas de armazenamento para as rotinas e sub-rotinas, fixando assim os valores absolutos de todos os endereços simbólicos.

ALU - Arithmetic Logic Unit (unidade lógico-aritmética) Parte do *hardware* do computador onde são executadas as operações lógicas e aritméticas.

Analógico - representação de dados de forma contínua, normalmente apresentados como grandezas físicas, como tensão, corrente, rotação, etc.

APL - abreviatura de *A Programming Language* (uma linguagem de programação). Linguagem com características de notação, ainda não completamente implementada, dedicada especialmente a operações matemáticas, como valores matrizes, etc.

Aquisição de dados - captação e reunião de dados, a fim de que possam ser aproveitados pelo computador.

Arquivo - conjunto de dados reunidos, formando uma unidade. Acumulação de informação na memória do computador.

Array - o mesmo que arranjo ou distribuição. Lista de dados indexados, que podem ser acessados diretamente.

ASCII - abreviação de *American Standard Code for Information Interchange* (código padrão americano para intercâmbio de informações). Codificação que emprega palavras de 8 *bits* (incluindo o *bit* verificador de paridade), adotada com o objetivo de facilitar a troca de dados entre sistemas de processamento ou entre o computador e seus periféricos.

Assembler - pode ser traduzido como "programa de montagem". Consiste em um programa encarregado de atuar sobre dados simbólicos para produzir, a partir deles, instruções de máquina capazes de efetuar determinadas funções, tais como: tradução de códigos de operação em instruções de computador, determinação de locais de memória para instruções secessivas ou computação de endereços absolutos a partir de endereços simbólicos.

Assembly - o mesmo que "montagem". Tradução de programa fonte que é escrito em linguagem de máquina.

B

Bandeira - *bit* de informação acrescentado a um caractere ou palavra, para indicar a fronteira de um campo de dados. Caractere que sinaliza a ocorrência de alguma condição, tal como o fim de uma palavra.

Barramento de dados (data bus) - meio físico de transporte de dados entre o microprocessador e os demais elementos que compõem o computador (memórias, registradores, etc.)

Barramento de endereços - (address bus) - meio físico de informação sobre as coordenadas ("endereços") que os dados devem ocupar na memória RAM.

BASIC - Abreviatura de *Beginner's All-purpose Symbolic Instruction Code* (código simbólico de instrução, universal, para principiantes). Linguagem de aplicação geral, desenvolvida para o ensino da programação de computadores; pode ser aplicada tanto em programas comerciais como científicos, mas adapta-se melhor a estes.

Baud - unidade de velocidade de sinais, igual ao número de dados por segundo. Deriva do nome Baudot, que deu origem também ao código de mesmo nome.

BCD - abreviação de *Binary-Coded Decimal* (decimal codificado em binário). Representação numérica pela qual os dígitos decimais são apresentados sob a forma de números binários. Exemplo: o número 13, em decimal, tem como equivalente 0001 0011, em BCD.

Biblioteca de programas - conjunto organizado de programas, rotinas ou de *software* específico ou genérico, para uso em um determinado sistema.

Binário - sistema numérico que emprega a base 2, ao invés da base 10 normal, dispondo assim dos dígitos 0 e 1, apenas, para formar números.

BIT - abreviatura de *Binary digit* (dígito binário). Caractere de um número binário. Unidade de informação ou de capacidade de memória.

Bit mais significativo - *bit* que contribui com a maior porção do valor de uma palavra. O *bit* mais à esquerda de um número binário.

Bit menos significativo - o inverso de *bit* mais significativo. O *bit* mais a direita de um número binário.
Branch - ver Ramo ou ramificação.
Busca - processo de recolher instruções que serão aproveitadas pela CPU, armazenando-as num registrador adequado.
Byte - número de *bits* que o computador considera como uma unidade. A menor unidade endereçável na memória de um computador. Normalmente, consiste de oito *bits* de dados e um *bit* de paridade.
Byte mais significativo - os oito *bits* mais significativos.
Byte menos significativo - os oito *bits* menos significativos.

C

Cadeia, cordão - seqüência interligada de caracteres, palavras ou outros elementos.
Caractere - símbolo utilizado na representação gráfica de dados.
Campo (field) - conjunto de um ou mais caracteres tratados como um todo. Área especificada de um registro usada para uma categoria específica de dados.
Carregar - preencher a memória interna de um computador com informações provindas de sistemas externos ou auxiliares de armazenagem.
Chamada de sub-rotina - passagem do controle do computador do programa principal para uma determinada sub-rotina.
Clock - dispositivo interno do computador encarregado de sincronizar os vários estágios do sistema. Também chamado de "relógio".
COBOL - abreviatura de *Common Business Oriented Language* (linguagem de orientação comercial). Trata-se de uma linguagem especialmente projetada para processamento de dados da área comercial, definida e desenvolvida por um comitê americano de fabricantes e usuários de computadores.
Código Baudot - codificação padronizada de dados para teletipo, em cinco canais, consistindo de um impulso de partida, cinco impulsos de caracteres, todos de igual extensão, e um impulso de parada.
Código objeto - codificação produzida por um compilador ou um *assembler* especial para ser executada num computador.
Código operacional - conjunto de símbolos que designam a execução de uma operação básica do computador. Parte da instrução que determina a operação lógica, aritmética ou de transferência que deve ser executada.
Comando - Sinal ou conjunto de sinais elétricos que dá início, encerra ou dá seqüência a uma operação. Parte da instrução que especifica a operação a ser efetuada. (Obs: é incorreto utilizar o termo "comando" como sinônimo de "instrução").
Compilação - conversão de um programa fonte em rotinas específicas, escritas em linguagem de máquina. Desenvolver ou produzir um programa ordenado lógica ou seqüencialmente, em linguagem de máquina, a partir de uma série de códigos operacionais mnemônicos ou simbólicos.
Compilador (compiler) - rotina de montagem de programas, capaz de produzir um pro-

grama específico para um determinado fim, ao estabelecer o sentido inerente a elementos de informação. Programa de computador ainda mais poderoso que o *assembler*; além da função de tradução, é também capaz de substituir certos itens de entrada por séries de instruções, normalmente chamadas de sub-rotinas.

Compressão de dados - Série de técnicas usadas para a redução de espaço, largura de banda, custo, transmissão, e armazenagem de dados. Tais técnicas visam à eliminação de repetições, a remoção de dados irrelevantes e o emprego de processos especiais de codificação. Também chamado de compactação de dados.

Comprimento de palavra - medida de extensão de uma palavra de computador, normalmente dividida em *bytes, bits*, etc.

Computador - dispositivo capaz de aceitar informações, aplicar processos pré-determinados às mesmas e fornecer os resultados desses processos. Consiste, normalmente, de dispositivos de entrada e saída (I/O), unidades de armazenagem, de cálculos aritméticos e lógicos, além de uma unidade de controle (CPU).

Computer-aided design - ver Projeto auxiliado por computador.

Contador de programa - registrador que retém a identificação da palavra de instrução prestes a ser executada. Também chamado de registrador de controle.

CPU - ver Unidade central de processamento.

Cross-assembler - tradutor de linguagem simbólica que "roda" em um tipo de computador, a fim de produzir linguagem de máquina para outro tipo de computador. Pode ser chamado, também, de *assembler* cruzado.

D

Dados - nome genérico dado aos elementos básicos de informação que podem ser processados ou produzidos por um computador.

Dados formatados - arranjo pré-determinado de caracteres, campos, linhas, pontuação, números de página, etc.

Debug - processo de localização e correção de quaisquer erros num programa de computador. Teste de programa, a fim de assegurar de que funciona corretamente.

Decisão - processo de verificação, normalmente por comparação, da existência ou não de uma certa condição, como resultado de uma ação alternativa. Operação, efetuada pelo computador, de determinar a existência de alguma relação entre palavras guardadas na memória ou em registrador e, a partir daí, seguir caminhos alternativos.

Decodificação - execução das operações internas pelas quais o computador determina o sentido do código operacional de uma instrução.

Delete - remover, eliminar dados, como, por exemplo, cancelar um registro de um arquivo principal.

Demodulação - processo de se obter o sinal original a partir de uma portadora modulada. Técnica normalmente utilizada para tornar os sinais de comunicação compatíveis com os sinais de máquinas comerciais.

Digitação - normalmente, a introdução de dados num computador através de teclado apropriado.

Digital - referente à utilização de números discretos integrais, numa determinada base, para representar todas as quantidades que ocorrem num problema ou cálculo.
Dígito - um dos "n" símbolos de valor inteiro, variando de 0 a n-1, pertencente a um sistema de numeração (Os dígitos 0 a 9 constituem o sistema decimal, enquanto 0 e 1 fazem parte do sistema binário, por exemplo).

E

Editar - processo de rearranjar dados ou informações, que pode consistir da remoção de dados indesejáveis, seleção de dados importantes, aplicação de técnicas de formatação, inserção de símbolos, aplicação de processos padronizados e teste de dados.
Enable - ver Habilitar.
Endereço - identificação do local, registrador ou unidade em que se encontram informações armazenadas.
Entrada/Saída (E/S) - termo genérico usado para o equipamento com que o computador se comunica com o exterior. Os dados envolvidos em tal comunicação.
Evento - ocasião ou ação que faz os dados afetarem o conteúdo dos arquivos.

F

Feed - ver Alimentar.
Fetch - ver Busca.
File - ver Arquivo.
Fim de arquivo (end of file) - término ou ponto final de uma certa quantidade de dados; indicações especiais de fim de arquivo, demarcam esse ponto.
Flag - ver Bandeira.
Fluxo - termo genérico que indica uma seqüência de eventos.
Fluxograma - representação gráfica de uma seqüência de operações, num programa, empregando símbolos para representar as várias operações, tais como comparação, salto, leitura, escrita, etc.
FORTRAN - abreviação de *FORmula TRANslator* (conversor de fórmulas). Sistema de programação, incluindo uma linguagem e um compilador, que permite escrever programas com notação matemática, foi desenvolvido originalmente pela IBM, destinado a fins científicos, mas atualmente presta-se também a resolução de inúmeros problemas comerciais.
Frase - na programação, consiste em uma expressão ou uma instrução generalizada em linguagem fonte.
Firmware - programas ou instruções armazenados em memória ROM. O firmware é análogo ao *software*, sob forma de *hardware*.

H

Habilitar - permitir o início ou a seqüência de operação de um determinado dispositivo.
Hardware - conjunto dos elementos mecânicos, elétricos, magnéticos e eletrônicos que compõem o computador.

Hexadecimal - sistema de numeração que emprega a base 16. Nesse sistema, os dígitos são representados pelos algarismos de 0 a 9 e pelas letras de A a F (no lugar dos números de 10 a 15).

I

Impressora - dispositivo que atua como periférico do computador, fornecendo dados impressos, sob a forma de uma listagem.

Indexação - modificação de endereços, normalmente efetuado pelos chamados registradores de indexação (ou indexadores).

Índice - tabela de palavras ou campos que contém endereços de dados localizados nos arquivos.

Indicador - registrador da CPU que contém os endereços de memória.

Interface - ponto ou dispositivo de contato entre dois elementos, no interior do computador, ou entre estes dispositivos externos.

Interrupção - quebra no fluxo normal de um sistema ou uma rotina, de forma que possa ser retomado mais tarde. Sinal de controle que desvia a atenção do computador, quando este está atarefado com o programa principal, para um endereço específico, diretamente relacionado com o tipo de interrupção ocorrida.

Instrução - passo codificado de programa que diz ao computador o que fazer a cada operação. Conjunto de caracteres que, juntamente com endereço, define uma operação e faz com que o computador aja como pedido sobre as quantidades indicadas.

I/O (input/output) - ver Entrada/Saída.

J

Jump - ver Salto.

L

Laço - um conjunto de instruções que deve ser repetido várias vezes.

Linguagem - em computação, um meio de comunicação entre o homem e a máquina, ou de partes do computador entre si ou, ainda, entre computadores.

Linguagem de alto nível - uma linguagem em que os comandos são passíveis de serem entendidos pelo operador, sem que o mesmo tenha que conhecer, necessariamente, a arquitetura do computador ou sua linguagem de máquina.

Linguagem de máquina - conjunto de símbolos, caracteres ou sinais e suas regras de combinação, para formar instruções para um determinado computador.

Listagem - impressão de um conjunto de dados, instruções ou resultados de um programa.

Load - ver Carregar.

Loader - rotina de serviço, cuja finalidade é ler programas para a memória central, para fins de execução.

Loop - ver Laço.

LSB - ver *byte* menos signficativo.

M

Macro-instrução - instrução de uma linguagem de alto nível, formada por várias instruções em linguagem de máquina. Ou então uma instrução de um microprocessador formada por várias micro-instruções internas do mesmo.

Mainframe - conjunto formado pela CPU, unidade de entrada/saída e memória. É o computador, menos os periféricos.

Memória - Dispositivo ou conjunto de dispositivos que armazena um dado ou conjunto de dados para aproveitamento posterior.

Memória dinâmica - Memória em que os dados precisam ser continuamente realimentados para que não se percam.

Memória estática - Memória que não precisa de realimentação para manter seus dados.

Memória rascunho - Memória em que os dados são resultados parciais de uma operação qualquer e, posteriormente, poderão ser modificados. Ao se terminar a operação, o resultado final será transferido para outra posição de memória, caso seja necessária a sua preservação para uso posterior ou para uma saída.

Memória Virtual - Memória com capacidade de usar um tipo de algoritmo de paginação ou segmentação. Por este meio, pode-se simular uma memória maior que aquela que realmente existe.

Memory Dump - Uma listagem do conteúdo da memória ou parte dele, visando à pesquisade erros ou defeitos no computador.

Microcomputador - Computador cuja CPU é formada por um microprocessador.

Micro-instrução - Uma instrução em linguagem de máquina, que é usada para formar uma instrução em linguagem de alto nível. Em uma unidade de controle microprogramada, cada uma das pequenas instruções que forma uma instrução em linguagem de máquina.

Microprocessador - a unidade central de processamento de um microcomputador, formada por um único CI.

Microprograma - programa em linguagem de máquina que forma uma instrução em uma linguagem de alto nível (Macro-instrução). Em uma unidade de controle microprogramada, um microprograma é um programa formado por micro-instruções, que forma uma instrução em linguagem de máquina.

Modem (modulador/demodulador) - tem a função, em teleprocessamento, de codificar e decodificar os dados em função de sua transmissão por via de cabos telefônicos.

MSB - Ver dígito mais significativo.

N

Nested subroutine - Ver Sub-rotina aninhada.

O

Octal - número na base oito.

On line (em linha) - equipamento ou dispositivo sob controle direto da CPU.

Op code - Ver código operacional.

P

Página - num agrupamento de memórias, cada uma dessas memórias forma uma página. Num microcomputador de oito *bits*, uma página pode ser constituída por uma memória de $2^8 = 256$ *bits*.

Palavra - um grupo de caracteres ocupando uma localização da memória de um computador. Para a unidade de controle, uma palavra é uma instrução e, para a unidade lógico-aritmética, uma quantidade.

Paridade - um método de checagem da precisão de um número binário. Um *bit* adicional, chamado *bit* de paridade, é acrescentado ao número. Se a paridade par for usada, a soma de todos os *bits* 1 do número, mais o *bit* de paridade, será sempre par; se for usada a paridade ímpar, esta soma sempre será ímpar; ocorrendo resultado par, houve erro.

PASCAL - linguagem estruturada, baseada no ALGOL 60, destinada ao ensino de programação e uso geral em microcomputadores.

Passo de programa - operação numa rotina de um computador. Cada uma das instruções executadas num programa.

Pattern - modelo, padrão.

Periférico - unidade auxiliar que pode ser colocada sob controle do processador central, tal como unidades de discos, terminais de vídeo, impressoras, gravadores, etc.

Pilha - tipo de memória rascunho usada durante operações de expressões numéricas, em que o primeiro dado que entra é o último que sai.

Pista - parte da área gravada de um meio de armazenamento (fita, tambor, disco ou disquete), a que se tem acesso por meio de um determinado posicionamento da cabeça de leitura.

Pointer - ver indicador.

Polling (interrogação periódica) - técnica em que cada um dos terminais que compartilham uma linha de comunicação é periodicamente interrogado, para determinar se necessitam de serviços da CPU.

Porta (Port) - dispositivo que comanda a entrada e saída de um sistema. Em comunicação de dados, a parte de um processador de dados que é dedicada exclusivamente a um único canal de dados, com o fim de recebê-los e transmiti-los para um ou mais terminais remotos.

Print - instrução comum a várias linguagens, que coloca os dados em uma saída de forma legível ao operador humano. Esta saída pode ser uma folha impressa por impressora ou dados num terminal de vídeo.

Prioridade - gradação atribuída a uma tarefa que determina sua precedência quanto ao uso dos recursos do sistema.

Procedimento (procedure) - curso de ação adotado para a solução de um determinado problema.

Programa - seqüência de instruções que determinam ao computador o procedimento adotado para resolver determinado problema.

Programa fonte (source program) - programa em linguagem simbólica.

Programa objeto (object program) - programa em linguagem de máquina executável pelo processador. É criado a partir de um programa compilado.

Programa compilado - programa em códigos relocáveis traduzido a partir do programa fonte.

Projeto e Fabricação Auxiliados por Computador (CAD/CAM) - uso do computador em automação industrial, engenharia, biologia, estatística, etc., para agilizar a compilação e análise de dados, projeto físico de equipamentos, cálculo estrutural, e outras tarefas que envolvam um grande número de informações.

R

RAM - Memória de Acesso Selecionável - Memória em que se tem acesso a qualquer posição, tanto para escrita quanto para leitura.

Randomize - Ver Aleatorizar.

Ramo ou ramificação - desvio na seqüência normal do programa, por uma instrução do tipo *jump.*

Read - instrução comum a várias linguagens, que indica ao computador que ele deve ler determinados dados vindos de um dispositivo de entrada (como em Cobol ou Fortran), ou de uma lista de dados dentro do próprio programa (como em Basic).

Reforço da memória - técnica usada em memórias dinâmicas, em que o dado presente em cada uma das células é realimentado para evitar que, com o passar do tempo, ele se perca.

Refresh - ver Reforço de memória.

Registrador - parte interna de um microprocessador que armazena dados para serem processados pela CPU.

Registrador de deslocamento - registrador em que os dados podem ser deslocados, quer para a direita, quer para a esquerda, de forma serial.

Registrador de instruções - um dos registradores de uma CPU cuja função é armazenar a instrução, enquanto esta está sendo interpretada.

Relocar - deslocar uma rotina ou bloco de dados dentro da memória física do computador, realizando as devidas correções no endereçamento envolvido.

Reset - recolocar um dispositivo em sua condição inicial de operação. O mesmo que inicializar.

ROM - memória apenas de leitura. Memória que contém funções e rotinas permanentes, necessárias ao funcionamento do computador e que não podem ser alteradas.

Rotina - conjunto de instruções dispostas em posição adequada que instrui o computador a realizar uma operação determinada.

Rotina de checagem (check routine) - rotina que tem por finalidade verificar se o computador está operando corretamente.

Run - instrução comum a várias linguagens, que indica ao computador para executar um determinado programa. O mesmo que rodar um programa.

S

Salto (jump) - tipo de instrução que ordena ao computador que se dirija a um ponto determinado do programa, que não seja a instrução seguinte na seqüência normal.

Scratchpad memory - ver Memória rascunho.
Serial - entrada ou saída em que os *bits* estão dispostos um após o outro, para facilitar a transmissão e recepção de dados.
Shift register - ver Registrador de deslocamento.
Sistema - conjunto de componentes ou equipamentos arranjados de forma a realizar uma ou várias ações determinadas.
Software - conjunto de programas, procedimentos e documentação relativos à operação de um sistema de processamento de dados.
Stack - ver Pilha.
Statement - ver Frase.
String - ver cadeia ou cordão.
Sub-rotina - rotina repetitiva. Em geral trata-se de um procedimento que é utilizado várias vezes dentro de um programa.
Sub-rotina aberta - procedimento que aparece em vários pontos de um mesmo programa.
Sub-rotina fechada - procedimento indexado. Toda vez que for necessária seu uso, será realizado um salto para esta sub-rotina e, ao terminar sua função, o processamento retorna ao endereço seguinte ao ponto de desvio.
Sub-rotina aninhada - sub-rotina que foi acionada por uma outra sub-rotina.

T

Teletipo (TTY, teletype) - equipamento tele-impressor usado em sistemas de comunicação. Pode ser usado também em transmissão de dados, como equipamento de entrada ou saída.
Tempo partilhado - meio de usar um sistema de computação, de forma a permitir executar e interagir vários programas de diferentes usuários ao mesmo tempo.
Tempo real - regime de trabalho de um computador ou controlador em que as respostas são dadas imediatamente após a entrada dos dados.
Terminal - dispositivo de entrada ou saída que permite a comunicação entre o usuário e a unidade central de processamento (CPU).
Time shared - ver Tempo partilhado.
Track - ver pista.
Trilha - ver pista.

U

Unidade central de processamento (UCP ou CPU) - parte de um computador que o controla e realiza as operações aritméticas e lógicas.

V

Variável em ponto flutuante - variável representada por números decimais, cujo ponto (que separa a parte inteira da fracionária) pode variar de posição ao ser realizada uma operação.

Variável em ponto fixo - variável inteira ou decimal, cujo número de casas decimais está pré-estabelecido.

Verificação de paridade - verificação da precisão de um número, visando detecção de erro, baseada no conceito de paridade.

W

Write - instrução comum a diversas linguagens, que indica ao computador que ele deve fornecer uma saída em uma forma legível, como um texto em uma impressora ou linhas em uma tela de vídeo.

Impresso em offset

GRAFICA EDITORA HAMBURG

Avenida Bogaert, 64
Vila das Mercês São Paulo
Fone: 914-0233
CEP 04298

com filmes fornecidos pelo editor